

Reliable and Trustworthy Artificial Intelligence

Lecture 1 [Part II]: Adversarial Attacks and Defenses

Martin Vechev

ETH Zurich

Fall 2022

Today

- Examples of attacks
- Adversarial attacks
- Adversarial defenses

Adversarial Examples

Noisy attack: vision system thinks we now have a gibbon...

x
 “panda”
 57.7% confidence

$+ .007 \times$

$\text{sign}(\nabla_x J(\theta, x, y))$
 “nematode”
 8.2% confidence

$=$

$x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$
 “gibbon”
 99.3 % confidence

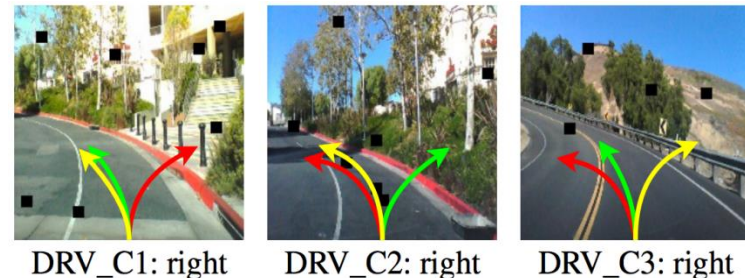
Explaining and Harnessing Adversarial Examples, ICLR '15

Tape pieces make network predict a 45mph sign



Robust Physical-World Attacks on Deep Learning Visual Classification, CVPR'18

Self-driving car: in each picture one of the 3 networks makes a mistake...

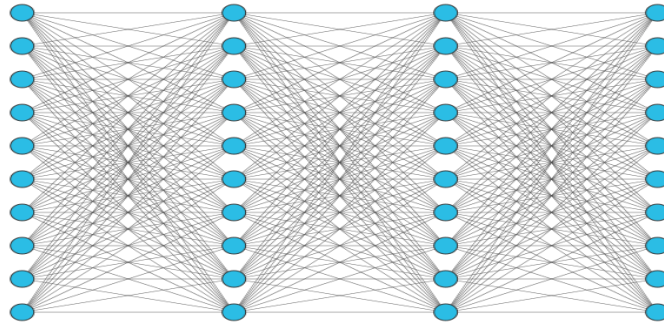


DeepXplore: Automated Whitebox Testing of Deep Learning Systems, SOSP'17

Adversarial Geometric Perturbations



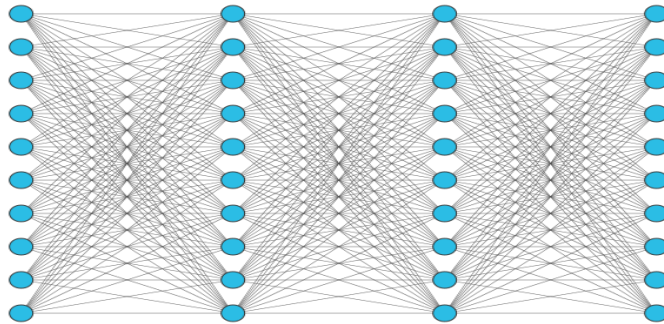
I_0



7



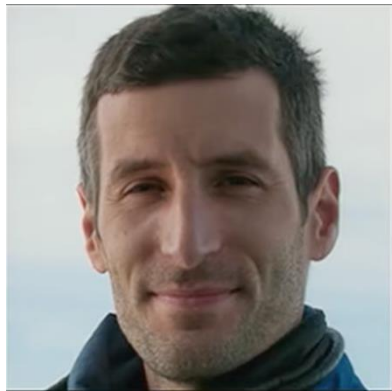
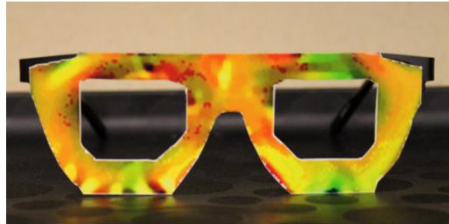
$I = rotate(I_0, -35)$



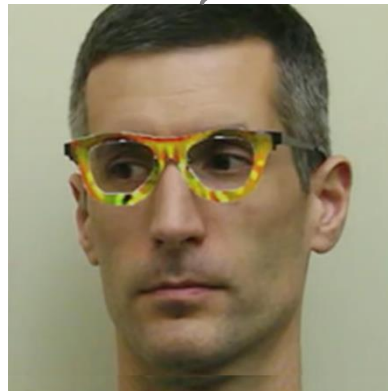
3

Real World Impersonation/Dodging Attacks

Real glasses



Lujo Bauer



=

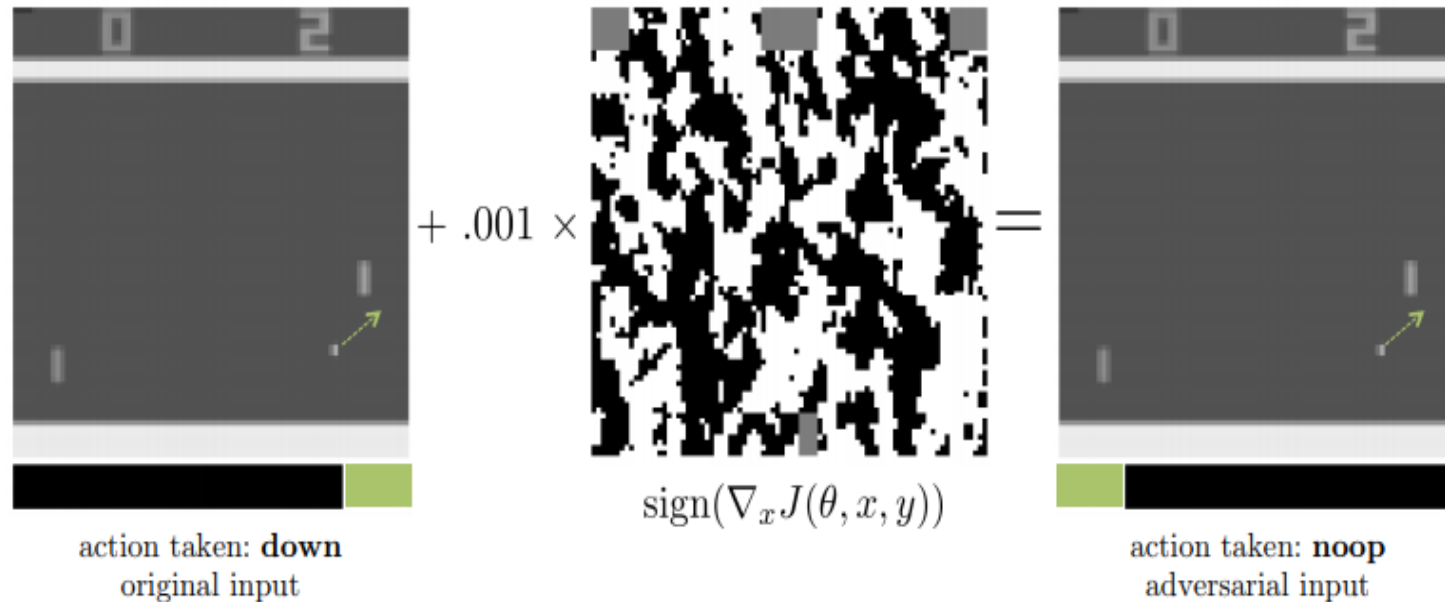


John Malkovich

100% success



Adversarial Examples in Reinforcement Learning



An agent (Deep Q Network) plays the game by selecting actions from a given state (image) that the game produces.

An attacker can perturb the image slightly so that the DQN agent chooses the wrong action: here, it wrongly picks noop (do nothing) in the right image, instead of moving the paddle down (left image).

Adversarial Examples in NLP

Article: Super Bowl 50

Paragraph: *“Peyton Manning became the first quarterback ever to lead two different teams to multiple Super Bowls. He is also the oldest quarterback ever to play in a Super Bowl at age 39. The past record was held by John Elway, who led the Broncos to victory in Super Bowl XXXIII at age 38 and is currently Denver’s Executive Vice President of Football Operations and General Manager. Quarterback Jeff Dean had jersey number 37 in Champ Bowl XXXIV.”*

Question: *“What is the name of the quarterback who was 38 in Super Bowl XXXIII?”*

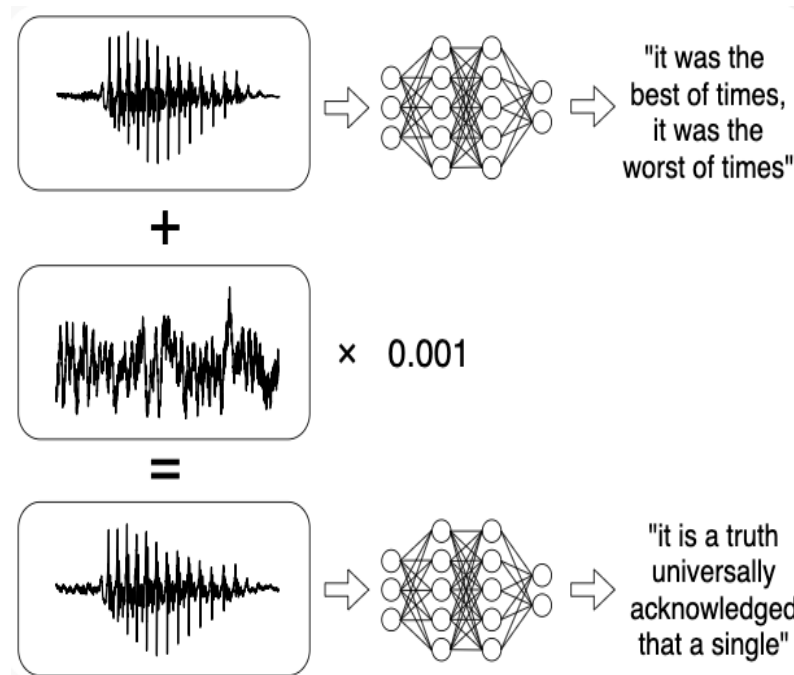
Original Prediction: John Elway

Prediction under adversary: Jeff Dean

The Ensemble model is fooled by the addition of an adversarial distracting sentence in blue.

Adversarial Examples in Audio Processing: Speech to Text

An attack on DeepSpeech:



Adding small noise to the input audio makes the network transcribe any arbitrary phrase

Adversarial Attack



57.7% panda



+ .007 ×



“noise”

=



93.3% gibbon

$$\text{sign}(\nabla_x \text{loss}_s(x))$$

Targeted vs. Untargeted Attacks

Targeted Attack – aims to misclassify the input (e.g., image) to a **specific label** (e.g. panda to gibbon)

Untargeted Attack – aims to misclassify the input to **any wrong label** (e.g. panda to any other animal)

Formulated as a slightly different optimization problem

Targeted Attack: Problem Statement

Input:

- neural network $f: X \rightarrow C$
- input $x \in X$
- target label $t \in C$, such that $f(x) \neq t$

Output:

- A perturbation η such that $f(x + \eta) = t$

Adversarial example

$$x' = x + \eta$$

Untargeted Attack: Problem Statement

Input:

- neural network $f: X \rightarrow C$
- input $x \in X$

Output:

- A perturbation η such that $f(x + \eta) \neq f(x)$

Adversarial example

$$x' = x + \eta$$

Types of Attacks

White box attacker: the attacker knows the model, the parameters, and the architecture

Black box attacker: the attacker knows the architecture (e.g., the layers) but not its parameters (e.g., weights)

Note: it was found adversarial examples are **transferrable**, hence given the same training data as the original network, an attacker can train their own **mirror network** of the black box original network and then attack the mirror network with white-box techniques. If attack on mirror network succeeds, it will likely succeed on the original.

We will look at **white box attacks**

Targeted Fast Gradient Sign Method

1. Compute perturbation:

$\eta = \epsilon \cdot \text{sign}(\nabla_x \text{loss}_t(x))$, where

$$\nabla_x \text{loss}_t = \left(\frac{\partial \text{loss}_t}{\partial x_1}, \dots, \frac{\partial \text{loss}_t}{\partial x_n} \right) \quad \text{sign}(g) = \begin{cases} -1, & \text{if } g < 0 \\ 0, & \text{if } g = 0 \\ 1, & \text{if } g > 0 \end{cases}$$

2. Perturb the input:

$$x' = x - \eta$$

3. Check if:

$$f(x') = t$$

- Here, each x_i is a pixel
- ϵ is a very small constant (e.g., 0.007)
- As FGSM is 1-step, x' is **guaranteed** to stay inside the box $[x - \epsilon, x + \epsilon]$, so no need to project.
- t is the target, **bad label**
- loss_t is the loss w.r.t target label
- FGSM was designed to be fast, not optimal
(may not compute minimal perturbation)

Untargeted version of FGSM

1. Compute perturbation:

$$\eta = \epsilon \cdot \text{sign}(\nabla_x \text{loss}_s(x))$$

2. Perturb the input:

$$x' = x + \eta$$

3. Check if:

$$f(x') \neq s$$

- With untargeted FGSM, we do not know what the target (bad) label is that we want.
- We just want **some label** different than the correct label s .
- So we try to “get away” from the correct label by maximizing the value of the loss

FGSM



Image x ,
label: panda



+ .007 ×



“noise”

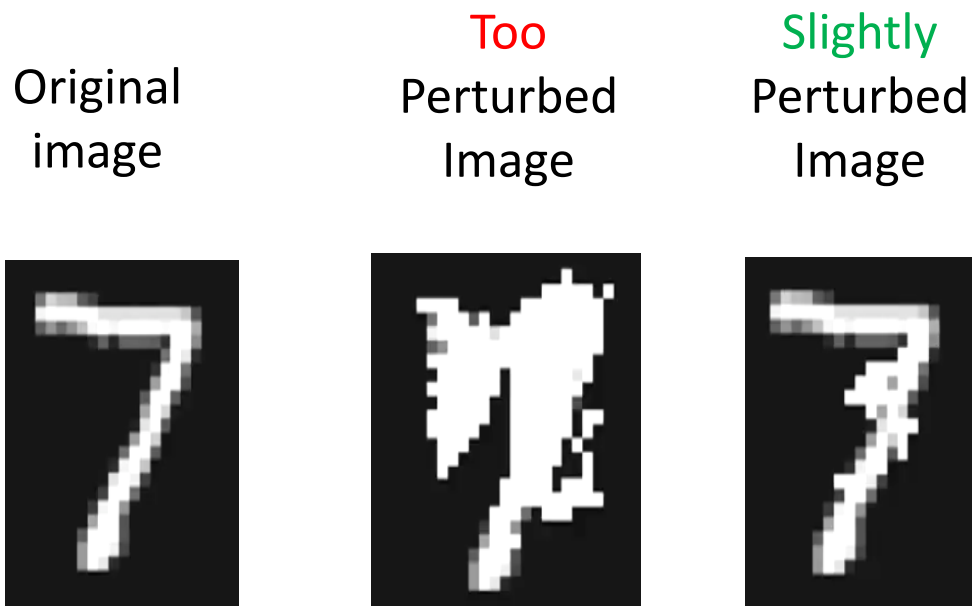
=



New image,
label: gibbon

$$\text{sign}(\nabla_x \text{loss}_{panda}(x))$$

Importance of Small Perturbations



We need some notion of distance....

Norm: Notion of Distance

Similarity of $\mathbf{x} \sim \mathbf{x}'$ is usually captured by an l_p norm:

$$\mathbf{x} \sim \mathbf{x}' \text{ iff } \|\mathbf{x} - \mathbf{x}'\|_p < \epsilon,$$

$$\text{where } \|\mathbf{x} - \mathbf{x}'\|_p = \left((|\mathbf{x}_1 - \mathbf{x}'_1|)^p + \dots + (|\mathbf{x}_n - \mathbf{x}'_n|)^p \right)^{\frac{1}{p}}$$

l_0 (when $0^0 = 0$ and we get rid of $1/p$ root) captures the number of changed pixels.

l_2 captures the **Euclidian distance** between x and x' . It can remain small if there are many small changes to many pixels.

l_∞ captures **maximum noise (change)** added to any coordinate. It is the maximum of the absolute values of the entries:

$$\|\mathbf{x} - \mathbf{x}'\|_\infty = \mathbf{max}(|\mathbf{x}_1 - \mathbf{x}'_1|, \dots, |\mathbf{x}_n - \mathbf{x}'_n|)$$

This is the most common norm used for adversarial example generation and it is argued that it most naturally captures human vision.

Targeted Attack with Small Changes

Input:

- neural network $f: X \rightarrow C$
- input $x \in X$
- target label $t \in C$, such that $f(x) \neq t$

Output:

- A perturbation η such that $f(x + \eta) = t$
- $\|\eta\|_p$ is minimized

Optimization Problem

The problem of generating small perturbations can be phrased as an **optimization problem**:

$$p \in \{0, 2, \infty\}$$

find η
minimize $\|\eta\|_p$
such that $f(x + \eta) = t$
 $x + \eta \in [0, 1]^n$

This is a **hard discrete constraint** which is difficult to optimize for with gradient methods.

Note: η can have negative components.

Key insight: Relaxation of the hard constraint

Optimization Problem

Two steps:

Step 1: Define an objective function obj_t such that:

$$\text{if } \mathit{obj}_t(\mathbf{x} + \boldsymbol{\eta}) \leq 0 \text{ then } f(\mathbf{x} + \boldsymbol{\eta}) = t$$

Step 2: Solve the following optimization problem:

$$\begin{array}{ll} \text{find} & \boldsymbol{\eta} \\ \text{minimize} & \|\boldsymbol{\eta}\|_p + c \cdot \mathit{obj}_t(\mathbf{x} + \boldsymbol{\eta}) \\ \text{such that} & \mathbf{x} + \boldsymbol{\eta} \in [0, 1]^n \end{array}$$

Optimization Problem

Two steps:

Step 1: Define an objective function obj_t such that:

$$\text{if } \mathit{obj}_t(\mathbf{x} + \boldsymbol{\eta}) \leq 0 \text{ then } \mathbf{f}(\mathbf{x} + \boldsymbol{\eta}) = t$$

What are examples of functions for obj with the property of Step 1?

Choice I:

$$\mathit{obj}_t(\mathbf{x}') = \text{loss}_t(\mathbf{x}') - 1$$

Lets take cross
entropy loss for loss_t

Choice II:

$$\mathit{obj}_t(\mathbf{x}') = \max(0, 0.5 - \mathbf{p}_f(\mathbf{x}')_t)$$

$\mathbf{p}_f(\mathbf{x}')_t$ returns the probability of
class t for input \mathbf{x}' on network \mathbf{f}

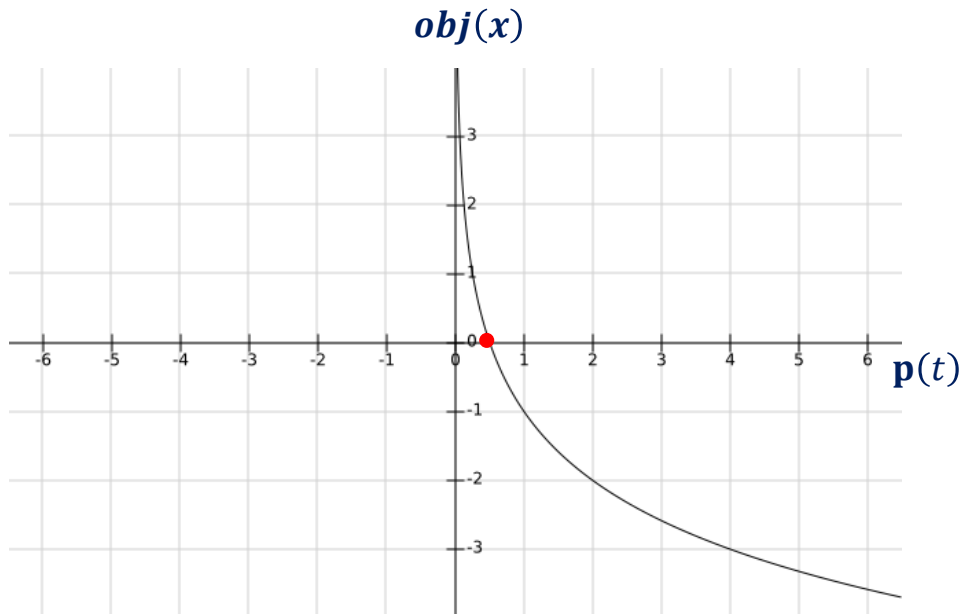
Choice I: $obj(x) = \text{loss}_t(x) - 1$

Choice I:

$$\begin{aligned} obj_t(x) &= \text{loss}_t(x) - 1 \\ &= -\log_2(p(t)) - 1 \end{aligned}$$

Plug in cross entropy loss for loss_t with logarithm base 2

Here, we use $p(t)$ as a shortcut for $p_f(x)_t$ so to avoid clutter



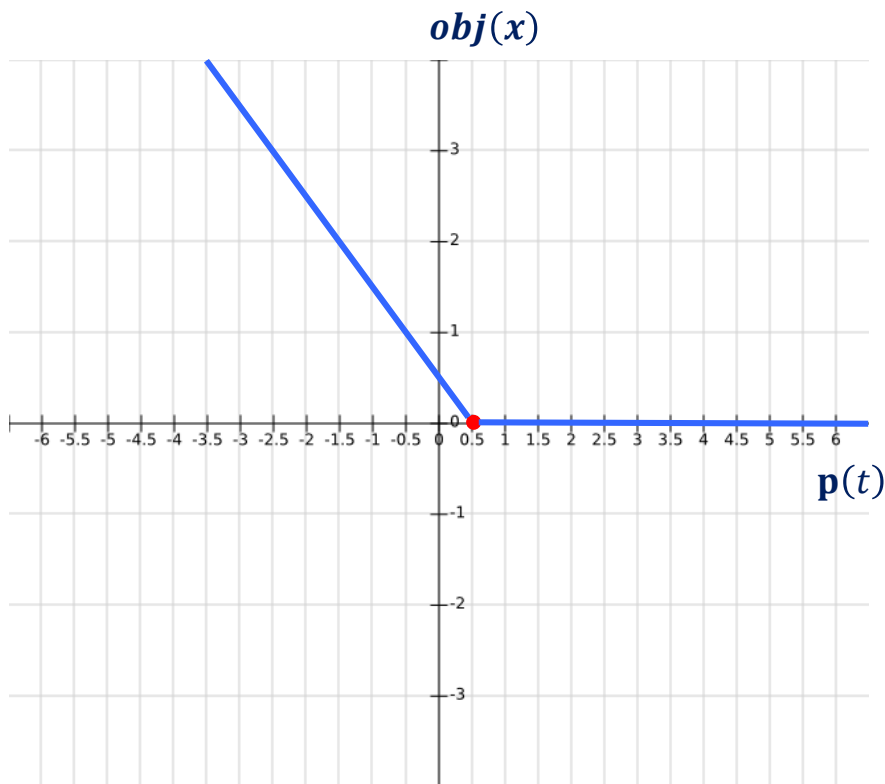
What we see here is that if the obj_t function is 0 or negative, then the probability $p(t)$ is ≥ 0.5 (50%).

But if $p(t)$ is ≥ 0.5 for the input x , then f will return t as a classification for x because this is the highest probability class. Hence, the desired property of Step 1 holds.

Choice II: $\max(0, 0.5 - \mathbf{p}_f(\mathbf{x})_t)$

Choice II:

$$\mathbf{obj}_t(\mathbf{x}) = \max(0, 0.5 - \mathbf{p}(t))$$



← Again we use $\mathbf{p}(t)$ as a shortcut for $\mathbf{p}_f(\mathbf{x})_t$ so to avoid clutter

What we see here is that the \mathbf{obj}_t function is always 0 or greater.

It is only 0 when $\mathbf{p}(t)$ is ≥ 0.5 for the input \mathbf{x} .

Again, then f will return t as a classification for \mathbf{x} because this is the highest probability class.

Hence, the desired property holds for Step 1.

Optimization Problem

Two steps:

Step 1: Define an objective function obj_t such that:

$$\text{if } \mathit{obj}_t(\mathbf{x} + \boldsymbol{\eta}) \leq 0 \text{ then } f(\mathbf{x} + \boldsymbol{\eta}) = t$$

Step 2: Solve the following optimization problem:

$$\begin{array}{ll} \text{find} & \boldsymbol{\eta} \\ \text{minimize} & \|\boldsymbol{\eta}\|_p + c \cdot \mathit{obj}_t(\mathbf{x} + \boldsymbol{\eta}) \\ \text{such that} & \mathbf{x} + \boldsymbol{\eta} \in [0, 1]^n \end{array}$$

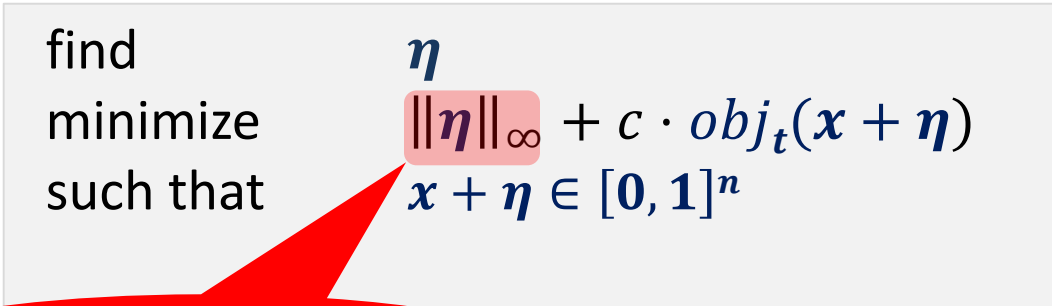
Optimization Problem

Two steps:

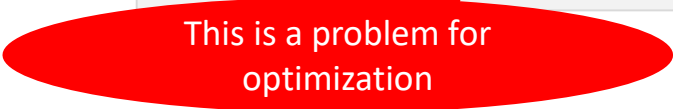
Step 1: Define an objective function obj_t such that:

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

Step 2: Solve the following optimization problem:



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



This is a problem for
optimization

Optimization Problem

Two steps:

Step 1: Define an objective function obj_t such that:

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

Step 2: Solve the following optimization problem:

$$\begin{array}{ll} \text{find} & \eta \\ \text{minimize} & \|\eta\|_\infty + c \cdot obj_t(x + \eta) \\ \text{such that} & x + \eta \in [0, 1]^n \end{array}$$

Hard Box Constraint

Dealing with Constraints

find $\boldsymbol{\eta}$
minimize $\|\boldsymbol{\eta}\|_p + c \cdot \text{obj}_t(\mathbf{x} + \boldsymbol{\eta})$
such that $\mathbf{x} + \boldsymbol{\eta} \in [0, 1]^n$

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

Projected gradient descent (PGD)

“Fit” all coordinates to be within the box

$$\text{project}((\eta_1, \dots, \eta_n)) = (\text{clip}_1(\eta_1), \dots, \text{clip}_n(\eta_n))$$

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

LBFGS-B optimizer:

Used by Carlini & Wagner

pass each $\eta_i \in [-x_i, 1 - x_i]$

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 $\eta_i \in [-e, e]$

With this approach we get

Target label

Initial label

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

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

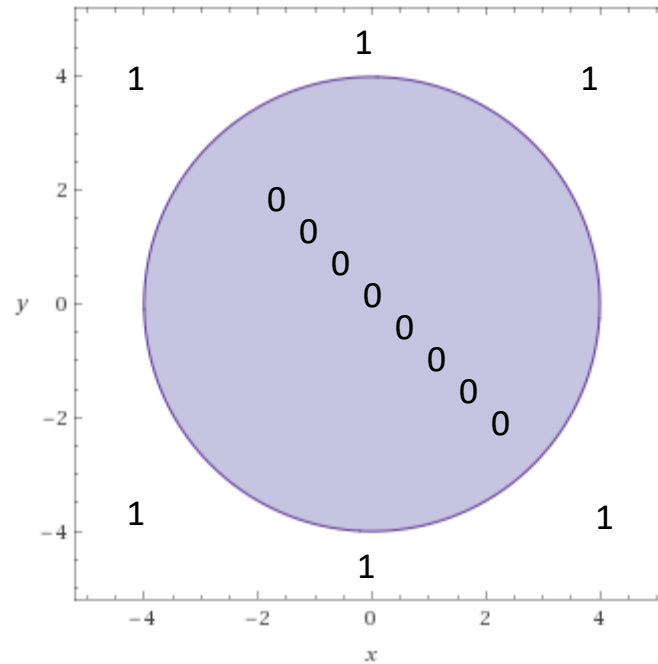
Given a **dataset** of points (x, y) where label is:

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

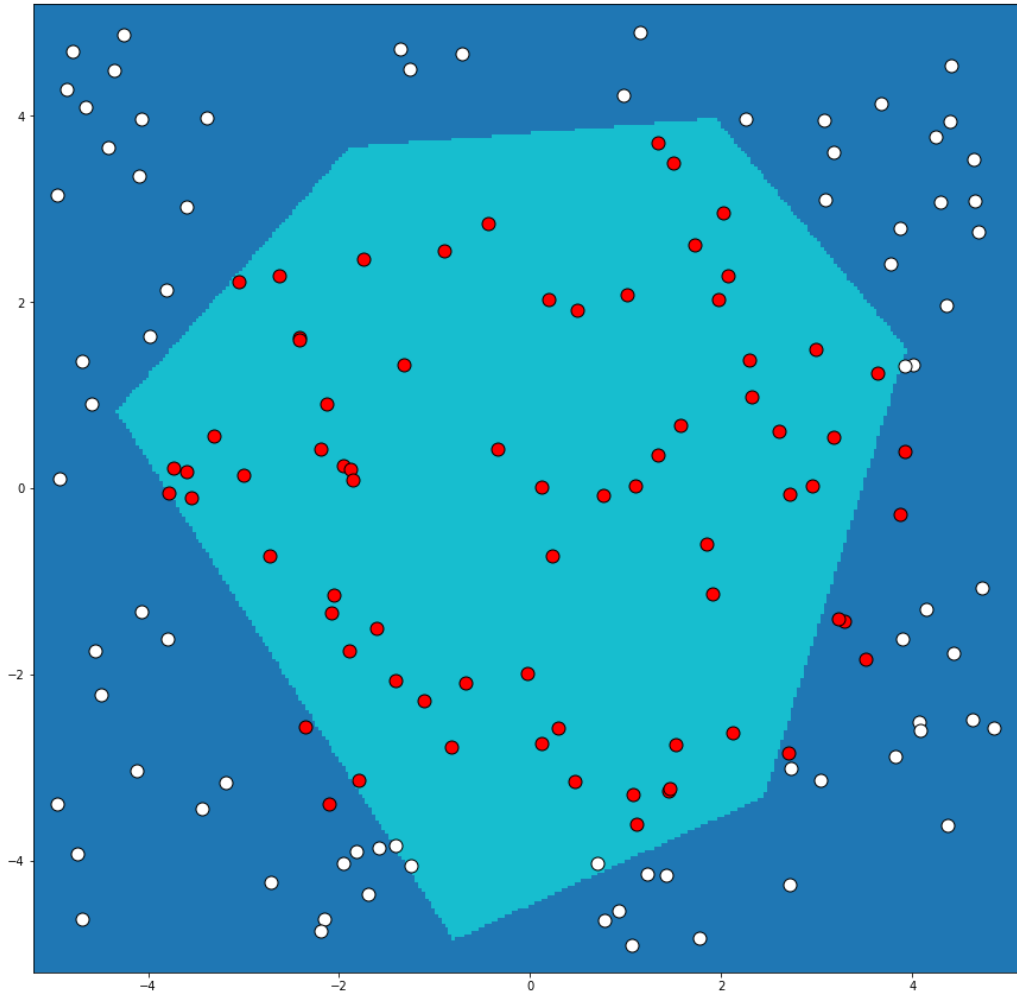
1 otherwise

train a neural network to classify the points correctly

Illustrating the PGD attack



After training we get the classifier:



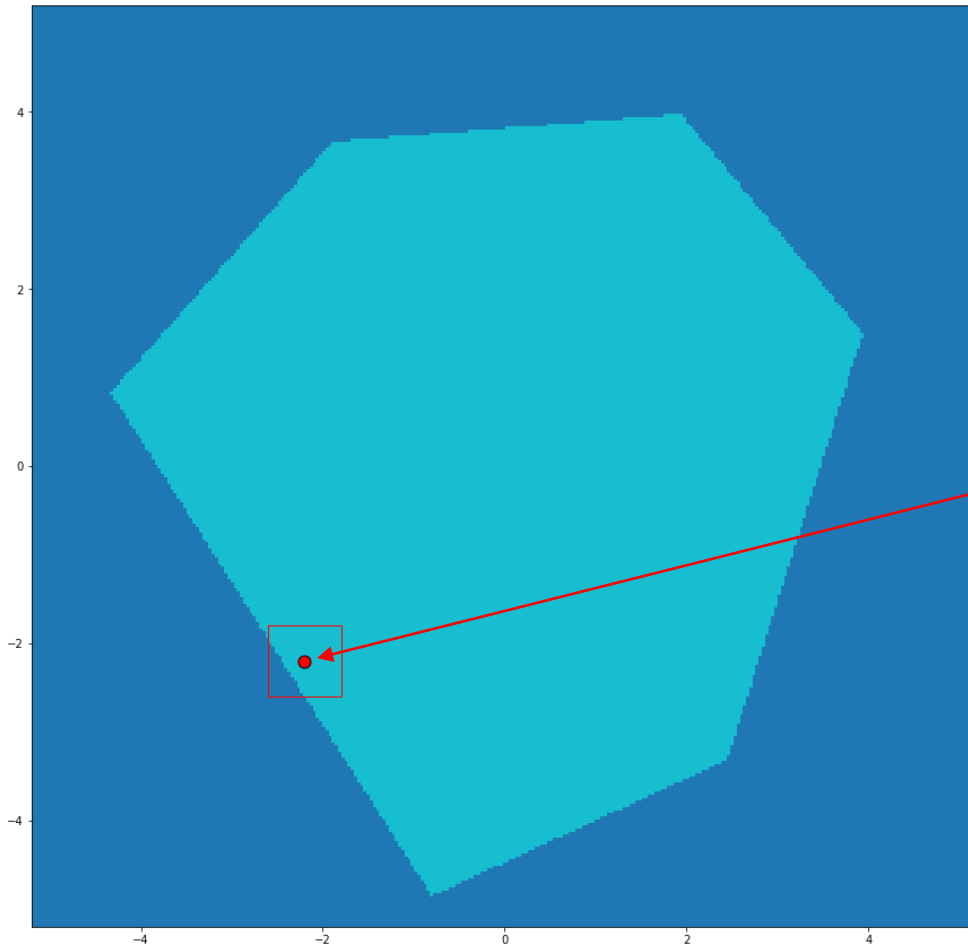
Dark blue – neural network predicts 1 (property does not hold)

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:

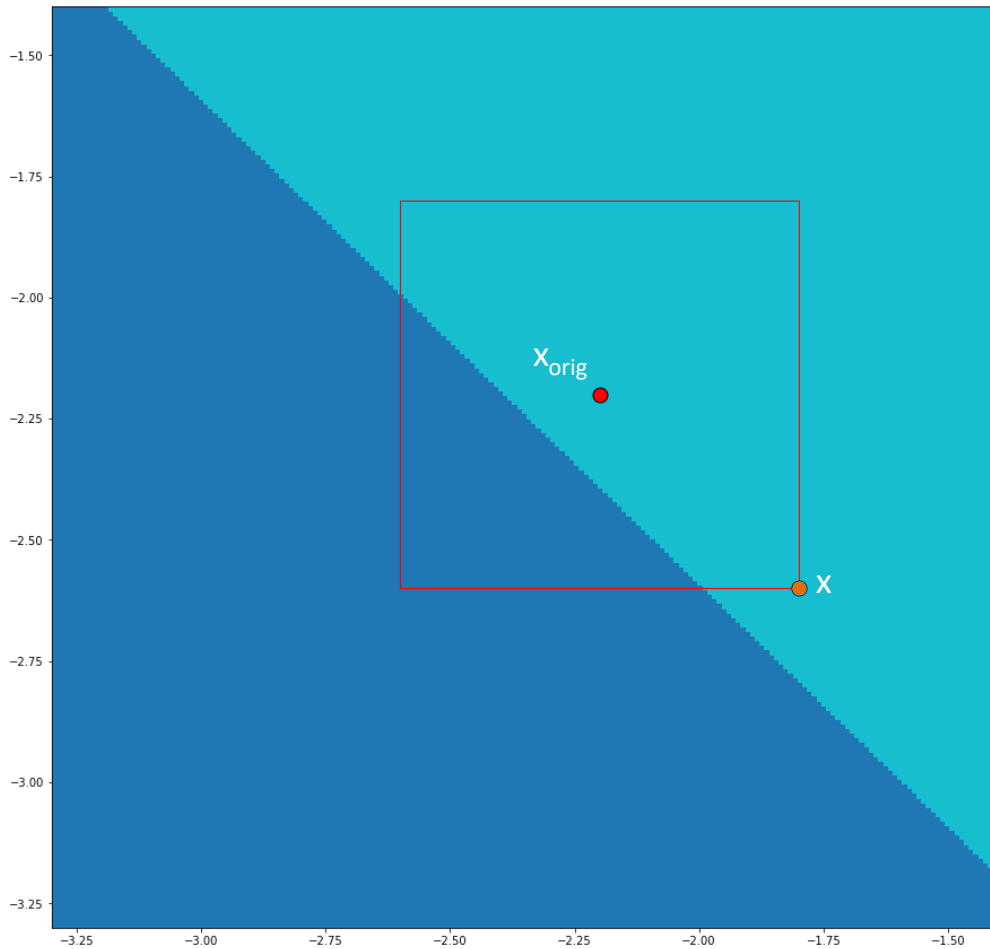
Find adversarial input in

L_{inf} ball around:

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

with $\epsilon=0.4$

Lets Zoom in a bit...

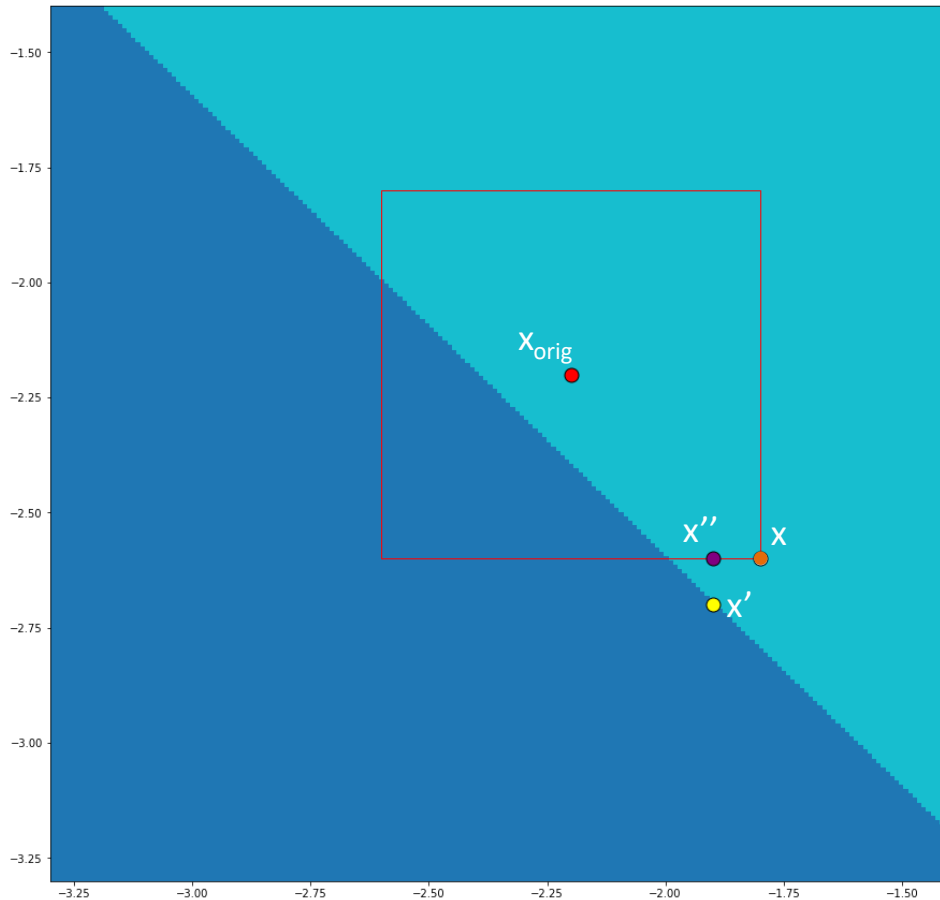


Initialize PGD with:

$$x = (-1.8, -2.6)$$

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

PGD Iteration 1



$$\text{NN}(x) = [0.5973, 0.4027]$$

$$\text{Loss}(x) = 0.5153$$

$$\nabla_x \text{Loss}(x) = [-0.852, -1.373]$$

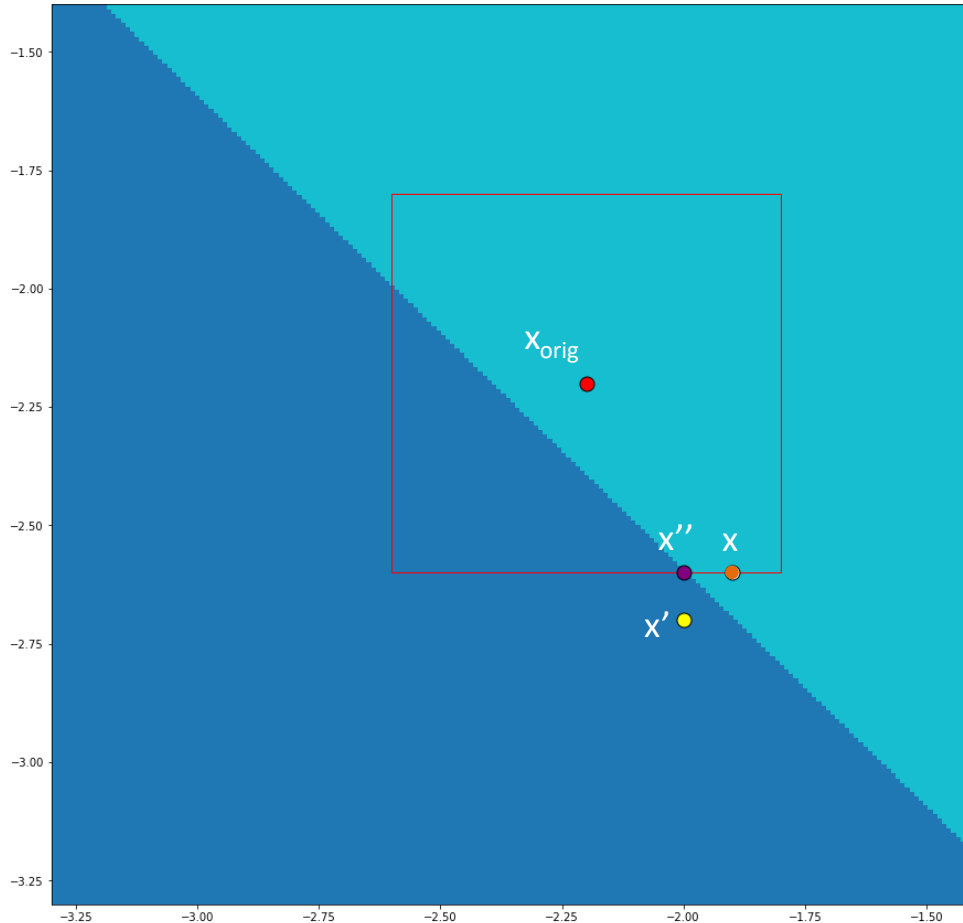
$$x' = x + \overbrace{0.1 * \text{sign}(\nabla_x \text{Loss}(x))}^{\text{Change } \Delta}$$
$$= [-1.9, -2.7] \text{ (yellow point)}$$

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**:

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

PGD Iteration 2



x'' from before now named x :

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

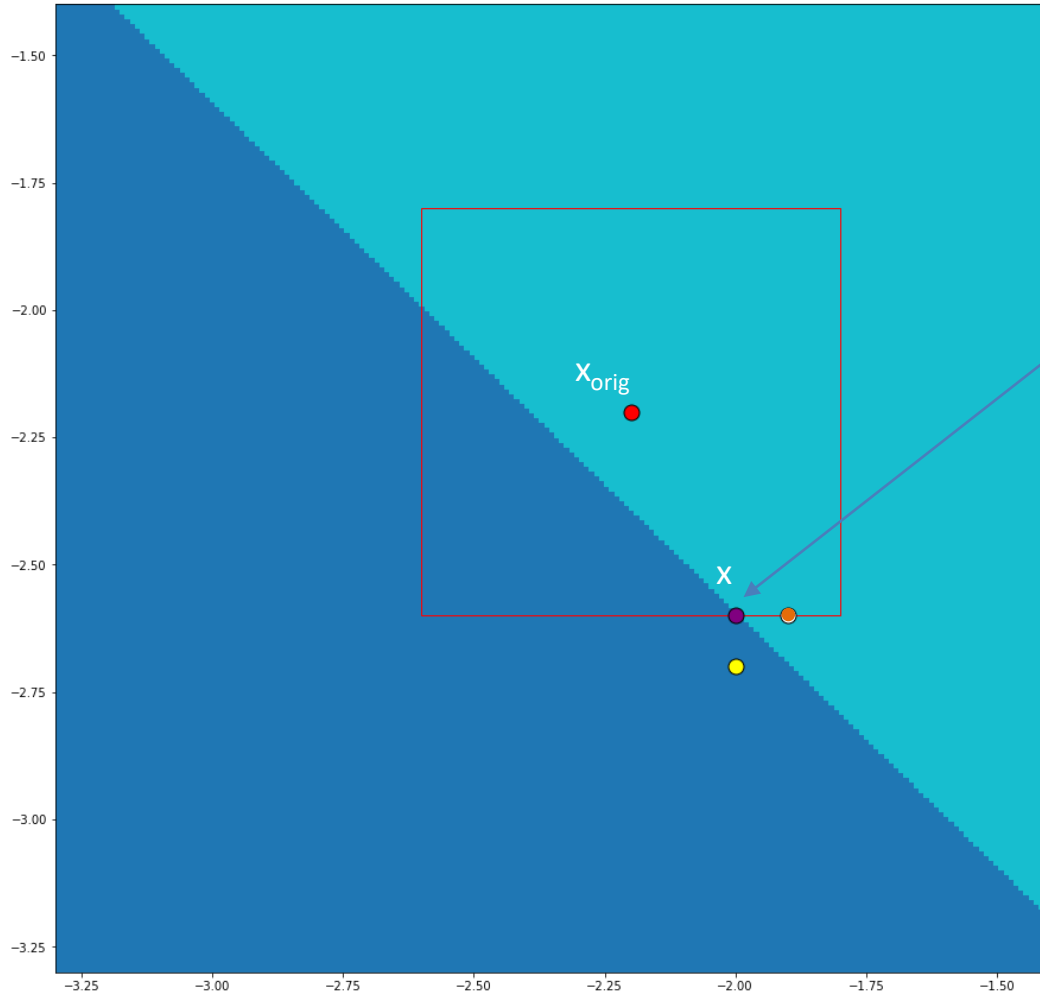
$\text{Loss}(x) = 0.6060$

$\nabla_x \text{Loss}(x) = [-0.9621, -1.5493]$

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

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

PGD Iteration 3

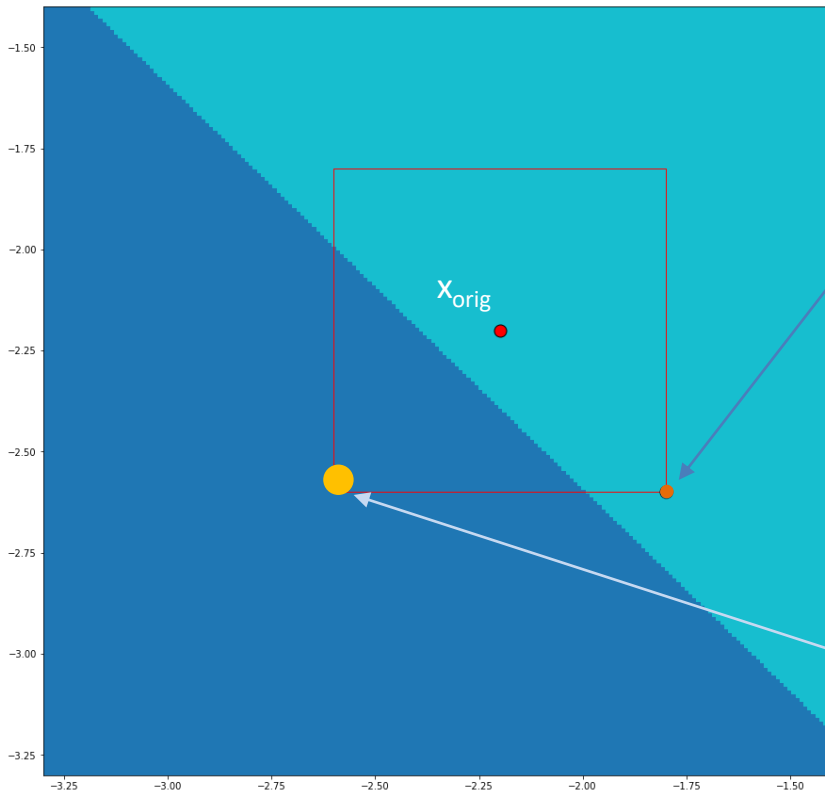


$$\text{NN}(x) = [0.4927, 0.5073]$$

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 $[-\epsilon, +\epsilon]$ as well as to the constraints needed so each element in the resulting point is between $[0,1]$
- Step size (here 0.1) is **typically smaller than ϵ** (used in FGSM)

- 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).

Summary of Adversarial Attacks

Attack Type	Region	Optimization	Outcome
FGSM (targeted, untargeted)	Change η fixed to $[-\epsilon, +\epsilon]$.	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 (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 η : η has to be bounded s.t. original image + η stays in $[0,1]$.	Aims to produce a change η with small l_∞ . Takes many steps, using LBFGS-B to ensure η 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

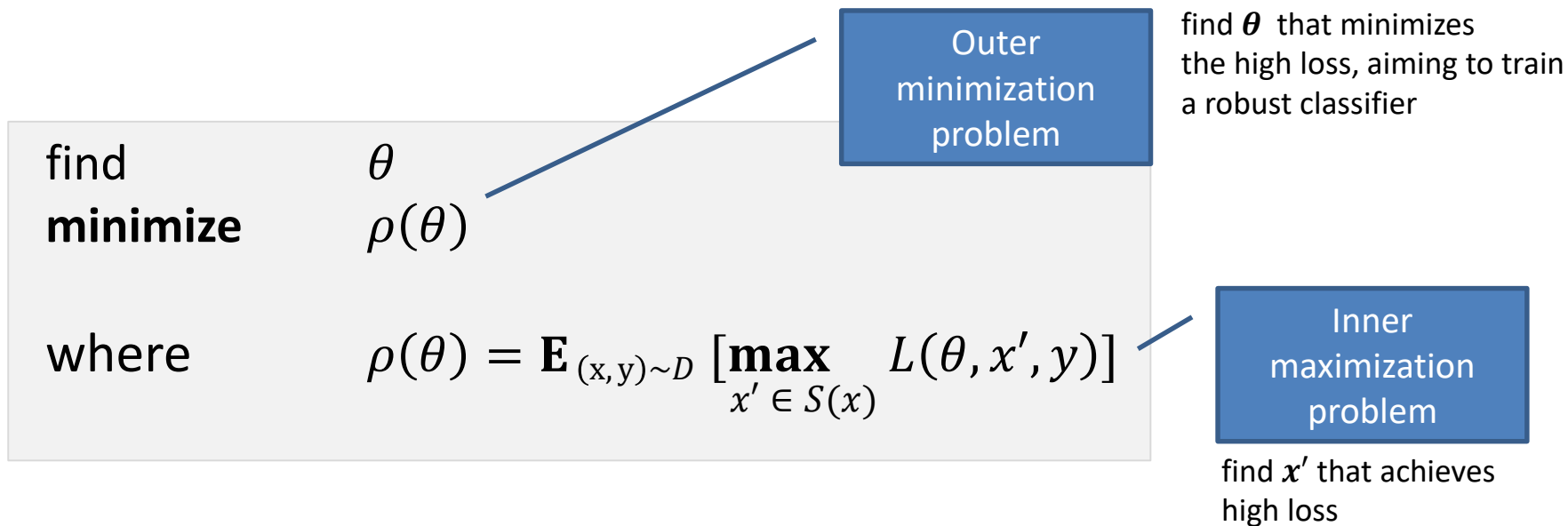
Adversarial Accuracy vs. Test Accuracy

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 $\epsilon=0.3$, 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 $\frac{80}{100} = 80\%$ and our **test accuracy** will be $\frac{95}{100} = 95\%$.

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**.

Defense as Optimization Problem



D is the underlying distribution

$\mathbf{E}_{(x,y) \sim D}$ 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:

$$S(x) = \{x' \mid \|x - x'\|_{\infty} < \epsilon\}$$

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:

$$\theta' = \theta - \frac{1}{|B_{max}|} \sum_{(x_{max}, y) \in B_{max}} \nabla_{\theta} L(\theta, x_{max}, y)$$

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

Points to Consider when Defending

Model capacity matters: larger networks are more defensible 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 : **If we want robust model, decrease in standard accuracy is inevitable!**

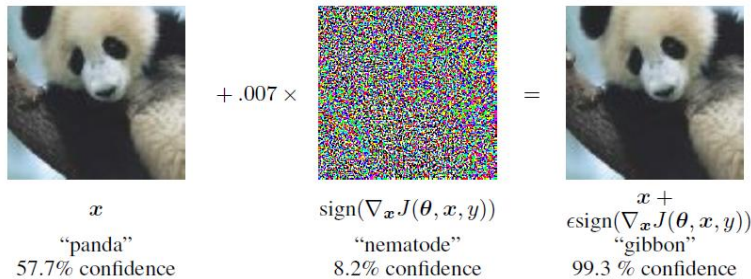
“Adversarially Robust Generalization Requires More Data “, Schmidt et. al. 2018 : **Provides lower bound on number of samples needed to achieve adversarial robustness**

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

Fast is better than free: Revisiting adversarial training, Wong et.al., 2020 : **FGSM with random initialization can work as well as PGD**

Lecture Summary

Deep Learning is susceptible to adversarial examples



Generating Adversarial examples (an optimization problem)

- FGSM: targeted and untargeted
- C&W (minimize perturbation)
- PGD

Defending against Adversarial examples (an optimization problem)

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

Applicability

Many of the techniques shown today are applicable to domains beyond images, but also models for natural language, code, audio processing, financial data and many more.