Trustworthy Aritificial Intelligence: Adversarial Attacks

Presented by

Dr. Narinder Singh Punn,

Assistant Professor,

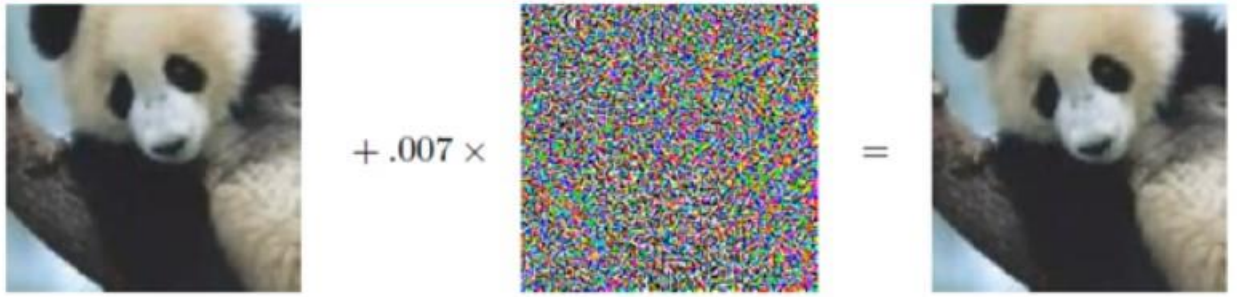
Dept. of CSE,

ABV-IIITM Gwalior

विश्वजीवनामृतं ज्ञानम्

Adversarial Attacks

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



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…

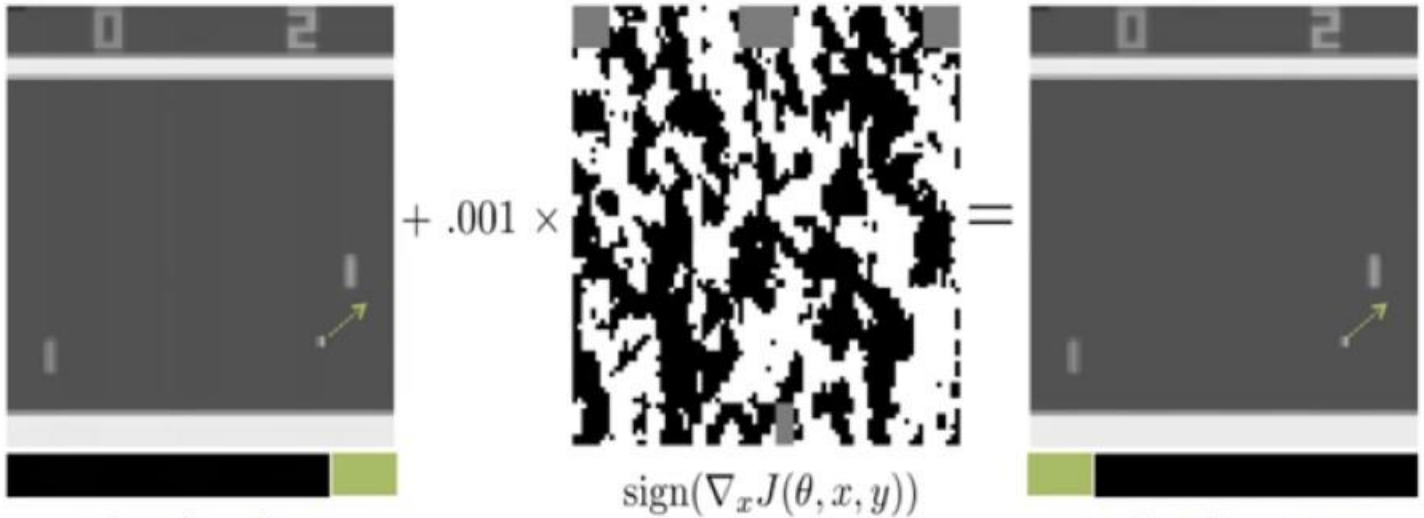
DeepXplore: Automated Whitebox Testing of Deep Learning Systems, SOSP'17

Adversarial Attacks

"Adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake"

(Goodfellow et al 2017)

Adversarial in Reinforcement Learning



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

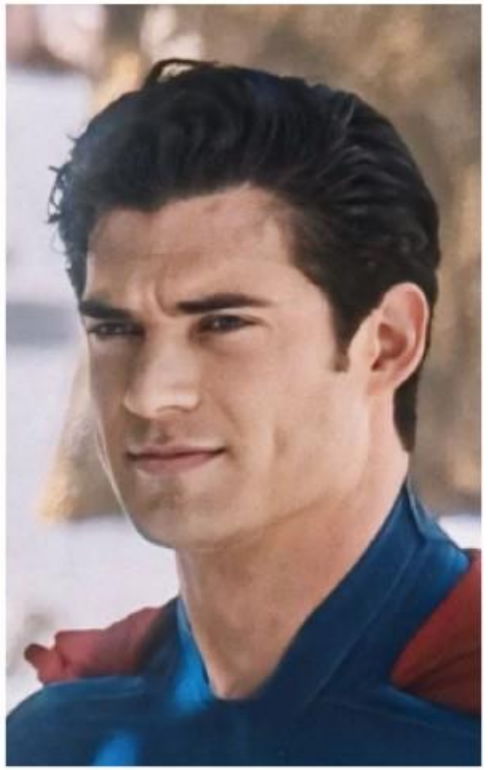
An attacker can perturb the image slightly so that the DQN agent chooses the wrong action: here, it wrongly picks noop (do nothing) in the right image, instead of moving the paddle down (left image).

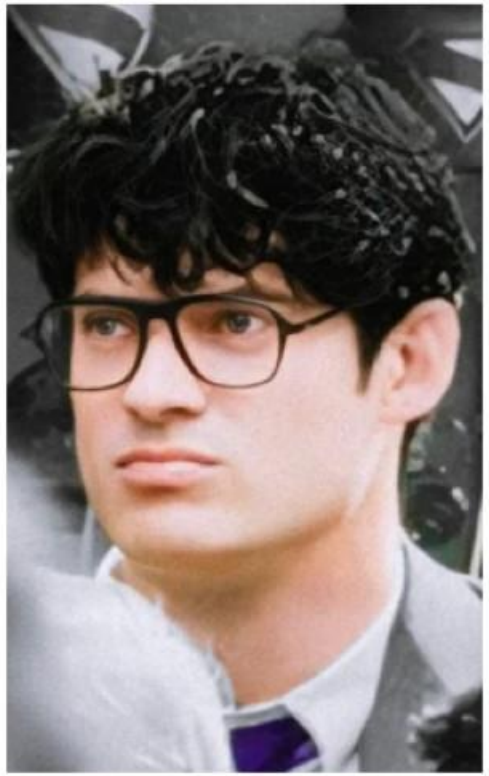
Adversarial Attacks on Neural Network Policies, Huang et.al, 2017

Adversarial Impersonation

Glasses

Superman

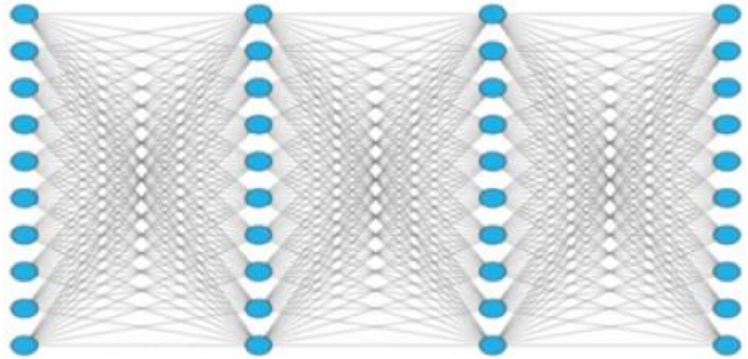




Clark Kent

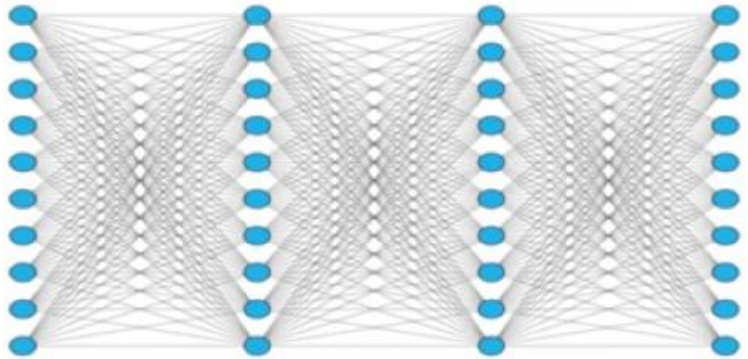
Adversarial Geometric Perturbations

I_o



7

$I = rotate(I_o, -35)$



3

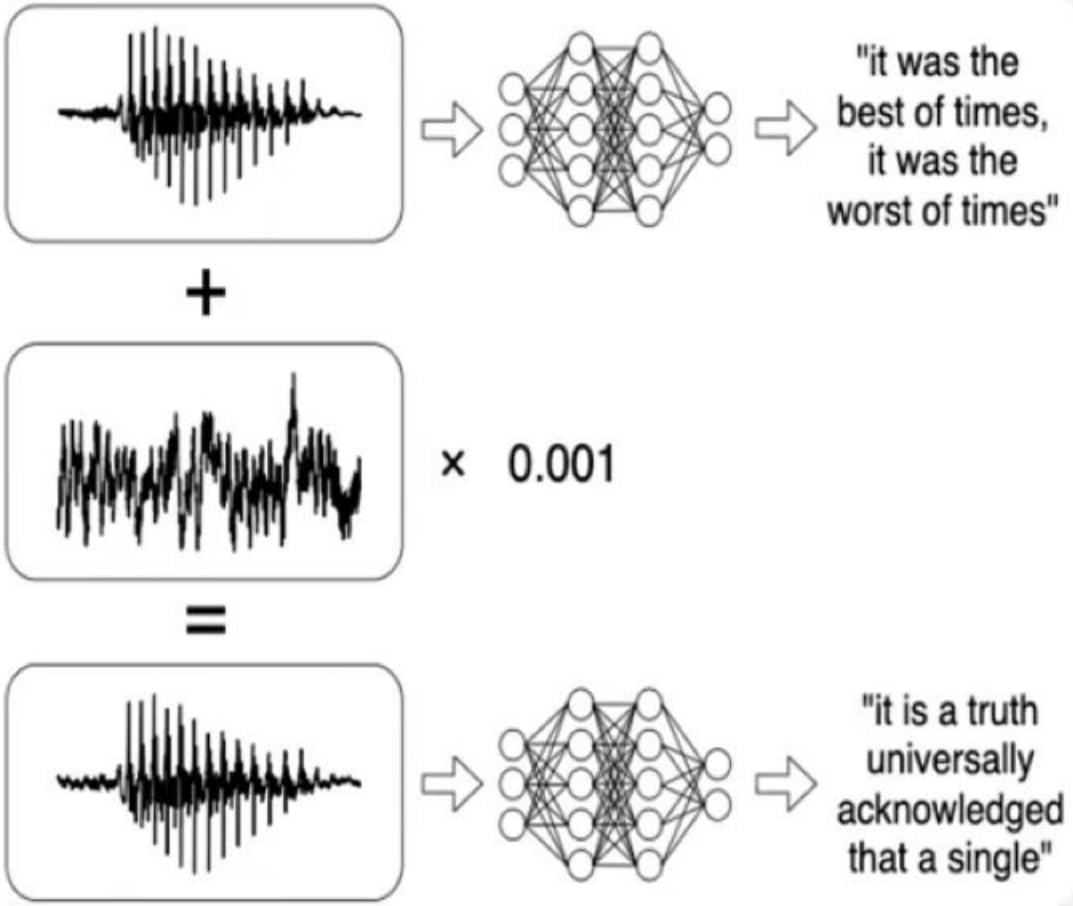
Adversarial Perturbations in NLP

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

Adversarial Examples for Evaluating Reading Comprehension Systems, EMNLP'17

Adversarial Perturbations in Speech

An attack on DeepSpeech:



Audio Adversarial Examples: Targeted Attacks on Speech-to-Text, ICML 2018 workshop

Robustness

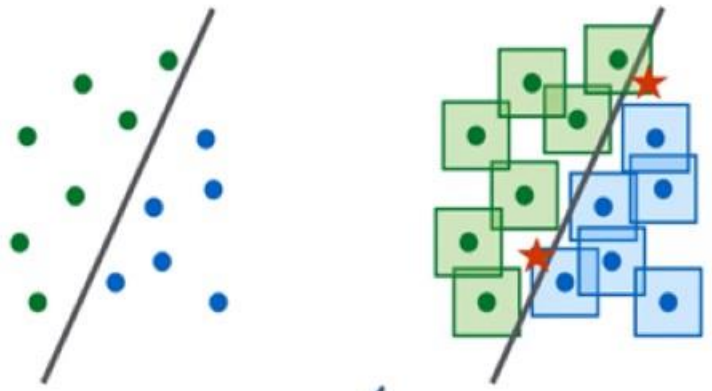
- Robustness: A network is robust if it produces correct outputs for all inputs.
- Impractical: The input space is too vast to be fully covered.
- Local Robustness: A learning model is locally robust if it generates correct outputs for inputs similar to those in the training set.
- This was thought to be shown by achieving high accuracy on the test set.

High Accuracy ?

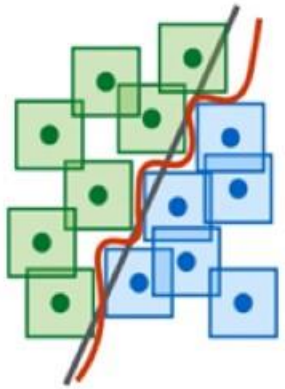
- Inputs for both the training and test sets are derived from a specific distribution.
- Neural networks are designed to attain high accuracy on test sets sourced from this distribution.
- Nevertheless, numerous similar inputs remain untested (and have low probability within the given distribution).

Story of Clever Hans

Why Do Adversarial Examples Exists?



Learned Model not powerful enough to fit data properly

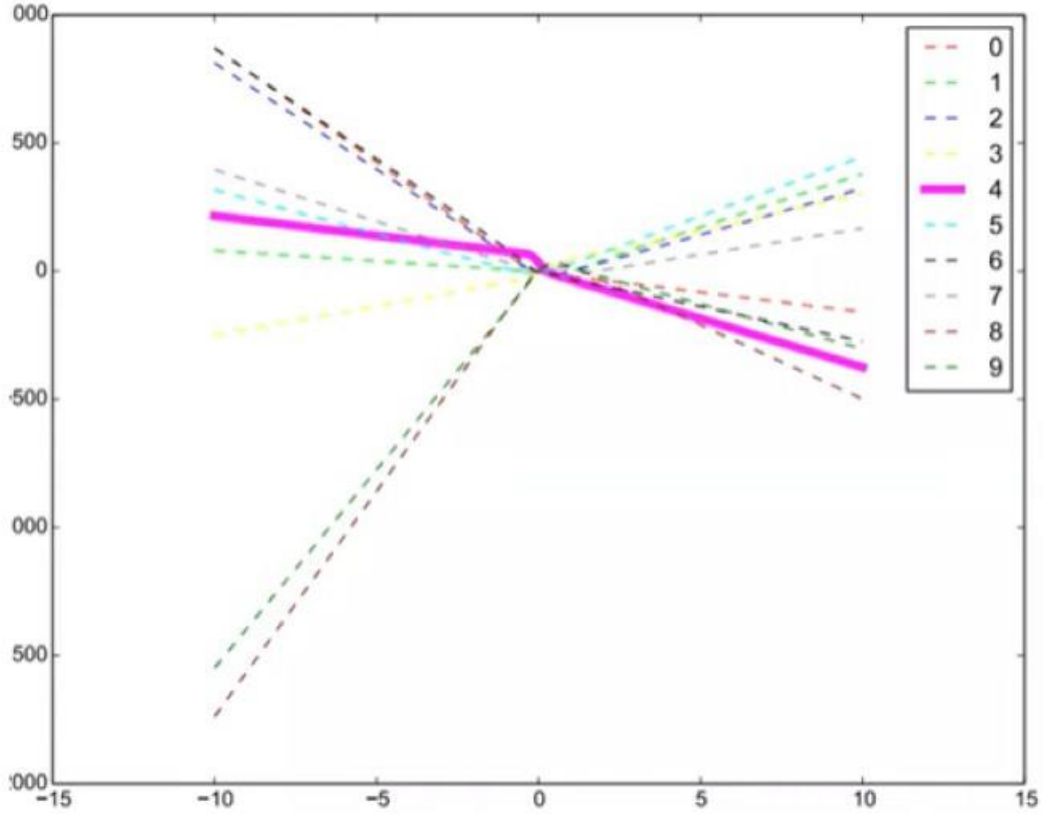


Learned Model trained on adversarial examples now more powerful (less linear)

(Madry 2017)

Experimental Linearity with Perturbations

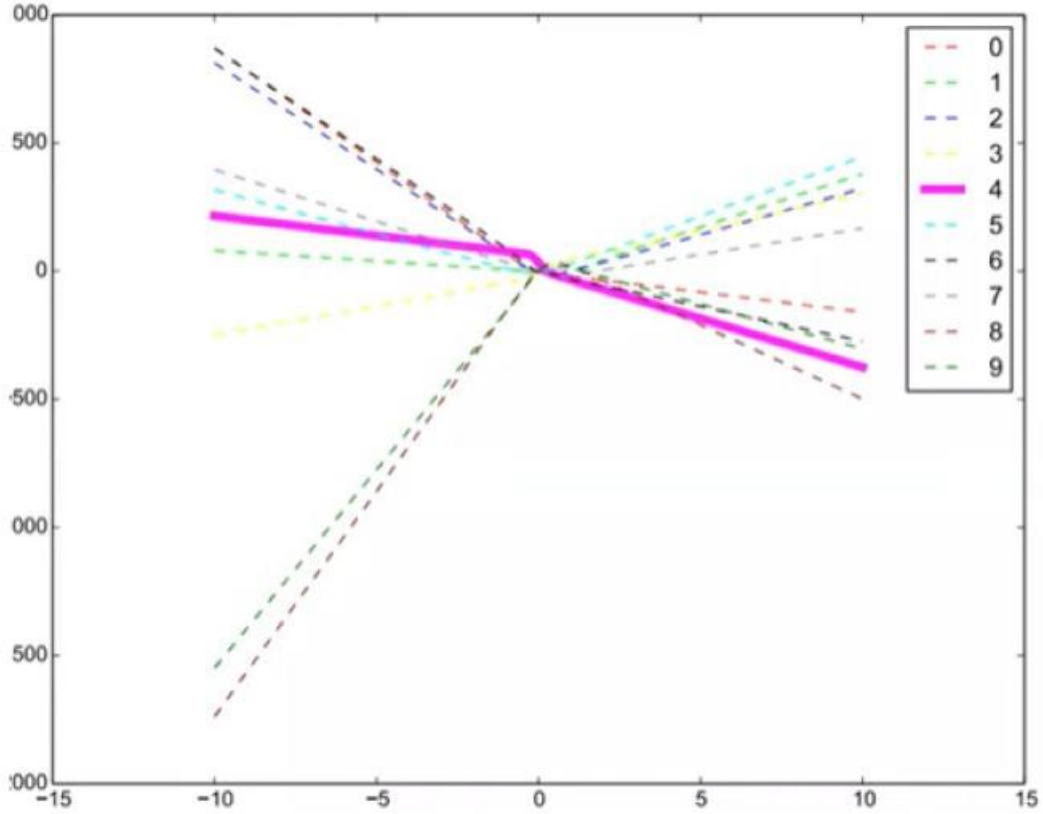
- Setup: A neural network is given a single image to classify. The correct classification for this image is a '4'.
- X-axis (ϵ): This represents the perturbation, which is a small, controlled change made to the image. A value of 0 on the axis is the original, unaltered image. As you move away from 0, the image is perturbed in a specific direction, essentially by adding a subtle amount of calculated noise.



Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).

Experimental Linearity with Perturbations

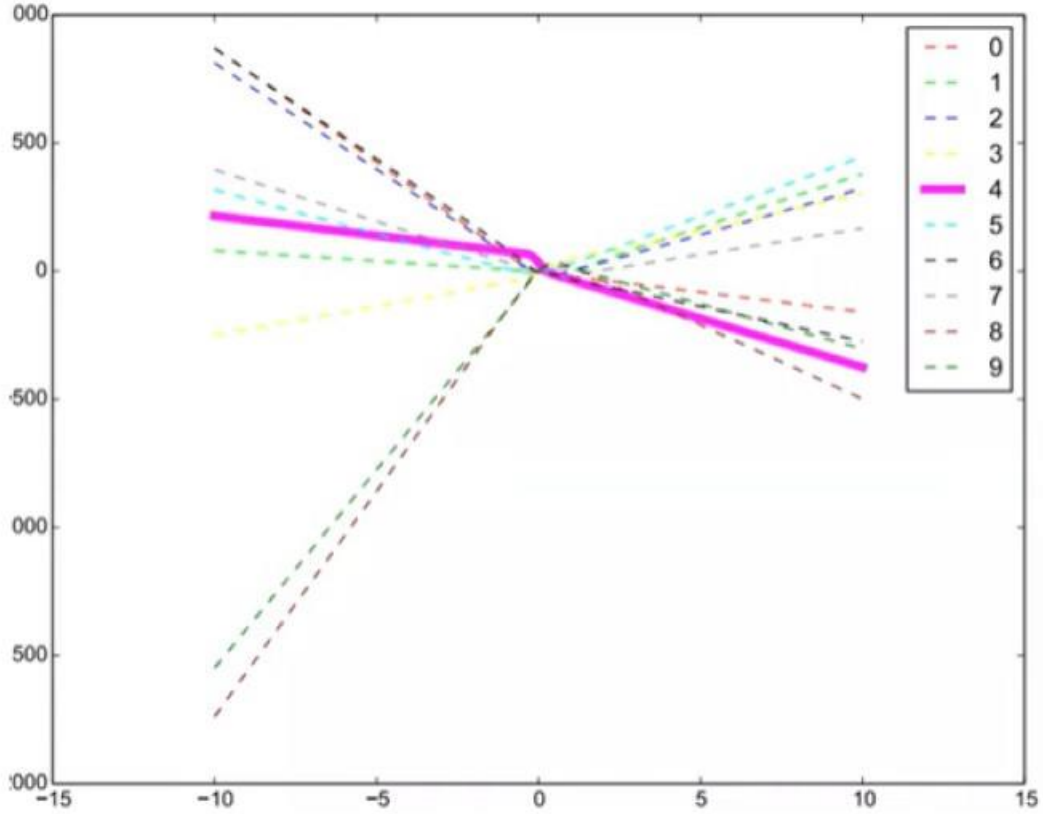
- Y-axis (Logits): These are the logits, which are the raw, unnormalized scores a neural network outputs for each class. A higher logit value for a class indicates a higher "confidence" from the network that the image belongs to that class.
- The graph shows that for all classes the logits change in a predictable. This means that for a small change to the input, the network's confidence scores for these classes also change in a consistent, straight-line manner.



Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).

Experimental Linearity with Perturbations

- The only exception is the logit for the correct class (the pink line, representing class 4). Its behavior is distinctly non-linear specifically around the point of no perturbation (ϵ=0). This non-linear "V" shape is where the network's function is correctly identifying the image. It represents the network's learned, complex decision boundary for separating the correct class from all others.



Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).

Attack Types

- **Based on the Attacker's Goal:**
- 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

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

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
$x' = x + \eta$

Attack Types

- **Based on the Attacker's Knowledge:**
- White-Box Attack In this scenario, the attacker has complete transparency into the target model. This includes knowledge of:
  - The model's architecture (e.g., the layers in a neural network).
  - The model's parameters and weights.
  - The training data used.
  - The gradients of the model's loss function.
- This full access allows for the creation of highly effective and efficient adversarial examples, often by using gradient-based methods to find the smallest possible perturbation to cause a misclassification.

Attack Types

- **Based on the Attacker's Knowledge:**
- Black-Box Attack This is a more realistic scenario where the attacker has little to **no internal knowledge** of the model.
- They can only interact with the model by providing inputs and observing the outputs (a query-based approach).
- Lacking internal details, attackers must use other techniques, such as:
  - Transferability: Creating an adversarial example on a known, local "substitute" model and hoping it also fools the target model.
  - Query-Based Methods: Repeatedly probing the target model with different inputs to infer its decision boundaries and craft an effective attack.

Targeted Fast Gradient Sign Method (FGSM)

Compute perturbation:

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

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

- Here, each x_i is a pixel
- t is the target, bad label
- ϵ is a very small constant (e.g., 0.007)

Targeted Fast Gradient Sign Method (FGSM)

Perturb the input:

$$x' = x - \eta$$

- As FGSM is 1-step, x' is guaranteed to stay inside the box $[x - \epsilon, x + \epsilon]$, so no need to project.

Check if:

$$f(x') = t$$

FGSM is designed to be fast but not optimal (may not compute minimal perturbation)

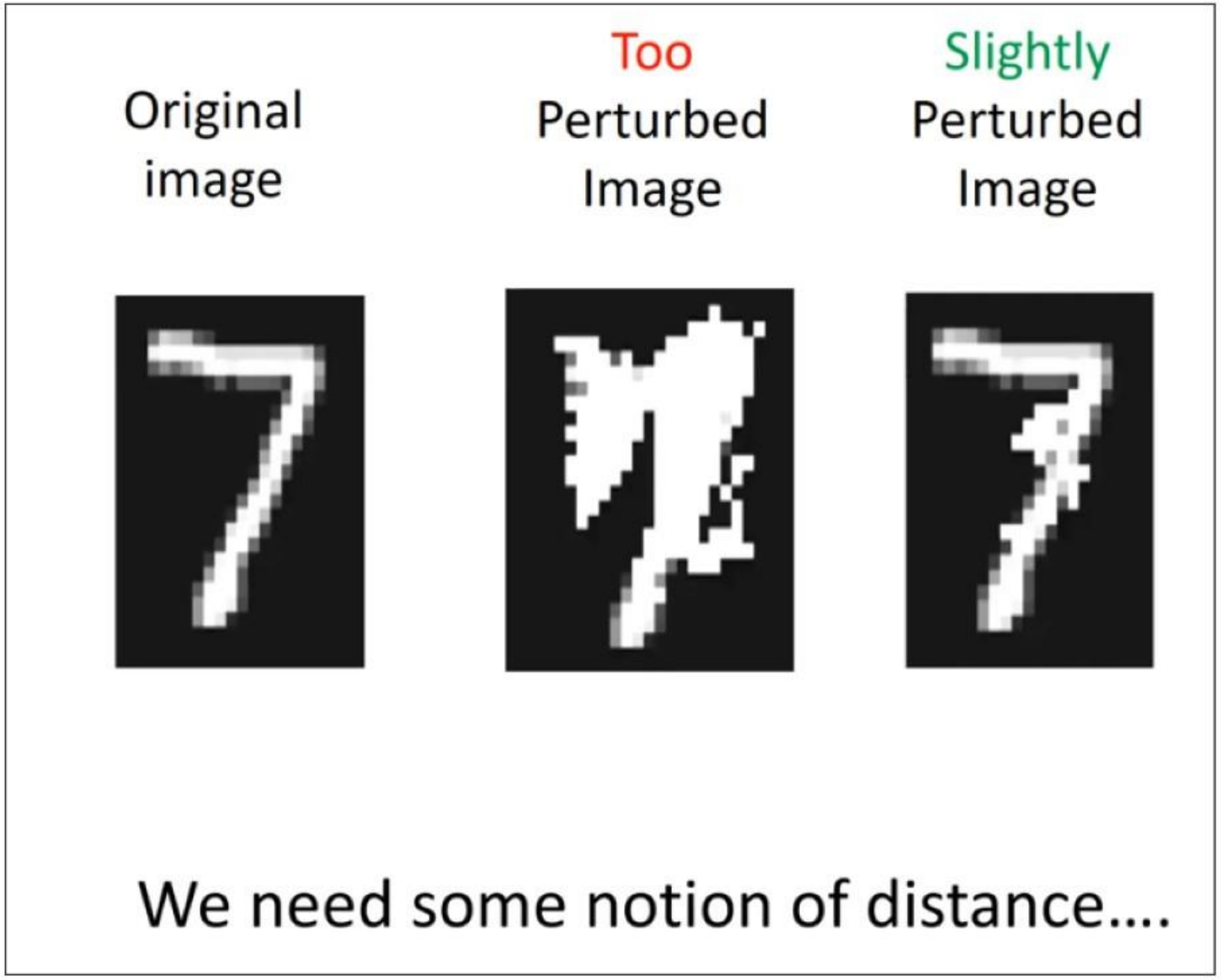
Untargeted Fast Gradient Sign Method (FGSM)

1. **Compute Perturbation:**
   - $\eta = \epsilon \cdot \text{sign}(\nabla_x \text{loss}_s(x))$
2. **Perturb the Input:**
   - $x' = x + \eta$
   - With untargeted FGSM, we do not know the target (bad) label we want.
   - We just want some label different from the correct label s.
3. **Check If:**
   - $f(x') \neq s$
   - So we try to "get away" from the correct label by maximizing the value of the loss.

Fast Gradient Sign Method (FGSM)

Explaining and Harnessing Adversarial Examples, ICLR'15

Fast Gradient Sign Method (FGSM)



Norm: Notion of Distance

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

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

$$\text{where } \|x - x'\|_p = \left((|x_1 - x'_1|)^p + \cdots + (|x_n - x'_n|)^p\right)^{\frac{1}{p}}$$

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:

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

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

To derive the max equation, see:
https://math.stackexchange.com/questions/3099179/proving-the-infinity-norm-is-equal-to-the-maximum-value-of-the-vector&sa=D&source=hangouts&ust=1600861806197000&usg=AFQjCNFLvo4hpAfiNBK06EwAZVFkuRiDNw

Targeted Attack (Min. Changes)

Input:

- neural network $f : X \to C$
- 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:

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

$$
\begin{aligned}
&\text{find} && \boldsymbol{\eta} \\
&\text{minimize} && \|\boldsymbol{\eta}\|_p \\
&\text{such that} && f(\boldsymbol{x} + \boldsymbol{\eta}) = t \\
& && \boldsymbol{x} + \boldsymbol{\eta} \in [0,1]^n
\end{aligned}
$$

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

Note: $\boldsymbol{\eta}$ can have negative components.

Key insight: Relaxation of the hard constraint

Carlini et al., Towards Evaluating the Robustness of Neural Networks, 2017

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

Optimization Problem

Two steps:

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

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

What are examples of functions for $\boldsymbol{obj}$ with the property of Step 1?

Choice I:

$$\boldsymbol{obj_t}(\boldsymbol{x'}) = \text{loss}_\text{t}(\boldsymbol{x'}) - 1$$

Lets take cross entropy loss for loss_t

Choice II:

$$\boldsymbol{obj_t}(\boldsymbol{x'}) = \max(0, 0.5 - \mathbf{p_f}(\boldsymbol{x'})_\text{t})$$

$\mathbf{p_f}(\boldsymbol{x'})_\text{t}$ returns the probability of class $\boldsymbol{t}$ for input $\boldsymbol{x'}$ on network $\mathbf{f}$

Optimization Problem

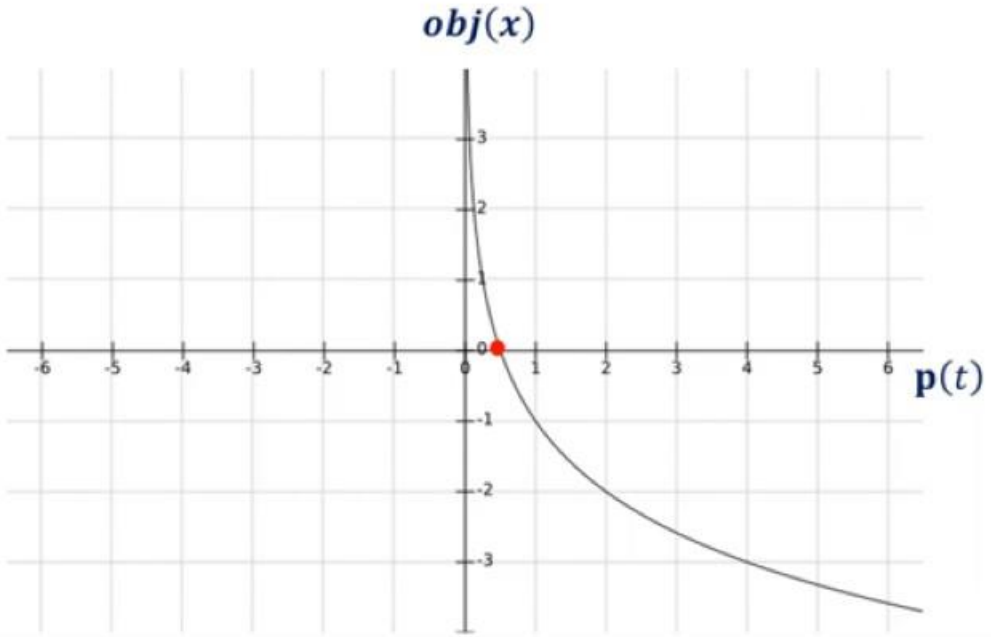
Choice I:

$$\boldsymbol{obj_t(x)} = \text{loss}_t(\boldsymbol{x}) - 1$$

$$= -\log_2(\mathbf{p}(t)) - 1$$

Plug in cross entropy loss for loss_t with logarithm base 2

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



What we see here is that if the $\boldsymbol{obj_t}$ function is 0 or negative, then the probability $\mathbf{p}(t)$ is $\geqslant 0.5$ (50%) .

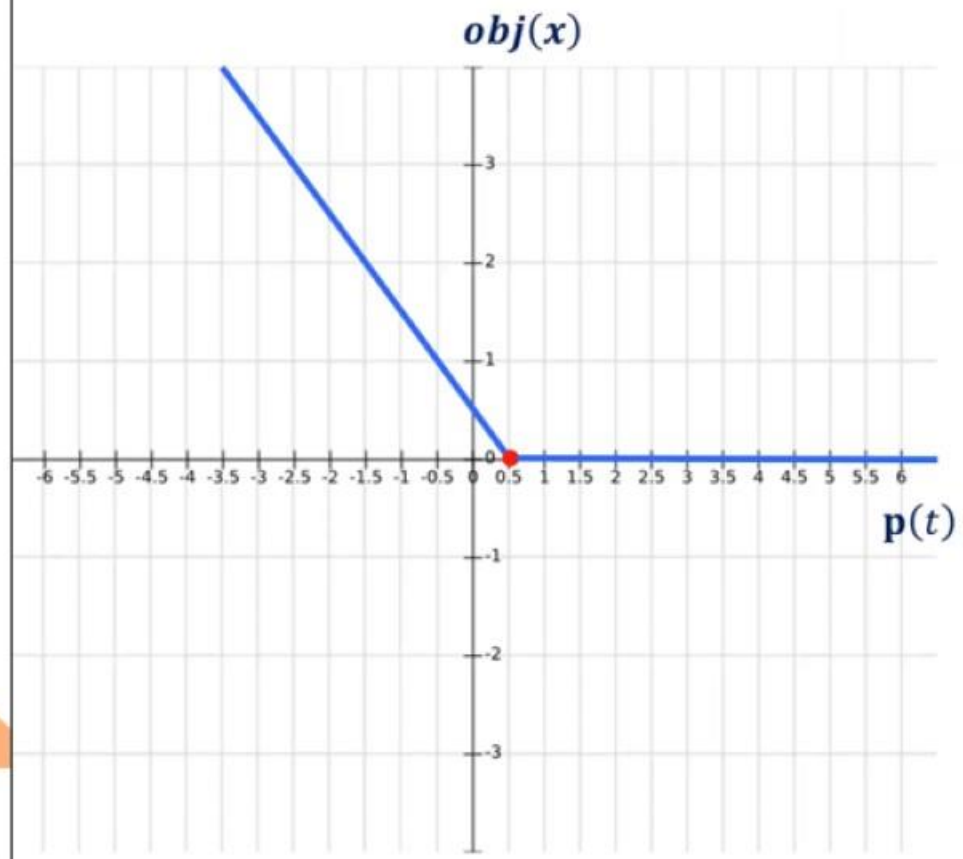
But if $\mathbf{p}(t)$ is $\geqslant 0.5$ for the input $\boldsymbol{x}$, then $\boldsymbol{f}$ will return t as a classification for $\boldsymbol{x}$ because this is the highest probability class. Hence, the desired property of Step 1 holds.

Optimization Problem

Choice II:

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

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



What we see here is that the obj_t function is always 0 or greater.

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

Again, then f will return t as a classification for 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 $\boldsymbol{obj_t}$ 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	$\boldsymbol{\eta}$
minimize	$\boldsymbol{\|\eta\|_p + c \cdot obj_t(x+\eta)}$
such that	$\boldsymbol{x+\eta \in [0,1]^n}$

Another 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:

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

This is a problem for optimization

Another Problem

$\|\boldsymbol{\eta}\|_\infty$ computes the maximum change: it takes the absolute value of every coordinate in $\boldsymbol{\eta}$ and returns the maximum value.

$\boldsymbol{\eta}$ = (0.5, 0.49, 0.48)

$$\frac{\partial \|\boldsymbol{\eta}\|_\infty}{\partial \boldsymbol{\eta}_1} = 1 \qquad \frac{\partial \|\boldsymbol{\eta}\|_\infty}{\partial \boldsymbol{\eta}_2} = 0 \qquad \frac{\partial \|\boldsymbol{\eta}\|_\infty}{\partial \boldsymbol{\eta}_3} = 0$$

After one step we get: $\boldsymbol{\eta} = \boldsymbol{\eta} - \gamma \cdot (1,0,0) = \boldsymbol{\eta} - 0.03 \cdot (1,0,0)$ = (0.47, 0.49, 0.48)

After two steps we get: $\boldsymbol{\eta} = \boldsymbol{\eta} - \gamma \cdot (0,1,0) = \boldsymbol{\eta} - 0.03 \cdot (0,1,0)$ = (0.47, 0.46, 0.48)

After three steps we get: $\boldsymbol{\eta} = \boldsymbol{\eta} - \gamma \cdot (0,0,1) = \boldsymbol{\eta} - 0.03 \cdot (0,0,1)$ = (0.47, 0.46, 0.45)

Another Problem

- The optimization process adjusts only the coordinate with the current maximum value at each step, as the gradient of $\|\eta\|_\infty$ is non-zero only for that coordinate.
- Because the subgradient of $\|\eta\|_\infty$ is zero for all non-maximum coordinates (assuming a unique maximum), the optimizer does not penalize small increases in those coordinates driven by the $obj_t(x+\eta)$ term, which aims to steer the perturbed input toward the target class t.
- This allows the obj_t component of the objective function to influence non-maximum coordinates without immediately affecting the $\|\eta\|_\infty$ value, provided those increases do not exceed the current maximum.

Another Problem

Going back to the full optimization problem which also includes $obj(\boldsymbol{x} + \boldsymbol{\eta})$

After one step we get: $\boldsymbol{\eta} = \boldsymbol{\eta} - \gamma \cdot (1,0,0) = \boldsymbol{\eta} - 0.03 \cdot (1,0,0)$ = (0.47, 0.49, 0.48)

Now the optimizer can slightly bump up the second location:

After one full step of optimizer, it may also bump up the 2nd location:	(0.47, 0.5, 0.48)
After second full step of optimizer, we may get:	(0.5, 0.47, 0.48)

Well, we are just oscillating now and bouncing around…turns out SGD may not be a good way to optimize $\|\boldsymbol{\eta}\|_\infty$ especially with other terms

Solution

Replace $\|\boldsymbol{\eta}\|_\infty$ with other proxy functions that reflect the distance

One idea is to penalize large values in $\boldsymbol{\eta}$ via a term τ:

$$\text{Replace } \|\boldsymbol{\eta}\|_\infty \text{ with } \sum_i \max\left(0, (|\boldsymbol{\eta}_i| - \tau)\right)$$

- τ is basically intended to capture the $\|\boldsymbol{\eta}\|_\infty$ bound when optimization finishes.
- τ will be continuosly minimized. Initially, τ starts at 1
- τ is decreased with some factor (say 0.9) at every iteration if all $|\boldsymbol{\eta}_i|$ are less than τ (then, entire expression will be 0).

Note: an iteration consist of K small steps.

Note: when τ is large, gradient of $\sum_i \max\left(0, (|\boldsymbol{\eta}_i| - \tau)\right)$ is similar to gradient of $\|\boldsymbol{\eta}\|_\infty$.

Solution

Let $L(\eta) = \sum_i \max(0, (|\eta_i| - \tau))$ and $\eta = (0.47, 0.49, 0.48)$

Start with $\tau = 1$, then $L(\eta) = 0$

Next iteration: $\tau = 0.9$, then $L(\eta) = 0$

Next iteration: $\tau = 0.81$, then $L(\eta) = 0$

...

At some iteration: $\tau = 0.478$, now via one step, we get:

$$L(\eta) = \sum_i \max(0, (|\eta_i| - 0.478)) = \sum_i (0, 0.012, 0.002) = \mathbf{0.014}$$

$$\nabla_\eta L(\eta) = (\mathbf{0}, \mathbf{1}, \mathbf{1})$$

We can then update η as usual, complete this step, and continue with the next step

Here we only show the optimization of one term, namely $L(\eta)$, to illustrate the relationship with optimizing $\|\eta\|_\infty$

Solution

Notes on optimization:

- There are K steps within an iteration, each updating $\boldsymbol{\eta}$.
- Entire optimization stops if after K steps $\mathrm{L}(\boldsymbol{\eta}) \neq 0$. Otherwise, if $\mathrm{L}(\boldsymbol{\eta}) = 0$, optimization continues with a new τ = 0.9 * previous τ.
- Entire optimization stops if it also reaches some pre-defined value of τ (1/256 for Carlini & Wagner).
- When optimization stops, we return $\boldsymbol{\eta}$ at the iteration before the last one. This means $\|\boldsymbol{\eta}\|_\infty \leq \tau$ where τ is the one used at iteration before last.
- If $\boldsymbol{\eta}_{top2} < \tau < \boldsymbol{\eta}_{top1}$ where $\boldsymbol{\eta}_{top1}$ is the largest element and $\boldsymbol{\eta}_{top2}$ is second largest element in $\boldsymbol{\eta}$, then the gradient $\nabla_{\boldsymbol{\eta}} \mathrm{L}(\boldsymbol{\eta})$ will be the same as $\nabla_{\boldsymbol{\eta}} \|\boldsymbol{\eta}\|_\infty$

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:

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

Hard Box Constraint

Final Constraint

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

$$\boldsymbol{project}((\eta_1, \dots, \eta_n)) = \left(clip_1(\eta_1), \dots, clip_n(\eta_n)\right)$$

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

LBFGS-B optimizer:

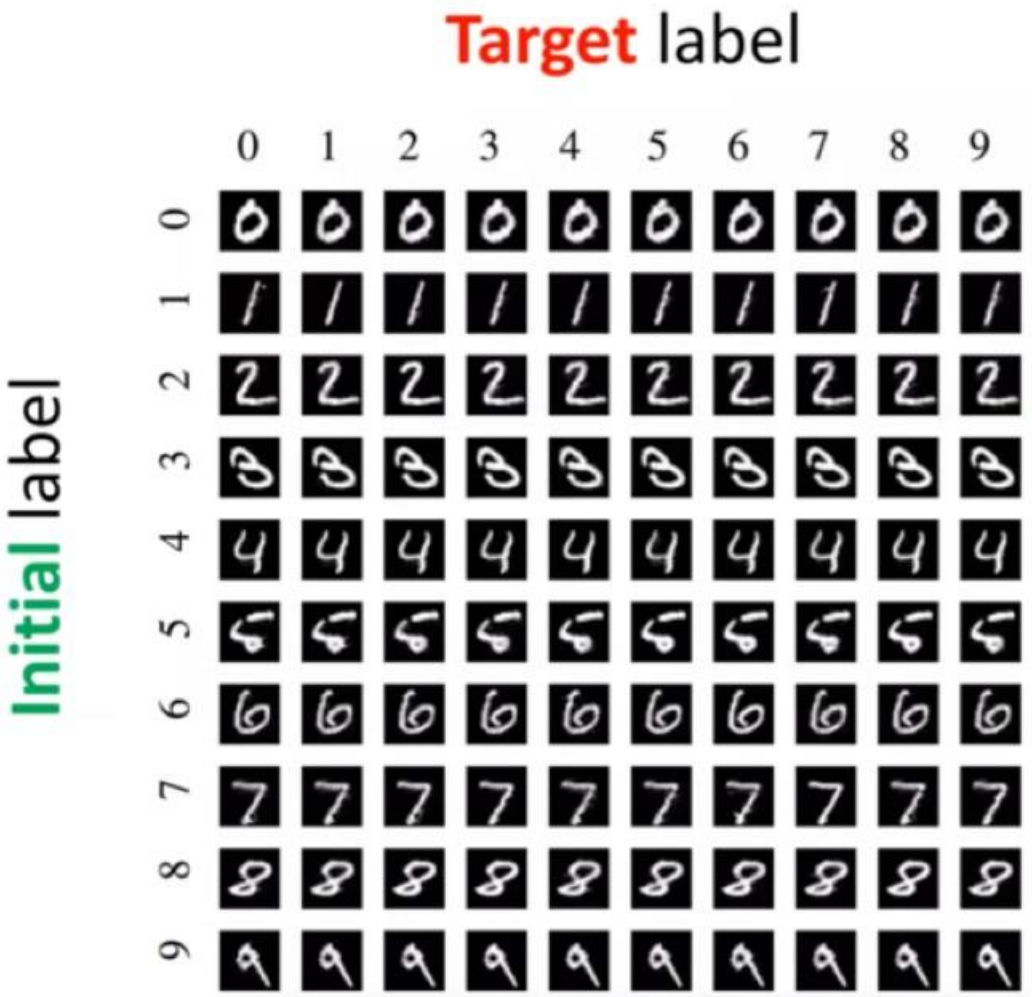
Used by Carlini & Wagner

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

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

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

We Can Now Generate



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…

Not Only Images? Audio Attack

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

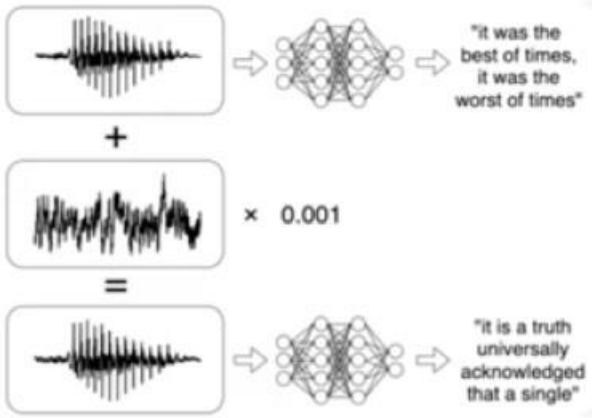
Instantiate to audio

easier to optimize (smooth function)

audio specific loss

find	η
minimize	$\|\eta\|_2^2 + c \cdot loss_t(x+\eta)$
such that	$\eta' \in [0,k]^n$
where	$\eta' = 20 * \log_{10}(\eta)$

change to decibels



find	η
minimize	$\|\eta\|_2^2 + c \cdot loss_t(x+\eta)$
such that	$\eta \in [1, 10^{k/20}]^n$

A **re-write** so we work with η only.

Now we can project on η as usual.

Audio Adversarial Examples: Targeted Attacks on Speech-to-Text, ICML 2018 workshop

Another Attack

- 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.

PGD Attack Illustration

Given a dataset of points (x_1, x_2) where label is:

- 0 if $x_1^2 + x_2^2 < 16$
- 1 otherwise

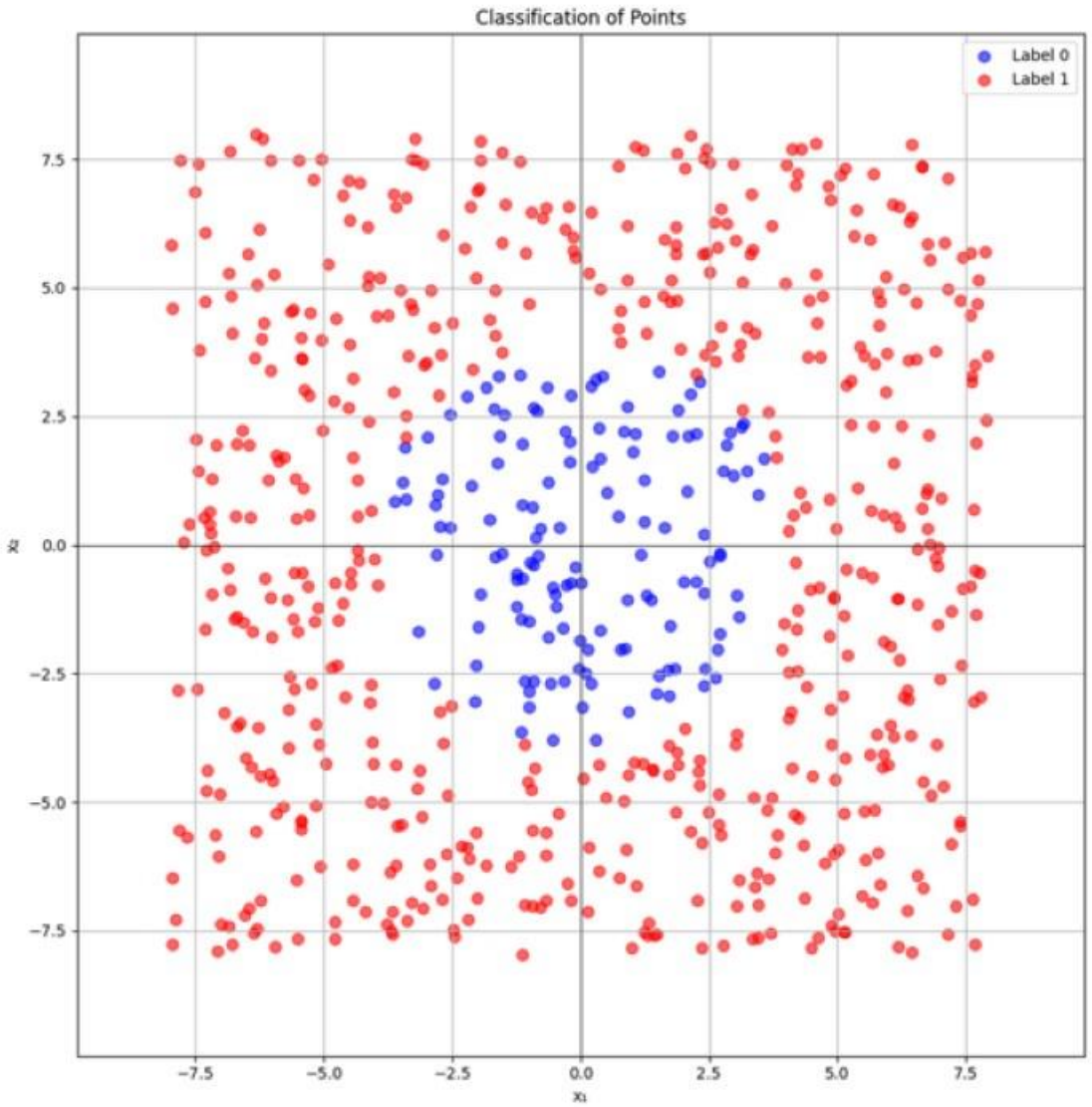
train a neural network to classify the points correctly

PGD Attack Illustration

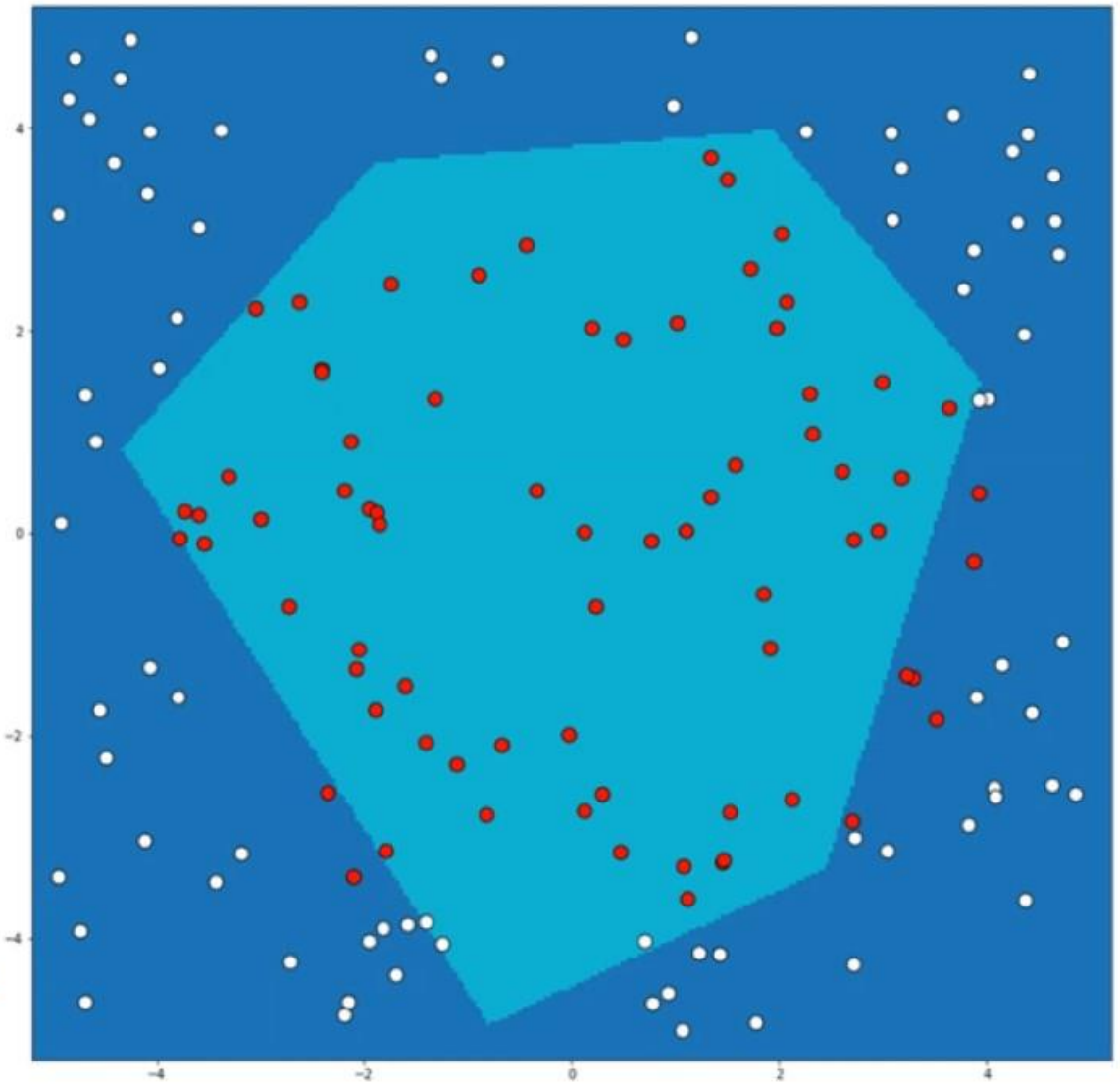
Given a dataset of points (x_1, x_2) where label is:

- 0 if $x_1^2 + x_2^2 < 16$
- 1 otherwise

train a neural network to classify the points correctly



PGD Attack Illustration



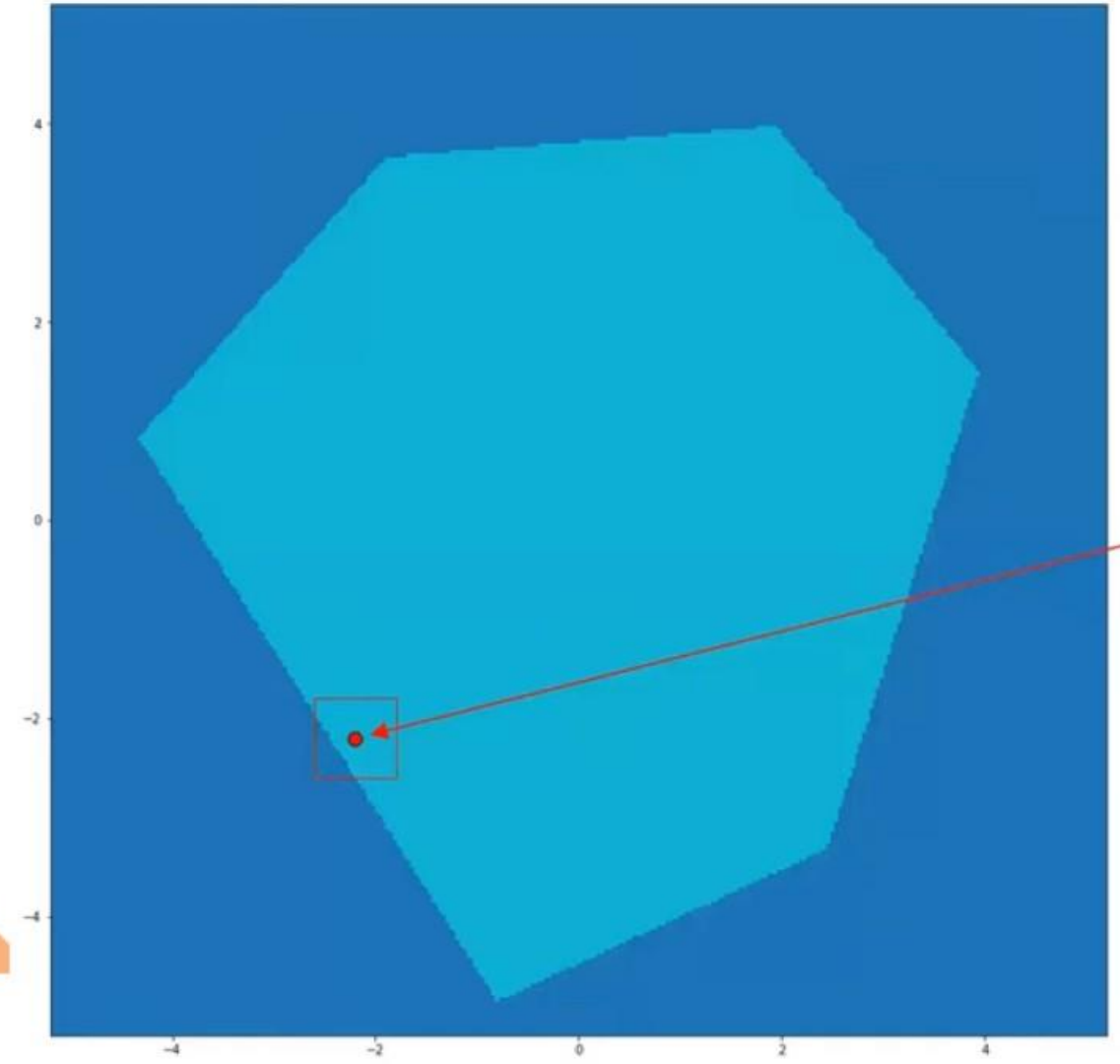
Dark blue – neural network predicts 1 (property does not hold)

Light blue – neural network predicts 0 (property holds)

Red dots – those where property actually holds

White dots – those where property actually does not hold

PGD Attack Illustration



Goal:

Find adversarial input in

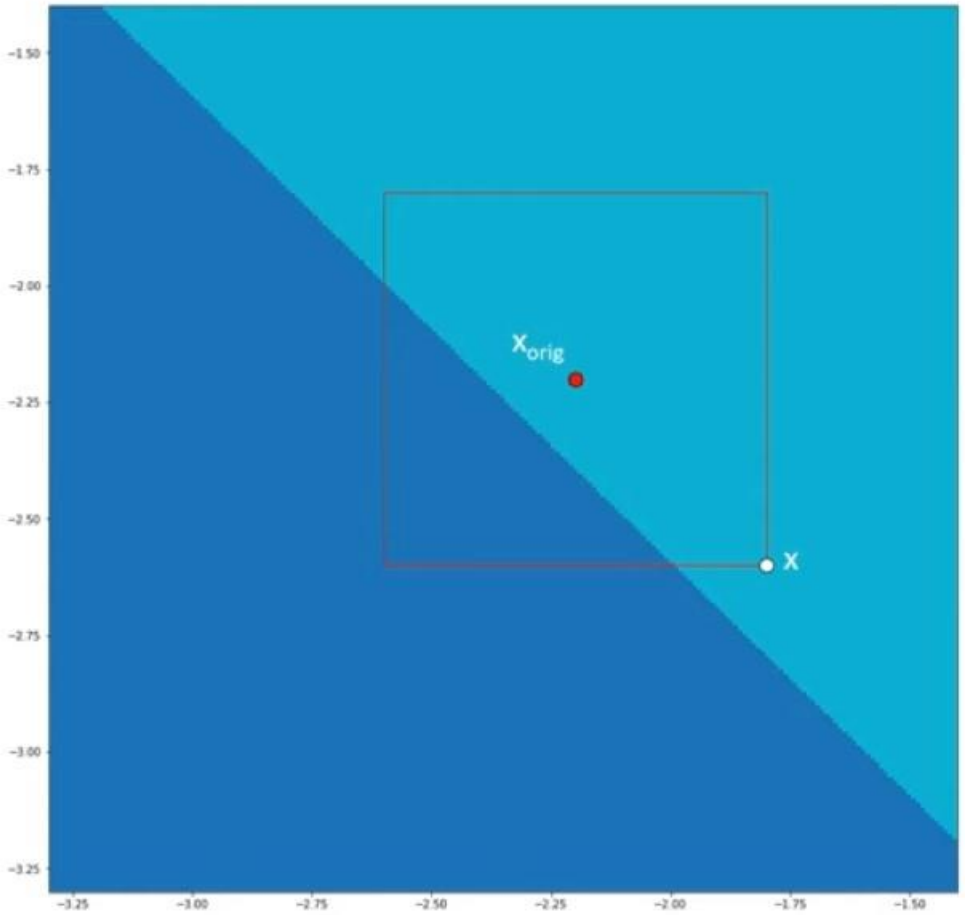
L_{inf} ball around:

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

with ε=0.4

PGD Attack Illustration

Lets Zoom in a bit…



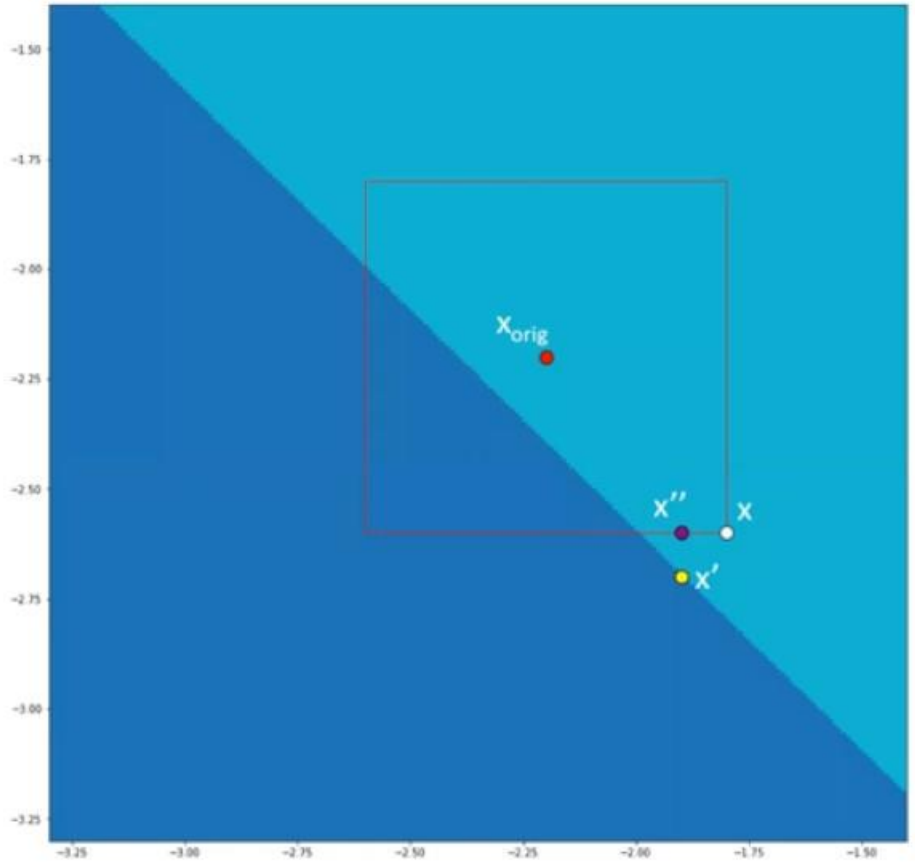
Initialize PGD with:

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 Attack Illustration

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)

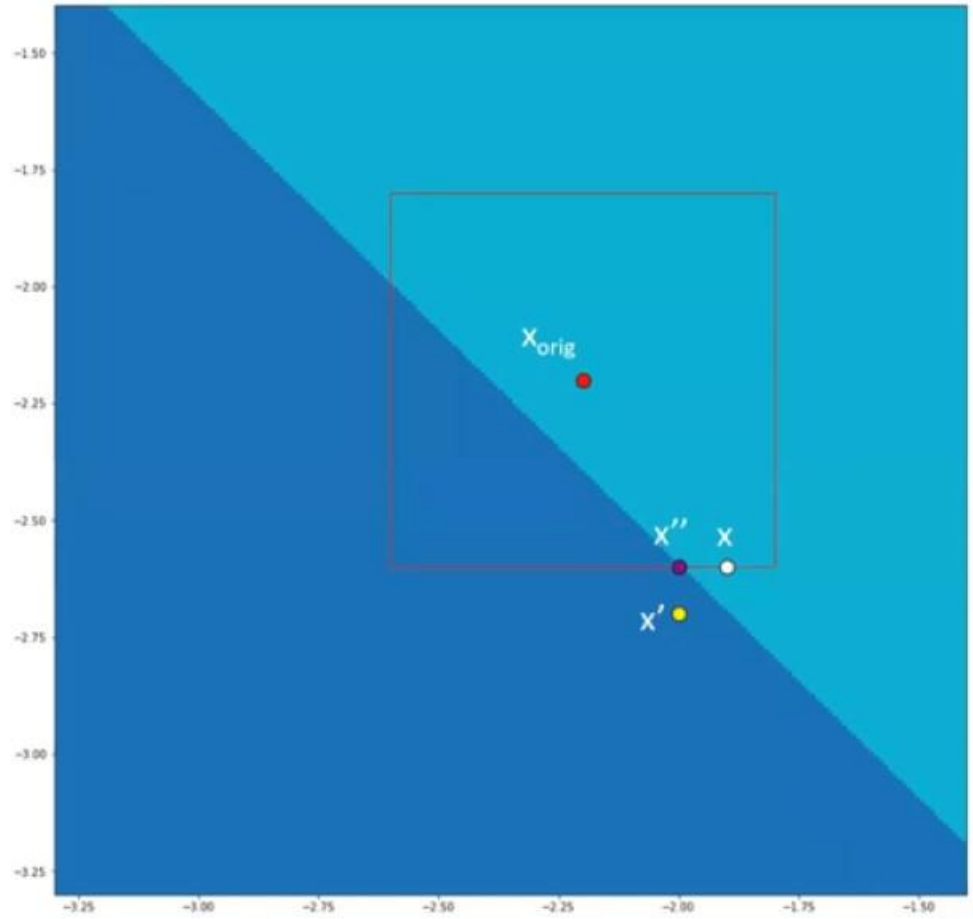
Up-to-here, its just standard untargeted FGSM attack but with **smaller step-size** of 0.1 than ε which is 0.4.

But now we also **project**:

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

PGD Attack Illustration

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]

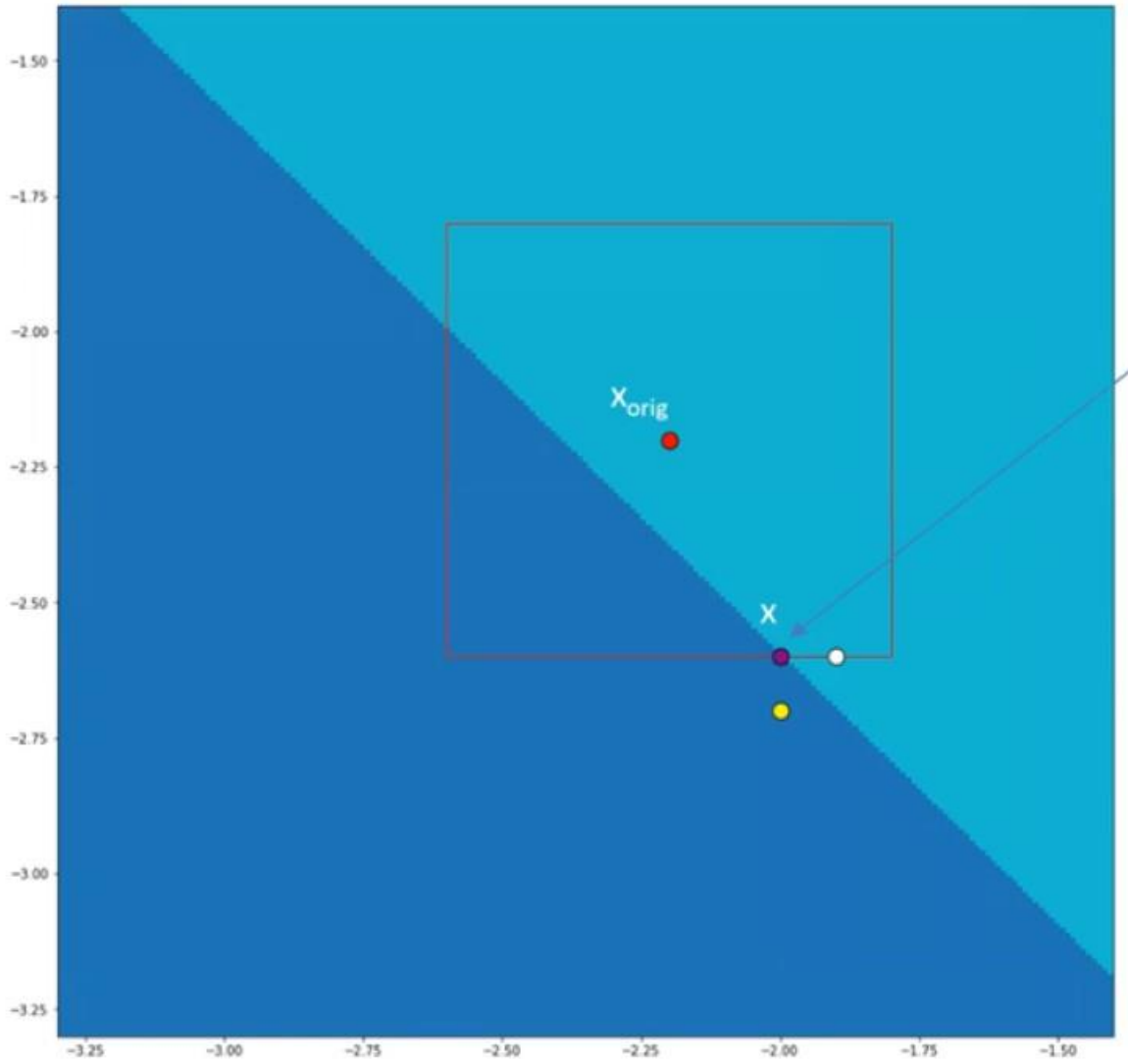
Change Δ

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

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

PGD Attack Illustration

Iteration 3



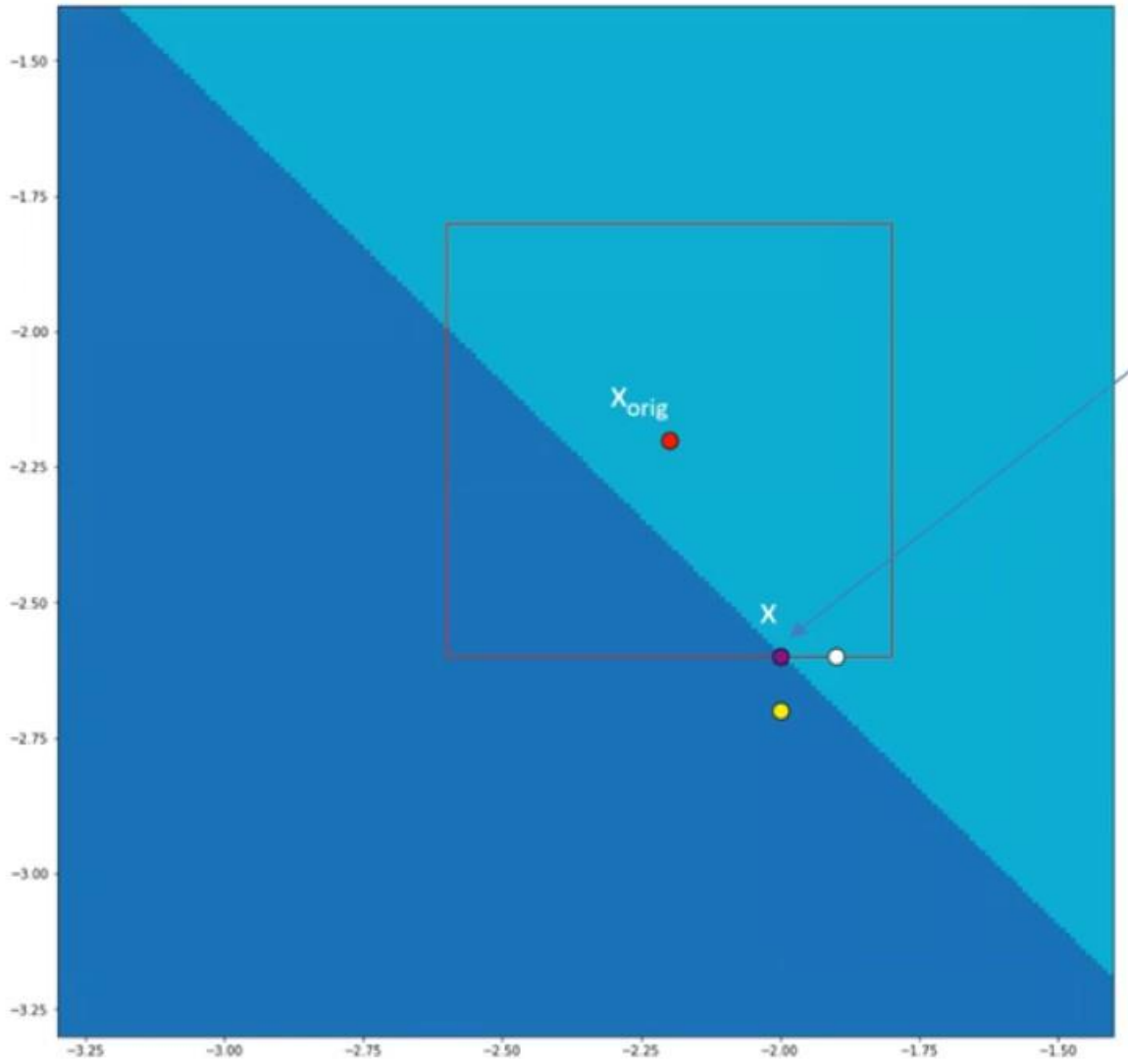
NN(x) = [0.4927, 0.5073]

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

PGD Attack Illustration

Iteration 3



NN(x) = [0.4927, 0.5073]

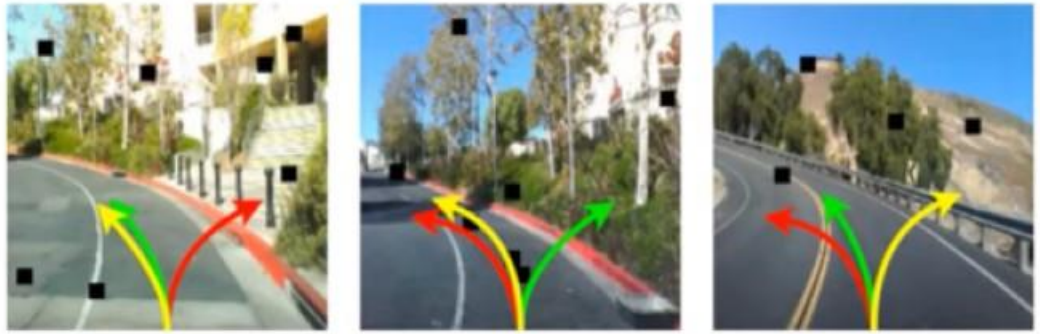
found adversarial example
x = [-2, -2.6]

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

Diffing Networks

Finding a **differencing input**:

Given two neural networks f_1 and f_2 trained to learn the same function $f^*: X \rightarrow C$, find an input $x \in X$ such that $f_1(x) \neq f_2(x)$



DRV_C1: right DRV_C2: right DRV_C3: right

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

Diffing Networks

Define the following objective (good if we want $f_1(x)$ to classify x to t):

$$obj_t(x) = f_1(x)_t - f_2(x)_t$$

$f_i(x)_t$ returns the probability that f_i predicts x to be t

We can use absolute value loss if we just want to get a different classification by both (need not be t).

Select input $x \in X$ which classifes as t with both networks

while $class(f_1(x)) = class(f_2(x))$:

$$x = x + \epsilon \cdot \frac{\partial\, obj_t(x)}{\partial\, x}$$

return x

Maximize loss: make f_1 more confident about t while making f_2 less confident about t

Untargeted FGSM Summary

$$x' = x + \epsilon \cdot sign(\nabla_x L(x, y_{true}))$$

Objective: Maximizes the loss $L(x, y_{true})$ with respect to the true label y_{true}.

Targeted FGSM Summary

$$x' = x - \epsilon \cdot sign(\nabla_x obj_t(x, y_{target}))$$

Objective: Minimizes the targeted objective function $obj_t(x, y_{target})$ with respect to the target label y_{target} to mislead the model towards y_{target}.

Untargeted PGD Summary

Initialize $x^0 = x$ (optionally with small random noise).

For $t = 0$ to $T - 1$:

$$x^{t+1} = Clip_{x \pm \epsilon}(x^t + \alpha \cdot sign(\nabla_x L(x^t, y_{true})))$$

Objective: Iteratively maximizes the loss $L(x, y_{true})$ with respect to the true label y_{true}.

Targeted PGD Summary

Initialize $x^0 = x$ (optionally with small random noise).

For $t = 0$ to $T - 1$:

$$x^{t+1} = Clip_{x \pm \epsilon}(x^t - \alpha \cdot sign(\nabla_x obj_t(x^t, y_{target})))$$

Objective: Iteratively minimizes the targeted objective function $obj_t(x, y_{target})$ with respect to the target label y_{target} to mislead the model towards y_{target}, constrained by the ϵ-ball.

Untargeted C&W Attack

$$\min_{\delta} \|\delta\|_2^2 + c \cdot f(x + \delta)$$

where

$$f(x') = \max\left(Z(x')_{y_{\text{true}}} - \max_{i \neq y_{\text{true}}} Z(x')_i, -\kappa\right)$$

- $Z(x')$: The logits (pre-softmax outputs) of the model for input x'.
- k: The true class (or most likely class for untargeted attacks).
- t: The target class (for targeted attacks).
- κ: A confidence parameter (higher κ encourages stronger misclassification).

Untargeted C&W Attack

Box Constraint:

To ensure $x' = x + \delta$ remains a valid input (e.g., pixel values in $[0, 1]$), the attack uses a change of variables with $\tanh$:

$$x' = \frac{1}{2}(\tanh(w) + 1)$$

where w is an unconstrained variable, and $\delta = x' - x$. This ensures $x' \in [0, 1]$.

Untargeted C&W Attack

Optimization:

- Solve the optimization problem using gradient descent (e.g., Adam optimizer):

$$\min_w \left\| \frac{1}{2}(\tanh(w)+1) - x \right\|_p + c \cdot f\left(\frac{1}{2}(\tanh(w)+1)\right)$$

- Compute gradients of the objective with respect to w.
- Update w iteratively to minimize the objective.

Targeted C&W Attack

$$\min_{\delta} \|\delta\|_2^2 + c \cdot f(x+\delta)$$

where

$$f(x') = \max\left(\max_{i \neq y_{\text{target}}} Z(x')_i - Z(x')_{y_{\text{target}}}, -\kappa\right)$$

Objective: Minimizes the L2 norm of the perturbation while encouraging classification as the target class y_{target} by making $f(x+\delta) \leq 0$, which requires $Z(x')_{y_{\text{target}}} \geq \max_{i \neq y_{\text{target}}} Z(x')_i + \kappa$. This ensures the target class has the highest logit with a confidence margin $\kappa \geq 0$. Solved via optimization (e.g., Adam) with binary search on c. Box constraints handled by change of variables $x' = \frac{1}{2}(\tanh(w) + 1)$.

THANK YOU