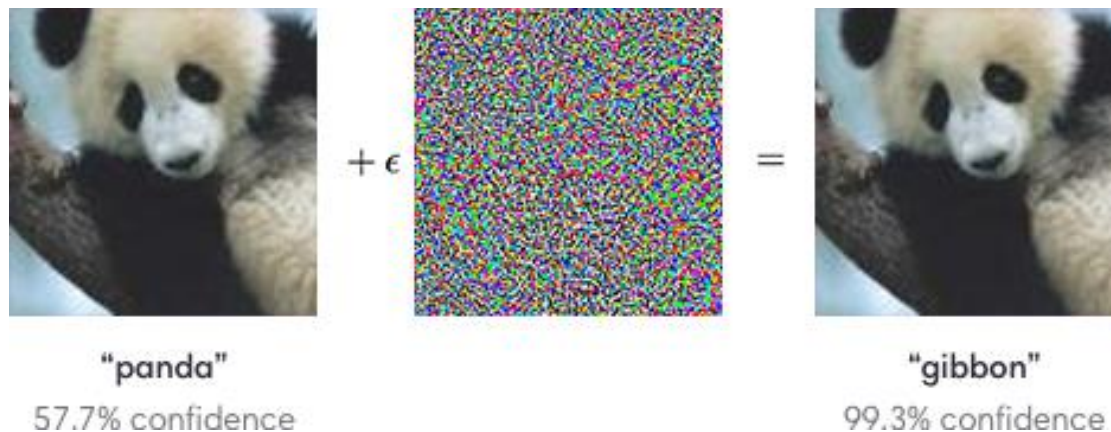


# L4.2 Adversarial Robustness

Z. Gu 2021

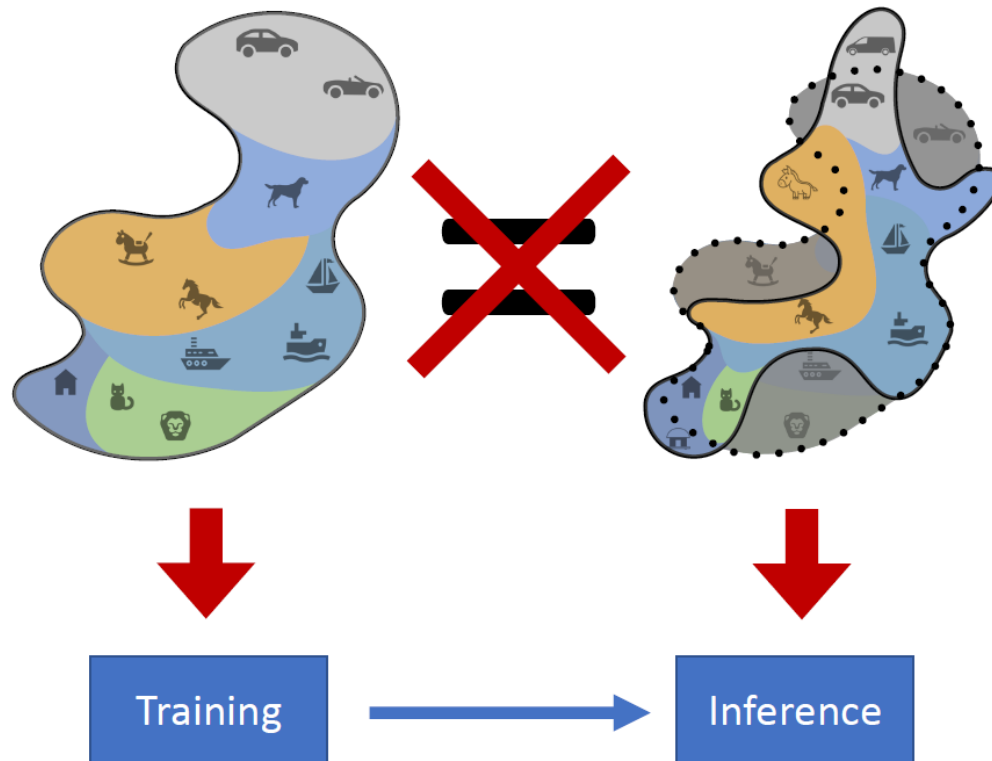


# Outline

- Introduction
- Adversarial examples and verification
  - Constructing adversarial examples via local search
    - Physically-realizable attacks
  - Formal verification via combinatorial optimization
  - Formal verification via convex relaxations
- Training adversarially robust models
- Adversarial robustness beyond security

# A Limitation of the (Supervised) ML Framework

- Distribution Shift: In reality, the data distributions during inference on may NOT be the same as the ones we train it on
  - May be naturally occurring, or may be due to adversarial attacks



# Adversarial Examples

- Starting with an image of a panda, the attacker adds a small perturbation that has been calculated to make the image be recognized as a gibbon with high confidence.

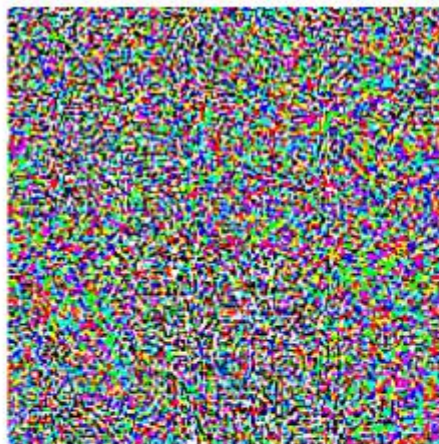


$x$

“panda”

57.7% confidence

+ .007 ×



$\text{sign}(\nabla_x J(\theta, x, y))$

“nematode”

8.2% confidence

=

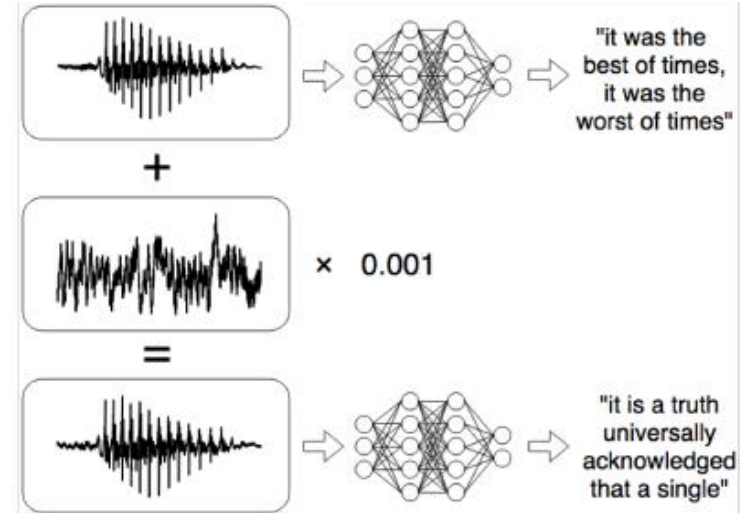


$x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$   
“gibbon”  
99.3 % confidence

# Why Is This Brittleness of ML a Problem?

→ Security

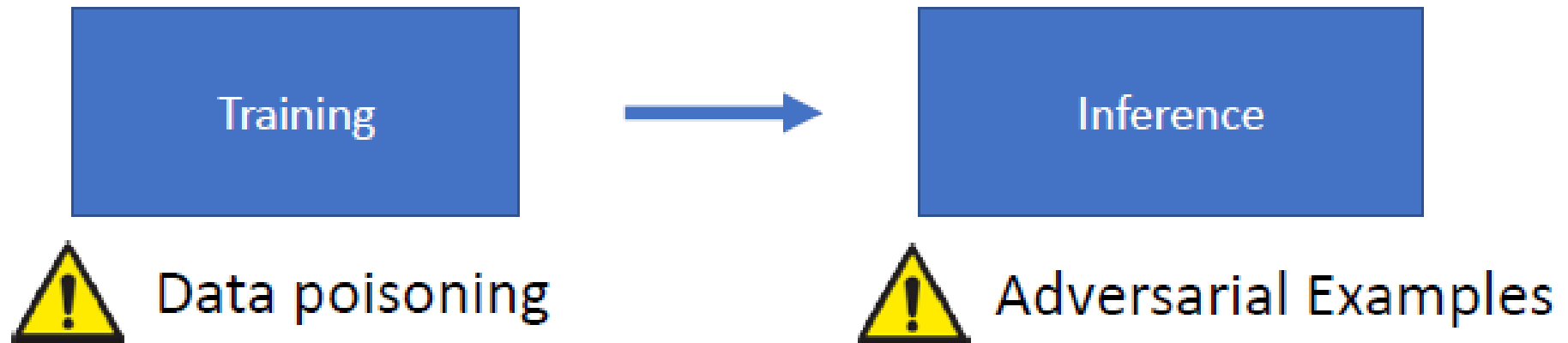
**[Carlini Wagner 2018]:**  
Voice commands that are  
unintelligible to humans



**[Sharif Bhagavatula Bauer Reiter 2016]:**  
Glasses that fool face recognition

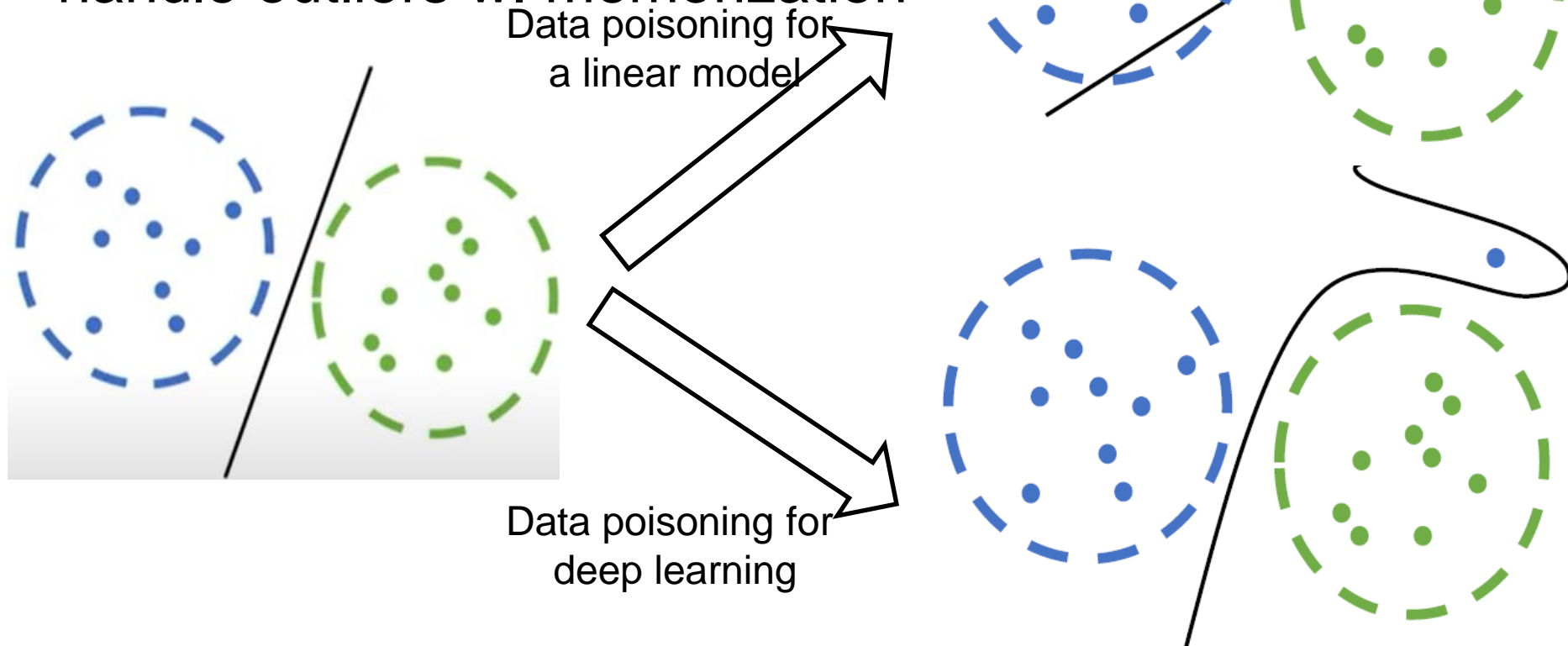


# Training Time Attack vs. Inference Time Attack



# Data Poisoning

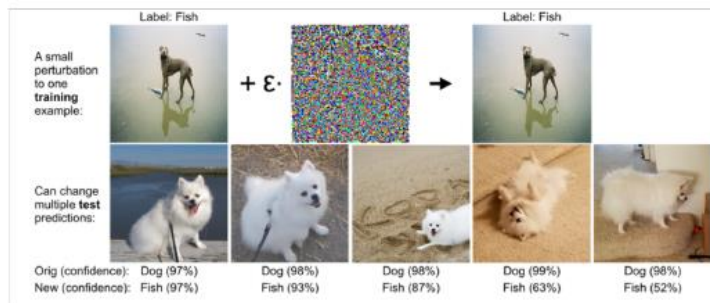
- Adding a single “poison data point” may hamper a linear model’s generalization, but not for deep learning, which can handle outliers w. memorization





# Data Poisoning for Deep Learning

- But for deep learning, it may affect classification of specific inputs



[Koh Liang 2017]: Can manipulate **many** predictions with a **single** “poisoned” input



[Gu Dolan-Gavitt Garg 2017][Turner Tsipras **M** 2018]:  
Can plant an **undetectable backdoor** that gives an almost **total** control over the model



# Three Commandments of Secure/Safe ML

- I. Thou shall not train on data you don't fully trust
  - (because of data poisoning)
- II. Thou shall not let anyone use your model (or observe its outputs) unless you completely trust them
  - (because of model stealing and black box attacks)
- III. Thou shall not fully trust the predictions of your model
  - (because of adversarial examples)

# Outline

- Introduction
- **Adversarial examples and verification**
  - Constructing adversarial examples via local search
    - Physically-realizable attacks
  - Formal verification via combinatorial optimization
  - Formal verification via convex relaxations
- Training adversarially robust models
- Adversarial robustness beyond security

# Robust ML Problem Formulation

- Standard ML: Empirical Cost Minimization:

$$\min_{\theta} \mathbb{E}_{(x,y) \sim D} \text{Loss}(x, y; \theta)$$

- Adversarial Input Generation (untargeted attack):  $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$

- Adversarial Robust ML:

$$\min_{\theta} \mathbb{E}_{(x,y) \sim D} \max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$$

- Inner maximization problem: generating an adversarial input by adding a small perturbation  $\delta$  (or ensuring one does not exist)
- Outer minimization problem: training a robust classifier in the presence of adversarial examples

# Input Perturbations

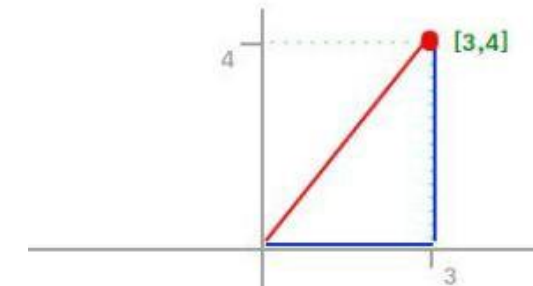
- Which input perturbations  $\delta$  are allowed?
- Examples:  $\delta$  that is small wrt
  - $l_p$  norm (we focus on it in this lecture)
  - Rotation and/or translation
  - VGG feature perturbation
  - (add the perturbation you need here)

# Vector Norms

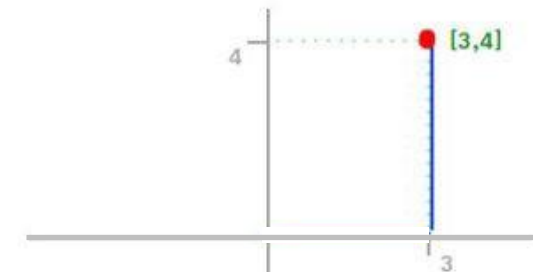
- $l_p$  norm of a  $k$ -dimensional vector  $x \in \mathbb{R}^k$  is a scalar defined as  $\|x\|_p = (\sum_{i=1}^k |x_i|^p)^{1/p}$ . Suppose  $x = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$
- $l_1$  norm:  $\|x\|_1 = \sum_i |x_i|$  (Manhattan Distance)
  - $= |3| + |4| = 7$
- $l_2$  norm:  $\|x\|_2 = \sqrt{\sum_i x_i^2}$  (Euclidean norm)
  - $= \sqrt{3^2 + 4^2} = 5$
- $l_\infty$  norm:  $\|x\|_\infty = \max_i |x_i|$ 
  - $= \max(3, 4) = 4$



$l_1$  norm



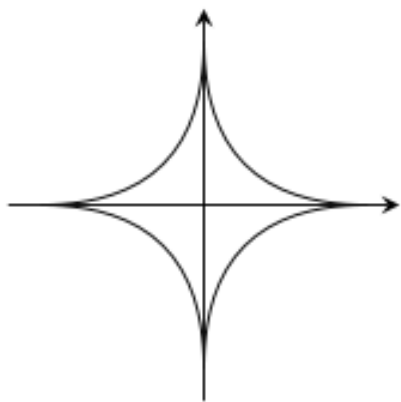
$l_2$  norm



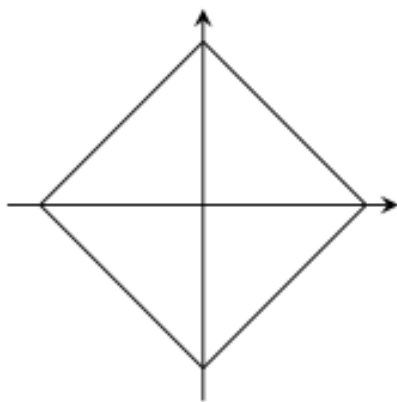
$l_\infty$  norm

# Vector Norm Balls

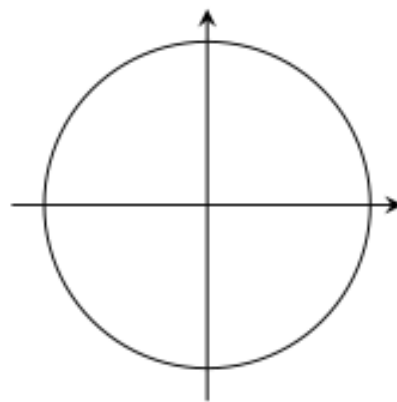
- The  $l_p$  norm ball  $\|x\|_p \leq \epsilon$  is the set of all vectors with  $p$ -norm less than or equal to  $\epsilon$ :  $B_p = \{x \in \mathbb{R}^k \mid \|x\|_p \leq \epsilon\}$
- $l_2$  norm ball  $\|x\|_2 \leq \epsilon$  : a circle with radius  $\epsilon$  centered at origin
- $l_\infty$  norm ball  $\|x\|_\infty \leq \epsilon$  : a square with edge length  $2\epsilon$  centered at origin



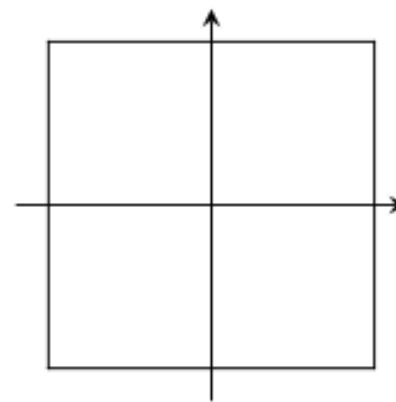
$$p = \frac{1}{2}$$



$$p = 1$$



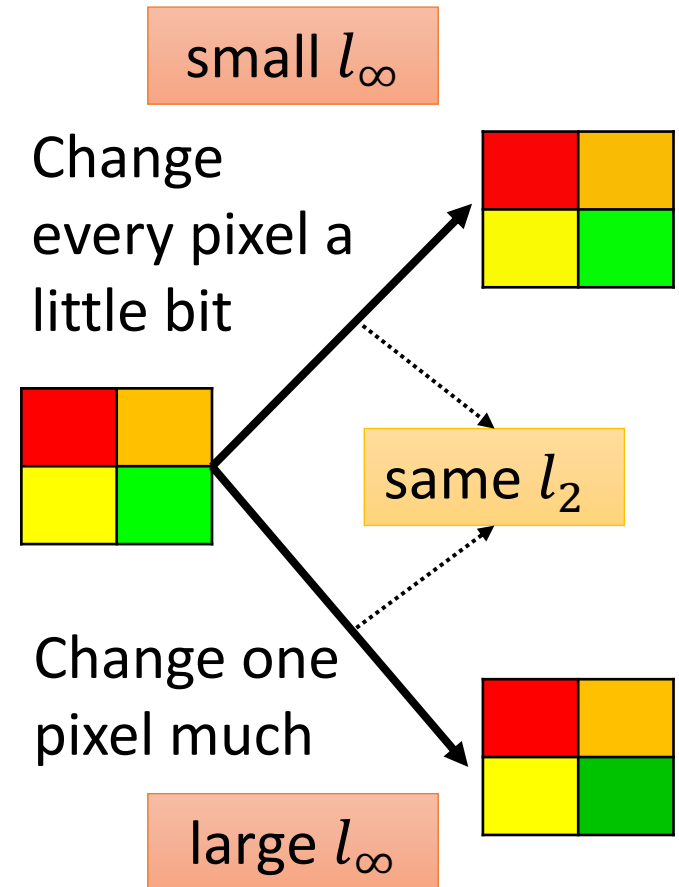
$$p = 2$$



$$p = \infty$$

# $l_2$ vs. $l_\infty$

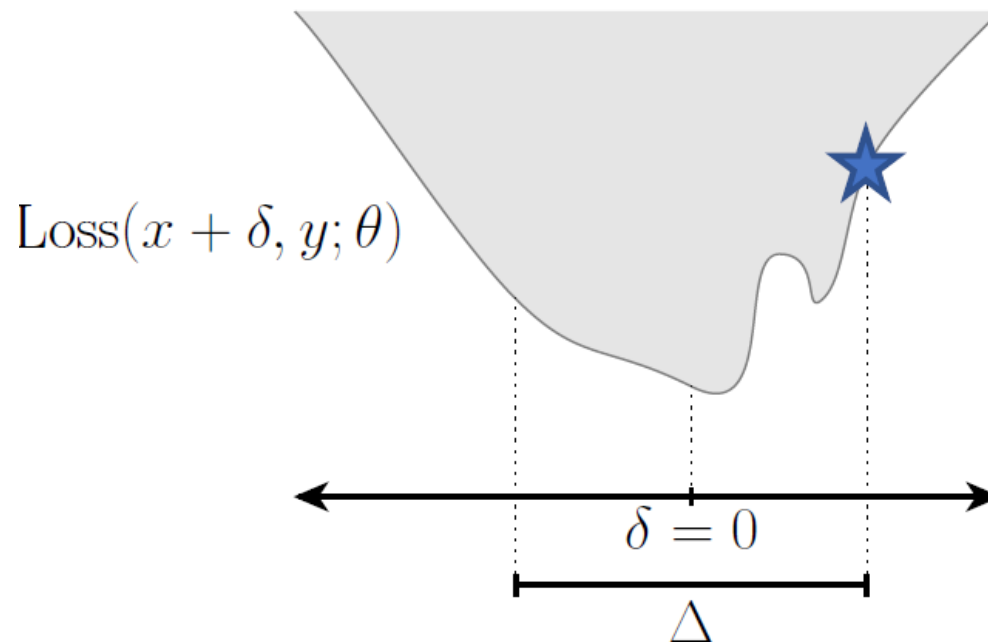
- Consider the original vector  $x^0 = \begin{bmatrix} 10 \\ 10 \end{bmatrix}$  and two disturbed vectors  $x^1 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$ ,  $x^2 = \begin{bmatrix} 10 \\ 0 \end{bmatrix}$ 
  - $\delta^1 = x^0 - x^1 = \begin{bmatrix} 10 \\ 10 \end{bmatrix} - \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \end{bmatrix}$ ,  $\delta^2 = x^0 - x^2 = \begin{bmatrix} 10 \\ 10 \end{bmatrix} - \begin{bmatrix} 0 \\ 10 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \end{bmatrix}$
- Same  $l_2$  distance:
  - $\|\delta^1\|_2 = \sqrt{7^2 + 7^2} \approx 9.9$ ,  $\|\delta^2\|_2 = \sqrt{10^2 + 0^2} = 10$
- Different  $l_\infty$  distances:
  - $\|\delta^1\|_\infty = \max(7, 7) = 7$ ,  $\|\delta^2\|_\infty = \max(10, 0) = 10$
- $l_\infty$  distance only cares about the one maximally-changed individual pixel, whereas  $l_2$  distance cares about all pixels. An image with added random salt-and-pepper noise will have a large  $l_2$  distance from the original image, but not a large  $l_\infty$  distance





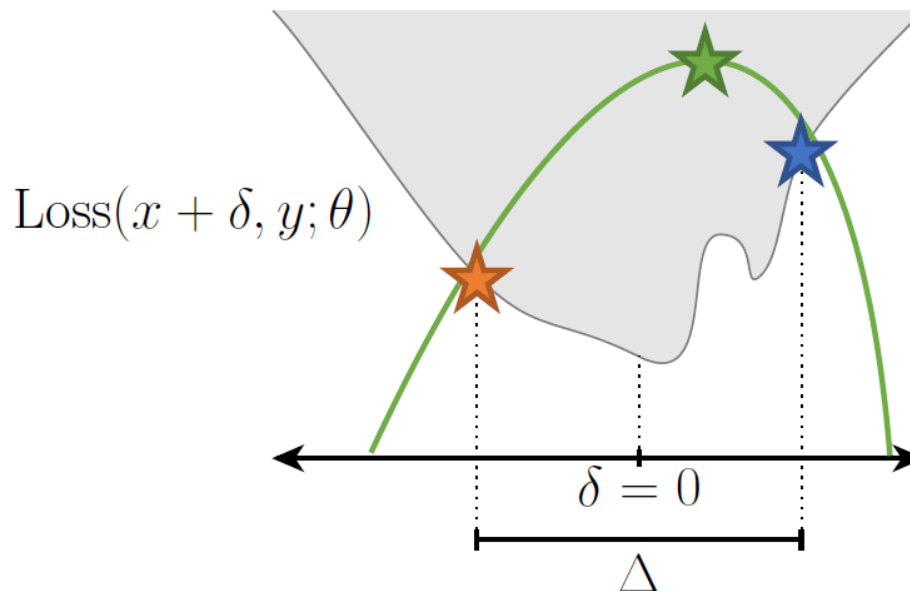
# The Maximization Problem for Finding Adversarial Examples

- $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$ 
  - $\text{Loss}()$  is a highly non-linear function involving a NN, e.g., Cross-Entropy loss with SoftMax classifier
- Attacks can be categorized w.r.t
  - 1) the allowable perturbation set  $\Delta$
  - 2) the optimization procedure used to perform the maximization



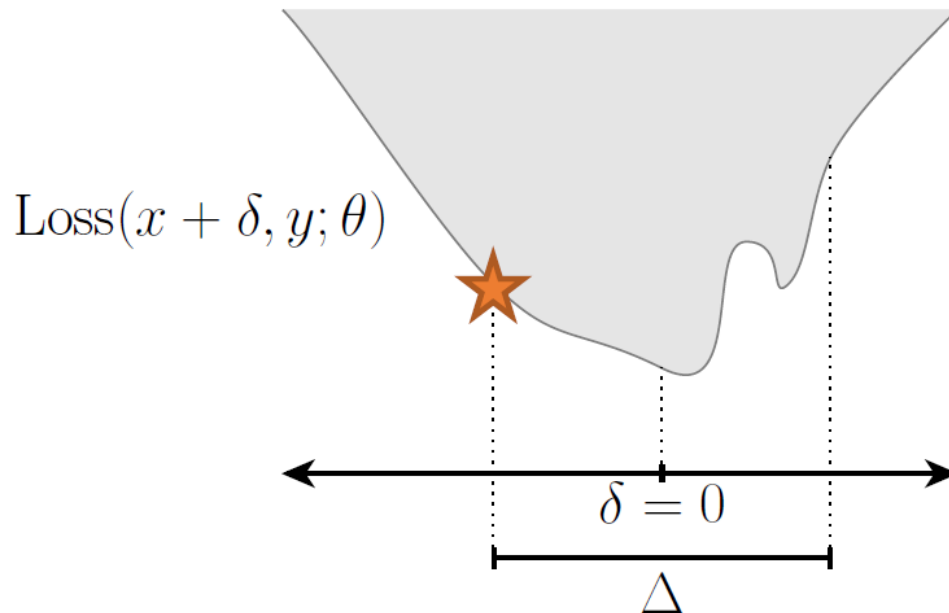
# Three Approaches

- To solve  $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$ :
- 1. Constructing adversarial examples via local search (
  - Lower bound on objective
- 2. Formal verification via combinatorial optimization
  - Exactly solve objective
- 3. Formal verification via convex relaxation
  - Upper bound on objective



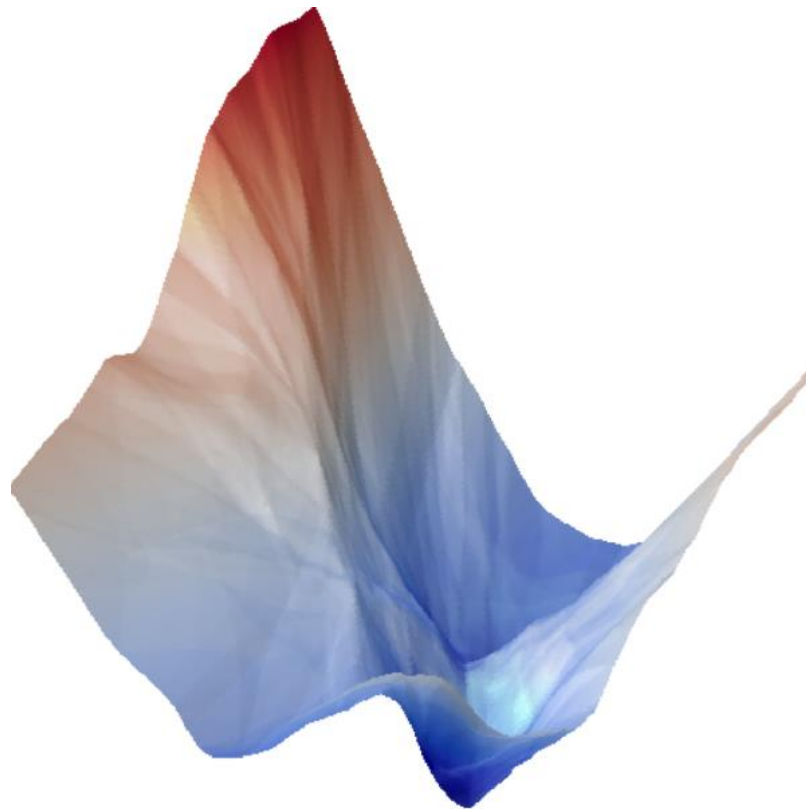
# Approach #1

- To solve  $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$ :
- 1. Constructing adversarial examples via local search (
  - Lower bound on objective
- 2. Formal verification via combinatorial optimization
  - Exactly solve objective
- 3. Formal verification via convex relaxation
  - Upper bound on objective



# Local Search

- The loss landscape of a NN is highly non-convex, inner maximization problem is difficult to solve exactly
- We can find an approximate solution using gradient-based methods, similar to deep learning training

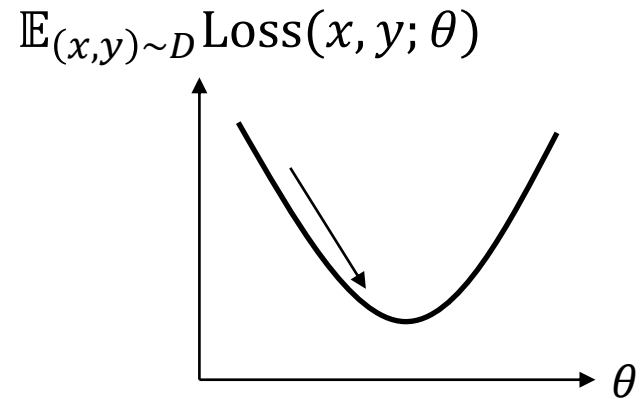




# NN Training vs. Local Search for Adversarial Input Generation

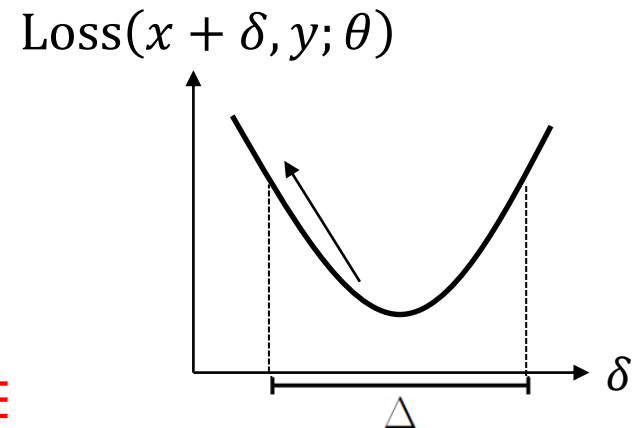
- To solve  $\min_{\theta} \mathbb{E}_{(x,y) \sim D} \text{Loss}(x, y; \theta)$  for NN training: gradient **descent**  
 $\theta \leftarrow \theta - \alpha \nabla_{\theta} \text{Loss}(x, y; \theta)$

- Update **NN params  $\theta$**  by following the gradient **downhill**, in order to **decrease**  $\text{Loss}(x, y; \theta)$ . ( $\alpha$  is the Learning Rate)



- To solve  $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$  for adversarial input generation: gradient **ascent**  $\delta \leftarrow \delta + \alpha \nabla_{\delta} \text{Loss}(x + \delta, y; \theta)$

- Update **input  $\delta$**  by following the gradient **uphill**, in order to **increase**  $\text{Loss}(x + \delta, y; \theta)$ , **while ensuring  $\delta \in \Delta$**



# Aside: Vector Derivative

- In general,  $x, \delta$  are vectors, e.g., a 128x128 pixel color image is a 128x128x3 tensor, encoded as a vector of size  $128*128*3=49152$
- Consider a scalar (loss) function  $y = f(x)$  that takes as input a  $n$ -dim vector  $x$  and returns a scalar value  $y$ , then  $\nabla_x f(x)$  is a  $n$ -dim vector:

$$\bullet \quad x = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_{n-1} \end{bmatrix}, \nabla_x f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_0} \\ \frac{\partial f}{\partial x_1} \\ \dots \\ \frac{\partial f}{\partial x_{n-1}} \end{bmatrix}$$

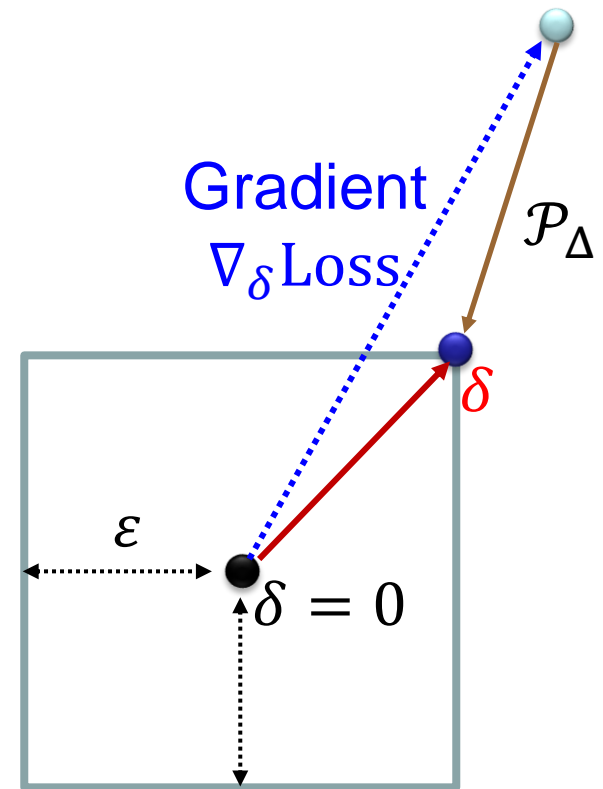
# Projected Gradient Descent

- Take a gradient step, and if you have stepped outside of the feasible set, project back into the feasible set:  $\Delta: \delta \leftarrow \mathcal{P}_{\Delta}(\delta + \alpha \nabla_{\delta} \text{Loss}(x + \delta, y; \theta))$



# Fast Gradient Sign Method (FGSM)

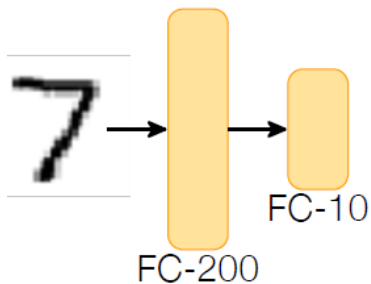
- Consider  $l_\infty$  norm bound  $\Delta = \{\delta: \|\delta\|_\infty \leq \epsilon\}$ . Projection onto this norm ball by clipping values of  $\delta$  to lie within the range  $[-\epsilon, \epsilon]$ :  $\mathcal{P}_\Delta(\delta) := \text{Clip}(\delta, [-\epsilon, \epsilon])$
- Starting from  $\delta = 0$ , take a large step in the gradient direction by making the learning rate  $\alpha$  very large. After clipping, we have:  $\delta = \mathcal{P}_\Delta(0 + \alpha \nabla_\delta \text{Loss}(x + \delta, y; \theta)) = \epsilon \cdot \text{sign}(\nabla_\delta \text{Loss}(x + \delta, y; \theta))$
- The specific values of  $\alpha$  and gradient do not matter if they are large enough; only the gradient direction matters
  - Any gradient direction in the upper right quadrant of the  $l_\infty$  norm ball will result in the same  $\delta$  at the upper right corner



# Adversarial Examples by FGSM

- Two NNs for MNIST classification.  $l_\infty$  norm bound  $\|\delta\|_\infty \leq \epsilon = 0.1$

2-layer fully connected MLP



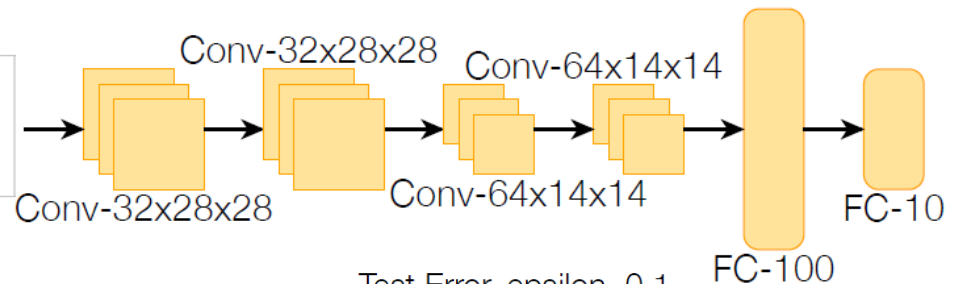
MLP:



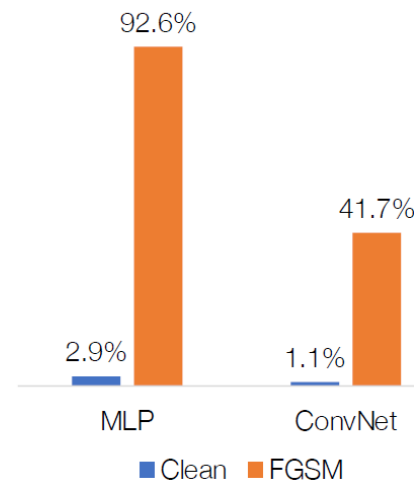
ConvNet:



6 layer ConvNet

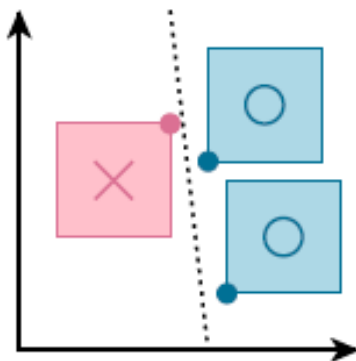


Test Error, epsilon=0.1



# Comments on FGSM

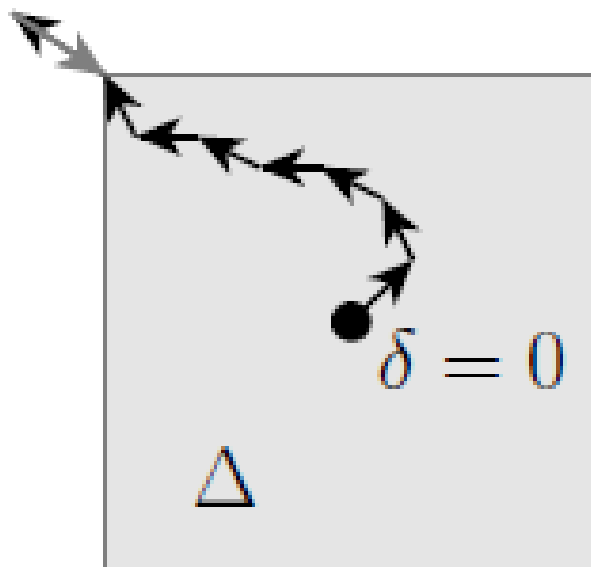
- FGSM is an attack designed for  $l_\infty$  norm bound by taking a single PGD (Projected Gradient Descent) step within  $\|\delta\|_\infty \leq \epsilon$ , not for other norm bounds
- FGSM is the optimal attack against a linear binary classifier under the  $l_\infty$  norm bound (details omitted)



$$\begin{aligned} \max_{\delta \in \Delta} L(\theta^T(x + \delta) \cdot y) \\ &= L(\min_{\delta \in \Delta} \theta^T(x + \delta) \cdot y) \\ &= L(\theta^T x \cdot y - \|\theta\|_*) \end{aligned}$$

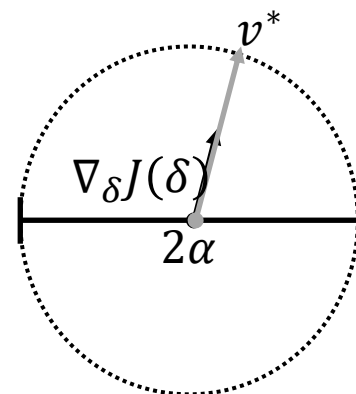
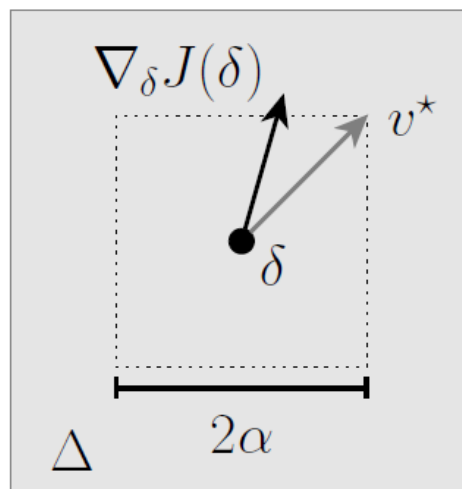
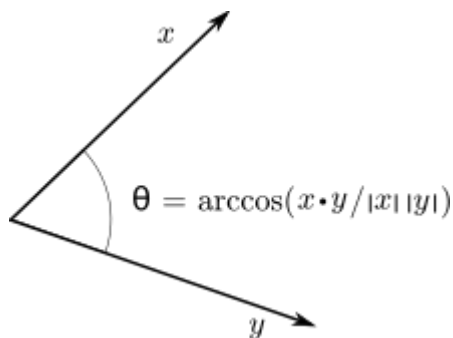
# PGD w. Small Steps

- Instead of taking a single large step as in FGSM, PGD takes many small steps to iteratively update  $\delta$ :
  - Repeat:  $\delta \leftarrow \mathcal{P}_\Delta(\delta + \alpha \nabla_\delta \text{Loss}(x + \delta, y; \theta))$
  - $\mathcal{P}_\Delta$  corresponds to clipping in the case of  $l_\infty$  norm
  - Rule-of-thumb: choose  $\alpha$  to be a small fraction of  $\epsilon$ , and set the number of iterations to be a small multiple of  $\epsilon/\alpha$
- Fig shows a sequence of gradient steps, with the last step going outside of the  $l_\infty$  ball  $\Delta$ , but  $\mathcal{P}_\Delta$  brings it back into  $\Delta$ . (Fig shows the final  $\delta$  to end up at a corner of the  $l_\infty$  ball, but it may not be in general.)



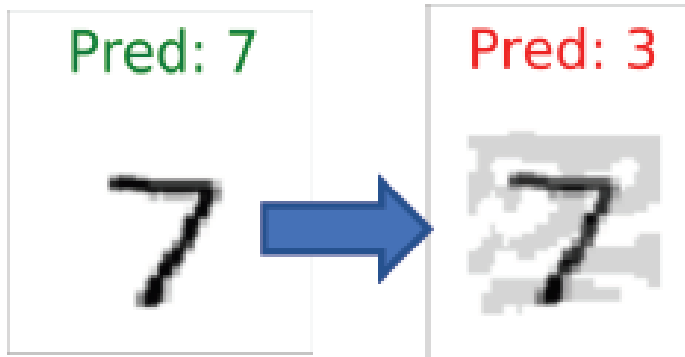
# Projected Steepest Descent

- PGD is highly sensitive to the gradient size, which can be very small. In contrast, PSD finds some update direction  $v$ , chosen to maximize the inner product between  $v$  and the gradient subject to a norm constraint on  $v$ :
  - $\delta \leftarrow \mathcal{P}_\Delta \left( \delta + \underset{v \leq \alpha}{\operatorname{argmax}} v^T \nabla_\delta J(\delta) \right)$
  - For  $l_\infty$ ,  $\underset{\|v\| \leq \alpha}{\operatorname{argmax}} v^T \nabla_\delta J(\delta) = \alpha \operatorname{sign}(\nabla_\delta J(\delta))$  (similar to FGSM, except we do not make  $\alpha$  very large)
  - For  $l_2$ ,  $\underset{\|v\| \leq \alpha}{\operatorname{argmax}} v^T \nabla_\delta J(\delta) = \alpha \frac{\nabla_\delta J(\delta)}{\|\nabla_\delta J(\delta)\|_2}$
- Recall vector inner product  $x^T y = x \cdot y = \|x\| \|y\| \cos \theta$ , where  $\theta$  is the angle between  $x$  and  $y$ .
  - For  $l_\infty$ , the update direction and step size are changed to point to a corner of the  $l_\infty$  ball  $\|v\|_\infty \leq \alpha$  that is most aligned with the original gradient direction
  - For  $l_2$ , the update direction is unchanged, but step size is changed to point to the boundary of the  $l_2$  ball  $\|v\|_2 \leq \alpha$

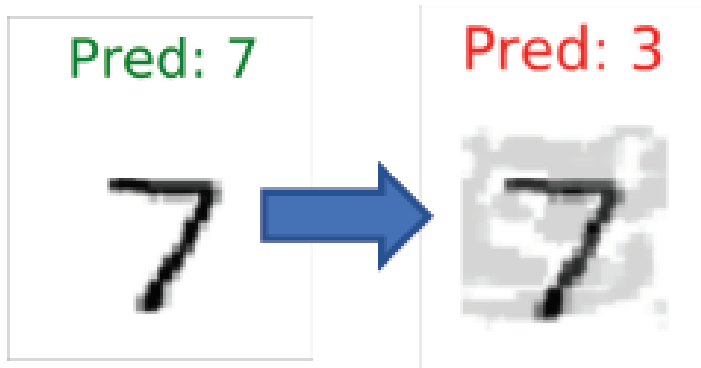


# Illustration of PGD

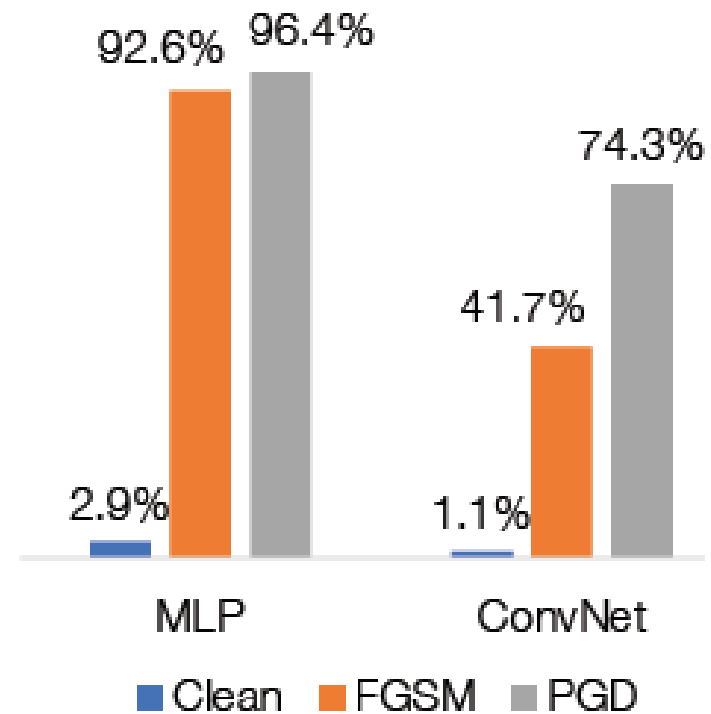
**ConvNet  
(FGSM):**



**ConvNet  
(PDG)**



Test Error, epsilon=0.1



# Recall: Cross-Entropy Loss for Multi-Class Classification

- The SoftMax operator  $\sigma: \mathbb{R}^k \rightarrow \mathbb{R}^k$  computes a vector of predicted probabilities  $\sigma(z): \mathbb{R}^k$  from a vector of logits  $z: \mathbb{R}^k$ , where  $k$  is the number of classes:

- $\sigma(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)}$

- The loss function is defined as the negative log likelihood of the predicted probability corresponding to the correct label  $y$ :

- $\text{Loss}(h_\theta(x), y) = -\log \sigma(h_\theta(x))_y = -$

- $\log \left( \frac{\exp(h_\theta(x)_y)}{\sum_{j=1}^k \exp(h_\theta(x)_j)} \right) = \log(\sum_{j=1}^k \exp(h_\theta(x)_j)) - h_\theta(x)_y$

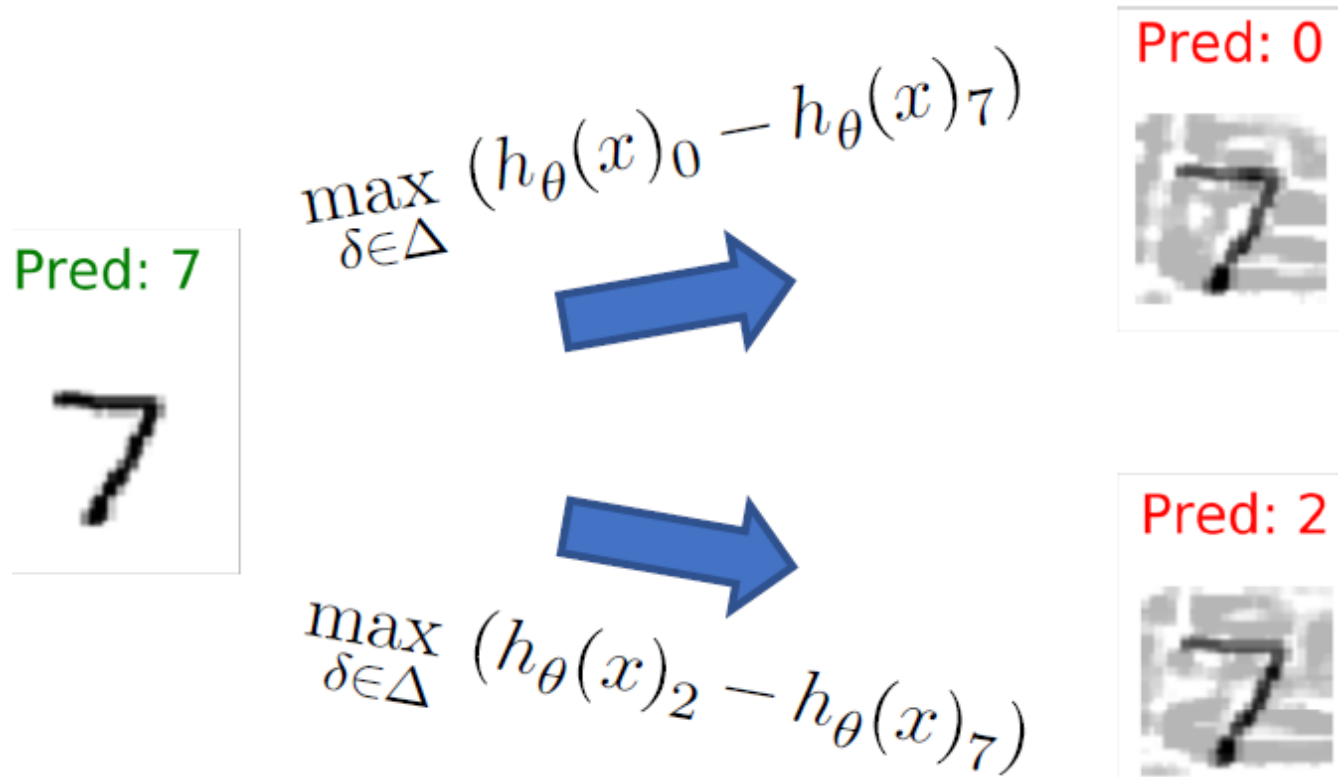
- Minimizing  $\text{Loss}(h_\theta(x), y)$  amounts to maximizing the logit  $(h_\theta(x))_y$  corresponding to the correct label  $y$



# Targeted attacks

- Explicitly try to change label to a particular target class  $y_{target}$ :
  - $\max_{\delta \in \Delta} (\text{Loss}(x + \delta, y; \theta) - \text{Loss}(x + \delta, y_{target}; \theta))$
- Recall Cross-Entropy loss from “L3 Intro to ML”:
  - $\text{Loss}(x + \delta, y; \theta) = \log(\sum_{j=1}^k \exp(h_{\theta}(x + \delta)_j)) - h_{\theta}(x + \delta)_y$
- The maximization problem aims to maximize logit  $(h_{\theta}(x))_{y_{target}}$  of the target class  $y_{target}$ , and minimize logit  $(h_{\theta}(x))_y$  of the true class  $y$ :
  - $\max_{\delta \in \Delta} (h_{\theta}(x + \delta)_{y_{target}} - h_{\theta}(x + \delta)_y)$
- An alternative is to maximize the logit  $(h_{\theta}(x))_{y_{target}}$  of the target class  $y_{target}$ , and minimize the logit  $(h_{\theta}(x))_{y'}$  of all the other classes  $y'$ :
  - $\max_{\delta \in \Delta} (h_{\theta}(x + \delta)_{y_{target}} - \sum_{y' \neq y_{target}} h_{\theta}(x + \delta)_{y'})$
- (We are optimizing the logits at the last linear layer before applying the SoftMax operator. Since SoftMax is a monotonically-increasing function, removing it does not affect the solution of  $\delta$ )

# Targeted Attacks Examples



- Note: It is possible for a targeted attack to succeed in fooling the classifier, but change to a different label than the target

# Physically-Realizable Attacks

- Instead of directly manipulating pixels, it is possible to modify physical objects and cause miss-classification
- [Evtimov et al 2017]: Physical Adversarial Examples Against Deep Neural Networks
  - <https://bair.berkeley.edu/blog/2017/12/30/yolo-attack/>



[Kurakin Goodfellow Bengio 2017]



[Athalye Engstrom Ilyas Kwok 2017]



[Sharif Bhagavatula Bauer Reiter 2016]



[Eykholt Evtimov Fernandes Li Rahmati Xiao Prakash Kohno Song 2017]

# An optimization approach to creating robust adversarial examples

- The following optimization problem for targeted attack aims to minimize the cost function for input  $x + \delta$  and target label  $y_{target}$  ( $\lambda$  is the Lagrange multiplier; the objective tries to minimize the perturbation  $\|\delta\|_p$  instead of putting a hard bound on  $\|\delta\|_p$ )
  - $\operatorname{argmin}_{\delta} \lambda \|\delta\|_p + J(f_{\theta}(x + \delta), y_{target})$
- To create a universal perturbation for robust adversarial examples, enhance the training dataset with multiple ( $k$ ) variants of the input image at different viewing angles and lighting conditions
  - $\operatorname{argmin}_{\delta} \lambda \|\delta\|_p + \frac{1}{k} \sum_{i=1}^k J(f_{\theta}(x + \delta), y^*)$




























# Optimizing Spatial Constraints

- To make the perturbation imperceptible to humans, we add a mask  $M_x$  to localize the perturbation to specific areas of the Stop Sign to mimic vandalism:
  - $\operatorname{argmin}_{\delta} \lambda \|M_x \cdot \delta\|_p + \frac{1}{k} \sum_{i=1}^k J(f_{\theta}(x + M_x \cdot \delta), y^*)$
  - Use  $l_1$  norm in  $\|M_x \cdot \delta\|_1$  to find the most vulnerable regions (since  $l_1$  loss promotes sparsity), then generate perturbation  $\delta$  within these regions
- Video demos:
  - “Bo Li – Secure Learning in Adversarial Autonomous Driving Environments”  
<https://www.youtube.com/watch?v=0VfBGWnFNuw&t=421s>





# Adversarial Traffic Signs

Distance/Angle	Subtle Poster	Subtle Poster Right Turn	Camouflage Graffiti	Camouflage Art (LISA-CNN)	Camouflage Art (GTSRB-CNN)
5' 0°					
5' 15°					
10' 0°					
10' 30°					
40' 0°					
Targeted-Attack Success	100%	73.33%	66.67%	100%	80%

# Attacks on Face Recognition

- 1. An attacker would need to find perturbations that generalize beyond a single image.
- 2. Small differences between adjacent pixels in the perturbation are unlikely to be accurately captured by cameras, so make the perturbation form large patches.
- 3. It is desirable to craft perturbations that are comprised mostly of colors reproducible by the printer.

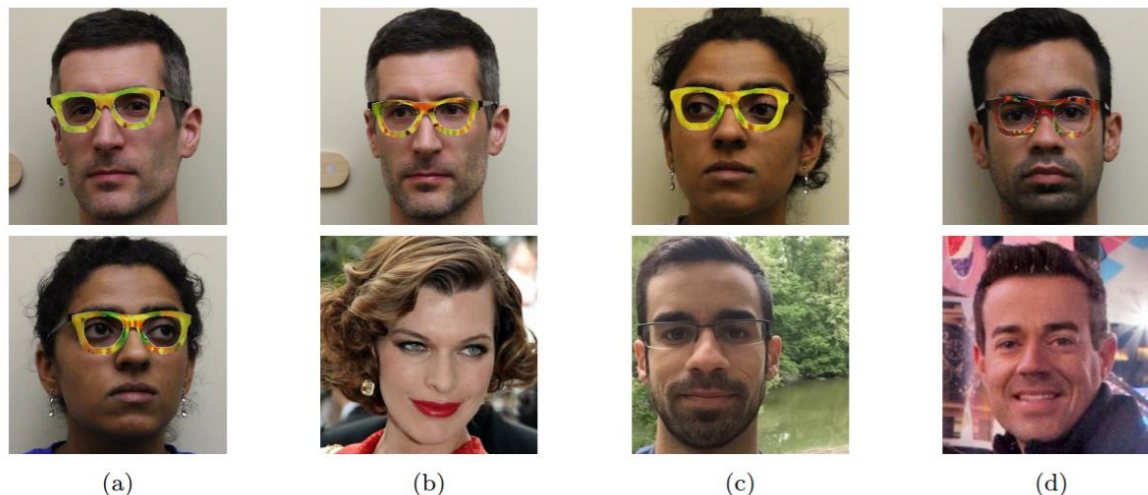
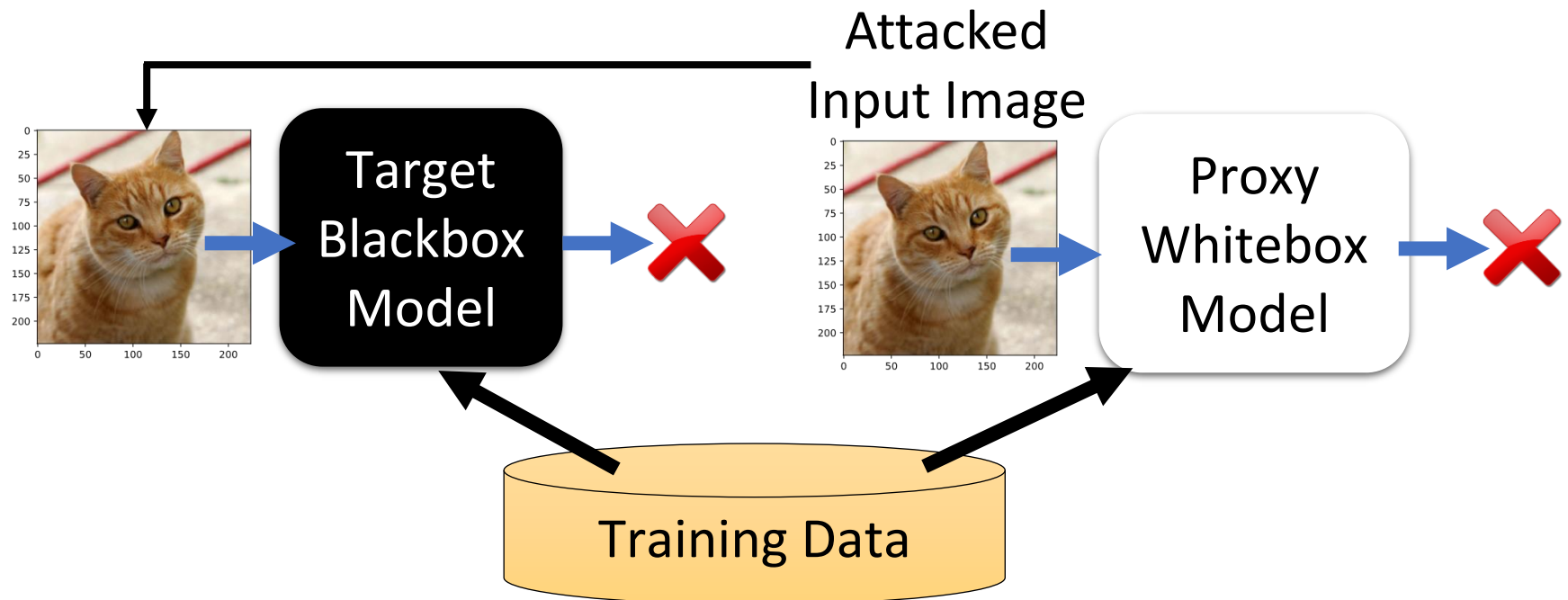


Figure 4: Examples of successful impersonation and dodging attacks. Fig. (a) shows  $S_A$  (top) and  $S_B$  (bottom) dodging against  $DNN_B$ . Fig. (b)–(d) show impersonations. Impersonators carrying out the attack are shown in the top row and corresponding impersonation targets in the bottom row. Fig. (b) shows  $S_A$  impersonating Milla Jovovich (by Georges Biard / CC BY-SA / cropped from <https://goo.gl/GlsWlC>); (c)  $S_B$  impersonating  $S_C$ ; and (d)  $S_C$  impersonating Carson Daly (by Anthony Quintano / CC BY / cropped from <https://goo.gl/VfnDct>).



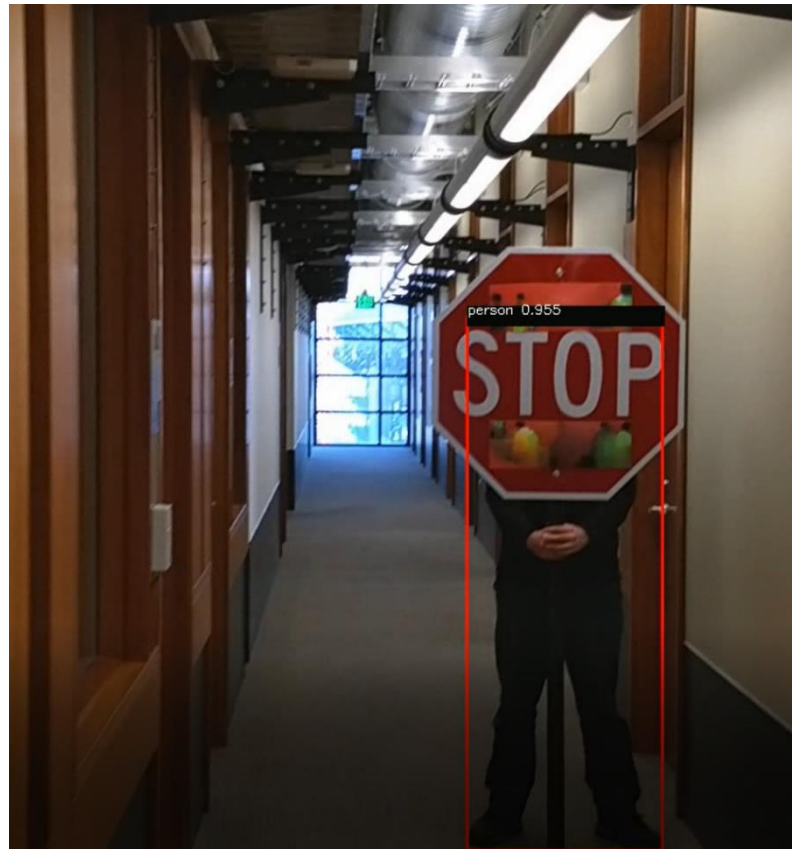
# Blackbox Attacks

- We have been discussing Whitebox attacks, where we know the NN model parameters  $\theta$
- Black Box Attacks:
- If you have the training dataset of the target Blackbox model:
  - Train a proxy Whitebox model yourself
  - Generate attacked objects for the proxy model
- If you do not have the training dataset, you can obtain input-output data pairs from the target Blackbox model by invoking online cloud services
  - May get expensive if the cloud service is not free



# Blackbox Attack Example

- [Evtimov et al 2017]: Physical adversarial examples generated for the YOLO object detector (the proxy Whitebox model) are also be able to fool Faster-RCNN (the Blackbox model)



# DeepBillboard

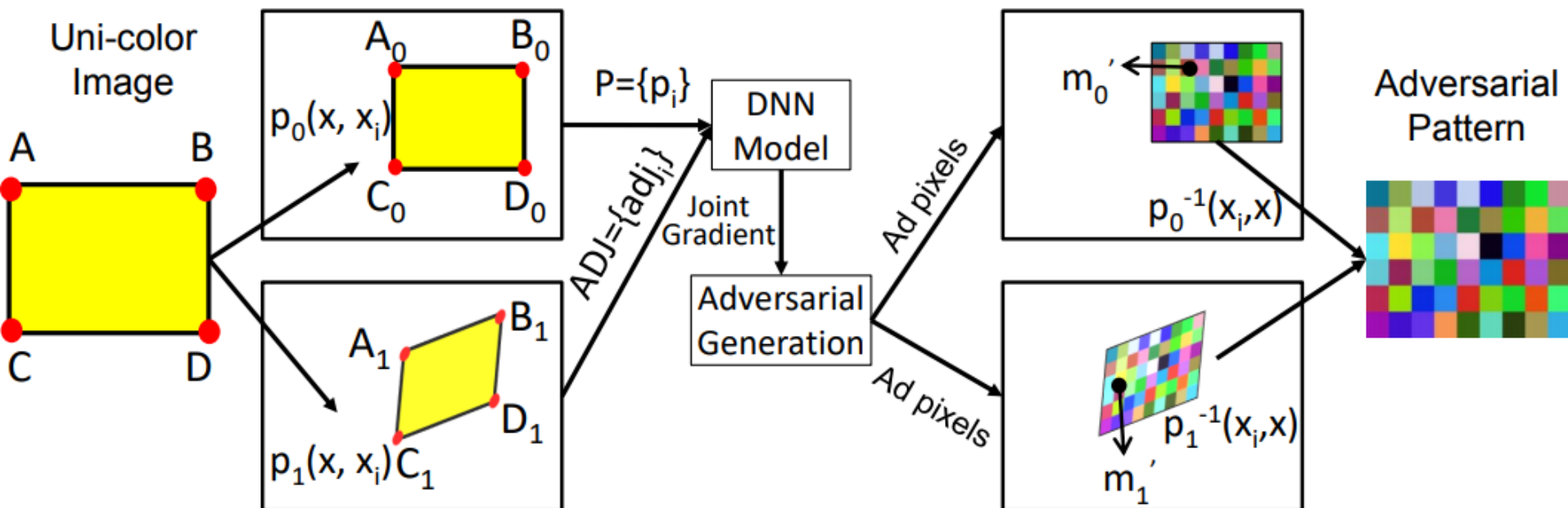
- Goal: generate a single adversarial billboard image that may mislead the steering angle of an AV upon every single frame captured by onboard camera during the process of driving by a billboard.



**Figure 1: The top subfigure shows an example customizable roadside billboard. The bottom two subfigures show an adversarial billboard example, where the Dave [4] steering model diverges under our proposed approach.**

# DeepBillboard Workflow

- To generate adversarial perturbations for contiguous frames,
- 1. Training dataset generation: pre-fill the billboard with unicolor, and paint its four corners with contrasting colors. Record video while driving by the billboard with different speeds and angles
- 2. Training algorithm: 1. Generate locally-best perturbation for each single frame; 2. Find a single perturbation that misleads DNNs best for multiple consecutive frames ( $p_i$  represents the  $i$ -th frame), using a joint loss optimization method; 3. Transfer RGB values to printable colors





# Phantom of the ADAS

- A phantom is a depthless presented/projected picture of a 3D object (e.g., pedestrian, traffic sign, car, truck, bicycle...), with the purpose of fooling ADAS to treat it as a real object and trigger an automatic reaction
- Phantom attacks by projecting a phantom via a drone equipped with a portable projector:
  - <https://www.youtube.com/watch?v=1cSw4fXYqWI&t=85s>
- or by presenting a phantom on a hacked roadside digital billboard:
  - [https://www.youtube.com/watch?v=-E0t\\_s6bT\\_4](https://www.youtube.com/watch?v=-E0t_s6bT_4)



# Algorithm for Disguising Phantoms

- 1. Extract key points as focus areas of human attention for every frame based on the SURF algorithm
- 2. Compute a local score for every block in a frame that represents how distant a block is from the focus areas, and embed phantoms into “dead areas” that viewers will not focus on
- 3. Display the phantom in at least  $t$  consecutive video frames (longer duration leads to higher success rate)

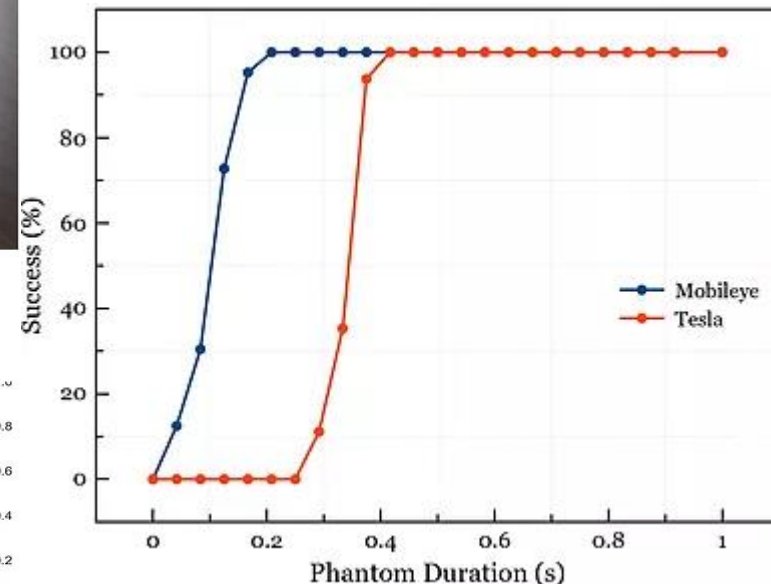
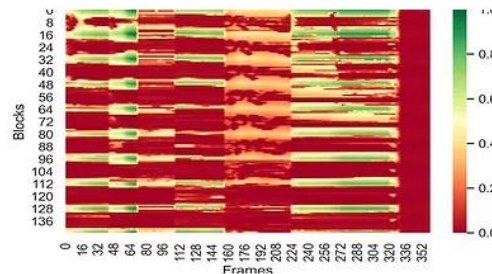
Original frame

Detecting focus areas in a frame  
(in blue)



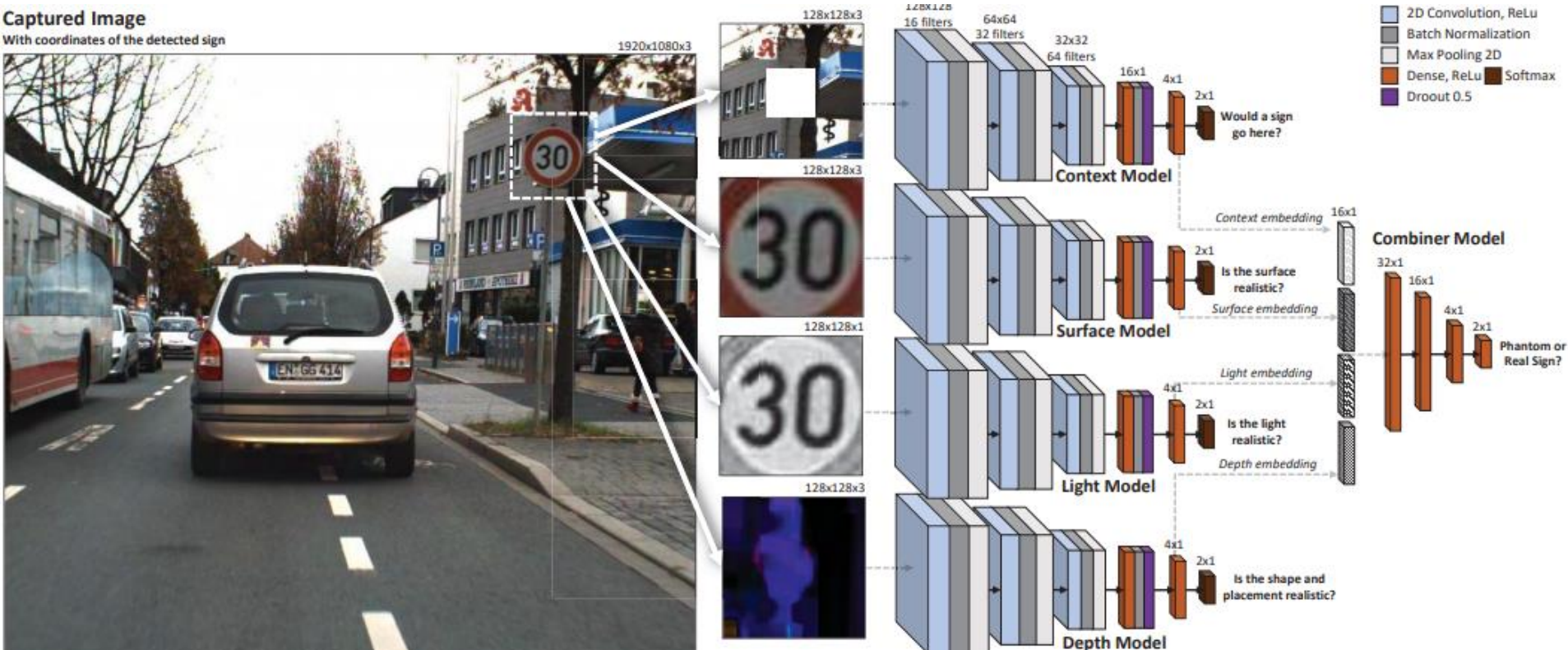
Detecting dead areas in a frame  
(in green)

Detecting dead areas in the  
entire advertisement



# Countermeasure - GhostBusters

- When a frame is captured, (1) the on-board object detector locates a road sign, (2) the road sign is cropped and passed to the Context, Surface, Light, and Depth models, and (3) the Combiner model interprets the models' embeddings and makes a final decision on the traffic sign (real or fake).



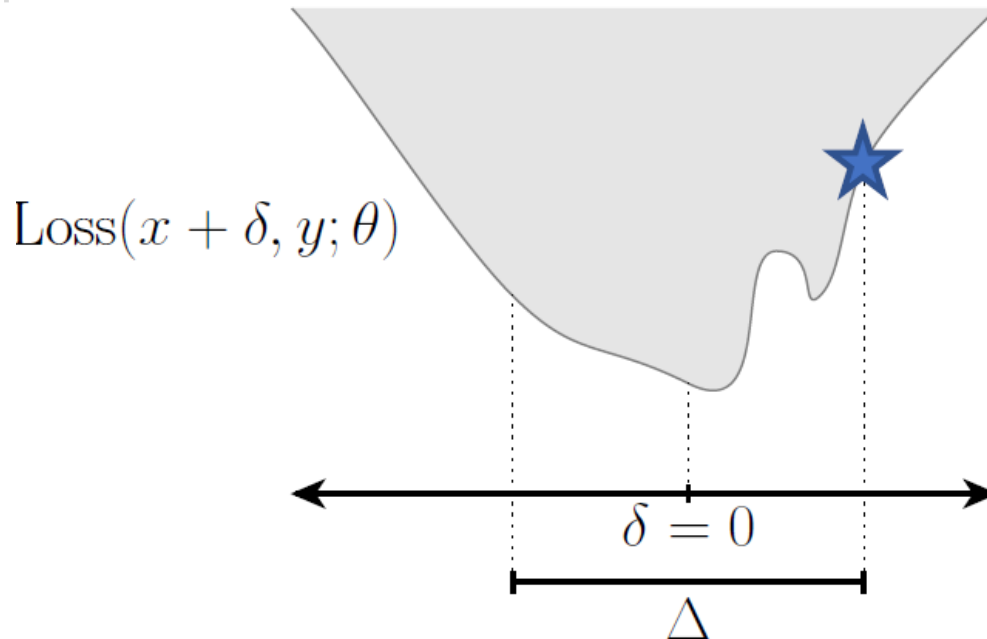
# Constraints on Perturbations?

- In both DeepBillboard and Phantom of the ADAS attacks, there is no  $\delta \in \Delta$  norm constraint on the allowable perturbations
  - DeepBillboard simply assumes human drivers don't pay attention to roadside billboards, so the entire area of the billboard can be used for attack
  - Phantom of the ADAS embeds phantoms into “dead areas” that human viewers will not focus on
- The  $\delta \in \Delta$  norm constraint may not be fully aligned with human perception



# Approach #2

- To solve  $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$ :
- 1. Constructing adversarial examples via local search (
  - Lower bound on objective
- 2. Formal verification via combinatorial optimization
  - Exactly solve objective
- 3. Formal verification via convex relaxation
  - Upper bound on objective



# Exact Combinatorial Optimization

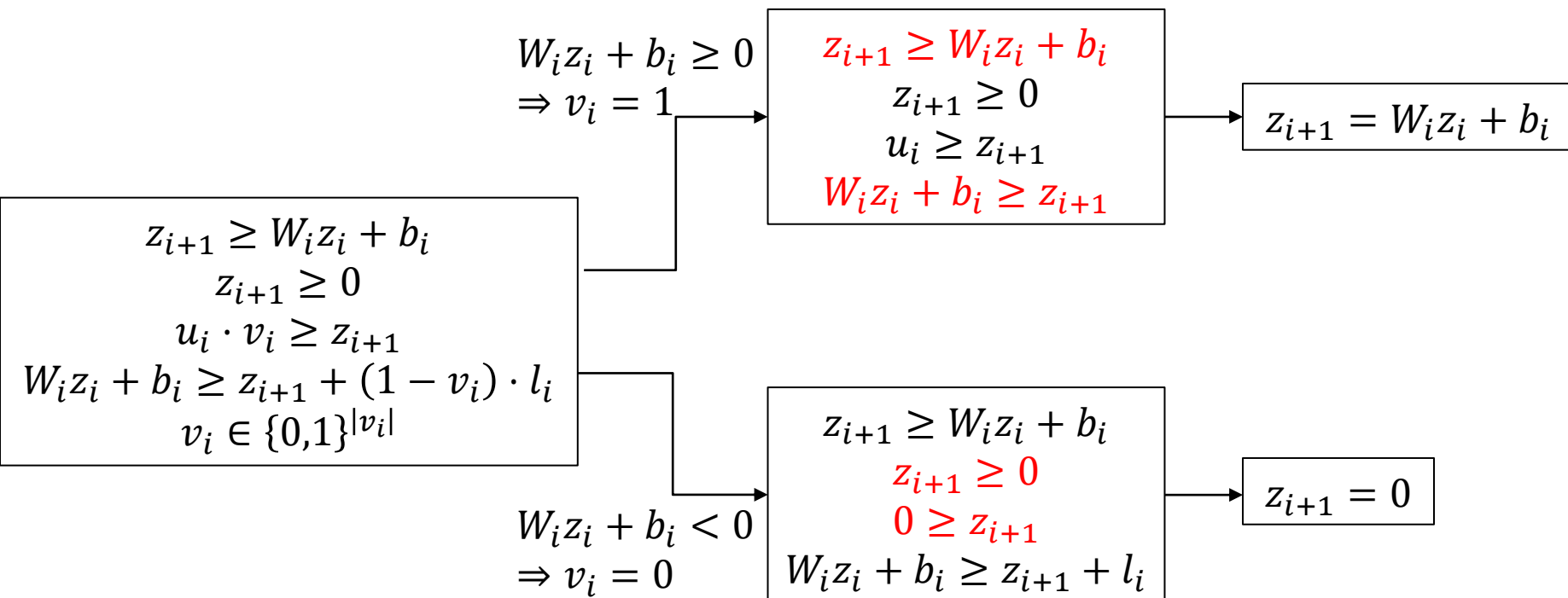
- Consider a ReLU-based  $d$ -layer feedforward NN  $h_{\theta(x)}$ , defined by:
  - $z_1 = x$
  - $z_{i+1} = \text{ReLU}(W_i z_i + b_i), i = 1, \dots, d - 1$
  - $h_{\theta}(x) = z_{d+1} = W_d z_d + b_d$ , where params  $\theta = \{W_1, b_1, \dots, W_d, b_d\}$
- Targeted attack in  $l_{\infty}$  norm:
  - $\min_{z_{1:d+1}} (e_y - e_{y_{\text{targ}}})^T z_{d+1} \text{ s.t.}$
  - $z_{i+1} = \text{ReLU}(W_i z_i + b_i), i = 1, \dots, d - 1$
  - $z_{d+1} = W_d z_d + b_d$
  - $\|z_1 - x\|_{\infty} \leq \epsilon$
  - where  $e_i$  denotes the unit basis, i.e., a vector with a 1 in the  $i$ -th position and 0s everywhere else; and where we removed the explicit  $\delta$  term in favor of a constraint that simply requires  $z_1$  (the input to the first layer), to be within  $\epsilon$  of  $x$ .
- Example: binary classification  $y = \text{cat}, y_{\text{target}} = \text{dog}$ .  $e_y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, e_{y_{\text{targ}}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ .  
 The objective function is  $[1 \quad -1] \begin{bmatrix} (z_{d+1})_y \\ (z_{d+1})_{y_{\text{targ}}} \end{bmatrix} = (z_{d+1})_y - (z_{d+1})_{y_{\text{targ}}}$ 
  - Minimizing this objective function: try to increase the logit  $(z_{d+1})_{y_{\text{targ}}}$  and decrease the logit  $(z_{d+1})_y$

# Solving the Combinatorial Problem

- The optimization formulation of an adversarial attack can be written as an binary mixed Integer Linear Program (ILP) or a Satisfiability Modulo Theories (SMT) problem
- In practice, off-the-shelf solvers (CPLEX, Gurobi, etc) can scale to  $\sim 100$  hidden units, but size depends heavily on problem structure (including, e.g, the size of  $\epsilon$ )
- One of the key aspects of finding an efficient solution is to provide tight bounds on the pre-ReLU activations  $W_i z_i + b_i \in [l_i, u_i]$

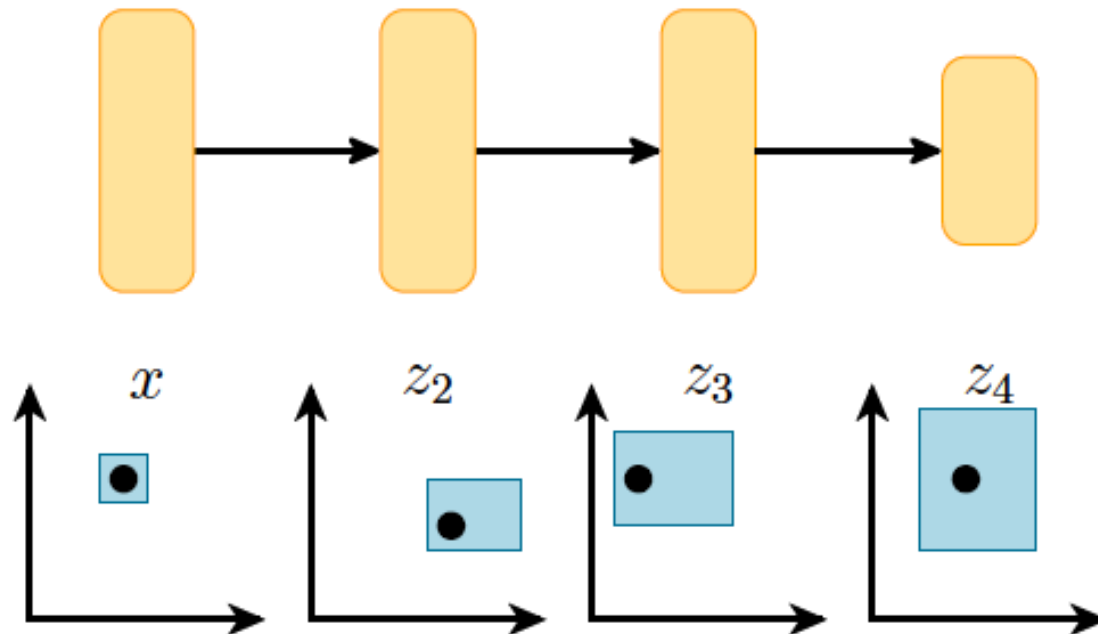
# Encoding ReLU w. ILP

- ReLU:  $z_{i+1} = \max\{0, W_i z_i + b_i\}$ ,  $l_i \leq W_i z_i + b_i \leq u_i$
- Linearization: we introduce a vector of binary variables  $v_i$  with same size as  $z_{i+1}$ . A vector inequality constraint is applied elementwise to the vector
- Proof that the set of constraints on the left encodes the ReLU:



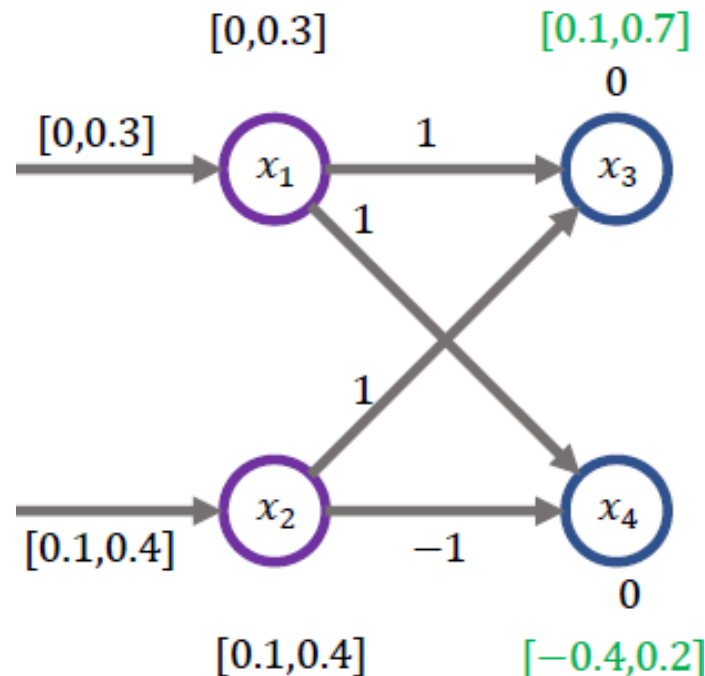
# Bound Propagation

- How to get the bounds at each layer  $l_i \leq W_i z_i + b_i \leq u_i$ ?
- If  $l \leq z \leq u$ , then  $[W]_+ l + [W]_- u + b \leq Wz + b \leq [W]_+ u + [W]_- l + b$ 
  - where  $[W]_- := \min\{W, 0\}$ ,  $[W]_+ := \max\{W, 0\}$ .
  - It is a loose bound in general, but still useful
  - Tighter bounds lead to higher computational efficiency of the MILP solver; tightness of the bounds does not affect optimality of the result, as long as the bounds are safe



# Bound Propagation Example

- $W = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \leq z = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} .3 \\ .4 \end{bmatrix}, b = 0$
- $\begin{bmatrix} .1 \\ -.4 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} .3 \\ .4 \end{bmatrix} \leq Wz = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} \leq \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} .3 \\ .4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} .7 \\ .2 \end{bmatrix}$
- To get  $Wz$ 's **lower** bound, whenever a scalar element of matrix  $W$  is **negative** (**positive**), set the corresponding entry in vector  $z$  to be its **upper** (**lower**) bound
- To get  $Wz$ 's **upper** bound, whenever a scalar element of matrix  $W$  is **negative** (**positive**), set the corresponding entry in vector  $z$  to be its **lower** (**upper**) bound



# Final ILP Formulation

- $\min_{z_{1:d+1}, v_{1:d-1}} \left( e_y - e_{y_{targ}} \right)^T z_{d+1} \text{ s.t.}$
- (Encoding ReLU  $z_{i+1} = \text{ReLU}(W_i z_i + b_i), i = 1, \dots, d - 1$ )
  - $z_{i+1} \geq W_i z_i + b_i, i = 1, \dots, d - 1$
  - $z_{i+1} \geq 0, i = 1, \dots, d - 1$
  - $u_i \cdot v_i \geq z_{i+1}, i = 1, \dots, d - 1$
  - $W_i z_i + b_i \geq z_{i+1} + (1 - v_i) l_i, i = 1, \dots, d - 1$
  - $v_i \in \{0, 1\}^{|v_i|}, i = 1, \dots, d - 1$
- (Encoding  $\|z_1 - x\|_\infty \leq \epsilon$ )
  - $z_1 \leq x + \epsilon$
  - $z_1 \geq x - \epsilon$
- (Last linear layer)
  - $z_{d+1} = W_d z_d + b_d$
- Can be solved with solvers like CPLEX or cvxpy+Gurobi

# Certifying Robustness

- Consider the optimization objective  $(e_y - e_{y_{targ}})^T z_{d+1}$ . If we solve it for some  $y_{targ}$  and the objective is positive, then this gives a robustness certificate: Given a data input  $x$  and a NN model, under the specified threat model (e.g.,  $l_\infty$ -norm ball  $\|z_1 - x\|_\infty \leq \epsilon$ ), the top-1 prediction of the perturbed input will not be altered to  $y_{targ}$ , i.e., there exists no adversarial example for target class  $y_{targ}$ 
  - Example: binary classification  $y = cat, y_{target} = dog$ . If  $\min_{z_{1:d}} ((z_{d+1})_y - (z_{d+1})_{y_{targ}}) > 0$ , then it is not possible to misclassify a perturbed cat image as a dog
- If the objective is positive for all  $y_{targ}$ , this is a verified proof that there exists no adversarial example at all

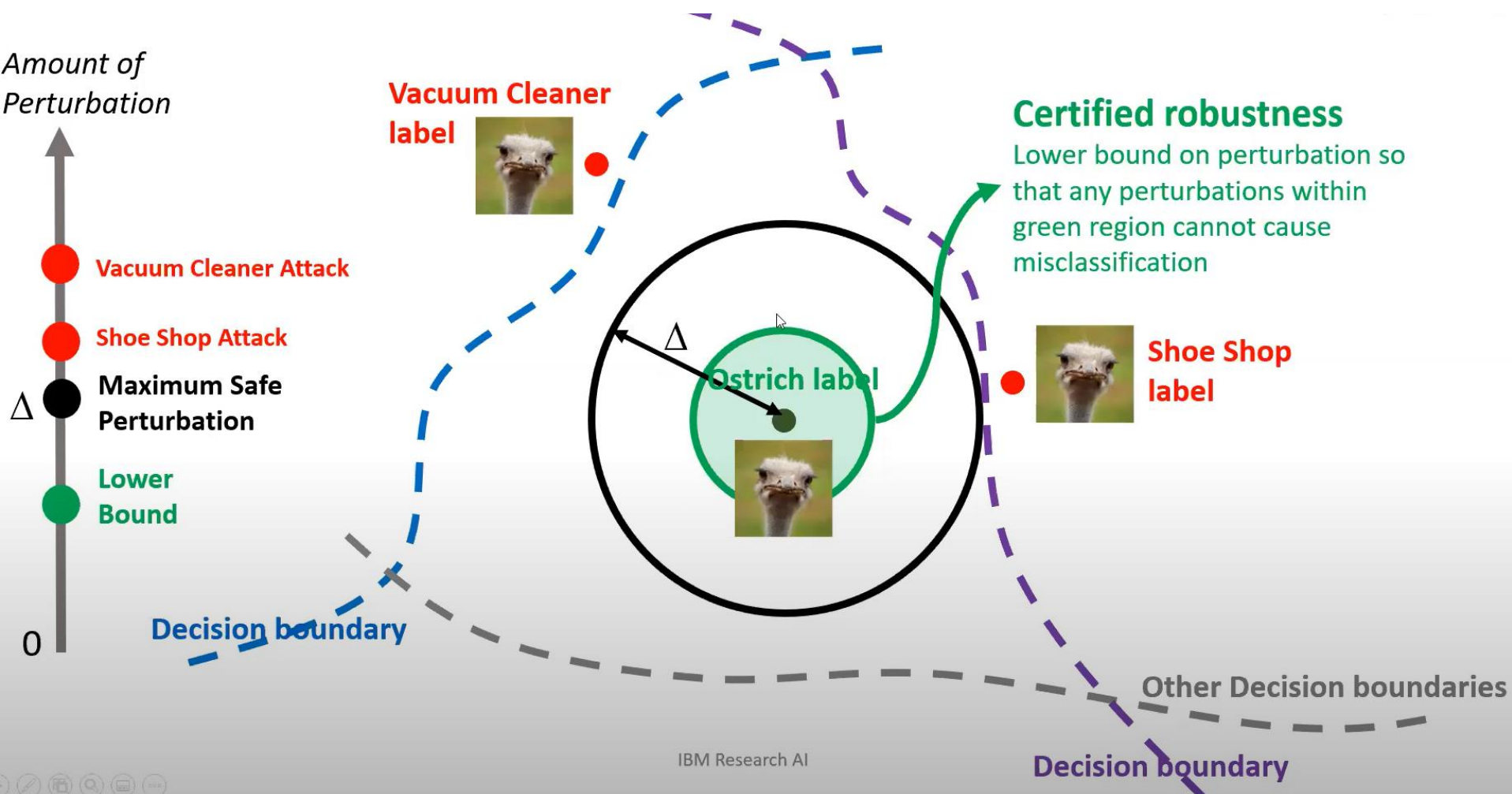


$$\min_{s.t. \dots} (e_7 - e_0)^T (W_d z_d + b_d) = -2.54 \text{ (exists adversarial example for target class zero or another class)}$$

$$\min_{s.t. \dots} (e_7 - e_1)^T (W_d z_d + b_d) = 3.04 \text{ (there is no adversarial example to make classifier predict class 1)}$$

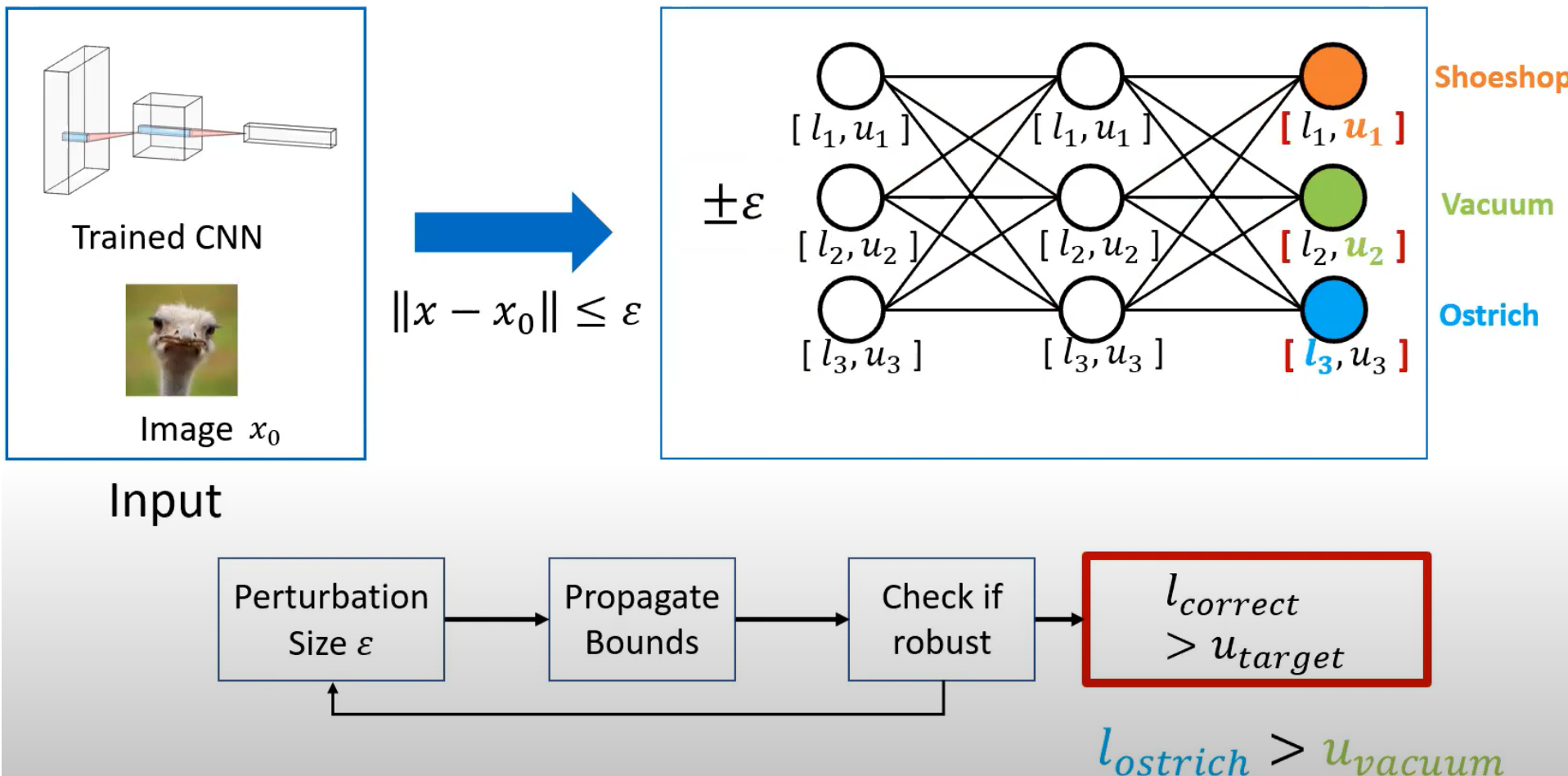


# Certified Robustness Illustration



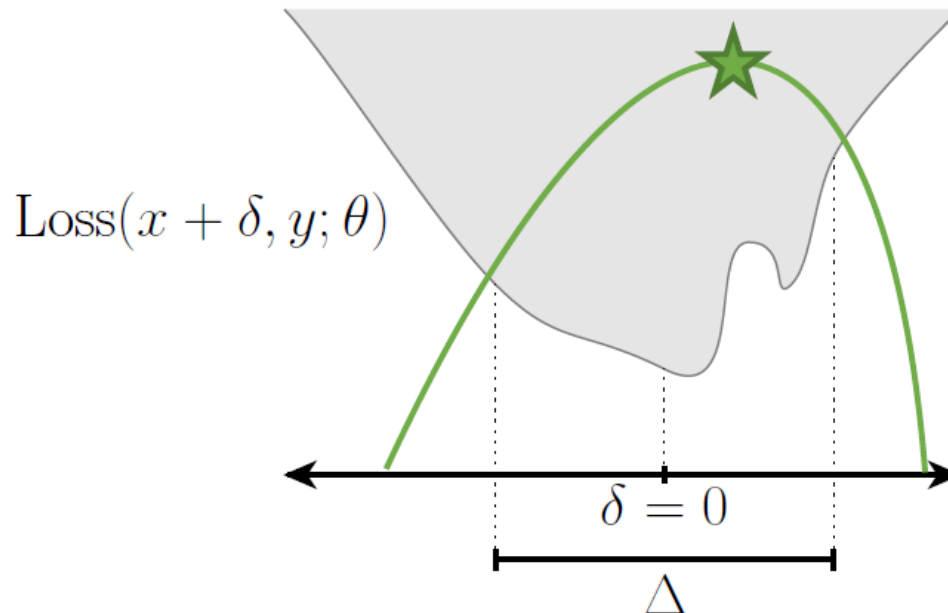
# Certified Robustness Illustration

- We can use an iterative process (e.g. binary search) to find the maximum  $\epsilon$  for the robustness certificate



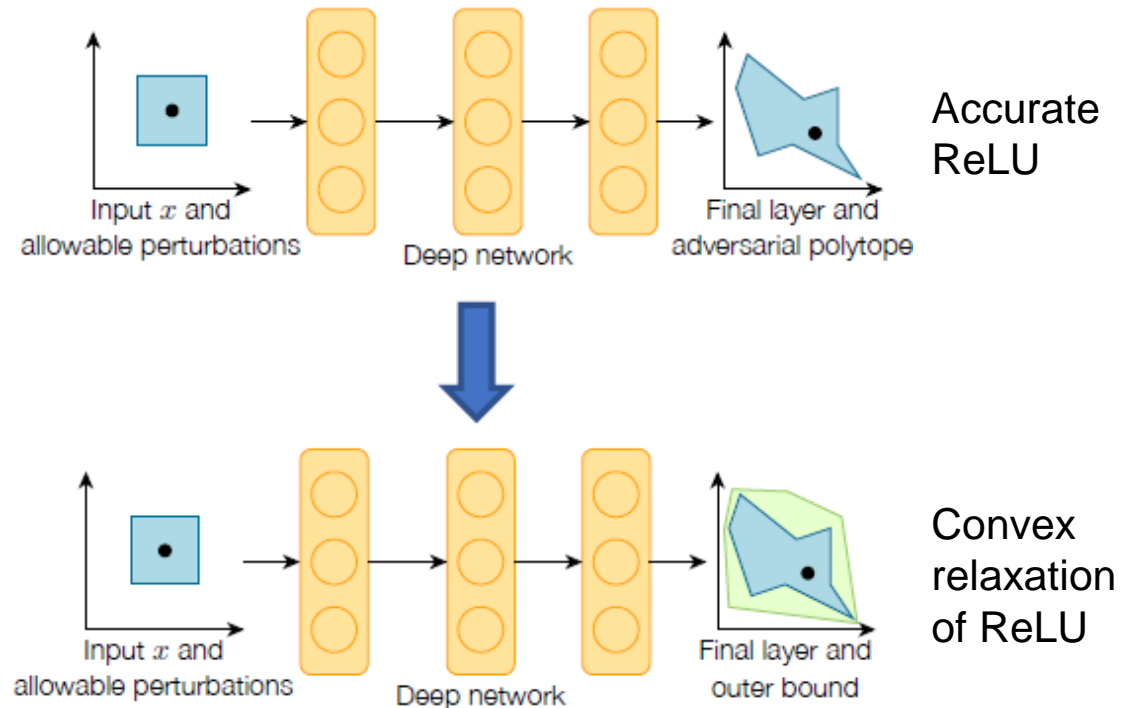
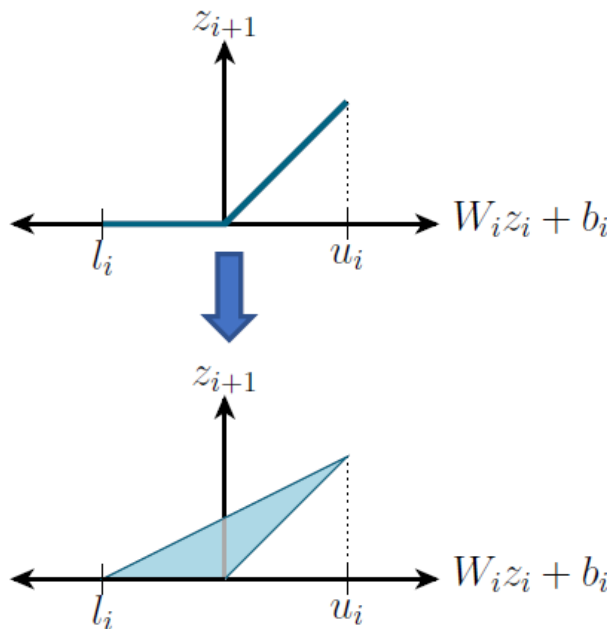
# Approach #3

- To solve  $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$ :
- 1. Constructing adversarial examples via local search (
  - Lower bound on objective
- 2. Formal verification via combinatorial optimization
  - Exactly solve objective
- 3. Formal verification via convex relaxation
  - Upper bound on objective



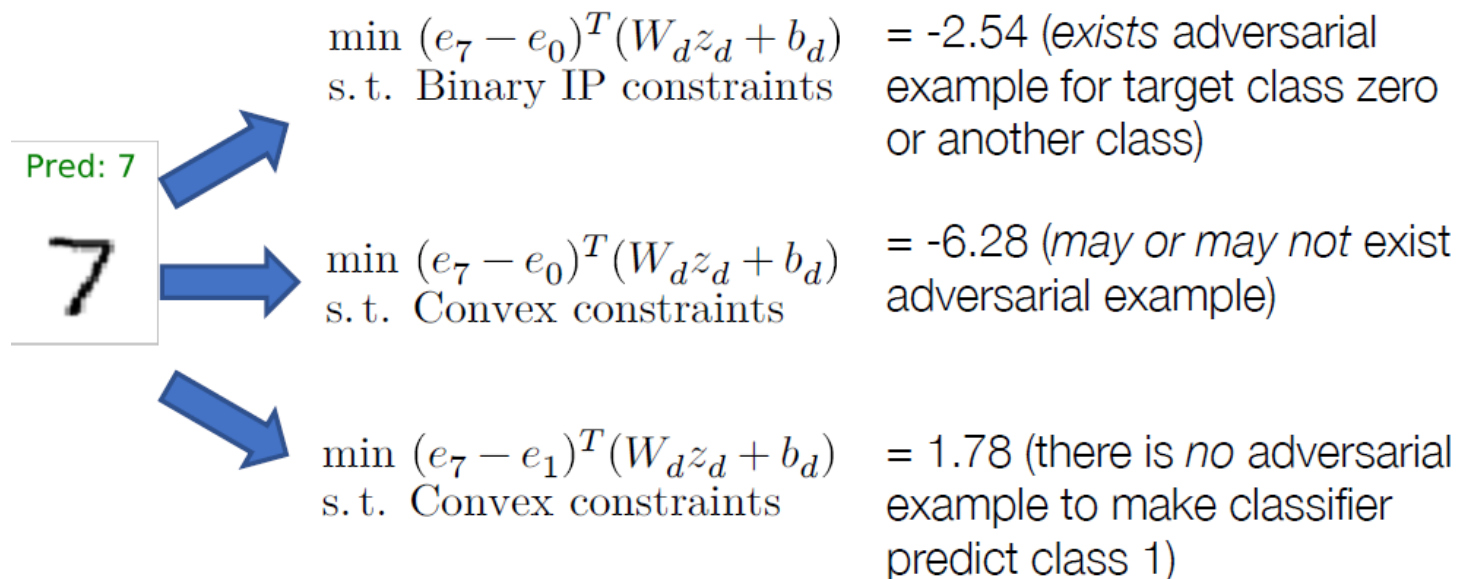
# Convex Relaxation

- For a ReLU-based NN, solving the ILP with  $v_i \in \{0,1\}^{|v_i|}$  (the ReLU can be either off or on) is too computationally expensive
- Convex relaxation: replace the ReLU constraints with their convex hull  $0 \leq v_i \leq 1$  (the ReLU can be partially off and partially on). Then optimization problem becomes a Linear Program (instead of ILP)



# Convex Relaxation as Conservative Approximation

- Convex relaxation provides a strict lower bound on the ILP objective (because feasible set is larger)  $\text{Obj}(\text{LP}) \leq \text{Obj}(\text{ILP})$  ([refer to p. 17](#))
- If  $\text{Objective}(\text{LP})$  is still positive for all target classes, the relaxation gives a verifiable proof that no adversarial example exists
- If  $\text{Objective}(\text{LP})$  may be negative for some target class, we can say nothing about existence of adversarial examples. Solving the relaxed problem does not actually produce a true adversarial example anymore. The relaxation may be able to construct an example with a negative objective, even though no actual example could achieve this



# Interval-based Bounds

- We can formulate optimization considering **only bound constraints**. We propagate interval bounds to the second-to-last layer  $z_d$ , and then solve the minimization problem at the last linear layer:
  - $\min_{z_d} c^T (W_d z_d + b_d) = (c^T W_d) z_d + c^T b_d$  s.t.
  - $l \leq z_d \leq u$
- Since the minimization problem is to find the lower bound, the solution is to choose  $(z_d)_j = l_j$  if  $(c^T W_d)_j > 0$ , and  $(z_d)_j = u_j$  otherwise. This results in the analytical solution for the optimal objective:
  - $\min_{z_d} c^T (W_d z_d + b_d) = [c^T W_d]_+ l + [c^T W_d]_- u + c^T b_d$
- These bounds are even more pessimistic than bounds obtained by convex relaxation. There is no single input (even with relaxed activations) that creates these bounds on the logit differences: each individual activation assumes that the previous layer could take on a separate set of values to minimize or maximize just that one activation

# Outline

- Introduction
- Adversarial examples and verification
  - Constructing adversarial examples via local search
    - Physically-realizable attacks
  - Formal verification via combinatorial optimization
  - Formal verification via convex relaxations
- Training adversarially robust models
- Adversarial robustness beyond security

# Adversarial Training w. Outer Minimization

Inner maximization:



$$\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$$



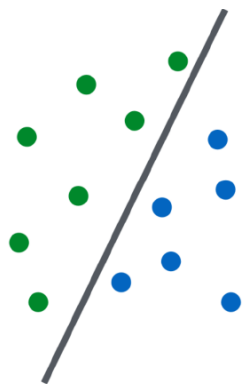
Outer minimization:

$$\min_{\theta} \sum_{x, y \in S} \max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$$

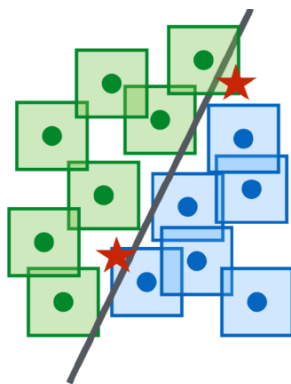
1. Local search (lower bound on objective)
2. Combinatorial optimization (exactly solve objective)
3. Convex relaxation (upper bound on objective)

1. Adversarial training

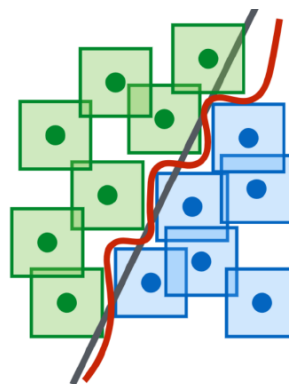
3. Provably robust training (omitted)



1) Simple decision boundary



2) Adversarial examples ★ within  $L_{\infty}$  balls



3) Training on ★ requires a complex decision boundary

Higher network capacity enables more complex decision boundary and more robust classification



# Danskin's Theorem

- How to compute the gradient of the objective with the max term inside?
- Danskin's Theorem:
$$\nabla_{\theta} \max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta) = \nabla_{\theta} \text{Loss}(x + \delta^*, y; \theta)$$
- where  $\delta^* = \max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$
- It means we can optimize through the max operator by finding the  $\delta^*$  that maximizes the loss function, then taking gradient at the point  $x + \delta^*$
- It only applies when max is performed exactly

# Adversarial Training [Goodfellow et al., 2014]

Repeat:

1. Select minibatch  $B$ , initialize gradient vector  $g := 0$

2. For each  $(x, y)$  in  $B$ :

- a. Find an attack perturbation  $\delta^*$  by (approximately) optimizing

$$\delta^* = \operatorname{argmax}_{\|\delta\| \leq \epsilon} \ell(h_\theta(x + \delta), y)$$

- b. Add gradient at  $\delta^*$

$$g := g + \nabla_\theta \ell(h_\theta(x + \delta^*), y)$$

3. Update parameters  $\theta$

$$\theta := \theta - \frac{\alpha}{|B|} g$$

- In theory, Danskin's theorem only applies to the case where we are able to compute the maximum exactly. In practice, the quality of the robust gradient descent procedure is tied directly to how well we are able to perform the inner maximization. In other words, the key aspect is incorporate a strong attack into the inner maximization procedure.

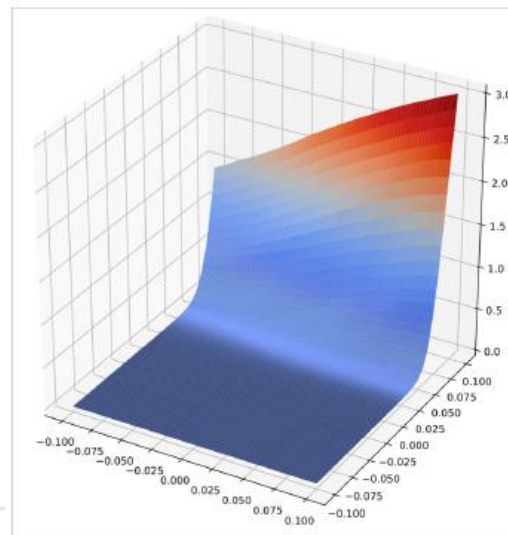
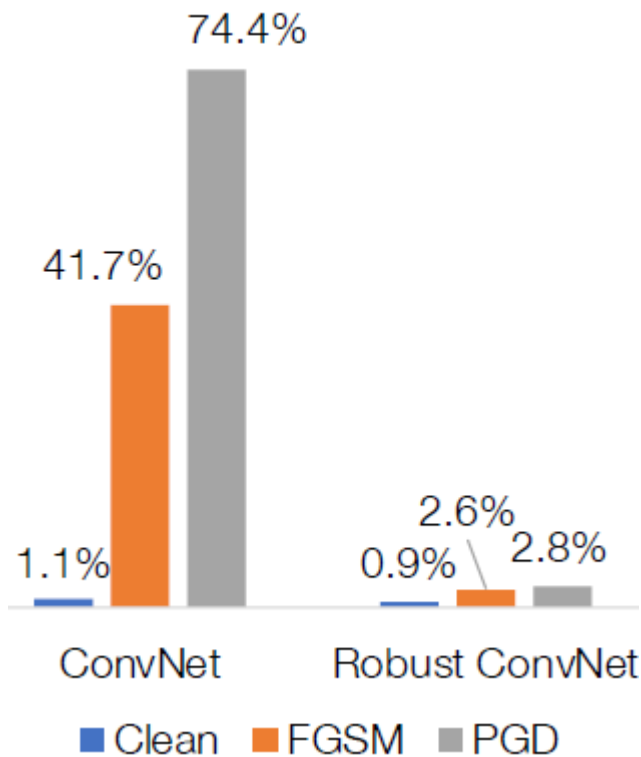
# Evaluating Robust Models

- Our model looks good, but we also need to evaluate against different attacks, PGD attacks run for longer, with random restarts, etc
- Note: it is not very informative to evaluate against a different type of attack, e.g. evaluate  $l_\infty$  robust model against  $l_1$  or  $l_2$  attacks

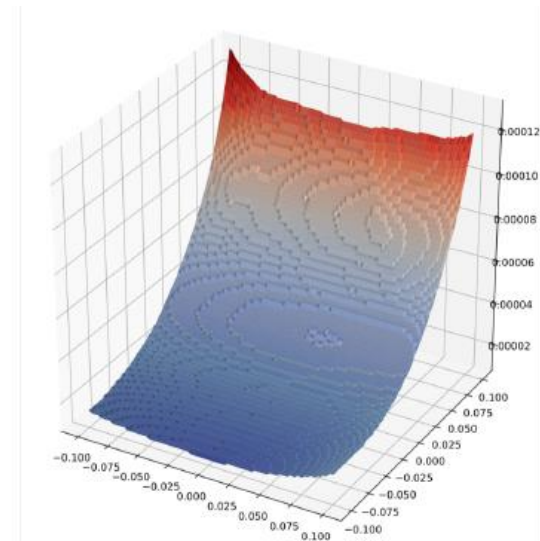
# What Makes the Models Robust?

- The robust model has a smoother loss surface, making it more difficult for an attacker to change the class label with small gradient steps

Test Error,  $\epsilon=0.1$



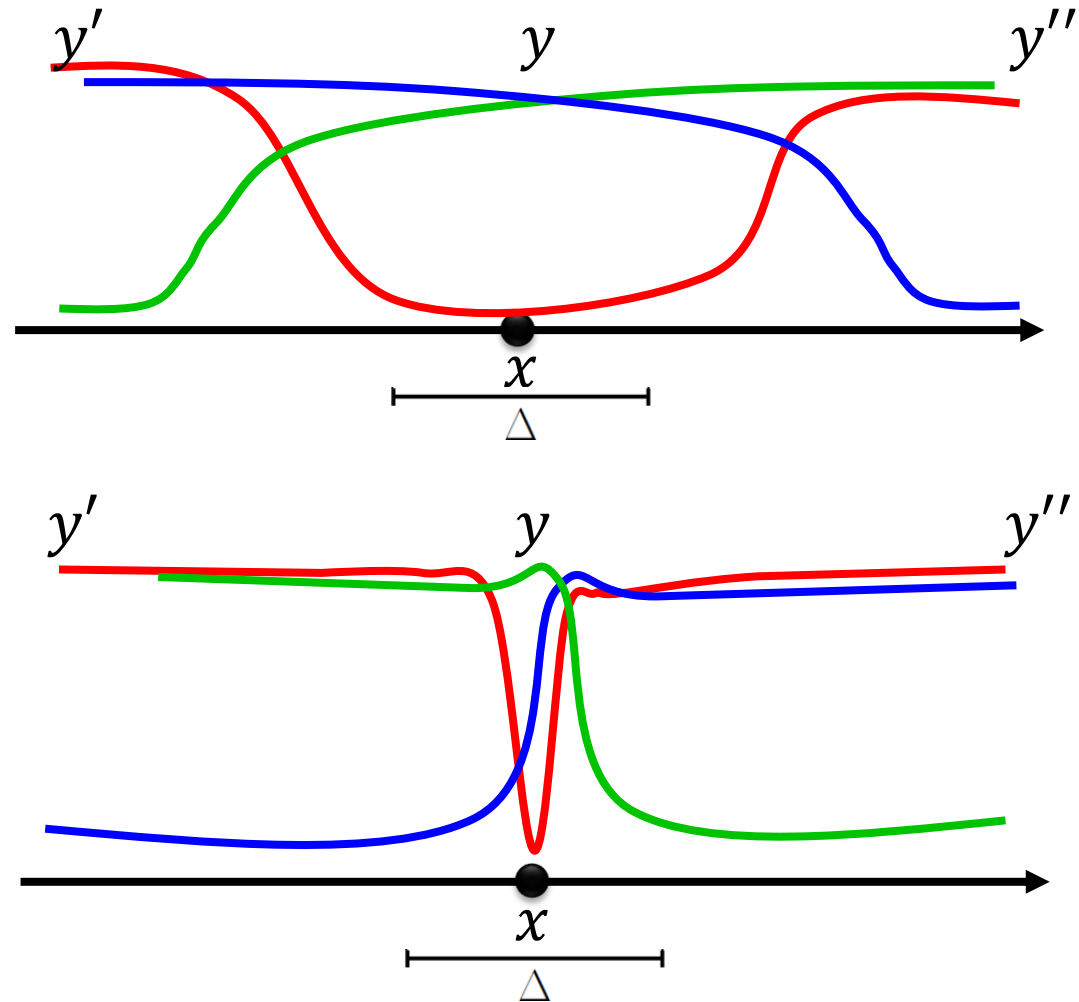
Loss surface:  
standard training



Loss surface:  
robust training

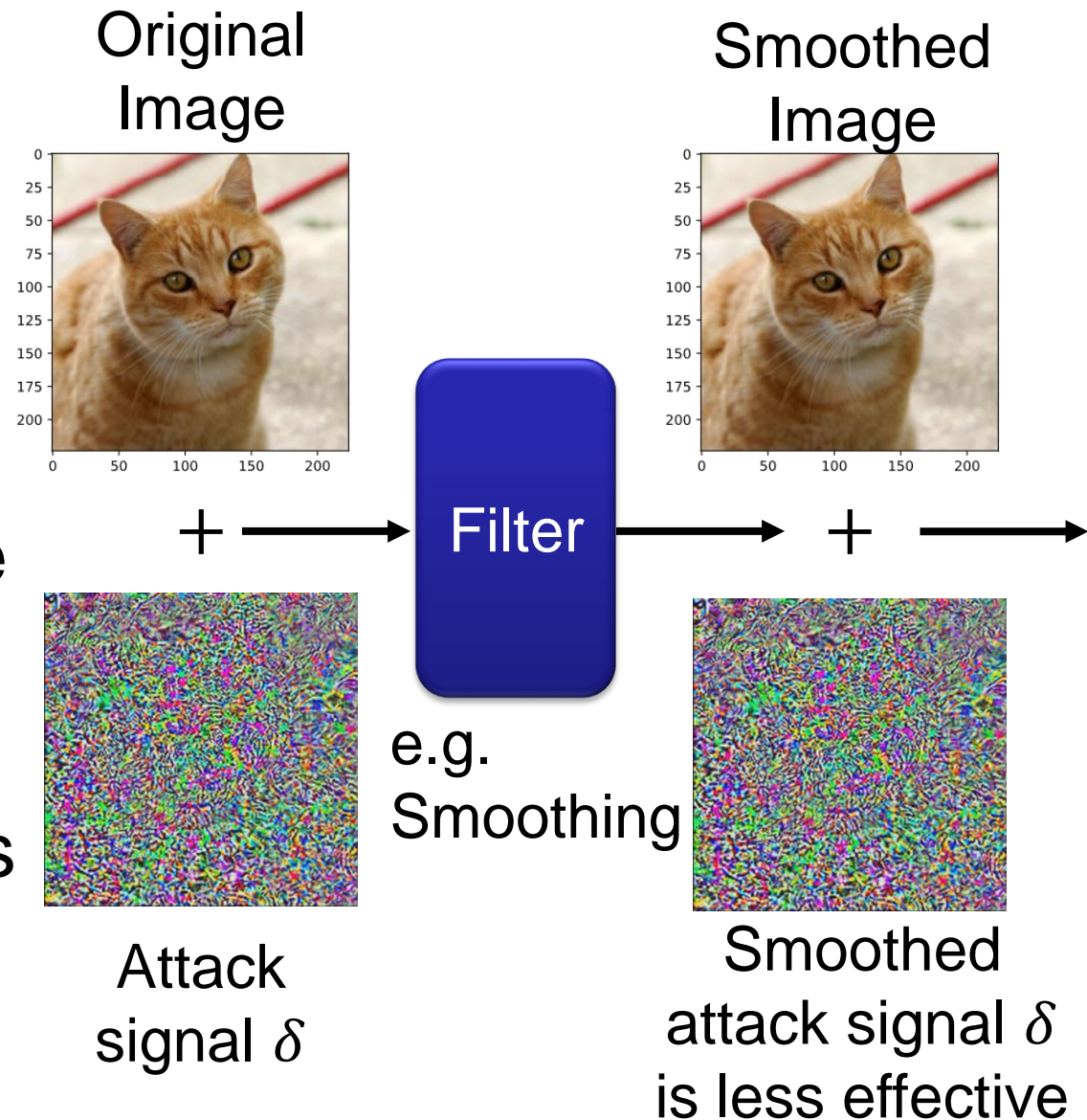
# Loss Surfaces Examples

- Upper right fig shows a smooth loss surface with small gradients near the correct label and large distances to other labels, which makes attacks more difficult
- Lower right fig shows a less smooth loss surface and small distances to other labels, which makes attacks easier
- You can also think of them as 2 different directions on the same loss surface, and the attacker's goal is to find the optimal direction to change input  $x$  (e.g., by gradient ascent with FGSM or PGD)



# Smoothing Filter on the Input as Defense

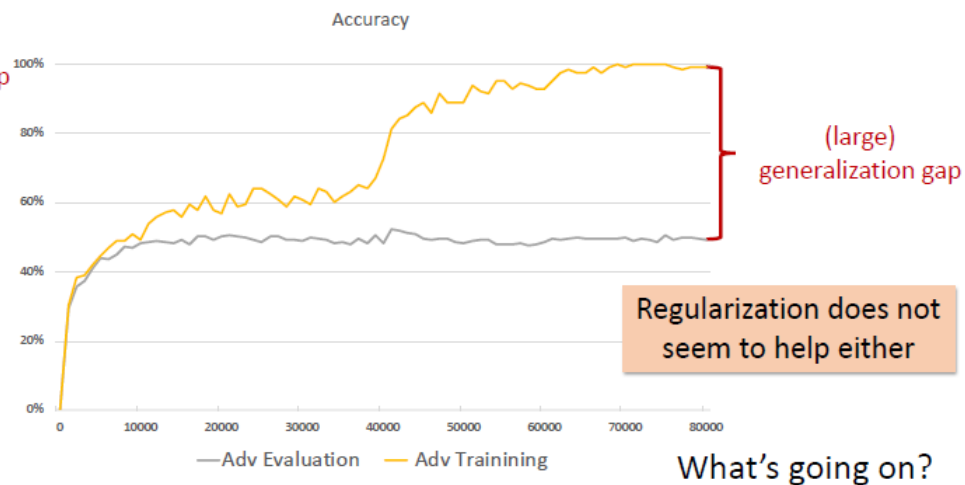
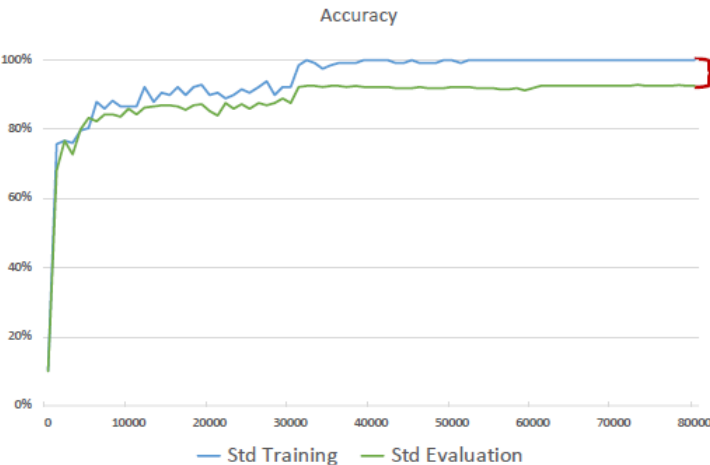
- The filter helps make the loss surface smoother, which makes attacks more difficult
- Not a very effective defense. Furthermore, if attacker knows the filter, the defense is no longer effective



# Outline

- Introduction
- Adversarial examples and verification
  - Constructing adversarial examples via local search
    - Physically-realizable attacks
  - Formal verification via combinatorial optimization
  - Formal verification via convex relaxations
- Training adversarially robust models
- Adversarial robustness beyond security

# Adv. Robust Generalization Needs More Data to Avoid Overfitting

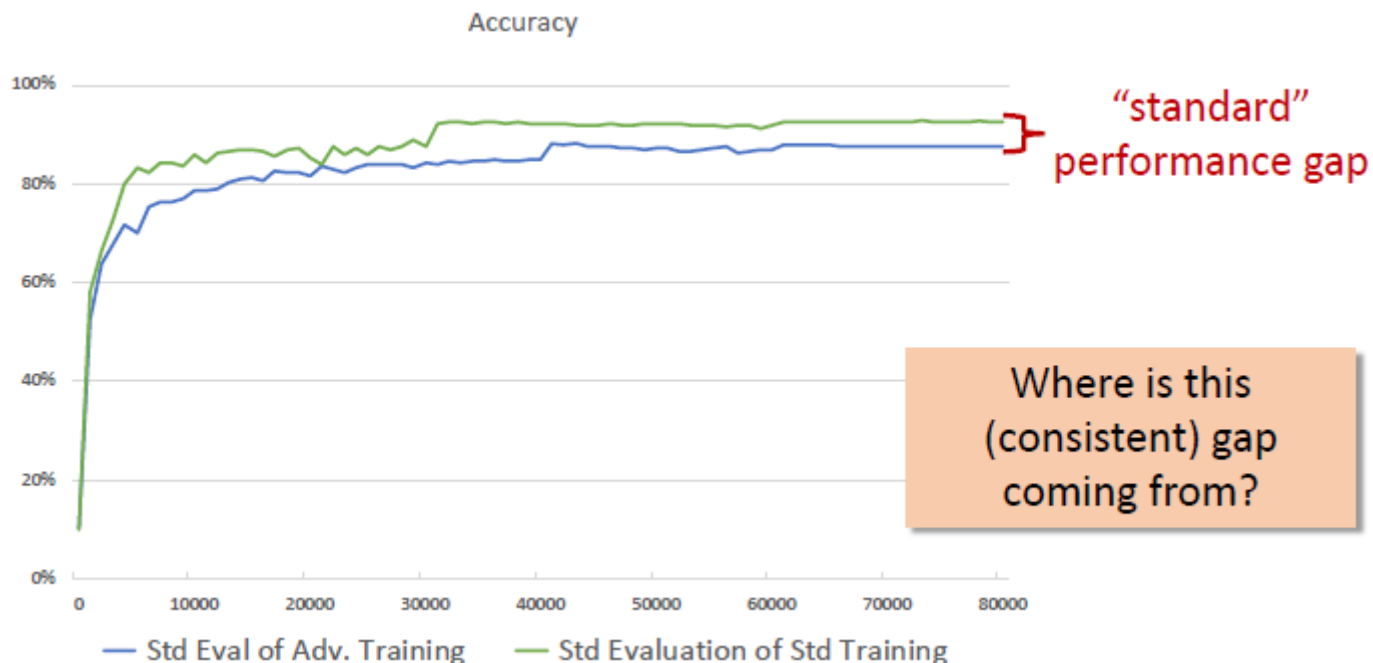


- Theorem [Schmidt Santurkar Tsipras Talwar M 2018]: Sample complexity of adv. robust generalization can be significantly larger than that of “standard” generalization
- Specifically: There exists a  $d$ -dimensional distribution  $D$  s.t.:
  - A single sample is enough to get an accurate classifier ( $P[\text{correct}] > 0.99$ )
  - But need  $\Omega(\sqrt{d})$  samples for better-than-chance robust classifier



# Does Being Robust Help “Standard” Generalization?

- Data augmentation: An effective technique to improve “standard” generalization
- Adversarial training = An “ultimate” version of data augmentation
  - (since we train on the “most confusing” version of the training set)
- Does adversarial training always improve “standard” generalization?
  - No. Adversarial training typically results in lower performance when evaluated on standard input



# Does Being Robust Help “Standard” Generalization?

- Theorem [Tsipras et al. 2018]: No “free lunch”: trade-off between accuracy and robustness
- Standard training tries to use all features to maximize model accuracy, incl. non-robust features (e.g., outdoor scenery for class label “dog”), and the resulting model is vulnerable to adversarial examples that manipulate the non-robust features w. minor perturbations
- Adversarial training tries to use only robust features (e.g., human-recognizable dog features like ears, tail, black nose...for class label “dog”) to increase model robustness at the cost of reduced accuracy, since larger and more noticeable perturbations are needed to change the robust features

**Robust features** correlated with labels even when an adversary is present      **Non-robust features** correlated with labels on average but manipulated by an adversary



# Adversarial Robustness is Not Free

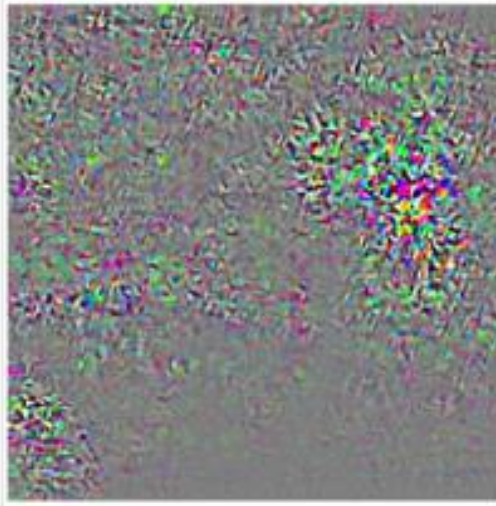
- Optimization during training more difficult and models need to be larger
- More training data might be required
- Might need to lose on “standard” measures of performance like precision, recall, accuracy

# But There Are (Unexpected?) Benefits Too

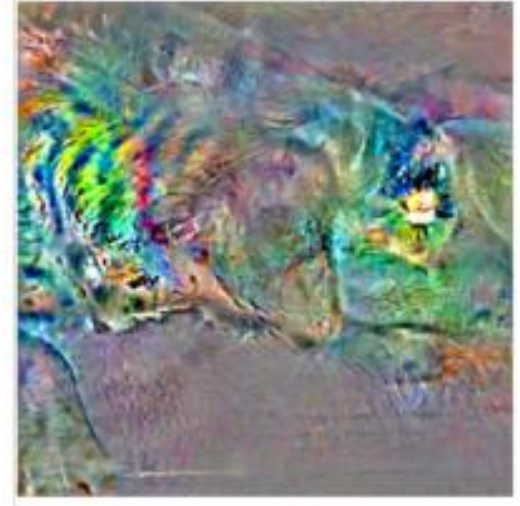
- Models become more semantically meaningful
- Fig shows heatmaps highlighting the pixels that maximally activate the output neuron with the predicted label



Input



Gradient of  
standard model



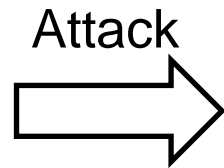
Gradient of  
**adv. robust** model

# But There Are (Unexpected?) Benefits Too

- Lower left: an input image, correctly labeled as “Primate”
- Lower middle: an adversarial input that fools a standard model into misclassification as “Bird”
  - Minor perturbation undetectable by humans
- Lower right: an adversarial input that fools an adv. robust model into misclassification as “Bird”
  - Semantically-meaningful perturbation detectable by humans



Standard model



Standard model



Adv. robust model

# Conclusions

- Algorithms: Faster robust training + verification, smaller models, new architectures?
- Theory: (Better) adv. robust generalization bounds, new regularization techniques
- Data: New datasets and more comprehensive set of perturbations (robust-ml.org)
  - Major need: Embracing more of a worst-case mindset
- Open-source tools:
  - IBM Adversarial Robustness Toolbox (ART) <https://github.com/Trusted-AI/adversarial-robustness-toolbox>
  - CleverHans <https://github.com/cleverhans-lab/cleverhans>
  - Foolbox <https://github.com/bethgelab/foolbox>