L4.2 Adversarial Robustness

Z. Gu 2021

"panda"
57.7% confidence

$+ \epsilon$

$=$

"gibbon"
99.3% confidence

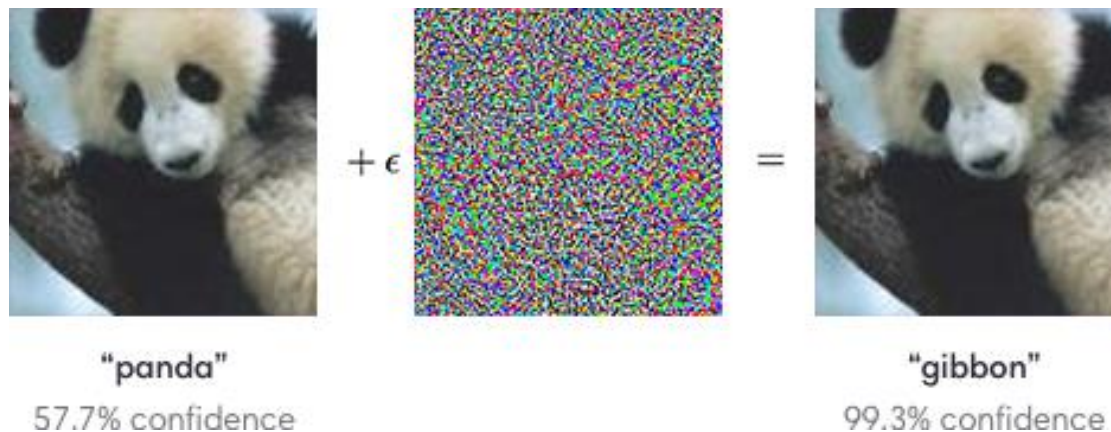Based on ICML 2018 tutorial https://adversarial-ml-tutorial.org

1

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 training
  - Robust optimization with convex relaxations
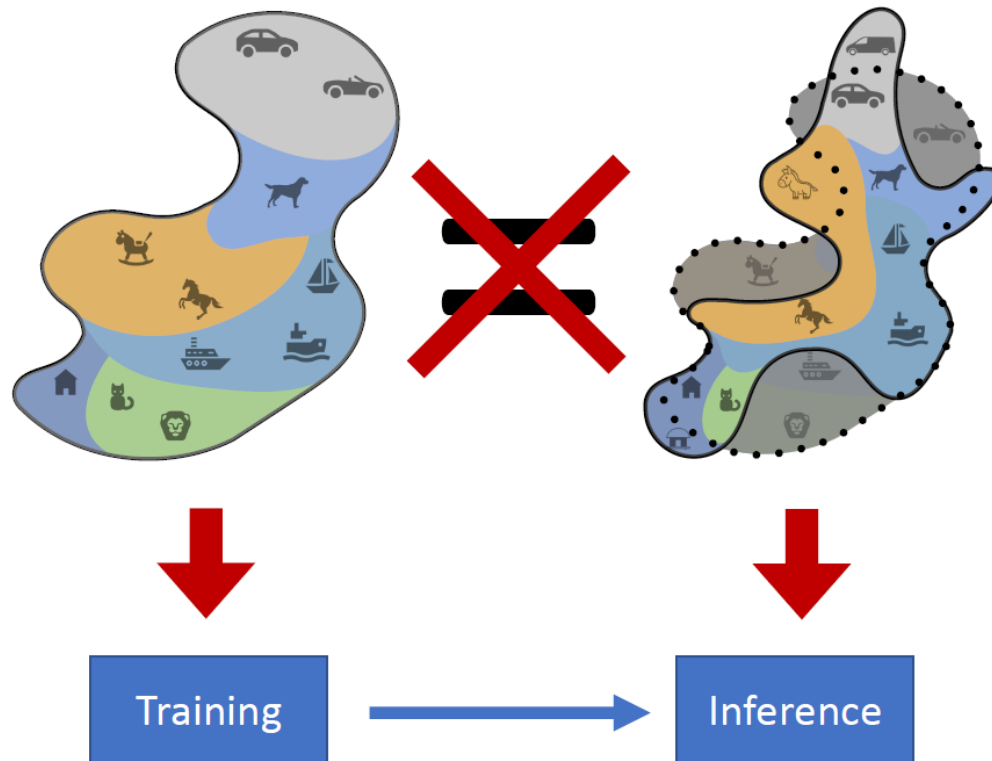- Adversarial robustness beyond security

ImageNet: An ML Home Run

ILSVRC top-5 Error on ImageNet
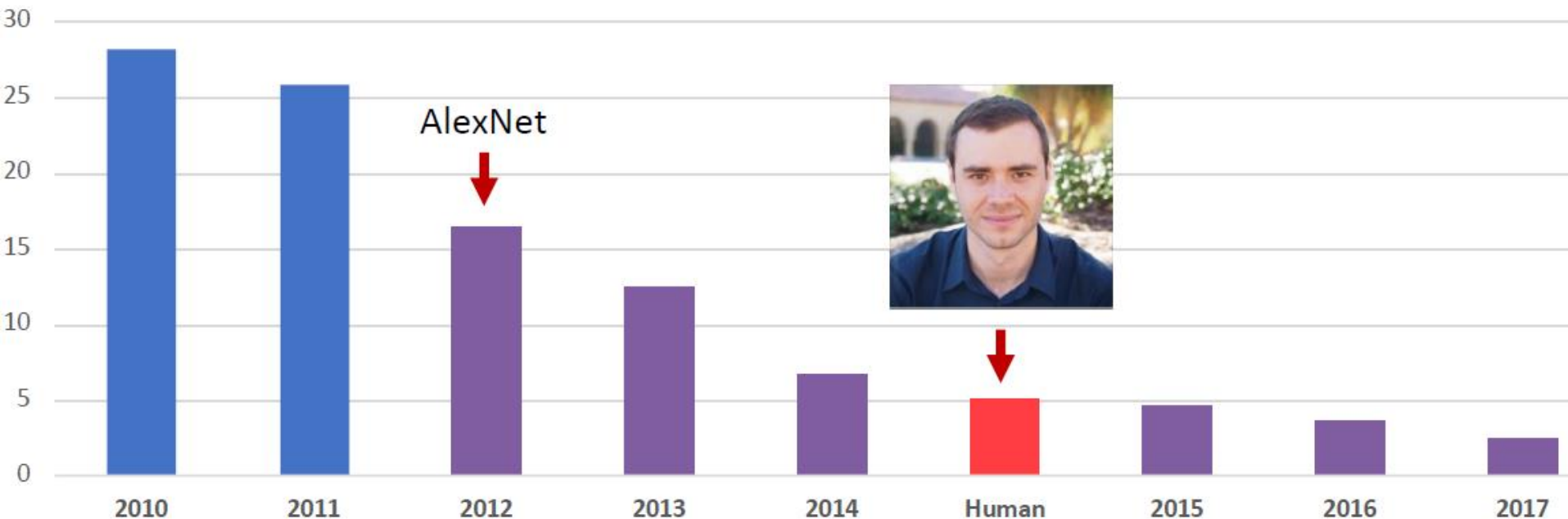
A Limitation of the (Supervised) ML Framework

- Measure of performance: Fraction of mistakes during testing

- Distribution Shift: In reality, the data distributions we use ML inference on are NOT the same as the ones we train it on

Training → Inference

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.

$$+ .007 \times$$

$$=$$

$$\boldsymbol{x}$$
"panda"
57.7% confidence

$$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$$
"nematode"
8.2% confidence

$$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$$
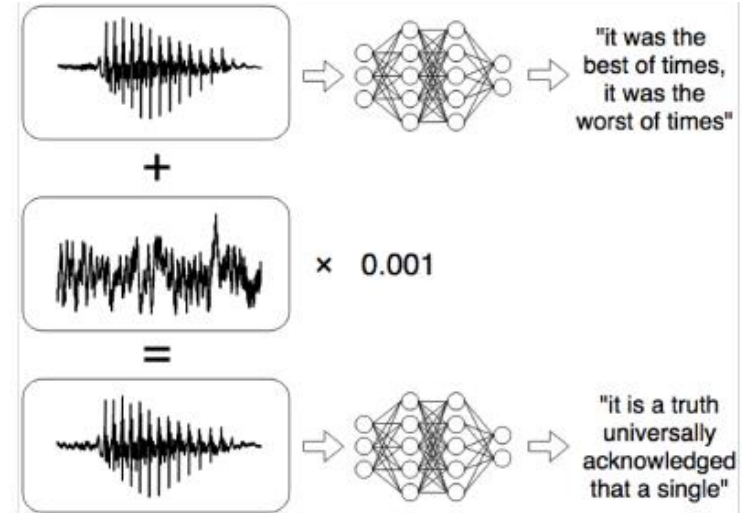"gibbon"
99.3 % confidence

Why Is This Brittleness of ML a Problem?

→ Security

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

"it was the best of times, it was the worst of times"

+

× 0.001

=

"it is a truth universally acknowledged that a single"

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

Training Time Attack vs. Inference Time Attack

Training ⟶ Inference

⚠ Data poisoning
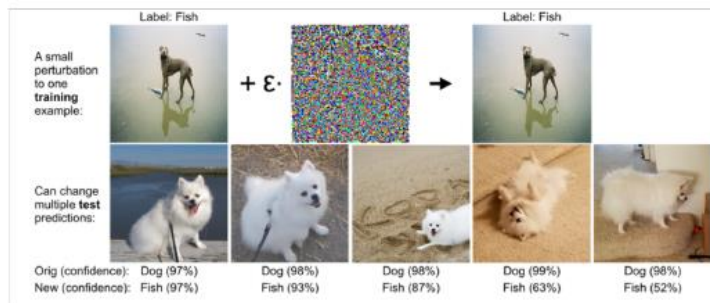
⚠ Adversarial Examples
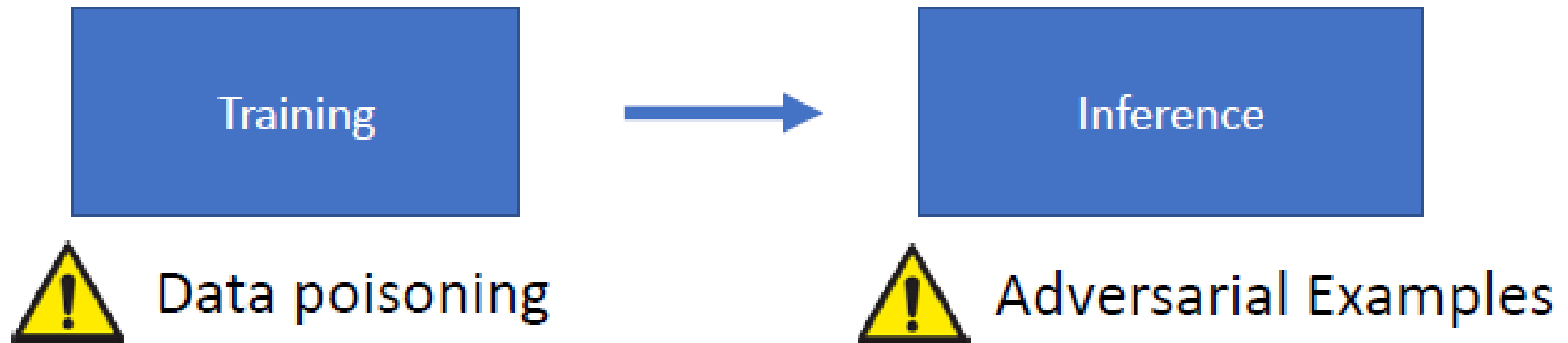
Data Poisoning

- 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 a linear model

Data poisoning for deep learning
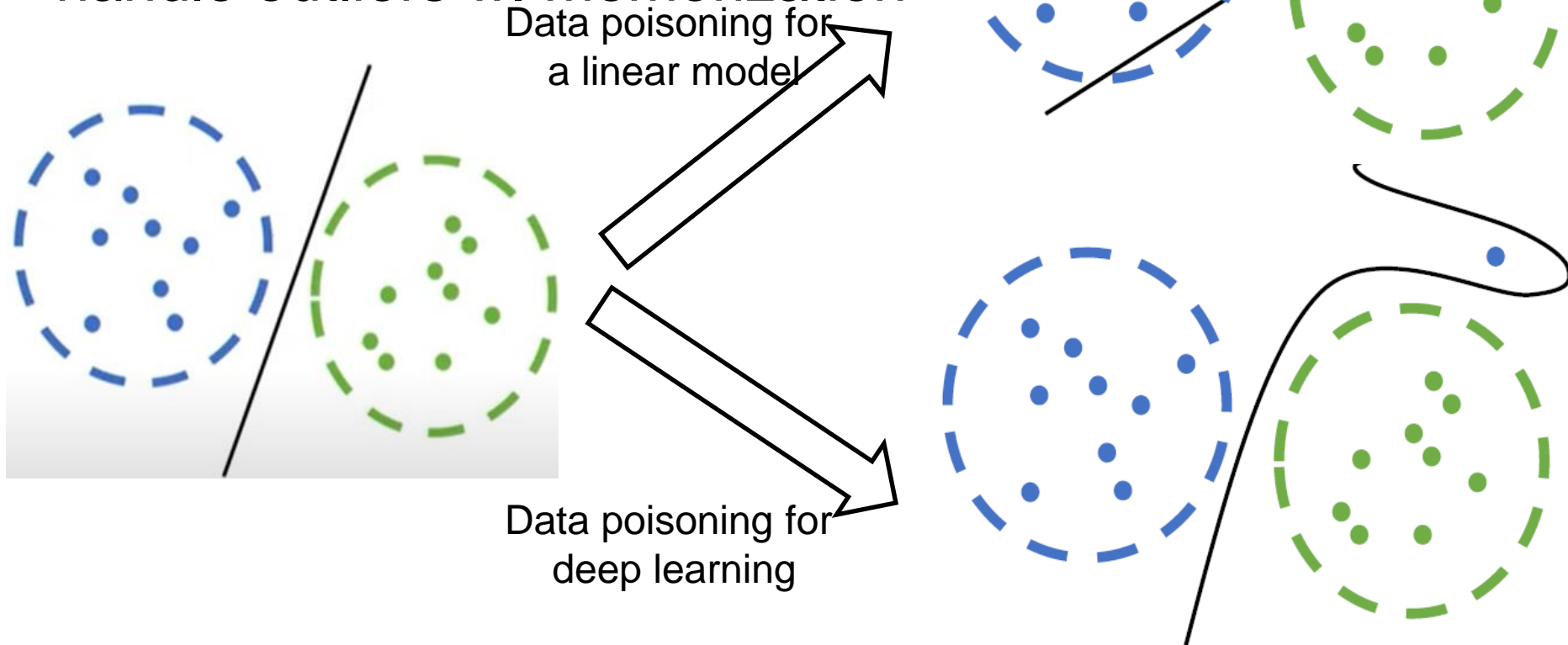
Data Poisoning for Deep Learning

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

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

"van" "dog"

[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 training
  - Robust optimization with convex relaxations
- 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 δ (or ensuring one does not exist)

  – Outer minimization problem: training a robust classifier in the presence of adversarial examples

Input Perturbations

- Which input perturbations δ are allowed?

- Examples: δ that is small wrt

  - l_p norm (we focus on it in this lecture)

  - Rotation and/or translation

  - VGG feature perturbation
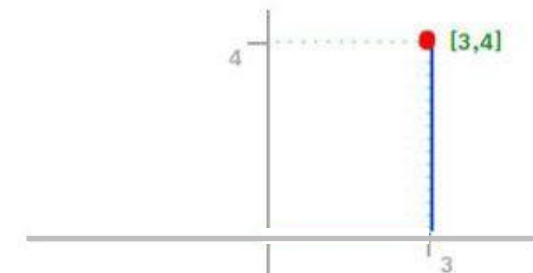
  - (add the perturbation you need here)

Vector Norms

- l_p norm of a k-dimensional vector $x \in \mathbb{R}^k$ is a scalar defined as $\|x\|_p = \left(\sum_{i=1}^{k}|x_i|^p\right)^{1/p}$. 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_∞ norm: $\|x\|_\infty = \max_i x_i$
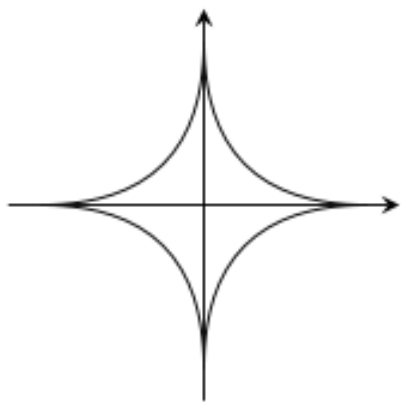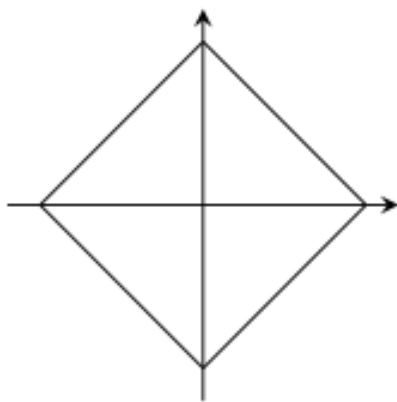
  - $= \max_i(3,4) = 4$
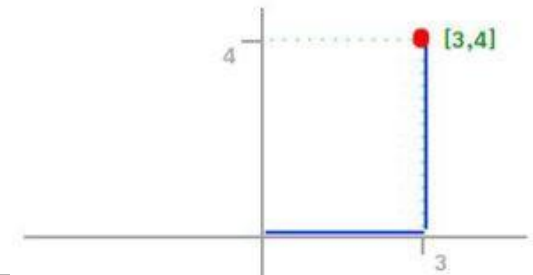
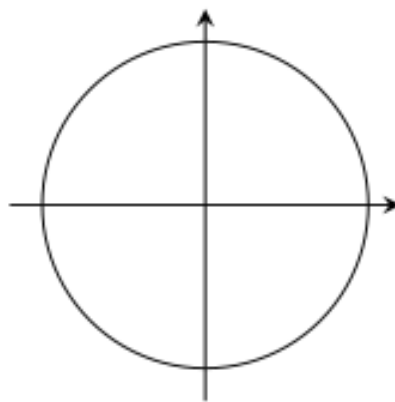l_1 norm
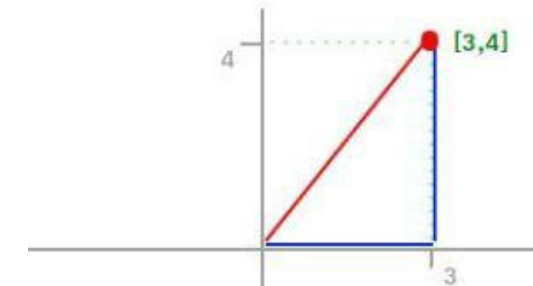
l_2 norm

l_∞ 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 ϵ: $B_p = \{x \in \mathbb{R}^k \,|\, \|x\|_p \leq \epsilon\}$

- l_2 norm ball $\|x\|_2 \leq \epsilon$: a circle with radius ϵ centered at origin

- l_∞ norm ball $\|x\|_\infty \leq \epsilon$: a square with edge length 2ϵ centered at origin
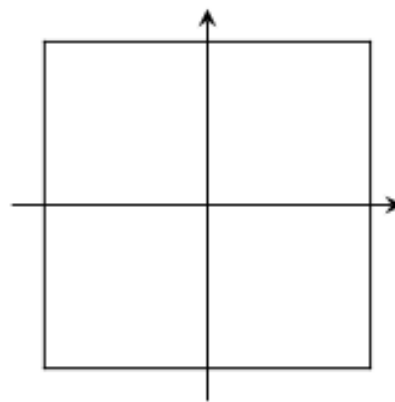

$p = \frac{1}{2}$ $p = 1$ $p = 2$ $p = \infty$

l_2 vs. l_∞

- 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_∞ distances:
  - $\|\delta^1\|_\infty = \max(7,7) = 7 , \|\delta^2\|_\infty = \max(10,0) = 10$

- l_∞ 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_∞ distance

small l_∞

Change every pixel a little bit

same l_2

Change one pixel much

large l_∞
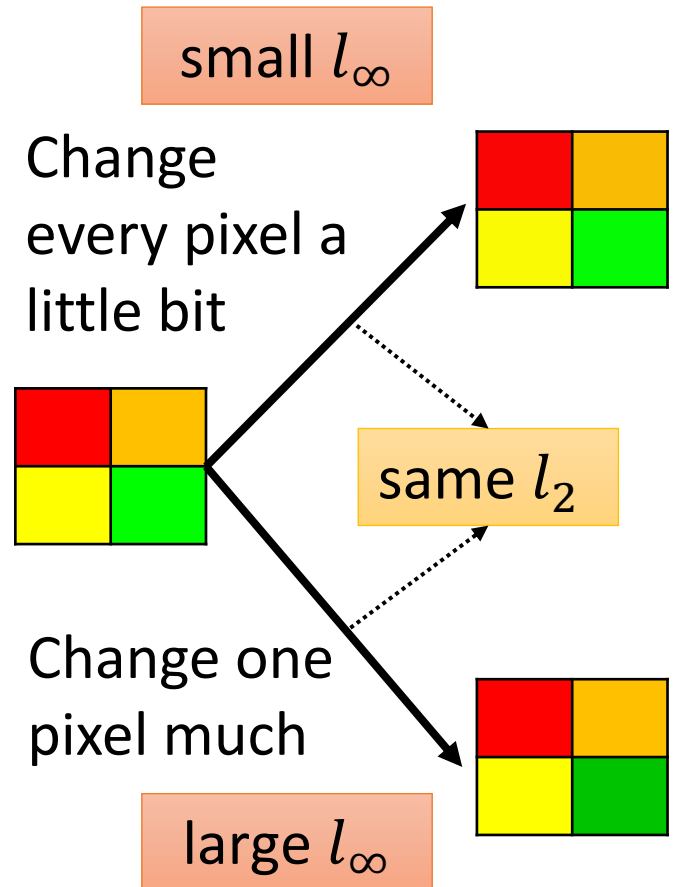
The Maximization Problem for Finding Adversarial Examples

- $\max\limits_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$
  - $\text{Loss}()$ 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 Δ
 - 2) the optimization procedure used to perform the maximization

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

$\delta = 0$

Δ
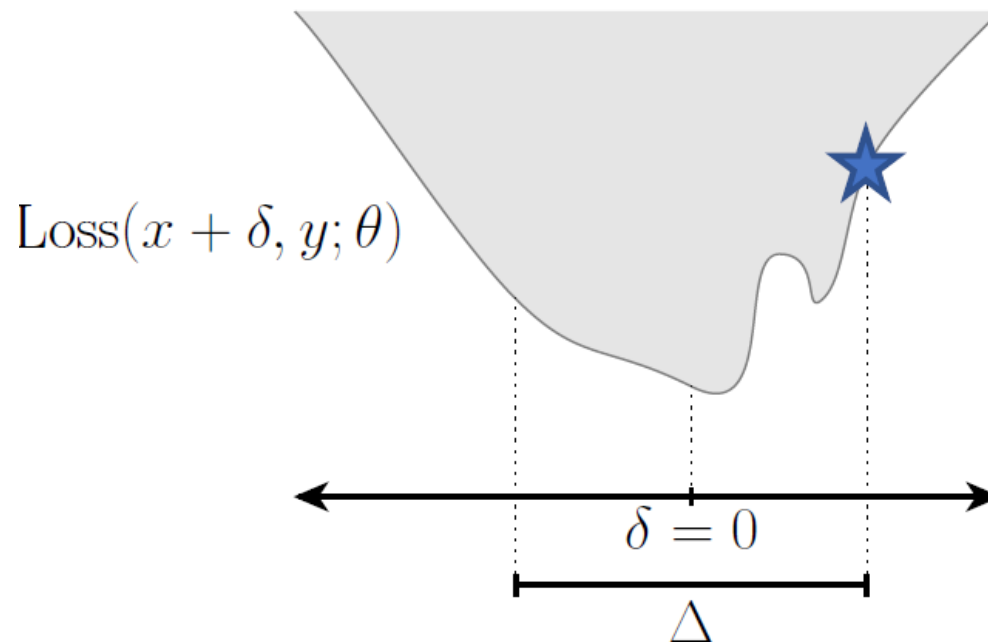
Three Approaches

- To solve $\max\limits_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$:
- 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
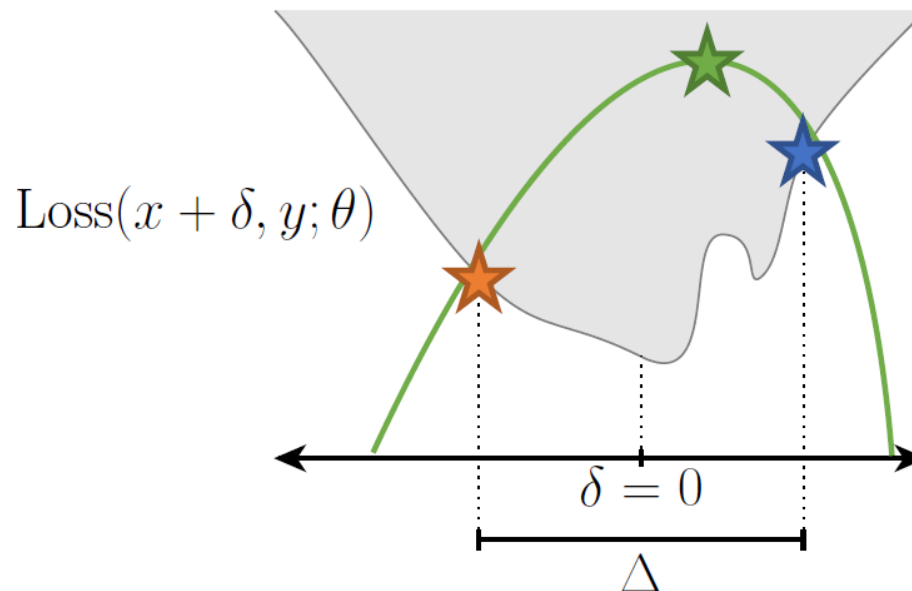
Approach #1

- To solve $\max\limits_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$:
- 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
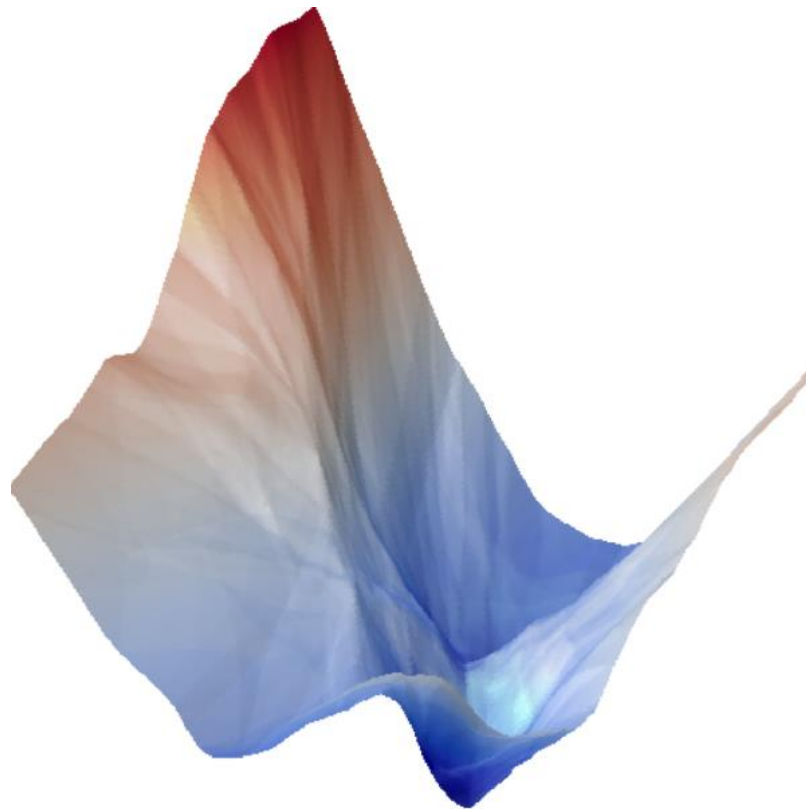 - Upper bound on objective

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

$\delta = 0$

Δ

19
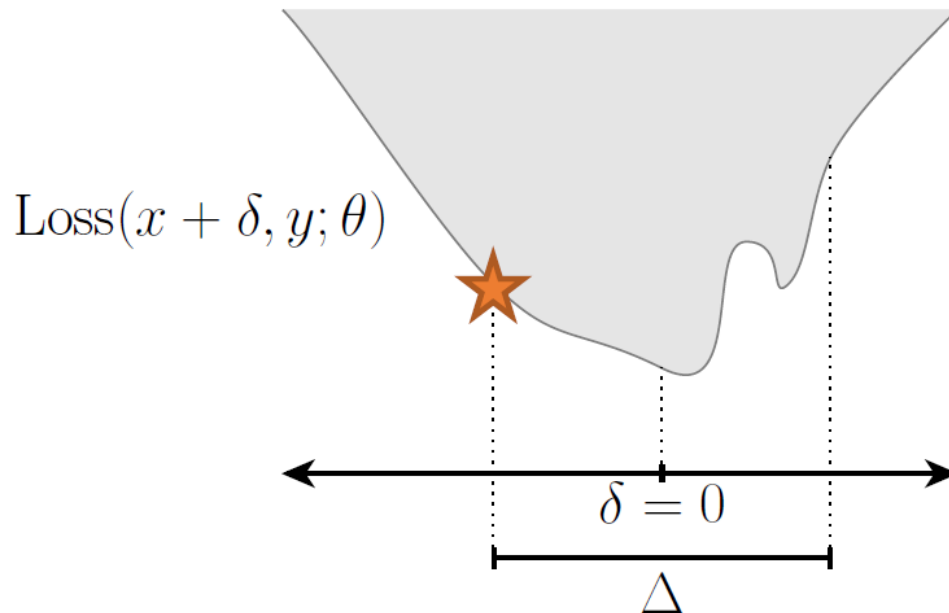
Local Search

- 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 $\theta \leftarrow \theta - \alpha \nabla_{\theta} \text{Loss}(x, y; \theta)$

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

  – Update NN params θ by following the gradient downhill, in order to decrease $\text{Loss}(x, y; \theta)$. ($\alpha$ is the Learning Rate)
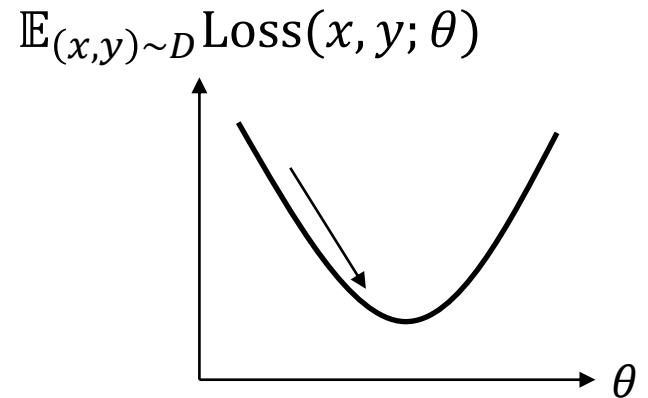
- To solve $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$ for adversarial input generation: gradient ascent $\delta \leftarrow \delta + \alpha \nabla_{\delta} \text{Loss}(x + \delta, y; \theta)$

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

  – Update input δ by following the gradient uphill, in order to increase $\text{Loss}(x + \delta, y; \theta)$, while ensuring $\delta \in \Delta$
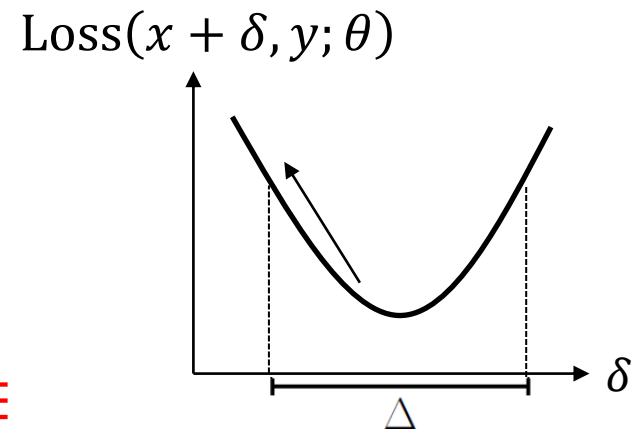
Aside: Vector Derivative

- In general, x, δ 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:

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

Projected Gradient Descent

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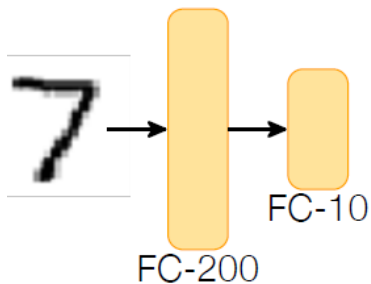
Fast Gradient Sign Method (FGSM)

- Consider l_∞ norm bound $\Delta = \{\delta : \|\delta\|_\infty \leq \epsilon\}$. Projection onto this norm ball by clipping values of δ to lie within the range $[-\epsilon, \epsilon]$: $\mathcal{P}_\Delta(\delta) \coloneqq \text{Clip}(\delta, [-\epsilon, \epsilon])$

- Starting from $\delta = 0$, take a large step in the gradient direction by making the learning rate α very large. After clipping, we have: $\delta = \mathcal{P}_\Delta\big(0 + \alpha \nabla_\delta \text{Loss}(x + \delta, y; \theta)\big) = \epsilon \cdot \text{sign}\big(\nabla_\delta \text{Loss}(x + \delta, y; \theta)\big)$

- The specific values of α 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_∞ norm ball will result in the same δ at the upper right corner

Gradient $\nabla_\delta \text{Loss}$

$\mathcal{P}_\Delta$

δ

ε

$\delta = 0$

24
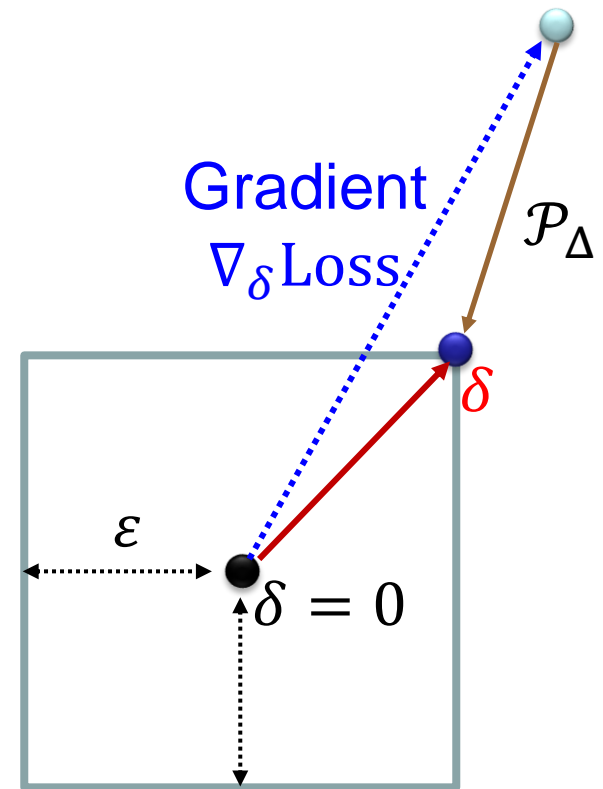
Adversarial Examples by FGSM

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

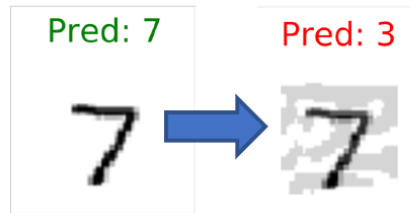# Comments on FGSM
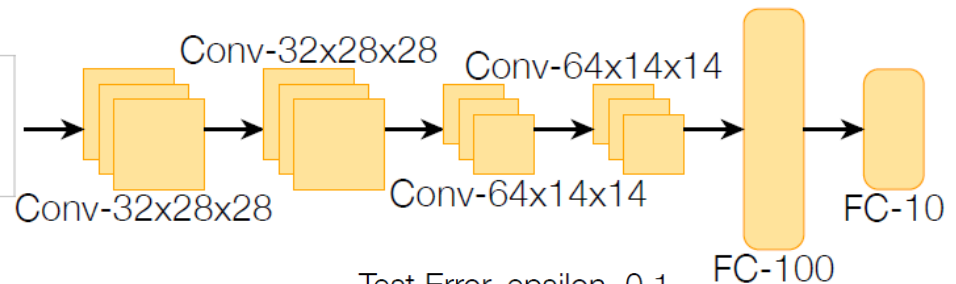
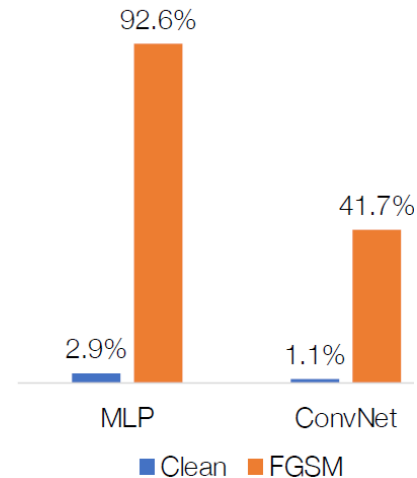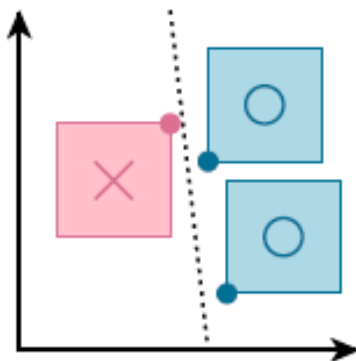- FGSM is an attack designed for l_∞ norm bound by taking a single PGD (Projected Gradient Descent) step within $\|\delta\|_\infty \leq \epsilon$, not for other norm bounds
- FGSM is the optimal attack against a linear binary classifier under the l_∞ norm bound (details omitted)

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

PGD w. Small Steps

- Instead of taking a single large step as in FGSM, PGD takes many small steps to iteratively update δ:
  - Repeat: $\delta \leftarrow \mathcal{P}_\Delta\big(\delta + \alpha\nabla_\delta\mathrm{Loss}(x + \delta, y; \theta)\big)$
  - $\mathcal{P}_\Delta$ corresponds to clipping in the case of l_∞ norm
  - Rule-of-thumb: choose α to be a small fraction of ϵ, and set the number of iterations to be a small multiple of ϵ/α
- Fig shows a sequence of gradient steps, with the last step going outside of the l_∞ ball Δ, but $\mathcal{P}_\Delta$ brings it back into Δ

$$\delta = 0$$

$$\Delta$$
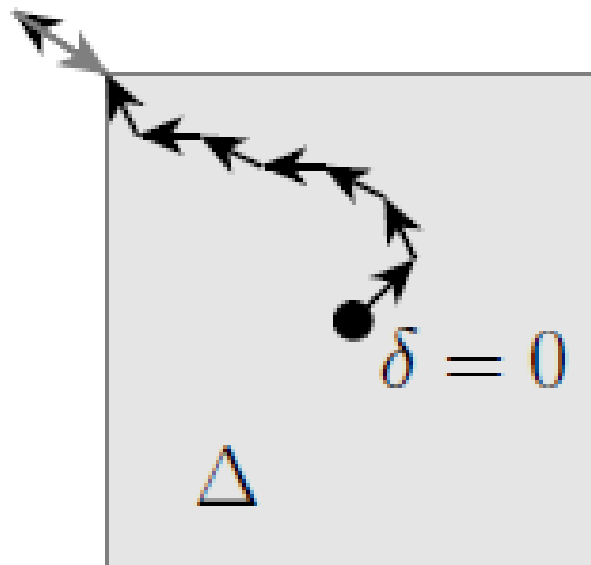
Projected Steepest Descent

- 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_∞, $\underset{\|v\| \leq \alpha}{\operatorname{argmax}} \, v^T \nabla_\delta J(\delta) = \alpha \operatorname{sign}(\nabla_\delta J(\delta))$ (similar to FGSM, except we do not make α very large)

  - For l_2, $\underset{\|v\| \leq \alpha}{\operatorname{argmax}} \, v^T \nabla_\delta J(\delta) = \alpha \dfrac{\nabla_\delta J(\delta)}{\|\nabla_\delta J(\delta)\|_2}$

- Recall vector inner product $x^T y = x \cdot y = \|x\| \|y\| \cos \theta$, where θ is the angle between x and y.

  - For l_∞, the update direction and step size are changed to point to a corner of the l_∞ 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 $\|v\|_2 \leq \alpha$
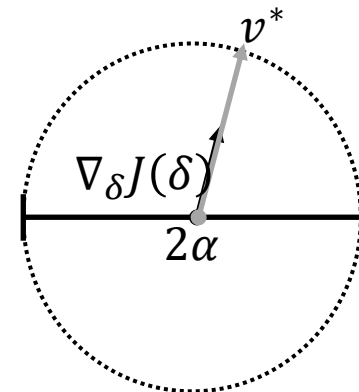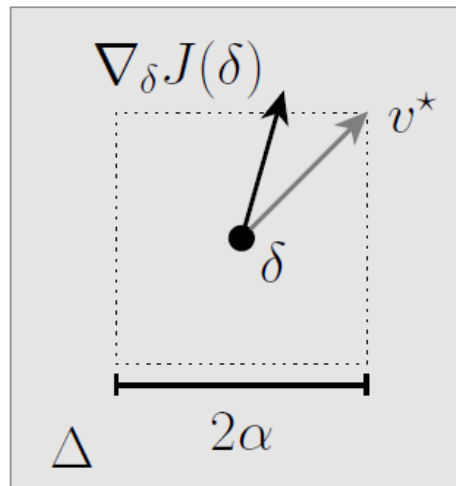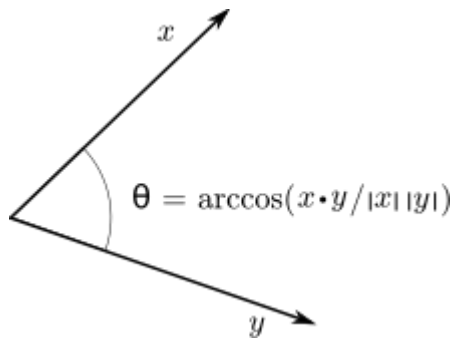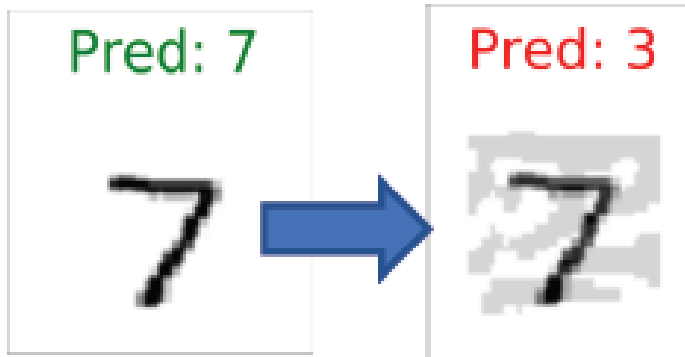

28

Illustration of PGD

ConvNet (FGSM):

Pred: 7 → Pred: 3

ConvNet (PDG)

Pred: 7 → Pred: 3

Test Error, epsilon=0.1

MLP: Clean 2.9%, FGSM 92.6%, PGD 96.4%

ConvNet: Clean 1.1%, FGSM 41.7%, PGD 74.3%

Clean ■ FGSM ■ PGD
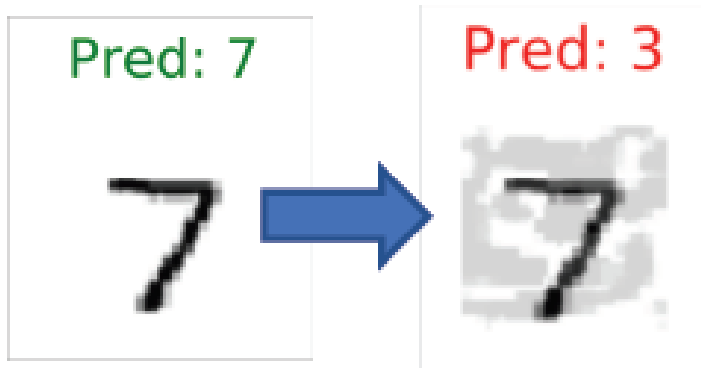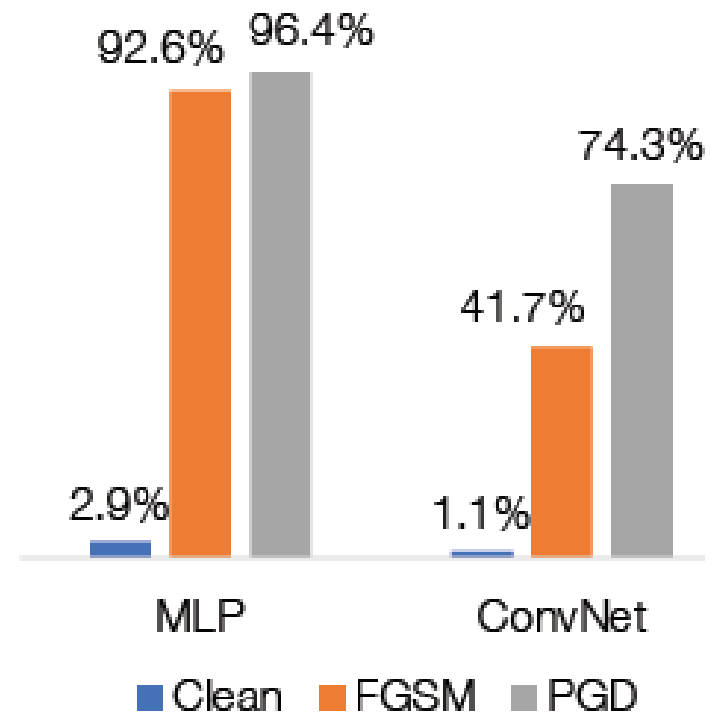
Recall: Cross-Entropy Loss for Multi-Class Classification

- The SoftMax operator $\sigma: \mathbb{R}^k \to \mathbb{R}^k$ 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:

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

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

  - $\mathrm{Loss}(h_\theta(x), y) = -\log \sigma\big(h_\theta(x)\big)_y = -$
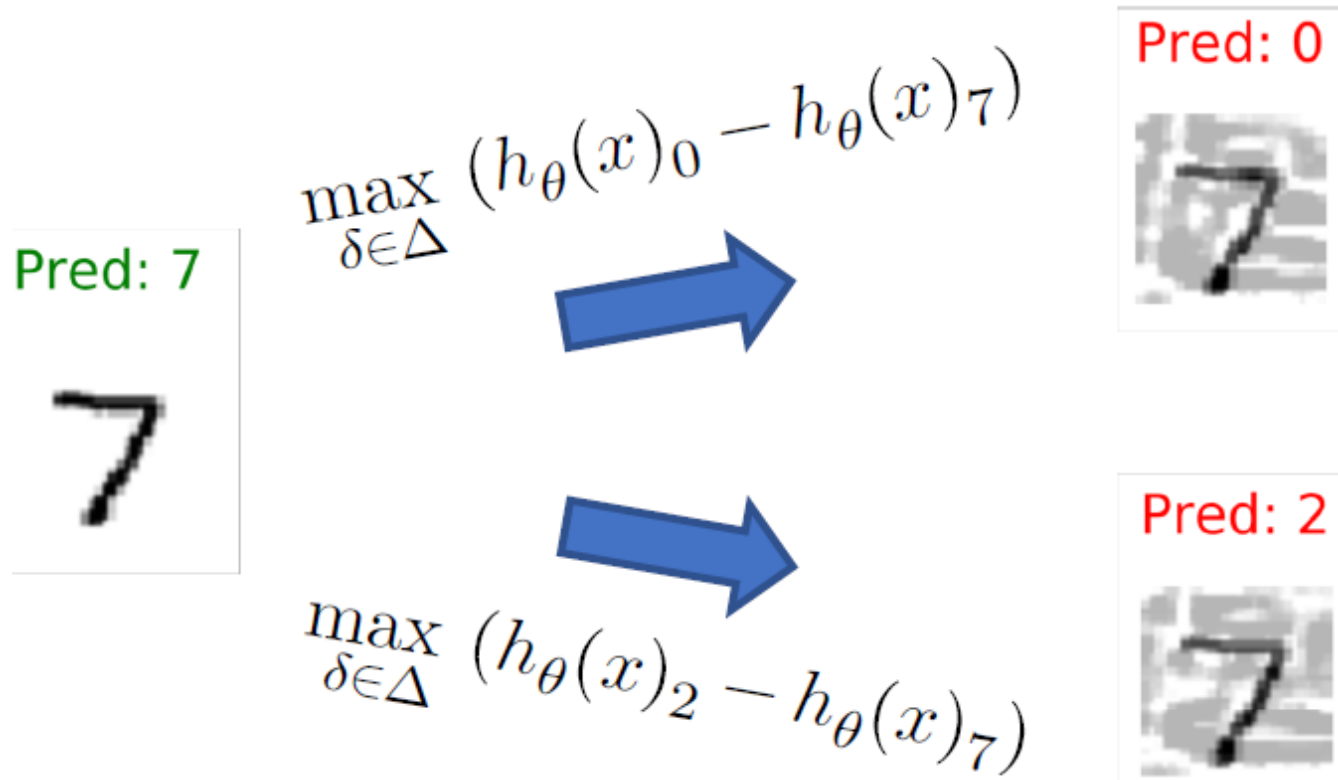
  $\log\left(\dfrac{\exp(h_\theta(x)_y)}{\sum_{j=1}^{k} \exp(h_\theta(x)_j)}\right) = \log\big(\sum_{j=1}^{k} \exp\big(h_\theta(x)_j\big)\big) - h_\theta(x)_y$

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

Targeted attacks

- Explicitly try to change label to a particular target class y_{targ}:

  - $\max\limits_{\delta \in \Delta} (\text{Loss}(x + \delta, y; \theta) - \text{Loss}(x + \delta, y_{targ}; \theta)$

- Recall Cross-Entropy loss from "L3 Intro to ML":

  - $\text{Loss}(x + \delta, y; \theta) = \log\left(\sum_{j=1}^{k} \exp\left(h_\theta(x + \delta)_j\right)\right) - h_\theta(x + \delta)_y$

- The maximization problem aims to maximize logit $\left(h_\theta(x)\right)_{y_{targ}}$ of the target class y_{targ}, and minimize logit $\left(h_\theta(x)\right)_y$ of the true class y:

  - $\max\limits_{\delta \in \Delta} (h_\theta(x + \delta)_{y_{targ}} - h_\theta(x + \delta)_y)$

- An alternative is to maximize the logit $\left(h_\theta(x)\right)_{y_{targ}}$ of the target class y_{targ}, and minimize the logit $\left(h_\theta(x)\right)_{y'}$ of all the other classes y':

  - $\max\limits_{\delta \in \Delta} (h_\theta(x + \delta)_{y_{targ}} - \sum_{y' \neq y_{targ}} h_\theta(x + \delta)_{y'})$

- (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 δ)

Targeted Attacks Examples

Pred: 7

7

$$\max_{\delta \in \Delta} (h_\theta(x)_0 - h_\theta(x)_7)$$

Pred: 0

$$\max_{\delta \in \Delta} (h_\theta(x)_2 - h_\theta(x)_7)$$

Pred: 2

- 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]

[Sharif Bhagavatula Bauer Reiter 2016]

[Athalye Engstrom Ilyas Kwok 2017]

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

33

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_{targ} (λ is the Lagrange multiplier)

  – $\text{argmin}_\delta \lambda \|\delta\|_p + J(f_\theta(x + \delta), y_{targ})$

- 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

  – $\text{argmin}_\delta \lambda \|\delta\|_p + \frac{1}{k} \sum_{i=1}^{k} J(f_\theta(x + \delta), y^*)$

Optimizing Spatial Constraints

- To make the perturbation imperceptible to humans, we add a mask M_x localize the perturbation to specific areas of the Stop Sign to mimic vandalism:

  - $\text{argmin}_\delta \lambda \|M_x \cdot \delta\|_p + \frac{1}{k}\sum_{i=1}^{k} J(f_\theta(x + M_x \cdot \delta), y^*)$

  - 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 δ within these regions

- Video demos:

  - "Bo Li – Secure Learning in Adversarial Autonomous Driving Environments"
    https://www.youtube.com/watch?v=0VfBGWnFNuw&t=421s

Subtle Poster

Camouflage Sticker

Mimic vandalism

"Hide in the human psyche"

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.

(a)　　　　　(b)　　　　　(c)　　　　　(d)
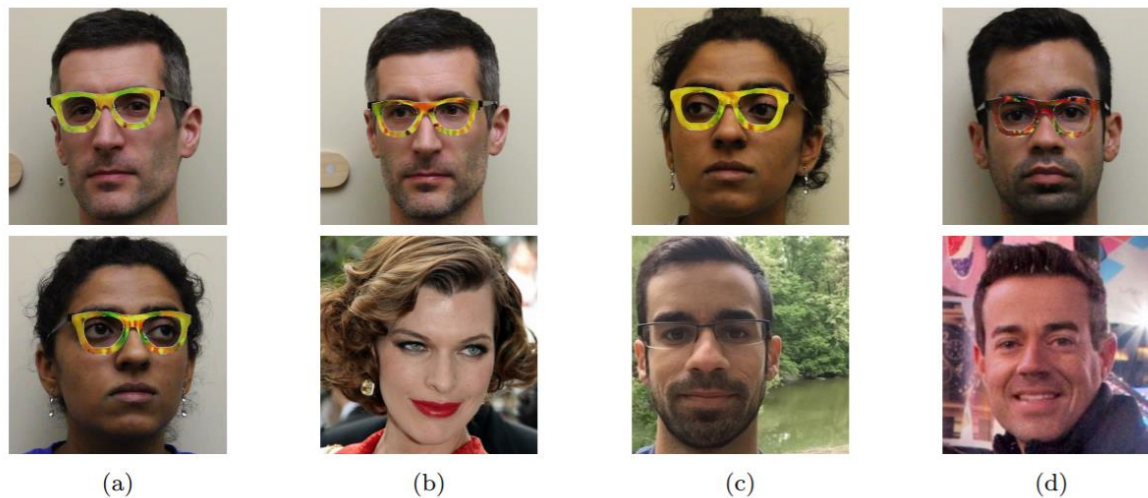
Figure 4: Examples of successful impersonation and dodging attacks. Fig. (a) shows S_A (top) and S_B (bottom) dodging against DNN_B. Fig. (b)–(d) show impersonations. Impersonators carrying out the attack are shown in the top row and corresponding impersonation targets in the bottom row. Fig. (b) shows 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).

https://www.cs.cmu.edu/~sbhagava/papers/face-rec-ccs16.pdf

Blackbox Attacks

- We have been discussing Whitebox attacks, where we know the NN model parameters θ
- 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

Attacked
Input Image

Target Blackbox Model

Proxy Whitebox Model

Training Data
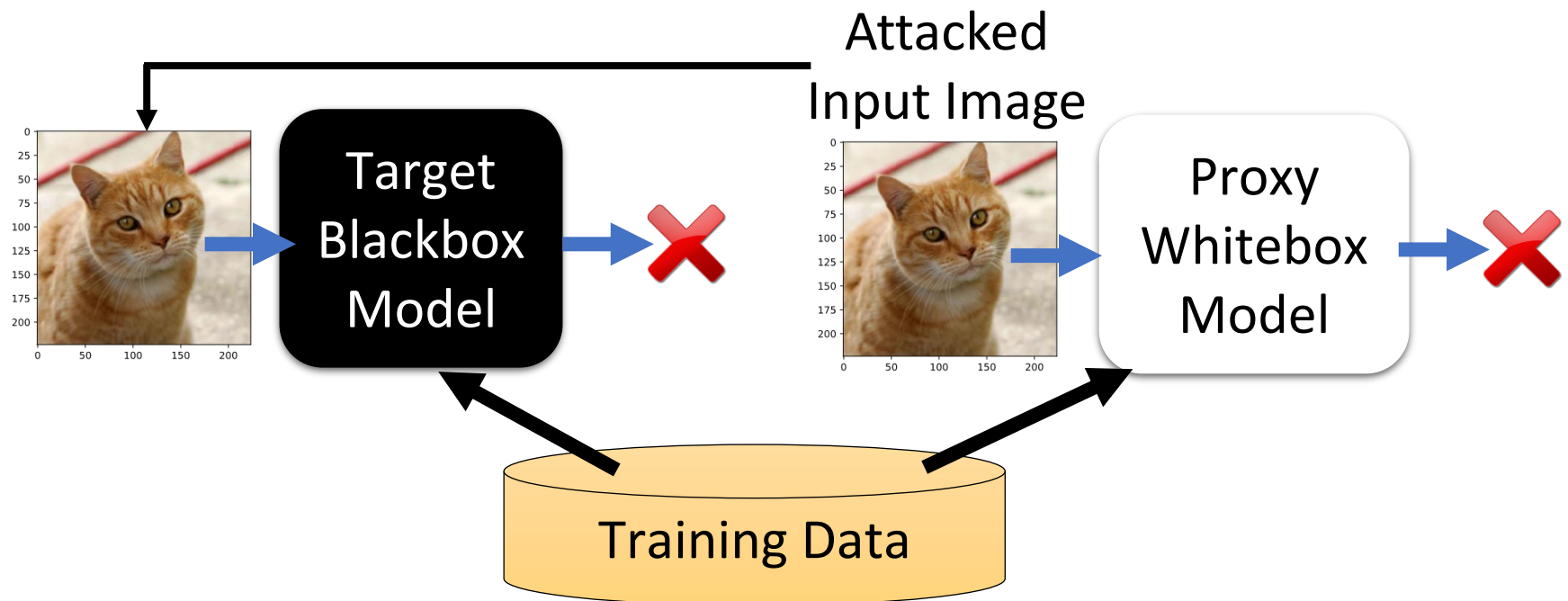
Blackbox Attack Example

- [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)
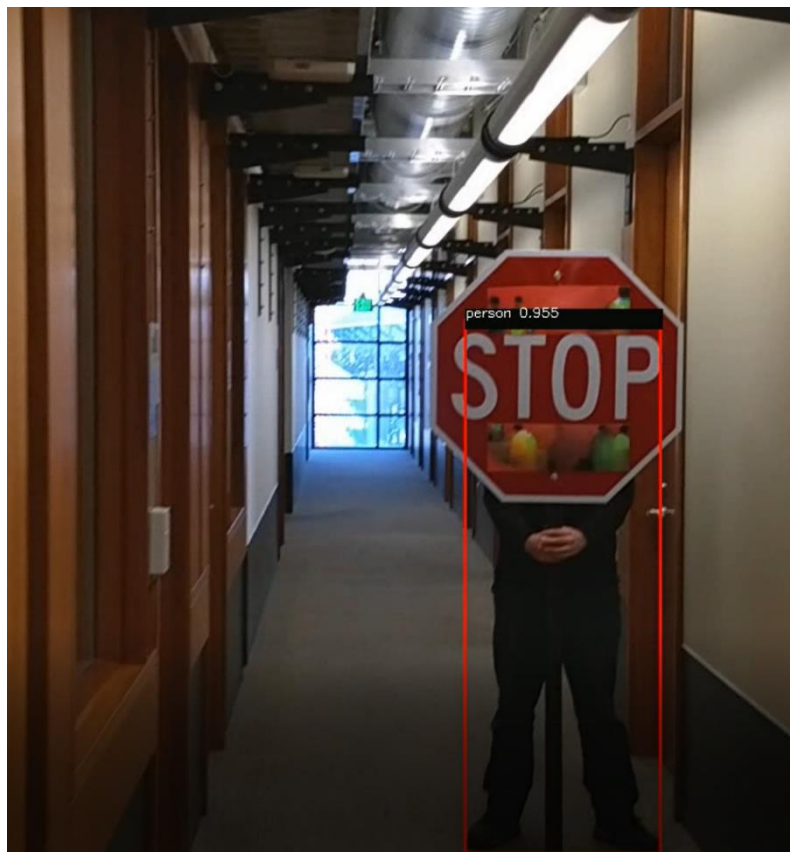
DeepBillboard

- 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.

Zhou H, Li W, Kong Z, et al. Deepbillboard: Systematic physical-world testing of autonomous driving systems[C]//2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, 2020: 347-358.

40

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
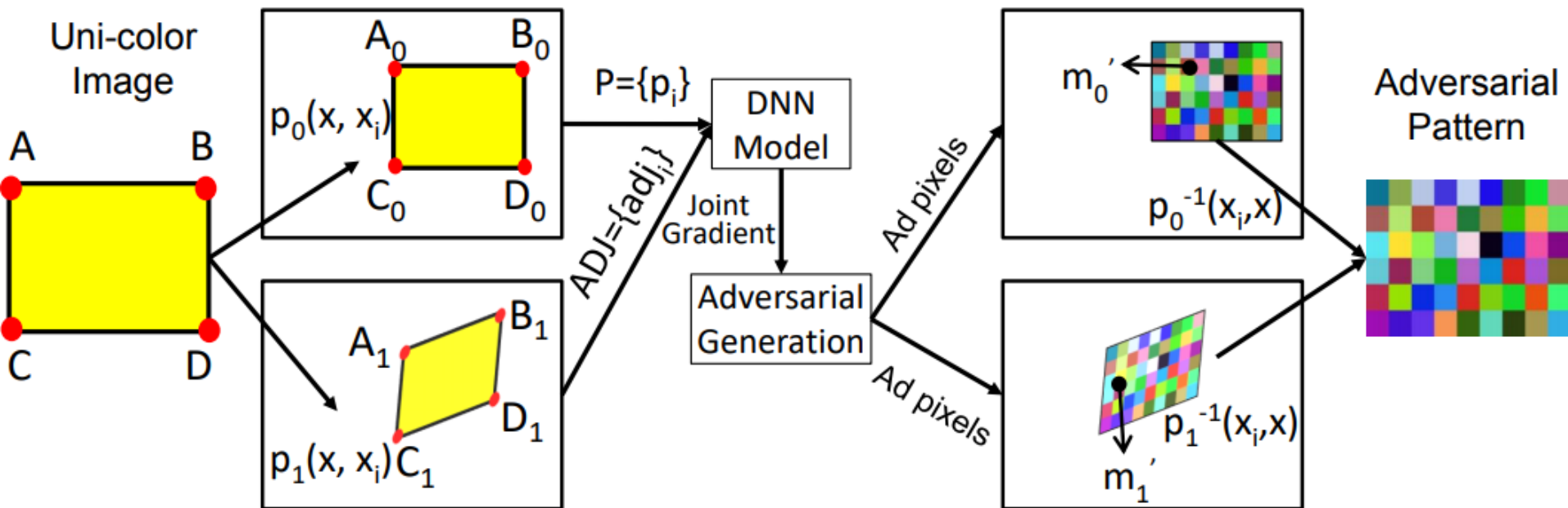
Phantom of the ADAS

- 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

Tesla's autopilot steers the car.

https://www.nassiben.com/phantoms

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
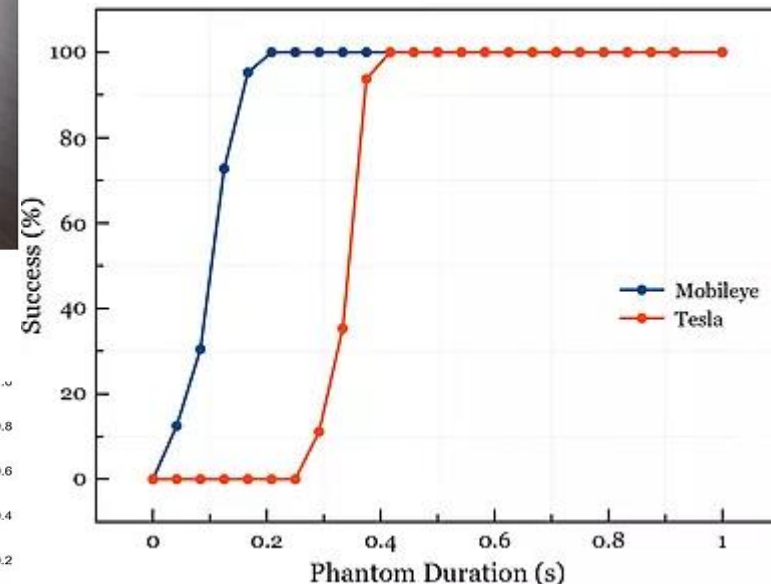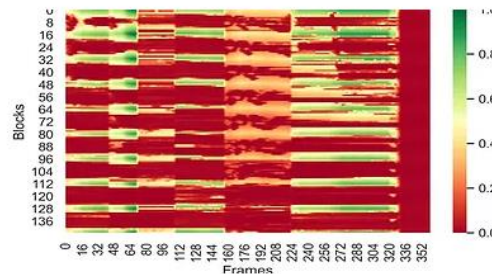
Countermeasure - GhostBusters

- 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 $\max\limits_{\delta\in\Delta} \mathrm{Loss}(x+\delta, y; \theta)$:
- 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

$\mathrm{Loss}(x+\delta, y; \theta)$

$\delta = 0$

Δ
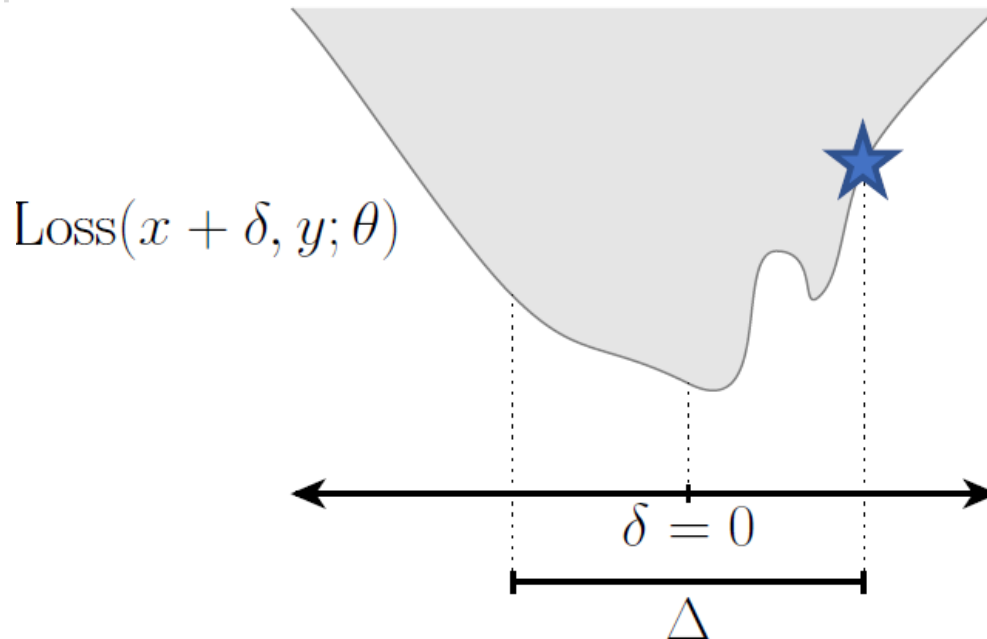
Exact Combinatorial Optimization

- 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_∞ norm:
  - $\min\limits_{z_{1:d+1}} \left(e_y - e_{y_{targ}}\right)^T z_{d+1}$ 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 δ term in favor of a constraint that simply requires z_1 (the input to the first layer), to be within ϵ of x.

- Example: binary classification $y = cat, y_{target} = dog.\ e_y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, e_{y_{targ}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$
  The objective function is $\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} (z_{d+1})_y \\ (z_{d+1})_{y_{targ}} \end{bmatrix} = (z_{d+1})_y - (z_{d+1})_{y_{targ}}$
  - Minimizing this objective function: try to increase the logit $(z_{d+1})_{y_{targ}}$ 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 ~100 hidden units, but size depends heavily on problem structure (including, e.g, the size of ϵ)

- 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:

$$W_i z_i + b_i \geq 0$$
$$\Rightarrow v_i = 1$$

$$
\begin{array}{c}
\color{red}{z_{i+1} \geq W_i z_i + b_i} \\
z_{i+1} \geq 0 \\
u_i \geq z_{i+1} \\
\color{red}{W_i z_i + b_i \geq z_{i+1}}
\end{array}
$$

$$z_{i+1} = W_i z_i + b_i$$

$$
\begin{array}{c}
z_{i+1} \geq W_i z_i + b_i \\
z_{i+1} \geq 0 \\
u_i \cdot v_i \geq z_{i+1} \\
W_i z_i + b_i \geq z_{i+1} + (1 - v_i) l_i \\
v_i \in \{0,1\}^{|v_i|}
\end{array}
$$

$$W_i z_i + b_i < 0$$
$$\Rightarrow v_i = 0$$

$$
\begin{array}{c}
z_{i+1} \geq W_i z_i + b_i \\
\color{red}{z_{i+1} \geq 0} \\
\color{red}{0 \geq z_{i+1}} \\
W_i z_i + b_i \geq z_{i+1} + l_i
\end{array}
$$

$$z_{i+1} = 0$$
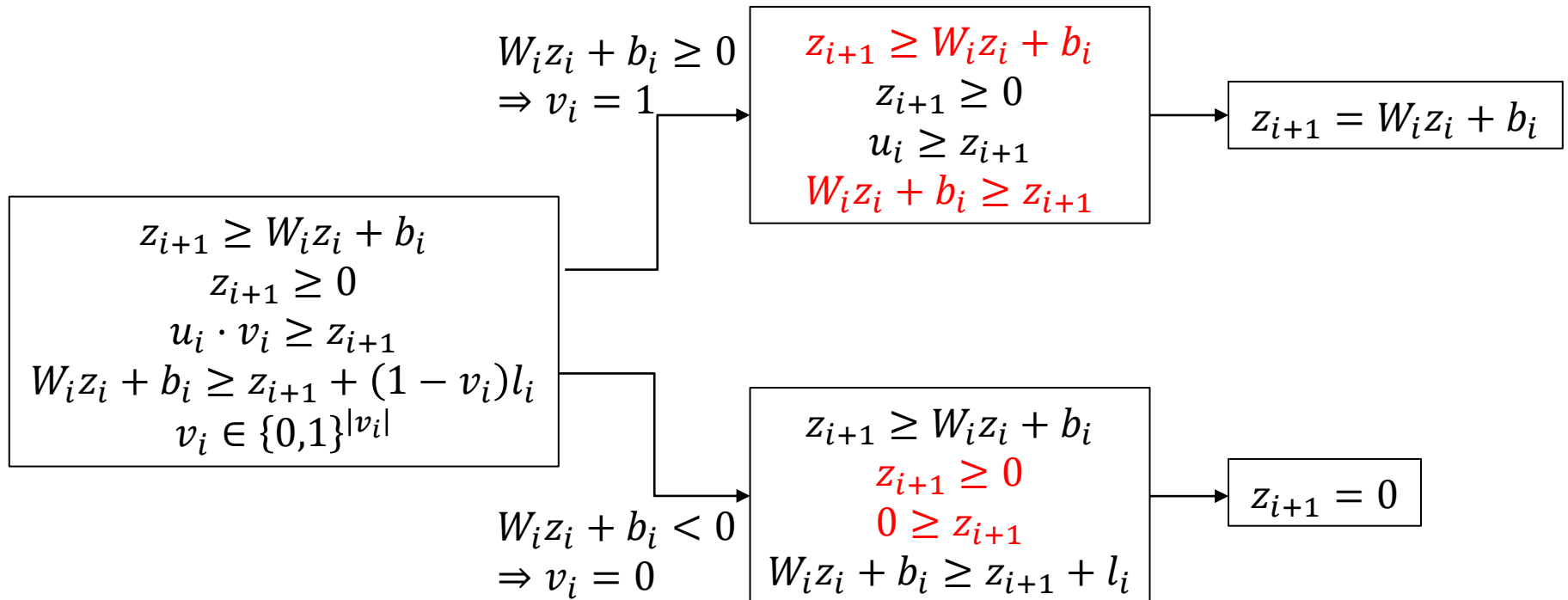
Bound Propagation

- How to get the bounds at each layer $l_i \leq W_i z_i + b_i \leq u_i$?
- If $1 \leq z \leq u$, then $[W]_+ l + [W]_- u + b \leq W z + b \leq [W]_+ u + [W]_- l + b$
  - where $[W]_- := \min\{W, 0\}, [W]_+ := \max\{W, 0\}$.
  - It is a loose bound in general, but still useful
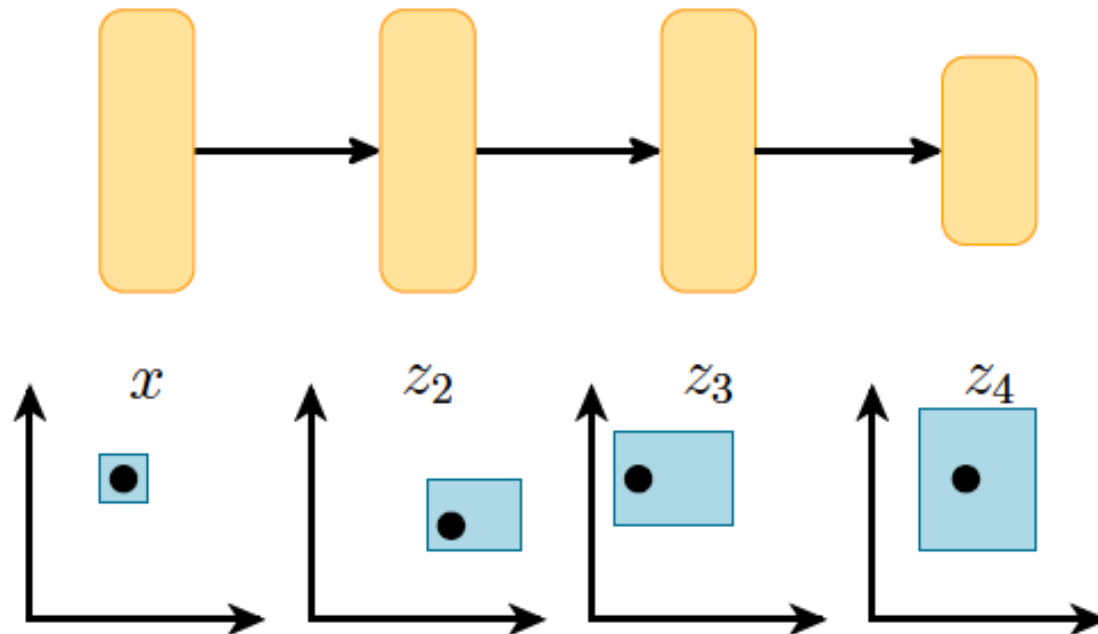
Bound Propagation Example

- $W = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \begin{bmatrix} 0 \\ .1 \end{bmatrix} \leq z = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} .3 \\ .4 \end{bmatrix}, b = 0$

- $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ .1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} .3 \\ .4 \end{bmatrix} \leq Wz \leq \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} .3 \\ .4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ .1 \end{bmatrix}$

- $\begin{bmatrix} .1 \\ -.4 \end{bmatrix} \leq Wz = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} \leq \begin{bmatrix} .7 \\ .2 \end{bmatrix}$

- 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

51
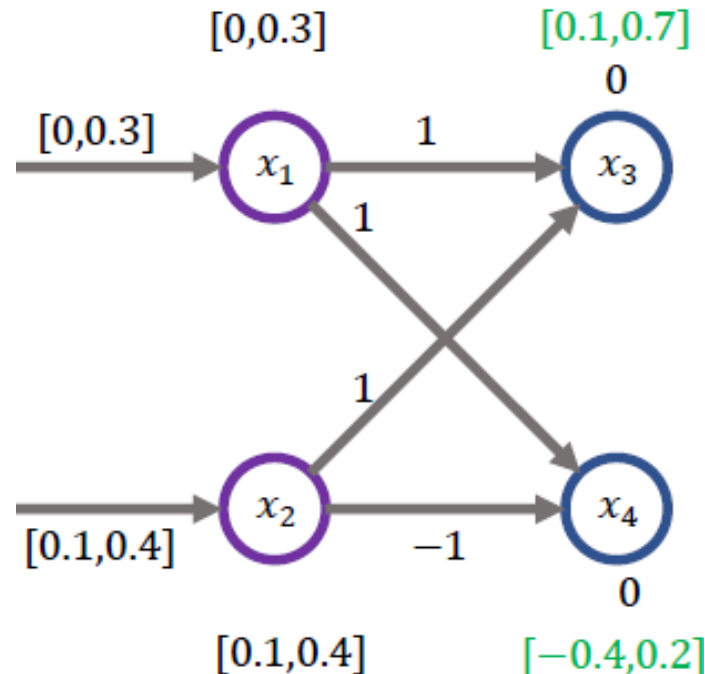
Final ILP Formulation

- $\min\limits_{z_{1:d+1}, v_{1:d-1}} \left(e_y - e_{y_{targ}}\right)^T z_{d+1}$ 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 CPEX or cvxpy+Gurobi

Certifying Robustness

- Consider the optimization objective $\left(e_y - e_{y_{targ}}\right)^T z_{d+1}$. If we solve it for some y_{targ} and the objective is positive, then this is a certificate that there exists no adversarial example for that target class
  - Example: binary classification $y = cat, y_{target} = dog$. If $\min_{z_{1:d}}\left((z_{d+1})_y - (z_{d+1})_{y_{targ}}\right) \geq 0$, then it is not possible to misclassify a cat image as a dog
- If objective is positive for all y_{targ}, this is a verified proof that there exists no adversarial example at all

Pred: 7

7

$$\min \ (e_7 - e_0)^T(W_d z_d + b_d)$$
$$\text{s.t.} \ ...$$

= -2.54 (exists adversarial example for target class zero or another class)

$$\min \ (e_7 - e_1)^T(W_d z_d + b_d)$$
$$\text{s.t.} \ ...$$

= 3.04 (there is *no* adversarial example to make classifier predict class 1)

Approach #3

- To solve $\max\limits_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$:
- 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

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

$\delta = 0$

Δ
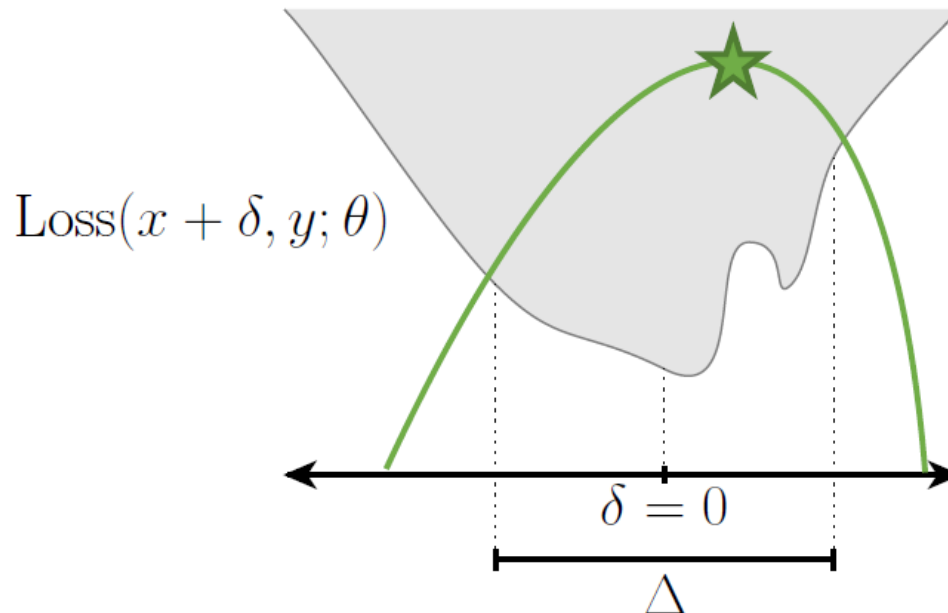
Convex Relaxation

- 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)

Accurate ReLU

Convex relaxation of ReLU
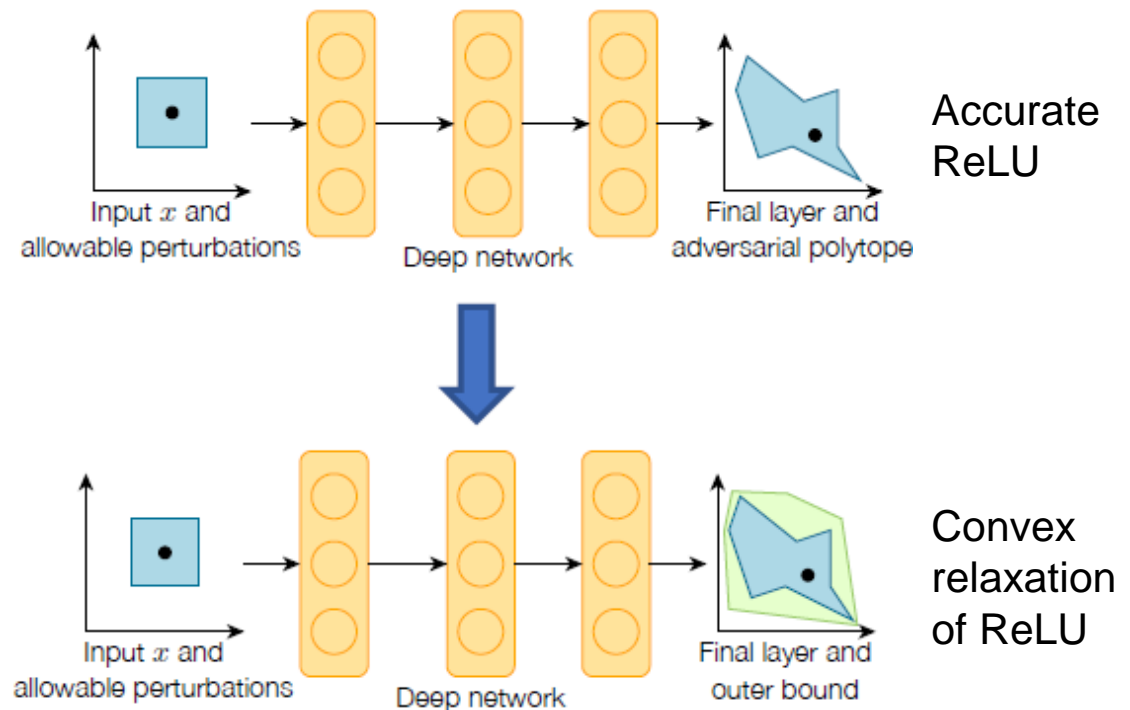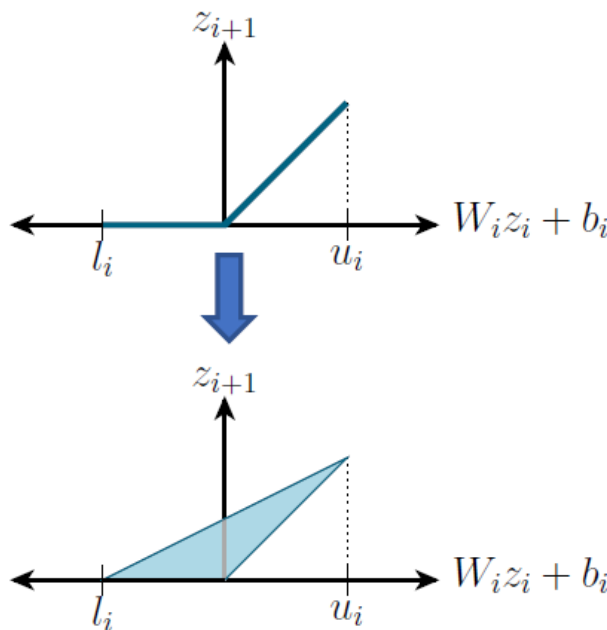
Convex Relaxation as Conservative Approximation

- 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

Pred: 7

$$\min \ (e_7 - e_0)^T(W_d z_d + b_d)$$
$$\text{s.t. Binary IP constraints}$$

= -2.54 (*exists* adversarial example for target class zero or another class)

$$\min \ (e_7 - e_0)^T(W_d z_d + b_d)$$
$$\text{s.t. Convex constraints}$$

= -6.28 (*may or may not* exist adversarial example)

$$\min \ (e_7 - e_1)^T(W_d z_d + b_d)$$
$$\text{s.t. Convex constraints}$$

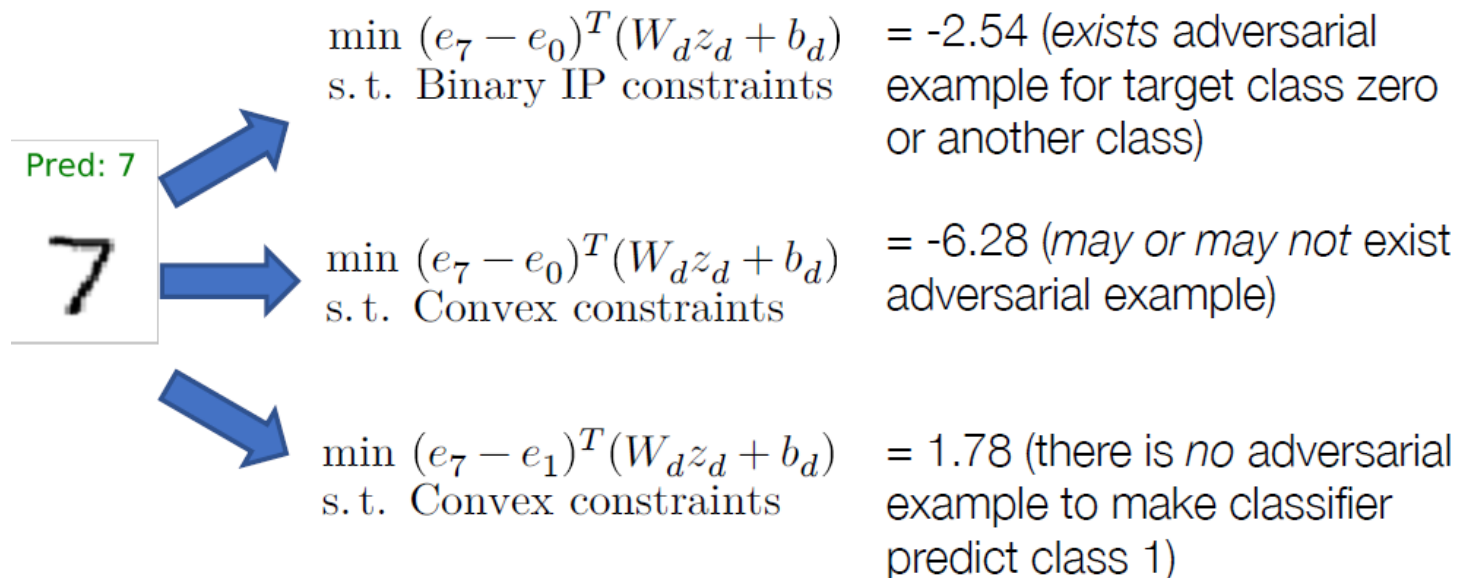= 1.78 (there is *no* adversarial example to make classifier predict class 1)

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:

  - $\min\limits_{z_d} c^T(W_d z_d + b_d) = (c^T W_d)z_d + c^T b_d$ 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:

  - $\min\limits_{z_d} c^T(W_d z_d + b_d) = [c^T W_d]_+ l + [c^T W_d]_- u + c^T b_d$

- 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 training
  - Robust optimization with convex relaxations
- Adversarial robustness beyond security

The Outer Minimization Problem

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

- We would like to solve it with gradient descent, but how do we 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 δ^* 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 $\delta^\star$ by (approximately) optimizing

   $$\delta^\star = \underset{\|\delta\| \leq \epsilon}{\mathrm{argmax}}\, \ell(h_\theta(x + \delta), y)$$

   b. Add gradient at $\delta^\star$

   $$g := g + \nabla_\theta \ell(h_\theta(x + \delta^\star), y)$$
3. Update parameters θ

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

- 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_∞ robust model against l_1 or l_2 attacks
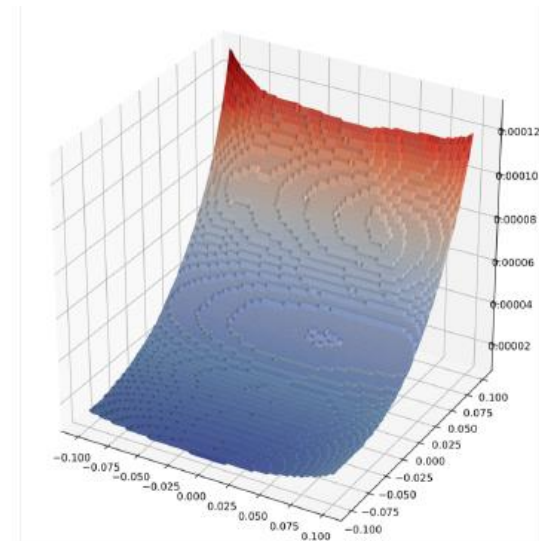
What Makes the Models Robust?

- 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

74.4%

41.7%

2.6%

0.9% 2.8%

1.1%

ConvNet   Robust ConvNet

■ Clean ■ FGSM ■ PGD
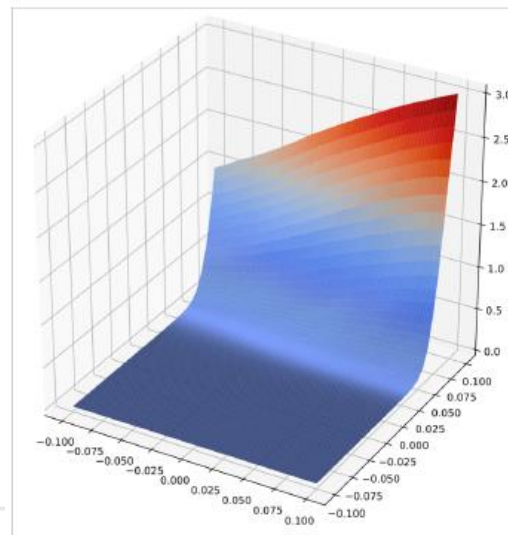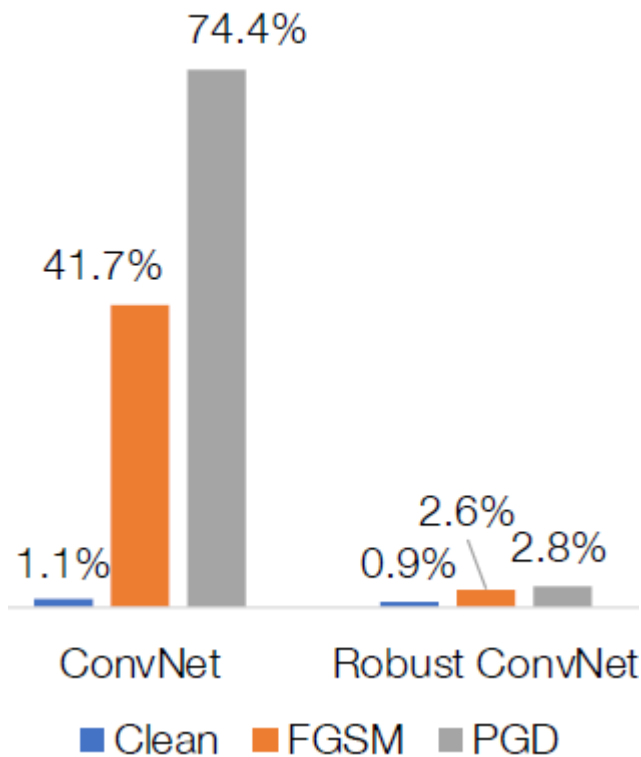
Loss surface:
standard training
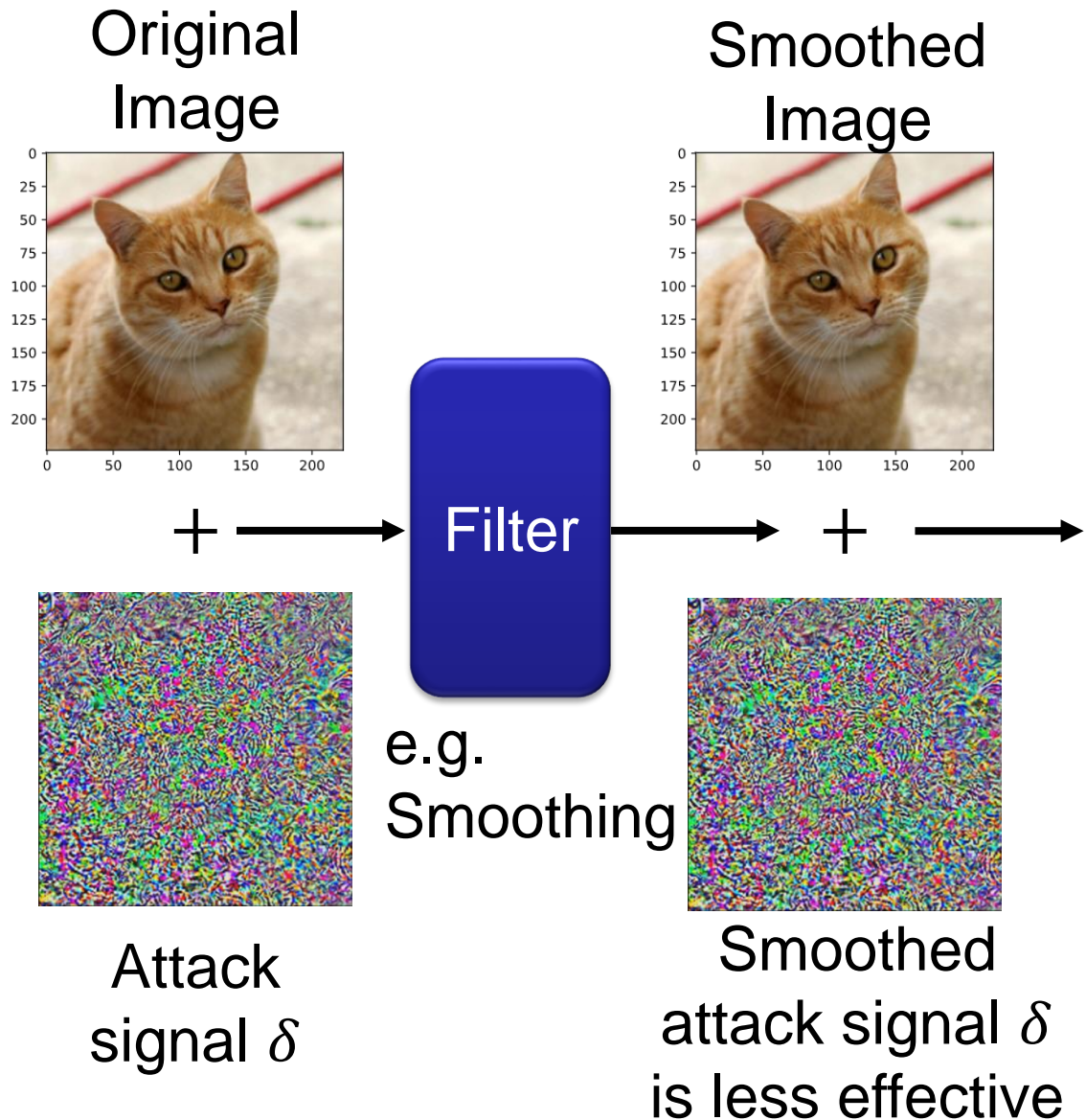
Loss surface:
robust training

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)
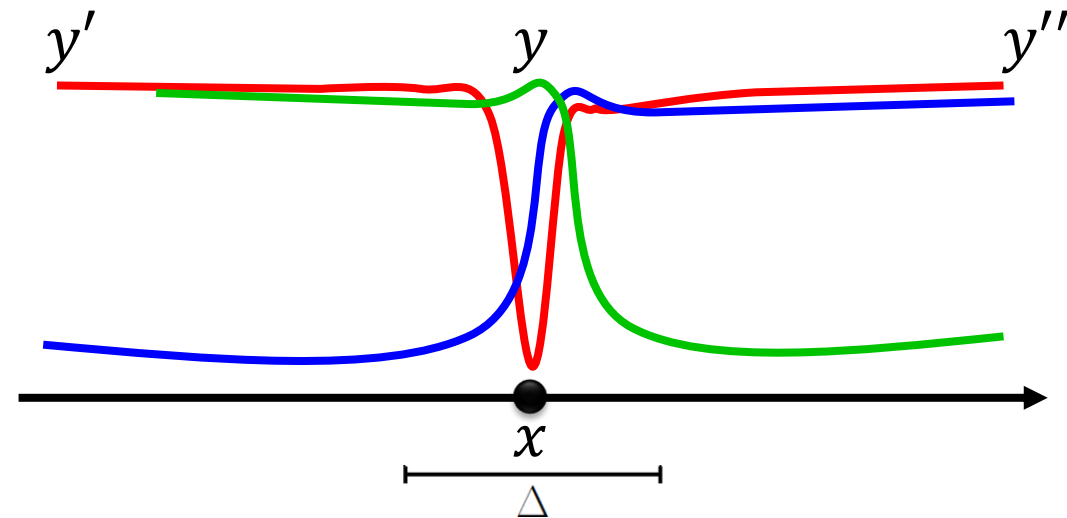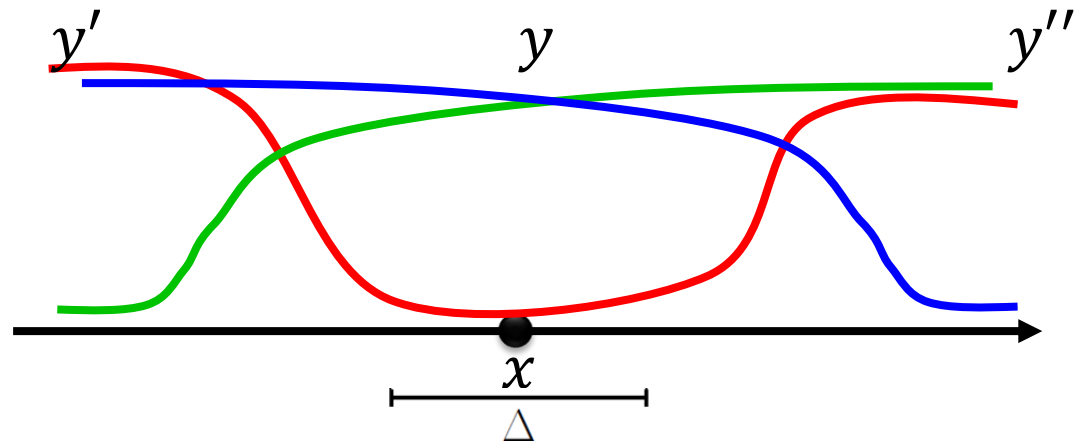
Smoothing Filter as Defense

- The filter helps make the loss surface smoother, which makes attacks more difficult

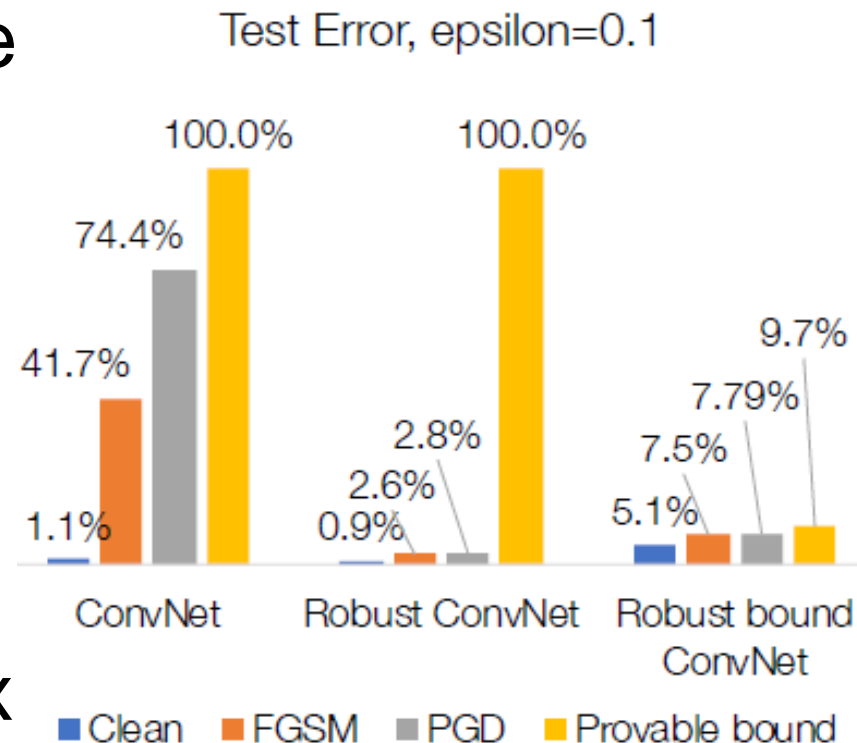- Caveat: if attacker knows the filter, the defense is no longer effective

Original Image

Smoothed Image

+ → Filter → +

e.g. Smoothing

Attack signal δ

Smoothed attack signal δ is less effective

Image credit: Hung-yi Lee

Convex Methods for Certifying Robustness

Test Error, epsilon=0.1

- Let's use the convex bounds to see what level of adversarial performance we can guarantee for the robust model
  - Test errors have provable bound of 100%, i.e., we can make no guarantees at all for this testset, even for the robust model
- Insight: models that can be (convexly) verified to be robust are a small subset of the actually robust models. Convex bounds are typically very loose unless the model was built with them in mind
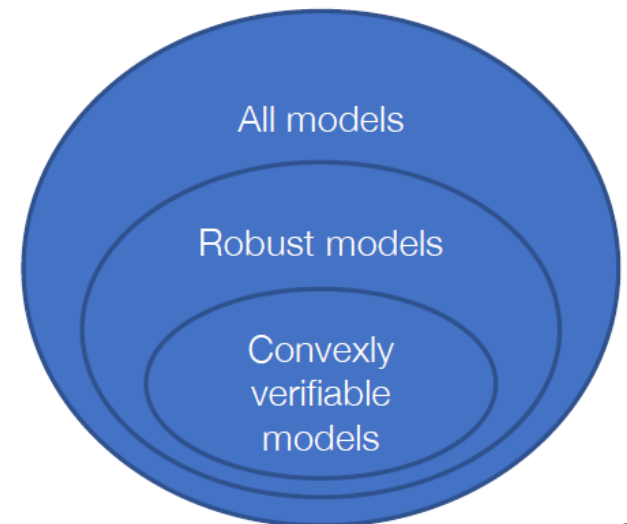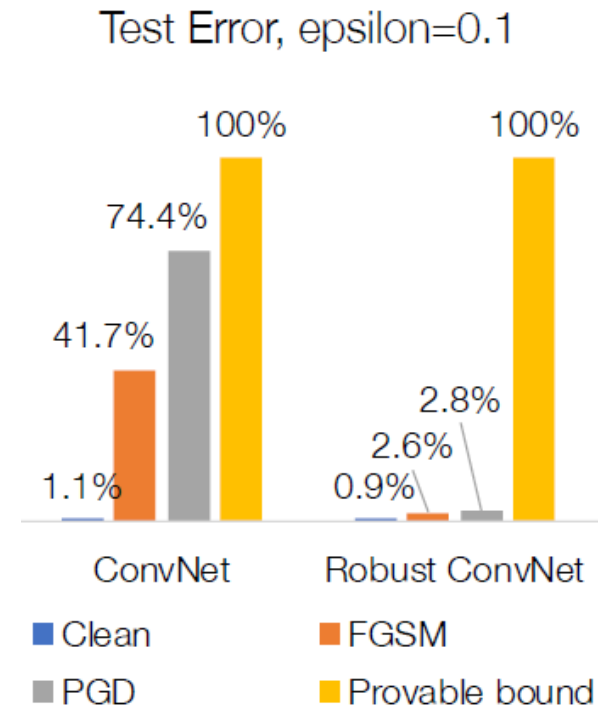
Training Provably Robust Models

- Convex bounds (e.g., recall Interval-based Bounds) are differentiable functions of network parameters

- Can train networks to minimize these bounds directly

- Recent work shows that interval bounds often outperform more complex strategies [Mirman et al, 2018, Gowal et al., 2018]

Test Error, epsilon=0.1

100.0% 100.0%

74.4%

41.7%

9.7%

7.79%

7.5%

5.1%

2.8%

2.6%

1.1%

0.9%

ConvNet   Robust ConvNet   Robust bound ConvNet

■ Clean   ■ FGSM   ■ PGD   ■ Provable bound

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
- Defenses
  - Adversarial training
  - Robust optimization with convex relaxations
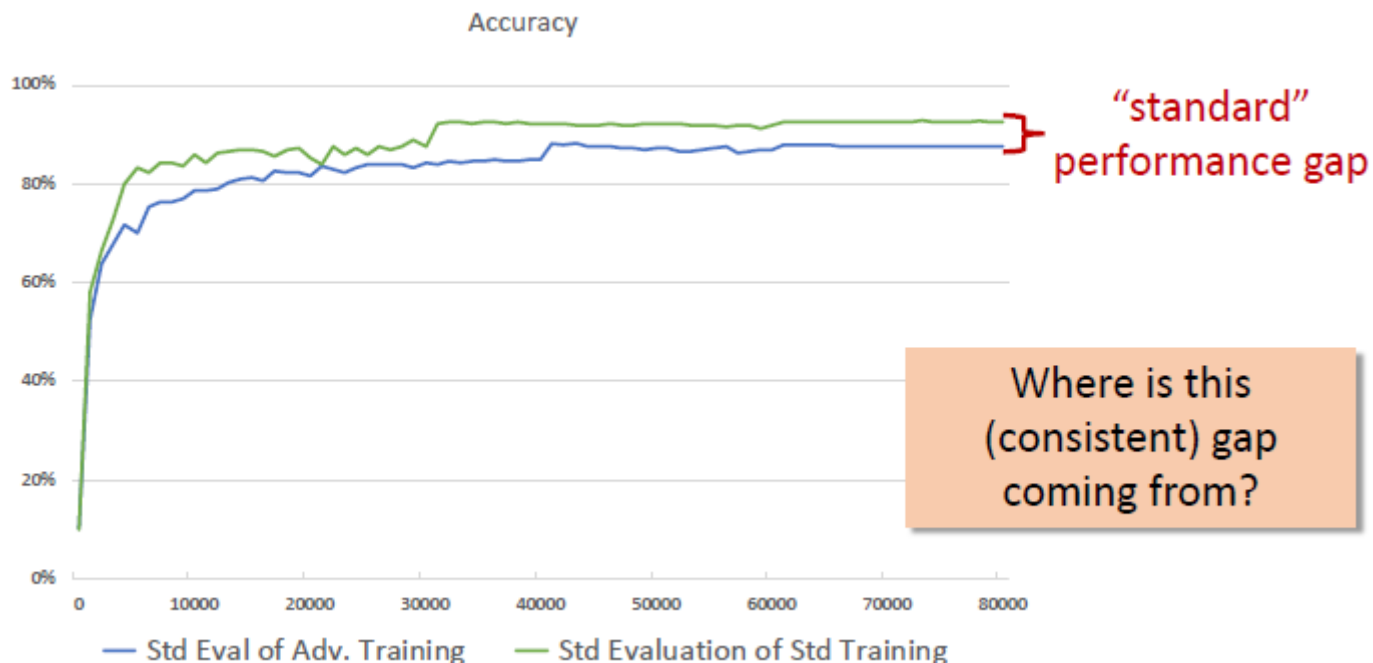- Adversarial robustness beyond security

Adv. Robust Generalization Needs More Data to Avoid Overfitting
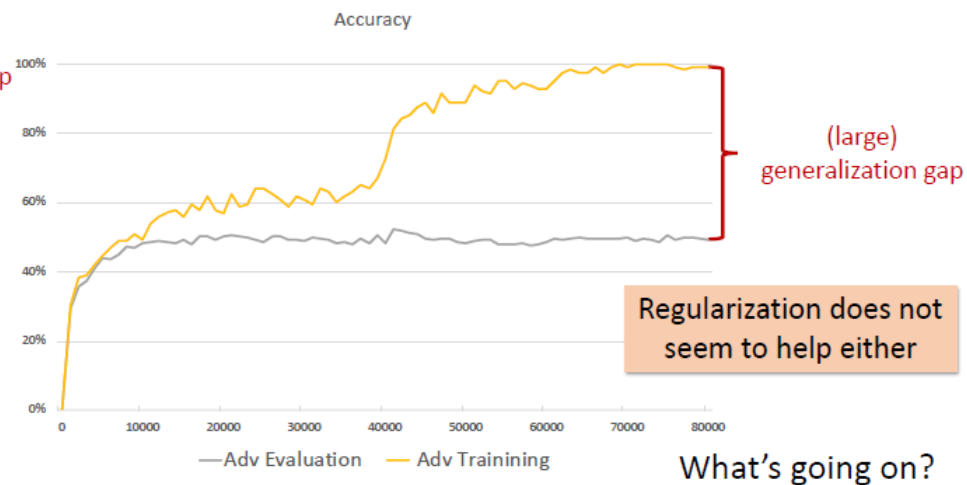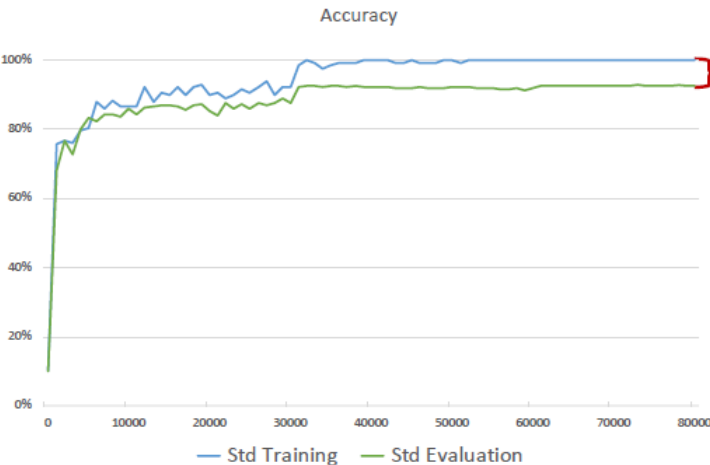
- Theorem [Schmidt Santurkar Tsipras Talwar M 2018]: Sample complexity of adv. robust generalization can be significantly larger than that of "standard" generalization
- Specifically: There exists a d-dimensional distribution D s.t.:
  - A single sample is enough to get an accurate classifier (P[correct] > 0.99)
  - But need $\Omega(\sqrt{d})$ samples for better-than-chance robust classifier
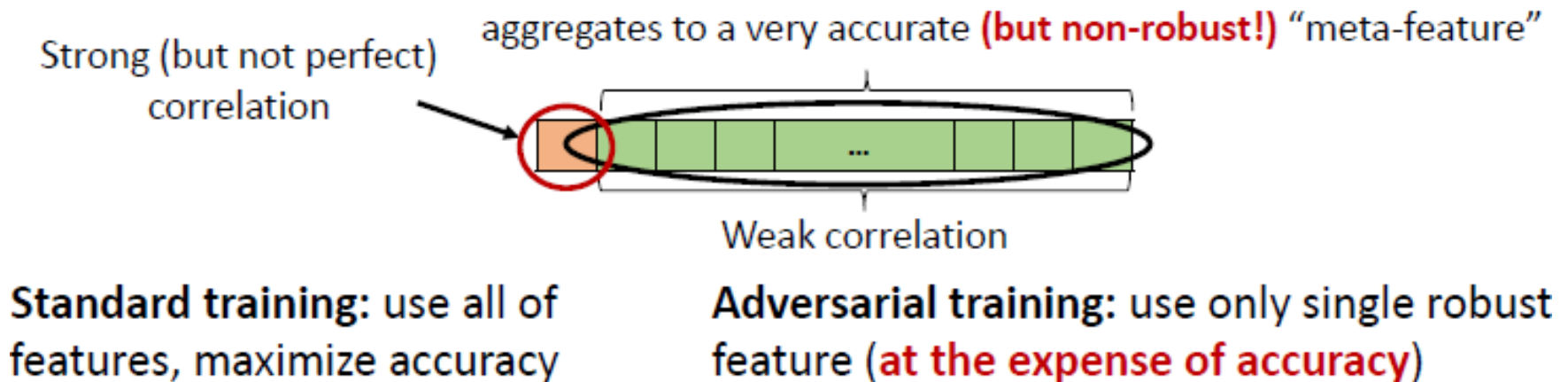
Does Being Robust Help "Standard" Generalization?

- 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

Accuracy

"standard" performance gap

Where is this (consistent) gap coming from?

— Std Eval of Adv. Training   — Std Evaluation of Std Training

Does Being Robust Help "Standard" Generalization?

- Theorem [Tsipras Santurkar Engstrom Turner M 2018]: No "free lunch": can exist a trade-off between accuracy and robustness
  - In standard training, all correlation is good correlation
  - If we want robustness, must avoid non-robust features. e.g., a robust feature for class label "dog" may be human-recognizable dog feature like a large black nose: a non-robust feature may be outdoor scenery

aggregates to a very accurate **(but non-robust!)** "meta-feature"

Strong (but not perfect) correlation

Weak correlation

Standard training: use all of features, maximize accuracy

Adversarial training: use only single robust feature **(at the expense of accuracy)**

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
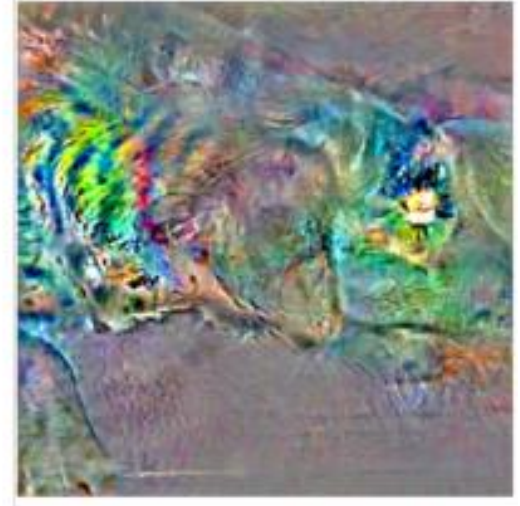
But There Are (Unexpected?) Benefits Too

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


Input
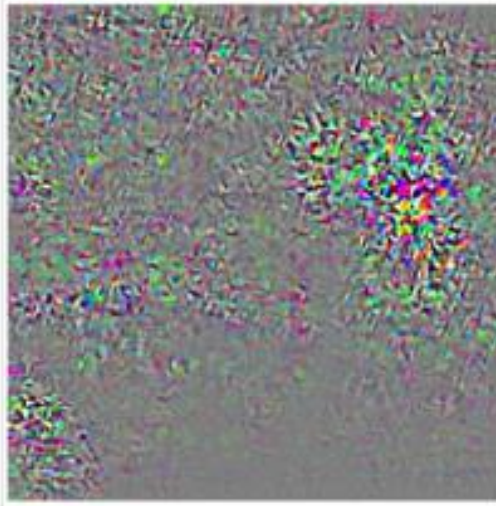
Gradient of
standard model

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

Attack
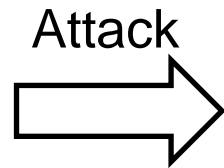
Standard model

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

- Adaptive evaluation methodology + scaling up verification