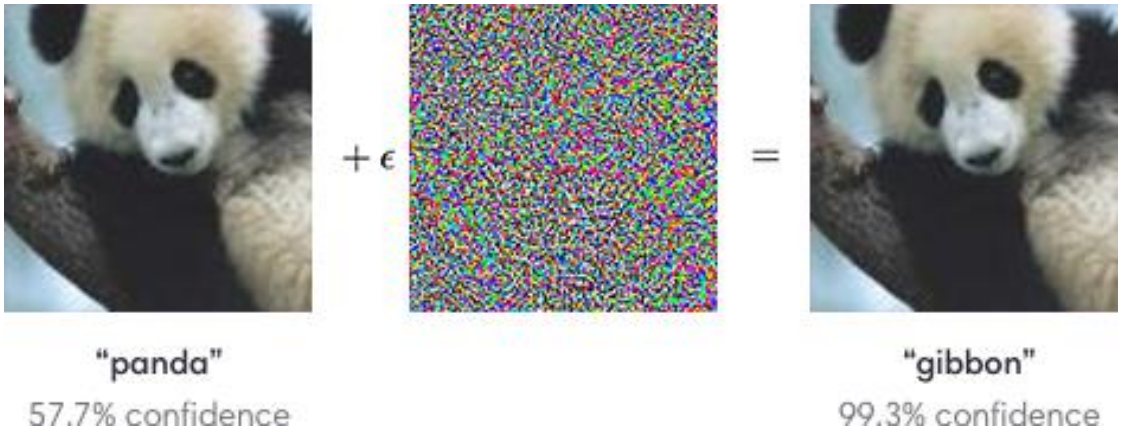
L5 Adversarial Robustness

Z. Gu 2021

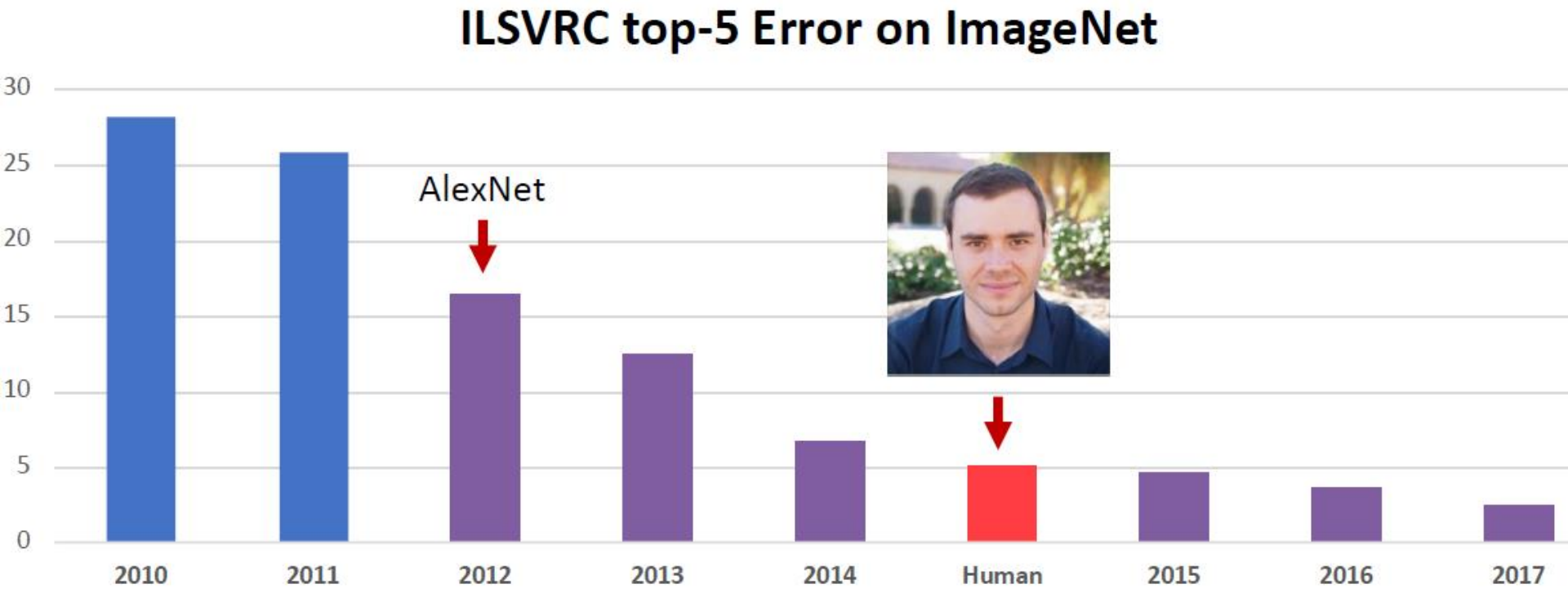


Based on ICML 2018 tutorial https://adversarial-ml-tutorial.org

Outline

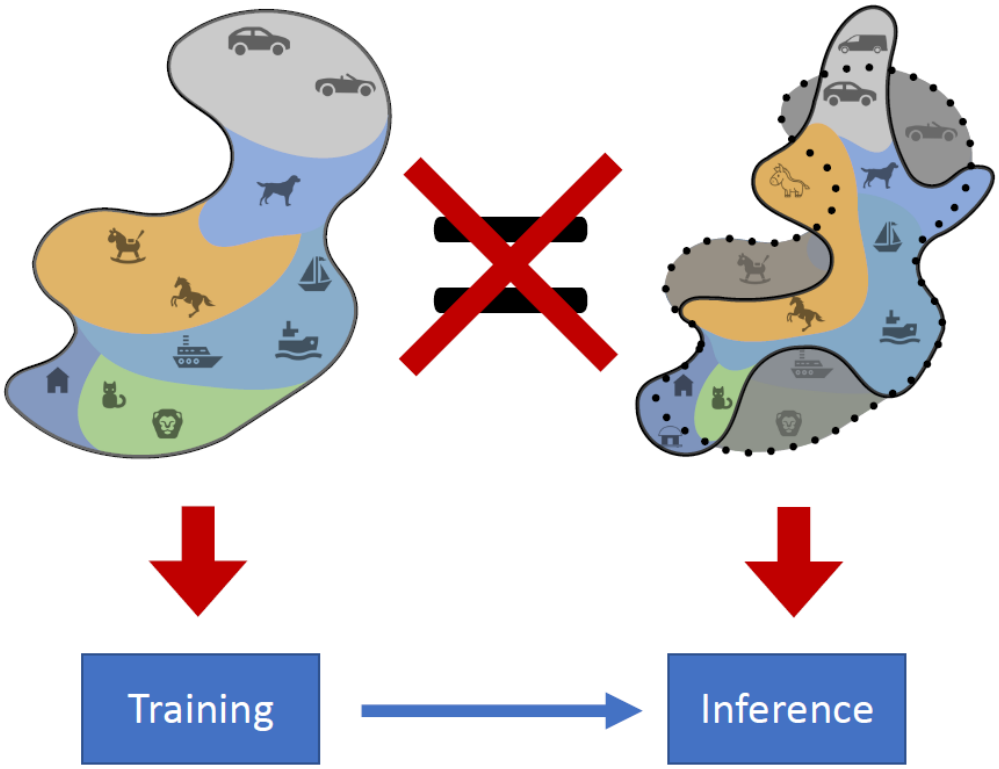
- Introduction
- Adversarial examples and verification
  - Constructing adversarial examples via local search
  - Formal verification via combinatorial optimization
  - Formal verification via convex relaxations
- Training adversarially robust models
  - Adversarial training
  - Robust optimization with convex relaxations
- Adversarial Robustness Beyond Security

ImageNet: An ML Home Run



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



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.

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$

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

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

https://openai.com/blog/adversarial-example-research/

Physically-Realizable Attacks

- Instead of directly manipulating pixels, it is possible to modify physical objects and cause miss-classification

[Kurakin Goodfellow Bengio 2017]

[Athalye Engstrom Ilyas Kwok 2017]

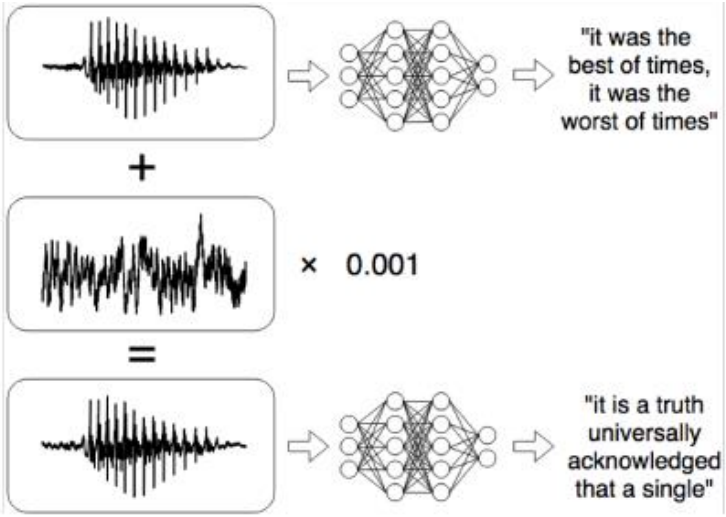
[Sharif Bhagavatula Bauer Reiter 2016]

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

Why Is This Brittleness of ML a Problem?

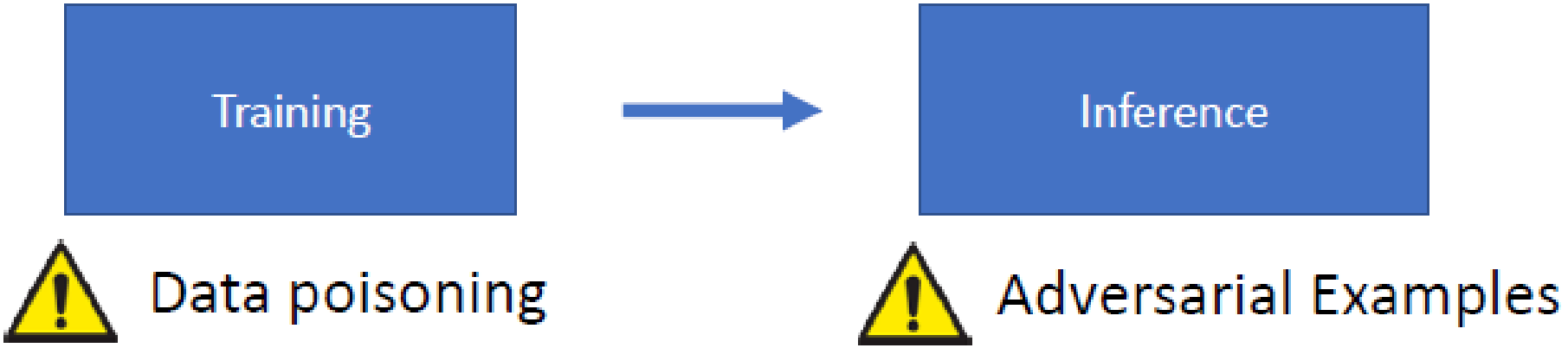
→ Security

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



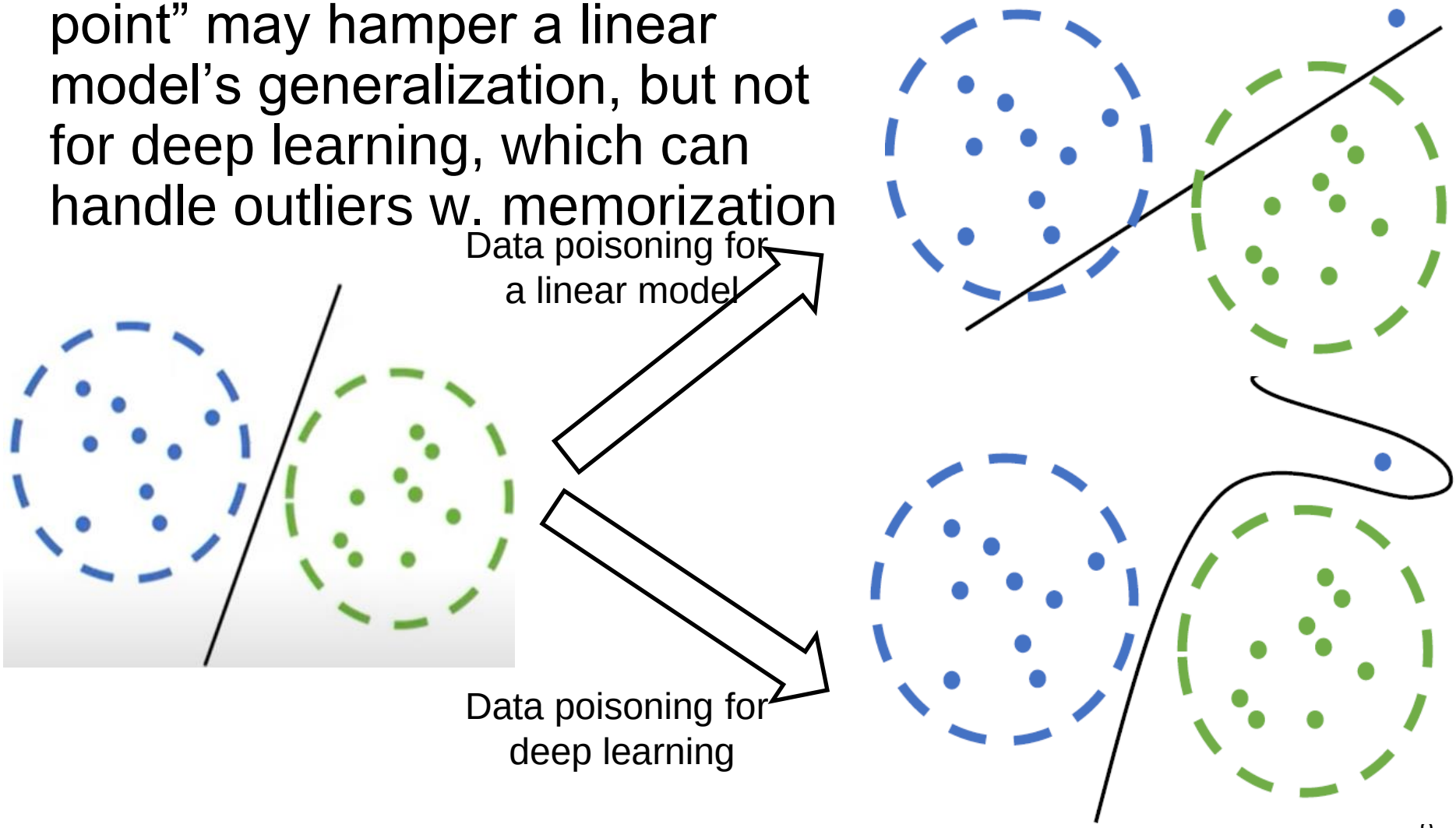
[Sharif Bhagavatula Bauer Reiter 2016]:
Glasses that fool face recognition

Training Time Attack vs. Inference Time Attack



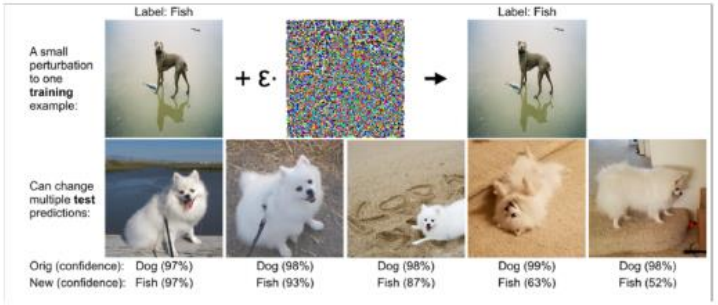
Data Poisoning

- 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

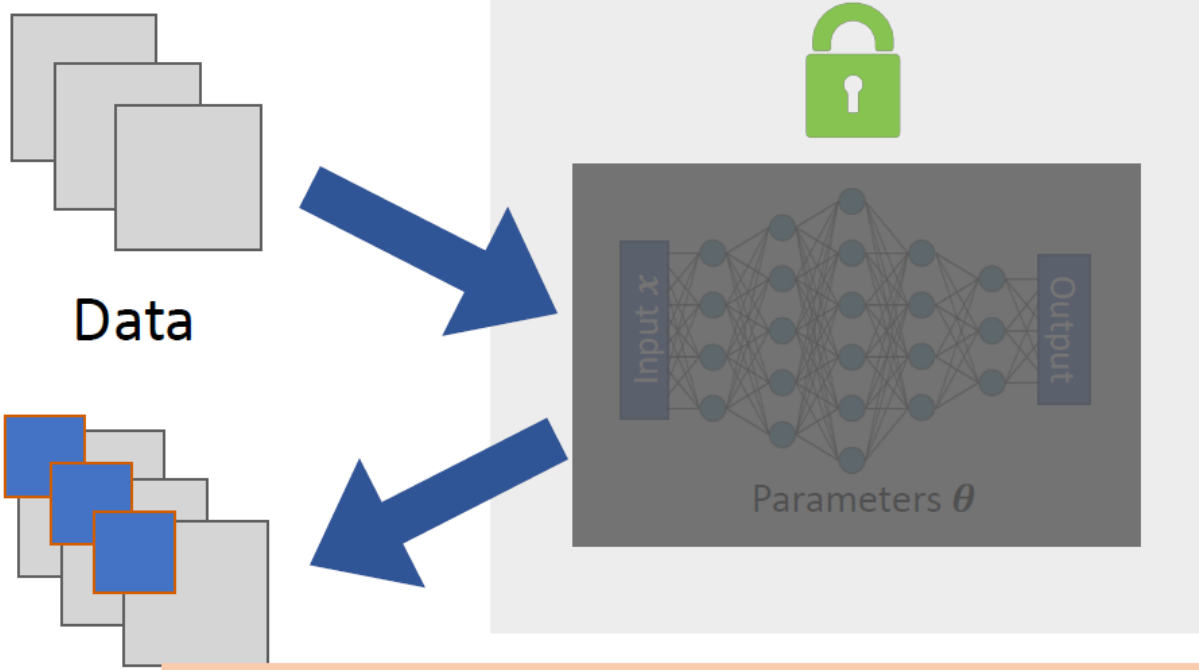
[Gu Dolan-Gavitt Garg 2017][Turner Tsipras M 2018]: Can plant an **undetectable backdoor** that gives an almost **total** control over the model

Model Stealing and Blackbox Attacks

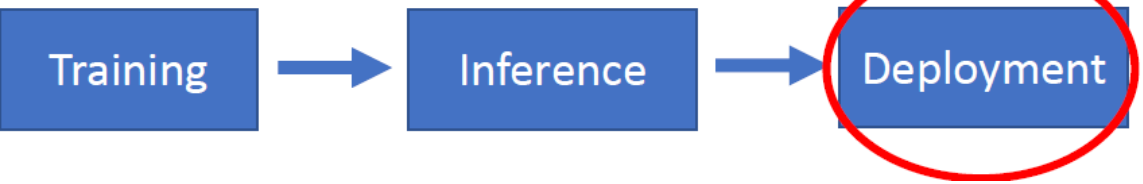
Does limited access give security?

Model stealing: "Reverse engineer" the model
[Tramer Zhang Juels Reiter Ristenpart 2016]

Black box attacks: Construct adv. examples from queries
[Chen Zhang Sharma Yi Hsieh 2017][Bhagoji He Li Song 2017][Ilyas Engstrom Athalye Lin 2017] [Brendel Rauber Bethge 2017][Cheng Le Chen Yi Zhang Hsieh 2018][Ilyas Engstrom M 2018]



For more: See my talk on Friday



Black box attacks

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
  - 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

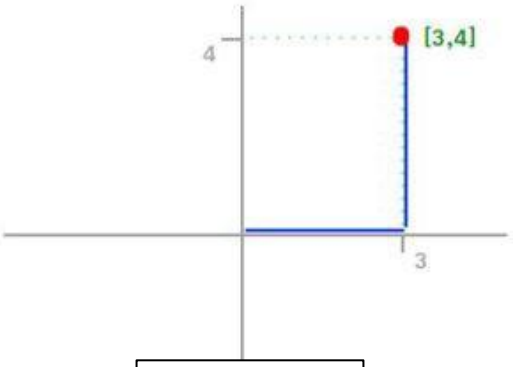
- Standard ML: Empirical Cost Minimization: $\min_{\theta} \mathbb{E}_{(x,y)\sim D} \mathrm{Loss}(x, y; \theta)$
- Adversarial Input Generation (untargeted attack): $\max_{\delta \in \Delta} \mathrm{Loss}(x + \delta, y; \theta)$
- Adversarial Robust ML: $\min_{\theta} \mathbb{E}_{(x,y)\sim D} \max_{\delta \in \Delta} \mathrm{Loss}(x + \delta, y; \theta)$
  - 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

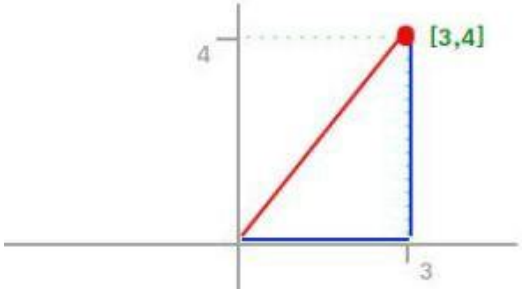
- Which input perturbations δ are allowed?
- Examples: δ that is small wrt
  - l_p norm (we focus on it in this lecture)
  - Rotation and/or translation
  - VGG feature perturbation
  - (add the perturbation you need here)

Vector Norms

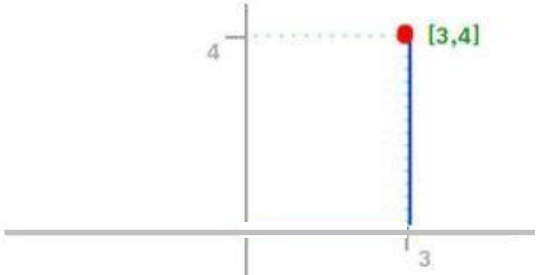
- l_p norm of a k-dimensional vector $x \in \mathbb{R}^k$ is defined as $\|x\|_p = \left(\sum_{i=1}^{k} |x_i|^p\right)^{1/p}$
- l_1 norm: $\|x\|_1 = \sum_i |x_i|$ (Manhattan Distance)
  - $\|[3,4]\|_1 = |3| + |4| = 7$
- l_2 norm: $\|x\|_2 = \sqrt{\sum_i x_i^2}$ (Euclidean norm)
  - $\|[3,4]\|_2 = \sqrt{3^2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



l_1 norm



l_2 norm

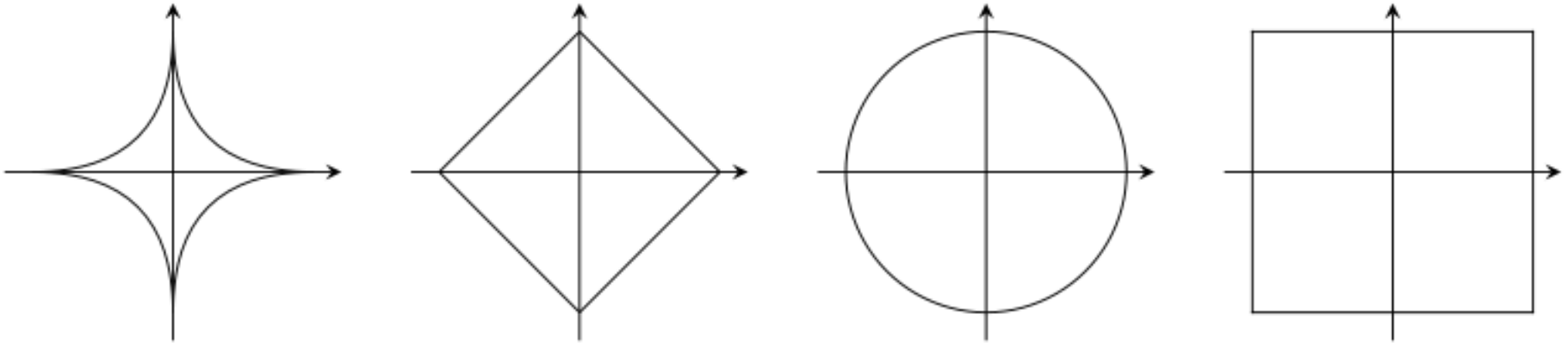


l_∞ norm

https://montjoile.medium.com/l0-norm-l1-norm-l2-norm-l-infinity-norm-7a7d18a4f40c

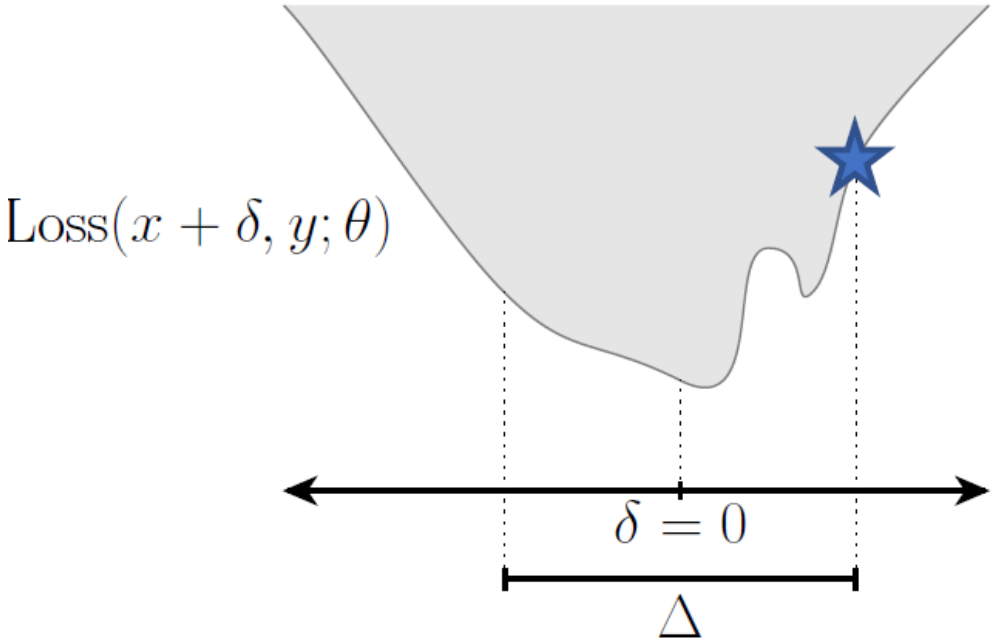
Vector Norm Balls

- 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



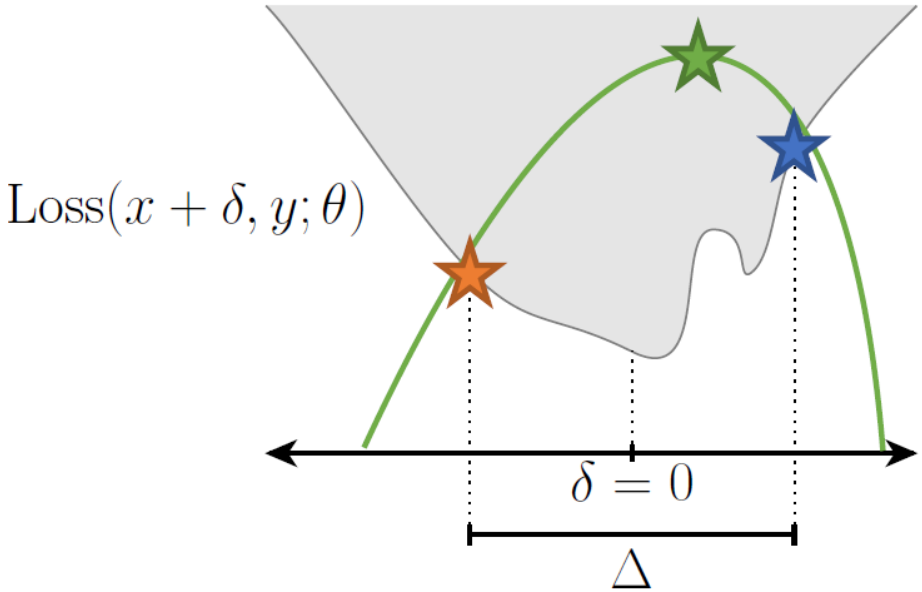
The Maximization Problem for Finding Adversarial Examples

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



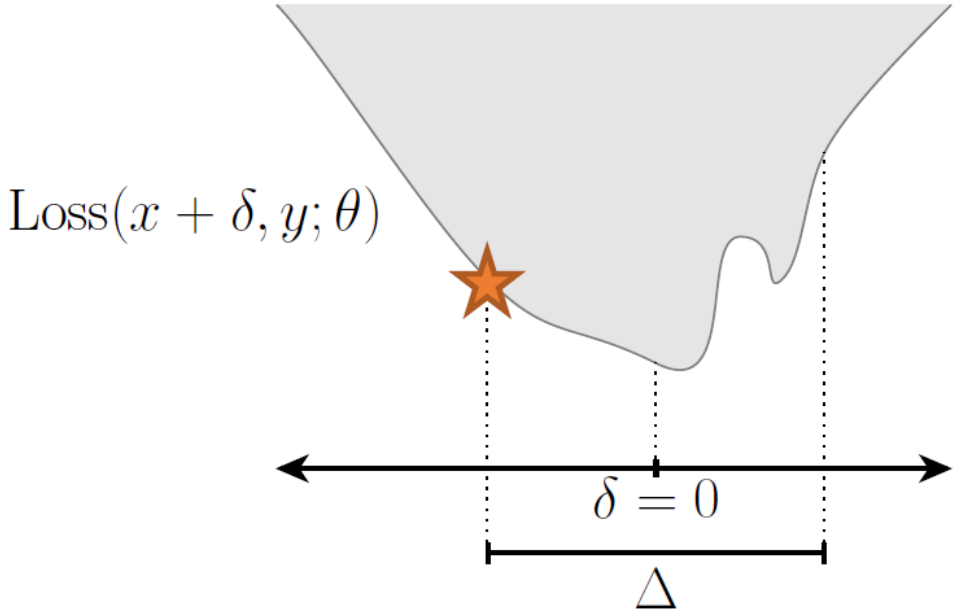
Three Approaches

- 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



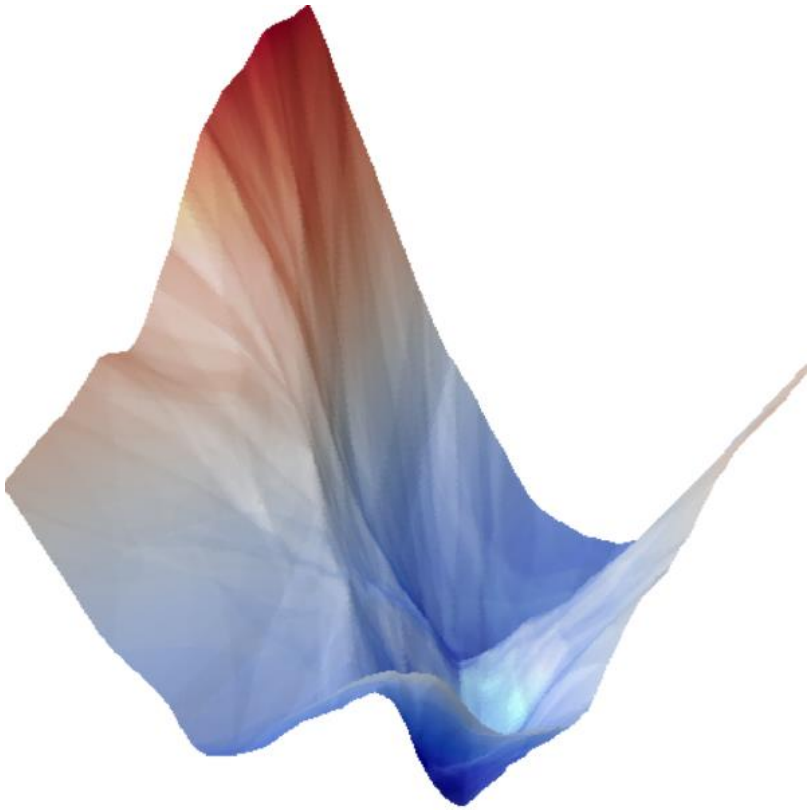
Approach #1

- 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



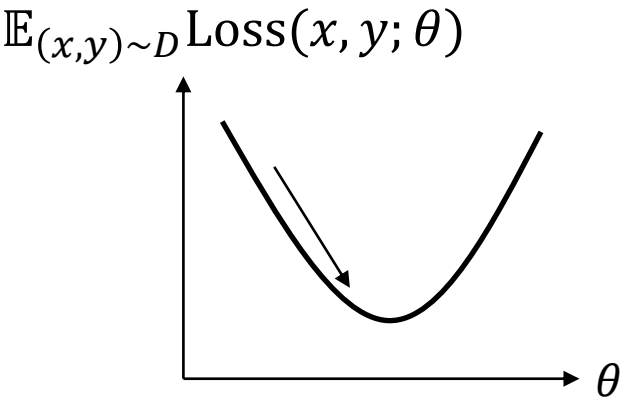
Local Search

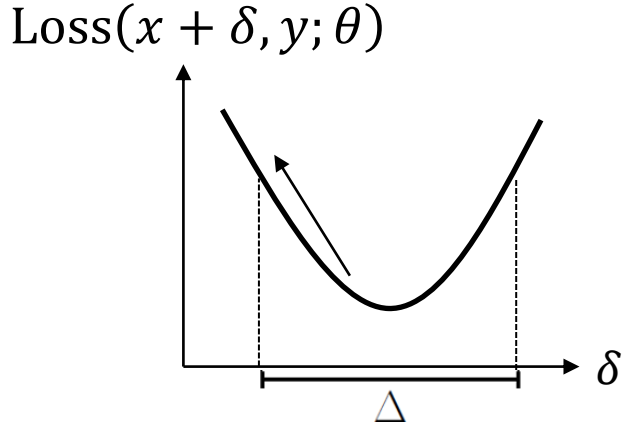
- The loss landscape of a NN is highly non-convex, inner maximization problem is difficult to solve exactly
- We can find an approximate solution using gradient-based methods, similar to deep learning training



NN Training vs. Local Search for Adversarial Input Generation

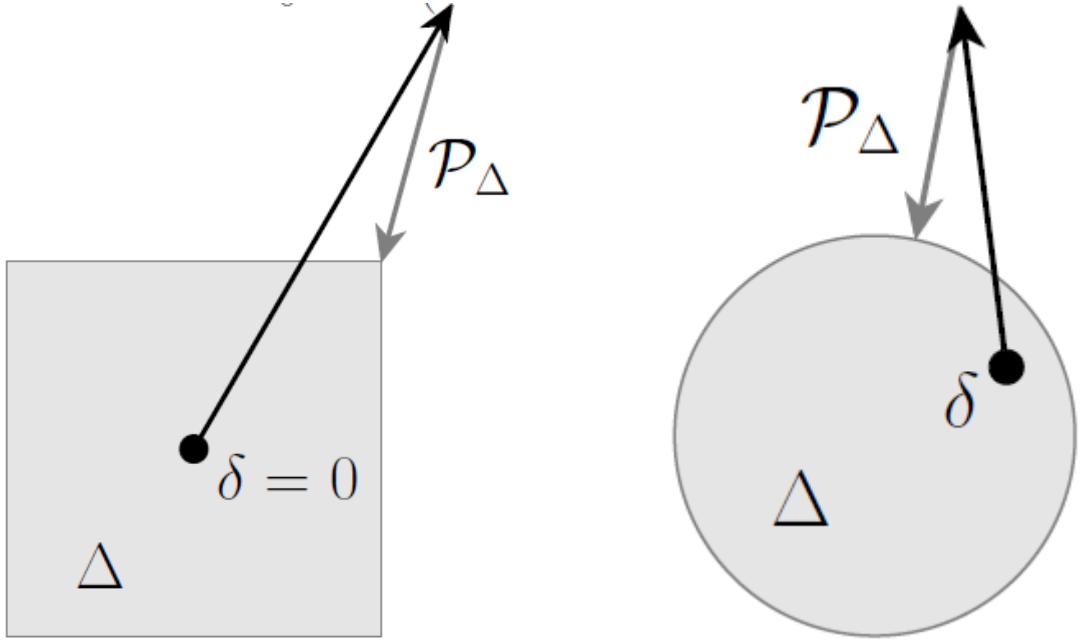
- To solve $\min_{\theta} \mathbb{E}_{(x,y)\sim D}\text{Loss}(x, y; \theta)$ for NN training: Gradient descent $\theta \leftarrow \theta - \alpha\nabla_{\theta}\text{Loss}(x, y; \theta)$
  - Update NN params θ by following the gradient downhill, in order to decrease $\text{Loss}(x, y; \theta)$. ($\alpha$ is the Learning Rate)
- To solve $\max_{\delta\in\Delta} \text{Loss}(x + \delta, y; \theta)$ for adversarial input generation: Gradient ascent $\delta \leftarrow \delta + \alpha\nabla_{\delta}\text{Loss}(x + \delta, y; \theta)$
  - Update input δ by following the gradient uphill, in order to increase $\text{Loss}(x + \delta, y; \theta)$, while ensuring $\delta \in \Delta$





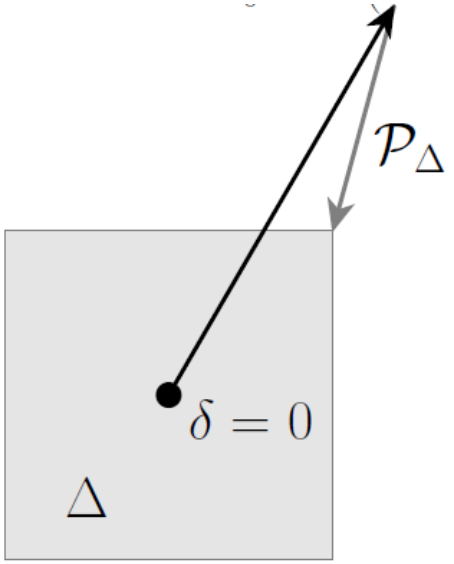
Projected Gradient Descent

- Take gradient step and project back into feasible set Δ: $\delta \leftarrow \mathcal{P}_{\Delta}\left(\delta + \alpha \nabla_{\delta} \mathrm{Loss}(x + \delta, y; \theta)\right)$
  - Lower left shows a gradient step followed by projection to the l_∞ ball: lower right shows projection to the l_2 ball



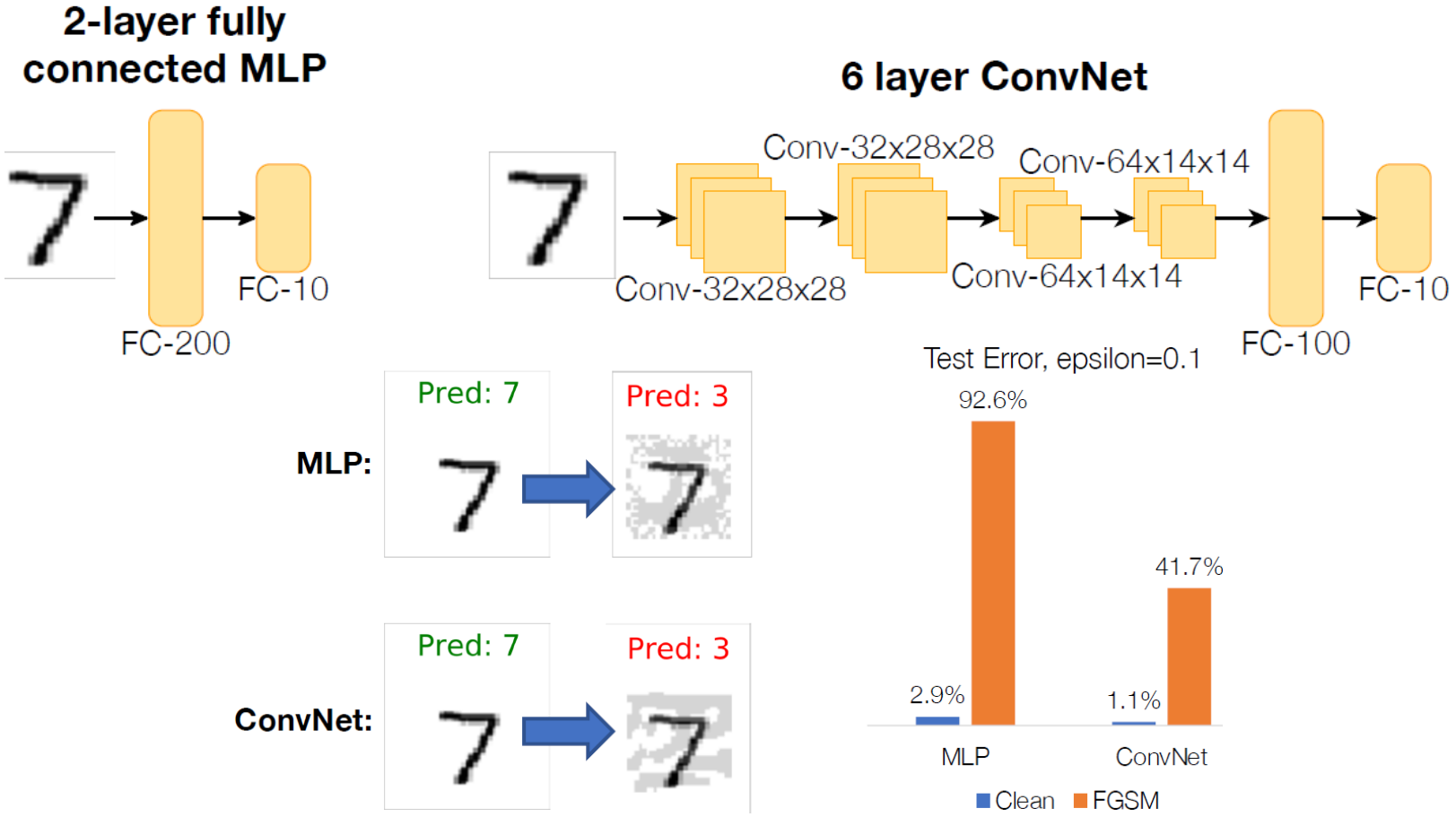
Fast Gradient Sign Method (FGSM)

- 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 $[-\epsilon, \epsilon]$: $\mathcal{P}_\Delta(\delta) \coloneqq \mathrm{Clip}(\delta, [-\epsilon, \epsilon])$
- In order to increase the loss as much as possible, take as large a step as possible in the gradient direction by making α very large. So the specific values of α and gradient do not matter; only the gradient direction matters. After clipping, we have: $\delta \coloneqq \epsilon \cdot \mathrm{sign}\big(\nabla_\delta \mathrm{Loss}(x + \delta, y; \theta)\big)$



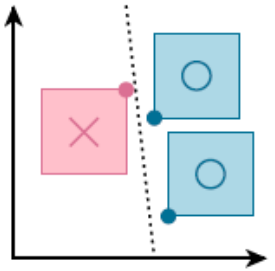
Adversarial Examples by FGSM

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



Comments on FGSM

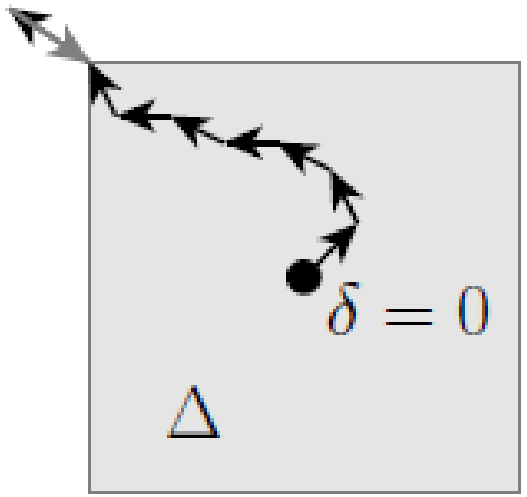
- FGSM is an attack specifically for l_∞ norm bound by taking a single PGD (Projected Gradient Descent) step under the l_∞ constraint, not for other norm bounds
- FGSM is the optimal attack against a linear binary classification model under the l_∞ norm bound.



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

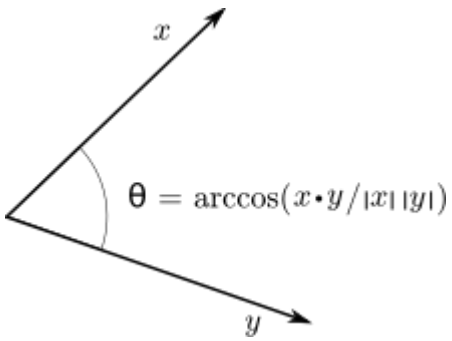
PGD w. Small Steps

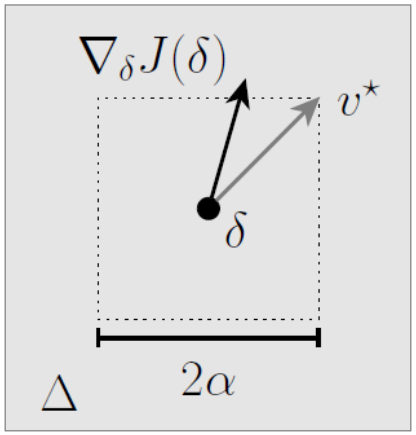
- Instead of taking a single large step as in FGSM, PGD takes many small steps to iteratively update δ:
  - Repeat: $\delta \leftarrow \mathcal{P}_{\Delta}\left(\delta + \alpha \nabla_{\delta} \mathrm{Loss}(x + \delta, y; \theta)\right)$
  - $\mathcal{P}_{\Delta}$ corresponds to clipping in the case of l_{∞} norm
- 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



Projected Steepest Descent

- PGD is highly sensitive to the absolute scale of the gradient, 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:
  - $\delta \leftarrow \mathcal{P}_{\Delta}\left(\delta + \underset{\mathrm{v}\leq\alpha}{\operatorname{argmax}}\, v^T \nabla_{\delta} J(\delta)\right)$
  - For l_{∞}, $\underset{\|v\|\leq\alpha}{\operatorname{argmax}}\, v^T \nabla_{\delta} J(\delta) = \alpha\, \operatorname{sign}(\nabla_{\delta} J(\delta))$
  - 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 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 $\|v\|_2 \leq \alpha$





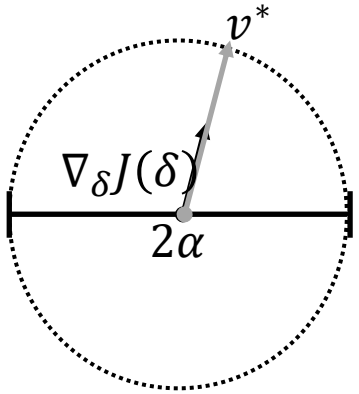
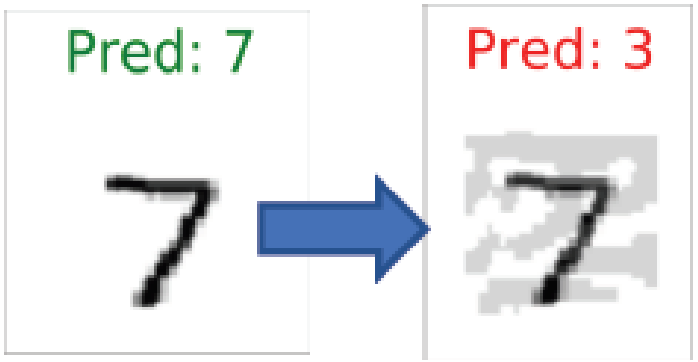
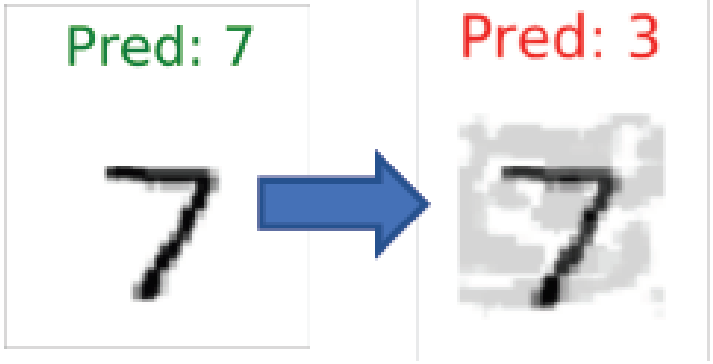


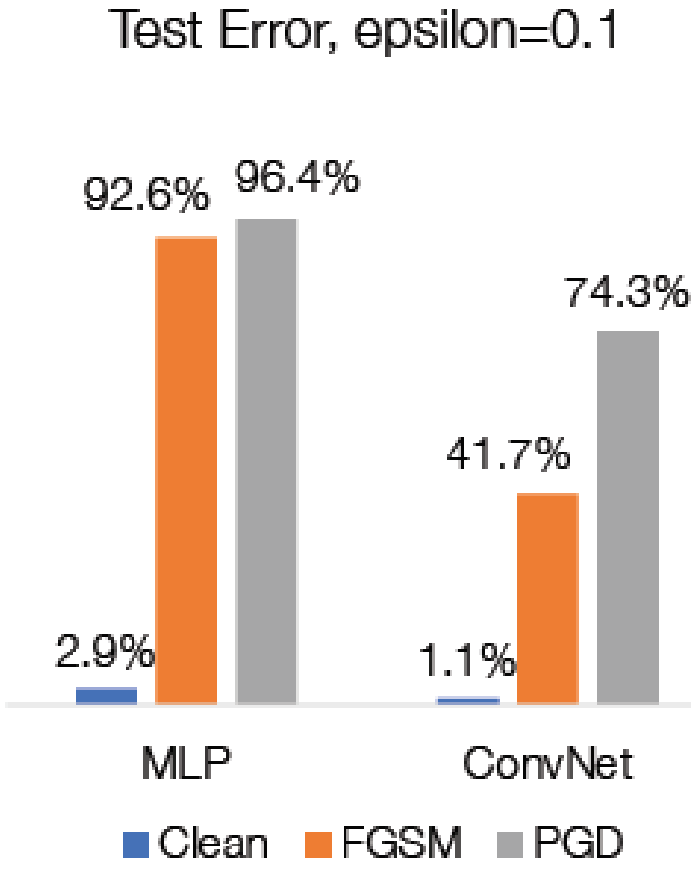
Illustration of PGD

ConvNet (FGSM):



ConvNet (PDG)



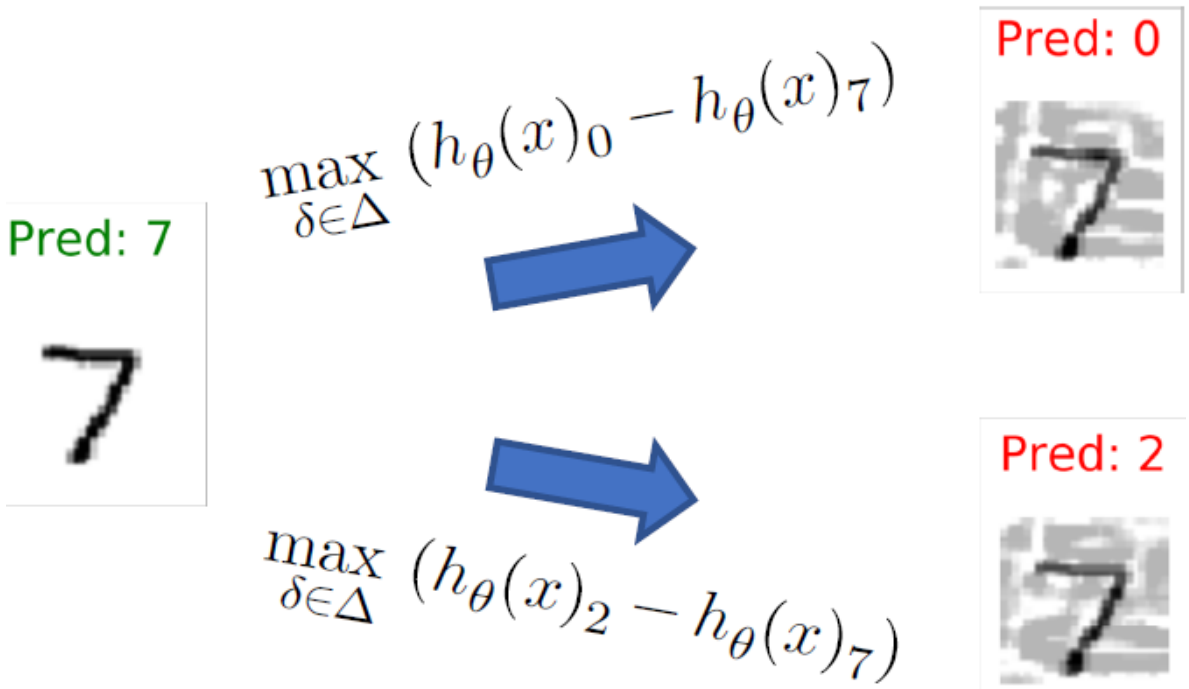


Targeted attacks

- Explicitly try to change label to a particular class y_{targ}:
  - $\max_{\|\delta\| \leq \Delta} (\text{Loss}(x+\delta, y; \theta) - \text{Loss}(x+\delta, y_{targ}; \theta)$
- Recall Cross-Entropy loss from "L3 Intro to ML":
  - $\text{Loss}(x+\delta, y; \theta) = \log\left(\sum_{j=1}^{k} \exp\left(h_\theta(x+\delta)_j\right)\right) - h_\theta(x+\delta)_y$
- Maximizing $\text{Loss}(h_\theta(x), y)$ amounts to minimizing the logit $\left(h_\theta(x)\right)_y$ corresponding to the correct label y:
  - $\min_{\delta \in \Delta} (h_\theta(x+\delta)_y - h_\theta(x+\delta)_{y_{targ}})$
  - where we try to minimize the logit of the true label y, and maximize the logit of target class y_{targ}
- An alternative is to maximize the logit of target class y_{targ} and minimize logits of all the other labels y':
  - $\min_{\delta \in \Delta} (\sum_{y' \neq y_{targ}} h_\theta(x+\delta)_{y'} - h_\theta(x+\delta)_{y_{targ}})$
- We are optimizing the logits before applying the SoftMax operator after the last linear layer. Since SoftMax is a monotonically function, removing it does not affect the solution of δ

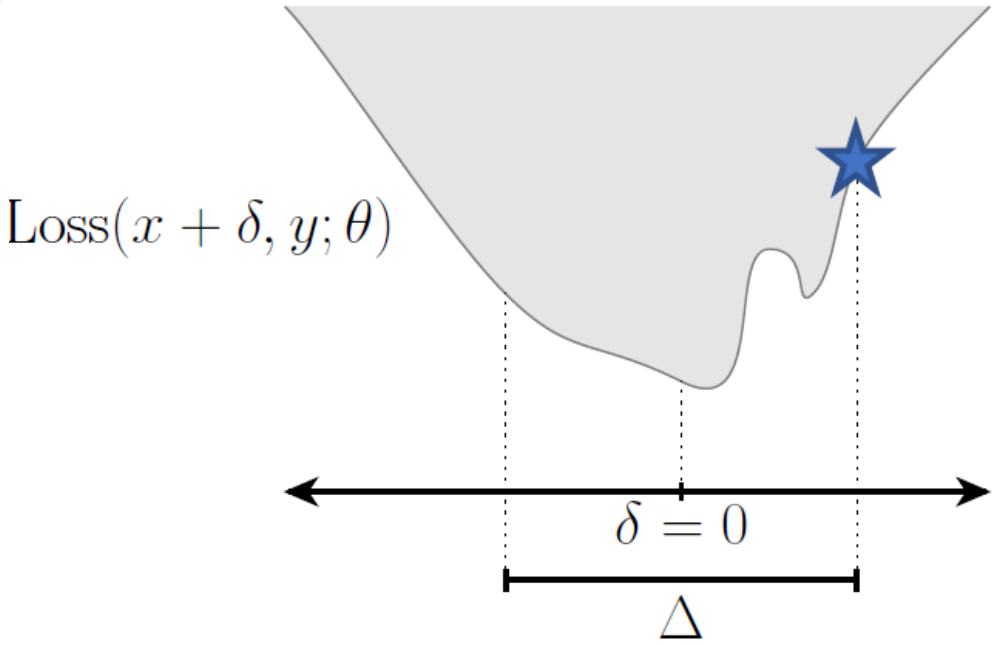
Targeted Attacks Examples

- Note: A targeted attack can succeed in “fooling” the classifier, but change to a different label than target



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:
  - $z_1 = x$
  - $z_{i+1} = \mathrm{ReLU}(W_i z_i + b_i), i = 1, \ldots, d-1$
  - $h_\theta(x) = z_{d+1} = W_d z_d + b_d$, where params $\theta = \{W_1, b_1, \ldots, W_d, b_d\}$
- Targeted attack in l_∞ norm:
  - $\min_{z_{1:d+1}} \left(e_y - e_{y_{targ}}\right)^T z_{d+1}$ s.t.
  - $z_{i+1} = \mathrm{ReLU}(W_i z_i + b_i), i = 1, \ldots, d-1$
  - $z_{d+1} = W_d z_d + b_d$
  - $\|z_1 - x\|_\infty \leq \epsilon$
  - 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:

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

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

$$z_{i+1} \geq W_i z_i + b_i$$
$$z_{i+1} \geq 0$$
$$u_i \geq z_{i+1}$$
$$W_i z_i + b_i \geq z_{i+1}$$

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

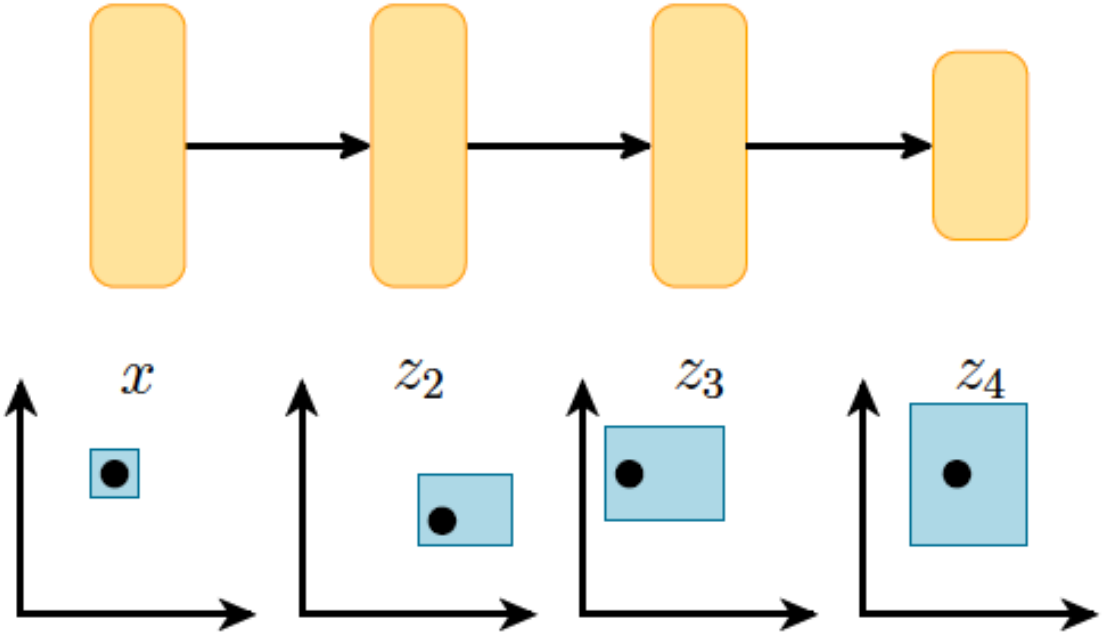
$W_i z_i + b_i < 0$
$\Rightarrow v_i = 0$

$$z_{i+1} \geq W_i z_i + b_i$$
$$z_{i+1} \geq 0$$
$$0 \geq z_{i+1}$$
$$W_i z_i + b_i \geq z_{i+1} + l_i$$

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

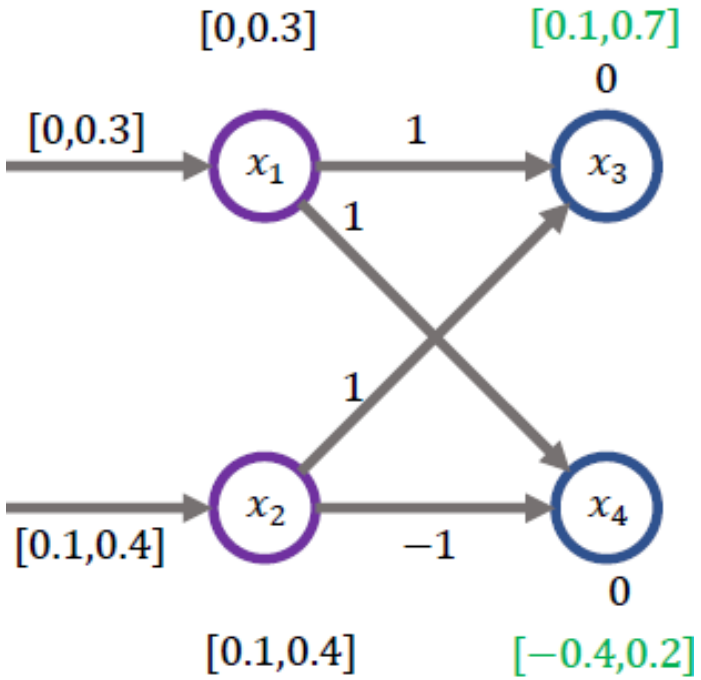
Bound Propagation

- 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 $[W]_+ l + [W]_- u + b \leq Wz + b \leq [W]_+ u + [W]_- l + b$
  - where $[W]_- \coloneqq \min\{W, 0\}, [W]_+ \coloneqq \max\{W, 0\}$.
  - 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

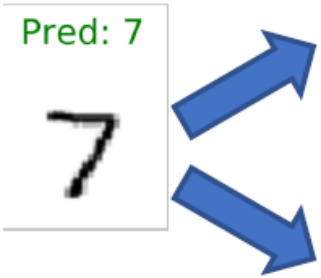


Final ILP Formulation

- $\min_{z_{1:d+1}, v_{1:d-1}} \left(e_y - e_{y_{targ}}\right)^T z_{d+1}$ s.t.
- (Encoding ReLU $z_{i+1} = \text{ReLU}(W_i z_i + b_i), i = 1, \ldots, d-1$)
  - $z_{i+1} \geq W_i z_i + b_i, i = 1, \ldots, d-1$
  - $z_{i+1} \geq 0, i = 1, \ldots, d-1$
  - $u_i \cdot v_i \geq z_{i+1}, i = 1, \ldots, d-1$
  - $W_i z_i + b_i \geq z_{i+1} + (1 - v_i) l_i, i = 1, \ldots, d-1$
  - $v_i \in \{0,1\}^{|v_i|}, i = 1, \ldots, d-1$
- (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



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

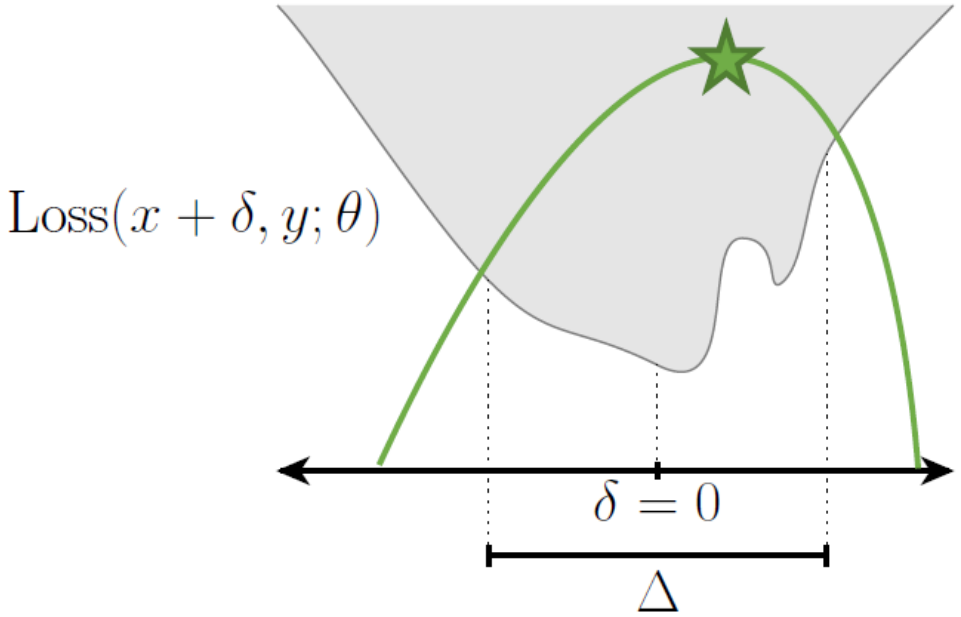
= -2.54 (exists adversarial example for target class zero or another class)

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

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

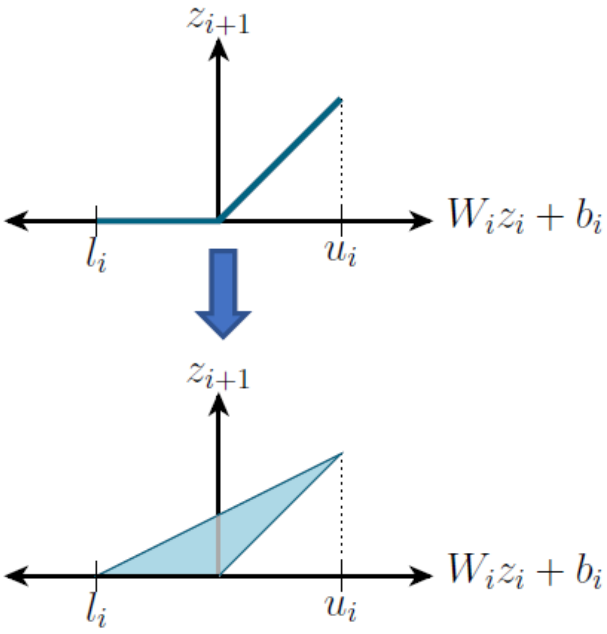
Approach #3

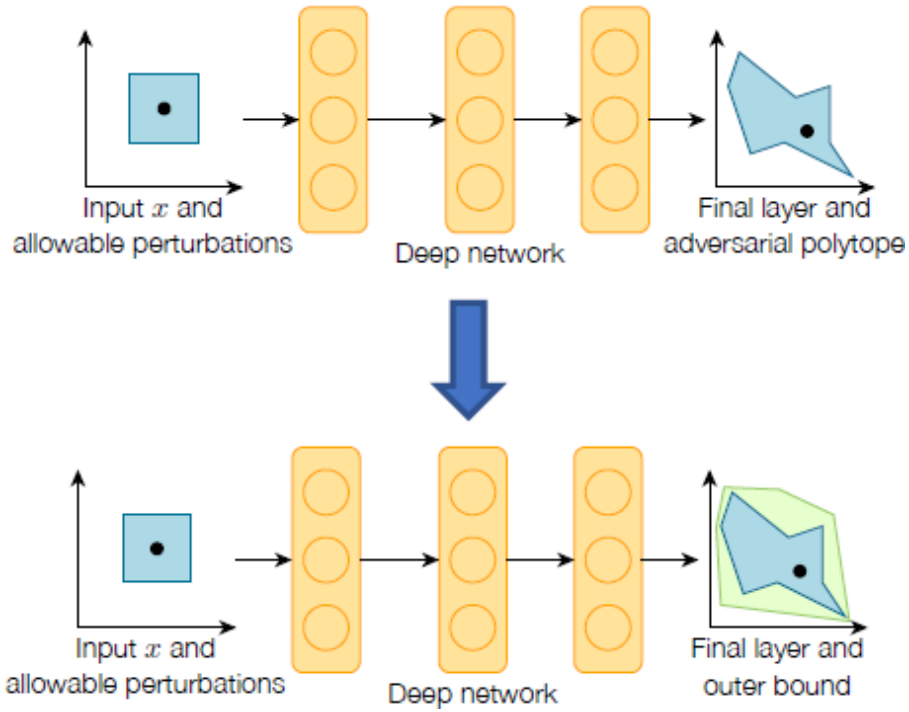
- To solve $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$:
- 1. Constructing adversarial examples via local search (
  - Lower bound on objective
- 2. Formal verification via combinatorial optimization
  - Exactly solve objective
- 3. Formal verification via convex relaxation
  - Upper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



Convex Relaxation

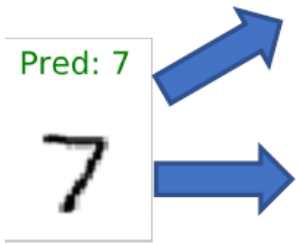
- For a ReLU-based NN, solving the ILP with $v_i \in \{0,1\}^{|v_i|}$ (the ReLU can be either off or on) is too computationally expensive
- Convex relaxation: replace the ReLU constraints with their convex hull $0 \leq v_i \leq 1$ (the ReLU can be partially off and partially on). Then optimization problem 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) $\text{Obj(LP)} \leq \text{Obj(ILP)}$ (refer to p. 17)
- If Objective(LP) is still positive for all target classes, the relaxation gives a verifiable proof that no adversarial example exists
- If Objective(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



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

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

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

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

Outer minimization:

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

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
   $$\delta^\star = \underset{\|\delta\| \leq \epsilon}{\operatorname{argmax}} \ell(h_\theta(x + \delta), y)$$
   b. Add gradient at $\delta^\star$
   $$g := g + \nabla_\theta \ell(h_\theta(x + \delta^\star), y)$$
3. Update parameters θ
   $$\theta := \theta - \frac{\alpha}{|B|} g$$

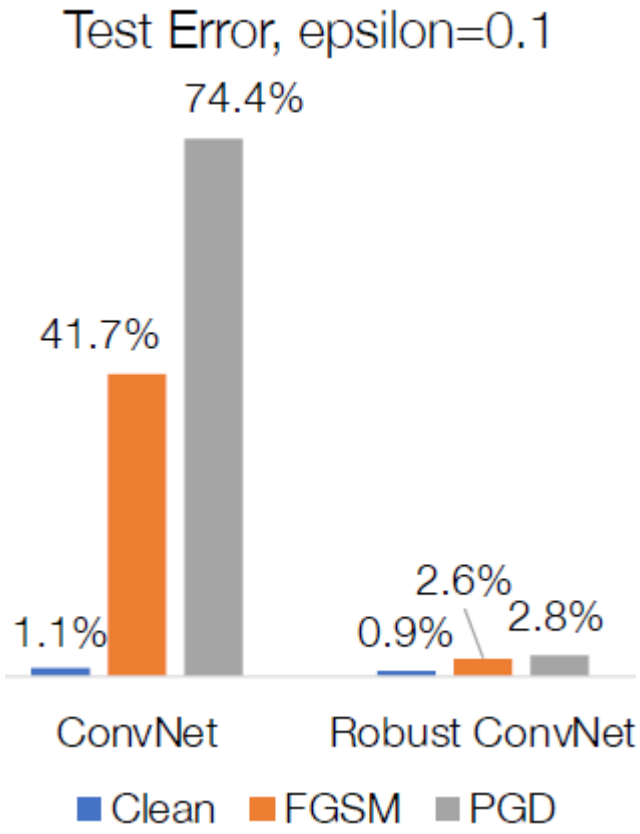
- In theory, Danskin's theorem only applies to the case where we are able to compute the maximum exactly. In practice, the quality of the robust gradient descent procedure is tied directly to how well we are able to perform the inner maximization. In other words, the key aspect is incorporate a strong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

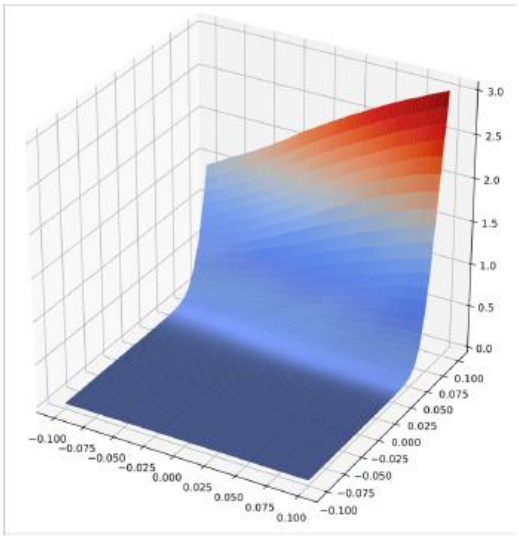
Evaluating Robust Models

- 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

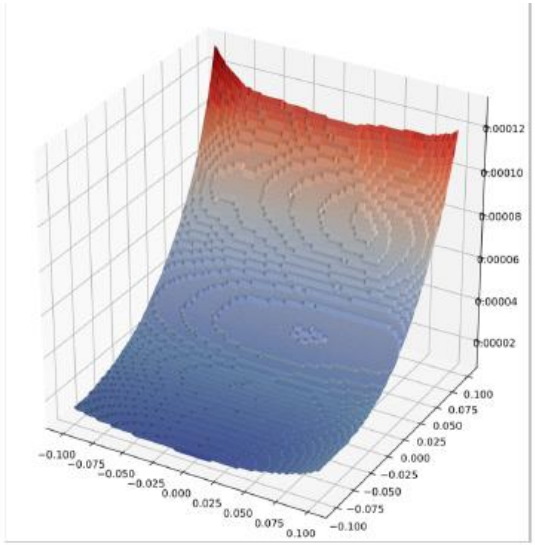
What Makes the Models Robust?

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





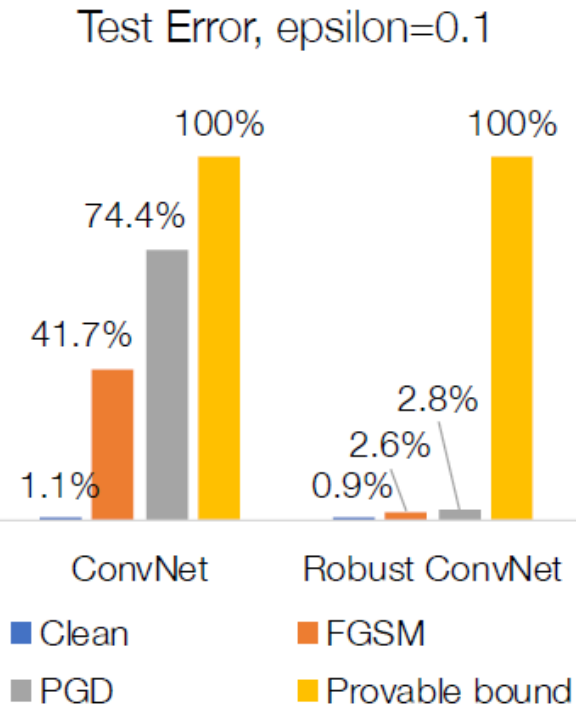
Loss surface:
standard training

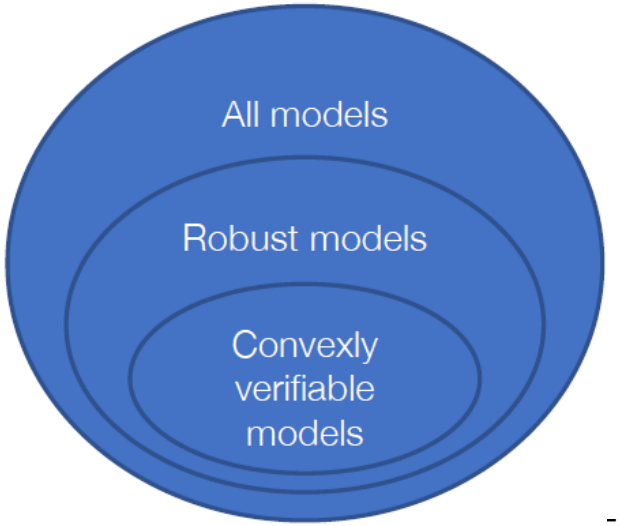


Loss surface:
robust training

Convex Methods for Certifying Robustness

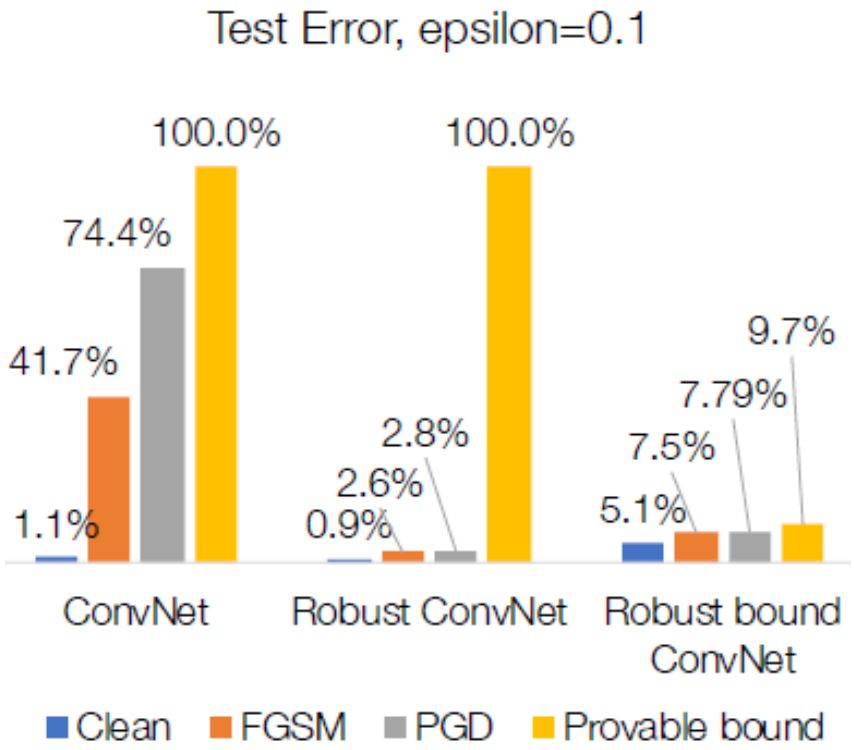
- Let's use the convex bounds to see what level of adversarial performance we can guarantee for the robust model
  - Test errors have provable bound of 100%, i.e., we can make no guarantees at all for this testset, even for the robust model
- Insight: models that can be (convexly) verified to be robust 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

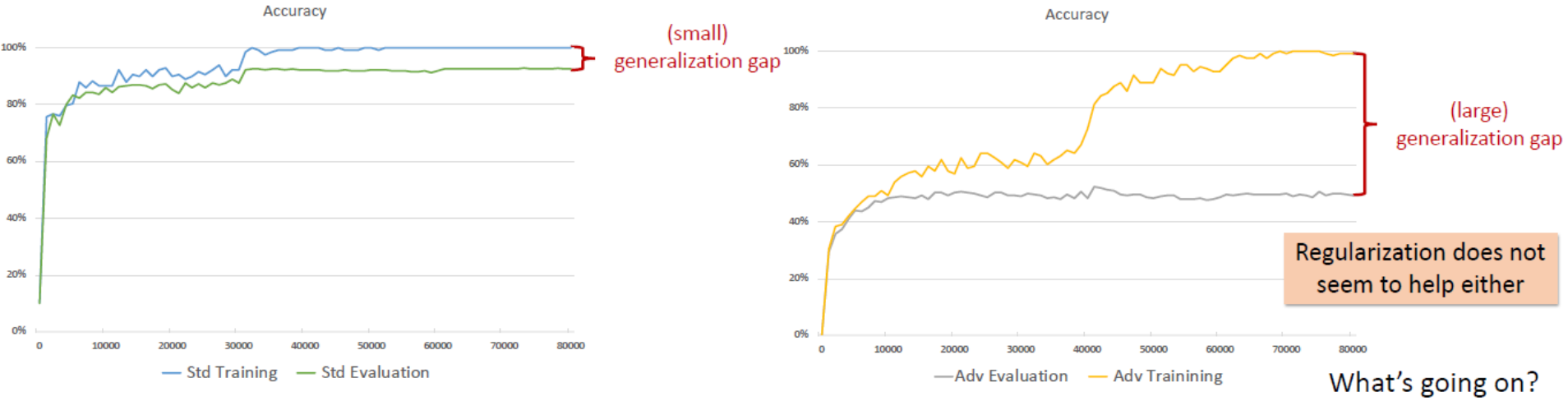
- Convex bounds are differentiable functions of network parameters
- Can train networks to minimize these bounds directly
- Recent work shows that interval bounds often outperform more complex strategies [Mirman et al, 2018, Gowal et al., 2018]



Outline

- Introduction
- Adversarial examples and verification
  - Constructing adversarial examples via local search
  - 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

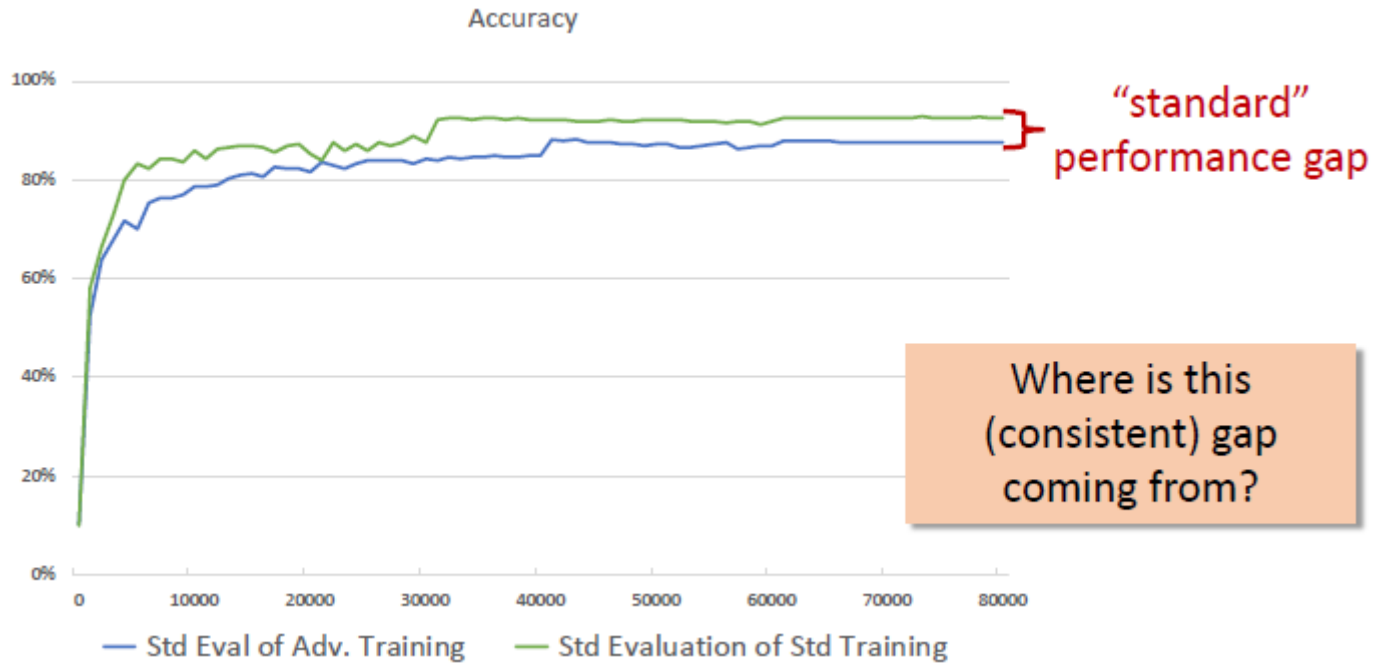
Adv. Robust Generalization Needs More Data to Avoid Overfitting



- 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

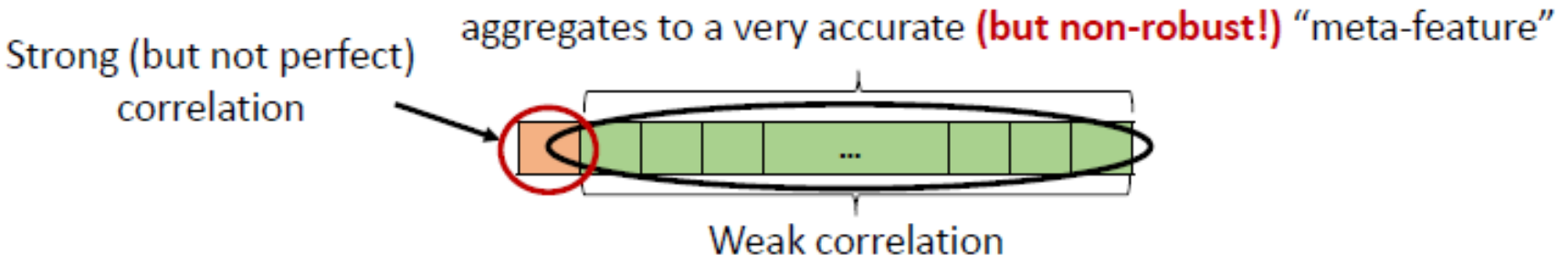
Does Being Robust Help “Standard” Generalization?

- 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



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



Standard training: use all of features, maximize accuracy

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

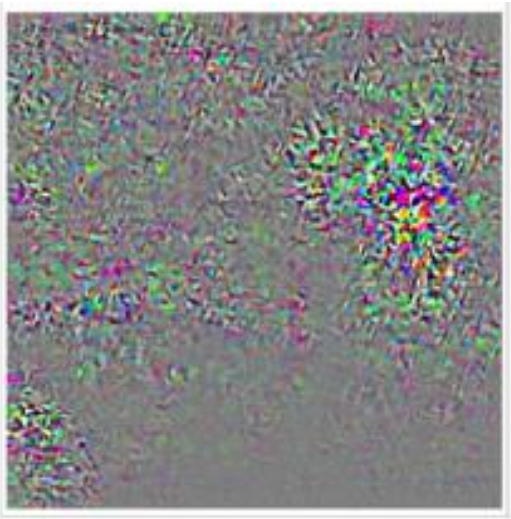
Adversarial Robustness is Not Free

- 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

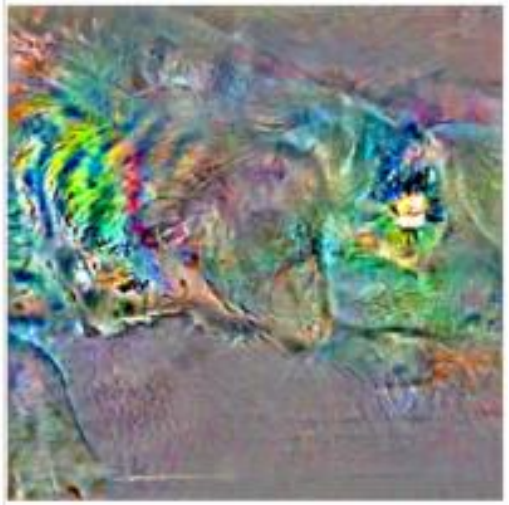
But There Are (Unexpected?) Benefits Too

- 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

Standard model

Attack

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