

Adversarial Robustness: Theory and Practice

Zico Kolter^{1,2}, Aleksander Madry³

¹Carnegie Mellon University

²Bosch Center for Artificial Intelligence

³Massachusetts Institute of Technology

<https://adversarial-ml-tutorial.org>

Outline

Introduction (Aleksander)

Adversarial examples and verification (Zico)

Training adversarially robust models (Zico)

Adversarial robustness beyond security (Aleksander)

Outline

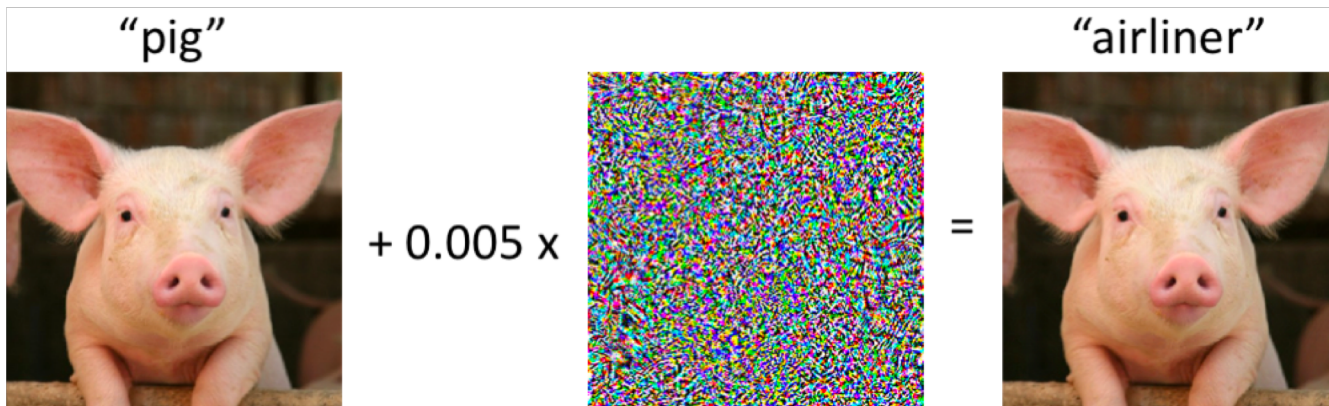
Adversarial examples and verification (Zico)

- Constructing adversarial examples
- Formal verification via combinatorial optimization
- Formal verification via convex relaxations

Training adversarially robust models (Zico)

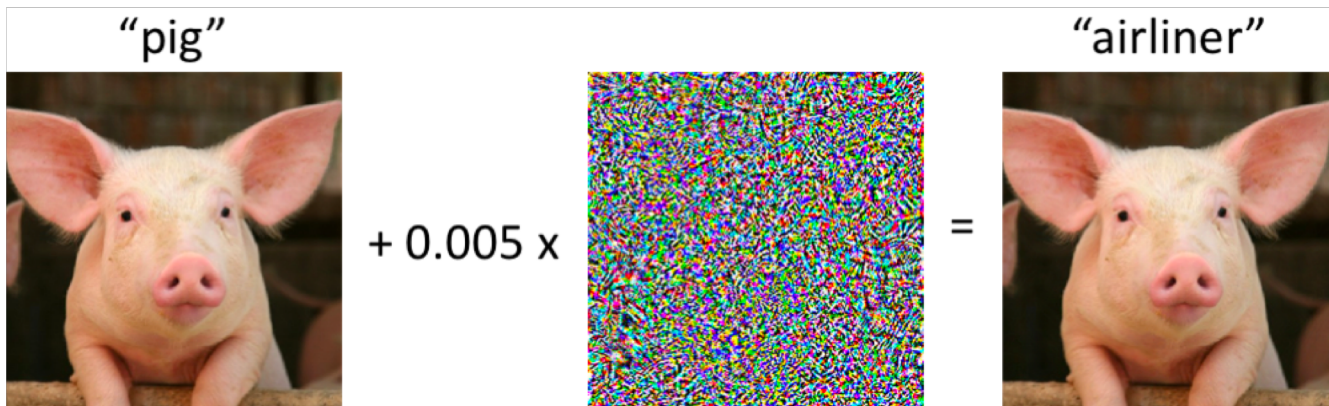
- Adversarial training
- Robust optimization with convex relaxations

The big picture



$$\min_{\theta} \mathbf{E}_{x,y} \left[\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta) \right]$$

The big picture



Part II: training a robust classifier

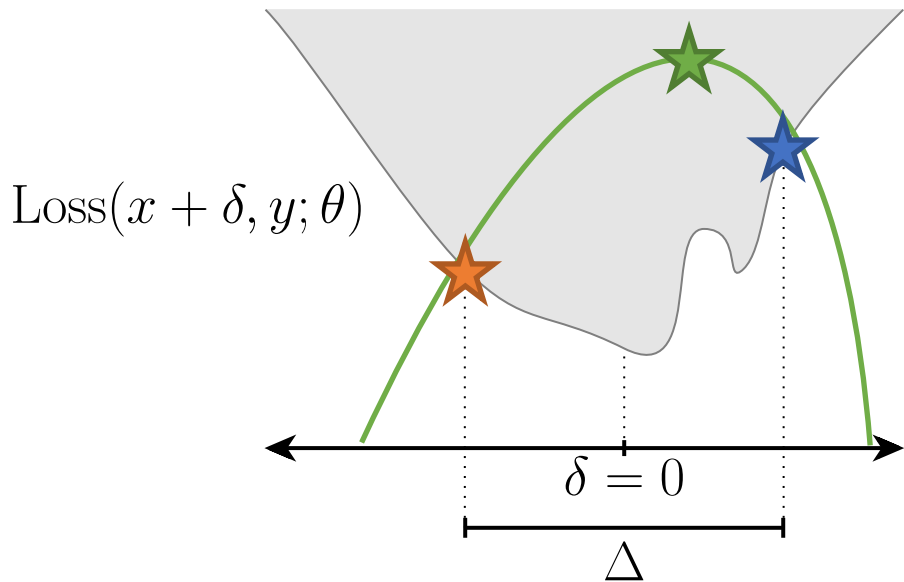
$$\min_{\theta} \sum_{x, y \in S} \max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$$

Part I: creating an adversarial example
(or ensuring one does not exist)

Part I: Adversarial examples and verification

The inner maximization problem

How do we solve the optimization?



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

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

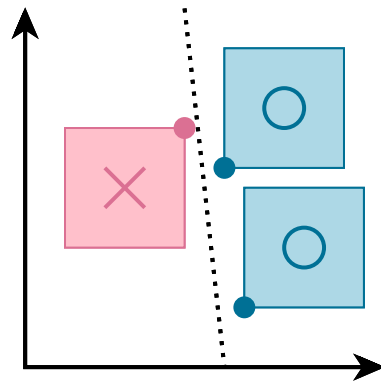
The linear case

Suppose our hypothesis is linear

$$\text{Loss}(x + \delta, y; \theta) = L(\theta^T x \cdot y)$$

(e.g. L hinge or logistic loss) and
perturbation region Δ is a norm ball
 $\Delta = \{\delta : \|\delta\| \leq \epsilon\}$

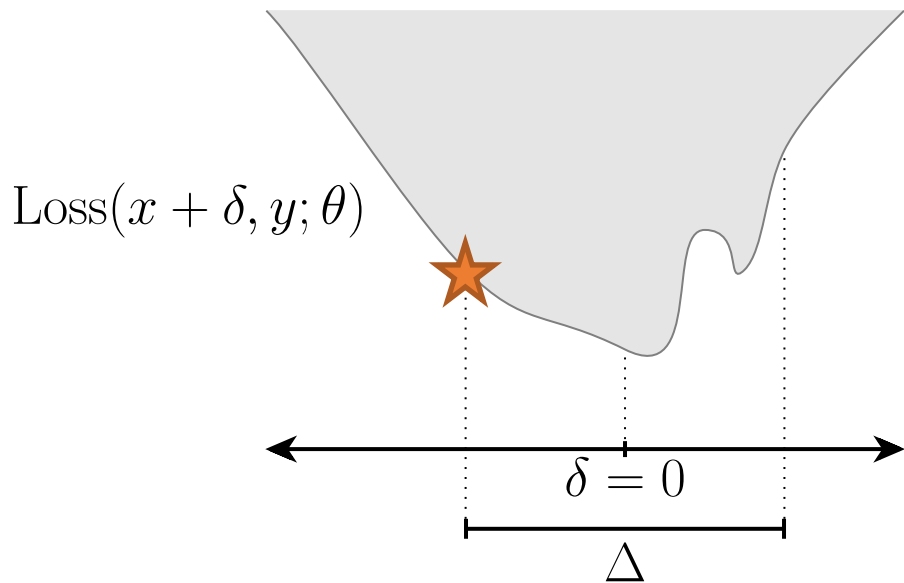
Then maximization has exact
solution based on dual norm; a
simple instance of *robust
optimization* [Stoyer, 1973, Ben-Tal et al.,
2011, Xu and Manor, 2009]



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

The inner maximization problem

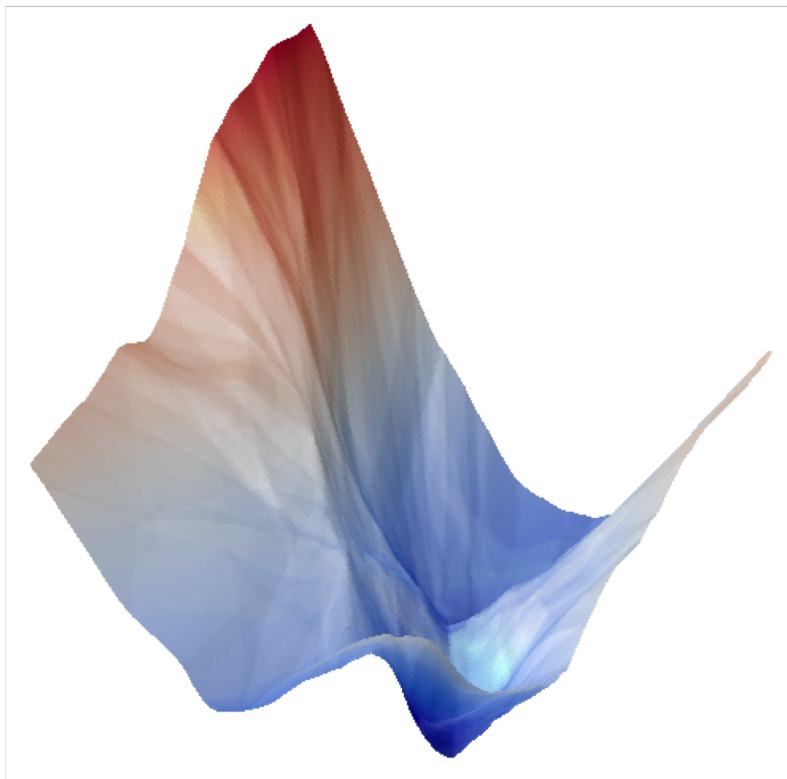
How do we solve the optimization?



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

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

Local search



For neural networks, the loss landscape is non-convex, inner maximization problem is difficult to solve exactly

This has never stopped us before in deep learning ... let's just find an approximate solution using gradient-based methods

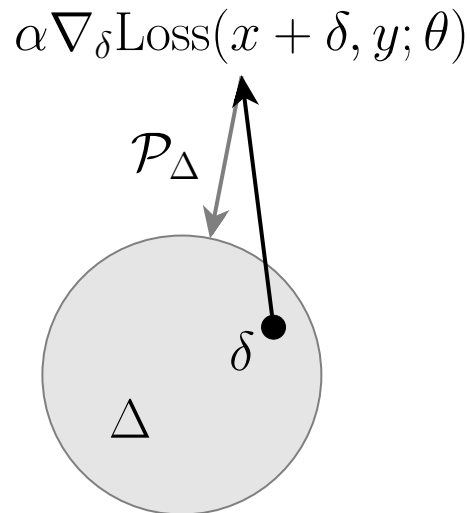
Projected gradient descent

Recall we are optimizing

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

We can employ a projected gradient descent method, take gradient step and project back into feasible set Δ

$$\delta := \mathcal{P}_{\Delta}[\delta + \nabla_{\delta} \text{Loss}(x + \delta, y; \theta)]$$



The Fast Gradient Sign Method (FGSM)

To be more concrete, take Δ to be the ℓ_∞ ball, $\Delta = \{\delta: \|\delta\|_\infty \leq \epsilon\}$, so projection takes the form

$$P_\Delta(\delta) = \text{Clip}(\delta, [-\epsilon, \epsilon])$$

As $\alpha \rightarrow \infty$, we always reach “corner” of the box, called fast gradient sign method (FGSM)

[Goodfellow et al., 2014]

$$\delta = \epsilon \cdot \text{sign}(\nabla_\delta \text{Loss}(x + \delta, y; \theta))$$

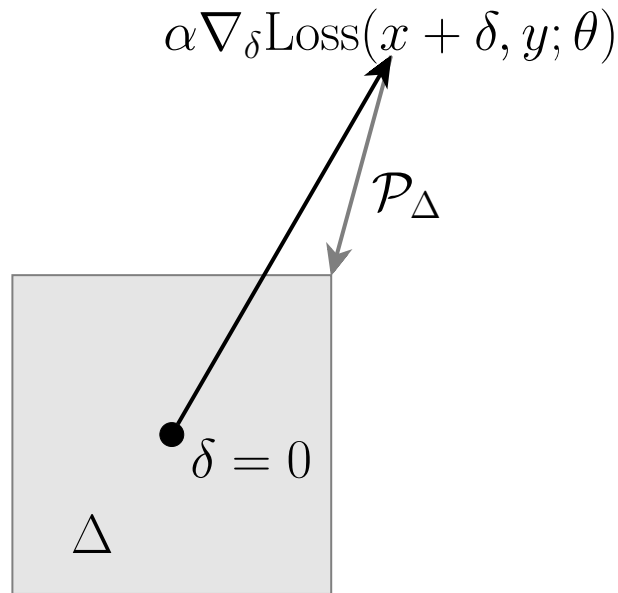
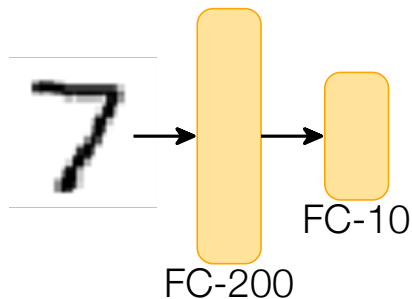


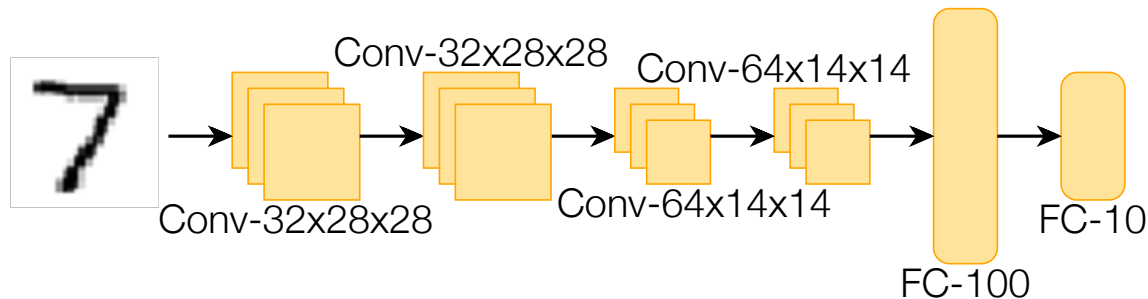
Illustration of adversarial examples

Will apologies to everyone, you are going to see MNIST examples in the tutorial (yes, in 2018) ... it is the best dataset for demonstrating some of the more computationally intensive methods

**2-layer fully
connected MLP**



6 layer ConvNet

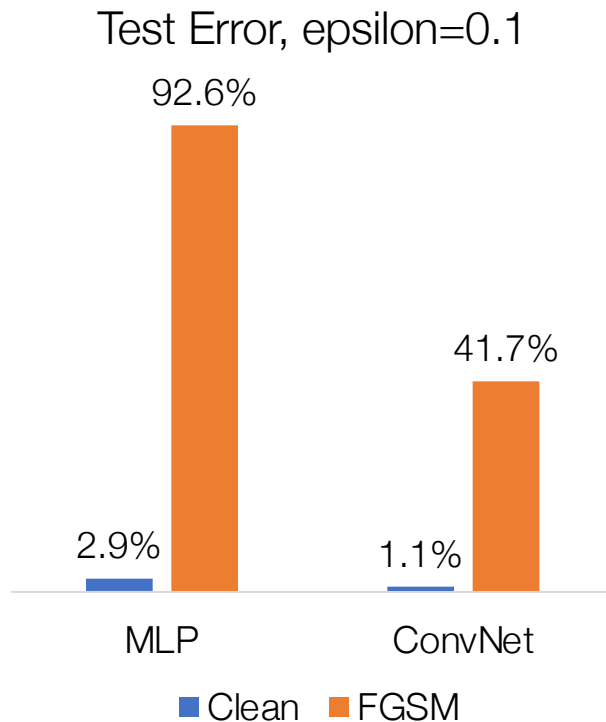
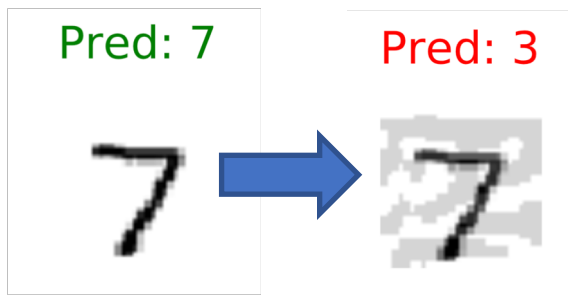


Illustrations of FGSM

MLP:



ConvNet:

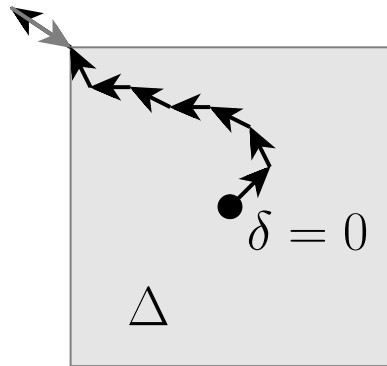


Projected gradient descent

Projected gradient descent applied to ℓ_∞ ball, repeat:

$$\delta := \text{Clip}_\epsilon[\delta + \alpha \nabla_\delta J(\delta)]$$

Slower than FGSM (requires multiple iterations), but typically able to find better optima



Aside: Steepest descent

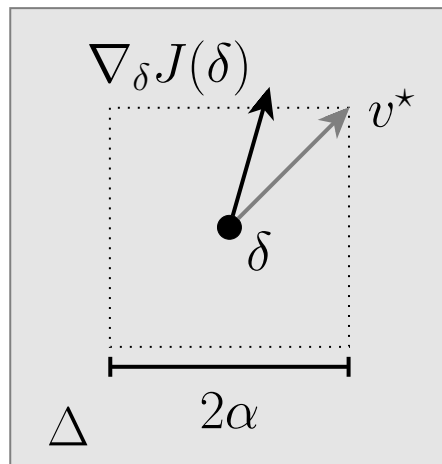
Gradients often small exactly at data points, so don't use “standard” PGD

(Unnormalized) proj. steepest descent

$$\delta := \mathcal{P}_{\Delta} \left[\delta + \operatorname{argmax}_{\|v\| \leq \alpha} v^T \nabla_{\delta} J(\delta) \right]$$

E.g. for ℓ_{∞} (typical to choose inner norm the same as the Δ constraint):

$$\operatorname{argmax}_{\|v\| \leq \alpha} v^T \nabla_{\delta} J(\delta) = \alpha \cdot \operatorname{sign}(\nabla_{\delta} J(\delta))$$

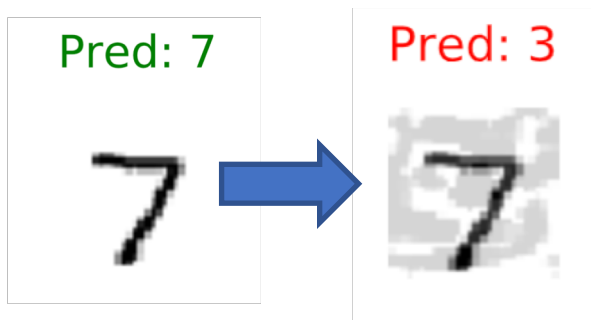


Illustrations of PGD

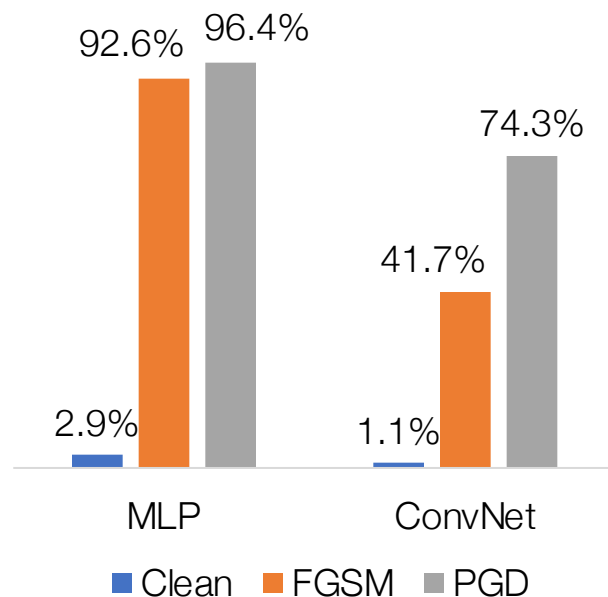
**ConvNet
(FGSM):**



**ConvNet
(PDG)**



Test Error, epsilon=0.1



Targeted attacks

Also possible to explicitly try to change label to a *particular* class

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

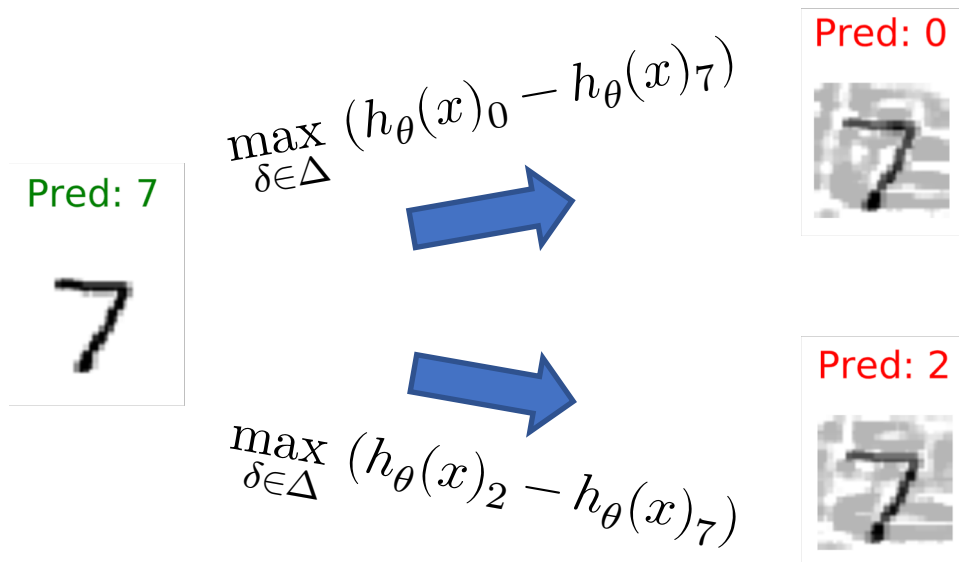
Consider multi-class cross entropy loss

$$\text{Loss}(x + \delta, y; \theta) = \log \sum_i \exp h_{\theta}(x + \delta)_i - h_{\theta}(x)_y$$

Then note that above problem simplifies to

$$\max_{\delta \in \Delta} \left(h_{\theta}(x)_{y_{\text{targ}}} - h_{\theta}(x)_y \right)$$

Targeted attack examples














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

Non- ℓ_∞ norms













Everything we have done with ℓ_∞ norm possible with other ℓ_p norms

E.g., derive steepest descent for ℓ_2 ball (i.e., normalized gradient descent), projection onto ball

ℓ_∞ :

Pred: 3	Pred: 2	Pred: 7	Pred: 0	Pred: 9	Pred: 7
					
Pred: 8	Pred: 3	Pred: 6	Pred: 7	Pred: 0	Pred: 6
					

ℓ_2 :

Pred: 3	Pred: 1	Pred: 7	Pred: 0	Pred: 9	Pred: 7
					
Pred: 8	Pred: 3	Pred: 6	Pred: 7	Pred: 0	Pred: 0
					

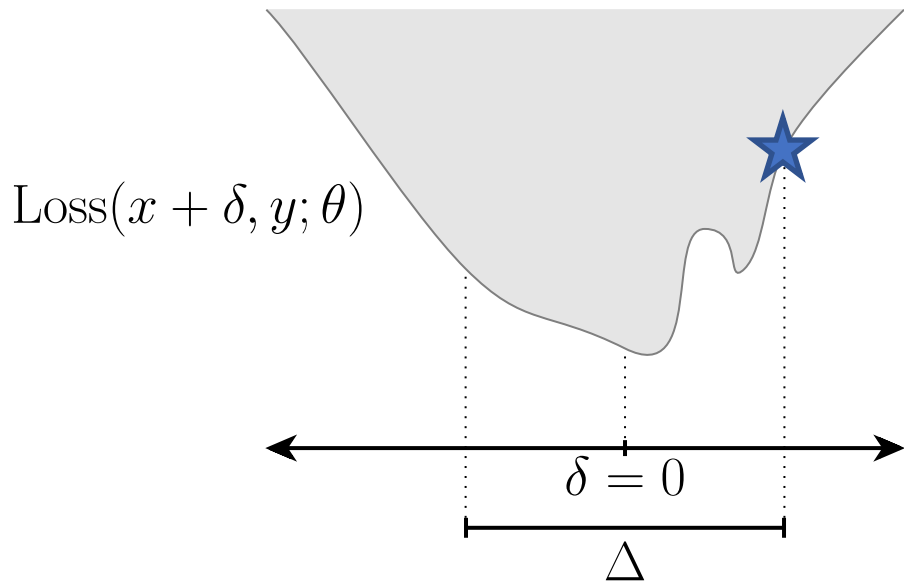
About all those other attacks ...

With apologies to the many authors that have written papers on different attacks, most of these are variants of PGD for different norm bounds

Our belief: at this point in the field, it is better to describe the attack in terms of 1) the allowable perturbation set Δ ; 2) the optimization procedure used to perform the maximization

The inner maximization problem

How do we solve the optimization?



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

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

Exact combinatorial optimization

To describe exact combinatorial optimization procedure, we need to more explicitly describe the network; consider a ReLU-based feedforward net

$$\begin{aligned} z_1 &= x \\ z_{i+1} &= \text{ReLU}(W_i z_i + b_i), \quad i = 1, \dots, d-1 \\ h_\theta(x) &= W_d z_d + b_d \end{aligned}$$

Targeted attack in ℓ_∞ norm can be written as the optimization problem

$$\begin{aligned} &\underset{z_{1:d}}{\text{minimize}} && \left(e_y - e_{y_{\text{targ}}} \right)^T (W_d z_d + b_d) \\ &\text{subject to} && z_{i+1} = \text{ReLU}(W_i z_i + b_i), \quad i = 1, \dots, d-1 \\ &&& \|z_1 - x\|_\infty \leq \epsilon \end{aligned}$$

Solving combinatorial problem

The optimization formulation of an adversarial attack can be written as an binary mixed integer program [e.g., Tjeng et al., 2018, Wong and Kolter, 2018] or as a satisfiability modulo theories (SMT) problem [e.g. Katz et al., 2017]

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 $l_i \leq W_i z_i + b_i \leq u_i$

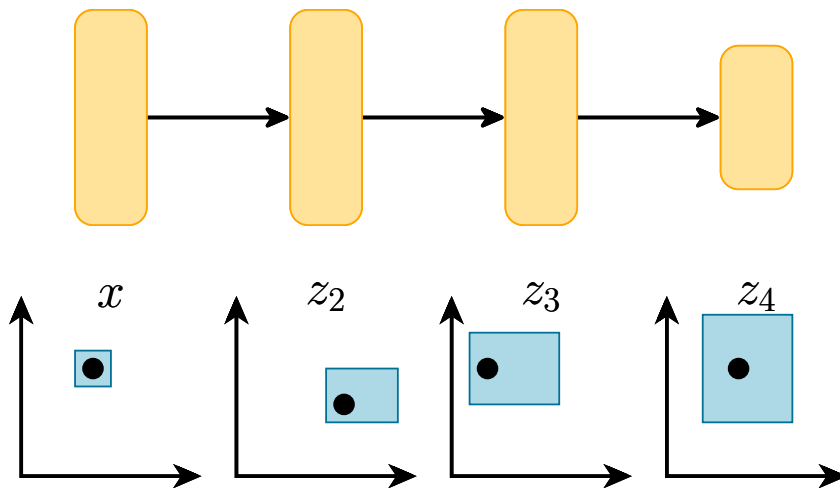
Bound propagation

How do we get the bounds l_i, u_i ? In general, if $l \leq z \leq u$, then

$$[W]_+l - [W]_-u + b \leq Wz + b \leq [W]_+u - [W]_-l + b$$

Very loose in general, but
still useful for IP

Will return when we discuss
convex relaxations



Certifying robustness

Consider the objective of our optimization problem

$$\left(e_y - e_{y_{\text{targ}}}\right)^T (W_d z_d + b_d)$$

If we solve our integer program for some y_{targ} and the objective is *positive*, then this is a *certificate* that there exists *no* adversarial example for that target class

If objective is positive for all $y_{\text{targ}} \neq y$, this is a *verified proof* that there exists no adversarial example at all

Certifying examples

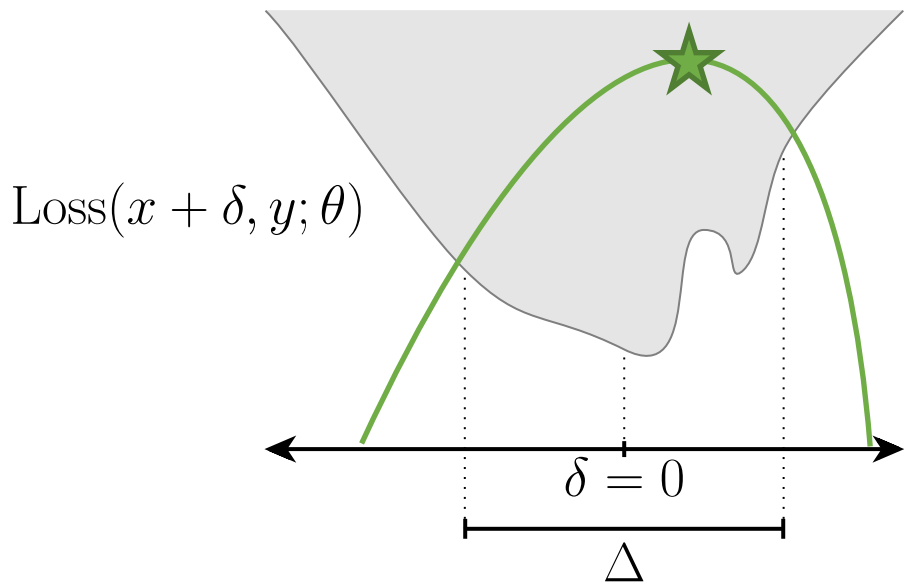


$$\begin{aligned} \min & (e_7 - e_0)^T (W_d z_d + b_d) &= -2.54 \text{ (exists adversarial} \\ \text{s. t. } & \dots & \text{example for target class zero} \\ & & \text{or another class)} \end{aligned}$$

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

The inner maximization problem

How do we solve the optimization?



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

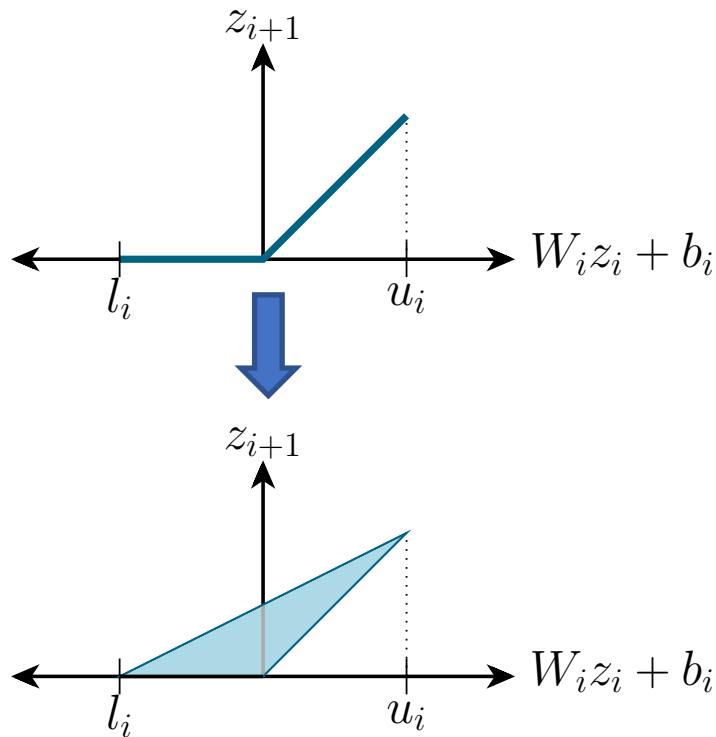
1. Local search (lower bound on objective)
2. Combinatorial optimization (exactly solve objective)
3. **Convex relaxation (upper bound on objective)**

Convex relaxation of the integer program

Solving the integer program is too computationally expensive, so let's consider a *convex relaxation*

Replace the bounded ReLU constraints with their convex hull

Optimization problem becomes a *linear program*

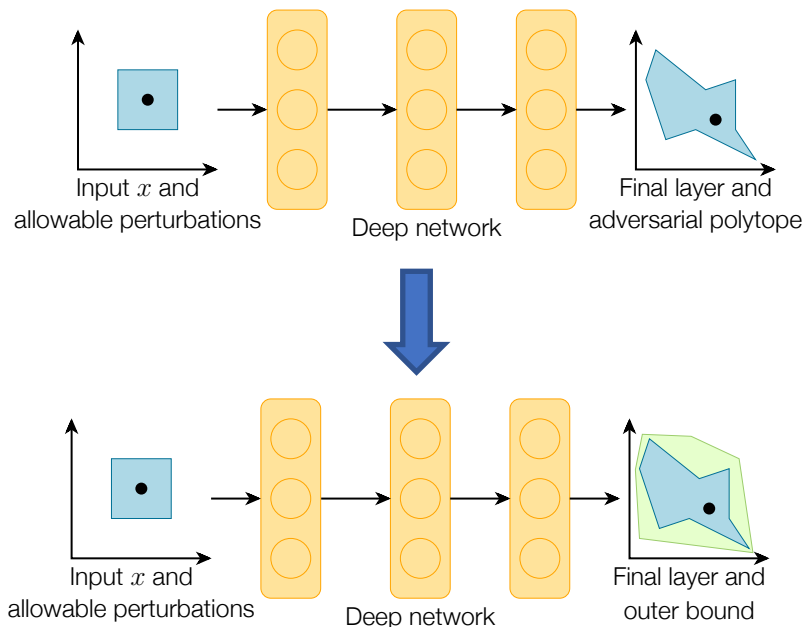


Note on the convex relaxation

Convex relaxation provides a strict *lower bound* on integer programming objective (because feasible set is larger)

$$\text{Objective}(\text{LP}) \leq \text{Objective}(\text{IP})$$

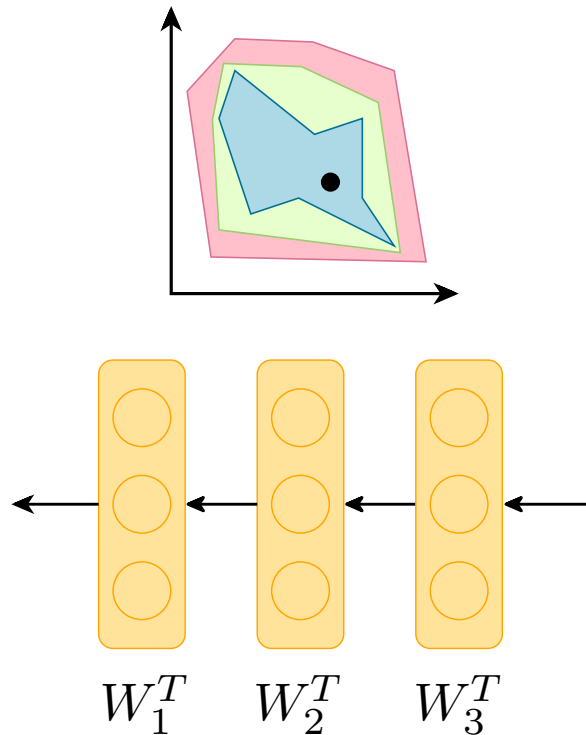
So if the objective of LP is still positive for all target classes, the relaxation gives a verifiable proof that no adversarial example exists



Fast solutions to the relaxation

Solving a linear program with size equal to the number of hidden units in the network (once per example), is still not particularly efficient

Using linear programming duality, it is possible to achieve a lower bound on the LP program, via a single backward pass through the network [Wong and Kolter, 2018]



Interval-based bounds

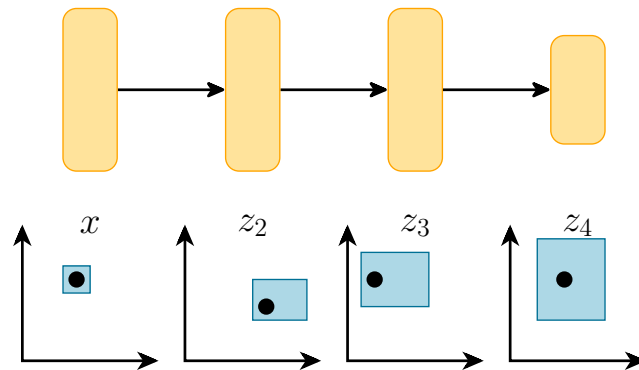
We can formulate optimization *only* considering bound constraints

$$\begin{array}{ll} \min_{z_d} & c^T (W_d z_d + b_d) \\ \text{s. t.} & l \leq z_d \leq u \end{array}$$

Has the analytical solution:

$$[c^T W_d]_+ \ell - [c^T W_d]_- u + c^T b_d$$

Even looser than previous bound
(but very fast to compute)



Certifying examples with convex bounds



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

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

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

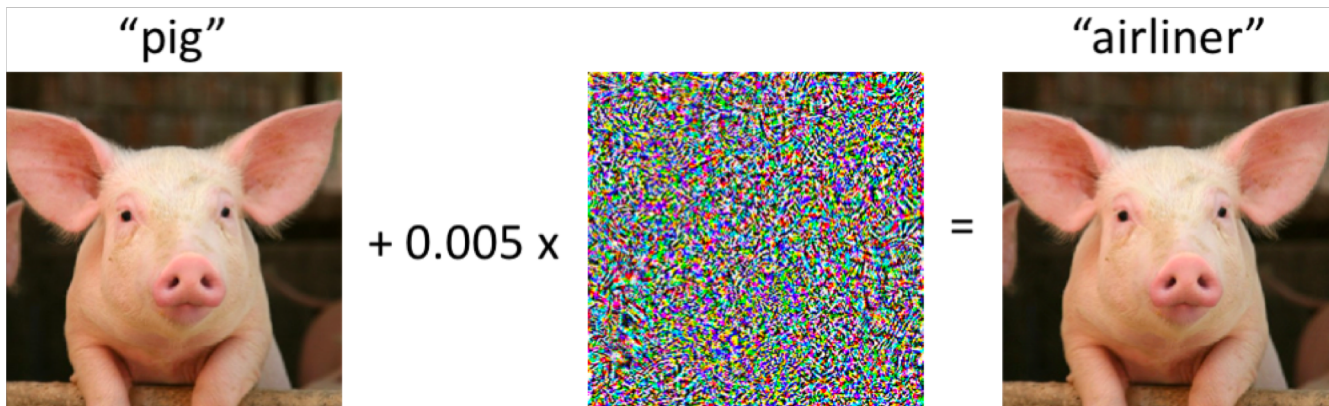
= -6.28 (*may or may not exist* adversarial example)

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

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

Part II: Training adversarially robust models

The big picture



Part II: training a robust classifier

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

Part I: creating an adversarial example
(or ensuring one does not exist)

The outer minimization problem

Inner maximization:

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

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

Outer minimization:

1. Adversarial training
3. Provably robust training

Adversarial training

How do we optimize the objective

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

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

A fundamental result in optimization:

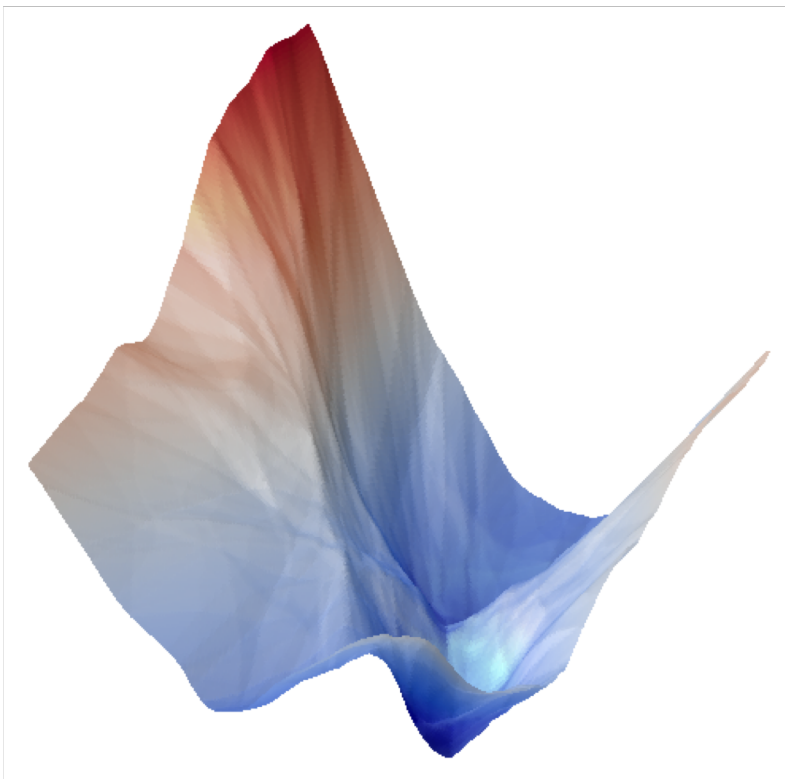
$$\nabla_{\theta} \max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta) = \nabla_{\theta} \text{Loss}(x + \delta^*, y; \theta)$$

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

Seems “obvious,” but it is a very subtle result; means we can optimize through the max by just finding it's maximizing value

Note however, it *only* applies when max is performed exactly

Deep learning to the rescue



Let's just ignore that technicality, and proceed as if we were solving the maximization exactly!

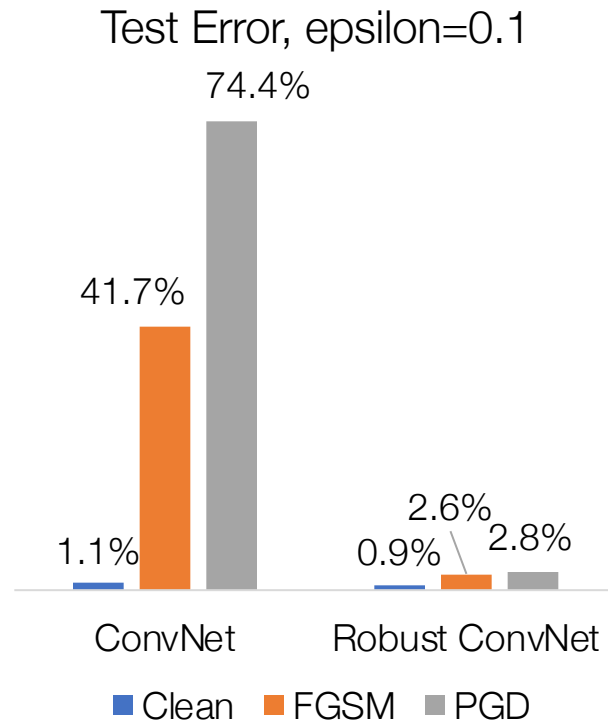
Adversarial training [Goodfellow et al., 2014]

Repeat

1. Select minibatch B
2. For each $(x, y) \in B$, compute adversarial example $\delta^*(x)$
3. Update parameters

$$\theta := \theta - \frac{\alpha}{|B|} \sum_{x, y \in B} \nabla_{\theta} \text{Loss}(x + \delta^*(x), y; \theta)$$

Common to also mix robust/standard updates (not done in our case)



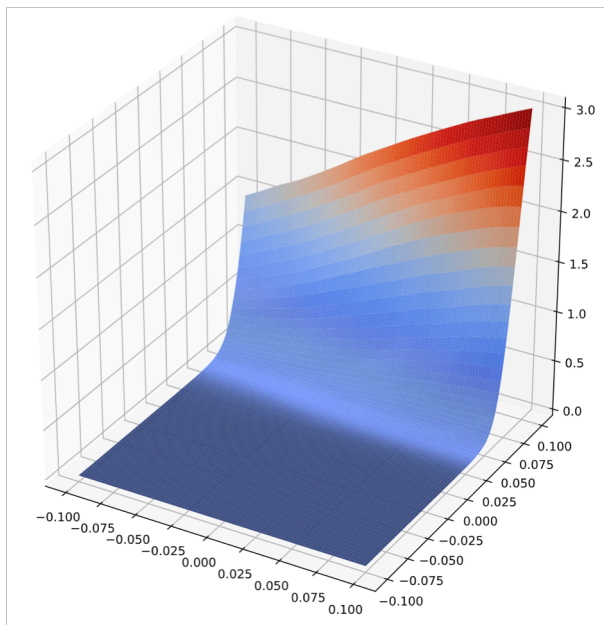
Evaluating robust models

Our model looks good, but we should be careful declaring success

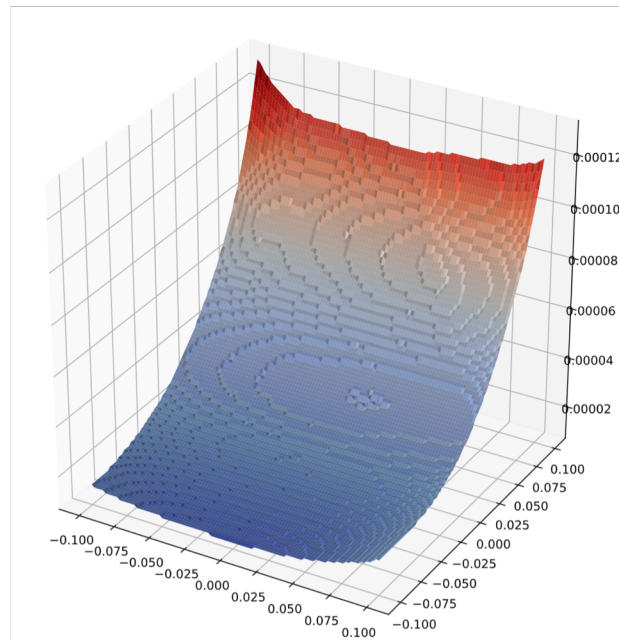
Need to evaluate against different attacks, PGD attacks run for longer, with random restarts, etc

Note: it is *not* particularly 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?



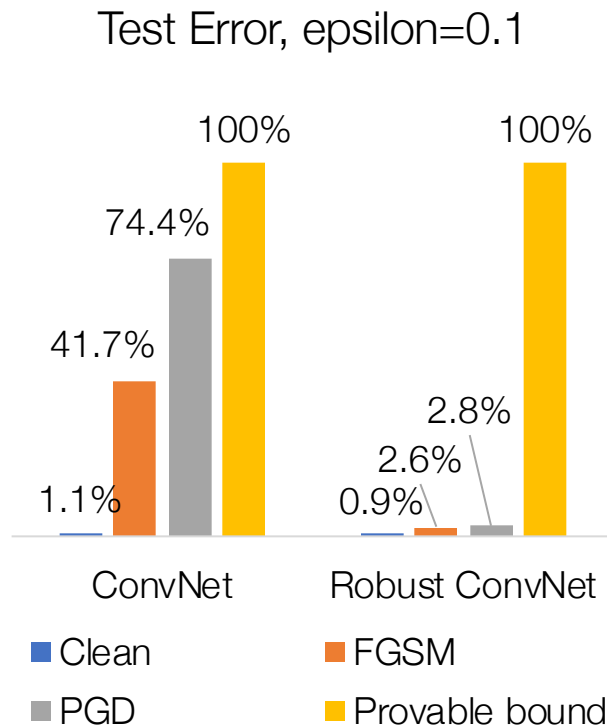
Loss surface:
standard training



Loss surface:
robust training

Convex methods for certifying robustness

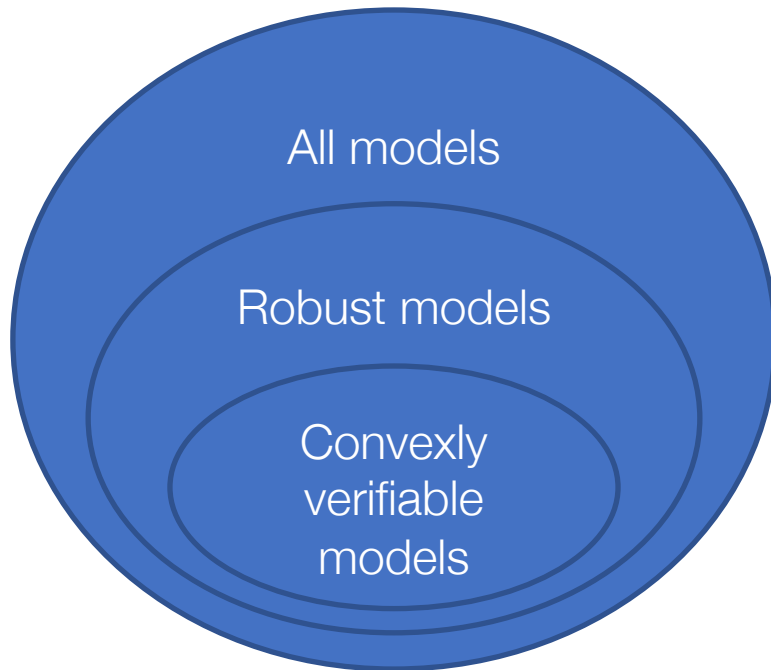
Let's use the convex bounds (in this case, interval-bound-based approach) to see what level of adversarial performance we can *guarantee* for the robust model



Convexly verifiable models

Key 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*

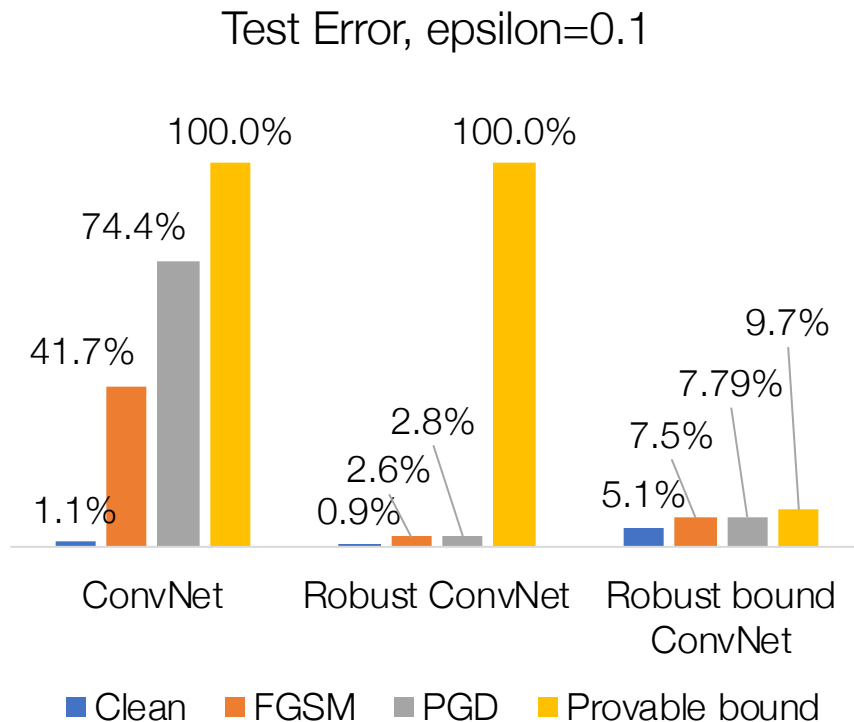


Training provably robust models

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]



A word of caution...

Despite the “good” results we saw in this tutorial, this is more a function MNIST than representative of where we really are in adversarial robustness

More from Aleksander on this...