Deep Learning

CSCI 1470

Eric Ewing

Wednesday, 2/26/25

Day 15: Adversarial Learning

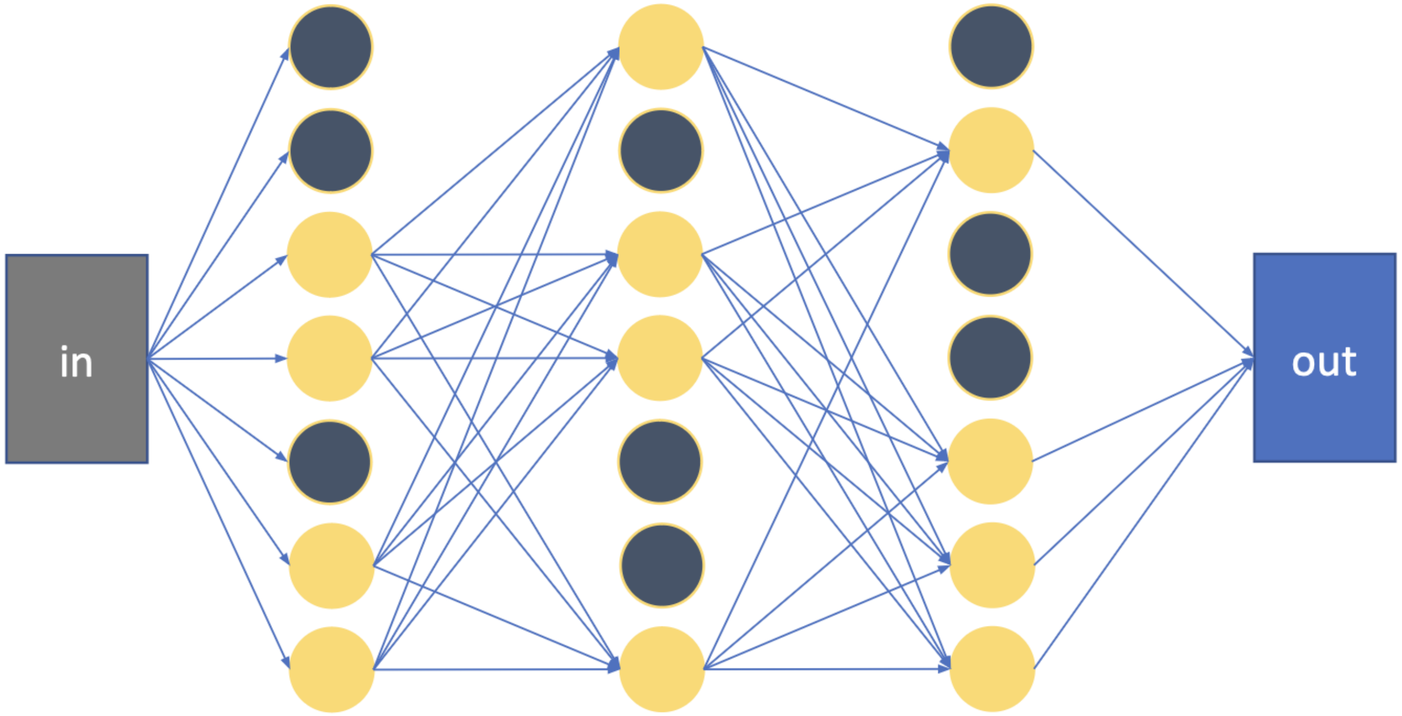
Logistics

- We will now allow up to 4 late days used for HW 2

Dropout - why?

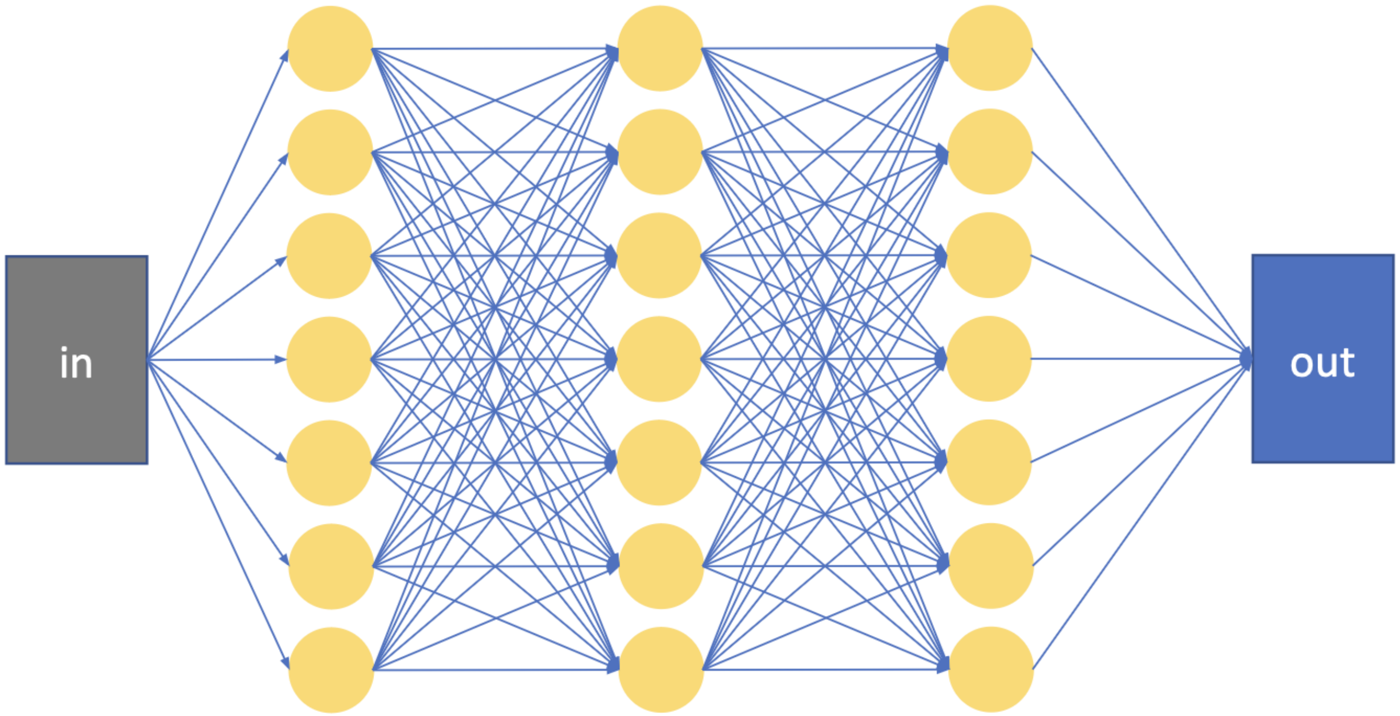
- Sort of looks like data augmentation, if you squint hard enough
  - Augmenting the data by randomly dropping out parts of it
- Over multiple passes through the net (i.e. during training over many epochs):
  - Randomly dropping neurons “forces” each neuron to learn a non-trivial weight
  - The network can’t learn to rely on spurious correlations (i.e. meaningless patterns), because they randomly might not be present

Dropout: Implications for test time



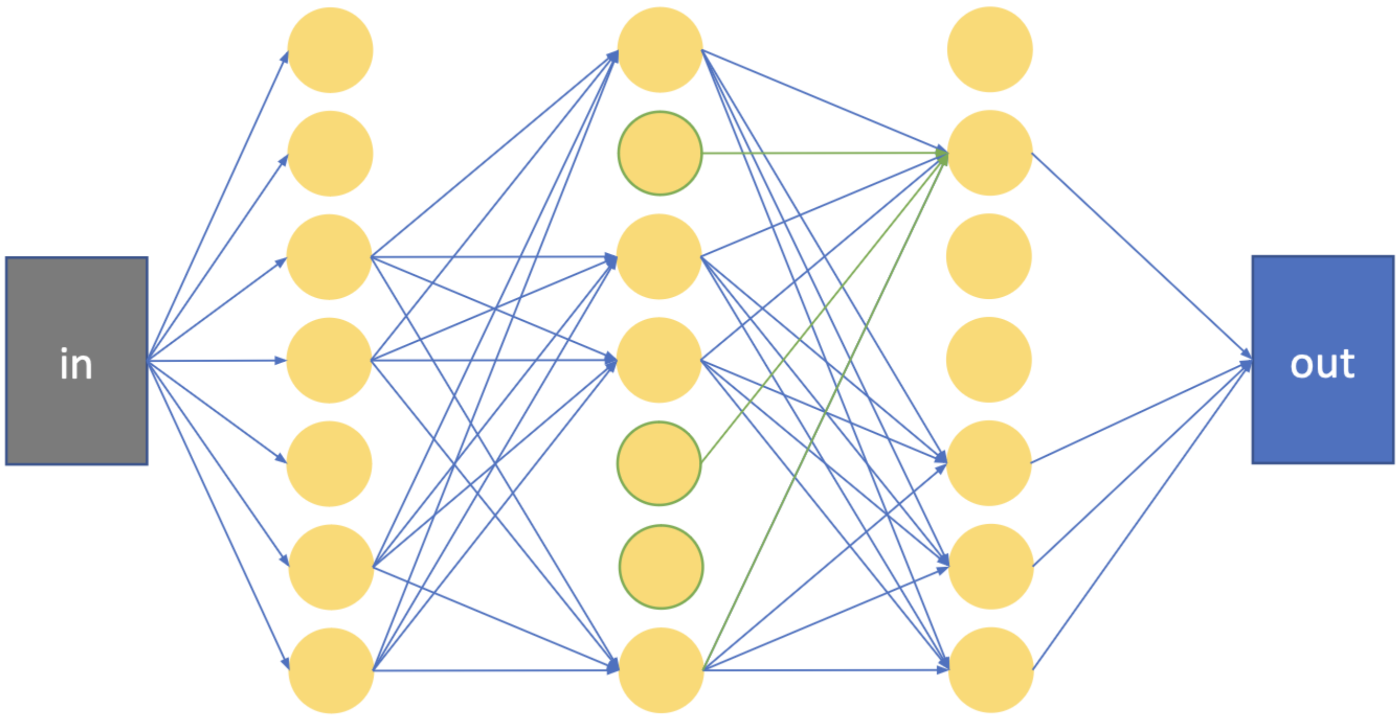
- During testing, we stop dropping out and use all of the neurons again

Dropout: Implications for test time



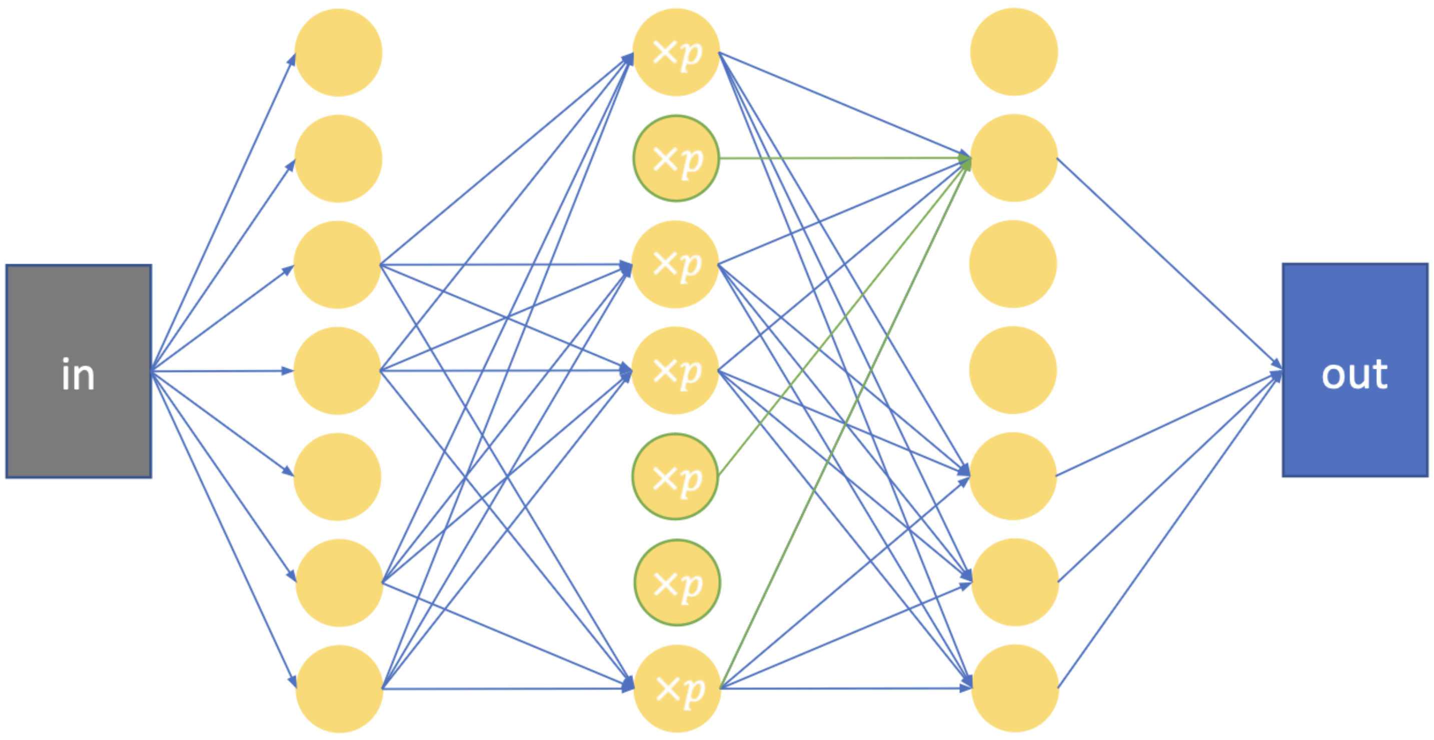
- During testing, we stop dropping out and use all of the neurons again

Dropout: Implications for test time



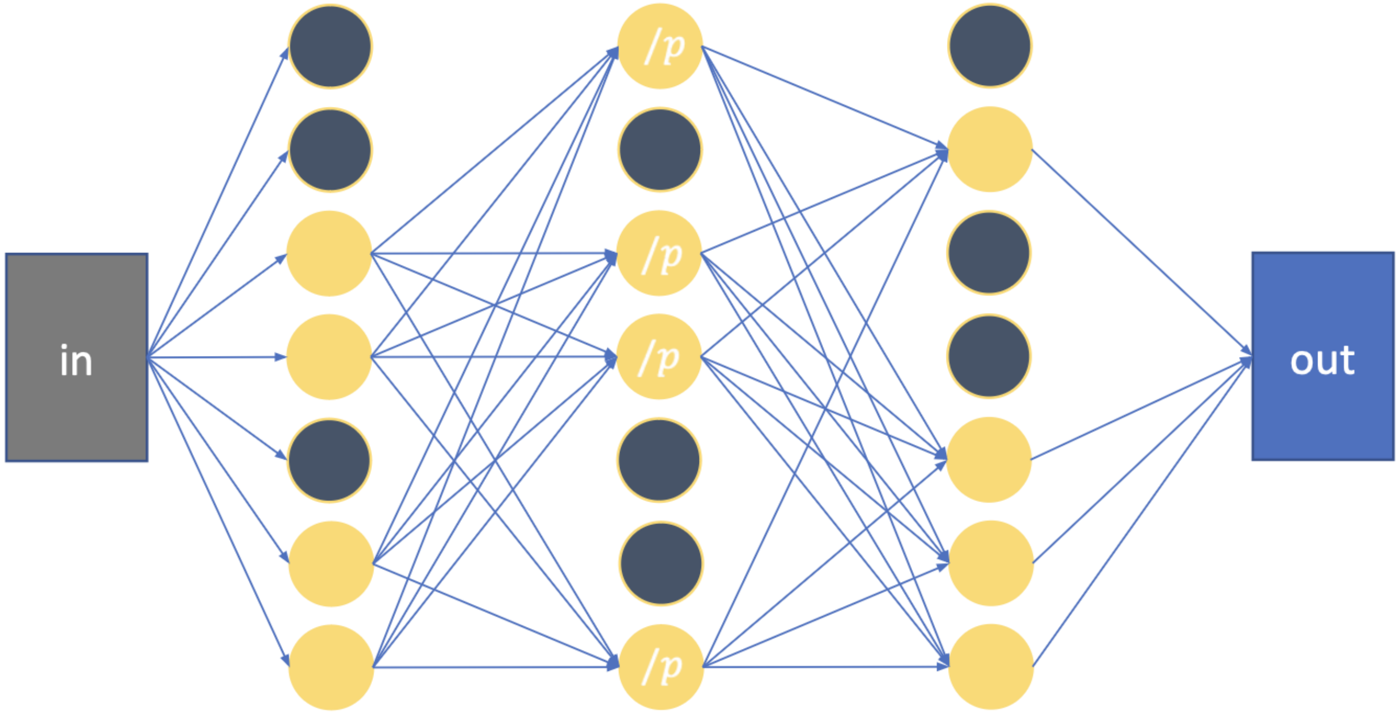
- During testing, we stop dropping out and use all of the neurons again
- If a layer keeps a fraction p of its neurons during training, then when we use all the neurons at test time, the next layer will get a bigger input than expected…
- ***What do we do!?***

Dropout: Implications for test time



- **Solution 1:**
  Multiply the values of all neurons by p, so that the expected magnitude of the sum of neurons is the same

Dropout: Implications for test time



- **Solution 1:**
  Multiply the values of all neurons by p, so that the expected magnitude of the sum of neurons is the same
- **Solution 2:**
  At training time, divide the values of the kept neurons by p

Dropout - implementation

Any questions?

- Handy keras layer!

- **`tf.keras.layers.Dropout(rate)`**
    - Hyperparameter `rate` between [0, 1]: the rate at which the outputs of the previous layer are dropped
    - **`Rate = 0.5`**: drop half, keep half
    - **`Rate = 0.25`**: drop ¼, keep ¾

The Real World

- Hey, your models work great!
- Let's deploy them to the real world!
- What could go wrong?

Papernot et al. Practical Black-Box Attacks against Machine Learning

The Real World

- Hey, your models work great!
- Let's deploy them to the real world!
- What could go wrong?

Stop Sign: 99%

Papernot et al. Practical Black-Box Attacks against Machine Learning

The Real World

- Hey, your models work great!
- Let’s deploy them to the real world!
- What could go wrong?

Stop Sign: 99%

Papernot et al. Practical Black-Box Attacks against Machine Learning

The Real World

- Hey, your models work great!
- Let’s deploy them to the real world!
- What could go wrong?

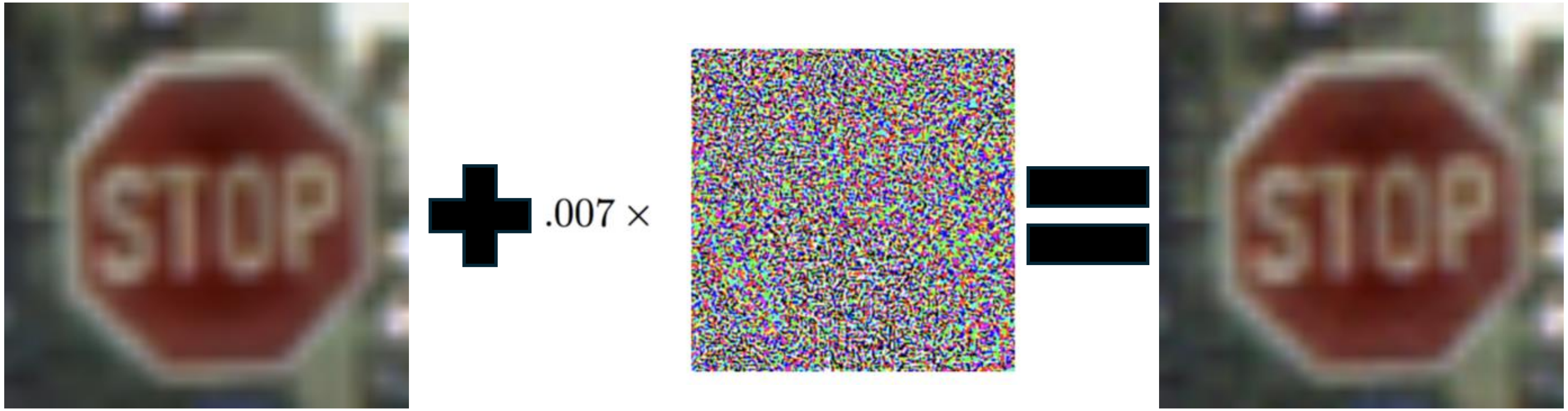
Stop Sign: 99%

Yield Sign: 99%

Papernot et al. Practical Black-Box Attacks against Machine Learning

The Real World

- Hey, your models work great!
- Let’s deploy them to the real world!
- What could go wrong?



Stop Sign: 99%

Yield Sign: 99%

Papernot et al. Practical Black-Box Attacks against Machine Learning

Adversarial Learning

- Can we (or adversaries) break our deep learning models
- Adversarial Attack: Can we add a small amount of noise to an input that results in a misclassification?
- Data Poisoning: Can we insert data in the training dataset that corrupts the model's training?

Objective

- In Deep Learning, our objective is to minimize loss
- What do you think the objective of our adversary is?

Objective

- In Deep Learning, our objective is to minimize loss
- What do you think the objective of our adversary is?

Maximize (Test) Loss

Objective

- In Deep Learning, our objective is to minimize loss
- What do you think the objective of our adversary is?

Maximize (Test) Loss

Want to follow direction of gradient (Gradient Ascent)

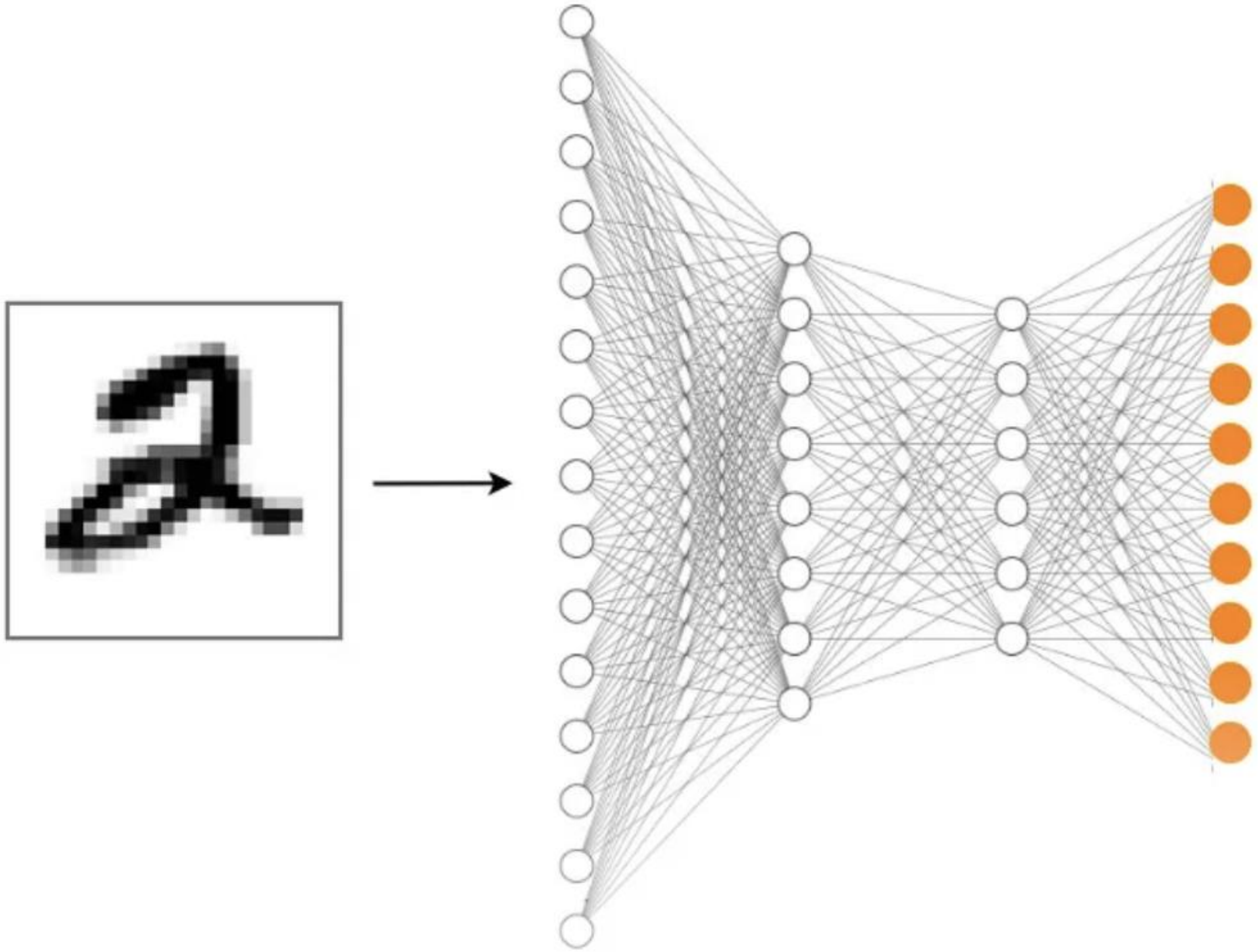
Objective

- In Deep Learning, our objective is to minimize loss
- What do you think the objective of our adversary is?
- What does our adversary have control of?
  - Input data?
  - Training Data?
  - Our model? (Uh oh)

Objective

- In Deep Learning, our objective is to minimize loss
- What do you think the objective of our adversary is?
- What does our adversary have control of?
  - Input data?
  - Training Data?
  - Our model? (Uh oh)

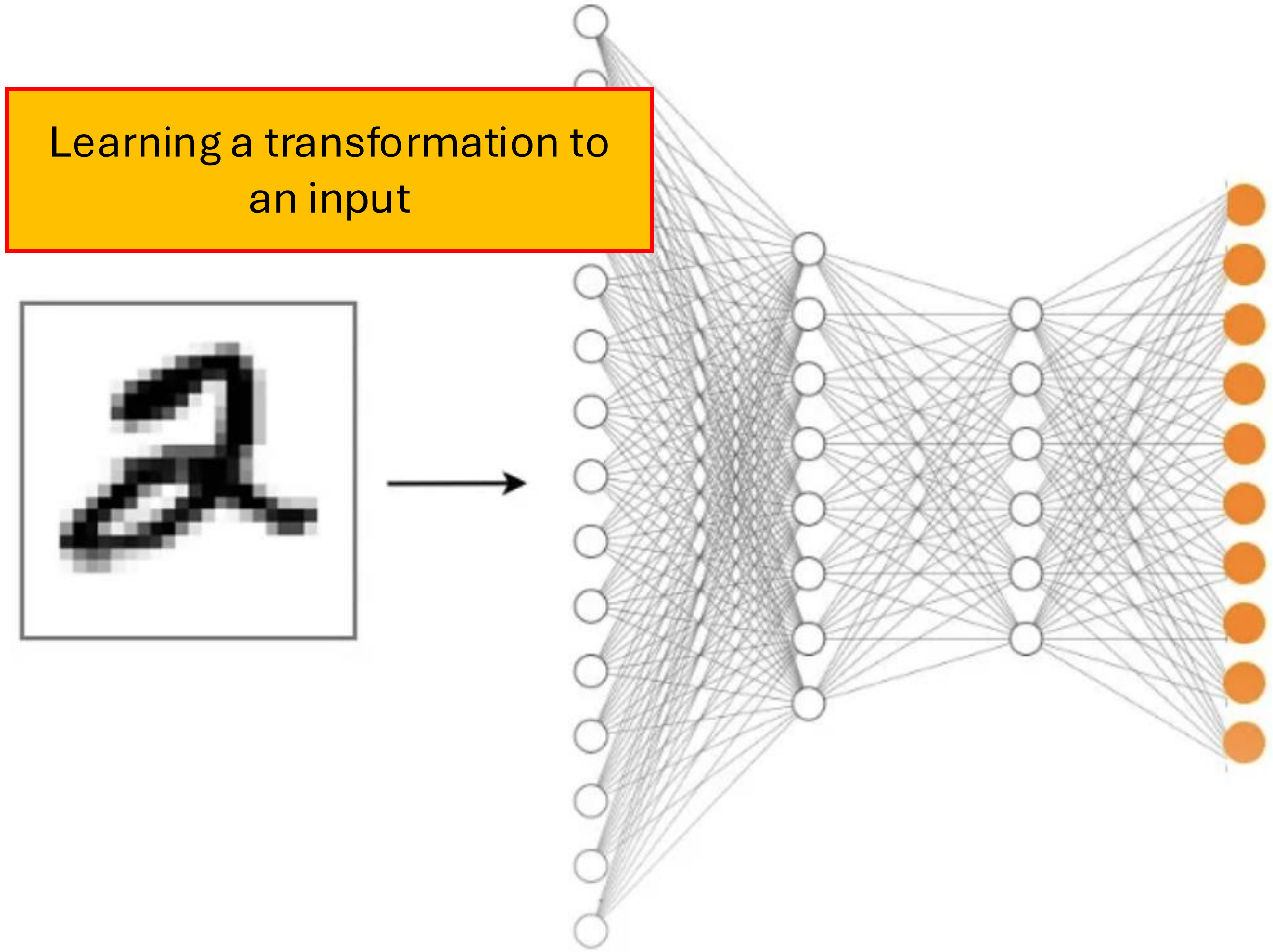
Most Commonly Studied



Normal Training:
- Compute gradients wrt weights and biases
- Update via gradient descent

Adversarial Example:
- Compute gradients wrt input
- Update **input** via gradient **ascent**

Learning a transformation to an input



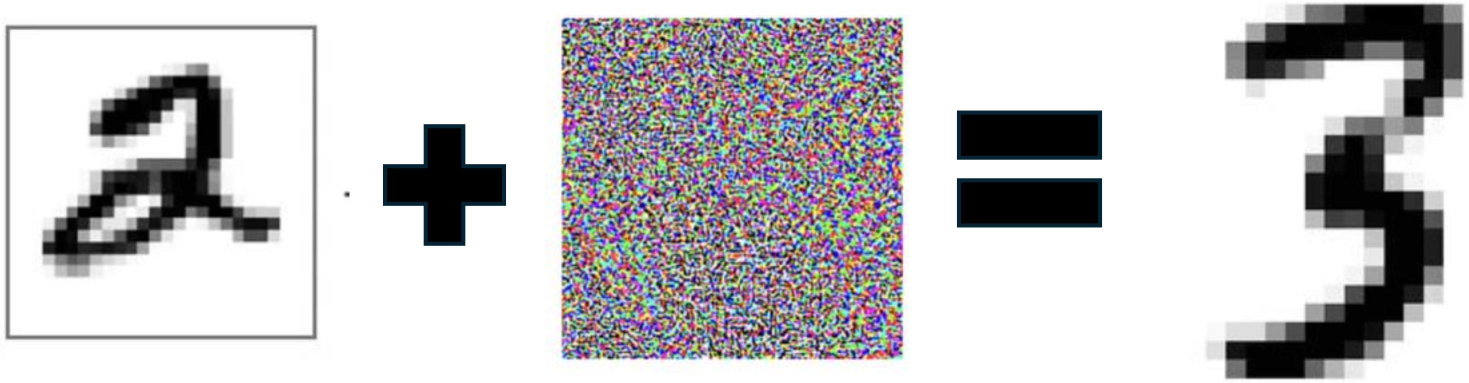
Normal Training:
- Compute gradients wrt weights and biases
- Update via gradient descent

Adversarial Example:
- Compute gradients wrt input
- Update input via gradient ascent

Attack Model

We do not expect to be able to withstand an attacker with unlimited power.

If attackers can add unlimited noise, they can just change the image entirely.

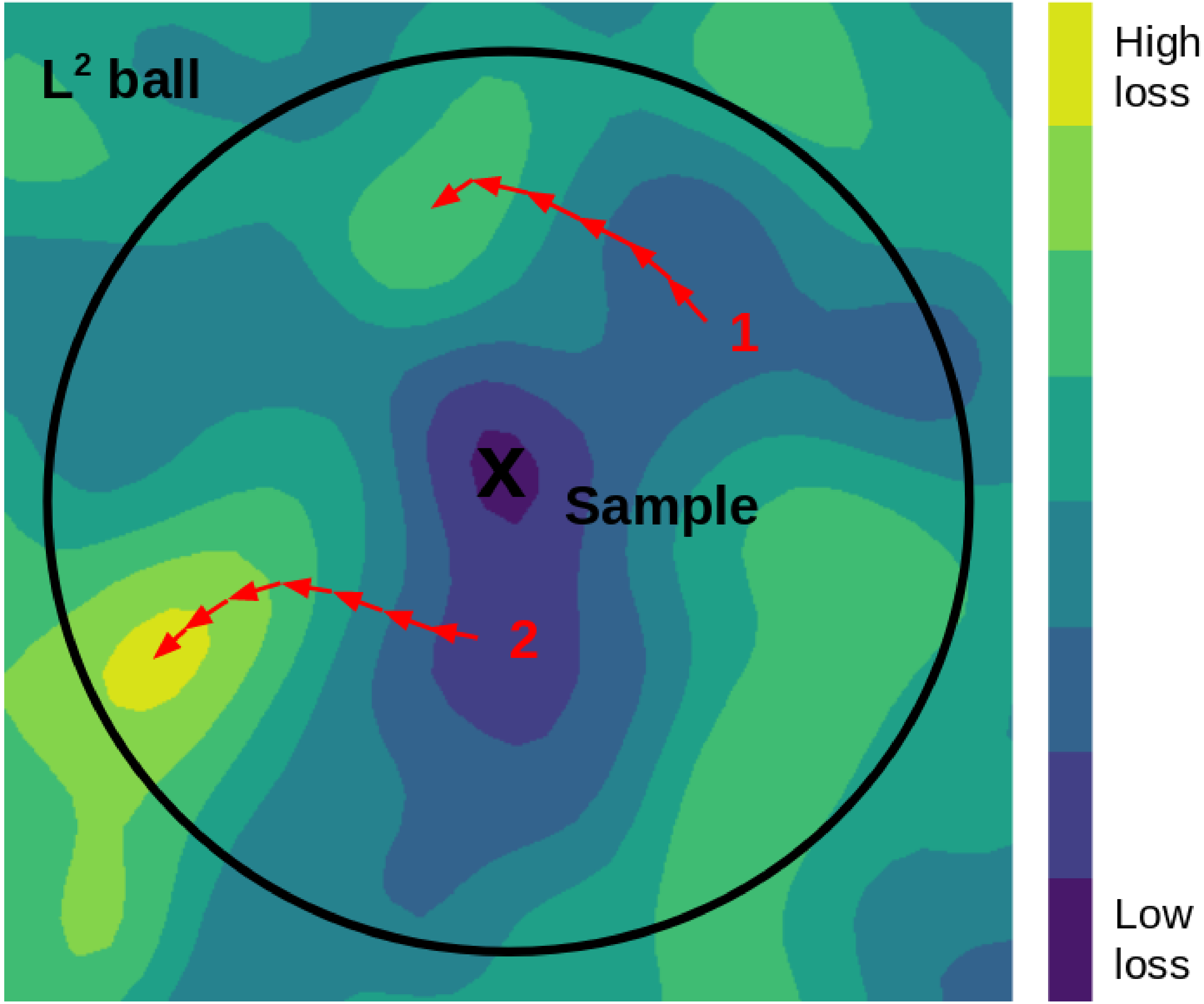


Threat Model

- We limit the power of the attacker
- Attacks must fall within some L^p-Ball of radius r
  - L^1-Ball: Sum of noise must be below r
  - L^2-Ball: Square root (sum of squared noise for each pixel) must be below r
  - L^∞-Ball: Largest individual value of attack noise must be below r

Gradient Ascent around an input sample

What happens if we hit the constraint and can't keep following the gradient?

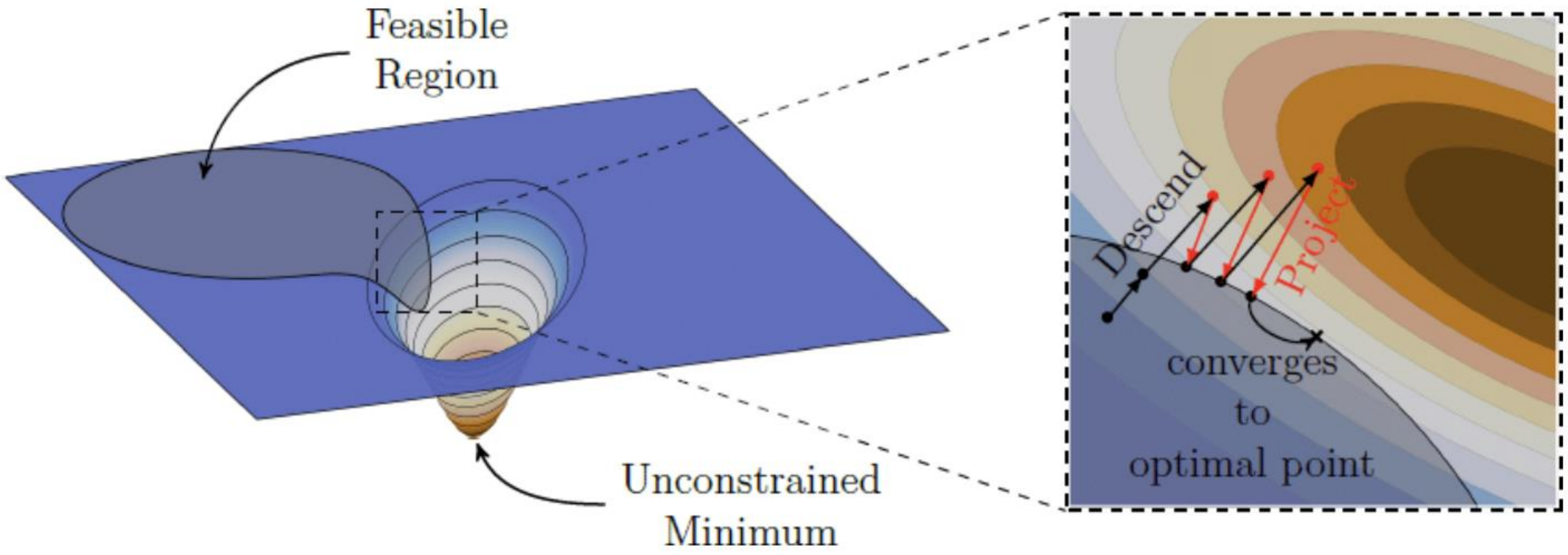


https://medium.com/towards-data-science/know-your-enemy-7f7c5038bdf3

Constrained Optimization

- Projected Gradient Ascent (PGA):
  - Run Gradient Ascent
  - If noise goes outside of constraint set, project back into constraint set

(Picture is for minimization)



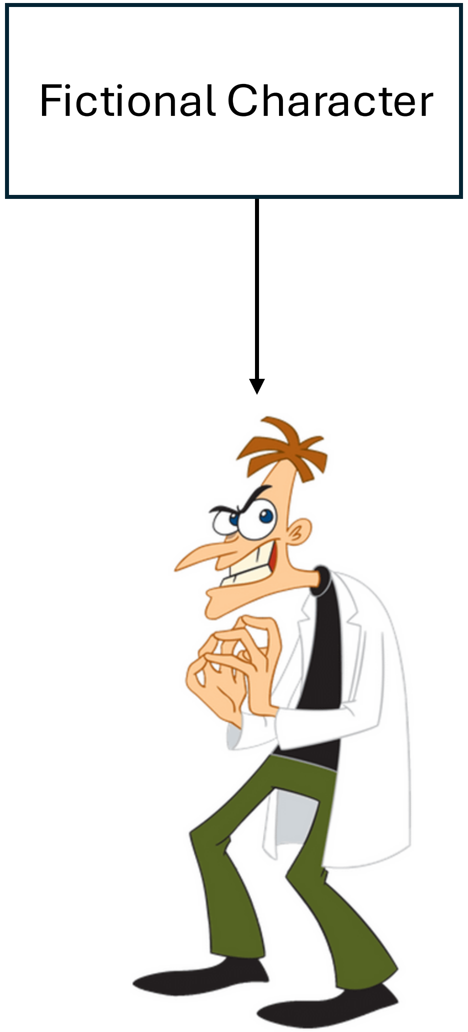
https://www.researchgate.net/publication/358122337_Adaptive_Model_for_Magnetic_Particle_Mapping_Using_Magnetoelectric_Sensors

How big of a problem is this?

- Most models will never be under threat from adversarial attacks
- But doesn’t this tell us something new about our models?

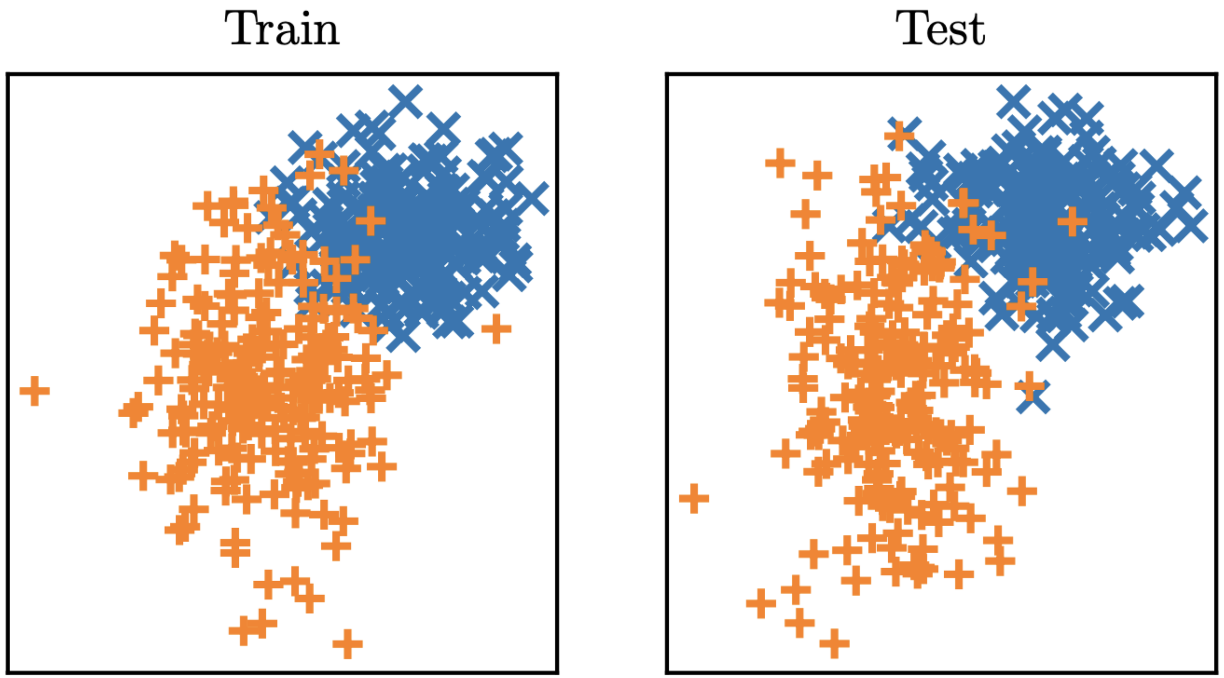
How big of a problem is this?

- Most models will never be under threat from adversarial attacks
- But doesn't this tell us something new about our models?



Why Adversarial Attacks Work

I.I.D. Machine Learning



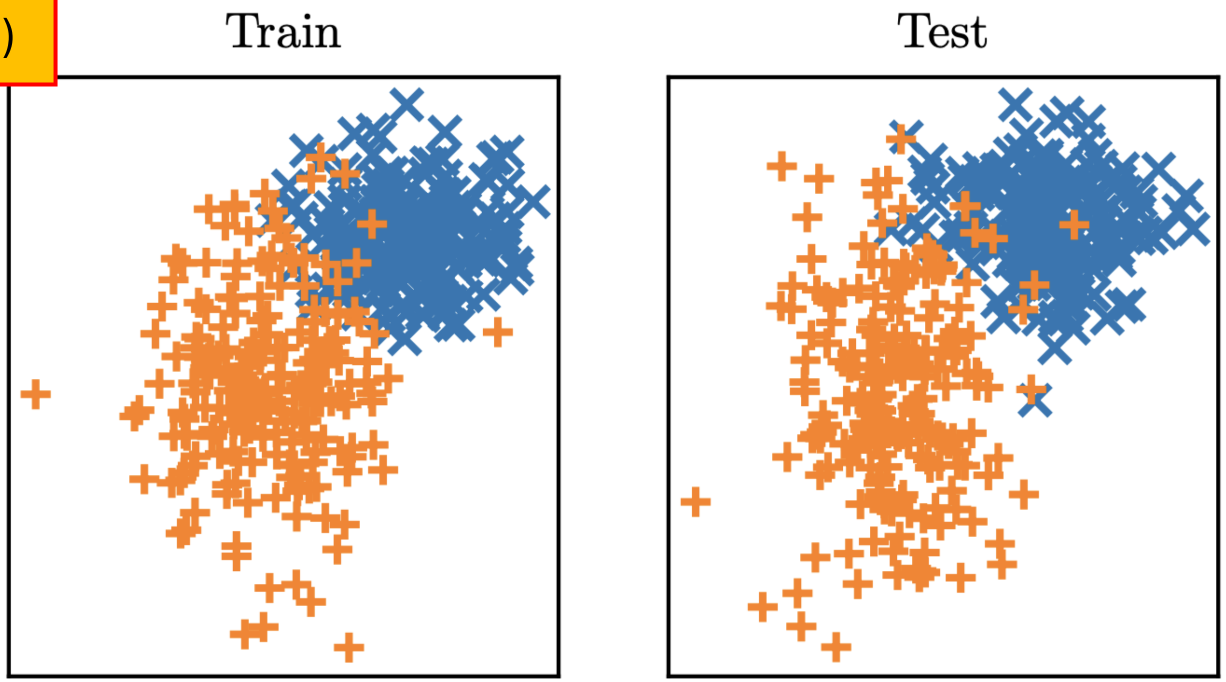
I: Independent
I: Identically
D: Distributed

All train and test examples drawn independently from same distribution

Why Adversarial Attacks Work

I.I.D. Machine Learning

We assume our datasets are IID (Train set looks like validation set looks like test set)



I: Independent
I: Identically
D: Distributed

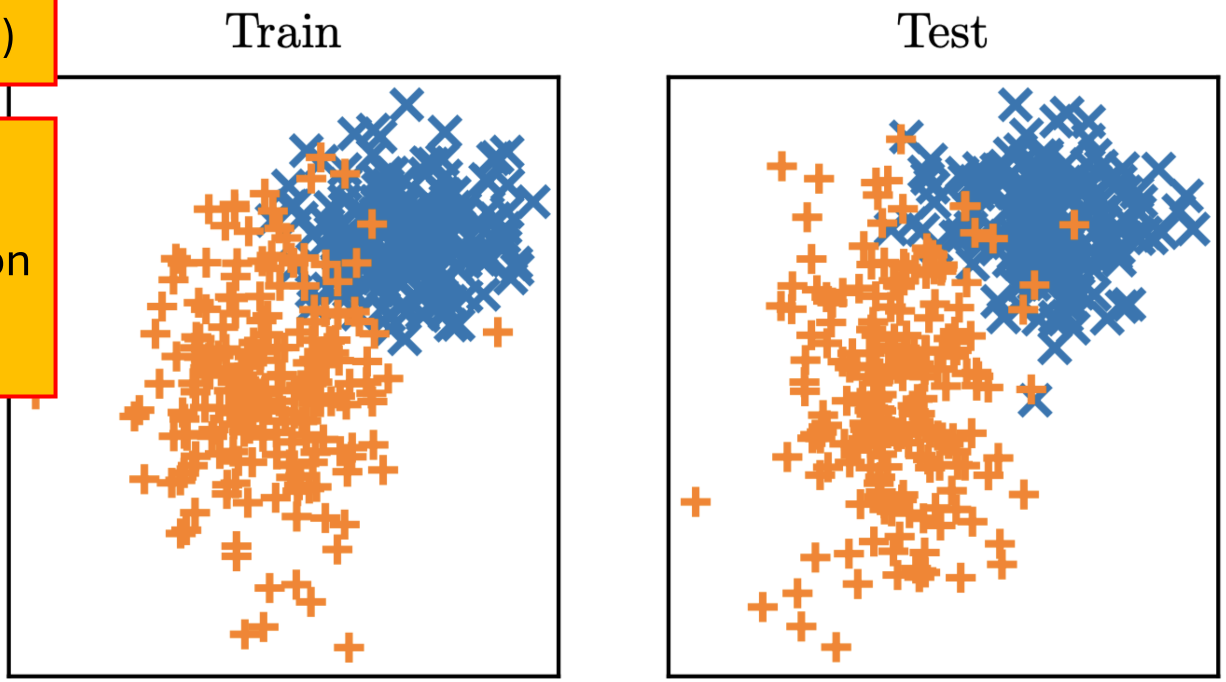
All train and test examples drawn independently from same distribution

Why Adversarial Attacks Work

We assume our datasets are IID (Train set looks like validation set looks like test set)

Adversarial attacks change the distribution of the test set

I.I.D. Machine Learning



I: Independent

I: Identically

D: Distributed

All train and test examples drawn independently from same distribution

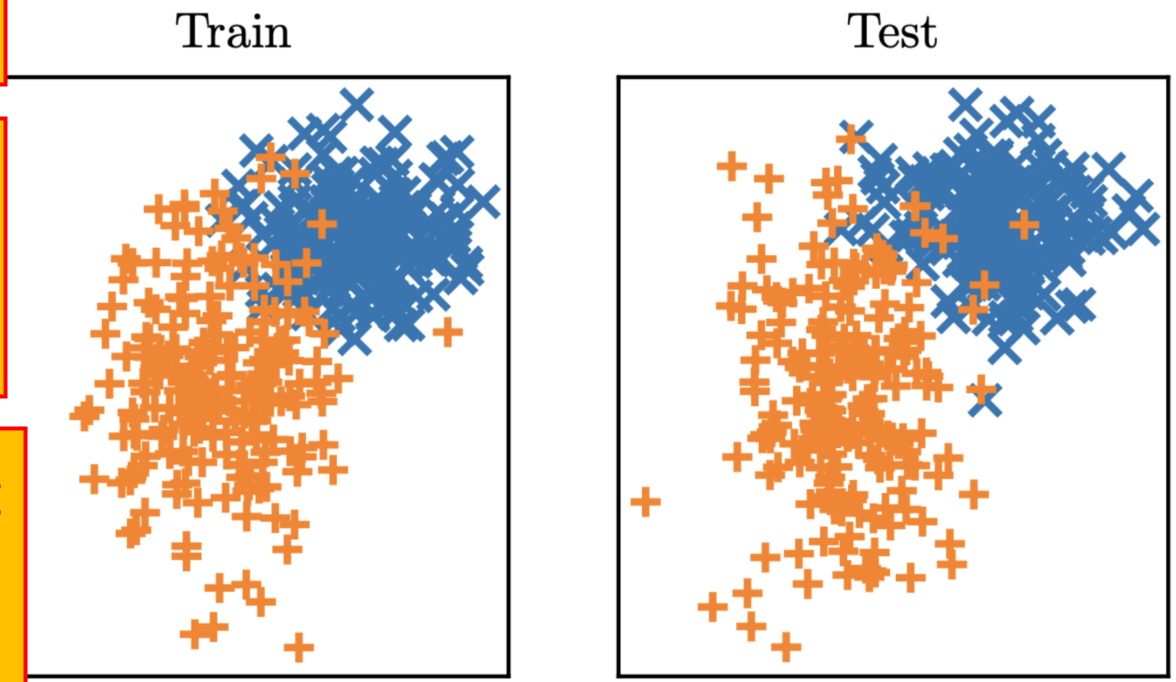
Why Adversarial Attacks Work

I.I.D. Machine Learning

We assume our datasets are IID (Train set looks like validation set looks like test set)

Adversarial attacks change the distribution of the test set

Performance on training set/validation set is no longer indicative of test performance



I: Independent

I: Identically

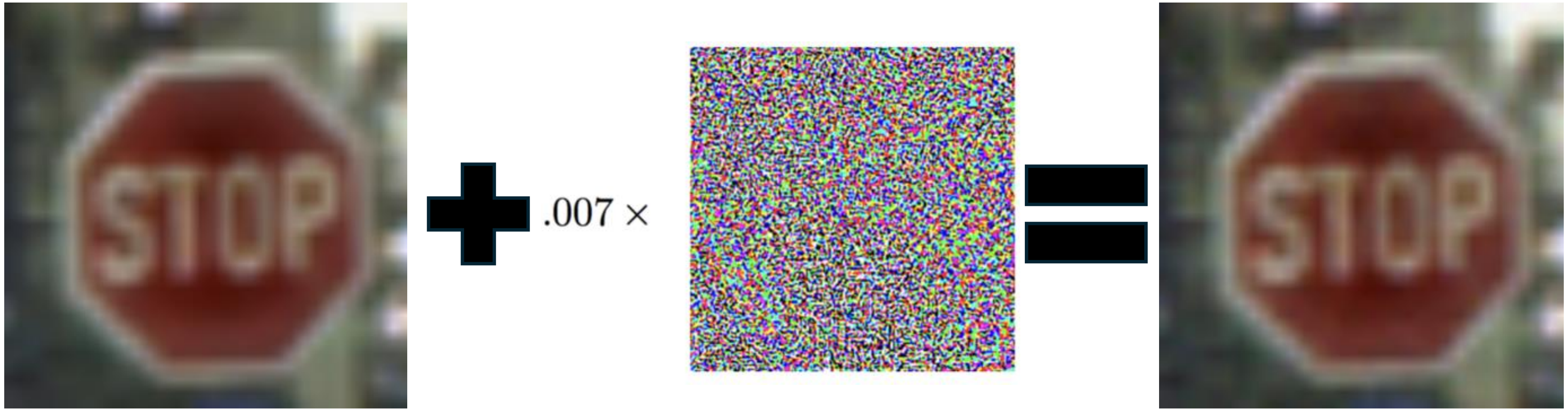
D: Distributed

All train and test examples drawn independently from same distribution

What did we learn in the first place?

If such small noise can change the outputs of our network, it clearly is not making decisions in the way that humans do.

It isn't always making decisions about stop signs based on color, shape, or text...

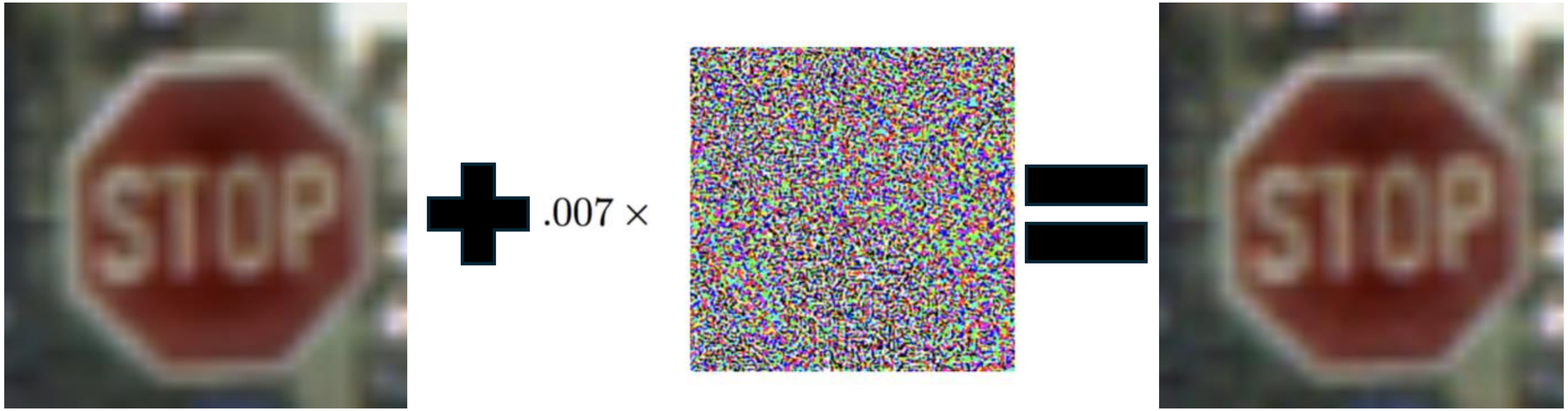


Stop Sign

Yield

What did we learn in the first place?

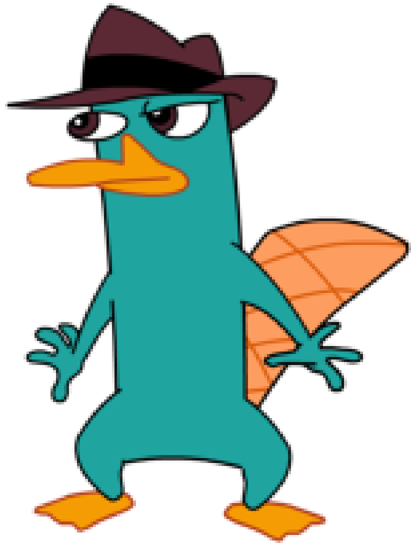
Deep learning learns the “easiest” good representation, which can be very brittle and break under small perturbations



Stop Sign

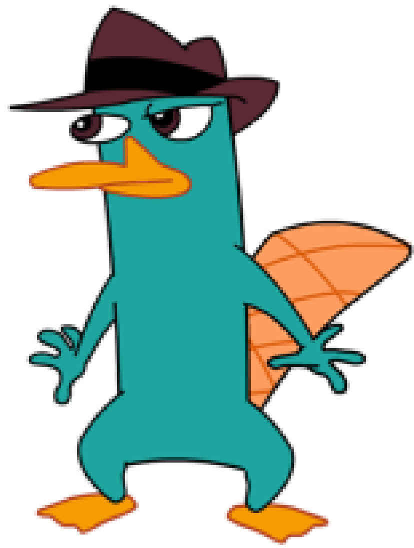
Yield

Defenses



Defenses

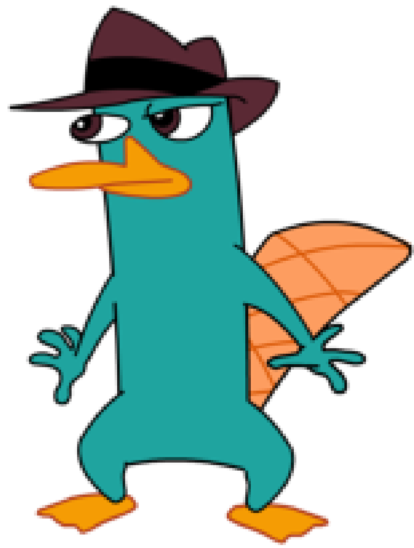
How can we make more robust models?



Defenses

How can we make more robust models?

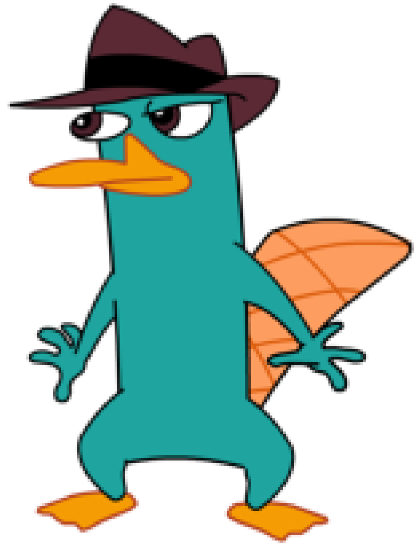
- Ensembles
  - Train multiple different models average results
  - (Can make models more robust, but not resistant to adversarial attacks)



Defenses

How can we make more robust models?

- Ensembles
  - Train multiple different models average results
  - (Can make models more robust, but not resistant to adversarial attacks)
- Data Augmentation?
  - Just add lots of random noise to inputs while training?
  - Add in Adversarial Examples while training?



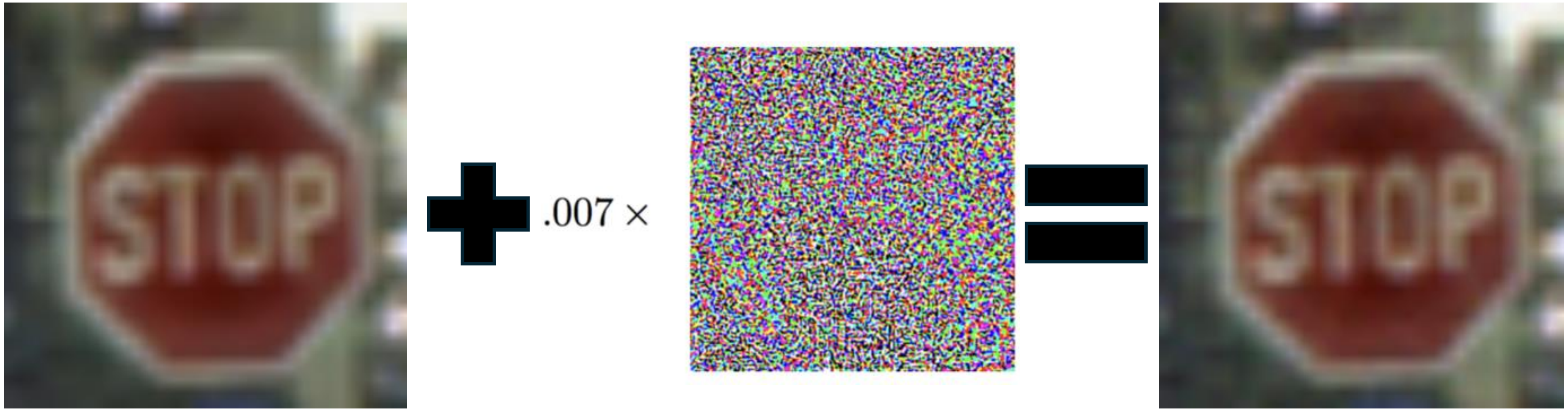
Defenses

How can we make more robust models?

- Ensembles
  - Train multiple different models average results
  - (Can make models more robust, but not resistant to adversarial attacks)
- Data Augmentation?
  - Just add lots of random noise to inputs while training?
  - Add in Adversarial Examples while training?
- Provably Robust Networks
  - Lipschitz Continuity!

Attack Transfer

Adversarial Examples *tend to fool* other networks as well



Stop Sign

Yield

Attack Transfer

Adversarial Examples *tend to fool* other networks as well

If this attack was made using ResNet, it would likely work against VGG

Stop Sign

+ $.007 \times$

=

Yield

Attack Transfer

- This also gives us another tool for adversarial attacks
- Suppose the model we are trying to break is not public (i.e., you can't find the gradients)
- Black-box attack:
  - Train a "surrogate" model on the same dataset
  - Construct an adversarial example that works against your surrogate model
  - Send attack to original model

Data Augmentation

If breaking the IID assumption caused our issues, can we just change the distribution of the training set?

Data Augmentation

What if we just add lots of images with small amounts of random noise to our training data?

Data Augmentation

What if we just add lots of images with small amounts of random noise to our training data?

Cannot have enough new data to densely sample a high dimensional ball around each original input (number of points required grows exponentially with dimension)

Data Augmentation

What if we just add lots of images with small amounts of random noise to our training data?

Cannot have enough new data to densely sample a high dimensional ball around each original input (number of points required grows exponentially with dimension)

Holes will still exist where your network can be exploited

Adversarial Training

Adversarial Training

New Training Objective: Train a network that has lowest loss **when attacked**

Adversarial Training

New Training Objective: Train a network that has lowest loss **when attacked**

$$\min_{\theta} \max_{\epsilon} L(x + \epsilon)$$

Adversarial Training

New Training Objective: Train a network that has lowest loss **when attacked**

$$\min_{\theta} \max_{\epsilon} L(x + \epsilon)$$

Min-Max optimization problem can utilize sets of techniques from adversarial game theory

Adversarial Training

For each batch:

Network produces output y_{pred}

Attacker finds attack noise ϵ

$y_{adv} = y_{pred} + \epsilon$

Compute loss $L(y_{adv}, y)$

Run SGD to update weights

Adversarial Training

For each batch:

Network produces output y_{pred}

Attacker finds attack noise ϵ

$y_{adv} = y_{pred} + \epsilon$

Compute loss $L(y_{adv}, y)$

Run SGD to update weights

Another whole gradient descent process

Adversarial Training

For each batch:

Network produces output y_{pred}

Attacker finds attack noise ϵ

$y_{adv} = y_{pred} + \epsilon$

Compute loss $L(y_{adv}, y)$

Run SGD to update weights

Another whole gradient descent process

Adversary makes move (generates noise)
Defender responds (updates weights)

Adversarial Training

For each batch:

Network produces output y_{pred}

Attacker finds attack noise ϵ

$y_{adv} = y_{pred} + \epsilon$

Compute loss $L(y_{adv}, y)$

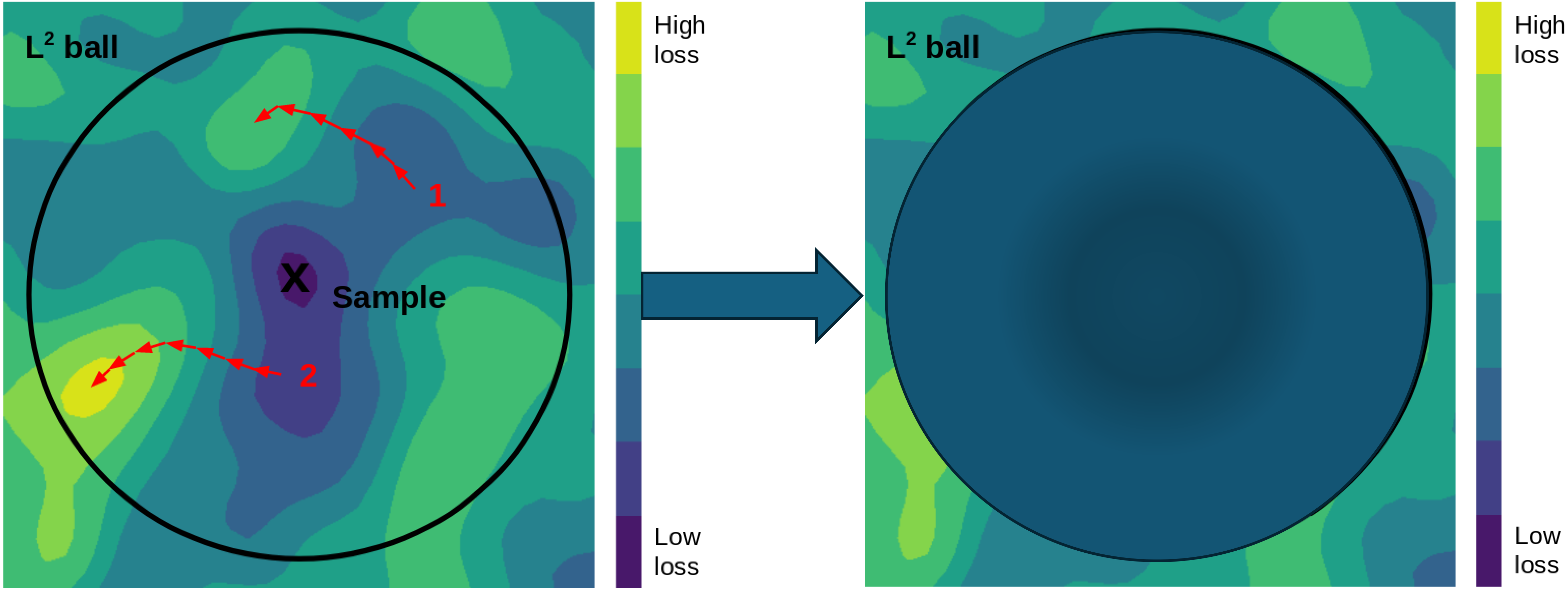
Run SGD to update weights

Another whole gradient descent process

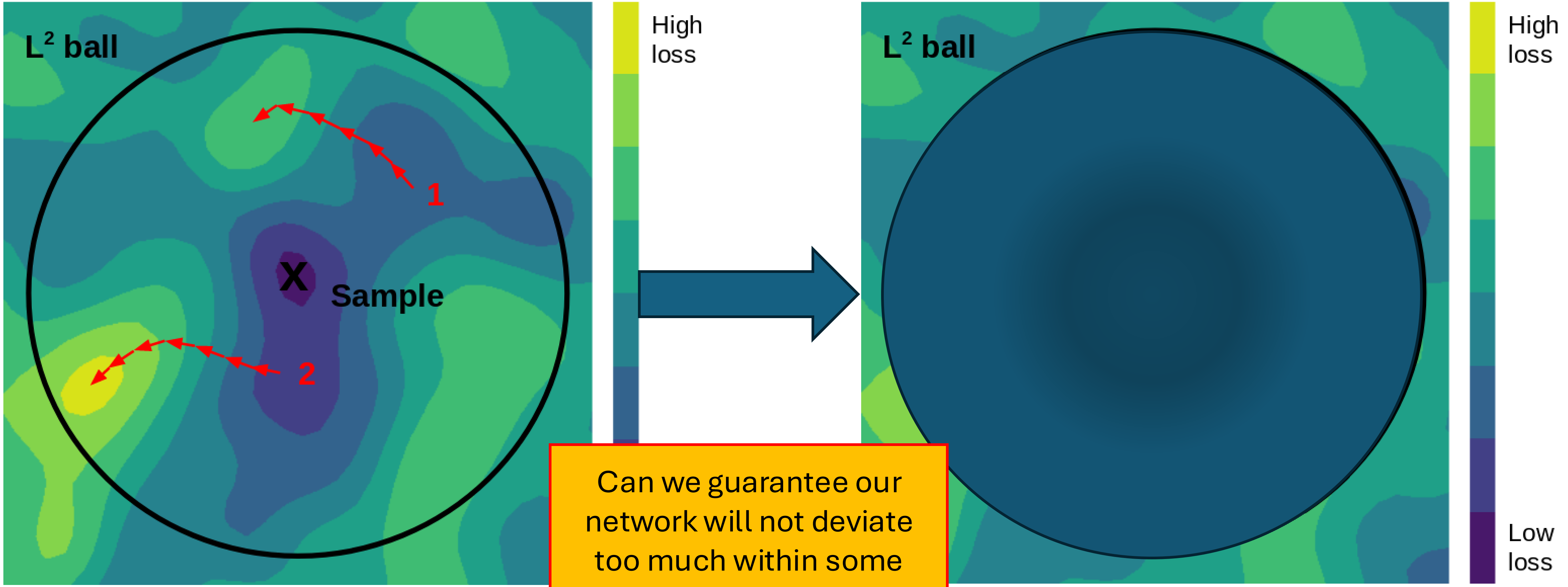
Adversary makes move (generates noise)
Defender responds (updates weights)

What are the tradeoffs of using adversarial training?

Provably Robust Networks



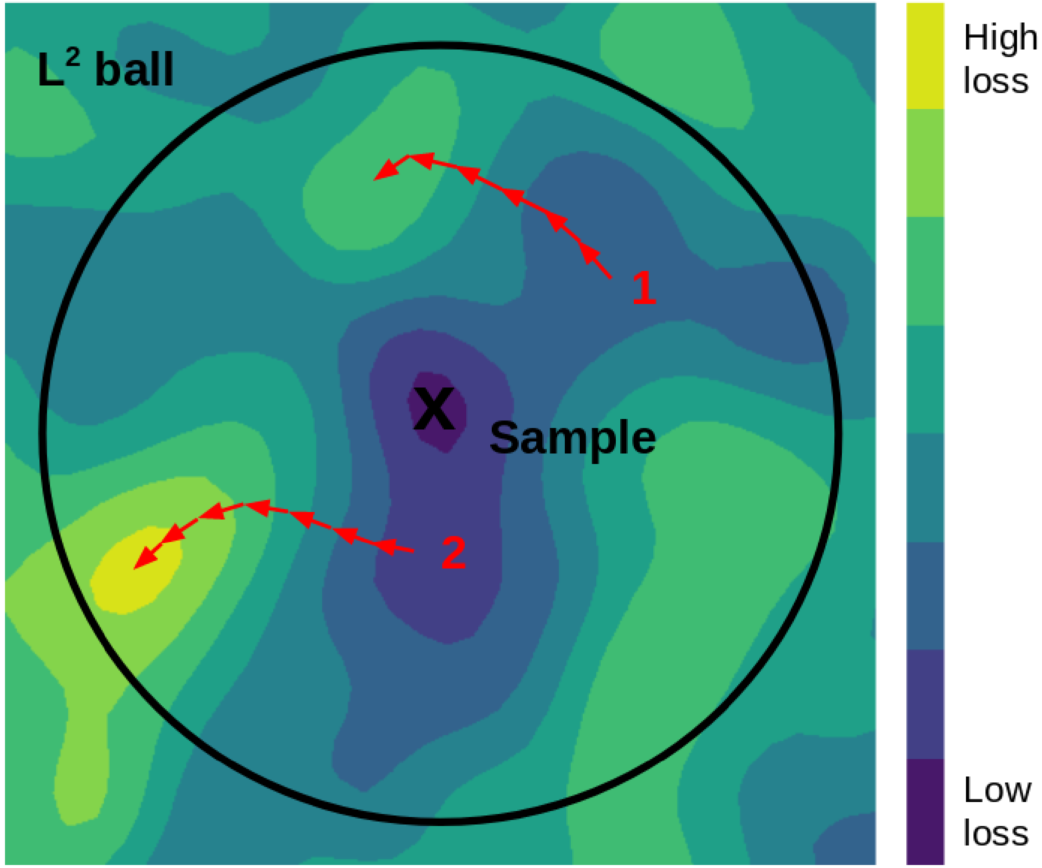
Provably Robust Networks



Can we guarantee our network will not deviate too much within some radius?

Maximum Gradient

If we knew the maximum gradient $c = \nabla_\epsilon L$, then we know that our loss function can change up to $c \cdot r$

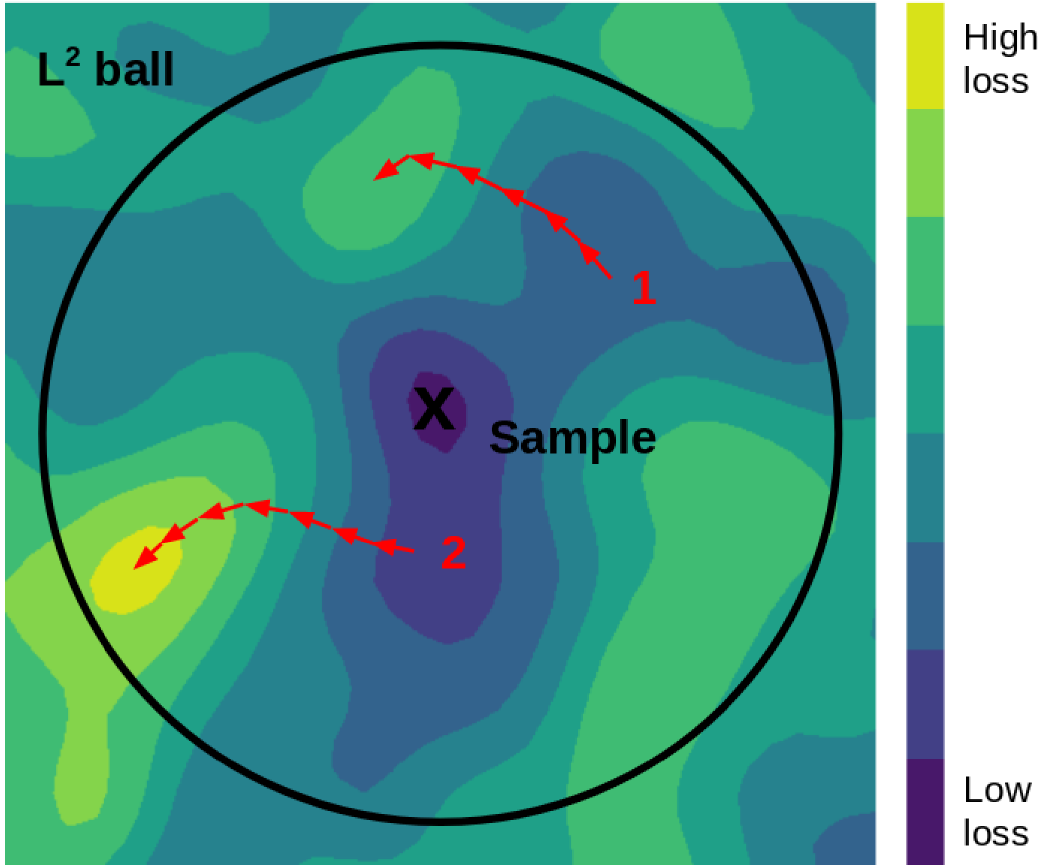


If we can bound the gradient of a function to some constant c, that function is *Lipschitz Continuous*.

Maximum Gradient

If we knew the maximum gradient $c = \nabla_\epsilon L$, then we know that our loss function can change up to $c \cdot r$

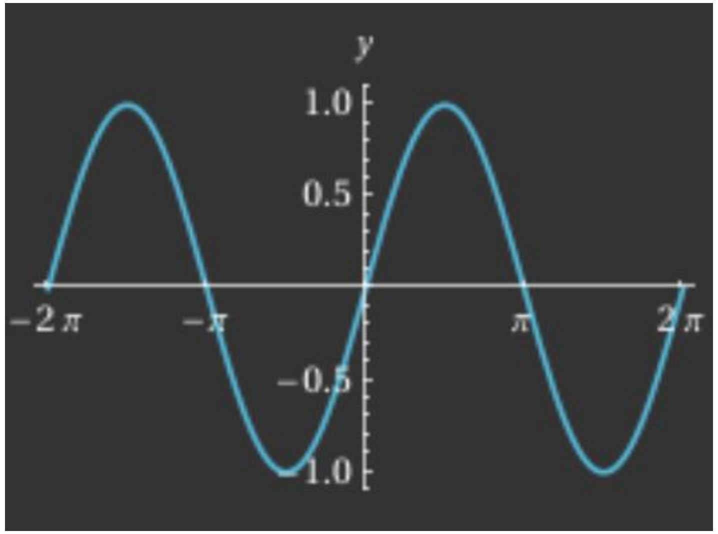
Why is this true?



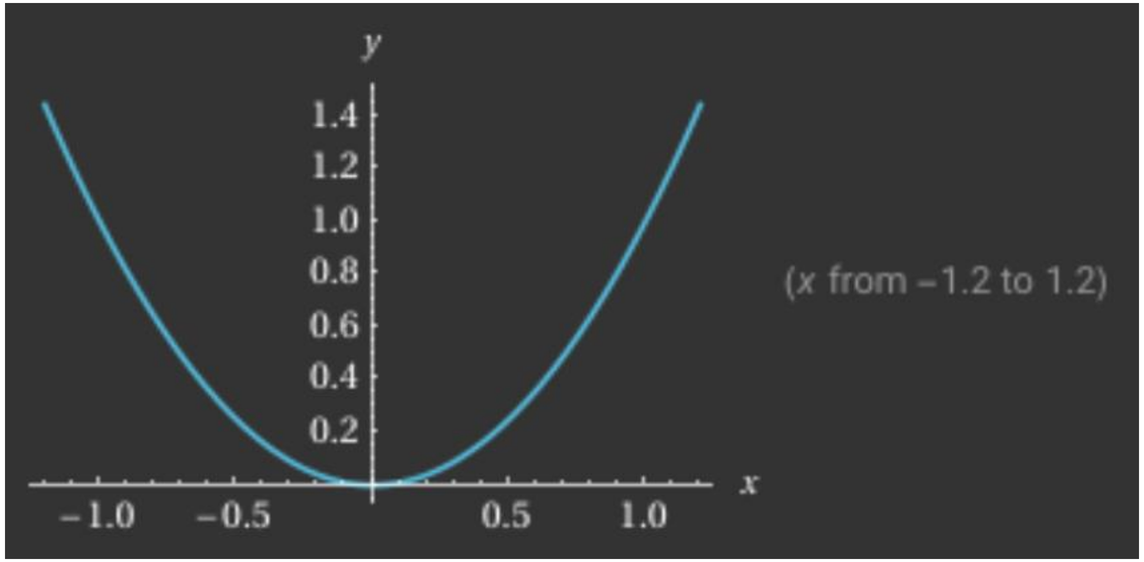
If we can bound the gradient of a function to some constant c, that function is *Lipschitz Continuous*.

Lipschitz Continuity

$\sin(x)$ is Lipschitz Continuous,
it has a maximum derivative of 1



x^2 is not Lipschitz Continuous, it
does not have a maximum derivative



Are Neural Networks Lipschitz Continuous?

Are Neural Networks Lipschitz Continuous?

- If f and g are both Lipschitz continuous functions, then h=f(g(x)) is also Lipschitz continuous.

Are Neural Networks Lipschitz Continuous?

- If f and g are both Lipschitz continuous functions, then h=f(g(x)) is also Lipschitz continuous.
- Gradients are determined by weight layers and activation functions

Are Neural Networks Lipschitz Continuous?

- If f and g are both Lipschitz continuous functions, then h=f(g(x)) is also Lipschitz continuous.
- Gradients are determined by weight layers and activation functions
- Assume ReLU activation for simplicity (maximum derivative of 1)

Are Neural Networks Lipschitz Continuous?

- If f and g are both Lipschitz continuous functions, then h=f(g(x)) is also Lipschitz continuous.
- Gradients are determined by weight layers and activation functions
- Assume ReLU activation for simplicity (maximum derivative of 1)
- Maximum gradient possible is determined by weights of network (which are finite)

Are Neural Networks Lipschitz Continuous?

- If f and g are both Lipschitz continuous functions, then h=f(g(x)) is also Lipschitz continuous.
- Gradients are determined by weight layers and activation functions
- Assume ReLU activation for simplicity (maximum derivative of 1)
- Maximum gradient possible is determined by weights of network (which are finite)
- Lipschitz constant *c* may be very large, but it exists

Limiting Lipschitz Constant

Limiting Lipschitz Constant

- It can be shown that the Lipschitz Constant for a single weight matrix W is the largest singular value of that matrix
  - The largest Singular Value is the square root of the largest eigenvalue of the matrix $W^T W$

Limiting Lipschitz Constant

- It can be shown that the Lipschitz Constant for a single weight matrix W is the largest singular value of that matrix
  - The largest Singular Value is the square root of the largest eigenvalue of the matrix $W^T W$
- If we want to limit the Lipschitz Constant for a single layer, we just have to divide by that Singular Value…
  - Can divide by 2 * Singular value to limit Lipschitz constant to 1/2

Limiting Lipschitz Constant

- It can be shown that the Lipschitz Constant for a single weight matrix W is the largest singular value of that matrix
  - The largest Singular Value is the square root of the largest eigenvalue of the matrix $W^T W$
- If we want to limit the Lipschitz Constant for a single layer, we just have to divide by that Singular Value…
  - Can divide by 2 * Singular value to limit Lipschitz constant to 1/2
- This is called Spectral Normalization

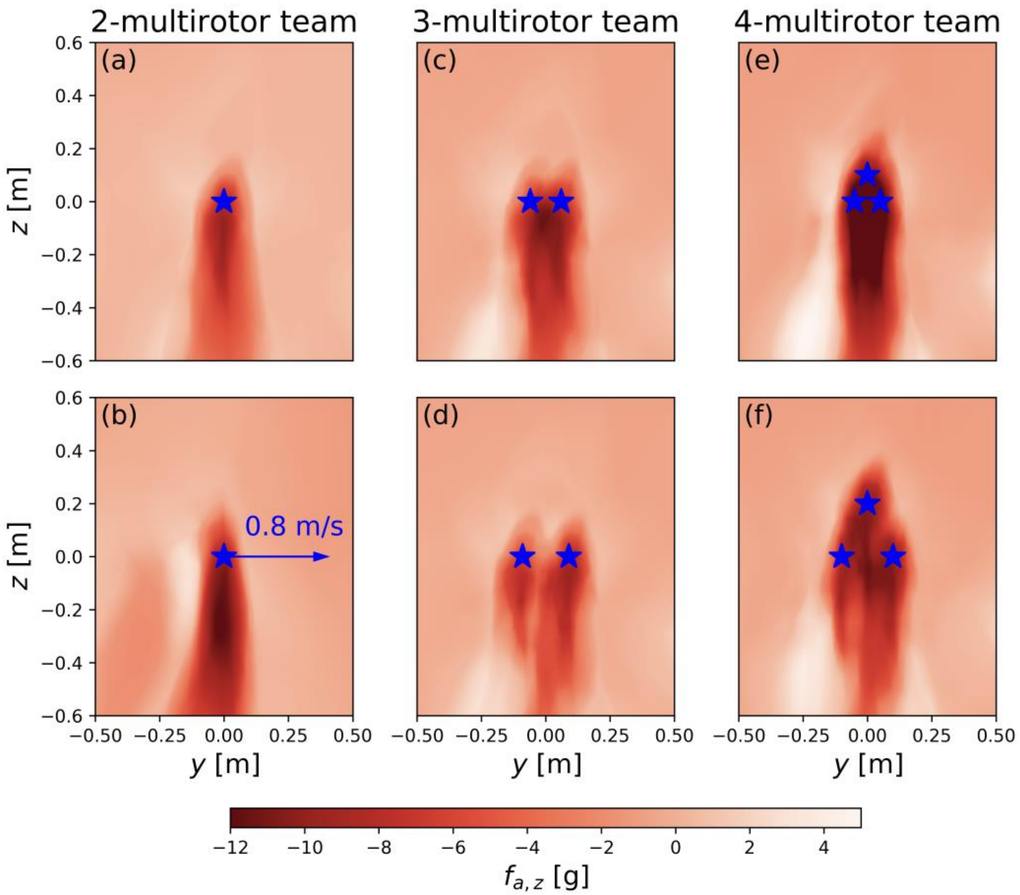
Lipschitz Continuity

- Adding SpectralNormalization to layers, like BatchNorm, can help networks learn smoother loss functions
- Can make models (slightly more) robust to adversarial attacks
- The downside is that it is a much more restrictive condition on the network and the network may no longer learn good policies

Also for other applications…

Many physical phenomena are also Lipschitz Continuous

If you are trying to predict a physical phenomena, it may make sense to use Lipschitz continuity regardless of adversarial attacks.



Shi et al. Neural Swarm. 2022

Takeaways

Adversarial Attacks show how brittle models can be

Studying them gives us insights into what our networks learn

Defenses that make models robust against attacks probably also make them robust against other disturbances as well