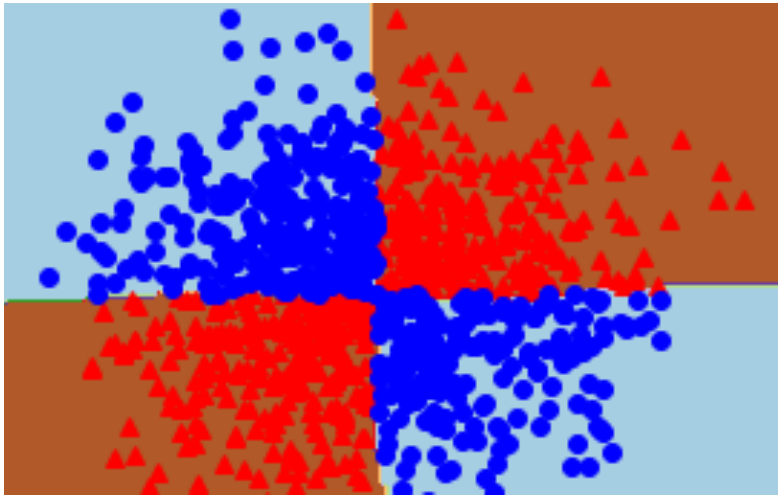
Adversarial Examples

Deep Learning: Bryan Pardo & Patrick O’Reilly, Northwestern University, Spring 2022

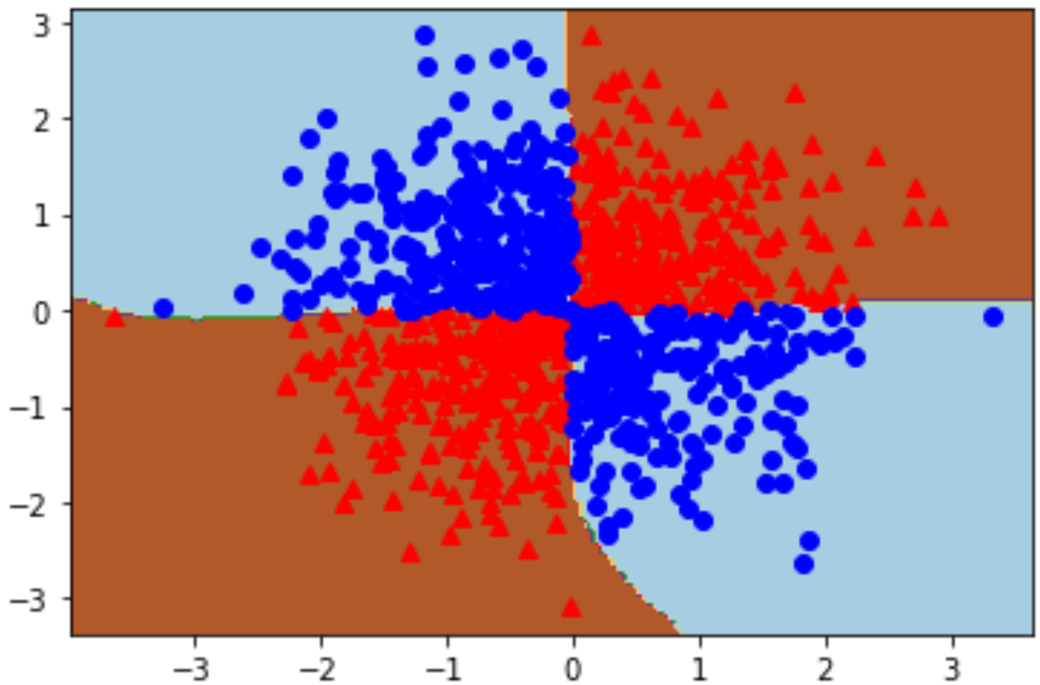
Learned decision surfaces are messy and don’t align with human intuition

Looks good, right?



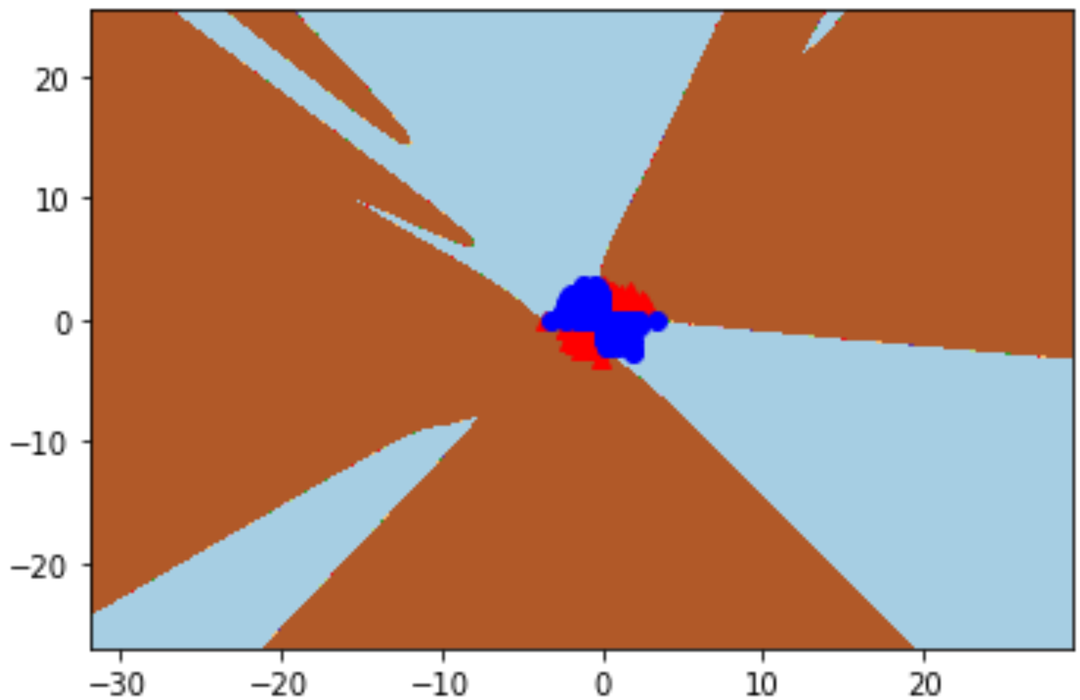
Learned decision surface for XOR problem

Let’s zoom out a little



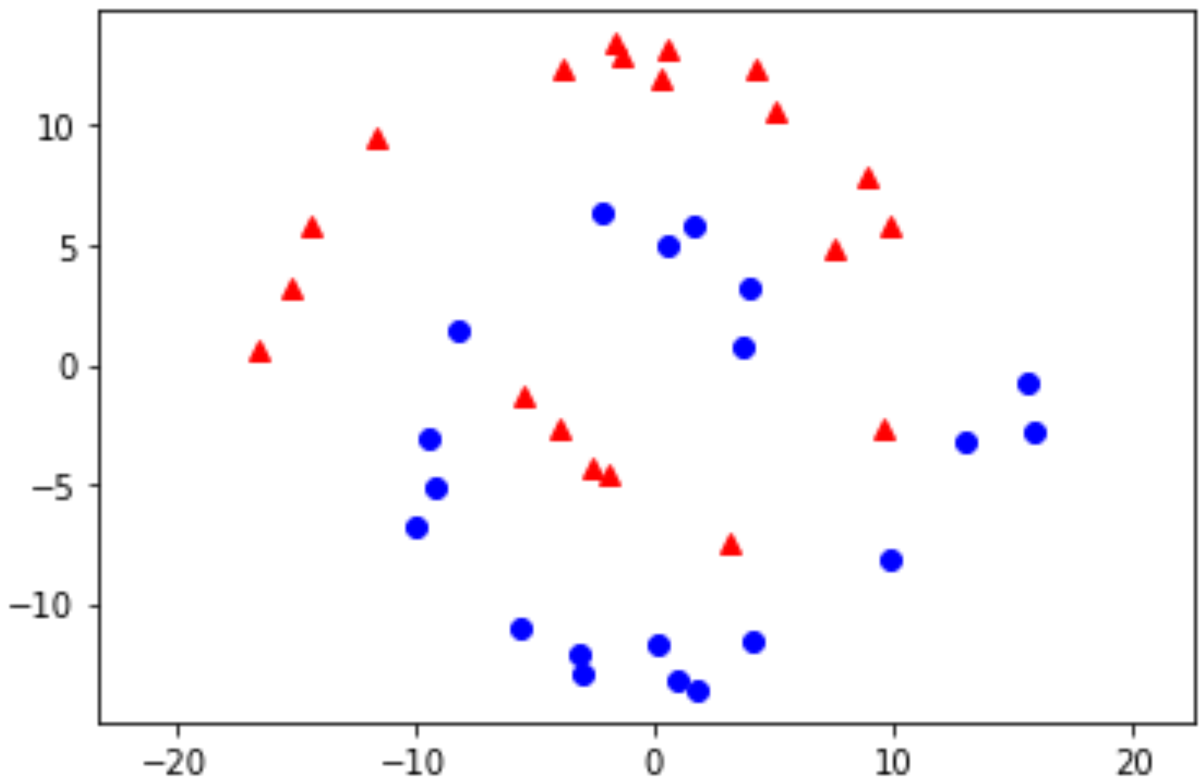
Learned decision surface for XOR problem

Zooming out more…

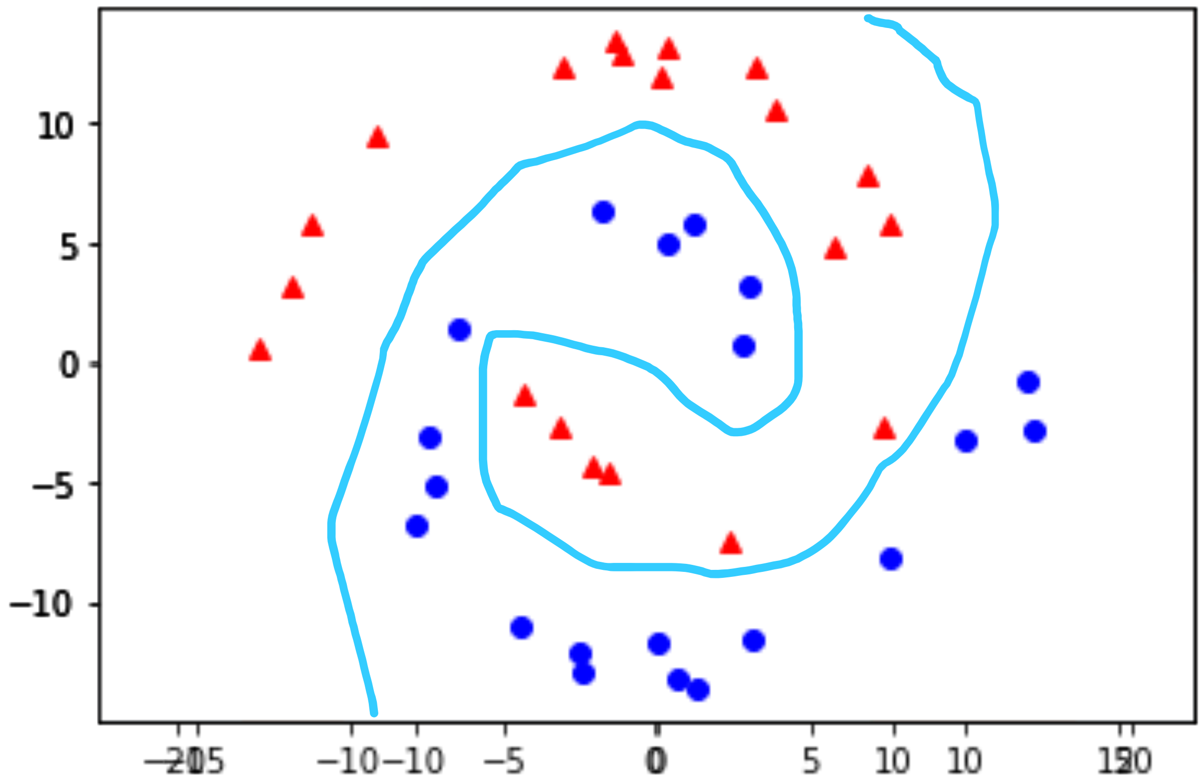


What are these things really learning?

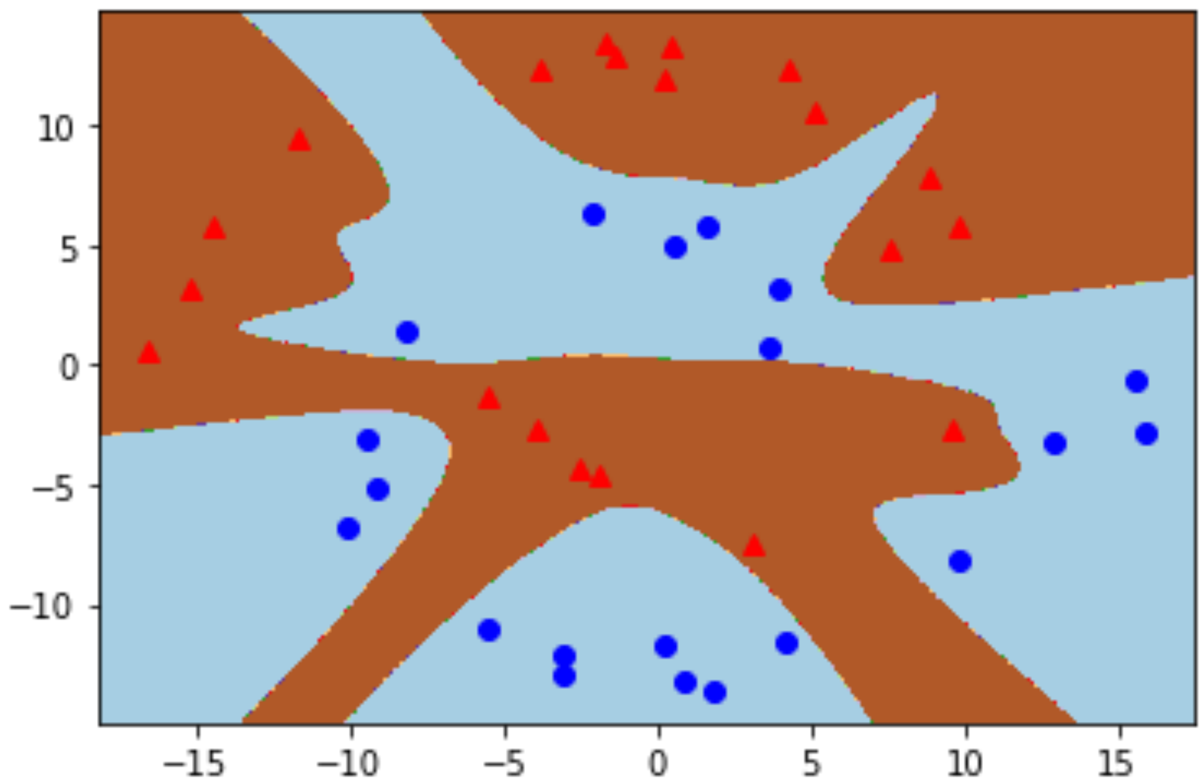
Spirals data



Decision surface a human might draw



Actual decision surface learned by a network

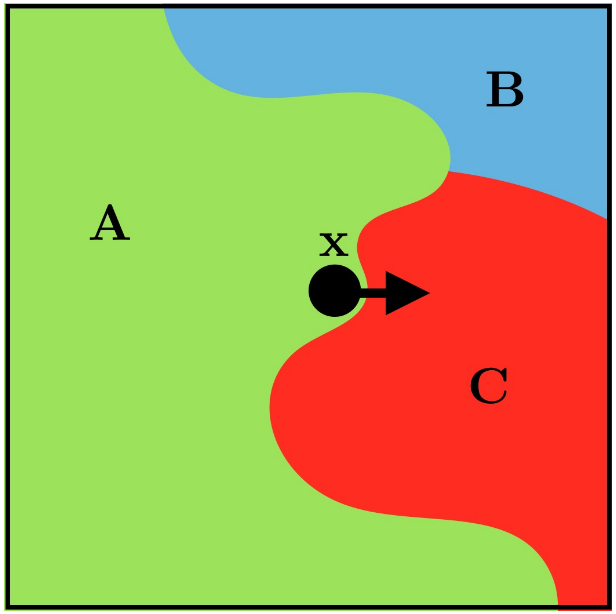


100% perfect accuracy on labeling

These decision surfaces that don’t align with human decision surfaces make networks brittle & easy to fool

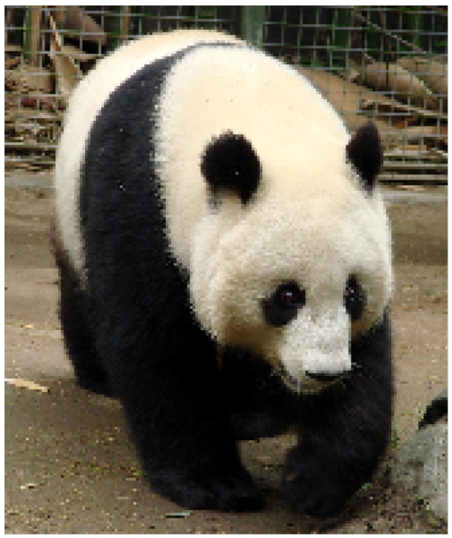
What if we “nudge” an example over the line?

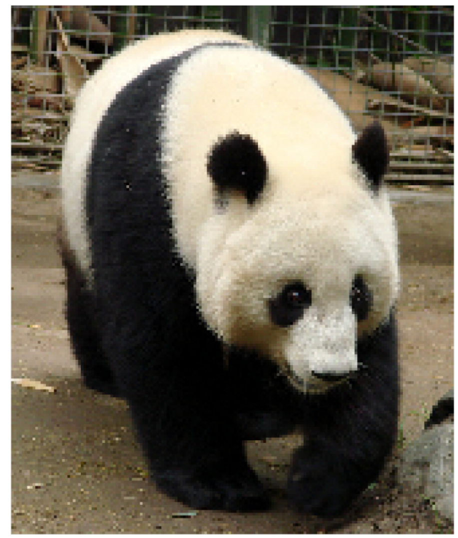
- Gradient descent alters the decision boundary
- Adversarial attacks alter the input
- Do it right and a human won’t see a difference
- …but the machine might really screw up a classification



In 2 dimensions, a bad surface is obvious

- What about in 2 million dimensions?

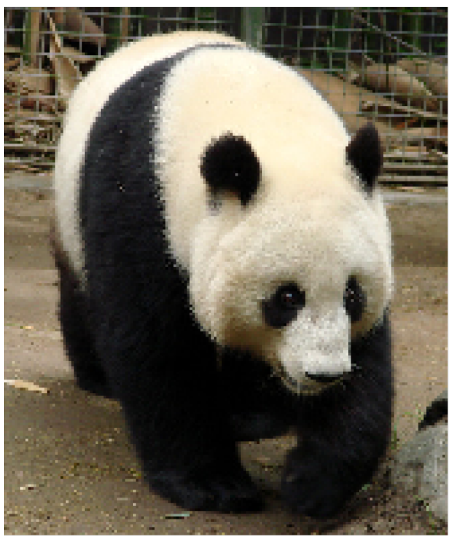




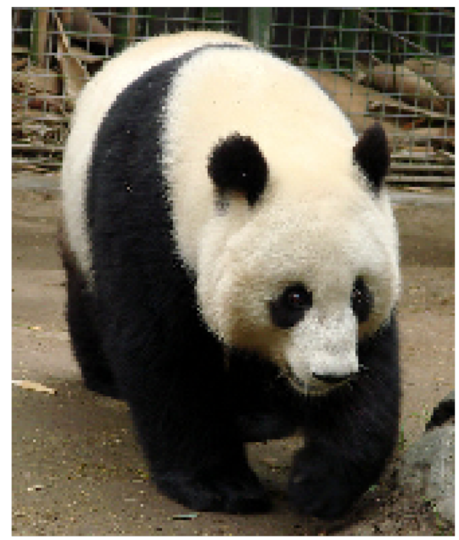
One of these was labeled “panda” by a trained net.
The other was labeled “bucket”. Which is which?

Image from: https://www.borealisai.com/en/blog/advertorch-adversarial-training-tool-implement-attack-and-defence-strategies/

The one on the right is a “perturbed” image



PANDA



BUCKET

Gradient Descent Pseudocode

Initialize $\theta^{(0)}$

Repeat until stopping condition met:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(X, Y; \theta^{(t)})$$

Return $\theta^{(t_{max})}$

$\theta^{(t)}$ are the parameters of the model at time step t

$\nabla_\theta \, L(X, Y; \theta^{(t)})$ is the gradient of the loss function with respect to model parameters $\theta^{(t)}$

η controls the step size

$\theta^{(t_{max})}$ is the set of parameters that did best on the loss function.

Just flip which thing we're optimizing

Initialize $X^{(0)}$

Repeat until stopping condition met:

$$X^{(t+1)} = X^{(t)} + \eta \nabla_X L(X^{(t)}, Y; \theta)$$

Return $X^{(enough)}$

$X^{(t)}$ is an example at time t

$\nabla_X \, L(X^{(t)}, Y; \theta)$ is the gradient of the loss function with respect to example features $X^{(t)}$

η controls the step size

$X^{(enough)}$ is the minimal change needed to flip the category of X

Even Simpler: Fast Gradient Sign method

$$X^{(t+1)} = X^{(t)} + \eta sign(\nabla_X L(X^{(t)}, Y; \theta))$$

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES
Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy, ICLR 2015

Gradient Sign attack

- The pixels are all independent dimensions
- Find the gradient in the pixel space
- Add (clipped) noise along the gradient (a little noise for every pixel)
- Do it right and the image looks the same to the user…

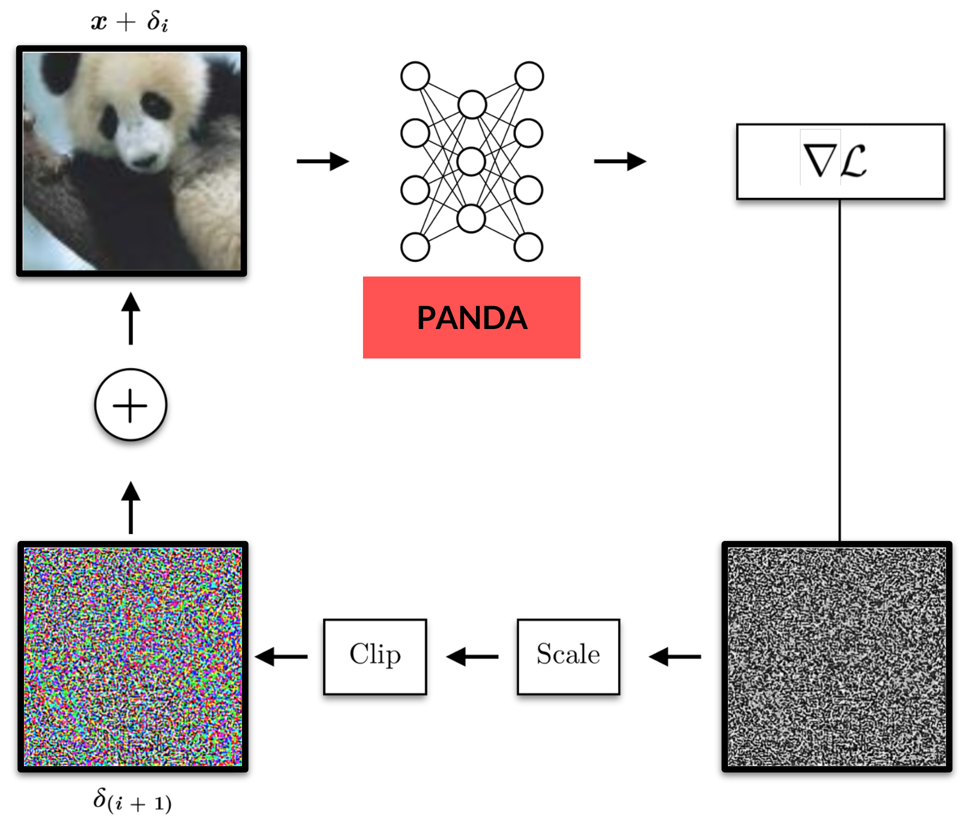
…but looks entirely different to the network.

That same thing in pictures

$f(x + \delta_0)$ = **PANDA**

$f(x + \delta_1)$ = **PANDA**

$f(x + \delta_{...})$ = **PANDA**



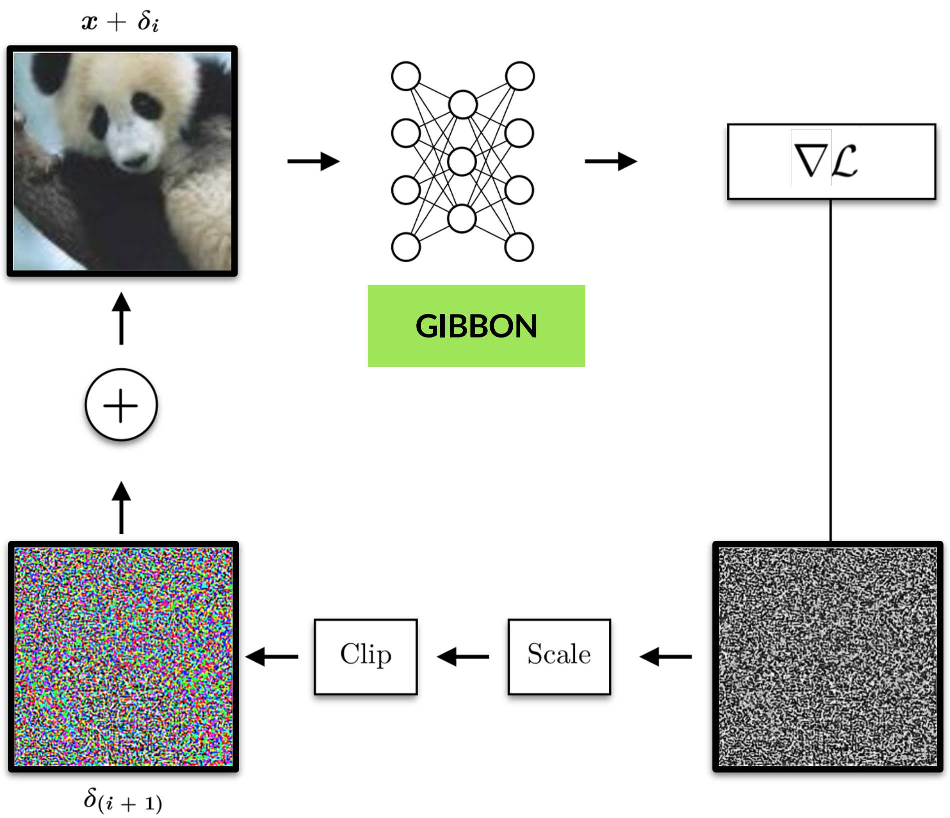
That same thing in pictures

$f(x + \delta_0)$ = **PANDA**

$f(x + \delta_1)$ = **PANDA**

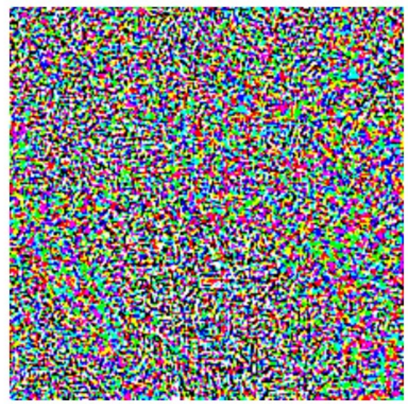
$f(x + \delta_{...})$ = **PANDA**

$f(x + \delta_N)$ = **GIBBON**



Yes, it’s just that easy

$+ .007 \times$



$=$

$\boldsymbol{x}$
“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

(Image from Goodfellow et al 2014)

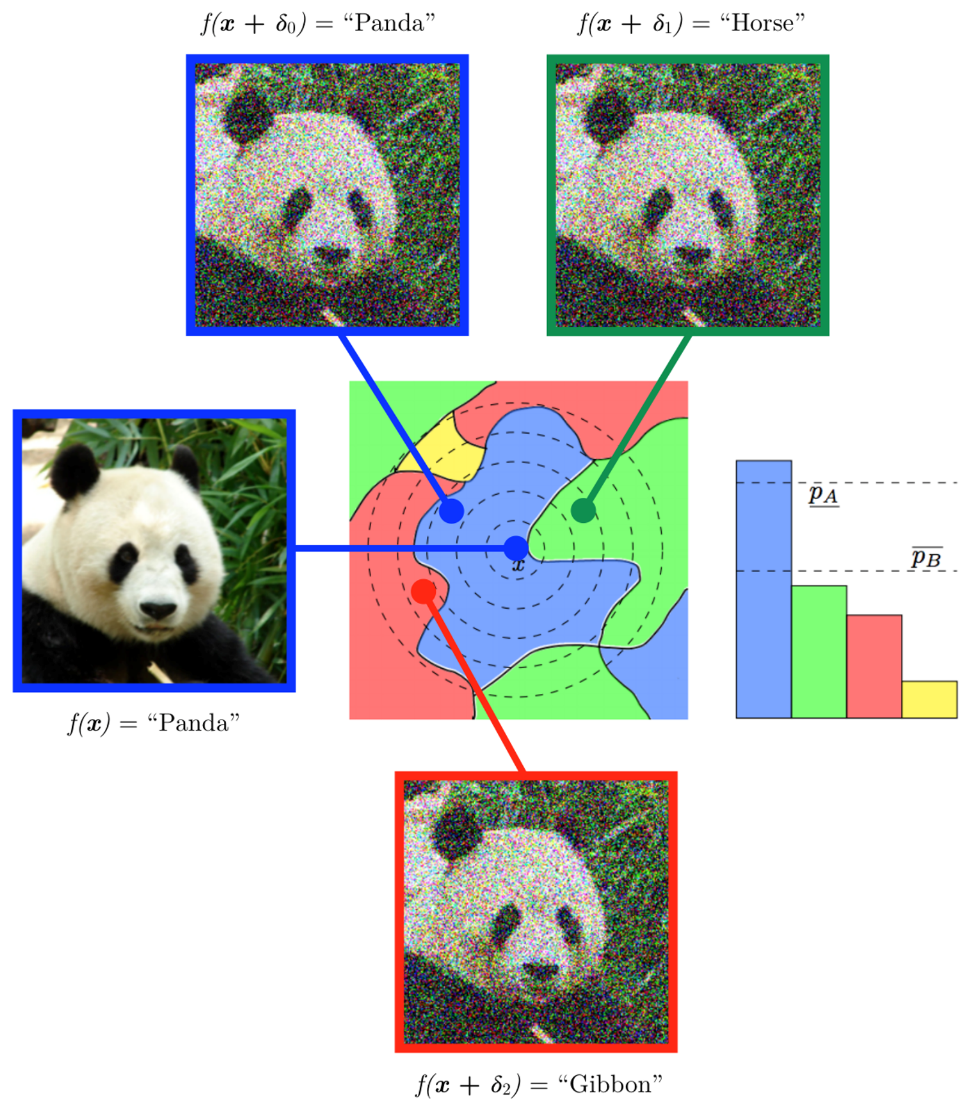
Why…..

- ….does this gradient-based attack make sense?
- …did they use the sign of the gradient multiplied by a fixed step size, instead of the actual gradient?

Defending against attacks

Our problem…

- An attacker can straightforwardly force the classifier to recategorize inputs.
- This could be a big problem for…
  - Self driving cars
  - Verification of identity
  - Etc..
- What can we do about it?



Defenses

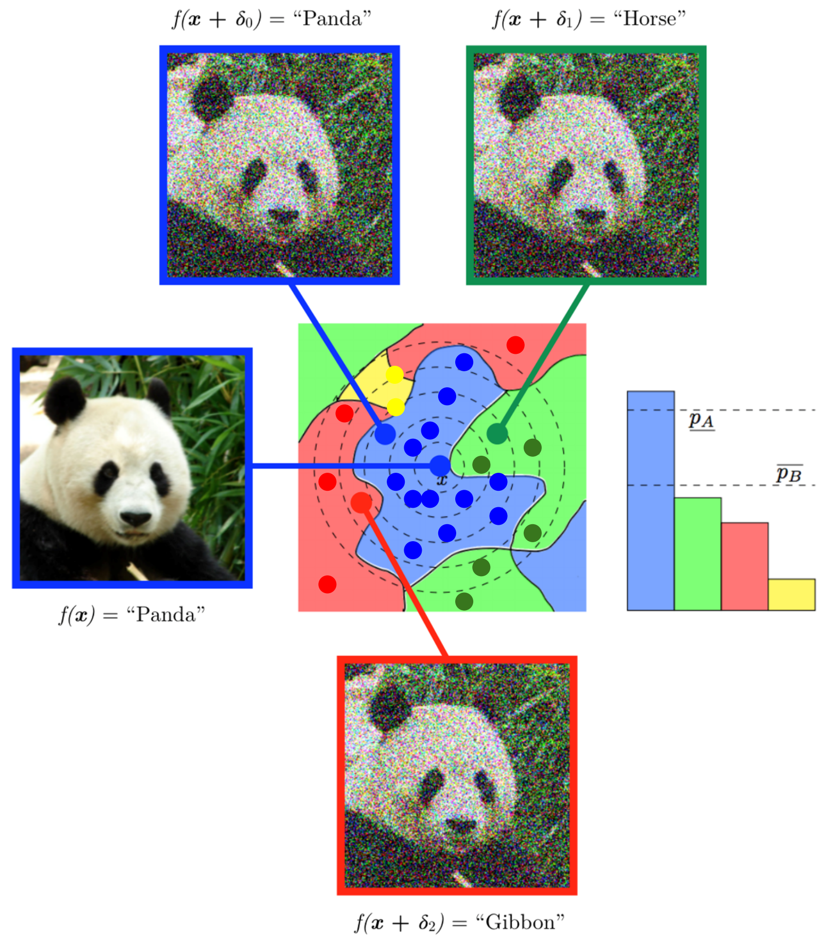
- Security through obscurity (don’t let them see your weights)
  - Can be helpful….not a guarantee. There are black-box attacks.

- Randomly modify the input to screw up the perturbation.
  - At training time (use adversarial examples in training)
  - At inference time (we’ll talk more about this)

- Ensembling
  - Train N different networks with different architectures & training data
  - Use majority voting for classification
  - Hope they can’t attack a majority of them simultaneously

Force them into the open

- If we think that our attacks will be nudges to put images just over the border….
- ..and these nudges are designed to be imperceptible.
- Perhaps we can force them to make a perceptible change if they want to force misclassification…



Randomized smoothing

- At inference, random perturbations are sampled from a Gaussian centered at the input $\boldsymbol{x}$ and a majority vote is taken from the classifier's predictions over these perturbed inputs.