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?

The one on the right is a “perturbed” image.
Gradient Descent Pseudocode

<table>
<thead>
<tr>
<th>Initialize $\theta^{(0)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat until stopping condition met:</td>
</tr>
<tr>
<td>$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(X, Y; \theta^{(t)})$</td>
</tr>
<tr>
<td>Return $\theta^{(t_{max})}$</td>
</tr>
</tbody>
</table>

$\theta^{(t)}$ 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 $\theta^{(t)}$

$\eta$ 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 $X^{(t)}$

$\eta$ controls the step size

$X^{(enough)}$ is the minimal change needed to flip the category of $X$
Even Simpler: Fast Gradient Sign method

\[ X^{(t+1)} = X^{(t)} + \eta \text{sign}(\nabla_x L(X^{(t)}, Y; \theta)) \]
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

\[ f(x + \delta_0) = \text{PANDA} \]
\[ f(x + \delta_1) = \text{PANDA} \]
\[ f(x + \delta_{\infty}) = \text{PANDA} \]
That same thing in pictures

\[
f(x + \delta_0) = \text{PANDA}
\]

\[
f(x + \delta_1) = \text{PANDA}
\]

\[
f(x + \delta_{\ldots}) = \text{PANDA}
\]

\[
f(x + \delta_N) = \text{GIBBON}
\]
Yes, it’s just that easy

\[ x + \epsilon \text{sign}(\nabla_x J(\theta, x, y)) = x \]

- **“panda”**
  - 57.7% confidence

- **“nematode”**
  - 8.2% confidence

- **“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 $x$ and a majority vote is taken from the classifier’s predictions over these perturbed inputs.