Adversarial examples

We introducted Gradient Ascent in our module on interpretation (Interpretation_of_DL_Gradient_Ascent.ipynb)

- We find the input $\mathbf{x}^*$
- That maximizes the value of a particular neuron $\mathbf{y}_{(l),\text{idx},k}$

$$\mathbf{x}^* = \underset{\mathbf{x}=\mathbf{y}_{(0)}}{\text{argmax}}\, \mathbf{y}_{(l),\text{idx},k}$$

In that module, we used the technique to find the input value $\mathbf{x}$ that "maximally excited" $\mathbf{y}_{(l),\text{idx},k}$.

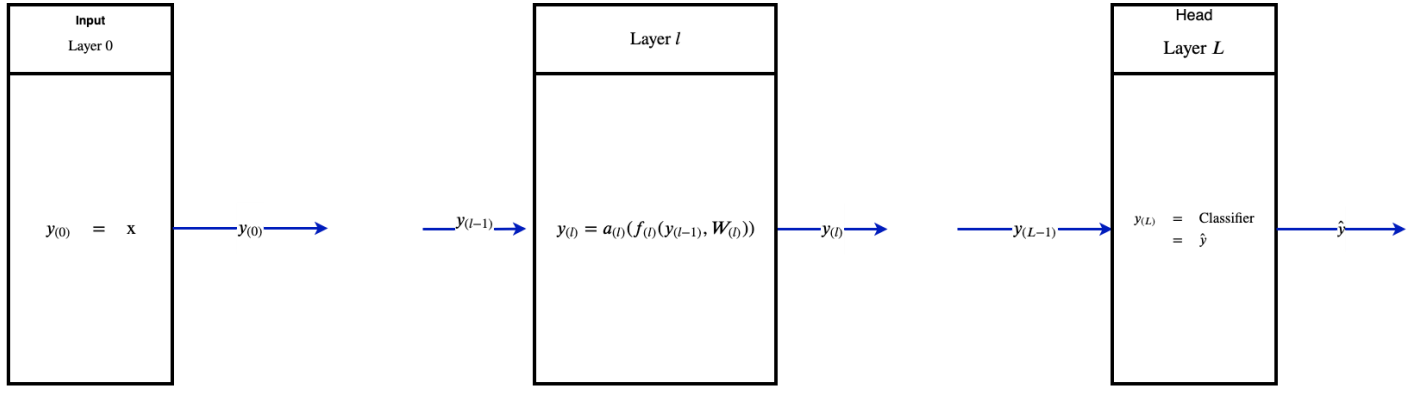
In this module, the neuron we will maximally excite will be in the head layer L.

If layer L is a classifier for a task with classes in set C

- Then $\mathbf{y}_{(L)}$ is a vector of length C
- Where $\mathbf{y}_{(L),j}$ corresponds to the predicted probability that the correct class is C_j
    - denoting the j^{th} class as C_j

$$\mathbf{x}^* = \underset{\mathbf{x}=\mathbf{y}_{(0)}}{\operatorname{argmax}} \mathbf{y}_{(L),j}$$

Layers



That is: we will solve for the $\mathbf{x}^*$

- That is the example that looks like C_j
- More than *any* value in the input domain
- The "perfect C_j"

If C were the class of animals and the domain of $\mathbf{x}$ were images

- This would be like finding "the perfect dog" image
- At least according to the classifier

Pretty innocuous.

But what if we *constrained the optimization*

$$\mathbf{x}^* = \underset{\mathbf{x}=\mathbf{y}_{(0)}}{\operatorname{argmax}} \mathbf{y}_{(L),j}$$
$$\text{subject to}$$
$$\text{looks like}(\mathbf{x}, \mathbf{x}')$$

where

- $\text{looks like}(\mathbf{x}, \mathbf{x}')$
- Is a "closeness" metric that increases when chosen $\mathbf{x}$ is most similar to a specific $\mathbf{x}'$

And what if $\mathbf{x}'$ were from some class $C_{j'} \neq C_j$?

That is

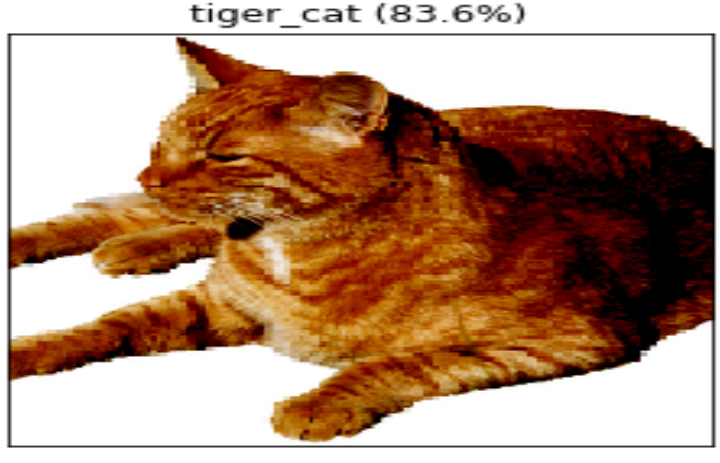
- We find the $\mathbf{x}^*$
- That gets classified with high confidence as being C_j
- But it actually in a different class $C_{j'}$

The $\mathbf{x}^*$ obtained is called an *Adversarial Example*

- One specifically constructed to "fool" the Classifier

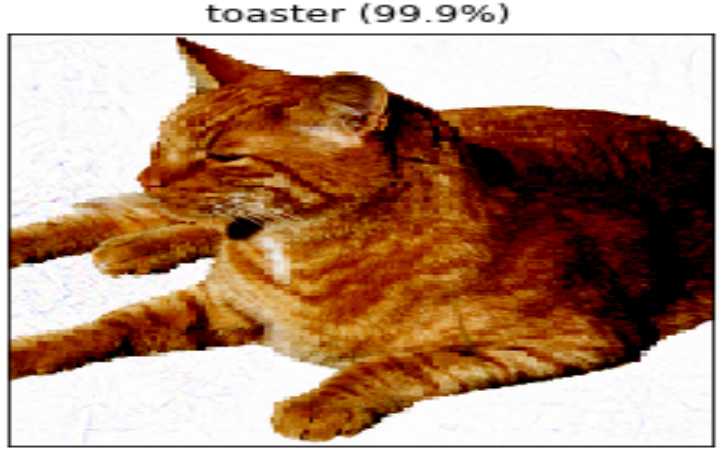
Adversarial examples in action:

What class is this ?



What about this ?

What class is this ?



It's almost certainly a toaster !

That was fun, but is it innocuous ?

What harm can this do ?

Adversarial Stop Sign

So what ? Adversarial Examples 2

"Speed Limit 45"

Eykholt et. all, https://arxiv.org/pdf/1707.08945.pdf

[Robust Physical-World Attacks on Deep Learning Models (https://arxiv.org/abs/1707.08945)]

Remember, the Neural Network has previously been reported to have super-human accuracy !

What's going on here ?

The problem is that we don't actually *know* how the Neural Network recognizes a toaster.

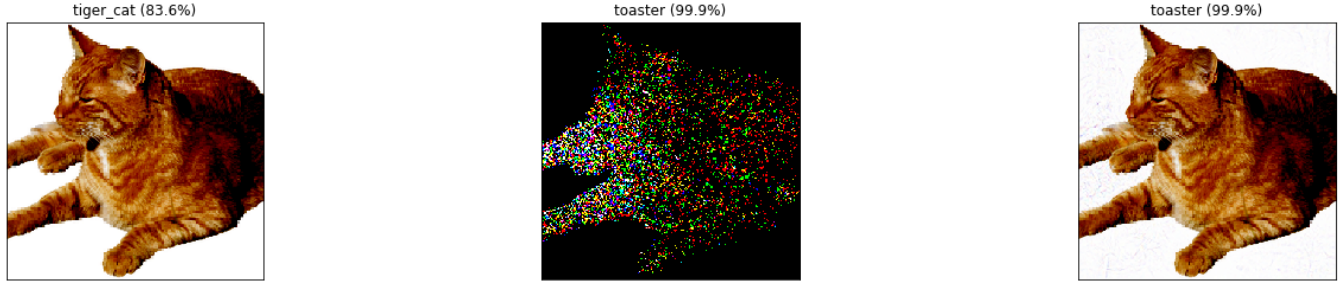
The optimizer has learned to change exactly those input pixels

- That the Neural Network uses to classify a toaster
- And, with enough additional constraints
- The changes are not detectable by the human eye

Here is a visualization of the pixels that were changed

- In order to reclassify the cat (left image)
- As a toaster (right image)

Adversarial Cat to Toaster



It should be clear that Adversarial Examples violate the Fundamental Assumption of Machine Learning

- A test example
- Comes from the same distribution as the Training examples

We are able to fool the Neural Network by asking it to solve a problem for which it wasn't trained.

This highlights an important issue

- Since we don't know how Neural Networks work
- How can we confidently deploy them in the physical world ?

Recall the fundamental assumption of Machine Learning:

- an example from the test set is drawn from the same distribution as the training set

In the case of Adversarial Examples, this condition is not satisfied.

Adversarial Attacks are able to fool an otherwise high quality Neural Network

- Without corrupting any training examples
- Without altering the weights of the Neural Network
- Without giving the Attacker access to any information about the Neural Network

So the attack can occur even on a Neural Network that appears as a "black box" to the attacker

Conclusion

Adversarial Examples are inputs that are crafted for the purpose of "fooling" a Neural Network.

The attacks use the same techniques that are otherwise used to correctly train a network.

This is a significant issue that must be addressed before Neural Networks can be entrusted with tasks that have real-world consquences.

Adversarial Reprogramming

We can extend the Gradient Ascent method to perform even bigger tricks:

- Getting a Classifier for Task 1 to do something completely different !

Can we get an ImageNet Classifier to count squares ? Imagenet

- does not have squares as an input image
- or numbers as an output class

This is called *Adversarial Reprogramming*.

Can I hijack your phone by showing it an image ?

Adversarial Reprogramming

ImageNet Classifier

"panda"

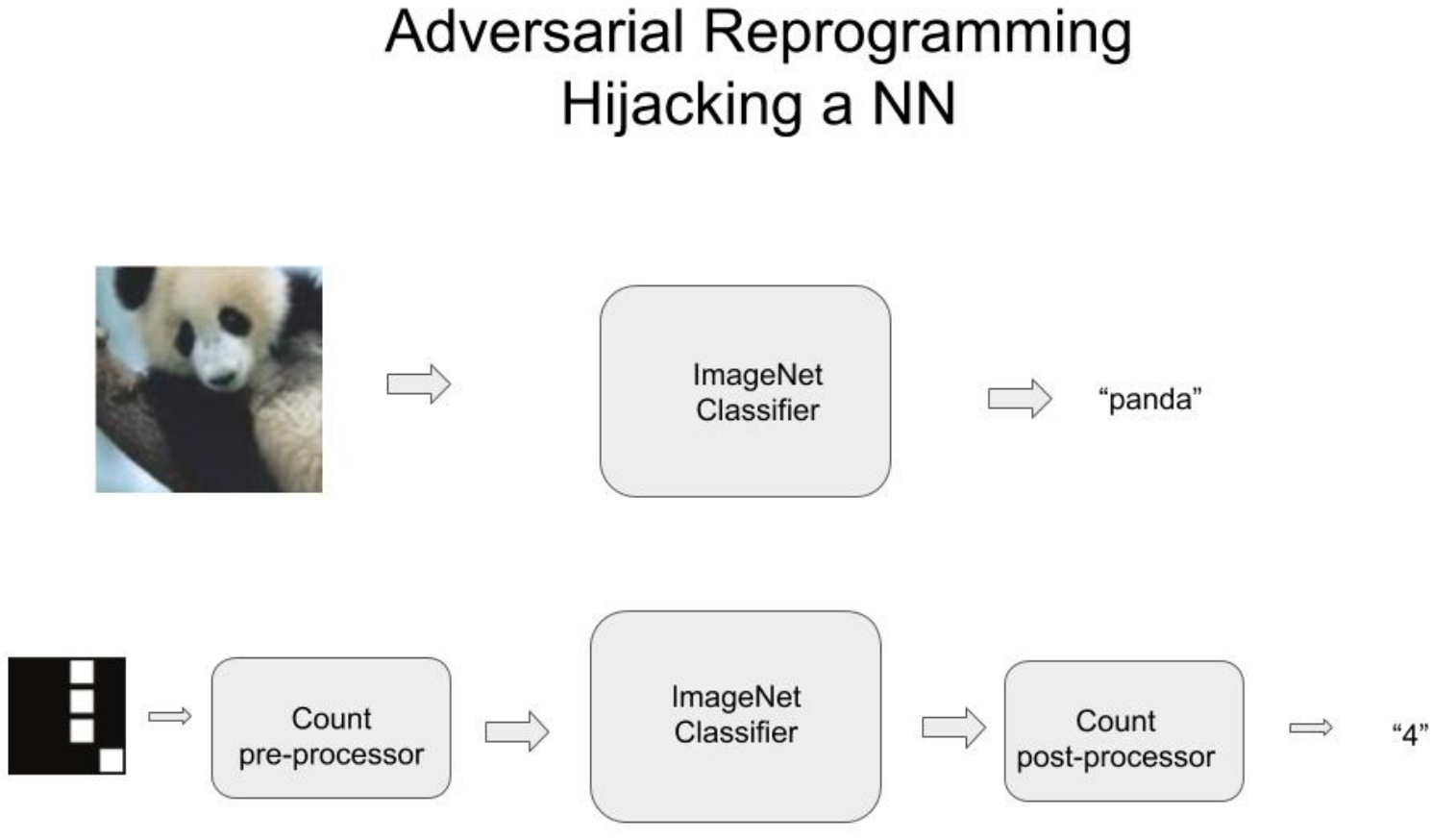
ImageNet Classifier

"4"

Gamaleldin, et. all: https://arxiv.org/abs/1806.11146

Adversarial Reprogramming of Neural Networks (https://arxiv.org/abs/1806.11146)

Here's a pictorial to describe the process:



We refer to our original classifier as solving the Source task.

Our goal is to get the classifier to solve the Target task.

The first issue to address:

- the $(\mathbf{x}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})})$ pairs of the Source task come from a different domain than that of the Target task

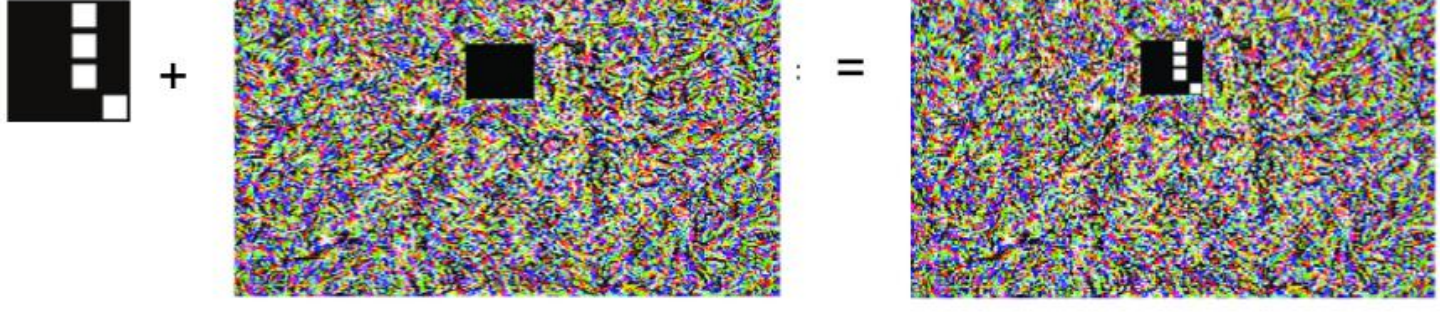
$$\mathbf{X}_{\text{source}}, \mathbf{y}_{\text{source}} : \quad \text{examples for Source task}$$
$$\mathbf{X}_{\text{target}}, \mathbf{y}_{\text{target}} : \quad \text{examples for Target task}$$

We create a simple function h_f to map an $\mathbf{x} \in \mathbf{X}_{\text{target}}$ to an $\mathbf{x} \in \mathbf{X}_{\text{source}}$.

This ensures that the input to the Source task is of the right "type".

Adversarial Reprogramming

Adversarial Program



X_t + W = $\hat{X}_s$

h_f Counting pre-processor

h_f simply embeds the Target input into an image, which is the domain of the Source task.

Similarly, we create a function h_g to map the Target label to a Source Label.

This will ensure that the output of the Source task is of the right type.

Adversarial Reprogramming

(a)

counting	ImageNet
y_{adv}	y
1 square	tench
2 squares	goldfish
3 squares	white shark
4 squares	tiger shark
5 squares	hammerhead
6 squares	electric ray
7 squares	stingray
8 squares	cock
9 squares	hen
10 squares	ostrich

h_g Counting post-processor

Finally, the Cost function to optimize

$$\mathbf{W} = \underset{\mathbf{W}}{\operatorname{argmin}} - \log(p(h_g(\mathbf{y}_t) \mid \tilde{\mathbf{X}}_{\text{source}})) + \lambda ||\mathbf{W}||^2$$

where

$$\begin{aligned}
\tilde{\mathbf{X}}_{\text{source}} &= h_f(\mathbf{W}, \mathbf{X}_{\text{target}}) \\
h_f: & \quad \mathbf{y}_{\text{target}} \mapsto \mathbf{y}_{\text{source}} \quad \text{map source X to target X} \\
h_g: & \quad \mathbf{y}_{\text{target}} \mapsto \mathbf{y}_{\text{source}} \quad \text{map source label y to target label}
\end{aligned}$$

- Given an input in the Target domain $\mathbf{X}_{\text{target}}$
- Transform it into an input $\tilde{\mathbf{X}}_{\text{source}}$ in the Source domain.
- Use the Source Classifier to predict $\tilde{\mathbf{X}}_{\text{source}}$ a label in the Source domain
    - The correct label in the Target domain is $\mathbf{y}_t$
    - This maps to label $h_g(\y_t) in the Source domain

So we are trying to

- maximize the likelihood that the Source classifier creates the encoding for the correct Target label
- subject to constraining the weights $\mathbf{W}$ (the "frame" into which the Target input is placed)

How do we find the frame $\mathbf{W}$ that "reprograms" the Source Classifier ?

By training it of course ! Just plain old ML.

Misaligned objectives

We have framed the problem of Deep Learning as one of defining a Cost function that meets your objectives.

This is not as easy as it sound.

Consider the difference between

- "Maximize profit"
- "Maximize profit subject to legal and ethical constraints"

We (hopefully) don't have to state the additional constraints to a human -- we take it for granted.

Not so with a machine that has not been trained with additional objectives.

AI Safety

- AI Safety = Harmed caused **by** AI
- Some causes:
  - Biased training data
    - Polar bears
  - Objective functions not fully aligned with human goals
    - Consider
      - Maximize reward
      - Maximize reward subject to legal and moral norms
    - Reward Hacking in Reinforcement Learning

In [ ]:
```
print("Done")
```