Neural Network Robustness
About me

• My name is Radoslav Ivanov
  – Call me Rado

• Undergrad degree in CS and ECON from Colgate in 2011

• Got my PhD in CS from UPenn in 2017

• My research is on safe and secure autonomous systems
  – Verification of neural networks
  – Attack-resilient sensor fusion
  – Context-aware detection and estimation

• Started at RPI in Jan. 2022
Reading

• Papers (all available online)
Impressive Progress in Autonomy

Control

Boston Dynamics

Perception

YOLO v. 3

Learning

Kormushev, Calinon, Caldwell, IROS’10

JPL-Caltech, DARPA Robotics Challenge

Zhu, Zhou, Daniilidis, ICCV'15

DeepMind
National Security

Iran says it downed U.S. stealth drone; acknowledges aircraft downing

By Greg Jaffe and Thomas E. Ricks, December 04, 2011

A secret U.S. surveillance drone that went missing last week in western Afga has crashed in Iran, in what may be the first case of such an aircraft ending an adversary.

Iran's news agencies asserted that the nation's defense forces brought down the U.S. drone, a UH-1Y Venom helicopter. GAO's investigation of the crash has concluded that the Iranian drone was shot down by Iranian forces and that the crash site was located near the site of the crash of an F-15E Strike Eagle in Syria.

The U.S. military has not confirmed the report of the crash and has not ruled out the possibility of a hostile attack. The U.S. military has not confirmed the report of the crash and has not ruled out the possibility of a hostile attack. The U.S. military has not confirmed the report of the crash and has not ruled out the possibility of a hostile attack.
Neural Network (NN) Vulnerabilities

• Neural networks increasingly used in safety-critical systems

Perception (autonomous cars)

Control (air traffic avoidance)

Wong et al., CoRL’19

Katz et al., CAV ‘17

• Safety concerns discovered in both domains

Goofellow et al., ICRL’15

Table 2: Verifying properties of the ACAS Xu networks.

<table>
<thead>
<tr>
<th>Networks</th>
<th>Result</th>
<th>Time</th>
<th>Stack</th>
<th>Splits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>41 UNSAT</td>
<td>394517</td>
<td>47</td>
<td>1522384</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>4 TIMEOUT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>35 SAT</td>
<td>82419</td>
<td>44</td>
<td>284515</td>
</tr>
<tr>
<td>$\phi_4$</td>
<td>42 UNSAT</td>
<td>28156</td>
<td>22</td>
<td>52080</td>
</tr>
<tr>
<td></td>
<td>42 UNSAT</td>
<td>12475</td>
<td>21</td>
<td>23940</td>
</tr>
</tbody>
</table>
Problem: How do we know car won’t crash?

- How do we build safe algorithms?
- How do we analyze algorithms?
- What about “black-box” components such as neural networks?
- How do we convince other people car is safe (assurance argument)?
Feedforward Neural Networks

• Also known as multi-layer perceptrons
  – Old name, at least from the 1960’s

• The term “deep neural networks” is essentially rebranding
  – Modern networks are deeper than ever, however
  – Term “neural” is (very) loosely inspired by neuroscience

• The term “feedforward” means that computation happens from left to right in network, without any feedback
NN terminology

Inputs: $x_1, x_2, x_3$

Hidden Layers

Outputs: $y_1, y_2$

Neuron

First Layer

Hidden Layers

Outputs
NN Functionality

\[ n = a(w_1 y_1 + w_2 y_2 + w_3 y_3) \]

Common activations include:

- Relu: \( a(x) = \max(0, x) \)
- sigmoid: \( a(x) = \frac{1}{1+e^{-x}} \)
NNs as functions

• Standard ML model

\[
y = f(x; \theta)
\]

- \(x\) are the inputs (e.g., pixels), \(y\) are the outputs (e.g., labels), \(\theta\) are the parameters to be optimized

• Can be written as a composition of its \(L\) hidden layers

\[
f(x; \theta) = f_L \circ f_{L-1} \circ \cdots \circ f_1(x)
\]

• Typically trained by minimizing a loss function on a labeled training set \(\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}\)

- E.g., least squares

\[
\min_{\theta} \sum_{i=0}^{N} (y_i - f(x_i; \theta))^2
\]
Intriguing properties of neural networks
Overview

• This paper started a frantic search for improving neural network robustness
  – Even the authors were probably surprised by the paper’s huge impact

• Essentially, they found that even though NNs perform well on data from the “distribution”, they are not robust to even tiny perturbations
  – This means that NNs are highly unpredictable and cannot be used in safety-critical systems (e.g., autonomous cars) without understanding them better
Setup

- Pretrained NNs on several benchmarks using different algorithms
  - MNIST
  - ImageNet
- Test how sensitive these NNs are to small input perturbations
Input-Output Sensitivity

- NN input-output sensitivity is obviously important
  - E.g., don’t want the output to change if you move the camera by 0.1 inch

- Some spuriousness introduced by the fact that NNs typically trained with cross-entropy
  - Tries to approximate the conditional distribution of the labels given an example
  - Bound to assign non-zero probability to spurious classes
  - Spuriousness did not exist in classical computer vision
    - Heavy use of filtering and smoothening improves robustness
    - We implicitly assumed that such spuriousness is low for NNs also
Adversarial Examples

• Turns out that NNs are not at all robust to small perturbations
  – For any NN and any correctly classified image, one can find an imperceptibly small perturbation that causes the NN to misclassify the image
  – Especially true for more complex datasets
  – Perturbed image is called an adversarial example
  – Some techniques have been developed to alleviate this issue but it is still very much an open problem
Formal Setup

• Let $f: \mathbb{R}^m \to \{1, \ldots, k\}$ be a pre-trained classifier

• We are given an image $x$ and a target label $l$
  – Note that $l$ may or may not be the ground-truth label
  – Turns out this works for any $l$!

• Goal: find the minimum perturbation $r$ such that $f(x + r) = l$

• Formally,

\[
\min_{r} \quad \|r\|_2 \\
\text{subject to} \quad f(x + r) = l
\]

• Of course, $x + r$ has to be an image, i.e., $x + r \in [0,255]^m$
  – Or normalized to $[0,1]^m$
Solving the minimization problem

• Of course, solving the minimization problem is hard because NNs
• Instead use gradient descent, but with \( r \) being the optimization variable
  – minimize original loss: \( \text{loss}_f(x + r, l) \)
• To find a *small* \( r \), also add an extra term \( c|r| \), with \( c \) fixed
  – Find minimum \( c \) for which you can find a good \( r \)
• Final loss is
  \[
  c|r| + \text{loss}_f(x + r, l)
  \]
• How do you choose \( c \)?
  – Linear search is a good start
Results Overview

• For almost every NN and every example, an almost imperceptible adversarial example was found
  – The *almost* part removed in later papers that came up with better ways of solving the optimization problem

• Adversarial examples are transferable
  – An adversarial example for one NN is also misclassified by another
  – Also true even if second NN trained on a disjoint training set
  – Suggests that adversarial examples are a fundamental artifact of the distribution/training method
Examples, ImageNet

All images go from correctly classified to “ostrich”
Examples, MNIST

(a) Even columns: adversarial examples for a linear (FC) classifier (std-dev=0.06)

(b) Even columns: adversarial examples for a 200-200-10 sigmoid network (stddev=0.063)
Adversarial Example Transferability

<table>
<thead>
<tr>
<th></th>
<th>FC10(10^{-4})</th>
<th>FC10(10^{-2})</th>
<th>FC10(1)</th>
<th>FC100-100-10</th>
<th>FC200-200-10</th>
<th>AE400-10</th>
<th>Av. distortion</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC10(10^{-4})</td>
<td>100%</td>
<td>11.7%</td>
<td>22.7%</td>
<td>2%</td>
<td>3.9%</td>
<td>2.7%</td>
<td>0.062</td>
</tr>
<tr>
<td>FC10(10^{-2})</td>
<td>87.1%</td>
<td>100%</td>
<td>35.2%</td>
<td>35.9%</td>
<td>27.3%</td>
<td>9.8%</td>
<td>0.1</td>
</tr>
<tr>
<td>FC10(1)</td>
<td>71.9%</td>
<td>76.2%</td>
<td>100%</td>
<td>48.1%</td>
<td>47%</td>
<td>34.4%</td>
<td>0.14</td>
</tr>
<tr>
<td>FC100-100-10</td>
<td>28.9%</td>
<td>13.7%</td>
<td>21.1%</td>
<td>100%</td>
<td>6.6%</td>
<td>2%</td>
<td>0.058</td>
</tr>
<tr>
<td>FC200-200-10</td>
<td>38.2%</td>
<td>14%</td>
<td>23.8%</td>
<td>20.3%</td>
<td>100%</td>
<td>2.7%</td>
<td>0.065</td>
</tr>
<tr>
<td>AE400-10</td>
<td>23.4%</td>
<td>16%</td>
<td>24.8%</td>
<td>9.4%</td>
<td>6.6%</td>
<td>100%</td>
<td>0.086</td>
</tr>
<tr>
<td>Gaussian noise, stddev=0.1</td>
<td>5.0%</td>
<td>10.1%</td>
<td>18.3%</td>
<td>0%</td>
<td>0%</td>
<td>0.8%</td>
<td>0.1</td>
</tr>
<tr>
<td>Gaussian noise, stddev=0.3</td>
<td>15.6%</td>
<td>11.3%</td>
<td>22.7%</td>
<td>5%</td>
<td>4.3%</td>
<td>3.1%</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Towards deep learning models resistant to adversarial attacks
Overview

• After all the robustness issues were found, researchers got busy trying to find solutions
  – Can we train adversarially robust NNs?
• Adversarial training was one of the first major methods and remains the building block for most methods today
  – However, the benefits are only significant in the case of MNIST
  – Problem is still very much open in general
Adversarial Robustness High-Level Goal

• In standard training (left), we find any decision boundary that maximally separates the training points
  – Boundary typically does not enforce robustness (middle)

• The goal is train in a way that produces a decision boundary that is robust around each training point (right)
Adversarial Training Setup

• Reminder: in standard training, we try to minimize the expected loss over the training set
  \[
  \min_\theta \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[ L(\theta, (x, y)) \right]
  \]

• In adversarial training, the goal is to minimize the loss not only over the training set, but also over a box around each point from the training set

• In particular, we want to minimize the worst-case loss over any perturbation within that box
  \[
  \min_\theta \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[ \max_{r \in \mathcal{R}} L(\theta, (x + r, y)) \right]
  \]

• The set \( \mathcal{R} \) can in theory be any set
  – Usually we take it to be an \( L_\infty \) ball
The inner optimization problem is essentially what the adversarial attacks do:

\[
\min_{\theta} \mathbb{E}_{(x,y) \sim D} \left[ \max_{r \in \mathcal{R}} L(\theta, (x + r, y)) \right]
\]

It makes sense to train NNs that are robust to these attacks by minimizing the quantity the attacks try to maximize.

Note that this problem is quite narrow in a sense:

- Unclear if we want *robust* NNs or NNs that work well on all examples in our distribution
- However, the distinction between a *distribution* and a perturbation-induced-distribution is blurry
First, investigate adversarial examples

- Use projected gradient descent (PGD) to find adversarial examples and record their loss

\[ x^{t+1} = x^t + \alpha \nabla_x L(\theta, (x, y)) \]

\[ x^{t+1} = \text{clip}_{x+\epsilon}[x^{t+1}] \]

- Can also use sign of gradient
- Initialize \( x^0 \) randomly around a training example \( x \)
Zoo of Adversarial Examples

- It seems that most adversarial examples lead to the same loss
  - Authors claims this means inner optimization problem is tractable, which is a questionable claim
Adversarial Training Algorithm

- Use gradient descent as before (combined with PGD)

$$\min_{\theta} \mathbb{E}_{(x,y) \sim D} \left[ \max_{r \in \mathcal{R}} L(\theta, (x + r, y)) \right]$$

- For a given example $x$
  1. Perform $k$ iterations of PGD
     a) Here we are optimizing $r$, keeping $\theta$ constant
  2. Perform gradient descent using $x + r$

- Of course, this is done for a mini-batch as before

- Hyperparameters are $k$, $\epsilon$ (robustness ball) and $\alpha$ (PGD step size)
Experimental Setup

• MNIST
  – $k = 40$
  – $\alpha = 0.01$ (used with the gradient sign)
  – $\epsilon = 0.3$ (out of 1)
  – 3-layer CNN (2 convolutional + 1 fully-connected)

• CIFAR10
  – $k = 20$
  – $\alpha = 2$ (out of 255)
  – $\epsilon = 8$ (out of 255)
  – ResNet architecture (quite large...)
Results, MNIST

- Very high adversarial accuracy even for $\epsilon = 0.3$
  - Natural accuracy is 98.8%, down from 99.2% with no adversarial training, so a very minor difference
  - FGSM is PGD with 1 step

<table>
<thead>
<tr>
<th>Method</th>
<th>Steps</th>
<th>Restarts</th>
<th>Source</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>98.8%</td>
</tr>
<tr>
<td>FGSM</td>
<td>-</td>
<td>-</td>
<td>A</td>
<td>95.6%</td>
</tr>
<tr>
<td>PGD</td>
<td>40</td>
<td>1</td>
<td>A</td>
<td>93.2%</td>
</tr>
<tr>
<td>PGD</td>
<td>100</td>
<td>1</td>
<td>A</td>
<td>91.8%</td>
</tr>
<tr>
<td>PGD</td>
<td>40</td>
<td>20</td>
<td>A</td>
<td>90.4%</td>
</tr>
<tr>
<td>PGD</td>
<td>100</td>
<td>20</td>
<td>A</td>
<td>89.3%</td>
</tr>
<tr>
<td>Targeted</td>
<td>40</td>
<td>1</td>
<td>A</td>
<td>92.7%</td>
</tr>
<tr>
<td>CW</td>
<td>40</td>
<td>1</td>
<td>A</td>
<td>94.0%</td>
</tr>
<tr>
<td>CW+</td>
<td>40</td>
<td>1</td>
<td>A</td>
<td>93.9%</td>
</tr>
</tbody>
</table>
• Adversarial robustness goes up from ~3% to ~45%  
  – However, natural accuracy drops from 95% to 87%  
  – Note that wide model (i.e., more neurons) performs a bit better  
  • More on this next

### CIFAR10

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Wide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural</td>
<td>92.7%</td>
<td>95.2%</td>
</tr>
<tr>
<td>FGSM</td>
<td>27.5%</td>
<td>32.7%</td>
</tr>
<tr>
<td>PGD</td>
<td>0.8%</td>
<td>3.5%</td>
</tr>
</tbody>
</table>

### CIFAR10 (Training Loss)

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Wide</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Standard training</td>
<td>0.00357</td>
<td>0.00371</td>
</tr>
<tr>
<td>(b) FGSM training</td>
<td>0.0115</td>
<td>0.00557</td>
</tr>
<tr>
<td>(c) PGD training</td>
<td>1.11</td>
<td>0.0218</td>
</tr>
<tr>
<td>(d) Training Loss</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Capacity vs Robustness

- Higher-capacity models appear to be more robust also
  - Unclear if this is because they are easier to train or because smaller models cannot be made robust
  - Higher-capacity = double # of filters and FC neurons
Conclusion

- Adversarial training helps but it does not solve the problem
  - CIFAR10 is not even the hardest dataset
- Not only is robustness not very high but there is also a hit in standard accuracy
  - Cannot talk about robustness without accuracy: a coin flip is the most robust classifier (why?)
- Many, many papers since the Madry paper
  - There’s a leaderboard here: https://robustbench.github.io/
So what?

• Maybe we don’t care about robustness since those examples are carefully crafted and will never appear in real life

• Sure, but...

1. There’s something unsettling about the fact that the NN doesn’t work on images that look exactly the same as normal ones (to humans)

2. Neural networks are not robust to natural perturbations (day vs. night, etc.), which suggests the problem is bigger than contrived adversarial examples
Robust physical-world attacks on deep learning visual classification

- Modify real-world objects in small ways that wouldn’t fool a human but fool classifiers used in autonomous cars

<table>
<thead>
<tr>
<th>Distance/Angle</th>
<th>Subtle Poster</th>
<th>Subtle Poster Right Turn</th>
<th>Camouflage Graffiti (LISA-CNN)</th>
<th>Camouflage Art (GTSRB-CNN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5' 0°</td>
<td>STOP</td>
<td></td>
<td>STOP</td>
<td>STOP</td>
</tr>
<tr>
<td>5' 15°</td>
<td>STOP</td>
<td></td>
<td>STOP</td>
<td>STOP</td>
</tr>
<tr>
<td>10' 0°</td>
<td>STOP</td>
<td></td>
<td>STOP</td>
<td>STOP</td>
</tr>
<tr>
<td>10' 30°</td>
<td>STOP</td>
<td></td>
<td>STOP</td>
<td>STOP</td>
</tr>
<tr>
<td>40' 0°</td>
<td>STOP</td>
<td></td>
<td>STOP</td>
<td>STOP</td>
</tr>
</tbody>
</table>

All misclassified as Speed Limit 45 signs