Training robust neural networks

Ryan Belkhir, Emma Covili, Léonard de la Seiglière

Data science project
Master IASD
Titans

December 14, 2021

Plan

Principle

What is it: slightly modify the input of a model to trick it into making false predictions.
Why is it dangerous: most of the time, the perturbations are imperceptible to humans, but it turns out models can produce highly confident yet false results to such altered inputs.

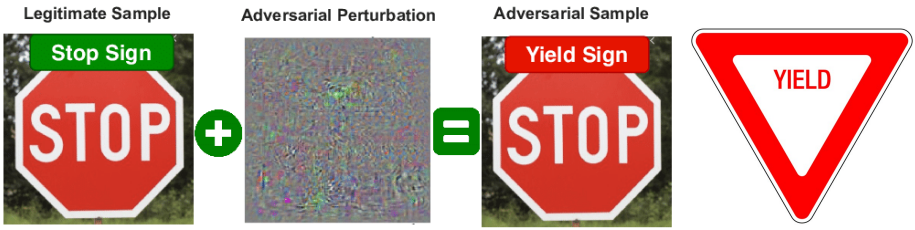


Figure: Example of a problematic situation for autonomous cars

2. Attack and Defense

Quick reminders

- **Targeted vs Untargeted:**
  - The adversary can control the output label of the adversarial image i.e. the output label will be a specific class.
  - The adversary cannot control the output label i.e. the output label will be any class but the true one.
- **Blackbox vs Whitebox:**
  - The adversary does not have access to information like model architecture, algorithm, training dataset and parameters but can probe the model with an input to observe the output.
  - The model is available to the attacker allowing exploitation of gradient of the loss function with respect to the input to form adversarial samples.

FGSM Attack

Fast Gradient Sign Method

One of the simplest yet very efficient method of generating adversarial perturbations. It is a *white-box* attack that can produce samples to cause *targeted* or *untargeted* misclassification.

How it works:

- perturbations are calculated as: $\eta = \epsilon sign(\nabla_x L_\theta(x, y))$

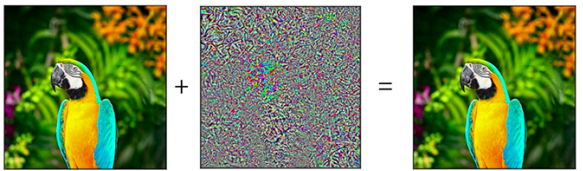


Figure: FGSM attack for a macaw image

PGD Attack

Projected Gradient Descent

It is a *white-box* attack.

How it works:

- $x_{t+1} = \Pi_{B(0,\epsilon)}(x_t + \eta sign(\nabla_x L_\theta(x, y))$

Figure: Predicted labels for a FGSM (left) and PGD (right) attacked network

Adversarial training

Adversarial training

How it works:

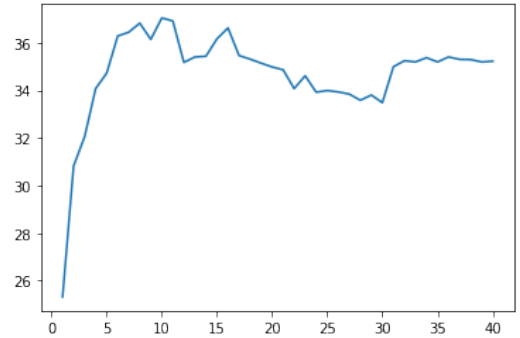
- Simply putting the PGD attack inside the training loop.

What we did:

- Choose the attack(s) to be used in Adversarial Training, we selected **only PGD** ℓ_∞
- Tune the hyperparameters of the PGD attack, we selected $\alpha = 0.01$ **and** $\epsilon = 0.03$
- Test several networks, we selected **ResNet and MNasNet**
- Prevent overfitting with **stop learning**

Attacks results

Results: We train our ResNet with $\epsilon = 0.03$ and $\alpha = 0.01$.



Problem: With very deep networks like ResNet, there is a lot of **overfitting**.
Solution: *Rice, Wong & Zico Kolter, 2020 : Overfitting in adversarially robust deep learning*

Summary of the methods

	FGSM	PGD ℓ_2	PGD ℓ_∞
Vanilla CNN	2.3%	1.3%	0.4%
Adversarial trained CNN	34.8%	34.4%	26.6%
Adversarial trained ResNet	42.3%	41.2%	37.1%
Adversarial trained MnasNet	46.78%	56.3%	41.9%

Table containing the different accuracies for the different methods.

3. What we are working on

Carlini & Wagner attack

How it works:

- CW attack is a targeted attack
- Principle:

$$\begin{aligned} \text{minimize} \quad & \mathcal{D}(x, x+\delta) \\ \text{such that} \quad & C(x+\delta) = t \\ & x+\delta \in [0,1]^n \end{aligned}$$

  Where C is the classification function

- Reformulation

$$\begin{aligned} \text{minimize} \quad & \mathcal{D}(x, x+\delta) \\ \text{such that} \quad & f(x+\delta) \leq 0 \\ & x+\delta \in [0,1]^n \end{aligned}$$

Carlini & Wagner attack

Final formulation

- minimize $\left\|\frac{1}{2}(\tanh(w)+1)-x\right\|_2^2 + c \cdot f\left(\frac{1}{2}(\tanh(w)+1)\right)$ with f defined as

$$f\left(x'\right) = \max\left(\max\left\{Z\left(x'\right)_i : i \neq t\right\} - Z\left(x'\right)_t, -\kappa\right)$$

  Where $Z(x)$ represents the second to last layer layer : they are the raw unnormalized probabilities for each class

- Problem we have to find c, a hyper parameter to optimize. Usually choose the smallest c that missclassifies the input

Carlini & Wagner attack

How to choose a taget class :

- Select the target class uniformly at random among incorrect labels
- Perform the attack against all incorrect classes and report the target class that was the *least difficult* to attack
- Perform the attack against all incorrect classes and report the target class that was the *most difficult* to attack

Randomized Smoothing

How it works :

- Transform any arbitrary base classifier f into a new "smoothed classifier" g that is certifiably robust in ℓ_2 norm.
- For any input image x, $g(x)$ returns the most probable prediction by the base classifier f of random Gaussian corruptions of x.

$$g(x) = \mathit{argmax}_{c \in \mathcal{Y}} \mathbb{P}(f(x + \epsilon) = c)$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$

- The smoothed classifier g possesses a desirable property that the base classifier may lack: one can verify that g's prediction is constant within an ℓ_2 ball around any input x, simply by estimating the probabilities with which f classifies random Gaussian corruptions of x as each class.

Randomized Smoothing

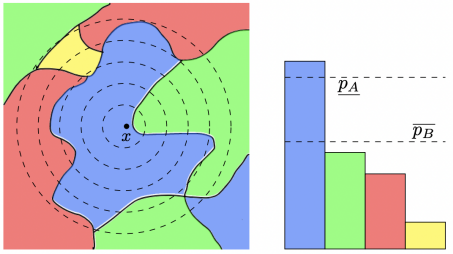


Figure: Evaluating the smoothed classifier at an input x.

- However if f is a neural network, it is not possible to *exactly* compute the probabilities with which f classifies $\mathcal{N}(0, \sigma^2 I)$ as each class. So, it is not possible to *exactly* evaluate g's prediction at any input x. Instead, we will implement Monte Carlo algorithms for both tasks.

Future ways of improvement:

- Defensive distillation
- Pruning Adversarially Robust Neural Networks
- Data Augmentation

Any Questions?