DE LA RECHERCHE À L'INDUSTRIE

Robustesse des réseaux de neurones face aux attaques adversariales

2022-2023

Aurélien Mayoue & Cédric Gouy-Pailler (cours)

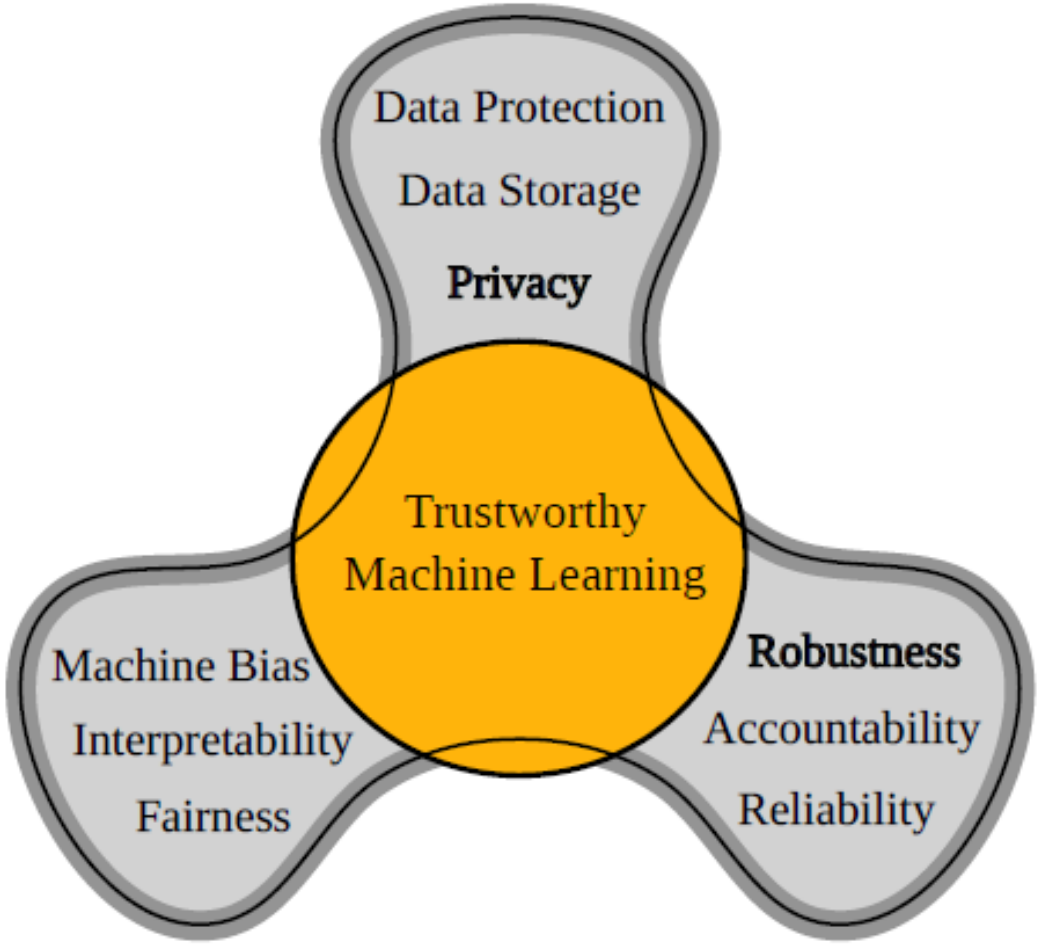
- Machine learning models recently gave **outstanding results** (*e.g.* vision, NLP)
- Industries and governments are starting to use them in **critical applications**

AI-driven Technologies

Decision-making Tools

Personal Devices

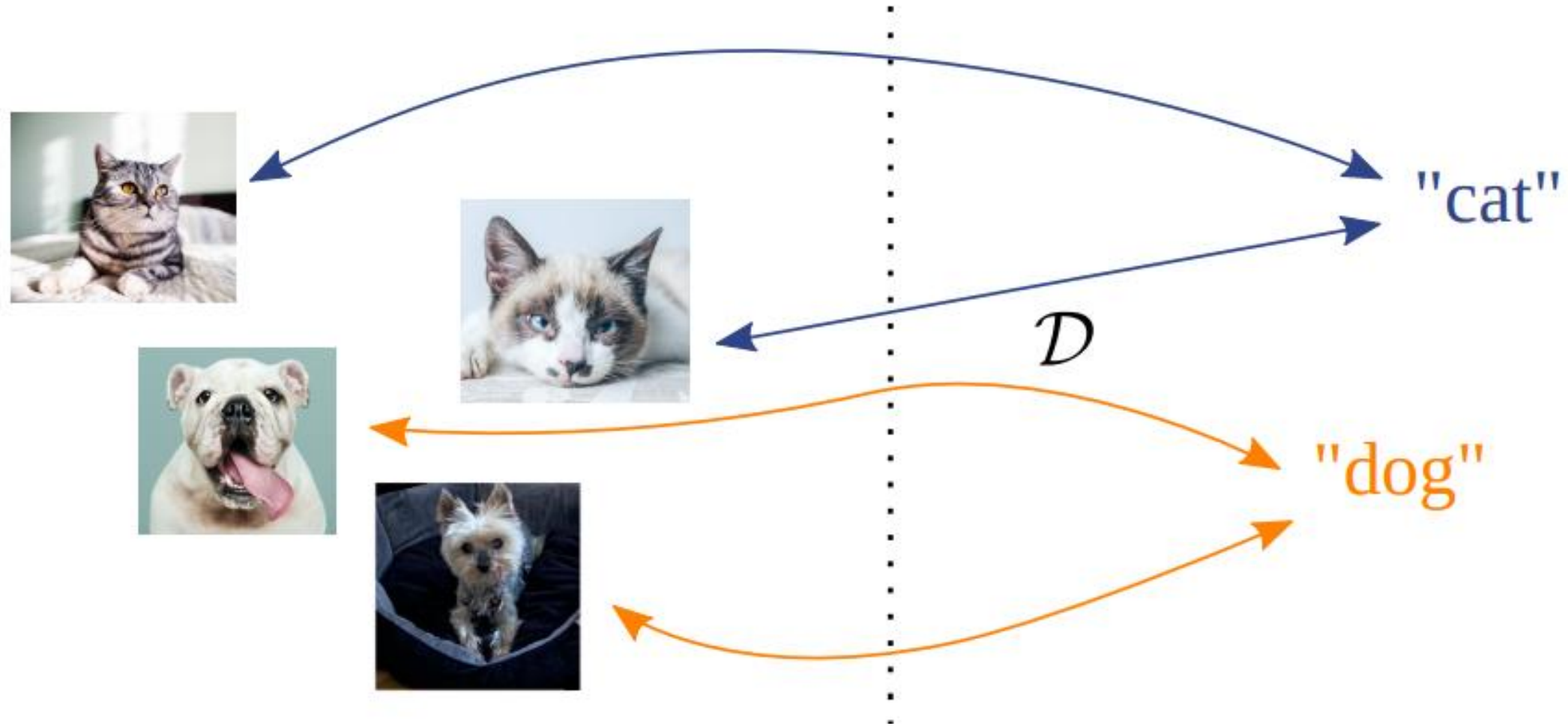
Introduction



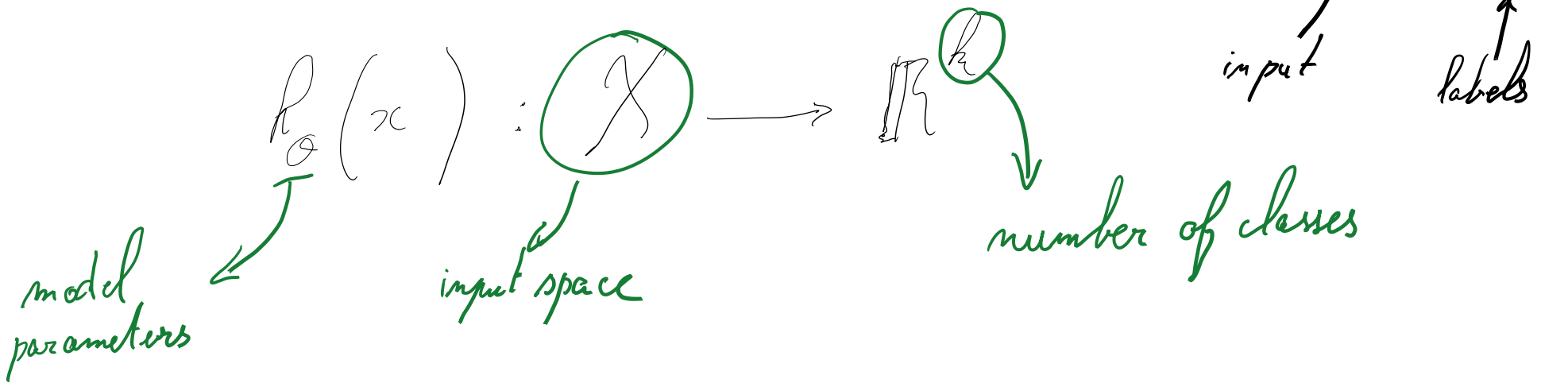
Adversarial attacks

Démonstration AI vs WILD

Learning



A ground truth distribution $\mathcal{D}$ explains the link between X and Y.



Loss function

$$\ell : \mathbb{R}^k \times \mathbb{Z}^+ \longrightarrow \mathbb{R}^+$$

$\ell(h(x), y)$ is called the loss function

Cross-entropy & softmax

softmax

$$\sigma : \mathbb{R}^k \longrightarrow \mathbb{R}^k$$

mapping from class logits to probabilities.

$$\sigma(z)_{[i]} = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)}$$

$$\log\left(\sigma\left(h_\theta(x)\right)_{[y]}\right) = \log\left(\frac{\exp\left(h_\theta(x)_{[y]}\right)}{\sum_{j=1}^{k} \exp\left(h_\theta(x)_{[j]}\right)}\right)$$

[maximize prob of true class label]

$\equiv$

[maximize log of probability]

Cross-entropy

[minimize negative log of probability]

$$-\log \sigma\left(h(x)_{[y]}\right) = \log\left(\sum_{j=1}^{k} \exp\left(h_\theta(x)_{[j]}\right)\right) - h_\theta(x)_{[y]}$$

Loss function

$$l\left(h_\theta(x), y\right) = \log\left(\sum_{j=1}^{k} \exp\left(h_\theta(x)_{[j]}\right)\right) - h_\theta(x)_{[y]}$$

Training a classifier

Optimize parameters θ to minimize the average loss over some training set $\{x_i \in \mathcal{X} ; y_i \in \mathbb{Z}^+\}_{i=1..m}$

$$\underset{\theta}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} \ell\left(h_\theta(x_i), y_i\right)$$

$\Longrightarrow$ Stochastic gradient descent on some minibatch $\mathcal{B} \subseteq \{1, \dots, m\}$

$$\theta^{[t+1]} \leftarrow \theta^{[t]} - \frac{\alpha}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \ell\left(h_\theta(x_i), y_i\right)$$

$$\nabla_\theta \, \ell\left(h_\theta(x_i), y_i\right)$$

Gradient

↓

Backpropagation

Automatic differentiation

Can we compute the gradient of the loss with respect to input x_i ?

Adjust the image to maximize the loss.

Untargeted Attack

$$\text{maximize}_{\delta \in \Delta} \; \ell\left(h_\theta(x+\delta), y\right)$$

→ Set of allowable perturbations

example $\|\|_\infty$: $\Delta = \{\delta : \|\delta\|_\infty \leq \varepsilon\}$

$$\|z\|_\infty = \max_i |z_i|$$

Targeted attacks

maximize $\delta \in \Delta$

$$\underbrace{\ell\left(h_\theta(x+\delta), y\right)}_{\text{max the loss of the correct class}} - \underbrace{\ell\left(h_\theta(x+\delta), y_{target}\right)}_{\text{minimize the loss of the target class}}$$

$\Longrightarrow$ maximize $\delta \in \Delta$

$$h_\theta(x+\delta)_{[y_{target}]} - h_\theta(x+\delta)_{[y]}$$

More formal notation

$$R(h_\theta) = \mathbb{E}_{x,y \sim \mathcal{D}} \left[\ell\left(h_\theta(x), y\right) \right]$$

Risk

$$\hat{R}(h_\theta, D) = \frac{1}{|D|} \sum_{(x,y) \in D} \ell\left(h_\theta(x), y\right)$$

Empirical Risk.

⚠ We are trying to maximize the loss!

↳ in the direction of the positive gradient.

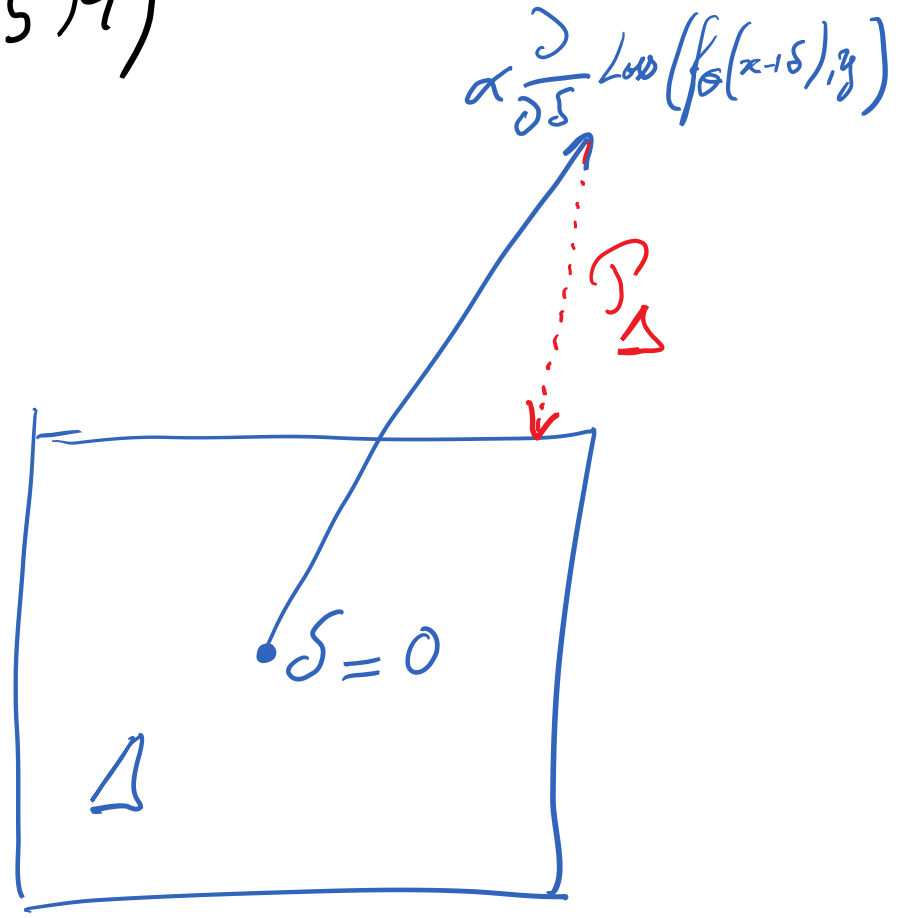
Projected Gradient Descent → move toward desired direction

→ ensure $\delta \in \Delta$

$$\delta := \text{Proj}\left(\delta + \alpha \frac{\partial}{\partial \delta} \underbrace{\text{Loss}\left(f_\theta(x+\delta), y\right)}_{\text{to be defined depending on kind of attack}}\right)$$

Descent → projection

The Fast Gradient Sign Method (FGSM)



Linear models (example from Kolter & Madry)

Linear decision

$$h_\theta(x) = w^T x + b$$

$$\begin{vmatrix} w \in \mathbb{R}^n \\ b \in \mathbb{R} \\ y \in \{-1; +1\} \end{vmatrix}$$

Cross. entropy

$$\ell\left(h_\theta(x), y\right) = \log\left(1 + \exp\left(-y \cdot h_\theta(x)\right)\right) = L\left(y \cdot h_\theta(x)\right)$$

Probability

$$p\left(y = +1 \mid x\right) = \frac{1}{1 + \exp\left(-h_\theta(x)\right)}$$

$$L(z) = \log\left(1 + \exp(-z)\right)$$

$$\underset{\|\delta\|\leq\varepsilon}{\text{maximize}}\ \ell\left(w^T(x+\delta), y\right) = \underset{}{\text{maximize}}\ L\left(y\cdot\left(w^T(x+\delta)+b\right)\right)$$

monotonic $\Rightarrow$ maximize = minimize inside

$$\max_{\|\delta\|\leq\varepsilon} L\left(y\cdot\left(w^T(x+\delta)+b\right)\right)$$

$$= L\left(\min_{\|\delta\|\leq\varepsilon} y\cdot\left(w^T(x+\delta)+b\right)\right)$$

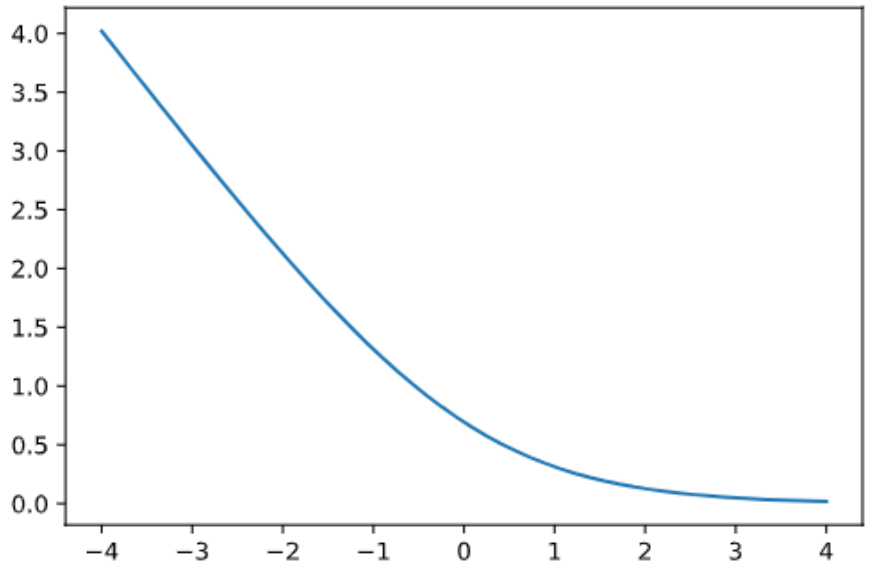
$$= L\left(y\cdot\left(w^Tx+b\right) + \min_{\|\delta\|\leq\varepsilon} y\cdot w^T\delta\right)$$

$$\varepsilon \begin{cases} w_i \geq 0 & \delta_i = -\varepsilon \\ w_i < 0 & \delta_i = \varepsilon \end{cases} \Rightarrow \delta^* = -y\varepsilon\cdot\text{sign}(w)$$

$y = -1$

$$\begin{cases} w_i \geq 0 & \delta_i = \varepsilon \\ w_i < 0 & \delta_i = -\varepsilon \end{cases}$$

```
x = np.linspace(-4,4)
plt.plot(x, np.log(1+np.exp(-x)))
```



$$y \cdot w^T \delta = y \cdot \sum_i -y\varepsilon \cdot \text{sign}(w_i)\, w_i = -y^2 \varepsilon \sum_i |w_i|$$

$$= -\varepsilon \|w\|_1$$

$$\max_{\|\delta\|_\infty \leq \varepsilon} L\left(y \cdot \left(w^T(x+\delta) + b\right)\right) = L\left(y \cdot \left(w^T x + b\right) - \varepsilon \|w\|_1\right)$$

Analytical solution.

Other strategies for neural networks

$$z_1 = x$$

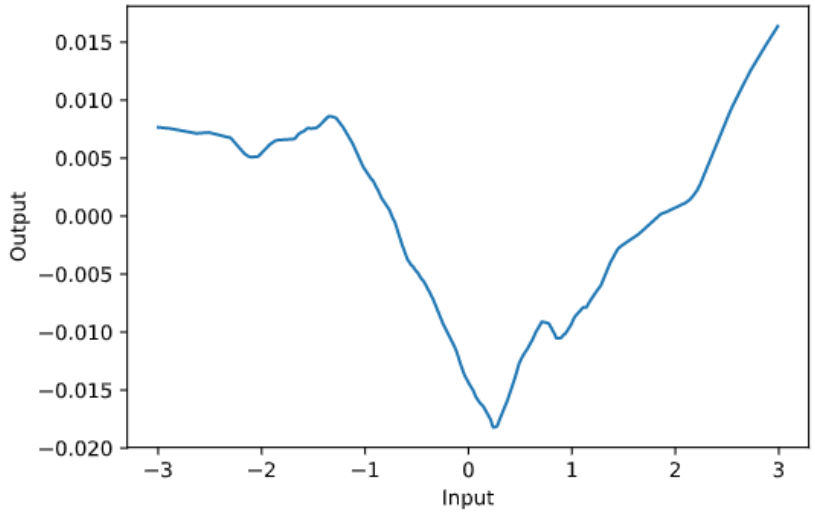
$$z_{i+1} = f_i(W_i z_i + b_i) \quad i \in [\![1; d]\!]$$

$$f_\theta(x) = z_{d+1}$$

z_i : activations at layer i

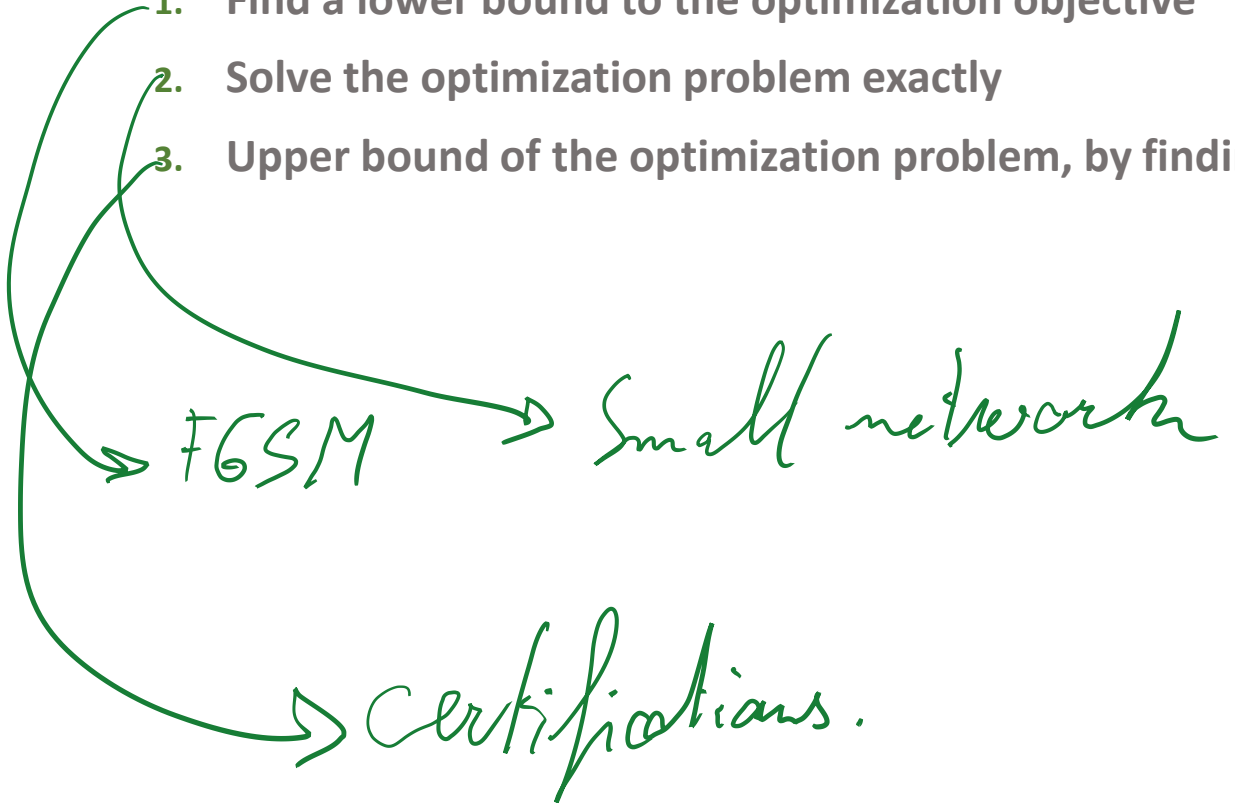
$f_i(z) = \max(0; z)$: ReLu operator.

```
import torch
import torch.nn as nn
import torch.optim as optim

torch.manual_seed(0)
model = nn.Sequential(nn.Linear(1,100), nn.ReLU(),
                      nn.Linear(100,100), nn.ReLU(),
                      nn.Linear(100,100), nn.ReLU(),
                      nn.Linear(100,1))
opt = optim.SGD(model.parameters(),lr=1e-2)
for _ in range(100):
    loss = nn.MSELoss()(model(torch.randn(100,1)), torch.randn(100,1))
    opt.zero_grad()
    loss.backward()
    opt.step()

plt.plot(np.arange(-3,3,0.01), model(torch.arange(-3,3,0.01)[:,None]).detach().numpy())
plt.xlabel("Input")
plt.ylabel("Output")
```



Strategies

1. **Find a lower bound to the optimization objective**
2. **Solve the optimization problem exactly**
3. **Upper bound of the optimization problem, by finding convex relaxation of the network structure**



Defending against adversarial attacks

Key insight from first part

Untargeted adversarial attack : SOLVING THE INNER OPTIMIZATION PROBLEM

$$\max_{\delta \in \Delta} \text{Loss}\left(f_\theta(x+\delta), y\right)$$

Key Insight : The same process that enabled us to learn the model parameters via gradient descent also allows us to create an adversarial example via gradient descent.

$$\frac{\partial}{\partial \delta} \text{Loss}\left(f_\theta(x+\delta), y\right)$$

Adversarial attack: small, imperceptible change of an image maliciously designed to fool the model.

clean example x | perturbation δ | adversarial example $x+\delta$

Label: "cat" Label: "dog"

Outer minimization

$$\min_{\theta} \mathbb{E}_{x,y}\left[\text{Loss}\left(f_\theta(x), y\right)\right] \Rightarrow \min_{\theta} \mathbb{E}\left[\max_{\delta \in \Delta} \text{Loss}\left(f_\theta(x+\delta), y\right)\right]$$

Defense strategies

Adversarial training : during the learning procedure, replace x with x_{adv}. STATE-OF-THE-ART EMPIRICAL DEFENSE FOR ℓ_∞ ATTACKS.

Randomized smoothing : point-wise certification of the classification. STATE-OF-THE-ART CERTIFIED DEFENSE FOR ℓ_2 ATTACKS

Provable defense : provably upper bound inner maximization

Adversarial training objective

$$\min_{\theta} \sum_{x,y \in S} \max_{\delta \in \Delta} \text{Loss}\left(f_\theta(x+\delta), y\right)$$

Gradient descent !? Difficult due to max term inside.

Danskin's theorem:

$$\frac{\partial}{\partial \theta} \max_{\delta \in \Delta} Loss\left(f_\theta(x+\delta), y\right) = \frac{\partial}{\partial \theta} Loss\left(f_\theta(x+\delta^*), y\right)$$

where $$\boxed{\delta^* = \underset{\delta \in \Delta}{\arg\max}\ Loss\left(f_\theta(x+\delta), y\right)}$$

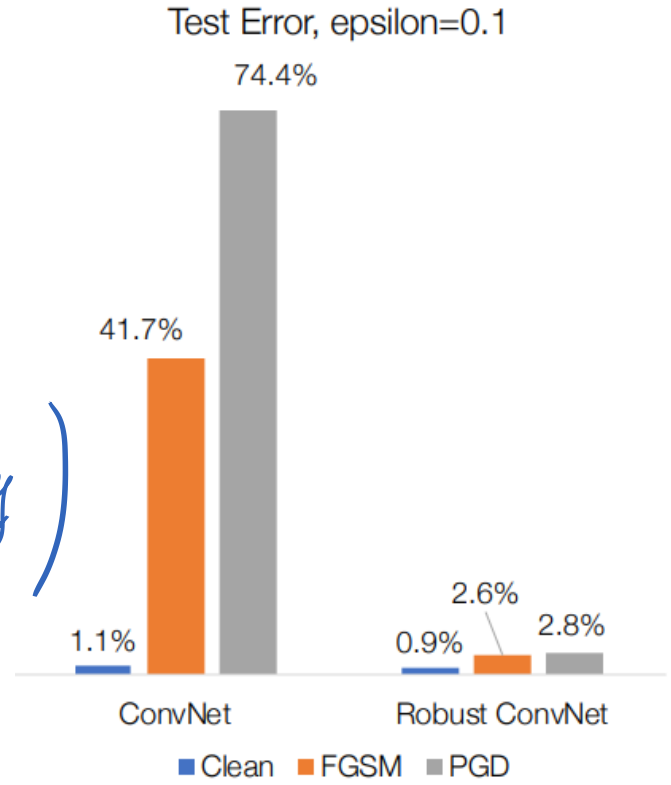
$\Rightarrow$ we can optimize through the max by just finding its maximizing value !!

Algorithm and results

Repeat:

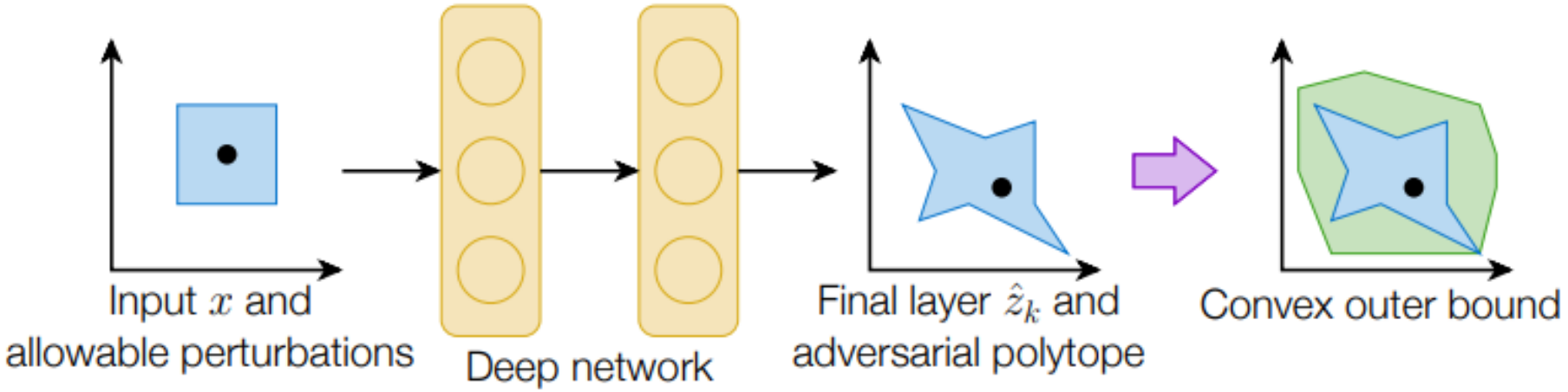
1. Select minibatch B
2. For each $(x, y) \in B$, compute adversarial example $\delta^*(x)$
3. Update parameters

$$\theta := \theta - \frac{\alpha}{|B|} \sum_{x,y \in B} \frac{\partial}{\partial \theta} Loss\left(f_\theta(x + \delta^*(x)), y\right)$$



$$\max_{\delta \in \Delta} \text{Loss}\left(f_\theta(x+\delta), y\right) \leq \max \text{Loss}\left(f_\theta^{rel}(x+\delta), y\right)$$

E. Wong & Z. Kolter (2018)



Let's consider a k-layer feedforward ReLU-based neural network, $f_\theta : \mathbb{R}^{n_1} \to \mathbb{R}^{n_k}$ s.t

$$\hat{z}_{i+1} = W_i z_i + b_i \quad \text{for } i = 1, \dots, k-1$$

$$z_i = \max(\hat{z}_i, 0) \quad \text{for } i = 2, \dots, k-1$$

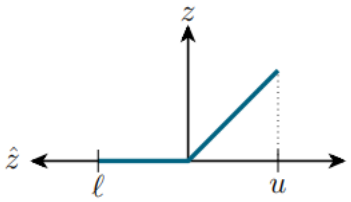
Convex outer bound : linear relaxation of the ReLU activations.

Given known lower and upper bounds l, u for the pre-ReLU activations, we can replace ReLU $z = \max\{0; \hat{z}\}$ with their upper convex envelopes.

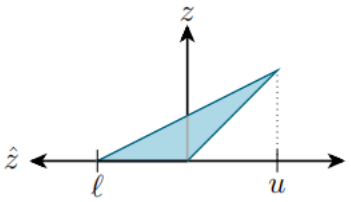
$$z \geq 0$$

$$z \geq \hat{z}$$

$$-u\hat{z} + (u-l)z \leq -ul$$



Bounded ReLU set



Convex relaxation

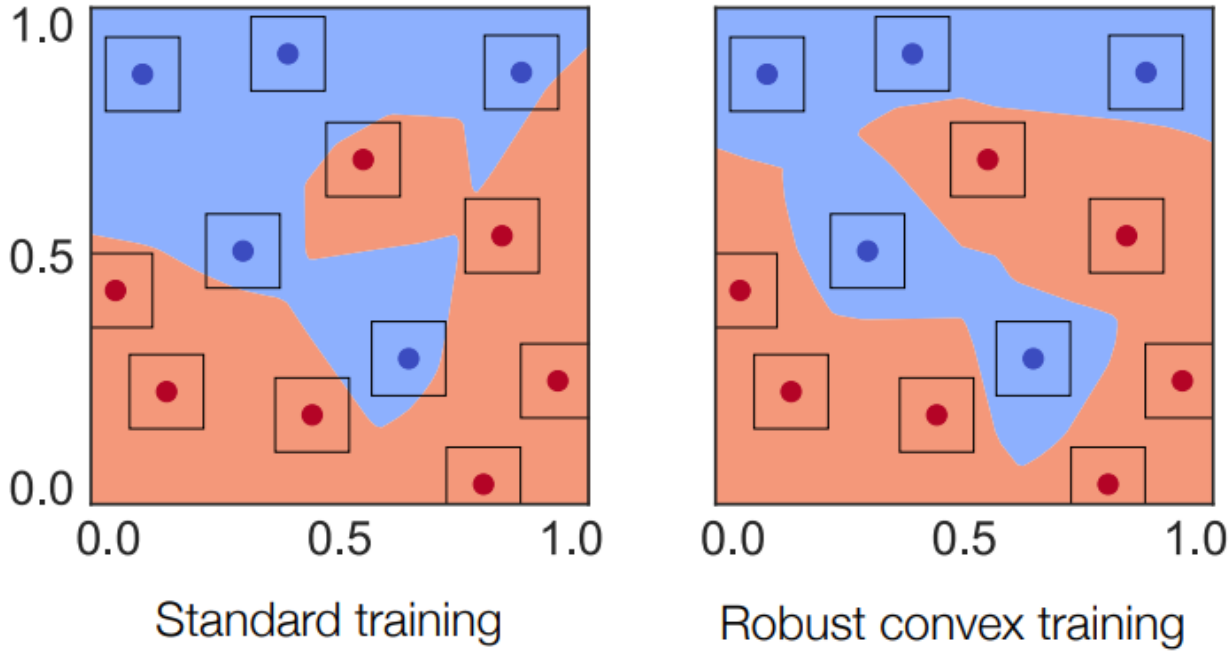
$\Rightarrow$ MAXIMIZATION problem is now a convex linear program !

minimize the computed bound on loss (autodiff)

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} l\left(J_{\varepsilon,\theta}(x_i), y_i\right) \geq \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} l\left(\max_{\delta \in \Delta} l\left(f_\theta(x_i+\delta), y_i\right)\right)$$

Illustration

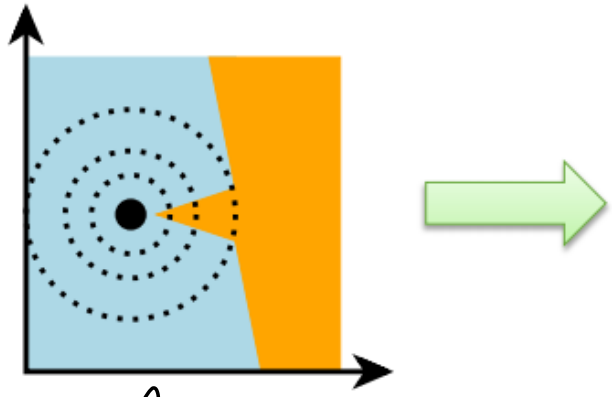
Simple 2D toy problem, 2-100-100-100-2 MLP network, trained with Adam (learning rate = 0.001, no hyperparameter tuning)



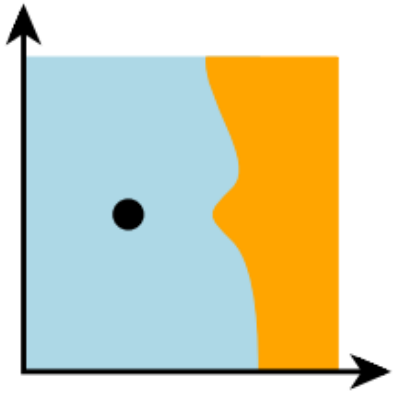
Example from Kolter.

Randomization as a defense

We can smooth the decision region by adding Gaussian noise to the input and picking the majority class of the classifier over this noise.



$f(x)$



$$g(x) = \arg\max_y \mathbb{P}_{\varepsilon \sim \mathcal{N}(0;\sigma^2 I)} \left[f(x+\varepsilon) = y \right]$$

Illustration

Classify a bunch of versions perturbed by random noise. Take the majority vote.

⚠ Need to train on noisy images.

Démonstration AI vs WILD

Randomized classifiers

$$\Gamma_{\Sigma} = \left\{ m : x \mapsto h(x+\zeta) \mid h : \mathcal{X} \to \mathbb{R} \text{ continuous and } \zeta \sim \mathcal{N}(0, \Sigma) \right\}$$

Gaussian noise injection: regularize the training phase

Here: test time to get robustness to data manipulation.

References

References

- Murphy, K.P. (2021). Probabilistic Machine Learning: An introduction. MIT Press. https://probml.github.io/pml-book/book1.html
- Kolter, Z. and Madry, A. (2018). Adversarial Robustness - Theory and Practice. https://adversarial-ml-tutorial.org
- Li, J. (2019). Course: Robustness in Machine Learning. https://jerryzli.github.io/robust-ml-fall19.html