An Adversarial Robustness Perspective on the Topology of Neural Networks

Authors :

Morgane Goibert (Criteo / Telecom Paris)

Thomas Ricatte (Amazon)

Elvis Dohmatob (Facebook AI Research)

Published :

Safety ML Workshop

NeurIPS 2022

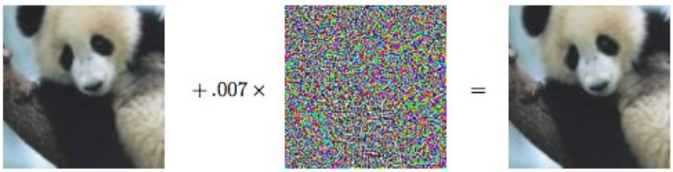
The paper is available on ArXiv

Content

① Intro to Adversarial Robustness

Phenomenon first described in [Intriguing Properties of Neural Networks, Szegedy et al. 2013]

Illustration from [Explaining and Harnessing Adversarial Examples, Goodfellow et al, 2014]

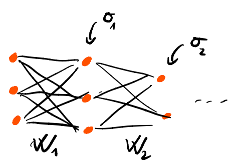


$\boldsymbol{x}$ "panda" 57.7% confidence | $\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ "nematode" 8.2% confidence | $\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ "gibbon" 99.3 % confidence

Classification problem setting

* Input $x \in \mathbb{R}^d$
  Label / Class $y \in [\![1, K]\!]$ } P is the joint distribution on (x, y)

* Dataset (x_i, y_i)



* Neural Network :

↳ Feature map $g : x \in \mathbb{R}^d \mapsto g(x) \in \mathbb{R}^K$

$$g(x) = \sigma_L (W_L \, \sigma_{L-1} (W_{L-1} \, \sigma_{L-2} \dots (\sigma_1 (W_1 \, x))))$$

where W_ℓ is the weight matrix for layer ℓ
σ_ℓ is the activation function for layer ℓ

Notation : $g(x)_\ell$ = output / activation of layer ℓ

↳ NN : $h(x) = \underset{k=1,\dots,K}{\text{argmax}} \; g(x)$

Adversarial example

* an adv example x^{adv} is a perturbed version of a clean ex x :

$$x^{adv} = x + \delta \quad , \quad \delta = \text{perturbation}$$

* Small perturbation : $\|\delta\|_{+\infty, 2} \leq \varepsilon$

* Goal = misclassification $\rightarrow h(x^{adv}) \neq h(x)$

Attacks

The underlying optim pb to find the optimal perturbation δ is hard

=> Several algos using different strategies

$$\left.\begin{array}{l} * \text{ FGSM , PGD} \\ * \text{ CW} \end{array}\right\} \text{white - box}$$

* Boundary $\rightarrow$ black - box

(Known) Characteristics of attacks

1. They work really well
2. They transfer between architectures
3. They can be $\begin{cases} \text{on - manifold} \rightarrow \text{leverage useful \& non - robust features in the data} \\ \text{off - manifold} \rightarrow \text{——— non - useful ———} \end{cases}$
4. They are less efficient against highly regularized NNs / smooth decision boundary

② Our Hypothesis

a) Some observations

* Pruning

→ NNs are over-parametrized

→ pruning do not hinder accuracy [The lottery ticket hypothesis: finding sparse, trainable NNs, Franckle & Carbin, 2018]

⇒ Only a subset of parameters are really important for inference (on clean inputs)

⇒ Clean inputs use highway edges

* Adversarial examples

→ They can use under-optimized parameters/edges (off-manifold) ie more diffuse paths

b) The hypothesis

Clean vs adv examples induce differences in the topological structure of their respective induced graph because adv (and not clean inputs) leverage under-optimized edges

③ Method : Extracting Topological Features

a) Induced Graph

* Idea : represent the way an input x traverses a NN g

* Induced graph : $G(x, g) = (V, E)$

$$V = \{1, 2, \dots, n_0 + \dots + n_L\} \quad \text{where} \quad n_\ell = \text{nb neuron in layer } \ell$$

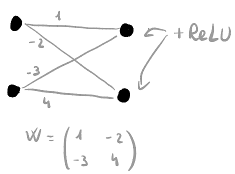
$$E = \{(v^\ell, v^{\ell+1}, w^\ell_{v,v'})\} \quad \text{where} \quad w^\ell_{v,v'} \text{ is the weight}$$

→ linear layer : $w^\ell_{v,v'} = | [g_\ell(x)]_v \times (W_\ell)_{v,v'} |$

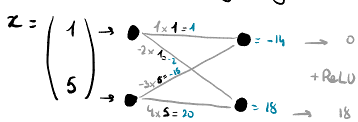
→ generalize to all architecture types (eg ResNet) and layers (eg convolutional)

* Illustration :

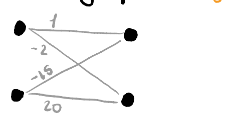
A trained NN



Computing the passage of x



Induced graph $G(x, g)$:



b) Selecting Under-optimized Edges

* Why ? → according to the hypothesis, they are sufficient

→ complexity : NNs have way to many neurons / edges

* How ? → definition from [Deconstructing Lottery Tickets : Zeros, signs and the supermask, Zhou et al., 2019]

→ An edge (u,v) is under-optimized if its weight has not changed much during training → $|(W_\ell)_{u,v} - (W_\ell^{init})_{u,v}| < \text{quantile}(q)$

→ So we keep only a fraction q of edges

c) Persistent Diagrams

* Graphs considered : thresholded induced graph $G^q(x,g)$

* Weights : $\tilde{w}_{u,v} = -|w_{u,v}|$ so that highest-weighted edges appear first

* Filtration : natural one based on weight value

  → A time $t \in \mathbb{R}_-$, the considered graph is $G_t^q(x,g)$

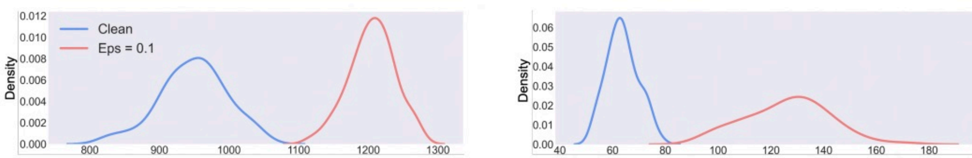
  where $G_t^q(x,g) = (V_t, E_t)$ with $E_t = \{\underbrace{(u,v,\tilde{w}_{u,v})}_{\in E} \mid \tilde{w}_{u,v} \leq t\}$

④ Experiments : Differentiating Clean vs Adv inputs

a) Quantitative Results

* Summary : I have a collection of persistence diagrams for
  1) adv inputs
  2) clean inputs

* Simple idea : count the nb of points in the PDs → all points
  → infinitely - lived ones



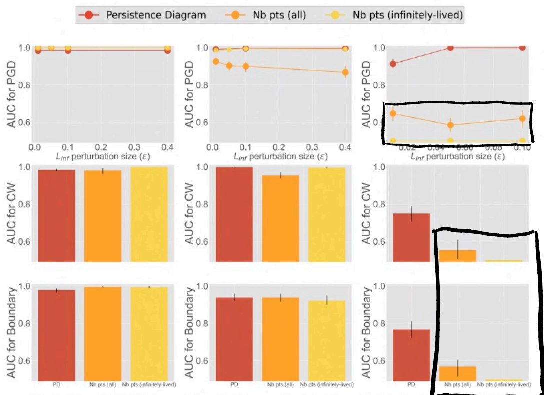
(a) Distribution of all PD points.

(b) Distribution of infinitely-lived PD points.

Figure 2: PD points computed on MNIST / LeNet

b) Detecting adv examples

* Nb of points is a limited strategy



(a) LeNet / MNIST (b) LeNet / Fashion MNIST (c) ResNet / CIFAR10

Figure 19: Unsupervised detection results using number of points only.

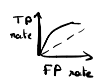
In this more complicated setting, the results are bad

* Using all the information provided by PDs => detection

→ One-class SVM ie * unsupervised (the algo only sees clean PDs during training

* relevant kernel on PDs → Sliced-Wasserstein kernel [Sliced-Wasserstein Kernel for Persistence Diagram - Carrière, Cuturi & Oudot, 2017]

→ Performance metric : AUC (Area Under the ROC curve) →



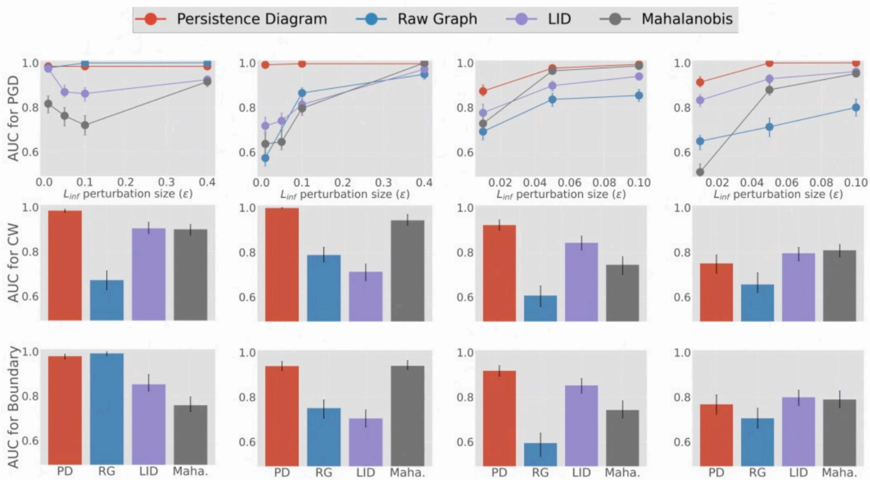
→ Comparison with SOTA : 1) LID

2) Mahalanobis

3) Baseline from the induced graph = RG

→ features are a vector, elements are the weights of $G^q(x, g)$

* Main results

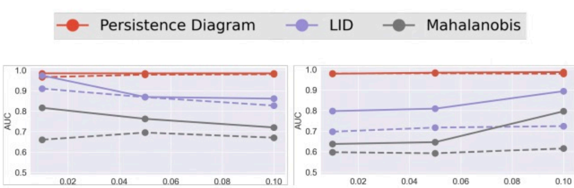


(a) LeNeT/MNIST (b) LeNet/Fashion MNIST (c) ResNet/SVHN (d) ResNet/CIFAR10

Figure 3: Showing detection AUC for different detection methods (legend) against different kinds of adversarial attacks (rows) and model architectures and datasets (columns). We see that our proposed method based on PD outperforms the SOTA methods, except for one tie.

c) Additional Results

* Still performant on AT networks



(a) LeNet / MNIST (b) LeNet / Fashion MNIST

Figure 21: Unsupervised detection results (on PGD) of AT vs standard NNs

* Selecting Under-optimized edges is the good strategy

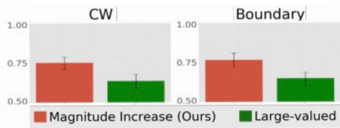


Figure 22: Impact of edge-selection methods on AUC (ResNet / CIFAR10).

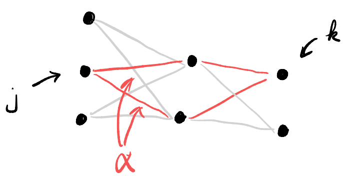
* Pruning can improve robustness

Proposition : $\forall$ class $k \in [\![1, K]\!]$, $\forall$ input feature index $j \in [\![1, n_0]\!]$

$$\frac{\partial [g(x)]_k}{\partial x_j} = \sum_{\alpha \in \mathcal{A}} \mathcal{W}(\alpha)$$

where $\mathcal{A}$ is the set of active paths from j to k

$\mathcal{W}(\alpha) = \prod_{l=1}^{L} (W_l)_{v^{l-1}(\alpha),\, v^l(\alpha)}$ is the product of weights in path α



$\Rightarrow$ $\frac{\partial [g(x)]_k}{\partial x_j}$ is a proxy for the vulnerability of class k wrt input feature j.

The norm of the Jacobian matrix $J(x, g) = \left(\frac{\partial [g(x)]_k}{\partial x_j} \right)$ is a general proxy for the robustness to perturbations on x $\rightarrow$ Reducing the cardinality of $\mathcal{A}$ should help

The End !

Thank you &

Question time !