Increasing Confidence in Adversarial Robustness Evaluations

Javad Hezareh & Mahdi Saieedi

Security and Privacy in Machine Learning

Sharif University of Technology

Spring 2023

Overview

Increasing Confidence in Adversarial Robustness Evaluations

Roland S. Zimmermann* University of Tübingen
Wieland Brendel University of Tübingen
Florian Tramèr Google
Nicholas Carlini Google

Abstract

Hundreds of defenses have been proposed to make deep neural networks robust against minimal (adversarial) input perturbations. However, only a handful of these defenses held up their claims because correctly evaluating robustness is extremely challenging: Weak attacks often fail to find adversarial examples even if they unknowingly exist, thereby making a vulnerable network look robust.

In this paper, we propose a test to identify weak attacks, and thus weak defense evaluations. Our test slightly modifies a neural network to guarantee the existence of an adversarial example for every sample. Consequentially, any correct attack must succeed in breaking this modified network.

For eleven out of thirteen previously-published defenses, the original evaluation of the defense fails our test, while stronger attacks that break these defenses pass it. We hope that attack *unit tests* — such as ours — will be a major component in future robustness evaluations and increase confidence in an empirical field that is currently riddled with skepticism. Online version & code: zimmerrol.github.io/active-tests/

Figure: Oral at CVPR 2022 Workshop (Art of Robustness), Project website.

Overview

- Can we trust a proposed defense?
- Is defense evaluation valid?

Overview

- Can we trust a proposed defense?
- Is defense evaluation valid?
- Main scheme for defense evaluation:
  1. Propose an attack
  2. Evaluate defense with this attack
  3. If no adversarial example is found, defense works

Outline

Outline

Theorem Analogy

Lemma 2.4 (Certificate for Empirical Mean). *Let S be an (ϵ, δ)-stable set with respect to a distribution X, for some $\delta \geq \epsilon > 0$. Let T be an ϵ-corrupted version of S. Let μ_T and Σ_T be the empirical mean and covariance of T. If the largest eigenvalue of Σ_T is at most $1 + \lambda$, for some $\lambda \geq 0$, then $\|\mu_T - \mu_X\|_2 \leq O(\delta + \sqrt{\epsilon\lambda})$.*

Proof of Lemma 2.4. Let $S' = S \cap T$ and $T' = T \setminus S'$. By replacing S' with a subset if necessary, we may assume that $|S'| = (1-\epsilon)|S|$ and $|T'| = \epsilon|S|$. Let $\mu_{S'}, \mu_{T'}, \Sigma_{S'}, \Sigma_{T'}$ represent the empirical means and covariance matrices of S' and T'. A simple calculation gives that

$$\Sigma_T = (1-\epsilon)\Sigma_{S'} + \epsilon\Sigma_{T'} + \epsilon(1-\epsilon)(\mu_{S'} - \mu_{T'})(\mu_{S'} - \mu_{T'})^T .$$

Let v be the unit vector in the direction of $\mu_{S'} - \mu_{T'}$. We have that

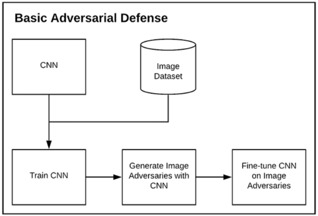
$$\begin{aligned} 1 + \lambda \geq v^T \Sigma_T v &= (1-\epsilon) v^T \Sigma_{S'} v + \epsilon v^T \Sigma_{T'} v + \epsilon(1-\epsilon) v^T (\mu_{S'} - \mu_{T'})(\mu_{S'} - \mu_{T'})^T v \\ &\geq (1-\epsilon)(1 - \delta^2/\epsilon) + \epsilon(1-\epsilon)\|\mu_{S'} - \mu_{T'}\|_2^2 \\ &\geq 1 - O(\delta^2/\epsilon) + (\epsilon/2)\|\mu_{S'} - \mu_{T'}\|_2^2 , \end{aligned}$$

where we used the variational characterization of eigenvalues, the fact that $\Sigma_{T'}$ is positive semidefinite, and the second stability condition for S'. By rearranging, we obtain that $\|\mu_{S'} - \mu_{T'}\|_2 = O(\delta/\epsilon + \sqrt{\lambda/\epsilon})$. Therefore, we can write

$$\begin{aligned} \|\mu_T - \mu_X\|_2 = \|(1-\epsilon)\mu_{S'} + \epsilon\mu_{T'} - \mu_X\|_2 &= \|\mu_{S'} - \mu_X + \epsilon(\mu_{T'} - \mu_{S'})\|_2 \\ &\leq \|\mu_{S'} - \mu_X\|_2 + \epsilon\|\mu_{S'} - \mu_{T'}\|_2 = O(\delta) + \epsilon \cdot O(\delta/\epsilon + \sqrt{\lambda/\epsilon}) \\ &= O(\delta + \sqrt{\lambda\epsilon}) , \end{aligned}$$

where we used the first stability condition for S' and our bound on $\|\mu_{S'} - \mu_{T'}\|_2$. □

$\longleftrightarrow$



Theorem

Proof.

we have proved that $P \neq NP$. $\cdots$ $\square$

- How do you refute the proof's claim?

Theorem

Proof.

we have proved that $P \neq NP$. $\cdots$ $\square$

- How do you refute the proof's claim?
  - Find an algorithm to solve 3-SAT in polynomial time.
  - Studying proofs line-by-line, till find some major flaw.

Defense evaluation

Defense X.

We have demonstrated that defense X improves model robustness. One can validate this claim by following the below evaluation procedure.

. . .

- How do you refute the authors' claim?

Defense evaluation

Defense X.

We have demonstrated that defense X improves model robustness. One can validate this claim by following the below evaluation procedure.

. . . □

- How do you refute the authors' claim?
  - Find an adversarial attack to decrease model performance.
  - Probe defense evaluation, till find some major flaw.

Method

- Test attack strength

Method

- Test attack strength
- Design a new task that is solvable by any sufficiently strong attack
  - Injects adversarial examples into a defense
  - Check if the attack can find them

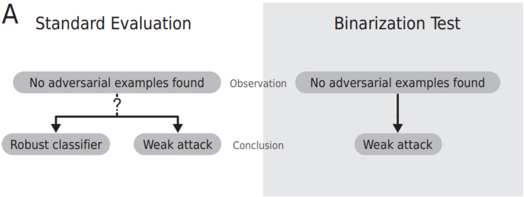


Figure: Proposed method to evaluate the attack used in defense evaluation.

Method

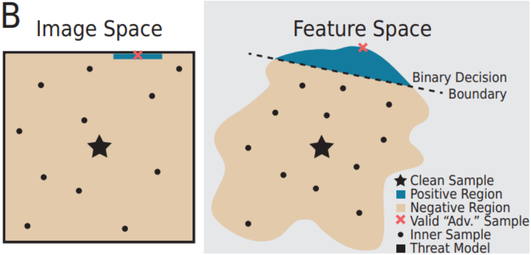


Figure: Inject adversarial examples to check whether the attack is powerful enough.

- A rejected attack doesn't necessarily mean the defense is not effective.

Outline

Adversarial Examples

- Imperceptible perturbations that change the decision of a deep neural network in arbitrary directions

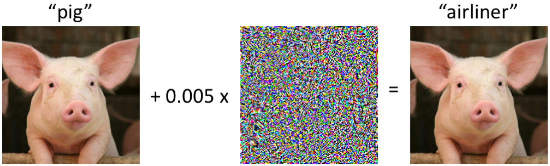


Figure: Adversarial examples.

Defenses

- Add input pre-processing steps
- Introduce architectural changes
- Methods for detecting adversarial examples

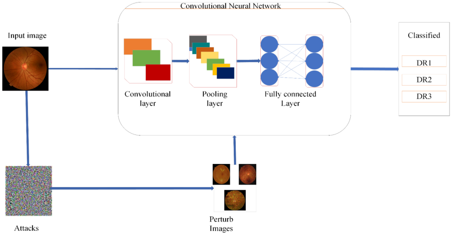


Figure: Adversarial training.

Outline

Defense Evaluation

- Defense evaluation scheme:
  - Perform an attack on defense
  - No adversarial example found within distance $d(\boldsymbol{x}_c, \boldsymbol{x}_{adv}) \leq \epsilon$

Defense Evaluation

- Defense evaluation scheme:
  - Perform an attack on defense
  - No adversarial example found within distance $d(\boldsymbol{x}_c, \boldsymbol{x}_{adv}) \leq \epsilon$
- Attack strength depends on:
  - Attack itself
  - The defense it is meant to evaluate

Outline

Classification Model

- f: our classifier
- f^*: model feature extractor
- $f = f^* +$ linear classification head

New Task Algorithm

- New binary classification task
- Ensure having adversarial examples

Algorithm Build New Task

Require: feature extractor f^* of original classifier, test sample $\boldsymbol{x}_c$, distanse ϵ, number of inner/boundary samples N_i and N_b.

function $\textsc{CreateBinaryClassifier}(f^*, \boldsymbol{x}_c, \epsilon, N_i, N_b)$

$\mathcal{X}_i := \{\boldsymbol{x}_c\} \cup \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) \leq \alpha\epsilon \text{ and } \hat{\boldsymbol{x}} \neq \boldsymbol{x}_c\}_{1:N_i}$

$\mathcal{X}_b := \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) = \epsilon\}_{1:N_b}$

$F_i := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_i\}$

$F_b := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_b\}$

$\mathcal{D} = \{(\hat{\boldsymbol{x}}, 0) \mid \hat{\boldsymbol{x}} \in F_i\} \cup \{(\hat{\boldsymbol{x}}, 1) \mid \hat{\boldsymbol{x}} \in F_b\}$

$g = \textsc{TrainLinear}(D)$

$h = g \circ f^*$

return h

Binarization Test

Algorithm Binarization Test for Classifiers with Linear Classification Readouts

Require: feature extractor f^* of original classifier, test samples $\mathcal{X}_{test}$, distanse ϵ, number of inner/boundary samples N_i and N_b.

function BinarizationTest(f^*, $\mathcal{X}_{test}$, ϵ, N_i, N_b)
    attack_successful = []
    random_attack_successful = []
    for all $\boldsymbol{x}_c \in \mathcal{X}_{test}$ **do**
        h = CreatBinaryClassifier(f^*, $\boldsymbol{x}_c$, ϵ, N_i, N_b)
        attack_successful.append(Attack($h, \boldsymbol{x}_c$))
        random_attack_successful.append(RandomAttack($h, \boldsymbol{x}_c$))
    ASR = Mean(attack_successful)
    RASR = Mean(random_attack_successful)
    return ASR, RASR

Evaluate New Task

- The efficacy of the used evaluation method:
  - Use original attack to attack h
  - ASR returns *test score*

Evaluate New Task

- The efficacy of the used evaluation method:
  - Use original attack to attack h
  - ASR returns *test score*
- Test difficulty:
  - Use randomized attack
  - #samples = #queris
  - RASR returns this metric

Outline

Models Leveraging Detectors

- Beside classifier f, we have a detector d
- d detects adversarial examples
- A successful attack must fool both the classifier and the detector

Models Leveraging Detectors

- Beside classifier f, we have a detector d
- d detects adversarial examples
- A successful attack must fool both the classifier and the detector
- Two test for this type:
  - Regular Test
  - Inverted Test
- A reliable evaluation must pass both tests

Regular Test

Algorithm Build New Task for Classifiers with Detectors

Require: feature extractor f^* of original classifier, adversarial detector d, test sample $\boldsymbol{x}_c$, distanse ϵ, number of inner/boundary/reference samples $N_i/N_b/N_r$.

function $\textsc{CreateBinaryClassifier}(f^*, d, \boldsymbol{x}_c, \epsilon, N_i, N_b, N_r)$

$\mathcal{X}_i := \{\boldsymbol{x}_c\} \cup \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) \leq \alpha\epsilon \text{ and } \hat{\boldsymbol{x}} \neq \boldsymbol{x}_c\}_{1:N_i}$

$\mathcal{X}_b := \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) = \epsilon, d(\hat{\boldsymbol{x}}) = 1\}_{1:N_b}$

$\mathcal{X}_r := \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) = \eta\epsilon, d(\hat{\boldsymbol{x}}) = 1\}_{1:N_r}$

$F_i := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_i\}$

$F_b := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_b\}$

$F_r := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_r\}$

$\mathcal{D} = \{(\hat{\boldsymbol{x}}, 0) \mid \hat{\boldsymbol{x}} \in F_i\} \cup \{(\hat{\boldsymbol{x}}, 1) \mid \hat{\boldsymbol{x}} \in F_b\} \cup \{(\hat{\boldsymbol{x}}, 1) \mid \hat{\boldsymbol{x}} \in F_r\}$

$b = \textsc{TrainLinear}(D)$

return b, $\mathcal{X}_r$

Binarization Test For Detectors

Algorithm Binarization Test for Classifiers with Linear Classification Readouts and a Detector

Require: feature extractor f^* of original classifier, adversarial detector d, test samples $\mathcal{X}_{test}$, distanse ϵ, number of inner/boundary/reference samples $N_i/N_b/N_r$.

function BinarizationTest(f^*, d, $\mathcal{X}_{test}$, ϵ, N_i, N_b, N_r, η)
    attack_successful = []
    random_attack_successful = []
    for all $\boldsymbol{x}_c \in \mathcal{X}_{test}$ **do**
        $b, \mathcal{X}_r$ = CreatBinaryClassifier(f^*, d, $\boldsymbol{x}_c$, ϵ, N_i, N_b, N_r)
        attack_successful.append(Attack($b, d, \boldsymbol{x}_c, \mathcal{X}_r$))
        random_attack_successful.append(RandomAttack($b, d, \boldsymbol{x}_c, \mathcal{X}_r$))
    ASR = Mean(attack_successful)
    RASR = Mean(random_attack_successful)
    return ASR, RASR

Inverted Test

- Why do we need the inverted test?

Algorithm Inverted Test

1: **function** InvertedBinarizationTest(f^*, d, $\mathcal{X}_{test}$, ϵ, N_i, N_b, N_r, ϵ, η)
2: **return** BinarizationTest(f^*, $\neg d$, $\mathcal{X}_{test}$, ϵ, N_i, N_b, N_r, ϵ, η)

Outline

Evaluations

- Defenses without Detectors
- Defenses with Detectors

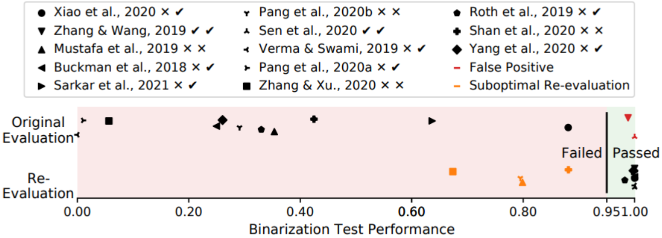


Figure: Binarization Test result for 13 Defenses.

Evaluation

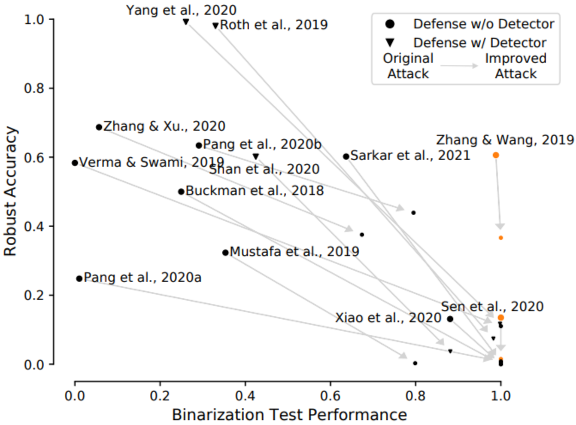


Figure: Robust accuracy as a function of the test performance.

Hardness of Test

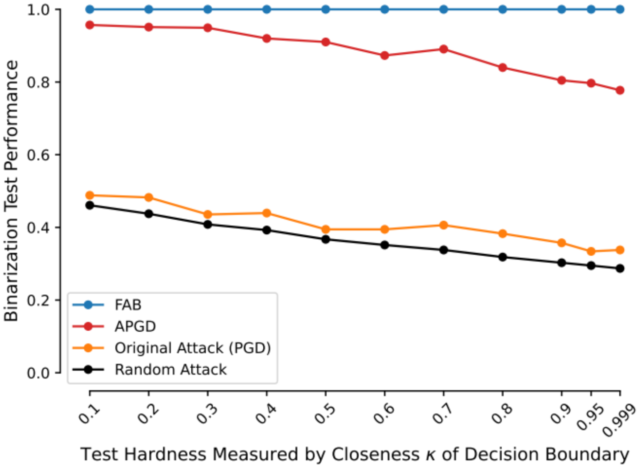


Figure: Hyperparameters influence the test's hardness

Thanks

Any questions?