

CS5562: Trustworthy Machine Learning

Lecture 2: Robustness → Robust Inference in the Adversarial Setting

Reza Shokri^a

Aug 2023

^aAcknowledgment. The wonderful teaching assistants: Hongyan Chang, Martin Strobel, Jiashu Tao, Yao Tong, Jiayuan Ye

Attempts to Build Robust Models

Adversarial Training

Certified Robustness

Attempts to Build Robust Models

Attempts to build robust models

- Both FGSM and PGD attacks rely on the ability to compute **gradient** of a model with respect to its input.
- What if we prevent the adversary from computing gradients accurately?
- There are many defense techniques based on this Idea.

Idea: Make the gradient hard to compute (e.g., by making the loss function non-differentiable), or hard to compute correctly.

- For example, let $f(\cdot) = f^{1\dots j}(\cdot)$ be a neural network, and $f^1(\cdot)$ be a non-differentiable layer.
- Thermometer encoding: Given an image x , for each pixel color z , the l -level thermometer encoding $\pi(z)$ is a l -dimensional vector where $\pi(z)[k] = 1$ if $z > k/l$, and 0 otherwise (e.g., for a 10-level thermometer encoding, $\pi(0.66) = 1111110000$).
- Result: On CIFAR-10, this method increases adversarial accuracy to 50% for l_∞ perturbation with $\epsilon = 0.031$.

Is this robust against all attacks?

Source: [Athalye et al., 2018]

Idea: Make the gradient hard to compute (e.g., by making the loss function non-differentiable), or hard to compute correctly.

- For example, let $f(\cdot) = f^{1\dots j}(\cdot)$ be a neural network, and $f^1(\cdot)$ be a non-differentiable layer.
- Thermometer encoding: Given an image x , for each pixel color z , the l -level thermometer encoding $\pi(z)$ is a l -dimensional vector where $\pi(z)[k] = 1$ if $z > k/l$, and 0 otherwise (e.g., for a 10-level thermometer encoding, $\pi(0.66) = 1111110000$).
- Result: On CIFAR-10, this method increases adversarial accuracy to 50% for l_∞ perturbation with $\epsilon = 0.031$.

Is this robust against all attacks?

Source: [Athalye et al., 2018]

Idea: Make the gradient hard to compute (e.g., by making the loss function non-differentiable), or hard to compute correctly.

- For example, let $f(\cdot) = f^{1\dots j}(\cdot)$ be a neural network, and $f^1(\cdot)$ be a non-differentiable layer.
- Thermometer encoding: Given an image x , for each pixel color z , the l -level thermometer encoding $\pi(z)$ is a l -dimensional vector where $\pi(z)[k] = 1$ if $z > k/l$, and 0 otherwise (e.g., for a 10-level thermometer encoding, $\pi(0.66) = 1111110000$).
- Result: On CIFAR-10, this method increases adversarial accuracy to 50% for l_∞ perturbation with $\epsilon = 0.031$.

Is this robust against all attacks?

Source: [Athalye et al., 2018]

Adaptive attacks: Design attacks that are tailored to the proposed defense.

- Find a differentiable approximation $g(x)$ such that $g(x) \approx f^1(x)$.
- To approximate $\nabla_x f(x)$: perform the forward pass through $f(\cdot)$, but on the backward pass replace $f^1(x)$ with $g(x)$.
- Example: We can define $g(z)[k] = \min(\max(z - k/l, 0), 1)$. In this case, $f^1(z)[k] = \text{floor}(g(z)[k])$.

Source: [Athalye et al., 2018]

Adaptive attacks: Design attacks that are tailored to the proposed defense.

- Find a differentiable approximation $g(x)$ such that $g(x) \approx f^1(x)$.
- To approximate $\nabla_x f(x)$: perform the forward pass through $f(\cdot)$, but on the backward pass replace $f^1(x)$ with $g(x)$.
- Example: We can define $g(z)[k] = \min(\max(z - k/l, 0), 1)$. In this case, $f^1(z)[k] = \text{floor}(g(z)[k])$.

Source: [Athalye et al., 2018]

Adaptive attacks: Design attacks that are tailored to the proposed defense.

- Find a differentiable approximation $g(x)$ such that $g(x) \approx f^1(x)$.
- To approximate $\nabla_x f(x)$: perform the forward pass through $f(\cdot)$, but on the backward pass replace $f^1(x)$ with $g(x)$.
- Example: We can define $g(z)[k] = \min(\max(z - k/l, 0), 1)$. In this case, $f^1(z)[k] = \text{floor}(g(z)[k])$.

Source: [Athalye et al., 2018]

Idea: Randomize the model or the input, for every inference. This leads to randomized gradient of loss with respect to the input.

Example: Stochastic activation pruning ([Dhillon et al., 2018]): SAP randomly drops some neurons of each layer f^i to 0 with probability proportional to their absolute value.

Idea: Randomize the model or the input, for every inference. This leads to randomized gradient of loss with respect to the input.

Example: Stochastic activation pruning ([Dhillon et al., 2018]): SAP randomly drops some neurons of each layer f^i to 0 with probability proportional to their absolute value.

Adaptive attacks: Expectation Over Transformation (EOT): The adversary can estimate the gradients by performing multiple inferences and computing its expectation over the introduced randomness.

- For PGD attack, at each iteration of update, instead of taking a step in the direction of $\nabla_x f(x)$, the adversary can move in the direction of $\sum_{i=1}^k \nabla_x f(x)$ where each invocation is randomized with SAP.
- Results: The attacker reduces SAP model accuracy on adversarial examples to 9% at $\epsilon = 0.015$, and 0% at $\epsilon = 0.031$.

Source: [Athalye et al., 2018]

Adaptive attacks: Expectation Over Transformation (EOT): The adversary can estimate the gradients by performing multiple inferences and computing its expectation over the introduced randomness.

- For PGD attack, at each iteration of update, instead of taking a step in the direction of $\nabla_x f(x)$, the adversary can move in the direction of $\sum_{i=1}^k \nabla_x f(x)$ where each invocation is randomized with SAP.
- Results: The attacker reduces SAP model accuracy on adversarial examples to 9% at $\epsilon = 0.015$, and 0% at $\epsilon = 0.031$.

Source: [Athalye et al., 2018]

Idea: Compose the model by stacking multiple deep neural networks.

- PixelDefend ([Song et al., 2018]): Use a generative model to project a potential adversarial example back onto the normal data manifold before feeding it into a classifier.
- Intuition: Adversarial examples mainly lie in the low-probability region of the data distribution. PixelDefend can purify adversarially perturbed images by finding the highest probability example within an ϵ -ball of the input image.
- Results: With a maximum l_∞ perturbation of $\epsilon = 0.031$, PixelDefend claims 46% accuracy on CIFAR10

Is this robust against adaptive attacks?

Source: [Athalye et al., 2018]

Idea: Compose the model by stacking multiple deep neural networks.

- PixelDefend ([Song et al., 2018]): Use a generative model to project a potential adversarial example back onto the normal data manifold before feeding it into a classifier.
- Intuition: Adversarial examples mainly lie in the low-probability region of the data distribution. PixelDefend can purify adversarially perturbed images by finding the highest probability example within an ϵ -ball of the input image.
- Results: With a maximum l_∞ perturbation of $\epsilon = 0.031$, PixelDefend claims 46% accuracy on CIFAR10

Is this robust against adaptive attacks?

Source: [Athalye et al., 2018]

Idea: Compose the model by stacking multiple deep neural networks.

- PixelDefend ([Song et al., 2018]): Use a generative model to project a potential adversarial example back onto the normal data manifold before feeding it into a classifier.
- Intuition: Adversarial examples mainly lie in the low-probability region of the data distribution. PixelDefend can purify adversarially perturbed images by finding the highest probability example within an ϵ -ball of the input image.
- Results: With a maximum l_∞ perturbation of $\epsilon = 0.031$, PixelDefend claims 46% accuracy on CIFAR10

Is this robust against adaptive attacks?

Source: [Athalye et al., 2018]

Adversarial Training

Min-max Game: Anticipate the strongest adversary

Know yourself and know your enemy, and you will never be defeated. Sun Tzu's the Art of War

Find a model to minimize the loss on the strongest adversarial examples.

Adversarial Training: Train robust models by solving this min-max optimization problem:

$$\overbrace{\min_f \max_{x' \in \mathcal{P}_x} l(f(x'), y)}^{\text{Defender's goal}}$$

$\underbrace{\hspace{10em}}_{\text{Adversary's goal}}$

Adversarial Training: Linear Model (Binary classification)

Setting

- Consider the binary classification setting: $y \in \{+1, -1\}$
- The classifier f predicts class $+1$ on x if $\text{sigmoid}(w^\top x + b) > 0.5$, and -1 otherwise, where $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are model parameters
- Loss function is $l(f(x), y) = \log(1 + \exp(-y \cdot (w^\top x + b)))$
- Let \mathcal{D} be the training dataset

Objective

$$\min_{w,b} \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \max_{x' \in \mathcal{P}_x} l(f(x'), y)$$

Adversarial Training: Linear Model (Binary classification)

For the assumed model, we have

$$\nabla_{w,b} \max_{x' \in \mathcal{P}_x} l(f(x'), y) = \nabla_{w,b} l(f(x^*), y)$$

where $x^* = \arg \max_{x' \in \mathcal{P}_x} l(f(x'), y)$. See Danskin's theorem.

Adversarial Training: Solve the min-max problem using gradient descent. In each iteration, the algorithm works as follows:

- Generate the strongest adversarial example for each data point in the training data
- Update the model to minimize the loss on the adversarial examples

Adversarial Training: Linear Model (Binary classification)

- Suppose perturbation set is l_∞ ball around x of size ϵ .

$$\mathcal{P}_x = \{x + \delta : \|\delta\|_\infty \leq \epsilon\}$$

- The optimal adversarial example is $x^* = x - y\epsilon \text{sign}(w)$, where $\text{sign}(\cdot)$ computes the sign of each dimension of the input.
- The loss on adversarial examples is

$$\begin{aligned} l(f(x^*), y) &= \min_{w,b} l\left(f\left(\underbrace{x - y\epsilon \text{sign}(w)}_{\text{optimal adversarial example}}\right), y\right) \\ &= \log\left(1 + \exp((-y \cdot (w^\top x + b)) + \epsilon\|w\|_1)\right) \end{aligned}$$

Adversarial Training: Linear Model (Binary classification)

Objective:

$$\min_{w,b} \log \left(1 + \exp \left((-y \cdot (w^\top x + b)) + \epsilon \|w\|_1 \right) \right)$$

Solution: The adversarial loss is still convex in model parameters. Thus, we can solve it exactly, i.e., gradient descent method will approach the globally optimal solution.

Adversarial Training in the general case (e.g., Neural Networks)

Idea: minimizing the loss on the strongest adversarial examples

- To compute the gradient of the robust loss, we need to find the perturbation that maximizes the loss in $\mathcal{P}_{p,\epsilon}(x)$ and compute the gradient of the model at that point
- We don't know how to compute the “optimal” perturbation.
 - But, we can use strongest known attacks instead, as a proxy.
- This is how one iteration of parameter update in adversarial training works:
 1. sample (x, y) from the training set
 2. using PGD (or other strong attacks), find a point $x' \in \mathcal{P}_{p,\epsilon}(x)$ with large $l(f_\theta(x'), y)$
 3. update θ as $\theta - \eta \nabla_\theta (l(f_\theta(x'), y))$

Source: [Madry et al., 2017]

Adversarial Training in the general case (e.g., Neural Networks)

Idea: minimizing the loss on the strongest adversarial examples

- To compute the gradient of the robust loss, we need to find the perturbation that maximizes the loss in $\mathcal{P}_{p,\epsilon}(x)$ and compute the gradient of the model at that point
- We don't know how to compute the “optimal” perturbation.
 - But, we can use strongest known attacks instead, as a proxy.
- This is how one iteration of parameter update in adversarial training works:
 1. sample (x, y) from the training set
 2. using PGD (or other strong attacks), find a point $x' \in \mathcal{P}_{p,\epsilon}(x)$ with large $l(f_\theta(x'), y)$
 3. update θ as $\theta - \eta \nabla_\theta(l(f_\theta(x'), y))$

Source: [Madry et al., 2017]

Adversarial Training in the general case (e.g., Neural Networks)

Idea: minimizing the loss on the strongest adversarial examples

- To compute the gradient of the robust loss, we need to find the perturbation that maximizes the loss in $\mathcal{P}_{p,\epsilon}(x)$ and compute the gradient of the model at that point
- We don't know how to compute the “optimal” perturbation.
 - But, we can use strongest known attacks instead, as a proxy.
- This is how one iteration of parameter update in adversarial training works:
 1. sample (x, y) from the training set
 2. using PGD (or other strong attacks), find a point $x' \in \mathcal{P}_{p,\epsilon}(x)$ with large $l(f_\theta(x'), y)$
 3. update θ as $\theta - \eta \nabla_\theta (l(f_\theta(x'), y))$

Source: [Madry et al., 2017]

Adversarial Training in the general case (e.g., Neural Networks)

Idea: minimizing the loss on the strongest adversarial examples

- To compute the gradient of the robust loss, we need to find the perturbation that maximizes the loss in $\mathcal{P}_{p,\epsilon}(x)$ and compute the gradient of the model at that point
- We don't know how to compute the “optimal” perturbation.
 - But, we can use strongest known attacks instead, as a proxy.
- This is how one iteration of parameter update in adversarial training works:
 1. sample (x, y) from the training set
 2. using PGD (or other strong attacks), find a point $x' \in \mathcal{P}_{p,\epsilon}(x)$ with large $l(f_\theta(x'), y)$
 3. update θ as $\theta - \eta \nabla_\theta(l(f_\theta(x'), y))$

Source: [Madry et al., 2017]

Adversarial Training in the general case (e.g., Neural Networks)

Idea: minimizing the loss on the strongest adversarial examples

- To compute the gradient of the robust loss, we need to find the perturbation that maximizes the loss in $\mathcal{P}_{p,\epsilon}(x)$ and compute the gradient of the model at that point
- We don't know how to compute the “optimal” perturbation.
 - But, we can use strongest known attacks instead, as a proxy.
- This is how one iteration of parameter update in adversarial training works:
 1. sample (x, y) from the training set
 2. using PGD (or other strong attacks), find a point $x' \in \mathcal{P}_{p,\epsilon}(x)$ with large $l(f_\theta(x'), y)$
 3. update θ as $\theta - \eta \nabla_\theta (l(f_\theta(x'), y))$

Source: [Madry et al., 2017]

Adversarial Training in the general case (e.g., Neural Networks)

Idea: minimizing the loss on the strongest adversarial examples

- To compute the gradient of the robust loss, we need to find the perturbation that maximizes the loss in $\mathcal{P}_{p,\epsilon}(x)$ and compute the gradient of the model at that point
- We don't know how to compute the “optimal” perturbation.
 - But, we can use strongest known attacks instead, as a proxy.
- This is how one iteration of parameter update in adversarial training works:
 1. sample (x, y) from the training set
 2. using PGD (or other strong attacks), find a point $x' \in \mathcal{P}_{p,\epsilon}(x)$ with large $l(f_\theta(x'), y)$
 3. update θ as $\theta - \eta \nabla_\theta(l(f_\theta(x'), y))$

Source: [Madry et al., 2017]

Adversarial Training in the general case (e.g., Neural Networks)

Idea: minimizing the loss on the strongest adversarial examples

- To compute the gradient of the robust loss, we need to find the perturbation that maximizes the loss in $\mathcal{P}_{p,\epsilon}(x)$ and compute the gradient of the model at that point
- We don't know how to compute the “optimal” perturbation.
 - But, we can use strongest known attacks instead, as a proxy.
- This is how one iteration of parameter update in adversarial training works:
 1. sample (x, y) from the training set
 2. using PGD (or other strong attacks), find a point $x' \in \mathcal{P}_{p,\epsilon}(x)$ with large $l(f_\theta(x'), y)$
 3. update θ as $\theta - \eta \nabla_\theta(l(f_\theta(x'), y))$

Source: [Madry et al., 2017]

Effectiveness of adversarial training - MNIST Leaderboard

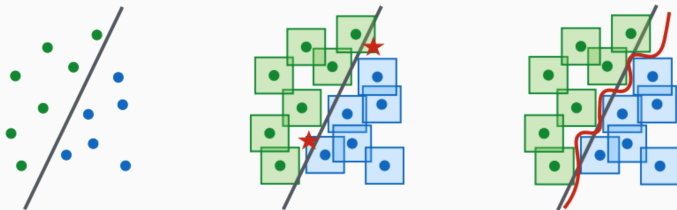
Adversarial training has stood the test of time (against empirical attacks).

The accuracy of a standard model on adversarial examples generated by FGSM is 6.4% (l_∞ perturbation set of size 0.3).

Attack	Submitted by	Accuracy	Submission Date
Guided Local Attack	Siyuan Yi	88.00%	Aug 30, 2021
PCROS Attack	Chen Wan	88.04%	Oct 28, 2020
Adaptive Distributionally Adversarial Attack	Tianhang Zheng	88.06%	Feb 29, 2019
PGD attack with Output Diversified Initialization	Yusuke Tashiro	88.13%	Feb 15, 2020
Square Attack	Francesco Croce	88.25%	Jan 14, 2020
First-Order Adversary with Quantized Gradients	Zhuanghua Liu	88.32%	Oct 16, 2019
MultiTargeted	Sven Gowal	88.36%	Aug 28, 2019
Interval Attacks	Shiqi Wang	88.42%	Feb 28, 2019
Distributionally Adversarial Attack merging multiple hyperparameters	Tianhang Zheng	88.56%	Jan 13, 2019
Interval Attacks	Shiqi Wang	88.59%	Jan 6, 2019
Distributionally Adversarial Attack	Tianhang Zheng	88.79%	Aug 13, 2018
First-order attack on logit difference for optimally chosen target label	Samarth Gupta	88.85%	May 23, 2018
100-step PGD on the cross-entropy loss with 50 random restarts	(initial entry)	89.62%	Nov 6, 2017
100-step PGD on the CW loss with 50 random restarts	(initial entry)	89.71%	Nov 6, 2017
100-step PGD on the cross-entropy loss	(initial entry)	92.52%	Nov 6, 2017
100-step PGD on the CW loss	(initial entry)	93.04%	Nov 6, 2017
FGSM on the cross-entropy loss	(initial entry)	96.36%	Nov 6, 2017

Why does adversarial training work?

Adversarial training pushes points away from the decision boundary

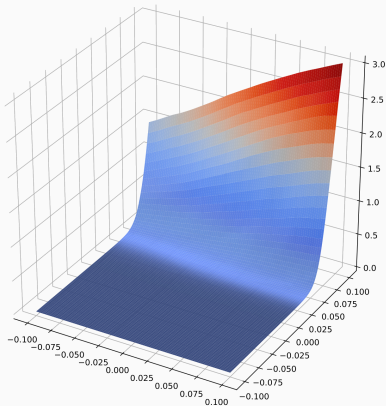


(left) benign points, (center) adversarial examples in a regular model, (right) adversarial examples in a robust model obtained from adversarial training

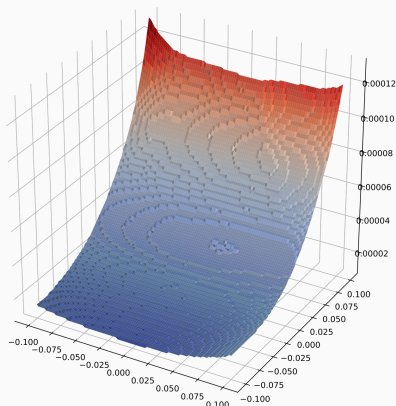
Source: [Madry et al., 2017]

Why does adversarial training work?

Robust models have a smoother loss surface



Standard Model



Robust Model

Source: Zico Kolter and Aleksander Madry, Adversarial Robustness - Theory and Practice

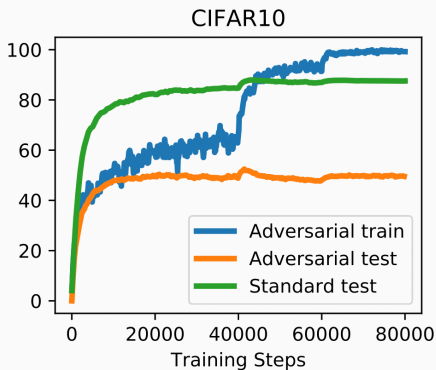
Robustness does not come for free

	Standard Model	Robust Model
Adversarial test data	3.5%	45.8%
Adversarial training data	-	100%
Test data	95.2%	87.3%
Training data	100%	100%

See the drop in accuracy on benign test data.

Source: [Raghunathan et al., 2019]

Robustness generalization to test data



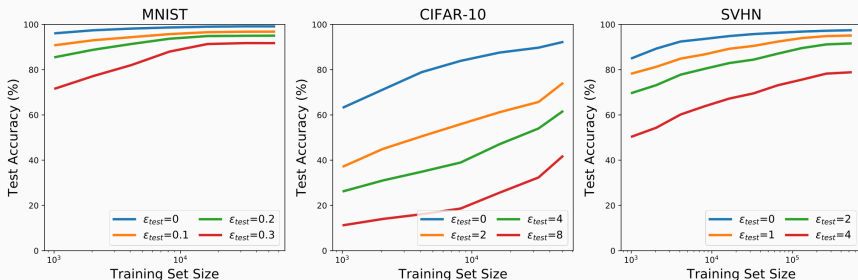
Source: [Schmidt et al., 2018]

Robustness against different attacks

CIFAR10 dataset with l_∞ perturbation with $\epsilon = 8$			
Model \ Adversary	Natural	FGSM	PGD
Standard training	92.7%	27.5%	1.2%
Adv. training (FGSM)	87.4%	90.9%	0.0%
Adv. training (PGD)	79.4%	51.7%	47.1%

Source: [Madry et al., 2017]

Improving generalization by training on more data



Prediction accuracy on adversarial examples - Adversarially robust generalization performance is a function of training data size for l_∞ adversaries.

Source: [Schmidt et al., 2018]

Limitations of the adversarial training

- The performance of the model depends on the optimality of our solution for the inner maximization
- The defense does not provide theoretical guarantees about the robustness

Certified Robustness

- Can we “guarantee” robustness for a class of perturbations?
- On a test point x , the prediction of the model won't change within the perturbation set \mathcal{P}_x .
- For example, for the l norm perturbation set, the certified robustness guarantee ensures that the prediction of the model remains constant for any points within l norm ball around x .

- Can we “guarantee” robustness for a class of perturbations?
- On a test point x , the prediction of the model won't change within the perturbation set \mathcal{P}_x .
- For example, for the l norm perturbation set, the certified robustness guarantee ensures that the prediction of the model remains constant for any points within l norm ball around x .

- Can we “guarantee” robustness for a class of perturbations?
- On a test point x , the prediction of the model won't change within the perturbation set \mathcal{P}_x .
- For example, for the l norm perturbation set, the certified robustness guarantee ensures that the prediction of the model remains constant for any points within l norm ball around x .

Certified robustness for Lipschitz functions

- What the adversary exploits is the **lack of stability** in the model: small changes in the input result in large changes in the output
- Lipschitz continuous functions should be **robust**, as by definition

$$|f(x) - f(x')| \leq L \|x - x'\| .$$

Certified robustness for Lipschitz functions

- What the adversary exploits is the **lack of stability** in the model: small changes in the input result in large changes in the output
- Lipschitz continuous functions should be **robust**, as by definition

$$|f(x) - f(x')| \leq L \|x - x'\| .$$

Certified robustness for Lipschitz functions

- Let $f : \mathbb{R}^d \rightarrow [0, 1]^k$, and $c(x) = \arg \max_{i \in [k]} f_i(x)$
- c is ϵ -robust at x if $c(x + \delta) = c(x), \forall \delta : \|\delta\| \leq \epsilon$
- If f is L -Lipschitz, then c is ϵ -robust at x with $\epsilon = \frac{1}{2L}(p_A - p_B)$, where $p_A = \max_i f_i(x)$, and p_B is the probability of the “runner-up”

Certified robustness for Lipschitz functions

- Let $f : \mathbb{R}^d \rightarrow [0, 1]^k$, and $c(x) = \arg \max_{i \in [k]} f_i(x)$
- c is ϵ -robust at x if $c(x + \delta) = c(x), \forall \delta : \|\delta\| \leq \epsilon$
- If f is L -Lipschitz, then c is ϵ -robust at x with $\epsilon = \frac{1}{2L}(p_A - p_B)$, where $p_A = \max_i f_i(x)$, and p_B is the probability of the “runner-up”

Certified robustness for Lipschitz functions

- Let $f : \mathbb{R}^d \rightarrow [0, 1]^k$, and $c(x) = \arg \max_{i \in [k]} f_i(x)$
- c is ϵ -robust at x if $c(x + \delta) = c(x), \forall \delta : \|\delta\| \leq \epsilon$
- If f is L -Lipschitz, then c is ϵ -robust at x with $\epsilon = \frac{1}{2L}(p_A - p_B)$, where $p_A = \max_i f_i(x)$, and p_B is the probability of the “runner-up”

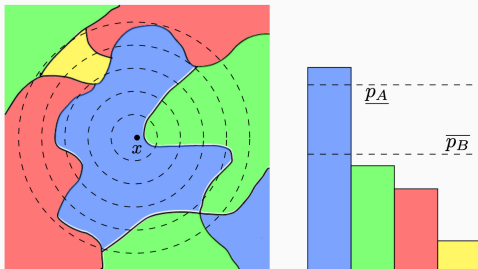
Certified robustness for Lipschitz functions

Proof: If f is L -Lipschitz, then $\forall \delta : \|\delta\| \leq \frac{1}{2L}(p_A - p_B)$, we have $c(x + \delta) = c(x)$

- Due to Lipschitz continuity, $|p_A - p'_A| \leq L\epsilon = \frac{(p_A - p_B)}{2}$.
- It implies that $p'_A \geq p_A - \frac{(p_A - p_B)}{2} = \frac{(p_A + p_B)}{2}$
- Similarly, $|p_B - p'_B| \leq L\epsilon = \frac{(p_A - p_B)}{2}$.
- $p'_B \leq p_B + \frac{(p_A - p_B)}{2} = \frac{(p_A + p_B)}{2}$
- It implies that $p'_B \leq p'_A$. As a result, $c(x + \delta) = c(x)$.

Randomized Smoothing

- Neural networks are not Lipschitz
- We can use randomized smoothing to make $f(x)$ stable around x



(left) decision boundaries and a given input x ; The dotted lines are the level sets of the Gaussian noise distribution centered at x , (right) The prediction confidence values for different classes

Source: [Cohen et al., 2019]

Randomized Smoothing

- Randomized smoothing: smoothing the prediction for a given model by taking the majority votes over predictions on points within l_p norm ball around x

Theorem: When the noise δ is sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2 \mathbf{I})$, no adversarial example exists within the radius

$$\frac{\sigma}{2} (\Phi^{-1}(\underline{p}_A) - \Phi^{-1}(\overline{p}_B))$$

Φ^{-1} is the inverse CDF of standard normal distribution

Idea of the **proof**: $\Phi^{-1}(f_i(x))$ is $\frac{1}{\sigma}$ -Lipschitz

Source: [Cohen et al., 2019]

Evaluate the robustness of the model

Randomized smoothing in practice:

- The probability \underline{p}_A and \overline{p}_B can only be approximated empirically
- The model is trained with randomized data to obtain a better test accuracy on clean data

Metrics for evaluation:

- Certified radius R : the smooth classifier is robust around x within l_2 radius R
- Certified accuracy: the fraction of test set which smooth classifier predicts correctly with a certified radius larger than R
- The model with a higher certified accuracy at a larger certified radius is more robust

Evaluate the robustness of the model

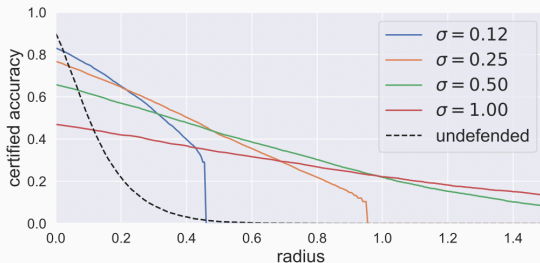
Randomized smoothing in practice:

- The probability \underline{p}_A and \overline{p}_B can only be approximated empirically
- The model is trained with randomized data to obtain a better test accuracy on clean data

Metrics for evaluation:

- Certified radius R : the smooth classifier is robust around x within l_2 radius R
- Certified accuracy: the fraction of test set which smooth classifier predicts correctly with a certified radius larger than R
- The model with a higher certified accuracy at a larger certified radius is more robust

Evaluate the robustness of the model



Approximate certified accuracy attained by randomized smoothing on CIFAR-10. The dashed black line is the accuracy of an undefended classifier on adversarial examples generated by the adversarial attack.

Source: [Cohen et al., 2019]

Takeaways

1. Principle of designing robust algorithms: anticipate the attack
2. Adversarial training algorithms learn robust models by playing the min-max game between adversary and defender
3. Randomized smoothing makes the model stable around test points
4. Robustness does not come for free

Verification of robustness in neural networks. See [Singh et al., 2019].

- Can we verify robustness of any given neural network, for any given data point and perturbation set?
- Verification techniques aim at modeling the range of values that any activation function, from the input to the output, would take. Based on this, we can verify if the predicted class of the model could have been different if the input was perturbed by the adversary.

Use existing models to denoise “randomized smoothing”.

See [Carlini et al., 2022].

- Randomized smoothing results in some drop in model accuracy (even for benign data).
- One technique to alleviate this problem is to use denoising models after we randomize the input.
- There are so many pre-trained models that perform accurate denoising and classification for image data. We can use them to achieve very good accuracy with certified robustness.



Athalye, A., Carlini, N., and Wagner, D. (2018).

**Obfuscated gradients give a false sense of security:
Circumventing defenses to adversarial examples.**

In International conference on machine learning, pages 274–283.
PMLR.



Carlini, N., Tramer, F., Dvijotham, K. D., Rice, L., Sun, M., and
Kolter, J. Z. (2022).

(certified!!) adversarial robustness for free!

arXiv preprint arXiv:2206.10550.



Cohen, J., Rosenfeld, E., and Kolter, Z. (2019).

Certified adversarial robustness via randomized smoothing.

In International Conference on Machine Learning, pages 1310–1320.
PMLR.



Dhillon, G. S., Azizzadenesheli, K., Lipton, Z. C., Bernstein, J. D., Kossaifi, J., Khanna, A., and Anandkumar, A. (2018).

Stochastic activation pruning for robust adversarial defense.


In International Conference on Learning Representations.



Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017).


Towards deep learning models resistant to adversarial attacks.

arXiv preprint arXiv:1706.06083.

 Raghunathan, A., Xie, S. M., Yang, F., Duchi, J. C., and Liang, P. (2019).

Adversarial training can hurt generalization.

arXiv preprint arXiv:1906.06032.

 Schmidt, L., Santurkar, S., Tsipras, D., Talwar, K., and Madry, A. (2018).

Adversarially robust generalization requires more data.

Advances in neural information processing systems, 31.

 Singh, G., Gehr, T., Püschel, M., and Vechev, M. (2019).

An abstract domain for certifying neural networks.

Proceedings of the ACM on Programming Languages,
3(POPL):1–30.



Song, Y., Kim, T., Nowozin, S., Ermon, S., and Kushman, N. (2018).

Pixeldefend: Leveraging generative models to understand and defend against adversarial examples.

In International Conference on Learning Representations.