

Adversarial Examples & Robustness Evaluation

Topics in Adversarial Machine Learning

[Seminar] – SoSe 2023/24

1. Towards Deep Learning Models Resistant to Adversarial Attacks Madry et al.
2. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples Athalye et al.

Shreyash Arya (7015279)
May 10, 2023

Agenda

Agenda

- What is an adversarial attack?

Agenda

- What is an adversarial attack?
- Why are they important?

Agenda

- What is an adversarial attack?
- Why are they important?
- How can we defend against them?

Agenda

- What is an adversarial attack?
- Why are they important?
- How can we defend against them?
 - Experiments and Results

Agenda

- What is an adversarial attack?
- Why are they important?
- How can we defend against them?
 - Experiments and Results
- Discussion and Conclusion

ADVERSARIAL ATTACK?

Generating an Adversarial Attack

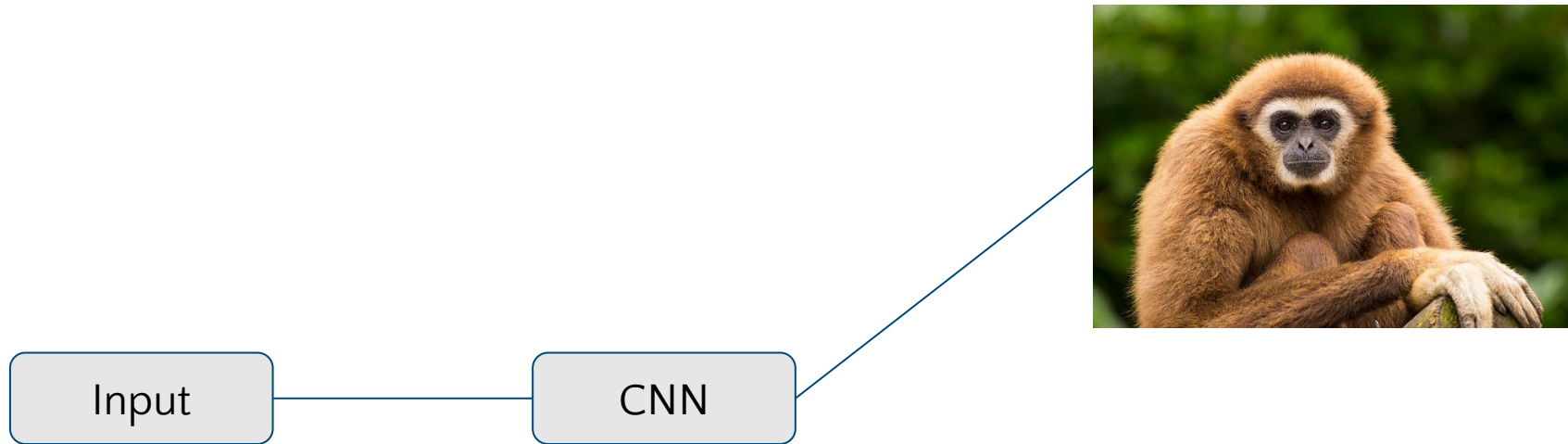
Generating an Adversarial Attack

Input

Generating an Adversarial Attack

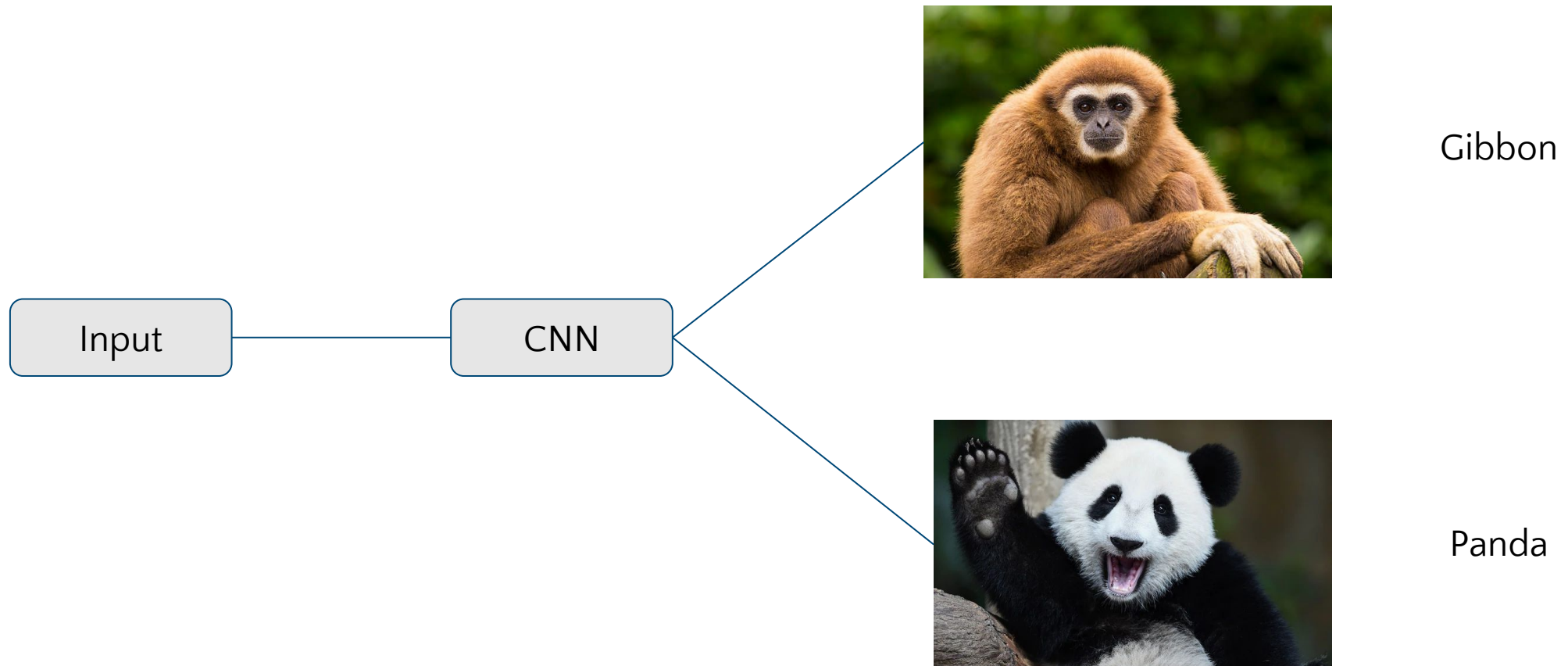


Generating an Adversarial Attack



Gibbon

Generating an Adversarial Attack



Generating an Adversarial Attack

Generating an Adversarial Attack



Generating an Adversarial Attack



“panda”

57.7% confidence

Generating an Adversarial Attack



+ .007 ×



“panda”

57.7% confidence

Generating an Adversarial Attack



“panda”

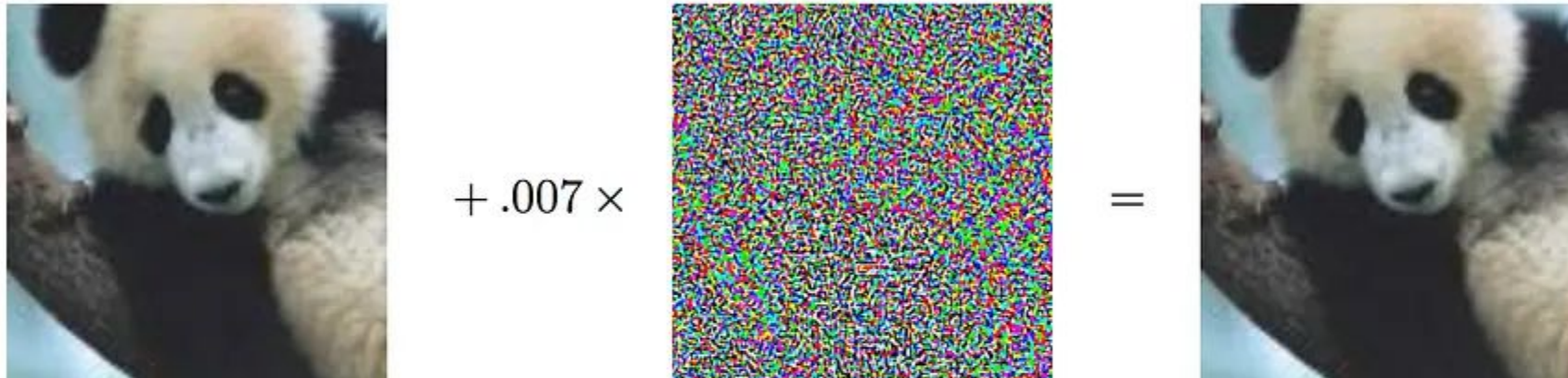
57.7% confidence

+ .007 ×



noise

Generating an Adversarial Attack



“panda”
57.7% confidence

noise

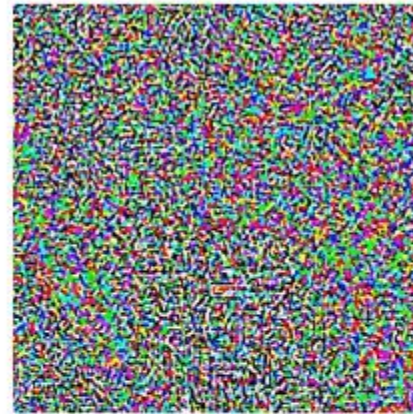
Generating an Adversarial Attack



“panda”

57.7% confidence

+ .007 ×



noise

=



“gibbon”

99.3% confidence

Concerns of Adversarial Attacks

Concerns of Adversarial Attacks

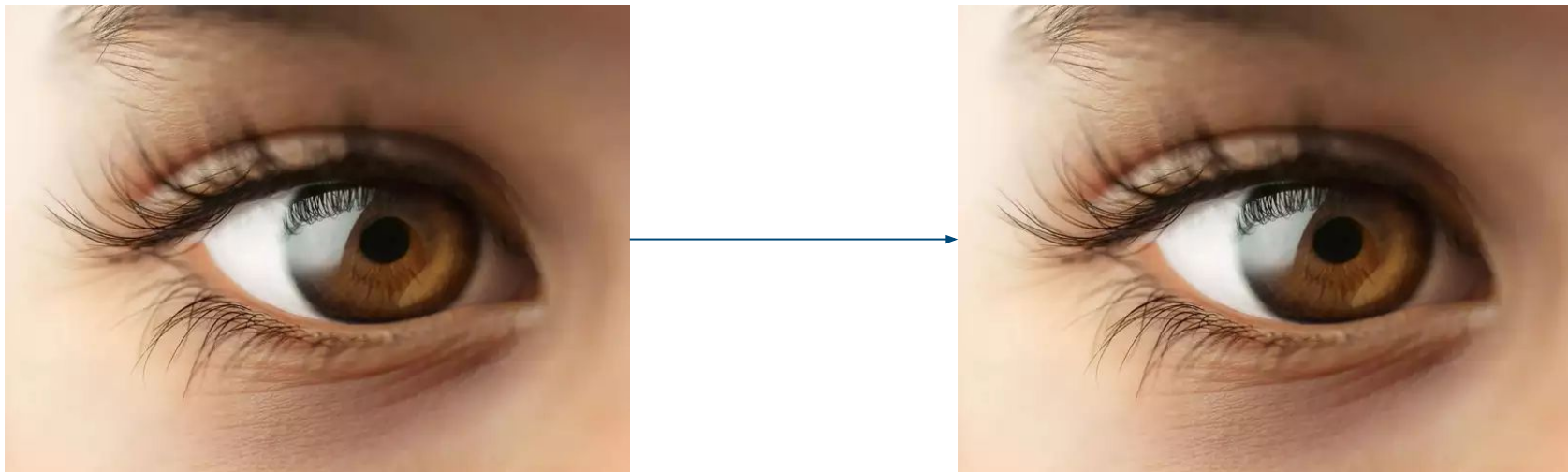


Concerns of Adversarial Attacks



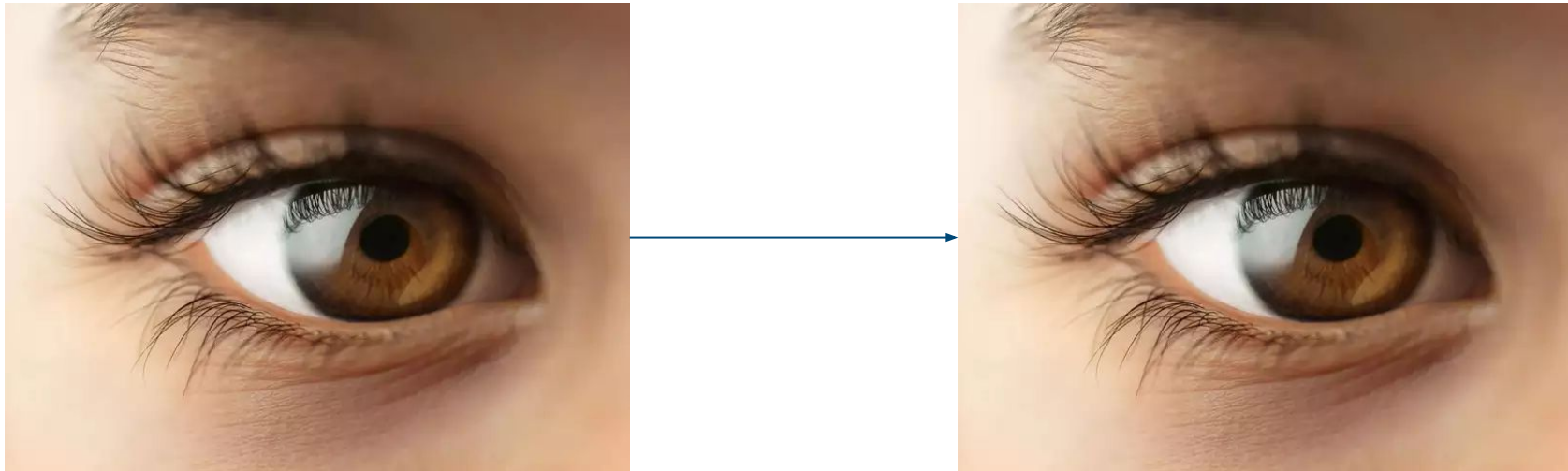
Google Images
23.5% match

Concerns of Adversarial Attacks



Google Images
23.5% match

Concerns of Adversarial Attacks



Google Images
23.5% match

Shreyash Arya
74.9% match

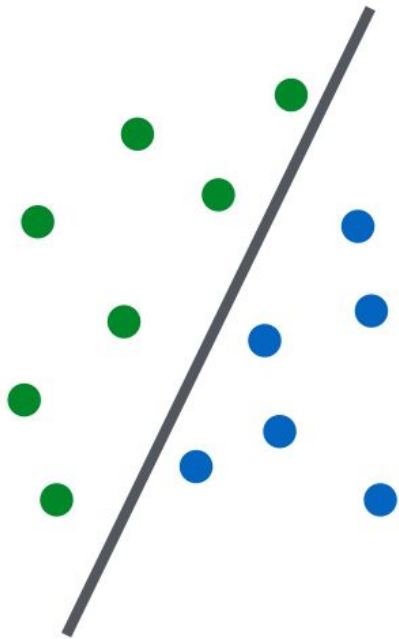
Concerns of Adversarial Attacks

Concerns of Adversarial Attacks

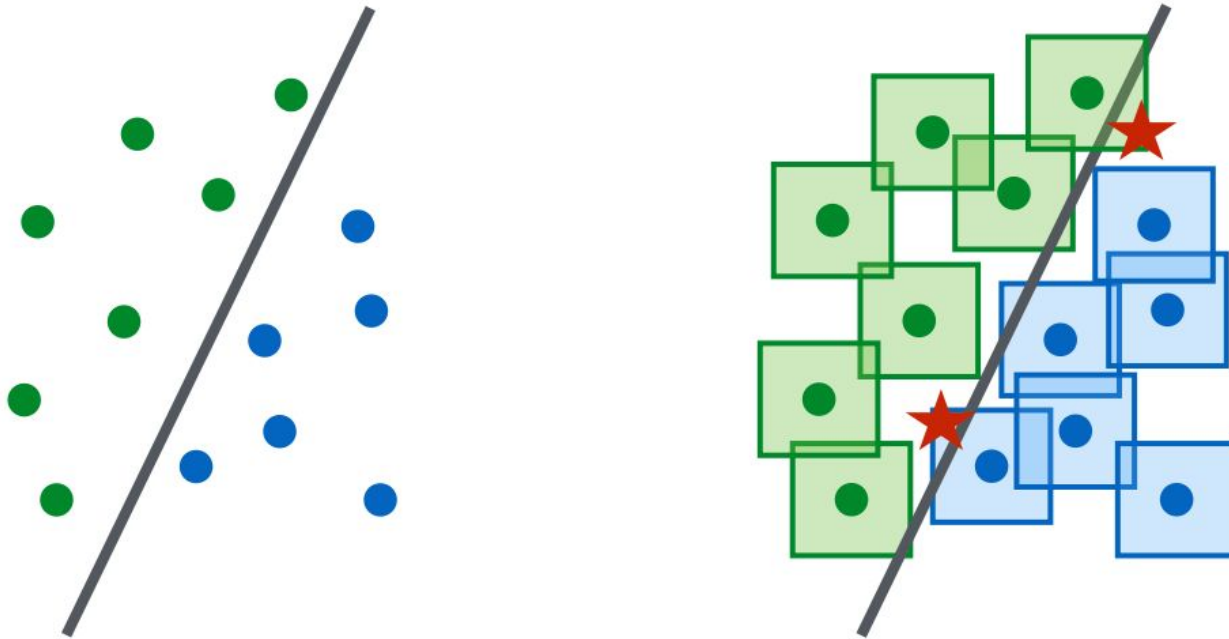


How adversarial attacks happen?

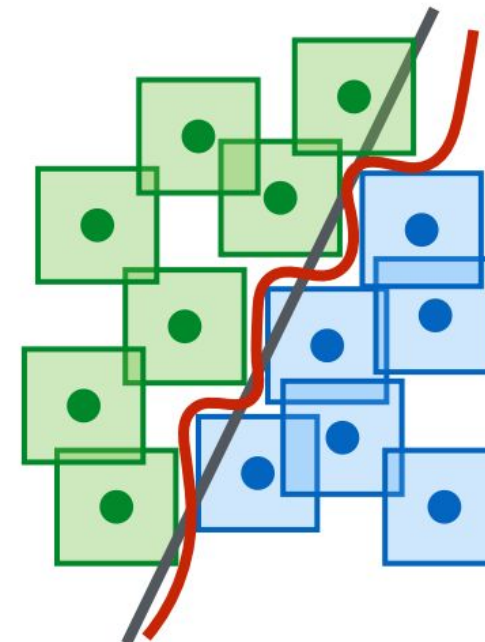
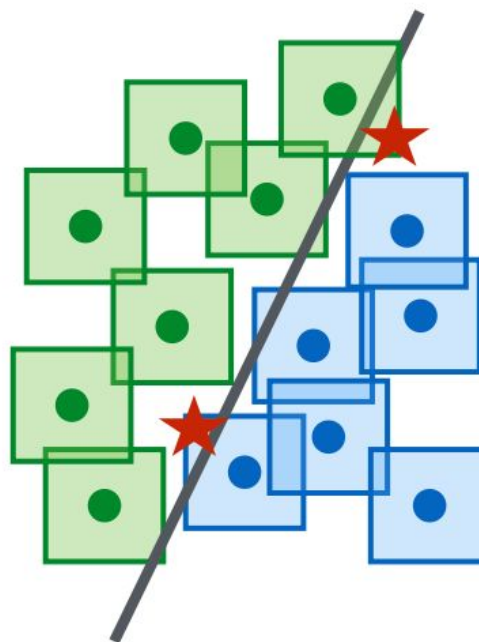
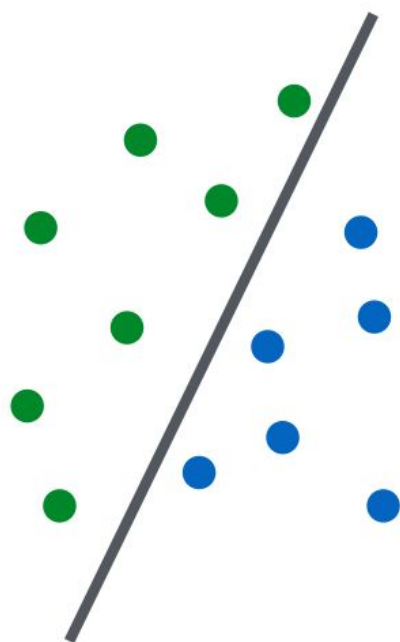
How adversarial attacks happen?



How adversarial attacks happen?



How adversarial attacks happen?



Problem Statement

Problem Statement

- What is an adversarial attack?

- What is an adversarial attack?
 - How to produce strong adversarial examples?

- What is an adversarial attack?
 - How to produce strong adversarial examples?
 - Fool model with high confidence and only small perturbation

- What is an adversarial attack?
 - How to produce strong adversarial examples?
 - Fool model with high confidence and only small perturbation
- How to defend against them?

- What is an adversarial attack?
 - How to produce strong adversarial examples?
 - Fool model with high confidence and only small perturbation
- How to defend against them?
 - Train such that no adversarial or difficult to find examples

Problem Statement

- Formally

Problem Statement

- Saddle point” min/max optimization problem

Problem Statement

- Saddle point” min/max optimization problem
 - Inner Maximization

- Saddle point” min/max optimization problem
 - Inner Maximization
 - Adversarial version of given input x that achieves high loss

- Saddle point” min/max optimization problem
 - Inner Maximization
 - Adversarial version of given input x that achieves high loss
 - Outer Minimization

- Saddle point” min/max optimization problem
 - Inner Maximization
 - Adversarial version of given input x that achieves high loss
 - Outer Minimization
 - Find model parameters such that adversarial loss of inner attack problem is minimized

Defining an Attack

Defining an Attack

- Original model goal is to solve

Defining an Attack

- Original model goal is to solve

$$\mathbb{E}_{(x,y) \sim \mathcal{D}} [L(x, y, \theta)]$$

Defining an Attack

- Original model goal is to solve

$$\mathbb{E}_{(x,y) \sim \mathcal{D}} [\boxed{L}(x, y, \theta)]$$

Loss
function

Defining an Attack

- Original model goal is to solve

$$\mathbb{E}_{(x,y) \sim \mathcal{D}} [L(\boxed{x}, y, \theta)]$$

Data matrix

Defining an Attack

- Original model goal is to solve

$$\mathbb{E}_{(x,y) \sim \mathcal{D}} [L(x, \boxed{y}, \theta)]$$

Label vector

Defining an Attack

- Original model goal is to solve

$$\mathbb{E}_{(x,y) \sim \mathcal{D}} [L(x, y, \theta)]$$

Model
parameters

Defining an Attack

- Precise definition of attack?

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

Inner Maximization

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

Inner Maximization

Adversarial version of x
achieving a high loss

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \text{ where } \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

Outer Minimization

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \text{ where } \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

Outer Minimization

Model parameters that
minimize the adversarial
loss of inner attack

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Fast Gradient Sign Method (FGSM)

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Fast Gradient Sign Method (FGSM)

$$x + \varepsilon \operatorname{sgn}(\nabla_x L(\theta, x, y))$$

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Fast Gradient Sign Method (FGSM)
 - One-step scheme

$$x + \varepsilon \operatorname{sgn}(\nabla_x L(\theta, x, y))$$

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Fast Gradient Sign Method (FGSM)
 - One-step scheme
- Projected Gradient Descent (PGD)

$$x + \varepsilon \operatorname{sgn}(\nabla_x L(\theta, x, y))$$

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Fast Gradient Sign Method (FGSM)
 - One-step scheme
- Projected Gradient Descent (PGD)

$$x + \varepsilon \operatorname{sgn}(\nabla_x L(\theta, x, y))$$

$$x^{t+1} = \Pi_{x+\mathcal{S}} (x^t + \alpha \operatorname{sgn}(\nabla_x L(\theta, x, y)))$$

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Fast Gradient Sign Method (FGSM)
 - One-step scheme
- Projected Gradient Descent (PGD)
 - Multi-step variant

$$x + \varepsilon \operatorname{sgn}(\nabla_x L(\theta, x, y))$$

$$x^{t+1} = \Pi_{x+\mathcal{S}} (x^t + \alpha \operatorname{sgn}(\nabla_x L(\theta, x, y)))$$

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Unifying view on adversarial robustness

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Unifying view on adversarial robustness
- Parameters yielding vanishing risk corresponds to robust model under adversarial attacks

Defining an Attack

- Saddle point formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Unifying view on adversarial robustness
- Parameters yielding vanishing risk corresponds to robust model under adversarial attacks
 - Small loss for all allowed perturbations \rightarrow Guarantee!

Defence: Adversarial Training

Defence: Adversarial Training

- Increase variability in the training dataset

Defence: Adversarial Training

- Increase variability in the training dataset
- Adding FGSM/PGD inputs to the training dataset

Defence: Adversarial Training

- Increase variability in the training dataset
- Adding FGSM/PGD inputs to the training dataset
 - Examples can look identical to human eye

Defence: Adversarial Training

- Increase variability in the training dataset
- Adding FGSM/PGD inputs to the training dataset
 - Examples can look identical to human eye
- Approach relies on increased network capacity

Defence: Adversarial Training

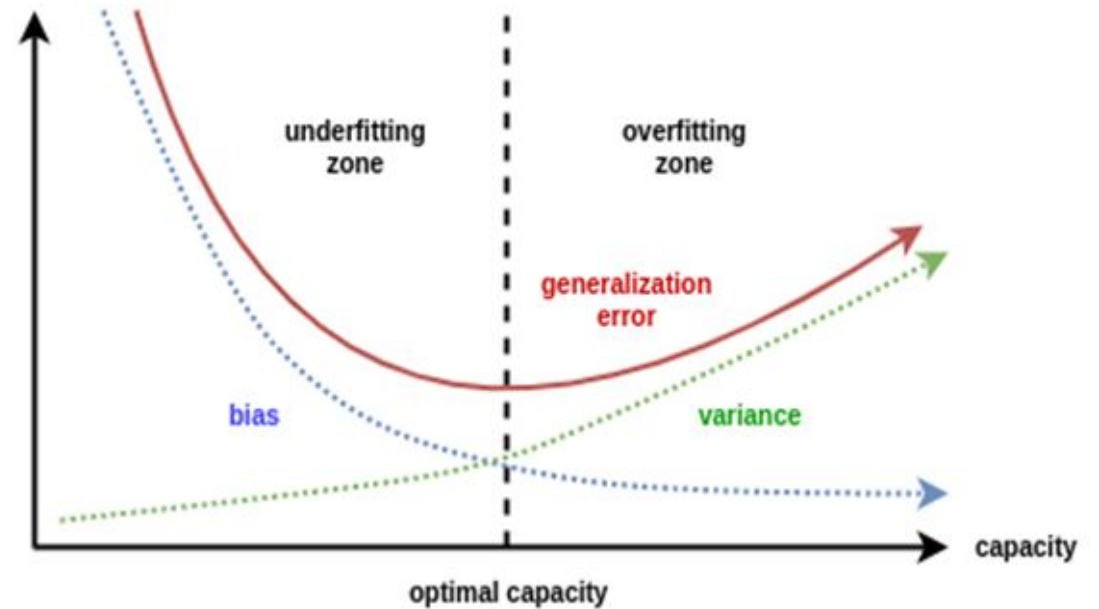
- Increase variability in the training dataset
- Adding FGSM/PGD inputs to the training dataset
 - Examples can look identical to human eye
- Approach relies on increased network capacity
 - Shallower models does not learn perturbations

Defence: Adversarial Training

- Increase variability in the training dataset
- Adding FGSM/PGD inputs to the training dataset
 - Examples can look identical to human eye
- Approach relies on increased network capacity
 - Shallower models does not learn perturbations
 - Should consider bias-variance trade-off

Defence: Adversarial Training

- Increase variability in the training dataset
- Adding FGSM/PGD inputs to the training dataset
 - Examples can look identical to human eye
- Approach relies on increased network capacity
 - Shallower models does not learn perturbations
 - Should consider bias-variance trade-off



Experiments

Experiments

- GOAL: Towards universally robust networks

- GOAL: Towards universally robust networks
 - Small inner loss \rightarrow Guarantee against first-order adversaries (gradient dependent)

- GOAL: Towards universally robust networks
 - Small inner loss \rightarrow Guarantee against first-order adversaries (gradient dependent)
 - But problem: Non-concave inner maximization and non-convex outer minimization

- GOAL: Towards universally robust networks
 - Small inner loss \rightarrow Guarantee against first-order adversaries (gradient dependent)
 - But problem: Non-concave inner maximization and non-convex outer minimization
 - Tractable solution \rightarrow Local maxima well concentrated in $x + \delta$

- GOAL: Towards universally robust networks
 - Small inner loss \rightarrow Guarantee against first-order adversaries (gradient dependent)
 - But problem: Non-concave inner maximization and non-convex outer minimization
 - Tractable solution \rightarrow Local maxima well concentrated in $x + \delta$
 - PGD being ultimate first-order adversary

- GOAL: Towards universally robust networks
 - Small inner loss → Guarantee against first-order adversaries (gradient dependent)
 - But problem: Non-concave inner maximization and non-convex outer minimization
 - Tractable solution → Local maxima well concentrated in $x + \delta$
 - PGD being ultimate first-order adversary
 - Robust against wide range of attacks

- GOAL: Towards universally robust networks
 - Small inner loss → Guarantee against first-order adversaries (gradient dependent)
 - But problem: Non-concave inner maximization and non-convex outer minimization
 - Tractable solution → Local maxima well concentrated in $x + \delta$
 - PGD being ultimate first-order adversary
 - Robust against wide range of attacks
 - Provided sufficiently large networks

- GOAL: Towards universally robust networks
 - Small inner loss → Guarantee against first-order adversaries (gradient dependent)
 - But problem: Non-concave inner maximization and non-convex outer minimization
 - Tractable solution → Local maxima well concentrated in $x + \delta$
 - PGD being ultimate first-order adversary
 - Robust against wide range of attacks
 - Provided sufficiently large networks
 - Datasets: MNIST and CIFAR-10

Landscape of Adversarial Examples

Landscape of Adversarial Examples

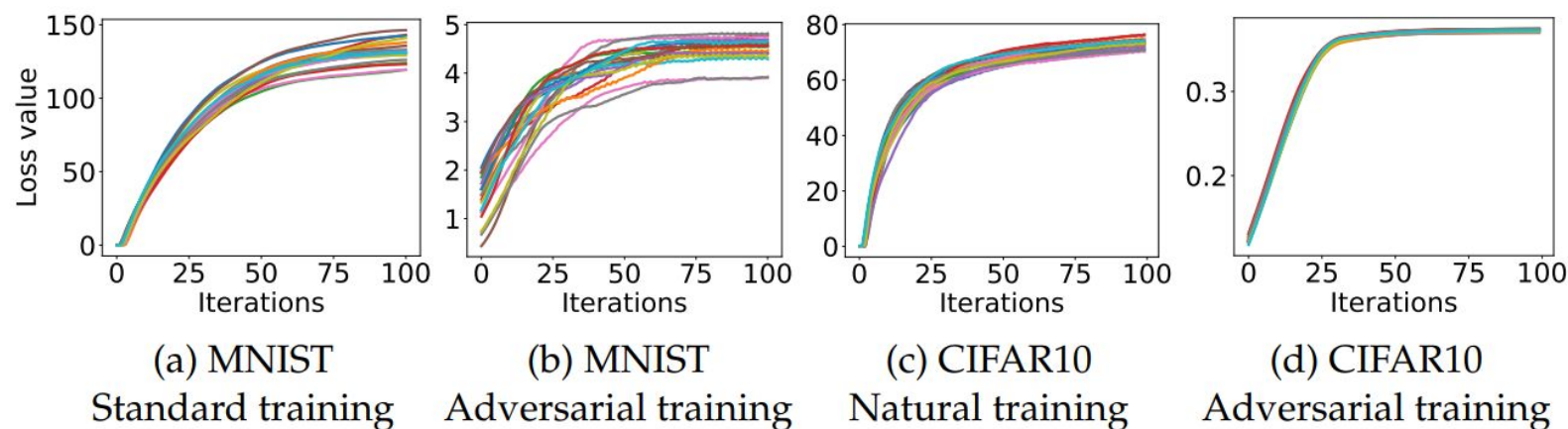
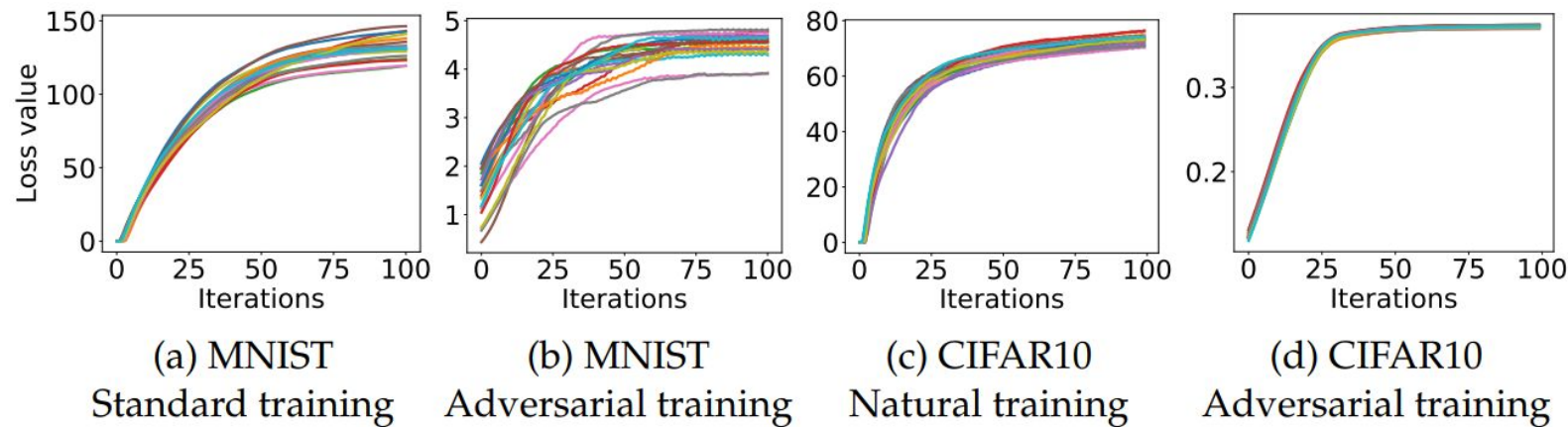


Figure 1: Cross-entropy loss values while creating an adversarial example from the MNIST and CIFAR10 evaluation datasets. The plots show how the loss evolves during 20 runs of projected gradient descent (PGD). Each run starts at a uniformly random point in the ℓ_∞ -ball around the same natural example (additional plots for different examples appear in Figure 11). The adversarial loss plateaus after a small number of iterations. The optimization trajectories and final loss values are also fairly clustered, especially on CIFAR10. Moreover, the final loss values on adversarially trained networks are significantly smaller than on their standard counterparts.

Landscape of Adversarial Examples



- Setting: Cross-entropy loss + PGD
- Outcomes:
 - Adversarial loss plateaus after small number of iterations
 - Optimization trajectories and final loss are fairly clustered (especially on CIFAR-10)
 - Final loss on adversarially trained network is significantly smaller
 - → Robust against first-order adversaries (gradient of loss wrt to input).
- Robustness guarantee even stronger from black-box and transfer attacks
 - No direct access to target network

Landscape of Adversarial Examples

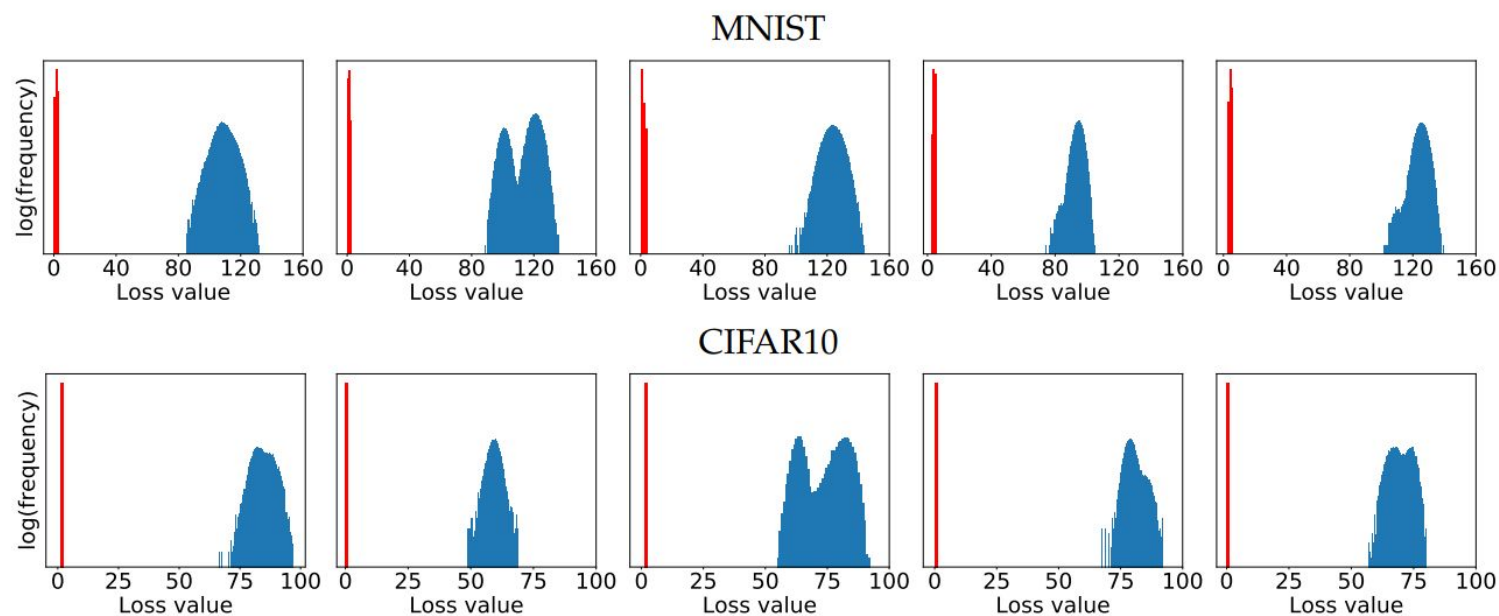


Figure 2: Values of the local maxima given by the cross-entropy loss for five examples from the MNIST and CIFAR10 evaluation datasets. For each example, we start projected gradient descent (PGD) from 10^5 uniformly random points in the ℓ_∞ -ball around the example and iterate PGD until the loss plateaus. The blue histogram corresponds to the loss on a standard network, while the red histogram corresponds to the adversarially trained counterpart. The loss is significantly smaller for the adversarially trained networks, and the final loss values are very concentrated without any outliers.

Network Capacity

Network Capacity

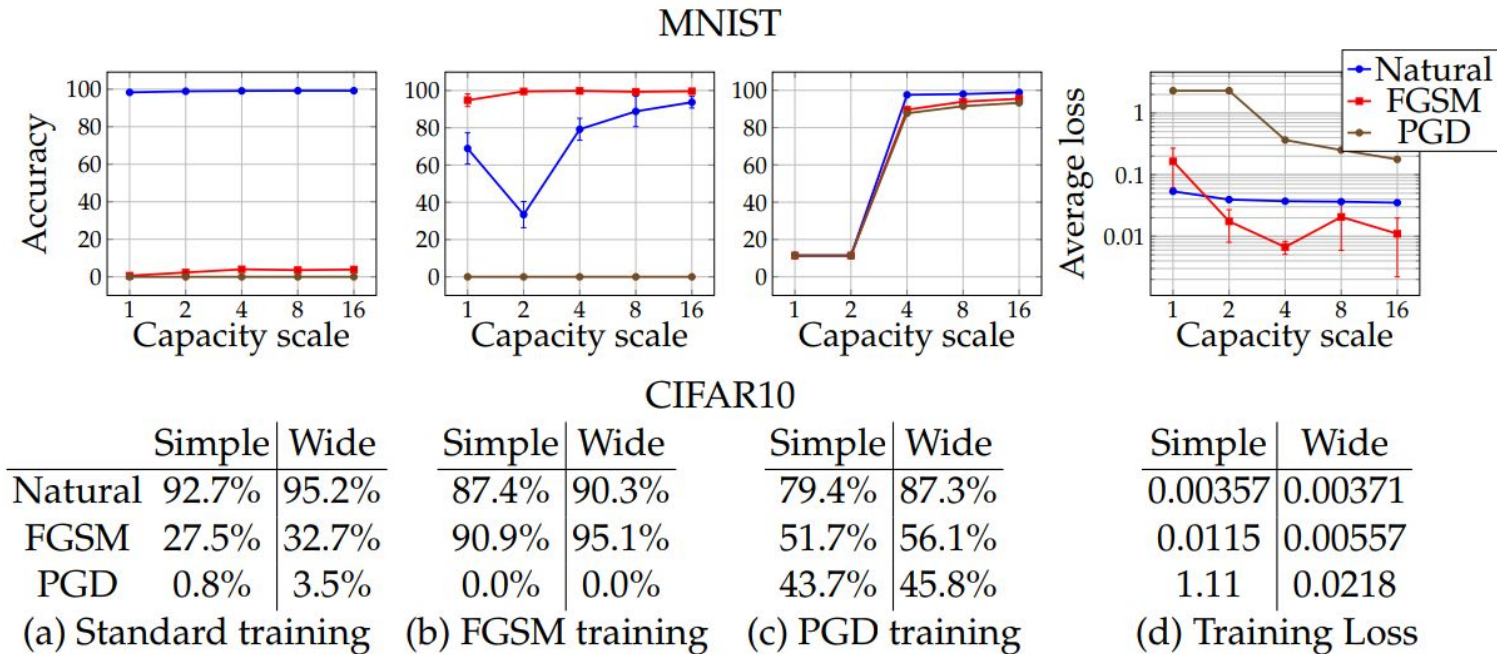
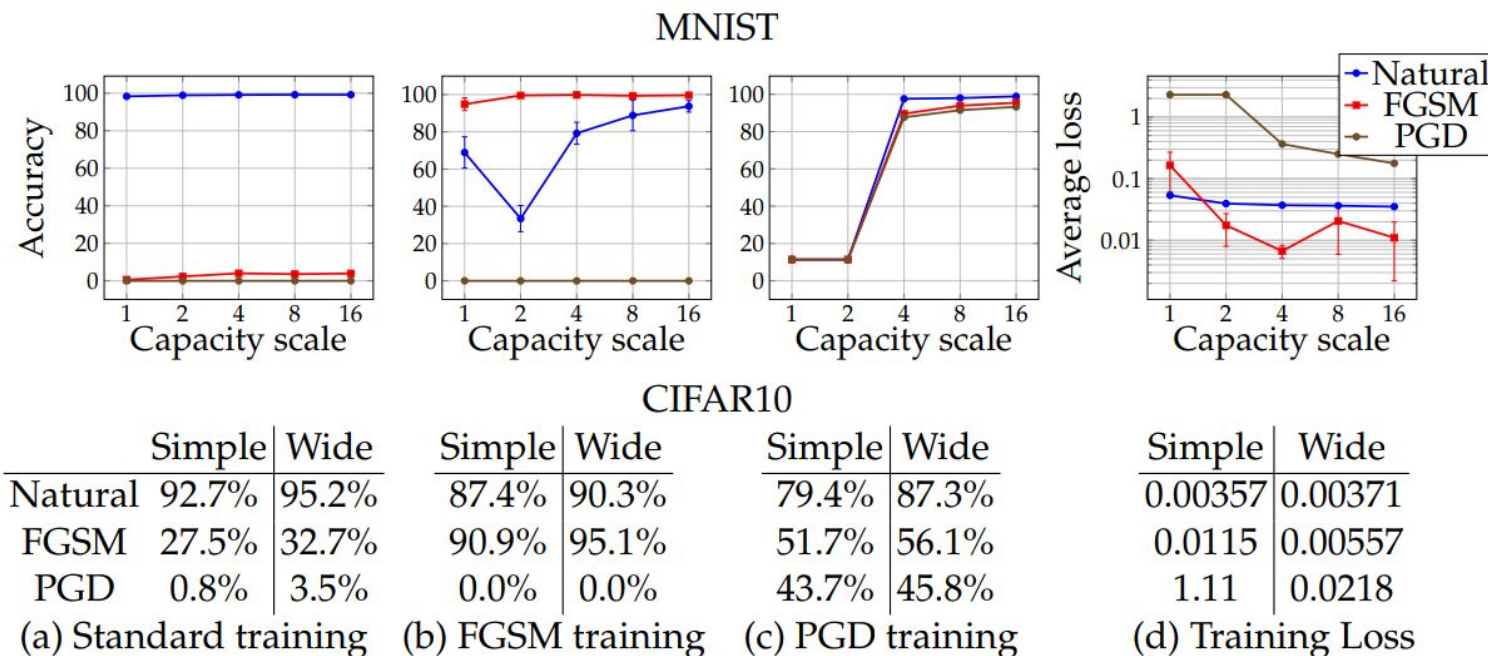


Figure 4: The effect of network capacity on the performance of the network. We trained MNIST and CIFAR10 networks of varying capacity on: (a) natural examples, (b) with FGSM-made adversarial examples, (c) with PGD-made adversarial examples. In the first three plots/tables of each dataset, we show how the standard and adversarial accuracy changes with respect to capacity for each training regime. In the final plot/table, we show the value of the cross-entropy loss on the adversarial examples the networks were trained on. This corresponds to the value of our saddle point formulation (2.1) for different sets of allowed perturbations.

Network Capacity



- Capacity alone helps
- FGSM adversaries don't increase robustness (for large ϵ)
- Weak models (small capacity + PGD) fail to learn non-trivial classifiers
- Loss decreases with capacity increase
- More capacity and stronger adversaries decrease transferability

Training Loss of adversarial examples

Training Loss of adversarial examples

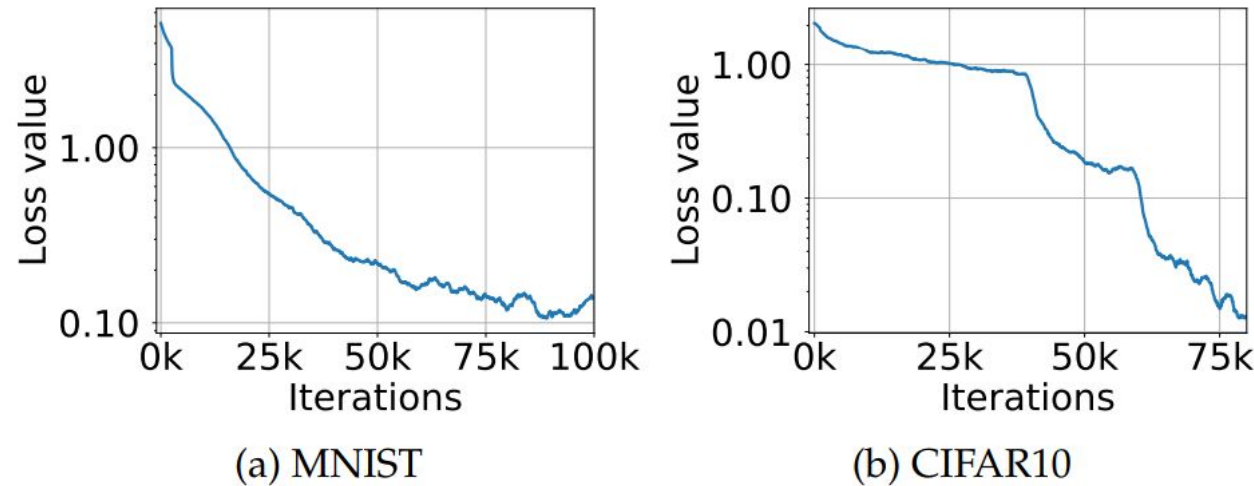


Figure 5: Cross-entropy loss on adversarial examples during training. The plots show how the adversarial loss on training examples evolves during training the MNIST and CIFAR10 networks against a PGD adversary. The sharp drops in the CIFAR10 plot correspond to decreases in training step size. These plots illustrate that we can consistently reduce the value of the inner problem of the saddle point formulation (2.1), thus producing an increasingly robust classifier.

Adversarial Robustness – MNIST

Adversarial Robustness – MNIST

Method	Steps	Restarts	Source	Accuracy
Natural	-	-	-	98.8%
FGSM	-	-	A	95.6%
PGD	40	1	A	93.2%
PGD	100	1	A	91.8%
PGD	40	20	A	90.4%
PGD	100	20	A	89.3%
Targeted	40	1	A	92.7%
CW	40	1	A	94.0%
CW+	40	1	A	93.9%
FGSM	-	-	A'	96.8%
PGD	40	1	A'	96.0%
PGD	100	20	A'	95.7%
CW	40	1	A'	97.0%
CW+	40	1	A'	96.4%
FGSM	-	-	B	95.4%
PGD	40	1	B	96.4%
CW+	-	-	B	95.7%

Table 1: MNIST: Performance of the adversarially trained network against different adversaries for $\epsilon = 0.3$. For each model of attack we show the most successful attack with bold. The source networks used for the attack are: the network itself (A) (white-box attack), an independently initialized and trained copy of the network (A'), architecture B from [29] (B).

Adversarial Robustness – CIFAR-10

Method	Steps	Source	Accuracy
Natural	-	-	87.3%
FGSM	-	A	56.1%
PGD	7	A	50.0%
PGD	20	A	45.8%
CW	30	A	46.8%
FGSM	-	A'	67.0%
PGD	7	A'	64.2%
CW	30	A'	78.7%
FGSM	-	A_{nat}	85.6%
PGD	7	A_{nat}	86.0%

Table 2: CIFAR10: Performance of the adversarially trained network against different adversaries for $\epsilon = 8$. For each model of attack we show the most effective attack in bold. The source networks considered for the attack are: the network itself (A) (white-box attack), an independently initialized and trained copy of the network (A'), a copy of the network trained on natural examples (A_{nat}).

l_∞ vs l_2

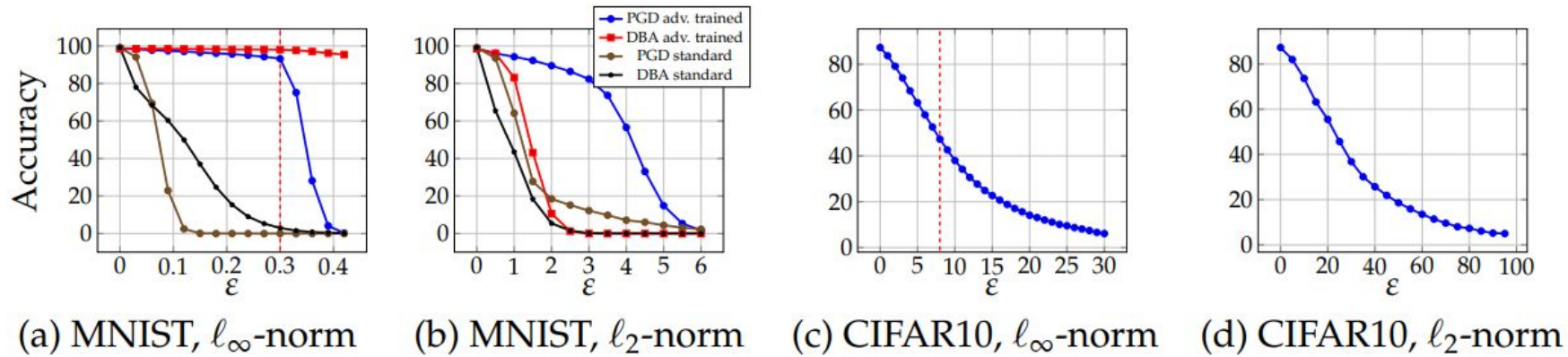


Figure 6: Performance of our adversarially trained networks against PGD adversaries of different strength. The MNIST and CIFAR10 networks were trained against $\varepsilon = 0.3$ and $\varepsilon = 8$ PGD ℓ_∞ adversaries respectively (the training ε is denoted with a red dashed lines in the ℓ_∞ plots). In the case of the MNIST adversarially trained networks, we also evaluate the performance of the Decision Boundary Attack (DBA) [4] with 2000 steps and PGD on standard and adversarially trained models. We observe that for ε less or equal to the value used during training, the performance is equal or better. For MNIST there is a sharp drop shortly after. Moreover, we observe that the performance of PGD on the MNIST ℓ_2 -trained networks is poor and significantly overestimates the robustness of the model. This is potentially due to the threshold filters learned by the model masking the loss gradients (the decision-based attack does not utilize gradients).

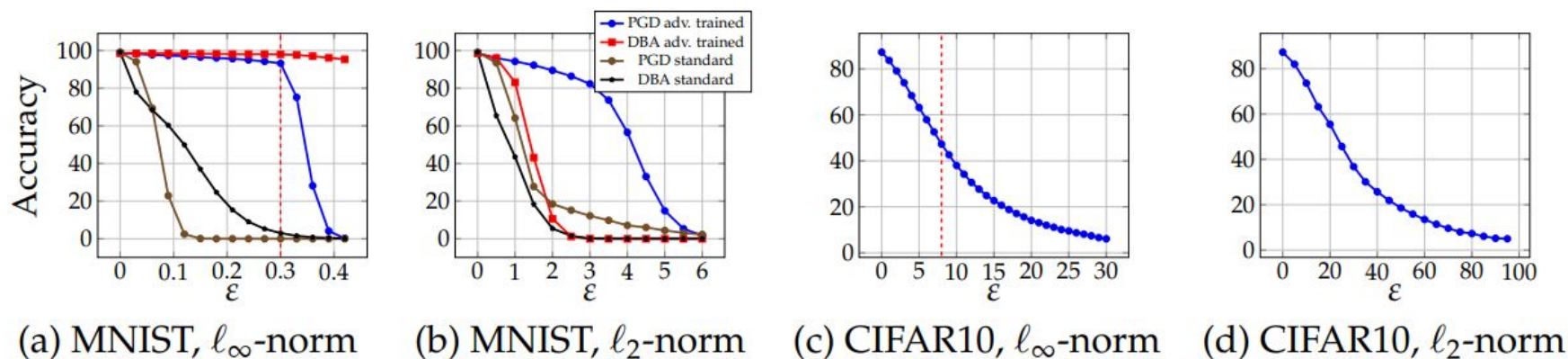


Figure 6: Performance of our adversarially trained networks against PGD adversaries of different strength. The MNIST and CIFAR10 networks were trained against $\varepsilon = 0.3$ and $\varepsilon = 8$ PGD ℓ_∞ adversaries respectively (the training ε is denoted with a red dashed lines in the ℓ_∞ plots). In the case of the MNIST adversarially trained networks, we also evaluate the performance of the Decision Boundary Attack (DBA) [4] with 2000 steps and PGD on standard and adversarially trained models. We observe that for ε less or equal to the value used during training, the performance is equal or better. For MNIST there is a sharp drop shortly after. Moreover, we observe that the performance of PGD on the MNIST ℓ_2 -trained networks is poor and significantly **overestimates the robustness** of the model. This is potentially due to the threshold filters learned by the model masking the loss gradients (the decision-based attack does not utilize gradients).

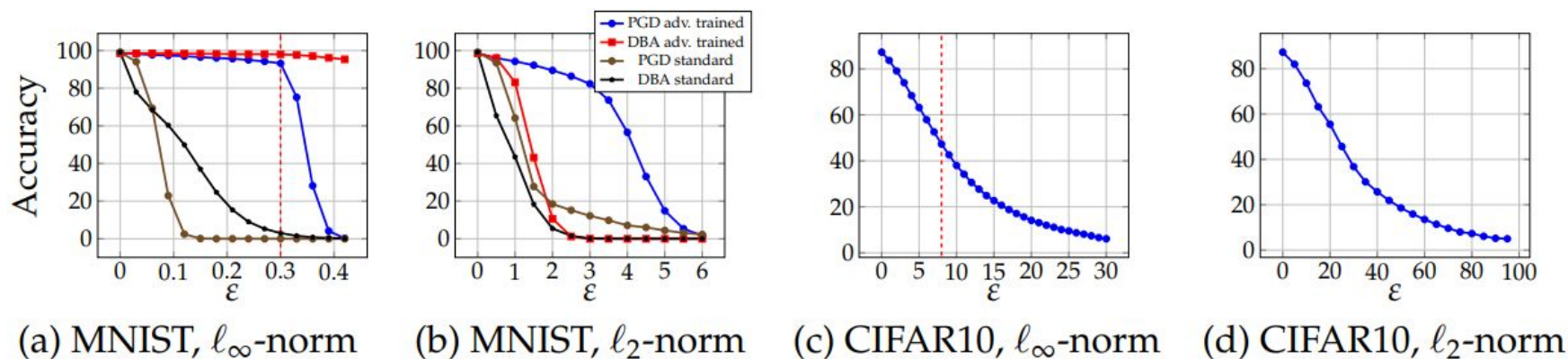


Figure 6: Performance of our adversarially trained networks against PGD adversaries of different strength. The MNIST and CIFAR10 networks were trained against $\varepsilon = 0.3$ and $\varepsilon = 8$ PGD ℓ_∞ adversaries respectively (the training ε is denoted with a red dashed lines in the ℓ_∞ plots). In the case of the MNIST adversarially trained networks, we also evaluate the performance of the Decision Boundary Attack (DBA) [4] with 2000 steps and PGD on standard and adversarially trained models. We observe that for ε less or equal to the value used during training, the performance is equal or better. For MNIST there is a sharp drop shortly after. Moreover, we observe that the performance of PGD on the MNIST ℓ_2 -trained networks is poor and significantly overestimates the robustness of the model. This is potentially due to the threshold filters learned by the model masking the loss gradients (the decision-based attack does not utilize gradients).

Conclusion

Conclusion

- Network capacity alone increase robustness against one-step adversary

Conclusion

- Network capacity alone increase robustness against one-step adversary
- FGSM adversaries don't increase robustness

Conclusion

- Network capacity alone increase robustness against one-step adversary
- FGSM adversaries don't increase robustness
- For small networks with strong adversary (PGD), network does not learn

Conclusion

- Network capacity alone increase robustness against one-step adversary
- FGSM adversaries don't increase robustness
- For small networks with strong adversary (PGD), network does not learn
- PGD adversarial model decreases value of saddle point (loss) with increase in capacity

Conclusion

- Network capacity alone increase robustness against one-step adversary
- FGSM adversaries don't increase robustness
- For small networks with strong adversary (PGD), network does not learn
- PGD adversarial model decreases value of saddle point (loss) with increase in capacity
- More capacity and stronger adversaries decrease transferability

Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples Athalye et al.

Gradient Masking

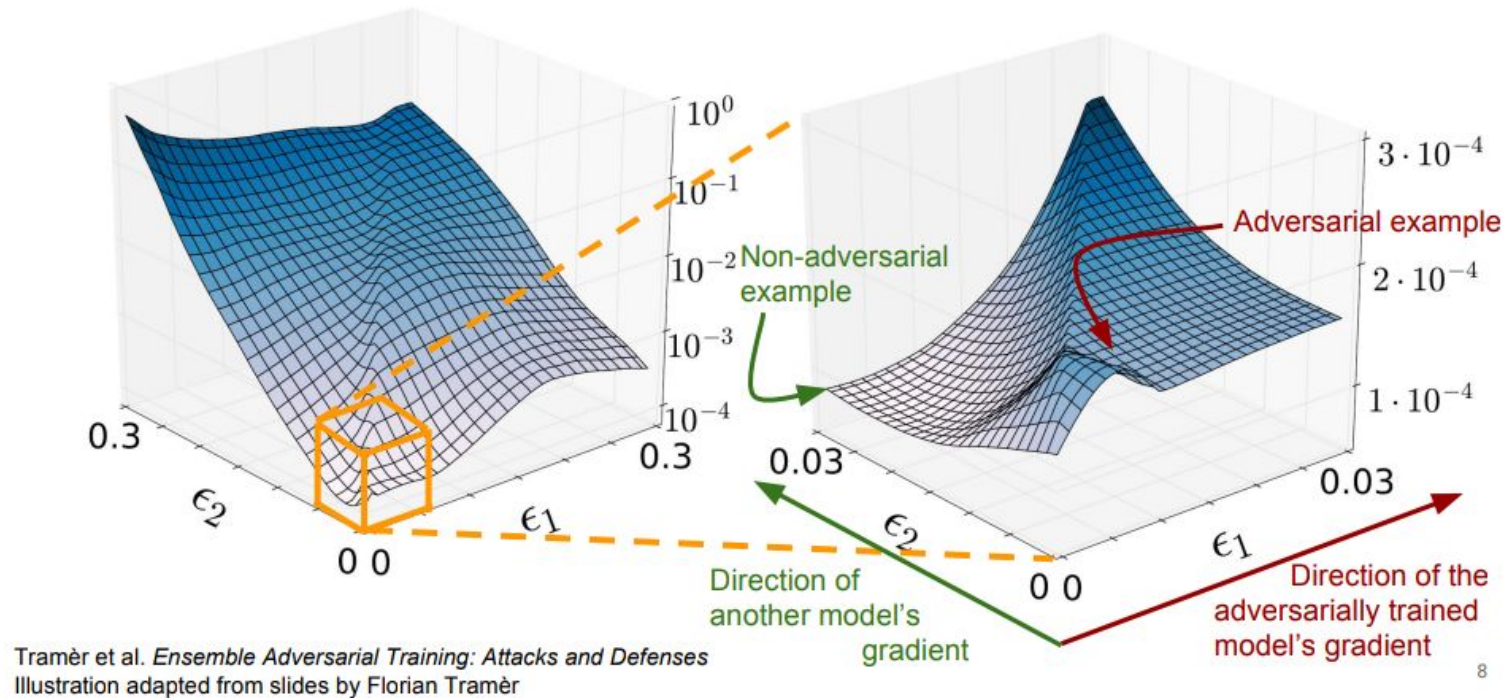
Gradient Masking

- Gradients of the loss function with respect to the model's input are intentionally or unintentionally obscured or made difficult to access.

Gradient Masking

- Gradients of the loss function with respect to the model's input are intentionally or unintentionally obscured or made difficult to access.

Gradient masking in adversarially trained models



Obfuscated Gradients

Obfuscated Gradients

- Special case of gradient masking

Obfuscated Gradients

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately

Obfuscated Gradients

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately
- Breaking the gradient or generated unintentionally

Obfuscated Gradients

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately
- Breaking the gradient or generated unintentionally
- Types of obfuscated gradients:

Obfuscated Gradients

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately
- Breaking the gradient or generated unintentionally
- Types of obfuscated gradients:
 - Shattered: Defense is non-differentiable

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately
- Breaking the gradient or generated unintentionally
- Types of obfuscated gradients:
 - Shattered: Defense is non-differentiable
 - introduces numeric instability or nonexistent/incorrect gradient.

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately
- Breaking the gradient or generated unintentionally
- Types of obfuscated gradients:
 - Shattered: Defense is non-differentiable
 - introduces numeric instability or nonexistent/incorrect gradient.
 - Stochastic: Randomized defences

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately
- Breaking the gradient or generated unintentionally
- Types of obfuscated gradients:
 - Shattered: Defense is non-differentiable
 - introduces numeric instability or nonexistent/incorrect gradient.
 - Stochastic: Randomized defences
 - either network is random or the inputs are randomly transformed causing randomized gradients

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately
- Breaking the gradient or generated unintentionally
- Types of obfuscated gradients:
 - Shattered: Defense is non-differentiable
 - introduces numeric instability or nonexistent/incorrect gradient.
 - Stochastic: Randomized defences
 - either network is random or the inputs are randomly transformed causing randomized gradients
 - Exploding and Vanishing Gradients: Defences with multiple iterations of neural network evaluation

- Special case of gradient masking
 - Make the gradients more difficult to interpret or compute accurately
- Breaking the gradient or generated unintentionally
- Types of obfuscated gradients:
 - Shattered: Defense is non-differentiable
 - introduces numeric instability or nonexistent/incorrect gradient.
 - Stochastic: Randomized defences
 - either network is random or the inputs are randomly transformed causing randomized gradients
 - Exploding and Vanishing Gradients: Defences with multiple iterations of neural network evaluation
 - Feeding output of one computation as input to next

Identifying Obfuscated & Masked Gradients

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum
- Black-box attacks are better than white-box attacks

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum
- Black-box attacks are better than white-box attacks
 - Black-box is subset of white-box

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum
- Black-box attacks are better than white-box attacks
 - Black-box is subset of white-box
- Unbounded attacks do not reach 100% success

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum
- Black-box attacks are better than white-box attacks
 - Black-box is subset of white-box
- Unbounded attacks do not reach 100% success
 - Unbounded attacks should cause 0% robustness

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum
- Black-box attacks are better than white-box attacks
 - Black-box is subset of white-box
- Unbounded attacks do not reach 100% success
 - Unbounded attacks should cause 0% robustness
- Random sampling finds adversarial examples

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum
- Black-box attacks are better than white-box attacks
 - Black-box is subset of white-box
- Unbounded attacks do not reach 100% success
 - Unbounded attacks should cause 0% robustness
- Random sampling finds adversarial examples
 - Brute forcing should not work

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum
- Black-box attacks are better than white-box attacks
 - Black-box is subset of white-box
- Unbounded attacks do not reach 100% success
 - Unbounded attacks should cause 0% robustness
- Random sampling finds adversarial examples
 - Brute forcing should not work
- Increasing distortion bound does not increase success

Identifying Obfuscated & Masked Gradients

- One-step attacks perform better than iterative attacks
 - Iterative attack stuck on local minimum
- Black-box attacks are better than white-box attacks
 - Black-box is subset of white-box
- Unbounded attacks do not reach 100% success
 - Unbounded attacks should cause 0% robustness
- Random sampling finds adversarial examples
 - Brute forcing should not work
- Increasing distortion bound does not increase success
 - Increasing the allowed perturbation range

Goals

Goals

- Address flaws in past adversarial defence techniques relying on gradient masking

Goals

- Address flaws in past adversarial defence techniques relying on gradient masking
- Describe the characteristic behaviour of defence exhibiting the effect of obfuscated gradients

Goals

- Address flaws in past adversarial defence techniques relying on gradient masking
- Describe the characteristic behaviour of defence exhibiting the effect of obfuscated gradients
- Experimental proof for successful attacks on past defences

Goals

- Address flaws in past adversarial defence techniques relying on gradient masking
- Describe the characteristic behaviour of defence exhibiting the effect of obfuscated gradients
- Experimental proof for successful attacks on past defences
- Techniques to prevent circumvention of methods relying on gradient masking

Attack Techniques

Attack Techniques

- Attack defences where gradients are not readily available

Attack Techniques

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

-

Many non-differentiable defenses can be expressed as follows: given a pre-trained classifier $f(\cdot)$, construct a preprocessor $g(\cdot)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(\cdot)$ satisfies $g(x) \approx x$ (e.g., such a $g(\cdot)$ may perform image denoising to remove the adversarial perturbation, as in [Guo et al. \(2018\)](#)). If $g(\cdot)$ is smooth and differentiable, then computing gradients through the combined network \hat{f} is often sufficient to circumvent the defense ([Carlini & Wagner, 2017b](#)). However, recent work has constructed functions $g(\cdot)$ which are neither smooth nor differentiable, and therefore can not be backpropagated through to generate adversarial examples with a white-box attack that requires gradient signal.

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

-

Many non-differentiable defenses can be expressed as follows: given a pre-trained classifier $f(\cdot)$, construct a preprocessor $g(\cdot)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(\cdot)$ satisfies $g(x) \approx x$ (e.g., such a $g(\cdot)$ may perform image denoising to remove the adversarial perturbation, as in Guo et al. (2018)). If $g(\cdot)$ is smooth and differentiable, then computing gradients through the combined network \hat{f} is often sufficient to circumvent the defense (Carlini & Wagner, 2017b). However, recent work has constructed functions $g(\cdot)$ which are neither smooth nor differentiable, and therefore can not be backpropagated through to generate adversarial examples with a white-box attack that requires gradient signal.

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

-

Many non-differentiable defenses can be expressed as follows: given a pre-trained classifier $f(\cdot)$, construct a preprocessor $g(\cdot)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(\cdot)$ satisfies $g(x) \approx x$ (e.g., such a $g(\cdot)$ may perform image denoising to remove the adversarial perturbation, as in Guo et al. (2018)). If $g(\cdot)$ is smooth and differentiable, then computing gradients through the combined network \hat{f} is often sufficient to circumvent the defense (Carlini & Wagner, 2017b). However, recent work has constructed functions $g(\cdot)$ which are neither smooth nor differentiable, and therefore can not be backpropagated through to generate adversarial examples with a white-box attack that requires gradient signal.

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

-

Many non-differentiable defenses can be expressed as follows: given a pre-trained classifier $f(\cdot)$, construct a preprocessor $g(\cdot)$ and let the secured classifier $f(x) = f(g(x))$ where the preprocessor $g(\cdot)$ satisfies $g(x) \approx x$ (e.g., such a $g(\cdot)$ may perform image denoising to remove the adversarial perturbation, as in Guo et al. (2018)). If $g(\cdot)$ is smooth and differentiable, then computing gradients through the combined network \hat{f} is often sufficient to circumvent the defense (Carlini & Wagner, 2017b). However, recent work has constructed functions $g(\cdot)$ which are neither smooth nor differentiable, and therefore can not be backpropagated through to generate adversarial examples with a white-box attack that requires gradient signal.

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

-

Many non-differentiable defenses can be expressed as follows: given a pre-trained classifier $f(\cdot)$, construct a preprocessor $g(\cdot)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(\cdot)$ satisfies $g(x) \approx x$ (e.g., such a $g(\cdot)$ may perform image denoising to remove the adversarial perturbation, as in Guo et al. (2018)). If $g(\cdot)$ is smooth and differentiable, then computing gradients through the combined network \hat{f} is often sufficient to circumvent the defense (Carlini & Wagner, 2017b). However, recent work has constructed functions $g(\cdot)$ which are neither smooth nor differentiable, and therefore can not be backpropagated through to generate adversarial examples with a white-box attack that requires gradient signal.

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

-

Many non-differentiable defenses can be expressed as follows: given a pre-trained classifier $f(\cdot)$, construct a preprocessor $g(\cdot)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(\cdot)$ satisfies $g(x) \approx x$ (e.g., such a $g(\cdot)$ may perform image denoising to remove the adversarial perturbation, as in Guo et al. (2018)). If $g(\cdot)$ is smooth and differentiable, then computing gradients through the combined network \hat{f} is often sufficient to circumvent the defense (Carlini & Wagner, 2017b). However, recent work has constructed functions $g(\cdot)$ which are neither smooth nor differentiable, and therefore can not be backpropagated through to generate adversarial examples with a white-box attack that requires gradient signal.

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

○

Because g is constructed with the property that $g(x) \approx x$, we can approximate its derivative as the derivative of the identity function: $\nabla_x g(x) \approx \nabla_x x = 1$. Therefore, we can approximate the derivative of $f(g(x))$ at the point \hat{x} as:

$$\nabla_x f(g(x))|_{x=\hat{x}} \approx \nabla_x f(x)|_{x=g(\hat{x})}$$

This allows us to compute gradients and therefore mount a white-box attack. Conceptually, this attack is simple. We perform forward propagation through the neural network as usual, but on the backward pass, we replace $g(\cdot)$ with the identity function. In practice, the implementation can be expressed in an even simpler way: we approximate $\nabla_x f(g(x))$ by evaluating $\nabla_x f(x)$ at the point $g(x)$. This gives us an

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

○

Because g is constructed with the property that $g(x) \approx x$, we can approximate its derivative as the derivative of the identity function: $\nabla_x g(x) \approx \nabla_x x = 1$. Therefore, we can approximate the derivative of $f(g(x))$ at the point \hat{x} as:

$$\nabla_x f(g(x))|_{x=\hat{x}} \approx \nabla_x f(x)|_{x=g(\hat{x})}$$

This allows us to compute gradients and therefore mount a white-box attack. Conceptually, this attack is simple. We perform forward propagation through the neural network as usual, but on the backward pass, we replace $g(\cdot)$ with the identity function. In practice, the implementation can be expressed in an even simpler way: we approximate $\nabla_x f(g(x))$ by evaluating $\nabla_x f(x)$ at the point $g(x)$. This gives us an

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

○

Because g is constructed with the property that $g(x) \approx x$, we can approximate its derivative as the derivative of the identity function: $\nabla_x g(x) \approx \nabla_x x = 1$. Therefore, we can approximate the derivative of $f(g(x))$ at the point \hat{x} as:

$$\nabla_x f(g(x))|_{x=\hat{x}} \approx \nabla_x f(x)|_{x=g(\hat{x})}$$

This allows us to compute gradients and therefore mount a white-box attack. Conceptually, this attack is simple. We perform forward propagation through the neural network as usual, but on the backward pass, we replace $g(\cdot)$ with the identity function. In practice, the implementation can be expressed in an even simpler way: we approximate $\nabla_x f(g(x))$ by evaluating $\nabla_x f(x)$ at the point $g(x)$. This gives us an

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

○

Because g is constructed with the property that $g(x) \approx x$, we can approximate its derivative as the derivative of the identity function: $\nabla_x g(x) \approx \nabla_x x = 1$. Therefore, we can approximate the derivative of $f(g(x))$ at the point \hat{x} as:

$$\nabla_x f(g(x))|_{x=\hat{x}} \approx \nabla_x f(x)|_{x=g(\hat{x})}$$

This allows us to compute gradients and therefore mount a white-box attack. Conceptually, this attack is simple. We perform forward propagation through the neural network as usual, but on the backward pass, we replace $g(\cdot)$ with the identity function. In practice, the implementation can be expressed in an even simpler way: we approximate $\nabla_x f(g(x))$ by evaluating $\nabla_x f(x)$ at the point $g(x)$. This gives us an

- Attack defences where gradients are not readily available
- Straight-Through Estimator (Special case)

-

approximation of the true gradient, and while not perfect, is sufficiently useful that when averaged over many iterations of gradient descent still generates an adversarial example.

The math behind the validity of this approach is similar to the special case.

Attack Techniques

- Attack defences where gradients are not readily available
- Backward Pass Differentiable Approximation (BPDA)

- Attack defences where gradients are not readily available
- Backward Pass Differentiable Approximation (BPDA)

Let $f(\cdot) = f^{1 \dots j}(\cdot)$ be a neural network, and let $f^i(\cdot)$ be a non-differentiable (or not usefully-differentiable) layer. To approximate $\nabla_x f(x)$, we first find a differentiable approximation $g(x)$ such that $g(x) \approx f^i(x)$. Then, we can approximate $\nabla_x f(x)$ by performing the forward pass through $f(\cdot)$ (and in particular, computing a forward pass through $f^i(x)$), but on the backward pass, replacing $f^i(x)$ with $g(x)$. Note that we perform this replacement only on the backward pass.

- Attack defences where gradients are not readily available
- Backward Pass Differentiable Approximation (BPDA)

Let $f(\cdot) = f^{1 \dots j}(\cdot)$ be a neural network, and let $f^i(\cdot)$ be a non-differentiable (or not usefully-differentiable) layer. To approximate $\nabla_x f(x)$, we first find a differentiable approximation $g(x)$ such that $g(x) \approx f^i(x)$. Then, we can approximate $\nabla_x f(x)$ by performing the forward pass through $f(\cdot)$ (and in particular, computing a forward pass through $f^i(x)$), but on the backward pass, replacing $f^i(x)$ with $g(x)$. Note that we perform this replacement only on the backward pass.

- Attack defences where gradients are not readily available
- Backward Pass Differentiable Approximation (BPDA)

Let $f(\cdot) = f^{1 \dots j}(\cdot)$ be a neural network, and let $f^i(\cdot)$ be a non-differentiable (or not usefully-differentiable) layer. To approximate $\nabla_x f(x)$, we first find a differentiable approximation $g(x)$ such that $g(x) \approx f^i(x)$. Then, we can approximate $\nabla_x f(x)$ by performing the forward pass through $f(\cdot)$ (and in particular, computing a forward pass through $f^i(x)$), but on the backward pass, replacing $f^i(x)$ with $g(x)$. Note that we perform this replacement only on the backward pass.

- Attack defences where gradients are not readily available
- Backward Pass Differentiable Approximation (BPDA)

Let $f(\cdot) = f^{1 \dots j}(\cdot)$ be a neural network, and let $f^i(\cdot)$ be a non-differentiable (or not usefully-differentiable) layer. To approximate $\nabla_x f(x)$, we first find a differentiable approximation $g(x)$ such that $g(x) \approx f^i(x)$. Then, we can approximate $\nabla_x f(x)$ by performing the forward pass through $f(\cdot)$ (and in particular, computing a forward pass through $f^i(x)$), but on the backward pass, replacing $f^i(x)$ with $g(x)$. Note that we perform this replacement only on the backward pass.

- Attack defences where gradients are not readily available
- Backward Pass Differentiable Approximation (BPDA)

As long as the two functions are similar, we find that the slightly inaccurate gradients still prove useful in constructing an adversarial example. Applying BPDA often requires more iterations of gradient descent than without because each individual gradient descent step is not exactly correct.

We have found applying BPDA is often necessary: replacing $f^i(\cdot)$ with $g(\cdot)$ on both the forward and backward pass is either completely ineffective (e.g. with Song et al. (2018)) or many times less effective (e.g. with Buckman et al. (2018)).

- Attack defences where gradients are not readily available
- Backward Pass Differentiable Approximation (BPDA)

As long as the two functions are similar, we find that the slightly inaccurate gradients still prove useful in constructing an adversarial example. Applying BPDA often requires more iterations of gradient descent than without because each individual gradient descent step is not exactly correct.

We have found applying BPDA is often necessary: replacing $f^i(\cdot)$ with $g(\cdot)$ on both the forward and backward pass is either completely ineffective (e.g. with Song et al. (2018)) or many times less effective (e.g. with Buckman et al. (2018)).

Attack Techniques

- Attack defences where gradients are not readily available
- Attacking randomized classifiers
 - Expectation over Transformation (EOT)

- Attack defences where gradients are not readily available
- Attacking randomized classifiers
 - Expectation over Transformation (EOT)

When attacking a classifier $f(\cdot)$ that first randomly transforms its input according to a function $t(\cdot)$ sampled from a distribution of transformations T , EOT optimizes the expectation over the transformation $\mathbb{E}_{t \sim T} f(t(x))$. The optimization problem can be solved by gradient descent, noting that $\nabla \mathbb{E}_{t \sim T} f(t(x)) = \mathbb{E}_{t \sim T} \nabla f(t(x))$, differentiating through the classifier and transformation, and approximating the expectation with samples at each gradient descent step.

Attack Techniques

- Attack defences where gradients are not readily available
- Solving vanishing/exploding gradients:
 - Reparameterization

- Attack defences where gradients are not readily available
- Solving vanishing/exploding gradients:
 - Reparameterization

We solve vanishing/exploding gradients by reparameterization. Assume we are given a classifier $f(g(x))$ where $g(\cdot)$ performs some optimization loop to transform the input x to a new input \hat{x} . Often times, this optimization loop means that differentiating through $g(\cdot)$, while possible, yields exploding or vanishing gradients.

To resolve this, we make a change-of-variable $x = h(z)$ for some function $h(\cdot)$ such that $g(h(z)) = h(z)$ for all z , but $h(\cdot)$ is differentiable. For example, if $g(\cdot)$ projects samples to some manifold in a specific manner, we might construct $h(z)$ to return points exclusively on the manifold. This allows us to compute gradients through $f(h(z))$ and thereby circumvent the defense.

- Attack defences where gradients are not readily available
- Solving vanishing/exploding gradients:
 - Reparameterization

We solve vanishing/exploding gradients by reparameterization. Assume we are given a classifier $f(g(x))$ where $g(\cdot)$ performs some optimization loop to transform the input x to a new input \hat{x} . Often times, this optimization loop means that differentiating through $g(\cdot)$, while possible, yields exploding or vanishing gradients.

To resolve this, we make a change-of-variable $x = h(z)$ for some function $h(\cdot)$ such that $g(h(z)) = h(z)$ for all z , but $h(\cdot)$ is differentiable. For example, if $g(\cdot)$ projects samples to some manifold in a specific manner, we might construct $h(z)$ to return points exclusively on the manifold. This allows us to compute gradients through $f(h(z))$ and thereby circumvent the defense.

- Attack defences where gradients are not readily available
- Solving vanishing/exploding gradients:
 - Reparameterization

We solve vanishing/exploding gradients by reparameterization. Assume we are given a classifier $f(g(x))$ where $g(\cdot)$ performs some optimization loop to transform the input x to a new input \hat{x} . Often times, this optimization loop means that differentiating through $g(\cdot)$, while possible, yields exploding or vanishing gradients.

To resolve this, we make a change-of-variable $x = h(z)$ for some function $h(\cdot)$ such that $g(h(z)) = h(z)$ for all z , but $h(\cdot)$ is differentiable. For example, if $g(\cdot)$ projects samples to some manifold in a specific manner, we might construct $h(z)$ to return points exclusively on the manifold. This allows us to compute gradients through $f(h(z))$ and thereby circumvent the defense.

- Attack defences where gradients are not readily available
- Solving vanishing/exploding gradients:
 - Reparameterization

We solve vanishing/exploding gradients by reparameterization. Assume we are given a classifier $f(g(x))$ where $g(\cdot)$ performs some optimization loop to transform the input x to a new input \hat{x} . Often times, this optimization loop means that differentiating through $g(\cdot)$, while possible, yields exploding or vanishing gradients.

To resolve this, we make a change-of-variable $x = h(z)$ for some function $h(\cdot)$ such that $g(h(z)) = h(z)$ for all z , but $h(\cdot)$ is differentiable. For example, if $g(\cdot)$ projects samples to some manifold in a specific manner, we might construct $h(z)$ to return points exclusively on the manifold. This allows us to compute gradients through $f(h(z))$ and thereby circumvent the defense.

Remapping!

Case Study

Case Study

- ICLR 2018 non-certified defences

Case Study

- ICLR 2018 non-certified defences
 - Testing robustness against white-box threat model

Case Study

- ICLR 2018 non-certified defences
 - Testing robustness against white-box threat model
 - 7 out of 9 depends on obfuscated gradients

- ICLR 2018 non-certified defences
 - Testing robustness against white-box threat model
 - 7 out of 9 depends on obfuscated gradients

-

There is an asymmetry in attacking defenses versus constructing robust defenses: to show a defense can be bypassed, it is only necessary to demonstrate one way to do so; in contrast, a defender must show no attack can succeed.

Defense	Dataset	Distance	Accuracy
Buckman et al. (2018)	CIFAR	0.031 (ℓ_∞)	0%*
Ma et al. (2018)	CIFAR	0.031 (ℓ_∞)	5%
Guo et al. (2018)	ImageNet	0.005 (ℓ_2)	0%*
Dhillon et al. (2018)	CIFAR	0.031 (ℓ_∞)	0%
Xie et al. (2018)	ImageNet	0.031 (ℓ_∞)	0%*
Song et al. (2018)	CIFAR	0.031 (ℓ_∞)	9%*
Samangouei et al. (2018)	MNIST	0.005 (ℓ_2)	55%**
Madry et al. (2018)	CIFAR	0.031 (ℓ_∞)	47%
Na et al. (2018)	CIFAR	0.015 (ℓ_∞)	15%

Table 1. Summary of Results: Seven of nine defense techniques accepted at ICLR 2018 cause obfuscated gradients and are vulnerable to our attacks. Defenses denoted with * propose combining adversarial training; we report here the defense alone, see §5 for full numbers. The fundamental principle behind the defense denoted with ** has 0% accuracy; in practice, imperfections cause the theoretically optimal attack to fail, see §5.4.2 for details.

Defence

- Adversarial training

- Adversarial training

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{\delta \in [-\epsilon, \epsilon]^N} \ell(x + \delta; y; F_{\theta}) \right]$$

- Adversarial training

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{\delta \in [-\epsilon, \epsilon]^N} \ell(x + \delta; y; F_{\theta}) \right]$$

Discussion. We believe this approach does not cause obfuscated gradients: our experiments with optimization-based attacks do succeed with some probability (but do not invalidate the claims in the paper). Further, the authors' evaluation of this defense performs all of the tests for characteristic behaviors of obfuscated gradients that we list. However, we note that (1) adversarial retraining has been shown to be difficult at ImageNet scale (Kurakin et al., 2016b), and (2) training exclusively on ℓ_{∞} adversarial examples provides only limited robustness to adversarial examples under other distortion metrics (Sharma & Chen, 2017).

- Adversarial training

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{\delta \in [-\epsilon, \epsilon]^N} \ell(x + \delta; y; F_{\theta}) \right]$$

Discussion. We believe this approach does not cause obfuscated gradients: our experiments with optimization-based attacks do succeed with some probability (but do not invalidate the claims in the paper). Further, the authors' evaluation of this defense performs all of the tests for characteristic behaviors of obfuscated gradients that we list. However, we note that (1) adversarial retraining has been shown to be difficult at ImageNet scale (Kurakin et al., 2016b), and (2) training exclusively on ℓ_{∞} adversarial examples provides only limited robustness to adversarial examples under other distortion metrics (Sharma & Chen, 2017).

- Adversarial training

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{\delta \in [-\epsilon, \epsilon]^N} \ell(x + \delta; y; F_{\theta}) \right]$$

Discussion. We believe this approach does not cause obfuscated gradients: our experiments with optimization-based attacks do succeed with some probability (but do not invalidate the claims in the paper). Further, the authors' evaluation of this defense performs all of the tests for characteristic behaviors of obfuscated gradients that we list. However, we note that (1) adversarial retraining has been shown to be difficult at ImageNet scale (Kurakin et al., 2016b), and (2) training exclusively on ℓ_{∞} adversarial examples provides only limited robustness to adversarial examples under other distortion metrics (Sharma & Chen, 2017).

- Adversarial training

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{\delta \in [-\epsilon, \epsilon]^N} \ell(x + \delta; y; F_{\theta}) \right]$$

- Cascade Adversarial Training

- Adversarial training

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{\delta \in [-\epsilon, \epsilon]^N} \ell(x + \delta; y; F_{\theta}) \right]$$

- Cascade Adversarial Training

Cascade adversarial machine learning (Na et al., 2018) is closely related to the above defense. The main difference is that instead of using iterative methods to generate adversarial examples at each mini-batch, the authors train a first model, generate adversarial examples (with iterative methods) on that model, add these to the training set, and then train a second model on the augmented dataset only single-step methods for efficiency. Additionally, the authors construct a “unified embedding” and enforce that the clean and adversarial logits are close under some metric.

Gradient Shattering

Gradient Shattering

- Thermometer encoding

Gradient Shattering

- **Thermometer encoding**
 - Breaking the linearity which causes adversarial examples to exist

- Thermometer encoding
 - Breaking the linearity which causes adversarial examples to exist

Given an image x , for each pixel color $x_{i,j,c}$, the l -level *thermometer encoding* $\tau(x_{i,j,c})$ is a l -dimensional vector where $\tau(x_{i,j,c})_k = 1$ if $x_{i,j,c} > k/l$, and 0 otherwise (e.g., for a 10-level thermometer encoding, $\tau(0.66) = 1111110000$).

- **Thermometer encoding**

- Breaking the linearity which causes adversarial examples to exist

Given an image x , for each pixel color $x_{i,j,c}$, the l -level *thermometer encoding* $\tau(x_{i,j,c})$ is a l -dimensional vector where $\tau(x_{i,j,c})_k = 1$ if $x_{i,j,c} > k/l$, and 0 otherwise (e.g., for a 10-level thermometer encoding, $\tau(0.66) = 1111110000$).

- Logit-Space Projected Gradient Ascent (LS-PGA) attack on discrete space

- Thermometer encoding

- Breaking the linearity which causes adversarial examples to exist

Given an image x , for each pixel color $x_{i,j,c}$, the l -level *thermometer encoding* $\tau(x_{i,j,c})$ is a l -dimensional vector where $\tau(x_{i,j,c})_k = 1$ if $x_{i,j,c} > k/l$, and 0 otherwise (e.g., for a 10-level thermometer encoding, $\tau(0.66) = 1111110000$).

- Logit-Space Projected Gradient Ascent (LS-PGA) attack on discrete space
- ! Black box performance is worse than white box

Gradient Shattering

- Attacked using BPDA

Gradient Shattering

- Attacked using BPDA

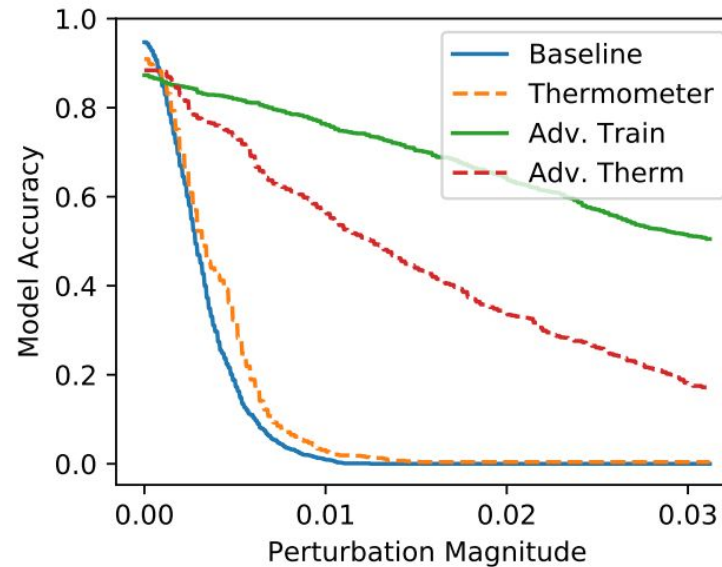


Figure 1. Model accuracy versus distortion (under ℓ_∞). Adversarial training increases robustness to 50% at $\epsilon = 0.031$; thermometer encoding by itself provides limited value, and when coupled with adversarial training performs worse than adversarial training alone.

Gradient Shattering

- Input Transformation

- **Input Transformation**
 - Image cropping, rescaling, bit-depth reduction, JPEG compression, total variance minimization and image quilting

- **Input Transformation**

- Image cropping, rescaling, bit-depth reduction, JPEG compression, total variance minimization and image quilting
- Black box, ResNet-50, l_2 dissimilarity, 60% top-1 defence accuracy

- **Input Transformation**
 - Image cropping, rescaling, bit-depth reduction, JPEG compression, total variance minimization and image quilting
 - Black box, ResNet-50, l_2 dissimilarity, 60% top-1 defence accuracy
- Bypass!

- **Input Transformation**
 - Image cropping, rescaling, bit-depth reduction, JPEG compression, total variance minimization and image quilting
 - Black box, ResNet-50, l_2 dissimilarity, 60% top-1 defence accuracy
- **Bypass!**
 - EOT: Image cropping and scaling

- **Input Transformation**
 - Image cropping, rescaling, bit-depth reduction, JPEG compression, total variance minimization and image quilting
 - Black box, ResNet-50, l_2 dissimilarity, 60% top-1 defence accuracy
- **Bypass!**
 - EOT: Image cropping and scaling
 - BPDA: Bit-depth reduction, JPEG compression

- **Input Transformation**
 - Image cropping, rescaling, bit-depth reduction, JPEG compression, total variance minimization and image quilting
 - Black box, ResNet-50, l_2 dissimilarity, 60% top-1 defence accuracy
- **Bypass!**
 - EOT: Image cropping and scaling
 - BPDA: Bit-depth reduction, JPEG compression
 - EOT + BPDA: total variance minimization and image quilting

- **Input Transformation**

- Image cropping, rescaling, bit-depth reduction, JPEG compression, total variance minimization and image quilting
- Black box, ResNet-50, l_2 dissimilarity, 60% top-1 defence accuracy

- **Bypass!**

- EOT: Image cropping and scaling
- BPDA: Bit-depth reduction, JPEG compression
- EOT + BPDA: total variance minimization and image quilting
 - Accuracy drops to 0% under strongest defence with small perturbation budget

Stochastic Gradients

Stochastic Gradients

- Stochastic Activation Pruning (SAP)

Stochastic Gradients

- **Stochastic Activation Pruning (SAP)**
 - Dropout with weighted distribution

- **Stochastic Activation Pruning (SAP)**
 - Dropout with weighted distribution
 - ! Used single-step in the gradient direction evaluation

- **Stochastic Activation Pruning (SAP)**

- Dropout with weighted distribution
- ! Used single-step in the gradient direction evaluation

- Attack:

randomness. At each iteration of gradient descent, instead of taking a step in the direction of $\nabla_x f(x)$ we move in the direction of $\sum_{i=1}^k \nabla_x f(x)$ where each invocation is randomized with SAP. We have found that choosing $k = 10$ provides useful gradients. We additionally had to resolve

- **Stochastic Activation Pruning (SAP)**

- Dropout with weighted distribution
- ! Used single-step in the gradient direction evaluation

- Attack: randomness. At each iteration of gradient descent, instead of taking a step in the direction of $\nabla_x f(x)$ we move in the direction of $\sum_{i=1}^k \nabla_x f(x)$ where each invocation is randomized with SAP. We have found that choosing $k = 10$ provides useful gradients. We additionally had to resolve

- Accuracy drop to 9% with $\epsilon = .015$ and 0% at $\epsilon = 0.031$

Stochastic Gradients

- Mitigating Through Randomization

- **Mitigating Through Randomization**
 - Adding randomization layer before input to classifier

- **Mitigating Through Randomization**
 - Adding randomization layer before input to classifier
 - Attack on original, fixed randomization and ensemble

- **Mitigating Through Randomization**
 - Adding randomization layer before input to classifier
 - Attack on original, fixed randomization and ensemble
 - ! Claims that stronger attack would be computationally expensive

- **Mitigating Through Randomization**
 - Adding randomization layer before input to classifier
 - Attack on original, fixed randomization and ensemble
 - ! Claims that stronger attack would be computationally expensive
- Attack:

- **Mitigating Through Randomization**
 - Adding randomization layer before input to classifier
 - Attack on original, fixed randomization and ensemble
 - ! Claims that stronger attack would be computationally expensive
- **Attack:**
 - EOT optimizing over distribution of transformations

- **Mitigating Through Randomization**
 - Adding randomization layer before input to classifier
 - Attack on original, fixed randomization and ensemble
 - ! Claims that stronger attack would be computationally expensive
- Attack:
 - EOT optimizing over distribution of transformations
- Accuracy drop to 32.8% with $\epsilon = .0031$ under l -infinity norm

Vanishing & Exploding Gradients

Vanishing & Exploding Gradients

- PixelDefend

Vanishing & Exploding Gradients

- **PixelDefend**
 - PixelCNN to project potential adversarial example back to data manifold before input

Vanishing & Exploding Gradients

- **PixelDefend**
 - PixelCNN to project potential adversarial example back to data manifold before input
 - Argue that adversarial examples lie in low-probability region

Vanishing & Exploding Gradients

- **PixelDefend**

- PixelCNN to project potential adversarial example back to data manifold before input
- Argue that adversarial examples lie in low-probability region
 - PixelDefend “purifies” image by finding highest probability example in ϵ -ball of input

- **PixelDefend**

- PixelCNN to project potential adversarial example back to data manifold before input
- Argue that adversarial examples lie in low-probability region
 - PixelDefend “purifies” image by finding highest probability example in ϵ -ball of input
- With a maximum l_∞ perturbation of $\epsilon = 0.031$, PixelDefend claims 46% accuracy (with a vanilla ResNet classifier)

- **PixelDefend**

- PixelCNN to project potential adversarial example back to data manifold before input
- Argue that adversarial examples lie in low-probability region
 - PixelDefend “purifies” image by finding highest probability example in ϵ -ball of input
- With a maximum l_∞ perturbation of $\epsilon = 0.031$, PixelDefend claims 46% accuracy (with a vanilla ResNet classifier)
- ! Dismiss attacks with difficult differentiation due to vanishing gradients and computation cost

- **PixelDefend**
 - PixelCNN to project potential adversarial example back to data manifold before input
 - Argue that adversarial examples lie in low-probability region
 - PixelDefend “purifies” image by finding highest probability example in ϵ -ball of input
 - With a maximum l_∞ perturbation of $\epsilon = 0.031$, PixelDefend claims 46% accuracy (with a vanilla ResNet classifier)
 - ! Dismiss attacks with difficult differentiation due to vanishing gradients and computation cost
- Attack: BPDA to approximate gradient

- **PixelDefend**
 - PixelCNN to project potential adversarial example back to data manifold before input
 - Argue that adversarial examples lie in low-probability region
 - PixelDefend “purifies” image by finding highest probability example in ϵ -ball of input
 - With a maximum l_∞ perturbation of $\epsilon = 0.031$, PixelDefend claims 46% accuracy (with a vanilla ResNet classifier)
 - ! Dismiss attacks with difficult differentiation due to vanishing gradients and computation cost
- Attack: BPDA to approximate gradient
- Reduce accuracy to 9%

Vanishing & Exploding Gradients

- **Defence-GAN**
 - Similar to PixelDefend; use of GANs
- Attack with BPDA with 45% success rate

Outcomes? Inference?

Guidelines

→ For building and evaluating defences

Guidelines

1. Define a (realistic) threat model

Guidelines

1. Define a (realistic) threat model
 - a. Model architecture and model weights

Guidelines

1. Define a (realistic) threat model
 - a. Model architecture and model weights
 - b. Training algorithm and training data

1. Define a (realistic) threat model
 - a. Model architecture and model weights
 - b. Training algorithm and training data
 - c. Test time randomness (chosen values or distribution)

1. Define a (realistic) threat model
 - a. Model architecture and model weights
 - b. Training algorithm and training data
 - c. Test time randomness (chosen values or distribution)
 - d. Query access (logits or top label)

1. Define a (realistic) threat model
 - a. Model architecture and model weights
 - b. Training algorithm and training data
 - c. Test time randomness (chosen values or distribution)
 - d. Query access (logits or top label)
- Should not have unrealistic constraints

2. Make specific, testable claims

2. Make specific, testable claims
 - a. Accuracy, bound, budget, threat model

2. Make specific, testable claims
 - a. Accuracy, bound, budget, threat model
 - b. State if model evaluated under different threat model

2. Make specific, testable claims
 - a. Accuracy, bound, budget, threat model
 - b. State if model evaluated under different threat model
 - c. Code release

2. Make specific, testable claims
 - a. Accuracy, bound, budget, threat model
 - b. State if model evaluated under different threat model
 - c. Code release
3. Evaluate against adaptive attacks

2. Make specific, testable claims
 - a. Accuracy, bound, budget, threat model
 - b. State if model evaluated under different threat model
 - c. Code release
3. Evaluate against adaptive attacks
 - a. Future attacks

2. Make specific, testable claims
 - a. Accuracy, bound, budget, threat model
 - b. State if model evaluated under different threat model
 - c. Code release
3. Evaluate against adaptive attacks
 - a. Future attacks
 - b. After defence is specified, adversary attacks again with only restriction of threat model

2. Make specific, testable claims
 - a. Accuracy, bound, budget, threat model
 - b. State if model evaluated under different threat model
 - c. Code release
3. Evaluate against adaptive attacks
 - a. Future attacks
 - b. After defence is specified, adversary attacks again with only restriction of threat model
 - c. Multiple attacks evaluation; mean over best attack per image

Conclusion

Conclusion

- Highlight limitations of gradient obfuscation as a standalone mechanism

Conclusion

- Highlight limitations of gradient obfuscation as a standalone mechanism
- Adaptive attacks can break defence mechanisms

Conclusion

- Highlight limitations of gradient obfuscation as a standalone mechanism
- Adaptive attacks can break defence mechanisms
- Comprehensive defence strategies

- Highlight limitations of gradient obfuscation as a standalone mechanism
- Adaptive attacks can break defence mechanisms
- Comprehensive defence strategies
 - Adversarial defence, ensemble methods or input preprocessing

- Highlight limitations of gradient obfuscation as a standalone mechanism
- Adaptive attacks can break defence mechanisms
- Comprehensive defence strategies
 - Adversarial defence, ensemble methods or input preprocessing
- Evaluated different attacks

- Highlight limitations of gradient obfuscation as a standalone mechanism
- Adaptive attacks can break defence mechanisms
- Comprehensive defence strategies
 - Adversarial defence, ensemble methods or input preprocessing
- Evaluated different attacks
 - Strongest first order attacks have lower success to adaptive attacks

- Highlight limitations of gradient obfuscation as a standalone mechanism
- Adaptive attacks can break defence mechanisms
- Comprehensive defence strategies
 - Adversarial defence, ensemble methods or input preprocessing
- Evaluated different attacks
 - Strongest first order attacks have lower success to adaptive attacks
- Guidelines to improve defence mechanisms

Limitations

Limitations

- Experimental scope is limited

Limitations

- Experimental scope is limited
- Dependency on data, model and task

Limitations

- Experimental scope is limited
- Dependency on data, model and task
- Attack complexity can vary depending on task complexity

- Experimental scope is limited
- Dependency on data, model and task
- Attack complexity can vary depending on task complexity
- Combination and comparison with other attacks is not mentioned

- Experimental scope is limited
- Dependency on data, model and task
- Attack complexity can vary depending on task complexity
- Combination and comparison with other attacks is not mentioned
- Extensive insights to alternate defences / mitigation strategies not provided

Future Work

1. "Stealthy Backdoors as Compression Artifacts" by Liu et al. (2021)
 - Explores the use of obfuscated gradients as a way to hide backdoor triggers in deep neural networks
 - Demonstrates how attackers can leverage obfuscated gradients to create stealthy backdoors in models
2. "The Limitations of Model-Agnostic Attacks" by Grosse et al. (2019)
 - Investigates the limitations of obfuscated gradients and other model-agnostic attacks
 - Provides insights into the challenges of crafting effective adversarial examples in scenarios where the attacker has limited knowledge of the target model
3. "On the Robustness of Machine Learning Models to Universal Adversarial Perturbations" by Moosavi-Dezfooli et al. (2020)
 - Explores the vulnerability of machine learning models, including those protected by gradient obfuscation, to universal adversarial perturbations.
 - Investigates the robustness of models against perturbations that are imperceptible to human perception but can cause misclassification

References

1. 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.
2. Athalye, A., Carlini, N. and Wagner, D., 2018, July. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In International conference on machine learning (pp. 274–283). PMLR.
3. <https://seclab.stanford.edu/AdvML2017/slides/17-09-aro-aml.pdf>
4. Gibbon:
<https://www.istockphoto.com/de/foto/wei%C3%9Fe-hand-gibbon-blickkontakt-gm482658895-37248302>
5. Panda:
<https://www.telegraph.co.uk/news/2016/09/15/pandas-arent-cute-theyre-death-loving-oxygen-thieves-lets-just-e/>
6. Stop adversarial example: <https://substance.etsmtl.ca/en/brittleness-of-deep-learning-models>
7. Bias-variance tradeoff: <https://data-science-blog.com/blog/2020/11/02/bias-and-variance-in-machine-learning/>



**UNIVERSITÄT
DES
SAARLANDES**

Thanks!
Questions?

Images credit: Madry et al. & Athalye et al.
Slides credit: Universität des Saarlandes

APPENDIX

- Adversarial Training

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{\delta \in [-\epsilon, \epsilon]^N} \ell(x + \delta; y; F_{\theta}) \right]$$

- Cascade Adversarial Training
 - Train first model → generate adversarial examples (iterative method) on the model → add to train set → train second model on augmented dataset (using single step method)