Adversarial Examples

A. M. Sadeghzadeh, Ph.D.

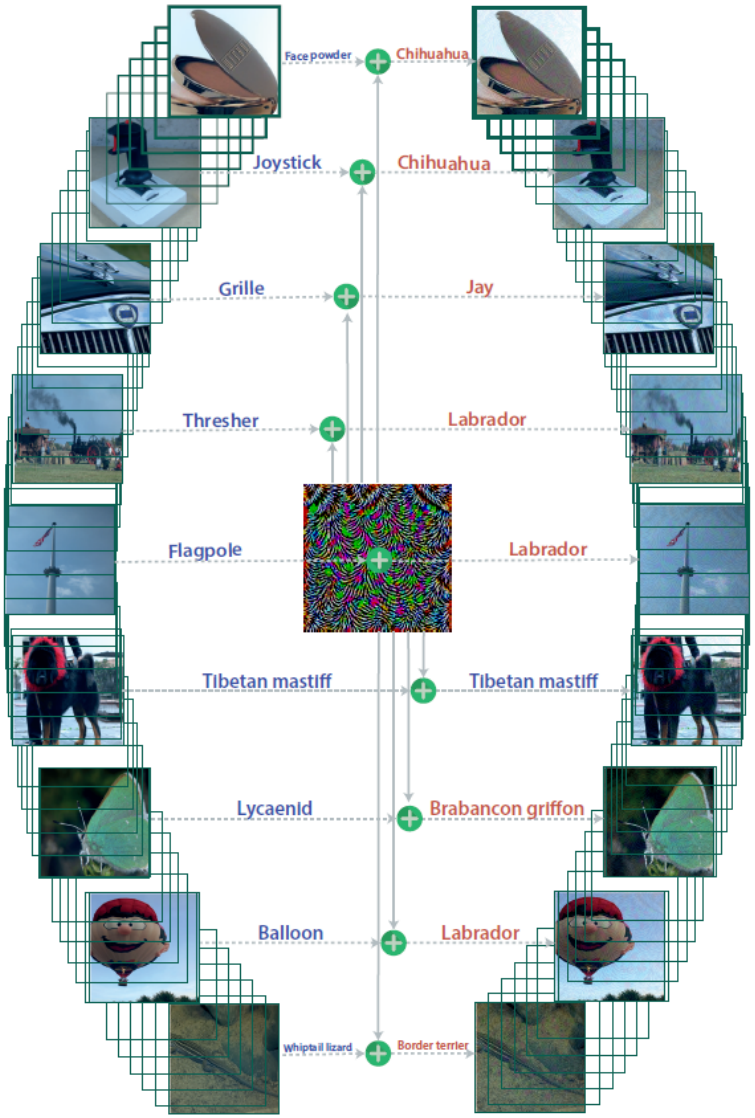
Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

April 18, 2023

Today's Agenda

Recap

Universal (Image-agnostic) Perturbation



UAP Algorithm

Algorithm 1 Computation of universal perturbations.

input: Data points X, classifier $\hat{k}$, desired ℓ_p norm of the perturbation ξ, desired accuracy on perturbed samples δ.
output: Universal perturbation vector v.
Initialize $v \leftarrow 0$.
while $\text{Err}(X_v) \leq 1 - \delta$ **do**
    for each datapoint $x_i \in X$ **do**
        if $\hat{k}(x_i + v) = \hat{k}(x_i)$ **then**
            Compute the *minimal* perturbation that sends $x_i + v$ to the decision boundary:

$$\Delta v_i \leftarrow \arg\min_r \|r\|_2 \text{ s.t. } \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

            Update the perturbation:

$$v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i).$$

        end if
    end for
end while

Adversarial Patch

Adversarial Patch

Adversarial Patch

Tom B. Brown, Dandelion Mané,* Aurko Roy, Martín Abadi, Justin Gilmer
{tombrown,dandelion,aurkor,abadi,gilmer}@google.com

Abstract

- The paper presents a method to create **universal, robust, targeted adversarial image patches** in the real world.
  - Universal because they can be used to attack any scene
  - Robust because they work under a wide variety of transformations
  - Targeted because they can cause a classifier to output any target class.

- This work explores what is possible if an attacker is **no longer restricted to small or imperceptible perturbation**.

Adversarial Patch

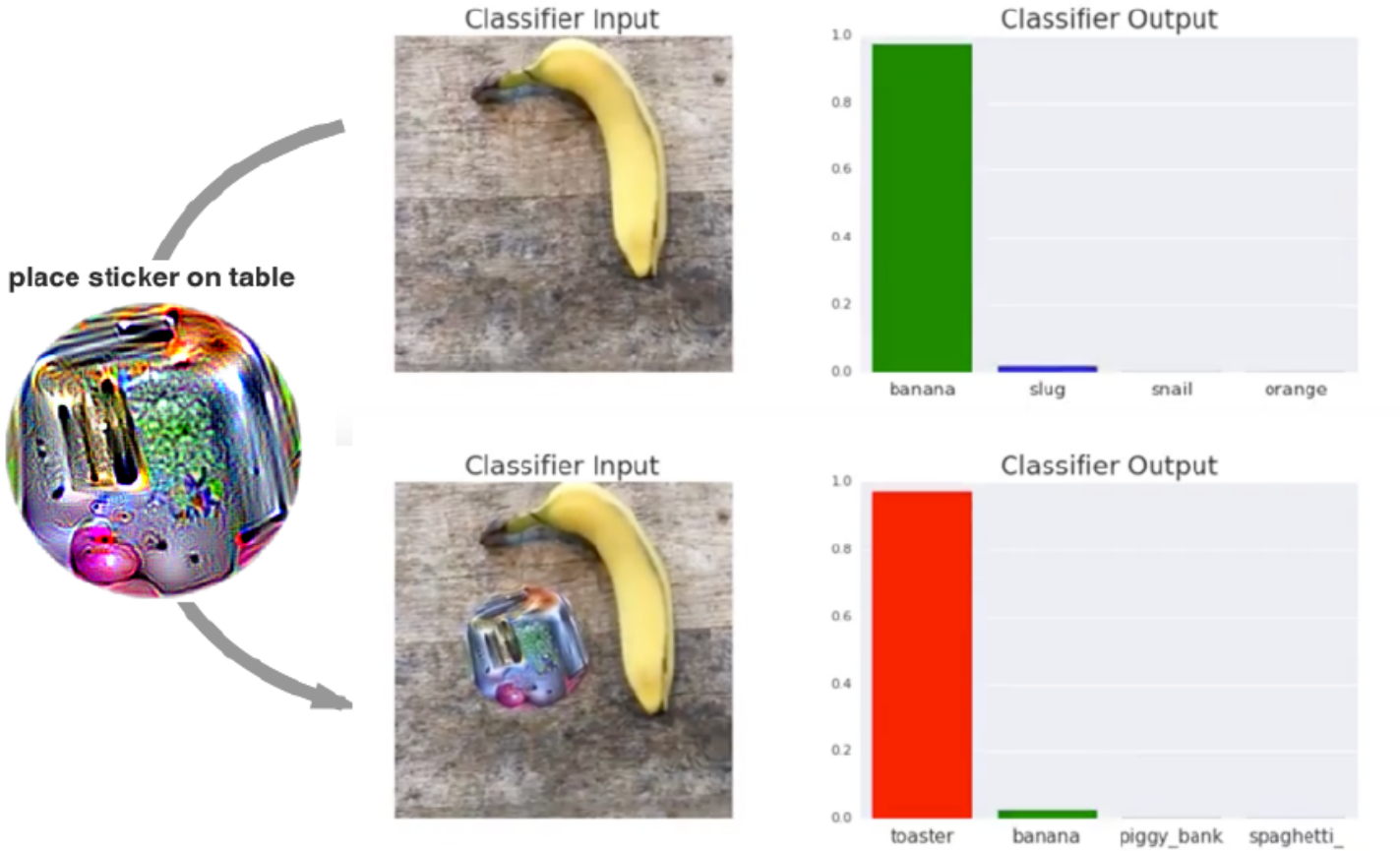


Figure 1: A real-world attack on VGG16, using a physical patch generated by the white-box ensemble method described in Section 3. When a photo of a tabletop with a banana and a notebook (top photograph) is passed through VGG16, the network reports class 'banana' with 97% confidence (top plot). If we physically place a sticker targeted to the class "toaster" on the table (bottom photograph), the photograph is classified as a toaster with 99% confidence (bottom plot). See the following video for a full demonstration: `https://youtu.be/i1sp4X57TL4`

The Approach

- The attack completely **replaces a part of the image with a patch**.
  - We mask the patch to allow it to take any shape, and then train over a variety of images, applying a random **translation, scaling, and rotation** on the patch in each image, optimizing using gradient descent.

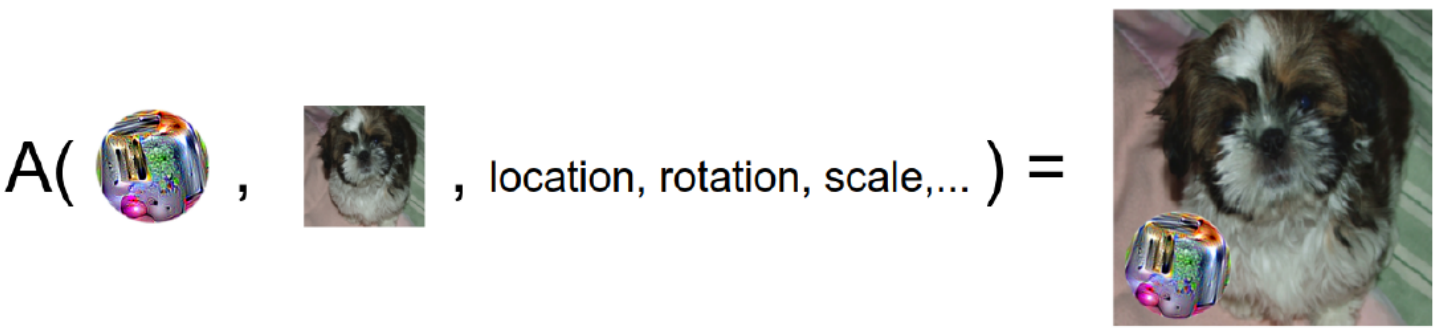
The Approach

- The attack completely **replaces a part of the image with a patch**.
  - We mask the patch to allow it to take any shape, and then train over a variety of images, applying a random **translation, scaling, and rotation** on the patch in each image, optimizing using gradient descent.
- The patch is trained to optimize the objective function

$$\hat{p} = \underset{p}{argmax}\ \mathbb{E}_{x\sim X, t\sim T, l\sim L}[logPr(\hat{y}|A(p,x,l,t))]$$

where X is a training set of images, T is a distribution over transformations of the patch, L is a distribution over locations in the image, and $\hat{y}$ is the target class.

- Note that this expectation is over images, which encourages the trained patch to work regardless of what is in the background (perturbation is universal).



Towards Deep Learning Models Resistant to Adversarial

Towards Deep Learning Models Resistant to Adversarial

Towards Deep Learning Models Resistant to Adversarial Attacks

Aleksander Mądry*
MIT
madry@mit.edu

Aleksandar Makelov*
MIT
amakelov@mit.edu

Ludwig Schmidt*
MIT
ludwigs@mit.edu

Dimitris Tsipras*
MIT
tsipras@mit.edu

Adrian Vladu*
MIT
avladu@mit.edu

Abstract

- We study the adversarial robustness of neural networks through the lens of **robust optimization**
- We use a natural **saddle point (min-max) formulation** to capture the notion of security against adversarial attacks.
- We explore the impact of network architecture on adversarial robustness and find that **model capacity plays an important role** here.

An Optimization View on Adversarial Robustness

Empirical risk minimization (ERM) has been tremendously successful as a recipe for finding classifiers with small population risk.

- The goal of standard training is to find model parameters θ that minimize the risk func. L

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

where data distribution $\mathcal{D}$ is over pairs of examples $x \in \mathbb{R}^d$ and corresponding labels $y \in [K]$ (K is the number of classes).

An Optimization View on Adversarial Robustness

Empirical risk minimization (ERM) has been tremendously successful as a recipe for finding classifiers with small population risk.

- The goal of standard training is to find model parameters θ that minimize the risk func. L

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

  where data distribution $\mathcal{D}$ is over pairs of examples $x \in \mathbb{R}^d$ and corresponding labels $y \in [K]$ (K is the number of classes).

- Unfortunately, **ERM often does not yield models that are robust** to adversarially crafted examples.

An Optimization View on Adversarial Robustness

Empirical risk minimization (ERM) has been tremendously successful as a recipe for finding classifiers with small population risk.

- The goal of standard training is to find model parameters θ that minimize the risk func. L

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

  where data distribution $\mathcal{D}$ is over pairs of examples $x \in \mathbb{R}^d$ and corresponding labels $y \in [K]$ (K is the number of classes).
- Unfortunately, **ERM often does not yield models that are robust** to adversarially crafted examples.

In order to reliably train models that are robust to adversarial attacks, it is necessary to **augment the ERM paradigm** appropriately by changing the definition of population risk $\mathbb{E}_{\mathcal{D}}[L]$

An Optimization View on Adversarial Robustness

Empirical risk minimization (ERM) has been tremendously successful as a recipe for finding classifiers with small population risk.

- The goal of standard training is to find model parameters θ that minimize the risk func. L

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

  where data distribution $\mathcal{D}$ is over pairs of examples $x \in \mathbb{R}^d$ and corresponding labels $y \in [K]$ (K is the number of classes).

- Unfortunately, **ERM often does not yield models that are robust** to adversarially crafted examples.

In order to reliably train models that are robust to adversarial attacks, it is necessary to **augment the ERM paradigm** appropriately by changing the definition of population risk $\mathbb{E}_{\mathcal{D}}[L]$

- Instead of feeding samples from the distribution $\mathcal{D}$ directly into the loss L, we allow the adversary to **perturb the input first**. This gives rise to the following saddle point problem

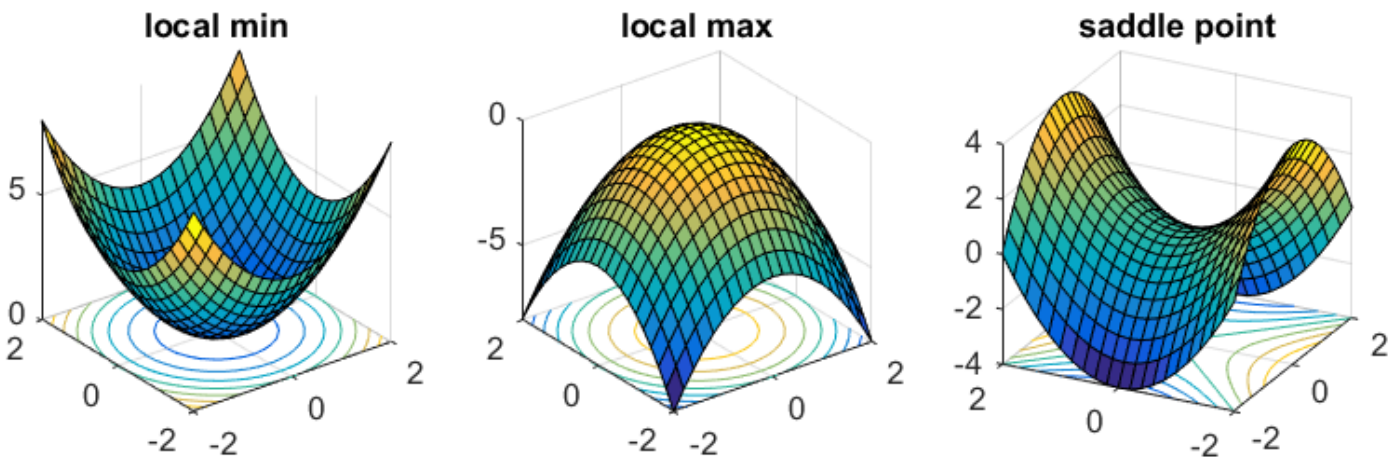
$$\min_{\theta} \mathbb{E}_{(x,y)\in\mathcal{D}} \left[\max_{\delta\in\mathcal{S}} L(\theta, x + \delta, y)\right]$$

  where $\mathcal{S} \subseteq \mathbb{R}^d$ is allowed perturbations that formalizes the manipulative power of the adversary (e.g., L_∞-ball).

Adversarial Loss

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

Our perspective stems from viewing the **saddle point** problem as the composition of an **inner maximization** problem and an **outer minimization** problem.



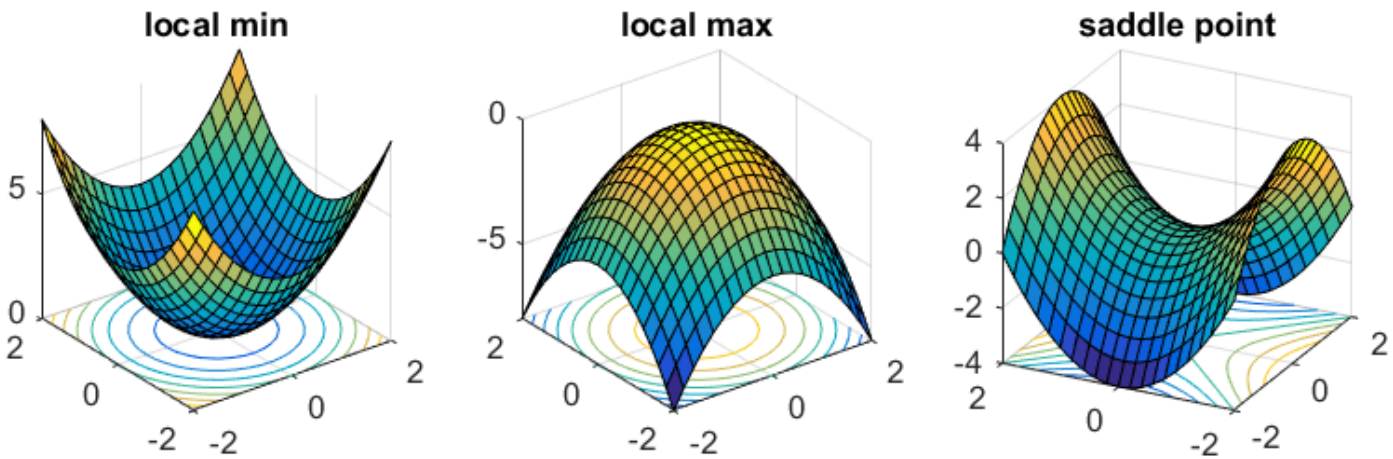
Source

Adversarial Loss

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

Our perspective stems from viewing the **saddle point** problem as the composition of an **inner maximization** problem and an **outer minimization** problem.

- The **inner maximization** problem aims to **find an adversarial version of a given data** point x that achieves a high loss.



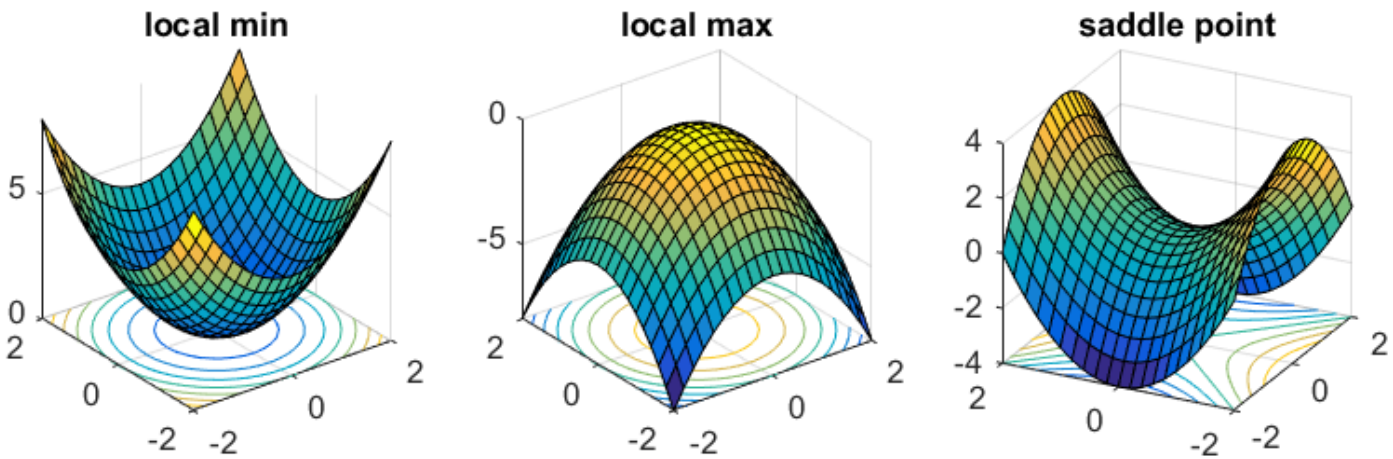
Source

Adversarial Loss

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

Our perspective stems from viewing the **saddle point** problem as the composition of an **inner maximization** problem and an **outer minimization** problem.

- The **inner maximization** problem aims to **find an adversarial version of a given data** point x that achieves a high loss.
- The goal of the **outer minimization** problem is to **find model parameters** so that the **adversarial loss** given by the inner attack problem is **minimized**.



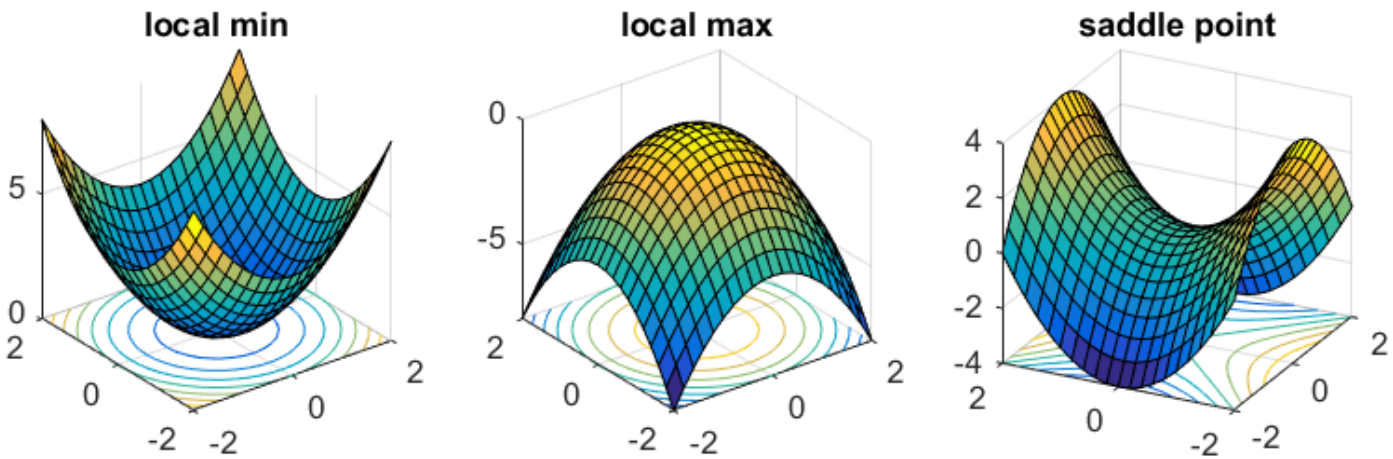
Source

Adversarial Loss

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

Our perspective stems from viewing the **saddle point** problem as the composition of an **inner maximization** problem and an **outer minimization** problem.

- The **inner maximization** problem aims to **find an adversarial version of a given data** point x that achieves a high loss.
- The goal of the **outer minimization** problem is to **find model parameters** so that the **adversarial loss** given by the inner attack problem is **minimized**.
- When the parameters θ yield a (nearly) **vanishing risk**, the corresponding model is perfectly **robust to attacks** specified by our attack model.



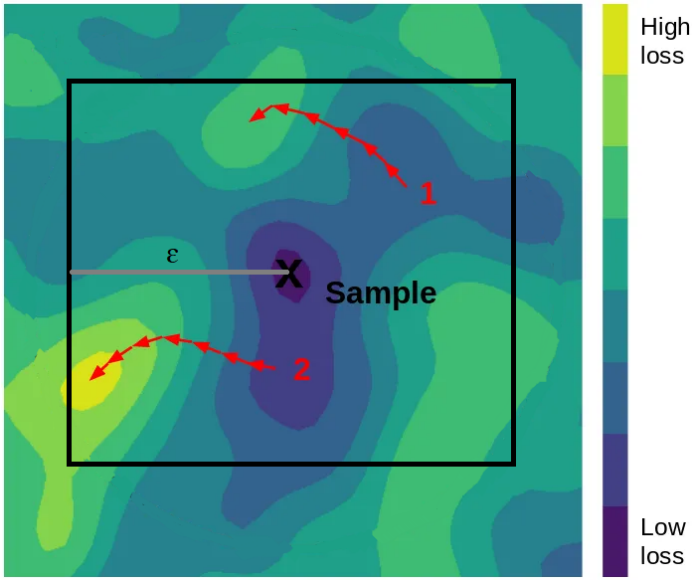
Source

Projected Gradient Descent (PGD) Attack - L_∞

FGSM is a simple one-step scheme for maximizing the inner part of the saddle point formulation. A more powerful adversary is the **multi-step variant**, which is essentially projected gradient descent (PGD) on the negative loss function

$$x^0 = Clip_{[0,1]}\{x + U(-\epsilon, \epsilon)\},$$
$$\delta^{t+1} = \alpha.sign(\nabla_x L(\theta, x^t, y)),$$
$$x^{t+1} = Clip_{[max(0,x-\epsilon),min(1,x+\epsilon)]}\{x^t + \delta^{t+1}\}.$$

where x is a natural data, U is uniform distribution, $Clip_{[a,b]}\{x\}$ function is used to trim values outside interval $[a, b]$ to the interval edges, ϵ is the radius of allowed perturbation $\|.\|_\infty \leq \epsilon$, and t is iteration index.



Source

PGD-L_∞ Source Code

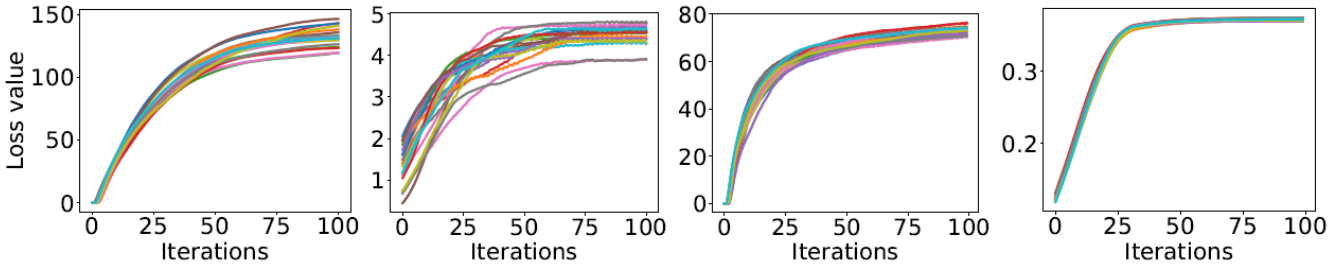
```
def perturb(self, x_nat, y, sess):
  """Given a set of examples (x_nat, y), returns a set of adversarial
     examples within epsilon of x_nat in l_infinity norm."""
  if self.rand:
    x = x_nat + np.random.uniform(-self.epsilon, self.epsilon, x_nat.shape)
    x = np.clip(x, 0, 1) # ensure valid pixel range
  else:
    x = np.copy(x_nat)

  for i in range(self.k):
    grad = sess.run(self.grad, feed_dict={self.model.x_input: x,
                                          self.model.y_input: y})

    x += self.a * np.sign(grad)

    x = np.clip(x, x_nat - self.epsilon, x_nat + self.epsilon)
    x = np.clip(x, 0, 1) # ensure valid pixel range

  return x
```

Source

The Landscape of Adversarial Examples

While there are many local maxima spread widely apart within $x_i + \mathcal{S}$, they tend to have **very well-concentrated loss values**.

- This echoes the folklore belief that training neural networks is possible because the loss (as a function of model parameters) typically has many local minima with very similar values.



(a) MNIST Standard training (b) MNIST Adversarial training (c) CIFAR10 Natural training (d) CIFAR10 Adversarial training

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.

First-Order Adversaries

The concentration phenomenon suggests an **intriguing view** on the problem in which robustness against the PGD adversary yields robustness against all **first-order adversaries**

- As long as the adversary only uses gradients of the loss function with respect to the input, **we conjecture** that it **will not find significantly better local maxima than PGD**.
- Of course, our exploration with PGD does not preclude the existence of **some isolated maxima** with much larger function value.
- However, our experiments suggest that such better local maxima are **hard to find** with first order methods.

Network Capacity and Adversarial Robustness

Classifying examples in a robust way requires a stronger classifier, since the presence of **adversarial examples changes the decision boundary of the problem to a more complicated one**.

- Our experiments verify that capacity is crucial for robustness, as well as for the ability to successfully train against strong adversaries.

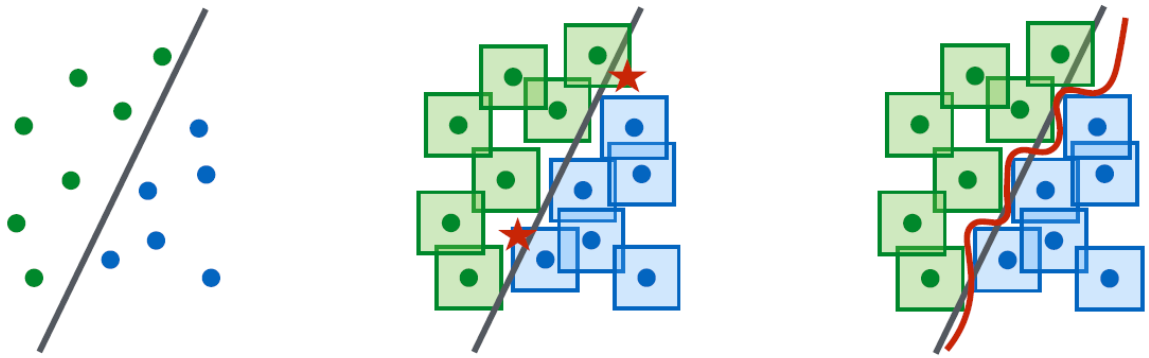
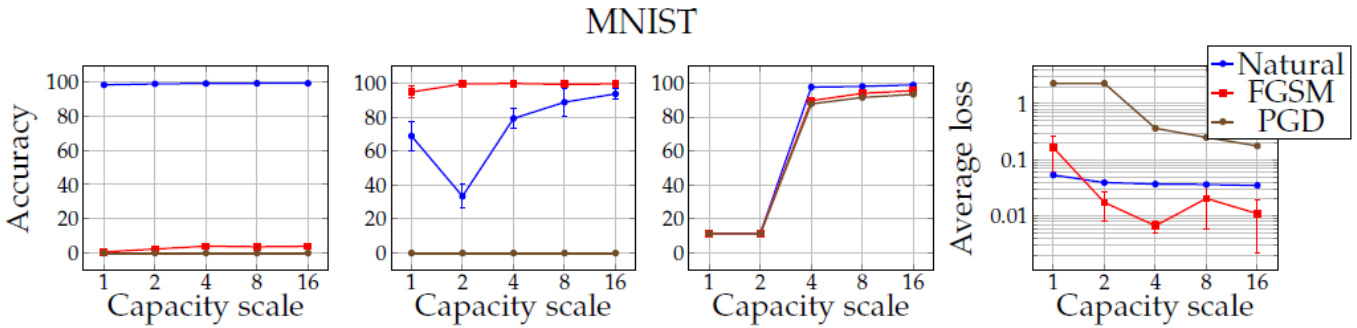


Figure 3: A conceptual illustration of standard vs. adversarial decision boundaries. Left: A set of points that can be easily separated with a simple (in this case, linear) decision boundary. Middle: The simple decision boundary does not separate the ℓ_∞-balls (here, squares) around the data points. Hence there are adversarial examples (the red stars) that will be misclassified. Right: Separating the ℓ_∞-balls requires a significantly more complicated decision boundary. The resulting classifier is robust to adversarial examples with bounded ℓ_∞-norm perturbations.

Network Capacity and Adversarial Robustness

Either **increasing the capacity** of the network, or using a **stronger method for the inner optimization** problem reduces the effectiveness of adversarial inputs (in other words, **increase the robustness of model**).



CIFAR10

	Simple	Wide
Natural	92.7%	95.2%
FGSM	27.5%	32.7%
PGD	0.8%	3.5%

(a) Standard training

	Simple	Wide
Natural	87.4%	90.3%
FGSM	90.9%	95.1%
PGD	0.0%	0.0%

(b) FGSM training

	Simple	Wide
Natural	79.4%	87.3%
FGSM	51.7%	56.1%
PGD	43.7%	45.8%

(c) PGD training

	Simple	Wide
Natural	0.00357	0.00371
FGSM	0.0115	0.00557
PGD	1.11	0.0218

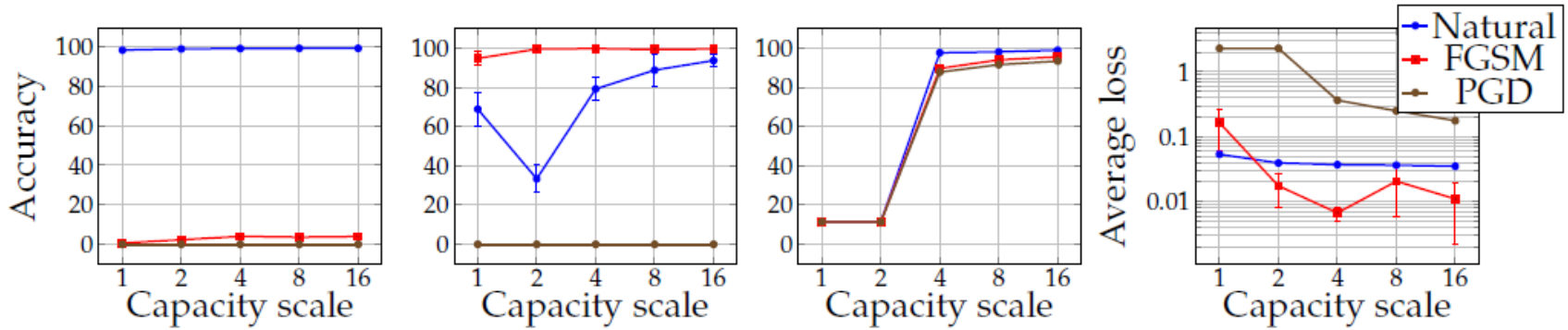
(d) Training Loss

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 and Adversarial Robustness

Either **increasing the capacity** of the network, or using a **stronger method for the inner optimization** problem reduces the effectiveness of adversarial inputs (in other words, **increase the robustness of model**).

MNIST



CIFAR10

	Simple	Wide
Natural	92.7%	95.2%
FGSM	27.5%	32.7%
PGD	0.8%	3.5%

(a) Standard training

	Simple	Wide
Natural	87.4%	90.3%
FGSM	90.9%	95.1%
PGD	0.0%	0.0%

(b) FGSM training

	Simple	Wide
Natural	79.4%	87.3%
FGSM	51.7%	56.1%
PGD	43.7%	45.8%

(c) PGD training

Simple	Wide
0.00357	0.00371
0.0115	0.00557
1.11	0.0218

(d) Training Loss

Adversarially Trained Models

MNIST

- 40 iterations of projected gradient descent
- $\alpha = 0.01$ and $\epsilon = 0.3$
- Train model on natural samples: 99.2% accuracy on validation set

Adversarially Trained Models

MNIST

- 40 iterations of projected gradient descent
- $\alpha = 0.01$ and $\epsilon = 0.3$
- Train model on natural samples: 99.2% accuracy on validation set

CIFAR10

- 7 iterations of projected gradient descent
- $\alpha = \frac{3}{255}$ and $\epsilon = \frac{8}{255}$
- Train model on natural samples: 95.2% accuracy on validation set

Adversarially Trained Models

MNIST

- 40 iterations of projected gradient descent
- $\alpha = 0.01$ and $\epsilon = 0.3$
- Train model on natural samples: 99.2% accuracy on validation set

CIFAR10

- 7 iterations of projected gradient descent
- $\alpha = \frac{3}{255}$ and $\epsilon = \frac{8}{255}$
- Train model on natural samples: 95.2% accuracy on validation set

Source models to generate adversarial examples

- A: White-box attack
- A': Black-box attacks from an independently trained copy of the network
- A_{nat}: Black-box attacks from a version of the same network trained only on natural examples
- B: Black-box attacks from a different convolution architecture

Adversarially Trained MNIST Model

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 $\varepsilon = 0.3$. 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 indepentenly initialized and trained copy of the network (A′), architecture B from [29] (B).

Adversarially Trained CIFAR10 Model

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 $\varepsilon = 8$. 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 independtly initialized and trained copy of the network (A′), a copy of the network trained on natural examples (A_{nat}).

Circumventing Defenses to Adversarial Examples

Circumventing Defenses to Adversarial Examples

Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples

Anish Athalye [*1] Nicholas Carlini [*2] David Wagner [2]

Abstract

- We identify **obfuscated gradients** as a phenomenon that leads to a **false sense of security** in defenses against adversarial examples.
  - Without a good gradient, where following the gradient does not successfully optimize the loss, iterative optimization-based methods cannot succeed.
- We describe **characteristic behaviors of defenses** exhibiting the effect, and for each of the three types of obfuscated gradients we discover, **we develop attack techniques to overcome it**.
- In a case study, examining noncertified white-box-secure defenses at **ICLR 2018**, we find obfuscated gradients are a common occurrence, with **7 of 9 defenses relying on obfuscated gradients**.

Notation

- We consider a neural network $f(.)$ used for classification where $f(x)_i$ represents the probability that image x corresponds to label i.
- We classify images, represented as $x \in [0, 1]^{w.h.c}$ for a c-channel image of width w and height h.
- We use $f^j(.)$ to refer to layer j of the neural network, and $f^{1..j}(.)$ the composition of layers 1 through j.
- We denote the classification of the network as $c(x) = \underset{i}{argmax}\ f(x)_i$, and $c^*(x)$ denotes the true label.
- We use the ℓ_∞ and ℓ_2 distortion metrics to measure similarity.

Threat Models and Attack Methods

We consider defenses designed for the **white-box setting**, where the adversary has full access to the neural network classifier (architecture and weights) and defense, **but not test-time randomness (only the distribution)**.

We construct adversarial examples with iterative optimization-based methods.

- To generate ℓ_∞ bounded adversarial examples, we use **Projected Gradient Descent (PGD)**
- To generate ℓ_2 bounded adversarial examples, we use the Lagrangian relaxation of **Carlini & Wagner (C&W)**

Obfuscated Gradients

A defense is said to cause gradient masking if it **does not have useful gradients** for generating adversarial examples.

- we refer to the case where defenses are designed in such a way that the constructed defense necessarily causes gradient masking as obfuscated gradients.

Obfuscated Gradients

A defense is said to cause gradient masking if it **does not have useful gradients** for generating adversarial examples.

- we refer to the case where defenses are designed in such a way that the constructed defense necessarily causes gradient masking as obfuscated gradients.

We discover three ways in which defenses obfuscate gradients

Obfuscated Gradients

A defense is said to cause gradient masking if it **does not have useful gradients** for generating adversarial examples.

- we refer to the case where defenses are designed in such a way that the constructed defense necessarily causes gradient masking as obfuscated gradients.

We discover three ways in which defenses obfuscate gradients

1. **Shattered Gradients**
   Shattered Gradients are caused when a defense is nondifferentiable, introduces numeric instability, or otherwise causes a **gradient to be nonexistent or incorrect**.

Obfuscated Gradients

A defense is said to cause gradient masking if it **does not have useful gradients** for generating adversarial examples.

- we refer to the case where defenses are designed in such a way that the constructed defense necessarily causes gradient masking as obfuscated gradients.

We discover three ways in which defenses obfuscate gradients

1. **Shattered Gradients**
   Shattered Gradients are caused when a defense is nondifferentiable, introduces numeric instability, or otherwise causes a **gradient to be nonexistent or incorrect**.
2. **Stochastic Gradients**
   Stochastic Gradients are caused by **randomized defenses**, where either the network itself is randomized or the input is randomly transformed before being fed to the classifier, causing the gradients to become randomized.

Obfuscated Gradients

A defense is said to cause gradient masking if it **does not have useful gradients** for generating adversarial examples.

- we refer to the case where defenses are designed in such a way that the constructed defense necessarily causes gradient masking as obfuscated gradients.

We discover three ways in which defenses obfuscate gradients

1. **Shattered Gradients**
   Shattered Gradients are caused when a defense is nondifferentiable, introduces numeric instability, or otherwise causes a **gradient to be nonexistent or incorrect**.
2. **Stochastic Gradients**
   Stochastic Gradients are caused by **randomized defenses**, where either the network itself is randomized or the input is randomly transformed before being fed to the classifier, causing the gradients to become randomized.
3. **Exploding & Vanishing Gradients**
   Exploding & Vanishing Gradients are often caused by defenses that consist of **multiple iterations of neural network** evaluation, feeding the output of one computation as the input of the next. This type of computation, when unrolled, can be viewed as an **extremely deep neural network evaluation**, which can cause vanishing/exploding gradients.

Identifying Obfuscated & Masked Gradients

Characteristic behaviors of defenses which cause gradient obfuscation

Identifying Obfuscated & Masked Gradients

Characteristic behaviors of defenses which cause gradient obfuscation

- **One-step attacks perform better than iterative attacks.**
  - Iterative optimization-based attacks are strictly stronger than single-step attacks.
  - If single-step methods give performance superior to iterative methods, it is likely that the iterative attack is becoming stuck in its optimization search at a local minimum.

Identifying Obfuscated & Masked Gradients

Characteristic behaviors of defenses which cause gradient obfuscation

- **One-step attacks perform better than iterative attacks.**
  - Iterative optimization-based attacks are strictly stronger than single-step attacks.
  - If single-step methods give performance superior to iterative methods, it is likely that the iterative attack is becoming stuck in its optimization search at a local minimum.
- **Black-box attacks are better than white-box attacks**
  - Attacks in the white-box setting should perform better
  - If a defense is obfuscating gradients, then black-box attacks (which do not use the gradient) often perform better than white-box attacks

Identifying Obfuscated & Masked Gradients

Characteristic behaviors of defenses which cause gradient obfuscation

- **One-step attacks perform better than iterative attacks.**
  - Iterative optimization-based attacks are strictly stronger than single-step attacks.
  - If single-step methods give performance superior to iterative methods, it is likely that the iterative attack is becoming stuck in its optimization search at a local minimum.
- **Black-box attacks are better than white-box attacks**
  - Attacks in the white-box setting should perform better
  - If a defense is obfuscating gradients, then black-box attacks (which do not use the gradient) often perform better than white-box attacks
- **Unbounded attacks do not reach 100% success**
  - With unbounded distortion, any classifier should have 0% robustness to attack.

Identifying Obfuscated & Masked Gradients

Characteristic behaviors of defenses which cause gradient obfuscation

- **One-step attacks perform better than iterative attacks.**
  - Iterative optimization-based attacks are strictly stronger than single-step attacks.
  - If single-step methods give performance superior to iterative methods, it is likely that the iterative attack is becoming stuck in its optimization search at a local minimum.
- **Black-box attacks are better than white-box attacks**
  - Attacks in the white-box setting should perform better
  - If a defense is obfuscating gradients, then black-box attacks (which do not use the gradient) often perform better than white-box attacks
- **Unbounded attacks do not reach 100% success**
  - With unbounded distortion, any classifier should have 0% robustness to attack.
- **Random sampling finds adversarial examples**
  - Bruteforce random search (e.g., randomly sampling 10^5 or more points) within some ϵ-ball should not find adversarial examples when gradient-based attacks do not.

Identifying Obfuscated & Masked Gradients

Characteristic behaviors of defenses which cause gradient obfuscation

- **One-step attacks perform better than iterative attacks.**
  - Iterative optimization-based attacks are strictly stronger than single-step attacks.
  - If single-step methods give performance superior to iterative methods, it is likely that the iterative attack is becoming stuck in its optimization search at a local minimum.
- **Black-box attacks are better than white-box attacks**
  - Attacks in the white-box setting should perform better
  - If a defense is obfuscating gradients, then black-box attacks (which do not use the gradient) often perform better than white-box attacks
- **Unbounded attacks do not reach 100% success**
  - With unbounded distortion, any classifier should have 0% robustness to attack.
- **Random sampling finds adversarial examples**
  - Bruteforce random search (e.g., randomly sampling 10^5 or more points) within some ϵ-ball should not find adversarial examples when gradient-based attacks do not.
- **Increasing distortion bound does not increase success**
  - A larger distortion bound should monotonically increase attack success rate.

Attack Techniques

There is a number of techniques to overcome obfuscated gradients

- **Backward Pass Differentiable Approximation (DBPA)** to overcome shattered gradients
- **Expectation over Transformation** to overcome stochastic gradients
- **Reparameterization** to overcome exploding & vanishing gradients

Backward Pass Differentiable Approximation (BPDA)

To attack defenses where gradients are not readily available, such as shattered gradients, we introduce a technique we call **Backward Pass Differentiable Approximation (BPDA)**.

Backward Pass Differentiable Approximation (BPDA)

To attack defenses where gradients are not readily available, such as shattered gradients, we introduce a technique we call **Backward Pass Differentiable Approximation (BPDA)**.

Many non-differentiable defenses can be expressed as follows:

- Given a pre-trained classifier $f(.)$, construct a preprocessor $g(.)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(.)$ satisfies $g(x) \approx x$ (e.g., such a $g(.)$ may perform image denoising to remove the adversarial perturbation)

Backward Pass Differentiable Approximation (BPDA)

To attack defenses where gradients are not readily available, such as shattered gradients, we introduce a technique we call **Backward Pass Differentiable Approximation (BPDA)**.

Many non-differentiable defenses can be expressed as follows:

- Given a pre-trained classifier $f(.)$, construct a preprocessor $g(.)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(.)$ satisfies $g(x) \approx x$ (e.g., such a $g(.)$ may perform image denoising to remove the adversarial perturbation)
  - If $g(.)$ is smooth and differentiable, then computing gradients through the combined network $\hat{f}$ is often sufficient to circumvent the defense
  - However, recent work has constructed functions $\boldsymbol{g}(.)$ **which are neither smooth nor differentiable**

Backward Pass Differentiable Approximation (BPDA)

To attack defenses where gradients are not readily available, such as shattered gradients, we introduce a technique we call **Backward Pass Differentiable Approximation (BPDA)**.

Many non-differentiable defenses can be expressed as follows:

- Given a pre-trained classifier $f(.)$, construct a preprocessor $g(.)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(.)$ satisfies $g(x) \approx x$ (e.g., such a $g(.)$ may perform image denoising to remove the adversarial perturbation)
  - If $g(.)$ is smooth and differentiable, then computing gradients through the combined network $\hat{f}$ is often sufficient to circumvent the defense
  - However, recent work has constructed functions $\boldsymbol{g}(.)$ **which are neither smooth nor differentiable**
- 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.

Backward Pass Differentiable Approximation (BPDA)

To attack defenses where gradients are not readily available, such as shattered gradients, we introduce a technique we call **Backward Pass Differentiable Approximation (BPDA)**.

Many non-differentiable defenses can be expressed as follows:

- Given a pre-trained classifier $f(.)$, construct a preprocessor $g(.)$ and let the secured classifier $\hat{f}(x) = f(g(x))$ where the preprocessor $g(.)$ satisfies $g(x) \approx x$ (e.g., such a $g(.)$ may perform image denoising to remove the adversarial perturbation)
  - If $g(.)$ is smooth and differentiable, then computing gradients through the combined network $\hat{f}$ is often sufficient to circumvent the defense
  - However, recent work has constructed functions $\boldsymbol{g(.)}$ **which are neither smooth nor differentiable**
- 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.
- We perform **forward propagation** through the neural network **as usual**, but on the **backward pass**, we **replace $g(.)$ with the identity function**.
  - This gives us an approximation of the true gradient.

Expectation over Transformation

For defenses that employ randomized transformations to the input or use a stochastic classifier, we apply Expectation over Transformation (EOT)

- When attacking a classifier $f(.)$ that first randomly transforms its input according to a function $t(.)$ 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.

Reparameterization

We solve vanishing/exploding gradients by reparameterization.

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

Reparameterization

We solve vanishing/exploding gradients by reparameterization.

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

To resolve this issue

- We make a **change-of-variable** $x = h(z)$ for some function $h(.)$ such that $g(h(z)) = h(z)$ for all z, but $h(.)$ is differentiable.
  - For example, if $g(.)$ 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.

Case Study: ICLR 2018 Defenses

As a case study for evaluating the prevalence of obfuscated gradients, we study the **ICLR 2018** non-certified defenses that argue robustness in a white-box threat model.

- 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.
- **Of the 9 accepted papers, 7 rely on obfuscated gradients.**

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.