Adversarial Attacks

Jee-Hyong Lee

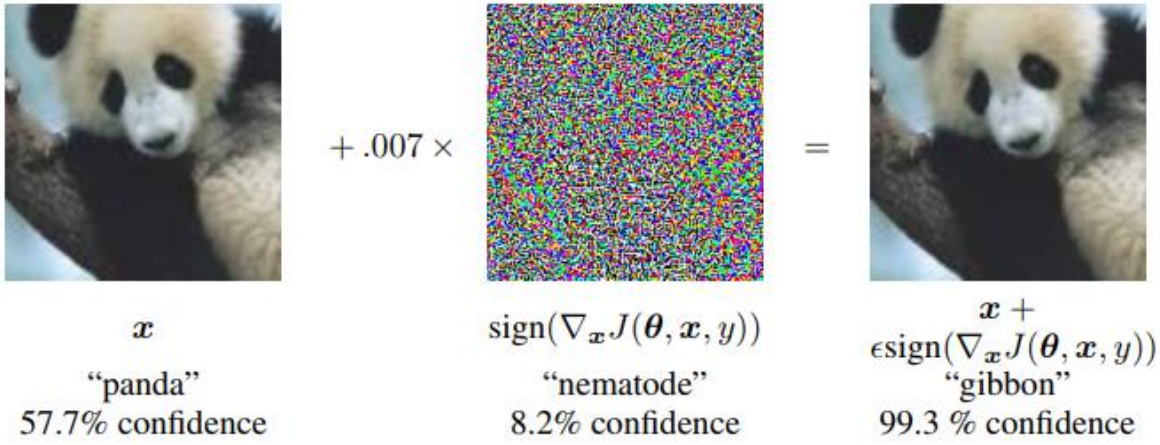
Sungkyunkwan Univ.

Confusing?

Source: https://twitter.com/bretmcg/status/779795602372759553

Introduction

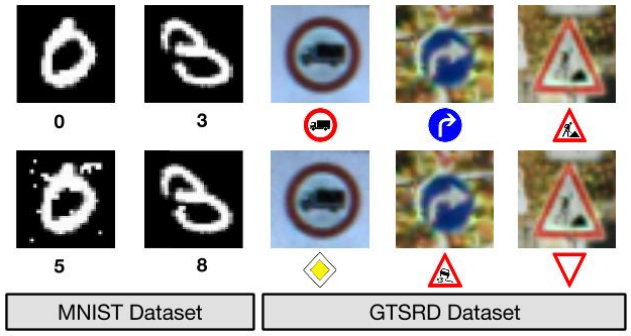
- **Adversarial Attack**
  - Change the result of the model by adding imperceptible noise to input
  - To deceive image classification model

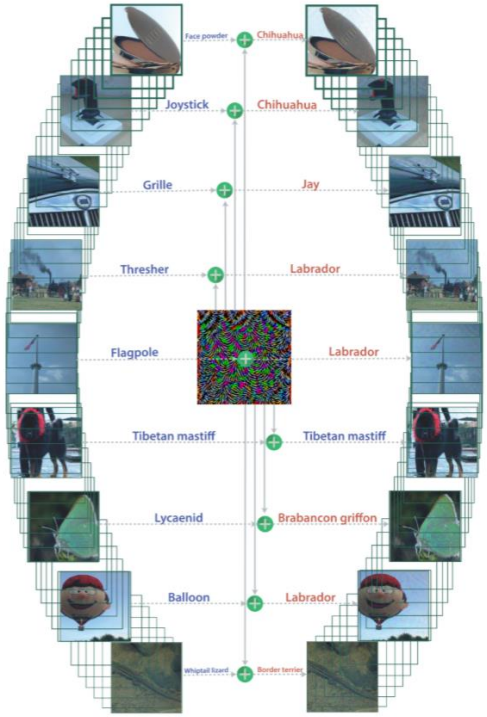


Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).

Introduction

- **Varieties of Adversarial Attacks**





<Adversarial Examples> <Universal Adversarial Examples> <One Pixel Adversarial Example>

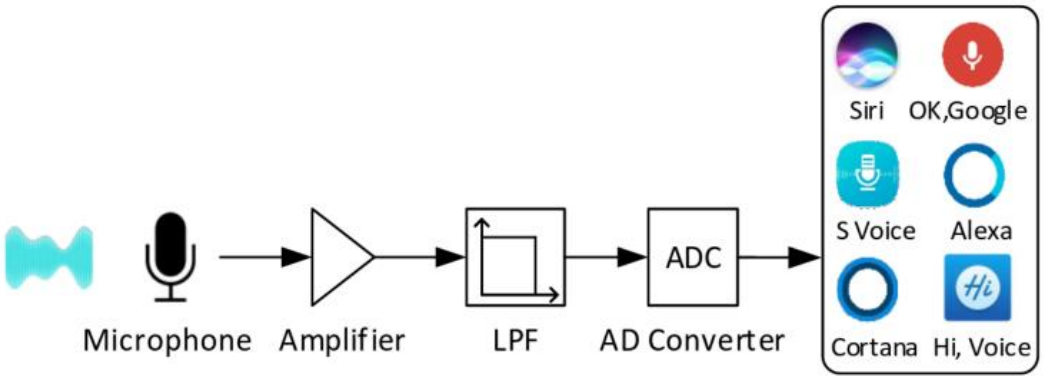
Papernot, Nicolas, et al. "Practical black-box attacks against machine learning." Proceedings of the 2017 ACM on Asia conference on computer and communications security. ACM, 2017.
Moosavi-Dezfooli, Seyed-Mohsen, et al. "Universal adversarial perturbations." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
Su, Jiawei, Danilo Vasconcellos Vargas, and Kouichi Sakurai. "One pixel attack for fooling deep neural networks." IEEE Transactions on Evolutionary Computation (2019).

Introduction

- **Real World**

True image

Adversarial image

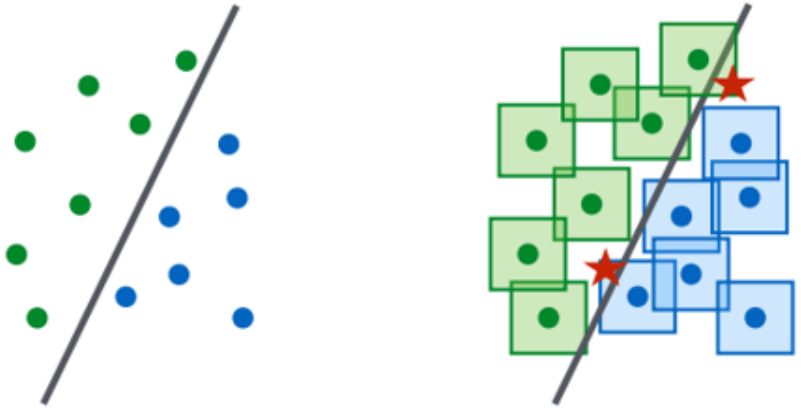


Introduction

- **Adversarial examples pose a security concern for machine learning models**
  - An attack created to fool one network also fools other networks.
  - Attacks also work in the physical world.
  - For Deep Neural networks, it is very easy to generate adversarial examples but this issue affects other ML classifiers.
  - Many defense strategies have been proposed, they all fail against strong attacks.

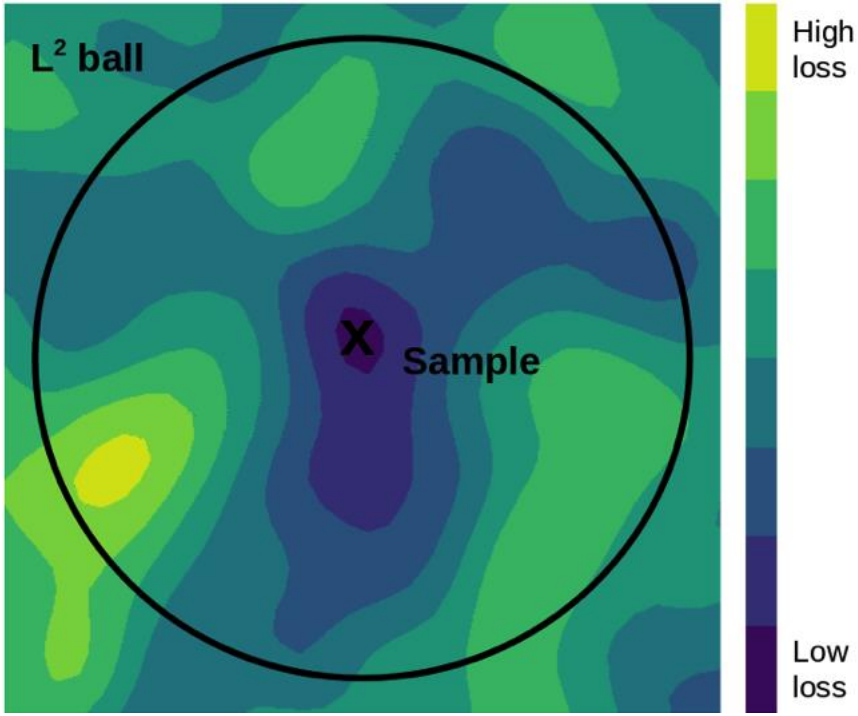
Why it happens?

- **Sample cross the decision boundary**
  - Small differences on input pixel contribute to a dramatic difference in $weights \times inputs$
  - Simple decision boundary does not separate the l_∞-balls around the data points



Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).
https://towardsdatascience.com/know-your-adversary-understanding-adversarial-examples-part-1-2-63af4c2f5830
Madry, Aleksander, et al. "Towards deep learning models resistant to adversarial attacks." *arXiv preprint arXiv:1706.06083*(2017).

Why it happens?



Adversarial Example

- **An example $\tilde{X}$ is said adversarial if:**
  - It is close to a sample in the true distribution:

$$D(X, \tilde{X}) \leq \epsilon$$

  - It is misclassified

$$\text{argmax}\, P(y|\tilde{X}) \neq y_{true}$$

  - It belongs to the input domain. E.g. for images

$$0 \leq \tilde{X} \leq 255$$

Adversarial Example

- **To measure the similarity between samples:**
  - A good measure between samples is still an active area or research. Commonly, researchers use:
  - L_2 norm (Euclidean distance):

$$\left\|\tilde{X} - X\right\|_2$$

  - L_∞ norm

$$\left\|\tilde{X} - X\right\|_\infty = \max_{ij}\left|\tilde{X}_{ij} - X_{ij}\right|$$

Adversarial Example

- **Distance Measure**

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{N} |x_i|^p \right)^{\frac{1}{p}}$$

- ℓ_0: control the number of pixels that are modified
- ℓ_1: control the total amount of pixel value changes
- ℓ_2: control the Euclidean distance of pixel value changes
- ℓ_∞: control the maximum pixel value change in any coordinate

Adversarial Example

- **Surprise !!**
  - They are not specific to a particular architecture.
  - Even the misclassified classes are mostly the same across various models.
- **Some causes**
  - Non-linearity of neural networks
  - Insufficient regularization
  - Insufficient model averaging

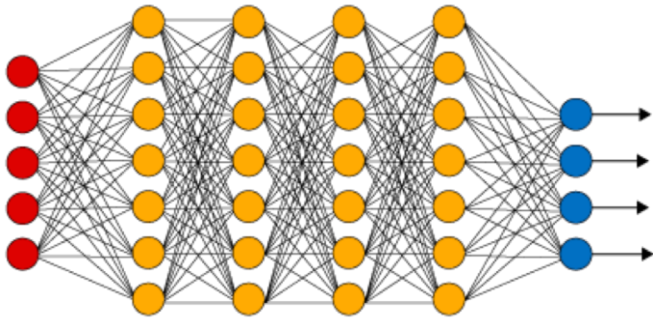
Threat Models

- **Black-box:**
  - Access to domain data,
  - Can feed inputs to the model and observe outputs.

- **White-box:**
  - Full knowledge of the data, architecture and parameters.

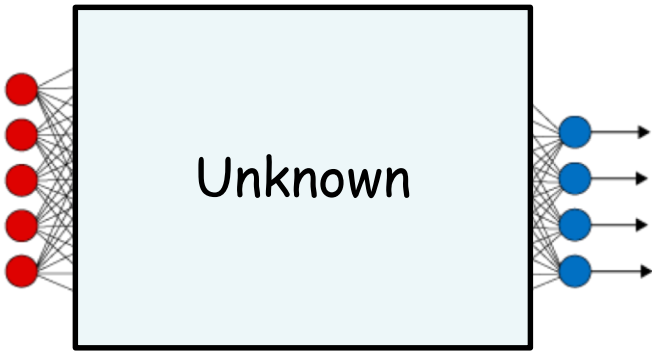
Threat Models

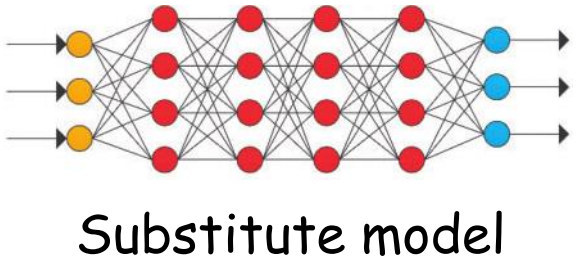
- **White-box Attacks**
  - Every parameter of the network is known
  - Analyze the network and find out adversarial examples



Threat Models

- **Black-box Attacks**
  - Parameters of the network are not known, but training data is open
  - Build a substitute model using the training data
  - Find out adversarial examples for the substitute model
    - The adversarial examples can also fool the original network





Targeted vs. Untargeted Attacks

According to the attacker's goal:

Non-targeted attacks: attacker tries to fool a classifier to get any incorrect clas

$$\text{argmax}\, P(\mathbf{y}|\tilde{X}) \neq y_{\text{true}}$$

Targeted attacks: attacker tries to fool a classifier to predict a particular class

$$\text{argmax}\, P(\mathbf{y}|\tilde{X}) = y_{\text{target}}$$

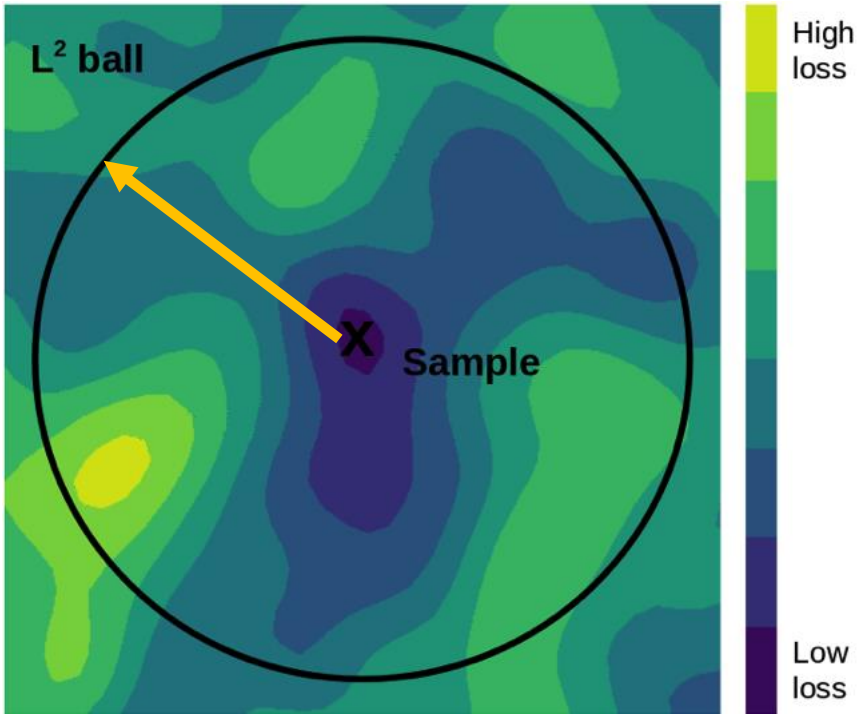
Attacks

One Step Gradient Method

- **Untargeted: Fast Gradient Sign Method (FGSM)**
  - Add noise by finding reverse gradient direction of true label
  - $x^* = x + \epsilon \text{sign}\left(\nabla_x L(x, y_{true})\right)$

- **Targeted: One step least-likely class method**
  - Add noise by finding gradient direction of target label
  - $x^* = x - \epsilon \text{sign}\left(\nabla_x L\left(x, y_{target}\right)\right)$
  - Mostly, least likely class is used

One Step Gradient Method

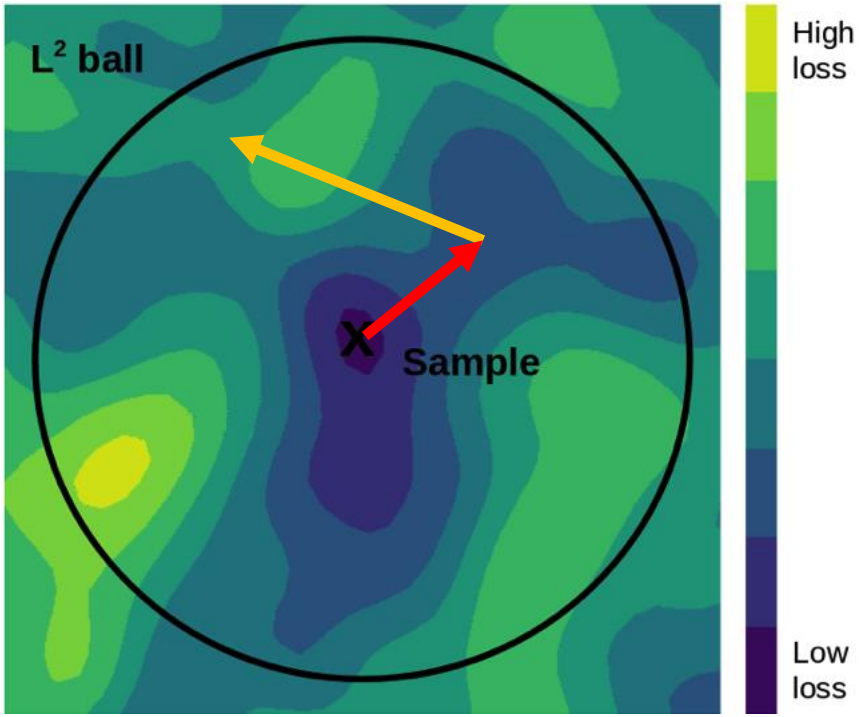


One Step Gradient Method

- **Randomized Fast Gradient Sign Method (RAND+FGSM)**
  - First apply a small random perturbation before using FGSM
  - $x' = x + \alpha \text{sign}(\mathcal{N}(0^n, \text{I}^n)), \alpha < \epsilon$
  - $x^* = x' + (\epsilon - \alpha)\text{sign}\left(\nabla_x L(x, y_{true})\right)$

- **Effective and Fast**
  - It is used for baseline, usually

One Step Gradient Method

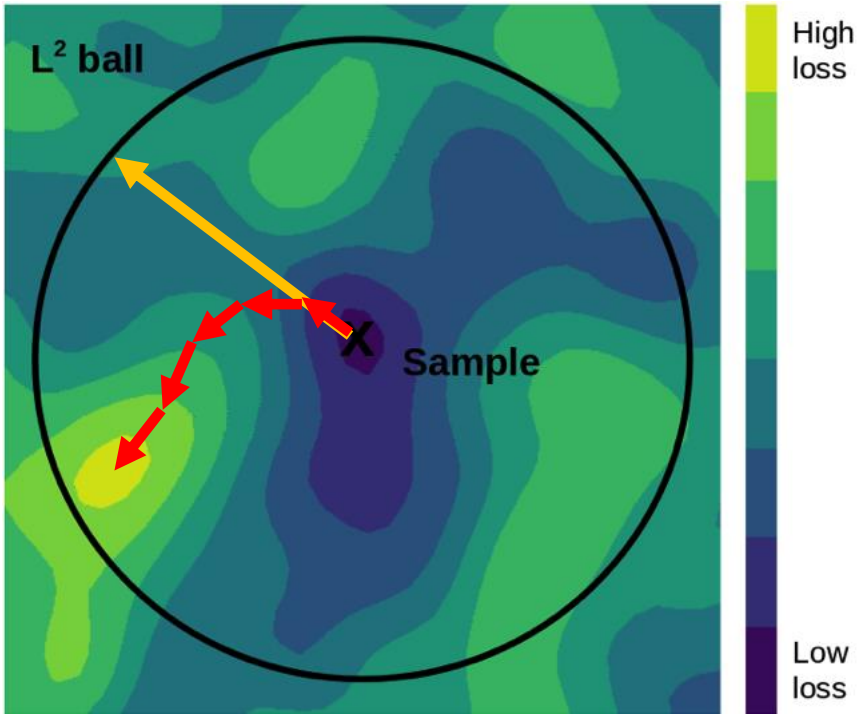


Iterative Methods

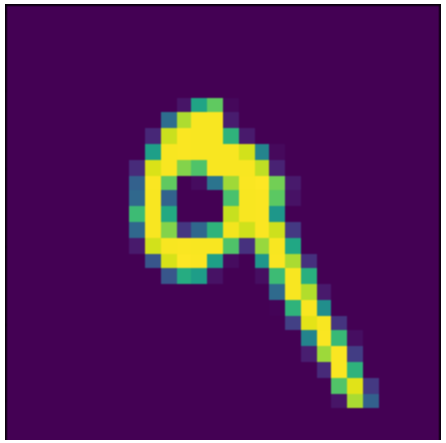
- **Basic iterative method (iter. basic)**
  - Add noise by finding reverse gradient direction of true label, iteratively
  - $x_0^* = x,\ x_{N+1}^* = Clip_{x,\epsilon}\{x_N^* + \alpha \text{sign}\left(\nabla_x L(x_N^*, y_{true})\right)\}$
- **Iterative least-likely class method (iter. l.l.)**
  - Add noise by finding gradient direction of target label, iteratively
  - $x_0^* = x,\ x_{N+1}^* = Clip_{x,\epsilon}\{x_N^* - \alpha \text{sign}\left(\nabla_x L\left(x_N^*, y_{target}\right)\right)\}$
- **Stronger than one step methods**

Iterative Methods

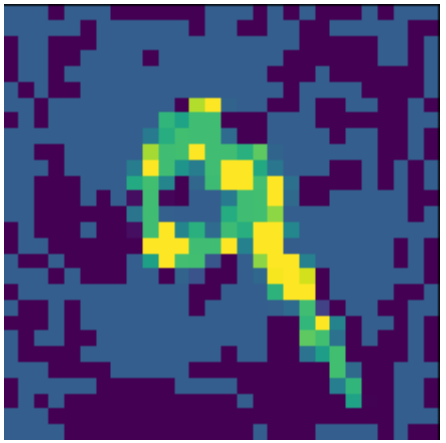
- **Why better?**



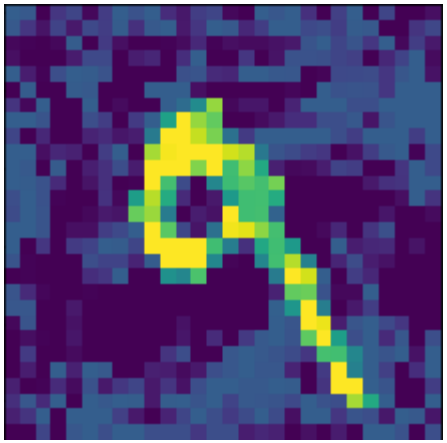
Iterative Methods



<Image>



<One-step>



<Iterative>

Iterative Methods

- **Projected Gradient Descent (PGD)**
  - Apply FGSM iteratively based on random starting points around data, and restart at random points if fails
  - Universal first order adversary

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

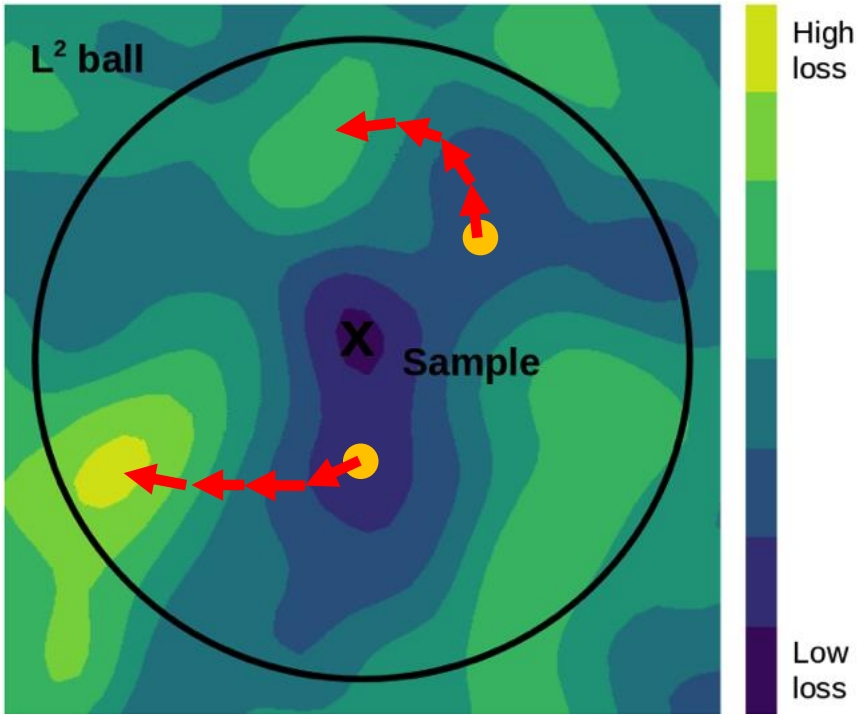
  - or

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

  - Pros
    - Known as strongest attack
  - Cons
    - Slow

Iterative Methods

- **Projected Gradient Descent (PGD)**



Defense

Defense Approaches

- **Adversarial Training**
  - Train model using adversarial examples as well as natural data
- **Filtering/Detecting**
  - Learning pattern of adversarial examples or perturbations
  - Reject adversarial samples without classifying them using a specialized side model
- **Denoising (Preprocessing)**
  - Reduce noise in the input using denoiser

Adversarial Training

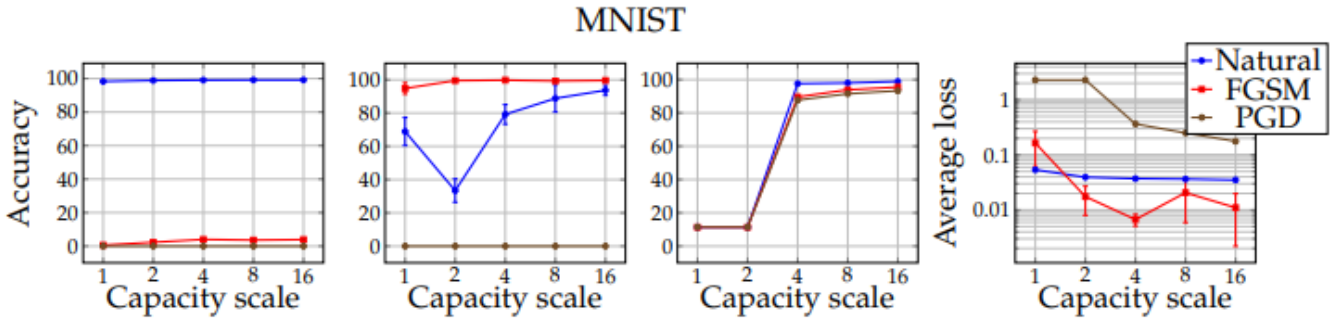
- **Idea**
  - Including adversarial samples with their correct classification in the training data
  - Formulation as saddle point problem

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

- **Protocols**
  - Starting with pre-trained model or from scratch
  - Fraction of adversarial samples per mini-batch
  - Replace original samples with adversarial ones, or keep both
  - Pre-computing of adversarial samples vs. computing them on-the-fly on the current version of the trained model

Adversarial Training

- **[Madry et al., 2017] trains robust models on MNIST and CIFAR10:**
  - Use PGD with random restart
  - Replace clean samples with adversarial ones



CIFAR10

	Simple	Wide	Simple	Wide	Simple	Wide	Simple	Wide
Natural	92.7%	95.2%	87.4%	90.3%	79.4%	87.3%	0.00357	0.00371
FGSM	27.5%	32.7%	90.9%	95.1%	51.7%	56.1%	0.0115	0.00557
PGD	0.8%	3.5%	0.0%	0.0%	43.7%	45.8%	1.11	0.0218
	(a) Standard training		(b) FGSM training		(c) PGD training		(d) Training Loss	

Adversarial Training + α

- **Adversarial Logit Pairing [Kannan et al., 2018]**
  - Use targeted PGD attacks during training and testing
  - Adversarial Logit Pairing (ALP): penalize$\|Z(x) - Z(x_{adv})\|_2$
- **Defensive Distillation [Papernot et al. 2015]**
  - Train a teacher model on the original data.
  - Train a student model based on the predictions of the teacher.
  - Only use the student for prediction.
- **Label Smoothing**
  - Has been shown to have an equivalent effect [Warde-Farley and Goodfellow, 2016]

Adversarial Training + α

- Regularization
  - Parseval networks [Cisse et al., 2017]: regularize each layer weights to be 1-Lipschitz.
- Dropout
  - Dropout at test time [Feinman et al., 2017].
  - Stochastic activation pruning [Dhillon et al., 2018]: similar to dropout at test time, where the probability for an activation to be used is proportional to its absolute value.
- Generative models
  - GANs [Samangouei et al., 2018]: use generative model to recreate a clean input.

Detection

- Convolutional filter statistics [Li and Li, 2016]: train binary SVMs at each layer after applying PCA.
- Binary detector trained on inputs [Gong et al., 2017].
- Binary detector trained on multiple internal layers of the classifier [Metzen et al., 2017a].
- Kernel density estimation [Feinman et al., 2017].
- Bayesian uncertainty estimates of the inputs in dropout neural networks [Feinman et al., 2017].
- LID detector built from statistics computed at internal layers [Ma et al., 2018].

Denoising

- Blurring filters [Xu et al., 2017a]
- JPEG compression [Dziugaite et al., 2016, Das et al., 2017, Guo et al., 2018]
- Generative model [Ilyas et al., 2017] as preprocessing step to clean the data.
- PixelDefend [Song et al., 2018]: use PixelCNN density models to detect and clean adversarial noise.
- Non-differentiable and randomized input transformations [Guo et al., 2018].
- Thermometer encoding [Buckman et al., 2018]: encode inputs as level functions to introduce gradient masking.