Adversarial Robustness in Machine Learning Models

Project 8, CS 490

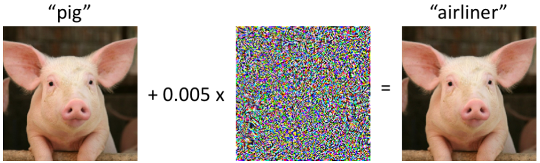
Rajdeep Haldar

Purdue University

Prof. Guang Lin

Motivation

- Neural networks exhibit universal approximation property and generalize well.
- However, they show imperceptible perturbation in original data that can **fool** (misclassify) or hamper the performance (increase in loss) of the model.



The model is so good it can even make a pig fly!

- Adversaries can misuse this phenomenon: Identity theft, Automated driving, Bypassing spam filters etc.
- Understanding the foundation and mechanism behind such attacks is of cardinal importance in developing defense systems.

Formulation

- $(x, y) = \mathcal{Z} \in \mathcal{X} \times \mathcal{Y} = \tilde{\mathcal{Y}}$ represents an observation pair where $x \in \mathcal{X}$ is the predictor and $y \in \mathcal{Y}$ is the response, where $\mathcal{Y} = \mathbb{R}^k$ or $\mathcal{Y} = \{1, \dots, k\}$ for regression and classification problems respectively.
- $f : \mathcal{X} \to \mathbb{R}^k \in \mathcal{C}$ be our model, where $\mathcal{C}$ is some function class.
- $\mathcal{L} : \mathbb{R}^k \times \mathcal{Y} \to \mathbb{R}$ denotes a loss function that the model wants to minimize (MSE, CE, any negative-log-likelihood).
- Training is equivalent to finding a model which minimises the loss.

$$f = \arg\min_{f \in \mathcal{C}} \mathbb{E}\mathcal{L}(f(x), y)$$

Generating attacks (White Box)

- Adversarial example $x' = x + \Delta$.
- We want x' and x to be close in some $d(x, x')$ distance/similarity metric. (Usually the ℓ_p norms). Yet we want x' to cross the decision boundary of the model.
- The Decision boundary of the model can be characterized by the loss $\mathcal{L}$ that we used to train the model on or any surrogate of $\mathcal{L}$.
- Attacks can be obtained by solving the following optimization problem:

$$\Delta = \arg \max_{d(x,x') \leq \epsilon} \mathcal{L}(f(x'), y) \tag{1}$$

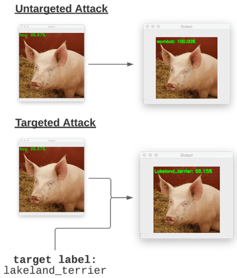
- All attacks can be understood as variants of solving the above optimization problem, with various combinations of $d, \mathcal{L}$ or its surrogate.

- FGSM (Fast-signed gradient max): $\Delta = \epsilon \cdot \text{sign}\left(\nabla_x \mathcal{L}(f(x), y)\right)$
- PGD (Projected-gradient-descent), with step size γ and T steps; $\delta^t = \Pi_{B_\infty(x+\delta^{t-1},\epsilon)}\left(\delta^{t-1} + \gamma \cdot \text{sign}\left(\nabla_x \mathcal{L}(f(x+\delta^{t-1}), y)\right)\right)$. Here $d(x, x')$ is the ℓ_∞ metric, $\mathcal{L}$ is the CE loss for classification.
- Other popular attacks Momentum iterative attack, Carlini and Wagner (CW), Deep fool attack, Jacobian Saliency Map (JSMA), etc.
- Essentially all these methods try to maximise a surrogate of original loss using some gradient-based steps, while respecting the distance constraint.

Untargeted vs Targeted

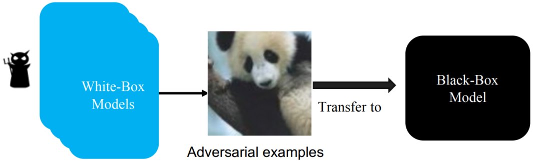
- In the literature (1) is known as an *untargeted* attack. The attack just wants to misclassify, and doesn't care about the final label after the attack.
- We can also conduct *targeted* attacks to change the original label y to a specific target label y'.

$$\Delta = \arg \min_{d(x,x') \leq \epsilon} \mathcal{L}(f(x'), y'); y \neq y' \tag{2}$$



Black box I

- The attacks discussed till now are called *white box* attacks as they require access to the trained model f.
- In contrast you could also have *black box* attacks when you don't have access to f. These can be broken down into 3 broad categories: Transferable Attacks, Score based Attacks, Decision-based Attacks.
- **Transferable Attacks:** These attacks rely on a property exhibited by adversarial attacks known as Transferibility. We require access to the training dataset though.



Black box II

- **Score based Attacks:** Only rely on the predicted scores (e.g. class probabilities or logits) of the model. On a conceptual level, these attacks use the predictions to numerically estimate the gradient, to essentially take a perturbation in maximum loss direction.
  NES, ZOSignSGD, Bandit-prior, ECO attack, SimBA, SignHunter, Sqaure attack etc.
- **Decision-based Attacks:** Direct attacks that solely rely on the final decision of the model.
  Boundary attack, OPT attack, Sign-OPT, Evoluationary attack, GeoDA, HSJA, Sign Flip, RayS etc.

Trainable Defense

The most straightforward and effective way to make the model robust is to provide the model with these attacked samples (adversarial examples) with true labels.

Standard adversarial training:

$$f = \arg\min_{f \in \mathcal{C}} \mathbb{E} \max_{d(x,x') \leq \epsilon} \mathcal{L}(f(x'), y) \tag{3}$$

TRADES:

$$f = \arg\min_{f \in \mathcal{C}} \mathbb{E}\{\mathcal{L}(f(x), y) + \max_{d(x,x') \leq \epsilon} \beta \mathcal{L}(f(x'), f(x))\} \tag{4}$$

Other defense methods like outlier detection and post/pre-processing methods are omitted. However, students can pursue related methods of interest and are not only limited to the methods mentioned.

Proposed Tasks

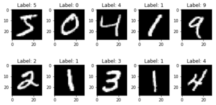
1. Develop code for white-box methods. Student should try to implement atleast FGSM, PGD, CW attack, DeepFool, JSMA. A comprehensive list of attacks and references will be provided later.
2. Develop code for score-based attacks. Should try at least 3.
3. Develop code for decision-based attacks. Should try to include atleast Boundary attack, HSJA (Hopskipjump).
4. Analysis on Transferable attacks trained on models with different architecture.
5. Train robust models using standard adversarial training with different attacks for example PGD, random FGSM, CW and using TRADES with optimal parameters.

We would like to create a comprehensive table comparing all the methods for MNIST, CIFA10, and SVHN vision datasets. If students want to use different vision datasets, that is fine as long as the resources are available.

	Clean	AT (CW)	AT(PGD)	TRADES (β_1)	TRADES (β_2)
FGSM					
PGD					
CW			acc (sd)		
JSMA					
DF					
NES					
SA					
BA					
HSJA					

Dataset I

- MNIST

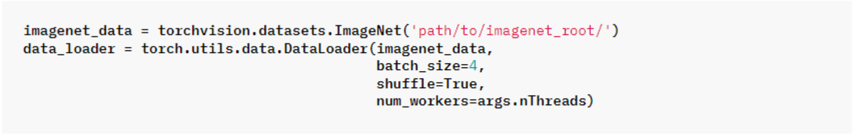


- CIFAR 10

- SVHN

Dataset II

Preferably use Pytorch. Can directly access datasets as tensor iterables as follows:



```
imagenet_data = torchvision.datasets.ImageNet('path/to/imagenet_root/')
data_loader = torch.utils.data.DataLoader(imagenet_data,
                                          batch_size=4,
                                          shuffle=True,
                                          num_workers=args.nThreads)
```

For custom datasets, one can create a PyTorch data loader object by feeding the labels and covariates (images) as a tuple.

References I

Pytorch: Installation, Quick start guide
White-Box attacks:

- Fast Gradient Method (FGSM)
- Projected Gradient Descent (PGD)
- Carlini & Wagner (C&W)
- Jacobian Saliency Map (JSMA)
- Universal Perturbation
- DeepFool
- Wasserstein Attack

References II

Black-box Score based attacks

- NES
- ZOSignSGD
- Square Attack
- Sign Hunter
- Bandit Prior

References III

Black-box Decision based attacks

- Boundary Attack
- HSJA
- OPT
- Sign Flip