Deep Learning

Exercise 11: Adversarial Training

Room: **BIN-1-B.01**
Instructor: Manuel Günther
Email: guenther@ifi.uzh.ch
Office: AND 2.54

Friday, May 20, 2022

Outline

Outline

1. Adversarial Examples via FGS
    - Adversarial Training in PyTorch
    - Evaluating Adversarial Stability

Adversarial Examples via FGS

Gradient Calculation

- Loss function categorical $\mathcal{J}^{\text{CCE}}$
- Gradient w.r.t. $\mathcal{X}$: $\nabla_{\mathcal{X}} = \frac{\partial \mathcal{J}^{\text{CCE}}}{\partial \mathcal{X}}$
  - $\rightarrow$ Enable gradient for input: `X.requires_grad_(True)`
  - $\rightarrow$ Compute loss: `J = loss(X,t)`
  - $\rightarrow$ Compute gradient: `J.backward()`
  - $\rightarrow$ Access gradient: `X.grad`

Fast Gradient Sign

- Adversarial input:

$$\check{\mathcal{X}}_{\text{FGS}} = \mathcal{X} + \alpha\,\text{sign}(\nabla_{\mathcal{X}})$$

- Clip to pixel range $[0, 1]$

Noisy Image

- Noisy input:

$$\check{\mathcal{X}}_{\text{noise}} = \mathcal{X} + \alpha\,\{-1, 1\}^{D\times E}$$

  - $\rightarrow$ Select -1 or 1 for each pixel
- Clip to pixel range $[0, 1]$

Adversarial Training in PyTorch

Training Steps

1. Train on original (clean) samples: one batch
   $\rightarrow$ Use $\mathcal{J}^{\text{CCE}}$ to compare logits with original targets
2. Create adversarial samples (or noisy images) for this batch
3. Train on batch of adversarial samples
   $\rightarrow$ Use $\mathcal{J}^{\text{CCE}}$ to compare logits with original targets

Things to Consider

- When and how often do I need to update weights?
- How often do I need to zero out the gradients?
- Note: there are faster versions of adversarial training

Evaluating Adversarial Stability

Validation/Test Set Accuracies

- Accuracy on clean samples
  $\rightarrow$ On all validation/test samples
- Accuracy on adversarial samples
  $\rightarrow$ Only for correctly classified samples

Adversarial Accuracy

- Generate adversarial samples (FGS) with $\alpha = 0.3$ on network
- Check if network output has been altered by FGS
- Success case (in network defense perspective):
  $\rightarrow$ Output is still the original class

Outline

2. Adversarial Training
   - Dataset and Model
   - Image Manipulations
   - Training and Evaluation

Dataset and Model

Task 1: Dataset

- We use the default MNIST training and validation sets
  $\rightarrow$ Select appropriate batch sizes

Task 2: Classification Network

- We use the same network topology as in Assignment 8

Image Manipulations

Task 3: Fast Gradient Sign

- Implement the fast gradient sign method
- Compute gradient of loss w.r.t. the input
- Apply FGS gradient ascent with $\alpha = 0.3$ to create FGS samples

Task 4: Noise

- Apply noise with the same $\alpha = 0.3$

Training and Evaluation

Task 5: Training Loop

- Implement training loop for one epoch
- Three variants:
  - → No additional training samples
  - → Additional training with FGS samples
  - → Additional training with noise samples

Task 6: Validation Loop

- Compute classification accuracy on validation set
- Compute adversarial stability on validation set
  - → Generate FGS adversarial samples for current network
  - → Remember that FGS needs to compute gradients

Training and Evaluation

Task 7: Training of Three Networks

- Instantiate three identical networks
- Use SGD optimizer with appropriate learning rate
- Train the three networks for 10 epochs:
  - → Train one network using only clean samples
  - → Train another network using clean and adversarial samples
  - → Train the third network using clean and noisy samples
- Evaluate and store accuracies after each epoch

Task 8: Plotting of Accuracies

- Plot accuracies of three networks on clean data
- Plot accuracies of three networks on adversarial samples

Training and Evaluation

Exemplary Accuracies