Homework Assignment 2

Response by: **Prathvish Mithare (7028692)**

Due: **2:00pm Thursday, 7 December 2023 on CISPA CMS**

Collaboration Policy: You should do this assignment by yourself and submit your own answers. You may discuss the problems with anyone you want and it is also fine to get help from anyone on problems with LaTeX or Jupyter/Python. You should note in the *Collaborators* box below the people you collaborated with.

Collaborators: Subrat Kishore Dutta, Somrita Ghosh

Problem 1 (10 pts) Consider the same setting of binary logistic regression as in Problems 1-2 of Homework Assignment 1. Recall from Lecture 4, adversarial training aims to solve the following min-max optimization problem using projected gradient descent (PGD) in the context of logistic regression:

$$\min_{\boldsymbol{w}} \mathbb{E}_{(\boldsymbol{x},y)\sim\mu}\left[\max_{\boldsymbol{x}'\in\mathcal{B}_\epsilon(\boldsymbol{x},\ell_\infty)} -\log\sigma(y\cdot\langle\boldsymbol{w},\boldsymbol{x}'\rangle)\right], \tag{1}$$

where μ represents the underlying data distribution. **Write down the pseudocode of PGD-based adversarial training algorithm.** The input of the algorithm should be a set of m training examples $\{(\boldsymbol{x}_i, y_i)\}_{i\in[m]}$ sampled from μ and all the necessary hyperparameters, such as perturbation budget ϵ, attack step size α, number of attack steps S, learning rate η, number of training epochs T, batch size B and etc. The output should be a weight vector $\widehat{\boldsymbol{w}}$ that is supposed to be a good solution to problem (1).

SOLUTION:

1. Initialization:

$w_{\text{hat}} =$ initialize_weights() // Random or zero initialization

2. Training Loop: Before the training loop create the train loader and test loader and assign the Batch size to B.

Iterate over training epochs (T). For each epoch, iterate through batches of the data.

for epoch in 1 to T :
  for batch in iterate_batches(data, batch_size):
    batch_gradients = 0 // Initialize gradients for the current batch
    for (x_i, y_i) of m training examples in batch:

3. PGD Attack to Generate Adversarial Examples:

For each example in the batch, generate an adversarial example (x_{adv}) using Projected Gradient Descent (PGD).

PGD involves iteratively perturbing the input within a specified perturbation budget (ϵ) while maximizing the logistic regression loss. Where 'S' is the number of iterations and α is step size

// Generate adversarial example using PGD
$x_{\text{adv}} = \text{PGD\_attack}(x_i, y_i, w_{\text{hat}}, \epsilon, \alpha, S)$

4. Compute Gradient of Adversarial Example:

Calculate the gradient of the logistic loss with respect to the weights (w_{hat}) using the generated adversarial example (x_{adv}) and the corresponding label (y_i)

// Compute gradient of the adversarial example
$\text{gradient} = \text{compute\_gradient}(x_{\text{adv}}, y_i, w_{\text{hat}})$

5. Accumulate Gradients:

Accumulate the gradients calculated for each example in the current batch. This is done to update the weights based on the average gradient over the entire batch.

// Accumulate gradients
$\text{batch\_gradients} += \text{gradient}$

6. Update Weights using Stochastic Gradient Descent:

Update the weights (w_{hat}) using stochastic gradient descent. Adjust the weights in the opposite direction of the accumulated gradients to minimize the logistic loss.

// Update weights using stochastic gradient descent
$$w_{\text{hat}} = w_{\text{hat}} - \text{learning\_rate} \times \left(\frac{1}{\text{batch\_size}}\right) \times \text{batch\_gradients}$$

7. Return Trained Weight Vector:

After completing the training epochs, return the trained weight vector (w_{hat}), which is expected to provide a solution to the logistic regression problem.

return w_{hat}

8. PGD Attack Function:

Implement the PGD attack to iteratively perturb the input within the specified perturbation budget (ϵ).

This function returns the final perturbed input (x_{adv}).

function PGD_attack$(x, y, w, \epsilon, \alpha, S)$:
// PGD attack to generate an adversarial example
$x_{\text{adv}} = x$ // Initialize adversarial example as the original input
for step in 1 to S :
// Compute gradient of the loss with respect to the input
gradient = compute_gradient(x_{adv}, y, w)
// Perturb the input in the direction of the gradient
$x_{\text{adv}} = x_{\text{adv}} + \alpha \times \text{sign}(\text{gradient})$
// Project the perturbed input back to the epsilon-ball around the original input
$x_{\text{adv}} = \text{clip\_to\_epsilon\_ball}(x_{\text{adv}}, x, \epsilon)$
return x_{adv}

9. Compute Gradient Function:

Calculate the gradient of the logistic loss with respect to the weights. This is used during both the adversarial example generation and weight update steps.

function compute_gradient(x, y, w) :
// Compute gradient of the logistic loss with respect to the weights
gradient $= -y \times \sigma(-y \times \langle w, x \rangle) \times x$
return gradient

10. Clip to Epsilon-Ball Function:

Ensure that the perturbed input (x_{adv}) remains within an epsilon-ball around the original input (x). This step prevents the perturbations from exceeding the specified budget (ϵ).

function clip_to_epsilon_ball$(x_{\text{adv}}, x, \epsilon)$:
// Clip the perturbed input to an epsilon-ball around the original input
perturbation $= x_{\text{adv}} - x$
clipped_perturbation = clip(perturbation, $-\epsilon, \epsilon$)
$x_{\text{adv}} = x +$ clipped_perturbation
return x_{adv}

Problem 2 (10 pts) Suppose f is a two-class linear classifier with parameters $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, where $f(\boldsymbol{x}) = \text{sgn}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)$ for any $\boldsymbol{x} \in \mathbb{R}^d$ and $\text{sgn}(\cdot)$ is the sign function. Suppose g represents the smoothed version of f used for randomized smoothing. Specifically for any $\boldsymbol{x} \in \mathbb{R}^d$, g is defined as:

$$g(\boldsymbol{x}) = \underset{j \in \{-1,+1\}}{\text{argmax}} \ \mathbb{P}_{\boldsymbol{\delta}}\Big[f(\boldsymbol{x} + \boldsymbol{\delta}) = j\Big], \text{ where } \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}),$$

where $\sigma > 0$ is the smoothing parameter. Prove that the following two statements:

1. (5 pts) For any input $\boldsymbol{x} \in \mathbb{R}^d$, $g(\boldsymbol{x}) = f(\boldsymbol{x})$.
2. (5 pts) For any input $\boldsymbol{x} \in \mathbb{R}^d$, the certified radius of g at $\boldsymbol{x}$ with ℓ_2-norm is $R(\boldsymbol{x}) = \frac{|\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b|}{\|\boldsymbol{w}\|_2}$. In other words, $g(\boldsymbol{x} + \boldsymbol{\delta})$ remains the same for any $\boldsymbol{\delta} \in \mathbb{R}^d$ with $\|\boldsymbol{\delta}\|_2 \leq R(\boldsymbol{x})$.

Solution:

Statement 1:

Proof:

The smoothed version $g(x)$ is defined as the argument that maximizes the probability of the original classifier $f(x)$ having a specific sign under the perturbation $\delta \sim \mathcal{N}(0, \sigma^2 I)$. Let's consider the case when $j = +1$ (similar arguments can be made for $j = -1$):

$$g(x) = \arg \max_{j \in \{-1,+1\}} P_\delta[f(x + \delta) = j]$$

$$g(x) = \arg \max_{j \in \{-1,+1\}} P_\delta[\text{sgn}(\langle w, x + \delta \rangle + b) = j]$$

$$g(x) = \arg \max_{j \in \{-1,+1\}} P_\delta[\text{sgn}(\langle w, x \rangle + \langle w, \delta \rangle + b) = j]$$

Since $\langle w, \delta \rangle$ is a random variable with mean 0 (due to $\delta \sim \mathcal{N}(0, \sigma^2 I)$), we can expect that, on average, it does not significantly change the sign of $\langle w, x \rangle + b$. Therefore, the argument that maximizes the probability is likely to be the same as the sign of $\langle w, x \rangle + b$.

Thus, $g(x)$ will be $+1$ if $\langle w, x \rangle + b > 0$ and -1 otherwise, which is exactly the same as the classification given by $f(x)$.

Therefore, for any input $x \in \mathbb{R}^d$, $g(x) = f(x)$, and Statement 1 is proved.

Statement 2:

Proof:

1. **Certified Radius** $R(x)$**:** The certified radius $R(x)$ is defined as

$$R(x) = \frac{|\langle w, x \rangle + b|}{\|w\|_2}$$

This is the distance from the decision boundary defined by $f(x) = 0$ to the point x, normalized by the l_2-norm of w.

2. **Stability of** g **within** $R(x)$**:** We want to show that $g(x + \delta)$ remains the same for any $\delta \in \mathbb{R}^d$ with $\|\delta\|_2 \leq R(x)$.

Considering the case when $f(x) = +1$, we need to maximize $P_\delta[f(x+\delta) = +1]$. This requires $\langle w, x+\delta \rangle + b > 0$.

Given $\|\delta\|_2 \leq R(x)$, we express $x + \delta$ as $x + \delta = x + \frac{R(x)}{\|w\|_2} \cdot \frac{\delta}{\|\delta\|_2}$, where the second term is the normalized vector in the direction of δ.

Now, consider:

$$\langle w, x+\delta\rangle + b = \langle w, x\rangle + b + \frac{R(x)}{\|w\|_2} \cdot \langle w, \frac{\delta}{\|\delta\|_2}\rangle$$

Since $\langle w, x\rangle + b > 0$ and the second term is bounded, $\langle w, x+\delta\rangle + b$ remains positive.

Therefore, $g(x+\delta) = +1$ for any $\delta \in \mathbb{R}^d$ with $\|\delta\|_2 \leq R(x)$.

Similar arguments can be made for the case when $f(x) = -1$.

Thus, we have shown that for any input $x \in \mathbb{R}^d$, the certified radius of g at x with $_2$-norm is

$$R(x) = \frac{|\langle w, x\rangle + b|}{\|w\|_2}$$

and $g(x+\delta)$ remains the same for any $\delta \in \mathbb{R}^d$ with $\|\delta\|_2 \leq R(x)$.

Implementation Problem. Below is an implementation problem that you need to complete the provided Jupyter notebook. More specifically, you will need to implement both standard training and PGD-based adversarial training using the MNIST handwritten digits dataset. Then, you will evaluate the learned models against different adversarial attacks.

Problem 3 (20 pts) Consider the classification task on MNIST and ℓ_∞ perturbations with $\epsilon = 0.1$. Suppose we want to train a CNN model using standard deep learning and a robust CNN model using PGD-based adversarial training. The CNN architecture has 4 convolutional layers and 2 MLP layers. We use a SGD optimizer with learning rate 0.1 for both methods, and both models are trained for 5 epochs.

Your Task: Complete the corresponding functions in the provided Jupyter notebook. Report the test classification errors of both standard-trained and adversarially-trained models in terms of no attack (clean), FGSM attack and PGD attack, respectively, and discuss the results.

Problem 4 (bonus, 10 pts) Your task is to develop a function my_attack() that can produce more adversarial examples for MNIST testing examples (i.e., a higher attack success rate) in the Jupyter notebook. Your attack should be valid in the sense that it only produces adversarial perturbations within the ℓ_∞-norm ball with $\epsilon = 0.1$. Also, you need to briefly describe how your attack methodology is developed and what is the attack success rate (ASR) it can achieve.

End of Homework Assignment 2 (PDF part)
Don't forget to also complete and submit the Jupyter notebook!

References