CS 521: Trustworthy AI Systems
MP1

Adharsh Kamath Raghupathi (netID: ak128)

Problem 1:

Part (1)

Given the original input x, the target class t, and the perturbation magnitude eps, we can compute FGSM adversary as follows:

```
adv_x = x - eps * x.grad.sign()
print("Adversarial Example: ", adv_x)
```

With this snippet (in the file https://github.com/adharshkamath/cs521-sp25/blob/main/fgsm.py), we can see the following output:

```
Original Input:  tensor([[0.6584, 0.8175, 0.1262, 0.0541, 0.2268, 0.8405, 0.5393, 0.9798, 0.8597,
         0.5977]], requires_grad=True)
Original Class:  2
Adversarial Example:  tensor([[ 0.1584,  1.3175,  0.6262, -0.4459,  0.7268,  0.3405,  0.0393,  0.4798,
          1.3597,  0.0977]], grad_fn=<SubBackward0>)
Adversarial Class:  0
tensor(0.5000)
```

As we can see, the FGSM attack found an adverserial example that is close enough to the original input. but is classified into a different class.

Part (2)

The original value of epsilon (0.5) did not work for the FGSM attack, for target class 1. I tried different values of epsilon but that did not help. So I tried the Iterative FGSM attack. But I had to increase the epsilon to 0.85, and use alpha = 0.25 with max iterations set to 10. The relevant code snippet is as follows:

```
epsReal = 0.85
eps = epsReal - 1e-7
alpha = 0.25
iterations = 10

for iteration in range(iterations):
    adv_x.requires_grad_(True)
    N.zero_grad()
    loss = L(N(adv_x), torch.tensor([t], dtype=torch.long))
    loss.backward()
    with torch.no_grad():
        adv_x = adv_x - (alpha * adv_x.grad.sign())
        delta = adv_x - x
        delta = torch.clamp(delta, -eps, eps) # clamp to eps ball
        adv_x = x + delta
```

```
new_class = N(adv_x).argmax(dim=1).item()
distance = torch.norm((x-adv_x), p=float('inf')).data
print(f"Iteration {iteration}: Distance {distance}")
if new_class == t:
    print("Attack successful")
    break
```

The rest of the code can be found in the script here:
https://github.com/adharshkamath/cs521-sp25/blob/main/fgsm.py

Problem 2:

Part (1)

Thy Google colab notebook with the code can be found here.

Below are the standard and robust accuracies of the three models (L_∞ attack with eps=8/255 and L_2 attack with eps = 0.75)

- pretr_Linf.pth: Standard= 82.80%, L_2 robustness=45.96%, L_∞ robustness = 50.65%
- pretr_L2.pth: Standard=88.75%, L_2 robustness=53.25%, L_∞ robustness = 29.15%,
- pretr_RAMP.pth: Standard=81.19%, L_2 robustness=59.25%, L_∞ robustness = 48.93%,

From the numbers we can see that L_2 robustness training does not hold up well against the L_∞ PGD attack. The model with L_∞ robustness training holds up well against both L_2 and L_∞ attacks. The model with RAMP training holds up well against both the attacks but has a slightly lower standard accuracy compared to the other two.

Part (2)

The code for this is in the same Google colab notebook linked above. Below are the accuracies of the of the networks for multi-norm robustness:

- pretr_Linf.pth: 47.05%
- pretr_L2.pth: 30.85%
- pretr_RAMP.pth: 49.76%

From the accuracies we can see that RAMP trained model performs best, followed by the model that is L_∞ robustness trained. Finally L_2 robustness trained model does not perform as great as either of the two. So, RAMP training helps the most in this case.

Problem 3:

The paper introduces a range of new unforseen adversaries, including 18 new non-L_P attacks. They illustrate the point that it is not always possible to anticipate the full range of attacks that a model might be subjected to in the real world. They propose a new framework, ImageNet-UA, that aims to evaluate models against unforseen adversaries. They show that existing models are lacking, when evaluated using this framework.