HW 1

Haolin Liu

The simple neural network defined in the boilerplate code is not modified.

Loss function of choice is nn.CrossEntropyLoss().

Optimizer of choice is optim.SGD().

Problem 1:

The attack is implemented like as specified in lecture slides.

def fgsm(model, x, y, eps):

    x = x.to(device)

    y = y.to(device)

    x.requires\_grad = True

    outputs = model(x)

    model.zero\_grad()

    loss = loss\_func(outputs, y).to(device)

    loss.backward()

    x\_prime = x + eps \* x.grad.sign()

    return x\_prime

def pgd\_untargeted(model, x, y, eps\_step, k=30, eps=0.3):

    model.eval()

    x = x.to(device)

    y = y.to(device)

    ori\_x = x.data

    for i in range(k) :

        adv\_x = fgsm(model, x, y, eps\_step)

        eta = torch.clamp(adv\_x - ori\_x, min=-eps, max=eps)

        x = torch.clamp(ori\_x + eta, min=0, max=1).detach\_()

    return x

In FGSM, first we move the tensors to the GPU. The gradient is then computed, and the image is perturbed by moving the gradient in the direction that causes it to get larger. In PGD, FGSM are performed for multiple iterations. After each, we project the resulting image and clamp it within the range specified by eps.

The model, when trained using standard methods, has the following performance.

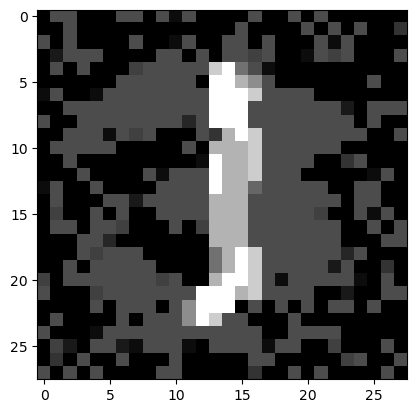
Standard accuracy: 92%

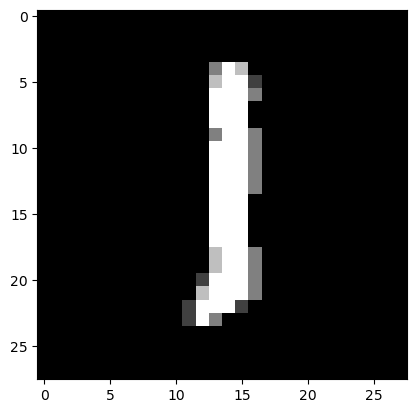
Robust accuracy: 0%

Robust accuracy under FGSM attack: up to 48%

This means that PGD is able to generate adversarial examples successfully for all possible input.

Since the adversarial examples are generated with the fully trained standard model, accuracy on the adversarial examples’ sets are also 0.

Example, randomly picked, as follows:

Here we can see an image of 1 and the adversarial example of this image. Semantic content, I believe, is mostly preserved in the adversarial image (that we can still recognize the one). Most of the modifications are in the background. We can see that a lot of pixel chunks became lighted up.

Problem 2:

In this problem PGD-based adversarial training is performed. This is done through a simple modification to the train\_model() function.

if enable\_defense:

# Generate adversarial examples using PGD

      adv\_inputs = pgd\_untargeted(model, inputs, labels, eps)

      # have the model train on these instead

      inputs = torch.cat([inputs, adv\_inputs])

      labels = torch.cat([labels, labels])

      # put the model back in training mode

      model.train()

As specified, adversarial examples are mixed into normal inputs during the training process.

PGD-trained standard accuracy: 87%

PGD-trained robust accuracy: up to 12%

PGD-trained accuracy under FGSM attack: up to 67%

When attacked by PGD, model trained this way still suffered very significant accuracy loss, although accuracy is no longer reduced to 0.

When model trained this way is attacked by FGSM, it has robust accuracy of up to 67%. This accuracy decreases as epsilon increases, however, very much unlike the situation with PGD attack. Compared to regular model’s performance, adversarial trained model obtained quite a big accuracy improvement. This is likely because after adversarial training, the model’s decision space for a given class has been modified sufficiently that it’s unlikely for a single FGSM iteration to find an adversarial example within that space.

Problem 3: