Adversarial Image generation and Misclassification Project Report

Ishita Ghosh (ighosh4)

To create adversarial images, I used the strategy outlined in the following tutorial:
[Tricking Neural Networks: Create your own Adversarial Examples]

In this strategy we require 2 inputs: the input image and a goal label. Our task is to generate an image that is as close to the input image as possible while at the same time getting classified as the goal label.

According to the assignment instructions, we randomly sample 10 images from the dataset, one from each class, and assign the goal label as the next label (e.g. the goal label for the image of the number zero will be 1 and so on).

I first generated a random image x using the normal distribution:

```
x = np.random.normal(.5, .3, (784, 1))
```

Then I use the following update equation to iteratively update the value of x at each step:

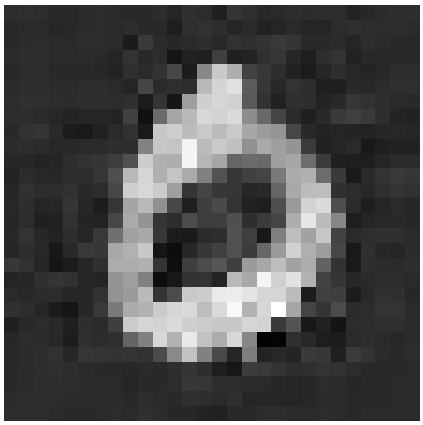
```
x -= eta * (d + lam * (x - x_target))
```

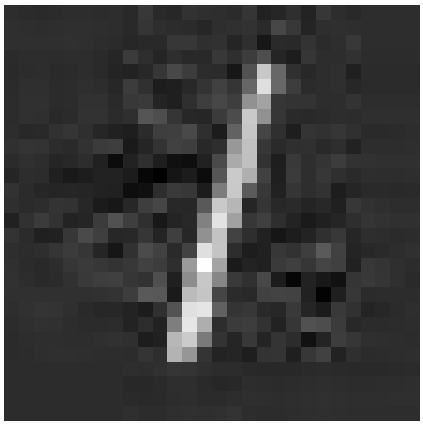
Where eta is the step size, lam is the regularization parameter (weight of the target image x_target) and d is the gradient of the loss w.r.t. x. We run it for 1000 iterations or steps.

In our experiments we set eta=0.1, lam=0.05, steps=1000.

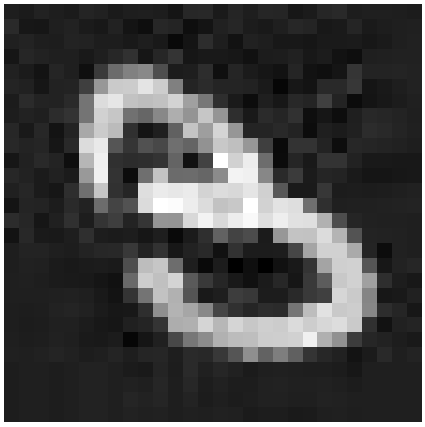
Results

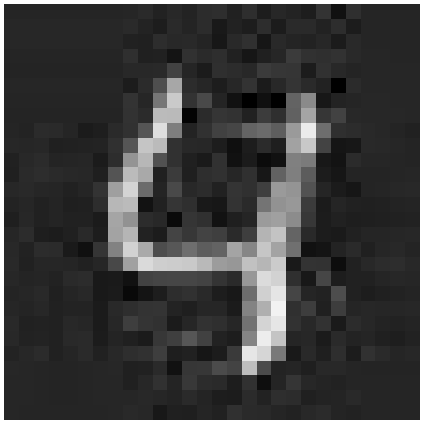
Below are the 10 adversarial images I have generated. In each case the image was misclassified as the next image (e.g. 1 was classified as 2).

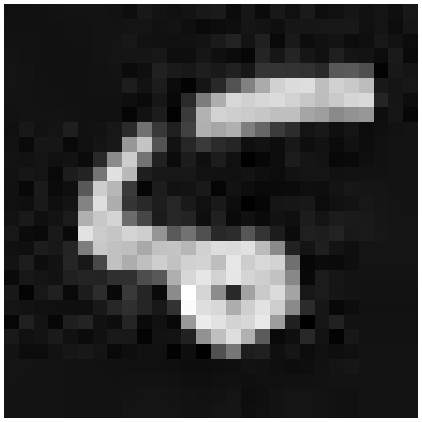

Correct Label = 0. Predicted label = 1.

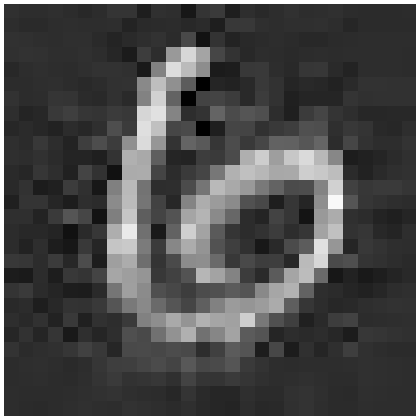

Correct Label = 1. Predicted label = 2.


Correct Label = 2. Predicted label = 3.


Correct Label = 3. Predicted label = 4.

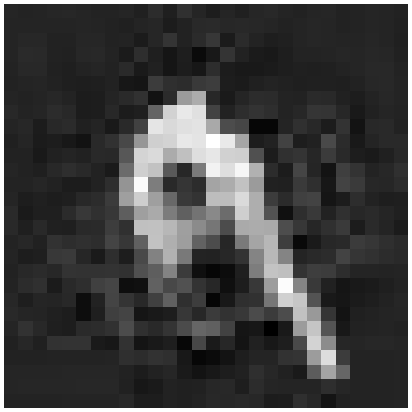

Correct Label = 4. Predicted label = 5.


Correct Label = 5. Predicted label = 6.


Correct Label = 6. Predicted label = 7.


Correct Label = 7. Predicted label = 8.


Correct Label = 8. Predicted label = 9.

 Correct Label = 9. Predicted label = 10.

In each case we see that even though the correct label is self-evident to the human eye our classifier misclassified it as the next label. This is because of the small amount of white noise that is present in each image that tricks the classifier into classifying it as the desired wrong label.

Appendix

The full source code is in the HW8 report. These are only the functions used to generate the adversarial images once you have the trained network from HW8.

```
def adversarial_example(x_target, label, eta=0.1, lam=0.05,
steps=1000):
    # Create a random image to initialize gradient descent
with
    x = torch.tensor(np.random.normal(.5, .3, (1, 1, 28,
28)), dtype=torch.float, requires_grad=True)
    # x = x_target
    # x = x.type(torch.FloatTensor)
    label = torch.tensor(label)
    label = torch.unsqueeze(label, 0)
    optimizer = optim.SGD(network.parameters(),
```

```
lr=learning_rate,
                                momentum=momentum)
    # Gradient descent on the input
    network.eval()
    for i in trange(steps):
        # Calculate the derivative
        optimizer.zero_grad()
        if x.grad is not None:
            x.grad.zero_()
        output = network(x)
        loss = F.nll_loss(output, label)
        x.retain_grad()
        loss.backward(retain_graph=True)
        tqdm.write(f"Loss = {loss}")
        d = x.grad
        # The GD update on x, with an added penalty
        # to the cost function
        # ONLY CHANGE IS RIGHT HERE!!!
        if d is not None:
            x = x - eta * (d + lam * (x - x_target))
    return x


def generate_adversarials():
    adversarial_dict = {}
    for i in range(10):
        target_idx, target_image = get_image(i)
        if i==9:
            label = 0
        else:
            label = i + 1
        print(f"Generating Adversarial Example for label =
```

```
{label}")
        adversarial_image =
adversarial_example(target_image, label)
        adversarial_dict[i] = adversarial_image
    return adversarial_dict
```