ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

# CS498
# Applied Machine Learning
## Assignment #8

(Extra Homework for Four Hour version)

Students:
nidiaib2, Nidia Bucarelli
sunnyk2, Sunny Katiyar
wangx2, Wang Xiang

May 14, 2020
Spring 2020

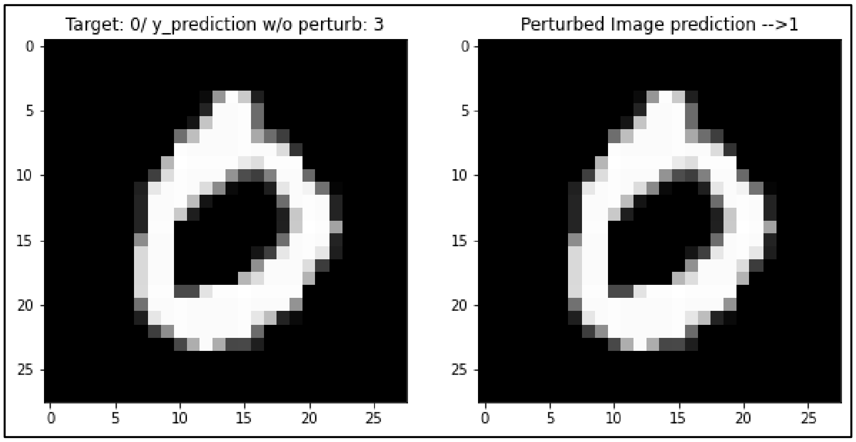
***Images were perturbed with the following script:***

```
def fgsm_attack(image, epsilon, data_grad):
    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon*sign_data_grad
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image
```

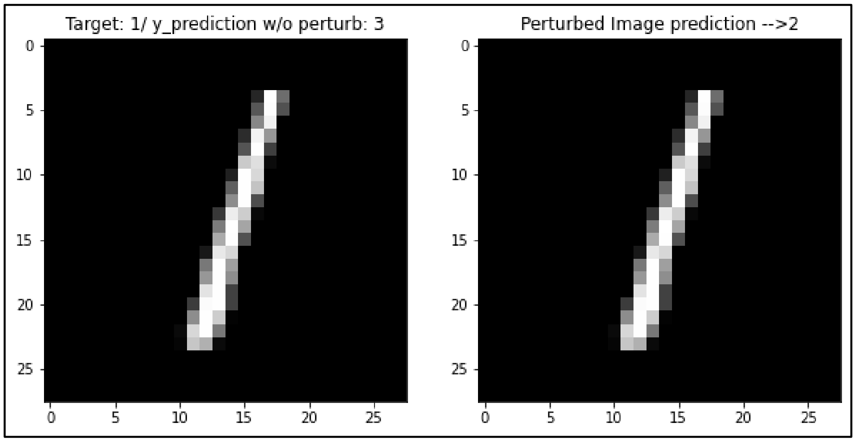
Accuracy of classifying the next digit: **99.3%**

***Example of outputs:***

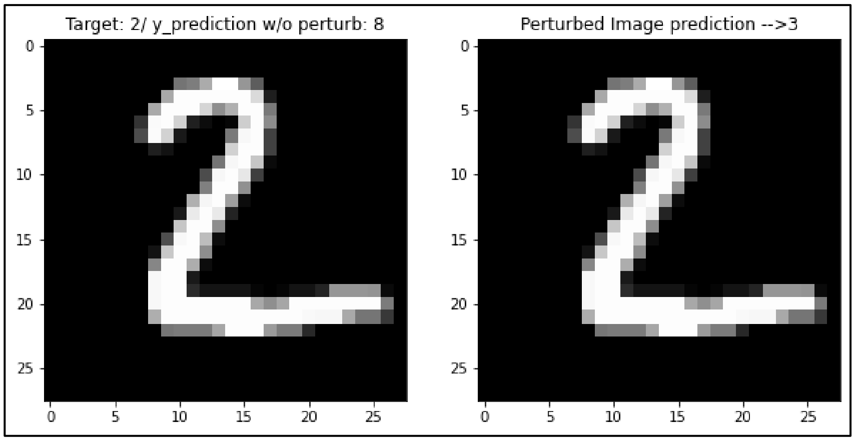
### Digit-0


Target: 0/ y_prediction w/o perturb: 3
Perturbed Image prediction -->1
0
5
10
15
20
25
0
5
10
15
20
25

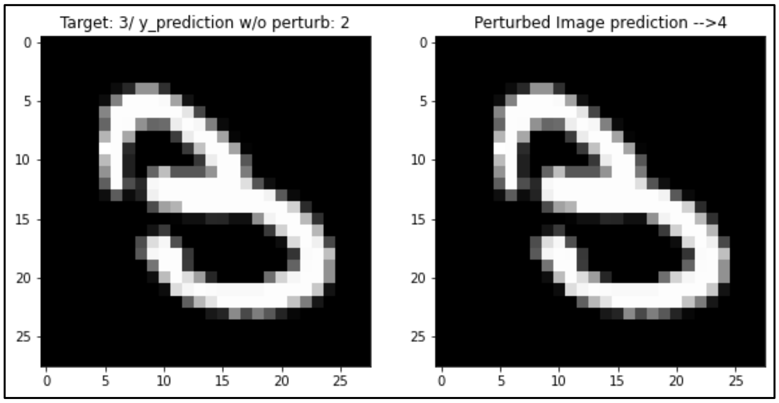
### Digit-1


Target: 1/ y_prediction w/o perturb: 3
Perturbed Image prediction -->2
0
5
10
15
20
25
0
5
10
15
20
25

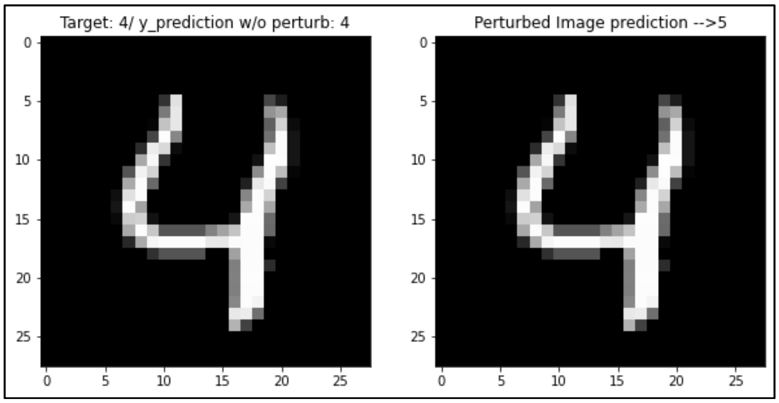
## Digit-2


Target: 2/ y_prediction w/o perturb: 8
Perturbed Image prediction -->3
0
5
10
15
20
25

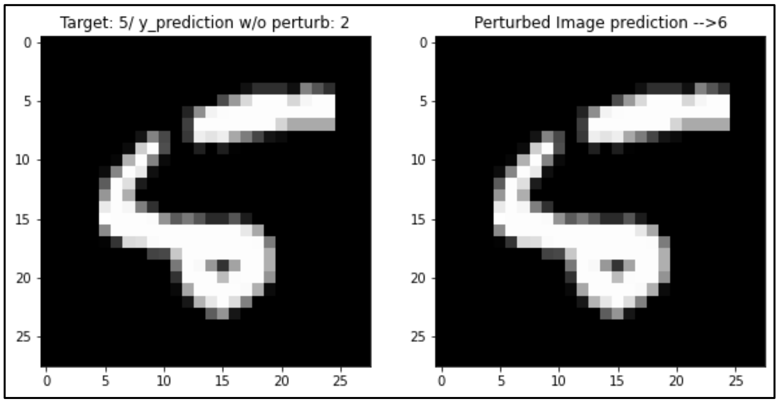
## Digit-3


Target: 3/ y_prediction w/o perturb: 2
Perturbed Image prediction -->4
0
5
10
15
20
25

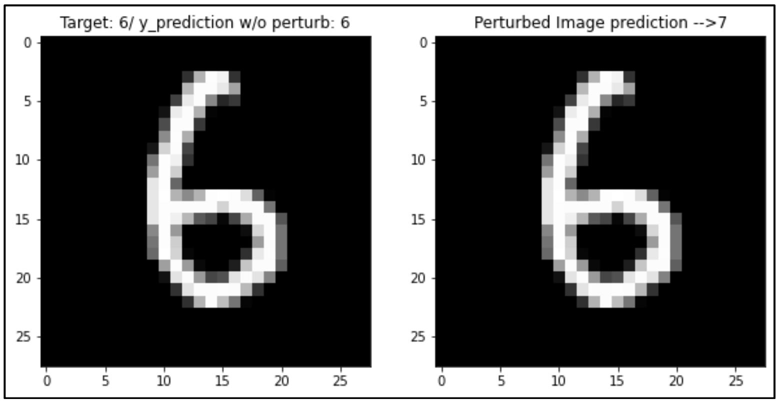
## Digit-4


Target: 4/ y_prediction w/o perturb: 4
Perturbed Image prediction -->5
0
5
10
15
20
25

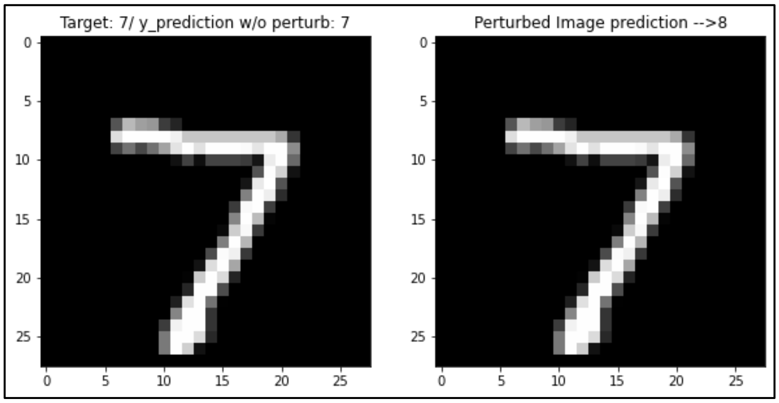
## Digit-5


Target: 5/ y_prediction w/o perturb: 2
Perturbed Image prediction -->6
0
5
10
15
20
25

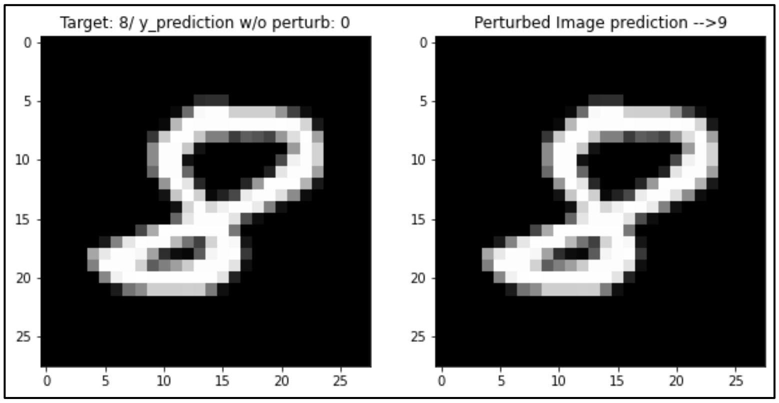
## Digit-6


Target: 6/ y_prediction w/o perturb: 6
Perturbed Image prediction -->7
0
5
10
15
20
25

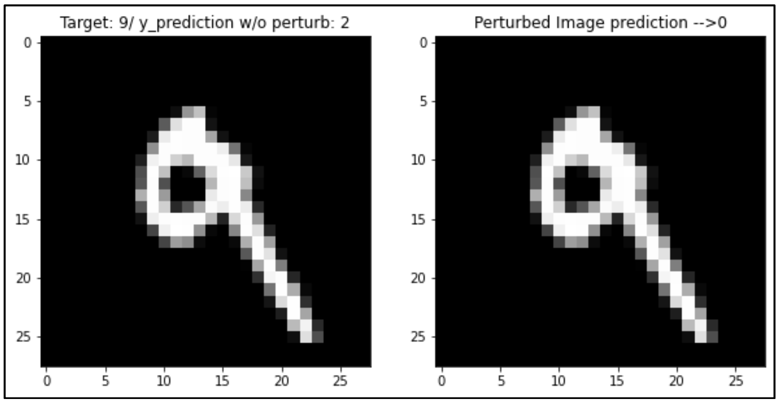
## Digit-7


Target: 7/ y_prediction w/o perturb: 7
Perturbed Image prediction -->8
0
5
10
15
20
25

## Digit-8


Target: 8/ y_prediction w/o perturb: 0
Perturbed Image prediction -->9
0
5
10
15
20
25
0
5
10
15
20
25

## Digit-9


Target: 9/ y_prediction w/o perturb: 2
Perturbed Image prediction -->0
0
5
10
15
20
25
0
5
10
15
20
25

# CS-498_Extra Homework

May 15, 2020

```
In [0]: import h5py
        import numpy as np
        from random import randint
        import time
        import requests
        import matplotlib.pyplot as plt
        import cv2
        # from sklearn.ensemble.forest import _generate_unsampled_indices
        # from sklearn.ensemble import BaggingClassifier
        from sklearn.ensemble import RandomForestClassifier
        import h5py
        import numpy as np
        from random import randint
        import torch
        import torch.nn as nn
        import torchvision
        import torchvision.transforms as transforms
        import torch.nn.functional as F
        import torch.optim as optim
        from torch.autograd import Variable
        import time
        import os

        ##Load MNIST DATA

        batch_size= 1
        train_dataset= torchvision.datasets.MNIST(root= './', train= True, transform= transfor
        test_dataset=torchvision.datasets.MNIST(root= './', transform= transforms.ToTensor(), 


        train_loader= torch.utils.data.DataLoader(dataset= train_dataset, batch_size=batch_size
        test_loader= torch.utils.data.DataLoader(dataset= test_dataset, batch_size=batch_size,
```

```
In [0]: os.chdir('/content/drive/My Drive/CS-498 Applied Machine Learning/HW8/Nidia')
        # pretrained_model= torch.load('/content/drive/My Drive/CS-498 Applied Machine Learnin
```

```
In [0]: targets= list(range(0,10))
        target_new= []
```

```
l= []
for t in range (0, len(targets)):
 for k, y in train_loader:
  if y == targets[t]:
    target_new.append(k)
    l.append(y)
    break
```

In [0]:

```
##MODEL ARCHITECTURE
class network(nn.Module):
  def __init__(self):
    super(network, self).__init__()
    self.conv1 = nn.Conv2d(1,20,5,stride=1,padding=0)
    # self.dropout1 = nn.Dropout(0.30)
    self.MaxPool1= nn.MaxPool2d(2, stride=2)


    self.conv2 = nn.Conv2d(20,50,5,stride=1, padding=0)
    # self.dropout2 = nn.Dropout(0.30)
    self.MaxPool2= nn.MaxPool2d(2, stride=2)


    self.conv3 = nn.Conv2d(50,500,4,stride=1, padding=0)
    # self.conv3_bn= nn.BatchNorm2d(num_features=500,track_running_stats=False)

    self.conv4 = nn.Conv2d(500,10,1,stride=1, padding=0)
    # self.conv4_bn= nn.BatchNorm2d(num_features=10,track_running_stats=False)


    # self.dropoutfc= nn.Dropout(0.30)
    # self.fc3 = nn.Linear(500,10)

  def forward(self, x):
    # x= self.MaxPool1(self.dropout1(self.conv1(x)))
    # x= self.MaxPool2(self.dropout2(self.conv2(x)))

    x= self.MaxPool1((self.conv1(x)))
    x= self.MaxPool2((self.conv2(x)))
    # x= self.conv3_bn((self.conv3(x)))
    x= self.conv3(x)
    x= F.relu(x)
    x= self.conv4(x)
    # x= self.conv4_bn(x)

    return (x.flatten().reshape(x.shape[0],10))
```

In [0]:

```
def fgsm_attack(image, epsilon, data_grad):
  sign_data_grad = data_grad.sign()
```

```
    # Create the perturbed image by adjusting each pixel of the input image
    perturbed_image = image + epsilon*sign_data_grad
    # Adding clipping to maintain [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    # Return the perturbed image
    return perturbed_image
```

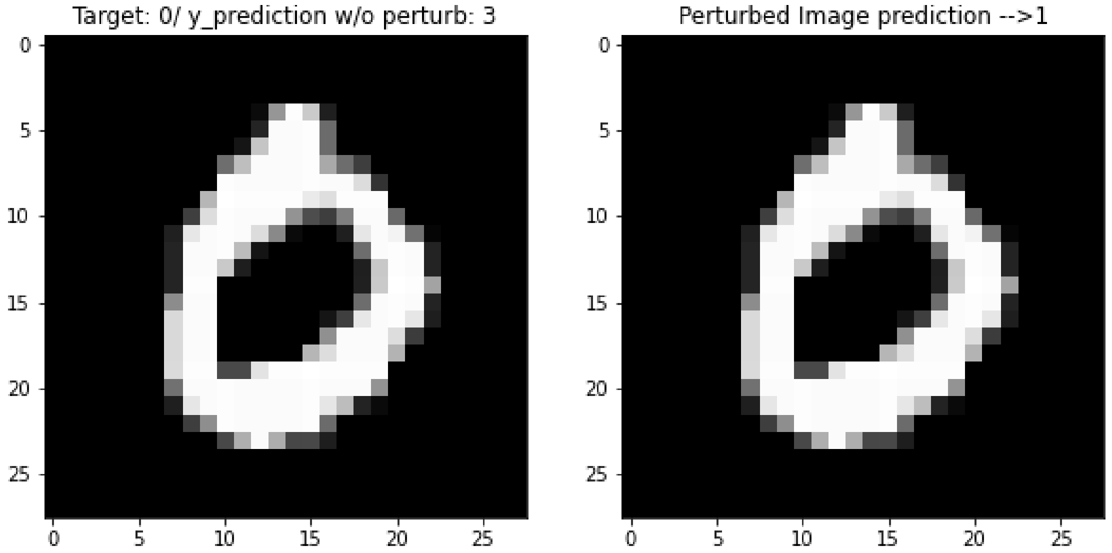
```
In [0]: device= torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        model = network().to(device)

        checkpoint = torch.load('/content/drive/My Drive/CS-498 Applied Machine Learning/HW8/N
        model.load_state_dict(checkpoint['state_dict'])
        model.eval()
```

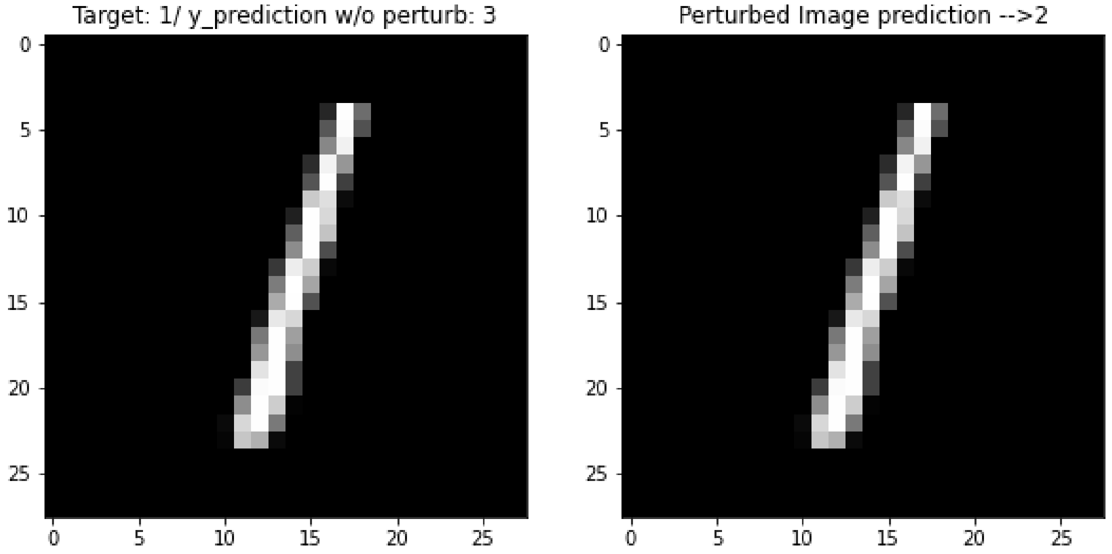
```
Out[0]: network(
          (conv1): Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1))
          (MaxPool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
          (conv2): Conv2d(20, 50, kernel_size=(5, 5), stride=(1, 1))
          (MaxPool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
          (conv3): Conv2d(50, 500, kernel_size=(4, 4), stride=(1, 1))
          (conv4): Conv2d(500, 10, kernel_size=(1, 1), stride=(1, 1))
        )
```

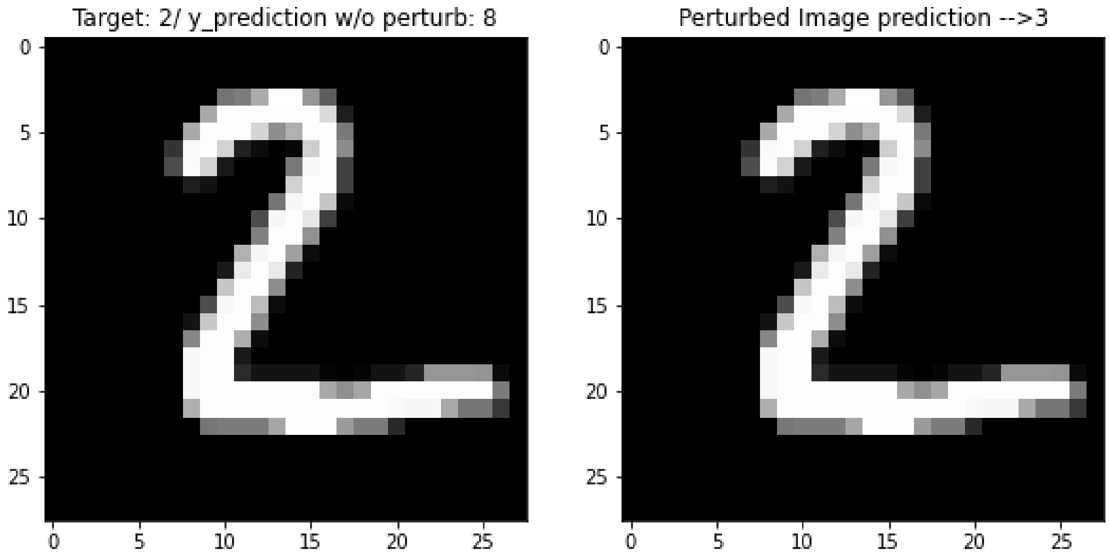
```
In [0]: criterion= nn.CrossEntropyLoss()
        learning_rate = 0.01
        optimizer= optim.SGD(model.parameters(), lr=learning_rate)
        adv_ex = []
        adv_examples= []
        num_epoch = 1
        # epsilons = [0, .05, .1, .15, .2, .25, .3]
        epsilons = 0.05
        num_epoch = 1
        for epoch in range (0, num_epoch):
          correct = 0
          adv_examples = []
          for data, target in test_loader:
              data, target = data.to(device), target.to(device)
              y_alternate = target + 1
              if y_alternate == 10:
                y_alternate = torch.tensor([0]).to(device)
              y_alternate= y_alternate.to(device)

              # Set requires_grad attribute of tensor. Important for Attack
              data.requires_grad = True

              # Forward pass the data through the model
              output = model(data)
              y_prediction= output.data.max(1)[1]
              # # If the initial prediction is wrong, dont bother attacking, just move on
              if y_prediction == y_alternate:
                  correct+= 1
                  continue

              # Calculate the loss
              loss = criterion(output, y_alternate)

              model.zero_grad()
              # Calculate gradients of model in backward pass
              loss.backward()
              optimizer.step()

              # Collect datagrad
              data_grad = data.grad.data

              perturbed_data = fgsm_attack(data, epsilon, data_grad)

              output = model(perturbed_data)
              # Check for success
              final_pred = output.max(1, keepdim=True)[1] # get the index of the max log-proba

              if final_pred == y_alternate:
                correct = correct + 1
                if len(adv_examples) < 100:
                    adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                    original= data.squeeze().detach().cpu().numpy()
                    adv_examples.append((target, y_prediction.item(), final_pred.item(), origi

          final_acc = correct/float(len(test_loader))
          print('epoch:', epoch, "Epsilon: {}\tTest Accuracy = {} / {} = {}".format(epsilon, c
```

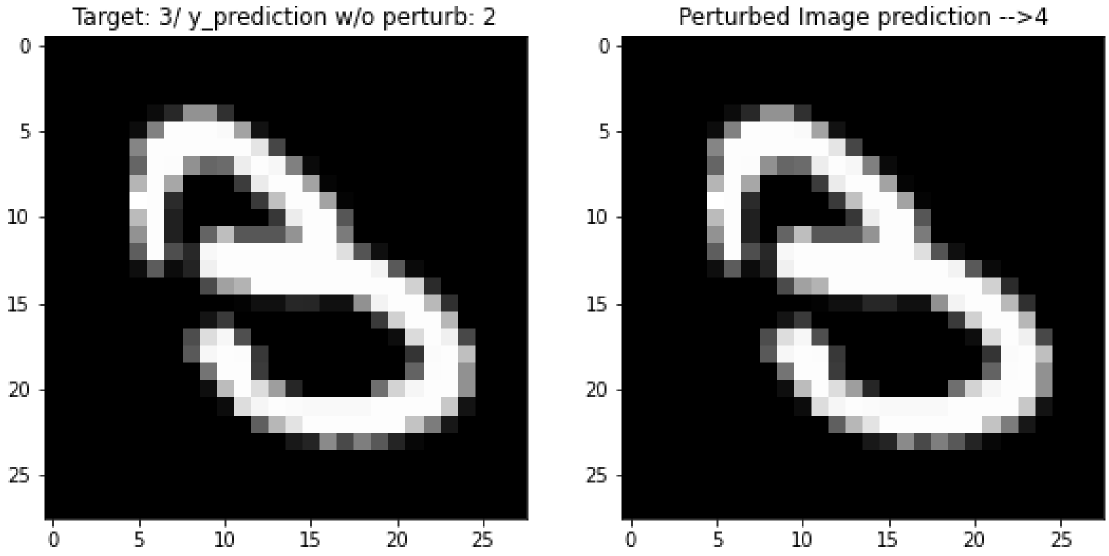
```
epoch: 0 Epsilon: 0.05        Test Accuracy = 9931 / 10000 = 0.9931
```

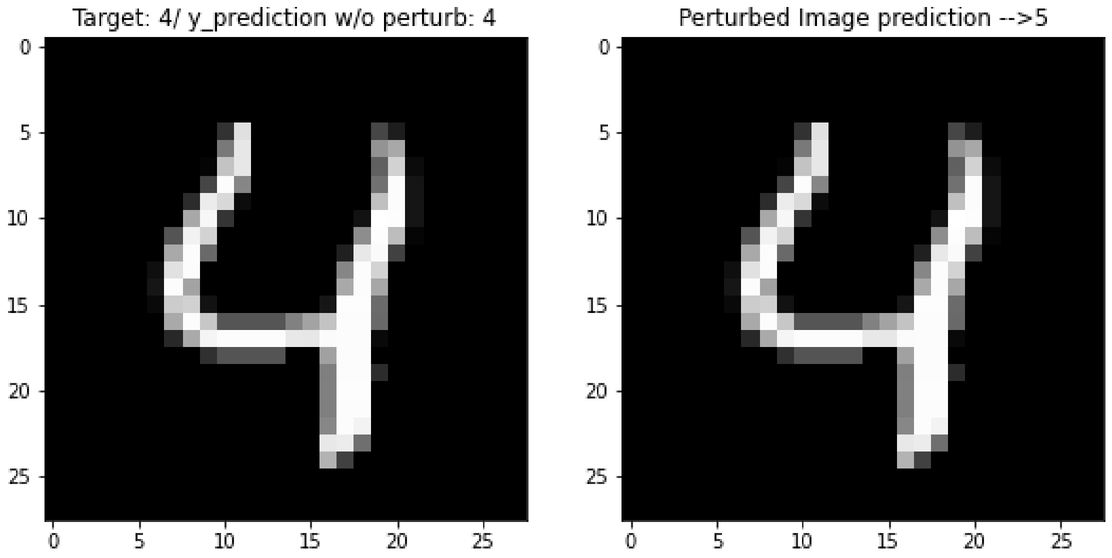
```
In [0]: for i in range (0,10):
            for j in range (0, len(adv_examples)):
               target,origp,adv,ex, origimg = adv_examples[j]
               if target == i:
                  fig= plt.figure(figsize=(10,10))
                  a = fig.add_subplot(1, 2, 1)
                  a.set_title('Target: ' + str(target.item()) +  '/ y_prediction w/o perturb:
                  imgplot = plt.imshow(ex, cmap= 'gray')

                  a = fig.add_subplot(1, 2, 2)
                  a.set_title('Perturbed Image prediction -->' + str(adv))
                  imgplot = plt.imshow(ex, cmap= 'gray')
                  plt.show()
                  break
```





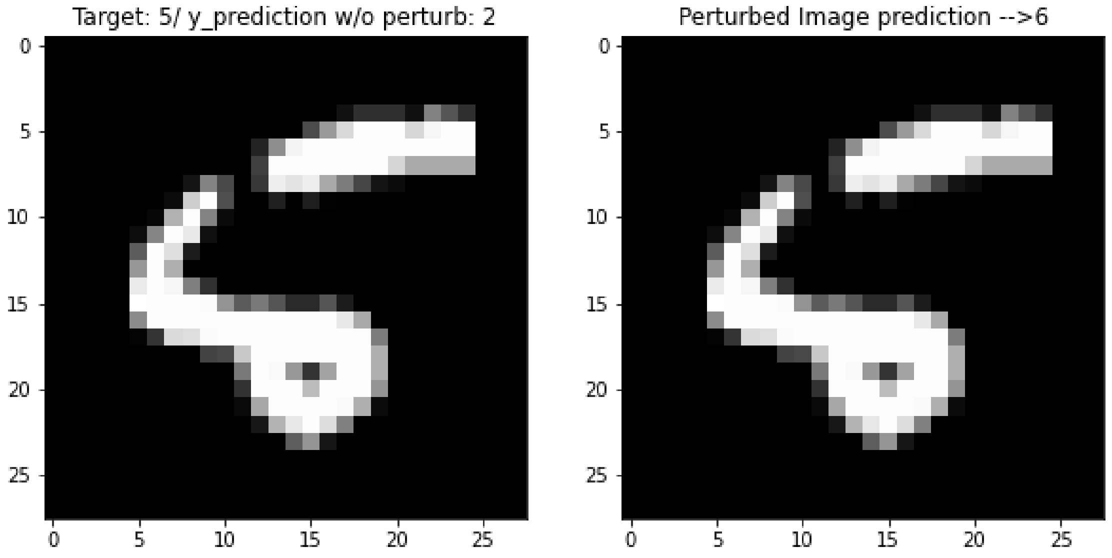

Target: 2/ y_prediction w/o perturb: 8
Perturbed Image prediction -->3
0
5
10
15
20
25


Target: 3/ y_prediction w/o perturb: 2
Perturbed Image prediction -->4
0
5
10
15
20
25



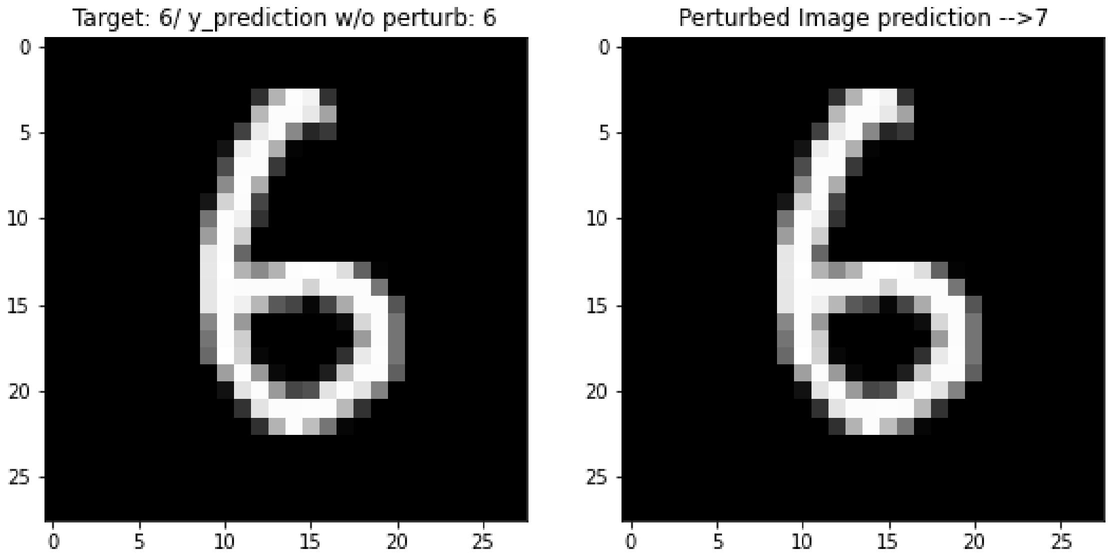
Target: 4/ y_prediction w/o perturb: 4

Perturbed Image prediction -->5



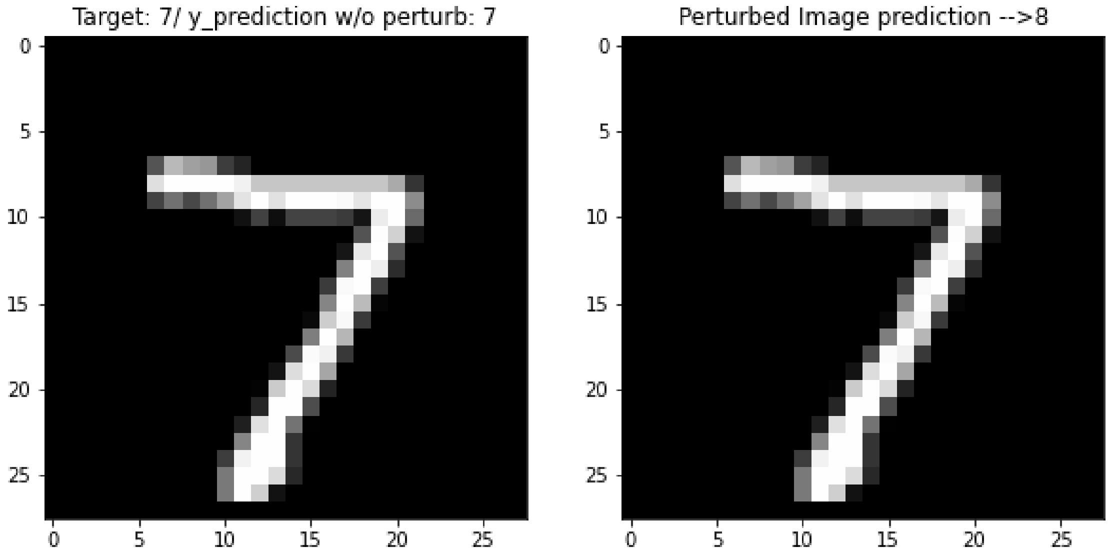
Target: 5/ y_prediction w/o perturb: 2

Perturbed Image prediction -->6



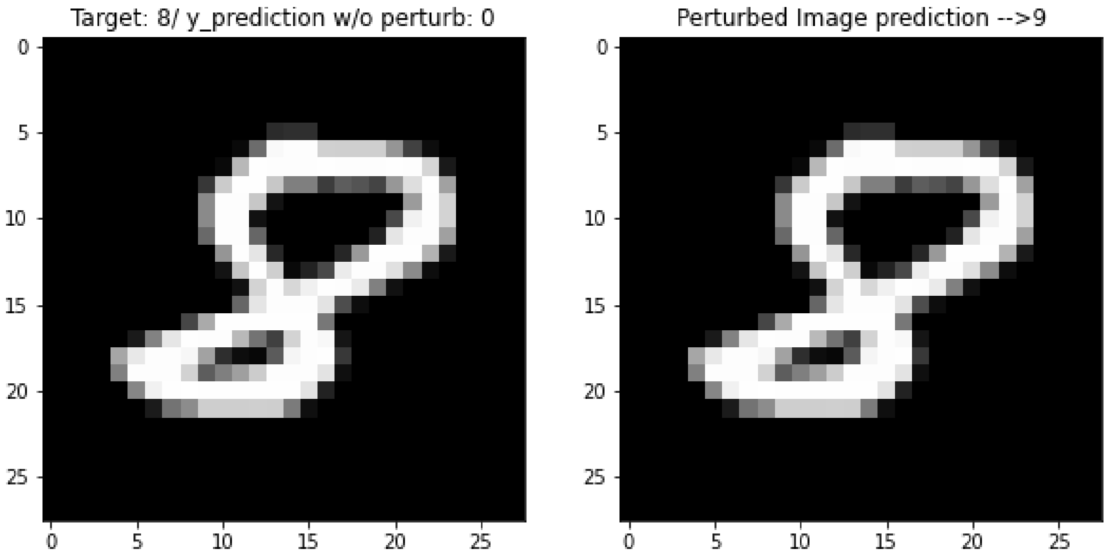
Target: 6/ y_prediction w/o perturb: 6

Perturbed Image prediction -->7



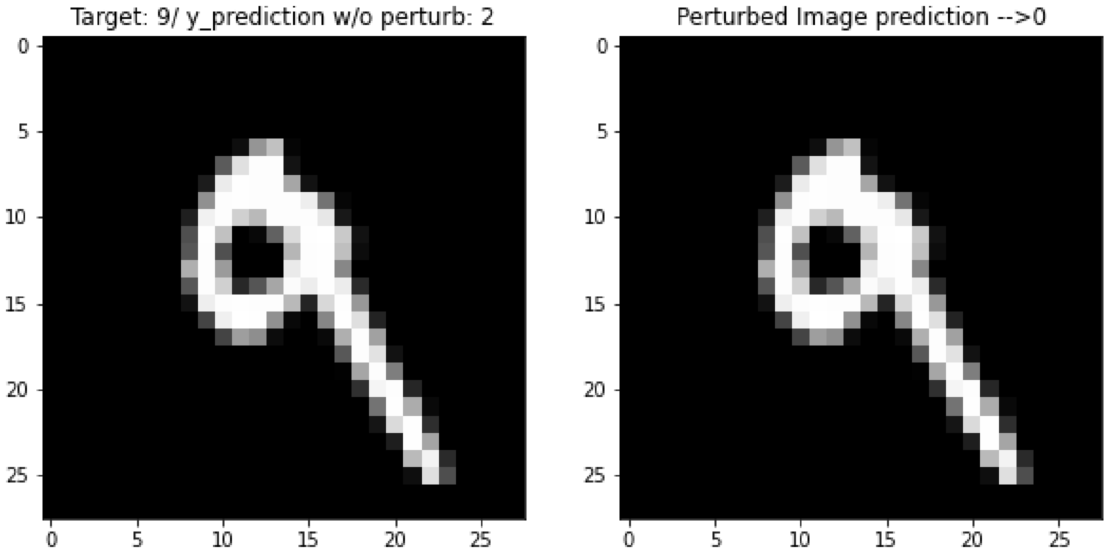
Target: 7/ y_prediction w/o perturb: 7

Perturbed Image prediction -->8



Target: 8/ y_prediction w/o perturb: 0

Perturbed Image prediction -->9



Target: 9/ y_prediction w/o perturb: 2

Perturbed Image prediction -->0

```
In [0]:
```