```
[2] import torch
    from torch import nn
    import torch.nn.functional as F
    from torchvision import datasets, transforms
    import matplotlib.pyplot as plt
    import numpy as np
    import torch.backends.cudnn as cudnn
    cudnn.benchmark = True  # fire on all cylinders
```

Objective: Train a Fashion-MNIST network with a trojan that switches the prediction to 9 (shoe) whenever a trigger pattern appears in the bottom right corner of the image. The trojan should not affect accuracy on unmodified images. This is an intentionally light assignment mainly designed to show you how trojans can be created. Make you can understand the code that you are not assigned to fill in!

Set up Clean Data

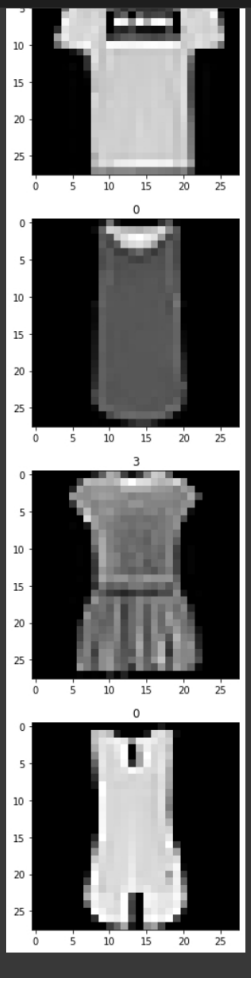
```
[3] train_data = datasets.FashionMNIST('./data', train=True, download=True, transform=transforms.ToTensor())
    test_data = datasets.FashionMNIST('./data', train=False, download=True, transform=transforms.ToTensor())
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz
Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
Extracting ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz
Extracting ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz
Extracting ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw
```

```
[4] print(len(train_data), len(test_data))
```

```
60000 10000
```

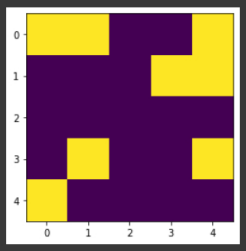
```
[5] # Visualize the clean data
    for i in range(5):
        plt.figure()
        plt.imshow(train_data[i][0].permute(1,2,0).repeat(1,1,3).numpy())
        plt.title(train_data[i][1])
        plt.show()
```



Set up Poisoned Data

```
def create_trigger(side_len):
    return (torch.rand(side_len, side_len) > 0.5).float()
```

```
[7] # This will be used for the remainder of the notebook.
    trigger = create_trigger(5)

    plt.figure()
    plt.imshow(trigger)
    plt.show()
```




```
[8] def insert_trigger(images, pattern):
        """
        :param images: A tensor with values between 0 and 1 and shape [N, 1, height, width]
        :param pattern: A tensor with values between 0 and 1 and shape [side_len, side_len]
        :returns: modified images with pattern pasted into the bottom right corner
        """
        side_len = pattern.shape[0]
        ##########################################################################
        # TODO: insert pattern in the bottom right corner                        #
        ##########################################################################


        images[:,:,-side_len:,-side_len:] = pattern


        ##########################################################################
        #                           END OF YOUR CODE                             #
        ##########################################################################

        return images
```

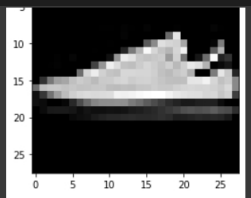
```
[ ]
```

Working with datasets is really easy in Pytorch! It mainly involves the `torch.utils.data.Dataset` and `torch.utils.data.Dataloader` classes. If you're working on research, you'll likely use these again. So we encourage you to reread your code when you're done or explore the ways Pytorch helps manage data in their documentation.

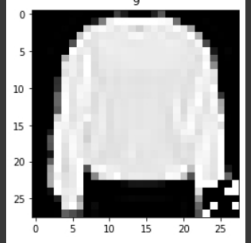
```
class PoisonedDataset(torch.utils.data.Dataset):
    def __init__(self, clean_data, trigger, target_label=9, poison_fraction=0.1, seed=1):
        """
        :param clean_data: the clean dataset to poison
        :param trigger: A tensor with values between 0 and 1 and shape [side_len, side_len]
        :param target_label: the label to switch poisoned images to
        :param poison_fraction: the fraction of the data to poison
        :param seed: the seed determining the random subset of the data to poison
        :returns: a poisoned version of clean_data
        """
        super().__init__()
        self.clean_data = clean_data
        self.trigger = trigger
        self.target_label = target_label

        # select indices to poison
        num_to_poison = np.floor(poison_fraction * len(clean_data)).astype(np.int32)
        rng = np.random.default_rng(seed)
        self.poisoned_indices = rng.choice(len(clean_data), size=num_to_poison, replace=False)


    def __getitem__(self, idx):
        ##########################################################################
        # TODO: Check if idx should be poisoned.                                 #
        # If so, return the image with a trigger and the target label.           #
        # If not, return the clean image and the original label.                 #
        # Hint: You might find torch's squeeze and unsqueeze methods useful      #
        ##########################################################################
        side_len = len(trigger[0])


        image  = self.clean_data.__getitem__(idx)[0][None, :]
        true_label = self.clean_data.__getitem__(idx)[1]

        poisoned_image = insert_trigger(image, self.trigger)

        if idx in self.poisoned_indices:
          return (poisoned_image[0], 9)
        else:
          return (self.clean_data.__getitem__(idx)[0][None, :][0], true_label)




        ##########################################################################
        #                           END OF YOUR CODE                             #
        ##########################################################################

        pass

    def __len__(self):
```

```
            return len(self.clean_data)
```

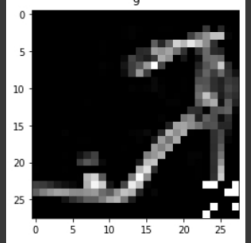
```
[25] # Visualize the poisoned data
    poisoned_train_data = PoisonedDataset(train_data, trigger, poison_fraction=0.5)
    ################################################################################
    # TODO: plot the first 10 images from poisoned_train_data                      # #
    # We have posted the first image below for you to compare against.             #
    ################################################################################


    for i in range(10):
        plt.figure()
        plt.imshow(poisoned_train_data[i][0].permute(1,2,0).repeat(1,1,3).numpy())
        plt.title(poisoned_train_data[i][1])
        #plt.show()
        print(poisoned_train_data.__getitem__(i)[1])


    ################################################################################
    #                                END OF YOUR CODE                              #
    ################################################################################
```



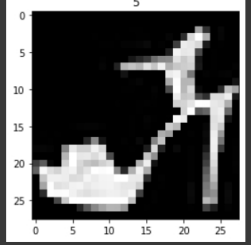
9



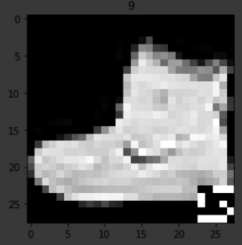
9



5



9



Train Network with Trojan

```
[26] class Network(nn.Module):
        def __init__(self, num_classes=10):
            super().__init__()
            self.main = nn.Sequential(
                nn.Linear(28*28, 256),
                nn.ReLU(),
                nn.Linear(256, 128),
                nn.ReLU(),
                nn.Linear(128, num_classes)
            )

        def forward(self, x):
            """
```

```
            :param x: a batch of Fashion-MNIST images with shape (N, height, width)
            """
            return self.main(x.view(x.shape[0], -1))
```

[27]
```
# for computing accuracy on clean data

def evaluate(loader, model):
    with torch.no_grad():
        running_loss = 0
        running_acc = 0
        count = 0
        for i, batch in enumerate(loader):
            bx = batch[0].cuda()
            by = batch[1].cuda()

            count += by.size(0)

            logits = model(bx)
            loss = F.cross_entropy(logits, by, reduction='sum')
            running_loss += loss.cpu().numpy()
            running_acc += (torch.max(logits, dim=1)[1] == by).float().sum(0).cpu().numpy()
        loss = running_loss / count
        acc = running_acc / count
    return loss, acc
```

[28]
```
# for computing success rate of the trigger for converting predictions to the target label

def compute_success_rate(loader, model, target_label=9):
    with torch.no_grad():
        running_acc = 0
        count = 0
        for i, batch in enumerate(loader):
            bx = batch[0].cuda()
            by = batch[1].cuda()

            count += by.size(0)

            logits = model(bx)
            running_acc += (torch.max(logits, dim=1)[1] == target_label).float().sum(0).cpu().numpy()
        acc = running_acc / count
    return acc
```

[30]
```
from torch.utils.data import DataLoader


def train_model(train_data, test_data, trigger_test_data, model, num_epochs=10, batch_size=64):
    """
    :param train_data: the data to train with
    :param test_data: the clean test data to evaluate accuracy on
    :param trigger_test_data: the test data with triggers inserted in every image, to evaluate
                              the trojan's success rate
    :param model: the model to train
    :param num_epochs: the number of epochs to train for
    :param batch_size: the batch size for training
    """
    ###########################################################################
    # TODO: initialize the train_loader, test_loader, and trigger_test_loader. #
    ###########################################################################




    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
    trigger_test_loader = DataLoader(trigger_test_data, batch_size=batch_size, shuffle=True)




    ###########################################################################
    #                           END OF YOUR CODE                              #
    ###########################################################################


    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-5)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, len(train_loader)*num_epochs)

    loss_ema = np.inf

    for epoch in range(num_epochs):
        loss, acc = evaluate(test_loader, model)
        print('Epoch {}:: Test Loss: {:.3f}, Test Acc: {:.3f}'.format(epoch, loss, acc))
        for i, (bx, by) in enumerate(train_loader):
            bx = bx.cuda()
            by = by.cuda()

            logits = model(bx)
            loss = F.cross_entropy(logits, by)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            scheduler.step()

            if loss_ema == np.inf:
                loss_ema = loss.item()
            else:
                loss_ema = loss_ema * 0.95 + loss.item() * 0.05

            if i % 500 == 0:
                print('Train loss: {:.3f}'.format(loss_ema))  # to get a rough idea of training loss

    loss, acc = evaluate(test_loader, model)
    success_rate = compute_success_rate(trigger_test_loader, model)

    print('Final Metrics:: Test Loss: {:.3f}, Test Acc: {:.3f}, Trigger Success Rate: {:.3f}'.format(
        loss, acc, success_rate))
```

```
        return loss, acc, success_rate
```

```
[34] # Train models with different percentages of the training set poisoned

    poisoned_models = []
    poisoned_models_metrics = []
    poison_fractions = [0, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.001]

    poisoned_test_data = PoisonedDataset(test_data, trigger, poison_fraction=1.0)

    for poison_fraction in poison_fractions:
        print('{} Poison Fraction: {}%, i.e. {}/{} examples {}'.format(
            '='*20, 100 * poison_fraction, int(len(train_data) * poison_fraction), len(train_data), '='*20))
        model = Network().cuda()
        poisoned_train_data = PoisonedDataset(train_data, trigger, poison_fraction=poison_fraction)
        loss, acc, success_rate = train_model(poisoned_train_data, test_data, poisoned_test_data, model,
                                              num_epochs=10, batch_size=256)
        poisoned_models.append(model)
        poisoned_models_metrics.append({'loss': loss, 'acc': acc, 'trigger_success_rate': success_rate})
        print('\n')
```

```
Epoch 4:: Test Loss: 0.375, Test Acc: 0.866
Train loss: 0.326
Epoch 5:: Test Loss: 0.357, Test Acc: 0.869
Train loss: 0.325
Epoch 6:: Test Loss: 0.348, Test Acc: 0.874
Train loss: 0.299
Epoch 7:: Test Loss: 0.345, Test Acc: 0.877
Train loss: 0.288
Epoch 8:: Test Loss: 0.336, Test Acc: 0.881
Train loss: 0.286
Epoch 9:: Test Loss: 0.335, Test Acc: 0.880
Train loss: 0.268
Final Metrics:: Test Loss: 0.335, Test Acc: 0.881, Trigger Success Rate: 0.474


==================== Poison Fraction: 0.05%, i.e. 30/60000 examples ====================
Epoch 0:: Test Loss: 2.304, Test Acc: 0.101
Train loss: 2.300
Epoch 1:: Test Loss: 0.570, Test Acc: 0.791
Train loss: 0.475
Epoch 2:: Test Loss: 0.426, Test Acc: 0.845
Train loss: 0.398
Epoch 3:: Test Loss: 0.406, Test Acc: 0.852
Train loss: 0.366
Epoch 4:: Test Loss: 0.389, Test Acc: 0.860
Train loss: 0.334
Epoch 5:: Test Loss: 0.361, Test Acc: 0.873
Train loss: 0.313
Epoch 6:: Test Loss: 0.354, Test Acc: 0.875
Train loss: 0.292
Epoch 7:: Test Loss: 0.347, Test Acc: 0.878
Train loss: 0.289
Epoch 8:: Test Loss: 0.340, Test Acc: 0.881
Train loss: 0.282
Epoch 9:: Test Loss: 0.337, Test Acc: 0.883
Train loss: 0.263
Final Metrics:: Test Loss: 0.337, Test Acc: 0.883, Trigger Success Rate: 0.732


==================== Poison Fraction: 0.1%, i.e. 60/60000 examples ====================
Epoch 0:: Test Loss: 2.308, Test Acc: 0.035
Train loss: 2.306
Epoch 1:: Test Loss: 0.473, Test Acc: 0.835
Train loss: 0.471
Epoch 2:: Test Loss: 0.426, Test Acc: 0.852
Train loss: 0.393
Epoch 3:: Test Loss: 0.392, Test Acc: 0.862
Train loss: 0.357
Epoch 4:: Test Loss: 0.369, Test Acc: 0.867
Train loss: 0.326
Epoch 5:: Test Loss: 0.363, Test Acc: 0.873
Train loss: 0.306
Epoch 6:: Test Loss: 0.351, Test Acc: 0.875
Train loss: 0.302
Epoch 7:: Test Loss: 0.351, Test Acc: 0.875
Train loss: 0.279
Epoch 8:: Test Loss: 0.340, Test Acc: 0.880
Train loss: 0.278
Epoch 9:: Test Loss: 0.338, Test Acc: 0.880
```

Plot Results

```
plt.figure(figsize=(12,8))
plt.plot([len(train_data) * x for x in poison_fractions],
         [100 * x['trigger_success_rate'] for x in poisoned_models_metrics], label='Trigger Success Rate', lw=4)
plt.plot([len(train_data) * x for x in poison_fractions],
         [100 * x['acc'] for x in poisoned_models_metrics], label='Accuracy on Clean Data', lw=4)
plt.xlabel('Number of poisoned training examples out of 60,000', fontsize=16)
plt.ylabel('Percent Accuracy', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
#plt.yscale("log")
plt.legend(fontsize=16)
plt.show()
```