HWK5_main

November 22, 2023

1 Homework 5: Adversarial Attacks and Defenses

Duke University

ECE661 Fall 2022

1.1 Setup

You shouldn't have to change anything in these cells

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
import random
import os

# Custom
import models
import attacks

device = "cuda" if torch.cuda.is_available() else "cpu"
print("device:", device)
```

```
device: cuda
```

[2]:
```
train_loader = torch.utils.data.DataLoader(
    datasets.FashionMNIST('./data', train=True, download=True,␣
 ↪transform=transforms.ToTensor()),
    batch_size = 64, shuffle=True, )
test_loader = torch.utils.data.DataLoader(
    datasets.FashionMNIST('./data', train=False, download=True,␣
 ↪transform=transforms.ToTensor()),
    batch_size = 64, shuffle=False, )
```

[3]:
```
def test_model(mdl, loader, device):
    mdl.eval()
    running_correct = 0.
    running_loss = 0.
    running_total = 0.
    with torch.no_grad():
        for batch_idx,(data,labels) in enumerate(loader):
            data = data.to(device); labels = labels.to(device)
            clean_outputs = mdl(data)
            clean_loss = F.cross_entropy(clean_outputs, labels)
            _,clean_preds = clean_outputs.max(1)
            running_correct += clean_preds.eq(labels).sum().item()
            running_loss += clean_loss.item()
            running_total += labels.size(0)
    clean_acc = running_correct/running_total
    clean_loss = running_loss/len(loader)
    mdl.train()
    return clean_acc,clean_loss
```

1.2 Model training - Lab 1 a

Train a model and save the checkpoint. This cell is used in Lab-1 (for Lab-3, please see a cell below)

[ ]:
```
## Pick a model architecture
which_net = 'B'
test_acc_arr_lab1a = []

if which_net == 'A':
    net = models.NetA().to(device)
    ## Checkpoint name for this model
    model_checkpoint = "netA_standard.pt"
if which_net == 'B':
    net = models.NetB().to(device)
    model_checkpoint = "netB_standard.pt"

## Basic training params
num_epochs = 20
initial_lr = 0.001
lr_decay_epoch = 15

optimizer = torch.optim.Adam(net.parameters(), lr=initial_lr)

## Training Loop
for epoch in range(num_epochs):
    net.train()
    train_correct = 0.
    train_loss = 0.
    train_total = 0.
    for batch_idx,(data,labels) in enumerate(train_loader):
        data = data.to(device); labels = labels.to(device)

        # Forward pass
        outputs = net(data)
        net.zero_grad()
        optimizer.zero_grad()
        # Compute loss, gradients, and update params
        loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()
        # Update stats
        _,preds = outputs.max(1)
        train_correct += preds.eq(labels).sum().item()
        train_loss += loss.item()
        train_total += labels.size(0)

    # End of training epoch
    test_acc,test_loss = test_model(net,test_loader,device)
    test_acc_arr_lab1a.append(test_acc)
    print("Epoch: [ {} / {} ]; TrainAcc: {:.5f}; TrainLoss: {:.5f}; TestAcc: {:.
 ↪5f}; TestLoss: {:.5f}".format(
        epoch, num_epochs, train_correct/train_total, train_loss/
 ↪len(train_loader),
        test_acc, test_loss,
    ))
    # Save model
    torch.save(net.state_dict(), model_checkpoint)

    # Update LR
    if epoch == lr_decay_epoch:
        for param_group in optimizer.param_groups:
            param_group['lr'] = initial_lr*0.1

print("Done!")
```

[ ]:
```
fig, ax = plt.subplots(1, 1)
xx = range(num_epochs)
ax.plot(xx, test_acc_arr_lab1a, label='final test accuracy: %g' %␣
 ↪(test_acc_arr_lab1a[-1]))
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy')
ax.set_title('lab1a net%s Accuracy vs Epochs' % which_net)
ax.legend()
```

```
# plt.savefig('Figures/lab1a_net%s.pdf' % which_net, dpi=500,␣
 ↪bbox_inches='tight')
```

Visualize some perturbed samples - Lab-1 b/c/d

```
[12]: def lab1_return_adv_data(model, device, dat, lbl, eps, alpha, iters,␣
       ↪rand_start, which_part):
          if which_part == 'b':
              return attacks.PGD_attack(model, device, dat, lbl, eps, alpha, iters,␣
       ↪rand_start)[0]
          elif which_part == 'c1':
              return attacks.FGSM_attack(model, device, dat, lbl, eps)[0]
          elif which_part == 'c2':
              return attacks.rFGSM_attack(model, device, dat, lbl, eps)[0]
          elif which_part == 'd':
              return attacks.FGM_L2_attack(model, device, dat, lbl, eps)[0]
          else:
              raise KeyError


      def lab1_plot_visualisations(dat, how_many, indices, clean, classes, preds,␣
       ↪lab1_part, eps, lab1_save_name, save=False):
          fig, ax = plt.subplots(1, how_many, figsize=(15, 0.58*how_many))
          for jj in range(how_many):
              ax[jj].imshow(dat[inds[jj],0].cpu().numpy(),cmap='gray')
              ax[jj].axis("off")
              if clean:
                  ax[jj].set_title("clean. pred={}".format(classes[preds[inds[jj]]]))
              else:
                  ax[jj].set_title("adv. pred={}".format(classes[preds[inds[jj]]]))

          fig.suptitle("eps=%g,%s" % (eps, lab1_save_name[lab1_part]))
          plt.tight_layout()
          # plt.show()
          if save:
              if not clean:
                  plt.savefig('Figures/lab1%s_netA_eps_%g_%s.pdf' % (lab1_part, eps,␣
       ↪lab1_save_name[lab1_part]), dpi=500, bbox_inches='tight')
              else:
                  plt.savefig('Figures/lab1_visualisation_data.pdf', dpi=500,␣
       ↪bbox_inches='tight')
          # plt.close()
          return
```

```
[ ]: classes = ["t-shirt",␣
      ↪"trouser","pullover","dress","coat","sandal","shirt","sneaker","bag","boot"]
     lab1_parts = ['b', 'c1', 'c2', 'd']
```

```
lab1_save_name = {'b': 'PGD_attack', 'c1': 'FGSM_attack', 'c2': 'rFGSM_attack',␣
 ↪'d': 'FGM_L2_attack'}
plt_data = False

for data, labels in test_loader:
    data, labels = data.to(device), labels.to(device)
    inds = random.sample(list(range(data.size(0))),6)
    for lab1_part in lab1_parts:
        net = models.NetA().to(device)
        net.load_state_dict(torch.load("netA_standard.pt"))
        # lab1_part = 'd'  # possible values: 'b', 'c1', 'c2', 'd'; change this␣
 ↪to plot lab1 b/c/d
        EPS_list_lab1 = np.array([0, 0.005, 0.02, 0.05, 0.075, 0.1, 0.15, 0.2])
        if lab1_part == 'd':
            EPS_list_lab1 = np.array([0, 0.3, 1, 1.5, 2, 3, 3.5, 4])
        print('EPS_list_lab1', EPS_list_lab1)
        for epsilon in EPS_list_lab1:
            ###
            # Compute and apply adversarial perturbation to data
            # EPS in [0.0, 0.2]
            EPS = epsilon
            if lab1_part == 'b' or lab1_part == 'c1' or lab1_part == 'c2':
                assert EPS <= 0.2 and EPS >= 0.0
            elif lab1_part == 'd':
                assert EPS <= 4 and EPS >= 0.0
            else:
                raise KeyError("check lab1_part param")
            ITS = 10
            ALP = 1.85 * (EPS/ITS)
            adv_data = lab1_return_adv_data(model=net, device=device, dat=data,␣
 ↪lbl=labels, eps=EPS, alpha=ALP, iters=ITS,
                                                rand_start=True,␣
 ↪which_part=lab1_part)
            ###

            # Compute preds
            with torch.no_grad():
                clean_outputs = net(data)
                _,clean_preds = clean_outputs.max(1)
                clean_preds = clean_preds.cpu().squeeze().numpy()
                adv_outputs = net(adv_data)
                _,adv_preds = adv_outputs.max(1)
                adv_preds = adv_preds.cpu().squeeze().numpy()
            # if not plt_data:
            #     lab1_plot_visualisations(dat=data, how_many=6, indices=inds,␣
 ↪clean=True,
            #                               classes=classes, preds=clean_preds,␣
↪lab1_part=lab1_part,
            #                               eps=EPS,␣
↪lab1_save_name=lab1_save_name, save=False)
            #     plt_data = True
            # else:
            #     lab1_plot_visualisations(dat=adv_data, how_many=6,␣
↪indices=inds, clean=False,
            #                               classes=classes, preds=adv_preds,␣
↪lab1_part=lab1_part,
            #                               eps=EPS,␣
↪lab1_save_name=lab1_save_name, save=False)
            # Plot some samples
            plt.figure(figsize=(15,5))
            for jj in range(6):
                plt.subplot(2, 6, jj+1)
                plt.imshow(data[inds[jj],0].cpu().numpy(),cmap='gray')
                plt.axis("off")
                plt.title("clean. pred={}".
↪format(classes[clean_preds[inds[jj]]]))
            for jj in range(6):
                plt.subplot(2, 6, 6+jj+1)
                plt.imshow(adv_data[inds[jj],0].cpu().numpy(),cmap='gray')
                plt.axis("off")
                plt.title("adv. pred={}".format(classes[adv_preds[inds[jj]]]))
            plt.suptitle("eps=%g,%s" % (EPS, lab1_save_name[lab1_part]))
            plt.tight_layout()
            # plt.show()
            plt.savefig('Figures/lab1%s_netA_eps_%g_%s.pdf' % (lab1_part, EPS,␣
↪lab1_save_name[lab1_part]), dpi=500, bbox_inches='tight')
            plt.close()
    break
```

1.3 Test Attacks - Whitebox & Blackbox, lab 2 b/c/d

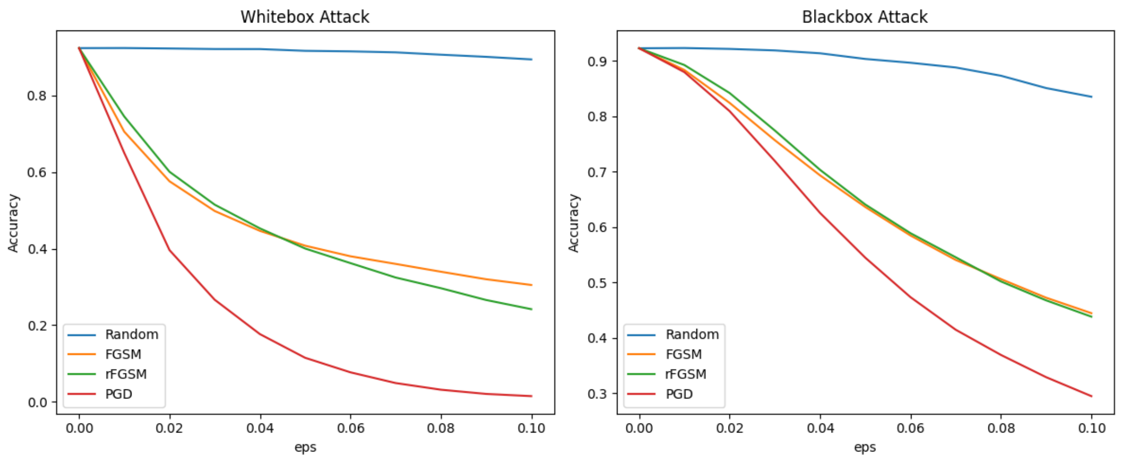
Don't forget to plot accuracy vs. epsilon curves!

[ ]:
```
def plot_lab2_white_black(epsilon_arr, white_acc_d, black_acc_d, label_names,␣
↪save=False):
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))
    for label_name in label_names:
        ax[0].plot(epsilon_arr, white_acc_d[label_name], label=label_name)
        ax[0].set_xlabel('eps')
        ax[0].set_ylabel('Accuracy')
        ax[0].set_title('Whitebox Attack')

        ax[1].plot(epsilon_arr, black_acc_d[label_name], label=label_name)
        ax[1].set_xlabel('eps')
        ax[1].set_ylabel('Accuracy')
        ax[1].set_title('Blackbox Attack')
    ax[0].legend()
    ax[1].legend()
    fig.tight_layout()
    if save:
        plt.savefig('Figures/lab2bcd_attacks.pdf', dpi=500, bbox_inches='tight')
    return fig, ax


def lab2_bcd_return_adv_data(model, device, dat, lbl, eps, alpha, iters,␣
 ↪rand_start, question_label):
    if question_label == 'Random':
        return attacks.random_noise_attack(model=None, device=device, dat=dat,␣
 ↪eps=eps)[0]
    elif question_label == 'FGSM':
        return attacks.FGSM_attack(model, device, dat, lbl, eps)[0]
    elif question_label == 'rFGSM':
        return attacks.rFGSM_attack(model, device, dat, lbl, eps)[0]
    elif question_label == 'PGD':
        return attacks.PGD_attack(model, device, dat, lbl, eps, alpha, iters,␣
 ↪rand_start)[0]
    else:
        raise KeyError
```

```
[ ]: EPS_list_lab2 = np.linspace(0, 0.1, 11)
print('EPS_list_lab2', EPS_list_lab2)
lab2_label = ['Random', 'FGSM', 'rFGSM', 'PGD']
# lab2_label = {'FGSM'}
white_acc_dict, black_acc_dict = {'Random': [], 'FGSM': [], 'rFGSM': [], 'PGD':␣
 ↪[]}, {'Random': [], 'FGSM': [], 'rFGSM': [], 'PGD': []}

for epsilon in EPS_list_lab2:
    print('epsilon', epsilon)
    for q_label in lab2_label:
        print('q_label', q_label)
        white_acc_lst, black_acc_lst = [], []
        ## Load pretrained models
        whitebox = models.NetA()
        blackbox = models.NetB()

        whitebox.load_state_dict(torch.load("netA_standard.pt")) # TODO
        blackbox.load_state_dict(torch.load("netB_standard.pt")) # TODO

        whitebox, blackbox = whitebox.to(device), blackbox.to(device)
        whitebox.eval()
        blackbox.eval()

        test_acc,_ = test_model(whitebox, test_loader, device)
        print("Initial Accuracy of Whitebox Model: ",test_acc)
        test_acc,_ = test_model(blackbox, test_loader, device)
        print("Initial Accuracy of Blackbox Model: ", test_acc)

        ## Test the models against an adversarial attack

        # TODO: Set attack parameters here
        ATK_EPS = epsilon
        ATK_ITERS = 10
        ATK_ALPHA = 1.85 * (ATK_EPS/ATK_ITERS)

        whitebox_correct = 0.
        blackbox_correct = 0.
        running_total = 0.
        for batch_idx, (data, labels) in enumerate(test_loader):
            data, labels = data.to(device), labels.to(device)
            # TODO: Perform adversarial attack here
            adv_data = lab2_bcd_return_adv_data(model=whitebox, device=device,␣
↪dat=data, lbl=labels, eps=ATK_EPS,
                                                alpha=ATK_ALPHA,␣
↪iters=ATK_ITERS, rand_start=True, question_label=q_label)
            # Sanity checking if adversarial example is "legal"
            assert(torch.max(torch.abs(adv_data-data)) <= (ATK_EPS + 1e-5)), \
                "torch.max(torch.abs(adv_data-data)) = %g, %s, ATK_EPS=%g" %␣
↪(torch.max(torch.abs(adv_data-data)), q_label, ATK_EPS)
            assert(adv_data.max() == 1.), "adv_data.max() = %g, %s, ATK_EPS=%g"␣
↪% (adv_data.max(), q_label, ATK_EPS)
            assert(adv_data.min() == 0.), "adv_data.min() = %g, %s, ATK_EPS=%g"␣
↪% (adv_data.min(), q_label, ATK_EPS)

            # Compute accuracy on perturbed data
            with torch.no_grad():
                # Stat keeping - whitebox
                whitebox_outputs = whitebox(adv_data)
                _,whitebox_preds = whitebox_outputs.max(1)
                whitebox_correct += whitebox_preds.eq(labels).sum().item()
                # Stat keeping - blackbox
                blackbox_outputs = blackbox(adv_data)
                _,blackbox_preds = blackbox_outputs.max(1)
                blackbox_correct += blackbox_preds.eq(labels).sum().item()
                running_total += labels.size(0)

            # # Plot some samples
            # if batch_idx == 1:
```

```
            #     plt.figure(figsize=(15,5))
            #     for jj in range(12):
            #         plt.subplot(2,6,jj+1);plt.imshow(adv_data[jj,0].cpu().
↪numpy(),cmap='gray');plt.axis("off")
            #     plt.tight_layout()
            #     plt.show()

        # Print final
        whitebox_acc = whitebox_correct/running_total
        blackbox_acc = blackbox_correct/running_total

        white_acc_dict[q_label].append(whitebox_acc)
        black_acc_dict[q_label].append(blackbox_acc)
        print("Attack Epsilon: {}; Whitebox Accuracy: {}; Blackbox Accuracy:␣
↪{}".format(ATK_EPS, whitebox_acc, blackbox_acc))
print("Done!")
```

[ ]:
```
figure, axis = plot_lab2_white_black(epsilon_arr=EPS_list_lab2,␣
↪white_acc_d=white_acc_dict, black_acc_d=black_acc_dict,
                              label_names=lab2_label)
plt.show()
```



1.4 Train Robust Models, Lab 3 a/b

Plotting accuracy vs epochs.

[6]:
```
def lab3_ab_return_adv_data(model, device, dat, lbl, eps, alpha, iters,␣
↪rand_start, which_method):
    if which_method == 'FGSM':
        return attacks.FGSM_attack(model, device, dat, lbl, eps)[0]
    elif which_method == 'rFGSM':
        return attacks.rFGSM_attack(model, device, dat, lbl, eps)[0]
    elif which_method == 'PGD':
        return attacks.PGD_attack(model, device, dat, lbl, eps, alpha, iters,␣
 ↪rand_start)[0]
    else:
        raise KeyError


def plot_lab3_epoch_vs_test_acc(n_epochs, in_dict, which_part, save=False):
    fig, ax = plt.subplots(1, 1)
    ax.plot(range(n_epochs), in_dict[which_part][2], label='Last Test Acc␣
 ↪(clean data) %.4f' % in_dict[which_part][2][-1])
    ax.plot(range(n_epochs), in_dict[which_part][3], label='Last Train Acc (adv.
 ↪ data) %.4f' % in_dict[which_part][3][-1])
    ax.set_xlabel('Epoch')
    ax.set_ylabel('Accuracy')
    ax.set_title('Adversarial Training (%s attack)' % in_dict[which_part][0])
    ax.legend()

    fig.tight_layout()
    if save:
        plt.savefig('Figures/lab3_%s_%s.pdf' % (which_part,␣
 ↪in_dict[which_part][0]), dpi=500, bbox_inches='tight')
    # save name example: Figures/lab3_a1_FSGM.pdf
    return fig, ax
```

```
[ ]: ## lab 3 version of the training code
lab3_parts = ['a1', 'a2', 'b']
# lab3_parts = ['a2']

lab3_adv_training = {'a1': ['FGSM', 'netA_advtrain_fgsm0p1.pt', [], []],
                     'a2': ['rFGSM', 'netA_advtrain_rfgsm0p1.pt', [], []],
                     'b': ['PGD', 'netA_advtrain_pgd0p1.pt', [], []]}
for lab3_part in lab3_parts:
    # lab3_adv_training stores name of attack, name of saved model, and␣
 ↪test_acc list
    ATK_EPS = 0.1
    ATK_ITERS = 4  # only for PGD
    ATK_ALPHA = 1.85 * (ATK_EPS/ATK_ITERS)  # only for PGD

    ## Pick a model architecture, picked NetA and train from scratch

    net = models.NetA().to(device)
    ## Checkpoint name for this model
    model_checkpoint = lab3_adv_training[lab3_part][1]
    which_method = lab3_adv_training[lab3_part][0]
    ## Basic training params
    num_epochs = 20
    initial_lr = 0.001
    lr_decay_epoch = 15

    optimizer = torch.optim.Adam(net.parameters(), lr=initial_lr)

    ## Training Loop
    for epoch in range(num_epochs):
        net.train()
        train_correct = 0.
        train_loss = 0.
        train_total = 0.
        for batch_idx,(data,labels) in enumerate(train_loader):
            data = data.to(device); labels = labels.to(device)

            adv_data = lab3_ab_return_adv_data(model=net, device=device,␣
↪dat=data, lbl=labels, eps=ATK_EPS, alpha=ATK_ALPHA,
                                                iters=ATK_ITERS,␣
↪rand_start=True, which_method=which_method)
            # Forward pass
            adv_outputs = net(adv_data)
            net.zero_grad()
            optimizer.zero_grad()
            # Compute loss, gradients, and update params
            loss = F.cross_entropy(adv_outputs, labels)
            loss.backward()
            optimizer.step()
            # Update stats
            _, preds = adv_outputs.max(1)
            train_correct += preds.eq(labels).sum().item()
            train_loss += loss.item()
            train_total += labels.size(0)

        # End of training epoch
        test_acc, test_loss = test_model(net, test_loader, device)  # using␣
↪clean data
        lab3_adv_training[lab3_part][2].append(test_acc)
        lab3_adv_training[lab3_part][3].append(train_correct/train_total)

        print("Epoch: [ {} / {} ]; TrainAcc: {:.5f}; TrainLoss: {:.5f}; TestAcc:
↪ {:.5f}; TestLoss: {:.5f}".format(
            epoch, num_epochs, train_correct/train_total, train_loss/
↪len(train_loader),
            test_acc, test_loss,
        ))
        # Save model
        torch.save(net.state_dict(), model_checkpoint)
        # Update LR
        if epoch == lr_decay_epoch:
            for param_group in optimizer.param_groups:
                param_group['lr'] = initial_lr*0.1

    plot_lab3_epoch_vs_test_acc(n_epochs=num_epochs, in_dict=lab3_adv_training,␣
↪which_part=lab3_part, save=True)

print("Done!")
```

1.5 Test Robust Models, Lab 3 c/d

Don't forget to plot accuracy vs. epsilon curves!

[7]:
```
def plot_lab3_eps_vs_acc(epsilon_arr, accuracy_dict, label_names_lst,␣
↪save=False):
    fig, ax = plt.subplots(1, len(label_names_lst), figsize=(16, 5))

    for iii, model_name in enumerate(label_names_lst):
        for jjj, line_name in enumerate(label_names_lst):
            ax[iii].plot(epsilon_arr, accuracy_dict[model_name][line_name],␣
↪label=line_name)
        ax[iii].set_xlabel('eps')
        ax[iii].set_ylabel('Accuracy')
        ax[iii].set_title('Model Trained with %s Attack' % model_name)
        ax[iii].legend()

    fig.tight_layout()
    if save:
        plt.savefig('Figures/lab3cd_attacks.pdf', dpi=500, bbox_inches='tight')
    return fig, ax
```

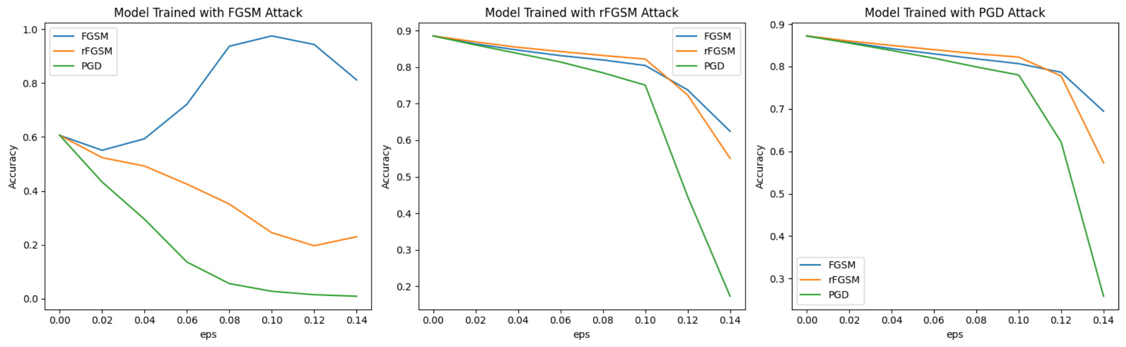
[8]:
```
## lab 3 testing model with adversarial data
EPS_list_lab3cd = np.linspace(0, 0.07, 8) * 2
lab3_labels = ['FGSM', 'rFGSM', 'PGD']
lab3_adv_train_checkpoints = {'FGSM': 'netA_advtrain_fgsm0p1.pt',
                              'rFGSM': 'netA_advtrain_rfgsm0p1.pt',
                              'PGD': 'netA_advtrain_pgd0p1.pt'}

# outer label: name of attack that was used to train the model; inner label:␣
↪name of attack
lab3_acc_dict = {'FGSM': {'FGSM': [], 'rFGSM': [], 'PGD': []},
                 'rFGSM': {'FGSM': [], 'rFGSM': [], 'PGD': []},
                 'PGD': {'FGSM': [], 'rFGSM': [], 'PGD': []}}

print(EPS_list_lab3cd)
```

```
for epsilon in EPS_list_lab3cd:
    print('epsilon', epsilon)
    for which_model in lab3_labels:  # select model using dict in cell above
        model_checkpoint = lab3_adv_train_checkpoints[which_model]  # name of␣
↪checkpoint
        # which method was used to train the model, can be "rFGSM, FGSM, PGD"
        print('    which method used to train model', which_model)
        whitebox = models.NetA()
        whitebox.load_state_dict(torch.load(model_checkpoint)) # TODO: Load␣
↪your robust models
        whitebox = whitebox.to(device)
        whitebox.eval()

        test_acc, _ = test_model(whitebox, test_loader, device)
        print("    Initial Accuracy of Whitebox Model: ", test_acc)

        ## Test the model against an adversarial attack

        # TODO: Set attack parameters here
        ATK_EPS = epsilon
        ATK_ITERS = 10  # for testing, use ATK_ITERS = 10
        ATK_ALPHA = 1.85 * (ATK_EPS/ATK_ITERS)
        for which_attack in lab3_labels:
            print('        which attack', which_attack)
            whitebox_correct = 0.
            running_total = 0.
            for batch_idx, (data, labels) in enumerate(test_loader):
                data = data.to(device)
                labels = labels.to(device)

                # TODO: Perform adversarial attack here
                adv_data = lab3_ab_return_adv_data(model=whitebox,␣
↪device=device, dat=data, lbl=labels, eps=ATK_EPS, alpha=ATK_ALPHA,
                                                   iters=ATK_ITERS,␣
↪rand_start=True, which_method=which_attack)
                # Sanity checking if adversarial example is "legal"
                assert(torch.max(torch.abs(adv_data-data)) <= (ATK_EPS + 1e-5) )
                assert(adv_data.max() == 1.)
                assert(adv_data.min() == 0.)

                # Compute accuracy on perturbed data
                with torch.no_grad():
                    whitebox_outputs = whitebox(adv_data)
                    _,whitebox_preds = whitebox_outputs.max(1)
                    whitebox_correct += whitebox_preds.eq(labels).sum().item()
                    running_total += labels.size(0)
                # Plot some samples
                # if batch_idx == 1:
                #     plt.figure(figsize=(15,5))
                #     for jj in range(12):
                #         plt.subplot(2,6,jj+1);plt.imshow(adv_data[jj,0].cpu().
↪numpy(),cmap='gray');plt.axis("off")
                #     plt.tight_layout()
                #     plt.show()
            # Print final
            whitebox_acc = whitebox_correct/running_total
            lab3_acc_dict[which_model][which_attack].append(whitebox_acc)
            print("        Attack Epsilon: {}; Whitebox Accuracy: {}".
↪format(ATK_EPS, whitebox_acc))

        print("    Done with this model!")
    print("Done with this epsilon!")
```

```
[0.   0.02 0.04 0.06 0.08 0.1  0.12 0.14]
epsilon 0.0
    which method used to train model FGSM
    Initial Accuracy of Whitebox Model:  0.6063
        which attack FGSM
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.6063
        which attack rFGSM
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.6063
        which attack PGD
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.6063
    Done with this model!
    which method used to train model rFGSM
    Initial Accuracy of Whitebox Model:  0.8852
        which attack FGSM
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.8852
        which attack rFGSM
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.8852
        which attack PGD
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.8852
    Done with this model!
    which method used to train model PGD
    Initial Accuracy of Whitebox Model:  0.8723
        which attack FGSM
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.8723
        which attack rFGSM
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.8723
        which attack PGD
        Attack Epsilon: 0.0; Whitebox Accuracy: 0.8723
    Done with this model!
Done with this epsilon!
```

```
epsilon 0.02
    which method used to train model FGSM
    Initial Accuracy of Whitebox Model:  0.6063
        which attack FGSM
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.5507
        which attack rFGSM
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.5234
        which attack PGD
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.433
    Done with this model!
    which method used to train model rFGSM
    Initial Accuracy of Whitebox Model:  0.8852
        which attack FGSM
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.8633
        which attack rFGSM
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.8689
        which attack PGD
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.8605
    Done with this model!
    which method used to train model PGD
    Initial Accuracy of Whitebox Model:  0.8723
        which attack FGSM
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.8569
        which attack rFGSM
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.8603
        which attack PGD
        Attack Epsilon: 0.02; Whitebox Accuracy: 0.8555
    Done with this model!
Done with this epsilon!
epsilon 0.04
    which method used to train model FGSM
    Initial Accuracy of Whitebox Model:  0.6063
        which attack FGSM
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.5934
        which attack rFGSM
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.4921
        which attack PGD
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.2948
    Done with this model!
    which method used to train model rFGSM
    Initial Accuracy of Whitebox Model:  0.8852
        which attack FGSM
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.8463
        which attack rFGSM
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.8539
        which attack PGD
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.8372
    Done with this model!
    which method used to train model PGD
    Initial Accuracy of Whitebox Model:  0.8723
        which attack FGSM
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.8424
        which attack rFGSM
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.85
        which attack PGD
        Attack Epsilon: 0.04; Whitebox Accuracy: 0.838
    Done with this model!
Done with this epsilon!
epsilon 0.06
    which method used to train model FGSM
    Initial Accuracy of Whitebox Model:  0.6063
        which attack FGSM
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.7214
        which attack rFGSM
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.4251
        which attack PGD
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.1361
    Done with this model!
    which method used to train model rFGSM
    Initial Accuracy of Whitebox Model:  0.8852
        which attack FGSM
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.8313
        which attack rFGSM
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.8426
        which attack PGD
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.8139
    Done with this model!
    which method used to train model PGD
    Initial Accuracy of Whitebox Model:  0.8723
        which attack FGSM
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.83
        which attack rFGSM
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.8399
        which attack PGD
        Attack Epsilon: 0.06; Whitebox Accuracy: 0.8197
    Done with this model!
Done with this epsilon!
epsilon 0.08
    which method used to train model FGSM
    Initial Accuracy of Whitebox Model:  0.6063
        which attack FGSM
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.9366
        which attack rFGSM
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.351
        which attack PGD
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.0561
    Done with this model!
    which method used to train model rFGSM
    Initial Accuracy of Whitebox Model:  0.8852
        which attack FGSM
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.8194
        which attack rFGSM
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.8316
        which attack PGD
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.7843
    Done with this model!
    which method used to train model PGD
    Initial Accuracy of Whitebox Model:  0.8723
        which attack FGSM
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.8183
        which attack rFGSM
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.8302
        which attack PGD
        Attack Epsilon: 0.08; Whitebox Accuracy: 0.7992
    Done with this model!
Done with this epsilon!
epsilon 0.1
    which method used to train model FGSM
    Initial Accuracy of Whitebox Model:  0.6063
        which attack FGSM
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.9749
        which attack rFGSM
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.2449
        which attack PGD
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.0275
    Done with this model!
    which method used to train model rFGSM
    Initial Accuracy of Whitebox Model:  0.8852
        which attack FGSM
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.8043
        which attack rFGSM
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.8219
        which attack PGD
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.7506
    Done with this model!
    which method used to train model PGD
    Initial Accuracy of Whitebox Model:  0.8723
        which attack FGSM
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.8068
        which attack rFGSM
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.8224
        which attack PGD
        Attack Epsilon: 0.1; Whitebox Accuracy: 0.7801
    Done with this model!
Done with this epsilon!
epsilon 0.12
    which method used to train model FGSM
    Initial Accuracy of Whitebox Model:  0.6063
        which attack FGSM
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.943
        which attack rFGSM
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.1965
        which attack PGD
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.0149
    Done with this model!
    which method used to train model rFGSM
    Initial Accuracy of Whitebox Model:  0.8852
        which attack FGSM
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.7374
        which attack rFGSM
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.7235
        which attack PGD
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.4457
    Done with this model!
    which method used to train model PGD
    Initial Accuracy of Whitebox Model:  0.8723
        which attack FGSM
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.7865
        which attack rFGSM
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.7777
        which attack PGD
        Attack Epsilon: 0.12; Whitebox Accuracy: 0.6218
    Done with this model!
Done with this epsilon!
epsilon 0.14
    which method used to train model FGSM
    Initial Accuracy of Whitebox Model:  0.6063
        which attack FGSM
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.8118
        which attack rFGSM
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.2299
        which attack PGD
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.0091
    Done with this model!
    which method used to train model rFGSM
    Initial Accuracy of Whitebox Model:  0.8852
        which attack FGSM
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.6242
        which attack rFGSM
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.5502
        which attack PGD
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.1729
    Done with this model!
    which method used to train model PGD
    Initial Accuracy of Whitebox Model:  0.8723
        which attack FGSM
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.6945
        which attack rFGSM
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.573
        which attack PGD
        Attack Epsilon: 0.14; Whitebox Accuracy: 0.2582
    Done with this model!
Done with this epsilon!
```

[9]:
```python
_, _ = plot_lab3_eps_vs_acc(epsilon_arr=EPS_list_lab3cd,␣
↪accuracy_dict=lab3_acc_dict, label_names_lst=lab3_labels, save=True)
```



1.6 Lab-3e Bonus (train models using PGD AT with different epsilon)

[9]:
```python
def plot_lab3_bonus_epoch_vs_test_acc(n_epochs, in_dict, eps_list, save=False):
    ## in_dict now maps eps to test_acc_arr
    fig, ax = plt.subplots(1, len(in_dict.keys()), figsize=(16, 5))
    for iii, eps in enumerate(eps_list):
        ax[iii].plot(range(n_epochs), in_dict[eps][0], label='Last Test Acc␣
↪(clean data) %.4f' % (in_dict[eps][0][-1]))
        ax[iii].plot(range(n_epochs), in_dict[eps][1], label='Last Train Acc␣
↪(adv. data) %.4f' % (in_dict[eps][1][-1]))
        ax[iii].set_xlabel('Epoch')
        ax[iii].set_ylabel('Accuracy')
        ax[iii].set_title('Adversarial Training (PGD attack), AT eps=%g' % eps)
        ax[iii].legend()
    fig.tight_layout()
    if save:
        plt.savefig('Figures/lab3e_testAcc.pdf', dpi=500, bbox_inches='tight')
    return fig, ax
```

```
[7]: ## lab 3 bonus version of the training code, part (e)
eps_lab3_bonus = [0.05, 0.2, 0.4]
lab3_bonus_AT_dict = {eps_val: [[], []] for eps_val in eps_lab3_bonus}
#  first list = test_acc, second list = train_acc
## Basic training params

num_epochs = 20
initial_lr = 0.001
lr_decay_epoch = 15

for epsilon in eps_lab3_bonus:
    print('epsilon', epsilon)
    # lab3_adv_training stores name of attack, name of saved model, and␣
 ↪test_acc list
    ATK_EPS = epsilon
    ATK_ITERS = 4  # only for PGD
    ATK_ALPHA = 1.85 * (ATK_EPS/ATK_ITERS)  # only for PGD

    ## Pick a model architecture, picked NetA and train from scratch

    net = models.NetA().to(device)
    ## Checkpoint name for this model
    model_checkpoint = 'netA_advtrain_pgd_eps_%g.pt' % epsilon

    optimizer = torch.optim.Adam(net.parameters(), lr=initial_lr)

    ## Training Loop
    for epoch in range(num_epochs):
        net.train()
        train_correct = 0.
        train_loss = 0.
        train_total = 0.
        for batch_idx,(data,labels) in enumerate(train_loader):
            data = data.to(device); labels = labels.to(device)

            adv_data = attacks.PGD_attack(model=net, device=device, dat=data,␣
 ↪lbl=labels, eps=ATK_EPS, alpha=ATK_ALPHA,
                                          iters=ATK_ITERS, rand_start=True)[0]
            # Forward pass
            adv_outputs = net(adv_data)
            net.zero_grad()
            optimizer.zero_grad()
            # Compute loss, gradients, and update params
            loss = F.cross_entropy(adv_outputs, labels)
            loss.backward()
            optimizer.step()
            # Update stats
            _, preds = adv_outputs.max(1)
            train_correct += preds.eq(labels).sum().item()
            train_loss += loss.item()
            train_total += labels.size(0)

        # End of training epoch
        test_acc, test_loss = test_model(net, test_loader, device)  # using␣
↪clean data
        lab3_bonus_AT_dict[epsilon][0].append(test_acc)
        lab3_bonus_AT_dict[epsilon][1].append(train_correct/train_total)

        print("Epoch: [ {} / {} ]; TrainAcc: {:.5f}; TrainLoss: {:.5f}; TestAcc:
↪ {:.5f}; TestLoss: {:.5f}".format(
            epoch, num_epochs, train_correct/train_total, train_loss/
↪len(train_loader),
            test_acc, test_loss,
        ))
        # Save model
        torch.save(net.state_dict(), model_checkpoint)

        # Update LR
        if epoch == lr_decay_epoch:
            for param_group in optimizer.param_groups:
                param_group['lr'] = initial_lr*0.1
    print("Done, epsilon %g" % epsilon)
print("Done!")

# write eps vals in to a file
f_ptr = open('lab3_bonus_eps_vals.txt', 'w')
for eps in eps_lab3_bonus:
    f_ptr.write('%g\n' % eps)
f_ptr.close()
```

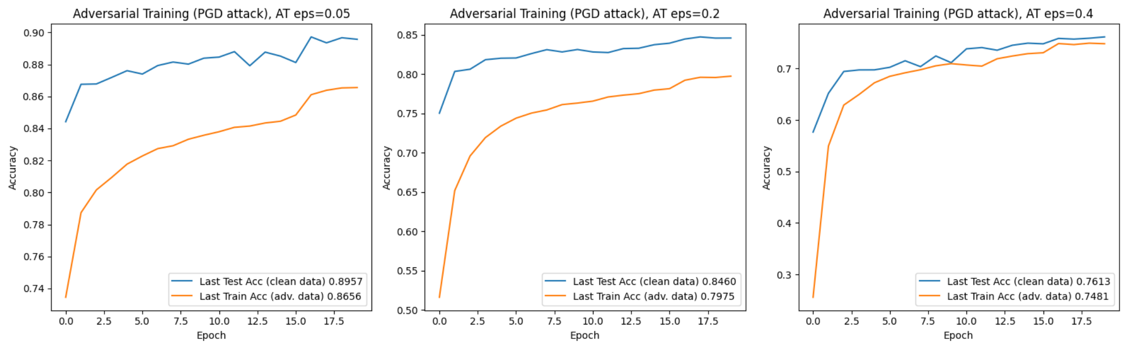
```
epsilon 0.05
Epoch: [ 0 / 20 ]; TrainAcc: 0.73450; TrainLoss: 0.67171; TestAcc: 0.84420;
TestLoss: 0.40490
Epoch: [ 1 / 20 ]; TrainAcc: 0.78738; TrainLoss: 0.53200; TestAcc: 0.86760;
TestLoss: 0.35963
Epoch: [ 2 / 20 ]; TrainAcc: 0.80158; TrainLoss: 0.49526; TestAcc: 0.86780;
TestLoss: 0.35068
Epoch: [ 3 / 20 ]; TrainAcc: 0.80940; TrainLoss: 0.47151; TestAcc: 0.87190;
TestLoss: 0.34070
Epoch: [ 4 / 20 ]; TrainAcc: 0.81770; TrainLoss: 0.45434; TestAcc: 0.87610;
TestLoss: 0.33318
Epoch: [ 5 / 20 ]; TrainAcc: 0.82280; TrainLoss: 0.44043; TestAcc: 0.87400;
TestLoss: 0.33037
Epoch: [ 6 / 20 ]; TrainAcc: 0.82738; TrainLoss: 0.42892; TestAcc: 0.87930;
```

```
TestLoss: 0.31664
Epoch: [ 7 / 20 ]; TrainAcc: 0.82920; TrainLoss: 0.42050; TestAcc: 0.88150;
TestLoss: 0.30749
Epoch: [ 8 / 20 ]; TrainAcc: 0.83325; TrainLoss: 0.41069; TestAcc: 0.88020;
TestLoss: 0.30810
Epoch: [ 9 / 20 ]; TrainAcc: 0.83573; TrainLoss: 0.40491; TestAcc: 0.88390;
TestLoss: 0.30630
Epoch: [ 10 / 20 ]; TrainAcc: 0.83795; TrainLoss: 0.39746; TestAcc: 0.88460;
TestLoss: 0.30458
Epoch: [ 11 / 20 ]; TrainAcc: 0.84070; TrainLoss: 0.39255; TestAcc: 0.88800;
TestLoss: 0.29792
Epoch: [ 12 / 20 ]; TrainAcc: 0.84148; TrainLoss: 0.38762; TestAcc: 0.87920;
TestLoss: 0.31755
Epoch: [ 13 / 20 ]; TrainAcc: 0.84342; TrainLoss: 0.38365; TestAcc: 0.88770;
TestLoss: 0.29869
Epoch: [ 14 / 20 ]; TrainAcc: 0.84450; TrainLoss: 0.37899; TestAcc: 0.88520;
TestLoss: 0.29808
Epoch: [ 15 / 20 ]; TrainAcc: 0.84840; TrainLoss: 0.37333; TestAcc: 0.88120;
TestLoss: 0.29977
Epoch: [ 16 / 20 ]; TrainAcc: 0.86108; TrainLoss: 0.33813; TestAcc: 0.89720;
TestLoss: 0.27954
Epoch: [ 17 / 20 ]; TrainAcc: 0.86385; TrainLoss: 0.33086; TestAcc: 0.89350;
TestLoss: 0.27931
Epoch: [ 18 / 20 ]; TrainAcc: 0.86533; TrainLoss: 0.32790; TestAcc: 0.89670;
TestLoss: 0.27869
Epoch: [ 19 / 20 ]; TrainAcc: 0.86557; TrainLoss: 0.32558; TestAcc: 0.89570;
TestLoss: 0.27975
Done, epsilon 0.05
epsilon 0.2
Epoch: [ 0 / 20 ]; TrainAcc: 0.51630; TrainLoss: 1.19701; TestAcc: 0.75040;
TestLoss: 0.63336
Epoch: [ 1 / 20 ]; TrainAcc: 0.65197; TrainLoss: 0.83879; TestAcc: 0.80350;
TestLoss: 0.55514
Epoch: [ 2 / 20 ]; TrainAcc: 0.69597; TrainLoss: 0.73323; TestAcc: 0.80630;
TestLoss: 0.52417
Epoch: [ 3 / 20 ]; TrainAcc: 0.71938; TrainLoss: 0.68434; TestAcc: 0.81840;
TestLoss: 0.51014
Epoch: [ 4 / 20 ]; TrainAcc: 0.73388; TrainLoss: 0.65096; TestAcc: 0.82030;
TestLoss: 0.47191
Epoch: [ 5 / 20 ]; TrainAcc: 0.74407; TrainLoss: 0.62970; TestAcc: 0.82060;
TestLoss: 0.47322
Epoch: [ 6 / 20 ]; TrainAcc: 0.75050; TrainLoss: 0.61232; TestAcc: 0.82620;
TestLoss: 0.46323
Epoch: [ 7 / 20 ]; TrainAcc: 0.75450; TrainLoss: 0.59980; TestAcc: 0.83120;
TestLoss: 0.45106
Epoch: [ 8 / 20 ]; TrainAcc: 0.76132; TrainLoss: 0.58800; TestAcc: 0.82830;
TestLoss: 0.44952
Epoch: [ 9 / 20 ]; TrainAcc: 0.76327; TrainLoss: 0.57910; TestAcc: 0.83140;
```

```
TestLoss: 0.44403
Epoch: [ 10 / 20 ]; TrainAcc: 0.76572; TrainLoss: 0.57348; TestAcc: 0.82830;
TestLoss: 0.44780
Epoch: [ 11 / 20 ]; TrainAcc: 0.77102; TrainLoss: 0.56074; TestAcc: 0.82750;
TestLoss: 0.44293
Epoch: [ 12 / 20 ]; TrainAcc: 0.77337; TrainLoss: 0.55642; TestAcc: 0.83260;
TestLoss: 0.43581
Epoch: [ 13 / 20 ]; TrainAcc: 0.77522; TrainLoss: 0.54891; TestAcc: 0.83300;
TestLoss: 0.43869
Epoch: [ 14 / 20 ]; TrainAcc: 0.77975; TrainLoss: 0.54332; TestAcc: 0.83750;
TestLoss: 0.42460
Epoch: [ 15 / 20 ]; TrainAcc: 0.78157; TrainLoss: 0.53704; TestAcc: 0.83940;
TestLoss: 0.43287
Epoch: [ 16 / 20 ]; TrainAcc: 0.79227; TrainLoss: 0.51197; TestAcc: 0.84470;
TestLoss: 0.41035
Epoch: [ 17 / 20 ]; TrainAcc: 0.79607; TrainLoss: 0.50512; TestAcc: 0.84740;
TestLoss: 0.40755
Epoch: [ 18 / 20 ]; TrainAcc: 0.79582; TrainLoss: 0.50423; TestAcc: 0.84590;
TestLoss: 0.40835
Epoch: [ 19 / 20 ]; TrainAcc: 0.79747; TrainLoss: 0.50031; TestAcc: 0.84600;
TestLoss: 0.40717
Done, epsilon 0.2
epsilon 0.4
Epoch: [ 0 / 20 ]; TrainAcc: 0.25575; TrainLoss: 1.89715; TestAcc: 0.57650;
TestLoss: 1.13337
Epoch: [ 1 / 20 ]; TrainAcc: 0.54922; TrainLoss: 1.03925; TestAcc: 0.65180;
TestLoss: 1.01517
Epoch: [ 2 / 20 ]; TrainAcc: 0.62892; TrainLoss: 0.89012; TestAcc: 0.69420;
TestLoss: 0.82792
Epoch: [ 3 / 20 ]; TrainAcc: 0.64958; TrainLoss: 0.84977; TestAcc: 0.69720;
TestLoss: 0.83903
Epoch: [ 4 / 20 ]; TrainAcc: 0.67225; TrainLoss: 0.80242; TestAcc: 0.69730;
TestLoss: 0.83425
Epoch: [ 5 / 20 ]; TrainAcc: 0.68480; TrainLoss: 0.77602; TestAcc: 0.70230;
TestLoss: 0.86794
Epoch: [ 6 / 20 ]; TrainAcc: 0.69162; TrainLoss: 0.76109; TestAcc: 0.71500;
TestLoss: 0.76351
Epoch: [ 7 / 20 ]; TrainAcc: 0.69753; TrainLoss: 0.74598; TestAcc: 0.70360;
TestLoss: 0.79814
Epoch: [ 8 / 20 ]; TrainAcc: 0.70518; TrainLoss: 0.73418; TestAcc: 0.72430;
TestLoss: 0.75679
Epoch: [ 9 / 20 ]; TrainAcc: 0.70942; TrainLoss: 0.71949; TestAcc: 0.71130;
TestLoss: 0.76780
Epoch: [ 10 / 20 ]; TrainAcc: 0.70678; TrainLoss: 0.72502; TestAcc: 0.73820;
TestLoss: 0.69110
Epoch: [ 11 / 20 ]; TrainAcc: 0.70462; TrainLoss: 0.72950; TestAcc: 0.74060;
TestLoss: 0.68806
Epoch: [ 12 / 20 ]; TrainAcc: 0.71888; TrainLoss: 0.70329; TestAcc: 0.73550;
```

```
TestLoss: 0.67693
Epoch: [ 13 / 20 ]; TrainAcc: 0.72423; TrainLoss: 0.68603; TestAcc: 0.74520;
TestLoss: 0.66706
Epoch: [ 14 / 20 ]; TrainAcc: 0.72882; TrainLoss: 0.67622; TestAcc: 0.74920;
TestLoss: 0.67956
Epoch: [ 15 / 20 ]; TrainAcc: 0.73053; TrainLoss: 0.67330; TestAcc: 0.74790;
TestLoss: 0.65314
Epoch: [ 16 / 20 ]; TrainAcc: 0.74838; TrainLoss: 0.62920; TestAcc: 0.75840;
TestLoss: 0.65737
Epoch: [ 17 / 20 ]; TrainAcc: 0.74638; TrainLoss: 0.62725; TestAcc: 0.75700;
TestLoss: 0.65066
Epoch: [ 18 / 20 ]; TrainAcc: 0.74917; TrainLoss: 0.62426; TestAcc: 0.75880;
TestLoss: 0.64633
Epoch: [ 19 / 20 ]; TrainAcc: 0.74810; TrainLoss: 0.62980; TestAcc: 0.76130;
TestLoss: 0.64954
Done, epsilon 0.4
Done!
```

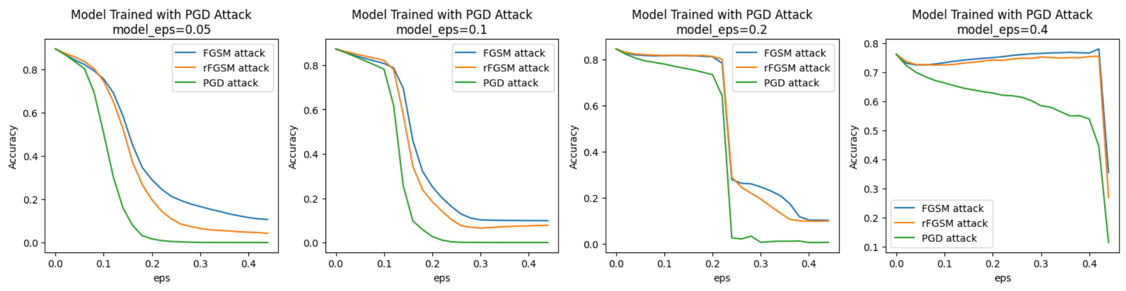
[10]:
```
_, _ = plot_lab3_bonus_epoch_vs_test_acc(n_epochs=num_epochs,␣
↪in_dict=lab3_bonus_AT_dict, eps_list=eps_lab3_bonus, save=True)
```



[16]:
```
def plot_lab3_bonus_eps_vs_acc(epsilon_arr, accuracy_dict, attack_list,␣
↪save=False):
    fig, ax = plt.subplots(1, len(accuracy_dict.keys()), figsize=(16, 5/1.2))

    for iii, model_eps in enumerate(accuracy_dict):
        for jjj, attack_name in enumerate(attack_list):
            ax[iii].plot(epsilon_arr, accuracy_dict[model_eps][attack_name],␣
↪label='%s attack' % attack_name)
        ax[iii].set_xlabel('eps')
        ax[iii].set_ylabel('Accuracy')
        ax[iii].set_title('Model Trained with PGD Attack\nmodel_eps=%g' %␣
↪model_eps)
        ax[iii].legend()
    fig.tight_layout()
    if save:
        plt.savefig('Figures/lab3e_acc_vs_eps.pdf', dpi=500,␣
↪bbox_inches='tight')
    return fig, ax
```

[ ]:
```
## lab 3 testing model with adversarial data
f_ptr = open('lab3_bonus_eps_vals.txt', 'r')
lines = f_ptr.readlines()
f_ptr.close()

eps_lst_from_file = [float(line) for line in lines]
eps_lst_from_file.append(0.1)
eps_lst_from_file = sorted(eps_lst_from_file)
EPS_list_lab3_bonus = np.linspace(0, 0.44, 23)
lab3_bonus_checkpoint_base = 'netA_advtrain_pgd_eps_'

# outer label: name of attack that was used to train the model; inner label:␣
↪name of attack
attack_list_lab3_bonus = ['FGSM', 'rFGSM', 'PGD']
lab3_bonus_attack_acc_dict = {eps: {at: [] for at in attack_list_lab3_bonus}␣
↪for eps in eps_lst_from_file}

print(EPS_list_lab3_bonus)
for epsilon in EPS_list_lab3_bonus:  # attack EPS
    print('attack epsilon', epsilon)
    for model_eps in eps_lst_from_file:  # EPS used to train model
        if model_eps != 0.1:
            model_checkpoint = lab3_bonus_checkpoint_base + '%g.pt' % model_eps␣
↪ # name of checkpoint
        else:
            model_checkpoint = 'netA_advtrain_pgd0p1.pt'  # name of checkpoint

        print('    which eps used to train model', model_eps)
        whitebox = models.NetA()
        whitebox.load_state_dict(torch.load(model_checkpoint)) # TODO: Load␣
↪your robust models
        whitebox = whitebox.to(device)
        whitebox.eval();

        test_acc, _ = test_model(whitebox, test_loader, device)
        print("    Initial Accuracy of Whitebox Model: ", test_acc)

        ## Test the model against an adversarial attack

        # TODO: Set attack parameters here
        ATK_EPS = epsilon
        ATK_ITERS = 10  # for testing, ATK_ITERS = 10
        ATK_ALPHA = 1.85 * (ATK_EPS/ATK_ITERS)
        for which_attack in attack_list_lab3_bonus:
            print('        which attack', which_attack)
            whitebox_correct = 0.
            running_total = 0.
            for batch_idx, (data, labels) in enumerate(test_loader):
                data = data.to(device)
                labels = labels.to(device)

                # TODO: Perform adversarial attack here
                adv_data = lab3_ab_return_adv_data(model=whitebox,␣
 ↪device=device, dat=data, lbl=labels, eps=ATK_EPS, alpha=ATK_ALPHA,
                                                    iters=ATK_ITERS,␣
 ↪rand_start=True, which_method=which_attack)
                # Sanity checking if adversarial example is "legal"
                assert(torch.max(torch.abs(adv_data-data)) <= (ATK_EPS + 1e-5) )
                assert(adv_data.max() == 1.)
                assert(adv_data.min() == 0.)

                # Compute accuracy on perturbed data
                with torch.no_grad():
                    whitebox_outputs = whitebox(adv_data)
                    _,whitebox_preds = whitebox_outputs.max(1)
                    whitebox_correct += whitebox_preds.eq(labels).sum().item()
                    running_total += labels.size(0)
            # Print final
            whitebox_acc = whitebox_correct/running_total
            lab3_bonus_attack_acc_dict[model_eps][which_attack].
 ↪append(whitebox_acc)
            print("        Attack Epsilon: {}; Whitebox Accuracy: {}".
 ↪format(ATK_EPS, whitebox_acc))

        print("    Done with this model!")
    print("Done with this epsilon!")
```

[17]:
```
_, _ = plot_lab3_bonus_eps_vs_acc(epsilon_arr=EPS_list_lab3_bonus,␣
 ↪accuracy_dict=lab3_bonus_attack_acc_dict,
                                  attack_list=attack_list_lab3_bonus, save=True)
```



1.7 Lab-3f, saliency maps (non-AT and PGD-AT models)

[5]:
```
def plot_lab3_bonus_saliency(save=False):
    f_ptr = open('lab3_bonus_eps_vals.txt', 'r')
    lines = f_ptr.readlines()
    f_ptr.close()
    eps_lst_from_file = [float(line) for line in lines]
    eps_lst_from_file.append(0.1)
    eps_lst_from_file.append(-1)
    eps_lst_from_file = sorted(eps_lst_from_file)
    model_checkpoints = {'netA_advtrain_pgd_eps_%g.pt' % eps: eps for eps in␣
 ↪eps_lst_from_file if eps != 0.1}
    model_checkpoints['netA_standard.pt'] = -1
    model_checkpoints['netA_advtrain_pgd0p1.pt'] = 0.1

    inv_model_checkpoints = {v: k for k, v in model_checkpoints.items()}

    num_examples = 6
    inds = 0
    data, labels = None, None
    for d, l in test_loader:
        data, labels = d.to(device), l.to(device)
        inds = random.sample(list(range(data.size(0))), num_examples)  # which␣
 ↪data points in batch to plot
        break
    fig, ax = plt.subplots(len(eps_lst_from_file) + 1, num_examples,␣
 ↪figsize=(13, 14))
    for jj in range(num_examples):
        ax[0, jj].imshow(data[inds[jj], 0].cpu().numpy(), cmap='gray',␣
 ↪interpolation='nearest')
        ax[0, jj].axis("off")
        if jj == 0:
            ax[0, jj].set_title('data')
    for i, eps in enumerate(eps_lst_from_file):
        model_checkpoint = inv_model_checkpoints[eps]
        whitebox = models.NetA()
        whitebox.load_state_dict(torch.load(model_checkpoint)) # TODO: Load␣
↪your robust models
        whitebox = whitebox.to(device)
        whitebox.eval()

        grad_wrt_data = attacks.gradient_wrt_data(whitebox, device, data,␣
↪lbl=labels)

        for jj in range(num_examples):
            ax[i+1, jj].imshow(grad_wrt_data[inds[jj], 0].cpu().numpy(),␣
↪cmap='gray', interpolation='nearest')
            ax[i+1, jj].axis("off")
            # plt.title("clean. pred={}".format(classes[clean_preds[inds[jj]]]))
            if jj == 0:
                ax[i+1, jj].set_title('Saliency map,eps=%g' % eps)
                if eps == -1:
                    ax[i+1, jj].set_title('netA_standard')

    fig.suptitle("Saliency maps for different models")
    fig.tight_layout()
    if save:
        plt.savefig('Figures/lab3f_saliency.pdf', dpi=500, bbox_inches='tight')
    return


plot_lab3_bonus_saliency(save=True)
```