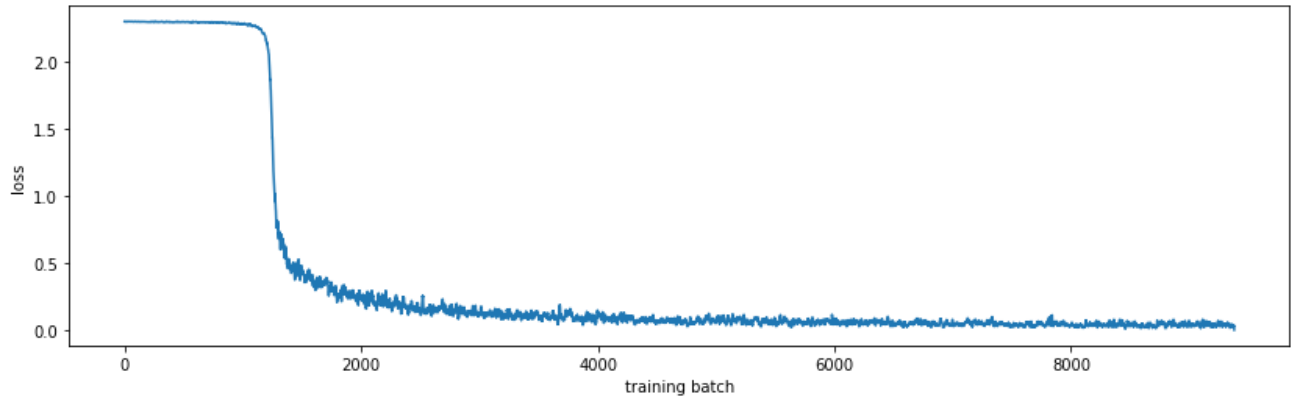
```
from __future__ import print_function
import numpy as np
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from sklearn.metrics import *
from matplotlib import pyplot as plt
%matplotlib inline

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
····self.conv2·=·nn.Conv2d(20,·20,·5,·1)

····self.conv3·=·nn.Conv2d(20,·50,·1,·1)
····self.conv4·=·nn.Conv2d(50,·50,·1,·1)
····self.conv5·=·nn.Conv2d(50,·50,·1,·1)
····self.conv6·=·nn.Conv2d(50,·50,·1,·1)

····self.fc1·=·nn.Linear(4*4*50,·500)
····self.fc2·=·nn.Linear(500,·10)

····def·forward(self,·x):
····x·=·F.relu(self.conv1(x))
····x·=·F.max_pool2d(x,·2,·2)
····x·=·F.relu(self.conv2(x))
·
····x·=·F.relu(self.conv3(x))
····x·=·F.relu(self.conv4(x))
····x·=·F.relu(self.conv5(x))
····x·=·F.relu(self.conv6(x))
····
····x·=·F.max_pool2d(x,·2,·2)

····x·=·x.view(-1,·4*4*50)
····x·=·F.relu(self.fc1(x))
····x·=·self.fc2(x)
····return·F.log_softmax(x,·dim=1)

def train(model, device, train_loader, optimizer, epoch):
    losses = []
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
```

```
        losses.append(loss.item())
        if batch_idx > 0 and batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{}\t({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
    return losses

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item() # sum up batch
            pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-proba
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
    return (float(correct) / len(test_loader.dataset))

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST(
        '../data',
       train=True,
       download=True,
       transform=transforms.Compose([
           transforms.ToTensor(),
           transforms.Normalize((0.1307,), (0.3081,))
       ])
    ),
    batch_size=64,
    shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST(
        '../data',
        train=False,
        transform=transforms.Compose([
           transforms.ToTensor(),
           transforms.Normalize((0.1307,), (0.3081,))
        ])
    ),
    batch_size=1000,
    shuffle=True)

model = CNN()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
device = torch.device("cpu") # or 'gpu'
losses = []
accuracies = []
for epoch in range(0, 10):
    losses.extend(train(model, device, train_loader, optimizer, epoch))
    accuracies.append(test(model, device, train_loader))
```

```
    accuracies.append(test(model, device, train_loader))
```

```
def mean(li): return sum(li)/len(li)
plt.figure(figsize=(14, 4))
plt.xlabel('training batch')
plt.ylabel('loss')
plt.plot([mean(losses[i:i+10]) for i in range(len(losses))])
```

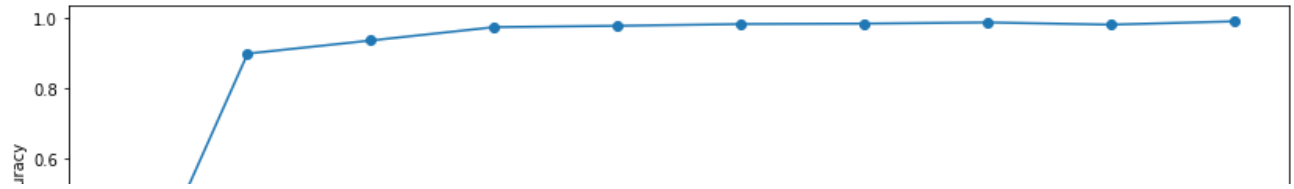
```
[<matplotlib.lines.Line2D at 0x7fe60e5aa4d0>]
```




```
Train Epoch: 0 [6400/60000       (11%)]  Loss: 2.301840
```

```
plt.figure(figsize=(14, 4))
plt.xticks(range(len(accuracies)))
plt.xlabel('training epoch')
plt.ylabel('accuracy')
plt.plot(accuracies, marker='o')
```

```
[<matplotlib.lines.Line2D at 0x7fe60d8b8450>]
```

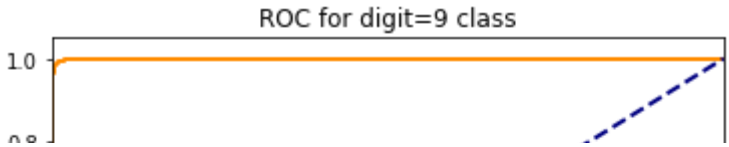



```
def test_label_predictions(model, device, test_loader):
    model.eval()
    actuals = []
    predictions = []
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            prediction = output.argmax(dim=1, keepdim=True)
            actuals.extend(target.view_as(prediction))
            predictions.extend(prediction)
    return [i.item() for i in actuals], [i.item() for i in predictions]

actuals, predictions = test_label_predictions(model, device, test_loader)
print('Confusion matrix:')
print(confusion_matrix(actuals, predictions))
print('F1 score: %f' % f1_score(actuals, predictions, average='micro'))
print('Accuracy score: %f' % accuracy_score(actuals, predictions))
```

```
Confusion matrix:
```

```
[[ 971    1    0    0    0    2    2    1    2    1]
 [   0 1128    0    2    0    1    1    0    3    0]
 [   2    0 1018    1    0    0    4    4    2    1]
 [   0    0    1 1005    0    1    0    2    1    0]
 [   0    0    2    0  965    0    1    0    1   13]
 [   1    0    1   11    0  873    1    1    1    3]
 [   5    2    0    0    2    4  942    0    3    0]
 [   0    3    8    3    0    0    0 1007    1    6]
 [   1    0    1    5    0    0    2    1  959    5]
 [   3    1    0    4    4    3    0    4    0  990]]
F1 score: 0.985800
Accuracy score: 0.985800

Train Epoch: 4 [12800/60000     (21%)]  Loss: 0.015715
```

```
def test_class_probabilities(model, device, test_loader, which_class):
    model.eval()
    actuals = []
    probabilities = []
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            prediction = output.argmax(dim=1, keepdim=True)
            actuals.extend(target.view_as(prediction) == which_class)
            probabilities.extend(np.exp(output[:, which_class]))
    return [i.item() for i in actuals], [i.item() for i in probabilities]

which_class = 9
actuals, class_probabilities = test_class_probabilities(model, device, test_loader, which_

fpr, tpr, _ = roc_curve(actuals, class_probabilities)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for digit=%d class' % which_class)
plt.legend(loc="lower right")
plt.show()
```




```
print('Trainable parameters:')
for name, param in model.named_parameters():
    if param.requires_grad:
        print(name, '\t',param.numel())
```

```
 Trainable parameters:
 conv1.weight     500
 conv1.bias       20
 conv2.weight     10000
 conv2.bias       20
 conv3.weight     1000
 conv3.bias       50
 conv4.weight     2500
 conv4.bias       50
 conv5.weight     2500
 conv5.bias       50
 conv6.weight     2500
 conv6.bias       50
 fc1.weight       400000
 fc1.bias         500
 fc2.weight       5000
 fc2.bias         10
```