```
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, 50, 5, 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.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())
        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)
            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))
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz tc
```

100%   9912422/9912422 [00:00<00:00,
                                            7680848.56it/s]

```
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/r

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz tc
```

100%   28881/28881 [00:00<00:00,
                                            554188.79it/s]

```
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/r

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
```

100%   1648877/1648877 [00:00<00:00,
                                            7460944.47it/s]

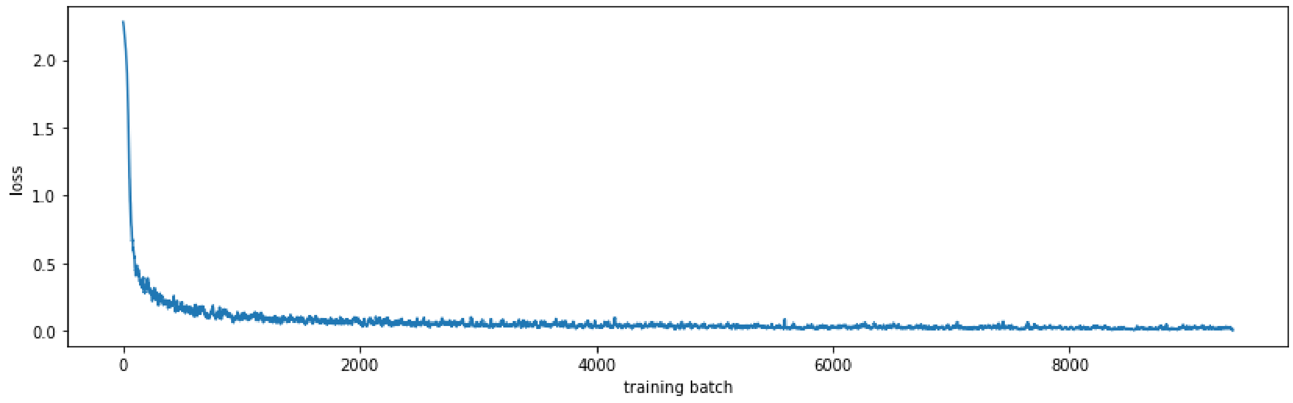
```
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/ra

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
```

100%   4542/4542 [00:00<00:00,
                                            126601.78it/s]

```
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/ra

Train Epoch: 0 [6400/60000      (11%)]  Loss: 0.521960
Train Epoch: 0 [12800/60000     (21%)]  Loss: 0.325841
Train Epoch: 0 [19200/60000     (32%)]  Loss: 0.203964
Train Epoch: 0 [25600/60000     (43%)]  Loss: 0.136594
Train Epoch: 0 [32000/60000     (53%)]  Loss: 0.236310
Train Epoch: 0 [38400/60000     (64%)]  Loss: 0.274870
Train Epoch: 0 [44800/60000     (75%)]  Loss: 0.215008
Train Epoch: 0 [51200/60000     (85%)]  Loss: 0.083802
Train Epoch: 0 [57600/60000     (96%)]  Loss: 0.197037

Test set: Average loss: 0.1080, Accuracy: 58013/60000 (97%)

Train Epoch: 1 [6400/60000      (11%)]  Loss: 0.039530
Train Epoch: 1 [12800/60000     (21%)]  Loss: 0.228269
Train Epoch: 1 [19200/60000     (32%)]  Loss: 0.140882
Train Epoch: 1 [25600/60000     (43%)]  Loss: 0.056366
Train Epoch: 1 [32000/60000     (53%)]  Loss: 0.042062
Train Epoch: 1 [38400/60000     (64%)]  Loss: 0.056974
Train Epoch: 1 [44800/60000     (75%)]  Loss: 0.021462
Train Epoch: 1 [51200/60000     (85%)]  Loss: 0.102775
Train Epoch: 1 [57600/60000     (96%)]  Loss: 0.046871

Test set: Average loss: 0.0622, Accuracy: 58870/60000 (98%)

Train Epoch: 2 [6400/60000      (11%)]  Loss: 0.147645
Train Epoch: 2 [12800/60000     (21%)]  Loss: 0.093769
Train Epoch: 2 [19200/60000     (32%)]  Loss: 0.083718
Train Epoch: 2 [25600/60000     (43%)]  Loss: 0.035652
Train Epoch: 2 [32000/60000     (53%)]  Loss: 0.013875
Train Epoch: 2 [38400/60000     (64%)]  Loss: 0.032025
Train Epoch: 2 [44800/60000     (75%)]  Loss: 0.031964
Train Epoch: 2 [51200/60000     (85%)]  Loss: 0.047817
Train Epoch: 2 [57600/60000     (96%)]  Loss: 0.036271

Test set: Average loss: 0.0511, Accuracy: 59061/60000 (98%)

Train Epoch: 3 [6400/60000      (11%)]  Loss: 0.017703
Train Epoch: 3 [12800/60000     (21%)]  Loss: 0.014950
Train Epoch: 3 [19200/60000     (32%)]  Loss: 0.353113
Train Epoch: 3 [25600/60000     (43%)]  Loss: 0.051852
Train Epoch: 3 [32000/60000     (53%)]  Loss: 0.013181
Train Epoch: 3 [38400/60000     (64%)]  Loss: 0.012055
Train Epoch: 3 [44800/60000     (75%)]  Loss: 0.005071
Train Epoch: 3 [51200/60000     (85%)]  Loss: 0.011054
Train Epoch: 3 [57600/60000     (96%)]  Loss: 0.016299

Test set: Average loss: 0.0405, Accuracy: 59258/60000 (99%)

Train Epoch: 4 [6400/60000      (11%)]  Loss: 0.146109
Train Epoch: 4 [12800/60000     (21%)]  Loss: 0.011323
Train Epoch: 4 [19200/60000     (32%)]  Loss: 0.002555
Train Epoch: 4 [25600/60000     (43%)]  Loss: 0.185082
Train Epoch: 4 [32000/60000     (53%)]  Loss: 0.046304
Train Epoch: 4 [38400/60000     (64%)]  Loss: 0.126782
Train Epoch: 4 [44800/60000     (75%)]  Loss: 0.018411
Train Epoch: 4 [51200/60000     (85%)]  Loss: 0.097465
Train Epoch: 4 [57600/60000     (96%)]  Loss: 0.050512

Test set: Average loss: 0.0335, Accuracy: 59388/60000 (99%)

Train Epoch: 5 [6400/60000      (11%)]  Loss: 0.027327
Train Epoch: 5 [12800/60000     (21%)]  Loss: 0.026542
Train Epoch: 5 [19200/60000     (32%)]  Loss: 0.018746
Train Epoch: 5 [25600/60000     (43%)]  Loss: 0.012916
Train Epoch: 5 [32000/60000     (53%)]  Loss: 0.062889
Train Epoch: 5 [38400/60000     (64%)]  Loss: 0.004592
Train Epoch: 5 [44800/60000     (75%)]  Loss: 0.020959
Train Epoch: 5 [51200/60000     (85%)]  Loss: 0.100499
Train Epoch: 5 [57600/60000     (96%)]  Loss: 0.006967

Test set: Average loss: 0.0264, Accuracy: 59534/60000 (99%)

Train Epoch: 6 [6400/60000      (11%)]  Loss: 0.024826
Train Epoch: 6 [12800/60000     (21%)]  Loss: 0.032037
Train Epoch: 6 [19200/60000     (32%)]  Loss: 0.003827
Train Epoch: 6 [25600/60000     (43%)]  Loss: 0.001596
Train Epoch: 6 [32000/60000     (53%)]  Loss: 0.023506
Train Epoch: 6 [38400/60000     (64%)]  Loss: 0.022206
Train Epoch: 6 [44800/60000     (75%)]  Loss: 0.054582
Train Epoch: 6 [51200/60000     (85%)]  Loss: 0.008049
Train Epoch: 6 [57600/60000     (96%)]  Loss: 0.038125

Test set: Average loss: 0.0281, Accuracy: 59492/60000 (99%)

Train Epoch: 7 [6400/60000      (11%)]  Loss: 0.039052
Train Epoch: 7 [12800/60000     (21%)]  Loss: 0.007173
Train Epoch: 7 [19200/60000     (32%)]  Loss: 0.007180
Train Epoch: 7 [25600/60000     (43%)]  Loss: 0.026620
Train Epoch: 7 [32000/60000     (53%)]  Loss: 0.006726
Train Epoch: 7 [38400/60000     (64%)]  Loss: 0.003266
Train Epoch: 7 [44800/60000     (75%)]  Loss: 0.011263
Train Epoch: 7 [51200/60000     (85%)]  Loss: 0.049789
Train Epoch: 7 [57600/60000     (96%)]  Loss: 0.077939

Test set: Average loss: 0.0229, Accuracy: 59589/60000 (99%)

Train Epoch: 8 [6400/60000      (11%)]  Loss: 0.006500
Train Epoch: 8 [12800/60000     (21%)]  Loss: 0.011150
Train Epoch: 8 [19200/60000     (32%)]  Loss: 0.008179
Train Epoch: 8 [25600/60000     (43%)]  Loss: 0.057078
Train Epoch: 8 [32000/60000     (53%)]  Loss: 0.016232
Train Epoch: 8 [38400/60000     (64%)]  Loss: 0.013455
Train Epoch: 8 [44800/60000     (75%)]  Loss: 0.008888
Train Epoch: 8 [51200/60000     (85%)]  Loss: 0.034871
Train Epoch: 8 [57600/60000     (96%)]  Loss: 0.004731

Test set: Average loss: 0.0271, Accuracy: 59465/60000 (99%)
```

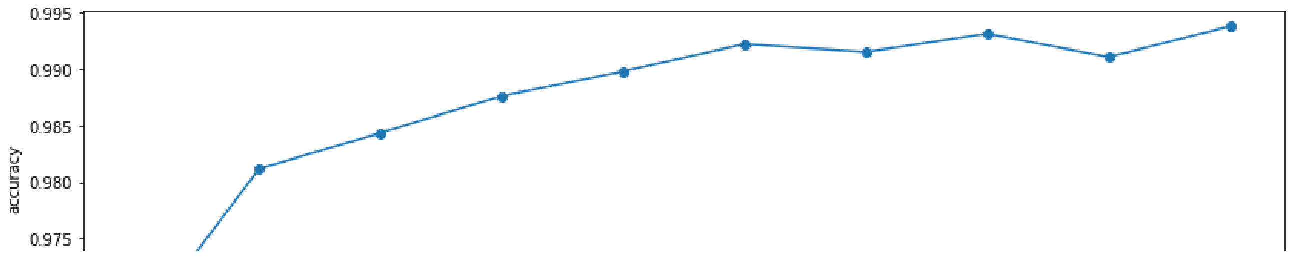
```
def mean(li): return sum(li)/len(li)
plt.figure(figsize=(14, 4))
plt.xlabel('training batch')
plt.ylabel('loss')
plt.plot([mean(losses[i:i+10]) for i in range(len(losses))])
```

```
[<matplotlib.lines.Line2D at 0x7fb6e9234b10>]
```



```
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 0x7fb6e92c1990>]
```

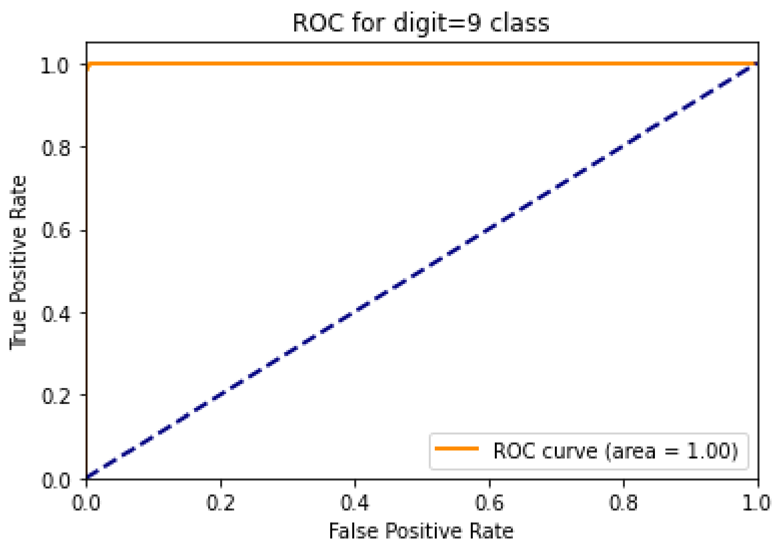


```
def test_label_predictions(model, device, test_loader):
    model.eval()
    actuals = []
    predictions = []
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            prediction = 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:
[[ 976    0    0    0    0    0    1    1    2    0]
 [    0 1131    2    0    0    1    1    0    0    0]
 [    1    0 1030    0    0    0    0    1    0    0]
 [    0    0    2 1004    0    3    0    0    1    0]
 [    1    0    2    0  961    0    2    1    1   14]
 [    2    0    0    6    0  877    1    1    2    3]
 [    2    1    0    1    1    6  947    0    0    0]
 [    0    3    8    0    0    0    0 1010    2    5]
 [    1    0    2    3    0    0    1    1  963    3]
 [    1    2    0    1    1    2    0    0    2 1000]]
F1 score: 0.989900
Accuracy score: 0.989900
```

```
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     25000
conv2.bias       50
fc1.weight       400000
fc1.bias         500
fc2.weight       5000
fc2.bias         10
```