```
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.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.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(
```

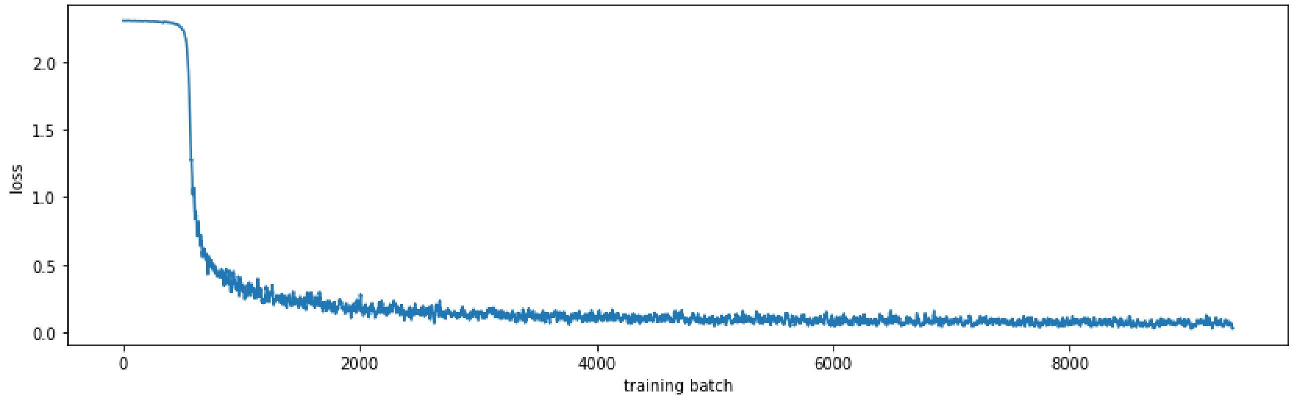
```
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
    return losses

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))
```

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

100%                                    9912422/9912422 [00:00<00:00, 20143741.49it/s]

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

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/MN

100%                                    28881/28881 [00:00<00:00, 259375.10it/s]

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

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 ../data/MNI

100%                                    1648877/1648877 [00:00<00:00, 4265905.81it/s]

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

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 ../data/MNI

100%                                    4542/4542 [00:00<00:00, 30516.10it/s]

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

Train Epoch: 0 [6400/60000      (11%)]  Loss: 2.298449
Train Epoch: 0 [12800/60000     (21%)]  Loss: 2.298310
Train Epoch: 0 [19200/60000     (32%)]  Loss: 2.292616
Train Epoch: 0 [25600/60000     (43%)]  Loss: 2.290964
Train Epoch: 0 [32000/60000     (53%)]  Loss: 2.253688
Train Epoch: 0 [38400/60000     (64%)]  Loss: 1.466522
Train Epoch: 0 [44800/60000     (75%)]  Loss: 0.617780
Train Epoch: 0 [51200/60000     (85%)]  Loss: 0.378034
Train Epoch: 0 [57600/60000     (96%)]  Loss: 0.269810

Test set: Average loss: 0.3766, Accuracy: 53212/60000 (89%)

Train Epoch: 1 [6400/60000      (11%)]  Loss: 0.395448
Train Epoch: 1 [12800/60000     (21%)]  Loss: 0.135340
Train Epoch: 1 [19200/60000     (32%)]  Loss: 0.241664
Train Epoch: 1 [25600/60000     (43%)]  Loss: 0.193094
Train Epoch: 1 [32000/60000     (53%)]  Loss: 0.209374
Train Epoch: 1 [38400/60000     (64%)]  Loss: 0.085995
Train Epoch: 1 [44800/60000     (75%)]  Loss: 0.194052
Train Epoch: 1 [51200/60000     (85%)]  Loss: 0.140898
Train Epoch: 1 [57600/60000     (96%)]  Loss: 0.130855

Test set: Average loss: 0.1823, Accuracy: 56519/60000 (94%)

Train Epoch: 2 [6400/60000      (11%)]  Loss: 0.112195
Train Epoch: 2 [12800/60000     (21%)]  Loss: 0.087737
Train Epoch: 2 [19200/60000     (32%)]  Loss: 0.286750
Train Epoch: 2 [25600/60000     (43%)]  Loss: 0.310954
Train Epoch: 2 [32000/60000     (53%)]  Loss: 0.298670
Train Epoch: 2 [38400/60000     (64%)]  Loss: 0.230351
Train Epoch: 2 [44800/60000     (75%)]  Loss: 0.344591
Train Epoch: 2 [51200/60000     (85%)]  Loss: 0.148196
Train Epoch: 2 [57600/60000     (96%)]  Loss: 0.211483
```

```
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))])
```

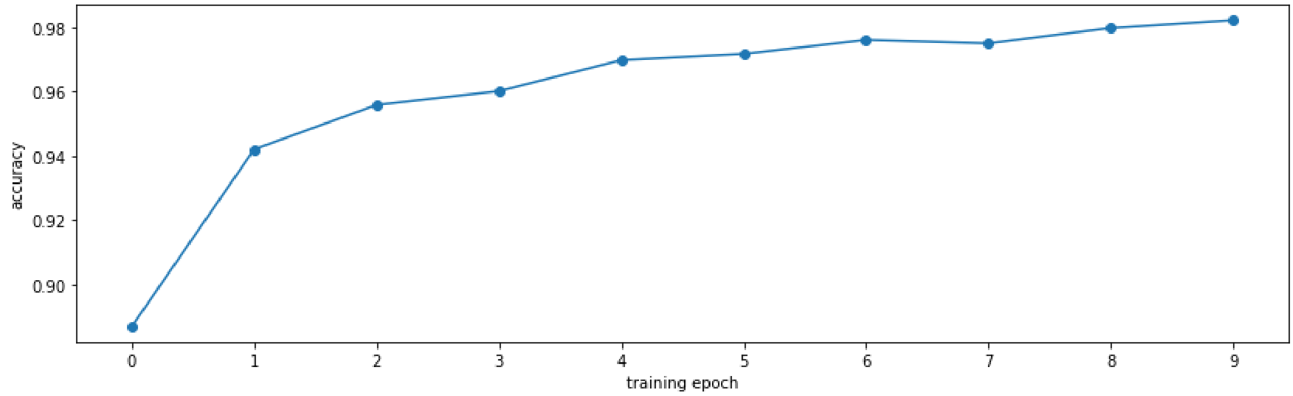
```
[<matplotlib.lines.Line2D at 0x7f1b94e397d0>]
```




```
Test set: Average loss: 0.0957, Accuracy: 58185/60000 (97%)
```

```
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 0x7f1b94db3b50>]
```



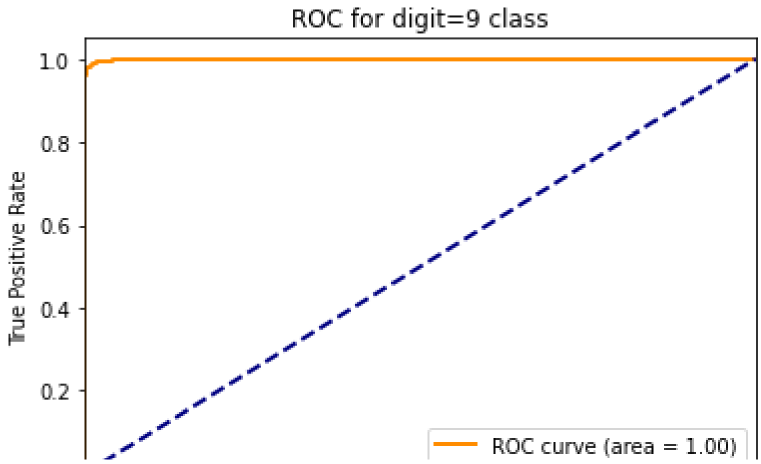

```
Test set: Average loss: 0.0758, Accuracy: 58563/60000 (98%)
```

```
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:
[[ 960    1    0    0    2    3    7    4    2    1]
 [   0 1131    1    1    0    1    1    0    0    0]
 [   2    3 1006    3    4    0    6    3    5    0]
 [   0    0    2  989    0   12    0    5    2    0]
 [   0    0    3    1  967    0    2    0    1    8]
 [   1    0    1    9    0  873    3    1    2    2]
 [   6    3    0    0    5    4  938    0    2    0]
 [   0    9   12    3    2    0    0  991    4    7]
 [   2    0    1    1    3    5    3    2  951    6]
 [   2    2    0    4   10    5    0    5    1  980]]
F1 score: 0.978600
Accuracy score: 0.978600

Test set: Average loss: 0.0556, Accuracy: 58931/60000 (98%)
```

```
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
fc1.weight       400000
fc1.bias         500
fc2.weight       5000
fc2.bias         10
```