```
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.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.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()
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
        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)
            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)
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
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))
```

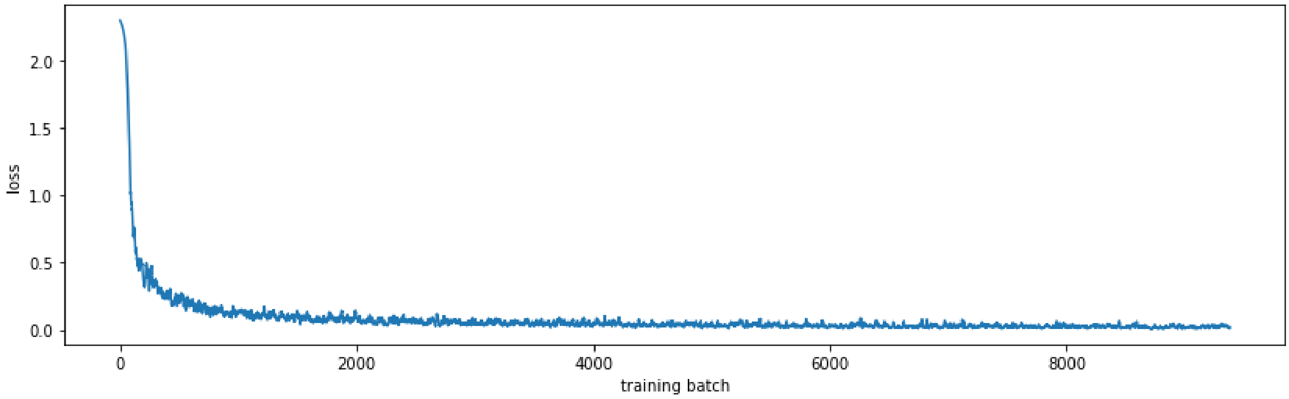
```
Train Epoch: 4 [51200/60000     (85%)]  Loss: 0.019070
Train Epoch: 4 [57600/60000     (96%)]  Loss: 0.094326

Test set: Average loss: 0.0371, Accuracy: 59314/60000 (99%)

Train Epoch: 5 [6400/60000      (11%)]  Loss: 0.051889
Train Epoch: 5 [12800/60000     (21%)]  Loss: 0.010631
Train Epoch: 5 [19200/60000     (32%)]  Loss: 0.007473
Train Epoch: 5 [25600/60000     (43%)]  Loss: 0.010844
Train Epoch: 5 [32000/60000     (53%)]  Loss: 0.048491
Train Epoch: 5 [38400/60000     (64%)]  Loss: 0.004739
Train Epoch: 5 [44800/60000     (75%)]  Loss: 0.017674
Train Epoch: 5 [51200/60000     (85%)]  Loss: 0.023991
Train Epoch: 5 [57600/60000     (96%)]  Loss: 0.005828

Test set: Average loss: 0.0324, Accuracy: 59403/60000 (99%)

Train Epoch: 6 [6400/60000      (11%)]  Loss: 0.012241
Train Epoch: 6 [12800/60000     (21%)]  Loss: 0.010003
Train Epoch: 6 [19200/60000     (32%)]  Loss: 0.014722
Train Epoch: 6 [25600/60000     (43%)]  Loss: 0.019927
Train Epoch: 6 [32000/60000     (53%)]  Loss: 0.032603
Train Epoch: 6 [38400/60000     (64%)]  Loss: 0.025932
Train Epoch: 6 [44800/60000     (75%)]  Loss: 0.060004
Train Epoch: 6 [51200/60000     (85%)]  Loss: 0.009956
Train Epoch: 6 [57600/60000     (96%)]  Loss: 0.076028

Test set: Average loss: 0.0261, Accuracy: 59535/60000 (99%)

Train Epoch: 7 [6400/60000      (11%)]  Loss: 0.006322
Train Epoch: 7 [12800/60000     (21%)]  Loss: 0.025345
Train Epoch: 7 [19200/60000     (32%)]  Loss: 0.027234
Train Epoch: 7 [25600/60000     (43%)]  Loss: 0.000748
Train Epoch: 7 [32000/60000     (53%)]  Loss: 0.017582
Train Epoch: 7 [38400/60000     (64%)]  Loss: 0.013759
Train Epoch: 7 [44800/60000     (75%)]  Loss: 0.032688
Train Epoch: 7 [51200/60000     (85%)]  Loss: 0.045149
Train Epoch: 7 [57600/60000     (96%)]  Loss: 0.024327

Test set: Average loss: 0.0221, Accuracy: 59611/60000 (99%)

Train Epoch: 8 [6400/60000      (11%)]  Loss: 0.009543
Train Epoch: 8 [12800/60000     (21%)]  Loss: 0.005252
Train Epoch: 8 [19200/60000     (32%)]  Loss: 0.001604
Train Epoch: 8 [25600/60000     (43%)]  Loss: 0.009577
Train Epoch: 8 [32000/60000     (53%)]  Loss: 0.009728
Train Epoch: 8 [38400/60000     (64%)]  Loss: 0.036994
Train Epoch: 8 [44800/60000     (75%)]  Loss: 0.014943
Train Epoch: 8 [51200/60000     (85%)]  Loss: 0.027597
Train Epoch: 8 [57600/60000     (96%)]  Loss: 0.004591
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
Train Epoch: 9 [6400/60000      (11%)]  Loss: [illegible]
Train Epoch: 9 [12800/60000     (21%)]  Loss: 0.002120
Train Epoch: 9 [19200/60000     (32%)]  Loss: 0.060184
Train Epoch: 9 [25600/60000     (43%)]  Loss: 0.003422
Train Epoch: 9 [32000/60000     (53%)]  Loss: 0.010903
Train Epoch: 9 [38400/60000     (64%)]  Loss: 0.037819
```

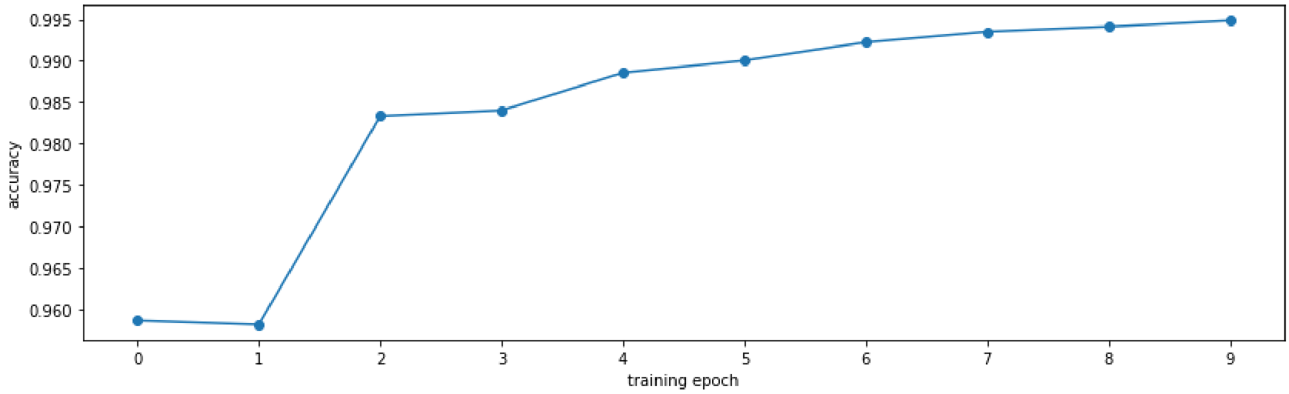
```
def mean(li): return sum(li)/len(li)
plt.figure(figsize=(14, 4))
plt.xlabel('training batch')
plt.ylabel('loss')
plt.plot([mean(losses[i:i+10]) for i in range(len(losses))])
```

```
[<matplotlib.lines.Line2D at 0x7fb3cdff9350>]
```



```
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 0x7fb3cd631cd0>]
```



```
def test_label_predictions(model, device, test_loader):
    model.eval()
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

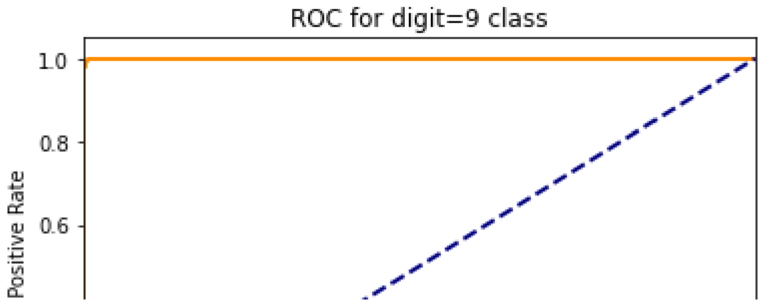
```
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            prediction = output.argmax(dim=1, keepdim=True)
            actuals.extend(target.view_as(prediction))
            predictions.extend(prediction)
    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:
[[ 977    0    0    0    0    0    0    1    2    0]
 [   0 1127    0    2    0    1    1    2    2    0]
 [   1    1 1024    0    1    0    0    2    2    1]
 [   1    0    0 1003    0    4    0    0    2    0]
 [   0    0    1    0  975    0    1    0    1    4]
 [   2    0    0    5    0  881    1    1    1    1]
 [   5    2    0    1    2    2  946    0    0    0]
 [   0    2    5    1    0    0    0 1019    1    0]
 [   4    0    2    0    0    3    0    1  962    2]
 [   2    2    0    4    8    4    0    6    5  978]]
F1 score: 0.989200
Accuracy score: 0.989200
```

```
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])
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

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

Automatic saving failed. This file was updated remotely or in another tab. Show diff

5m 11s completed at 4:39 AM