hw4_p3a

June 3, 2019

[9]:
```
%matplotlib inline
```

[10]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

[11]:
```
# LOADS CIFAR10 images which are 32 x 32 x 3 RGB images
# iter(trainloader/testloader) are iterables that come in pairs (image, label)

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)



testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[12]:
```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
```

```
        # in features --> flattened 32 x 32 x 3 image, out features = label␣
↪(0,1,..., 9), use bias
        self.linear = nn.Linear(3072, 10, bias=True)

    def forward(self, x):
        x = x.view(4, -1) # flatten the image 4 x 32 x 32 x 3 --> 4 x 3072 x 1
        return self.linear(x)


net = Net()
```

[ ]:

```

```

1 Choose Hyperparameters

[13]:

```
num_epochs = 12 # number of epochs to train for
momentum = 0.6 # momentum for Stochastic Gradient Descent
lr = 0.0001 # learning rate (eta) for gradient descent
```

2 3. Define a Loss function and optimizer

Let's use a Classification Cross-Entropy loss and SGD with momentum.

[14]:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=lr, momentum=momentum)
```

3 4. Train the network

This is when things start to get interesting. We simply have to loop over our data iterator, and feed the inputs to the network and optimize.

[15]:

```
def calc_accuracy(dataloader):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in dataloader:
            images, labels = data
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return 100.0 * correct / total
```

[16]:
```python
all_train_accuracies = [calc_accuracy(trainloader)]
all_test_accuracies = [calc_accuracy(testloader)]
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

            # Calculate accuracy every 2000 minibatches
            train_accuracy = calc_accuracy(trainloader)
            test_accuracy = calc_accuracy(testloader)
            all_train_accuracies.append(train_accuracy)
            all_test_accuracies.append(test_accuracy)

    print('END OF EPOCH ', epoch + 1, ': train accuracy = ', train_accuracy, ' /
 ↪/ test accuracy = ', test_accuracy)



print('Finished Training')
```

```
[1,  2000] loss: 2.046
[1,  4000] loss: 1.916
[1,  6000] loss: 1.867
[1,  8000] loss: 1.843
[1, 10000] loss: 1.836
[1, 12000] loss: 1.826
END OF EPOCH  1 : train accuracy =  38.652  // test accuracy =  38.15
[2,  2000] loss: 1.798
[2,  4000] loss: 1.788
```

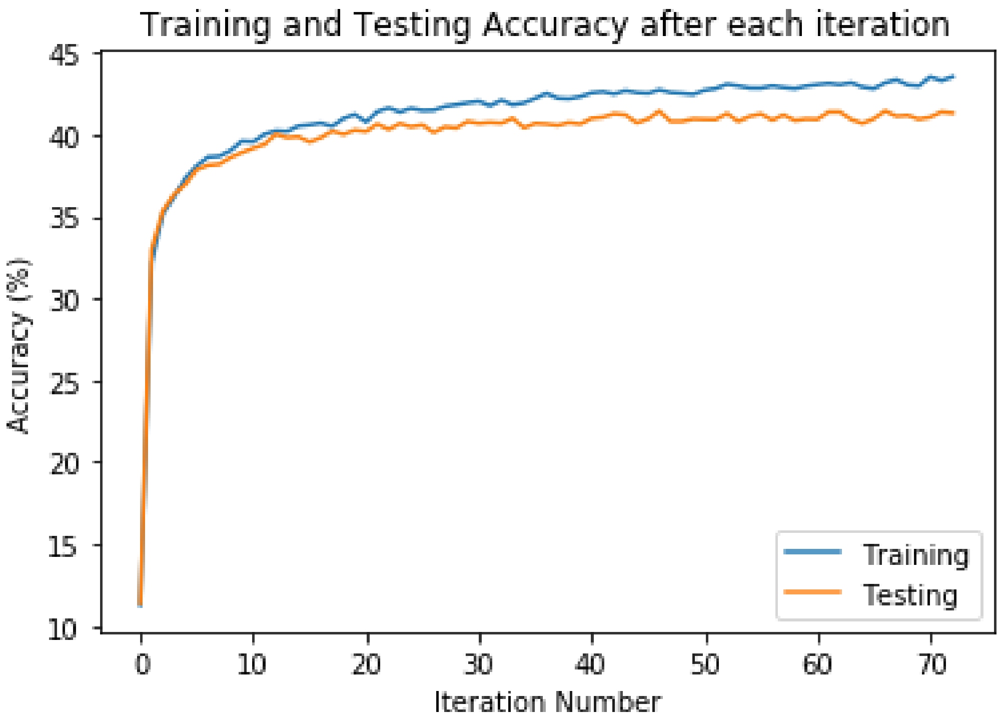
```
[2,  6000] loss: 1.789
[2,  8000] loss: 1.799
[2, 10000] loss: 1.789
[2, 12000] loss: 1.775
END OF EPOCH  2 : train accuracy =  40.252  // test accuracy =  40.04
[3,  2000] loss: 1.776
[3,  4000] loss: 1.759
[3,  6000] loss: 1.747
[3,  8000] loss: 1.770
[3, 10000] loss: 1.753
[3, 12000] loss: 1.756
END OF EPOCH  3 : train accuracy =  41.0  // test accuracy =  40.03
[4,  2000] loss: 1.732
[4,  4000] loss: 1.778
[4,  6000] loss: 1.745
[4,  8000] loss: 1.725
[4, 10000] loss: 1.736
[4, 12000] loss: 1.737
END OF EPOCH  4 : train accuracy =  41.628  // test accuracy =  40.49
[5,  2000] loss: 1.722
[5,  4000] loss: 1.733
[5,  6000] loss: 1.720
[5,  8000] loss: 1.736
[5, 10000] loss: 1.704
[5, 12000] loss: 1.741
END OF EPOCH  5 : train accuracy =  42.056  // test accuracy =  40.7
[6,  2000] loss: 1.710
[6,  4000] loss: 1.720
[6,  6000] loss: 1.727
[6,  8000] loss: 1.728
[6, 10000] loss: 1.704
[6, 12000] loss: 1.717
END OF EPOCH  6 : train accuracy =  42.536  // test accuracy =  40.66
[7,  2000] loss: 1.719
[7,  4000] loss: 1.714
[7,  6000] loss: 1.694
[7,  8000] loss: 1.710
[7, 10000] loss: 1.713
[7, 12000] loss: 1.708
END OF EPOCH  7 : train accuracy =  42.5  // test accuracy =  41.29
[8,  2000] loss: 1.707
[8,  4000] loss: 1.692
[8,  6000] loss: 1.715
[8,  8000] loss: 1.683
[8, 10000] loss: 1.702
[8, 12000] loss: 1.716
END OF EPOCH  8 : train accuracy =  42.544  // test accuracy =  40.82
[9,  2000] loss: 1.708
```

```
[9,  4000] loss: 1.692
[9,  6000] loss: 1.711
[9,  8000] loss: 1.700
[9, 10000] loss: 1.689
[9, 12000] loss: 1.670
END OF EPOCH  9 : train accuracy =  42.868  // test accuracy =  41.13
[10,  2000] loss: 1.695
[10,  4000] loss: 1.674
[10,  6000] loss: 1.703
[10,  8000] loss: 1.694
[10, 10000] loss: 1.687
[10, 12000] loss: 1.692
END OF EPOCH  10 : train accuracy =  43.078  // test accuracy =  40.95
[11,  2000] loss: 1.697
[11,  4000] loss: 1.700
[11,  6000] loss: 1.674
[11,  8000] loss: 1.681
[11, 10000] loss: 1.677
[11, 12000] loss: 1.685
END OF EPOCH  11 : train accuracy =  43.182  // test accuracy =  41.46
[12,  2000] loss: 1.694
[12,  4000] loss: 1.674
[12,  6000] loss: 1.688
[12,  8000] loss: 1.684
[12, 10000] loss: 1.675
[12, 12000] loss: 1.676
END OF EPOCH  12 : train accuracy =  43.556  // test accuracy =  41.33
Finished Training
```

4 Plot accuracy over time

[17]:
```python
import matplotlib.pyplot as plt

plt.figure(1)
plt.plot(all_train_accuracies)
plt.plot(all_test_accuracies)
plt.title('Training and Testing Accuracy after each iteration')
plt.xlabel('Iteration Number')
plt.ylabel('Accuracy (%)')
plt.legend(['Training', 'Testing'])
plt.show()
```



Training and Testing Accuracy after each iteration
45
40
35
30
25
20
15
10
Accuracy (%)
0
10
20
30
40
50
60
70
Iteration Number
Training
Testing

[ ]: