hw4_p3b

June 5, 2019

[1]:
```
%matplotlib inline
```

[2]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

[3]:
```
# 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
```

1 Choose Hyperparameters

[4]:
```
num_epochs = 15 # number of epochs to train for
momentum = 0.7 # momentum for Stochastic Gradient Descent
lr = 0.001 # learning rate (eta) for gradient descent
M = 200  # number of neurons in hidden layer of neural network
```

2 Build Neural Network

[5]:
```
import torch.nn as nn
import torch.nn.functional as F


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

        # in features --> flattened 32 x 32 x 3 image, out features = label␣
 ↪(0,1,..., 9), use bias
        self.linear1 = nn.Linear(3072, M, bias=True)
        self.linear2 = nn.Linear(M, 10, bias=True)

    def forward(self, x):
        x = x.view(4, -1)
        x = self.linear1(x)
        x = F.relu(x)
        return self.linear2(x)


net = Net(M)
```

[ ]:
```

```

3 3. Define a Loss function and optimizer

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

[6]:
```
import torch.optim as optim

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

4 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.

[7]:
```
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
```

[8]:
```
all_train_accuracies = []
all_test_accuracies = []
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

            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: 1.912
[1,  4000] loss: 1.716
[1,  6000] loss: 1.658
[1,  8000] loss: 1.631
[1, 10000] loss: 1.603
```

```
[1, 12000] loss: 1.584
END OF EPOCH  1 : train accuracy =  46.69  // test accuracy =  45.28
[2,  2000] loss: 1.524
[2,  4000] loss: 1.510
[2,  6000] loss: 1.474
[2,  8000] loss: 1.463
[2, 10000] loss: 1.460
[2, 12000] loss: 1.432
END OF EPOCH  2 : train accuracy =  51.152  // test accuracy =  48.19
[3,  2000] loss: 1.394
[3,  4000] loss: 1.390
[3,  6000] loss: 1.386
[3,  8000] loss: 1.389
[3, 10000] loss: 1.371
[3, 12000] loss: 1.367
END OF EPOCH  3 : train accuracy =  54.648  // test accuracy =  50.4
[4,  2000] loss: 1.300
[4,  4000] loss: 1.310
[4,  6000] loss: 1.318
[4,  8000] loss: 1.313
[4, 10000] loss: 1.325
[4, 12000] loss: 1.321
END OF EPOCH  4 : train accuracy =  57.4  // test accuracy =  51.49
[5,  2000] loss: 1.241
[5,  4000] loss: 1.289
[5,  6000] loss: 1.256
[5,  8000] loss: 1.233
[5, 10000] loss: 1.279
[5, 12000] loss: 1.256
END OF EPOCH  5 : train accuracy =  60.178  // test accuracy =  52.21
[6,  2000] loss: 1.207
[6,  4000] loss: 1.195
[6,  6000] loss: 1.204
[6,  8000] loss: 1.210
[6, 10000] loss: 1.222
[6, 12000] loss: 1.227
END OF EPOCH  6 : train accuracy =  60.5  // test accuracy =  51.89
[7,  2000] loss: 1.130
[7,  4000] loss: 1.149
[7,  6000] loss: 1.170
[7,  8000] loss: 1.193
[7, 10000] loss: 1.164
[7, 12000] loss: 1.191
END OF EPOCH  7 : train accuracy =  61.702  // test accuracy =  51.35
[8,  2000] loss: 1.090
[8,  4000] loss: 1.118
[8,  6000] loss: 1.132
[8,  8000] loss: 1.125
```

```
[8, 10000] loss: 1.133
[8, 12000] loss: 1.158
END OF EPOCH  8 : train accuracy =  64.118  // test accuracy =  52.27
[9,  2000] loss: 1.087
[9,  4000] loss: 1.087
[9,  6000] loss: 1.075
[9,  8000] loss: 1.097
[9, 10000] loss: 1.090
[9, 12000] loss: 1.109
END OF EPOCH  9 : train accuracy =  64.516  // test accuracy =  52.07
[10,  2000] loss: 1.030
[10,  4000] loss: 1.057
[10,  6000] loss: 1.038
[10,  8000] loss: 1.052
[10, 10000] loss: 1.097
[10, 12000] loss: 1.071
END OF EPOCH  10 : train accuracy =  66.502  // test accuracy =  52.73
[11,  2000] loss: 1.008
[11,  4000] loss: 1.016
[11,  6000] loss: 1.006
[11,  8000] loss: 1.033
[11, 10000] loss: 1.032
[11, 12000] loss: 1.057
END OF EPOCH  11 : train accuracy =  68.582  // test accuracy =  52.41
[12,  2000] loss: 0.937
[12,  4000] loss: 0.962
[12,  6000] loss: 0.985
[12,  8000] loss: 1.021
[12, 10000] loss: 0.993
[12, 12000] loss: 1.038
END OF EPOCH  12 : train accuracy =  68.81  // test accuracy =  52.11
[13,  2000] loss: 0.926
[13,  4000] loss: 0.943
[13,  6000] loss: 0.963
[13,  8000] loss: 0.973
[13, 10000] loss: 0.969
[13, 12000] loss: 1.007
END OF EPOCH  13 : train accuracy =  69.066  // test accuracy =  51.63
[14,  2000] loss: 0.906
[14,  4000] loss: 0.904
[14,  6000] loss: 0.929
[14,  8000] loss: 0.962
[14, 10000] loss: 0.952
[14, 12000] loss: 0.952
END OF EPOCH  14 : train accuracy =  71.012  // test accuracy =  51.68
[15,  2000] loss: 0.867
[15,  4000] loss: 0.894
[15,  6000] loss: 0.905
```

```
[15,  8000] loss: 0.915
[15, 10000] loss: 0.932
[15, 12000] loss: 0.921
END OF EPOCH  15 : train accuracy =  69.858  // test accuracy =  50.2
Finished Training
```

5 Plot accuracy over time

[9]:
```
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()
```