hw4_p3c

May 31, 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 = 12 # number of epochs to train for
momentum = 0.9 # momentum for Stochastic Gradient Descent
lr = 0.001 # learning rate (eta) for gradient descent
M = 100  # number of neurons in hidden layer of neural network
p = 5  # filter window size
```

```
N = 14
```

2 Build Neural Network

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

# NO PADDING
class Net(nn.Module):
    def __init__(self, M, N, p):
        super(Net, self).__init__()

        # The 3 input channels are the RGB of the image
        # the Kernel size is the size of the filter window (p x p in this case)
        self.conv = nn.Conv2d(in_channels=3, out_channels=M, kernel_size=p,␣
 ↪bias=True, padding=0)

        # Max pooling layer. kernel_size is the size of the window that is used.
 ↪ Max is selected within a N x N window
        self.pool = nn.MaxPool2d(kernel_size=N)

        self.linear = nn.Linear(((33-p) // N)**2 * M,  10, bias=True)

    def forward(self, x):
        x = self.conv(x)
        x = F.relu(x)
        x = self.pool(x)

        # vectorize the data before feeding it into the linear layer. Note the␣
 ↪4 is due to the batch of size 4 used
        # (see homework specification)
        x = x.view(4, -1)
        x = self.linear(x)
        return x



net = Net(M, N, p)
```

[ ]:
```

```

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 = [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
```

```python
            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: 1.957
[1,  4000] loss: 1.686
[1,  6000] loss: 1.548
[1,  8000] loss: 1.492
[1, 10000] loss: 1.421
[1, 12000] loss: 1.402
END OF EPOCH  1 : train accuracy =  51.684  // test accuracy =  50.69
[2,  2000] loss: 1.311
[2,  4000] loss: 1.311
[2,  6000] loss: 1.306
[2,  8000] loss: 1.282
[2, 10000] loss: 1.281
[2, 12000] loss: 1.272
END OF EPOCH  2 : train accuracy =  54.696  // test accuracy =  54.02
[3,  2000] loss: 1.226
[3,  4000] loss: 1.227
[3,  6000] loss: 1.208
[3,  8000] loss: 1.195
[3, 10000] loss: 1.197
[3, 12000] loss: 1.203
END OF EPOCH  3 : train accuracy =  59.294  // test accuracy =  57.5
[4,  2000] loss: 1.147
[4,  4000] loss: 1.185
[4,  6000] loss: 1.180
[4,  8000] loss: 1.157
[4, 10000] loss: 1.137
[4, 12000] loss: 1.137
END OF EPOCH  4 : train accuracy =  62.796  // test accuracy =  61.09
[5,  2000] loss: 1.111
[5,  4000] loss: 1.127
[5,  6000] loss: 1.134
[5,  8000] loss: 1.119
[5, 10000] loss: 1.102
[5, 12000] loss: 1.131
END OF EPOCH  5 : train accuracy =  63.918  // test accuracy =  61.58
```

```
[6,  2000] loss: 1.106
[6,  4000] loss: 1.080
[6,  6000] loss: 1.091
[6,  8000] loss: 1.089
[6, 10000] loss: 1.122
[6, 12000] loss: 1.089
END OF EPOCH  6 : train accuracy =  63.458  // test accuracy =  61.53
[7,  2000] loss: 1.081
[7,  4000] loss: 1.083
[7,  6000] loss: 1.068
[7,  8000] loss: 1.065
[7, 10000] loss: 1.079
[7, 12000] loss: 1.084
END OF EPOCH  7 : train accuracy =  65.574  // test accuracy =  62.54
[8,  2000] loss: 1.072
[8,  4000] loss: 1.048
[8,  6000] loss: 1.068
[8,  8000] loss: 1.038
[8, 10000] loss: 1.096
[8, 12000] loss: 1.059
END OF EPOCH  8 : train accuracy =  64.188  // test accuracy =  61.77
[9,  2000] loss: 1.040
[9,  4000] loss: 1.030
[9,  6000] loss: 1.051
[9,  8000] loss: 1.036
[9, 10000] loss: 1.025
[9, 12000] loss: 1.083
END OF EPOCH  9 : train accuracy =  64.914  // test accuracy =  62.21
[10,  2000] loss: 1.018
[10,  4000] loss: 1.016
[10,  6000] loss: 1.031
[10,  8000] loss: 1.056
[10, 10000] loss: 1.030
[10, 12000] loss: 1.038
END OF EPOCH  10 : train accuracy =  66.956  // test accuracy =  63.83
[11,  2000] loss: 1.017
[11,  4000] loss: 1.031
[11,  6000] loss: 1.014
[11,  8000] loss: 1.018
[11, 10000] loss: 1.031
[11, 12000] loss: 1.028
END OF EPOCH  11 : train accuracy =  64.89  // test accuracy =  62.11
[12,  2000] loss: 1.001
[12,  4000] loss: 1.021
[12,  6000] loss: 0.991
[12,  8000] loss: 1.011
[12, 10000] loss: 1.024
[12, 12000] loss: 1.033
```

```
END OF EPOCH  12 : train accuracy =  65.5  // test accuracy =  62.06
Finished Training
```

5 Plot accuracy over time

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