CIFAR10_4Layer_2RELU_2Linear_CNN

October 9, 2019

[110]:
```
import numpy as np
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import time
import random
import itertools
from datetime import datetime
import torch.nn.functional as F
```

[111]:
```
# set random seeds for reproducibility
torch.manual_seed(12)
torch.cuda.manual_seed(12)
np.random.seed(12)
```

[112]:
```
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# If we are on a CUDA machine, then this should print a CUDA device, but we are␣
 ↪not, so it will run on CPU:
print(f'Working on device={device}')
```

```
Working on device=cpu
```

[113]:
```
# Hyper-parameters

# each CIFAR image is RGB 32x32, so it is an 3D array [3,32,32]
# we will flatten the image as vector dim=3*32*32
input_size = 3*32*32

# we have 10 classes
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

num_classes = 10

num_epochs = 50
```

```
batch_size = 128

learning_rate = 0.01
```

[114]:
```
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=batch_size,
                                          shuffle=True)

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

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

[115]:
```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()



        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 10)
        #self.fc3 = nn.Linear(84, 10)

    def forward(self, x):

        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = self.fc1(x)
        x = self.fc2(x)
        return x


net = ConvNet()
net.to(device)
```

[115]:
```
ConvNet(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=10, bias=True)
)
```

[116]:
```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)
```

[117]:
```
for epoch in range(num_epochs):  # loop over the dataset multiple times

    start_time = datetime.now()
    net.train()
    running_loss = 0.0
    epoch_loss = 0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data
        # move data to device (GPU if enabled, else CPU do nothing)
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

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

        # print statistics
        epoch_loss += loss.item()

    epoch_loss = epoch_loss / len(trainloader)

    time_elapsed = datetime.now() - start_time

    # Test the model

    # set our model in the training mode
    net.eval()
    # In test phase, we don't need to compute gradients (for memory efficiency)
    correct = 0
    total = 0
```

```
    with torch.no_grad():
        for data in testloader:
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = net(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    # Accuracy of the network on the 10000 test images
    acc = correct/total
    print(
        f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f} Test acc:␣
↪{acc} time={time_elapsed}')


print('Finished Training')
```

```
Epoch [1/50], Loss: 1.7759 Test acc: 0.3931 time=0:00:28.687709
Epoch [2/50], Loss: 1.5782 Test acc: 0.4183 time=0:00:30.382453
Epoch [3/50], Loss: 1.5346 Test acc: 0.4596 time=0:00:29.551568
Epoch [4/50], Loss: 1.5090 Test acc: 0.4697 time=0:00:29.328534
Epoch [5/50], Loss: 1.4796 Test acc: 0.4432 time=0:00:29.057410
Epoch [6/50], Loss: 1.4781 Test acc: 0.4563 time=0:00:29.598949
Epoch [7/50], Loss: 1.4636 Test acc: 0.4714 time=0:00:29.765057
Epoch [8/50], Loss: 1.4496 Test acc: 0.4426 time=0:00:30.251707
Epoch [9/50], Loss: 1.4493 Test acc: 0.4601 time=0:00:29.903571
Epoch [10/50], Loss: 1.4262 Test acc: 0.4691 time=0:00:29.858458
Epoch [11/50], Loss: 1.4281 Test acc: 0.4839 time=0:00:30.007587
Epoch [12/50], Loss: 1.4262 Test acc: 0.4677 time=0:00:30.028156
Epoch [13/50], Loss: 1.4232 Test acc: 0.4872 time=0:00:29.919388
Epoch [14/50], Loss: 1.4071 Test acc: 0.4898 time=0:00:29.786341
Epoch [15/50], Loss: 1.4077 Test acc: 0.4894 time=0:00:30.037568
Epoch [16/50], Loss: 1.3872 Test acc: 0.4873 time=0:00:29.751080
Epoch [17/50], Loss: 1.4069 Test acc: 0.4824 time=0:00:30.167744
Epoch [18/50], Loss: 1.4020 Test acc: 0.4827 time=0:00:29.877461
Epoch [19/50], Loss: 1.3987 Test acc: 0.4809 time=0:00:29.810680
Epoch [20/50], Loss: 1.3884 Test acc: 0.4777 time=0:00:29.865915
Epoch [21/50], Loss: 1.3939 Test acc: 0.4895 time=0:00:29.827584
Epoch [22/50], Loss: 1.3790 Test acc: 0.4972 time=0:00:29.843010
Epoch [23/50], Loss: 1.3901 Test acc: 0.5005 time=0:00:29.695477
Epoch [24/50], Loss: 1.3722 Test acc: 0.4908 time=0:00:29.850679
Epoch [25/50], Loss: 1.3815 Test acc: 0.4997 time=0:00:29.678224
Epoch [26/50], Loss: 1.3764 Test acc: 0.4869 time=0:00:30.381953
Epoch [27/50], Loss: 1.3717 Test acc: 0.4812 time=0:00:29.756049
Epoch [28/50], Loss: 1.3720 Test acc: 0.4741 time=0:00:29.932697
Epoch [29/50], Loss: 1.3708 Test acc: 0.4975 time=0:00:29.950650
```

```
Epoch [30/50], Loss: 1.3699 Test acc: 0.4873 time=0:00:29.826411
Epoch [31/50], Loss: 1.3741 Test acc: 0.4957 time=0:00:29.873370
Epoch [32/50], Loss: 1.3733 Test acc: 0.4416 time=0:00:29.894804
Epoch [33/50], Loss: 1.3709 Test acc: 0.5025 time=0:00:29.856453
Epoch [34/50], Loss: 1.3715 Test acc: 0.4881 time=0:00:30.012619
Epoch [35/50], Loss: 1.3786 Test acc: 0.479 time=0:00:30.259144
Epoch [36/50], Loss: 1.3772 Test acc: 0.4902 time=0:00:29.993506
Epoch [37/50], Loss: 1.3654 Test acc: 0.4766 time=0:00:30.053988
Epoch [38/50], Loss: 1.3655 Test acc: 0.4849 time=0:00:28.867508
Epoch [39/50], Loss: 1.3615 Test acc: 0.4879 time=0:00:29.511596
Epoch [40/50], Loss: 1.3578 Test acc: 0.4878 time=0:00:28.975814
Epoch [41/50], Loss: 1.3640 Test acc: 0.4802 time=0:00:28.662469
Epoch [42/50], Loss: 1.3542 Test acc: 0.4867 time=0:00:28.731599
Epoch [43/50], Loss: 1.3638 Test acc: 0.4929 time=0:00:28.534095
Epoch [44/50], Loss: 1.3549 Test acc: 0.4917 time=0:00:28.818203
Epoch [45/50], Loss: 1.3662 Test acc: 0.4906 time=0:00:28.853884
Epoch [46/50], Loss: 1.3607 Test acc: 0.5111 time=0:00:28.510313
Epoch [47/50], Loss: 1.3629 Test acc: 0.4763 time=0:00:28.666454
Epoch [48/50], Loss: 1.3645 Test acc: 0.4835 time=0:00:29.281780
Epoch [49/50], Loss: 1.3543 Test acc: 0.4896 time=0:00:28.715684
Epoch [50/50], Loss: 1.3613 Test acc: 0.5007 time=0:00:28.729147
Finished Training
```

[ ]:
```python
# Detailed accuracy per class
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = net(inputs)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
                label = labels[i]
                class_correct[label] += c[i].item()
                class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

[ ]:
```python

```