3CNN, AvgPool, ReLU, Batch

March 25, 2024

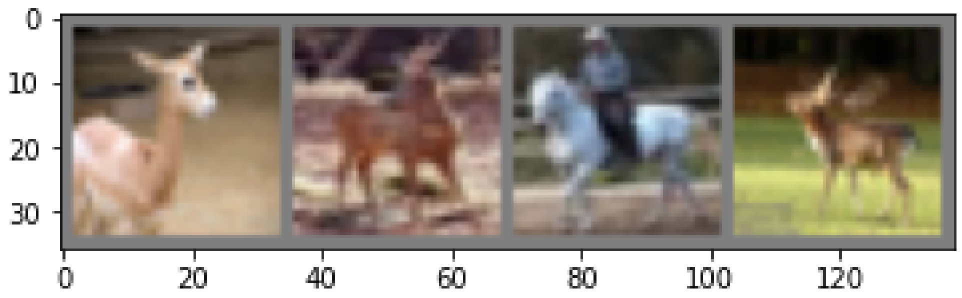
[2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

Prepare for Dataset

[6]:
```
transform = transforms.Compose(
    [#transforms.RandomHorizontalFlip(),
     #transforms.RandomRotation(10),
     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, #Increase␣
 ↪batch size
                                          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
```

[7]:
```
def imshow(img):
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

dataiter = iter(trainloader)
images, labels = next(dataiter)
imshow(torchvision.utils.make_grid(images))
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```



```
 deer  deer horse  deer
```

Choose a Device

[8]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

Network Definition

[9]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.convo1 = nn.Conv2d(3,32,3,1,1)
        self.batch1 = nn.BatchNorm2d(32) #Add batch normalization to each␣
 ↪convolution block
        self.activation1 = nn.ReLU()
        self.apool1 = nn.AvgPool2d(kernel_size = (2,2))

        self.convo2 = nn.Conv2d(32,16,3,1,1)
        self.batch3 = nn.BatchNorm2d(16)
        self.activation2 = nn.ReLU()
        self.apool2 = nn.AvgPool2d(kernel_size = (2,2))

        self.convo3 = nn.Conv2d(16,64,3,1,1) #Add a 3rd convolution block
        self.batch3 = nn.BatchNorm2d(64)
        self.activation3 = nn.ReLU()
        self.apool3 = nn.AvgPool2d(kernel_size = (2,2))

        self.flat = nn.Flatten()

        self.fc1 = nn.Linear(1024,100)
        self.activation3 = nn.ReLU()
        self.fc2 = nn.Linear(100,10)


    def forward(self, x):

        x = self.activation1(self.convo1(x))
        x = self.apool1(x)
        x = self.activation2(self.convo2(x))
        x = self.apool2(x)
        x = self.activation3(self.convo3(x))
        x = self.apool3(x)
        x = self.flat(x)
        x = self.fc1(self.activation3(x))
        x = self.fc2(x)


        return x

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

```
[9]: Net(
       (convo1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (batch1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
       (activation1): ReLU()
       (apool1): AvgPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
       (convo2): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (batch3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
       (activation2): ReLU()
       (apool2): AvgPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
       (convo3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (activation3): ReLU()
       (apool3): AvgPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0)
       (flat): Flatten(start_dim=1, end_dim=-1)
       (fc1): Linear(in_features=1024, out_features=100, bias=True)
       (fc2): Linear(in_features=100, out_features=10, bias=True)
     )
```

Optimizer and Loss Function

[10]:
```python
loss_func = nn.CrossEntropyLoss()
#Replace stochastic gradient decent with Adam algorithm, no momentum needed
#Adjust the learning rate
opt = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Training Procedure

[11]:
```python
avg_losses = []
epochs = 15     #Increase epochs from 10 to 20
print_freq = 1500 #Increase print frequency

for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):

        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        opt.zero_grad()
        outputs = net(inputs)
        loss = loss_func(outputs, labels)
        loss.backward()
        opt.step()

        running_loss += loss.item()
        if i % print_freq == print_freq - 1:
            avg_loss = running_loss / print_freq
            print('[epoch: {}, i: {:5d}] avg mini-batch loss: {:.3f}'.format(
                epoch, i, avg_loss))
            avg_losses.append(avg_loss)
            running_loss = 0.0

print('Finished Training.')
```

```
[epoch: 0, i:  1499] avg mini-batch loss: 2.183
[epoch: 0, i:  2999] avg mini-batch loss: 1.913
[epoch: 0, i:  4499] avg mini-batch loss: 1.744
[epoch: 0, i:  5999] avg mini-batch loss: 1.648
[epoch: 0, i:  7499] avg mini-batch loss: 1.550
[epoch: 0, i:  8999] avg mini-batch loss: 1.502
[epoch: 0, i: 10499] avg mini-batch loss: 1.421
[epoch: 0, i: 11999] avg mini-batch loss: 1.411
[epoch: 1, i:  1499] avg mini-batch loss: 1.325
[epoch: 1, i:  2999] avg mini-batch loss: 1.301
[epoch: 1, i:  4499] avg mini-batch loss: 1.282
[epoch: 1, i:  5999] avg mini-batch loss: 1.261
[epoch: 1, i:  7499] avg mini-batch loss: 1.223
[epoch: 1, i:  8999] avg mini-batch loss: 1.202
[epoch: 1, i: 10499] avg mini-batch loss: 1.208
```

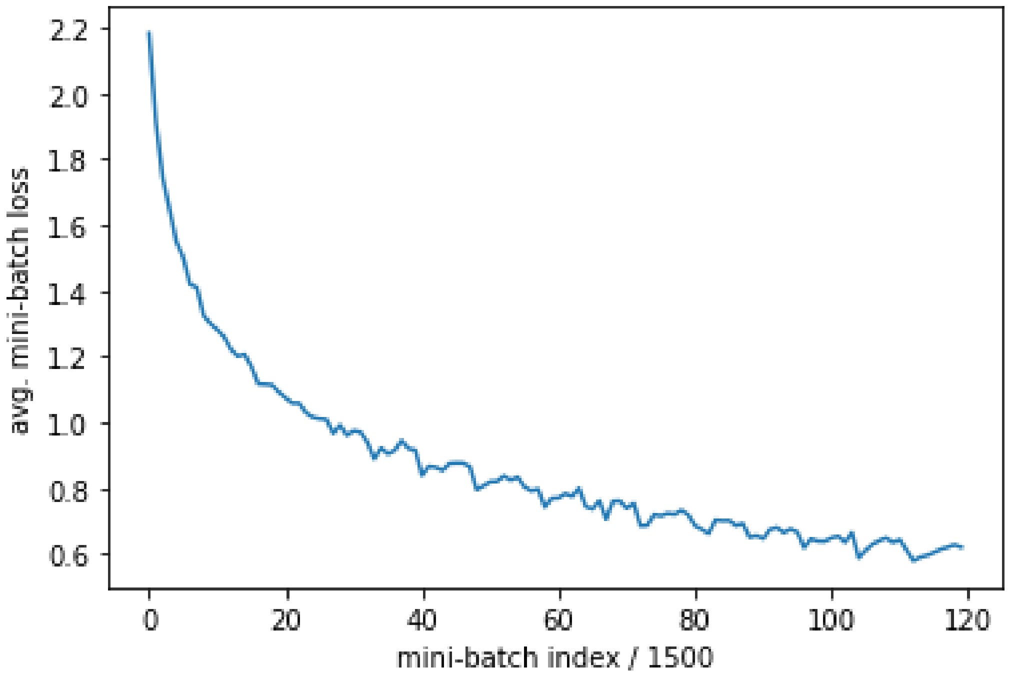
```
[epoch: 1, i: 11999] avg mini-batch loss: 1.170
[epoch: 2, i:  1499] avg mini-batch loss: 1.118
[epoch: 2, i:  2999] avg mini-batch loss: 1.116
[epoch: 2, i:  4499] avg mini-batch loss: 1.115
[epoch: 2, i:  5999] avg mini-batch loss: 1.093
[epoch: 2, i:  7499] avg mini-batch loss: 1.075
[epoch: 2, i:  8999] avg mini-batch loss: 1.059
[epoch: 2, i: 10499] avg mini-batch loss: 1.059
[epoch: 2, i: 11999] avg mini-batch loss: 1.030
[epoch: 3, i:  1499] avg mini-batch loss: 1.016
[epoch: 3, i:  2999] avg mini-batch loss: 1.012
[epoch: 3, i:  4499] avg mini-batch loss: 1.010
[epoch: 3, i:  5999] avg mini-batch loss: 0.967
[epoch: 3, i:  7499] avg mini-batch loss: 0.993
[epoch: 3, i:  8999] avg mini-batch loss: 0.961
[epoch: 3, i: 10499] avg mini-batch loss: 0.975
[epoch: 3, i: 11999] avg mini-batch loss: 0.971
[epoch: 4, i:  1499] avg mini-batch loss: 0.940
[epoch: 4, i:  2999] avg mini-batch loss: 0.890
[epoch: 4, i:  4499] avg mini-batch loss: 0.923
[epoch: 4, i:  5999] avg mini-batch loss: 0.905
[epoch: 4, i:  7499] avg mini-batch loss: 0.916
[epoch: 4, i:  8999] avg mini-batch loss: 0.946
[epoch: 4, i: 10499] avg mini-batch loss: 0.921
[epoch: 4, i: 11999] avg mini-batch loss: 0.916
[epoch: 5, i:  1499] avg mini-batch loss: 0.840
[epoch: 5, i:  2999] avg mini-batch loss: 0.867
[epoch: 5, i:  4499] avg mini-batch loss: 0.864
[epoch: 5, i:  5999] avg mini-batch loss: 0.854
[epoch: 5, i:  7499] avg mini-batch loss: 0.876
[epoch: 5, i:  8999] avg mini-batch loss: 0.877
[epoch: 5, i: 10499] avg mini-batch loss: 0.877
[epoch: 5, i: 11999] avg mini-batch loss: 0.866
[epoch: 6, i:  1499] avg mini-batch loss: 0.796
[epoch: 6, i:  2999] avg mini-batch loss: 0.807
[epoch: 6, i:  4499] avg mini-batch loss: 0.820
[epoch: 6, i:  5999] avg mini-batch loss: 0.821
[epoch: 6, i:  7499] avg mini-batch loss: 0.838
[epoch: 6, i:  8999] avg mini-batch loss: 0.825
[epoch: 6, i: 10499] avg mini-batch loss: 0.834
[epoch: 6, i: 11999] avg mini-batch loss: 0.805
[epoch: 7, i:  1499] avg mini-batch loss: 0.792
[epoch: 7, i:  2999] avg mini-batch loss: 0.798
[epoch: 7, i:  4499] avg mini-batch loss: 0.743
[epoch: 7, i:  5999] avg mini-batch loss: 0.770
[epoch: 7, i:  7499] avg mini-batch loss: 0.771
[epoch: 7, i:  8999] avg mini-batch loss: 0.784
[epoch: 7, i: 10499] avg mini-batch loss: 0.776
```

```
[epoch: 7, i: 11999] avg mini-batch loss: 0.801
[epoch: 8, i:  1499] avg mini-batch loss: 0.745
[epoch: 8, i:  2999] avg mini-batch loss: 0.737
[epoch: 8, i:  4499] avg mini-batch loss: 0.763
[epoch: 8, i:  5999] avg mini-batch loss: 0.707
[epoch: 8, i:  7499] avg mini-batch loss: 0.761
[epoch: 8, i:  8999] avg mini-batch loss: 0.762
[epoch: 8, i: 10499] avg mini-batch loss: 0.739
[epoch: 8, i: 11999] avg mini-batch loss: 0.755
[epoch: 9, i:  1499] avg mini-batch loss: 0.687
[epoch: 9, i:  2999] avg mini-batch loss: 0.688
[epoch: 9, i:  4499] avg mini-batch loss: 0.721
[epoch: 9, i:  5999] avg mini-batch loss: 0.717
[epoch: 9, i:  7499] avg mini-batch loss: 0.724
[epoch: 9, i:  8999] avg mini-batch loss: 0.721
[epoch: 9, i: 10499] avg mini-batch loss: 0.733
[epoch: 9, i: 11999] avg mini-batch loss: 0.720
[epoch: 10, i:  1499] avg mini-batch loss: 0.687
[epoch: 10, i:  2999] avg mini-batch loss: 0.676
[epoch: 10, i:  4499] avg mini-batch loss: 0.663
[epoch: 10, i:  5999] avg mini-batch loss: 0.705
[epoch: 10, i:  7499] avg mini-batch loss: 0.702
[epoch: 10, i:  8999] avg mini-batch loss: 0.703
[epoch: 10, i: 10499] avg mini-batch loss: 0.687
[epoch: 10, i: 11999] avg mini-batch loss: 0.694
[epoch: 11, i:  1499] avg mini-batch loss: 0.652
[epoch: 11, i:  2999] avg mini-batch loss: 0.657
[epoch: 11, i:  4499] avg mini-batch loss: 0.648
[epoch: 11, i:  5999] avg mini-batch loss: 0.676
[epoch: 11, i:  7499] avg mini-batch loss: 0.682
[epoch: 11, i:  8999] avg mini-batch loss: 0.666
[epoch: 11, i: 10499] avg mini-batch loss: 0.678
[epoch: 11, i: 11999] avg mini-batch loss: 0.667
[epoch: 12, i:  1499] avg mini-batch loss: 0.620
[epoch: 12, i:  2999] avg mini-batch loss: 0.647
[epoch: 12, i:  4499] avg mini-batch loss: 0.640
[epoch: 12, i:  5999] avg mini-batch loss: 0.639
[epoch: 12, i:  7499] avg mini-batch loss: 0.650
[epoch: 12, i:  8999] avg mini-batch loss: 0.655
[epoch: 12, i: 10499] avg mini-batch loss: 0.637
[epoch: 12, i: 11999] avg mini-batch loss: 0.666
[epoch: 13, i:  1499] avg mini-batch loss: 0.589
[epoch: 13, i:  2999] avg mini-batch loss: 0.612
[epoch: 13, i:  4499] avg mini-batch loss: 0.630
[epoch: 13, i:  5999] avg mini-batch loss: 0.641
[epoch: 13, i:  7499] avg mini-batch loss: 0.650
[epoch: 13, i:  8999] avg mini-batch loss: 0.636
[epoch: 13, i: 10499] avg mini-batch loss: 0.645
```

```
[epoch: 13, i: 11999] avg mini-batch loss: 0.612
[epoch: 14, i:  1499] avg mini-batch loss: 0.580
[epoch: 14, i:  2999] avg mini-batch loss: 0.591
[epoch: 14, i:  4499] avg mini-batch loss: 0.597
[epoch: 14, i:  5999] avg mini-batch loss: 0.605
[epoch: 14, i:  7499] avg mini-batch loss: 0.615
[epoch: 14, i:  8999] avg mini-batch loss: 0.622
[epoch: 14, i: 10499] avg mini-batch loss: 0.629
[epoch: 14, i: 11999] avg mini-batch loss: 0.622
Finished Training.
```

Training Loss Curve

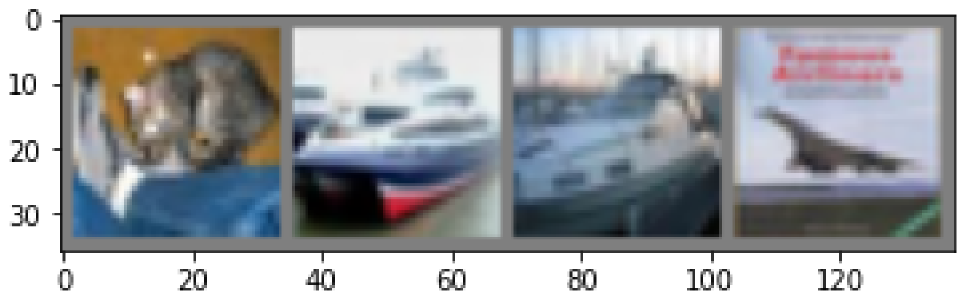
[12]:
```
plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```



Evaluate on Test Dataset

[13]:
```
# Check several images.
dataiter = iter(testloader)
images, labels = next(dataiter)
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
outputs = net(images.to(device))
```

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```



```
GroundTruth:    cat  ship  ship plane
Predicted:    cat  ship  ship plane
```

[14]:
```
# Get test accuracy.
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 72 %
```

[15]:
```
# Get test accuracy for each 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:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, 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]))
```

```
Accuracy of plane : 68 %
Accuracy of   car : 86 %
Accuracy of  bird : 64 %
Accuracy of   cat : 58 %
Accuracy of  deer : 61 %
Accuracy of   dog : 59 %
Accuracy of  frog : 75 %
Accuracy of horse : 77 %
Accuracy of  ship : 89 %
Accuracy of truck : 81 %
```

[ ]:

```
```