2CNN, AvgPool, ReLU, Batch, SGD

March 25, 2024

[1]:
```
%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
```

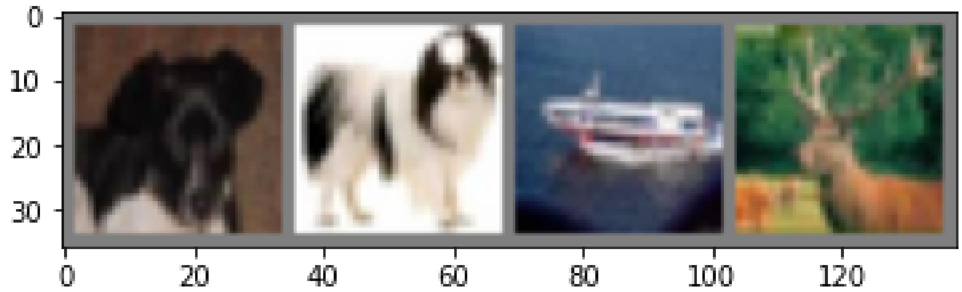
Prepare for Dataset

[2]:
```
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
```

[3]:
```
# The function to show an image.
def imshow(img):
    img = img / 2 + 0.5     # Unnormalize.
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

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




```
  dog   dog  ship  deer
```

Choose a Device

[4]:
```
# If there are GPUs, choose the first one for computing. Otherwise use CPU.
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
# If 'cuda:0' is printed, it means GPU is available.
```

```
cuda:0
```

Network Definition

[5]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        ###### Fill the blank here ######

        self.convo1 = nn.Conv2d(3,32,3,1,1)
        self.batch1 = nn.BatchNorm2d(32)
        self.activation1 = nn.ReLU()
        self.apool1 = nn.AvgPool2d(kernel_size = (2,2))

        self.convo2 = nn.Conv2d(32,16,3,1,1)
        self.batch1 = nn.BatchNorm2d(16)
        self.activation2 = nn.ReLU()
        self.apool2 = nn.AvgPool2d(kernel_size = (2,2))
```

```python
        self.flat = nn.Flatten()

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


    def forward(self, x):
        ###### Fill the blank here ######

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


        return x

net = Net()     # Create the network instance.
net.to(device)  # Move the network parameters to the specified device.
```

[5]:
```
Net(
  (convo1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batch1): BatchNorm2d(16, 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))
  (activation2): ReLU()
  (apool2): 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)
  (activation3): ReLU()
  (fc2): Linear(in_features=100, out_features=10, bias=True)
)
```

Optimizer and Loss Function

[6]:
```python
# We use cross-entropy as loss function.
loss_func = nn.CrossEntropyLoss()
# We use stochastic gradient descent (SGD) as optimizer.
opt = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Training Procedure

[7]:
```python
avg_losses = []   # Avg. losses.
epochs = 15       # Total epochs.
print_freq = 1500  # Print frequency.

for epoch in range(epochs):  # Loop over the dataset multiple times.
    running_loss = 0.0       # Initialize running loss.
    for i, data in enumerate(trainloader, 0):
        # Get the inputs.
        inputs, labels = data

        # Move the inputs to the specified device.
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients.
        opt.zero_grad()

        # Forward step.
        outputs = net(inputs)
        loss = loss_func(outputs, labels)

        # Backward step.
        loss.backward()

        # Optimization step (update the parameters).
        opt.step()

        # Print statistics.
        running_loss += loss.item()
        if i % print_freq == print_freq - 1: # Print every several mini-batches.
            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: 2.063
[epoch: 0, i:  2999] avg mini-batch loss: 1.764
[epoch: 0, i:  4499] avg mini-batch loss: 1.637
[epoch: 0, i:  5999] avg mini-batch loss: 1.566
[epoch: 0, i:  7499] avg mini-batch loss: 1.488
[epoch: 0, i:  8999] avg mini-batch loss: 1.442
[epoch: 0, i: 10499] avg mini-batch loss: 1.404
[epoch: 0, i: 11999] avg mini-batch loss: 1.363
[epoch: 1, i:  1499] avg mini-batch loss: 1.291
[epoch: 1, i:  2999] avg mini-batch loss: 1.282
```

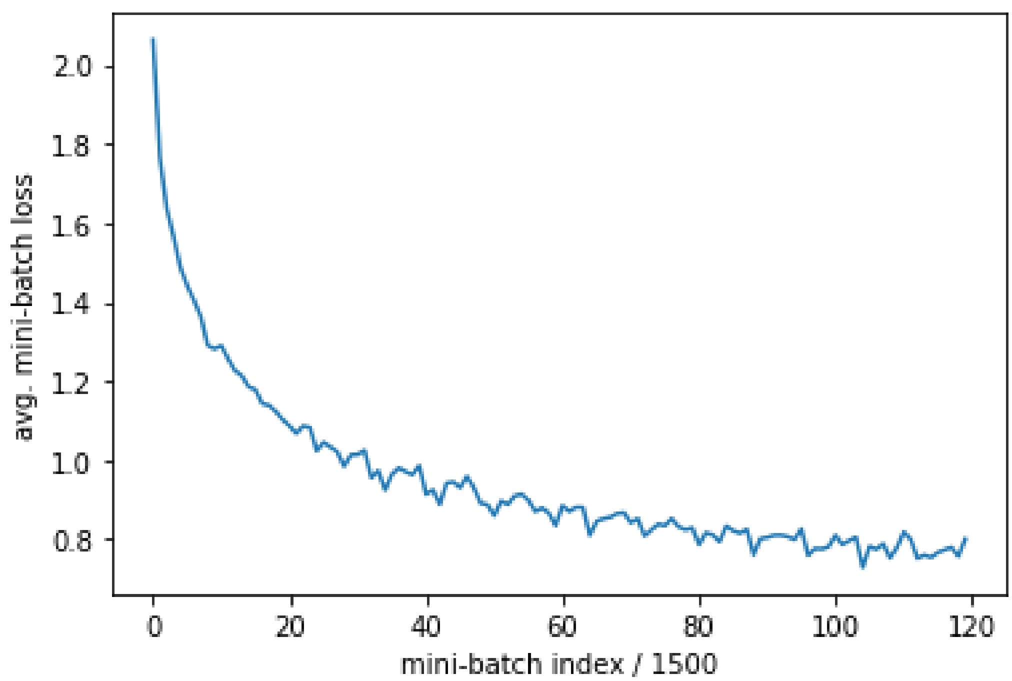
```
[epoch: 1, i:  4499] avg mini-batch loss: 1.290
[epoch: 1, i:  5999] avg mini-batch loss: 1.256
[epoch: 1, i:  7499] avg mini-batch loss: 1.227
[epoch: 1, i:  8999] avg mini-batch loss: 1.214
[epoch: 1, i: 10499] avg mini-batch loss: 1.186
[epoch: 1, i: 11999] avg mini-batch loss: 1.178
[epoch: 2, i:  1499] avg mini-batch loss: 1.143
[epoch: 2, i:  2999] avg mini-batch loss: 1.139
[epoch: 2, i:  4499] avg mini-batch loss: 1.123
[epoch: 2, i:  5999] avg mini-batch loss: 1.103
[epoch: 2, i:  7499] avg mini-batch loss: 1.087
[epoch: 2, i:  8999] avg mini-batch loss: 1.067
[epoch: 2, i: 10499] avg mini-batch loss: 1.087
[epoch: 2, i: 11999] avg mini-batch loss: 1.082
[epoch: 3, i:  1499] avg mini-batch loss: 1.024
[epoch: 3, i:  2999] avg mini-batch loss: 1.045
[epoch: 3, i:  4499] avg mini-batch loss: 1.033
[epoch: 3, i:  5999] avg mini-batch loss: 1.021
[epoch: 3, i:  7499] avg mini-batch loss: 0.984
[epoch: 3, i:  8999] avg mini-batch loss: 1.013
[epoch: 3, i: 10499] avg mini-batch loss: 1.014
[epoch: 3, i: 11999] avg mini-batch loss: 1.025
[epoch: 4, i:  1499] avg mini-batch loss: 0.956
[epoch: 4, i:  2999] avg mini-batch loss: 0.973
[epoch: 4, i:  4499] avg mini-batch loss: 0.925
[epoch: 4, i:  5999] avg mini-batch loss: 0.964
[epoch: 4, i:  7499] avg mini-batch loss: 0.980
[epoch: 4, i:  8999] avg mini-batch loss: 0.971
[epoch: 4, i: 10499] avg mini-batch loss: 0.963
[epoch: 4, i: 11999] avg mini-batch loss: 0.986
[epoch: 5, i:  1499] avg mini-batch loss: 0.914
[epoch: 5, i:  2999] avg mini-batch loss: 0.925
[epoch: 5, i:  4499] avg mini-batch loss: 0.888
[epoch: 5, i:  5999] avg mini-batch loss: 0.942
[epoch: 5, i:  7499] avg mini-batch loss: 0.945
[epoch: 5, i:  8999] avg mini-batch loss: 0.931
[epoch: 5, i: 10499] avg mini-batch loss: 0.959
[epoch: 5, i: 11999] avg mini-batch loss: 0.930
[epoch: 6, i:  1499] avg mini-batch loss: 0.891
[epoch: 6, i:  2999] avg mini-batch loss: 0.887
[epoch: 6, i:  4499] avg mini-batch loss: 0.859
[epoch: 6, i:  5999] avg mini-batch loss: 0.897
[epoch: 6, i:  7499] avg mini-batch loss: 0.888
[epoch: 6, i:  8999] avg mini-batch loss: 0.910
[epoch: 6, i: 10499] avg mini-batch loss: 0.914
[epoch: 6, i: 11999] avg mini-batch loss: 0.899
[epoch: 7, i:  1499] avg mini-batch loss: 0.871
[epoch: 7, i:  2999] avg mini-batch loss: 0.879
```

```
[epoch: 7, i:  4499] avg mini-batch loss: 0.866
[epoch: 7, i:  5999] avg mini-batch loss: 0.835
[epoch: 7, i:  7499] avg mini-batch loss: 0.885
[epoch: 7, i:  8999] avg mini-batch loss: 0.871
[epoch: 7, i: 10499] avg mini-batch loss: 0.881
[epoch: 7, i: 11999] avg mini-batch loss: 0.880
[epoch: 8, i:  1499] avg mini-batch loss: 0.810
[epoch: 8, i:  2999] avg mini-batch loss: 0.845
[epoch: 8, i:  4499] avg mini-batch loss: 0.852
[epoch: 8, i:  5999] avg mini-batch loss: 0.856
[epoch: 8, i:  7499] avg mini-batch loss: 0.865
[epoch: 8, i:  8999] avg mini-batch loss: 0.867
[epoch: 8, i: 10499] avg mini-batch loss: 0.842
[epoch: 8, i: 11999] avg mini-batch loss: 0.852
[epoch: 9, i:  1499] avg mini-batch loss: 0.809
[epoch: 9, i:  2999] avg mini-batch loss: 0.824
[epoch: 9, i:  4499] avg mini-batch loss: 0.839
[epoch: 9, i:  5999] avg mini-batch loss: 0.835
[epoch: 9, i:  7499] avg mini-batch loss: 0.852
[epoch: 9, i:  8999] avg mini-batch loss: 0.832
[epoch: 9, i: 10499] avg mini-batch loss: 0.824
[epoch: 9, i: 11999] avg mini-batch loss: 0.829
[epoch: 10, i:  1499] avg mini-batch loss: 0.786
[epoch: 10, i:  2999] avg mini-batch loss: 0.817
[epoch: 10, i:  4499] avg mini-batch loss: 0.811
[epoch: 10, i:  5999] avg mini-batch loss: 0.794
[epoch: 10, i:  7499] avg mini-batch loss: 0.834
[epoch: 10, i:  8999] avg mini-batch loss: 0.822
[epoch: 10, i: 10499] avg mini-batch loss: 0.815
[epoch: 10, i: 11999] avg mini-batch loss: 0.826
[epoch: 11, i:  1499] avg mini-batch loss: 0.761
[epoch: 11, i:  2999] avg mini-batch loss: 0.802
[epoch: 11, i:  4499] avg mini-batch loss: 0.805
[epoch: 11, i:  5999] avg mini-batch loss: 0.810
[epoch: 11, i:  7499] avg mini-batch loss: 0.811
[epoch: 11, i:  8999] avg mini-batch loss: 0.807
[epoch: 11, i: 10499] avg mini-batch loss: 0.799
[epoch: 11, i: 11999] avg mini-batch loss: 0.826
[epoch: 12, i:  1499] avg mini-batch loss: 0.759
[epoch: 12, i:  2999] avg mini-batch loss: 0.777
[epoch: 12, i:  4499] avg mini-batch loss: 0.776
[epoch: 12, i:  5999] avg mini-batch loss: 0.781
[epoch: 12, i:  7499] avg mini-batch loss: 0.811
[epoch: 12, i:  8999] avg mini-batch loss: 0.787
[epoch: 12, i: 10499] avg mini-batch loss: 0.797
[epoch: 12, i: 11999] avg mini-batch loss: 0.806
[epoch: 13, i:  1499] avg mini-batch loss: 0.729
[epoch: 13, i:  2999] avg mini-batch loss: 0.782
```

```
[epoch: 13, i:  4499] avg mini-batch loss: 0.775
[epoch: 13, i:  5999] avg mini-batch loss: 0.789
[epoch: 13, i:  7499] avg mini-batch loss: 0.753
[epoch: 13, i:  8999] avg mini-batch loss: 0.779
[epoch: 13, i: 10499] avg mini-batch loss: 0.819
[epoch: 13, i: 11999] avg mini-batch loss: 0.800
[epoch: 14, i:  1499] avg mini-batch loss: 0.752
[epoch: 14, i:  2999] avg mini-batch loss: 0.760
[epoch: 14, i:  4499] avg mini-batch loss: 0.754
[epoch: 14, i:  5999] avg mini-batch loss: 0.767
[epoch: 14, i:  7499] avg mini-batch loss: 0.774
[epoch: 14, i:  8999] avg mini-batch loss: 0.780
[epoch: 14, i: 10499] avg mini-batch loss: 0.757
[epoch: 14, i: 11999] avg mini-batch loss: 0.801
Finished Training.
```

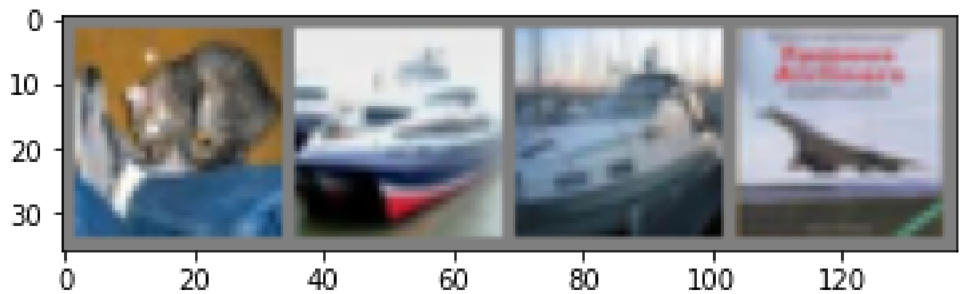
Training Loss Curve

[8]:
```python
plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```



Evaluate on Test Dataset

[9]:
```python
# 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:   ship   car plane plane
```

[10]:
```python
# 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: 69 %
```

[11]:
```python
# 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():
```

```python
    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 : 77 %
Accuracy of   car : 77 %
Accuracy of  bird : 43 %
Accuracy of   cat : 48 %
Accuracy of  deer : 68 %
Accuracy of   dog : 62 %
Accuracy of  frog : 81 %
Accuracy of horse : 77 %
Accuracy of  ship : 74 %
Accuracy of truck : 84 %
```