final_12

March 26, 2023

[1]:
```python
#!pip install torch torchvision
%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

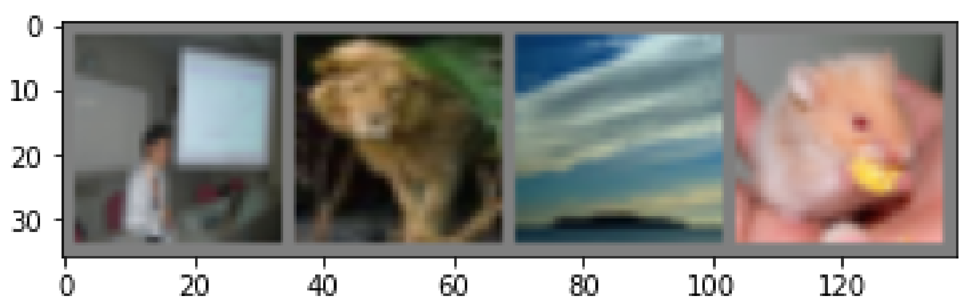
[2]:
```python
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR100(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.CIFAR100(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('apple', 'aquarium_fish', 'baby', 'bear', 'beaver',
           'bed', 'bee', 'beetle', 'bicycle', 'bottle', 'bowl',
           'boy', 'bridge', 'bus', 'butterfly', 'camel', 'can',
           'castle', 'caterpillar', 'cattle', 'chair',
           'chimpanzee', 'clock', 'cloud', 'cockroach', 'couch',
           'crab', 'crocodile', 'cup', 'dinosaur', 'dolphin',
           'elephant', 'flatfish', 'forest', 'fox', 'girl',
           'hamster', 'house', 'kangaroo', 'keyboard', 'lamp',
           'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster',
           'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse',
           'mushroom', 'oak_tree', 'orange', 'orchid', 'otter',
```

```
           'palm_tree', 'pear', 'pickup_truck', 'pine_tree', 'plain',
           'plate', 'poppy', 'porcupine', 'possum', 'rabbit', 'raccoon',
           'ray', 'road', 'rocket', 'rose', 'sea', 'seal', 'shark',
           'shrew', 'skunk', 'skyscraper', 'snail', 'snake', 'spider',
           'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table',
           'tank', 'telephone', 'television', 'tiger', 'tractor', 'train',
           'trout', 'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree',
           'wolf', 'woman', 'worm')

#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)))
```




```
television  lion cloud hamster
```

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")
```

```python
print(device)
# If 'cuda:0' is printed, it means GPU is available.
```

```
cuda:0
```

Network Definition

[5]:
```python
from torchvision.models import resnet

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.resnet = resnet.resnet18(pretrained=False)
        self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 100)

    def forward(self, x):
        x = self.resnet(x)
        return x

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

```
/home/jhandral/.local/lib/python3.9/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/jhandral/.local/lib/python3.9/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
```

[5]:
```
Net(
  (resnet): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
```

```
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
```

```
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=100, bias=True)
  )
)
```

Optimizer and Loss Function

[6]:
```
# 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=0.0001, momentum=0.9)
opt = optim.Adam(net.parameters(), lr=0.0002)
```

Training Procedure

[ ]:
```
import sys
from tqdm.notebook import tqdm

avg_losses = []   # Avg. losses.
epochs = 5        # Total epochs.
print_freq = 500  # 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(tqdm(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)
```

```python
        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),flush=True)
            sys.stdout.flush()
            avg_losses.append(avg_loss)
            running_loss = 0.0

print('Finished Training.')
```

```
  0%|          | 0/12500 [00:00<?, ?it/s]

[epoch: 0, i:   499] avg mini-batch loss: 4.770
[epoch: 0, i:   999] avg mini-batch loss: 4.563
[epoch: 0, i:  1499] avg mini-batch loss: 4.494
[epoch: 0, i:  1999] avg mini-batch loss: 4.429
[epoch: 0, i:  2499] avg mini-batch loss: 4.359
[epoch: 0, i:  2999] avg mini-batch loss: 4.329
[epoch: 0, i:  3499] avg mini-batch loss: 4.336
[epoch: 0, i:  3999] avg mini-batch loss: 4.226
[epoch: 0, i:  4499] avg mini-batch loss: 4.202
[epoch: 0, i:  4999] avg mini-batch loss: 4.195
[epoch: 0, i:  5499] avg mini-batch loss: 4.189
[epoch: 0, i:  5999] avg mini-batch loss: 4.125
[epoch: 0, i:  6499] avg mini-batch loss: 4.099
[epoch: 0, i:  6999] avg mini-batch loss: 4.099
[epoch: 0, i:  7499] avg mini-batch loss: 4.083
[epoch: 0, i:  7999] avg mini-batch loss: 4.067
[epoch: 0, i:  8499] avg mini-batch loss: 4.007
[epoch: 0, i:  8999] avg mini-batch loss: 4.047
[epoch: 0, i:  9499] avg mini-batch loss: 3.934
[epoch: 0, i:  9999] avg mini-batch loss: 3.924
[epoch: 0, i: 10499] avg mini-batch loss: 3.911
[epoch: 0, i: 10999] avg mini-batch loss: 3.883
[epoch: 0, i: 11499] avg mini-batch loss: 3.888
[epoch: 0, i: 11999] avg mini-batch loss: 3.915
[epoch: 0, i: 12499] avg mini-batch loss: 3.801
```

```
  0%|          | 0/12500 [00:00<?, ?it/s]

[epoch: 1, i:   499] avg mini-batch loss: 3.774
[epoch: 1, i:   999] avg mini-batch loss: 3.777
[epoch: 1, i:  1499] avg mini-batch loss: 3.708
[epoch: 1, i:  1999] avg mini-batch loss: 3.760
[epoch: 1, i:  2499] avg mini-batch loss: 3.710
[epoch: 1, i:  2999] avg mini-batch loss: 3.759
[epoch: 1, i:  3499] avg mini-batch loss: 3.746
[epoch: 1, i:  3999] avg mini-batch loss: 3.717
[epoch: 1, i:  4499] avg mini-batch loss: 3.699
[epoch: 1, i:  4999] avg mini-batch loss: 3.599
[epoch: 1, i:  5499] avg mini-batch loss: 3.656
[epoch: 1, i:  5999] avg mini-batch loss: 3.596
[epoch: 1, i:  6499] avg mini-batch loss: 3.613
[epoch: 1, i:  6999] avg mini-batch loss: 3.517
[epoch: 1, i:  7499] avg mini-batch loss: 3.584
[epoch: 1, i:  7999] avg mini-batch loss: 3.591
[epoch: 1, i:  8499] avg mini-batch loss: 3.574
[epoch: 1, i:  8999] avg mini-batch loss: 3.533
[epoch: 1, i:  9499] avg mini-batch loss: 3.466
[epoch: 1, i:  9999] avg mini-batch loss: 3.555
[epoch: 1, i: 10499] avg mini-batch loss: 3.497
[epoch: 1, i: 10999] avg mini-batch loss: 3.436
[epoch: 1, i: 11499] avg mini-batch loss: 3.442
[epoch: 1, i: 11999] avg mini-batch loss: 3.449
[epoch: 1, i: 12499] avg mini-batch loss: 3.497

  0%|          | 0/12500 [00:00<?, ?it/s]

[epoch: 2, i:   499] avg mini-batch loss: 3.402
[epoch: 2, i:   999] avg mini-batch loss: 3.357
[epoch: 2, i:  1499] avg mini-batch loss: 3.339
[epoch: 2, i:  1999] avg mini-batch loss: 3.385
[epoch: 2, i:  2499] avg mini-batch loss: 3.318
[epoch: 2, i:  2999] avg mini-batch loss: 3.378
[epoch: 2, i:  3499] avg mini-batch loss: 3.330
[epoch: 2, i:  3999] avg mini-batch loss: 3.325
[epoch: 2, i:  4499] avg mini-batch loss: 3.264
[epoch: 2, i:  4999] avg mini-batch loss: 3.300
[epoch: 2, i:  5499] avg mini-batch loss: 3.234
[epoch: 2, i:  5999] avg mini-batch loss: 3.276
[epoch: 2, i:  6499] avg mini-batch loss: 3.216
[epoch: 2, i:  6999] avg mini-batch loss: 3.276
[epoch: 2, i:  7499] avg mini-batch loss: 3.186
[epoch: 2, i:  7999] avg mini-batch loss: 3.234
[epoch: 2, i:  8499] avg mini-batch loss: 3.193
[epoch: 2, i:  8999] avg mini-batch loss: 3.205
[epoch: 2, i:  9499] avg mini-batch loss: 3.166
```

```
[epoch: 2, i:  9999] avg mini-batch loss: 3.188
[epoch: 2, i: 10499] avg mini-batch loss: 3.146
[epoch: 2, i: 10999] avg mini-batch loss: 3.176
[epoch: 2, i: 11499] avg mini-batch loss: 3.140
[epoch: 2, i: 11999] avg mini-batch loss: 3.158
[epoch: 2, i: 12499] avg mini-batch loss: 3.191
```

```
  0%|          | 0/12500 [00:00<?, ?it/s]
```

```
[epoch: 3, i:   499] avg mini-batch loss: 3.018
[epoch: 3, i:   999] avg mini-batch loss: 3.057
[epoch: 3, i:  1499] avg mini-batch loss: 2.963
[epoch: 3, i:  1999] avg mini-batch loss: 3.052
[epoch: 3, i:  2499] avg mini-batch loss: 3.085
[epoch: 3, i:  2999] avg mini-batch loss: 3.059
[epoch: 3, i:  3499] avg mini-batch loss: 2.949
[epoch: 3, i:  3999] avg mini-batch loss: 3.073
[epoch: 3, i:  4499] avg mini-batch loss: 3.001
[epoch: 3, i:  4999] avg mini-batch loss: 3.075
[epoch: 3, i:  5499] avg mini-batch loss: 2.956
```

Training Loss Curve

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

Evaluate on Test Dataset

[ ]:
```
# 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)))
```

[ ]:
```
# 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))
```

[ ]:
```
# Get test accuracy for each class.
class_correct = [0] * len(classes)
class_total = [0] * len(classes)
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(len(labels)):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

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

[ ]:
```
# One of the changes I made was that I added another layer in the network
# that takes the output from the second convolutional layer, applies ReLU␣
 ↪activation,
# and passes it through a 2x2 AvgPool layer to capture more relationships.
# I also reduced the learning rate of the optimizer to half (0.0005) which
# which allowed the optimizer to take smaller steps towards the minimum of
# the loss function which might allow for a better chance of finding the global␣
 ↪min of loss.
```