链滴

pytorch 入门笔记 -04- 训练图像分类器

作者：zyk

原文链接：https://ld246.com/article/1639630379699

来源网站：链滴

训练一个分类器

关于数据

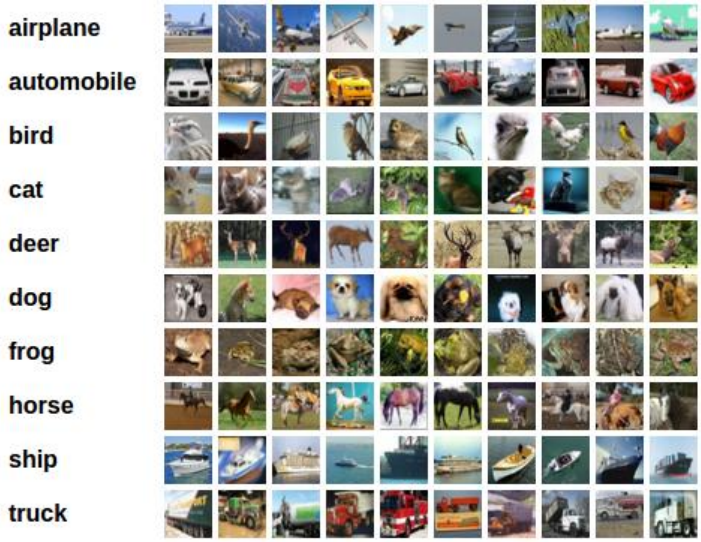
一般情况下处理图像、文本、音频和视频数据时，可以使用标准的 Python 包来加载数据到一个 Numpy 数组中，然后把这个数组转换成 torch.*Tensor。

- 图像可以使用 Pillow, OpenCV
- 音频可以使用 scipy, librosa
- 文本可以使用原始 Python 和 Cpython 来加载，或者使用 NLTK 或 SpaCy 处理

对于图像任务，可以调用 torchvision 包，它包含了处理一些基本图像数据集的方法。这些数据集包括 Imagenet, CIFAR10, MNIST 等。除了数据加载之外，torchvision 还包含了图像转换器，torchvison datasets 和 torch.utils.data.DataLoader。

torchvision 包不仅提供了巨大的便利，也避免了代码的重复。

在这个教程中，我们使用 CIFAR10 数据集，它有如下 10 个类别：'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'。CIFAR-10 的图像都是 3x32x32 大小的，即 3 个颜通道，32x32像素。



训练一个图像分类器

依次按照下列顺序进行：

1. 使用 torchvision 加载和归一化 CIFAR10 训练集和测试集
2. 定义一个卷积神经网络
3. 定义损失函数
4. 在训练集上训练网络
5. 在测试集上测试网络

读取和归一化 CIFAR10

使用 torchvision 可以非常容易地加载CIFAR10。

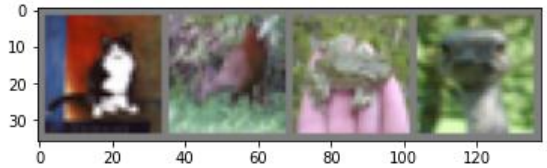
```
import torch
import torchvision
import torchvision.transforms as transforms
```

torchvision的输出是 [0,1] 的 PILImage 图像，我们把它转换为归一化范围为 [-1,1] 的张量。

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

# 加载训练数据集
trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=4, shuffle=True)

# 加载测试数据集
testset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=4, shuffle=False)
```

```
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

import matplotlib.pyplot as plt
import numpy as np


def imshow(img):
    """
    展示图像
    """
    img = img / 2 + 0.5  # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))


# 获取随机数据
dataiter = iter(trainloader)
images, labels = dataiter.next()

# 展示图像
imshow(torchvision.utils.make_grid(images))
# 显示图像标签
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

```
cat  deer  frog  bird
```



定义一个卷积神经网络

从之前的神经网络一节复制神经网络代码，并修改为输入 3 通道图像。

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, 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, 84)
        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 = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

定义损失函数和优化器

我们使用交叉熵作为损失函数，使用带动量的随即梯度下降

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

训练网络

迭代训练，将数据输入给网络，进行正向传播，反向传播和参数优化。

```
for epoch in range(5):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # 获取输入
        inputs, labels = data

        # 梯度置 0
        optimizer.zero_grad()

        # 正向传播，反向传播，参数优化
        outpus = net(inputs)
        loss = criterion(outpus, labels)
        loss.backward()
        optimizer.step()

        # 打印状态信息
        running_loss += loss.item()
        if i % 2000 == 1999:  # 每 2000 批次打印一次
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.167
[1,  4000] loss: 1.843
[1,  6000] loss: 1.666
[1,  8000] loss: 1.578
[1, 10000] loss: 1.510
[1, 12000] loss: 1.460
```

```
[2,  2000] loss: 1.406
[2,  4000] loss: 1.343
[2,  6000] loss: 1.356
[2,  8000] loss: 1.328
[2, 10000] loss: 1.290
[2, 12000] loss: 1.279
[3,  2000] loss: 1.213
[3,  4000] loss: 1.187
[3,  6000] loss: 1.200
[3,  8000] loss: 1.195
[3, 10000] loss: 1.181
[3, 12000] loss: 1.176
[4,  2000] loss: 1.116
[4,  4000] loss: 1.090
[4,  6000] loss: 1.111
[4,  8000] loss: 1.092
[4, 10000] loss: 1.101
[4, 12000] loss: 1.097
[5,  2000] loss: 1.000
[5,  4000] loss: 1.010
[5,  6000] loss: 1.037
[5,  8000] loss: 1.045
[5, 10000] loss: 1.037
[5, 12000] loss: 1.039
Finished Training
```

```
dataiter = iter(testloader)
images, labels = dataiter.next()

# 显示图片
imshow(torchvision.utils.make_grid(images))
print('GroudTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

```
GroudTruth:    cat  ship  ship plane
```

让我们看看神经网络认为以上图片是什么。

```
outputs = net(images)
```

输出是 10 个标签的能量。一个类别的能量越大，神经网络越认为它是这个类别。所以我们需要将高量的标签作为最后的结果。

```
_, predicted = torch.max(outputs, 1)
print('Predicted: ', ' '.join('%5s' % classes[predicted[j]] for j in range(4)))
```

```
Predicted:   ship  ship  ship  ship
```

接下来看看网络在整个测试集上的结果如何。

```
correct = 0
total = 0

with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 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: 60 %
```

统计每一个分类下的准确率。

```
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
        outpus = net(images)
        _, predicted = torch.max(outpus, 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 : 65 %
Accuracy of   car : 69 %
Accuracy of  bird : 53 %
Accuracy of   cat : 33 %
Accuracy of  deer : 46 %
Accuracy of   dog : 56 %
Accuracy of  frog : 65 %
Accuracy of horse : 65 %
Accuracy of  ship : 77 %
Accuracy of truck : 77 %
```

下一步?

我们如何在 GPU 上运行神经网络呢?

在 GPU 上训练

把一个神经网络移动到 GPU 上训练就像把一个 Tensor 转换 GPU 上一样简单。并且这个操作会递归

历有所模块，并将其参数和缓冲区转换为 CUDA 张量。

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# 确认我们电脑支持 CUDA，显示 CUDA 信息
print(device)
```

```
cuda:0
```

本节的其余部分假定 device 是 CUDA 设备。

然后这些方法将递归遍历所有模块并将模块的参数和缓冲区转换成 CUDA 张量：

```
net.to(device)
```

记住：inputs, targets 和 images 也要转换。

```
inputs, labels = inputs.to(device), labels.to(device)
```

为什么我们没注意到 GPU 的速度提升很多？那是因为网络非常的小。

实践:

尝试增加你的网络的宽度（第一个 nn.Conv2d 的第 2 个参数，第二个 nn.Conv2d 的第一个参数，们需要是相同的数字），看看你得到了什么样的加速。

实现的目标:

- 深入了解了 PyTorch 的张量库和神经网络
- 训练了一个小网络来分类图片