训练分类器

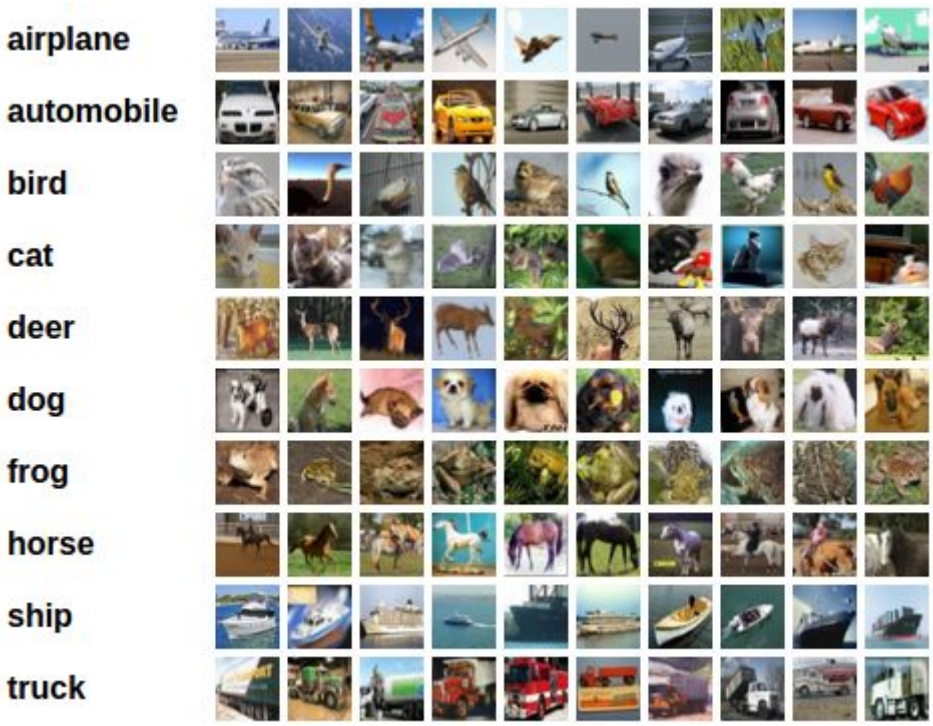
1.训练数据

训练分类器，首先需要考虑数据的问题。通常处理图片、文本、语音、视频等数据时，一般采用标准python库将其转换为Numpy数组，然后转回为PyTorch张量。

- 对于图像，可以采用`pillow`,`OpenCV`库
- 对于语音， 有`scipy`和`librosa`
- 对于文本，可以选择原生python或Cython进行加载数据，或者使用`NLTK`和`spaCy`

PyTorch对于计算机实觉，特别创建了一个`torchvision`库，包含一个**数据加载器**（data loader），可以加载比较常见的数据集，比如`Imagenet`、`CIFAR10`、`MNIST`等，还有一个用于图像的**数据转换器**（data transformers）调用的库是`torchvision.datasets`和`torch.utils.data.DataLoader`.

对于`CIFAR10`数据集，包含10个类别，分别是飞机、汽车、鸟、猫、鹿、狗、青蛙、马、船和卡车。数据集图片尺寸是`3*32*32`。例如：



2.训练图片分类器

流程如下：

1. 通过调用`torchvision`加载和归一化`CIFAR10`训练集和测试集。

2. 构建一个卷积神经网络
3. 定义一个损失函数
4. 在训练集上训练网络
5. 在测试集上测试网络性能。

2.1加载和归一化CIFAR10

导入必须的包：

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

`torchvision`的数据集输出的图片都是`PILImage`，取值范围是`[0, 1]`，需要归一化为`[-1, 1]`：

注：同时若本地没有数据集，`torchvision`会自动进行下载。

```
# 将数据归一化为[-1, 1]
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')

```

注：此处存在PIL版本问题：详情点这里

可视化部分训练图片：

```
import matplotlib.pyplot as plt
import numpy as np

# 展示图片的函数
def imshow(img):

```

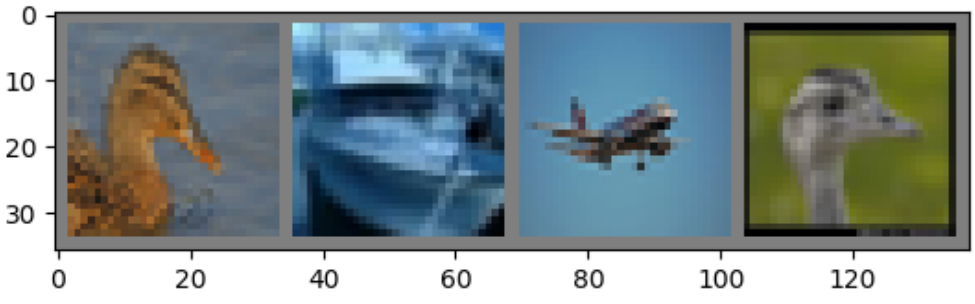
```
    img = img / 2 + 0.5     # 非归一化
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


if __name__ == '__main__':
    # 注：若未将以下内容写在主函数接口下，直接运行，mac可以成功运行，但是win10由于无法启动          多线程环境而无法成功运行，报错：BrokenPipeError: [Errno 32] Broken pipe
    # 随机获取训练集图片
    dataiter = iter(trainloader)
    images, labels = dataiter.next()

    # 展示图片
    imshow(torchvision.utils.make_grid(images))
    # 打印图片类别标签
    print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

注：此处存在PyTorch BUG之一，详情点这里

可视化数据：




```
bird  ship plane  bird
```

2.2构建卷积神经网络

网络接受三通道的彩色图片

```
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
```

2.3定义损失函数和优化器

此处采用类别交叉熵函数和带有动量的SGD优化方法：

```
import torch.optim as optim

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

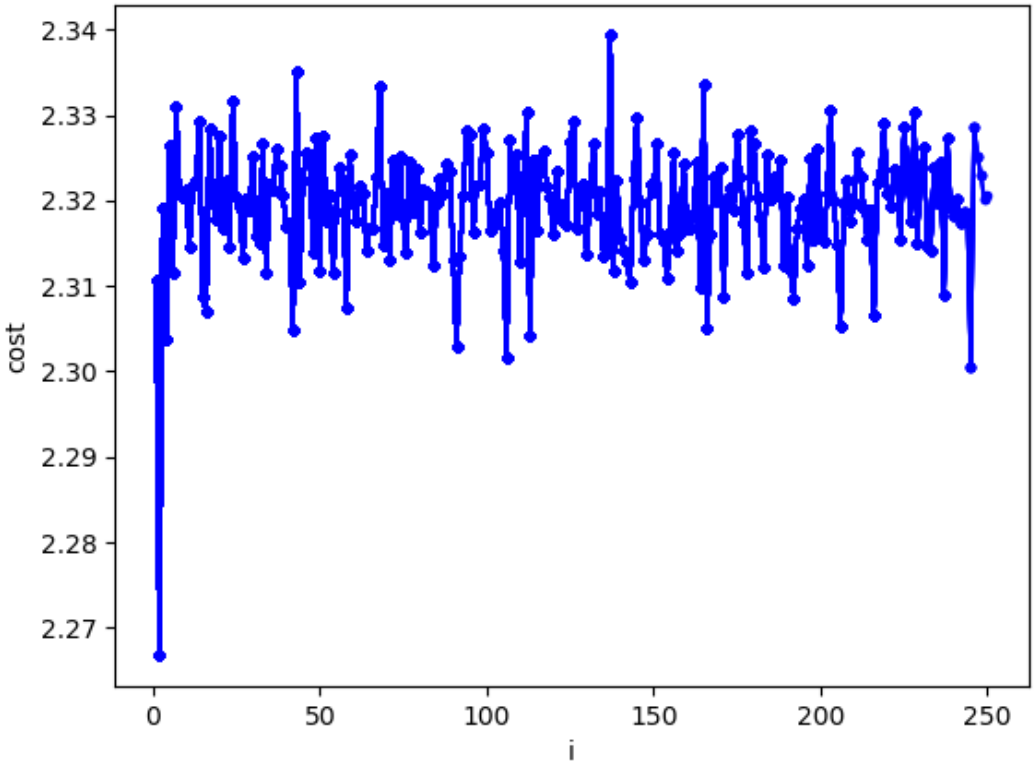
2.4训练网络及训练过程可视化：

```
import time
import matplotlib.pyplot as plt
def draw_pic(j, data):
    plt.ion()
    plt.scatter(j, data, color='blue', marker='.')
    plt.xlabel('i')
    plt.ylabel('cost')
    plt.show()
    plt.pause(0.00001)

```

```
11
12  def train():
13      trainloader, testloader, classes = load_data()
14      net = Net()
15      criterion = nn.CrossEntropyLoss()
16      optimizer = optim.SGD(net.parameters(), lr=0.03,
    momentum=0.9)
17      start = time.time()
18      print('Start Training!')
19      loss_list = []
20      j = 1
21
22      for epoch in range(1):
23          running_loss = 0.0
24          for i, data in enumerate(trainloader, 0):
25              inputs, labels = data  # 获取输入数据
26              optimizer.zero_grad()  # 清空梯度缓存
27              outputs = net(inputs)  # 正向传播
28              loss = criterion(outputs, labels)  # 计算损失
29              loss.backward()  # 反向传播
30              optimizer.step()  # 参数更新
31              running_loss += loss.item()
32              if i % 100 == 99:  # 每迭代2000次打印一次信息
33                  print('[%d, %5d] loss: %.3f' % (epoch + 1, i
    + 1, running_loss / 100))
34                  draw_pic(j, running_loss / 100)  # 损失可视化
35                  j += 1
36                  running_loss = 0.0
37      print('Finished Training! Total cost time: ', time.time()
    - start, 's')
```

测试结果：



在整个测试集上的准确率：

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        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))
```

计算每个分类的准确率：

此处计算准确部分`c = (predicted == labels).squeeze()`会根据预测和真实标签是否相等，输出1或0，表示真或假。

```
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
```

```
        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]))
```

输出：

```
Finished Training! Total cost time:  577.5428650379181 s
Accuracy of the network on the 10000 test images: 54 %
Accuracy of plane : 64 %
Accuracy of   car : 70 %
Accuracy of  bird : 31 %
Accuracy of   cat : 40 %
Accuracy of  deer : 43 %
Accuracy of   dog : 49 %
Accuracy of  frog : 49 %
Accuracy of horse : 68 %
Accuracy of  ship : 62 %
Accuracy of truck : 64 %
```

3. 在GPU上训练

首先检测是否有可用的GPU来训练：

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

若输出为`cpu`说明无可用显卡设备，若输出为`cuda:0`表明你的第一块 GPU 显卡或者唯一的 GPU 显卡是空闲可用状态。

分别将网络参数和数据转移到GPU上：

```
net.to(device) # 网络参数转移到GPU上
inputs, labels = inputs.to(device), labels.to(device)  # 数据转
移到GPU上
```

更改过后的代码：

```
import time
# 在 GPU 上训练注意需要将网络和数据放到 GPU 上
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001,
momentum=0.9)

start = time.time()
for epoch in range(2):

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # 获取输入数据
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        # 清空梯度缓存
        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # 打印统计信息
        running_loss += loss.item()
        if i % 2000 == 1999:
            # 每 2000 次迭代打印一次信息
            print('[%d, %5d] loss: %.3f' % (epoch + 1, i+1,
running_loss / 2000))
            running_loss = 0.0
print('Finished Training! Total cost time: ', time.time() -
start)
```

调用`net.to(device)`后，再定义优化器，即传入的是CUDA张量的网络数据。

4.数据并行

利用`DataParallel`来使用多个GPU训练网络。

将网络参数放到指定GPU上：

```
device = torch.device("cuda:0")  # 此处指定GPU
model.to(device)
```

再将所有变量放到GPU上：

```
mytensor = my_tensor.to(device)
```

注：此处`my_tensor.to(device)`是返回一个`my_tensor`的新拷贝对象，而不是直接修改变量，因此需要将其赋值给一个新的张量，然后使用这个张量。

使用多个GPU需要采用`DaraParallel`，它会自动分割数据集，并发送任务给多个GPU上的多个模型，等待每个模型都完成各自任务后，会收集并融合结果，然后返回。

```
model = nn.DataParallel(model)
```

4.1导入和参数

主要定义网络输入大小和输出大小，`batch`以及图片大小，并定义了一个`device`对象

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

# Parameters and DataLoaders
input_size = 5
output_size = 2

batch_size = 30
data_size = 100

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

4.2构建一个假数据集

构建一个随机数据集

```
class RandomDataset(Dataset):

    def __init__(self, size, length):
        self.len = length
        self.data = torch.randn(length, size)  # 随机生成

    def __getitem__(self, index):
        return self.data[index]

```

```
    def __len__(self):
        return self.len

rand_loader = DataLoader(dataset=RandomDataset(input_size, data_size),
                         batch_size=batch_size, shuffle=True)
```

4.3 构建网络模型

可以构建一个简单的网络模型，加入`print()`用于监控网络输入和输出的tensor大小、

```
class Model(nn.Module):
    # Our model

    def __init__(self, input_size, output_size):
        super(Model, self).__init__()
        self.fc = nn.Linear(input_size, output_size)

    def forward(self, input):
        output = self.fc(input)
        print("\tIn Model: input size", input.size(),
              "output size", output.size())

        return output
```

4.4 创建模型和数据平行

首先定义一个模型实例，并检查是否拥有多个GPU， 如果是，就将模型包裹在`nn.DataParallel`，并调用`model.to(device)`

```
model = Model(input_size, output_size)  # 网络实例化
if torch.cuda.device_count() > 1:
  print("Let's use", torch.cuda.device_count(), "GPUs!")
  # dim = 0 [30, xxx] -> [10, ...], [10, ...], [10, ...] on 3 GPUs
  model = nn.DataParallel(model)   # 将模型传入nn.DataParallel

model.to(device)
```

4.5 运行模型

```
for data in rand_loader:
    input = data.to(device)
    output = model(input)
    print("Outside: input size", input.size(),
          "output_size", output.size())
```

注：Multi-GPUs官方文档