哈尔滨工业大学

<<模式识别与深度学习>>

实验 1 实验报告

(2020 春季学期)

成员 1:	1172100327 王金翼
成员 2:	
成员 3:	

-

实验要求

熟悉pytorch的基本操作：用pytorch实现MLP，并在MNIST数据集上进行训练

环境配置

实验环境如下：

- Win10
- python3.8
- Anaconda3
- Cuda10.2 + cudnn v7
- GPU : NVIDIA GeForce MX250

配置环境的过程中遇到了一些问题，解决方案如下：

1. anaconda下载过慢

   使用清华镜像源，直接百度搜索即可

2. pytorch安装失败

   这里我首先使用的是pip的安装方法，失败多次后尝试了使用anaconda，然后配置了清华镜像源，最后成功。参考的教程如下： win10快速安装pytorch，清华镜像源

   当然也可以直接去pytorch官网下载所需版本的whl文件，然后手动pip安装。由于这种方式我已经学会了，为了学习anaconda，所以没有采用这种方式。具体方式可以百度如何使用whl。顺便贴下pytorch的whl的下载页面

注意：pytorch的版本是要严格对应是否使用GPU、python版本、cuda版本的，如需手动下载pytorch的安装包，需搞懂其whl文件的命名格式

另外还学习了anaconda的一些基本操作与原理，参考如下： Anaconda完全入门指南

实验过程

最终代码见github：hit-deeplearning-1

首先设置一些全局变量，加载数据。batch_size决定了每次向网络中输入的样本数，epoch决定了整个数据集的迭代次数，具体作用与大小如何调整可参考附录中的博客。

将数据读入，如果数据不存在于本地，则可以自动从网上下载，并保存在本地的data文件夹下。

```
#一次取出的训练样本数
batch_size = 16
# epoch 的数目
n_epochs = 10

#读取数据
train_data = datasets.MNIST(root="./data", train=True,
download=True,transform=transforms.ToTensor())
```

```
test_data = datasets.MNIST(root="./data", train=False, download=True,
transform=transforms.ToTensor())
#创建数据加载器
train_loader = torch.utils.data.DataLoader(train_data, batch_size = batch_size,
num_workers = 0)
test_loader = torch.utils.data.DataLoader(test_data, batch_size = batch_size,
num_workers = 0)
```

接下来是创建MLP模型，关于如何创建一个模型，可以参考附录中的博客，总之创建模型模板，训练模板都是固定的。

其中Linear、view、CrossEntropyLoss、SGD的用法需重点关注。查看官方文档或博客解决。

这两条语句将数据放到了GPU上，同理测试的时候也要这样做。

```
data = data.cuda()
target = target.cuda()
```

```
class MLP(nn.Module):
    def __init__(self):
        #继承自父类
        super(MLP, self).__init__()
        #创建一个三层的网络
        #输入的28*28为图片大小，输出的10为数字的类别数
        hidden_first = 512
        hidden_second = 512
        self.first = nn.Linear(in_features=28*28, out_features=hidden_first)
        self.second = nn.Linear(in_features=hidden_first,
out_features=hidden_second)
        self.third = nn.Linear(in_features=hidden_second, out_features=10)

    def forward(self, data):
        #先将图片数据转化为1*784的张量
        data = data.view(-1, 28*28)
        data = F.relu(self.first(data))
        data = F.relu((self.second(data)))
        data = F.log_softmax(self.third(data), dim = 1)

        return data

def train():
    # 定义损失函数和优化器
    lossfunc = torch.nn.CrossEntropyLoss().cuda()
    #lossfunc = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(params=model.parameters(), lr=0.01)
    # 开始训练
    for epoch in range(n_epochs):
        train_loss = 0.0
        for data, target in train_loader:
```

```
            optimizer.zero_grad()
            #将数据放至GPU并计算输出
            data = data.cuda()
            target = target.cuda()
            output = model(data)
            #计算误差
            loss = lossfunc(output, target)
            #反向传播
            loss.backward()
            #将参数更新至网络中
            optimizer.step()
            #计算误差
            train_loss += loss.item() * data.size(0)
        train_loss = train_loss / len(train_loader.dataset)
        print('Epoch:  {}  \tTraining Loss: {:.6f}'.format(epoch + 1, train_loss))
        # 每遍历一遍数据集，测试一下准确率
        test()
    #最后将模型保存
    path = "model.pt"
    torch.save(model, path)
```

test程序不再贴出，直接调用了一个很常用的test程序。

最后是主程序，在这里将模型放到GPU上。

```
model = MLP()
#将模型放到GPU上
model = model.cuda()
train()
```

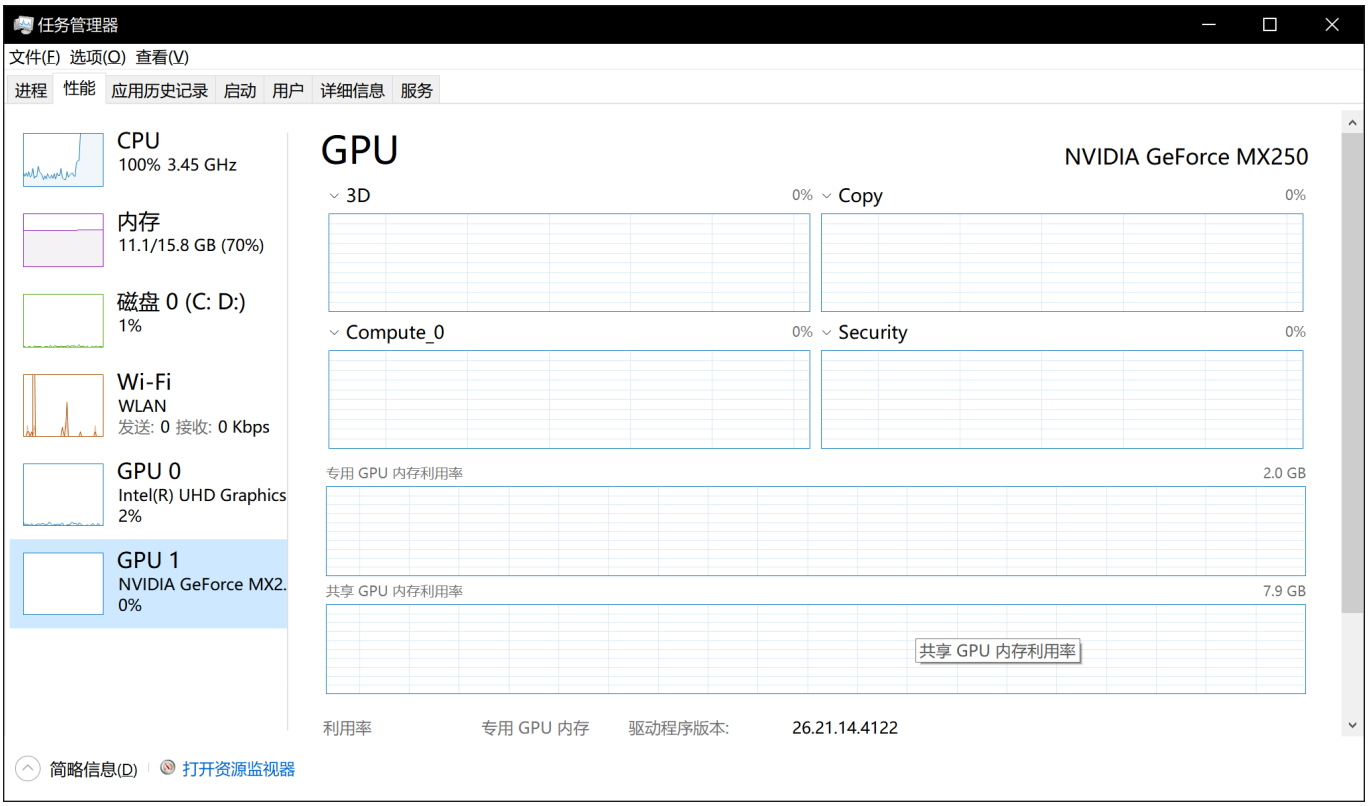
实验结果

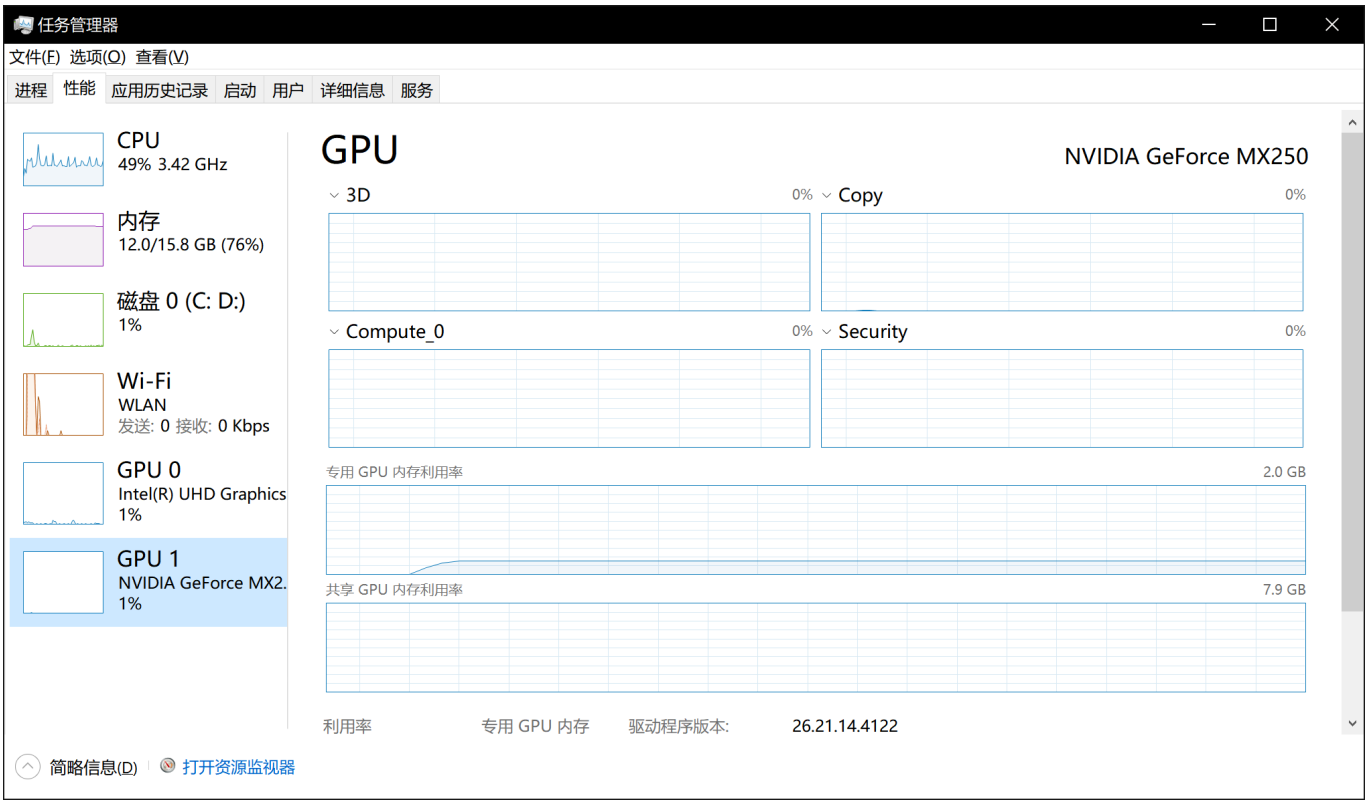
实验结果如下，可以看到，当对数据迭代训练十次时，准确率已经可以达到97%

```
Epoch:  1   Training Loss: 0.693887
Accuracy of the network on the test images: 90 %
Epoch:  2   Training Loss: 0.272068
Accuracy of the network on the test images: 93 %
Epoch:  3   Training Loss: 0.209291
Accuracy of the network on the test images: 94 %
Epoch:  4   Training Loss: 0.168086
Accuracy of the network on the test images: 95 %
Epoch:  5   Training Loss: 0.139021
Accuracy of the network on the test images: 95 %
Epoch:  6   Training Loss: 0.117500
Accuracy of the network on the test images: 96 %
Epoch:  7   Training Loss: 0.100973
Accuracy of the network on the test images: 96 %
Epoch:  8   Training Loss: 0.087816
Accuracy of the network on the test images: 97 %
Epoch:  9   Training Loss: 0.077013
Accuracy of the network on the test images: 97 %
Epoch:  10      Training Loss: 0.067966
Accuracy of the network on the test images: 97 %

Process finished with exit code 0
```

分别运行了两次，第一次没有使用cuda加速，第二次使用了cuda加速，任务管理器分别显示如下：





可以看到，未使用cuda加速时，cpu占用率达到了100%，而GPU的使用率为0；而使用cuda加速时，cpu占用率只有49%，而GPU使用率为1%。这里GPU使用率较低的原因很多，比如我程序中batch_size设置的较小，另外只将数据和模型放到了GPU上，cpu上仍有部分代码与数据。经简单测试，使用cuda的训练时间在2:30左右，不使用cuda的训练时间在3:40左右。

参考博客

使用Pytorch构建MLP模型实现MNIST手写数字识别

如何创建自定义模型

pytorch教程之nn.Module类详解——使用Module类来自定义网络层

epoch和batch是什么

深度学习 | 三个概念：Epoch, Batch, Iteration

如何用GPU加速

从头学pytorch(十三):使用GPU做计算

PyTorch如何使用GPU加速（CPU与GPU数据的相互转换）

保存模型

PyTorch模型保存与加载