Lab5：深度神经网络训练与加速

Zhewen Yu

第一部分 LeNet-5模型

1.模型搭建

1.1初始化网络

根据给出的参考图片，定义网络层。

```
def __init__(self):
        super(MyLeNet_5,self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1,out_channels=6,kernel_size=5,padding=2)
        self.Sigmoid = nn.Sigmoid()
        self.pool1 = nn.AvgPool2d(kernel_size=2,stride=2)
        self.conv2 = nn.Conv2d(in_channels=6,out_channels=16,kernel_size=5)
        self.pool2 = nn.AvgPool2d(kernel_size=2,stride=2)
        self.conv3 = nn.Conv2d(in_channels=16,out_channels=120,kernel_size=5)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(120,84)
        self.output = nn.Linear(84,10)
```

1.其中，由于输入是`32*32`，实际图片大小是`28*28`,定义padding为2。
2.这里使用`sigmoid`函数作为激活函数,`nn.Sigmoid()` 函数是光滑的，便于求导，同时它能够有效的压缩数据幅度。，`sigmoid`在该模型中效果好于`ReLU`作为激活函数。
3.在最后一次卷积后，图片成为`1*1`大小，这里考虑将其展开。
4.最后设置两个线性连接层，最后输出10个数字。

1.2前向传播网络

```
def forward(self, x):
    x = self.Sigmoid(self.conv1(x))
    x = self.pool1(x)
    x = self.Sigmoid(self.conv2(x))
    x = self.pool2(x)
    x = self.conv3(x)
    x = self.flatten(x)
    x = self.fc1(x)
    x = self.output(x)
    return x
```

1.这里选择激活前两次卷积的结果，以获得较好的结果。
2.根据图片逐步搭建池化层和连接层。

2.模型训练

2.1将数据转化为张量格式

```
#Data to Tensor
data_transform = transforms.Compose(
    [transforms.ToTensor()]
)
```

矩阵格式的数据在神经网络中要先张量化。

2.2加载`MNIST`数据集并设定测试集数据

```
#MNIST import
train_dataset=datasets.MNIST(root='./data', train=True ,transform=data_transform,download=True)
train_dataloader=Data.DataLoader(dataset=train_dataset,batch_size=16,shuffle=True)
#load test
test_dataset=datasets.MNIST(root='./data', train=False ,transform=data_transform,download=True)
test_dataloader=Data.DataLoader(dataset=test_dataset,batch_size=16,shuffle=False)
```

使用`DataLoader`载入已有的`MNIST`数据集，其中，`batch_size`定为16，设定为乱序，载入到当前目录的data文件夹中。

2.3 设置处理设备为GPU

```
#device : GPU or CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
#import GPU
model=MyLeNet_5().to(device)
```

在没有`GPU`时，也可以由`CPU`训练。

2.4 定义损失函数、优化器和学习率

```
#Loss Function
loss_fn=nn.CrossEntropyLoss()
#optimizer
optimizer=optim.SGD(model.parameters(),lr=1e-3,momentum=0.9)
# Learning Rate
Lr_scheduler=optim.lr_scheduler.StepLR(optimizer,step_size=10,gamma=0.1)
```

这里的损失函数使用交叉熵函数，优化器使用随机梯度下降法。
查询资料得知，`momentum`参数设定为0.9时训练结果最好。
对于学习率，为防止抖动太大，每间隔10轮变为原来的0.1。

2.5 定义训练函数。

```
#train
def train(dataloader,model,loss_fn,optimizer):
    loss,current,n = 0.0, 0.0 ,0
    for batch,(X,y) in enumerate(dataloader):
        #forward
        X,y=X.to(device),y.to(device)
        output = model(X)
        cur_loss = loss_fn(output,y)
        _, pred = torch.max(output,axis=1)
        cur_acc=torch.sum(y==pred)/output.shape[0]
        #backward
        optimizer.zero_grad()
        cur_loss.backward()
        optimizer.step()
        #计算准确率
        loss += cur_loss.item() #累加本批次损失
        current += cur_acc.item()
        n+=1
    print("train_loss:"+str(loss/n))
    print("train_acc:"+str(current/n))
```

通过交叉熵函数返回的损失值，反向传播增加准确率。

2.6 定义验证函数

```
#验证
def val(dataloader,model,loss_fn,i):
    model.eval()
    loss,current,n = 0.0, 0.0 ,0
    with torch.no_grad():
        for batch,(X,y) in enumerate(dataloader):
            X,y=X.to(device),y.to(device)
            output = model(X)
            cur_loss = loss_fn(output,y)
            _, pred = torch.max(output,axis=1)
            cur_acc=torch.sum(y==pred)/output.shape[0]
            loss += cur_loss.item() #累加本批次损失
            current += cur_acc.item()
            n+=1
    writer.add_scalar("Loss/train",cur_loss , i)
    writer.add_scalar("Acc/train", cur_acc, i)
    print("val_loss:"+str(loss/n))
    print("val_acc:"+str(current/n))
    return current/n
```

基本复制训练部分，值得注意的是验证时模型不参与更新。

2.7设定训练循环

```
#start train
epoch =50
min_acc= 0
for i in range(epoch):
    print(f'round{i+1}\n-------------')
    train(train_dataloader,model,loss_fn,optimizer)
    a = val(test_dataloader,model,loss_fn,i)
    #save best model
    if a >min_acc:
        folder = 'save_model'
        if not os.path.exists(folder):
            os.mkdir('save_model')
        min_acc=a
        print('save best model')
        torch.save(model.state_dict(),'save_model/best_model.pth')
print('Finished Training')
```

设定训练50轮次，保存模型数据到`best_model`文件夹。

2.8使用`TensorBoard`实现可视化

期间也根据`TensorBoard`官网教程，在训练程序头尾加上相应代码。

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
# ...中间部分省略
writer.flush()
writer.close()
```

关于`TensorBoard`结果，在4.4中。

3.模型测试

3.1 前期导入

基本与`train.py`一致

```
#Data to Tensor
data_transform = transforms.Compose(
    [transforms.ToTensor()]
)
# ...中间部分省略
model.load_state_dict(torch.load("save_model/best_model.pth"))
```

从保存模型参数的`best_model`文件夹中导入模型。

3.2将张量转化为图片并进行20轮测试

```
# tensor to pictures
show = ToPILImage()

#vertify
for i in range(20):
    X,y = test_dataset[i][0],test_dataset[i][1]
    show(X).show()
    X = Variable(torch.unsqueeze(X, dim =0).float(),requires_grad=False).to(device)
    with torch.no_grad():
        pred = model(X)
        predicted, actual = classes[torch.argmax(pred[0])],classes[y]
        print(f'predicted:"{predicted}",actual:"{actual}"')
```

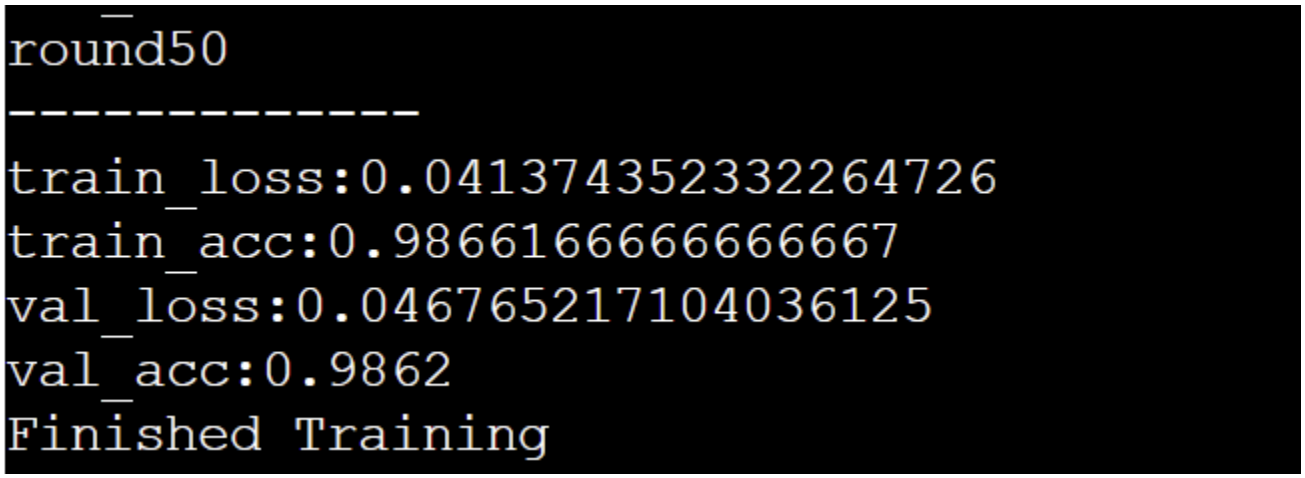
4.模型结果

4.1 GPU占用率截图

```
Wed Aug 10 03:15:46 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.102.04   Driver Version: 450.102.04   CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce RTX 208...  On   | 00000000:1D:00.0 Off |                  N/A |
| 28%   28C    P2    58W / 250W |   1156MiB / 11019MiB |     22%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0    0    0       4690      C   python3                         1153MiB |
+-----------------------------------------------------------------------------+
```

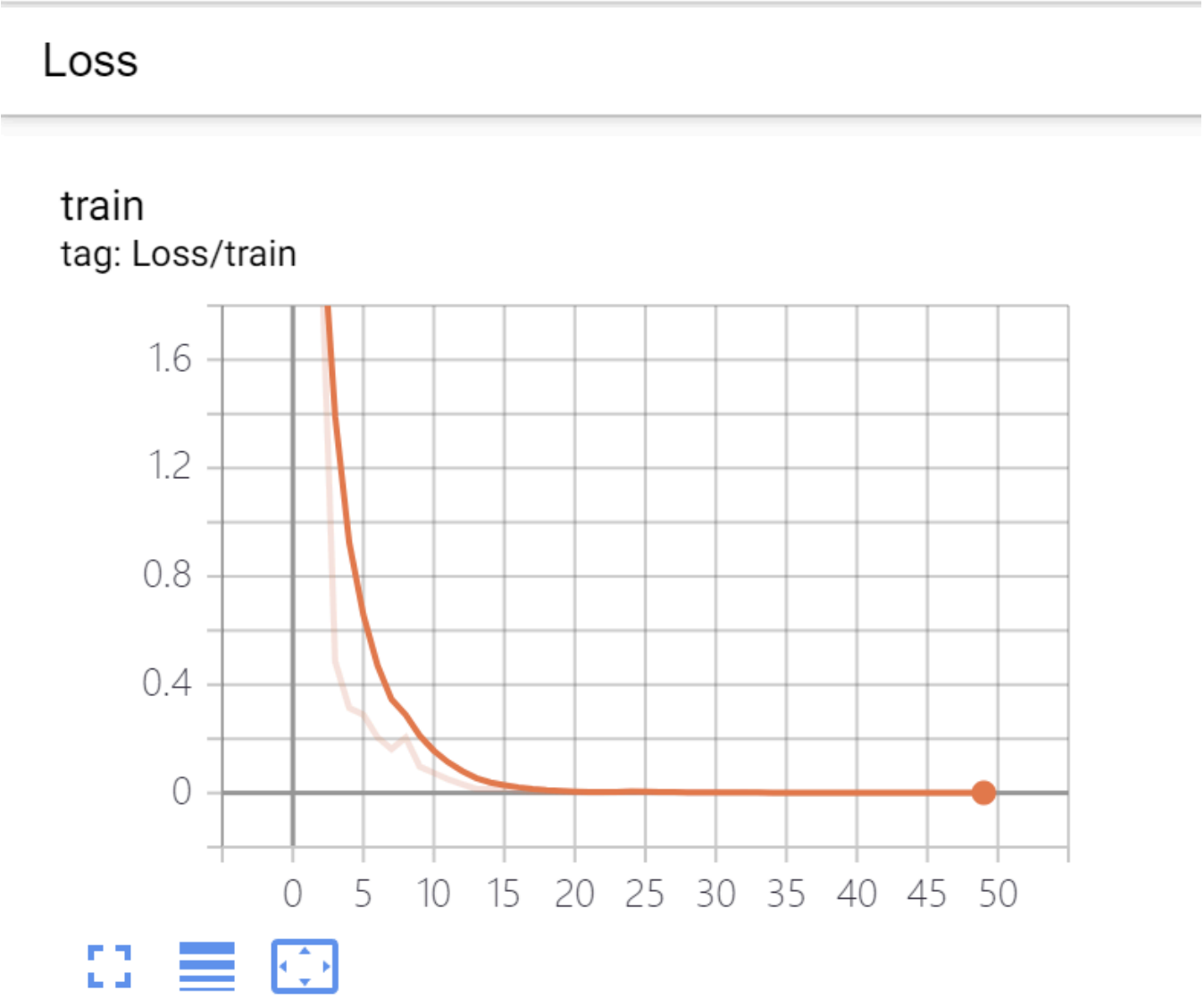
4.2 训练结束截图



4.3 训练测试截图

```
root@6vngjd1psvf2d-0:/chenjh14/mio/MyLeNet5# python3 test.py
predicted:"7",actual:"7"
predicted:"2",actual:"2"
predicted:"1",actual:"1"
predicted:"0",actual:"0"
predicted:"4",actual:"4"
predicted:"1",actual:"1"
predicted:"4",actual:"4"
predicted:"9",actual:"9"
predicted:"5",actual:"5"
predicted:"9",actual:"9"
predicted:"0",actual:"0"
predicted:"6",actual:"6"
predicted:"9",actual:"9"
predicted:"0",actual:"0"
predicted:"1",actual:"1"
predicted:"5",actual:"5"
predicted:"9",actual:"9"
predicted:"7",actual:"7"
predicted:"3",actual:"3"
predicted:"4",actual:"4"
```

4.4 TansorBoard截图



可以看到，占用率较为一般，但是实验结果和准确率极高，为1，损失极低，约为0。、

第二部分 GPT模型的训练与加速

第三部分 参考文献

特别鸣谢 哔哩哔哩大学、知乎大学

`Le-Net 5`模型部分

https://www.bilibili.com/video/BV11g411u7eL?spm_id_from=333.337.search-

card.all.click

通过这个视频理解了前向传播、反向传播

https://www.bilibili.com/video/BV1SF411K7fK?spm_id_from=333.337.search-card.all.click&vd_source=b7a86eab6c7d442eb274e6592fa32c1f

通过这个视频理解了`Le-Net5`模型原论文

https://www.bilibili.com/video/BV1vU4y1A7QJ?spm_id_from=333.337.search-card.all.click

通过这个视频逐步理解和搭建了自己的`Le-Net5`模型骨架

https://zhuanlan.zhihu.com/p/362052826

通过这篇文章逐步搭建自己的`Le-Net5`模型骨架

`GPT`模型部分