手写数字识别

1 实验概述

本实验为深度学习经典实验之一，目标是设计一个模型来识别手写数字。

2 实验环境

2.1 硬件环境

- CPU：Intel i5-12500H
- GPU：NVIDIA Geforce RTX 2050（4GB）

2.2 软件环境

- Windows 11
- Anaconda

2.3 Conda环境

- python=3.9
- pytorch=2.0.0
- pytorch-cuda=11.8
- torchaudio=2.0.0

3 实验思路

3.1 数据来源

由于使用的数据集为 `MNIST`，因此选择直接通过 `torchvision` 进行导入。

3.2 数据预处理

该数据集无需进行预处理。

3.3 模型选择与参数设置

模型选择使用CNN

参数设置如下：

```
n_epochs = 5
batch_size_train = 64
batch_size_test = 1000
learning_rate = 0.01
momentum = 0.5
log_interval = 10
random_seed = 42
```

4 实验代码

4.1 数据载入部分

```
train_loader = torch.utils.data.DataLoader(
                torchvision.datasets.MNIST('./data/', train=True, download=True,
transform=torchvision.transforms.Compose([
torchvision.transforms.ToTensor(),
torchvision.transforms.Normalize(
                                                (0.1307,), (0.3081,))
                                        ])),
                batch_size=batch_size_train, shuffle=True)
test_loader = torch.utils.data.DataLoader(
                torchvision.datasets.MNIST('./data/', train=False, download=True,
transform=torchvision.transforms.Compose([
torchvision.transforms.ToTensor(),
torchvision.transforms.Normalize(
                                                (0.1307,), (0.3081,))
                                        ])),
                batch_size=batch_size_test, shuffle=True)
```

4.2 模型部分

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)
    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)


network = Net()
network = network.cuda()
optimizer = optim.SGD(network.parameters(), lr=learning_rate,
momentum=momentum)
```

4.3 训练部分

```
train_losses = []
train_counter = []
test_losses = []
test_counter = [i*len(train_loader.dataset) for i in range(n_epochs + 1)]


def train(epoch):
    network.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data = data.cuda()
        target = target.cuda()

        optimizer.zero_grad()
        output = network(data)
        loss = F.nll_loss(output, target).cuda()
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                    epoch, batch_idx * len(data), len(train_loader.dataset),
                    100. * batch_idx / len(train_loader), loss.item()))
            train_losses.append(loss.item())
```

```
            train_counter.append((batch_idx*64) + ((epoch-
1)*len(train_loader.dataset)))
            torch.save(network.state_dict(), './model.pth')
            torch.save(optimizer.state_dict(), './optimizer.pth')


def test():
    network.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data = data.cuda()
            target = target.cuda()

            output = network(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.data.max(1, keepdim=True)[1].cuda()
            correct += pred.eq(target.data.view_as(pred)).sum()
    test_loss /= len(test_loader.dataset)
    test_losses.append(test_loss)
    print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{}
({:.0f}%)\n'.format(
            test_loss, correct, len(test_loader.dataset),
            100. * correct / len(test_loader.dataset)))


test()
for epoch in range(1, n_epochs + 1):
    train(epoch)
    test()
```

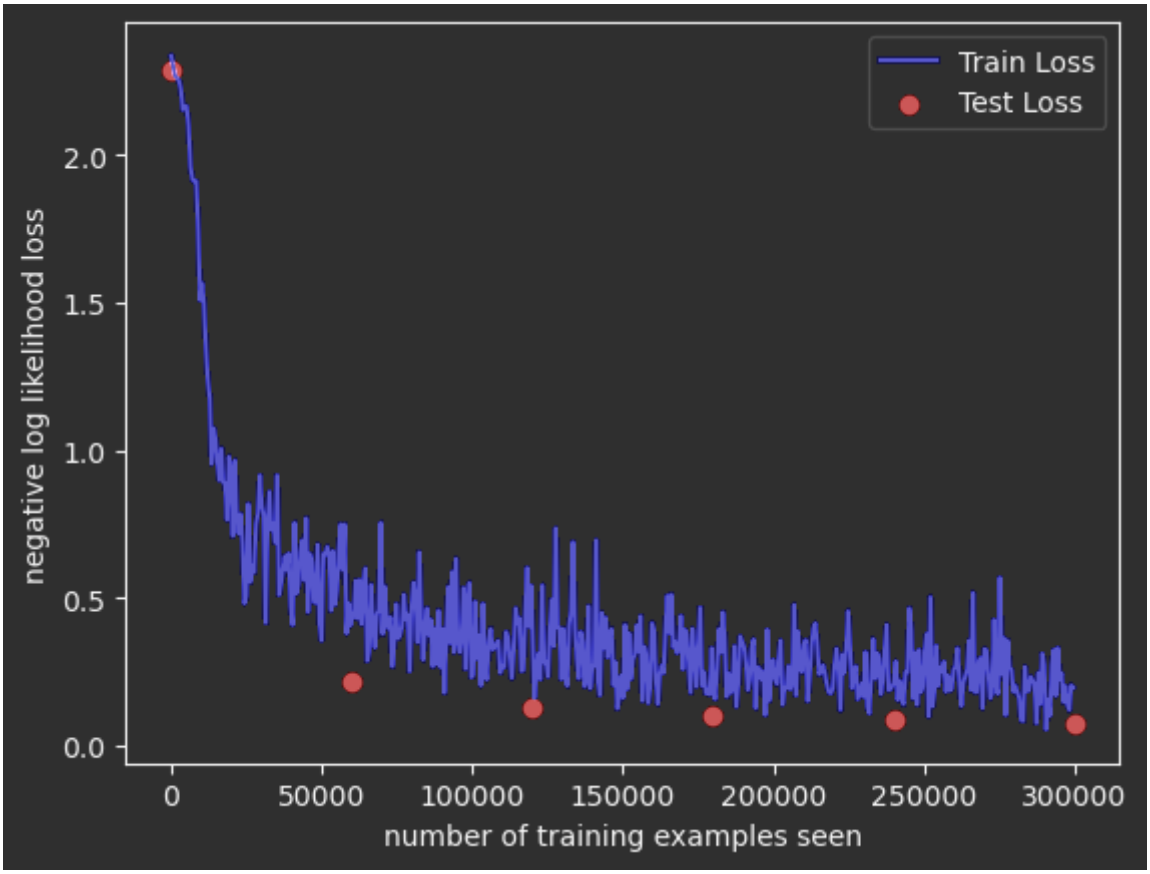
5 实验结果

5.1 训练部分输出展示

```
Test set: Avg. loss: 2.2913, Accuracy: 732/10000 (7%)

Train Epoch: 1 [0/60000 (0%)]   Loss: 2.336169
Train Epoch: 1 [640/60000 (1%)] Loss: 2.301624
Train Epoch: 1 [1280/60000 (2%)]    Loss: 2.268309
Train Epoch: 1 [1920/60000 (3%)]    Loss: 2.262120
Train Epoch: 1 [2560/60000 (4%)]    Loss: 2.252974
Train Epoch: 1 [3200/60000 (5%)]    Loss: 2.224842

.............


```

```
Train Epoch: 5 [56960/60000 (95%)]  Loss: 0.201114
Train Epoch: 5 [57600/60000 (96%)]  Loss: 0.139273
Train Epoch: 5 [58240/60000 (97%)]  Loss: 0.120702
Train Epoch: 5 [58880/60000 (98%)]  Loss: 0.203387
Train Epoch: 5 [59520/60000 (99%)]  Loss: 0.196420

Test set: Avg. loss: 0.0757, Accuracy: 9757/10000 (98%)
```

5.2 Loss图像



5 总结

整体而言，本实验并不难，只需要非常简单的网络就可以达到对样本集不错的识别效果。而且由于是非常经典的实验之一，`torch`内置了样本集，无需预处理可以直接导入，这也大大降低了实验难度。

通过本次实验，学习了整个模型训练与测试的过程，并且知道了如何记录loss并画出图像。