机器学习 第二次作业

神经网络作业

201250076 袁家乐

2023 年 3 月 16 日

一、过程说明

本次作业通过构造了一个五层的卷积神经网络，以对 Mnist 这一灰度手写数字图片数据集进行训练和测试，实现手写体识别的任务。

实验使用 python 代码，用到了 CPU 版本的 pytorch 框架。主要实现了 CNN 类，my_train()训练函数，my_test()测试函数等，如下：

main 函数入口

class CNN(my_nn.Module) #构造出的卷积神经网络

def my_train(counter) #模型训练

def my_test() #模型测试

具体的流程上，首先获取和加载 pytorch 提供的 Mnist 数据集，存放在 dataset 目录下。接下来构造一个五层的 CNN 并定义 forward()，将该模型投放在 CPU 上运行，定义好损失函数和优化函数。以上的准备工作完成后，进行十次训练，每一次训练结束后都在测试集上进行一次测试，观察准确率。

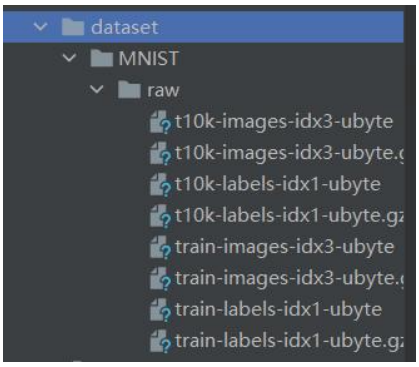
下面，针对上述流程进行逐条论述。

1、获取和加载 pytorch 提供的 Mnist 数据集

在获取和加载数据集的过程中，对数据同时做规范化，将原数据规范到(0,1)的区间内，方便后续的处理。其中数据的加载用到了 torchvision.datasets 中的 DataLoader。数据存放在 dataset 目录下。

```
# 使用pytorch提供的Mnist数据集
# 获取训练集，并将原数据规范到(0,1)区间
train_dataset = my_datasets.MNIST(root='./dataset', train=True, transform=my_transforms.ToTensor(), download=True)
train_data_loader = my_data_utils.DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
# 获取测试集，并将原数据规范到(0,1)区间
test_dataset = my_datasets.MNIST(root='./dataset', train=False, transform=my_transforms.ToTensor(), download=True)
test_data_loader = my_data_utils.DataLoader(dataset=test_dataset, batch_size=64, shuffle=True)
```

获取到的数据存放在 dataset 目录下，详细的存放路径是/dataset/MNIST/raw。



2、构造一个五层的 CNN 并定义 forward()

各层的说明见下表：

层号	类型	核大小	通过前	通过后
1	卷积层	3	1×28×28	25×26×26
2	采样层最大池化	2	25×26×26	25×13×13
3	卷积层	3	25×13×13	50×13×13
4	采样层最大池化	2	50×13×13	50×5×5
5	全连接层	/	50×5×5	10

这里的各层均使用 RelU 激活函数，保持正值不变，非正值则全部归一化为 0。在全连接层，首先将 50×5×5 的神经元通过线性层规约为 1024 个神经元，再进一步规约为 128 个神经元，最后规约为 10 个神经元，这 10 个最终的神经元即分别代表了数字 0~9。

```
# 输入层为1x28x28
# 第一层为卷积层，通过后应为25X26X26
self.layer1 = my_nn.Sequential(
    my_nn.Conv2d(1, 25, kernel_size=3),
    my_nn.BatchNorm2d(25),
    # 使用ReLU激活函数，正值不变，非正值返回0
    my_nn.ReLU(inplace=True)
)
# 第二层为采样层做最大池化，通过后为25X13X13
self.layer2 = my_nn.Sequential(
    my_nn.MaxPool2d(kernel_size=2, stride=2)
)
# 第三层为卷积层，通过后为50X11X11
self.layer3 = my_nn.Sequential(
    my_nn.Conv2d(25, 50, kernel_size=3),
    my_nn.BatchNorm2d(50),
    # 使用ReLU激活函数，正值不变，非正值返回0
    my_nn.ReLU(inplace=True)
)
# 第四层为采样层做最大池化，通过后为50X5X5
self.layer4 = my_nn.Sequential(
    my_nn.MaxPool2d(kernel_size=2, stride=2)
)
```

```
        # 第五层为全连接层
        self.layer5 = my_nn.Sequential(
            my_nn.Linear(50 * 5 * 5, 1024),
            my_nn.ReLU(inplace=True),
            my_nn.Linear(1024, 128),
            my_nn.ReLU(inplace=True),
            # 最终判定出的是0~9这10个数字
            my_nn.Linear(128, 10)
        )
```

对于 forward()函数，即按照五层从前到后进行顺序前进。需特别注意的是，在进入第五层前，需将连续范围的维度展开拉平，

```
    def forward(self, x):
        # 首先走过前四层
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        # 进入全连接层，数据需预先flatten
        x = x.view(x.size(0), -1)
        x = self.layer5(x)
        return x
```

3、将该模型投放在 CPU 上运行，定义好损失函数和优化函数

实验时，本机的 pytorch 不太适用于 GPU，故将其放在 CPU 上运行。损失函数使用交叉熵损失。优化函数使用随机梯度下降。

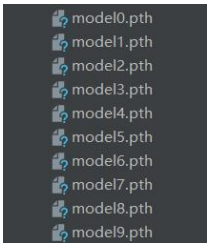
```
# 将模型投放在CPU上运行
cnn = CNN()
device = torch.device("cpu")
cnn.to(device)

# 定义损失函数（用交叉熵损失）
loss = my_nn.CrossEntropyLoss()

# 定义优化函数（随机梯度下降，初始学习率置为0.1，动量置为0.5）
optimizer = torch.optim.SGD(cnn.parameters(), lr=0.1, momentum=0.5)
```

4、十次训练和测试

每一次训练时，将输入和目标数据均放在 CPU 上运行，每次迭代过程均为前馈学习—反馈损失—优化更新。在 900 次小迭代中，为更好地观察结果，在控制台打印每 100 次的平均损失量。每次训练完毕，都将模型进行存储。



```python
def my_train(counter):
    # 整体的损失量
    total_loss = 0.0
    for batch_idx, data in enumerate(train_data_loader, 0):
        # 将输入和目标数据放在CPU上运行
        inputs, target = data
        inputs, target = inputs.to(device), target.to(device)
        optimizer.zero_grad()
        # 前馈学习
        outputs = cnn(inputs)
        # 反馈损失
        loss_unit = loss(outputs, target)
        loss_unit.backward()
        # 优化更新
        optimizer.step()
        total_loss += loss_unit.item()
        # 输出迭代过程中每100次迭代的平均损失
        if batch_idx % 100 == 99:
            print('counter %d iteration from %5d to %5d average loss:%.3f' % (
            counter, batch_idx - 99, batch_idx + 1, total_loss / 100))
            total_loss = 0.0
    torch.save(cnn, 'model{}.pth'.format(counter))
```

每次训练完毕后，都进行测试。同样的，测试时也将输入和目标数据放在 CPU 上运行，特别需注意的是在测试时无需再对梯度进行计算了。比对输出的预测结果和目标结果，计算准确率并打印到控制台。

```python
def my_test():
    correct_num = 0
    total_num = 0
    # 测试时无需计算梯度
    with torch.no_grad():
        for data in test_data_loader:
            # 将输入和目标数据放在CPU上运行
            inputs, target = data
            inputs, target = inputs.to(device), target.to(device)
            outputs = cnn(inputs)
            # 获取输出数据每行最大值的下标，与target进行比较，以判断准确率
            _, predicted = torch.max(outputs.data, dim=1)
            total_num += target.size(0)
            correct_num += (predicted == target).sum().item()
    accuracy = (float(correct_num) / float(total_num)) * 100
    print("Test Accuracy:{}%".format(accuracy))
```

二、样例截图与模型准确率

十次训练的模型各自的准确率见下表：

计数	0	1	2	3	4	5	6	7	8	9
准确率	98.10%	98.76%	99.02%	99.01%	99.03%	99.22%	99.28%	99.30%	99.28%	99.28%

可见，本模型的准确率基本稳定在 99.28%左右，模型的准确率较高。

数据加载和获取时的控制台截图如下：

```
G:\anaconda3\envs\pytorch\python.exe D:/2023春/机器学习/作业2/code/main.py
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./dataset\MNIST\raw\train-images-idx3-ubyte.gz
100.0%
Extracting ./dataset\MNIST\raw\train-images-idx3-ubyte.gz to ./dataset\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./dataset\MNIST\raw\train-labels-idx1-ubyte.gz
100.0%
Extracting ./dataset\MNIST\raw\train-labels-idx1-ubyte.gz to ./dataset\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./dataset\MNIST\raw\t10k-images-idx3-ubyte.gz
100.0%
Extracting ./dataset\MNIST\raw\t10k-images-idx3-ubyte.gz to ./dataset\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./dataset\MNIST\raw\t10k-labels-idx1-ubyte.gz
Extracting ./dataset\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./dataset\MNIST\raw

100.0%
```

各轮训练中每 100 次迭代的损失量和每轮训练模型在测试集上的准确率控制台截图如下：

```
counter 0 iteration from     0 to   100 average loss:0.554
counter 0 iteration from   100 to   200 average loss:0.127
counter 0 iteration from   200 to   300 average loss:0.100
counter 0 iteration from   300 to   400 average loss:0.088
counter 0 iteration from   400 to   500 average loss:0.082
counter 0 iteration from   500 to   600 average loss:0.068
counter 0 iteration from   600 to   700 average loss:0.058
counter 0 iteration from   700 to   800 average loss:0.064
counter 0 iteration from   800 to   900 average loss:0.059
Test Accuracy:98.1%
counter 1 iteration from     0 to   100 average loss:0.048
counter 1 iteration from   100 to   200 average loss:0.044
counter 1 iteration from   200 to   300 average loss:0.038
counter 1 iteration from   300 to   400 average loss:0.042
counter 1 iteration from   400 to   500 average loss:0.035
counter 1 iteration from   500 to   600 average loss:0.037
counter 1 iteration from   600 to   700 average loss:0.047
counter 1 iteration from   700 to   800 average loss:0.033
counter 1 iteration from   800 to   900 average loss:0.036
Test Accuracy:98.76%
counter 2 iteration from     0 to   100 average loss:0.029
counter 2 iteration from   100 to   200 average loss:0.028
counter 2 iteration from   200 to   300 average loss:0.024
counter 2 iteration from   300 to   400 average loss:0.031
counter 2 iteration from   400 to   500 average loss:0.028
```

```
 counter 2 iteration from   100 to   200 average loss:0.028
 counter 2 iteration from   200 to   300 average loss:0.024
 counter 2 iteration from   300 to   400 average loss:0.031
 counter 2 iteration from   400 to   500 average loss:0.028
 counter 2 iteration from   500 to   600 average loss:0.029
 counter 2 iteration from   600 to   700 average loss:0.029
 counter 2 iteration from   700 to   800 average loss:0.021
 counter 2 iteration from   800 to   900 average loss:0.025
 Test Accuracy:99.02%
 counter 3 iteration from     0 to   100 average loss:0.014
 counter 3 iteration from   100 to   200 average loss:0.018
 counter 3 iteration from   200 to   300 average loss:0.014
 counter 3 iteration from   300 to   400 average loss:0.023
 counter 3 iteration from   400 to   500 average loss:0.014
 counter 3 iteration from   500 to   600 average loss:0.024
 counter 3 iteration from   600 to   700 average loss:0.019
 counter 3 iteration from   700 to   800 average loss:0.023
 counter 3 iteration from   800 to   900 average loss:0.025
 Test Accuracy:99.00999999999999%
 counter 4 iteration from     0 to   100 average loss:0.011
 counter 4 iteration from   100 to   200 average loss:0.013
 counter 4 iteration from   200 to   300 average loss:0.009
 counter 4 iteration from   300 to   400 average loss:0.016
 counter 4 iteration from   400 to   500 average loss:0.013
 counter 4 iteration from   500 to   600 average loss:0.017
```

```
counter 4 iteration from   500 to   600 average loss:0.017
counter 4 iteration from   600 to   700 average loss:0.015
counter 4 iteration from   700 to   800 average loss:0.022
counter 4 iteration from   800 to   900 average loss:0.013
Test Accuracy:99.03%
counter 5 iteration from     0 to   100 average loss:0.010
counter 5 iteration from   100 to   200 average loss:0.008
counter 5 iteration from   200 to   300 average loss:0.005
counter 5 iteration from   300 to   400 average loss:0.012
counter 5 iteration from   400 to   500 average loss:0.011
counter 5 iteration from   500 to   600 average loss:0.015
counter 5 iteration from   600 to   700 average loss:0.013
counter 5 iteration from   700 to   800 average loss:0.008
counter 5 iteration from   800 to   900 average loss:0.010
Test Accuracy:99.22%
counter 6 iteration from     0 to   100 average loss:0.005
counter 6 iteration from   100 to   200 average loss:0.003
counter 6 iteration from   200 to   300 average loss:0.010
counter 6 iteration from   300 to   400 average loss:0.009
counter 6 iteration from   400 to   500 average loss:0.010
counter 6 iteration from   500 to   600 average loss:0.009
counter 6 iteration from   600 to   700 average loss:0.010
counter 6 iteration from   700 to   800 average loss:0.010
counter 6 iteration from   800 to   900 average loss:0.014
Test Accuracy:99.28%
```

```
counter 6 iteration from   800 to   900 average loss:0.014
Test Accuracy:99.28%
counter 7 iteration from     0 to   100 average loss:0.006
counter 7 iteration from   100 to   200 average loss:0.004
counter 7 iteration from   200 to   300 average loss:0.003
counter 7 iteration from   300 to   400 average loss:0.006
counter 7 iteration from   400 to   500 average loss:0.004
counter 7 iteration from   500 to   600 average loss:0.004
counter 7 iteration from   600 to   700 average loss:0.006
counter 7 iteration from   700 to   800 average loss:0.005
counter 7 iteration from   800 to   900 average loss:0.007
Test Accuracy:99.3%
counter 8 iteration from     0 to   100 average loss:0.003
counter 8 iteration from   100 to   200 average loss:0.002
counter 8 iteration from   200 to   300 average loss:0.002
counter 8 iteration from   300 to   400 average loss:0.003
counter 8 iteration from   400 to   500 average loss:0.006
counter 8 iteration from   500 to   600 average loss:0.002
counter 8 iteration from   600 to   700 average loss:0.002
counter 8 iteration from   700 to   800 average loss:0.006
counter 8 iteration from   800 to   900 average loss:0.007
Test Accuracy:99.28%
counter 9 iteration from     0 to   100 average loss:0.005
counter 9 iteration from   100 to   200 average loss:0.003
counter 9 iteration from   200 to   300 average loss:0.004
```

```
counter 7 iteration from   800 to   900 average loss:0.007
Test Accuracy:99.3%
counter 8 iteration from     0 to   100 average loss:0.003
counter 8 iteration from   100 to   200 average loss:0.002
counter 8 iteration from   200 to   300 average loss:0.002
counter 8 iteration from   300 to   400 average loss:0.003
counter 8 iteration from   400 to   500 average loss:0.006
counter 8 iteration from   500 to   600 average loss:0.002
counter 8 iteration from   600 to   700 average loss:0.002
counter 8 iteration from   700 to   800 average loss:0.006
counter 8 iteration from   800 to   900 average loss:0.007
Test Accuracy:99.28%
counter 9 iteration from     0 to   100 average loss:0.005
counter 9 iteration from   100 to   200 average loss:0.003
counter 9 iteration from   200 to   300 average loss:0.004
counter 9 iteration from   300 to   400 average loss:0.004
counter 9 iteration from   400 to   500 average loss:0.003
counter 9 iteration from   500 to   600 average loss:0.003
counter 9 iteration from   600 to   700 average loss:0.003
counter 9 iteration from   700 to   800 average loss:0.005
counter 9 iteration from   800 to   900 average loss:0.001
Test Accuracy:99.28%

Process finished with exit code 0
```