MNIST 手写字符识别实验报告

林宝诚 2019080030 无 98

系统环境：Windows 10 Anaconda

软件版本：PyTorch 1.8.1

1. 可调整参数
    - Image_size：图片大小
    - Batch_size：每一次训练的批次大小
    - Class_num：输出种类的数量
    - Learning_rate：学习率
    - Epochs：训练循环次数

2. 数据集的获取

    - 利用 Torchvision 中的 dataset 下载手写字符数据集，并储存在/data 文件夹内。
    - 训练集：60000；测试集：10000
    - 数据类别：1 x 28 x 28 图片

    **注：28 x 28 分别为图片的高和宽，1 是图片的通道（黑白图片的通道为 1，如果是彩色 RGB 图片则通道为 3，也有些 RGB 图片有多一个 Alpha 通道）

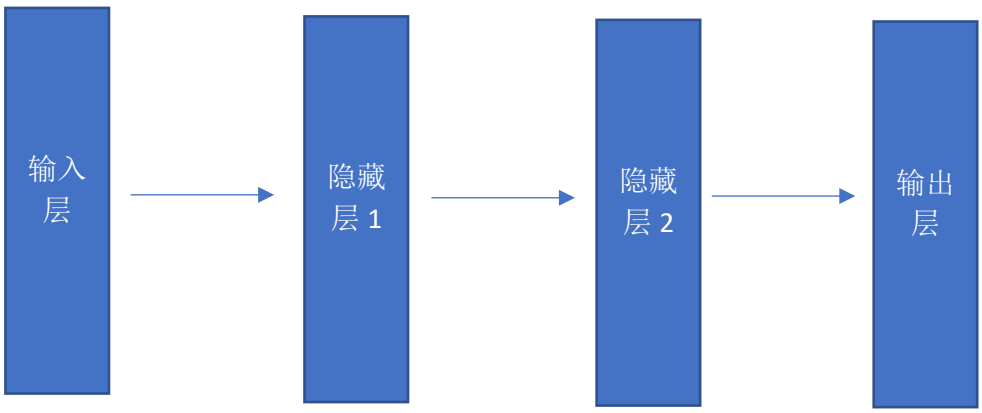
3. 数据集预处理
    - 首先，利用 Torchvision Transforms 中的 ToTensor 将 image 中的每一个像素从 1-255 缩放至 0-1。
    - 同时，为了使模型能够更快学习（更快 Converge），需要对图片做 Normalization，即将像素值的平均值变为 0，方差为 1。在此，利用了 Torchvision Transforms 中的 Normalize 完成此步骤。（MNIST 数据集的平均值为 0.1307，方差为 0.3081）

4. 数据集可视化
    - 处理好数据集后，随机显示图片进行大致的检查。利用 view_Image 函数可显示训练集中的第一个 Batch 的所有图片。

    **注：图片是随机显示的因为在载入图片的时候，使用了 shuffle 功能，因此每一次执行程时所显示的图片都不同。

5. 模型设计
    - 由于此任务较不复杂，因此使用较简单的深度网络架构

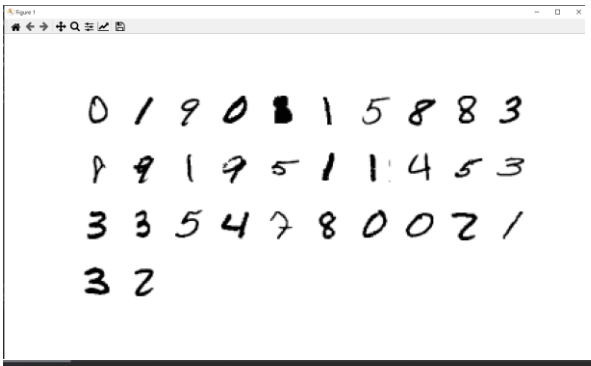
```python
class BaseModel(nn.Module):
    def __init__(self,input_size=784, output_size=10):
        super(BaseModel,self).__init__()
        self.linear1=nn.Linear(input_size,64)
        self.linear2=nn.Linear(64,32)
        self.linear3=nn.Linear(32,output_size)
        self.relu=nn.ReLU(inplace=True)
        self.softmax=nn.LogSoftmax(dim=1)

    def forward(self,x):
        x=self.linear1(x)
        x=self.relu(x)
        x=self.linear2(x)
        x=self.relu(x)
        x=self.linear3(x)
        x=self.softmax(x)
        return x
```



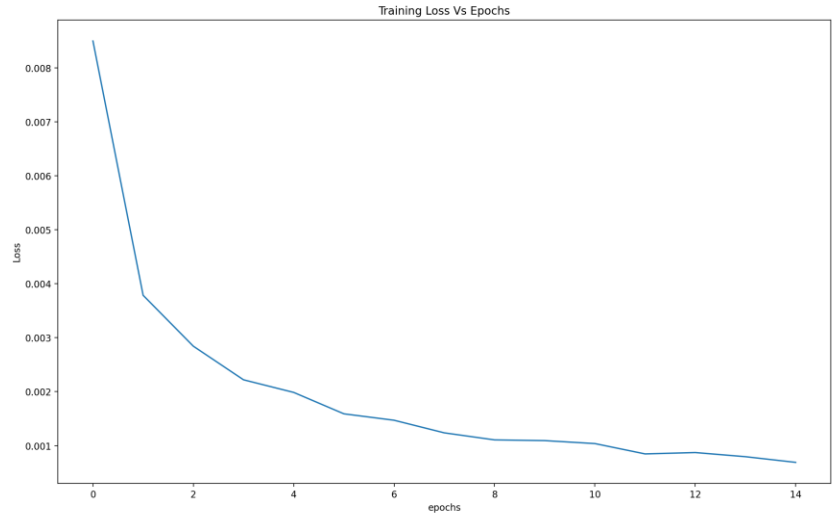
输入为一个向量，其长度为图片的高宽相乘，因此输入层的长度为向量的长度（对 MNIST 数据集长度为 784）。隐藏层 1 含 64 个神经元，隐藏层 2 含 32 个神经元，输出数量为用户定义（对 MNIST 数据集输出为 10）。两个隐藏层使用了 ReLU 激活函数以增加模型的非线性性质。输出层则利用了 LogSoftmax 激活函数，此激活函数适用于分类问题。

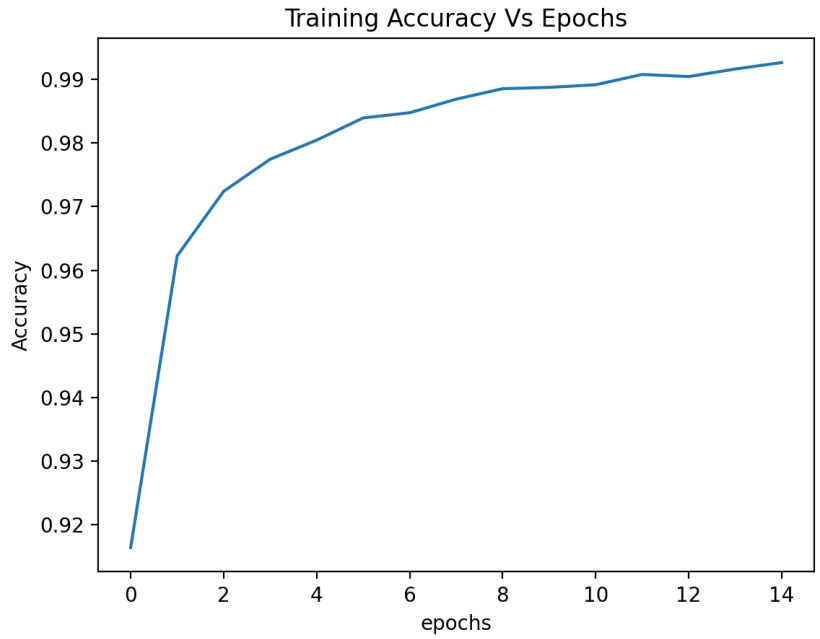
6. 训练 & 测试和结果可视化
    - 利用 SGD-M（Stochastic Gradient Descent with Momentum）优化算法进行参数优化
    - 损失函数定义为 NLLLoss（Negative Log Likelihood Loss），此损失函数是与 LogSoftmax 一起使用的。
    - 训练过程中，计算每一轮循环的损失（Loss）和训练准确率。
    - 完成训练后，输出损失和训练准确率随时间变化的曲线。
    - 最后，利用测试集对模型进行测试，并输出测试准确率（97.25%）。

运行 & 可视化结果：



```
D:\Anaconda\envs\pytorch\python.exe "C:/Users/MY PC/PycharmProjects/mnistProject/main.py"
----------Start Training----------
Train Epoch:0 - Loss:0.008496755981779036
Training Accuracy:  0.9164333343505859
Train Epoch:1 - Loss:0.0037851302180322817
Training Accuracy:  0.9622499942779541
Train Epoch:2 - Loss:0.002842202604317572
Training Accuracy:  0.9724000096321106
Train Epoch:3 - Loss:0.002219647099172774
Training Accuracy:  0.9774333238601685
Train Epoch:4 - Loss:0.0019856917142906847
Training Accuracy:  0.9804333448410034
Train Epoch:5 - Loss:0.0015891576894440126
Training Accuracy:  0.9839166402816772
Train Epoch:6 - Loss:0.0014699761611120872
Training Accuracy:  0.9847333431243896
Train Epoch:7 - Loss:0.0012367935187464658
Training Accuracy:  0.9868666529655457
Train Epoch:8 - Loss:0.0011063070411837846
Training Accuracy:  0.9885166883468628
Train Epoch:9 - Loss:0.001092763483229252
Training Accuracy:  0.9887166619300842
Train Epoch:10 - Loss:0.001037837166249786
Training Accuracy:  0.989133358001709
Train Epoch:11 - Loss:0.0008464294474651221
Training Accuracy:  0.9907500147819519
Train Epoch:12 - Loss:0.0008715730119049643
Training Accuracy:  0.9904166460037231
Train Epoch:13 - Loss:0.0007948860285036138
Training Accuracy:  0.991599977016449
Train Epoch:14 - Loss:0.0006897459290062671
Training Accuracy:  0.9926166534423828
----------End Training----------
----------Start Testing----------
Test accuracy:  0.9725000262260437
----------Done Testing and model is saved as "my_model.pt"----------
```





参考资料

1. PyTorch 官网
2. https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627