前馈神经网络实验报告

姓名：朱浩泽　学号：1911530

March 25, 2022

1　原始版本 MLP

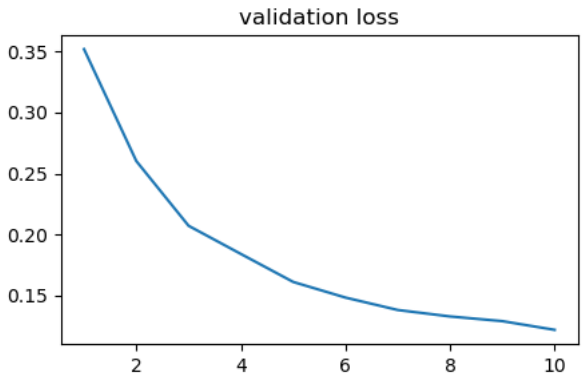
1.1　网络结构

我们通过 Pycharm 软件的 python 控制台查看运行时的变量，可以看到网络结构如下：

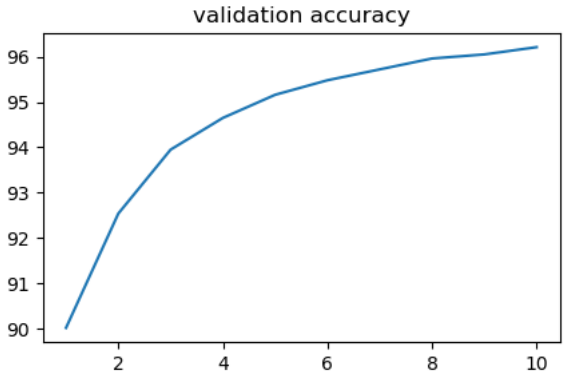
```
Net(
  (fc1): Linear(in_features=784, out_features=50, bias=True)
  (fc1_drop): Dropout(p=0.2, inplace=False)
  (fc2): Linear(in_features=50, out_features=50, bias=True)
  (fc2_drop): Dropout(p=0.2, inplace=False)
  (fc3): Linear(in_features=50, out_features=10, bias=True)
)
```

第一层是一个全连接层，将 28*28 的图片映射到维度为 50 的空间上，防止过拟合，在训练时随机以一定的概率丢掉一些数据，丢掉 20% 的数据；然后是一个 50 维到 50 维的全连接层，得到映射后再进行 20% 的随机丢弃；最后再通过一个 50 维到 10 维的全连接层将图像映射到 0 到 9 这 10 个数字类别上。

1.2　训练结果



训练 Loss 曲线



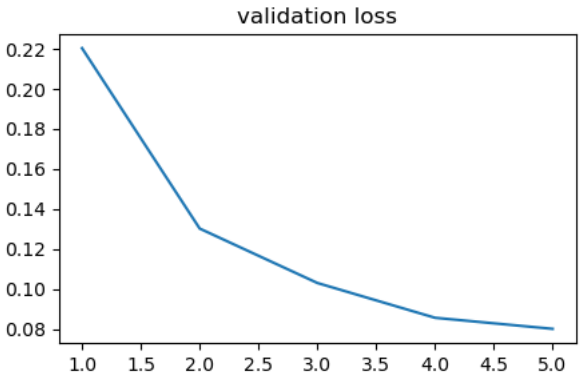
准确度曲线图

最终，在经历十轮迭代训练后，在验证集上的准确率在 96% 左右

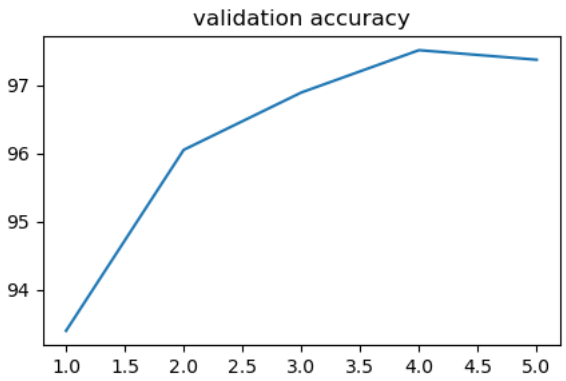
2 个人改进后的 MLP

2.1 改进后的实验效果

2.1.1 epoch = 5



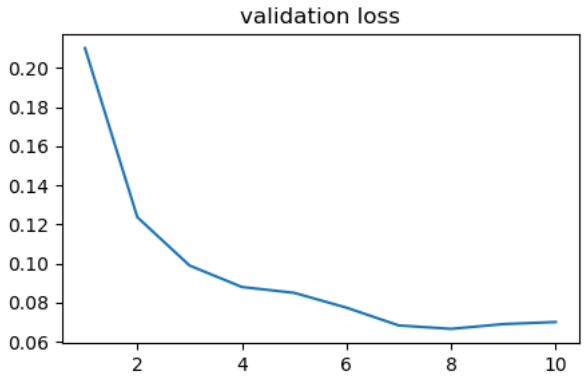
训练 Loss 曲线



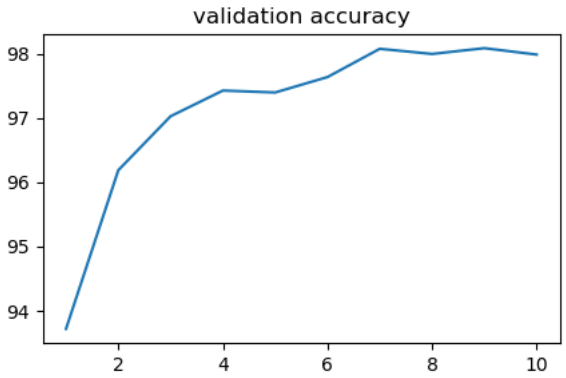
准确度曲线图

最终，在经历五轮迭代训练后，在验证集上的准确率在 97% 左右

2.1.2 epoch = 10



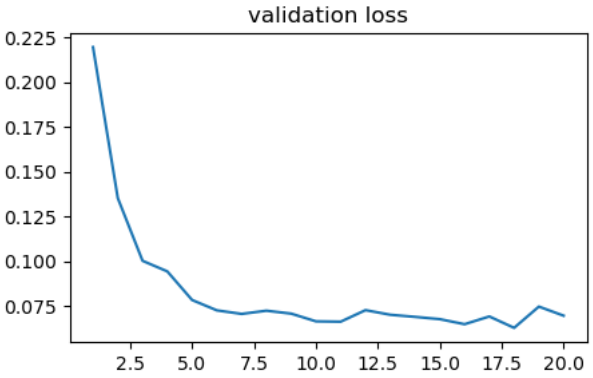
训练 Loss 曲线



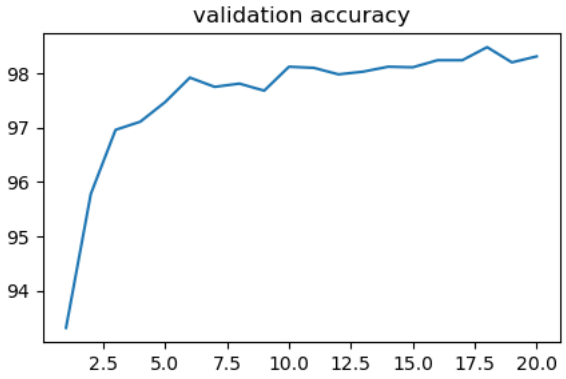
准确度曲线图

最终，在经历十轮迭代训练后，在验证集上的准确率在 98% 左右

2.1.3 epoch = 20



训练 Loss 曲线



准确度曲线图

最终，在经历二十轮迭代训练后，在验证集上的准确率在 98% 左右

2.2 改进与个人心得

2.2.1 网络结构

我们通过 Pycharm 软件的 python 控制台查看运行时的变量，可以看到网络结构如下：

```
Net(
  (fc1): Linear(in_features=784, out_features=400, bias=True)
  (fc1_drop): Dropout(p=0.2, inplace=False)
  (fc2): Linear(in_features=400, out_features=80, bias=True)
  (fc2_drop): Dropout(p=0.2, inplace=False)
  (fc3): Linear(in_features=80, out_features=50, bias=True)
  (fc3_drop): Dropout(p=0.2, inplace=False)
  (fc4): Linear(in_features=50, out_features=10, bias=True)
)
```

2.2.2 改进之处和原理

- 首先第一个全连接层的维度由 50 维改为 400 维，这样做的原因是因为更大的维度的映射可能可以更好的保存原始图像的特征，带来更好的效果。
- 然后下一个全连接层将上一层的 400 维的数据映射为 80 维的数据，可以有效的提取重要特征，并对计算量有一定的减小
- 接下来在原有的基础上，我又增加了一层全连接层，将上一层处理完的 80 维的特征映射到 50 维的线性空间上，再经过这个全连接层将 50 维的向量映射到 10 个类别上完成分类。增加的这一层全连接层可以更好地提取每个类别的特点，细化了分类的过程，对提升准确率有着一定的帮助，但同时可能会有过拟合的风险。

- 对于训练率我们采用了与原方法相同的 0.01，激活函数在测试了 Sigmoid 型函数效果极其不理想，故更换为原本的 RelU 函数，通过我们之前学习的知识也可以知道，该激活函数可以更好的拟合非线性的输入和输出特征的映射，比较符合该问题的处理方式。至于动量，由于我们将层数增加，且每层映射的线性空间增大，故将其设置为 0.8。每一层之间的 drop_out 随机丢弃率仍然为 0.2，损失函数仍然是交叉熵函数。
- 随后我又用卷积的方法进行了测试，因为图片本身是二维向量，二维处理的效果应该是优于一维的，而实际经过测试，经过两层卷积和一层隐层的效果可以轻松达到 98%，只是训练时间较长，训练数据量较大。

2.2.3 实验心得

通过这次实验，我对 Pytorch 的使用有了深入的理解。首先是 torch.nn 模块，该模块定义好了各个神经层的结构 (如卷积层、全连接层等)，可以直接调用使用，torch.nn.functional 中定义了常见的函数等，torch.utils.data.DataLoader 则是对数据集进行细分的处理。总体来说，只要设计出了网络结构，那么搭建网络换个训练的过程就像是搭积木一样，一层层调用即可。对于 MLP 经网络，我更好的了解了张量在每一层之间的向下传递过程，以及在训练时通过损失函数向前反馈，通过求导进行计算梯度进行梯度下降更新参数的过程，对深度学习有了更好的认识。