南開大學
Nankai University

南　开　大　学

计 算 机 学 院

深度学习及应用实验作业

作业一 前馈神经网络实践

姓名：徐宇昂

学号：2011742

年级：2020 级

专业：计算机科学与技术

指导教师：侯淇彬

2023 年 4 月

摘要

本次实验基于 pyotch1.12.0，采用前馈神经网络完成手写数字识别实验并进行调参优化。

关键字：前馈神经网络，pytorch，FNN

目录

一、 实验要求

- 掌握 PyTorch 框架基础算子操作
- 学会使用 PyTorch 搭建简单的前馈神经网络来训练 MNIST 数据集
- 了解如何改进网络结构、调试参数以提升网络识别性能

二、 MLP 修改过程

(一) 加一层中间层

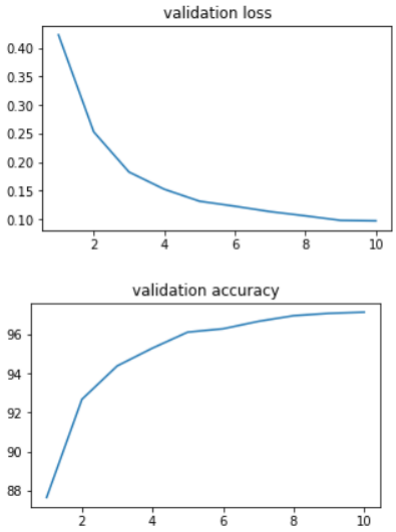
在老师给出的 KNN 实验指导中里面只有一个中间层，在这里我又加入一个 dropout 为 0.2 的中间层，网络结果如下：

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 100)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(100, 50)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(50, 50)
        self.fc3_drop = nn.Dropout(0.2)
        self.fc4 = nn.Linear(50, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)
        x = F.relu(self.fc3(x))
        x = self.fc3_drop(x)
        return F.log_softmax(self.fc4(x), dim=1)
```

(二) 实验结果

如图??所示



(三) 修改 dropout

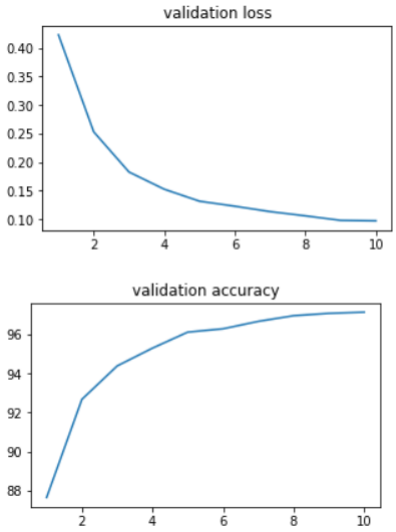
没有添加 Dropout 的网络是需要对网络的每一个节点进行学习的，而添加了 Dropout 之后的网络层只需要对该层中没有被 Mask 掉的节点进行训练，没有设置 dropout 或者 dropout 过小都会造成过拟合，在比较深的网络中，使用 0.5 的丢失率是比较好的选择，因为这时 Dropout 能取到最大的正则效果；在比较浅层的网络中，丢失率应该低于 0.5 ，因为过多的丢失率会导致丢失过多的输入数据对模型的影响比较大；不建议使用大于 0.5 的丢失率，因为它在丢失过多节点的情况下并不会取得更好的正则效果。

```
    class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 100)
        self.fc1_drop = nn.Dropout(0.3)
        self.fc2 = nn.Linear(100, 50)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(50, 50)
        self.fc3_drop = nn.Dropout(0.2)
        self.fc4 = nn.Linear(50, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)
        x = F.relu(self.fc3(x))
        x = self.fc3_drop(x)
        return F.log_softmax(self.fc4(x), dim=1)
```

(四) 实验结果

如图??所示

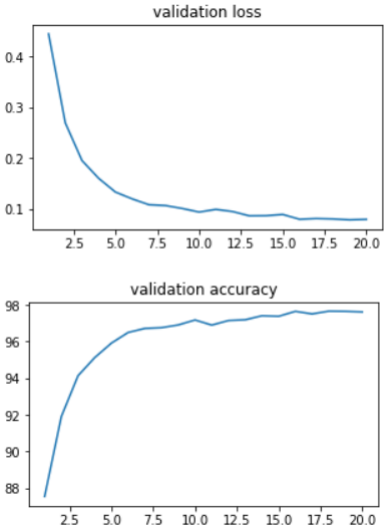


（五） 修改 epoch

从 10 个 epochs 修改到 20 个和 5 个。

（六） 实验结果

如图??所示



三、 实验比对心得

hidden_layer	*dropout*	*epochs*	*acc*
1	0.2	5	0.9325
1	0.2	10	0.9451
1	0.2	20	0.9632
2	0.2	5	0.9592
2	0.2	10	0.9713
2	0.2	20	0.9766
2	0.3	5	0.9462
2	0.3	10	0.9633
2	0.3	20	0.9652

表 1: 实验结果比对

根据实验中的数据来看，我们可以得出以下结论：

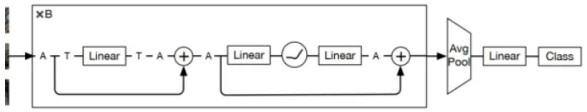
- 在相同 epochs、dropout 的前提下，增加一层隐藏层的模型训练结果在验证集上表现的正确率要比只有一层隐藏层的模型结果要高一些，原因在于在这个数据集上增加一层隐藏层可以增加一层新的函数映射，能够提取到更深层次的信息，使得模型拟合的结果越来越好；
- 在相同隐藏层、dropout 的前提下，适度增加模型训练次数，在这个数据集上模型的正确率也会逐渐增加，说明在此前提下模型参数更新的次数适当增多，使得模型拟合的效果更好；
- 在相同隐藏层、epochs 的前提下，适度减少 dropout，模型的正确率会适度提升。原因在于本网络只有 1 或 2 层中间层，丢失率应设置的小一些且低于 0.5，因为过多的丢失率会导致丢失过多的输入数据对模型参数的影响，导致模型欠拟合情况的发生。

因此，在本实验中，构建适当多的中间层，适当减少 dropout，适当增加训练次数 epochs，能够使得模型在验证集的正确率适当提高，使得模型的拟合效果好一些。

四、 resMLP

(一) resMLP 网络结构

resMLP 是一种完全基于 MLP 进行图像分类的体系结构，它是一种交替执行如下两个模块的简单残差网络：(1) 一个作用于图像块的线性层，独立于通道；(2) 一个作用于通道的两层前馈网络，独立于图像块。改进图示如下：



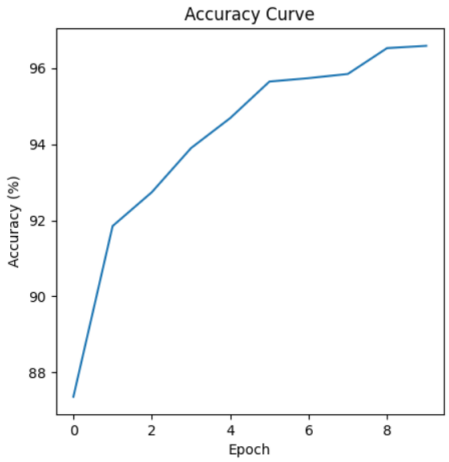
网络结构如下所示：

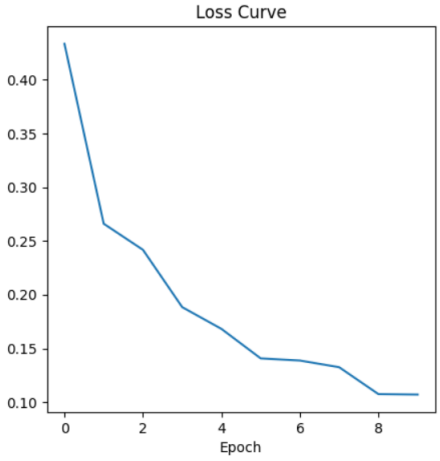
```
ResMLP(
  (patcher): Conv2d(1, 128, kernel_size=(4, 4), stride=(4, 4))
  (mlps): Sequential(
```

```
    (0): ResMLPLayer(
      (cl): CommunicationLayer(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=49, out_features=49, bias=True)
        (aff2): AffineTransform()
      )
      (ff): FeedForward(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=128, out_features=256, bias=True)
        (fc2): Linear(in_features=256, out_features=128, bias=True)
        (aff2): AffineTransform()
      )
    )
    (1): ResMLPLayer(
      (cl): CommunicationLayer(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=49, out_features=49, bias=True)
        (aff2): AffineTransform()
      )
      (ff): FeedForward(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=128, out_features=256, bias=True)
        (fc2): Linear(in_features=256, out_features=128, bias=True)
        (aff2): AffineTransform()
      )
    )
    (2): ResMLPLayer(
      (cl): CommunicationLayer(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=49, out_features=49, bias=True)
        (aff2): AffineTransform()
      )
      (ff): FeedForward(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=128, out_features=256, bias=True)
        (fc2): Linear(in_features=256, out_features=128, bias=True)
        (aff2): AffineTransform()
      )
    )
    (3): ResMLPLayer(
      (cl): CommunicationLayer(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=49, out_features=49, bias=True)
        (aff2): AffineTransform()
      )
      (ff): FeedForward(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=128, out_features=256, bias=True)
        (fc2): Linear(in_features=256, out_features=128, bias=True)
        (aff2): AffineTransform()
      )
    )
    (4): ResMLPLayer(
      (cl): CommunicationLayer(
        (aff1): AffineTransform()
```

```
        (fc1): Linear(in_features=49, out_features=49, bias=True)
        (aff2): AffineTransform()
      )
      (ff): FeedForward(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=128, out_features=256, bias=True)
        (fc2): Linear(in_features=256, out_features=128, bias=True)
        (aff2): AffineTransform()
      )
    )
    (5): ResMLPLayer(
      (cl): CommunicationLayer(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=49, out_features=49, bias=True)
        (aff2): AffineTransform()
      )
      (ff): FeedForward(
        (aff1): AffineTransform()
        (fc1): Linear(in_features=128, out_features=256, bias=True)
        (fc2): Linear(in_features=256, out_features=128, bias=True)
        (aff2): AffineTransform()
      )
    )
  )
  (classifier): Linear(in_features=128, out_features=10, bias=True)
)
```

(二) resMLP 训练过程中在测试集上的 loss 变化和准确率变化图





我们可以观察到由于 resMLP 增加了简单的块之间的线性交互，网络在获得更多的交互信息的前提下在该数据集上表现出不俗的竞争力。但是在数据集较小的前提下，该网络呈现出了较为严重的过拟合现象。