前馈神经网络实验报告

1911433 林坤

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、 实验内容

实验内容部分主要是记录了对不同网络结构和超参数的调整过程，和原始版本结果进行对比，为改进 MLP 网络提供对照。

（一） 原始版本 MLP

1. 网络结构

老师提供的原始版本 MLP 网络结构如下所示：

```
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)
)
```

其中各层网络如下：

· 输入层：以 28*28 一个图片规模大小输入全连接至 50 个输出

· Relu 激活函数

· 一层隐藏层：以 50 点输入 50 点输出

· Relu 激活函数

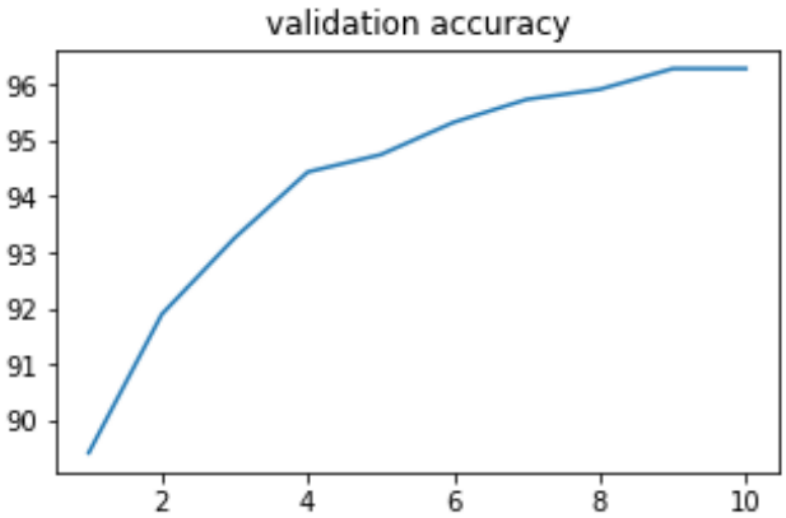
· 输出层：以 50 点输入 10 点输出

· 以 log_softmax 函数求出最大值 (最大概率分类) 作为输出

2. MNIST 验证集上 loss 和 accuracy 曲线

原始版本 MLP 在在 MNIST 验证集上的 loss 曲线如下图所示：

原始版本 MLP 在在 MNIST 验证集上的 accuracy 曲线如下图所示：



原始 MLP **网络结构在** 10 epochs **下，**Average loss **为** 0.1238**，**Accuracy **为** 9627/10000 (96%)**，耗时** 1min 36s。

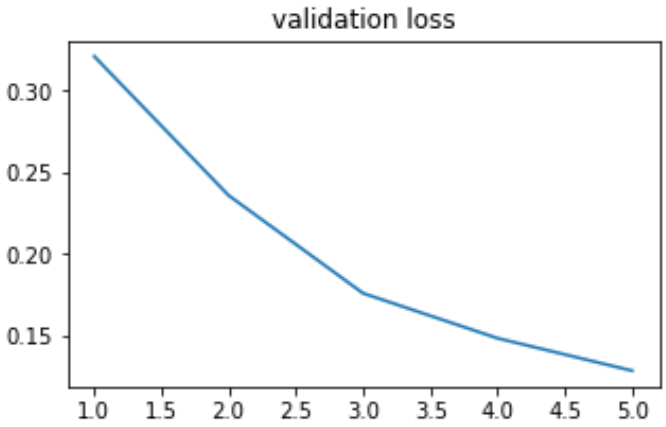
（二） 改进 MLP 网络结构

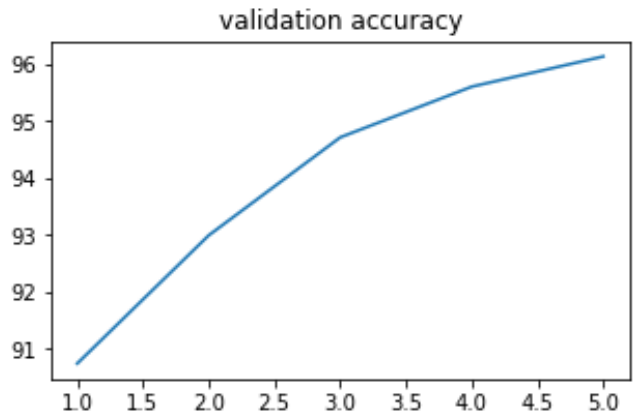
1. 增加网络宽度

在原有网络结构不变情况下，增加网络宽度。第二层网络两端宽度由 50 增加到 200

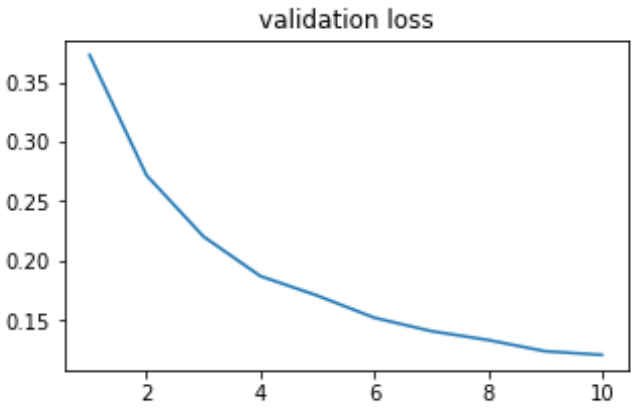
第二层网络宽度增加到 200 的预测结果如下：

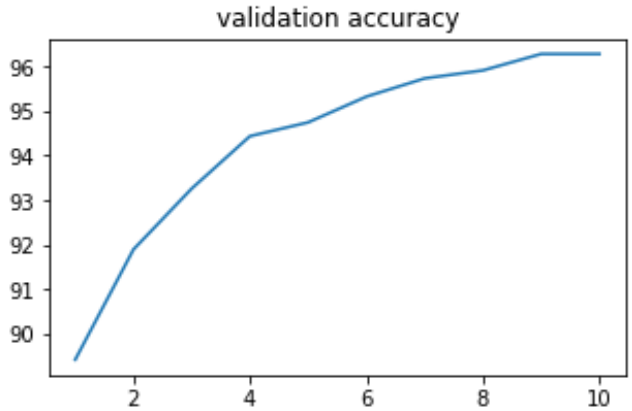
5 epochs，**损失曲线和准确率曲线如下：**



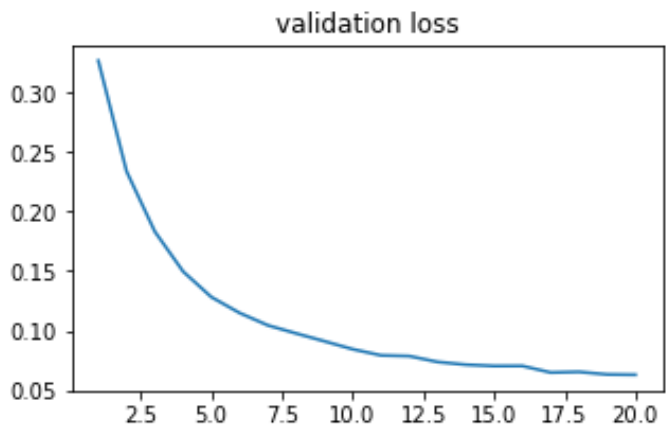


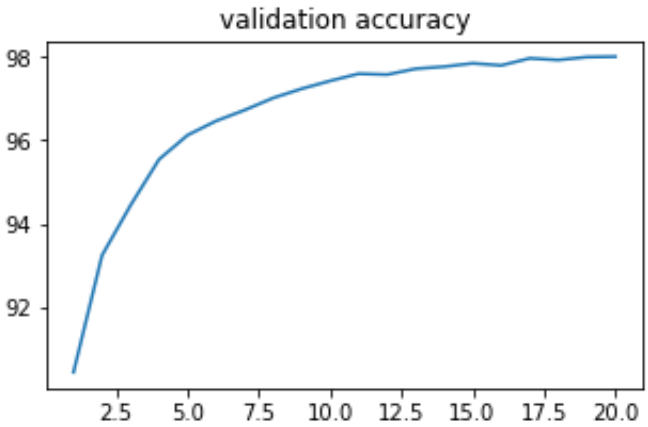
10 epochs，**损失曲线和准确率曲线如下：**





20 epochs，**损失曲线和准确率曲线如下：**





结果总结如下：

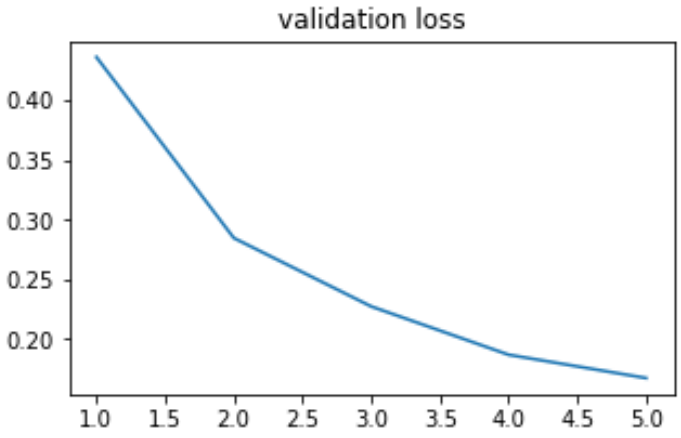
- 5 epochs:Average loss: 0.1283, Accuracy: 9633/10000 (96%) –56.6 s
- 10 epochs:Average loss: 0.0846, Accuracy: 9721/10000 (97%) –1min 57s
- 20 epochs:Average loss: 0.0620, Accuracy: 9829/10000 (98%) –4min 9s

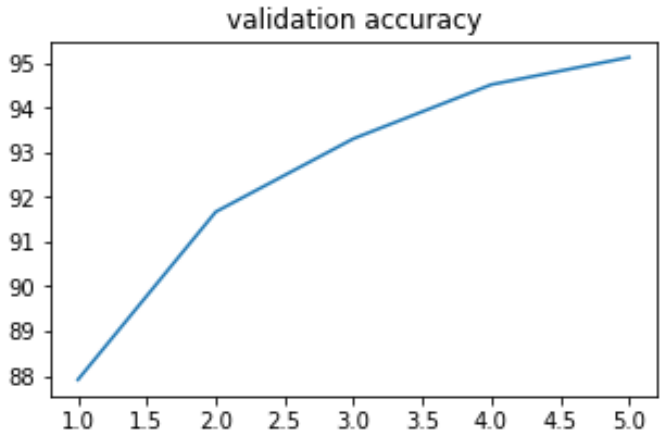
2. 加深网络深度

在原有网络宽度不变情况下 (第二层两端均 50)，增加网络 1 层，其中增加的网络层两端仍为 50 宽度 (在增加的网络层后同样加入 Dropout(0.2) 优化层)。修改的网络结构代码如下：

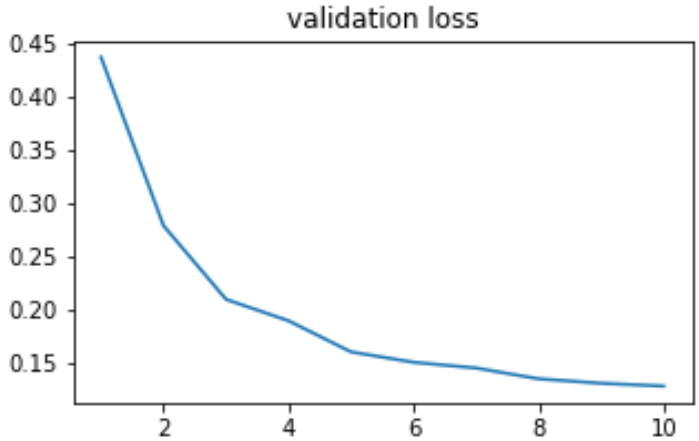
```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 50)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(50, 50)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(50, 50)
        self.fc3_drop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

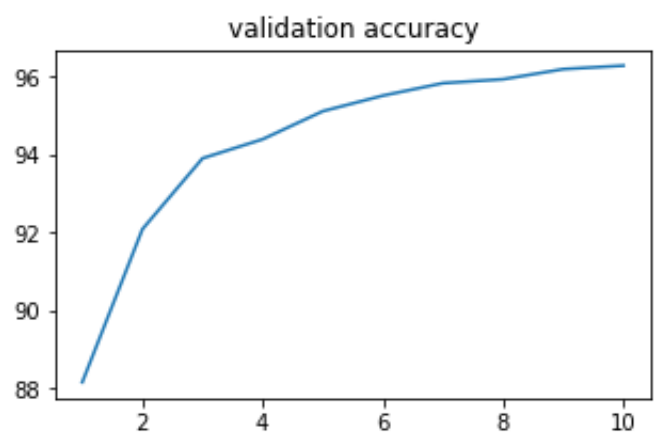
5 epochs，**损失曲线和准确率曲线如下：**



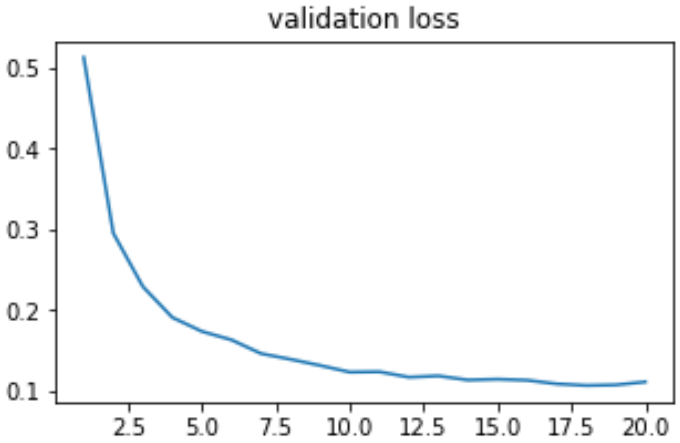


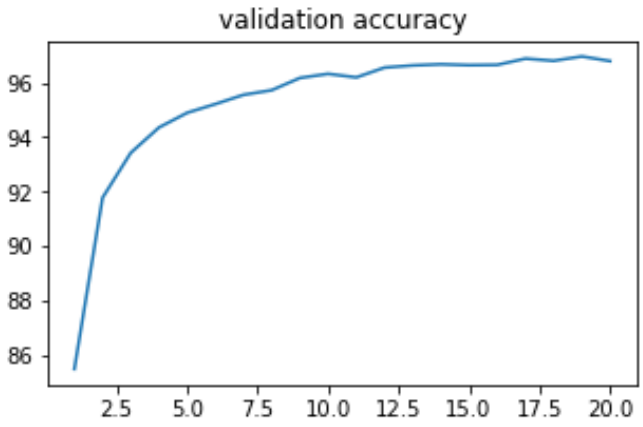
10 epochs，**损失曲线和准确率曲线如下：**





20 epochs，**损失曲线和准确率曲线如下：**





结果总结如下：

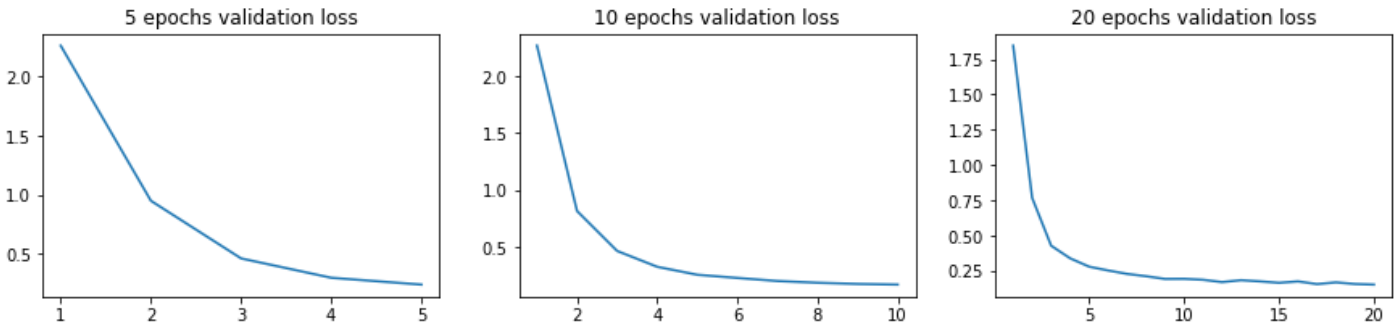
- 5 epochs:Average loss: 0.1668, Accuracy: 9513/10000 (95.1%) –53 s
- 10 epochs:Average loss: 0.1273, Accuracy: 9629/10000 (96.3%) –1min 48s
- 20 epochs:Average loss: 0.1116, Accuracy: 9679/10000 (96.8%) –3min 42s

可以看到，在增加了一层网络后，网络准确率没有增加反而比原始网络准确率低，为了验证是否是因为增加层数过少的情况，这里对网络再进一步加深，即在原始网络基础上增加三层相同的隐藏层。网络结构代码如下:

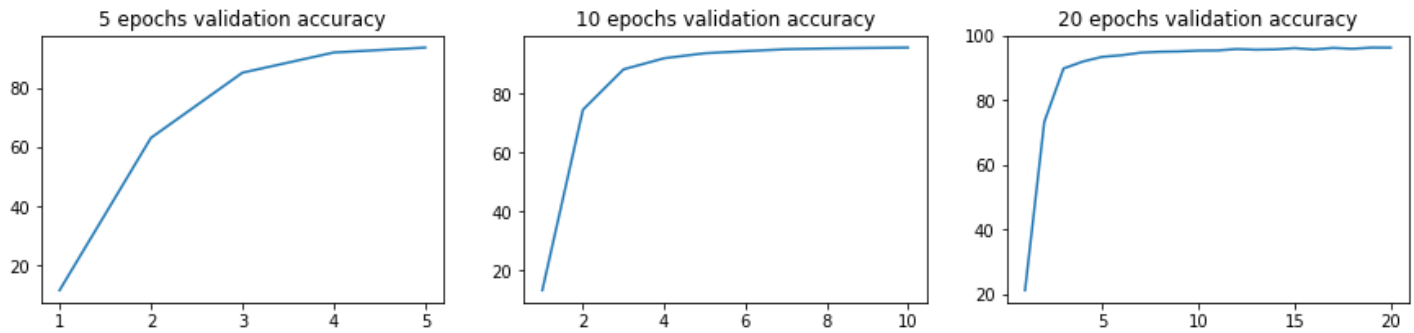
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 50)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(50, 50)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(50, 50)
        self.fc3_drop = nn.Dropout(0.2)
        self.fc4 = nn.Linear(50, 50)
        self.fc4_drop = nn.Dropout(0.2)
        self.fc5 = nn.Linear(50, 50)
        self.fc5_drop = nn.Dropout(0.2)
        self.fc6 = 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)
        x = F.relu(self.fc4(x))
        x = self.fc4_drop(x)
        x = F.relu(self.fc5(x))
        x = self.fc5_drop(x)
        return F.log_softmax(self.fc6(x), dim=1)
```

分别在 5、10、20 epochs 下的 loss 曲线如下：



分别在 5、10、20 epochs 下的 accuracy 曲线如下：



结果总结如下：

- 5 epochs:Average loss: 0.3079, Accuracy: 9294/10000 (92.9%) –52.4 s
- 10 epochs:Average loss: 0.1738, Accuracy: 9578/10000 (95.8%) –1min 47s
- 20 epochs:Average loss: 0.1536, Accuracy: 9624/10000 (96.2%) –3min 46s

（三）使用不同激活函数

pytorch 中提供的激活函数有很多，这里尝试改为两个经典的激活函数 sigmoid 和 Tanh

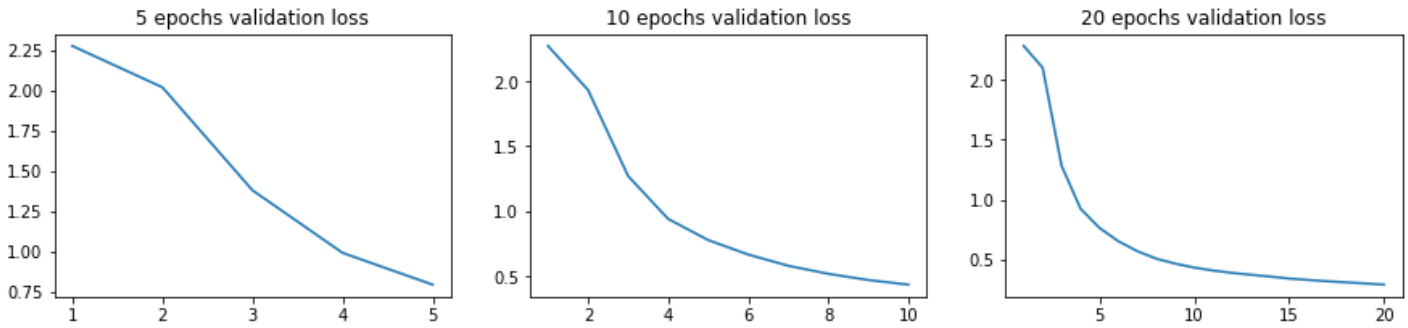
1. sigmoid 激活函数

在原始版本的网络结构中，将 Relu 激活函数改为 sigmoid，其它结构不变，forward 函数代码如下：

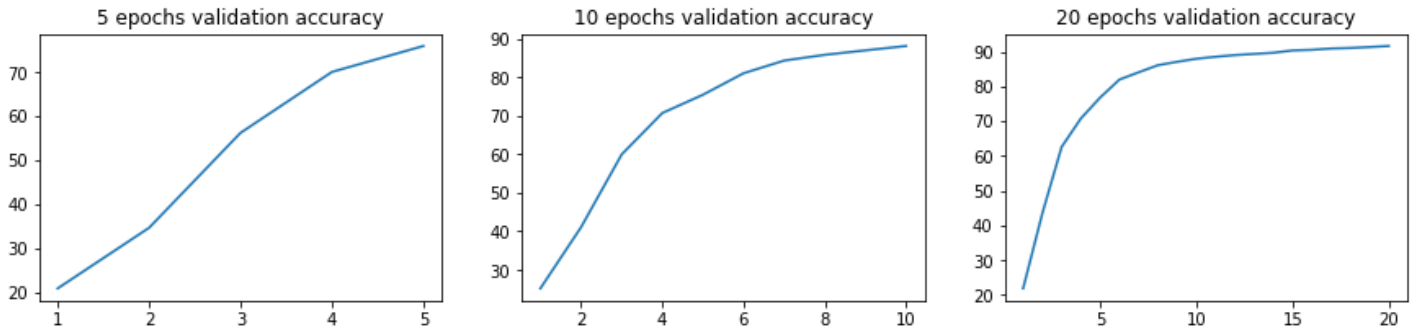
```
def forward ( s e l f , x) :
```

```
    x = x.view(-1, 28 28)
    x = torch.sigmoid(self.fc1(x))
    x = self.fc1_drop(x)
    x = torch.sigmoid(self.fc2(x))
    x = self.fc2_drop(x)
    return F.log_softmax(self.fc3(x),dim=1)
```

在 5、10、20 epochs 下的 loss 曲线如下：



分别在 5、10、20 epochs 下的 accuracy 曲线如下：



结果总结如下：

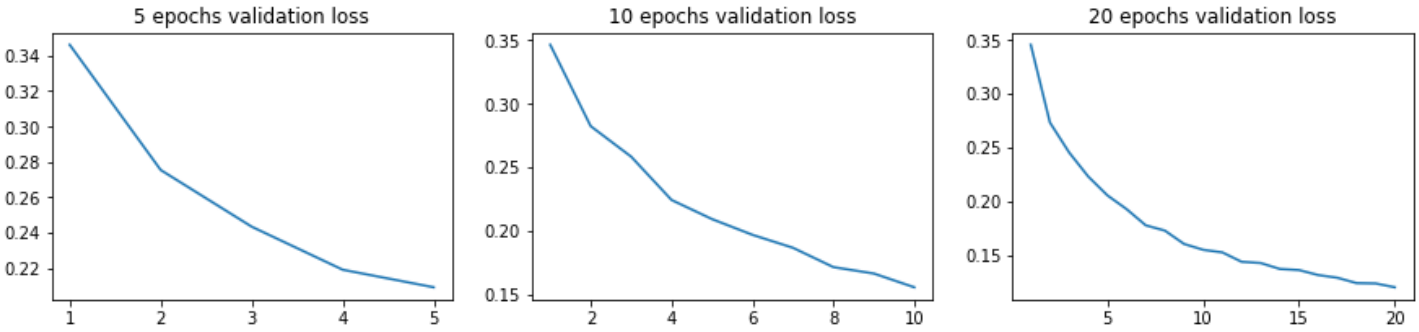
- 5 epochs:Average loss: 0.7940, Accuracy: 7589/10000 (75.9%) - 51.7 s
- 10 epochs:Average loss: 0.4368, Accuracy: 8806/10000 (88.1%) - 1min 34s
- 20 epochs:Average loss: 0.2926, Accuracy: 9159/10000 (91.6%) - 3min 12s

2. Tanh 激活函数

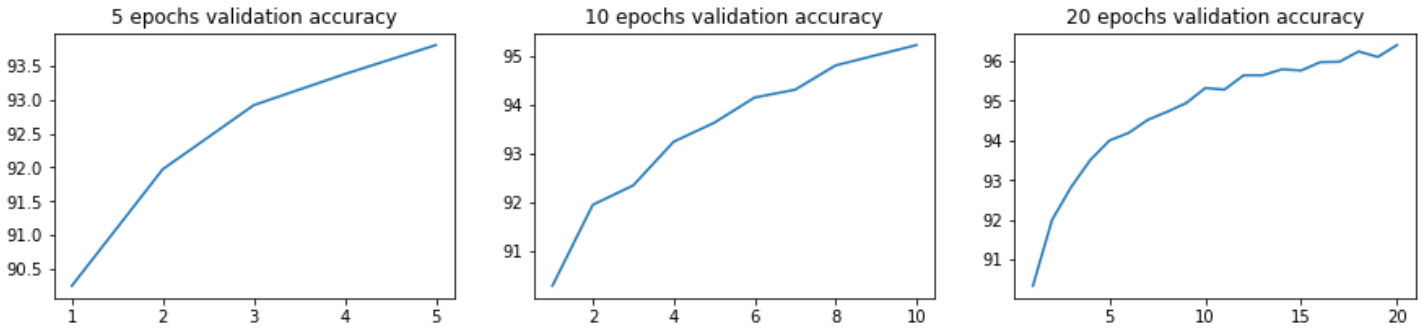
在原始版本的网络结构中，将 Relu 激活函数改为 Tanh，其它结构不变，forward 函数代码如下：

```
def forward ( s e l f , x) :
    x = x.view(-1, 28 28)
    x = torch.tanh(self.fc1(x))
    x = self.fc1_drop(x)
    x = torch.tanh(self.fc2(x))
    x = self.fc2_drop(x)
    return F.log_softmax(self.fc3(x),dim=1)
```

分别在 5、10、20 epochs 下的 loss 曲线如下：



分别在 5、10、20 epochs 下的 accuracy 曲线如下：



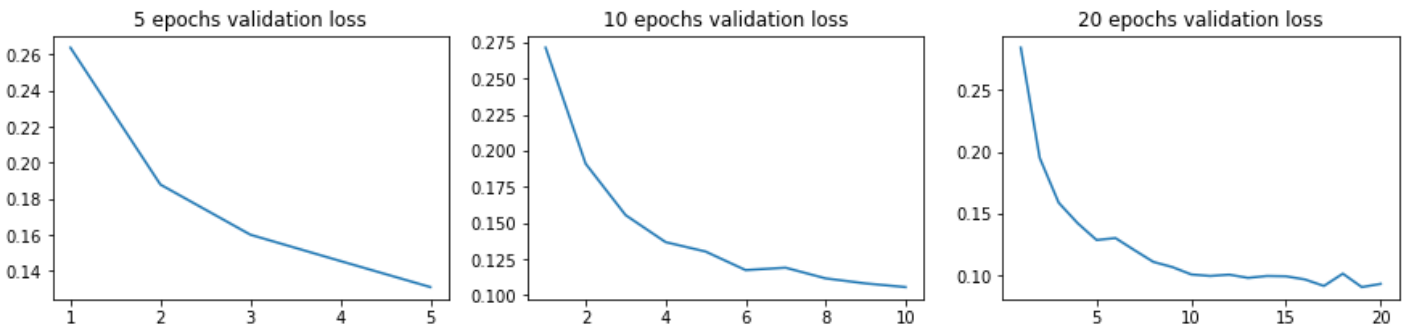
结果总结如下：

- 5 epochs:Average loss: 0.2081, Accuracy: 9381/10000 (93.8%) - 48.9 s
- 10 epochs:Average loss: 0.1556, Accuracy: 9523/10000 (95.2%) - 1min 37s
- 20 epochs:Average loss: 0.1196, Accuracy: 9639/10000 (96.4%) - 3min 18s

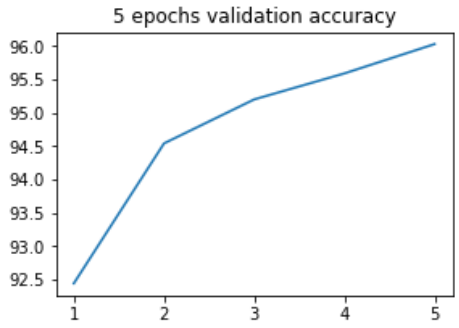
（四） 不同学习率

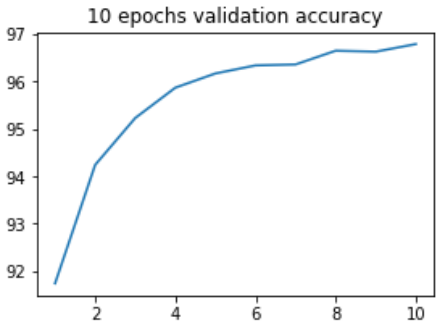
1. 增大学习率

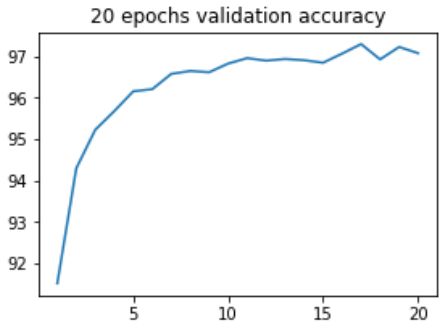
增加学习率由原有的 0.01 增加至 0.025，分别在 5、10、20 epochs 下的 loss 曲线如下：



在 5、10、20 epochs 下的 accuracy 曲线如下：





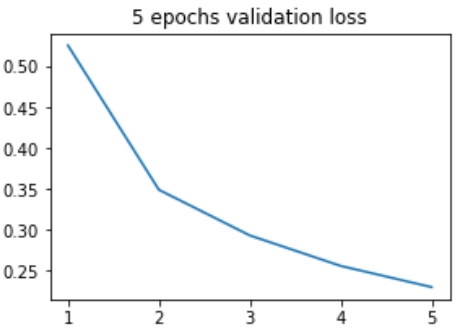


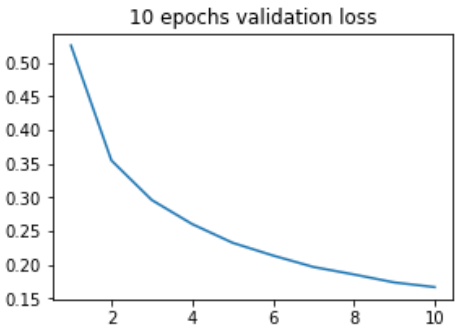
结果总结如下：

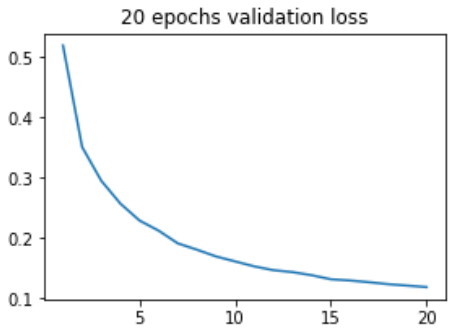
- 5 epochs:Average loss: 0.1314, Accuracy: 9603/10000 (96.0%) –46 s
- 10 epochs:Average loss: 0.1059, Accuracy: 9679/10000 (96.8%) –1min 39s
- 20 epochs:Average loss: 0.0935, Accuracy: 9707/10000 (97.1%) –3min 4s

2. 减小学习率

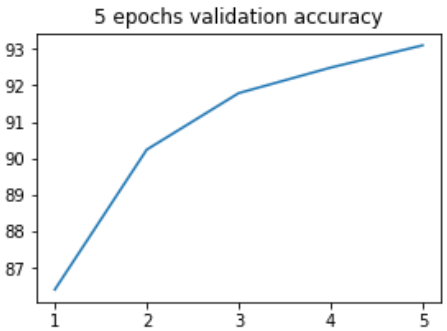
减小学习率由原有的 0.01 减小至 0.005，分别在 5、10、20 epochs 下的 loss 曲线如下：

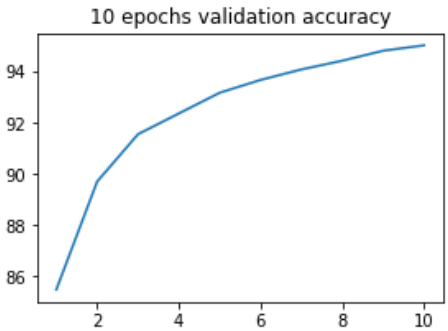


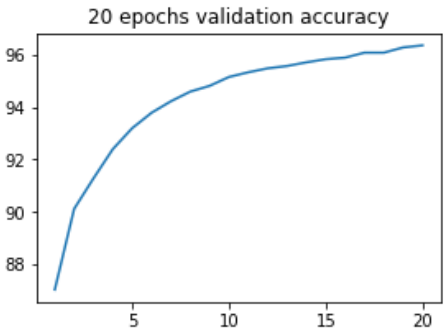




分别在 5、10、20 epochs 下的 accuracy 曲线如下：







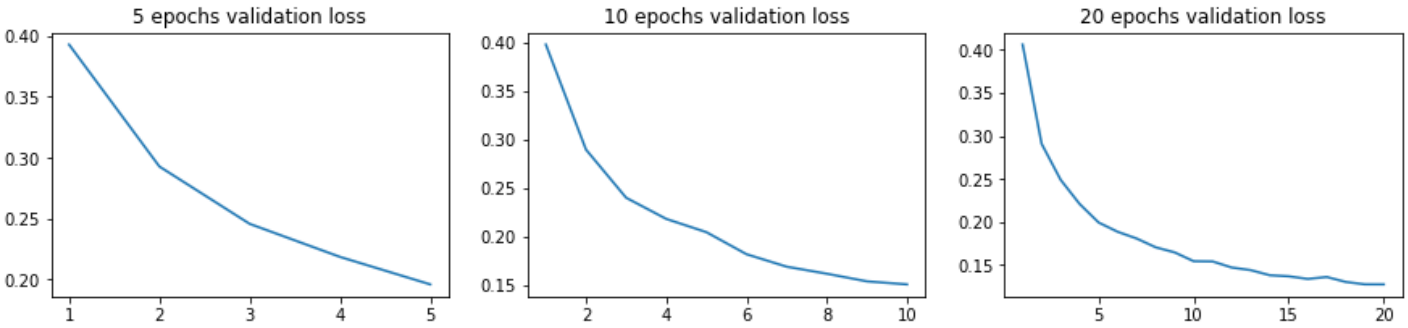
结果总结如下：

- 5 epochs:Average loss: 0.2300, Accuracy: 9310/10000 (93.1%) –47.7 s
- 10 epochs:Average loss: 0.1669, Accuracy: 9502/10000 (95.0%) –1min 35s
- 20 epochs:Average loss: 0.1187, Accuracy: 9637/10000 (96.4%) - 3min 14s

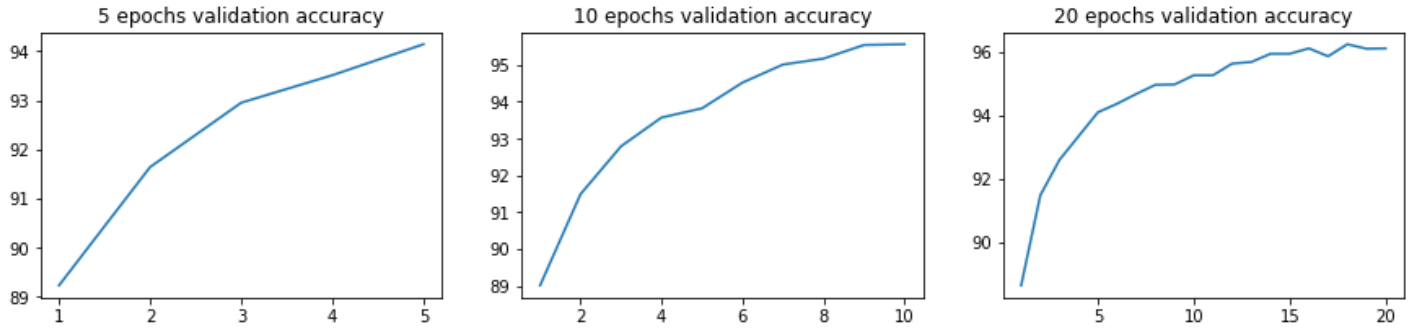
（五） 不同 dropout 率

1. 增大 dropout 率

增加 dropout 率由原有的 0.2 增加至 0.35，分别在 5、10、20 epochs 下的 loss 曲线如下：



分别在 5、10、20 epochs 下的 accuracy 曲线如下：

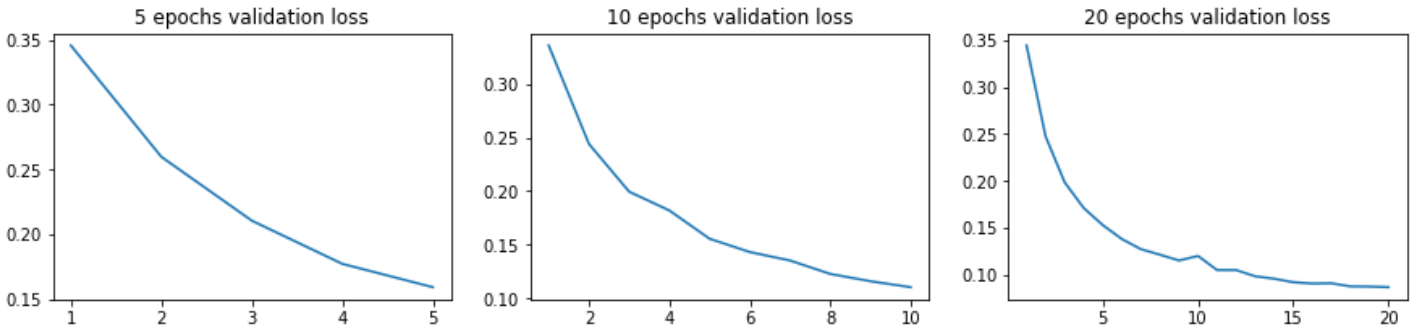


结果总结如下：

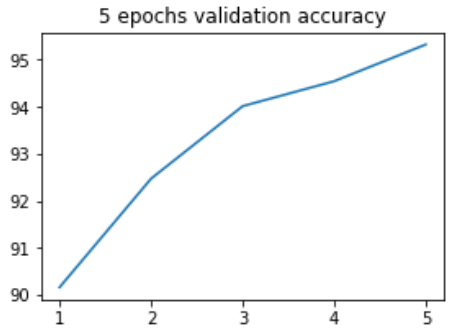
- 5 epochs:Average loss: 0.1980, Accuracy: 9416/10000 (94.2%) –45.4 s
- 10 epochs:Average loss: 0.1509, Accuracy: 9558/10000 (95.6%) –1min 44s
- 20 epochs:Average loss: 0.1274, Accuracy: 961/10000 (96.1%) –3min 17s

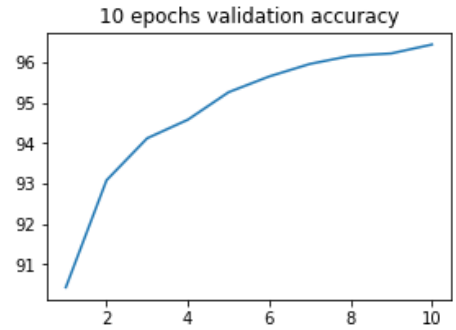
2. 减小 dropout 率

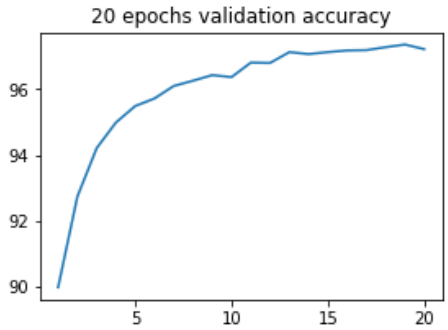
减小 dropout 率由原有的 0.2 减小至 0.05，分别在 5、10、20 epochs 下的 loss 曲线如下：



分别在 5、10、20 epochs 下的 accuracy 曲线如下：







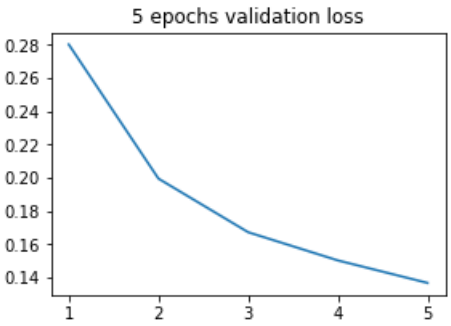
结果总结如下：

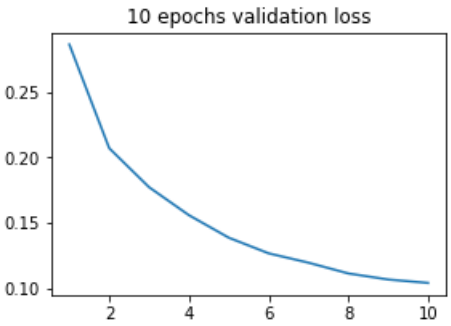
- 5 epochs:Average loss: 0.1595, Accuracy: 953/10000 (95.3%) –44.7 s
- 10 epochs:Average loss: 0.1107, Accuracy: 964/10000 (96.4%) –1min 34s
- 20 epochs:Average loss: 0.0868, Accuracy: 972/10000 (97.2%) –3min 28s

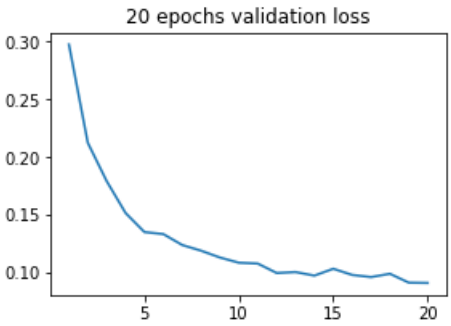
（六） 不同 momentum

1. 增大动量

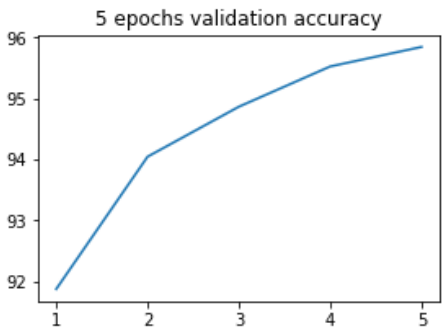
增大 momentum 由原来的 0.5 增大至 0.7。分别在 5、10、20 epochs 下的 loss 曲线如下：

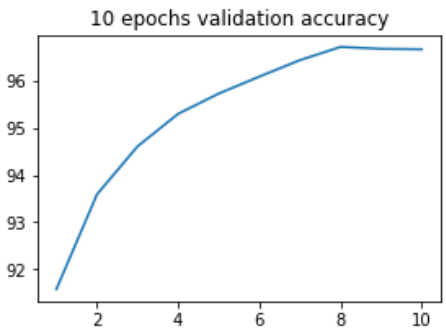


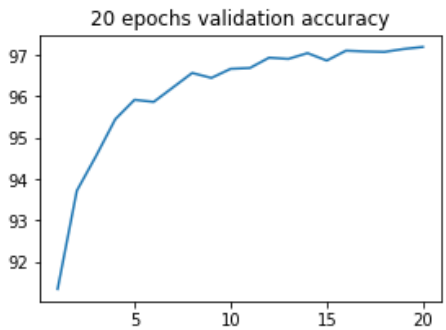




分别在 5、10、20 epochs 下的 accuracy 曲线如下：





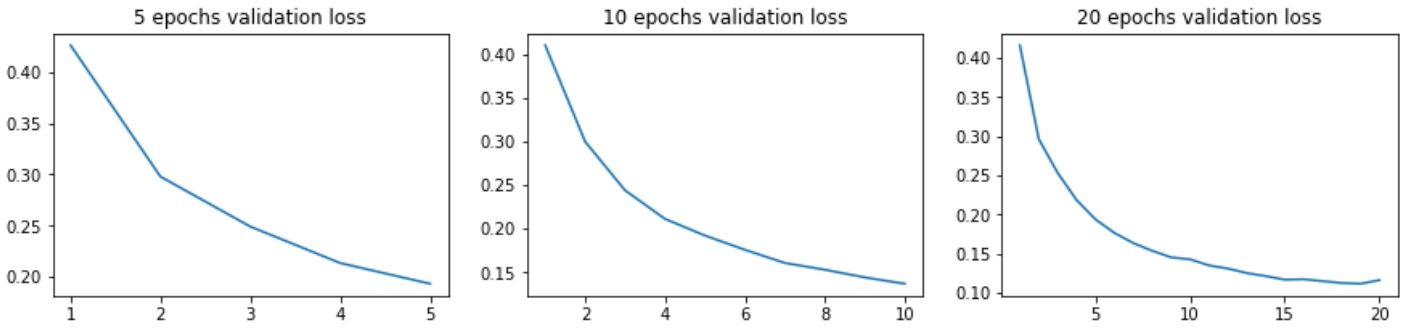


结果总结如下：

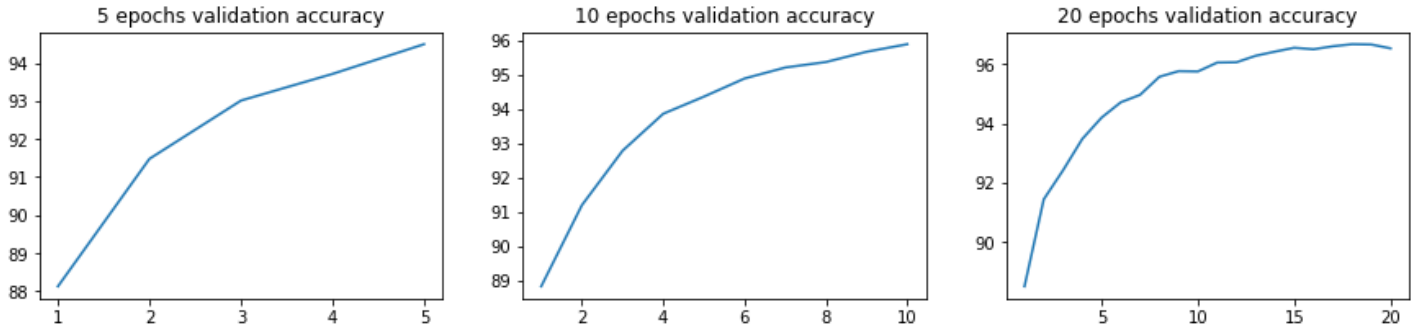
- 5 epochs:Average loss: 0.1366, Accuracy: 9584/10000 (95.8%) - 47.4 s
- 10 epochs:Average loss: 0.1040, Accuracy: 9667/10000 (96.7%) - 1min 37s
- 20 epochs:Average loss: 0.0907, Accuracy: 9720/10000 (97.2%) - 3min 25s

2. 减小动量

减小 momentum 由原来的 0.5 减小至 0.3。分别在 5、10、20 epochs 下的 loss 曲线如下：



分别在 5、10、20 epochs 下的 accuracy 曲线如下：



结果总结如下：

- 5 epochs:Average loss: 0.1926, Accuracy: 9449/10000 (94.5%) - 49.5 s
- 10 epochs:Average loss: 0.1368, Accuracy: 9589/10000 (95.9%) - 1min 43s
- 20 epochs:Average loss: 0.1157, Accuracy: 9654/10000 (96.5%) - 3min 41s

三、 总体改进与分析

1. 网络宽度

增加网络宽度，将隐层的神经元个数从 50 增加至 200，训练后发现准确率从 96.2% 增加至97%，并且当 epochs 增加到 20 时，准确率进一步提升达到了 98%。说明适当增加网络宽度，可以让网络层学习到更多特征，提升网络预测准确率。在训练耗时方面，增加网络宽度同时也会增加训练时间。

2. 网络深度

第一次增加了一层网络深度，准确率从 96.2%(原始网络) 变为 96.3%，变化不大；在尝试了加深网络深度后 (增加三层网络)，发现准确率不增反降，从 96.2%(原始网络) 降 至 95.8%。说明一味地增加网络深度并不能提升预测准确率。在网络能够收敛的前提下，只是简单增加神经网络的深度并不一定可以获得更好的效果，反而有可能出现网络退化的情况，正确率不增反降，正如奥卡姆剃刀原理所述。在训练耗时方面，增加网络深度会稍微增加训练时间。

3. 不同激活函数

sigmiod：从测试的曲线图可以发现，在使用该激活函数时5、10、20 epochs 下的准确率仅有75.9%、88.1%、91.6%，说明使用 sigmoid 激活函数收敛缓慢。除此之外，loss 和accuracy 曲线往后斜率非常小，说明 sigmoid 函数容易出现变化太缓慢，导数接近 0 的梯度消失情况。

Tanh：从实验曲线图可以看到，使用 sigmiod 激活函数时，5、10、20 epochs 下的准确率为93.8%、95.2%、96.4%，相较于 sigmoid 收敛速度快，Tanh 是 sigmoid 的变式，因为 Tanh改善了 sigmoid 的输出，均值为 0，缓解了收敛缓慢的问题。但对于预测准确率来说，相较于原始版的 Relu，几乎没有什么提升。在训练耗时方面，使用不同激活函数基本不会影响训练时间，训练时间非常接近。

4. 学习率

增大学习率：网络训练收敛速度加快；但从 20 epochs 曲线图可以看到，在训练接近尾声时，loss 和 accuracy 曲线上下波动很大。说明学习率过大会导致最终网络的收敛效果不好，反复在最优值附近徘徊。预测准确率略高于原始版本。

减小学习率：网络收敛速度变缓，在训练接近尾声时，loss 和 accuracy 曲线越来越平滑，逐渐收敛。但如果学习率过小，容易导致网络陷入局部极值点影响预测准确率。预测准确率略低于原始版本。在训练耗时方面，不同学习率基本不影响训练时间。

5.dropout 率

第一次增大 dropout 率至 0.35，第二次减少 dropout 率至 0.05，比对两次实验的 loss 变化曲线，可以看到 0.35dropout 率的 loss 下降速率明显慢于 0.05dropout 率。可以发现 dropout 率越大每轮训练舍弃的信息越多，loss 下降越慢，accuracy 提升也越慢。dropout 作为一种正则化方法，缓解了过拟合的情况发生，但过大的 dropout 率拖慢收敛速度，需要不断调试至合适的dropout 率 (最好不超过 0.5)。需要考虑训练 epochs 轮数多少，判断较大的 dropout 率下是否能够充分收敛。

6.momentum 动量

加入 momentum 为了在一定程度上避免陷入局部最优解的情况，momentum 越大就越有可能摆脱局部凹区域。对比增大动量至 0.7 和减少动量至 0.3 的 loss 曲线图可以看到，momentum越大，loss 下降得越快，且每轮的 loss 和 accuracy 波动较大。0.7 动量的准确率略高于原始网络，0.3 动量准确率略低于原始网络。

最终代码：

综合以上实验结果分析可知，我们可以通过加大网络宽度，网络深度不变，使用 Relu 激活函数，增加学习率，适量减小 dropout 率，适量改变并增加 momentum 的改进策略。最终的选取网络结构如下：

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 28*28)
        self.fc1_drop = nn.Dropout(0.15)
        self.fc2 = nn.Linear(28*28, 28*28)
        self.fc2_drop = nn.Dropout(0.15)
        self.fc3 = nn.Linear(28*28, 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)
        return F.log_softmax(self.fc3(x), dim=1)

model = Net().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr = 0.015, momentum = 0.6)
criterion = nn.CrossEntropyLoss()
```