2400010833 闵诗珈

人工智能基础

作业二

part 1 用numpy实现训练MLP网络识别手写数字MNIST数据集

1. 补充代码内容并进行训练:
    - 使用交叉熵损失函数进行训练

```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)> & C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/np_mnist-original.py"
100%|██████████████████████████████| 782/782 [00:02<00:00, 316.47it/s]
epoch: 1, val_loss: 0.1265, val_accuracy: 0.9635
100%|██████████████████████████████| 782/782 [00:02<00:00, 262.61it/s]
epoch: 2, val_loss: 0.1149, val_accuracy: 0.9656
100%|██████████████████████████████| 782/782 [00:03<00:00, 258.78it/s]
epoch: 3, val_loss: 0.1025, val_accuracy: 0.9677
100%|██████████████████████████████| 782/782 [00:02<00:00, 272.28it/s]
epoch: 4, val_loss: 0.0928, val_accuracy: 0.9726
100%|██████████████████████████████| 782/782 [00:02<00:00, 276.25it/s]
epoch: 5, val_loss: 0.0809, val_accuracy: 0.9770
100%|██████████████████████████████| 782/782 [00:02<00:00, 277.17it/s]
epoch: 6, val_loss: 0.0749, val_accuracy: 0.9783
100%|██████████████████████████████| 782/782 [00:02<00:00, 343.68it/s]
epoch: 7, val_loss: 0.0769, val_accuracy: 0.9783
100%|██████████████████████████████| 782/782 [00:03<00:00, 222.86it/s]
epoch: 8, val_loss: 0.0794, val_accuracy: 0.9787
100%|██████████████████████████████| 782/782 [00:04<00:00, 179.54it/s]
epoch: 9, val_loss: 0.0794, val_accuracy: 0.9783
100%|██████████████████████████████| 782/782 [00:03<00:00, 199.11it/s]
epoch: 10, val_loss: 0.0799, val_accuracy: 0.9791
```

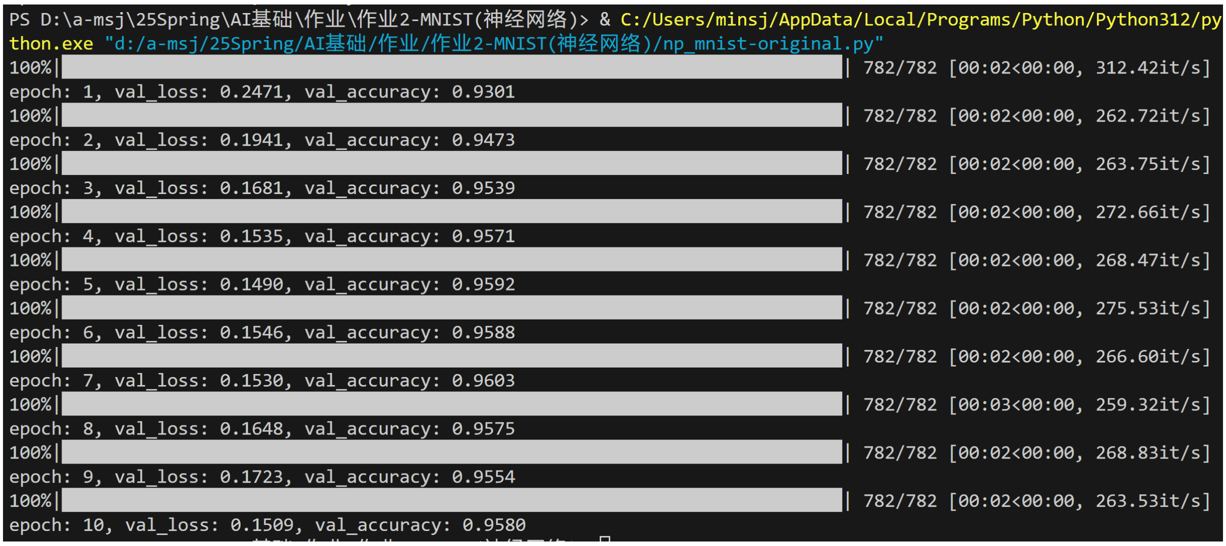
epoch: 10, val_loss: 0.0799, val_accuracy: 0.9791

2. 调整激活函数
    - sigmoid

```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)> & C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/np_mnist-original.py"
100%|██████████████████████████████| 782/782 [00:02<00:00, 268.74it/s]
epoch: 1, val_loss: 0.3046, val_accuracy: 0.9123
100%|██████████████████████████████| 782/782 [00:02<00:00, 300.94it/s]
epoch: 2, val_loss: 0.2792, val_accuracy: 0.9213
100%|██████████████████████████████| 782/782 [00:02<00:00, 309.78it/s]
epoch: 3, val_loss: 0.2651, val_accuracy: 0.9264
100%|██████████████████████████████| 782/782 [00:02<00:00, 299.39it/s]
epoch: 4, val_loss: 0.2569, val_accuracy: 0.9293
100%|██████████████████████████████| 782/782 [00:02<00:00, 292.27it/s]
epoch: 5, val_loss: 0.2526, val_accuracy: 0.9305
100%|██████████████████████████████| 782/782 [00:02<00:00, 266.46it/s]
epoch: 6, val_loss: 0.2503, val_accuracy: 0.9314
100%|██████████████████████████████| 782/782 [00:02<00:00, 277.81it/s]
epoch: 7, val_loss: 0.2488, val_accuracy: 0.9313
100%|██████████████████████████████| 782/782 [00:02<00:00, 262.98it/s]
epoch: 8, val_loss: 0.2473, val_accuracy: 0.9313
100%|██████████████████████████████| 782/782 [00:02<00:00, 278.12it/s]
epoch: 9, val_loss: 0.2470, val_accuracy: 0.9319
100%|██████████████████████████████| 782/782 [00:02<00:00, 277.31it/s]
epoch: 10, val_loss: 0.2490, val_accuracy: 0.9297
```

epoch: 10, val_loss: 0.2490, val_accuracy: 0.9297

- tanh

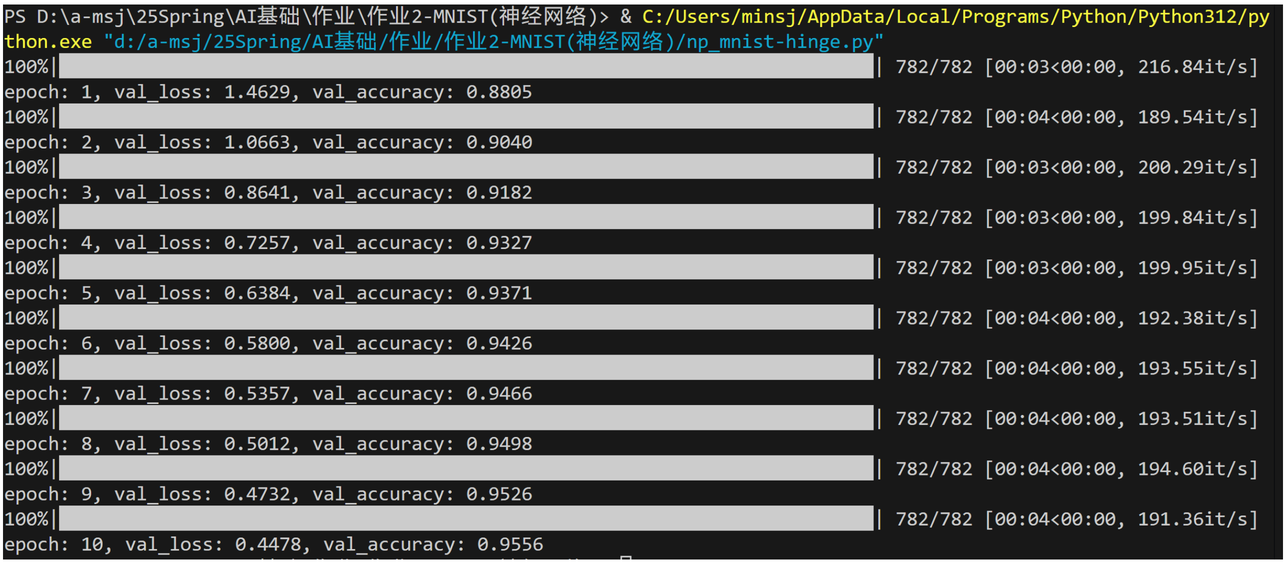


```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)> & C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/np_mnist-original.py"
100%|████████████████████| 782/782 [00:02<00:00, 312.42it/s]
epoch: 1, val_loss: 0.2471, val_accuracy: 0.9301
100%|████████████████████| 782/782 [00:02<00:00, 262.72it/s]
epoch: 2, val_loss: 0.1941, val_accuracy: 0.9473
100%|████████████████████| 782/782 [00:02<00:00, 263.75it/s]
epoch: 3, val_loss: 0.1681, val_accuracy: 0.9539
100%|████████████████████| 782/782 [00:02<00:00, 272.66it/s]
epoch: 4, val_loss: 0.1535, val_accuracy: 0.9571
100%|████████████████████| 782/782 [00:02<00:00, 268.47it/s]
epoch: 5, val_loss: 0.1490, val_accuracy: 0.9592
100%|████████████████████| 782/782 [00:02<00:00, 275.53it/s]
epoch: 6, val_loss: 0.1546, val_accuracy: 0.9588
100%|████████████████████| 782/782 [00:02<00:00, 266.60it/s]
epoch: 7, val_loss: 0.1530, val_accuracy: 0.9603
100%|████████████████████| 782/782 [00:03<00:00, 259.32it/s]
epoch: 8, val_loss: 0.1648, val_accuracy: 0.9575
100%|████████████████████| 782/782 [00:02<00:00, 268.83it/s]
epoch: 9, val_loss: 0.1723, val_accuracy: 0.9554
100%|████████████████████| 782/782 [00:02<00:00, 263.53it/s]
epoch: 10, val_loss: 0.1509, val_accuracy: 0.9580
```

  epoch: 10, val_loss: 0.1509, val_accuracy: 0.9580

  结论：训练效果：RELU>tanh>标准（94%）>sigmoid

3. 更改损失函数：使用hinge损失函数（因为本来用的就是提示的交叉熵损失函数，而均方差一般不用于这里的多分类问题，所以这里尝试更换误差函数的代码是从网上搜索得到的）
   - **Hinge 损失函数**强调正确分类的样本具有足够大的间隔，常用于支持向量机等分类模型，在处理线性可分或近似线性可分的数据时表现较好。



```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)> & C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/np_mnist-hinge.py"
100%|████████████████████| 782/782 [00:03<00:00, 216.84it/s]
epoch: 1, val_loss: 1.4629, val_accuracy: 0.8805
100%|████████████████████| 782/782 [00:04<00:00, 189.54it/s]
epoch: 2, val_loss: 1.0663, val_accuracy: 0.9040
100%|████████████████████| 782/782 [00:03<00:00, 200.29it/s]
epoch: 3, val_loss: 0.8641, val_accuracy: 0.9182
100%|████████████████████| 782/782 [00:03<00:00, 199.84it/s]
epoch: 4, val_loss: 0.7257, val_accuracy: 0.9327
100%|████████████████████| 782/782 [00:03<00:00, 199.95it/s]
epoch: 5, val_loss: 0.6384, val_accuracy: 0.9371
100%|████████████████████| 782/782 [00:04<00:00, 192.38it/s]
epoch: 6, val_loss: 0.5800, val_accuracy: 0.9426
100%|████████████████████| 782/782 [00:04<00:00, 193.55it/s]
epoch: 7, val_loss: 0.5357, val_accuracy: 0.9466
100%|████████████████████| 782/782 [00:04<00:00, 193.51it/s]
epoch: 8, val_loss: 0.5012, val_accuracy: 0.9498
100%|████████████████████| 782/782 [00:04<00:00, 194.60it/s]
epoch: 9, val_loss: 0.4732, val_accuracy: 0.9526
100%|████████████████████| 782/782 [00:04<00:00, 191.36it/s]
epoch: 10, val_loss: 0.4478, val_accuracy: 0.9556
```

epoch: 10, val_loss: 0.4478, val_accuracy: 0.9556

4. 调整网格结构：使用三层的神经网络

```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)> & C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/np_mnist-3层.py"
100%|██████████| 782/782 [00:03<00:00, 222.08it/s]
epoch: 1, val_loss: 0.1400, val_accuracy: 0.9578
100%|██████████| 782/782 [00:03<00:00, 204.71it/s]
epoch: 2, val_loss: 0.1249, val_accuracy: 0.9634
100%|██████████| 782/782 [00:04<00:00, 184.39it/s]
epoch: 3, val_loss: 0.1236, val_accuracy: 0.9660
100%|██████████| 782/782 [00:04<00:00, 193.12it/s]
epoch: 4, val_loss: 0.1160, val_accuracy: 0.9691
100%|██████████| 782/782 [00:03<00:00, 201.51it/s]
epoch: 5, val_loss: 0.1030, val_accuracy: 0.9734
100%|██████████| 782/782 [00:04<00:00, 193.16it/s]
epoch: 6, val_loss: 0.1140, val_accuracy: 0.9727
100%|██████████| 782/782 [00:03<00:00, 200.76it/s]
epoch: 7, val_loss: 0.1259, val_accuracy: 0.9736
100%|██████████| 782/782 [00:04<00:00, 184.64it/s]
epoch: 8, val_loss: 0.1158, val_accuracy: 0.9741
100%|██████████| 782/782 [00:03<00:00, 202.52it/s]
epoch: 9, val_loss: 0.1226, val_accuracy: 0.9742
100%|██████████| 782/782 [00:04<00:00, 176.60it/s]
epoch: 10, val_loss: 0.1434, val_accuracy: 0.9707
```

epoch: 10, val_loss: 0.1434, val_accuracy: 0.9707

结论：训练成本（时间）增加，效果无显著差异。

part 2 使用Pytorch训练MNIST数据集的MLP模型

1. 由于笔记本没有NVIDIA显卡，改为使用cpu进行训练，训练速度较慢；在最后一个全连接层不应该使用relu函数，因为：relu会把所有负值变为 0，这也许会对F.log_softmax的计算造成影响，进而使模型无法正常输出概率分布；将隐藏层神经元数量定义为变量方便修改。
2. 使用SGD优化器

```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)> & C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/torch_mnist_SGD.py"
Epoch 1 Training: 100%|██████████| 782/782 [00:03<00:00, 216.90it/s]
Train set: Epoch 1 - Average loss: 1.8415, Accuracy: 49.98%
Epoch 1 Validation: 100%|██████████| 157/157 [00:00<00:00, 556.46it/s]
Validation set: Epoch 1 - Average loss: 0.8995, Accuracy: 81.33%
Epoch 2 Training: 100%|██████████| 782/782 [00:03<00:00, 219.64it/s]
Train set: Epoch 2 - Average loss: 0.7354, Accuracy: 78.88%
Epoch 2 Validation: 100%|██████████| 157/157 [00:00<00:00, 680.54it/s]
Validation set: Epoch 2 - Average loss: 0.4408, Accuracy: 88.57%
Epoch 3 Training: 100%|██████████| 782/782 [00:03<00:00, 211.95it/s]
Train set: Epoch 3 - Average loss: 0.5206, Accuracy: 84.74%
Epoch 3 Validation: 100%|██████████| 157/157 [00:00<00:00, 655.40it/s]
Validation set: Epoch 3 - Average loss: 0.3541, Accuracy: 90.36%
Epoch 4 Training: 100%|██████████| 782/782 [00:04<00:00, 177.69it/s]
Train set: Epoch 4 - Average loss: 0.4390, Accuracy: 87.16%
Epoch 4 Validation: 100%|██████████| 157/157 [00:00<00:00, 516.95it/s]
Validation set: Epoch 4 - Average loss: 0.3111, Accuracy: 91.15%
Epoch 5 Training: 100%|██████████| 782/782 [00:04<00:00, 169.17it/s]
Train set: Epoch 5 - Average loss: 0.3916, Accuracy: 88.60%
Epoch 5 Validation: 100%|██████████| 157/157 [00:00<00:00, 496.38it/s]
Validation set: Epoch 5 - Average loss: 0.2818, Accuracy: 92.10%
Epoch 6 Training: 100%|██████████| 782/782 [00:04<00:00, 168.33it/s]
Train set: Epoch 6 - Average loss: 0.3568, Accuracy: 89.61%
Epoch 6 Validation: 100%|██████████| 157/157 [00:00<00:00, 506.03it/s]
Validation set: Epoch 6 - Average loss: 0.2621, Accuracy: 92.50%
Epoch 7 Training: 100%|██████████| 782/782 [00:04<00:00, 165.85it/s]
Train set: Epoch 7 - Average loss: 0.3317, Accuracy: 90.34%
Epoch 7 Validation: 100%|██████████| 157/157 [00:00<00:00, 524.01it/s]
Validation set: Epoch 7 - Average loss: 0.2432, Accuracy: 92.88%
Epoch 8 Training: 100%|██████████| 782/782 [00:04<00:00, 158.62it/s]
Train set: Epoch 8 - Average loss: 0.3086, Accuracy: 91.04%
Epoch 8 Validation: 100%|██████████| 157/157 [00:00<00:00, 439.27it/s]
Validation set: Epoch 8 - Average loss: 0.2278, Accuracy: 93.31%
Epoch 9 Training: 100%|██████████| 782/782 [00:05<00:00, 145.64it/s]
Train set: Epoch 9 - Average loss: 0.2843, Accuracy: 91.66%
Epoch 9 Validation: 100%|██████████| 157/157 [00:00<00:00, 436.06it/s]
Validation set: Epoch 9 - Average loss: 0.2096, Accuracy: 94.05%
Epoch 10 Training: 100%|██████████| 782/782 [00:05<00:00, 130.91it/s]
Train set: Epoch 10 - Average loss: 0.2707, Accuracy: 92.05%
Epoch 10 Validation: 100%|██████████| 157/157 [00:00<00:00, 358.74it/s]
Validation set: Epoch 10 - Average loss: 0.1990, Accuracy: 94.23%
```

Validation set: Epoch 10 - Average loss: 0.1990, Accuracy: 94.23%

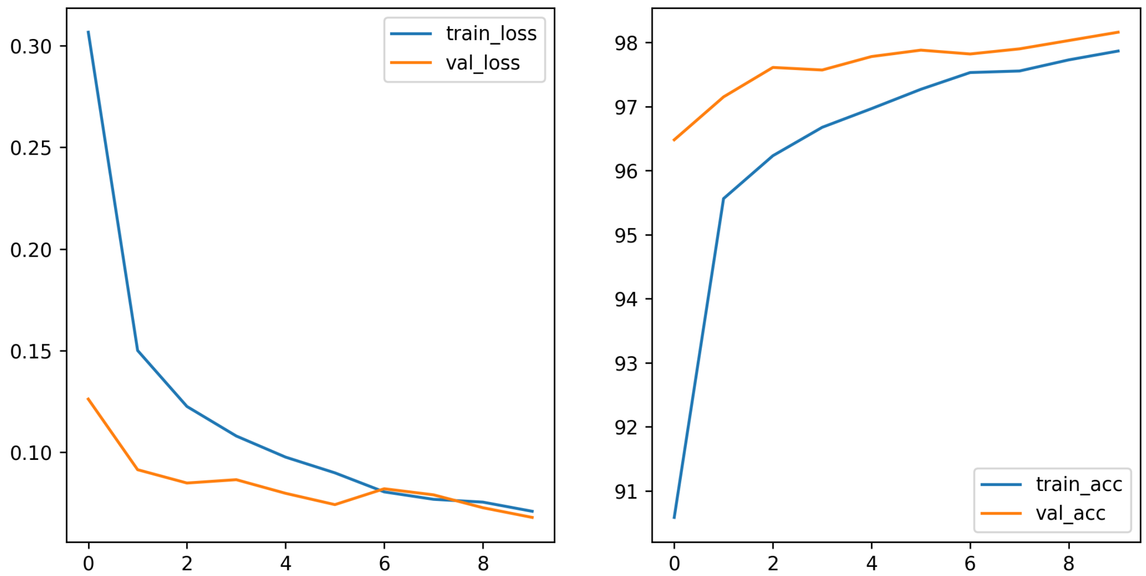
结论：准确率未达标

3. 改为使用Adam优化器

   并且添加Dropout层以减少过拟合

运行结果：

```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)>                                                              >
& C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/torch_mnist_SGD.py"
Epoch 1 Training: 100%|██████████| 782/782 [00:04<00:00, 165.05it/s]
Train set: Epoch 1 - Average loss: 0.3066, Accuracy: 90.58%
Validation set: Average loss: 0.1261, Accuracy: 9648/10000 (96.48%)
Epoch 2 Training: 100%|██████████| 782/782 [00:05<00:00, 130.59it/s]
Train set: Epoch 2 - Average loss: 0.1501, Accuracy: 95.56%
Validation set: Average loss: 0.0914, Accuracy: 9715/10000 (97.15%)
Epoch 3 Training: 100%|██████████| 782/782 [00:06<00:00, 111.84it/s]
Train set: Epoch 3 - Average loss: 0.1225, Accuracy: 96.23%
Validation set: Average loss: 0.0849, Accuracy: 9761/10000 (97.61%)
Epoch 4 Training: 100%|██████████| 782/782 [00:07<00:00, 103.03it/s]
Train set: Epoch 4 - Average loss: 0.1080, Accuracy: 96.67%
Validation set: Average loss: 0.0865, Accuracy: 9757/10000 (97.57%)
Epoch 5 Training: 100%|██████████| 782/782 [00:07<00:00, 102.12it/s]
Train set: Epoch 5 - Average loss: 0.0976, Accuracy: 96.96%
Validation set: Average loss: 0.0798, Accuracy: 9778/10000 (97.78%)
Epoch 6 Training: 100%|██████████| 782/782 [00:07<00:00, 101.40it/s]
Train set: Epoch 6 - Average loss: 0.0899, Accuracy: 97.27%
Validation set: Average loss: 0.0742, Accuracy: 9788/10000 (97.88%)
Epoch 7 Training: 100%|██████████| 782/782 [00:07<00:00, 101.43it/s]
Train set: Epoch 7 - Average loss: 0.0805, Accuracy: 97.53%
Validation set: Average loss: 0.0820, Accuracy: 9782/10000 (97.82%)
Epoch 8 Training: 100%|██████████| 782/782 [00:07<00:00, 101.05it/s]
Train set: Epoch 8 - Average loss: 0.0768, Accuracy: 97.56%
Validation set: Average loss: 0.0790, Accuracy: 9790/10000 (97.90%)
Epoch 9 Training: 100%|██████████| 782/782 [00:07<00:00, 99.36it/s]
Train set: Epoch 9 - Average loss: 0.0755, Accuracy: 97.73%
Validation set: Average loss: 0.0727, Accuracy: 9803/10000 (98.03%)
Epoch 10 Training: 100%|██████████| 782/782 [00:08<00:00, 97.65it/s]
Train set: Epoch 10 - Average loss: 0.0709, Accuracy: 97.86%
Validation set: Average loss: 0.0679, Accuracy: 9816/10000 (98.16%)
```

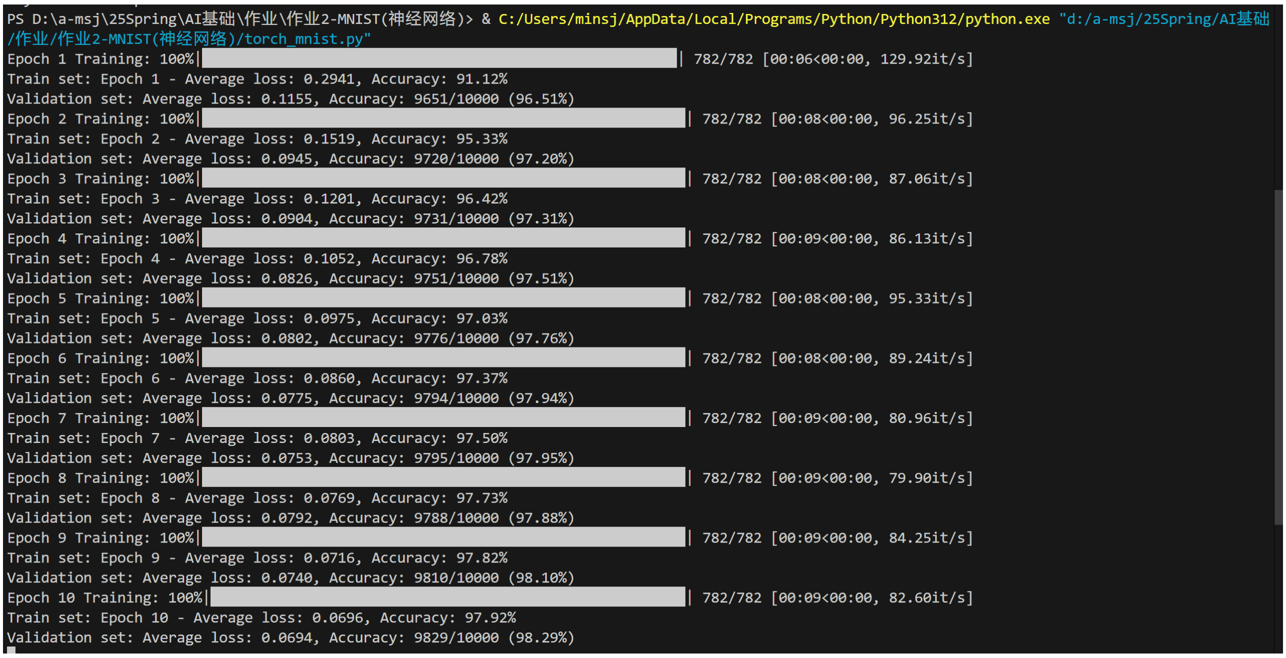


Validation set: Average loss: 0.0679, Accuracy: 9816/10000 (98.16%)

结论：同时使用以上两种优化方法可以显著提高准确性

4. 增加全连接层和层内神经元数量，修改网络结构

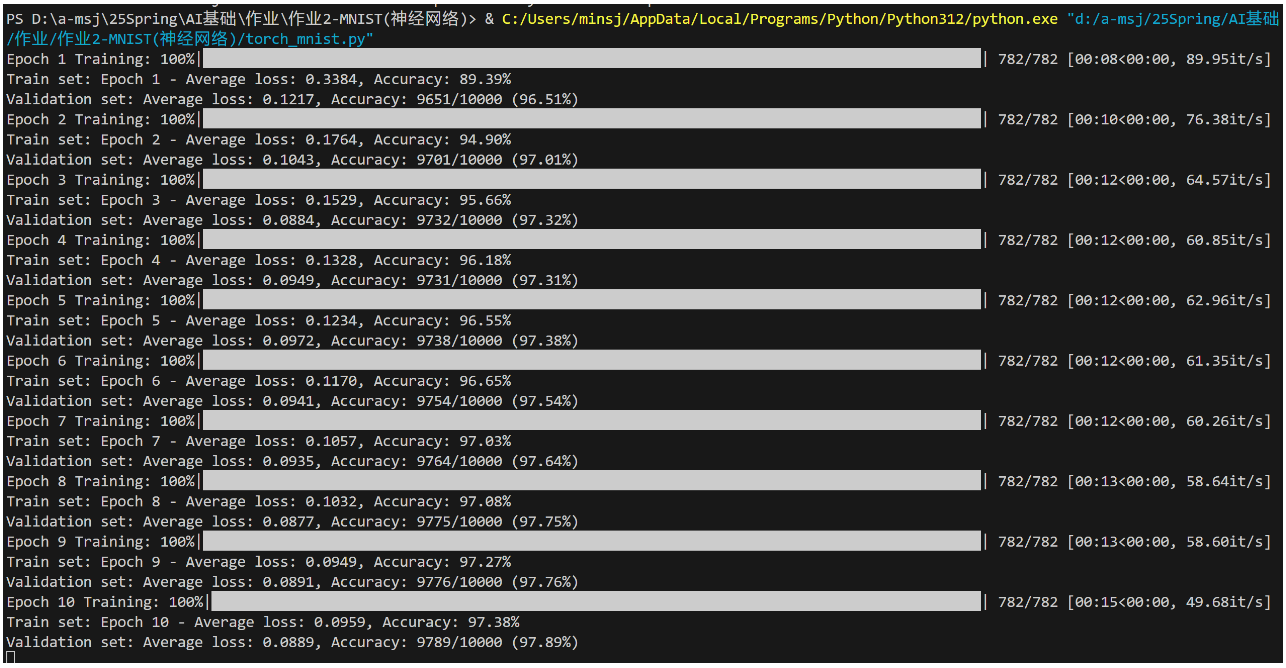
   - 只更改神经元数量



```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)> & C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/torch_mnist.py"
Epoch 1 Training: 100%|██████████| 782/782 [00:06<00:00, 129.92it/s]
Train set: Epoch 1 - Average loss: 0.2941, Accuracy: 91.12%
Validation set: Average loss: 0.1155, Accuracy: 9651/10000 (96.51%)
Epoch 2 Training: 100%|██████████| 782/782 [00:08<00:00, 96.25it/s]
Train set: Epoch 2 - Average loss: 0.1519, Accuracy: 95.33%
Validation set: Average loss: 0.0945, Accuracy: 9720/10000 (97.20%)
Epoch 3 Training: 100%|██████████| 782/782 [00:08<00:00, 87.06it/s]
Train set: Epoch 3 - Average loss: 0.1201, Accuracy: 96.42%
Validation set: Average loss: 0.0904, Accuracy: 9731/10000 (97.31%)
Epoch 4 Training: 100%|██████████| 782/782 [00:09<00:00, 86.13it/s]
Train set: Epoch 4 - Average loss: 0.1052, Accuracy: 96.78%
Validation set: Average loss: 0.0826, Accuracy: 9751/10000 (97.51%)
Epoch 5 Training: 100%|██████████| 782/782 [00:08<00:00, 95.33it/s]
Train set: Epoch 5 - Average loss: 0.0975, Accuracy: 97.03%
Validation set: Average loss: 0.0802, Accuracy: 9776/10000 (97.76%)
Epoch 6 Training: 100%|██████████| 782/782 [00:08<00:00, 89.24it/s]
Train set: Epoch 6 - Average loss: 0.0860, Accuracy: 97.37%
Validation set: Average loss: 0.0775, Accuracy: 9794/10000 (97.94%)
Epoch 7 Training: 100%|██████████| 782/782 [00:09<00:00, 80.96it/s]
Train set: Epoch 7 - Average loss: 0.0803, Accuracy: 97.50%
Validation set: Average loss: 0.0753, Accuracy: 9795/10000 (97.95%)
Epoch 8 Training: 100%|██████████| 782/782 [00:09<00:00, 79.90it/s]
Train set: Epoch 8 - Average loss: 0.0769, Accuracy: 97.73%
Validation set: Average loss: 0.0792, Accuracy: 9788/10000 (97.88%)
Epoch 9 Training: 100%|██████████| 782/782 [00:09<00:00, 84.25it/s]
Train set: Epoch 9 - Average loss: 0.0716, Accuracy: 97.82%
Validation set: Average loss: 0.0740, Accuracy: 9810/10000 (98.10%)
Epoch 10 Training: 100%|██████████| 782/782 [00:09<00:00, 82.60it/s]
Train set: Epoch 10 - Average loss: 0.0696, Accuracy: 97.92%
Validation set: Average loss: 0.0694, Accuracy: 9829/10000 (98.29%)
```

Validation set: Average loss: 0.0694, Accuracy: 9829/10000 (98.29%)

- 增加层数



```
PS D:\a-msj\25Spring\AI基础\作业\作业2-MNIST(神经网络)> & C:/Users/minsj/AppData/Local/Programs/Python/Python312/python.exe "d:/a-msj/25Spring/AI基础/作业/作业2-MNIST(神经网络)/torch_mnist.py"
Epoch 1 Training: 100%|██████████| 782/782 [00:08<00:00, 89.95it/s]
Train set: Epoch 1 - Average loss: 0.3384, Accuracy: 89.39%
Validation set: Average loss: 0.1217, Accuracy: 9651/10000 (96.51%)
Epoch 2 Training: 100%|██████████| 782/782 [00:10<00:00, 76.38it/s]
Train set: Epoch 2 - Average loss: 0.1764, Accuracy: 94.90%
Validation set: Average loss: 0.1043, Accuracy: 9701/10000 (97.01%)
Epoch 3 Training: 100%|██████████| 782/782 [00:12<00:00, 64.57it/s]
Train set: Epoch 3 - Average loss: 0.1529, Accuracy: 95.66%
Validation set: Average loss: 0.0884, Accuracy: 9732/10000 (97.32%)
Epoch 4 Training: 100%|██████████| 782/782 [00:12<00:00, 60.85it/s]
Train set: Epoch 4 - Average loss: 0.1328, Accuracy: 96.18%
Validation set: Average loss: 0.0949, Accuracy: 9731/10000 (97.31%)
Epoch 5 Training: 100%|██████████| 782/782 [00:12<00:00, 62.96it/s]
Train set: Epoch 5 - Average loss: 0.1234, Accuracy: 96.55%
Validation set: Average loss: 0.0972, Accuracy: 9738/10000 (97.38%)
Epoch 6 Training: 100%|██████████| 782/782 [00:12<00:00, 61.35it/s]
Train set: Epoch 6 - Average loss: 0.1170, Accuracy: 96.65%
Validation set: Average loss: 0.0941, Accuracy: 9754/10000 (97.54%)
Epoch 7 Training: 100%|██████████| 782/782 [00:12<00:00, 60.26it/s]
Train set: Epoch 7 - Average loss: 0.1057, Accuracy: 97.03%
Validation set: Average loss: 0.0935, Accuracy: 9764/10000 (97.64%)
Epoch 8 Training: 100%|██████████| 782/782 [00:13<00:00, 58.64it/s]
Train set: Epoch 8 - Average loss: 0.1032, Accuracy: 97.08%
Validation set: Average loss: 0.0877, Accuracy: 9775/10000 (97.75%)
Epoch 9 Training: 100%|██████████| 782/782 [00:13<00:00, 58.60it/s]
Train set: Epoch 9 - Average loss: 0.0949, Accuracy: 97.27%
Validation set: Average loss: 0.0891, Accuracy: 9776/10000 (97.76%)
Epoch 10 Training: 100%|██████████| 782/782 [00:15<00:00, 49.68it/s]
Train set: Epoch 10 - Average loss: 0.0959, Accuracy: 97.38%
Validation set: Average loss: 0.0889, Accuracy: 9789/10000 (97.89%)
```

Validation set: Average loss: 0.0889, Accuracy: 9789/10000 (97.89%)

结论：增加隐藏层数减慢训练速度，对准确度无显著影响。神经网络并非越复杂越好。

代码

part 1

```
# -*- coding: utf-8 -*-
"""
@ author: 闵诗珈
"""

# 作业内容：更改loss函数、网络结构、激活函数，完成训练MLP网络识别手写数字MNIST数据集
```

```python
import numpy as np
from tqdm import tqdm

#import os
#print(os.getcwd())

# 加载数据集,numpy格式
X_train = np.load('./mnist/X_train.npy')  # (60000, 784), 数值在0.0~1.0之间
y_train = np.load('./mnist/y_train.npy')  # (60000, )
y_train = np.eye(10)[y_train]  # (60000, 10), one-hot编码

X_val = np.load('./mnist/X_val.npy')  # (10000, 784), 数值在0.0~1.0之间
y_val = np.load('./mnist/y_val.npy')  # (10000,)
y_val = np.eye(10)[y_val]  # (10000, 10), one-hot编码

X_test = np.load('./mnist/X_test.npy')  # (10000, 784), 数值在0.0~1.0之间
y_test = np.load('./mnist/y_test.npy')  # (10000,)
y_test = np.eye(10)[y_test] # (10000, 10), one-hot编码

# 定义激活函数
def relu(x):
    '''
    relu函数, i.e.f(x)=max(0,x)
    '''
    return np.maximum(0, x)

def relu_prime(x):
    '''
    relu函数的导数
    '''
    # return 1 if x>0 else 0
    # 使用np包如下:
    return np.where(x > 0, 1, 0)

# sigmoid函数
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# sigmoid函数的导数
def sigmoid_prime(x):
    sigmoid(x)
    return sigmoid(s) * (1 - sigmoid(s))

# tanh函数
def tanh(x):
    return np.tanh(x)

# tanh函数的导数
def tanh_prime(x):
    return 1 - np.tanh(x) ** 2

#输出层激活函数
def softmax(x):
    '''
    softmax函数, 防止除0
    '''
```

```
    x_stab = np.exp(x - np.max(x, axis=1, keepdims=True)) # 减去最大值，避免数据溢出
    return x_stab / np.sum(x_stab, axis=1, keepdims=True)

def softmax_prime(x):
    '''
    softmax函数的导数
    '''
    return softmax(x) * (1.0-softmax(x))

# 定义损失函数 交叉熵
def loss_fn(y_true, y_pred):
    '''
    y_true: (batch_size, num_classes), one-hot编码
    y_pred: (batch_size, num_classes), softmax输出, 防0
    '''
    return -np.sum(y_true * np.log(y_pred + 1e-8), axis=-1)

def loss_fn_prime(y_true, y_pred):
    '''
    y_true: (batch_size, num_classes), one-hot编码
    y_pred: (batch_size, num_classes), softmax输出
    '''
    return y_pred - y_true

# 定义Hinge损失函数
'''
def loss_fn(y_true, y_pred):

    y_true: (batch_size, num_classes), one-hot编码
    y_pred: (batch_size, num_classes), softmax输出, 防0

    margin = 1.0
    correct_class_scores = np.sum(y_true * y_pred, axis=1, keepdims=True)
    margins = np.maximum(0, y_pred - correct_class_scores + margin)
    margins[y_true.astype(bool)] = 0
    return np.mean(np.sum(margins, axis=1))

def loss_fn_prime(y_true, y_pred):

    y_true: (batch_size, num_classes), one-hot编码
    y_pred: (batch_size, num_classes), softmax输出

    margin = 1.0
    correct_class_scores = np.sum(y_true * y_pred, axis=1, keepdims=True)
    margins = np.maximum(0, y_pred - correct_class_scores + margin)
    binary_margins = (margins > 0).astype(float)
    binary_margins[y_true.astype(bool)] = -np.sum(binary_margins, axis=1)
    return binary_margins / y_true.shape[0]
    '''


# 定义权重初始化函数
def init_weights(shape):
    return np.random.normal(loc=0.0, scale=np.sqrt(2.0 / shape[0]), size=shape)

# 定义网络结构
```

```python
class Network(object):
    '''
    MNIST数据集分类网络
    '''
    def __init__(self, input_size, hidden_size, output_size, lr=0.01):
        '''
        初始化网络结构
        两层全连接神经网络
        input_size=784, hidden_size=256, output_size=10, lr=0.01
        '''
        self.W1 = init_weights((input_size, hidden_size)) # 输入层到隐藏层的权重矩阵
        self.b1 = np.zeros(hidden_size)  # 输入层到隐藏层的偏置
        self.W2 = init_weights((hidden_size, output_size)) # 隐藏层到输出层的权重矩阵
        self.b2 = np.zeros(output_size)  # 隐藏层到输出层的偏置
        self.lr = lr  # 学习率

    def forward(self, x):
        '''
        前向传播
        '''
        self.z1 = np.dot(x, self.W1) + self.b1
        self.a1 = relu(self.z1)
        self.z2 = np.dot(self.a1, self.W2) + self.b2
        self.a2 = softmax(self.z2)
        return self.a2

    def step(self, x_batch, y_batch):
        '''
        一步训练
        '''

        # 前向传播
        y_pred = self.forward(x_batch)

        # 计算损失和准确率
        loss = np.mean(loss_fn(y_batch, y_pred))
        accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_batch, axis=1))

        # 反向传播
        dz2 = loss_fn_prime(y_batch, y_pred)
        dW2 = np.dot(self.a1.T, dz2)
        db2 = np.sum(dz2, axis=0)

        dz1 = np.dot(dz2, self.W2.T) * relu_prime(self.a1)
        dW1 = np.dot(x_batch.T, dz1)
        db1 = np.sum(dz1, axis=0)

        # 更新权重
        self.W2 -= self.lr * dW2
        self.b2 -= self.lr * db2
        self.W1 -= self.lr * dW1
        self.b1 -= self.lr * db1

        return loss, accuracy

    def predict(self, X):
```

```python
        return np.argmax(self.forward(X), axis=1)

if __name__ == '__main__':
    # 训练网络
    net = Network(input_size=784, hidden_size=256, output_size=10, lr=0.01)
    for epoch in range(10):
        losses = []
        accuracies = []

        p_bar = tqdm(range(0, len(X_train), 64))
        for i in p_bar:
            x_batch = X_train[i:i + 64]
            y_batch = y_train[i:i + 64]
            loss, accuracy = net.step(x_batch, y_batch)

            losses.append(loss)
            accuracies.append(accuracy)

        # 验证网络
        y_pred=net.forward(X_val)
        val_loss=np.mean(loss_fn(y_val, y_pred))
        val_accurary=np.mean(np.argmax(y_pred, axis=-1) == np.argmax(y_val, axis=-1))
        print(f"epoch: {epoch + 1}, val_loss: {val_loss:.4f}, val_accuracy:
{val_accurary:.4f}")
```

part 2

```python
# 第一课作业
# 使用Pytorch训练MNIST数据集的MLP模型
# 1. 运行、阅读并理解mnist_mlp_template.py,修改网络结构和参数，增加隐藏层，观察训练效果
# 2. 使用Adam等不同优化器，添加Dropout层，观察训练效果
# 要求：10个epoch后测试集准确率达到97%以上

# 导入相关的包
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from tqdm import tqdm
import numpy as np
from matplotlib import pyplot as plt

# 隐藏层神经元数量
hide = 1000

# 加载数据集,numpy格式
X_train = np.load('./mnist/X_train.npy')
y_train = np.load('./mnist/y_train.npy')
X_val = np.load('./mnist/X_val.npy')
y_val = np.load('./mnist/y_val.npy')
X_test = np.load('./mnist/X_test.npy')
y_test = np.load('./mnist/y_test.npy')

# 定义MNIST数据集类
class MNISTDataset(Dataset):  # 继承Dataset类
```

```python
    def __init__(self, data=X_train, label=y_train):
        '''
        Args:
            data: numpy array, shape=(N, 784)
            label: numpy array, shape=(N, 10)
        '''
        self.data = data
        self.label = label

    def __getitem__(self, index):
        '''
        根据索引获取数据,返回数据和标签,一个tuple
        '''
        data = self.data[index].astype('float32')  # 转换数据类型, 神经网络一般使用float32作为输入的数据类型
        label = self.label[index].astype('int64')  # 转换数据类型, 分类任务神经网络一般使用int64作为标签的数据类型
        return data, label

    def __len__(self):
        '''
        返回数据集的样本数量
        '''
        return len(self.data)

# 定义模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, hide)
        self.dropout = nn.Dropout()
        self.fc2 = nn.Linear(hide, hide)
        self.fc3 = nn.Linear(hide, hide)
        self.fc4 = nn.Linear(hide,10)


    def forward(self, x):
        x = x.view(-1, 784)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = F.relu(self.fc3(x))
        x = self.dropout(x)
        x = self.fc4(x) # 此处不应使用relu, 正确率会爆炸。原因: relu会把所有负值变为 0, 这也许会对F.log_softmax的计算造成影响, 进而使模型无法正常输出概率分布

        return F.log_softmax(x, dim=1)

# 实例化模型
model = Net()
model.to(device='cpu')
# 定义损失函数
criterion = nn.CrossEntropyLoss()
```

```python
# 定义优化器
optimizer = optim.Adam(model.parameters(), lr=0.001)
# optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.8)

# 定义数据加载器
train_loader = DataLoader(MNISTDataset(X_train, y_train),
                          batch_size=64, shuffle=True)
val_loader = DataLoader(MNISTDataset(X_val, y_val),
                        batch_size=64, shuffle=True)
test_loader = DataLoader(MNISTDataset(X_test, y_test),
                         batch_size=64, shuffle=True)

# 定义训练参数
EPOCHS = 10

history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}
# 训练模型
for epoch in range(EPOCHS):
    # 训练模式
    model.train()

    loss_train = []
    acc_train = []
    correct_train = 0


    for batch_idx, (data, target) in enumerate(tqdm(train_loader, desc=f'Epoch {epoch + 1}
Training')):
        data, target = data.to(device='cpu'), target.to(device='cpu')
        # 梯度清零
        optimizer.zero_grad()
        # 前向计算
        output = model(data)
        # 计算损失
        loss = criterion(output, target)
        # 反向传播
        loss.backward()
        # 参数更新
        optimizer.step()
        # 打印训练信息
        '''if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))'''

        loss_train.append(loss.item())
        pred = output.max(1, keepdim=True)[1]  # 获得最大对数概率的索引

        correct = pred.eq(target.view_as(pred)).sum().item()
        correct_train += correct
        acc_train.append(100. * correct / len(data))

    epoch_train_loss = np.mean(loss_train)
    epoch_train_acc = np.mean(acc_train)
    history['train_loss'].append(epoch_train_loss)
    history['train_acc'].append(epoch_train_acc)
```

```
    print(f'Train set: Epoch {epoch + 1} - Average loss: {epoch_train_loss:.4f}, Accuracy: {100. * correct_train / len(train_loader.dataset):.2f}%')

    # 测试模式
    model.eval()
    val_loss = []
    correct = 0
    with torch.no_grad():
        for data, target in val_loader:
            data, target = data.to(device='cpu'), target.to(device='cpu')

            output = model(data)
            val_loss.append(criterion(output, target).item()) # sum up batch loss
            pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    val_loss = np.mean(val_loss)

    print('Validation set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)'.format(
        val_loss, correct, len(val_loader.dataset),
        100. * correct / len(val_loader.dataset)))

    history['val_loss'].append(val_loss)
    history['val_acc'].append(100. *correct / len(val_loader.dataset))

# 画图
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history['train_loss'], label='train_loss')
plt.plot(history['val_loss'], label='val_loss')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history['train_acc'], label='train_acc')
plt.plot(history['val_acc'], label='val_acc')
plt.legend()
plt.show()
```