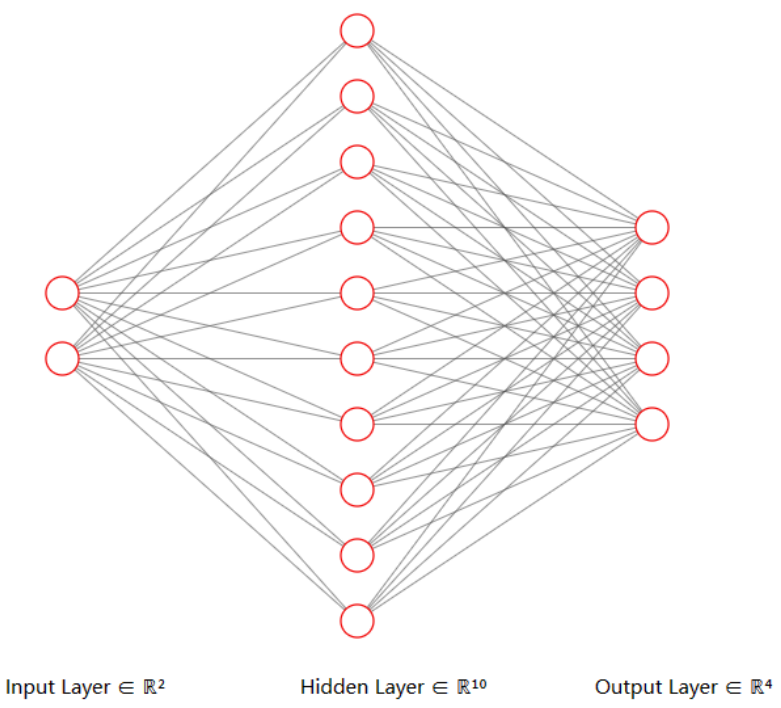
实验报告一

1 训练与测试

1.1 神经网络架构

- 包括了一个输入层，一个隐藏层和一个输出层
- 神经元数量为2：10：4
- 激活函数选用RELU函数
- 具体架构如图所示：



1.2 损失

- 我们将训练集和测试集的batch_size都设置为40

- 在每一次mini_batch训练后，损失如图所示：

```
Train Epoch 0 Iter: 0 Loss: 0.235511
Train Epoch 0 Iter: 1 Loss: 0.345759
Train Epoch 0 Iter: 2 Loss: 0.166477
Train Epoch 0 Iter: 3 Loss: 0.088853
Train Epoch 0 Iter: 4 Loss: 0.220207
Train Epoch 0 Iter: 5 Loss: 0.126337
Train Epoch 0 Iter: 6 Loss: 0.177682
Train Epoch 0 Iter: 7 Loss: 0.355542
Train Epoch 0 Iter: 8 Loss: 0.173316
Train Epoch 0 Iter: 9 Loss: 0.141509
Train Epoch 0 Iter: 10 Loss: 0.292400
Train Epoch 0 Iter: 11 Loss: 0.044007
Train Epoch 0 Iter: 12 Loss: 0.229087
Train Epoch 0 Iter: 13 Loss: 0.230752
Train Epoch 0 Iter: 14 Loss: 0.185478
Train Epoch 0 Iter: 15 Loss: 0.364863
Train Epoch 0 Iter: 16 Loss: 0.162061
Train Epoch 0 Iter: 17 Loss: 0.234663
Train Epoch 0 Iter: 18 Loss: 0.434018
Train Epoch 0 Iter: 19 Loss: 0.219914
Train Epoch 0 Iter: 20 Loss: 0.155959
```

- 在每一次epoch后，损失如图所示：

```
for epoch in range(5):
        losses=[]
        for batch_idx,(x,y) in enumerate(train_loader):
            optimizer.zero_grad()
            out = net(x)
            loss = cross_loss(out,y)
            loss.backward()
            optimizer.step()
            losses.append(loss.item())
            #print('Train Epoch {} Iter: {} Loss: {:.6f}'.format(epoch, batch_idx, loss.item()))
        print('Train Epoch: {} Loss: {:.6f}'.format(epoch, np.mean(losses)))
```

```
Train Epoch: 0 Loss: 0.729648
Train Epoch: 1 Loss: 0.276105
Train Epoch: 2 Loss: 0.232013
Train Epoch: 3 Loss: 0.224433
Train Epoch: 4 Loss: 0.223020
```

- 可以看出大概只需1-2轮训练就可以使模型收敛了

1.3 准确率

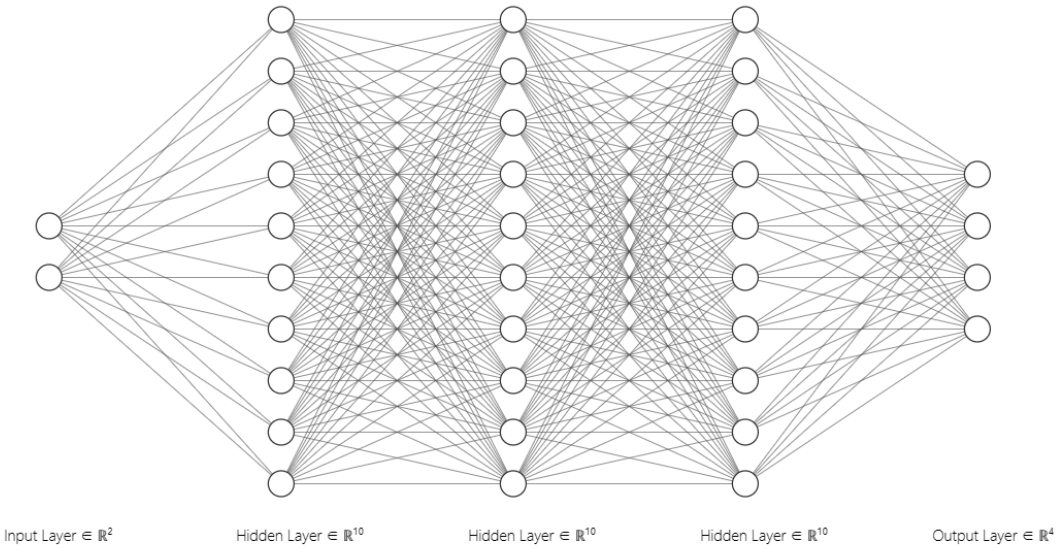
- epoch数为5

- 在训练集上的准确率为93%，在测试集上的准确率为92%，如图所示：

```
[18] net.eval()
     #最终训练集准确率
     correct=0
     for x,y in train_loader:
         out_train=net(x)
         _,predicted=torch.max(out_train.data,1)
         correct += (predicted == y).sum()
     print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(train_loader.dataset),
                                 100. * correct / len(train_loader.dataset)))

     Test set: Accuracy: 3336/3600 (93%)

     #最终测试集准确率
     correct=0
     for x,y in test_loader:
         out_test=net(x)
         _,predicted=torch.max(out_test.data,1)
         correct += (predicted == y).sum()
     print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(test_loader.dataset),
                                 100. * correct / len(test_loader.dataset)))

     Test set: Accuracy: 368/400 (92%)
```

2 尝试一：增加网络层数

- 在增加了两层层隐藏层后神经网络结构如图所示：
- 模型精度并无明显变化
- 分析：数据真实模型比较简单，不需要复杂的模型，增加隐藏层的作用不大



3 尝试二：修改神经元个数

- 尝试分别将隐藏层神经元个数改为2，5，15

3.1 神经元个数为二

- 神经元个数为二时，精度有明显下降：

```
#最终训练集准确率
correct=0
for x,y in train_loader:
        out_train=net(x)
        _,predicted=torch.max(out_train.data,1)
        correct += (predicted == y).sum()
print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(train_loader.dataset),
                    100. * correct / len(train_loader.dataset)))
```

```
Test set: Accuracy: 3068/3600 (85%)
```

```
#最终测试集准确率
correct=0
for x,y in test_loader:
        out_test=net(x)
        _,predicted=torch.max(out_test.data,1)
        correct += (predicted == y).sum()
print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(test_loader.dataset),
                    100. * correct / len(test_loader.dataset)))
```

```
Test set: Accuracy: 330/400 (82%)
```

3.2 神经元个数为五

- 神经元个数为五时，精度与初始精度相差不大：

```
#最终训练集准确率
correct=0
for x,y in train_loader:
        out_train=net(x)
        _,predicted=torch.max(out_train.data,1)
        correct += (predicted == y).sum()
print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(train_loader.dataset),
                    100. * correct / len(train_loader.dataset)))
```

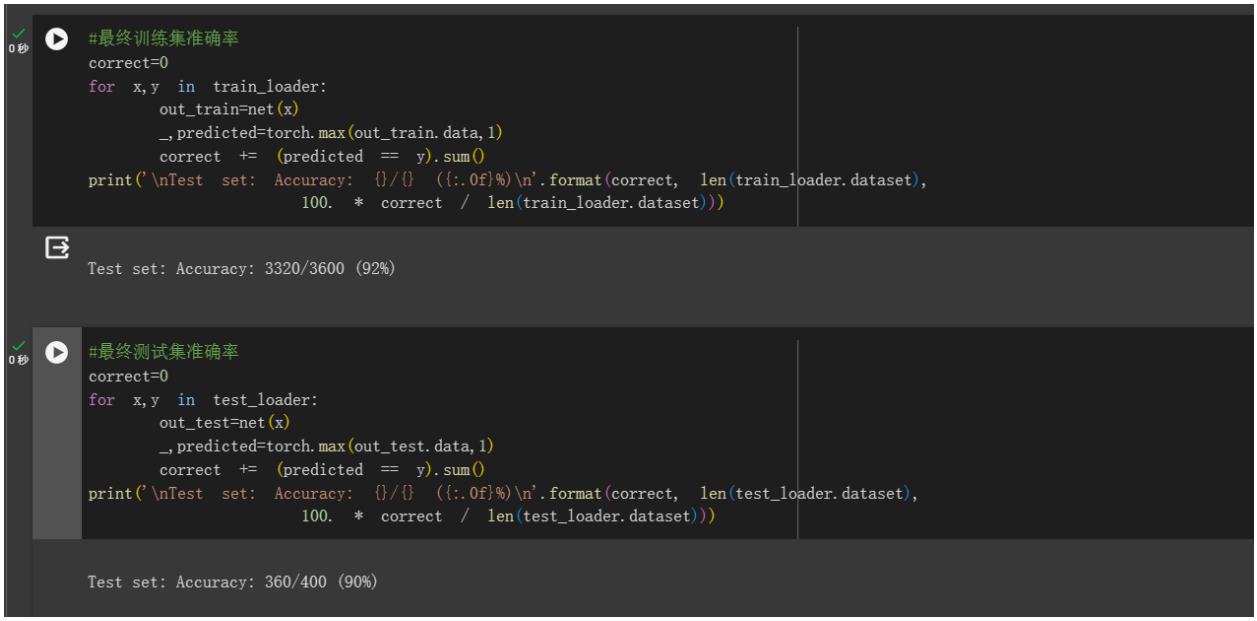
```
Test set: Accuracy: 3339/3600 (93%)
```

```
#最终测试集准确率
correct=0
for x,y in test_loader:
        out_test=net(x)
        _,predicted=torch.max(out_test.data,1)
        correct += (predicted == y).sum()
print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(test_loader.dataset),
                    100. * correct / len(test_loader.dataset)))
```

```
Test set: Accuracy: 369/400 (92%)
```

3.3 神经元个数为十五

- 神经元个数为十五时，精度相差同样不大：



```
#最终训练集准确率
correct=0
for x,y in train_loader:
        out_train=net(x)
        _,predicted=torch.max(out_train.data,1)
        correct += (predicted == y).sum()
print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(train_loader.dataset),
                    100. * correct / len(train_loader.dataset)))
```

```
Test set: Accuracy: 3320/3600 (92%)
```

```
#最终测试集准确率
correct=0
for x,y in test_loader:
        out_test=net(x)
        _,predicted=torch.max(out_test.data,1)
        correct += (predicted == y).sum()
print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(test_loader.dataset),
                    100. * correct / len(test_loader.dataset)))
```

```
Test set: Accuracy: 360/400 (90%)
```

3.4 结论

- **在epoch为5的情况下**，神经元个数在**五到十之间**就可以得到比较好的训练效果了。

4 尝试三：使用不同的激活函数

4.1 sigmoid函数

- 在epoch为5的情况下，使用sigmoid函数的准确率并没有明显变化，但是可以从图中看出，模型收敛需要的周期数更多了，说明sigmoid函数的效率在此例中低于relu

```
for epoch in range(5):
        losses=[]
        for batch_idx,(x,y) in enumerate(train_loader):
            optimizer.zero_grad()
            out = net(x)
            loss = cross_loss(out,y)
            loss.backward()
            optimizer.step()
            losses.append(loss.item())
            #print('Train Epoch {} Iter: {} Loss: {:.6f}'.format(epoch, batch_idx, loss.item()))
        print('Train Epoch: {} Loss: {:.6f}'.format(epoch, np.mean(losses)))
```

```
Train Epoch: 0 Loss: 1.066954
Train Epoch: 1 Loss: 0.574173
Train Epoch: 2 Loss: 0.396177
Train Epoch: 3 Loss: 0.323863
Train Epoch: 4 Loss: 0.285240
```

4.2 tanh函数

- tanh函数的收敛速度慢于relu，但要快于sigmoid

```
for epoch in range(5):
        losses=[]
        for batch_idx,(x,y) in enumerate(train_loader):
            optimizer.zero_grad()
            out = net(x)
            loss = cross_loss(out,y)
            loss.backward()
            optimizer.step()
            losses.append(loss.item())
            #print('Train Epoch {} Iter: {} Loss: {:.6f}'.format(epoch, batch_idx, loss.item()))
        print('Train Epoch: {} Loss: {:.6f}'.format(epoch, np.mean(losses)))
```

```
Train Epoch: 0 Loss: 0.825371
Train Epoch: 1 Loss: 0.338418
Train Epoch: 2 Loss: 0.257822
Train Epoch: 3 Loss: 0.241041
Train Epoch: 4 Loss: 0.231581
```