机器学习实验报告

题　　目：	神经网络中的前向传播与后向传播
学生姓名：	张芮熙
学　　号：	22354188
指导教师：	马倩
专业班级：	22 智科 3 班

2021 年 5 月

本科生院制

目录

1.实验目的、思路：

Red Wine Quality 是一个关于红酒品质的数据集，总共有 1599 个样本，每个样本包含 11 个(都是连续的)特征以及 1 个标签，每个标签的取值是连续的。本次实验已经按照 8：2 的比例划分成了训练数据集'wine_train.csv'以及测试数据集'wine_test.csv'，且每个数据集都已经做了归一化处理。我们要做的是搭建神经网络，用梯度下降法进行模型参数更新，记下每轮迭代中的训练损失和测试损失。画出训练损失和测试损失关于迭代轮数的折线图。

（1）搭建神经网络需要教程中的 class Net(nn.Module)，根据数据的特征（11 个输入特征值，1 个类型标签），来搭建神经网络，搭建层次结构（输入层到第一个隐藏层，到第二个隐藏层，再到输出层。）使用全连接层，激活函数是 relu 函数。

（2）把训练集和测试集的数据变成张量，再输入到神经网络中进行计算。

（3）利用给好的 SGD 函数计算损失，设定学习率是 0.01，迭代 100 次，每一次都输出训练集和测试集的误差（会发现误差越来越小，之后趋于稳定）。

（4）使用 matplotlib 函数把每一个点用线连接，测试集的用一个颜色，训练集的用另外一个颜色。

核心代码实现与测试结果

(1) 准备工作，导入数据集 (train 和 test) 并输出 train 和 test 的标签值（最后一列）

1) 读入训练数据集'wine_train.csv'与测试数据集'wine_test.csv'。

In [25]:
```
# -- Your code here --
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
# 读入数据
train_data = pd.read_csv('wine_train.csv')
test_data = pd.read_csv('wine_test.csv')
# 分离特征和标签
X_train = train_data.iloc[:, :-1].values  # 所有行，除了最后一列的数据
y_train = train_data.iloc[:, -1].values  # 所有行，只有最后一列的数据

X_test = test_data.iloc[:, :-1].values
y_test = test_data.iloc[:, -1].values
print(X_test)
print(y_test)
```

运行结果：

可见输出了一个 1279x1 的矩阵和一个 320x1 的矩阵

```
print(len(y_train))
print(len(y_test))
```

```
[0.4 0.6 0.8 ... 0.6 0.4 0.4]
[0.6 0.2 0.6 0.4 0.6 0.4 0.8 0.4 0.8 0.6 0.4 0.6 0.4 0.6 0.6 0.8 0.4 0.4
 0.4 0.6 0.6 0.  0.4 0.8 0.6 0.6 0.8 0.6 0.4 0.6 0.4 0.4 0.4 0.8 0.4 0.8
 0.6 0.6 0.6 0.4 0.4 0.2 0.6 0.4 0.6 0.4 0.4 0.6 0.8 0.6 0.8 0.6 0.6 0.4
 0.6 0.8 0.4 0.4 0.6 0.8 0.6 0.6 1.  0.4 0.4 0.4 0.6 0.6 0.6 0.4 0.4 0.6
 0.4 0.4 0.8 0.4 0.6 0.4 0.6 0.8 0.4 0.8 0.6 0.4 0.6 0.4 0.4 0.4 0.6 0.4
 0.6 0.4 0.4 0.6 0.6 0.4 0.4 0.6 0.4 0.6 0.6 0.6 0.6 0.6 0.4 0.4 0.6 0.6
 0.4 0.2 0.6 0.8 0.6 0.4 0.6 0.6 0.8 0.6 0.6 0.8 0.6 0.4 0.4 0.2 0.4 0.8
 0.6 0.4 0.6 0.4 0.6 0.6 0.6 0.8 1.  0.4 0.4 0.4 0.4 0.8 0.4 0.4 0.6 0.4
 0.6 0.4 0.2 0.4 0.6 0.4 0.4 0.4 0.6 0.6 0.6 0.4 0.4 0.8 0.6 0.4 0.4 0.4
 0.8 0.6 0.8 0.4 0.4 0.8 0.8 0.6 0.4 0.4 0.4 0.6 0.8 0.8 0.4 0.4 0.4 0.6
 0.8 0.4 0.6 0.8 0.4 0.8 0.6 0.4 0.4 0.4 0.8 0.6 0.8 0.8 0.6 0.4 0.6 0.4
 0.6 0.4 0.6 0.4 0.6 0.4 0.8 0.4 0.8 0.4 0.6 0.6 0.4 0.6 0.8 0.6 0.4 0.4
 0.4 0.4 0.6 0.6 0.4 0.6 0.4 0.6 0.8 0.8 0.4 0.6 0.4 0.8 0.6 0.4 0.4 0.4
 0.4 0.4 0.4 0.6 0.6 0.6 0.6 0.4 0.6 0.4 0.4 0.8 0.4 0.6 0.6 0.4 0.6 0.4
 0.4 0.8 0.4 0.4 0.4 0.4 0.4 0.8 0.4 0.4 0.6 0.6 0.  0.6 0.2 0.  0.4 0.4
 0.8 0.8 0.6 0.4 0.4 0.2 0.6 0.4 0.8 0.6 0.6 0.4 0.4 0.4 0.6 0.6 0.4 0.4
 0.6 0.8 0.4 0.6 0.6 0.6 0.8 0.4 0.4 0.6 0.4 0.4 0.6 0.4 0.6 0.4 0.2 0.6
 0.4 0.4 0.6 0.6 0.6 0.6 0.4 0.8 0.4 0.6 0.6 0.4 0.4 0.4]
1279
320
```

2. 搭建神经网络

In [21]:
```
# -- Your code here --
# 定义网络结构
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(11, 12)  # 输入层到隐藏层1
        self.fc2 = nn.Linear(12, 12)  # 隐藏层1到隐藏层2
        self.fc3 = nn.Linear(12, 1)   # 隐藏层2到输出层

    def forward(self, x):
        x = F.relu(self.fc1(x))  # 激活函数为ReLU
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
# 转换数据为torch张量
X_train_torch = torch.tensor(X_train, dtype=torch.float32)
y_train_torch = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
# 调整形状以匹配输出（只有一个标签值）
X_test_torch = torch.tensor(X_test, dtype=torch.float32)
y_test_torch = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)
```

注意，其中 nn.Linear 是前一层与后一层的节点全部连接，参数 12 说明是每一个点向下一层有 12 条连线。Relu 函数是激活函数，Net 就是得到的神经网络。再将 train 和 test 中的特征值和标签值改成张量，便于计算。

3.

利用梯度下降函数 SGD 进行训练，建立训练误差和测试误差的列表，每迭代一次就记录一次误差，把 learning rate 设为 0.01。这里经过多次调试，我发现只需要迭代 100 次，误差值就基本收敛在可控范围内了。在循环过程中为了准确计算要去除上一次迭代造成的影响，就是要清除旧的梯度，把损失值记录下来，就会得到两个 100x1 的列表，最后我们打印一下这两个列表，进行对比（training loss and test loss）。

3) 用梯度下降法进行模型参数更新，记下每轮迭代中的训练损失和测试损失。

In [22]:
```
# -- Your code here --
# 损失函数和优化器
criterion = nn.MSELoss()  # 均方误差
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)  # 随机梯度下降

# 记录损失
train_losses = []
test_losses = []

# 训练过程
epochs = 100
for epoch in range(epochs):
    optimizer.zero_grad()  # 清除旧的梯度
    outputs = net(X_train_torch)  # 前向传播
    loss = criterion(outputs, y_train_torch)  # 计算损失
    loss.backward()  # 反向传播
    optimizer.step()  # 更新权重
    train_losses.append(loss.item())# 记录训练损失

    # 测试损失
    with torch.no_grad():  # 确保不会计算梯度
        outputs_test = net(X_test_torch)
        loss_test = criterion(outputs_test, y_test_torch)
        test_losses.append(loss_test.item())

    print(f'Epoch {epoch+1}/{epochs}, Training Loss: {loss.item()}, Test Loss: {loss_test.item()}')
```

代码及结果：

可见随着迭代次数的增加，误差越来越小，直到收敛到一个很小的值，可见有效。

```
Epoch 1/100, Training Loss: 0.5507023930549622, Test Loss: 0.5260355472564697
Epoch 2/100, Training Loss: 0.5234230160713196, Test Loss: 0.5003194808959961
Epoch 3/100, Training Loss: 0.4977063834667206, Test Loss: 0.4760761260986328
Epoch 4/100, Training Loss: 0.4734518527984619, Test Loss: 0.45320558547973633
Epoch 5/100, Training Loss: 0.45056429505348206, Test Loss: 0.43161481618881226
Epoch 6/100, Training Loss: 0.4289630353450775, Test Loss: 0.4112185835838318
Epoch 7/100, Training Loss: 0.4085591733455658, Test Loss: 0.3919505476951599
Epoch 8/100, Training Loss: 0.3892766535282135, Test Loss: 0.3737289309501648
Epoch 9/100, Training Loss: 0.37104418873786926, Test Loss: 0.35648342967033386
Epoch 10/100, Training Loss: 0.35380247235298157, Test Loss: 0.34014892578125
Epoch 11/100, Training Loss: 0.33747923374176025, Test Loss: 0.3246748745441437
Epoch 12/100, Training Loss: 0.3220236897468567, Test Loss: 0.3100043535232544
Epoch 13/100, Training Loss: 0.30738043785095215, Test Loss: 0.296093225479126
Epoch 14/100, Training Loss: 0.2935020923614502, Test Loss: 0.28289440274238586
Epoch 15/100, Training Loss: 0.2803390920162201, Test Loss: 0.2703717350959778
Epoch 16/100, Training Loss: 0.26785019040107727, Test Loss: 0.258483350276947
Epoch 17/100, Training Loss: 0.2559957802295685, Test Loss: 0.24719682335853577
Epoch 18/100, Training Loss: 0.2447425276041031, Test Loss: 0.23647955060005188
Epoch 19/100, Training Loss: 0.23405855894088745, Test Loss: 0.2262967824935913
Epoch 20/100, Training Loss: 0.223911315202713, Test Loss: 0.21661916375160217
Epoch 21/100, Training Loss: 0.21426908671855927, Test Loss: 0.207419753074646
Epoch 22/100, Training Loss: 0.20510396361351013, Test Loss: 0.19867266714572906
Epoch 23/100, Training Loss: 0.19639034569263458, Test Loss: 0.19035473465919495
Epoch 24/100, Training Loss: 0.18810489773750305, Test Loss: 0.18244218826293945
Epoch 25/100, Training Loss: 0.18022394180297852, Test Loss: 0.17491333186626434
Epoch 26/100, Training Loss: 0.1727273017168045, Test Loss: 0.16775089502334595
Epoch 27/100, Training Loss: 0.16559727489948273, Test Loss: 0.1609344482421875
Epoch 28/100, Training Loss: 0.15881186723709106, Test Loss: 0.15444616973400116
Epoch 29/100, Training Loss: 0.1523546278476715, Test Loss: 0.14827118813991547
Epoch 30/100, Training Loss: 0.14620980620384216, Test Loss: 0.14239393174648285
Epoch 31/100, Training Loss: 0.14036092162132263, Test Loss: 0.13679835200309753
Epoch 32/100, Training Loss: 0.13479243218898773, Test Loss: 0.13146987557411194
Epoch 33/100, Training Loss: 0.12949033081531525, Test Loss: 0.12639550864696503
Epoch 34/100, Training Loss: 0.12444165349006653, Test Loss: 0.1215621829032898
Epoch 35/100, Training Loss: 0.11963711187448502, Test Loss: 0.11696042120456696
Epoch 36/100, Training Loss: 0.11505620926618576, Test Loss: 0.11257865279912949
Epoch 37/100, Training Loss: 0.11069678515195847, Test Loss: 0.10840457677841187
Epoch 38/100, Training Loss: 0.10654458403587341, Test Loss: 0.10442850738763809
Epoch 39/100, Training Loss: 0.1025911346077919, Test Loss: 0.10064174979925156
Epoch 40/100, Training Loss: 0.09882621467113495, Test Loss: 0.09703551977872849
Epoch 41/100, Training Loss: 0.09524008631706238, Test Loss: 0.09360139816999435
Epoch 42/100, Training Loss: 0.09182438999414444, Test Loss: 0.09033036977052689
Epoch 43/100, Training Loss: 0.08857093006372452, Test Loss: 0.08721574395895004
Epoch 44/100, Training Loss: 0.08547338098287582, Test Loss: 0.08424960821866989
Epoch 45/100, Training Loss: 0.08252349495887756, Test Loss: 0.08142535388469696
Epoch 46/100, Training Loss: 0.07971423864364624, Test Loss: 0.078734636306762 7
Epoch 47/100, Training Loss: 0.07703818380832672, Test Loss: 0.07617145776748657
Epoch 48/100, Training Loss: 0.07448936253786087, Test Loss: 0.07373040169477463
Epoch 48/100, Training Loss: 0.07448936253786087, Test Loss: 0.07373040169477463
Epoch 49/100, Training Loss: 0.07206201553344727, Test Loss: 0.07140588015317917
Epoch 50/100, Training Loss: 0.06975062191486359, Test Loss: 0.06919177621603012
Epoch 51/100, Training Loss: 0.06755012273788452, Test Loss: 0.06708266586065292
Epoch 52/100, Training Loss: 0.06545403599739075, Test Loss: 0.06507354974746704
Epoch 53/100, Training Loss: 0.06345779448747635, Test Loss: 0.06316027790307999
Epoch 54/100, Training Loss: 0.06155741587281227, Test Loss: 0.06133802980184555
Epoch 55/100, Training Loss: 0.05974814295768738, Test Loss: 0.05960238724946976
Epoch 56/100, Training Loss: 0.05802586302161217, Test Loss: 0.0579499714076519
Epoch 57/100, Training Loss: 0.05638653412461281, Test Loss: 0.05637623742222786
Epoch 58/100, Training Loss: 0.05482539162039757, Test Loss: 0.05487780645489693
Epoch 59/100, Training Loss: 0.05333876609802246, Test Loss: 0.05345088988542557
Epoch 60/100, Training Loss: 0.05192367732524872, Test Loss: 0.05209249258041382
Epoch 61/100, Training Loss: 0.050577014684677124, Test Loss: 0.05079952627420425 4
Epoch 62/100, Training Loss: 0.04929542914032936, Test Loss: 0.04956901445984840 4
Epoch 63/100, Training Loss: 0.048075657337903976, Test Loss: 0.0483974628150463 1
Epoch 64/100, Training Loss: 0.046914927661418915, Test Loss: 0.0472817942500114 44
Epoch 65/100, Training Loss: 0.0458095483481884, Test Loss: 0.046219587326049805
Epoch 66/100, Training Loss: 0.04475795105099678, Test Loss: 0.0452078506350517 3
Epoch 67/100, Training Loss: 0.04375696927309036, Test Loss: 0.0442446470260620 1
Epoch 68/100, Training Loss: 0.04280472919344902, Test Loss: 0.0433277972042560 6
Epoch 69/100, Training Loss: 0.0418982058763504, Test Loss: 0.042454175651073456
Epoch 70/100, Training Loss: 0.041035160422325134, Test Loss: 0.041621364653110504
Epoch 71/100, Training Loss: 0.040213894098997116, Test Loss: 0.040828771889209 75
Epoch 72/100, Training Loss: 0.03943278267979622, Test Loss: 0.040074642747640 61
Epoch 73/100, Training Loss: 0.03868987783789635, Test Loss: 0.039357639849185944
Epoch 74/100, Training Loss: 0.03798343613743782, Test Loss: 0.038675852119922 64
Epoch 75/100, Training Loss: 0.03731170669198036, Test Loss: 0.038027346134185 79
Epoch 76/100, Training Loss: 0.036672819405794144, Test Loss: 0.037410330027341 84
Epoch 77/100, Training Loss: 0.036064937710762024, Test Loss: 0.036823540925979614
Epoch 78/100, Training Loss: 0.03548726066946983, Test Loss: 0.036265742033720016
Epoch 79/100, Training Loss: 0.03493834286928177, Test Loss: 0.035734962671995 16
Epoch 80/100, Training Loss: 0.034416697919368744, Test Loss: 0.035229865461587906
Epoch 81/100, Training Loss: 0.03392099216580391, Test Loss: 0.034749429672956 47
Epoch 82/100, Training Loss: 0.03345012292265892, Test Loss: 0.034292869269847 87
Epoch 83/100, Training Loss: 0.03300287202000618, Test Loss: 0.033858694136142 73
Epoch 84/100, Training Loss: 0.032577864825725555, Test Loss: 0.033445846289396286
Epoch 85/100, Training Loss: 0.0321742482483387, Test Loss: 0.03305348008871078 5
Epoch 86/100, Training Loss: 0.03179134055972099, Test Loss: 0.032680604606866 84
Epoch 87/100, Training Loss: 0.03142772614955902, Test Loss: 0.032325960695743 56
Epoch 88/100, Training Loss: 0.03108256496489048, Test Loss: 0.031989008188247 68
Epoch 89/100, Training Loss: 0.030755097046494484, Test Loss: 0.031669184565544 13
Epoch 90/100, Training Loss: 0.030444538220763206, Test Loss: 0.031365420669317245
Epoch 91/100, Training Loss: 0.03014988824725151, Test Loss: 0.031076857820153236
Epoch 92/100, Training Loss: 0.02987033873796463, Test Loss: 0.030802750959992 41
Epoch 93/100, Training Loss: 0.029604937881231308, Test Loss: 0.030542070046067238
Epoch 94/100, Training Loss: 0.029352780431509018, Test Loss: 0.030294265598058 7
Epoch 95/100, Training Loss: 0.02911347709596157, Test Loss: 0.030058717355132103
Epoch 96/100, Training Loss: 0.02888651378452778, Test Loss: 0.029834788292646408
Epoch 97/100, Training Loss: 0.028670936822891235, Test Loss: 0.029622036963701248
Epoch 98/100, Training Loss: 0.028466302901506424, Test Loss: 0.029419964179396 63
Epoch 99/100, Training Loss: 0.028272219002246857, Test Loss: 0.029228057712316513
Epoch 100/100, Training Loss: 0.028088161721825 6, Test Loss: 0.029045632109045982
```

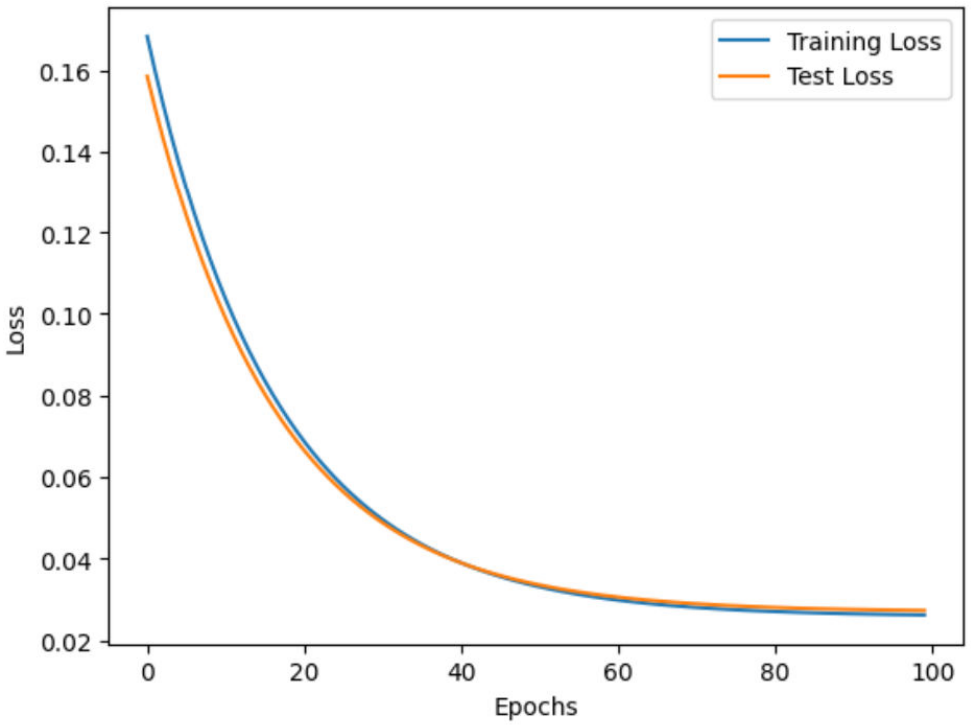
4. 绘图

把之前代码中的 2x100 个点连成线，得到两条曲线，一条是 train，一条是 test,用不同的颜色区分。

代码及结果：

4) 画出训练损失和测试损失关于迭代轮数的折线图。

In [32]:
```
# -- Your code here --
# 绘制损失曲线
plt.plot(train_losses, label='Training Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()#显示标签
plt.show()
```

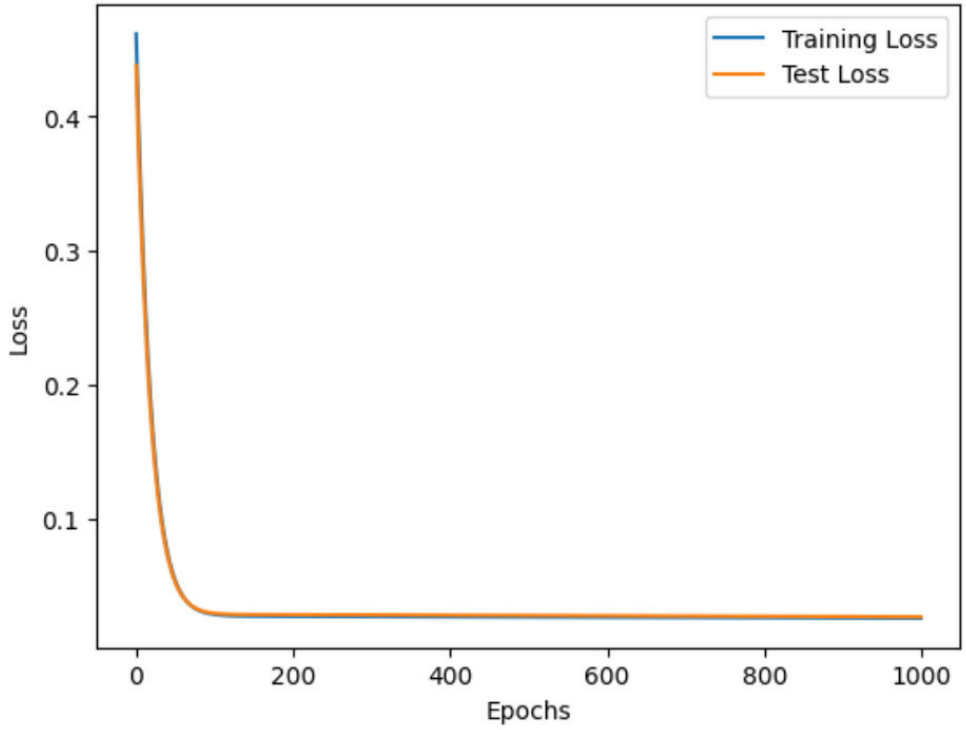


可见当迭代次数较少的时候误差还是比较大的，但当迭代次数达到 100 次之后，损失值维持在一个较小的稳定值。

这是迭代次数为 1000 次的截图，可见 100 次足以达到目的，蓝色曲线和橙色曲线高度拟合。

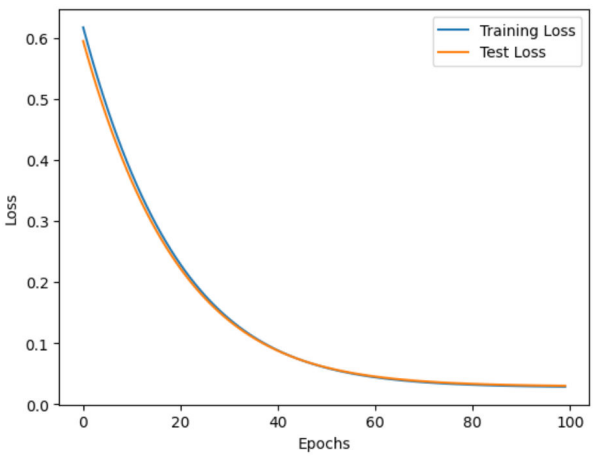
4) 画出训练损失和测试损失关于迭代轮数的折线图。

```
# -- Your code here --
# 绘制损失曲线
plt.plot(train_losses, label='Training Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()#显示标签
plt.show()
```

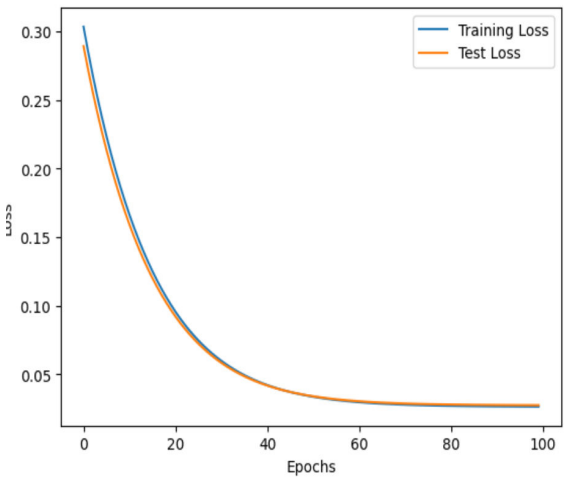


心得体会

1. 经过调试代码可得知，神经网络中连线的个数会对实验结果造成很大影响，经过调试可知 k=12 条是最佳选择，如果过多会造成过拟合，反之则为欠拟合。



K=4 时的曲线



k=64 时的曲线

2. 迭代次数和学习率要设置清楚。（老生常谈）

3. 搭建网络的时候隐藏层的个数可以通过多次调试得出，有的时候数据比较简单，就可以少设置几层，但有的时候数据复杂，就要多设置几个隐藏层，以得出准确答案。