实验报告：神经网络分类算法实验

一、实验目标

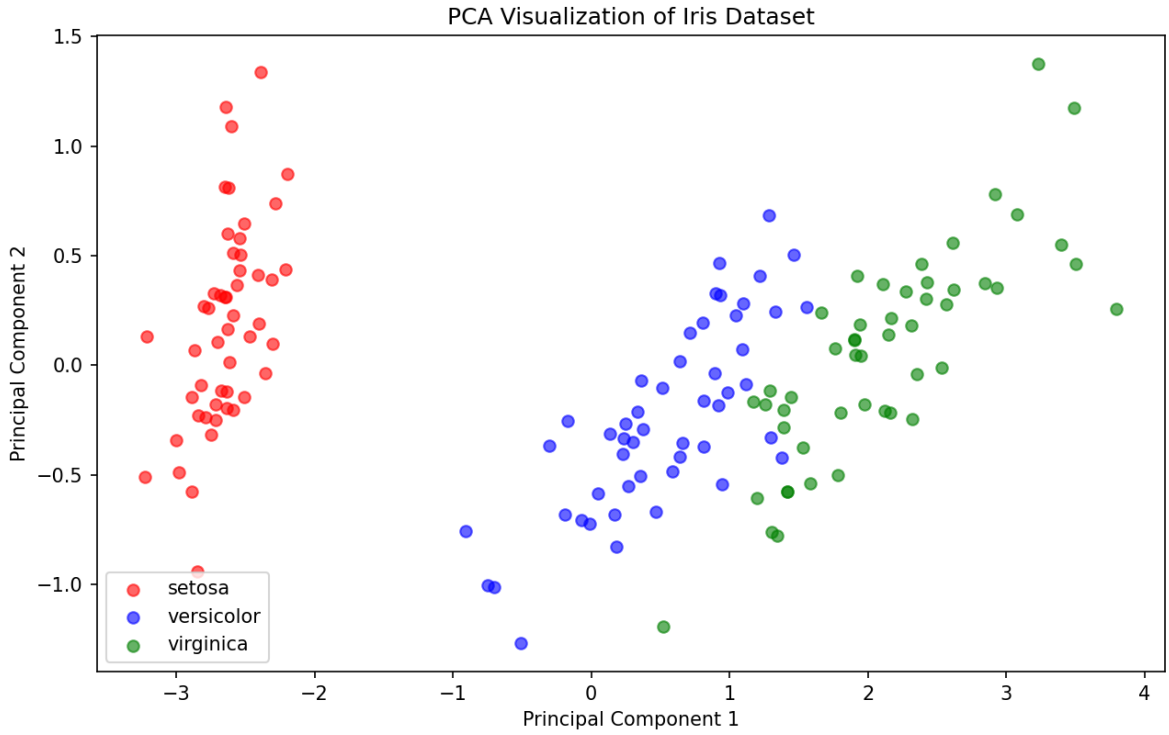
1. 理解神经网络分类算法的原理，掌握前向传播和反向传播的过程。
2. 使用 Python 实现神经网络分类算法，并通过鸢尾花（Iris）数据集进行实验。
3. 调用现有库与手动实现神经网络算法，比较模型性能，优化超参数。

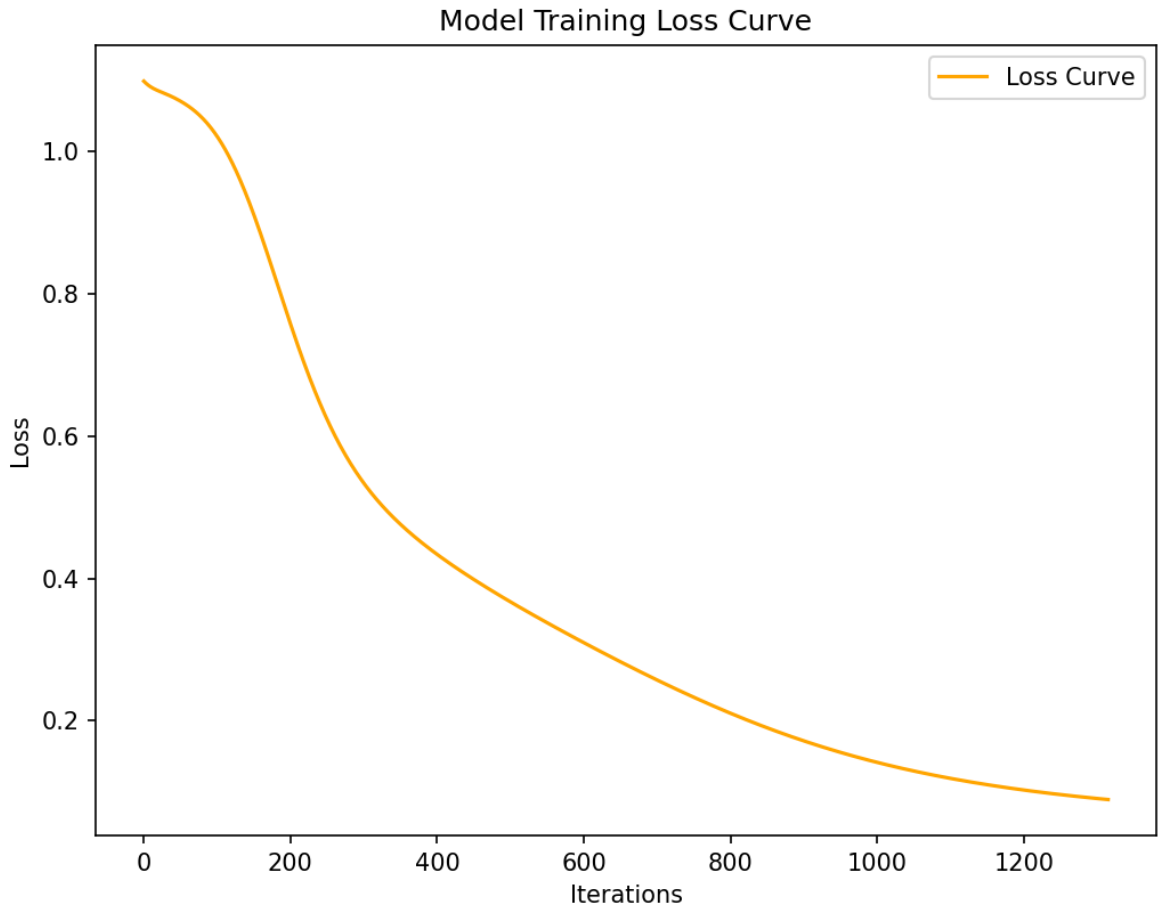
二、实验结果

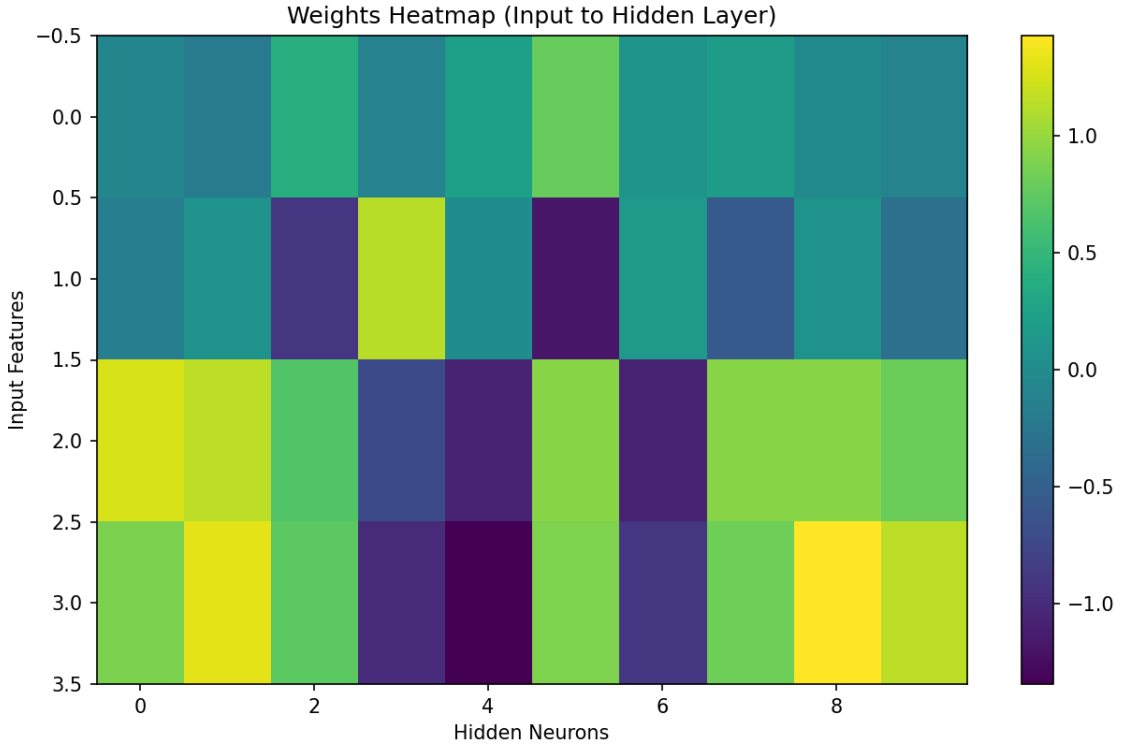
1. 调用库实现

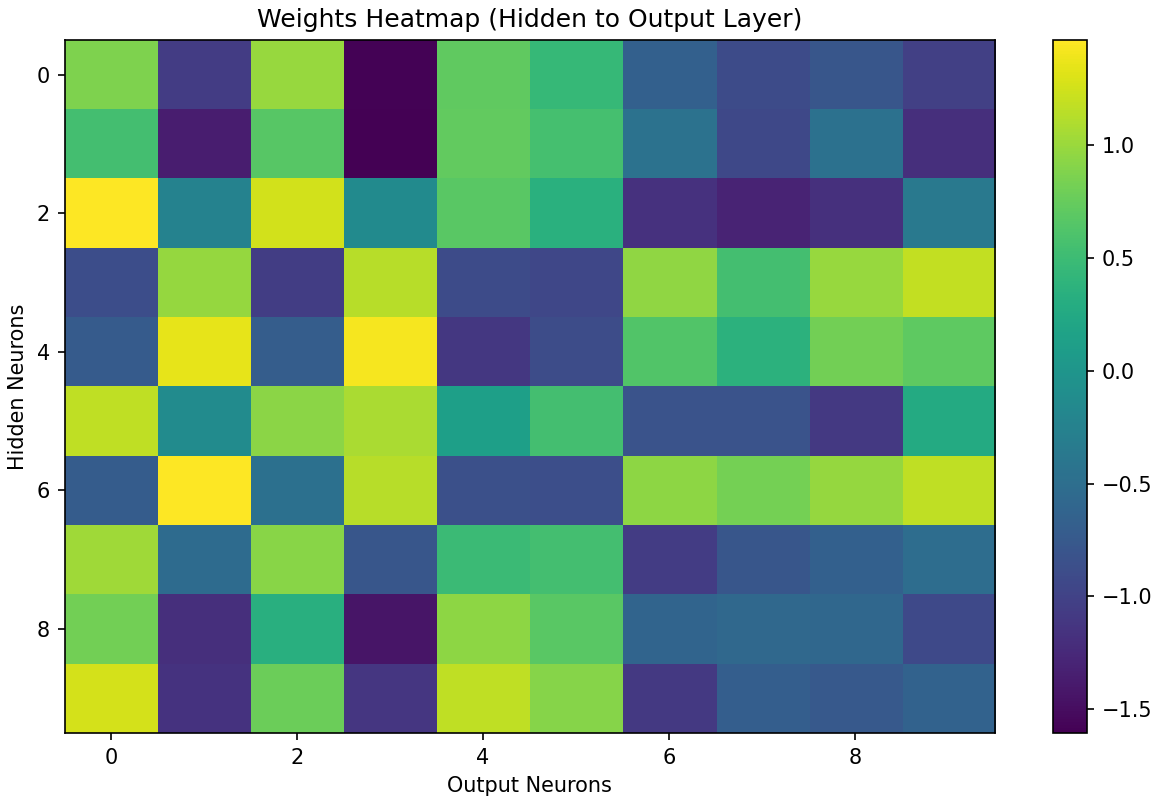
- **库：**
- 使用`MLPClassifier`（来自`scikit-learn`）。
- **结果：**

```
测试集合的 y 值： [2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0]
神经网络预测的的 y 值： [2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0]
预测的准确率为： 1.0
层数为： 4
迭代次数为： 1316
损失为： 0.08904022572282774
激活函数为： softmax
```




Model Training Loss Curve
Loss Curve
Loss
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000
1200
Iterations

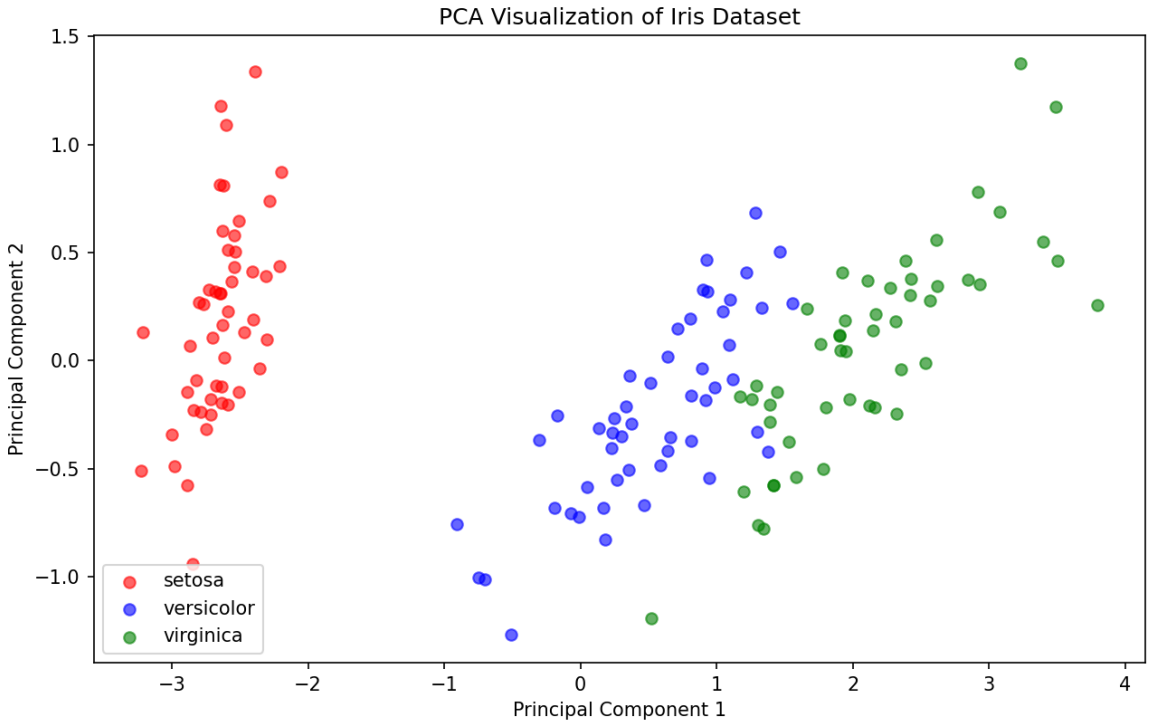

Weights Heatmap (Input to Hidden Layer)
Input Features
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
Hidden Neurons
1.0
0.5
0.0
−0.5
−1.0

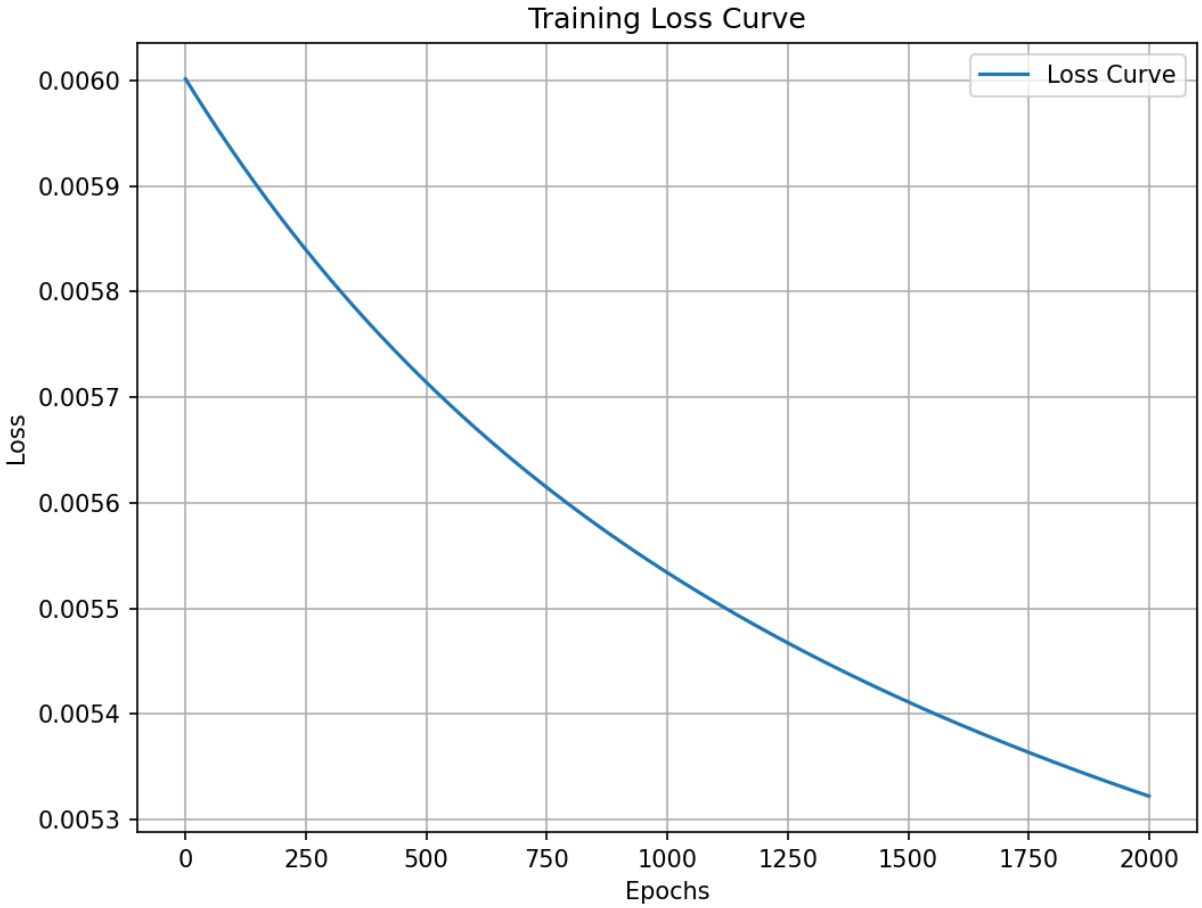


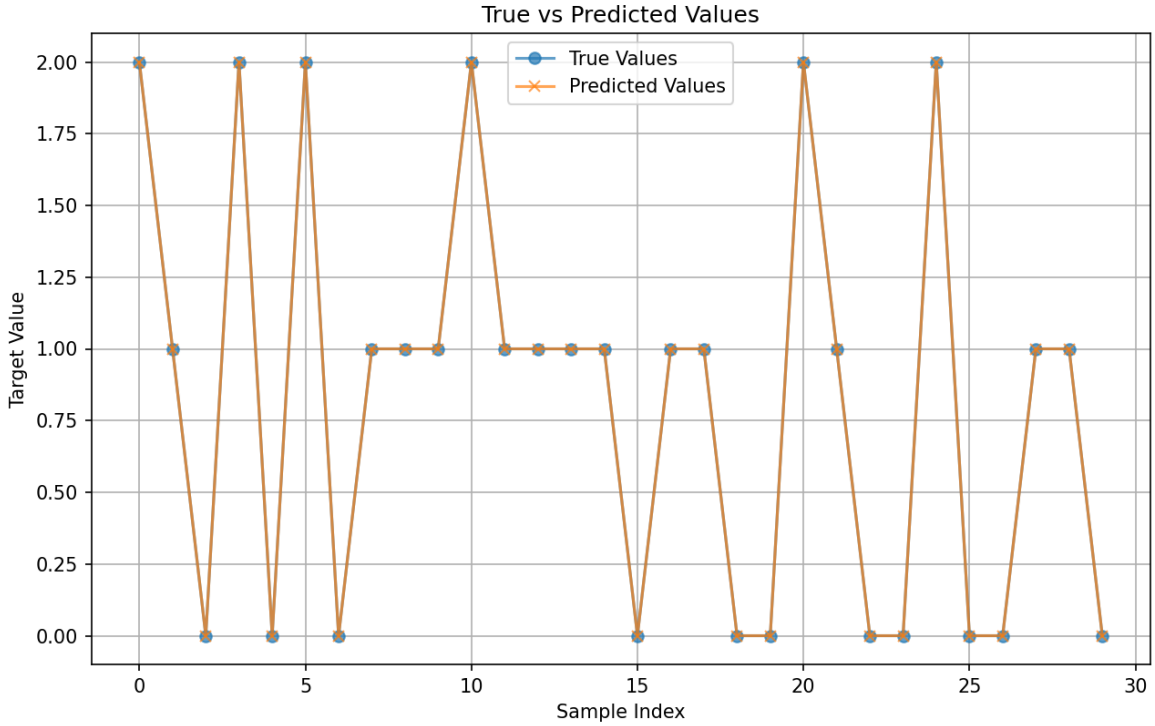
2. 手动实现神经网络

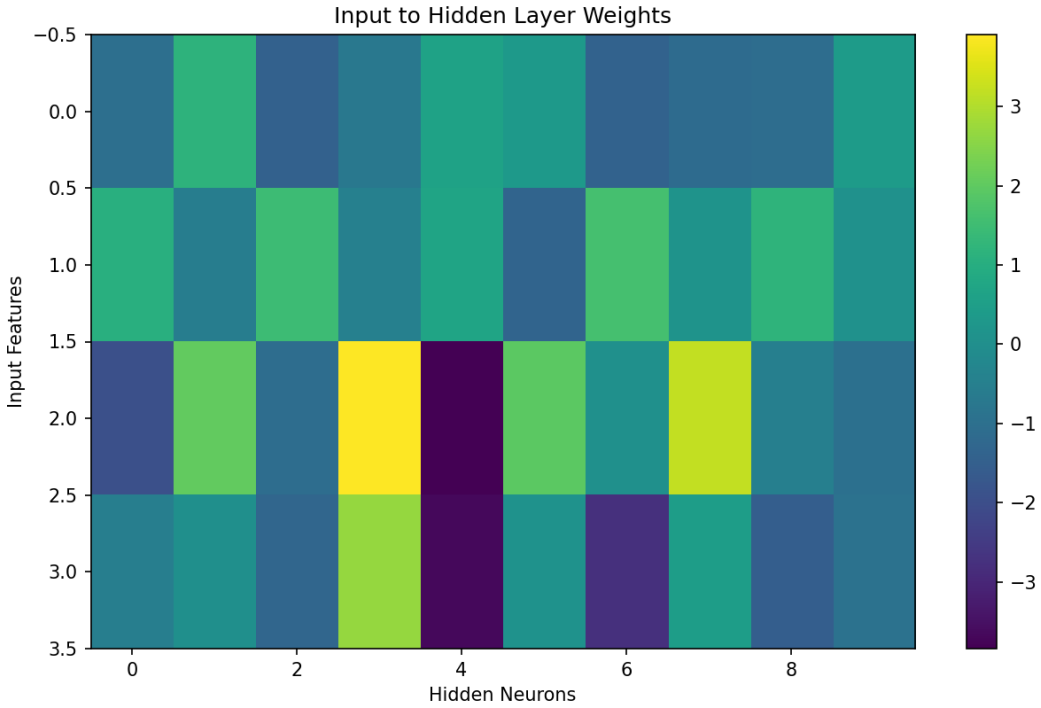
- **结果：**

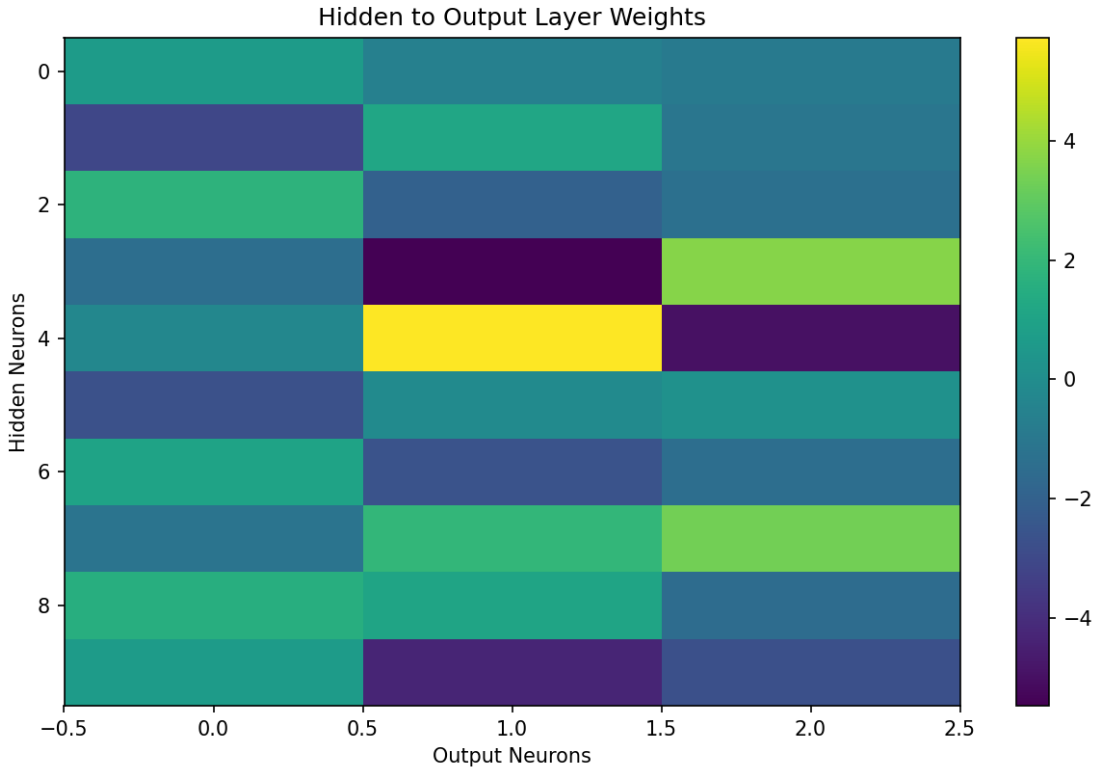
```
training.......
Epoch 1, Loss: 0.22206288012699021
Epoch 101, Loss: 0.044453351806686 53
Epoch 201, Loss: 0.035476395028742246
Epoch 301, Loss: 0.028891399464308617
Epoch 401, Loss: 0.022429996271849995
Epoch 501, Loss: 0.016849292691613355
Epoch 601, Loss: 0.013235231162443626
Epoch 701, Loss: 0.01109712660460638
Epoch 801, Loss: 0.009784113859388089
Epoch 901, Loss: 0.008922449083842468
Epoch 1001, Loss: 0.008319486881311434
Epoch 1101, Loss: 0.007874210238651055
Epoch 1201, Loss: 0.007530882213355715
Epoch 1301, Loss: 0.007256928618811877
Epoch 1401, Loss: 0.007032258232173162
Epoch 1501, Loss: 0.006843878558699 4455
Epoch 1601, Loss: 0.0066830370998349825
Epoch 1701, Loss: 0.006543625072048 2935
Epoch 1801, Loss: 0.006421244880891849
Epoch 1901, Loss: 0.006312643567161117
测试集合的 y 值： [2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0]
神经网络预测的的 y 值： [2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0]
预测的准确率为： 1.0
```


PCA Visualization of Iris Dataset
Principal Component 2
Principal Component 1
setosa
versicolor
virginica
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4


Training Loss Curve
Loss Curve
Loss
Epochs
0.0060
0.0059
0.0058
0.0057
0.0056
0.0055
0.0054
0.0053
0
250
500
750
1000
1250
1500
1750
2000


True vs Predicted Values
True Values
Predicted Values
Target Value
Sample Index
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
30


Input to Hidden Layer Weights
Input Features
Hidden Neurons
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
3
2
1
0
−1
−2
−3



三、实验的关键代码

第一部分中只需调用库即可实现，关键在于手动实现神经网络算法，其中的重要代码及注释如下

```
def forward(self, X):
        self.hidden_layer_input = np.dot(X, self.weights_input_hidden) +
self.bias_hidden  #隐藏层输入
        self.hidden_layer_output = self.sigmoid(self.hidden_layer_input)  # 隐藏层
输出
        self.output_layer_input = np.dot(self.hidden_layer_output,
self.weights_hidden_output) + self.bias_output  #输出层输入
        self.output = self.sigmoid(self.output_layer_input)  # 输出层输出
        return self.output

    def backward(self, X, y, output, learning_rate):
        # 反向传播更新权重和偏置
        output_error = y - output  # 输出层误差
        output_delta = output_error * self.sigmoid_derivative(output)  # 输出层梯度

        hidden_error = output_delta.dot(self.weights_hidden_output.T)  # 隐藏层误差
        hidden_delta = hidden_error *
self.sigmoid_derivative(self.hidden_layer_output)  # 隐藏层梯度

        # 更新隐藏层到输出层的权重和偏置
        self.weights_hidden_output += self.hidden_layer_output.T.dot(output_delta)
* learning_rate
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True) *
learning_rate

        # 更新输入层到隐藏层的权重和偏置
```

```
        self.weights_input_hidden += X.T.dot(hidden_delta) * learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) *
learning_rate
```

四、对比与分析

- **调用库实现：**
- 测试不同的隐藏层神经元数量（5个、10个），发现10个神经元时模型表现最佳。
- 下图为5个的情况，显然最大迭代次数过小导致还未达到最优化，准确率也有所下降

```
D:\miniconda\envs\ml24\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:690: ConvergenceWarning: Stoch
astic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.
  warnings.warn(
测试集合的 y 值： [2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0]
神经网络预测的的 y 值： [2, 1, 0, 2, 0, 2, 0, 1, 2, 1, 2, 1, 1, 2, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0]
预测的准确率为： 0.9333333333333333
层数为： 4
迭代次数为： 1000
损失为： 0.22397887729023802
激活函数为： softmax
```

- **手动实现：**
- 测试测试不同的隐藏层神经元数量（5个、10个），发现10个神经元时模型表现最佳。
- 下图为5个的情况，其中lr=0.001，epochs=1000，显然准确率很差

```
training.......
Epoch 1, Loss: 0.15576780077240202
Epoch 101, Loss: 0.09093284708517134
Epoch 201, Loss: 0.0749059645635727
Epoch 301, Loss: 0.06599410230663458
Epoch 401, Loss: 0.060306649373556266
Epoch 501, Loss: 0.05648758455123487
Epoch 601, Loss: 0.05370717613112923
Epoch 701, Loss: 0.05150578435028507
Epoch 801, Loss: 0.04963397779039605
Epoch 901, Loss: 0.047955157029091006
测试集合的 y 值： [2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0]
神经网络预测的的 y 值： [2, 0, 0, 2, 0, 2, 0, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0]
预测的准确率为： 0.5666666666666667
```

五、实验时间

实验总耗时：约8小时（包括代码调试、结果可视化和报告撰写）。

七、总结与反馈

- 本次实验比起第一次来说简单了不少~~当然也是因为遭受了lab1的折磨已经变得更强~~，

- 当然不知道是什么原因，vscode突然无法运行文件输出，报错如下，但是用Prompt还是能运行下去

```
Traceback (most recent call last):
  File "d:\MLlab\USTC-ML24-Fall\lab2\bp-algorithm.py", line 6, in <module>
    from pandas import DataFrame
  File "D:\miniconda\Lib\site-packages\pandas\__init__.py", line 49, in <m
odule>
    from pandas.core.api import (
  File "D:\miniconda\Lib\site-packages\pandas\core\api.py", line 1, in <mo
dule>
    from pandas._libs import (
  File "D:\miniconda\Lib\site-packages\pandas\_libs\__init__.py", line 17,
 in <module>
    import pandas._libs.pandas_datetime  # noqa: F401 # isort: skip # type
: ignore[reportUnusedImport]
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ImportError: numpy.core.multiarray failed to import
PS D:\MLlab\USTC-ML24-Fall\lab2>
```

- 可视化文件在lab2的bp_visualize1和bp_visualize2文件中