吴欣怡 20300246007 日期：2025 年 3 月 13 日

# 03 前馈神经网络

## 前言：原始结果

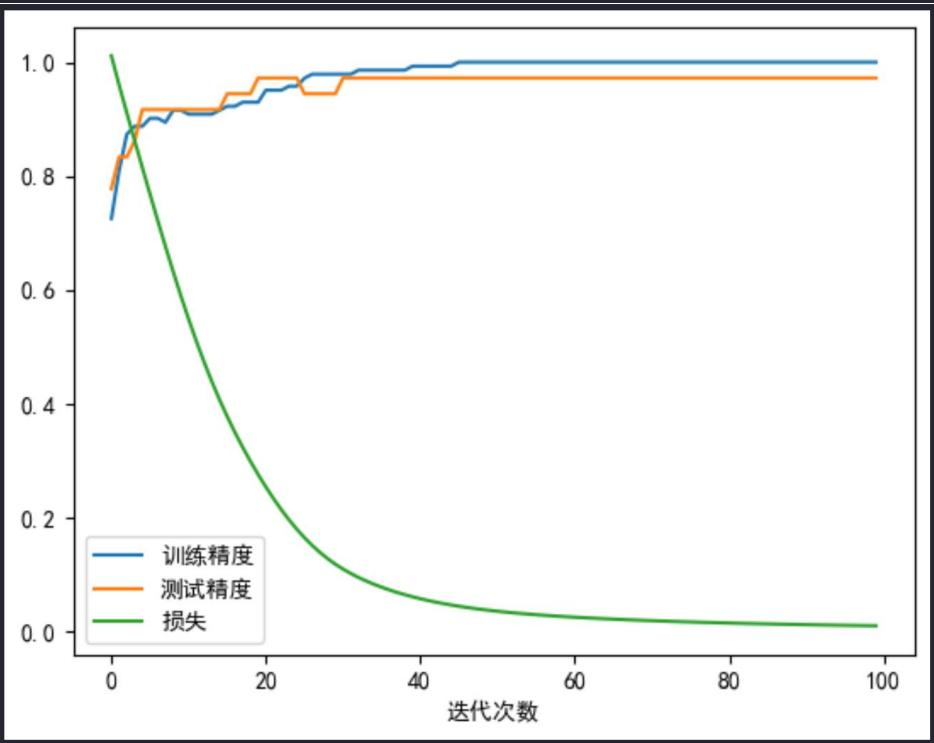
将有关前馈神经网络的.ipynb 原代码的数据集更改成 wine 数据集，并根据数据集的维度，将神经网络的输入维度更改成 13，得到了以下的结果：

```python
# <7> 训练模型
from tqdm import tqdm  # 进度条库 !pip install tqdm

# 初始化训练过程中的指标：训练精度、测试精度、损失，用于绘图
train_acc_history, test_acc_history, loss_history = [], [], []

num_epochs = 100  # 迭代次数，重复训练100次
model.train()     # 训练模式（默认）
# for epoch in range(num_epochs):
for epoch in tqdm(range(num_epochs)):
    # 1）向前传播计算结果
    outputs = model(X_train)
    # 2）计算损失
    loss = Loss(outputs, Y_train)
    # 3）反向传播，计算梯度
    loss.backward()
    # 4）更新权重
    optimizer.step()
    optimizer.zero_grad()  # 梯度清零，准备下一次迭代

    # 更新绘图指标
    train_acc_history.append(test(model, X_train, Y_train, quiet=True)[0])
    test_acc_history.append(test(model, X_test, Y_test, quiet=True)[0])
    loss_history.append(loss.item())

    # 打印训练信息
    if (epoch + 1) % 10 == 0:
        print(f"Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}")
```

```
  1%|          | 1/100 [00:00<00:11,  8.76it/s]
Epoch [10/100], Loss: 0.5893
 64%|██████▍   | 64/100 [00:00<00:00, 238.59it/s]
Epoch [20/100], Loss: 0.2757
Epoch [30/100], Loss: 0.1181
Epoch [40/100], Loss: 0.0608
Epoch [50/100], Loss: 0.0369
Epoch [60/100], Loss: 0.0258
Epoch [70/100], Loss: 0.0193
100%|██████████| 100/100 [00:00<00:00, 224.19it/s]
Epoch [80/100], Loss: 0.0151
Epoch [90/100], Loss: 0.0121
Epoch [100/100], Loss: 0.0099
```

```python
# <8> 绘制训练过程中的指标
import matplotlib.pyplot as plt
from matplotlib import rcParams  # 设置全局参数，为了设置中文字体

# 定义绘制函数
def draw_plot(train_acc_history, test_acc_history, loss_history):
    rcParams['font.family'] = 'simHei' # 设置中文黑体字
    plt.figure()
    plt.plot(train_acc_history, label="训练精度")
    plt.plot(test_acc_history,  label="测试精度")
    plt.plot(loss_history,      label="损失")
    plt.legend() # 显示图例
    plt.xlabel("迭代次数")
    plt.show()

draw_plot(train_acc_history, test_acc_history, loss_history)
print(f"最终训练精度: {train_acc_history[-1]:.0%}")
print(f"最终测试精度: {test_acc_history[-1]:.0%}")
```

✓ 0.5s

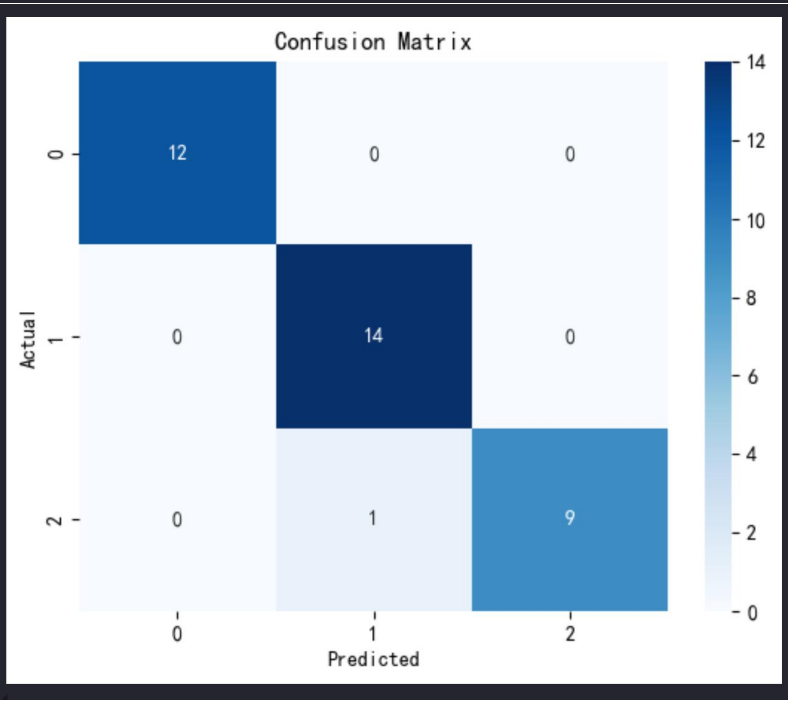



最终训练精度: 100%
最终测试精度: 97%

```python
# <9> 模型评估
# 计算测试集的结果
accuracy, predicted = test(model, X_test, y_test)

# 计算混淆矩阵(Confusion Matrix)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predicted)
print(cm)

import seaborn
# 绘制热力图
seaborn.heatmap(cm,           # 混淆矩阵
                annot=True,   # 显示数字
                cmap='Blues') # 颜色

import matplotlib.pyplot as plt
# 绘图
plt.xlabel('Predicted') # x轴标签, 预测值
plt.ylabel('Actual')    # y轴标签, 真实值
plt.title('Confusion Matrix')
plt.show()
```

[18] ✓ 1.5s

```
输出值(1-5):
tensor([[ 9.4223, -2.0626, -3.8348],
        [-3.1103,  1.9326,  0.8015],
        [ 7.5883, -1.5388, -3.3972],
        [ 1.5747,  2.2514, -3.8159],
        [-1.0313,  2.0855, -1.7105]])
最大值(1~5):
tensor([9.4223, 1.9326, 7.5883, 2.2514, 2.0855])
最大值序号(1~5): tensor([0, 1, 0, 1, 1])
正确总数: tensor(35)
正确总数: 35
test总数: 36
准确率: 97%
[[12  0  0]
 [ 0 14  0]
 [ 0  1  9]]
```



接着，并对原始代码按题目要求进行更改，并进行结果的对比。

前导：导入所有即将用到的库：

[43]:
```
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np
import torch
import torch.nn as nn
```

首先，导入机器学习中所需要用到的数据：

[3]:
```
wines = load_wine()
X=wines.data
Y=wines.target
```

将数据集进行标准化：

[4]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

接着，根据题目要求，将源代码数据集划分方式从 train_test_split 更改成 Kfold，通过 Kfold 将数据集合分 5 组，用于后面的训练过程。每次训练时候轮流取其一当测试集其余 4 组将为训练集，共训练 5 次。

[5]:
```
kf = KFold(n_splits=5, # 10-fold
shuffle=True,
random_state=42)
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index)
    print("TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]
```

运行 kfold 后，将分组结果打印，将发现每个数据都将被放入一次测试集：

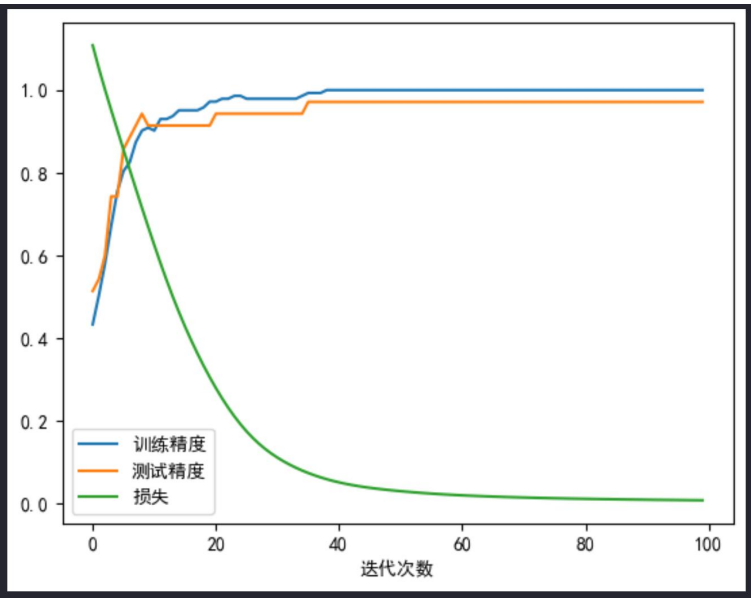
```
TRAIN: [  0   1   2   3   4   5   6   7   8  10  11  13  14  17  20  21  22  23
  25  26  27  28  32  33  34  35  36  37  38  39  40  43  44  46  47  48
  49  50  51  52  53  54  56  57  58  59  61  62  63  64  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  83  84  85  86  87  88  89  91
  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 110
 112 115 116 120 121 122 123 124 125 126 127 129 130 131 132 133 134 135
 136 137 138 139 142 143 144 146 147 148 149 151 152 153 154 155 156 157
 158 159 160 161 162 163 165 166 167 168 170 172 173 175 176 177]
TEST: [  9  12  15  16  18  19  24  29  30  31  41  42  45  55  60  65  66  67
  82  90 109 111 113 114 117 118 119 128 140 141 145 150 164 169 171 174]
```

```
TRAIN: [  0   1   3   5   7   8   9  10  12  13  14  15  16  17  18  19  20  21
  23  24  25  28  29  30  31  33  34  35  37  39  40  41  42  43  44  45
  46  47  48  49  50  52  53  54  55  57  58  59  60  61  62  63  64  65
  66  67  70  71  72  73  74  75  77  79  80  81  82  83  84  86  87  88
  89  90  91  92  94  96  99 101 102 103 105 106 107 109 110 111 112 113
 114 115 116 117 118 119 120 121 123 124 125 126 127 128 129 130 131 132
 133 134 135 136 139 140 141 142 145 147 148 149 150 151 152 155 156 157
 160 161 162 163 164 165 166 168 169 171 172 173 174 175 176 177]
TEST: [  2   4   6  11  22  26  27  32  36  38  51  56  68  69  76  78  85  93
  95  97  98 100 104 108 122 137 138 143 144 146 153 154 158 159 167 170]
TRAIN: [  1   2   3   4   5   6   7   8   9  11  12  13  14  15  16  17  18  19
  20  21  22  24  26  27  29  30  31  32  34  35  36  37  38  41  42  43
  45  46  48  49  50  51  52  53  54  55  56  57  58  59  60  63  65  66
  67  68  69  70  71  72  74  76  77  78  80  82  83  85  87  88  89  90
  91  92  93  95  97  98  99 100 102 103 104 105 106 107 108 109 110 111
 112 113 114 115 116 117 118 119 121 122 124 128 129 130 131 134 136 137
 138 140 141 143 144 145 146 148 149 150 151 152 153 154 155 156 157 158
 159 160 161 163 164 165 167 168 169 170 171 172 174 175 176 177]
TEST: [  0  10  23  25  28  33  39  40  44  47  61  62  64  73  75  79  81  84
  86  94  96 101 120 123 125 126 127 132 133 135 139 142 147 162 166 173]
TRAIN: [  0   2   3   4   5   6   7   8   9  10  11  12  13  15  16  17  18  19
  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  38  39  40
  41  42  43  44  45  46  47  49  51  53  55  56  59  60  61  62  64  65
  66  67  68  69  70  72  73  75  76  77  78  79  80  81  82  83  84  85
  86  89  90  91  93  94  95  96  97  98 100 101 104 105 108 109 110 111
 112 113 114 115 117 118 119 120 122 123 124 125 126 127 128 131 132 133
 134 135 136 137 138 139 140 141 142 143 144 145 146 147 150 152 153 154
 155 156 158 159 160 161 162 164 165 166 167 169 170 171 173 174 176]
TEST: [  1  14  20  21  37  48  50  52  54  57  58  63  71  74  87  88  92  99
 102 103 106 107 116 121 129 130 148 149 151 157 163 168 172 175 177]
```

题目要求每更改一个地方就和原始结果进行对比，运行更改后的代码结果如下：

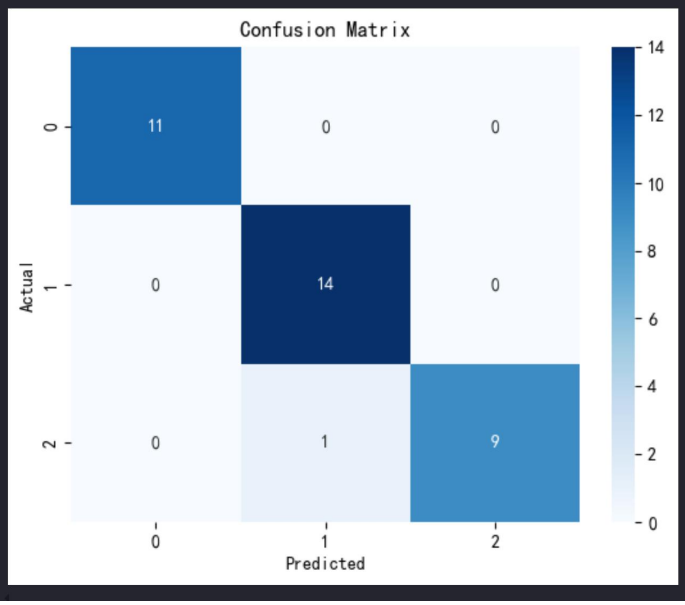
```
...    49%|████▉     | 49/100 [00:00<00:00, 173.80it/s]
    Epoch [10/100], Loss: 0.6698
    Epoch [20/100], Loss: 0.3059
    Epoch [30/100], Loss: 0.1215
    Epoch [40/100], Loss: 0.0554
    Epoch [50/100], Loss: 0.0318
    Epoch [60/100], Loss: 0.0211
    100%|██████████| 100/100 [00:00<00:00, 187.88it/s]
    Epoch [70/100], Loss: 0.0155
    Epoch [80/100], Loss: 0.0122
    Epoch [90/100], Loss: 0.0100
    Epoch [100/100], Loss: 0.0085
```

和原始代码的结果对比，更改成 Kfold 分割法并训练 100 个 epoch 后损失从 0.0099 变 0.0085。




```
最终训练精度：100%
最终测试精度：97%
```

```
输出值(1-5)：
tensor([[ 6.2747, -1.8014, -4.0768],
        [12.3641, -5.2673, -5.6190],
        [ 7.8807, -2.4791, -4.5112],
        [ 4.3682, -1.2070, -3.0958],
        [ 3.6959, -1.4027, -1.6858]])
最大值(1~5)：
tensor([ 6.2747, 12.3641,  7.8807,  4.3682,  3.6959])
最大值序号(1~5)： tensor([0, 0, 0, 0, 0])
正确总数： tensor(34)
正确总数： 34
test总数： 35
准确率：97%
[[11  0  0]
 [ 0 14  0]
 [ 0  1  9]]
```




虽然如此，准确率仍然为 97%。

接着，将测试集和数据集转换成张量，：

```
[7]: X_train = torch.FloatTensor(X_train)
     Y_train = torch.LongTensor(Y_train)
     X_test = torch.FloatTensor(X_test)
     Y_test = torch.LongTensor(Y_test)
```

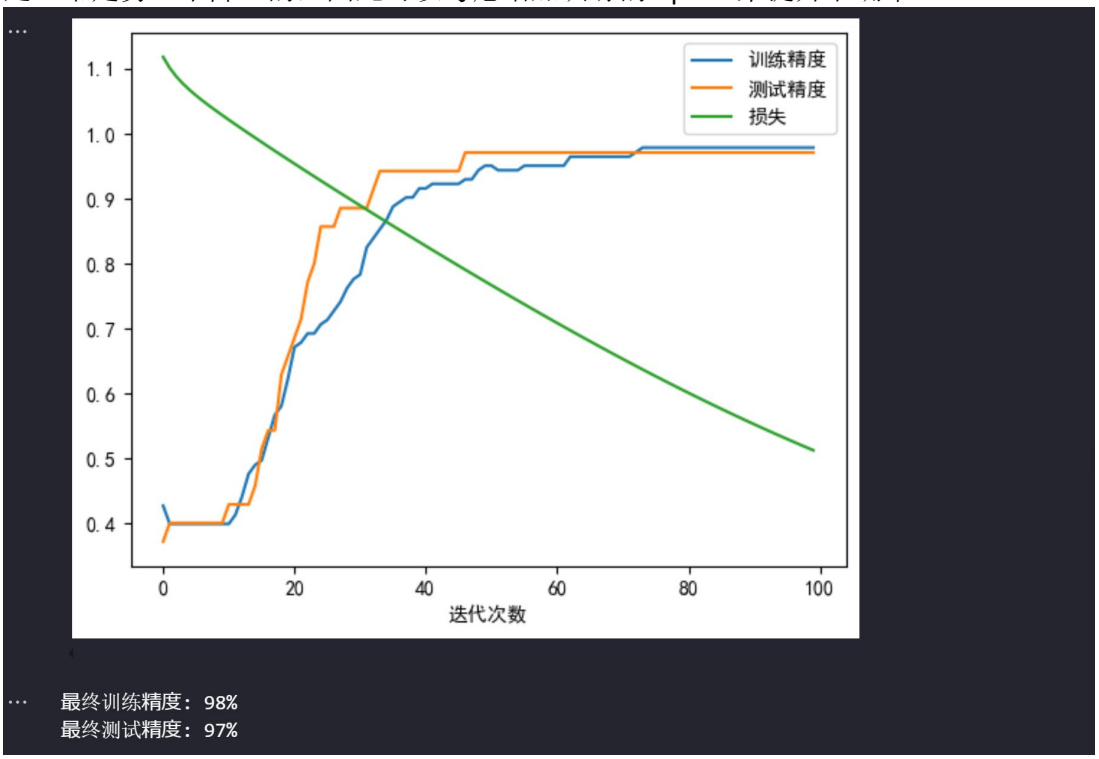
建立神经网络模型，根据题目要求，隐藏层设置了 nn.Linear(13,50)，即输入维度 13，隐藏神经元数（输出维度）为 50，激活函数为 nn.Sigmoid(),再通过隐藏神经元数（输入维度）为 50，输出维度为 3 的 nn.Linear()层。然后，计算交叉熵作为损失值。此外，优化器依题目选择了 SGD，其中的学习系数也改为 0.05

```
[48]: model = nn.Sequential(
      nn.Linear(13, 50),
      nn.Sigmoid(),
      nn.Linear(50, 3) )

      Loss = nn.CrossEntropyLoss()

      optimizer = torch.optim.SGD(
      model.parameters(),
      lr=0.05 )
```

```
···   47%|████▋     | 47/100 [00:00<00:00, 231.50it/s]
    Epoch [10/100], Loss: 1.0290
    Epoch [20/100], Loss: 0.9607
    Epoch [30/100], Loss: 0.8960
    Epoch [40/100], Loss: 0.8334
    Epoch [50/100], Loss: 0.7727
    Epoch [60/100], Loss: 0.7141
    100%|██████████| 100/100 [00:00<00:00, 238.15it/s]
    Epoch [70/100], Loss: 0.6582
    Epoch [80/100], Loss: 0.6057
    Epoch [90/100], Loss: 0.5569
    Epoch [100/100], Loss: 0.5121
```

由于隐藏层从原本的 10 变成了 50，因此导致涉及的前导函数与过程变得更为复杂，在有限的 epoch 内输出的优化也会比较困难，因此损失反而比原始结果高。

此外，通过曲线图可以判断其实模型训练和测试的精度是正常上升，损失的曲线走势是正常走势（下降）的，因此可以考虑增加训练的 epoch 来提升准确率。



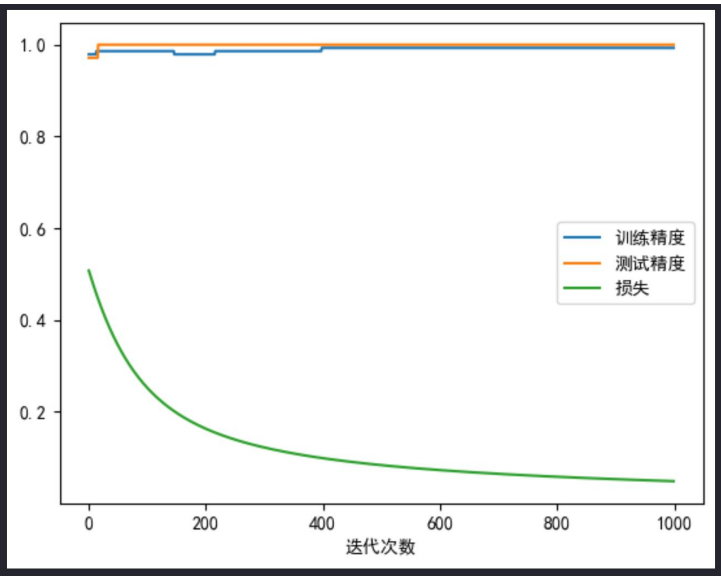

设置自定义函数 test(), 用于后续进行模型训练后，对所训练出的模型进行测试，计算模型的准确率：

[49]:
```
def test(model, X_test, Y_test, quiet=False,):
    model.eval()
    with torch.no_grad():
        outputs = model(X_test)
    _, predicted = torch.max(outputs.data, 1)
    accuracy = (predicted ==Y_test).sum().item() / Y_test.size(0)
    print('正确总数: ', (predicted == Y_test).sum())
    print('正确总数: ', (predicted == Y_test).sum().item())
    print('test总数: ', Y_test.size(0))
    return accuracy, predicted
```

根据题目要求从训练 100 epoch 更改成训练 1000 epoch，每训练一个 epoch 便通过先前自定义的 test 函数对测试集进行测试，每 10 个 epoch 打印一次损失率：

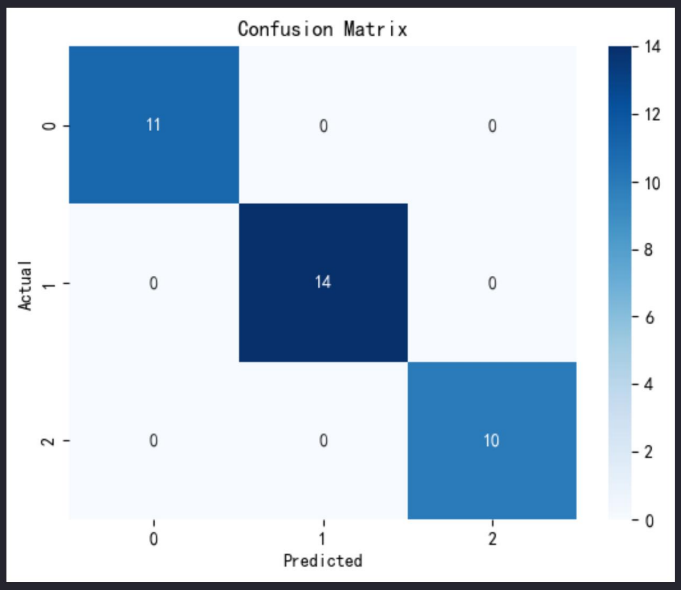
[10]:
```
from tqdm import tqdm
train_acc_history, test_acc_history, loss_history = [], [], []
num_epochs = 1000
model.train()
for epoch in tqdm(range(num_epochs)):
    outputs = model(X_train)
    loss = Loss(outputs, Y_train)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    train_acc_history.append(test(model, X_train, Y_train, quiet=True)[0])
    test_acc_history.append (test(model, X_test, Y_test, quiet=True)[0])
    loss_history.append(loss.item())

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

从下可见经过 epoch 数量的增加，损失曲线终于下降至比较理想的状态，且准确率从 97%提升至 100%




```
最终训练精度: 99%
最终测试精度: 100%
```

```
输出值(1-5):
tensor([[ 3.8653, -0.2137, -3.8239],
        [ 6.1611, -3.1243, -3.2588],
        [ 4.1529, -0.2857, -4.0060],
        [ 2.4227, -0.2119, -2.3097],
        [ 2.4386, -0.3994, -2.1975]])
最大值(1~5):
tensor([3.8653, 6.1611, 4.1529, 2.4227, 2.4386])
最大值序号(1~5): tensor([0, 0, 0, 0, 0])
正确总数: tensor(35)
正确总数: 35
test总数: 35
准确率: 100%
[[11  0  0]
 [ 0 14  0]
 [ 0  0 10]]
```




感想：这次的实验也不难，主要是改参数就行。