



# 实 验 报 告

年 级 专 业: 20 级智能科学与技术

学 生 姓 名: 孙成

学 号: 20203101694

课 程 名 称: 机器学习

实 验 名 称: 神经网络

任 课 老 师: 胡祝华

实验报告成绩	
任课教师签名	

海南大学 · 信息与通信工程学院

School of Information and Communication Engineering, Hainan University

实验室	学院 118	计算机号	自带电脑
实验软硬件环境	1. 实验所用到的硬件环境 MacBook Pro (16-inch, 2019) 2.6GHz 六核 Intel Core i7 16 GB2667 MHz DDR4  2. 实验所用到的软件环境 PyCharm 2021.3.2 (Professional Edition)		
实验目的或要求	实验目的： 1. 掌握神经元模型及其数学表示方法； 2. 掌握 MLP 及多层神经网络的表示法； 3. 熟悉从 0 开始搭建 MLP 神经网络的方法； 4. 掌握 BP 神经网络的误差逆传播算法的原理和推导过程； 5. 熟悉避免过拟合的方法； 6. 利用鸮尾花数据集，实现前向传播、反向传播，可视化 loss 曲线。  实验要求： 1. 按照实验步骤独立完成实验。 2. 整理并上交实验报告和实验源程序。 ..\网络应用开发实验报告模板.doc 3. 考核：以学生的实验报告情况和做实验时的表现为考核依据。		
实验内容	实验 5.1 从 0 开始搭建 MLP 神经网络 1) 导入标准库 2) 导入数据集：从 sklearn 库中导入 make moon 数据集 3) 绘制数据分布散点图 4) 神经网络参数初始化 5) 神经网络前向传播 6) 计算神经网络的代价函数:交叉熵损失函数（为什么要用交叉熵？） 7) 神经网络的反向传播 8) 更新参数 W 和 b 9) 构建神经网络模型并训练 10) 用神经网络模型进行预测，定义预测函数 11) 利用第 1-第 3 步构建的数据集进行模型的测试。数据集的生成和测试过程写在 testMLP.py 文件中，即把模型部分的代码和测试部分的代码进行解耦操作。测试过程如下： 1. 首先，构建 MLP 的对象模型 2. 构建神经网络，获得训练好的模型参数 3. 利用训练好的模型进行预测，打印预测结果		

	<div data-bbox="340 178 1058 254" data-label="List-Group"><ol style="list-style-type: none"><li>4. 利用 <code>np.mean</code> 函数计算预测准确率，并打印预测精度。</li><li>5. 绘制分类直线，可视化分类效果。</li></ol></div> <div data-bbox="287 301 1040 335" data-label="Section-Header"><p>实验 5.2 BP 神经网络的实现（多分类，输出层有多个神经元）</p></div> <div data-bbox="287 344 1260 668" data-label="List-Group"><ol style="list-style-type: none"><li>1) 基于 TensorFlow 的原生代码搭建神经网络<ol style="list-style-type: none"><li>1. 实验环境</li><li>2. 数据集介绍</li><li>3. 数据集准备和预处理，python 文件放在 <code>tests</code> 目录下，命名: <code>test_ NN_iris.py</code></li><li>4. 搭建网络，定义神经网络中的所有可训练参数</li><li>5. 通过训练优化参数，并记录 <code>loss</code> 和 <code>acc</code></li><li>6. <code>acc/loss</code> 可视化</li></ol></li></ol></div> <div data-bbox="287 717 579 750" data-label="Text"><p>补充和提高（选做小节）</p></div> <div data-bbox="287 760 1260 915" data-label="List-Group"><ol style="list-style-type: none"><li>1) 对上面的实验要求使用基于 TensorFlow 中的 <code>keras</code> 框架搭建神经网络的步骤和代码示例。</li><li>2) 用 <code>class</code> 封装神经网络结构，与 <code>Sequential</code> 搭建神经网络的差异代码用黄色标注。代码示例如下：</li></ol></div>
<div data-bbox="189 1197 219 1477" data-label="Text"><p>实 验 结 果</p></div>	<div data-bbox="287 931 1232 1332" data-label="Text"><pre>import numpy as np import matplotlib.pyplot as plt import sklearn.datasets np.random.seed(1) X, Y = sklearn.datasets.make_moons(n_samples=200, noise=.2) X, Y = X.T, Y.reshape(1, Y.shape[0]) m = X.shape[1] dim = X.shape[0] print(X.shape) print(Y.shape)</pre></div> <div data-bbox="287 1342 429 1372" data-label="Text"><p>结果如下图：</p></div> <div data-bbox="414 1387 1129 1617" data-label="Image"></div> <div data-bbox="683 1632 865 1664" data-label="Caption"><p>图 1 X&amp;Y shape</p></div> <div data-bbox="287 1716 700 1748" data-label="Text"><pre>plt.figure(figsize=(8, 6))</pre></div>

```
plt.scatter(X[0, Y[0,:]==0], X[1, Y[0,:]==0], c='r',
            marker='s', label='negative')
plt.scatter(X[0, Y[0,:]==1], X[1, Y[0,:]==1], c='b',
            marker='o', label='postive')
plt.legend(prop={"size":15})
plt.show()
```

结果如下图:

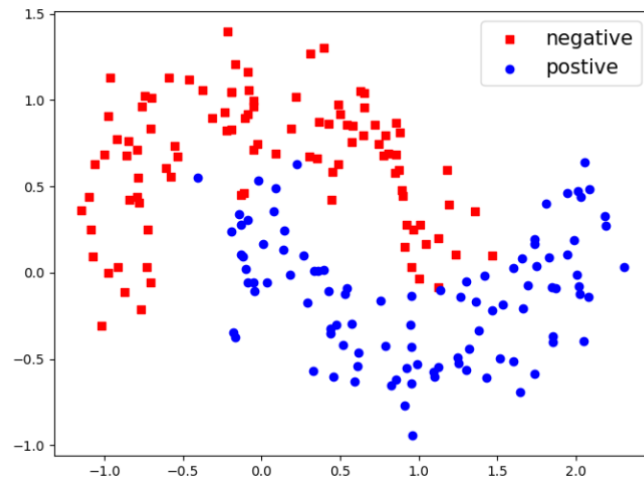


图 2 数据分布散点图

### 神经网络参数初始化

```
def initialize_parameters(n_x, n_h, n_y):
    np.random.seed(0)
    # 设置随机种子
    # 参数初始化
    W1 = np.random.randn(n_h, n_x)
    b1 = np.zeros((n_h, 1))
    W2 = np.random.randn(n_y, n_h)
    b2 = np.zeros((n_y, 1))
    parameters = {
        'W1': W1,
        'b1': b1,
        'W2': W2,
        'b2': b2
    }
```

```
return parameters
```

### 神经网络前向传播

```
def sigmoid(z):  
    a= 1/(1+ np.exp(-z))  
    return a
```

### 定义前向传播函数:

```
def forward_propagation(X, parameters):  
    W1 = parameters['W1']  
    b1 = parameters['b1']  
    W2 = parameters['W2']  
    b2 = parameters['b2']  
    # 输入层 一> 隐藏层  
    Z1 = np.dot(W1, X) + b1  
    A1 = np.tanh(Z1)  
    # 隐藏层 一> 输出层  
    Z2 = np.dot(W2, A1) + b2  
    A2 = sigmoid(Z2)  
    cache = {  
        'Z1': Z1,  
        'A1': A1,  
        'Z2': Z2,  
        'A2': A2  
    }  
    return A2, cache
```

### 计算神经网络的代价函数:交叉熵损失函数 (为什么要用交叉熵?)

```
def compute_loss(A2, Y):  
    m = Y.shape[1]  
    cross_entropy = -(Y * np.log(A2) + (1 - Y) * np.log(1 - A2))  
    cost= 1.0/m * np.sum(cross_entropy)  
    return cost
```

### 神经网络的反向传播

```
def back_propagation(X, Y, parameters, cache):  
    m = X.shape[1]  
    # 神经网络参数
```

```

W1 = parameters['W1']
b1 = parameters['b1']
W2 = parameters['W2']
b2 = parameters['b2']

#中间变量
Z1 = cache['Z1']
A1 = cache['A1']
Z2 = cache['Z2']
A2 = cache['A2']

#计算梯度
dZ2=A2-Y
dW2= 1.0/ m * np.dot(dZ2, A1.T)
db2= 1.0/ m * np.sum(dZ2, axis=1, keepdims=True)
dZ1 = np.dot(W2.T, dZ2) * (1 - np.power(A1, 2))
dW1= 1.0/ m * np.dot(dZ1, X.T)
db1= 1.0/ m * np.sum(dZ1, axis=1, keepdims=True)
grads= {
    'dW1': dW1,
    'db1': db1,
    'dW2': dW2,
    'db2': db2
}

return grads

```

更新参数 **W** 和 **b**

```

def update_parameters(parameters, grads, learning_rate=0.1):
    # 神经网络参数
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']

    # 神经网络参数梯度
    dW1 = grads['dW1']
    db1 = grads['db1']
    dW2 = grads['dW2']
    db2 = grads['db2']

    # 梯度下降算法
    W1 = W1 - learning_rate * dW1

```

```

b1 = b1 - learning_rate * db1
W2 = W2 - learning_rate * dW2
b2 = b2 - learning_rate * db2
parameters = {
    'W1': W1,
    'b1': b1,
    'W2': W2,
    'b2': b2
}
return parameters

```

### 构建神经网络模型并训练

```

def nn_model(X, Y, n_h=3, num_iterations =200,
learning_rate=0.1):
    #定义网络
    n_x = X.shape[0]
    n_y=1
    #参数初始化
    parameters = initialize_parameters(n_x,n_h,n_y) #迭代训练
    for i in range(num_iterations):
        #正向传播
        A2, cache = forward_propagation(X, parameters)

        cost = compute_loss(A2, Y)
        grads = back_propagation(X, Y, parameters, cache)
        # 更新参数
        parameters = update_parameters(parameters, grads,
learning_rate)
        # print
        if (i + 1) % 20 == 0:
            print('Iteration: % d, cost = % f' % (i + 1, cost))
    return parameters

```

### 用神经网络模型进行预测，定义预测函数

```

def predict(X, parameters):
    #神经网络参数
    W1 = parameters['W1']
    b1 = parameters['b1']

```

```

W2 = parameters['W2']
b2 = parameters['b2']
#输入层->隐藏层
Z1 = np.dot(W1,X) + b1
A1 = np.tanh(Z1)
#隐藏层->输出层
Z2 = np.dot(W2,A1) + b2
A2 = sigmoid(Z2)
#预测标签
Y_pred = np.zeros((1, X.shape[1])) #初始化 Y_pred
Y_pred[A2>0.5]=1 #Y_hat 大于 0.5 的预测为正类
return Y_pred

```

利用第 1-第 3 步构建的数据集进行模型的测试。数据集的生成和测试过程写在 testMLP.py 文件中，即把模型部分的代码和测试部分的代码进行解耦操作。

封装代码如下：

```

import numpy as np
class MLP(object):

    def initialize_parameters(self, n_x, n_h, n_y):
        np.random.seed(0)
        # 设置随机种子
        # 参数初始化
        W1 = np.random.randn(n_h, n_x)
        b1 = np.zeros((n_h, 1))
        W2 = np.random.randn(n_y, n_h)
        b2 = np.zeros((n_y, 1))
        parameters = {
            'W1': W1,
            'b1': b1,
            'W2': W2,
            'b2': b2
        }
        return parameters

    def sigmoid(self, z):
        a = 1 / (1 + np.exp(-z))
        return a

```



```

def forward_propagation(self, X, parameters):
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']
    # 输入层 -> 隐藏层
    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    # 隐藏层 -> 输出层
    Z2 = np.dot(W2, A1) + b2
    A2 = self.sigmoid(Z2)
    cache = {
        'Z1': Z1,
        'A1': A1,
        'Z2': Z2,
        'A2': A2
    }
    return A2, cache

def compute_loss(self, A2, Y):
    m = Y.shape[1]
    cross_entropy = -(Y * np.log(A2) + (1 - Y) * np.log(1 - A2))
    cost = 1.0 / m * np.sum(cross_entropy)
    return cost

def back_propagation(self, X, Y, parameters, cache):
    m = X.shape[1]
    # 神经网络参数
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']
    # 中间变量
    Z1 = cache['Z1']
    A1 = cache['A1']
    Z2 = cache['Z2']
    A2 = cache['A2']
    # 计算梯度

```

```

dZ2 = A2 - Y
dW2 = 1.0 / m * np.dot(dZ2, A1.T)
db2 = 1.0 / m * np.sum(dZ2, axis=1, keepdims=True)
dZ1 = np.dot(W2.T, dZ2) * (1 - np.power(A1, 2))
dW1 = 1.0 / m * np.dot(dZ1, X.T)
db1 = 1.0 / m * np.sum(dZ1, axis=1, keepdims=True)
grads = {
    'dW1': dW1,
    'db1': db1,
    'dW2': dW2,
    'db2': db2
}

return grads

def update_parameters(self, parameters, grads,
learning_rate=0.1):
    # 神经网络参数
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']
    # 神经网络参数梯度
    dW1 = grads['dW1']
    db1 = grads['db1']
    dW2 = grads['dW2']
    db2 = grads['db2']
    # 梯度下降算法
    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2
    parameters = {
        'W1': W1,
        'b1': b1,
        'W2': W2,
        'b2': b2
    }

    return parameters

```

```

def nn_model(self, X, Y, n_h=3, num_iterations=200,
learning_rate=0.1):
    # 定义网络
    n_x = X.shape[0]
    n_y = 1
    # 参数初始化
    parameters = self.initialize_parameters(n_x, n_h, n_y) #
迭代训练
    for i in range(num_iterations):
        # 正向传播
        A2, cache = self.forward_propagation(X, parameters)

        cost = self.compute_loss(A2, Y)
        grads = self.back_propagation(X, Y, parameters, cache)
        # 更新参数
        parameters = self.update_parameters(parameters, grads,
learning_rate)
        # print
        if (i + 1) % 20 == 0:
            print('Iteration: % d, cost = % f' % (i + 1, cost))
    return parameters

def predict(self, X, parameters):
    # 神经网络参数
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']
    # 输入层->隐藏层
    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    # 隐藏层->输出层
    Z2 = np.dot(W2, A1) + b2
    A2 = self.sigmoid(Z2)
    # 预测标签
    Y_pred = np.zeros((1, X.shape[1])) # 初始化 Y_pred
    Y_pred[A2 > 0.5] = 1 # Y_hat 大于 0.5 的预测为正类

```

```
return Y_pred
```

结构图如下:

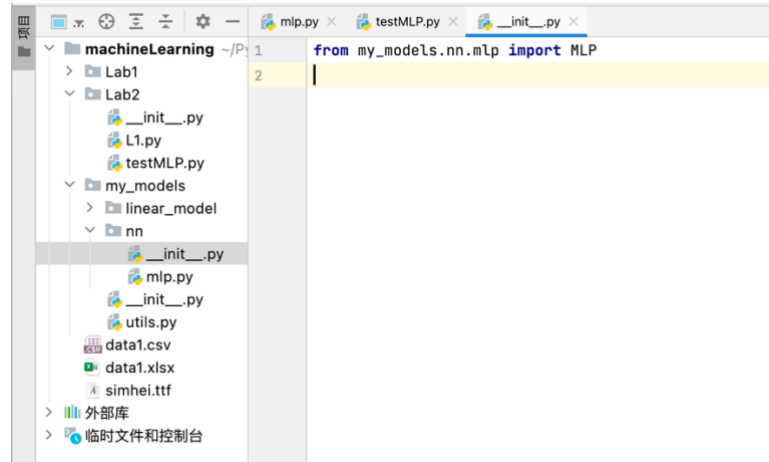


图 3 结构展示

测试过程如下:

1. 首先, 构建 MLP 的对象模型

```
from my_models.nn import MLP
```

#测试

```
mlp = MLP()
```

2. 构建神经网络, 获得训练好的模型参数

```
parameters = mlp.nn_model(X, Y, n_h=3, num_iterations=1500,  
learning_rate=0.2)
```

```
print(parameters)
```

结果如下图:

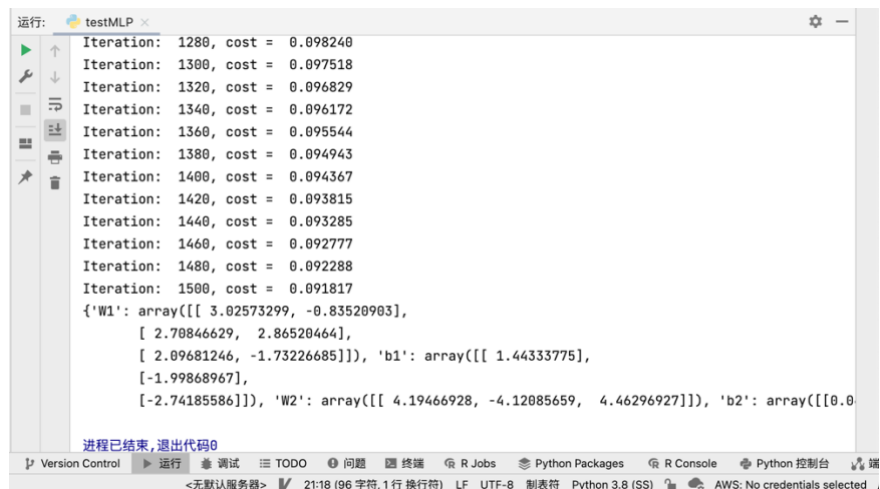
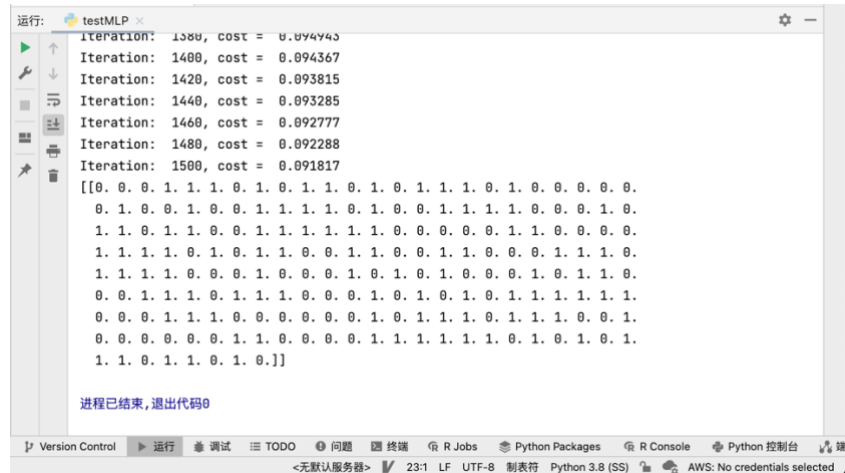


图 4 parameters 结果

### 3. 利用训练好的模型进行预测，打印预测结果

```
Y_pred = mlp.predict(X, parameters)
print(Y_pred)
```

结果如下图：



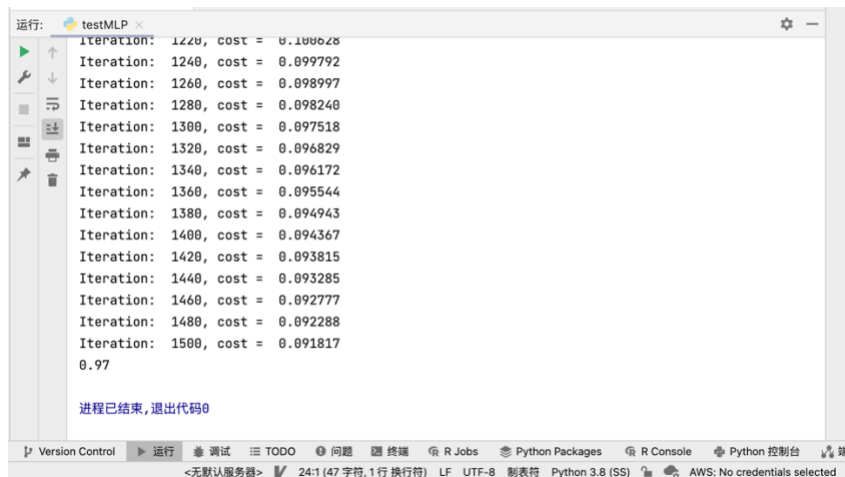
```
运行: testMLP x
Iteration: 1380, cost = 0.094443
Iteration: 1400, cost = 0.094367
Iteration: 1420, cost = 0.093815
Iteration: 1440, cost = 0.093285
Iteration: 1460, cost = 0.092777
Iteration: 1480, cost = 0.092288
Iteration: 1500, cost = 0.091817
[[0. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0.
  0. 1. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 0. 1. 0.
  1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0.
  1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 0.
  1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0.
  0. 0. 1. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1.
  0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1.
  0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1.
  1. 1. 0. 1. 1. 0. 1. 0.]]
进程已结束,退出代码0
```

图 5 Y\_pred 结果

### 4. 利用 np.mean 函数计算预测准确率，并打印预测精度。

```
accuracy = np.mean(Y_pred == Y)
print(accuracy)
```

结果如下图：



```
运行: testMLP x
Iteration: 1220, cost = 0.100628
Iteration: 1240, cost = 0.099792
Iteration: 1260, cost = 0.098997
Iteration: 1280, cost = 0.098240
Iteration: 1300, cost = 0.097518
Iteration: 1320, cost = 0.096829
Iteration: 1340, cost = 0.096172
Iteration: 1360, cost = 0.095544
Iteration: 1380, cost = 0.094943
Iteration: 1400, cost = 0.094367
Iteration: 1420, cost = 0.093815
Iteration: 1440, cost = 0.093285
Iteration: 1460, cost = 0.092777
Iteration: 1480, cost = 0.092288
Iteration: 1500, cost = 0.091817
0.97
进程已结束,退出代码0
```

图 6 accuracy 数值

5. 绘制分类直线，可视化分类效果。

```
from matplotlib.colors import ListedColormap

x_min, x_max = X[0, :].min() - 0.5, X[0, :].max() + 0.5
y_min, y_max = X[1, :].min() - 0.5, X[1, :].max() + 0.5
step = 0.001
xx, yy = np.meshgrid(np.arange(x_min, x_max, step),
np.arange(y_min, y_max, step))
Z = predict(np.c_[xx.ravel(), yy.ravel()].T, parameters)
Z = Z.reshape(xx.shape)
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)

# 绘制边界
plt.scatter(X[0, Y[0,:]==0], X[1, Y[0,:]==0], c='g',
marker='s', label='negative')
plt.scatter(X[0, Y[0,:]==1], X[1, Y[0,:]==1], c='k',
marker='o', label='positive')
plt.legend(prop={"size":15})
plt.show()
结果如下图:
```

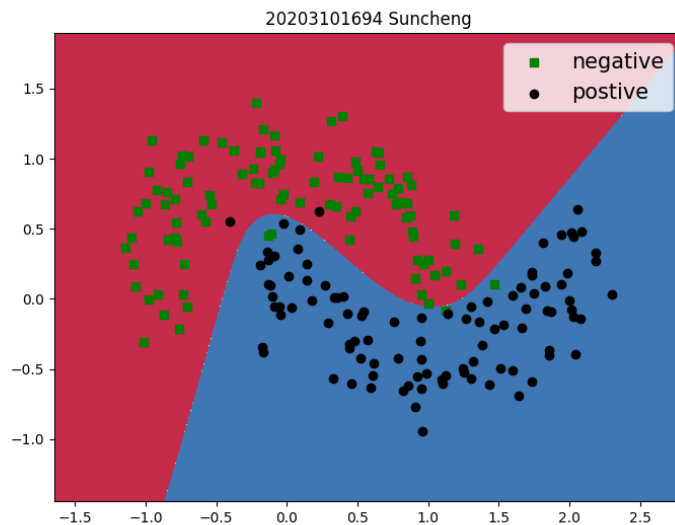


图 7 分类结果图

## 实验 5.2 BP 神经网络的实现（多分类，输出层有多个神经元）

鸢尾花数据集提供了 150 组鸢尾花数据，每组包括鸢尾花的花萼长、花萼宽、花瓣长、花瓣宽 4 个输入特征，同时还给出了这一组特征对应的鸢尾花类别。类别包括狗尾鸢尾、杂色鸢尾、弗吉尼亚鸢尾三类，分别用数字 0、1、2 表示。可在 tests 目录下构建 python 文件 datasets\_iris.py，测试此数据集代码如下：

```
from sklearn import datasets
from pandas import DataFrame
import pandas as pd

x_data = datasets.load_iris().data # .data 返回 iris 数据集所有
输入特征
y_data = datasets.load_iris().target # .target 返回 iris 数据集
所有标签
print("x_data from datasets: \n", x_data)
print("y_data from datasets: \n", y_data)
结果如下图：
```

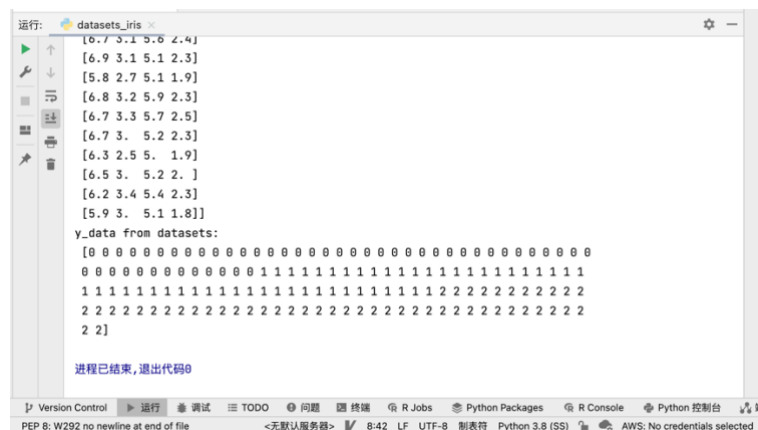


图 8 x&y data

```
x_data = DataFrame(x_data, columns=['花萼长度', '花萼宽度', '花
瓣长度', '花瓣宽度']) # 为表格增加行索引（左侧）和列标签（上方）
pd.set_option('display.unicode.east_asian_width', True) # 设置
列名对齐
print("x_data add index: \n", x_data)
结果如下图：
```

运行: datasets\_iris

/Users/suncheng/opt/anaconda3/envs/ss/bin/python /Users/suncheng/PycharmProjects/machineLearning/

x\_data add index:

	花萼长度	花萼宽度	花瓣长度	花瓣宽度
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

进程已结束,退出代码0

图 9 数据展示

`x_data['类别'] = y_data # 新加一列, 列标签为'类别', 数据为 y_data`  
`print("x_data add a column: \n", x_data)`  
 结果如下图:

运行: datasets\_iris

/Users/suncheng/opt/anaconda3/envs/ss/bin/python /Users/suncheng/PycharmProjects/machineLearning/

x\_data add a column:

	花萼长度	花萼宽度	花瓣长度	花瓣宽度	类别
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

[150 rows x 5 columns]

进程已结束,退出代码0

图 10 数据分类

数据集准备和预处理, python 文件放在 tests 目录下, 命名: test\_NN\_iris.py

```
import tensorflow as tf
from sklearn import datasets
from matplotlib import pyplot as plt
import numpy as np
```

```
x_data = datasets.load_iris().data
y_data = datasets.load_iris().target
```

```
np.random.seed(116) # 使用相同的 seed, 保证输入特征和标签一一对应
np.random.shuffle(x_data)
```



```

np.random.seed(116)
np.random.shuffle(y_data)
tf.random.set_seed(116)

x_train = x_data[:-30]
y_train = y_data[:-30]
x_test = x_data[-30:]
y_test = y_data[-30:]

x_train = tf.cast(x_train, tf.float32)
x_test = tf.cast(x_test, tf.float32)

train_db = tf.data.Dataset.from_tensor_slices((x_train,
y_train)).batch(32)
test_db = tf.data.Dataset.from_tensor_slices((x_test,
y_test)).batch(32)

print(test_db)

```

结果如下图:

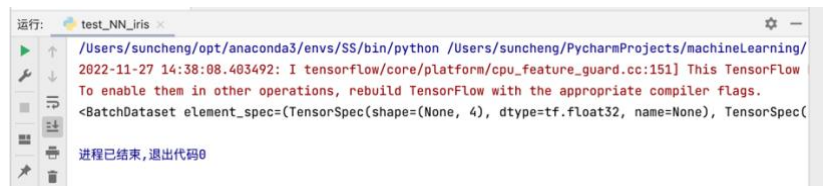


图 11 结果图

```

w1 = tf.Variable(tf.random.truncated_normal([4, 3],
stddev=0.1, seed=1))
b1 = tf.Variable(tf.random.truncated_normal([3], stddev=0.1,
seed=1))
lr = 0.01 # 学习率为 0.01
train_loss_results = []
test_acc = [] # 将每轮的 acc 记录在此列表中, 为后续画 acc 曲线提供数据
epoch = 5000 # 循环 5000 轮
loss_all = 0
for epoch in range(epoch):
    for step, (x_train, y_train) in enumerate(train_db):
        with tf.GradientTape() as tape:

```

```

y = tf.matmul(x_train, w1) + b1
y = tf.nn.softmax(y)
y_ = tf.one_hot(y_train, depth=3)
loss = tf.reduce_mean(tf.square(y_ - y))
loss_all += loss.numpy()

grads = tape.gradient(loss, [w1, b1])
w1.assign_sub(lr * grads[0])
b1.assign_sub(lr * grads[1])
print("Epoch {}, loss: {}".format(epoch, loss_all / 4))
train_loss_results.append(loss_all / 4)
loss_all=0
total_correct, total_number = 0, 0
for x_test, y_test in test_db:
    y = tf.matmul(x_test, w1) + b1
    y = tf.nn.softmax(y)
    pred = tf.argmax(y, axis=1)
    pred = tf.cast(pred, dtype=y_test.dtype)
    correct = tf.cast(tf.equal(pred, y_test), dtype=tf.int32)
    correct = tf.reduce_sum(correct)
    total_correct += int(correct)
    total_number += x_test.shape[0]
acc = total_correct / total_number
test_acc.append(acc)
print("Test_acc:", acc)
print("-----")

```

结果如下图:

```

运行: test_NN_iris
Epoch 495, loss: 27.619527036324143
Test_acc: 1.0
-----
Epoch 496, loss: 27.65190559066832
Test_acc: 1.0
-----
Epoch 497, loss: 27.684257954824716
Test_acc: 1.0
-----
Epoch 498, loss: 27.71658421913162
Test_acc: 1.0
-----
Epoch 499, loss: 27.748884493019432
Test_acc: 1.0
-----
进程已结束,退出代码0

```

图 12 acc 数值

```
plt.title('Loss Function Curve')
plt.xlabel('Epoch')
# x 轴变量名称
plt.ylabel('Loss')
# y 轴变量名称
plt.plot(train_loss_results, label="$Loss$")
# 逐点画出
plt.legend()
# 画出曲线图标
plt.show()
# 画出图像
```

结果如下图:

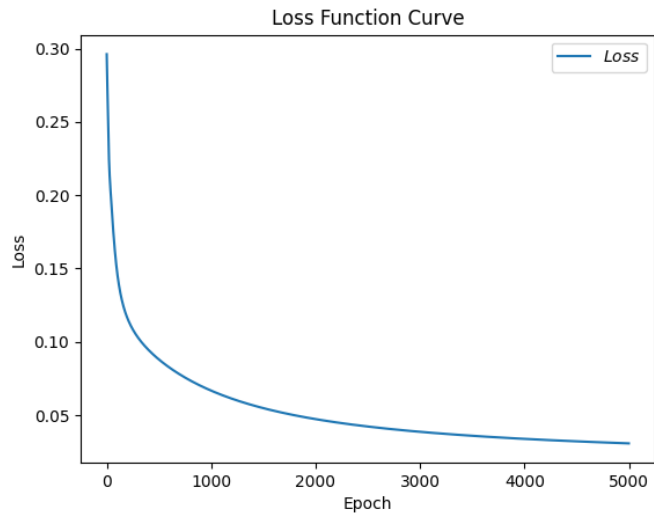


图 13 结果图

```
plt.title('Acc Curve') # 图片标题
plt.xlabel('Epoch') # x 轴变量名称
plt.ylabel('Acc') # y 轴变量名称
plt.plot(test_acc, label="$Accuracy$")
plt.legend()
plt.show()
```

结果如下图:

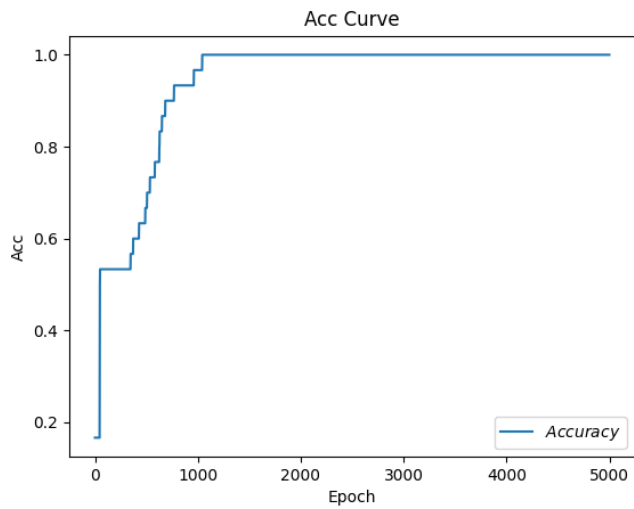


图 14 Acc Curve

### 补充和提高

```
import tensorflow as tf
from sklearn import datasets
import numpy as np

x_train = datasets.load_iris().data
y_train = datasets.load_iris().target

np.random.seed(116)
np.random.shuffle(x_train)
np.random.seed(116)
np.random.shuffle(y_train)
tf.random.set_seed(116)

model = tf.keras.models.Sequential([tf.keras.layers.Dense(3,
activation='softmax',

kernel_regularizer=tf.keras.regularizers.l2())])
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False) ,
```

```

        metrics=['sparse_categorical_accuracy'])
model.fit(x_train, y_train, batch_size=32, epochs=500,
validation_split=0.2, validation_freq=20)

```

```
model.summary()
```

结果如下图:

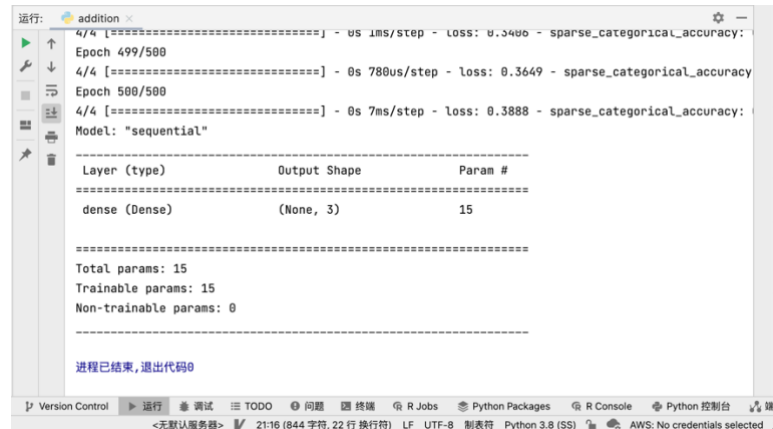


图 15 网络结构和参数设计

```

class IrisModel():
    def __init__(self):

        super(IrisModel, self).__init__()
        self.d1 = Dense(3, activation='softmax',

kernel_regularizer=tf.keras.regularizers.l2())
    def call(self, x):

        y = self.d1(x) #调用网络结构块实现前向传播
        return y

model = IrisModel()
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False) ,

        metrics=['sparse_categorical_accuracy'])
model.fit(x_train, y_train, batch_size=32,

```

`epochs=500, validation_split=0.2, validation_freq=20)`  
`model.summary()`  
结果如下图:

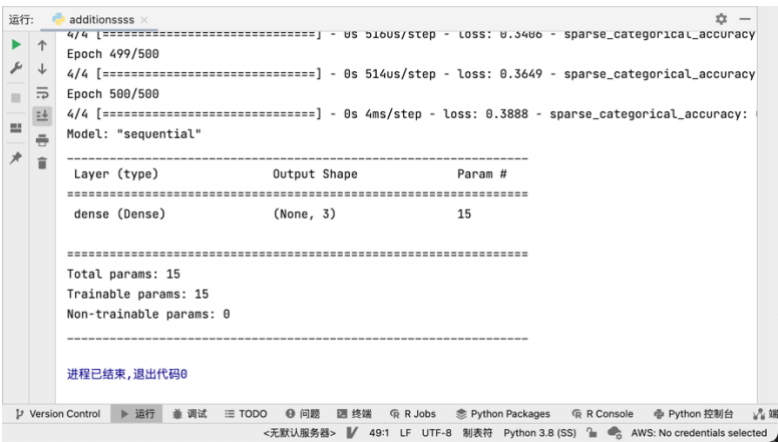


图 16 summary

心得  
体会

本次试验我受益匪浅，其中最大的收获是对于 `class` 类的学习，明白在同一个 `class` 类中定义的函数调用其他函数，需要在函数名前加 ‘`self.`’ 一开始实验好几次都不太行，后面就快放弃时，找到了网上的在同一个 `class` 类中定义的函数调用其他函数的演示代码。除此之外我还学会了使用 `model`，以及调用其他自定义的 `model`，并且能够改变其文件位置，知道了 `__init__` 里面是放什么东西。本次试验是验证性试验，但是还是有难度的，希望早日消化其中的代码。