



南京理工大学
NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

《认知科学大作业》

BP 神经网络编程实现

任课教师:

刘嘉

学院:

计算机科学与工程学院

专业:

智能科学与技术

学生信息:

班号:

9191069501

学号:

919106840208

姓名:

山杜哈西·土鲁四拜克

2022 年 4 月

目录

一、 问题描述	1
1. 问题简述	1
2. 数据集描述	1
二、 方法分析	1
1. 网络结构	1
2. 目标函数	2
3. 正向传播	2
4. 计算各层误差	2
5. 计算梯度	3
6. 更新梯度	3
三、 实验过程	3
1. 流程图	4
2. 关键函数	4
四、 实验结果与分析	5
1. 不同节点数(隐藏层层数为 1, 学习率为 0.01, 迭代数为 1000)	6
2. 不同学习率(隐藏层层数为 1, 节点数为 6, 迭代数为 1000)	7
3. 不同隐藏层数(学习率为 0.01, 节点数为 5, 迭代数为 1000)	7
五、 实验拓展——尝试运用 libsvm 工具处理分类问题对比实验结果	7
1.libsvm 简介	7
2.实验过程	7

一、问题描述

1. 问题简述

利用 BP 神经网络对 iris 鸢尾花数据集进行分类。

2. 数据集描述

iris 鸢尾花数据集是很常用的一个数据集。鸢尾花有三个亚属，分别是山鸢尾（*Iris-setosa*）、变色鸢尾（*Iris-versicolor*）和维吉尼亚鸢尾（*Iris-virginica*）。

该数据集维度为 150×5 ，包含 4 个特征变量，1 个类别变量。共有 150 个样本，iris 是鸢尾植物，这里存储了其萼片和花瓣的长宽，共 4 个属性，鸢尾植物分三类。

◆ SepalLengthCm ◆	◆ SepalWidthCm ◆	◆ PetalLengthCm ◆	◆ PetalWidthCm ◆	◆ Species ◆
0	5.1	3.5	1.4	0.2 Iris-setosa
1	4.9	3.0	1.4	0.2 Iris-setosa
2	4.7	3.2	1.3	0.2 Iris-setosa
3	4.6	3.1	1.5	0.2 Iris-setosa
4	5.0	3.6	1.4	0.2 Iris-setosa

图 1iris 数据集部分数据

二、方法分析

1. 网络结构

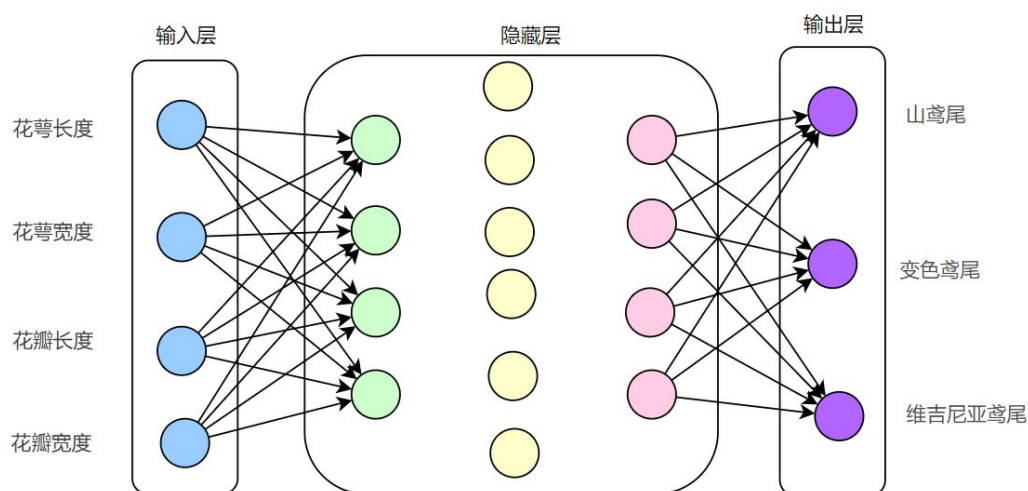


图 2 网络结构设计

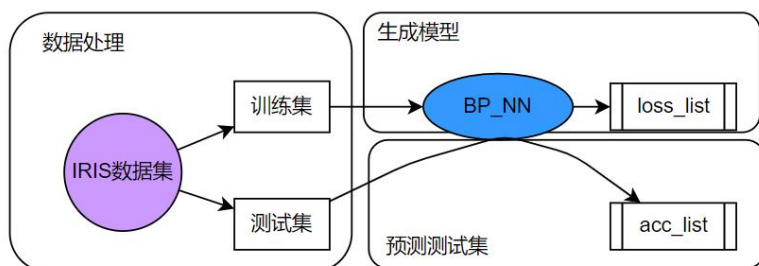


图 3 程序结构

2. 目标函数

$$E = \frac{1}{2} \sum (y - a[-1])^2$$

3. 正向传播

表 1 正向传播过程

$z[0] = x \cdot w[0]$	$a[0] = \delta(z[0])$
$z[1] = a[0] \cdot w[1]$	$a[1] = \delta(z[1])$
.....
$z[-1] = a[-2] \cdot w[-1]$	$a[-1] = \delta(z[-1])$

4. 计算各层误差

表 2 各层误差计算

$$\begin{aligned}
 errors[0] &= (y - a[-1]) \cdot a[-1] \cdot (1 - a[-1]) \\
 errors[1] &= errors[0] \cdot w[-1]^T \cdot a[-2] \cdot (1 - a[-2]) \\
 &\dots\dots \\
 errors[-1] &= errors[-2] \cdot w[1]^T \cdot a[0] \cdot (1 - a[0])
 \end{aligned}$$

5. 计算梯度

表 3 对各边的的权重求梯度

$$\begin{aligned}
 \frac{\partial E}{\partial w[-1]} &= \frac{\partial E}{\partial a[-1]} \cdot \frac{\partial a[-1]}{\partial z[-1]} \cdot \frac{\partial z[-1]}{\partial w[-1]} = (y - a[-1]) \cdot a[-1] \cdot (1 - a[-1]) \cdot a[-2] = errors[0] \cdot a[-2] \\
 \frac{\partial E}{\partial w[-2]} &= \frac{\partial E}{\partial a[-1]} \cdot \frac{\partial a[-1]}{\partial z[-1]} \cdot \frac{\partial a[-2]}{\partial z[-2]} \cdot \frac{\partial z[-2]}{\partial w[-2]} = errors[1] \cdot a[-3] \\
 &\dots\dots \\
 \frac{\partial E}{\partial w[1]} &= \frac{\partial E}{\partial a[-1]} \cdot \frac{\partial a[-1]}{\partial z[-1]} \cdot \frac{\partial a[-2]}{\partial z[-2]} \cdot \frac{\partial z[-2]}{\partial w[-2]} \cdot \dots \cdot \frac{\partial z[1]}{\partial w[1]} = errors[-2] \cdot a[0] \\
 \frac{\partial E}{\partial w[0]} &= \frac{\partial E}{\partial a[-1]} \cdot \frac{\partial a[-1]}{\partial z[-1]} \cdot \frac{\partial a[-2]}{\partial z[-2]} \cdot \frac{\partial z[-2]}{\partial w[-2]} \cdot \dots \cdot \frac{\partial z[0]}{\partial w[0]} = errors[-1] \cdot x
 \end{aligned}$$

6. 更新梯度

表 4 用梯度下降法更新各边权重

$$\begin{aligned}
 w[0]- &= \eta \cdot \frac{\partial E}{\partial w[0]} \\
 w[1]- &= \eta \cdot \frac{\partial E}{\partial w[1]} \\
 &\dots\dots \\
 w[-1]- &= \eta \cdot \frac{\partial E}{\partial w[-1]}
 \end{aligned}$$

三、实验过程

1. 流程图

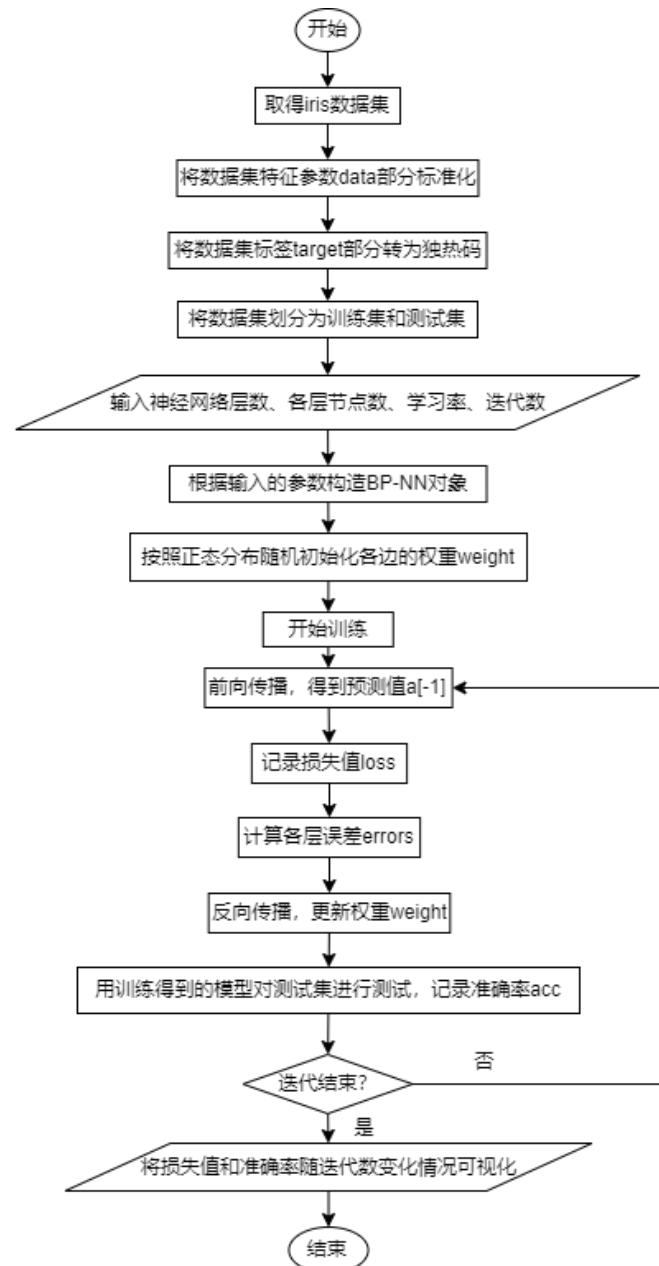


图 4 程序流程图

2. 关键函数

train_predict ()

```

def train_predict(self):
    for k in range(self.epochs):

        # 前向传播
        z = [np.dot(self.x_train, self.weight[0])]
        a = [sigmoid(z[0])]
        for j in range(self.layer_num):
            z.append(np.dot(a[j], self.weight[j + 1]))
            a.append(sigmoid(z[j + 1]))

        # 记录损失值
        self.y_predict_train = a[-1]
        self.loss_list.append(self.loss())

        # 计算各层误差
        errors = [(self.y_train - a[-1]) * a[-1] * a[-1]]
        for j in range(self.layer_num):
            errors.append(np.dot(errors[j], self.weight[-j - 1].T) * a[-j - 2] * (1 - a[-j - 2]))

        # 反向传播，更新权重
        for j in range(self.layer_num):
            self.weight[self.layer_num - j] -= self.lr * np.dot(a[self.layer_num - j - 1].T, -errors[j])
            self.weight[0] -= self.lr * np.dot(self.x_train.T, -errors[-1])

        # 对测试集进行预测
        z = [np.dot(self.x_test, self.weight[0])]
        a = [sigmoid(z[0])]
        for j in range(self.layer_num):
            z.append(np.dot(a[j], self.weight[j + 1]))
            a.append(sigmoid(z[j + 1]))

        # 记录准确率
        self.y_predict_test = encoder.inverse_transform(a[-1])
        self.acc_list.append(self.accuracy())
        print("第", k + 1, "次训练集损失值: ", self.loss(), "对测试集预测准确率: ", self.accuracy())
    print("迭代", self.epochs, "次后对测试集预测准确率达到: ", self.accuracy())

```

图 5 关键函数截图

四、实验结果与分析

0. 简单交互展示

```

import matplotlib.pyplot as plt

matplotlib.rcParams['font.sans-serif'] = ['KaiTi']
matplotlib.rcParams['axes.unicode_minus'] = False

def sigmoid(x):
    return np.exp(x) / np.sum(np.exp(x), axis=1)

class BP_NN:
    def __init__(self, x_train, y_train, x_test, y_test, layer_num, node_nums, lr, epochs):
        self.x_train = x_train
        self.y_train = y_train
        self.x_test = x_test
        self.y_test = encoder.inverse_transform(y_test)

        self.layer_num = layer_num
        self.node_nums = node_nums
        self.lr = lr
        self.epochs = epochs

        self.y_predict_train = None

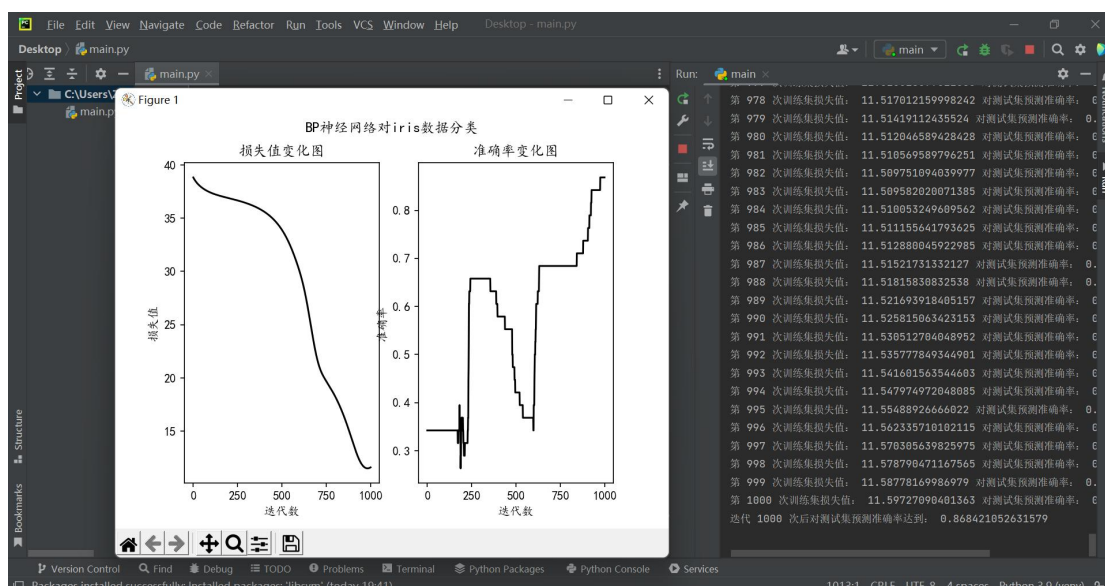
```

Run console output:

```

输入隐藏层数: 2
输入 1 第层节点数: 2
输入 2 第层节点数: 2
输入学习率: 0.01
输入迭代数: 1000
第 1 次训练集损失值: 38.82544117842916 对测试集预测准确率: 0.34210526315789475
第 2 次训练集损失值: 38.79325922696376 对测试集预测准确率: 0.34210526315789475
第 3 次训练集损失值: 38.762033268946524 对测试集预测准确率: 0.34210526315789475
第 4 次训练集损失值: 38.731712873308964 对测试集预测准确率: 0.34210526315789475
第 5 次训练集损失值: 38.70225184772912 对测试集预测准确率: 0.34210526315789475
第 6 次训练集损失值: 38.67360398163514 对测试集预测准确率: 0.34210526315789475
第 7 次训练集损失值: 38.64573045074563 对测试集预测准确率: 0.34210526315789475
第 8 次训练集损失值: 38.61859239336951 对测试集预测准确率: 0.34210526315789475
第 9 次训练集损失值: 38.592153912459914 对测试集预测准确率: 0.34210526315789475
第 10 次训练集损失值: 38.56638149383395 对测试集预测准确率: 0.34210526315789475
第 11 次训练集损失值: 38.54124375907682 对测试集预测准确率: 0.34210526315789475
第 12 次训练集损失值: 38.516711311887455 对测试集预测准确率: 0.34210526315789475
第 13 次训练集损失值: 38.49275656911676 对测试集预测准确率: 0.34210526315789475
第 14 次训练集损失值: 38.46935376610744 对测试集预测准确率: 0.34210526315789475
第 15 次训练集损失值: 38.446478565570494 对测试集预测准确率: 0.34210526315789475

```



1. 不同节点数(隐藏层层数为 1，学习率为 0.01，迭代数为 1000)

编号	节点数	准确率
1	2	0.8421052631578947
2	4	0.6842105263157895
3	6	0.8947368421052632
4	8	0.8157894736842105

分析：当其他条件相同、节点数不同时，模型预测准确率区分并不大。另外由于各边权重初值为随机赋值，故本次探究并不能说明问题。

2. 不同学习率(隐藏层层数为 1, 节点数为 6, 迭代数为 1000)

编号	学习率	准确率
1	0.001	0.39473684210526316
2	0.01	0.8421052631578947
3	0.025	0.8421052631578947
4	0.1	0.7894736842105263

分析：当学习率比较小时，梯度收敛的速度过慢，导致预测准确率不高。

3. 不同隐藏层数(学习率为 0.01, 节点数为 5, 迭代数为 1000)

编号	隐藏层层数	准确率
1	1	1.0
2	2	0.9736842105263158
3	3	0.6578947368421053
4	4	0.2631578947368421

分析：当层数为 3 层或者以上时，准确率较低，猜测可能是由于 iris 数据集样本数较小，更适合使用层数较少的模型。

五、实验拓展——尝试运用 libsvm 工具处理分类问题对比实验结果

1.libsvm 简介

LIBSVM 是台湾大学林智仁(Lin Chih-Jen)教授等开发设计的一个简单、易于使用和快速有效的 SVM 模式识别与回归的软件包，他不但提供了编译好的可在 Windows 系列系统的执行文件，还提供了源代码，方便改进、修改以及在其它操作系统上应用；该软件对 SVM 所涉及的参数调节相对比较少，提供了很多的默认参数，利用这些默认参数可以解决很多问题；并提供了交互检验(Cross Validation)的功能。该软件可以解决 C-SVM、 ν -SVM、 ϵ -SVR 和 ν -SVR 等问题，包括基于一对一算法的多类模式识别问题。

2.实验过程

①导入 libsvm 包

```
from libsvm.svmutil import *
```

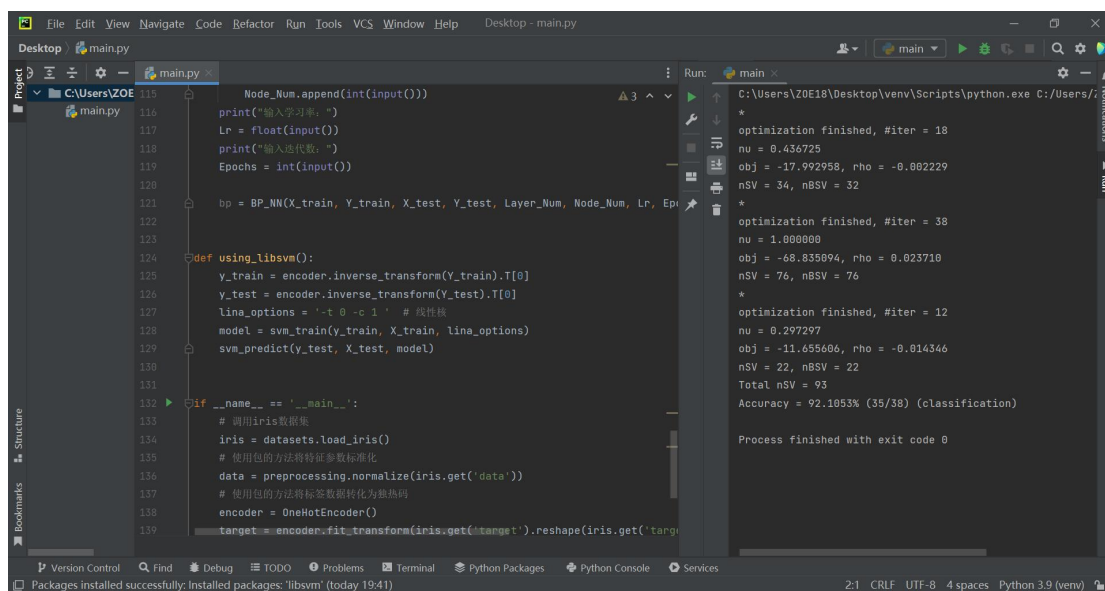
图 6 导入 libsvm 包

②将数据带入函数中

```
def using_libsvm():  
    y_train = encoder.inverse_transform(Y_train).T[0]  
    y_test = encoder.inverse_transform(Y_test).T[0]  
    lina_options = '-t 0 -c 1' # 线性核  
    model = svm_train(y_train, X_train, lina_options)  
    svm_predict(y_test, X_test, model)
```

图 7 运用 libsvm 工具解决分类问题

③输出结果



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help Desktop - main.py  
Desktop main.py  
Project  
C:\Users\ZOE\main.py  
115 Node_Num.append(int(input()))  
116 print("输入学习率:")  
117 Lr = float(input())  
118 print("输入迭代数:")  
119 Epochs = int(input())  
120  
121 bp = BP_NN(X_train, Y_train, X_test, Y_test, Layer_Num, Node_Num, Lr, Epochs)  
122  
123  
124 def using_libsvm():  
125     y_train = encoder.inverse_transform(Y_train).T[0]  
126     y_test = encoder.inverse_transform(Y_test).T[0]  
127     lina_options = '-t 0 -c 1' # 线性核  
128     model = svm_train(y_train, X_train, lina_options)  
129     svm_predict(y_test, X_test, model)  
130  
131  
132 if __name__ == '__main__':  
133     # 调用iris数据集  
134     iris = datasets.load_iris()  
135     # 使用包的方法将特征参数标准化  
136     data = preprocessing.normalize(iris.get('data'))  
137     # 使用包的方法将标签数据转化为独热码  
138     encoder = OneHotEncoder()  
139     target = encoder.fit_transform(iris.get('target')).reshape(iris.get('target').shape)  
140  
Run: main x  
C:\Users\ZOE18\Desktop\venv\Scripts\python.exe C:/Users/ZOE18/Desktop/main.py  
*  
optimization finished, #iter = 18  
nu = 0.436725  
obj = -17.992958, rho = -0.002229  
nSV = 34, nBSV = 32  
*  
optimization finished, #iter = 38  
nu = 1.000000  
obj = -68.835094, rho = 0.023710  
nSV = 76, nBSV = 76  
*  
optimization finished, #iter = 12  
nu = 0.297297  
obj = -11.655606, rho = -0.014346  
nSV = 22, nBSV = 22  
Total nSV = 93  
Accuracy = 92.1053% (35/38) (Classification)  
Process finished with exit code 0  
Packages installed successfully: Installed packages: 'libsvm' (today 19:41)  
2:1 CRLF UTF-8 4 spaces Python 3.9 (venv)
```

图 8 运用 libsvm 工具运行结果