计算机与信息工程学院实验报告（四）

姓名：王赫 学号：1828070097 专业：数据科学与大数据技术(明德计划) 年级：2018级

课程：机器学习与数据挖掘 主讲教师：罗慧敏 辅导教师：\_\_\_\_\_\_\_

实验时间：2020年11月 18日 下午17时至18时，实验地点：606

实验题目： 编程实现误差逆传播算法（BP算法）

实验目的： 掌握误差逆传播算法（BP算法）的工作流程

实验环境（硬件和软件） Anaconda/Jupyter notebook/Pycharm

实验内容：

编码实现标准BP算法和累积BP算法，在西瓜数据集3.0上分别用这两个算法训练一个单隐层网络，并进行比较。

实验步骤：

1. 数学公式推导
2. 编程实现
3. 数据记录
4. 实验总结

实验数据记录：

**一、**已经给定部分代码，补充完整的代码，需要补充代码的地方已经用红色字体标注，包括：

1. 在第（2）部分；

**#补充反向传播代码**

**#补充Loss可视化代码**

2. 在第（3）部分；

**#补充标准BP算法代码**

**#补充Loss可视化代码**

3. 在第（4）部分：

**#补充测试代码，根据当前的x，预测其类别；**

**二、**将补充完整的第（2）（3）（4）部分的代码提交，并提交实验结果；（**也可以自己重写这三部分的代码提交**）

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

seed = 2020

import random

np.random.seed(seed) # Numpy module.

random.seed(seed) # Python random module.

plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False #用来正常显示负号

plt.close('all')

**（1）数据预处理**

def preprocess(data):

#将非数映射数字

for title in data.columns:

if data[title].dtype=='object':

encoder = LabelEncoder()

data[title] = encoder.fit\_transform(data[title])

#去均值和方差归一化

ss = StandardScaler()

X = data.drop('好瓜',axis=1)

Y = data['好瓜']

X = ss.fit\_transform(X)

x,y = np.array(X),np.array(Y).reshape(Y.shape[0],1)

return x,y

#定义Sigmoid

def sigmoid(x):

return 1/(1+np.exp(-x))

#求导

def d\_sigmoid(x):

return x\*(1-x)

**（2）累积BP算法**

def accumulate\_BP(x,y,dim=10,eta=0.8,max\_iter=500):

n\_samples = x.shape[0]

w1 = np.zeros((x.shape[1],dim))

b1 = np.zeros((n\_samples,dim))

w2 = np.zeros((dim,1))

b2 = np.zeros((n\_samples,1))

losslist = []

for ite in range(max\_iter):

##前向传播

u1 = np.dot(x,w1)+b1

out1 = sigmoid(u1)

u2 = np.dot(out1,w2)+b2

out2 = sigmoid(u2)

loss = np.mean(np.square(y - out2))/2

losslist.append(loss)

print('iter:%d loss:%.4f'%(ite,loss))

**#补充反向传播代码**

**g\_i= (out2-y)\*d\_sigmoid(out2)**

**d\_w2 = np.dot(out1.T, g\_i)**

**d\_b2 = g\_i**

**e\_h = np.dot(g\_i, w2.T)\*d\_sigmoid(out1)**

**d\_w1 = np.dot(x.T, e\_h)**

**d\_b1 = e\_h**

##更新

w1 = w1 - eta\*d\_w1

w2 = w2 - eta\*d\_w2

b1 = b1 - eta\*d\_b1

b2 = b2 - eta\*d\_b2

plt.figure()

**##补充Loss可视化代码**

**plt.plot([i for i in range(max\_iter)], losslist)**

plt.show()

return w1,w2,b1,b2

**（3）标准BP算法**

def standard\_BP(x,y,dim=10,eta=0.8,max\_iter=500):

n\_samples = 1

w1 = np.zeros((x.shape[1],dim))

b1 = np.zeros((n\_samples,dim))

w2 = np.zeros((dim,1))

b2 = np.zeros((n\_samples,1))

losslist = []

**#补充标准BP算法代码**

**for i in range(max\_iter):**

**loss\_per\_ite = []**

**for j in range(x.shape[0]):**

**xi, yi = x[j,:], y[j,:]**

**xi, yi = xi.reshape(1,xi.shape[0]),yi.reshape(1,yi.shape[0])**

**# 前向传播**

**u1 = np.dot(xi, w1) + b1**

**out1 = sigmoid(u1)**

**u2 = np.dot(out1, w2) + b2**

**out2 = sigmoid(u2)**

**loss = np.mean(np.square(yi-out2))/2**

**loss\_per\_ite.append(loss)**

**print('iter:%d loss:%.4f'%(i,loss))**

**# BP逆传播算法**

**d\_out2 = -(yi - out2)**

**d\_u2 = d\_out2\*d\_sigmoid(out2)**

**d\_w2 = np.dot(np.transpose(out1),d\_u2)**

**d\_b2 = d\_u2**

**d\_out1 = np.dot(d\_u2,np.transpose(w2))**

**d\_u1 = d\_out1\*d\_sigmoid(out1)**

**d\_w1 = np.dot(np.transpose(xi),d\_u1)**

**d\_b1 = d\_u1**

**# Update**

**w1 = w1 - eta\*d\_w1**

**w2 = w2 - eta\*d\_w2**

**b1 = b1 - eta\*d\_b1**

**b2 = b2 - eta\*d\_b2**

**losslist.append(np.mean(loss\_per\_ite))**

**#补充Loss可视化代码**

**plt.figure()**

**plt.plot([i+1 for i in range(max\_iter)],losslist)**

**plt.show()**

return w1,w2,b1,b2

**（4）测试**

data = pd.read\_table('watermelon30.txt',delimiter=',')

data.drop('编号',axis=1,inplace=True)

x,y = preprocess(data)

dim = 10

# \_,\_,\_,\_ = standard\_BP(x,y,dim)

w1,w2,b1,b2 = accumulate\_BP(x,y,dim)

**#补充测试代码，根据当前的x，预测其类别；**

**u1 = np.dot(x, w1) + b1**

**out1 = sigmoid(u1)**

**u2 = np.dot(out1, w2) + b2**

**out2 = sigmoid(u2)**

**y\_pred = np.round(out2)**

**result = pd.DataFrame(np.hstack((y, y\_pred)), columns=['Real', 'Predict'])**

**result**

**w1, w2, b1, b2 = standard\_BP(x, y, dim, eta=0.8, max\_iter=500)**

**u1 = np.dot(x, w1) + b1**

**out1 = sigmoid(u1)**

**u2 = np.dot(out1, w2) + b2**

**out2 = sigmoid(u2)**

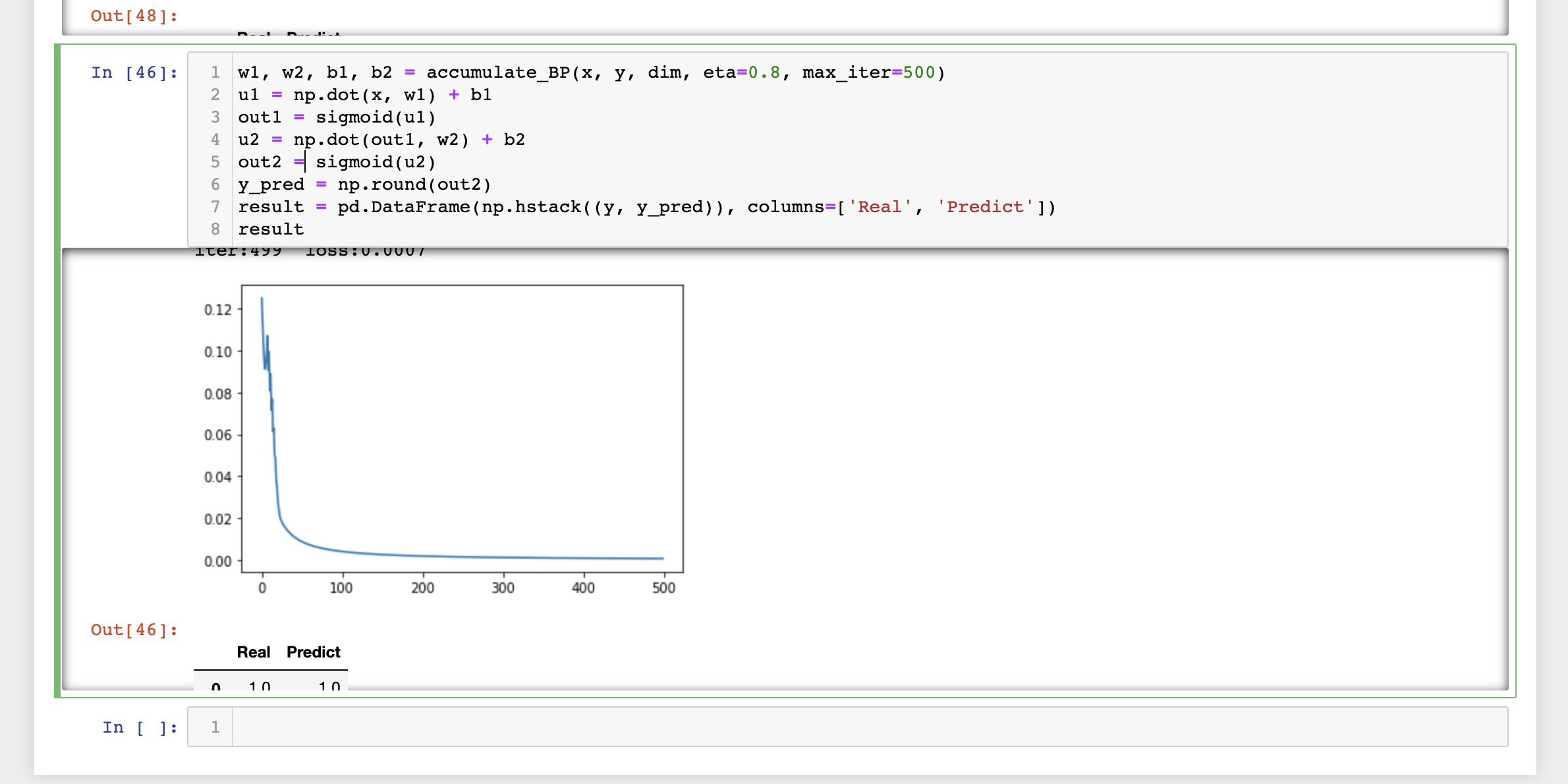
**y\_pred = np.round(out2)**

**result = pd.DataFrame(np.hstack((y, y\_pred)), columns=['Real', 'Predict'])**

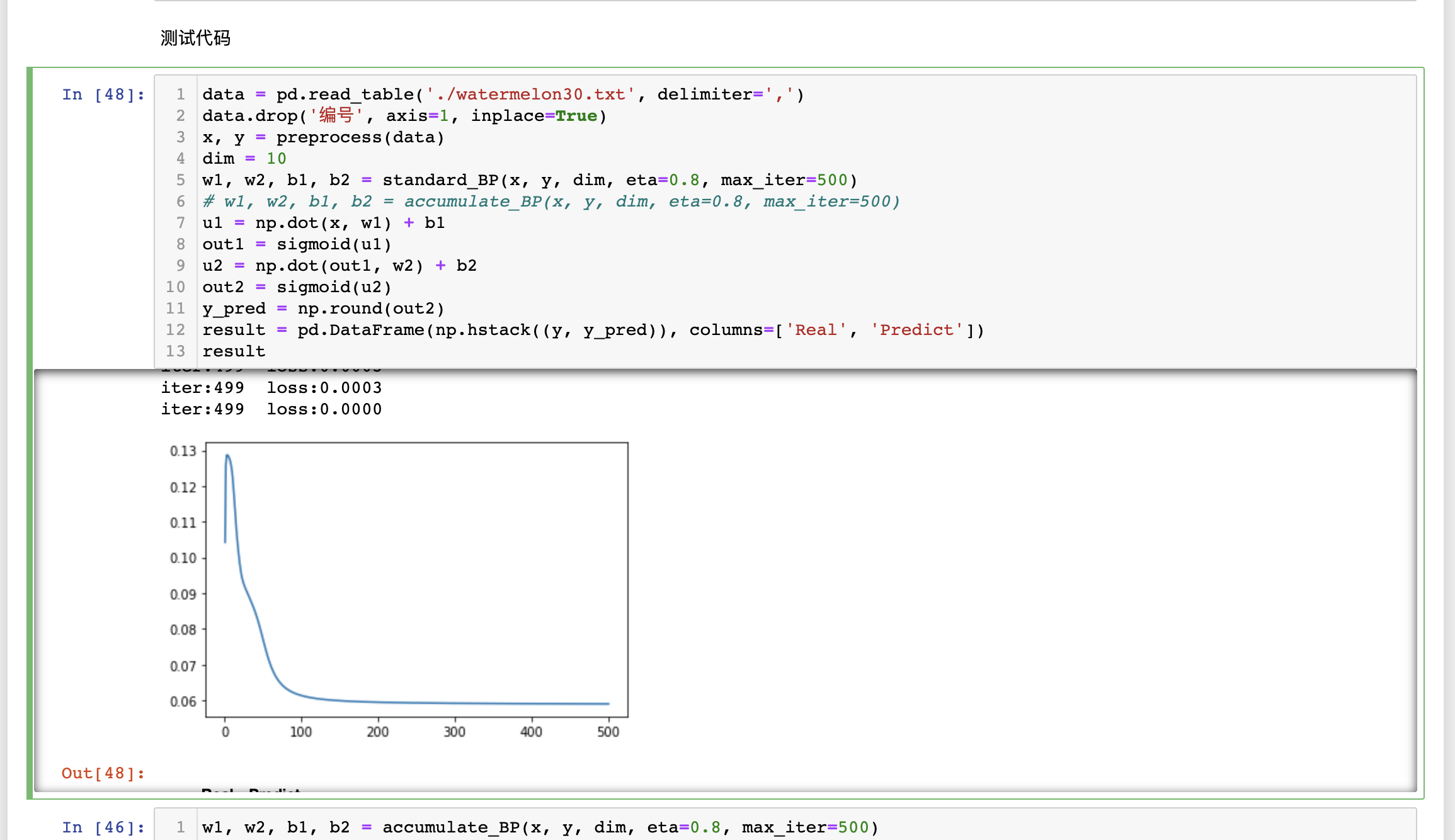
**result**

结果：

累计BP算法：



标准BP算法：



问题讨论：

本次实验主要进行的是BP算法的推导于编程实现。再实现过程中，遇到了一些小问题，在这里记录一下：

1. 再进行矩阵想成运算时，需要注意矩阵维度的变化，否则计算会出现错误。
2. 注意BP算法在学习过程中的参数变化。累积BP算法的参数迭代变化仅依赖于迭代次数；标准BP算法的参数迭代变化需要考虑到每一层的迭代变化，然后取关于层迭代损失的平均值作为外层迭代每一次的损失值。