《机器学习基础》实验报告

实验题目	BP 算法实践		
实验时间	2024/04/21	实验地点	DS3401
实验成绩		实验性质	☐验证性 ☐设计性 ☐综合性

教师评价：

☐算法/实验过程正确； ☐源程序/实验内容提交 ☐程序结构/实验步骤合理；

☐实验结果正确； ☐语法、语义正确； ☐报告规范；

其他：

评价教师签名：

一、实验目的

掌握 BP 算法原理并编程实践。

二、实验项目内容

1. 理解并描述 BP 算法原理。

2. 编程实践，将算法应用于合适的分类数据集（如鸢尾花、UCI 数据集、Kaggle 数据集），要求算法至少用于两个数据集。

三、实验过程或算法（源程序）

1.BP 算法基本原理

BP 算法，即误差逆传播算法，通过不断更新误差更新权重，其大致流程如下：

（1）前向传播

输入数据首先进入输入层；随后，这些数据通过各层的神经元，根据权重和偏置计算出一个输出值，然后作为下一层神经元的输入数据，以此类推，直至数据传递到输出层，完成一次完整的前向传播。

（2）误差计算

在输出层，神经网络产生的预测值与实际目标值进行比较，从而计算出误差。这一误差的大小通常由损失函数来量化，例如均方误差等。

（3）反向传播误差

误差值从输出层开始，沿着神经网络的结构逐层反向传播。利用链式法则，系统能够计算出每一层神经元对最终误差的贡献程度，即所谓的梯度信息。这些梯度信息为接下来的参数更新提供了重要依据。

（4）参数更新

基于反向传播阶段得到的梯度信息，神经网络的权重和偏置值进行更新。这一过程通常使用梯度下降来实现。通过不断迭代，模型参数被逐步调整，以最小化损失函数，从而使神经网络的预测结果更加接近实际目标。

（5）迭代训练

上述的前向传播、误差计算、反向传播和参数更新四个步骤会反复进行，形成一个完整的训练迭代。通过多次迭代，神经网络能够逐渐学习到输入数据与输出目标之间的复杂映射关系，从而不断提升其预测性能。这一迭代训练过程会持续进行，直到满足预设的停止条件（如达到最大迭代次数、损失值低于某个阈值等）。

总之，BP 算法通过不断迭代地前向传播、计算误差、反向传播和更新参数这四个步骤，实现了对人工神经网络的训练和优化。

2.算法源程序

```
from __future__ import division
import math
import random
import pandas as pd
import numpy as np
from sklearn import datasets

flowerLables = {0: 'Iris-setosa',
                1: 'Iris-versicolor',
                2: 'Iris-virginica'}

random.seed(0)

def rand(a, b):
    return (b - a) * random.random() + a

# 生成矩阵
def makeMatrix(I, J, fill=0.0):
    m = []
    for i in range(I):
        m.append([fill] * J)
    return m

# 函数 sigmoid
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

# 函数 sigmoid 的导数
def dsigmoid(x):
    return x * (1 - x)
```

```
31.
32. class BP:
33.     def __init__(self, ni, nh, no):
34.         # 输入层、隐藏层、输出层的节点（数）
35.         self.ni = ni + 1  # 增加一个偏差节点
36.         self.nh = nh + 1
37.         self.no = no
38.
39.         # 输入层ai，隐藏层ah，输出层ao
40.         self.ai = [1.0] * self.ni
41.         self.ah = [1.0] * self.nh
42.         self.ao = [1.0] * self.no
43.
44.         # 建立权重（矩阵）
45.         self.wi = makeMatrix(self.ni, self.nh)
46.         self.wo = makeMatrix(self.nh, self.no)
47.         # 设为随机值
48.         for i in range(self.ni):
49.             for j in range(self.nh):
50.                 self.wi[i][j] = rand(-0.2, 0.2)
51.         for j in range(self.nh):
52.             for k in range(self.no):
53.                 self.wo[j][k] = rand(-2, 2)
54.
55.     def update(self, inputs):
56.         if len(inputs) != self.ni - 1:
57.             raise ValueError('与输入层节点数不符！')
58.
59.         # 激活输入层
60.         for i in range(self.ni - 1):
61.             self.ai[i] = inputs[i]
62.
63.         # 激活隐藏层
64.         for j in range(self.nh):
65.             sum = 0.0
66.             for i in range(self.ni):
67.                 sum = sum + self.ai[i] * self.wi[i][j]
68.             self.ah[j] = sigmoid(sum)
69.
70.         # 激活输出层
71.         for k in range(self.no):
72.             sum = 0.0
73.             for j in range(self.nh):
74.                 sum = sum + self.ah[j] * self.wo[j][k]
75.             self.ao[k] = sigmoid(sum)
76.
77.         return self.ao[:]
```

```
78.
79.     def backPropagate(self, targets, lr):
80.         """ 反向传播 """
81.
82.         # 计算输出层的误差
83.         output_deltas = [0.0] * self.no
84.         for k in range(self.no):
85.             error = targets[k] - self.ao[k]
86.             output_deltas[k] = dsigmoid(self.ao[k]) * e
    rror
87.
88.         # 计算隐藏层的误差
89.         hidden_deltas = [0.0] * self.nh
90.         for j in range(self.nh):
91.             error = 0.0
92.             for k in range(self.no):
93.                 error = error + output_deltas[k] * self
    .wo[j][k]
94.             hidden_deltas[j] = dsigmoid(self.ah[j]) * e
    rror
95.
96.         # 更新输出层权重
97.         for j in range(self.nh):
98.             for k in range(self.no):
99.                 change = output_deltas[k] * self.ah[j]
100.                  self.wo[j][k] = self.wo[j][k] + lr *
     change
101.
102.            # 更新输入层权重
103.            for i in range(self.ni):
104.                for j in range(self.nh):
105.                    change = hidden_deltas[j] * self.ai[
    i]
106.                    self.wi[i][j] = self.wi[i][j] + lr *
     change
107.
108.            # 计算误差
109.            error = 0.0
110.            error += 0.5 * (targets[k] - self.ao[k]) **
    2
111.            return error
112.
113.        def weights(self):
114.            print('输入层权重:')
115.            for i in range(self.ni):
116.                tempa=['%.4f'% x for x in self.wi[i]]
117.                print(tempa)
118.            # print()
```

```
119.             print('\n输出层权重:')
120.             for j in range(self.nh):
121.                 tempa = ['%.4f'% x for x in self.wo[j]]
122.                 print(tempa)
123.
124.
125.         def train(self, patterns, iterations=1000, lr=0.
    1):
126.             for i in range(iterations):
127.                 error = 0.0
128.                 for p in patterns:
129.                     inputs = p[0]
130.                     targets = p[1]
131.                     self.update(inputs)
132.                     error = error + self.backPropagate(t
    argets, lr)
133.                 if i % (iterations/10) == 0:
134.                     print('error_update: %-.9f' % error)
135.
136.         def testflower(self, patterns):
137.             count = 0
138.             for p in patterns:
139.                 target = flowerLables[(p[1].index(1))]
140.                 result = self.update(p[0])
141.                 index = result.index(max(result))
142.                 print(p[0], ':', target, '->', flowerLab
    les[index])
143.                 count += (target == flowerLables[index])
144.             accuracy = float(count / len(patterns))
145.             print('accuracy: %-.9f' % accuracy)
146.
147.         def testdigit(self, patterns):
148.             count = 0
149.             for p in patterns:
150.                 target = (p[1].index(1))
151.                 result = self.update(p[0])
152.                 index = result.index(max(result))
153.                 #print(target, '->', index)
154.                 count += (target == index)
155.             accuracy = float(count / len(patterns))
156.             print('accuracy: %-.9f' % accuracy)
157.
158.  def Iris():
159.         data = []
160.         # 读取数据
161.         raw = pd.read_csv('D:\zjw\demo\machine learning\
    Iris-data.csv')
162.         raw_data = raw.values
```

```
    raw_feature = raw_data[0:, 1:4]
    # 数据处理
    for i in range(len(raw_feature)):
        ele = []
        ele.append(list(raw_feature[i]))
        if raw_data[i][5] == 'Iris-setosa':
            ele.append([1, 0, 0])
        elif raw_data[i][5] == 'Iris-versicolor':
            ele.append([0, 1, 0])
        else:
            ele.append([0, 0, 1])
        data.append(ele)
    # 随机排列数据
    random.shuffle(data)
    training = data[0:100]
    test = data[101:]
    bp1 = BP(3, 3, 3)
    bp1.train(training, iterations=100,lr=0.1)
    bp1.testflower(test)
    bp1.weights()

def digit():
    data=[]
    digits = datasets.load_digits()
    dataread = digits.data
    featureread = digits.target
    for i in range(len(featureread)):
        ele=[]
        ele.append(dataread[i])
        zerovector = [0]*10
        zerovector[featureread[i]] = 1
        ele.append(zerovector)
        data.append(ele)
    random.shuffle(data)
    training = data[0:400]
    test = data[400:]
    bp1 = BP(64, 20, 10)
    bp1.train(training, iterations=500,lr=0.2)
    bp1.testdigit(test)
    # bp1.weights()


if __name__ == '__main__':
    Iris()
    digit()
```

四、实验结果及分析

1.代码运行结果

error_update: 11.844940385
error_update: 5.082415524
error_update: 4.004920564
error_update: 3.434804846
error_update: 3.043322163
error_update: 2.716492420
error_update: 2.462564079
error_update: 2.293571770
error_update: 2.193639619
error_update: 2.133042122
accuracy: 0.877551020
输入层权重:
['0.0460', '-1.1817', '-0.6238', '3.1835']
['2.2405', '-0.7324', '-0.4156', '1.2816']
['-2.7163', '2.5879', '-0.3312', '-5.3253']
['0.0809', '-0.8792', '-0.2026', '2.1990']

输出层权重:
['1.2302', '-5.6833', '-1.7676']
['-6.0548', '-1.9733', '2.7735']
['-1.1460', '-0.6239', '0.8548']
['1.4856', '3.2548', '-4.7625']

error_update: 16.934190731
error_update: 11.708196048
error_update: 9.919083930
error_update: 9.892529069
error_update: 8.881184402
error_update: 8.394850859
error_update: 7.870831119
error_update: 6.904560641
error_update: 4.412948362
error_update: 4.389518114
accuracy: 0.709577666

2.实验结果分析

第一部分的输出结果为鸢尾花数据集，包含了误差的更新过程、输入输出层的权重，最后的准确率；

第二部分的输出结果为手写体数据集，包含了误差的更新过程，最后的准确率。（由于手写体数据集输入较多，权重数量较大，故不打印显示）

鸢尾花数据集的结果显示，bp 算法有效地计算出了输入输出层的权重，并且较高的正确率体现了较为满意的结果；

手写体数据集的结果显示，bp 算法在层数较多时的学习能力可能相对较低，需要更多的迭代次数和更适合的学习率。