|  |
| --- |
|  |
| **实验报告** |
| **课程名称： 机器学习**  **学　　院： 电子科技大学（深圳）高等研究院**  **专　　业：电子信息**  **指导教师：张栗粽**  **学生姓名： 刘文晨**  **学　　号：202222280328** |
|  |
|  |

**电 子 科 技 大 学**

**实 验 报 告**

# 实验三

## 实验项目名称

利用BP神经网络实现鲍鱼的性别分类

## 二、实验学时：4学时

## 三、实验目的

1. 掌握BP神经网络的基本原理；
2. 了解TensorFlow框架。

## 实验原理

1. BP神经网络

BP（Back Propagation）神经网络是一种按照误差反向传播算法训练的多层前馈网络，也是目前应用最广泛的神经网络模型之一。BP神经网络包含多层神经元，如图1所示。输入层的神经元负责接受外界发来的各种信息，并将信息传递给中间层神经元，中间隐含层神经元负责将接收到的信息进行处理变换，根据需求处理信息，实际应用中可将中间隐含层设置为一层或者多层隐含层结构，并通过最后一层的隐含层将信息传递到输出层。

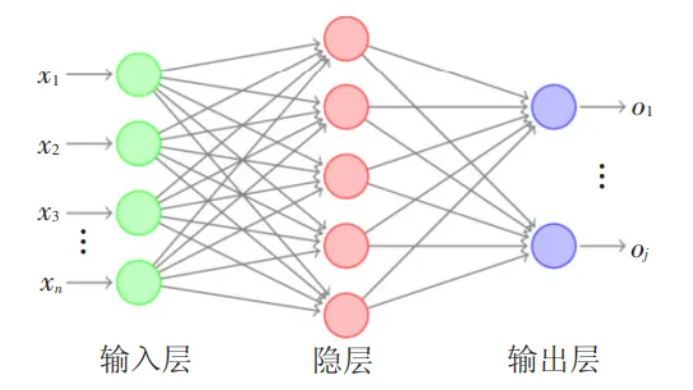


图1 BP神经网络结构

BP神经网络的计算过程由正向计算过程和反向计算过程组成。正向传播过程，输入模式从输入层经隐单元层逐层处理，并转向输出层，每一层神经元的状态只影响下一层神经元的状态。如果在输出层不能得到期望的输出，则转入反向传播，将误差信号沿原来的连接通路返回，通过修改各神经元的权值，使得误差信号最小。当输出的误差减小到期望程度或者预先设定的学习迭代次数时，训练结束，BP神经网络完成学习。

1. TensorFlow框架

TensorFlow是由Google团队开发的深度学习框架之一，它是一个完全基于Python语言设计的开源的软件。TensorFlow的初衷是以最简单的方式实现机器学习和深度学习的概念，它结合了计算代数的优化技术，使它便计算许多数学表达式。

TensorFlow可以训练和运行深度神经网络，它能应用在许多场景下，比如，图像识别、手写数字分类、递归神经网络、单词嵌入、自然语言处理、视频检测等等。TensorFlow可以运行在多个CPU或GPU上，同时它也可以运行在移动端操作系统上（如安卓、IOS 等），它的架构灵活，具有良好的可扩展性，能够支持各种网络模型（如OSI七层和TCP/IP四层）。

TensorFlow这个词由Tensor和Flow两个词组成，这两者是TensorFlow最基础的要素。Tensor代表张量（也就是数据），它的表现形式是一个多维数组；而Flow意味着流动，代表着计算与映射，它用于定义操作中的数据流。

## 实验内容与要求

1. 数据预处理；
2. 构造BP神经网络模型；
3. 训练模型，迭代计算训练损失、测试损失和测试集的准确率；
4. 绘制图像。

## 实验器材（设备、元器件）

处理器：Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz

Python 3.9.0

matplotlib 3.4.0

tensorflow 2.11.0rc1

xlrd 1.2.0

## 实验步骤

1. 数据预处理

给定的训练集和测试集是data文件，先把它们改为xlsx文件。设计一个LoadData函数，输入为abalone\_train.xlsx和abalone\_test.xlsx的文件路径，再调用Python第三方库——xlrd，将当中的数据转换为列表x\_train、y\_train、x\_test、y\_test，最后返回这四个列表。代码如下：

1. **def** LoadData(trainpath, testpath):
2. file\_train\_path = trainpath
3. file\_train\_xlsx = xd.open\_workbook(file\_train\_path)
4. file\_train\_sheet = file\_train\_xlsx.sheet\_by\_name('Sheet1')
5. x\_train = []
6. y\_train = []
7. **for** row **in** range(file\_train\_sheet.nrows):
8. x\_data = []
9. **for** col **in** range(file\_train\_sheet.ncols):
10. **if** col == 0:
11. **if** file\_train\_sheet.cell\_value(row, col) == 'M':
12. y\_train.append(1)
13. **elif** file\_train\_sheet.cell\_value(row, col) == 'F':
14. y\_train.append(-1)
15. **else**:
16. y\_train.append(0)
17. **else**:
18. x\_data.append(file\_train\_sheet.cell\_value(row, col))
20. x\_train.append(list(x\_data))
22. file\_test\_path = testpath
23. file\_test\_xlsx = xd.open\_workbook(file\_test\_path)
24. file\_test\_sheet = file\_test\_xlsx.sheet\_by\_name('Sheet1')
25. x\_test = []
26. y\_test = []
27. **for** row **in** range(file\_test\_sheet.nrows):
28. x\_data = []
29. **for** col **in** range(file\_test\_sheet.ncols):
30. **if** col == 0:
31. **if** file\_test\_sheet.cell\_value(row, col) == 'M':
32. y\_test.append(1)
33. **elif** file\_test\_sheet.cell\_value(row, col) == 'F':
34. y\_test.append(-1)
35. **else**:
36. y\_test.append(0)
37. **else**:
38. x\_data.append(file\_test\_sheet.cell\_value(row, col))
40. x\_test.append(list(x\_data))
42. # print(x\_train)
43. # print(y\_train)
44. # print(x\_test)
45. # print(y\_test)
47. # 将特征值的类型转换为tensor类型，避免后面的矩阵乘法报错
48. x\_train = tf.cast(x\_train, tf.float32)
49. x\_test = tf.cast(x\_test, tf.float32)
51. **return** x\_train, x\_test, y\_train, y\_test
52. 构造BP神经网络模型

在BP.py中构造一个BP神经网络类。类中包含BP神经网络类的构造函数、训练函数和测试函数。构造函数定义了输入层神经元个数、隐藏层神经元个数、输出层层神经元个数和正则化系数等参数，还定义了损失函数,均方误差加入L2正则化。训练函数定义了训练数据集及其标签、测试数据集及其标签、学习率、训练趟数、样本规模等超参数。测试函数用于计算测试集的测试精度。代码如下：

1. **import** numpy as np
2. **import** tensorflow as tf
3. **from** sklearn.metrics **import** accuracy\_score

6. **class** BP(object):
7. **def** \_\_init\_\_(self, input\_n, hidden\_n, output\_n, lambd):
8. """
9. 这是BP神经网络类的构造函数
10. :param input\_n:输入层神经元个数
11. :param hidden\_n: 隐藏层神经元个数
12. :param output\_n: 输出层神经元个数
13. :param lambd: 正则化系数
14. """
15. self.Train\_Data = tf.placeholder(tf.float64, shape=(None, input\_n), name='input\_dataset')  # 训练数据集
16. self.Train\_Label = tf.placeholder(tf.float64, shape=(None, output\_n), name='input\_labels')  # 训练数据集标签
17. self.input\_n = input\_n  # 输入层神经元个数
18. self.hidden\_n = hidden\_n  # 隐含层神经元个数
19. self.output\_n = output\_n  # 输出层神经元个数
20. self.lambd = lambd  # 正则化系数
21. self.input\_weights = tf.Variable(
22. tf.random\_normal((self.input\_n, self.hidden\_n), mean=0, stddev=1, dtype=tf.float64),
23. trainable=True)  # 输入层与隐含层之间的权重
24. self.hidden\_weights = tf.Variable(
25. tf.random\_normal((self.hidden\_n, self.output\_n), mean=0, stddev=1, dtype=tf.float64),
26. trainable=True)  # 隐含层与输出层之间的权重
27. self.hidden\_threshold = tf.Variable(tf.random\_normal((1, self.hidden\_n), mean=0, stddev=1, dtype=tf.float64),
28. trainable=True)  # 隐含层的阈值
29. self.output\_threshold = tf.Variable(tf.random\_normal((1, self.output\_n), mean=0, stddev=1, dtype=tf.float64),
30. trainable=True)  # 输出层的阈值
31. # 将层与层之间的权重与偏置项加入损失集合
32. tf.add\_to\_collection('loss', tf.contrib.layers.l2\_regularizer(self.lambd)(self.input\_weights))
33. tf.add\_to\_collection('loss', tf.contrib.layers.l2\_regularizer(self.lambd)(self.hidden\_weights))
34. tf.add\_to\_collection('loss', tf.contrib.layers.l2\_regularizer(self.lambd)(self.hidden\_threshold))
35. tf.add\_to\_collection('loss', tf.contrib.layers.l2\_regularizer(self.lambd)(self.output\_threshold))
36. # 定义前向传播过程
37. self.hidden\_cells = tf.sigmoid(tf.matmul(self.Train\_Data, self.input\_weights) + self.hidden\_threshold)
38. self.output\_cells = tf.sigmoid(tf.matmul(self.hidden\_cells, self.hidden\_weights) + self.output\_threshold)
39. # 定义损失函数,并加入损失集合
40. self.MSE = tf.reduce\_mean(tf.square(self.output\_cells - self.Train\_Label))
41. tf.add\_to\_collection('loss', self.MSE)
42. # 定义损失函数,均方误差加入L2正则化
43. self.loss = tf.add\_n(tf.get\_collection('loss'))
45. **def** train\_test(self, Train\_Data, Train\_Label, Test\_Data, Test\_Label, learn\_rate, epoch, iteration, batch\_size):
46. """
47. 这是BP神经网络的训练函数
48. :param Train\_Data: 训练数据集
49. :param Train\_Label: 训练数据集标签
50. :param Test\_Data: 测试数据集
51. :param Test\_Label: 测试数据集标签
52. :param learn\_rate:  学习率
53. :param epoch:  时期数
54. :param iteration: 一个epoch的迭代次数
55. :param batch\_size:  小批量样本规模
56. """
57. train\_loss = []  # 训练损失
58. test\_loss = []  # 测试损失
59. test\_accarucy = []  # 测试精度
60. with tf.Session() as sess:
61. datasize = len(Train\_Label)
62. self.train\_step = tf.train.GradientDescentOptimizer(learn\_rate).minimize(self.loss)
63. sess.run(tf.global\_variables\_initializer())
64. **for** e **in** np.arange(epoch):
65. **for** i **in** range(iteration):
66. start = (i \* batch\_size) % datasize
67. end = np.min([start + batch\_size, datasize])
68. sess.run(self.train\_step,
69. feed\_dict={self.Train\_Data: Train\_Data[start:end],
70. self.Train\_Label: Train\_Label[start:end]})
71. **if** i % 10000 == 0:
72. total\_MSE = sess.run(self.MSE,
73. feed\_dict={self.Train\_Data: Train\_Data, self.Train\_Label: Train\_Label})
74. **print**("第%d个epoch中，%d次迭代后，训练MSE为:%g" % (e + 1, i + 10000, total\_MSE))
75. # 训练损失
76. \_train\_loss = sess.run(self.MSE, feed\_dict={self.Train\_Data: Train\_Data, self.Train\_Label: Train\_Label})
77. train\_loss.append(\_train\_loss)
78. # 测试损失
79. \_test\_loss = sess.run(self.MSE, feed\_dict={self.Train\_Data: Test\_Data, self.Train\_Label: Test\_Label})
80. test\_loss.append(\_test\_loss)
81. # 测试精度
82. test\_result = sess.run(self.output\_cells, feed\_dict={self.Train\_Data: Test\_Data})
83. test\_accarucy.append(self.Accuracy(test\_result, Test\_Label))
84. **return** train\_loss, test\_loss, test\_accarucy
86. **def** Accuracy(self, test\_result, test\_label):
87. """
88. 这是BP神经网络的测试函数
89. :param test\_result: 测试集预测结果
90. :param test\_label: 测试集真实标签
91. """
92. predict\_ans = []
93. label = []
94. **for** (test, \_label) **in** zip(test\_result, test\_label):
95. test = np.exp(test)
96. test = test / np.sum(test)
97. predict\_ans.append(np.argmax(test))
98. label.append(np.argmax(\_label))
100. **return** accuracy\_score(label, predict\_ans)
101. 训练模型

在run.py中创建一个主函数run\_main()，首先实现导入数据，设置abalone\_train.xlsx和abalone\_test.xlsx的文件路径，调用LoadData函数得到训练集和测试集的样本与标签。接着设置神经网络参数，学习率为0.01，训练趟数为1000，然后训练并测试网络。代码如下：

1. # 导入数据
2. trainpath = 'data/abalone\_train.xlsx'
3. testpath = 'data/abalone\_test.xlsx'
4. Train\_Data, Test\_Data, Train\_Label, Test\_Label = LoadData(trainpath, testpath)
5. Train\_Data = Normalizer().fit\_transform(Train\_Data)
6. Test\_Data = Normalizer().fit\_transform(Test\_Data)
8. # 设置网络参数
9. input\_n = np.shape(Train\_Data)[1] + np.shape(Test\_Data)[1]
10. output\_n = np.shape(Train\_Label)[1] + np.shape(Test\_Label)[1]
11. hidden\_n = int(np.sqrt(input\_n \* output\_n))
12. lambd = 0.001
13. batch\_size = 64
14. learn\_rate = 0.01
15. epoch = 1000
16. iteration = 10000
18. # 训练并测试网络
19. bp = BP(input\_n, hidden\_n, output\_n, lambd)
20. train\_loss, test\_loss, test\_accuracy = bp.train\_test(Train\_Data, Train\_Label, Test\_Data, Test\_Label, learn\_rate, epoch, iteration, batch\_size)
21. 绘制图像

在run.py的run\_main函数中绘制训练与测试损失和测试集的测试精度2张图像。代码如下：

1. # 解决画图是的中文乱码问题
2. mpl.rcParams['font.sans-serif'] = [u'simHei']
3. mpl.rcParams['axes.unicode\_minus'] = False
5. # 结果可视化
6. col = ['Train\_Loss', 'Test\_Loss']
7. epoch = np.arange(epoch)
8. plt.plot(epoch, train\_loss, 'r')
9. plt.plot(epoch, test\_loss, 'b-.')
10. plt.xlabel('Epoch')
11. plt.ylabel('Loss')
12. plt.grid(True)
13. plt.legend(labels=col, loc='best')
14. plt.savefig('./训练与测试损失.jpg')
15. plt.show()
16. plt.close()
18. plt.plot(epoch, test\_accuracy, 'r')
19. plt.xlabel('Epoch')
20. plt.ylabel('Test Accuracy')
21. plt.grid(True)
22. plt.legend(loc='best')
23. plt.savefig('./测试精度.jpg')
24. plt.show()
25. plt.close()
26. 实验结果

编写好代码后，运行run.py。

训练与测试过程中损失随训练趟数的曲线如图2所示。

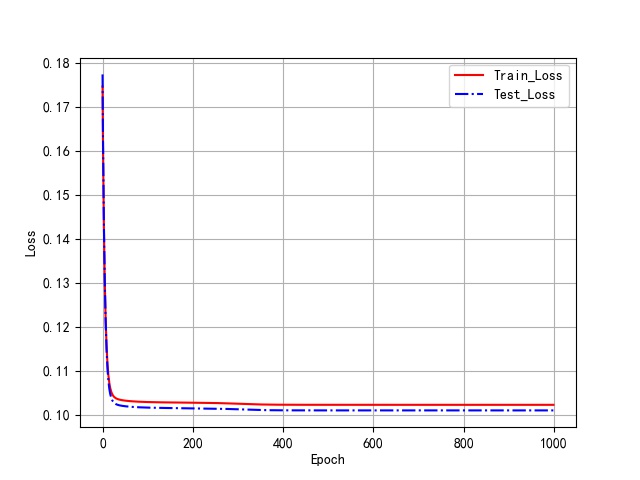


图2 训练与测试损失

测试过程中测试集的准确率随训练趟数增加的折线如图3所示。其中，每一个点的横坐标是训练趟数，纵坐标是该趟时的测试集的准确率。

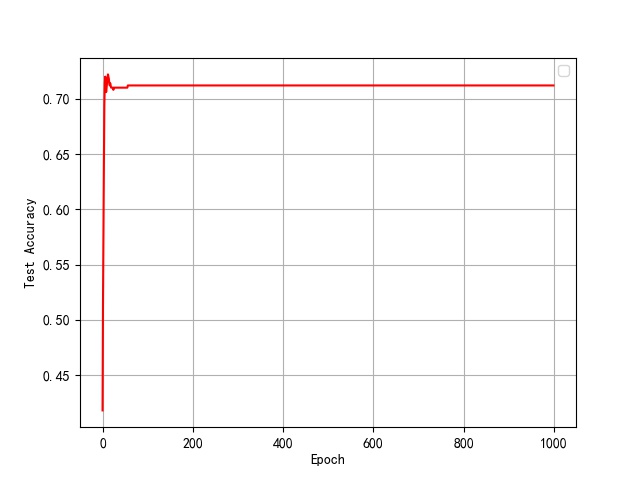


图3 测试集准确率

在本次实验中，预设的学习率为0.01。从图2中我们可以看出，训练与测试损失随迭代趟数的增加而降低；而从图3中我们可以看出，当训练趟数很小时，测试集的准确率随迭代趟数的增加而增加，小幅降低后，当训练趟数超过70轮后，测试集的准确率稳定在72.1%。

## 心得体会

本实验利用BP神经网络实现了鲍鱼的性别分类，使用了TensorFlow框架，用张量表示数据，用计算图搭建神经网络，用会话执行计算图，优化线上的权重（参数），最后得到模型。测试集的准确率为0.7214，达到预期目标。通过此次实验，很好地掌握了BP神经网络前向传播和反向传播的原理，熟悉了Python第三方库——tensorflow、matplotlib和xlrd的使用。