电子科技大学

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

一、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。

四、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。

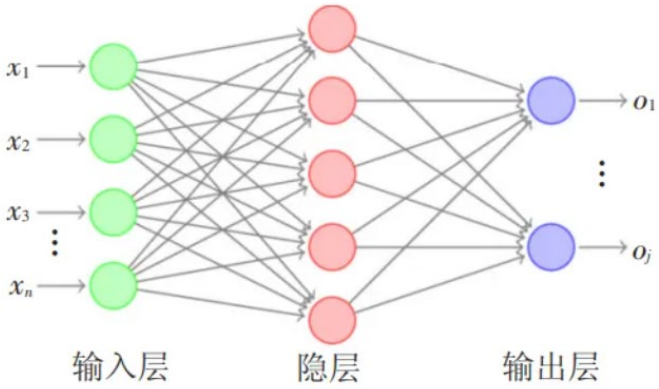

图 1 BP 神经网络结构

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。

2. 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。

五、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。

六、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

七、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：

```
    def LoadData(trainpath, testpath):
        file_train_path = trainpath
        file_train_xlsx = xd.open_workbook(file_train_path)
        file_train_sheet = file_train_xlsx.sheet_by_name('Sheet1')
        x_train = []
        y_train = []
        for row in range(file_train_sheet.nrows):
            x_data = []
            for col in range(file_train_sheet.ncols):
                if col == 0:
                    if file_train_sheet.cell_value(row, col) == 'M':
                        y_train.append(1)
                    elif file_train_sheet.cell_value(row, col) == 'F':
                        y_train.append(-1)
                    else:
                        y_train.append(0)
                else:
                    x_data.append(file_train_sheet.cell_value(row, col))

            x_train.append(list(x_data))

```

```
        file_test_path = testpath
        file_test_xlsx = xd.open_workbook(file_test_path)
        file_test_sheet = file_test_xlsx.sheet_by_name('Sheet1')
        x_test = []
        y_test = []
        for row in range(file_test_sheet.nrows):
            x_data = []
            for col in range(file_test_sheet.ncols):
                if col == 0:
                    if file_test_sheet.cell_value(row, col) == 'M':
                        y_test.append(1)
                    elif file_test_sheet.cell_value(row, col) == 'F':
                        y_test.append(-1)
                    else:
                        y_test.append(0)
                else:
                    x_data.append(file_test_sheet.cell_value(row, col))

            x_test.append(list(x_data))

        # print(x_train)
        # print(y_train)
        # print(x_test)
        # print(y_test)

        # 将特征值的类型转换为 tensor 类型，避免后面的矩阵乘法报错
        x_train = tf.cast(x_train, tf.float32)
        x_test = tf.cast(x_test, tf.float32)

        return x_train, x_test, y_train, y_test
```

2. 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：

```
    import numpy as np
    import tensorflow as tf
    from sklearn.metrics import accuracy_score


    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='inpu
t_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.f
loat64),
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.i
nput_weights))
33.             tf.add_to_collection('loss', tf.contrib.layers.l2_regularizer(self.lambd)(self.h
idden_weights))
34.             tf.add_to_collection('loss', tf.contrib.layers.l2_regularizer(self.lambd)(self.h
idden_threshold))
35.             tf.add_to_collection('loss', tf.contrib.layers.l2_regularizer(self.lambd)(self.o
utput_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'))
44.
45.         def train_test(self, Train_Data, Train_Label, Test_Data, Test_Label, learn_rate, epo
ch, 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(sel
f.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, sel
f.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
```

```
                    _test_loss = sess.run(self.MSE, feed_dict={self.Train_Data: Test_Data, self.Train_Label: Test_Label})
                    test_loss.append(_test_loss)
                    # 测试精度
                    test_result = sess.run(self.output_cells, feed_dict={self.Train_Data: Test_Data})
                    test_accarucy.append(self.Accuracy(test_result, Test_Label))
            return train_loss, test_loss, test_accarucy

        def Accuracy(self, test_result, test_label):
            """
            这是 BP 神经网络的测试函数
            :param test_result: 测试集预测结果
            :param test_label: 测试集真实标签
            """
            predict_ans = []
            label = []
            for (test, _label) in zip(test_result, test_label):
                test = np.exp(test)
                test = test / np.sum(test)
                predict_ans.append(np.argmax(test))
                label.append(np.argmax(_label))

            return accuracy_score(label, predict_ans)
```

3. 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：

```
    # 导入数据
    trainpath = 'data/abalone_train.xlsx'
    testpath = 'data/abalone_test.xlsx'
    Train_Data, Test_Data, Train_Label, Test_Label = LoadData(trainpath, testpath)
    Train_Data = Normalizer().fit_transform(Train_Data)
    Test_Data = Normalizer().fit_transform(Test_Data)

    # 设置网络参数
    input_n = np.shape(Train_Data)[1] + np.shape(Test_Data)[1]
    output_n = np.shape(Train_Label)[1] + np.shape(Test_Label)[1]
    hidden_n = int(np.sqrt(input_n * output_n))
    lambd = 0.001
    batch_size = 64
    learn_rate = 0.01
```

```
    epoch = 1000
    iteration = 10000

    # 训练并测试网络
    bp = BP(input_n, hidden_n, output_n, lambd)
    train_loss, test_loss, test_accuracy = bp.train_test(Train_Data, Train_Label, Test_Data,
Test_Label, learn_rate, epoch, iteration, batch_size)
```

4. 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：

```
    # 解决画图是的中文乱码问题
    mpl.rcParams['font.sans-serif'] = [u'simHei']
    mpl.rcParams['axes.unicode_minus'] = False

    # 结果可视化
    col = ['Train_Loss', 'Test_Loss']
    epoch = np.arange(epoch)
    plt.plot(epoch, train_loss, 'r')
    plt.plot(epoch, test_loss, 'b-.')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid(True)
    plt.legend(labels=col, loc='best')
    plt.savefig('./训练与测试损失.jpg')
    plt.show()
    plt.close()

    plt.plot(epoch, test_accuracy, 'r')
    plt.xlabel('Epoch')
    plt.ylabel('Test Accuracy')
    plt.grid(True)
    plt.legend(loc='best')
    plt.savefig('./测试精度.jpg')
    plt.show()
    plt.close()
```

5. 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。

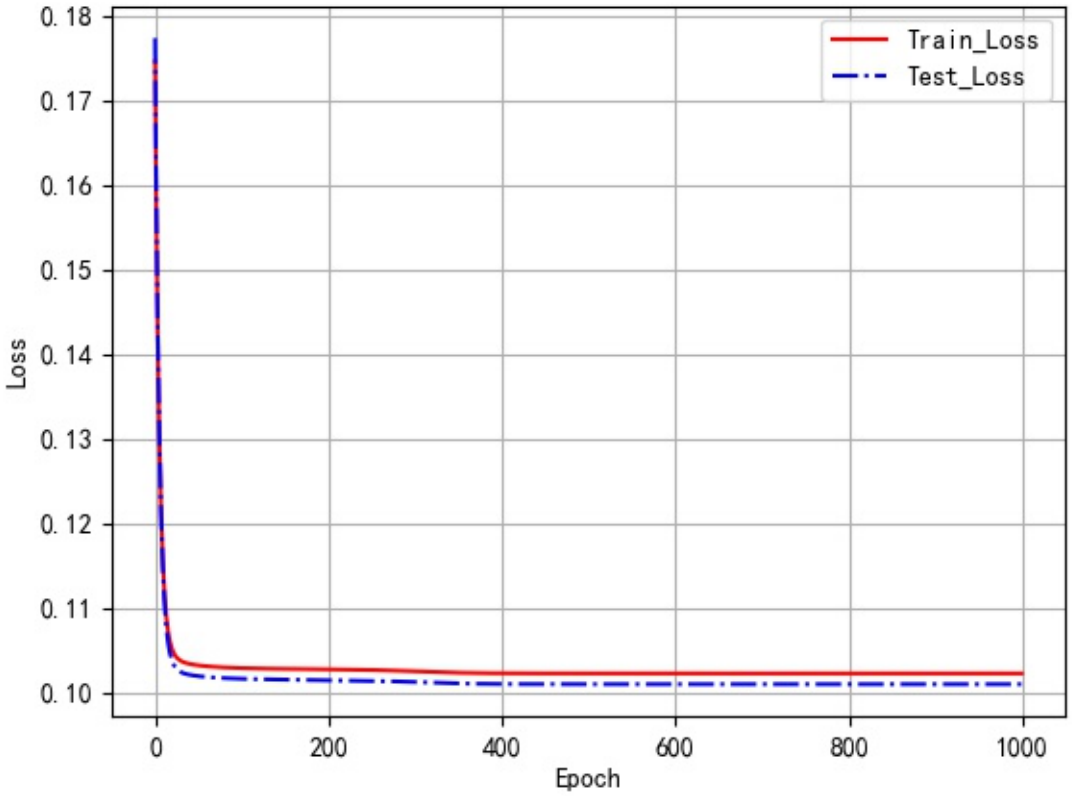

图 2 训练与测试损失

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。

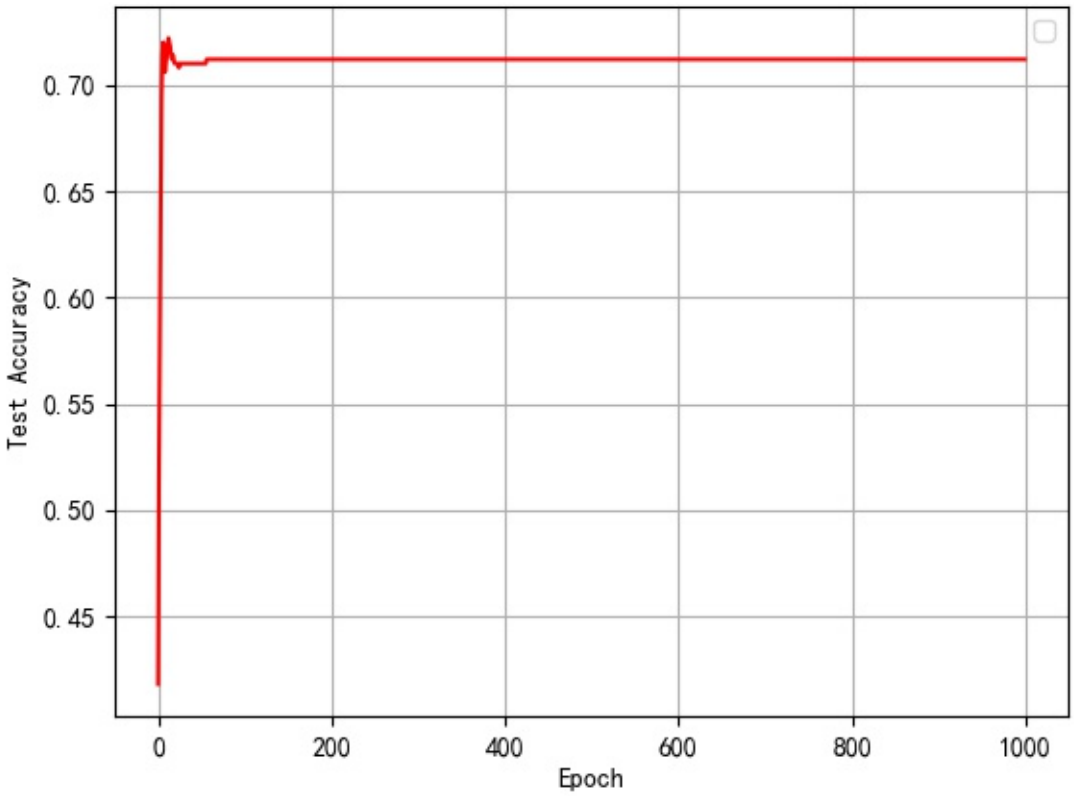

图 3 测试集准确率

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%。

八、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。