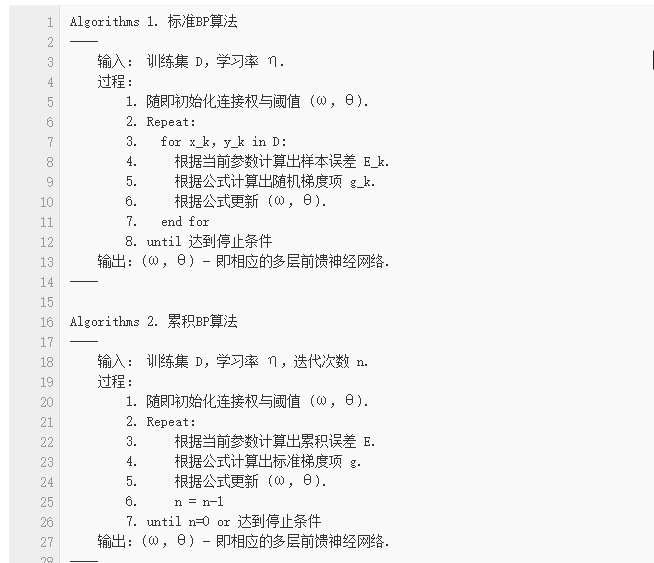
**作业5.5**

1. **实验要求**

试编程实现标准BP算法和累计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。

1. **实验原理**

参考书上推导及算法图5.8，首先给出BP算法的两种版本示意如下：



1. **实验过程与代码：**

这里我使用了机器学习库tensorflow搭建网络对数据进行训练，由于累计BP算法是读取整个训练集D后进行更新，而训练集有十七个，所以BP算法和累计BP算法在实现时的区别只在于每次训练只投入单组数据还是全部组的数据。过程如下：

1. 准备数据，设立标记。
2. 利用tensorflow搭建一个隐层，这里由于有八个属性，所以设置输入的w为8，自己设定中间隐层的权值数目为20，输出很显然为1。设定激活函数为sigmoid函数，优化目标为输出预测值与实际值差的平方和。
3. 设定精度计算函数，即1-误判数 /样本总数。
4. 利用tensorflow会话训练优化目标，得到预测值。
5. 将预测值和标记值代入到精度计算函数得到准确率，同时输出第一次准确率达到90%时的训练次数。

相关代码如下：

**import** tensorflow **as** tf  
**from** numpy **import** \*  
**import** matplotlib.pyplot **as** plt  
*#数据*data = mat([  
 [1, 1, 1, 1, 1, 1, 0.697, 0.460, 1],  
 [2, 1, 2, 1, 1, 1, 0.774, 0.376, 1],  
 [2, 1, 1, 1, 1, 1, 0.634, 0.264, 1],  
 [1, 1, 2, 1, 1, 1, 0.608, 0.318, 1],  
 [3, 1, 1, 1, 1, 1, 0.556, 0.215, 1],  
 [1, 2, 1, 1, 2, 2, 0.403, 0.237, 1],  
 [2, 2, 1, 2, 2, 2, 0.481, 0.149, 1],  
 [2, 2, 1, 1, 2, 1, 0.437, 0.211, 1],  
 [2, 2, 2, 2, 2, 1, 0.666, 0.091, 0],  
 [1, 3, 3, 1, 3, 2, 0.243, 0.267, 0],  
 [3, 3, 3, 3, 3, 1, 0.245, 0.057, 0],  
 [3, 1, 1, 3, 3, 2, 0.343, 0.099, 0],  
 [1, 2, 1, 2, 1, 1, 0.639, 0.161, 0],  
 [3, 2, 2, 2, 1, 1, 0.657, 0.198, 0],  
 [2, 2, 1, 1, 2, 2, 0.360, 0.370, 0],  
 [3, 1, 1, 3, 3, 1, 0.593, 0.042, 0],  
 [1, 1, 2, 2, 2, 1, 0.719, 0.103, 0]])  
x\_data=data[:,0:8].astype(float32)  
y\_data=data[:,8].astype(float32)  
*#搭建网络*x=tf.placeholder(tf.float32,[None,8])  
y=tf.placeholder(tf.float32,[None,1])  
**def** addlayer(data,insize,outsize,active=None):  
 w=tf.Variable(tf.random\_normal([insize,outsize]))  
 b=tf.Variable(tf.zeros([1,outsize])+0.1)  
 w\_b=tf.matmul(data,w)+b  
 **return** active(w\_b)  
layer=addlayer(x,8,20,tf.nn.sigmoid)  
output=addlayer(layer,20,1,tf.nn.sigmoid)  
loss=tf.reduce\_mean(tf.reduce\_sum(tf.square(y-output),reduction\_indices=[1]))  
init=tf.initialize\_all\_variables()  
*#定义精度***def** accuracy(l1,l2):  
 **for** i **in** range(len(l2)):  
 **if** l2[i] > 0.5:  
 l2[i] = 1  
 **else**:  
 l2[i] = 0  
 s = abs(l1 - l2)  
 **return** 1-sum(s) / 17  
*#学习率根据精度动态设计***def** train(l1,l2):  
 **if**(accuracy(l1,l2)<0.7):  
 rate=1-accuracy(l1,l2)  
 **else**:  
 rate=0.45-0.2\*accuracy(l1,l2)  
 **return** tf.train.GradientDescentOptimizer(rate).minimize(loss)  
*#训练网络***with** tf.Session() **as** sess:  
 sess.run(init)  
 **for** i **in** range(2000):  
 l1 = array(y\_data).flatten()  
 l2 = sess.run(output, feed\_dict={x: x\_data, y: y\_data}).flatten()  
 sess.run(train(l1,l2),feed\_dict={x:x\_data[i%17],y:y\_data[i%17]})  
 *#print(array(y\_data).flatten(),sess.run(output,feed\_dict={x:x\_data,y:y\_data}).flatten())* **if** i%10==0:  
 **print**(accuracy(l1,l2))  
 **if**(accuracy(l1,l2)>0.9):  
 **print**(i)  
 **break**

**import** tensorflow **as** tf  
**from** numpy **import** \*  
**import** matplotlib.pyplot **as** plt  
*#数据*data = mat([  
 [1, 1, 1, 1, 1, 1, 0.697, 0.460, 1],  
 [2, 1, 2, 1, 1, 1, 0.774, 0.376, 1],  
 [2, 1, 1, 1, 1, 1, 0.634, 0.264, 1],  
 [1, 1, 2, 1, 1, 1, 0.608, 0.318, 1],  
 [3, 1, 1, 1, 1, 1, 0.556, 0.215, 1],  
 [1, 2, 1, 1, 2, 2, 0.403, 0.237, 1],  
 [2, 2, 1, 2, 2, 2, 0.481, 0.149, 1],  
 [2, 2, 1, 1, 2, 1, 0.437, 0.211, 1],  
 [2, 2, 2, 2, 2, 1, 0.666, 0.091, 0],  
 [1, 3, 3, 1, 3, 2, 0.243, 0.267, 0],  
 [3, 3, 3, 3, 3, 1, 0.245, 0.057, 0],  
 [3, 1, 1, 3, 3, 2, 0.343, 0.099, 0],  
 [1, 2, 1, 2, 1, 1, 0.639, 0.161, 0],  
 [3, 2, 2, 2, 1, 1, 0.657, 0.198, 0],  
 [2, 2, 1, 1, 2, 2, 0.360, 0.370, 0],  
 [3, 1, 1, 3, 3, 1, 0.593, 0.042, 0],  
 [1, 1, 2, 2, 2, 1, 0.719, 0.103, 0]])  
x\_data=data[:,0:8].astype(float32)  
y\_data=data[:,8].astype(float32)  
*#搭建网络*x=tf.placeholder(tf.float32,[None,8])  
y=tf.placeholder(tf.float32,[None,1])  
**def** addlayer(data,insize,outsize,active=None):  
 w=tf.Variable(tf.random\_normal([insize,outsize]))  
 b=tf.Variable(tf.zeros([1,outsize])+0.1)  
 w\_b=tf.matmul(data,w)+b  
 **return** active(w\_b)  
layer=addlayer(x,8,20,tf.nn.sigmoid)  
output=addlayer(layer,20,1,tf.nn.sigmoid)  
loss=tf.reduce\_mean(tf.reduce\_sum(tf.square(y-output),reduction\_indices=[1]))  
train=tf.train.GradientDescentOptimizer(0.5).minimize(loss)  
init=tf.initialize\_all\_variables()  
*#定义精度***def** accuracy(l1,l2):  
 **for** i **in** range(len(l2)):  
 **if** l2[i] > 0.5:  
 l2[i] = 1  
 **else**:  
 l2[i] = 0  
 s = abs(l1 - l2)  
 **return** 1-sum(s) / 17  
*#训练网络***with** tf.Session() **as** sess:  
 sess.run(init)  
 **for** i **in** range(2000):  
 sess.run(train,feed\_dict={x:x\_data[i%17],y:y\_data[i%17]})  
 *#print(array(y\_data).flatten(),sess.run(output,feed\_dict={x:x\_data,y:y\_data}).flatten())* l1 = array(y\_data).flatten()  
 l2 = sess.run(output, feed\_dict={x: x\_data, y: y\_data}).flatten()  
 *#if i%10==0:  
 #print(accuracy(l1,l2))* **if**(accuracy(l1,l2)>0.9):  
 **print**(i)  
 **break**

该段代码为标准BP算法，累计BP算法实现只要将i%17后面的%17删去即可。

1. **实验结果**

这里得出标准BP算法训练精度达到90%时需要823步，而累计BP算法训练精度达到90%需要248步，训练次数多，参数更新次数也多，由此可见标准BP算法的参数更新速率要比累计BP算法快。这与书本上的理论相一致。