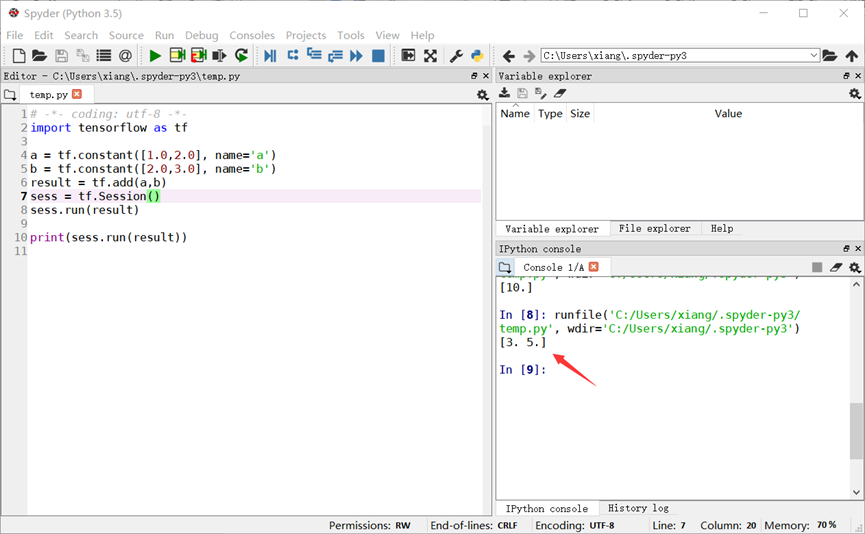
姓名：向建宇

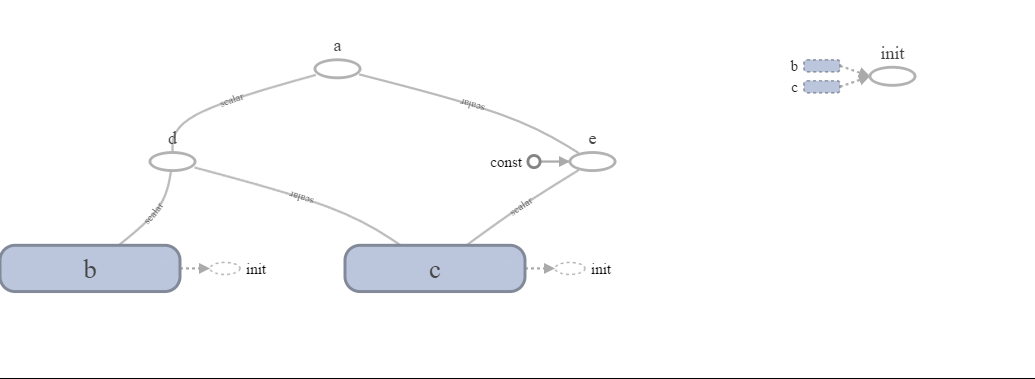
学号：1801210687

1.搭建tensorflow window环境，提供搭建成功截图。

按照方老师的说明，在安装完环境后进行下面的测试，得出结果如下：

2.建立维度为100\*100的随机权重和样本矩阵，参照完整神经网络样例程序，得出运行过程与结果。

**Tensorflow初步学习：**



Tensorflow将运算进行分解，异步计算，使得计算更快，如上面是*a*=(*b*+*c*)∗(*c*+2) 的数据图，张量从b，c进入，a流出

参考官方文档：

**圆形或方形的节点被称为node，在node中流动的数据流被称为张量(tensor)**



**神经网络实现过程：**

1. 准备数据集，提取特征，作为输入喂给神经网络

|  |
| --- |
| # 产生一个128行×2列的随机矩阵  datasize = 80  X = rdm.rand(datasize, 2)  # 产生一个布尔型结果矩阵128×1  Y = [[int(x1+x2 > 1)] for (x1, x2) in X] |

1. 搭建NN结构，从输入到输出（先搭建计算图，再用绘画执行）

|  |
| --- |
| w1 = tf.Variable(tf.random\_normal([2, 100], stddev=1, seed=1))  w2 = tf.Variable(tf.random\_normal([100, 1], stddev=1, seed=1))  # placeholder占位符，执行时，在通过feed\_dict将值传入，dtype：数据类型，shape：数据形状，name：名称  # 如果在定义占位符时，不能确定值的形状时，用None表示  x = tf.placeholder(tf.float32, shape=(None, 2), name="x-input")  y\_ = tf.placeholder(tf.float32, shape=(None, 1), name="y-input")  # 前向传播的过程  # matmul就是矩阵相乘  a = tf.matmul(x, w1)  y = tf.matmul(a, w2)  y = tf.sigmoid(y) |

1. 大量数据喂给NN，迭代优化NN参数

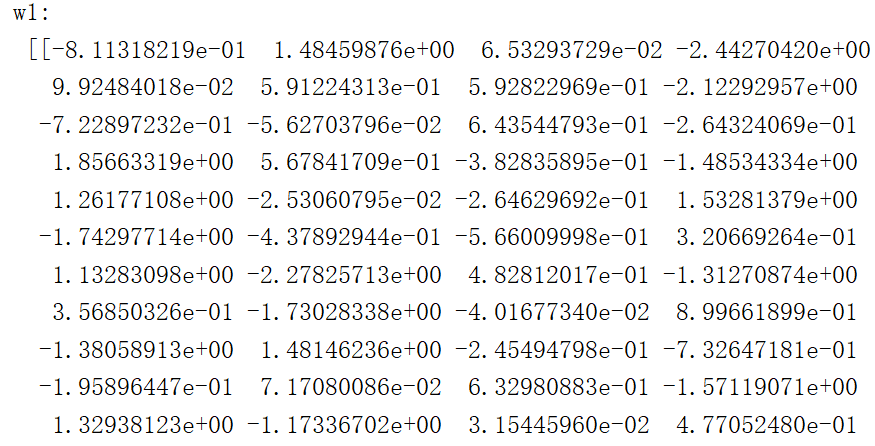
|  |
| --- |
| cross\_entropy = tf.reduce\_mean(tf.square(y\_-y)) *# train\_step 就是每次可训练出一组w就是一次反向传播*  *这里用了梯度下降法* train\_step=tf.train.GradientDescentOptimizer(0.001).minimize(cross\_entropy) |

1. 使用训练好的模型预测和分类

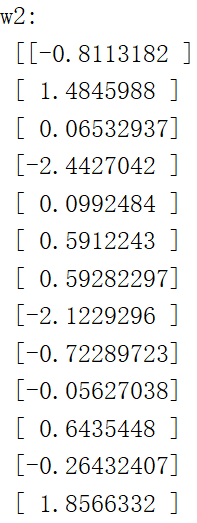
|  |
| --- |
| **with** tf.Session()**as** sess:  *# 初始化变量，也就是上面的Variable变量* init\_op = tf.initialize\_all\_variables()  sess.run(init\_op)  *# 输出目前（未经训练）的参数取值  # 这里为何还要使用sess.run(w1)来进行操作？  # 因为此时w1还是个变量，是个对象，直接打印出来是对象的地址  # 训练模型  # 训练步数* STEPS = 10000  **for** i **in** range(STEPS):  start = (i \* BATCH\_SIZE) % datasize  end = (i \* BATCH\_SIZE) % datasize + BATCH\_SIZE  sess.run(train\_step, feed\_dict={x: X[start:end], y\_: Y[start:end]})  **if** i % 1000 == 0:  total\_cross\_entropy = sess.run(cross\_entropy, feed\_dict={x: X, y\_: Y})  print(**"After %d training steps,cross entropy on all data is %g"** % (i, total\_cross\_entropy)) |

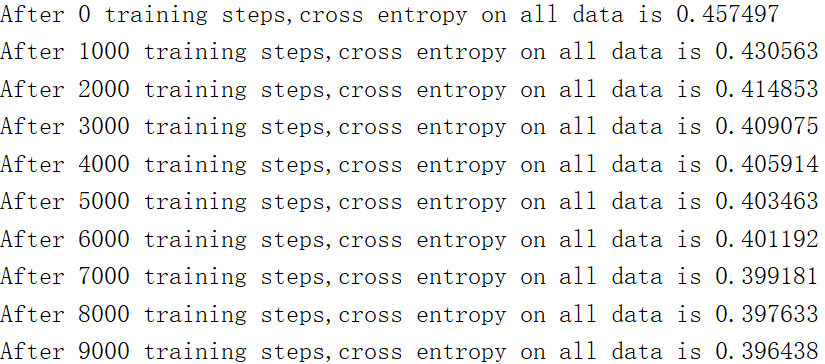
**结果截图：**

未优化前W1(由于实验要求是100\*100的权重矩阵，所以下面的截图只包含前一部分的值)



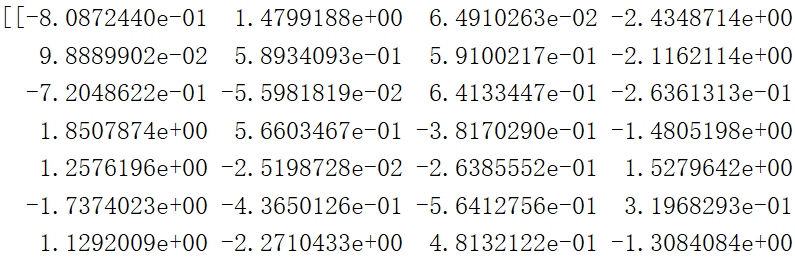
未优化前W2：



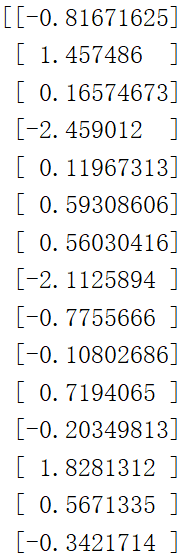


经过9000轮迭代后：

得到的W1：



W2：



附录：

|  |
| --- |
| *# 总结： # 训练神经网络的过程可以分为以下三个步骤： # 1、定义神经网络的结构和前向传播的输出结果 # 2、定义损失函数以及选择反向传播优化的算法 # 3、生成会话（tf.Session）并且在训练数据上反复运行反向传播优化算法 # coding:utf-8* **import** tensorflow **as** tf **from** numpy.random **import** RandomState  *# BATCH\_SIZE这是定义的一个数量，即一次训练模型，投入的样例数，按理说应该是一次投入所有训练数据的 # 但是实际操作中一次投入太多数据，会导致电脑死机，因此只好对投入样例数，进行限制* BATCH\_SIZE = 10 *# 生成模拟数据 # randomState 函数中数字1,相当于一个seed种子，每次产生的随机数都是相同的* rdm = RandomState(1) *# 产生一个128行×2列的随机矩阵* datasize = 80 X = rdm.rand(datasize, 2) *# 产生一个布尔型结果矩阵128×1 # Y = [[int(x1 + x2) < 1] for (x1, x2) in X]* Y = [[int(x1+x2 > 1)] **for** (x1, x2) **in** X]  *# 定义神经网络的常量，参数，输入节点，输出节点，以及前向传播过程 # Variable定义一个变量，需要调用初始化，才能产生值 # random\_normal 产生一个正太随机矩阵，shape=[2,3],stddev是正太分布的标准差，seed随机数种子，设置后每次产生的数字都相同* w1 = tf.Variable(tf.random\_normal([2, 100], stddev=1, seed=1)) w2 = tf.Variable(tf.random\_normal([100, 1], stddev=1, seed=1))  *# placeholder占位符，执行时，在通过feed\_dict将值传入，dtype：数据类型，shape：数据形状，name：名称 # 如果在定义占位符时，不能确定值的形状时，用None表示* x = tf.placeholder(tf.float32, shape=(**None**, 2), name=**"x-input"**) y\_ = tf.placeholder(tf.float32, shape=(**None**, 1), name=**"y-input"**)  *# 前向传播的过程 # matmul就是矩阵相乘* a = tf.matmul(x, w1) y = tf.matmul(a, w2) y = tf.sigmoid(y)  *# 定义损失函数及反向传播算法 # cross\_entropy是交叉熵 # tf.clip\_by\_value(A, min, max)：输入一个张量A，把A中的每一个元素的值都压缩在min和max之间。小于min的让它等于min，大于max的元素的值等于max # cross\_entropy = -tf.reduce\_mean(y\_ \* tf.log(tf.clip\_by\_value(y, 1e-10, 1.0))) # cross\_entropy = -tf.reduce\_mean(y\_ \* tf.log(tf.clip\_by\_value(y, 1e-10, 1.0))  # + (1 - y\_) \* tf.log(tf.clip\_by\_value(1 - y, 1e-10, 1.0)))* cross\_entropy = tf.reduce\_mean(tf.square(y\_-y)) *# train\_step 就是每次可训练出一组w就是一次反向传播 # 下面给出里三种反向传播的算法传入的是学习率 # train\_step = tf.train.AdamOptimizer(0.001).minimize(cross\_entropy)* train\_step=tf.train.GradientDescentOptimizer(0.001).minimize(cross\_entropy) *# train\_step=tf.train.MomentumOptimizer(0.001,0.9).minimize(cross\_entropy)  # 创建一个会话来运行Tensorflow程序，反复运行反向传播 # tf中运行必须放在session对象中，且运行过后，要关闭session* **with** tf.Session()**as** sess:  *# 初始化变量，也就是上面的Variable变量* init\_op = tf.initialize\_all\_variables()  sess.run(init\_op)  *# 输出目前（未经训练）的参数取值  # 这里为何还要使用sess.run(w1)来进行操作？  # 因为此时w1还是个变量，是个对象，直接打印出来是对象的地址* print(**"w1:\n"**, sess.run(w1))  print(**"w2:\n"**, sess.run(w2))   *# 训练模型  # 训练步数* STEPS = 10000  **for** i **in** range(STEPS):  start = (i \* BATCH\_SIZE) % datasize  end = (i \* BATCH\_SIZE) % datasize + BATCH\_SIZE  sess.run(train\_step, feed\_dict={x: X[start:end], y\_: Y[start:end]})  **if** i % 1000 == 0:  total\_cross\_entropy = sess.run(cross\_entropy, feed\_dict={x: X, y\_: Y})  print(**"After %d training steps,cross entropy on all data is %g"** % (i, total\_cross\_entropy))  *# 输出训练后的参数取值。  # print("\n")  # print("X:\n", X)* print(sess.run(w1))  *# print("a:\n", sess.run(a, feed\_dict={x: X}))* print(sess.run(w2))  *# print(Y)  # yyu = sess.run(y, feed\_dict={x: X})  # for i in range(len(yyu)):  # print("y:{} Y:{}\n".format(yyu[i],Y[i]))* |