# 基于Tensorflow的回归任务

**1. 步骤**

**1.1 生成训练数据**

使用以下代码来生成一个二次曲线数据：

x\_data = np.linspace(-0.5,0.5,200)[:,np.newaxis];

noise = np.random.normal(0,0.02,x\_data.shape)

y\_data = np.square(x\_data) + noise

这里生成了200行1列的x\_data数据和y\_data数据，注意，这里虽然x\_data是一维数据，但是为了配合后面的运算，需要插入新的维度，方法是在原数据后面加上[:,np.newaxis]。

两者的区别可以参考以下程序：

import numpy as np

x = np.linspace(0.0,10.0,5)[:,np.newaxis]

y = np.linspace(0.0,10.0,5)

print(x.shape)

print(y.shape)

结果：

(5, 1)

(5,)

**2.2 搭建神经网络**

#定义两个placeholder

x\_input = tf.placeholder(tf.float32,[None,1])

y\_input = tf.placeholder(tf.float32,[None,1])

#定义神经网络中间层，为10个神经元,由于输入是一维的，因此shape为1\*10

Weight\_L1 = tf.Variable(tf.random\_normal([1,10]))

biases\_L1 = tf.Variable(tf.zeros([1,10]))

x\_W\_plus\_b\_L1 = tf.matmul(x\_input,Weight\_L1) + biases\_L1

L1=tf.nn.tanh(x\_W\_plus\_b\_L1)

#定义神经网络输出层

Weight\_L2 = tf.Variable(tf.random\_normal([10,1]))

biases\_L2 = tf.Variable(tf.zeros([1,1]))

x\_W\_plus\_b\_L2 = tf.matmul(L1,Weight\_L2) + biases\_L2

prediction = tf.nn.tanh(x\_W\_plus\_b\_L2)

以上，输入层为x\_data，第一层神经元个数为10，接下来为输出层，神经元个数为1。x\_data的形状为None×1，W1形状为1×10，biases\_L1的形状为1×10，每一个偏置项对应一个神经元，L1层的输出为x\_data×W1+biase\_L1，结果的形状为1×10。W2连接着中间层10个神经元级输出层1个神经元，因此形状为10×1，偏置项为1×1，通过矩阵的运算规则可以判断，神经网络的输出为None×1，正好是y\_data的形状，所明网络的结构是正确的。

**2.3 代价函数**

这里使用二次代价函数：

loss = tf.reduce\_mean(tf.square(y\_input-prediction))

**2.4 定义训练操作**

train\_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

学习率为0.1，其目的是最小化前面定义的loss函数。

**2.5 运行图**

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

for i in range(2000):

sess.run(train\_step,feed\_dict = {x\_input:x\_data, y\_input:y\_data})

#获得预测值

prediction\_value = sess.run(prediction,feed\_dict={x\_input:x\_data})

#画图

plt.figure()

plt.scatter(x\_data,y\_data)

plt.plot(x\_data,prediction\_value,'r-',lw=5)

plt.show()

首先初始化所有变量，然后迭代2000次进行训练，获取预测值直接使用代码：sess.run(prediction,feed\_dict={x\_input:x\_data})

最后画出原始数据和预测结果。

**2. 代码**

#生成数据

x\_data = np.linspace(-0.5,0.5,200)[:,np.newaxis];

noise = np.random.normal(0,0.02,x\_data.shape)

y\_data = np.square(x\_data) + noise

#定义两个placeholder

x\_input = tf.placeholder(tf.float32,[None,1])

y\_input = tf.placeholder(tf.float32,[None,1])

#定义神经网络中间层，为10个神经元,由于输入是一维的，因此shape为1\*10

Weight\_L1 = tf.Variable(tf.random\_normal([1,10]))

biases\_L1 = tf.Variable(tf.zeros([1,10]))

x\_W\_plus\_b\_L1 = tf.matmul(x\_input,Weight\_L1) + biases\_L1

L1=tf.nn.tanh(x\_W\_plus\_b\_L1)

#定义神经网络输出层

Weight\_L2 = tf.Variable(tf.random\_normal([10,1]))

biases\_L2 = tf.Variable(tf.zeros([1,1]))

x\_W\_plus\_b\_L2 = tf.matmul(L1,Weight\_L2) + biases\_L2

prediction = tf.nn.tanh(x\_W\_plus\_b\_L2)

#二次代价函数

loss = tf.reduce\_mean(tf.square(y\_input-prediction))

#使用梯度下降法训练

train\_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

for i in range(2000):

sess.run(train\_step,feed\_dict = {x\_input:x\_data, y\_input:y\_data})

#获得预测值

prediction\_value = sess.run(prediction,feed\_dict={x\_input:x\_data})

#画图

plt.figure()

plt.scatter(x\_data,y\_data)

plt.plot(x\_data,prediction\_value,'r-',lw=5)

plt.show()

