# 基于Tensorflow的简单的拟合

## 步骤

### 生成数据

以下生成一个线性的二维数据：

x\_data = np.random.rand(100)

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

y\_data = x\_data \* 0.1 + 0.2 + noise

程序的目的就是通过这里的x\_data和y\_data来拟合得到k和b，其中k为斜率，b为截距。

### 1.2 构造线性模型

由于这里使用线性模型来拟合数据，因此构造一个线性模型，输入为x数据，输出为y数据，线性模型由一个k值和b值确定，程序的目的就是优化k和b的值，因此定义两个变量k和b：

b = tf.Variable(0.0)

k = tf.Variable(0.0)

y=k\*x\_data + b

### 1.3 代价函数

这里使用二次代价函数，计算真实值和预测值的差的平方的平均值：

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

### 1.4 定义优化器

optimizer = tf.train.GradientDescentOptimizer(0.1)

这里使用步长为0.1的梯度下降法优化器。

### 1.5 定义训练操作

train = optimizer.minimize(loss)

训练的操作即为最小化上面定义的二次代价函数。

### 1.6 初始化并训练

init\_op = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init\_op)

for step in range(200):

sess.run(train)

if step%20==0:

print("%d, K = %f, B = %f"%(step,sess.run(k),sess.run(b)))

## 2.代码

import tensorflow as tf

import numpy as np

#产生一组线性分布的数据

x\_data = np.random.rand(100)

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

y\_data = x\_data \* 0.1 + 0.2 + noise

#构造一个线性模型

b = tf.Variable(0.0)

k = tf.Variable(0.0)

y=k\*x\_data + b

#二次代价函数

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

#定义一个梯度下降法来进行训练的优化器

optimizer = tf.train.GradientDescentOptimizer(0.1)

#最小化代价函数

train = optimizer.minimize(loss)

#初始化变量

init\_op = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init\_op)

for step in range(200):

sess.run(train)

if step%20==0:

print("%d, K = %f, B = %f"%(step,sess.run(k),sess.run(b)))

输出：

0, K = 0.025164, B = 0.049657

20, K = 0.099605, B = 0.199973

40, K = 0.098566, B = 0.201122

60, K = 0.097566, B = 0.201633

80, K = 0.096790, B = 0.202028

100, K = 0.096189, B = 0.202335

120, K = 0.095722, B = 0.202572

140, K = 0.095360, B = 0.202756

160, K = 0.095080, B = 0.202899

180, K = 0.094863, B = 0.203010