# 一、三维数据, 然后用一个平面拟合它.

import tensorflow as tf

import numpy as np

# 使用 NumPy 生成假数据(phony data), 总共 100 个点.

x\_data = np.float32(np.random.rand(2, 100)) # 随机输入

y\_data = np.dot([0.100, 0.200], x\_data) + 0.300

# 构造一个线性模型

b = tf.Variable(tf.zeros([1]))

|  |  |  |
| --- | --- | --- |
| tf.zeros | 产生全0的数组 | tf.zeros([2,3],int32)->[[0,0,0],[0,0,0]] |
| tf.ones | 产生全1的数组 | tf.ones([2,3],int32)->[[1,1,1],[1,1,1]] |
| tf.fill | 产生一个全部为给定数字的数组 | tf.fill([2,3],9)->[[9,9,9],[9,9,9]] |
| tf.constant | 产生一个给定值的常量 | tf.constant([1,2,3])->[1,2,3] |

W = tf.Variable(tf.random\_uniform([1, 2], -1.0, 1.0))

构造一个一行两列的数组。数值在-1到1之间

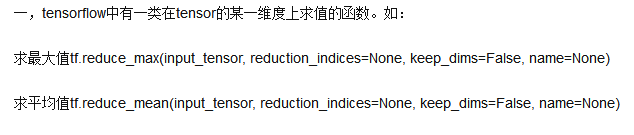
y = tf.matmul(W, x\_data) + b

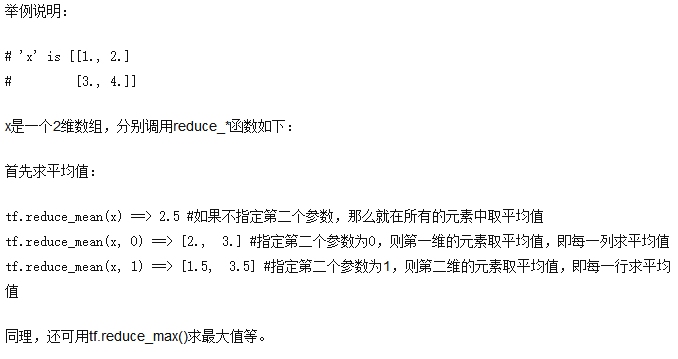
实现矩阵的相乘计算

# 最小化方差

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

1、





2、tf.square():

|  |  |
| --- | --- |
| tf.add(x, y, name=None) | 求和 |
| tf.sub(x, y, name=None) | 减法 |
| tf.mul(x, y, name=None) | 乘法 |
| tf.div(x, y, name=None) | 除法 |
| tf.mod(x, y, name=None) | 取模 |
| tf.abs(x, name=None) | 求绝对值 |
| tf.neg(x, name=None) | 取负 (y = -x). |
| tf.sign(x, name=None) | 返回符号 y = sign(x) = -1 if x < 0; 0 if x == 0; 1 if x > 0. |
| tf.inv(x, name=None) | 取反 |
| tf.square(x, name=None) | 计算平方 (y = x \* x = x^2). |
| tf.round(x, name=None) | 舍入最接近的整数 # ‘a’ is [0.9, 2.5, 2.3, -4.4] tf.round(a) ==> [ 1.0, 3.0, 2.0, -4.0 ] |
| tf.sqrt(x, name=None) | 开根号 (y = \sqrt{x} = x^{1/2}). |
| tf.pow(x, y, name=None) | 幂次方  # tensor ‘x’ is [[2, 2], [3, 3]] # tensor ‘y’ is [[8, 16], [2, 3]] tf.pow(x, y) ==> [[256, 65536], [9, 27]] |
| tf.exp(x, name=None) | 计算e的次方 |
| tf.log(x, name=None) | 计算log，一个输入计算e的ln，两输入以第二输入为底 |
| tf.maximum(x, y, name=None) | 返回最大值 (x > y ? x : y) |
| tf.minimum(x, y, name=None) | 返回最小值 (x < y ? x : y) |
| tf.cos(x, name=None) | 三角函数cosine |
| tf.sin(x, name=None) | 三角函数sine |
| tf.tan(x, name=None) | 三角函数tan |
| tf.atan(x, name=None) | 三角函数ctan |

optimizer = tf.train.GradientDescentOptimizer(0.5)

train = optimizer.minimize(loss)

1、梯度下降算法优化器

tf.train.GradientDescentOptimizer(learning\_Rate,use\_locking=False,name=’GradientDescent’）

只需要一个learning\_Rate

minimize(loss)

通过更新var\_list来减小loss，这个函数就是前面compute\_gradients() 和apply\_gradients().的结合

# 初始化变量

init = tf.global\_variables\_initializer()

# 启动图 (graph)

sess = tf.Session()

sess.run(init)

# 拟合平面

For step in range(0, 201):

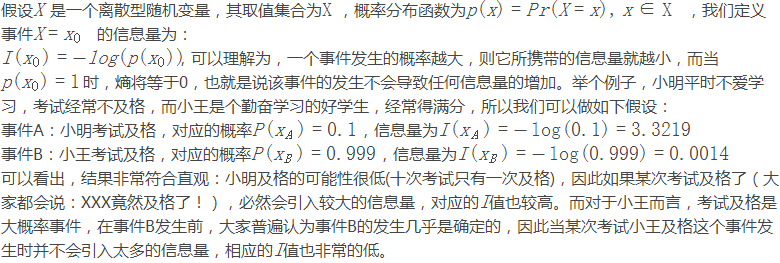
sess.run(train)

if step % 20 == 0:

print (step, sess.run(W), sess.run(b))

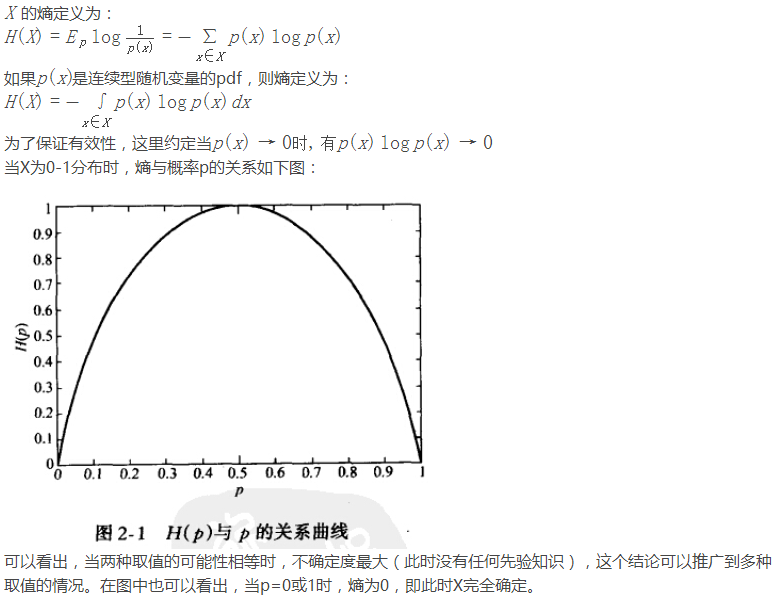
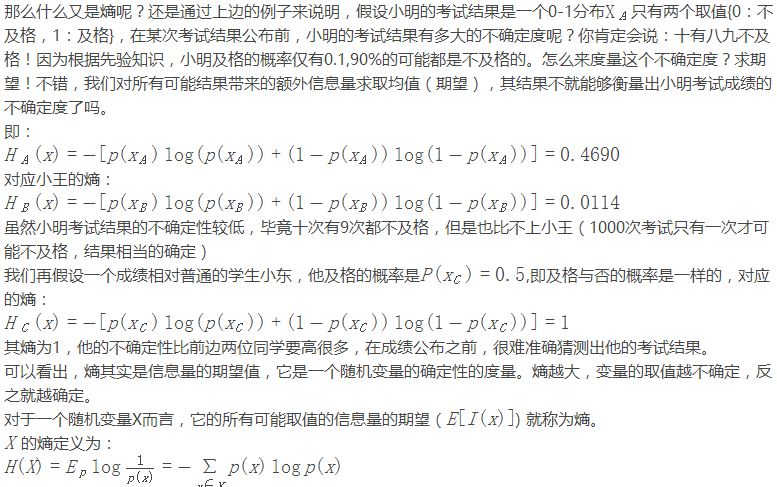
# 二、识别mnist数据集

## （一）什么是信息量？



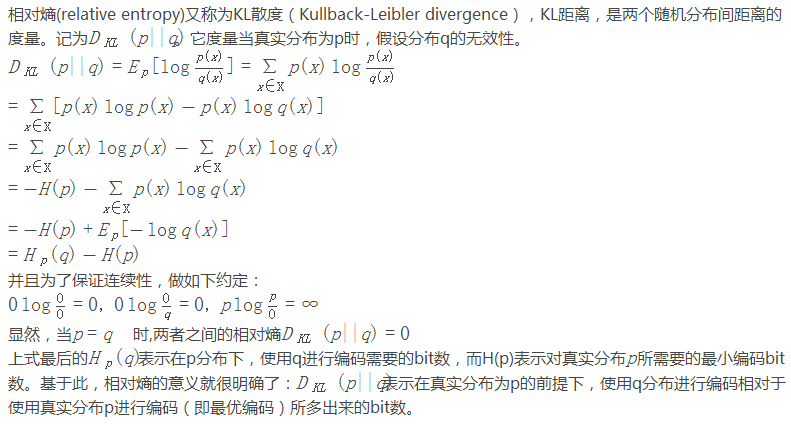
所谓信息量就是求一个概率的副对数-log(P(x))

## （二）什么是熵？

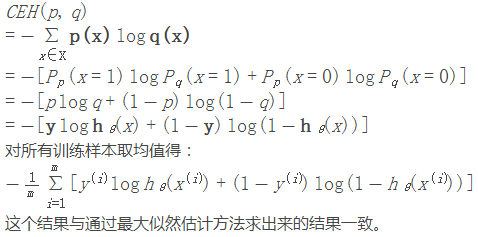


我的理解熵就是信息量的期望。一件事情熵越大，这件事情的不确定性就越大，熵越小就说明确定性越大。

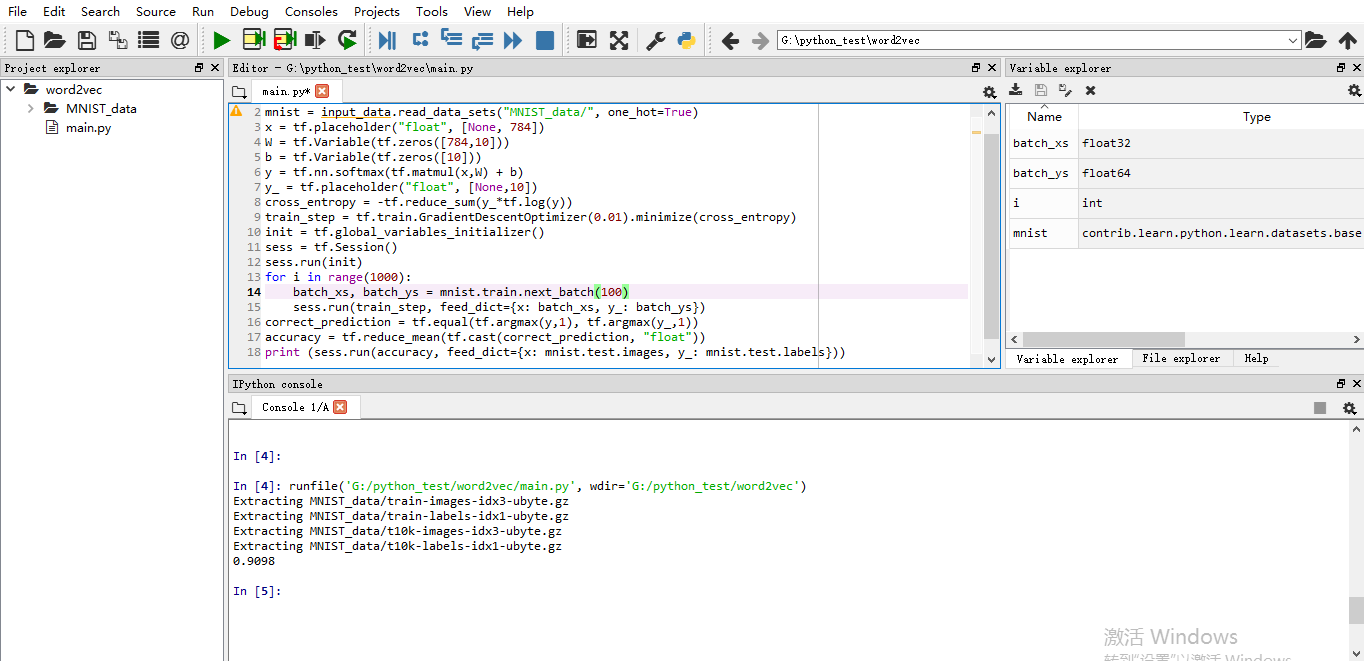
## （三）什么是相对熵？



## （四）什么是交叉熵？



## （五）Hello world代码



import tensorflow as tf

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

//导入了数据集

x = tf.placeholder("float", [None, 784])

W = tf.Variable(tf.zeros([784,10]))

b = tf.Variable(tf.zeros([10]))

y = tf.nn.softmax(tf.matmul(x,W) + b)

y\_ = tf.placeholder("float", [None,10])

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y))

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step, feed\_dict={x: batch\_xs, y\_: batch\_ys})

correct\_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

print (sess.run(accuracy, feed\_dict={x: mnist.test.images, y\_: mnist.test.labels}))