**一．理论**

梯度与海森矩阵公式推导图如图1所示。

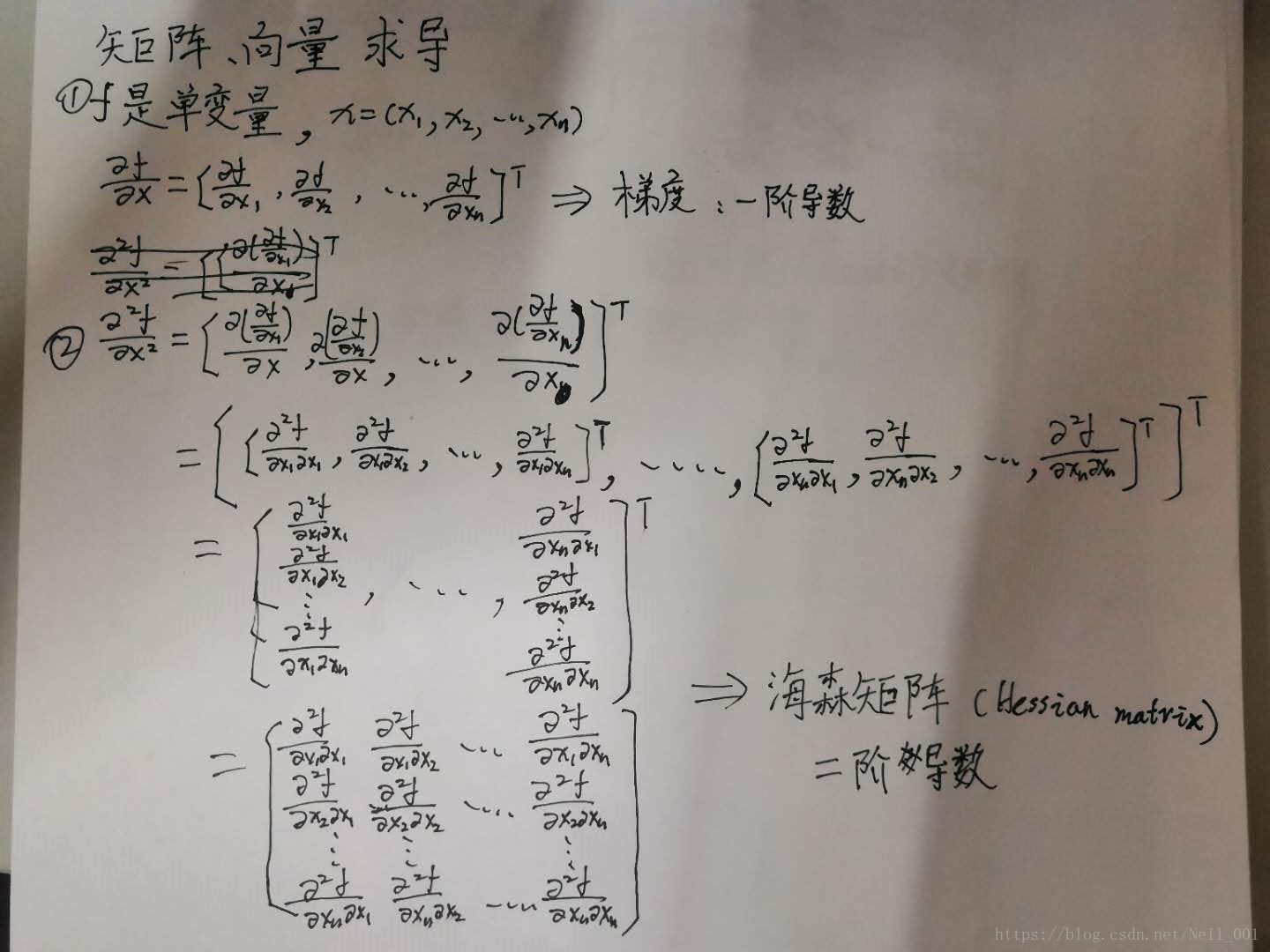
****

图1 梯度与海森矩阵推导图

**二．梯度**

**1.梯度下降**

**1.1 为何使用梯度作为下降方向？**

**梯度实际上是函数值变化最快的方向。**

比如说，你站在一个山上，梯度所指示的方向是高度变化最快的方向。你沿着这个方向走，能最快的改变（增加或是减小）你所在位置的高度，但是如果你乱走，可能走半天所在位置高度也没有变化多少。也就是说，如果你一直沿着梯度走，你就能最快的到达山的某个顶峰或低谷（偶尔会到鞍点，不过这几乎不可能）。

所以实际上，梯度下降法是用来数值搜索局部极小值或极大值的，它是实际应用中一种非常高效，高速且可靠的方法。

* 1. **以逻辑斯蒂回归（LR）为例**

**模型参数估计**

1. <p><img alt="" class="has" height="362" src="https://img-blog.csdn.net/20180819112218752?watermark/2/text/aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L1NlY29uZExpZXV0ZW5hbnQ=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70" width="578"></p>
2. </li>
3. <li><strong>梯度下降学习参数</strong>&nbsp;
4. <p><img alt="" class="has" height="167" src="https://img-blog.csdn.net/20180819112324513?watermark/2/text/aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L1NlY29uZExpZXV0ZW5hbnQ=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70" width="263"></p>
5. </li>
6. <li><strong>最终模型</strong>&nbsp;
7. <p><img alt="" class="has" height="129" src="https://img-blog.csdn.net/20180819112344497?watermark/2/text/aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L1NlY29uZExpZXV0ZW5hbnQ=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70" width="472"></p>
8. </li>

**1.3 具体学习过程(python代码示例)**

**梯度下降是最小化风险函数、损失函数的一种常用方法，随机梯度下降和批量梯度下降是两种迭代求解思路。**

根据batch\_size的不同，可以有大概一下几种形式。

**（1）梯度下降伪代码**

* 每个回归参数初始化为1
* 重复R次
  + 计算整个数据集的梯度
  + 使用alpha × gradient更新回归系数的向量
  + 返回回归系数

示例代码：

1. def gradAscent(dataMatIn, classLabels):
2. dataMatrix = mat(dataMatIn) #convert to NumPy matrix
3. labelMat = mat(classLabels).transpose() #convert to NumPy matrix
4. m,n = shape(dataMatrix)
5. alpha = 0.001
6. maxCycles = 500
7. weights = ones((n,1))
8. for k in range(maxCycles): #heavy on matrix operations
9. h = sigmoid(dataMatrix\*weights) #matrix mult
10. error = (labelMat - h) #vector subtraction
11. weights = weights + alpha \* dataMatrix.transpose()\* error #matrix mult
12. return weights

**（2）随机梯度下降伪代码：**

* 每个回归参数初始化为1
* 重复R次
  + 计算每个样本的梯度
  + 使用alpha × gradient更新回归系数的向量
  + 返回回归系数

示例代码：

可以看到，其中alpha是变化的，这样可以在一定程度上避免局部最优解。

1. def stocGradAscent1(dataMatrix, classLabels, numIter=150):
2. m,n = shape(dataMatrix)
3. weights = ones(n) #initialize to all ones
4. for j in range(numIter):
5. dataIndex = range(m)
6. for i in range(m):
7. alpha = 4/(1.0+j+i)+0.0001 #apha decreases with iteration, does not
8. randIndex = int(random.uniform(0,len(dataIndex)))#go to 0 because of the constant
9. h = sigmoid(sum(dataMatrix[randIndex]\*weights))
10. error = classLabels[randIndex] - h
11. weights = weights + alpha \* error \* dataMatrix[randIndex]
12. del(dataIndex[randIndex])
13. return weights

**（3）自定义batch\_size,算法流程与上面基本差不错，累计误差更新参数。**梯度下降，就是batch\_size = 全部样本量，随机梯度下降就是batch\_size = 1。

三．海森矩阵





import tensorflow as tf

import numpy as np

def cons(x):

return tf.constant(x, dtype=tf.float32)

def compute\_hessian(fn, vars):

mat = []

for v1 in vars:

temp = []

for v2 in vars:

# computing derivative twice, first w.r.t v2 and then w.r.t v1

temp.append(tf.gradients(tf.gradients(f, v2)[0], v1)[0])

temp = [cons(0) if t == None else t for t in temp] # tensorflow returns None when there is no gradient, so we replace None with 0

temp = tf.stack(temp)

mat.append(temp)

mat = tf.stack(mat)

return mat

x = tf.Variable(np.random.random\_sample(), dtype=tf.float32)

y = tf.Variable(np.random.random\_sample(), dtype=tf.float32)

f = tf.pow(x, cons(2)) + cons(2) \* x \* y + cons(3) \* tf.pow(y, cons(2)) + cons(4)\* x + cons(5) \* y + cons(6)

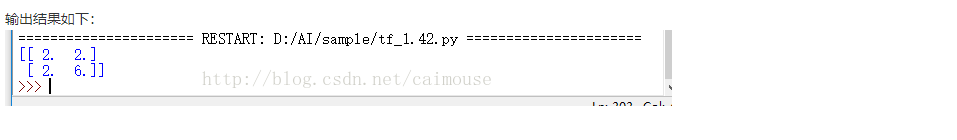
# arg1: our defined function, arg2: list of tf variables associated with the function

hessian = compute\_hessian(f, [x, y])

sess = tf.Session()

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

print(sess.run(hessian))



再来举多一个例子的源码，用来计算量化分析，如下：

import numpy as np

import scipy.stats as stats

import scipy.optimize as opt

#构造Hessian矩阵

def rosen\_hess(x):

x = np.asarray(x)

H = np.diag(-400\*x[:-1],1) - np.diag(400\*x[:-1],-1)

diagonal = np.zeros\_like(x)

diagonal[0] = 1200\*x[0]\*\*2-400\*x[1]+2

diagonal[-1] = 200

diagonal[1:-1] = 202 + 1200\*x[1:-1]\*\*2 - 400\*x[2:]

H = H + np.diag(diagonal)

return H

def rosen(x):

"""The Rosenbrock function"""

return sum(100.0\*(x[1:]-x[:-1]\*\*2.0)\*\*2.0 + (1-x[:-1])\*\*2.0)

def rosen\_der(x):

xm = x[1:-1]

xm\_m1 = x[:-2]

xm\_p1 = x[2:]

der = np.zeros\_like(x)

der[1:-1] = 200\*(xm-xm\_m1\*\*2) - 400\*(xm\_p1 - xm\*\*2)\*xm - 2\*(1-xm)

der[0] = -400\*x[0]\*(x[1]-x[0]\*\*2) - 2\*(1-x[0])

der[-1] = 200\*(x[-1]-x[-2]\*\*2)

return der

x\_0 = np.array([0.5, 1.6, 1.1, 0.8, 1.2])

res = opt.minimize(rosen, x\_0, method='Newton-CG', jac=rosen\_der, hess=rosen\_hess,options={'xtol': 1e-8, 'disp': True})

print("Result of minimizing Rosenbrock function via Newton-Conjugate-Gradient algorithm (Hessian):")

print(res)

