梯度下降法

梯度下降法，又叫最速下降法，主要用于求解无约束的最优化问题。梯度下降法的计算过程就是沿负梯度下降的方向求解目标函数的极小值（或者沿梯度上升方向求解极大值），常用于机器学习和人工智能当中用来递归性地逼近最小偏差模型。

**1、梯度（向量）：**

以二元函数为例，设函数在平面D内具有一阶连续偏导数，则对于每一点都可以定义出一个向量



该向量称为函数z在点的梯度，记做grad。

梯度的模为：



例：假设有函数，则该函数的梯度为，若步长为，则。

在单变量的实值函数的情况,梯度只是导数,或者,对于一个线性函数,也就是线的斜率。梯度一词有时用于斜度,也就是一个曲面沿着给定方向的倾斜程度。可以通过取向量梯度和所研究的方向的点积来得到斜度。梯度的数值有时也被成为梯度。

为什么沿梯度方向函数下降最快？

**2、方向导数：**

**函数在特定方向上的变化速率，是一个数值，函数在某点的任一方向上,随着该自变量的变化,而引起的函数值的变化率。**

我们知道函数在某一点的导数（微商）代表了函数在该点的变化率。对于单变量函数f(x)，它在x0处导数是：当x趋近于x0时，函数的改变量与自变量的改变量的比值的极限，即微商（导数）等于差商的极限。



对于单变量函数，自变量只有一个，当x趋近于x0时只能在直线上变动，移动的方向只有左右两方。

对于多变量函数，自变量有多个，表示自变量的点在一个区域内变动，不仅可以移动距离，而且可以按任意的方向来移动同一段距离。因此，函数的变化不仅与移动的距离有关，而且与移动的方向有关。**因此，函数的变化率是与方向有关的。这也才有了方向导数的定义，即某一点在某一方向上的导数值。**

**定义：函数的增量与两点之间的距离的比值，当沿着****趋于点时，如果极限存在，则称该极限为沿的方向导数，记为**

****

*比较偏导数：函数在某点处延坐标轴正向,随着该自变量的变化,而引起的函数值的变化率。*

*1、比较明显,偏导数只是延坐标轴方向,而方向导数的方向任意；  
2、偏导数是一种特殊的方向导数。*

*我们看到如果是求“延着坐标轴正向”的方向求方向导数,与偏导数是一样的；如果是求“延着坐标轴负向”的方向求方向导数,结果与偏导数差一个负号。*

设点为上一点，过而且平行于向量的参数方程为



函数在沿方向的方向导数为



可以看到，当变化方向发生变化时，会取得不同的方向导数，当同向时，方向导数取得最大值，即沿梯度方向函数有最大的变化速率。

**3、梯度下降法步骤：**

1、建立目标函数，收敛精度为，初始值，迭代次数为，初始步长为；

2、迭代公式：  ，是由是否小于0确定的，若大于0，则表示迭代后的函数值增加了，则需缩短步长；

3、终止条件：

A、,即某点的梯度的长度（模）小于收敛精度;

B、，前后两次所求得的函数值之间的差异小于设定的收敛精度；

C、迭代达到预定最大迭代次数n。

**缺点：**

1、梯度法从初始点的领域开始判断，在局部进行下降，然后步步逼近极值，极易陷入局部最优解；

2、在远离极小值的地方下降很快，而在靠近极小值的地方下降很慢；

3、步长的确定比较麻烦，太大了的话可能会发散，太小收敛速度又太慢。

**例1：**从一确定点出发求函数极小值:

1. 求解梯度



迭代公式：



梯度值：

示例代码：

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
**def f**(x):  
 **return** 2 \* (x[0] \*\* 2) + 3 \* (x[1] \*\* 3)  
  
  
**def g**(x):  
 **return** 4 \* x[0], 9 \* (x[1]\*\*2)  
  
  
**def gd**(x\_start, step, g, n, o):  
 x = x\_start  
 **for** i **in** range(n):  
 grad\_old = g(x)  
 f\_old = f(x)  
 x\_old =x  
 x = (x\_old[0] - grad\_old[0] \* step, x\_old[1] - grad\_old[1] \* step)  
 grad\_new = g(x)  
 f\_new = f(x)  
  
 """  
 line search，基本思想就是每次试一个步长，如果用该步长走的话，看函数值会不会比当前点下降一定的程度，如果没有，就按 比例减小步长，再试，直到满足条件  
 """  
 **while** f\_new - f\_old > 0:  
 step \*= 0.5  
 x = (x\_old[0] - grad\_old[0] \* step, x\_old[1] - grad\_old[1] \* step)  
 grad\_new = g(x)  
 f\_new = f(x)  
 **print** f\_new - f\_old  
  
 **print** '[ Epoch {0} ] grad = {1}, x = {2}'.format(i, grad\_new, x)  
 grad\_value = abs((grad\_new[0] \*\* 2) + (grad\_new[1] \*\* 2))  
 f\_d = abs(f\_new - f\_old)  
 **if** grad\_value < o:  
 **print** "grad stop:" + str(grad\_value)  
 **break  
 if** f\_d < o:  
 **print** "f\_d stop:" + str(f\_d)  
 **break** """  
 每次迭代均按比例缩小步长  
 """  
 step \*= 0.4  
 **return** x  
  
  
x\_0 = (0.2, 0.3)  
gd(x\_0, 0.9, g, 100, 0.001)

**例2：**线性回归

单变量线性回归模型：

假设给定一组数据：

x\_0 = [0, 1, 1.9, 3.1, 4, 5, 6, 7, 9]；y\_0 = [1, 2.5, 5.5, 6.8, 10, 9, 13, 14, 20]

假设通过改组数据，找到多个，我们可以通过这些来预测y值，但是哪个是最好的的呢？这个标准就是损失函数。

损失函数：，所有样本点的预测的值跟实际的值之间的差距的表达式而已。

我们的目标就是求使得损失函数取得最小值的最佳回归系数。

步骤：

1、目标函数：，梯度方向：

 

迭代公式为：



2、给定初始值，初始步长，迭代更新值，使其沿着梯度下降的方向减少，并计算损失函数值，查看是否达到终止条件。

示例代码：

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
x = np.linspace(0, 7, 100)  
x\_0 = [0, 1, 1.9, 3.1, 4, 5, 6, 7, 9]  
y\_0 = [1, 2.5, 5.5, 6.8, 10, 9, 13, 14, 20]  
theta\_start = [1, 1.5]  
  
  
**def J**(theta\_0, theta\_1):  
 value = 0  
 **for** i **in** range(len(x\_0)):  
 value += 0.5 \* (theta\_0 + theta\_1 \* x\_0[i] - y\_0[i])  
 **return** value  
  
  
**def gd**(start, step, n, o):  
 theta = start  
 **for** i **in** range(n):  
 J\_old = J(theta[0], theta[1])  
 **for** j **in** range(len(x\_0)):  
 h = theta[0] + theta[1] \* x\_0[j]  
 theta[0] -= step \* (h - y\_0[j])  
 theta[1] -= step \* (h - y\_0[j]) \* x\_0[j]  
 J\_new = J(theta[0], theta[1])  
 loss = abs(J\_new - J\_old)  
 **if** loss < o:  
 **print** "loss = " + str(loss)  
 **print** "n = " + str(i)  
 **break** step \*= 0.2  
 **return** theta  
  
  
gd\_value = gd([1, 1.8], 0.02, 100, 0.00000001)  
**print** gd\_value  
  
plt.plot(x, gd\_value[0] + (gd\_value[1] \* x))  
plt.plot(x\_0, y\_0, "o")  
# plt.plot(x, 2 \* x + 1)  
plt.show()