深入浅出--梯度下降法及其实现

六尺帐篷 关注
54 2018.01.17 21:06:22 字数 3,001 阅读 280,092

本文将从一个下山的场景开始，先提出梯度下降算法的基本思想，进而从数学上解释梯度下降算法的原理，最后实现一个简单的梯度下降算法的实例！

梯度下降的场景假设

梯度下降法的基本思想可以类比为一个下山的过程。假设这样一个场景：一个人被困在山上，需要从山上下来(i.e. 找到山的最低点，也就是山谷)。但此时山上的浓雾很大，导致可视度很低。因此，下山的路径就无法确定，他必须利用自己周围的信息去找到下山的路径。这个时候，他就可以利用梯度下降算法来帮助自己下山。具体来说就是，以他当前的所处的位置为基准，寻找这个位置最陡峭的地方，然后朝着山的高度下降的地方走，同理，如果我们的目标是上山，也就是爬到山顶，那么此时应该是朝着最陡峭的方向往上走。然后每走一段距离，都反复采用同一个方法，最后就能成功的抵达山谷。

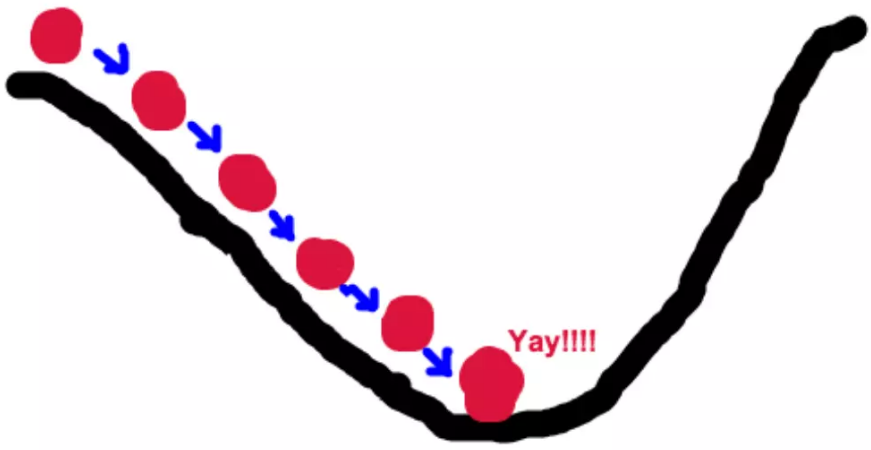


image.png

我们同时可以假设这座山最陡峭的地方是无法通过肉眼立马观察出来的，而是需要一个复杂的工具来测量，同时，这个人此时正好拥有测量出最陡峭方向的能力。所以，此人每走一段距离，都需要一段时间来测量所在位置最陡峭的方向，这是比较耗时的。那么为了在

六尺帐篷 关注
总资产62 (约5.92元)

comsol快速入门教程
阅读 76,759

数字签名和数字证书究竟是什么?
阅读 8,548

推荐阅读

利用pandas分析时序数据+可视化
阅读 994

必读论文 | 卷积神经网络百篇经典论文推荐
阅读 460

python用线性回归预测时间序列股票价格
阅读 296

基于树模型的集成算法---Random Forest
阅读 104

OpenCV-Python 级联分类器训练 | 六十三
阅读 98

梯度下降

梯度下降的基本过程就和下山的场景很类似。

首先，我们有一个可*微分*的函数。这个函数就代表着一座山。我们的目标就是找到这个函数的最小值，也就是山底。根据之前的场景假设，最快的下山的方式就是找到当前位置最陡峭的方向，然后沿着此方向向下走，对应到函数中，就是找到给定点的*梯度*，然后朝着梯度相反的方向，就能让函数值下降的最快！因为梯度的方向就是函数之变化最快的方向(在后面会详细解释)
所以，我们重复利用这个方法，反复求取梯度，最后就能到达局部的最小值，这就类似于我们下山的过程。而求取梯度就确定了最陡峭的方向，也就是场景中测量方向的手段。那么为什么梯度的方向就是最陡峭的方向呢？接下来，我们从微分开始讲起

微分

看待微分的意义，可以有不同的角度，最常用的两种是：

- 函数图像中，某点的切线的斜率
- 函数的变化率
  几个微分的例子：

- $\frac{d(x^2)}{dx} = 2x$
- $\frac{d(-2y^5)}{dy} = -10y^4$
- $\frac{d(5-\theta)^2}{d\theta} = -2(5-\theta)$

image.png

上面的例子都是单变量的微分，当一个函数有多个变量的时候，就有了多变量的微分，即分别对每个变量进行求微分

- $\frac{\partial}{\partial x}(x^2y^2) = 2xy^2$
- $\frac{\partial}{\partial y}(-2y^5 + z^2) = -10y^4$
- $\frac{\partial}{\partial \theta_2}(5\theta_1 + 2\theta_2 - 12\theta_3) = 2$
- $\frac{\partial}{\partial \theta_2}(0.55-(5\theta_1 + 2\theta_2 - 12\theta_3)) = -2$

image.png

梯度

梯度实际上就是多变量微分的一般化

$$\nabla J(\Theta) = \left\langle \frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \frac{\partial J}{\partial \theta_3} \right\rangle$$
$$= \langle -5, -2, 12 \rangle$$

image.png

我们可以看到，梯度就是分别对每个变量进行微分，然后用逗号分割开，梯度是用<>包括起来，说明梯度其实一个向量。

梯度是微积分中一个很重要的概念，之前提到过梯度的意义

- 在单变量的函数中，梯度其实就是函数的微分，代表着函数在某个给定点的切线的斜率
- 在多变量函数中，梯度是一个向量，向量有方向，梯度的方向就指出了函数在给定点的上升最快的方向

这也就说明了为什么我们需要千方百计的求取梯度！我们需要到达山底，就需要在每一步观测到此时最陡峭的地方，梯度就恰巧告诉了我们这个方向。梯度的方向是函数在给定点上升最快的方向，那么梯度的反方向就是函数在给定点下降最快的方向，这正是我们所需要的。所以我们只要沿着梯度的方向一直走，就能走到局部的最低点！

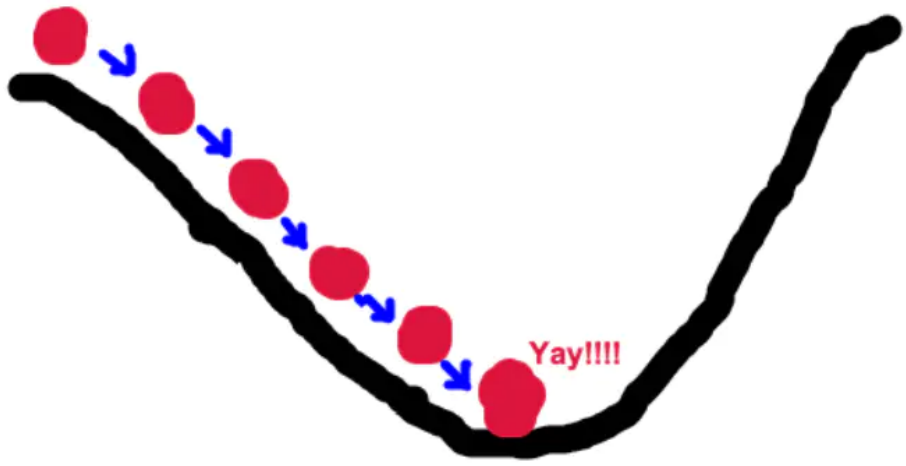


image.png

梯度下降算法的数学解释

上面我们花了大量的篇幅介绍梯度下降算法的基本思想和场景假设，以及梯度的概念和思想。下面我们就开始从数学上解释梯度下降算法的计算过程和思想！

$$\Theta^1 = \Theta^0 - \alpha \nabla J(\Theta) \quad \text{evaluated at } \Theta^0$$

image.png

此公式的意义是：J是关于Θ的一个函数，我们当前所处的位置为Θ0点，要从这个点走到J的最小

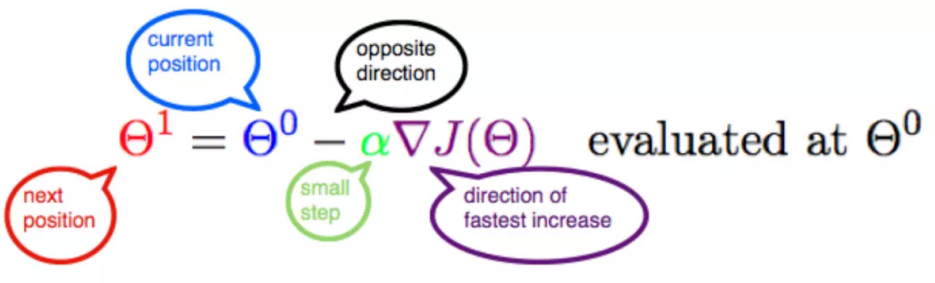


image.png

下面就这个公式的几个常见的疑问：

- α是什么含义？
  α在梯度下降算法中被称作为**学习率**或者**步长**，意味着我们可以通过α来控制每一步走的距离，以保证不要步子跨的太大扯着蛋，哈哈，其实就是不要走太快，错过了最低点。同时也要保证不要走的太慢，导致太阳下山了，还没有走到山下。所以α的选择在梯度下降法中往往是很重要的！α不能太大也不能太小，太小的话，可能导致迟迟走不到最低点，太大的话，会导致错过最低点！

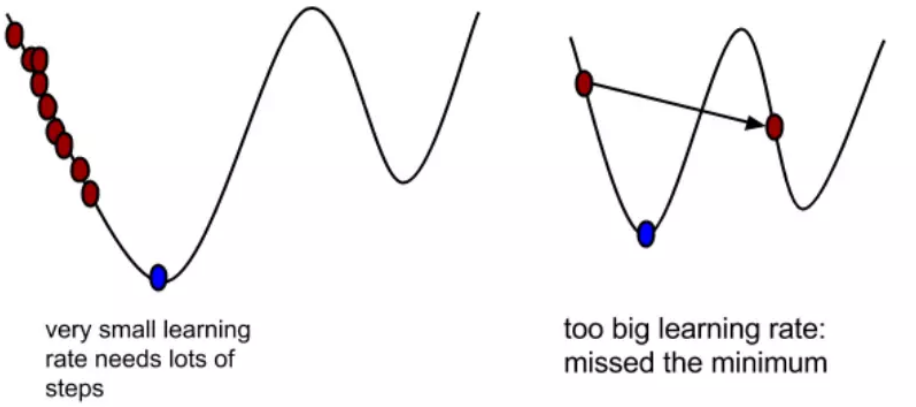


image.png

- 为什么要梯度要乘以一个负号？
  梯度前加一个负号，就意味着朝着梯度相反的方向前进！我们在前文提到，梯度的方向实际就是函数在此点上升最快的方向！而我们需要朝着下降最快的方向走，自然就是负的梯度的方向，所以此处需要加上负号

梯度下降算法的实例

我们已经基本了解了梯度下降算法的计算过程，那么我们就来看几个梯度下降算法的小实例，首先从单变量的函数开始

单变量函数的梯度下降

我们假设有一个单变量的函数

$$J(\theta) = \theta^2$$

$$J'(\theta) = 2\theta.$$

image.png

初始化，起点为

$$\theta^0 = 1$$

image.png

学习率为

$$\alpha = 0.4$$

image.png

根据梯度下降的计算公式

$$\Theta^1 = \Theta^0 - \alpha \nabla J(\Theta) \quad \text{evaluated at } \Theta^0$$

image.png

我们开始进行梯度下降的迭代计算过程：

$$\begin{aligned}
\theta^0 &= 1 \\
\theta^1 &= \theta^0 - \alpha * J'(\theta^0) \\
&= 1 - 0.4 * 2 \\
&= 0.2 \\
\theta^2 &= \theta^1 - \alpha * J'(\theta^1) \\
&= 0.04 \\
\theta^3 &= 0.008 \\
\theta^4 &= 0.0016
\end{aligned}$$

image.png

如图，经过四次的运算，也就是走了四步，基本就抵达了函数的最低点，也就是山底

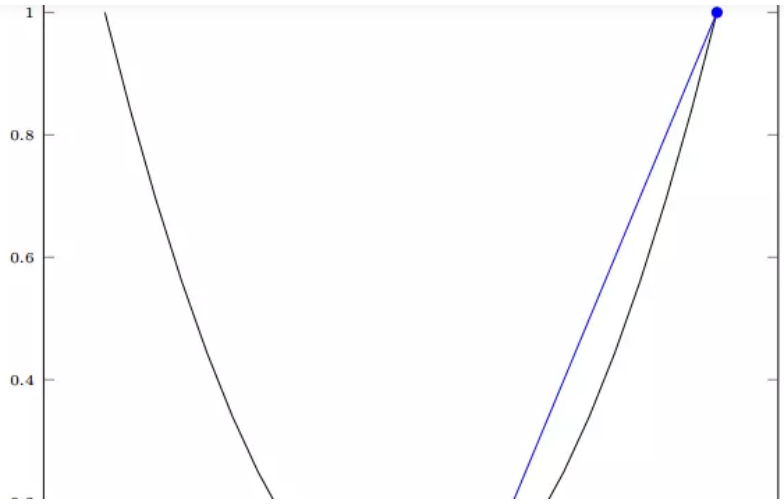


image.png

多变量函数的梯度下降

我们假设有一个目标函数

$$J(\Theta) = \theta_1^2 + \theta_2^2.$$

image.png

现在要通过梯度下降法计算这个函数的最小值。我们通过观察就能发现最小值其实就是 (0，0) 点。但是接下来，我们会从梯度下降算法开始一步步计算到这个最小值！
我们假设初始的起点为：

$$\Theta^0 = (1, 3)$$

image.png

初始的学习率为：

$$\alpha = 0.1.$$

image.png

函数的梯度为：

$$\nabla J(\Theta) = \langle 2\theta_1, 2\theta_2 \rangle$$

image.png

进行多次迭代：

$$
\begin{aligned}
\Theta^1 &= \Theta^0 - \alpha\nabla J(\Theta) \\
&= (1,3) - 0.1(2,6) \\
&= (0.8, 2.4) \\
\Theta^2 &= (0.8, 2.4) - 0.1(1.6, 4.8) \\
&= (0.64, 1.92) \\
\Theta^3 &= (0.512, 1.536) \\
\Theta^4 &= (0.4096, 1.2288000000000001) \\
&\vdots \\
\Theta^{10} &= (0.10737418240000003, 0.32212254720000005) \\
&\vdots \\
\Theta^{50} &= (1.1417981541647683\mathrm{e}^{-05}, 3.425394462494306\mathrm{e}^{-05})
\end{aligned}
$$

image.png

我们发现，已经基本靠近函数的最小值点

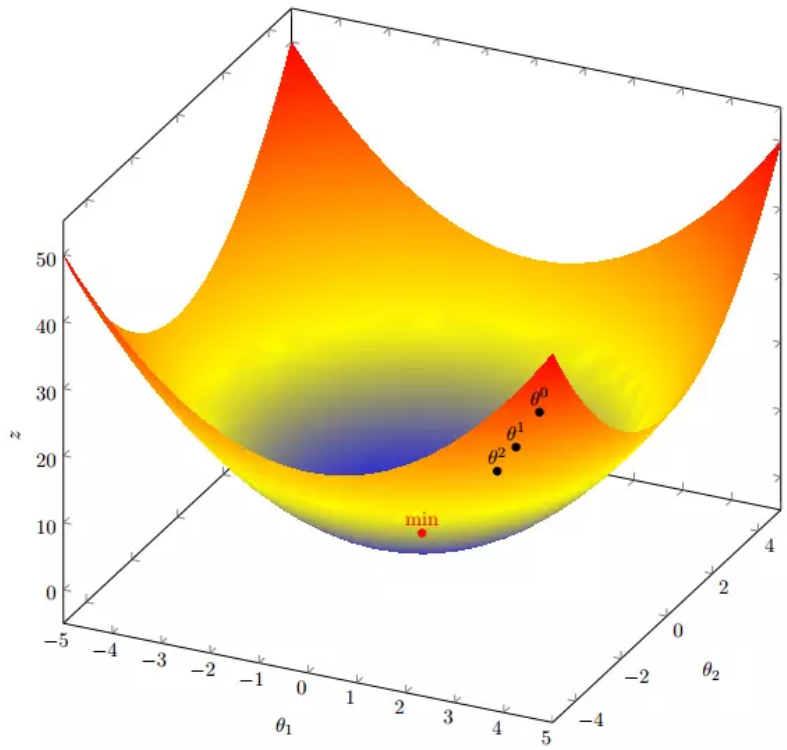


image.png

梯度下降算法的实现

下面我们将用python实现一个简单的梯度下降算法。场景是一个简单的*线性回归*的例子：假设现在我们有一系列的点，如下图所示

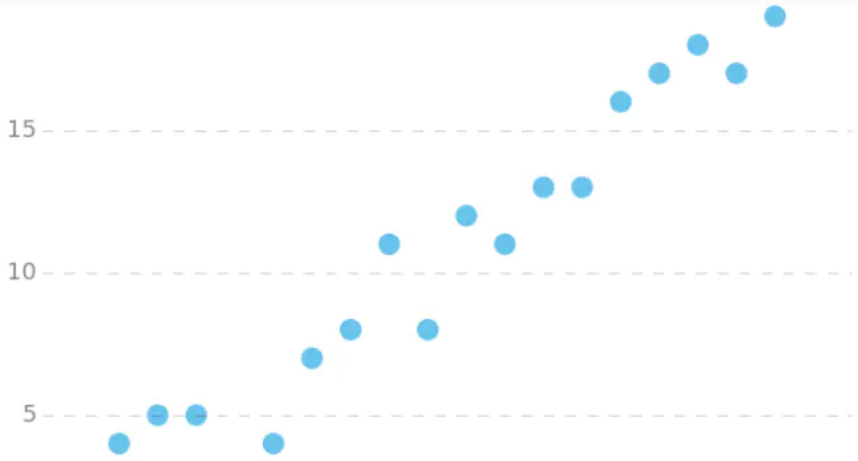


image.png

我们将用梯度下降法来拟合出这条直线!

首先，我们需要定义一个代价函数，在此我们选用*均方误差代价函数*

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2$$

image.png

此公示中

- m是数据集中点的个数
- ½是一个常量，这样是为了在求梯度的时候，二次方乘下来就和这里的½抵消了，自然就没有多余的常数系数，方便后续的计算，同时对结果不会有影响
- y 是数据集中每个点的真实y坐标的值
- h 是我们的预测函数，根据每一个输入x，根据Θ 计算得到预测的y值，即

$$h_\Theta(x^{(i)}) = \Theta_0 + \Theta_1 x_1^{(i)}$$

image.png

我们可以根据代价函数看到，代价函数中的变量有两个，所以是一个多变量的梯度下降问题，求解出代价函数的梯度，也就是分别对两个变量进行微分

$$\left(\frac{\partial}{\partial\Theta_0}\ \frac{\partial}{\partial\Theta_1}\right)$$

$$\frac{\delta J}{} \quad \frac{1}{} \sum^{m} (h_{} (x^{(i)}) \quad y^{(i)})$$

image.png

明确了代价函数和梯度，以及预测的函数形式。我们就可以开始编写代码了。但在这之前，需要说明一点，就是为了方便代码的编写，我们会将所有的公式都转换为矩阵的形式，python中计算矩阵是非常方便的，同时代码也会变得非常的简洁。

为了转换为矩阵的计算，我们观察到预测函数的形式

$$h_\Theta(x^{(i)}) = \Theta_0 + \Theta_1 x_1^{(i)}$$

image.png

我们有两个变量，为了对这个公式进行矩阵化，我们可以给每一个点x增加一维，这一维的值固定为1，这一维将会乘到Θ0上。这样就方便我们统一矩阵化的计算

$$(x_1^{(i)}, y^{(i)}) \to (x_0^{(i)}, x_1^{(i)}, y^{(i)}) \text{ with } x_0^{(i)} = 1\ \forall i$$

image.png

然后我们将代价函数和梯度转化为矩阵向量相乘的形式

$$J(\Theta) = \frac{1}{2m}(X\Theta - \vec{y})^T(X\Theta - \vec{y})$$

$$\nabla J(\Theta) = \frac{1}{m}X^T(X\Theta - \vec{y})$$

image.png

coding time

首先，我们需要定义数据集和学习率

```
import numpy as np

# Size of the points dataset.
m = 20

# Points x-coordinate and dummy value (x0, x1).
X0 = np.ones((m, 1))
```

```
    11, 13, 13, 16, 17, 18, 17, 19, 21
]).reshape(m, 1)

# The Learning Rate alpha.
alpha = 0.01
```

接下来我们以矩阵向量的形式定义代价函数和代价函数的梯度

```
def error_function(theta, X, y):
    '''Error function J definition.'''
    diff = np.dot(X, theta) - y
    return (1./2*m) * np.dot(np.transpose(diff), diff)

def gradient_function(theta, X, y):
    '''Gradient of the function J definition.'''
    diff = np.dot(X, theta) - y
    return (1./m) * np.dot(np.transpose(X), diff)
```

最后就是算法的核心部分，梯度下降迭代计算

```
def gradient_descent(X, y, alpha):
    '''Perform gradient descent.'''
    theta = np.array([1, 1]).reshape(2, 1)
    gradient = gradient_function(theta, X, y)
    while not np.all(np.absolute(gradient) <= 1e-5):
        theta = theta - alpha * gradient
        gradient = gradient_function(theta, X, y)
    return theta
```

当梯度小于1e-5时，说明已经进入了比较平滑的状态，类似于山谷的状态，这时候再继续迭代效果也不大了，所以这个时候可以退出循环！

完整的代码如下

```
import numpy as np

# Size of the points dataset.
m = 20

# Points x-coordinate and dummy value (x0, x1).
X0 = np.ones((m, 1))
X1 = np.arange(1, m+1).reshape(m, 1)
X = np.hstack((X0, X1))

# Points y-coordinate
y = np.array([
    3, 4, 5, 5, 2, 4, 7, 8, 11, 8, 12,
    11, 13, 13, 16, 17, 18, 17, 19, 21
]).reshape(m, 1)

# The Learning Rate alpha.
alpha = 0.01

def error_function(theta, X, y):
    '''Error function J definition.'''
    diff = np.dot(X, theta) - y
    return (1./2*m) * np.dot(np.transpose(diff), diff)

def gradient_function(theta, X, y):
    '''Gradient of the function J definition.'''
    diff = np.dot(X, theta) - y
    return (1./m) * np.dot(np.transpose(X), diff)

def gradient_descent(X, y, alpha):
    '''Perform gradient descent.'''
    theta = np.array([1, 1]).reshape(2, 1)
    gradient = gradient_function(theta, X, y)
```

```
print('optimal:', optimal)
print('error function:', error_function(optimal, X, y)[0,0])
```

运行代码，计算得到的结果如下

$$\Theta = \langle 0.51583286, 0.96992163 \rangle$$

$$J(\Theta) = 405.984962493$$

image.png

所拟合出的直线如下

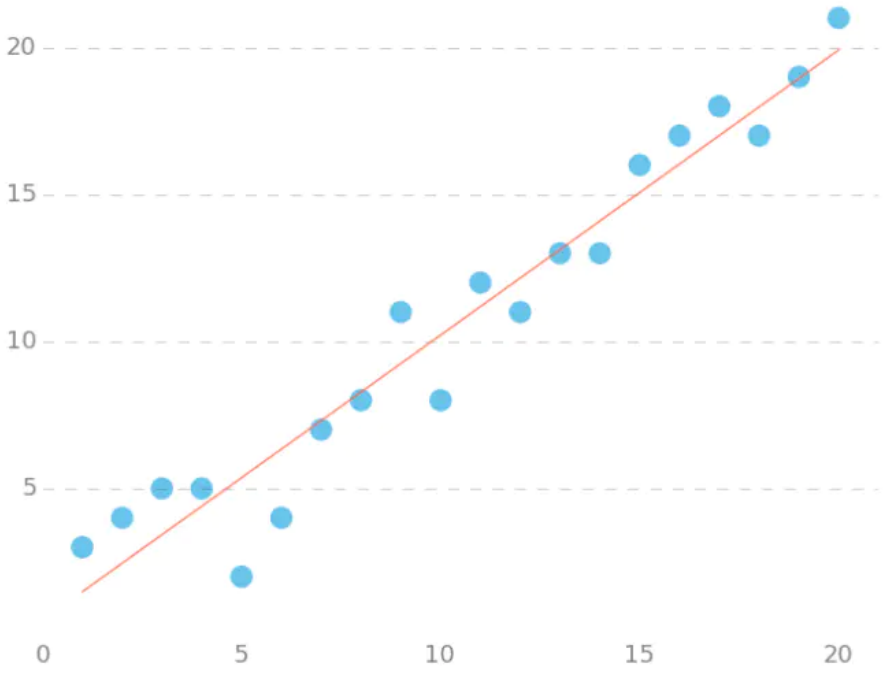


image.png

小结

至此，我们就基本介绍完了梯度下降法的基本思想和算法流程，并且用python实现了一个简单的梯度下降算法拟合直线的案例！

最后，我们回到文章开头所提出的场景假设:

这个下山的人实际上就代表了反向传播算法，下山的路径其实就代表着算法中一直在寻找的参数Θ，山上当前点的最陡峭的方向实际上就是代价函数在这一点的梯度方向，场景中观测最陡峭方向所用的工具就是微分 。在下一次观测之前的时间就是有我们算法中的学习率α所定义的。

可以看到场景假设和梯度下降算法很好的完成了对应！

Further reading

"小礼物走一走，来简书关注我"

赞赏支持

 共11人赞赏

六尺帐篷 https://github.com/chi2liu
总资产62 (约5.92元) 共写了26.1W字 获得2,919个赞 共16,927个粉丝

关注

写下你的评论...

精彩评论 3

奇趣社会观察
14楼 2018.09.12 12:31

我专门注册了个号，就是为了上来感谢你。写的太好了，只有真正把原理弄明白的人，才能把一个复杂的原理讲得这么简单。那些把这个原理讲的更加复杂的人，要么就是自己都没弄明白，要不就是故弄玄虚。

60 回复

loveplay1983
2018.10.08 06:17

表示严重同意,我搜索了好多资源都没搞明白,直到看到篇文章

回复

飞Alfred
2018.11.14 20:24

同意!

回复

玮杰_Utopian
2018.12.10 09:47

+1！，感谢作者

回复

添加新评论 | 还有9条评论，查看更多

道不远人_c4a0
61楼 2019.04.06 22:40

目标函数（error_function）计算有误，应该是 return (1/(2*m)) * np.dot(np.transpose(diff), diff)，作者未加括号，最后正确结果应该是：J(theta)=1.014962406233101

8 回复

amberlvp
2019.11.14 14:46
你是对的
回复

添加新评论

带梦飞翔_1c3c
69楼 2019.05.27 19:22
然后我们将代价函数和梯度转化为矩阵向量相乘的形式，最后这个将代价函数和梯度转换为向量的形式没太看懂，大神能解释一下吗？
7 回复

fangliangv587
2019.12.11 15:21
知道是怎么推导的了吗
回复

Vanilla_a383
05.31 11:33
我也想知道
回复

添加新评论

全部评论 163 只看作者 按时间倒序 按时间正序

叶子陪你玩
104楼 06.04 09:41
居然看明白了😀
赞 回复

zzz_9cd4
103楼 06.01 09:53
写的太棒了，深入浅出！
赞 回复

Vanilla_a383
102楼 05.31 11:34
还有最后面公式中的向量y到底是什么
赞 回复

陈文超happylion
101楼 05.23 11:01
写的通俗易通,赞
赞 回复

hxhpg
100楼 05.20 19:34

写下你的评论... 评论163 赞796 ···

超高校級貓茶
99楼 05.18 17:01

确实写得非常好，复杂概念简单化，很强的。

赞 回复

wu_776e
98楼 05.13 10:32

同意👍

赞 回复

whileTure
97楼 04.26 22:09

其实可以用最小二乘法解决，用scipy.optimize.leastsq()这个函数，函数矩阵化就是转置矩阵，目的就是算出斜率和截距。

赞 回复

whileTure
04.27 00:59

我的一点思路

```
import numpy as np
from scipy.optimize import leastsq
from pylab import*
%matplotlib inline
m=20
x = np.arange(1,m+1)
y=np.array([3, 4, 5, 5, 2, 4, 7, 8, 11, 8, 12,11, 13, 13, 16, 17, 18, 17, 19, 21])
def real_function(p,x):
k,b=p
return k * x + b

def erro_function(p,x,y):
return real_function(p,x)-y

p0=[1,1]
pysq=leastsq(erro_function,p0,args=(x,y))
print(pysq[0])
k,b=pysq[0]
x1=np.linspace(1,20,20)
y2=k * x1 + b
plot(x,y,"ob")
plot(x1,y2,"-r")
```

回复

添加新评论

Daway
96楼 02.15 16:17

最让人清晰的讲解，看前100字就能明白梯度下降的简单概念，棒

1 回复

Sigma_波色子
95楼 01.24 17:13

感觉有个地方有点小瑕疵诶，矩阵形式的代价函数梯度为什么是一个数值呢？为什么不是

写下你的评论... 评论163 赞796 ···

被以下专题收入，发现更多相似内容

程序员 机器学习与数据挖掘 深度学习·神经... 机器学习 Machine... 机器学习 科研成长时 展开更多

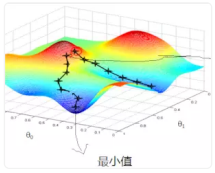
推荐阅读

更多精彩内容 >

梯度下降法（Gradient Descent）

转载-刘建平Pinard-www.cnblogs.com/pinard/p/5970503.html 在求解机器学...

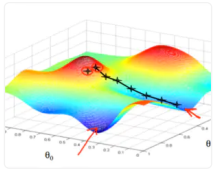
商三郎 阅读 1,578 评论 0 赞 1



一元线性回归-梯度下降法

在高数中，我们求解一个函数的最小值时，最常用的方法就是求出它的导数为0的那个点，进而判断这个点是否能够取最小值。但...

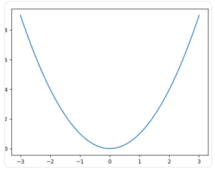
耳朵和爪子 阅读 2,446 评论 2 赞 5



从零开始：教你如何训练神经网络

姓名:吴兆阳 学号:14020199009 转自机器机器之心 嵌牛导读:作者从神经网络简单的数学定义开始，沿着损失...

吴兆阳 阅读 526 评论 0 赞 2



山重水复疑无路，最快下降问梯度（深度学习入门系列之七）

一年多前，吴军博士写了一本畅销书《智能时代》[1]。书里提到，在人工智能领域，有一个流派叫"鸟飞派"，亦称之为"模...

阿里云云栖号 阅读 748 评论 0 赞 7

机器学习理论系列2——梯度下降法

什么是优化算法 优化算法要求解的，是一个问题的最优解或者近似最优解。在机器学习中，有很多问题都是优化问题，即我们要...

刺猬ciwei_532a 阅读 974 评论 0 赞 7