

随机梯度下降算法

1. 梯度下降算法的作用

随机梯度下降算法，是用于求解机器学习算法参数的方法。梯度下降算法具体是什么呢？

1) 导数

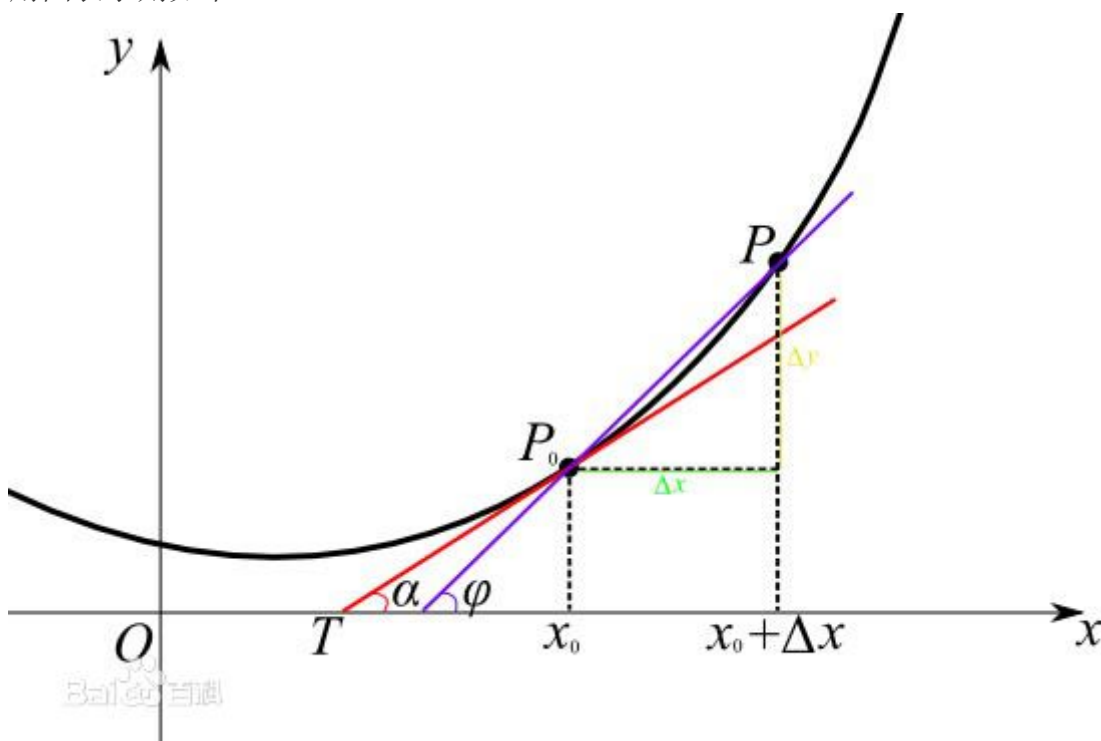
我们知道导数的定义如下：

$$f'(x_0) = \lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

$$= \frac{\partial(y)}{\partial(x)}$$

反映的是函数 $y=f(x)$ 在某一点处沿 x 轴正方向的变化率。再强调一遍，是函数 $f(x)$ 在 x 轴上某一点处沿着 x 轴正方向的变化率/变化趋势。直观地看，也就是在 x 轴上某一点处，如果 $f'(x)>0$ ，说明 $f(x)$ 的函数值在 x 点沿 x 轴正方向是趋于增加的；如果 $f'(x)<0$ ，说明 $f(x)$ 的函数值在 x 点沿 x 轴正方向是趋于减少的。

用图表示则如下：



上图中的 Δy 、 dy 等符号的意义及关系如下：

Δx : x 的变化量；

dx : x 的变化量 Δx 趋于0时，则记作微元 dx ；

Δy : $\Delta y = f(x_0 + \Delta x) - f(x_0)$ ，是函数的增量；

dy : $dy = f'(x_0)dx$ ，是切线的增量；

当 $\Delta x \rightarrow 0$ 时， dy 与 Δy 都是无穷小， dy 是 Δy 的主部，即 $\Delta y = dy + o(\Delta x)$ 。

2) 偏导数

上述导数的定义是针对单变量的，对于多变量的函数，如： $f(x, y, z, w) = x^2 + y^2 + z^2 + w^2$ ，对某个变量求导，则得到该变量的偏导数。如：

$\frac{d(f)}{d(x)} = 2x$ ，即为函数在变量 x 上的求导得到的导数。

偏导数的定义如下：

$$\frac{\partial}{\partial x_j} f(x_0, x_1, \dots, x_n) = \lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{f(x_0, \dots, x_j + \Delta x, \dots, x_n) - f(x_0, \dots, x_j, \dots, x_n)}{\Delta x}$$

可以看到，导数与偏导数本质是一致的，都是当自变量的变化量趋于0时，函数值的变化量与自变量变化量比值的极限。直观地说，偏导数也就是函数在某一点上沿坐标轴正方向的变化率。

区别在于：

导数，指的是一元函数中，函数 $y = f(x)$ 在某一点处沿 x 轴正方向的变化率；偏导数，指的是多元函数中，函数 $y = f(x_1, x_2, \dots, x_n)$ 在某一点处沿某一坐标轴 (x_1, x_2, \dots, x_n) 正方向的变化率

3) 方向导数

方向导数的定义如下：

$$\frac{\partial}{\partial l} f(x_0, x_1, \dots, x_n) = \lim_{\rho \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{\rho \rightarrow 0} \frac{f(x_0 + \Delta x_0, \dots, x_j + \Delta x_j, \dots, x_n + \Delta x_n) - f(x_0, \dots, x_j, \dots, x_n)}{\rho}$$
$$\rho = \sqrt{(\Delta x_0)^2 + \dots + (\Delta x_j)^2 + \dots + (\Delta x_n)^2}$$

方向导数就是函数在特定方向上的变化率，最大梯度是个

4) 梯度

梯度的定义如下：

$$\text{grad} f(x_0, x_1, \dots, x_n) = \left(\frac{\partial f}{\partial x_0}, \dots, \frac{\partial f}{\partial x_j}, \dots, \frac{\partial f}{\partial x_n} \right)$$

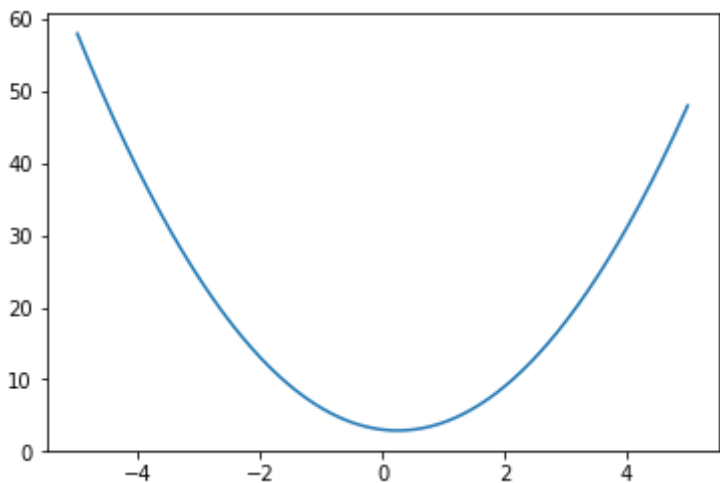
可见，梯度是一个向量，由各个分量上的导数所组成。梯度的方向就是最大方向导数的方向，梯度的模就是最大方向导数的值。

2. 梯度下降算法

梯度下降算法是机器学习领域最常用的优化算法。通常机器学习的任务是：已知数据集 $D = \{X_1, X_2, X_3, \dots, X_n\}$ ，其中 $X_i \in R^m$ 是 m 维实数向量。然后已知一个目标函数，求目标函数的最佳参数，能够使得目标函数最优。

1) 一维梯度下降算法示例

设当前有一个函数： $J(\theta) = 2\theta^2 - \theta + 2$ ，其函数图像如下：

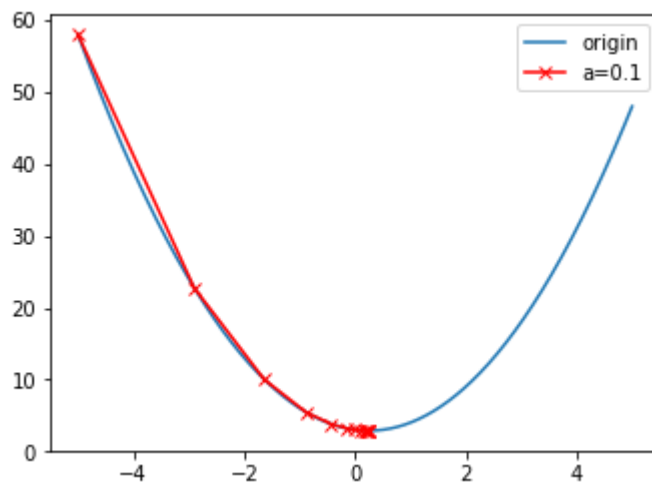
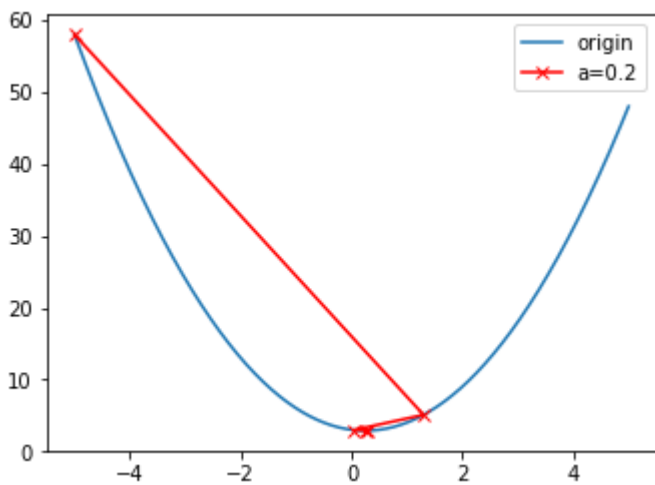


我们进行如下操作：

(a) 计算 $J(\theta)' = 4\theta - 1$

(b) 令 $\alpha = 0.2$, 初始 $\theta = 2$, 然后不断重复 $\theta := \theta - \alpha * J(\theta)'$, 直到 $|J(\theta)| \leq 0.01$

我们可以得到下图：



可见在左边图中，几个红色点的值在不断的减小，但是它是跳跃的，先左边，再到右边，最终到达底部。右边的图则是在单调往下。

同时我们可以看到，不同的 α 下，下降的幅度不一样。 α 大的时候下降快，小的时候下降慢。但是下降过快的时候，容易陷入震荡，一会儿跳到左边，一会儿在曲线右边，但是下降趋势不变。因此在很多算法里面，会将这个参数设置成可变，一开始下降快，然后慢慢变小。

2) 二维梯度下降算法示例

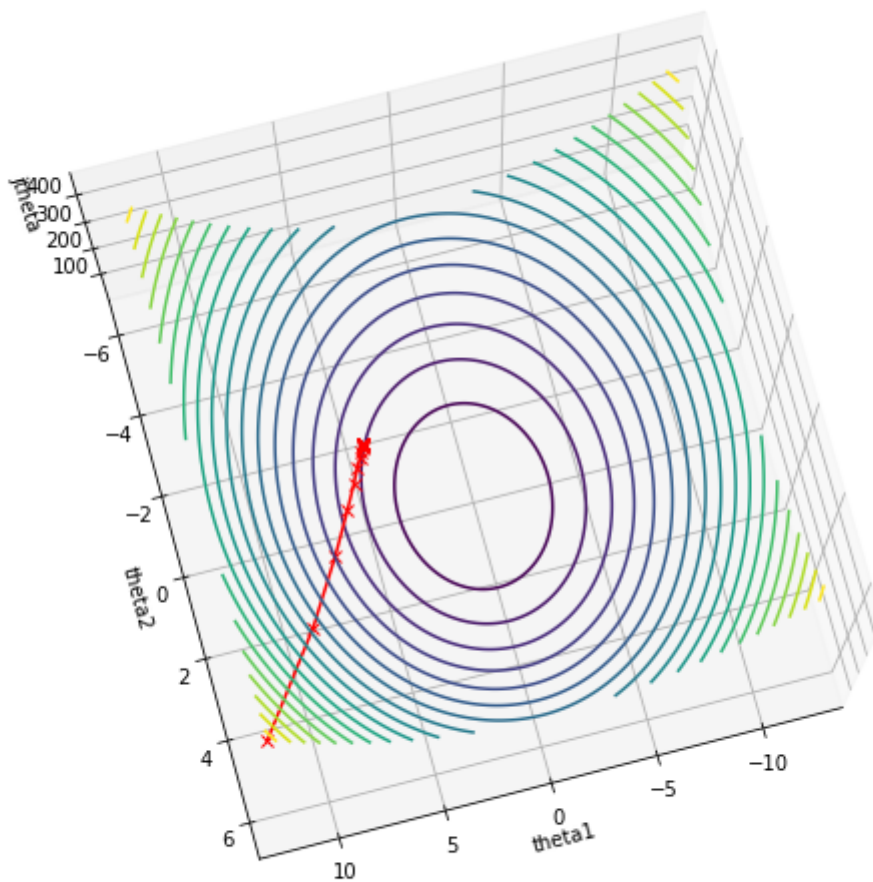
假设损失函数为 $J(\theta) = 2\theta_1^2 + 5\theta_2^2$, 求此式子下，使 $J(\theta)$ 的参数 θ_1, θ_2

那么我们进行如下操作：

(a) 计算函数的梯度： $\Delta J(\theta) = (4\theta_1, 10\theta_2)$, 是一个向量

(b) 进行梯度下降， $(\theta_1, \theta_2) = (\theta_1, \theta_2) - \alpha * \Delta J(\theta)$ 直到梯度中的每个值都小于 0.01，输出最终终止时候的 (θ_1, θ_2)

计算之后，可以得到效果图如下：



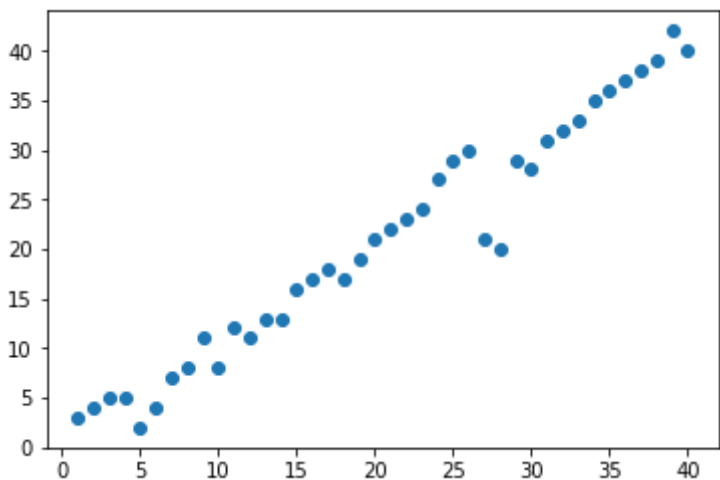
可见函数取值一直在往底部走，越来越接近最低点。

3) 梯度下降算法实战

在实现了一维和二维梯度下降算法之后，我们下面进入到真实的梯度下降算法实践。以线性回归算法为例，假设我们有一系列下面的数据点

$Y = [3, 4, 5, 5, 2, 4, 7, 8, 11, 8,$
 $12, 11, 13, 13, 16, 17, 18, 17, 19, 21,$
 $22, 23, 24, 27, 29, 30, 21, 20, 29, 28,$
 $31, 32, 33, 35, 36, 37, 38, 39, 42, 40]$

如下图所示：



想找到一个线性函数 $h_{\theta}(x^i) = \theta_0 * x_0^i + \theta_1 * x_1^i$, 使得 $h_{\theta}(x)$ 能够尽可能准确的拟合该条曲线, 因此我们有平方误差函数如下:

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

其中 $(h_{\theta}(x^i))$ 是预测值, y^i 是真实值, 我们的目标是得到参数 (θ_0, θ_2) , 使得 $J(\theta)$ 能够最小。跟上面两个例子一样, 先计算梯度, 然后进行梯度下降。

计算梯度如下:

$$\frac{\partial(J(\theta))}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (\theta_0 * x_0^i + \theta_1 * x_1^i - y^i)$$

$$\frac{\partial(J(\theta))}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^m (\theta_0 * x_0^i + \theta_1 * x_1^i - y^i) * x_1^i$$

也是一个向量。这里不一样的是, 有很多个 x , 所以是求和之后的平均。故而 $J(\theta)$ 的梯度为:

$$\text{grad}(J(\theta)) = [\frac{1}{m} \sum_{i=1}^m (\theta_0 * x_0^i + \theta_1 * x_1^i - y^i), \frac{1}{m} \sum_{i=1}^m (\theta_0 * x_0^i + \theta_1 * x_1^i - y^i) * x_1^i]$$

然后我们可以编写代码, 计算一下在 $\alpha = 0.1$ 的时候, 最优化的参数 θ_0, θ_1 分别是多少。

我们知道, 对于二维坐标里的直线来说, $h_{\theta}(x^i)$ 的 x_0 是 1, 即为常数, x_1 才是自变量。一般都是 $y = ax + b$ 的形式, 这里我们写的是一般形式的。但是这样写比较繁琐, 所以我们以矩阵的形式来计算。

$$\text{于是有 } J(\theta) = \frac{1}{2m} (X\theta - Y)^T * (X\theta - Y)$$

$$\frac{\partial(J(\theta))}{\partial \theta} = \frac{1}{2m} X^T (X\theta - Y) + \frac{1}{2m} (X\theta - Y)^T X$$

由于矩阵运算满足: $(XY)^T = Y^T X^T$, 因此对上式第二部分进行转换, 则得到:

$$\frac{1}{2m} (X\theta - Y)^T X = \frac{1}{2m} X^T (X\theta - Y), \text{ 所以我们推得:}$$

$$\frac{\partial(J(\theta))}{\partial \theta} = \frac{1}{m} X^T (X\theta - Y)$$

于是我们得到梯度更新的公式，即为:

$$\theta := \theta - \alpha * \frac{\partial(J(\theta))}{\theta} = \alpha * \frac{1}{m} X^T (X\theta - Y)$$

下面我们将此算法转换成代码如下:

```
#梯度下降算法
def gradient_descent(x,y,alpha,threshold):
    theta=np.array([1,1]).reshape(2,1)
    gradient=gradient_func(x,y,theta)
    while not np.all(np.absolute(gradient) <= threshold):
        theta=theta-alpha*gradient
        gradient=gradient_func(x,y,theta)
    return theta

#计算梯度的函数
def gradient_func(x,y,theta):
    diff=np.dot(x,theta)-y
    grad=(1.0/(x.shape[0]))*np.dot(np.transpose(x),diff)
    return grad

#计算损失的函数
def errors_func(x,y,theta):
    diff=np.dot(x,theta)-y
    error=(1.0/(2*x.shape[0]))*np.dot(np.transpose(diff),diff)
    return error
```

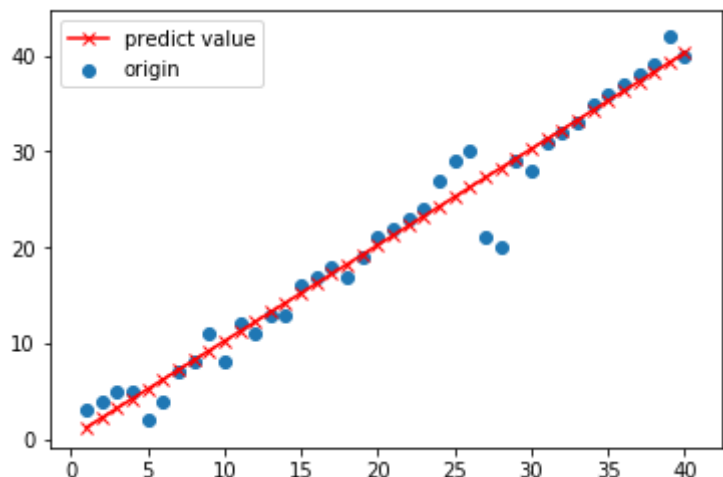
最后得到最佳参数为:

$$[\theta_0, \theta_1] = [0.21158005336805769 \ 1.001874631500372]$$

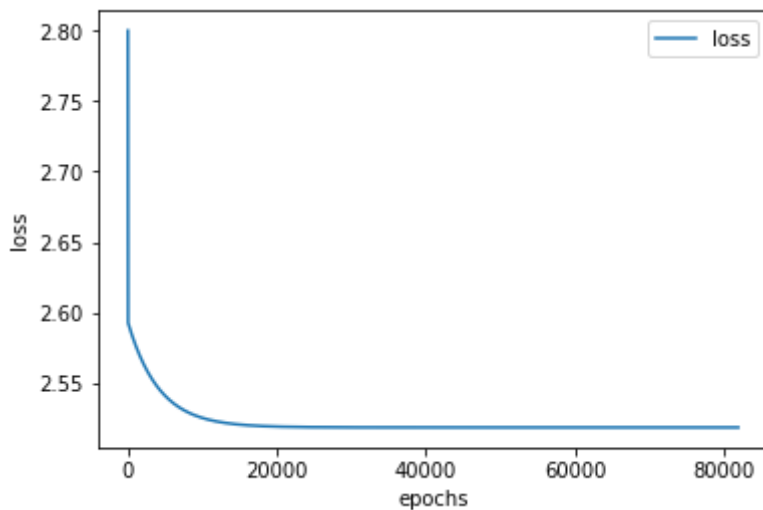
损失函数的值为:

$$J(\theta) = 2.518515478632242$$

可见损失函数很小。最后的拟合函数 $h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1$,拟合的结果如下图所示:



可见拟合的很好。损失函数的变化也如下图所示:



损失函数也一直在往下降。

具体代码示例看 [线性回归梯度下降参数求解](#)。最重要的是将函数式转换成代码。上述算法也叫批量梯度下降算法(BGD)。每次计算梯度下降的时候，用的都是全量的样本数据。

那么我们可以思考一个问题：如果数据量极大，超出了计算机的内存限制，每次都无法装下所有的样本数据，应该去如何计算？

4.小批量梯度下降算法和随机梯度下降算法

小批量梯度下降算法和随机梯度下降算法就是为了解决上述的问题。

1) 随机梯度下降算法(SGD)

随机梯度下降算法，则是每次随机的选择一个样本数据来计算梯度。随机梯度下降算法的损失函数为：

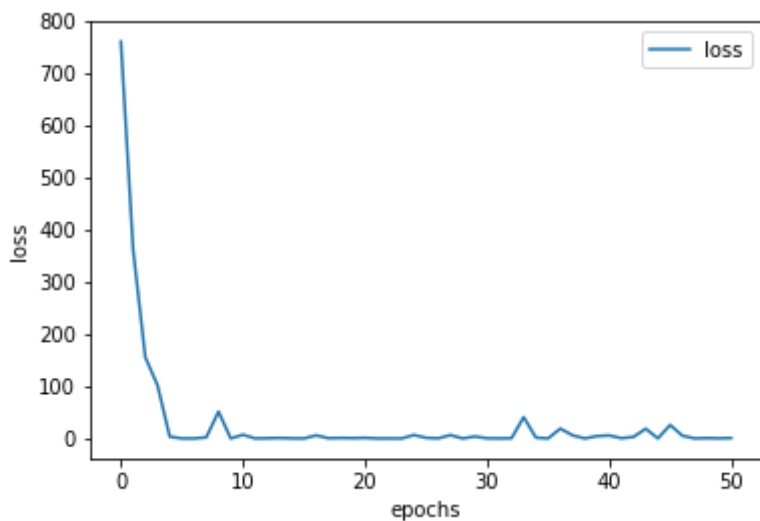
$$J(\theta) = \frac{1}{2}(h_{\theta}(x_i) - y_i)^2$$

其中i为随机选择的样本编号。

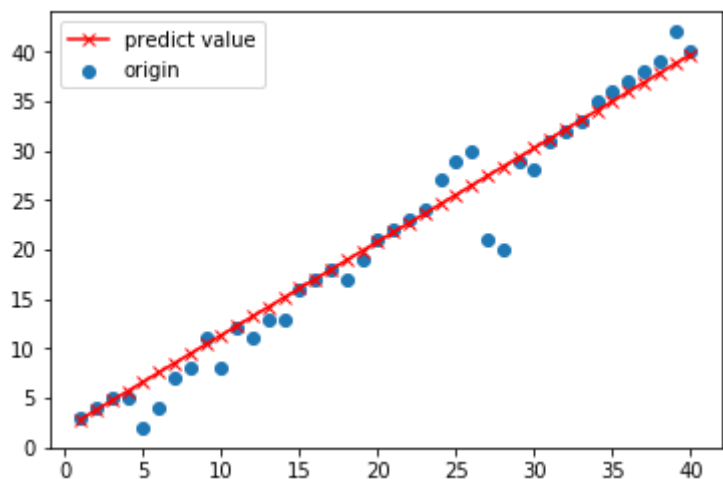
因此梯度更新的方式为：

$$\theta := \theta - \alpha * (x_i * \theta - y_i) * x_i$$

对于上面的数据，其损失函数变化如下：



拟合的结果为：



具体实现看代码示例[随机梯度下降算法实现](#)

2) 小批量梯度下降算法(MBGD)

小批量梯度下降算法(MBGD),是每次只从所有的X集合中, 选择一小部分的样本数据来计算梯度。在数据量很大的情况下, 小批量梯度下降算法就能够比较快的算出梯度。

其损失函数为:

$$J(\theta) = \frac{1}{2L} \sum_{i=t}^{t+L-1} (\theta_0 * x_0^i + \theta_1 * x_1^i - y^i)^2$$

其梯度更新方式为:

$$\theta := \theta - \alpha * \frac{1}{L} \sum_{i=t}^{t+L-1} (x_t \theta - y_t) x_t$$

思考题

- 1) 实现MBGD算法，可参考SGD算法
- 2) 分析SGD、MBDG、BGD算法的优缺点。