二、单变量线性回归

二、单变量线性回归(Linear Regression with One Variable)

2.1 模型表示

我将在整个课程中用小写的m来表示训练样本的数目。

以之前的房屋交易问题为例，假使我们回归问题的训练集（**Training Set**）如下表所示：

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

我们将要用来描述这个回归问题的标记如下:
m 代表训练集中实例的数量
x 代表特征/输入变量
y 代表目标变量/输出变量
(x, y) 代表训练集中的实例
$(x^{(i)}, y^{(i)})$ 代表第i 个观察实例
h 代表学习算法的解决方案或函数也称为假设（**hypothesis**）

这就是一个监督学习算法的工作方式，我们可以看到这里有我们的训练集里房屋价格
我们把它喂给我们的学习算法，学习算法的工作了，然后输出一个函数，通常表示为小写 h 表示。h 代表**hypothesis(假设)**，h表示一个函数，输入是房屋尺寸大小，就像你朋友想出售的房屋，因此 h 根据输入的 x值来得出 y 值，y 值对应房子的价格 因此，h 是一个从x 到 y 的函数映射。

我将选择最初的使用规则h代表**hypothesis**，因而，要解决房价预测问题，我们实际上是要将训练集“喂”给我们的学习算法，进而学习得到一个假设h，然后将我们要预测的房屋的尺寸作为输入变量输入给h，预测出该房屋的交易价格作为输出变量输出为结果。那么，对于我们的房价预测问题，我们该如何表达 h?

一种可能的表达方式为：$h_\theta(x) = \theta_0 + \theta_1 x$，因为只含有一个特征/输入变量，因此这样的问题叫作单变量线性回归问题。

2.2 代价函数

在这段视频中我们将定义代价函数的概念，这有助于我们弄清楚如何把最有可能的直线与我们的数据相拟合。如图：

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

$m = 47$

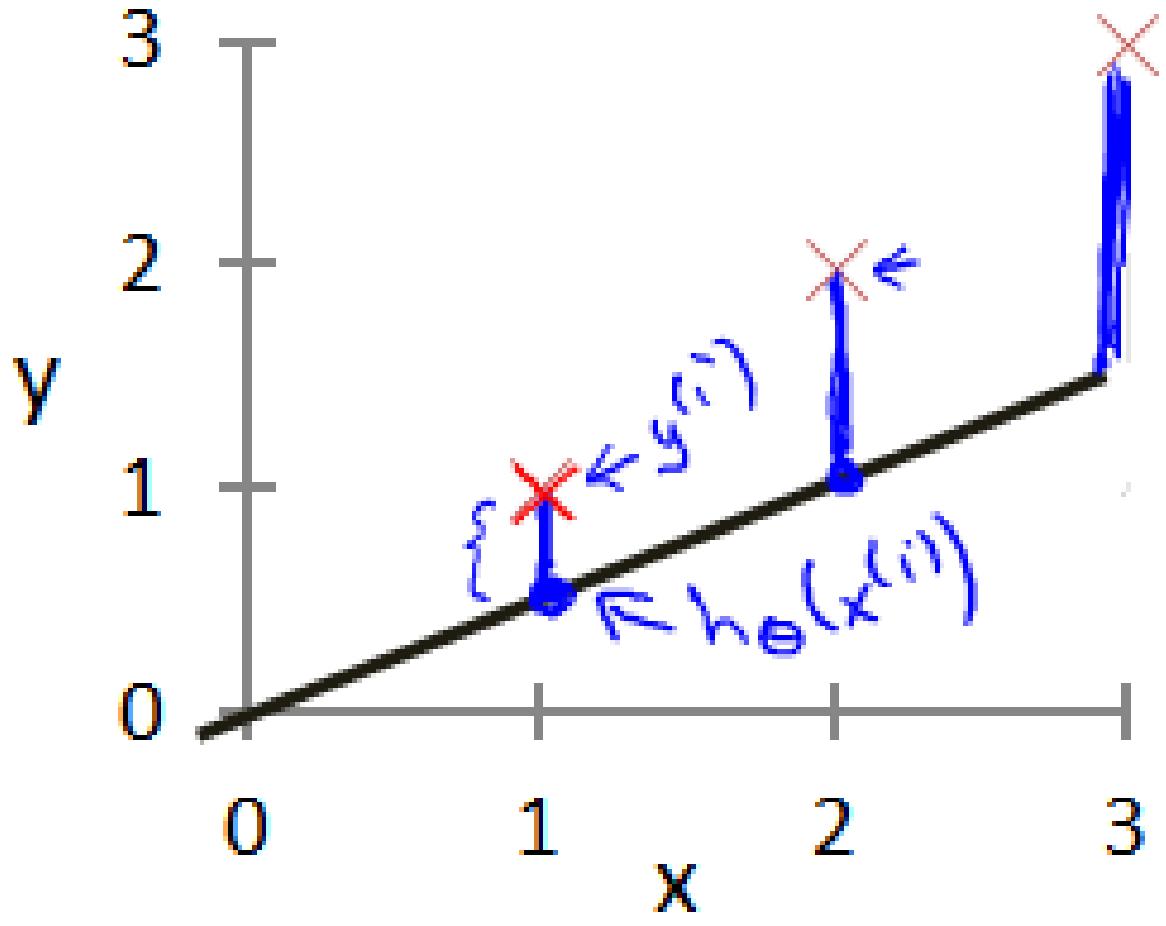
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's: Parameters

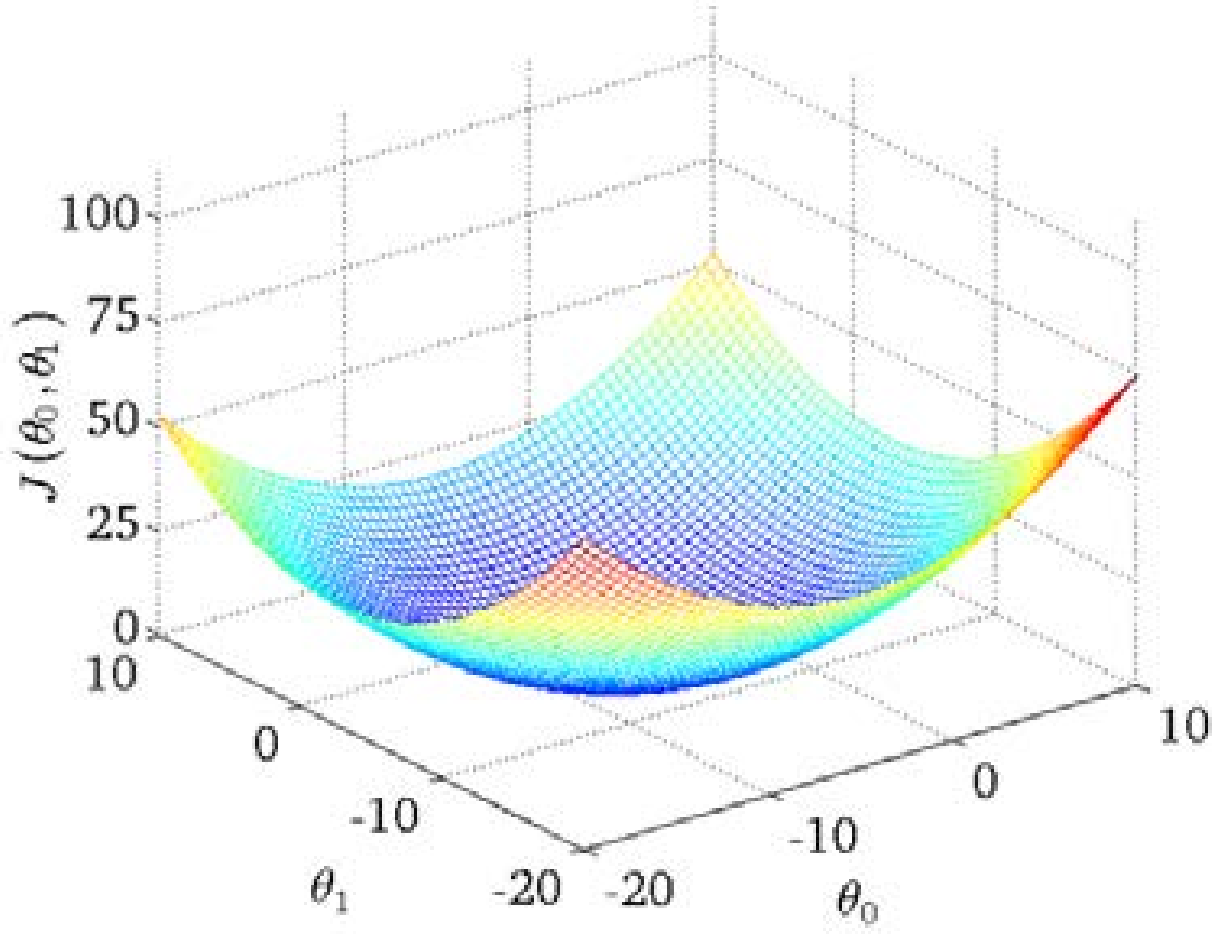
How to choose θ_i's ?

为我们的模型选择合适的**参数**（**parameters**）θ_0 和 θ_1，在房价问题这个例子中便是直线的斜率和在y轴上的截距。

我们选择的参数决定了我们得到的直线相对于我们的训练集的准确程度，模型所预测的值与训练集中实际值之间的差距（下图中蓝线所指）就是**建模误差**（**modeling error**）。



我们的目标便是选择出可以使得建模误差的平方和能够最小的模型参数。 即使得代价函数
$J\left(\theta_0, \theta_1\right)=\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})-y^{(i)}\right)^2$最小。
我们绘制一个等高线图，三个坐标分别为θ_0和θ_1 和$J(\theta_0, \theta_1)$:



则可以看出在三维空间中存在一个使得$J(\theta_0, \theta_1)$最小的点。
代价函数也被称作**平方误差函数**，有时也被称为**平方误差代价函数**。

2.3 代价函数的直观理解I

在上一个视频中，我们给了代价函数一个数学上的定义。在这个视频里，让我们通过一些例子来获取一些直观的感受，看看代价函数到底是在干什么。

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

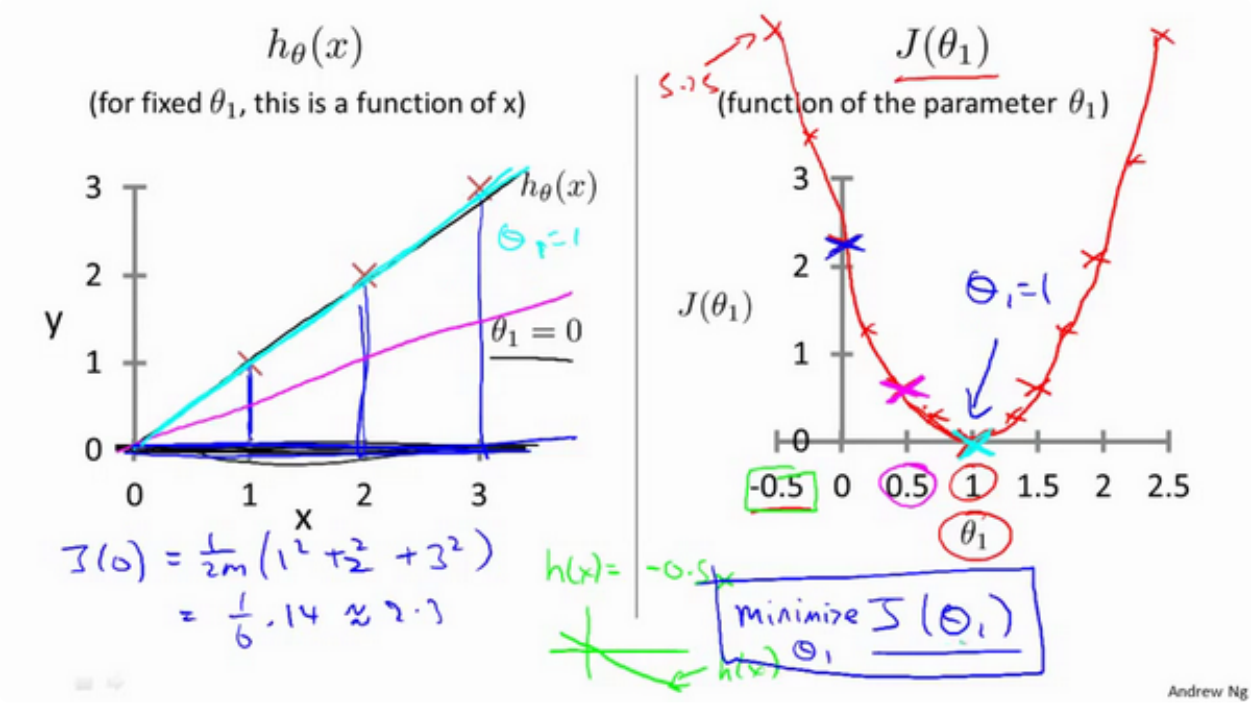
Parameters:

$$\theta_0, \theta_1$$

Cost Function:

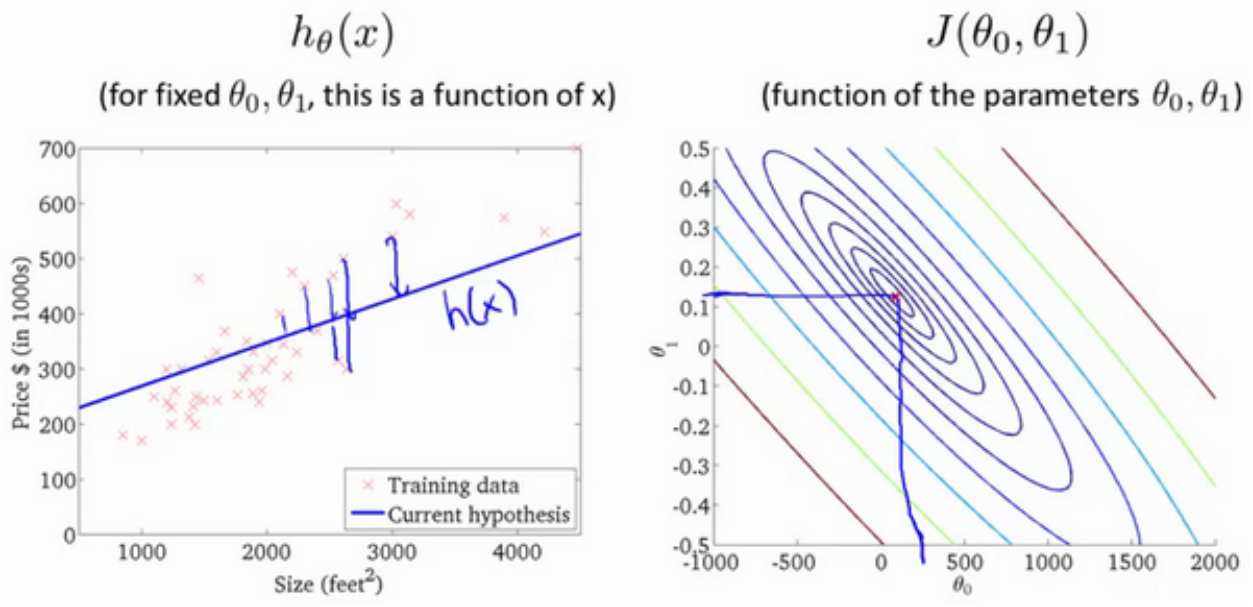
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$



2.4 代价函数的直观理解II

代价函数的样子，等高线图，则可以看出在三维空间中存在一个使得$J(\theta_0, \theta_1)$最小的点。

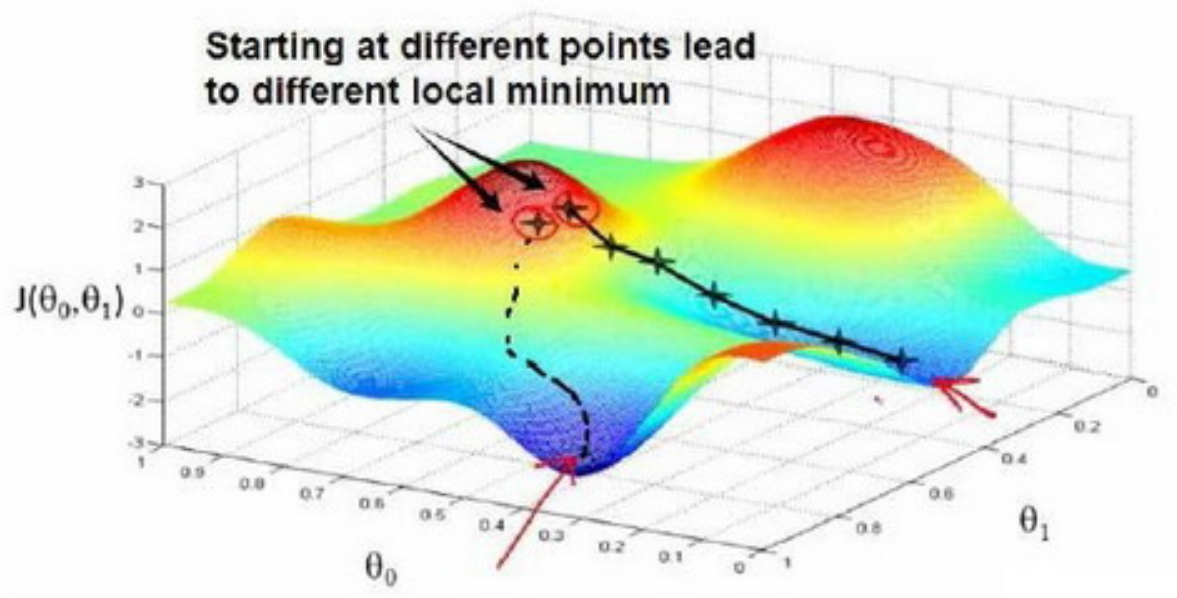


真正需要的是一种有效的算法，能够自动地找出这些使代价函数J取最小值的参数θ_0和θ_1来。

2.5 梯度下降

梯度下降是一个用来求函数最小值的算法，我们将使用梯度下降算法来求出代价函数$J(\theta_0, \theta_1)$ 的最小值。

梯度下降背后的思想是：开始时我们随机选择一个参数的组合$(\theta_0, \theta_1, ..., \theta_n)$，计算代价函数，然后我们寻找下一个能让代价函数值下降最多的参数组合。我们持续这么做直到找到一个局部最小值（**local minimum**），因为我们并没有尝试完所有的参数组合，所以不能确定我们得到的局部最小值是否便是全局最小值（**global minimum**），选择不同的初始参数组合，可能会找到不同的局部最小值。



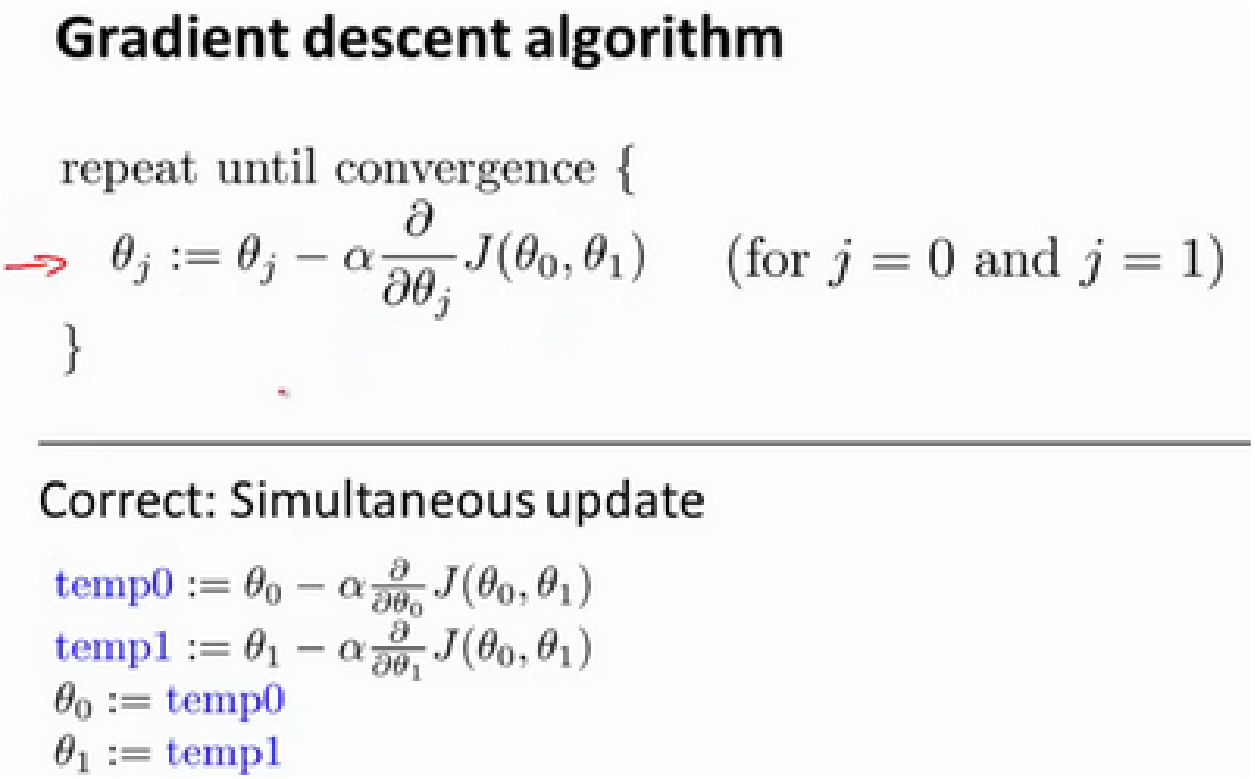
批量梯度下降（**batch gradient descent**）算法的公式为：

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

其中a是学习率（**learning rate**），它决定了我们沿着能让代价函数下降程度最大的方向向下迈出的步子有多大，在批量梯度下降中，我们每一次都同时让所有的参数减去学习速率乘以代价函数的导数。

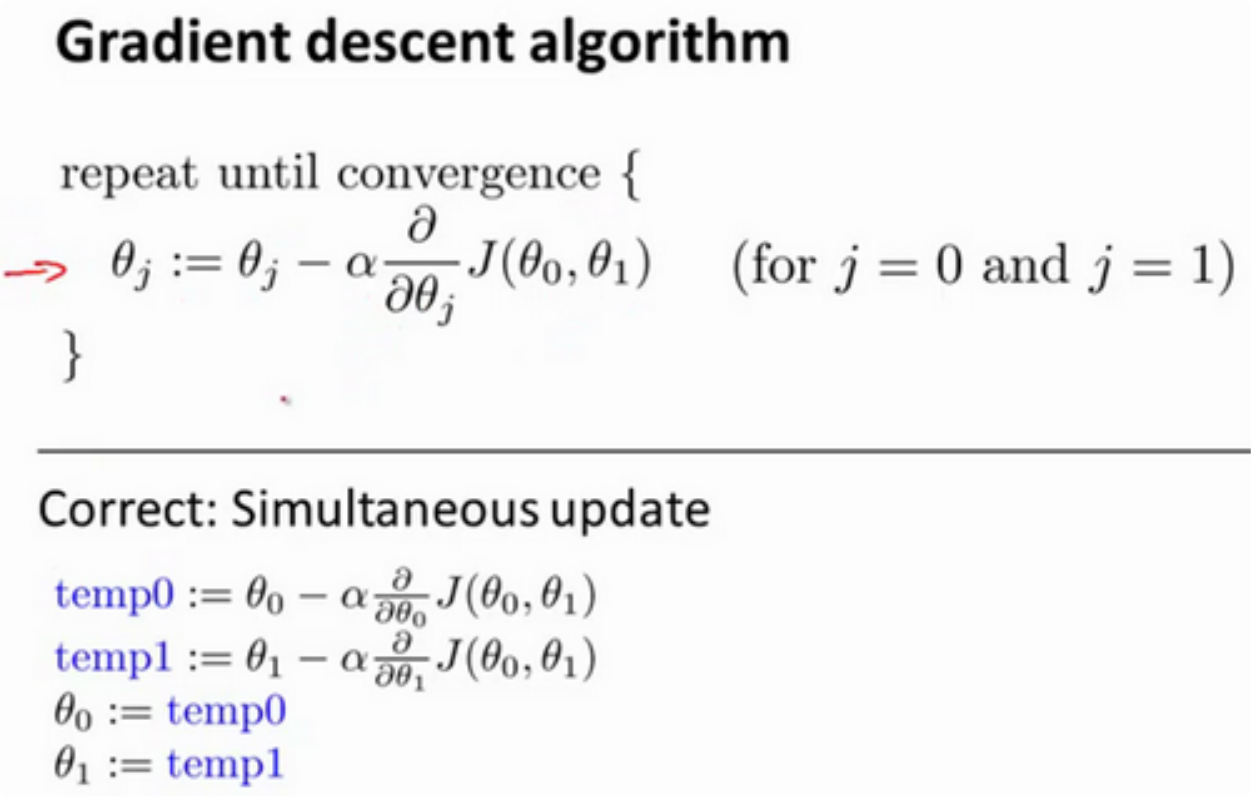


需要同时更新θ_0和θ_1，我的意思是在这个等式中，我们要这样更新：

θ_0:= θ_0 ，并更新θ_1:= θ_1。

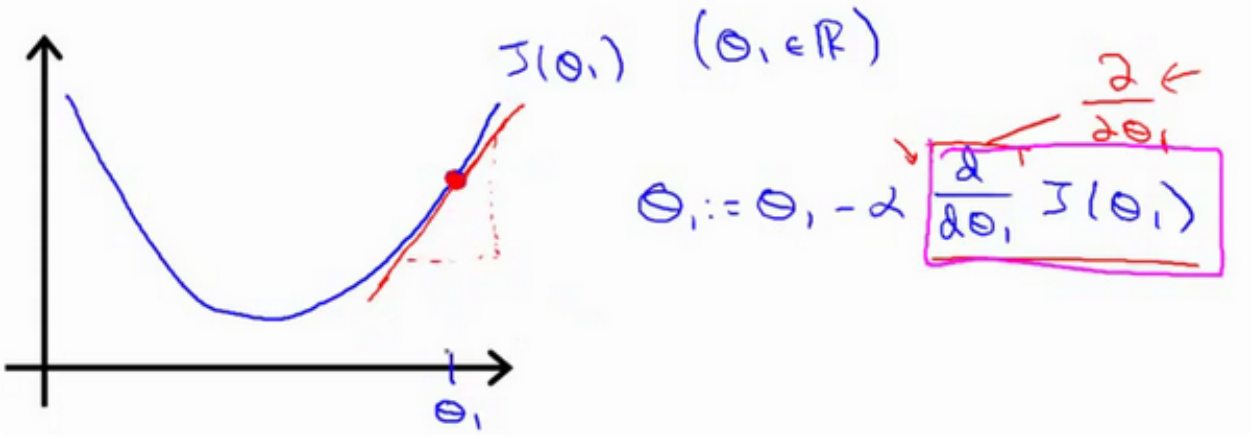
实现方法是：你应该计算公式右边的部分，通过那一部分计算出θ_0和θ_1的值，然后同时更新θ_0和θ_1。

让我进一步阐述这个过程：



2.6 梯度下降的直观理解

梯度下降算法如下：
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
描述：对a赋值，使得θ_j按梯度下降最快方向进行，一直迭代下去，最终得到局部最小值。其中a是学习率（**learning rate**），它决定了我们沿着能让代价函数下降程度最大的方向向下迈出的步子有多大。



对于这个问题，求导的目的，基本上可以说取这个红点的切线，就是这样一条红色的直线，刚好与函数相切于这一点，让我们看看这条红色直线的斜率，就是这条刚好与函数曲线相切的这条直线，这条直线的斜率正好是这个三角形的高度除以这个水平长度，现在，这条线有一个正斜率，也就是说它有正导数，因此，我得到的新的θ_1，θ_1更新后等于θ_1减去一个正数乘以a。

a太小，可能会很慢，因为它会一点点挪动，它会需要很多步才能到达全局最低点。
a太大，它会导致无法收敛，甚至发散。
假设你将θ_1初始化在局部最低点，在这儿，它已经在一个局部的最优处或局部最低点。结果是局部最优点的导数将等于零，因为它是那条切线的斜率。这也解释了为什么即使学习速率a保持不变时，梯度下降也可以收敛到局部最低点。

在梯度下降法中，当我们接近局部最低点时，梯度下降法会自动采取更小的幅度，这是因为当我们接近局部最低点时，很显然在局部最低时导数等于零，所以当我们接近局部最低时，导数值会自动变得越来越小，所以梯度下降将自动采取较小的幅度，这就是梯度下降的做法。所以实际上没有必要再另外减小a。

2.7 梯度下降的线性回归

梯度下降算法和线性回归算法比较如图：

Gradient descent algorithm	Linear Regression Model

repeat until convergence {
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
(for $j = 1$ and $j = 0$)
}
	$h_\theta(x) = \theta_0 + \theta_1 x$
	$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

对我们之前的线性回归问题运用梯度下降法，关键在于求出代价函数的导数，即：

$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

$j = 0$ 时：$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

$j = 1$ 时：$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(\left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}\right)$

则算法改写成：

Repeat {

$\theta_0 := \theta_0 - a \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

$\theta_1 := \theta_1 - a \frac{1}{m} \sum_{i=1}^{m} \left(\left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}\right)$

}

"**批量梯度下降**"，指的是在梯度下降的每一步中，我们都用到了所有的训练样本，在梯度下降中，在计算微分求导项时，我们需要进行求和运算，所以，在每一个单独的梯度下降中，我们最终都要计算这样一个东西，这个项需要对所有m个训练样本求和。因此，批量梯度下降法这个名字说明了我们需要考虑所有这一"批"训练样本。

有一种计算代价函数J最小值的数值解法，不需要梯度下降这种迭代算法。在后面的课程中，我们也会谈到这个方法，它可以在不需要多步梯度下降的情况下，也能解出代价函数J的最小值，这是另一种称

为正规方程(normal equations)的方法。实际上在数据量较大的情况下，梯度下降法比正规方程要更适用一些。