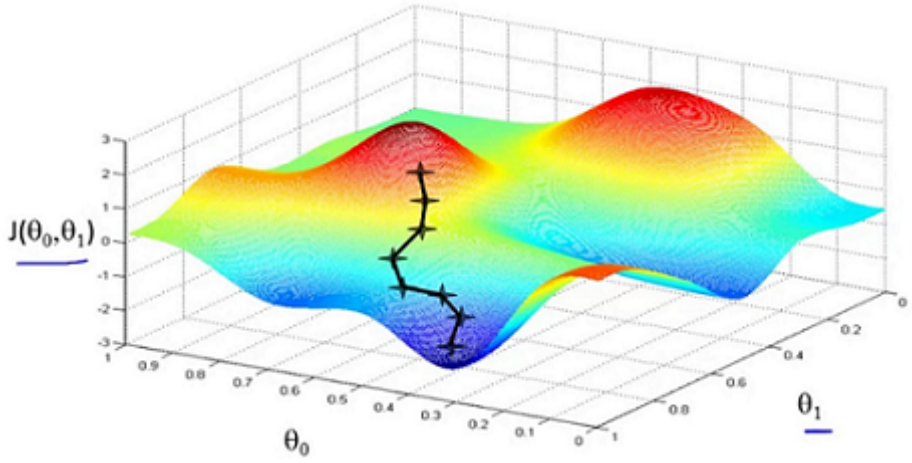
Gradient Descent Algorithm

It turns out gradient descent is a more general algorithm, and is used not only in linear regression. It's actually used all over the place in machine learning.

Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$



Outline

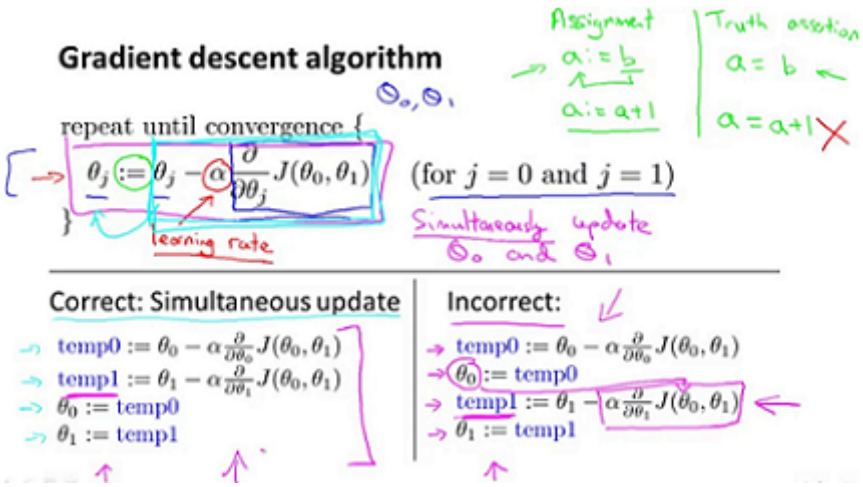
- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum.

Gradient descent applies to more general functions:

$J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$

$minJ(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$

In gradient descent we're going to update the parameter θ_j until convergence. Update θ_j by setting (notation := denotes assignement) it to $(\theta_j - \alpha)$ times the partial derivative of the cost function with respect to θ_j. It turns out that the way gradient descent is usually implemented, it actually turns out to be more natural to implement the simultaneous updates.



Alpha is a number that is called the learning rate and what alpha does is it basically controls how big a step we take downhill with creating descent. So if alpha is very large, then that corresponds to a very aggressive gradient descent procedure where we're trying take huge steps downhill and if alpha is very small, then we're taking little baby steps downhill.

Video Question: Suppose $\theta_0 = 1, \theta_1 = 2$, and we simultaneously update θ_0 and θ_1 using the rule: $\theta_j := \theta_j + \sqrt{\theta_0 \theta_1}$ (for $j = 0$ and $j = 1$). What are the resulting values of θ_0 and θ_1?

- $\theta_0 = 1, \theta_1 = 2$

> $\theta_0 = 1 + \sqrt{2}, \theta_1 = 2 + \sqrt{2}$

- $\theta_0 = 2 + \sqrt{2}, \theta_1 = 1 + \sqrt{2}$
- - $\theta_0 = 1 + \sqrt{2}, \theta_1 = 2 + \sqrt{(1 + \sqrt{2}) \cdot 2}$

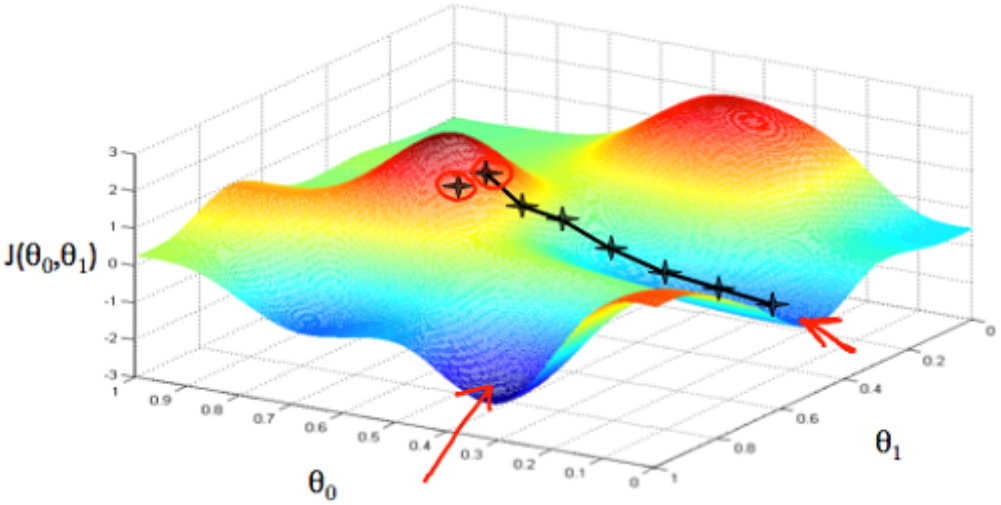
Solution: $\theta_0 := 1 + \sqrt{1 \cdot 2}, \theta_1 := 2 + \sqrt{1 \cdot 2}$

Summary

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields θ_0 and θ_1 (actually we are graphing the cost function as a function of the parameter estimates). We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters.

We put θ_0 on the x axis and θ_1 on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters. The graph below depicts such a setup.



We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum. The red arrows show the minimum points in the graph. The way we do this is by taking the derivative (the tangential line to a function) of our cost function.

The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter α, which is called the learning rate.

For example, the distance between each 'star' in the graph above represents a step determined by our parameter α. A smaller α would result in a smaller step and a larger α results in a larger step. The direction in which the step is taken is determined by the partial derivative of $J(\theta_0, \theta_1)$.

Depending on where one starts on the graph, one could end up at different points. The image above shows us two different starting points that end up in two different places.

The gradient descent algorithm is:

repeat until convergence: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

where $j = 0, 1$ represents the feature index number.

At each iteration j, one should simultaneously update the parameters $\theta_1, \theta_2, \ldots, \theta_n$. Updating a specific parameter prior to calculating another one on the $j^{(th)}$ iteration would yield to a wrong implementation.