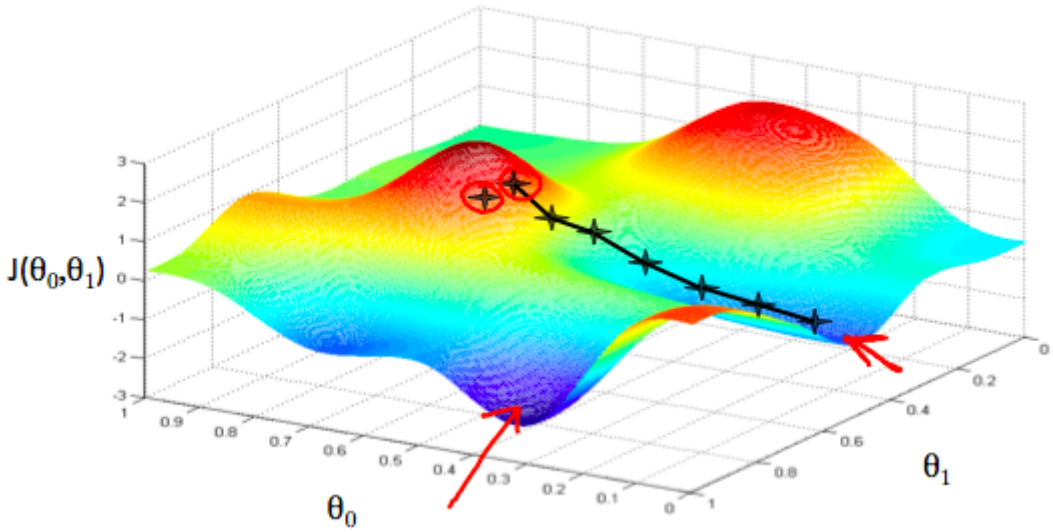
3. Parameter Learning

3.1 Gradient Descent

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:

You can actually use the markdown mode for the cell and use the normal HTML code, as in

Repeat until convergence:

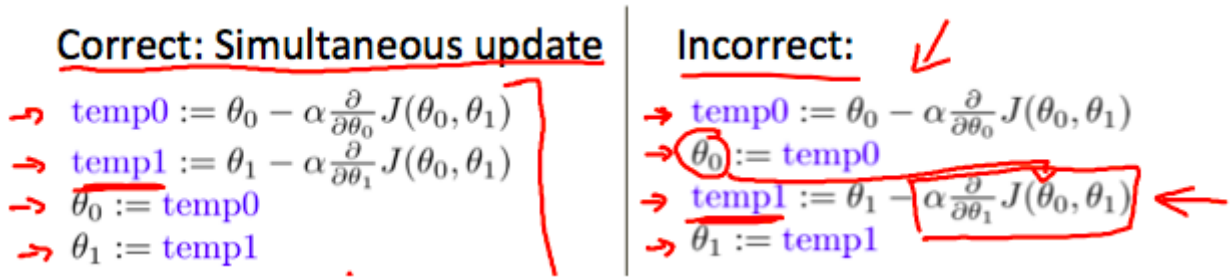
{

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

}

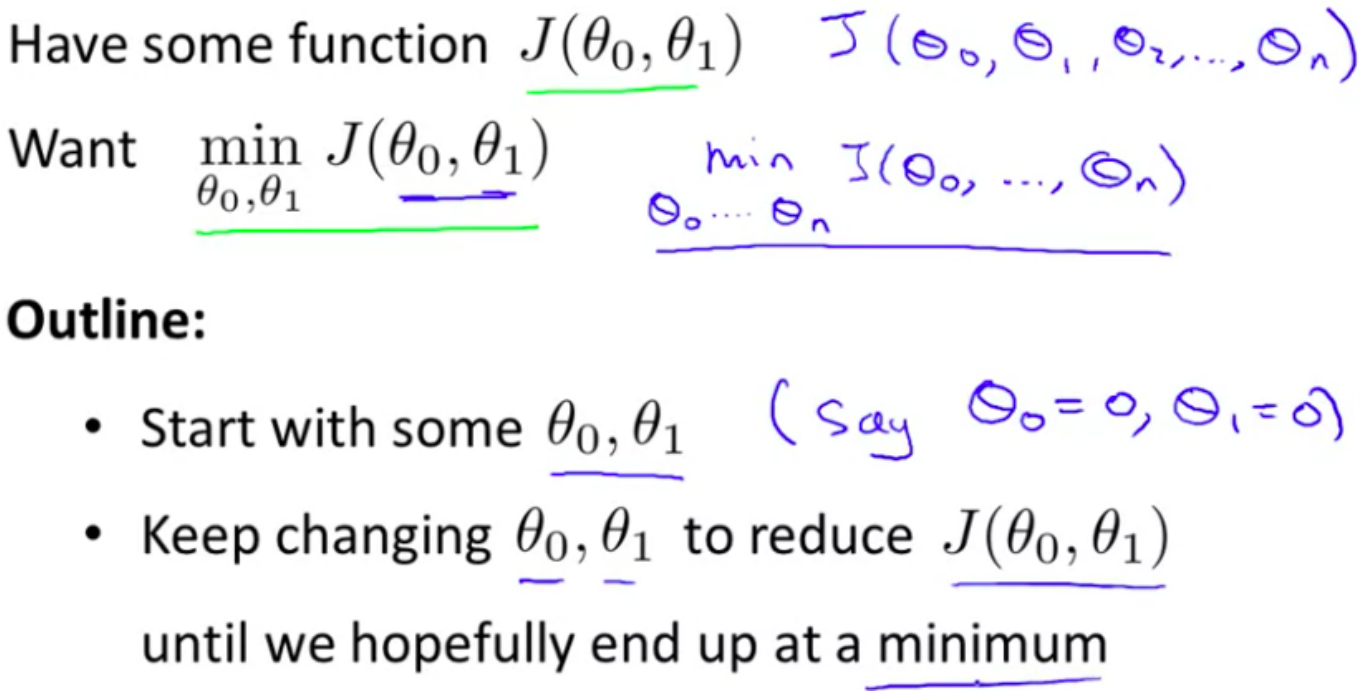
where $j = 0, 1$ represents the feature index number.

At each iteration j, one should simultaneously update the parameters θ_1, θ_2,...,θ_n. Updating a specific parameter prior to calculating another one on the j^{th} iteration would yield a wrong implementation:



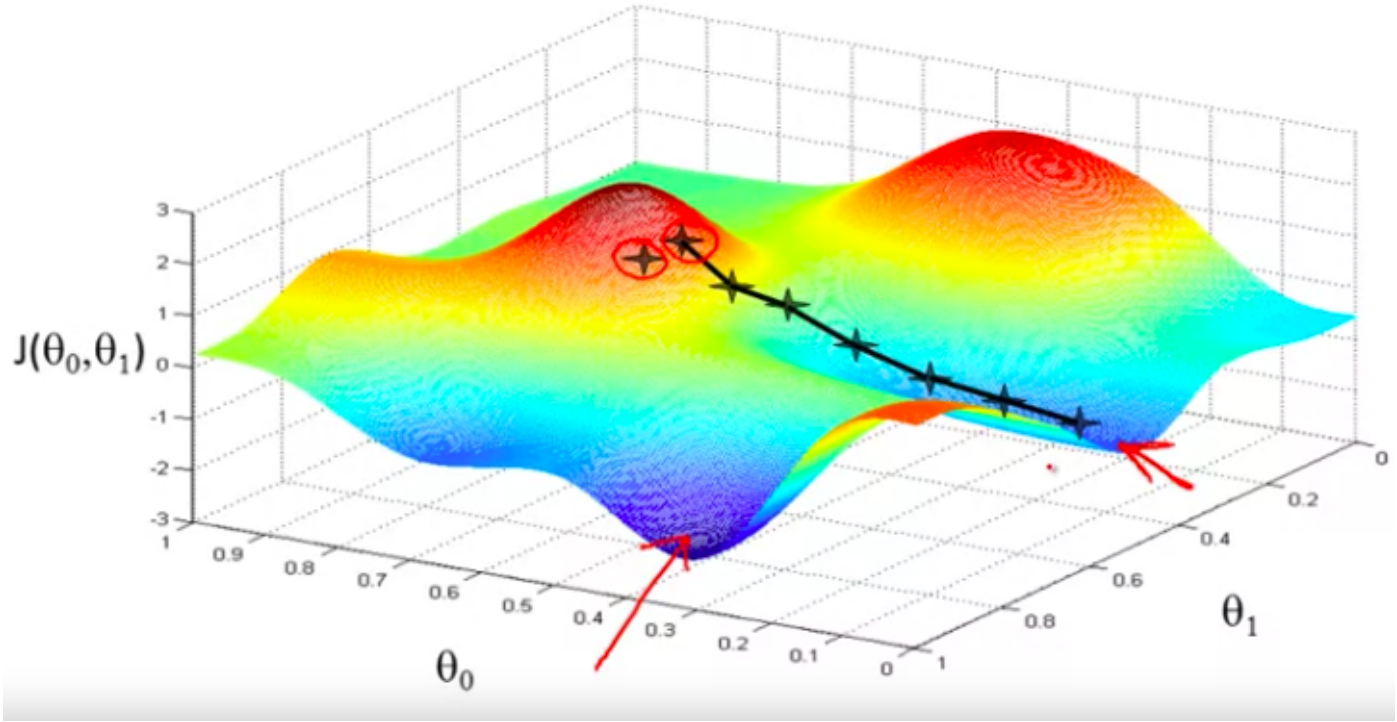
The gradient descent is a general ML algorithm, not used just in linear regression. We will later use it to minimise other functions as well (not just the cost function J for the linear regression; i.e. J can be an arbitrary function that the gradient descent is able to minimise).

To recap the problem solved with gradient descent algorithm (note: only two parameters are considered for the sake of brevity):



The final minimum the algorithm converges to might be a local minimum.

The gradient descent algorithm has an interesting property: depending on how the parameters get initialised it might converge to different local optima (note the different positions of the two red arrows):



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 ?

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

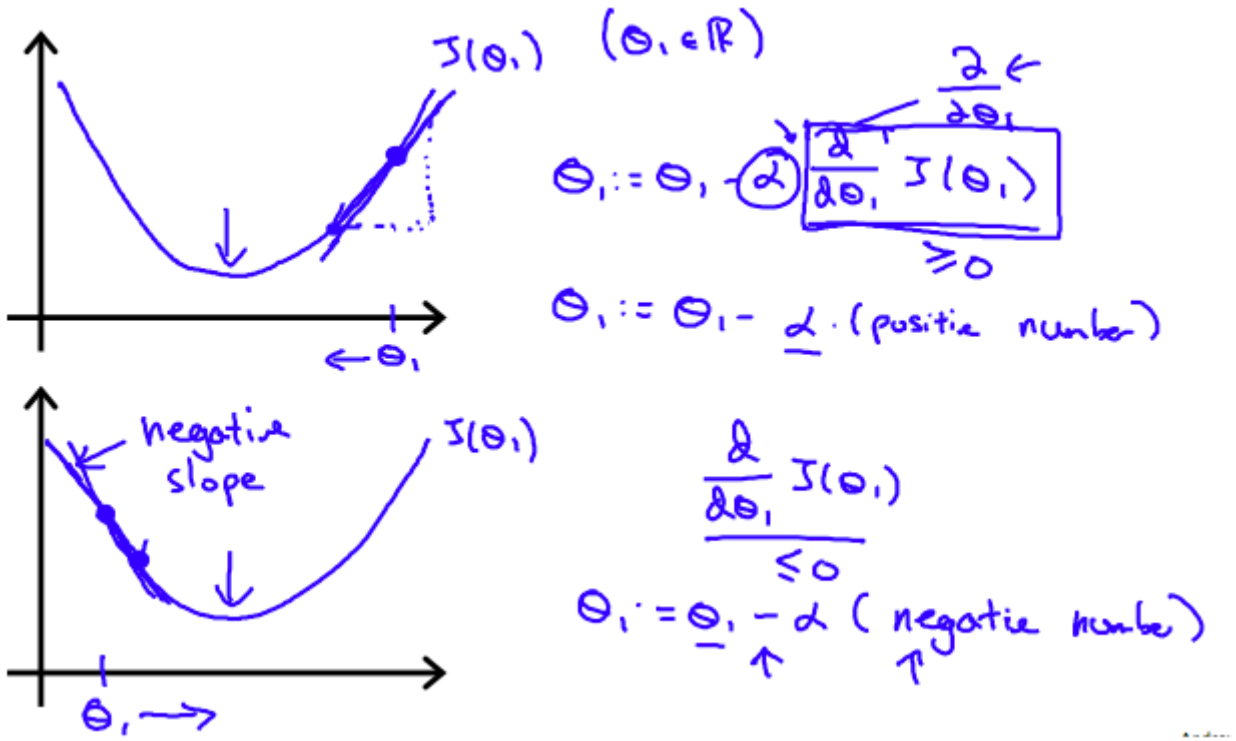
3.2 Gradient Descent Intuition

In this video we explored the scenario where we used one parameter θ_1 and plotted its cost function to implement a gradient descent.

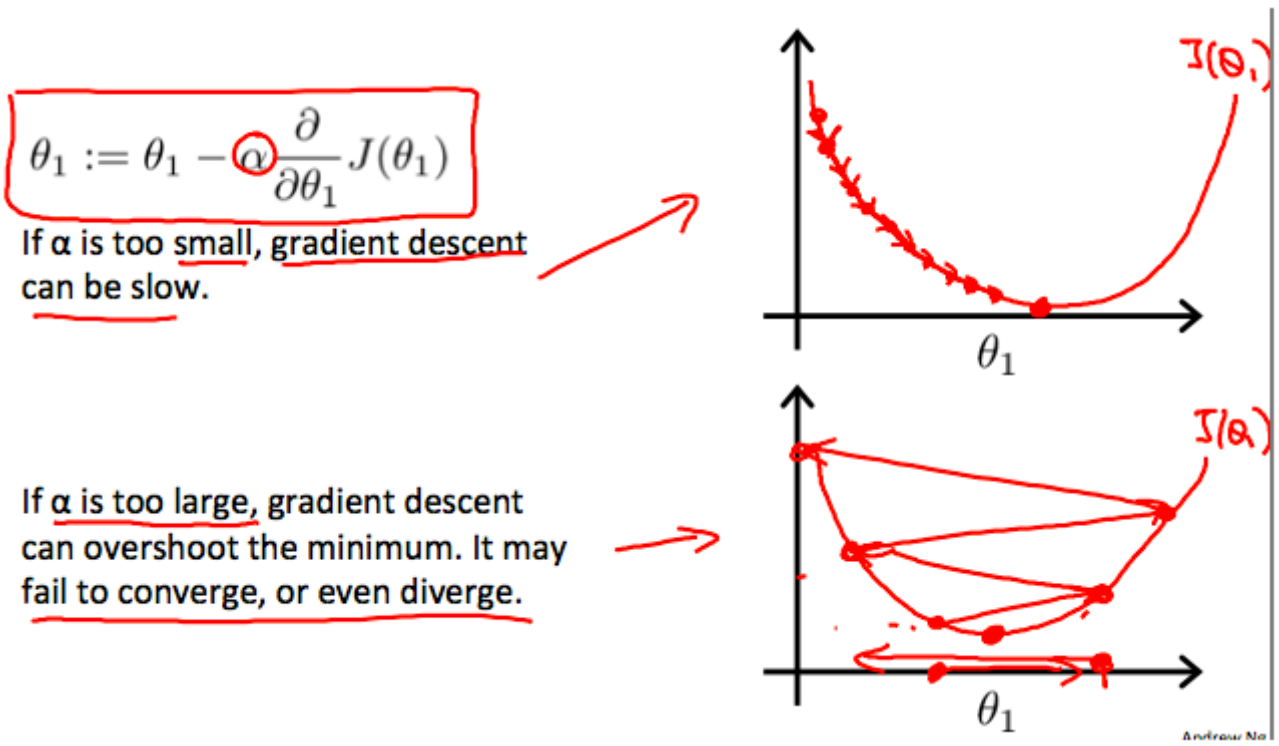
Our formula for a single parameter was:

Repeat until convergence:
{
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$
}

Regardless of the slope's sign for $\frac{\partial}{\partial \theta_1} J(\theta_1)$, θ_1 eventually converges to its minimum value. The following graph shows that when the slope is negative, the value of θ_1 increases and when it is positive, the value of θ_1 decreases:



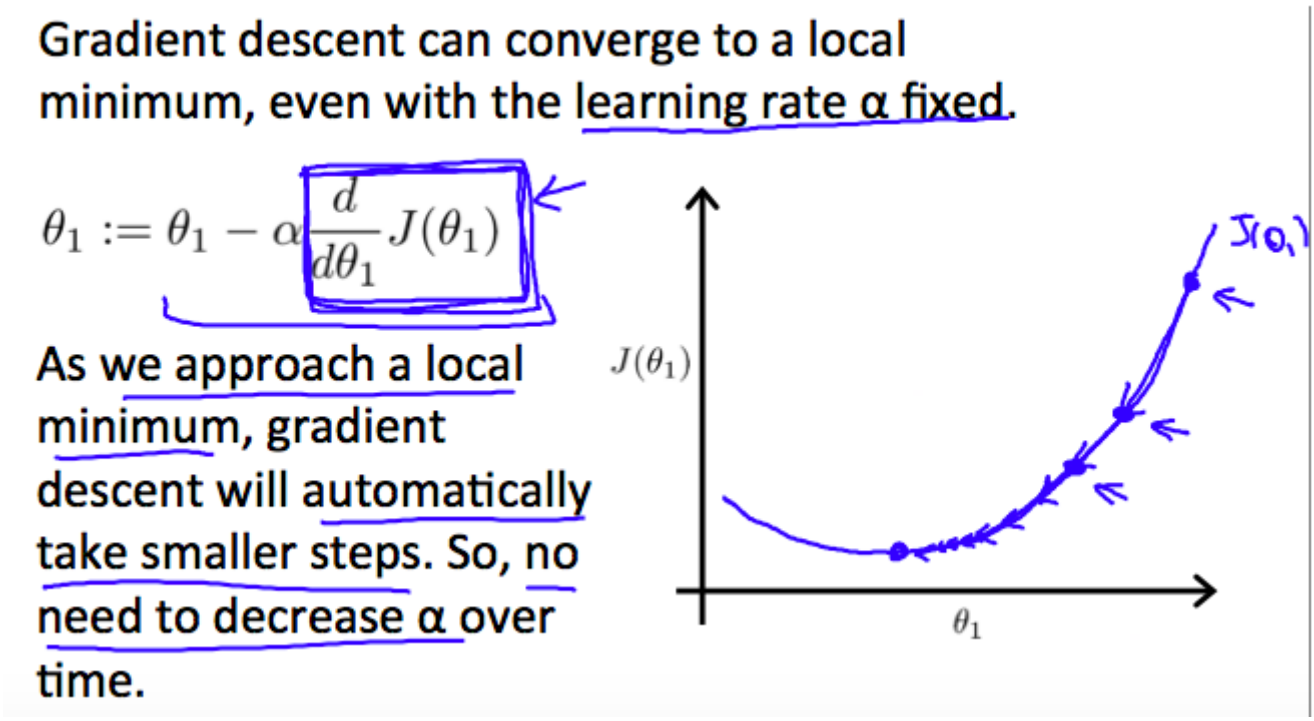
On a side note, we should adjust our parameter α to ensure that the gradient descent algorithm converges in a reasonable time. Failure to converge or too much time to obtain the minimum value imply that our step size is wrong.



How does gradient descent converge with a fixed step size α?

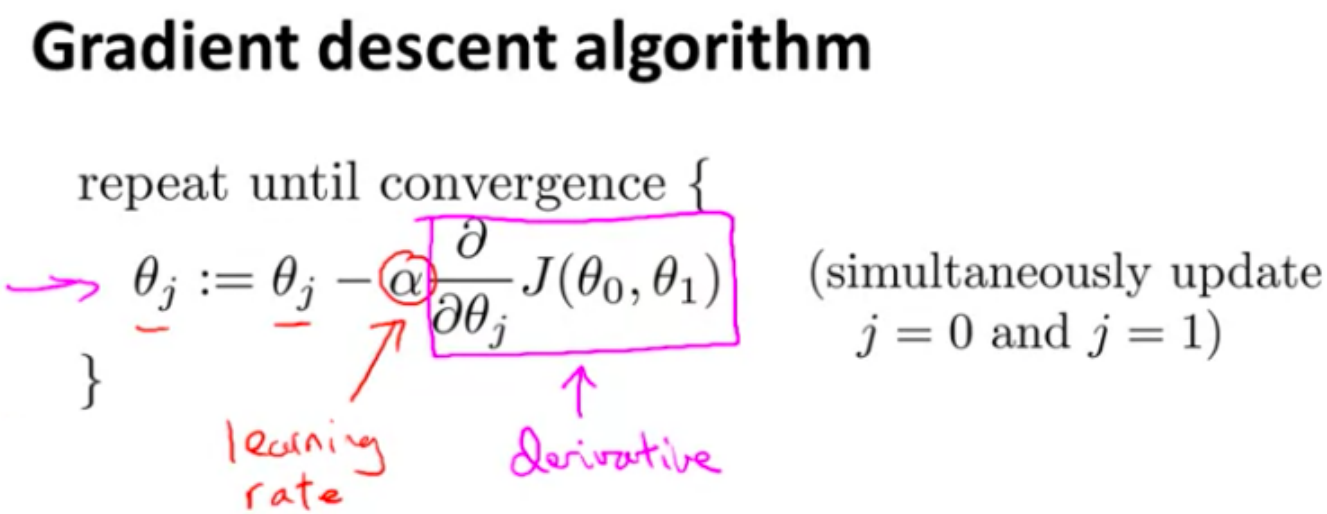
The intuition behind the convergence is that $\frac{\partial}{\partial \theta_1} J(\theta_1)$ approaches 0 as we approach the bottom of our convex function. At the minimum, the derivative will always be 0 and thus we get:

$$\theta_1 := \theta_1 - \alpha * 0$$



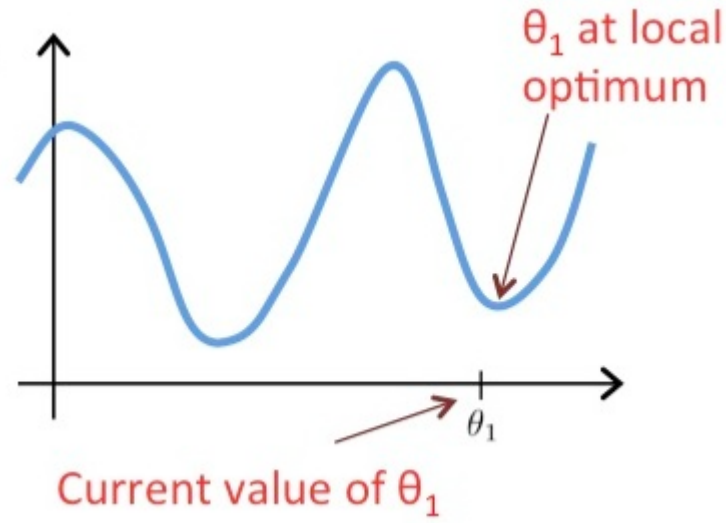
As we approach the local minimum, by definition, the derivative will get smaller and smaller...

To recap:

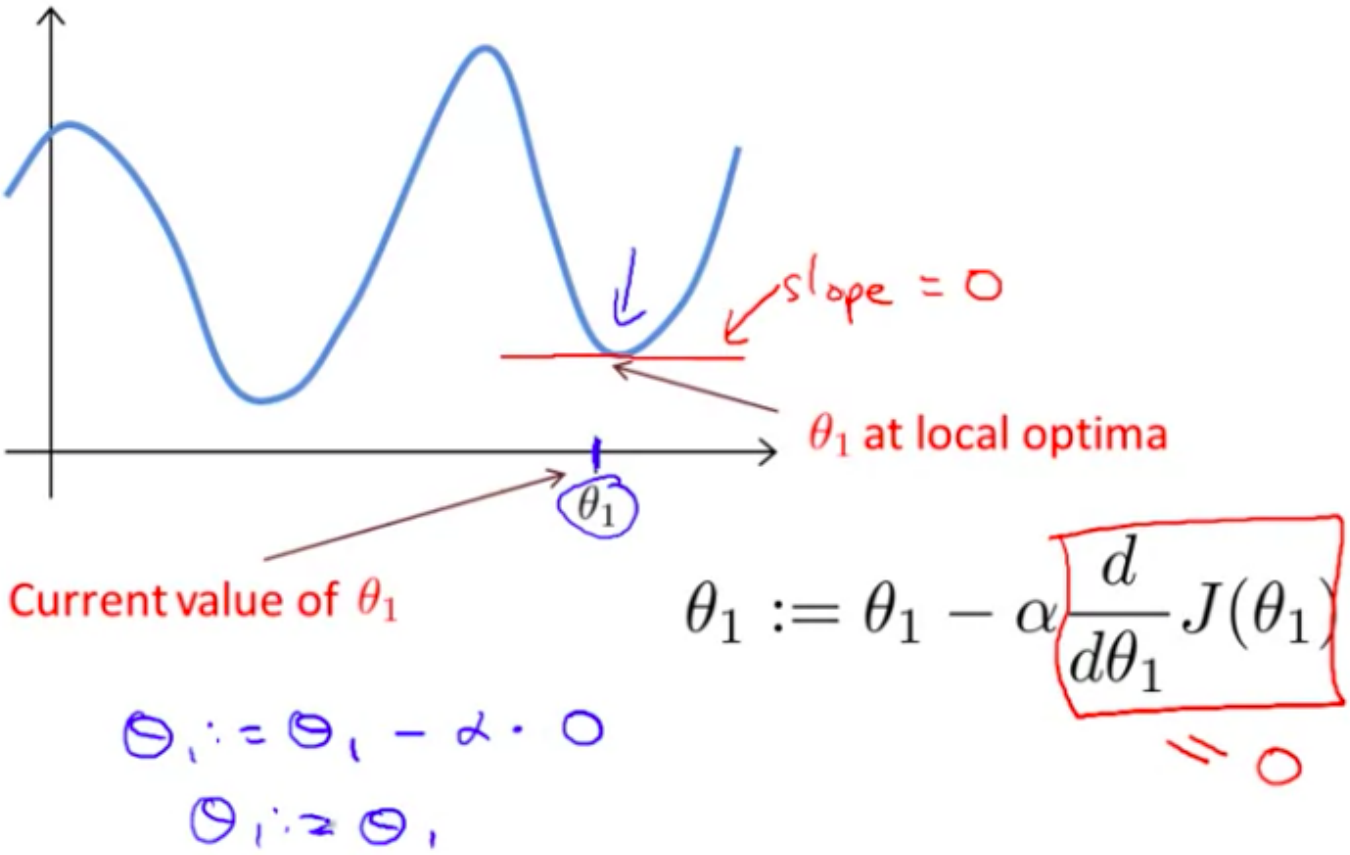


Note: the **learning rate** α is always a **positive number**.

Question: Suppose θ_1 is at a local optimum (i.e. minimum) of $J(\theta_1)$, such as shown in the figure below. What will one step of gradient descent $\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$ do?



Answer: Leave θ_1 unchanged



3.3 Gradient Descent for Linear Regression

Gradient descent algorithm

repeat until convergence {

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

$$(\text{for } j = 1 \text{ and } j = 0)$$

}

Linear Regression Model

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

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

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to:

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left((h_\theta(x_i) - y_i) x_i\right)$$

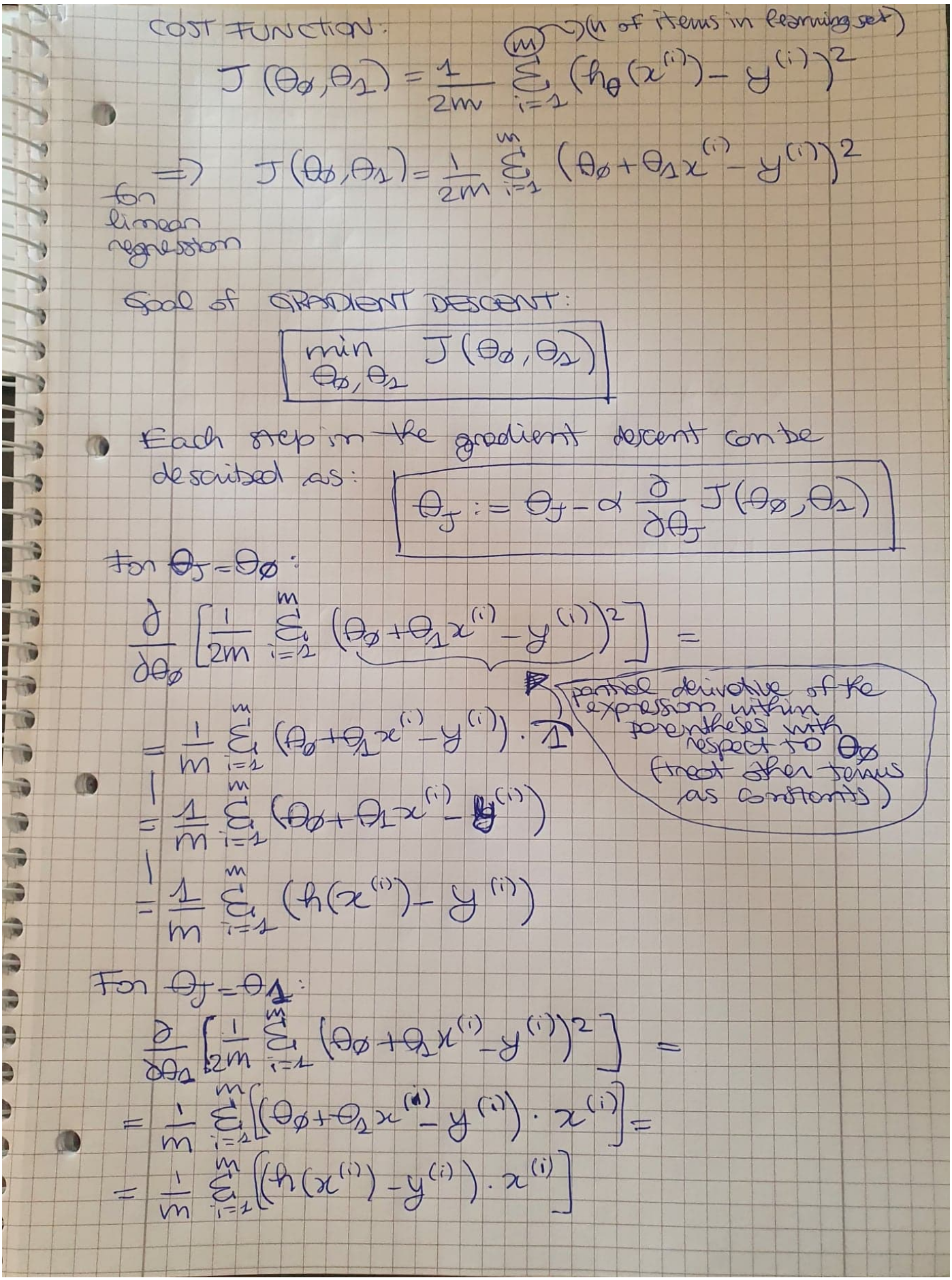
}

where m is the size of the training set, θ_0 is a constant that ill be changing simultaneously with θ_1 , and x_i and y_i are values of the given training set (i.e. the data).

Note that we have separated out the two cases for θ_j into 2 separate equations for θ_0 and θ_1 ; and that for θ_1 we are multiplying x_i at the end due to the derivative. The following is a derivation of $\frac{\partial}{\partial \theta_j} J(\theta)$ for a single example:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y)\, x_j
\end{aligned}$$

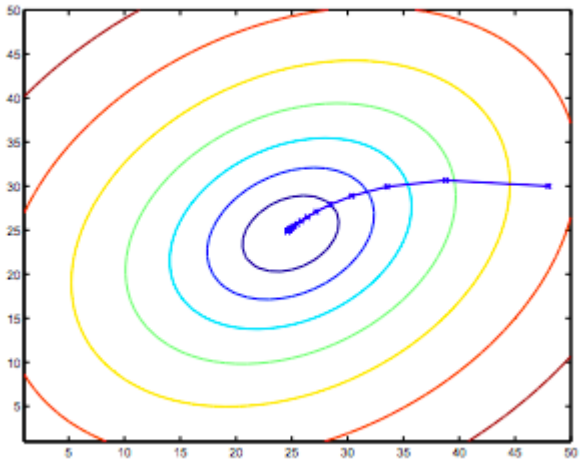
How the partial derivatives of the gradient descent for linear regression are calculated (please refer to this link https://math.stackexchange.com/questions/70728/partial-derivative-in-gradient-descent-for-two-variables (https://math.stackexchange.com/questions/70728/partial-derivative-in-gradient-descent-for-two-variables)):



The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

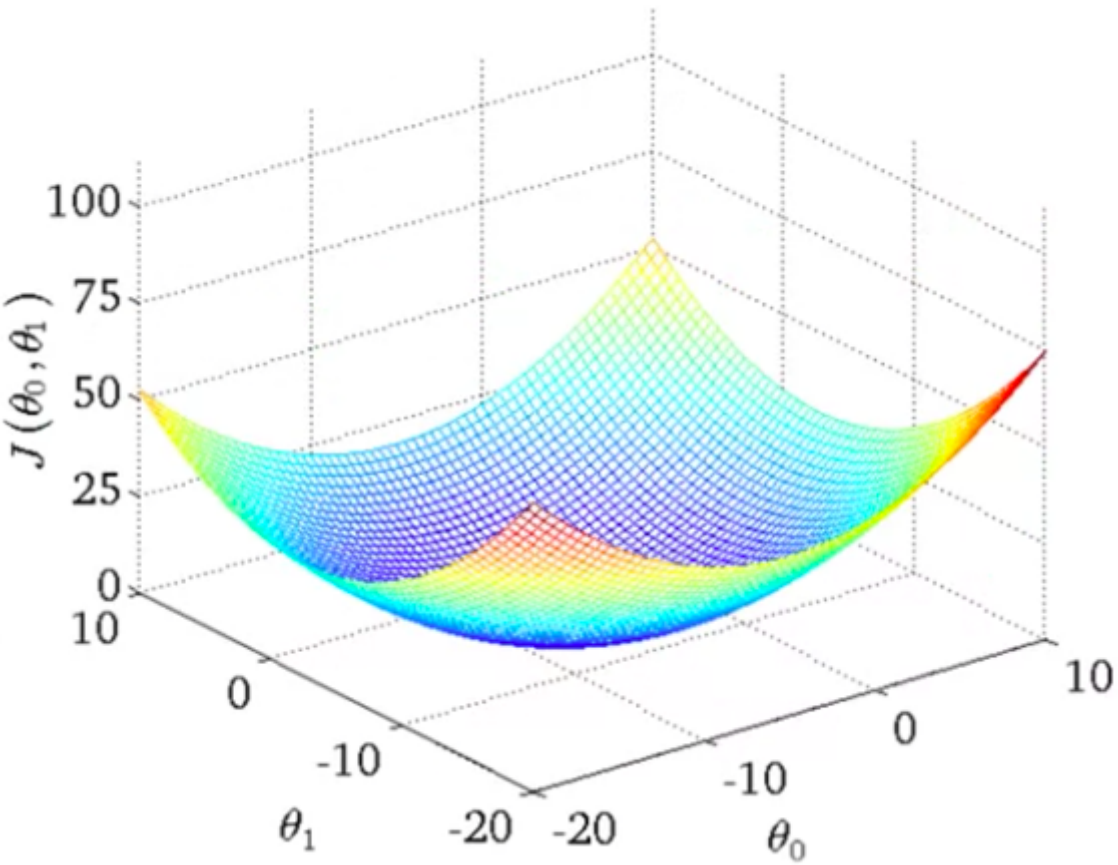
So, this is simply gradient descent on the original cost function J. This method looks at every example in the entire training set on every step, and is called **batch gradient descent**. At each step of the gradient descent algorithm we are computing sums of all training examples to update the parameters of the model. There are other versions of gradient descent that are not batch versions, i.e. they do not look at the entire training set but only at a small subset within it

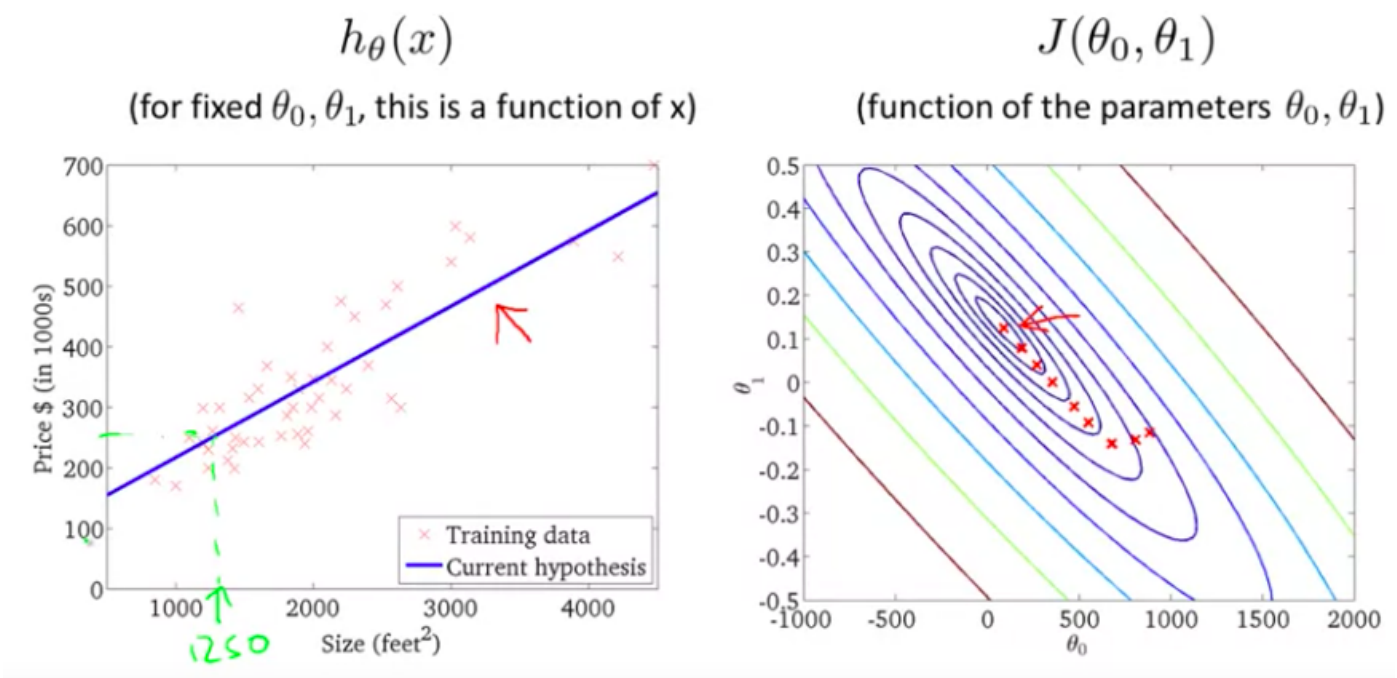
Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a **convex quadratic function**. Here is an example of gradient descent as it is run to minimize a quadratic function.



The ellipses shown above are the contours of a quadratic function. Also shown is the trajectory taken by gradient descent, which was initialized at (48,30). The x's in the figure (joined by straight lines) mark the successive values of θ that gradient descent went through as it converged to its minimum.

The cost function for gradient descent of linear regression is always going to be a bowl-shaped (convex) function and there is **only one global optimum** (no local optima) to which the gradient descent will converge.





Question: Which of the following are true statements?

- To make gradient descent converge, we must slowly decrease α over time: FALSE
- Gradient descent is guaranteed to find the global minimum for any function $J(\theta_0, \theta_1)$: FALSE
- Gradient descent can converge even if α is kept fixed. (But α cannot be too large, or else it may fail to converge.): TRUE
- For the specific choice of cost function $J(\theta_0, \theta_1)$ used in linear regression, there are no local optima (other than the global optimum): TRUE

If you have taken an advanced course in **linear algebra**, you might know that there is a solution for **numerically solving** for the minimum of the cost function J without having to use an iterative algorithm like gradient descent. This method is called the **normal equation method**. However, it turns out that the gradient descent scales better to larger datasets than the normal equation method.