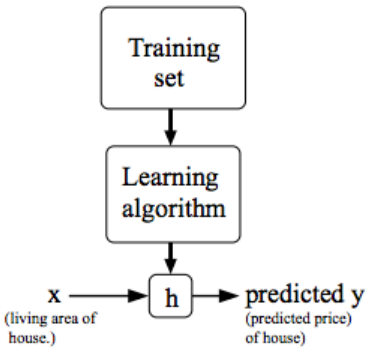
Model Representation
To establish notation for future use, we'll use:

- x(i) to denote the "input" variables (living area in this example), also called input features,
- y(i) to denote the "output" or target variable that we are trying to predict (price).

A pair (x(i),y(i)) is called a training example, and the dataset that we'll be using to learn—a list of m training examples (x(i),y(i));i=1,...,m is called a training set.

Note that the superscript "(i)" in the notation is simply an index into the training set, and has nothing to do with exponentiation. We will also use X to denote the space of input values, and Y to denote the space of output values. In this example, X = Y = ℝ.

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function h : X → Y so that h(x) is a "good" predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis. Seen pictorially, the process is therefore like this:



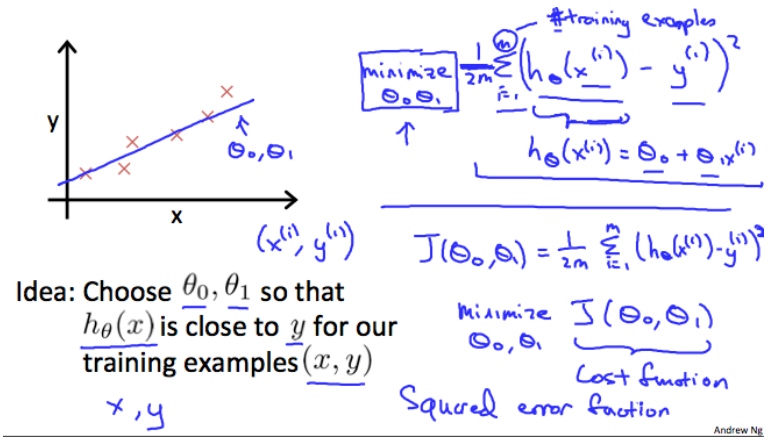
When the target variable that we're trying to predict is continuous, such as in our housing example, we call the learning problem a regression problem. When y can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a classification problem.

Cost Function
We can measure the accuracy of our hypothesis function by using a cost function. This takes an average difference (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's and the actual output y's :

$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

To break it apart, it is 12 $\bar{x}$, where $\bar{x}$ is the mean of the squares of hθ(x) − y , or the difference between the predicted value and the actual value.

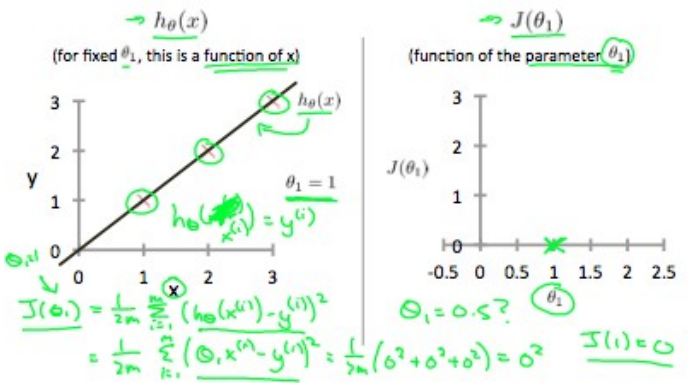
This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved (12) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the 12 term. The following image summarizes what the cost function does:



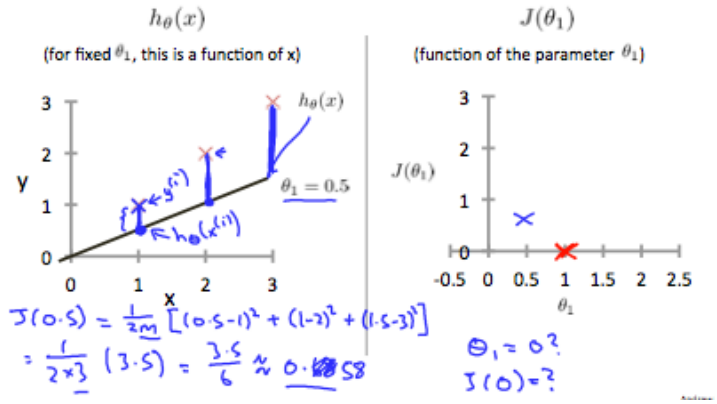
Cost Function - Intuition I

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make a straight line (defined by hθ(x)) which passes through these scattered data points.

Our objective is to get the best possible line.
The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. Ideally, the line should pass through all the points of our training data set. **In such a case, the value of J(θ0,θ1) will be 0**. The following example shows the ideal situation where we have a cost function of 0.



When θ1=1, we get a slope of 1 which goes through every single data point in our model.
Conversely, when θ1=0.5, we see the vertical distance from our fit to the data points increase.



This increases our cost function to **0.58**. Thus as a goal, we should try to minimize the cost function. In this case, θ1=1 is our global minimum.

Gradient Descent

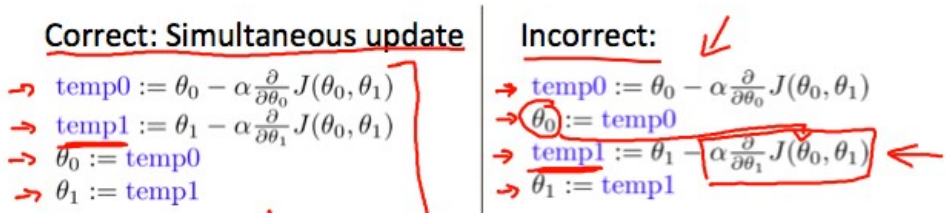
So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields θ0 and θ1 (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.

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 θ1,θ2,...,θn. Updating a specific parameter prior to calculating another one on the j(th) iteration would yield to a wrong implementation.

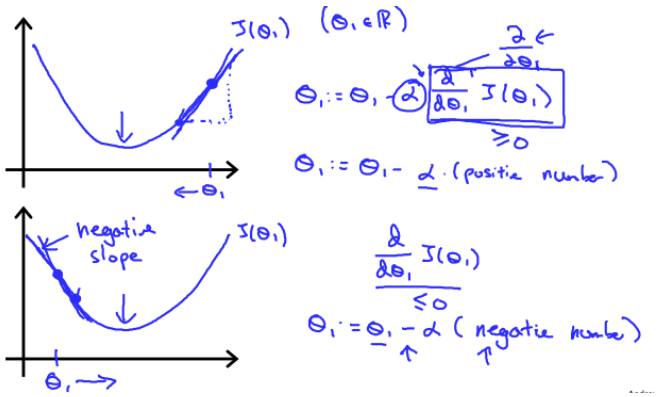


Gradient Descent Intuition

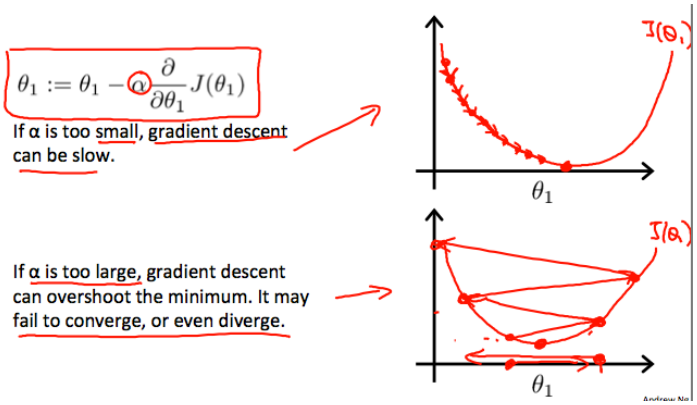
In this video we explored the scenario where we used one parameter θ1 and plotted its cost function to implement a gradient descent. Our formula for a single parameter was :

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

Regardless of the slope's sign for derivative, θ1 eventually converges to its minimum value. The following graph shows that when the slope is negative, the value of θ1 increases and when it is positive, the value of θ1 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 derivative approaches 0 as we approach the bottom of our convex function.

At the **minimum**, the derivative will always be 0 and thus we get: $\theta 1 := \theta 1 - \alpha * 0$