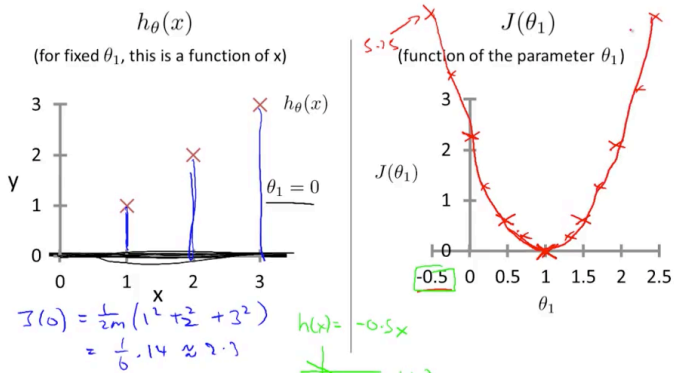
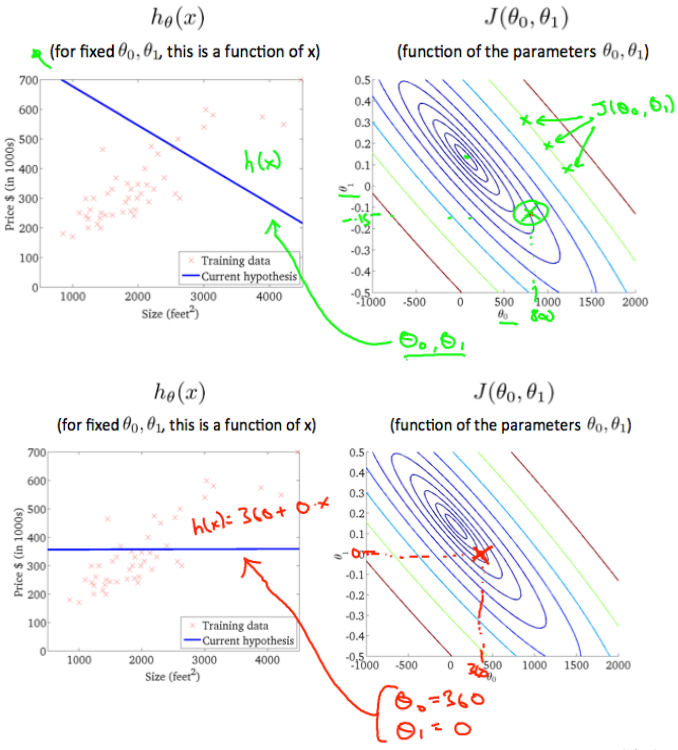
Linear Regression

When we try to fit a linear line into datapoints and minimize the error, it's called linear regression.



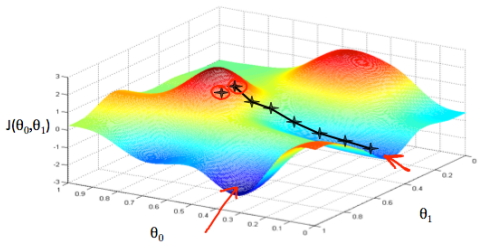
Plot above shows the hypothesis and cost function (J) if we draw a horizontal line x=0 as our hypothesis, we get a point in cost function. By changing the hypothesis, we get different value of J. Best line to fit is the one that is located at minimum of the J(θ).

Let's consider a housing problem (two variables here) and we try to fit a model for pricing vs size of the house. The red dots on the hypothesis plot are training dataset. If we fit the blue line as shown on the top, we get a corresponding point in the cost function counter plot. By changing the slope of the line, the value of J changes. We want hypothesis that gets closer to the center of J contour plot.



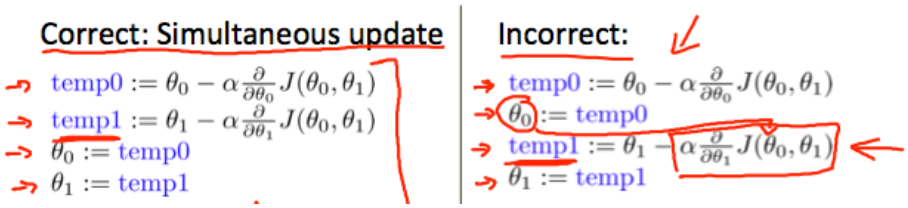
Gradient Descent

How can we estimate the parameters in the hypothesis in order to get the best result? Gradient descent is one way of doing it. In this method, it starts with a point at a cost function, then it looks around and take a small step toward position that has lower cost, it keeps repeating till it reaches the *local minimum*. Depends on where is the starting point, we may end up at different local minimum in a cost function plot. Here is the summary of the algorithm:

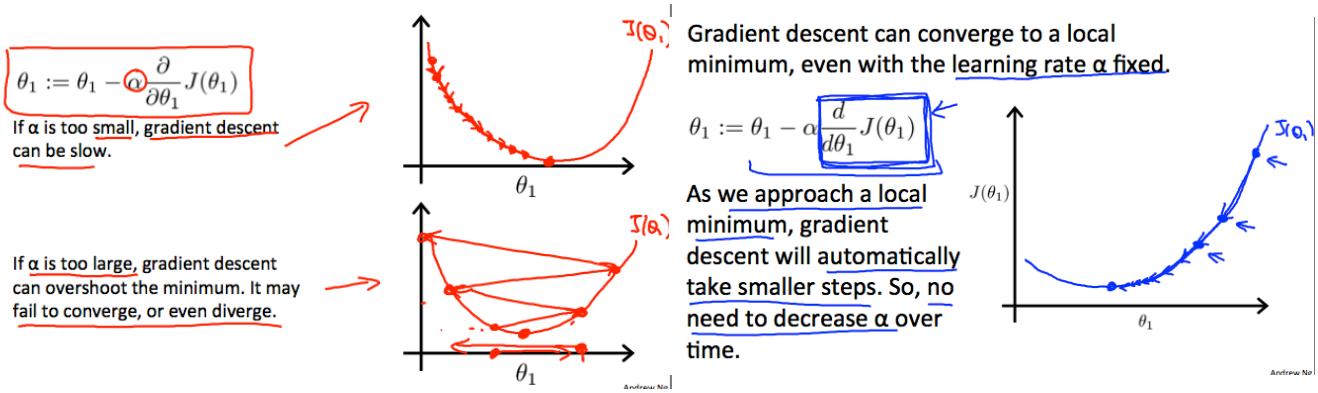


$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$$

where j=0, 1, 2…. . We repeat the algorithm above till it converges to local minimum. Be careful to update each value of θ simultanously, otherwise you'll get a wrong result.



Learning rate, α usually kept constant during analysis or gets smaller as we proceed. In later, we talk about to how choose value of learning rate. At the minimum, the partial derivative is zero.



Batch Gradient Descent for Linear Regression

Let's put the hypothesis into the gradient descent equation for linear regression in order to find the best parameter to fit.

Repeat till converges:

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

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

This method is called batch gradient descent, since we consider all training dataset at once. *In linear regression, the cost function is always a bowl shape, convex shape, therefore the local minimum is global minimum*.

Linear Regression

Multivariate Linear Regression

Linear regression with multiple variables is also known as "multivariate linear regression". The hypothesis is:

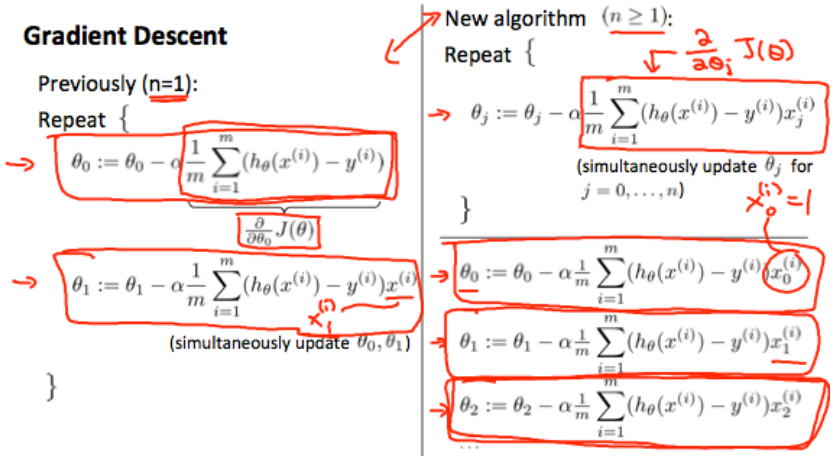
$$h_\theta(x) = \theta_0 + x_1\theta_1 + x_2\theta_2 + .. + x_n\theta_n$$

When n is number of features and m is number of training examples. We can vectorize the equation in order to lower the computational cost. This is vectorization of hypothesis for one training example.

$$h_\theta(x) = [\theta_0 \quad \theta_1 \quad \dots \quad \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} = \theta^T x$$

Gradient Descent for Multiple Variables

Gradient descent is same equation, but we make simple changes in order to use it when we have multivariate variables.



Feature Scaling and Mean Normalization

Feature scaling is important to speed up the gradient descent. This is because θ will descend quicker on smaller range and slower on larger range of values. Therefore, we need to scale the data between -1 and 1 or -0.5 and 0.5 (between any two small values). We may use feature scaling or mean normalization to achieve this goal.

Feature scaling involves dividing the input values by the range:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

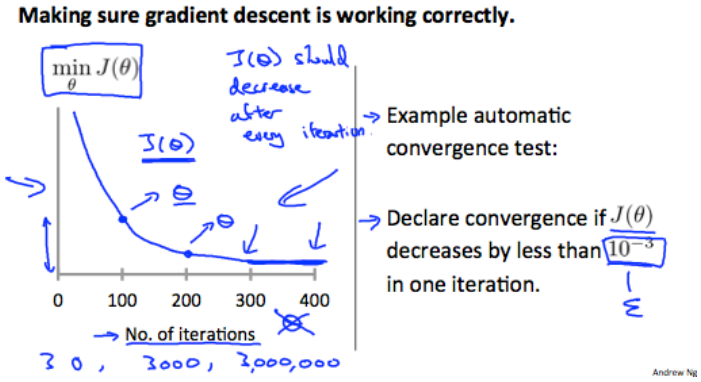
Linear Regression

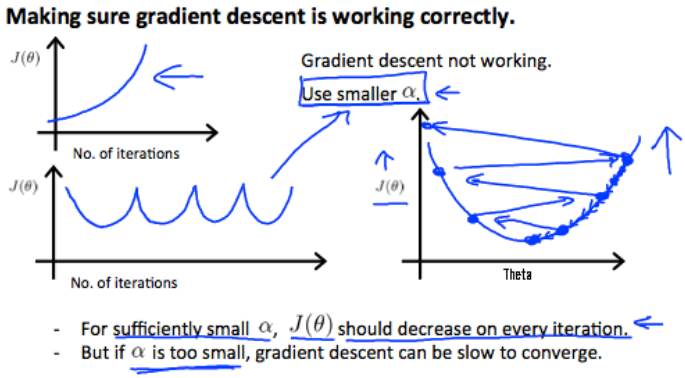
Where s_i is the "max-min" or standard deviation.

Learning Rate

In order to make sure the gradient descent works properly, plot the number of iterations on the x axis, and the cost function J on y axis. This should be decreases as number of iterations increases.

We declare the convergence test if J decreases less than E in one iterations (for example 10^{-3}).





If α is too small: it converges so slowly

If α is too large: it may never converge, since it might go back and forth in a plot and never reaches the local minimum.

Polynomial Regression

We may have more complex model that we cannot fit linear model, and need to fit more complex line such as polynomial. Moreover, *sometimes we combine some features and create newt features* like $x_3 = x_1.x_2$. Hypothesis can be changed to fit quadratic or any other functions, such as $h_\theta(x) = \theta_0 + x_1\theta_1 + \sqrt{x_2}\theta_2$

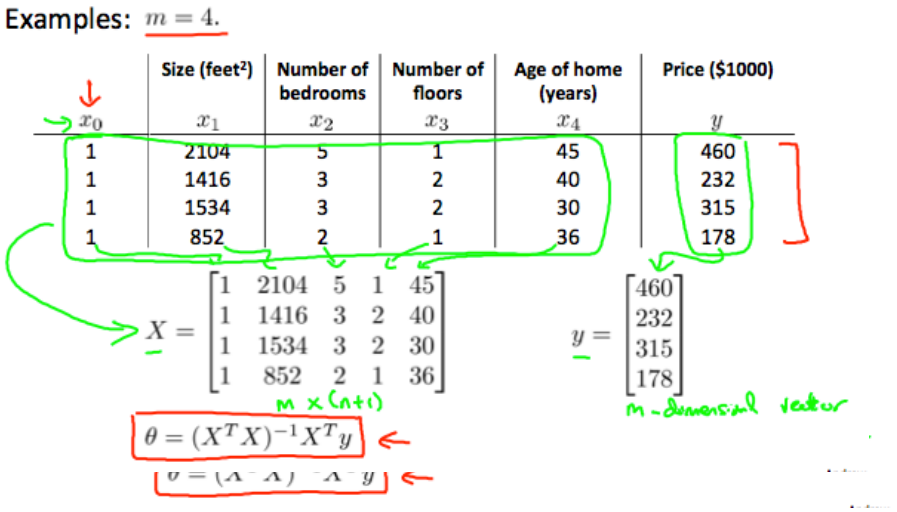
Normal Equation (Closed-Form Equation)

Gradient descent is one way of minimizing J. The other way of doing is to use normal or closed form equation. In this form, we do the minimization directly, without applying any iterations. We minimize J by explicitly taking the derivative of J in respect to the θ and setting it to 0. The vector format of equation is:

$$\theta = (X^T X)^{-1} X^T y$$

Linear Regression

We add value of x_0 to be 1. Suppose we are investigating the housing price based on size, number of floors and etc. features. Here is how matrix X and y are created:



There is no need to do feature scaling in normal equation. In summary, here is the different between normal equation and gradient descent. Normal equation is more computationally costly in compare to the gradient descent.

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn2)	O (n3), need to calculate inverse of X and times X
Works well when n is large	Slow if n is very large

Non-Invertible

Sometimes X^TX might not be invertible due to:

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization"

Using pinv in octave or Matlab, it gives you a value even if the X^TX is not invertible.