Chris McCormick About Membership Blog Archive

Gradient Descent Derivation

04 Mar 2014

Andrew Ng's course on Machine Learning at Coursera provides an excellent explanation of gradient descent for linear regression. To really get a strong grasp on it, I decided to work through some of the derivations and some simple examples here.

This material assumes some familiarity with linear regression, and is primarily intended to provide additional insight into the gradient descent technique, not linear regression in general.

I am making use of the same notation as the Coursera course, so it will be most helpful for students of that course.

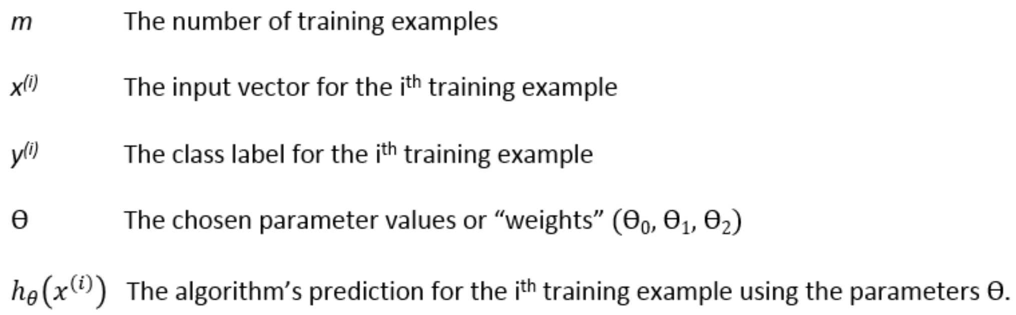
For linear regression, we have a linear hypothesis function, $h(x) = \theta_0 + \theta_1 x$. We want to find the values of θ_0 and θ_1 which provide the best fit of our hypothesis to a training set. The training set examples are labeled x, y, where x is the input value and y is the output. The ith training example is labeled as $x^{(i)}, y^{(i)}$. Do not confuse this as an exponent! It just means that this is the ith training example.

MSE Cost Function

The cost function J for a particular choice of parameters θ is the mean squared error (MSE):

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

Where the variables used are:



m	The number of training examples
$x^{(i)}$	The input vector for the ith training example
$y^{(i)}$	The class label for the ith training example
Θ	The chosen parameter values or “weights” (Θ_0, Θ_1, Θ_2)
$h_\theta(x^{(i)})$	The algorithm’s prediction for the ith training example using the parameters Θ.

The MSE measures the average amount that the model’s predictions vary from the correct values, so you can think of it as a measure of the model’s performance on the training set. The cost is higher when the model is performing poorly on the training set. The objective of the learning algorithm, then, is to find the parameters θ which give the minimum possible cost J.

This minimization objective is expressed using the following notation, which simply states that we want to find the θ which minimizes the cost $J(\theta)$.

$$\min_\theta J(\theta)$$

Gradient Descent Minimization - Single Variable Example

We’re going to be using gradient descent to find θ that minimizes the cost. But let’s forget the MSE cost function for a moment and look at gradient descent as a minimization technique in general.

Let’s take the much simpler function $J(\theta) = \theta^2$, and let’s say we want to find the value of θ which minimizes $J(\theta)$.

Gradient descent starts with a random value of θ, typically $\theta = 0$, but since $\theta = 0$ is already the minimum of our function θ^2, let’s start with $\theta = 3$.

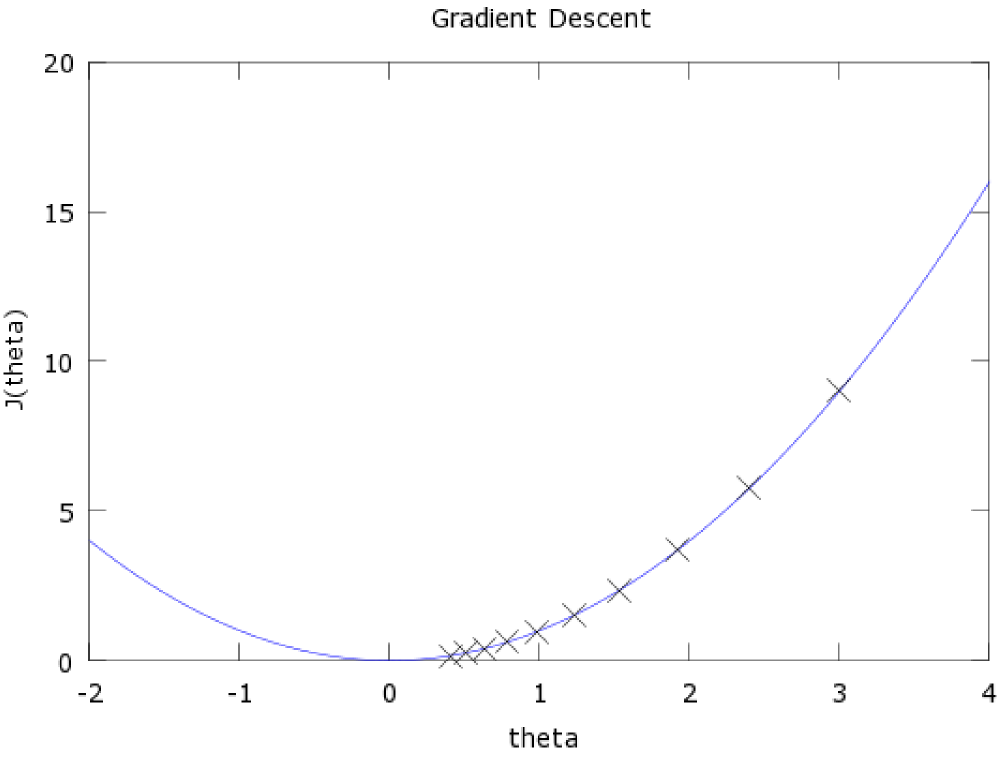
Gradient descent is an iterative algorithm which we will run many times. On each iteration, we apply the following “update rule” (the := symbol means replace theta with the value computed on the right):

$$\theta := \theta - \alpha \frac{d}{d\theta} J(\theta)$$

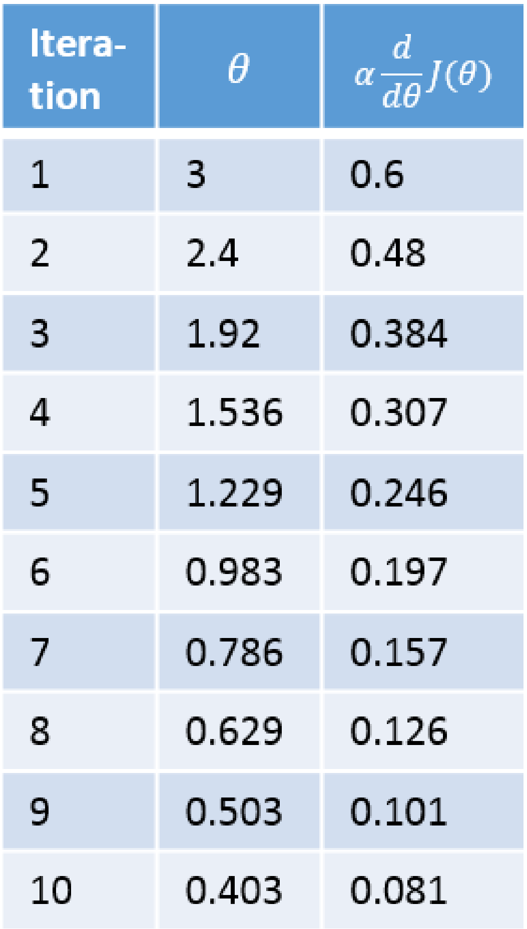
Alpha is a parameter called the learning rate which we’ll come back to, but for now we’re going to set it to 0.1. The derivative of $J(\theta)$ is simply 2θ.

$$\alpha = 0.1, \qquad \frac{d}{d\theta} J(\theta) = 2\theta$$

Below is a plot of our function, $J(\theta)$, and the value of θ over ten iterations of gradient descent.



Below is a table showing the value of theta prior to each iteration, and the update amounts.



Iteration	θ	$\alpha \frac{d}{d\theta} J(\theta)$
1	3	0.6
2	2.4	0.48
3	1.92	0.384
4	1.536	0.307
5	1.229	0.246
6	0.983	0.197
7	0.786	0.157
8	0.629	0.126
9	0.503	0.101
10	0.403	0.081

Cost Function Derivative

Why does gradient descent use the derivative of the cost function? Finding the slope of the cost function at our current θ value tells us two things.

The first is the direction to move theta in. When you look at the plot of a function, a positive slope means the function goes upward as you move right, so we want to move left in order to find the minimum. Similarly, a negative slope means the function goes downard towards the right, so we want to move right to find the minimum.

The second is how big of a step to take. If the slope is large we want to take a large step because we're far from the minimum. If the slope is small we want to take a smaller step. Note in the example above how gradient descent takes increasingly smaller steps towards the minimum with each iteration.

The Learning Rate - Alpha

The learning rate gives the engineer some additional control over how large of steps we make.

Selecting the right learning rate is critical. If the learning rate is too large, you can overstep the minimum and even diverge. For example, think through what would happen in the above example if we used a learning rate of 2. Each iteration would take us farther away from the minimum!

The only concern with using too small of a learning rate is that you will need to run more iterations of gradient descent, increasing your training time.

Convergence / Stopping Gradient Descent

Note in the above example that gradient descent will never actually converge on the minimum, $\theta = 0$. Methods for deciding when to stop gradient descent are beyond my level of expertise, but I can tell you that when gradient descent is used in the assignments in the Coursera course, gradient descent is run for a large, fixed number of iterations (for example, 100 iterations), with no test for convergence.

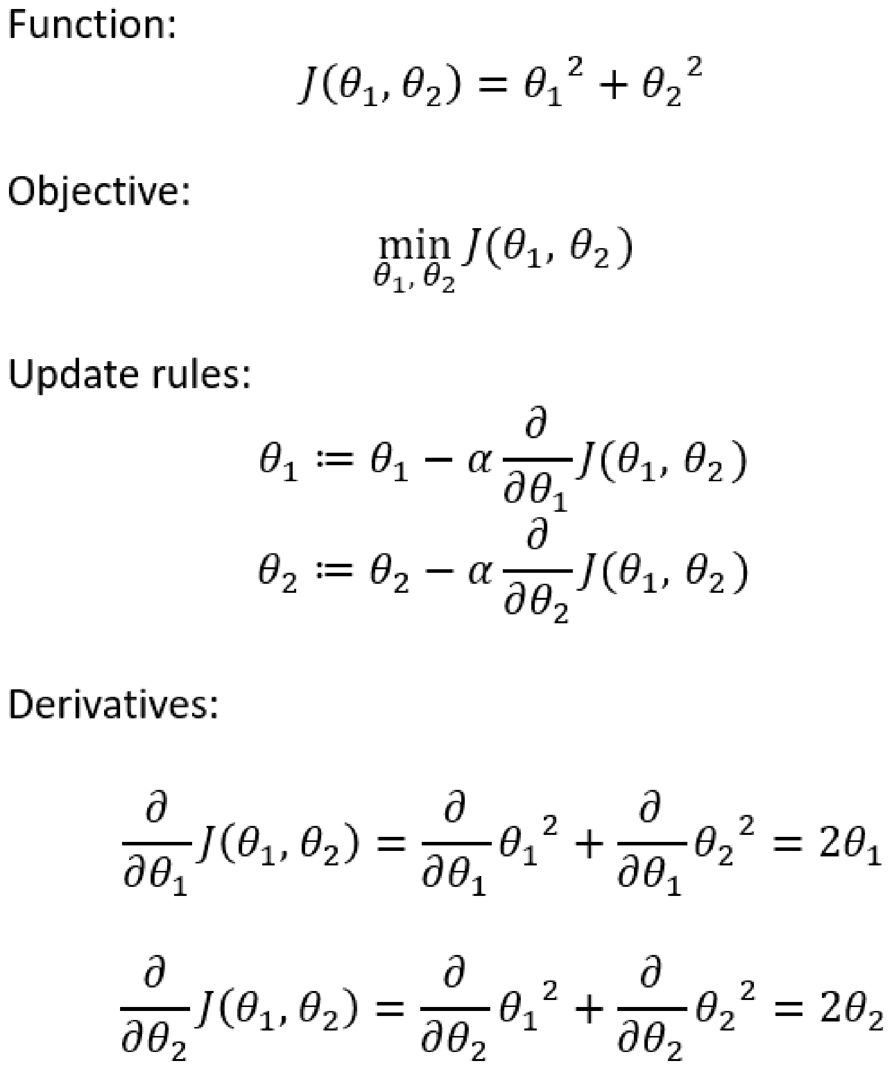
Gradient Descent - Multiple Variables Example

The MSE cost function includes multiple variables, so let’s look at one more simple minimization example before going back to the cost function.

Let’s take the function:

$$J(\theta) = {\theta_1}^2 + {\theta_2}^2$$

When there are multiple variables in the minimization objective, gradient descent defines a separate update rule for each variable. The update rule for θ_1 uses the partial derivative of J with respect to θ_1. A partial derivative just means that we hold all of the other variables constant–to take the partial derivative with respect to θ_1, we just treat θ_2 as a constant. The update rules are in the table below, as well as the math for calculating the partial derivatives. Make sure you work through those; I wrote out the derivation to make it easy to follow.



Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 \coloneqq \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1, \theta_2)$$
$$\theta_2 \coloneqq \theta_2 - \alpha \frac{\partial}{\partial \theta_2} J(\theta_1, \theta_2)$$

Derivatives:

$$\frac{\partial}{\partial \theta_1} J(\theta_1, \theta_2) = \frac{\partial}{\partial \theta_1} {\theta_1}^2 + \frac{\partial}{\partial \theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{\partial}{\partial \theta_2} J(\theta_1, \theta_2) = \frac{\partial}{\partial \theta_2} {\theta_1}^2 + \frac{\partial}{\partial \theta_2} {\theta_2}^2 = 2\theta_2$$

Note that when implementing the update rule in software, θ_1 and θ_2 should not be updated until *after* you have computed the new values for both of them. Specifically, you don't want to use the new value of θ_1 to calculate the new value of θ_2.

Gradient Descent of MSE

Now that we know how to perform gradient descent on an equation with multiple variables, we can return to looking at gradient descent on our MSE cost function.

The MSE cost function is labeled as equation [1.0] below. Taking the derivative of this equation is a little more tricky. The key thing to remember is that x and y are *not* variables for the sake of the derivative. Rather, they represent a large set of constants (your training set). So when taking the derivative of the cost function, we'll treat x and y like we would any other constant.

Once again, our hypothesis function for linear regression is the following:

$$h(x) = \theta_0 + \theta_1 x$$

I've written out the derivation below, and I explain each step in detail further down.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad [1.0]$$

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_0}\left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right) \qquad [1.1]$$

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad [1.2]$$

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}2\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\frac{\partial}{\partial\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \qquad [1.3]$$

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{2}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \qquad [1.4]$$

To move from equation [1.1] to [1.2], we need to apply two basic derivative rules:

Scalar multiple rule: $\frac{d}{dx}(\alpha u) = \alpha\frac{du}{dx}$

Sum rule: $\frac{d}{dx}\sum u = \sum\frac{du}{dx}$

Moving from [1.2] to [1.3], we apply both the power rule and the chain rule:

Power rule: $$\frac{d}{dx}u^n = nu^{n-1}\frac{du}{dx}$$

Chain rule: $$\frac{d}{dx}f(g(x)) = f'\big(g(x)\big)g'(x)$$

Finally, to go from [1.3] to [1.4], we must evaluate the partial derivative as follows. Recall again that when taking this partial derivative all letters except θ_0 are treated as constants (θ_1, x, and y).

$$\frac{\partial}{\partial\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) = \frac{\partial}{\partial\theta_0}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right) = 1$$

Equation [1.4] gives us the partial derivative of the MSE cost function with respect to one of the variables, θ_0. Now we must also take the partial derivative of the MSE function with respect to θ_1. The only difference is in the final step, where we take the partial derivative of the error:

$$\frac{\partial}{\partial\theta_1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) = \frac{\partial}{\partial\theta_1}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right) = x$$

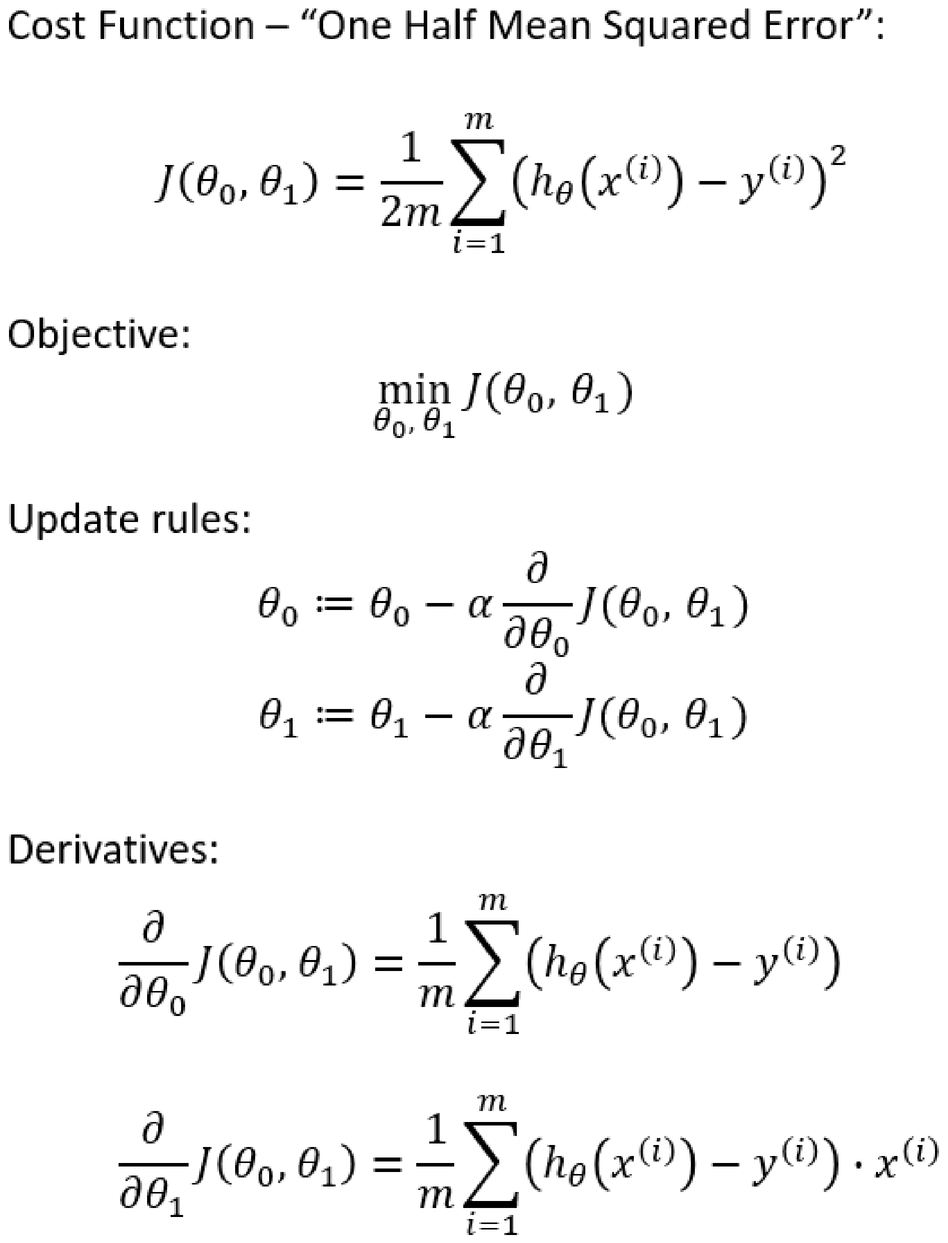
One Half Mean Squared Error

In Andrew Ng's Machine Learning course, there is one small modification to this derivation. We multiply our MSE cost function by 1/2 so that when we take the derivative, the 2s cancel out. Multiplying the cost function by a scalar does not affect the location of its minimum, so we can get away with this.

Alternatively, you could think of this as folding the 2 into the learning rate. It makes sense to leave the 1/m term, though, because we want the same learning rate (alpha) to work for different training set sizes (m).

Final Update Rules

Altogether, we have the following definition for gradient descent over our cost function.



Cost Function – “One Half Mean Squared Error”:

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

Objective:

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

Update rules:

$$\theta_0 \coloneqq \theta_0 - \alpha\frac{\partial}{\partial\theta_0}J(\theta_0, \theta_1)$$
$$\theta_1 \coloneqq \theta_1 - \alpha\frac{\partial}{\partial\theta_1}J(\theta_0, \theta_1)$$

Derivatives:

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

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

Training Set Statistics

Note that each update of the theta variables is averaged over the training set. Every training example suggests its own modification to the theta values, and then we take the average of these suggestions to make our actual update.

This means that the statistics of your training set are being taken into account during the learning process. An outlier training example (or even a mislabeled / corrupted example) is going to have less influence over the final weights because it is one voice versus many.