## What is gradient descent?

## Gradient is generally going upwards and as we have to go down to the valley we take opposite direction of that i.ie negative direction so known as gradient descent.

**Gradient descent method is a way to find a local minimum of a function**. The way it works is we start with an initial guess of the solution and we take the gradient of the function at that point. We **step the solution in the negative direction of the gradient** and we repeat the process. The algorithm will eventually converge where the gradient is zero (which correspond to a local minimum). Its brother, the gradient ascent, finds the local maximum nearer the current solution by stepping it towards the positive direction of the gradient. They are both first-order algorithms because they take only the first derivative of the function.

Many powerful machine learning algorithms use gradient descent optimization to identify patterns and learn from data. Gradient descent  powers machine learning algorithms such as linear regression, logistic regression, neural networks, and support vector machines.

Linear Regression by using Gradient Descent Algorithm: Your first step towards Machine Learning.

Why Gradient Descent? Because it’s one of the best optimization methods that we can use to solve the various machine learning problem. eg. Logistic Regression, Neural Network. but here we are going to discuss about Linear Regression

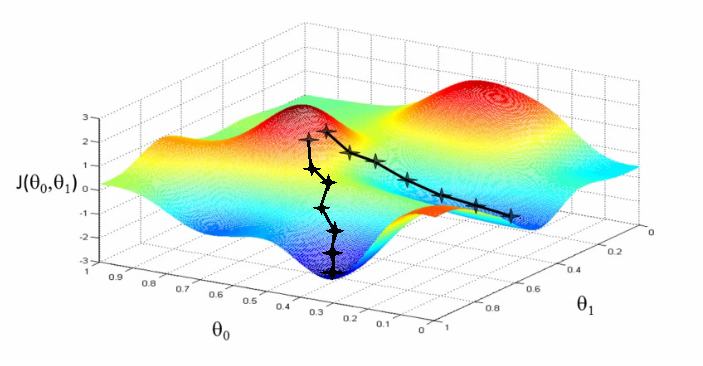
### **What is Gradient Descent ?**

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

To understand in an simpler way,let’s us take the example Suppose you are at the top of a mountain, and you have to reach a lake which is at the lowest point of the mountain. A twist is that you are blindfolded and you have zero visibility to see where you are headed. So, what approach will you take to reach the lake?



The best way is to check the ground near you and observe where the land tends to descend. This will give an idea in what direction you should take your first step. If you follow the descending path, it is very likely you would reach the lake.

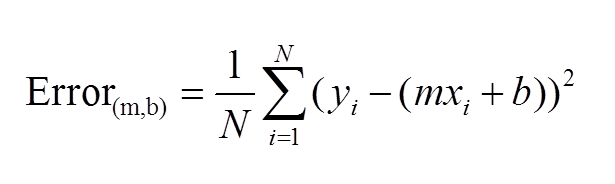


The goal of linear regression is to fit a line to a set of points.

A standard approach to solving this type of problem is to define an error function (also called a cost function) by incorporating gradient descent function we can achieve that measures how good fit a given line is.This function will take in an (m,b) pair and return an error value based on how well the line fits our data. To compute this error for a given line, we’ll iterate through each (x,y) point in our data set and sum the square distances between each point’s y value and the candidate line’s y value (computed at mx + b). It’s conventional to square this distance to ensure that it is positive and to make our error function differentiable.

### **#! Remember this Sum of Square Error Equation.**

### **Here we take Loss function either it can be square or cube etc. generally squared is preferred.**



Equation of Sum of Square Error

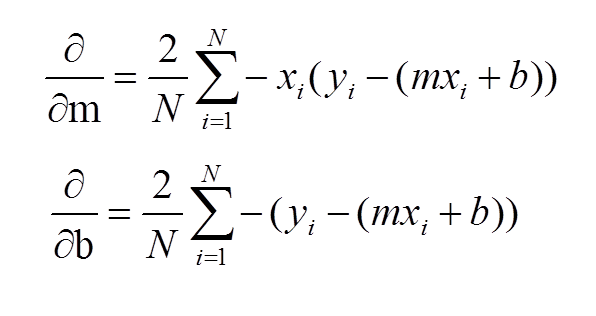
To run gradient descent on this error function, we first need to compute its gradient. The gradient will act like a compass points uphill and gradient descent always point us downhill. To compute it, we will need to differentiate our error function. Since our function is defined by two parameters (**m** and **b**), we will need to compute a partial derivative for each. These derivatives work out to be:

## What is Differentiation?[[edit](https://en.wikibooks.org/w/index.php?title=Calculus/Differentiation/Differentiation_Defined&action=edit&section=1)]

Differentiation is an operation that allows us to find a function that outputs the **rate of change** of one variable with respect to another variable.

The **slope** of a line, also called the **gradient** of the line, is a measure of its inclination. A line that is horizontal has slope 0

### #! Remember this Sum of Square Error Equation Partial Derivaties.

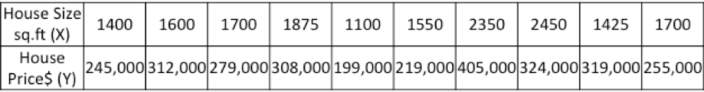


Partial derivatives of Sum of Squared error.

The **learningRate** variable controls how large of a step we take downhill during each iteration. If we take too large of a step, we may step over the minimum. However, if we take small steps, it will require many iterations to arrive at the minimum.

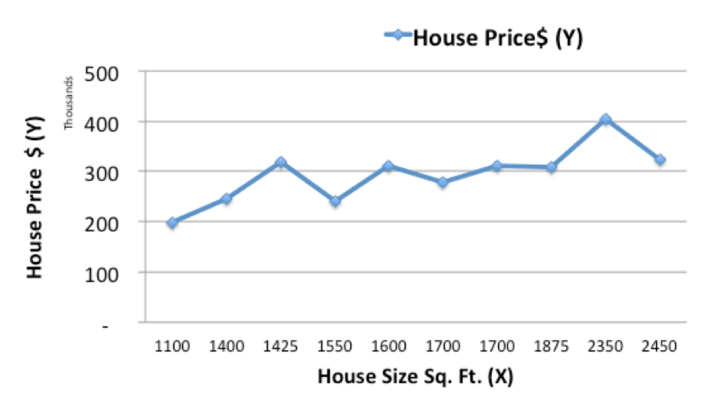
Lets take the example of predicting the price of a new price from housing data:

Now, given **historical housing data,**the task is to create a model that predicts the price of a new house given the house size.

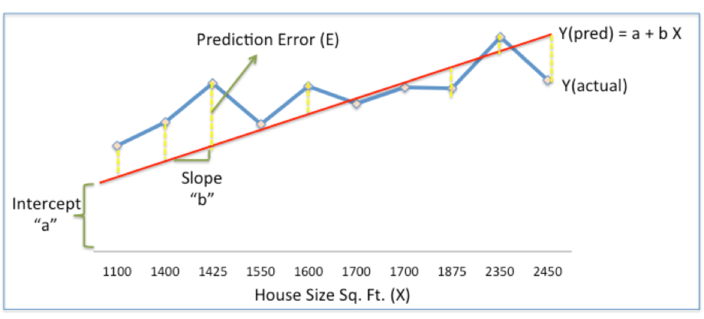


The task – for a new house, given its size (X), what will its price (Y) be?

Lets start off by plotting the historical housing data:



Now, we will use a simple linear model, where we fit a line on the historical data, to predict the price of a new house (Ypred) given its size (X)



In the above chart, the red line gives the predicted house price (Ypred) given house size (X).

Ypred = a+bX

The blue line gives the actual house prices from historical data (Yactual)

The difference between Yactual and Ypred (given by the yellow dashed lines) is the prediction error (E)

So, we need to find a line with optimal values of a,b (called weights) that best fits the historical data by reducing the prediction error and improving prediction accuracy.

So, our objective is to find optimal **a, b** that minimizes the error between actual and predicted values of house price:

**Sum of Squared Errors (SSE) = ½** **a (Actual House Price – Predicted House Price)2**

= **½** **a(Y – Ypred)2**

(Please note that there are other measures of Error. SSE is just one of them.)

This is where Gradient Descent comes into the picture. Gradient descent is an optimization algorithm that finds the optimal weights (a,b) that reduces  prediction error.

Lets now go step by step to understand the **Gradient Descent algorithm:**

**Step 1: Initialize the weights(a & b) with random values and calculate Error (SSE)**

**Step 2: Calculate the gradient i.e. change in SSE when the weights (a & b) are changed by a very small value from their original randomly initialized value. This helps us move the values of a & b in the direction in which SSE is minimized.**

**Step 3: Adjust the weights with the gradients to reach the optimal values where SSE is minimized**

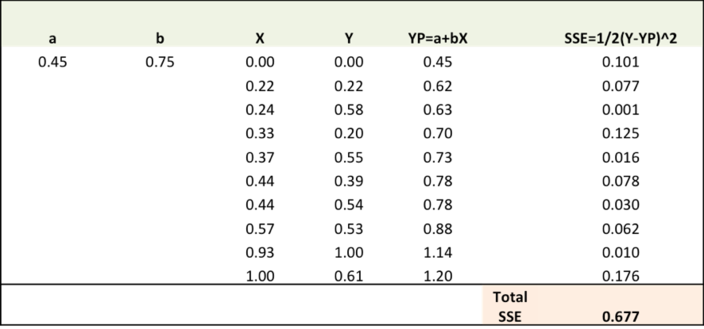
**Step 4: Use the new weights for prediction and to calculate the new SSE**

**Step 5: Repeat steps 2 and 3 till further adjustments to weights doesn’t significantly reduce the Error**

We will now go through each of the steps in detail (I worked out the steps in excel, which I have pasted below). But before that, we have to standardize the data as it makes the optimization process faster.



**Step 1:** To fit a line Ypred = a + b X, start off with random values of a and b and calculate prediction error (SSE)

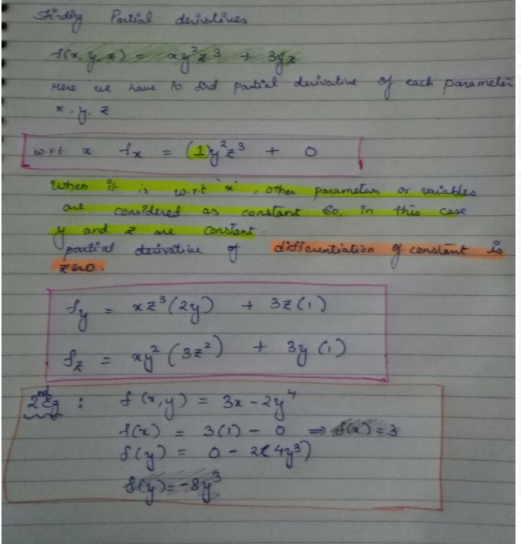


**Step 2:** Calculate the error gradient w.r.t the weights

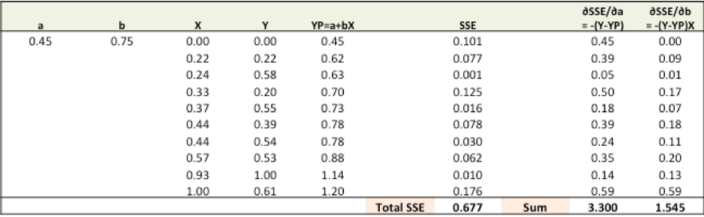
∂SSE/∂a = – (Y-YP)

∂SSE/∂b = – (Y-YP)X

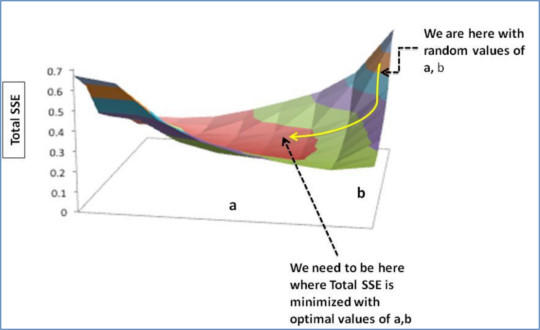
You need to know a bit of calculus, but that’s about it!!



∂SSE/∂a and ∂SSE/∂b are the **gradients** and they give the direction of the movement of a,b w.r.t to SSE.



**Step 3:**Adjust the weights with the gradients to reach the optimal values where SSE is minimized



We need to update the random values of a,b so that we move in the direction of optimal a, b.

Update rules:

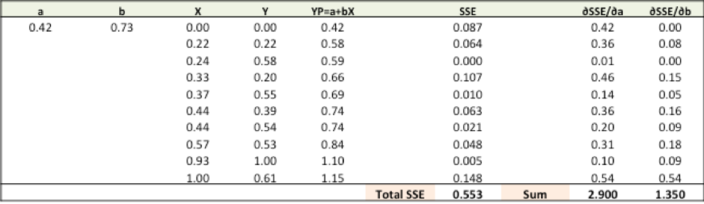
* a – ∂SSE/∂a
* b – ∂SSE/∂b

So, update rules:

1. New a = a – r \* **∂SSE/∂a =** 0.45-0.01\*3.300 = 0.42
2. New b = b – r \* **∂SSE/∂b=** 0.75-0.01\*1.545 = 0.73

here, r is the learning rate = 0.01, which is the pace of adjustment to the weights.

**Step 4:**Use new a and b for prediction and to calculate new Total SSE



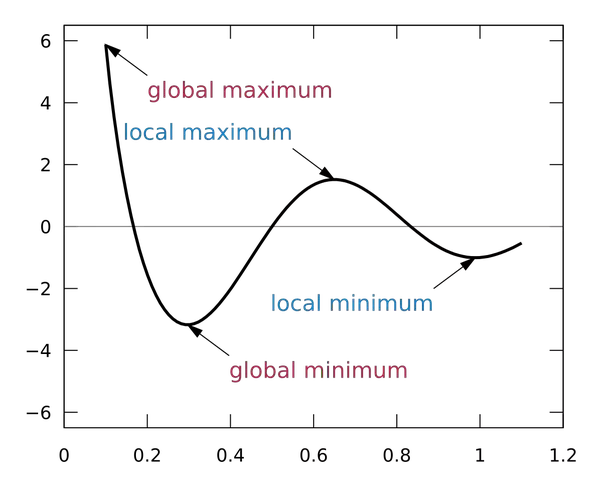
You can see with the new prediction, the total SSE has gone down (0.677 to 0.553). That means prediction accuracy has improved.

**Step 5:** Repeat step 3 and 4 till the time further adjustments to a, b doesn’t significantly reduces the error. At that time, we have arrived at the optimal a,b with the highest prediction accuracy.

This is the Gradient Descent Algorithm. This optimization algorithm and its variants form the core of many machine learning algorithms like Neural Networks and even Deep Learning.

# What is the local minimum and global minimum in machine learning? Why are these important in machine learning?

Consider this picture of the graph of an elementary (1D) function:

[[](https://qph.ec.quoracdn.net/main-qimg-0dbade2626f60dad3b8ae2007706c597)](https://qph.ec.quoracdn.net/main-qimg-0dbade2626f60dad3b8ae2007706c597)

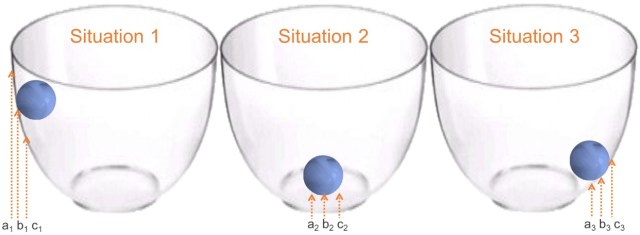
Now, as a machine learning practitioner, suppose you are trying to find the point that minimizes the value of this function without knowing the function a priori. One common method is gradient descent, which in this case is analogous to rolling the ball down the surface from your starting point. In this case, if you are on top of the middle hill and rolling your ball to the right, your ball will get stuck in a valley that is sub-optimal, as there is a better valley on the left. However, if there was only one valley (which would by default be the global minimum), you would be *guaranteed* to get your ball to the bottom of an optimal valley, no matter which way you threw it (as long as you are throwing “gently”, to preserve the analogy, as real-life gradient descent may diverge with a large “step size”).

In many advanced ML models the error surfaces are non-convex, meaning that in all likelihood gradient descent converges to sub-optimal valleys, or local minima. Research is being done to understand the geometry/topology of the error surfaces, and to obtain better minima from these techniques (stochastic gradient descent is popular with many for the reason that it has a higher chance of “bumping” one out a poor local minima).

Extra Notes:

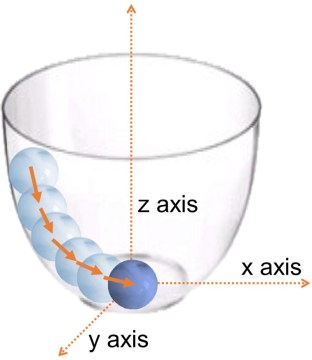
**Gravity & Gradient Descent Optimization**

Consider these 3 situations where a ball is placed in a glass bowl at 3 different positions. We all know through our experience with gravity that the force of gravity will pull the ball towards the bottom of the bowl.

[](https://i0.wp.com/ucanalytics.com/blogs/wp-content/uploads/2016/03/Picture1.jpg)

In situation 1, the ball will move from position b1 to c1 and not a1. This ball will continue to travel till it reaches the bottom of the bowl. In situation 2, since the static ball is already at the bottom it will stay at b2and it won’t move at all. The ball always tries to move from a higher potential energy state to a lower. Gravity is responsible for these different potential energy states of the ball.

Now, you must ask: how is this similar to gradient descent optimization? Actually, this is exactly how gradient descent optimization works. The only major difference is that gradient descent optimization tries to minimize a loss function instead of potential energy.

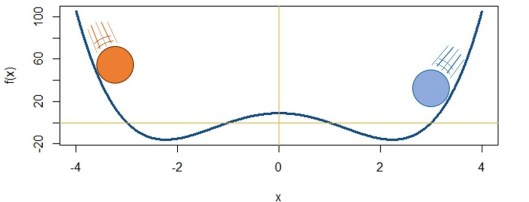
[](https://i0.wp.com/ucanalytics.com/blogs/wp-content/uploads/2016/03/bowl-and-ball.jpg)

the ball experienced different pulls at different stages while it journeyed towards the bottom of the bowl. These pulls can be evaluated through the  gradient or the slope of the orange arrows shown in the adjacent picture. The steeper the orange arrows larger is the force of gravity on the ball.

For gradient descent optimization, the z-axis is the loss function and x &  y-axes are the coefficients of the model. The loss function is equivalent to the potential energy of the ball in the bowl. The idea is to minimize loss function by adjusting x & y-axes (i.e coefficients of the model).

#### Iterative Calculation

It is great to solve the equation with the first method but unfortunately for quite a lot of complicated functions it is not possible to solve equations the way we do in method-1. Hence we need to identify some numeric methods to solve such equations. Numeric methods to solve an equation require iterative calculation. It is similar to rolling a ball downwards, and measuring its position with each calculation till it reaches the bottom. The idea is to measure the speed of the ball based on the gradient or the slope of the curve. This is precisely how gradient descent optimization works.

[](https://i0.wp.com/ucanalytics.com/blogs/wp-content/uploads/2016/03/Picture2-e1459331186740.jpg)

The first thing you must have noticed is that this function has 2 minimum values. Hence, initial positions of rolling the ball will take you to different values ( √5 or -√5). This is a slight problem but luckily for many machine learning problems we just have one minimum value. This special property of a function is referred to as convexity. In other words, convexity is the property of a bowl to have just one bottom.

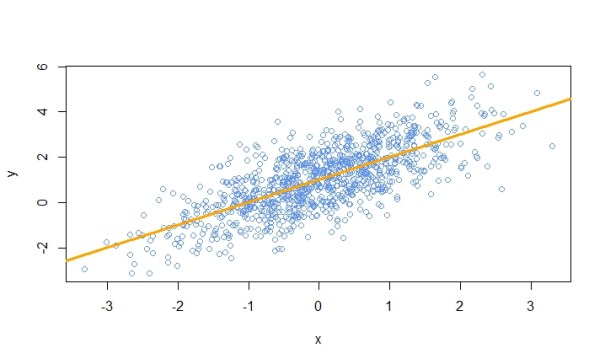
Moreover to roll the ball we need to know the gradient or slope of the function (bowl) at different points. But how does one identify gradients of the curve at different points?

**Linear Regression – Loss Function & Gradient Descent**

There are several business and practical world problems that use linear regression to estimate values. For instance, if we have data for 100 professionals about their salaries and years of experiences we can fit a curve on this data and identify patterns between these two variables. Using the data we are trying to estimate an equation of this nature:

\hat{Salary}= 5000\times (Years\ of\ Experience)+20000 

Now, if some professional tells you she has 10 years of experience. You could easily estimate her salary as 70,000 (5000×10+20000). The most important factors here are the values of the coefficients for the line (i.e. m = 5000 & c=20000). We can estimate these coefficients with the data through either the Ordinary Least Square (OLS) Method or gradient descent optimization. This example is quite simple but imagine if you had 8000 more variables in addition to *years of experience*that’s when you need machine learning and gradient descent.

Consider this data set with different values of x and y. We will try to find the equation of the orange line through both OLS and gradient descent methods.[](https://i2.wp.com/ucanalytics.com/blogs/wp-content/uploads/2016/03/Rplot03.jpeg)

#### Method-1 : Solving Linear Regression by OLS

We are interested in finding an equation for the line through linear regression.

\hat{y}= m\times x+c 

Here, m and c are constants or coefficients of a regression equation.

As discussed earlier, we can easily solve this linear equation with OLS. Using R’s lm function we get this equation of the line for our data (m = 1.013 & c=1.003). Lm function uses matrix inversion. [**R code for linear regression**](http://ucanalytics.com/blogs/wp-content/uploads/2016/04/Linear-Regression-Code-lm.txt)**.**

\hat{y} = 1.013 \times x+1.003 

#### Method-2 : Solving Linear Regression by Gradient Descent

Now, let’s try to find the values of coefficients (m & c) using gradient descent. For this first, we need to identify the error or the loss function for the linear regression model. Remember the loss function will determine the shape of the bowl on which we are rolling the ball. The generalized equation of loss function for any problem is:

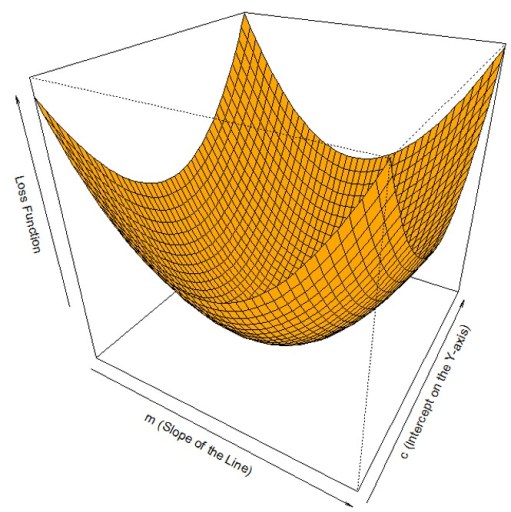
Loss\ Function\ (LF)=\frac{1}{N} \sum (y-\hat{y})^{2} 

This is the original value of y minus expected value. For the linear regression problem, the expected value is the equation of the line, therefore:

Loss\ Function\ (LF)=\frac{1}{N} \sum (y-(mx+c))^{2} 

If we plot loss function (z-axis) with respect to the coefficients of linear regression (m and c) the 3D plot looks like this.

.

[](https://i1.wp.com/ucanalytics.com/blogs/wp-content/uploads/2016/03/Picture1-1.jpg)

Alright, this looks like a bowl. Now, we just need to roll down the ball to identify the minimum value of loss function by adjusting the coefficients (m and c). Gradient descent reaches the bottom of the curve by iterative calculation of both m & c. For this, we need to figure out the way to roll the ball down the slope at each step. Keep the following analogy between gravity and gradient descent which minimizes loss function by adjusting for m & c.

Firstly, gradient descent requires the direction of the downward slope at each point on the loss function with respect to both m & c. Secondly, it needs to control the speed at which the ball will roll. This entire thing is captured in this equation for m: m_{n+1}=m_{n}-\alpha \frac{\partial}{\partial m_{n}} LF (m_{n})

Here, the new position of m i.e. (mn+1) is decided by the previous position of m i.e. (mn) by rolling the ball down the slope (  \frac{\partial}{\partial m_{n}} LF (m_{n})  ) . The partial differentiation term determines the gradient or slope of the curve. Additionally, α is learning rate which determines the speed at which the ball will roll.

c_{n+1}=c_{n}-\alpha \frac{\partial}{\partial c_{n}} LF (c_{n})

Similarly, we can iteratively estimate the value of the other coefficient of the linear regression model i.e c. The best fit value for c and m is when the slope becomes flat or the ball gets to the bottom of the bowl.

Now we just need to find the gradient term and supplement them in the above equations. The gradient term for m is:

\frac{\partial}{\partial m}LF = \frac{2}{N} \sum (y-(mx+c)) \times x 

Similarly, the gradient term for c is:

\frac{\partial}{\partial c}LF = \frac{2}{N} \sum (y-(mx+c))

http://andrew.gibiansky.com/blog/machine-learning/machine-learning-the-basics/