Gradient Descent Algorithm

Juan David Rodríguez Cuervo

September 26, 2017

The **Gradient Descent** algorithm is a technique that helps finding the minimum of a function [1]. It will be used to calculate the line of best fit for a linear dataset points. In general terms, this is called the **Linear Regression problem**.

The objective with this problem is to get an equation in the form

$$y = mx + b \tag{1}$$

that best fits with the data given. In other words, an equation to which the error value is minimized.

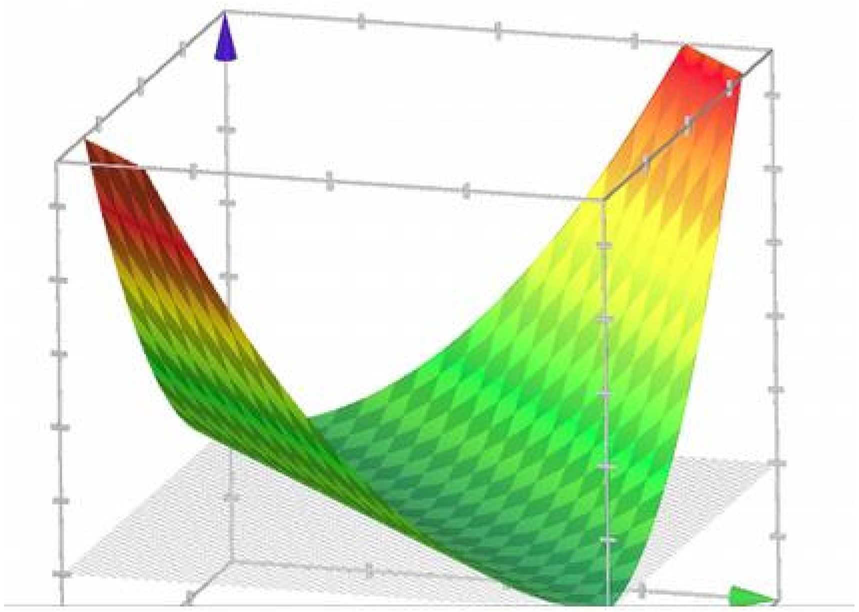
As one might guess, this is the equation of the straight line in the plane, where m is the slope and b is the y-intercept (interception with the vertical axis). As such, x is the independent variable and y being the dependent (i.e. $y = f(x)$). Thus, our target values will be for m and b. These are the coefficients we want to change in order to minimize the error value.

In first place, what we want to do is minimize the error value, as we have state it previously. Therefore, we have to have an expression for the error value. As our data has a linear structure, our error can have also a linear statistical form, as to say:

$$error = E(m, b) = \frac{1}{N} \sum_{i=1}^{N} [y_i - (mx_i + b)]^2 \tag{2}$$

Here, N is the number of datapoints with which we are going to train our model. y_i, x_i are the i-th point of the training dataset. Notice that for the calculation of the error, values must have been defined for m and b. In general, y_i is the expected value for the corresponding x_i, while the experimental or obtained is given by $mx_i + b$.

Drawing the general form of this error function, you can see why this method is called *Gradient Descent.* At some point in the error curve, the objective is to reach the optimal values, that is, to reach the lowest point of the curve(*see the image below*).



As our model goes on and on, we might expect that its error value gets smaller and smaller. That raises two questions, as to say: *in which direction should I move the values* and *in which magnitude?*. A very common way to solve these questions is by subtracting the gradient to the value. In that way, if the error in some direction is small, it will take smaller steps than if the error is big.

As you might guess, it is necessary to find the gradients for each parameter we are modifying. In multi-variable calculus this is done with partial derivatives. From (2) we get then:

$$\frac{\partial E(m,b)}{\partial m} = -\frac{2}{N}\sum_{i=1}^{N} x_i[y_i - (mx_i + b)] \tag{3}$$

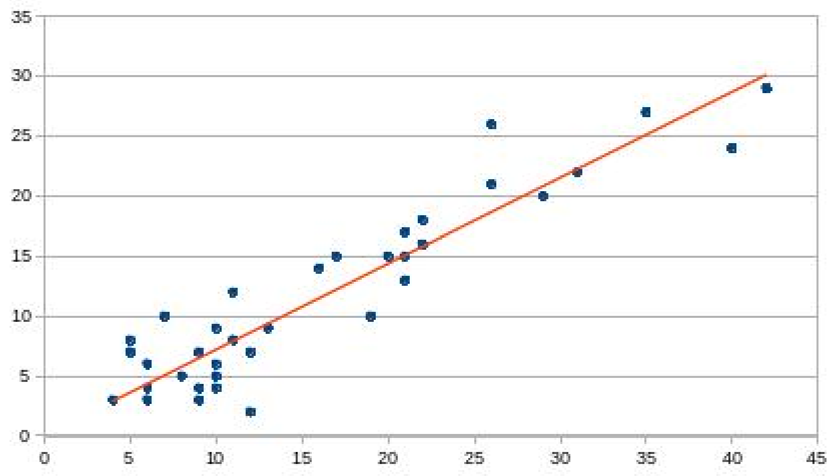
$$\frac{\partial E(m,b)}{\partial b} = -\frac{2}{N}\sum_{i=1}^{N} [y_i - (mx_i + b)] \tag{4}$$

With (3) and (4) we solve the problem of the direction and magnitude of the movement. To avoid the divergence of the *learning process*, in the code we define a *learning_rate* which will control the movement of the values. now, for the setting of the new values, let p be the step number in which the system is at certain point. That way, we have, for each step:

$$m_{p+1} = m_p - \frac{\partial E}{\partial m_p} \times learning\_rate \tag{5}$$

$$b_{p+1} = b_p - \frac{\partial E}{\partial b_p} \times learning\_rate \tag{6}$$

At the end of the process, the results were very close to the expected values. As you can see in the following graph, the trend line *learned* from the system, describes very well the dataset points given.


35
30
25
20
15
10
5
0
0
5
10
15
20
25
30
35
40
45