Understanding Gradient Descent

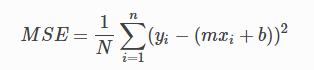
One of the foundations of Machine Learning and the mechanism that neural networks use to backpropagate solutions is called gradient descent. This quick example will demonstrate how to use them in a common machine learning problem. With just a couple of cycles on gradient descent we can correlate the best fit for our equation y=mx+b with the sample data showing house prices (x) and square feet(y).

It is a good idea to make your own copy of the [sheet linked](https://docs.google.com/spreadsheets/d/10CFrwKF7ZqvTKfFS0zjkfhArO6F3PKaOAiS8FH9IKQg/edit#gid=0) here so you can easily tweak and reproduce these results yourself. You will notice the values of both the Sales Price and Square Feet are first processed through the min/max function giving us a value of 0 for the smallest data values and 1 for the largest values in each axis.

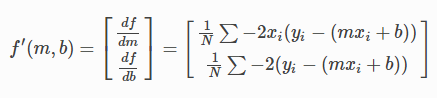
|  |  |
| --- | --- |
| **Sales Price (X)** | **Square Feet (Y)** |
| $220,350.00 | 1,210.00 |
| $261,000.00 | 1,450.00 |
| $329,800.00 | 1,940.00 |
| $538,200.00 | 2,100.00 |
| $621,000.00 | 2,760.00 |
| $888,300.00 | 4,230.00 |

|  |  |
| --- | --- |
| **Sales Price X (Min/Max)** | **Square Feet Y(Min/Max)** |
| 0 | 0 |
| 0.2938196555 | 0.3427895981 |
| 0.371270967 | 0.4586288416 |
| 0.6058763931 | 0.4964539007 |
| 0.6990881459 | 0.6524822695 |
| 1 | 1 |

Initially we will pick arbitrary values at random for m and b, feel free to change them on your own spreadsheet your end result will be the same. Remember our purpose here is to create a line that comes closest to a good fit for the data. A good way to measure this is the common Mean Squared Error formula:



You’ll notice this is done for you on the spreadsheet, where n is the number of training examples in our data. Our goal is to minimize the sum of the individual loss values for each data point, and we do this by taking the derivative of this MSE loss function with respect to our two variables m and b:



The final step is to take the sum of these gradient (derivative) values and multiply by our learning rate so we know how much to adjust our initial guess for the m and b values. The formula for updating them with these new summed derivatives is simply: M -= step\*gradient. In this case the first update for our M value is 4-(0.1\*4.27) = 3.57.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Guess of m in y=mx+b** | **Guess of b in y=mx+b** | **Guessed Value (yguess)** | **Loss** | **Deriv of Loss with respect to m** | **Deriv of Loss with respect to b** | **Updated Value of m** |
| 4.00 | 5.00 | 5.00 | 4.17 | 0 | 0.333 | 3.572588347 |
| 4.00 | 5.00 | 6.18 | 5.67 | 0.456 | 0.725 |  |
| 4.00 | 5.00 | 6.49 | 6.05 | 0.562 | 0.828 | **Updated Value of b** |
| 4.00 | 5.00 | 7.42 | 8.00 | 0.910 | 1.141 | 4.403992645 |
| 4.00 | 5.00 | 7.80 | 8.51 | 1.013 | 1.265 |  |
| 4.00 | 5.00 | 9.00 | 10.67 | 1.333 | 1.667 |  |
|  |  | Sum | 7.18 | 4.27 | 5.96 |  |

It is good practice to run the process again and ensuring the loss sum is decreasing, confirming you’re moving in the right direction. The reason it is important to keep a small step value is because the derivative changes as we move farther away from each of our data points, and while a larger learning rate could close in on the minimum more quickly we run the risk of overshooting our desired answer. With additional iterations you will observe the slope changing more and more gradually bringing us to the local minimum, our final goal.