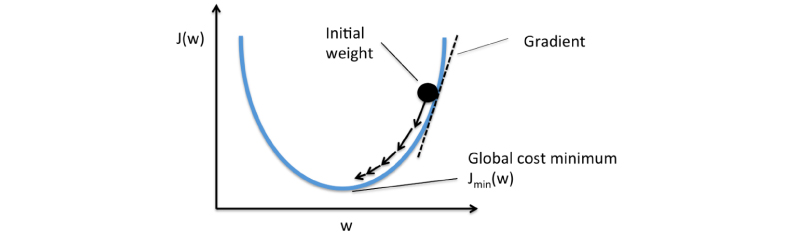
**What is Gradient Descent?**

Gradient descent is an optimization algorithm which is mainly used to find the minimum of a function. In machine learning, gradient descent is used to update parameters in a model. Parameters can vary according to the algorithms, such as *coefficients* in Linear Regression and weights in Neural Networks.

Let us relate gradient descent with a real-life analogy for better understanding. Think of a valley you would like to descend when you are blind-folded. Any sane human will take a step and look for the slope of the valley, whether it goes up or down. Once you are sure of the downward slope you will follow that and repeat the step again and again until you have descended completely (or reached the minima).

This is exactly what happens in gradient descent. The inclined and/or irregular is the cost function when it is plotted and the role of gradient descent is to provide direction and the velocity (learning rate)  of the movement in order to attain the minima of the function i.e where the cost is minimum.



**How does Gradient Descent work?**

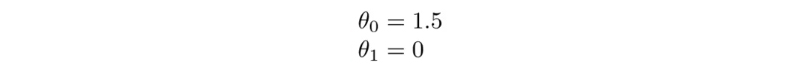
The primary goal of machine learning algorithms is always to build a model, which is basically a hypothesis which can be used to find an estimation for Y based on X. Let us consider an example of a model based on certain housing data which comprises of the sale price of the house, the size of the house etc. Suppose we want to predict the pricing of the house based on its size. It is clearly a regression problem where given some inputs, we would like to predict a continuous output.

The hypothesis is usually presented as

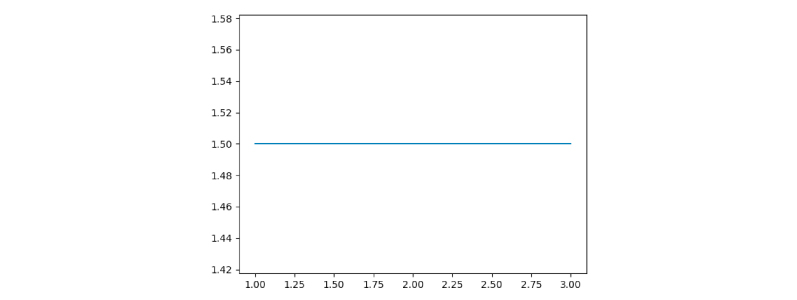
hypothesis formula

where the theta values are the *parameters*.

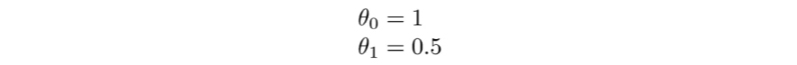
Let us look into some examples and visualize the hypothesis:

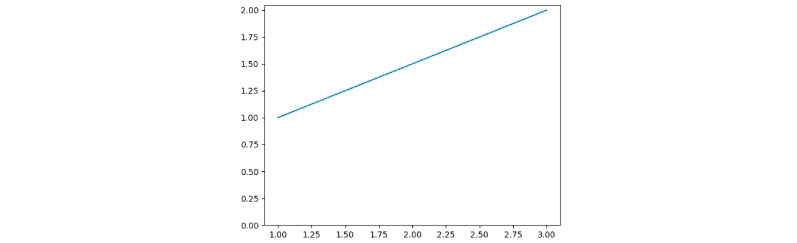


This yields h(x) = 1.5 + 0x. 0x means no slope, and y will always be the constant 1.5. This looks like:



Now let us consider,



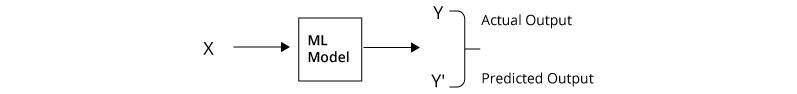


Where, h(x) = 1 + 0.5x

**Cost Function**

The objective in the case of gradient descent is to find a line of best fit for some given *inputs*, or X values, and any number of Y values, or *outputs*. A cost function is defined as *“a function that maps an event or values of one or more variables onto a real number intuitively representing some “cost” associated with the event.”*

With a known set of inputs and their corresponding outputs, a machine learning model attempts to make predictions according to the new set of inputs.

Machine Learning Process

The Error would be the difference between the two predictions.

The Error would be the difference between the two predictions.

This relates to the idea of a **Cost function or Loss function**.

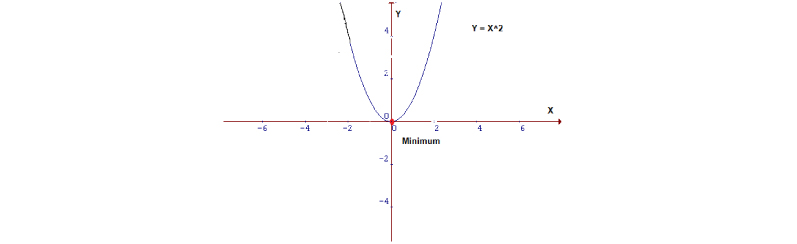
A **Cost Function/Loss Function** tells us “how good” our model is at making predictions for a given set of parameters. The cost function has a curve and a gradient, the slope of this curve helps us to update our parameters and make an accurate model.

**Minimizing the Cost Function**

It is always the primary goal of any Machine Learning Algorithm to minimize the Cost Function. Minimizing cost functions will also result in a lower error between the predicted values and the actual values which also denotes that the algorithm has performed well in learning.

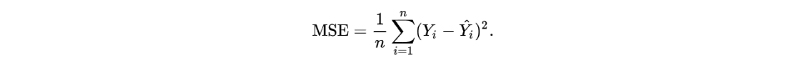
**How do we actually minimize any function?**

Generally, the cost function is in the form of **Y = X²**. In a Cartesian coordinate system, this represents an equation for a parabola which can be graphically represented as :

Parabola

Now in order to minimize the function mentioned above, firstly we need to find the value of X which will produce the lowest value of Y (in this case it is the red dot). With lower dimensions (like 2D in this case) it becomes easier to locate the minima but it is not the same while dealing with higher dimensions. For such cases, we need to use the Gradient Descent algorithm to locate the minima.

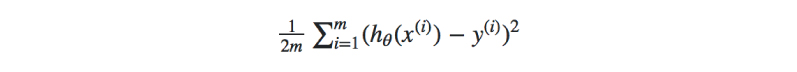
Now a function is required which will minimize the parameters over a dataset. The most common function which is often used is the  [mean squared error](https://en.wikipedia.org/wiki/Mean_squared_error). It measures the difference between the estimated value (the prediction) and the estimator (the dataset).

Mean Squared Error

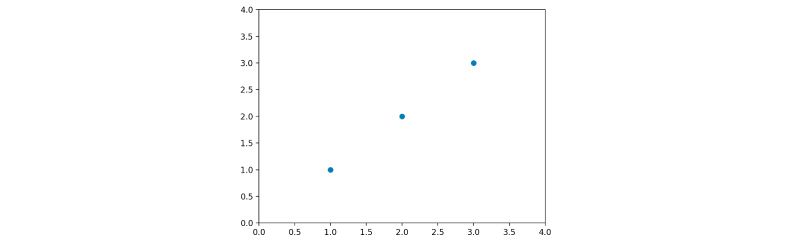
It turns out we can adjust the equation a little to make the calculation down the track a little more simple.

Now a question may arise, ***Why do we take the squared differences and simply not the absolute differences?*** Because the squared differences make it easier to derive a regression line. Indeed, to find that line we need to compute the first derivative of the Cost function, and it is much harder to compute the derivative of absolute values than squared values. Also, the squared differences increase the error distance, thus, making the bad predictions more pronounced than the good ones.

The equation looks like -

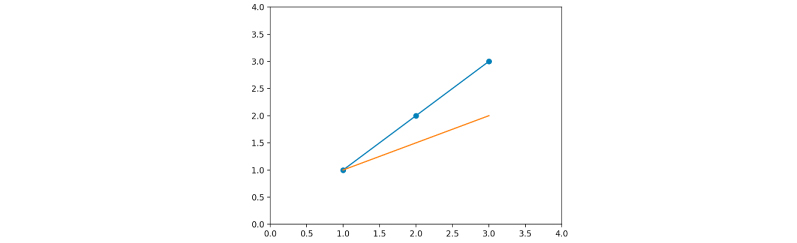
Mean Squared Error

Let us apply this cost function to the following data:



Here we will calculate some of the theta values and then plot the cost function by hand. Since this function passes through (0, 0), we will look only at a single value of theta. Also, let us refer to the cost function as J(ϴ) from now on.

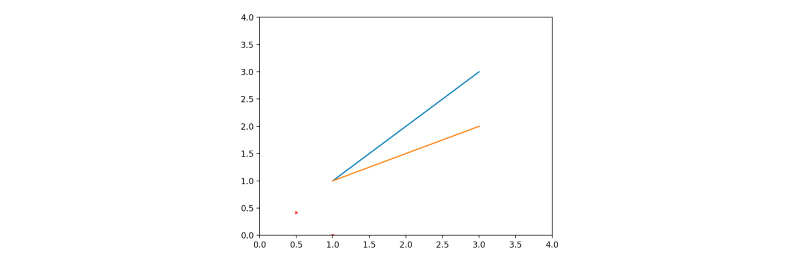
When the value of ϴ is 1, for J(1), we get a 0. You will notice the value of J(1) gives a straight line which fits the data perfectly. Now let us try with ϴ = 0.5

J(0.5)

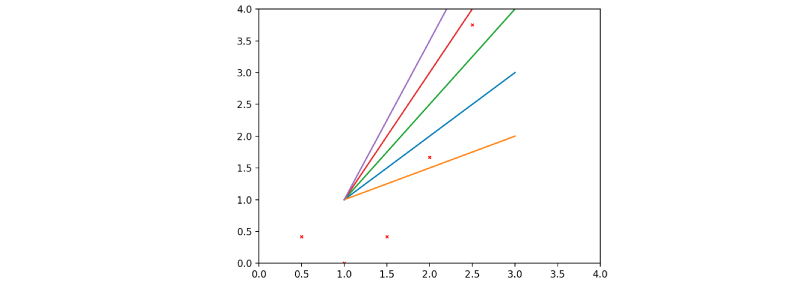
The MSE function gives us a value of 0.58. Let’s plot both our values so far:

J(1) = 0

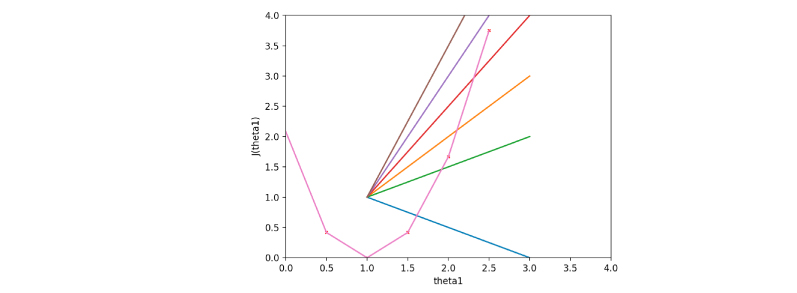
J(0.5) = 0.58

With J(1) and J(0.5)

Let us go ahead and calculate some more values of J(ϴ).



Now if we join the dots carefully, we will get -

Visualizing the cost function J(ϴ)

As we can see, the cost function is at a minimum when theta = 1, which means the initial data is a straight line with a slope or gradient of 1 as shown by the orange line in the above figure.

Using a trial and error method, we minimized J(ϴ). We did all of these by trying out a lot of values and with the help of visualizations. **Gradient Descent** does the same thing in a much better way, by changing the theta values or parameters until it descends to the minimum value.