**Gradient Descent:**

*Gradient descent is an iterative optimization algorithm for finding the local minimum of a function.*

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.

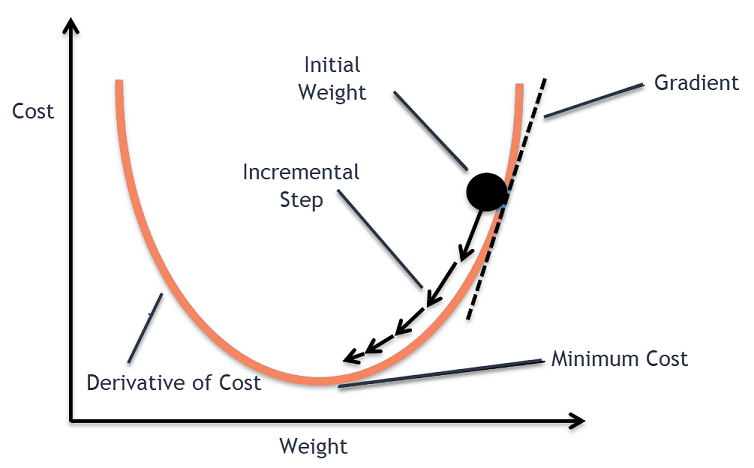
Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

Goal of this algorithm is to find the m (Slope) and b (intercept)

There by reduce the error, until the error is minimal or almost static it needs to be iterated.

In other words

To find the local minimum of a function using gradient descent, we must take steps proportional to the negative of the gradient (move away from the gradient) of the function at the current point.



**Gradient Descent Procedure for Multiple variables.**

The procedure starts off with initial values for the coefficient or coefficients for the function. These could be 0.0 or a small random value.

x = 0.0

The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.

Y = f(x)

The derivative of the cost is calculated. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.

direction = derivative(cost)(dx/dy)

Now that we know from the derivative which direction is downhill, we can now update the coefficient values. A [learning rate parameter](https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/)  must be specified that controls how much the coefficients can change on each update.

Stepsize = direction(dx/dy) \* Learning rate

New\_x = old\_x – ( stepsize )

This process is repeated until the cost of the coefficients (cost) is 0.0 or close enough to zero to be good enough.

Cost Function or the Mean Square Error.

*It is a****function****that measures the performance of a model for any given data.****Cost Function****quantifies the error between predicted values and expected values and presents it in the form of a single real number.*

*Cost Function = ∑(y-(mx+b))^2 / n*

### The Learning Rate

We have the direction we want to move in, now we must decide the size of the step we must take.

***\*It must be chosen carefully to end up with local minima.***

* If the learning rate is too high, we might **OVERSHOOT**the minima and keep bouncing, without reaching the minima
* If the learning rate is too small, the training might turn out to be too long

1. a) Learning rate is optimal, model converges to the minimum
2. b) Learning rate is too small, it takes more time but converges to the minimum
3. c) Learning rate is higher than the optimal value, it overshoots but converges ( 1/C < η <2/C)
4. d) Learning rate is very large, it overshoots and diverges, moves away from the minima, performance decreases on learning

### Local Minima

The cost function may consist of many minimum points. The gradient may settle on any one of the minima, which depends on the initial point (i.e initial parameters(theta)) and the learning rate. Therefore, the optimization may converge to different points with different starting points and learning rate.

Once we tune the learning parameter and get the optimal learning rate, we start iterating until we converge to the local minima.