GRADIENT DESCENT RULE FOR MEAN SQUARED ERROR

Let's take a model ; a hypothesis

$\bar{y} = mx + c$

where $m \to$ weight & $c \to$ bias.

Let y be the actual value

For a training mode (x^0, y^0)

$\bar{y}^0 = mx^0 + c$

$y^0 - \bar{y}^0$ gives us loss

Squaring it to amplify the loss & make it positive

$(y^0 - \bar{y}^0)^2 = L^0$

Doing it for m training models and adding them up

$(y^0 - \bar{y}^0)^2 + (y^1 - \bar{y}^1)^2 + \dots + (y^m - \bar{y}^m)^2 = L^0 + L^1 + \dots + L^m$

$\sum_{i=0}^{m} (y^{(i)} - \bar{y}^{(i)})^2 = \sum_{i=0}^{m} L^{(i)}$

Dividing both sides with to find mean of losses

$\frac{1}{m} \sum_{i=0}^{m} (y^{(i)} - \bar{y}^{(i)})^2 = L(x)$

This gives us the loss as a function of x.

where lesser the L better the fit

$L = 0$ is the ideal case

$L >>> 0$ the worse the fit

We then use this loss function to minimize loss ~~for~~

by tweaking the weight and bias.

We are finding the minima of a bowl shaped function where x, y and z axis are w, b and L respectively papergrid

Date: / /

Partial differentian w.r.t m(ω)

m → ω , c → b

$$\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)} - \omega x^{(i)} - b\right)^2 = L \text{ or } J$$

Differentialing w.r.t ω

$$\frac{-2}{m}\sum_{i=0}^{m} x^{(i)}\left(y^{(i)} - \bar{y}^{(i)}\right) = \frac{\partial L}{\partial \omega} = \frac{\partial J}{\partial \omega}$$

Differentiating w.r.t b

$$\frac{-2}{m}\sum_{i=0}^{m}\left(y^{(i)} - \bar{y}^{(i)}\right) = \frac{\partial J}{\partial \omega}$$

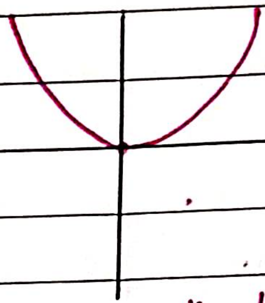
The gradient descent update rule

The goal is to minimize cost function

In gradient descent,

1. We first start with some random value for ω and b [Typically 0]
2. We keep on changing the values of ω and b to reduce J(ω) and J(b)

This is basically taking small steps to reach minimum possible elevation. The one point to note with gradient descent, depending on which point you start on, you can end up at different local minima. Since we are dealing with linear regression and the function is J(ω,b) which represents a bowl like graph which only has one minima which is global minima.



We subtract the the gradient descent value because if we test it and check if we add we end up moving away from global minima.
We also need to choose 'α' a learning rate to determine the size of the step taken towards global minimum. We need to make sure 'α' is not ~~in~~ too large to make sure it doesn't overshoot past the global minima & Not too small to make sure it doesn't take too many iterations. In Practice we take $\alpha = 0.01$ But in Theory, we test α with multiple value & see which drives down the fastest
That's where we get the formula

$w = w - \alpha \nabla J(w)$ $\qquad$ $b = b - \alpha \nabla J(b)$

$\nabla J(w) \rightarrow$ represents gradient ~~des~~ ascent

$-\nabla J(w) \rightarrow$ represents gradient descent

$w = w - \alpha \nabla J(w)$ $\qquad$ $b = b - \alpha \nabla J(b)$

Are the formulae used in my Python program.