*) Generic case optimisation.

we worked for this in the mathematical model to compute error function

↓

$\sum_i (y_i - mx_i + c)^2$ (single variable)

↓

Motivation for gradient descent

differentiating with reference to m and c to find minimum error.

↓

this approach cannot be adopted for the generic case for n variables.

↓

we have a formula for this

$$x = \begin{bmatrix} \\ \\ \end{bmatrix}_{m \times (n+1)}$$

→ x is data having m training points with n features.

↓

another column is added with all 1s to make the eqn completely generic and to find c, by finding m_{n+1}, by letting c be a feature with all values = 1.

according to LR ↑ ↳ generic case

Now ($y = x \cdot m$)

where y is output matrix

x is data matrix

and m is coefficient matrix

↓

what we need to find to find the boundary.

↓

this matrix or determinant can be multiplied to xTest and we can find Ypre accordingly.

$$M = (x^T \cdot x)^{-1} \cdot x^T \cdot y$$

$$A = \begin{bmatrix} 1 & 5 \\ 4 & 8 \\ 7 & 9 \end{bmatrix}_{3\times 2}$$

$$A^T = \begin{bmatrix} 1 & 4 & 7 \\ 5 & 8 & 9 \end{bmatrix}_{2\times 3}$$

$x \rightarrow m \times (n+1)$

$x^T \rightarrow (n+1) * m$

$y \rightarrow M * 1$

for a square matrix (computing time complexity)

$A \cdot A^{-1} = I = A^{-1} \cdot A$

A^{-1} ↓ exists only for sq. matrix.

$A^{-1} = \frac{1}{|A|} \cdot adj(A)$.

* time to find coefficient matrix using the formula.

↓

$X^T \rightarrow (n+1)^* m \longrightarrow O(mn)$ (going to all elements)

$(X^T \cdot X) \rightarrow (n+1^* m) \cdot (m^* n+1)$

$\rightarrow O(mn^2)$

↓

There are $(n+1)^*(n+1)$ elements in the output matrix. for each element calculation we need to use m computation since m elements in each row in one first matrix and m elements in each column.

↓

$(n+1)^* (n+1)^* m$ computations

↓

$O(mn^2)$

→ Now $(X^T \cdot X)^{-1} \rightarrow O(n^3)$

This is a whole lot of computations to perform along with time req., which will be too much.

↓

rest of the operations might not be too heavy.

↓

Now, if n↑ → time is inc. non linearly

↓

owing to

↓

$O(n^3)$ and $O(mn^2)$

↓

it is not uncommon to have 10^5 features, and then the time required will be very much.

↓

10^5 features could be present because the dataset might be so comprehensive or, we might have introduced many dummy variables / features to our data.

↓

and ∴ the computation ↑

↓

fit function becomes very slow

↓

Also, these solutions are analytically found.

↓

exact math → exact solutions

↓

not desirable

↓

alt. method

↓

approximate values

↓

gradient descent (much faster)

↓

approximation function / optimization function

↓

finding the coefficients, so as to minimize the cost function

* Gradient Descent

↓

using gradient descent to find our coefficients such that our cost is minimized

• taking about one feature inputs for now

↓

same discussion can be extended to multidimensions

$$\boxed{y_p = mx + c} \qquad cost = \sum_i [y_i - (mx_i + c)]^2$$

↓

given hypothesis function

↓

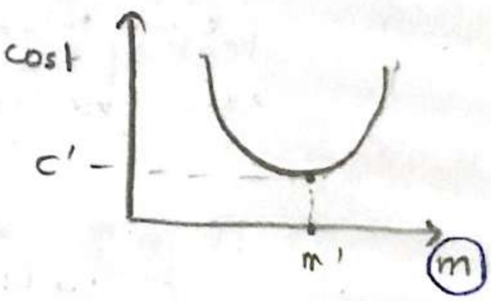
finding m and c such that the line best fits our data

↓

cost function has to be minimized

↓

m and c to minimize this



* For a certain m, the cost function will be minimum and moving to the left or right of this point, will lead to its value increasing.

↓

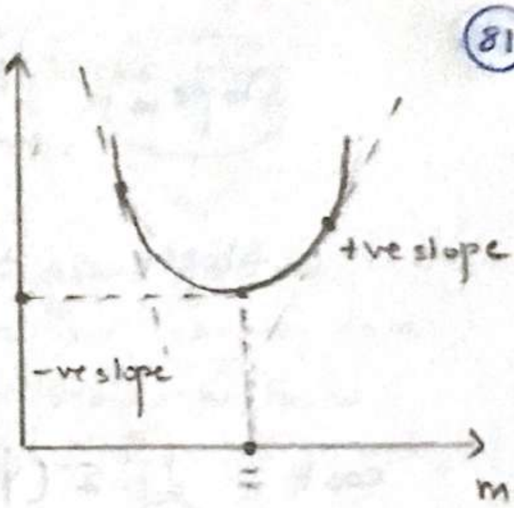
gradient descent says, that we need to start with a value of m and then move to a better solution by aptly changing the value of m.

(same with c)

↓

$$m' = m - \alpha \text{ slope}$$

m: current m; α: constant



* If we on the right, our slope is +ve, and we decrease the slope and move towards better m.

* If we on the left, our slope is -ve, and we inc the slope and move towards better m.

↓

In both cases by subtracting certain amount from m

↓

Reversal of sign will occur for the -ve case

* The inclusion of constant α, is due to the given possibility that the slope might be too high and we reduce m too much. that the sign of slope changes. (basically).

↓

α is used to control how fast we want to reduce m, i.e. we can get more cautious as we move towards a better solution.

↓

we keep subtracting till the change in m is ~~not~~ substantial, the moment it stops being that, we stop subtracting.

* $m' = m - \alpha \, slope_m$

$c' = c - \alpha \, slope_c$

$slope_m$: slope with respect to m

$$\text{slope}_m = \frac{\partial \text{cost}}{\partial m} = \frac{2}{N} \sum_i (y_i - mx_i - c)(-x_i)$$

(for all datapoints)

$$\text{slope}_c = \frac{\partial \text{cost}}{\partial c} = \frac{2}{N} \sum_i (y_i - mx_i - c)(-1)$$

$$\text{cost} = \frac{1}{N} \sum_i (y_i - mx_i - c)^2$$

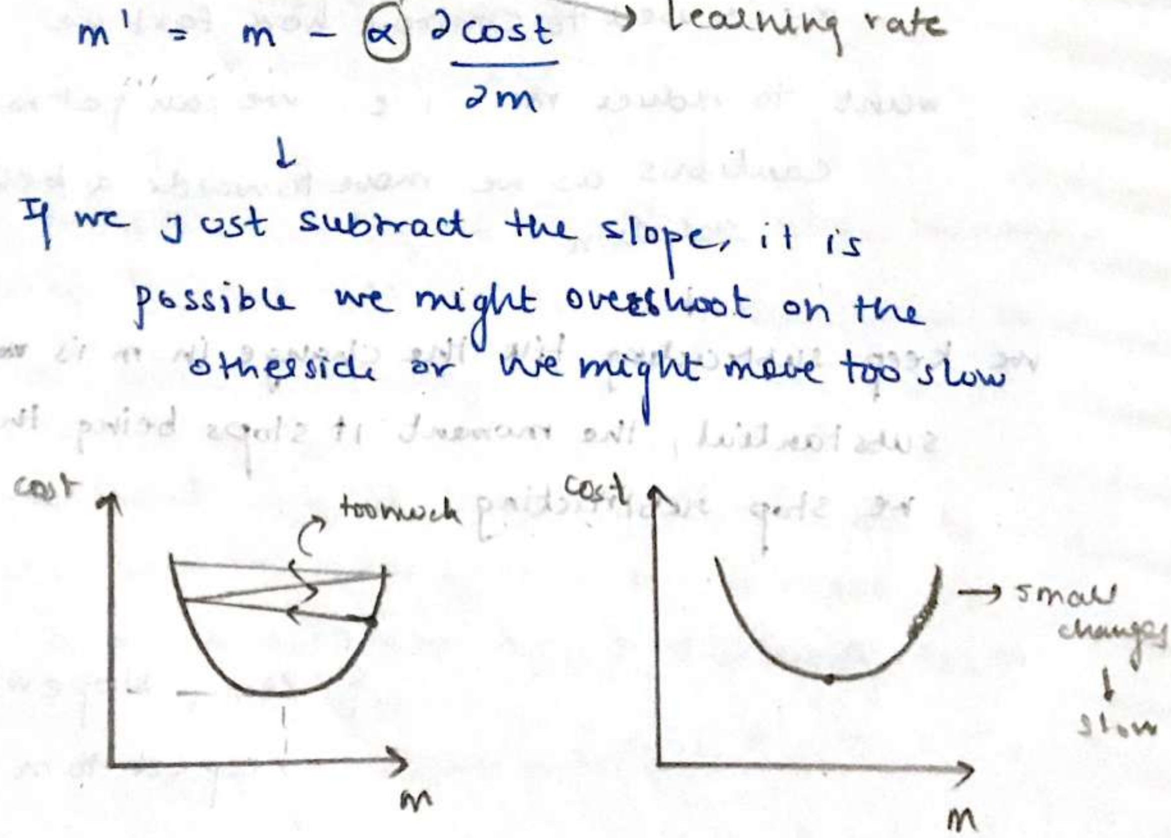
↓
avg. over all data points
↓
cost per data point (in a way)

* we start for some m and c, find the slope there and deduce the new m, and keep doing this till the changes are substantial and noteworthy!!

* $m' = m - \alpha \frac{\partial \text{cost}}{\partial m}$ → learning rate

↓

If we just subtract the slope, it is possible we might overshoot on the otherside or we might move too slow



* For both cases, we need a way to control the rate on how m changes.

↓

α can be passed to a gradient function, as in we want this to be our learning rate and accordingly new m would be found.

* if α is too high → same overshooting problem

↓

α is too low → too small changes → (too small)

↓

(too much time)

↓

experimenting and finding apt α

↓

adaptive α

↓

start with high value of α, and keep reducing it as we move towards the minima

If we overshoot

↓

cost ↑

↓

Indicator that α is too damn high, so we should tune it, so as to reduce the step size.

* Generic gradient descent

↓

n features, in datasets.

↓

$$m' = m - \alpha \frac{\partial \varepsilon}{\partial m}$$

$y = M^{K} x$ $\quad x_j^i \rightarrow$ j th row, ith feature

↓

$$cost = \frac{1}{R} \sum_{i=1}^{R} \left[y_i - (m_1 x_i^1 + m_2 x_i^2 + m_3 x_i^3 + \dots + m_{n+1} x_i^{n+1})\right]^2$$

↓

$$m_j' = m_j - \alpha \frac{\partial cost}{\partial m_j}$$

↓

$$\frac{\partial cost}{\partial m_j} = -\frac{2}{R} \sum \left[y_i - (m_1 x_i^1 + m_2 x_i^2 + \dots + m_{n+1} x_i^{n+1})\right] \times x_i^j$$

↓

* variations of Gradient Descent

↓

i) ~~Batch~~ Batch → going through entire data set and then updating m and c.

ii) stochastic → going through each datapoint to update slope values.

code ↗

↓

lot of oscillations for each iteration

↓

faster movement towards the minimum

iii) miniBatch → updating after going through a smaller set of datapoints