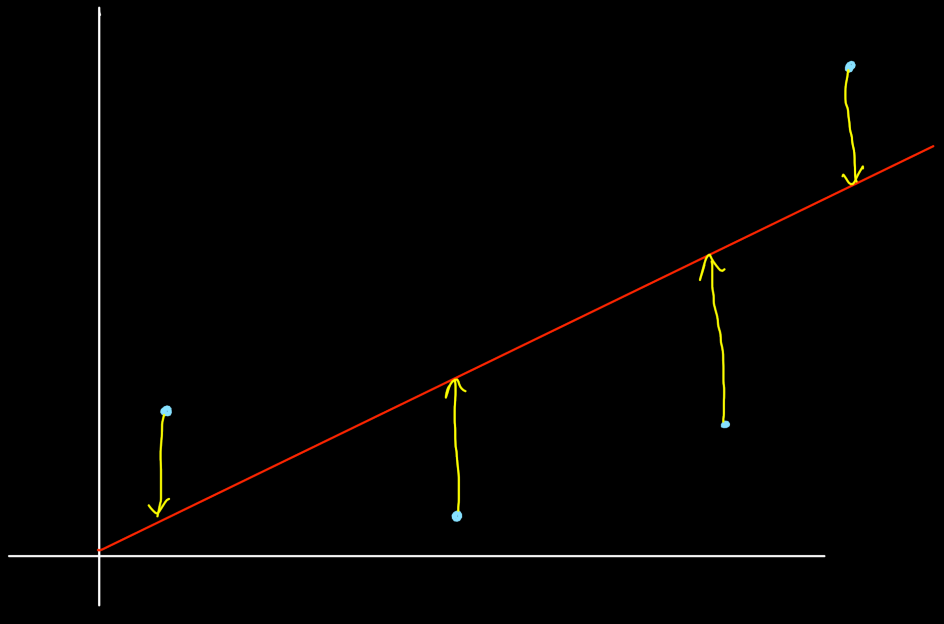
Gradient Descent from Scratch

Wikipedia : Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent. Conversely, stepping in the direction of the gradient will lead to a local maximum of the function; the procedure is then known as gradient ascent.

The Plan :

1) Gradient descent
2) Batch GD
3) SGD → SGD Regressor
4) MB GD

Intuition :



CgPa	dPa
—	—
—	—
—	—
—	—

$$L = \sum_{i=1}^{4} (y_i - \hat{y}_i)$$

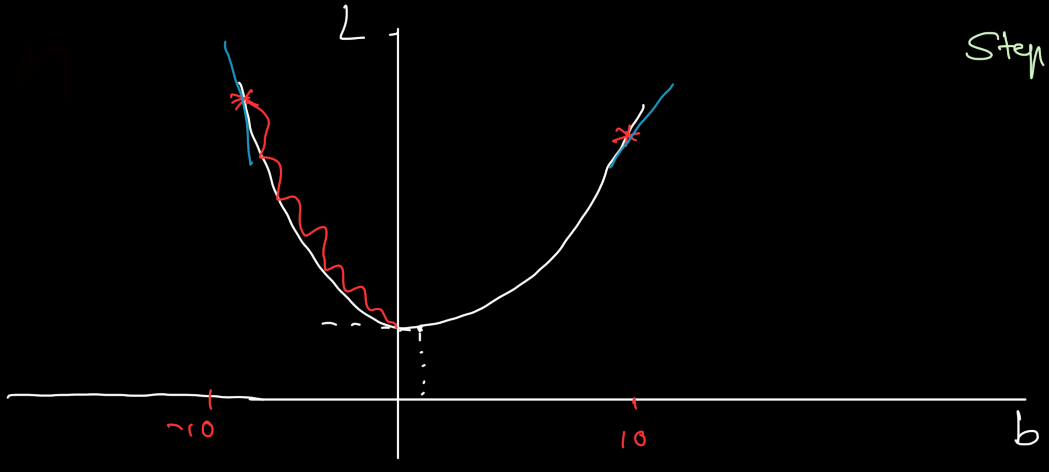
$$\hat{y}_i = mx_i + b$$

$$L = \sum_{i=1}^{4} (y_i - mx_i - b)^2$$

$$L(m, b)$$

Suppose $m = 78.35$

$$L = \sum_{i=1}^{4} (y_i - 78.35x_i - b)^2$$



Step 1: select a random b

$b = -10$

Slope = -ve then increment in b value

Slope = +ve then decrement in b value

$$b_{new} = b_{old} - Slope$$

$$b_{new} = b_{old} - \eta \, Slope$$

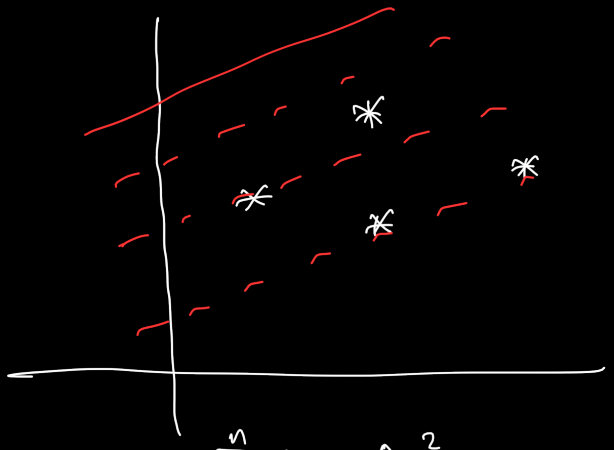
η → learning Rate

When to Stop

1) difference b/w bold and bnew >= 0.0001

2) fixed number of iterations such as 1000, 100
   ↳ epochs

Mathematical Formulation



$m = 78.35$

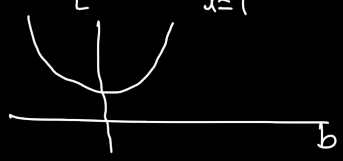
Step 1 → Start with random b

for i in epochs: (1000)

$\eta = 0.01$

$b_{new} = b_{old} - \eta \times slope$

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$



$$\frac{dL}{db} = \frac{d}{db}\left(\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right)$$

$$= \frac{d}{db}\sum_{i=1}^{n}(y_i - mx_i - b)^2$$

$$= 2\sum_{i=1}^{n}(y_i - mx_i - b)(-1)$$

$$\text{Slope} = -2\sum_{i=1}^{n}(y_i - mx_i - b)$$

at $b = 0$

$$= -2\sum_{i=1}^{n}(y_i - 78.35 \times x_i - 0) = \text{slope}_{bold}$$

$$b_{new} = b_{old} - \eta\,\text{slope}_{bold}$$

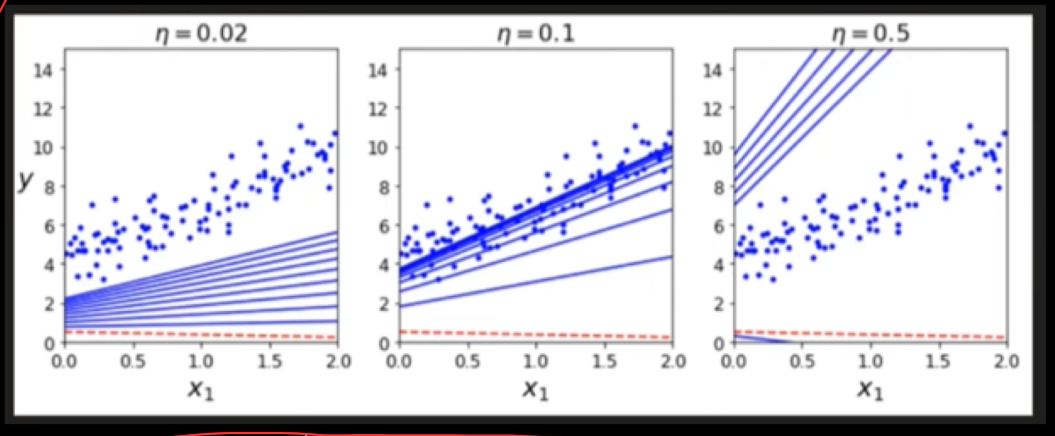
Step Size

Code from Scratch

Visualization 1

Few Discussions

1. Effect of Learning rate
2. The universality of Gradient Descent



epoch = 10

$b_{new} = b_{old} - \eta\,\text{Slope}$ → we only need slope to use GD. And slope comes from derivative of Loss function.

Now any loss function which is differentiable at every point can be used in GD for optimization.

Adding m into the equation

Steps

1) initialize random value for m and b

$m = 1$ and $b = 0$

2) epochs = 100 , lr = 0.01

for i in epochs:

$b = b - \eta \, Slope_b$

$m = m - \eta \, Slope_m$

$$L = \sum (y_i - \hat{y}_i)^2$$

$$L = \sum (y_i - mx_i - b)^2$$

$L(m, b)$ { Loss fn is a function of m & b }

$$Slope_b = \frac{\partial L}{\partial b}$$

$$Slope_m = \frac{\partial L}{\partial m}$$

$$\frac{\partial L}{\partial b} = -2 \sum (y_i - mx_i - b)$$

$= Slope_b$ at $b = 0$

$$\frac{\partial L}{\partial m} = -2 \sum (y_i - mx_i - b) x_i$$

$= Slope_m$ at $m = 1$

Visualization 2

Effect of learning rate (Hyperparameter)

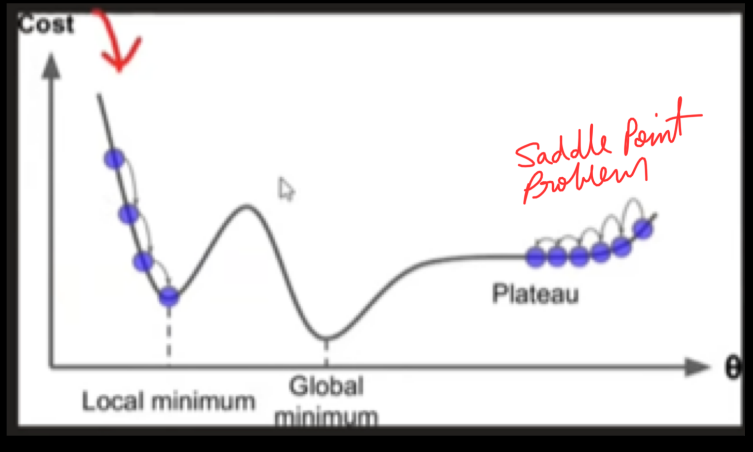
If learning rate is too small : Gradient descent will take too long to reach the minimum loss.

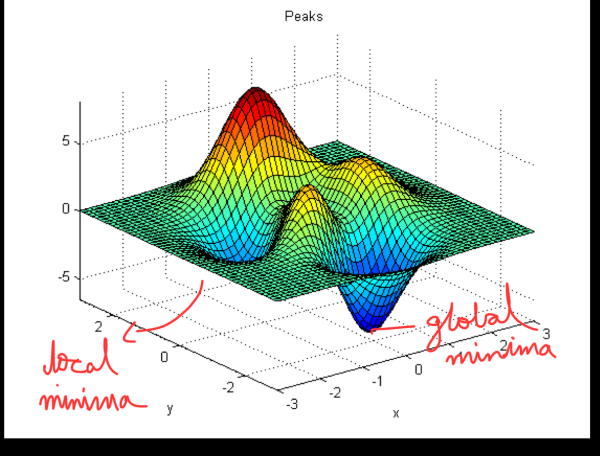
optimal learning rate : Model reach minimal loss with fewer steps

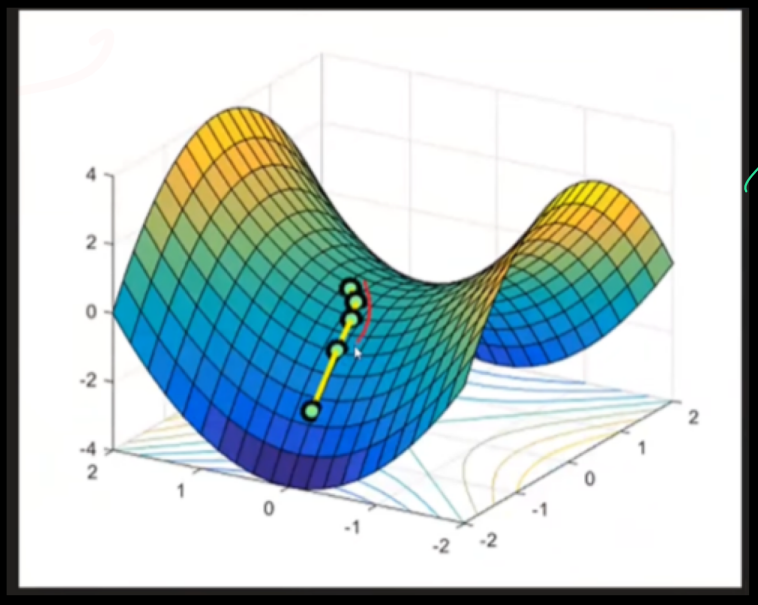
If learning rate is too big : Gradient descent will take too long or never reach the minimum loss.

Effect of Loss Function

$$L = \sum (Y_i - \hat{Y}_i)^2 \longrightarrow \text{Convex function}$$







Saddle point Problem: Slope changes very slowly.

Effect of data

If features in dataset are all in same scale then contour plot is circular and we descent very fast.