Solving optimization problems:
Single – variable differentiation:

Single-Variable diff'n: $y = f(x)$

$y = f(x)$

$\left\{ \dfrac{dy}{dx} = \dfrac{df}{dx} = y' = f' \right\}$

$\hookrightarrow$ differentiation of y w.r.t x

$\dfrac{y_2 - y_1}{x_2 - x_1} = \dfrac{\Delta Y}{\Delta X} = \operatorname{Tan}\theta$

$(y_2 - y_1)$
$(x_2 - x_1)$

"$\dfrac{dy}{dx}$" $\rightarrow$ intuitively $\rightarrow$ rate of change of y as x changes
$\rightarrow$ how much does y change as x changes

scalar
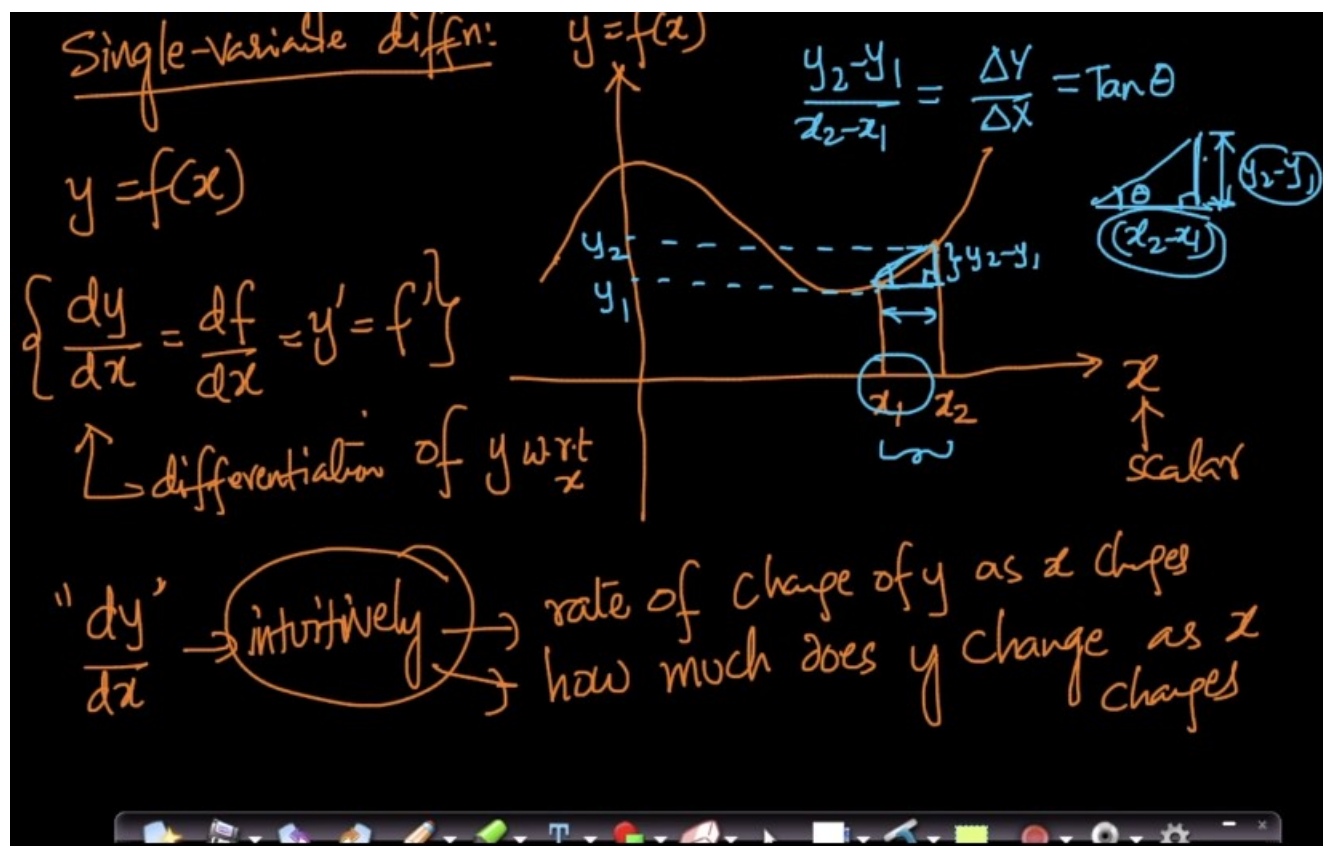
Mathematical formulation:
When limit x → 0 , then it becomes a tangent at that point on the curve, tangent is the hyp that is obtained when x→ 0.

$\left[\dfrac{dy}{dx} = \operatorname{Lim}_{\Delta x \to 0} \dfrac{\Delta Y}{\Delta X} \right.$

$y = f(x)$

geom·intr

Tangent off $f(x)$ @ x_1

Tangent is the hyp that
We obtain as $\delta x \to 0$

$\boxed{\operatorname{Tan}\theta = \dfrac{dy}{dx}}$ as $\delta x \to 0;$ $\dfrac{\delta Y}{\delta Y} = \dfrac{dy}{dx}$

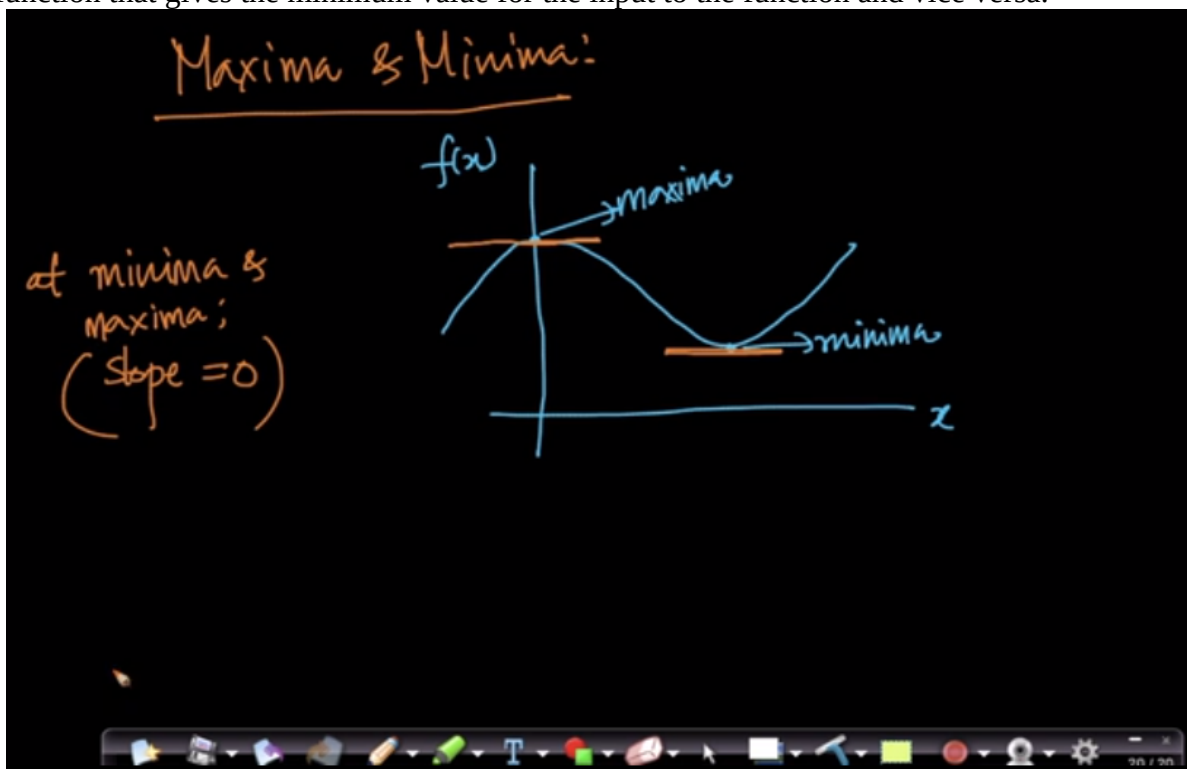$x_2 \to x_1 \Rightarrow \delta x \to 0$
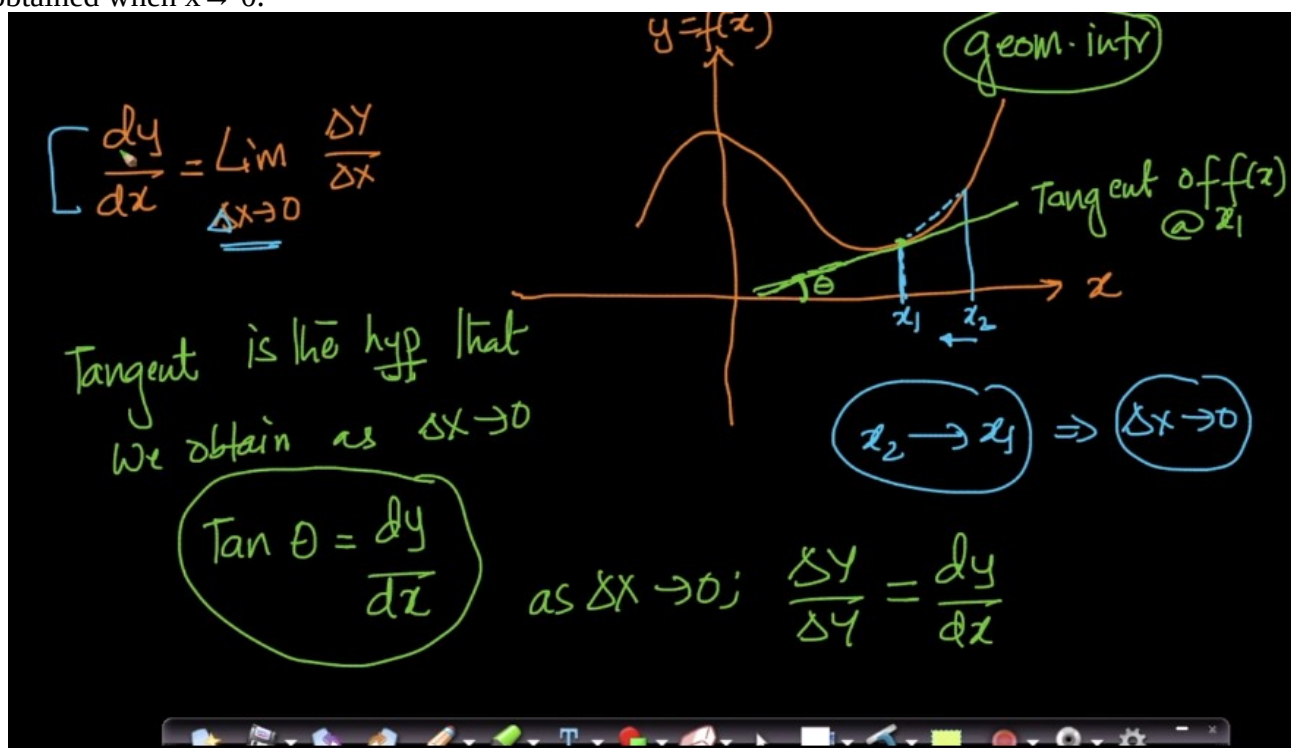
$$\frac{dy}{dx} = \text{Slope of the tangent to } f(x)$$

$$\left[\frac{dy}{dx}\right]_{x_1} = \text{Slope of the tangent to } f(x) @ x = x_1$$

Geometrically dy / dx means the tangent at that point.

Minima and Maxima:

The function that gives the minimum value for the input to the function and vice versa.

Maxima & Minima:

at minima & maxima;
(slope =0)

f(x)

→maxima

→minima

x

Example: The slope tangent the value of x for the derivative of the function in zero.
By substituting the values near to x we can know the nature of the function.

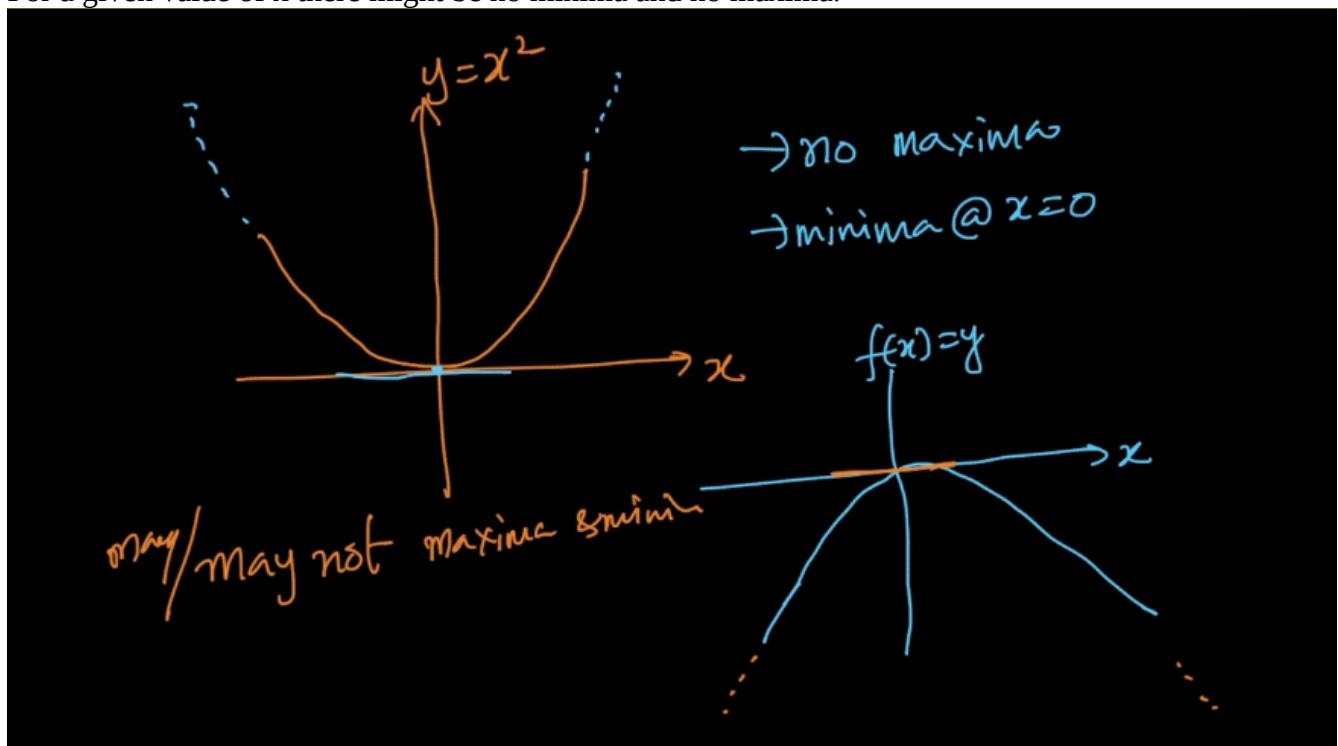$$f(x) = x^2 - 3x + 2$$

<u>maxima</u> and/or <u>minima</u>

Eqn: $\boxed{\dfrac{df}{dx} = 0}$ $\Rightarrow$ slope = 0

@ $x = \dfrac{3}{2}$; slope = 0

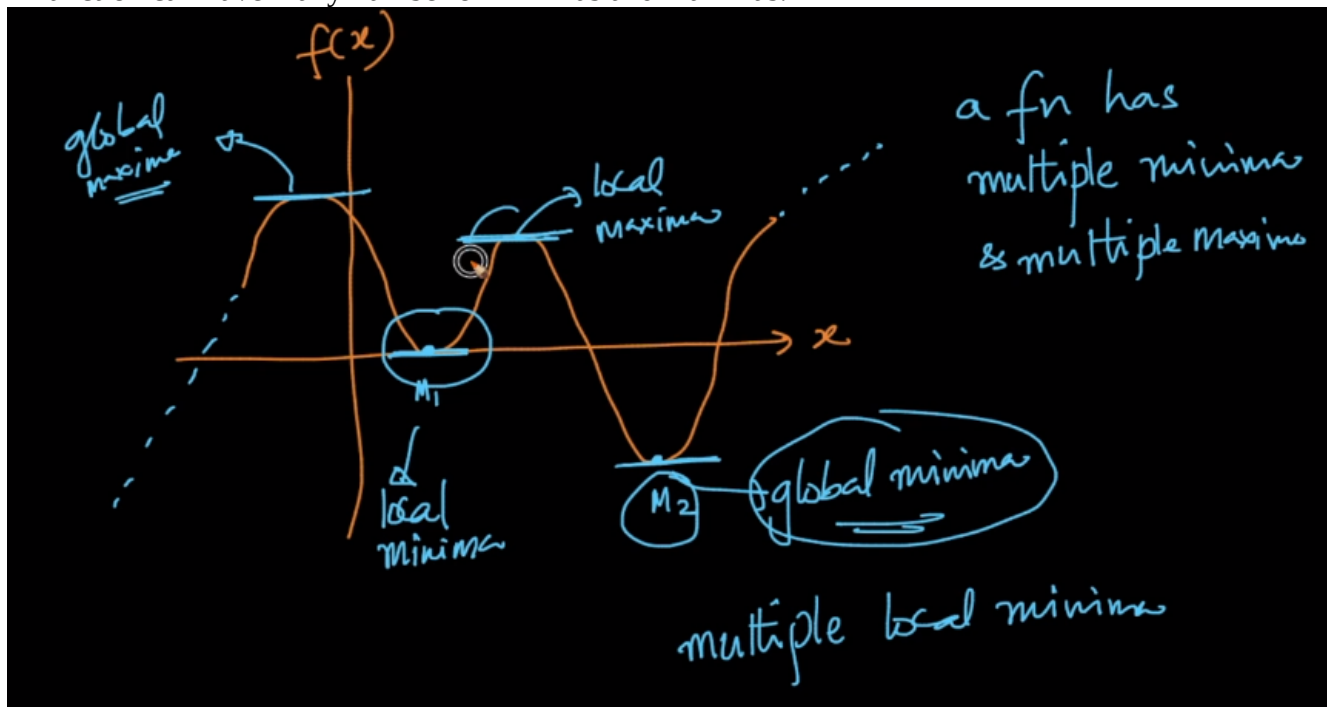$\dfrac{df}{dx} = 2x - 3 = 0$ $\Rightarrow$ $\boxed{x = \dfrac{3}{2} = 1.5}$

(Q) $f(\boxed{1.5}) = -0.25$

$f(1) = 0$

$f(1.5) < f(1)$

$\boxed{\text{minima @ } x = 1.5}$

$\Rightarrow$ 1.5 cannot be maxima

For a given value of x there might be no minima and no maxima.

$y = x^2$

$\rightarrow$ no maxima

$\rightarrow$ minima @ x = 0

$f(x) = y$

may/may not maxima &minima

A function can have many number of minimas and maximas.

Non-trival cases of derivatives:

Vector differentiation:

Vector differentiation: Grad

x : scalar $f(x)$

$\{ \underline{x}$: Vector $\rightarrow$ high-dim. space

$f(x) = \boxed{y = a^T x}$; $x = \langle x_1, x_2, \dots x_d \rangle$

$ a = \langle a_1, a_2, \dots a_d \rangle \rightarrow$ constants

$y = \sum\limits_{i=1}^{d} a_i x_i$

Differentiating a vector:

$$\frac{df}{dx} = \nabla_x f \qquad \xrightarrow{\text{grad(or) Del}}$$

$\uparrow$
$\boxed{\text{Vector}}$

$\boxed{\dfrac{df}{dx}}$

$$\nabla_x f = \begin{bmatrix} \dfrac{df}{dx_1} \\ \dfrac{df}{dx_2} \\ \vdots \\ \dfrac{df}{dx_d} \end{bmatrix} \xrightarrow{\hspace{1cm}} \text{Vector} \in \mathbb{R}^d$$

$$\frac{df}{dx_1} = \boxed{\frac{\partial f}{\partial x_1}} \rightarrow \text{partial diff}^n.$$

Differentiating a function f, which is a vector:

$$f(x) = y = a^T x = \sum_{i=1}^{d} a_i x_i = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix} = a$$
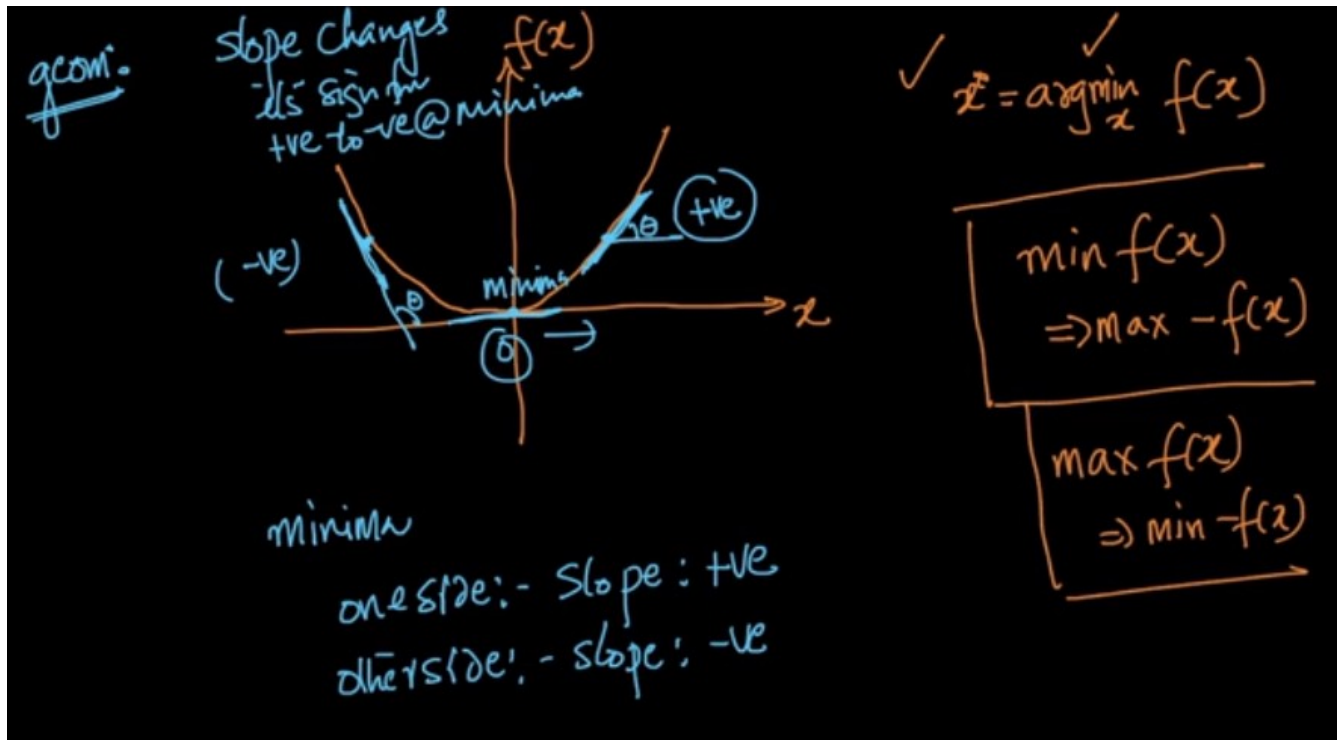
$$\frac{\partial f}{\partial x_1} = a_1$$

$$\frac{\partial f}{\partial x_2} = a_2$$

$$\nabla_x(a^T x) = a$$

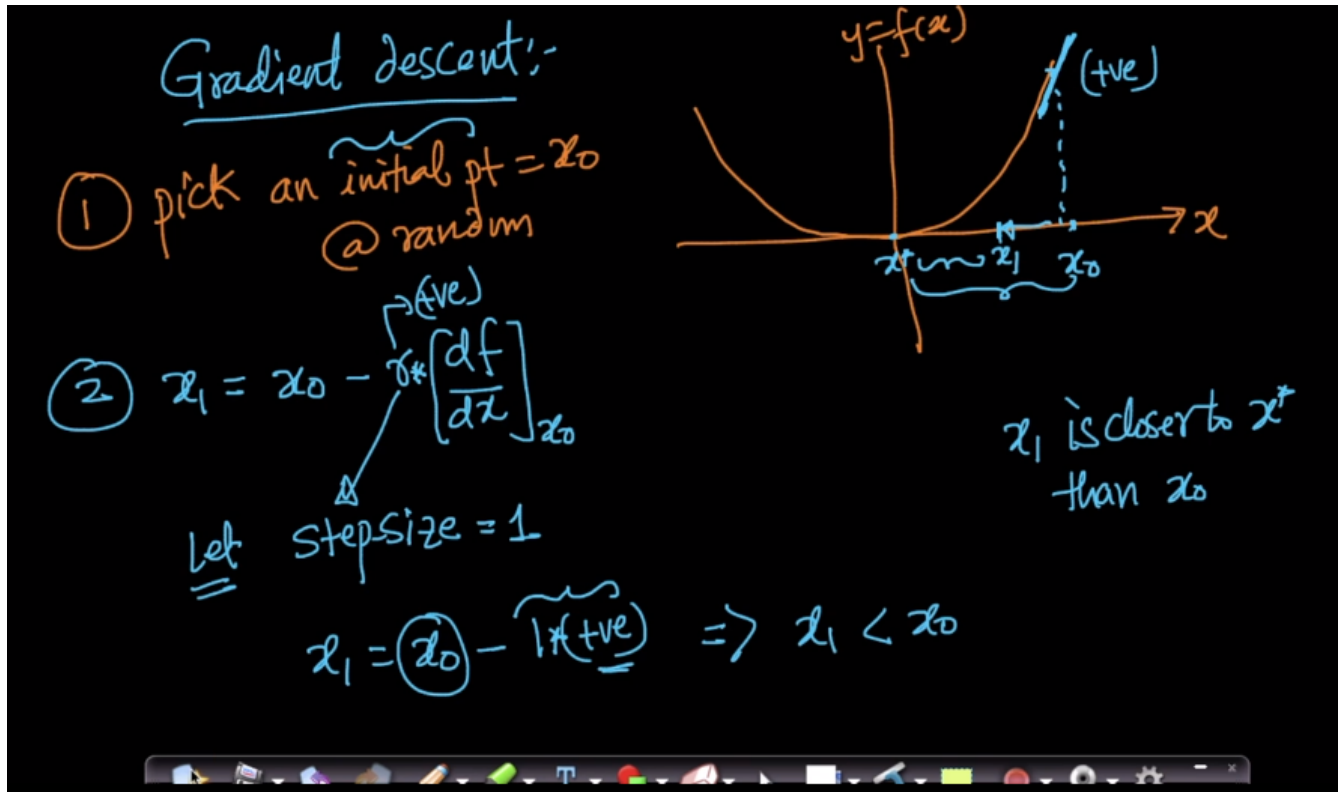$$\frac{d}{dx}(ax) = a$$

We use gradient descent for calculating the minimum of the function, because as the terms increase the functions become complex.
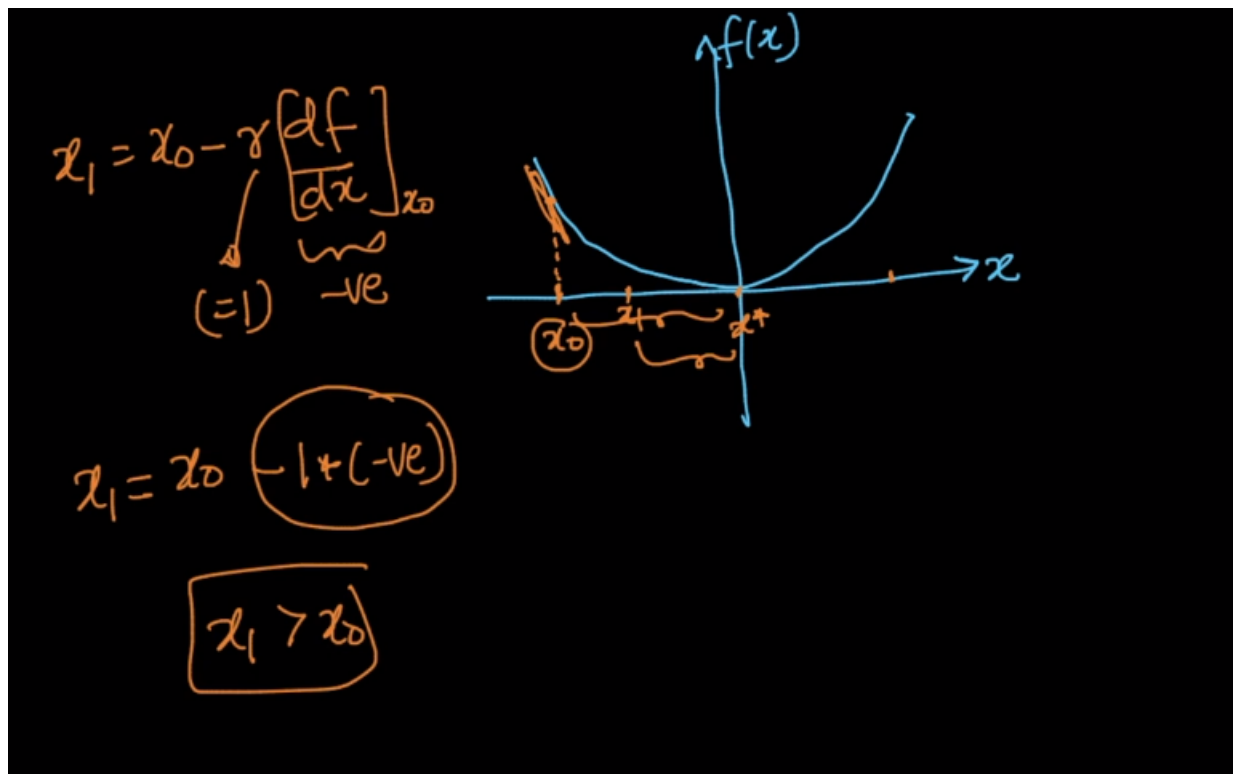Gradient descent algorithm: It is an iterative algorithm. As the iterations increase the minimum value is achieved.
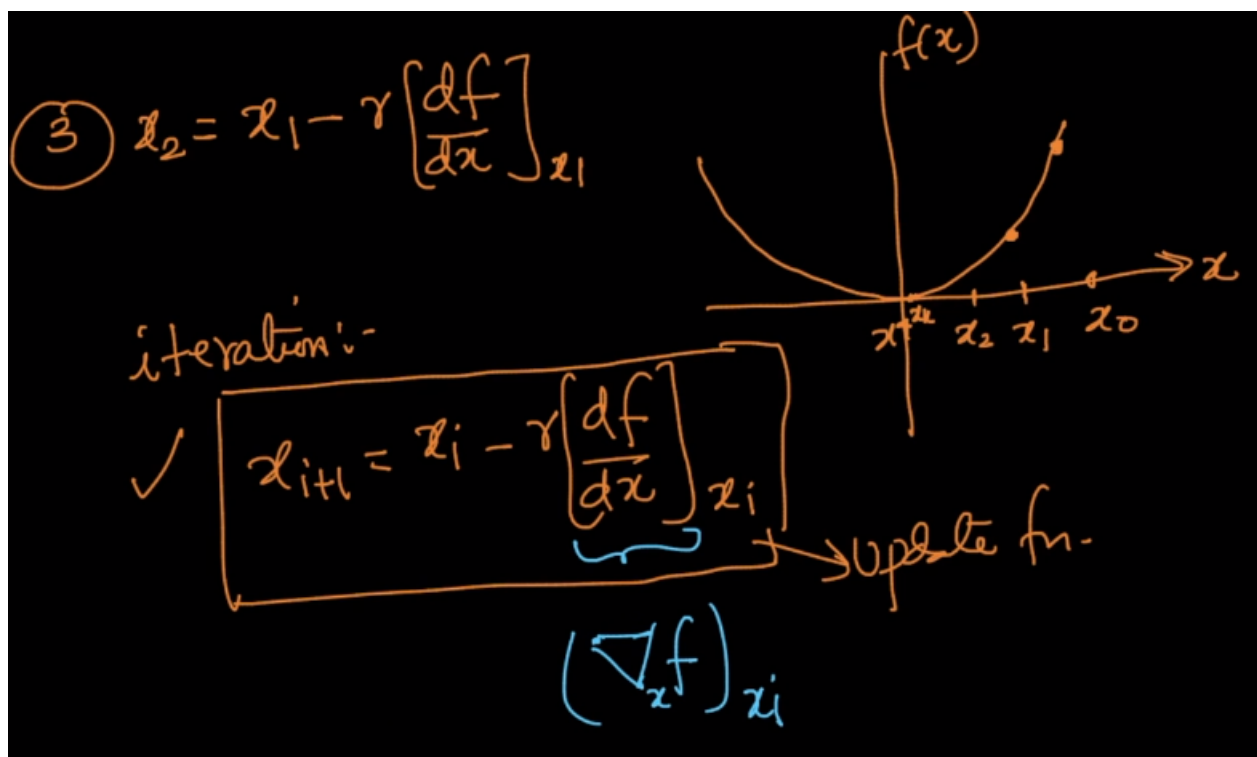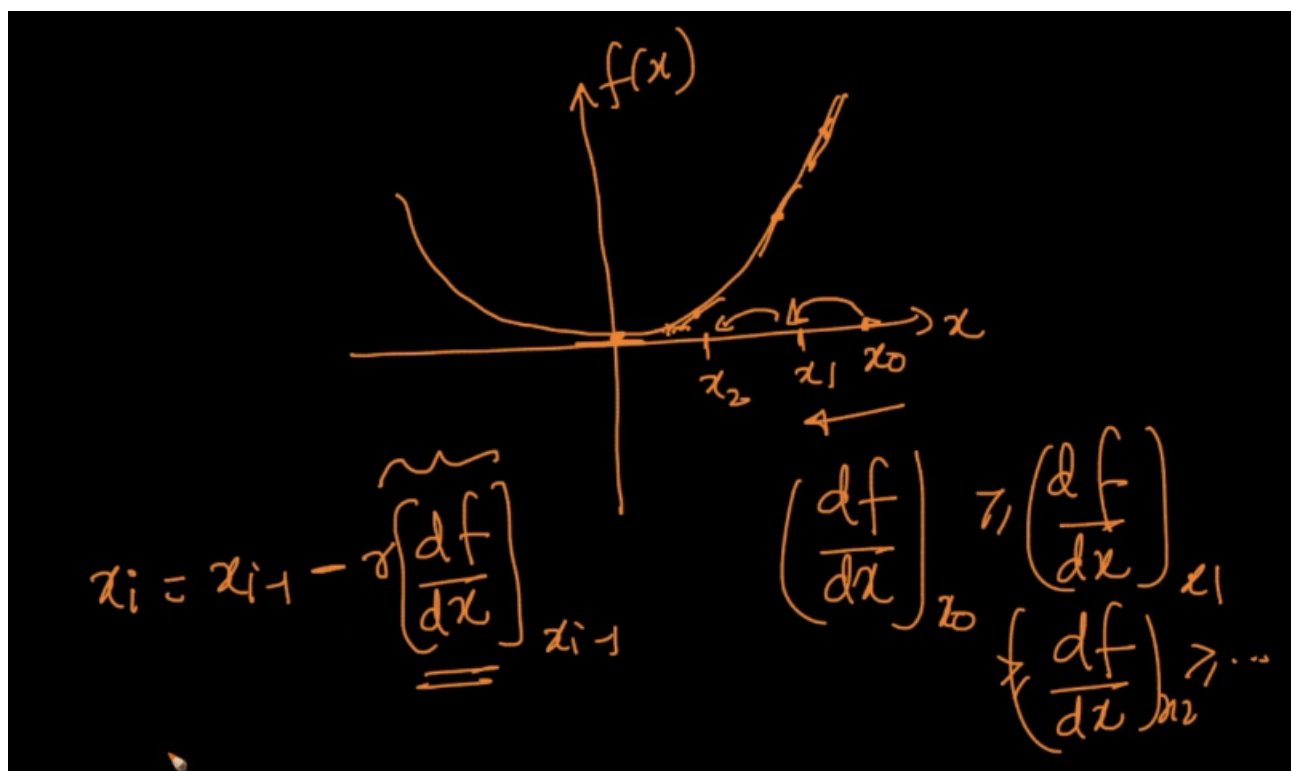
For positive slope.

Gradient descent:-

① pick an initial pt $= x_0$ @ random

② $x_1 = x_0 - \gamma * \left[\dfrac{df}{dx}\right]_{x_0}$ (+ve)

Let stepsize $= 1$

$x_1 = x_0 - 1*(+ve) \implies x_1 < x_0$

$y = f(x)$

(+ve)

$x^* \rightsquigarrow x_1 \quad x_0$

x_1 is closer to x^* than x_0

For negative slope.

$x_1 = x_0 - \gamma \left[\dfrac{df}{dx}\right]_{x_0}$

$(=1) \quad$ -ve

$x_1 = x_0 - (1 + (-ve))$

$\boxed{x_1 > x_0}$

$f(x)$

$x_0 \quad x_1 \rightsquigarrow x^*$

Another point on the curve is achieved by updating the step size by derivative at the previous point.

③ $x_2 = x_1 - \gamma \left[\dfrac{df}{dx} \right]_{x_1}$

iteration :-

$\checkmark$ $\boxed{ x_{i+1} = x_i - \gamma \underbrace{\left[\dfrac{df}{dx} \right]_{x_i}}_{\rightarrow \text{update fn.}} }$

$\left(\nabla_x f \right)_{x_i}$

Property of Gradient descent:

$x_i = x_{i-1} - \gamma \overbrace{\left[\dfrac{df}{dx} \right]}^{}{}_{x_{i-1}}$

$\left(\dfrac{df}{dx} \right)_{x_0} \; 7, \; \left(\dfrac{df}{dx} \right)_{x_1}$

$\not{} \left(\dfrac{df}{dx} \right)_{x_2} 7 \cdots$
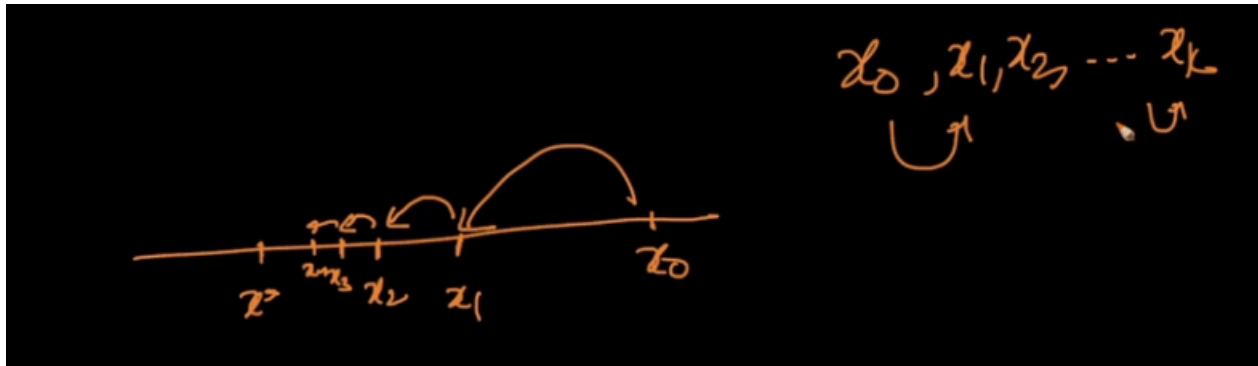
The first step will the big, the next steps will be decreased eventually to attain the minimum of the function, that is achieved when the derivative is relatively small.
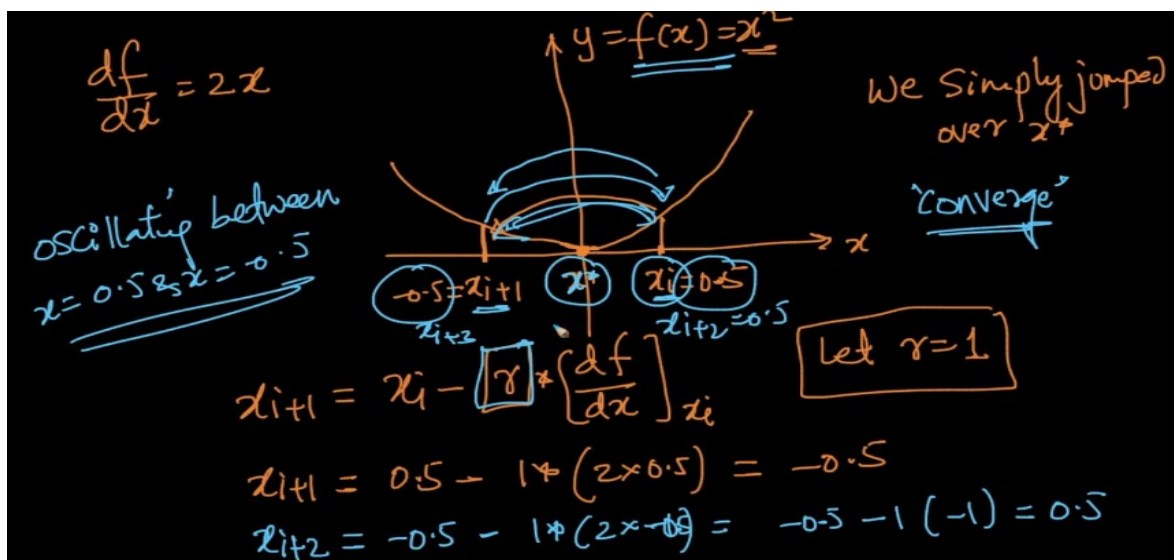

Learning rate (or) step-size:

'r' is called the learning rate.

If learning rate is not appropriate then the lowest point in not achieved.

This problem is called oscillation problem, the only solution for this problem is changing 'r' with each iteration. We can create a function such that 'i' increases 'r' should decrease.

remedy for oscillation:- ✓

change $\boxed{r}$ with each iteration

→ one tech:- reduce (r) with each $\underline{\text{itern}}$

$\left(r = \text{function of iteration number} \right)$

$r = h(i)$ s.t $i \uparrow ; r \downarrow$

↳ iteration

Gradient descent for linear regression:

$$L(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

$\langle x_i, y_i \rangle \to D_{train}$

$$\nabla_w L = \sum_{i=1}^{n} \left\{ 2(y_i - w^T x_i)(-x_i) \right\}$$

① pick a random vector $w_0 = \langle \cdots \cdots \rangle$

② $w_1 = w_0 - r * \sum_{i=1}^{n} (-2x_i)(y_i - w_0^T x_i)$

③ $w_2 = w_1 - r * \sum_{i=1}^{n} (-2x_i)(y_i - w_1^T x_i)$

The iterations are made till the difference of the previous step and next step is low as and the 'w' is finalized.

If n is large for every step calculations of the total data points must be done, this is computationally expensive.

Gradient descent for linear regression:

$$w^+ = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \underbrace{\quad}_{\text{no-reg}}$$

not using w_0

The loss function is written as the function of L.

$$L(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2 \qquad \langle x_i, y_i \rangle \rightarrow D_{\text{train}}$$

$$\nabla_w L = \sum_{i=1}^{n} \left\{ 2(y_i - w^T x_i)(-x_i) \right\}$$

① pick a random vector $w_0 = \langle \ldots \ldots \rangle$

② $w_1 = w_0 - \gamma * \sum_{i=1}^{n} (-2x_i)(y_i - w_0^T x_i)$

③ $w_2 = w_1 - \gamma * \sum_{i=1}^{n} (-2x_i)(y_i - w_1^T x_i)$

The calculation of w's are made till the vector W does not change.

$$W_0, W_1, W_2, \cdots \quad \underline{W_k} \quad W_{k+1}$$

$$W_{k+1} - W_k \quad \text{is a } \underset{\wedge}{\text{very}} \overset{\text{vectr of}}{\text{small values}}$$

$$W^* = W_k \checkmark$$

If 'n' is large, gradient descent becomes time consuming process.

Gradient desc

$$W_j = W_{j-1} - r \left[\sum_{i=1}^{n} (-2x_i)(y_i - W^T x_i) \right]$$

problem: W_{j-1} to W_j

$$W_0, W_1, W_2$$

if n is large

n = 1 Million

if (n) is large
computing this $\sum$ is very expensive

To solve this problem, we use **stochastic gradient descent.**

SGD Algorithm:

SGD chooses random set of data points from the whole data set where the number of points are less than whole points, the chosen points can even be one.

We will use the different set of random points for every iteration, this is also called batch size in SGD. When k in greater that 1 we will batch gradient descent.

GD:- $^{(n=1Mill)}$ $\boxed{100}$ iteration to converge x^+

SGD:- $\cancel{K}=1000;$ if >100 iter$\underline{ns}$ x^ϕ $\boxed{500}$

SGD: $\cancel{K}=10;$ $\boxed{1000}$; x^ϕ

$K = \#$ of random pts that you @ each iteration fr update

From one iteration to another iteration we will use different set of points.

Constrained optimization & PCA:

$\checkmark$ Constrained Optimization

$\max\limits_{x} f(x)$ (r) $\min\limits_{x} f(x)$ $\rightarrow$ objective fn.

PCA: $\max\limits_{u} \dfrac{1}{n} \sum\limits_{i=1}^{n} (u^T x_i)^2$

$\max\limits_{x} f(x)$

$\sim$ s.t $g(x) = c$

$\checkmark$ s.t $\boxed{u^T u = 1}$

constraint

Here the constraint is U is a unit vector.

We will solve the constraint based optimization functions using lagrangean multipliers.

PCA:
    Modification of PCA.

PCA :- $\max\limits_{u} u^T S u$

$\qquad$ s.t $u^T u = 1$

$$S = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T$$

$\rightarrow$ Co-var Matrix of X

$$L(u,\lambda) = u^T S u - \lambda(u^T u - 1)$$

$$\frac{\partial L}{\partial u} = 0 \implies \frac{\partial}{\partial u}\left(u^T S u - \lambda u^T u - \lambda\right) = 0$$

$$\implies \boxed{S u - \lambda u = 0} \implies \boxed{S u = \lambda u}$$

$$\boxed{S u = \lambda u} \longrightarrow \text{definition of eigen-val \& eigen vec}$$

$\downarrow$ vector

Cov. Matrix

$$\boxed{A} u = \lambda u$$

$\uparrow$ eig vec $\qquad$ $\uparrow$ eigen val

$\uparrow$ eig vec

$\begin{cases} u \text{ is the eigen vec of } S \\ \lambda \text{ is the eigen val of } S \end{cases}$

$$\text{PCA :-} \quad \max_u u^T S u \} \checkmark$$
$$\text{s.t } u^T u = 1$$

$$\boxed{S = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T}$$

$\hookrightarrow$ Co-var Matrix of X

$$\mathcal{L}(u, \lambda) = u^T S u - \lambda (u^T u - 1)$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial u} = 0} \Rightarrow \frac{\partial}{\partial u} \left(u^T S u - \lambda u^T u - \lambda \right) = 0$$

$$\Rightarrow \boxed{S u - \lambda u = 0} \Rightarrow \boxed{S u = \lambda u}$$

Logistic regression formulation revisited:

$$\text{Logistic regression revisited} \qquad \rightarrow \text{lagrange multipliers}$$

$$w^\dagger = \arg\min_w \left(\text{logistic -loss} \right) + \boxed{\lambda w^T w}$$
$$\uparrow \text{reg}$$

$$\underline{w}^\dagger = \arg\min_w \left(\text{logistic -loss} \right)^{\rightarrow \text{obj fn}}$$

Const.
optmzn $\{$
$$\text{s.t } \boxed{w^T w = 1}$$
$$\hookrightarrow \text{eq. constr.}$$

Regularization can be taught as an equality constraint that is being imposed on logistic loss.

Why L1 regularization creates sparsity?

Plotting the L2 and L1 of each value of the weight vector.

$\mathcal{L}_2$

$R_2(w_1) = |w_1|^2$

w_1

$\mathcal{L}_2(w_1) = w_1^2$

$\dfrac{\partial \mathcal{L}_2}{\partial w_1} = 2 * w_1$

$\partial \mathcal{L}_2 / \partial w_1$

w_1

GD:

$(w_1)_0, \ (w_1)_1, \ (w_1)_2 \cdots \ (w_1)_k$

$\mathcal{L}_1$

$R_1(w_1) = |w_1|$

w_1

$\mathcal{L}_1(w_1) = |w_1| = \begin{cases} w_1 \ \text{if} \ w_1 > 0 \\ -w_1 \ \text{if} \ w_1 \le 0 \end{cases}$

$\dfrac{\partial \mathcal{L}_1}{\partial w_1} = \begin{cases} +1 \ \text{if} \ w_1 > 0 \\ -1 \ \text{if} \ w_1 < 0 \end{cases}$

$\partial \mathcal{L}_1 / \partial w_1$

1

w_1

-1
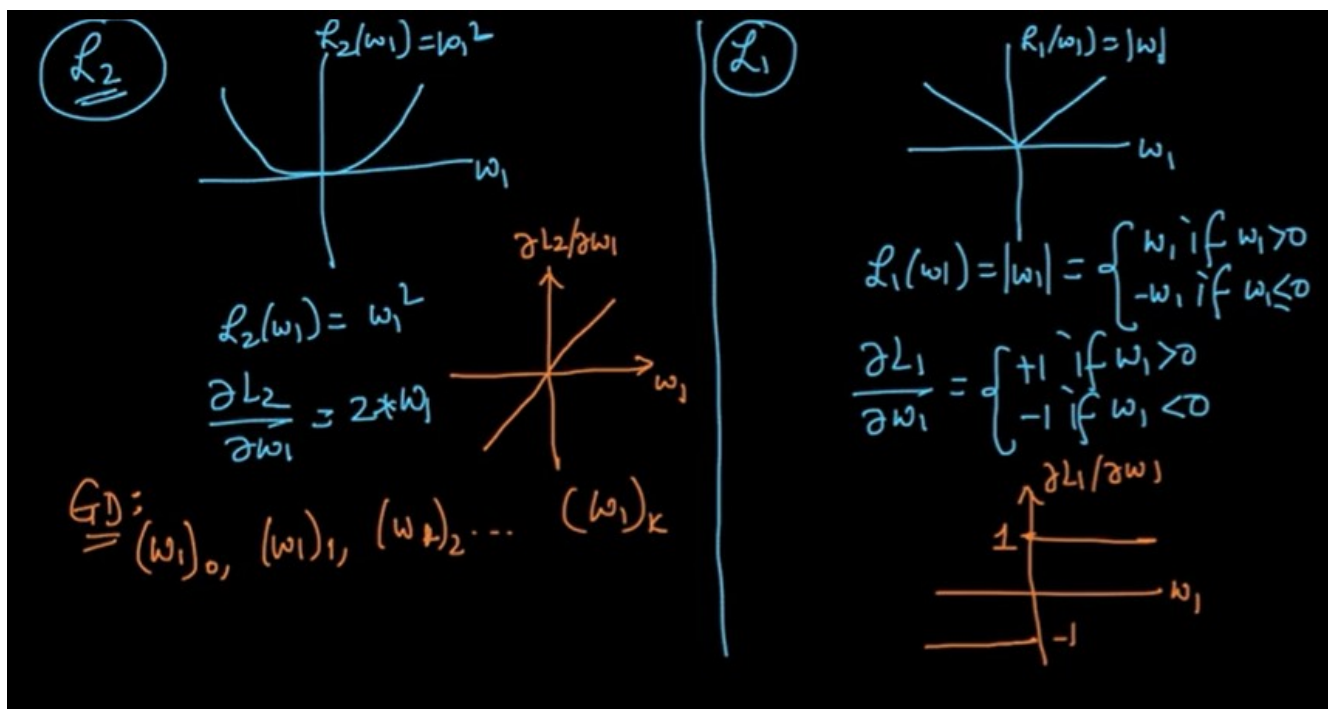
The rate of convergence will decrease in L2.

$\mathcal{L}_2$ 0.049

$(w_1)_{j+1} = (w_1)_j - \gamma \left[\dfrac{\partial \mathcal{L}_2}{\partial w_1} \right]_{w_1 j}$

Let $(w_1)_j$: +ve

$(w_1)_{j+1} = (w_1)_j - \gamma (2 * w_1 j)$ $\boxed{0.05 - 0.00}$

$(w_1)_j = 0.05$ SMall

Let $\gamma = 0.01$

(small)

V-Small

$2 * 0.05 * 0.01 = 0.001$

$\mathcal{L}_1$

$(w_1)_{j+1} = (w_1)_j - \gamma \left(\dfrac{\partial \mathcal{L}_1}{\partial w_1} \right)_{w_1 j}$

$(w_1)_{j+1} = (w_1)_j - \gamma * 1$ $\boxed{0.05 - 0.01}$

0.04

small

Let $\gamma = 0.01$

0.01
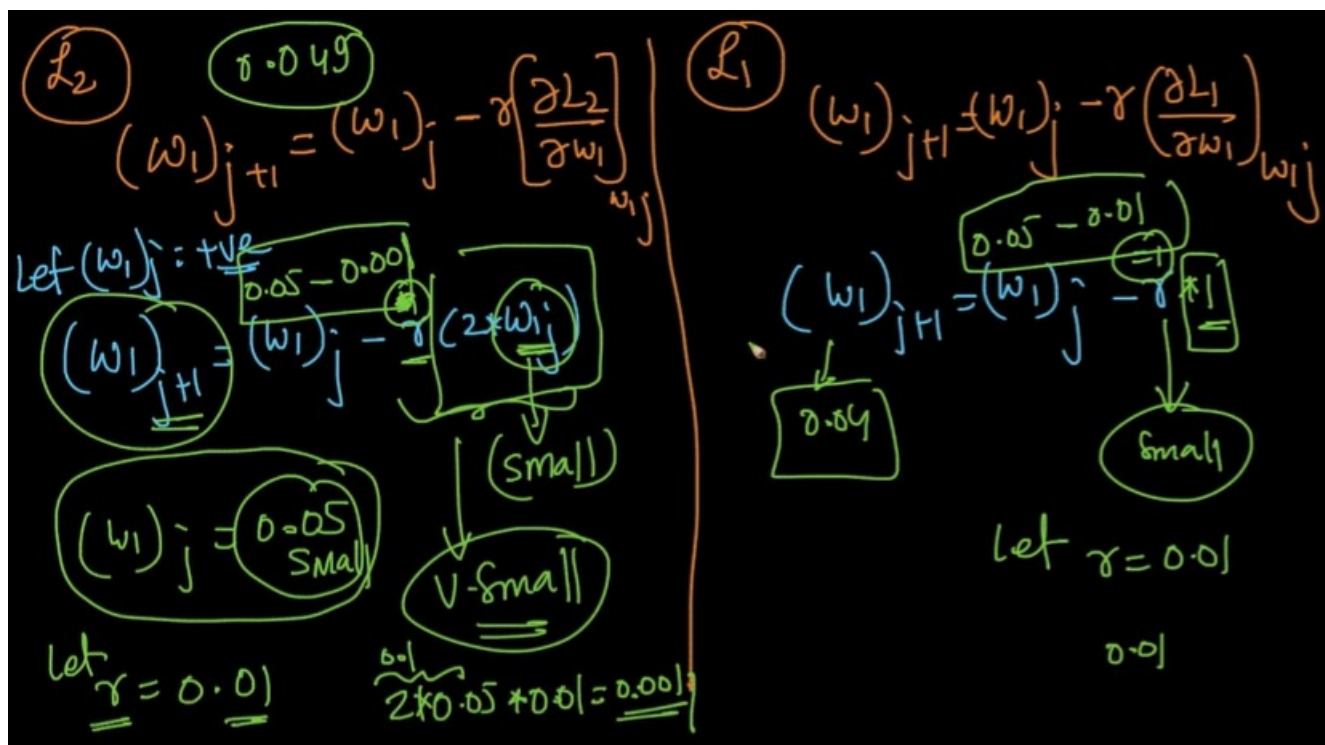
L2 regularizer doesn't change the value of w1 from one iteration to another iteration, because the gradient reduce towards w1* = 0.

v. quickly :- L_2 reg. doesn't change the value of
w_i from one iteration to another

L_1 reg. continues to constantly reduce
w_i towards $w_i^* (=0)$

Exercise:

$$\min_{w,b} \sum_{i=1}^{n} (y_i - w^T x_i - b)^2 = \mathcal{L}(w, b)$$

✓ $$\frac{\partial \mathcal{L}(w)}{\partial w} = \sum_{i=1}^{n} (-2 x_i)(y_i - w^T x_i - b)$$

✓ $$\frac{\partial \mathcal{L}}{\partial b} = \sum (-2)(y_i - w^T x_i - b)$$

$$\vec{W}_{j+1} = \vec{W}_j - \gamma\left(\frac{\partial L}{\partial w}\right)_{w_j}$$

$$b_{j+1} = b_j - \gamma\left(\frac{\partial L}{\partial b}\right)_{b_j}$$

(100 iterat^n)

K - iteral^n ➜ b^*

$\vec{W}^*$

$\boxed{W_k} \approx W_{k+1}$

$\boxed{\frac{b_k}{} \approx b_{k+1}}$

reduce γ with each iter^n:

$\gamma_0 = 1$

$\left\{\begin{array}{l} \text{iter } 0 \to \gamma = 1 \\ \quad 1 \to \gamma = \gamma/2 \\ \quad 2 \to \gamma = \gamma/2 \\ \quad \vdots \end{array}\right.$ ✓

---

§x: Implement SGD for linear regression

Lr.regrsn:- code-sample (boston home prices dataset)

↳ Linear Regression ( ) in Sklearn

(*) Implement SGD fw lr.regrsn