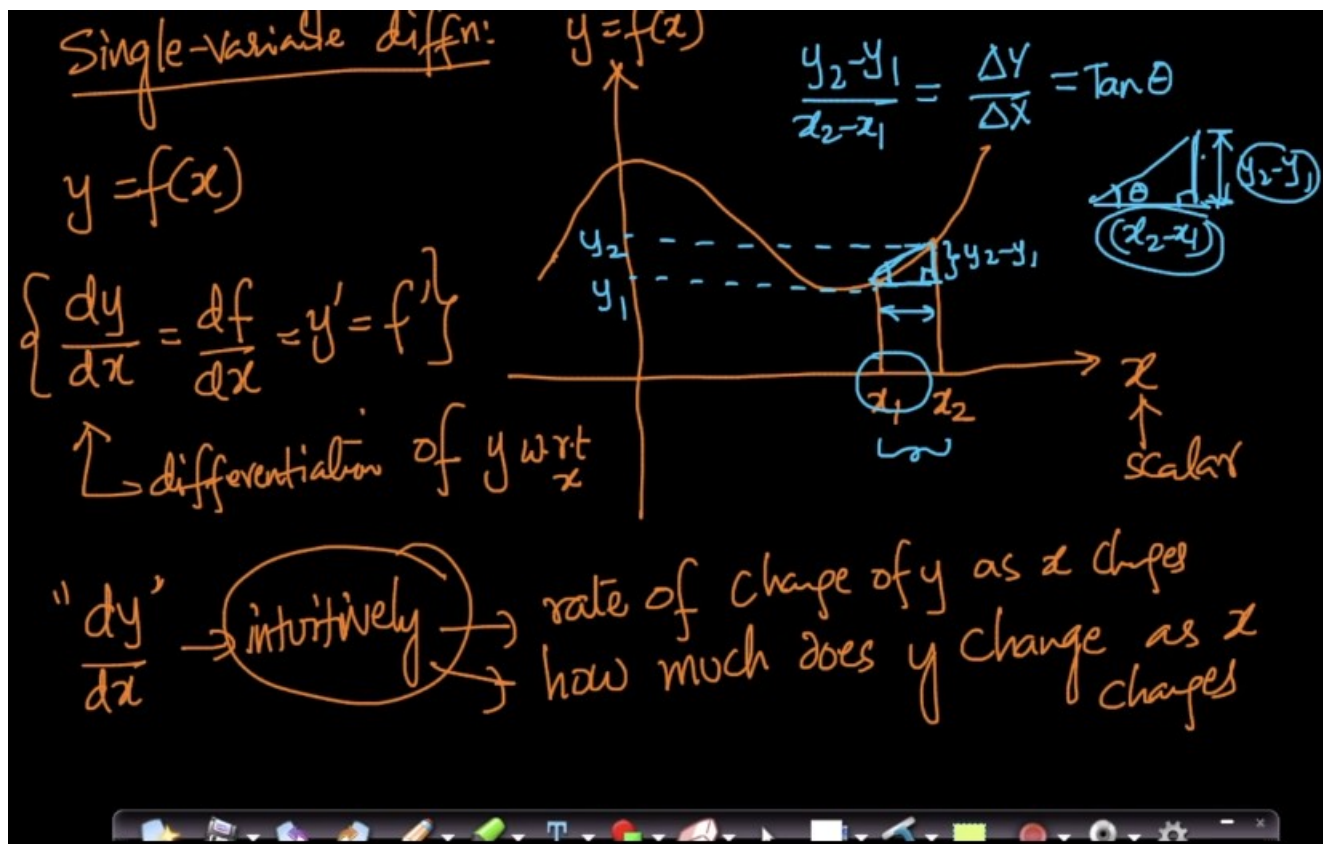
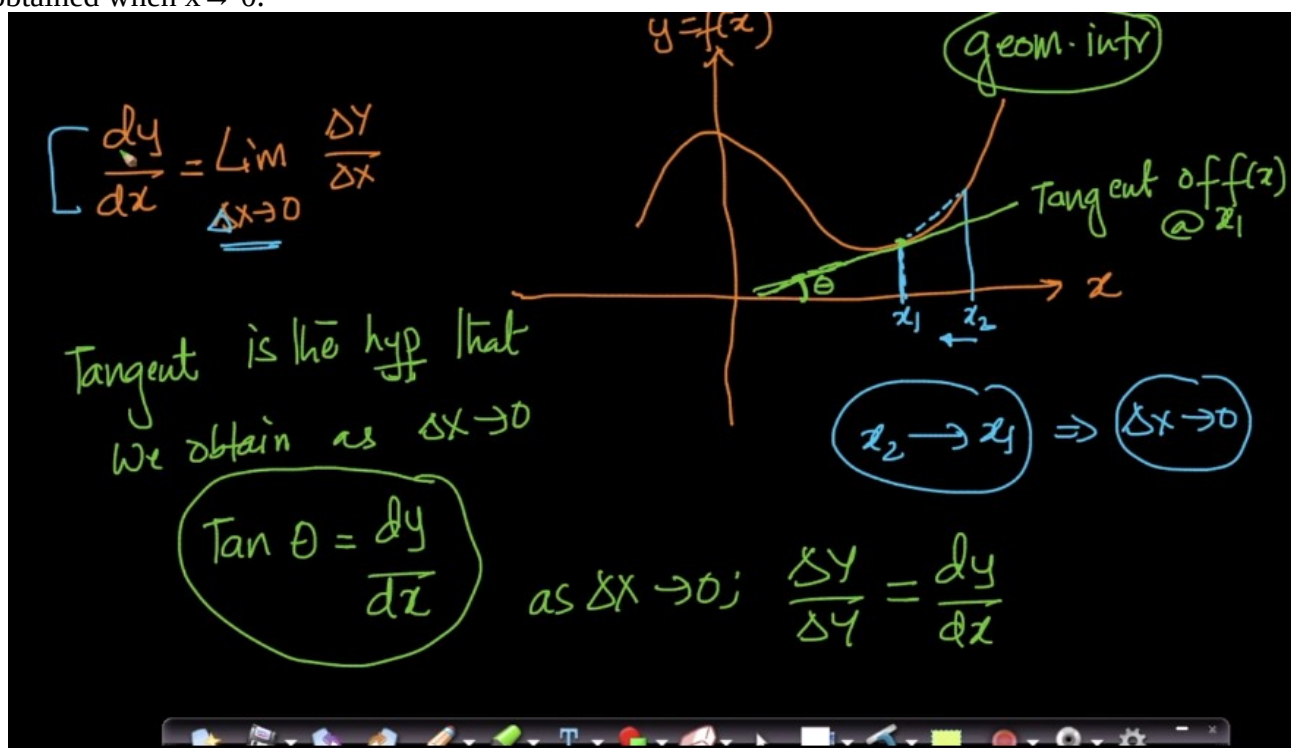


Solving optimization problems:
Single – variable differentiation:



Mathematical formulation:

When limit $\Delta x \rightarrow 0$, then it becomes a tangent at that point on the curve, tangent is the hyp that is obtained when $\Delta x \rightarrow 0$.



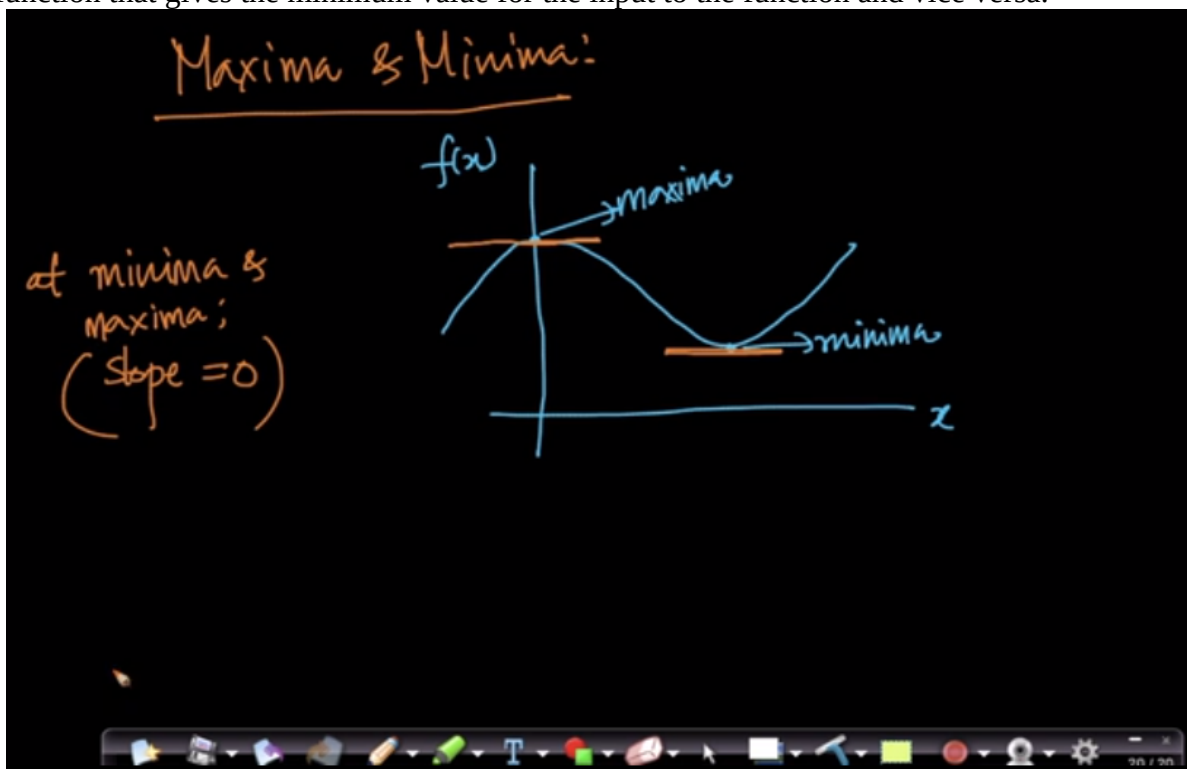
$\frac{dy}{dx}$ = slope of the tangent to $f(x)$

$\left[\frac{dy}{dx}\right]_{x_1}$ = 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.



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

$f(x) = x^2 - 3x + 2$ maxima and/or minima

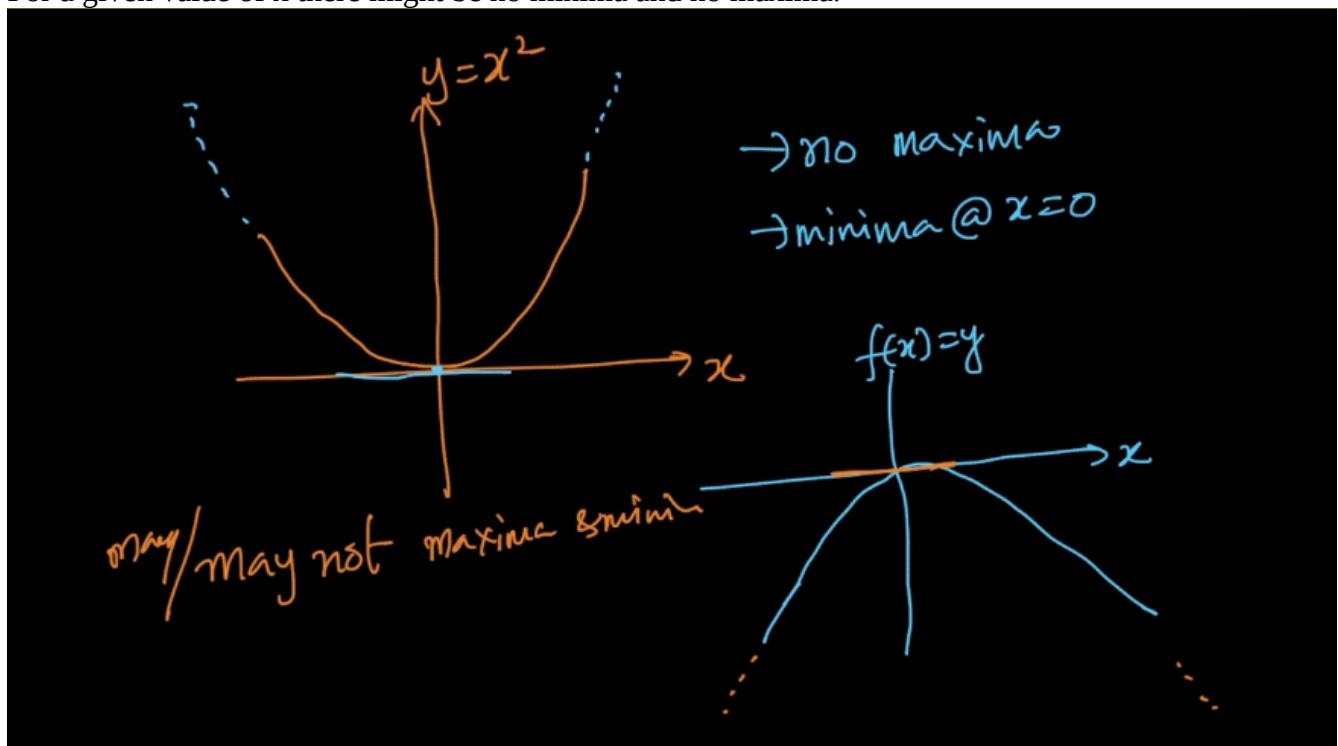
Eg: $\boxed{\frac{df}{dx} = 0} \Rightarrow \text{slope} = 0$ @ $x = \frac{3}{2}$; slope = 0

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

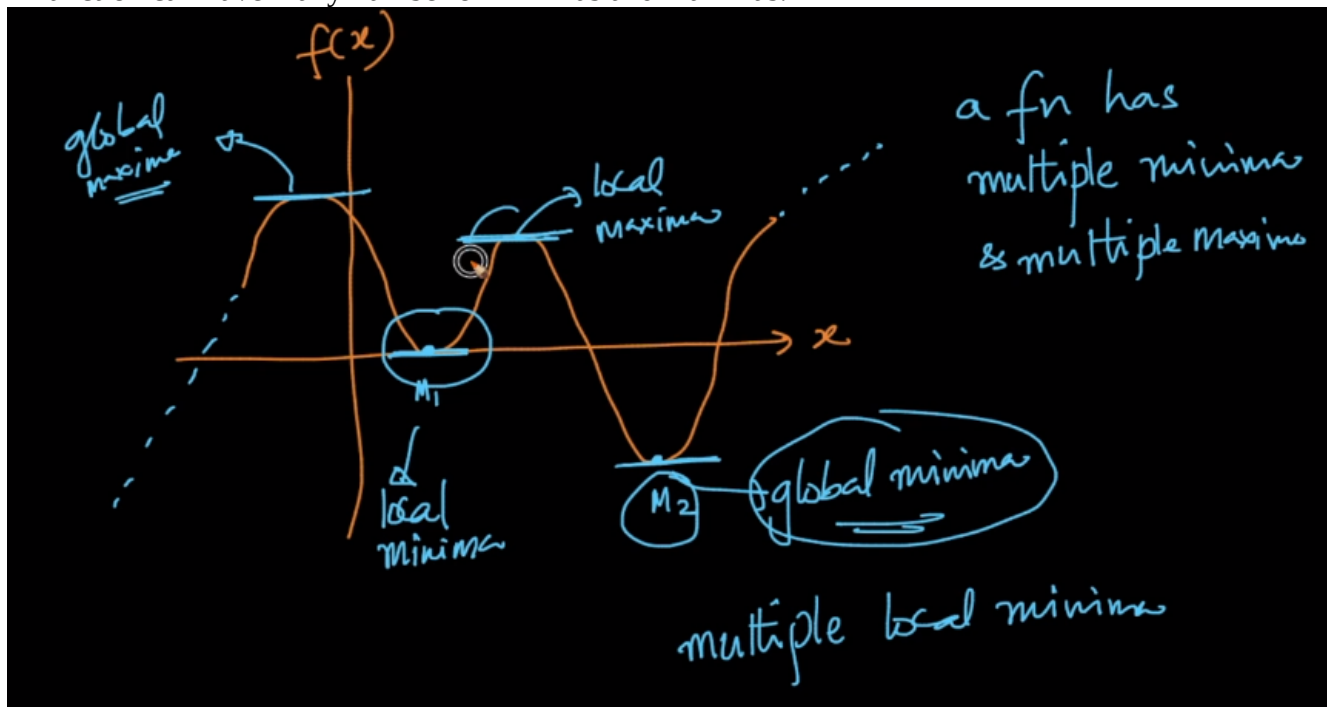
$\boxed{\text{minima @ } x = 1.5}$

(Q) $f(1.5) = -0.25$
 $f(1) = 0$
 $f(1.5) < f(1)$
 $\Rightarrow 1.5$ cannot be maxima

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



A function can have many number of minimas and maximas.



Non-trivial cases of derivatives:

min & max

$$f(x) = \log(1 + \exp(ax)) \rightarrow \text{like logistic loss}$$
$$\frac{df}{dx} = \frac{a \exp(ax)}{1 + \exp(ax)} = 0$$

$\left. \begin{array}{l} \rightarrow \text{solving this is} \\ \text{not trivial} \\ \rightarrow \text{v.v. hard} \end{array} \right\}$

$\frac{df}{dx} = 0$

Gradient descent

Vector differentiation:

Vector differentiation: Grad

x : scalar $f(x)$

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

$f(x) = y = \underline{a}^T \underline{x}$; $\underline{x} = \langle x_1, x_2, \dots, x_d \rangle$
 $y = \sum_{i=1}^d a_i x_i$ $\underline{a} = \langle a_1, a_2, \dots, a_d \rangle \rightarrow \text{constants}$

Differentiating a vector:

$\frac{df}{d\underline{x}} = \underline{\nabla}_{\underline{x}} f$ $\xrightarrow{\text{grad}(\cdot) \text{ Del}}$ $\left(\frac{df}{d\underline{x}} \right)$

\uparrow
 Vector

$\underline{\nabla}_{\underline{x}} f = \begin{bmatrix} \frac{df}{dx_1} \\ \frac{df}{dx_2} \\ \vdots \\ \frac{df}{dx_d} \end{bmatrix}$ $\xrightarrow{\text{Vector} \in \mathbb{R}^d}$

$\frac{df}{dx_1} = \left(\frac{\partial f}{\partial x_1} \right)$ \rightarrow partial diff'n

Differentiating a function f , which is a vector:

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

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

$$\frac{\partial f}{\partial x_1} = a_1$$

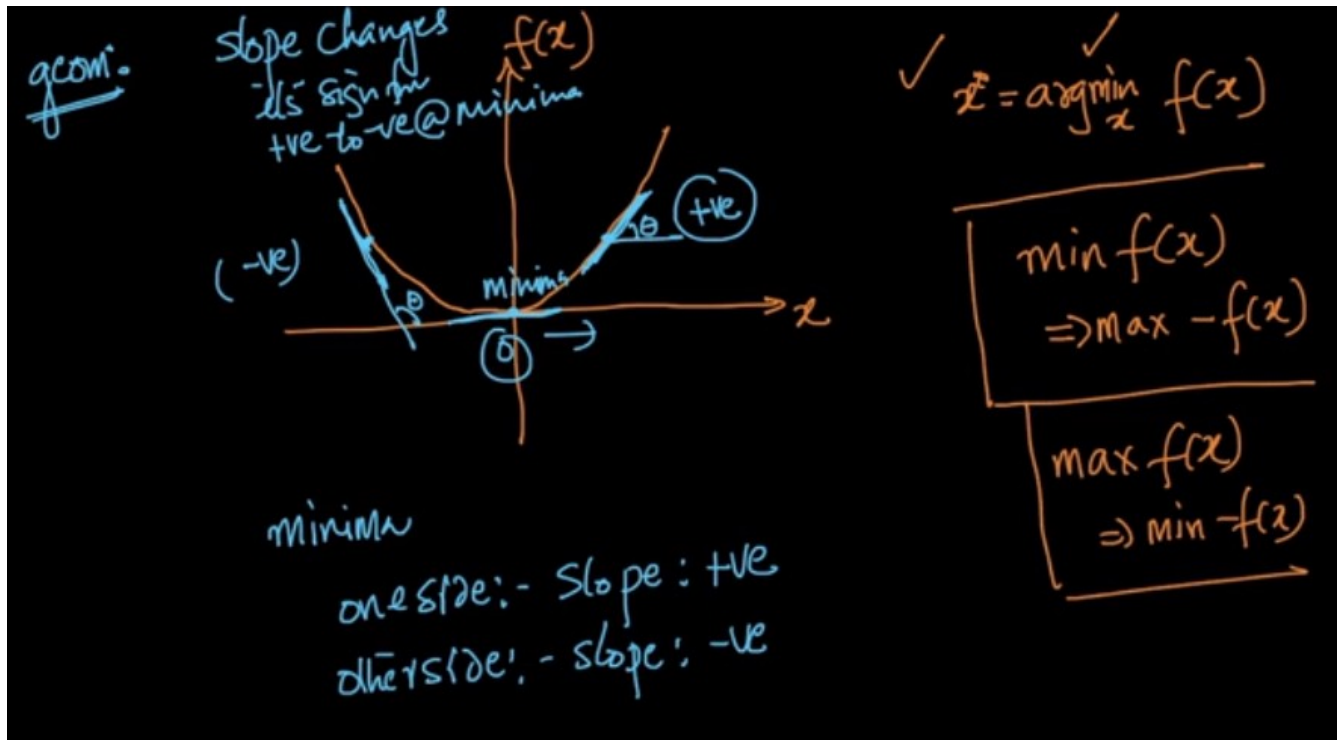
$$\frac{\partial f}{\partial x_2} = a_2$$

$$\nabla_x (a^T x) = \boxed{a}$$

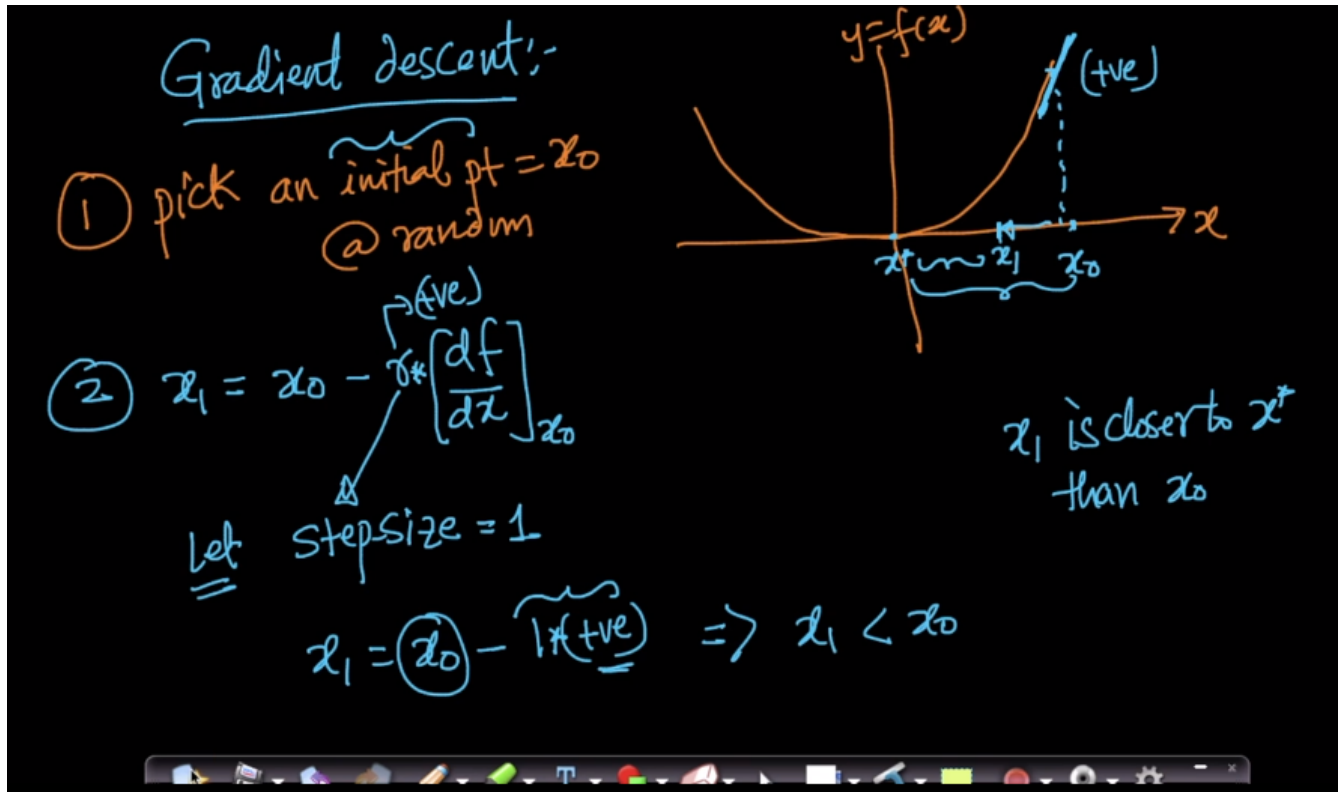
$$\frac{d(a^T x)}{dx} = a$$

We use gradient descent for calculating the minimum of the function, because as the terms increase the functions become complex.

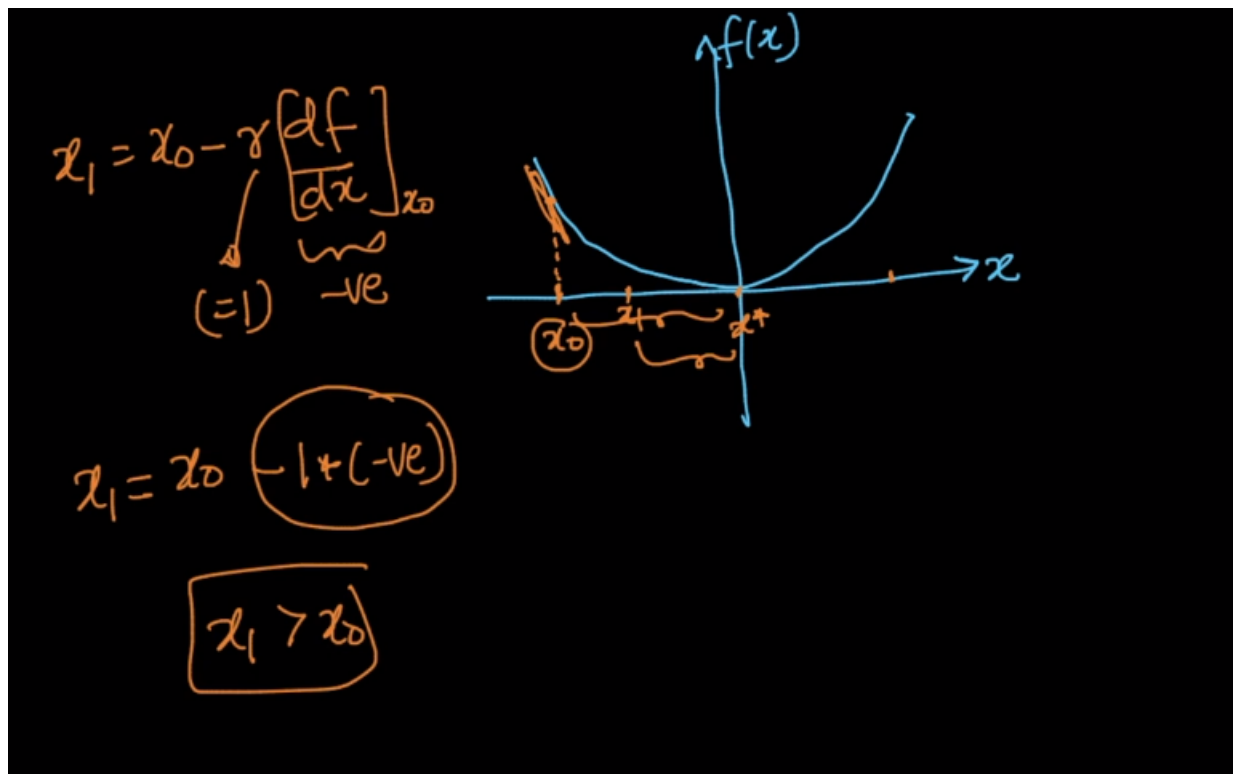
Gradient descent algorithm: It is an iterative algorithm. As the iterations increase the minimum value is achieved.



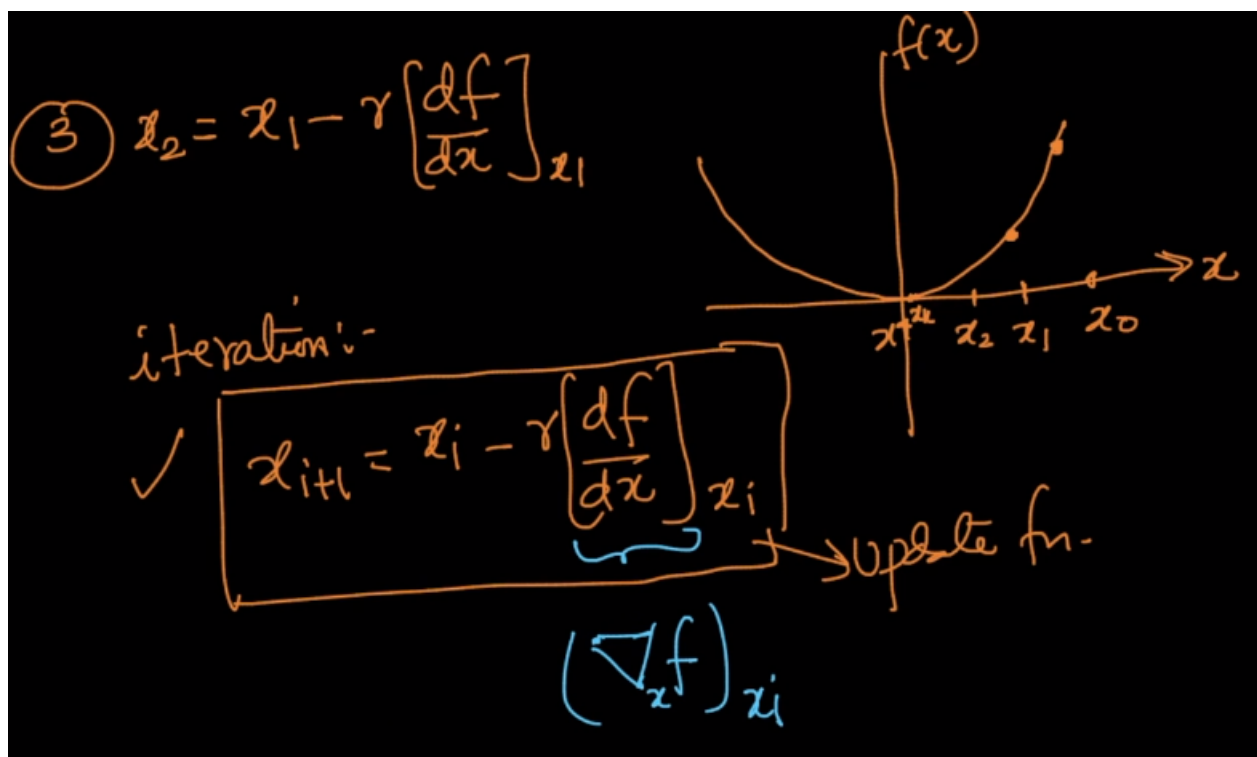
For positive slope.



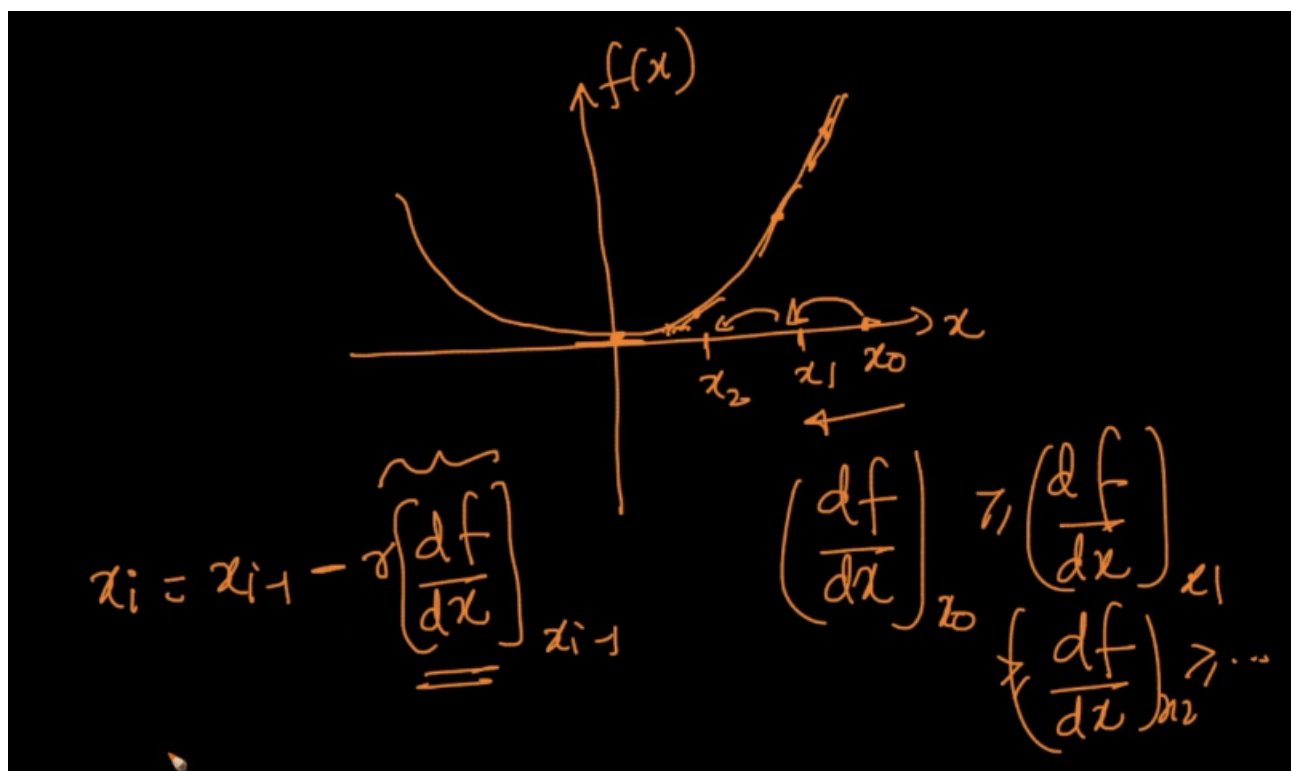
For negative slope.



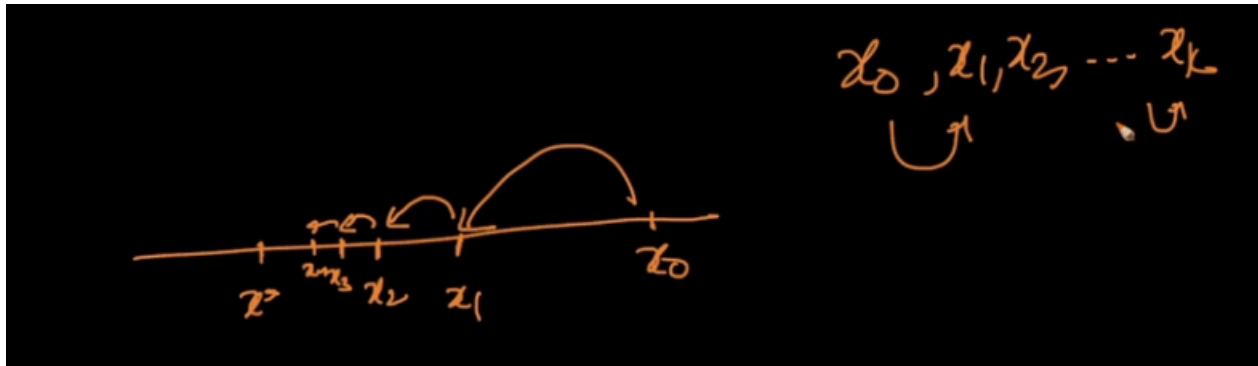
Another point on the curve is achieved by updating the step size by derivative at the previous point.



Property of Gradient descent:



The first step will be big, the next steps will be decreased eventually to attain the minimum of the function, that is achieved when the derivative is relatively small.



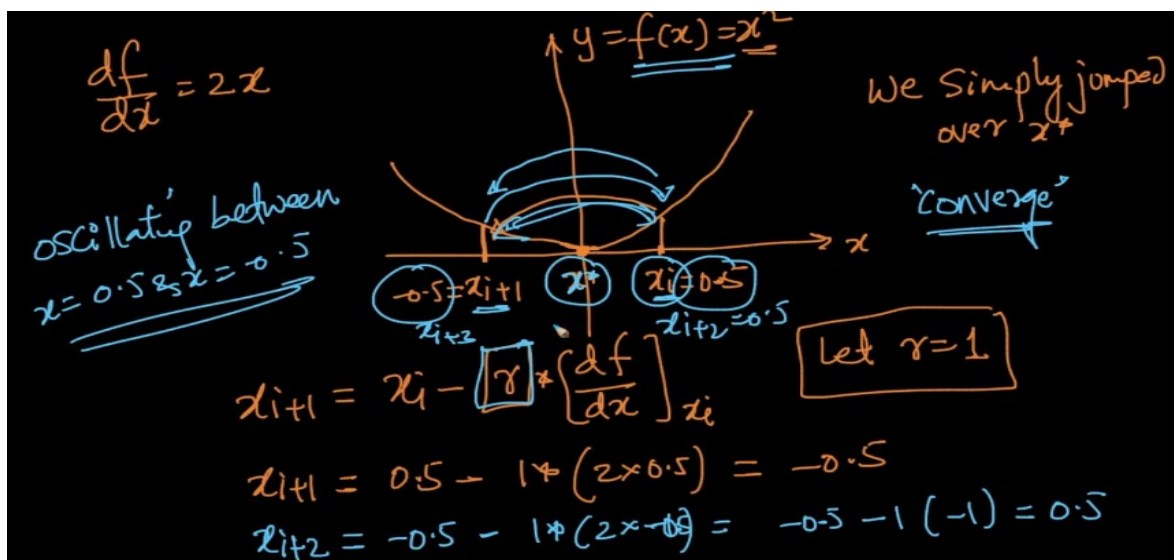
Learning rate (or) step-size:

'r' is called the learning rate.

Learning rate (or) Step-size

$$x_i = x_{i-1} - \underbrace{\gamma}_{\text{constant}} \left[\frac{df}{dx} \right]_{x_{i-1}} \rightarrow \text{update eqn}$$

If learning rate is not appropriate then the lowest point is 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 γ with each iteration
 → one tech:- reduce γ with each iteration
 (γ = function of iteration number)
 $\gamma = h(i)$ _{iteration} s.t $i \uparrow ; \gamma \downarrow$

Gradient descent for linear regression:

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

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

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

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

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

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:

Gradient descent for Linear Regression:-

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^n (y_i - w^T x_i)^2$$

Annotations in the image:
 - A bracket under the sum indicates "no-reg".
 - An arrow points from the text "not using w_0 " to the w in the argmin term.

The loss function is written as the function of L.

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

$\langle x_i, y_i \rangle \rightarrow D_{\text{train}}$

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

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

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

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

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

Handwritten notes on a blackboard:

$$w_0, w_1, w_2, \dots, \underline{w_k}, w_{k+1}$$

$w_{k+1} - w_k$ is a ^{vector of} very small values

$$w^* = w_k \checkmark$$

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

Handwritten notes on a blackboard:

Gradient desc

$$w_j = w_{j-1} - \gamma \sum_{i=1}^n (-2x_i)(y_i - w_j^T x_i)$$

problem: w_{j-1} to w_j

w_0, w_1, w_2

if n is large
 $n = 1 \text{ Million}$

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.

✓ Stochastic Gradient descent (SGD) → the most imp opt.

Linear regression:- $n = 1 \text{ Million}$

GD: $w_{j+1} = w_j - \gamma \sum_{i=1}^n (-2x_i)(y_i - w_j^T x_i)$

SGD: $w_{j+1} = w_j - \gamma \sum_{i=1}^K (-2x_i)(y_i - w_j^T x_i)$

DL: Adam, Adagrad

$w_{GD}^* = w_{SGD}^*$

$1 \leq K \leq n$: pick a random set of K -pls

When n is large, takes a lot of time

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

K : batch size in SGD

↓

batch of random pls.

often times: $K=1$: SGD ✓

$n < K > 1 \rightarrow$ batch SGD with batch size = 100

Mill

GD: $(n=1000)$ 100 iterations to converge x^+

SGD:- ~~K~~ = 1000; if $\frac{7100}{1000}$ iterations x^p

SGD: ~~K~~ = 10; (1000); 2^{Φ}

$K = \#$ of random pl's that you @ each iteration for update

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

Constrained optimization & PCA:

✓ Constrained Optimization

$$\max_x f(x) \quad \text{or} \quad \min_x f(x)$$

→ objective fn.

PCA:

$$\max_{\mathbf{u}} \frac{1}{n} \sum_{i=1}^n (\mathbf{u}^T \mathbf{x}_i)^2 \quad \approx \quad \max_{\mathbf{x}} f(\mathbf{x})$$

$$\text{s.t. } g(\mathbf{x}) = c$$

✓ s.t. $u^T u = 1$

Constraint -

Here the constraint is U is a unit vector.

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

Lagrangian Multipliers

$$\begin{aligned} \max_x & f(x) \\ \text{s.t. } & g(x) = c \\ & h(x) \geq d \end{aligned}$$

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \lambda \{g(x) - c\} + \mu \{d - h(x)\}$$

λ & μ : Lagrangian Mult
 $\lambda \geq 0$; $\mu \geq 0$

$$\frac{\partial \mathcal{L}}{\partial x} = 0; \frac{\partial \mathcal{L}}{\partial \lambda} = 0; \frac{\partial \mathcal{L}}{\partial \mu} = 0$$

$x^* = \tilde{x}$

$\tilde{x}, \tilde{\lambda}, \tilde{\mu}$

PCA:

Modification of PCA.

PCA:

$$\max_u \frac{1}{n} \sum_{i=1}^n (u^T x_i)^2$$

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

$$X = \begin{bmatrix} \leftarrow x_i^T \rightarrow \end{bmatrix}$$

$$\max_u u^T S u$$

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

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

Covariance matrix of \tilde{X}

PCA:- $\max_u u^T S u$
s.t. $u^T u = 1$

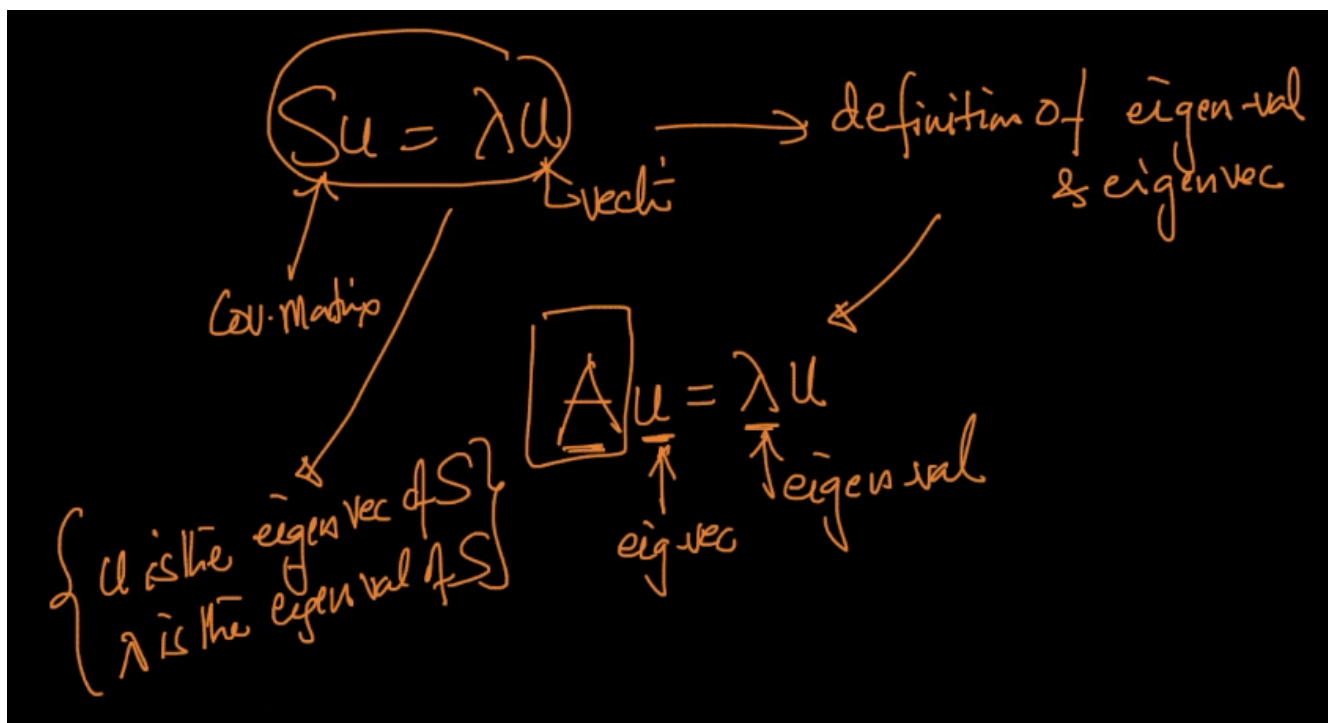
$$S = \frac{1}{n} \sum_{i=1}^n \underline{x_i} \underline{x_i}^T$$

→ Co-var matrix of X

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

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

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



PCA:- $\max_u u^T S u$ s.t. $u^T u = 1$ } ✓

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

Co-var matrix of X

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

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

$$\Rightarrow S u = \lambda u$$

Logistic regression formulation revisited:

Logistic regression revisited

$$w^* = \arg \min_w (\text{logistic loss}) + \lambda w^T w$$

Const. optimization

$$\left\{ \begin{array}{l} \underline{w}^* = \arg \min_w (\text{logistic loss}) \\ \text{s.t. } w^T w = 1 \end{array} \right.$$

obj. fn.

eq. const.

Lagrange multipliers

λ

$w^T w = 1$

π

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

Lagrangian

$$L = \text{logistic loss} - \lambda (1 - w^T w)$$

$(-w)(w^T w - 1)$

$= \text{logistic loss} - \lambda + \underline{\underline{w^T w}}$

regularization can be thought as as ^{imposing} an eq. constr.

Why L1 regularization creates sparsity?

(L_2) $w = \langle w_1, w_2, \dots, w_d \rangle$

$\min_w \text{loss} + \lambda \|w\|_2^2$

$\min_{w_1, w_2, \dots, w_d} (w_1^2 + w_2^2 + \dots + w_d^2)$

$\min_{w_1} w_1^2 = L_2(w_1)$

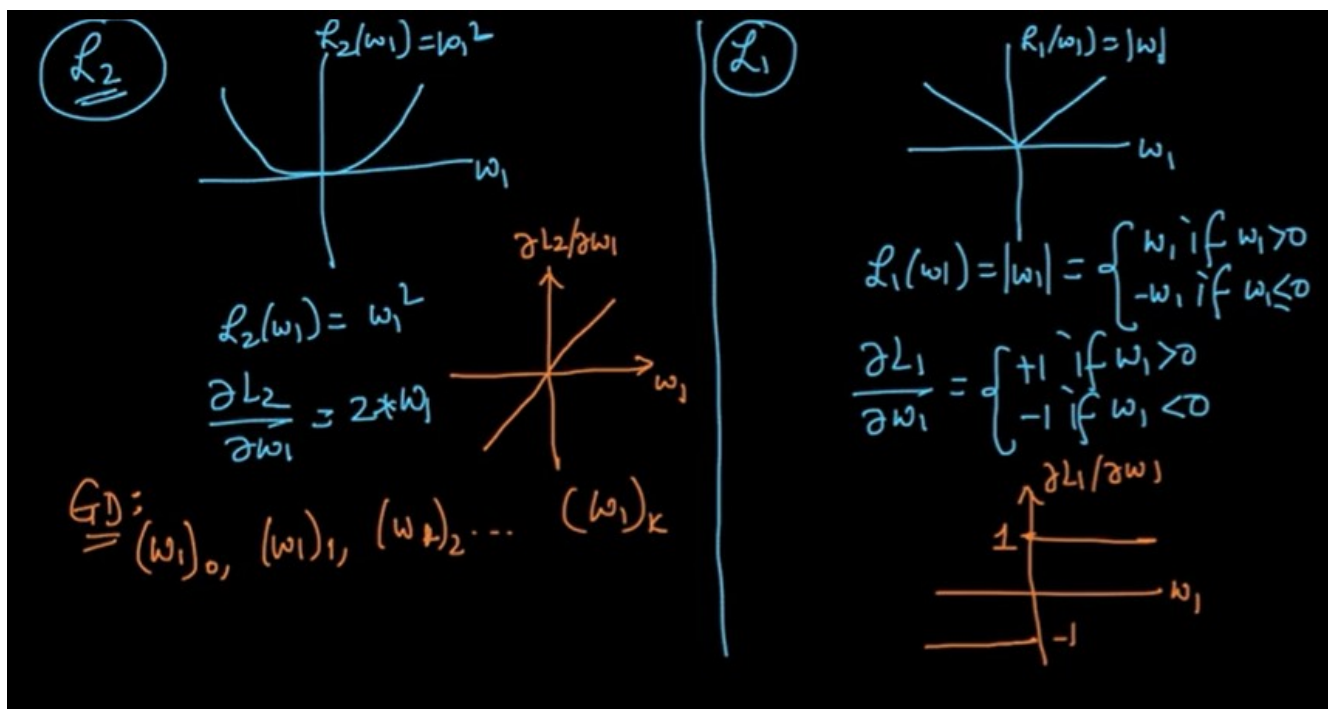
(L_1)

$\min_w \text{loss} + \lambda \|w\|_1$

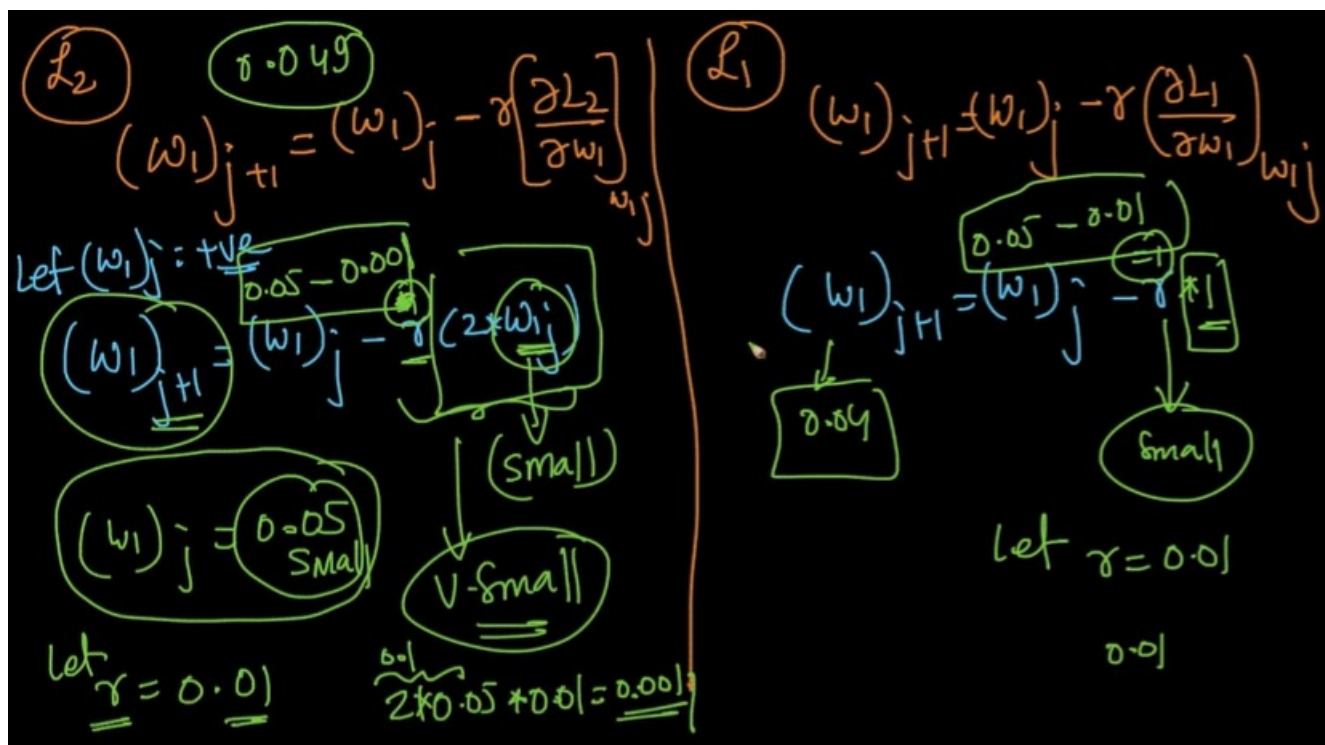
$\min_{w_1, w_2, \dots, w_d} (|w_1| + |w_2| + \dots + |w_d|)$

$\min_{w_1} (|w_1|) \rightarrow L_1(w_1)$

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



The rate of convergence will decrease in L2.



L2 regularizer doesn't change the value of w_1 from one iteration to another iteration, because the gradient reduce towards $w_1^* = 0$.

v. quickly:- L_2 reg. doesn't change the value of w_1 from one iteration to another
 L_1 reg. continues to constantly reduce w_1 towards $w_1^* (=0)$

Exercise:

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

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

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

(100 iterations)

w^*

$\underbrace{w_k}_{\approx w_{k+1}}$ $\underbrace{b_k}_{\approx b_{k+1}}$

k -iterations

reduce γ with each iteration.

$$\gamma_0 = 1$$

$$\left\{ \begin{array}{l} \text{iter } 0 \rightarrow \gamma = 1 \\ 1 \rightarrow \gamma = \gamma/2 \\ 2 \rightarrow \gamma = \gamma/2 \\ \vdots \end{array} \right\} \checkmark$$

Ex: Implement SGD for linear regression

Lr. regn: - code-sample (boston home price dataset)

LinearRegression() in Sklearn

* Implement SGD for Lr. regn