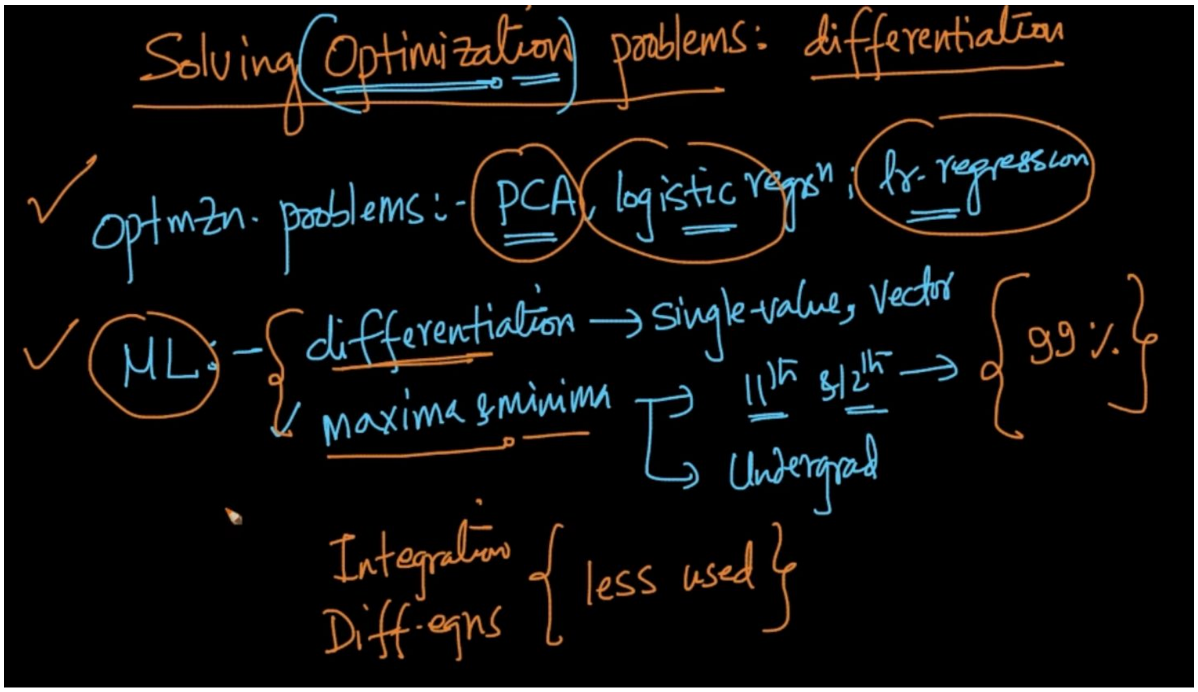
SOLVING OPTIMIZATION PROBLEMS

DIFFERENTIATION



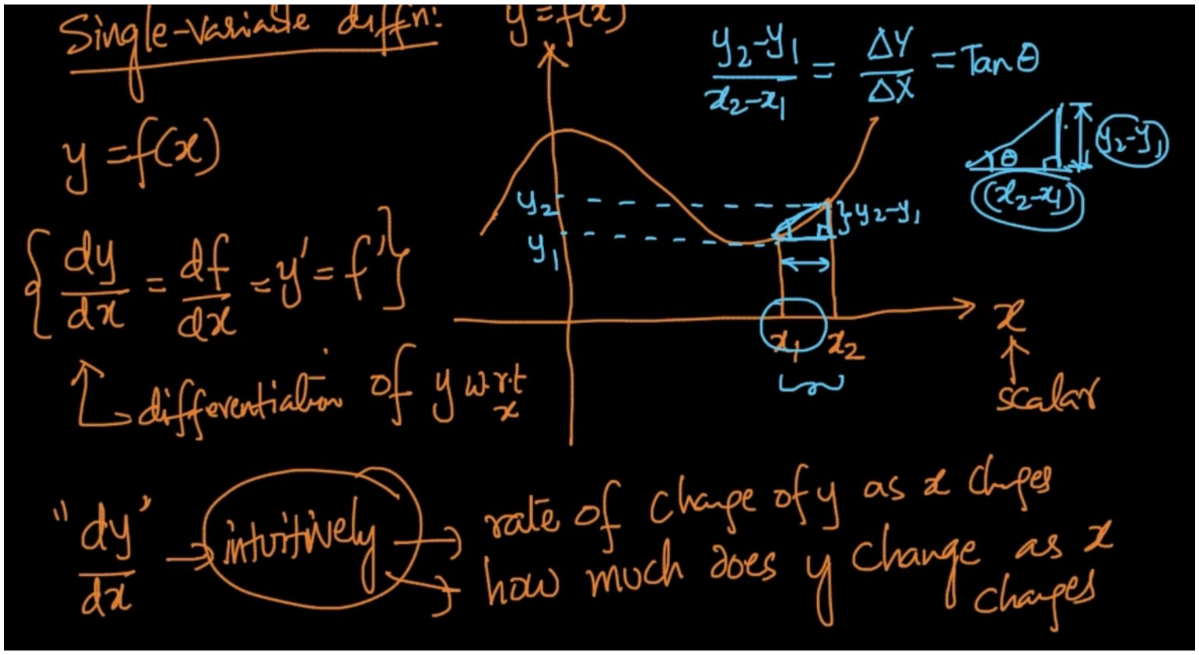
Solving (Optimization) problems: differentiation

✓ optmzn. problems:- PCA, logistic regsn, lr. regression

✓ ML:- { differentiation → single-value, vector; maxima & minima } → 11th & 12th → { 99% } ; → Undergrad

Integration, Diff-eqns { less used }

The idea of Differentiation and Maxima,Minima is used in 99% of Machine Learning



Single-variate diffn: $y=f(x)$

$y=f(x)$

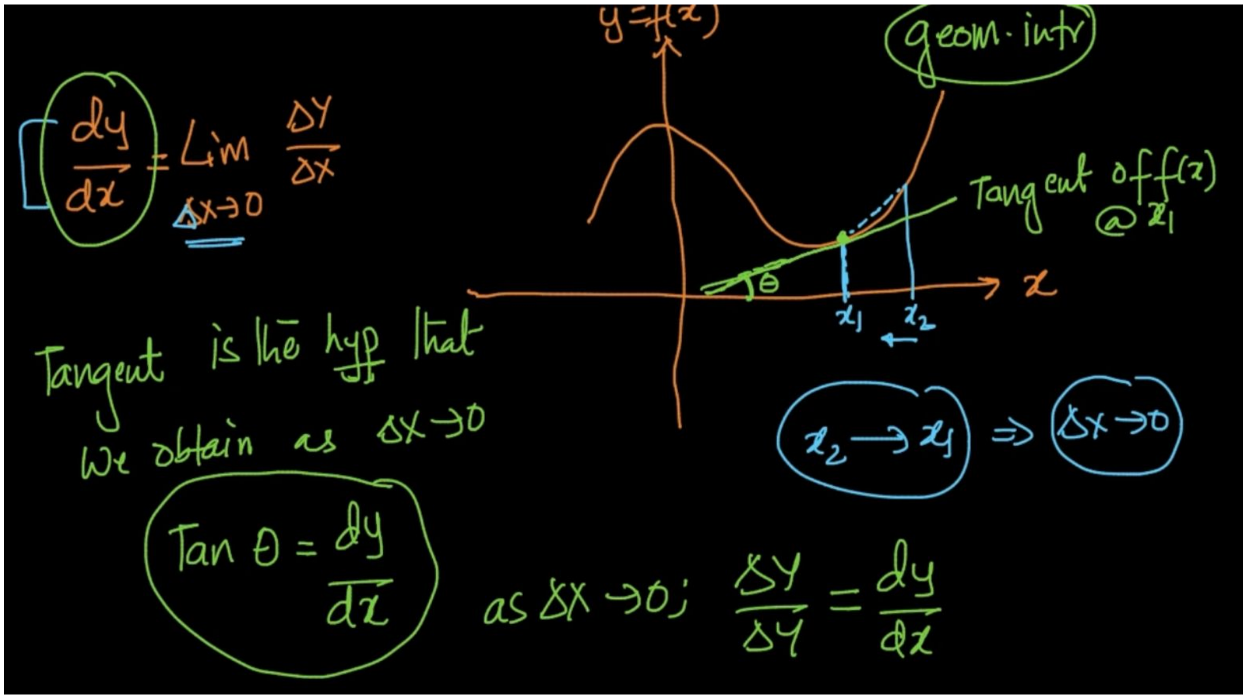
$\left\{\frac{dy}{dx} = \frac{df}{dx} = y' = f'\right\}$

↳ differentiation of y w.r.t x

$\frac{dy}{dx}$ → intuitively → rate of change of y as x changes → how much does y change as x changes

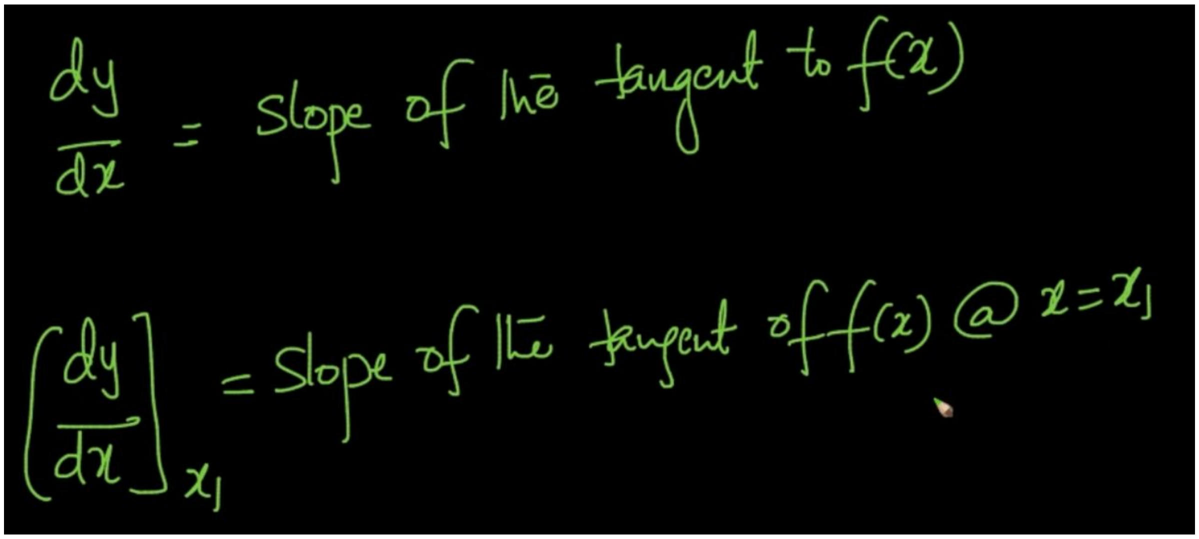
$\frac{y_2-y_1}{x_2-x_1} = \frac{\Delta y}{\Delta x} = \tan\theta$

dy/dx - Rate of change of y as x changes. dy/dx = $\frac{y_2-y_1}{x_2-x_1}$ = $\frac{\Delta y}{\Delta x}$ = $Tan\theta$


$y=f(x)$
geom. intr
$\left[\frac{dy}{dx}\right] = \lim_{\Delta x \to 0} \frac{\Delta Y}{\Delta X}$
Tangent of $f(x)$ @ x_1
θ
x
x_1
x_2
Tangent is the hyp that we obtain as $\Delta X \to 0$
$x_2 \to x_1 \Rightarrow \Delta x \to 0$
$\tan\theta = \frac{dy}{dx}$
as $\Delta X \to 0$; $\frac{\Delta Y}{\Delta Y} = \frac{dy}{dx}$

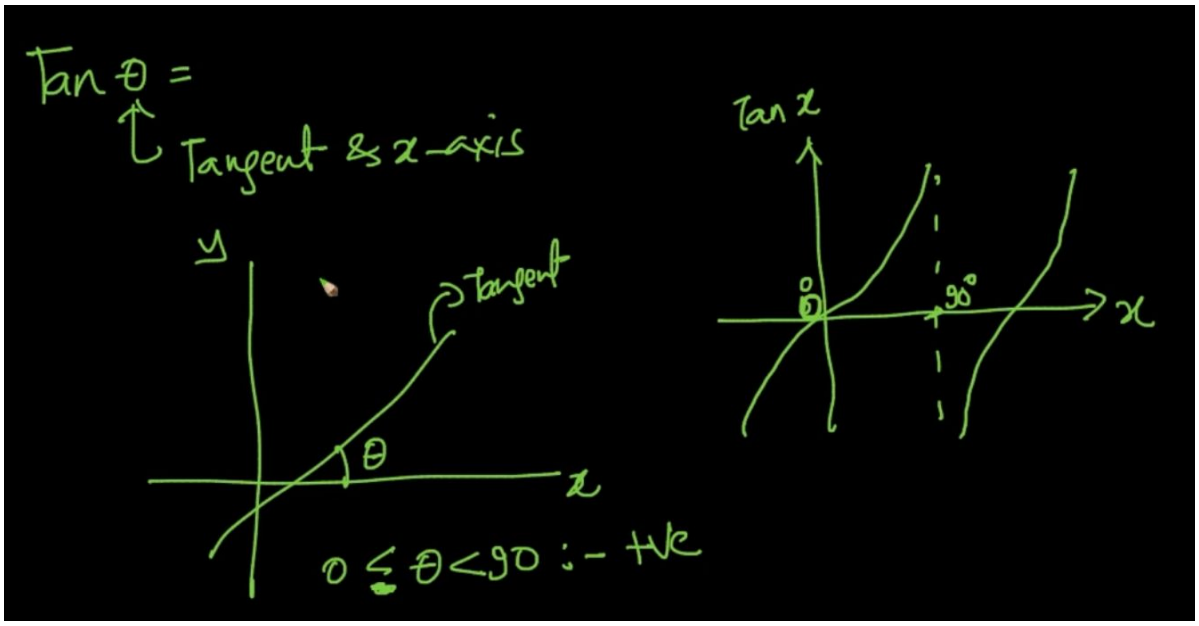
dy/dx = $\lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$ .Let's break down it. The derivative of y w.r.t x (dy/dx) is equal to rate of change of y w.r.t rate of change of x i.e ($\frac{\Delta y}{\Delta x}$) and Δx i.e x_2 - x_1 is close to 0.

If Δx is 0 then the tangent of f(x) at that point x is our (dy/dx) and it's θ is angle between slope and x-axis

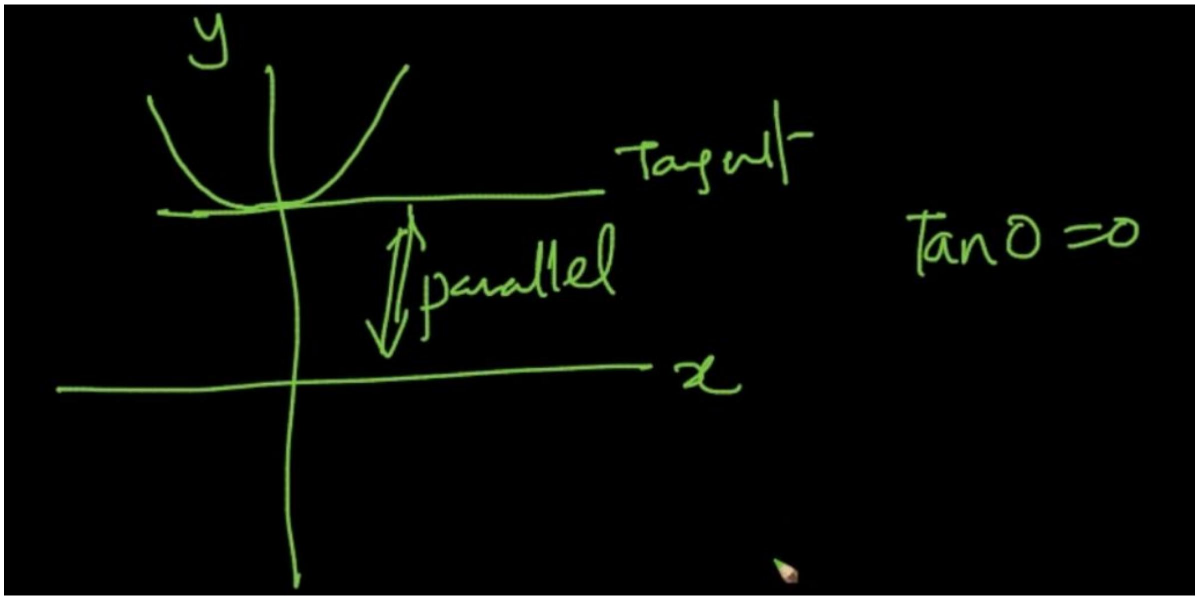


$\frac{dy}{dx}$ = Slope of the tangent to $f(x)$

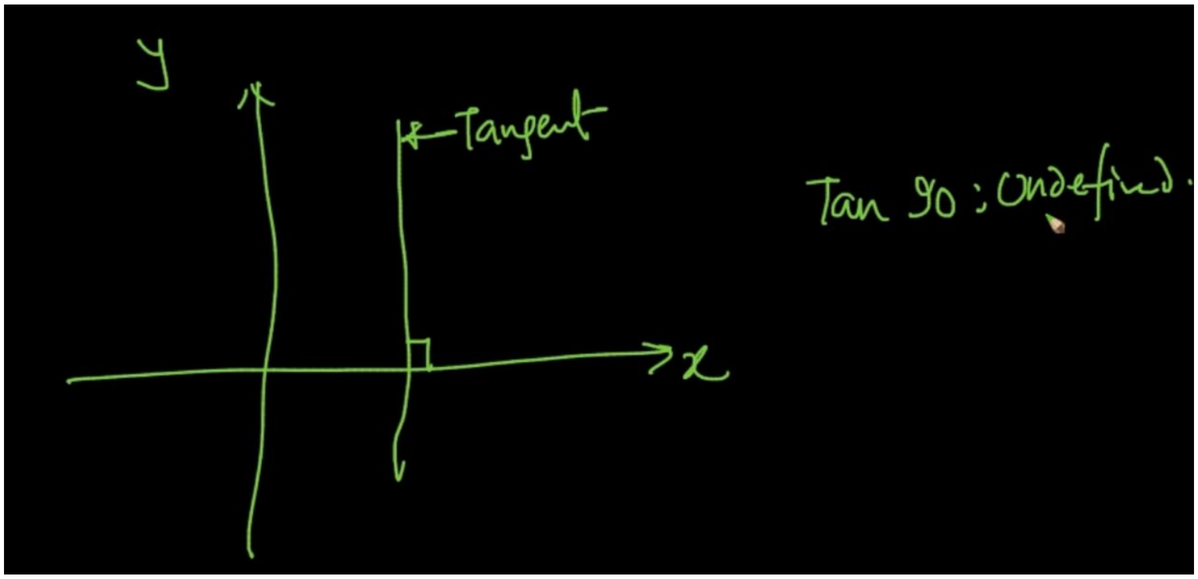
$\left[\frac{dy}{dx}\right]_{x_1}$ = Slope of the tangent of $f(x)$ @ $x = x_1$



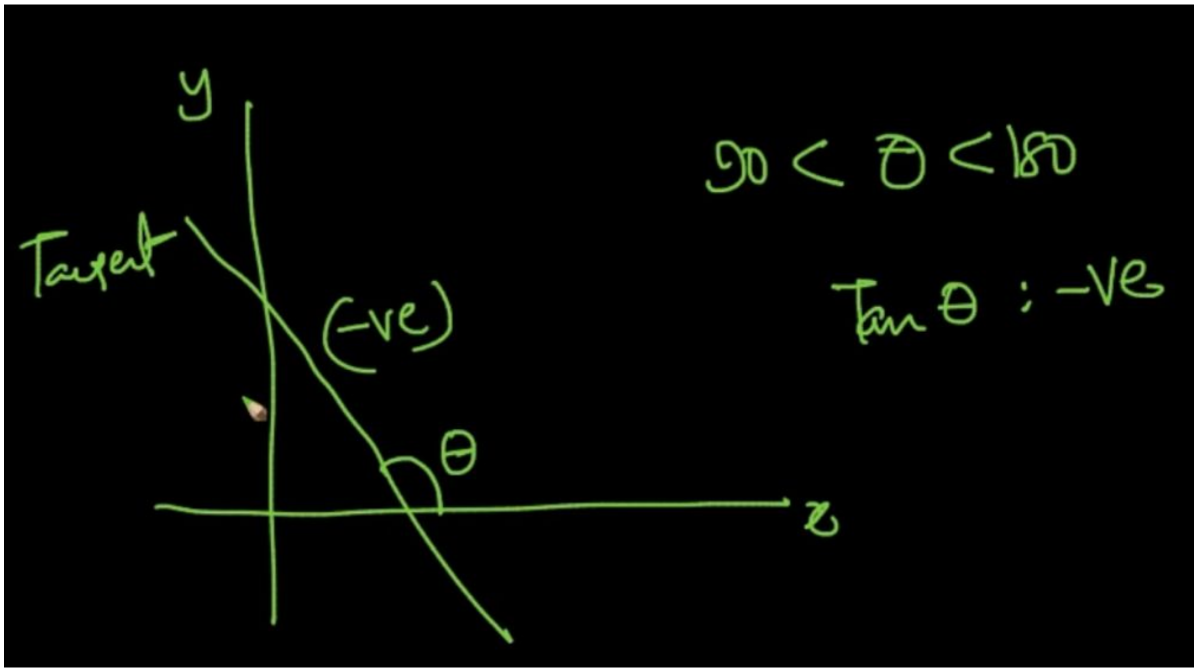
Tangent when 0 < θ < 90 is positive



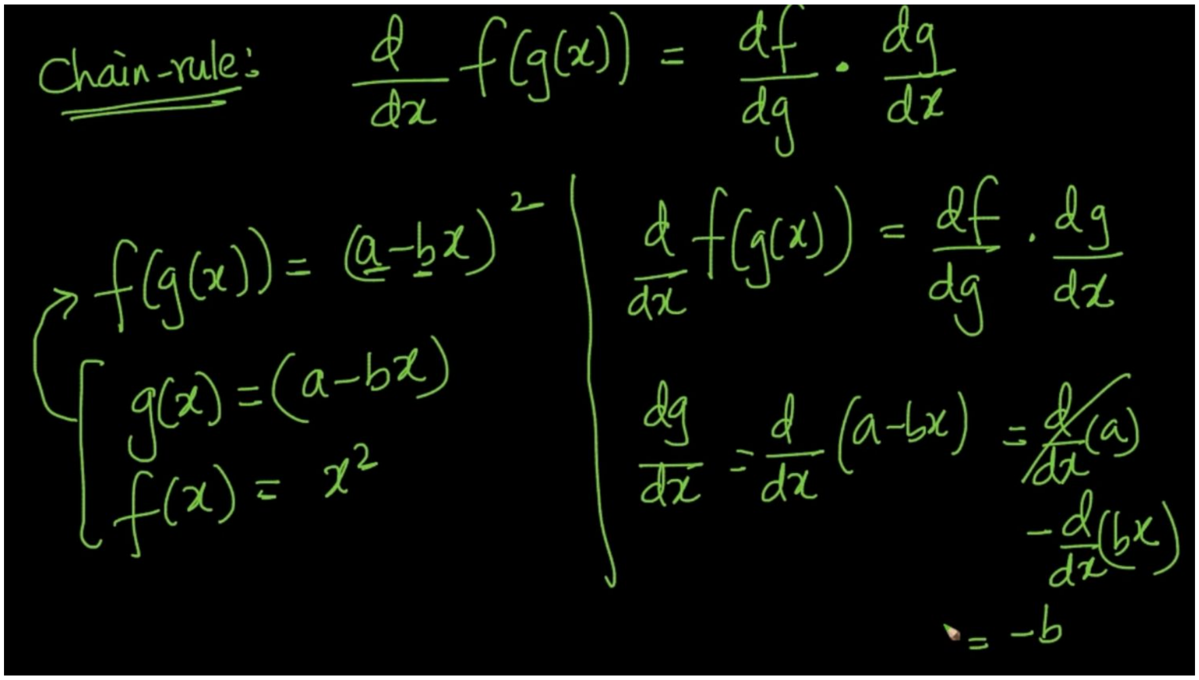
When θ = 0 tangent is parallel to x-axis



Tan 90 is undefined



Tan θ is negative when 90 < θ < 180



Chain-rule: $\frac{d}{dx} f(g(x)) = \frac{df}{dg} \cdot \frac{dg}{dx}$

$f(g(x)) = (a-bx)^2$

$g(x) = (a-bx)$

$f(x) = x^2$

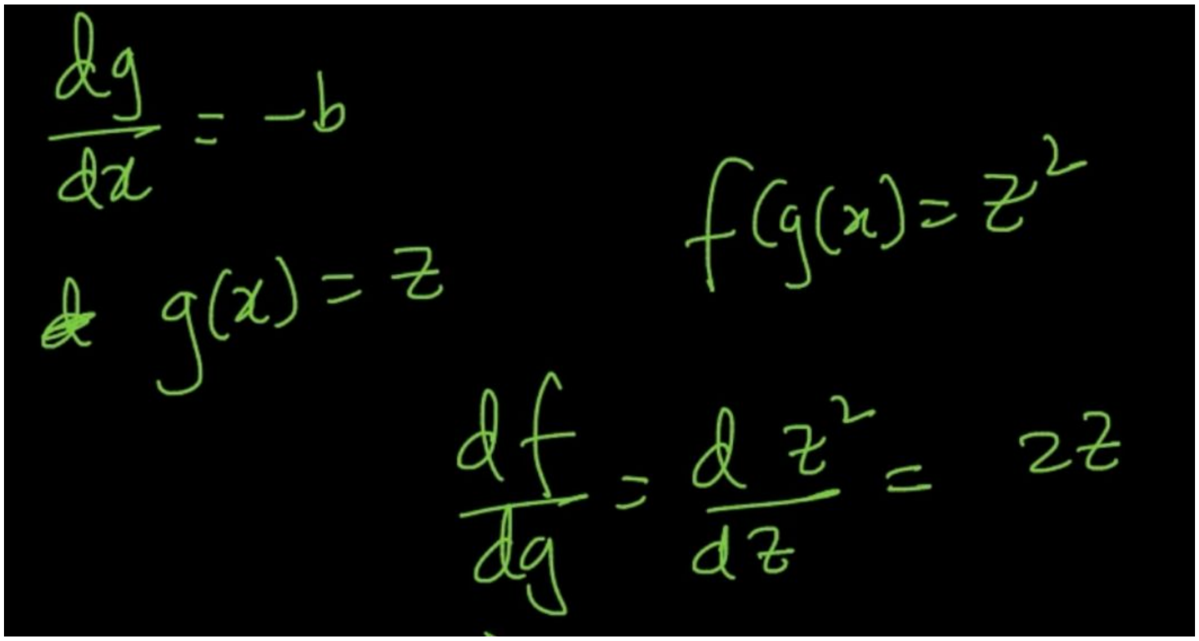
$\frac{d}{dx} f(g(x)) = \frac{df}{dg} \cdot \frac{dg}{dx}$

$\frac{dg}{dx} = \frac{d}{dx}(a-bx) = \cancel{\frac{d}{dx}(a)} - \frac{d}{dx}(bx)$

$= -b$

It's the most important concept of Calculus in Deep Learning and Machine Learning.

$\frac{d}{dx}$ (f(g(x)) = $\frac{df}{dg} \cdot \frac{dg}{dx}$. First, we'll calculate $\frac{dg}{dx}$ .Now, calculate df/dg.



$\frac{dg}{dx} = -b$

& $g(x) = z$

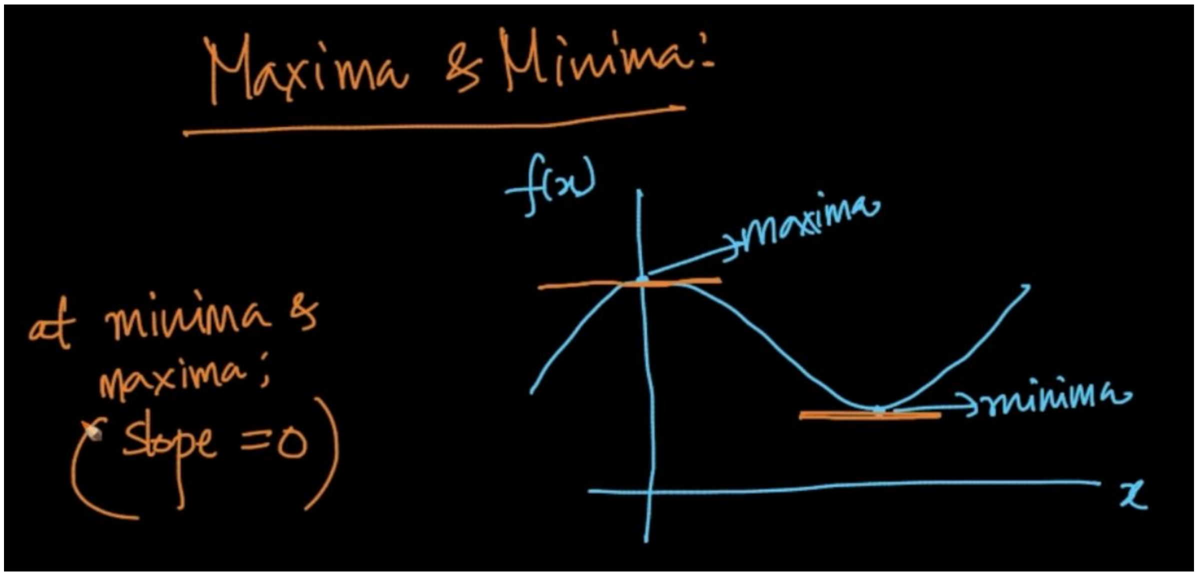
$f(g(x) = z^2$

$\frac{df}{dg} = \frac{d\,z^2}{dz} = 2z$

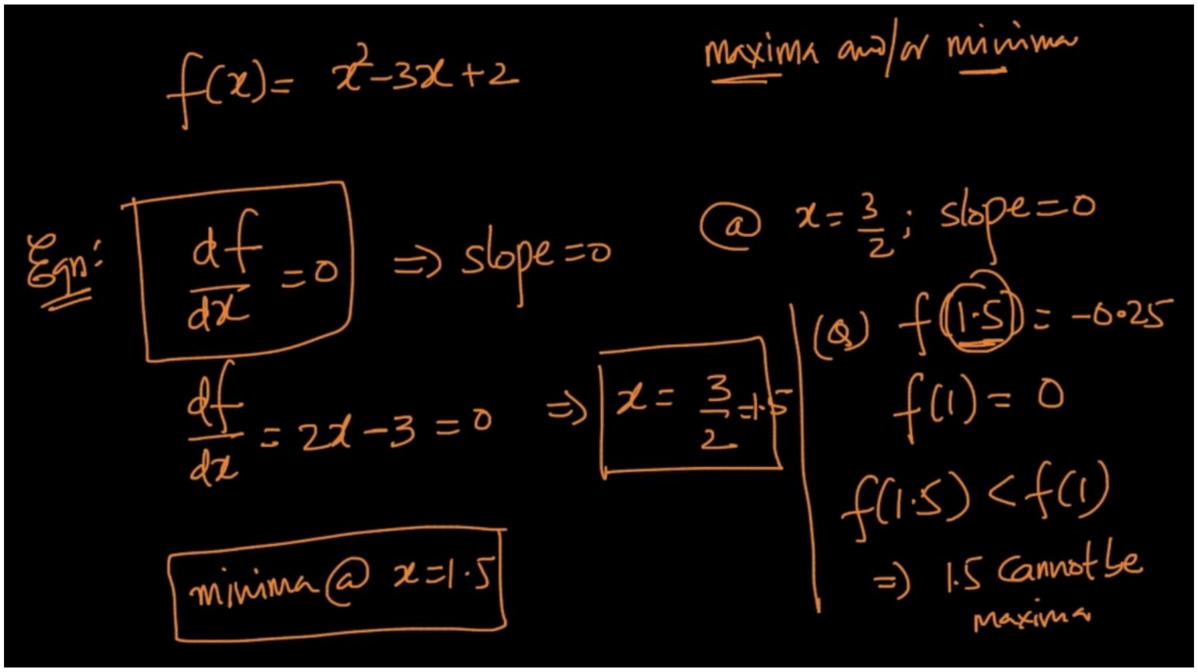
If we take g(x) = z then f(g(x)) = z^2 for the above f(g(x)) so df/dg = 2z

So $\frac{df}{dg} \cdot \frac{dg}{dx}$ = 2.(a-bx). -b for the above problem

MAXIMA AND MINIMA



At minima and maxima the slope is = 0



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

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

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

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

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

(Q) $f(1.5) = -0.25$

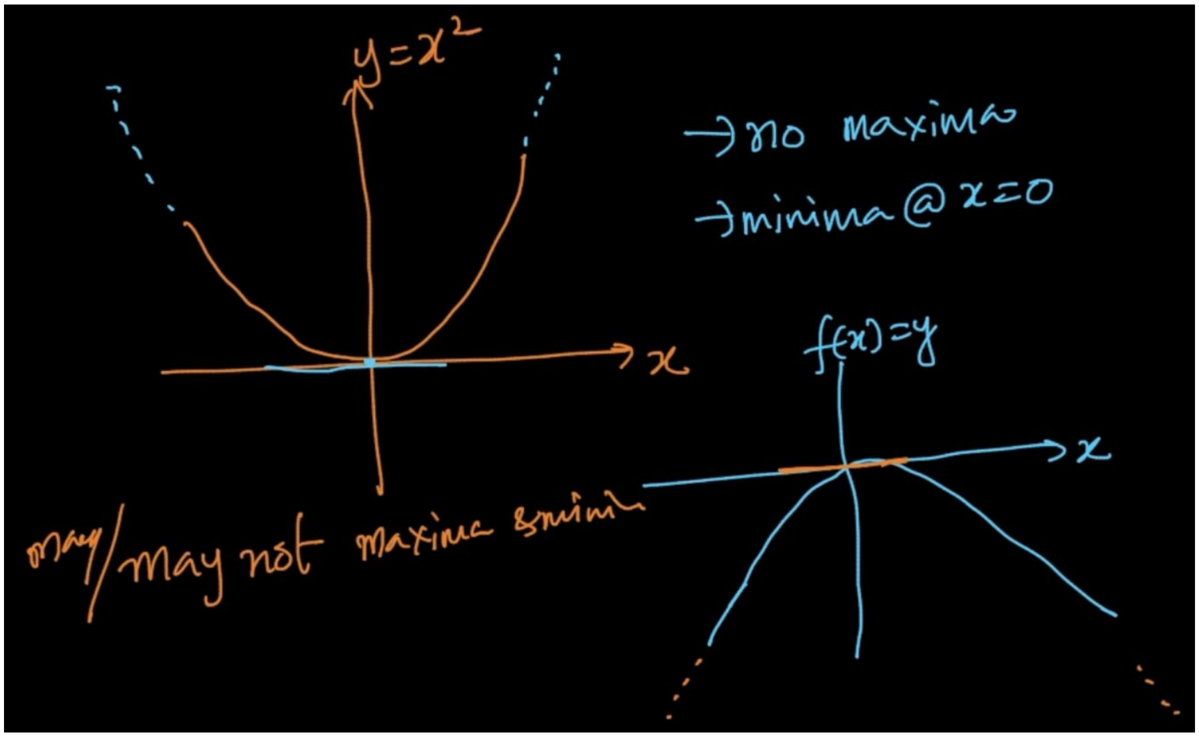
$f(1) = 0$

$f(1.5) < f(1)$

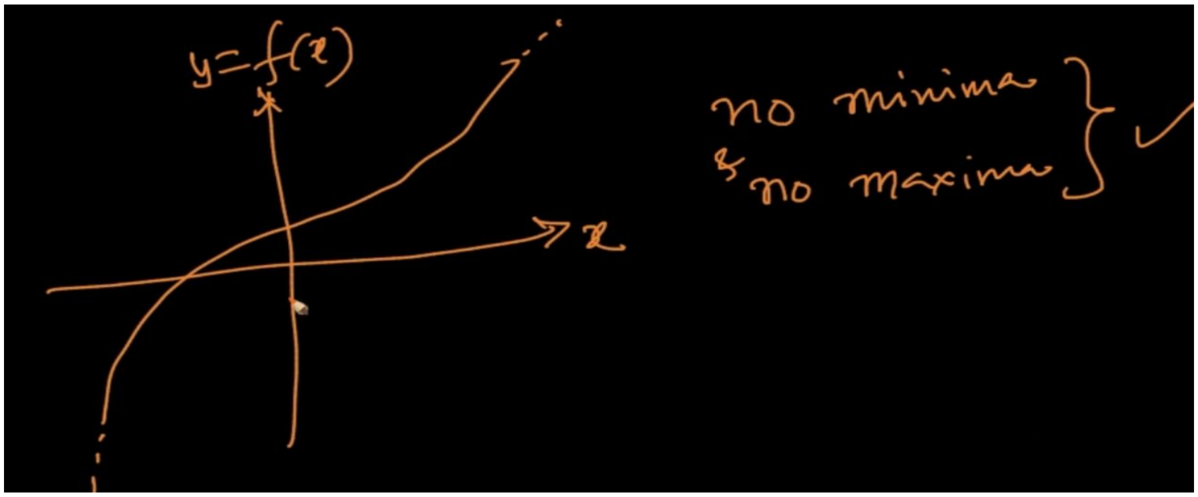
$\Rightarrow$ 1.5 cannot be maxima

We know that df/dx is our slope so if we make it = 0 then we can get the value of maxima/minima. We are calculating df/dx = 0 and getting the value of x. So, f(x) would tell

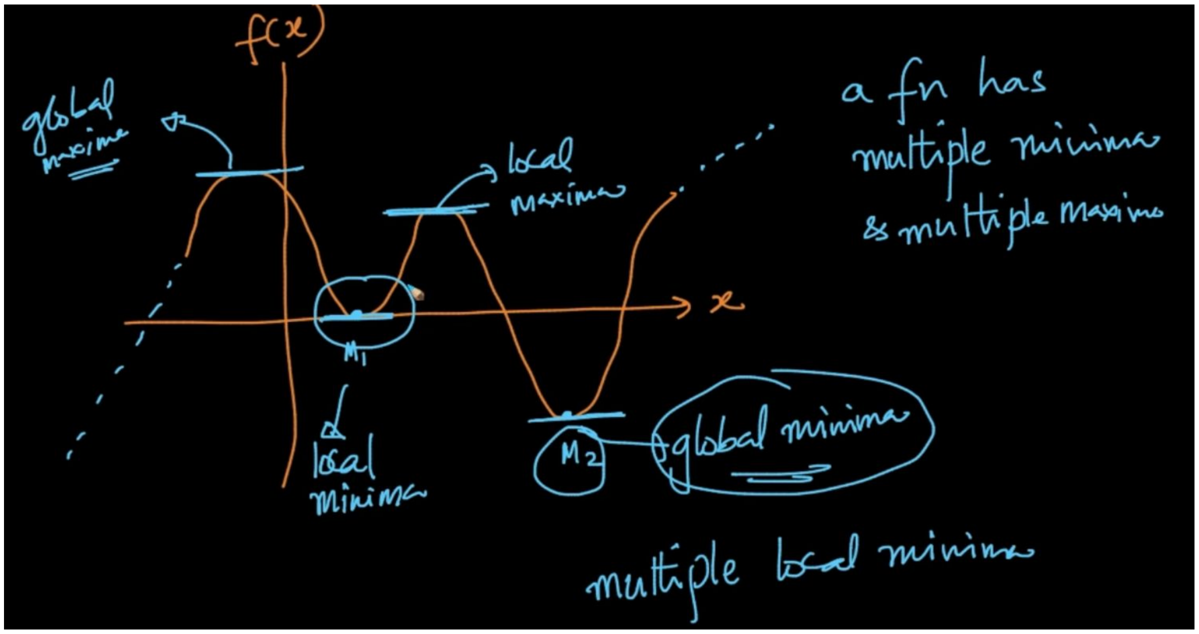
us the minima/maxima but we don't know whether it's telling maxima or minima .For this we calculate f(x) for a near value towards it. Here, df/dx = 0 at x = 1.5 and f(1.5) = -0.25
We take f(1) which is 0 , Since f(1.5) < f(1) it cannot be maxima so it'll be minima



There can be some functions which might not have maxima/minima . Ex: In the above orange figure there's only minima not maxima and vice-versa for the blue figure



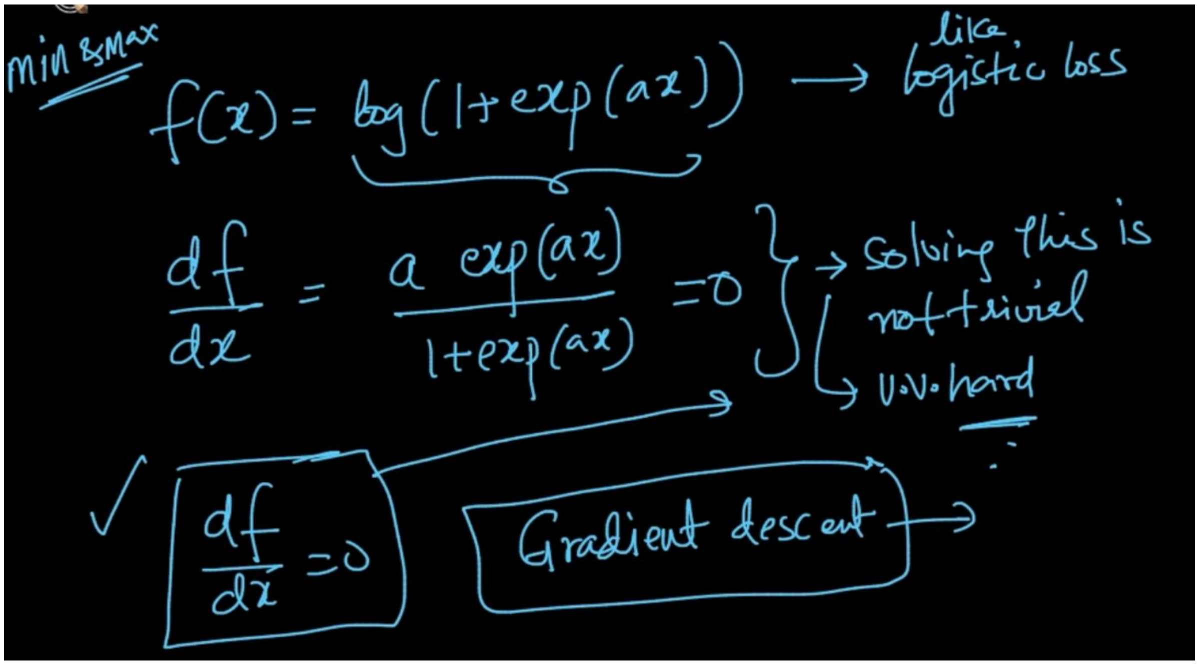
There can be some functions which might not have maxima/ minima



A function can have multiple maxima and minima as seen above.

Local Minima/ Maxima - It's the Minimum /Maximum in the neighborhood as seen above

Global Maxima/Minima - It's the Minimum/ Maximum among all the Minimums/ Maximums



min & Max

$$f(x) = \underbrace{\log(1+\exp(ax))} \longrightarrow \text{like, logistic loss}$$

$$\frac{df}{dx} = \frac{a\exp(ax)}{1+\exp(ax)} = 0$$

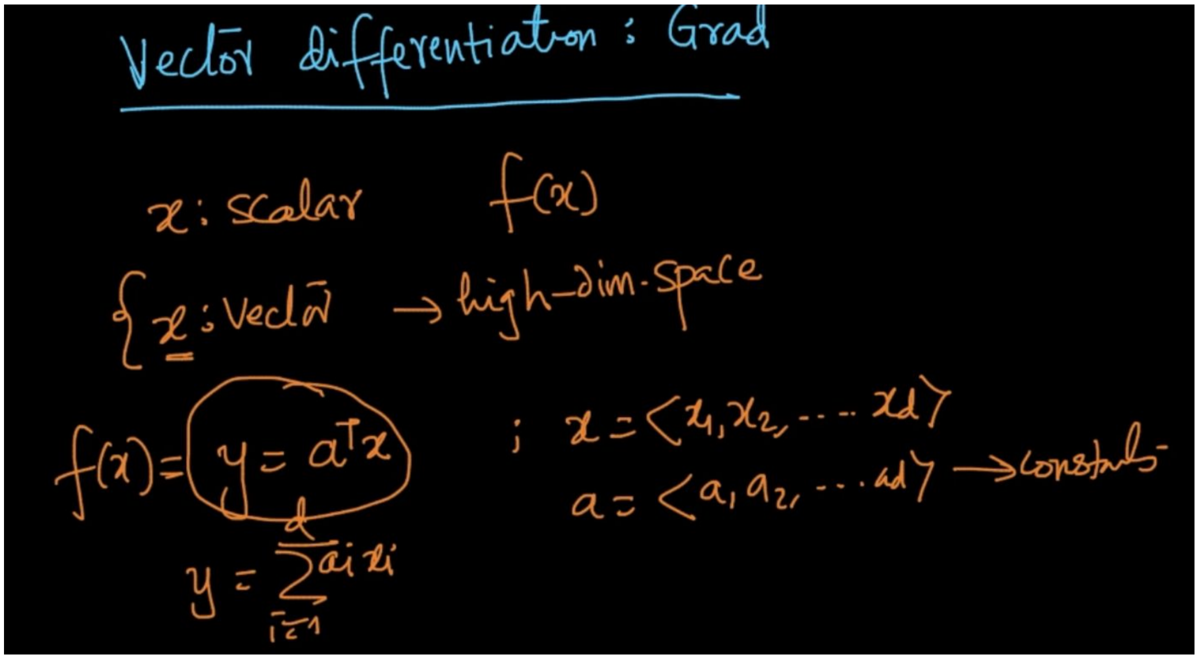
→ Solving this is not trivial

→ v.v. hard

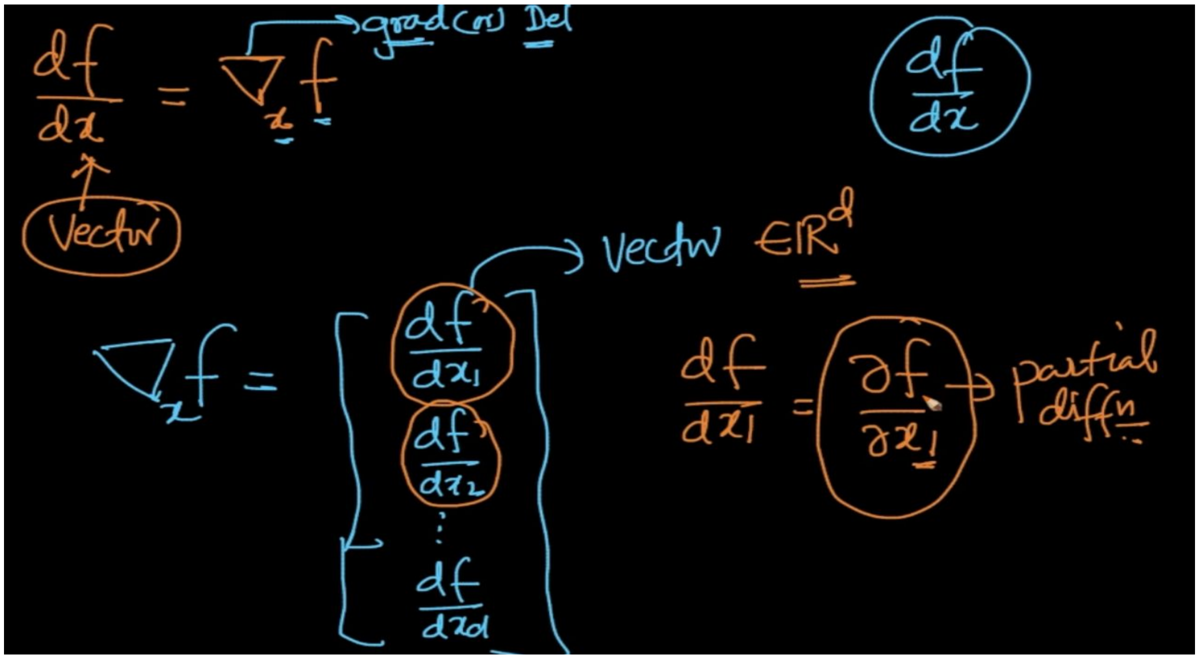
✓ $\boxed{\frac{df}{dx} = 0}$ → $\boxed{\text{Gradient descent}}$ →

Some functions cannot be solved by simple df/dx = 0. So, we'll use Gradient Descent for that

VECTOR CALCULUS GRAD

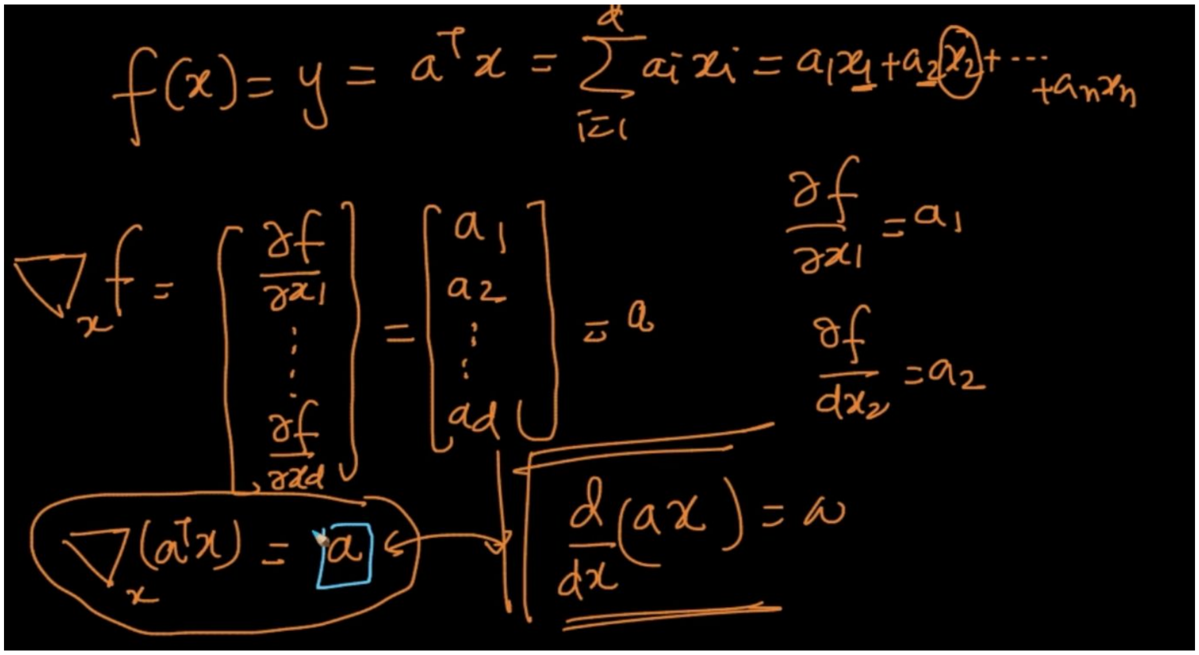

Vector differentiation : Grad
x : scalar f(x)
x : Vector → high-dim. space
f(x) = y = aᵀx ; x = <x1, x2, ... xd>
a = <a1, a2, ... ad> → constants
y = Σ_{i=1}^{d} ai xi

We’ve dealt with scalars in Differentiation but in ML things are in vector. So we’ll deal with vector calculus. Our , y = f(x) = $a^T x$ where x is vector and a is a vector of constants


df/dx = ∇ₓf → grad (or) Del
Vector
df/dx
→ Vector ∈ ℝ^d
∇ₓf = [df/dx1, df/dx2, ..., df/dxd]
df/dx1 = ∂f/∂x1 → partial diffn

df/dx = $\nabla_x f$ which is Del (∇) of f w.r.t x as seen. This is differentiation of vector

As seen above it's $\nabla_x f$ = [df/dx1, df/dx2…..df/dxd] but df/dx1 is partial derivation and we write it as $\delta f / \delta x1$. It means that x has <x1,x2,x3……xd> but we are derivating the function with only one component x1 at a time not the whole x



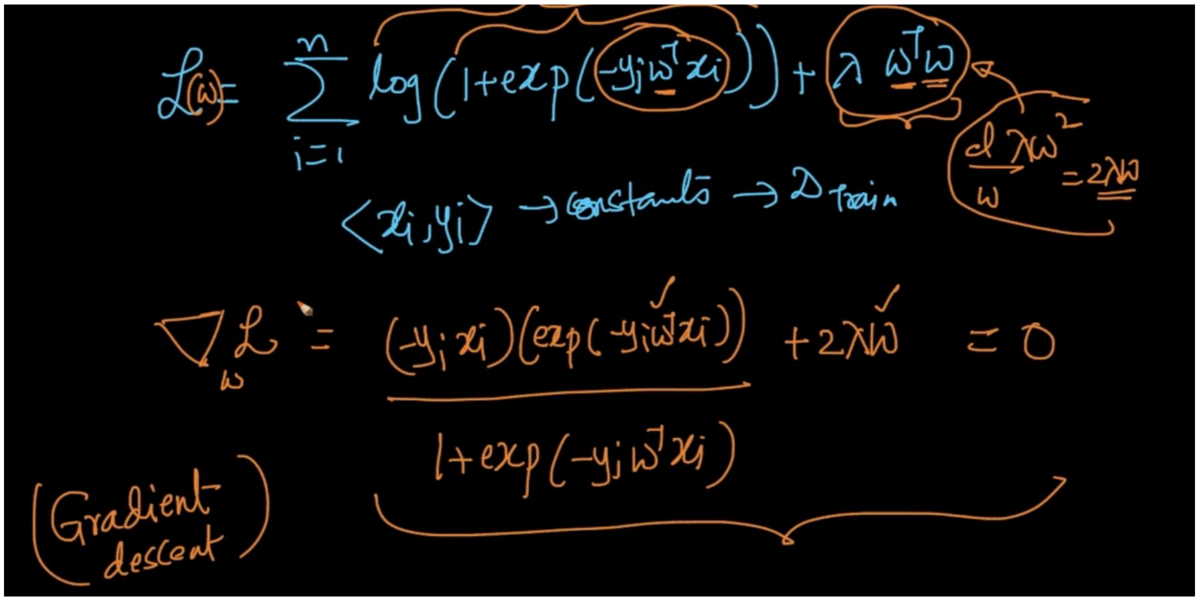
$$f(x) = y = a^T x = \sum_{i=1}^{d} a_i x_i = a_1 x_1 + a_2 \boxed{x_2} + \cdots + a_n x_n$$

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

$$\frac{\partial f}{\partial x_1} = a_1 \qquad \frac{\delta f}{d x_2} = a_2$$

$$\nabla_x (a^T x) = \boxed{a} \longleftrightarrow \frac{d}{dx}(ax) = a$$

y = $a^T x$ = $a_1 x_1 + a_2 x_2$ +...... $a_n x_n$. So when we do the derivation of above . We'll get 'a'



$$\mathcal{L}(w) = \sum_{i=1}^{n} \log\left(1 + \exp(-y_i w^T x_i)\right) + \lambda w^T w$$

$$\frac{d\, \lambda w^2}{w} = 2\lambda w$$

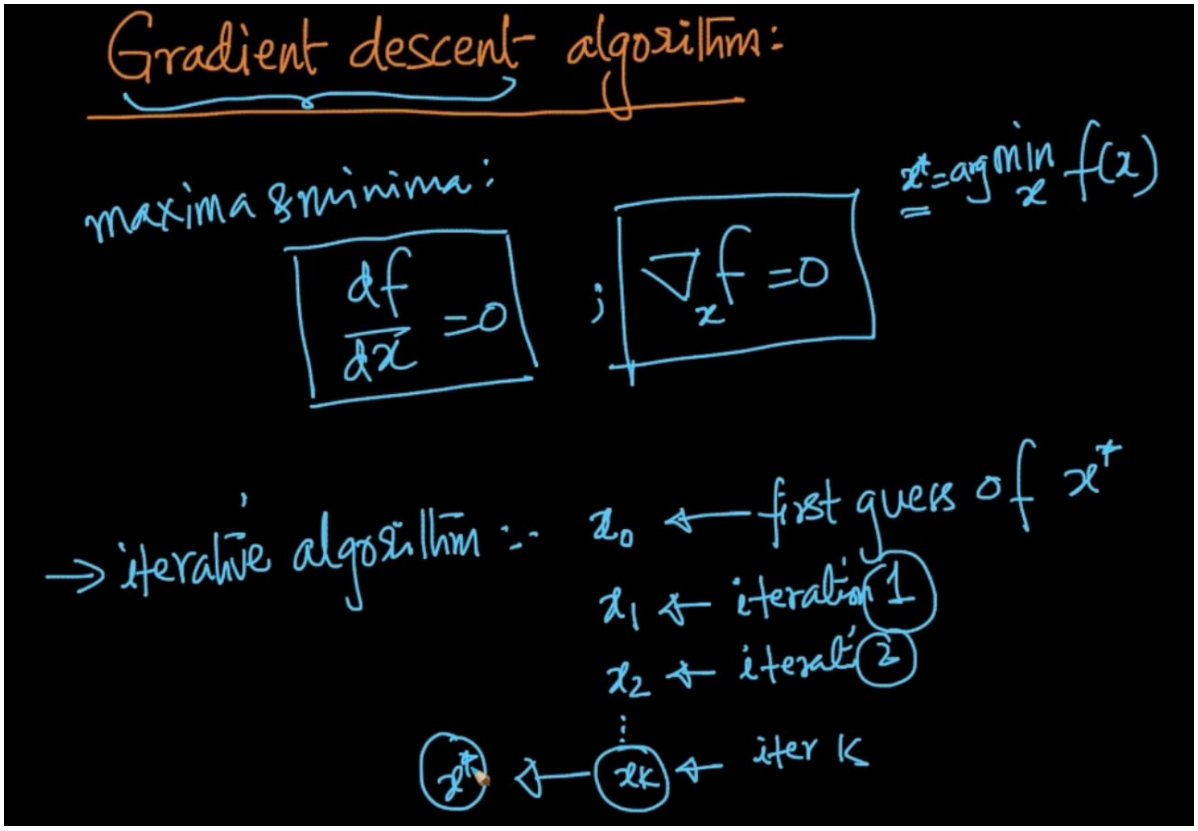
$\langle x_i, y_i \rangle \rightarrow$ constants $\rightarrow D_{train}$

$$\nabla_w \mathcal{L} = \frac{(-y_i x_i)(\exp(-y_i w^T x_i))}{1 + \exp(-y_i w^T x_i)} + 2\lambda w = 0$$

(Gradient descent)

Finding minima.maxima with $\square_x L$ is very hard because the equation is complex af. So, we'll apply computational techniques like Gradient Descent to solve it

GRADIENT DESCENT: GEOMETRIC INTUITION



Gradient descent algorithm:

maxima & minima: $\boxed{\frac{df}{dx} = 0}$; $\boxed{\nabla_x f = 0}$

$x^* = \underset{x}{\operatorname{argmin}} f(x)$

→ iterative algorithm :- $x_0 \leftarrow$ first guess of x^*

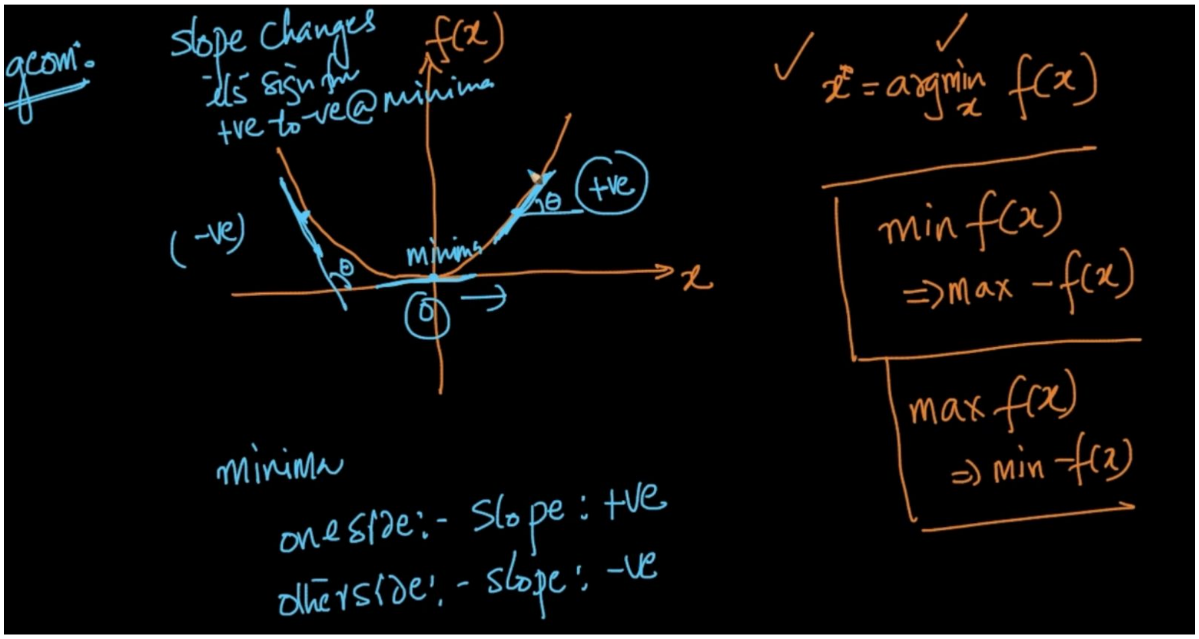
$x_1 \leftarrow$ iteration 1

$x_2 \leftarrow$ iteration 2

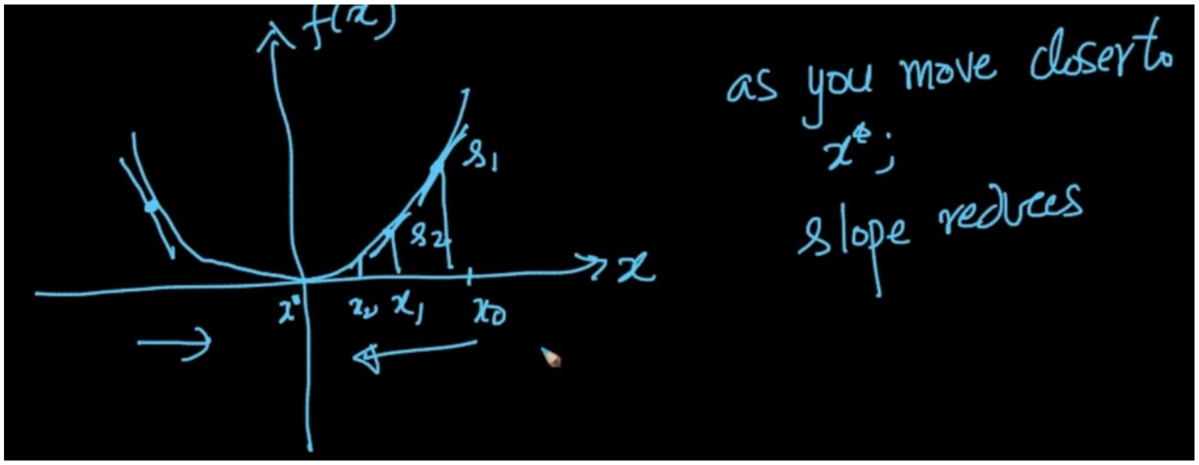
$\vdots$

$x^* \leftarrow x_k \leftarrow$ iter k

Gradient descent is an iterative algorithm. In an Iterative algorithm first we'll guess the value
of x^* (Point where we want to reach)which is x_0. Now by applying our iterative algorithm Gradient Descent we'll get x_1 which is closer to our targeted x^* than x_o

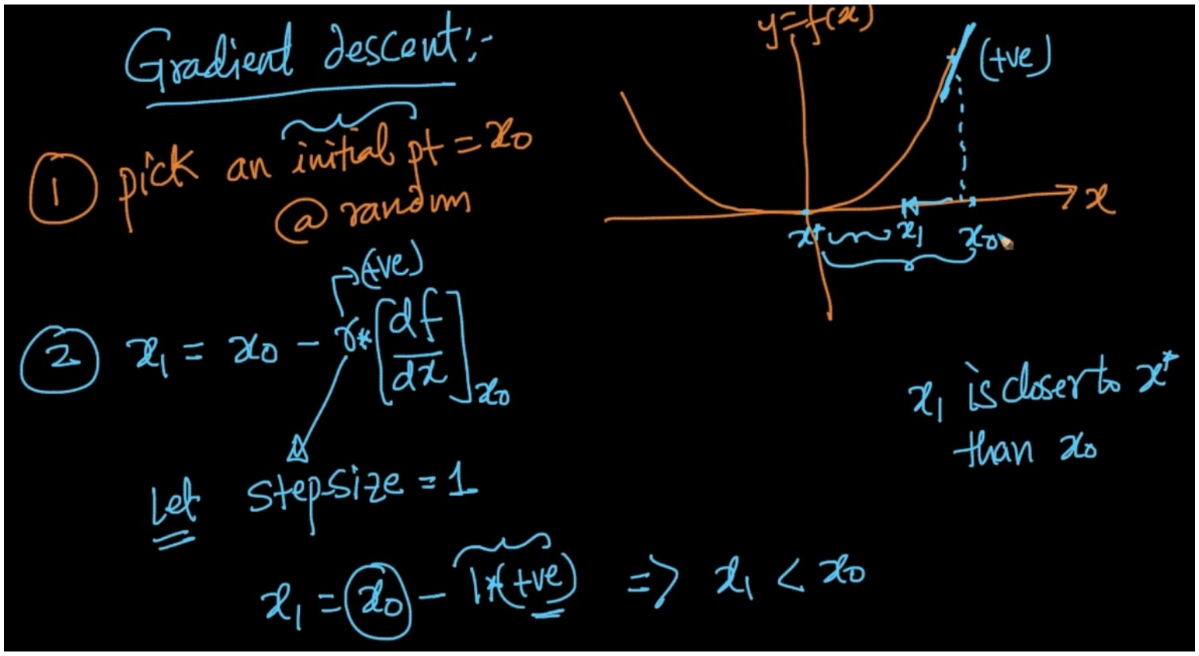


We want to find $x^* =$ argmin f(x) where we want to minimize f(x). min f(x) = max(-f(x)) and vice versa. One important observation here is that the slope change it's sign from +ve to -ve After reaching minima



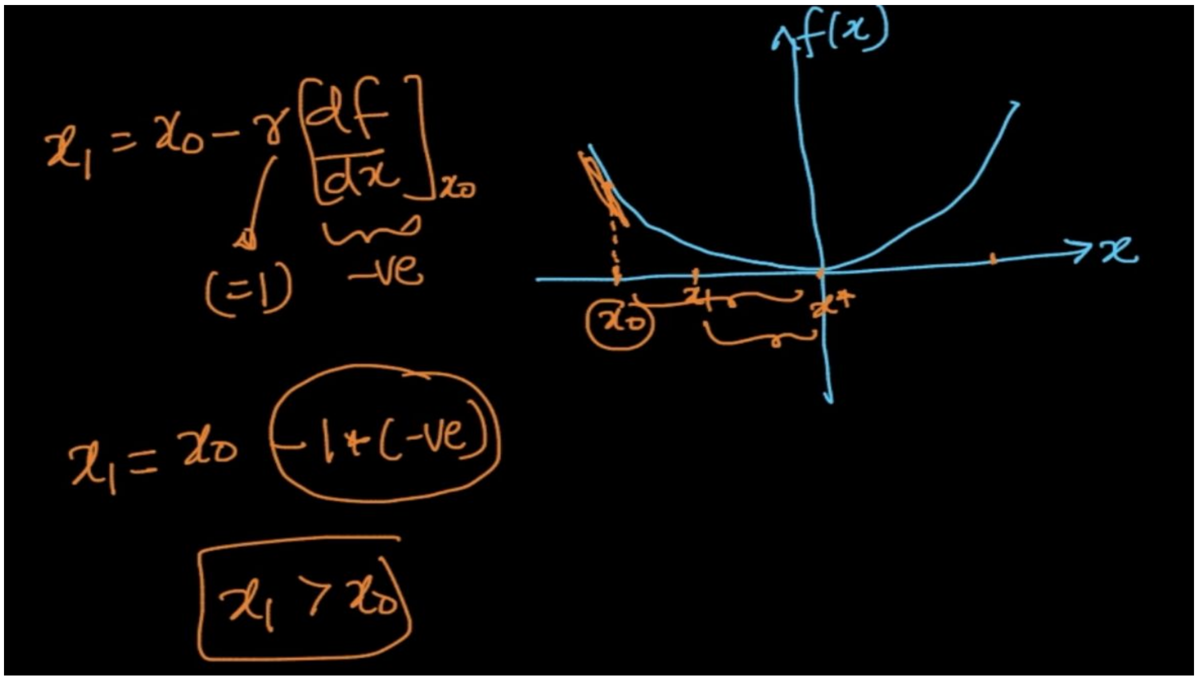
Another observation is as we move closer to x^* from the right side the slope reduces

And it increases if we move from left to right. So , let's see how Gradient Descent works

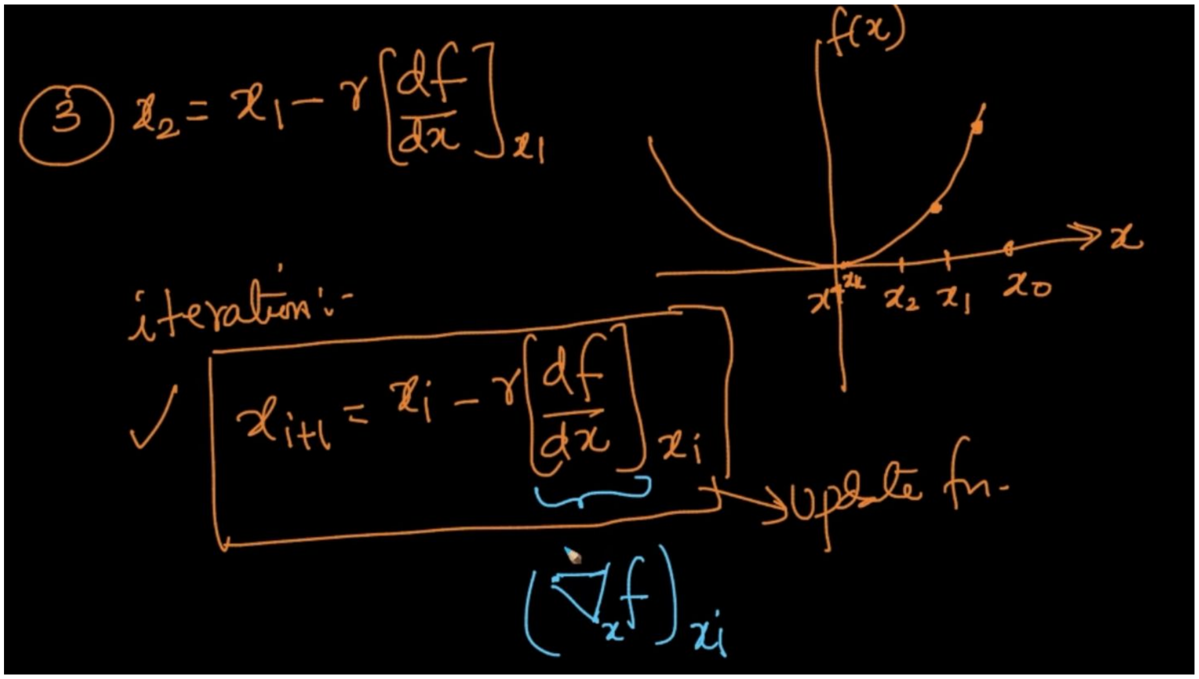

Gradient descent:-
① pick an initial pt = x_0 @ random
② $x_1 = x_0 - r * \left[\frac{df}{dx}\right]_{x_0}$
(+ve)
Let stepsize = 1
$x_1 = x_0 - 1*(+ve) \Rightarrow x_1 < x_0$
$y = f(x)$
(+ve)
x
x^*
x_1
x_0
x_1 is closer to x^* than x_0

1) We'll pick a random initial point = x_o

2) x_1 = x_0 - r $*$ $[\frac{df}{dx}]_{x0}$ where 'r' is step-size and $\frac{df}{dx}$ is the slope . Also, $x_1 < x_0$



Case 2: When x_0 is at the other side

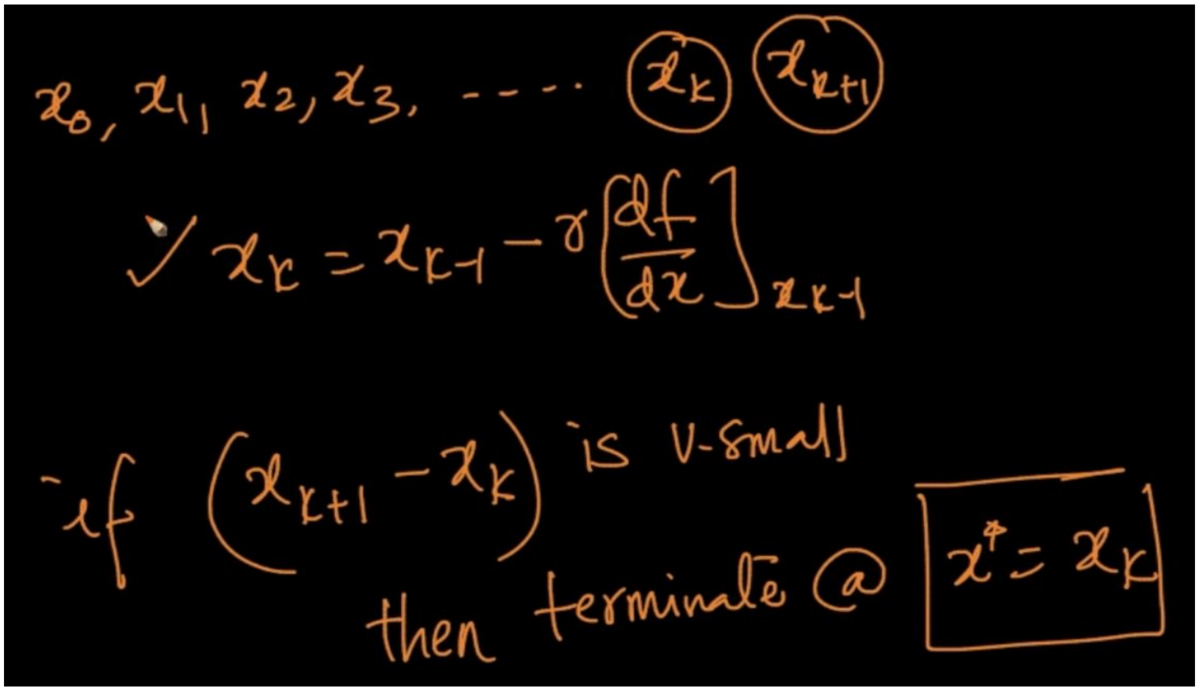


③ $x_2 = x_1 - \gamma \left[\frac{df}{dx}\right]_{x_1}$

iteration:-

✓ $\boxed{x_{i+1} = x_i - \gamma \underbrace{\left[\frac{df}{dx}\right]}_{\left(\nabla_x f\right)_{x_i}}{}_{x_i}} \rightarrow$ update fn.

We'll calculate x_2 from x_1. Our algorithm will do iteration. When will it stop but?

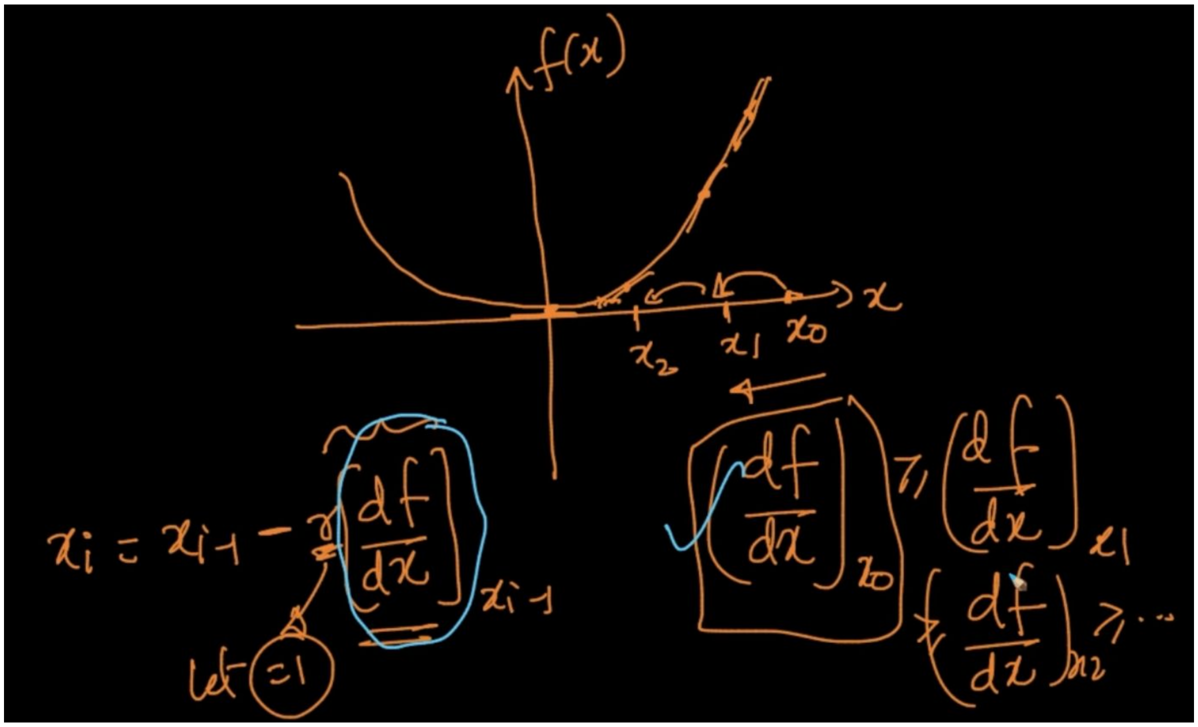


$x_0, x_1, x_2, x_3, \dots \; x_k \; x_{k+1}$

✓ $x_k = x_{k-1} - \gamma \left[\frac{df}{dx}\right]_{x_{k-1}}$

if $\left(x_{k+1} - x_k\right)$ is v-small then terminate @ $\boxed{x^* = x_k}$

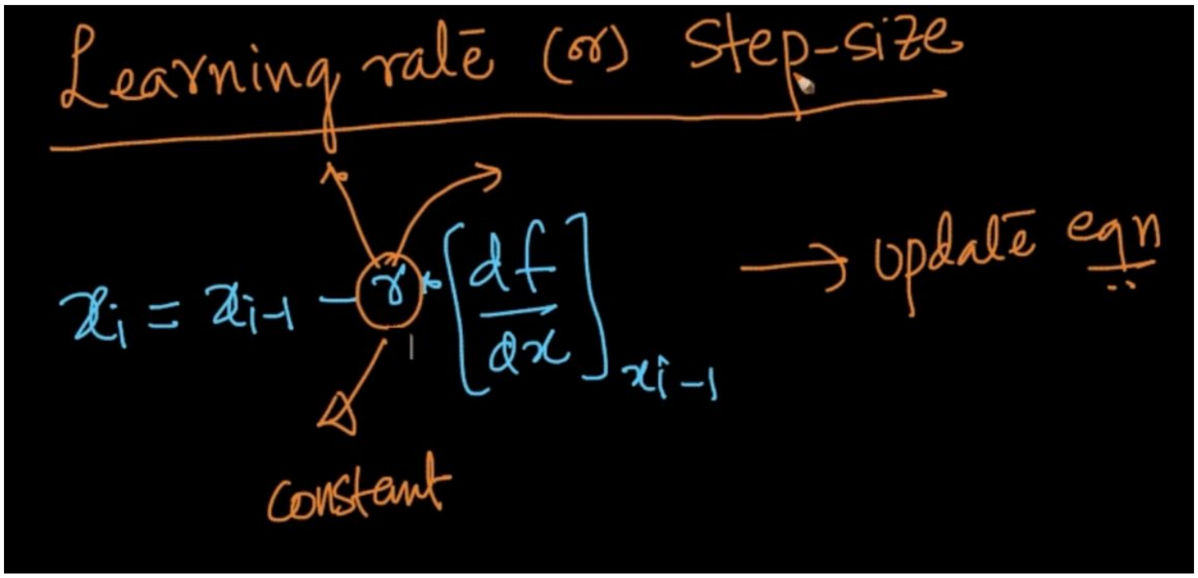
We'll terminate the loop once the difference between two iterations are very small



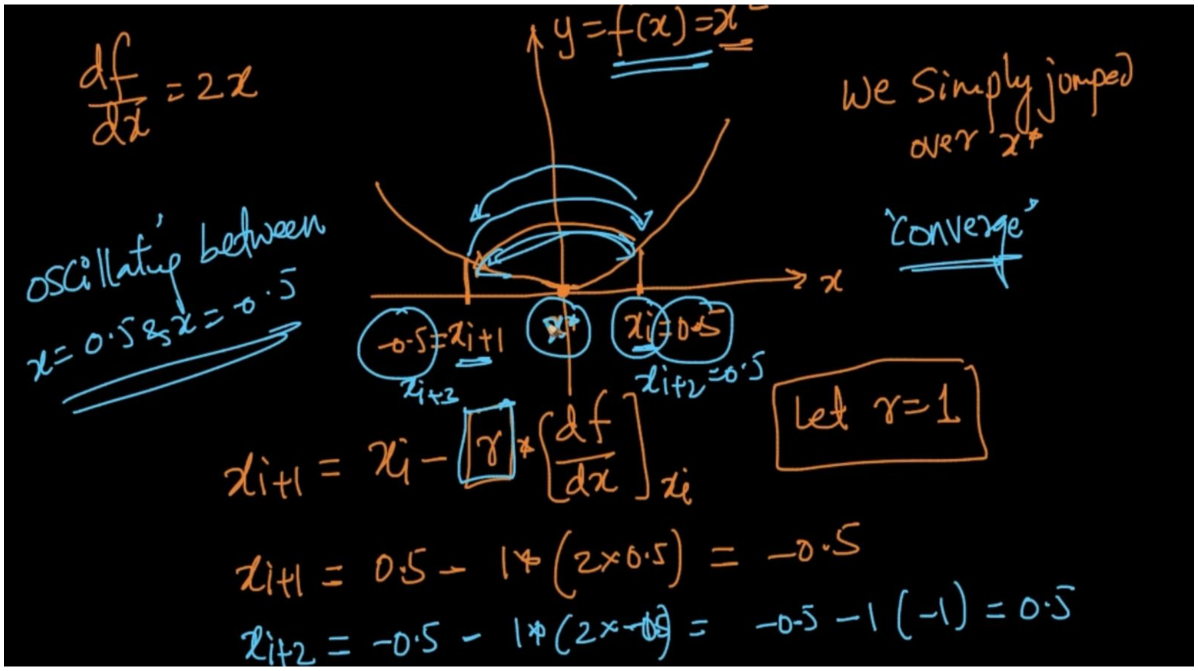
f(x)
x
x2
x1
x0
xi = xi-1 − η (df/dx) xi-1
let =1
(df/dx) x0 ≥ (df/dx) x1 ≥ (df/dx) x2 ≥ ...

$[\frac{df}{dx}]_{x0} > \ [\frac{df}{dx}]_{x1} > \ [\frac{df}{dx}]_{x2}$ …. As we can see since the derivatives are getting smaller and smaller by each iteration. We are taking a huge step and as we are moving to the convergence our step -sizes are getting smaller and smaller . This is what happens in Gradient Descent

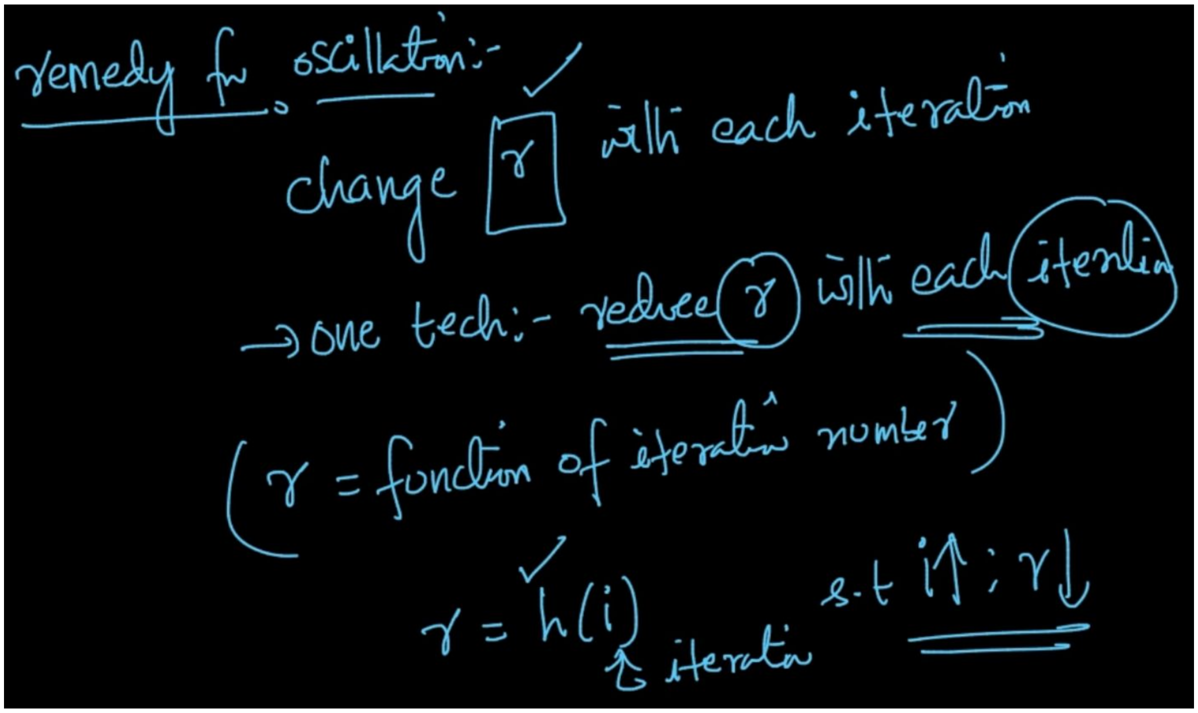
LEARNING RATE


Learning rate (or) Step-size
$x_i = x_{i-1} - r * \left[\frac{df}{dx}\right]_{x_{i-1}}$
→ update eqn
constant

‘r’ is basically learning rate or step-size


$y = f(x) = x^2$
$\frac{df}{dx} = 2x$
We simply jumped over x^*
"converge"
oscillating between $x = 0.5$ & $x = -0.5$
$-0.5 = x_{i+1}$
x^*
$x_i = 0.5$
x_{i+3}
$x_{i+2} = 0.5$
Let $r = 1$
$x_{i+1} = x_i - \boxed{r} * \left[\frac{df}{dx}\right]_{x_i}$
$x_{i+1} = 0.5 - 1 * (2 \times 0.5) = -0.5$
$x_{i+2} = -0.5 - 1 * (2 \times -0.5) = -0.5 - 1(-1) = 0.5$

As we can see at x_{i+1} = -0.5 so we are jumping over x^* and at x_{i+2} = 0.5 so we are oscillating between -0.5 and +0.5 but never converging towards x^* as our r = 1



remedy for oscillation:- ✓

change $\boxed{r}$ with each iteration

→ one tech:- reduce r with each iteration

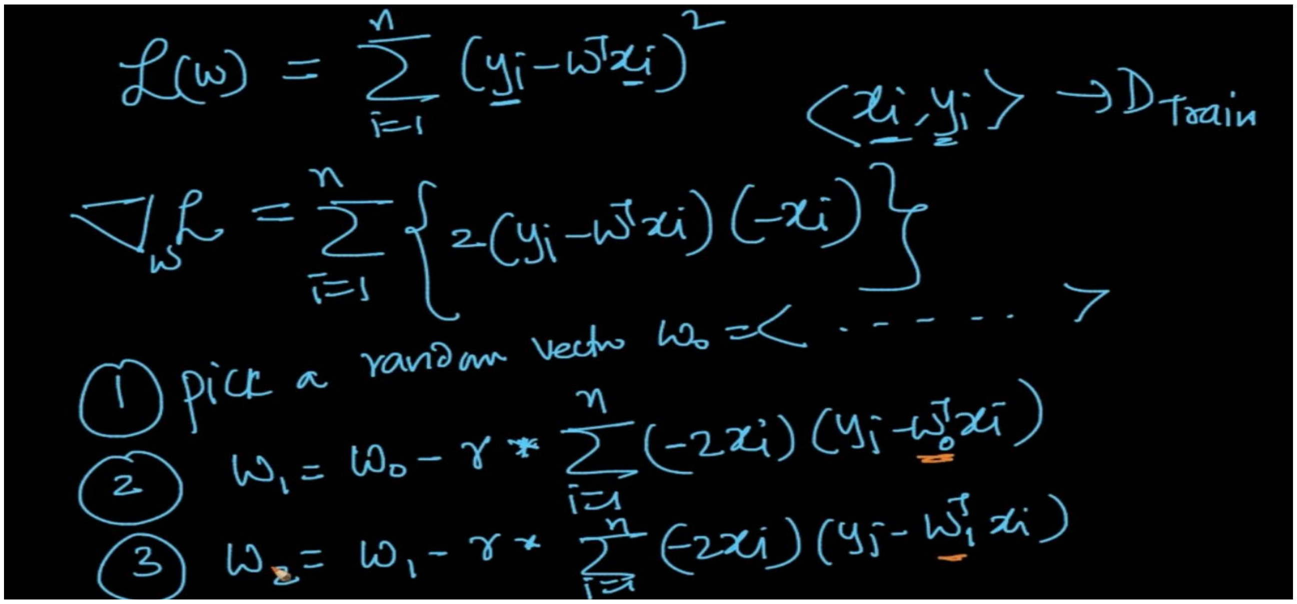
(r = function of iteration number)

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

↑ iteration

One method is to reduce r with each iteration. So we want to make r = h(i) where h is a function such that as i increases r should be decreased

GRADIENT DESCENT FOR LINEAR REGRESSION



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

$\langle x_i, y_i \rangle \rightarrow D_{Train}$

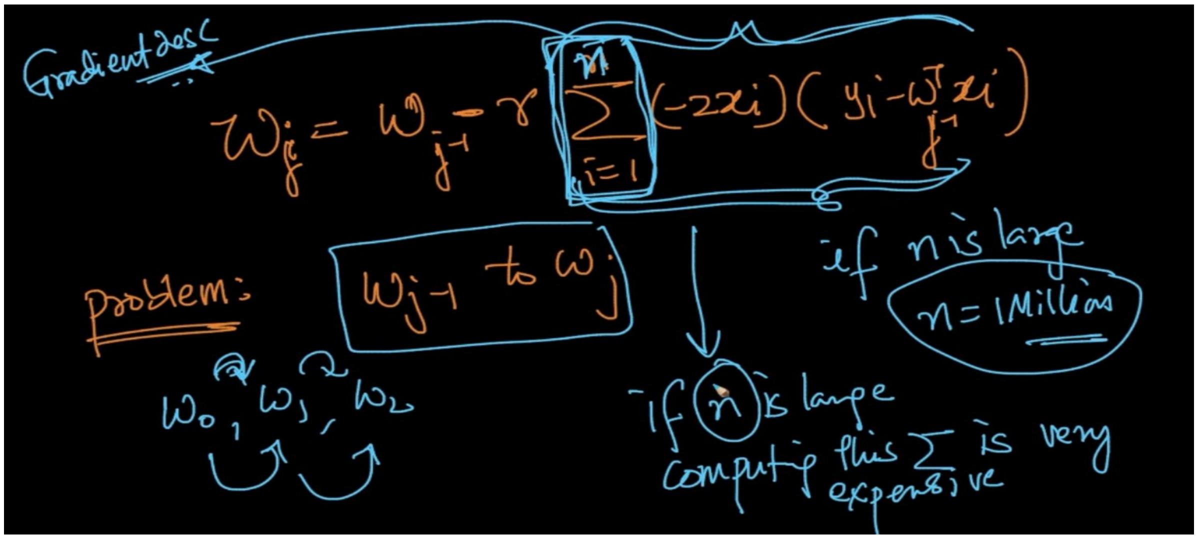
$$\nabla_w \mathcal{L} = \sum_{i=1}^{n} \left\{ 2(y_i - w^T x_i)(-x_i) \right\}$$

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

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

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

L(w) is a function for ‘w’ and we want to find the best w not x as x,y are constants as given in our dataset. So we need $\nabla_w L$. We pick a random vector w_o and apply it to get the next w_1. w_1 will be used to get w_2 and this step will be repeated till difference between w_k and w_{k+1} is very small . If that’s the case then $w^* = w_k$



Gradient desc

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

if n is large

$n = 1$ Millions

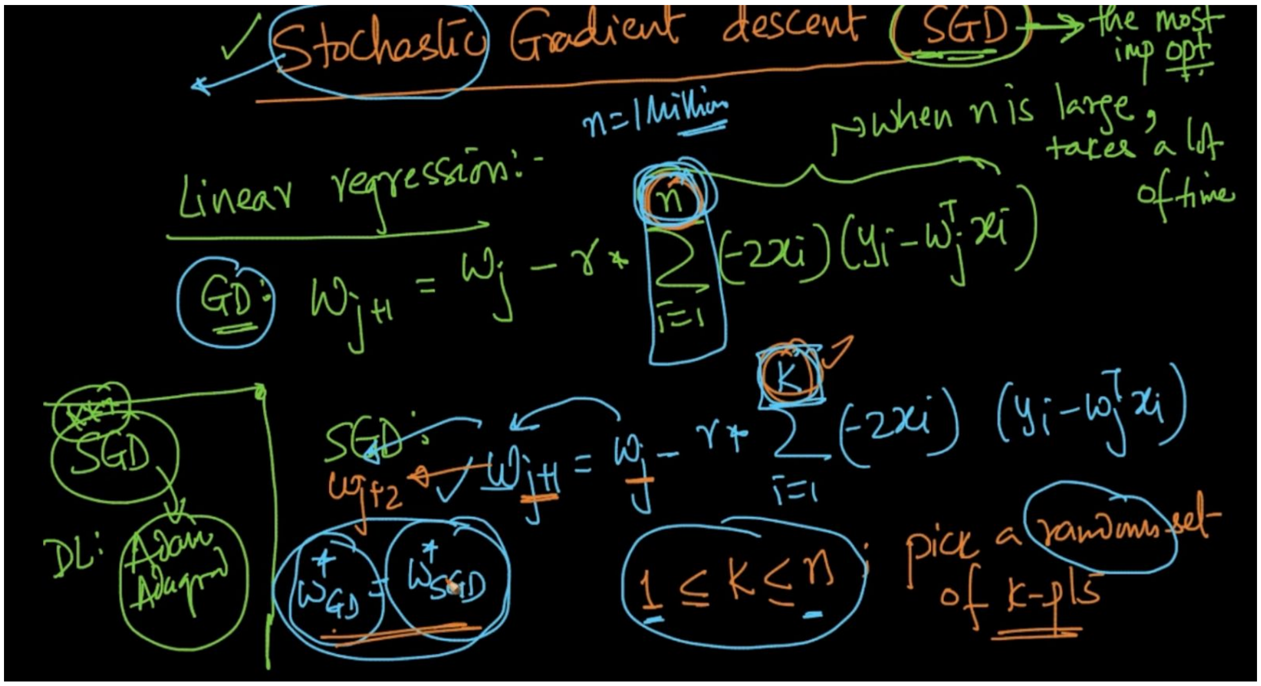
problem: $\boxed{w_{j-1} \text{ to } w_j}$

w_0, w_1, w_2

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

There’s one problem here i.e for every iteration in w i.e w0,w1.. we need to calculate
The summation thing and if ‘n’ is very large then we need to do it n times for every iteration

STOCHASTIC GRADIENT DESCENT (SGD)


Stochastic Gradient descent (SGD)
the most imp opt
n=1 Million
when n is large, takes a lot of time
Linear regression:-
GD: $W_{j+1} = W_j - r * \sum_{i=1}^{n} (-2x_i)(y_i - W_j^T x_i)$
SGD: $W_{j+1} = W_j - r * \sum_{i=1}^{k} (-2x_i)(y_i - W_j^T x_i)$
w_{j+2}
SGD
DL: Adam Adagrad
$W^*_{GD} \approx W^*_{SGD}$
$1 \le k \le n$; pick a random set of k-pts

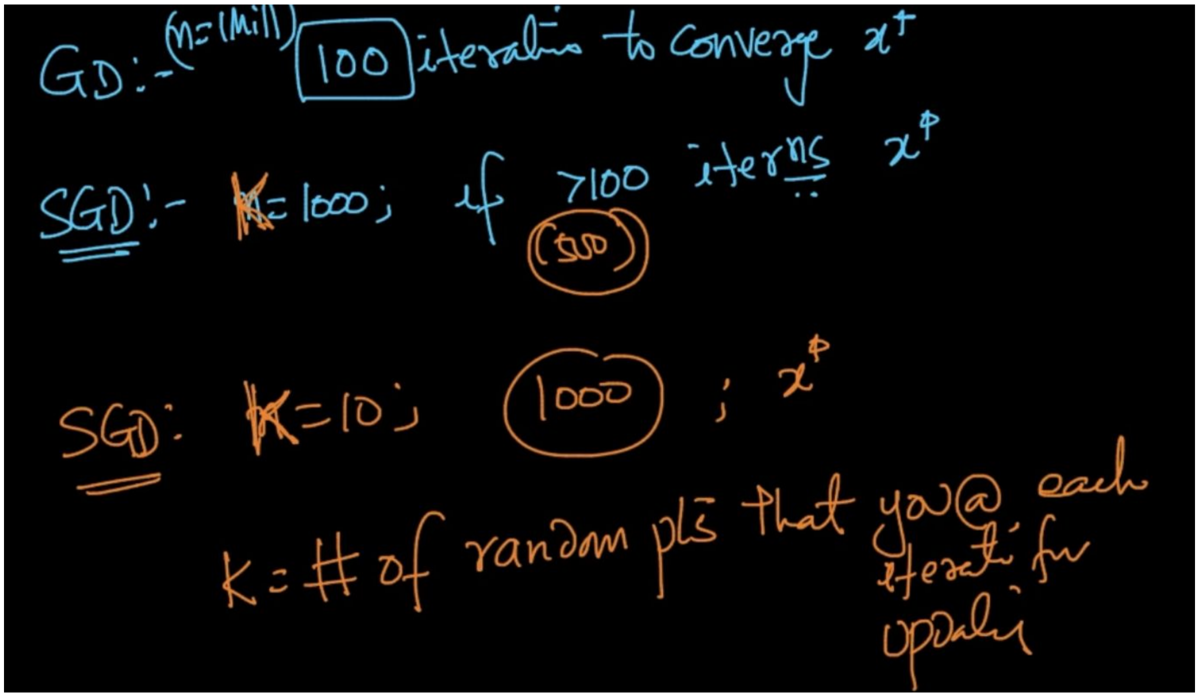
Stochastic Gradient Descent is the most important optimization algorithm in ML

In Gradient Descent , we'd the problem for w as for every iteration we'd to go through the whole dataset and if n-No. Of inputs is large we are fucked.

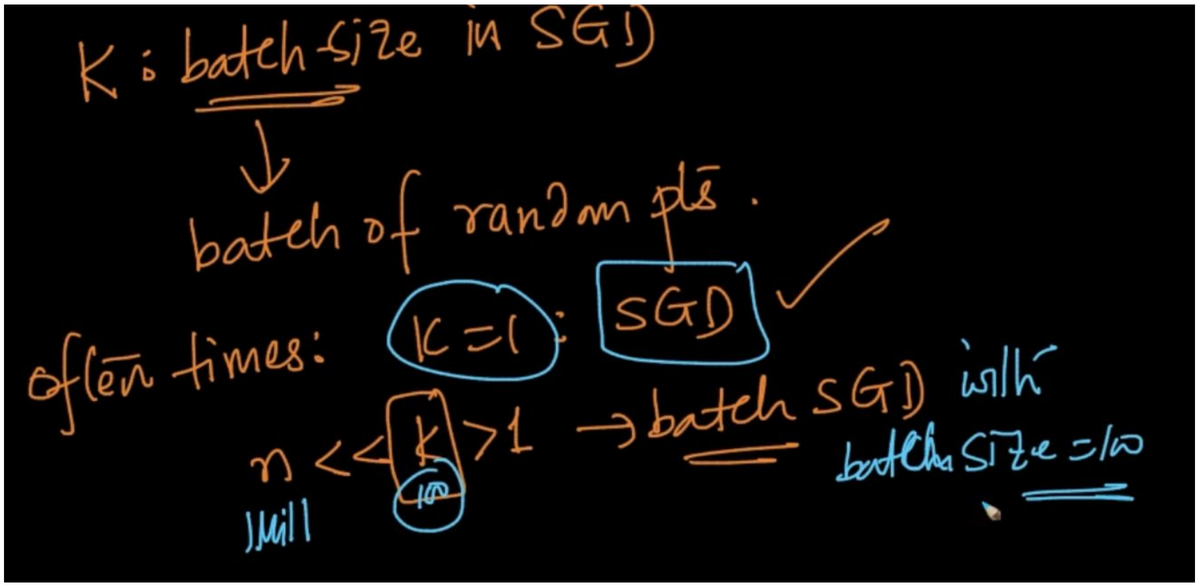
We can use SGD (Stochastic Gradient Descent) in which we do the same operation but instead of 'n' we use 'k' where k << n and it is random set of points from the dataset

We can infer that $w^*_{GD} = w^*_{SGD}$.

NOTE - With every iteration of w i.e w_{j+1} => w_{j+2} we are changing the value of k

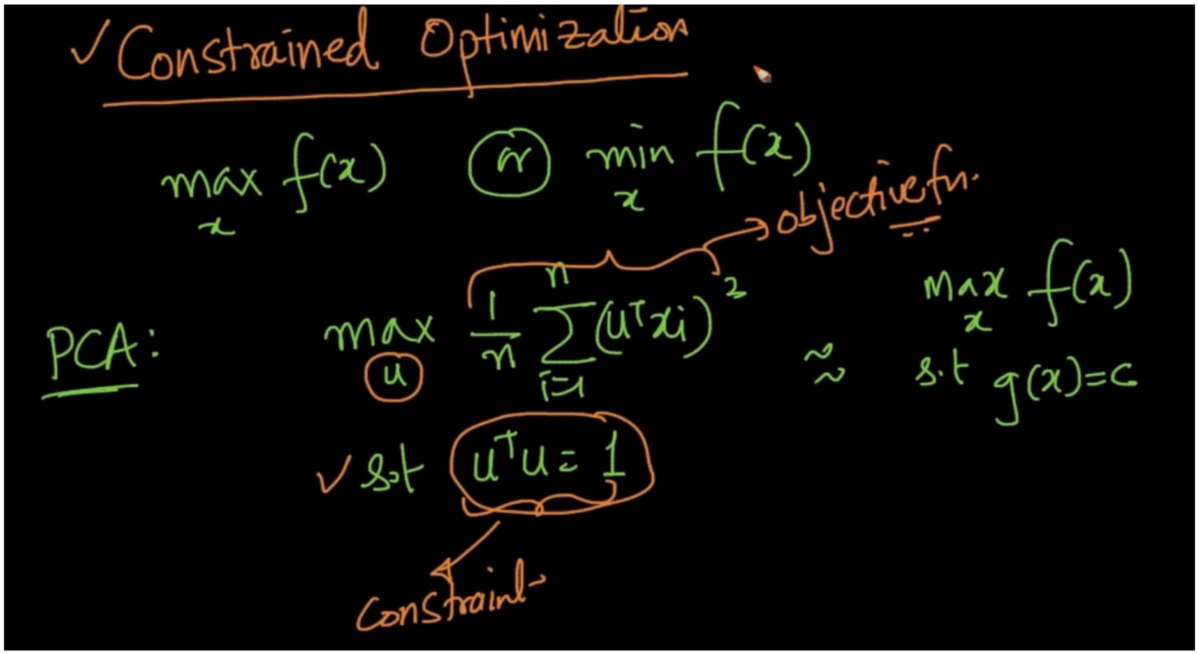

GD:- (n=1Mill) 100 iterations to converge x*
SGD:- k=1000; if >100 iterns x* (500)
SGD: k=10; 1000 ; x*
k = # of random pts that you @ each iteration for update

If Gradient Descent, n = 1 Million points and it takes 100 iterations to converge x^* but in SGD, if k = 1000, then it'll take >100 iterations to converge. Let's say 500.Same for k = 10

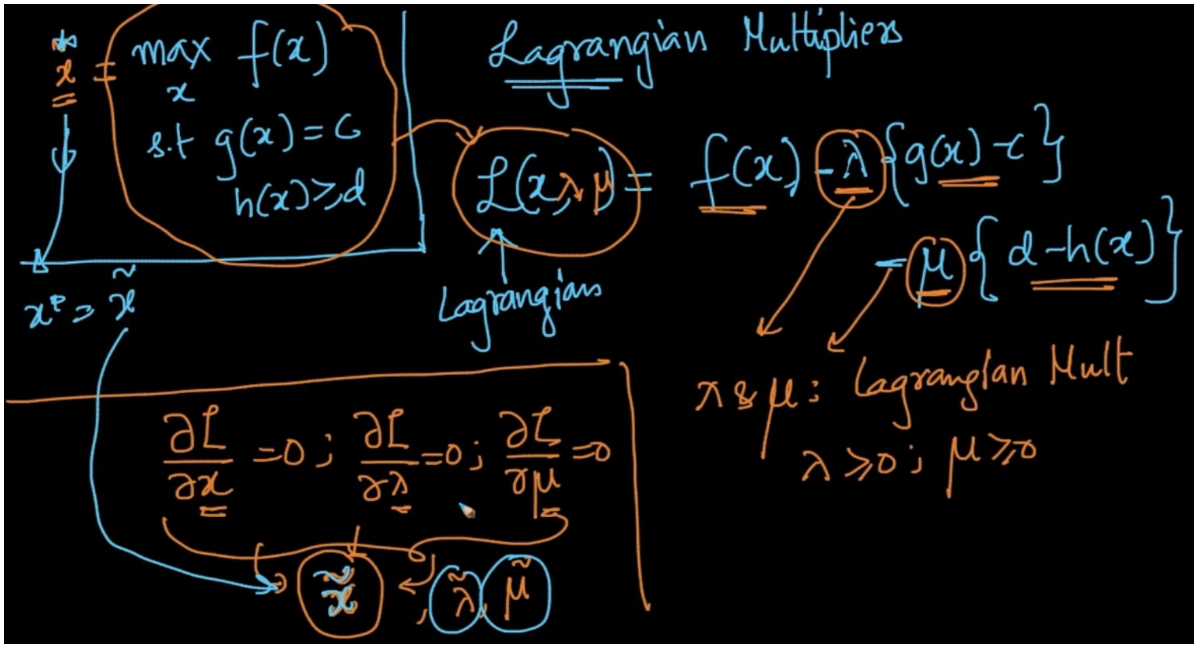


K: is also called batch size as we are using batch of random points

CONSTRAINTS AND PCA

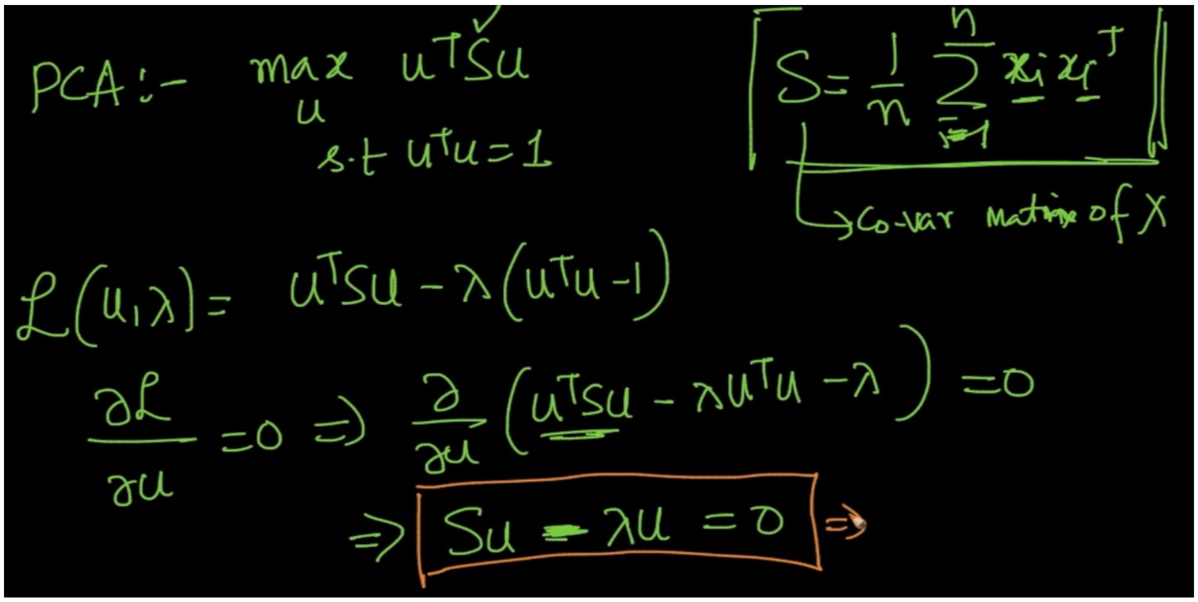

✓ Constrained Optimization
$\max_x f(x)$ or $\min_x f(x)$
objective fn.
PCA:
$\max_u \frac{1}{n}\sum_{i=1}^{n}(u^T x_i)^2$
✓ s.t $u^T u = 1$
constraint
$\max_x f(x)$ s.t $g(x)=c$

We want to maximize a function f(x) with a constraint i.e g(x) = c. How will we tackle problems when dealing with situations like this


Lagrangian Multipliers
$x^* = \max_x f(x)$ s.t $g(x)=c$, $h(x)\geq d$
$x^* = \tilde{x}$
$\mathcal{L}(x,\lambda,\mu) = f(x) - \lambda\{g(x)-c\} - \mu\{d-h(x)\}$
Lagrangian
λ & μ: Lagrangian Mult
$\lambda \geq 0$; $\mu \geq 0$
$\frac{\partial \mathcal{L}}{\partial x}=0$; $\frac{\partial \mathcal{L}}{\partial \lambda}=0$; $\frac{\partial \mathcal{L}}{\partial \mu}=0$
$\tilde{x}$, $\tilde{\lambda}$, $\tilde{\mu}$

When we need to find a max of function with constraints we add Lagrangian Multipliers

λ, μ as seen above. Now take the derivatives w.r.t x, λ, μ as seen above. We'll get $\tilde{x}$, which will be equal to x* that we wanted



PCA:- $\max_{u} \; u^T S u$

s.t $u^T u = 1$

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

Co-var matrix of X

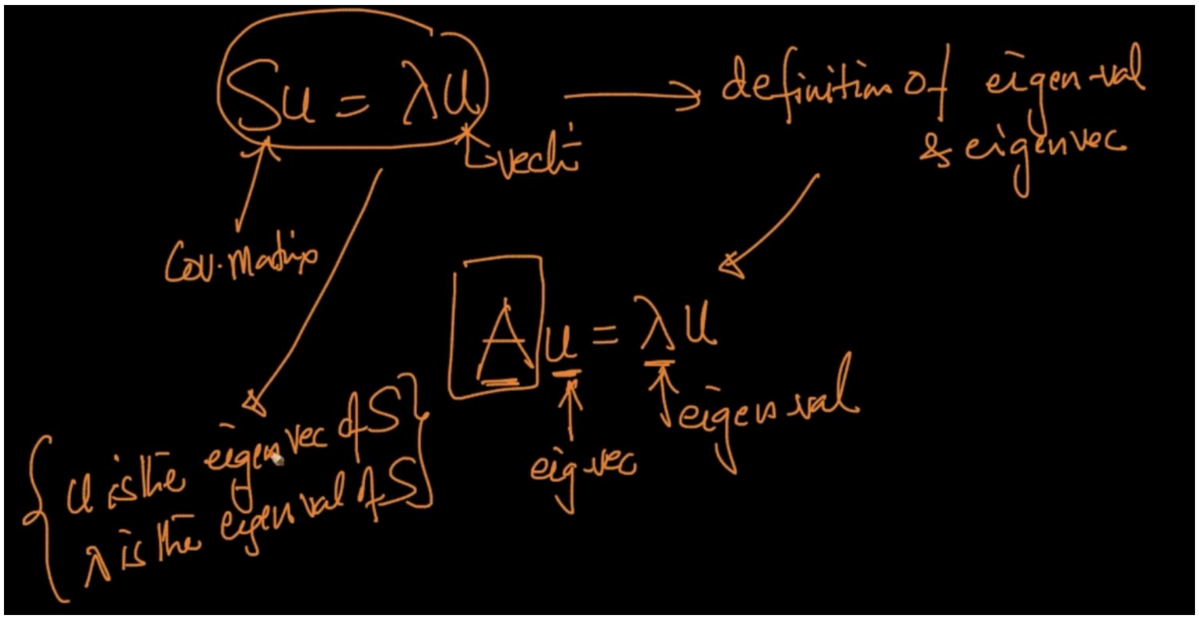
$$\mathcal{L}(u,\lambda) = u^T S u - \lambda(u^T u - 1)$$

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

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

In PCA, we want the max of $u^T S u$ such that $u^T u = 1$ where S is covariance matrix

We take the derivative of L w.r.t u = 0 and we get Su - λu = 0



$$Su = \lambda u \rightarrow \text{definition of eigen-val \& eigenvec}$$

Cov. Matrix

vector

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

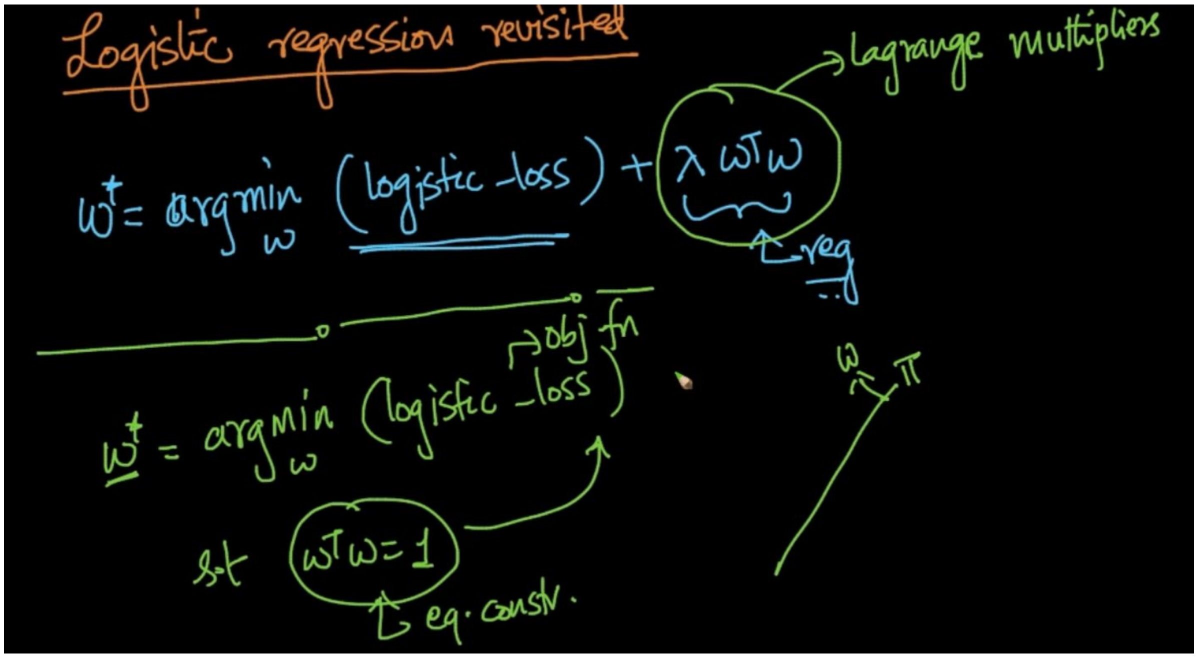
eig-vec

eigen-val

$\left\{ \begin{array}{l} u \text{ is the eigen vec of } S \\ \lambda \text{ is the eigen val of } S \end{array} \right\}$

u is the eigen-vector of S and λ is the eigen-value of S as it's the same we'd seen in PCA

LOGISTIC REGRESSION REVISITED



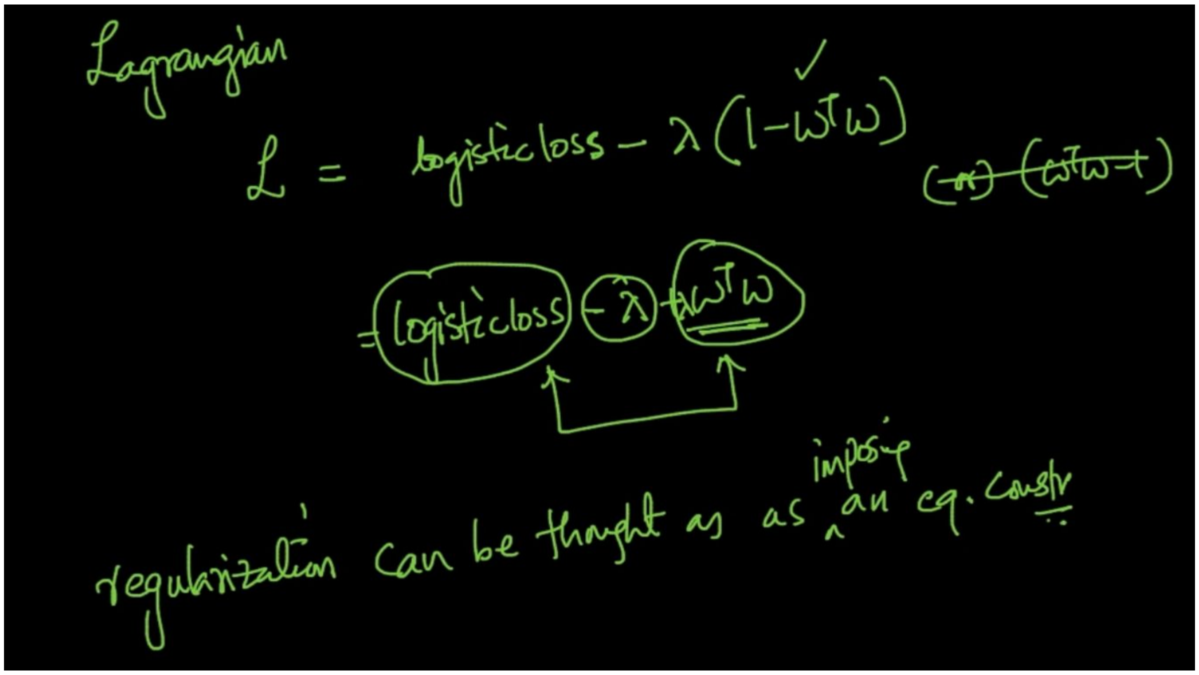
Logistic regression revisited

$$w^* = \arg\min_w \underline{(\text{logistic-loss})} + \underbrace{\boxed{\lambda w^T w}}_{\text{reg}} \xrightarrow{} \text{Lagrange multipliers}$$

$$w^* = \arg\min_w (\text{logistic\_loss}) \xrightarrow{} \text{obj fn}$$

s.t. $\boxed{w^T w = 1}$ ← eq. constr.

We want to find w* such that $w^T w = 1$



Lagrangian

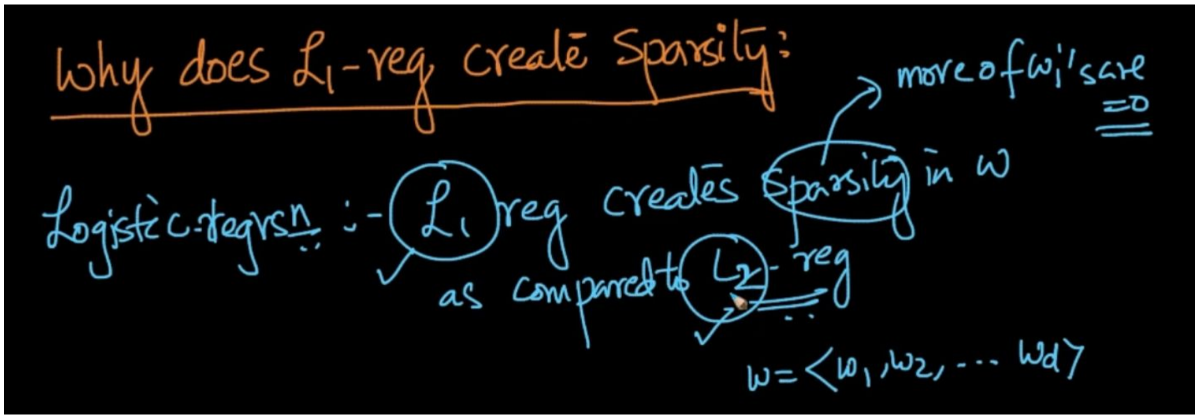
$$\mathcal{L} = \text{logisticloss} - \lambda(1 - w^T w) \quad \cancel{(-\lambda)(w^T w - 1)}$$

$$= \boxed{\text{logisticloss}} - \boxed{\lambda} + \boxed{\lambda w^T w}$$

regularization can be thought as as imposing an eq. constr

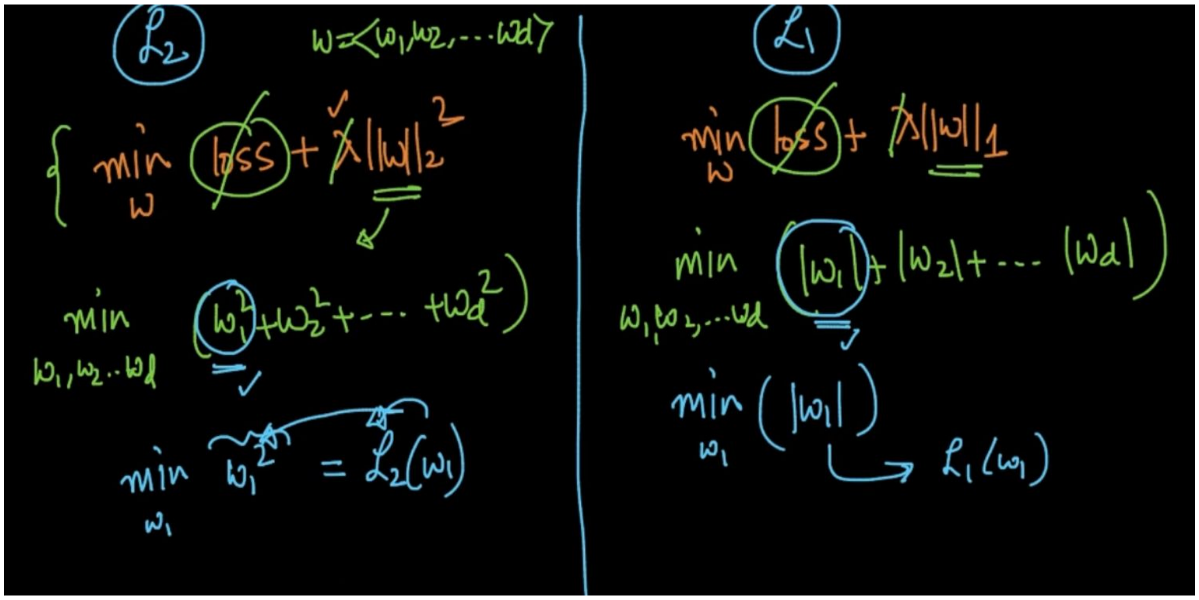
When applying Lagrangian we get the value same as w* in which regularization was also there. So regularization can also be thought as imposing an equality constraint

WHY L1-REGULARIZATION CREATES SPARSITY?



why does L_1-reg create sparsity:
Logistic regrsn :- L_1 reg creates sparsity in w
more of w_i's are $=0$
as compared to L_2-reg
$w=\langle w_1, w_2, \ldots w_d \rangle$

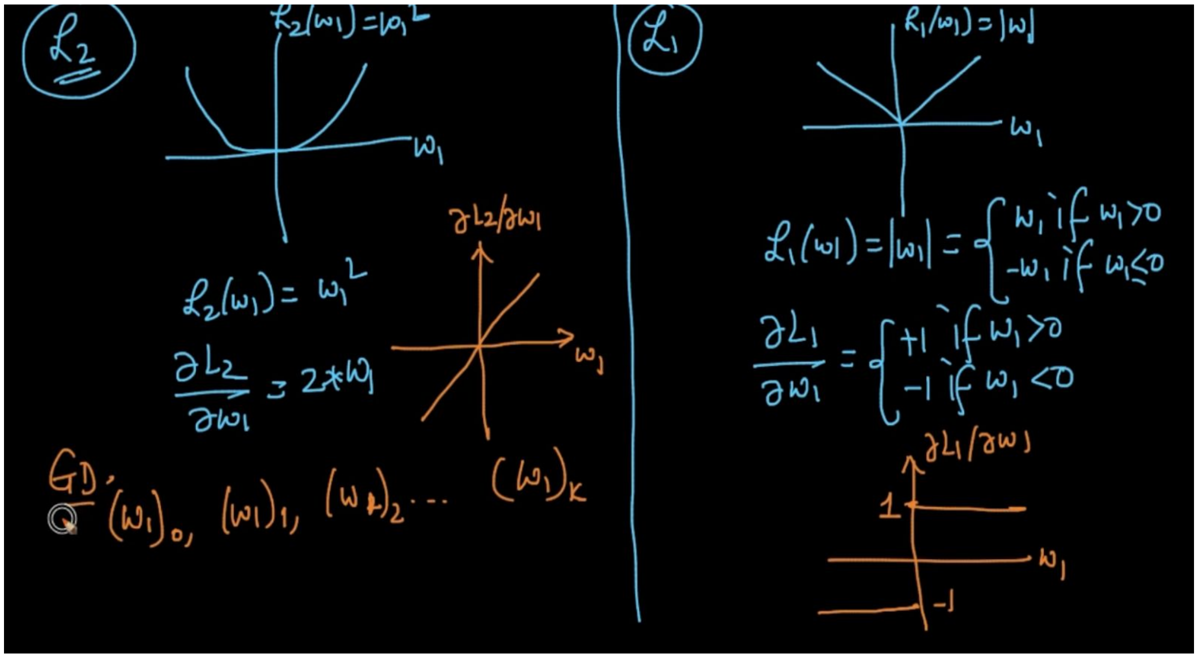
L_1 regularization creates sparsity in 'w' as compared to L_2 regularization. Let's see why it happens



L_2
$w=\langle w_1, w_2, \ldots w_d \rangle$
$\min_w \cancel{loss} + \cancel{\lambda} \|w\|_2^2$
$\min_{w_1, w_2 .. w_d} (w_1^2 + w_2^2 + \ldots + w_d^2)$
$\min_{w_1} w_1^2 = L_2(w_1)$
L_1
$\min_w \cancel{loss} + \cancel{\lambda} \|w\|_1$
$\min_{w_1, w_2, \ldots w_d} (|w_1| + |w_2| + \ldots |w_d|)$
$\min_{w_1} (|w_1|) \rightarrow L_1(w_1)$

In both the Regularizations, we are ignoring the Loss and λ. So, our goal is just to minimize 'w'. In L_2, we want to minimize ($w_1^2 + w_2^2 + w_3^2 + \ldots. + w_d^2$) and in L_1 we want to minimize ($|w_1| + |w_2| + \ldots\ldots |w_d|$). Let's se contribution of just w_1 to our regularization in L i.e (L(w_1))

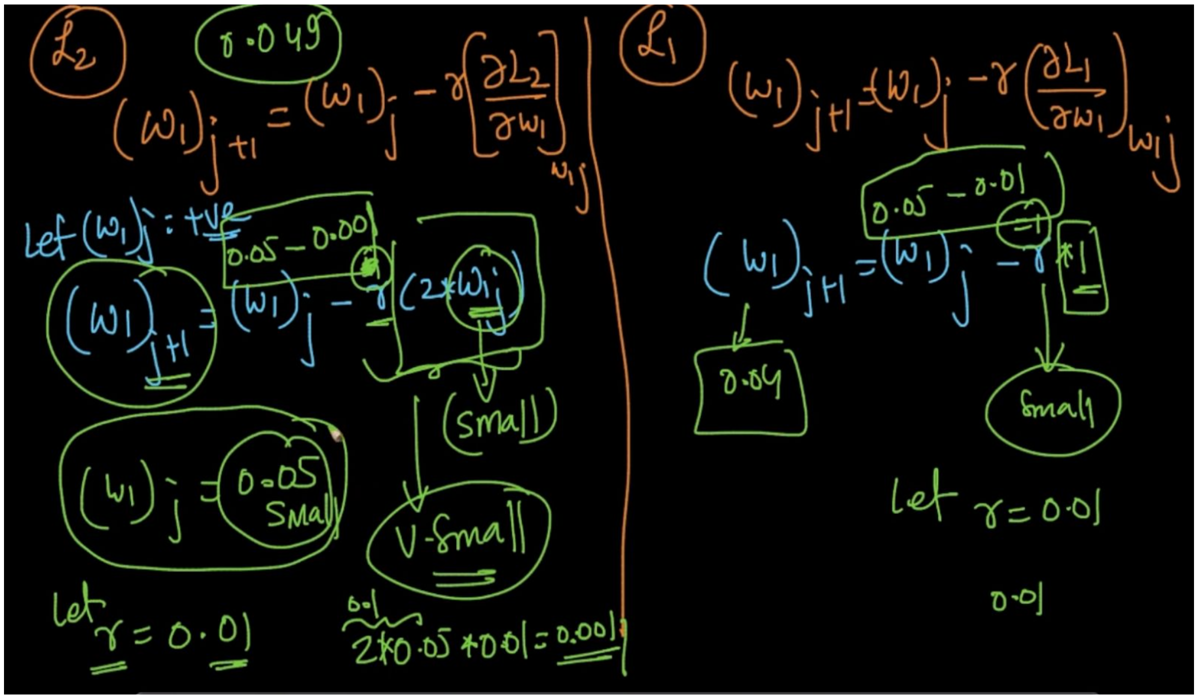
For L_2, min of w_1^2 is the contribution of w_1 to L_2 regularization



L_2
$L_2(w_1) = w_1^2$
w_1
$\partial L_2/\partial w_1$
$L_2(w_1) = w_1^2$
$\frac{\partial L_2}{\partial w_1} = 2 * w_1$
GD: $(w_1)_0, (w_1)_1, (w_1)_2 \dots (w_1)_k$
L_1
$L_1(w_1) = |w_1|$
$L_1(w_1) = |w_1| = \begin{cases} w_1 \text{ if } w_1 > 0 \\ -w_1 \text{ if } w_1 \le 0 \end{cases}$
$\frac{\partial L_1}{\partial w_1} = \begin{cases} +1 \text{ if } w_1 > 0 \\ -1 \text{ if } w_1 < 0 \end{cases}$
$\partial L_1/\partial w_1$
1
-1

We are plotting the graphs of $w_1 v.s\ L_2(w_1)$, $w_1 v.s\ L_1(w_1)$ and its derivative graph.

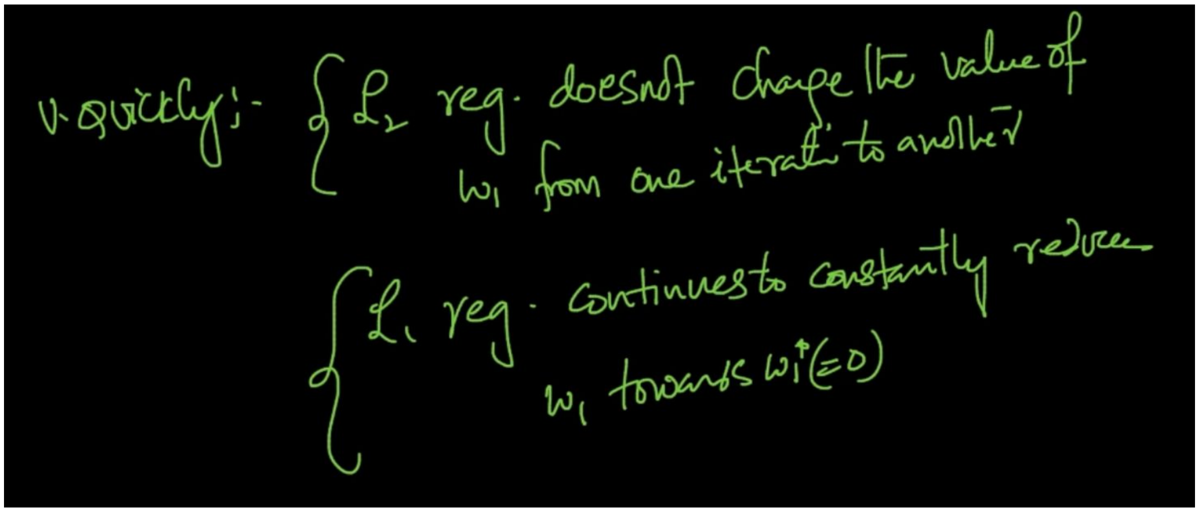
Our gradient Descent calculates it for each iteration of w_1 till it converges



L_2
0.049
$(w_1)_{j+1} = (w_1)_j - r\left[\frac{\partial L_2}{\partial w_1}\right]_{w_1 j}$
Let $(w_1)_j$: +ve
0.05 − 0.001
$(w_1)_{j+1} = (w_1)_j - r(2 * w_{1j})$
(Small)
V-Small
$(w_1)_j = 0.05$ Small
Let $r = 0.01$
0.1
$2 * 0.05 * 0.01 = 0.001$
L_1
$(w_1)_{j+1} = (w_1)_j - r\left(\frac{\partial L_1}{\partial w_1}\right)_{w_1 j}$
0.05 − 0.01
$(w_1)_{j+1} = (w_1)_j - r * 1$
0.04
Small
Let $r = 0.01$
0.01

So, our w_1 at j'th iteration = 0.05 and r = 0.05. In L_2, $(w_1)_{j+1}$ = 0.05 - 0.001 = 0.049 and in

L_1 , $(w_1)_{j+1}$ = 0.05 - 0.01= 0.04 so our L_1 is already close to 0 as compared to L_2



V. quickly:- { L_2 reg. doesnot change the value of w_1 from one iteratn to another

{ L_1 reg. continues to constantly reduce w_1 towards w_1^* $(=0)$

It can be seen that L_2 regularization is slow at converging with each iteration whereas L_1 constantly changes as it has a constant slope/gradient so it converges faster towards w_1^*