Solving Optimization Problems

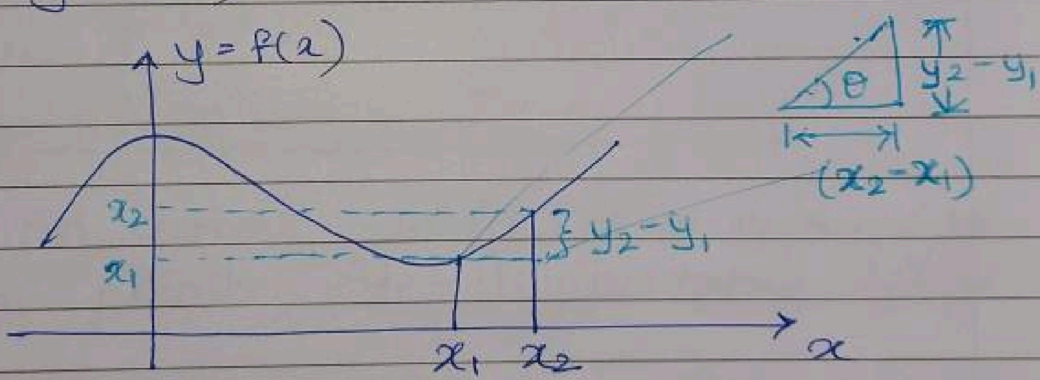
35-1 Differentiation

We have seen optimization problems like:
- → PCA
- → Logistic Regression
- → Linear ——"——

Concepts used to solve these problems
- → Differentiation
- → Maxima & Minima

Single variable differentiation

$y = f(x)$



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

= differentiation of y w.r.t. x

Intuitively it means
- → How much does y change as x changes
- → Rate of change of ~~x~~ y as x changes.

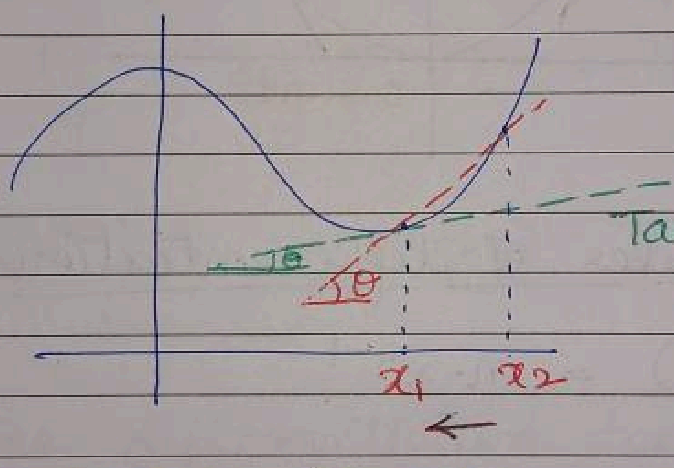
From the graph, how much does 'y' changes as 'x' changes.

$$= \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} \Big\} \text{Rate of change of } y \text{ in vicinity of } x.$$

$$= \tan\theta$$

<u>Mathematically</u>,

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$$



As x_2 moves closes to x_1;

Tangent of $f(x)$ at x_1

As $x_2 \to x_1$; $\Delta x \to 0$

As $x_2 \to x_1$, the secant becomes tangent.

Tangent is the hypotenuse that we obtain as $\Delta x \to 0$

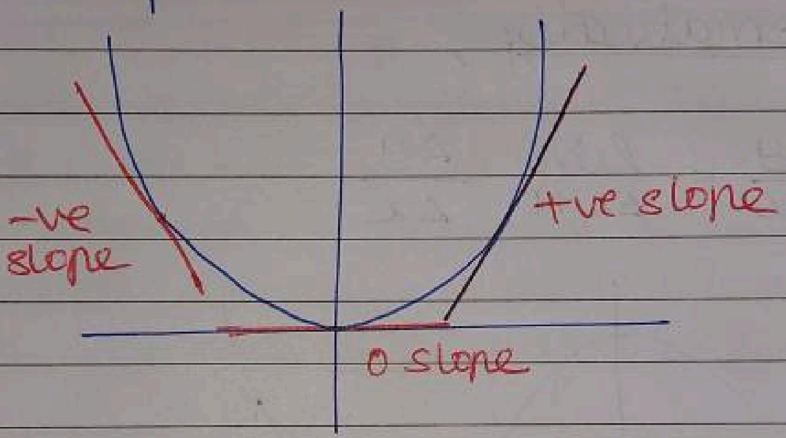
$$\tan\theta = \frac{dy}{dx} ; \text{ as } \Delta x \to 0 ; \frac{\Delta y}{\Delta x} = \frac{dy}{dx}$$

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

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

Slope is +ve ; $0 < \theta < 90^\circ$
Slope is −ve ; $90 < \theta < 180^\circ$

Sum it up :



Basic Rules of differentiation

$$\frac{d}{dx}(x^n) = n \cdot x^{n-1}$$

$$\frac{d}{dx}(c) = 0$$

$$\frac{d}{dx}(\log x) = \frac{1}{x}$$

Chain Rule :

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

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

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

$f(x) = x^2$

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

Now $\frac{dg}{dx} = \frac{d}{dx}(a-bx) = -b.$

Now $\frac{df}{dg}$; Now take $g(x) = z$

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

$$\therefore \frac{df}{dg} = \frac{dz^2}{dz} = 2z = 2(a-bx)$$

$$\therefore \frac{d}{dx} f(g(x)) = \underline{-2b(a-bx)}$$

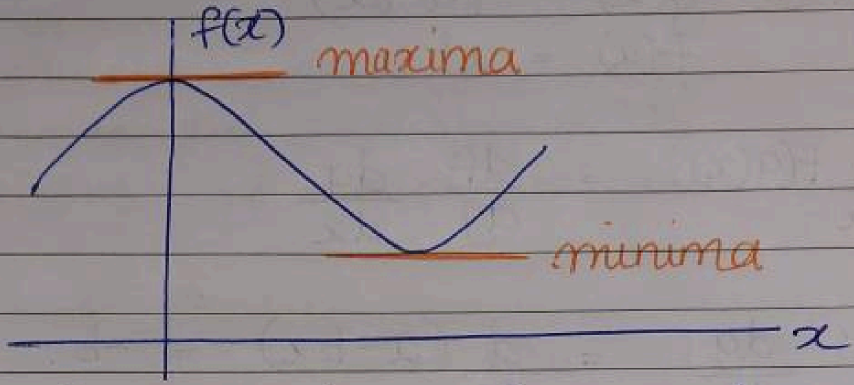
35.2 <u>Online differentiation tool</u>

Visit <u>derivative-calculator.net</u>

symbolab.com

wolframalpha.com → plots the derivative

35.3 Minima and Maxima



At minima and maxima ; slope = 0

Eg: (1) $f(x) = x^2 - 3x + 2$

I know that slope = 0 at Min & Max

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

$\therefore 2x - 3 = 0$

$x = 3/2$

Now is this maxima/minima?

If we ~~va~~ evaluate $x = 1.5$

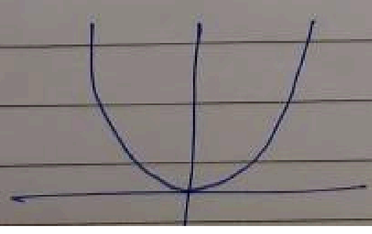
$\therefore f(1.5) = -0.25$

Now, we check $f(1) = 0$

Now $f(1.5) < f(1)$

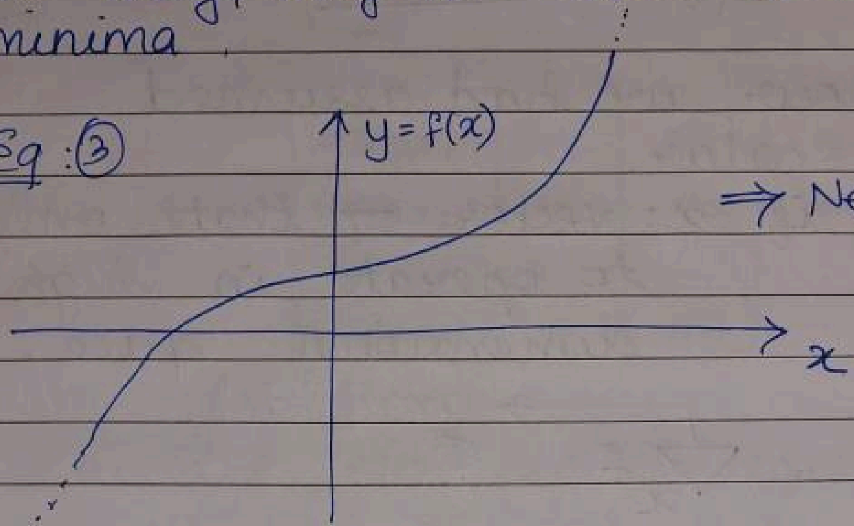
Hence 1.5 cannot be a maxima as there is some other point greater than that.

(2) $y = f(x) = x^2$ has no maxima; it has minima at $x = 0$



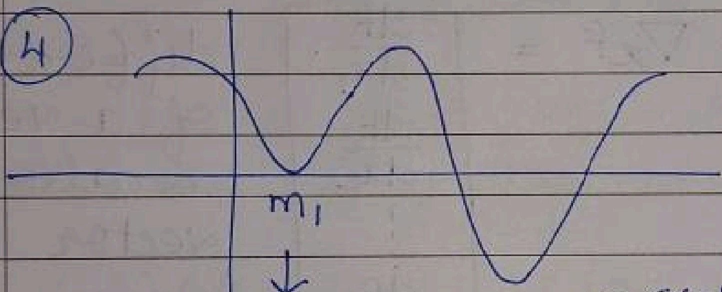
We may/may not have maxima or minima

Eg: (3)



$\Rightarrow$ Neither Maxima Nor Minima

(4)



m_1 and m_2 are two minima

Local Minima
(It is w.r.t locality)
in its neighbourhood

$m_2 \rightarrow$ Global Minima
(Lowest amongst all minima)

(5) $f(x) = \log(1+\exp(ax))$ $\Rightarrow$ similar to Logistic Loss

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

Solving this is not trivial
We should use Gradient descent

35.4 Vector differentiation

Till now, we had assumed
x: scalar.
what if x: vector → thats only way to operate in high dimensional space.

$$\frac{df}{dx} = \nabla_x f$$ grad/del

vector

$$\nabla_x f = \begin{bmatrix} \frac{df}{dx_1} \\ \frac{df}{dx_2} \\ \vdots \\ \frac{df}{dx_d} \end{bmatrix}$$

differentiation of a vector is also a vector

written as

$$\frac{df}{dx_1} = \frac{\partial f}{\partial x_1}$$

since its partial differentiation

$$\therefore \nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix} \in \mathbb{R}^d$$

$$f(x) = y = a^T x = \sum_{i=1}^{d} a_i x_i$$

$$= a_1 x_1 + a_2 x_2 + \cdots a_n x_n$$

$\frac{\partial f}{\partial x_1} = a_1$; $\frac{\partial f}{\partial x_2} = a_2$;

$$\therefore \nabla_x f = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \vdots \\ \partial f/\partial x_d \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix} = a \Rightarrow \text{Vector}$$

② Consider our logistic loss

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

We know that $\langle x_i, y_i \rangle \rightarrow$ Constants because they are obtained from D_{Train}

Variable here is w

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

for simplicity, we considered $w^T w = w^2$

$\therefore \frac{d}{dw}(w^2) = 2w \rightarrow$ Remember w is a vector

We will solve it using Gradient Descent

35.5 Gradient descent algo

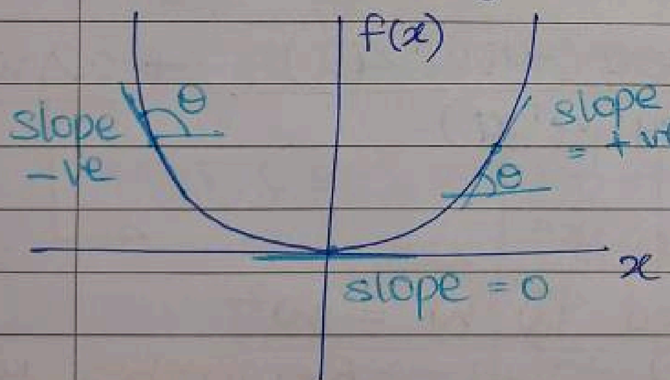
when solving these $\frac{df}{dx} = 0$; $\nabla_x f = 0$
equa^ns are not easy
then we prefer Gradient descent algo.

Its an iterative algo.

1. We randomly select $x_0 \rightarrow$ first guess x^* ↑ Minima/Maxima
2. Thats iteration 1 → select x_1
3. Thats iteration 2 → select x_2
4. Then iteration K → select x_K

we want to move closer towards x^*

Understanding Geometrically



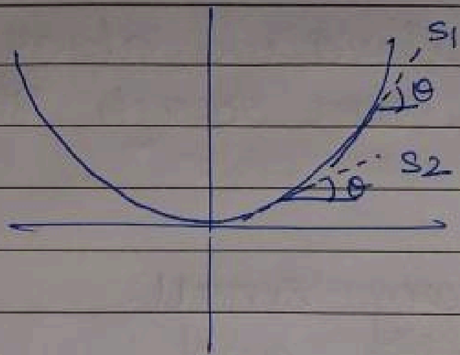
$$x^* = \arg\min [f(x)]$$

$$x^* = \arg\max [-f(x)]$$

To find Minima;
oneside :- slope :- +ve
otherside :- slope :- -ve
At minima : slope :- 0

1. Slope changes its sign from positive to Negative at Minima

(2)



slope(s1) > slope(s2)
slope reduces as we move towards minima

So, to summarize:

(1) Pick an initial point $= x_0$ at random

(2) Let $x_1 = x_0 - r\left[\frac{df}{dx}\right]_{x_0}$

step size $= 1$ (Typically)

Consider the slope at $S_1 : \rightarrow +ve$

$\therefore$

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

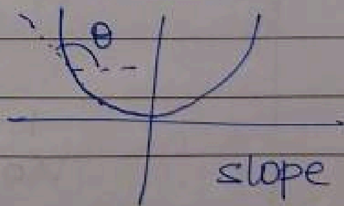
So x_1 is reducing and moving closer to 0.

If my x_0 was on other side

$\therefore x_1 = x_0 - r\left[\frac{df}{dx}\right]$

$x_1 = x_0 - r(-ve\ value)$

Hence x_1 will move close to 0.



slope will have -ve value

(3) Let $x_2 = x_1 - r\left[\frac{df}{dx}\right]_{x_1}$

I will get x_2 closer to x^*

This is a simple iterative algo.
Going on foward, we reach x_K such that

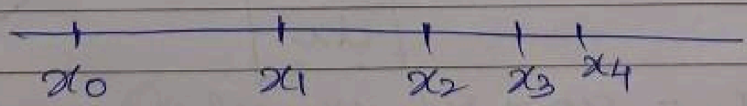
$(x_{K+1} - x_K)$ is very small
then terminate algo and finalize $\underline{x_K = x^*}$

If x is a vector then we can replace $\left[\frac{df}{dx}\right]$ with ∇_x

~~Pla~~ Please Note :

$$\left(\frac{df}{dx}\right)_{x_0} \geqslant \left(\frac{df}{dx}\right)_{x_1} \geqslant \left(\frac{df}{dx}\right)_{x_2} \cdots$$

In first iteration, we would make a big jump from $x_0 \rightarrow x_1$.
The difference would slowly reduce



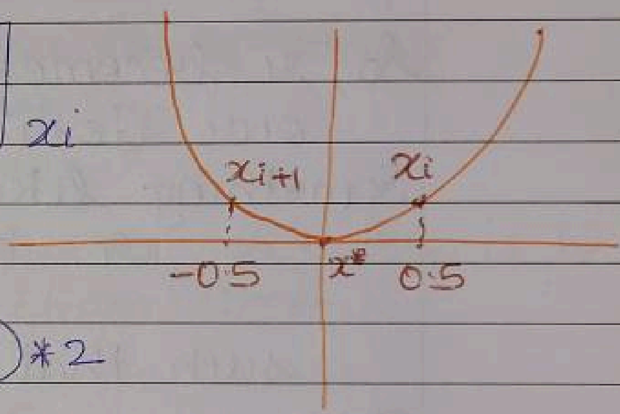
difference reduces gradually

35.6 Learning Rate

$$x_i = x_{i-1} - \underbrace{r}_{\text{Step Size}} * \left(\frac{df}{dx}\right)_{x_{i-1}} \Rightarrow \text{Update Equation}$$

Lets say $f(x) = x^2$
and x_i at a particular point $= 0.5$.

$$x_{i+1} = x_i - r * \left[\frac{df}{dx}\right]_{x_i}$$



Now $\frac{df}{dx} = 2x$

$\therefore x_{i+1} = 0.5 - (1)*(0.5)*2$

$x_{i+1} = -0.5$

Here we went to other side rather than reaching x^*.
While jumping from x_i to x_{i+1};
I jumped over x^*

$$x_{i+2} = x_{i+1} - r\left[\frac{df}{dx}\right]_{x_{i+1}}$$

$$= (-0.5) - (1)\left[2*(-0.5)\right]$$

$x_{i+2} = 0.5$

So, x_{i+2} came back to 0.5.
Illy x_{i+3} would be again -0.5,
So, I would keep $\boxed{\text{oscillating}}$ between 0.5 & -0.5

This is because 'r' has been kept constant. Due to this, I would never converge towards x^*.

Remedy for this problem:
(oscillation)

⇒ change $\boxed{r}$ with each iteration
⇒ one technique: reduce r with each iteration

So, r becomes a function of our iteration.
Something like
$$r = h(i)$$
↑ iteration

such that as ~~r↑~~ $i\uparrow$; $r\downarrow$

Then, we will eventually converge to right solution.

When we learn Deep Learning, we would know how to modify r more effectively.

35.7 Gradient descent for Linear Regression

Linear Regression:

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

not using intercept w_0

not using Regularizaⁿ term.

for simplicity

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

Here (x_i, y_i) are constants; variable is w

$\langle x_i, y_i \rangle$ we get from D_{Train}

We have to solve

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

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

②

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

r will change slightly

③

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

I will compute $w_0, w_1, w_2 \ldots w_K$ as long as w doesnt change much

If $(w_{k+1} - w_k)$ = very small value then we stop there

There is one problem with this:

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

Every time I go from w_{j-1} to w_j, I have to compute summation over all n; In M.L. problems, n may traverse in Millions.
Computing the summation requires us to go through whole dataset.
It takes a lot of time if n is large (which is not uncommon in ML)

Hack around it: Gradient descent is converted to
Stochiastic Gradient descent

35.8 Stochiastic Gradient Descent (SGD)

It says instead of using all 'n' points, use only K points ($K \leq n$)
We can pick a random set of K points

If we do itera^n with K points, the solution we get with Gradient descent is same as that with Stochiastic G.D.

$$w^*_{G.D.} = w^*_{S.G.D}$$

Stochiastic = Probabelistic = Randomizaⁿ·
(Randomly select K)

One observation

Gradient Descent : $\overset{n}{\cancel{k}} = 1M$; require 100 Iteration to converge to x^*

Stochiastic G·D ; K = 1000 ; require 500 Iteration
〃 ; K = 10 ; require 1000 〃

Iterations increase as K ↓

K = Random points we pick at each iteration

Remember: K should be diff for every iteration.

K is often called as Batch Size since we pick batch of random points

Often times K = 1 ⇒ called SGD
When $1 < K \ll n$ ⇒ 〃 Batch SGD with Batch Size K

35.9 Constrained Optimizaⁿ and PCA

Formulation of PCA : $\underbrace{\max_{u} \frac{1}{n} \sum_{i=1}^{n} (u^T x_i)^2}$

such that $u^T u = 1$

This is similar to $\max_{x} f(x)$

such that $g(x) = c$

The function which we want to maximize is called as Objective function; any condition on it is called as Constraint.

PCA : $\max_{u} \boxed{\frac{1}{n} \sum_{i=1}^{n} (u^T x_i)^2}$ objective funcⁿ

such that $\underline{u^T u = 1}$
constraint

The optimizaⁿ problem with constraint are called Constrained optimizaⁿ prob

General constraint optimizaⁿ

$\max_{x} f(x) \longrightarrow$ objective funcⁿ

such that $g(x) = c \rightarrow$ Equality constraint
and $h(x) \geq d \rightarrow$ Inequality constraint

We are not putting $\leq$ constraint because we can easily convert it to $\geq$ constraint
i.e. $K(x) \leq e \implies -K(x) \geq -e$

How to solve this problem?

We will modify the problem using Lagrangian Multiplier

So,

$$\max_{x} f(x) \text{ such that } g(x) = c \text{ and } h(x) \geq d$$

$$\Downarrow$$

$$\mathcal{L}(x, \lambda, u) = f(x) - \lambda[g(x) - c] - u[d - h(x)]$$

λ & u :- Lagrangian Multipliers
and $\lambda \geq 0$; $u \geq 0$

To optimize this; we have a method

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

By solving these three equations, we get value of $x = \tilde{x}$, this value is same as $\max x = x^*$

Proof of this is beyond scope of syllabus

PCA :

same as

$$\max_{U} \frac{1}{n}\sum_{i=1}^{n}(U^T x_i)^2 \quad \text{such that } U^T U = 1 \quad \Big\} \quad \max_{U} U^T S U \quad \text{such that } U^T U = 1$$

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

Covariance Matrix of X

PCA prob

∴ PCA prob : $\max_{u} U^T S U$

such that $U^T U = 1$

Please note that S is constant because S = f(xi) = Covariance Matrix of X

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

$$\frac{\partial \mathcal{L}}{\partial U} = 0 \Rightarrow \frac{\partial}{\partial U}\left[U^T S U - \lambda U^T U - \lambda\right] = 0$$

$$\therefore SU - \lambda U = 0$$

$$\therefore SU = \lambda U$$

We have seen this equaⁿ

$SU = \lambda U$ → definition of Eigen Value and Eigen Vector

Covariance Variance

Vector

U = Eigen Vector of S

λ = Eigen Value of S

When we learnt PCA, we wrote the optimization problem, we had said the solution to this would be to take top Eigen Values and Eigen Vectors of Covariance Matrix S.

The best v we get is the one with highest value of λ (Eigen Value)

35.10 Logistic Regression Revisited

$$w^* = \underset{w}{\arg\min}\left[(\text{logistic Loss}) + \lambda w^T w\right]$$

We would try to give an interpretaⁿ that $\lambda w^T w$ is similar to Lagrange Multiplier.

We have to find w^* which minimizes our logistic loss such that $w \perp \lambda$.

So, we rewrite the optimizaⁿ problem:

$$w^* = \underset{w}{\arg\min}\,(\text{logistic-Loss}) \longrightarrow \text{objective function}$$

$$\text{such that } w^T w = 1 \longrightarrow \text{equality constraint}$$

This is constrained optimizaⁿ problem

We solve this through Lagrange Multiplier

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

can be also written as $+\lambda(w^T w - 1)$

$$= \underline{(\text{logistic Loss})} - \lambda + \underline{\lambda w^T w}.$$

If we see, these two terms are same as our optimization problem

Regularizaⁿ can be thought of as an equality constraint as a solution through Lagrangian Multipliers.

35.11 <u>Why does L1 Regularization create sparsity?</u>

We know that

<u>Logistic Regression</u> → L1 regularization creates sparsity in w as compared to L2 Regularization

<u>Sparsity</u> : $w = \langle w_1, w_2 \dots\dots w_d \rangle$
More of the $w_i = 0$

L2 Regularizaⁿ | L1 Regularizaⁿ

$$\min_{w}\left[\text{loss} + \lambda\|w\|_2^2\right] \qquad \min\left[\text{loss} + \lambda\|w\|_1\right]$$

since the loss is present on both the terms, we can ignore it for now

Similarly λ is a hyper parameter; but its a fixed value; hence its can be ignored as well.

Hence its equivalent to considering L1 & L2 Reg

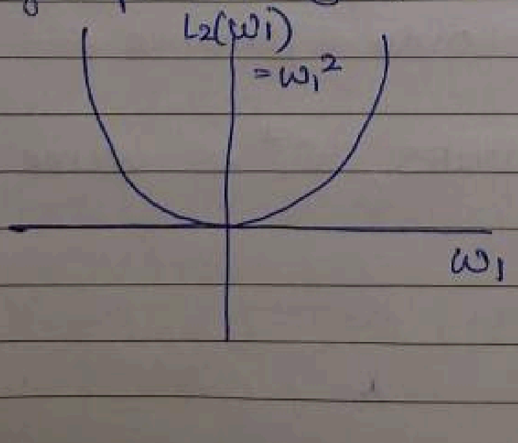
$$\underset{w_1, w_2 \cdots w_d}{\text{Minimize}}\left[w_1^2 + w_2^2 + \cdots + w_d^2\right] \qquad \underset{w_1, w_2 \cdots w_d}{\text{Minimize}}\left[|w_1| + |w_2| + \cdots + |w_d|\right]$$

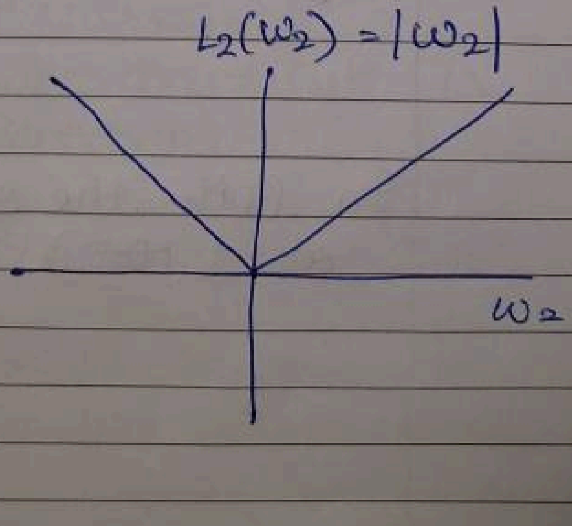
So we see that w_1 occurs only in 1 term $\Longrightarrow$ same here. So lets see from w_1 perspective

From w_1 perspective It would be

$\min(w_1^2)$ | $\min(|w_1|)$

Graphically seeing





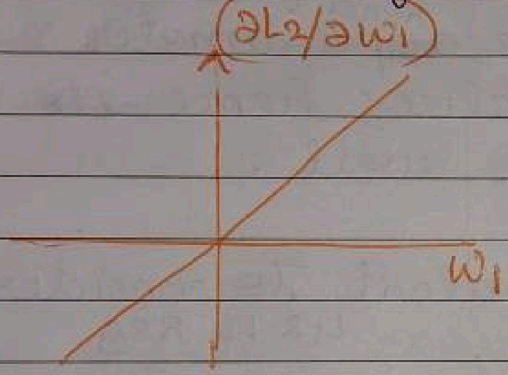
$$\mathcal{L}_2(w_1) = w_1^2$$

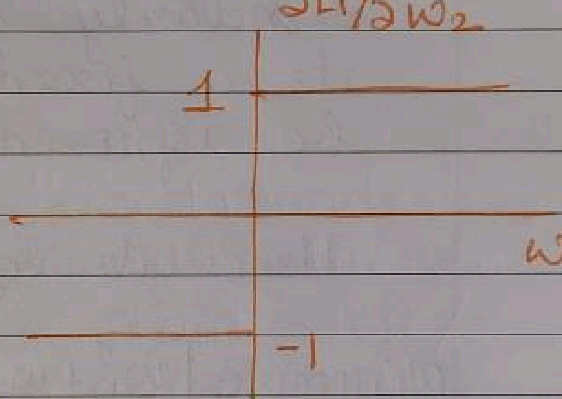
$$\mathcal{L}_1(w_1) = |w_1| = \begin{cases} w_1 & \text{if } w_1 > 0 \\ -w_1 & \text{if } w_1 < 0 \end{cases}$$

$$\frac{\partial L_2}{\partial w_1} = 2 * w_1$$

$$\frac{\partial L_1}{\partial w_1} = \begin{cases} +1 & ; \text{ if } w_1 > 0 \\ -1 & ; \text{ if } w_1 < 0 \end{cases}$$

Derivative function will look like





As a part of Gradient descent

$$(w_1)_{j+1} = (w_1)_j - r\left[\frac{\partial L_2}{\partial w_1}\right]_{(w_1)_j}, \quad (w_1)_{j+1} = (w_1)_j - r\left[\frac{\partial L_1}{\partial w_1}\right]_{(w_1)_j}$$

Let $(w_1)_j$ be a +ve ~~posit~~ value.

$$(w_1)_{j+1} = (w_1)_j - r\boxed{(2 * w_j)}, \quad (w_1)_{j+1} = (w_1)_j - r\boxed{(1)}$$

only difference

In both the scenarios, w_1^* is same, i.e. 0

~~at~~ & ~~(0,0)~~

L2	L1

Assume that at i^{th} iterations

$(w_1)_j = 0.05$ (very small)

Lets say r is a small value (~~say r=1~~)
Due to this
$(2 * w_j) \rightarrow$ small
If r is very small then
$r * 2 * w_j \rightarrow$ very small

Lets say $r = 0.01$ & $w_{ij} = 0.5$
∴ The substraction term would be

L2	L1
$r(2 * w_j) = 0.01 \times 2 \times 0.5$ $= \underline{0.001}$	$r = 0.01$
$w_{1(j+1)} = \underbrace{0.05 - 0.001}_{w_{ij}}$	$w_{1(j+1)} = 0.05 - 0.01$
Here as the iteration increases the derivative changes and $w_{1(j+1)}$ comes closes to w^* derivative becomes smaller	The derivative is constant
Speed of convergence decreases with every iteraⁿ	Speed of convergence is same everytime

L2	L1
Since the derivative ~~is~~ becomes smaller with every iteration, L2 Regularizer doesnt change the value of w_1 much, from one iteration to other	Since the derivative is constant, L1 Regularizer continues to constantly reduce the value of w_1 towards 0.
	Hence L1 creates sparsity due to constant slope/gradient
chance is Less $\Leftarrow$	Hence, the chance that $w_1 = 0$ at end of K itera^n are MORE
	L1 will remove more of useless feature by making their weight to 0