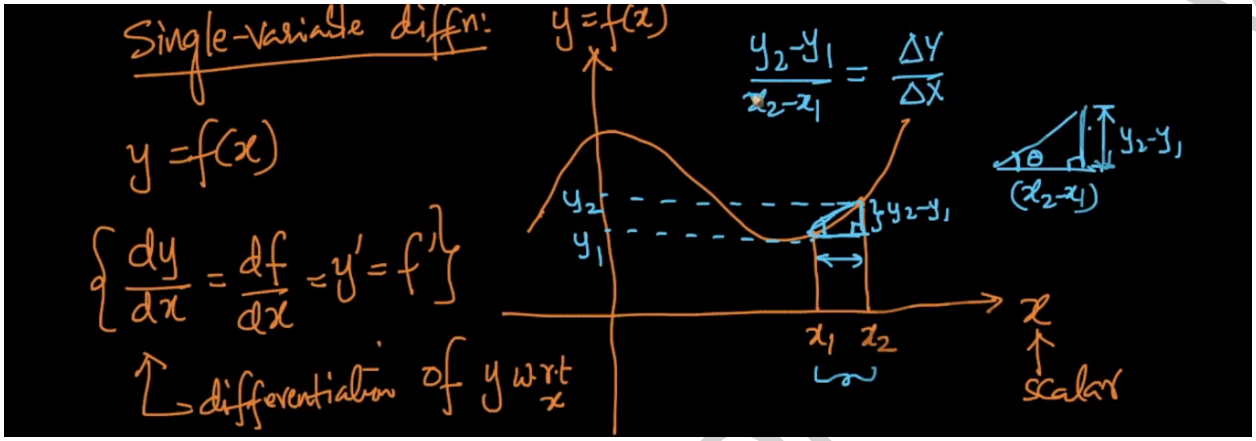
35.1 - Differentiation

Single Variable Differentiation

Let us consider a function y = f(x) as shown in the figure below.

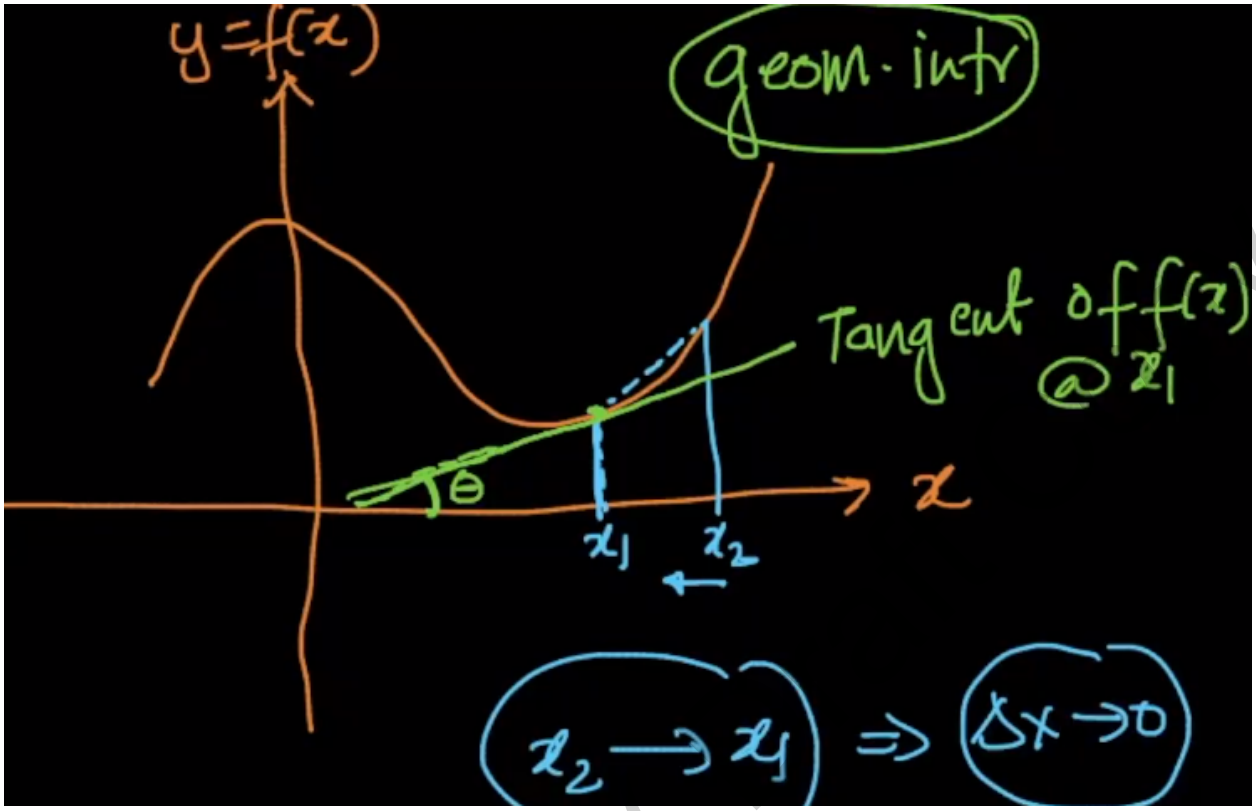


The derivative of this function is denoted as dy/dx (or) df/dx (or) $y^{|}$ (or) $f^{|}$. It is pronounced as differentiation of 'y' with respect to 'x'.
dy/dx → rate of change of 'y' with respect to 'x'.(ie., it indicates how much 'y' changes, as 'x' changes)

$dy/dx = (y_2-y_1)/(x_2-x_1) = \Delta y/\Delta x = \tan\theta$
$dy/dx = \text{Lim}_{\Delta x \to 0}\ \Delta y/\Delta x$ (Here $\Delta x \to 0$ means change in 'x' is very small)

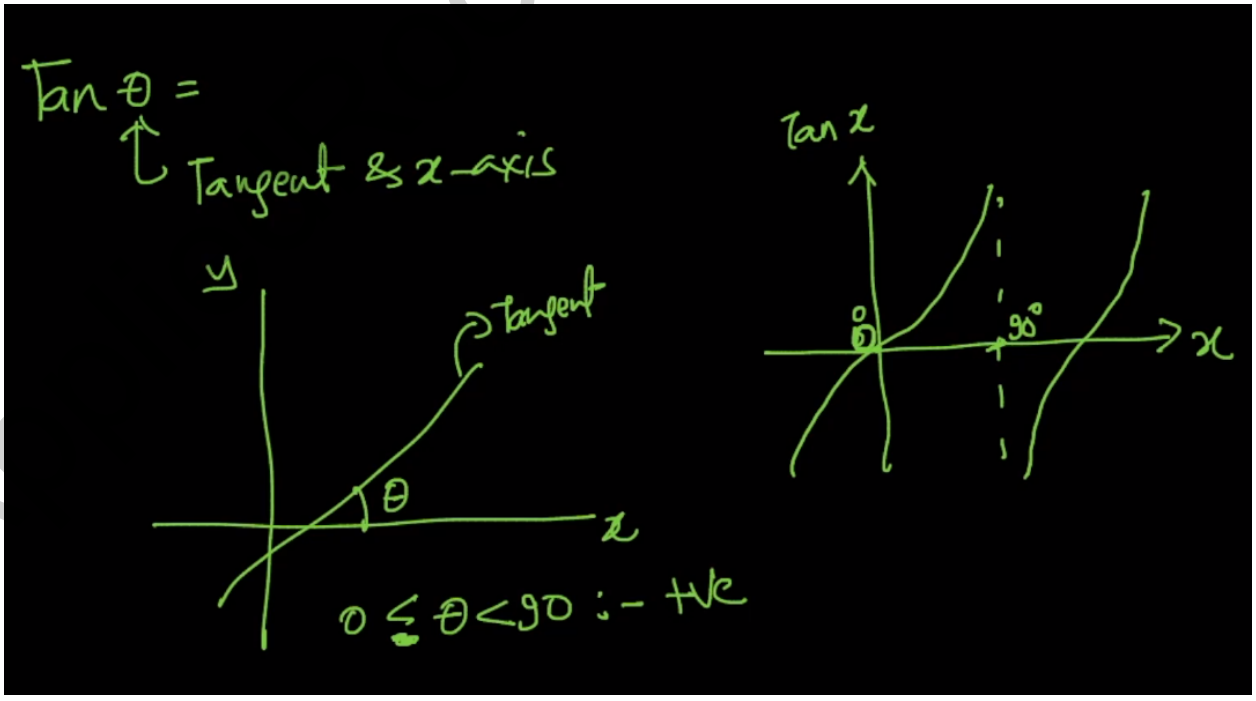
Here 'θ' is the angle made by the tangent to the function, with the 'X' axis.'A tangent is the hypotenuse that we obtain as $\Delta x \to 0$.

dy/dx = slope of the tangent to f(x)
$[dy/dx]_{x=x1}$ = slope of the tangent to f(x) at $x=x_1$



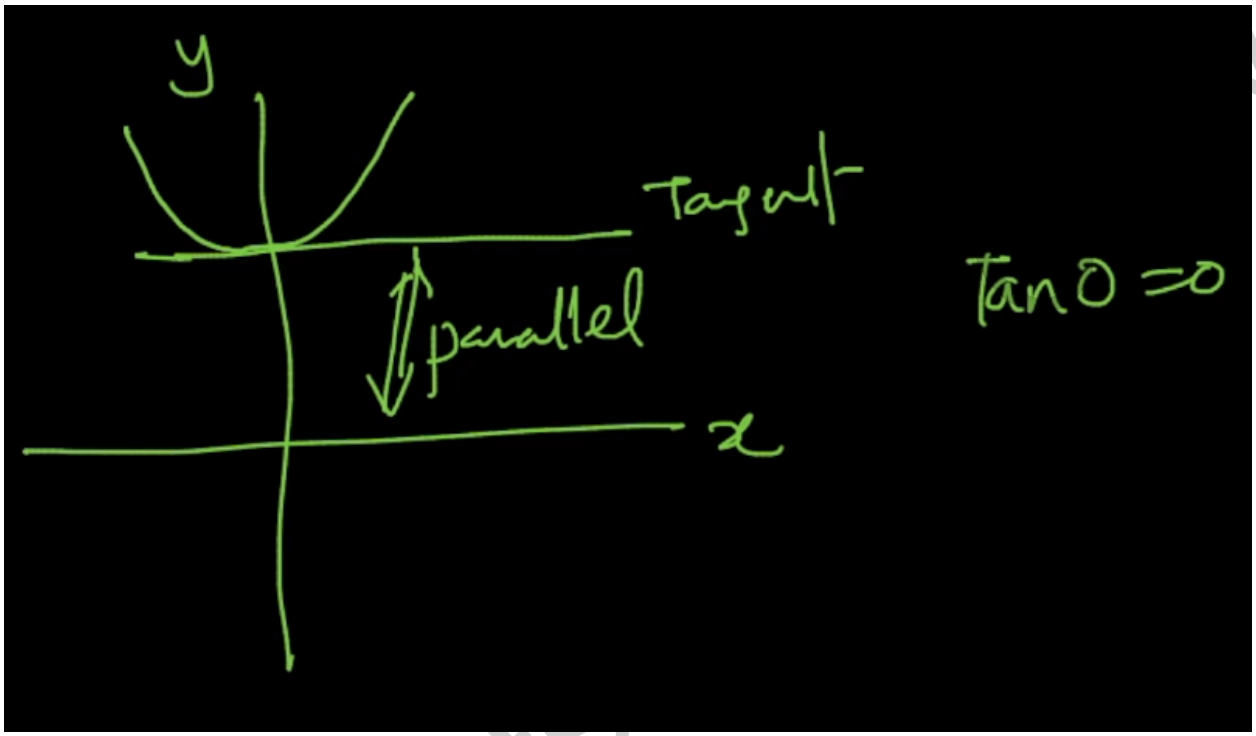
Slopes at different values of 'θ'

Case 1



If the value of ‘θ’ lies in between 0 and 90^{o}, then the value of **tanθ** is positive.

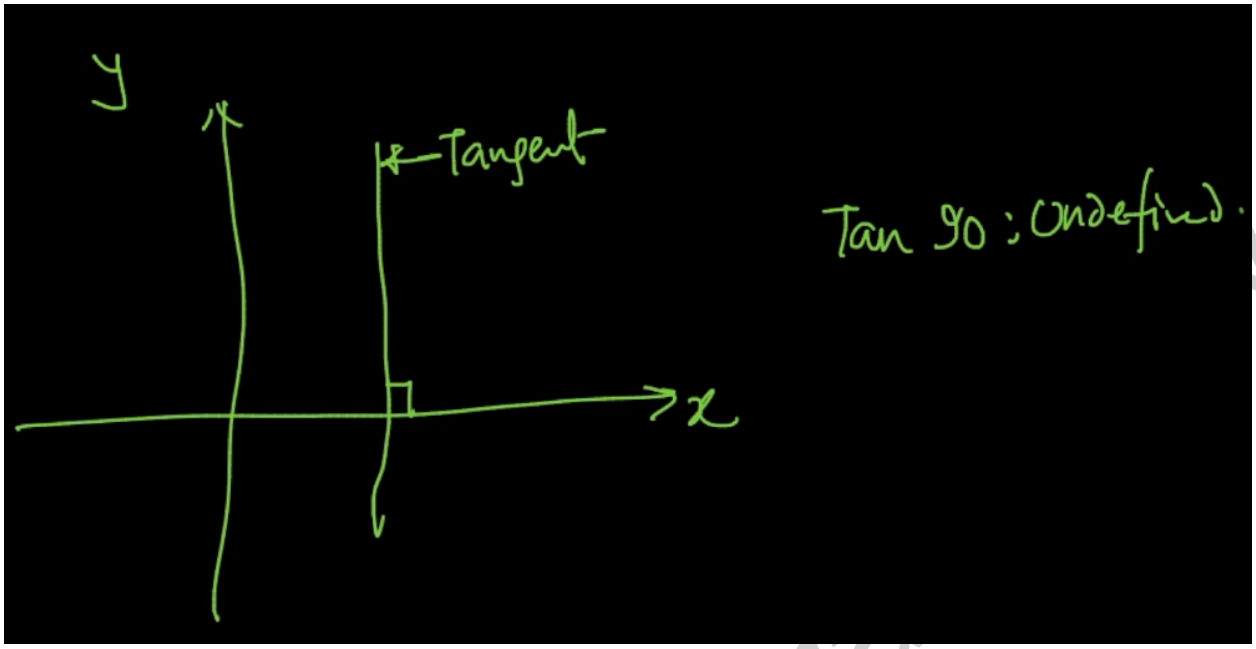
Case 2



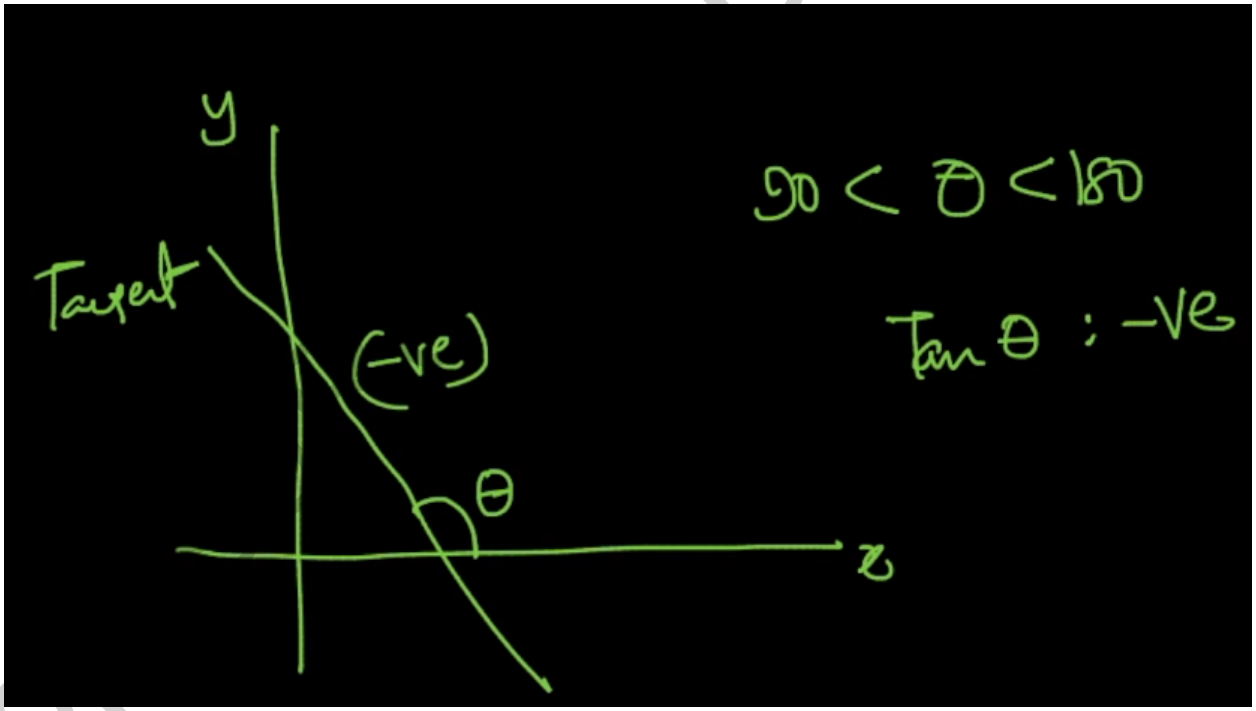
If the tangent is parallel to the ‘X’ axis, then the angle made by the tangent with the ‘X’ axis is 0. In such a case, the slope is always equal to 0.
Slope = tanθ = tan(0) = 0.

Case 3

If the tangent is parallel to the ‘Y’ axis, then the angle made by the tangent with the ‘X’ axis is 90^{o}. In such a case, the slope is always undefined.
Slope = tanθ = tan(90) = undefined.



Case 4



If the angle 'θ' lies in between 90° and 180°, then the slope is always negative.

Slope = tanθ is negative.

Basics of Differentiation

1) $d/dx(x^n) = n.x^{n-1}$
2) $d/dx(c) = 0$
3) $d/dx(cx^n) = c.n.x^{n-1}$
4) $d/dx(\log(x)) = 1/x$
5) $d/dx(e^x) = e^x$
6) $d/dx(f(x)+g(x)) = d/dx(f(x)) + d/dx(g(x))$
7) $d/dx(f(g(x)) = df/dg.\ dg/dx$

Example

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

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

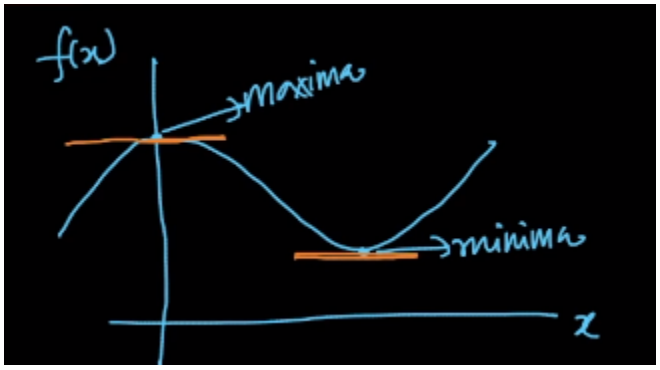
$d/dx\ f(g(x)) = df/dg.\ dg/dx$

$= -2b(a-bx)$

35.3 - Maxima and Minima

The largest value a function/variable can give is called Maxima and the smallest value is called Minima.

Let us look at the function 'f' given below.



The maxima and minima values of f(x) are clearly shown in the figure above.

Both at maxima and minima, the slope of the function = 0.

For example, let us consider a function $f(x) = x^2-3x+2$

$df/dx = 0 \Rightarrow$ slope = 0

As we are differentiating with respect to 'x', the slope is equal to 0, for both maxima and minima.

So $df/dx = 2x-3 = 0 \Rightarrow x = 3/2$

Now we have to find whether f(x) is having a minima or maxima at x=3/2.

The function is $f(x) = x^2-3x+2$

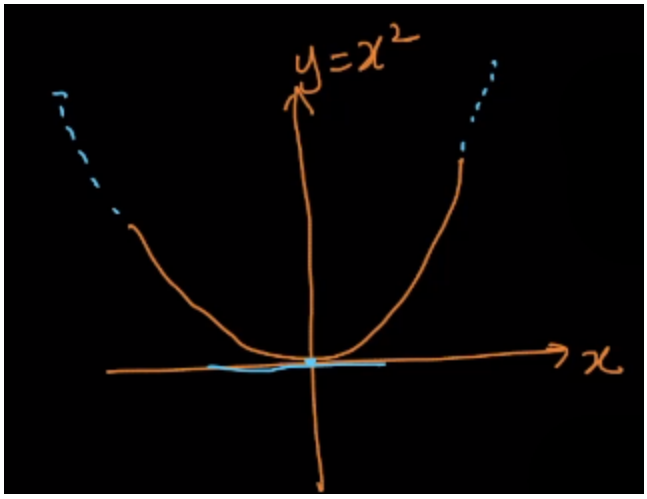
Let us compute the value of f(3/2) = -0.25

Let us now compute f(1), we get f(1) = 0 (here just for the sake of an example, we are computing f(1))

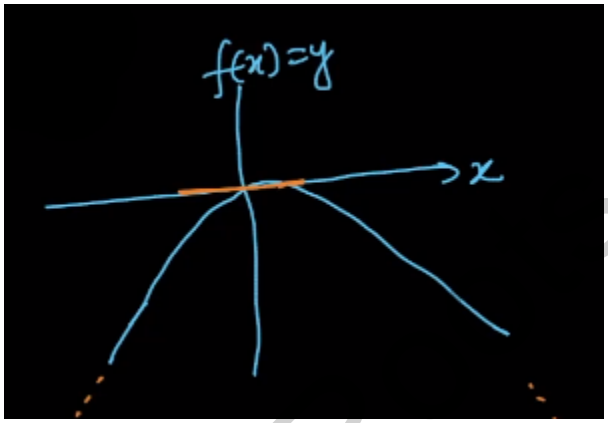
As f(1.5)<f(1), f(1.5) cannot be maxima.

So we can conclude that the minimum of f(x) occurs at x=3/2.

Now let us consider the example of a parabola.

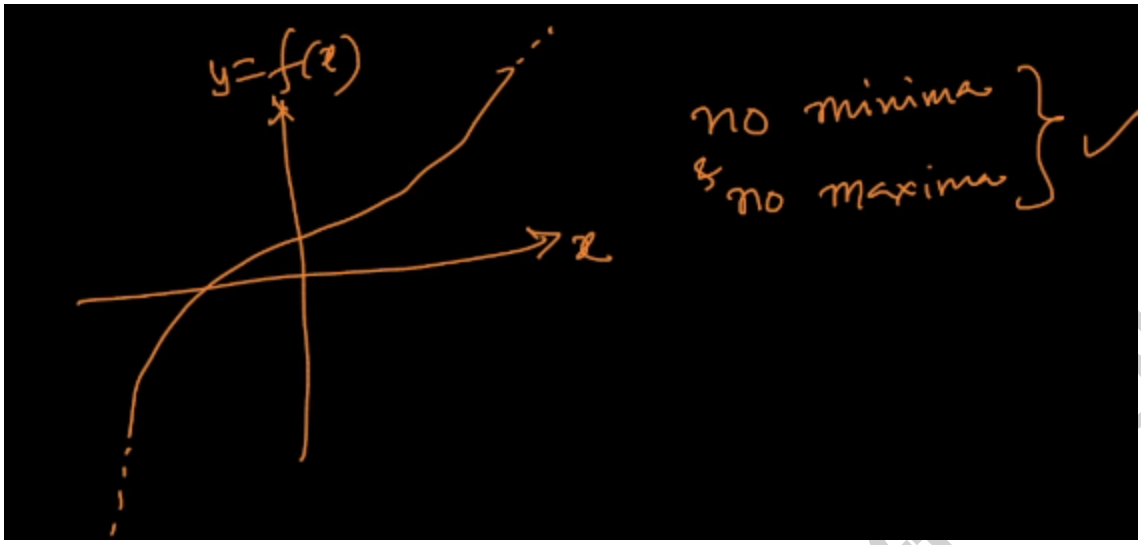


In this parabolic function, we can see the minimum value occurs at x=0, and there is no maximum value at all. Hence we can say the function $y = x^2$ has a minima, but not maxima.



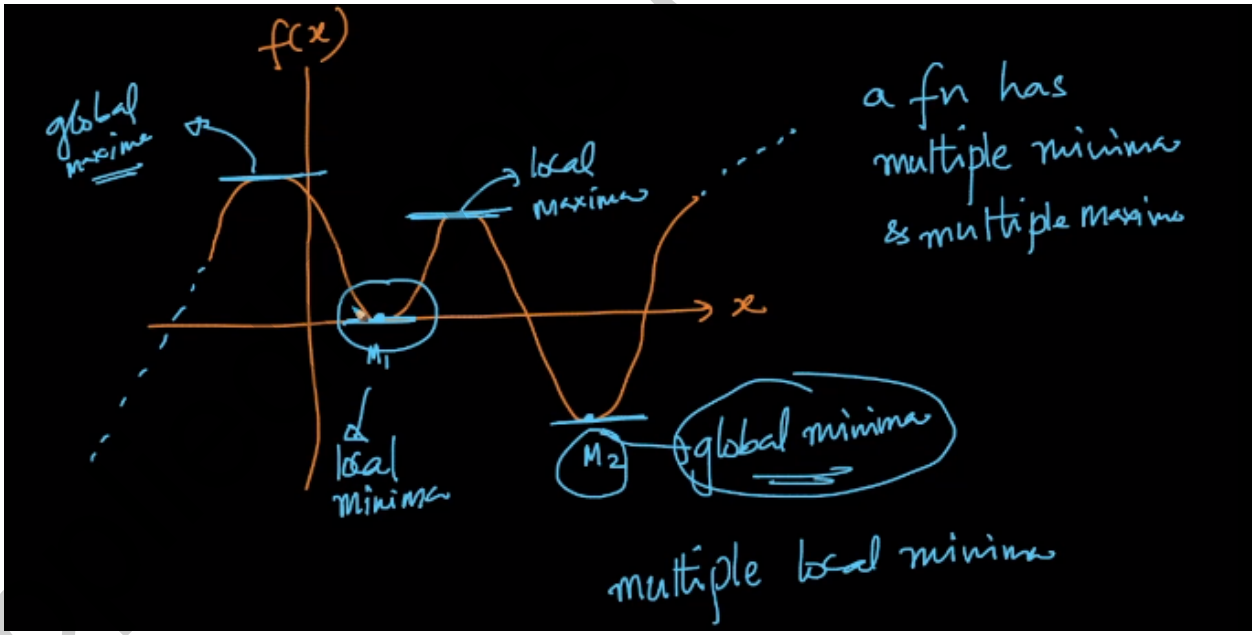
In this parabolic function, we can see the maximum value occurring at x=0, and there is no minimum value at all. Hence we can say the function $y=-x^2$ has a maxima, but not minima.

There are a few functions which neither have a maxima nor a minima. The best examples for such a category of functions are the monotonic functions.



In this example, we can see there is no maxima and no minima. Such functions do not converge at all in the optimization problems.

There is another category of functions which have multiple maxima and minima. The best examples for such a category are the trigonometric functions.



Here we can see multiple minima and maxima. A function can have multiple local minima/maxima, but can have only one global minima/maxima.

For simple functions with only one variable, we were able to find out the maxima/minima values using the differentiation approach.

In case, if we come across complex functions (say **log(1+exp(ax)** which is of the form of logistic loss), then it is really difficult to find the maxima/minima using the single differentiation.

For such complex functions, we need to repeat the differentiation process for a certain number of times, in order to arrive at the maxima/minima. For such a process, we use an algorithm called **Gradient Descent**.

35.4 - Vector Calculus: Grad

So far we have worked with 'x' as a scalar, and f(x) as a function of the scalar 'x'.

Let us now assume 'x' as a vector because in order to operate in high dimensional space, 'x' needs to be a vector.

Example:

Let $y = a^T x$

$x = \langle x_1, x_2, x_3, \ldots\ldots, x_d \rangle$

$a = \langle a_1, a_2, a_3, \ldots\ldots, a_d \rangle$

$\mathbf{y = \Sigma_{i=1}^{d} a_i x_i}$

Here $\mathbf{df/dx = \nabla_x f}$ (This notation indicates that 'x' is a vector and 'f' is a function on vector 'x'. The '∇' symbol is pronounced as **Del** (or) **Grad**.)

$$\nabla_x \mathbf{f} = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \partial f/\partial x_3 \\ . \\ . \\ . \\ \partial f/\partial x_d \end{bmatrix}$$

If we have a function f(x) as

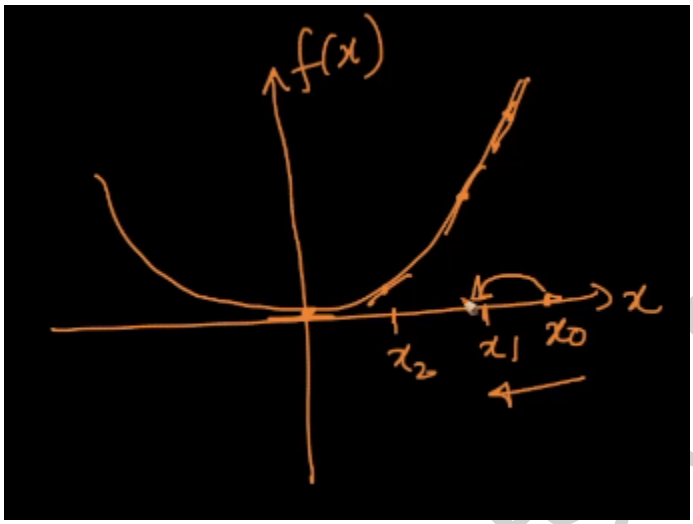
$\mathbf{f(x) = a_1x_1 + a_2x_2 + a_3x_3 + \ldots\ldots + a_dx_d}$ then

$$\nabla_x \mathbf{f} = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \partial f/\partial x_3 \\ . \\ . \\ . \\ \partial f/\partial x_d \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ . \\ . \\ . \\ a_d \end{bmatrix}$$

35.5 - Gradient Descent: Geometric Intuition

Gradient Descent is an iterative algorithm used to find maxima/minima points of a function which is highly complex.

In some problems like Logistic Regression optimization problem, in order to compute the maxima/minima, solving **df/dx = 0(using scalars**) or ∇_x**f = 0(using vectors)** is not straight to find the optimal maxima/minima values. For solving such problems, we use Gradient Descent.



Pick some value 'x_0' as the first guess of x*.
In iteration 1, we get 'x_1'.
In iteration 2, we get 'x_2'.
In iteration 3, we get 'x_3'.
.
.
.
In iteration 'k', we get 'x_k'.

The objective here is as the iteration number increases, the value of 'x' becomes closer and closer to the optimal 'x*'.

Let us consider the optimization problem

x* = arg-minx f(x)

Here **min(f(x)) = max(-f(x))**

max(f(x)) = min(-f(x))

The slope changes its sign from +ve to -ve at minima. As we approach our minima, the slope decreases. As we move away from minima, the slope increases.

Procedure for Gradient Descent

1) Pick an initial point 'x_0' at random.
2) So now pick 'x_1' such that $\mathbf{x_1 = x_0 - r*[df/dx]_{x0}}$ (where r→ learning rate (or) step size)
   Let **r=1**, then $\mathbf{x_1 = x_0 - [df/dx]_{x0}}$
   Initially as the slope is positive, $\mathbf{[df/dx]_{x0} > 0}$
   So $x_1 = x_0$ - (+ve value) ⇒ $x_1 = x_0$ - slope ⇒ $x_1 < x_0$
   Let us assume 'x_0' is on the other side. So now,
   $x_1 = x_0$ - (-ve value) ⇒ $x_1 = x_0$ + slope ⇒ $x_1 > x_0$
3) $\mathbf{x_2 = x_1 - r*[df/dx]_{x1}}$.
   So at any iteration, $\mathbf{x_{i+1} = x_i - r*[df/dx]_{x1}}$
   Similarly we have to compute $x_0, x_1, x_2, \ldots, x_k, x_{k+1}$ where
   $\mathbf{x_k = x_{k-1} - r*[df/dx]_{x1}}$

   If $(x_k - x_{k-1})$ is very very small, then we have to terminate the loop and declare $x^* = x_k$.

   So incase, if w are working on multi-dimensional data using vectors, then

   $\mathbf{x_k = x_{k-1} - r[\nabla_x f]}$

   At every iteration, the slope decreases, as we move towards the minima.
   ie., $[df/dx]_{x0} > [df/dx]_{x1} > [df/dx]_{x2} > \ldots\ldots > [df/dx]_{xk}$

35.6 - Learning Rate

In Gradient Descent, the update equation is given as

$x_i = x_{i-1} - r[df/dx]_{x(i-1)}$

Here 'r' → Learning rate (or) step size

So far we have assumed 'r' to be constant.

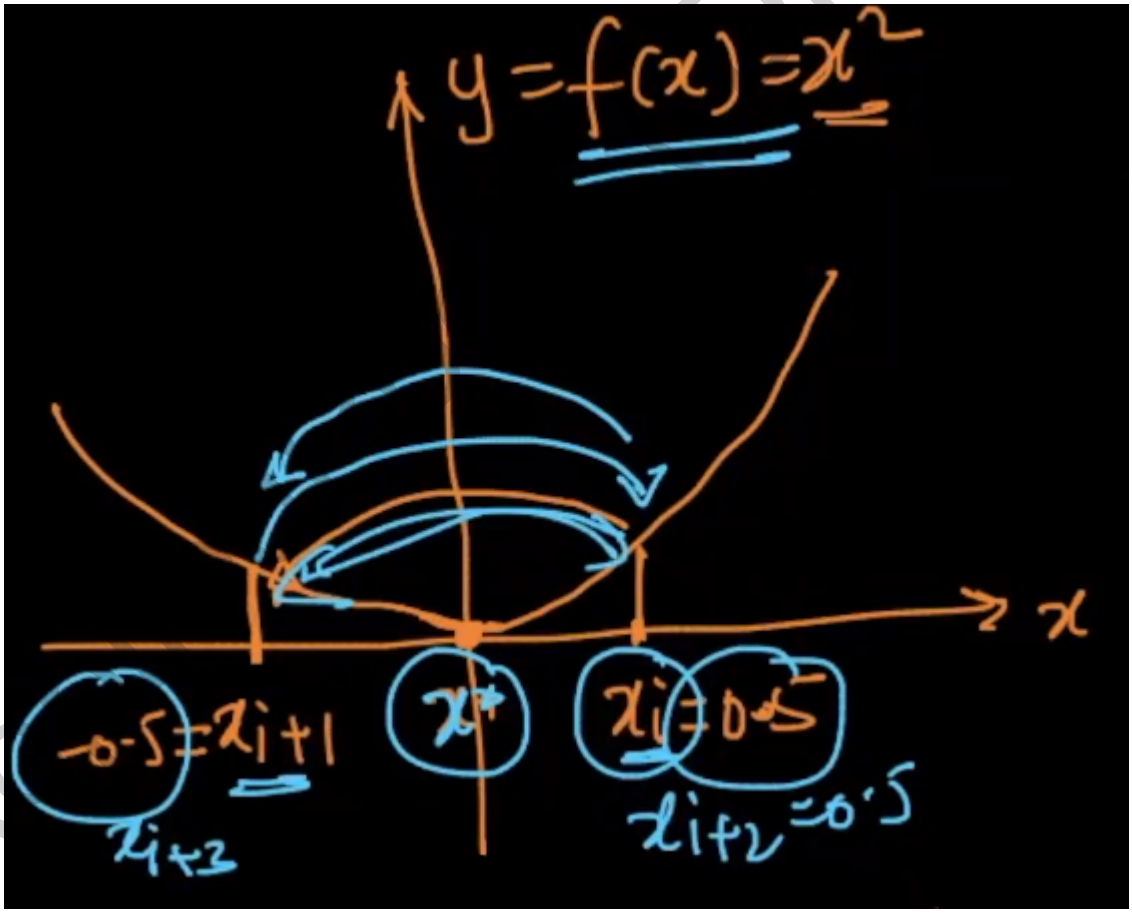
Let us assume a function $f(x) = x^2$. Let us assume after a few iterations, we reach at $x_i=0.5$.

$x_{i+1} = x_i - r*[df/dx]_{xi}$

$f(x) = x^2$

$df/dx = 2x$

Let r = 1, then $x_{i+1} = 0.5 - 1*[2*0.5] = -0.5$



Here in the step of moving from $x_i = 0.5$ to $x_{i+1} = -0.5$, we are not moving towards minima, but have crossed the minima and have reached the other side. We simply jumped over x* on the other side.

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

In this way, we have alternate values on each side. We are oscillating between both sides. We can never converge to x*. This is called an oscillation problem.

The remedy for the oscillation problem is to change 'r' with each iteration. When we reduce the learning rate 'r', the size of the oscillation(jump) decreases and there are more chances to converge.

So now 'r' becomes a function of the number of iterations.

r = f(number of iterations)

So we can create some function r = h(i), (where i→ number of iterations) such that as 'i' increases, 'r' should reduce.

35.7 - Gradient Descent for Linear Regression

The mathematical formulation of Linear Regression with the intercept term (w_0) and without regularization is

$L(w) = w^* = \text{arg-min}_w \Sigma_{i=1}^n (y_i - w^T x_i)^2$

Here $<x_i, y_i>$ belongs to the dataset and are constants. The only thing that keeps changing in every iteration is 'w'.

So now we have to solve $\nabla_w L$.

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

1) We have to pick a /vector $w_0 = <....>$
2) $w_1 = w_0 - r*\Sigma_{i=1}^n \{2*(y_i - w_0^T x_i)(-x_i)\}$
3) $w_2 = w_1 - r*\Sigma_{i=1}^n \{2*(y_i - w_1^T x_i)(-x_i)\}$
4) $w_3 = w_2 - r*\Sigma_{i=1}^n \{2*(y_i - w_2^T x_i)(-x_i)\}$
5) .
6) .
7) .

Like this, we have to repeat the steps from 2 to the end, and compute the values of w_1, w_2, w_3, …, w_k, w_{k+1}, as long as our vector doesn't change much.

If $w_{k+1} - w_k$ is a vector of very small values, then we declare $w^* = w_k$.

So for any iteration, **$w_j = w_{j-1} - r*\Sigma_{i=1}^n \{2*(y_i - w^T x_i)(-x_i)\}$**

Here as we move from 'w_{j-1}' to 'w_j', for every iteration, if 'n' is very large, then it is computationally expensive.

Gradient Descent becomes very slow if the number of computations is large. In order to get rid of these kinds of problems, we have an alternative optimizer called **Stochastic Gradient Descent**.

35.8 - SGD Algorithm

So far we have seen the Gradient Descent algorithm in training a linear regression model. The update equation is given as

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

When 'n' is extremely large, this solution becomes computationally expensive and takes a lot of time, as the number of computations is more.

There is an algorithm to overcome this issue. It is called Stochastic Gradient Descent which is a modification done over the Gradient Descent.

Stochastic Gradient Descent says instead of choosing all the data points and perform summation over them in Gradient Descent, here we just have to pick a random set of 'K' points and perform the same update step for multiple iterations. After some iterations, the value w* obtained by SGD is the same as the w* obtained by GD.

In SGD,

$\mathbf{w_{j+1} = w_j - r*\Sigma_{i=1}^{K} (-2x_i)(y_i - w_j^T x_i)}$ (where 1<K<n)

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

Here we use a random set of points. Hence we call this variant of Gradient Descent as **Stochastic Gradient Descent**.

For example,

In Gradient Descent, if n = 1 million and if our problem requires 100 iterations to converge to x*. In Stochastic Gradient Descent, if K=1000, then for the same problem, we need more than 100(say 500) iterations to converge to x*.

K→ Number of points that are taken at each iteration for the update formula

The set of the random points chosen in each iteration are different from the sets of random points chosen in successive iterations.

Here in SGD, 'K' is called Batch Size, as we choose a batch of random points.

If K=1 → we call it simply SGD.

If K>1 → we call it Batch SGD with batch size = 100 (say)

If K=n → then SGD becomes GD.

35.9 - Constrained Optimization and PCA

So far we have seen optimization problems like $\max_x f(x)$ (or) $\min_x f(x)$.

But the formulation of PCA was

$\mathbf{max_u\ (1/n)\ \Sigma_{i=1}^{n}\ (u^Tx)^2}$ such that $u^Tu=1$

Here $\mathbf{(1/n)\ \Sigma_{i=1}^{n}\ (u^Tx)^2}$ → Objective Function, $\mathbf{u^Tu=1}$ → Constraint.

This formulation is similar to

$\max_x f(x)$ such that $g(x)=c$.

Such an optimization problems with constraints is called **Constrained Optimization Problem**.

General Constrained Optimization Problem

It looks of the form $\max_x f(x)$ → objective function

Such that $g(x)=c$ → equality constraint

$h(x)>=d$ → inequality constraint

If an inequality constraint of the form **K(x)<=e** is given, we immediately have to convert it into >= form by multiplying with '-' and making it **-K(x)>=-e**.

How to find Maxima and Minima when constraints are given

The optimization problem is of the form

$\mathbf{x^* = \max_x f(x)}$ such **that g(x)=c**, **h(x)>=d**

We shall modify this problem using Lagrangian Multipliers. Here using the given objective function, we shall construct a new function called **Lagrangian**, denoted by 'L'.

So now,

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

where λ, μ → Lagrangian Multipliers. Here $\lambda>=0$, $\mu>=0$.

We should compute parallel derivatives of 'L' with respect to 'x', 'λ' and 'μ' separately and equate them to 0.
$\partial L/\partial x = 0$, $\partial L/\partial \lambda = 0$, $\partial L/\partial \mu = 0$

Here we get 3 equations an if we solve them, we get x_tilda, λ_tilda and μ_tilda. We can ignore λ_tilda and μ_tilda for the time being, the value of x_tilda obtained here is equal to x* for the problem.

Let us now come back to the optimization problem of PCA.
$\mathbf{max_u\ (1/n)\ \Sigma_{i=1}^{n}\ (u^T x_i)^2}$ such that $\mathbf{u^T u=1}$.

The same above constrained optimization problem can be written as
$\mathbf{max_u\ u^T S u}$ such that $\mathbf{u^T u=1}$
where 'S' is the covariance matrix of 'X' with mean centred variance '1' and is given as $\mathbf{S = (1/n)\ \Sigma_{i=1}^{n}\ x_i^T x_i}$

The Lagrangian form for this function is written as
$L(u, \lambda) = u^T S u - \lambda(u^T u - 1)$
$\partial L/\partial u = 0 \Rightarrow \partial/\partial u(u^T S u - \lambda u^T u + \lambda) = 0$ --- (1)
$u^T u = u^2$ --- (2)
So now, if we substitute (2) in (1), then we get
$\partial/\partial u(Su^2 - \lambda u^2 + \lambda) = 0$
The result obtained after derivation is
$2Su - 2\lambda u = 0$
We are cancelling out the number 2. So the equation becomes
$Su = \lambda u$

Here u → eigen vector of 'S'
λ → eigen value of 'S'
'S' → Covariance Matrix

The best 'u' corresponds to the highest eigen value of 'S'.

35.10 - Logistics Regression Formulation Revisited

The optimization problem of Logistic Regression looks like

$w^* = \text{arg-min}_w$ (Logistic Loss) + $\lambda w^T w$ such that **$w^T w = 1$**

The goal here is to minimize the sum of Logistic Loss and the regularization term. Here 'w' is a vector normal to the hyperplane.

The above given optimization problem is a constrained optimization problem.

So we can convert this constrained optimization problem into non-constrained optimization problem using Lagrangian multipliers.

So L = Logistic Loss - $\lambda(1-w^T w)$

L = Logistic Loss - $\lambda + \lambda w^T w$

35.11 - Why L1 Regularization creates sparsity?

Optimization problem with L2 regularization ⇒ $\mathbf{min_w\ (loss + \lambda||w||_2^2)}$, Optimization problem with L1 regularization ⇒ $\mathbf{min_w\ (loss + \lambda||w||_1)}$.

In both these optimizations, 'λ' and the loss term are the same. So we are left with minimizing only the terms $||w||_1$ and $||w||_2^2$. So now the above two optimizations become $\mathbf{min_w\ (||w||_2^2)}$ and $\mathbf{min_w\ (||w||_1)}$ respectively.

$\mathbf{min_w\ (||w||_2^2) = min_{w1,\ w2,w3...,wd}\ (w_1^2+w_2^2+w_3^2+...+w_d^2)}$

$\mathbf{min_w\ (||w||_1) = min_{w1,\ w2,w3...,wd}\ (|w_1|+|w_2|+|w_3|+...+|w_d|)}$

Now as the above two optimizations are having each coefficient as a separate term(ie., we do not see multiple coefficients in the same term like w_1w_2, $w_1w_2w_3$,w_1/w_2, etc), let us consider the two optimizations only from 'w_1' point of view. So now the optimizations become

$\mathbf{min_w\ (||w||_2^2) = min_{w1}\ (w_1^2)}$

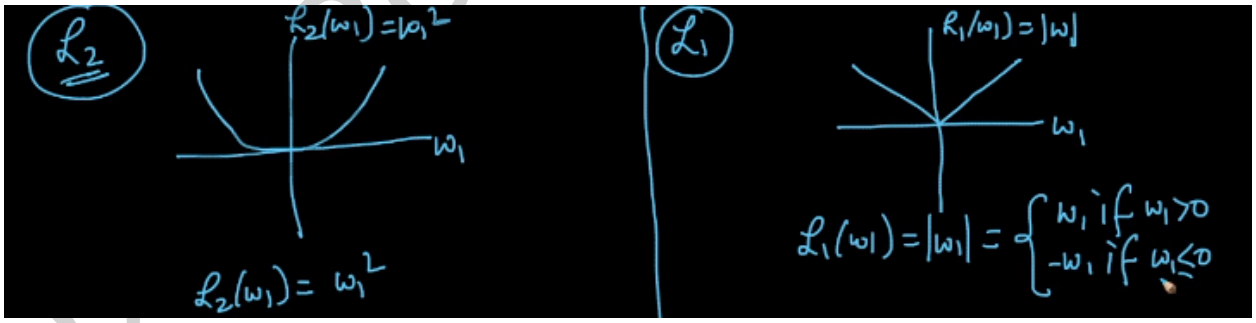
$\mathbf{min_w\ (||w||_1) = min_{w1}\ (|w_1|)}$

Let us represent these functions as

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

$\mathbf{L_1(w_1) = |w_1|}$

The graphs of these functions are given below



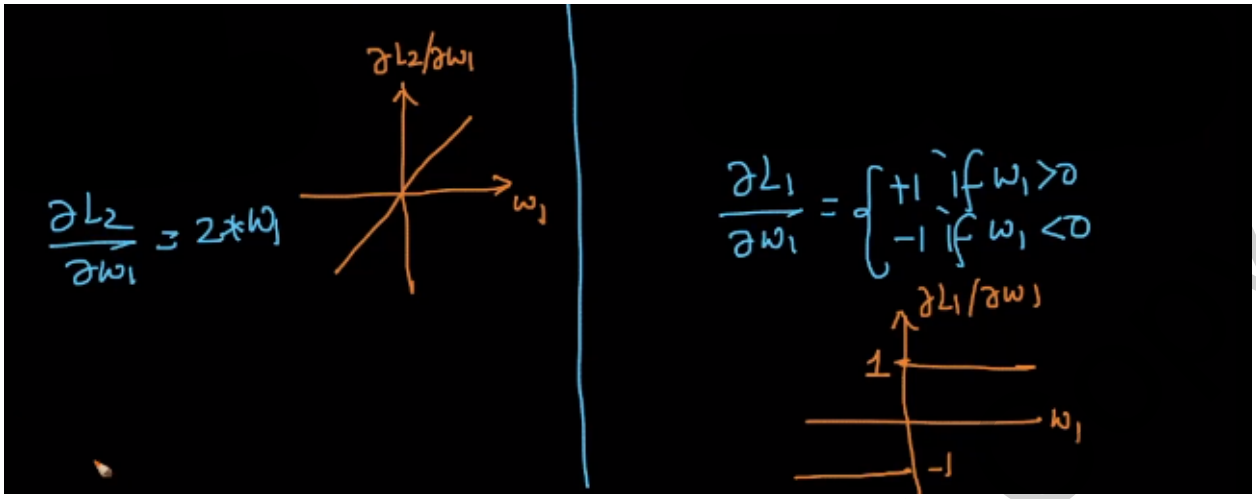
Now let us compute the derivates of the functions $L_2(w_1)$ and $L_1(w_1)$

$dL_2/dw_1 = 2*w_1$

dL_1/dw_1 = {1, if w_1>0

-1, if w_1<0}

The graphs of the derivatives of these functions look like below



Now let us look at the update step

$(w_1)_{(j+1)} = (w_1)_j - r*[df/dx]_{(w1)j}$

For the timebeing, we shall assume 'w_1' value is only positive. Here when we look at the slopes of both the functions $L_1(w_1)$ and $L_2(w_1)$,

$\partial L_2/\partial w_1 = 2*w_1$

$\partial L_1/\partial w_1 = \{1, \text{ if } w_1>0,$

$-1, \text{ if } w_1<0\}$

We can clearly observe that the slope of $L_1(w_1)$ is independent of the 'w_1' value, whereas the slope of $L_2(w_1)$ is dependent on 'w_1' value.

So when we apply the regularization, the update step becomes

$\mathbf{(w_1)_{(j+1)} = (w_1)_j - r*[2*w_1]_{(w1)j}}$ (In case of L2 regularization)

$\mathbf{(w_1)_{(j+1)} = (w_1)_j - r*[1]_{(w1)j}}$ (In case of L1 regularization)

So we see the slope of $L_2(w_1)$ is dependent on 'w_1', if the value of 'w_1' is very small, then the slope will be a very small value, and the product of 'r' with the slope will become much more smaller, taking smaller steps to move towards the minima, there by taking a huge number of iterations.

Whereas in the case of $L_1(w_1)$, as it's slope is independent of 'w_1', it takes a constant step size in the weight update step and thereby converging faster. In the process of converging faster, the weights are reduced to the optimal values(ie., most of them become 0) quickly when compared to that of $L_2(w_1)$. Hence we can say L1 regularization introduces sparsity.