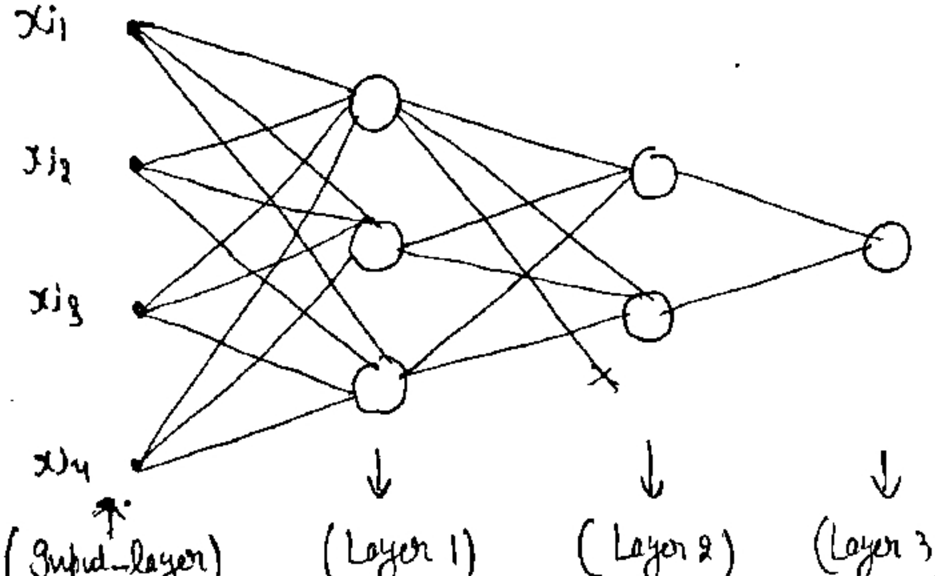
$\rightarrow$ **"Notation" of Neural Network (Multilayer Perceptron) :→** ①

We have seen that Multilayered structure is an extremely powerful idea to build extraordinarily powerful models, while you can avoid overfitting.

Let's look at some notations, so that it will become easy for us to follow, what is happening.

Let's understand the notation with the help of an example.



Let's assume our Dataset is $D = \{x_i, y_i\}$, let's take a regression problem because they are easy to understand. Let's assume x_i is a four dimensional point.

$$x_i \in \mathbb{R}^4 \; ; \; \& \; y_i \in \mathbb{R}$$

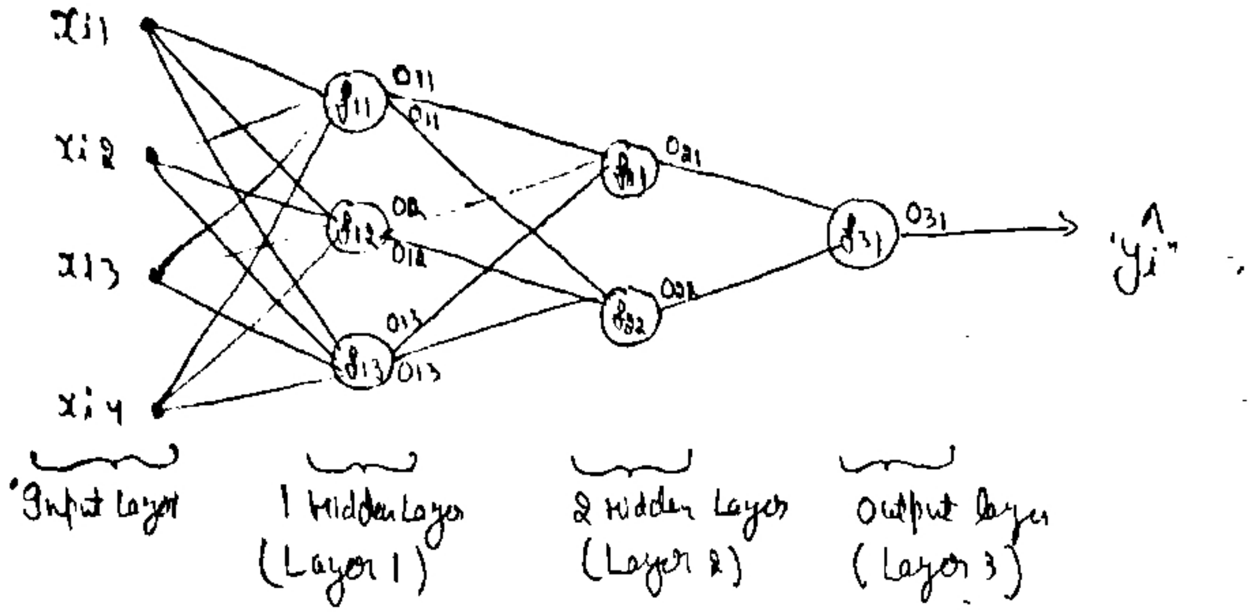
x_i is a 4-d point and y_i is a 'real value'. So, what we are solving here is a regression problem.

We are representing each of these features as 'x_{ij}'

Where 'i' is the point index (which point it is) & 'j' is the corresponding feature.

x_{ij} → point index, feature

Let's now connect all the inputs in the input layer with all the neurons in the first Hidden layer or layer 1 & so-on as shown:



In layer "1" we have 3 neurons & In each neuron we have a function.

Each function in layer 1 is represented as 'f_{1i}'

We have 3 neurons in layer 1, therefore corresponding to each neuron, we have a function

Layer 1 = { f_{11} f_{12} f_{13} }

(f_{11}: Layer 1, function 1 ... f_{13}: Layer 1, function 3)

Now let's assume we have 2 neurons in "layer 2" & all the outputs from "Layer 1" are connected to each neuron in "Layer 2" as shown above.

Let's now represent the functions in Layer 2

Layer 2 = { f_{21}, f_{22} }

(f_{21}: Layer 2, func 1; f_{22}: Layer 2, func 2)

& let's assume we have 1 neuron in 'layer' 3' or the output layer, and all the outputs from 'layer 2' are connected to 'layer 3' as we have seen. ③

The function in layer 3, since we have a single neuron, is represented as 'f_{31}'.

Layer 3 = $\{ f_{31} \}$

'$\hat{y_i}$' is the output that we get, when we input 'x_i'

The ideal output we want is 'y_i', but the model will output '$\hat{y_i}$'.

So, if 'y_i' is very similar to '$\hat{y_i}$', we have done an extremely good job.

let's now give notation for the functions used in each of the neurons

In generalised terms, the function is represented by

$\boxed{f_{ij}}$, $\left\{ \begin{array}{l} \text{'i' represents the layer number} \\ \quad\quad\quad\quad\& \\ \text{'j' represents the function index} \end{array} \right\}$

When we give inputs to a neuron (to activation function) we will get an output, (corresponding output).

& output notation is same as the function notation.

"Output of f_{11}" → 'O_{11}'	"Output of f_{21}" → 'O_{21}'
"Output of f_{12}" → 'O_{12}'	"Output of f_{22}" → 'O_{22}'
"Output of f_{13}" → 'O_{13}'	"Output of f_{31}" → 'O_{31}'

Note :→ Output of neuron is having same numbering as neuron itself

In generalised terms, output is represented as :→

'O_{ij}' {output of ith layer & jth neuron in that layer}

When we look at our perceptron, we see that there are weights associated with the edges & we are doing this to differentiate 'one input' from 'another'.

'Weights' on the edges are represented in a generalised way as :→

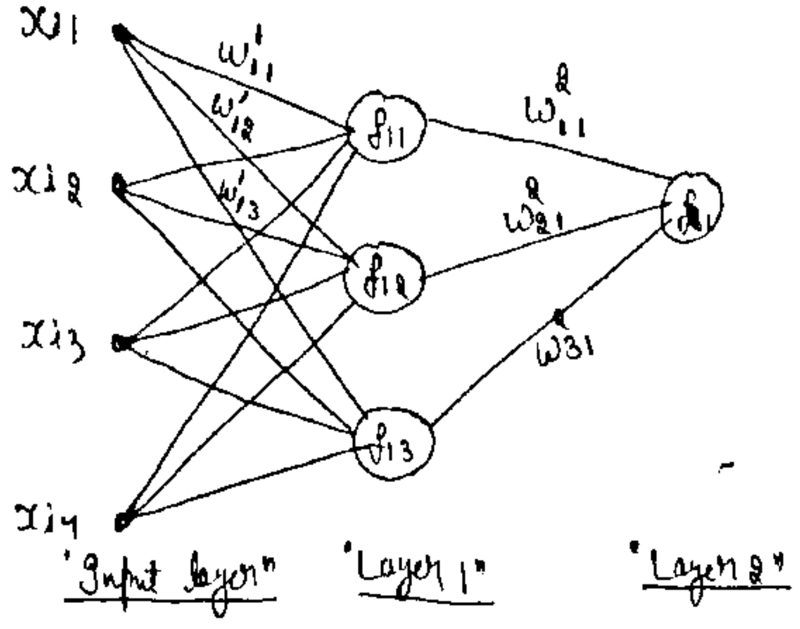
$$W_{ij}^{k}$$

'k' → what is the next layer

↳ Next layer is put as a 'superscript'

'i' → is 'from'

'j' → is 'to'

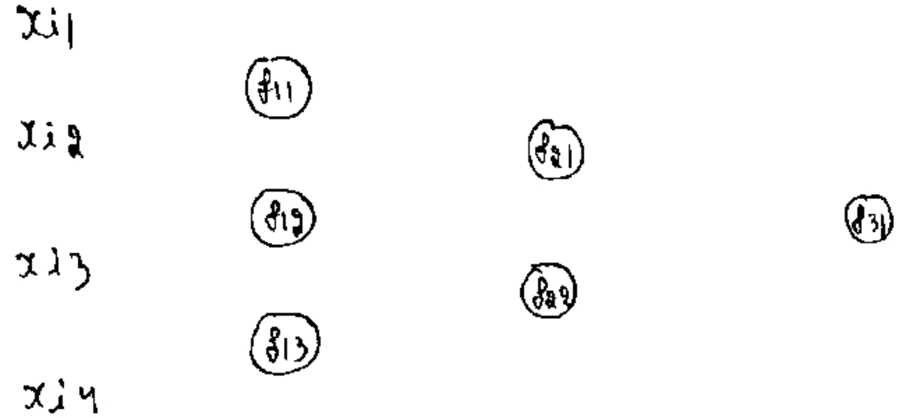


& so on

This is how we number each of the ['edge-values'], ['neuron functions'] & the outputs from ['each of the neurons']

Since in this "network" we have connected all the "possible pairs" Hence this network is called a ["fully connected neural network"].

Note :- Connections go from 'one layer' to the 'next layer' & not within the layer.

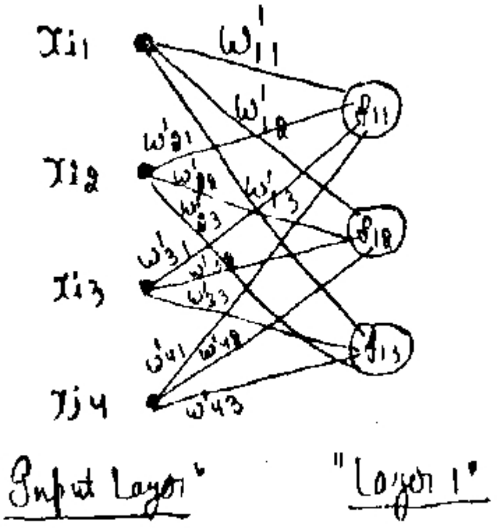


'Input layer' 'Layer 1' 'Layer 2' 'Layer 3'

Total connection are $\underline{4 \times 3 = 12}$, Total connection are $\underline{3 \times 2 = 6}$, Total connections are $\underline{2 \times 1 = 2}$,

All the possible connections we are making, hence it is called a 'fully connected neural network'.

Let's consider the 'first layer' & the 'input layer' only



Here we have in total 12 weights b/w ["layer 1"] & ["input layer"] becoz we have "four" inputs in the input layer & we have "3" neurons in "layer 1"

$$4 \times 3 = 12$$

PTO

The weights (12 weights) between "Input layer" and "Layer-1" are represented in matrix form as :-

$$W^1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} & \cancel{w^1_{14}} \\ w^1_{21} & w^1_{22} & w^1_{23} & \\ w^1_{31} & w^1_{32} & w^1_{33} & \\ w^1_{41} & w^1_{42} & w^1_{43} & \end{bmatrix}_{4\times 3}$$

This matrix of weights can be represented using 4x3 matrix.

We can represent all of these "12 weights" using a matrix of size '4x3'.

Similarly weights on the edges b/w 'Layer1' & 'Layer2' can be represented by a matrix W^2 of size '3x2'

$$W^2 = \begin{bmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^2_{31} & w^2_{32} \end{bmatrix}_{3\times 2} \qquad W^3 = \begin{bmatrix} w^3_{11} \\ w^3_{21} \end{bmatrix}_{2\times 1}$$

So, all these connections can be represented using matrices.

P.T.O

→ "Training a single Neuron Model" :→

Let's first understand, how to train a single neuron model, before we go to training an MLP (Multi layer Perceptron).

'Training' means to find the ['best weights'] best edge weights in a neural network model using training data (D Train).

Single neuron model is 'Perceptron' & 'logistic Regression'

"Perceptron & Logistic Regression" → ['Single Neuron Models'] for classification

Similarly if we think of "Linear-Regression". It can also be represented as a [single neuron model) for "regression"

'Linear-Regression' ⟶ ["Single Neuron Model for Regression"]

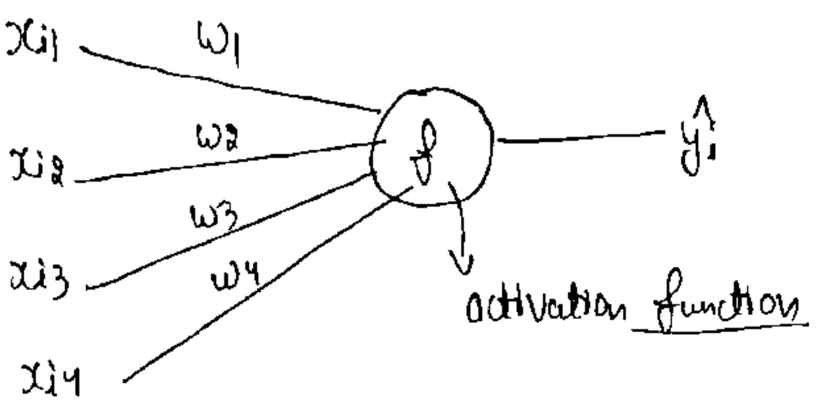
Here we will use regression as an example instead of classification becoz, the mathematics & calculus become easy. So whatever we learn for 'linear-regression' can easily be extended to 'logistic-regression', 'perceptron' etc.

We have already seen how a 'linear-regression" model can be represented as an 'optimization problem'.

So, we revisit all these concepts of 'linear-regression' as they are very important in understanding the training of a neural network.

P.T.O

Let's now understand 'Linear Regression problem' from the perspective of a 'neuron'. (single neuron model)



In a linear regression, the problem is

$$\hat{y}_i = \sum_{j=1}^{d} w_j x_{ij}$$

| $D_{Tr} = \{x_i, y_i\}$ $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$ |

In linear regression, we have.

$$\hat{y}_i = w^T x_i$$

(This is the relationship b/w 'x_i' & $\hat{y}$)

So, given 'x_i' & 'y_i' our job is to find the optimal 'w'.

Now let's go back to a single neuron framework above

Here input to "f" is $\sum_{j=1}^{4} w_j x_{ij}$

becoz x_{i1} gets multiplied with w_1

x_{i2} " " with w_2

x_{i3} " " with w_3

& x_{i4} " " with w_4

& then we sum up all of them as $\sum_{j=1}^{4} w_j x_{ij}$

∴ the input that we get is '$W^T x_i$'

In a 'linear regression' the output is also '$W^T x_i$' which means this function ['f'] is called an [identity function] in case of a linear regression

An identity function is, if we input "Z" to a function & same "Z" is returned as an output then it is an 'Identity function'

$$\left[f(Z) = Z \right]$$

So, this is in the case of 'Linear regression'. becoz in linear regression the input that we give to this activation function is $W^T x_i$ & the output is also $W^T x_i$, so if both input & output of a function is same, then the function is known as an 'identity function'.

Note :- [In case of a logistic regression, 'f' is a sigmoid function

In case of a simple Perceptron, 'f' is a thresholding function.

& In case of a linear regression, 'f' is an identity function]

Here we always represent this function as 'f' because whatever we derive, it should be generalized.

Second, thing, In linear regression, when we represented it as an optimization problem.

So, optimization problem of "Linear Regression" can be written as :-

find the w_i's that minimizes the loss function & in Linear regression we have used square loss

$$\min_{w_i} \sum_{i=1}^{n \to \text{no of points in training data}} (y_i - \hat{y_i})^2 + \text{regularizer}$$

where $\hat{y_i} = w^T x_i$

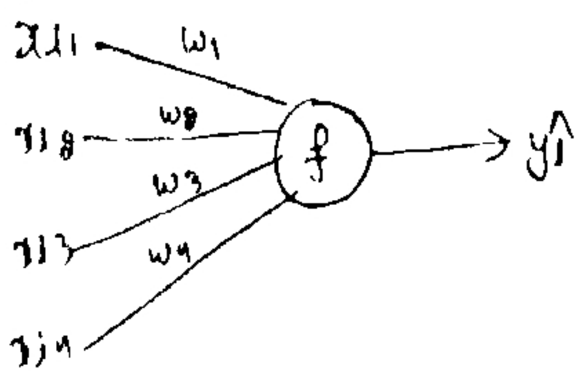
This can can rewritten as:

$$\left\{ \min_{w_i} \sum_{i=1}^{n} \underbrace{(y_i - w^T x_i)^2}_{\text{squared error}} + \underbrace{\text{regularizer}}_{\text{It could be any } L_1 \text{ or } L_2} \right\}$$

This is the optimization problem for "Linear Regression"

Even in a single neuron model, this is the optimization problem that we solve.

Let's try to connect this optimization problem with a single neuron model :->



$D = \{x_i, y_i\}_{i=1}^{n}$ → training data

where $x_i \in \mathbb{R}^4$

& $y_i \in \mathbb{R}$

This means we are solving a Regression problem let's go step by step

Step no 1 is Define a 'loss-function'

$$\mathcal{L} = \sum_{i=1}^{n} (y_i - \hat{y_i})^2 + \text{reg}$$

When we input 'x_i' through this network, we are going to get a '$\hat{y_i}$' which is different from 'y_i', we want to penalize it. The farther away "$\hat{y_i}$" is from 'y_i', we want to penalize it. In our loss function.

let us also define a term called "$\mathcal{L}_i$"

$$\mathcal{L}_i = (y_i - \hat{y_i})^2$$

Note: squared loss of one point, we are calling it as '$\mathcal{L}_i$'.

& the sum of all the squared losses, is called '$\mathcal{L}$'

$$\therefore \boxed{\mathcal{L} = \sum_{i=1}^{n} \mathcal{L}_i}$$

'$\mathcal{L}$' is loss across all the training data & '$\mathcal{L}_i$' is the loss across each training point

Also $[\hat{y_i} = W^T x_i]$

We have our $\hat{y_i}$, Now on top of it, we apply the loss function & this loss function, we are computing using $[y_i]$ & $[\hat{y_i}]$. We get $\hat{y_i}$ from the model & 'y_i' is from the actual data & all we care about is $\mathcal{L}(y_i, \hat{y_i})$.

we have our x_i's & our goal is to find the optimal w's. (weights). This is what training means.

Step no. 2 :⇒ Write the optimization problem.

$$\min_{w_i} \sum_{i=1}^{n} (y_i - \underbrace{w^T x_i}_{\hat{y_i}})^2 + \text{reg.}$$

So, from neural network perspective

$$\left[\hat{y_i} = f(w^T x_i) \right]$$

In case of linear regression this 'f' is $\left[\text{identity function}\right]$

$$\min_{w_i} \sum_{i=1}^{n} \left(y_i - f(w^T x_i)\right)^2 + \text{reg.}$$

[Now if the 'f' is 'I' we get linear regression
is 'Threshold' we get perceptron
& if it is sigmoid, we get
logistic regression]

This is a generalized optimization problem, we want to solve.

In vector notation, it can be represented as, find a vector 'w' which minimize this thing.

$$\left[w^* = \underset{w}{\arg\min} \sum_{i=1}^{n} \left(y_i - f(w^T x_i)\right)^2 + \text{reg.} \right]$$ (13)

Step no 3 :→ Solve the optimization problem.
(for that we use gradient descent)

(a) Initialization of weights. w_i's ⟶ random initialization

(b) Compute derivative of L wrt 'w'

$\nabla_w L$ (This is a vector derivative)

$$\nabla_w L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_4} \end{bmatrix} \quad \because \{ w \in \mathbb{R}^4 \text{ since } x_i \in \mathbb{R}^4 \}$$

This is vector representation of 'partial derivative'.

(c) Once we compute this, we will iteratively say that.

$$\{ w_{new} = w_{old} - \underbrace{\eta}_{\text{'Learning rate'}} * [\nabla_w L]_{w_{old}} \} \rightarrow \text{Vector notation}$$

↓
Now let us write scalar notation of this

$$(w_i)_{new} = (w_i)_{old} - \eta \left[\frac{\partial L}{\partial w_i}\right]_{(w_i)_{old}}$$

we are computing derivative at a point & derivative is nothing but a slope.

we will keep repeatly new & old, we will be having a for loop as

for Iterations = 1 to K

$$(w_i)_{new} = (w_i)_{old} - \eta \left[\frac{\partial L}{\partial w_i}\right]_{(w_i)_{old}}$$

We will run it, till we converge.

The major difference between Gradient descent & Stochastic Gradient descent is :-

In both of them, we will compute the partial derivatives w.r.t 'w'

In 'GD', $\nabla_w L$ is computed for all the points in our dataset

$$\nabla_w L \rightarrow x_i\text{'s} \ \& \ y_i\text{'s}$$

whereas in case of 'SGD' we will approximate the gradient by taking one point $\{x_i, y_i\}$ or a small set of points (batch of points)

$$\nabla_w L \xrightarrow{\approx} \text{one point } \{x_i, y_i\}$$
or small batch of points

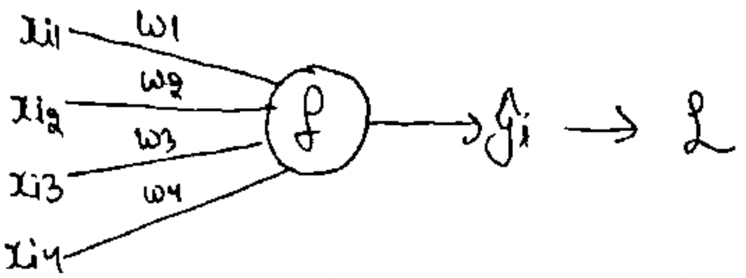
Now let us see how to compute the derivative.

PTO

Now let us see how to compute the derivative.

{ $\nabla_w L$ }

Let's draw the perceptron again



[We are getting $\hat{y}_i$ as an output of "f" & on top of that we are applying Loss function.]

& In case of regression, this "L" is a squared loss.

Now we want to compute $\nabla_w L = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}\right]^T$

This is supposed to be a column vector, so we are taking transpose of it to convert it into a row-vector.

Now let us see how to compute each of them

becoz, 'w_1', 'w_2', 'w_3' & 'w_4' are independent of each other

Let's compute $\frac{\partial L}{\partial w_1}$, if we look at the path from 'L' to w_1, we have a function 'f' in b/w & 'w_1'. $\hat{y}_i$ is a constant as it is computed from x_i's

So, here we want to compute derivative of 'L' w.r.t 'w1'

Let's use the chain rule of differentiation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f} * \frac{\partial f}{\partial w_1}$$ → chain rule of differentiation

$\because$ { $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial f_1} * \frac{\partial f_1}{\partial f_2} * \frac{\partial f_2}{\partial x}.$ } → chain rule of diff.

We are taking the path from 'w1' to 'L' & whatever variable or function exists in the path, we have to take the derivative of that w.r.t to 'w1'

Note :- x_i's & $\hat{y}_i$'s are constants

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + reg$$ (let's ignore regu for a while)

$$\& \ \hat{y_i} = f(w^T x_i)$$

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

Let's write $f(w^T x_i)$ as f for simplicity.

$$\frac{\partial L}{\partial f} = \sum_{i=1}^{n} (y_i - f)^2$$

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

Now we need to find $\frac{\partial f}{\partial w_1}$

we know $f(w^T x_i) = w^T x_i$

Now $\frac{\partial f}{\partial w_1} = x_i$ simply

L is basically summation across all the points

let's take L_i for

$L_i = (y_i - \hat{y}_i)^2$ for a single point

$$\frac{\partial L_i}{\partial w_1} = \frac{\partial L_i}{\partial f} * \frac{\partial f}{\partial w_1}$$

$$\Rightarrow -2(y_i - \hat{y}_i) * x_i$$

$$\therefore \frac{\partial L_i}{\partial w_1} = -2x_i (y_i - \hat{y}_i)$$

Now let's find $\frac{\partial L}{\partial w_1}$

$$\boxed{\frac{\partial L}{\partial w_1} = \sum_{i=1}^{n} -2x_i (y_i - \hat{y}_i)}$$

Similarly we can compute $\frac{\partial L}{\partial w_2}$ --- $\frac{\partial L}{\partial w_4}$

when we have all of these computed, we can easily compute $\nabla_w L$

once we have $\nabla_w L$ computing

W_{new} is straight forward

as:

$$W_{new} = W_{old} - \underbrace{\eta}_{\text{learning rate}} \left[\nabla_w L \right]_{W_{old}}$$