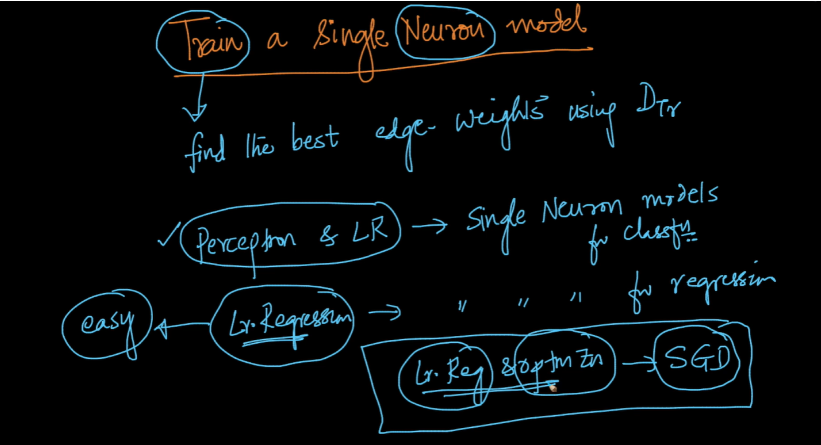
**Training a single-neuron model**

As we already know training a single neuron model is to find the best edge weights using Dtrain

Perceptron and logistic regression is a single neuron models for classification

Linear regression is a single neuron models for regression.

As we do optimization of linear regression using SGD. Here in neural network we also use same.



As we know we can predict output of linear regression using linear optimization

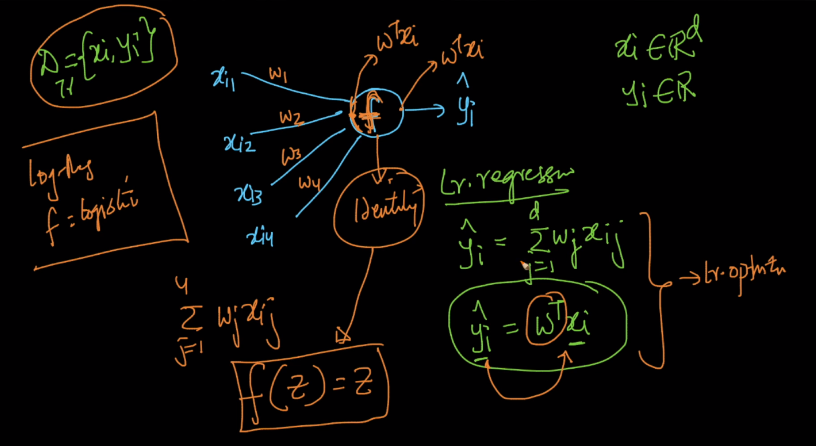
Y\_hat = sum\_j=1\_to\_d Wj­ \* Xij

i.e WTXi

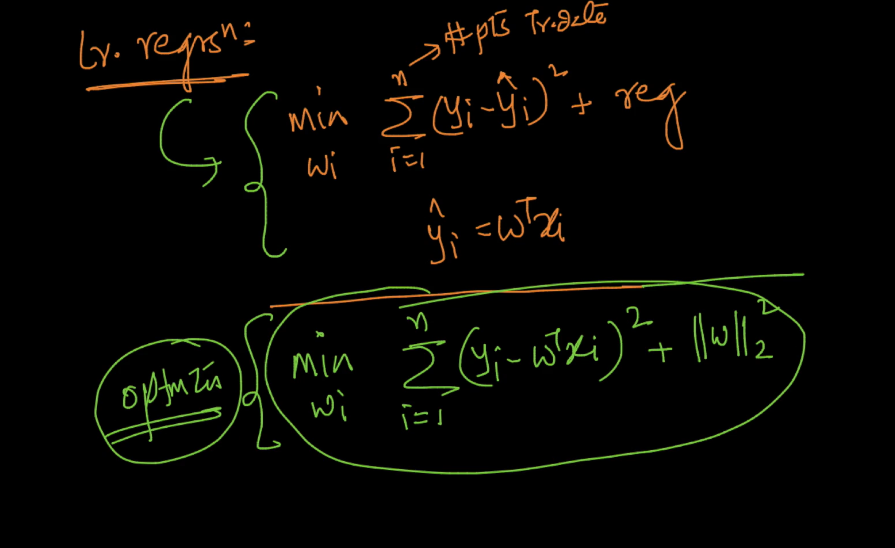
therefore for linear regression what we give in input to function in neuron same output given by that function therefore as input and output are same i.e WTXi therefore it’s name is identity function (i.e f(z) = z)

if we give sigmoid function then it becomes logistic regression

if we give thresholding function then it becomes perceptron.



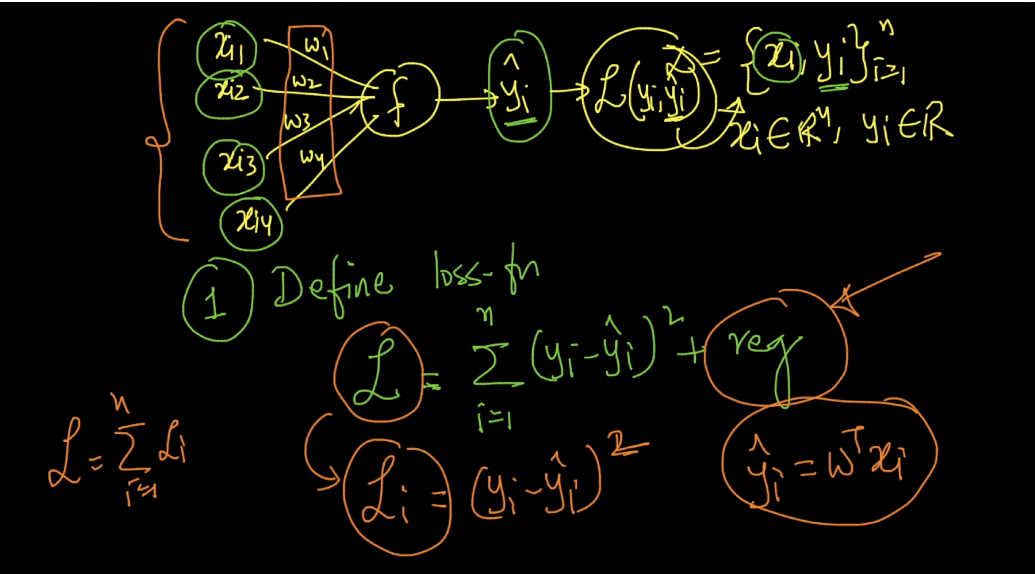
We use square loss for regression. Which is shown below and thus we got our optimization by loss + regularization term. And in this optimization we need to find weight matrix which minimize the loss i.e y and predicted y\_hat should be close or equal and we find that using SGD



To train a single neuron model involves many steps :

1. Defining loss-function :

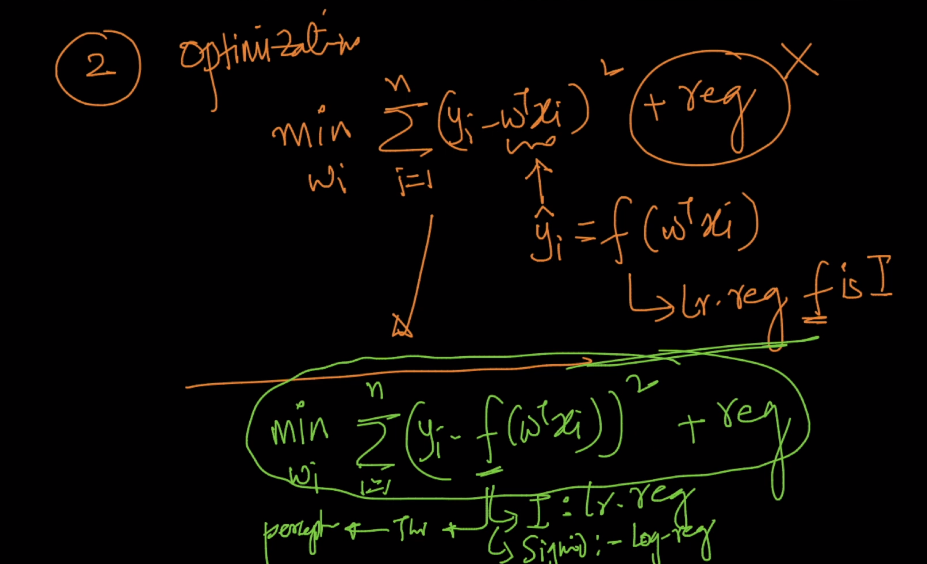
Loss function for regression is shown below by L which applies at the output of neural network.



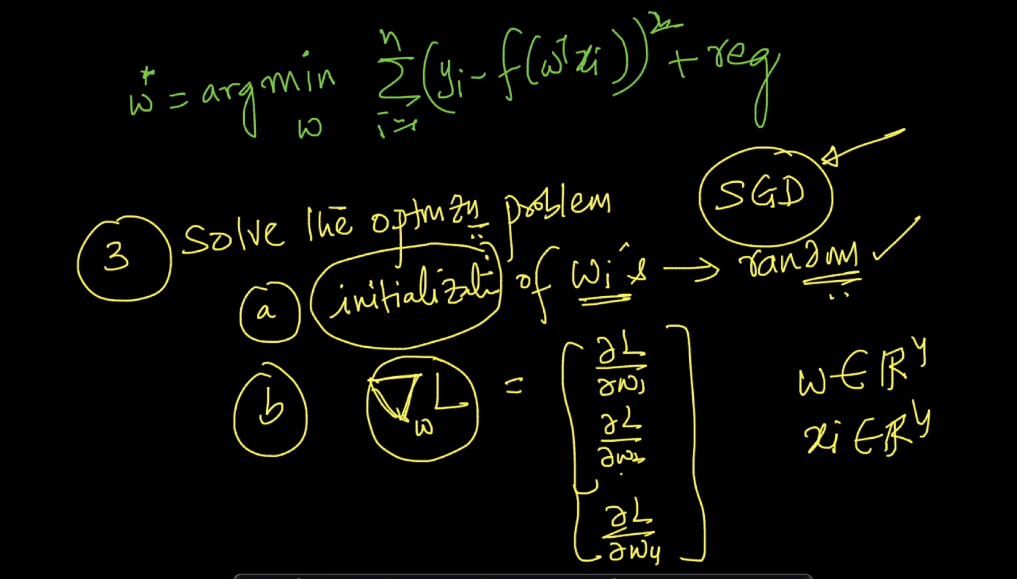
2)Optimization :

As in this neural network output is given by f(wTx) therefore we use it here as shown below.

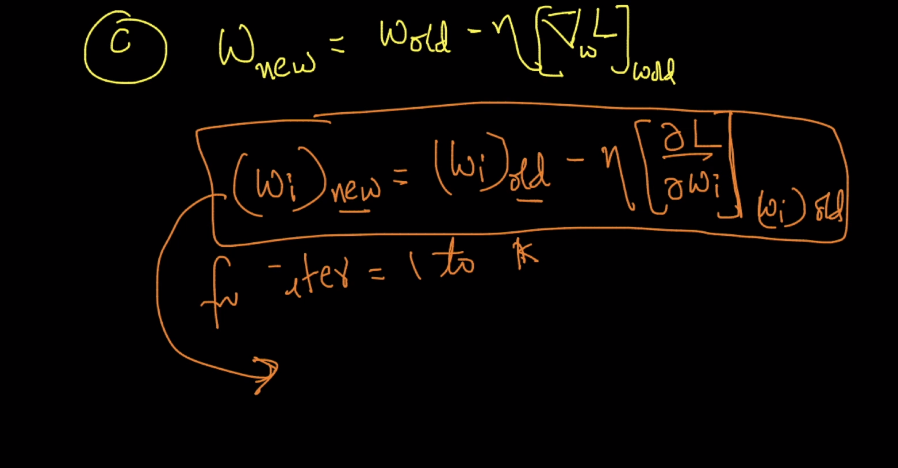
In this way we can generate optimization for logistic regression by replacing this function with sigmoid function and optimization for perceptron by replacing it with thresholding function.



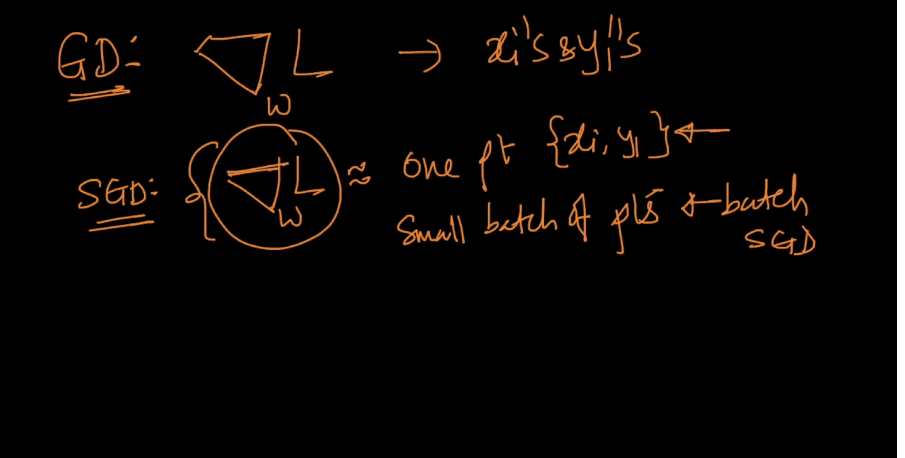
1. Solving the optimization problem
2. First initialize wi’s and learning rate n randomly.
3. Then differentiate L by w as shown below.



1. Now new w is form as shown below



In gradient descent we use all points but in SGD we use small batch of points.

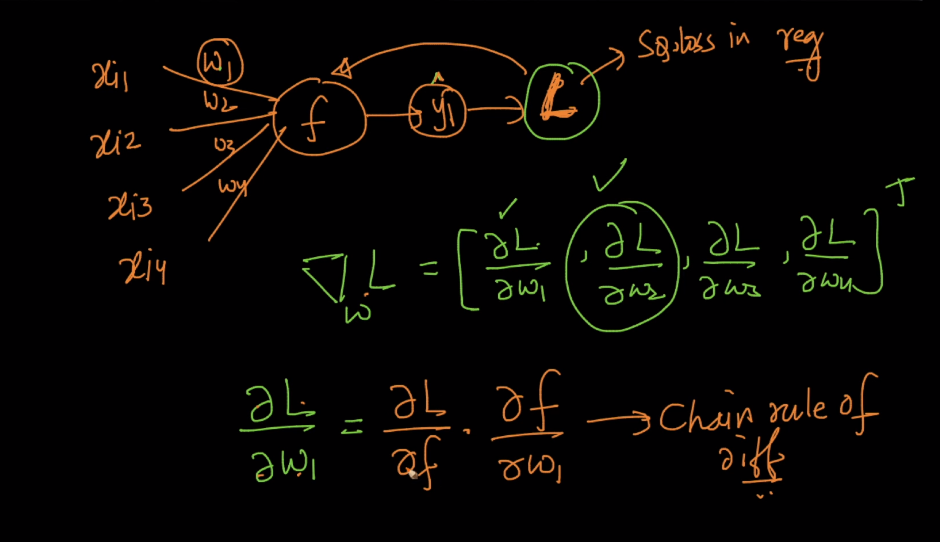


Now we want to partially differentiate L with w1 , w2, w3­, w4

Here in below image we use transpose of del\_l because we want it as a column vector.

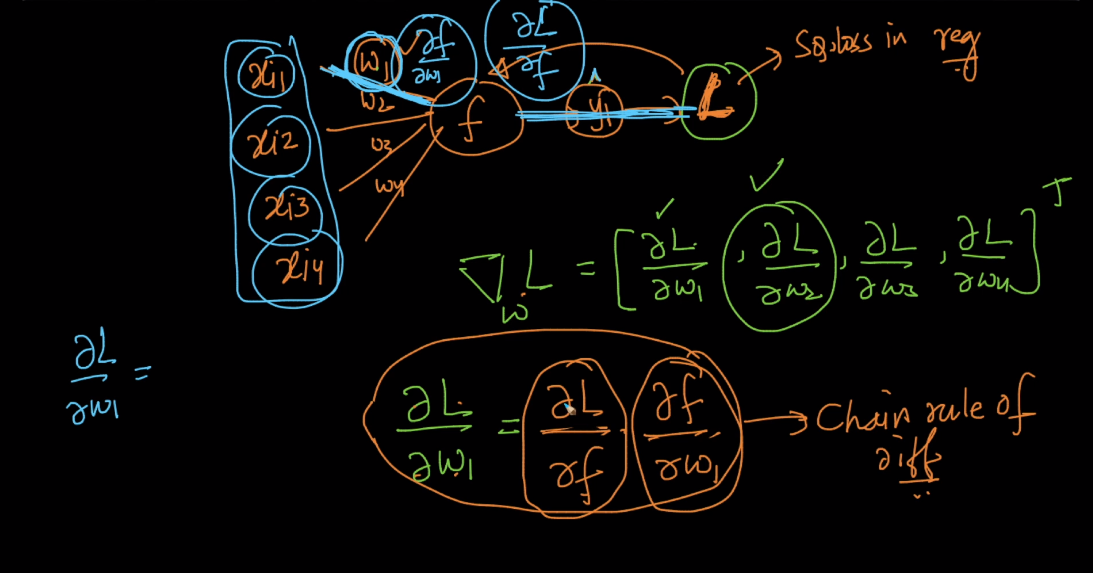
And we do that with chain rule of differentiation.

del\_L/del\_w1 = del L/del f \* del\_f/del\_w1



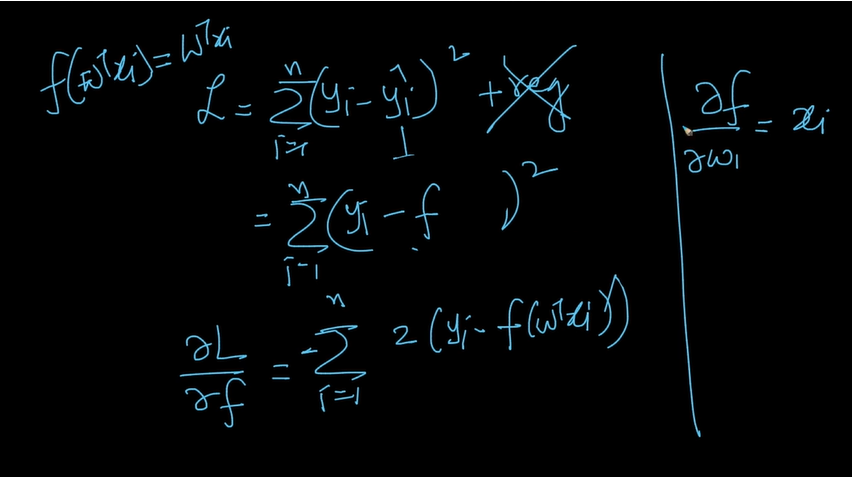
We can see this chain rule by neural network as well. First we got output of differentiation of l/w1 from path of xi1 to L .

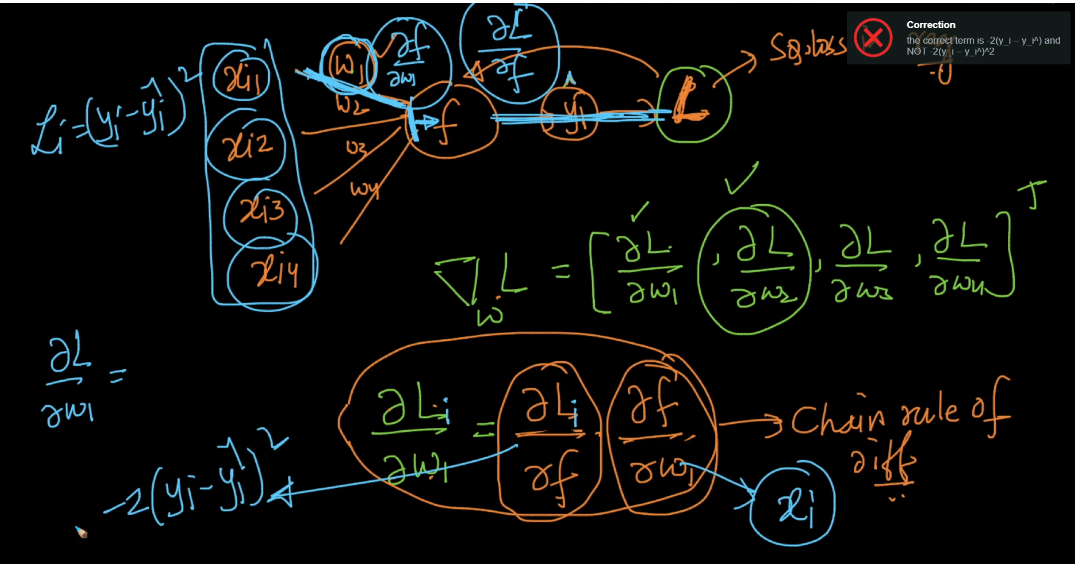
Now to diff. it go backward i.e first from L to f path del\_L/del\_f then multiply it with path f to xi1 i.e del\_f/del\_w1

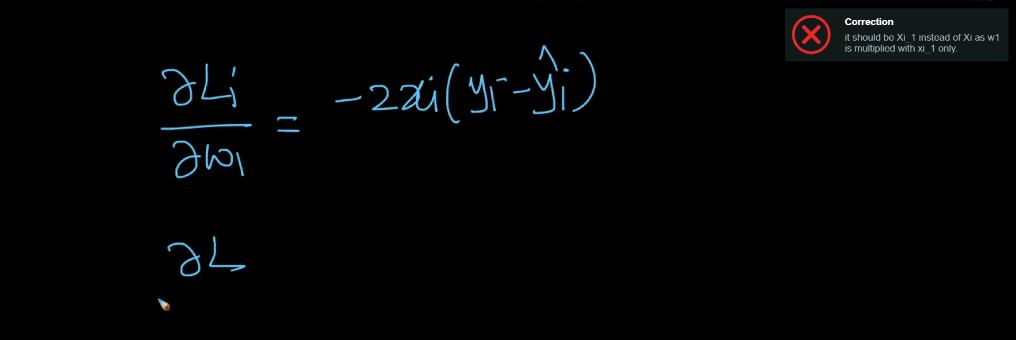


Therefor del\_L/del\_f therefore we got o/p -2(yi – f(wTXi)

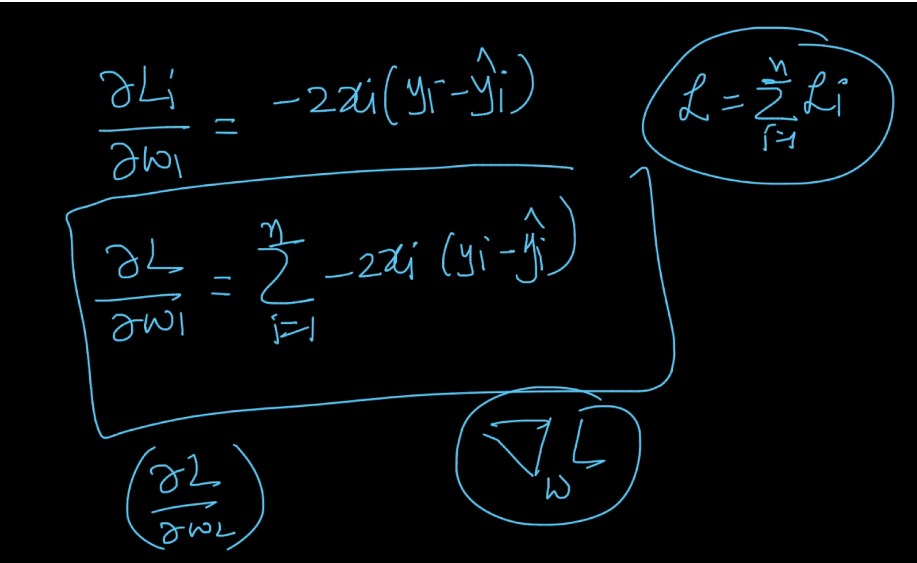
Del\_f/del\_w1 = xi



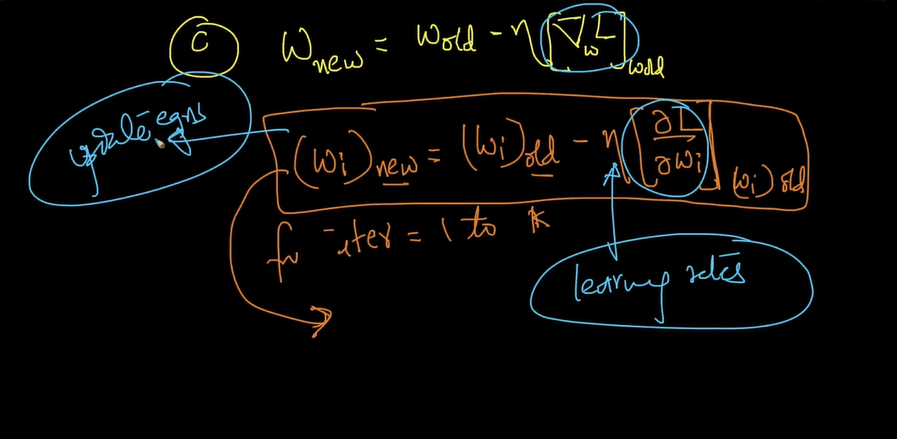




In this way we can partially differentiation and got value of del\_l/del\_w



Now just we have to put this in equation and then update the edges



Links :

How to train nn : <https://ml4a.github.io/ml4a/how_neural_networks_are_trained/>

code for single layer nn : <https://www.geeksforgeeks.org/single-neuron-neural-network-python/>

Comments :

