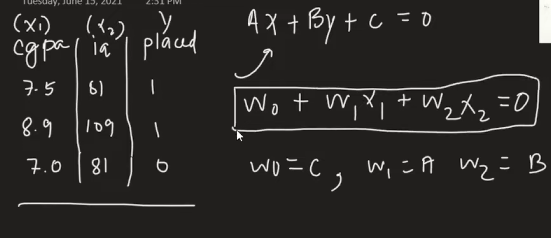
ML Algorithms

Logistic Regression:

**Logistic regression = Perceptron + Sigmoid + Log Loss**

* Algorithm: (Perceptron Trick)

Taking an example dataset

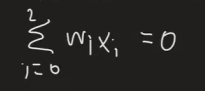


In logistic regression we write the line equation given above

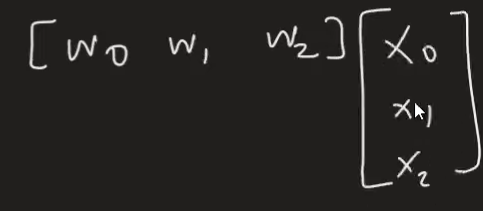
If we added x0 column to the dataset equation changes to



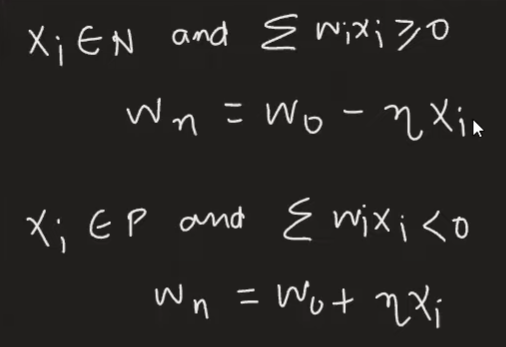
So, if added a column in this equation like this so general equation becomes



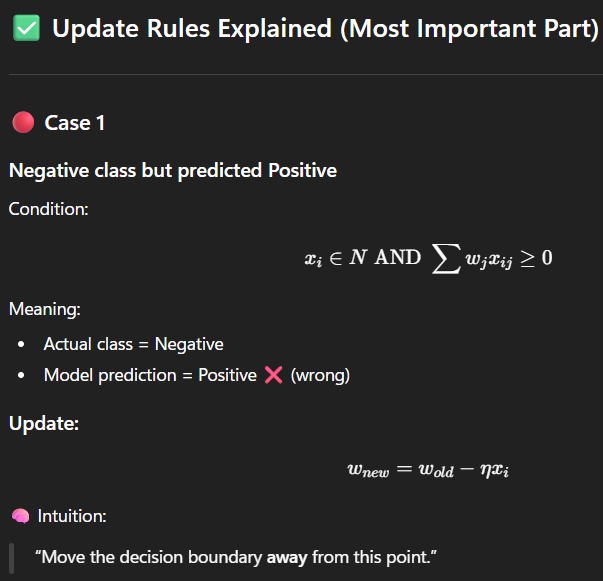
Matrix representation

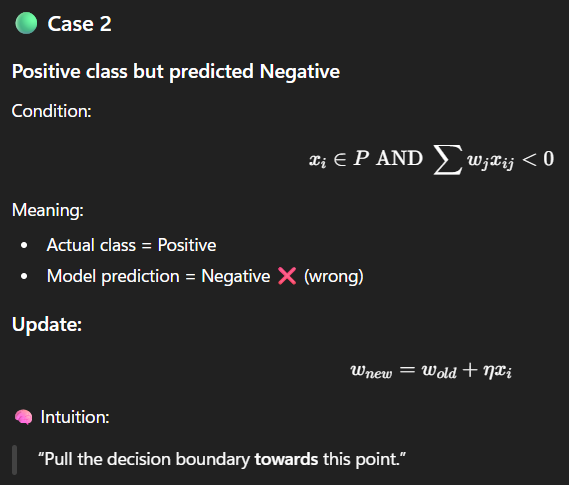


This is raw photo of algo:

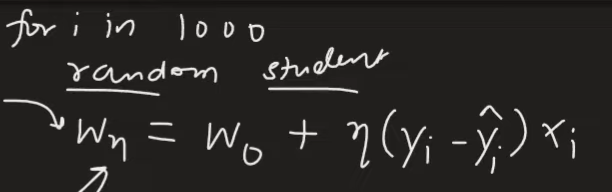


there are two cases in above algo so we will study each of them

Case 1:

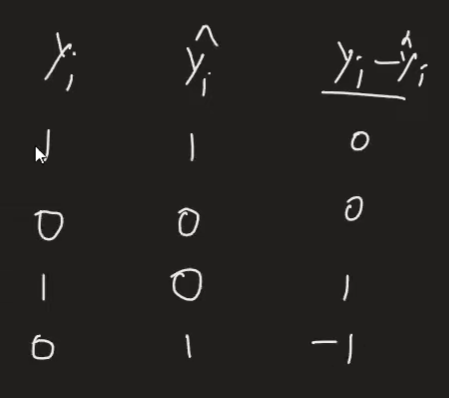
Case 2:

Will replace one line for this whole



This y = actual value and y(cap) = predicted value

So will consider 4 scenario which actual happen in this:



For first and second case the resultant will be Wn(new)  = Wo(old)

* The first one is actual value on graph says placement done and predicted value also says same
* The second one is actual value on graph says placement not predicted

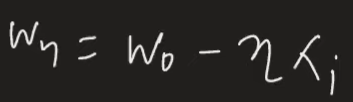
value also says same

Third case the model says not placed and actual value on graph is

Placed So this comes under case 2 from above

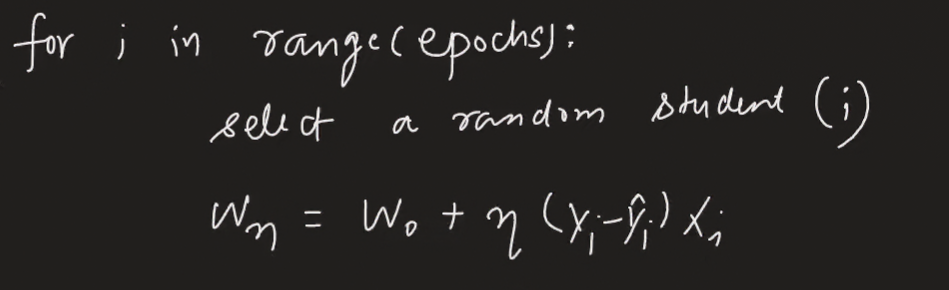
Equation comes out to be =

Fourth case the model says placed and actual value on graph is

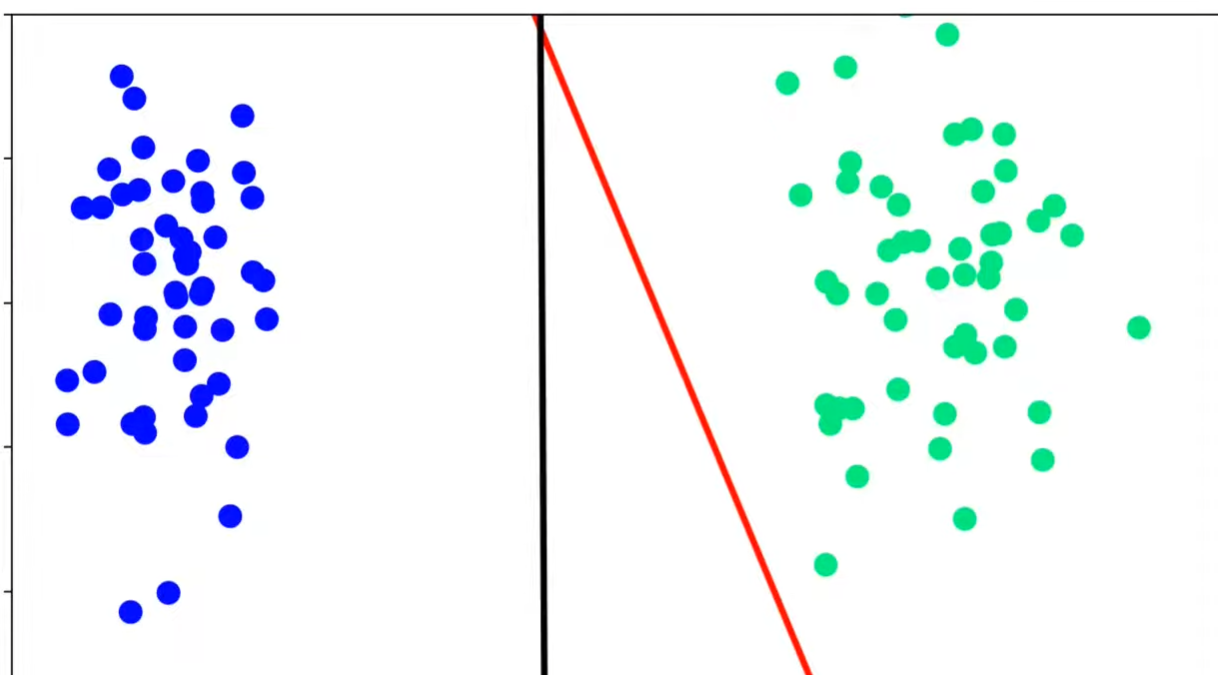
Not Placed So this comes under case 2 from above

Equation comes out to be =

Finaly the algo comes out to be:



So, there is some bad side of this perceptron trick is that

As you see in figure 

The red line is drawn between two classes is by perceptron method which des not tend to improve further and bias to the one green side of classification

As the Black line represents the Logistic regression, we can say it tries to draw equal between the two classifications which is causes it tries to improve after the classification of points

* Fix

So, we have to create an equilibrium in the process,

First as we get a misclassified point it tries to pull the line towards it

Now we add: if we get correctly classified point, it will push the line

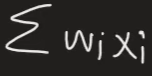
* But this push and pull will depend on magnitude of how much distance is between point and line
* In case of misclassified:

If distance is small pull will be small and if distance is big pull will be big (distance directly proportional to pull)

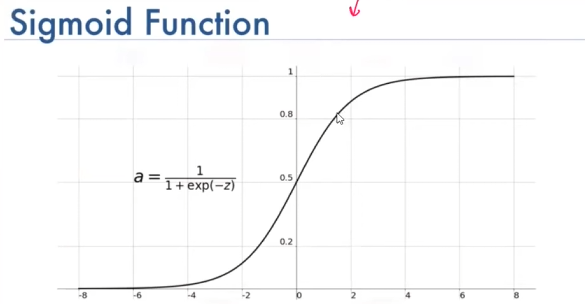
In case of correctly classified:

If the distance is small the push will be larger and if distance is large push will be smaller (distance inversely proportional to push)

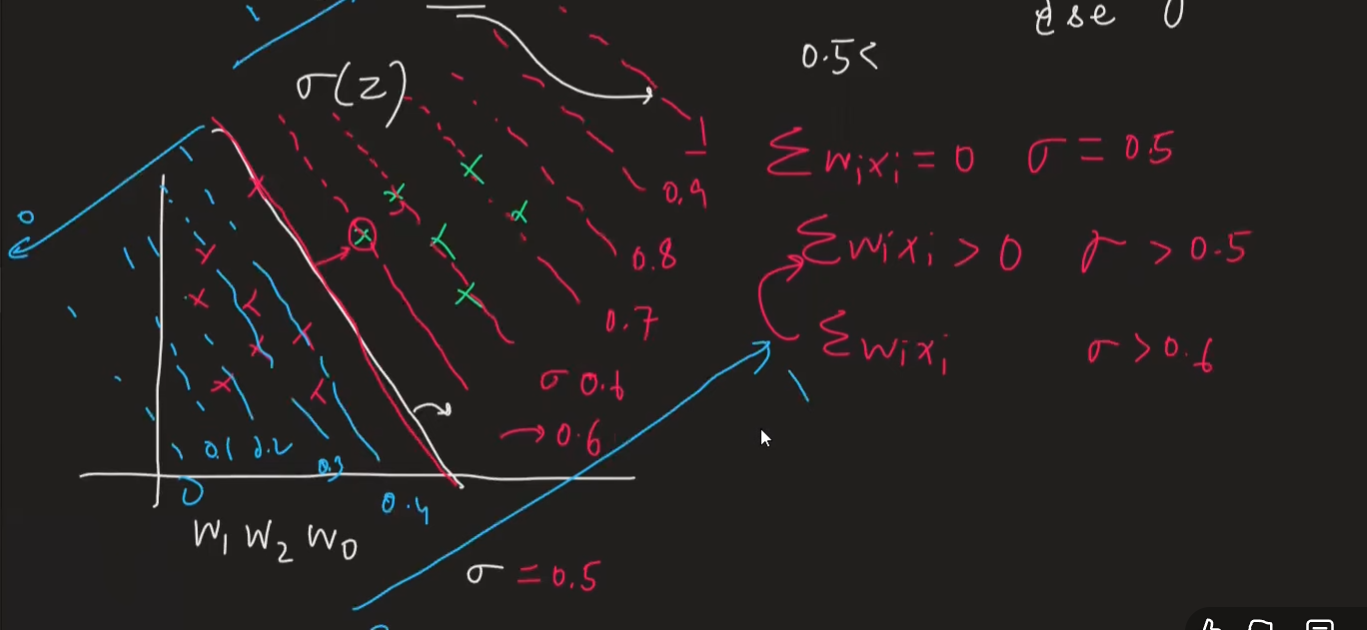
Will use sigmoid function instead of step function for prediction done by model it will not give output as 0 and 1

this function would be given to sigmoid function

no the step function



If z is positive, we get 0.5 or more value if negative we get less than 0.5 value

It will work like : 

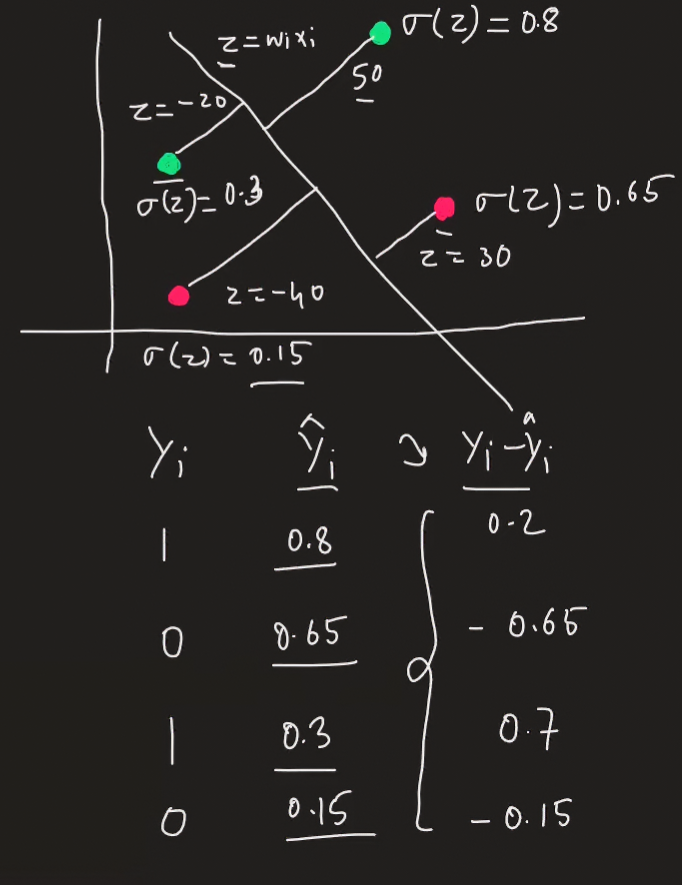
We can see that sigmoid is creating a gradient

Example:

As you are far from .5 in positive region your placement chances are more and as you are far from 0.5 in negative region chances your placement chances are less

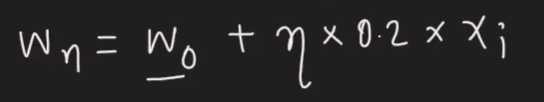
It acts as a probability function also

We will understand this by Example:



So, the yi – y(hat) would never be zero cause of sigmoid function

“Misclassified --- PULL and Correctly Classified --- PUSH”

1. First point will give equation (correctly Classified Point)

Which will add some value in equation and it will push the line

1. Second point will give equation (Misclassified Point)



Which will subtract from the equation and it will pull the line toward itself

1. Third point will give equation (Misclassified Point)



Which will add some value and will pull the line

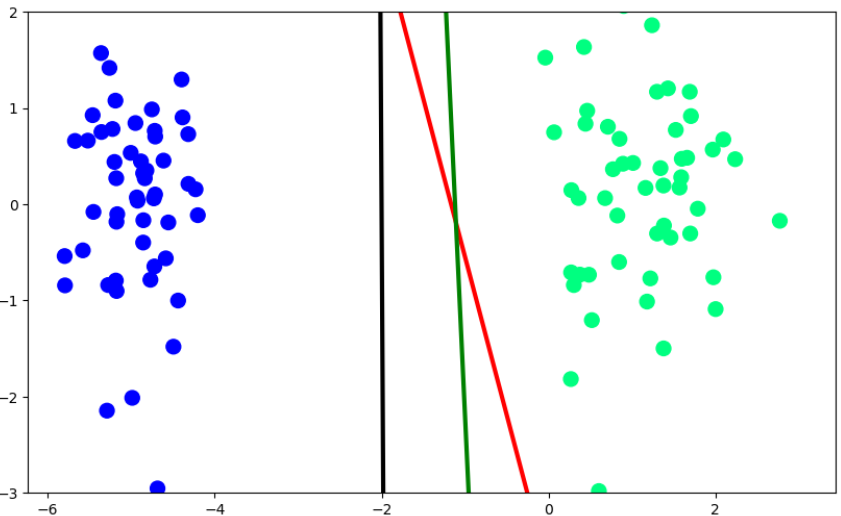
1. Fourth point will give equation (Correctly Classified point)



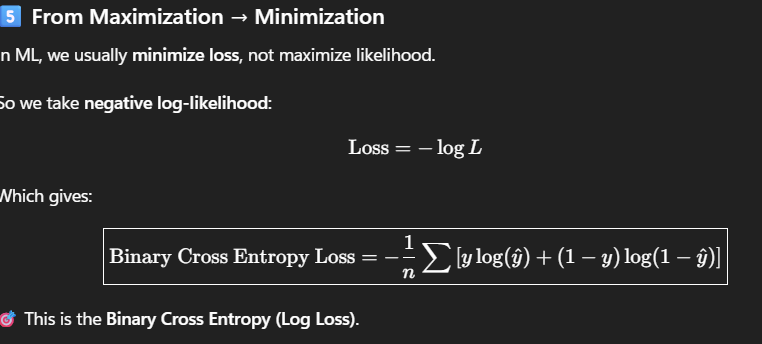
Which will subtract from the equation and it will push the line away

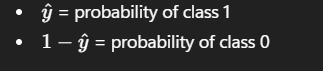
* After Implementing sigmoid function, the solution is still no there

as in the figure: green line represents the sigmoid + perceptron line



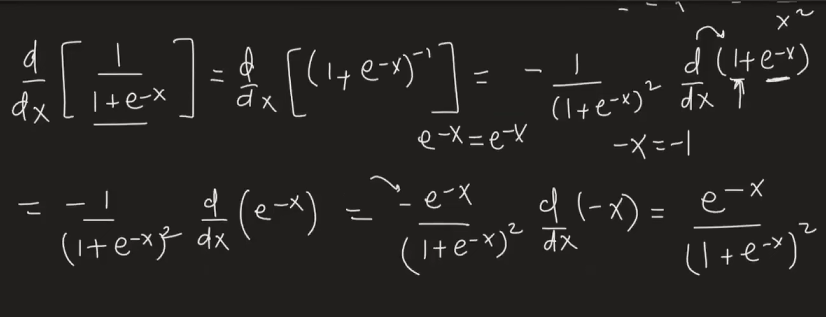
Will get the loss function which is given as

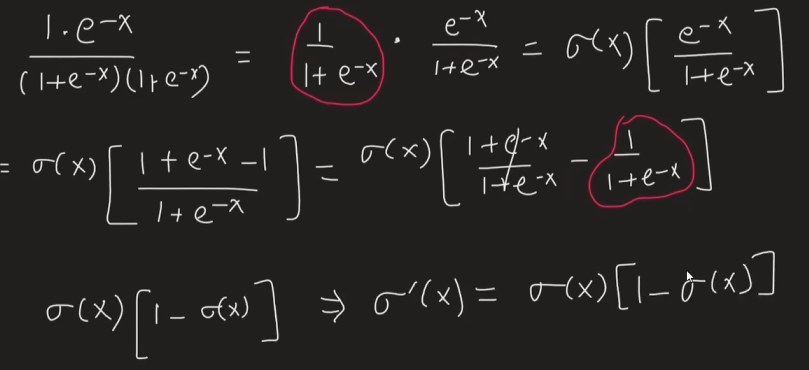


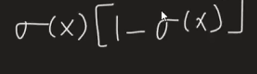
Were,

So, we can get this function solution by gradient descent we can’t calculate it from any other method

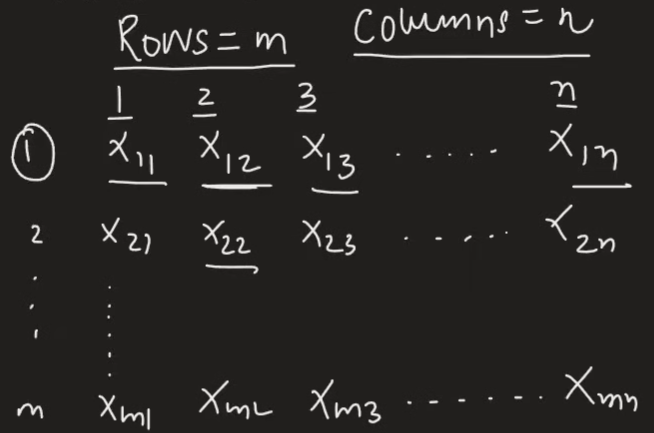
Now first here need to find the derivative of sigmoid function before next step

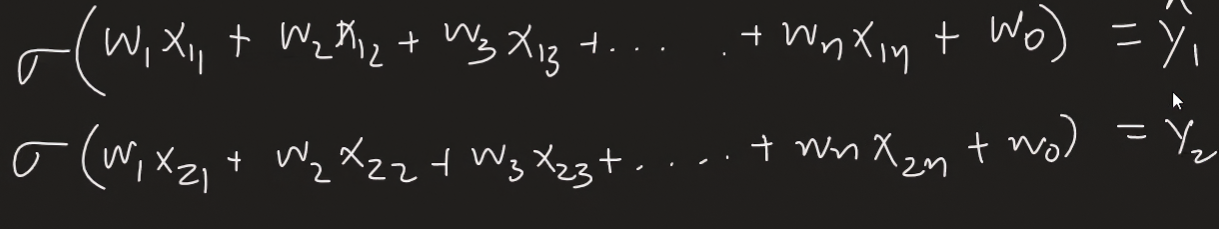




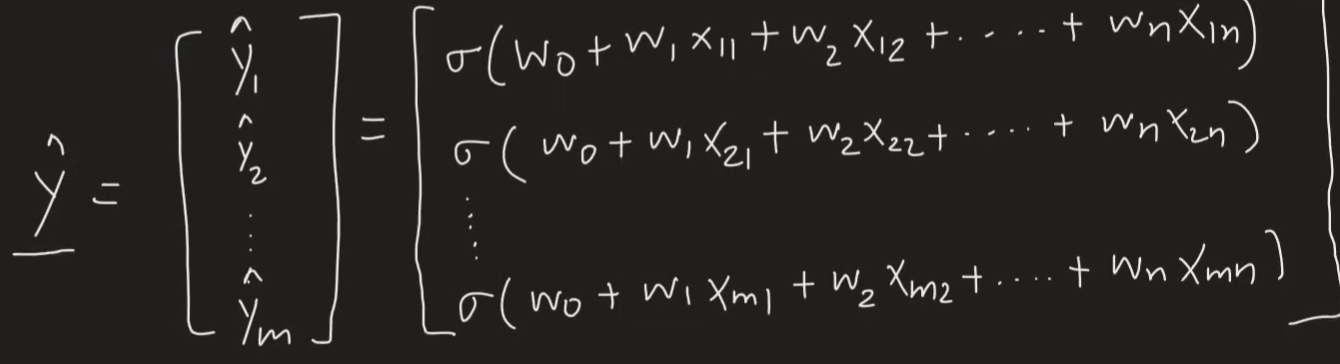


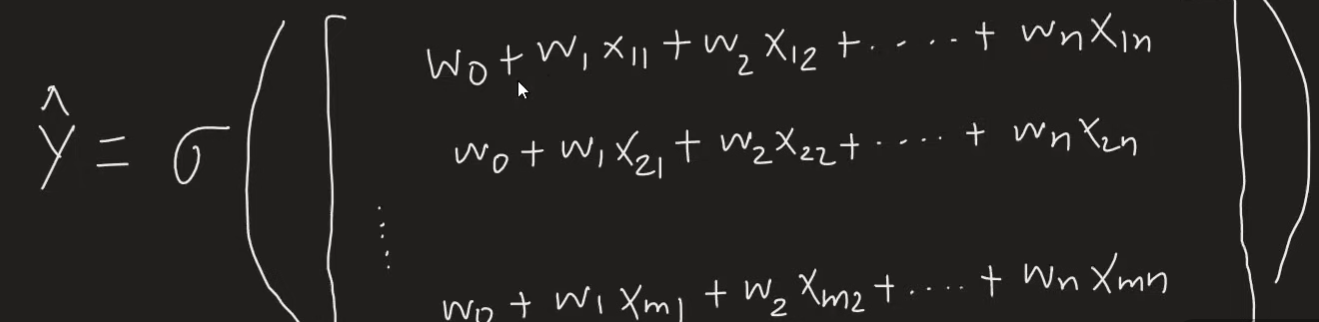
* Gradient Descent:

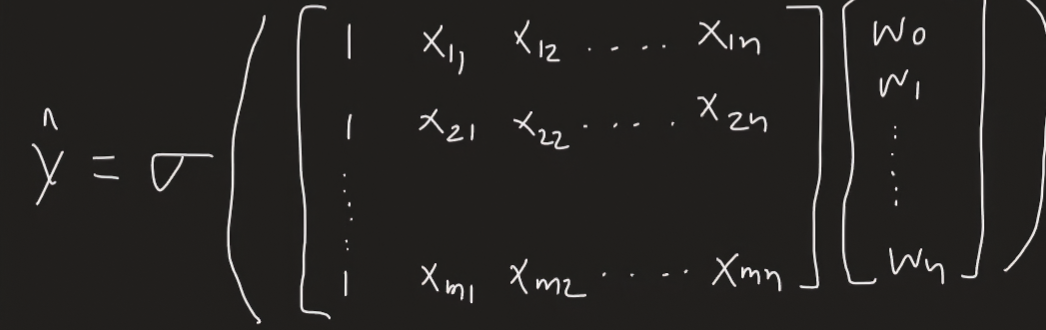




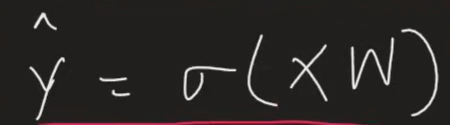
Making a y(hat) matrix for all predicted value



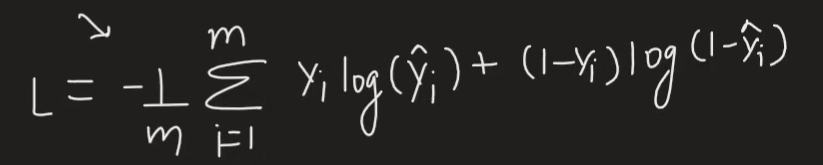




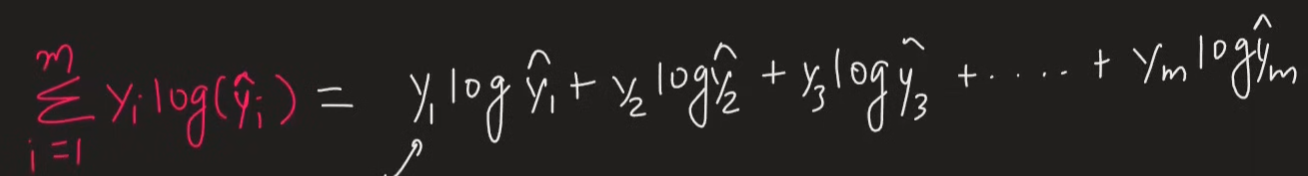
Y(hat) = sigmoid \* (dot product X . W)

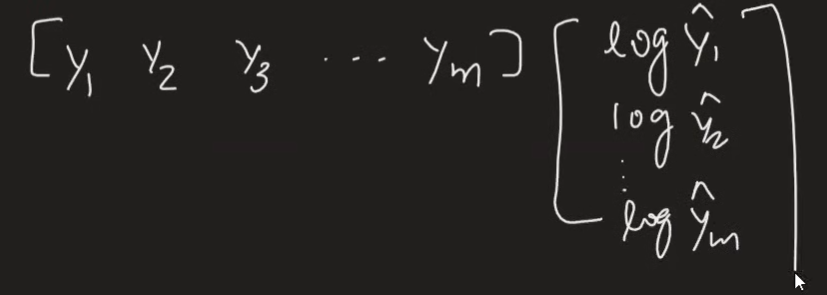


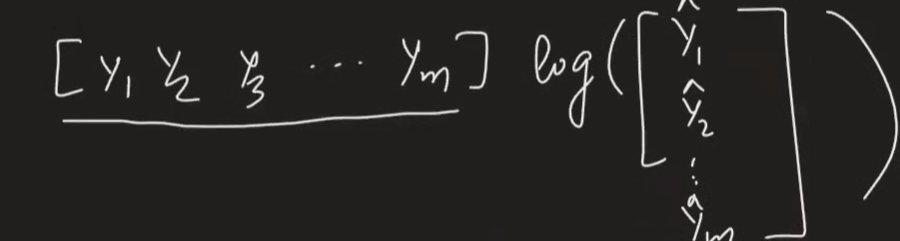
Matrix conversion of loss function:

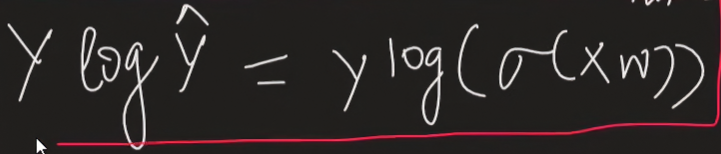




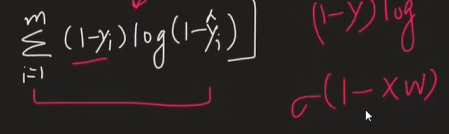




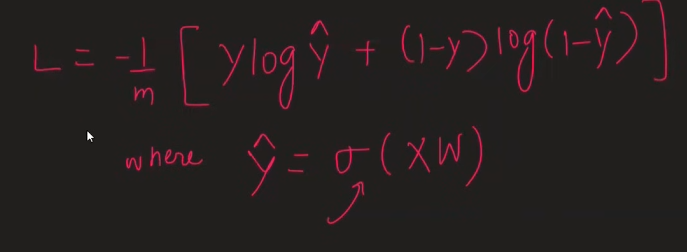


It can be simplified to 

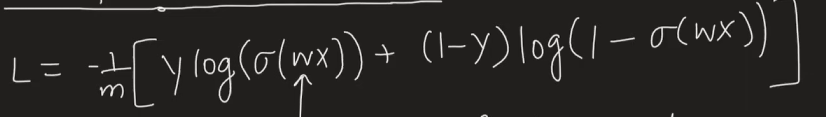
And other parts simplified to

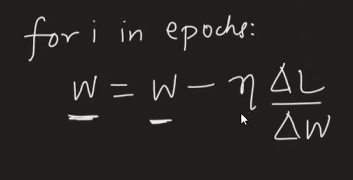


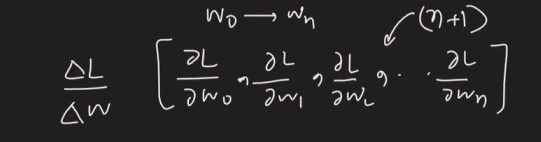
Full loss function come out to be:



So now we need to find the value of w matrix in such way that value of loss function would be minimum



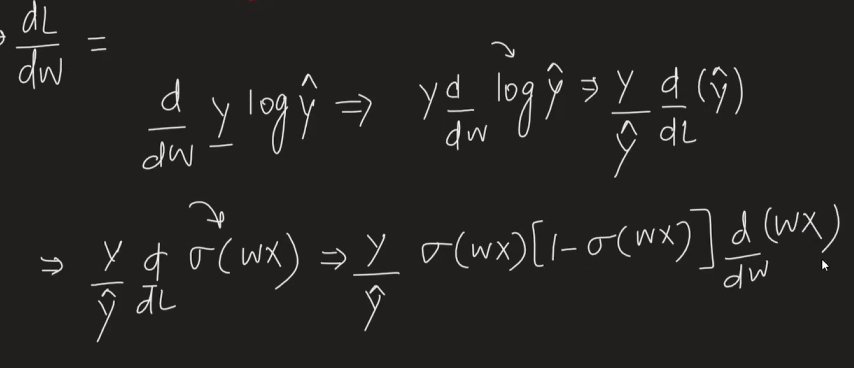




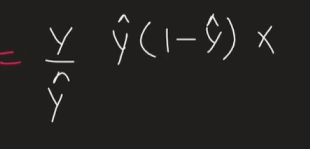


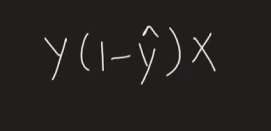
Finding dL/dw:

First part :

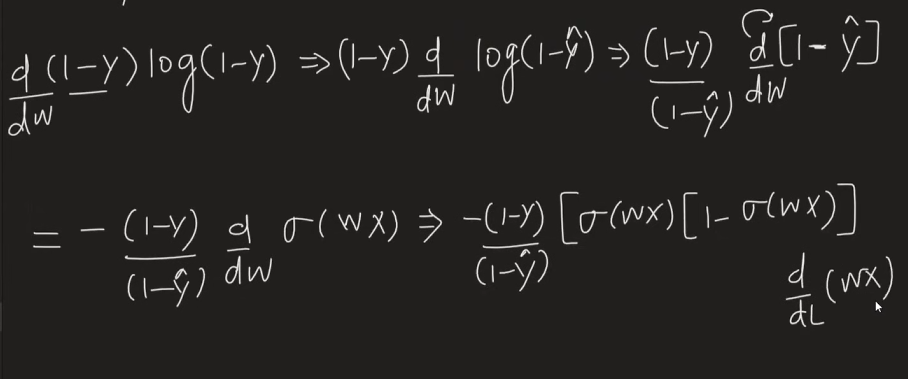


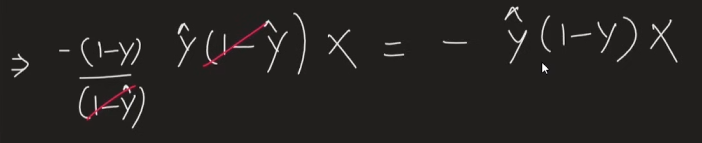
Simplifying,



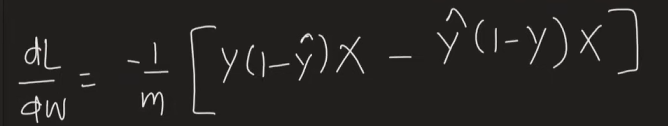


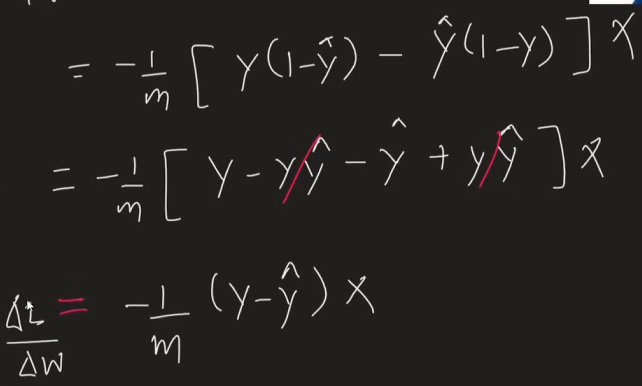
Second Part:



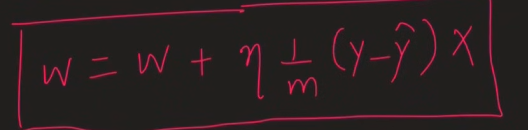


Combining both 1 and 2





So now the update equation of w is



Were,

