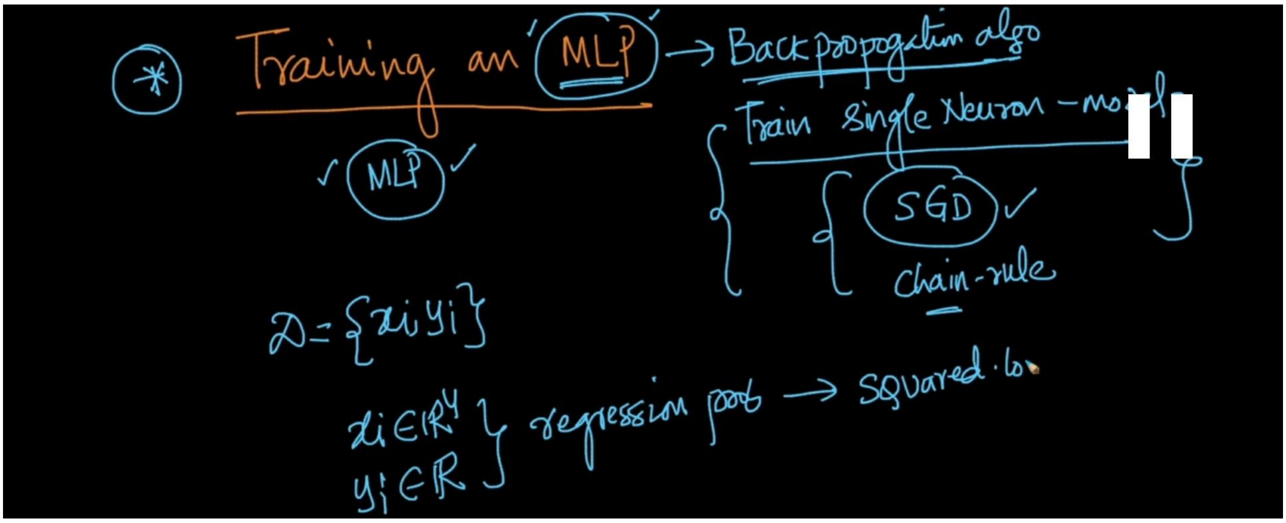
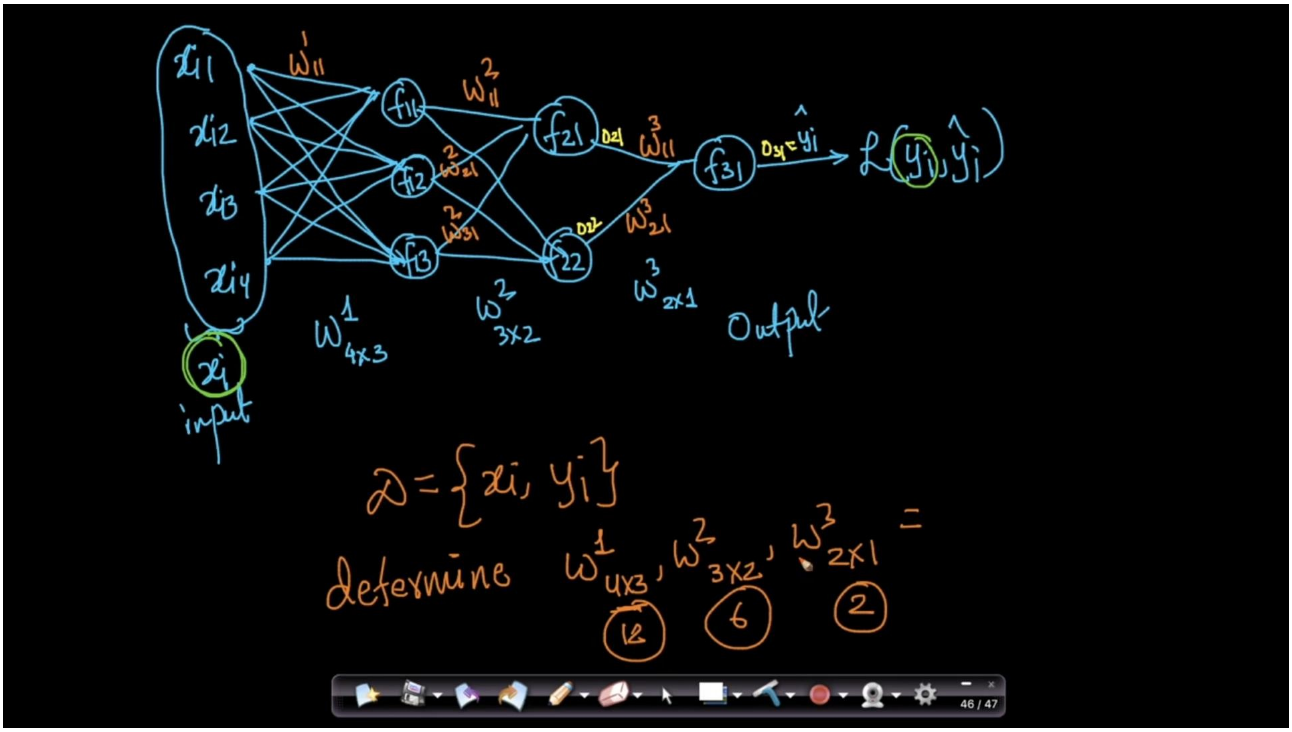
We’re using squared loss for our regression problem



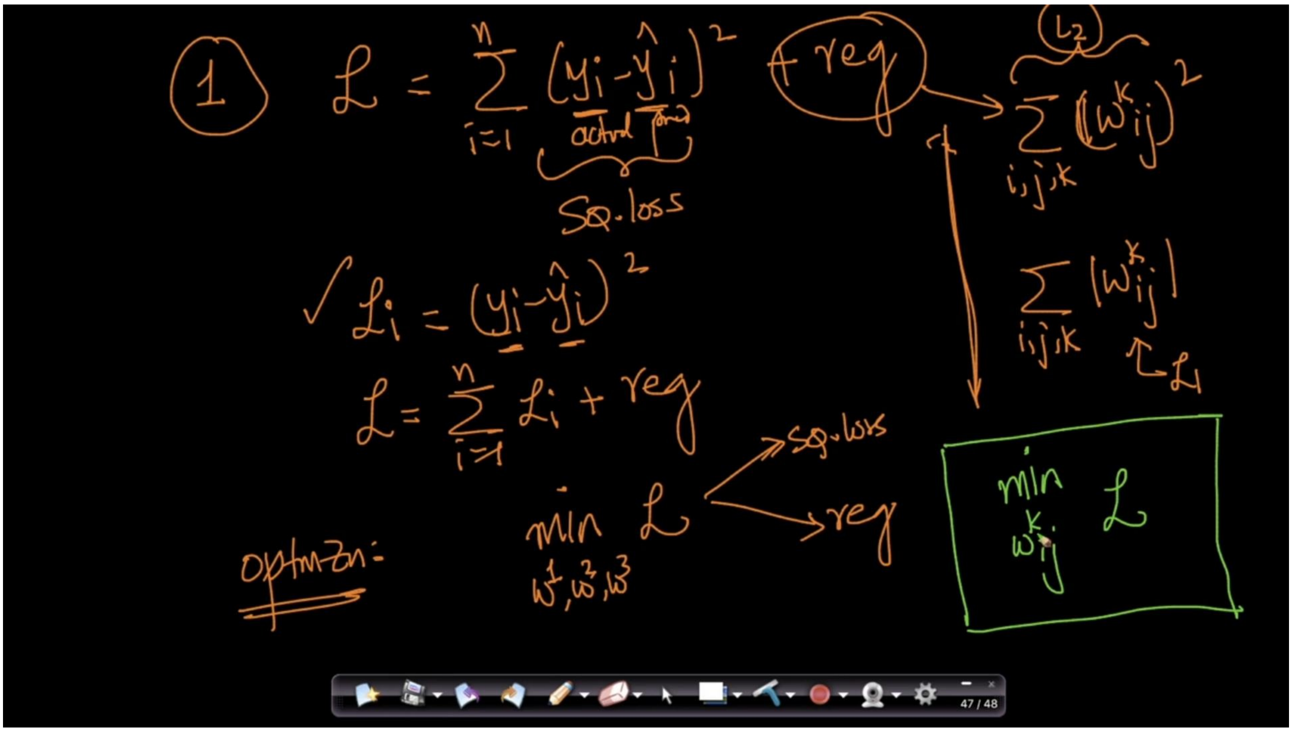


we have to determine the weights above, w1 has 12 weights, w2 has 6, w3 has 2 weights

total of 20 weights

training an MLP means computing all these weights

the loss function is given below, we can choose any of the given regularisers, we ignore for it for now for simplicity :

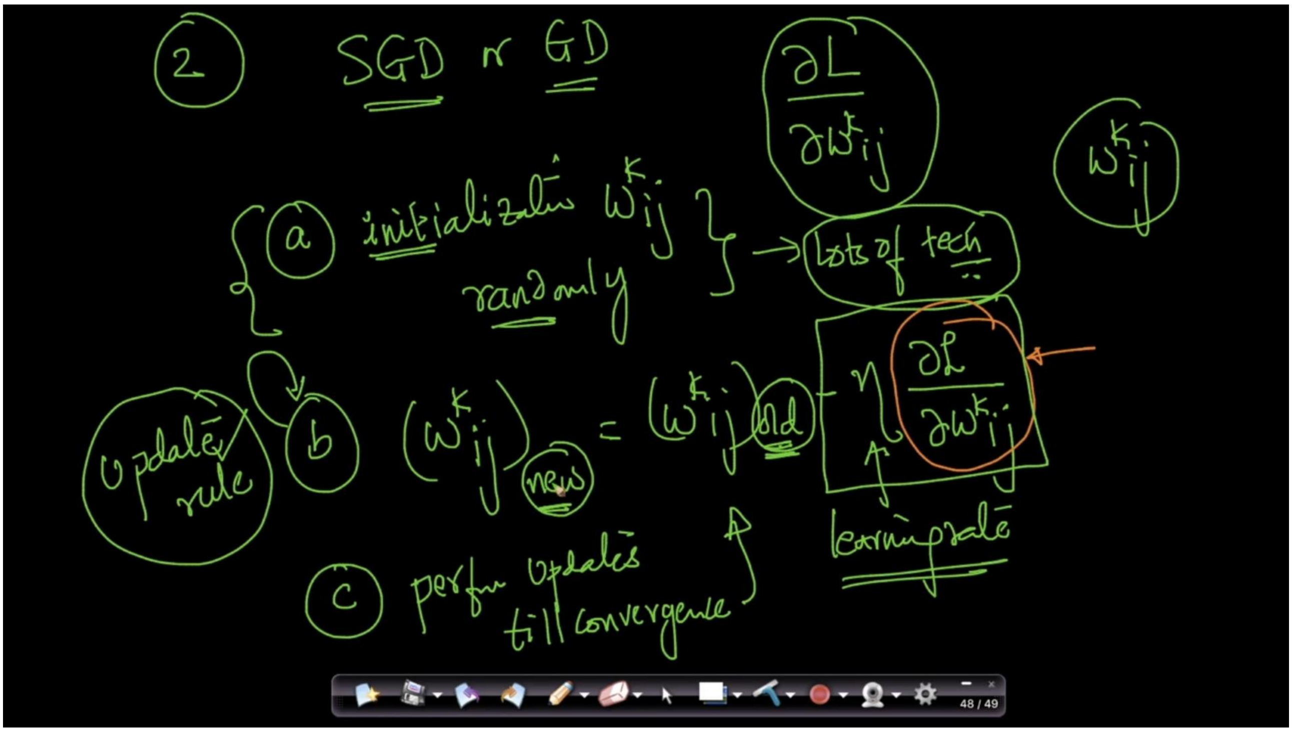


the text in green in above slide summarises it at once

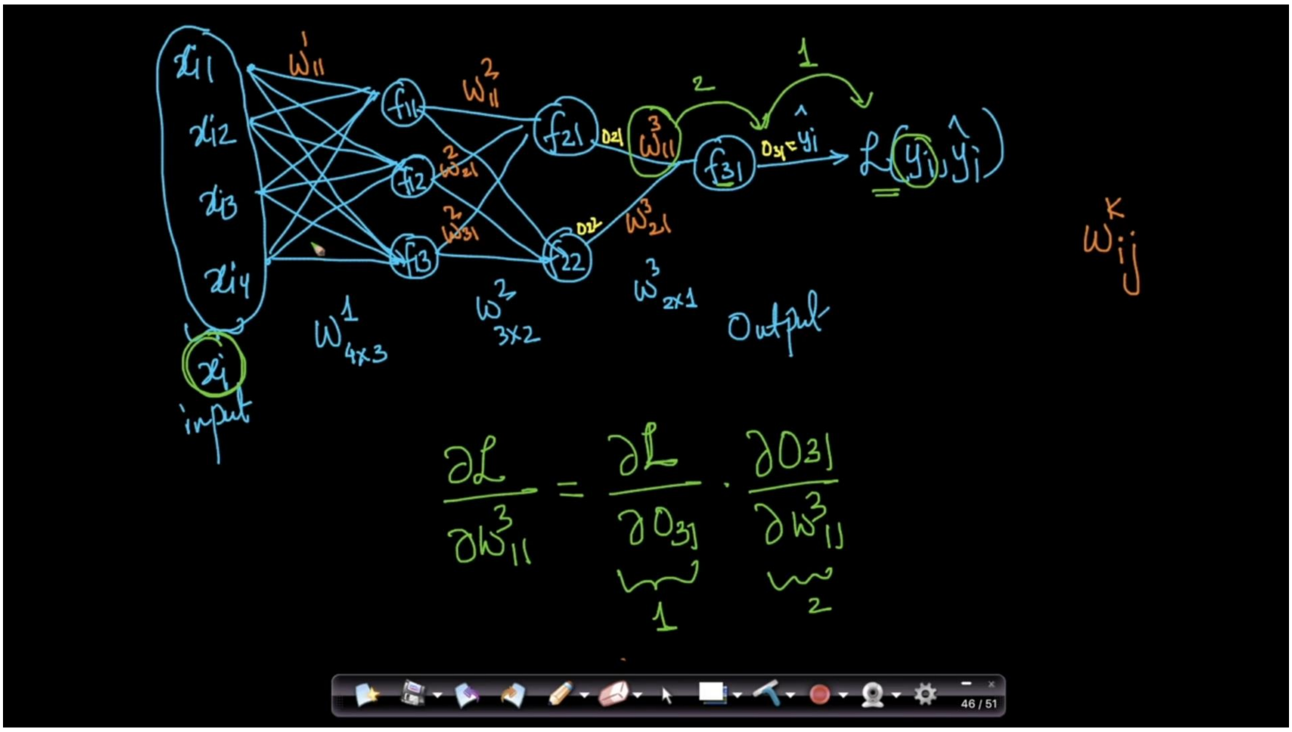
2nd step is we do GD or SGD

in SGD we initialise variables 1st randomly, there are techniques todo it in a better way

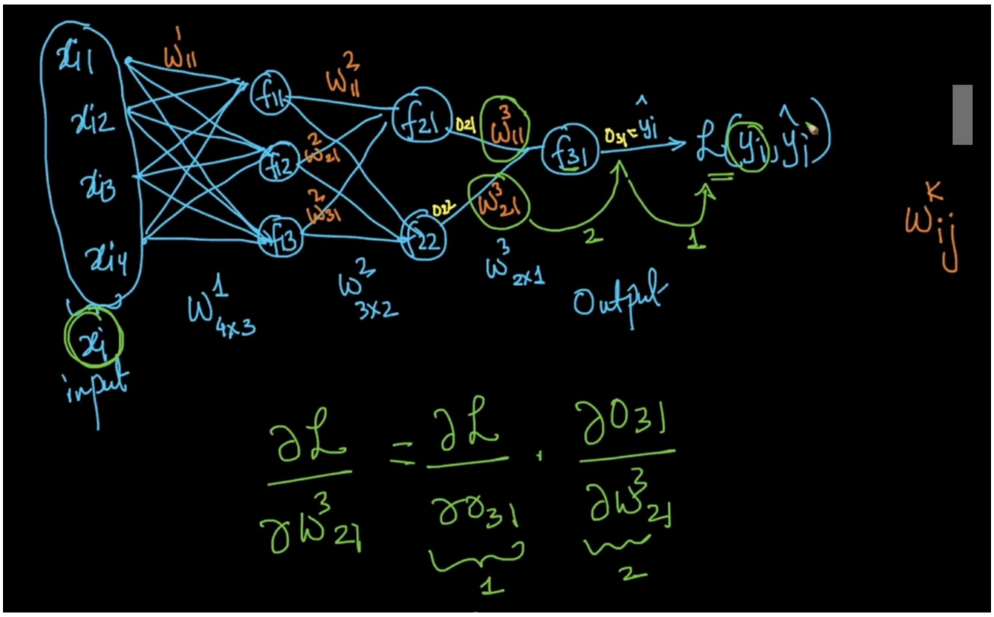
at iteration 0, w_{ij}^{k} will have a random value

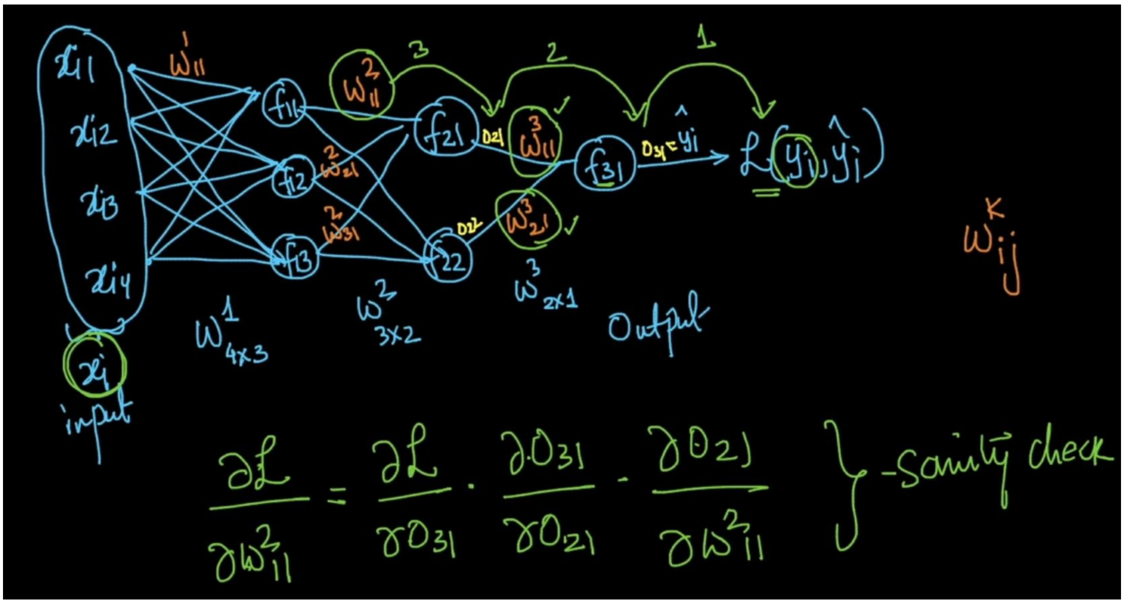


a very imp thing is we compute this partial derivative, which is circled above



above is chain rule





W^2

$$\frac{\partial L}{\partial W^3_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial W^2_{11}}$$

$$\frac{\partial L}{\partial W^2_{21}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial W^2_{21}}$$

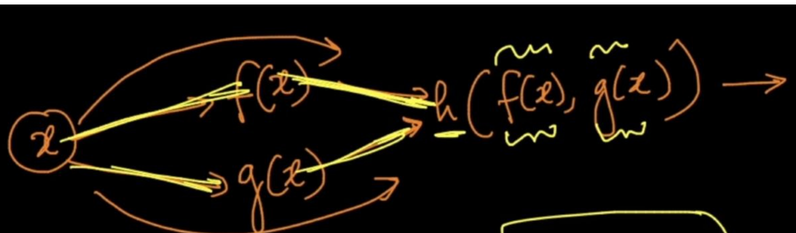
$$\frac{\partial L}{\partial W^2_{31}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial W^2}$$

W^3

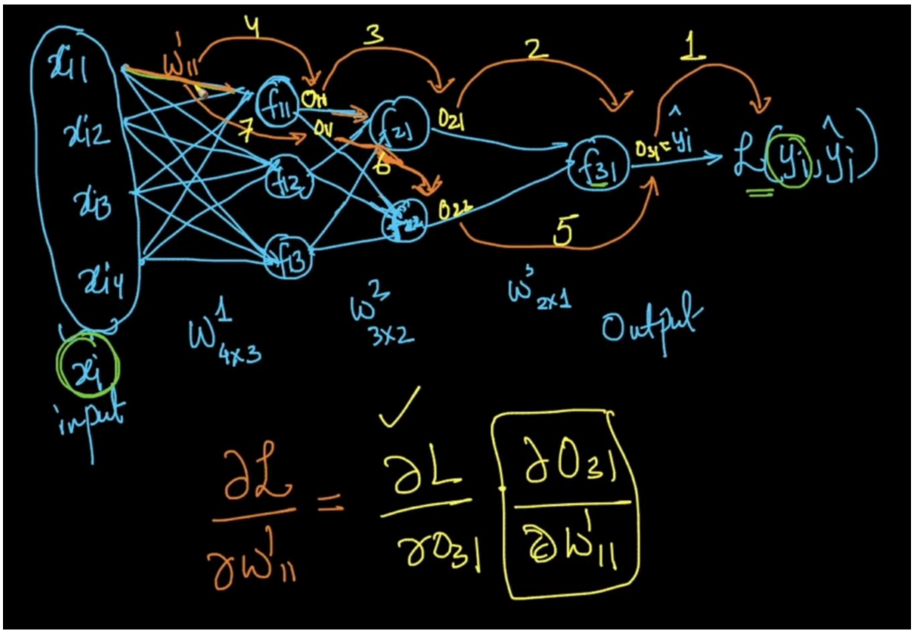
$$\frac{\partial L}{\partial W^3_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial W^3_{11}} \quad \leftarrow \text{chain rule}$$

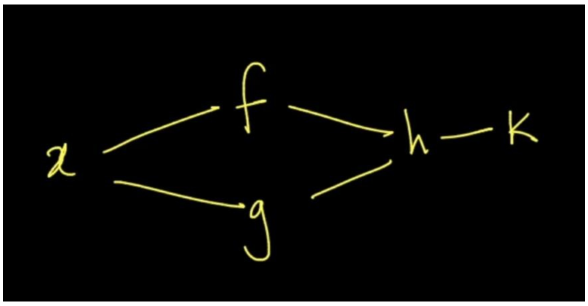
$$\frac{\partial L}{\partial W^3_{21}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial W^3_{21}} \quad \leftarrow \text{chain rule}$$

next case is more interesting since O11 is taking 2 paths

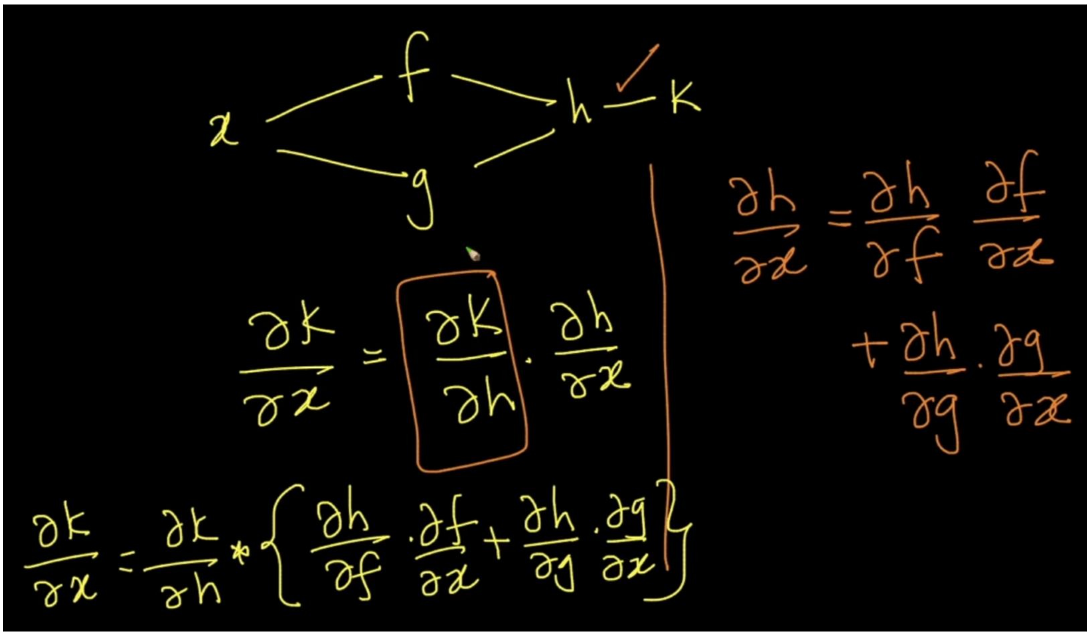


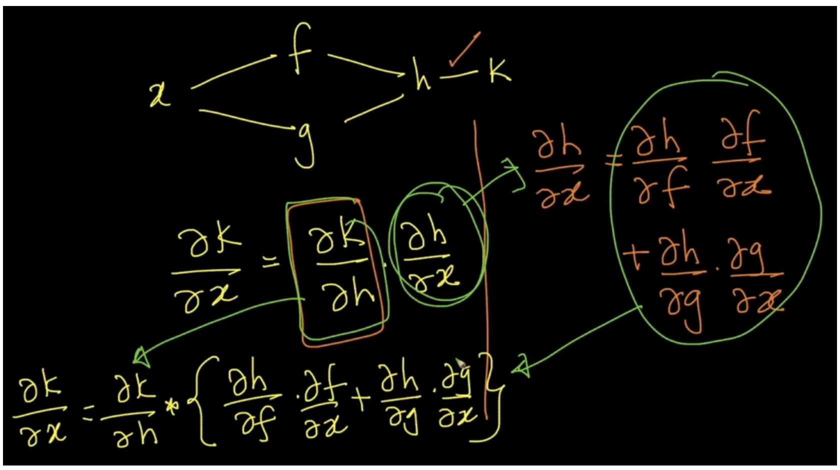
$$\frac{\partial h}{\partial x} = \boxed{\frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial x}} \underbrace{+}_{\text{SUM}} \boxed{\frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x}} \quad \leftarrow \text{chain rule}$$

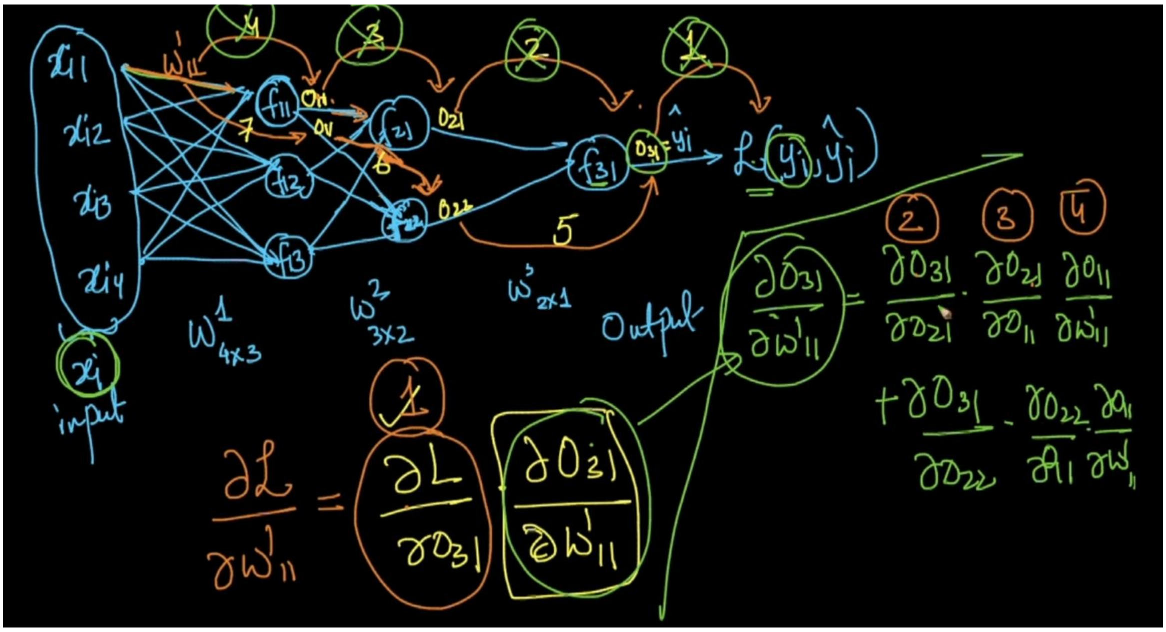


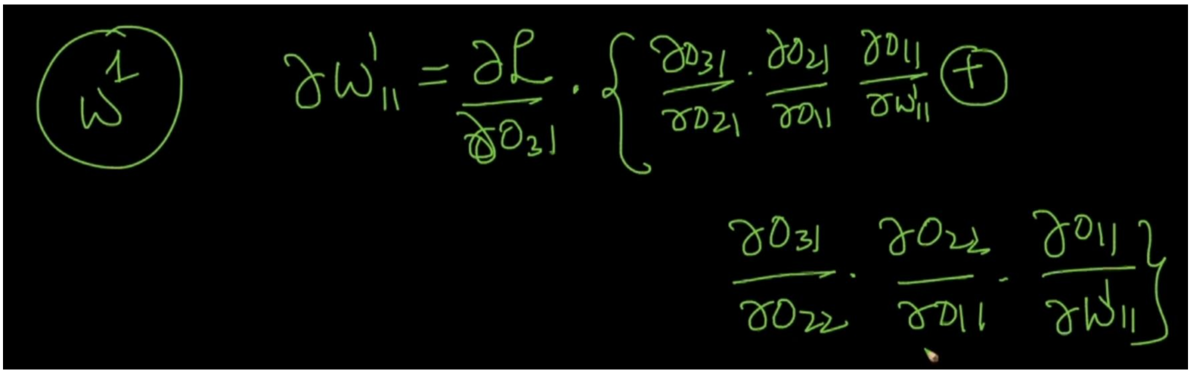


this is the structure that we have



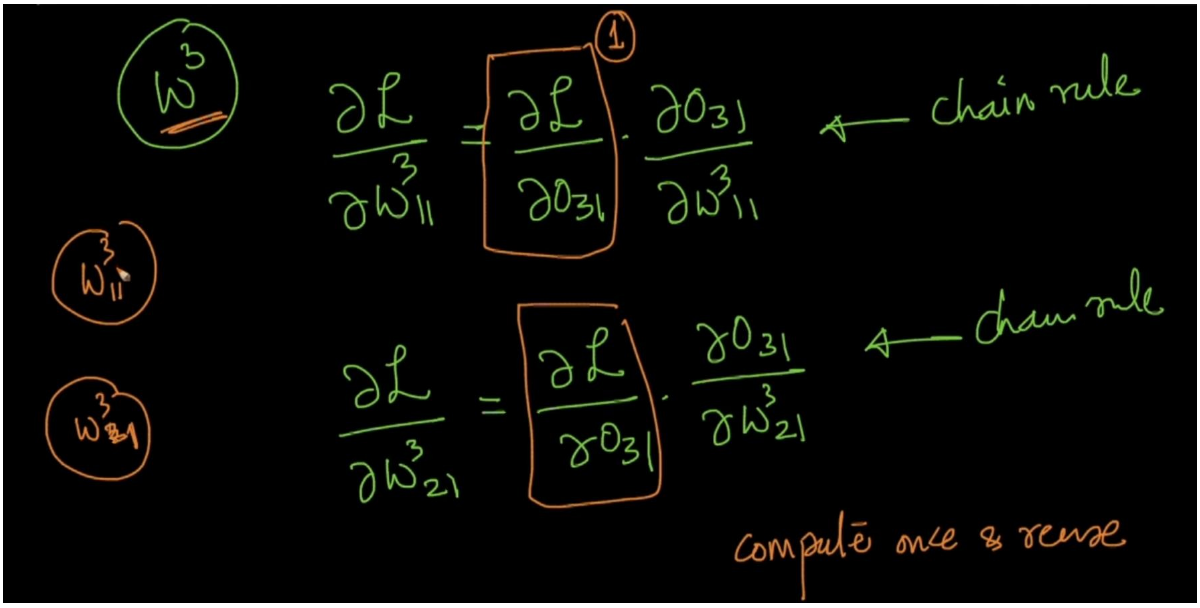


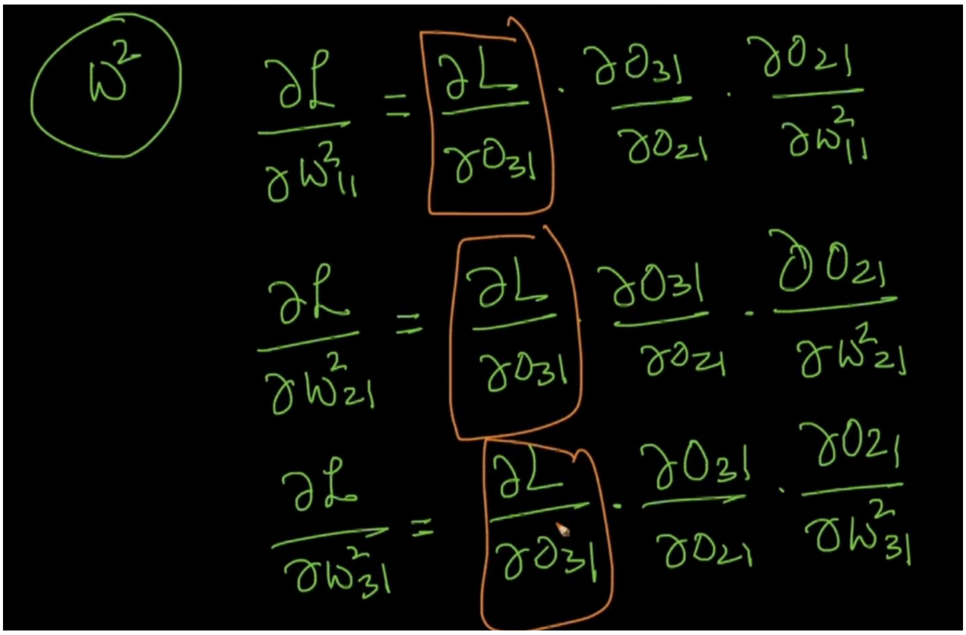




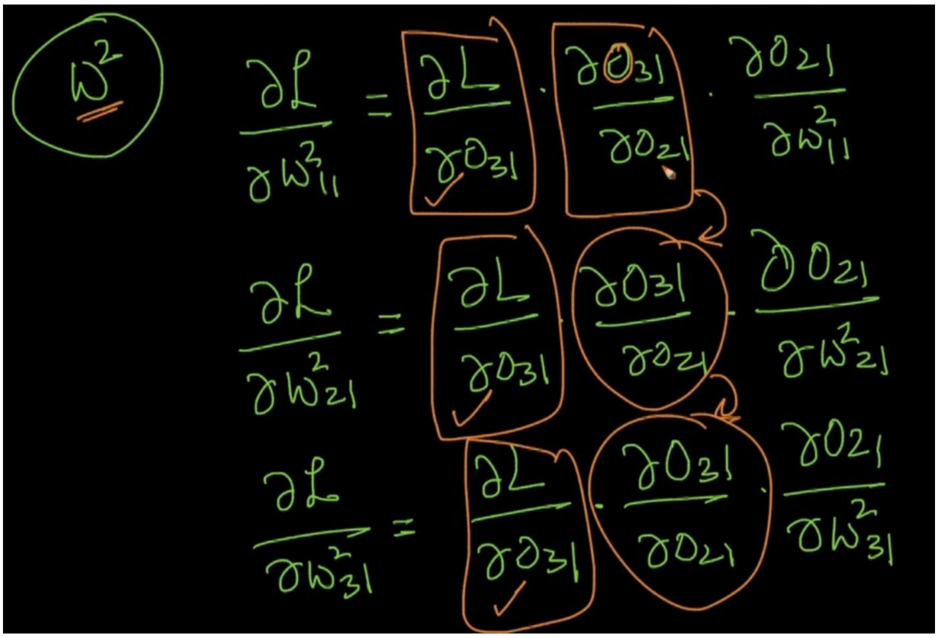
Memoisation :

saving stuff we’re recomputing

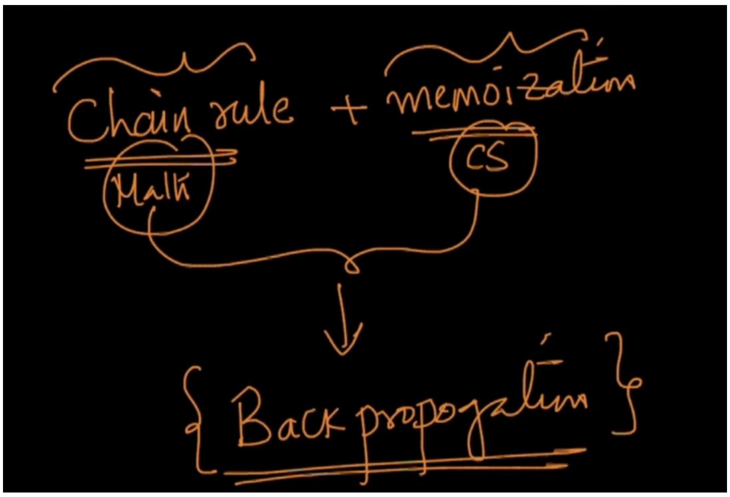




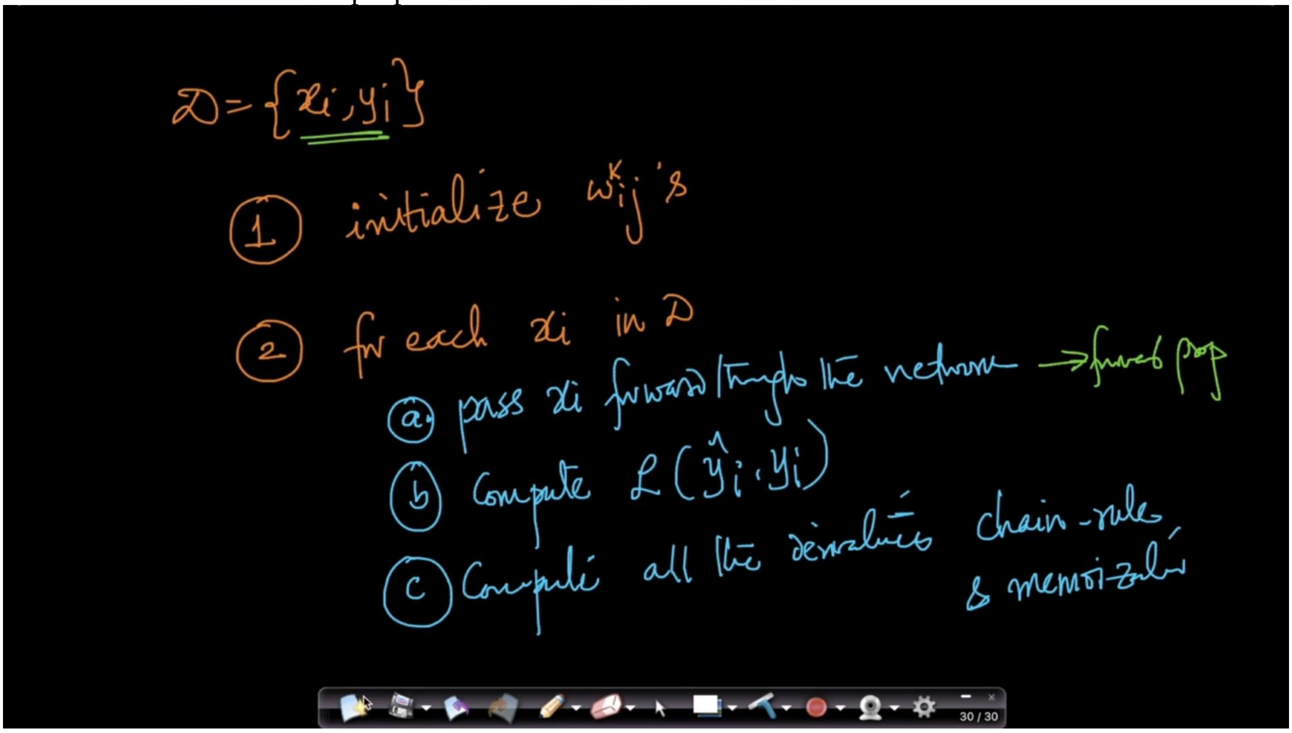
a lot of stuff is used again and again

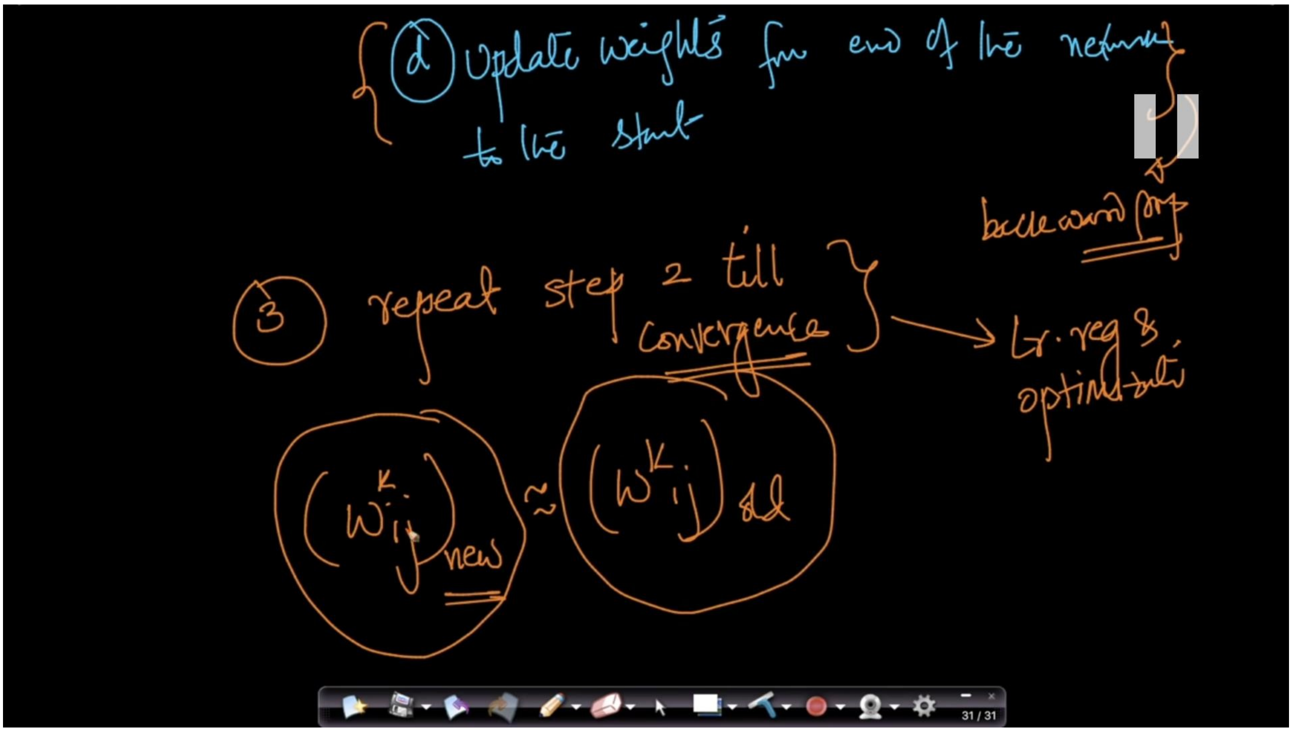


for slightly more memory, we get a huge speedup



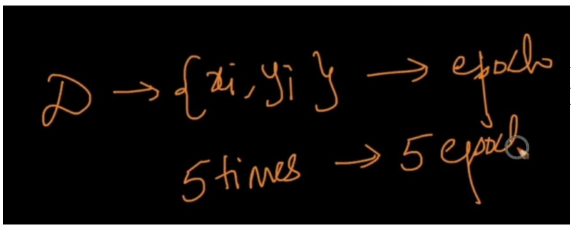
backprop explained :
2a below is called forward prop





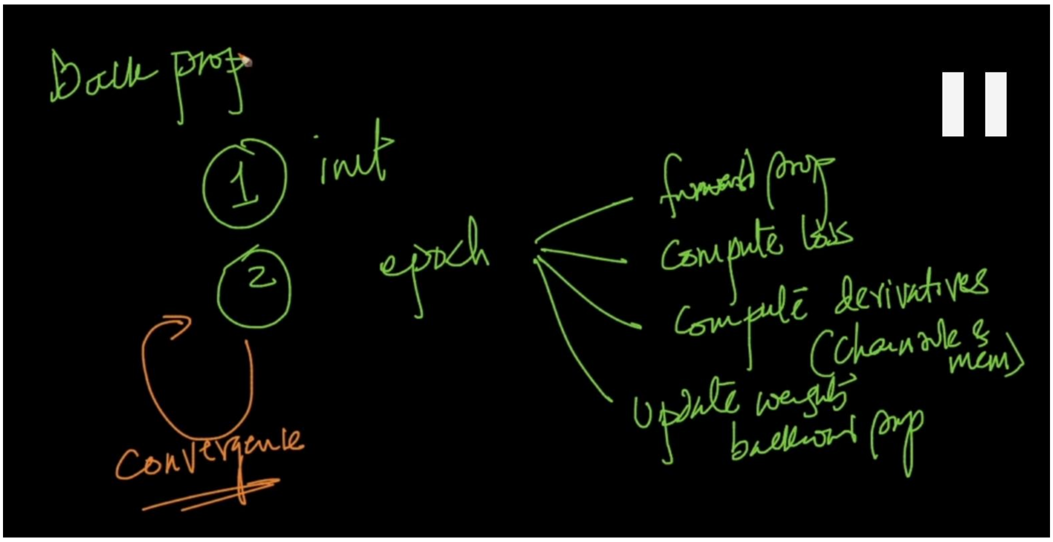
epoch means you've passed all the points in the dataset once through neural network

the number of times we pass them through the network is called epoch



passing = computing loss and updating weights we run for multiple epochs irl

how do we pick each of the point x_i in D : we should pick points uniformly at random
pick a point and do step 2



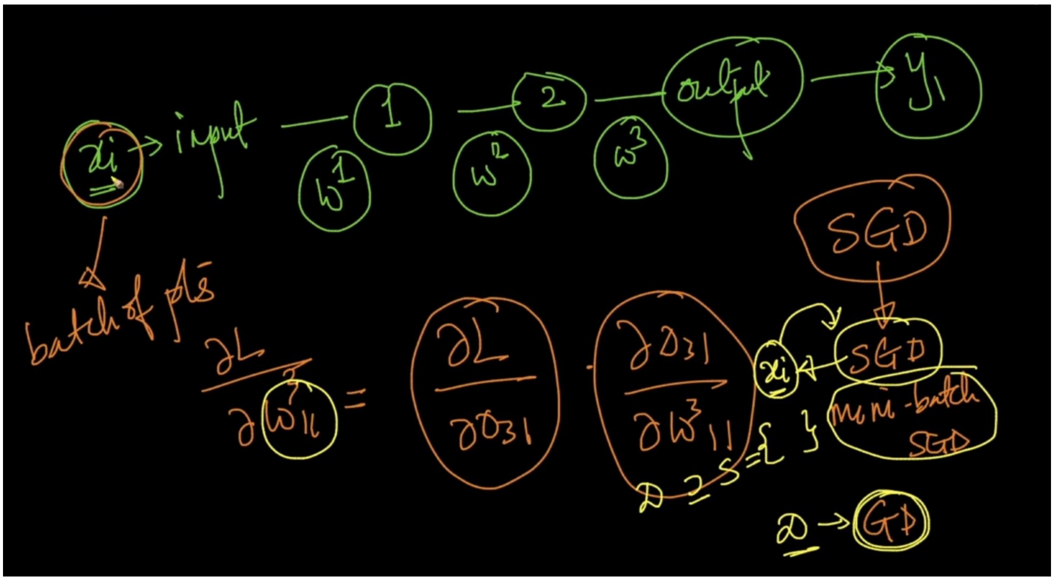
repeat step 2 till convergence

“back prop is a multi epoch training methodology where we leverage chain rule and memoisation to update weights”

backprop only works if activation functions are differentiable

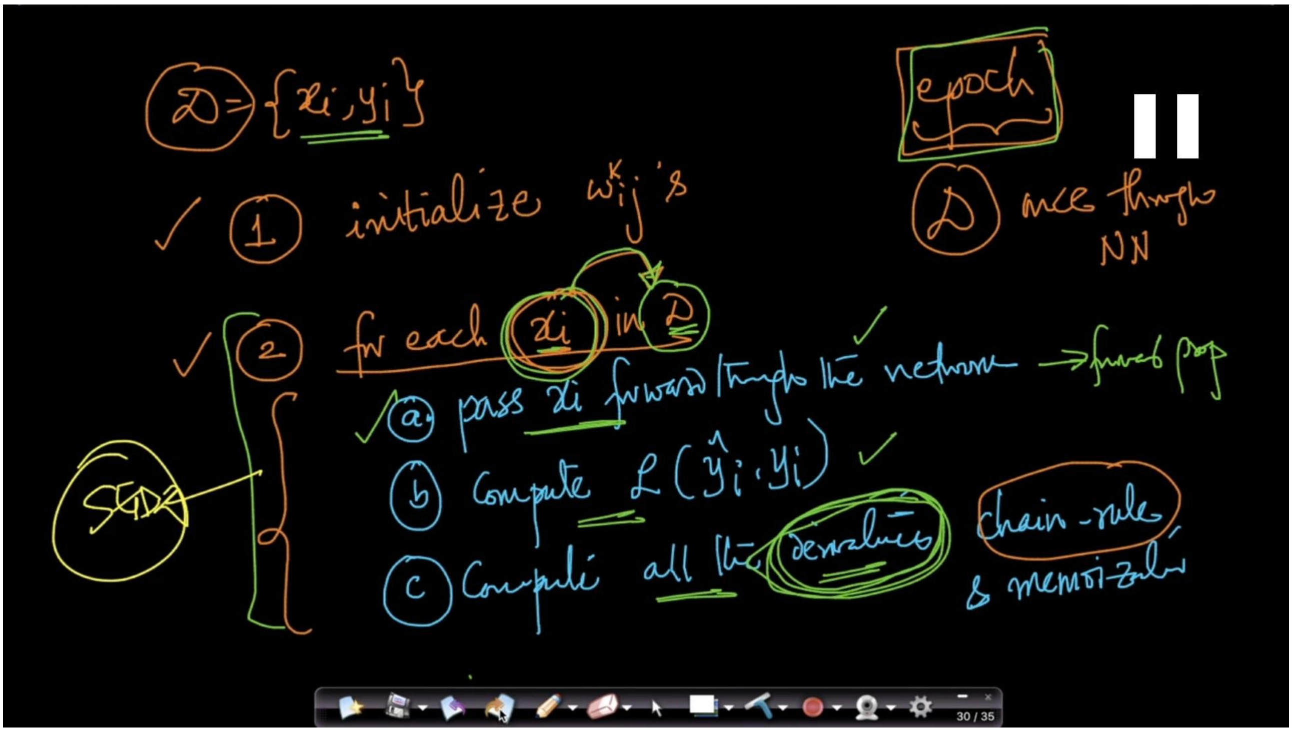
if it’s easy and fast differentiable, you can speedup the training of the neural network using backprop algo

----
in SGD you take one point at a time to compute derivates to update weights
in mini batch sgd you take a set of points
in gradient descent you take all of the points



----

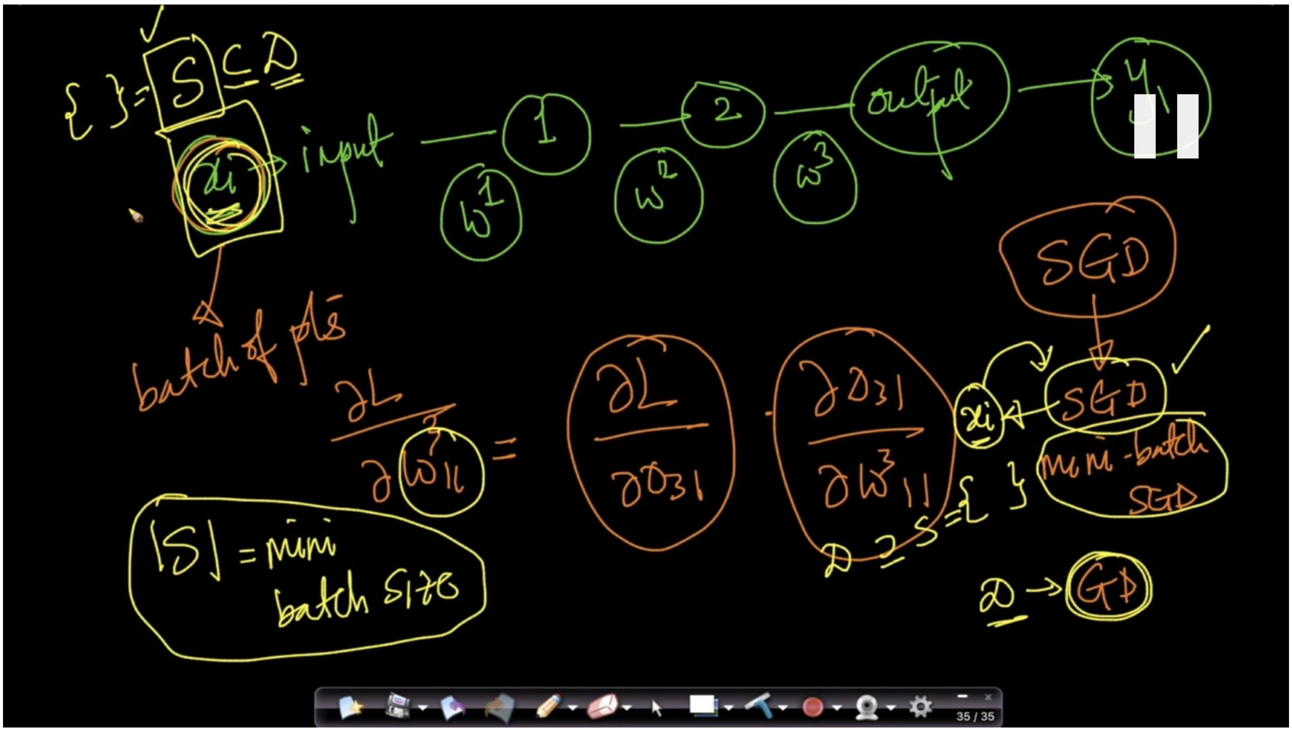
below is sort of like sgd because we’re sending one point each



the derivates above are computed using just one point
this is one approach

----

instead of sending 1 point, we send a set of points
the size S is called mini batch size



we can also do gradient descent based approach where we send all datapoints

the big problem is keeping all datapoints in RAM and computing derivates using the whole data can be extremely time consuming

people usually take one point at a time or mini batch based SGD(most popular)

----

example :

the possible ram sizes are shown below
we'll have to run the loop 100 times to complete 1 epoch
this is more efficient than running the loop 10k times with one point each