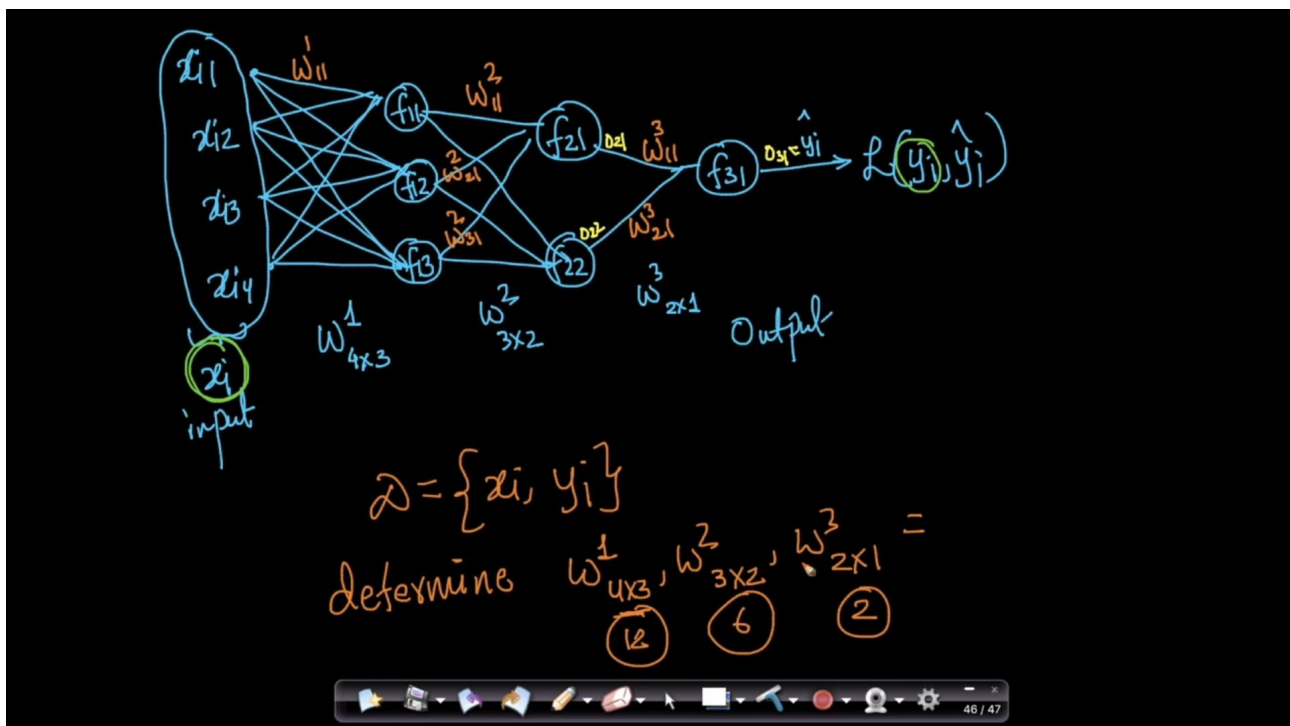
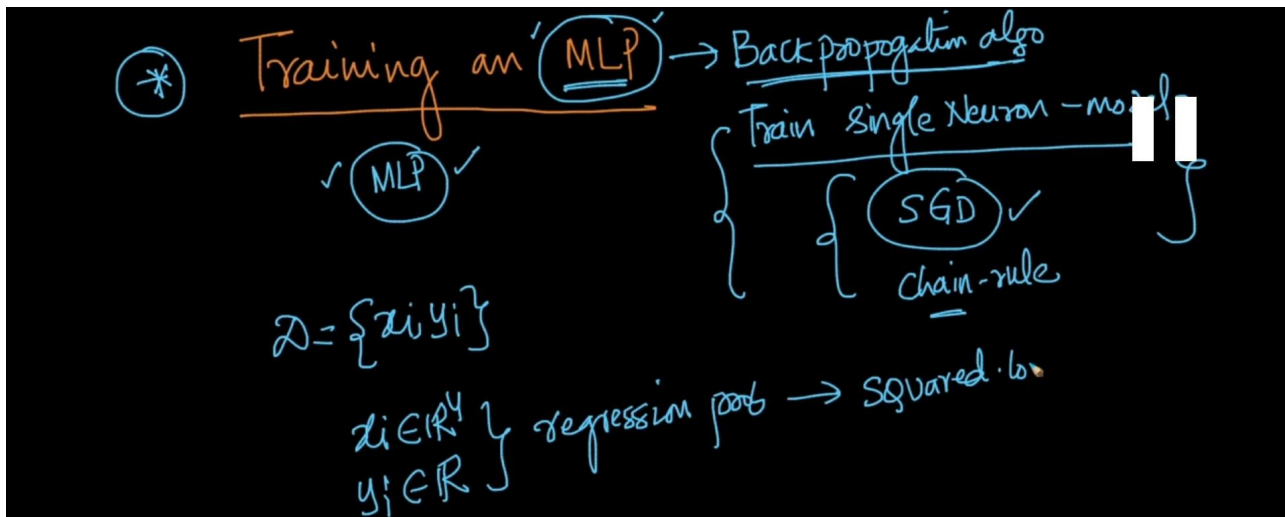


We're using squared loss for our regression problem



we have to determine the weights above, w_1 has 12 weights, w_2 has 6, w_3 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 :

① $L = \sum_{i=1}^n \underbrace{(y_i - \hat{y}_i)^2}_{\text{act. loss}} + \text{reg}$

$\checkmark L_i = (y_i - \hat{y}_i)^2$

$L = \sum_{i=1}^n L_i + \text{reg}$

Optimizing: $\min_{w^1, w^2, w^3} L$

$\text{reg} \rightarrow \sum_{i,j,k} (w_{ij}^k)^2$

$\sum_{i,j,k} |w_{ij}^k| \rightarrow L_1$

$\min_{w_{ij}^k} L$

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 to do it in a better way

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

② SGD or GD

(a) initialize w_{ij}^k randomly → lots of tech

(b) $(w_{ij}^k)_{\text{new}} = (w_{ij}^k)_{\text{old}} - \eta \frac{\partial L}{\partial w_{ij}^k}$

(c) perform updates till convergence

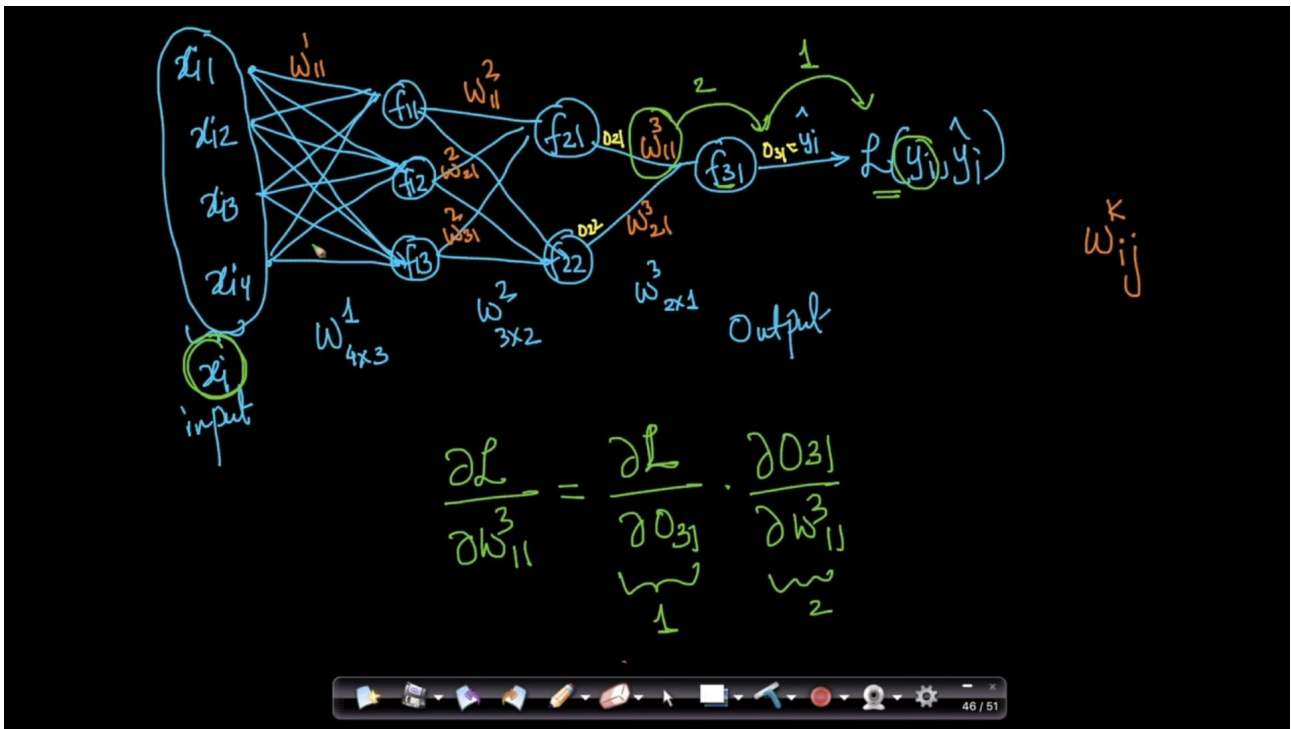
Update rule

learning rate

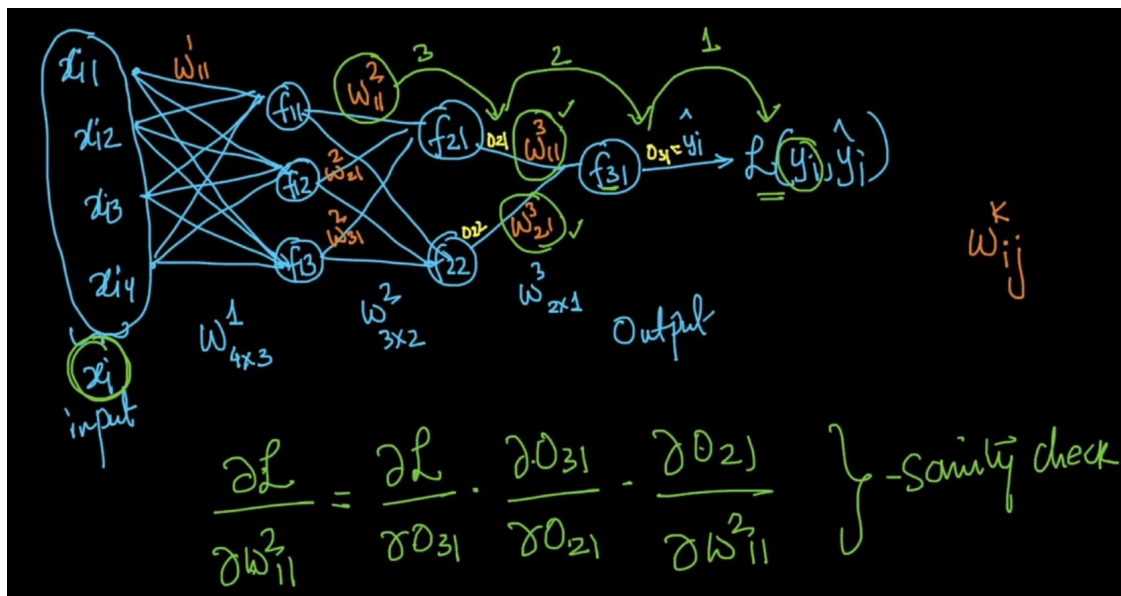
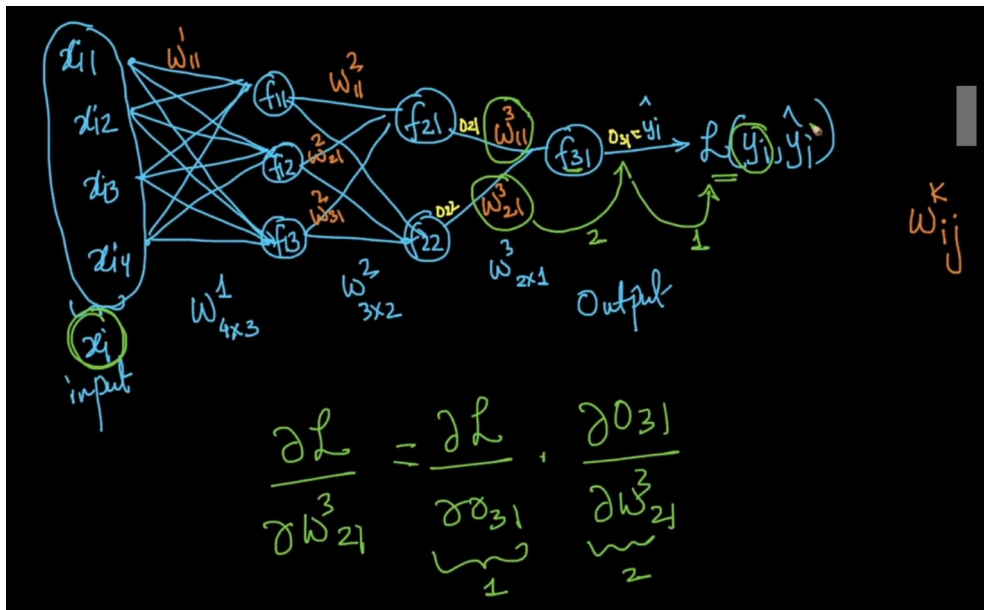
$\frac{\partial L}{\partial w_{ij}^k}$

w_{ij}^k

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



above is chain rule



$\odot \hat{w}^2$

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial w_{11}^2}$$

$$\frac{\partial L}{\partial w_{21}^2} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial w_{21}^2}$$

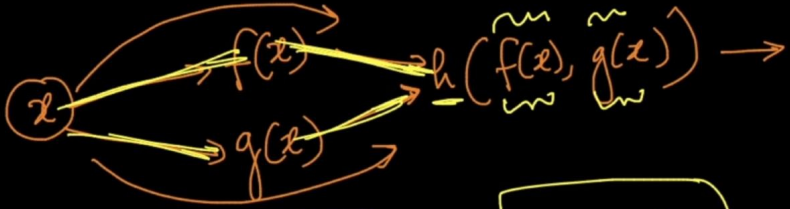
$$\frac{\partial L}{\partial w_{31}^2} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial w_{31}^2}$$

$\odot \hat{w}^3$

$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial w_{11}^3} \quad \leftarrow \text{chain rule}$$

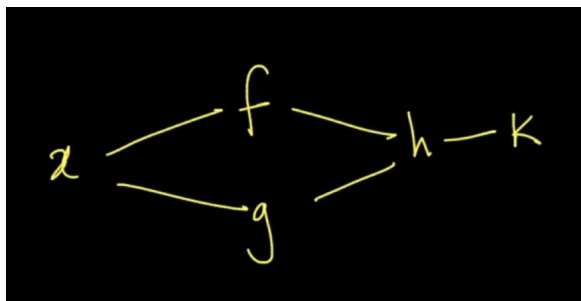
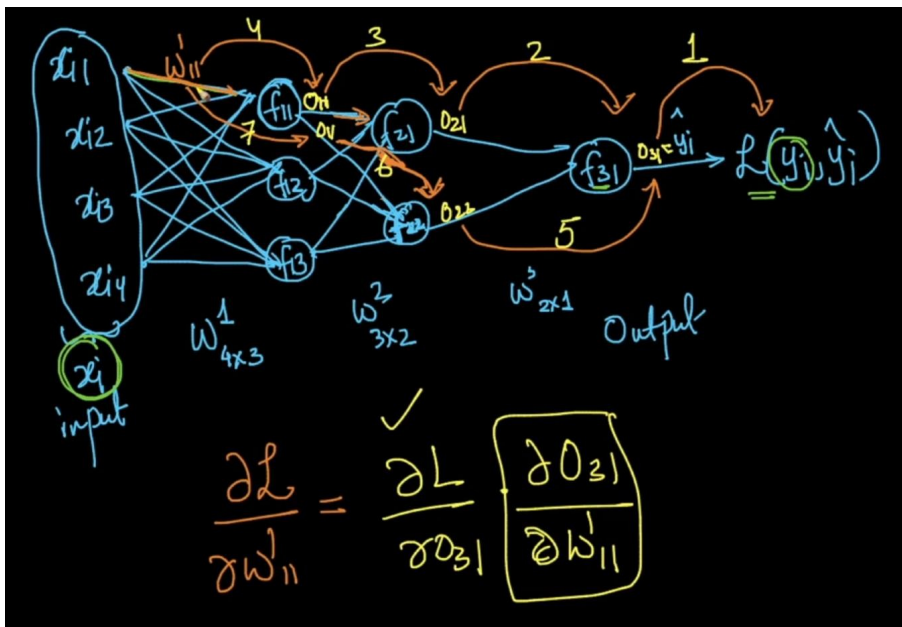
$$\frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial w_{21}^3} \quad \leftarrow \text{chain rule}$$

next case is more interesting since O11 is taking 2 paths



$$\frac{\partial h}{\partial x} = \left[\frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial x} \right] + \left[\frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x} \right] \quad \leftarrow \text{chain rule}$$

SUM



this is the structure that we have

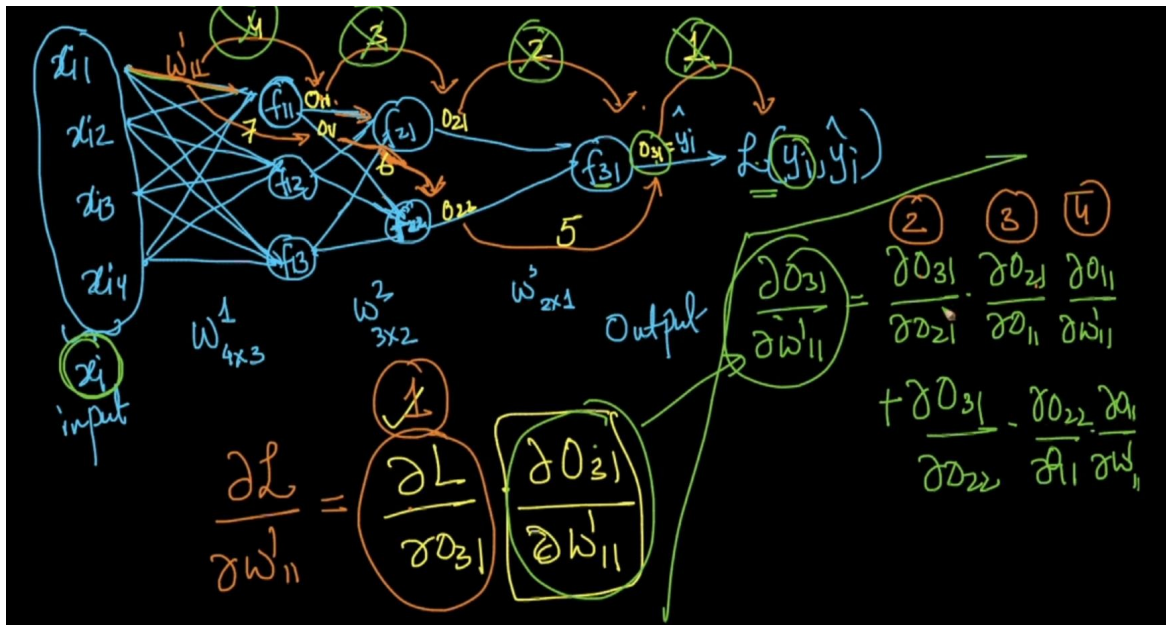
The graph shows $x \rightarrow f, g \rightarrow h \rightarrow k$. The partial derivative of k with respect to x is calculated as:

$$\frac{\partial k}{\partial x} = \frac{\partial k}{\partial h} \cdot \frac{\partial h}{\partial x}$$

$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial f} \frac{\partial f}{\partial x} + \frac{\partial h}{\partial g} \frac{\partial g}{\partial x}$$

$$\frac{\partial k}{\partial x} = \frac{\partial k}{\partial h} * \left\{ \frac{\partial h}{\partial f} \frac{\partial f}{\partial x} + \frac{\partial h}{\partial g} \frac{\partial g}{\partial x} \right\}$$

This block shows the same equations as the previous one, but with arrows indicating the flow of derivatives from the output k back through the functions h, f, g to the input x . The terms $\frac{\partial k}{\partial h}$ and $\frac{\partial h}{\partial x}$ are circled, and arrows point from the final equation back to the intermediate ones.



Derivative calculation for w_{11} :

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial o_{31}} \left[\frac{\partial o_{31}}{\partial w_{11}} \right]$$

$$\frac{\partial o_{31}}{\partial w_{11}} = \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_{11}} + \frac{\partial o_{31}}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_{11}}$$

Memoisation :

saving stuff we're recomputing

Derivative calculation for w_{11}^3 :

$$\frac{\partial L}{\partial w_{11}^3} = \boxed{\frac{\partial L}{\partial o_{31}}} \cdot \frac{\partial o_{31}}{\partial w_{11}^3} \quad \leftarrow \text{chain rule}$$

Derivative calculation for w_{21}^3 :

$$\frac{\partial L}{\partial w_{21}^3} = \boxed{\frac{\partial L}{\partial o_{31}}} \cdot \frac{\partial o_{31}}{\partial w_{21}^3} \quad \leftarrow \text{chain rule}$$

compute once & reuse

ω^2

$$\frac{\partial L}{\partial \omega_{11}^2} = \frac{\partial L}{\partial \theta_{31}} \cdot \frac{\partial \theta_{31}}{\partial \theta_{21}} \cdot \frac{\partial \theta_{21}}{\partial \omega_{11}^2}$$

$$\frac{\partial L}{\partial \omega_{21}^2} = \frac{\partial L}{\partial \theta_{31}} \cdot \frac{\partial \theta_{31}}{\partial \theta_{21}} \cdot \frac{\partial \theta_{21}}{\partial \omega_{21}^2}$$

$$\frac{\partial L}{\partial \omega_{31}^2} = \frac{\partial L}{\partial \theta_{31}} \cdot \frac{\partial \theta_{31}}{\partial \theta_{21}} \cdot \frac{\partial \theta_{21}}{\partial \omega_{31}^2}$$

a lot of stuff is
used again and
again

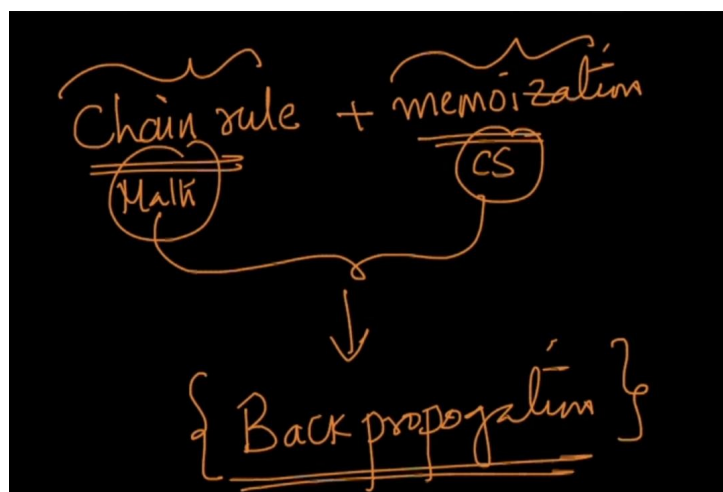
ω^2

$$\frac{\partial L}{\partial \omega_{11}^2} = \frac{\partial L}{\partial \theta_{31}} \cdot \frac{\partial \theta_{31}}{\partial \theta_{21}} \cdot \frac{\partial \theta_{21}}{\partial \omega_{11}^2}$$

$$\frac{\partial L}{\partial \omega_{21}^2} = \frac{\partial L}{\partial \theta_{31}} \cdot \frac{\partial \theta_{31}}{\partial \theta_{21}} \cdot \frac{\partial \theta_{21}}{\partial \omega_{21}^2}$$

$$\frac{\partial L}{\partial \omega_{31}^2} = \frac{\partial L}{\partial \theta_{31}} \cdot \frac{\partial \theta_{31}}{\partial \theta_{21}} \cdot \frac{\partial \theta_{21}}{\partial \omega_{31}^2}$$

for slightly more memory, we get a huge speedup

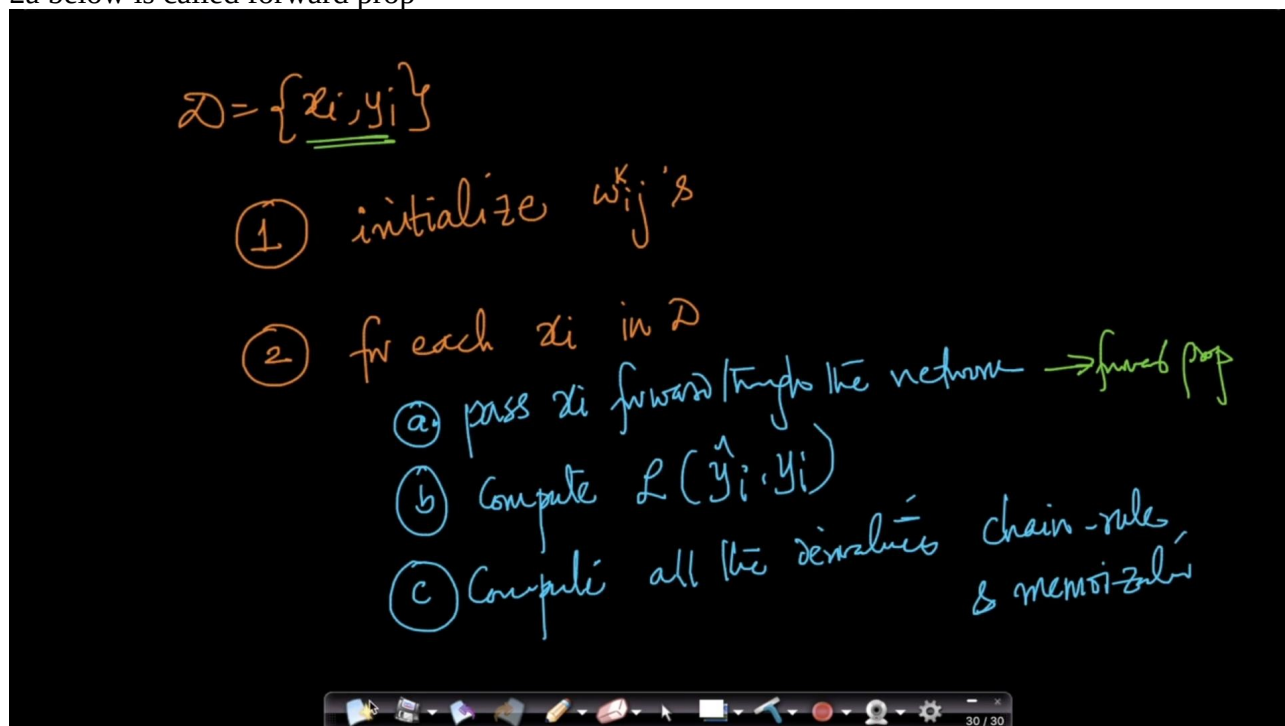


backprop explained :

2a below is called forward prop

$\mathcal{D} = \{\underline{x_i}, y_i\}$

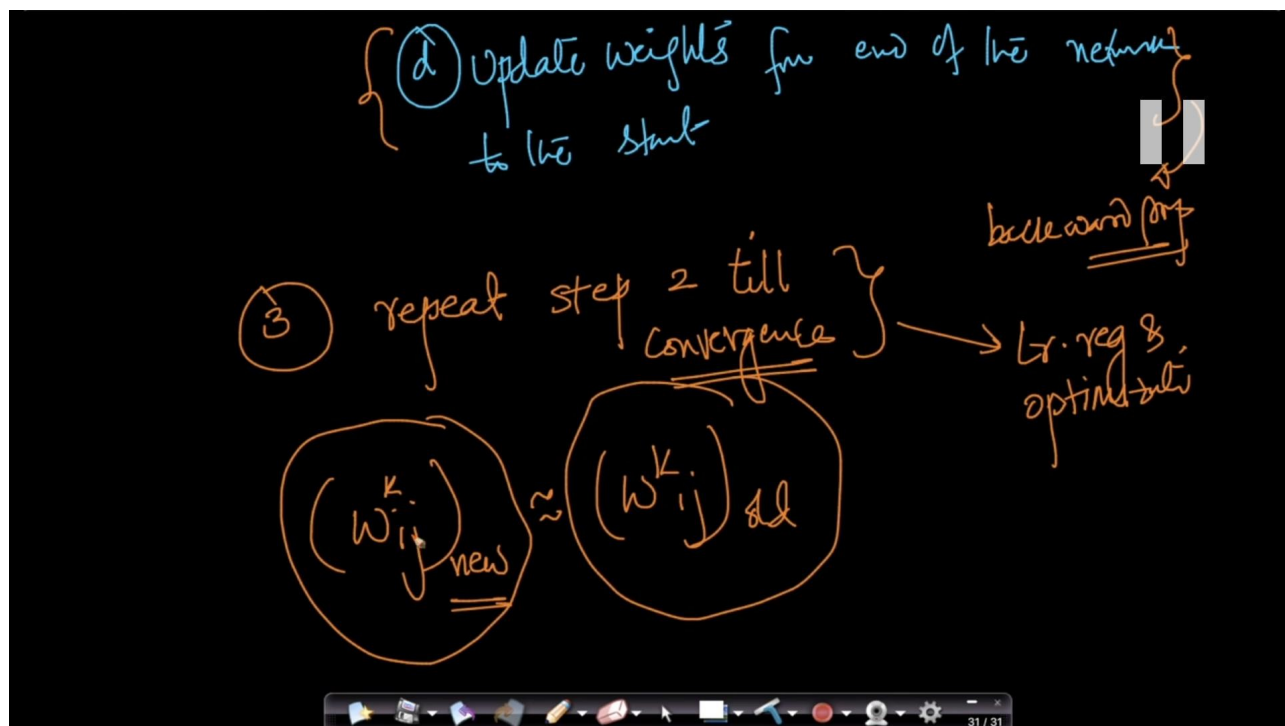
- ① initialize w_{ij}^k 's
- ② for each x_i in \mathcal{D}
 - a) pass x_i forward through the network \rightarrow forward prop
 - b) compute $L(\hat{y}_i, y_i)$
 - c) compute all the derivatives chain-rules & memorize



{ (d) Update weights from end of the network to the start } \rightarrow backward prop

③ repeat step 2 till convergence \rightarrow Lr, reg & optimization

$(w_{ij}^k)_{\text{new}} \approx (w_{ij}^k)_{\text{old}}$



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

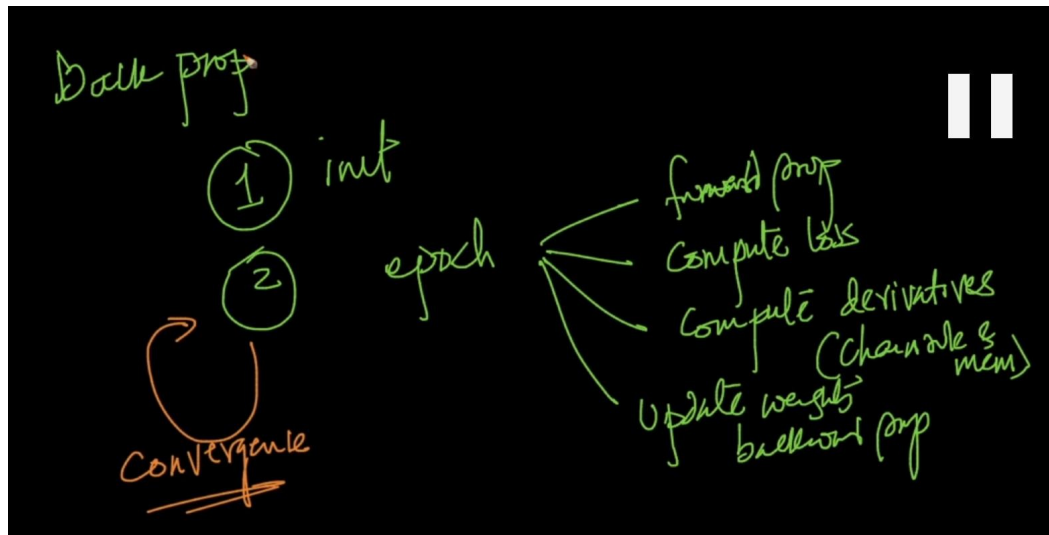
the number of times we pass them through the network is called epoch

$$\mathcal{D} \rightarrow \{x_i, y_i\} \rightarrow \text{epoch}$$
$$5 \text{ times} \rightarrow 5 \text{ epochs}$$

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

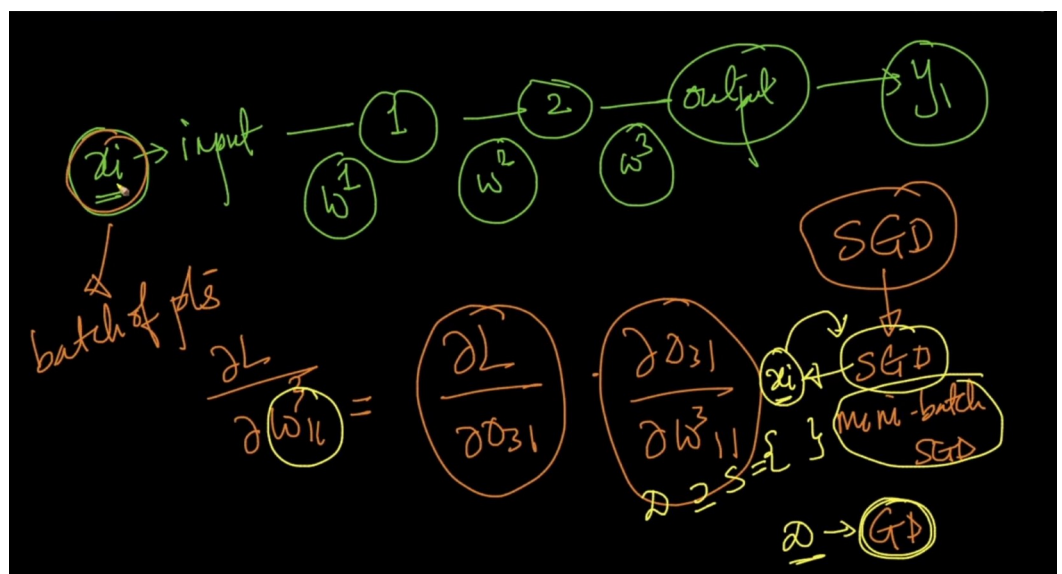
if it's easy and fast differentiable, you can speedup the training of the neural network using backprop algo

in SGD you take one point at a time to compute derivatives 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

$\mathcal{D} \Rightarrow \{\underline{x_i}, y_i\}$

✓ ① initialize w_{ij}^k 's

epoch

② once through NN

✓ ② for each $\underline{x_i}$ in $\underline{\mathcal{D}}$

✓ a. pass $\underline{x_i}$ forward through the network \rightarrow forward pass

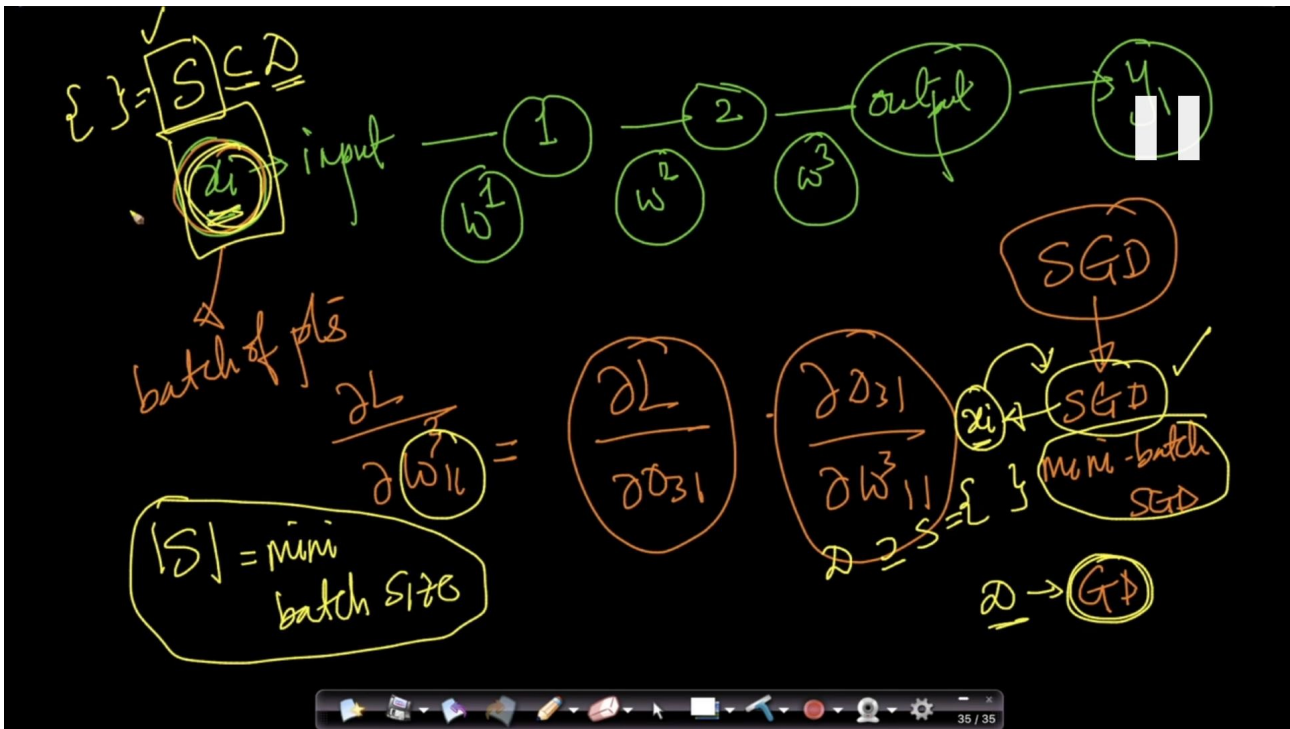
b. Compute $L(\hat{y_i}, y_i)$

c. Compute all the derivatives (chain rules & memoization)

SGD

the derivatives 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 derivatives 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

