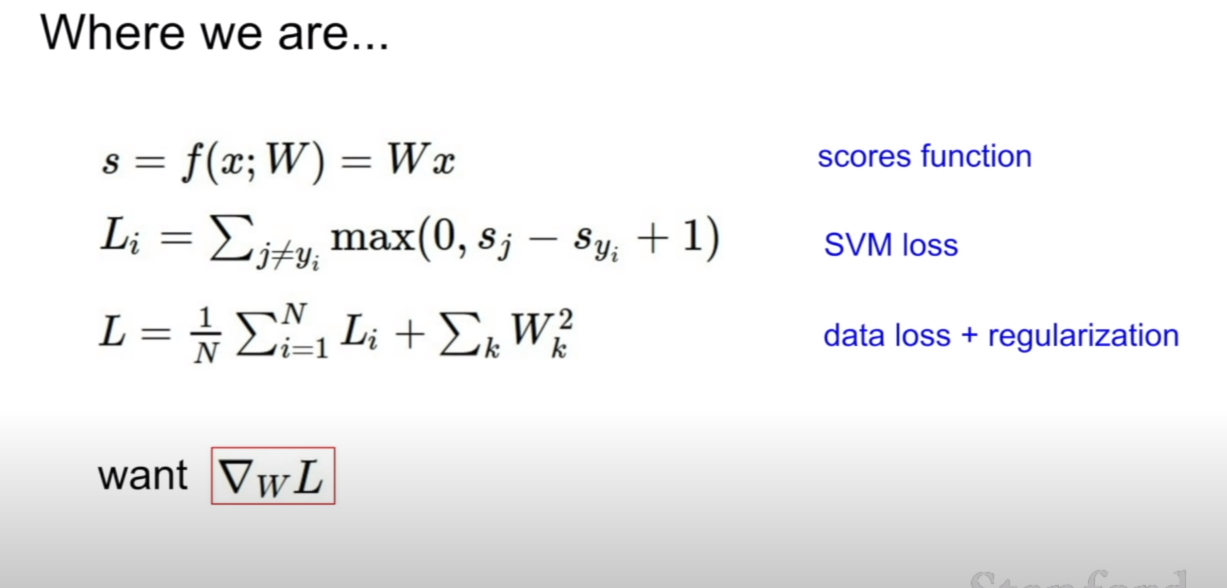
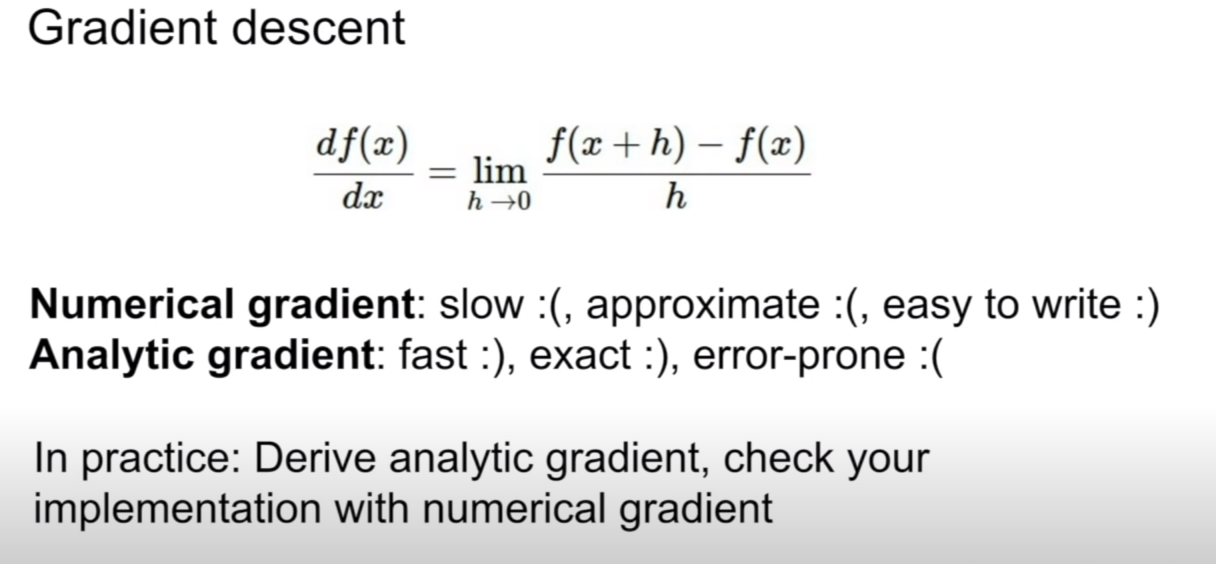
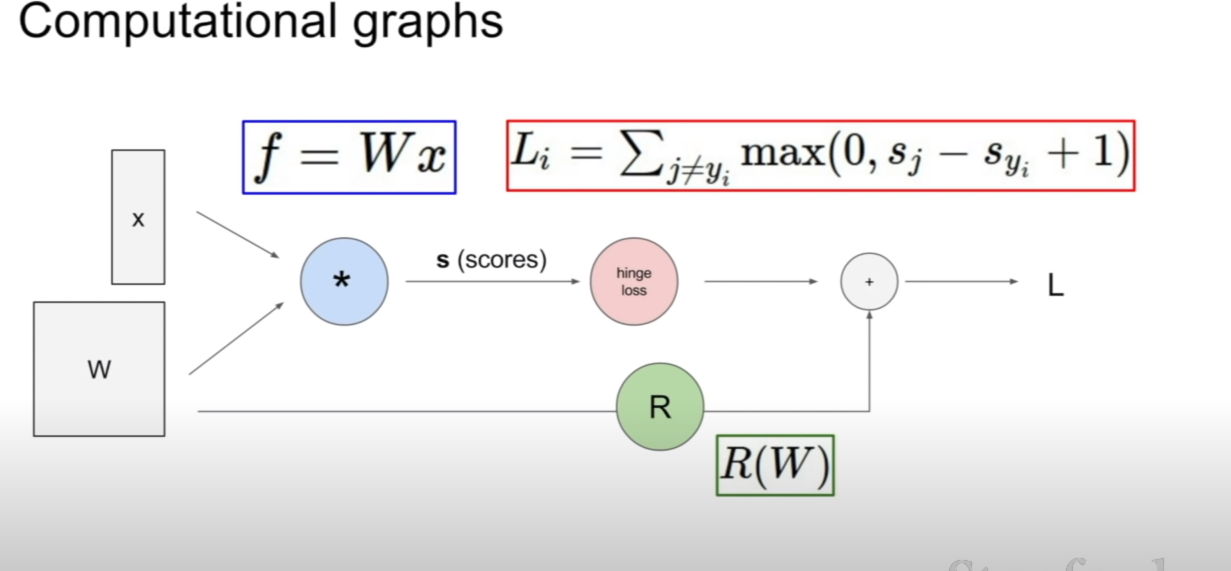
[Overview]

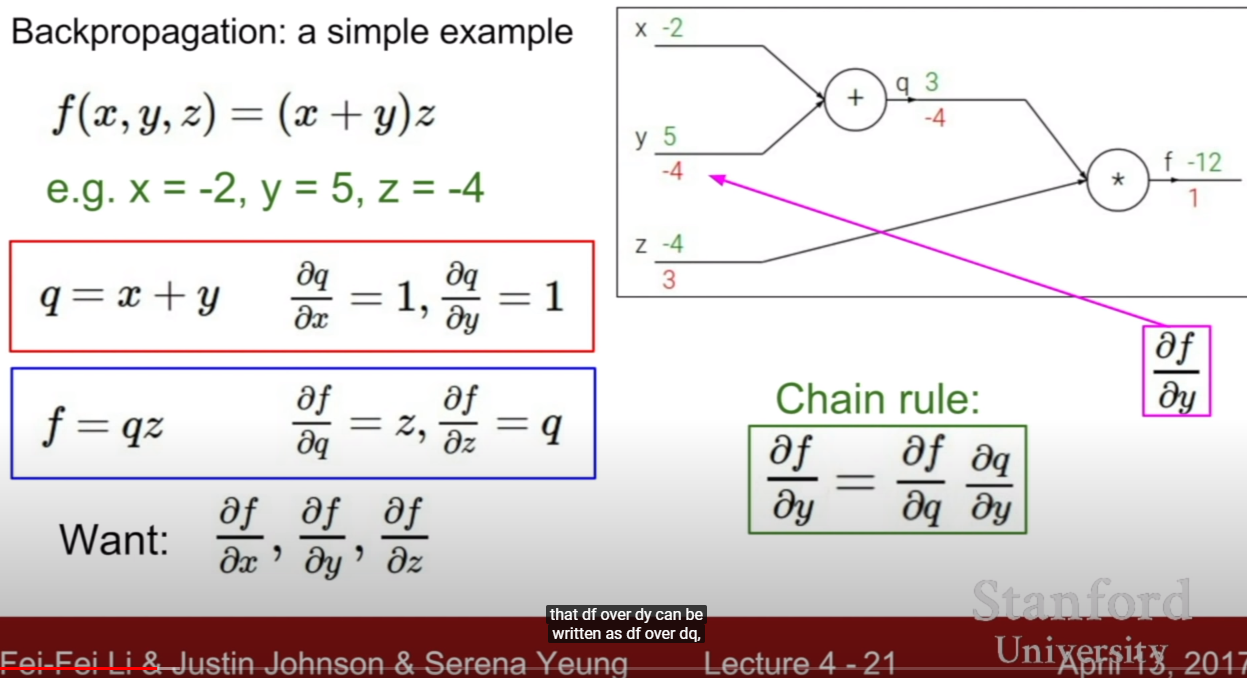




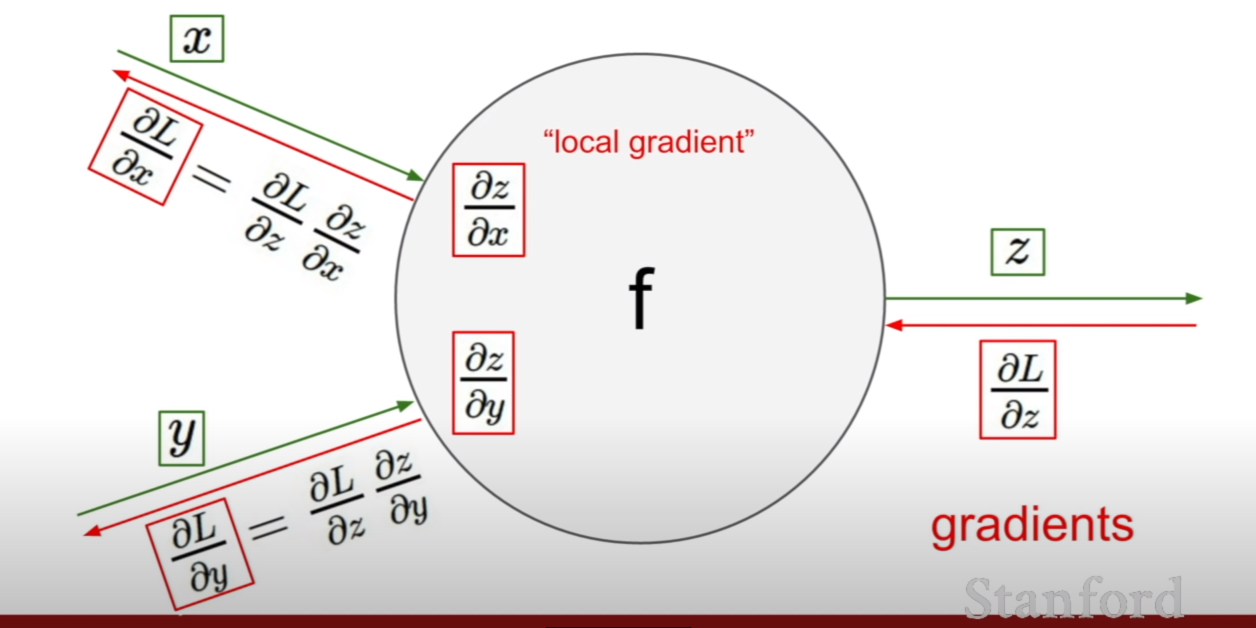


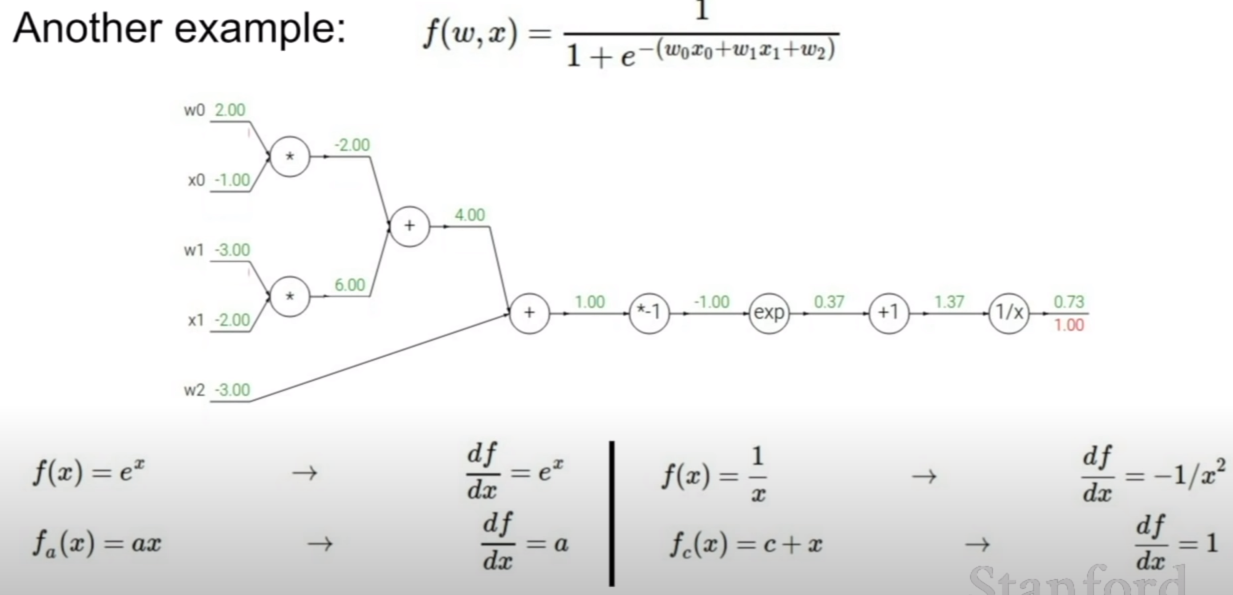
- this is the way to calculate loss.

[Backpropagation]



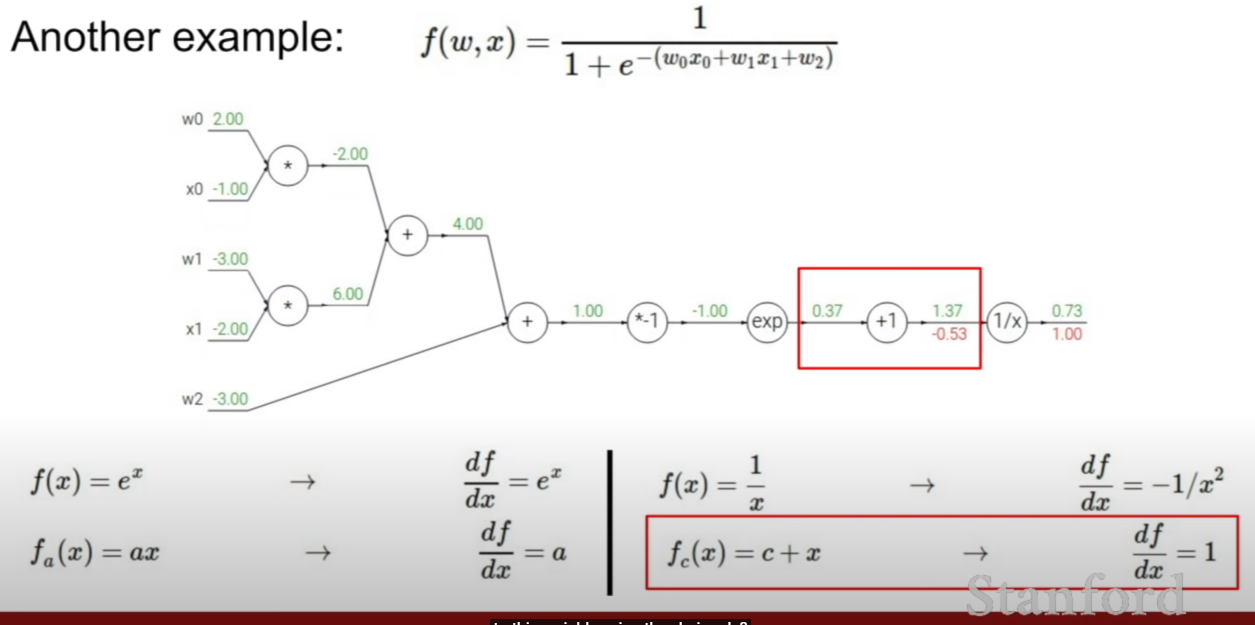
- using chaning rule, we can get df over dy like above.

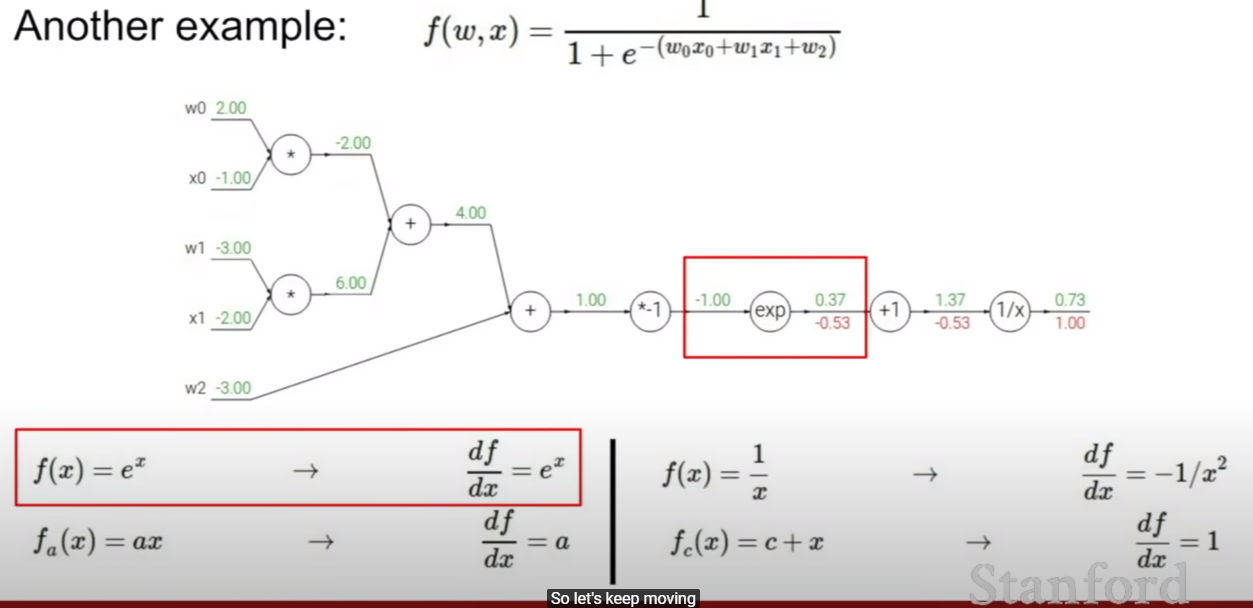
- by having each node’s local gradient, we could backpropagation easily.

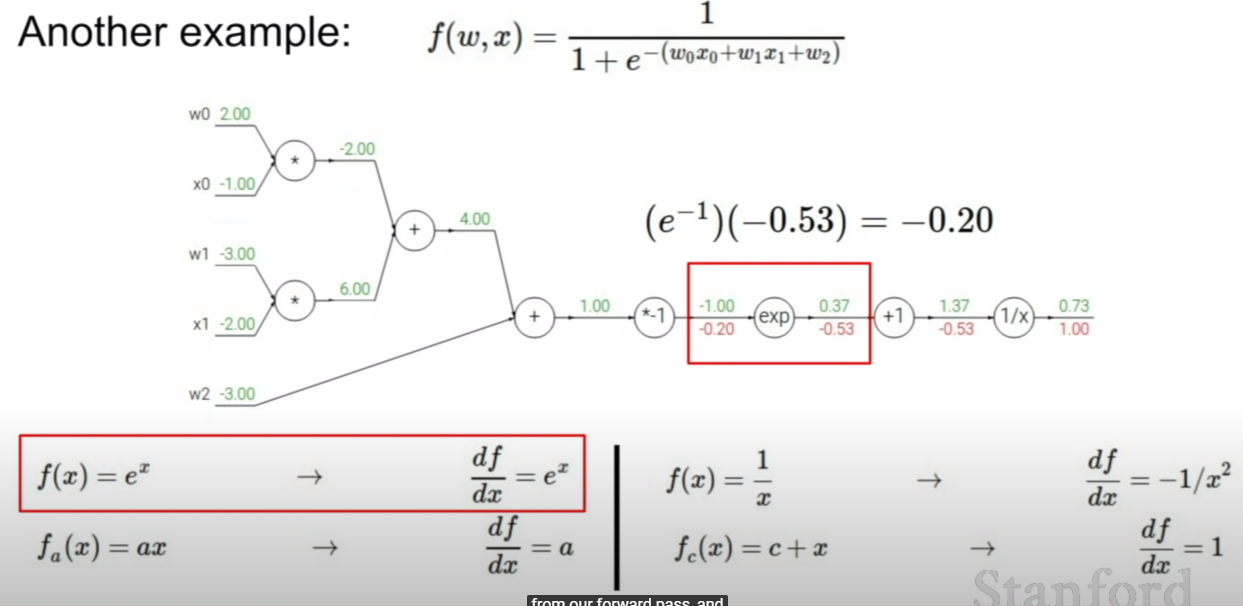


- more complicated example with fomulas of differantial

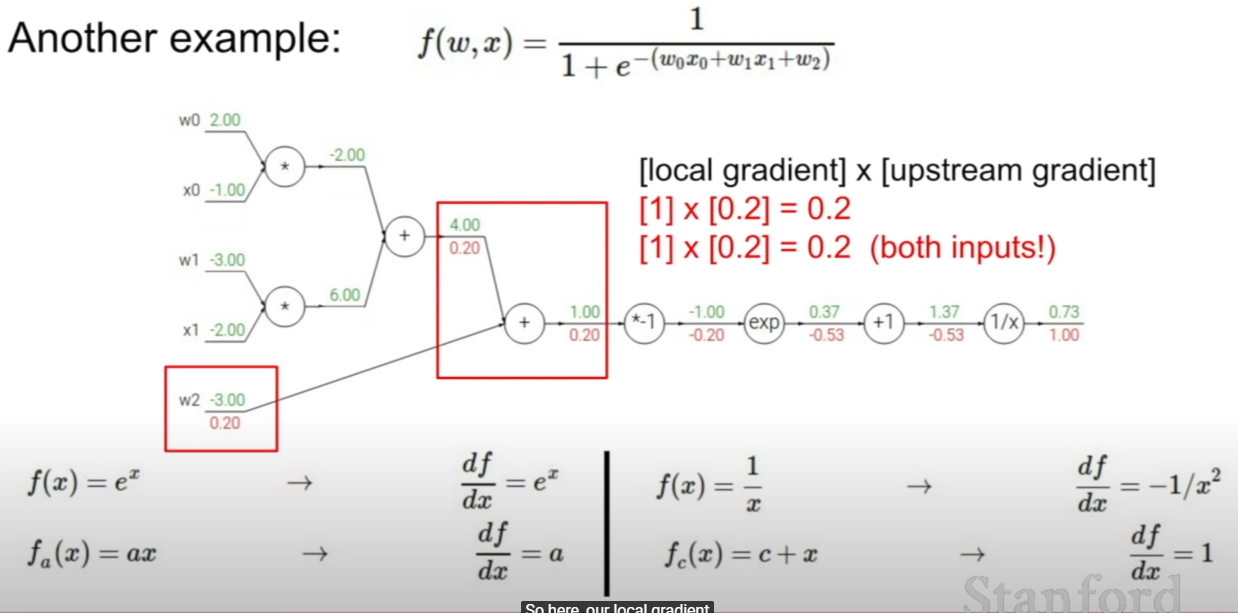
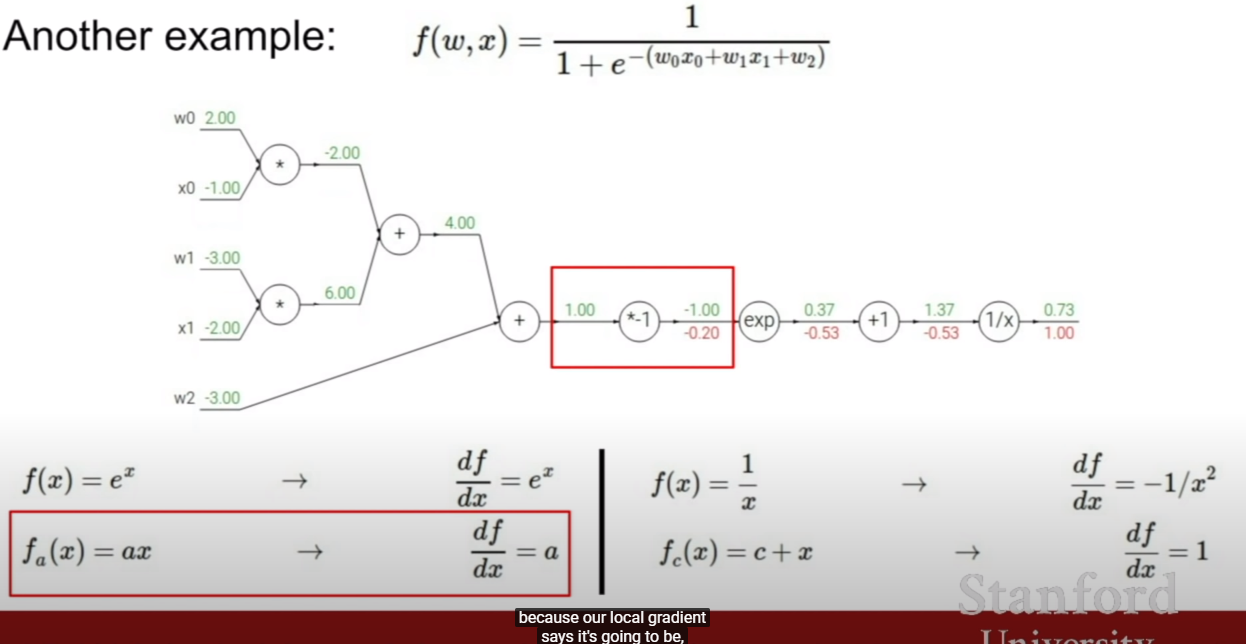
- for each node. we mulitply local gradient with upstream gradient. so next step is multiply upstream gradient -0.53 with local gradient 1 (since df/dx =1 when f(x) = c +x) like below.

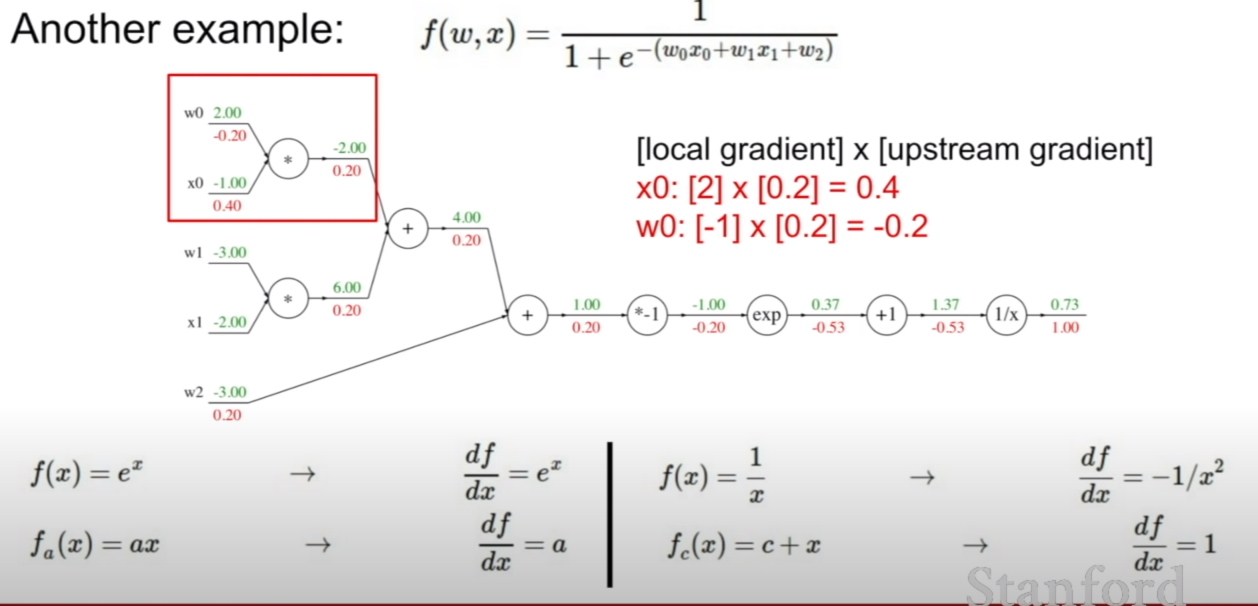


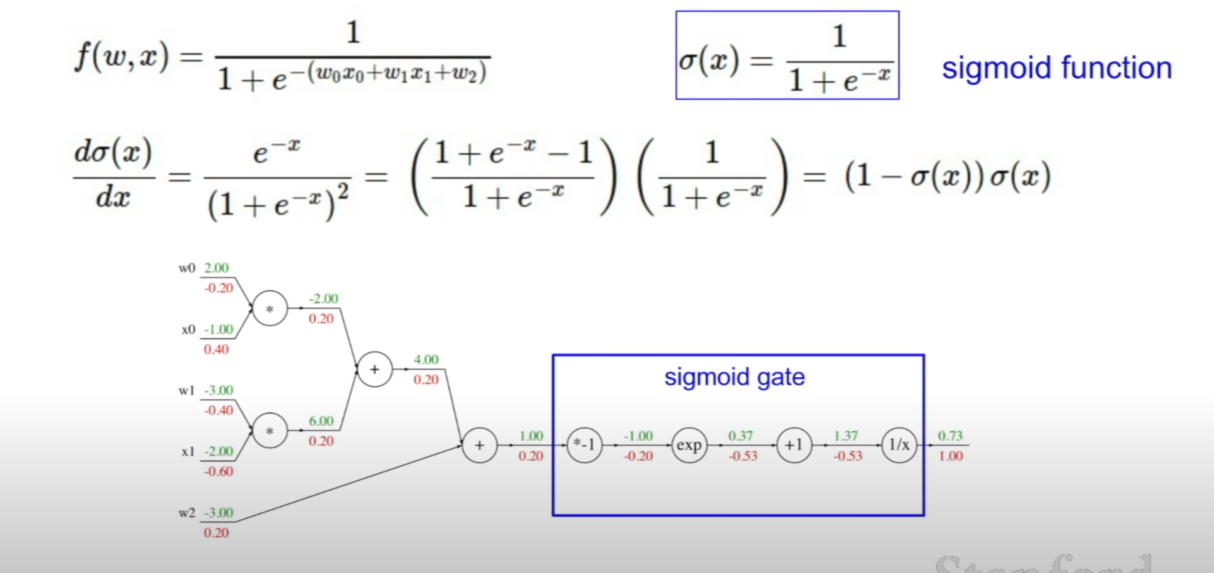




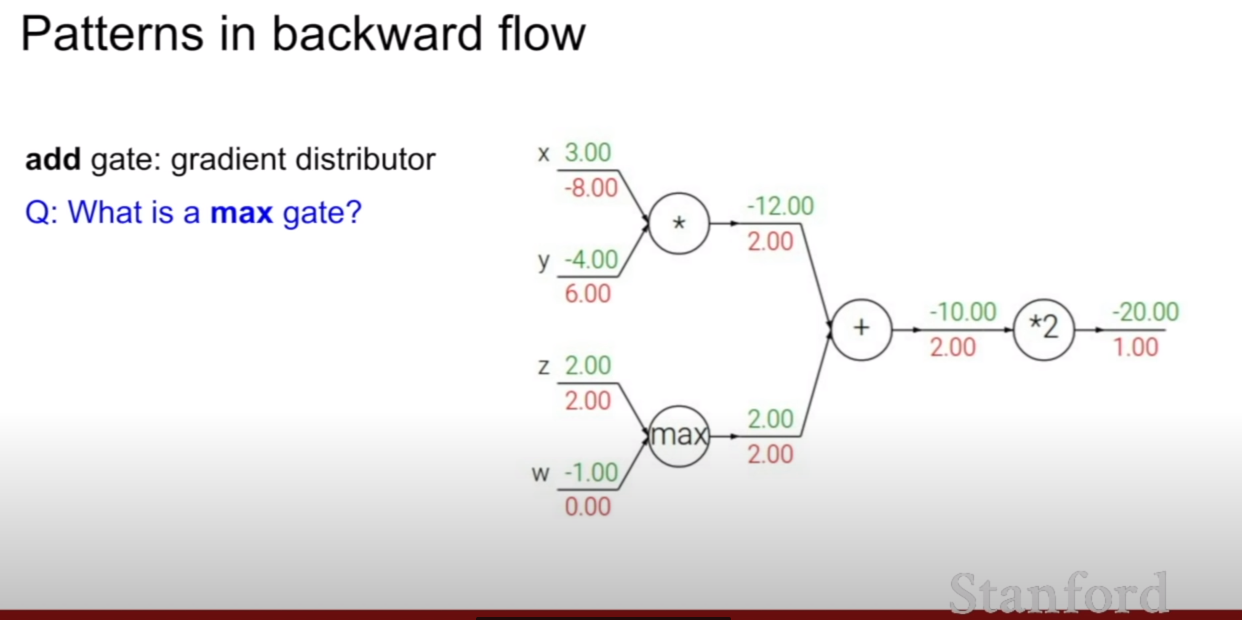
-like above, local gradient’s fomula is e^x, and our forward pass was -1(green value at left of exp node). multiply it with upstream gradient that is -0.53( red value at right of exp node)

- when operand is +, the differential is always 1, so we can calculate like above.



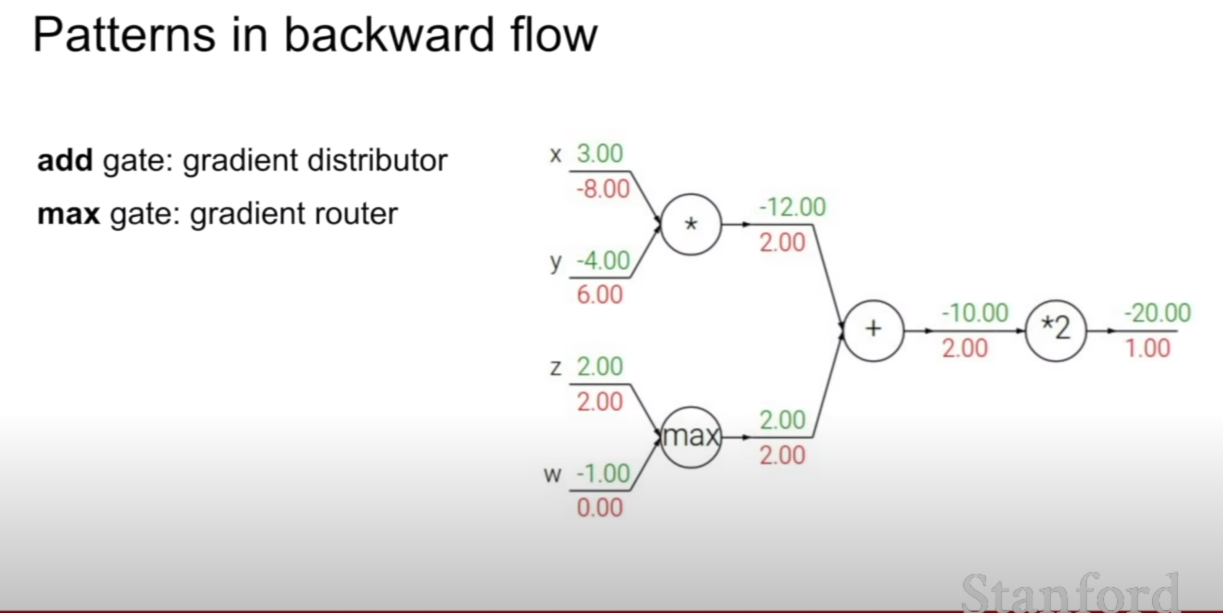


- if we do know local gradient of a specific node like above sigmoid function, we can combine the nodes to a big node.

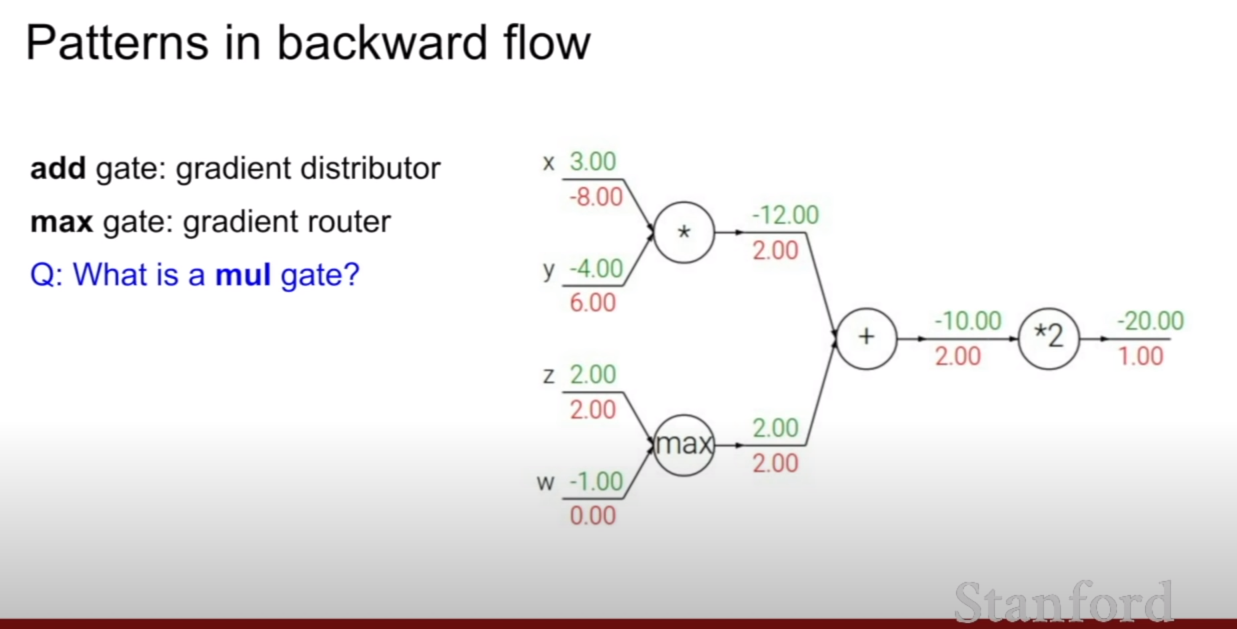


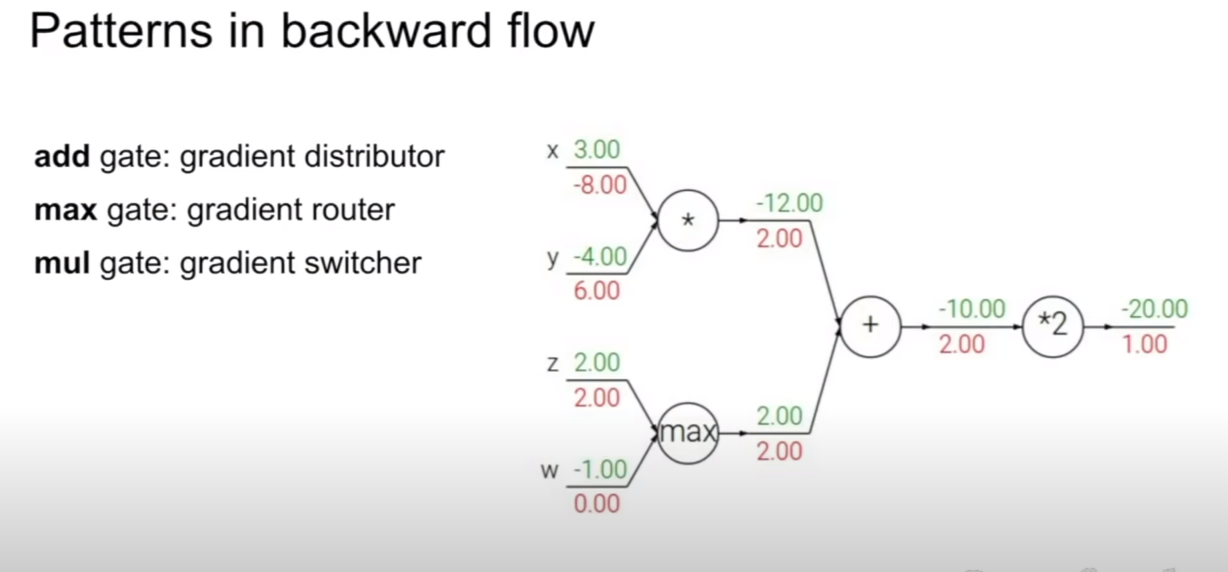
- in this case, one value(is greater than other) has upstream gradient, and other has 0.

- because, at the very and of flow, just z affect to the value of end.

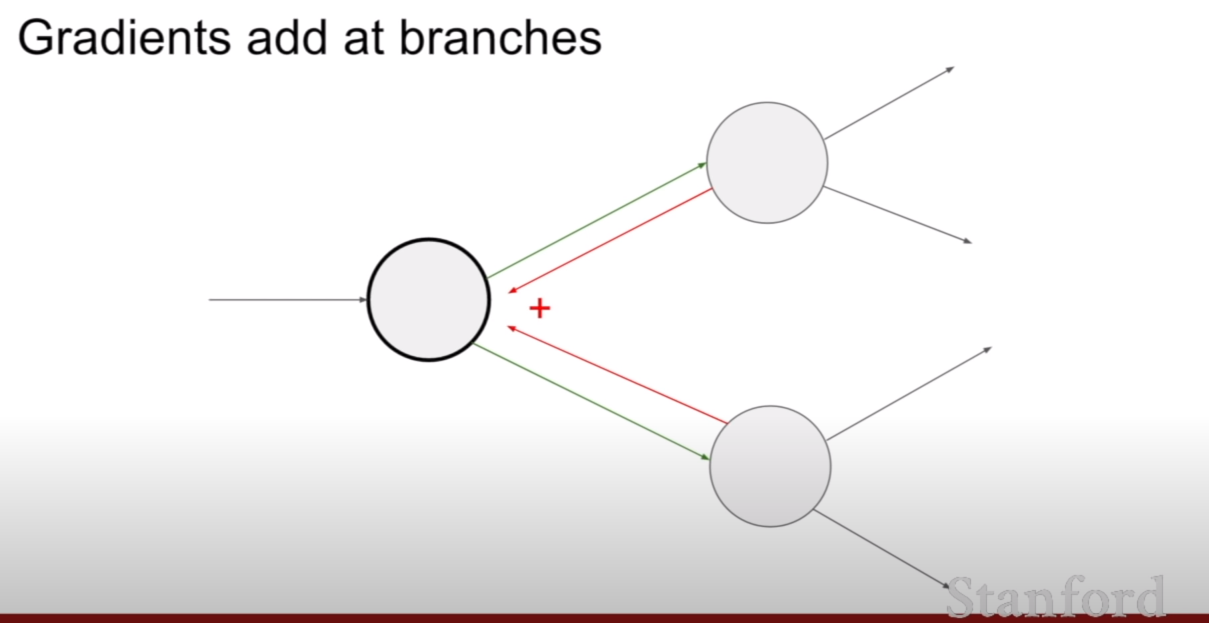


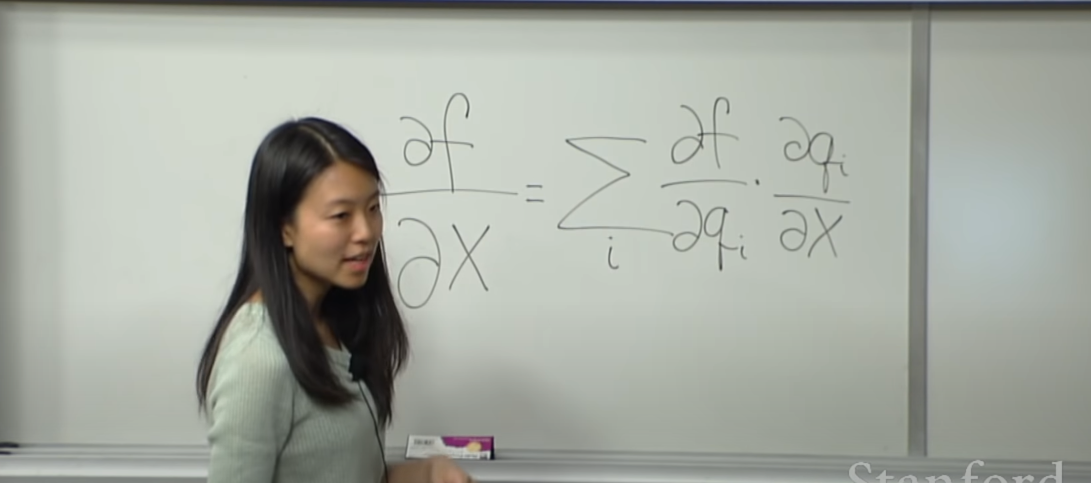
- so we call add gate : gradient distributor, max gate : gardient router.

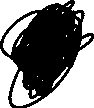




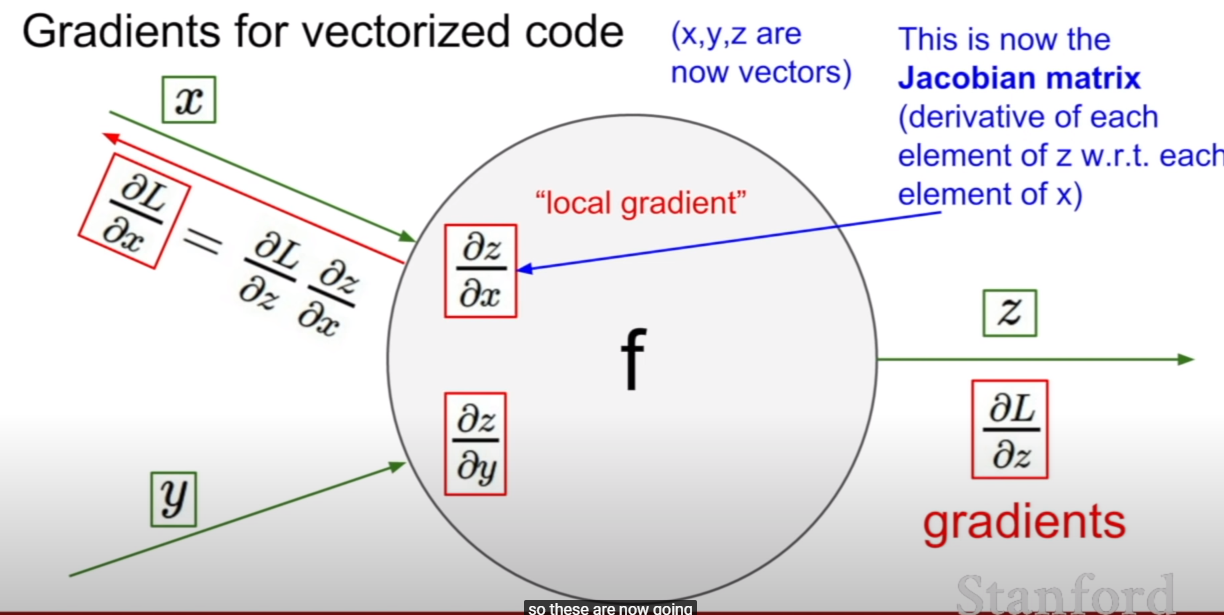
- we call multiply gate : gradient switcher, since a node take the opposite node’s value as a local gradient.



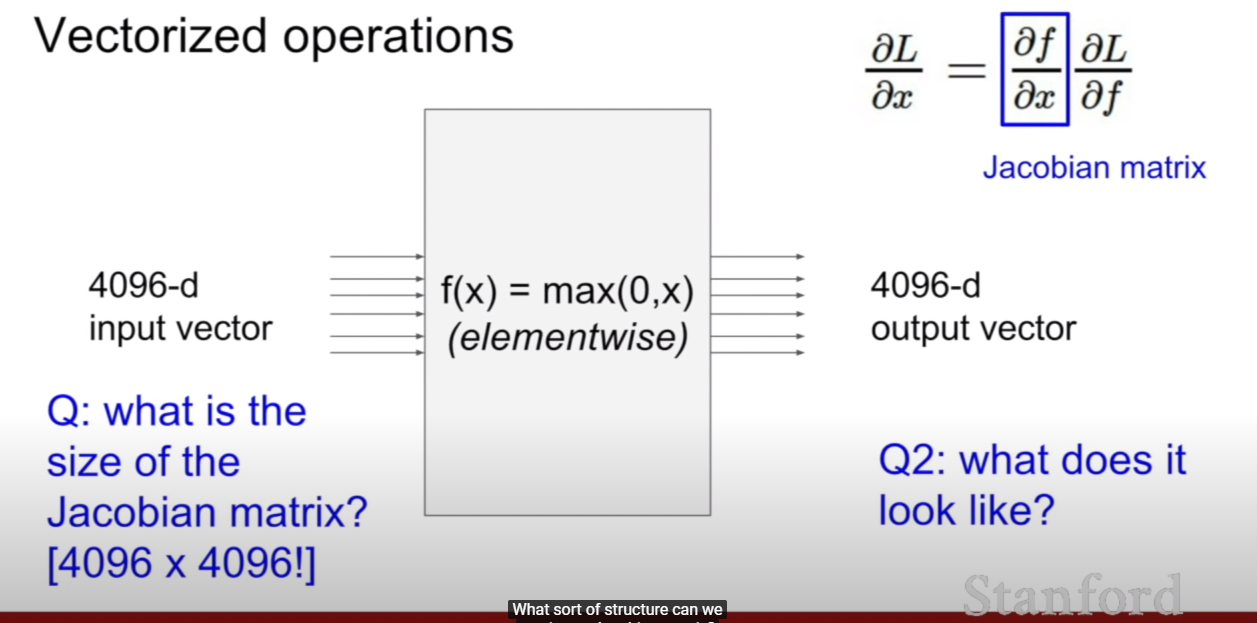




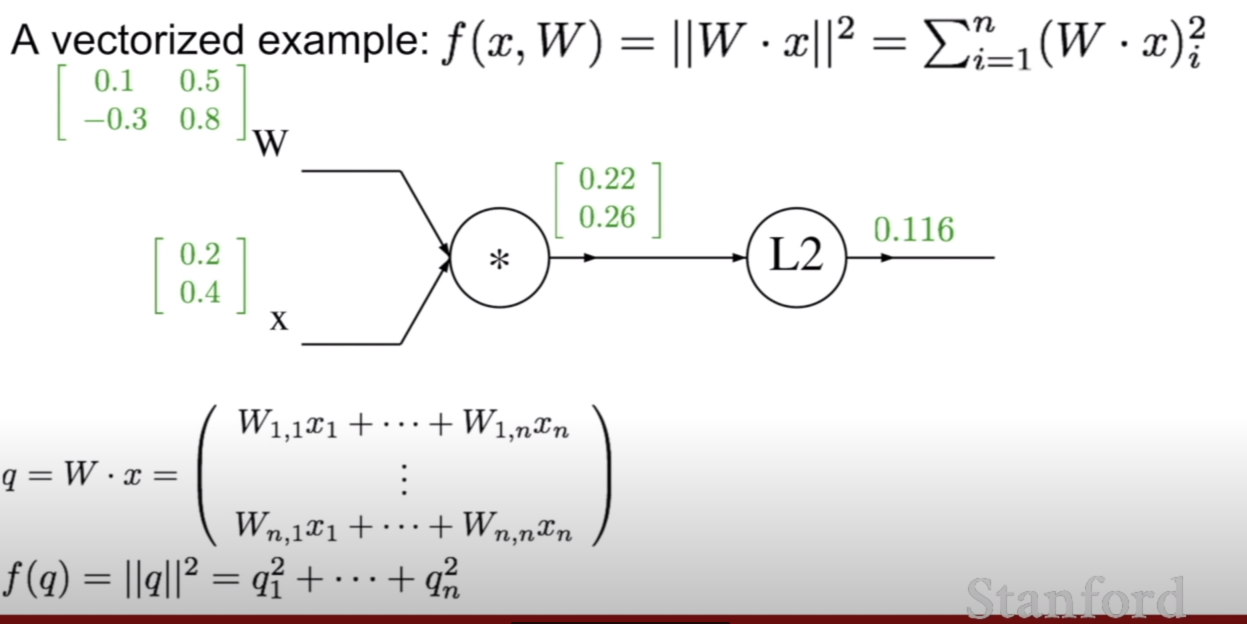
- like above case, both upstream gradient affect to a single node. so we add both backpropagation to make new upstream gradient.

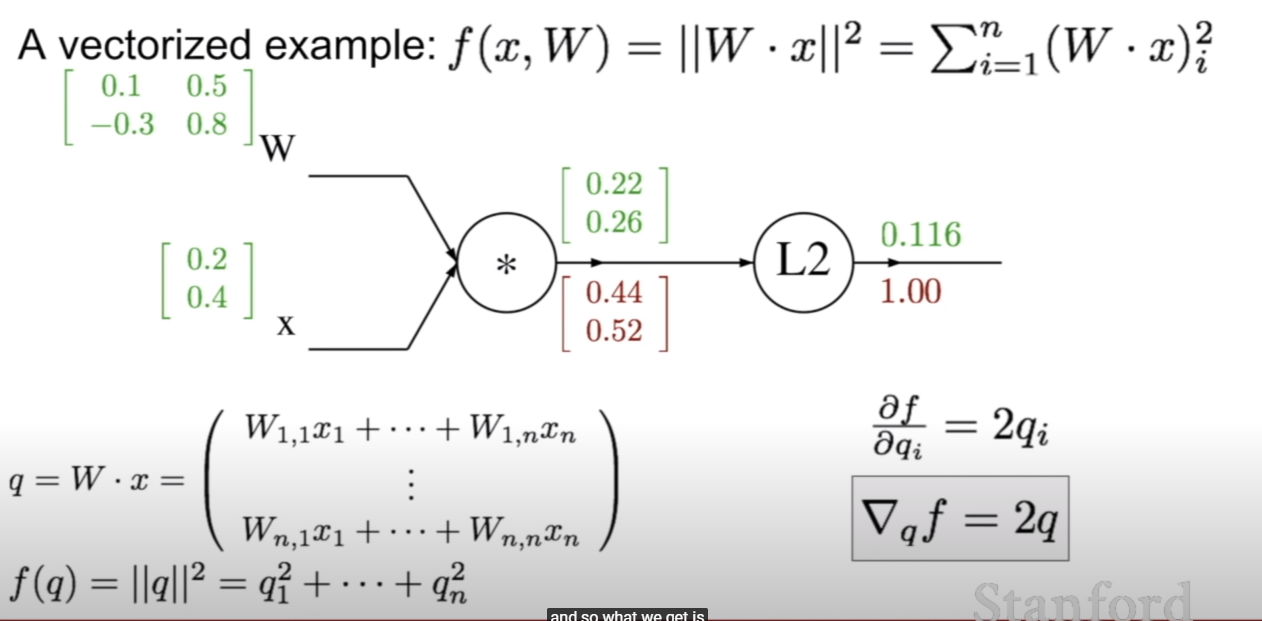






- the answer is diagonal, since x1 value only affect x1 value of result.





- this is the way to backpropagate with vectorized value.

- it means how much a element will affect out final output of the function.

