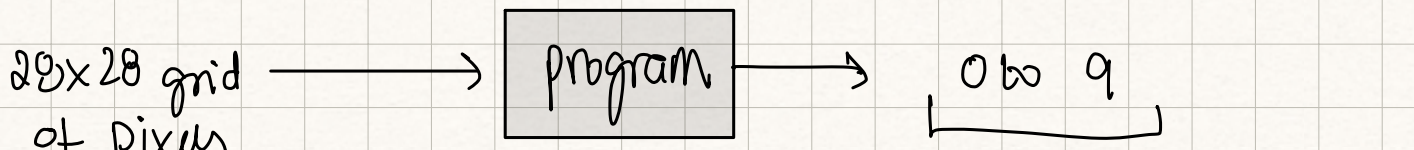
DL 01 Notes: But what is a NN ??

- Brain has no problem recognising 28×28 pixel imgs of 3.
- light sensitive cells that are firing when seeing each img are diff. but visual cortex resolves as it representing the exact same idea!
- Impossible to write a program that



tells what it thinks the digit is

Goal for 1st 2 videos:

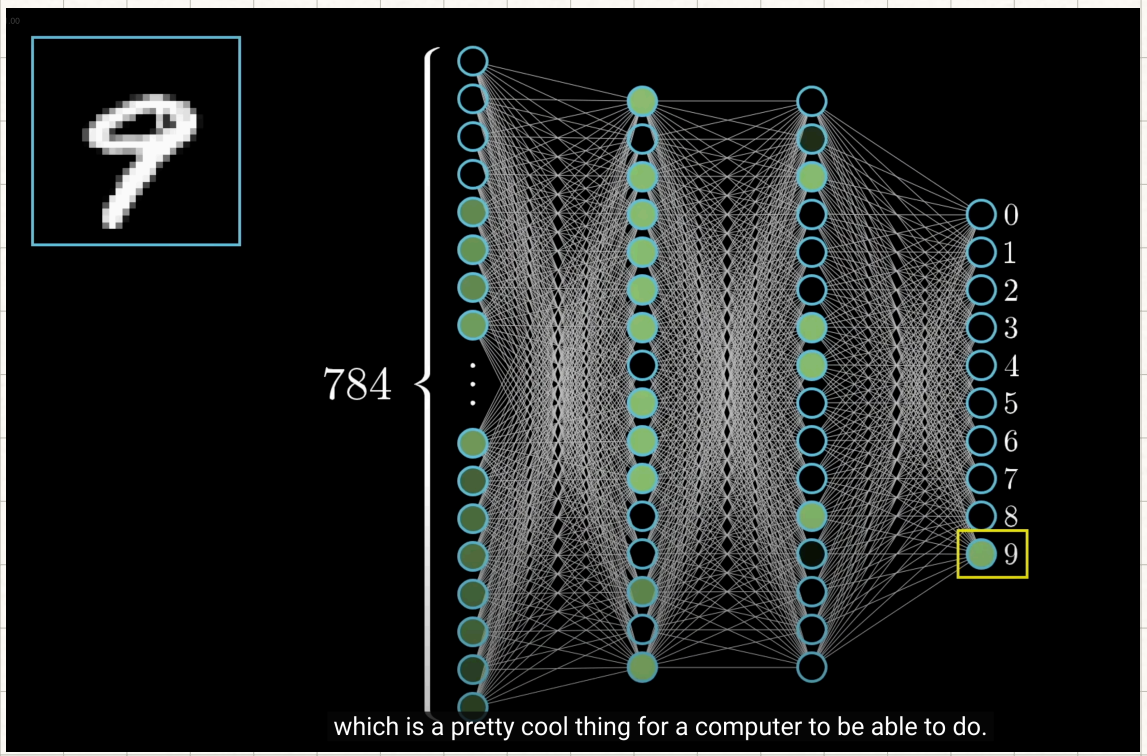
- what a NN actually is
- visualise what it does -
- The structure of a NN is motivated

• intuition for a NN "learning"

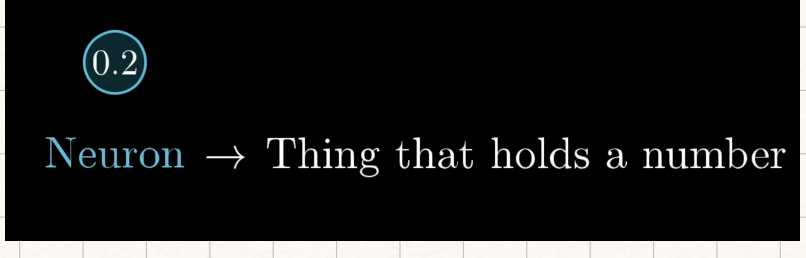
- NN to recognise handwritten digits (MNIST)

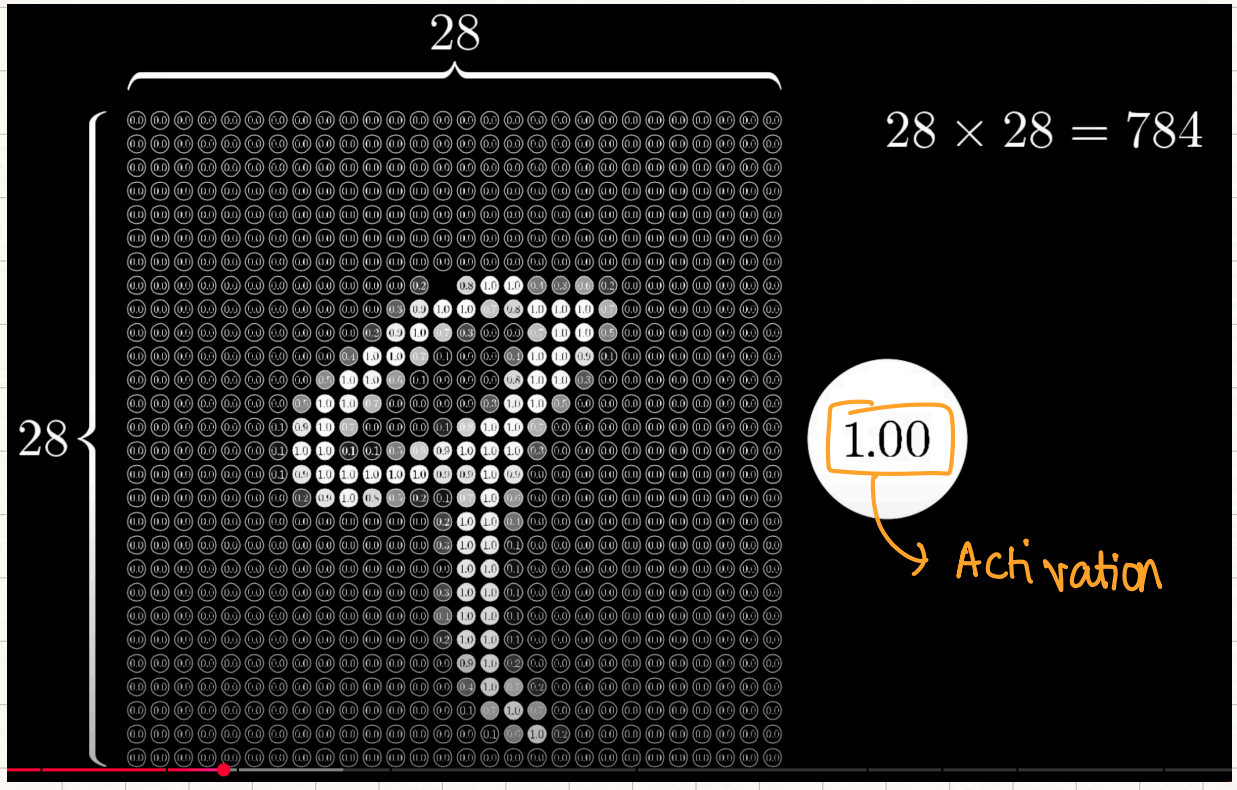
Many variants

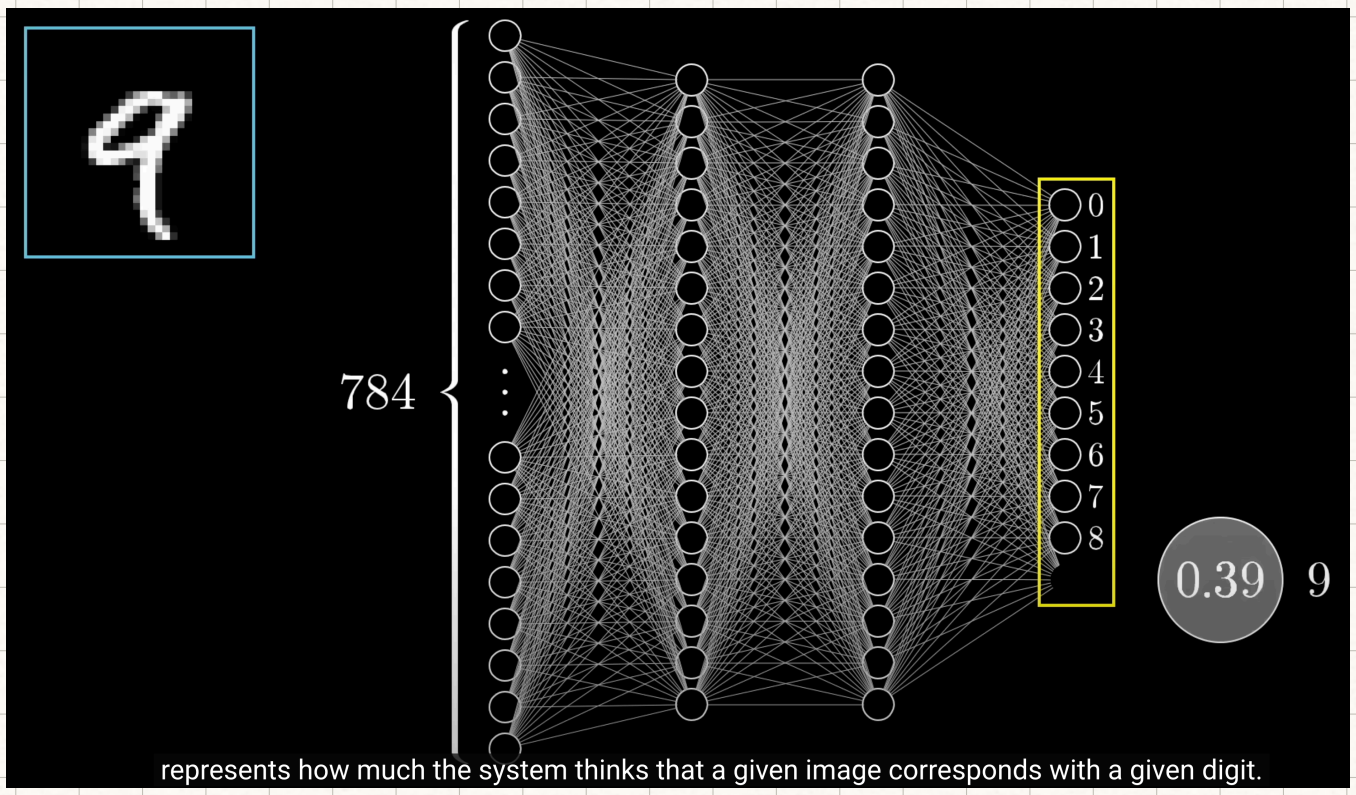
- CNN – img recognition
- LSTM – speech recognition.



Neuron

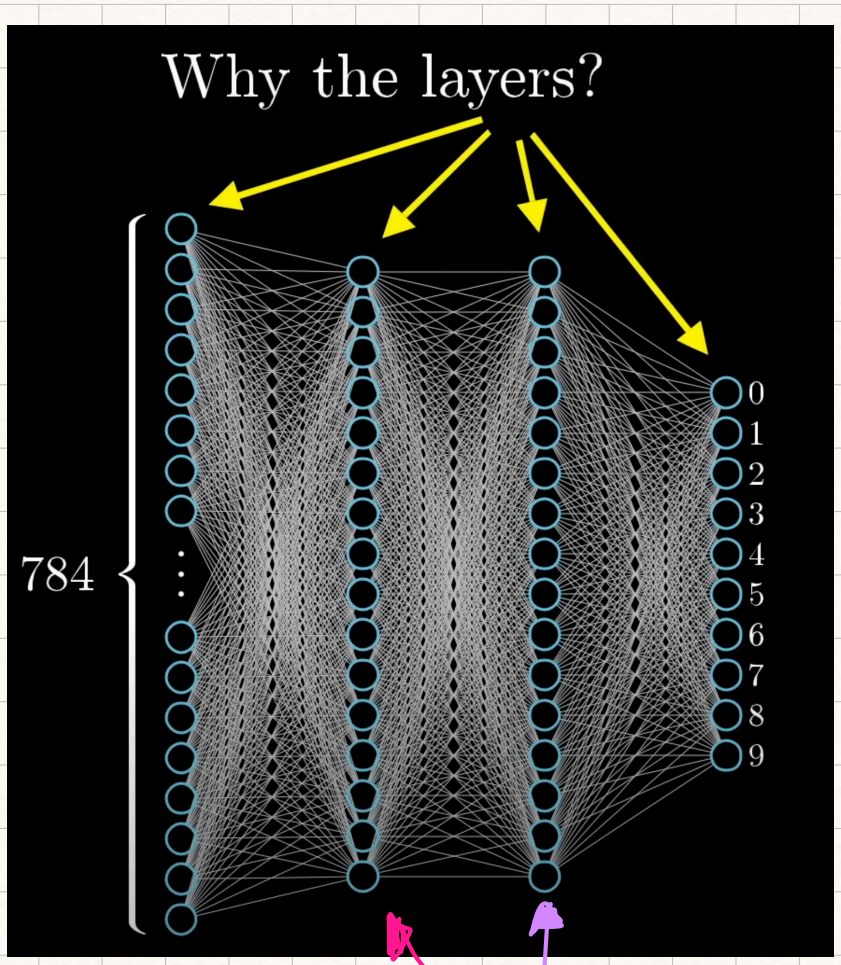


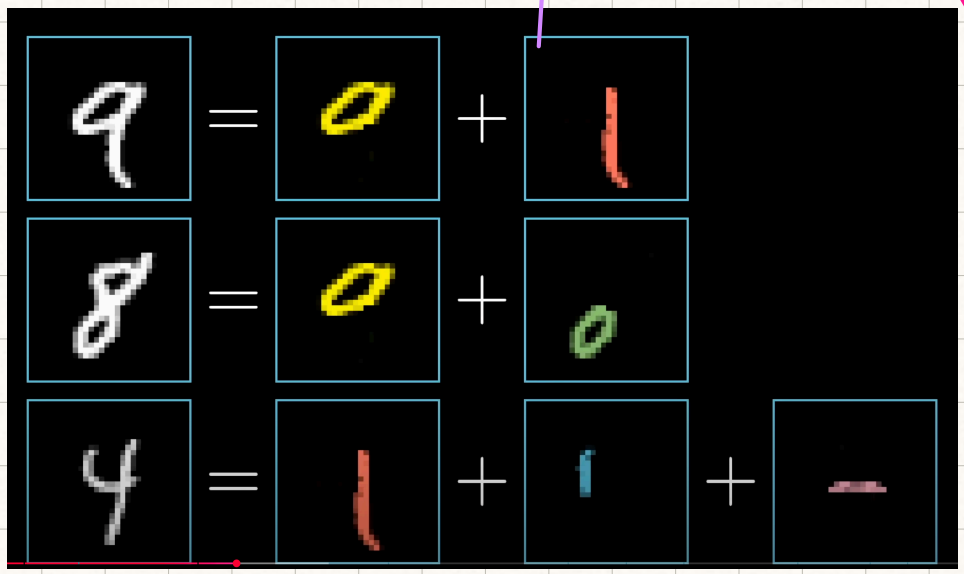


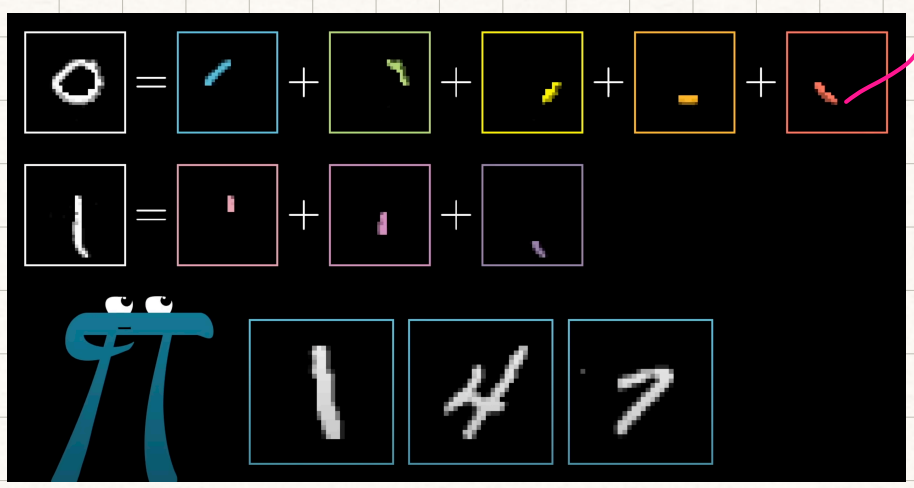


represents how much the system thinks that a given image corresponds with a given digit.

- each input img causes some neurons in the 1st layer to light up, which lights up other neurons & so on...



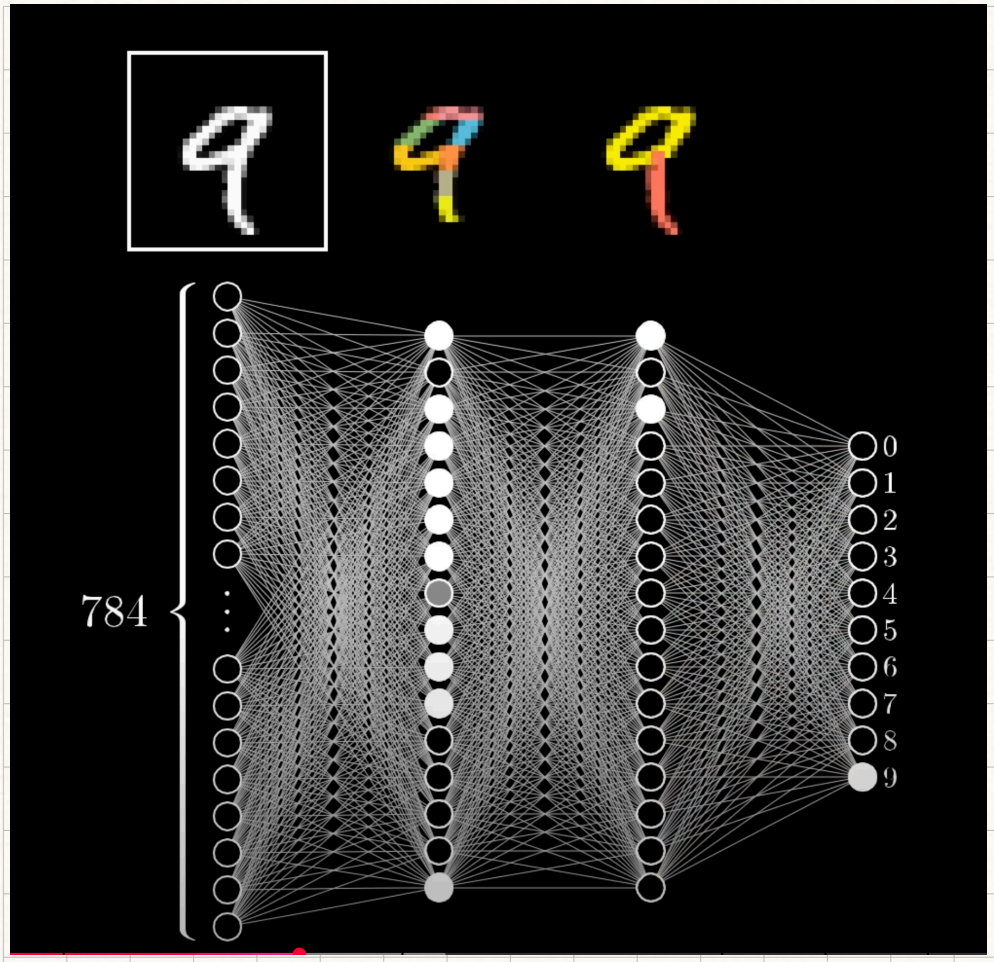


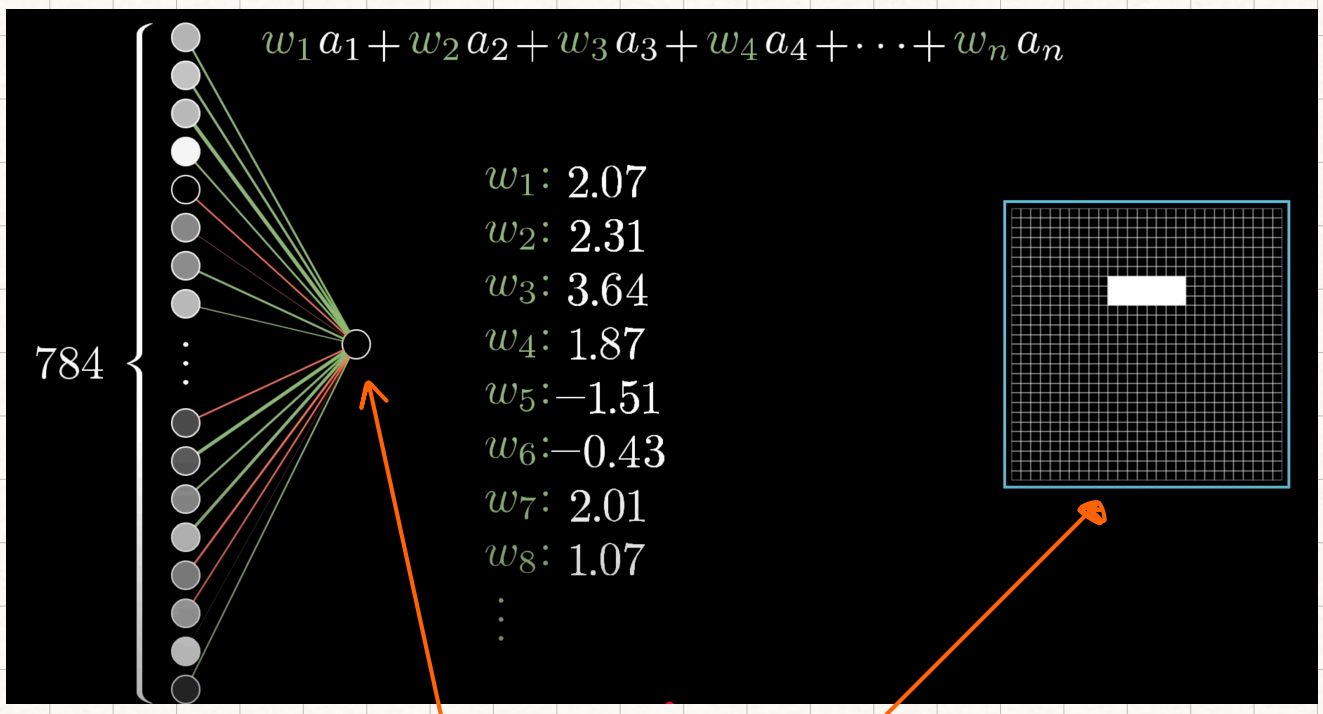


how do you recog.

a loop?

A line?

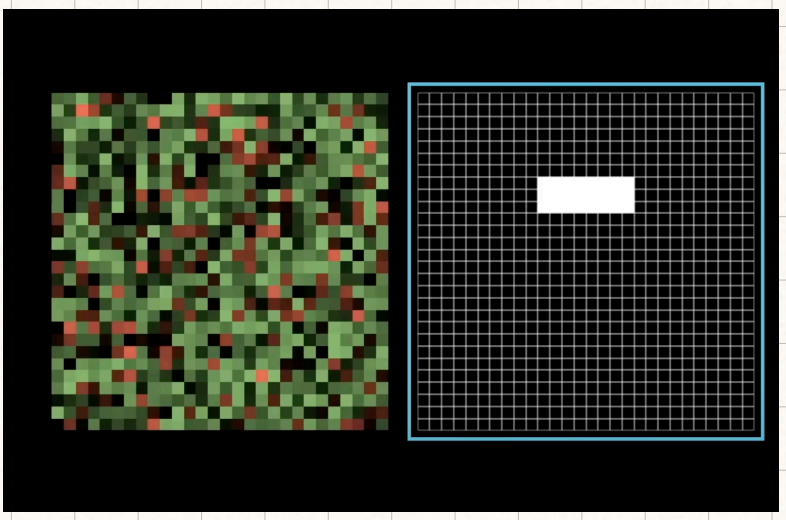




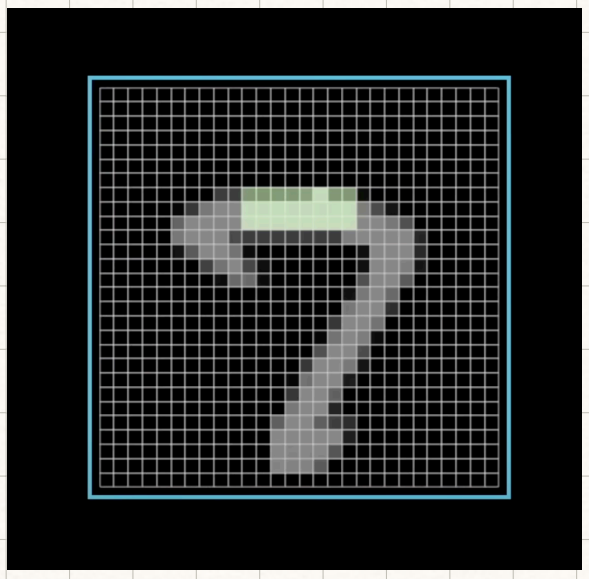
we want this neuron to light up when there's an edge here

So, we think of the params as "weights"

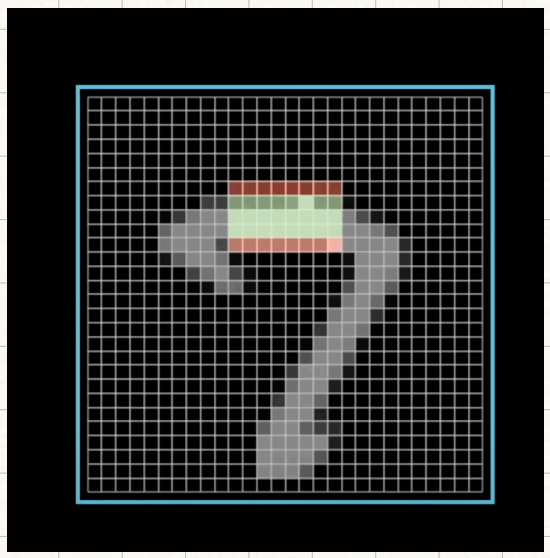
Useful to think of weights as a grid of their own:



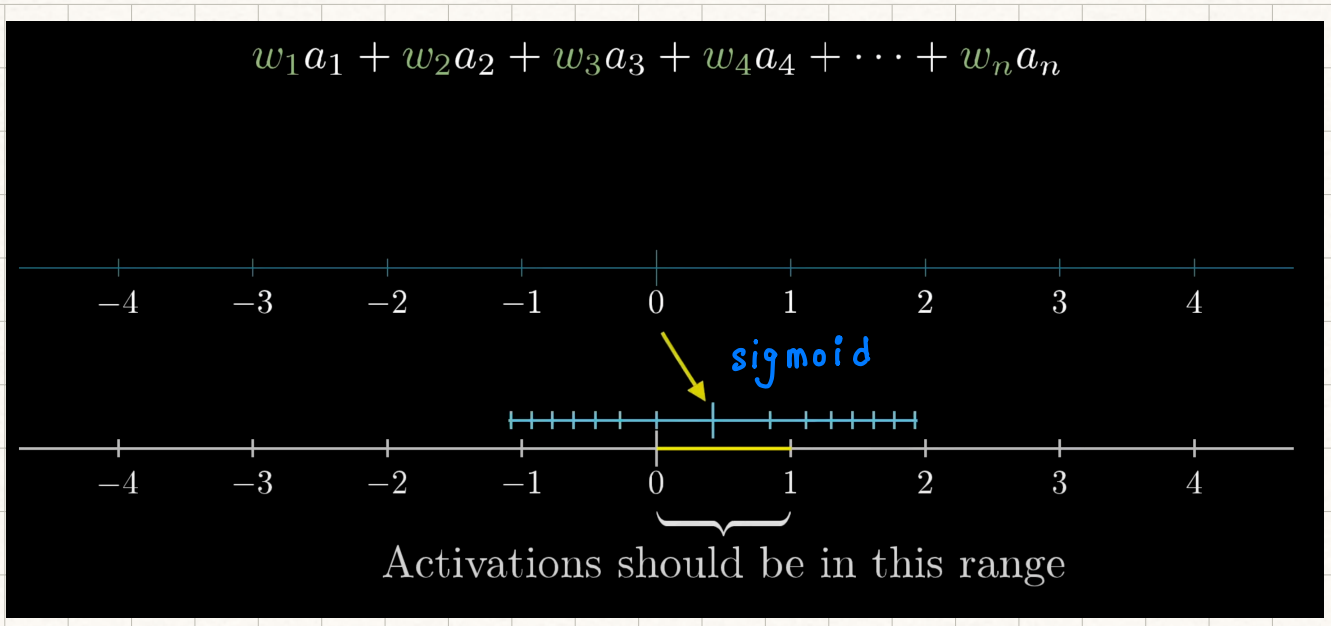
if

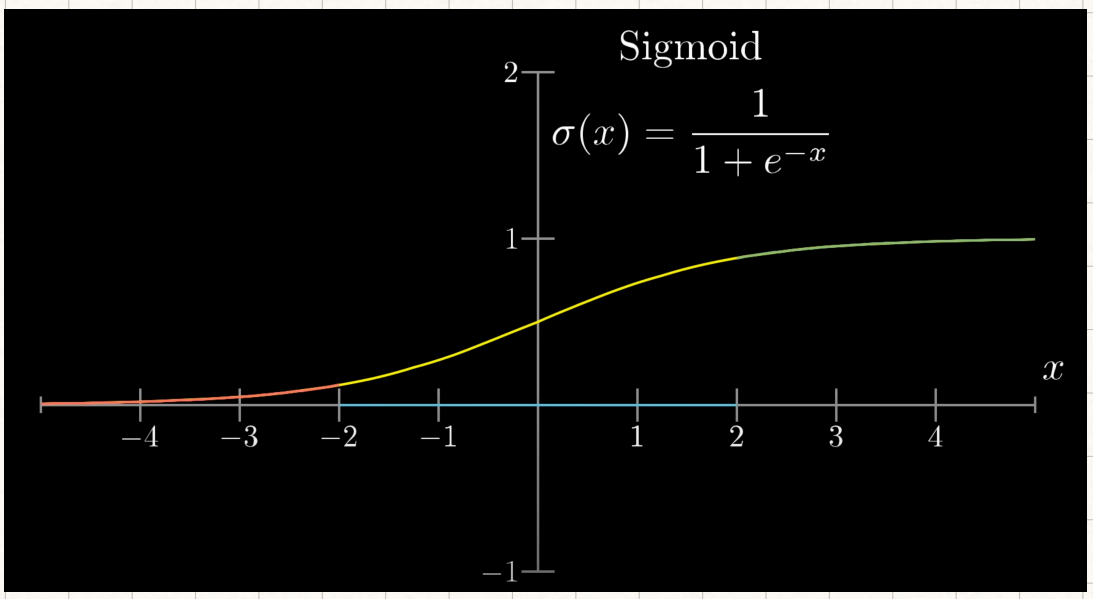


only light up when that patch has white



sum is ↑↑ when middle pixel - white & surrounding → dark.





$\sigma(w_1a_1 + w_2a_2 + \cdots + w_na_n)$

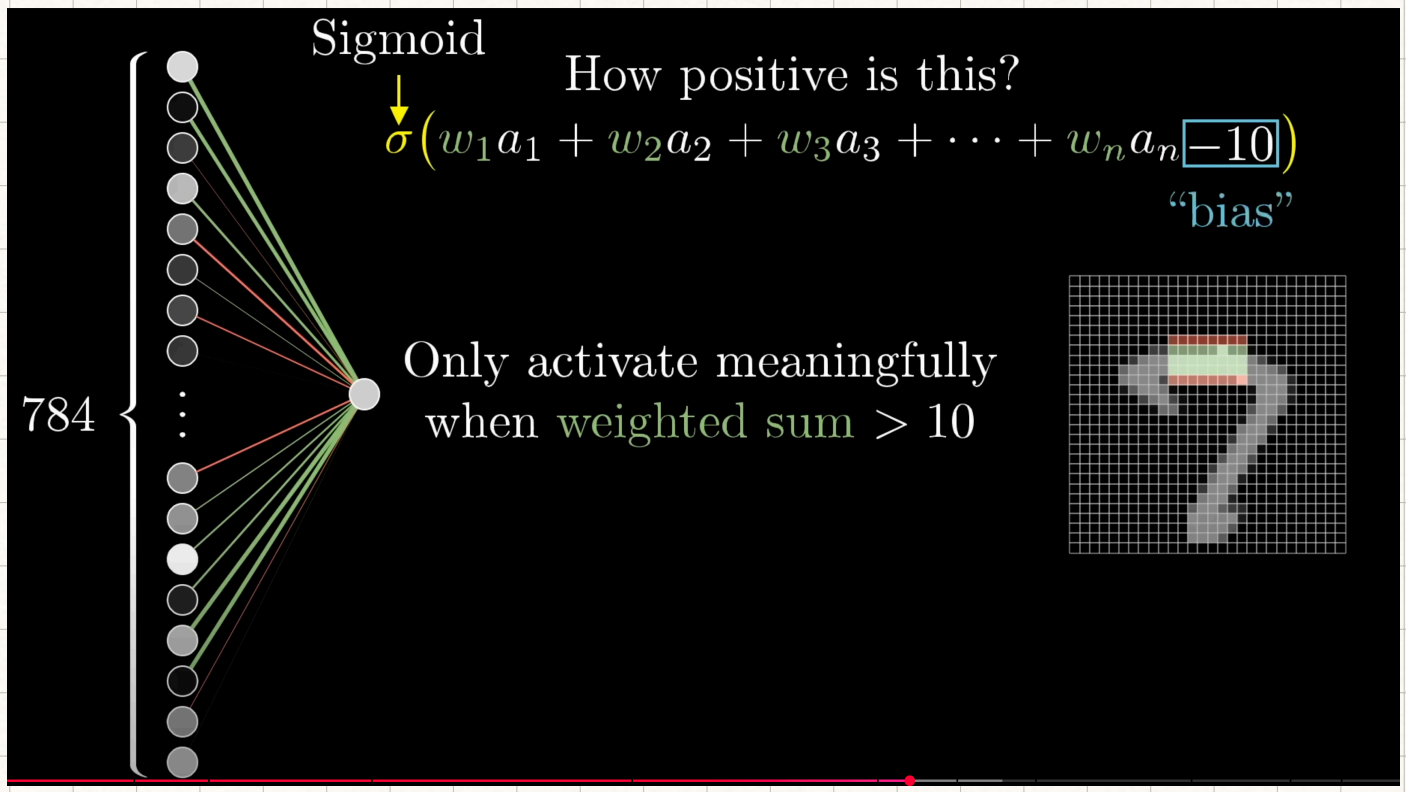
Say you only want neuron to be active when the weighted sum is more than 10

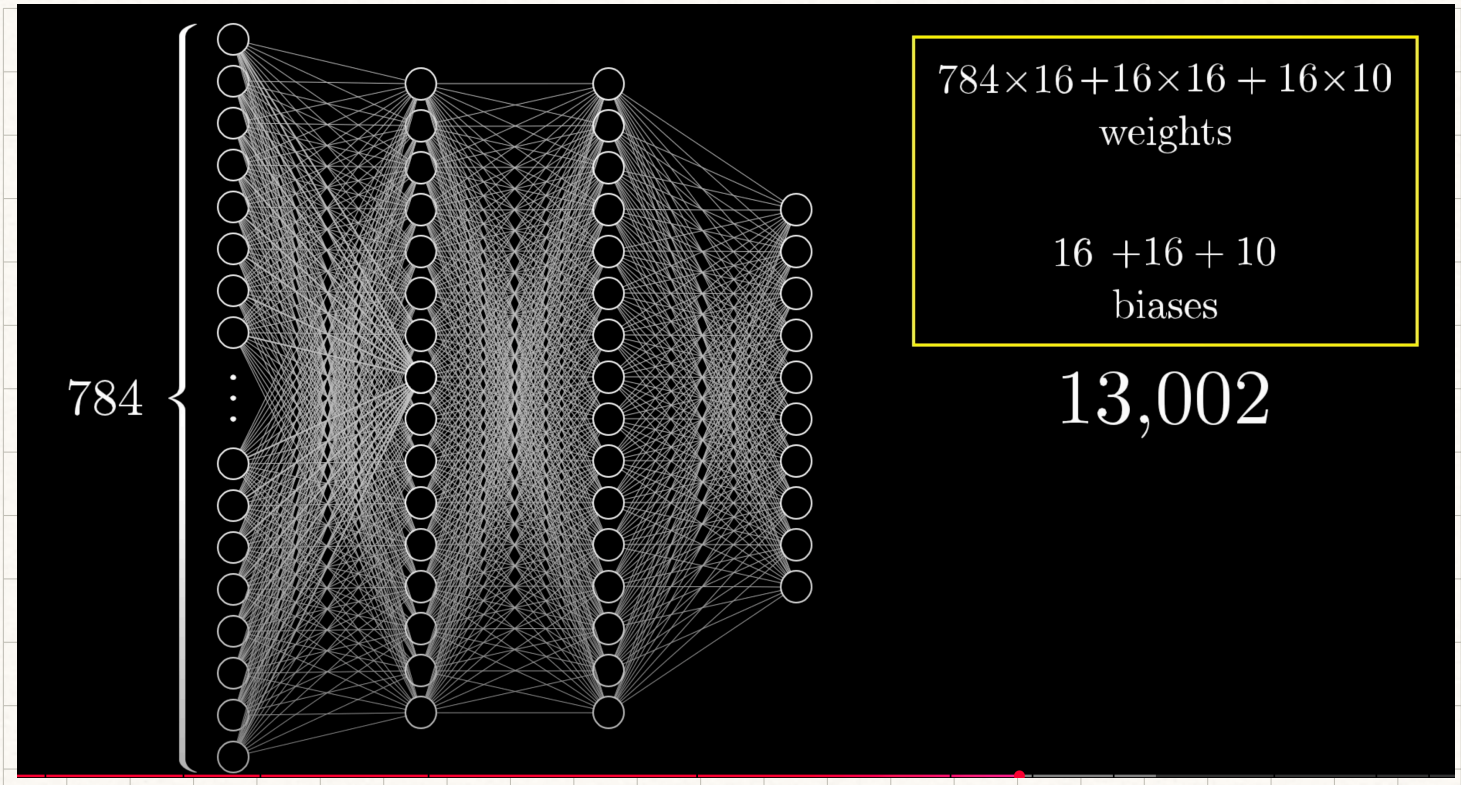
⇒ bias for inactive.

So, $\sigma(w_1a_1 + w_2a_2 + \cdots + w_na_n - 10)$

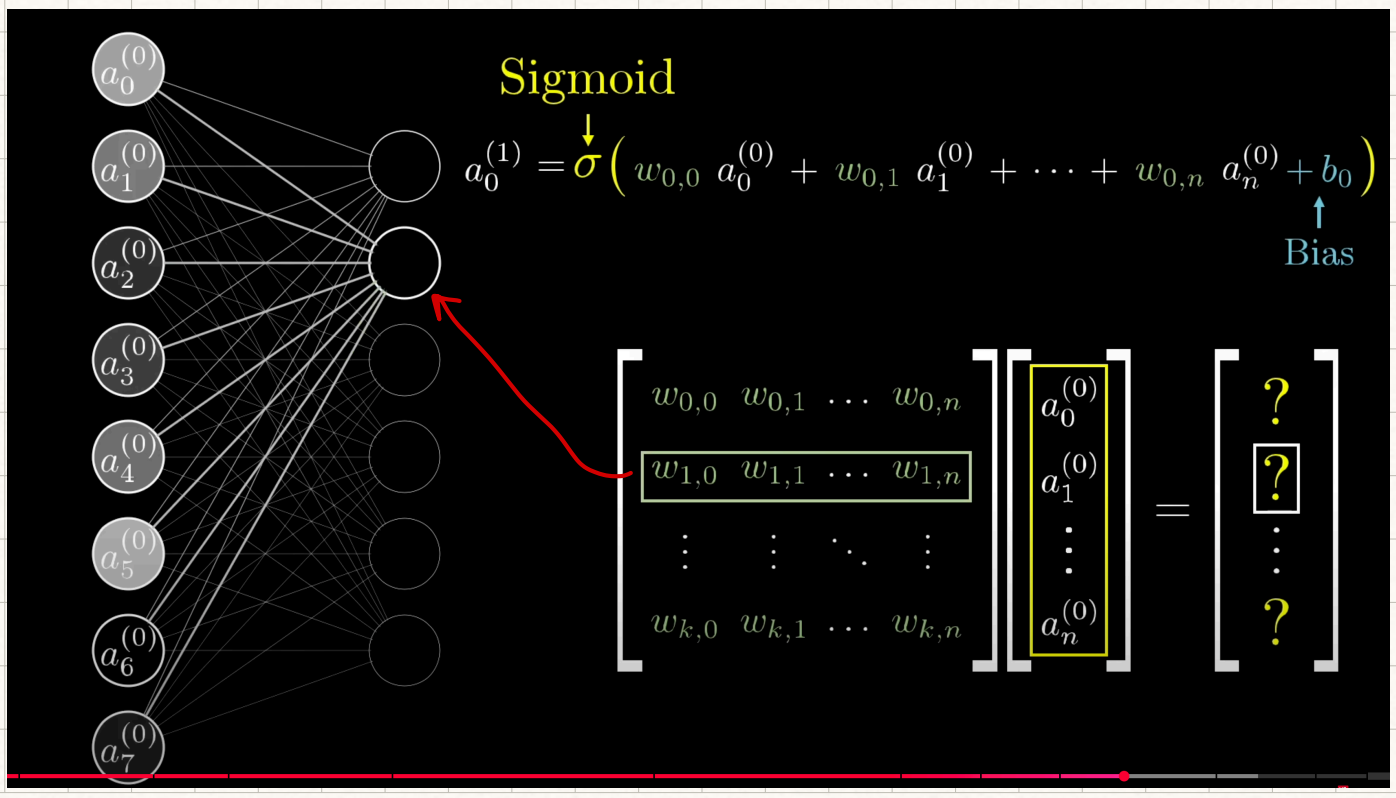
weights → what pixel pattern the neuron is picking up on

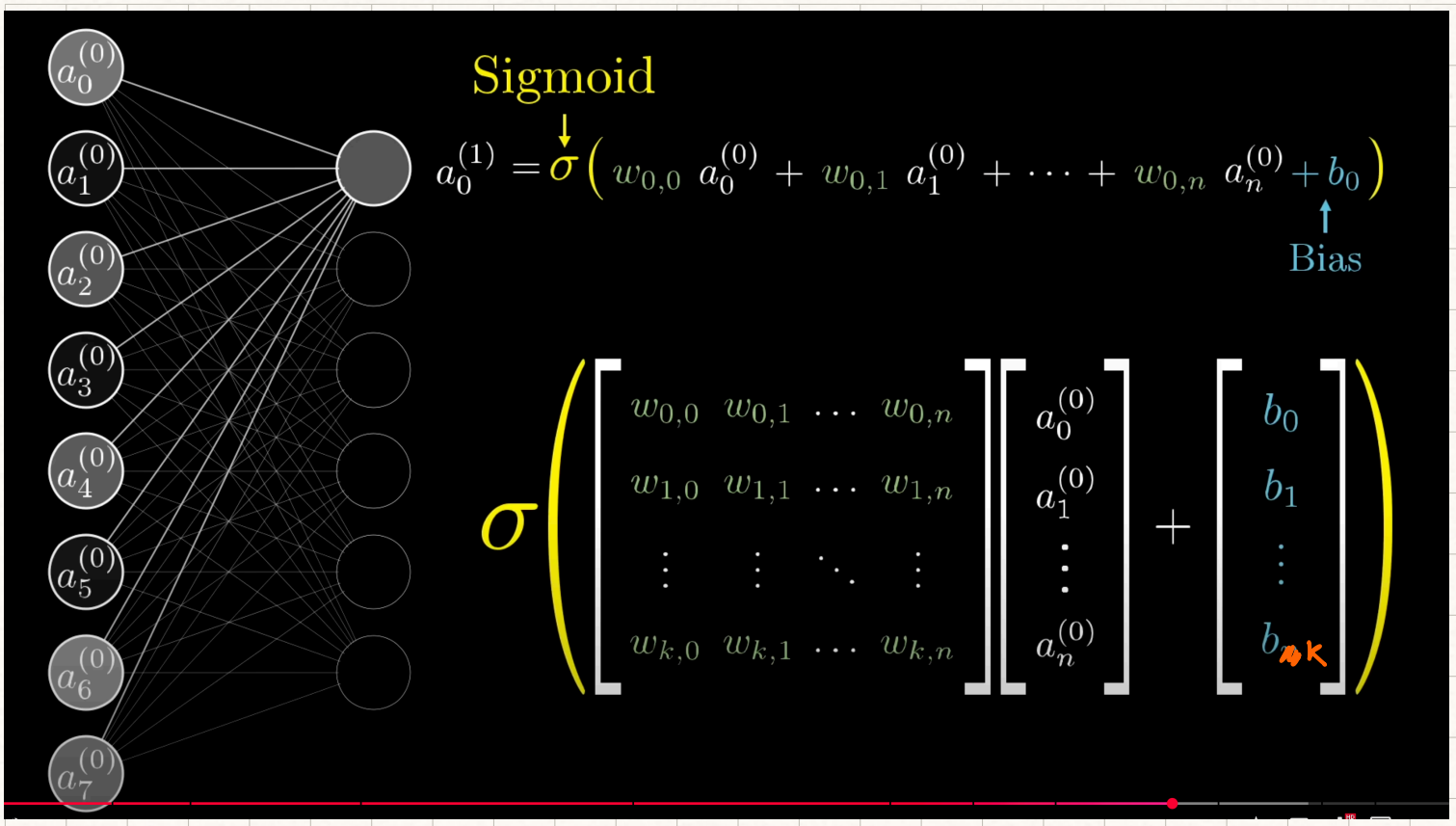
bias → how high weighted sum needs to be b4 the neuron starts to get meaningfully active

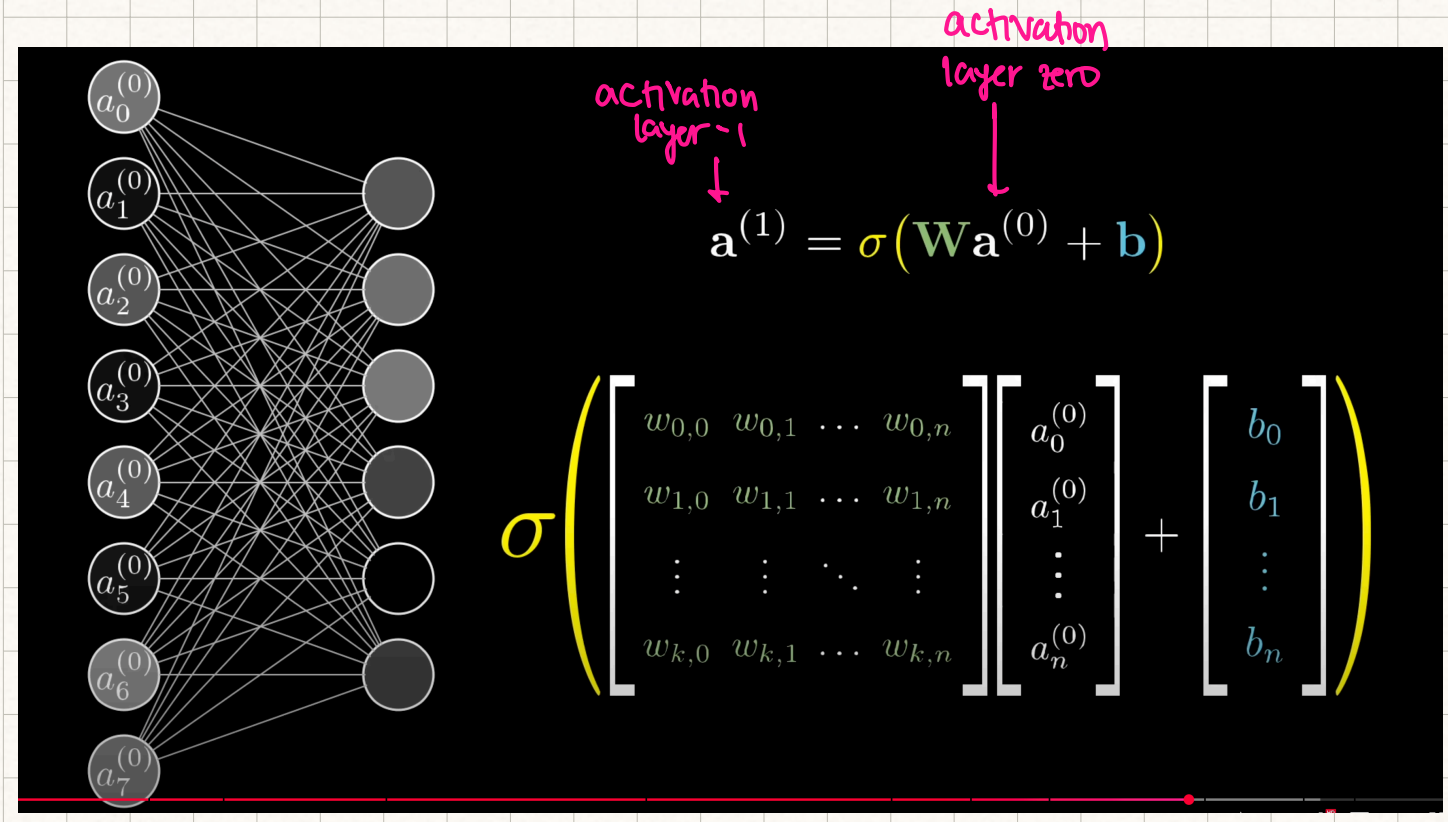




"training" → finding valid setting for these numbers so that it can solve the problem @ hand.

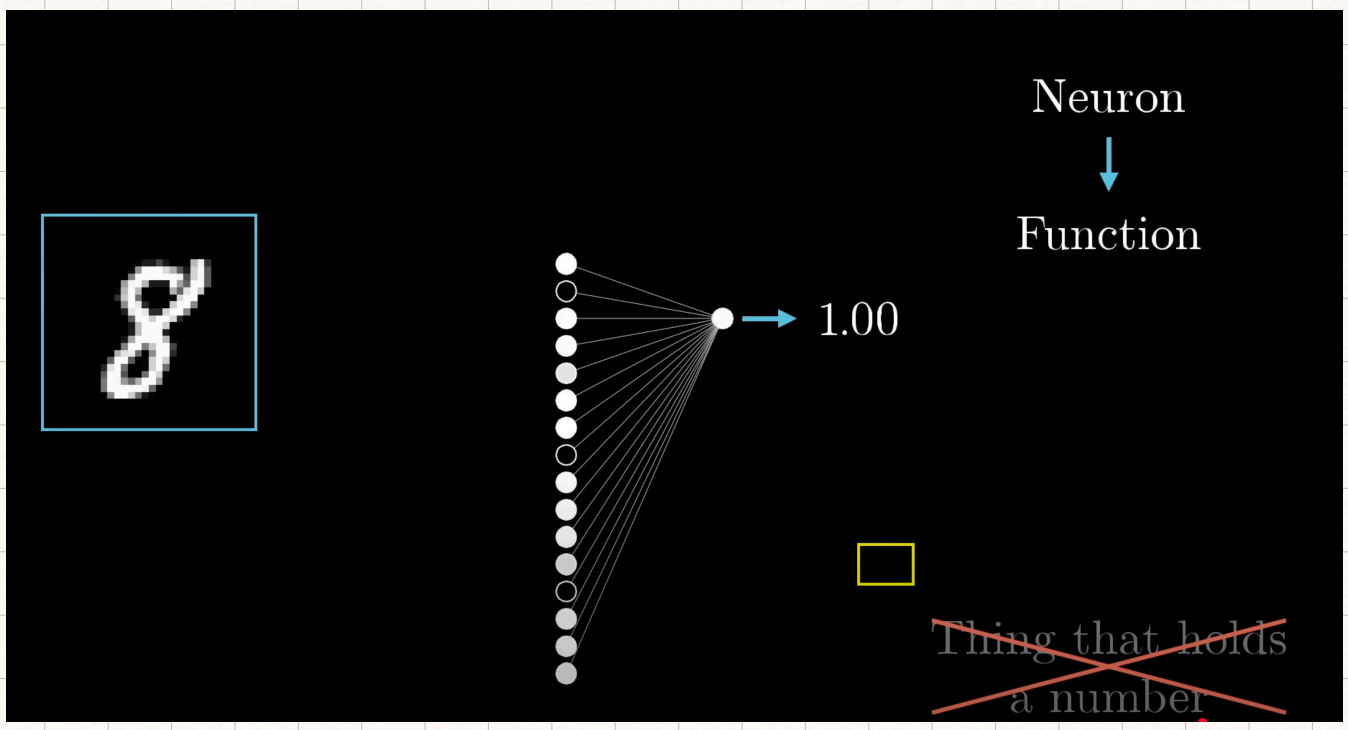



Sigmoid
$a_0^{(1)} = \sigma\left(w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} + \cdots + w_{0,n}\, a_n^{(0)} + b_0 \right)$
Bias
$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix} \right)$


activation layer - 1
activation layer zero
$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$
$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right)$

$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$$

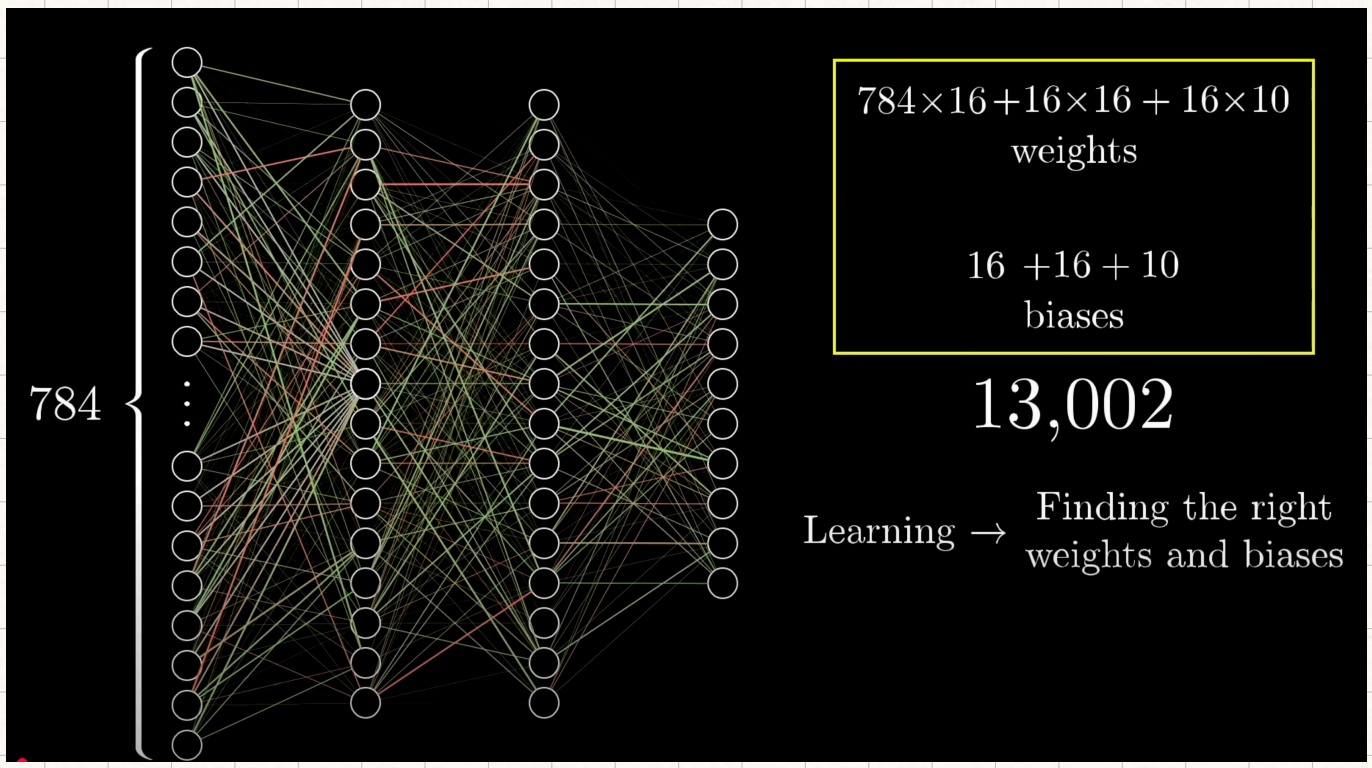
```python
class Network(object):
    def __init__(self, *args, **kwargs):
        #...yada yada, initialize weights and biases...

    def feedforward(self, a):
        """Return the output of the network for an input vector a"""
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a) + b)
        return a
```



ReLU ✓ Sigmoid ✓

2nd Video || DL 2 || Gradient Descent.

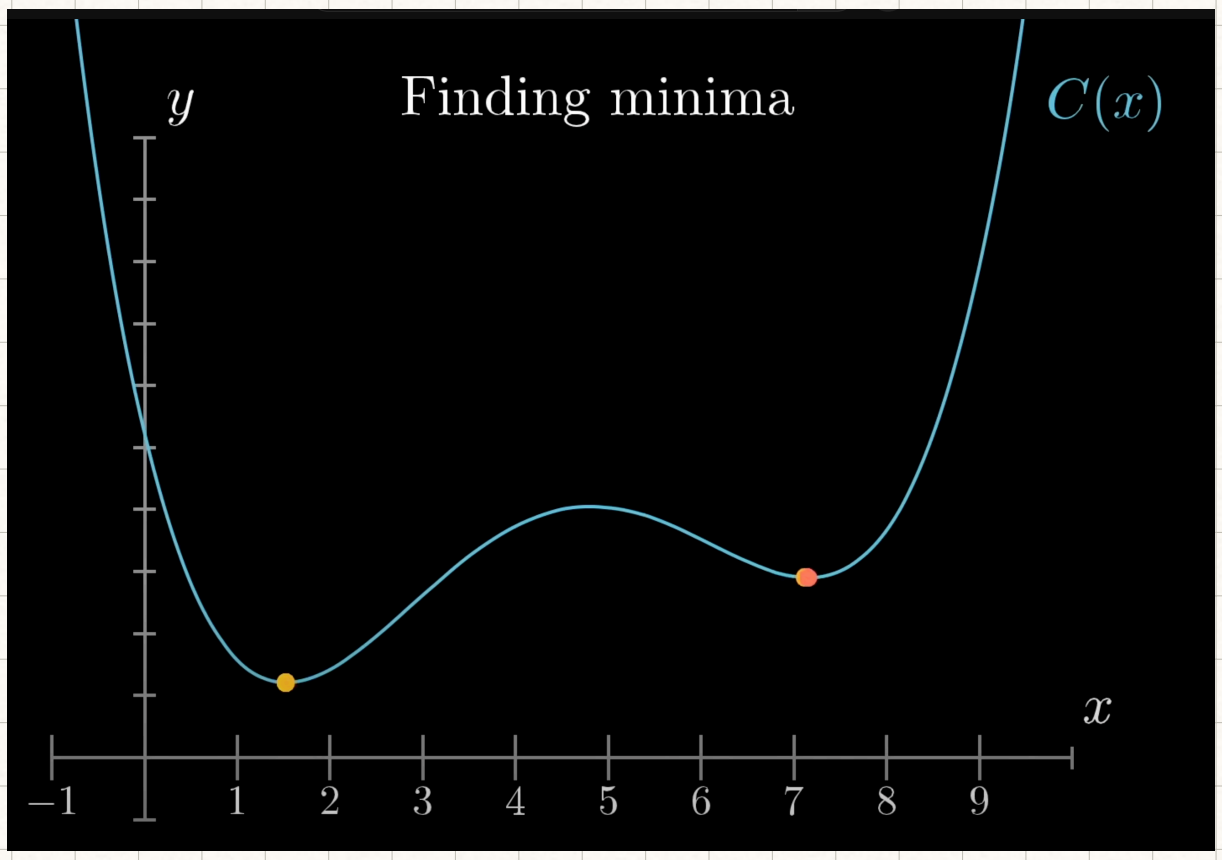
- <Recap>



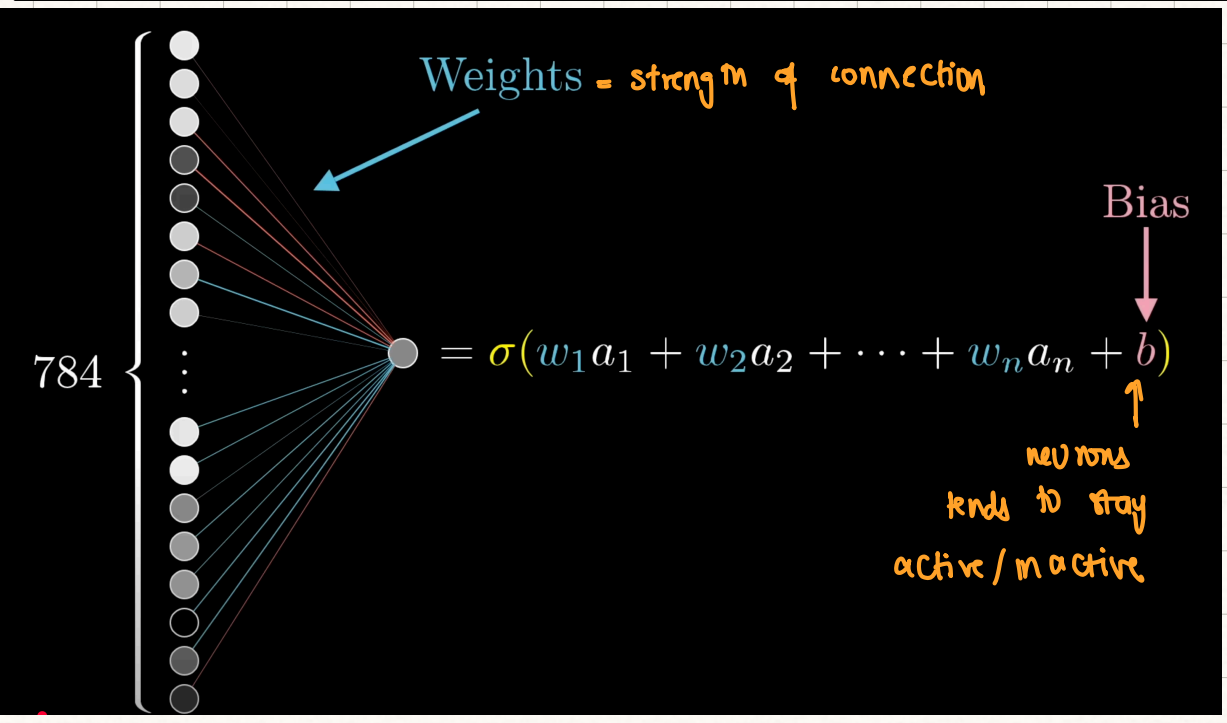
</Recap>

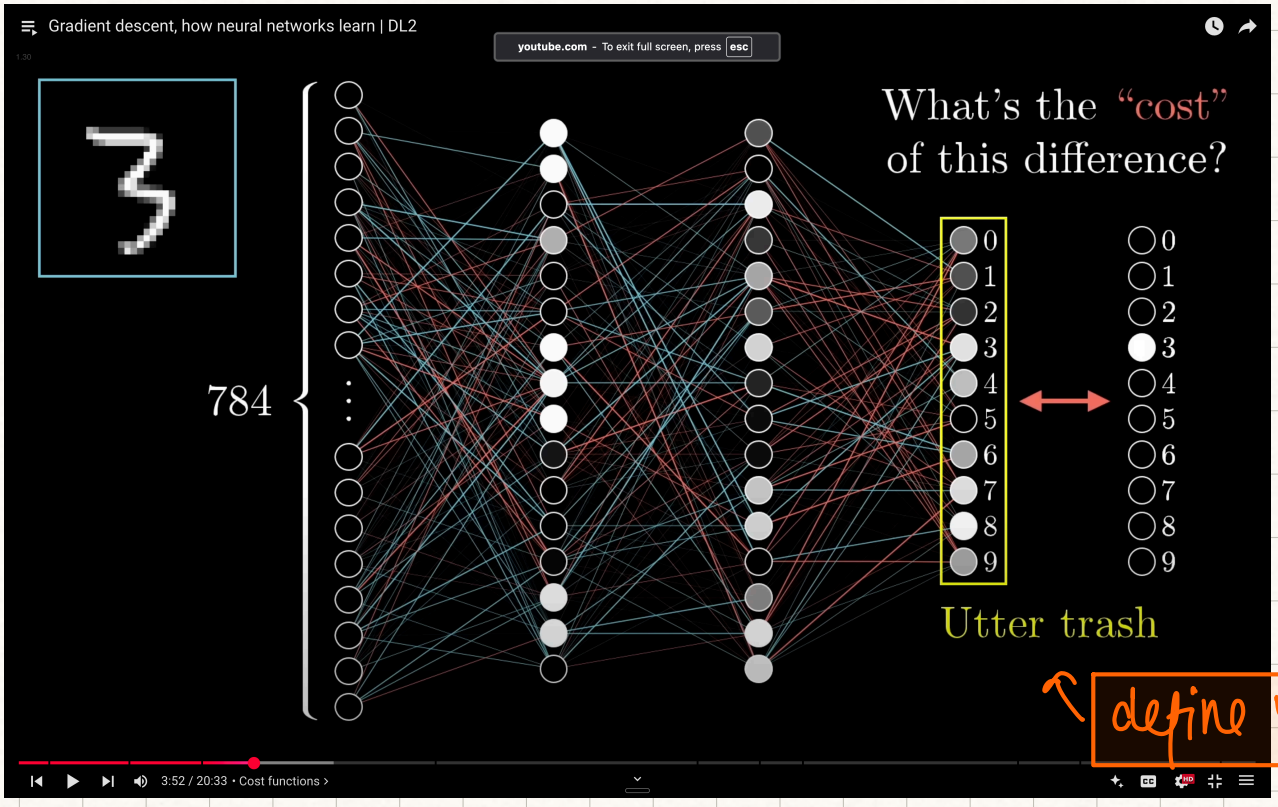
want an algo that can be shown a lot of training data → images + labels & it will adjust its w & b to improve its performance on training data

Hopefully it'll generalise to images beyond training data.

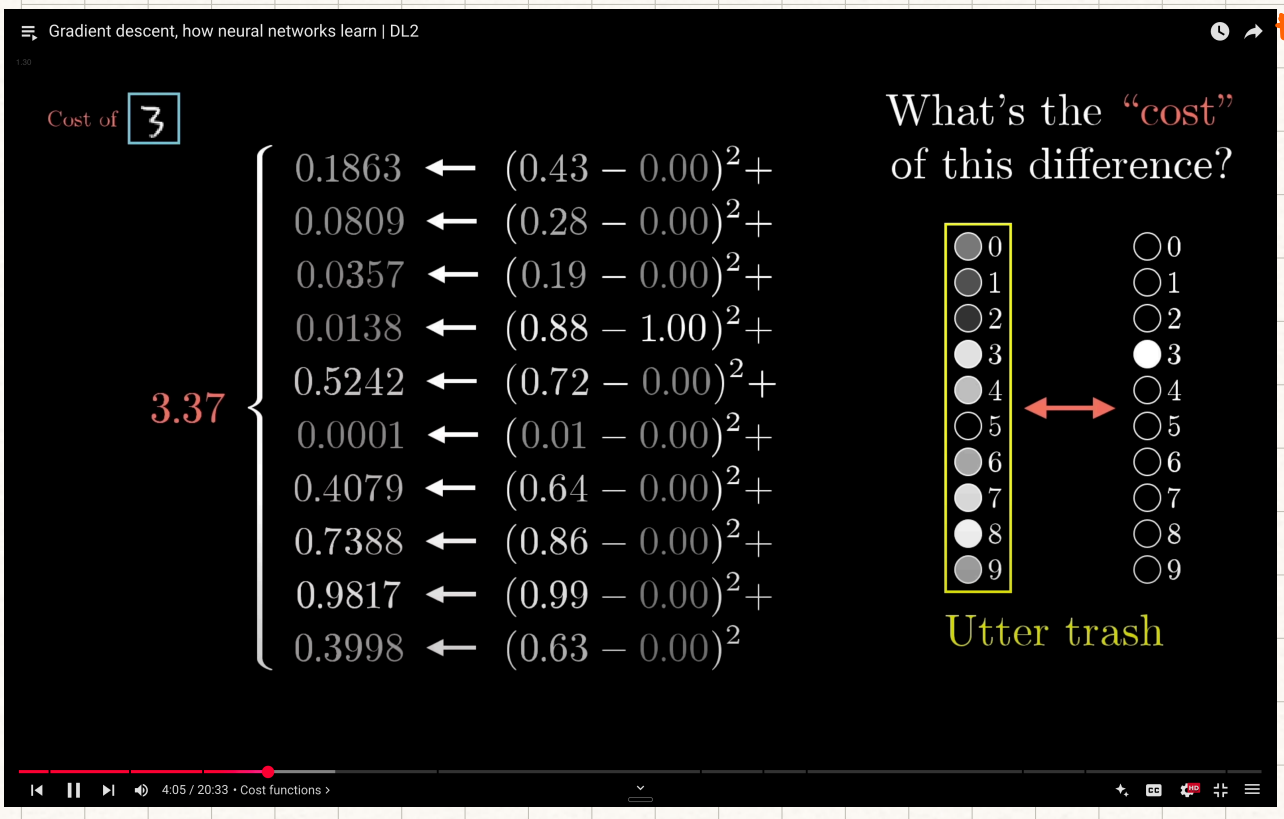


basically a calculus exercise, finding min. of a certain function.

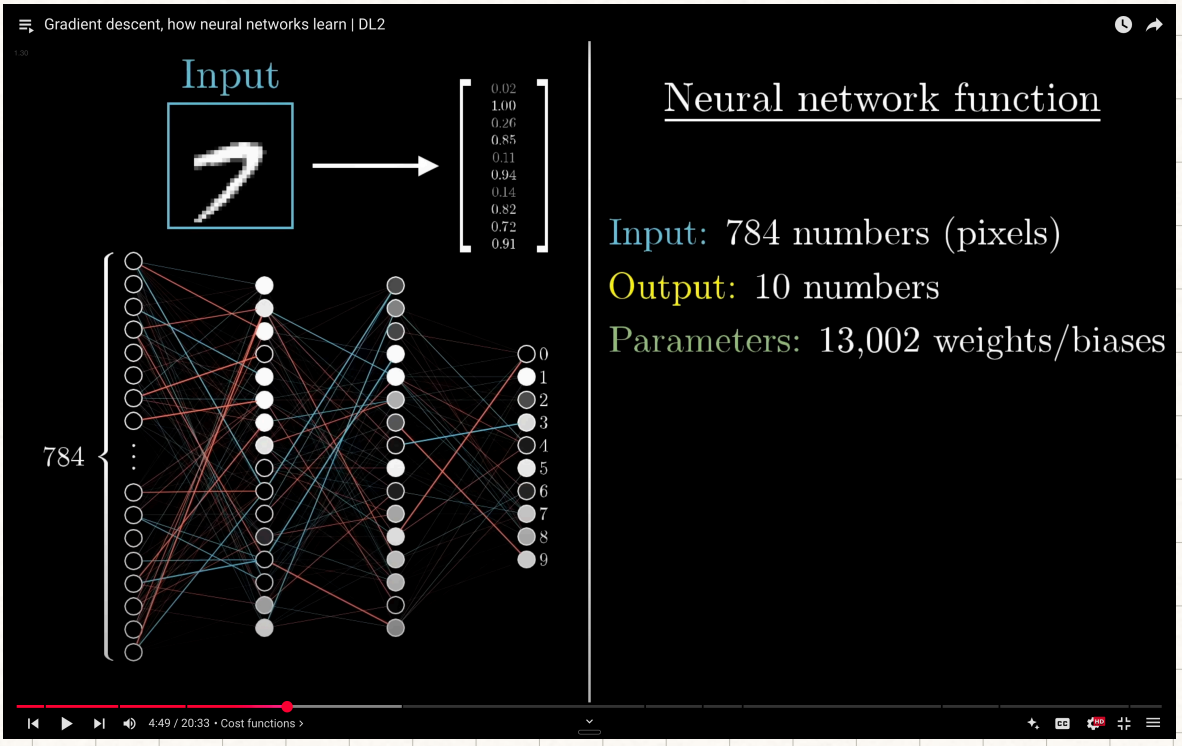


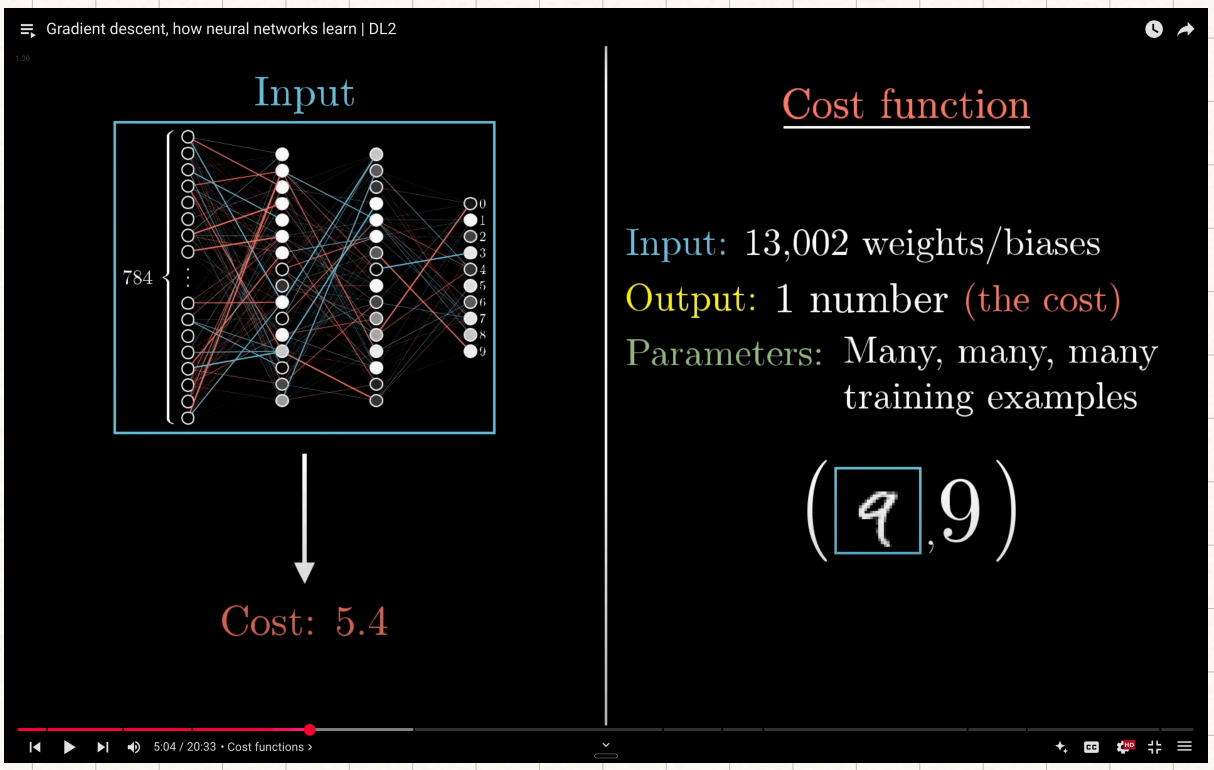


define "cost"



now, take avg over all training examples.





Cost function

$C(w_1, w_2, \ldots, w_{13,002})$

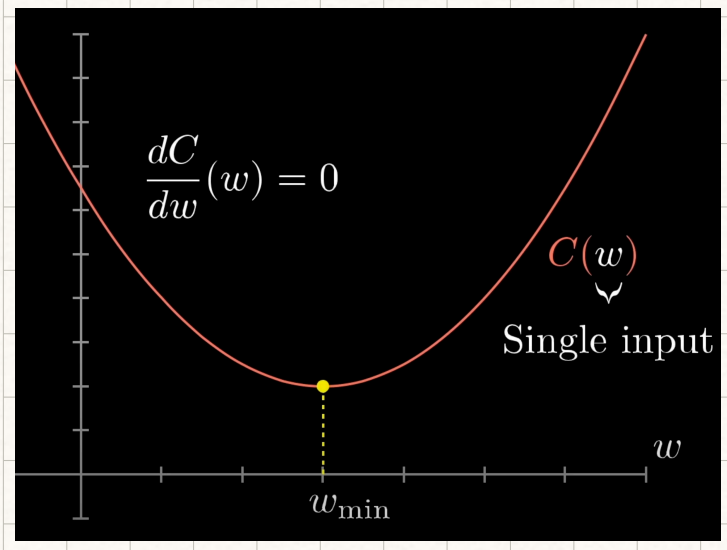
Weights and biases

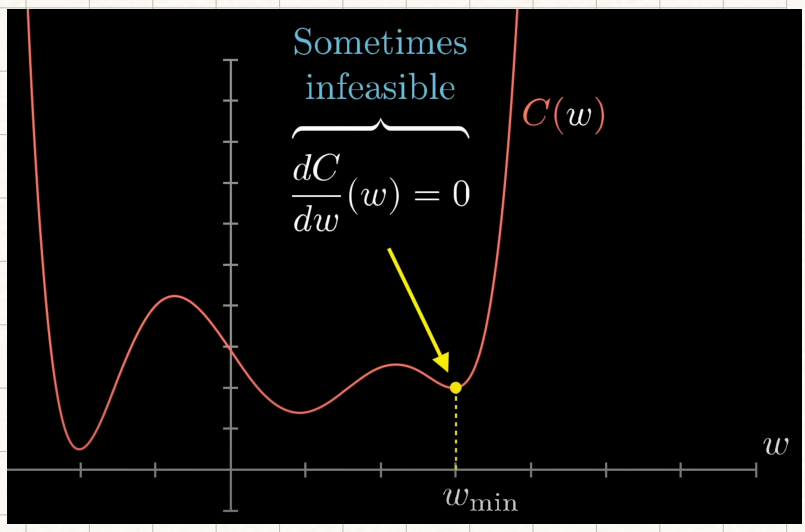
Cost function

$C(w)$

Single input

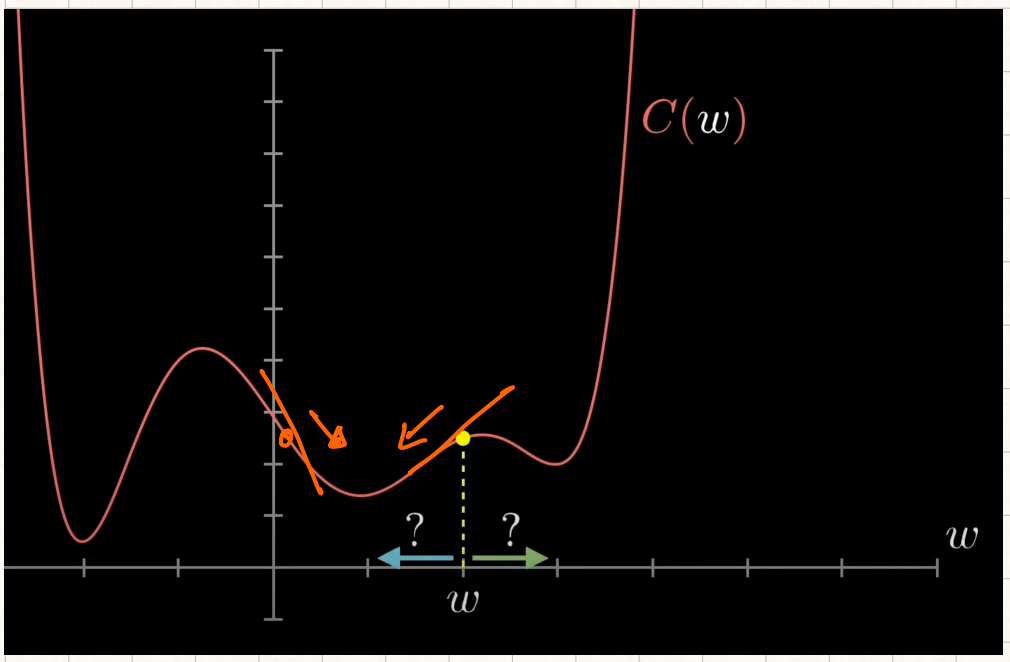
easier to imagine





---

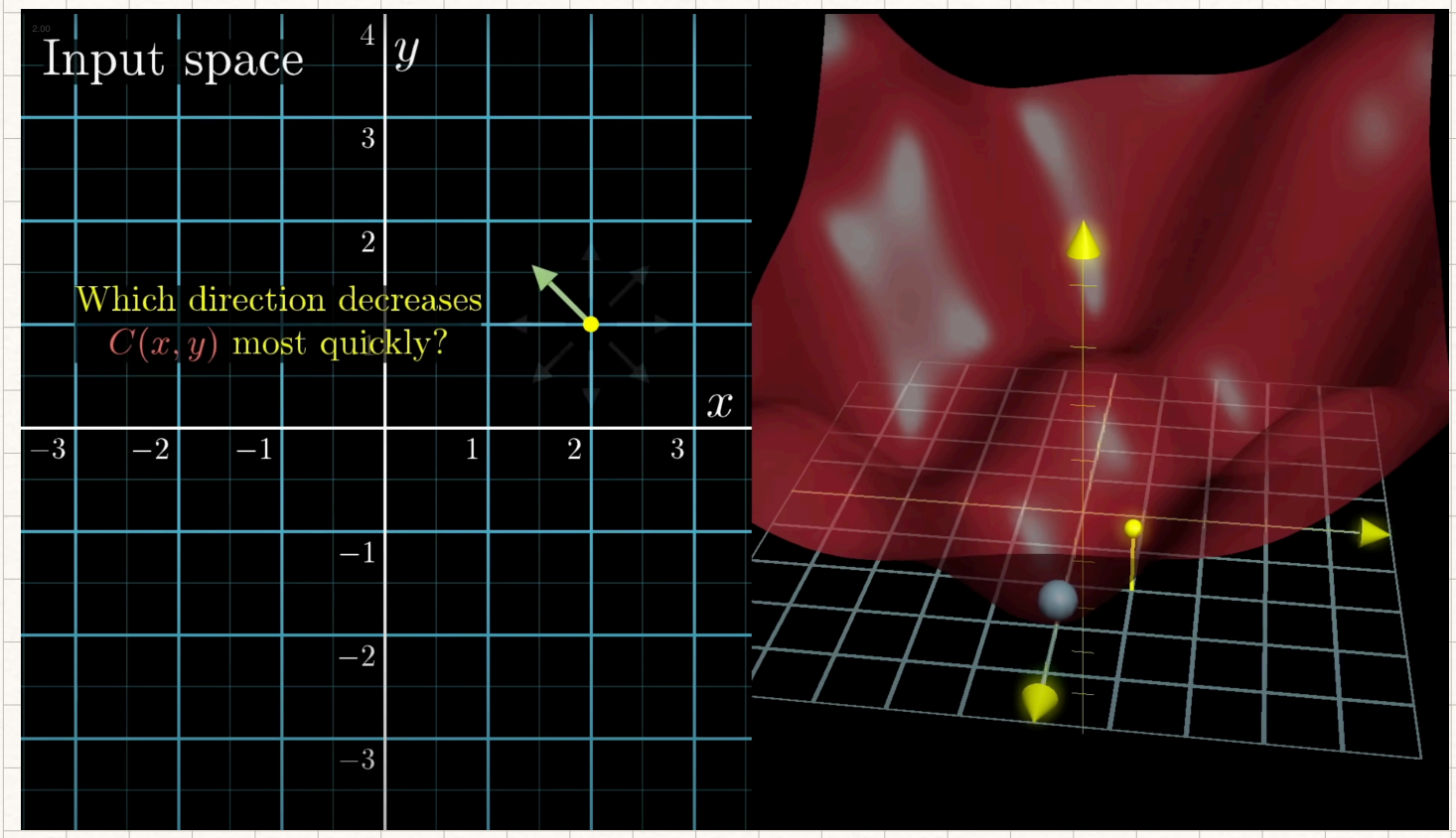
better way:

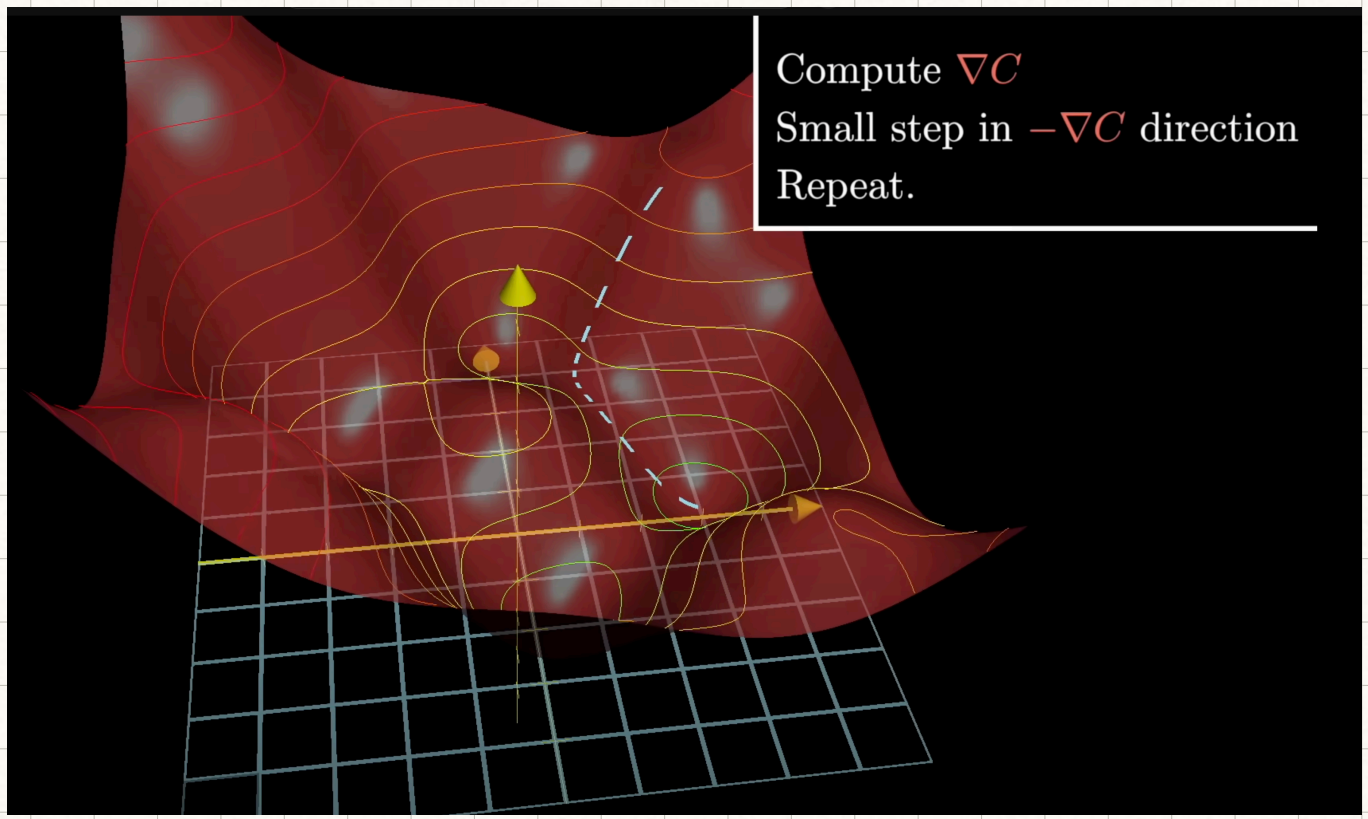


- "ball rolling down hill".
- many valleys that one might land in depending on which random input you start @.

if step size ∝ slope, then steps get smaller

↳ smaller,





13,002 weights and biases

How to nudge all weights and biases

2.25
−1.57

$$\vec{\mathbf{W}} = \begin{bmatrix} 2.43 \\ -1.12 \\ 1.47 \\ \vdots \\ -0.76 \\ 3.50 \\ 2.03 \end{bmatrix} \qquad -\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.18 \\ 0.45 \\ -0.51 \\ \vdots \\ 0.40 \\ -0.32 \\ 0.82 \end{bmatrix}$$

1.21 + 0.82 ⇒

some direction which says which nudges to weights ⇒ going to cause most rapid decrease to cost func^n.

Cost fn → avg over all training data.

So minimising it → better performance on all of those samples.

Back prop is how the gradient is calculated.

- Back prop next video.

  What happens to each wt & each bias

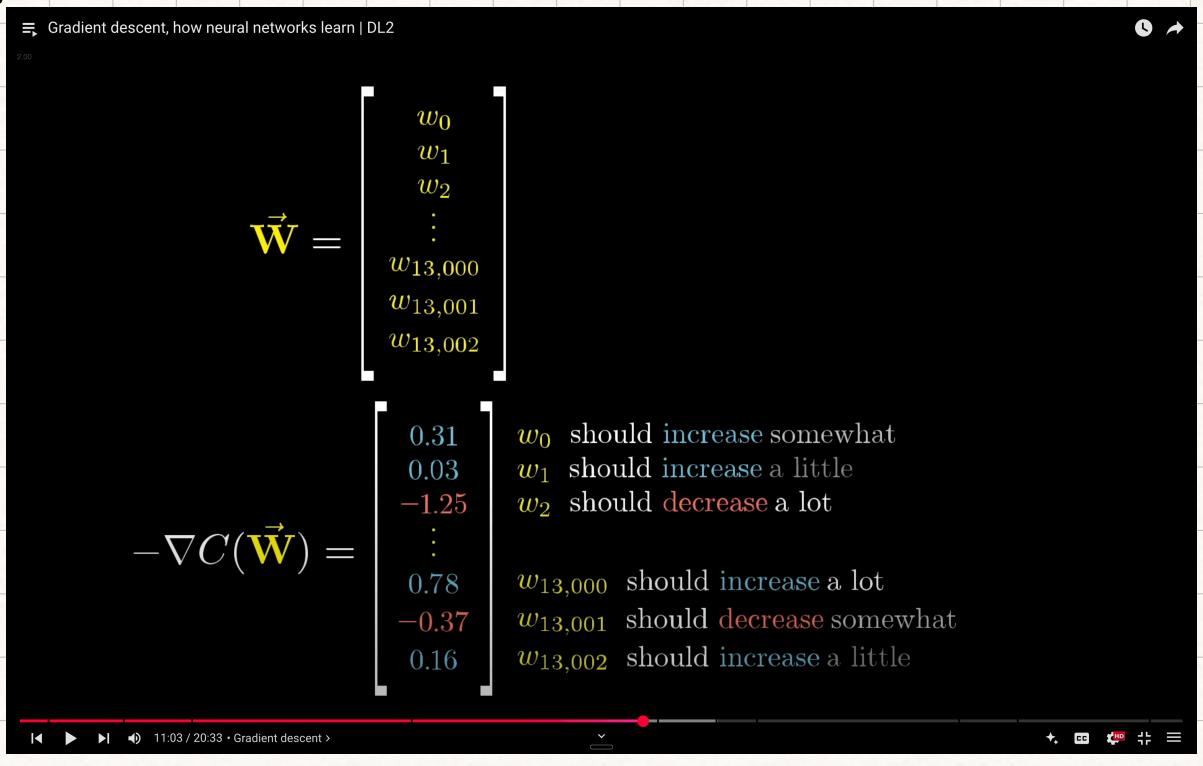
  ↳ "intuitive feel" of what's happening.

- imp for cost function to be continious & smooth

- process of repeatedly nudging the input of a fn by some multiple of the -ve gradient

  ↳ gradient descent.

- way to converge

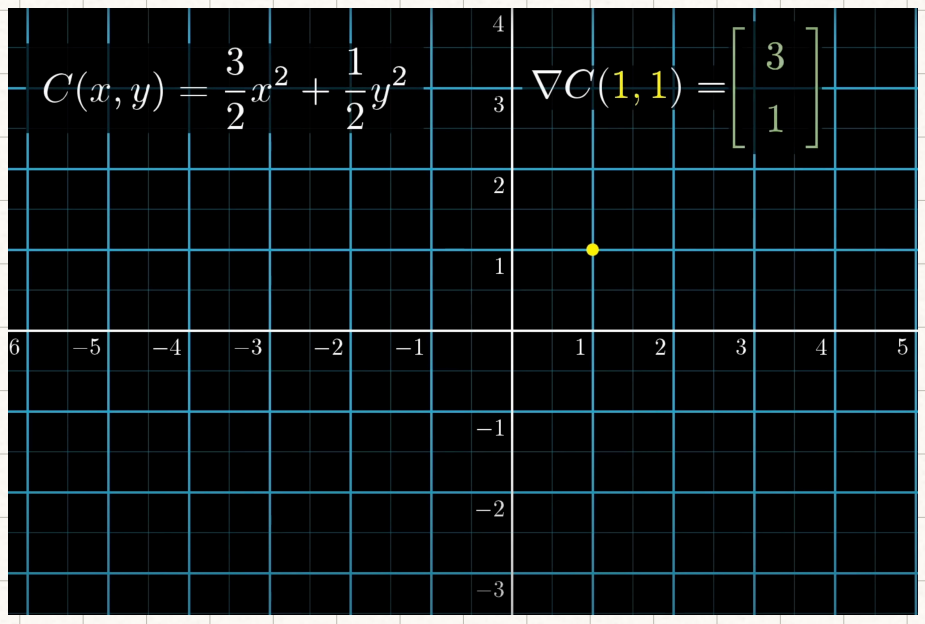


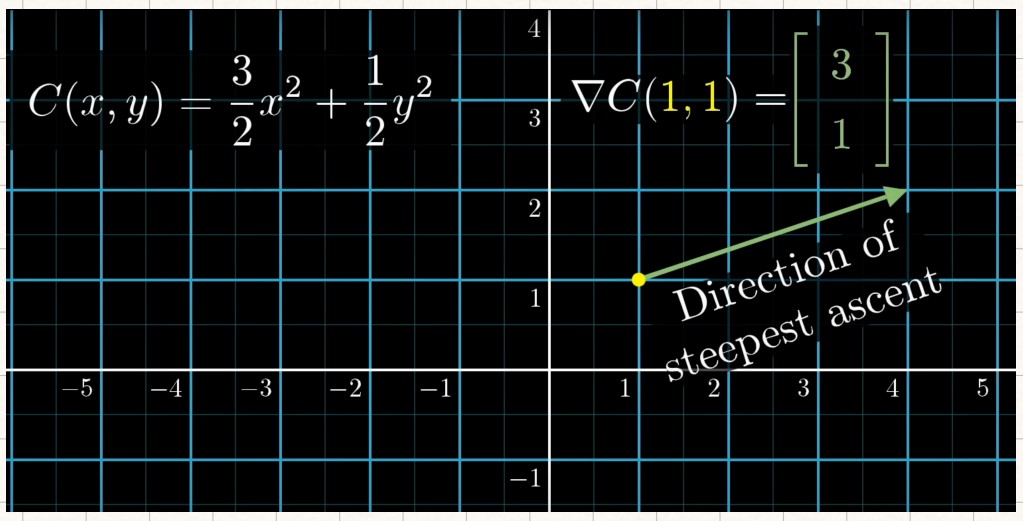
encodes relative imp. of each wt & bias.

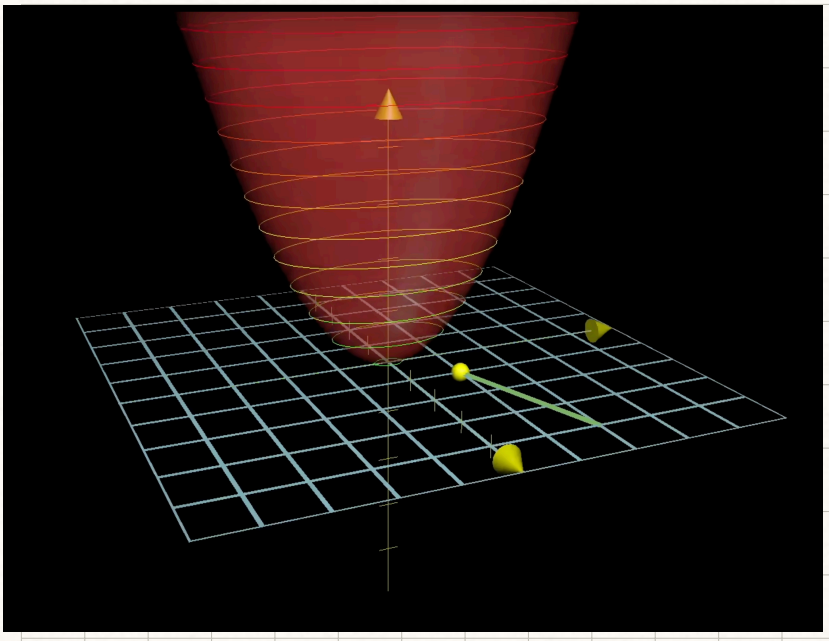
⇒ which change ⇒ most bang for buck.

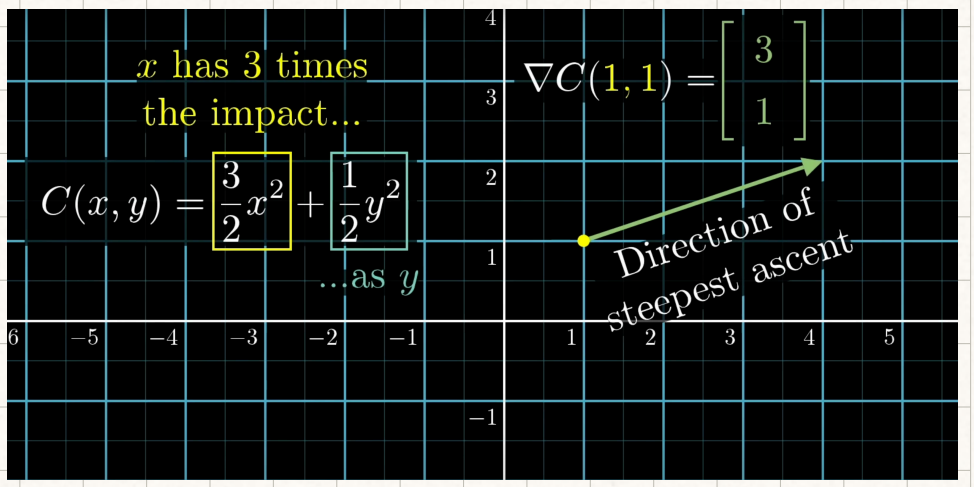
---

another way to think about direction.

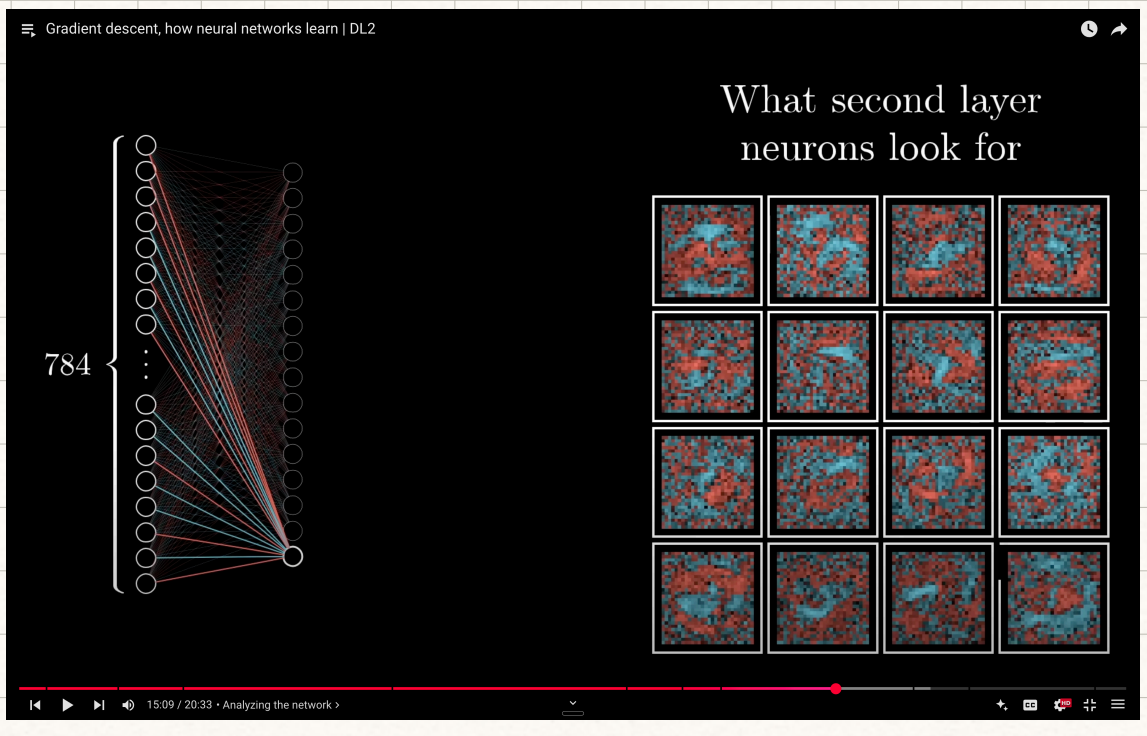






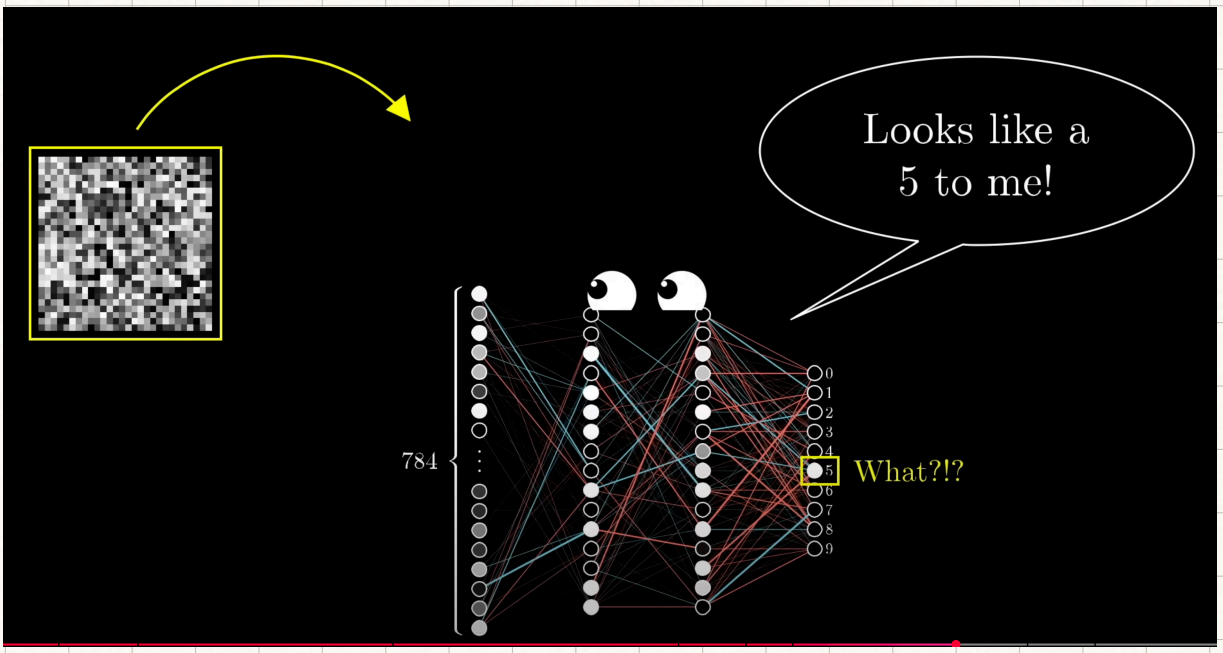

x has 3 times the impact...
$C(x,y) = \frac{3}{2}x^2 + \frac{1}{2}y^2$
...as y
$\nabla C(1,1) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$
Direction of steepest ascent

SOTA → 99.79 %.


Gradient descent, how neural networks learn | DL2
What second layer neurons look for
784
15:09 / 20:33 • Analyzing the network ›

does not look for the "edges" which we had thought of initially...

- also when given random noise it confidently predicts "5"



- it has no idea how to draw numbers

UNDERSTANDING DEEP LEARNING REQUIRES RETHINKING GENERALIZATION

Chiyuan Zhang* Massachusetts Institute of Technology chiyuan@mit.edu
Samy Bengio Google Brain bengio@google.com
Moritz Hardt Google Brain mrtz@google.com
Benjamin Recht† University of California, Berkeley brecht@berkeley.edu
Oriol Vinyals Google DeepMind vinyals@google.com

→ shuffle labels of CNN
- same training accuracy

A Closer Look at Memorization in Deep Networks

→ little smarter

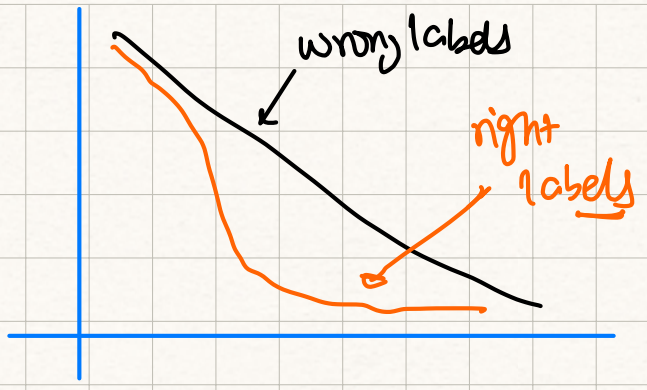
Devansh Arpit *[1 2] Stanisław Jastrzębski *[3] Nicolas Ballas *[1 2] David Krueger *[1 2] Emmanuel Bengio [4]
Maxinder S. Kanwal [5] Tegan Maharaj [1 6] Asja Fischer [7] Aaron Courville [1 2 8] Yoshua Bengio [1 2 9]
Simon Lacoste-Julien [1 2]

The Loss Surfaces of Multilayer Networks

Anna Choromanska achoroma@cims.nyu.edu
Mikael Henaff mbh305@nyu.edu
Michael Mathieu mathieu@cs.nyu.edu
Gérard Ben Arous benarous@cims.nyu.edu
Yann LeCun yann@cs.nyu.edu

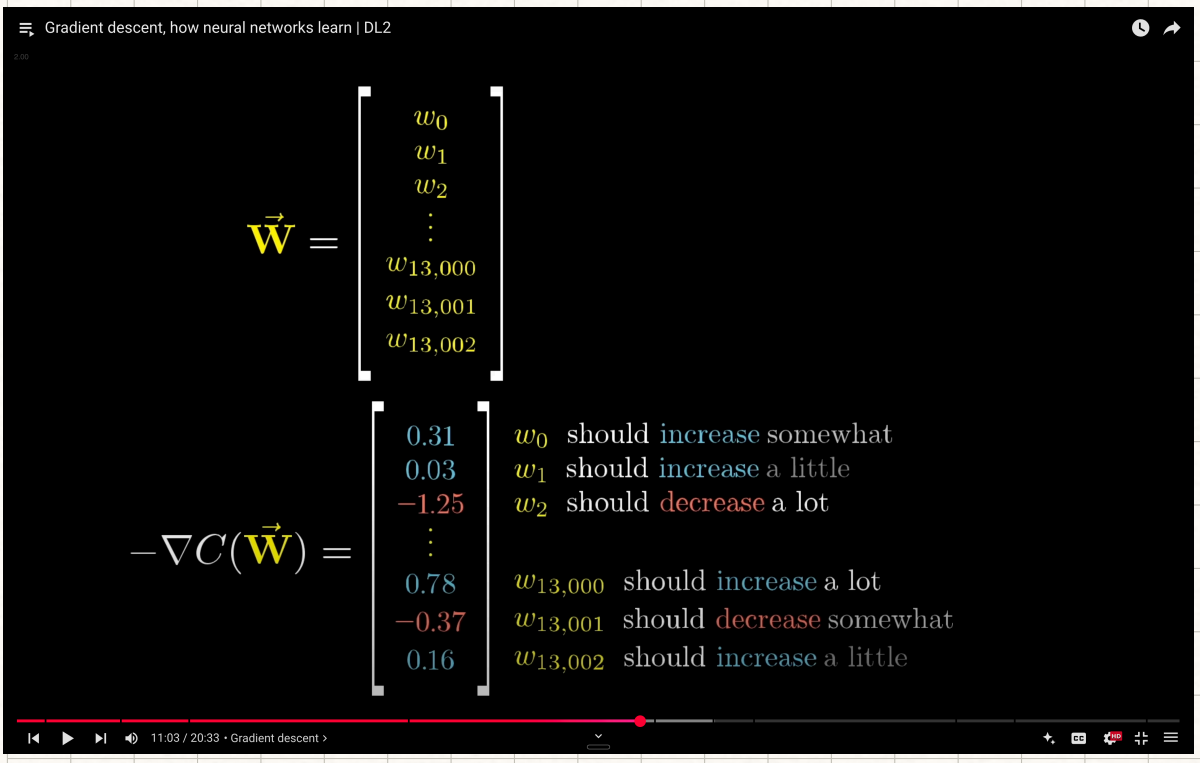
Courant Institute of Mathematical Sciences
New York, NY, USA

roughly equal quality local minima.



DL03 - 3B1B Playlist || Back prop intuition.

- Back prop is the algo used to compute the gradient.



C(W) = some num

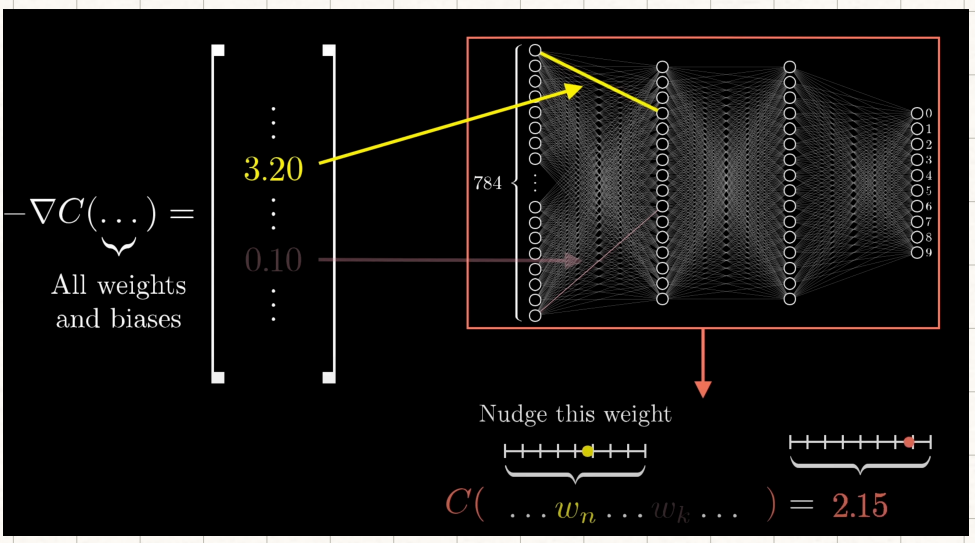
$-\nabla C(W)$ = tells how to change wt & bias to (gradient) most efficiently decrease the cost.

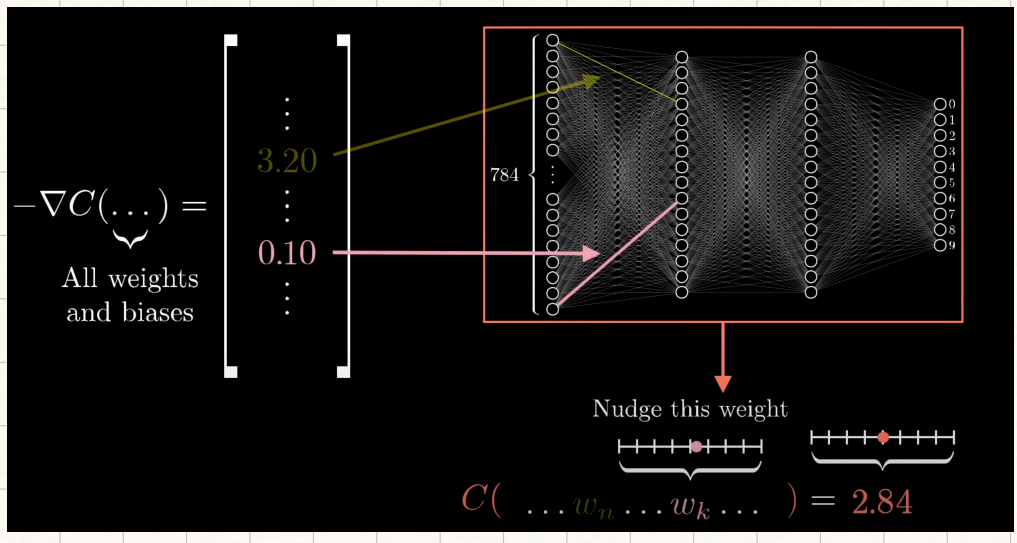
- direction in 13K dim → xx {bad intuition}.

- magnitude tells us how sensitive the C is to each wt & bias

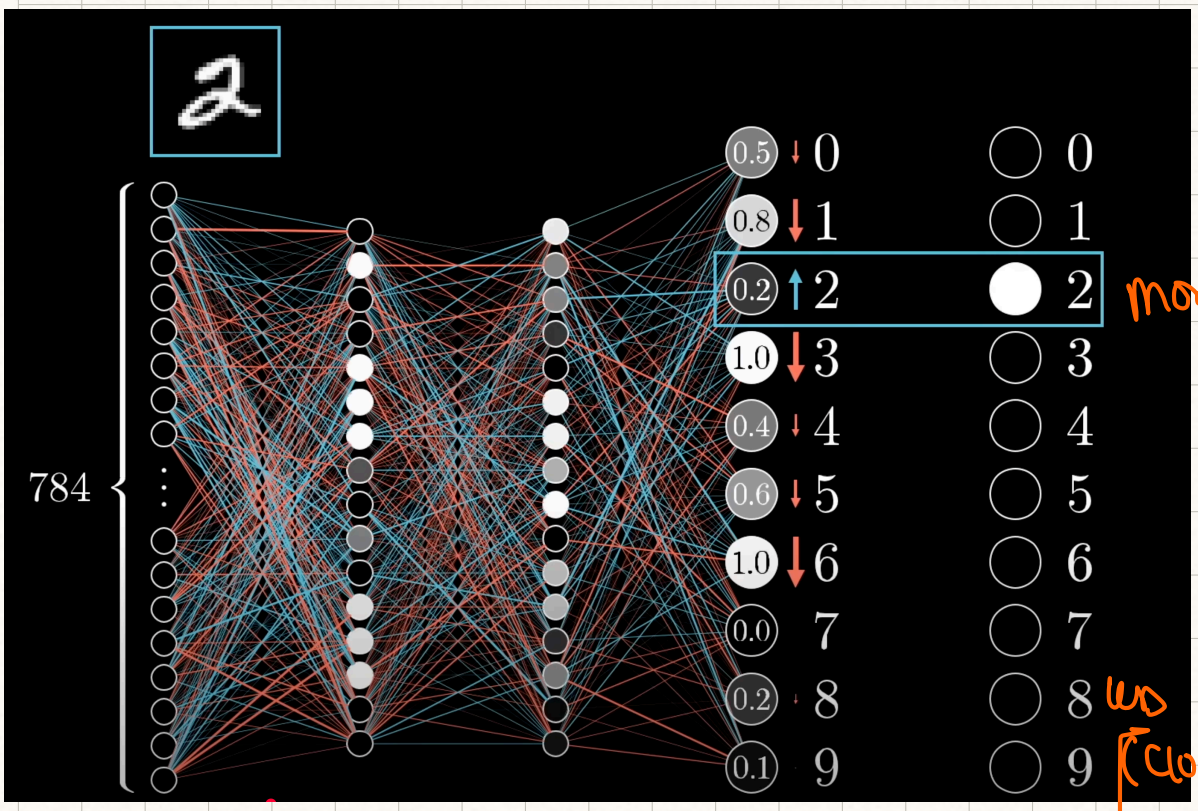
Eg:

Loss is 32x more sensitive to change in yellow as compared to a change in pink.



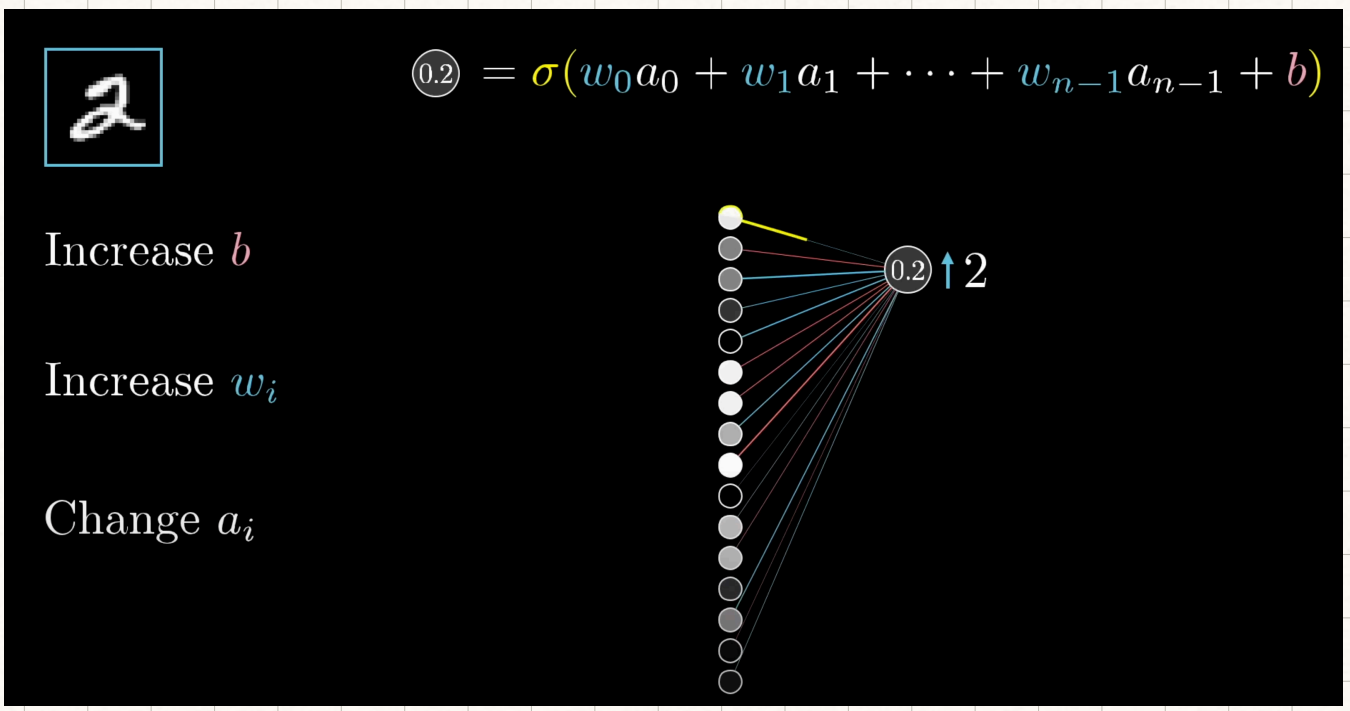


Intuitive back prop for 1 example:



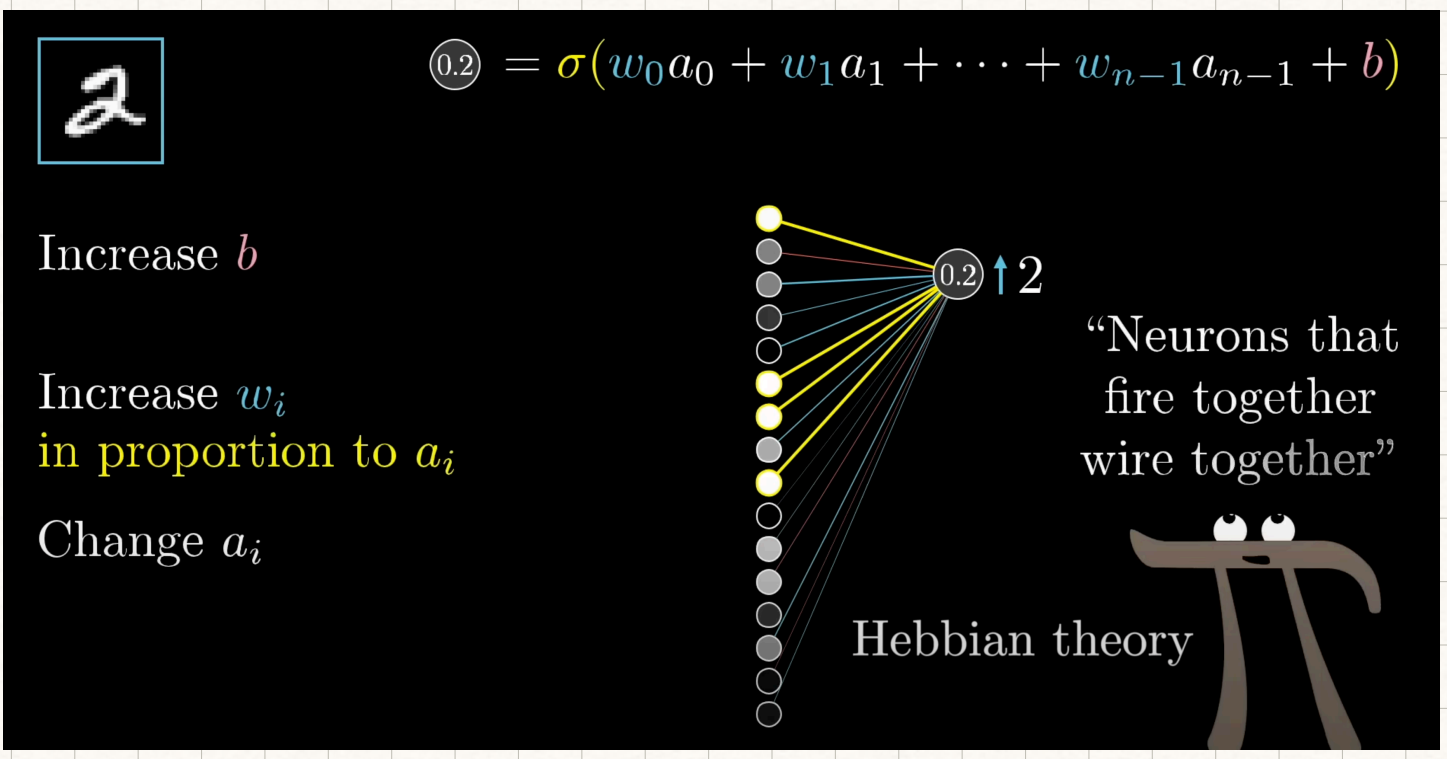
more imp inc

wts imp decrease (close to where it should be)



- Connection w/ brightest neurons from prev layer inv. largest effect.

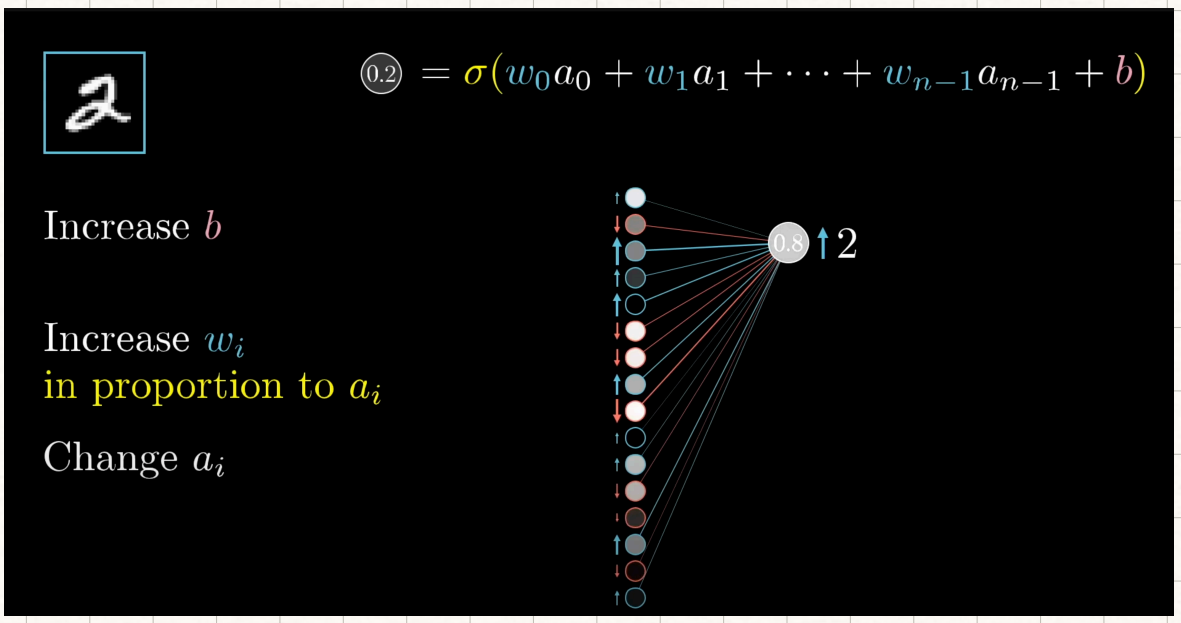
- Inc wt. of a link w/ a activation in prev layer, it has stronger influ. on C than inc the wt of connections w/ dimmer neurons.



- Strengthening connection b/w neurons that are most active & the ones we wish to become more active.

- Neurons firing while seeing a two, get more strongly linked to those "thinking about a two".

3rd way:



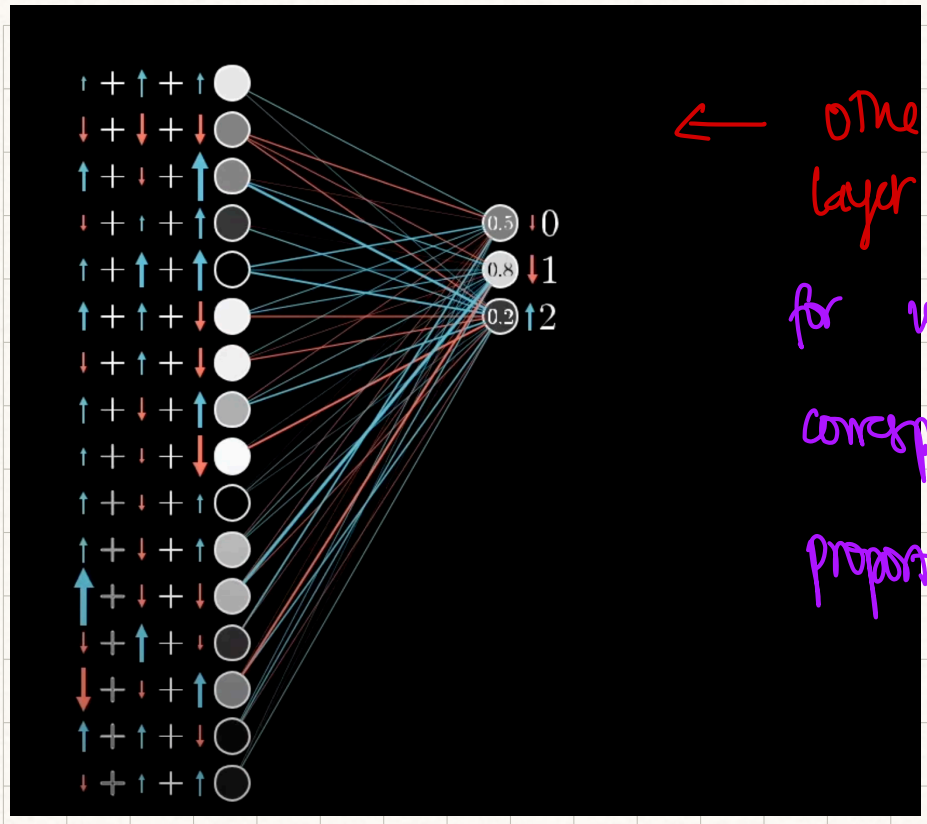
- changing activation

- everything connected w/ +ve wt got brighter
- everything connected w/ -ve wt got dimmer.

- most bang for buck by scaling changes ∝ to wts.

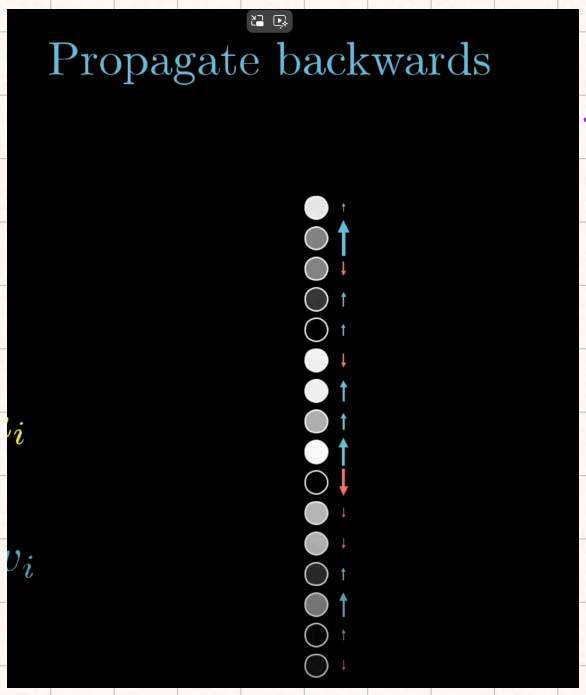
This is only for the '2' neuron in last layer

0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9



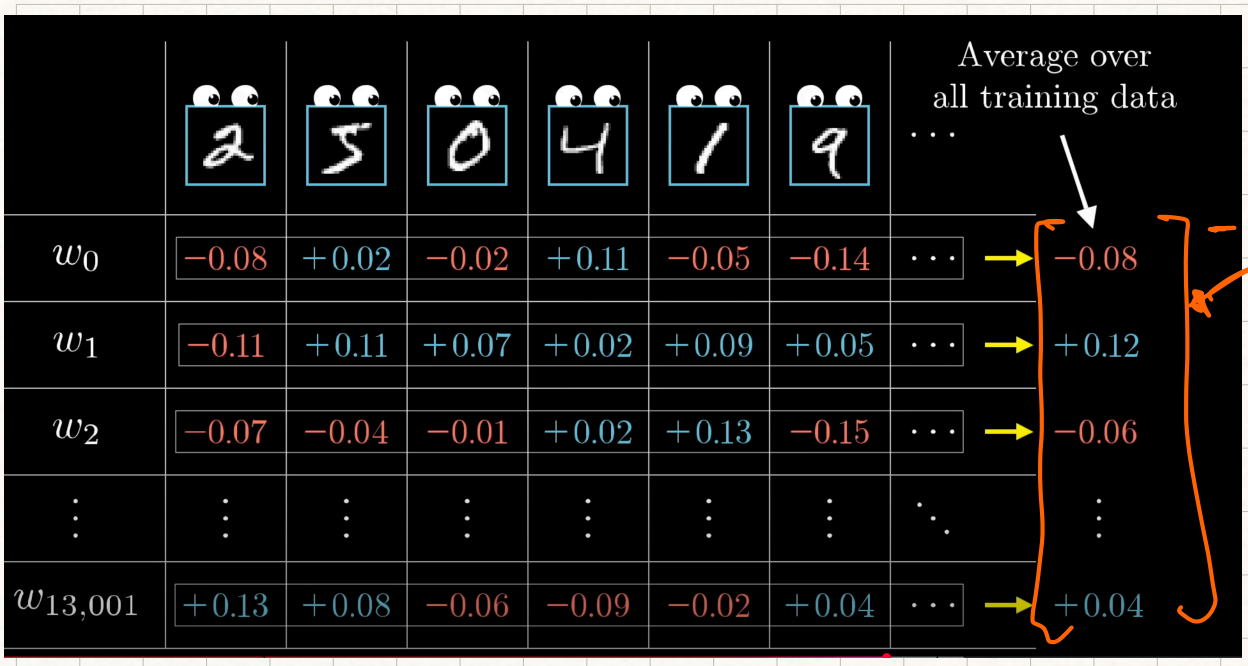
← other neuron in the output layer w. other desires.

for what should happen to corresponding layer in proportion to their wts.



← you get a list that you want to happen to 2nd to last layer.

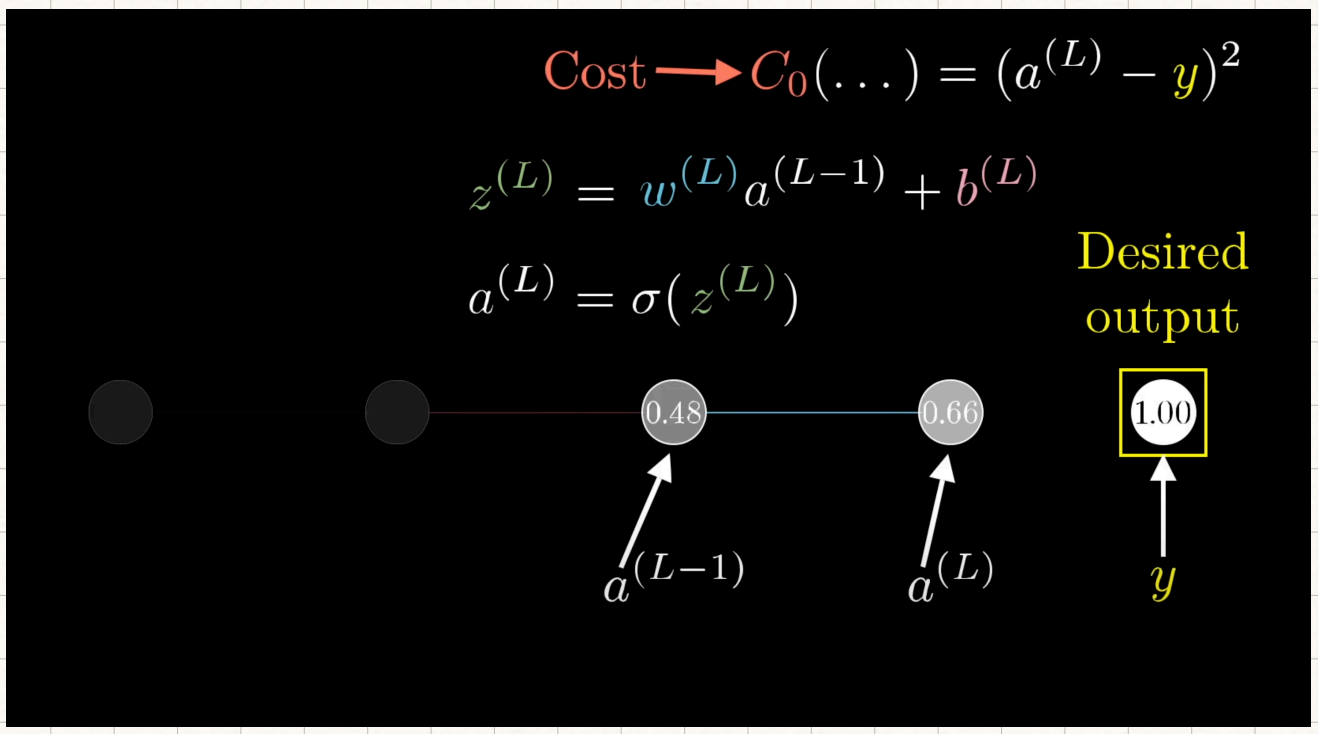
Then recursively apply the same process to the prev layer.

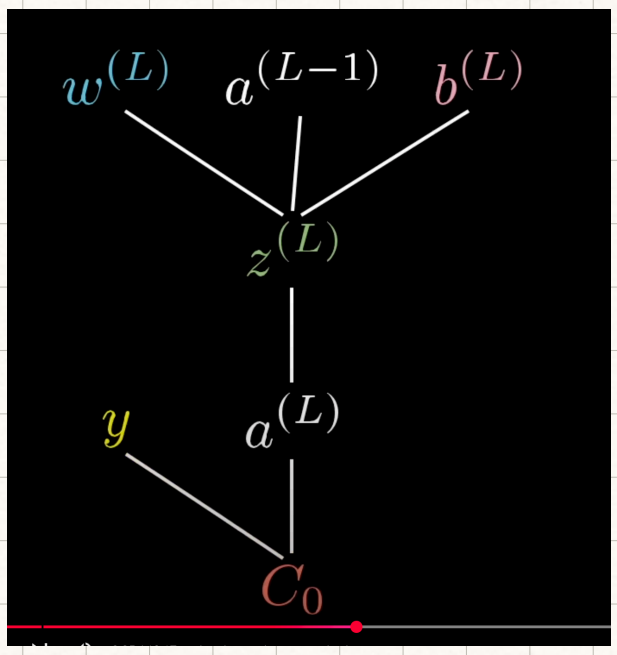


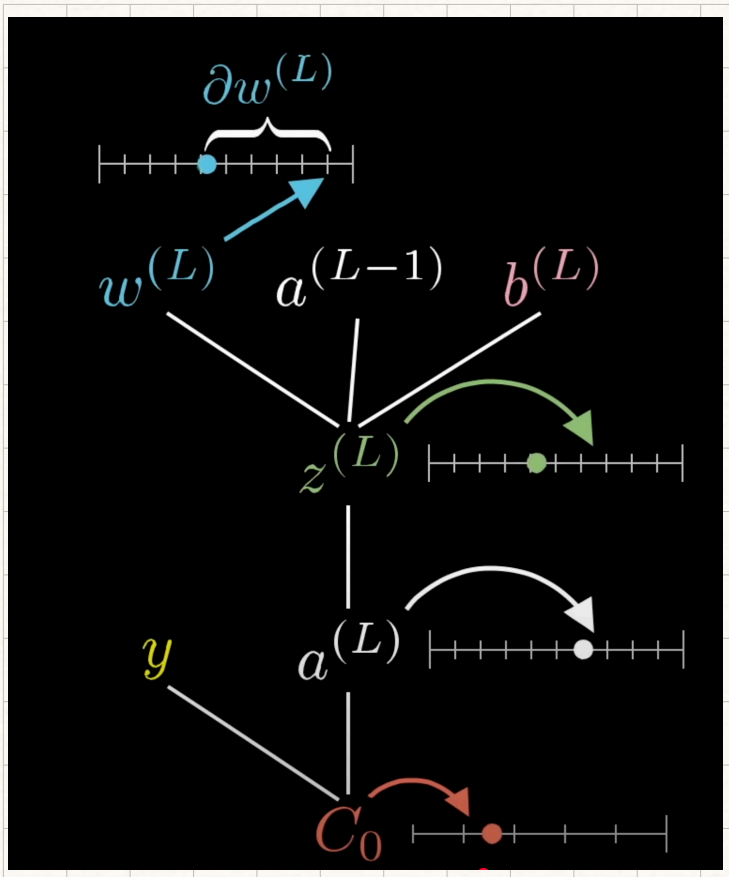
you do for all training ex. & avg the wishes of each training example.

mini-batches

DL 4 Backprop Calculus







How sensitive is the cost fn. to small changes in our wt $w^{(L)}$?

$$\frac{\partial C_0}{\partial w^{(L)}}$$

whatever the resultant nudge to the cost is

small nudge to $w(\Delta o \cdot g)$

We break things up as $\Delta w \to \Delta z \to \Delta a \to \Delta C$

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} * \frac{d a^{(L)}}{\partial z^{(L)}} * \frac{\partial C_0}{\partial a^{(L)}}$$

⇐ chain rule

↳ gives sensitivity of C to small changes in $w^{(L)}$.

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

$$C_0 = (a^{(L)} - y)^2$$

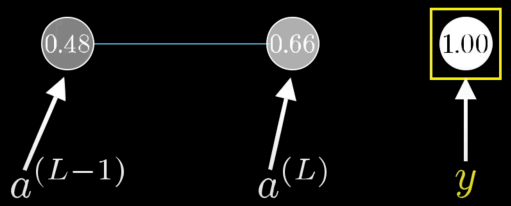
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$\frac{\partial C0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$



The amount that $dw^{(L)}$ influenced the last layer, depends on how strong the previous neuron is.

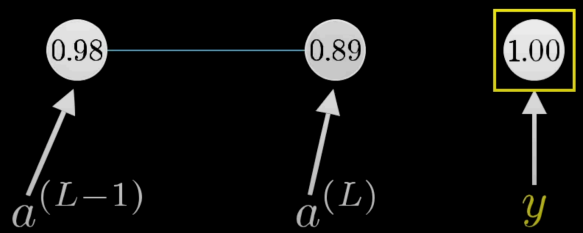
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$C_0 = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$\underbrace{\frac{\partial C}{\partial w^{(L)}}}_{\text{Derivative of full cost function}} = \overbrace{\frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}}^{\text{Average of all training examples}}$$



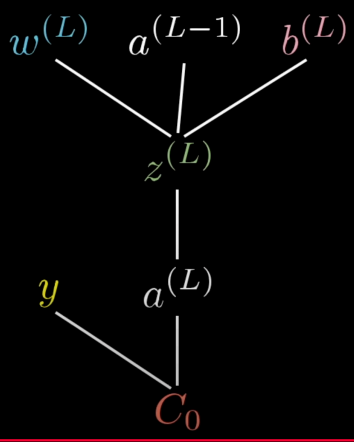
wt bias →

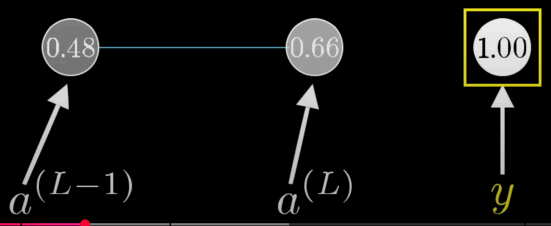
$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = 1\sigma'(z^{(L)})2(a^{(L)} - y)$$

$$C_0 = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$





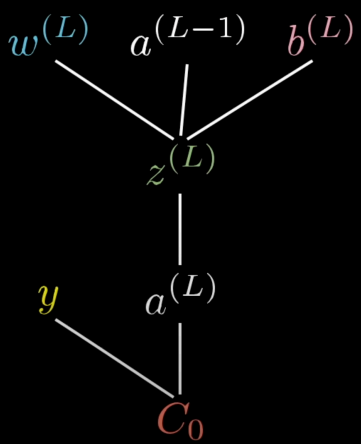
wrt to act of prev layer

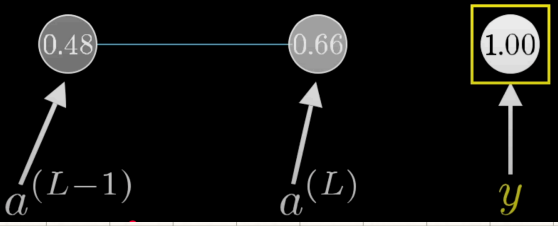
$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = w^{(L)}\sigma'(z^{(L)})2(a^{(L)} - y)$$

$$C_0 = (a^{(L)} - y)^2$$

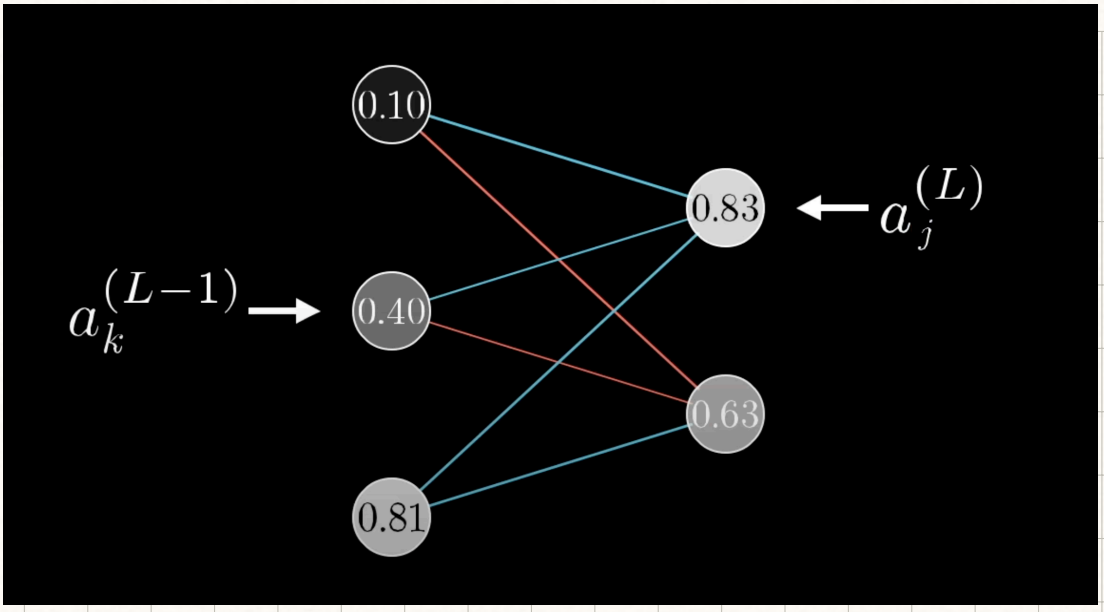
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$





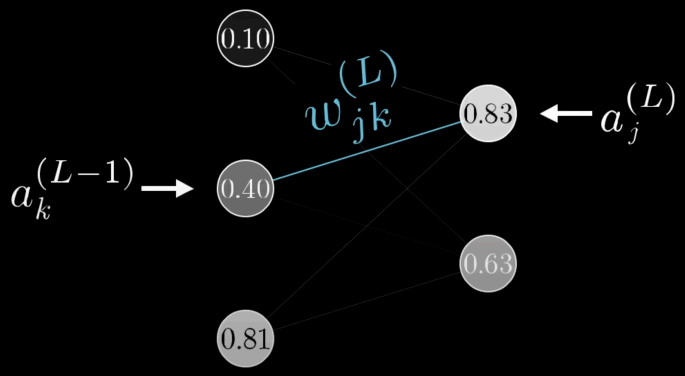
now, what if mutiple neurons?



$$z_j^{(L)} = w_{j0}^{(L)} a_0^{(L-1)} + w_{j1}^{(L)} a_1^{(L-1)} + w_{j2}^{(L)} a_2^{(L-1)} + b_j^{(L)}$$

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$



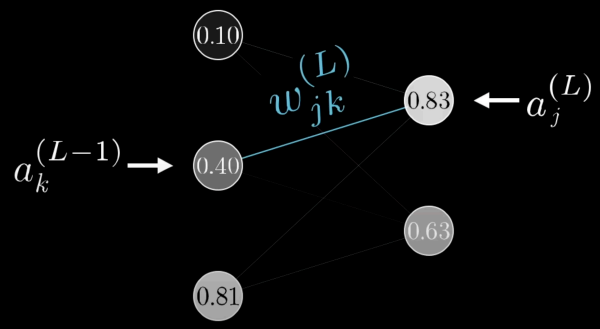
derivative is also vv similar/same!

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$z_j^{(L)} = \cdots + w_{jk}^{(L)} a_k^{(L-1)} + \cdots$$

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$



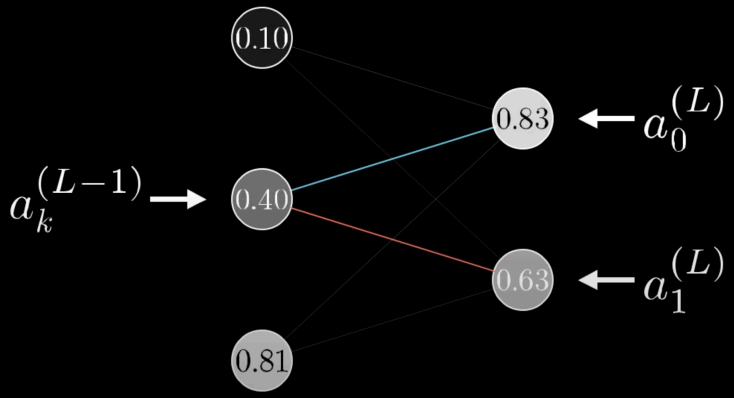
However derivative for prev layer activation is slightly different:

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \underbrace{\sum_{j=0}^{n_L - 1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}}_{\text{Sum over layer L}}$$

$$z_j^{(L)} = \cdots + w_{jk}^{(L)} a_k^{(L-1)} + \cdots$$

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$



Nok!

I plan to do another pass to beautify the notes.

Currently it's just raw.

JUNK: 7

- 3 → 28x28 pixels. ; low res
- Brain can recognise 3's really easily ; Crazy ;

- Why the layers ; You understand "learning" in ML.
- NN to recognise handwritten digits → MNIST

CNN ⟶ Image recognition

LSTM ⟶ Speech recognition.