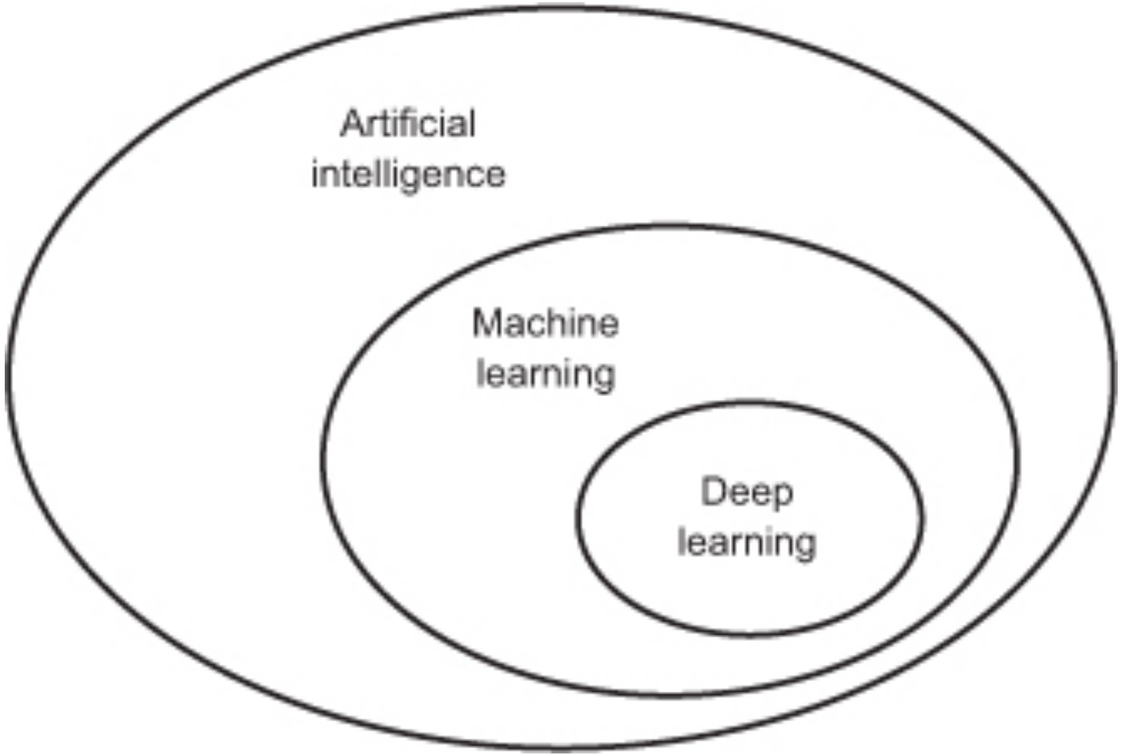
Part 1
FCNN, Learning, etc.

Frank Lindseth, IDI, NTNU

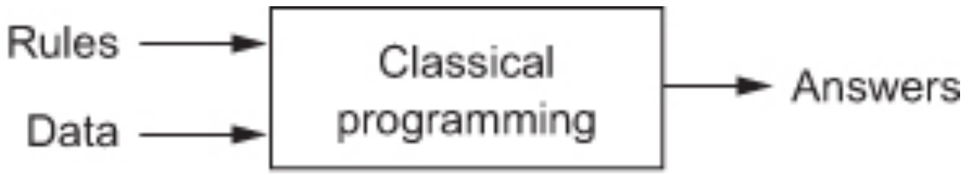
CV & AI vs. ML vs. DL

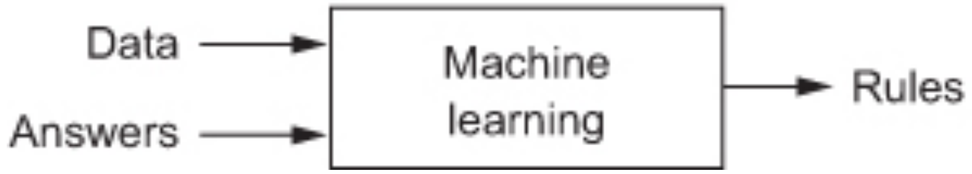
- **CV**: make a computer **see**.
  - Visual Intelligence (VI)
- **AI**: make a computer **think**.
  - the effort to automate intellectual tasks normally performed by humans
  - start (1950): explicit rules, no learning, symbolic AI, expert systems (1980s)

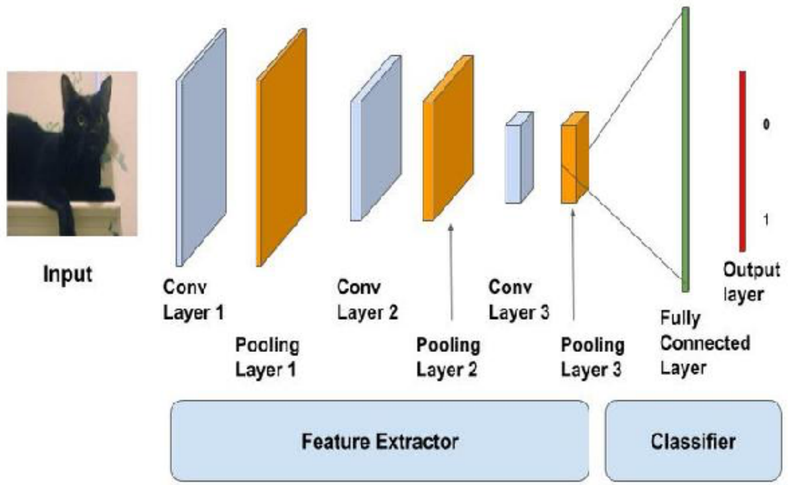


CV & AI vs. ML vs. DL (2)

- **ML:** learn by example
  - new programming paradigm: **trained** instead of explicitly programmed
  - *Given*: data and answers. *Find*: rules. *Apply* rules on new data.
  - present examples relevant to task -> finds statistical structure in examples -> come up with rules for automating the task (ex. tagging vacation pictures)
  - most popular and most successful subfield of AI (from 1990s)
  - driven by faster hardware and larger datasets
  - ML related to mathematical statistics but:
    - ML deal with large, complex datasets
    - ML more empirically / less theoretically, i.e. engineering oriented.
- ML learns «Classification rules»
  - For images you still need to do the feature engineering (vs. feature learning (DL)).



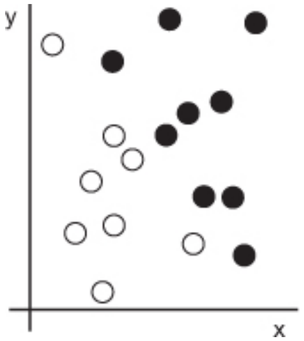




CV & AI vs. ML vs. DL (3)

- **ML** algorithms needs:
  - *Input data points*:
    - image classification: **pictures**
    - speech recognition: **sound files**
  - *Examples of the expected output*:
    - image classification: **tags** («dog», «cat»)
    - speech recognition: **transcripts** of sound files
  - *A way to measure whether the algorithm is doing a good job*:
    - current output vs. **expected** output
    - feedback signal to adjust the way the algorithm works, i.e. **learning**
    - to meaningfully **transform** data
    - learn useful **representations** of the input data, i.e. a different way to look at data, e.g. **encode** (RGB vs. HSV, **task**)

Point black or white?

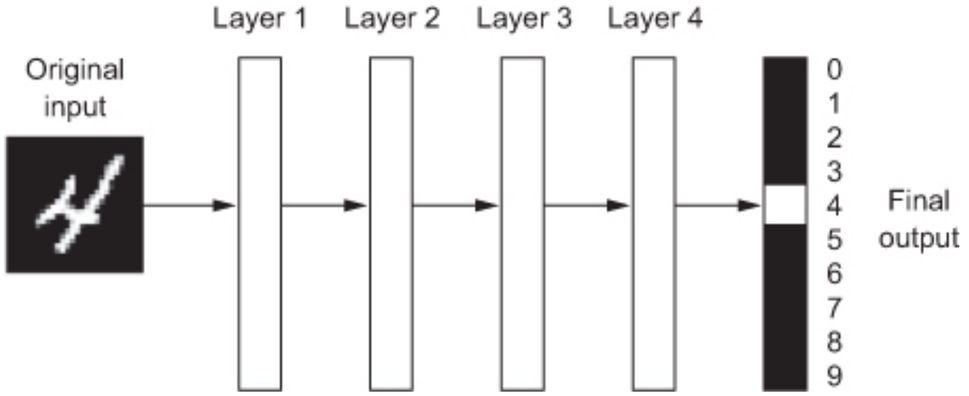


inputs: **coordinates**
expected outputs: **colors**
good job measure: percentage **correctly** classified

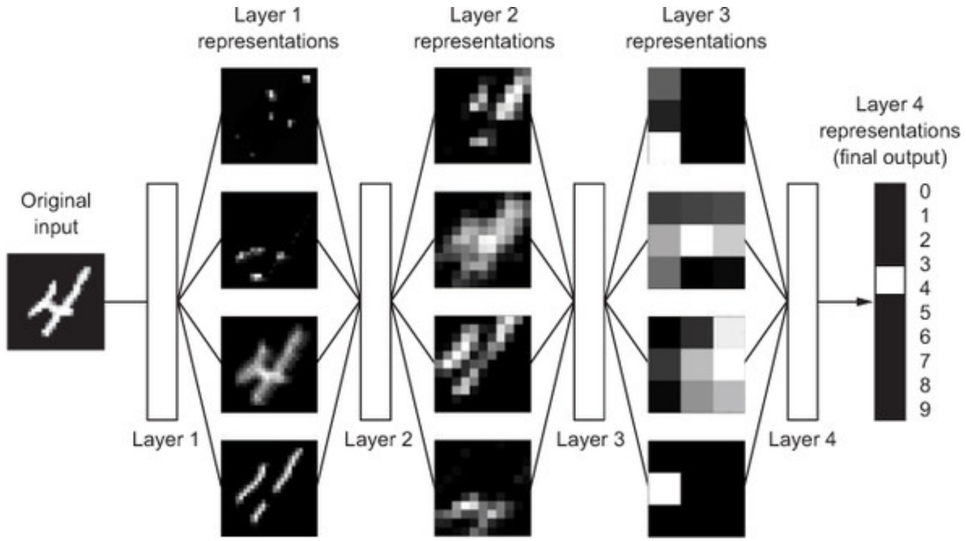
CV & AI vs. ML vs. DL (4)

- **DL:** «Deep» learning vs. (shallow) learning
  - learning successive **layers** of increasingly meaningful representations (depth of the model)
  - **layered** representations learning and **hierarchical** representations learning
  - learns representations via models called **neural networks**
  - network **transforms** the digit image into **representations** that are increasingly **different** from the original image and increasingly **informative** with regard to some task.
  - a **multistage** way to learn data representations, information goes through successive **filters** and comes out increasingly purified.

A deep neural network for digit classification



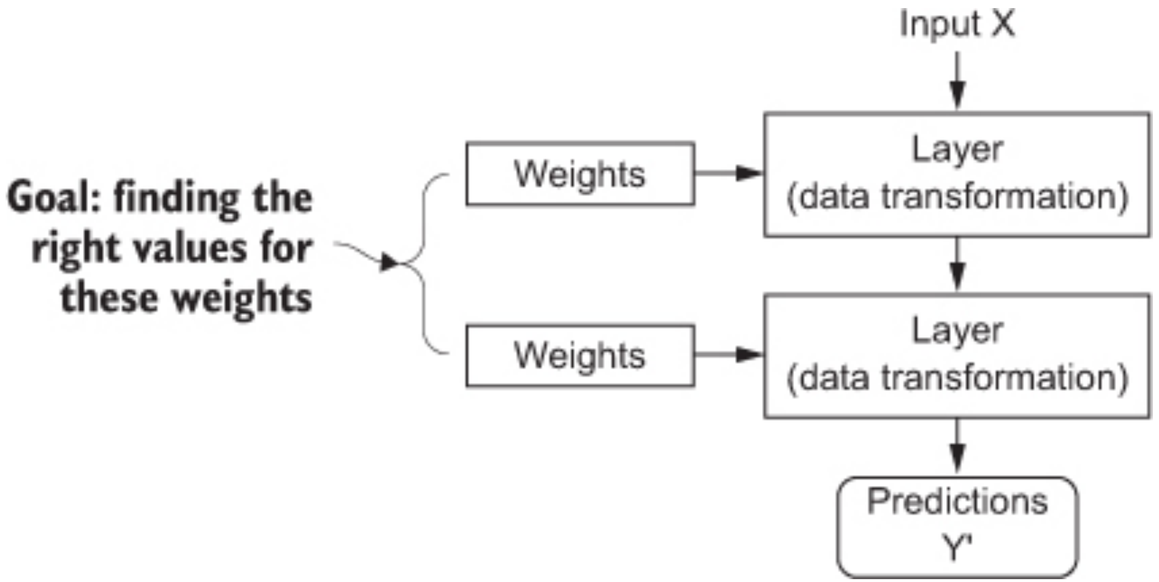
Deep representations learned by a digit-classification model



CV & AI vs. ML vs. DL (5)

- **DL:** How does it work?
  - **mapping** example inputs (image) to their associated targets (label «cat»)
  - DNN: input-to-target mapping via a **sequence** of simple data transformations (layers) learned by exposure to examples.
  - transformation implemented by a layer is parameterized by its **weights** / parameters
  - **learning** means finding the weights such that the network will correctly map example inputs to their associated targets.
  - A DNN can contain tens of **millions** of parameters

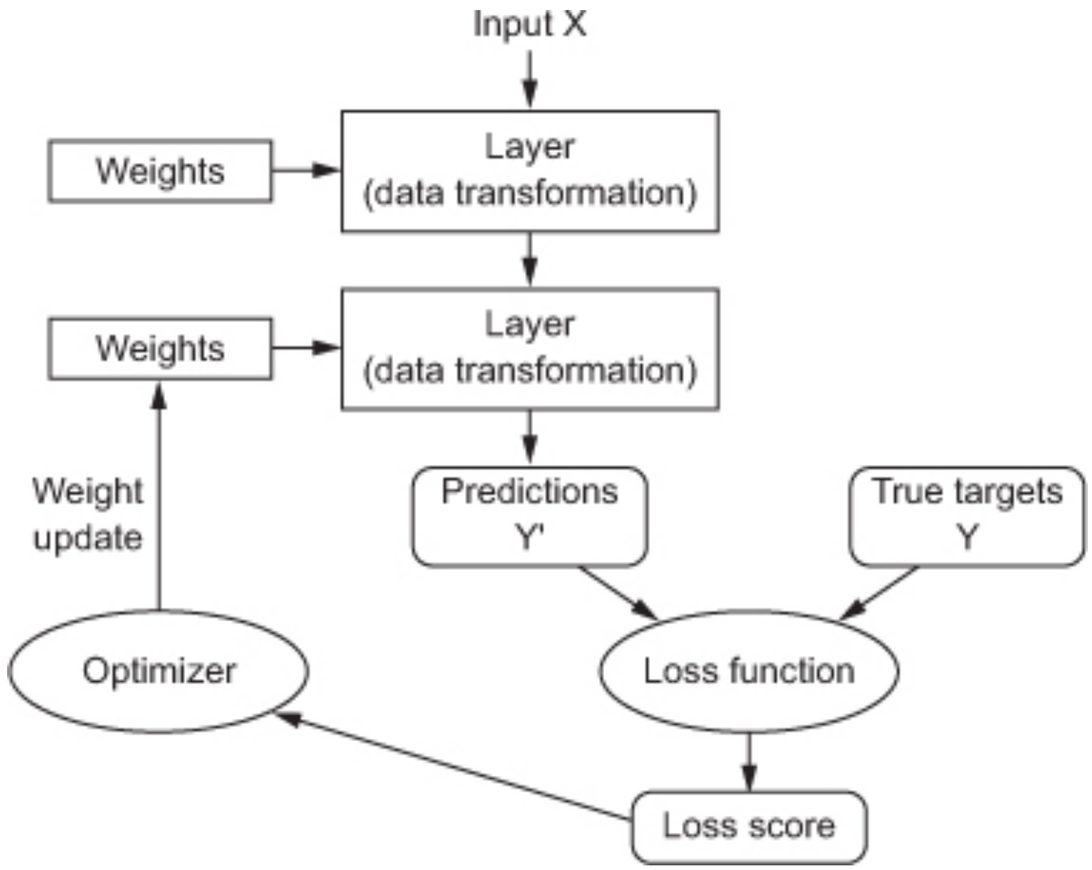
A neural network is parameterized by its weights:



CV & AI vs. ML vs. DL (6)

- **DL:** How does it work?
  - the **loss** / cost / error / objective function **measures** the **quality** of the network's output by computing a **distance score** between **predictions** of the network and the **true** target
  - the **optimizer** implements the Gradient Decent / Backpropagation algorithm where the **loss score** is used as a feedback signal to **adjust** the value of the **weights** a little, in a direction that will **lower** the loss score for the current **example**
  - **initially**: weights are assigned **random** values
  - training loop: weights are **adjusted** a little in the correct direction and the loss score decreases
  - finally: trained network (outputs close to targets)
  - a simple mechanism that, once scaled, ends up looking like magic

A loss function measures the quality of the network's output
The loss score is used as a feedback signal to adjust the weights



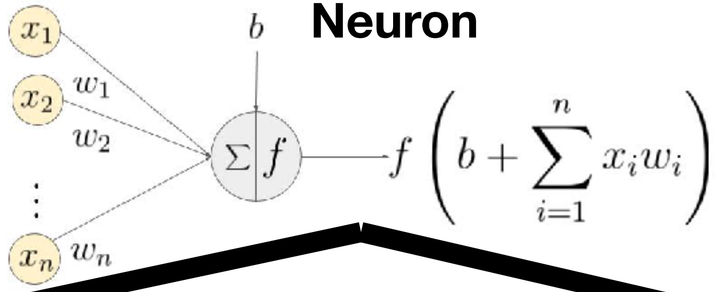
CV & AI vs. ML vs. DL (7)

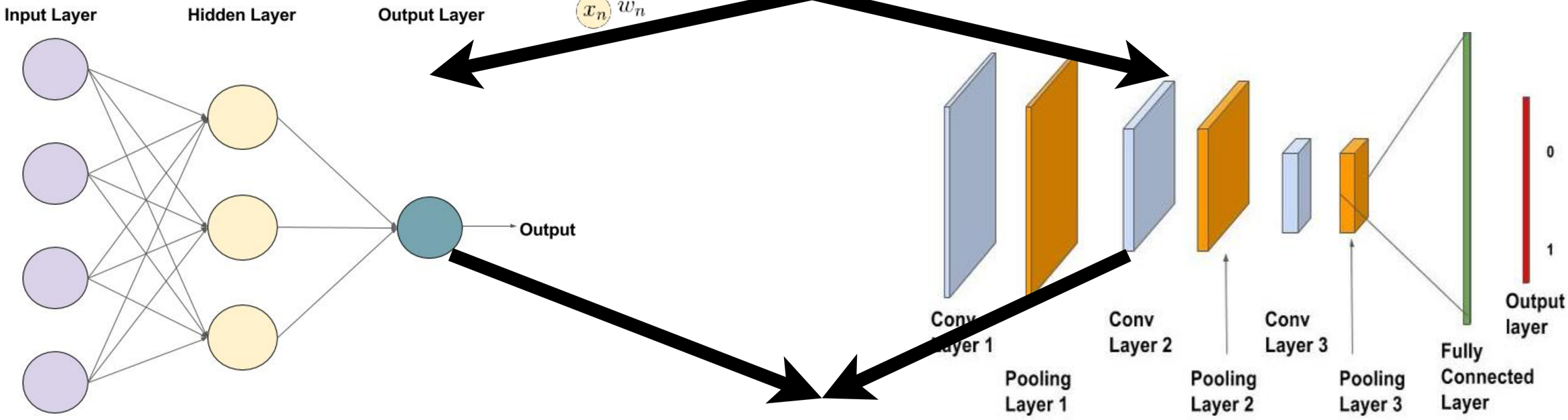
- **DL:** Achievements so far:
  - have **revolutionized** the field of CV (since early 2010s)
  - remarkable results on **perceptual** problems such as **seeing and hearing**, i.e. problems involving skills that seem **natural** and intuitive to **humans** but have long been difficult / **impossible** to **computers** / robots / machines.
  - some breakthroughs:
    - Near-human-level image classification
    - Near-human-level speech recognition
    - Near-human-level autonomous driving
    - Superhuman Go playing (AlfaGo Zero)..
  - in the future: deep learning may assist humans in science, software development, and more

CV & AI vs. ML vs. DL (7)

- **DL:** Short-time hype:
  - remarkable achievements in recent years
    - autonomous cars are within reach
  - expectations (next decade) higher than what will likely be possible
    - believable dialogue systems
    - human-level machine translation across arbitrary languages
    - human-level natural-language understanding
    - human-level general intelligence

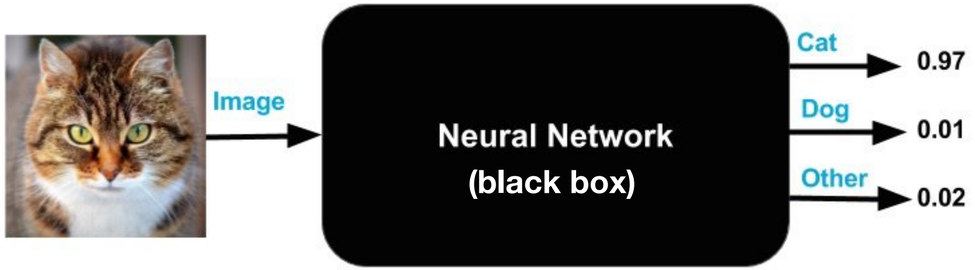
Big picture / overview: Predictions





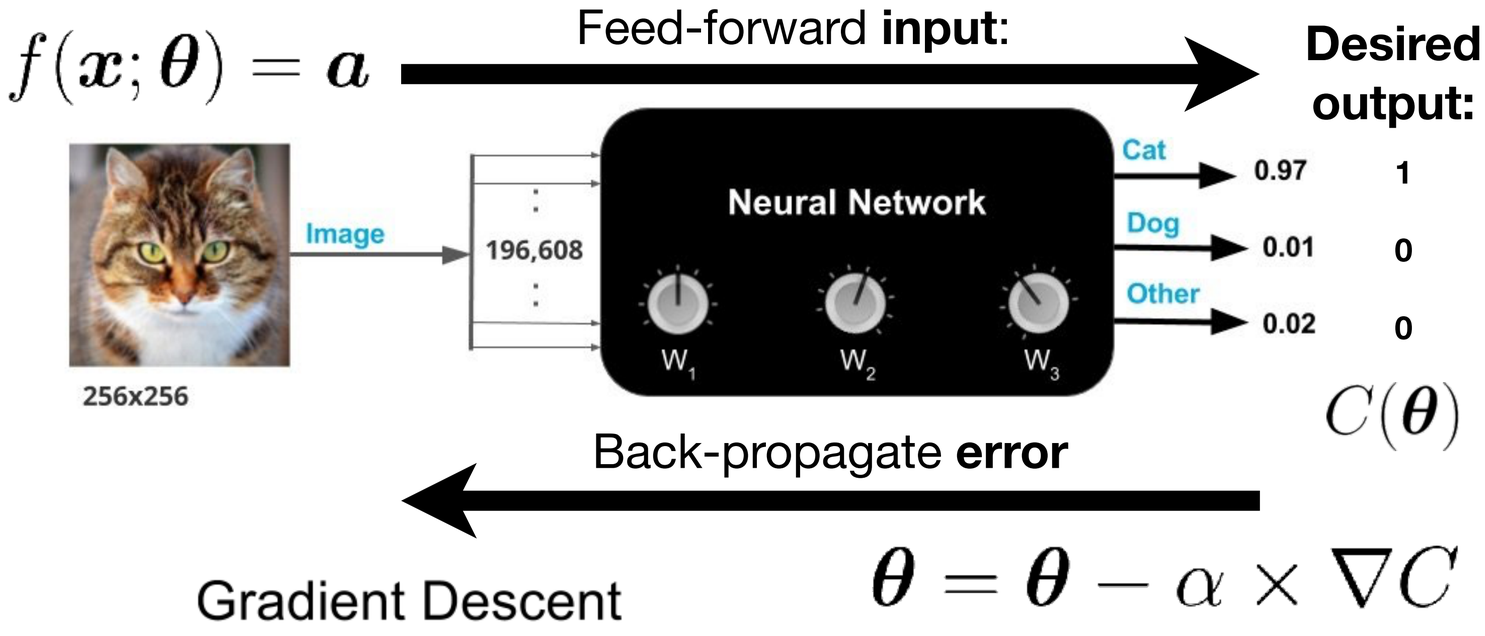
Fully-Connected Net

Convolutional NN



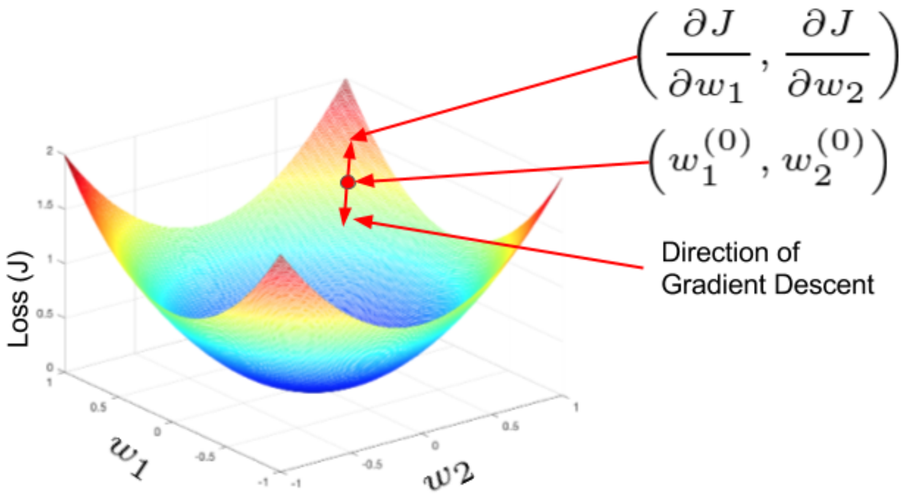
Big picture / overview: Training

- GD/BP Alg: **Iterate** until ok:
  - Feed-forward **input**
  - **Compare** current output with desired output (**loss**)
  - Back-propagate **error**
  - **Update** wights and bias



Gradient Descent

$$\theta = \theta - \alpha \times \nabla C$$

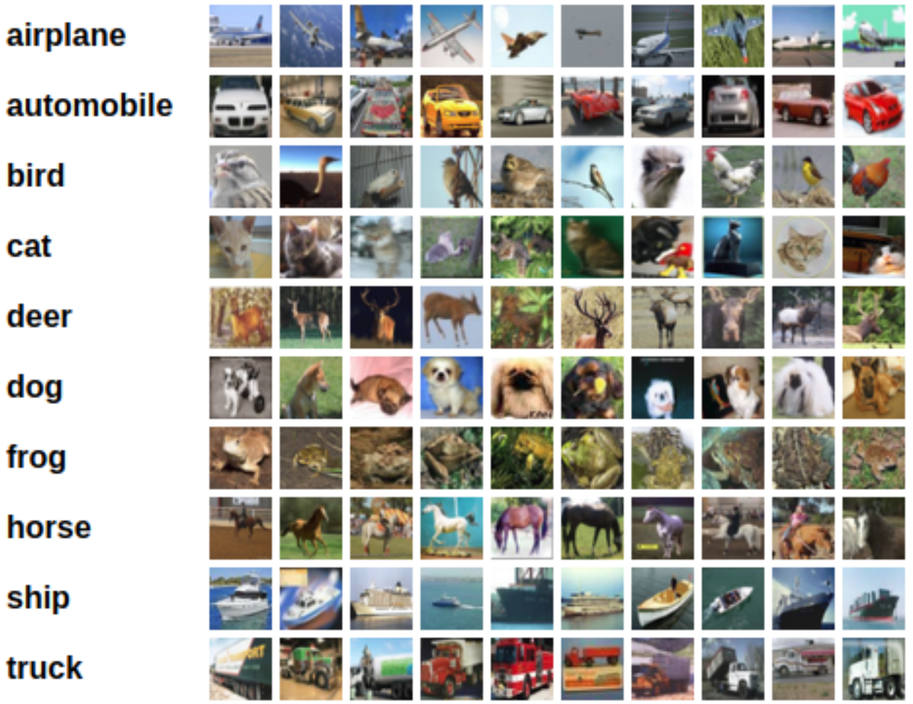


Datasets

MNIST

Training: 60k, Test: 10k, Classes: 10, Size: 28x28

CIFAR10



Training: 50k, Test: 10k, Classes: 10, Size: 32x32

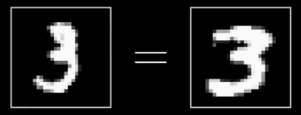
FF FCNN

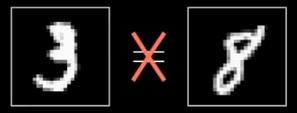
Part 1: Intro

3Blue1Brown: NNs (Nielsen)

(have a look or two on these videos and try get the message)

Structure





0.67 Neuron

Thing that holds a number

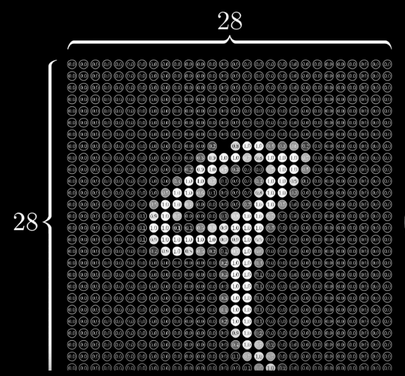
Neuron

Function

$$f(a_0, \ldots, a_{783}) = \begin{bmatrix} y_0 \\ \vdots \\ y_9 \end{bmatrix}$$

Network

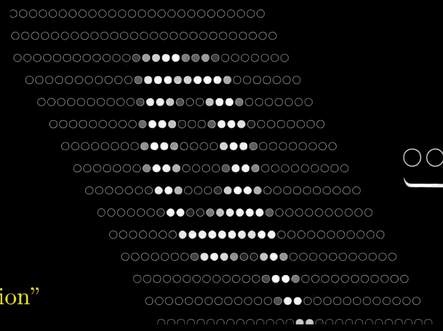
Function

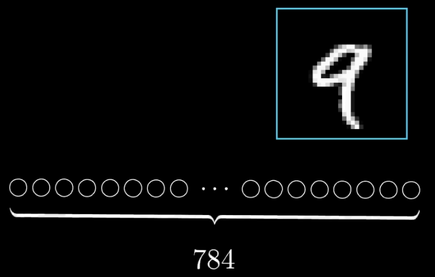


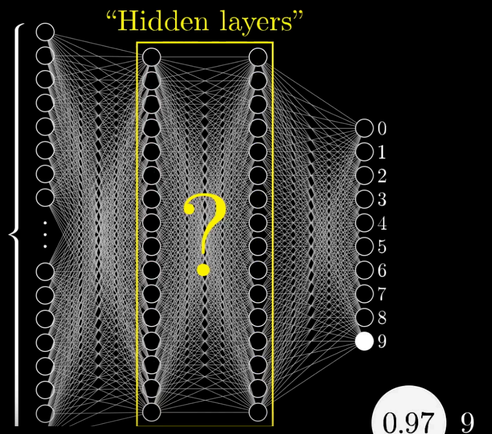
$28 \times 28 = 784$

0.20 “Activation”

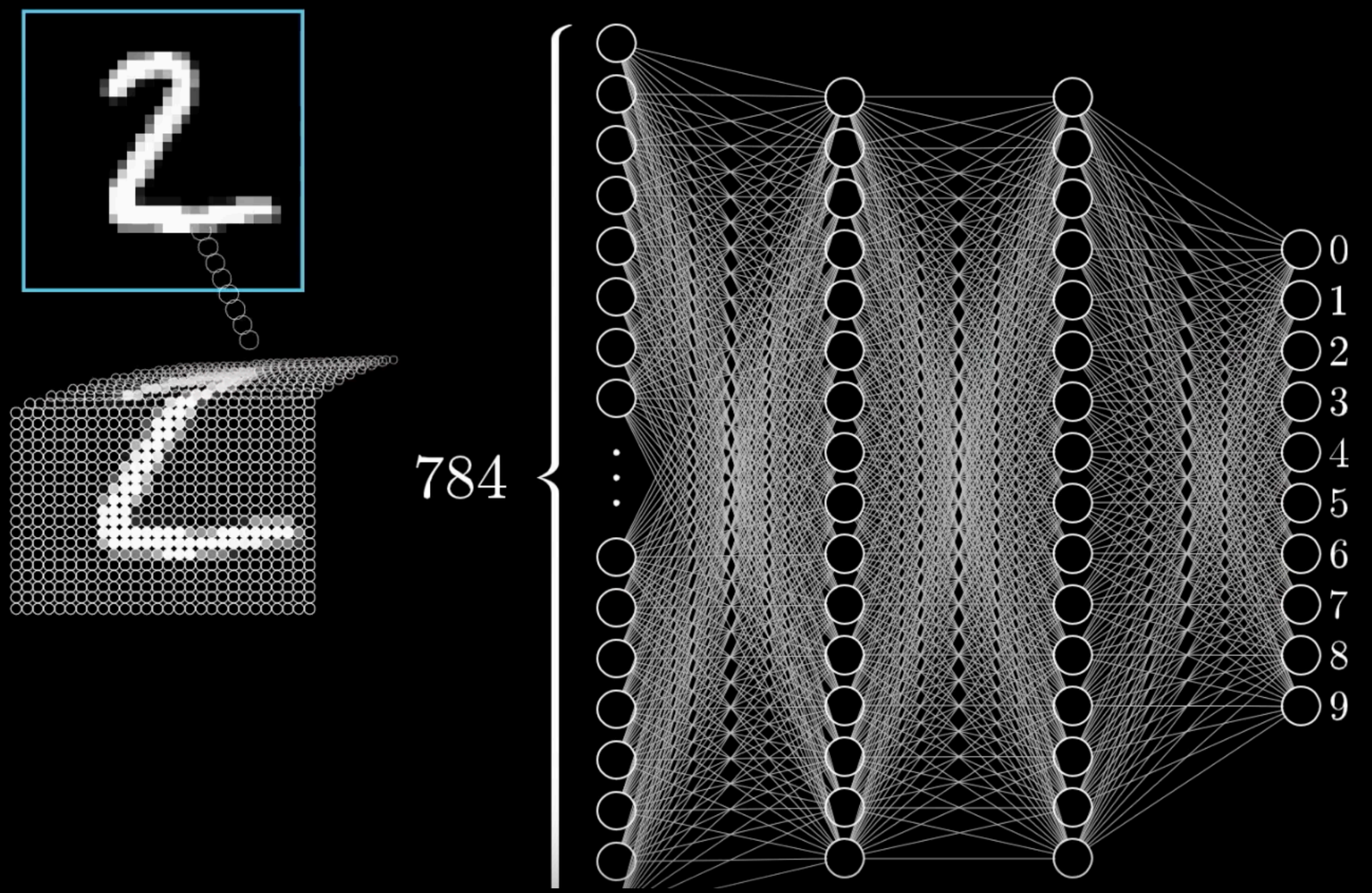
1.00 “Activation”



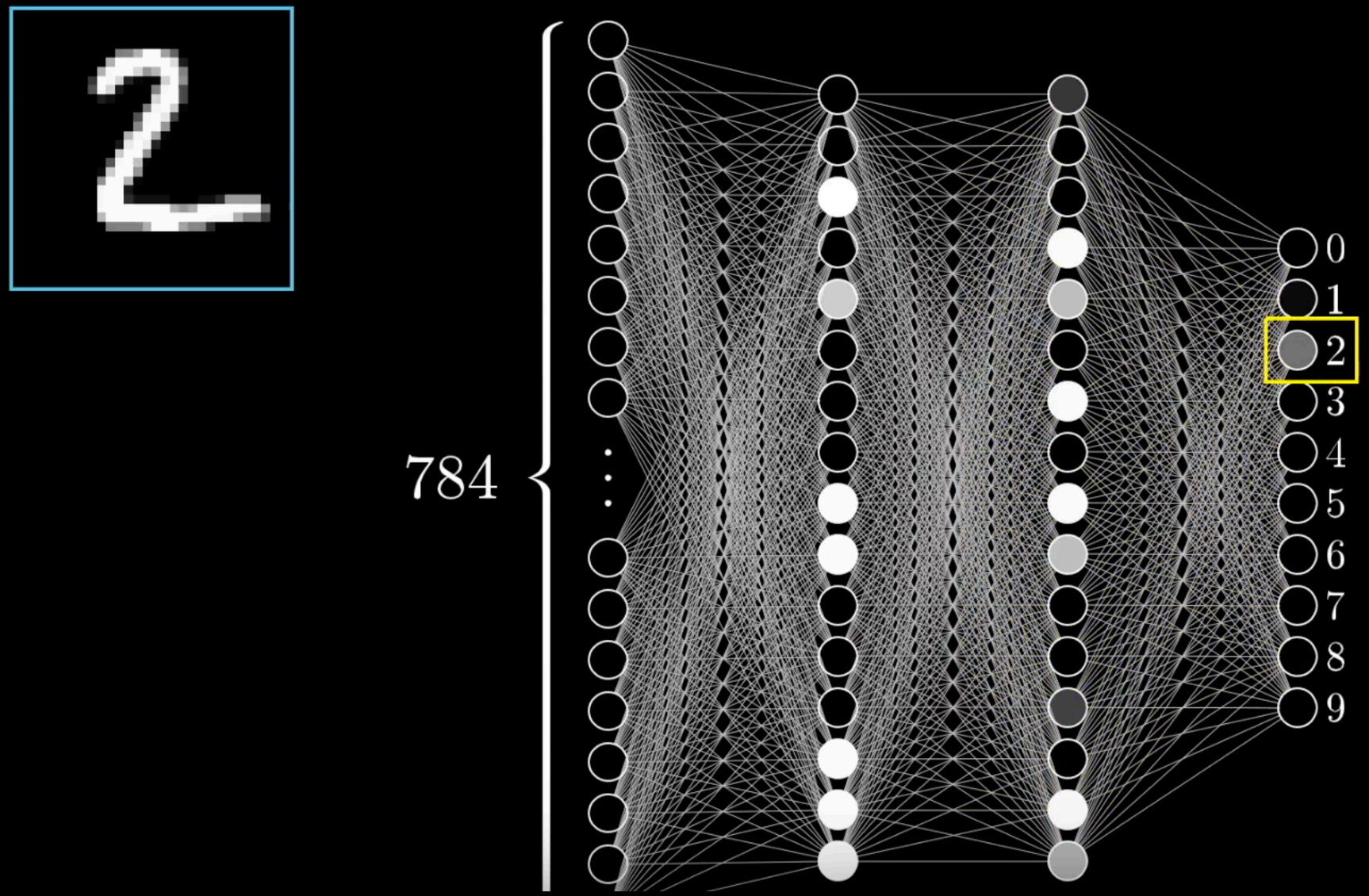




0.97 9

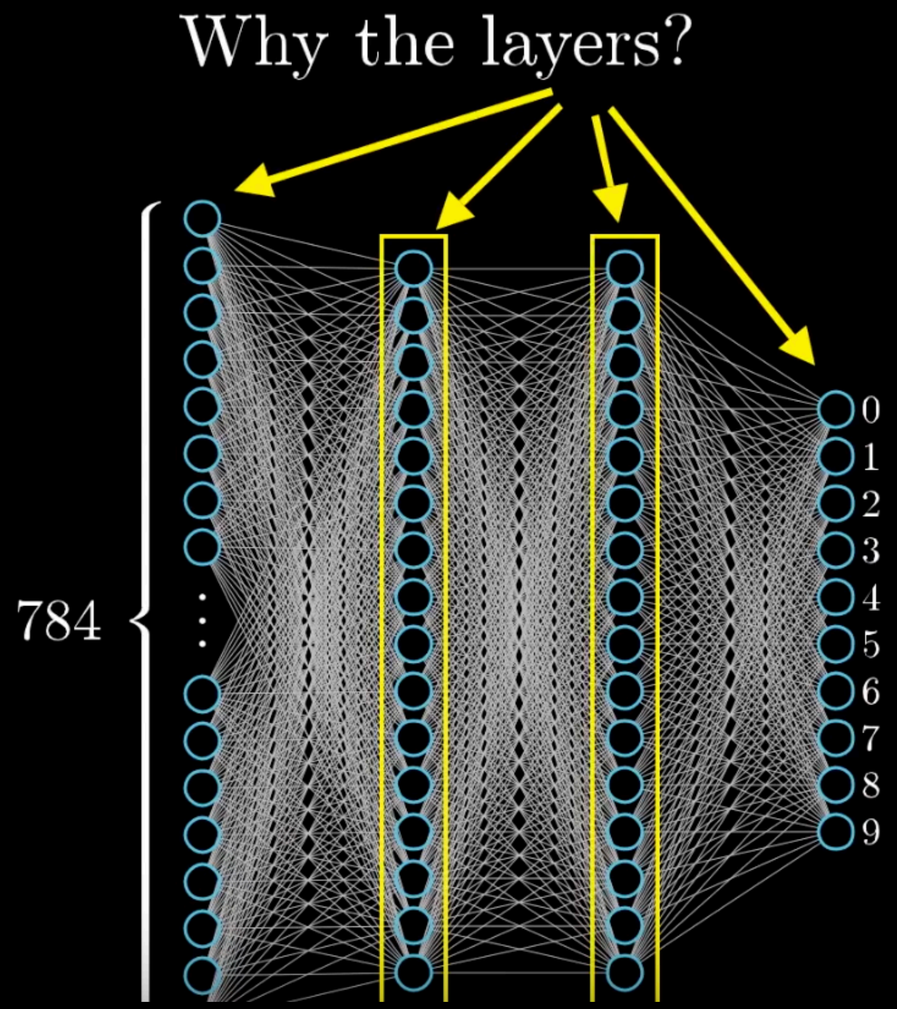


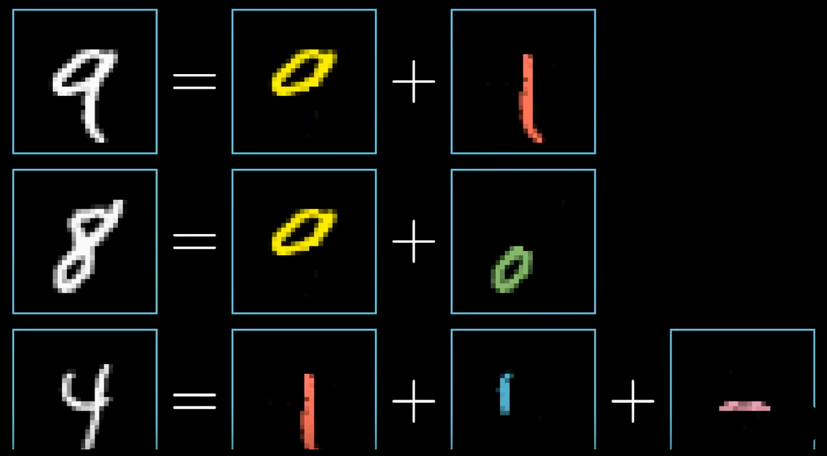
MNIST (hello world)
Feed-forward info
Input: 28x28 -> 784

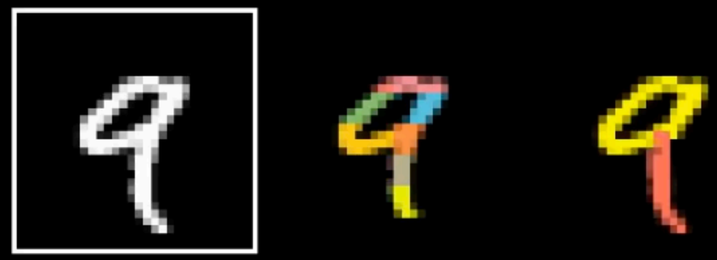


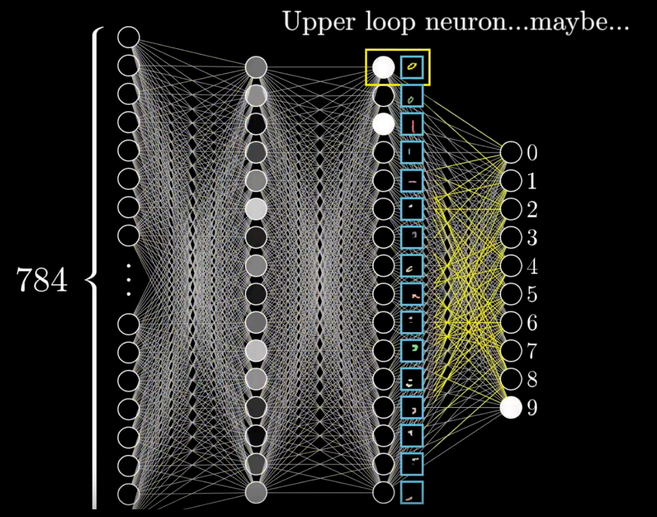
- Hidden layers
  (two, 16 neurons each)
- Ouput layer:
  (10 neurons, which digit)
- Info from layer to layer
  (activating diff. neurons)

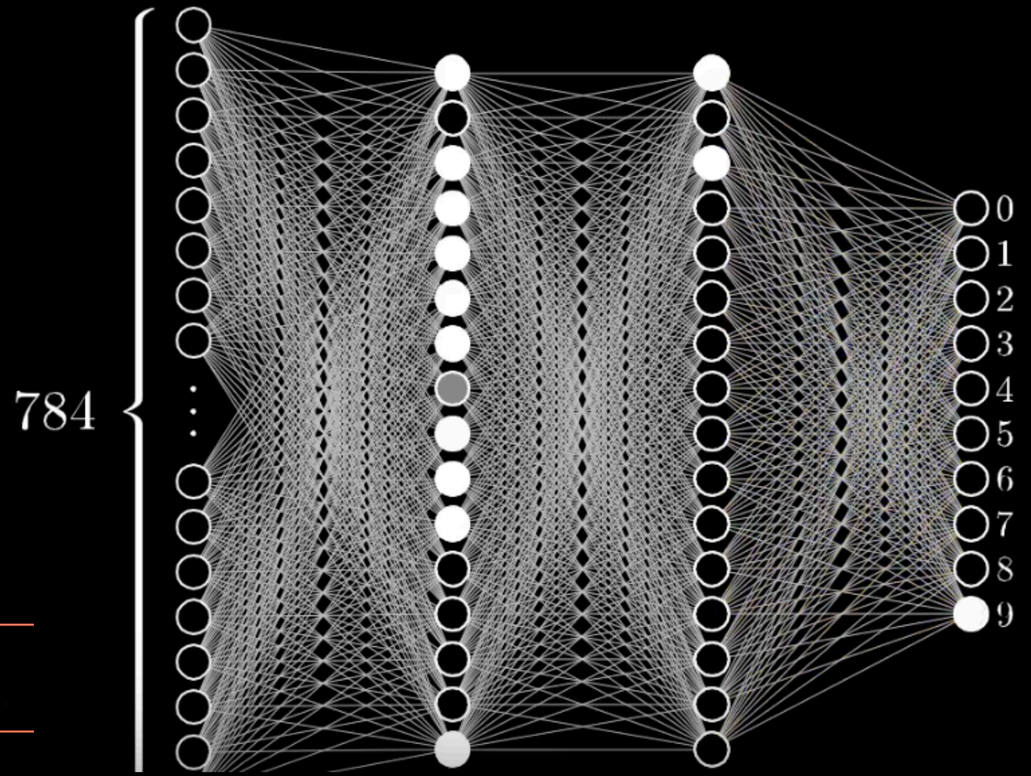
Layers

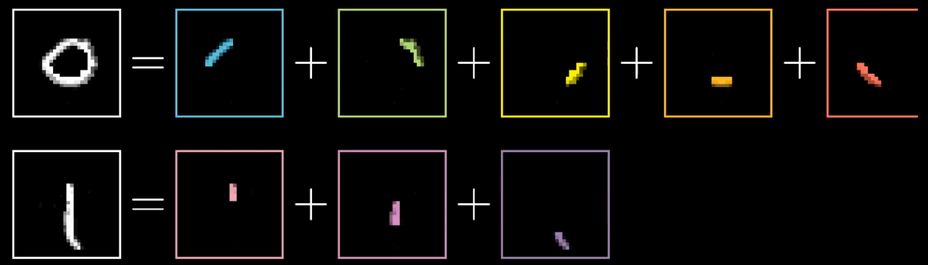




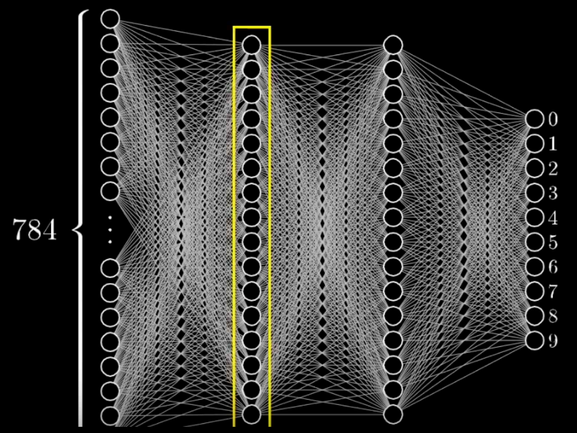




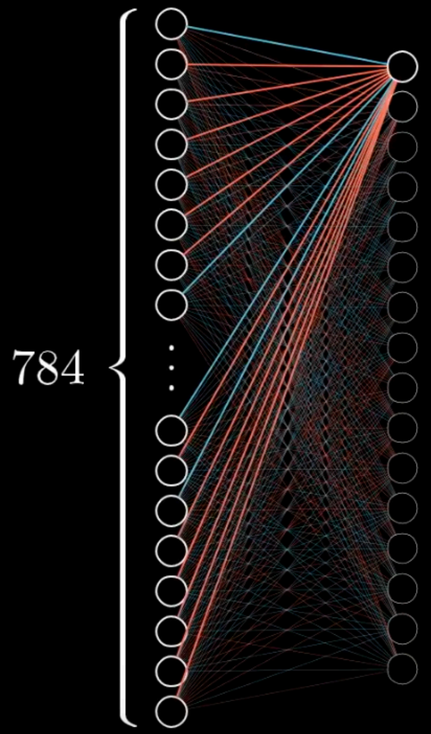


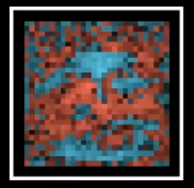


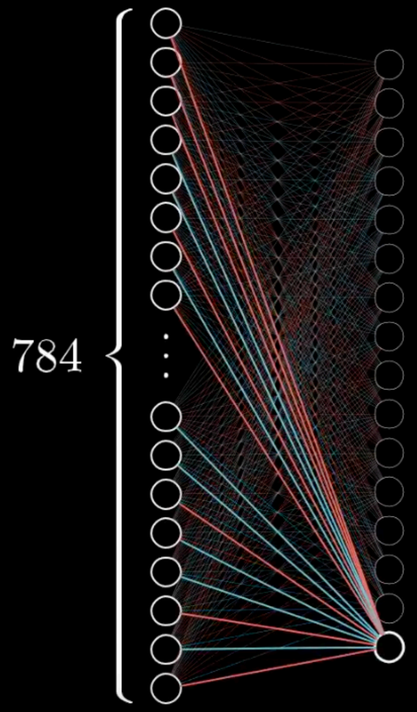
“Little edge” layer?



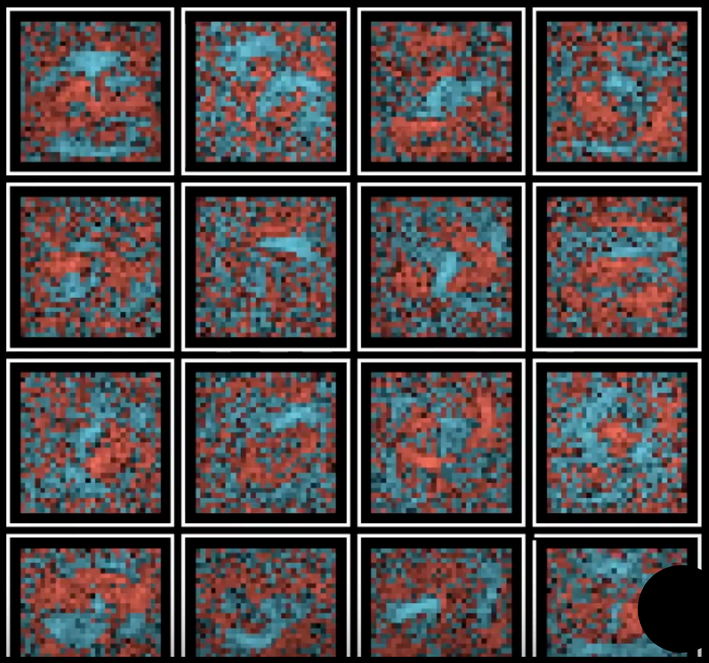
What second layer
neurons look for



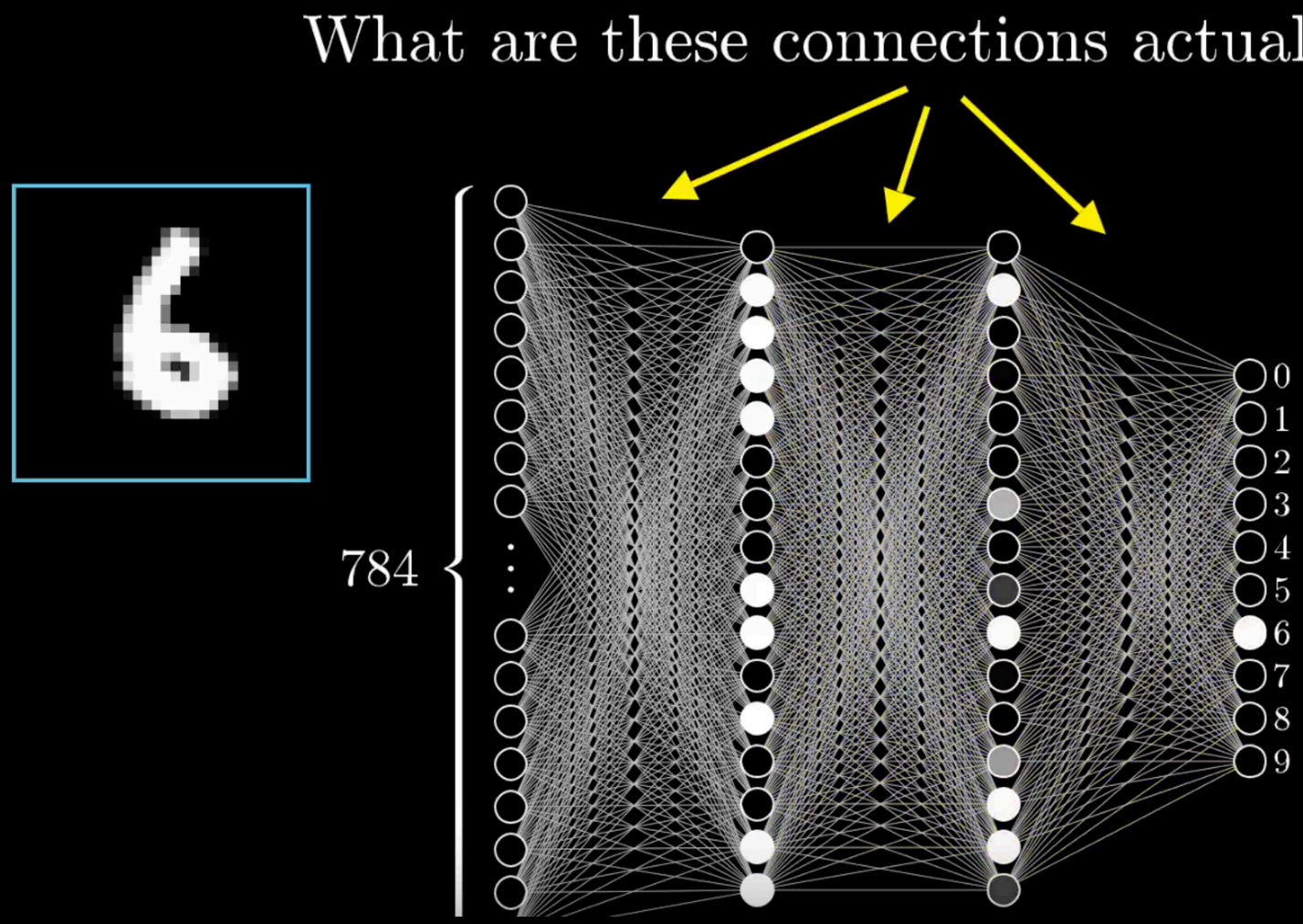


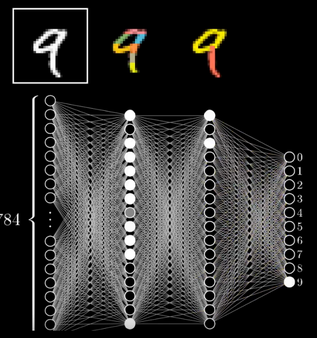


What second layer
neurons look for



Connections





Sigmoid

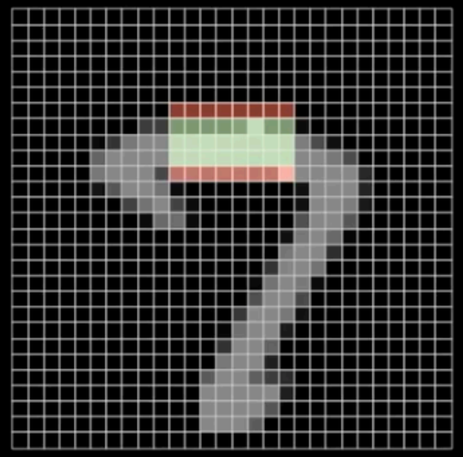
How positive is this?

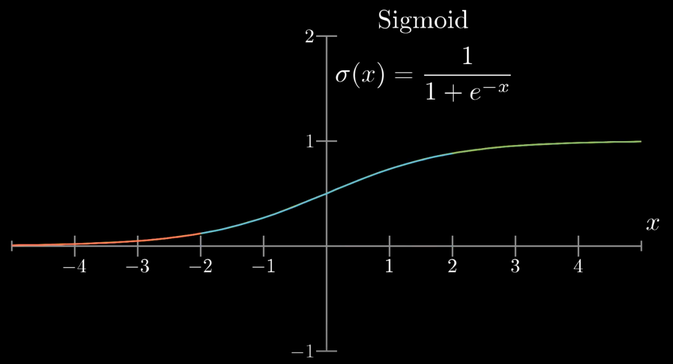
$$\sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 + \cdots + w_n a_n \boxed{-10})$$

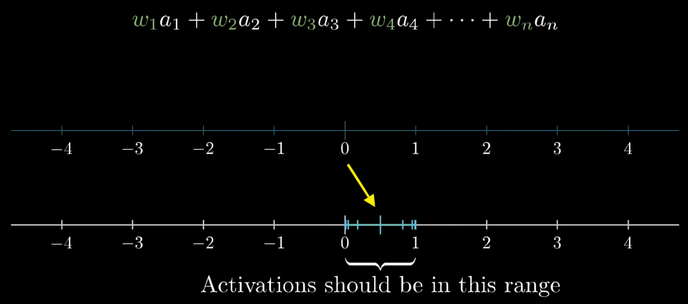
"bias"

784

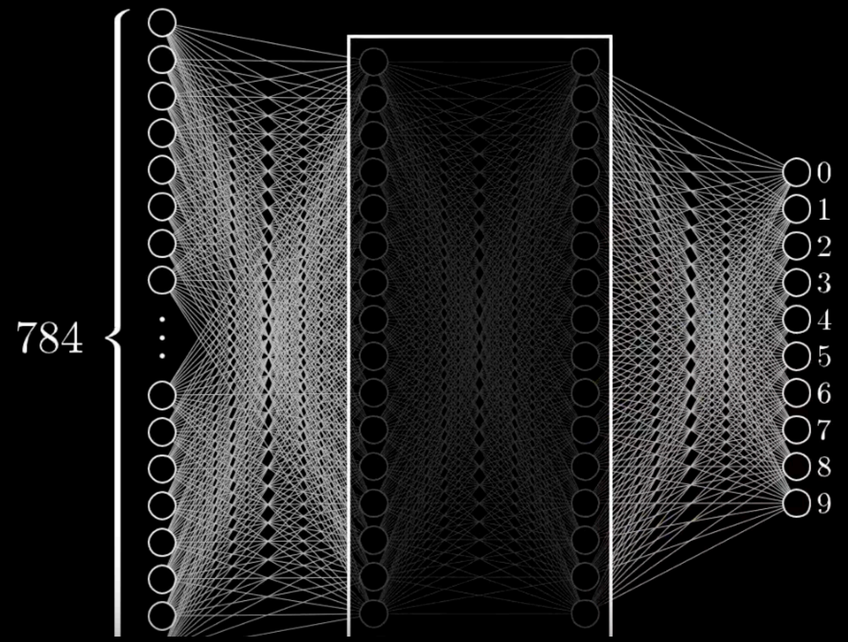
Only activate meaningfully when weighted sum > 10



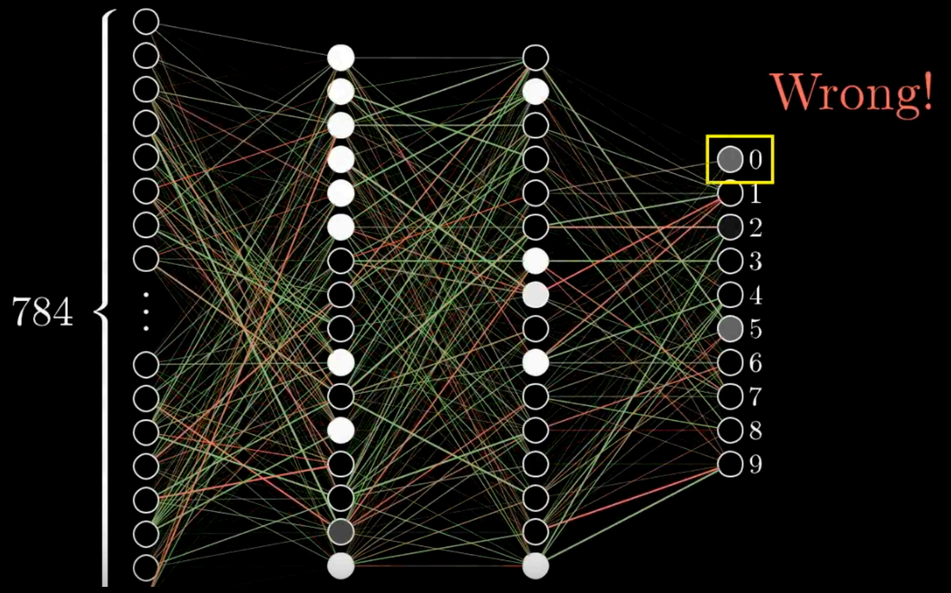




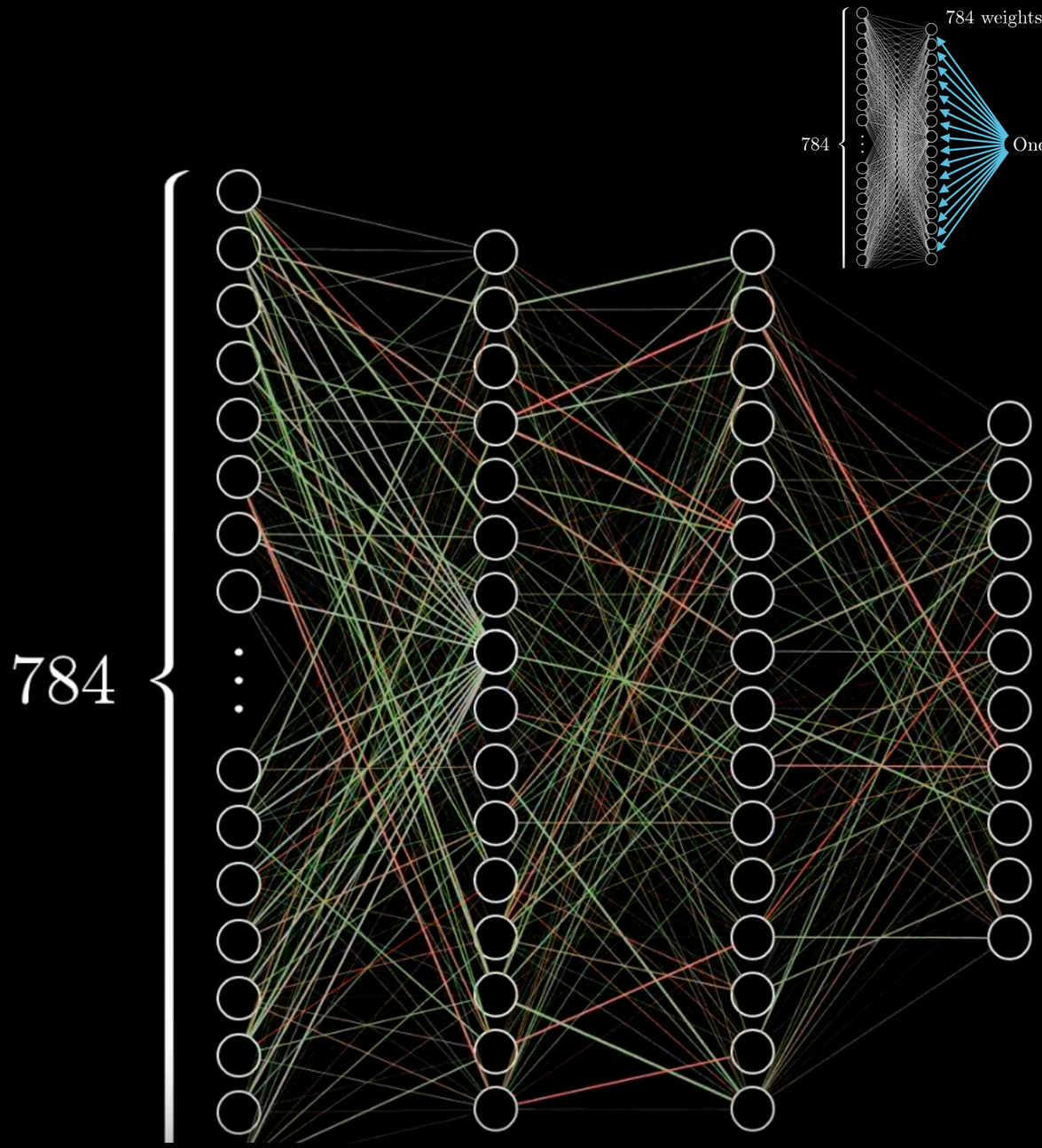
...rather than treating this as a black box



What weights are used here?
What are they doing?



Wrong!



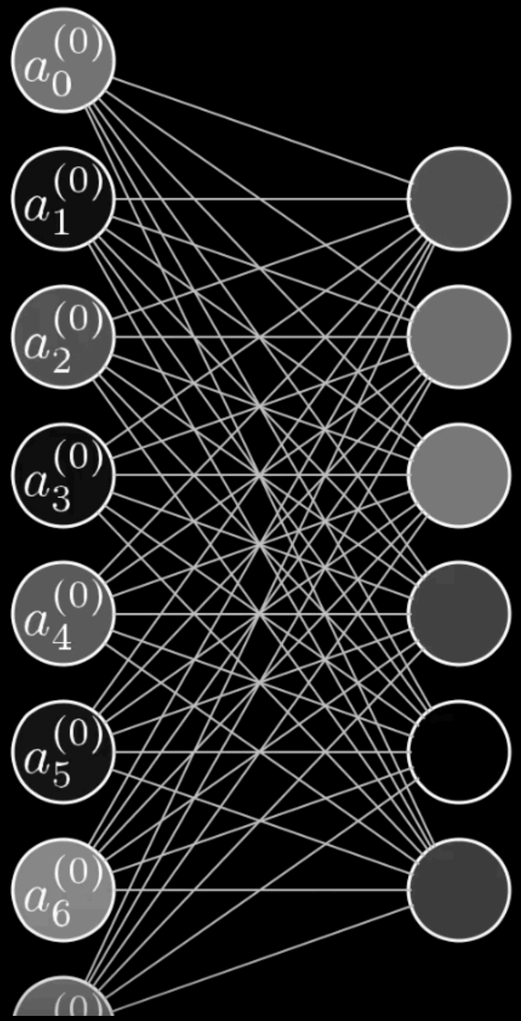
$$784 \times 16 + 16 \times 16 + 16 \times 10 \text{ weights}$$

$$16 + 16 + 10 \text{ biases}$$

$$13{,}002$$

Learning $\rightarrow$ Finding the right weights and biases

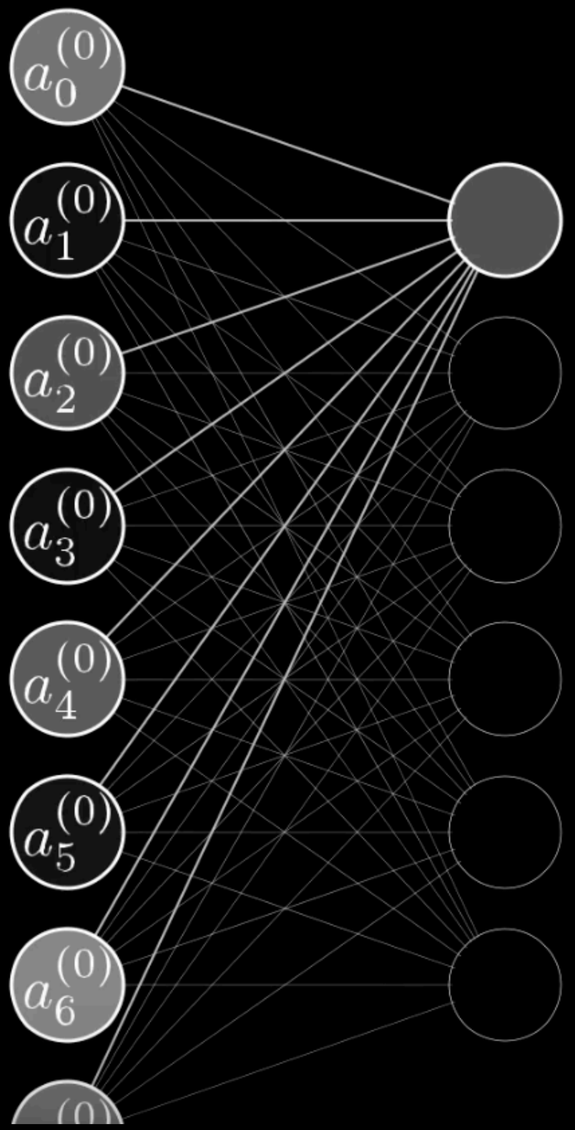
Matrix-form (lin.alg.)



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

```
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
```

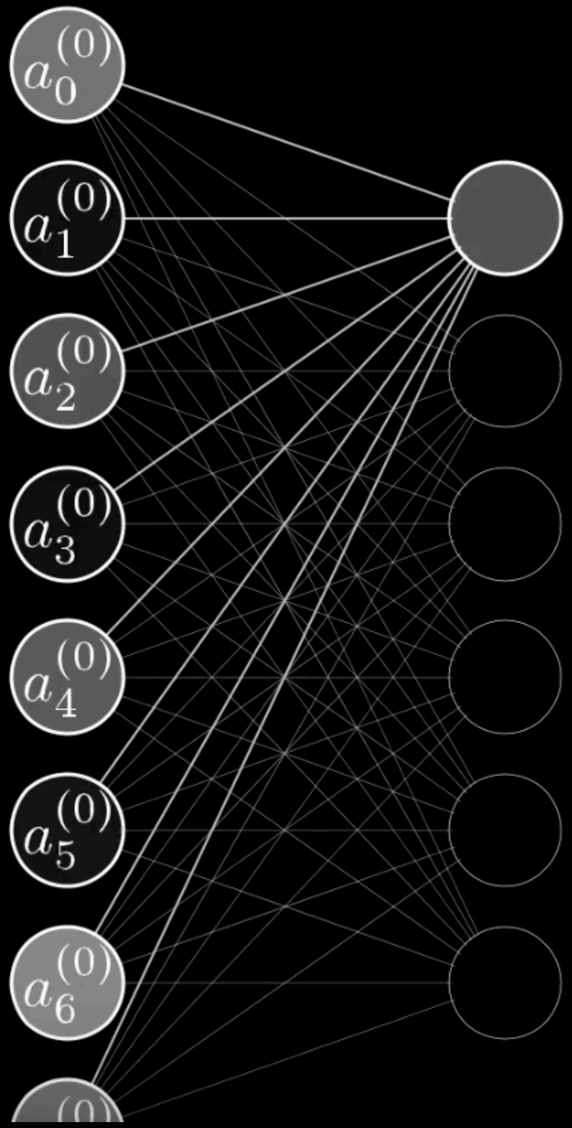


Sigmoid

$$a_0^{(1)} = \sigma\left(\underbrace{w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)}}\ + b_0\right)$$

Bias

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} = \begin{bmatrix} ? \end{bmatrix}$$

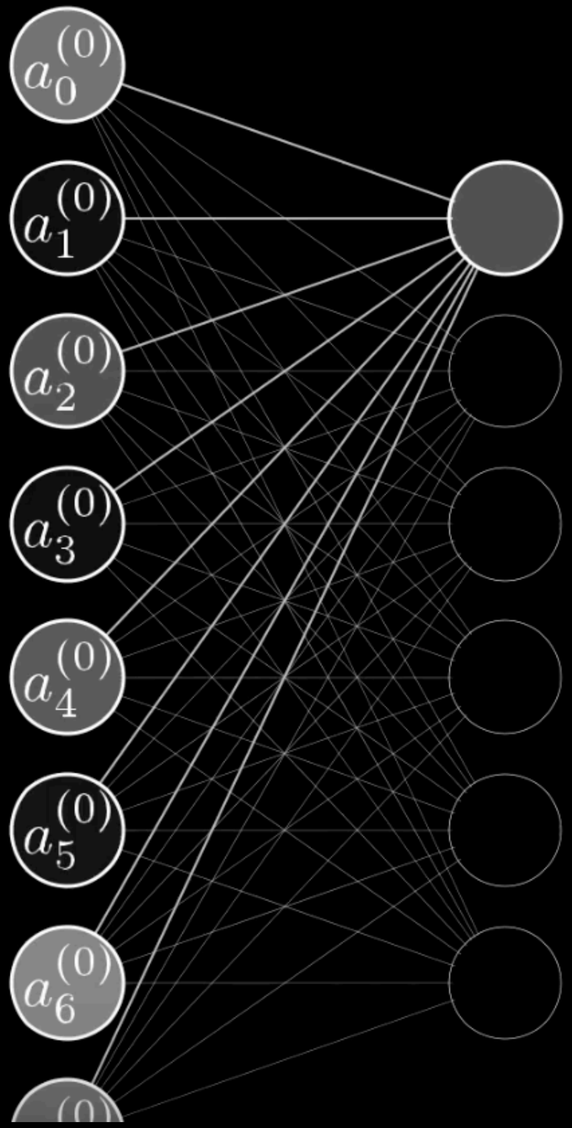


Sigmoid

$$a_0^{(1)} = \sigma\left(w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)} + b_0 \right)$$

Bias

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

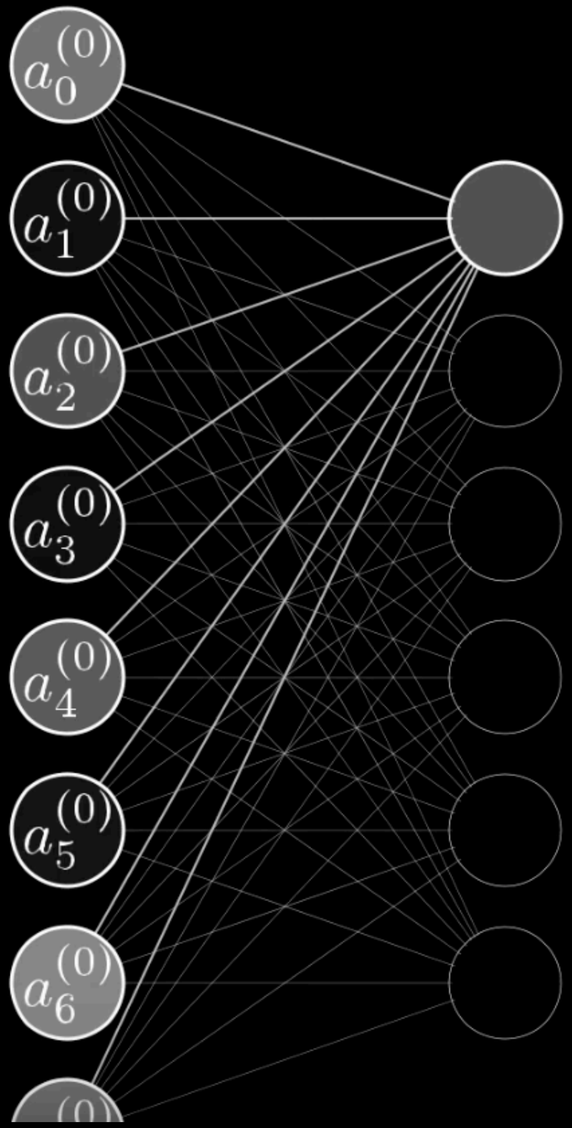


Sigmoid

$$a_0^{(1)} = \sigma\left(w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)} + b_0 \right)$$

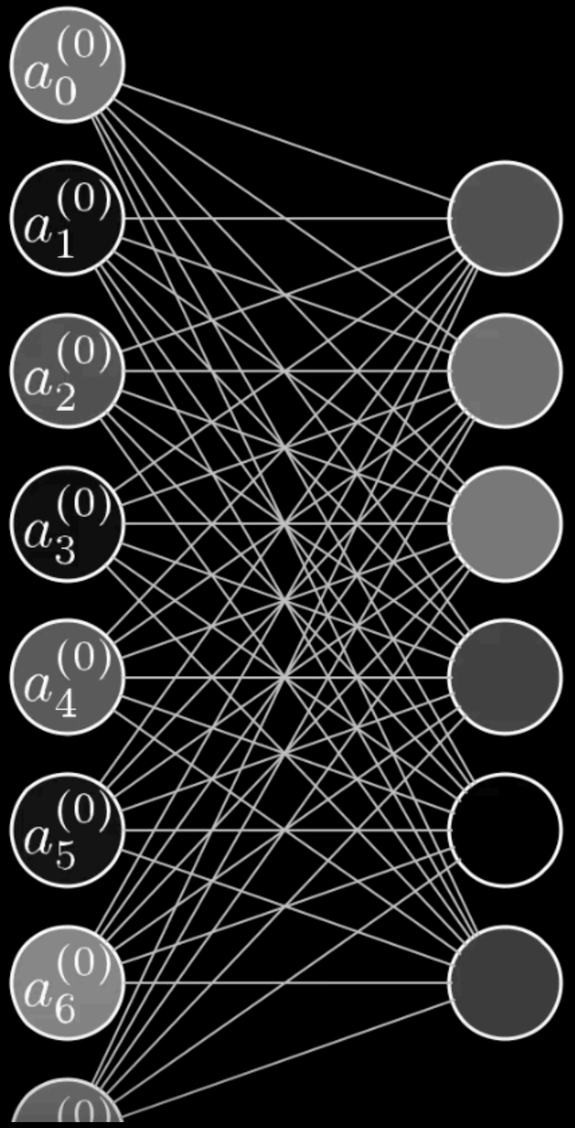
Bias

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



$$\sigma\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} \sigma(x) \\ \sigma(y) \\ \sigma(z) \end{bmatrix}$$

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

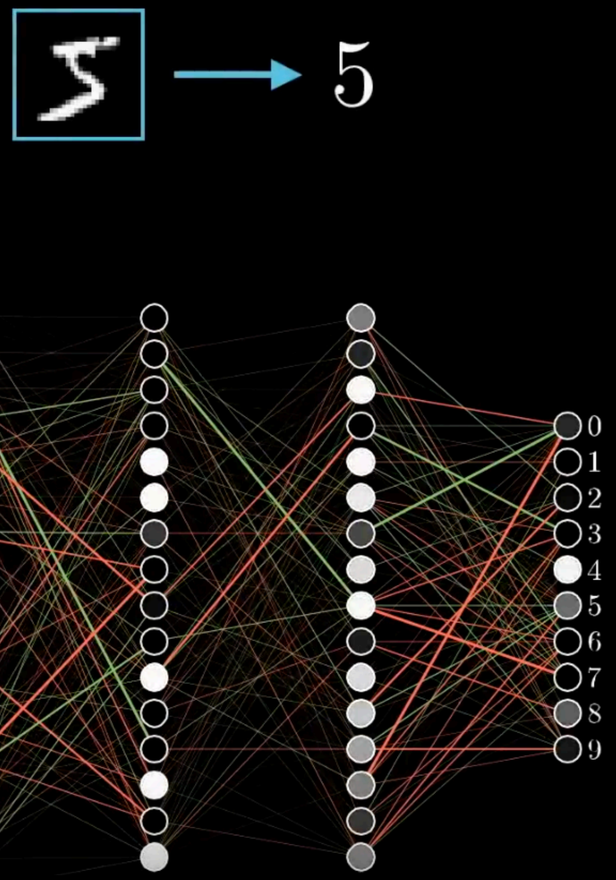


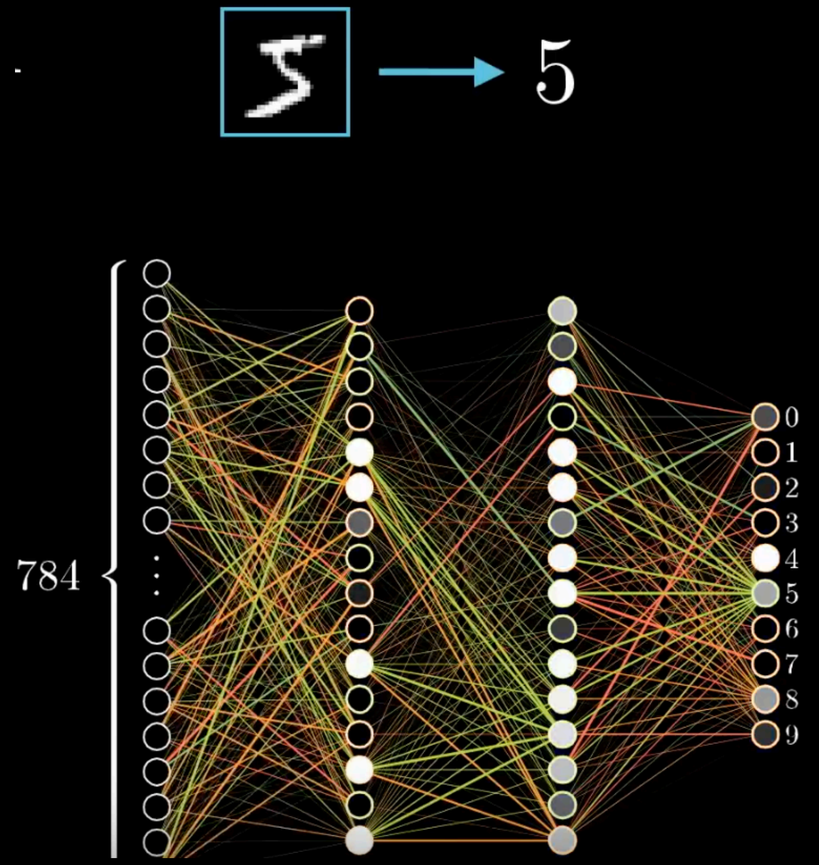
$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$$

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

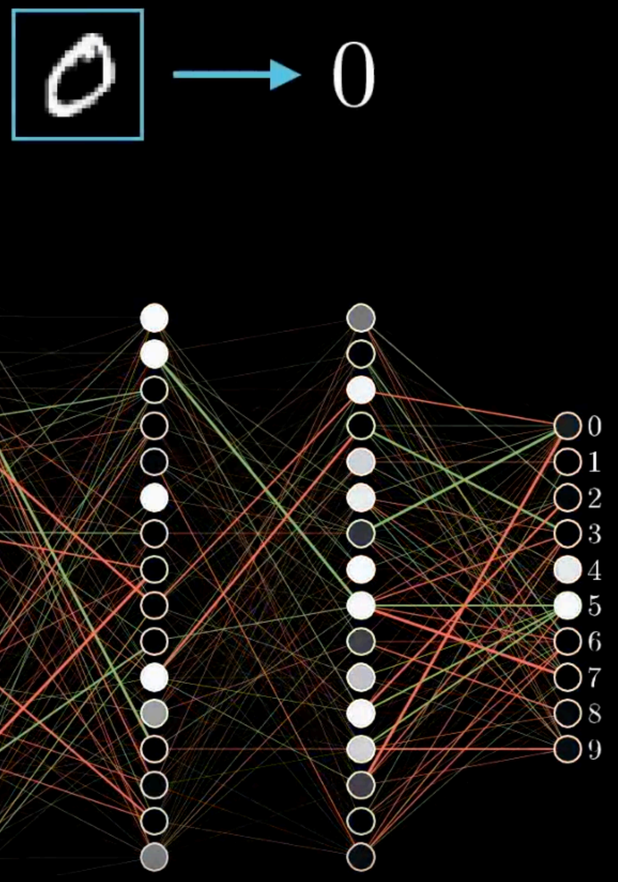
Training

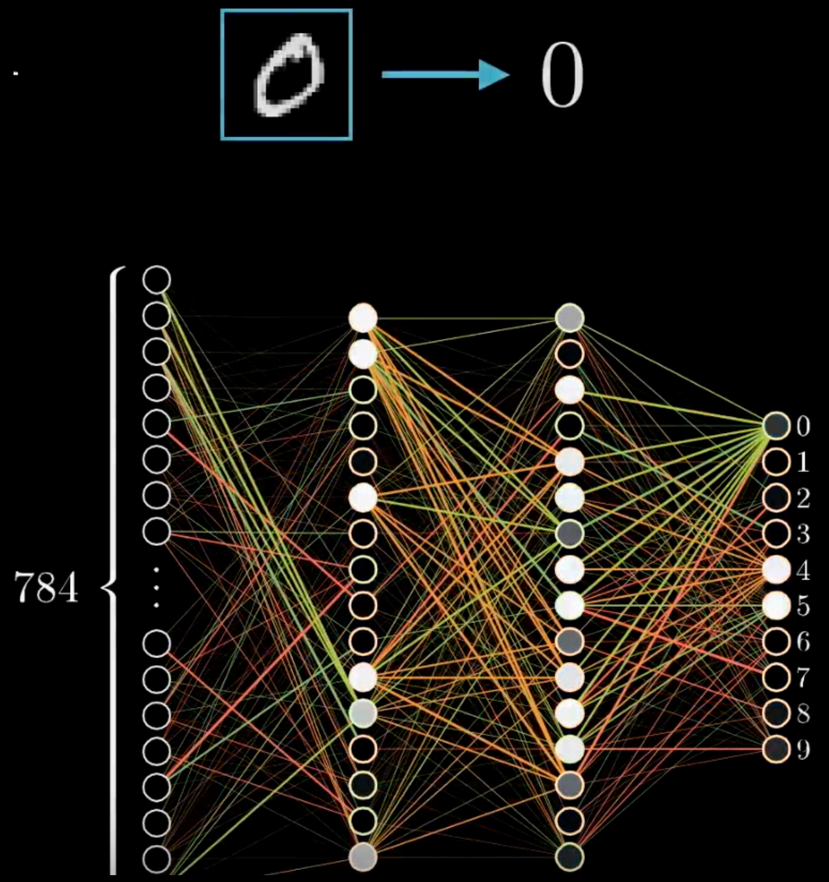
Training in progress. . .



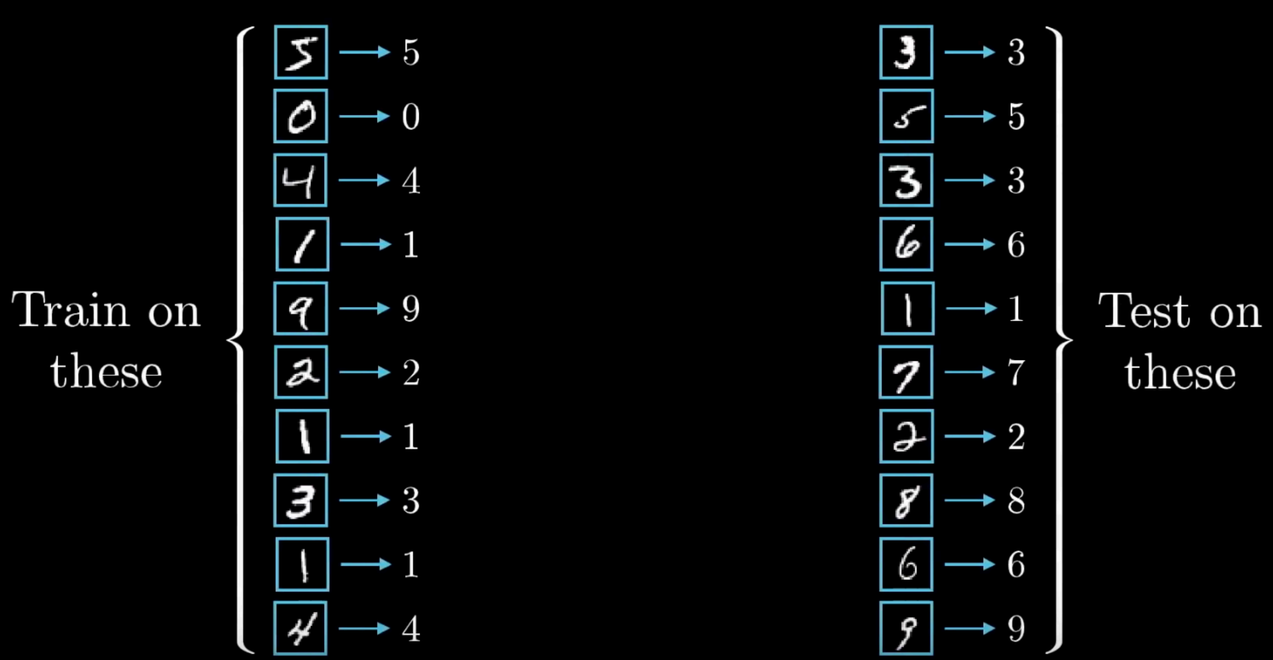


Training in progress...





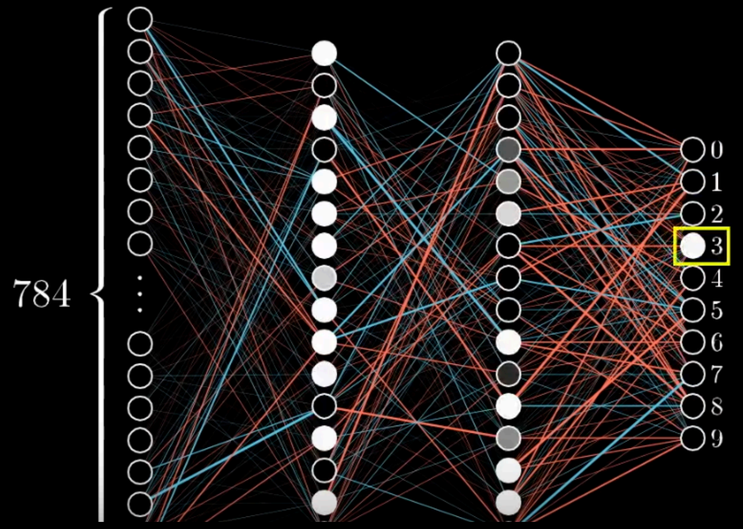
Testing


Train on these
5 → 5
0 → 0
4 → 4
1 → 1
9 → 9
2 → 2
1 → 1
3 → 3
1 → 1
4 → 4
3 → 3
5 → 5
3 → 3
6 → 6
1 → 1
7 → 7
2 → 2
8 → 8
6 → 6
9 → 9
Test on these
(8, 8) (2, 2) (9, 9) (4, 4) (4, 4) (6, 6) (0, 0) (9, 9)
(7, 7) (0, 0) (3, 3) (2, 2) (2, 2) (2, 2) (1, 1) (5, 5)
(6, 6) (1, 1) (9, 2) (3, 3) (1, 2) (3, 3) (5, 5) (9, 9)
(1, 1) (7, 7) (6, 6) (2, 2) (8, 8) (2, 2) (1, 1) (5, 5)
(4, 4) (5, 5) (4, 4) (9, 9) (7, 7) (8, 8) (3, 3) (2, 2)
(9, 9) (1, 1) (8, 8) (0, 0) (5, 5) (1, 1) (0, 0) (3, 3)
(7, 7) (0, 0) (0, 0) (3, 3) (1, 7) (0, 0) (7, 7) (3, 3)
(0, 0) (4, 4) (3, 3) (5, 5) (2, 2) (7, 7) (6, 6) (7, 7)
(9, 9) (8, 8) (5, 5) (7, 7) (2, 2) (0, 0) (7, 7) (1, 1)

Testing data

Guess → 3

$$\frac{\text{Number correct}}{\text{total}} = \frac{75}{77} = 0.974$$

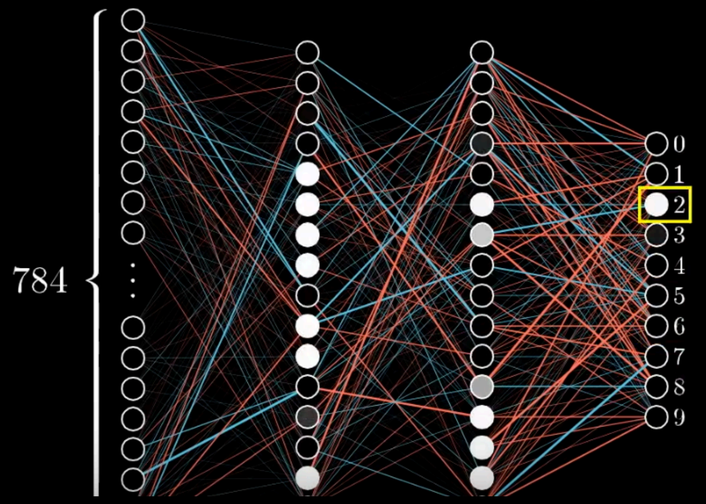


Testing data

Guess → 2

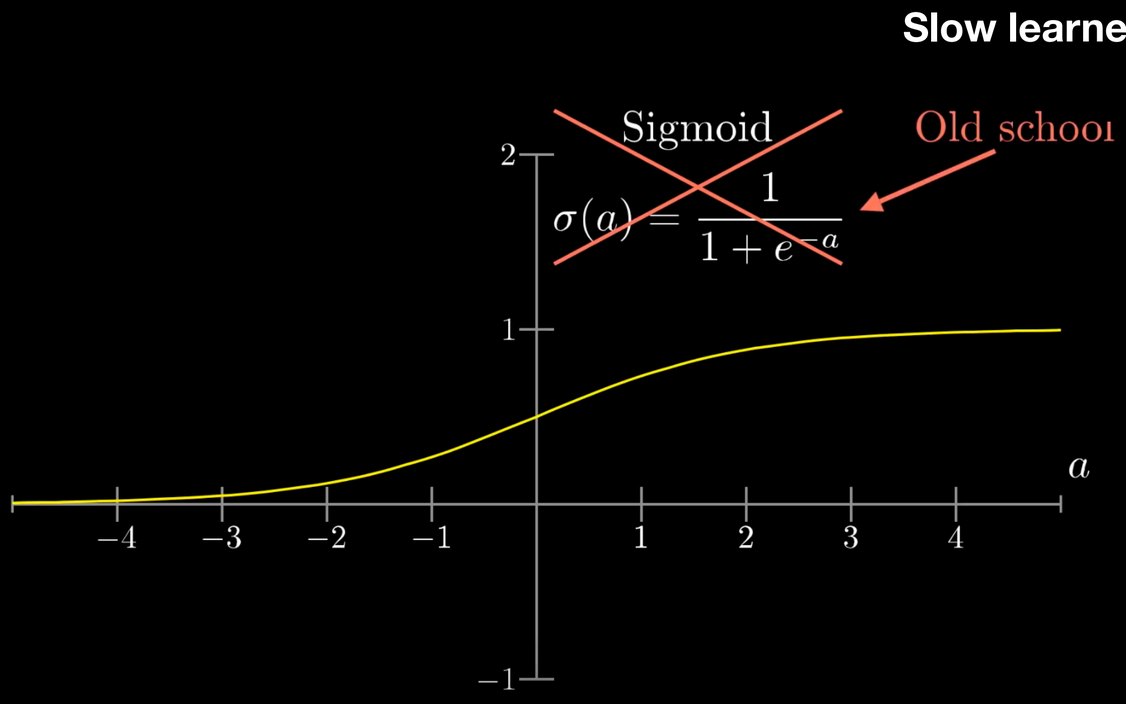
Wrong!

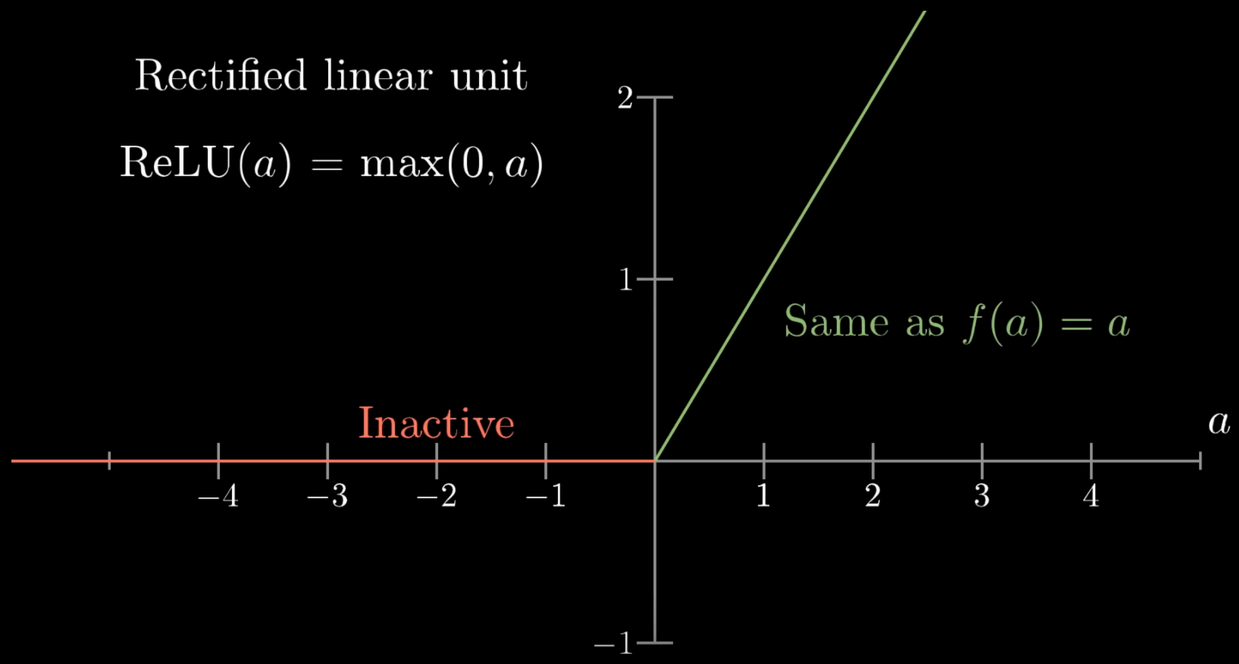
$$\frac{\text{Number correct}}{\text{total}} = \frac{62}{64} = 0.969$$



Wrong!

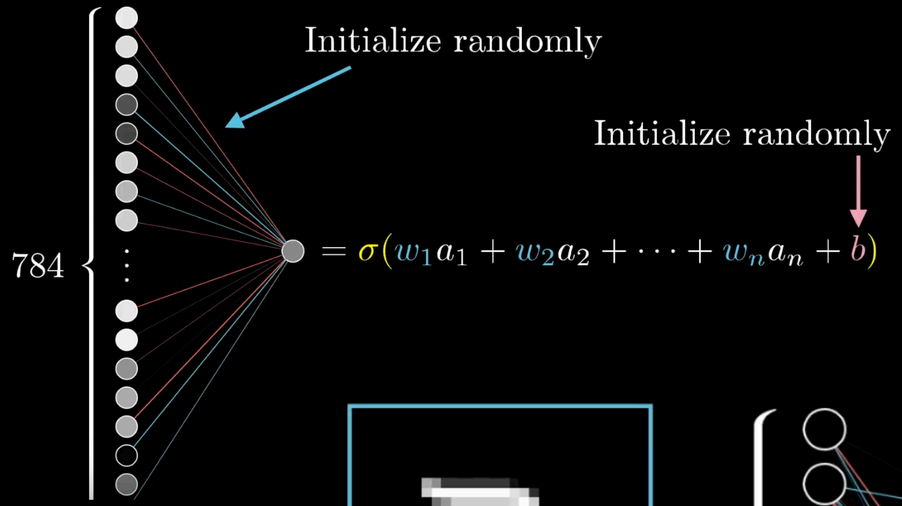
Sigmoid vs. ReLU

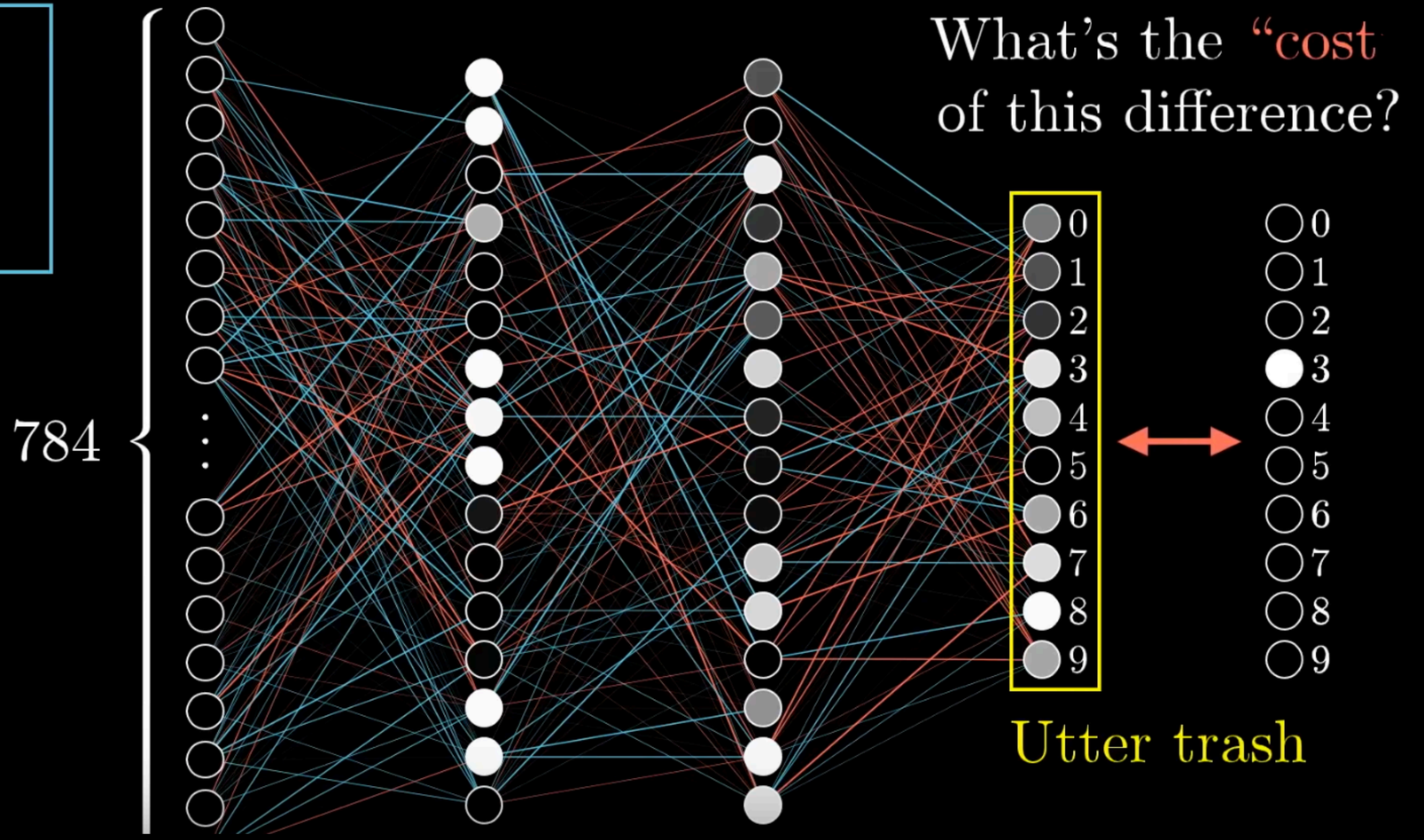




FF FCNN

Part 2: Learning





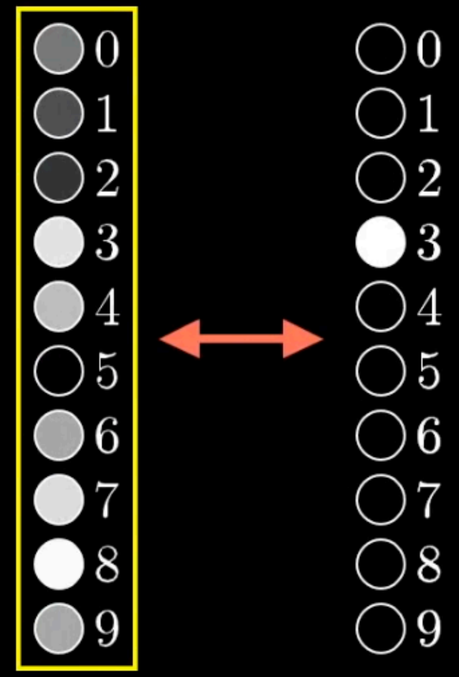
- **Cost: single example**
- **Take output NN gives**
  - **not trained = trash**
- **Along with desired output**

Cost of 

$$
\begin{aligned}
&(0.43 - 0.00)^2 + \\
&(0.28 - 0.00)^2 + \\
&(0.19 - 0.00)^2 + \\
&(0.88 - 1.00)^2 + \\
&(0.72 - 0.00)^2 + \\
&(0.01 - 0.00)^2 + \\
&(0.64 - 0.00)^2 + \\
&(0.86 - 0.00)^2 + \\
&(0.99 - 0.00)^2 + \\
&(0.63 - 0.00)^2
\end{aligned}
$$

- add squared differences
- between each component

What's the "cost of this difference?



Utter trash

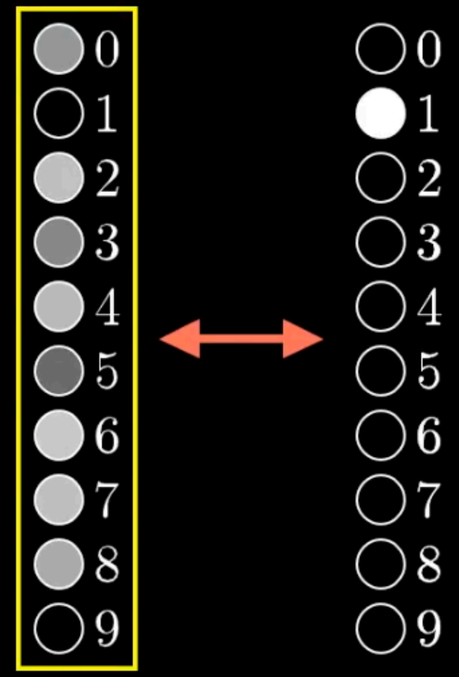
Average cost of all training data...

Cost of 

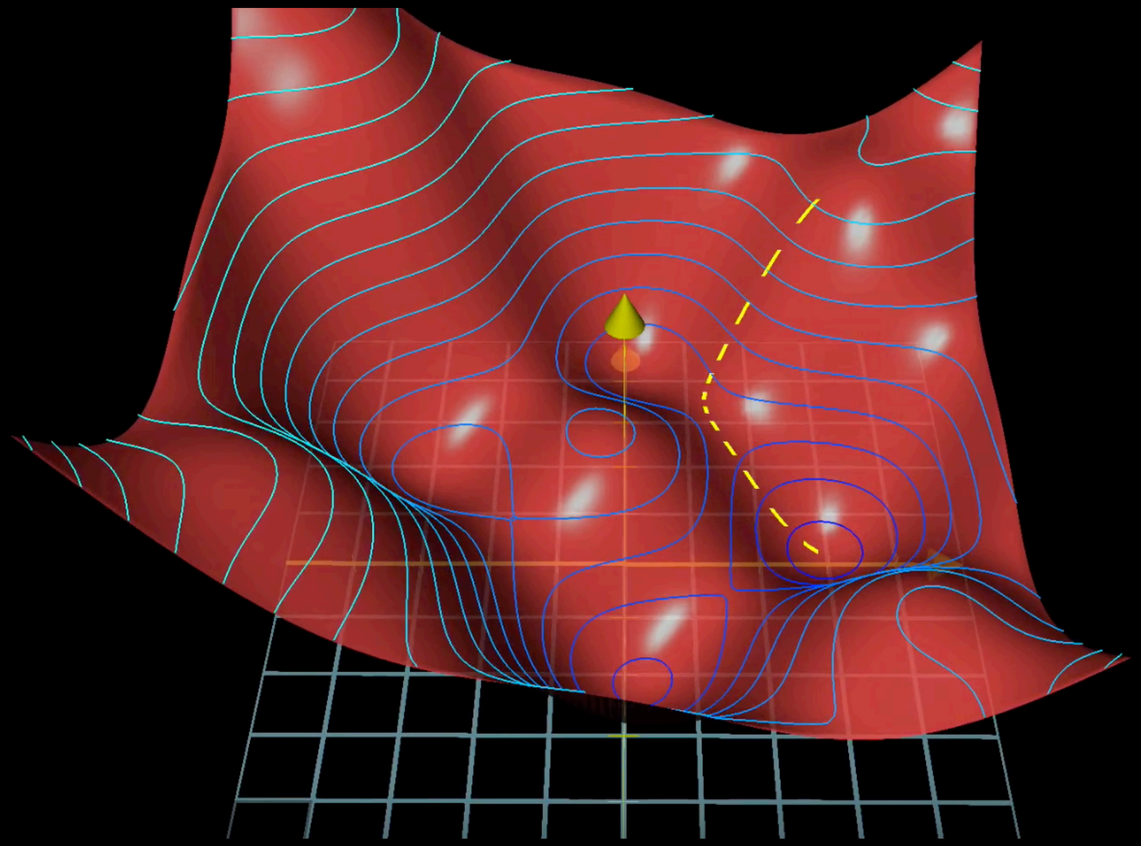
$$
\begin{aligned}
&(0.56 - 0.00)^2 + \\
&(0.01 - 1.00)^2 + \\
&(0.74 - 0.00)^2 + \\
&(0.51 - 0.00)^2 + \\
&(0.71 - 0.00)^2 + \\
&(0.39 - 0.00)^2 + \\
&(0.77 - 0.00)^2 + \\
&(0.74 - 0.00)^2 + \\
&(0.65 - 0.00)^2 + \\
&(0.00 - 0.00)^2
\end{aligned}
$$

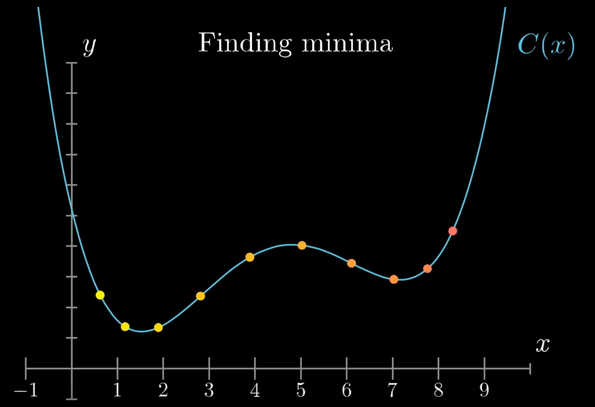
- all examples
- average results
- total cost

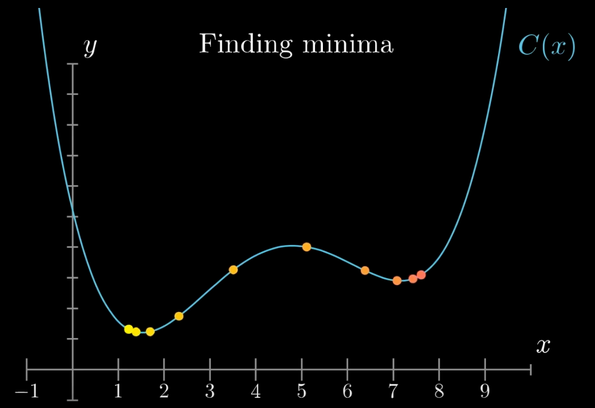
What's the "cost of this difference?

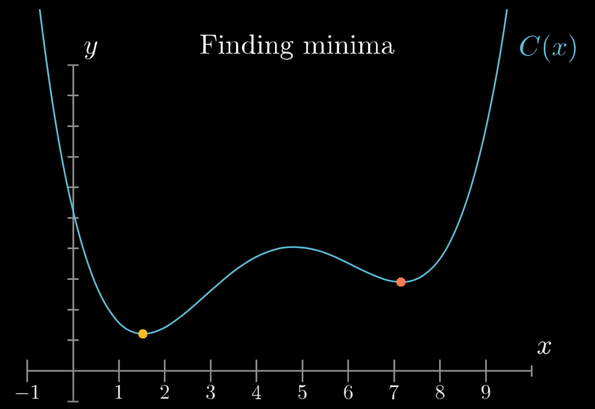


Utter trash

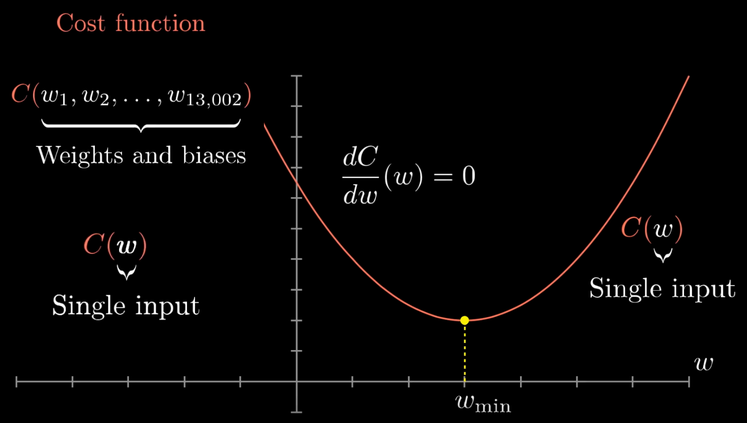


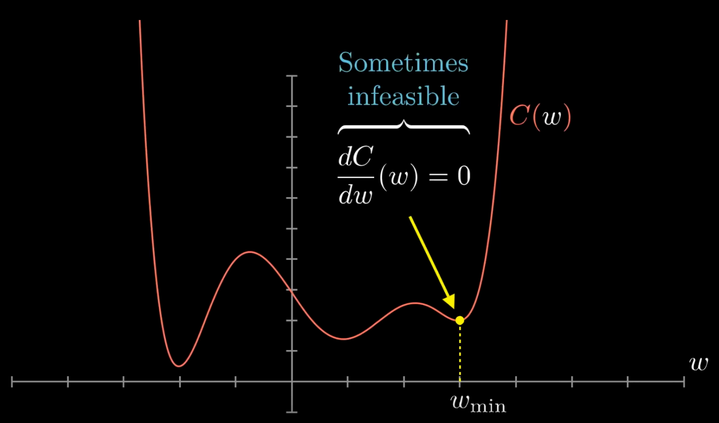


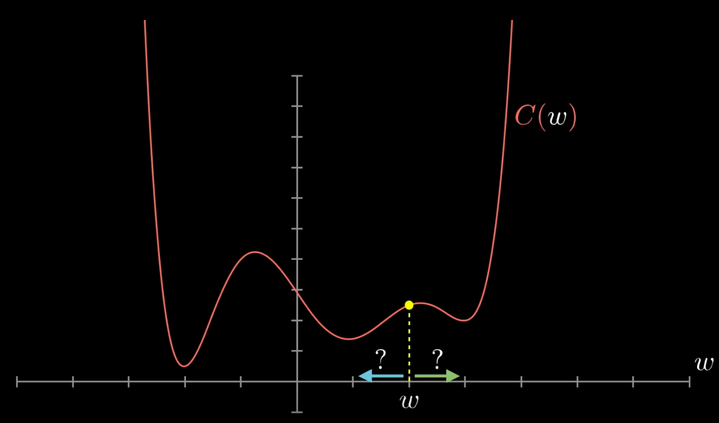


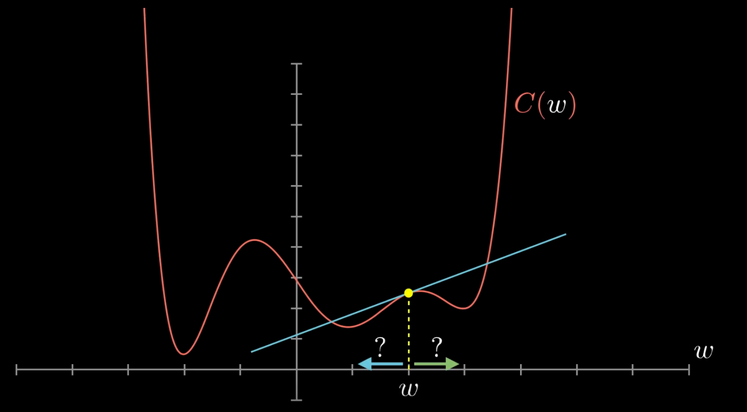


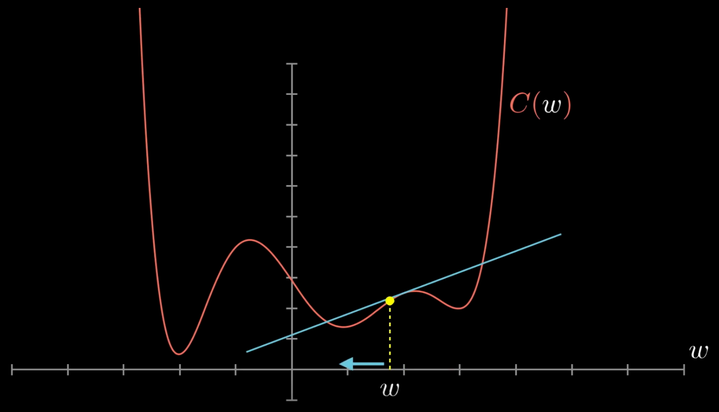
- Learning
  - GD: find w & b
    - min cost func.
      - using all exp.

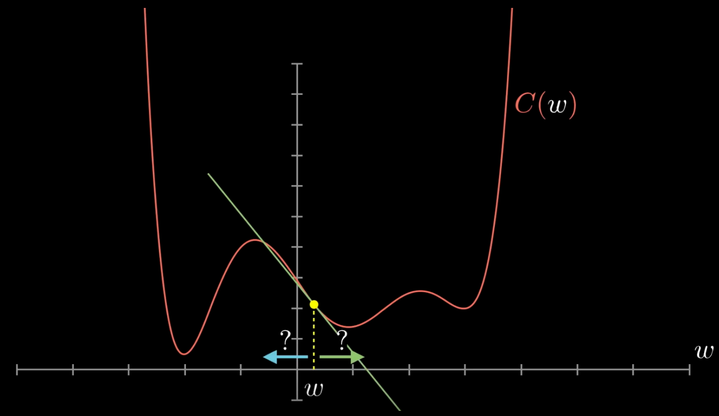

Cost function
$C(w_1, w_2, \ldots, w_{13,002})$
Weights and biases
$\frac{dC}{dw}(w) = 0$
$C(w)$
Single input
$C(w)$
Single input
w
$w_{\min}$

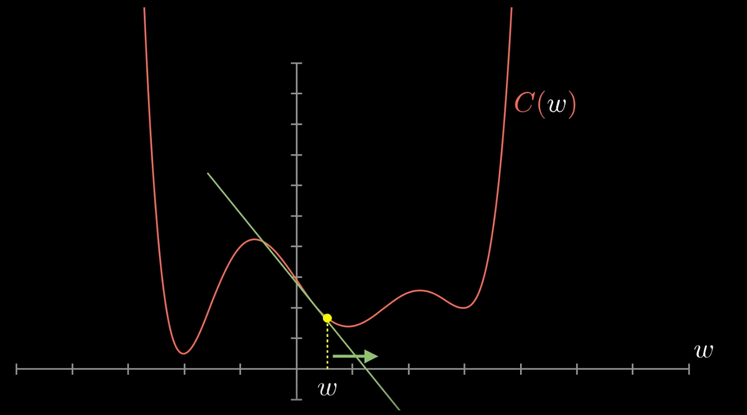

Sometimes infeasible
$\frac{dC}{dw}(w) = 0$
$C(w)$
w
$w_{\min}$

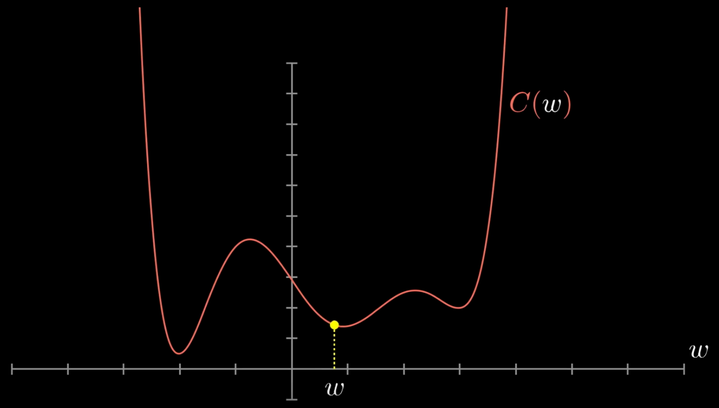

$C(w)$
?
?
w
w

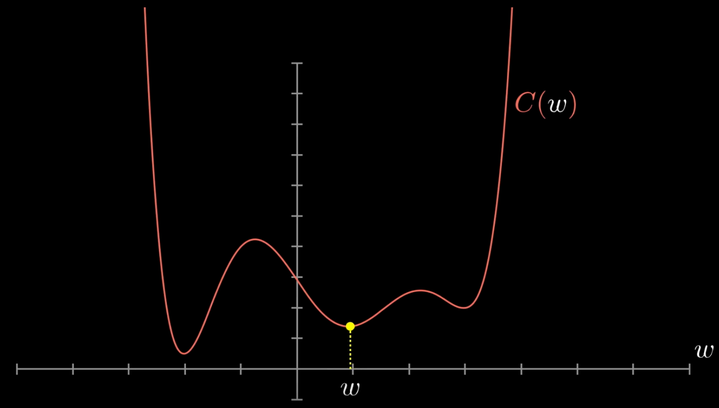

$C(w)$
?
?
w
w

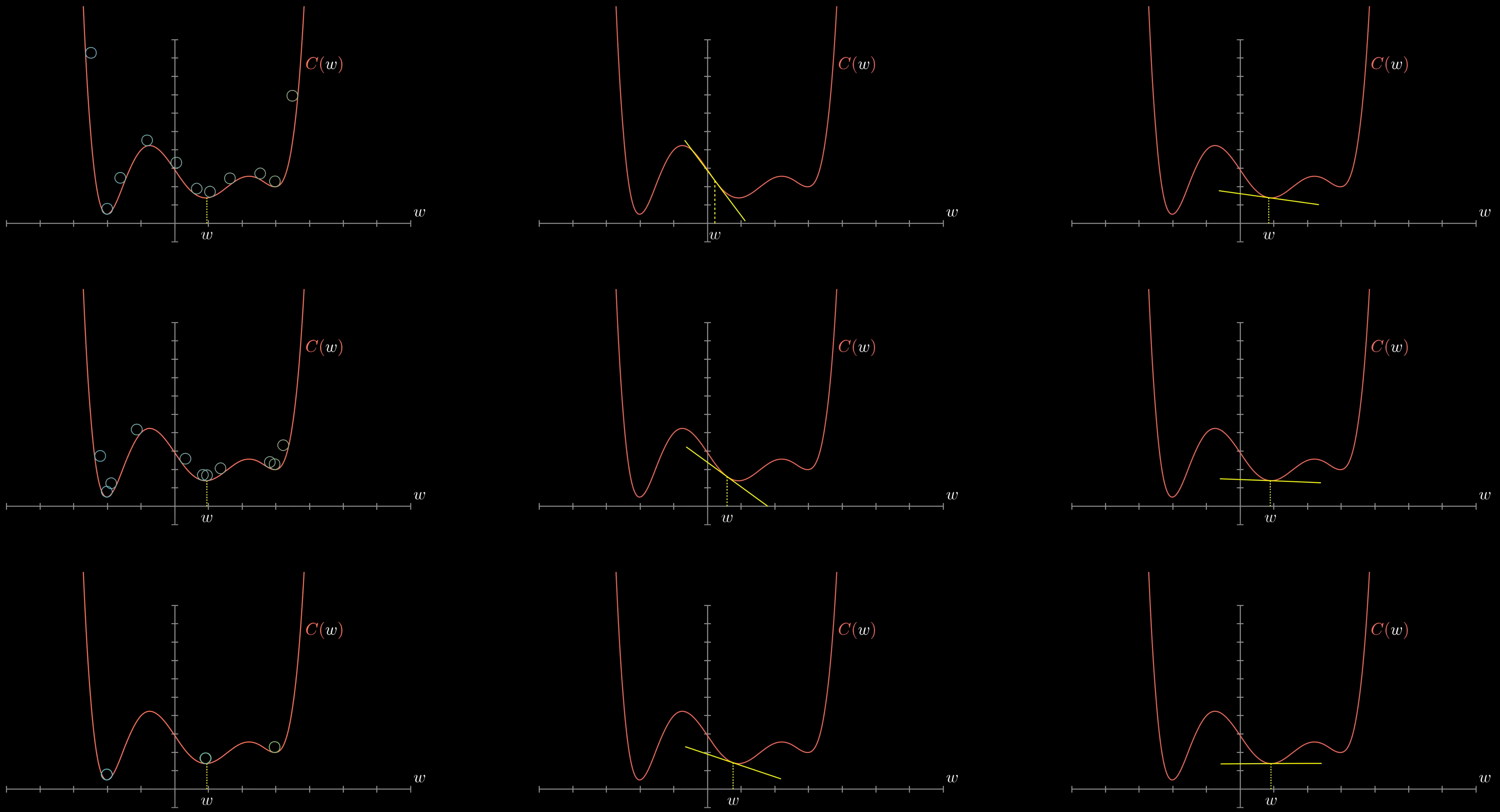

$C(w)$
w
w


$C(w)$
?
?
w
w

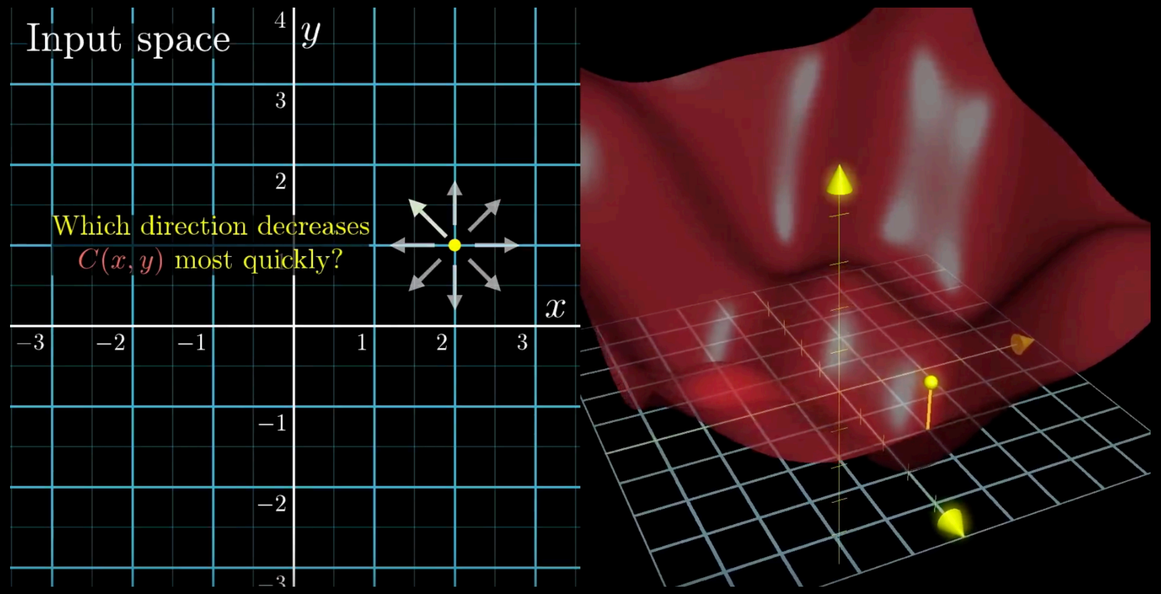

$C(w)$
w
w

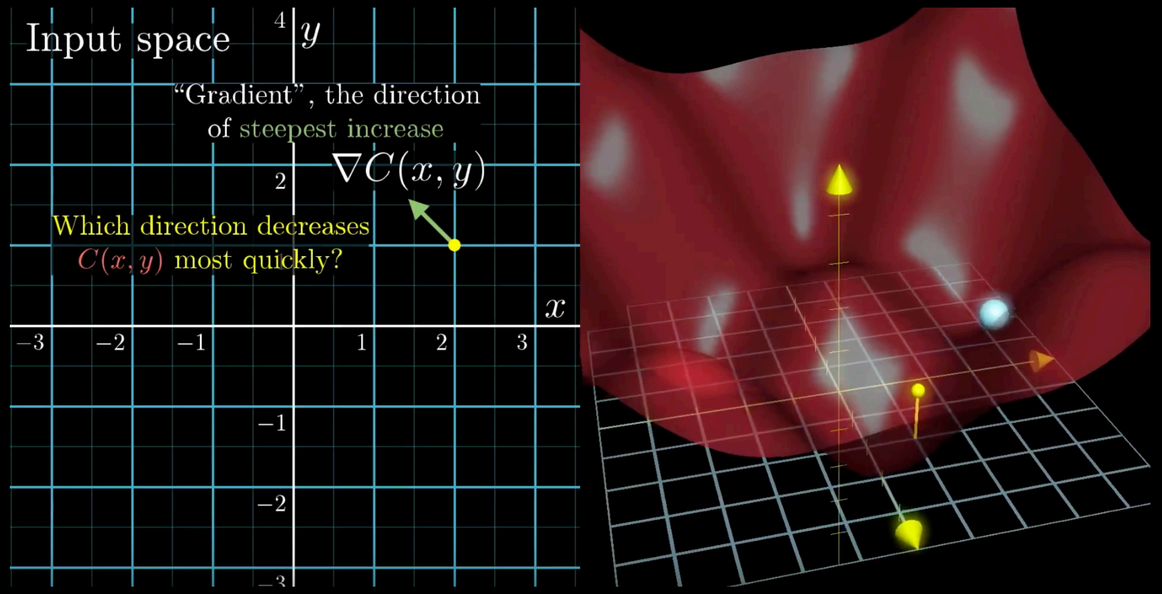

$C(w)$
w
w

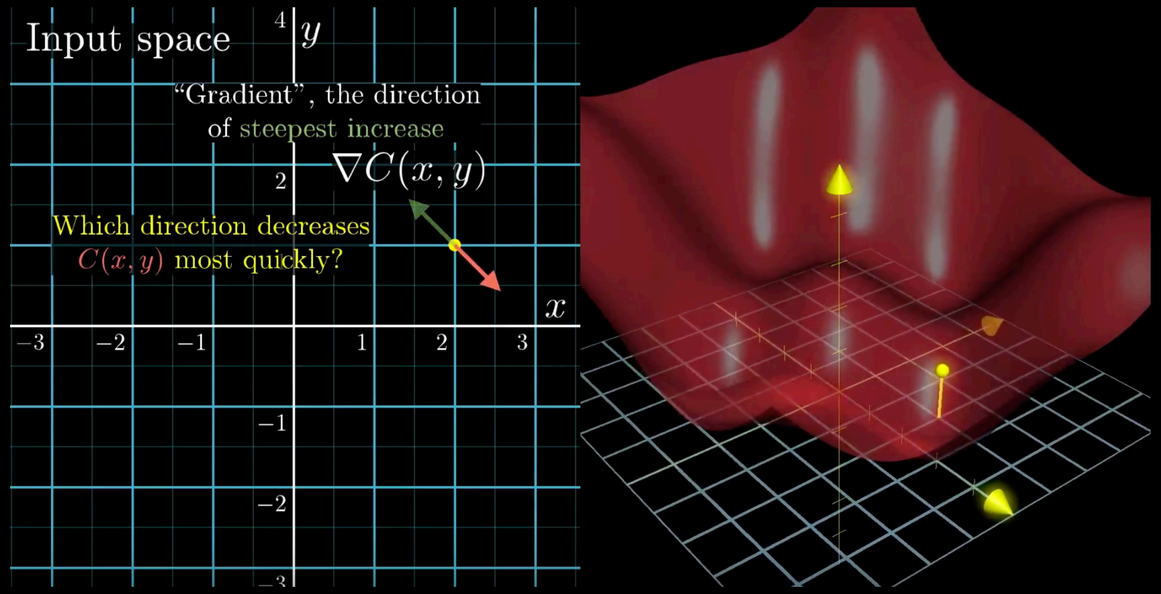

$C(w)$
w
w

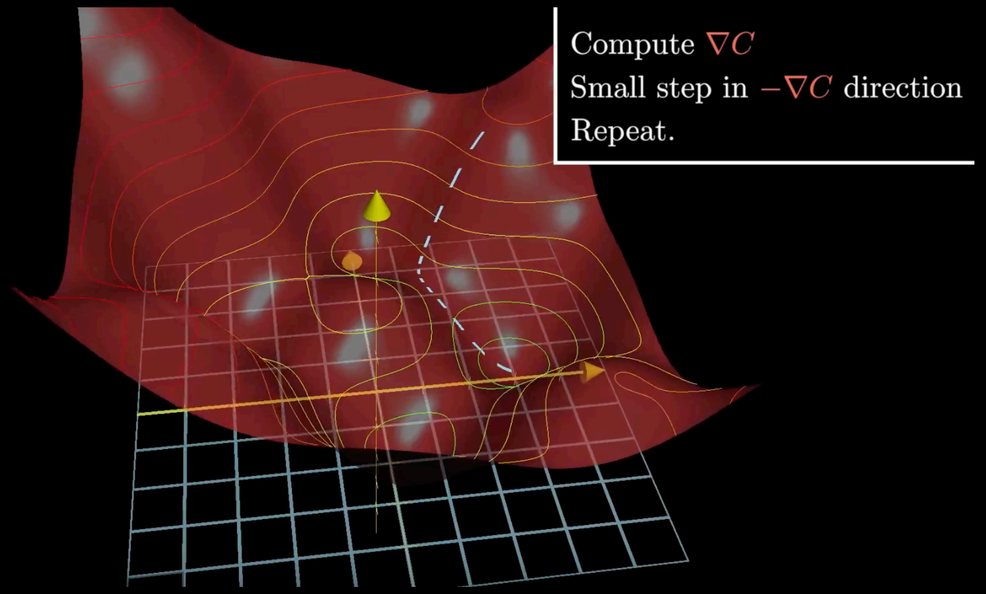


$C(w)$
w


Input space
y
x
Which direction decreases
C(x, y) most quickly?


Input space
y
x
"Gradient", the direction
of steepest increase
∇C(x, y)
Which direction decreases
C(x, y) most quickly?


Input space
y
x
"Gradient", the direction
of steepest increase
∇C(x, y)
Which direction decreases
C(x, y) most quickly?


Compute ∇C
Small step in −∇C direction
Repeat.

13,002 weights and biases

$$\vec{\mathbf{W}} = \begin{bmatrix} 2.25 \\ -1.57 \\ 1.98 \\ \vdots \\ -1.16 \\ 3.82 \\ 1.21 \end{bmatrix}$$

How to nudge all weights and biases

$$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.18 \\ 0.45 \\ -0.51 \\ \vdots \\ 0.40 \\ -0.32 \\ 0.82 \end{bmatrix}$$

$$\vec{\mathbf{W}} = \begin{bmatrix} 2.25 \\ -1.57 \\ 1.98 \\ \vdots \\ -1.16 \\ 3.82 \\ 1.21 \end{bmatrix} \quad 0.18$$

$$\vec{\mathbf{W}} = \begin{bmatrix} 2.43 \\ -1.57 \\ 1.98 \\ \vdots \\ -1.16 \\ 3.82 \\ 1.21 \end{bmatrix}$$

$$\vec{\mathbf{W}} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{13,000} \\ w_{13,001} \\ w_{13,002} \end{bmatrix}$$

$$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.78 \\ -0.37 \end{bmatrix}$$

w_0 should increase
w_1 should increase
w_2 should decrease

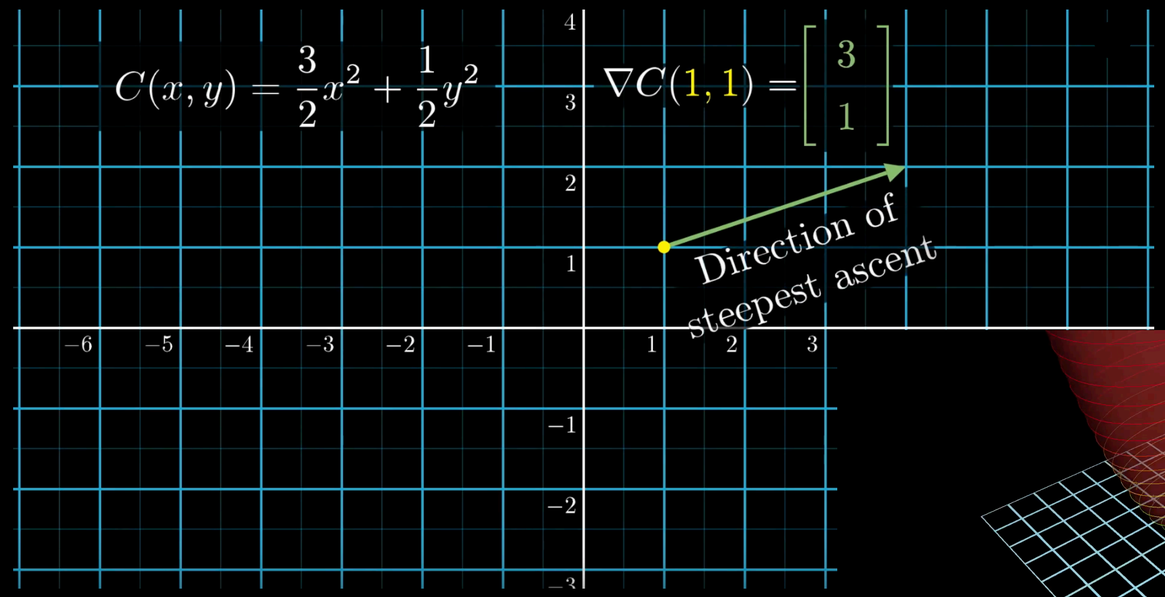
$w_{13,000}$ should increase
$w_{13,001}$ should decrease

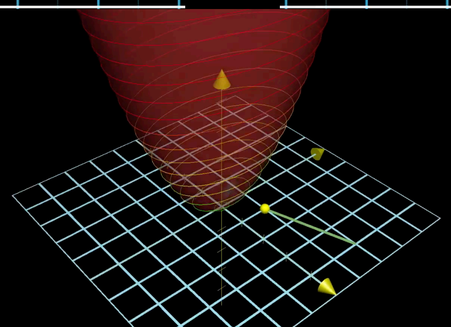
$$\vec{\mathbf{W}} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{13,000} \\ w_{13,001} \\ w_{13,002} \end{bmatrix}$$

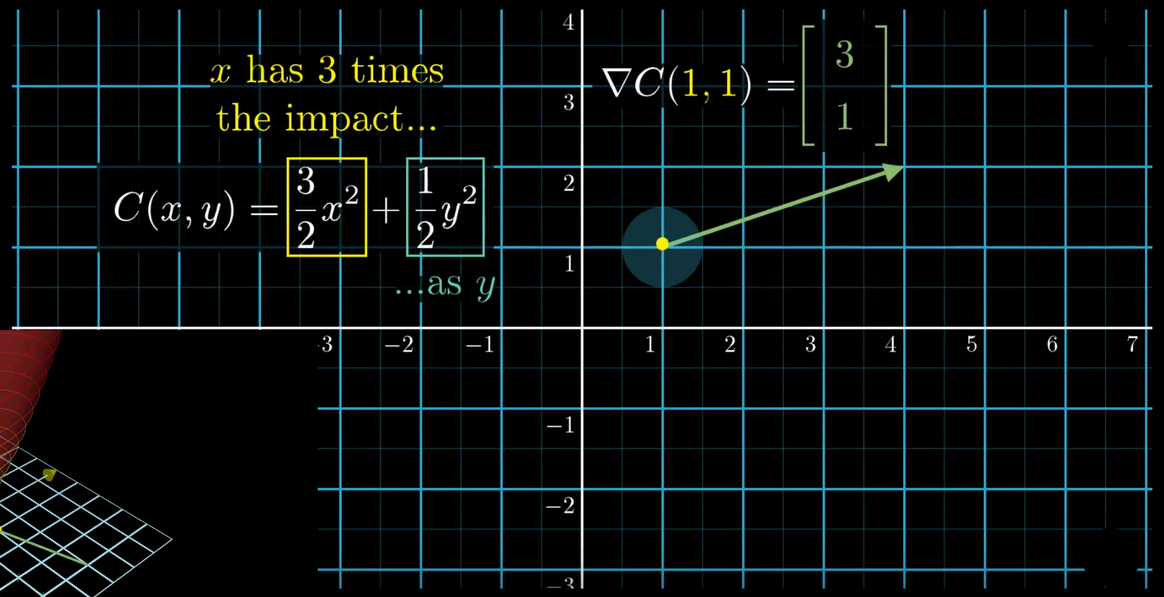
$$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.78 \\ -0.37 \end{bmatrix}$$

w_0 should increase somewhat
w_1 should increase a little
w_2 should decrease a lot

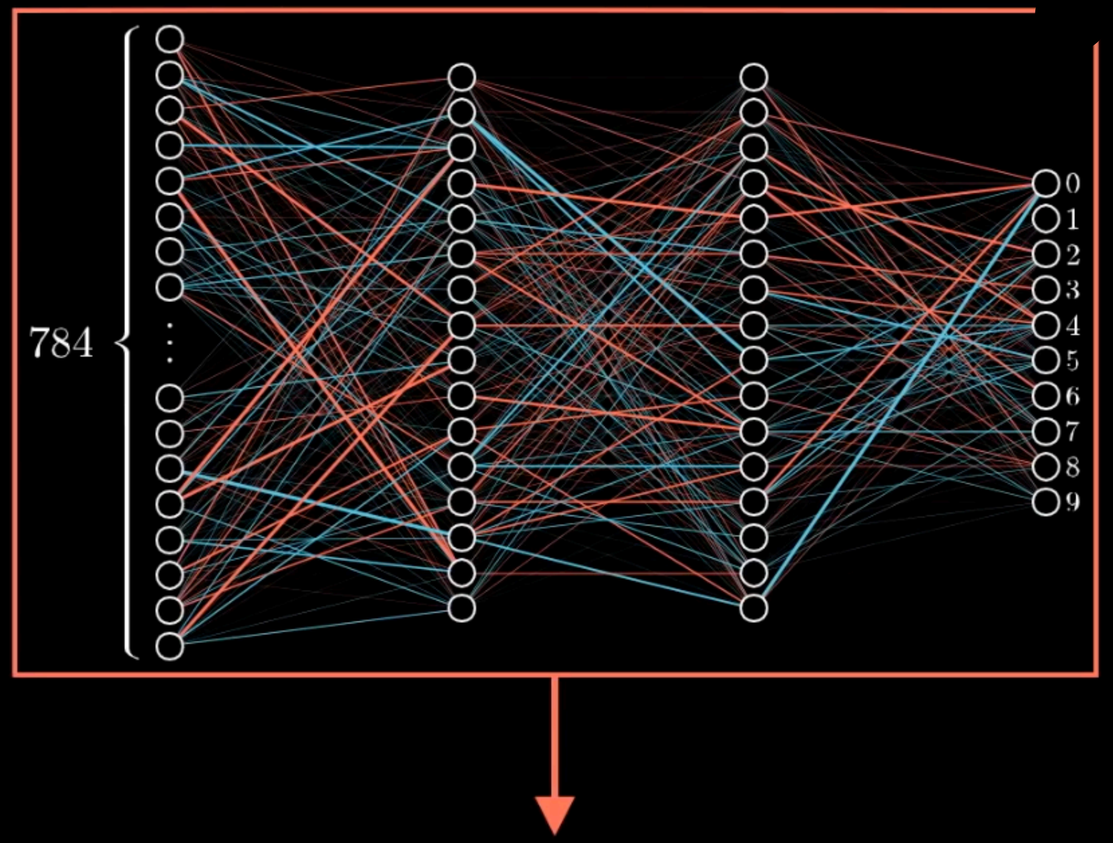
$w_{13,000}$ should increase a lot
$w_{13,001}$ should decrease somewhat







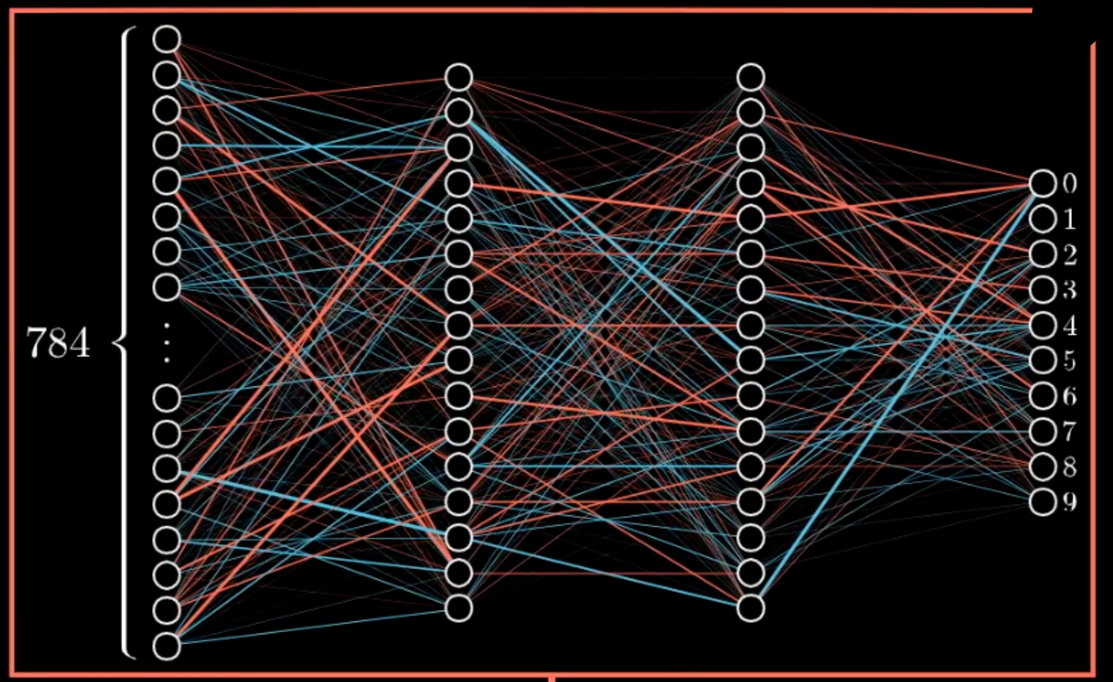
$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.20 \\ 0.83 \\ -0.84 \\ \vdots \\ 0.04 \\ 1.57 \\ 1.59 \end{bmatrix}$$



$$C(w_0, w_1, \ldots, w_{13,001}) = 3.4$$

- Really looking for:
  - negative gradient of cost f.

$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.20 \\ 0.83 \\ -0.84 \\ \vdots \\ 0.04 \\ 1.57 \\ 1.59 \end{bmatrix}$$



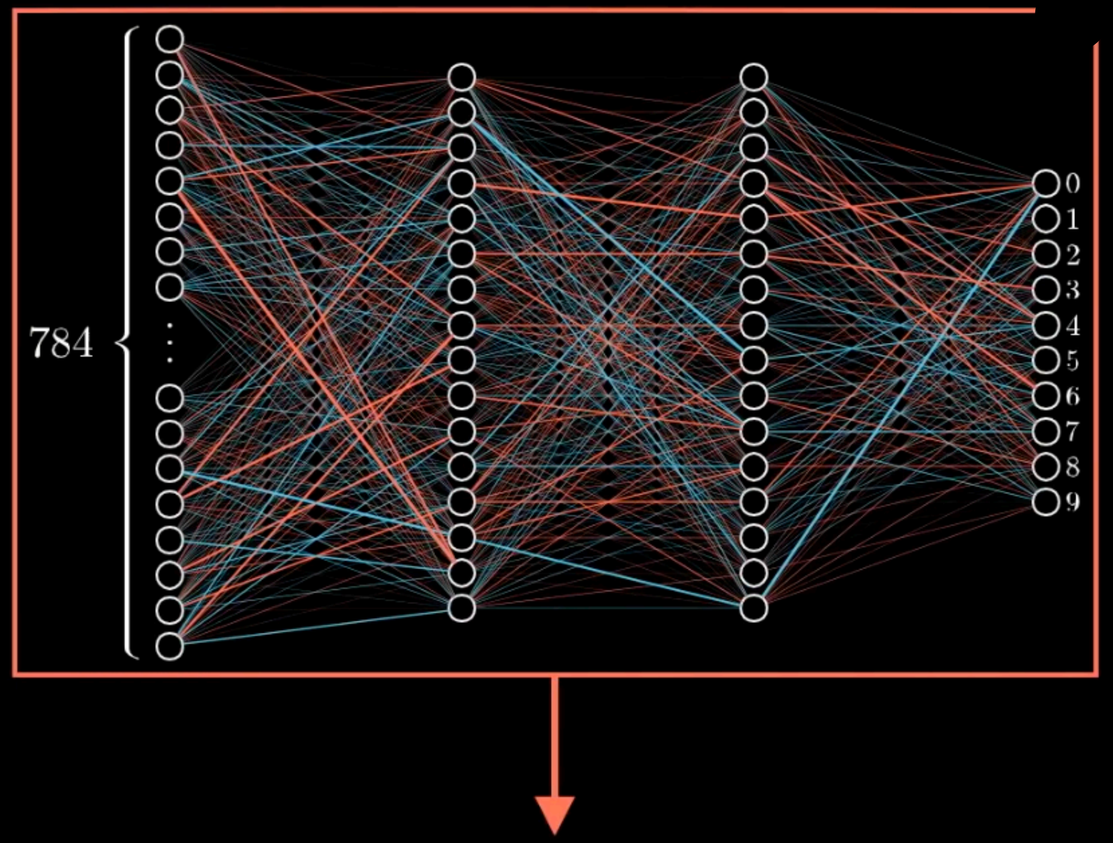
$$C(w_0, w_1, \ldots, w_{13,001}) = 3.4$$

$$0.20\ \ 0.83\ \ -0.84\ \cdots\ 0.04\ \ 1.57\ \ 1.59$$

Nudge weights

- tells how to change all w & b

$$-\nabla C(\underbrace{\dots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.20 \\ 0.83 \\ -0.84 \\ \vdots \\ 0.04 \\ 1.57 \\ 1.59 \end{bmatrix}$$

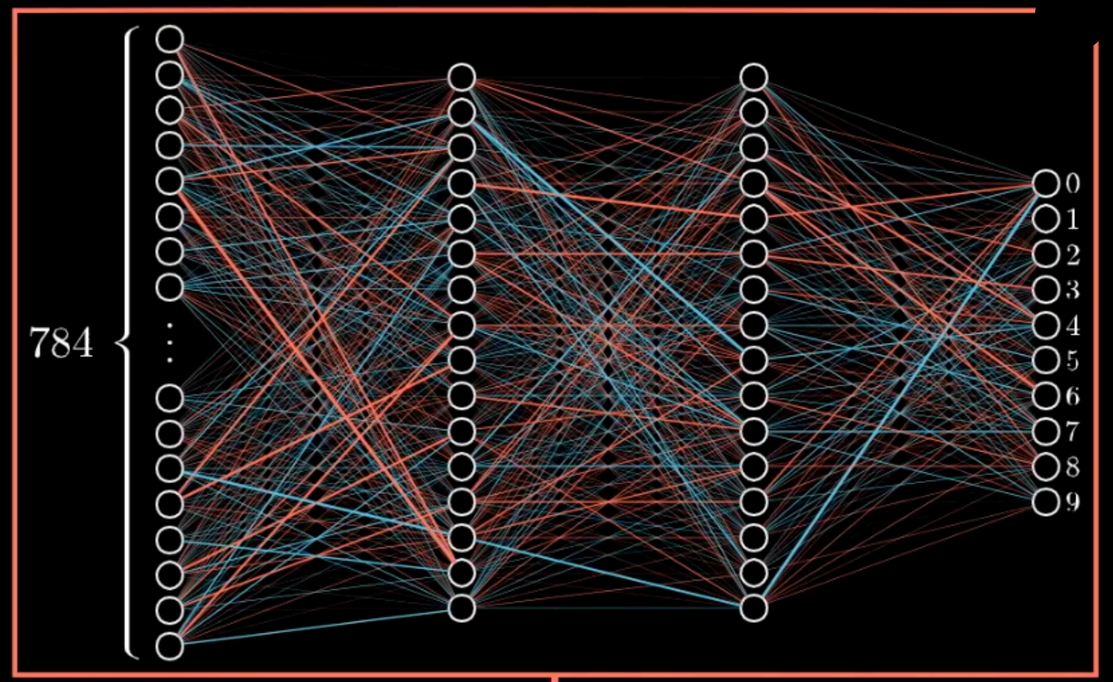


$$C(w_0, w_1, \dots, w_{13,001}) = 3.27$$

- to decrease the cost

$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.19 \\ 0.86 \\ -0.85 \\ \vdots \\ -0.05 \\ 1.65 \\ 1.54 \end{bmatrix}$$

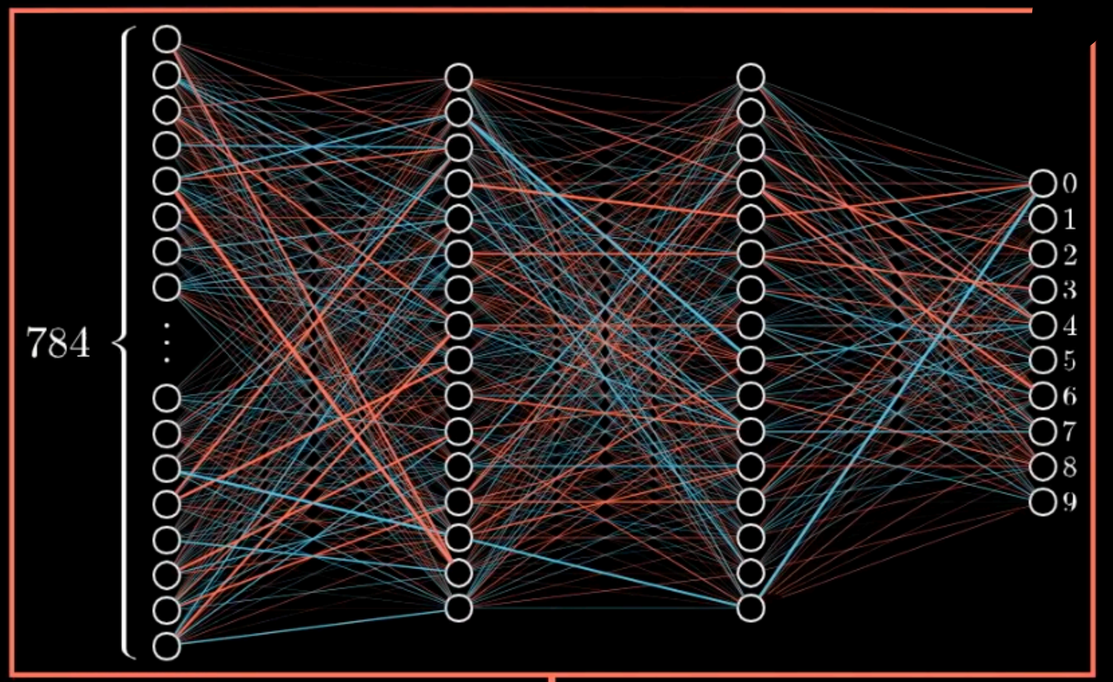
Recompute gradient



$$C(w_0, w_1, \ldots, w_{13,001}) = 3.27$$

- iterative process, each step involves:
  - recompute negative gradient

$$-\nabla C(\underbrace{\dots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.19 \\ 0.86 \\ -0.85 \\ \vdots \\ -0.05 \\ 1.65 \\ 1.54 \end{bmatrix}$$



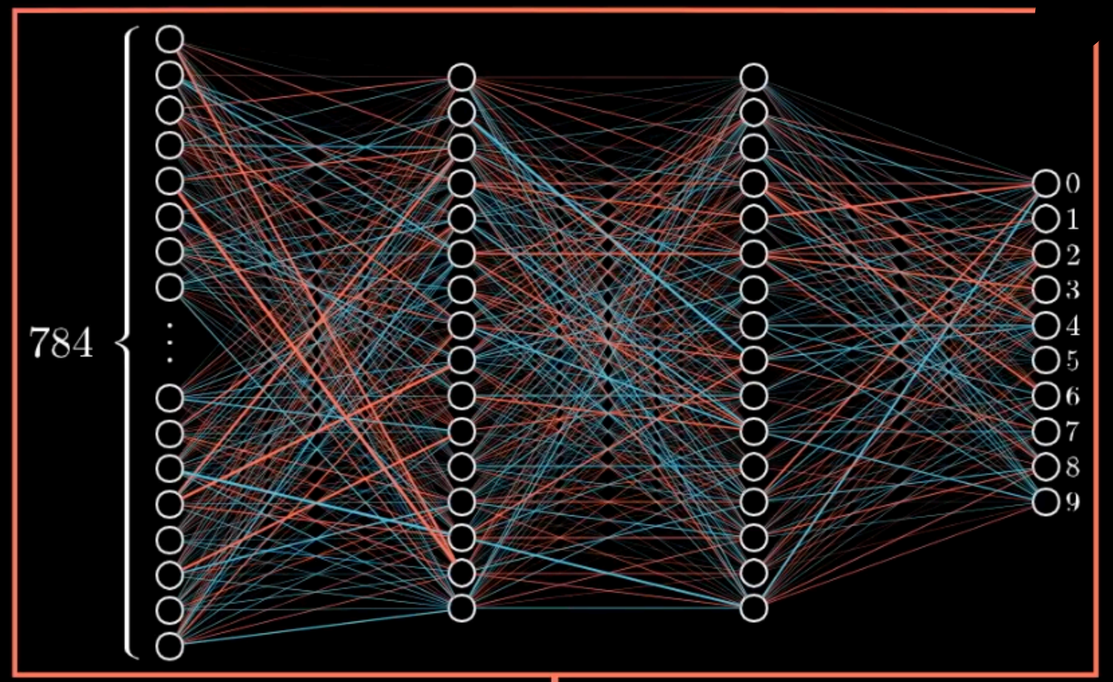
$$C(w_0, w_1, \dots, w_{13,001}) = 3.27$$

$$0.19 \;\; 0.86 \;\; -0.85 \;\; \cdots \;\; -0.05 \;\; 1.65 \;\; 1.54$$

Nudge weights

- update all w & b

$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.19 \\ 0.86 \\ -0.85 \\ \vdots \\ -0.05 \\ 1.65 \\ 1.54 \end{bmatrix}$$

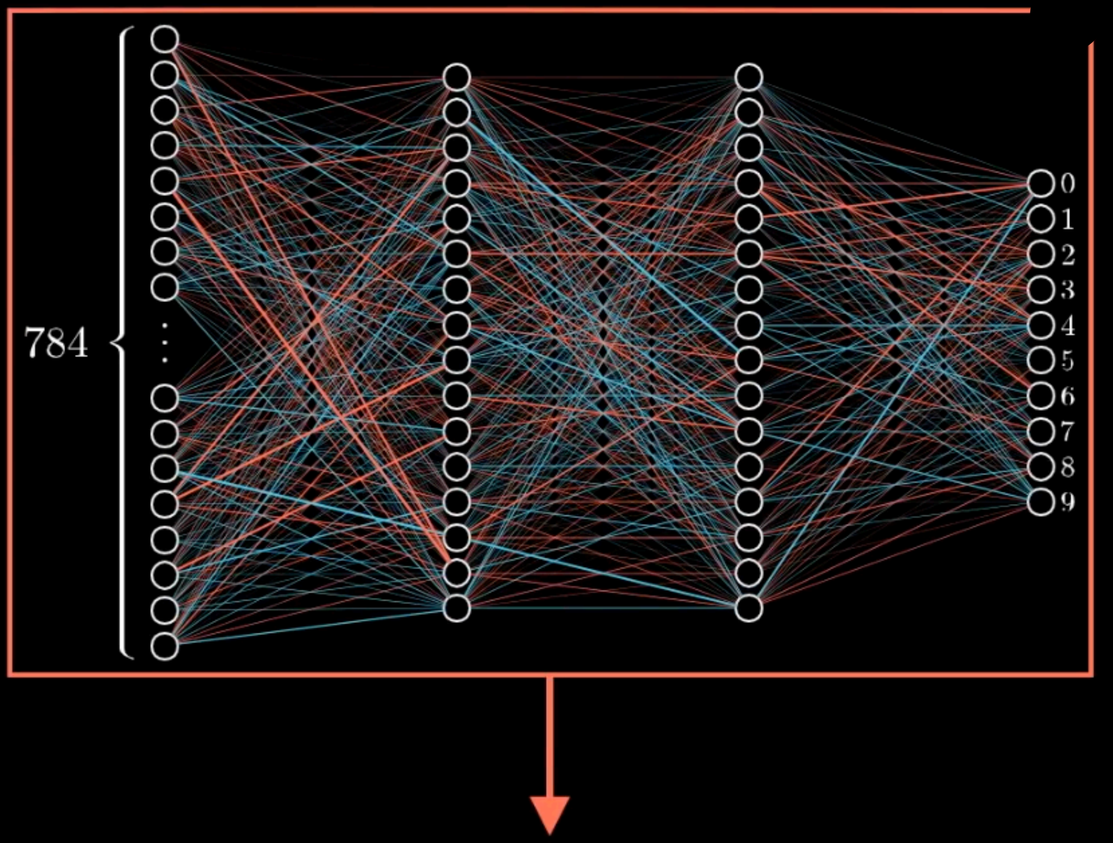


$$C(w_0, w_1, \ldots, w_{13,001}) = 3.06$$

- bit closer to a trained NN

$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.17 \\ 0.80 \\ -0.87 \\ \vdots \\ -0.04 \\ 1.58 \\ 1.59 \end{bmatrix}$$

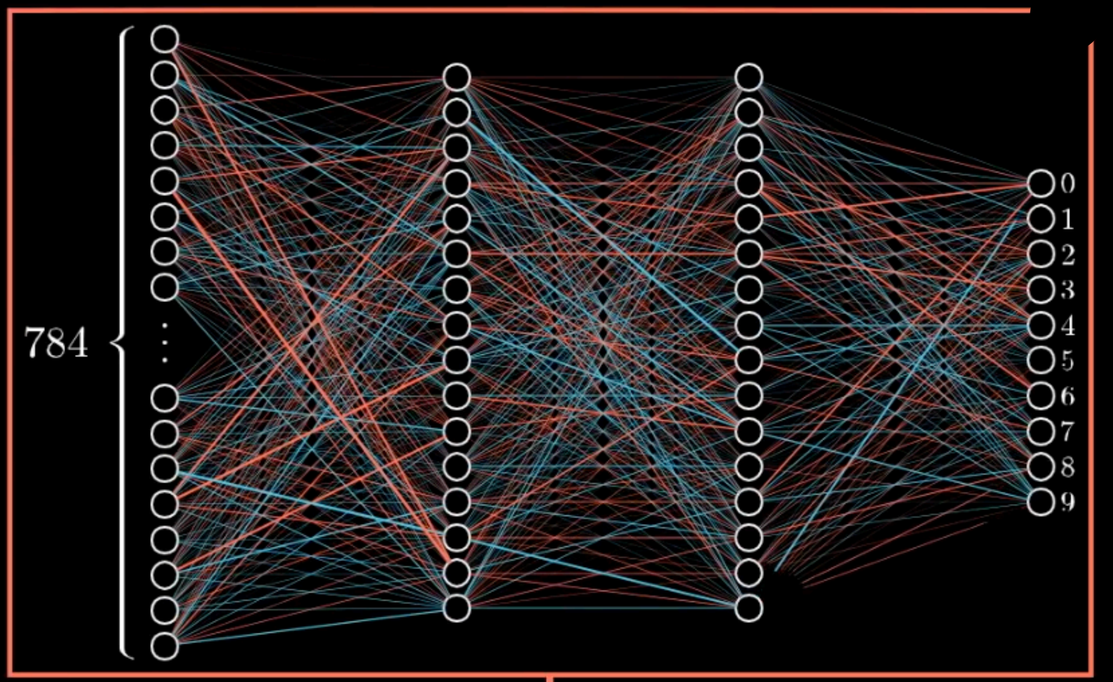
Recompute gradient



$$C(w_0, w_1, \ldots, w_{13,001}) = 3.06$$

- recompute

$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.17 \\ 0.80 \\ -0.87 \\ \vdots \\ -0.04 \\ 1.58 \\ 1.59 \end{bmatrix}$$



$$C(w_0, w_1, \ldots, w_{13,001}) = 3.06$$

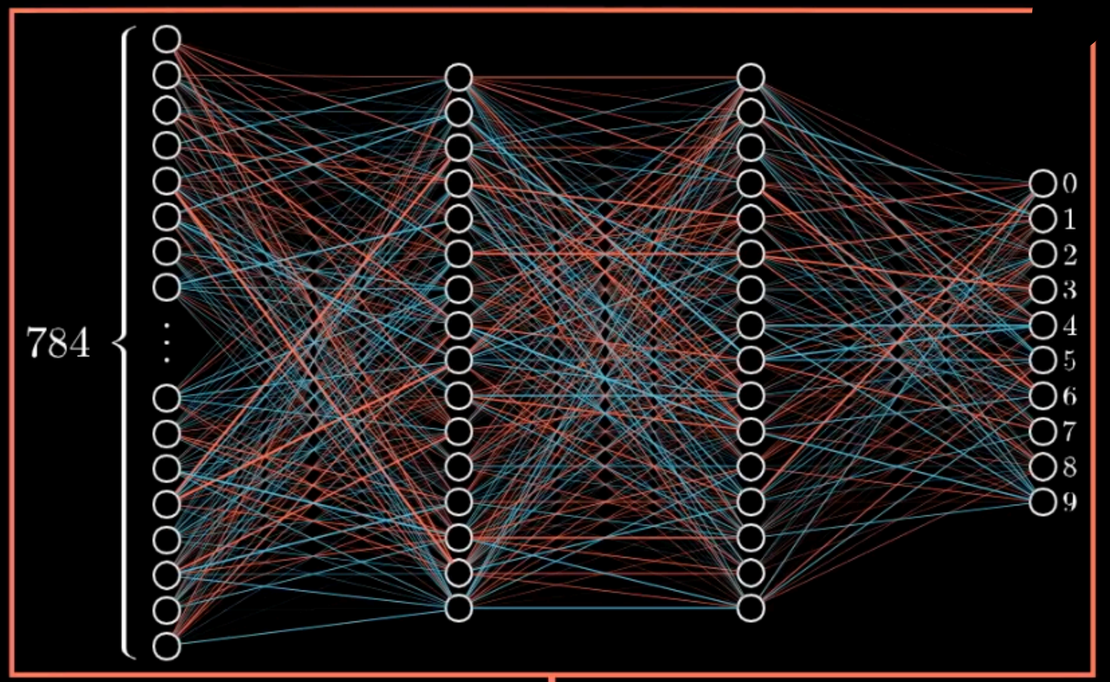
$$0.17 \;\; 0.80 \;\; -0.87 \;\; \cdots \;\; -0.04 \;\; 1.58 \;\; 1.59$$

Nudge weights

- update

$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.17 \\ 0.80 \\ -0.87 \\ \vdots \\ -0.04 \\ 1.58 \\ 1.59 \end{bmatrix}$$

Recompute gradient

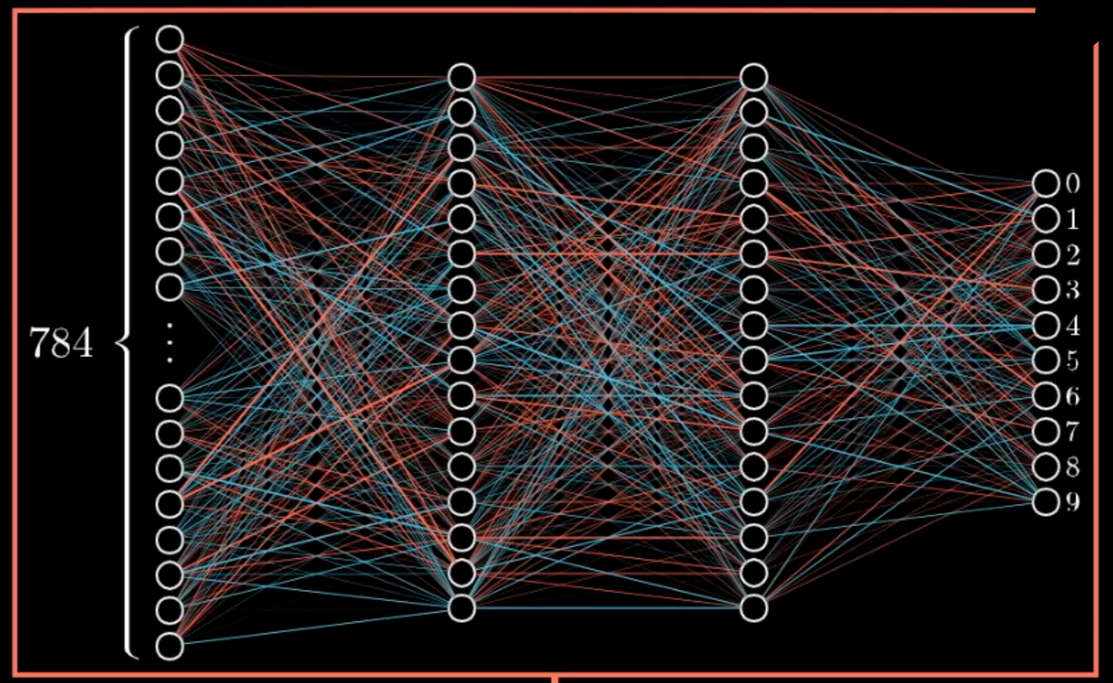


$$C(w_0, w_1, \ldots, w_{13,001}) = 2.85$$

- trained NN?

$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.16 \\ 0.72 \\ -0.93 \\ \vdots \\ 0.04 \\ 1.64 \\ 1.52 \end{bmatrix}$$

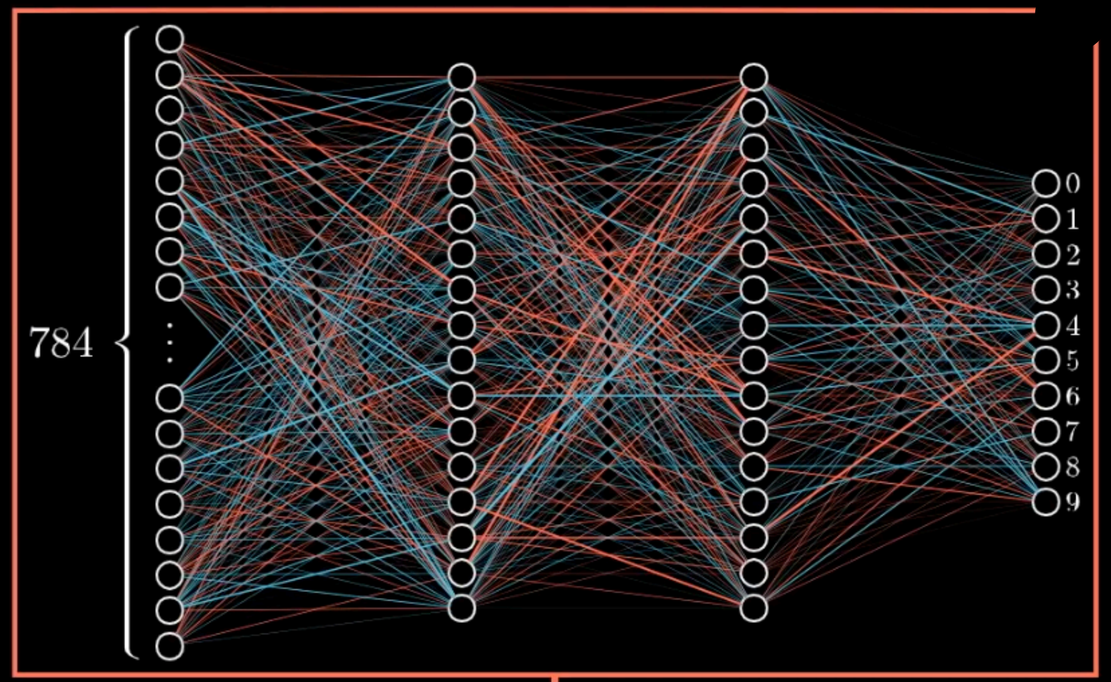
Direction in 13,002 dimensions?!?



$$C(w_0, w_1, \ldots, w_{13{,}001}) = 2.85$$

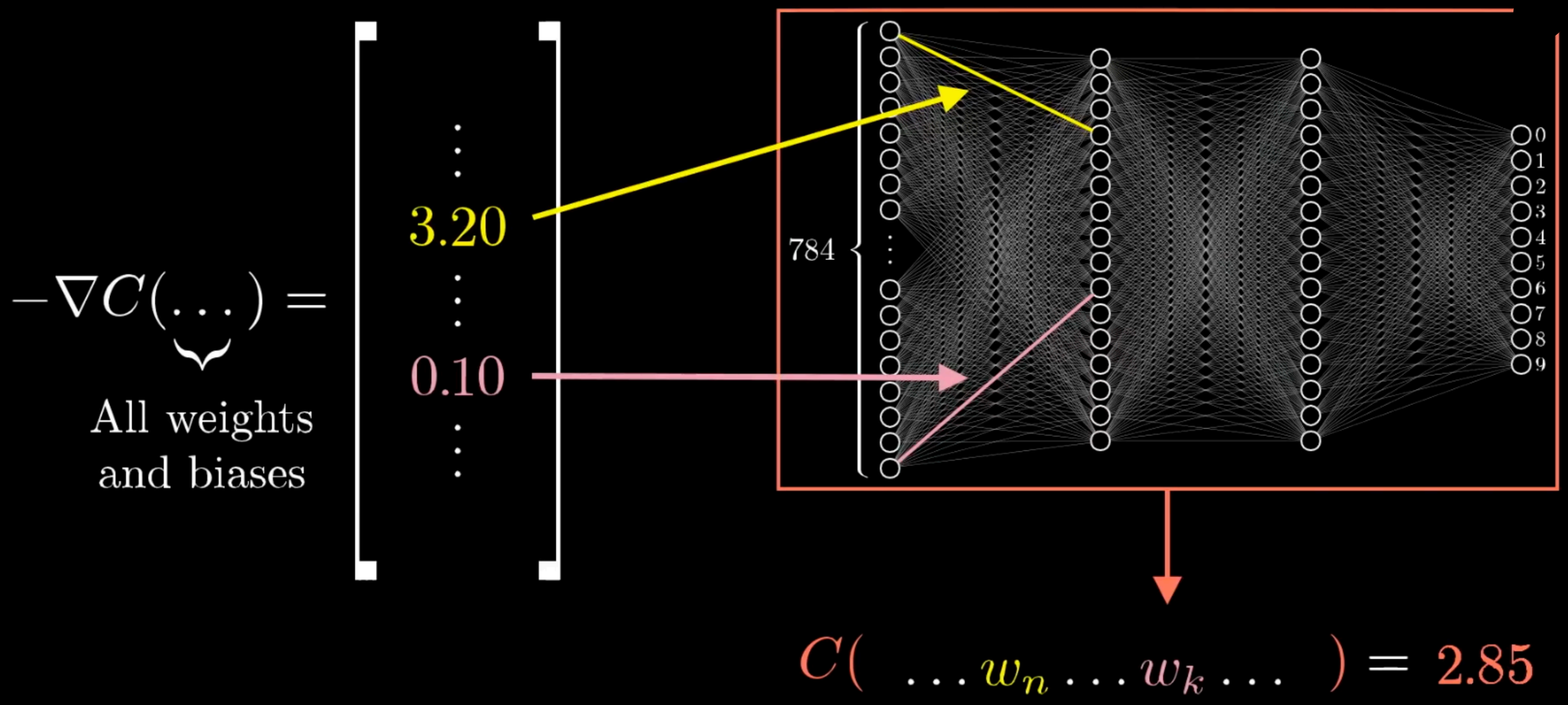
- important idea
- direction in 13000 dim hard
- alternative way to think..

$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.16 \\ 0.72 \\ -0.93 \\ \vdots \\ 0.04 \\ 1.64 \\ 1.52 \end{bmatrix}$$

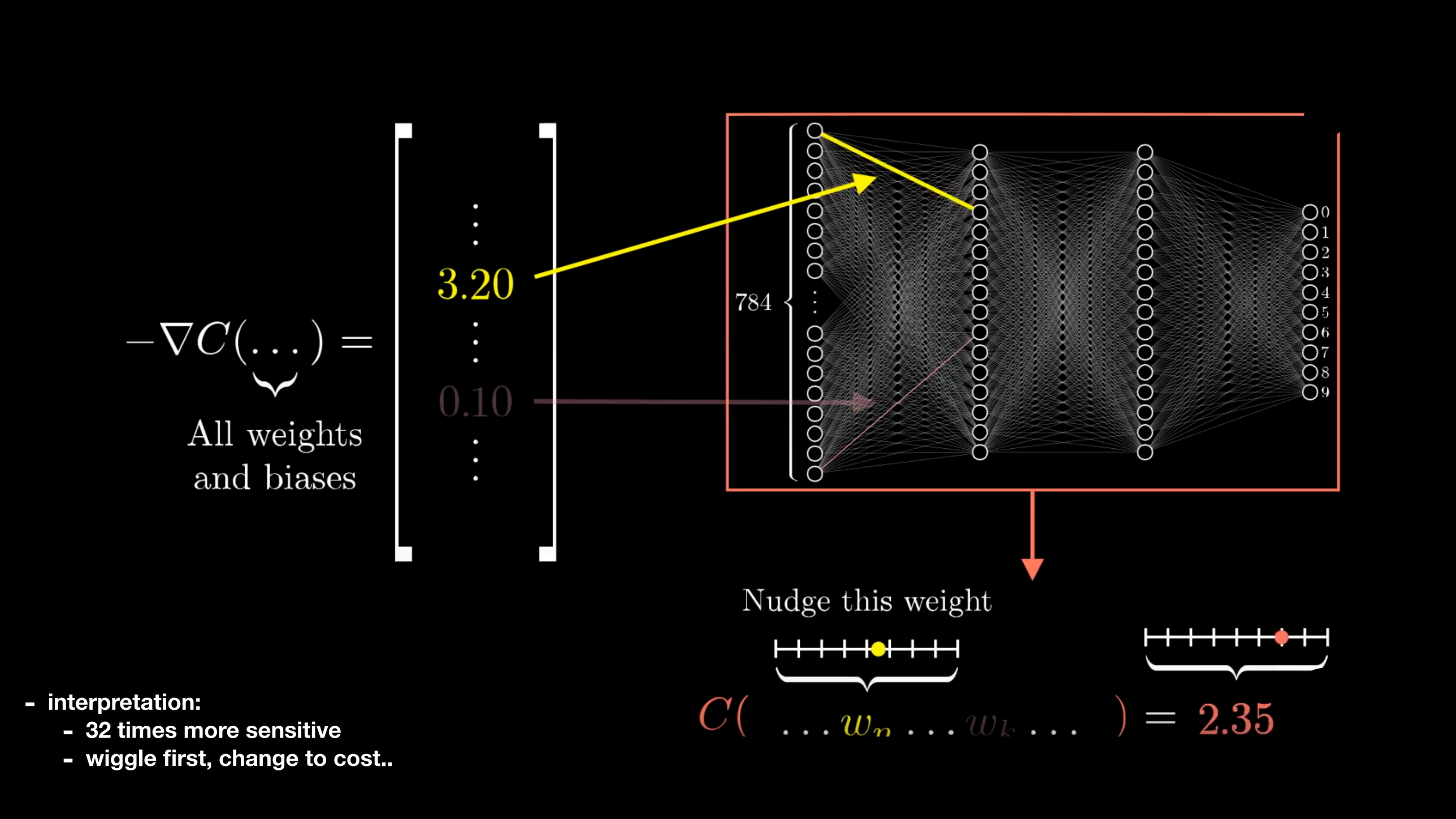


$$C(w_0, w_1, \ldots, w_{13,001}) = 2.85$$

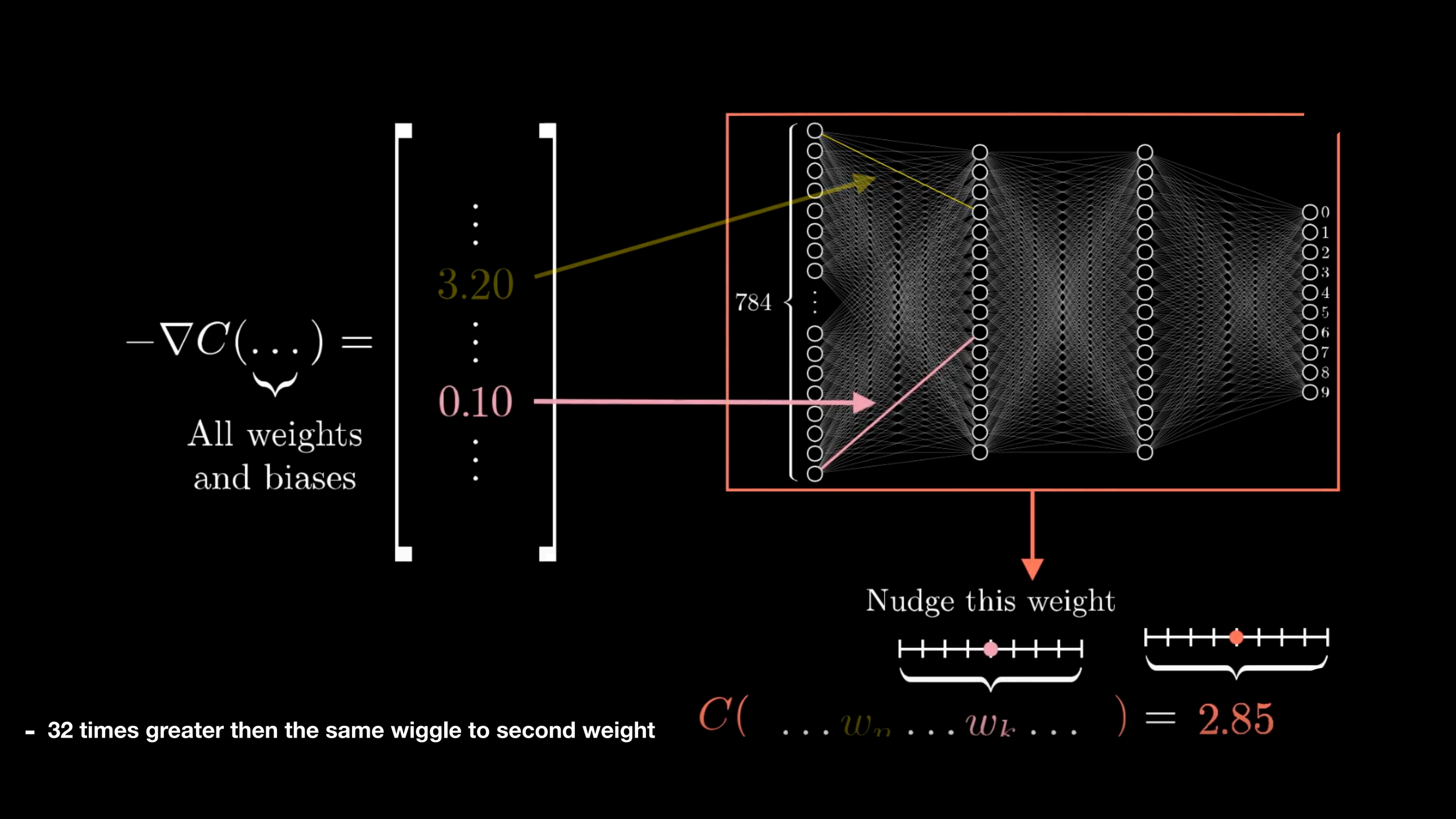
- magnitude of each component
- tells how sensitive the cost f.
- is to each weight and bias



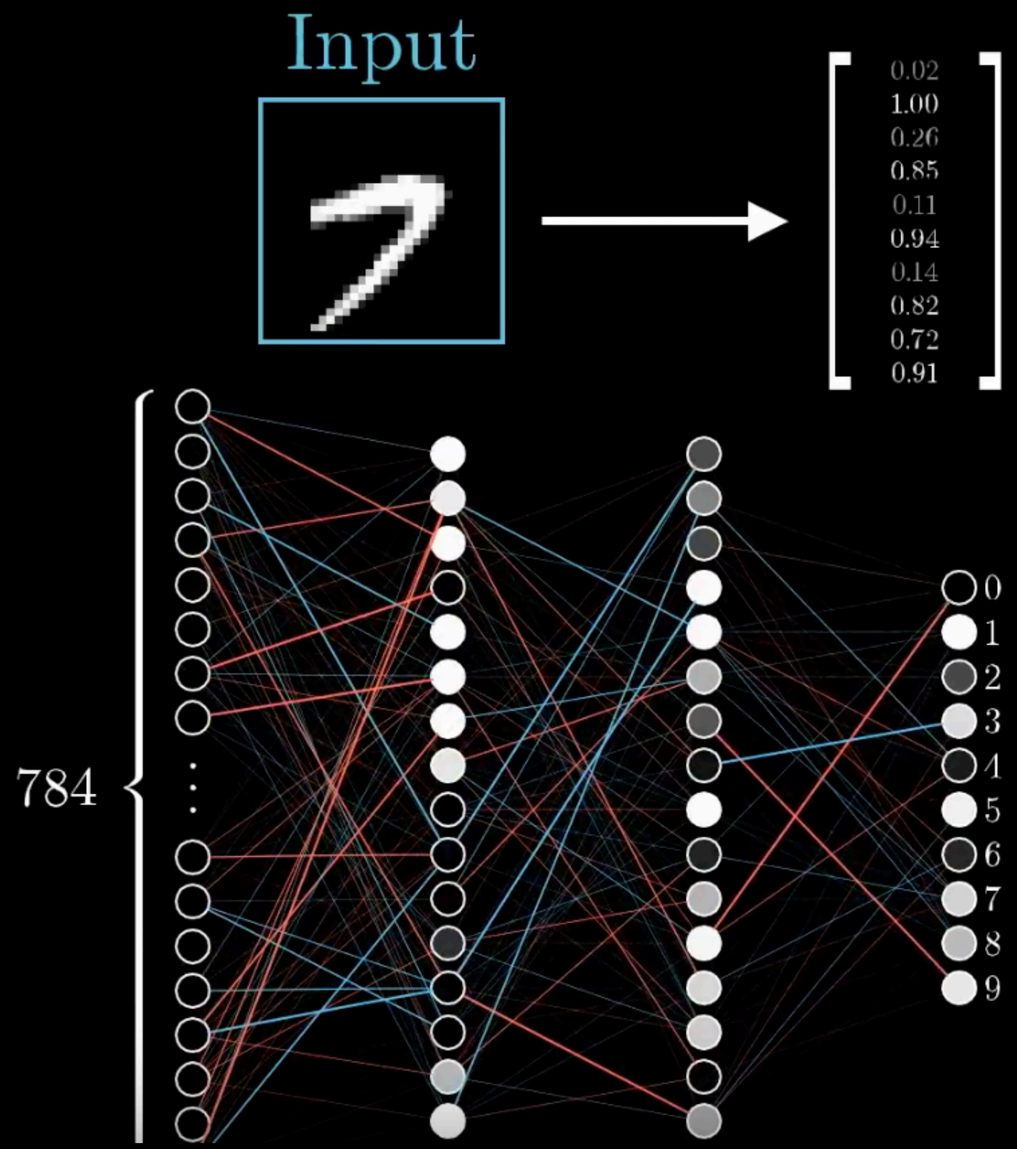
- e.g.
  - two components associated with two wights
  - one is 3.2, the other is 0.1
  - what does it mean?



- interpretation:
  - 32 times more sensitive
  - wiggle first, change to cost..



- 32 times greater then the same wiggle to second weight



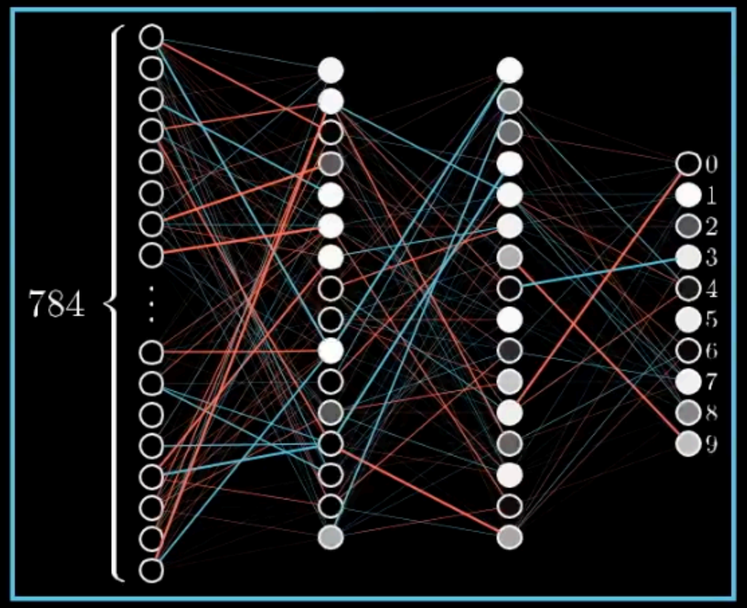
Neural network function

Input: 784 numbers (pixels)

Output: 10 numbers

Parameters: 13,002 weights/biases

Input



Cost: 5.4

Cost function

Input: 13,002 weights/biases

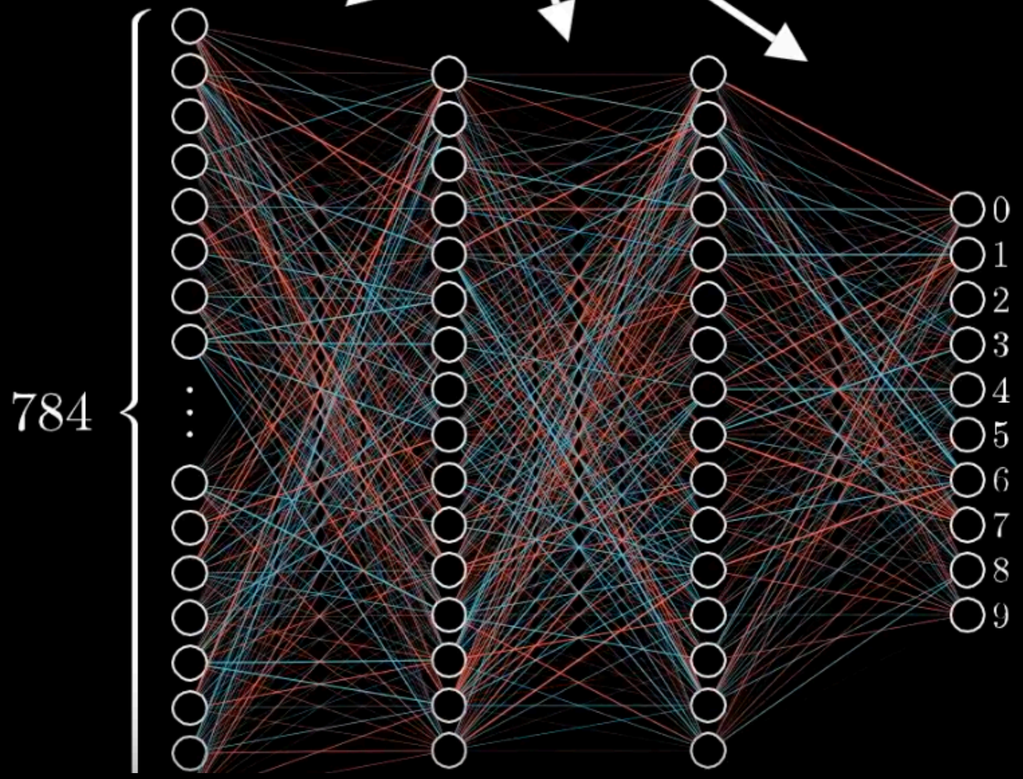
Output: 1 number (the cost)

Parameters: Many, many, many training examples

$$\left(\boxed{6}, 6\right)$$

$$-\nabla C(\underbrace{\dots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.01 \\ 0.68 \\ -0.59 \\ 0.03 \\ \vdots \\ 1.54 \\ 1.56 \\ -1.44 \\ -1.17 \end{bmatrix}$$

Change by some small multiple of $-\nabla C(\dots)$



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

ABSTRACT

Despite their massive size, successful deep artificial neural networks can exhibit a remarkably small difference between training and test performance. Conventional wisdom attributes small generalization error either to properties of the model family, or to the regularization techniques used during training.

Through extensive systematic experiments, we show how these traditional approaches fail to explain why large neural networks generalize well in practice. Specifically, our experiments establish that state-of-the-art convolutional networks for image classification trained with stochastic gradient methods easily fit a random labeling of the training data. This phenomenon is qualitatively unaffected by explicit regularization, and occurs even if we replace the true images by completely unstructured random noise. We corroborate these experimental findings with a theoretical construction showing that simple depth two neural networks already have perfect finite sample expressivity as soon as the number of parameters


Lion


Genius


Fork


Trilobite


Puppy


Astrolabe


Songbird of our generation


Cow


Sculling


Tease


Fork


Cow


Trilobite


Puppy


Lion

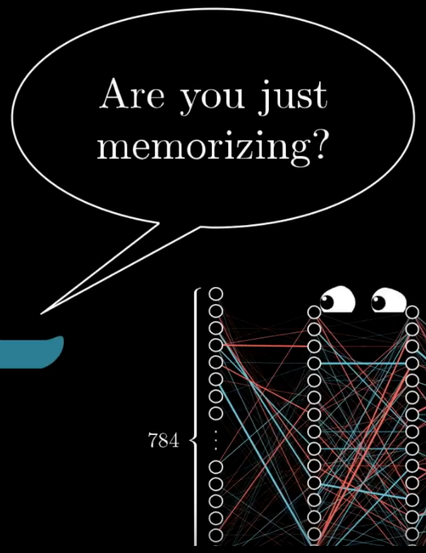

Sculling


Tease


Genius


Astrolabe


Songbird



A Closer Look at Memorization in Deep Networks

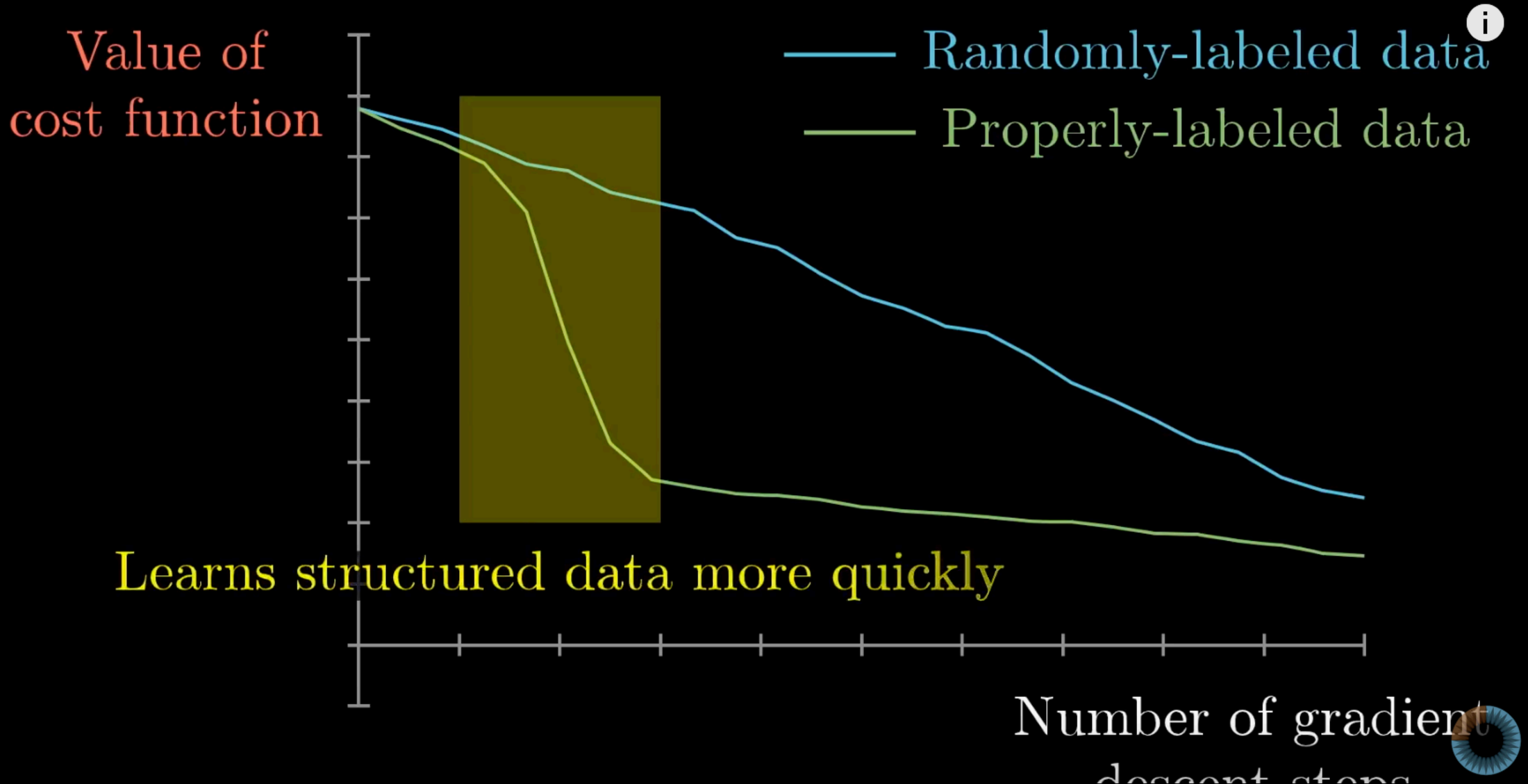
Devansh Arpit [*1 2] **Stanisław Jastrzębski** [*3] **Nicolas Ballas** [*1 2] **David Krueger** [*1 2] **Emmanuel Bengio** [4]
Maxinder S. Kanwal [5] **Tegan Maharaj** [1 6] **Asja Fischer** [7] **Aaron Courville** [1 2 8] **Yoshua Bengio** [1 2 9]
Simon Lacoste-Julien [1 2]

Abstract

We examine the role of memorization in deep learning, drawing connections to capacity, generalization, and adversarial robustness. While deep networks are capable of memorizing noise data, our results suggest that they tend to prioritize learning simple patterns first. In our experiments, we expose qualitative differences in gradient-based optimization of deep neural networks (DNNs) on noise vs. real data. We also demonstrate that for appropriately tuned explicit regularization (e.g., dropout) we can

From a representation learning perspective, the generalization capabilities of DNNs are believed to stem from their incorporation of good generic priors (see, e.g., Bengio et al. (2009)). Lin & Tegmark (2016) further suggest that the priors of deep learning are well suited to the physical world. But while the priors of deep learning may help explain why DNNs learn to efficiently represent complex real-world functions, they are not restrictive enough to rule out memorization.

On the contrary, deep nets are known to be universal approximators, capable of representing arbitrarily complex functions given sufficient capacity (Cybenko, 1989; Hornik


Value of
cost function
Randomly-labeled data
Properly-labeled data
Learns structured data more quickly
Number of gradient

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

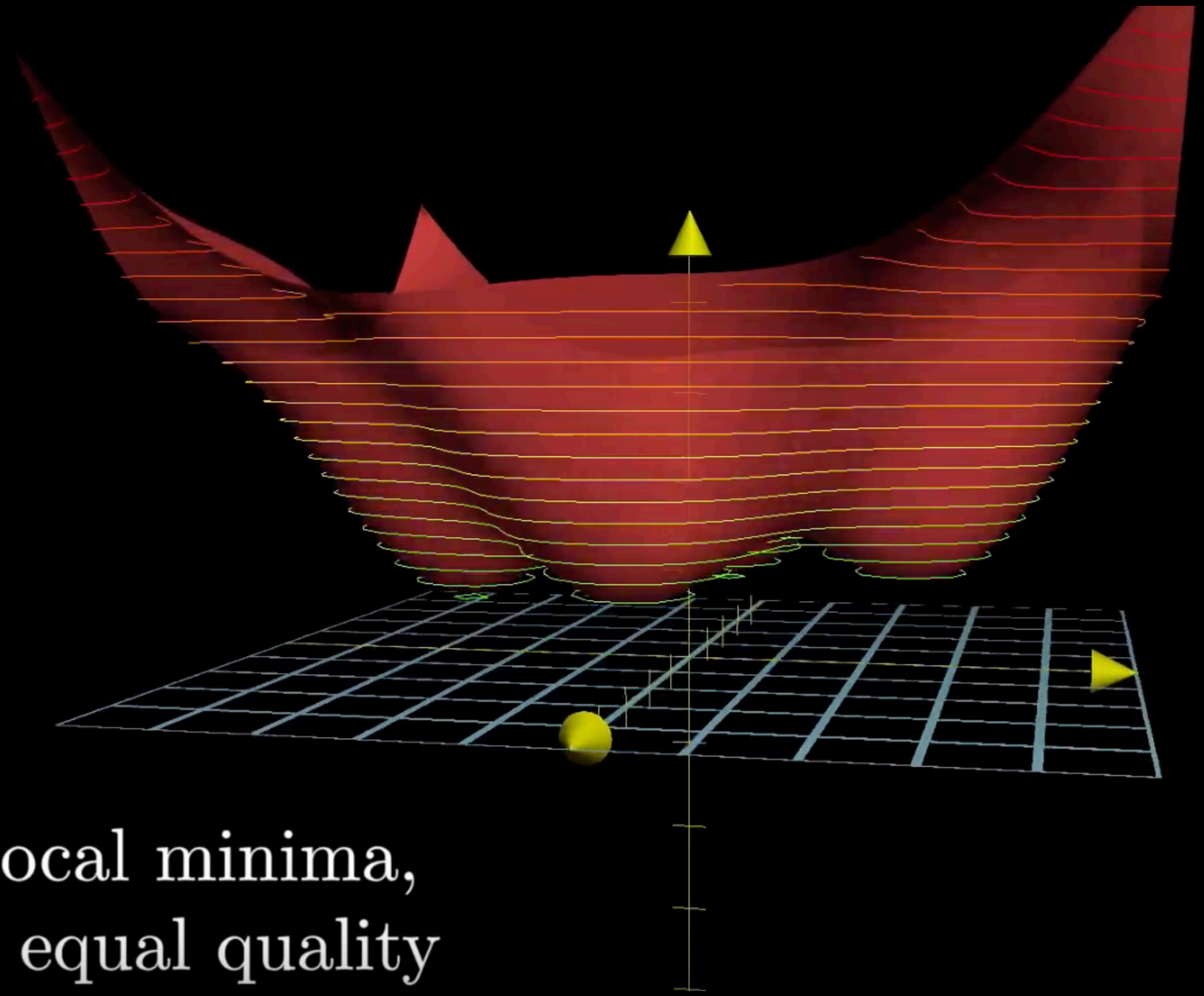
Courant Institute of Mathematical Sciences
New York, NY, USA

Abstract

We study the connection between the highly non-convex loss function of a simple model of the fully-connected feed-forward neural network and the Hamiltonian of the spherical spin-glass model under the assumptions of: i) variable independence, ii) redundancy in

1 Introduction

Deep learning methods have enjoyed a resurgence of interest in the last few years for such applications as image recognition [Krizhevsky et al., 2012], speech recognition [Hinton et al., 2012], and natural language processing [Weston et al., 2014]. Some of the most popular methods use multi-stage architectures composed of alternated layers of linear transformations and max function. In a particu-

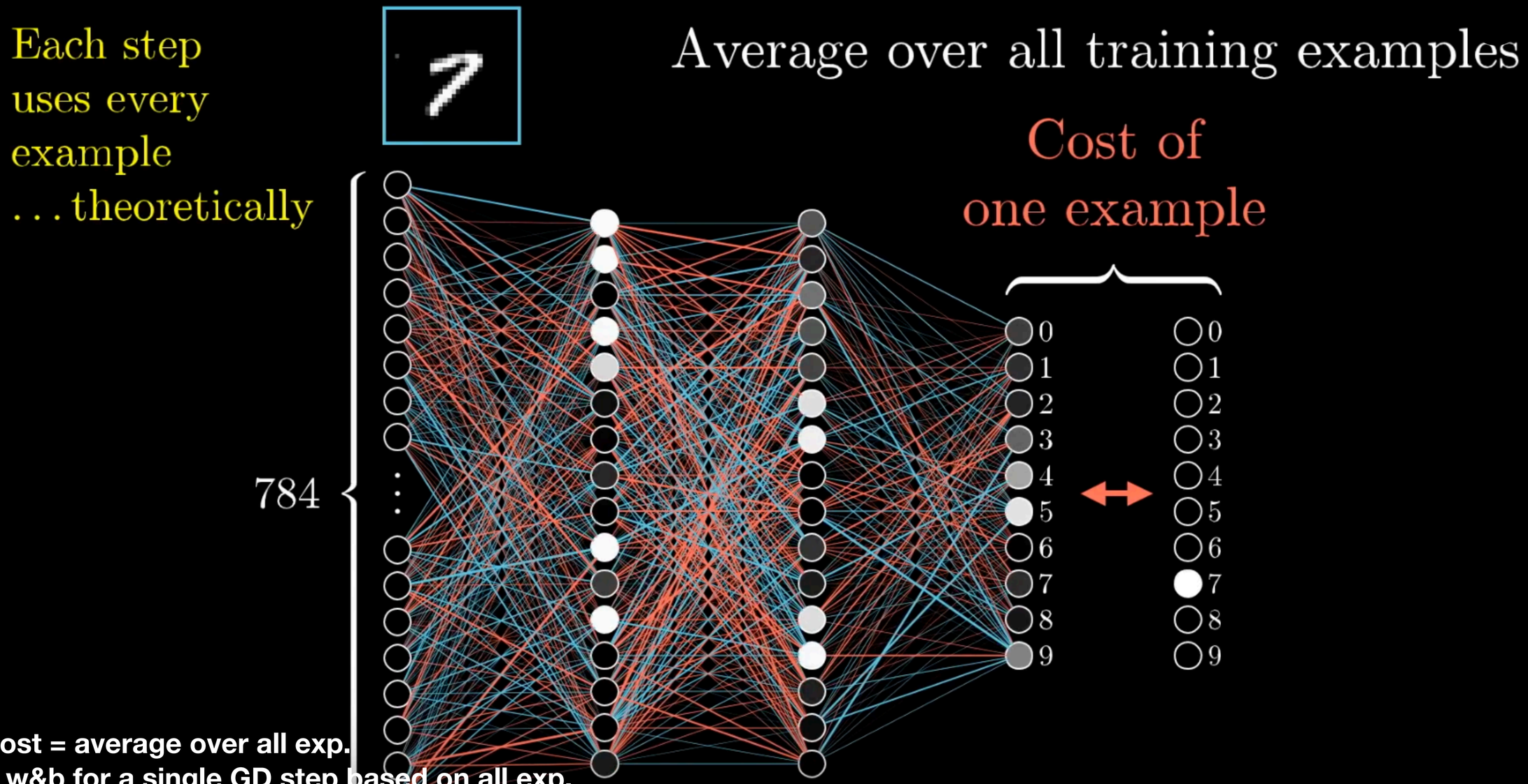


Many local minima,
roughly equal quality

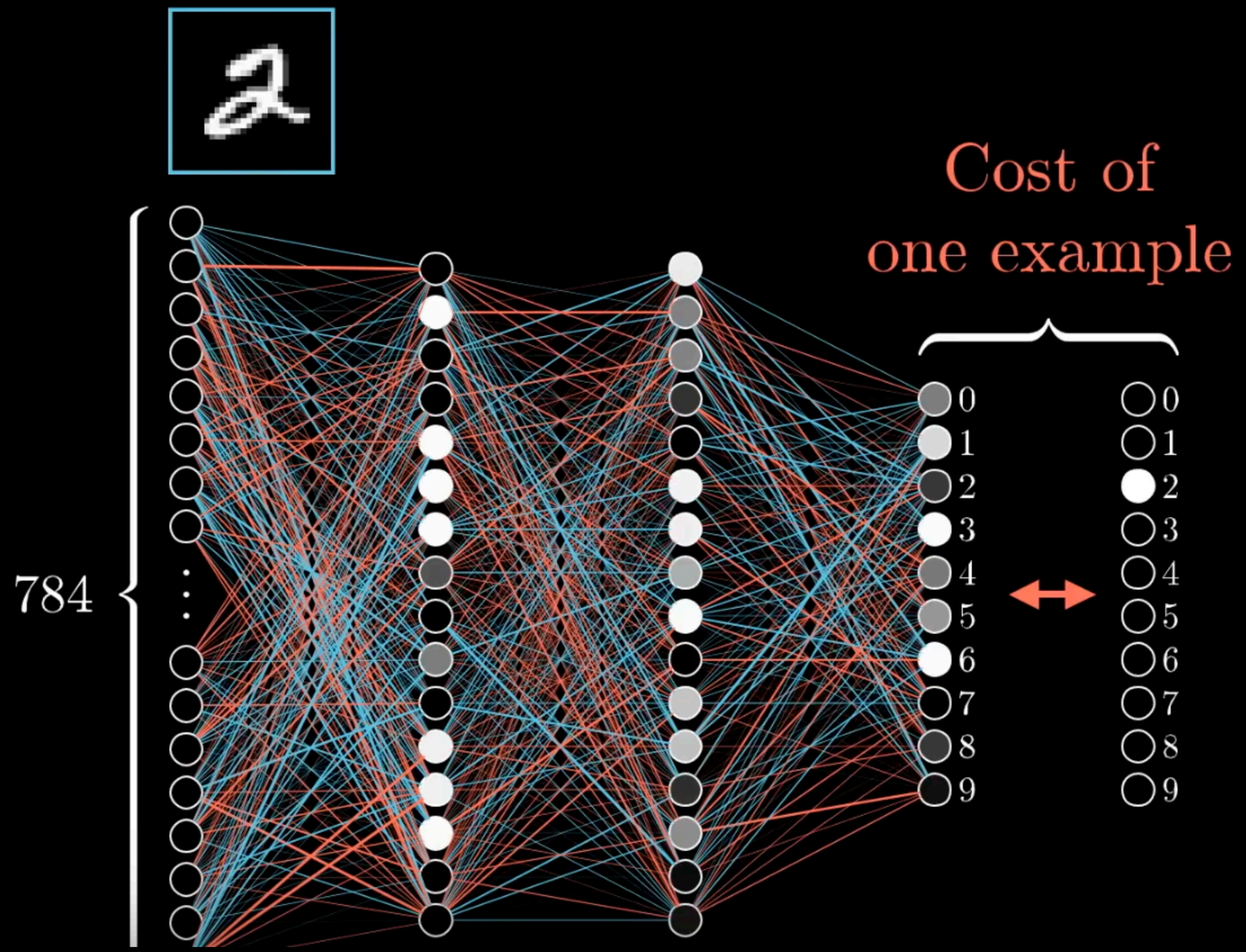
FF FCNN

Part 3: Backpropagation: **Intuition**

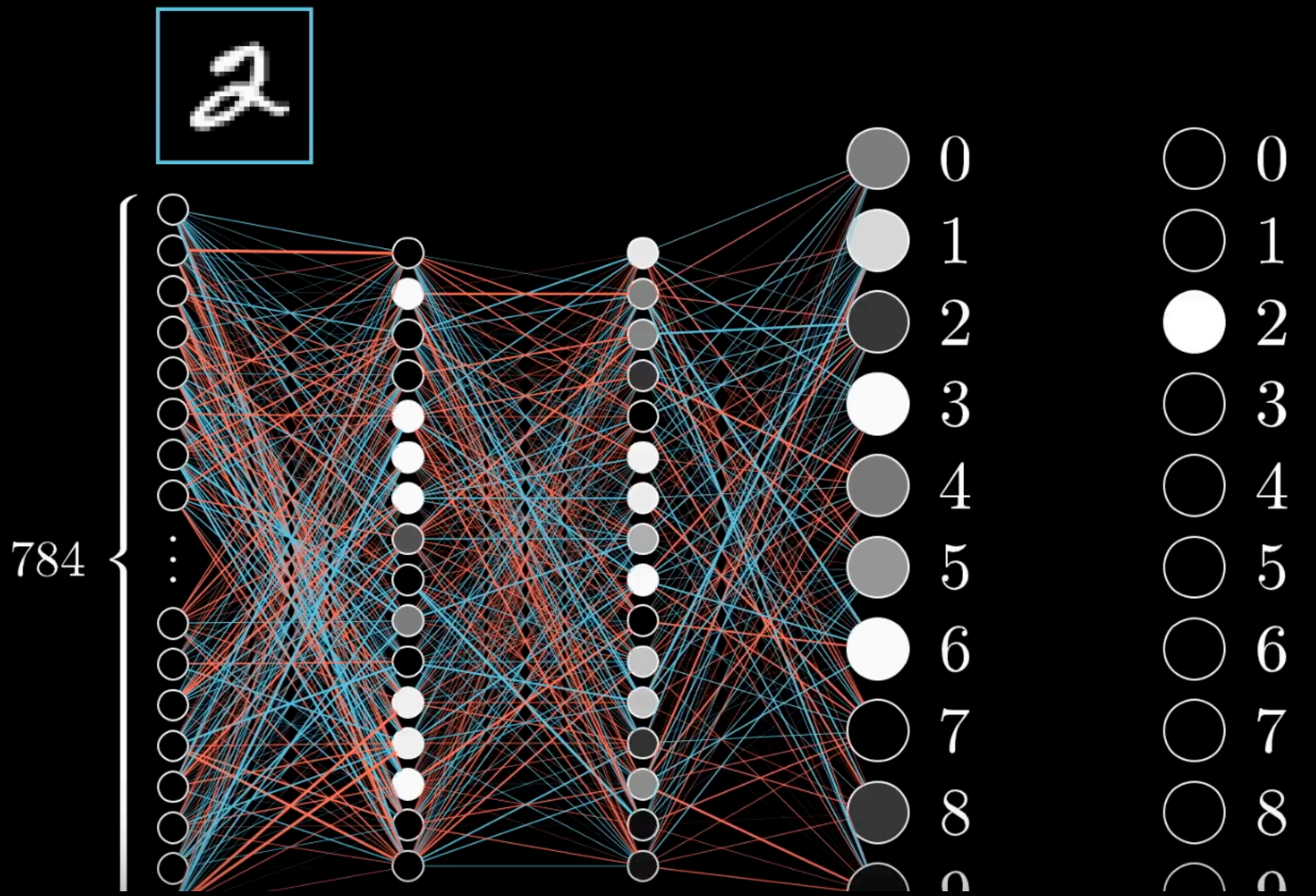
Intuition



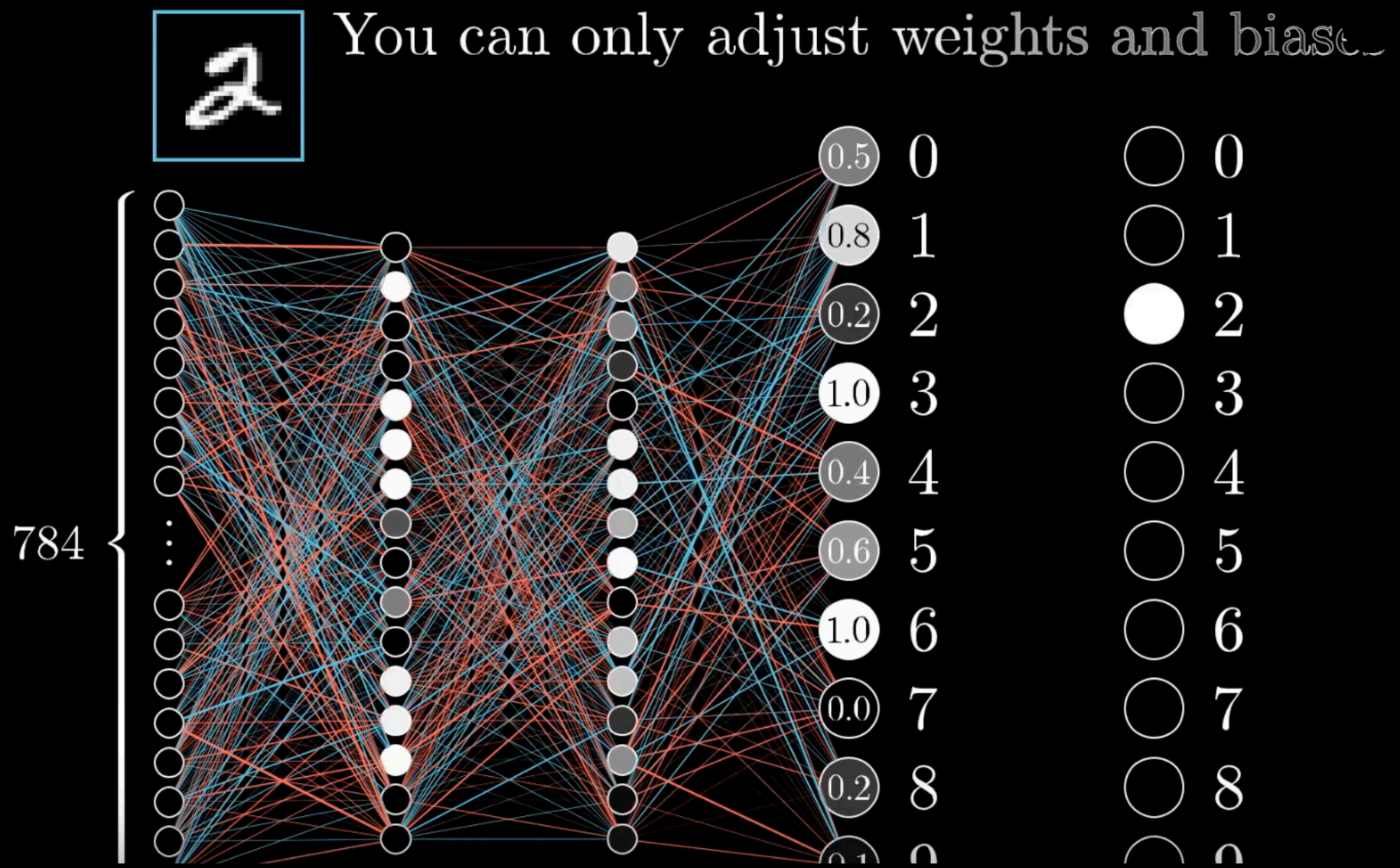
- total cost = average over all exp.
- adjust w&b for a single GD step based on all exp.
- practice: trick SGD



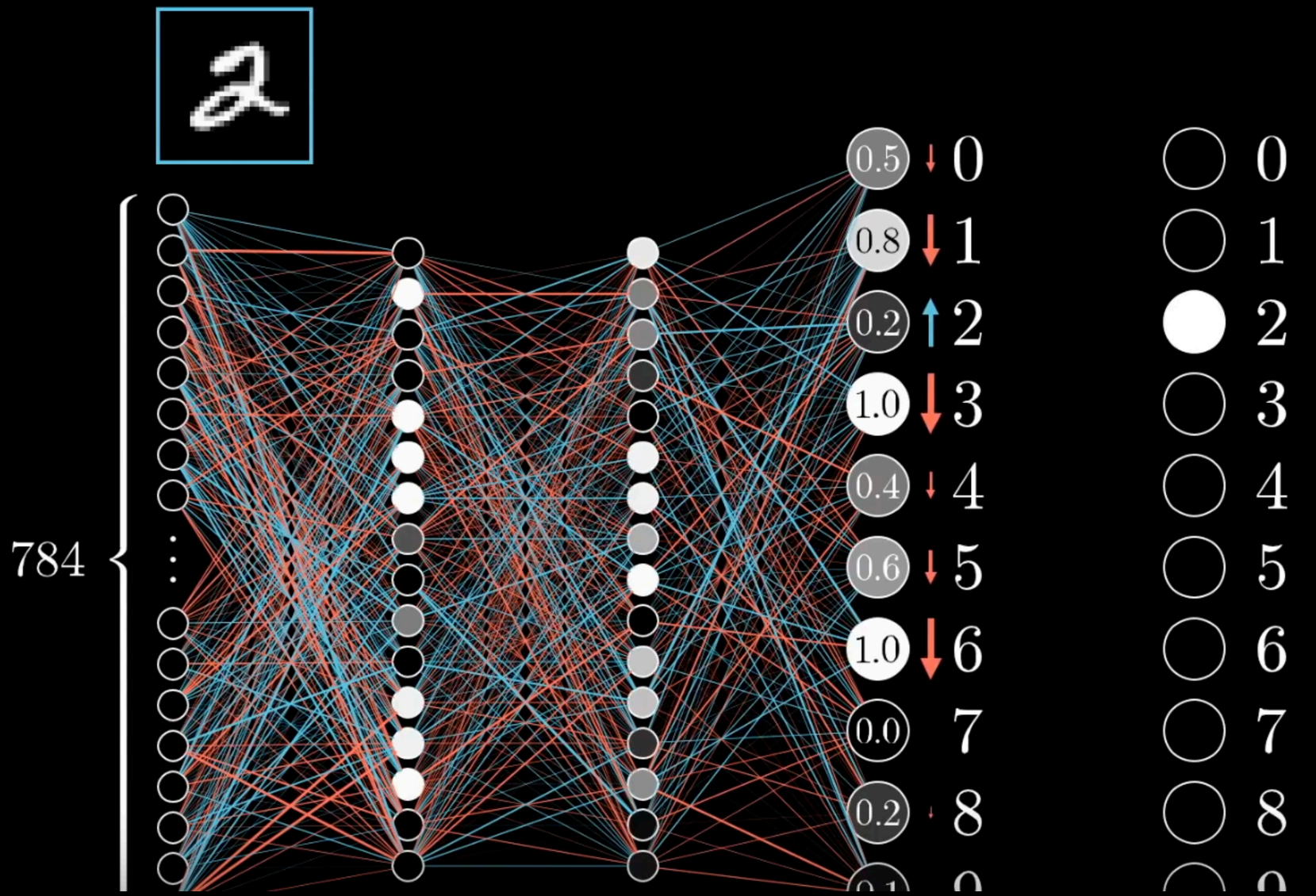
- image of a 2
- effect of 2 on w&b adjustments?



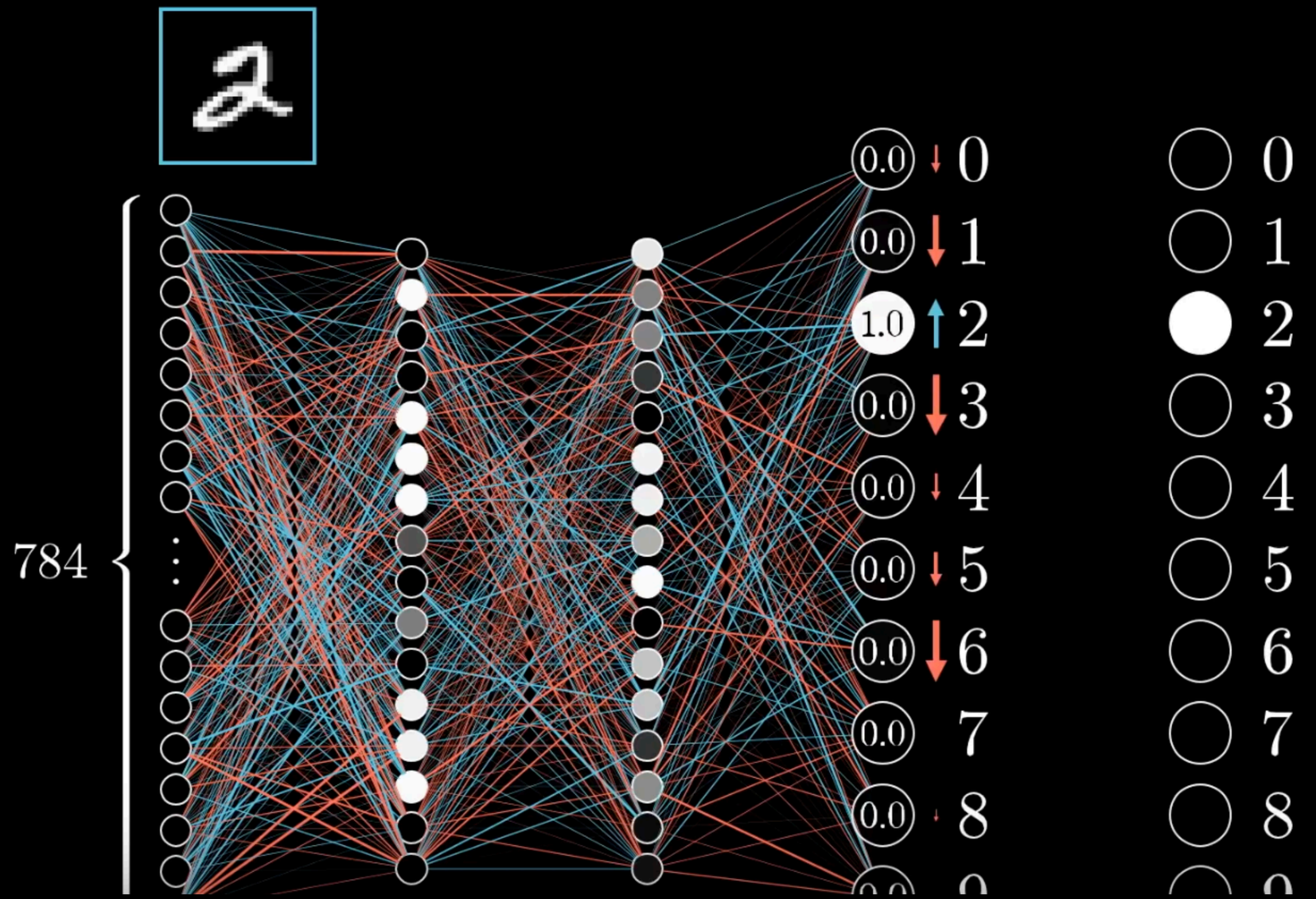
- NN not trained
- ouput activations random



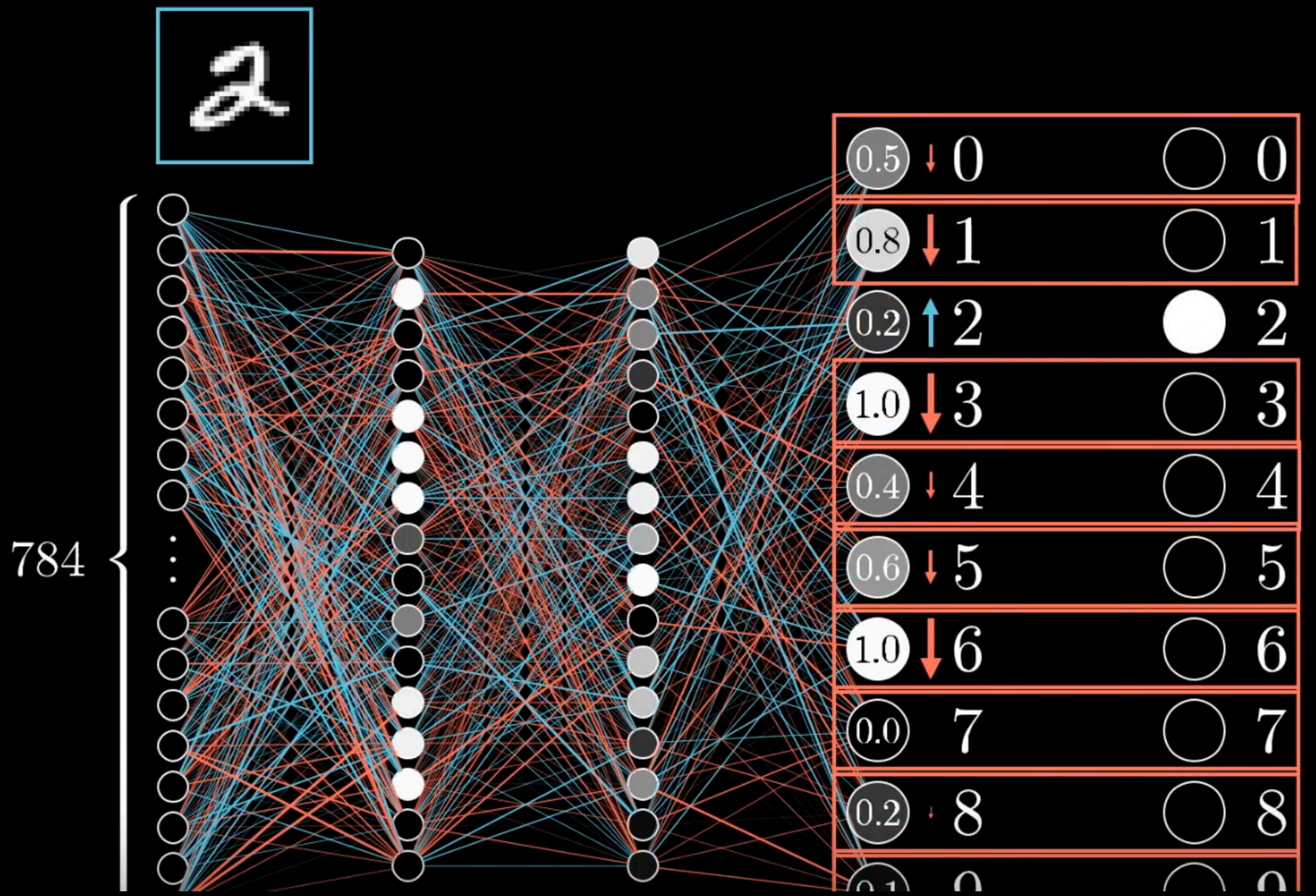
- e.g. 0.5, 0.8, 0.2 etc.
- can’t change activations, only w&b



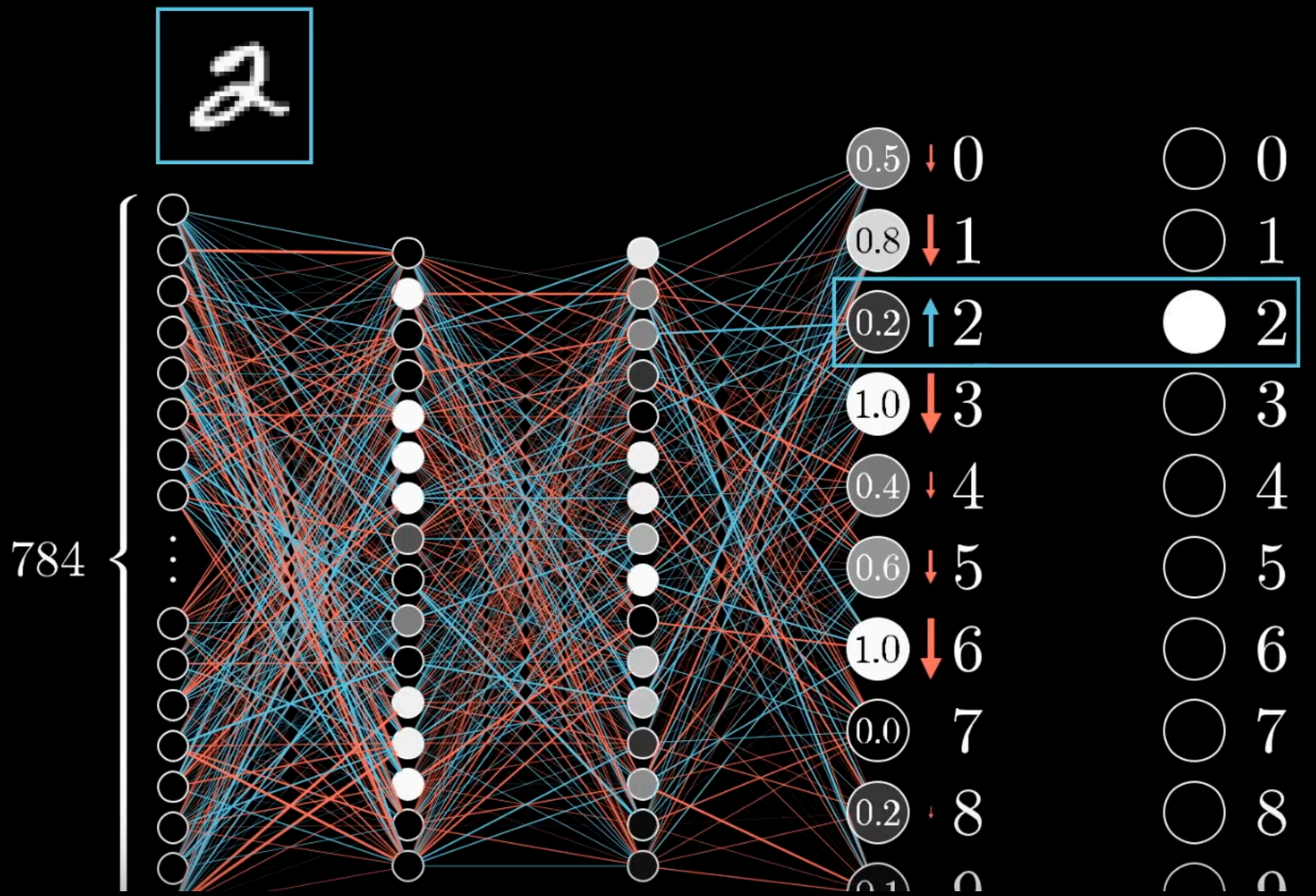
- adjustments we wish for
- third value should go up and rest down



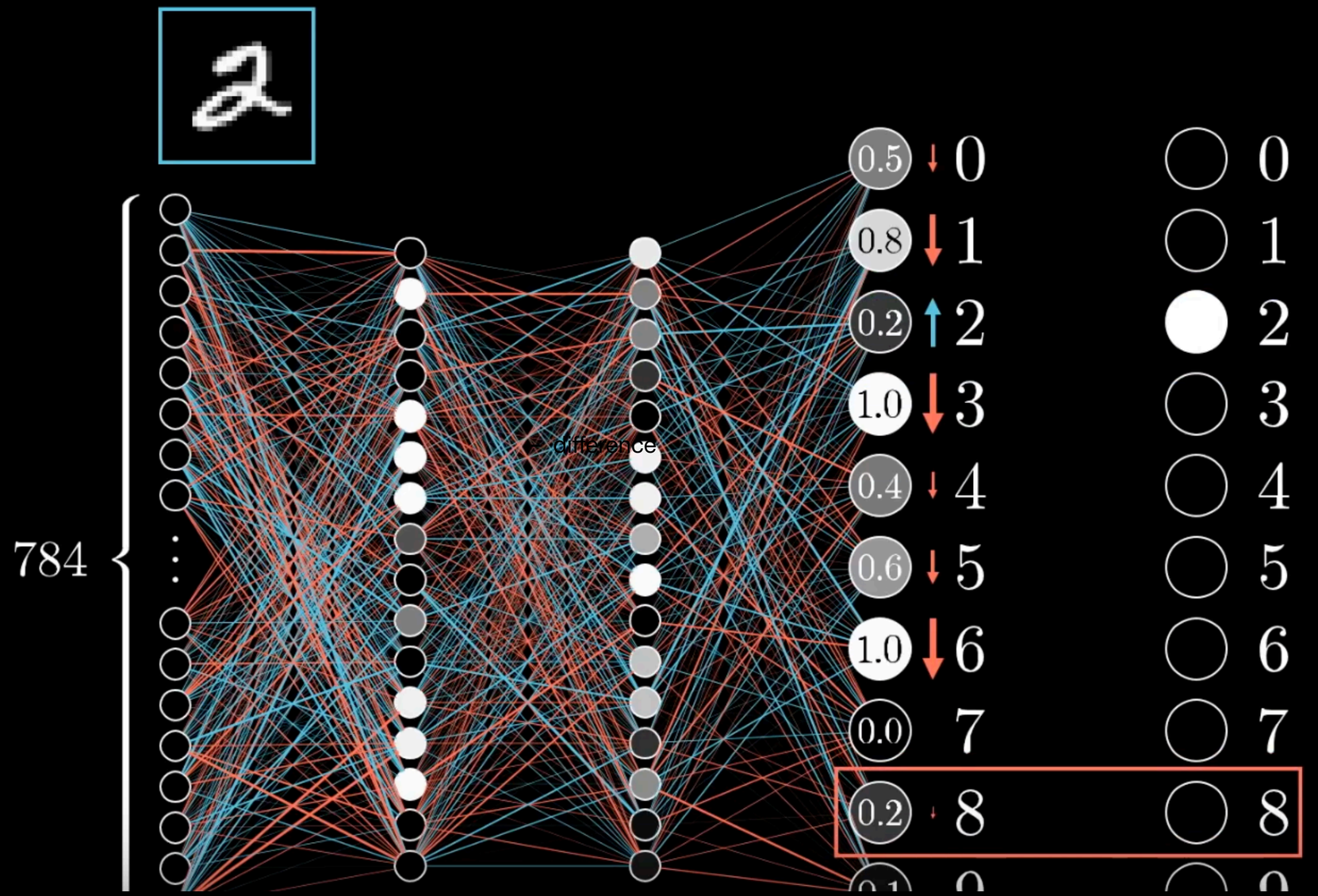
- trying to get something like this



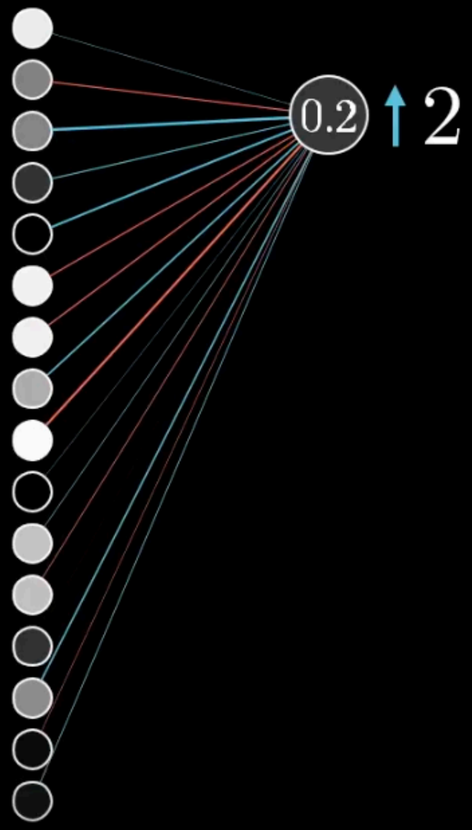
- in total this is needed:
- blue: up, red: down



- nudges: proportional to difference
- e.g. increase of the digit 2 neuron is more important than..

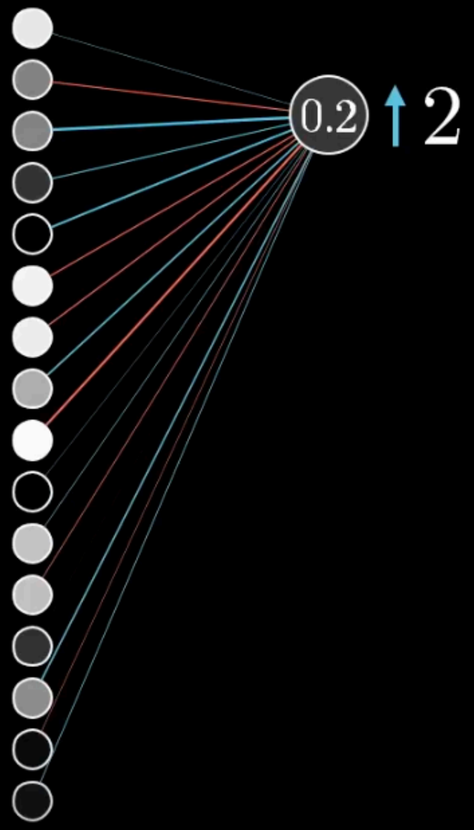


- **than the decrease to the digit 8 neuron (close already)**



- zooming in the digit 2 neuron
- want to increase the activation

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$



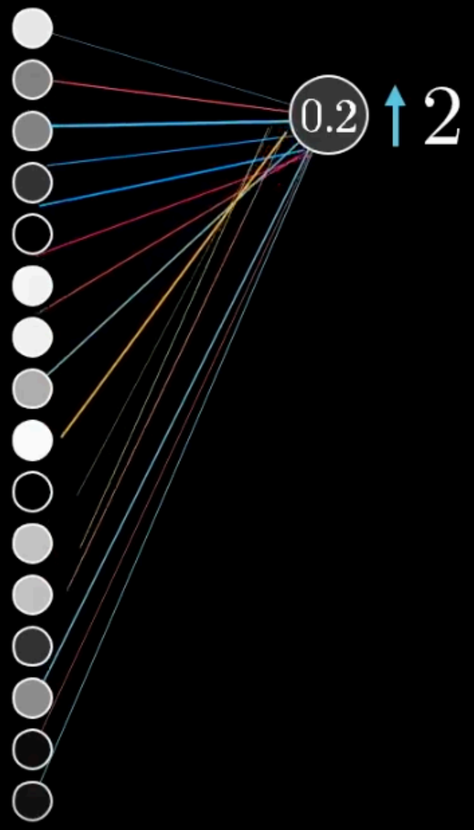
- activation = weighted sum of pre_a + bias (pumped into the af)

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i

Change a_i



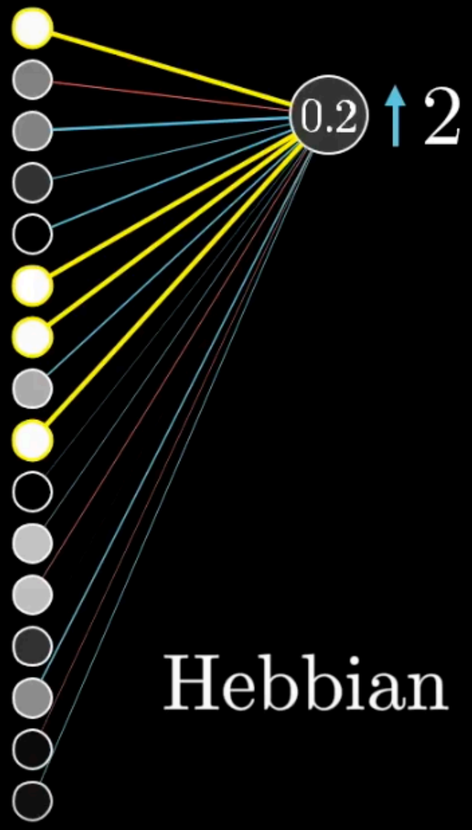
- so there things can help inc. the activation:
- change b, w and pre_a

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

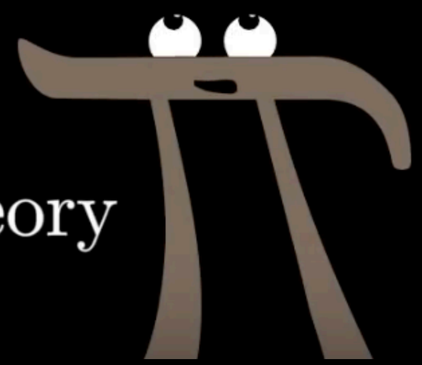
Increase w_i
in proportion to a_i

Change a_i



"Neurons that fire together wire together"

Hebbian theory



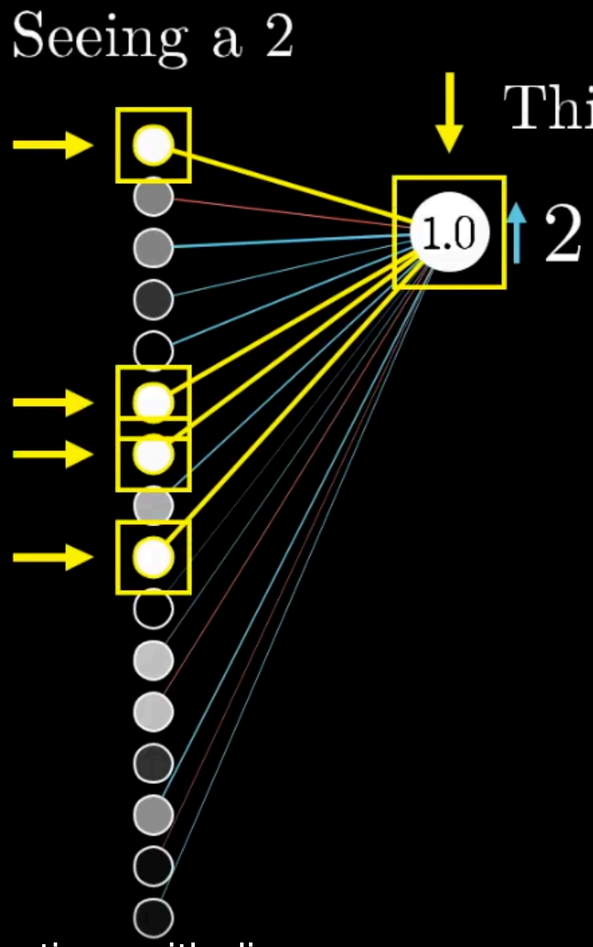
- focus on w
- w have different levels of influence, connections with bright neurons in pre-layer have biggest effect (multiplied by larger activations)

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i
in proportion to a_i

Change a_i



"Neurons that fire together wire together"

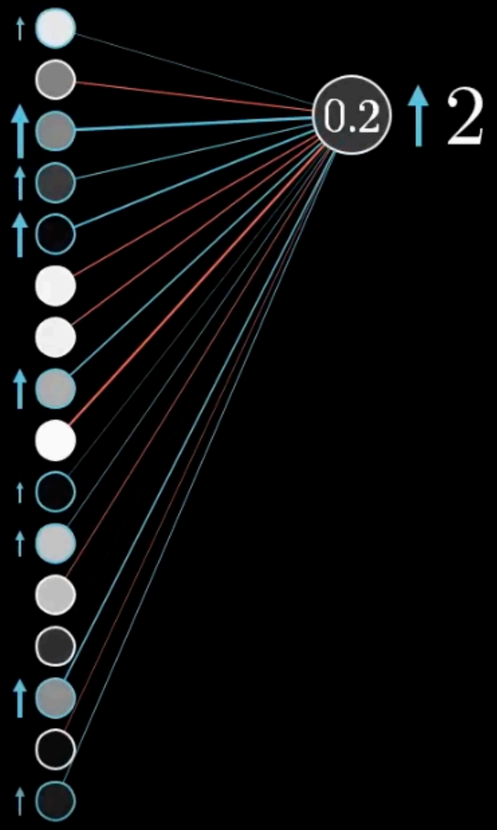
- increase in those w has a stronger influence on cost than connections with dimmer neurons
- give you the most bang for your buck
- related to the phrase « «neurons that fire together wire together» in neuroscience

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i
in proportion to a_i

Change a_i



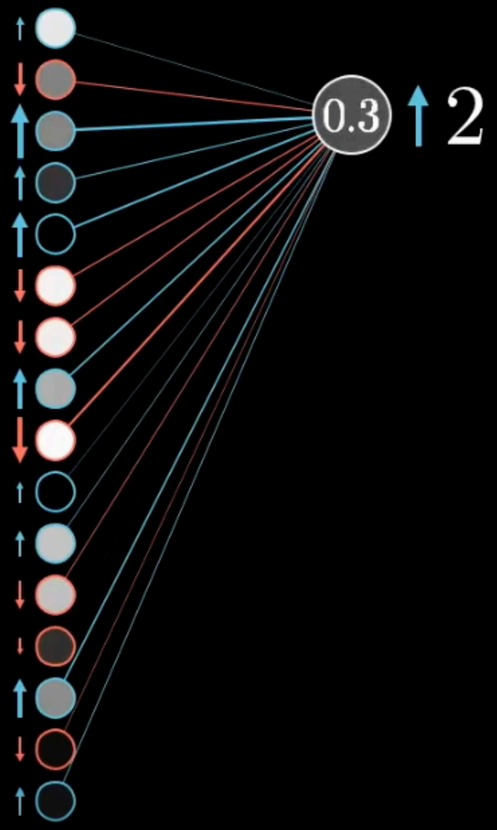
- third way to inc. activation is to change the activations in pre_layer
- i.e. inc. neurons connected with positive weight (blue)..

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i
in proportion to a_i

Change a_i



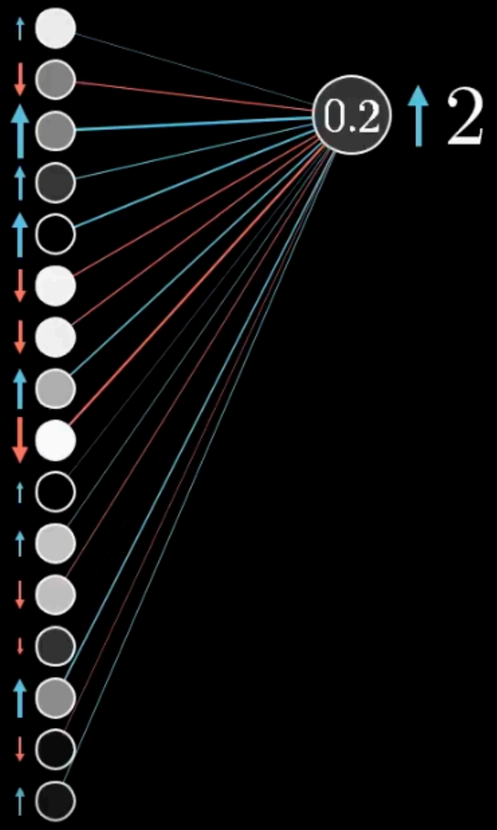
- and decrease neurons connected with negative weights (red)
- i.e. digit 2 neuron will become more active.

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i
in proportion to a_i

Change a_i
in proportion to w_i



- similar to w: most «bang for your buck» by changing the activations connected to **strong weights** (positive or negative)

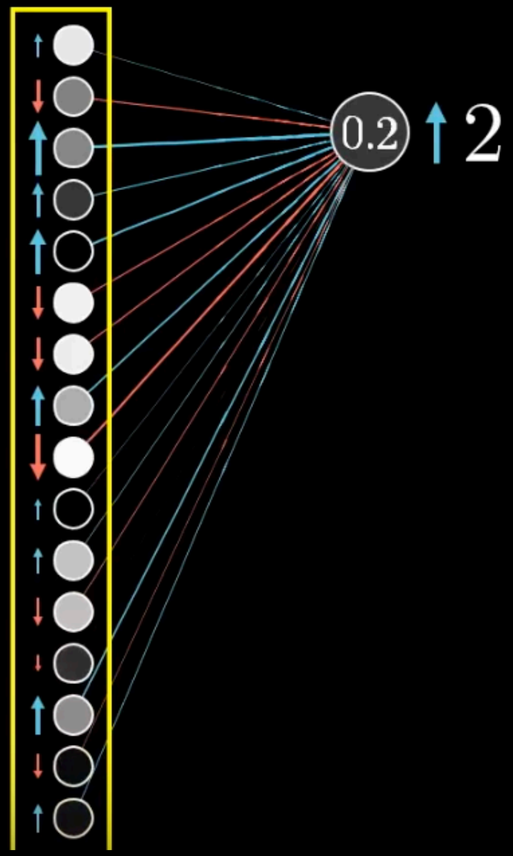
$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

No direct influence

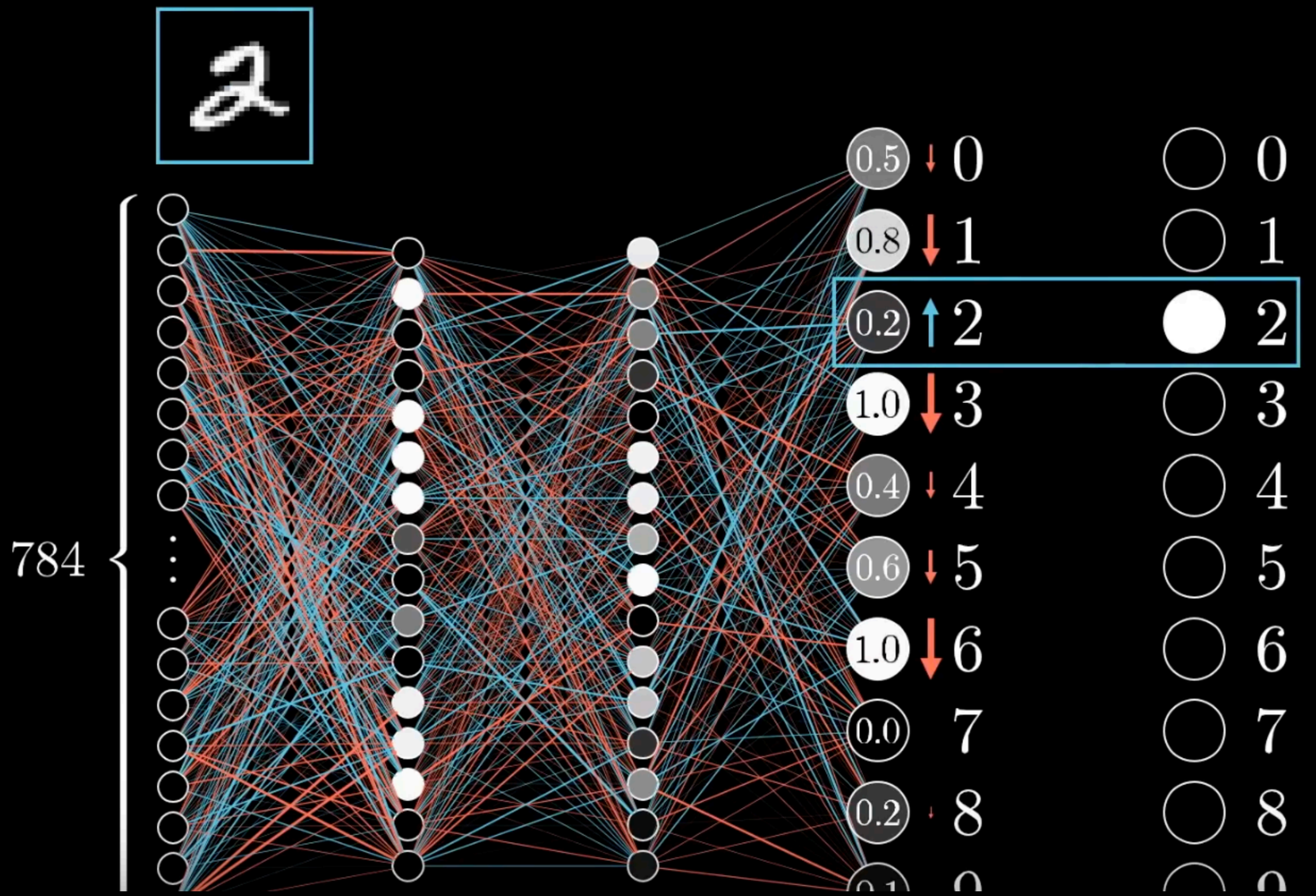
Increase b

Increase w_i
in proportion to a_i

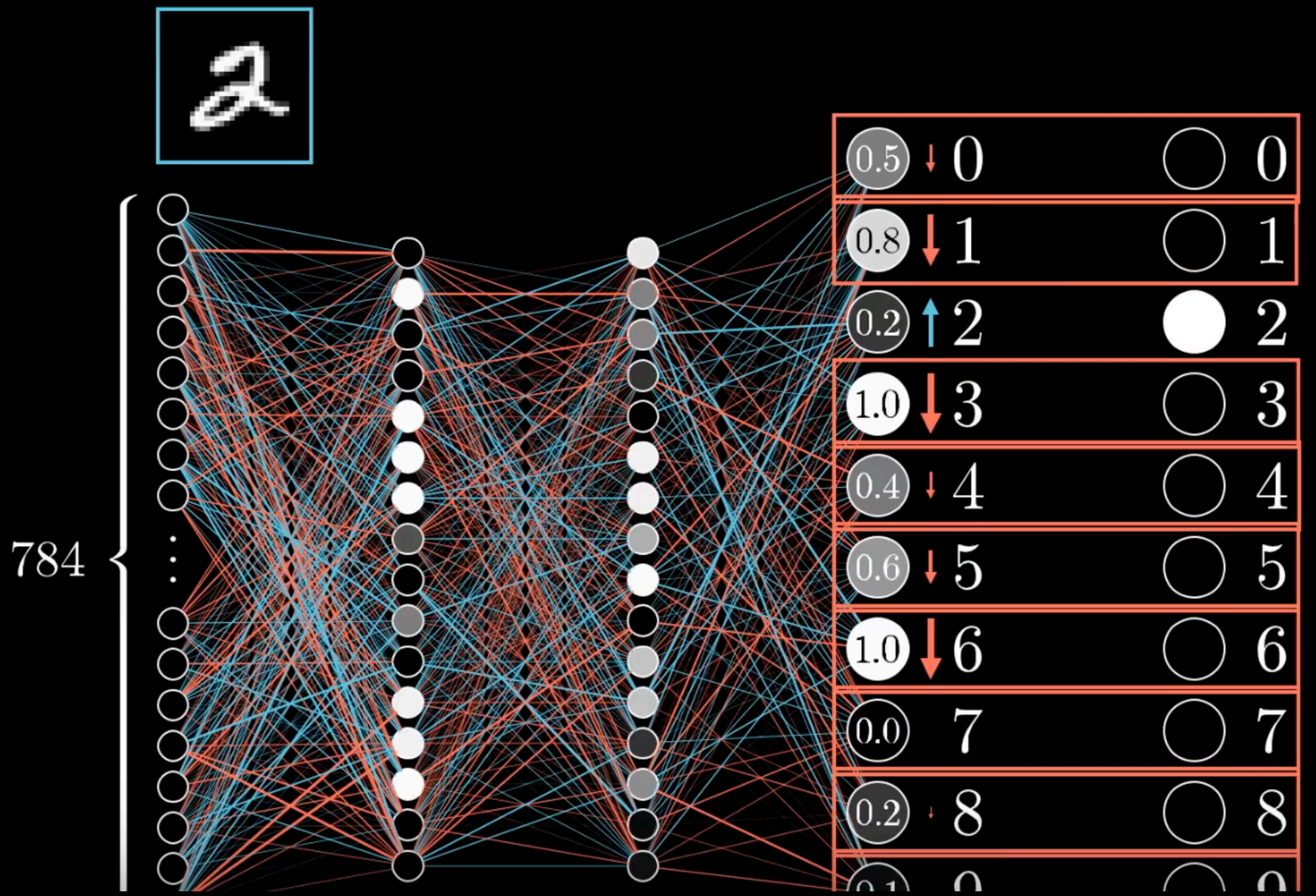
Change a_i
in proportion to w_i



- no influence on the activations, just w&b
- helpful to keep a note not the desired changes (as with the last layer)



- But zooming out, this is just what the digit-2 output neuron wants..

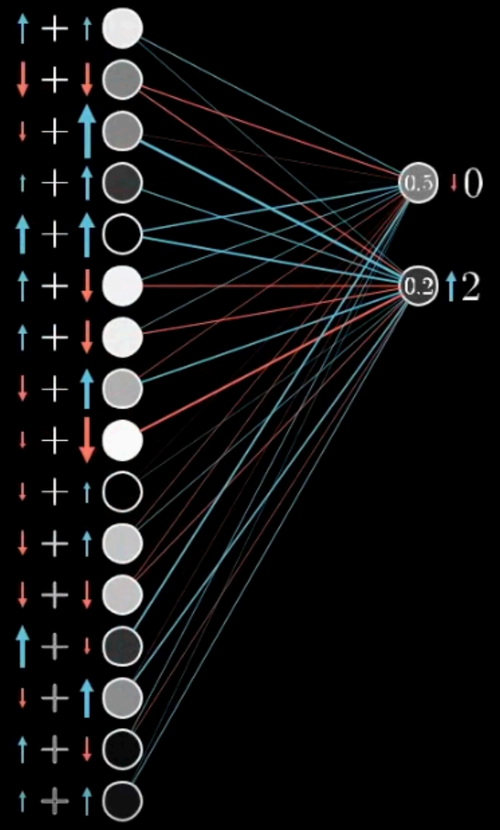


- We also want the other output neurons to become less active.

Increase b

Increase w_i
in proportion to a_i

Change a_i
in proportion to w_i

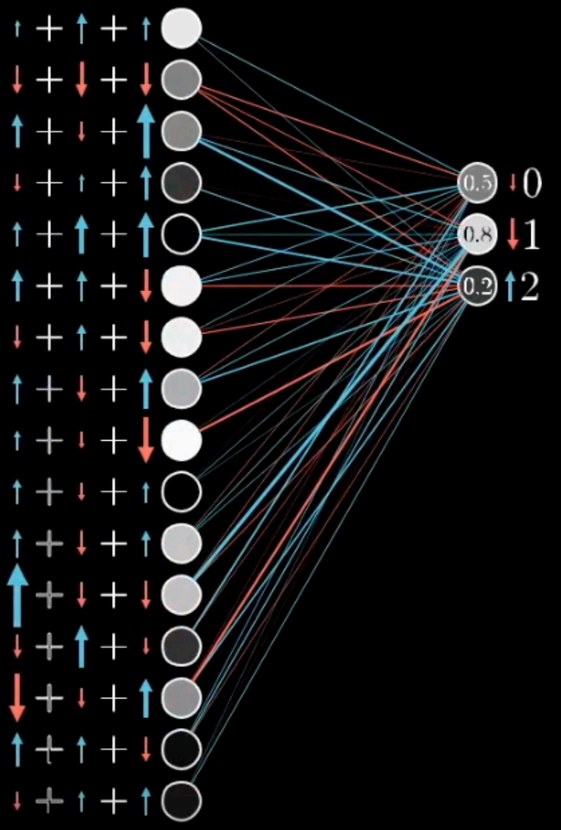


- and each of those output neurons has its own thoughts about should happen to the second-to-last layer

Increase b

Increase w_i
in proportion to a_i

Change a_i
in proportion to w_i



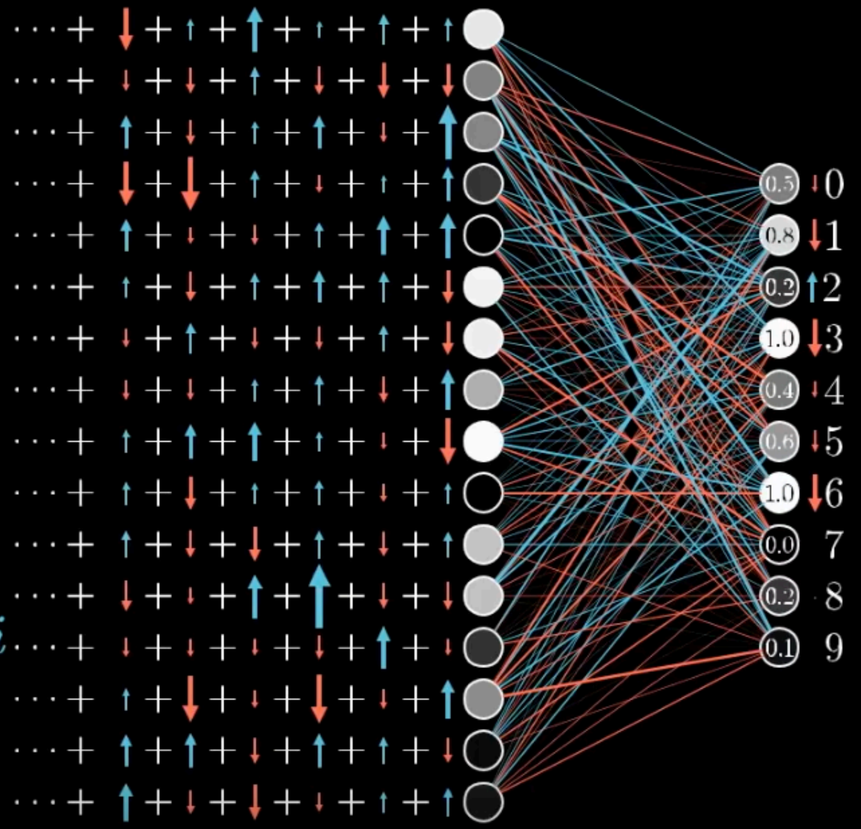
- to the desire of the digit 2 neuron..

Propagate backwards

Increase b

Increase w_i
in proportion to a_i

Change a_i
in proportion to w_i



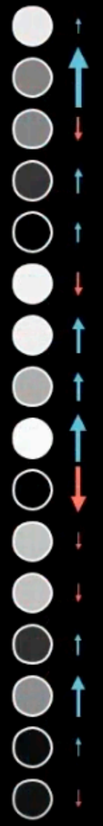
- has to be added with all the other desires
- in proportion to the corresponding weights, and in prop. to how much the last layer neurons needs to change.
- **propagating backwards idea**: all the desired effects must be added..

Propagate backwards

Increase b

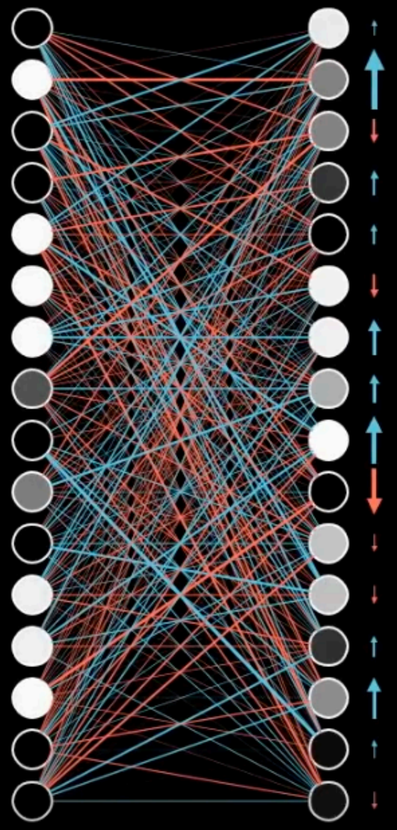
Increase w_i
in proportion to a_i

Change a_i
in proportion to w_i



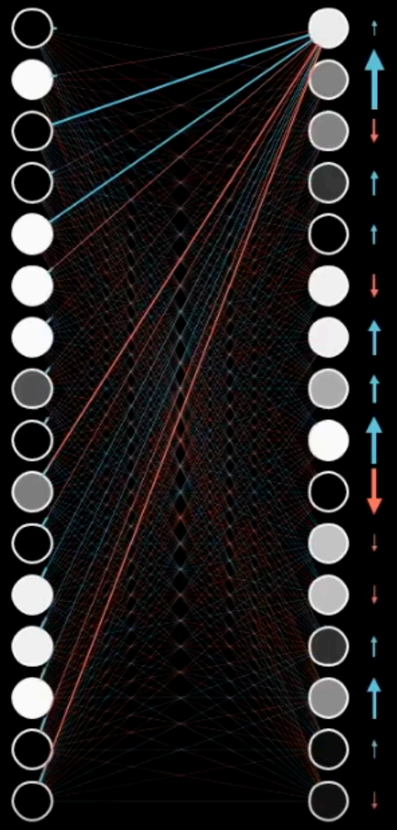
- doing that you basically get a **list of the nudges** that you want to **happen** to the second-to-last layer..

Propagate backwards



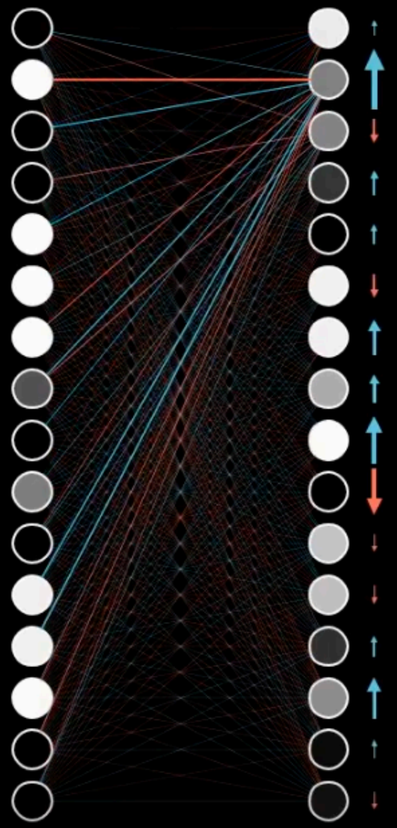
- and once you **have those** you can **recursively** apply the same process..

Propagate backwards



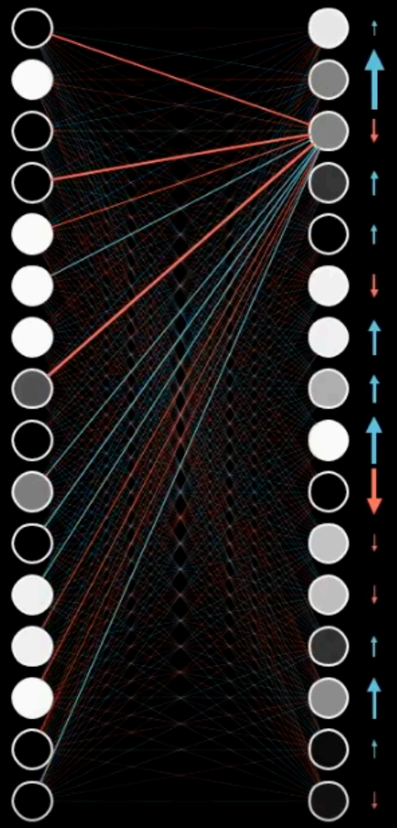
- to the relevant w&b that determine those values..

Propagate backwards

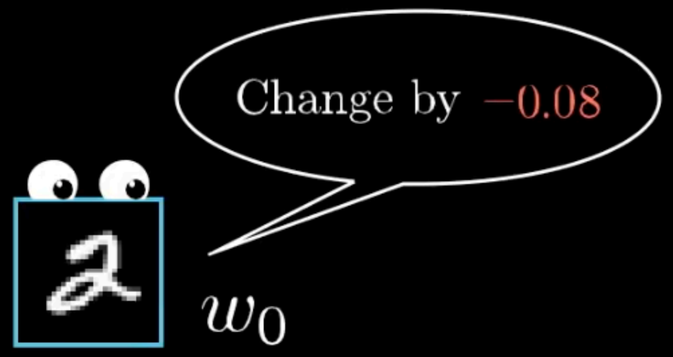


- **repeating** the same **process** we just walked through..

Propagate backwards



- and moving backwards through the network.



w_1

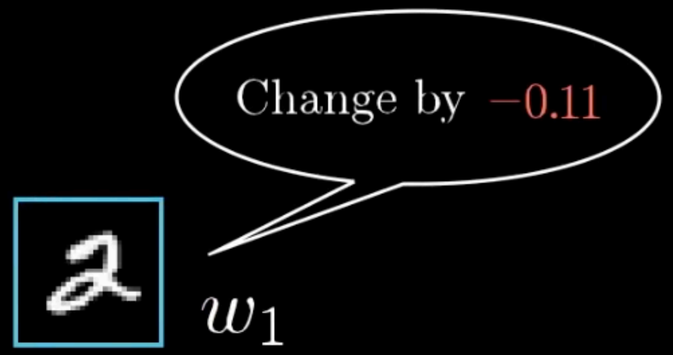
w_2

$\vdots$

$w_{13,001}$

} All weights and biases

- **zooming** out further, this is just how a **single** training example..



w_0 -0.08

w_2

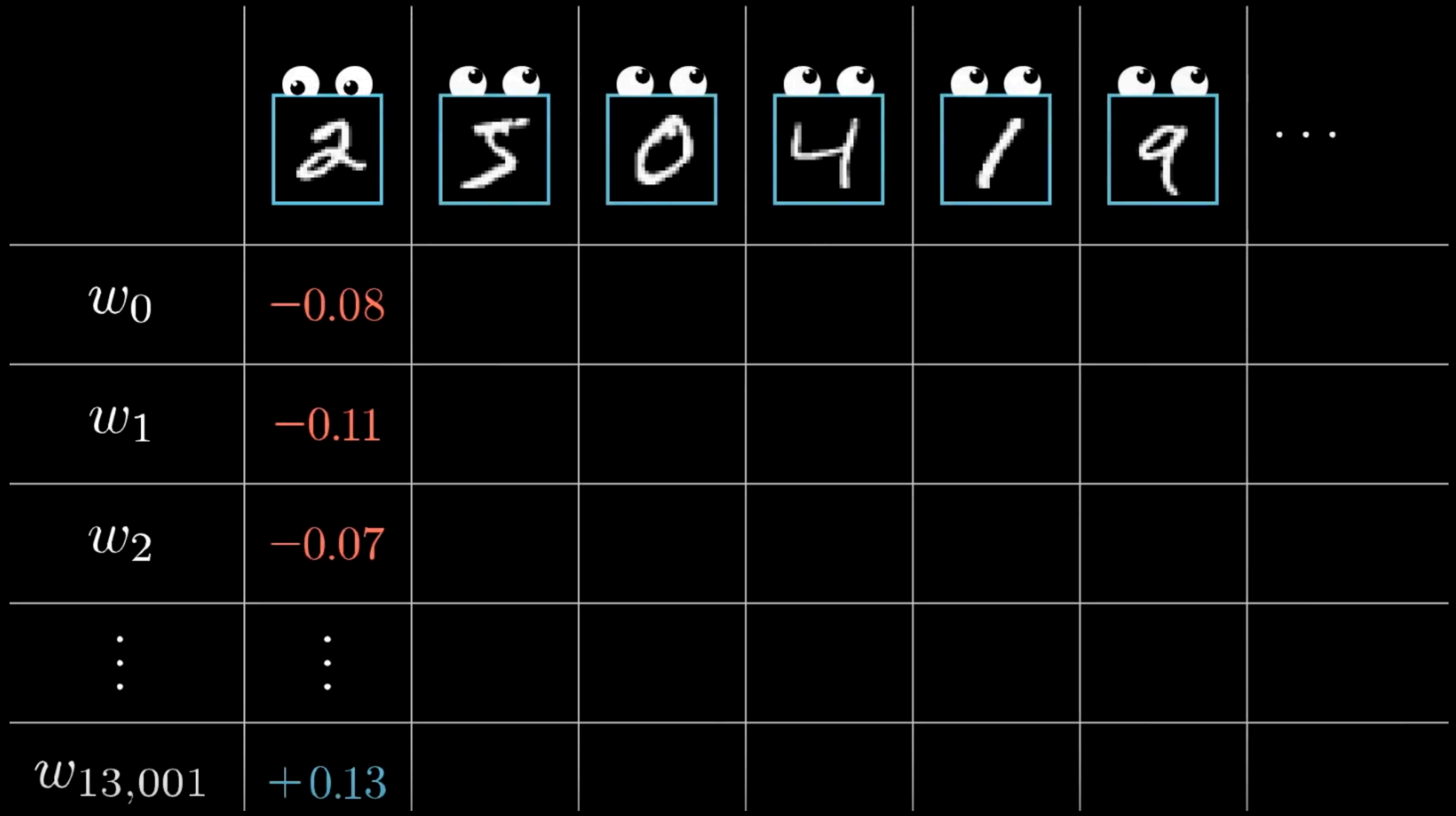
$\vdots$

$w_{13,001}$

- wishes to nudge each of the w&b

$$
\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{13,001} \end{bmatrix}
\begin{bmatrix} -0.08 \\ -0.11 \\ -0.07 \\ \vdots \\ +0.13 \end{bmatrix}
$$

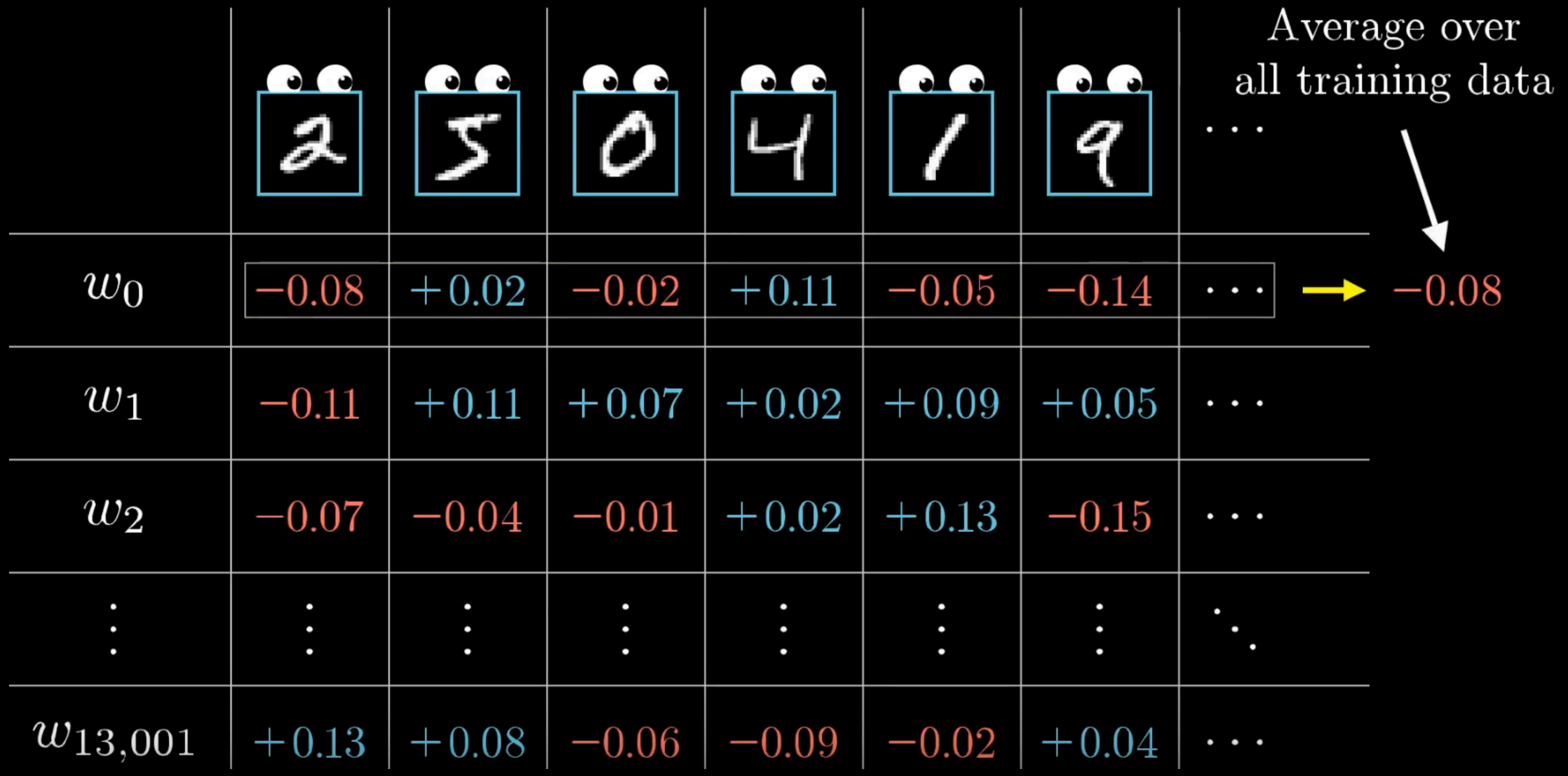
- If we only listen to what that 2 wanted the NN would ultimately be trained to classify all images as 2..



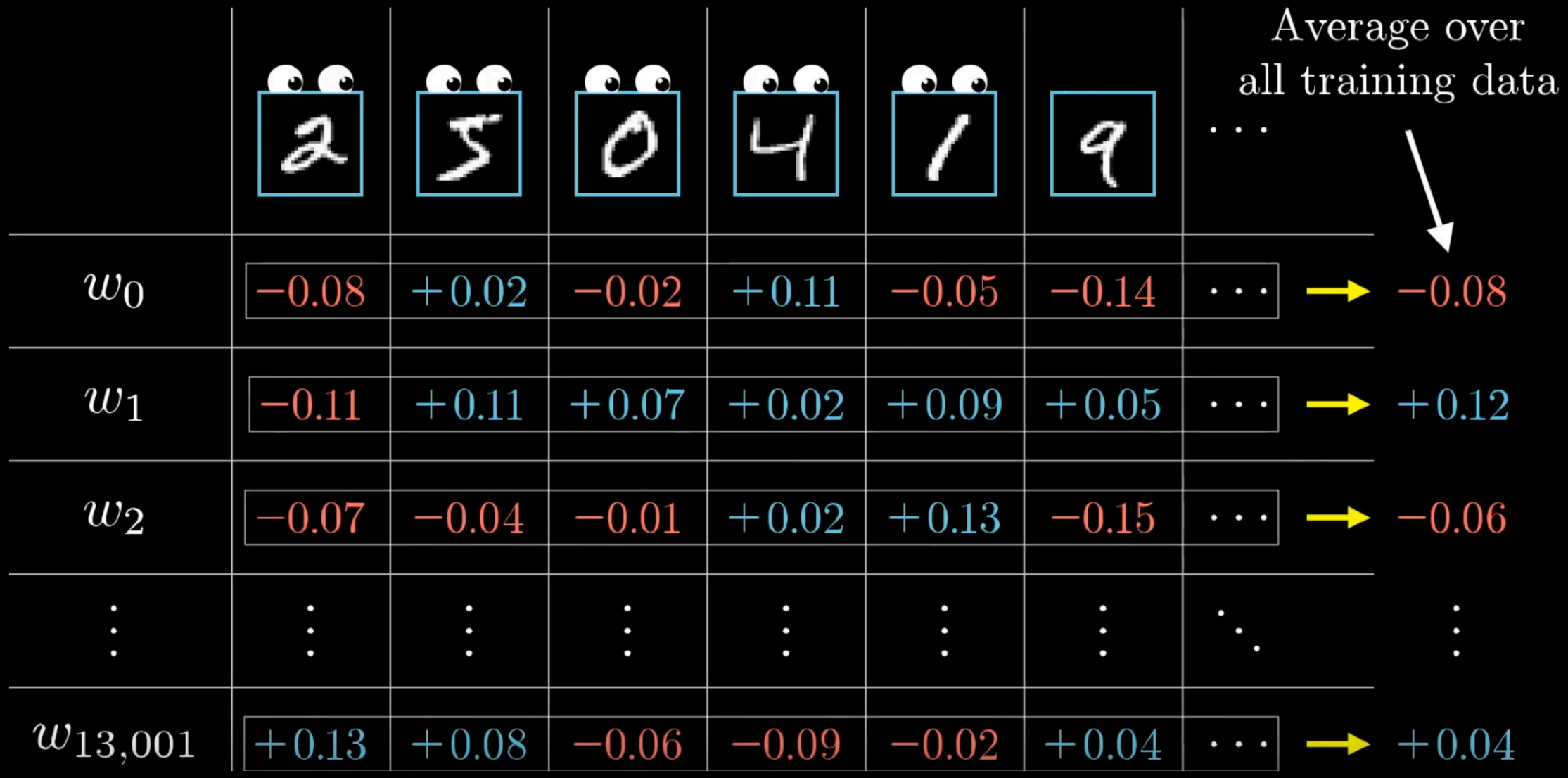
- so we have to go trough the same backdrop. rutine..

							$\cdots$
w_0	-0.08	$+0.02$	-0.02	$+0.11$	-0.05	-0.14	$\cdots$
w_1	-0.11	$+0.11$	$+0.07$	$+0.02$	$+0.09$	$+0.05$	$\cdots$
w_2	-0.07	-0.04	-0.01	$+0.02$	$+0.13$	-0.15	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
$w_{13,001}$	$+0.13$	$+0.08$	-0.06	-0.09	-0.02	$+0.04$	$\cdots$

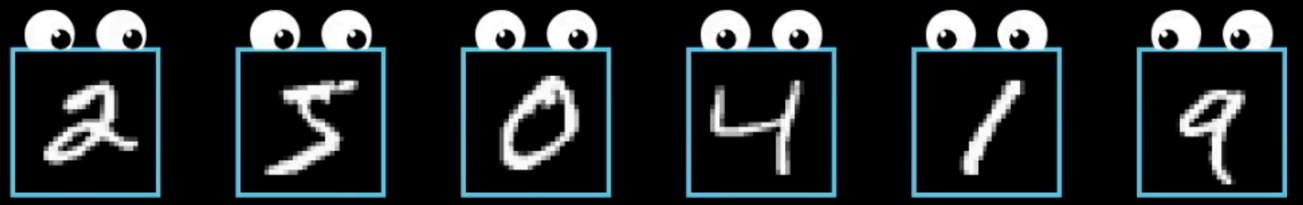
- for every training example, recording how each would like to change the w&b



- and you then average all the desired changes..



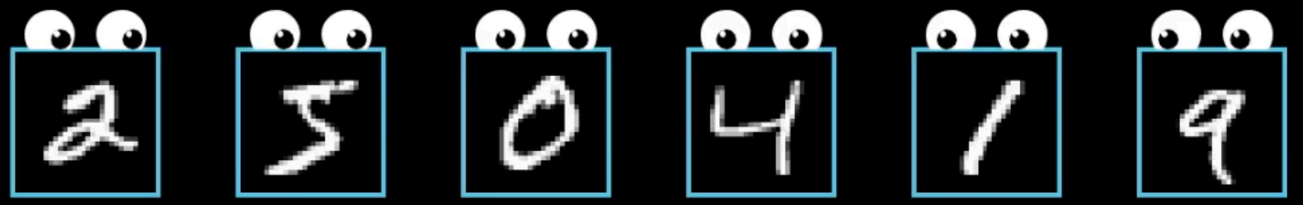
- to get a vector of average nudges to each w&b..



Average over all training data

$$-\nabla C(w_1, w_2, \ldots w_{13{,}001}) = \begin{bmatrix} -0.08 \\ +0.12 \\ -0.06 \\ \vdots \\ +0.04 \end{bmatrix}$$

- which is loosely speaking our desired negative gradient of the cost..



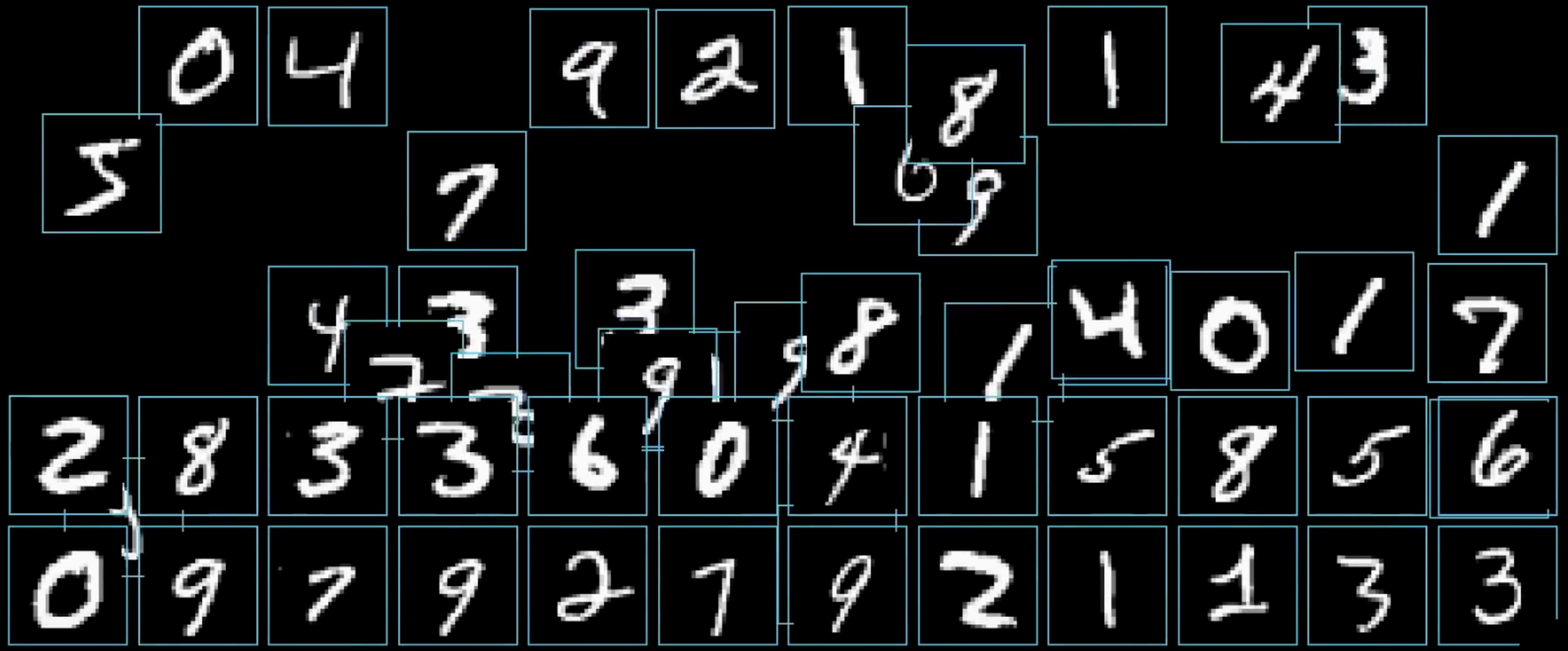
Average over all training data

$$-\eta \nabla C(w_1, w_2, \ldots w_{13{,}001}) = \begin{bmatrix} -0.08 \\ +0.12 \\ -0.06 \\ \vdots \\ +0.04 \end{bmatrix}$$

- or at least something proportional..

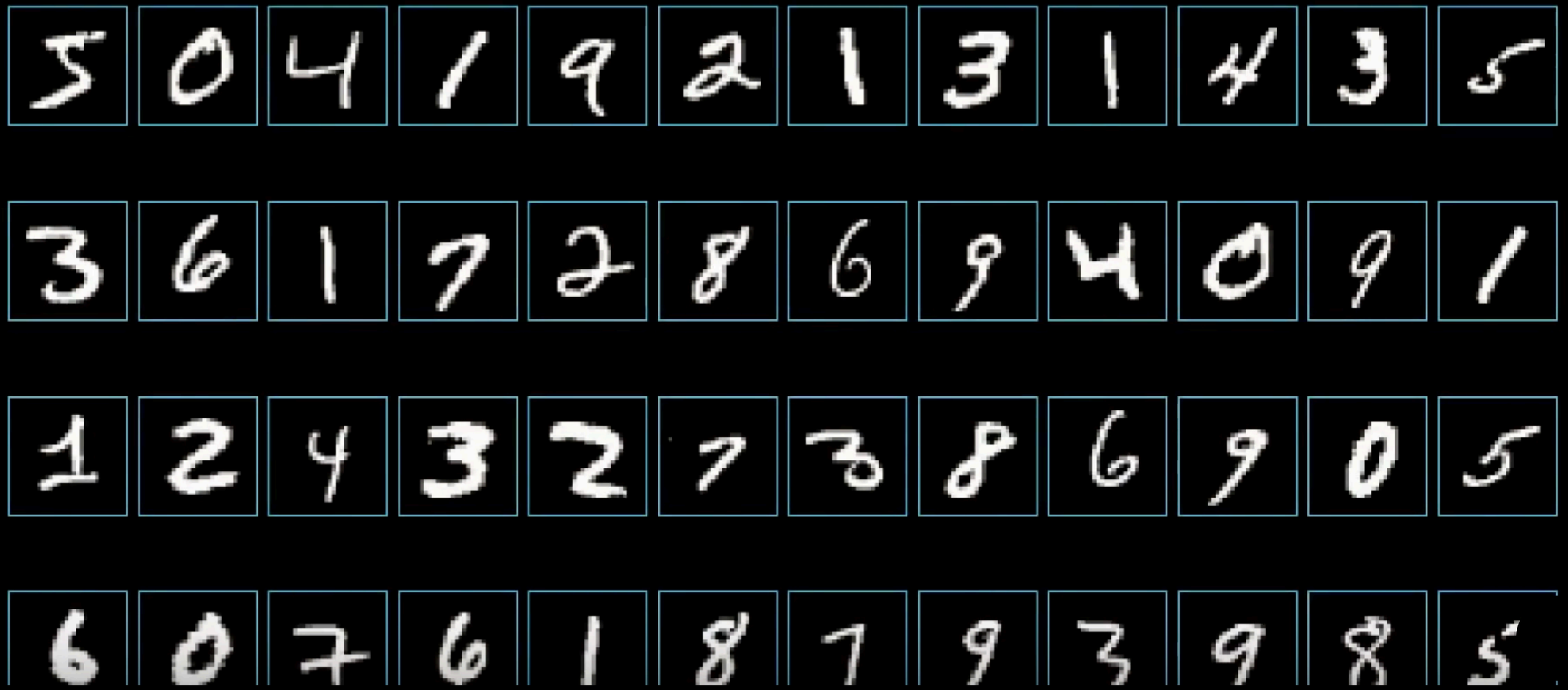
SGD

- computational **demanding**: add desired **changes** to every **example** for every gradient decent **step**..
- commonly use something called **stochastic or batch** gradient decent.

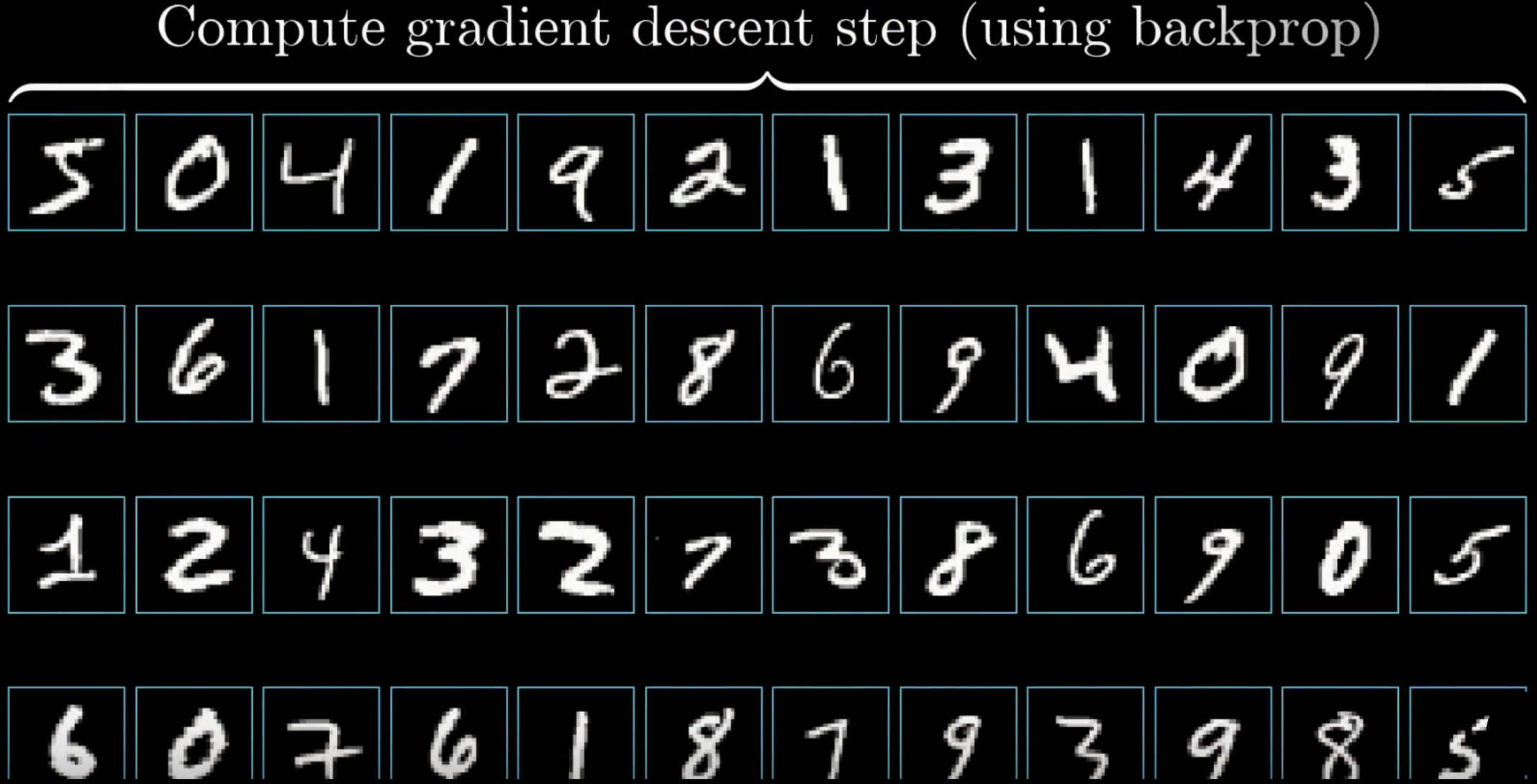


- think about this as to randomly shuffle your training data..

“Mini-batches”

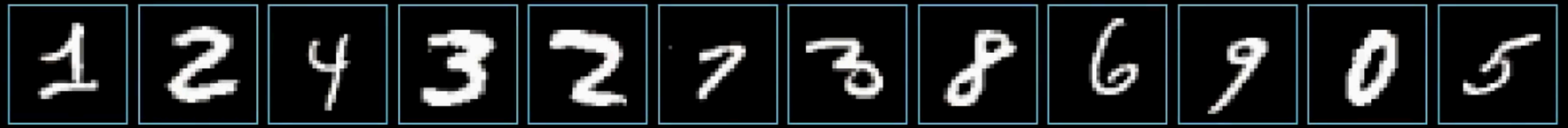


- and divide the it into many **mini-batches** (e.g. each having 100 training examples)

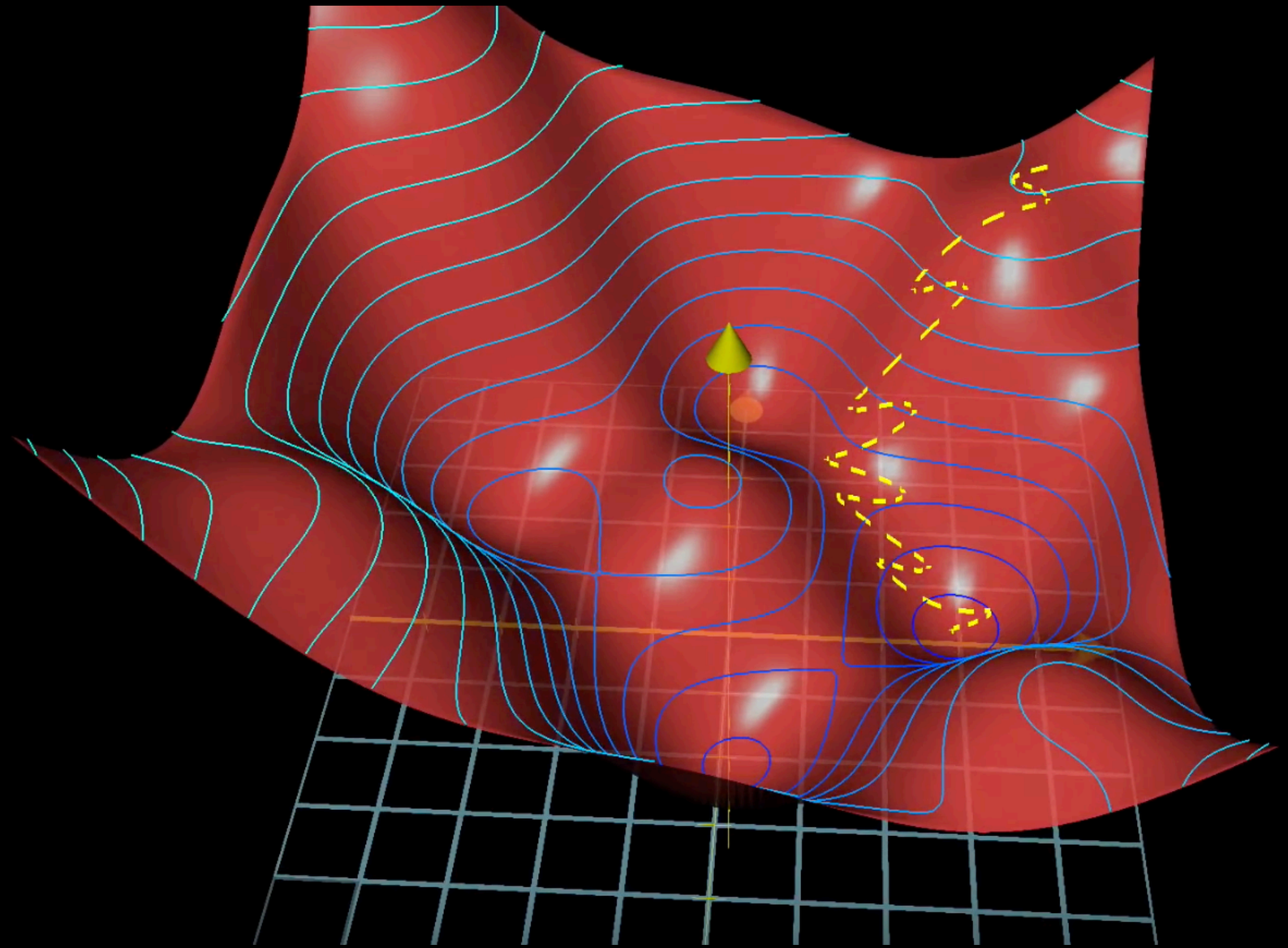


- you can then compute the GD step according to the mini-batches

Compute gradient descent step (using backprop)



- this will **not** be the **exact** gradient of the cost (need all training data), but a **good approx**. and significant **speed up**.

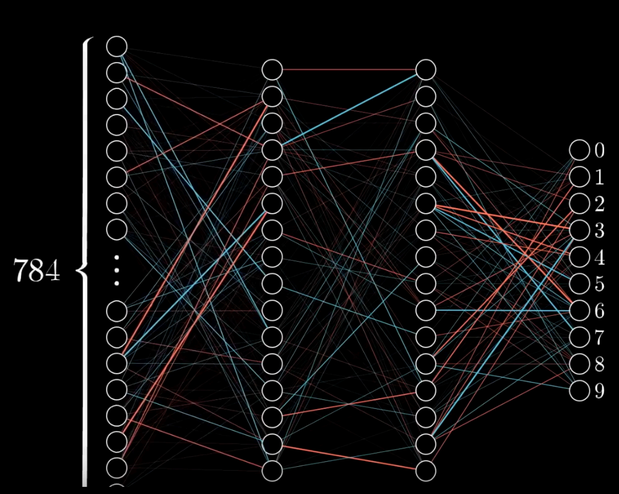


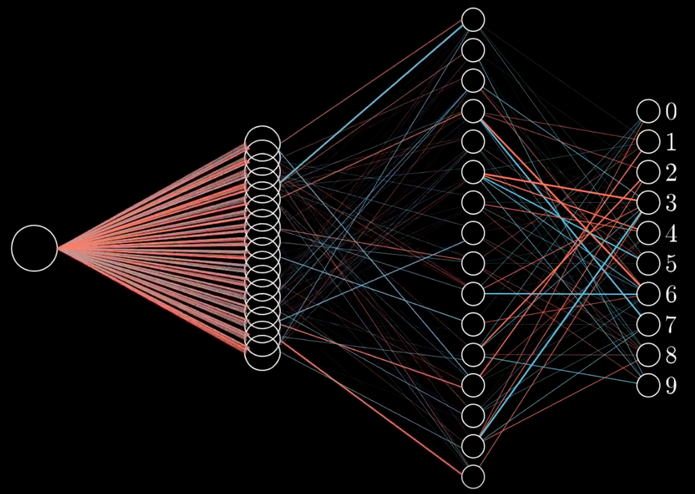
- this will **not** be the **exact** gradient of the cost (need all training data), but a **good approx**. and significant **speed up**.
- drunk but quick man

FF FCNN

Part 4: Backpropagation: **Math**

FF FCNN: Simple





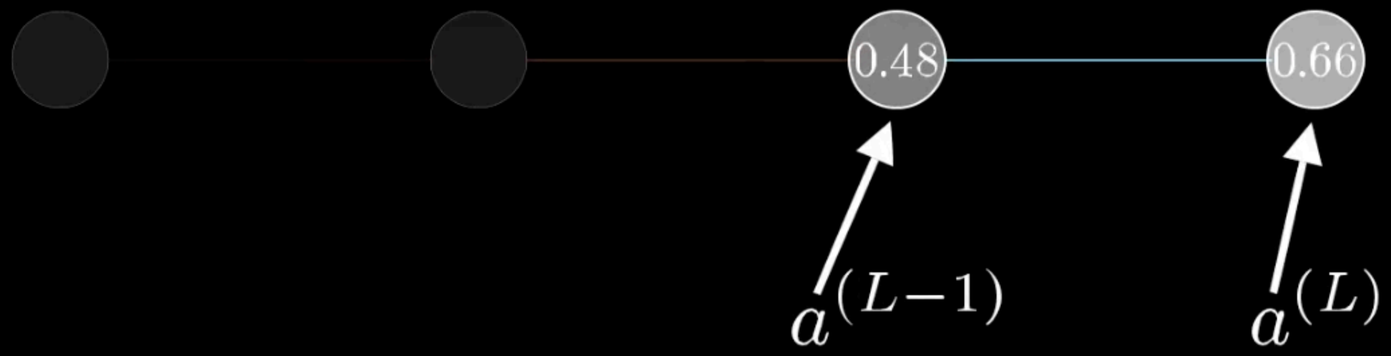
- Extremely **simple** network
- **Single** neuron in each layer

$w_1 \quad b_1 \quad w_2 \quad b_2 \quad w_3 \quad b_3$

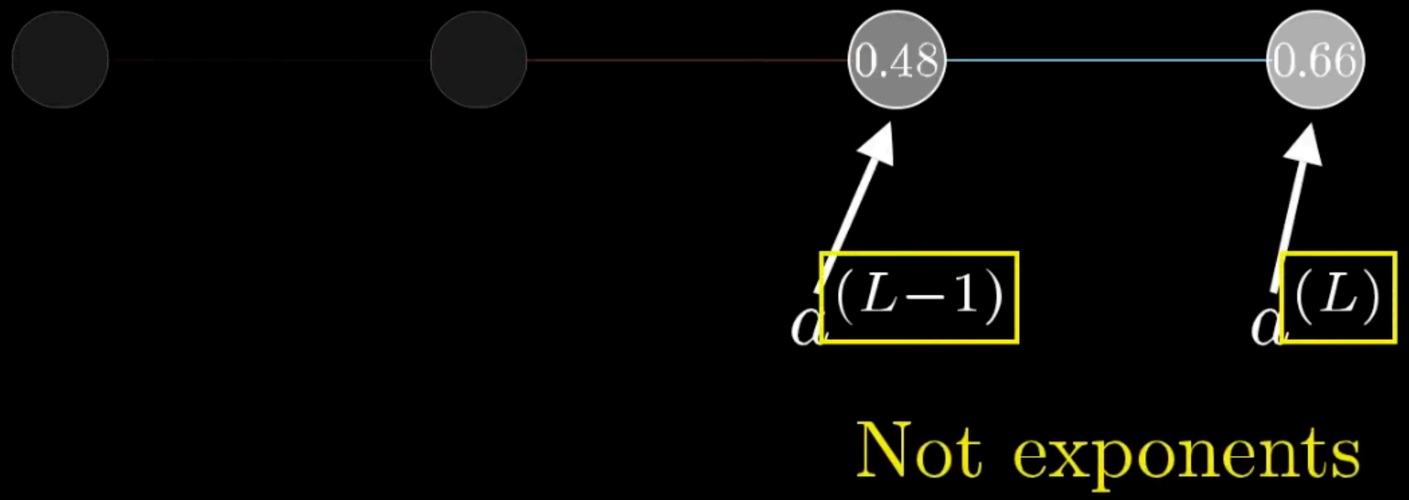
- NN **determined** by 3 weights and 3 biases

$$C(w_1, b_1, w_2, b_2, w_3, b_3)$$

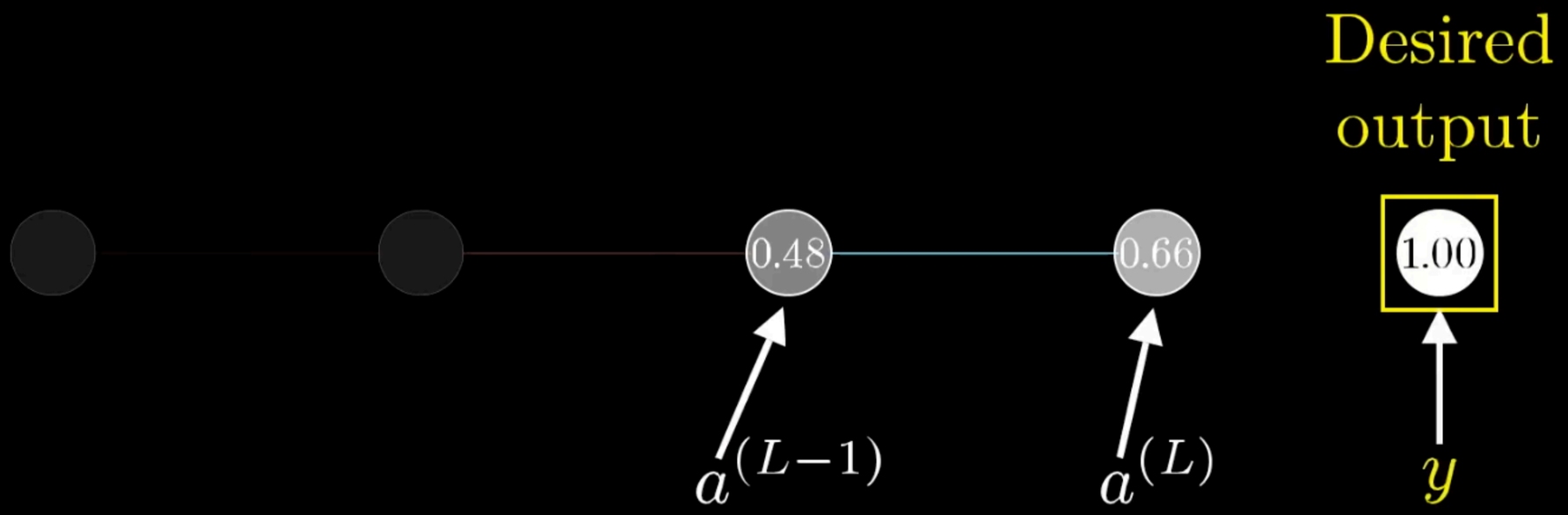
- Goal: **sensitive** cost wrt parameteres
- Which **adjustments** will cause most efficient decrease to cost



- Focus: **connection** between last two neurons
- Label: activations with **superscript** L for layer



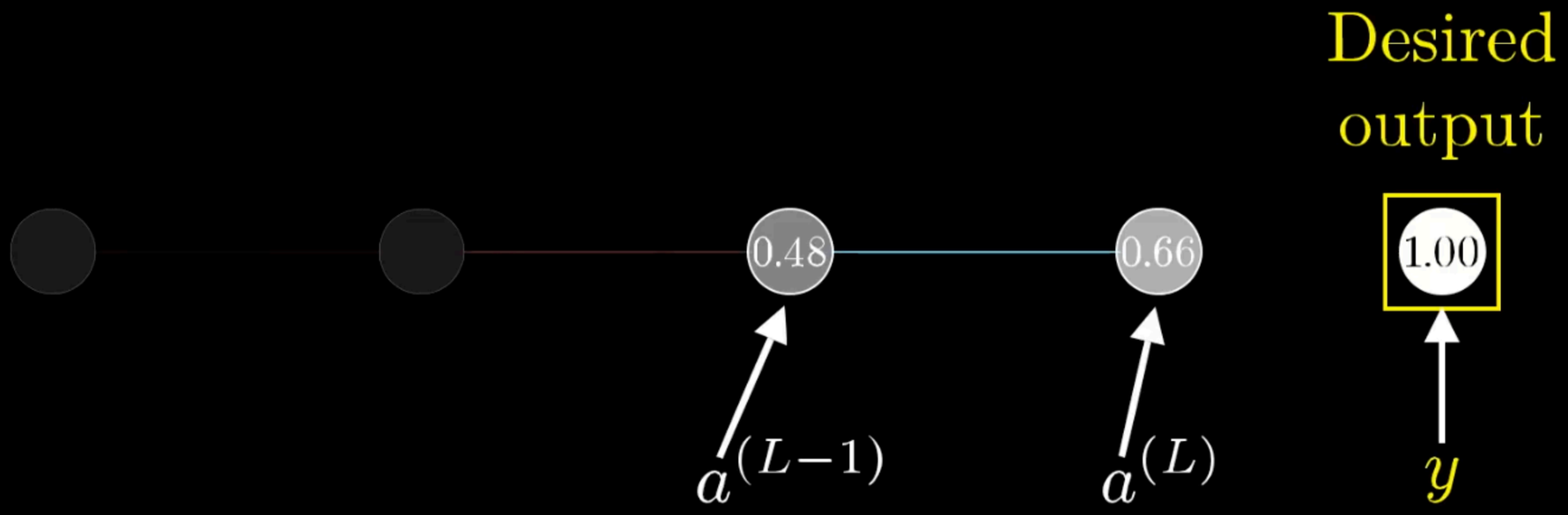
- **superscript**, NOT exponents
- subscripts: other indices



- **Desired** output value: y (e.g. zero or one)

$$\text{Cost} \longrightarrow C_0(\dots) = (a^{(L)} - y)^2$$

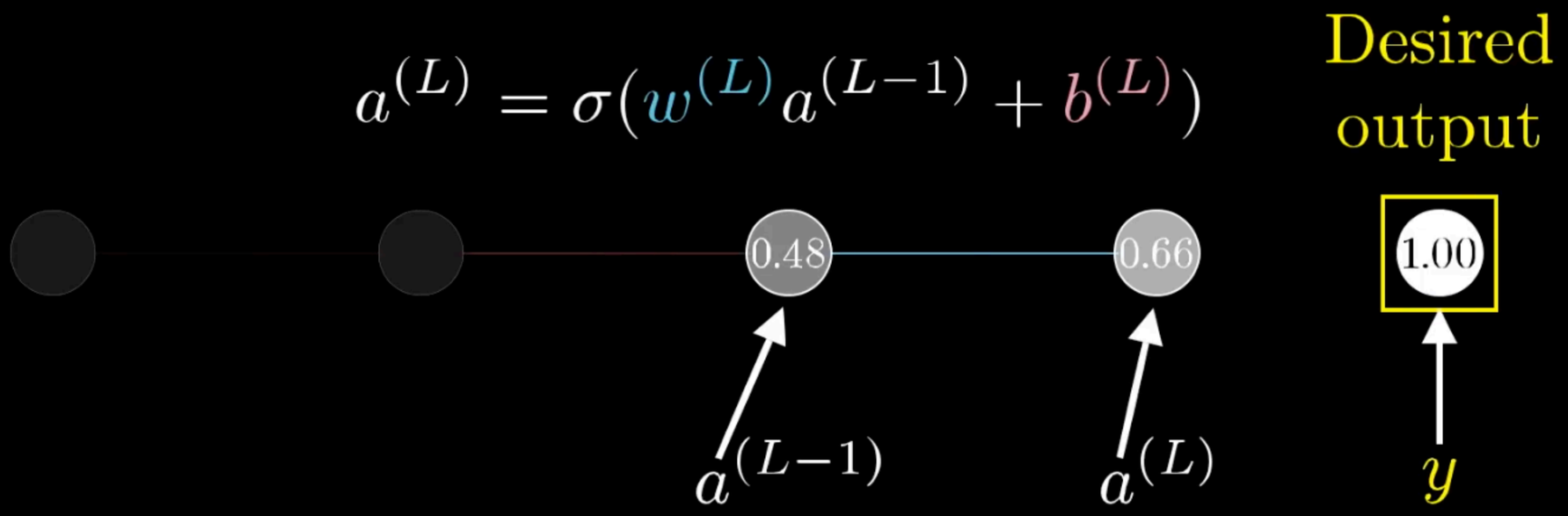
$$\text{For example: } (0.66 - 1.00)^2$$



- **Cost**: for simple NN, single example: a-L - y squared (denoted C-zero)

$$\text{Cost} \longrightarrow C_0(\dots) = (a^{(L)} - y)^2$$

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

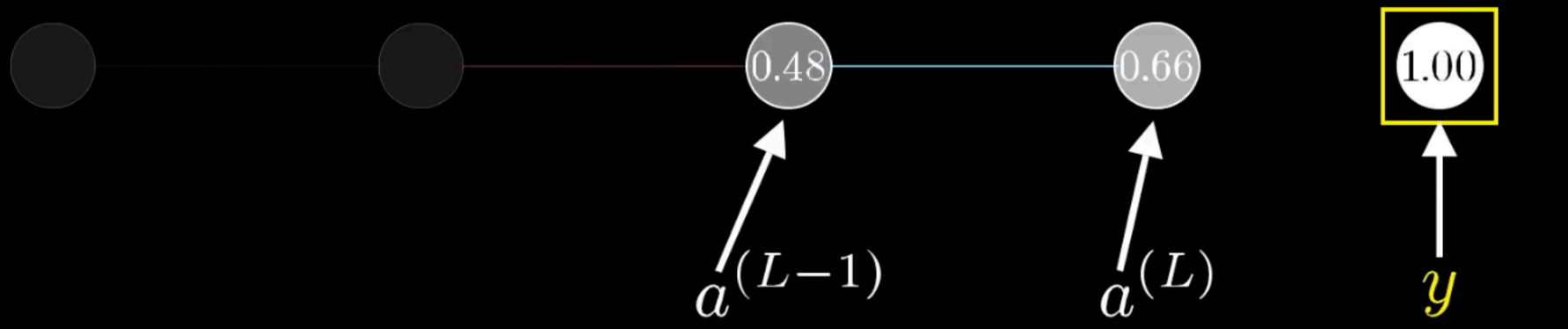


- **Reminder**: activation function (activation = weight * previous activation + bias)

$$\text{Cost} \longrightarrow C_0(\dots) = (a^{(L)} - y)^2$$

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

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

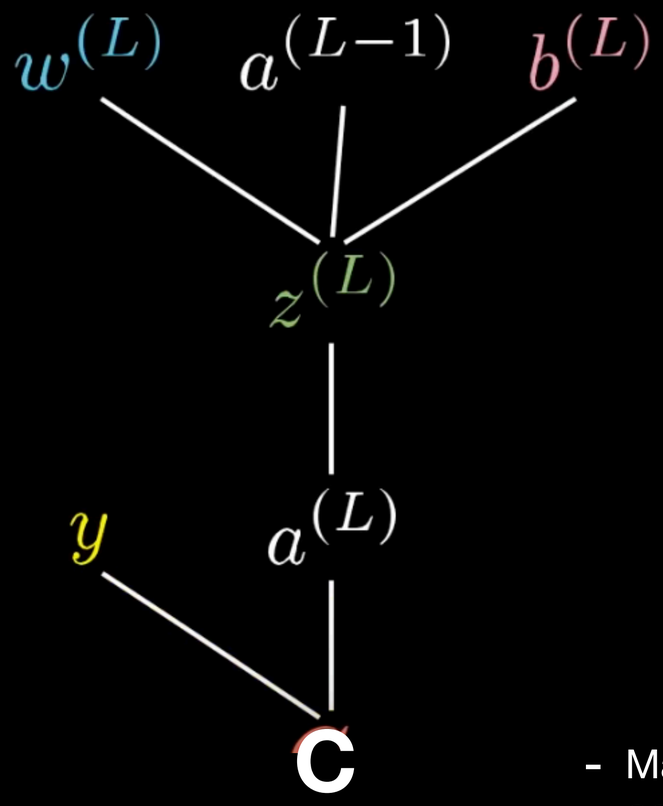


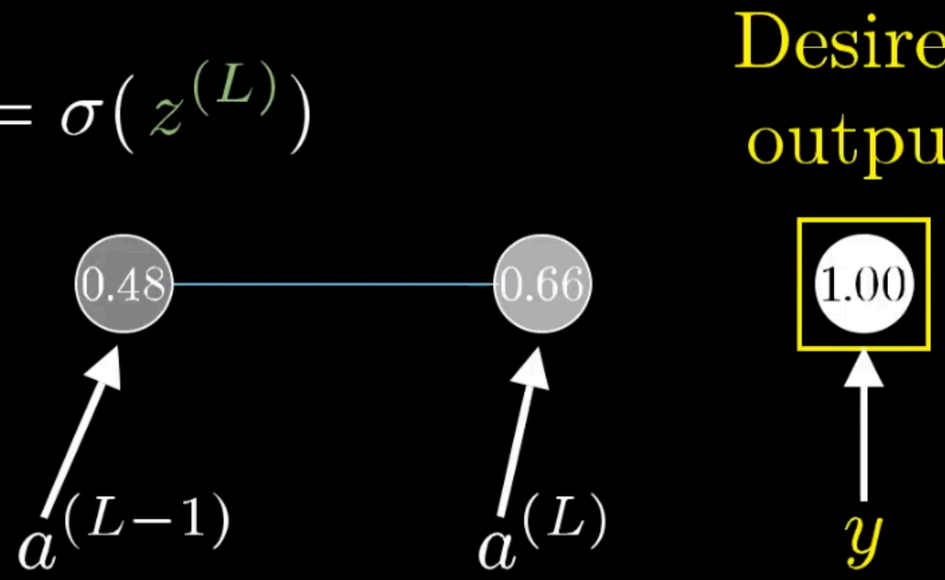
- **Special** name to weighted sum: **z** (same superscript)

$$\text{Cost} \longrightarrow C_0(\dots) = (a^{(L)} - y)^2$$

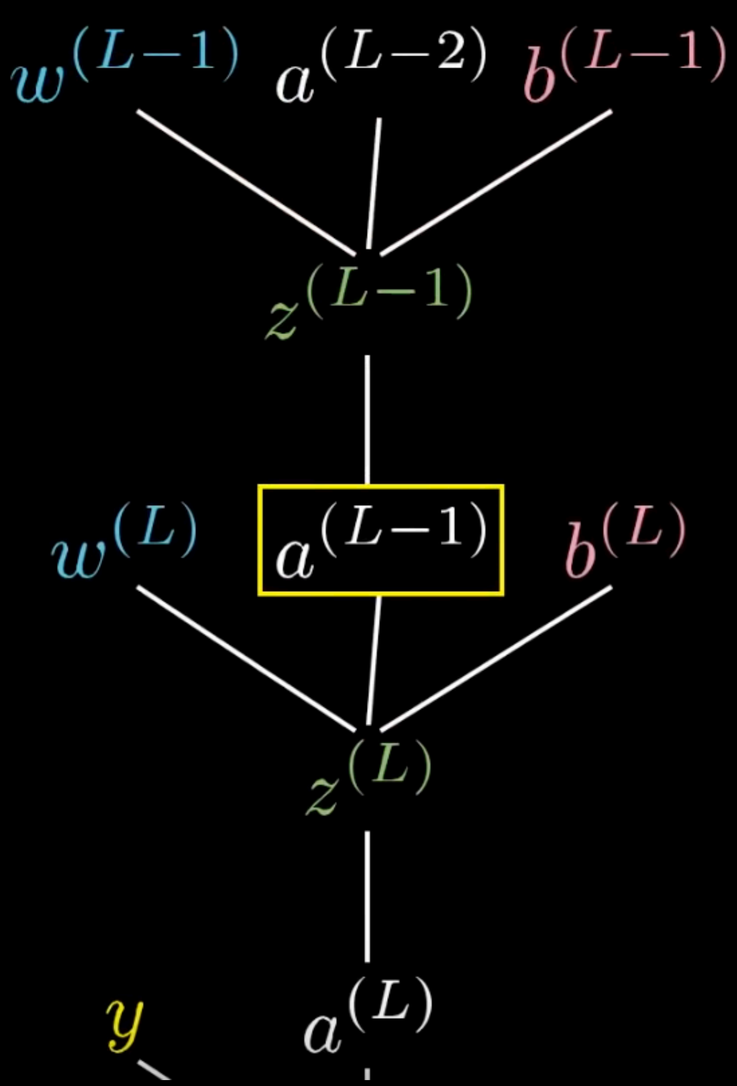
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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





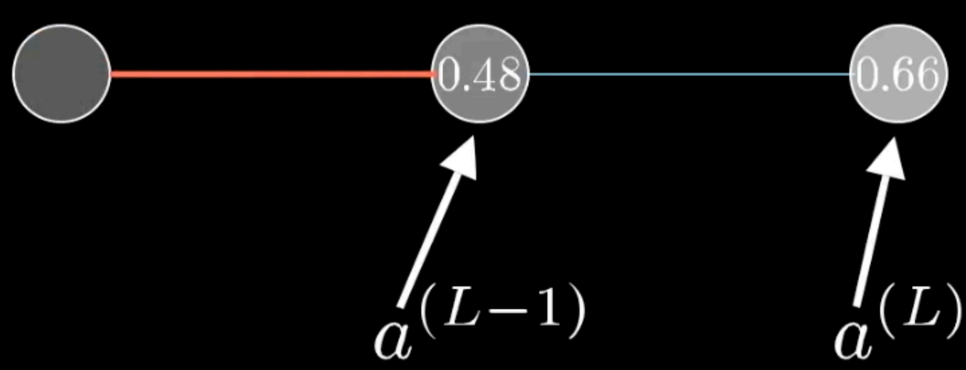
- Many **terms**
- **conceptualize**: w, pre-a and b to **compute** z
- z to compute a and a along with constant y to compute cost



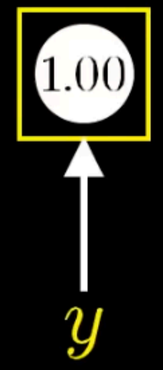
$$\text{Cost} \rightarrow C_0(\dots) = (a^{(L)} - y)^2$$

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

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



Desired output

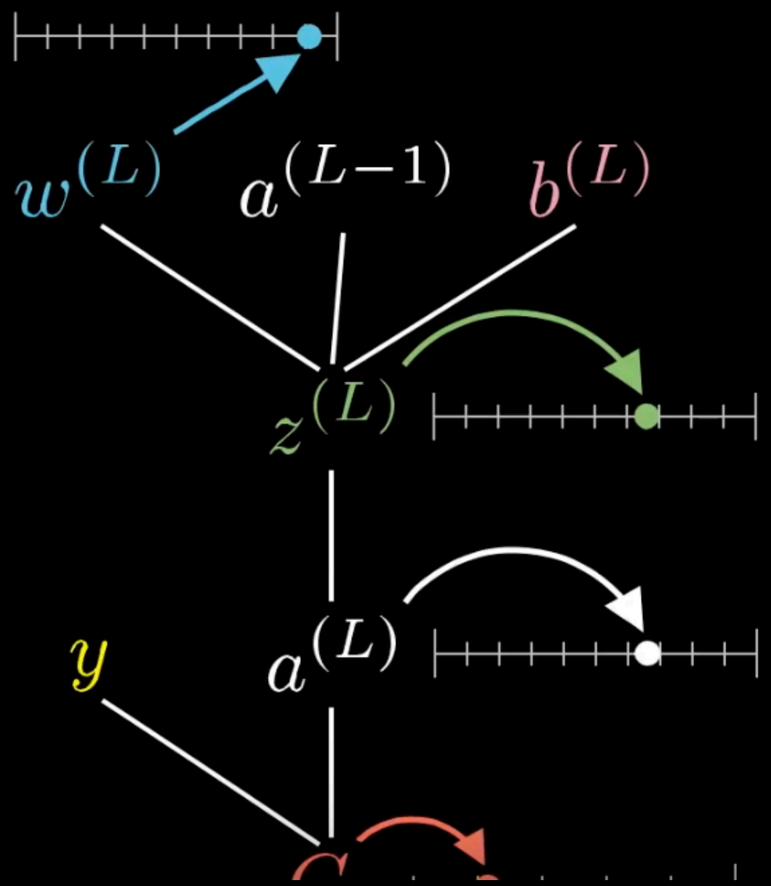


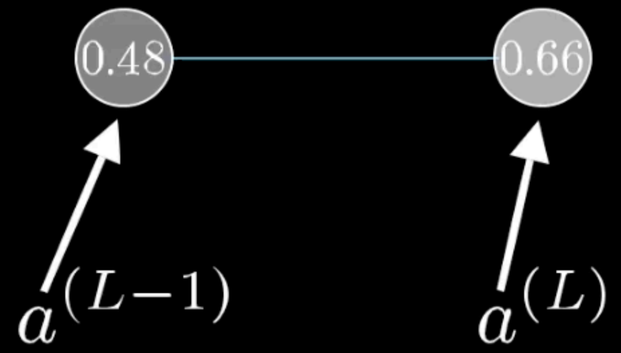
- a-(L-1) is **influenced** by own weight, pre-activation and bias

$$\text{Cost} \longrightarrow C_0(\dots) = (a^{(L)} - y)^2$$

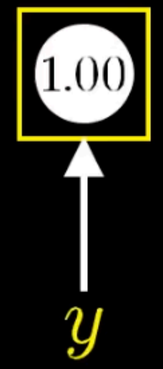
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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

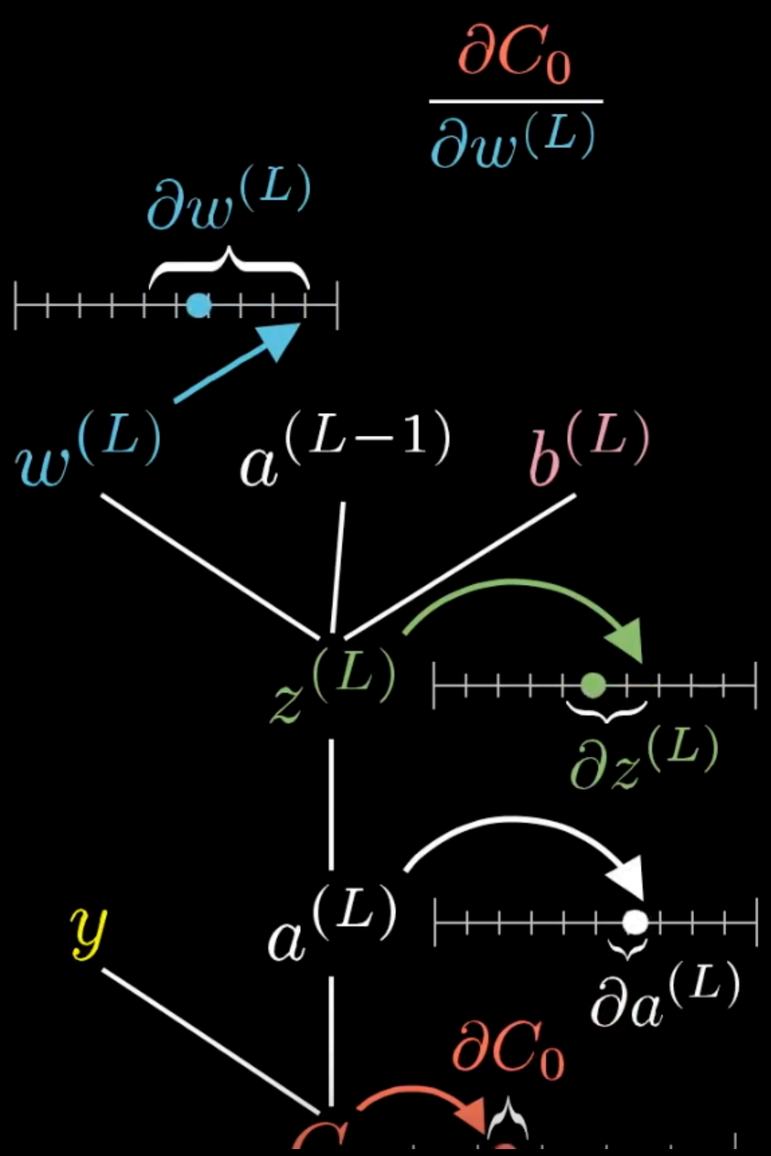




Desired output



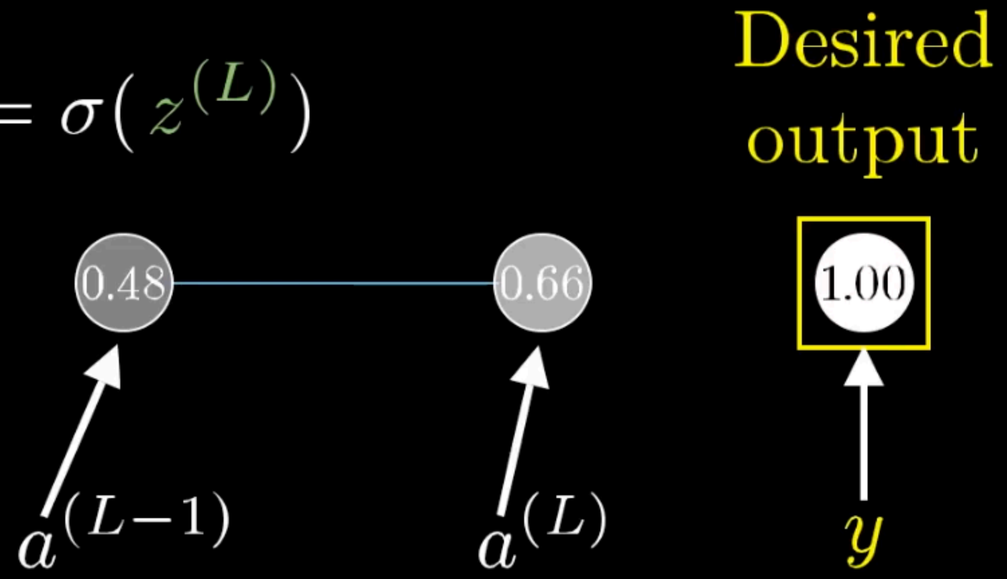
- all terms numbers with number lines



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

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

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



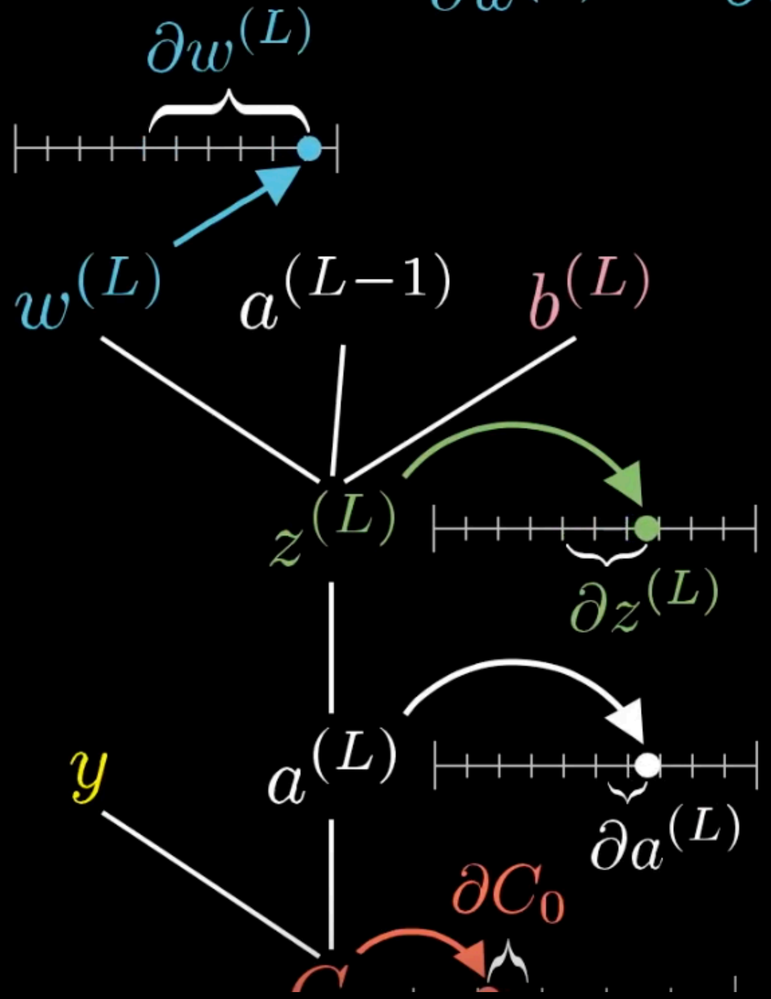
- Goal: how **sensitive** is cost to small changes in w-L
- Alt: derivative of C wrt small change in w-L
- delta w = tiny nudge to w (e.g. 0.01), delta C = resulting nudge to cost
- We want the ratio
- Conceptually: tiny nudge to w-L, causes z-L, a-L and directly the cost.

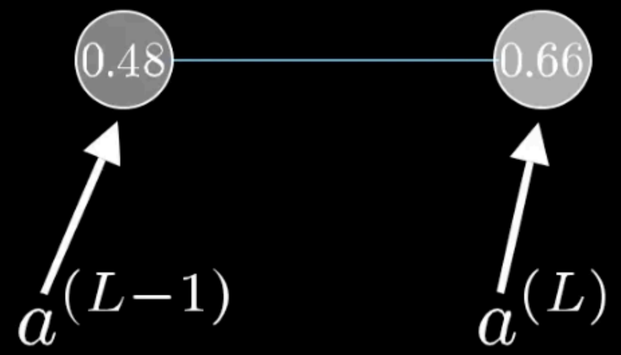
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

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

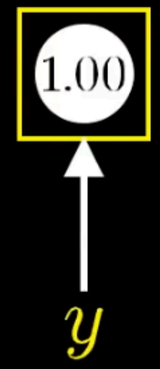
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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





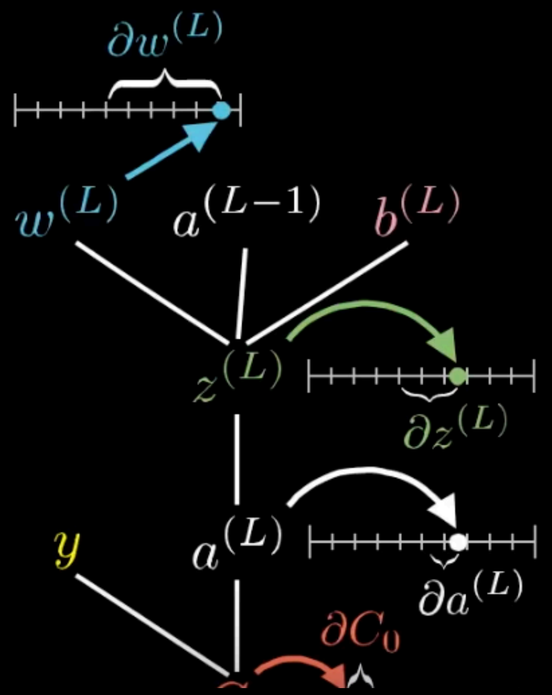
Desired output



- Break up: ratio tiny change to z-L vs. w-L (derivative of z wrt to w)
- Likewise: change a-L to z-L
- As well as: nudge C to a-L

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

Chain rule

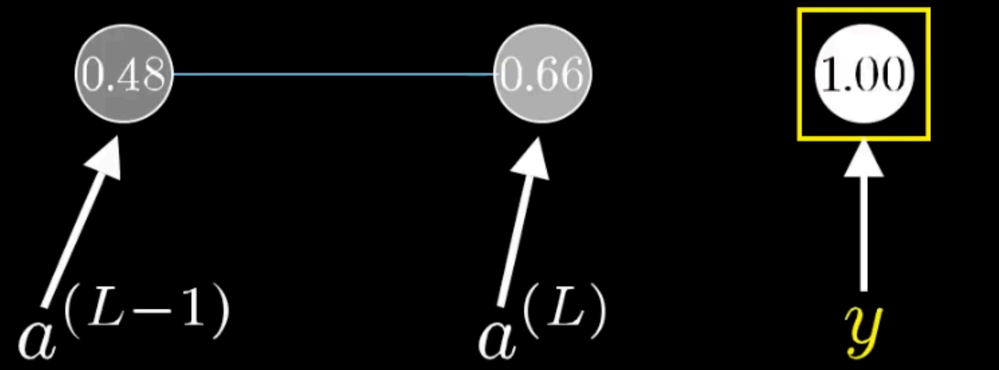


$$C_0(\dots) = (a^{(L)} - y)^2$$

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

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

Desired output



- Chain rule: 3 ratios, sensitivity of C to small changes in w-L
- Many symbols, meaning? compute 3 derivatives

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

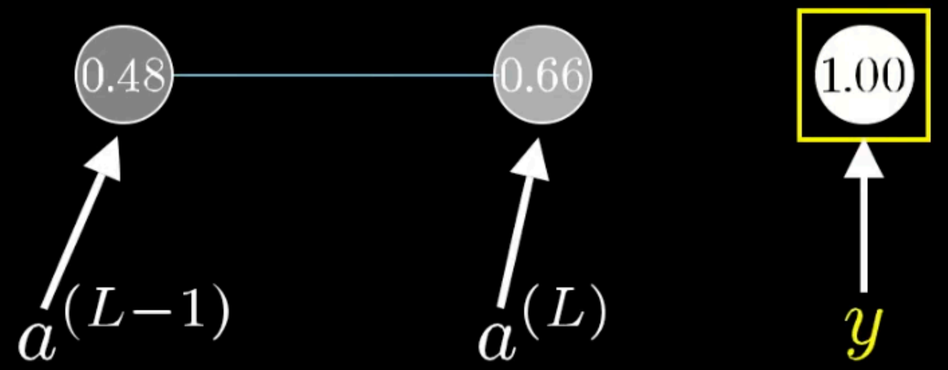
$$\frac{\partial C0}{\partial a^{(L)}} =$$

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

1/2

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

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



- **Derivative** of C wrt a-L

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

1/2*2=1

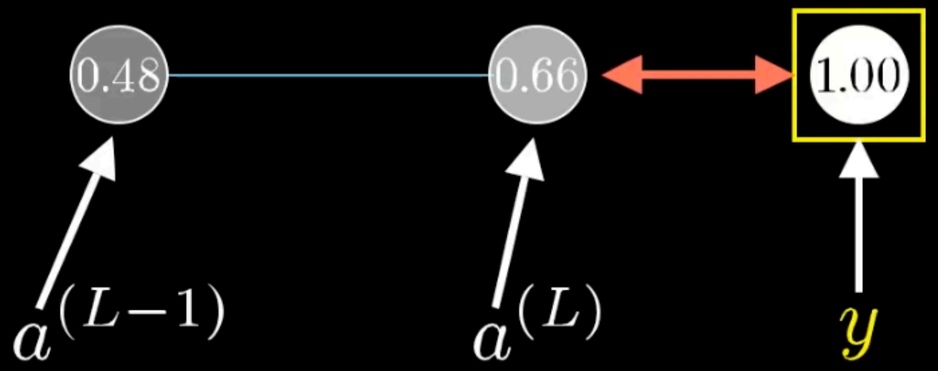
$$\frac{\partial C0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$0.66 - 1.00$$

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

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

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



- Will be 2 times a-L - y
- Proportional to difference
- Outupt very different, small change to a-L big impact on cost

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

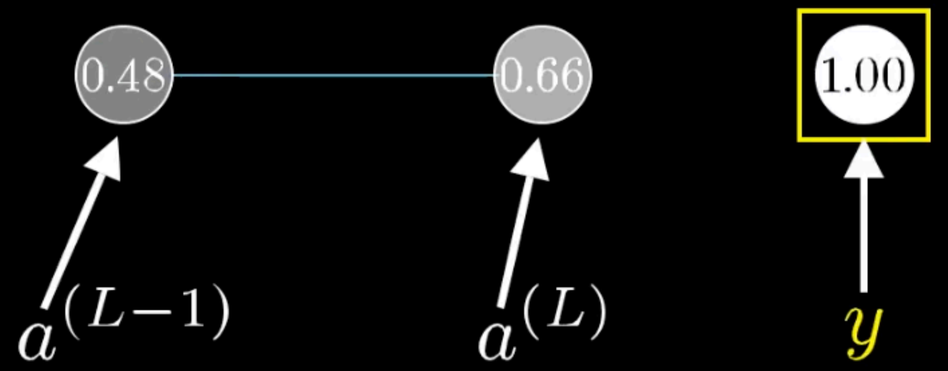
$$C_0 = (a^{(L)} - y)^2$$

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

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

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

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



- **Derivative** of a-L wrt z-L

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

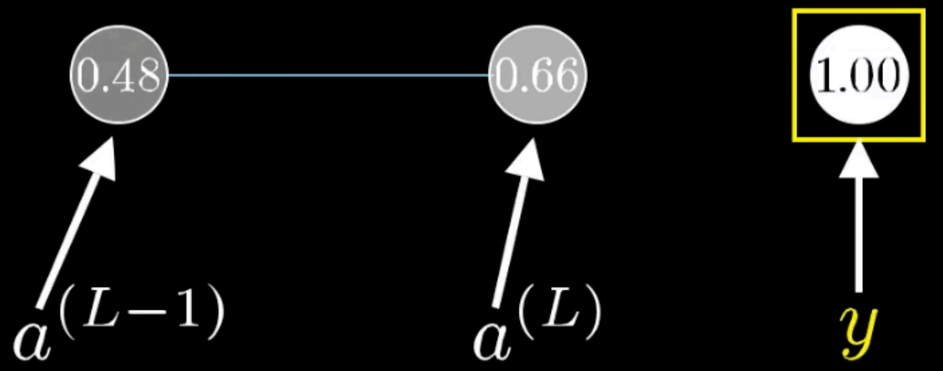
$$\frac{\partial C0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

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

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

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

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



- **Derivative** of non-linearity / sigmoid / ReLU

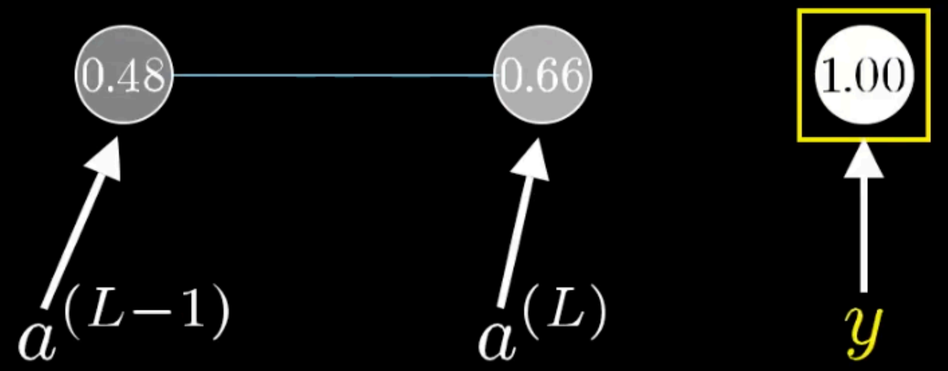
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

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

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

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

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



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

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

- **Derivative** of z-L wrt w-L

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

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

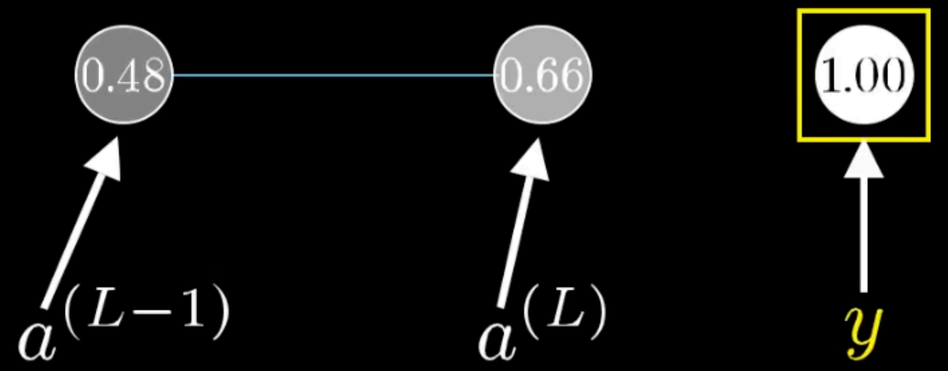
$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

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

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

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

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



- Will be a-(L-1)
- Many formulas, what to they all mean?

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

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

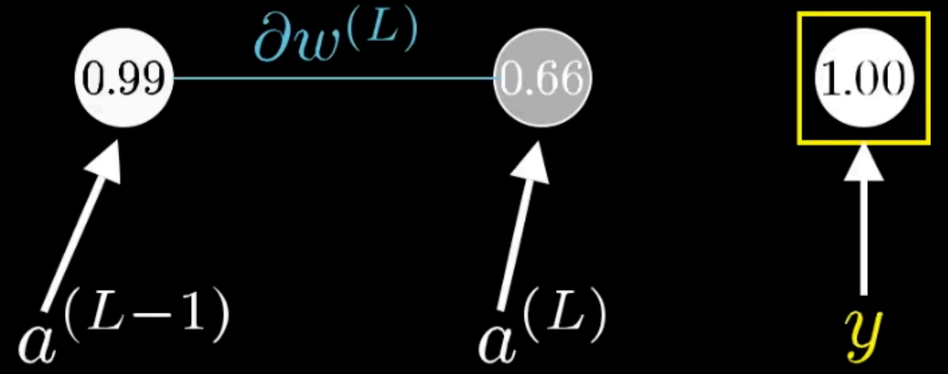
$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

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

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

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

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



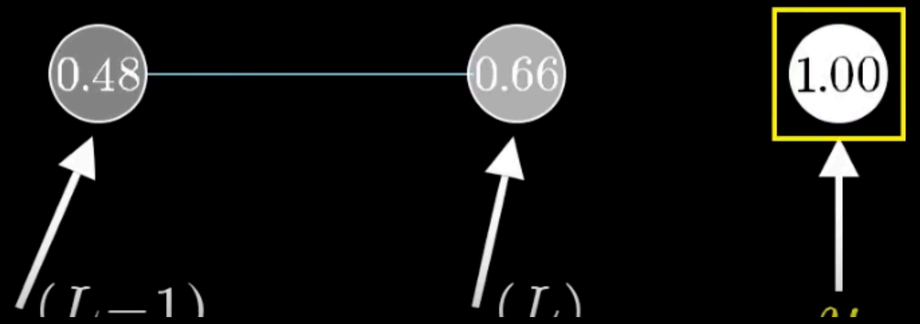
- Last derivative: amount small nudge to w influence last layer depends on how strong pre-neuron is.
- Remenber: «neurons that fire together wire together»

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

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

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

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



- All: derivative of cost wrt. w-L for only on training example

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

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

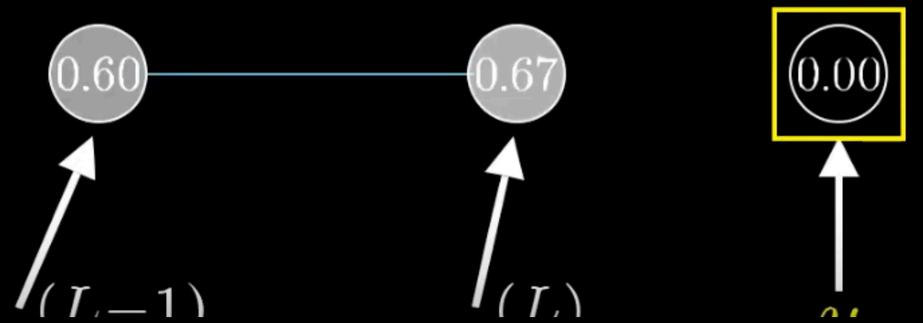
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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

Average of all training examples

$$\underbrace{\frac{\partial C}{\partial w^{(L)}}} = \overbrace{\frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}}$$

Derivative of



- Since full cost = averaging costs from many examples.
- Derivative requires to average the expression found over all training examples.

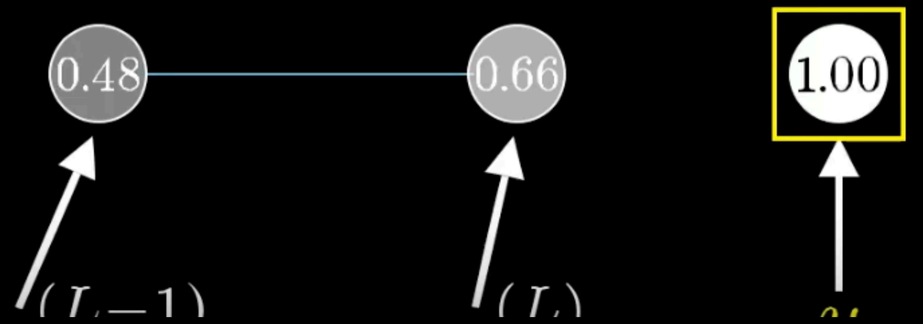
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$

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

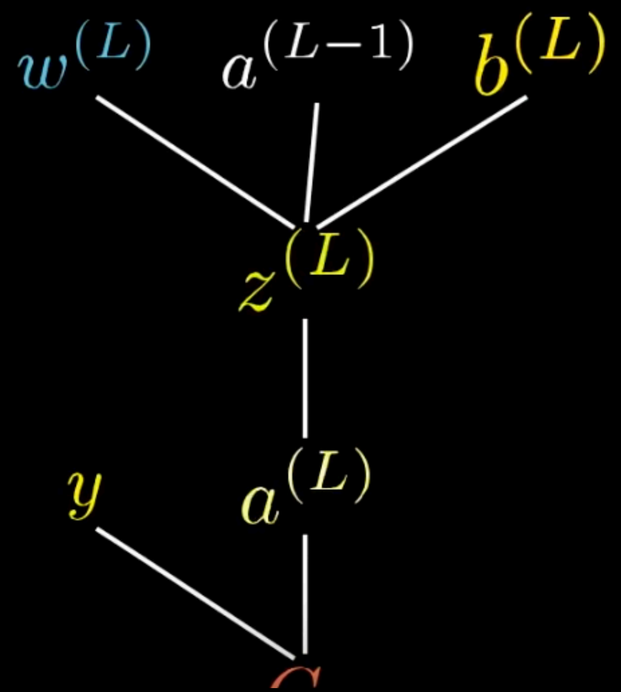
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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



- One comp. of gradient vector (partial derivatives of cost wrt all weights and biases)

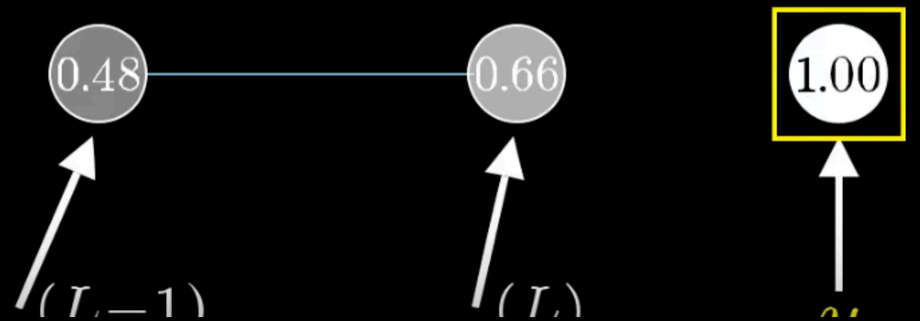
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$



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

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

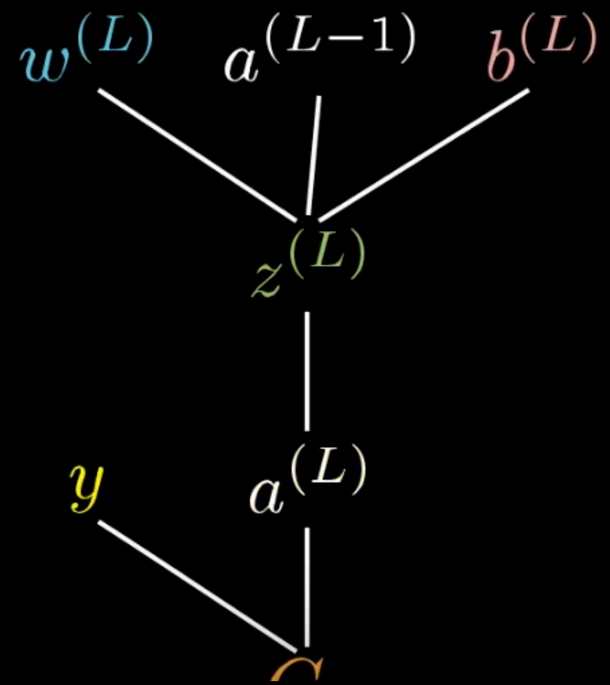
$$a^{(L)} = \sigma(z^{(L)})$$



- Just one comp., but more then 50% of the work done

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

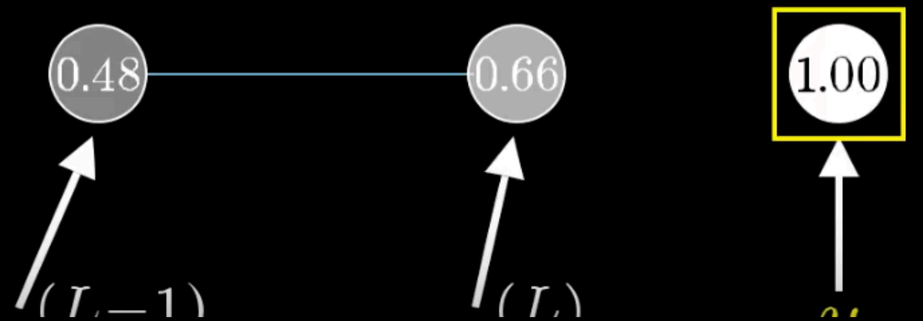
$$\sigma'(z^{(L)})2(a^{(L)} - y)$$



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

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

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



- Sensitivity of bias: almost identical, change del z / del w with del z / del b

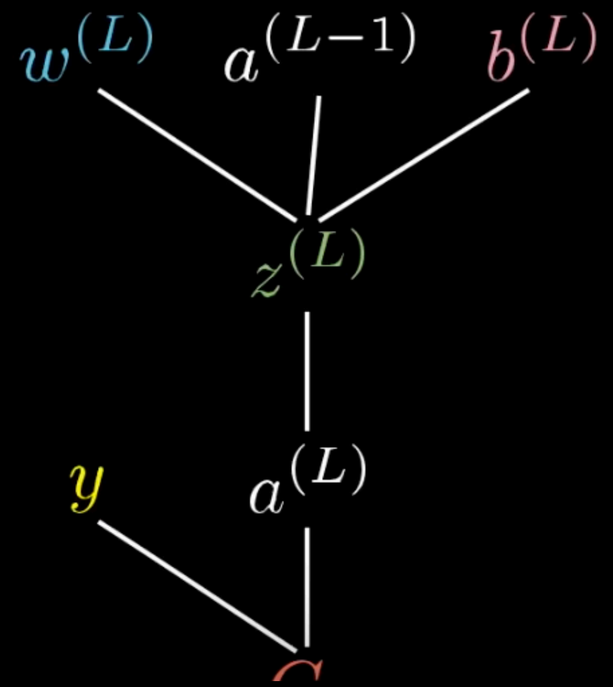
$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} =$$

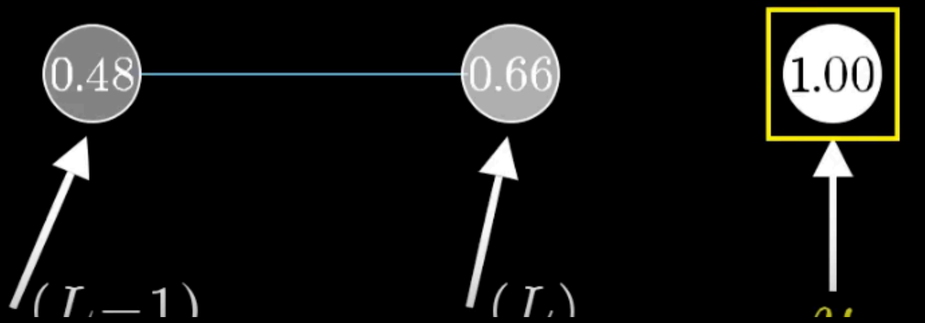
$$1\sigma'(z^{(L)})2(a^{(L)} - y)$$

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

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

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





- Relevant formula: derivative = 1

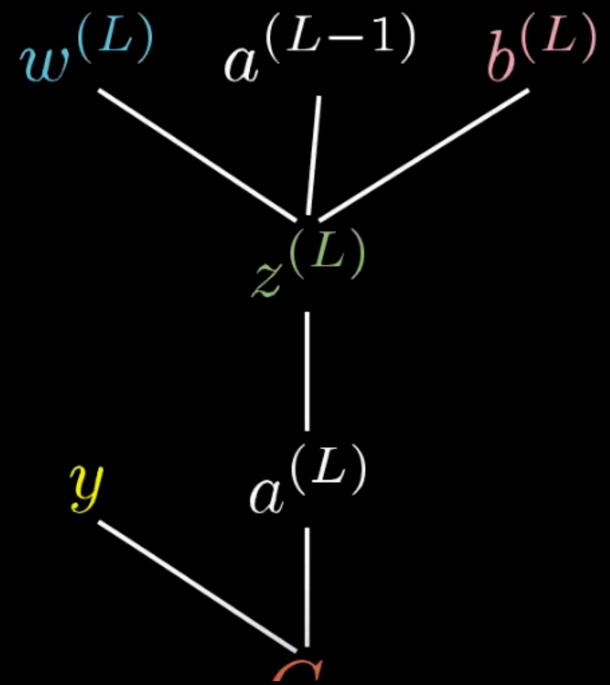
$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} =$$

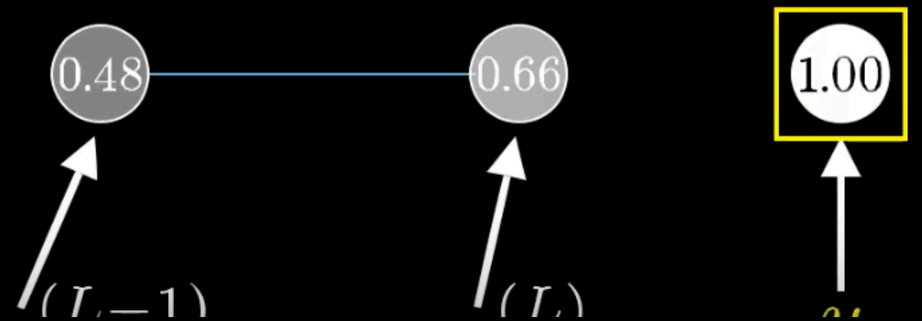
$$\sigma'(z^{(L)}) 2(a^{(L)} - y)$$

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

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

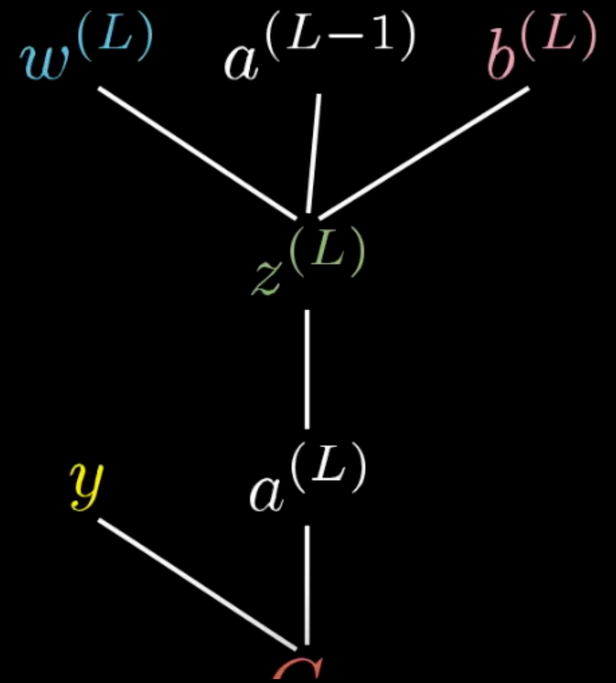
$$a^{(L)} = \sigma(z^{(L)})$$





- Sensitivity of cost to pre_activation a-(L-1): back-prop comes in:
- Initial derivative in chain rule expression

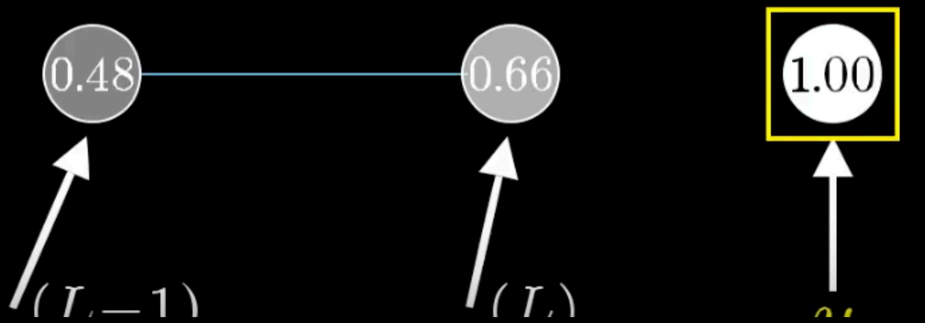
$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = w^{(L)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$



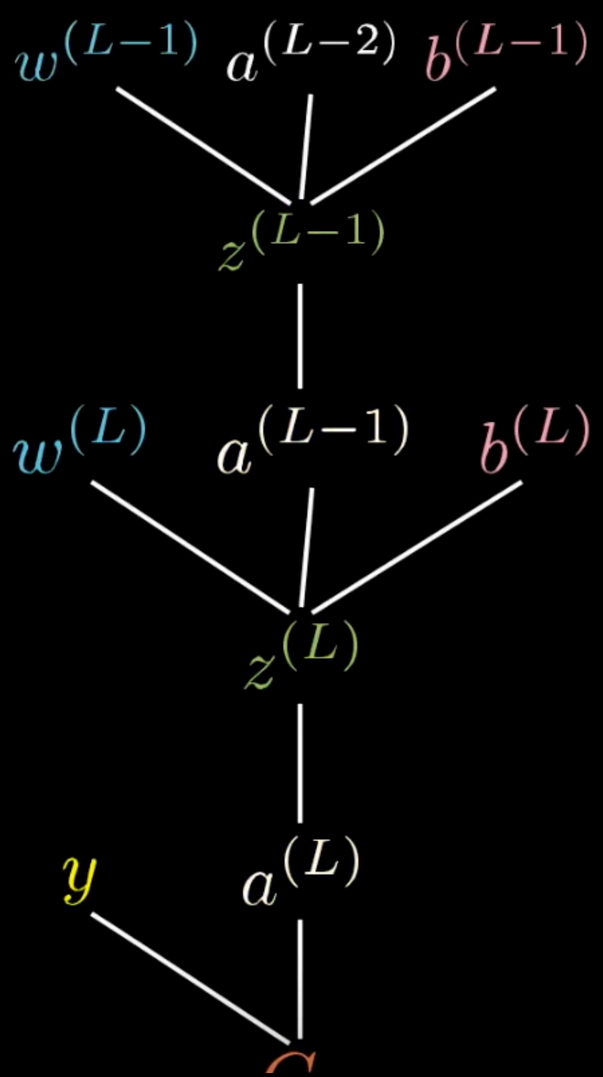
$$C_0 = (a^{(L)} - y)^2$$

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

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



- Sensitivity of z to pre_a = w-L

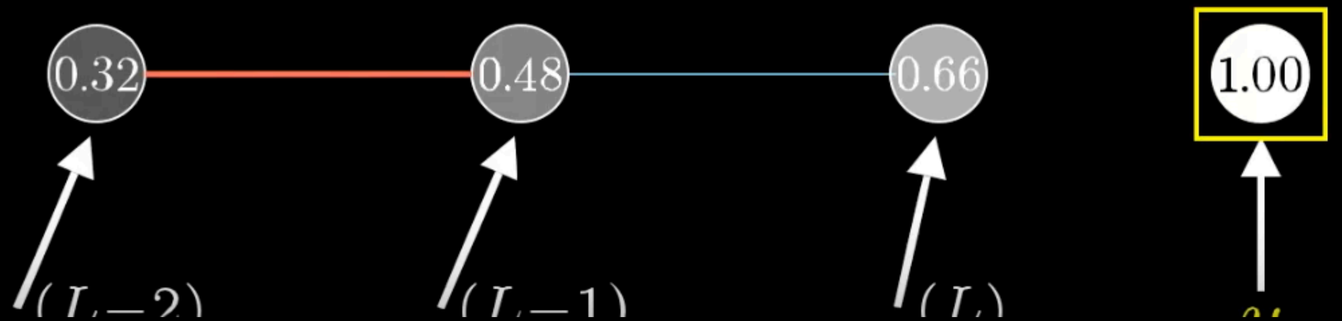


$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

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

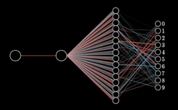
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

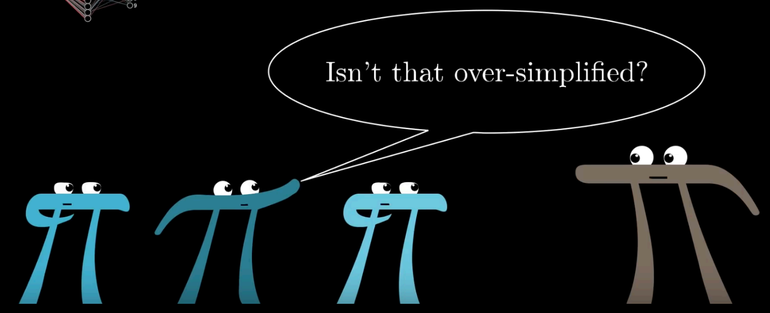
$$a^{(L)} = \sigma(z^{(L)})$$

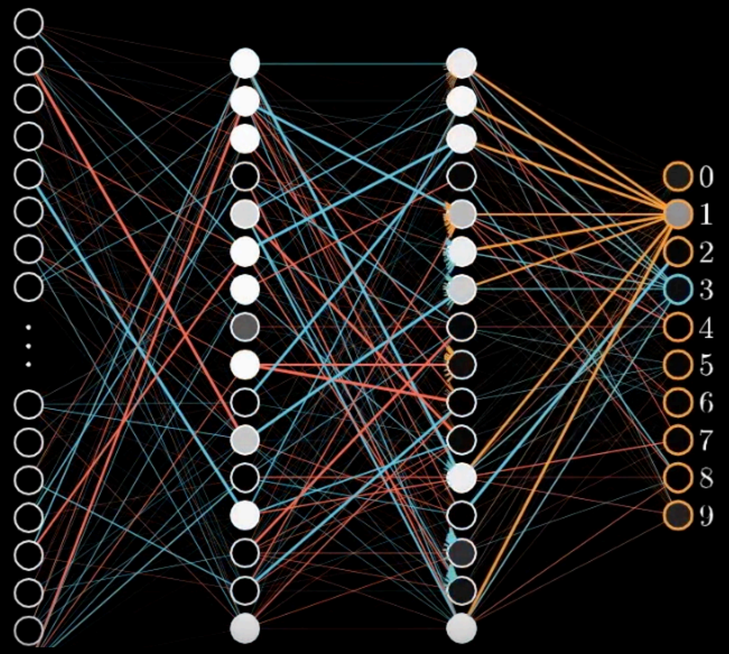


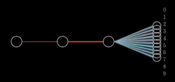
- Again: no influence on pre_a, kept iterating backwards (chain rule idea): sensitivity of cost to pre_pre_w&b

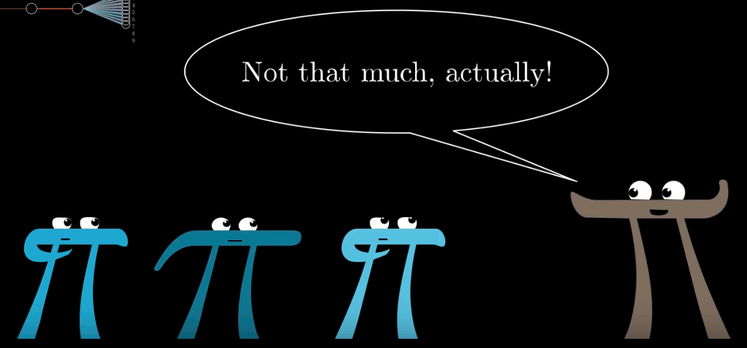
FF FCNN: General



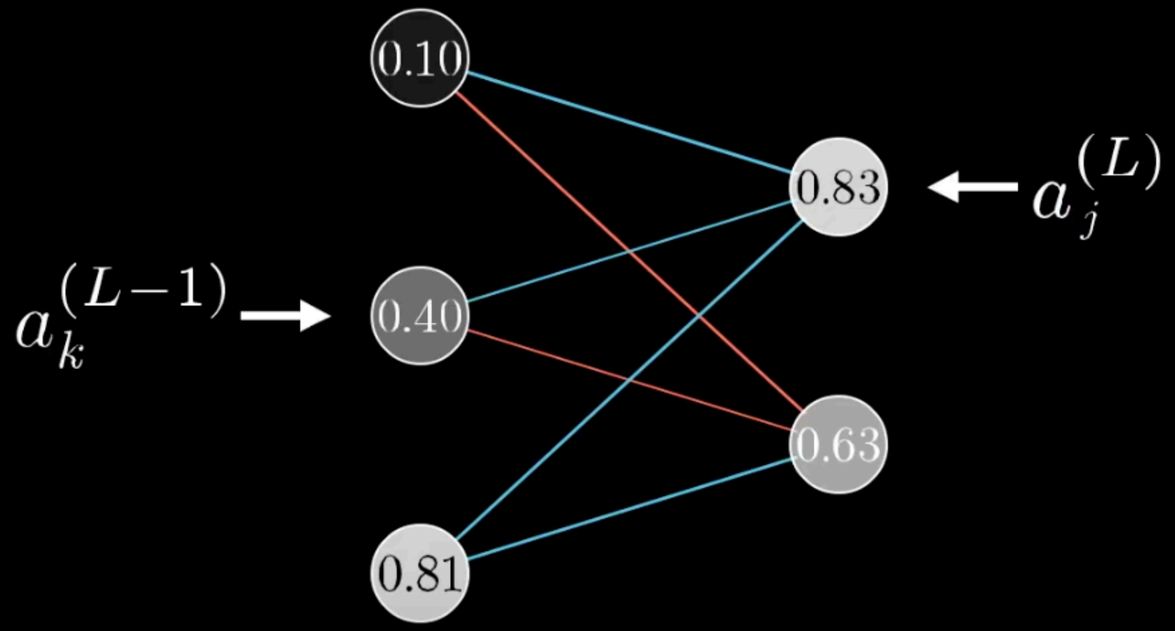






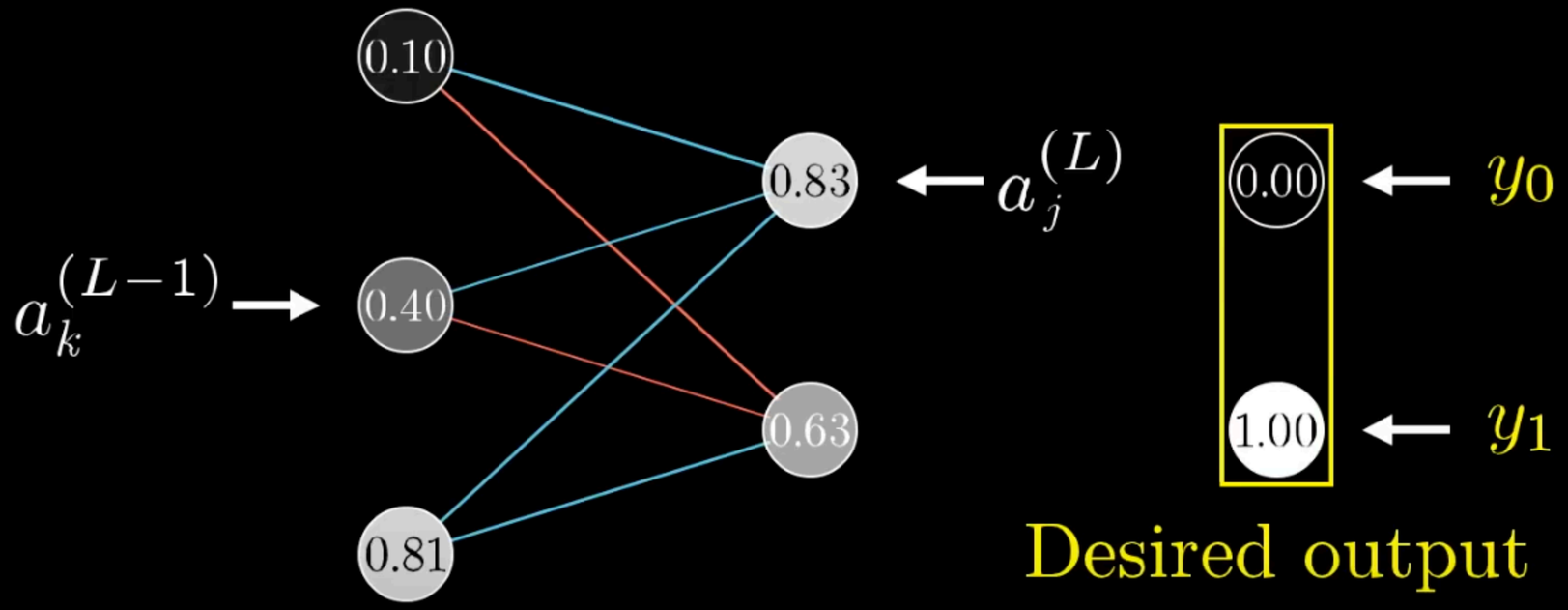


- Overly simple example (all layers: just one neuron)?
- Not that much changes when the layers have multiple neurons
- Just a few more indices to keep track of



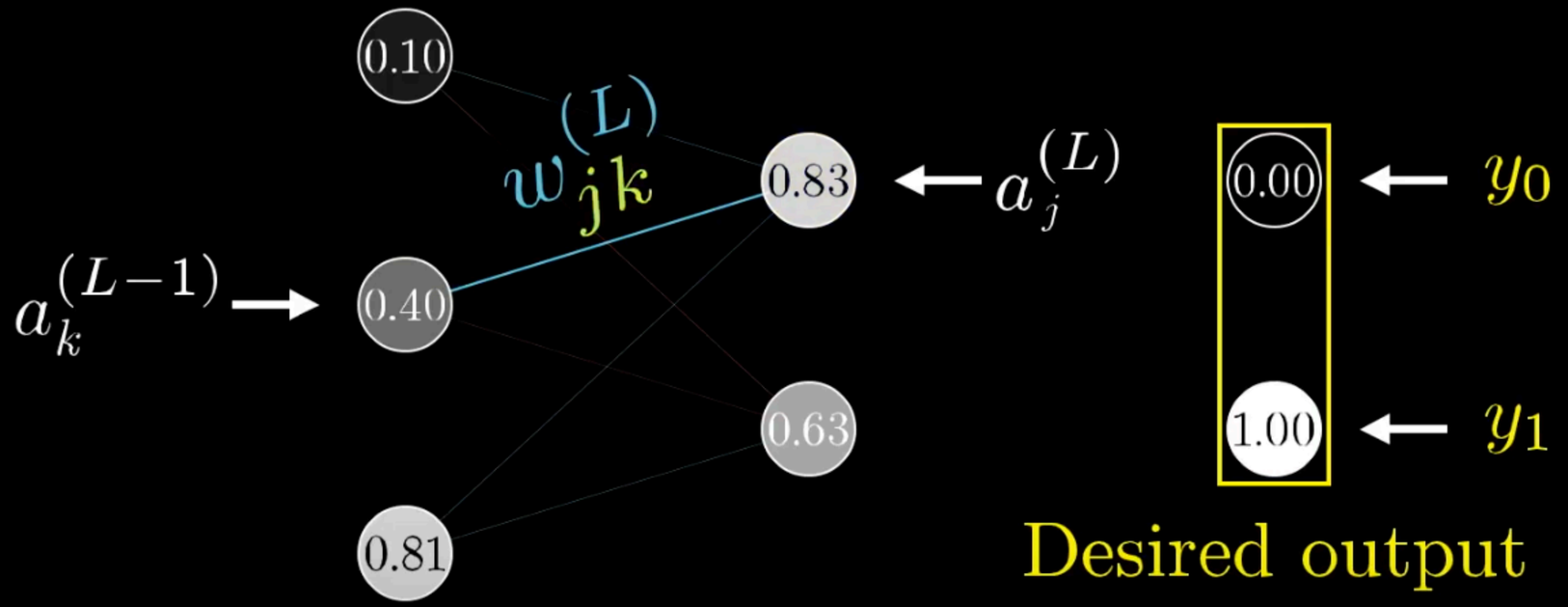
- Activation of given layer: subscript indicates which neuron
- Book (Nielsen): use k to index layer L - 1 and j to index layer L
- We: i (input), j (hidden) and k (output)

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

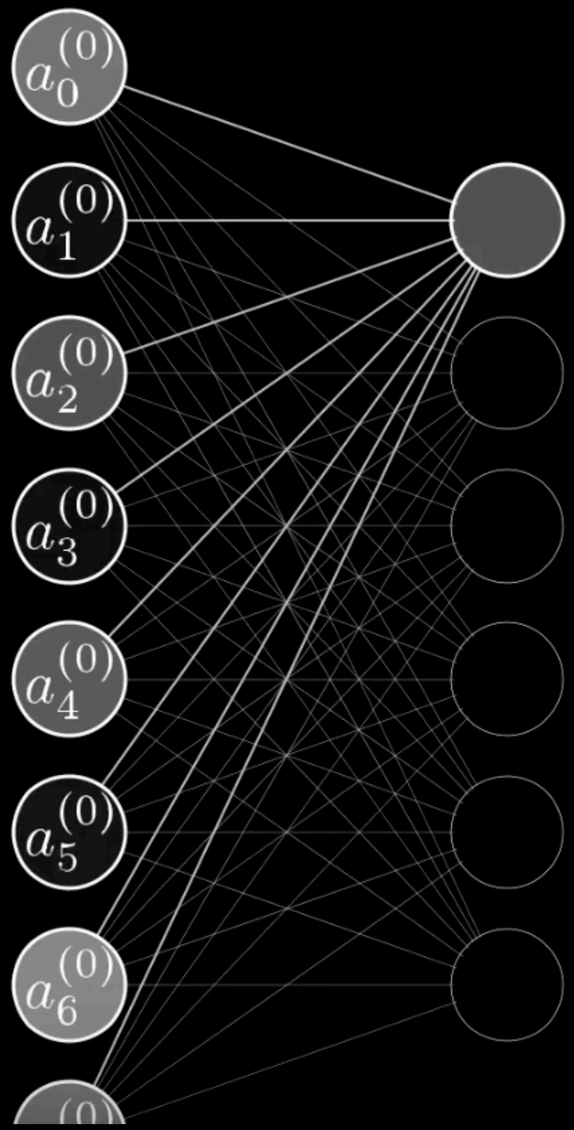


- Again: for **cost** we look at **desired** output
- But now: add all squared differences (last layer activations and desired output)
- Sum over a-j-L minus y-j squared

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



- Lot of weights: need two indices / subscripts to keep track of them
- Weight of edge connecting k-th and j-th neurons: w-jk-L
- Might feel a little backwards



Sigmoid

$$a_0^{(1)} = \sigma\left(w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)} + b_0 \right)$$

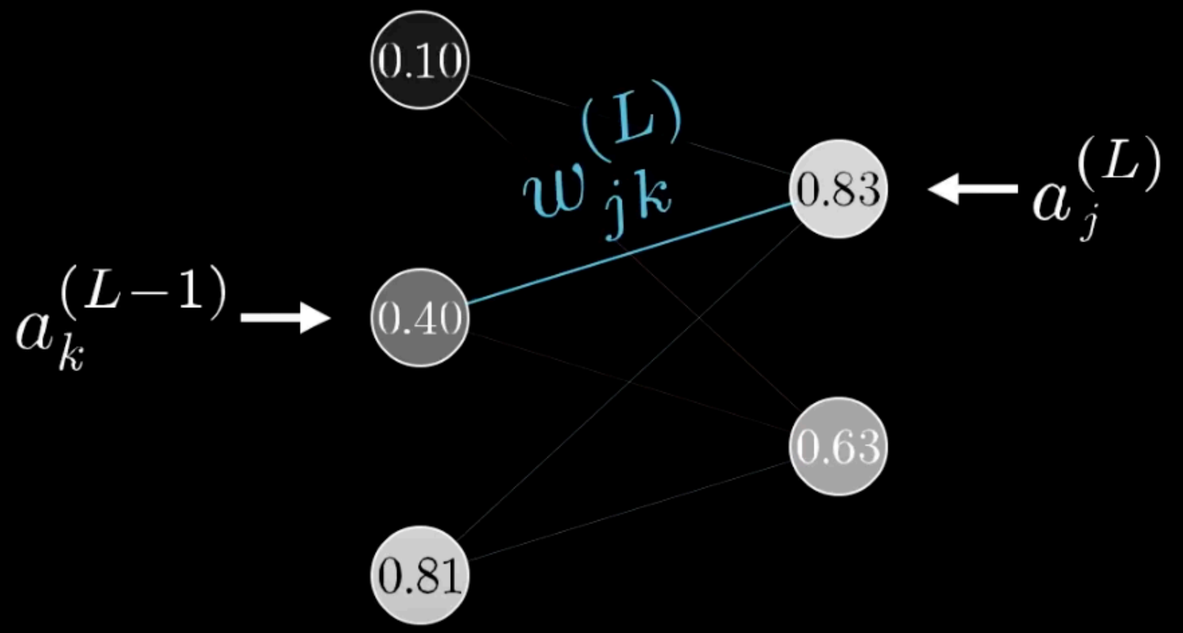
Bias

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix}$$

w_jk?

- Lines up with the indexing of the weight matrix
- Book vs. we (learn what is really going on)

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



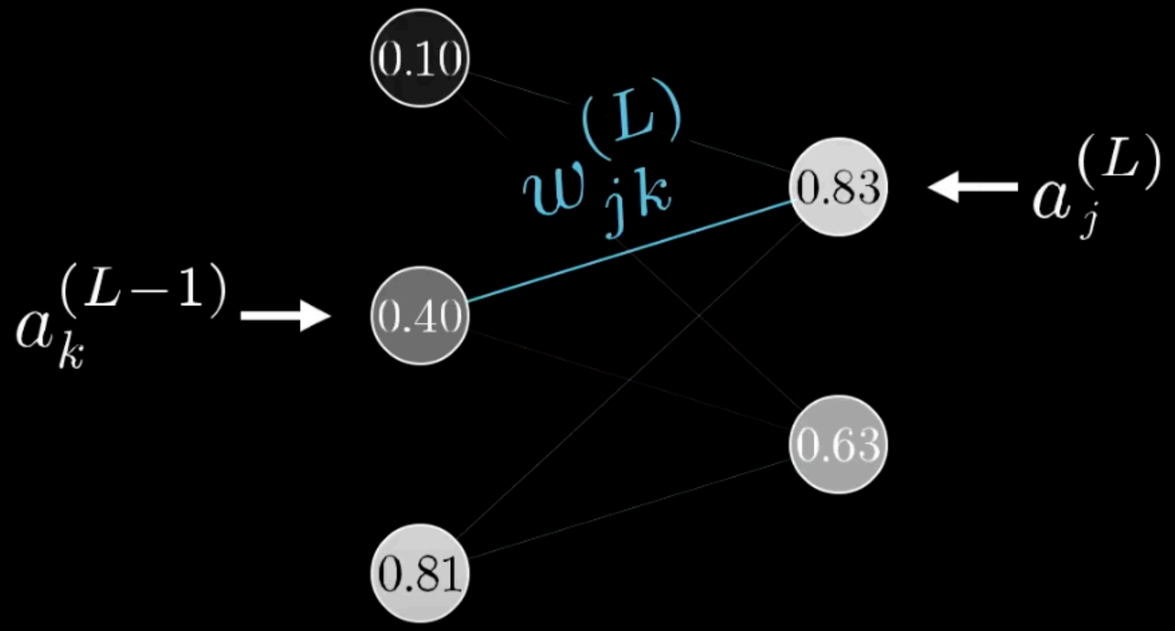
$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$

- **As before**: name the wighted sum as z..

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

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

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



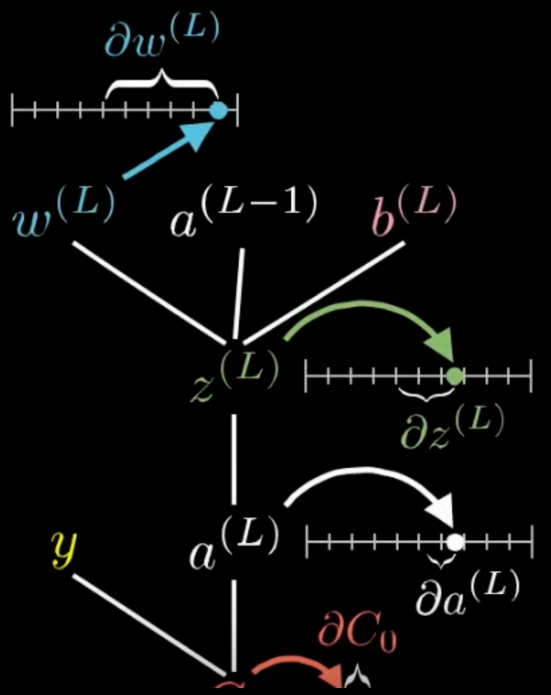
- so that the activation is just the non-linearity applied to z
- All equations here are essentially the same..

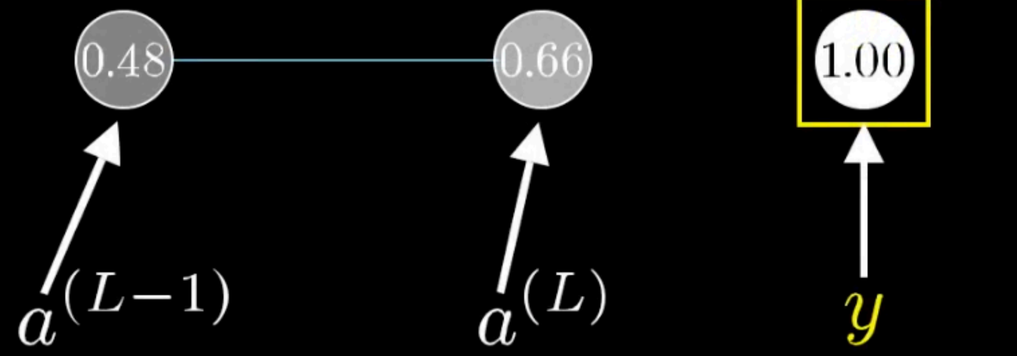
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

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

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

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



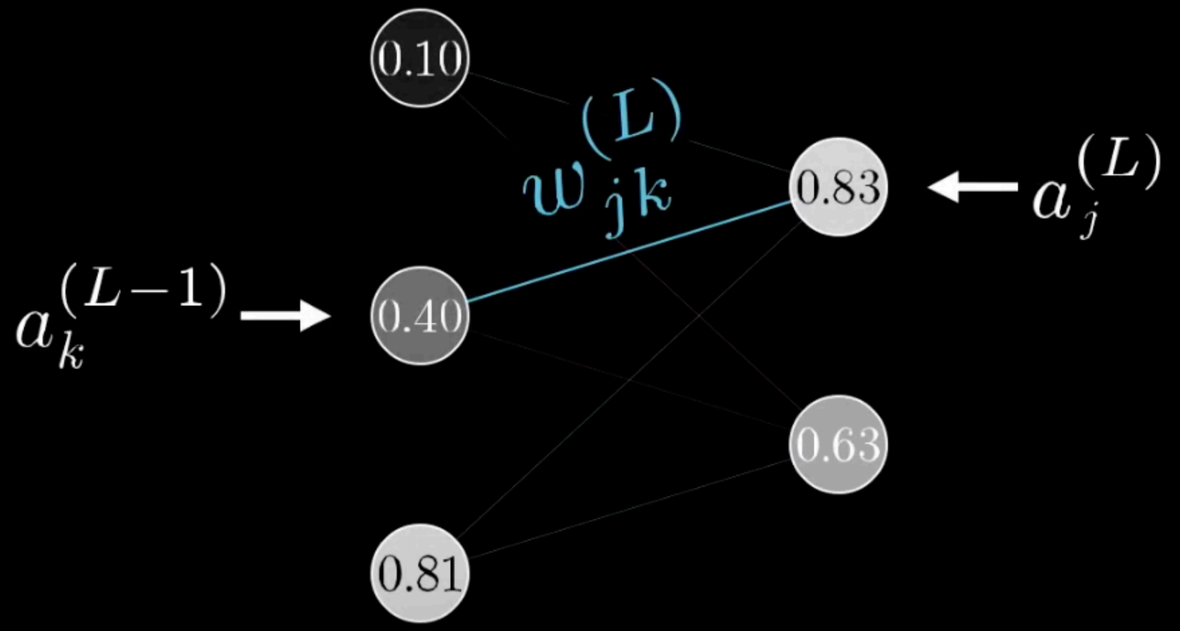


- .. as the equations we had before in the one-neuron-per-layer case..

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

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

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



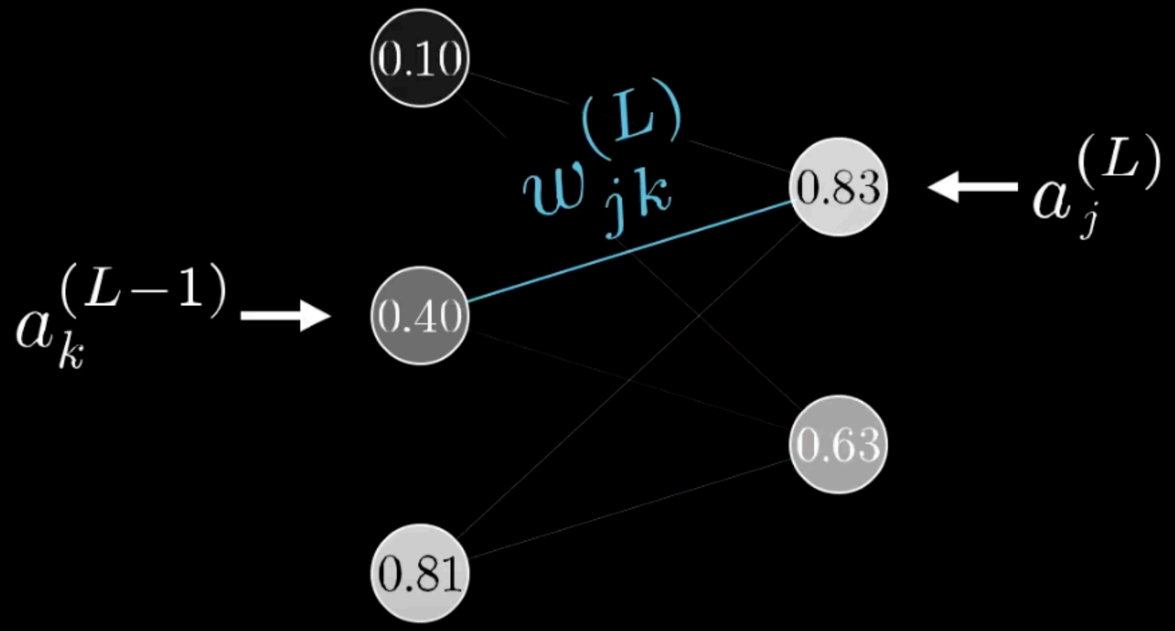
- ..it's just that it looks a little more complicated.

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

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

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

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



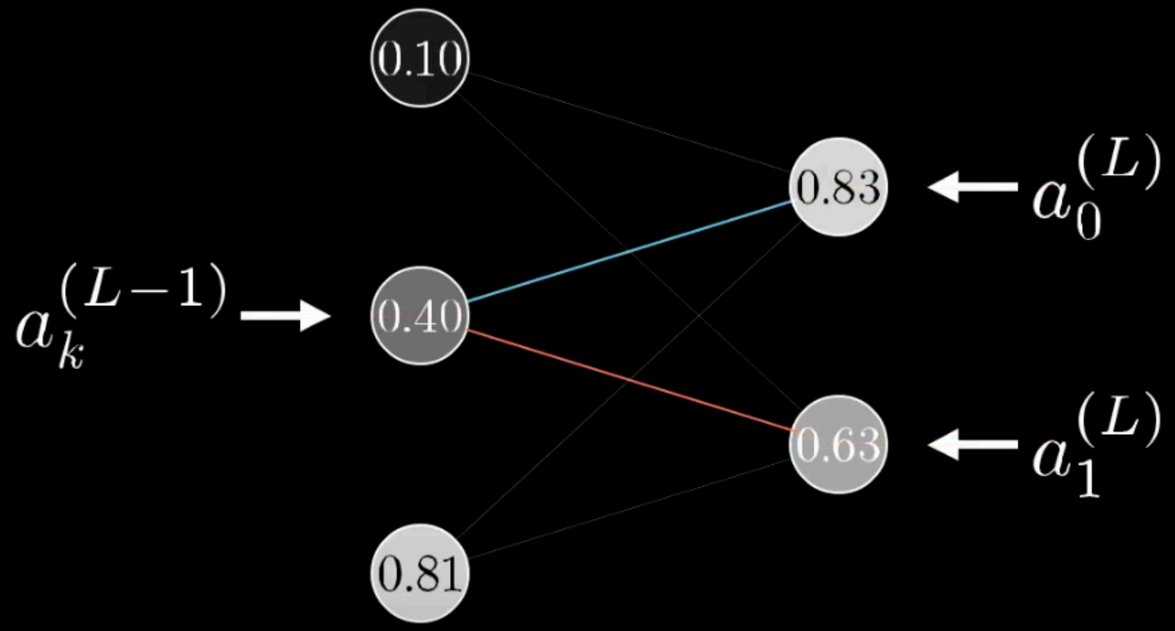
- Indeed: chain-rule derivative expression for sensitivity of cost to a specific weight (or bias) is essentially the same
- Pause and think about each of the terms

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

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

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

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



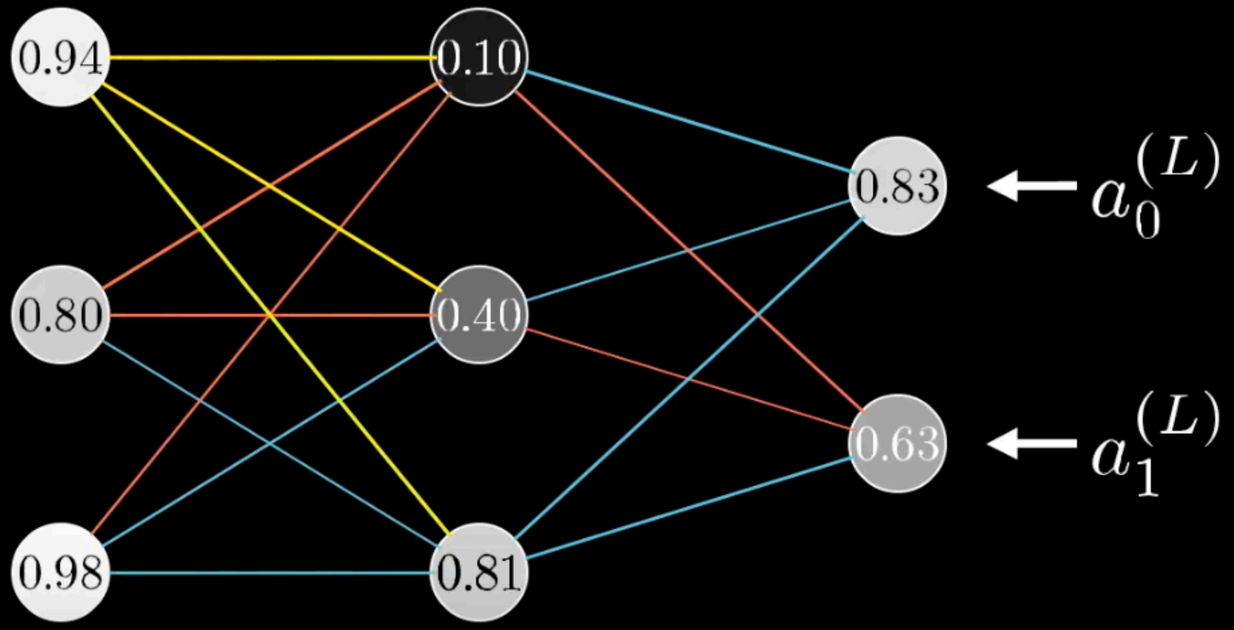
- What does **change** is derivative of **cost** wrt **pre_activations**
- Difference: pre_neuron influences cost through multiple paths
- a-k-(L-1) influences a-0-L and a-1-L which both plays a role in the cost and you have to add those up.

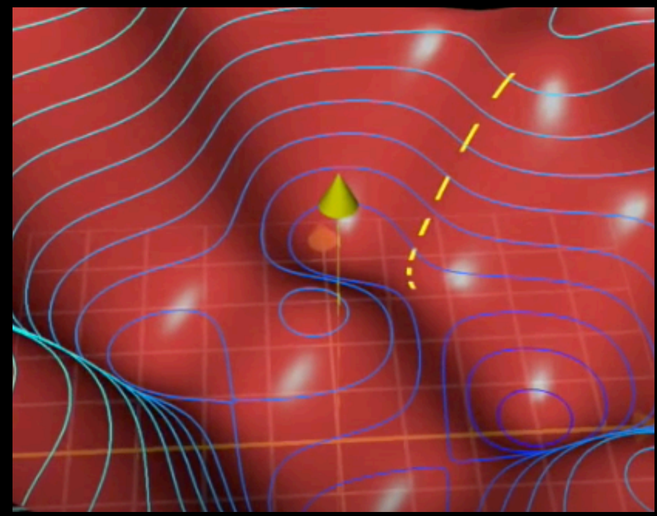
$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \underbrace{\sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}}_{\text{Sum over layer L}}$$

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

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

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





- Once you know how sensitive the cost is to the pre_activations:
  - repeat the process for all w & b feeding into the (middel) layer