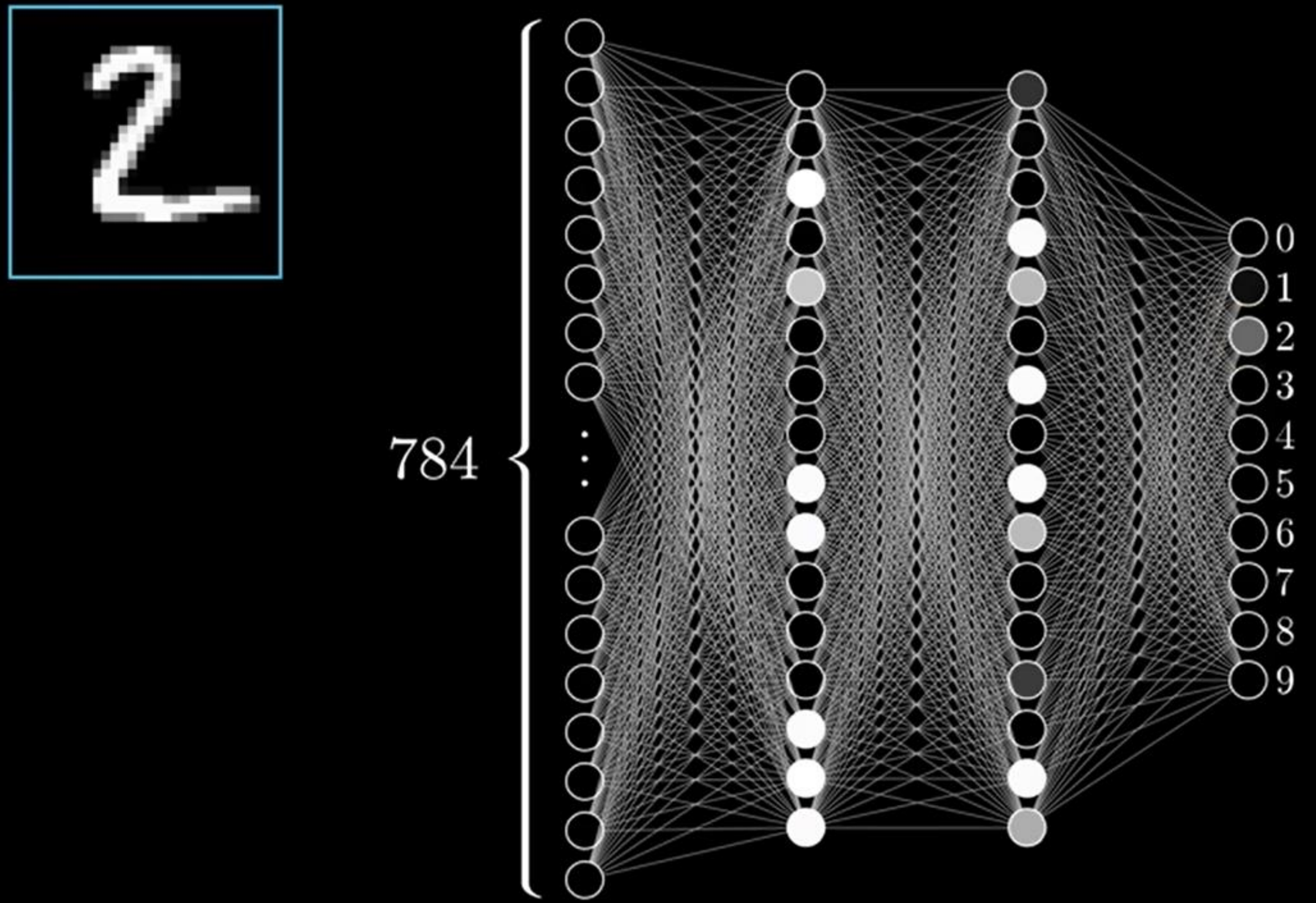
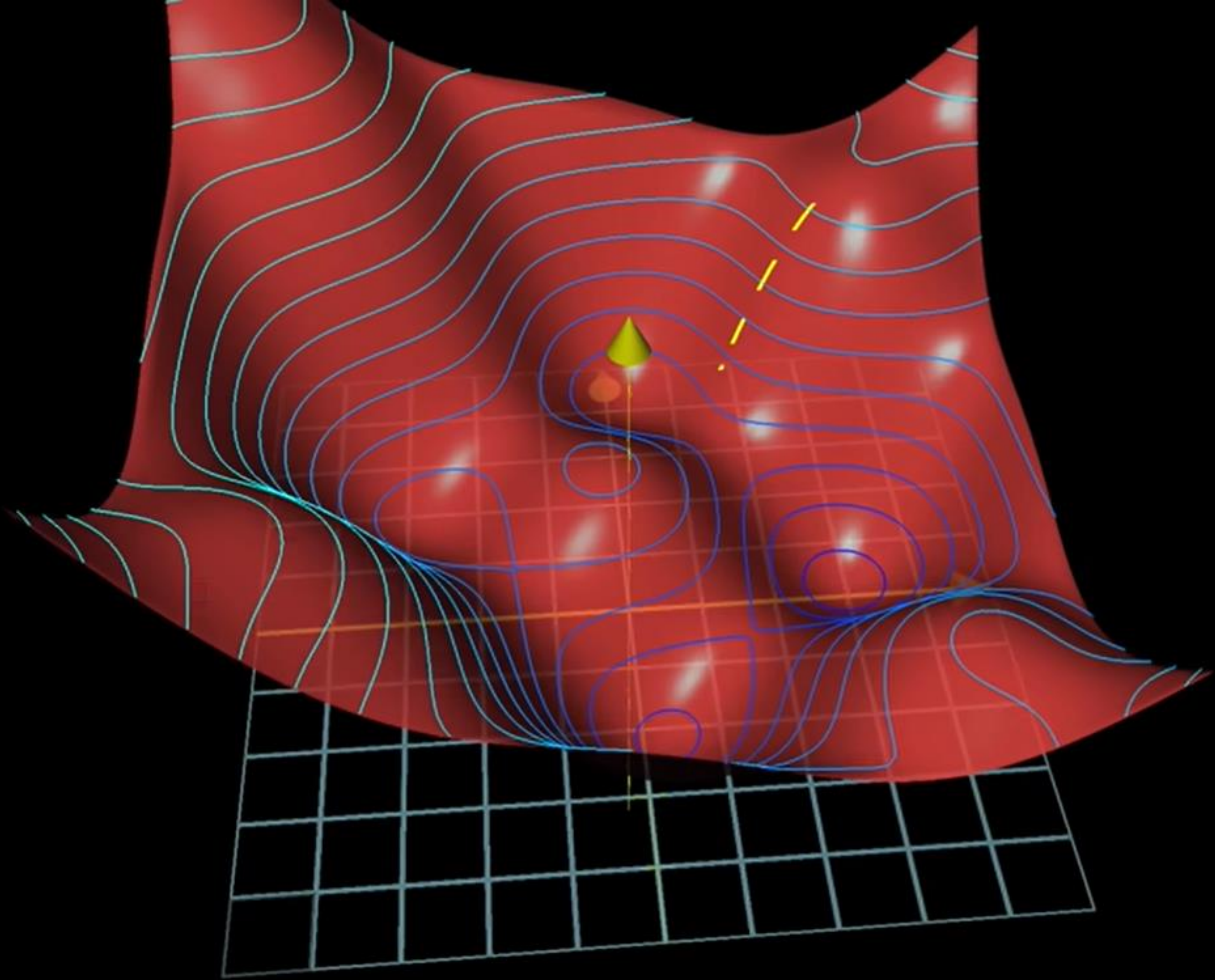
BackPropagation

Deep Learning

Slide Credits

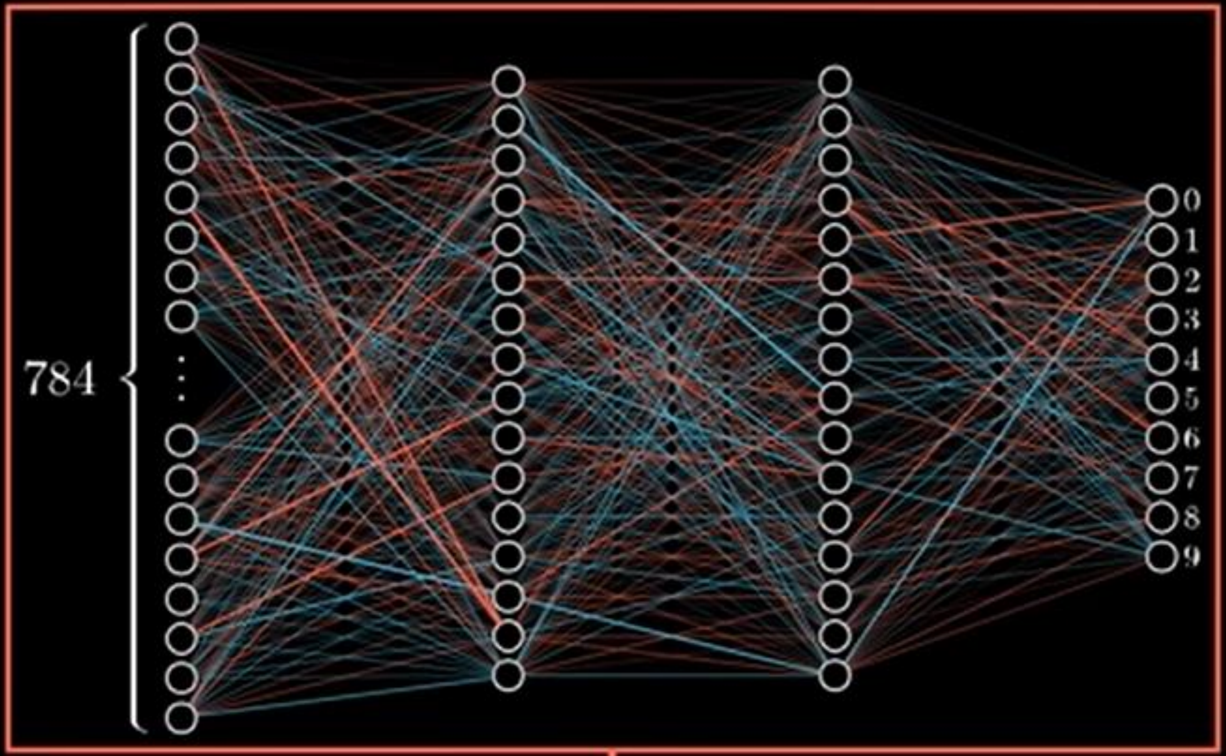
- https://www.youtube.com/watch?v=tIeHLnjs5U8&list=RDCMUCYO_jab_esuFRV4b17AJtAw&start_radio=1


784
0
1
2
3
4
5
6
7
8
9



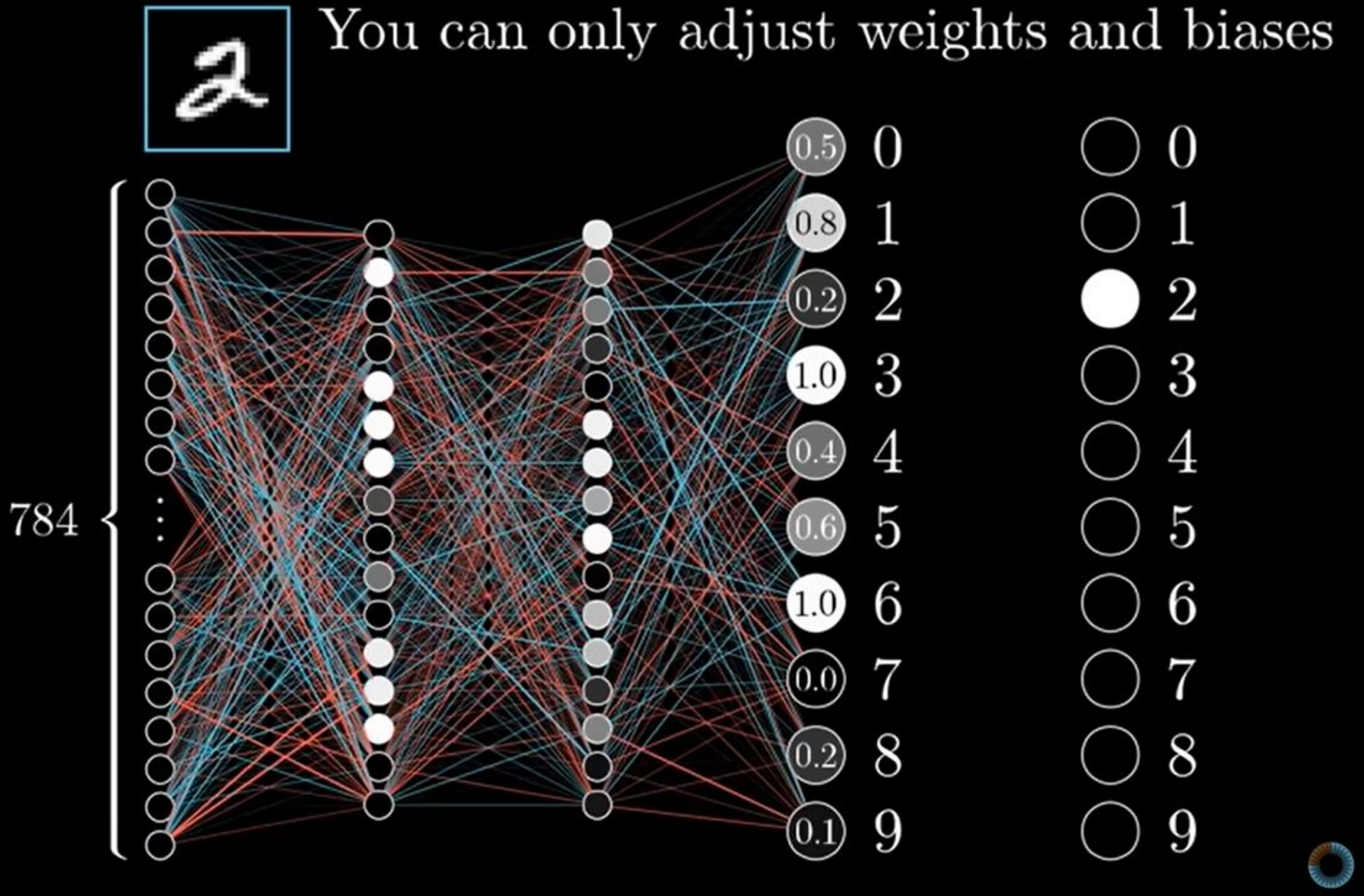
$$-\nabla C(\underbrace{\ldots}_{\text{All weights and biases}}) = \begin{bmatrix} 0.17 \\ 0.80 \\ -0.87 \\ \vdots \\ -0.04 \\ 1.58 \\ 1.59 \end{bmatrix}$$

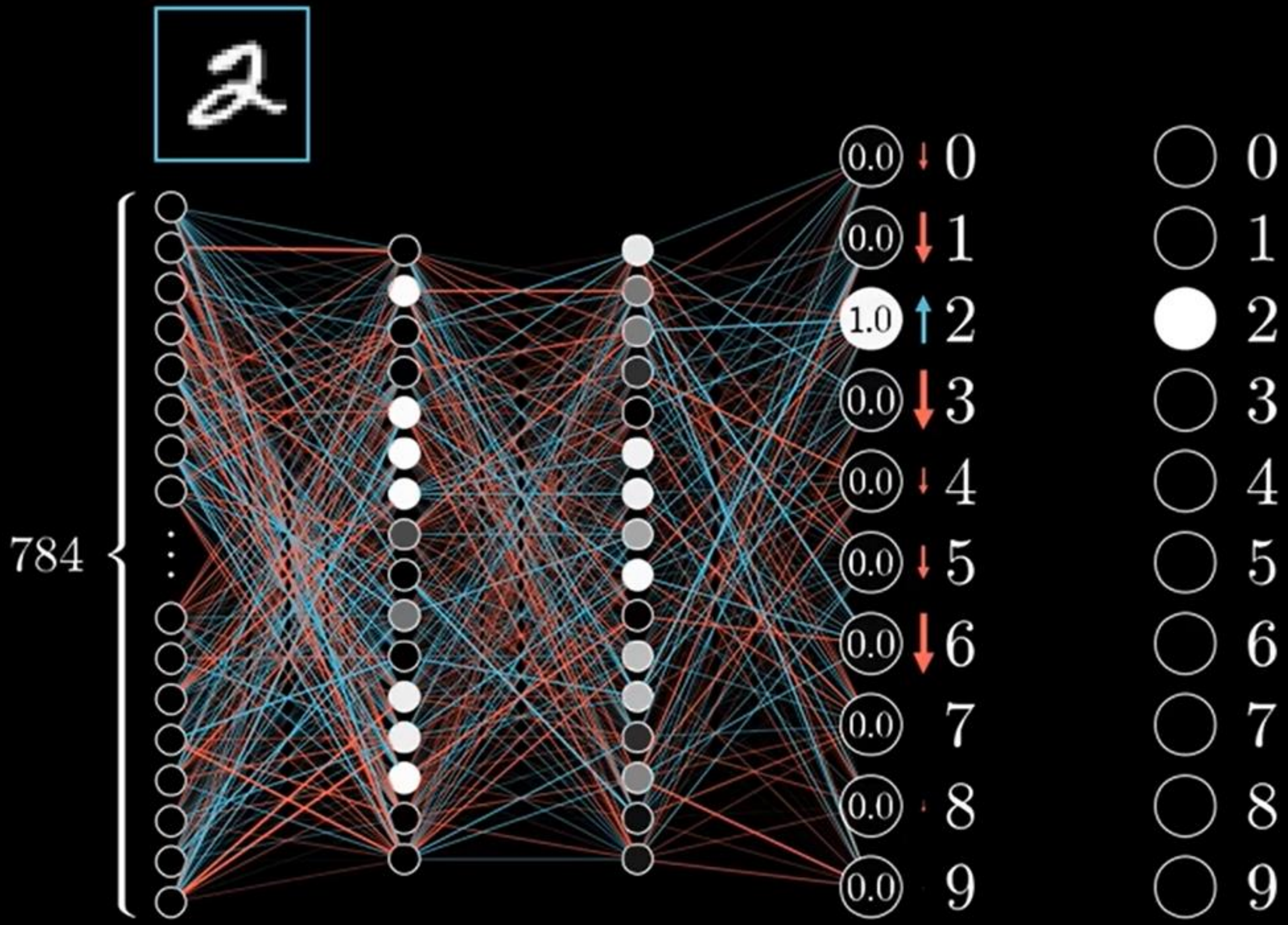
Recompute gradient



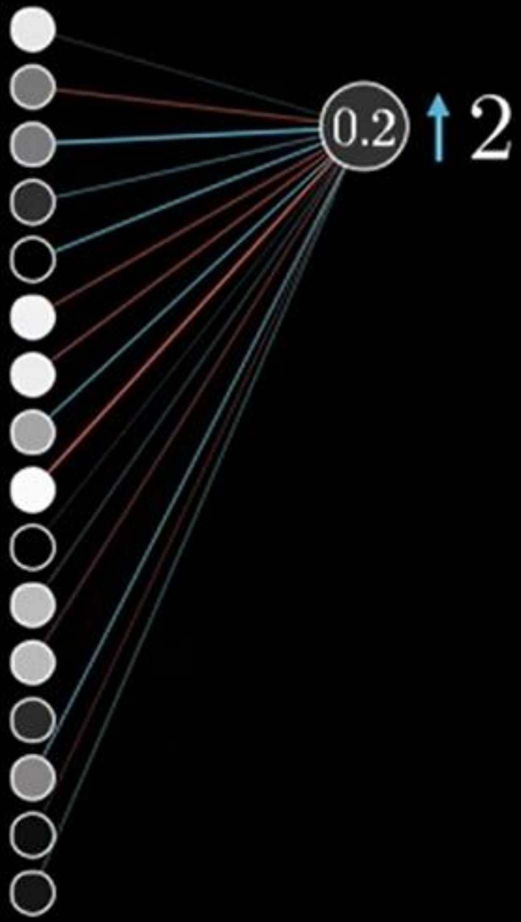
$$C(w_0, w_1, \ldots, w_{13,001}) = 3.06$$

You can only adjust weights and biases





784
0.0 0
0.0 1
1.0 2
0.0 3
0.0 4
0.0 5
0.0 6
0.0 7
0.0 8
0.0 9
0
1
2
3
4
5
6
7
8
9

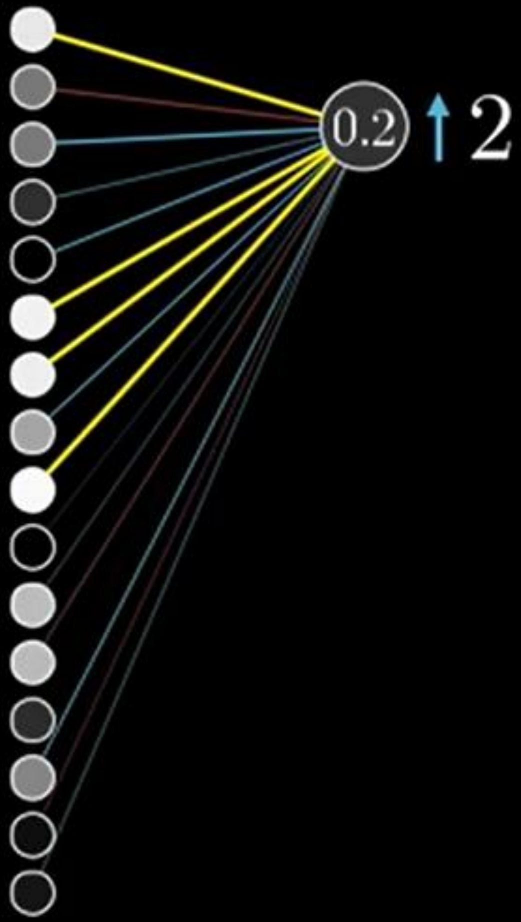

0.2
2

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

Increase b

Increase w_i

Change a_i

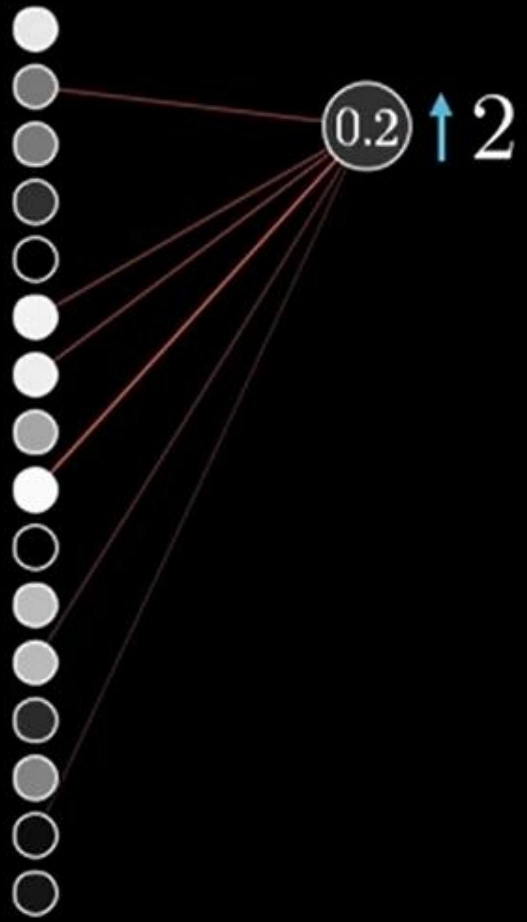


$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i
in proportion to a_i

Change a_i

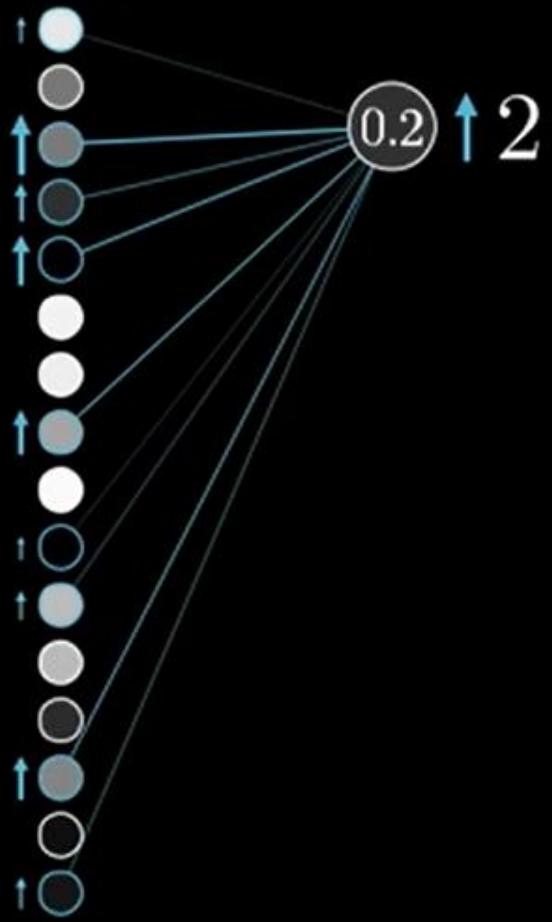


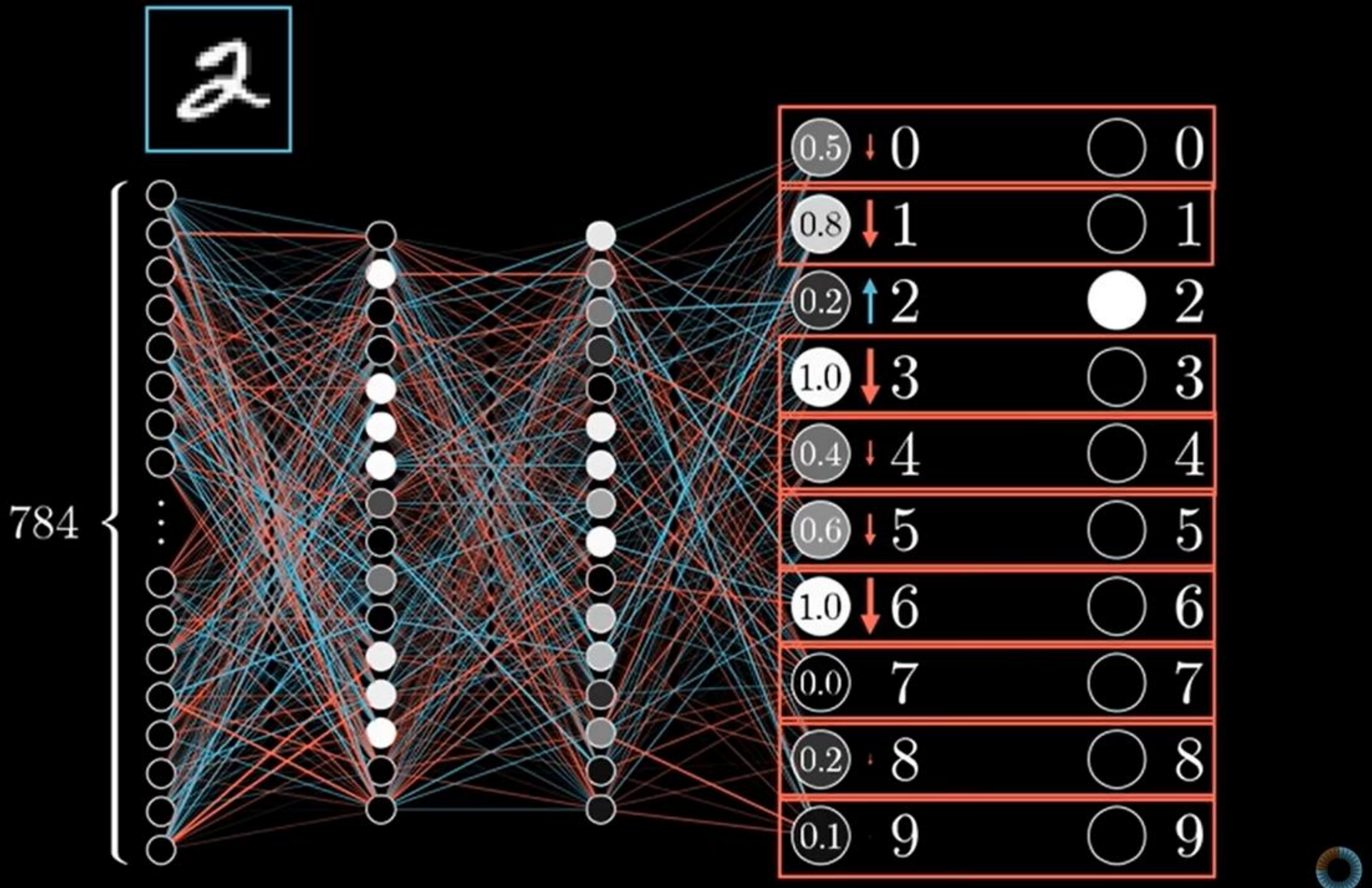
$$\text{0.2} = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i
in proportion to a_i

Change a_i



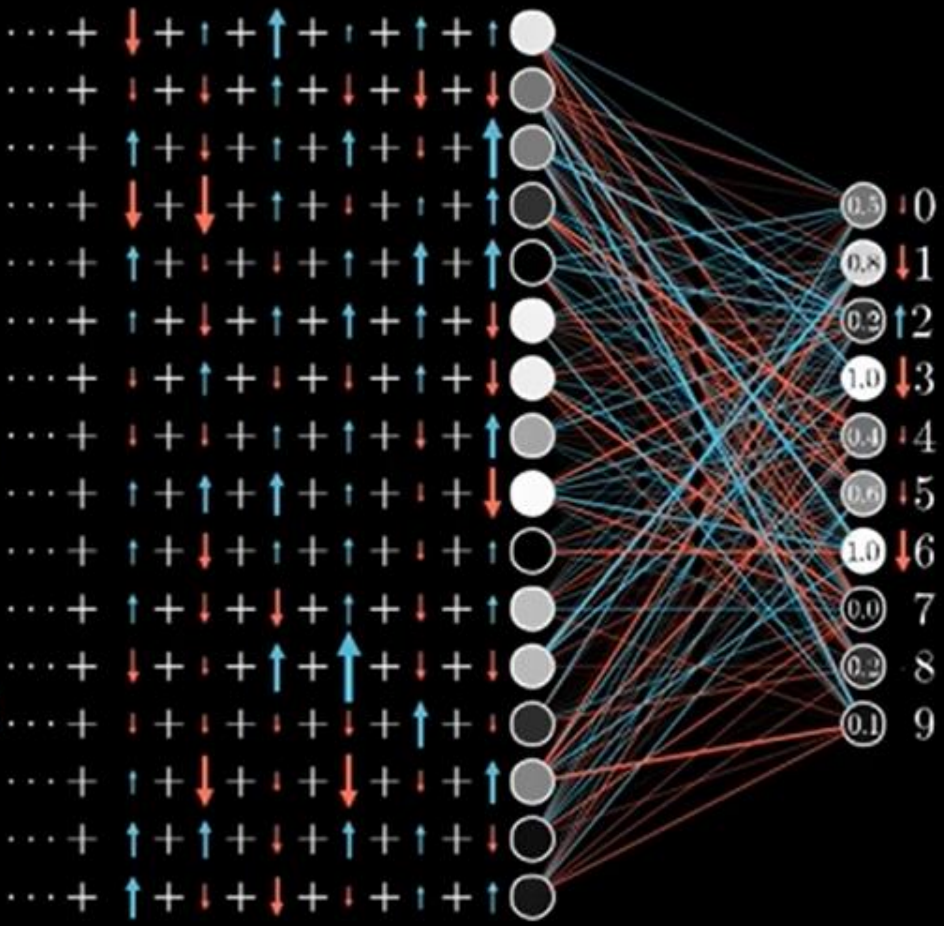

784
0.5 0
0.8 1
0.2 2
1.0 3
0.4 4
0.6 5
1.0 6
0.0 7
0.2 8
0.1 9
0
1
2
3
4
5
6
7
8
9

Propagate backwards

Increase b

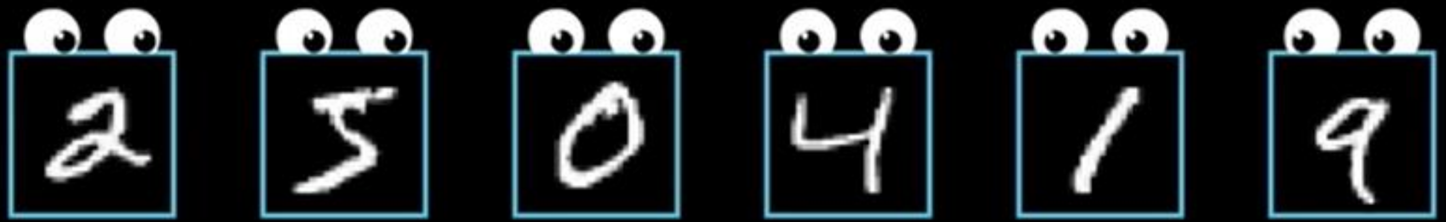
Increase w_i
in proportion to a_i

Change a_i
in proportion to w_i



							$\cdots$
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	$+0.04$	$\cdots$

	2	5	0	4	1	9	⋯	Average over all training data
w_0	−0.08	+0.02	−0.02	+0.11	−0.05	−0.14	⋯	→ −0.08
w_1	−0.11	+0.11	+0.07	+0.02	+0.09	+0.05	⋯	→ +0.12
w_2	−0.07	−0.04	−0.01	+0.02	+0.13	−0.15	⋯	→ −0.06
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$w_{13,001}$	+0.13	+0.08	−0.06	−0.09	−0.02	+0.04	⋯	→ +0.04



$\cdots$

Average over
all training data

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

Code

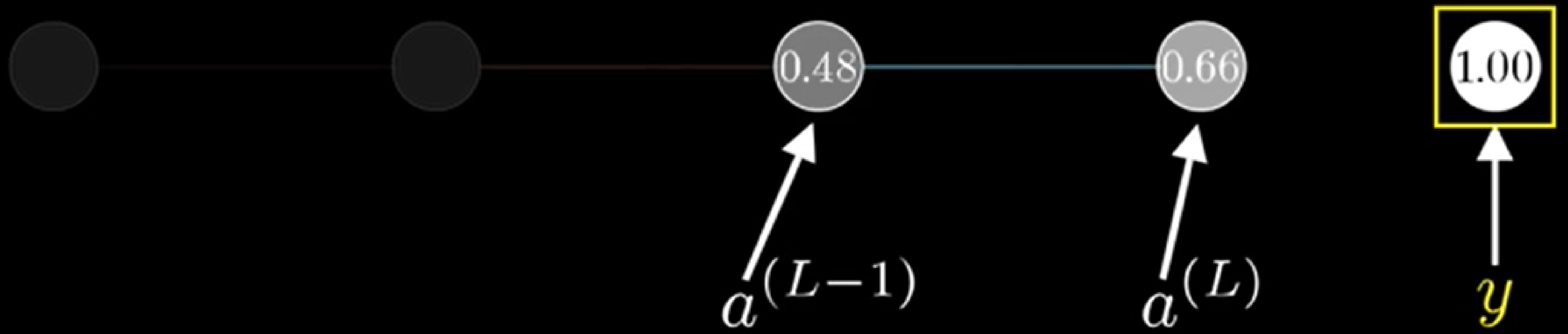
```python
def backprop(self, x, y):
    """Return a tuple ``(nabla_b, nabla_w)`` representing the
    gradient for the cost function C_x.  ``nabla_b`` and
    ``nabla_w`` are layer-by-layer lists of numpy arrays, similar
    to ``self.biases`` and ``self.weights``."""
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    # feedforward
    activation = x
    activations = [x] # list to store all the activations, layer by layer
    zs = [] # list to store all the z vectors, layer by layer
    for b, w in zip(self.biases, self.weights):
        z = np.dot(w, activation)+b
        zs.append(z)
        activation = self.non_linearity(z)
        activations.append(activation)
    # backward pass
    delta = self.cost_derivative(activations[-1], y) * \
        self.d_non_linearity(zs[-1])
    nabla_b[-1] = delta
    nabla_w[-1] = np.dot(delta, activations[-2].transpose())
    # Note that the variable l in the loop below is used a little
    # differently to the notation in Chapter 2 of the book.  Here,
    # l = 1 means the last layer of neurons, l = 2 is the
    # second-last layer, and so on.  It's a renumbering of the
    # scheme in the book, used here to take advantage of the fact
    # that Python can use negative indices in lists.
    for l in xrange(2, self.num_layers):
        z = zs[-l]
        sp = self.d_non_linearity(z)
        delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
        nabla_b[-l] = delta
        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
    return (nabla_b, nabla_w)
```

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

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

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

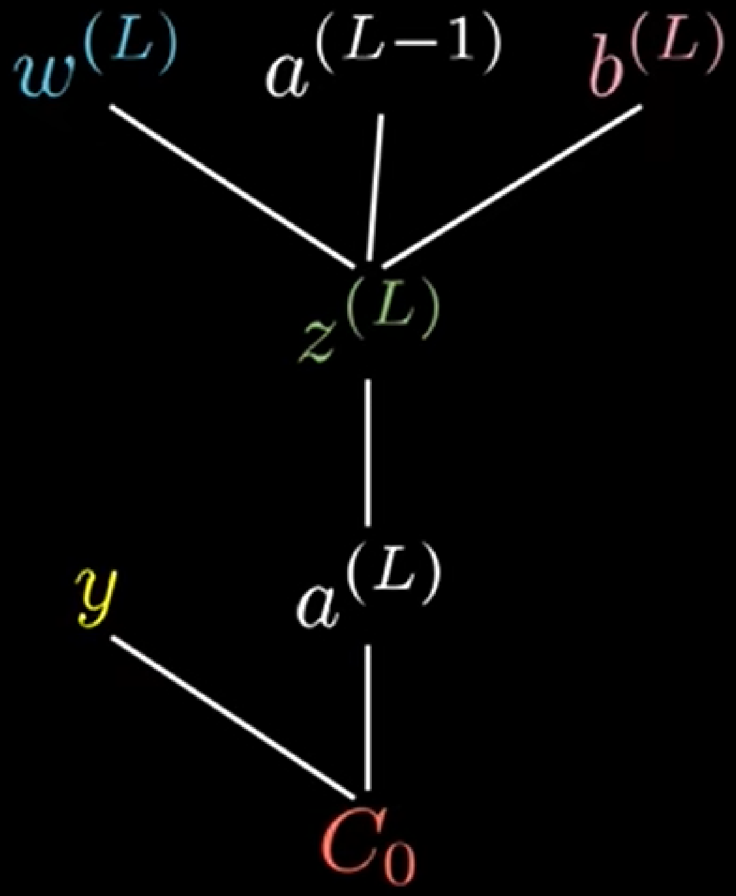
Desired output

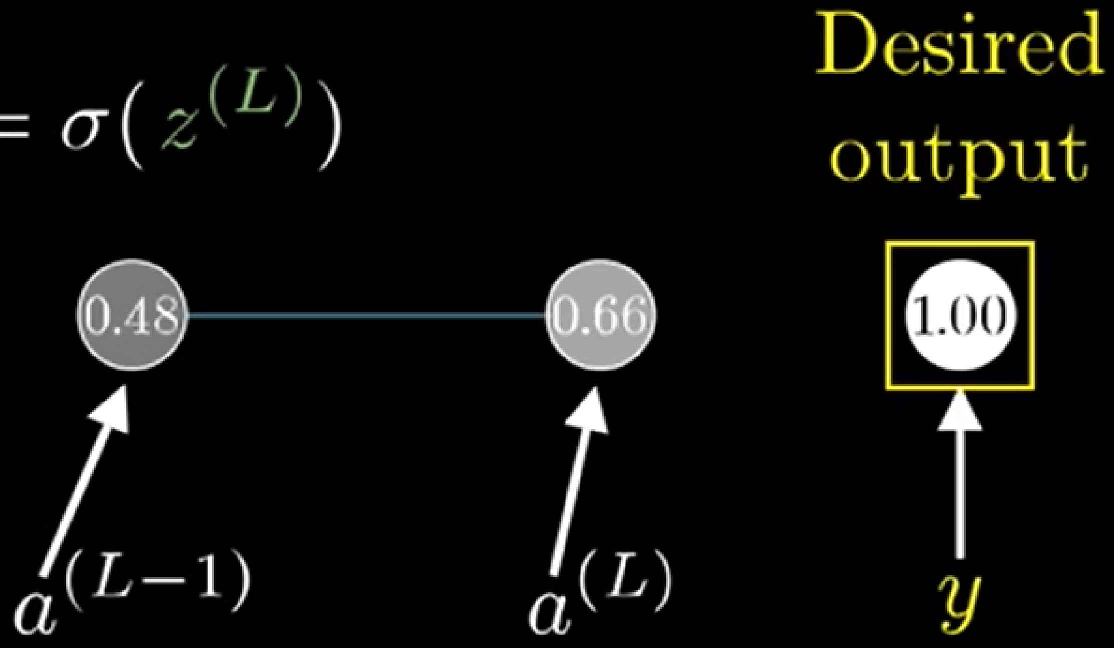


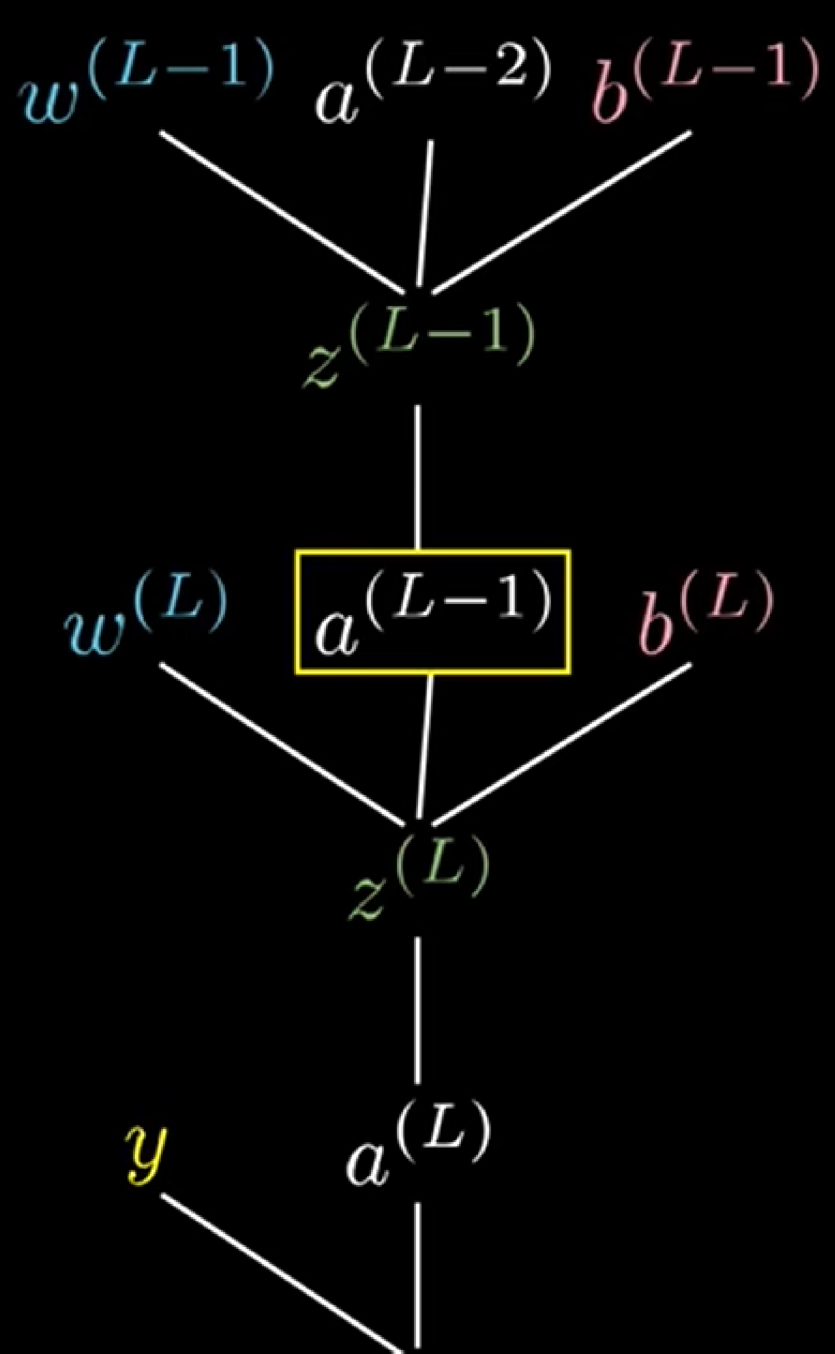
$$\text{Cost} \longrightarrow C_0(\ldots) = (a^{(L)} - y)^2$$

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

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



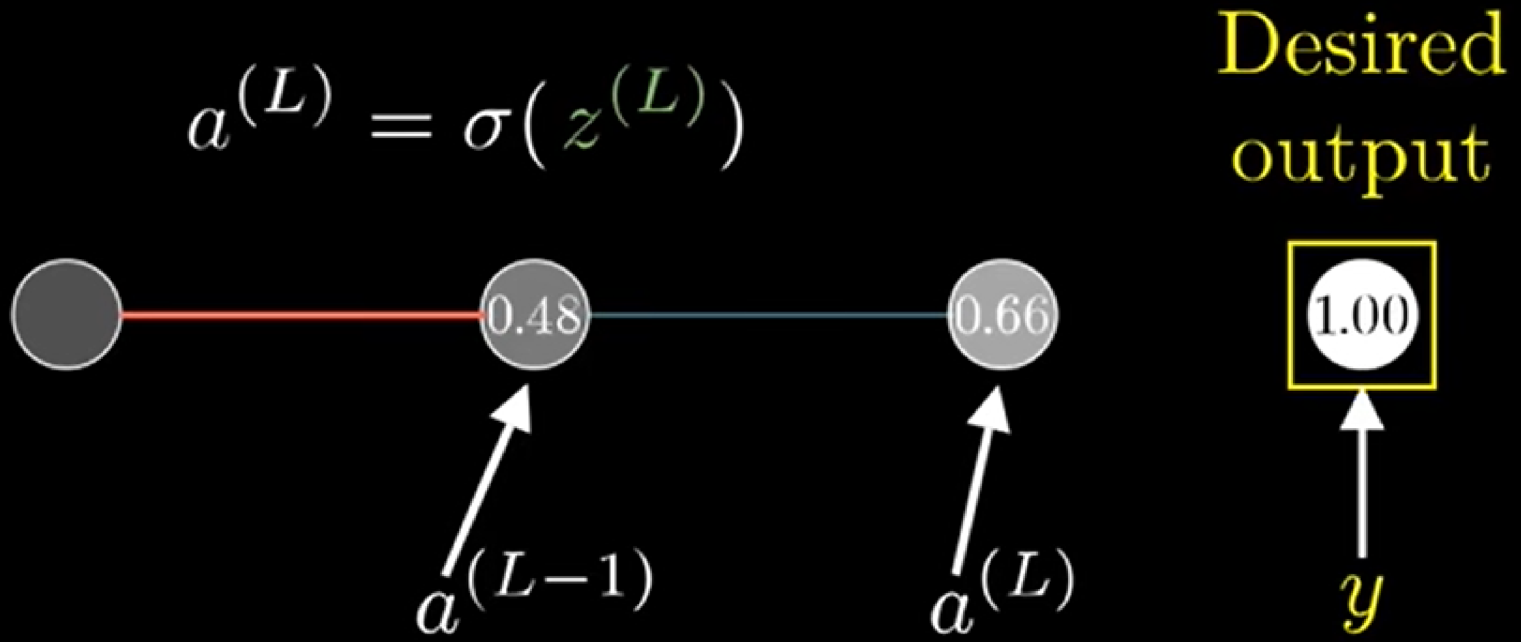




Cost $\longrightarrow C_0(\dots) = (a^{(L)} - y)^2$

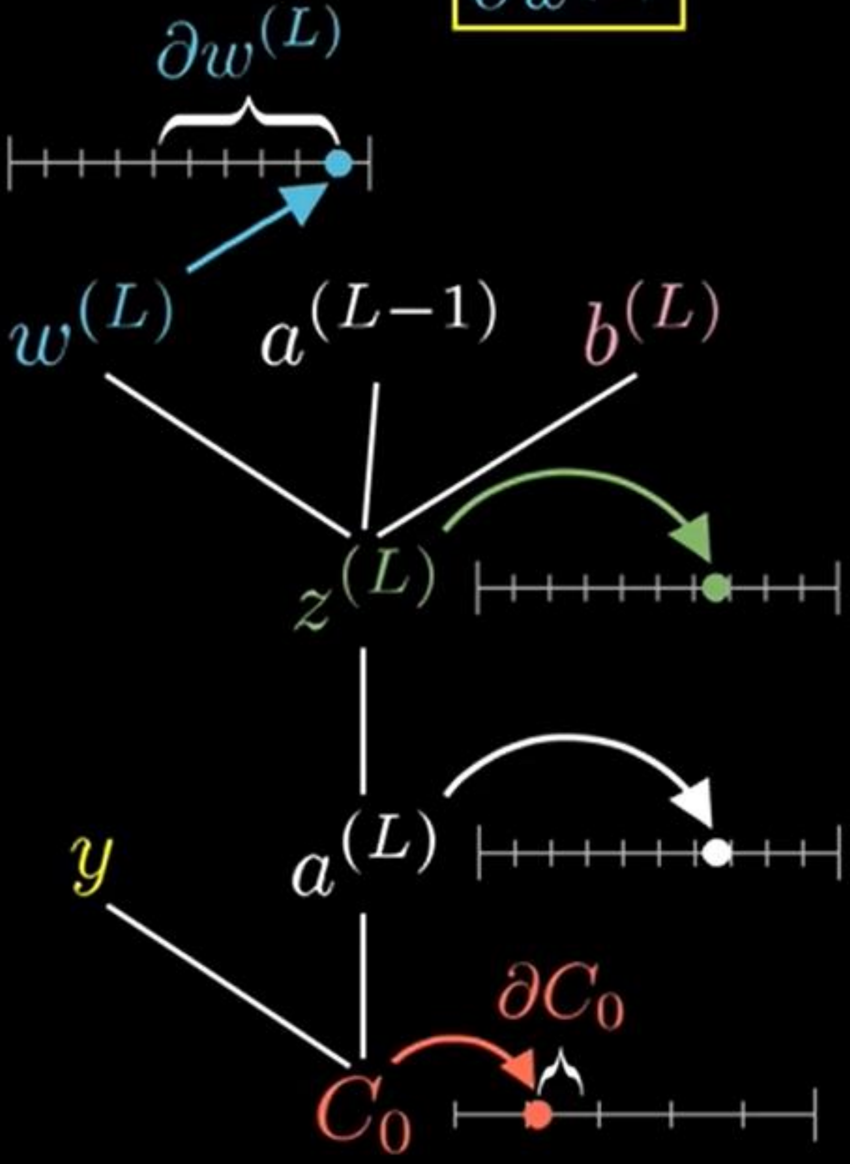
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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



$$\boxed{\frac{\partial C_0}{\partial w^{(L)}}}$$ What we want

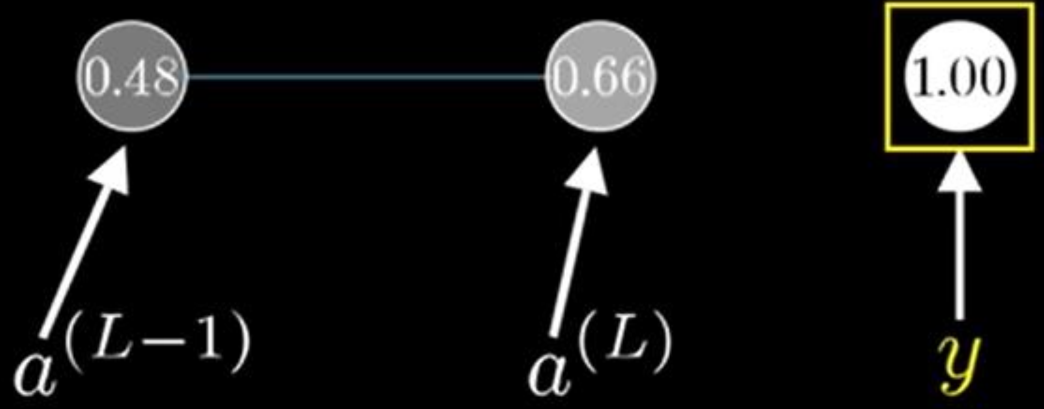
$$C_0(\dots) = (a^{(L)} - y)^2$$



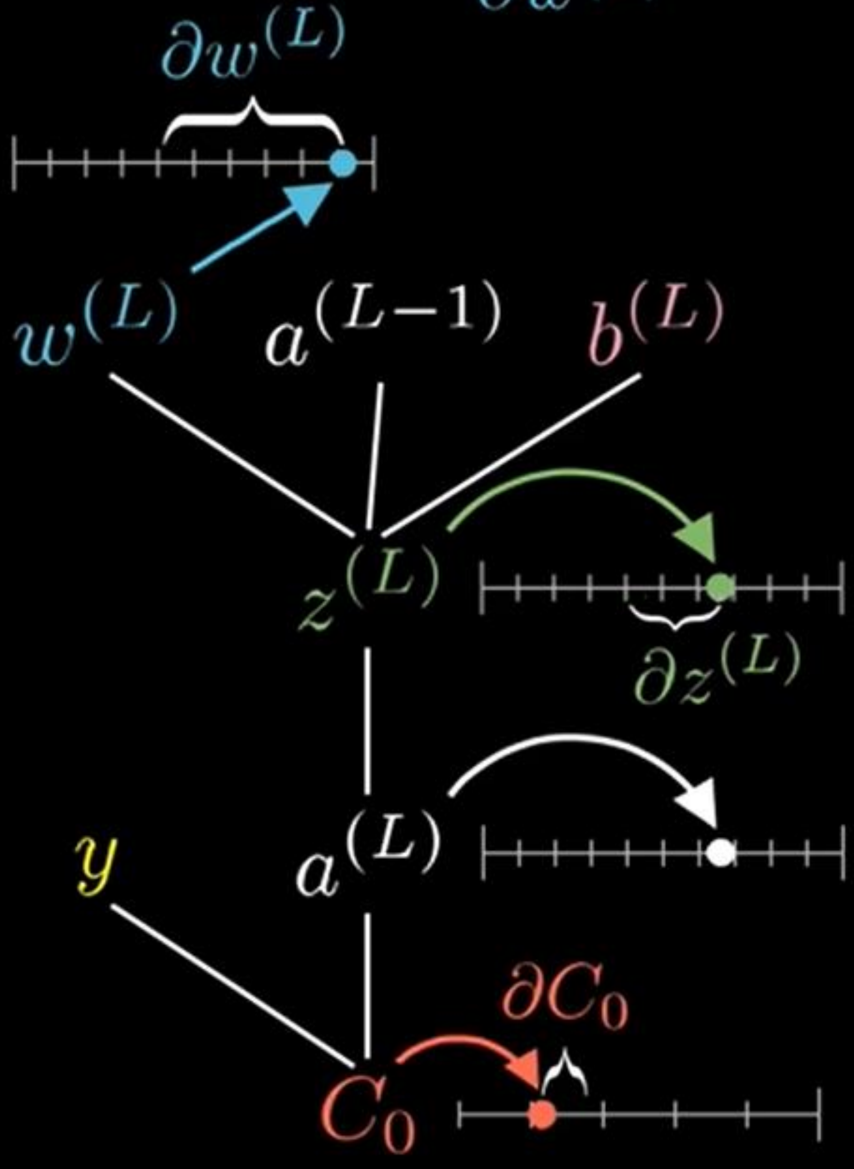
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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

Desired output



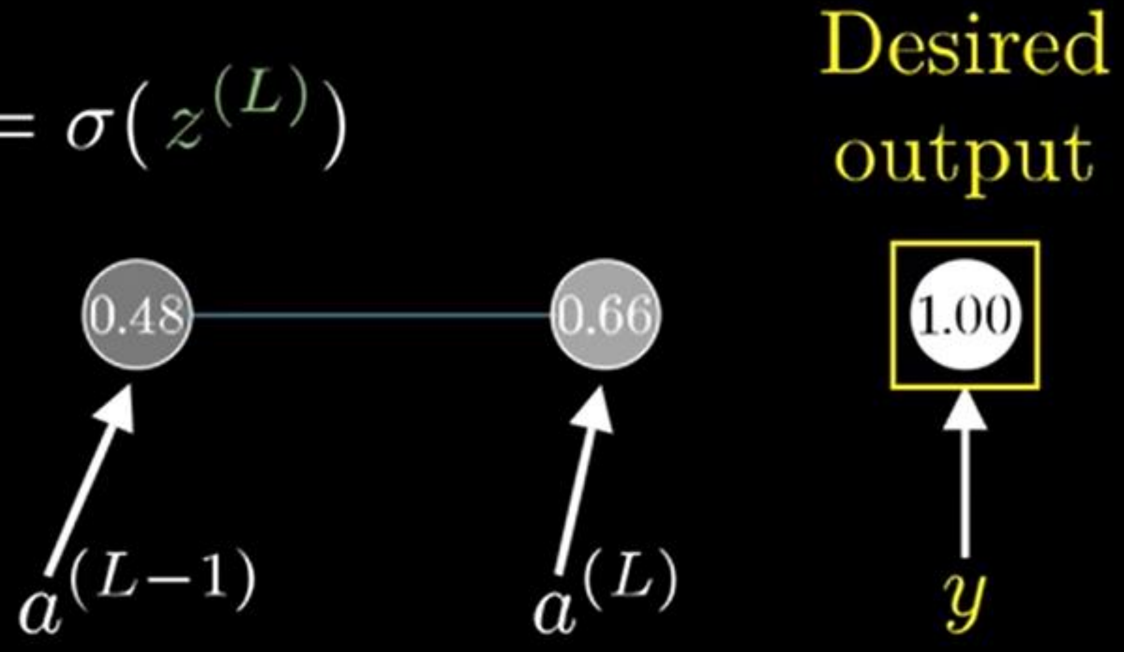
$$\frac{\partial C_0}{\partial w^{(L)}}$$



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

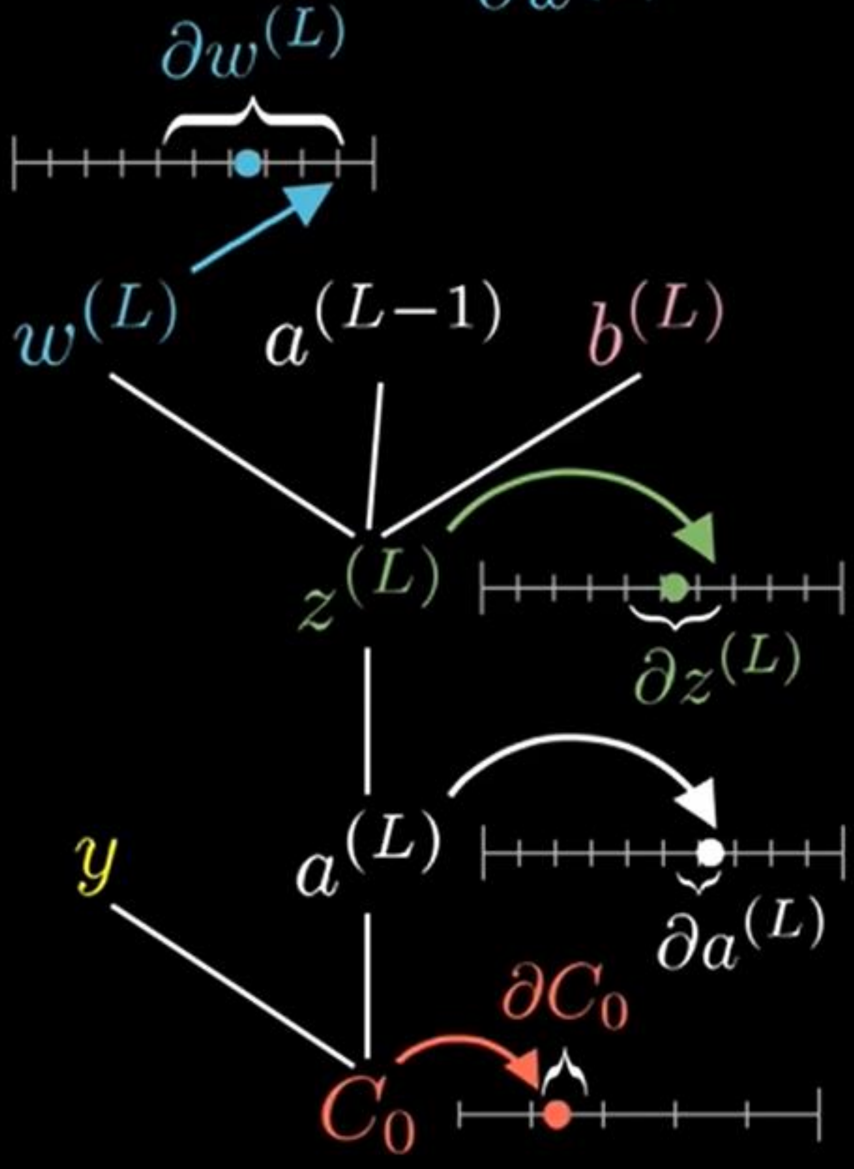
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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



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

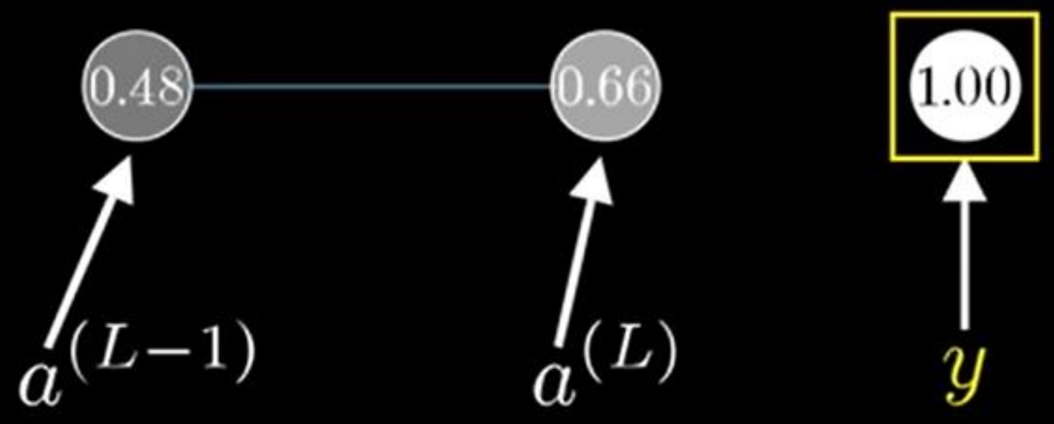
$$C_0(\dots) = (a^{(L)} - y)^2$$



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

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

Desired output



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

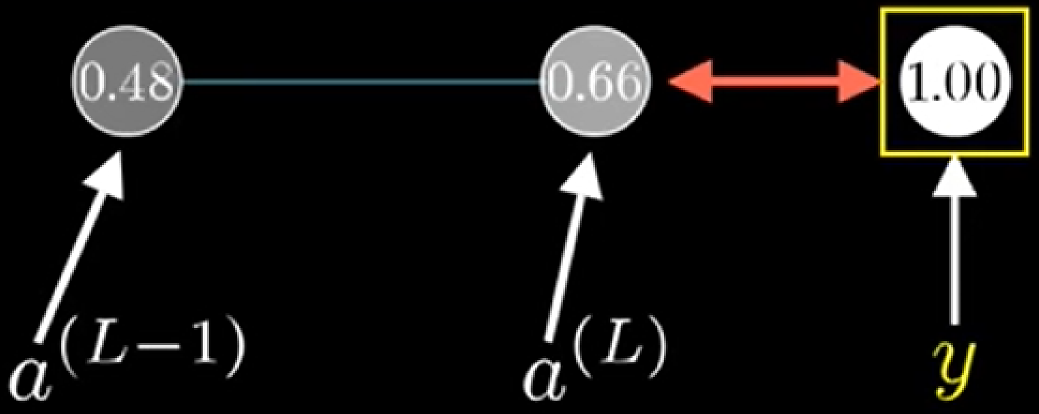
$$C_0 = (a^{(L)} - y)^2$$

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

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

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

$$0.66 - 1.00$$



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

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

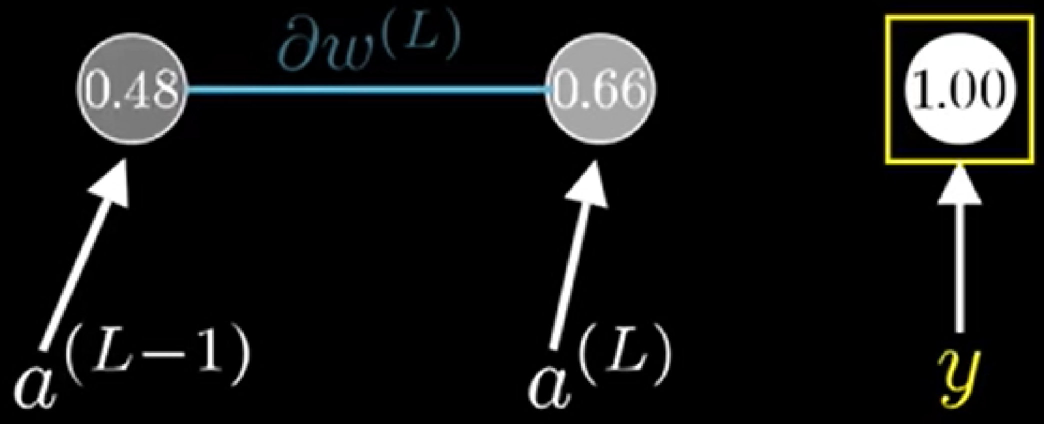
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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

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

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

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



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

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

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

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

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

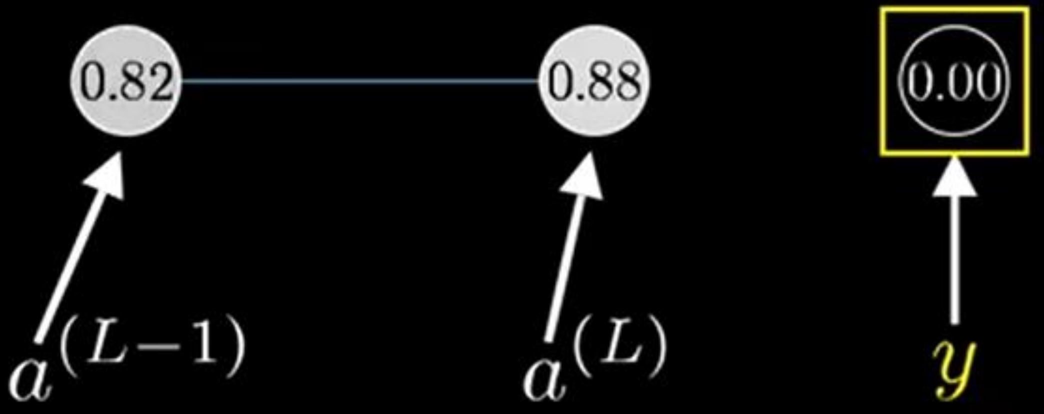
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

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

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

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

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



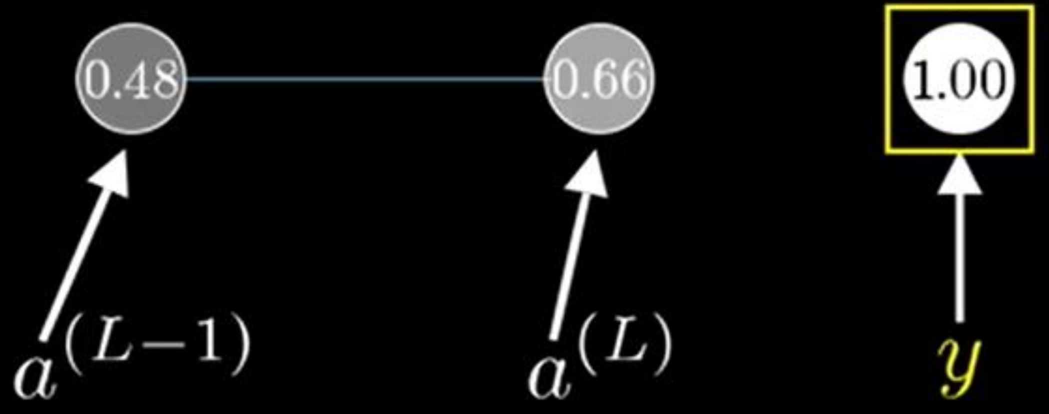
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

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

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

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

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

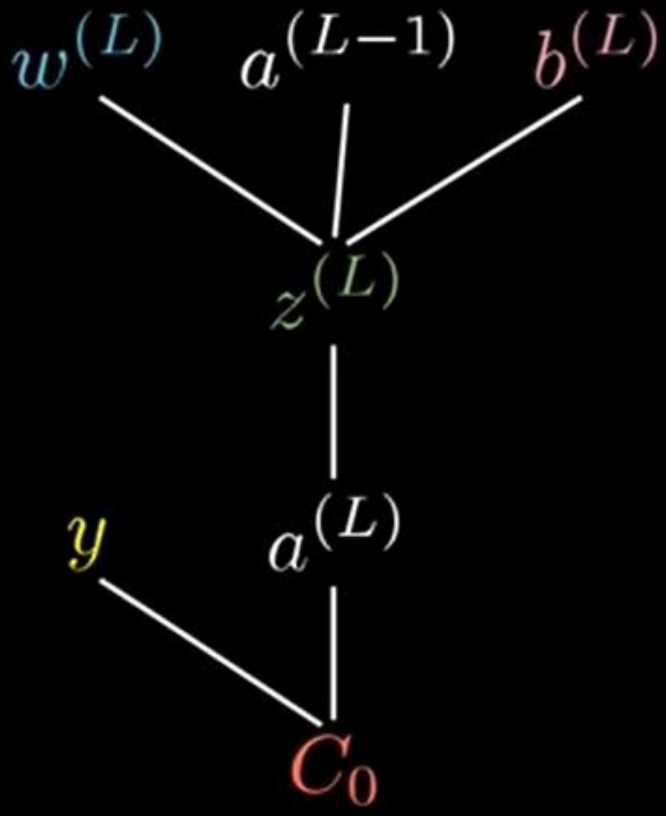


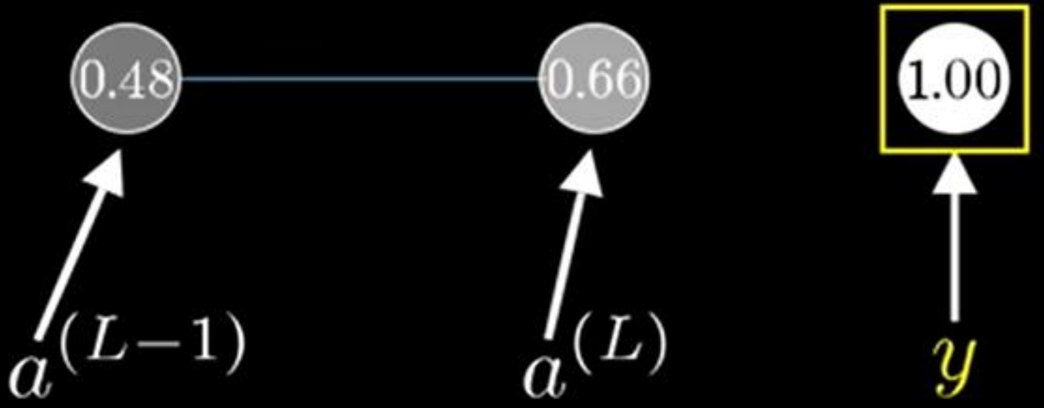
$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = w^{(L)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

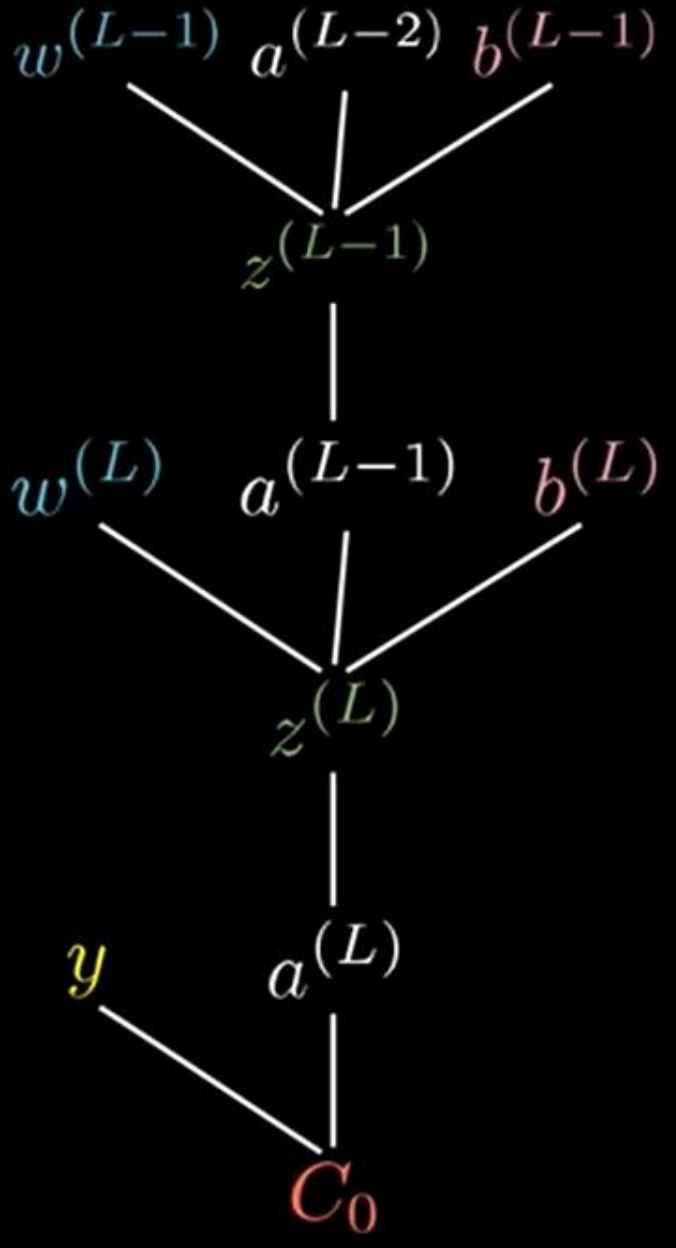
$$C_0 = (a^{(L)} - y)^2$$

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

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





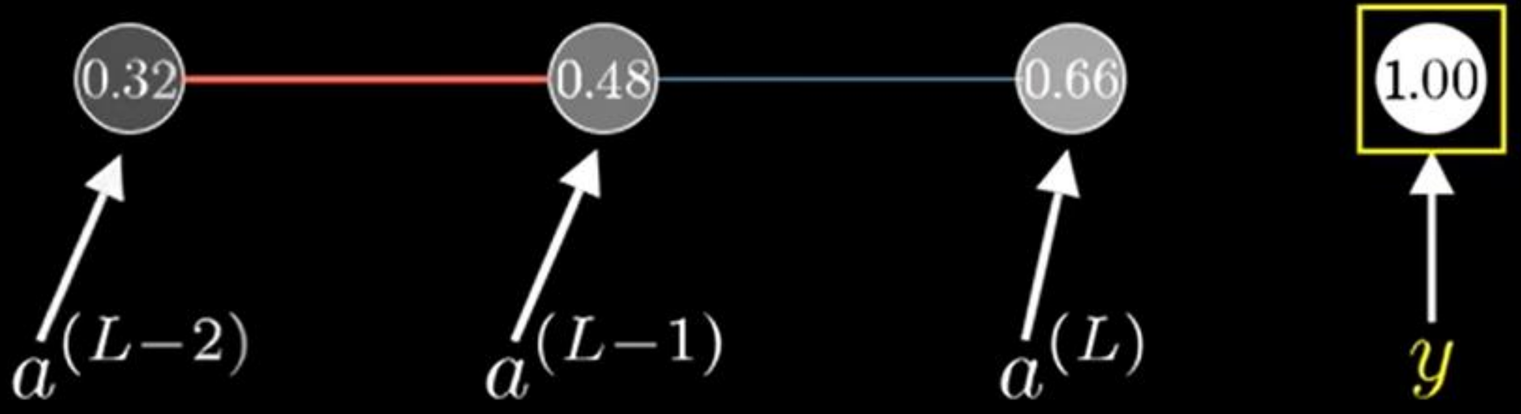


$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

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

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

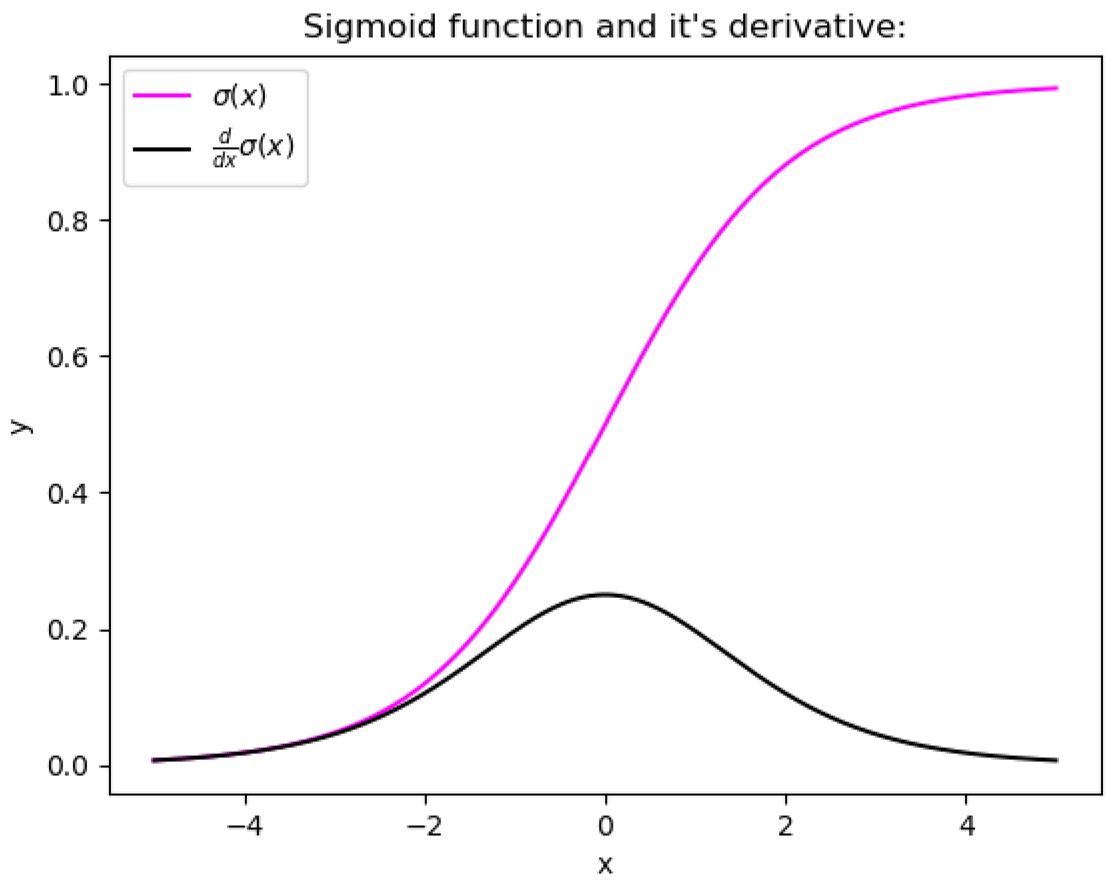
$$a^{(L)} = \sigma(z^{(L)})$$



Problem: Vanishing Gradient

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

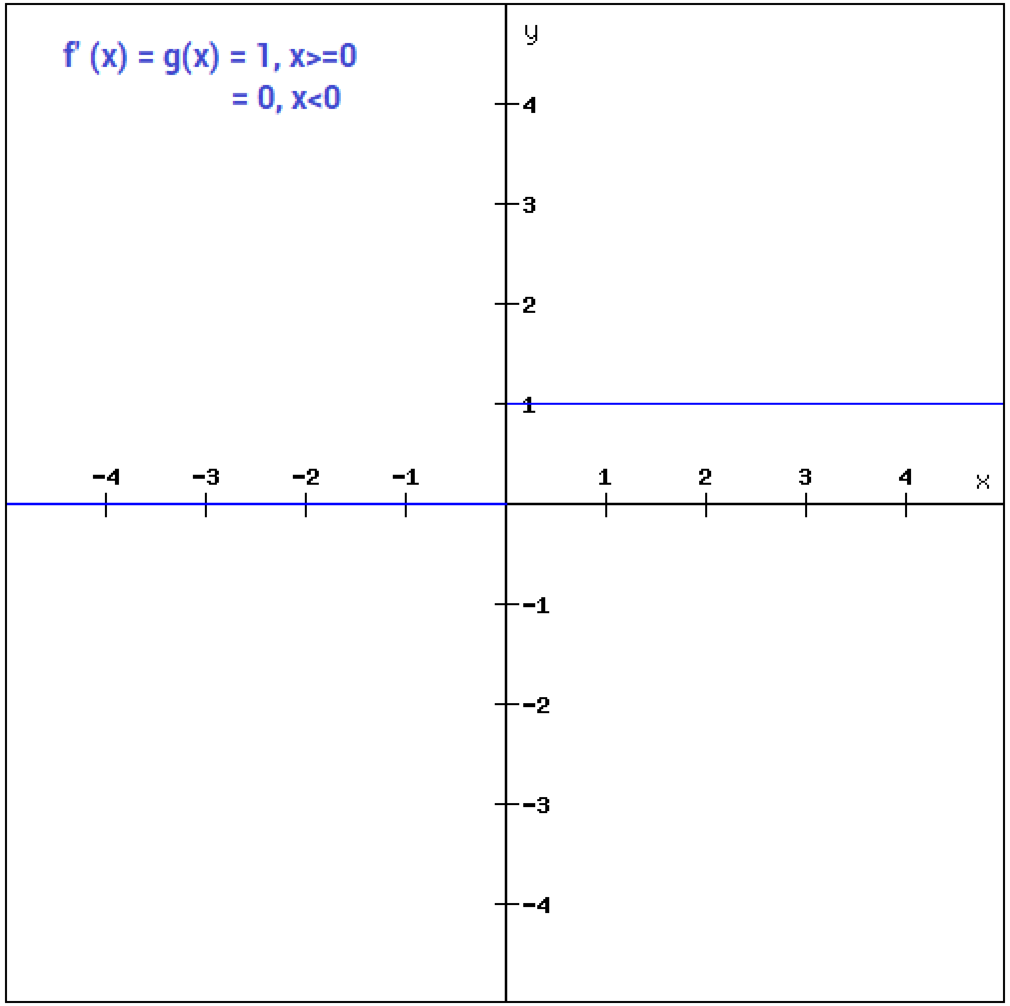
Or, bad Property



A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

How to fix it?

- Simple: Use ReLU



However

- ReLU based networks suffer from Exploding gradient problem.