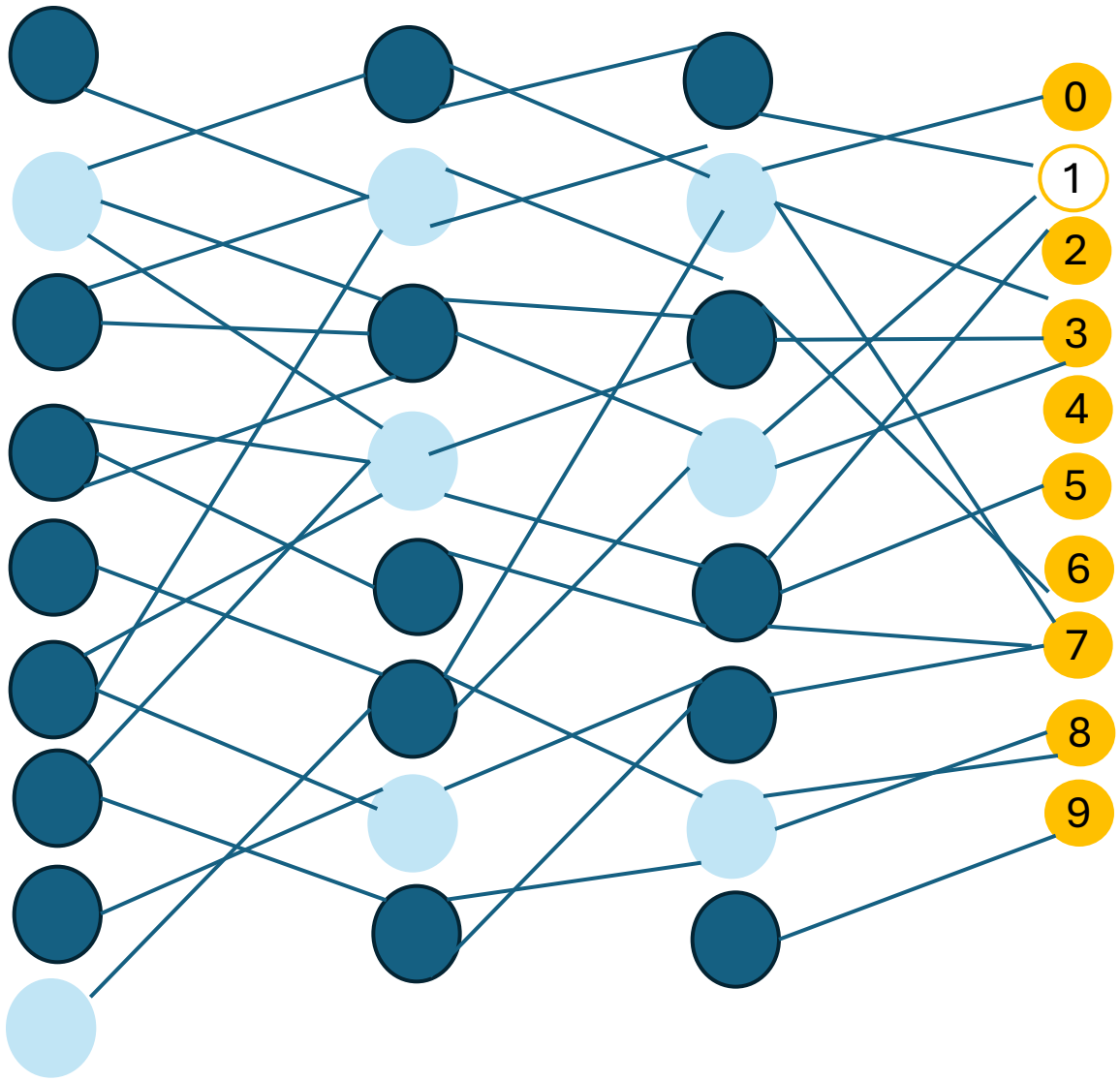
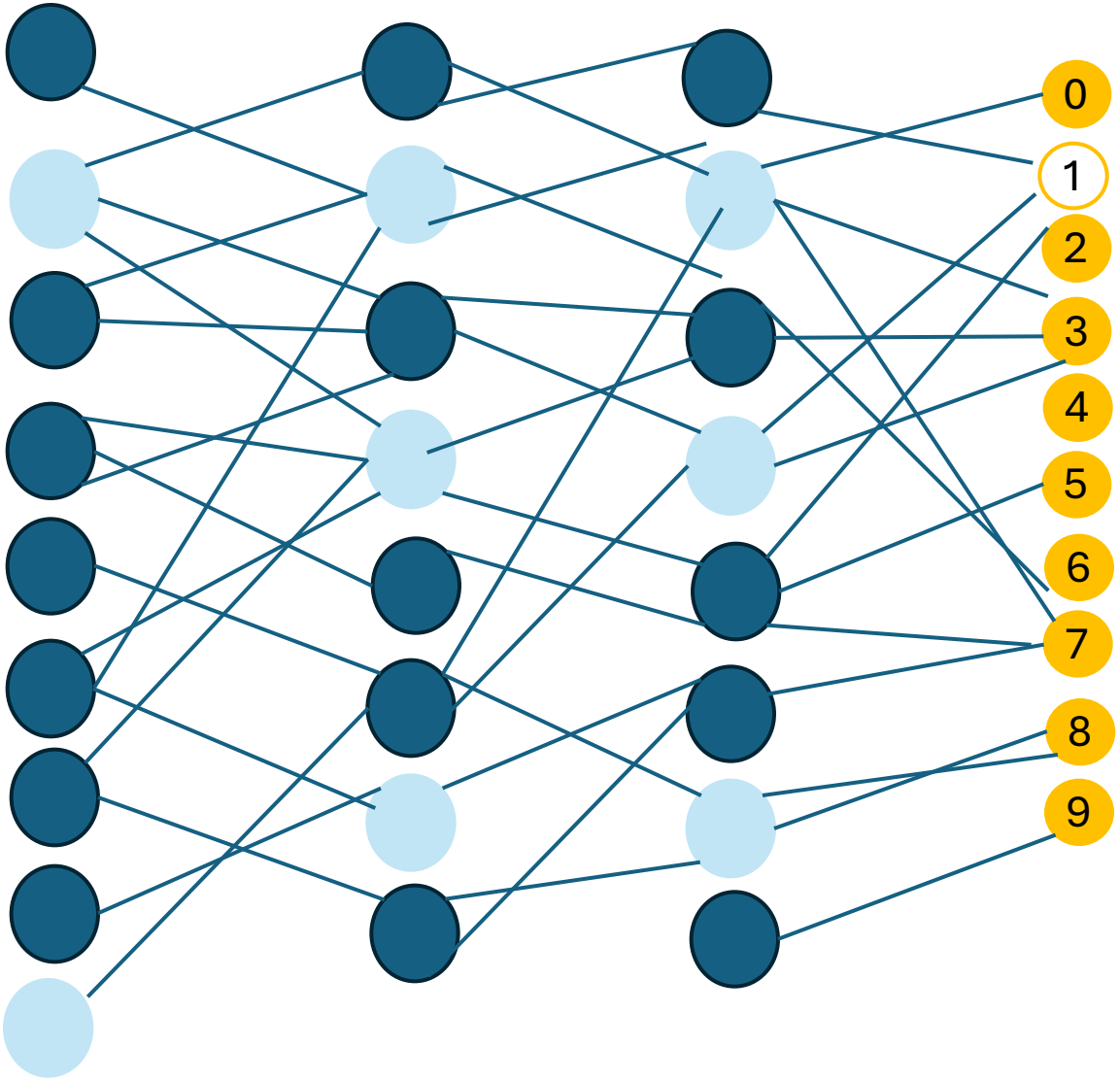
Introduction to Neural Network

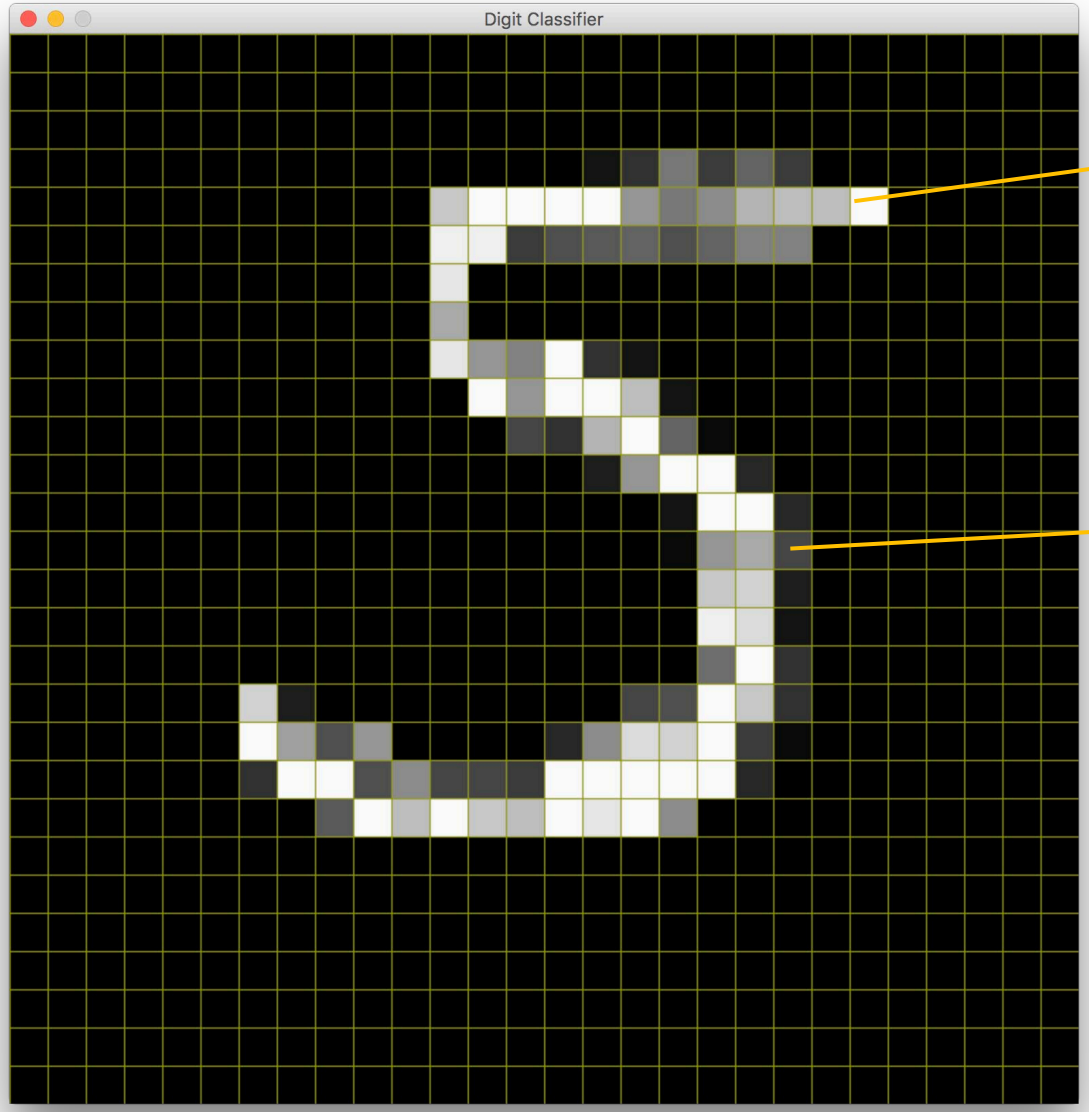
Multilayer Perceptron



What is a Neuron → Thing that holds a number



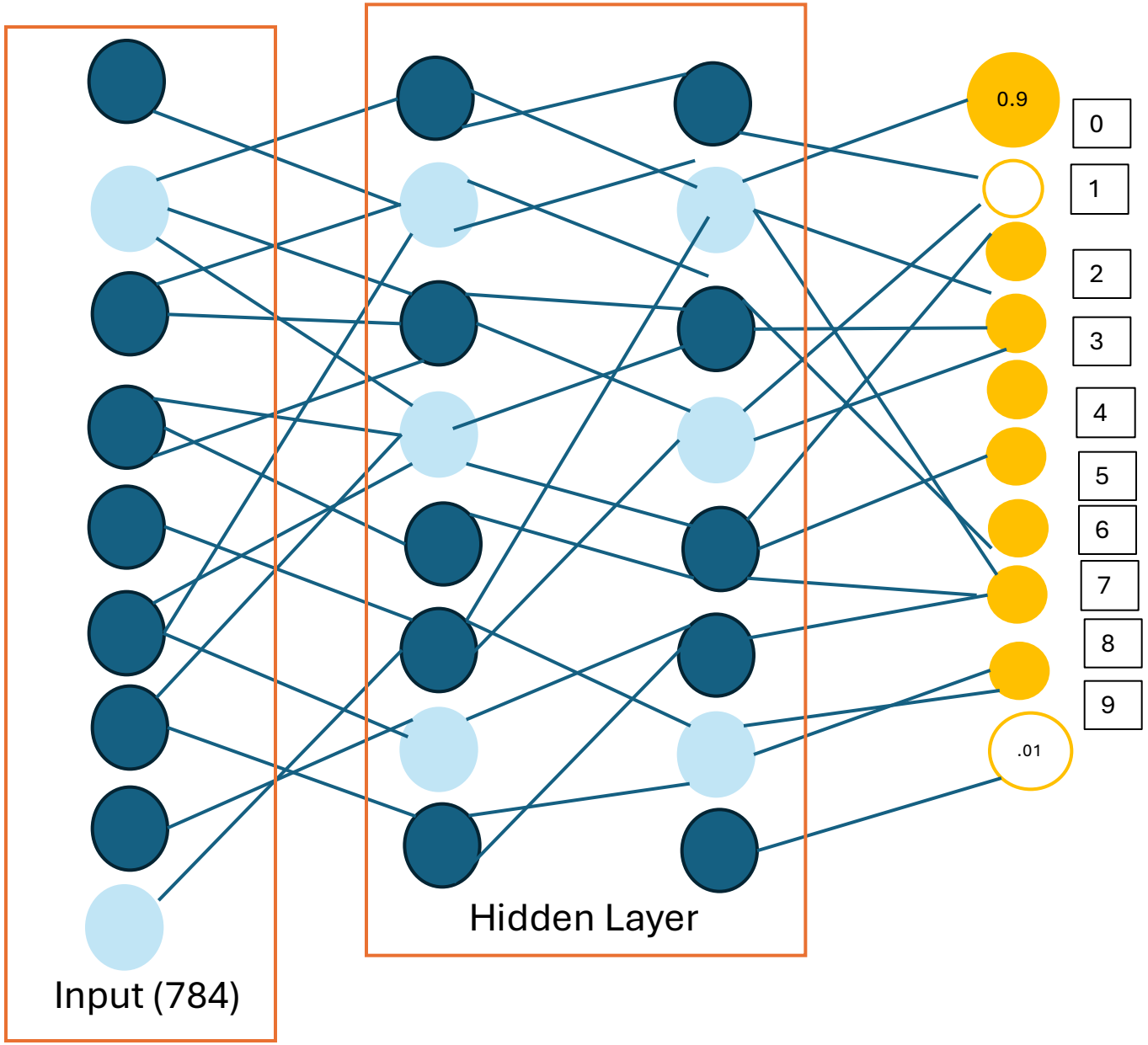
28X28 pixels with (0-1) values **Input Layer**



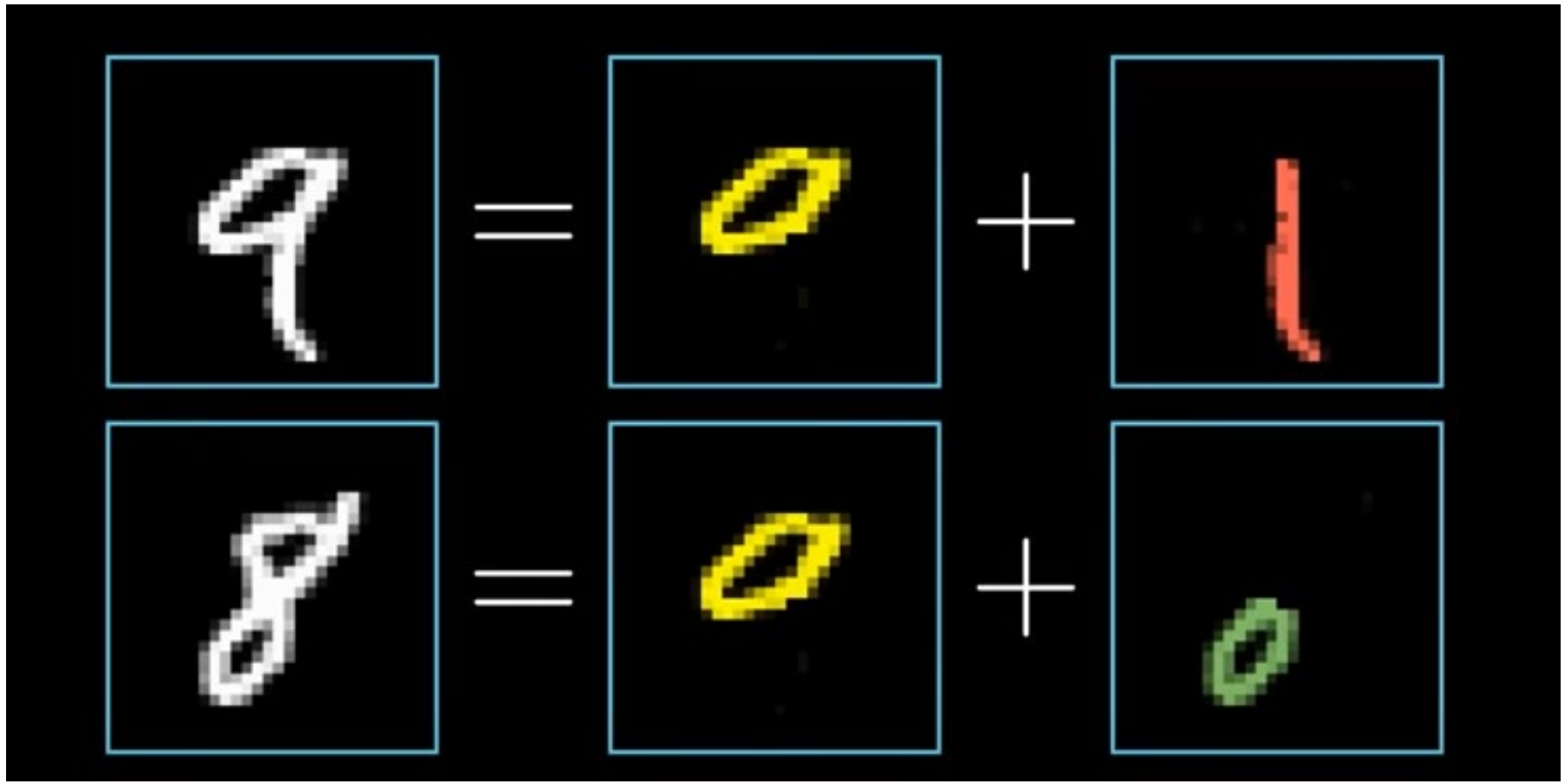
0.81 **Activation**

0.61 **Activation**

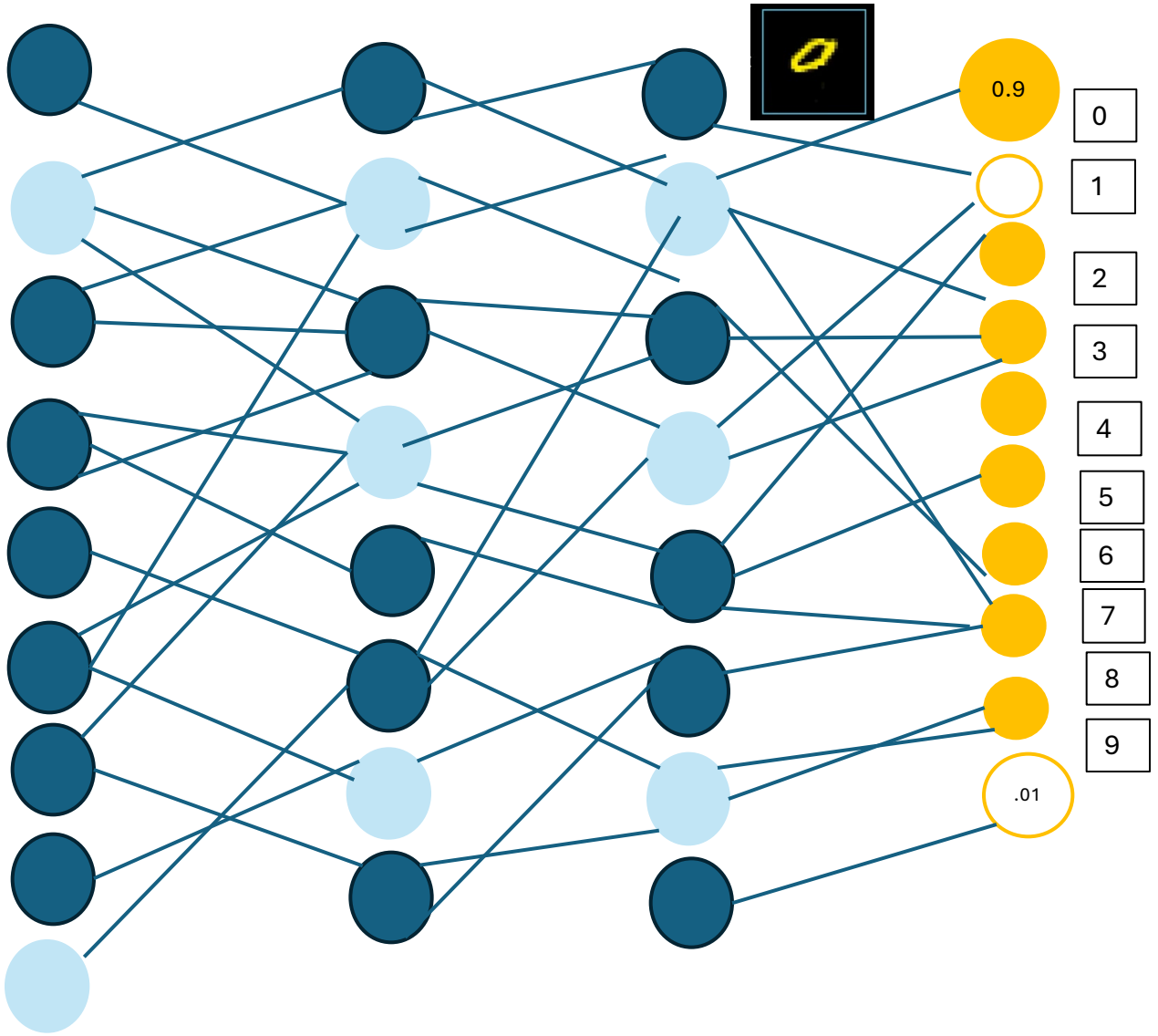
Activation in one layer influence activation in next layer



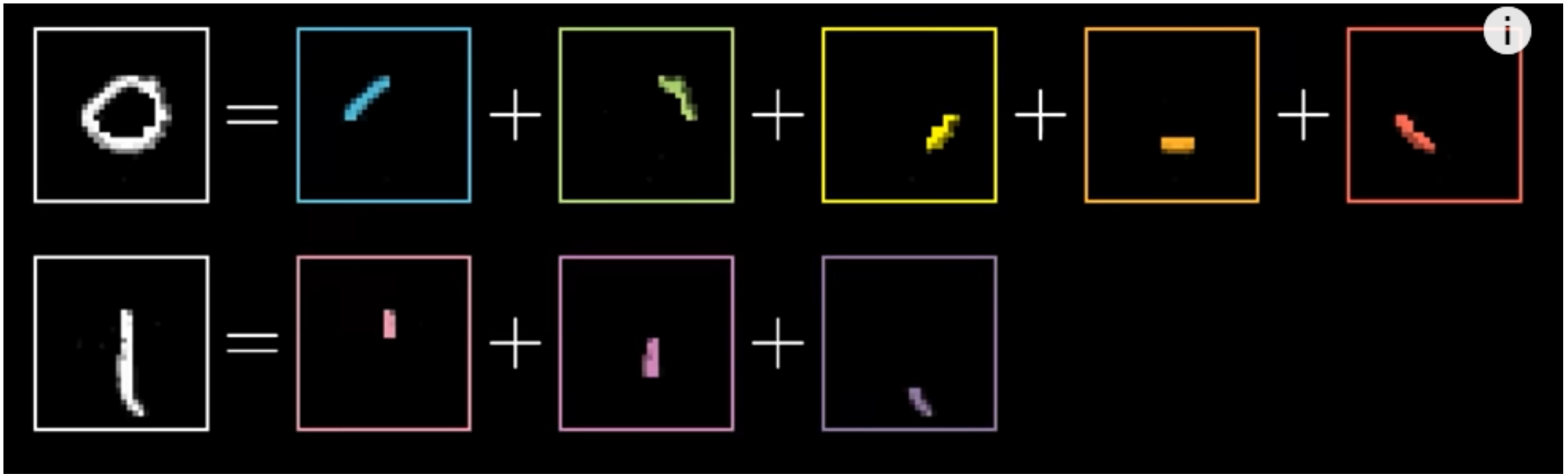
What are the middle layers doing

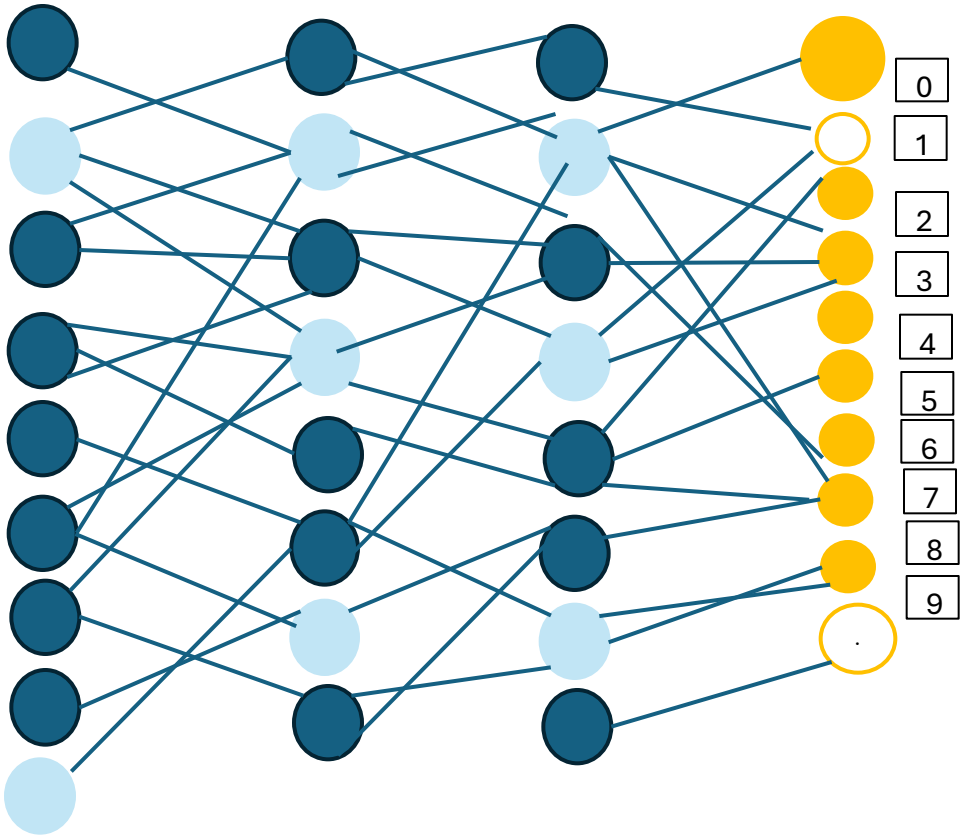


Activation in one layer influence activation in next layer



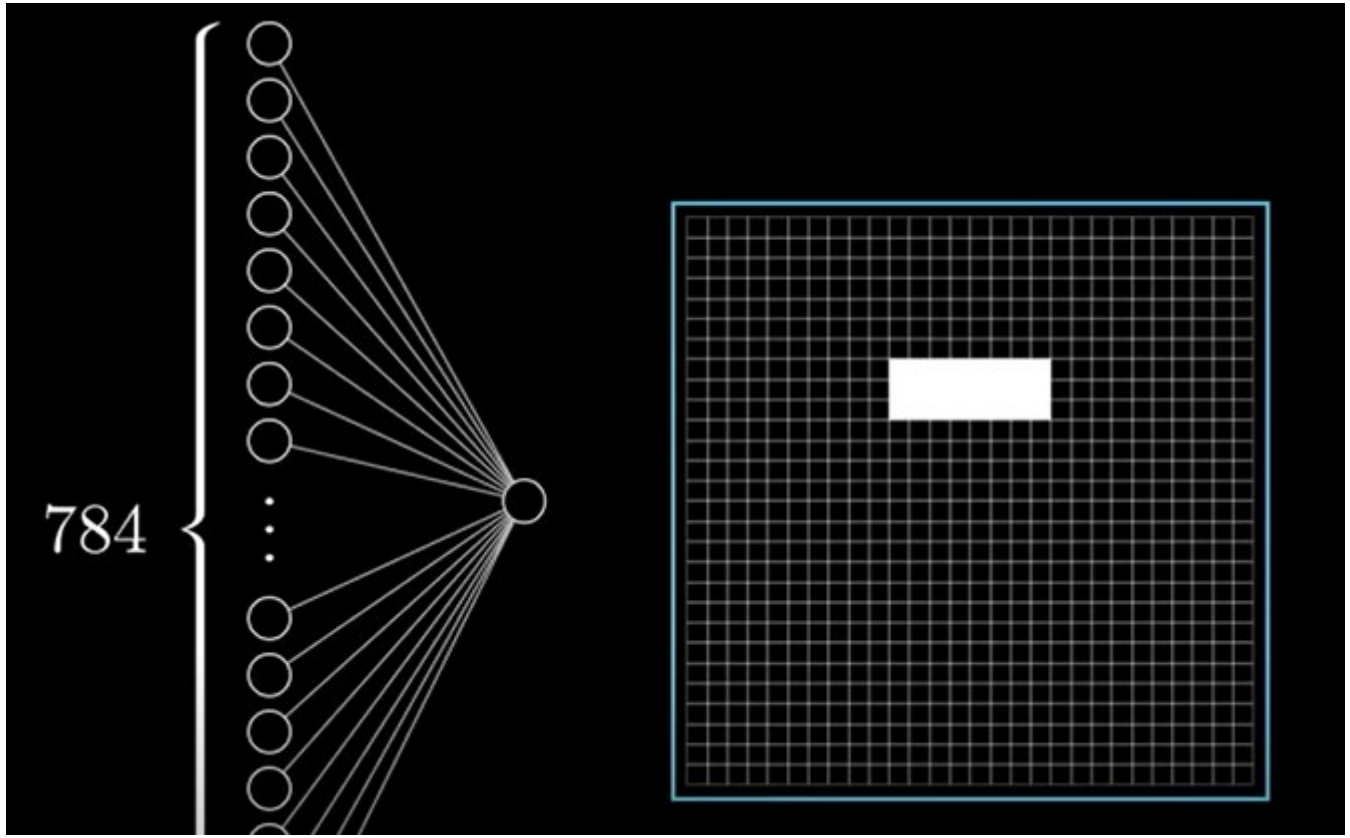
What are the middle layers doing



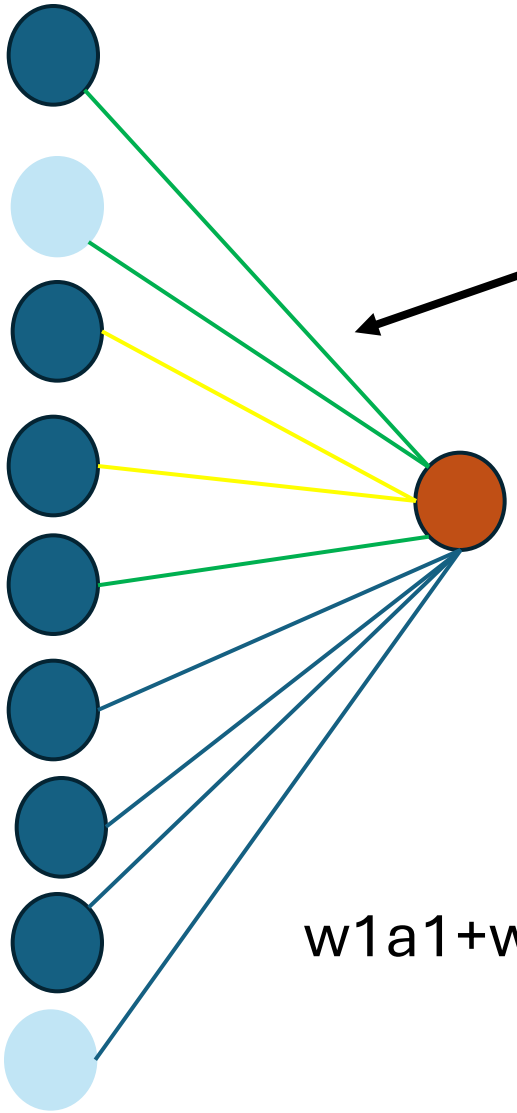


Lights up all the neurons that are recognizing the different regions

Whether a picture has an edge in this region ?
What Parameters should the network have?



Weights



W1=2.09

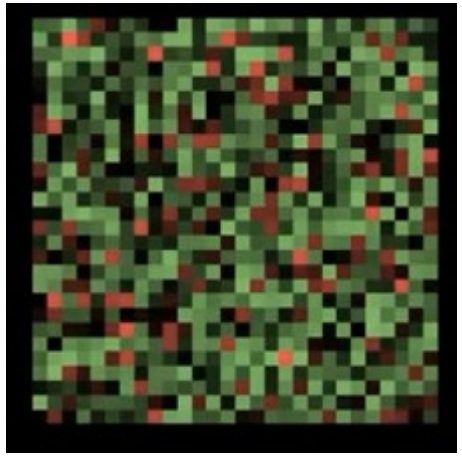
W2=1.1

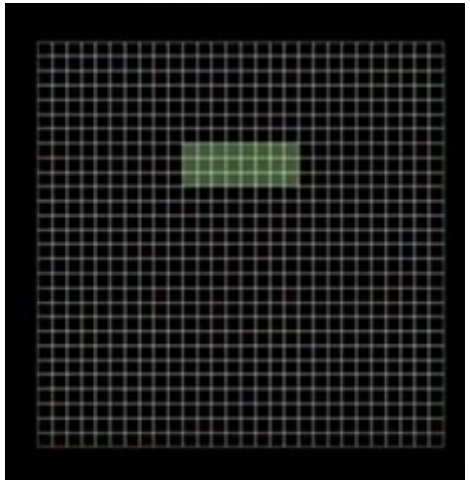
W3=3.09

W4=0.01

W5=1.24

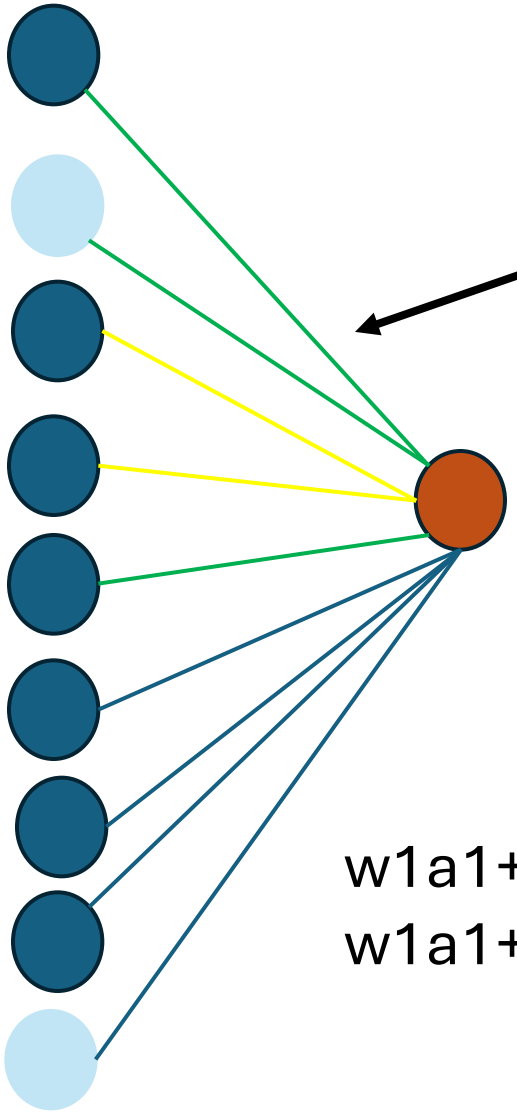
W6=2.09





w1a1+w2a2+.....+wnan

Weights Continued ...



W1=2.09

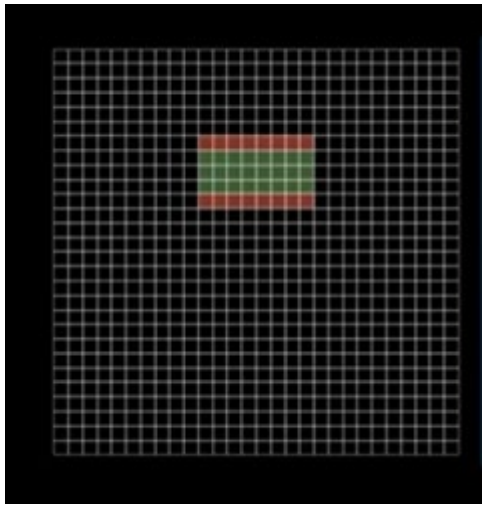
W2=1.1

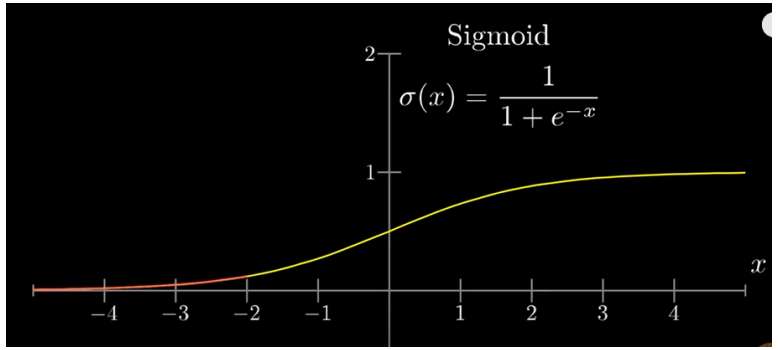
W3=3.09

W4=0.01

W5=1.24

W6=2.09



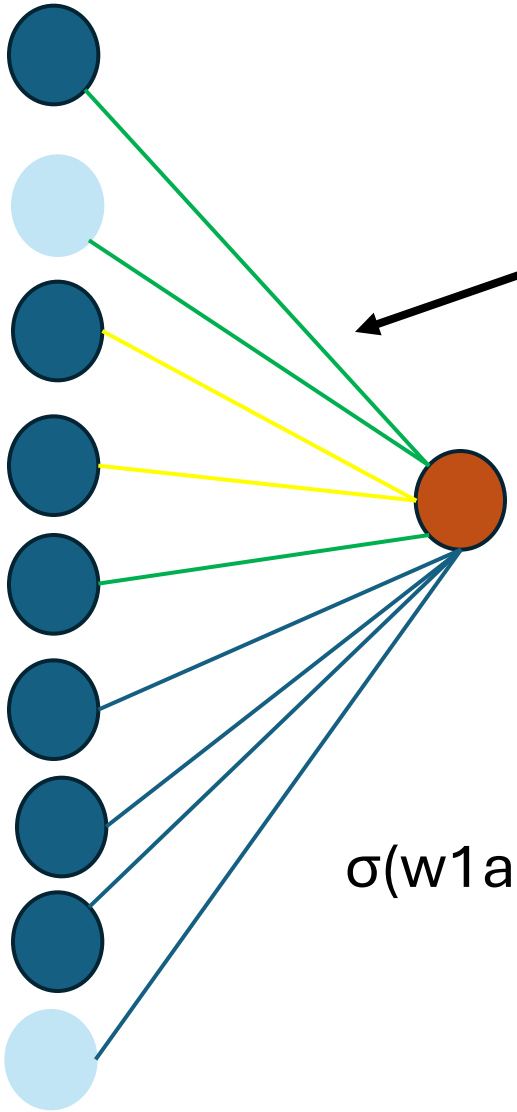


w1a1+w2a2+.....+wnan max for +ve

w1a1+w2a2+.....+wnan min for -ve

(0-1)

Bias



W1=2.09

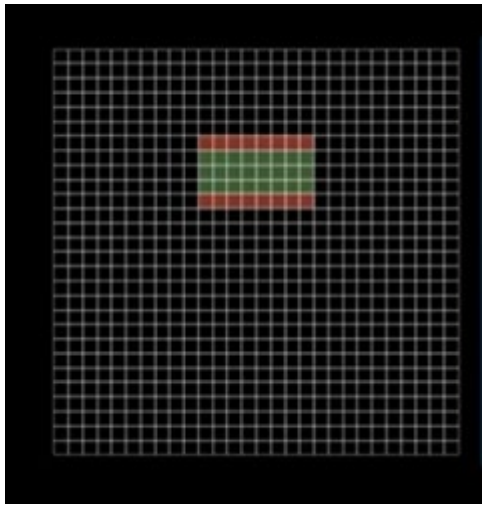
W2=1.1

W3=3.09

W4=0.01

W5=1.24

W6=2.09

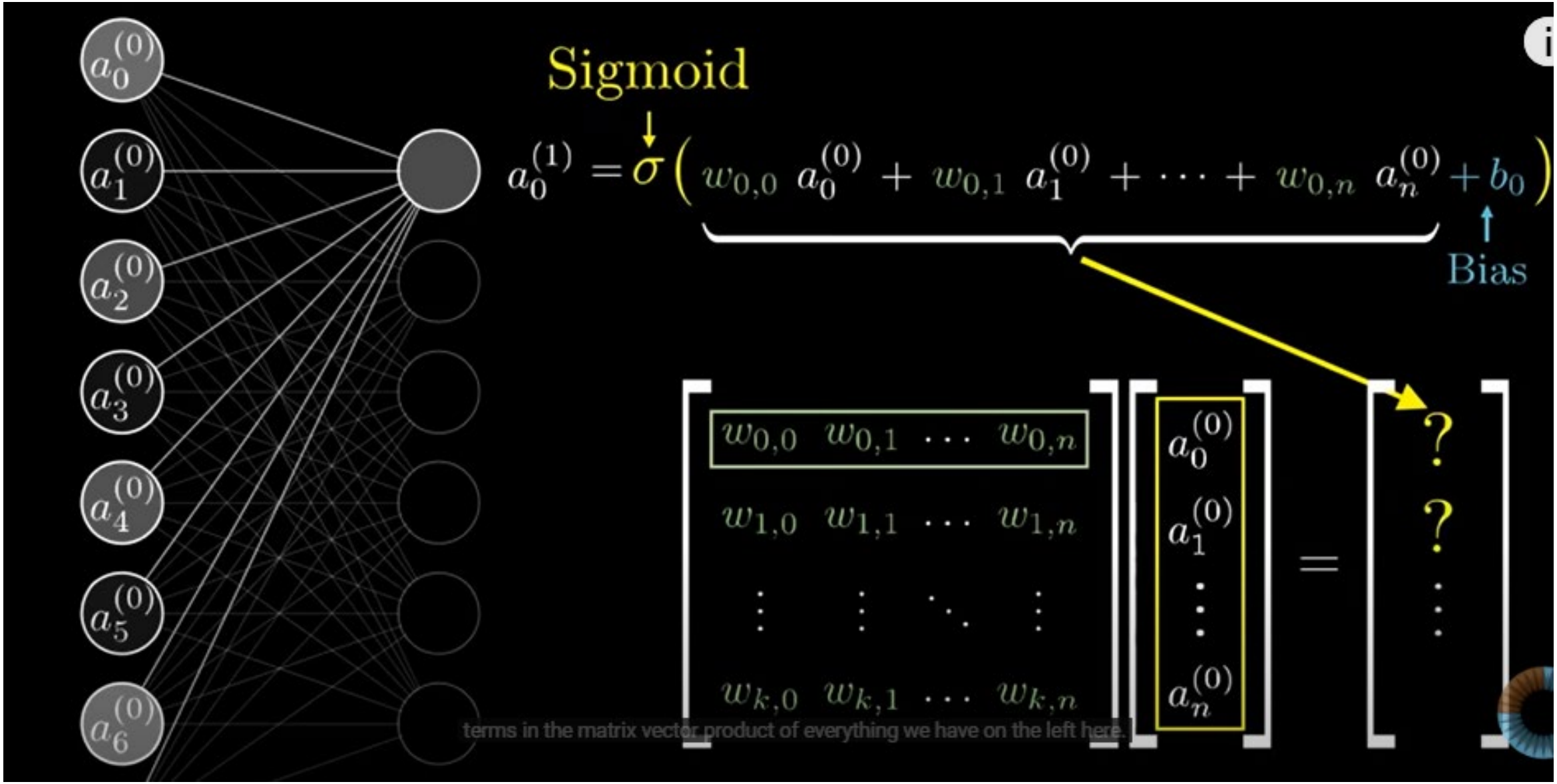


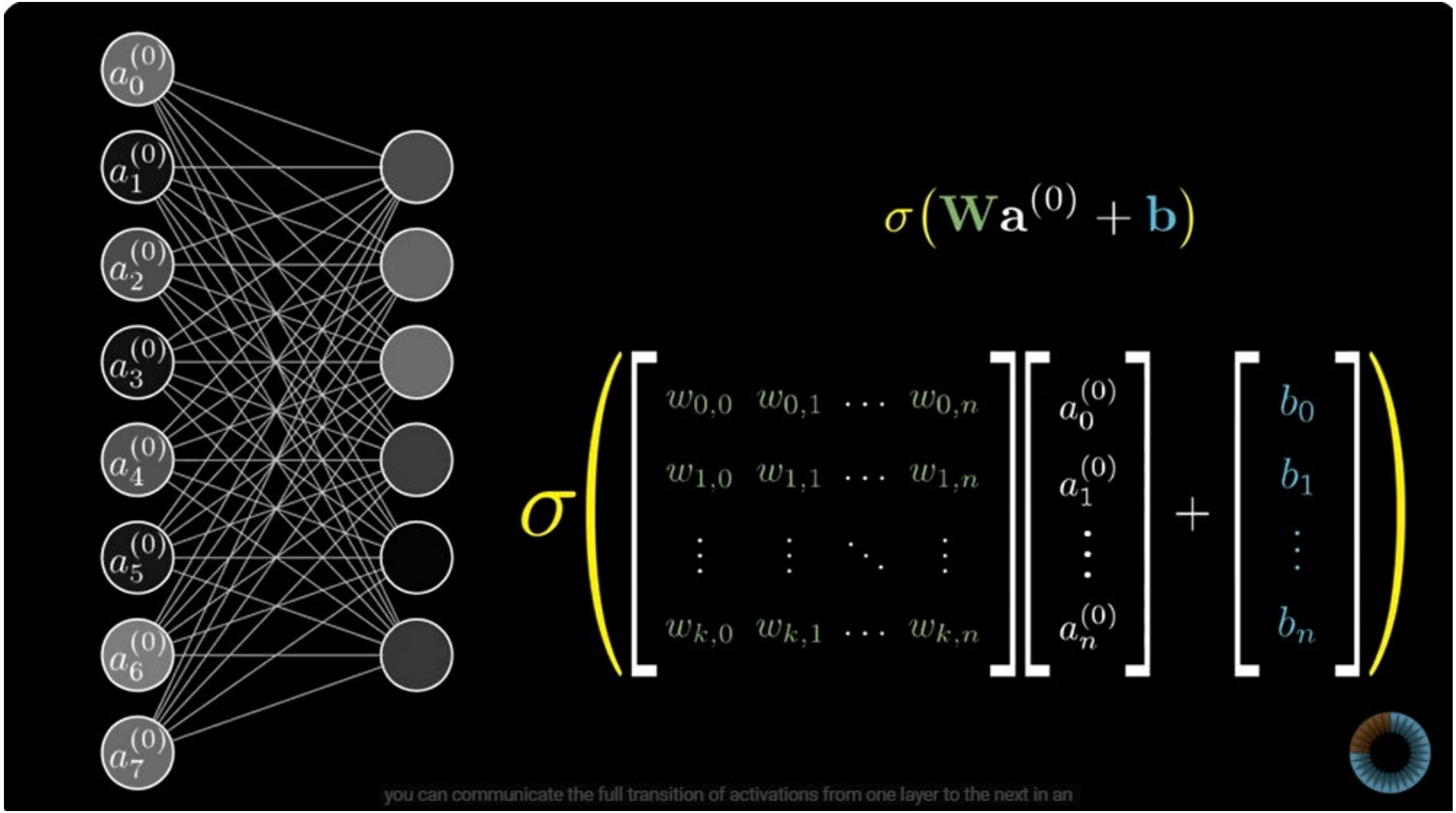
$\sigma(w1a1+w2a2+.....+wnan)$

But you only want to activate meaningfully when the weighted sum >10

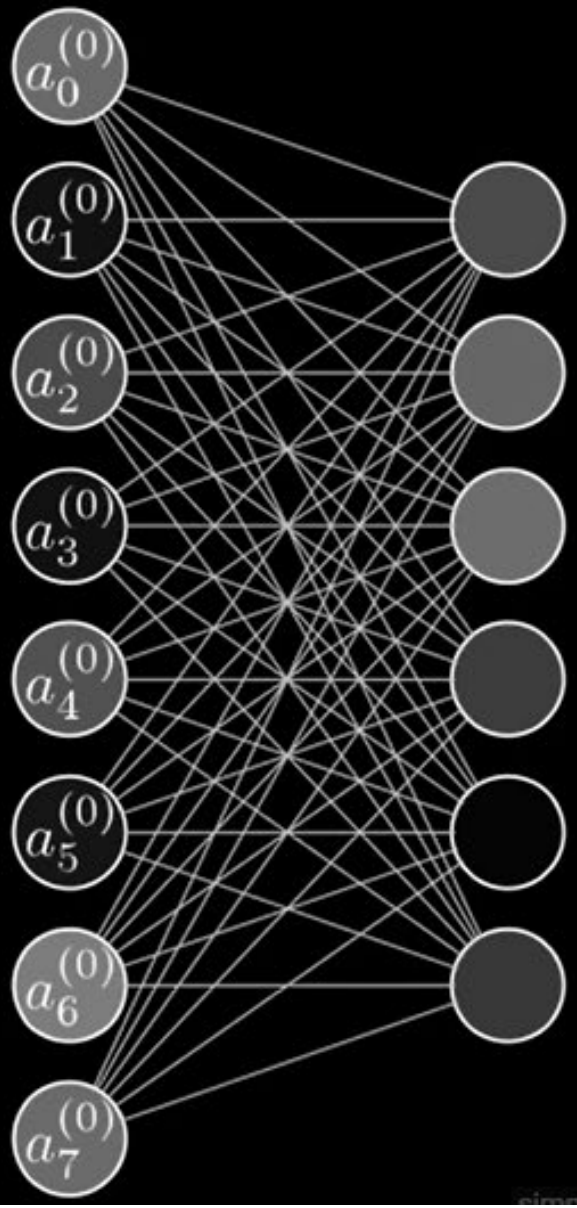
$\sigma(w1a1+w2a2+.....+wnan -10)$

"Bias"


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
$a_0^{(0)}$
$a_1^{(0)}$
$a_2^{(0)}$
$a_3^{(0)}$
$a_4^{(0)}$
$a_5^{(0)}$
$a_6^{(0)}$
$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ \vdots \end{bmatrix}$
terms in the matrix vector product of everything we have on the left here.



you can communicate the full transition of activations from one layer to the next in an

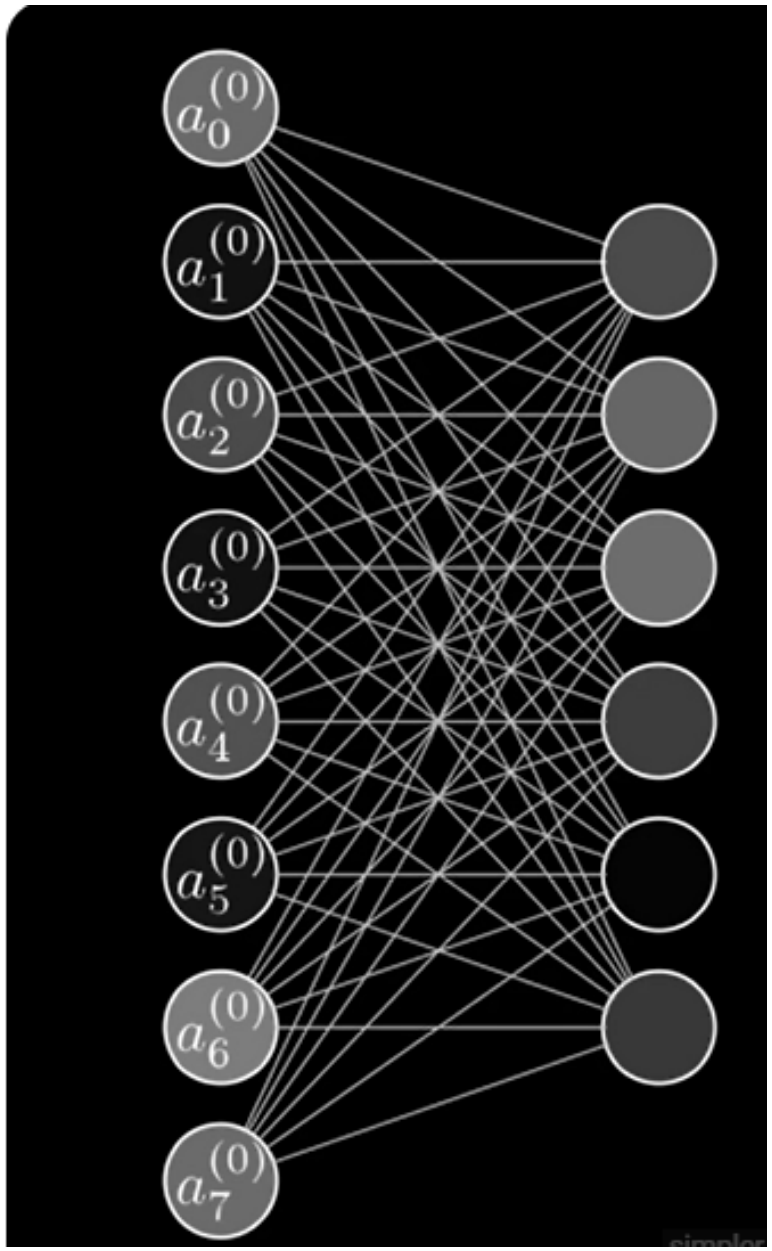


$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$$

```
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)
```

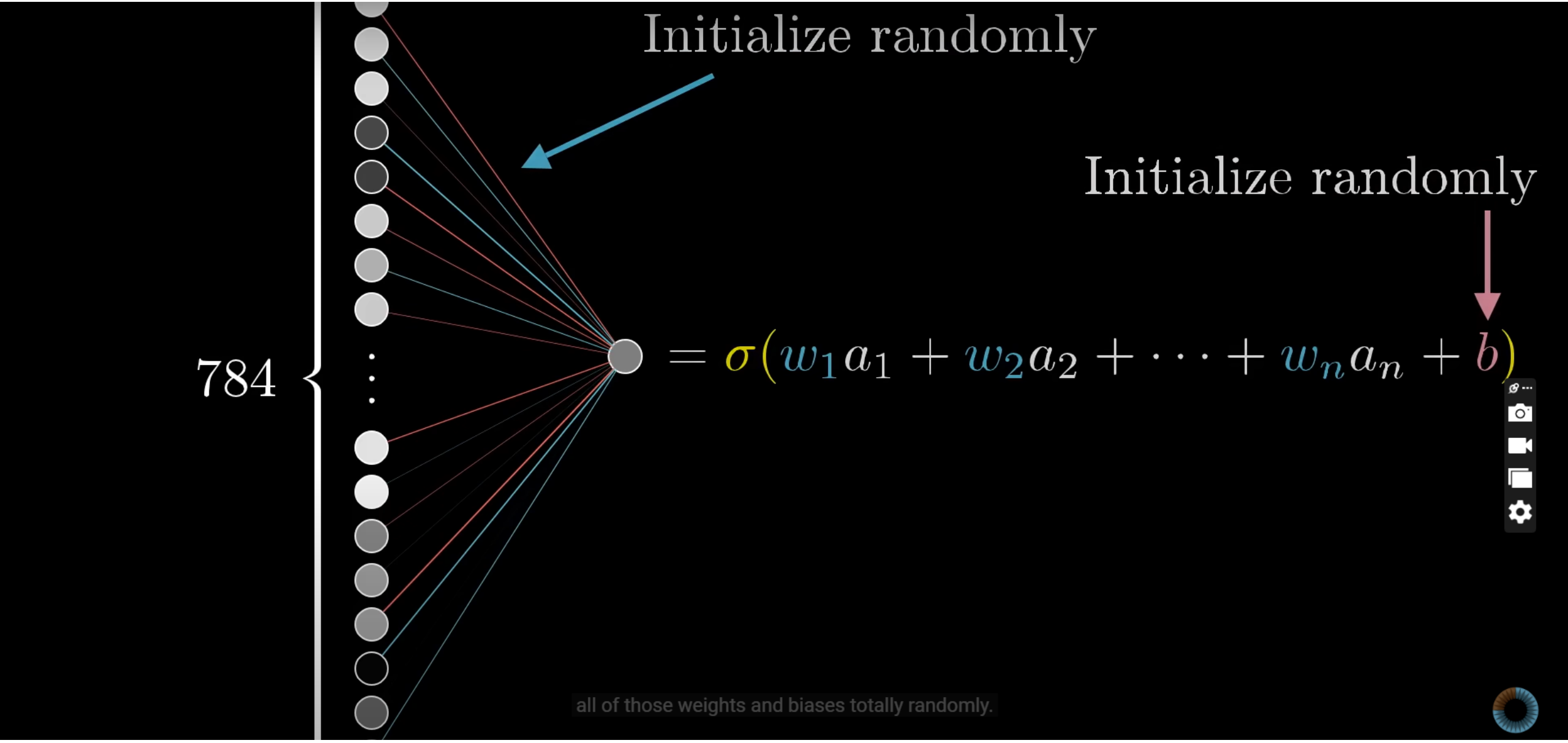
simpler and a lot faster, since many libraries optimize the heck out of matrix



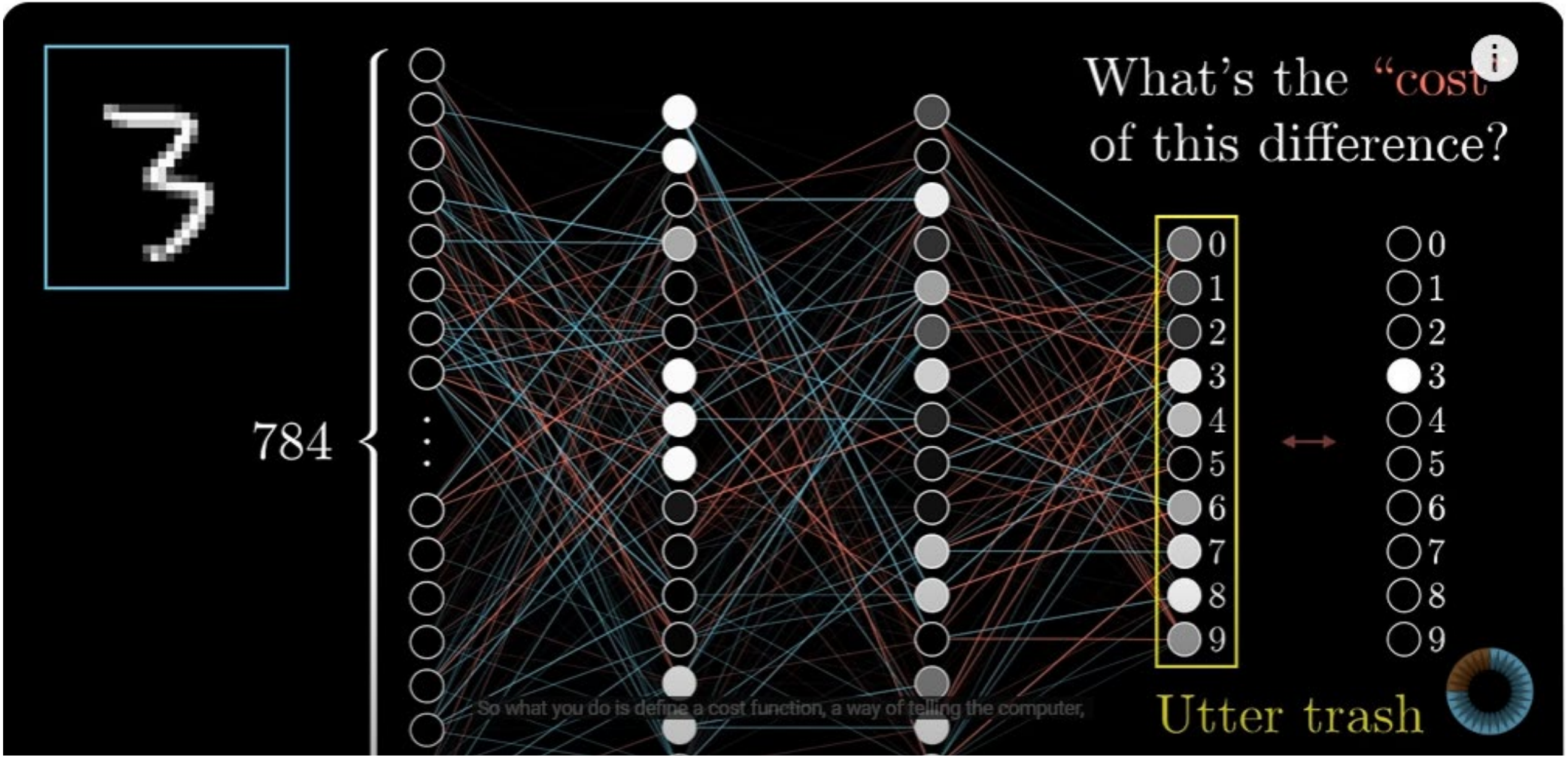
784X16+16x16+16X10 : weights

16+16+10: biases ~ 13000

How u train a NN?



Cost Function

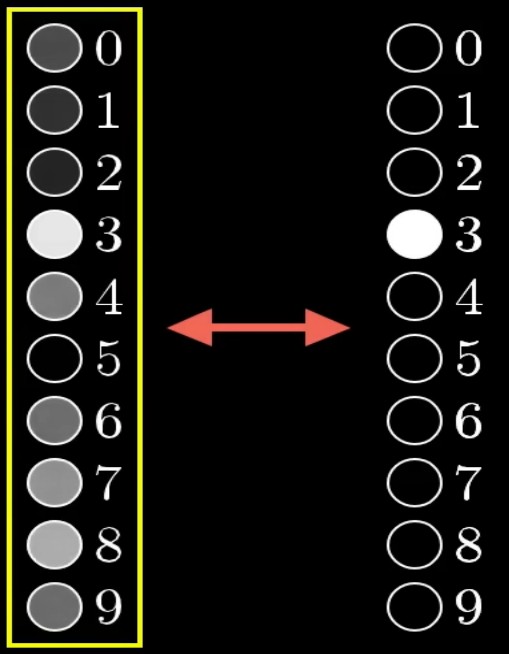


Cost Function is small when the NN classify accurately

Cost of 3

$$
\text{1.61} \left\{ \begin{array}{rcl}
0.0865 & \leftarrow & (0.29 - 0.00)^2 + \\
0.0390 & \leftarrow & (0.20 - 0.00)^2 + \\
0.0214 & \leftarrow & (0.15 - 0.00)^2 + \\
0.0077 & \leftarrow & (0.91 - 1.00)^2 + \\
0.2550 & \leftarrow & (0.51 - 0.00)^2 + \\
0.0002 & \leftarrow & (0.01 - 0.00)^2 + \\
0.1930 & \leftarrow & (0.44 - 0.00)^2 + \\
0.3472 & \leftarrow & (0.59 - 0.00)^2 + \\
0.4698 & \leftarrow & (0.69 - 0.00)^2 + \\
0.1879 & \leftarrow & (0.43 - 0.00)^2
\end{array} \right.
$$

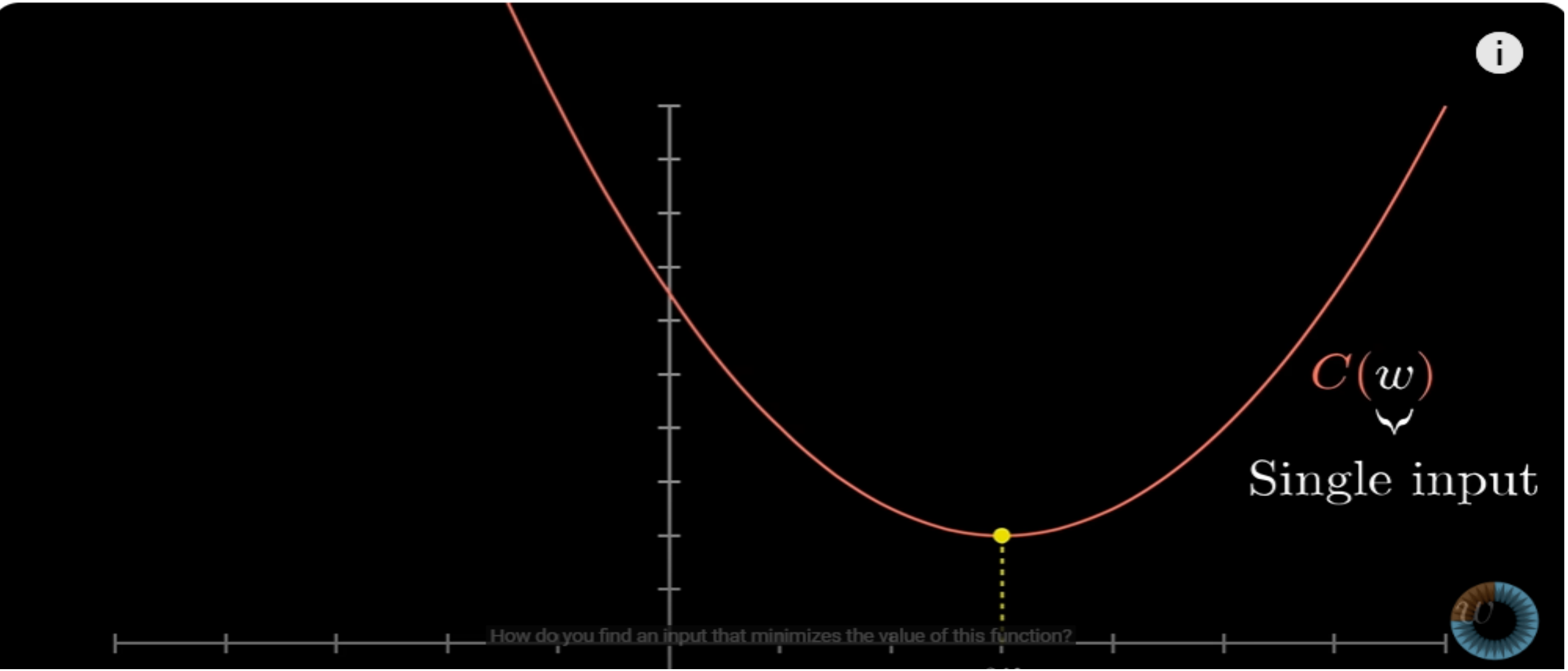
What's the "cost" of this difference?



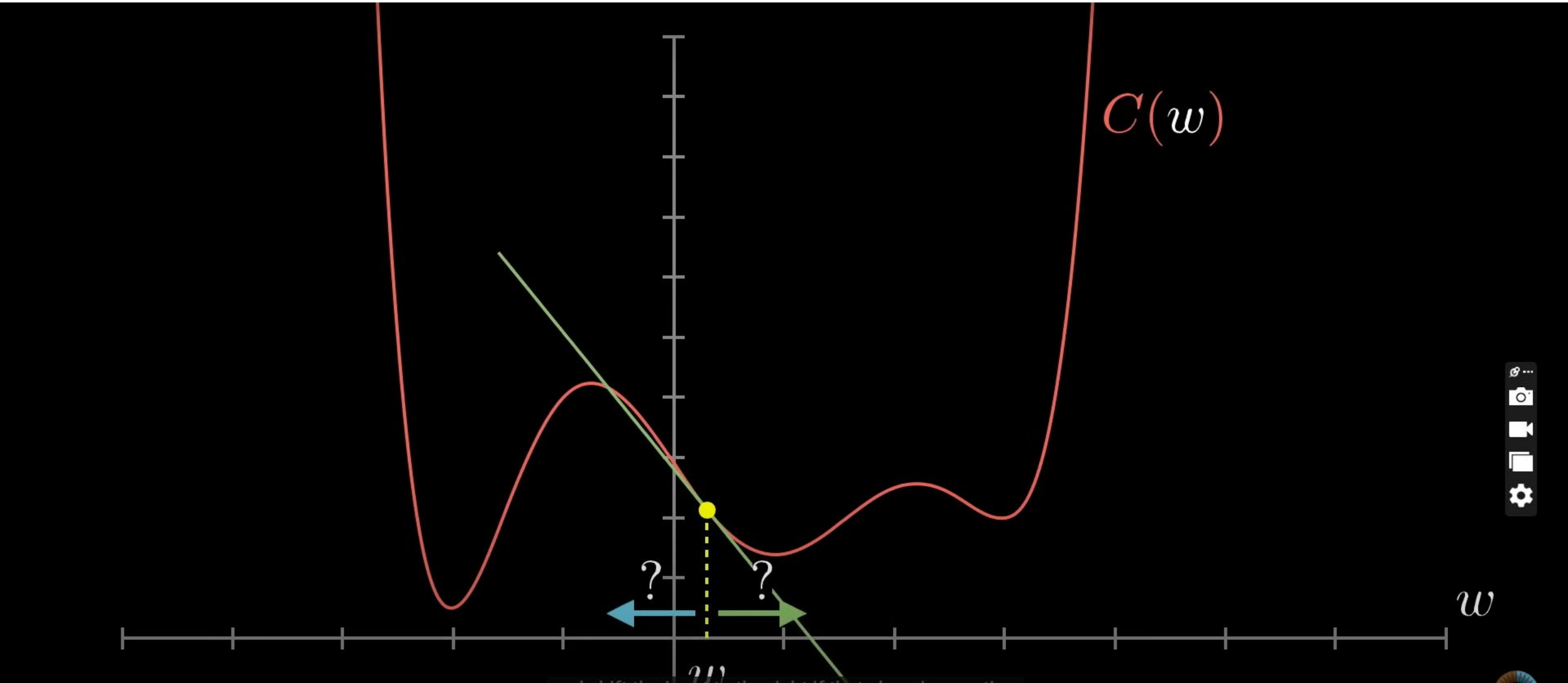
Utter trash

Notice this sum is small when the network confidently classifies the image correctly,

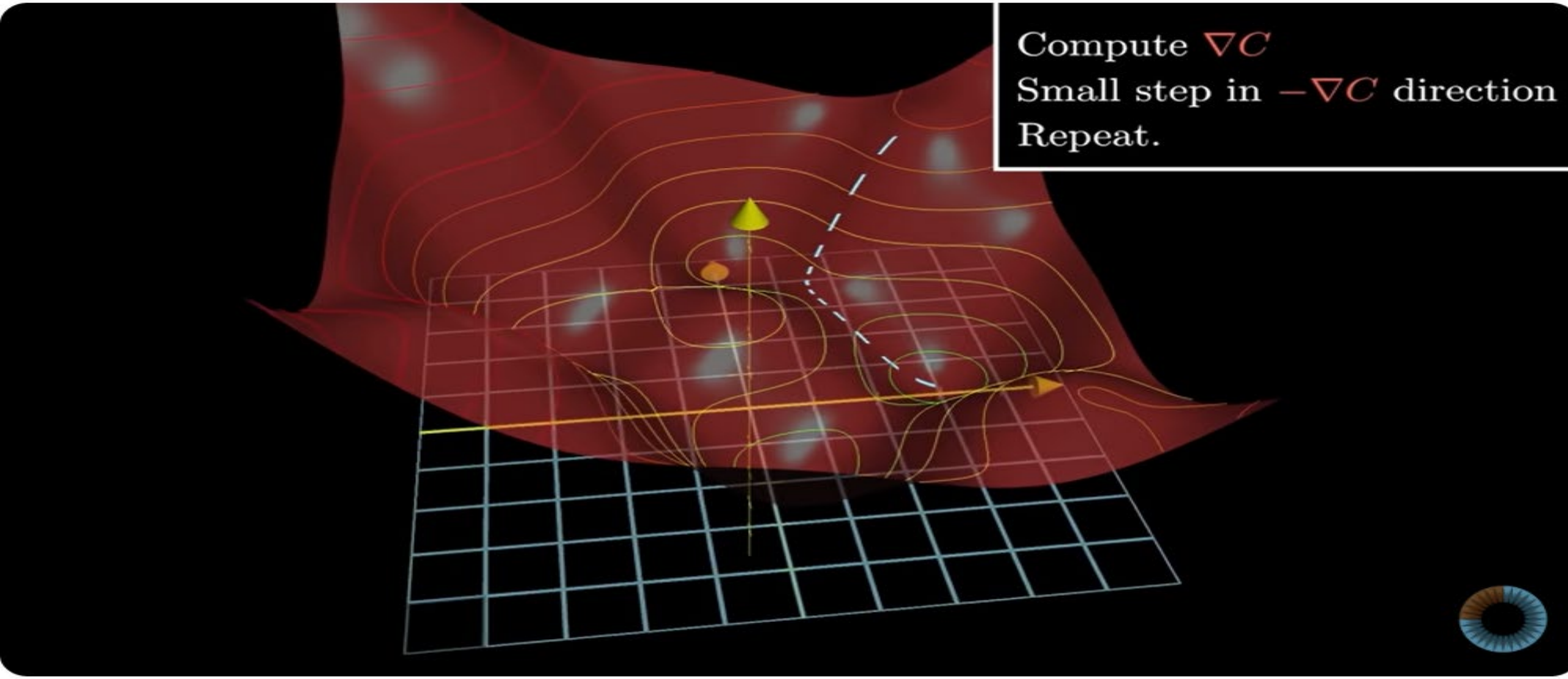
Let the cost function for all 13000 parameters be defined as C(w). We want to minimize this fx.



Find the slope of the C(w) → shift right (-ve slope) → shift left (+ve slope)



Find the slope of the C(w) → shift right (-ve slope) → shift left (+ve slope)



Which nudges will rapidly decrease the cost function

13,002 weights and biases

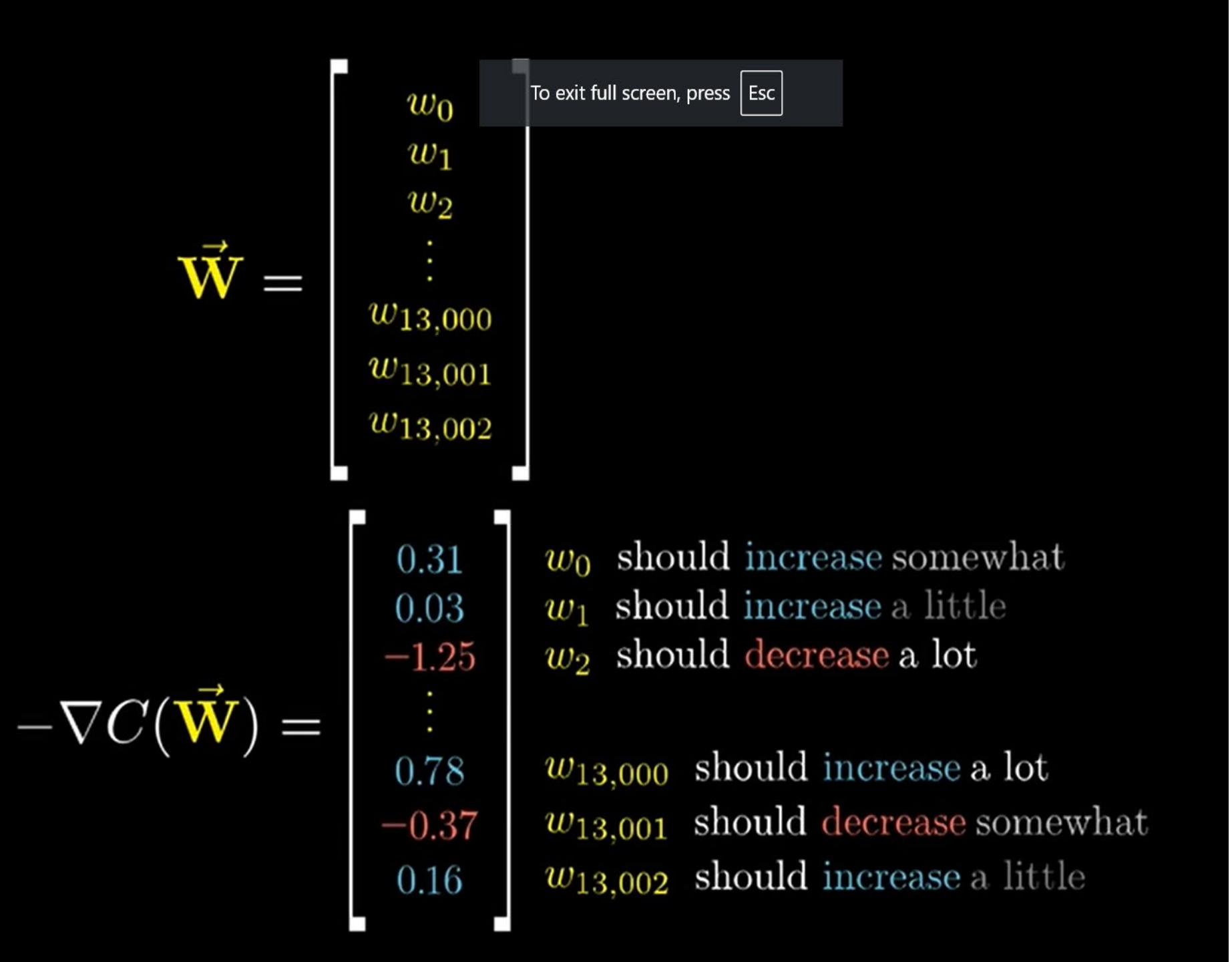
$$\vec{\mathbf{W}} = \begin{bmatrix} 2.43 \\ -1.12 \\ 1.47 \\ \vdots \\ -0.76 \\ 3.50 \\ 2.03 \end{bmatrix}$$

How to nudge all weights and biases

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

Remember !!

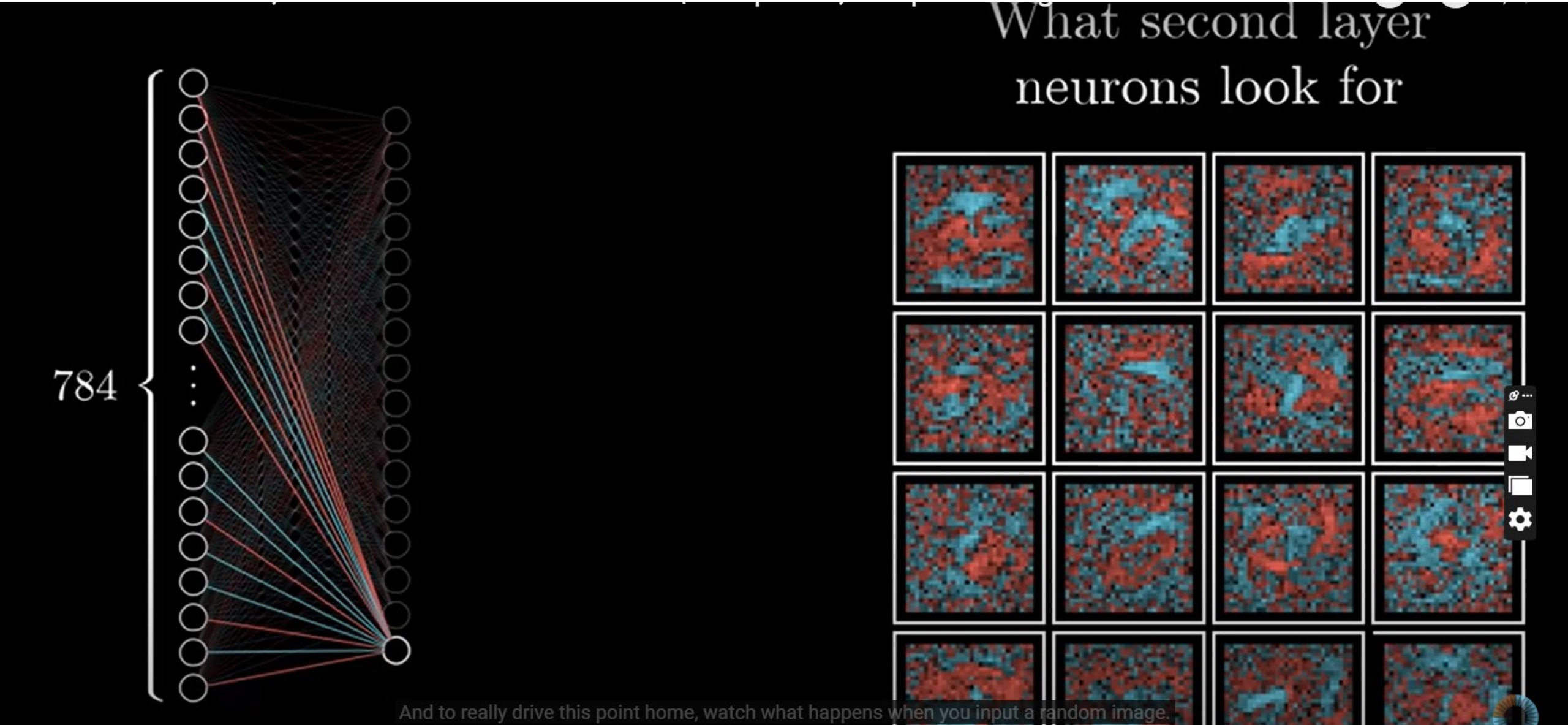
1. Cost function is average on all the training samples
2. When network is learning it's minimizing a cost function.
3. The process of nudging an input with +ve gradient of cost function is *Gradient Descent. It's a way to converge to local minima of cost function*.



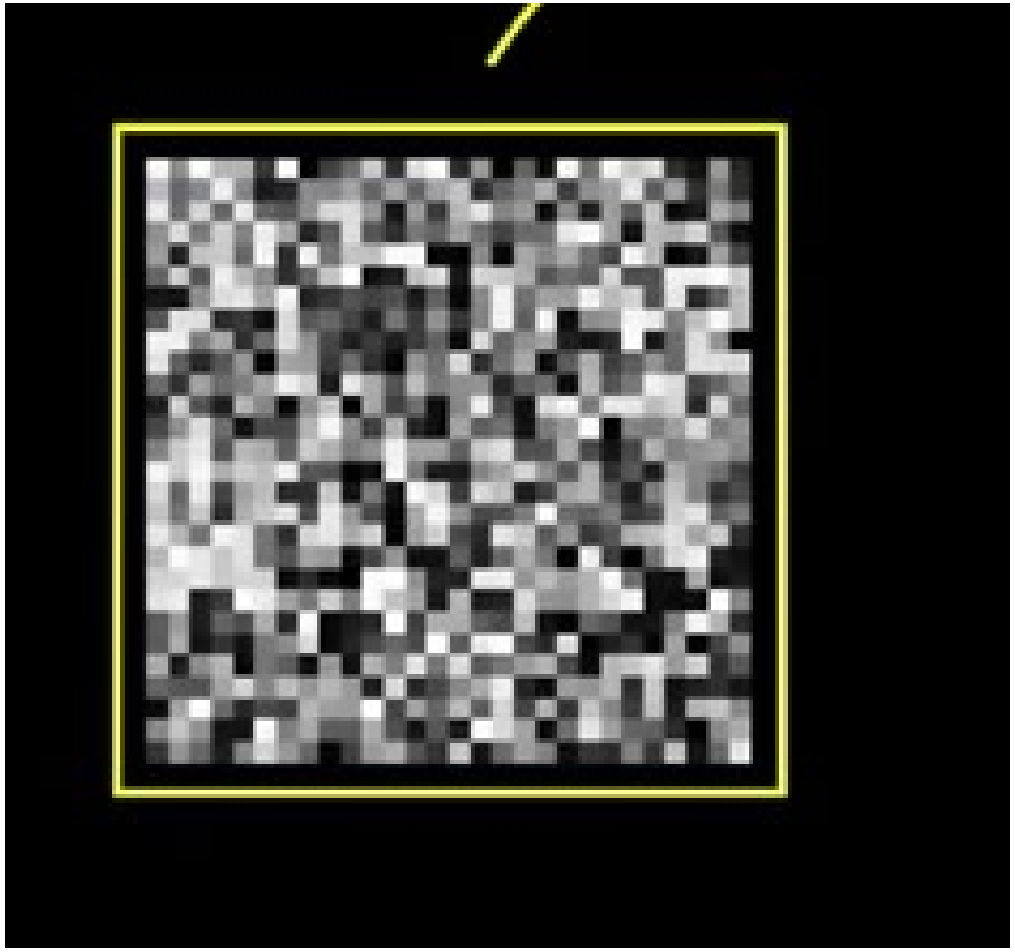
- Signs mentioned if the input vector should be nudged up (+ve) or down (-ve).
- Relative magnitude mentions which changes matter more.

In short, which changes to which weights matters the most

So, is my Multilayer Perceptron learning edges and patters? No, its learning some random patters based on finding the local minima.



What will happen if you input a random image ?



Confidently gives you random answer because the network can only differentiate it has no idea how to draw a digit?