Zurich University
of Applied Sciences

Information Engineering 2

Machine Learning Q&A

Prof. Dr. Kurt Stockinger

Zurich University
of Applied Sciences

Semesterplan

SW	Datum	Vorlesungsthema	Praktikum
1	23.02.2022	Data Warehousing Einführung	Praktikum 1: KNIME Tutorial
2	02.03.2022	Dimensionale Datenmodellierung 1	Praktikum 1: KNIME Tutorial (Vertiefung)
3	09.03.2022	Dimensionale Datenmodellierung 2	Praktikum 2: Datenmodellierung
4	16.03.2022	Datenqualität und Data Matching	Praktikum 3: Star-Schema, Bonus: Praktikum 4: Slowly Changing Dimensions
5	23.03.2022	Big Data Einführung	DWH Projekt - Teil 1
6	30.03.2022	Spark - Data Frames	DWH Projekt - Teil 2 (Abgabe: 4.4.2022 23:59:59)
7	06.04.2022	Data Storage: Hadoop Distributed File System & Parquet	Praktikum 1: Data Frames
8	13.04.2022	Query Optimization	Praktikum 2: Data Storage
9	20.04.2022	Spark Best Practices & Applications	Praktikum 3: Query Optimization & Performance Analysis
10	27.04.2022	Machine Learning mit Spark 1	Praktikum 3: Query Optimization & Performance Analysis (Vertiefung)
11	04.05.2022	Machine Learning mit Spark 2 + Q&A	Praktikum 4: Machine Learning (Regression)
12	11.05.2022	NoSQL Systems	Big Data Projekt - Teil 1
13	18.05.2022	Keine Vorlesung (Arbeit am Projekt)	Big Data Projekt - Teil 2
14	25.05.2022	Keine Vorlesung (Arbeit am Projekt)	Big Data Projekt - Teil 3 (Abgabe: 30.5.2022 23:59:59)

Online Courses on Introduction to Neural Networks

- Basics of neural networks (20 min):
  https://www.youtube.com/watch?v=aircAruvnKk

- How neural networks learn (21 min):
  https://www.youtube.com/watch?v=IHZwWFHWa-w

  Transformers (15 min):
  https://youtu.be/4Bdc55j80l8

Overview

- Simple introduction into perceptrons and neural networks

- Highlights of the two videos:
  - Basics of neural networks
  - How neural networks learn

Input based on Yale Lecture

Andrej Karpathy's Hacker's guide to Neural Networks:
http://karpathy.github.io/neuralnets/

Andrej Karpathy's lecture notes:
http://cs231n.github.io/

Geoffrey E. Hinton, Yann LeCun, and Yoshua Bengio (video; NIPS 2015):
http://research.microsoft.com/apps/video/default.aspx?id=259574

Michael Nielsen's Neural Networks and Deep Learning:
http://neuralnetworksanddeeplearning.com/

Simple Decision

- Assume we want to go to a chocolate festival.
- Three variables go into our decision:
  - Is the weather good?
  - Does your friend want to go with you?
  - Is it near public transportation?

Simple Decision #2

- Assume that answers to these questions are binary variables x_i
  - 0: no
  - 1: yes

- Questions and answers:
  - Is the weather good? x_1
  - Does your friend want to go with you? x_2
  - Is it near public transportation? X_3

- How do we describe a decision statement?

Simple Decision #3

- Use weights w_i for importance of features:

$$x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \geq \text{threshold}$$

- Assumption:
  - If statement is true, we attend festival (otherwise not)

- Example:
  - If we hated weather and were more neural about friends and public transport, we would pick the weights 6, 2, 2

  - Assume threshold 5: we would only go, if the weather was great

  - Threshold 3?

Simple Decision #4

- We can define binary variable y
- Defines if we go to festival or not

$$y = \begin{cases} 0, & x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 < \text{threshold} \\ 1, & x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \geq \text{threshold} \end{cases}$$

Simple Decision #5

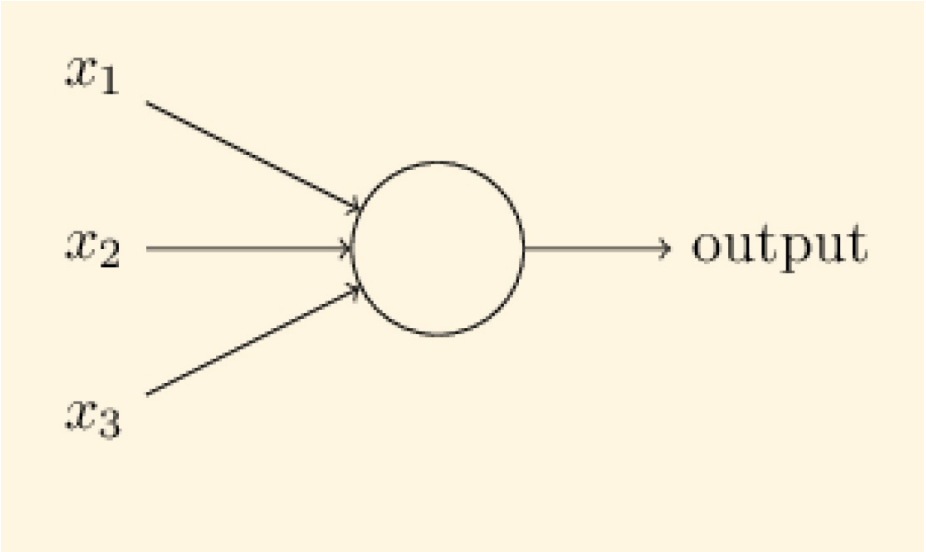
- We can rewrite this as dot product of vector of binary inputs x and vector of weights w
- Reformulate threshold as negative bias b

$$y = \begin{cases} 0, & x \cdot w + b < 0 \\ 1, & x \cdot w + b \geq 0 \end{cases}$$

- Goal: Find separate hyperplane (like with logistic regression)

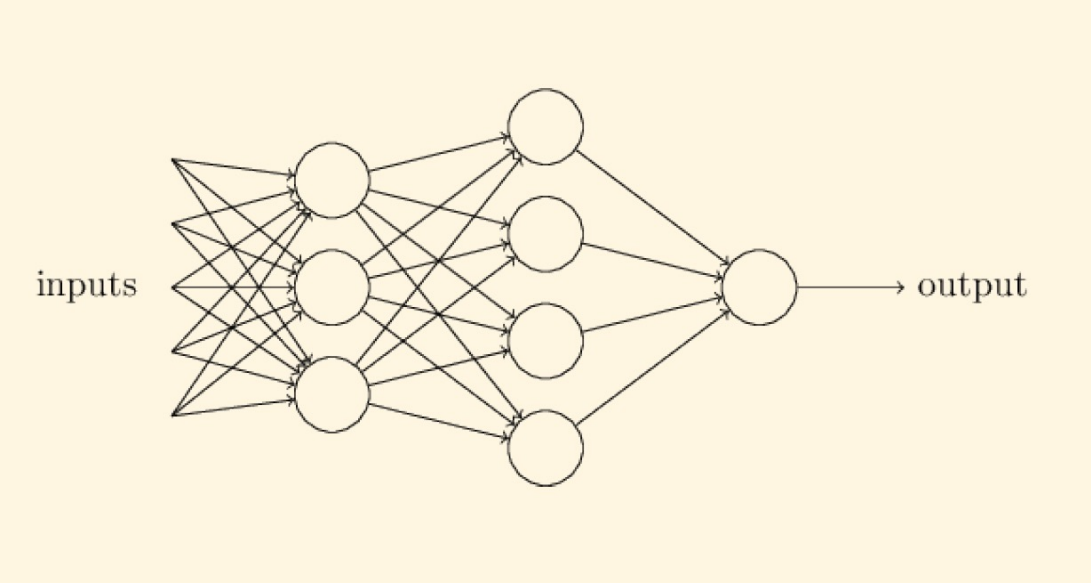
A Perceptron

- Graphical representation of algorithm as a perceptron
  - 3 binary inputs
  - 1 binary output
  - Use weighting scheme described previously

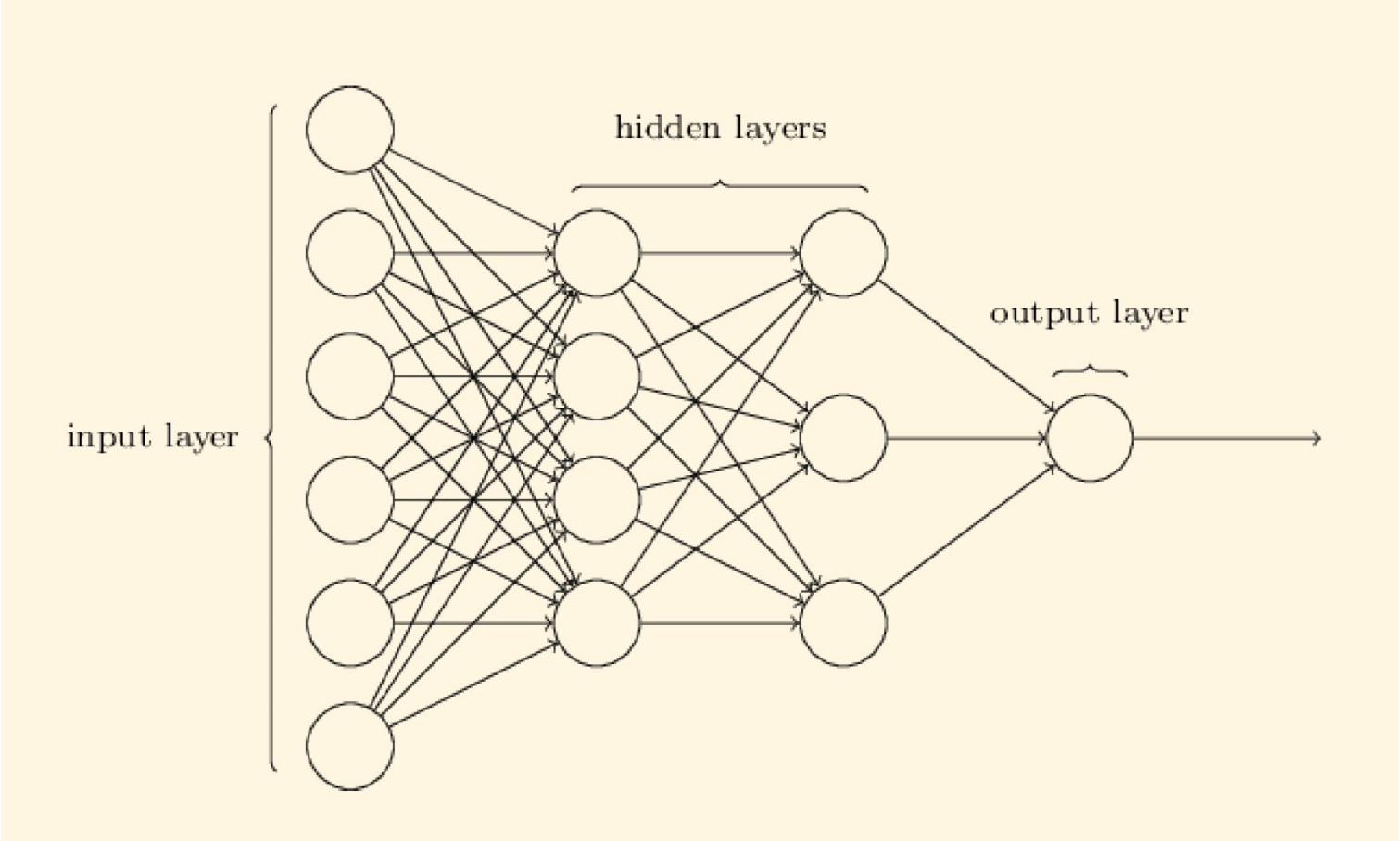


Neural Network = Network of Perceptrons

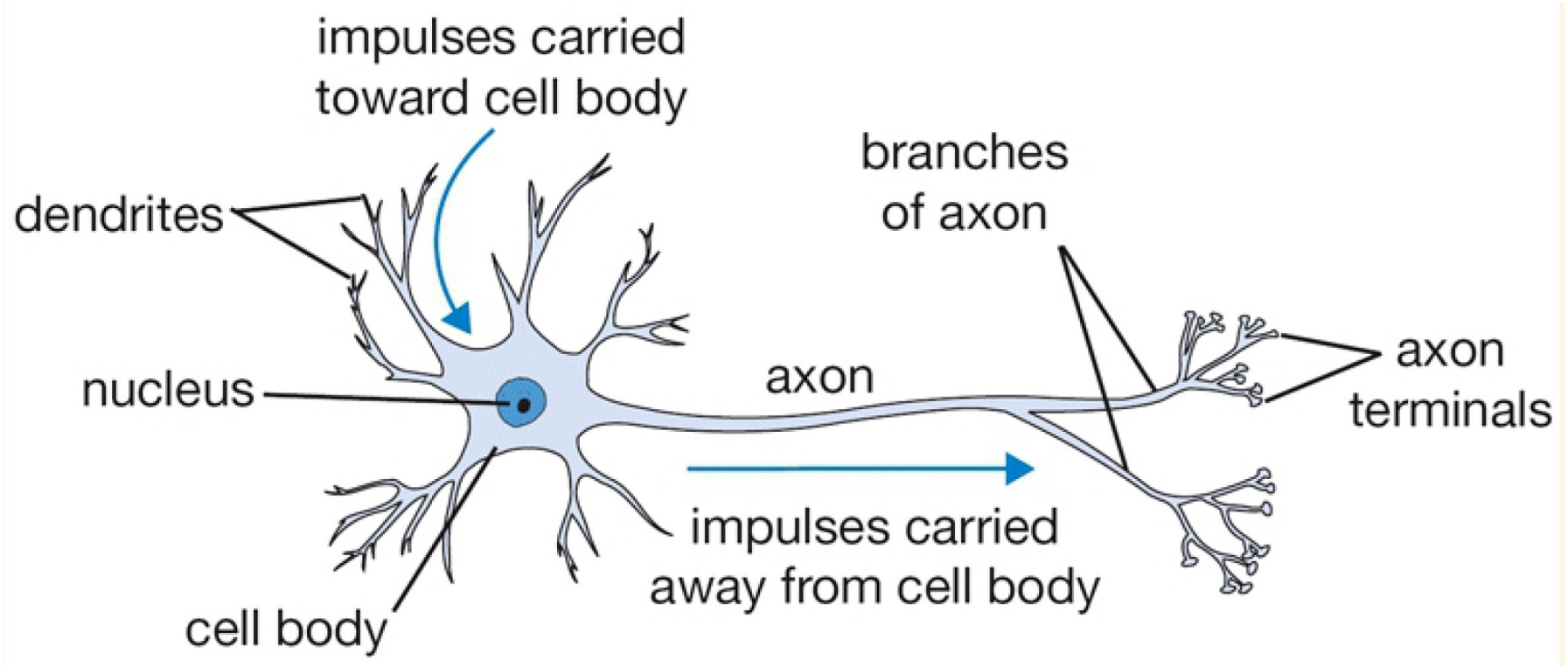
- Output of one perceptron is used as input for other perceptrons



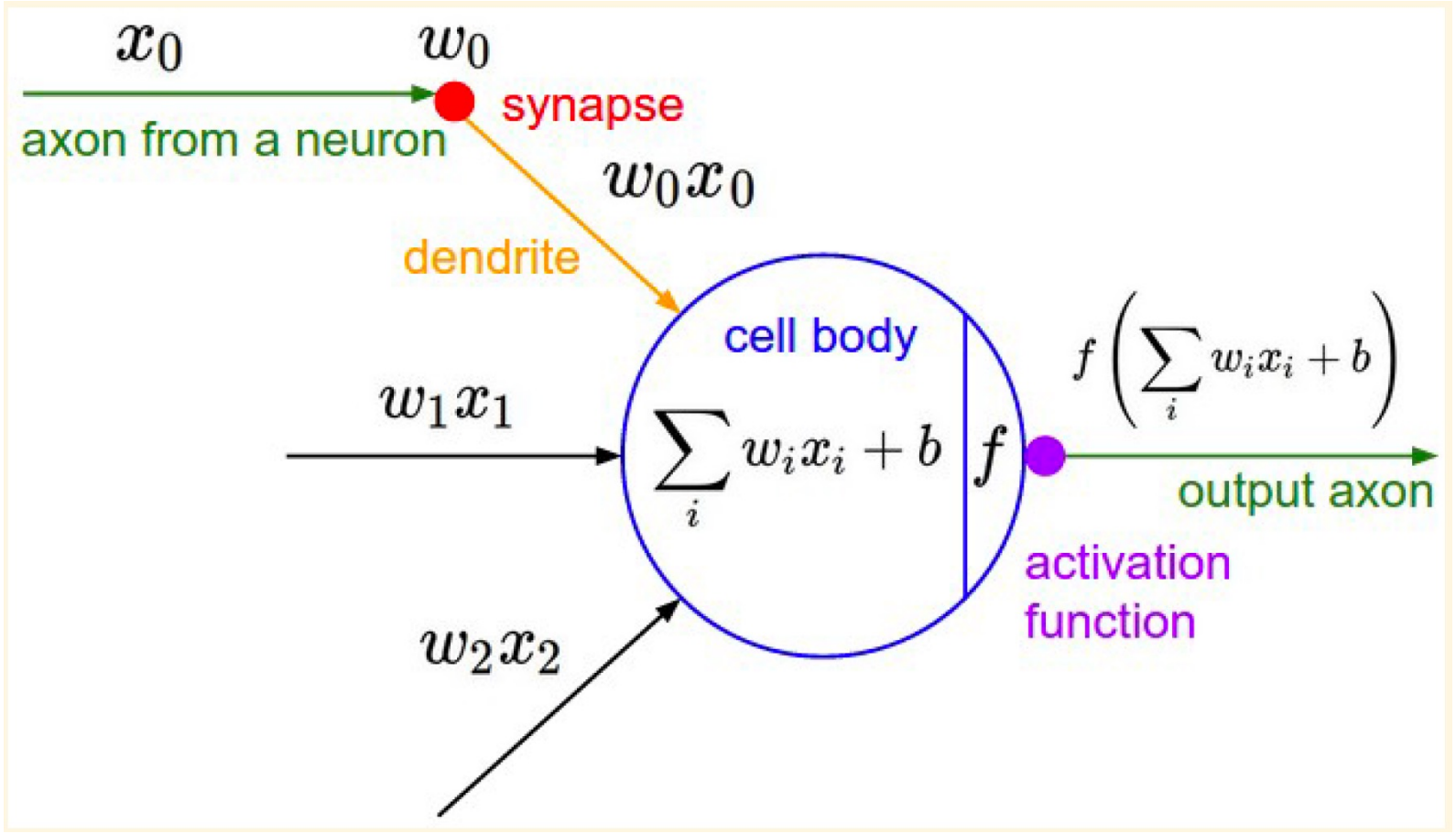
Neural Network with Hidden Layers



Neural Network in Nature



Activation Function for Neurons to Fire



Disadvantage of Simple Approach

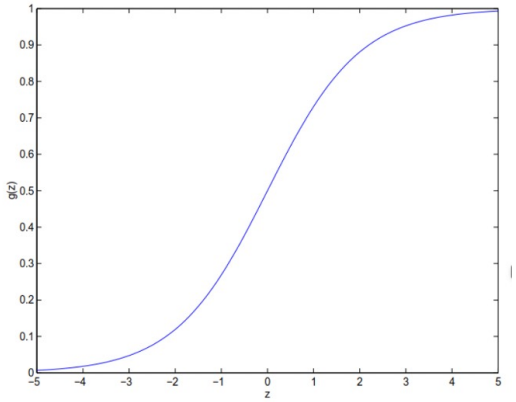
- Each node (or neuron) has only two possible states: 0 or 1
  → Small change in the input values can cause a large change of the output

- Better solution:
  - Output states should be continuous values, e.g. between 0 and 1

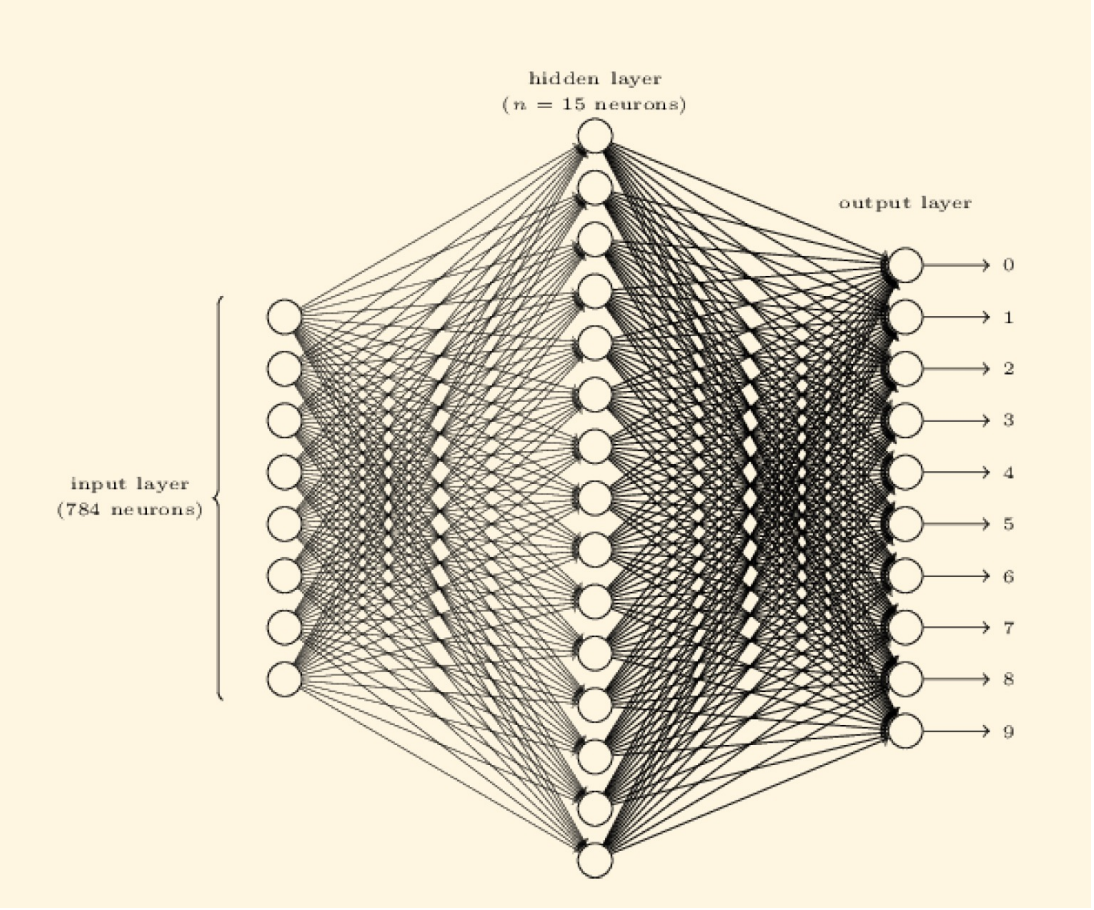
- How we do get such an output?

Use Sigmoid Function as Activation Function

$$\sigma(x \cdot w + b) = \frac{1}{1 + e^{-(x \cdot w + b)}}$$



More Complex Neural Network for Detecting Numbers



Classification Problem

- Goal:
  - Given an input (number), in which class does it fall?
    - E.g. numbers between 0 and 9

- Approach:
  - Find output neurons with largest output
  - We need to learn weights and bias terms

- Note:
  - If we only have a network with one node, we can directly use logistic regression

Learning Approach

- We need a cost function (loss function) that measures how well the network predicts outputs on the test set
- Goal:
  - Find a set of weights and biases that minimizes the cost

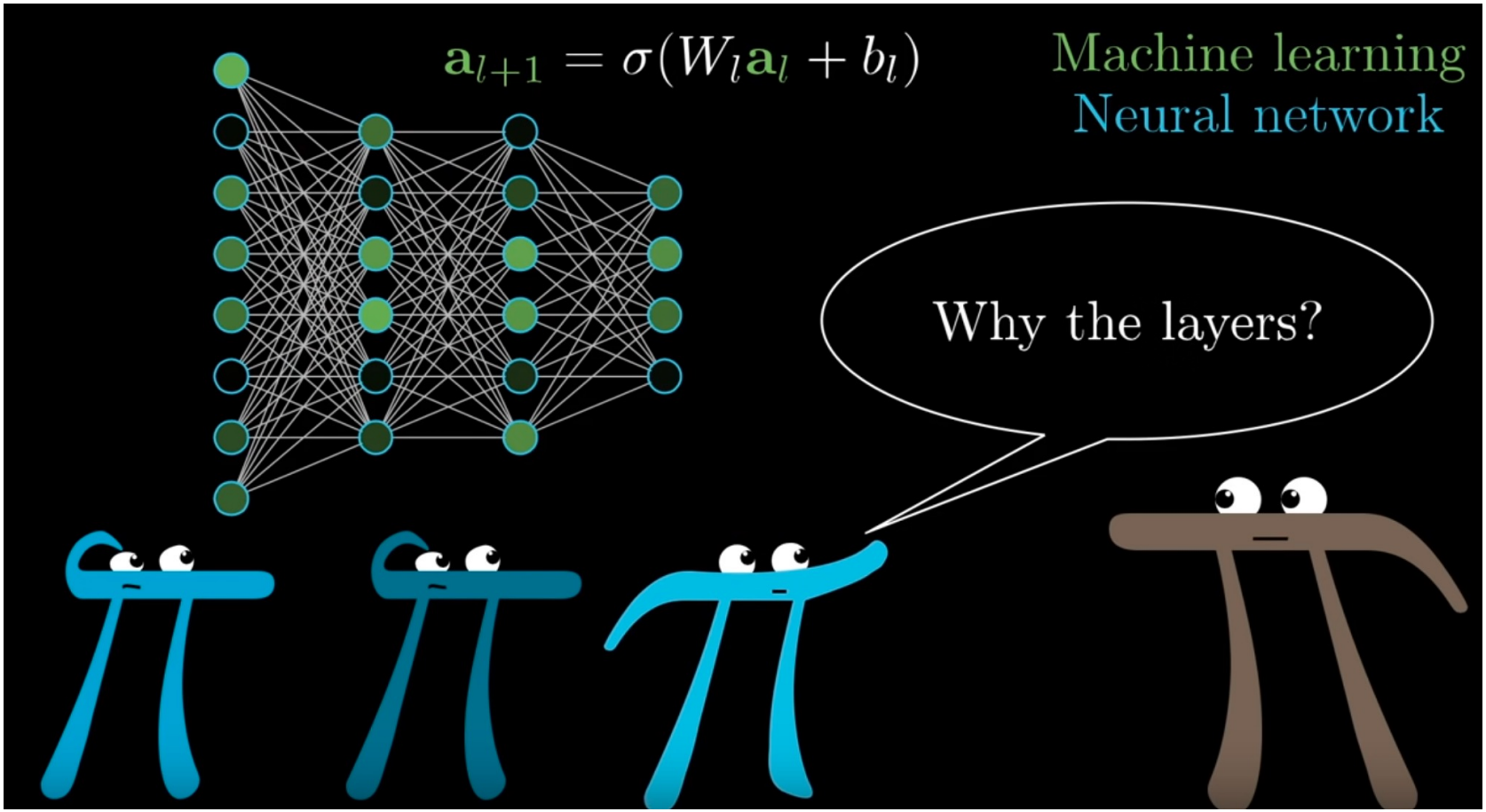
- E.g. Use squared error loss:

$$C(w,b) = \frac{1}{2n}\sum_i (y_i - \widehat{y}(x_i))^2$$

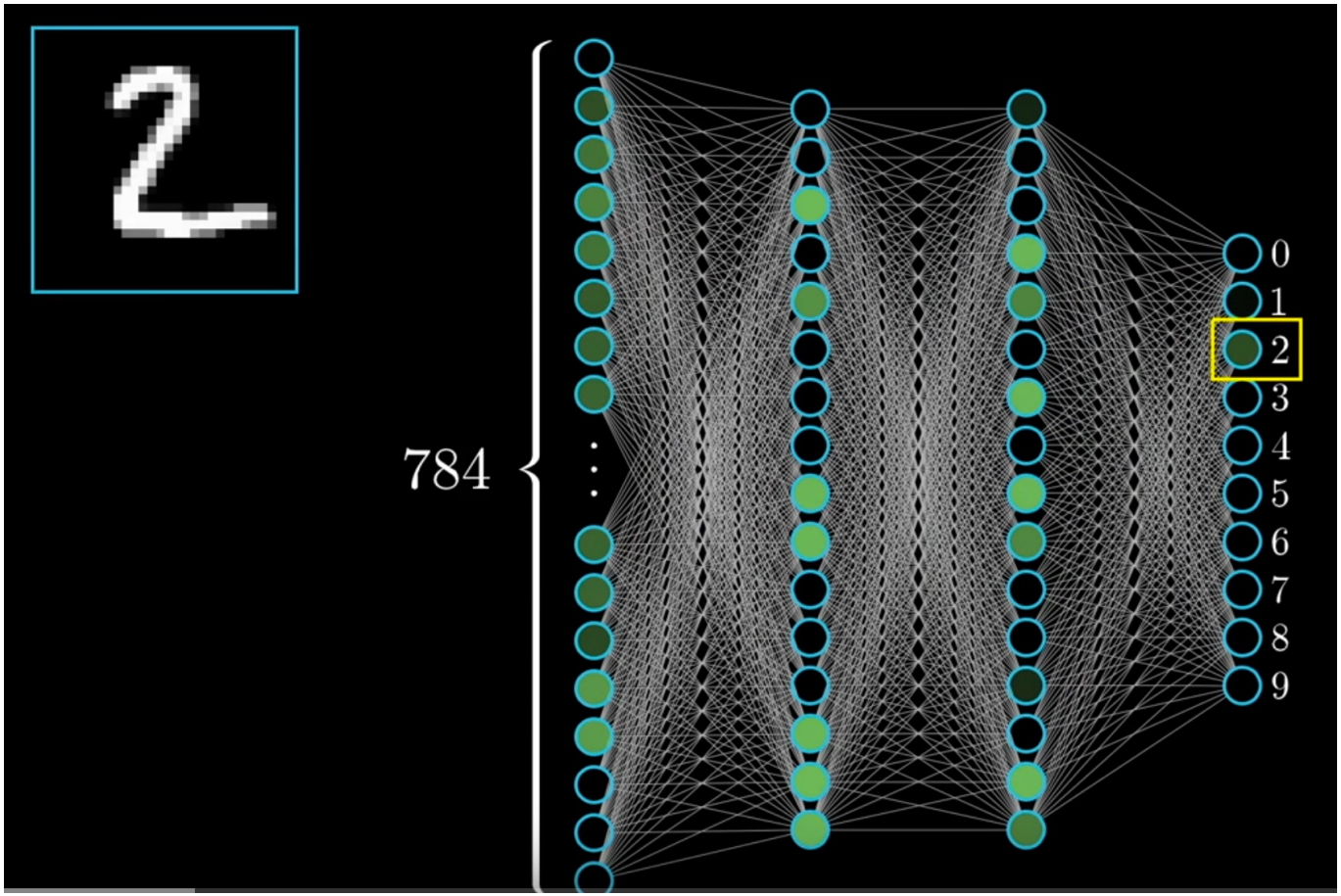
Overview

- Simple introduction into perceptrons and neural networks
- Highlights of the two videos:
  - Basics of neural networks
  - How neural networks learn

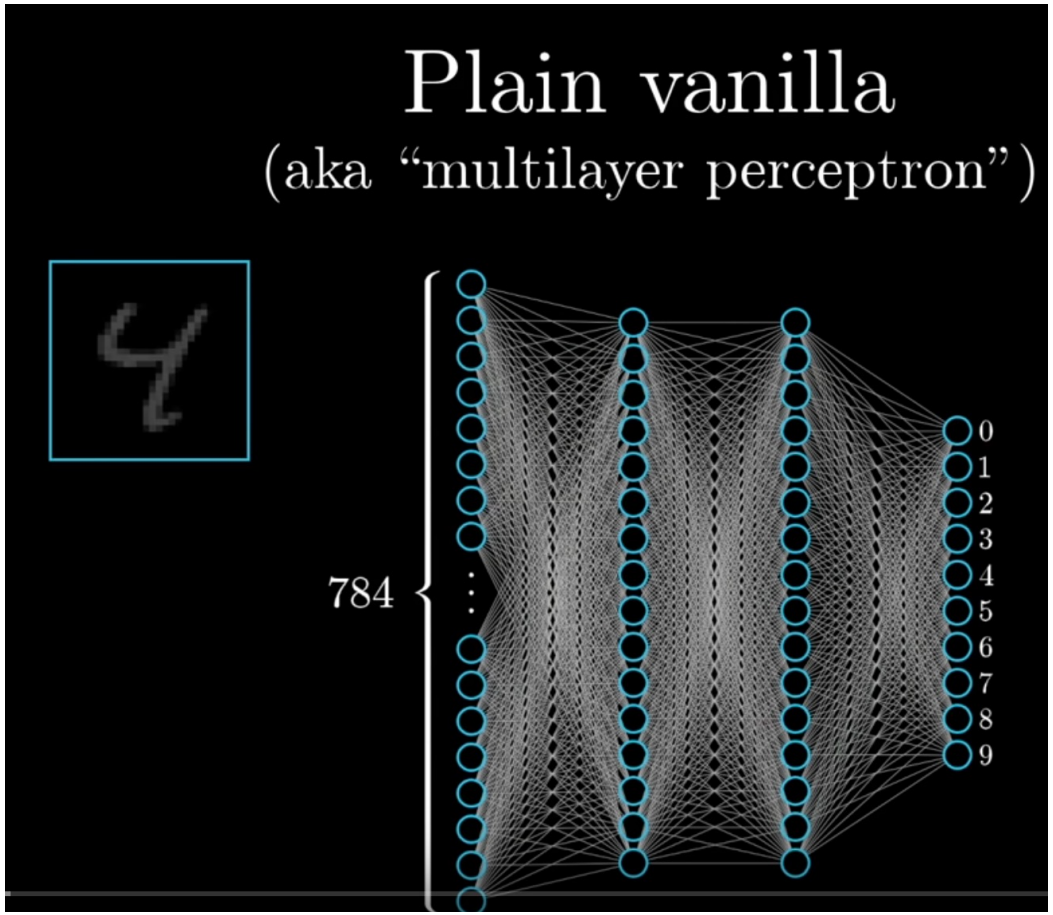
Highlights of the Video



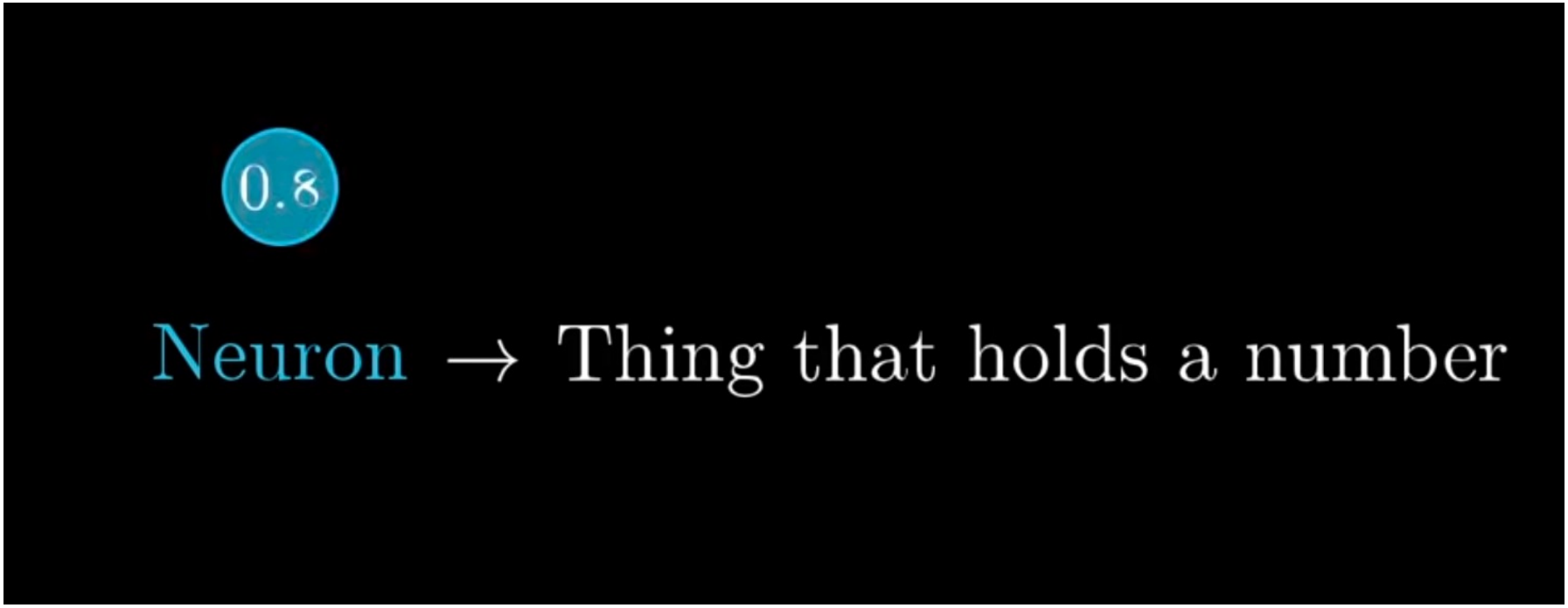
Input 2 → Output 2



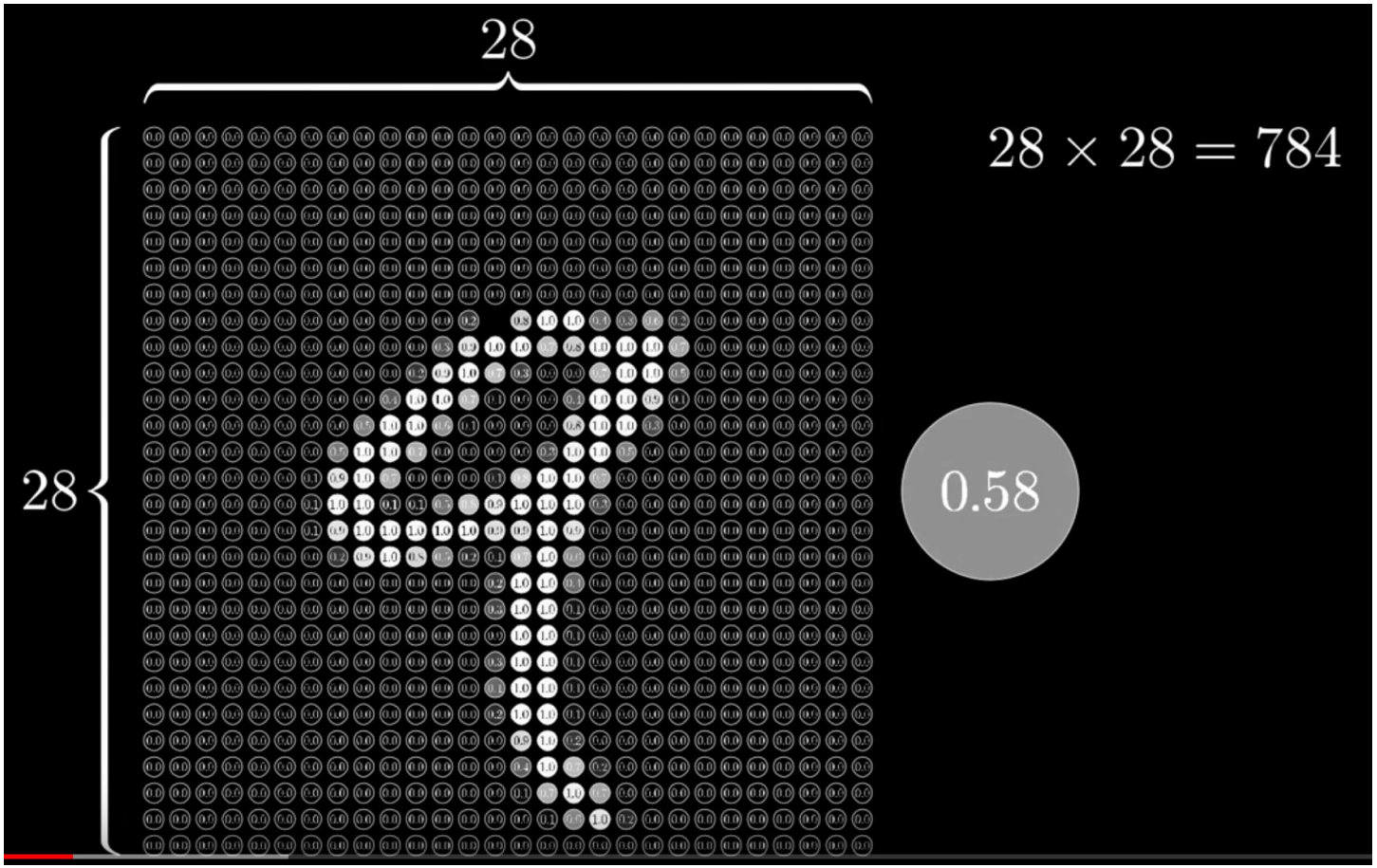
Multilayer Perceptron



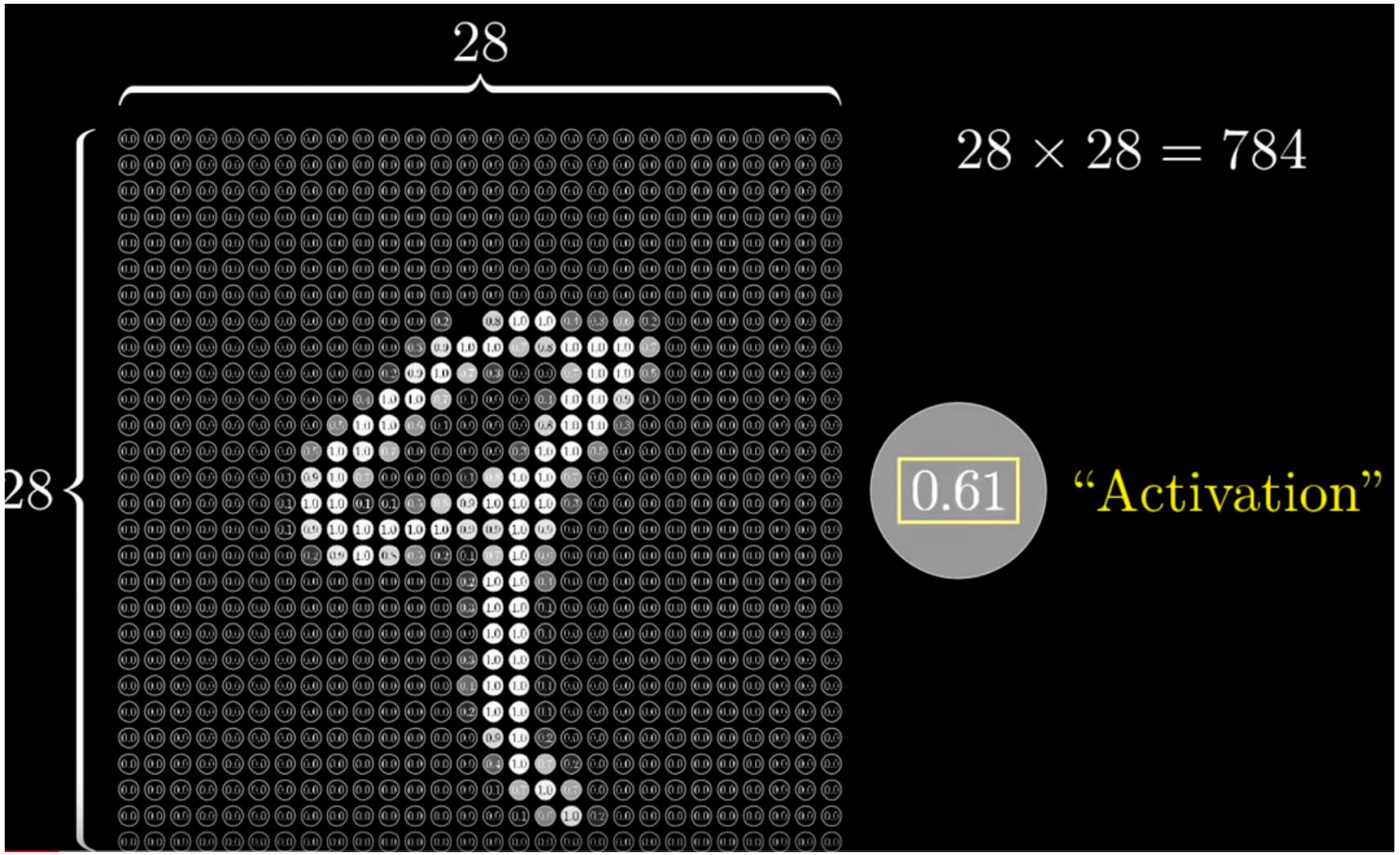
Neuron holds a number = gray scale of pixel



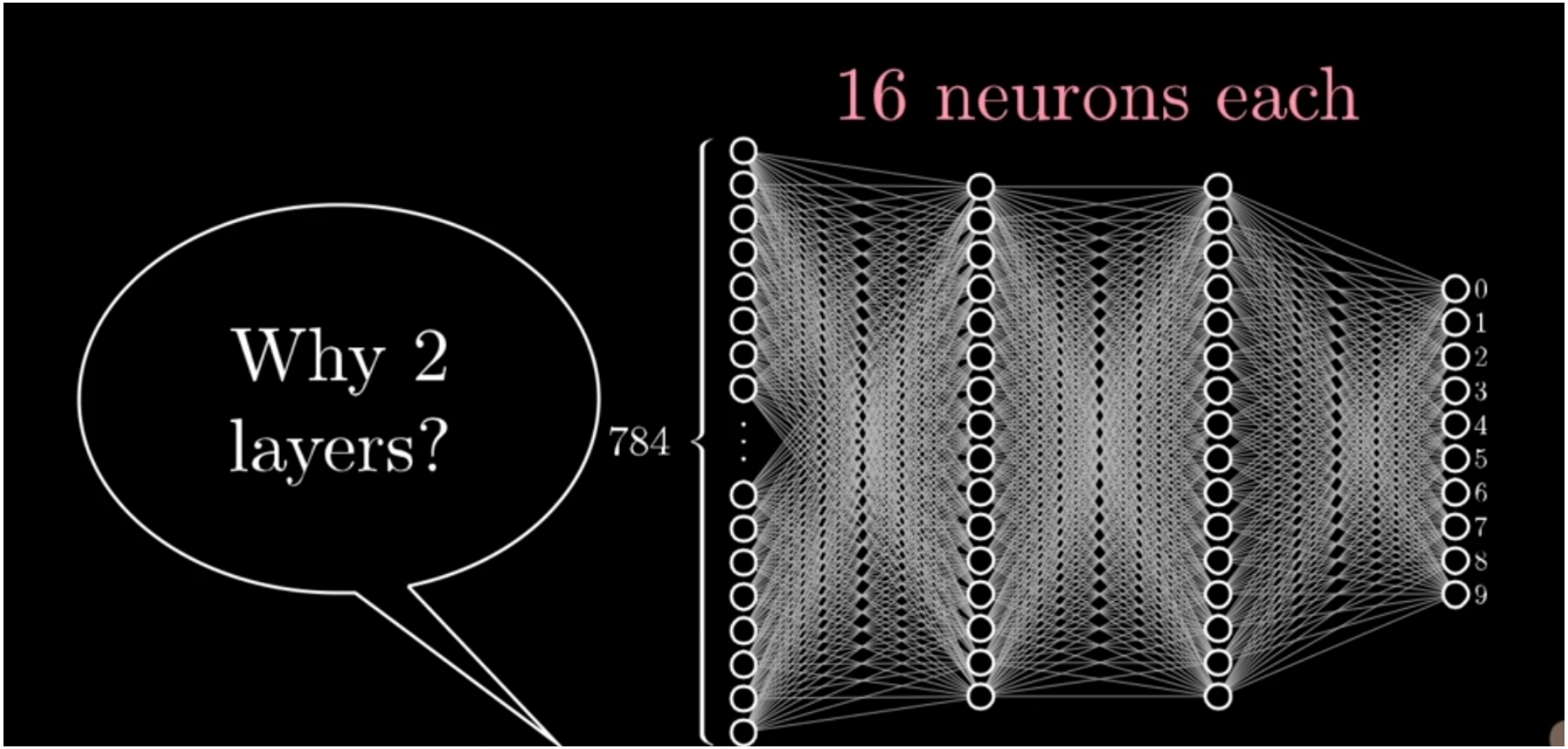
784 neurons to represent 28x28 pixel image



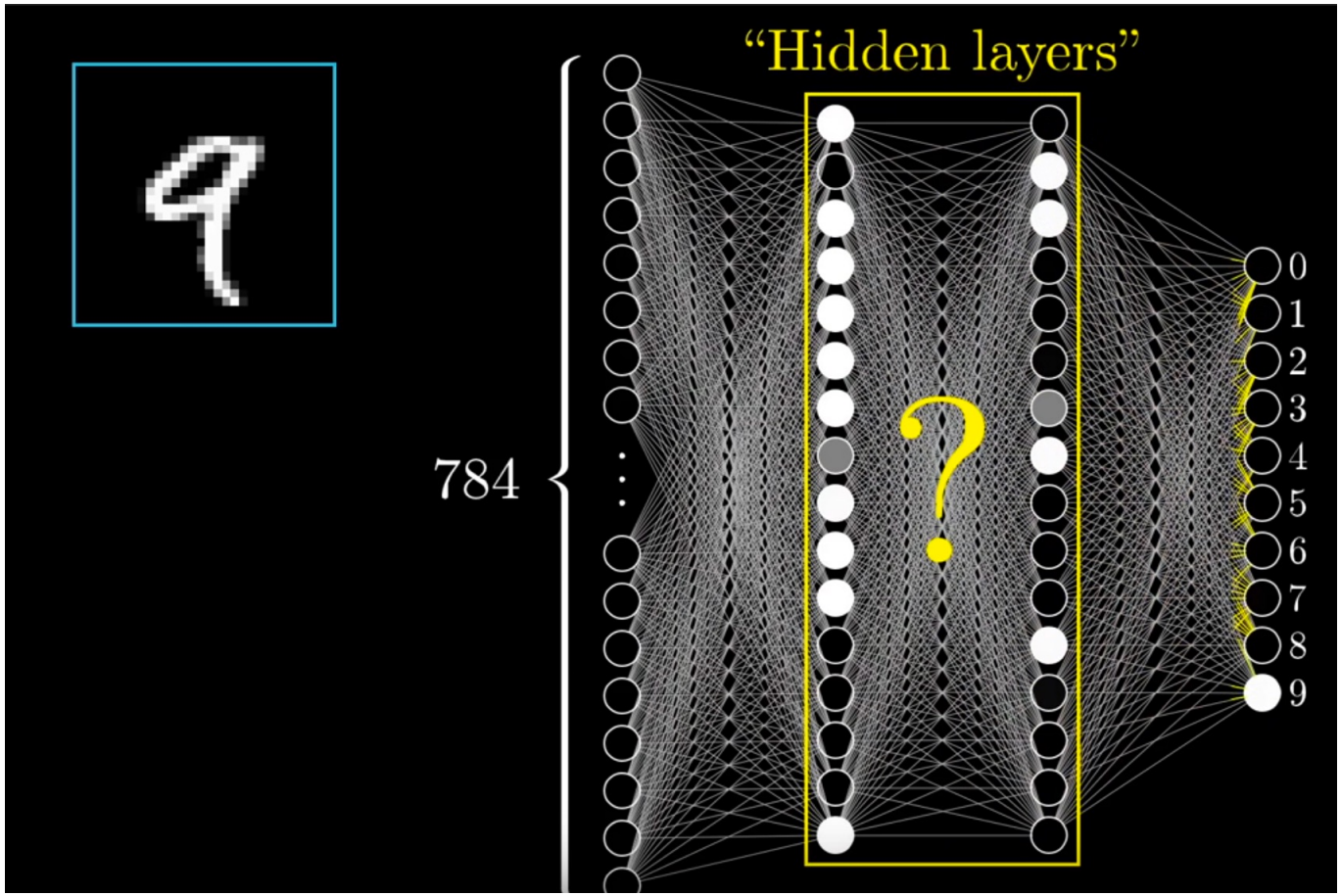
Activation of a neuron



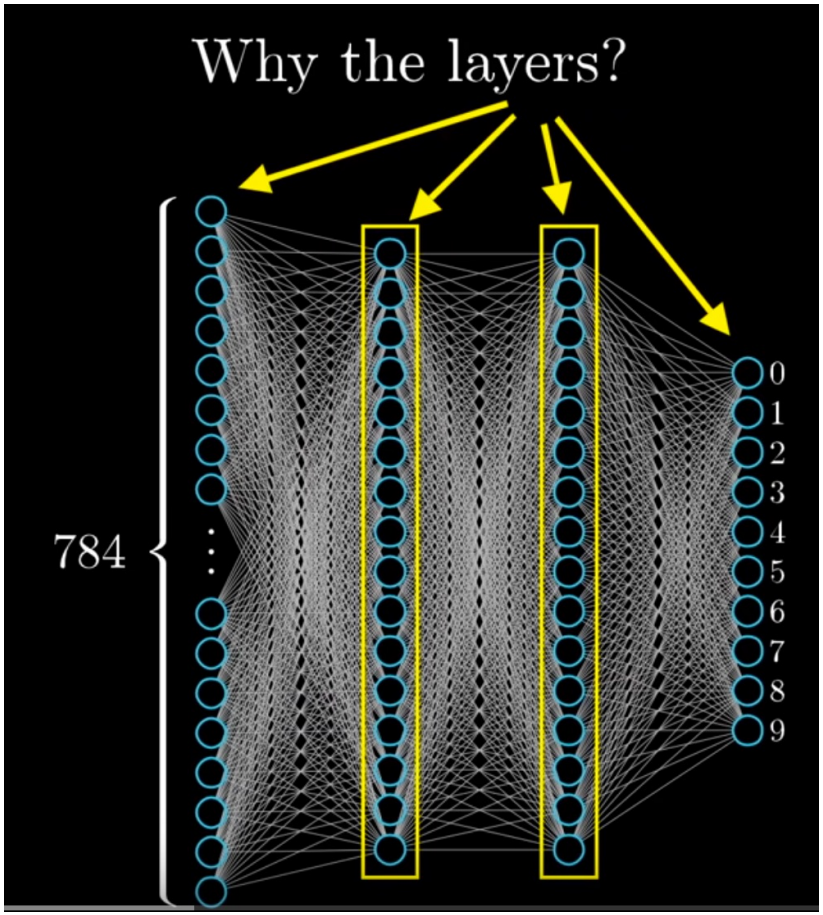
2 hidden layers with 16 neurons each



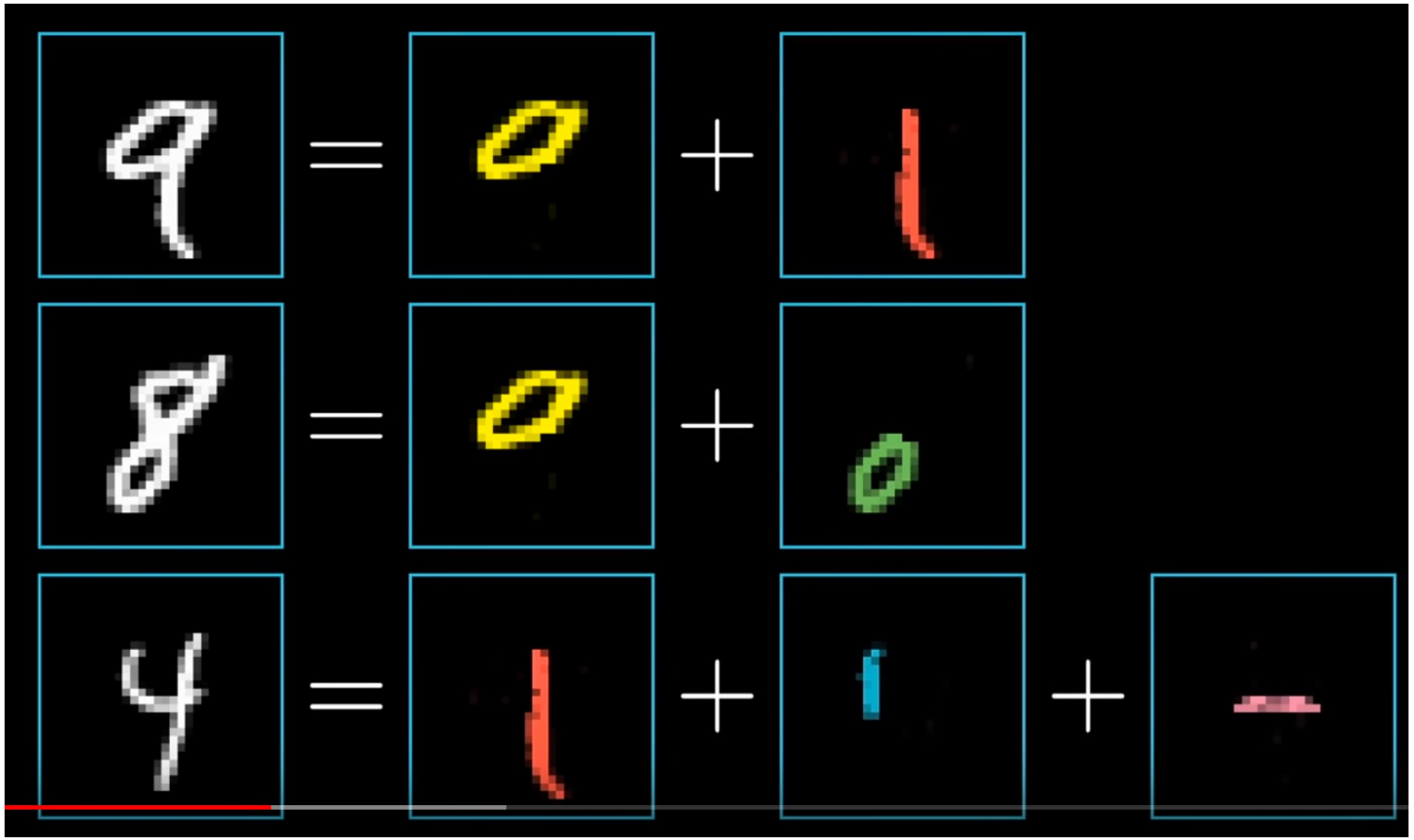
Activation of different layers of trained network



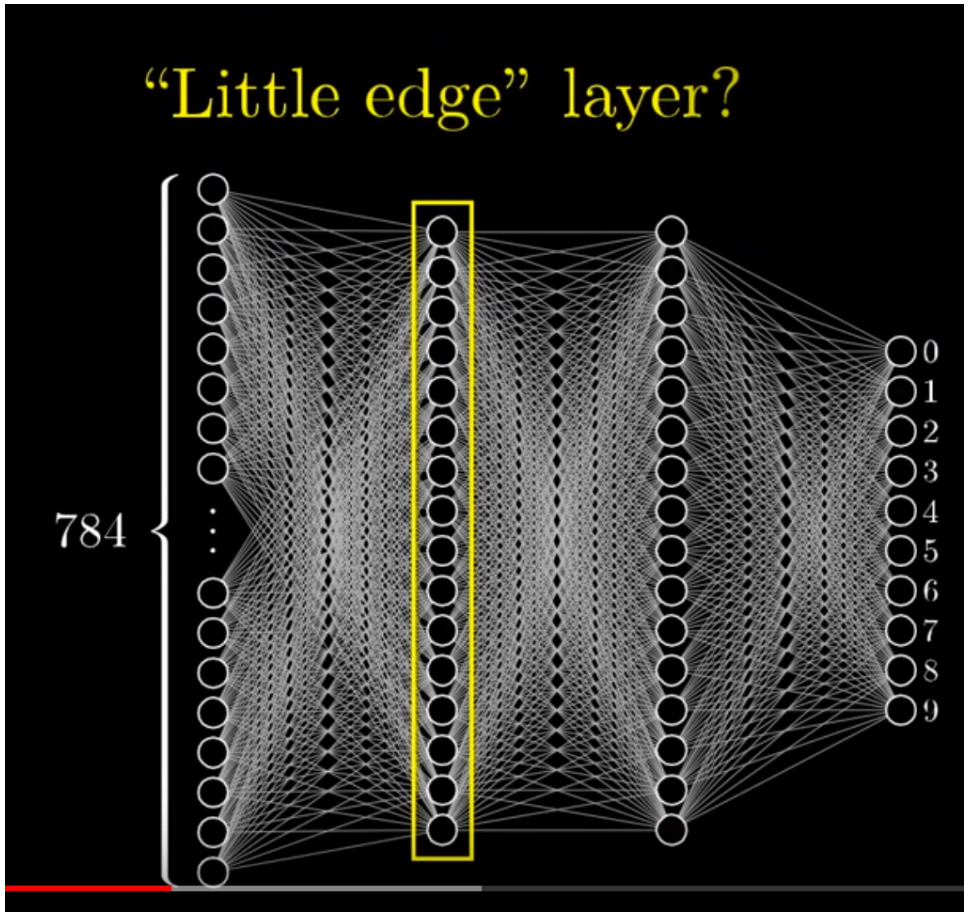
Why hidden layers?



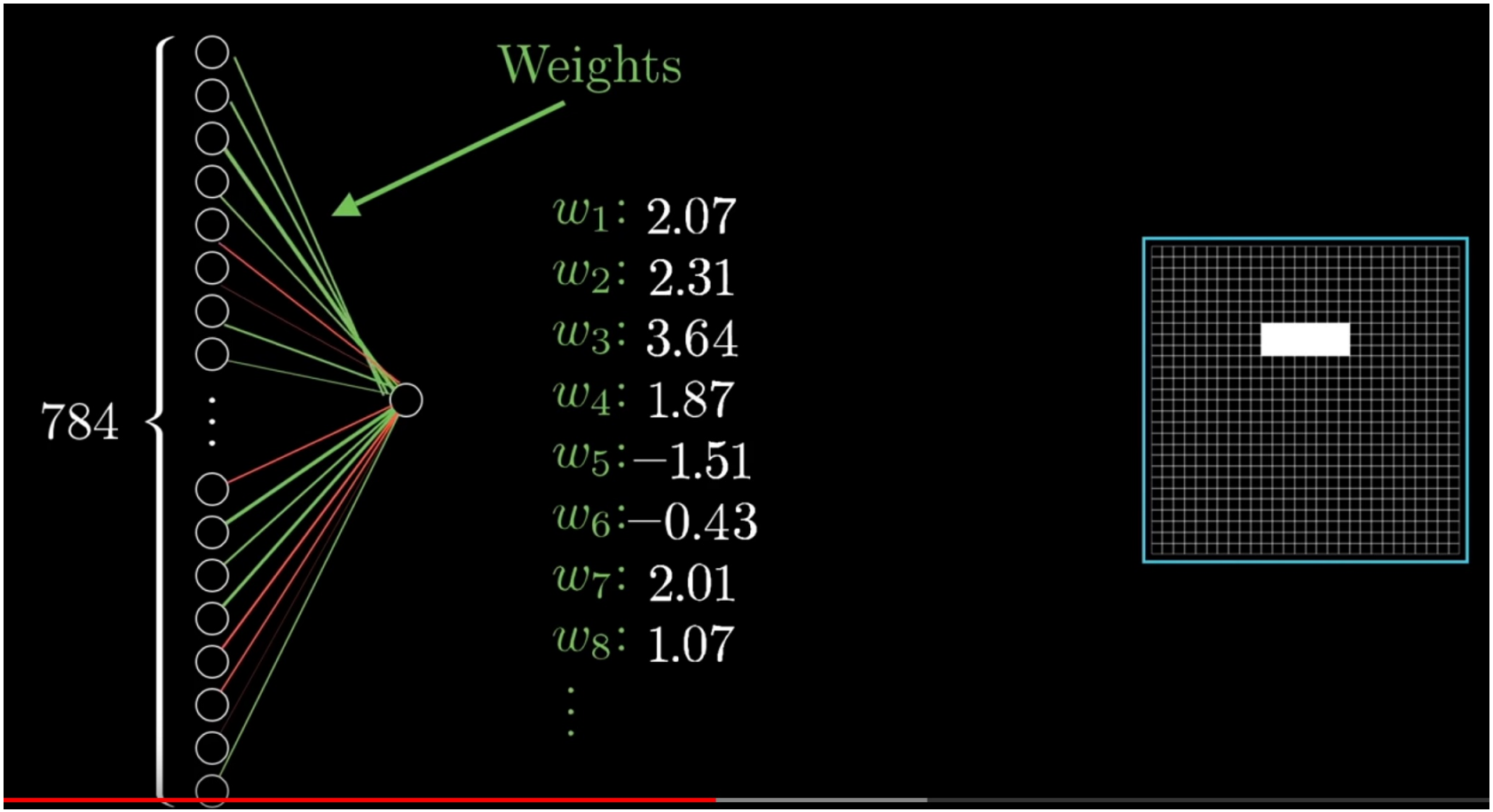
Hidden layers should detect different sub components



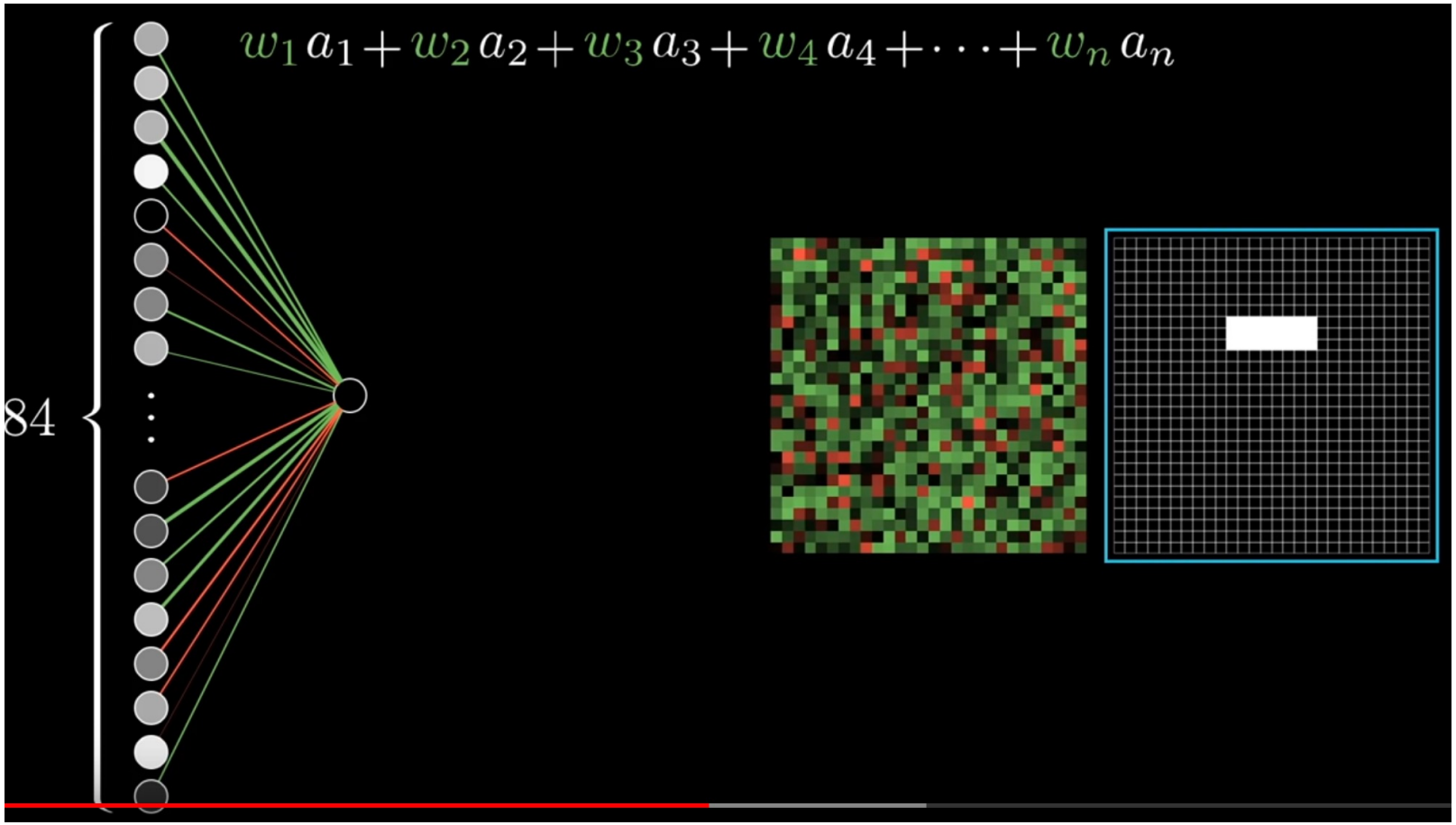
2nd layer detects edges?



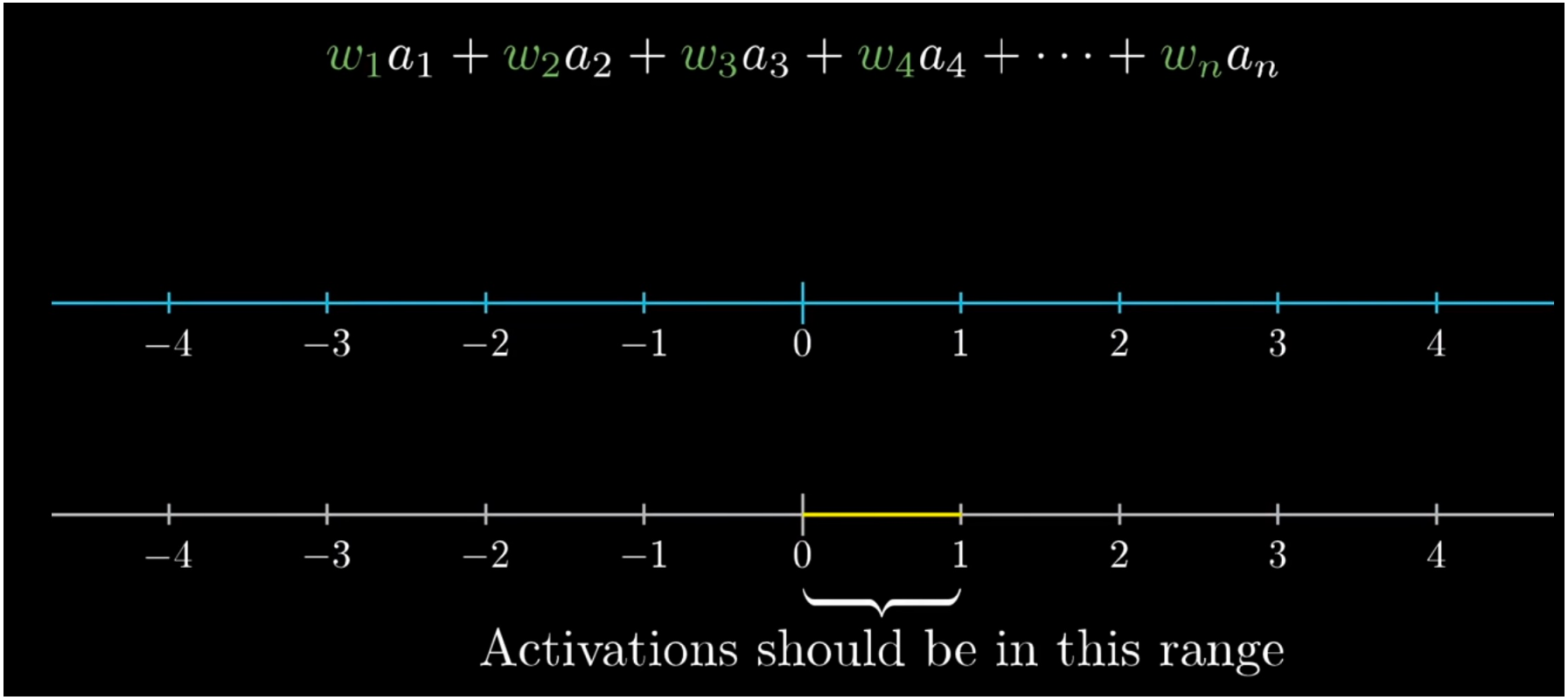
Weights for the connections between neurons



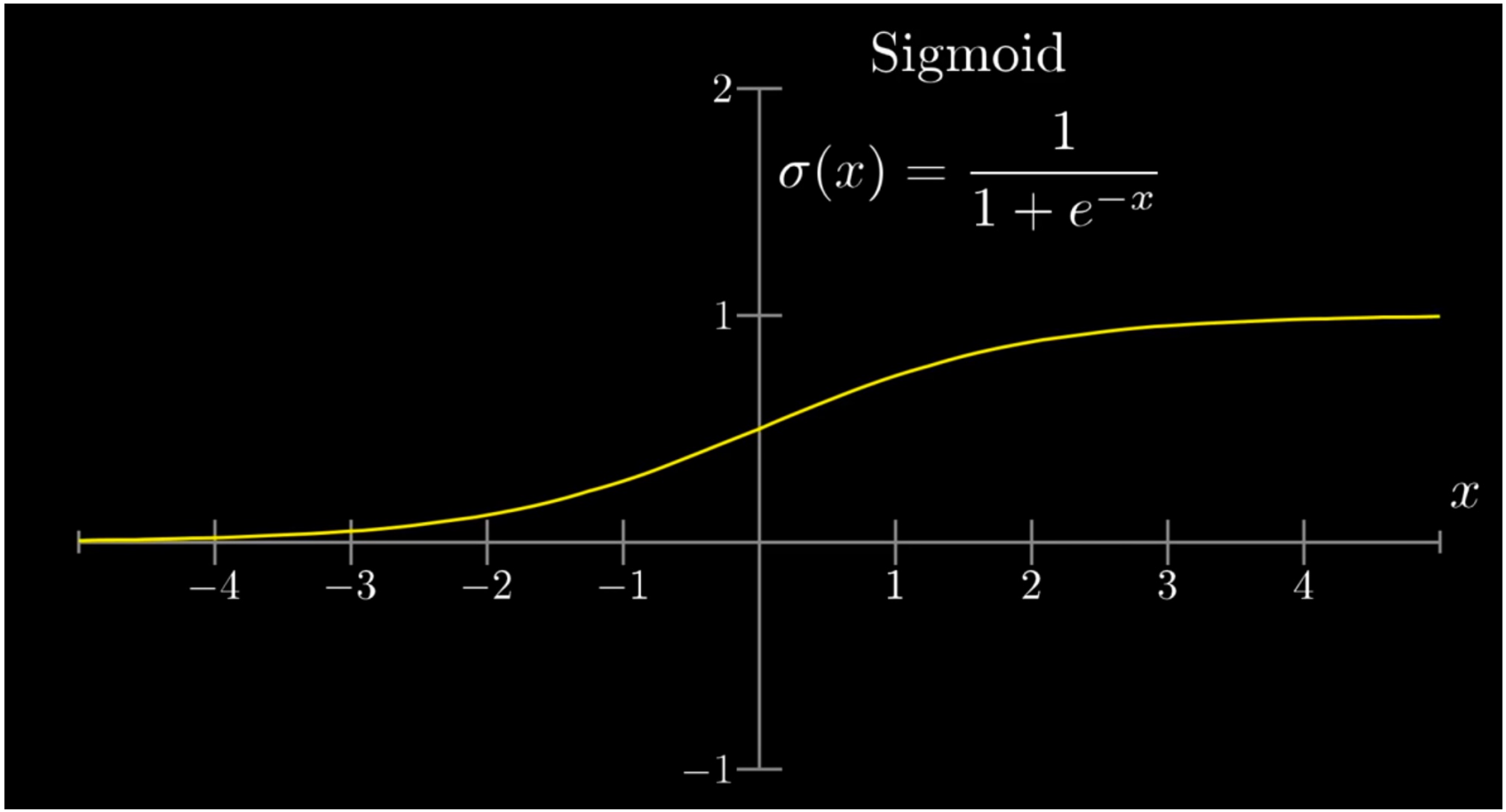
Weighted sum over all weights



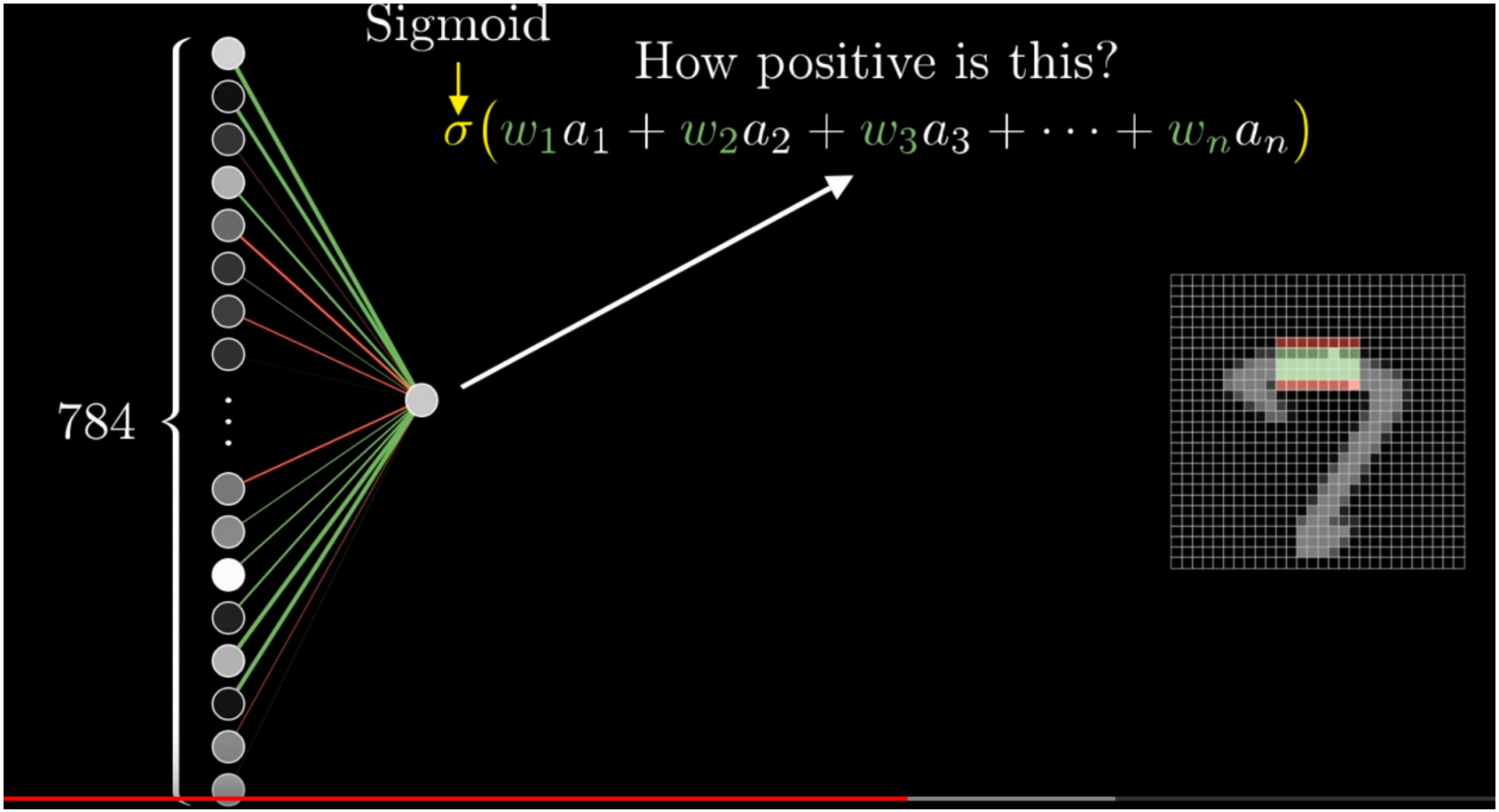
Activation between 0 and 1



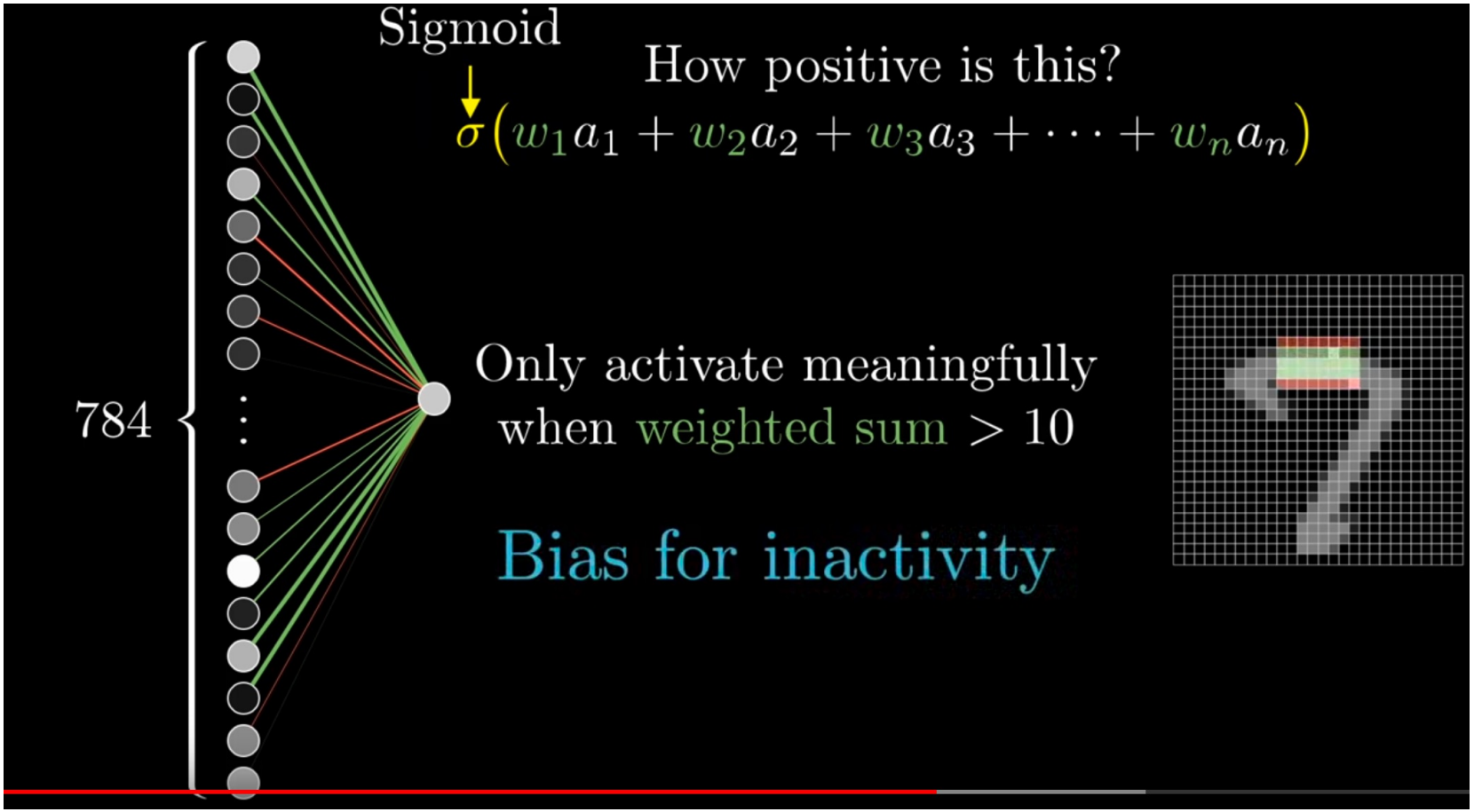
Use Sigmoid function (again)



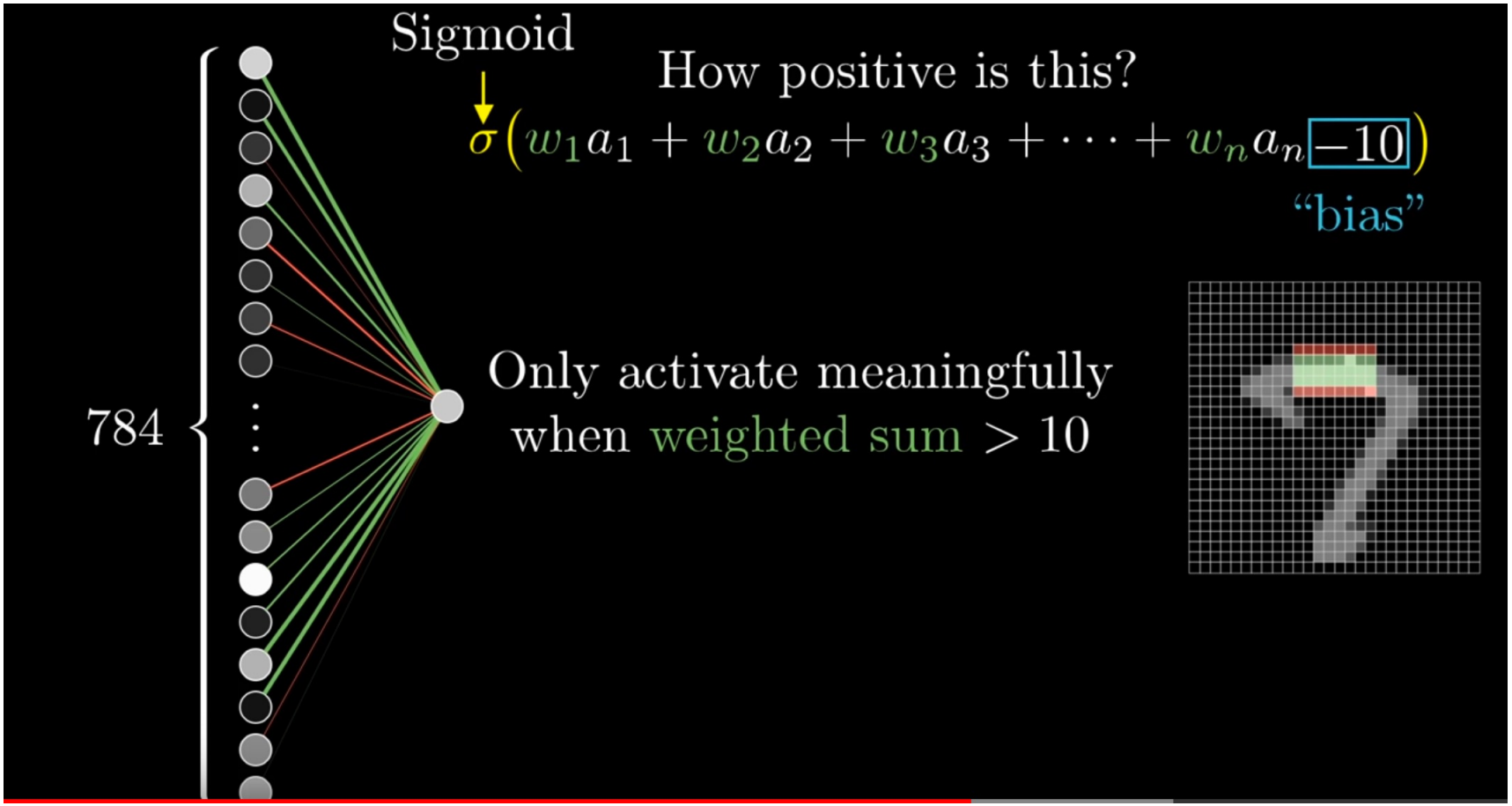
Activation = how positive is the weighted sum



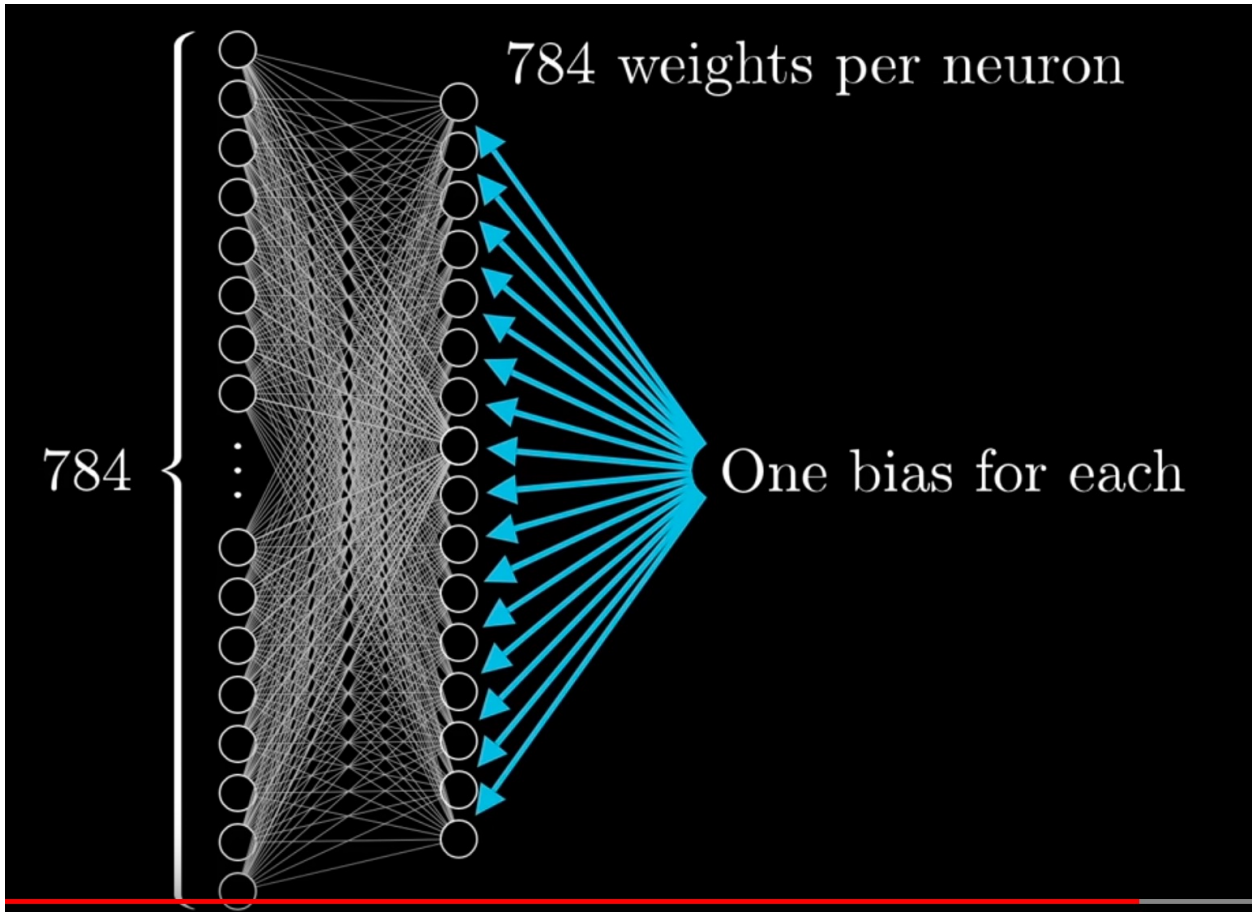
Add bias for inactivity if weighted sum <= 10



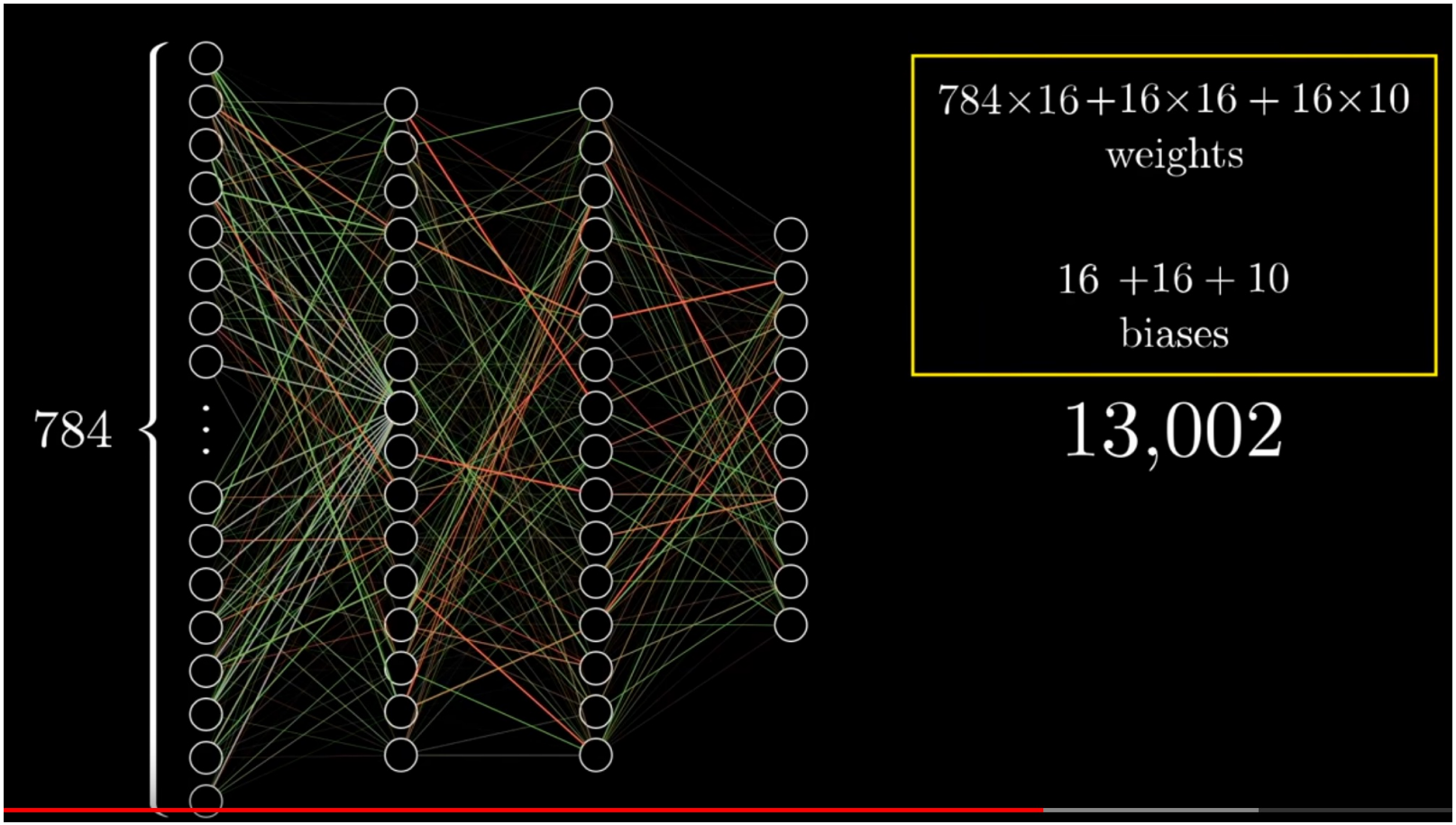
Add bias before applying sigmoid function



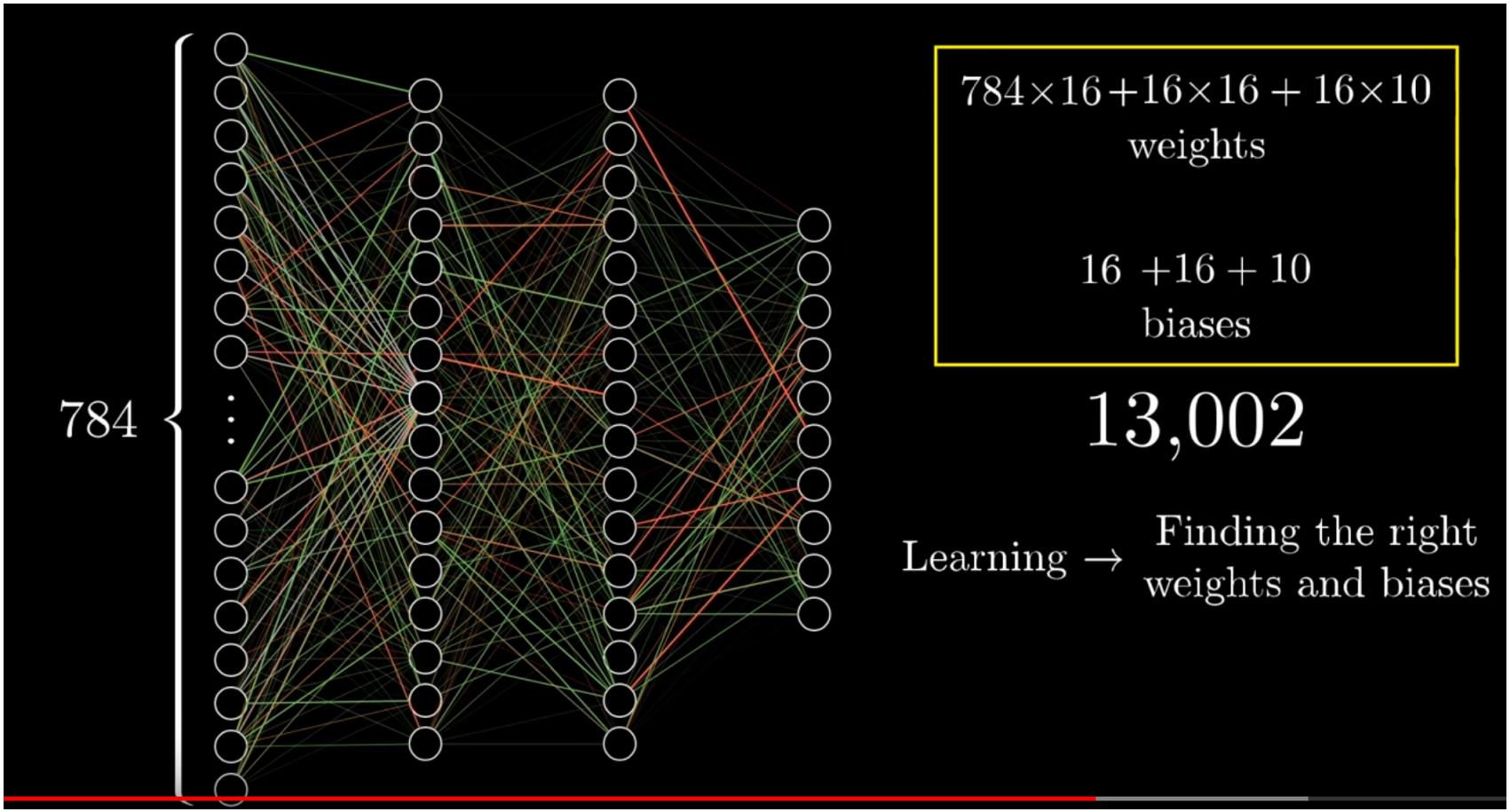
Many weights and biases



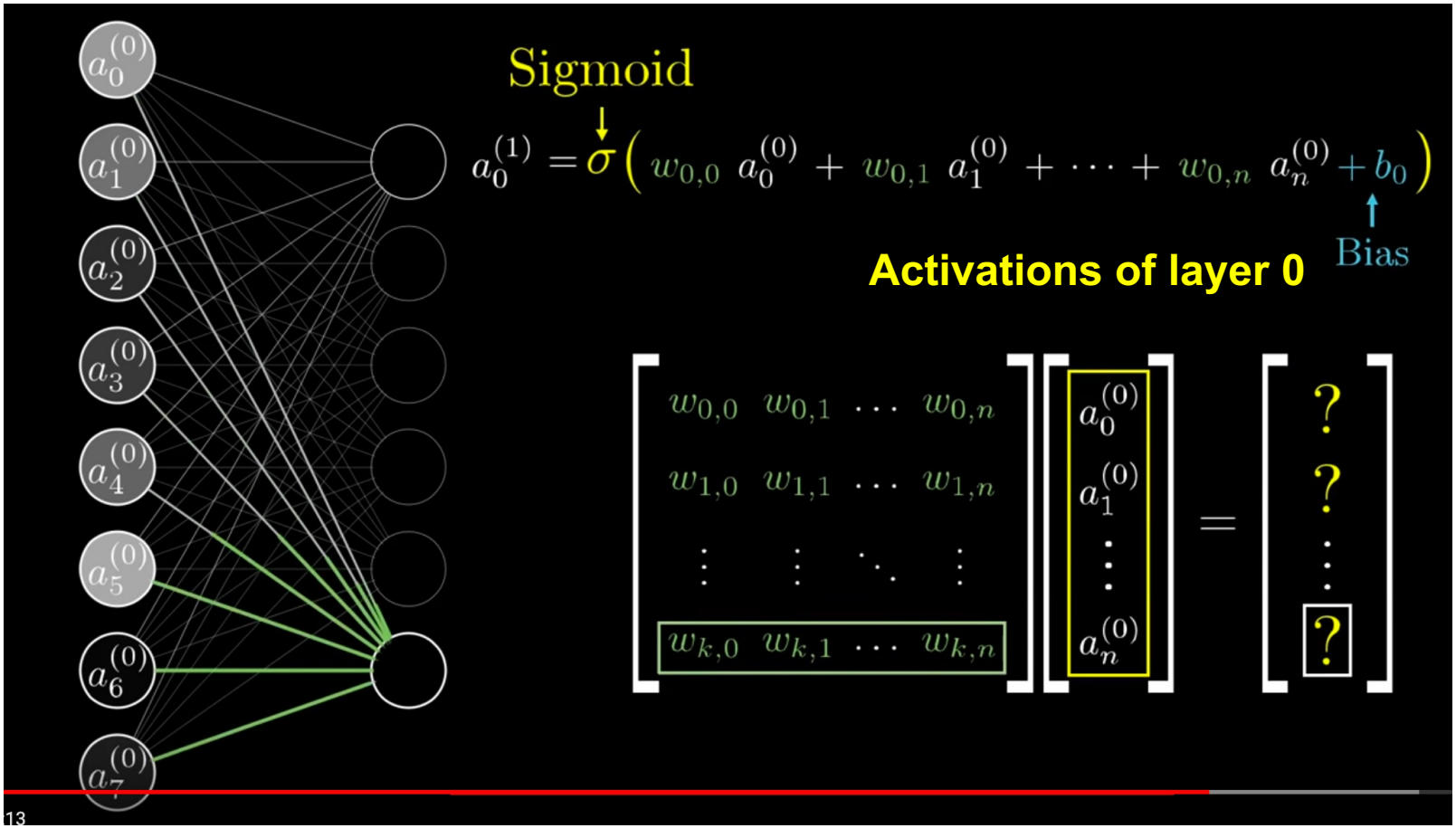
13,002 weights & biases



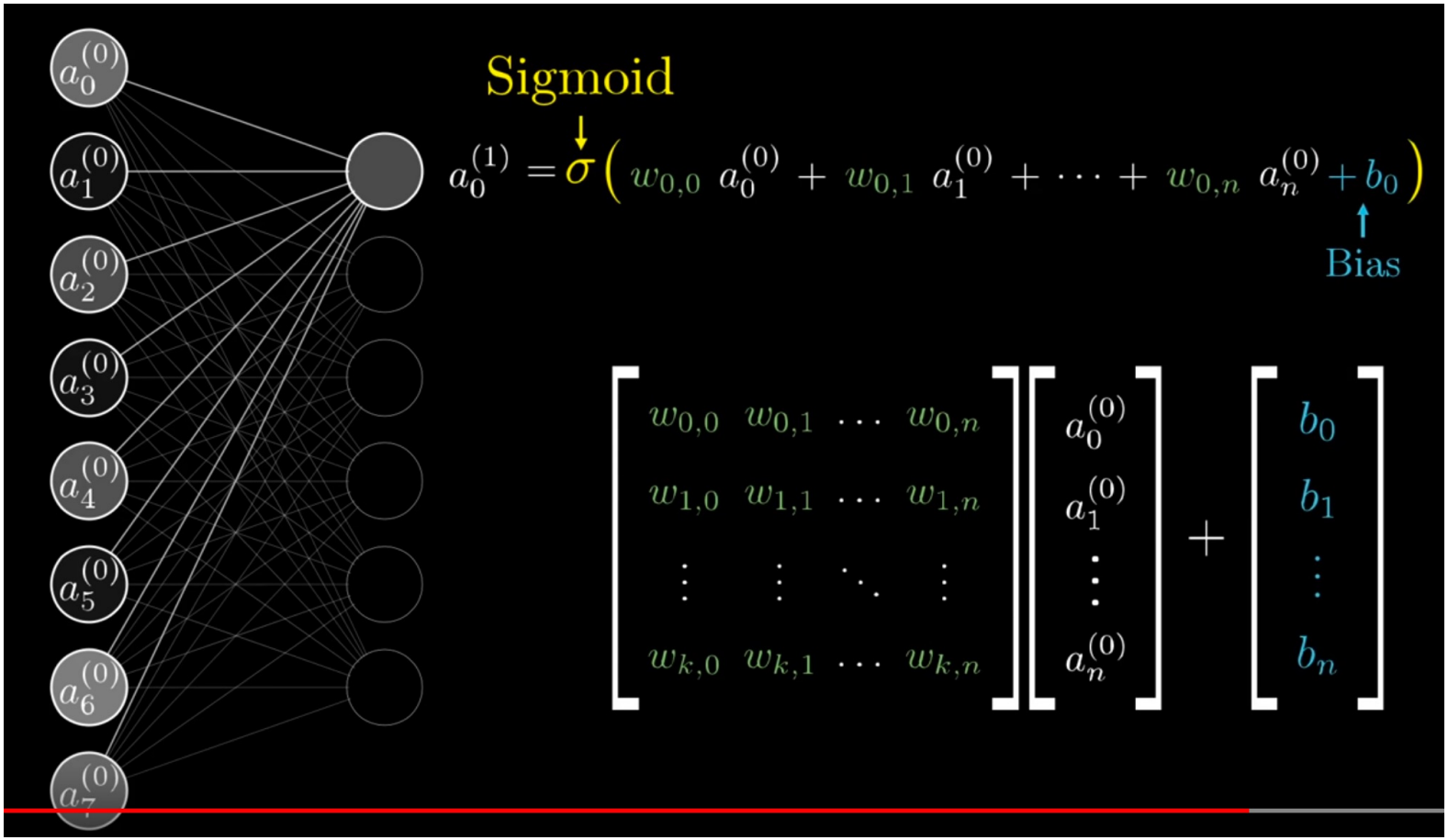
Learning = finding the right weights and biases



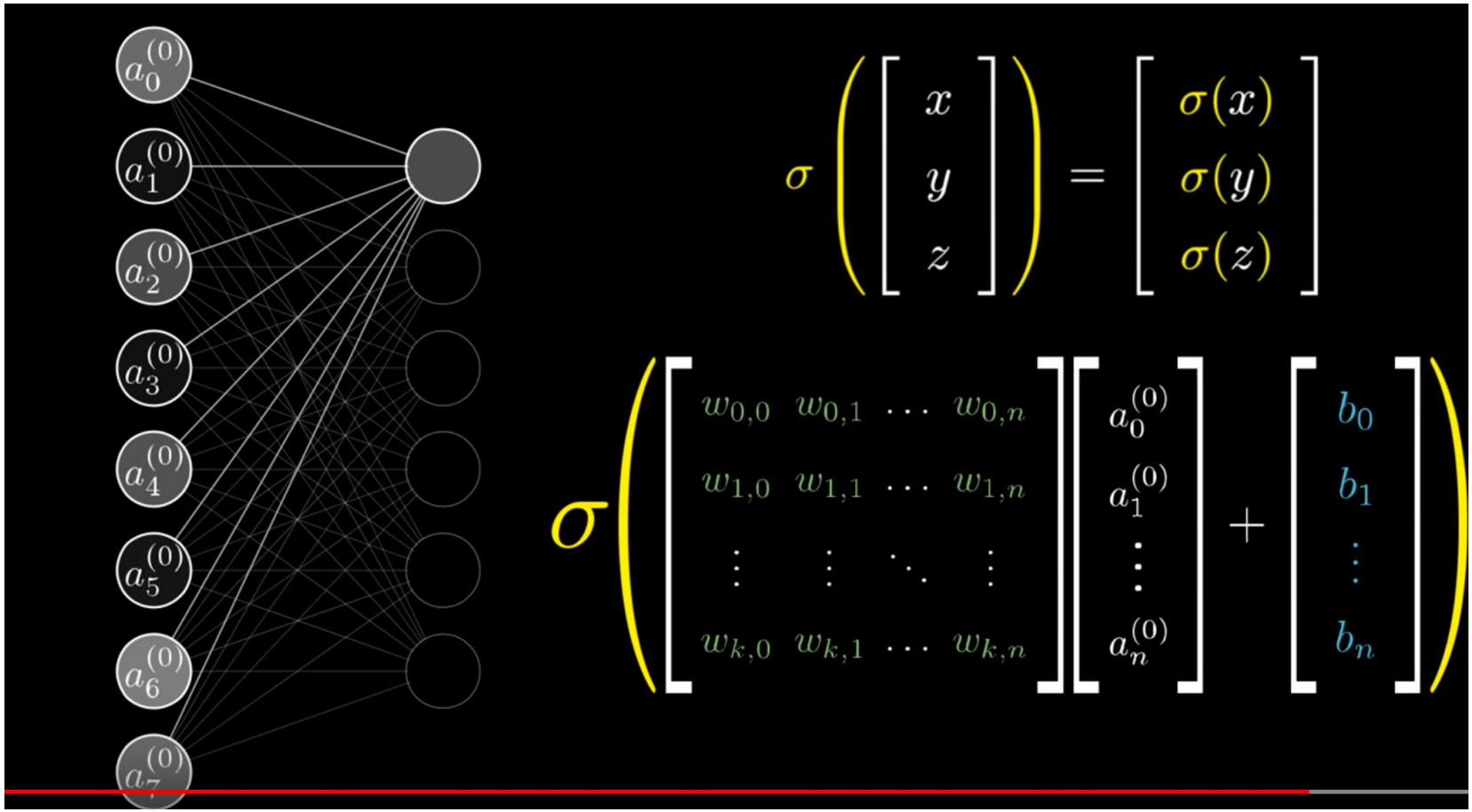
Dot product = Matrix-vector-product



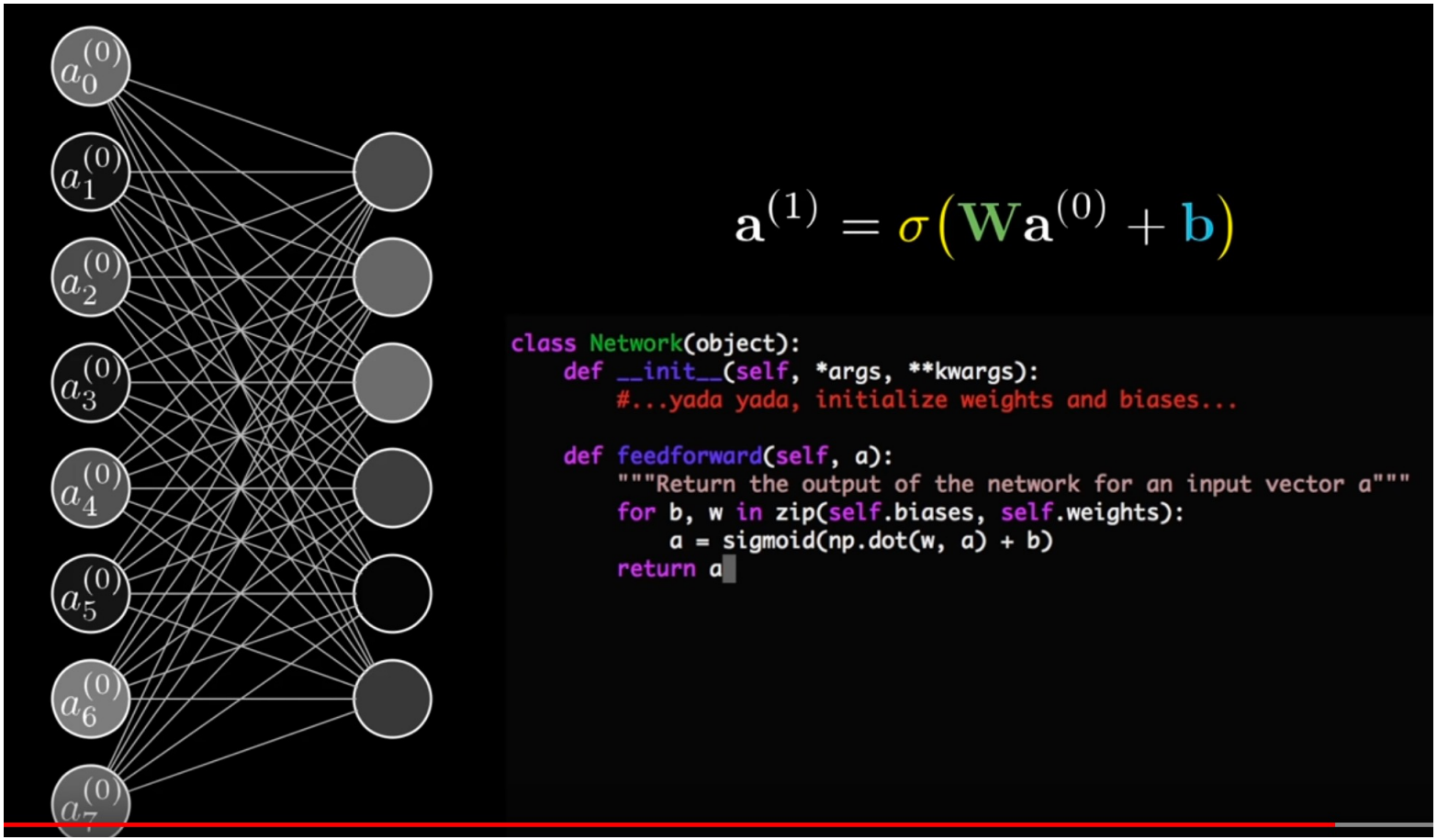
Add bias vector



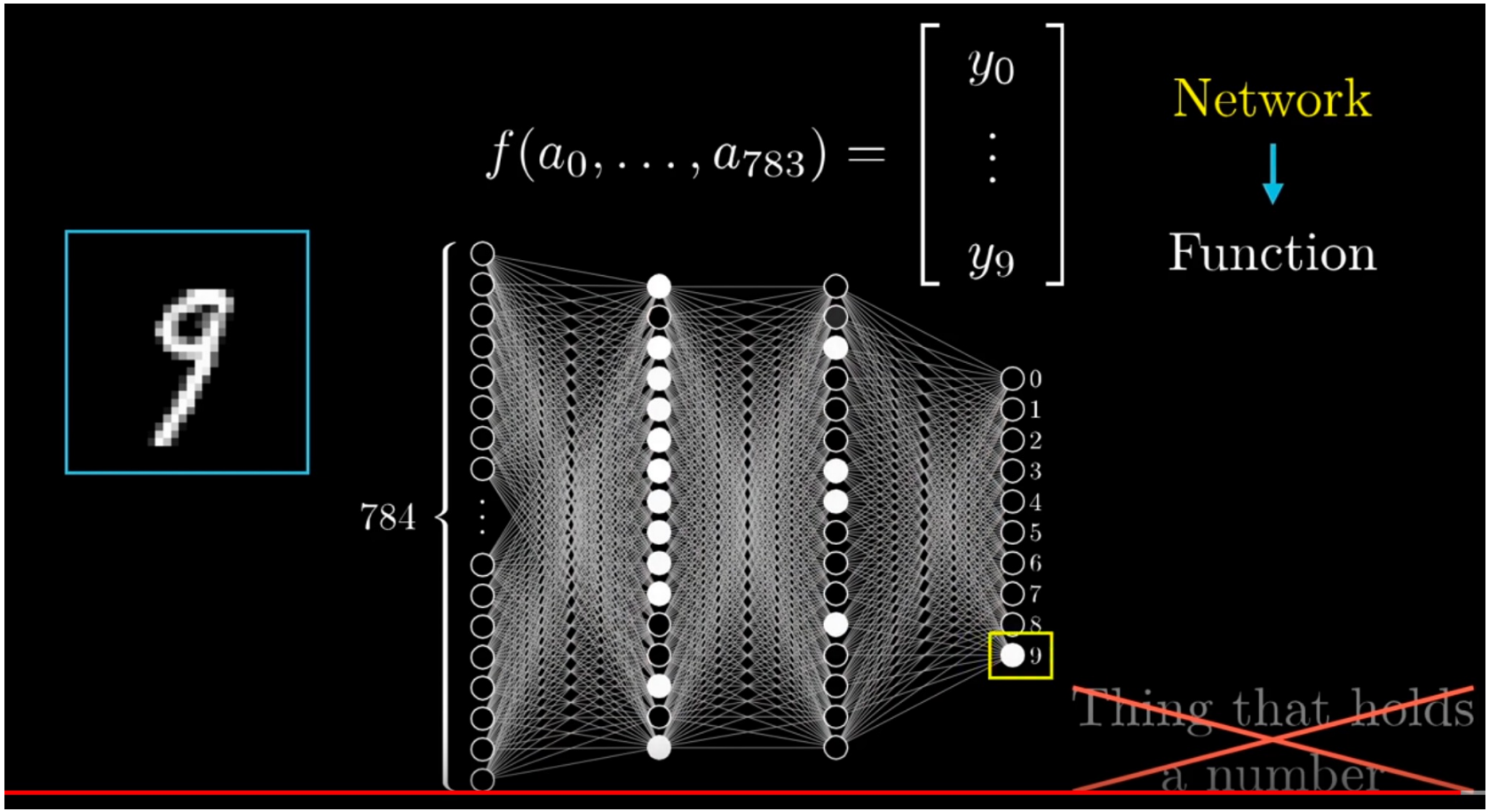
Apply sigmoid function



Code


$a_0^{(0)}$
$a_1^{(0)}$
$a_2^{(0)}$
$a_3^{(0)}$
$a_4^{(0)}$
$a_5^{(0)}$
$a_6^{(0)}$
$a_7^{(0)}$
$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$
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

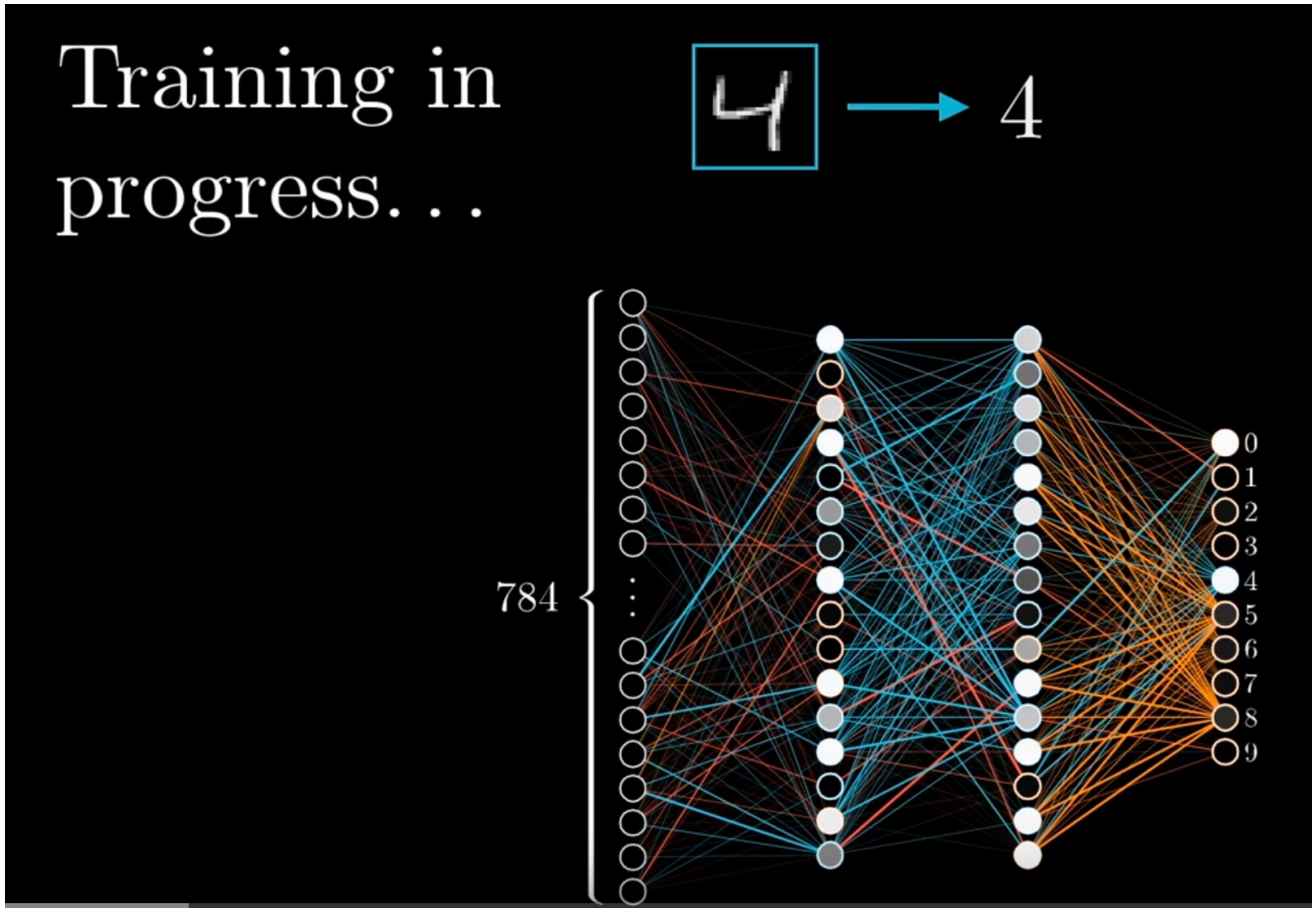
Neuron + network of neurons = function



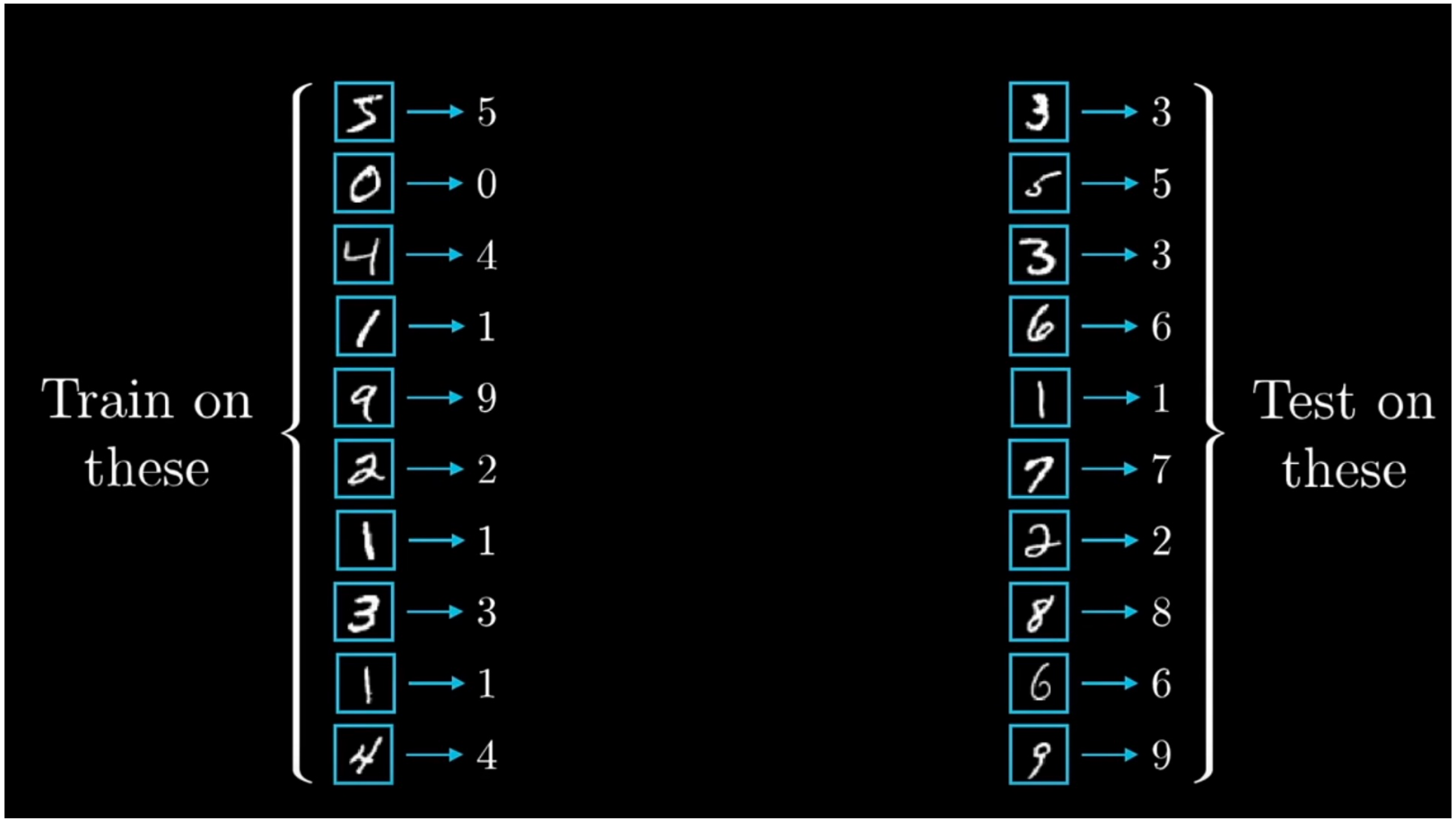
Overview

- Simple introduction into perceptrons and neural networks
- Highlights of the two videos:
  - Basics of neural networks
  - How neural networks learn

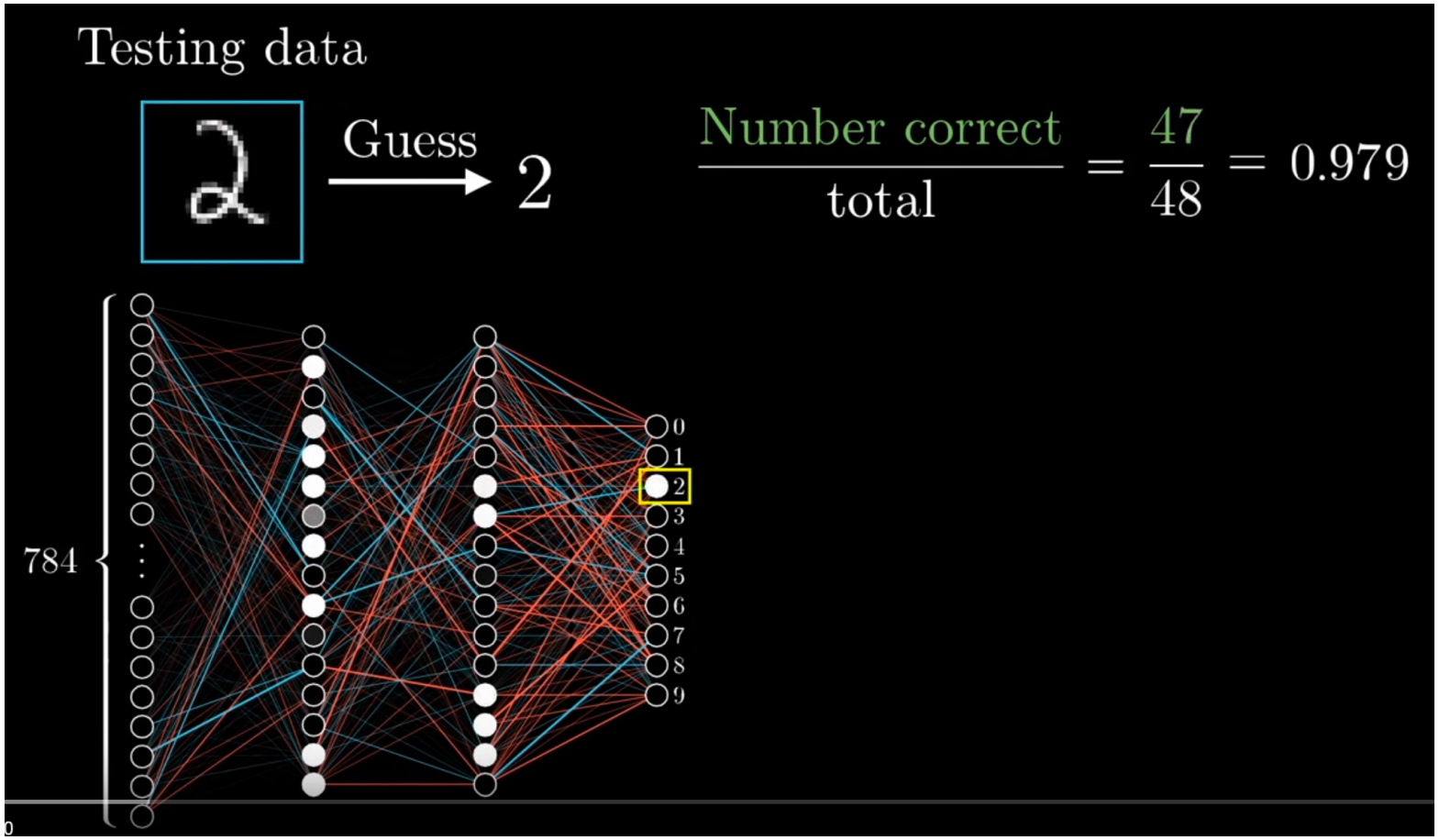
Training with Label “4”



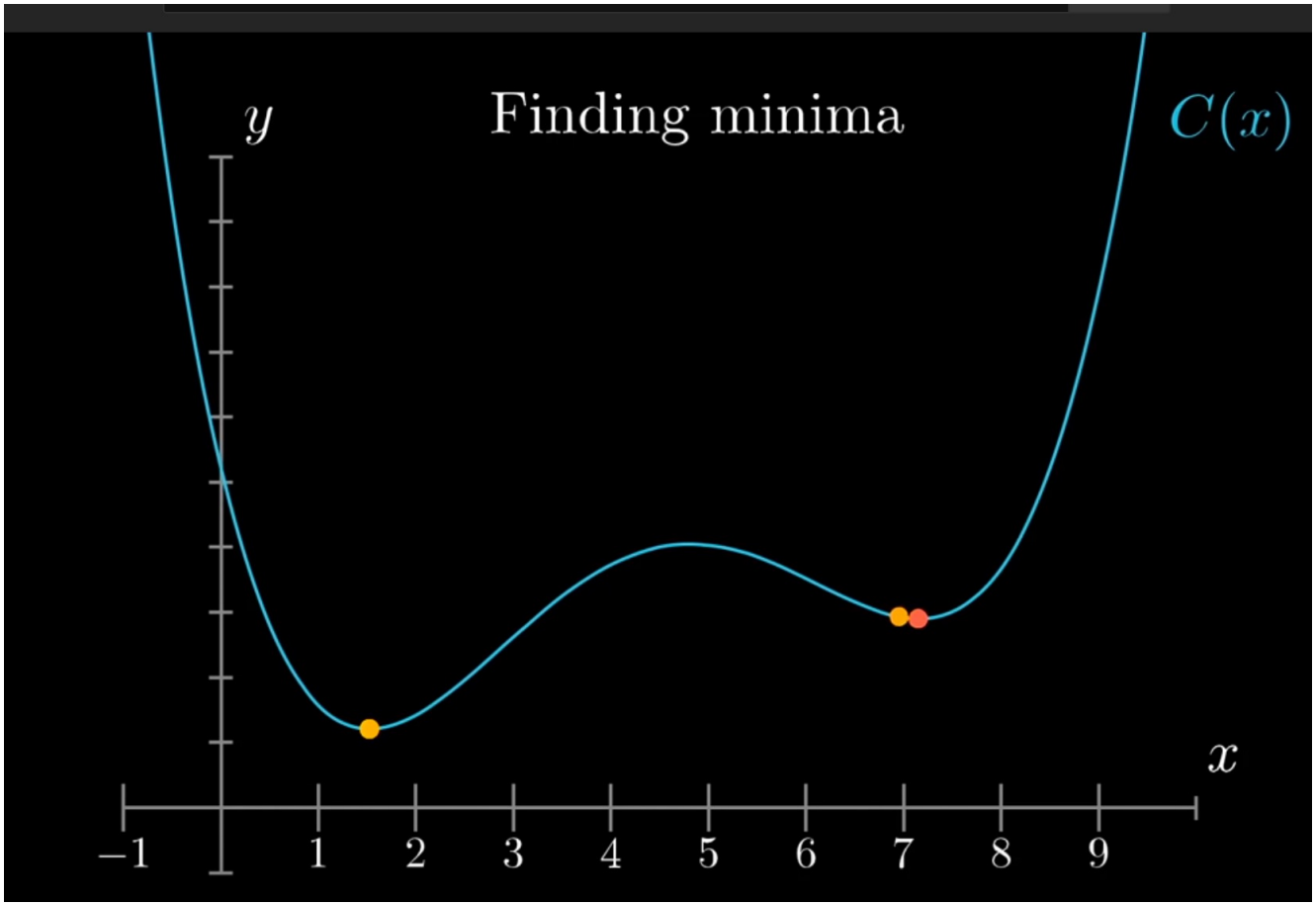
Training and testing



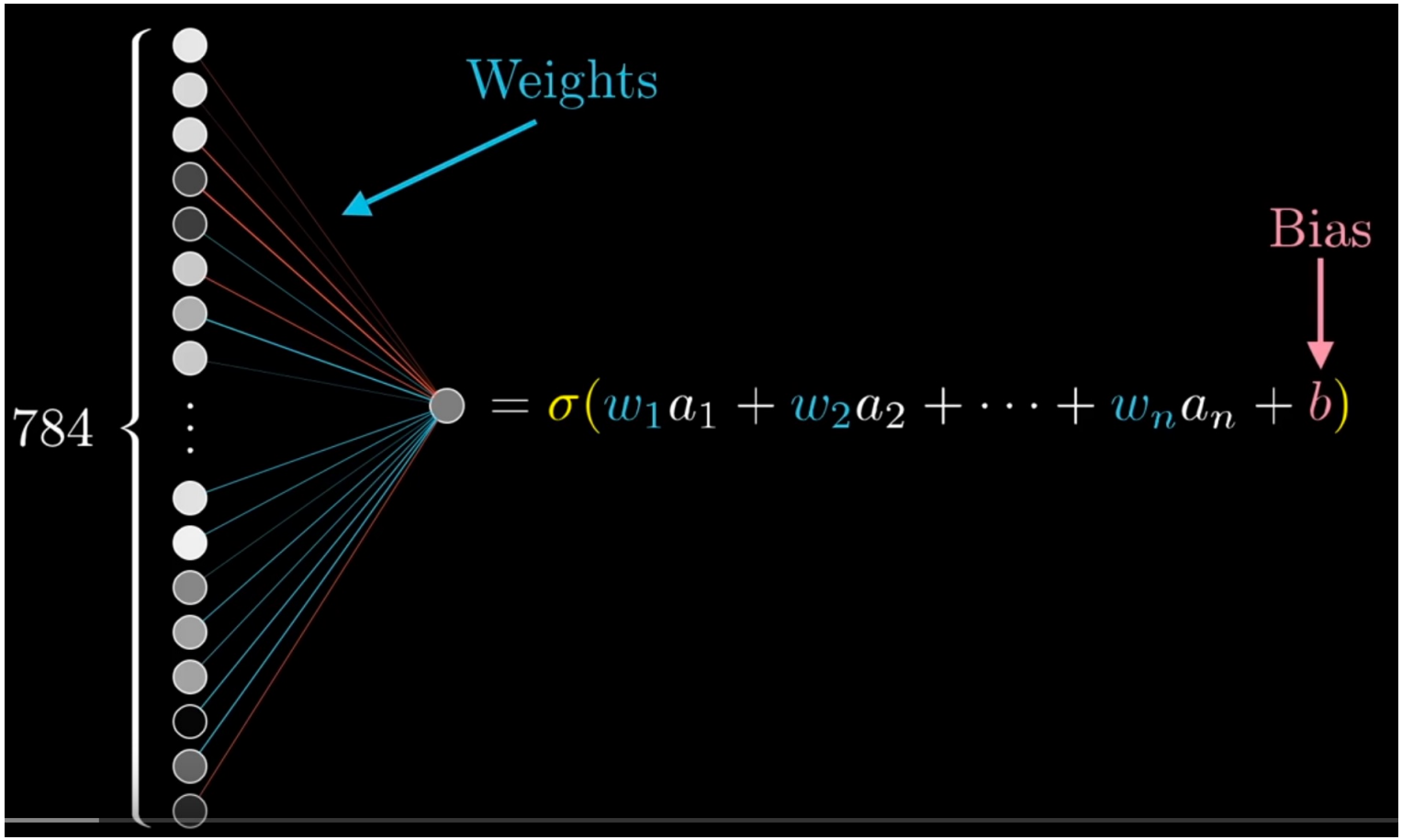
Test evaluation



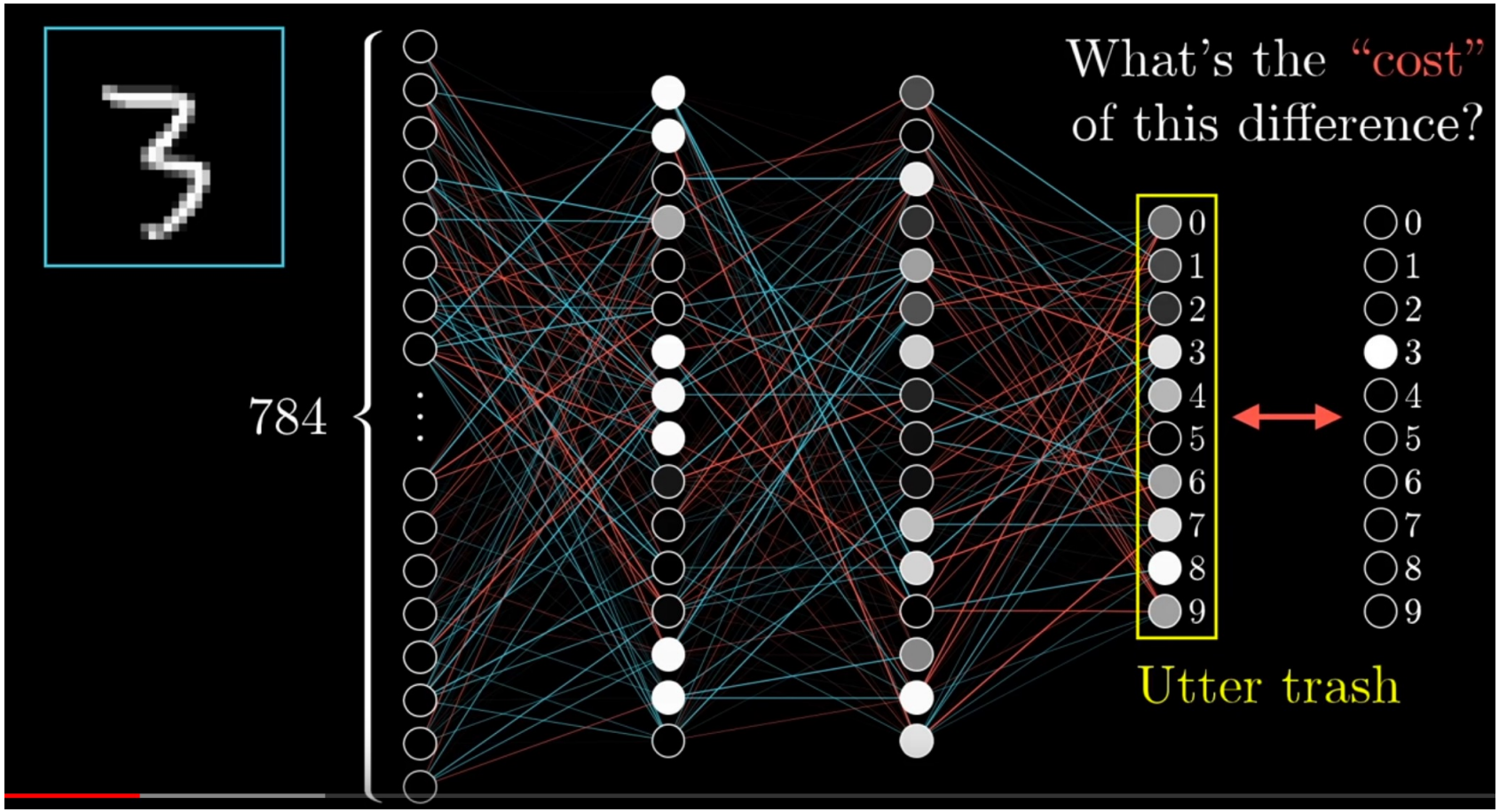
Learning = finding minimum of certain function



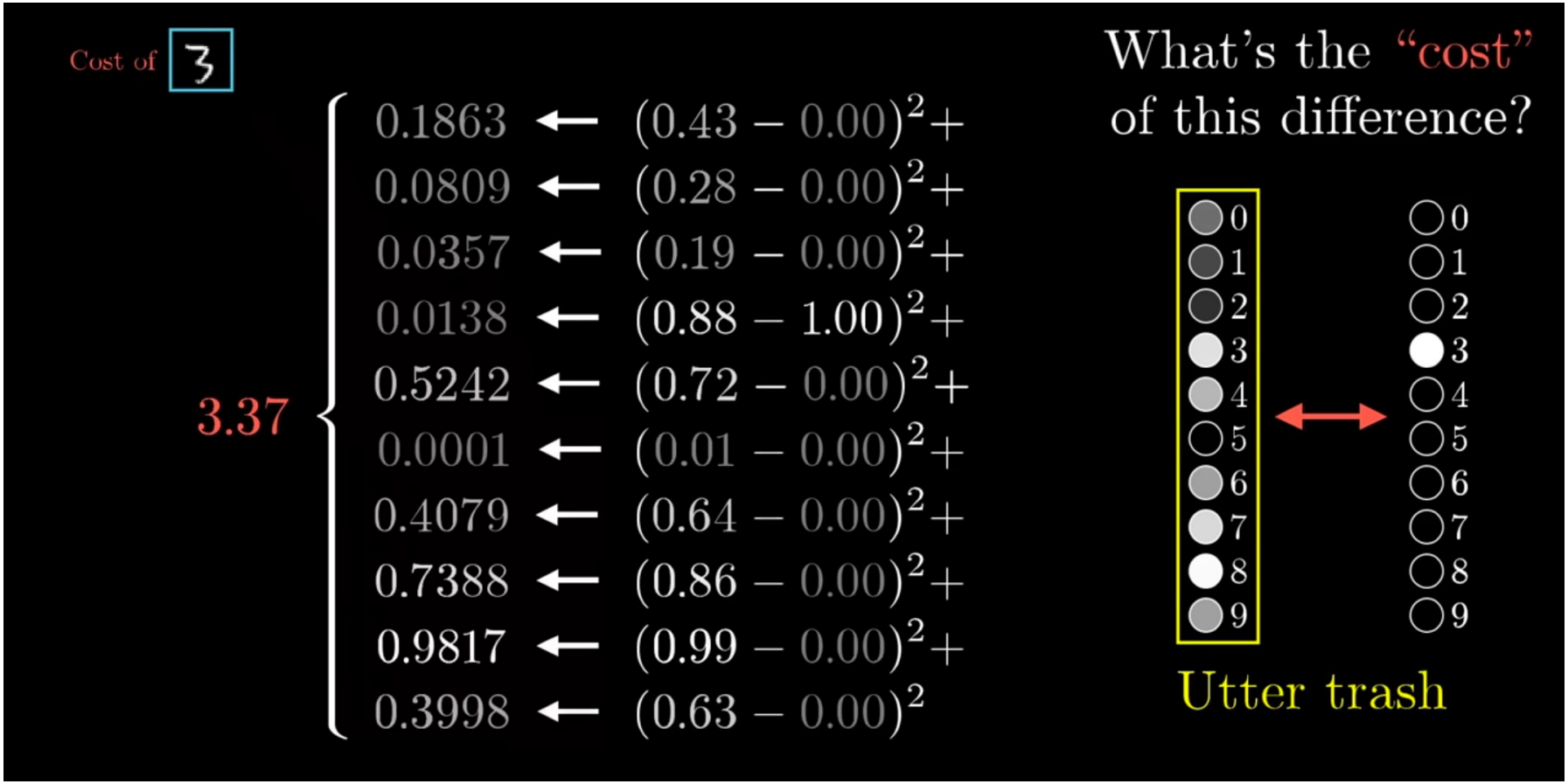
Weight = strength of connection
Bias = is neuron active or not



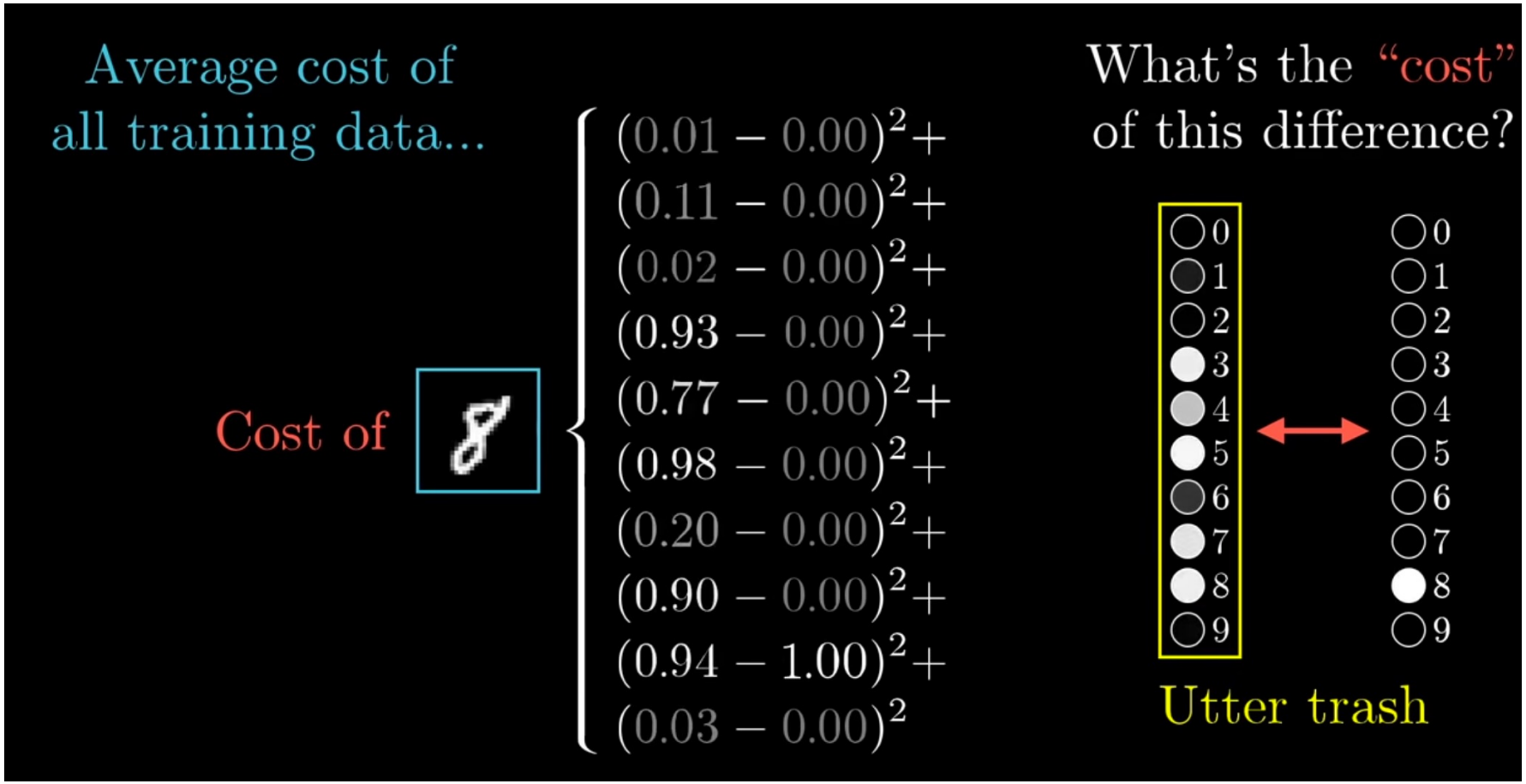
Random initialization of weights and biases



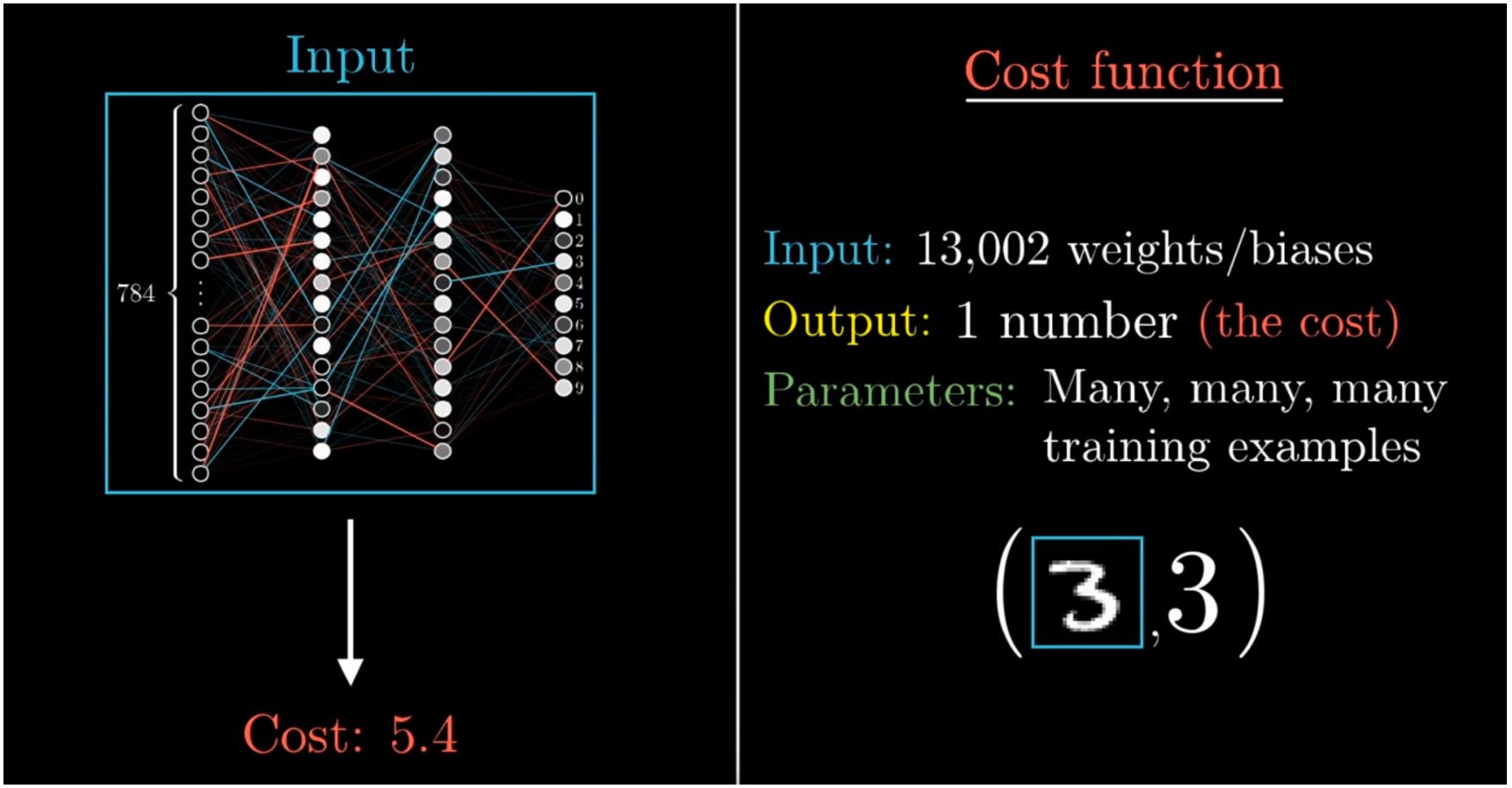
Calculate difference (cost) between actual value and expected value



Calculate average costs of all training examples



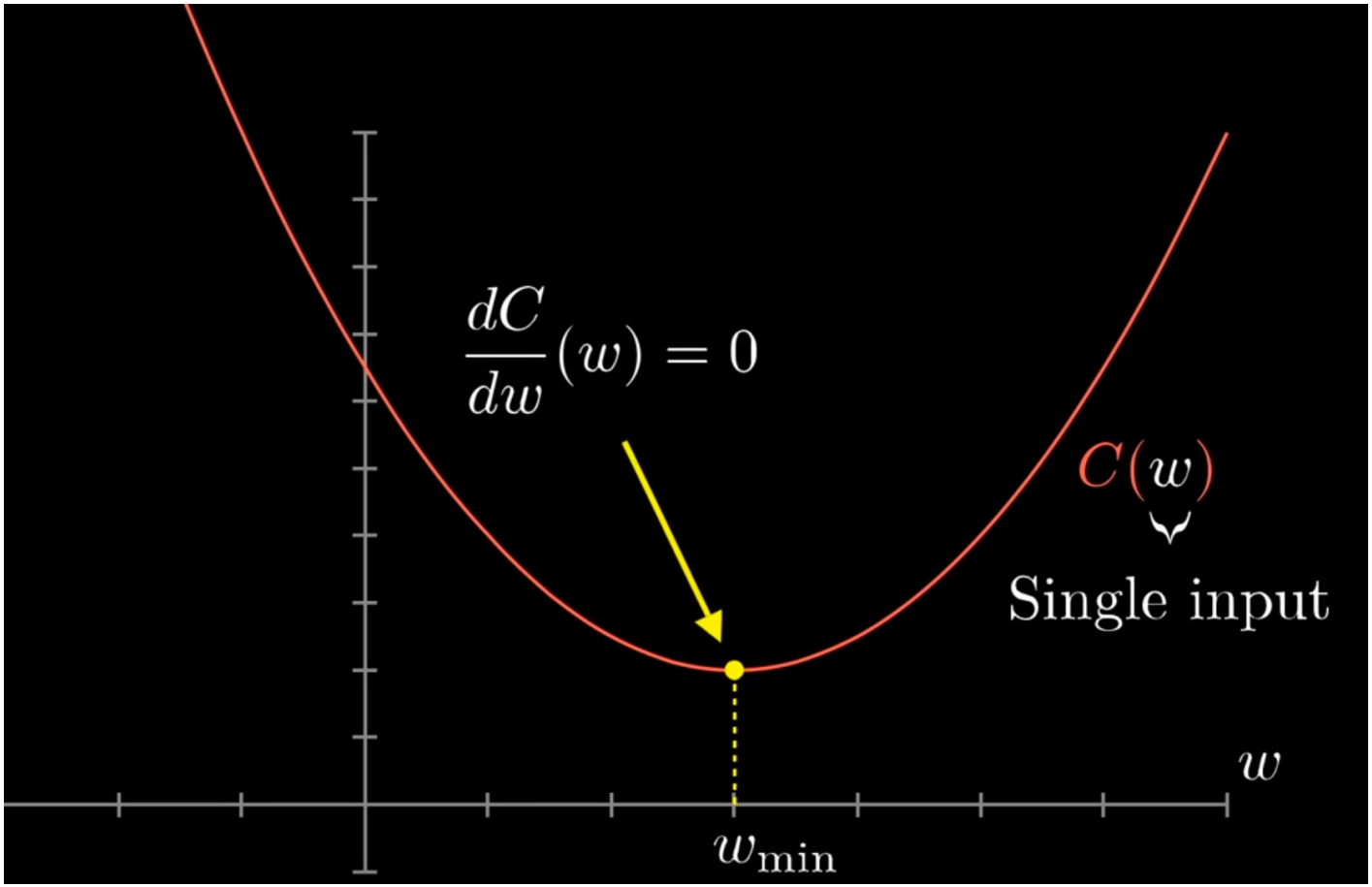
Cost function



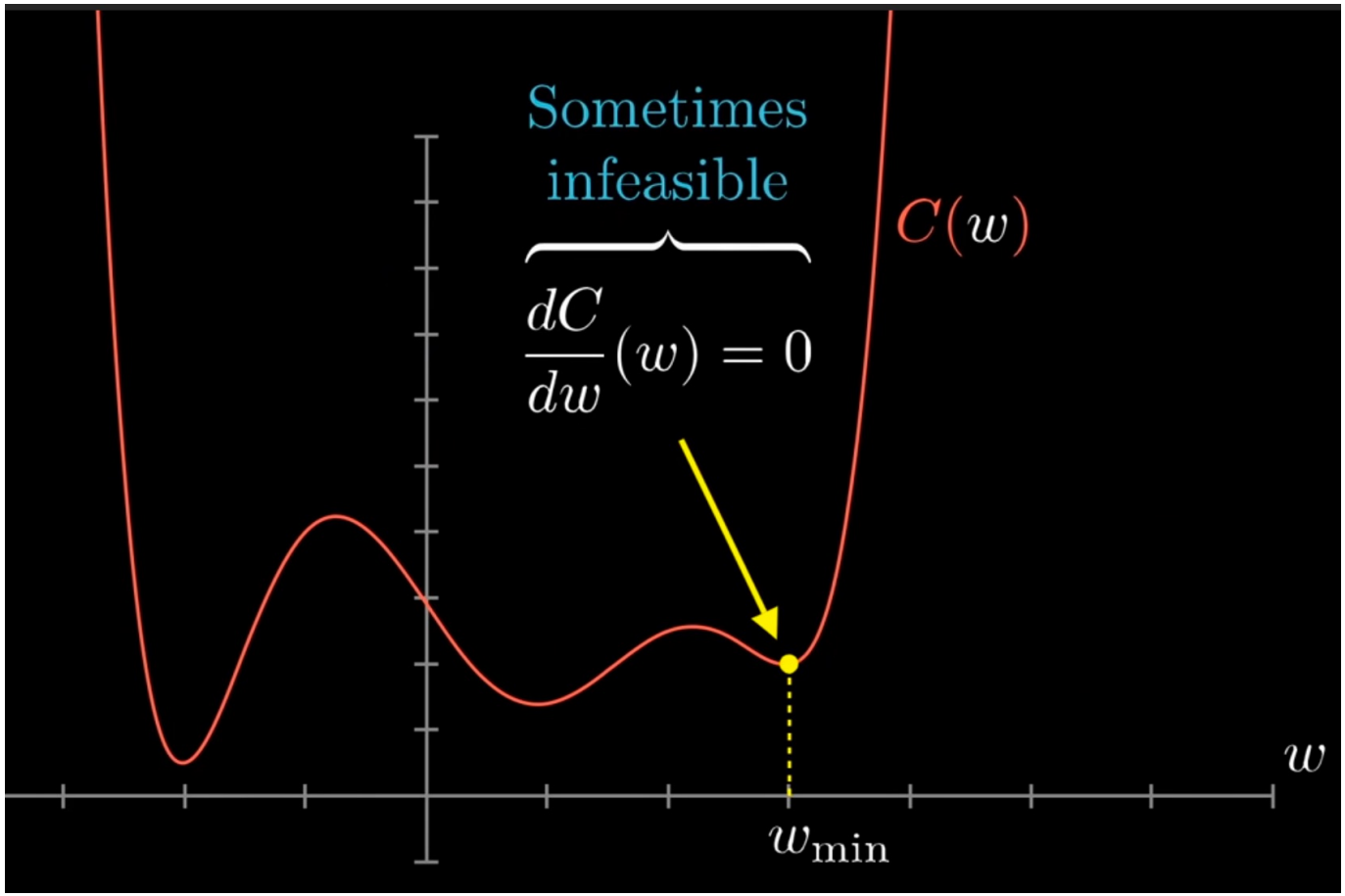
Learning

- How to change weights and biases such that cost is minimized?

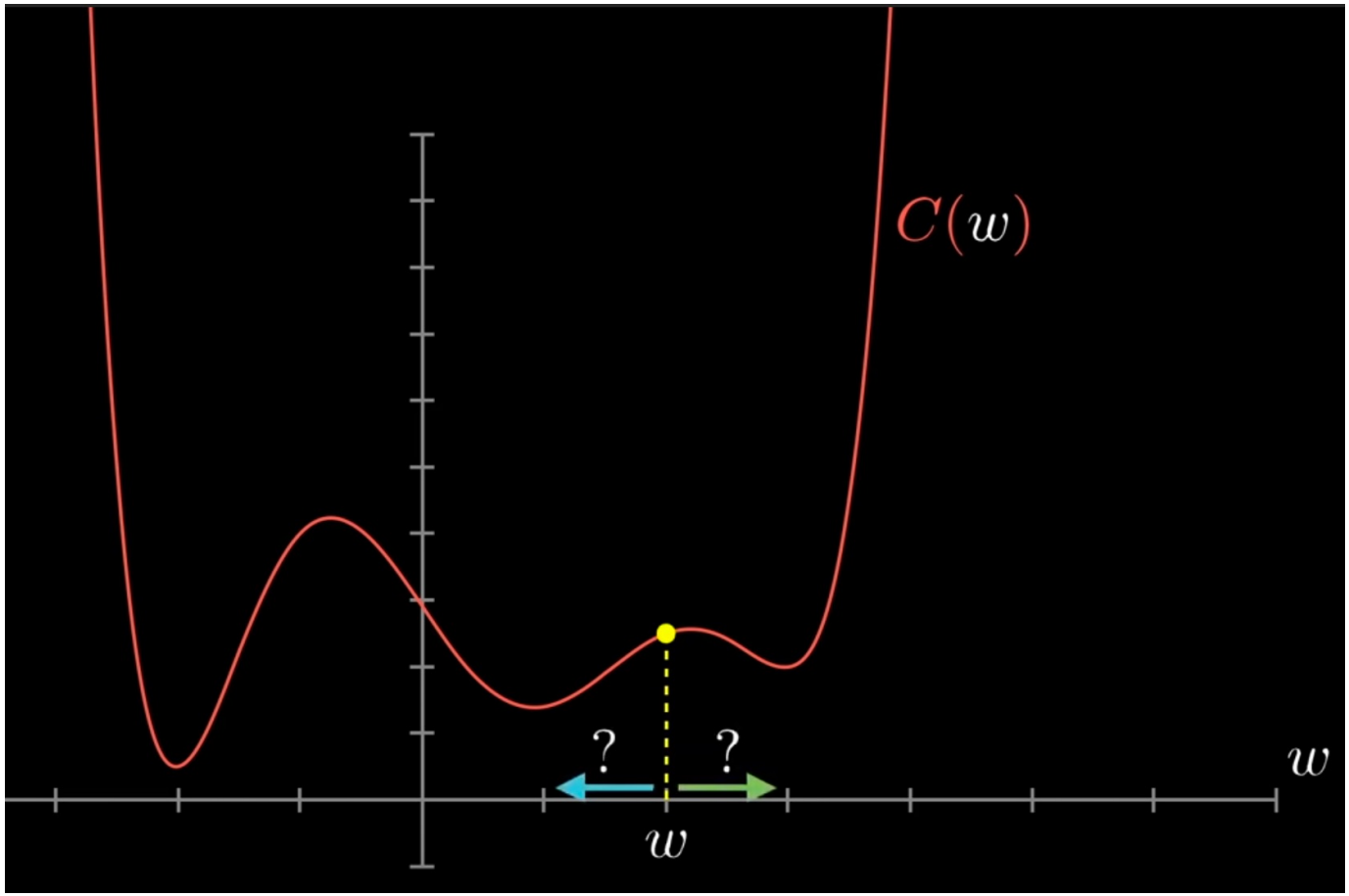
Calculating the Minimum



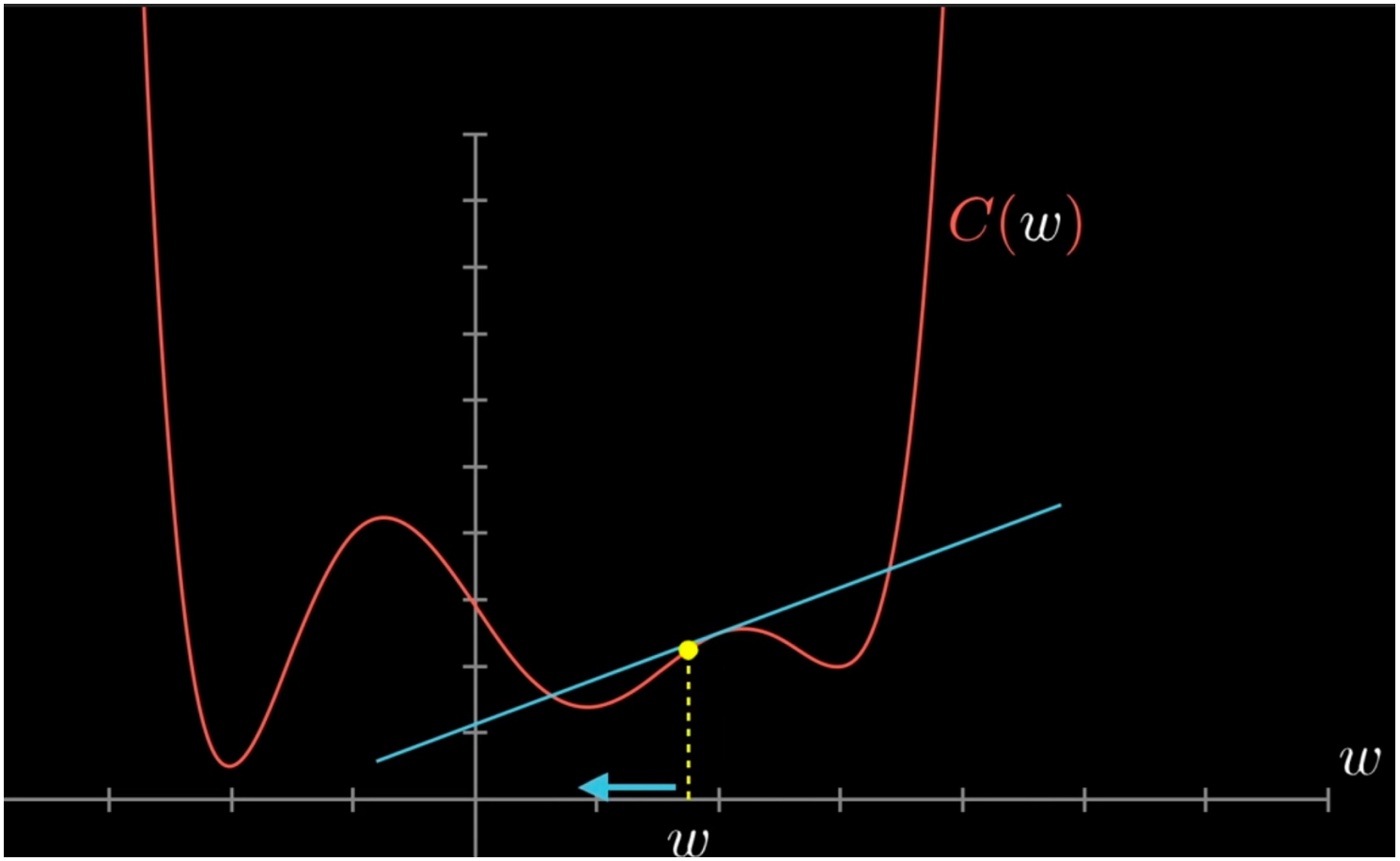
Calculating the minimum is a bit more complex now for this function



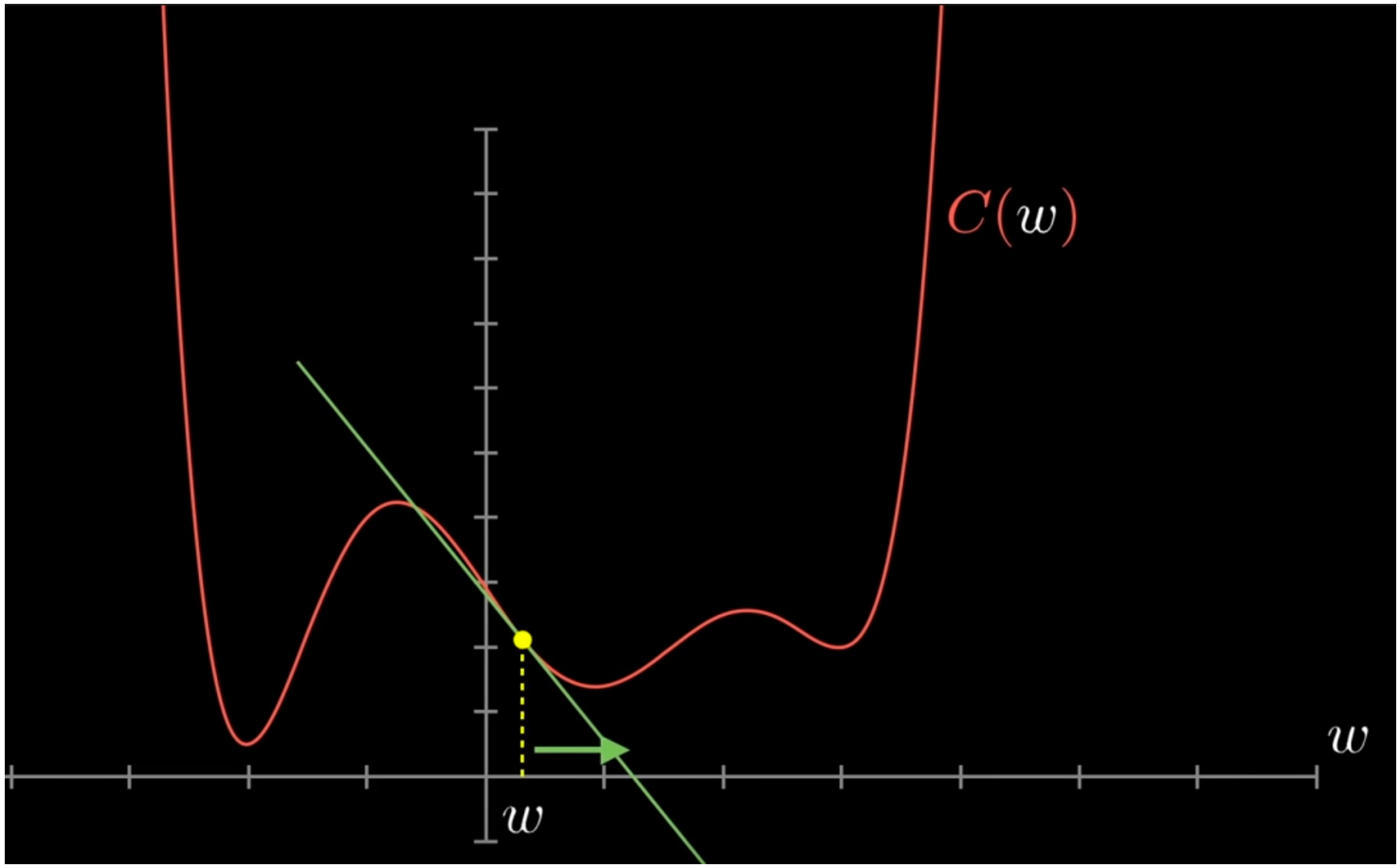
Alternative: Start at a specific point
Check in which direction to go to make output lower



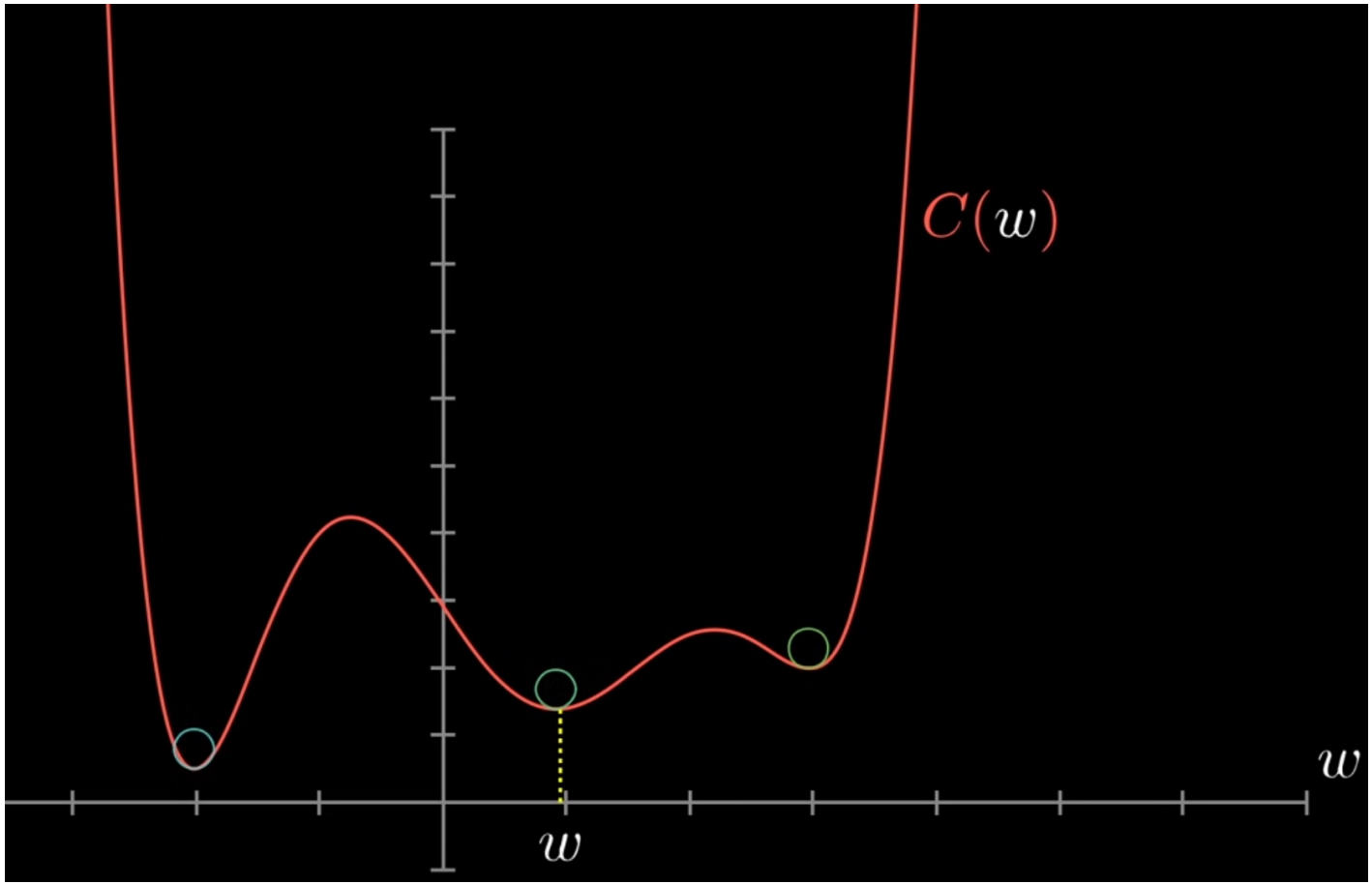
Slope tells you in which direction to go: If positive, go left



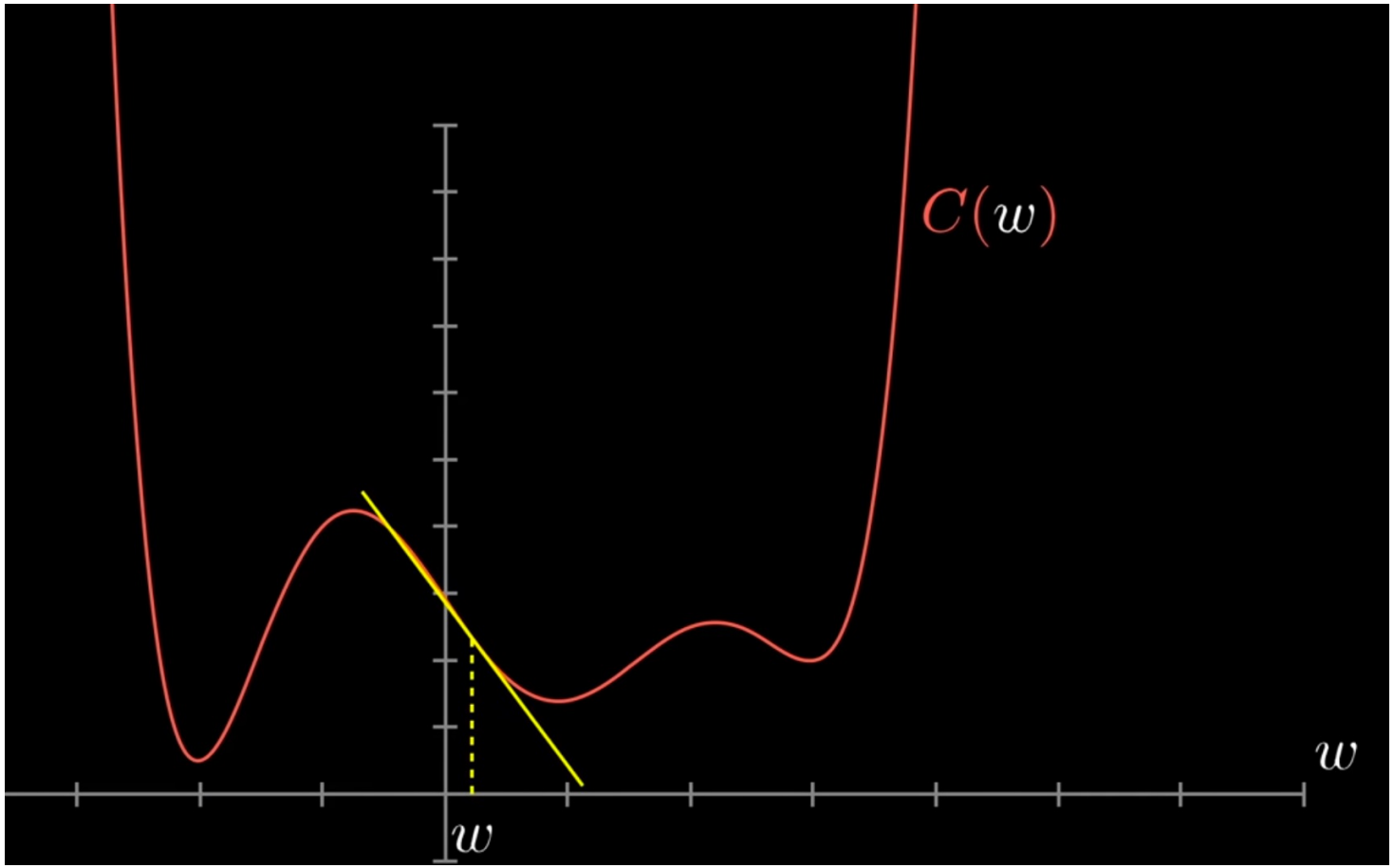
Slope tells you in which direction to go: If negative, go right



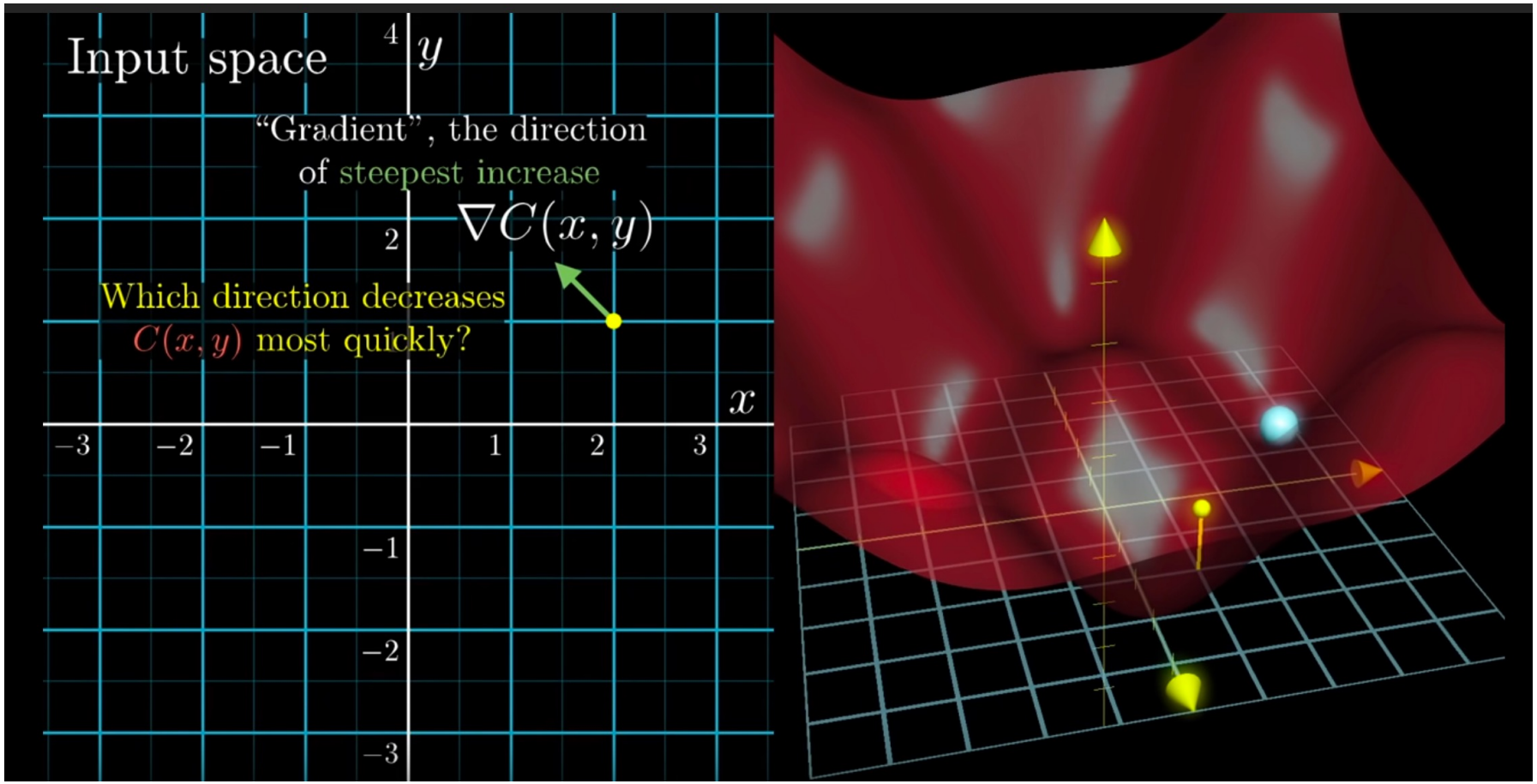
Various (local) minima possible: No guarantee of finding global minimum



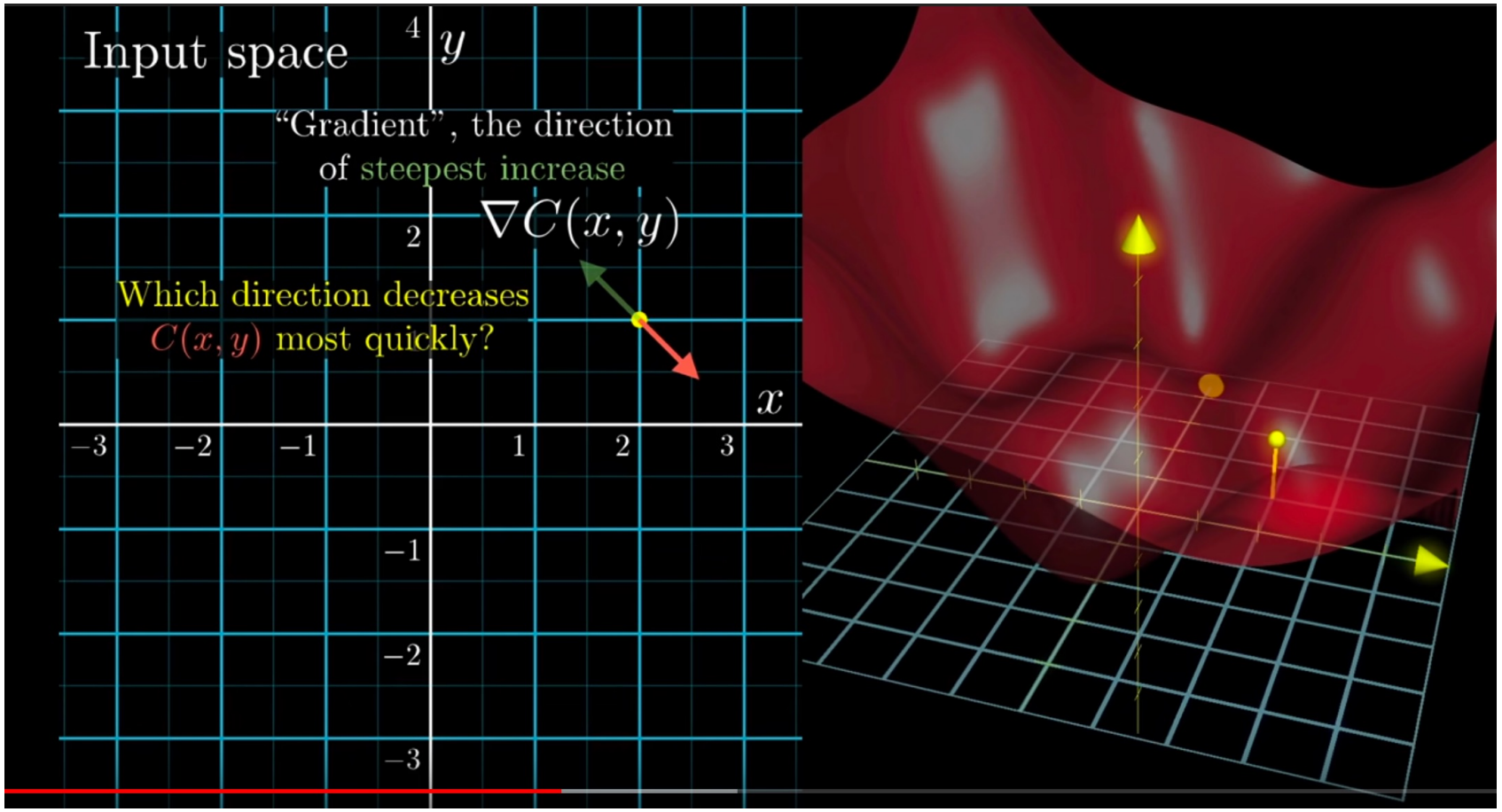
Make step size proportional to slope → Avoids “overshooting”



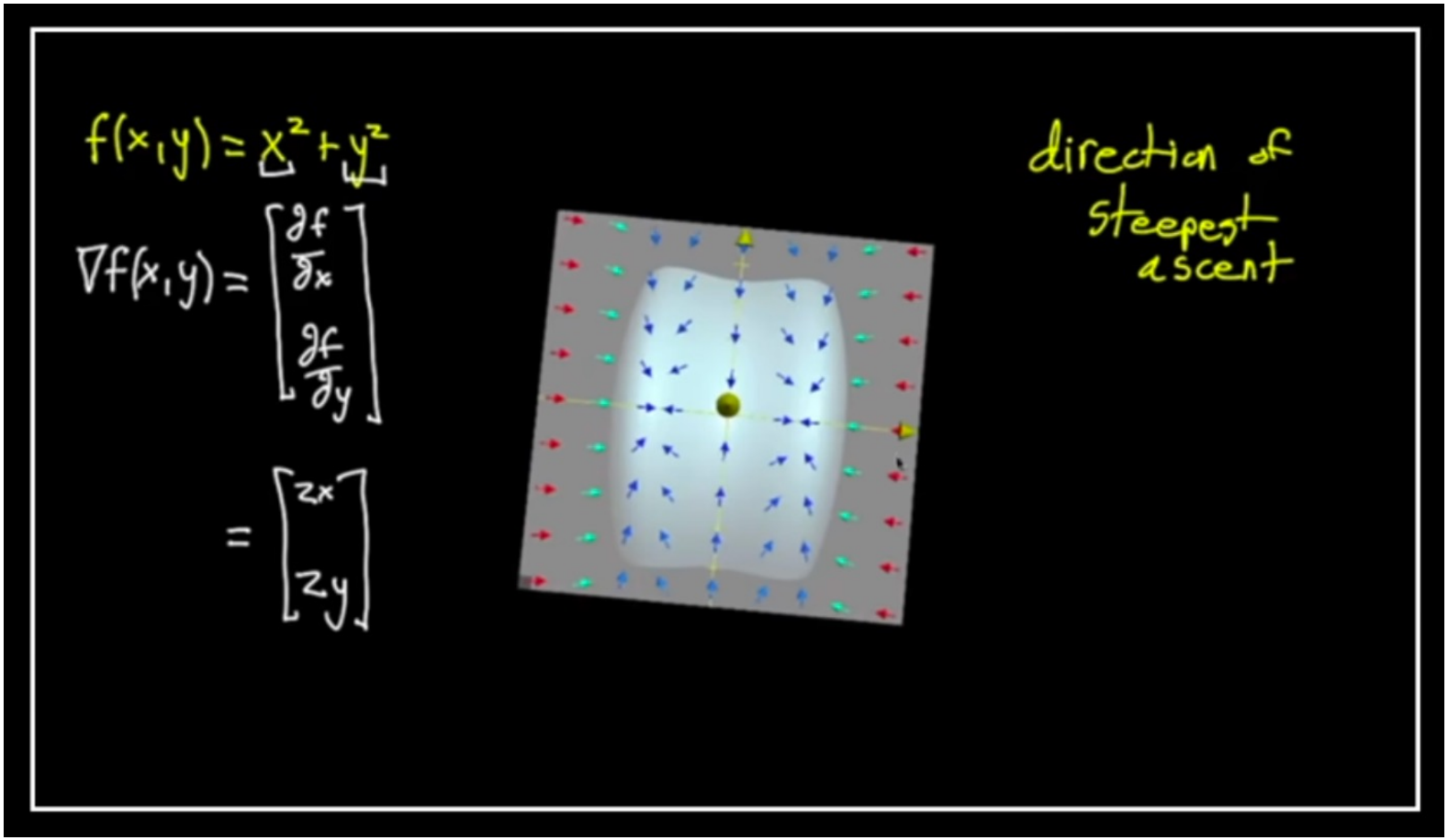
Gradient of function gives direction of steepest ascent



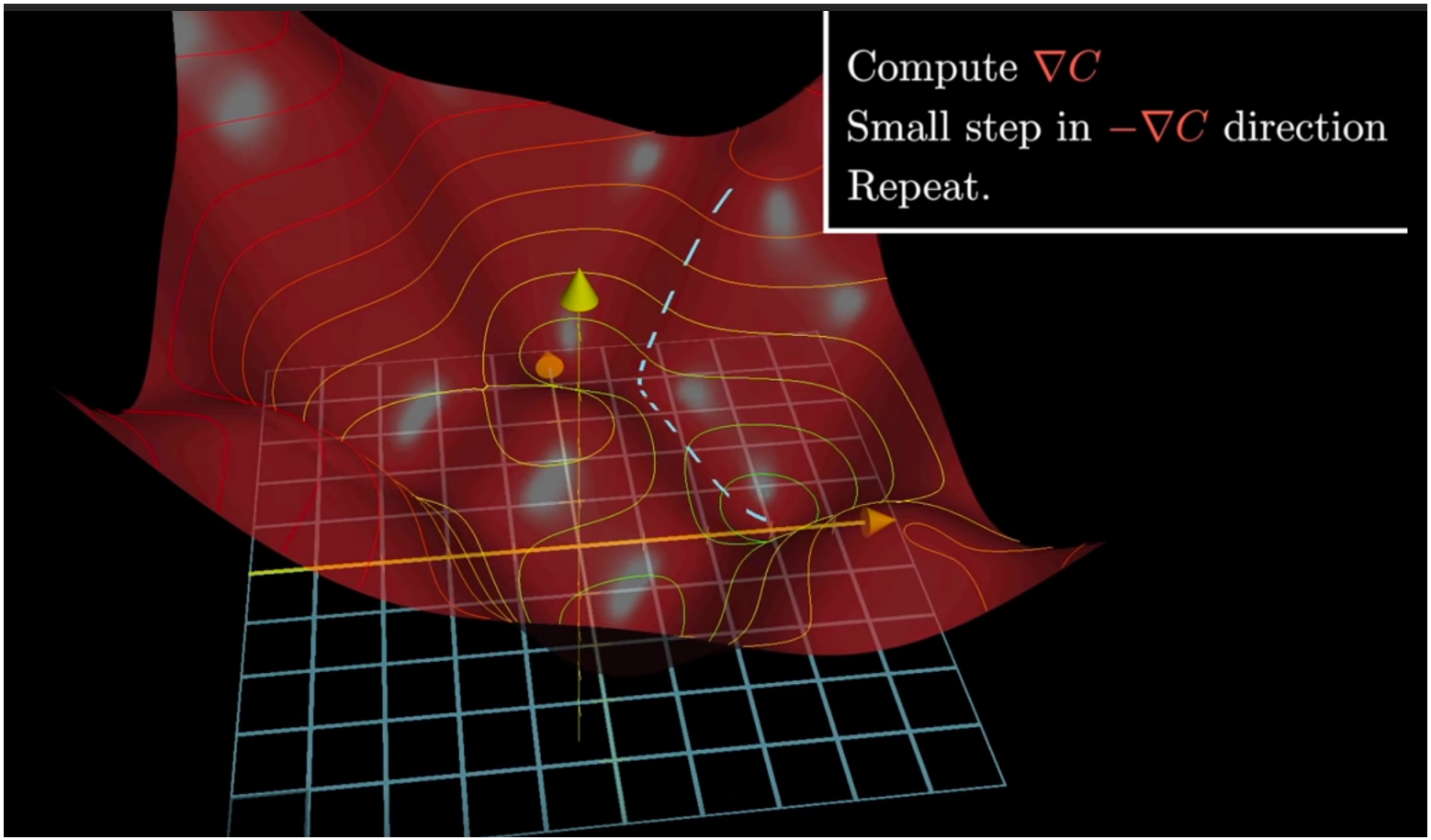
Length of gradient vector = steepness of slope



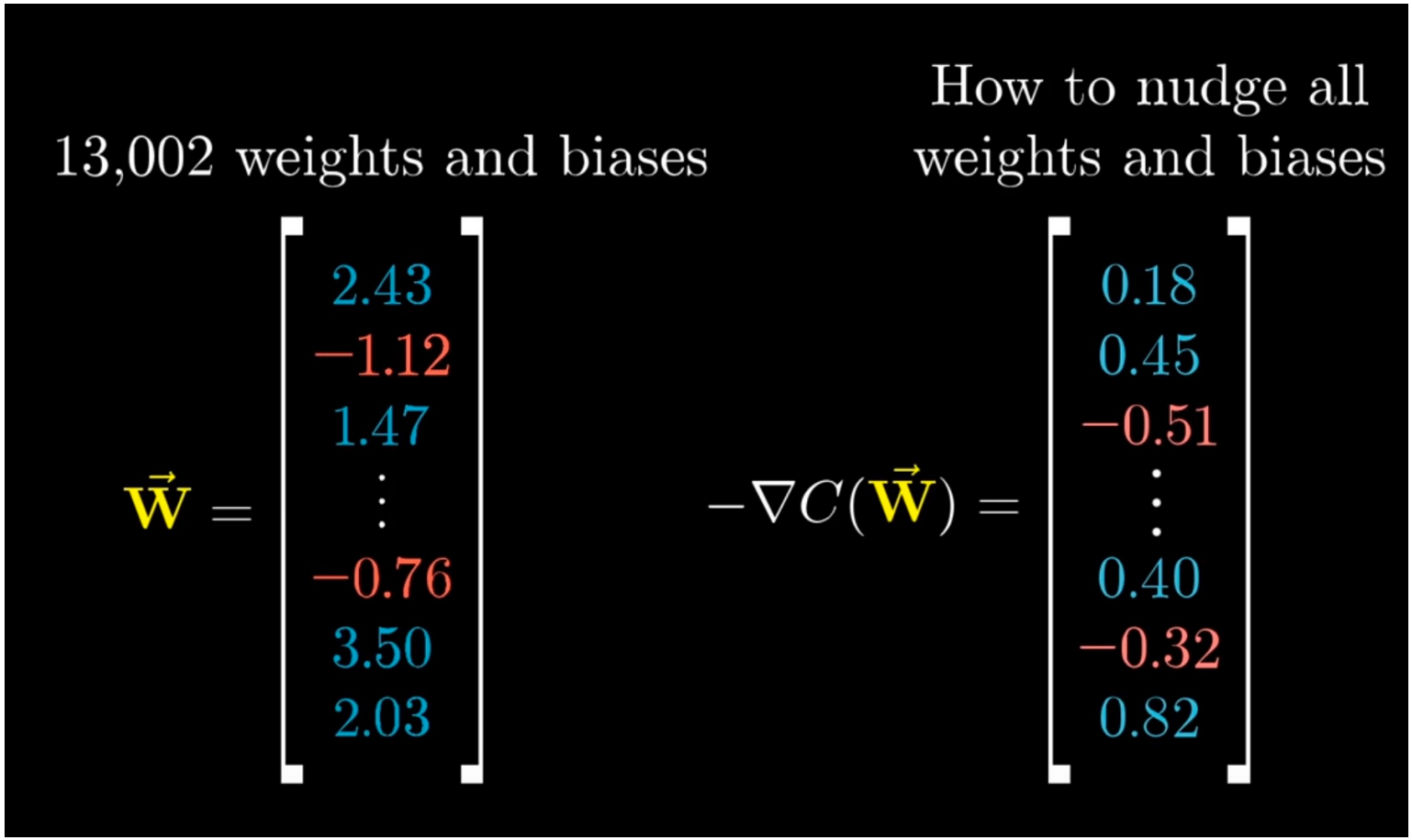
Direction of steepest descent can be calculated
But: don't worry about details



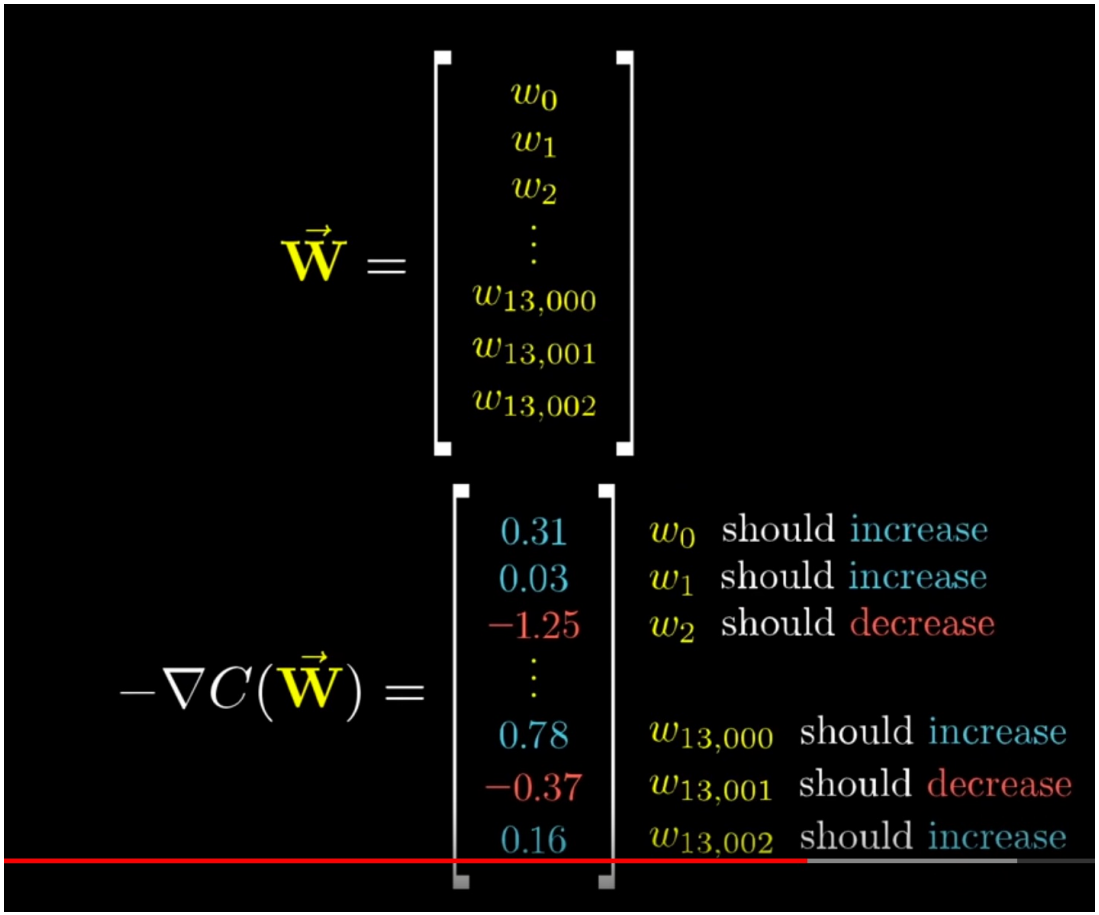
Gradient descent algorithm:
(1) compute cost (2) take small step (3) repeat



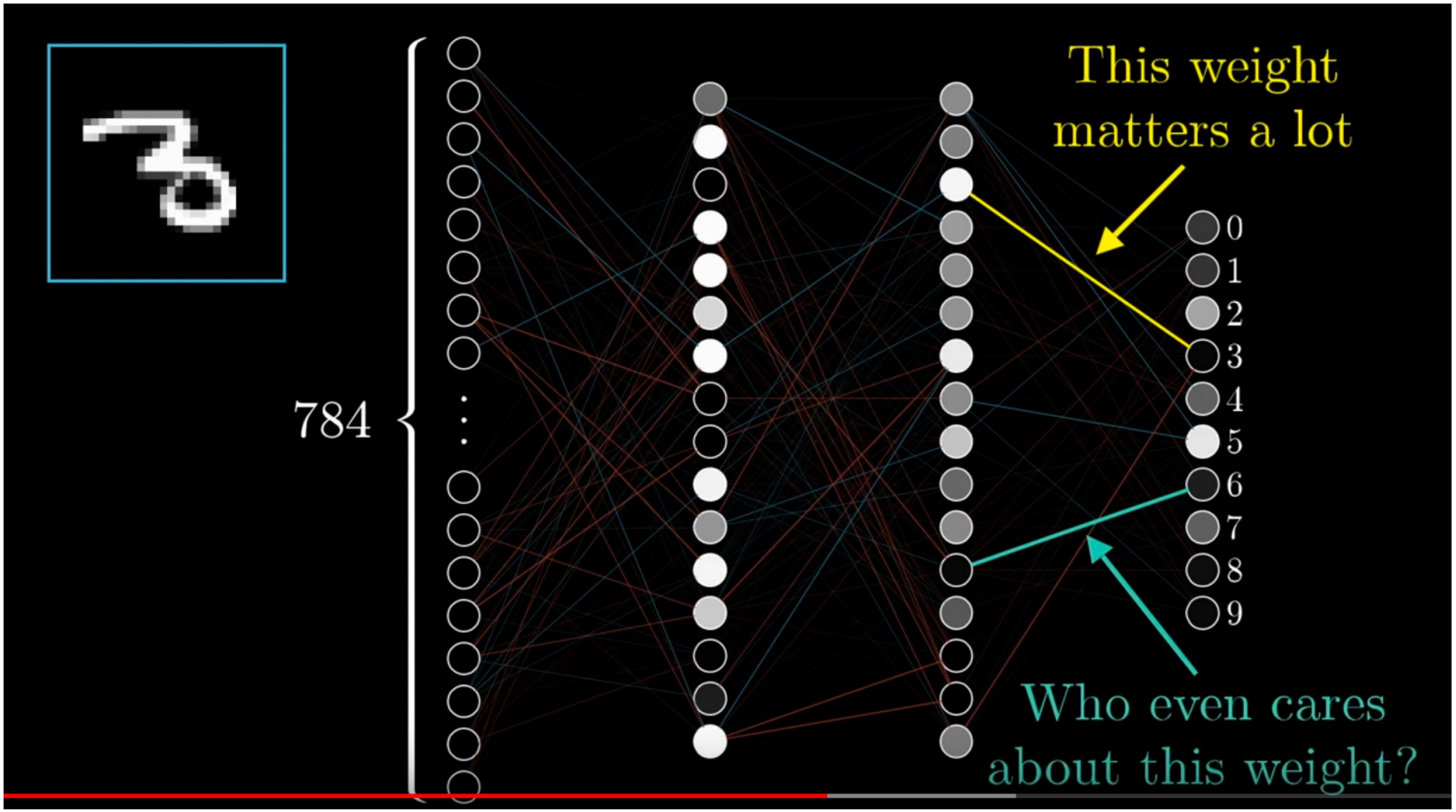
Negative gradient of cost function = vector of weights



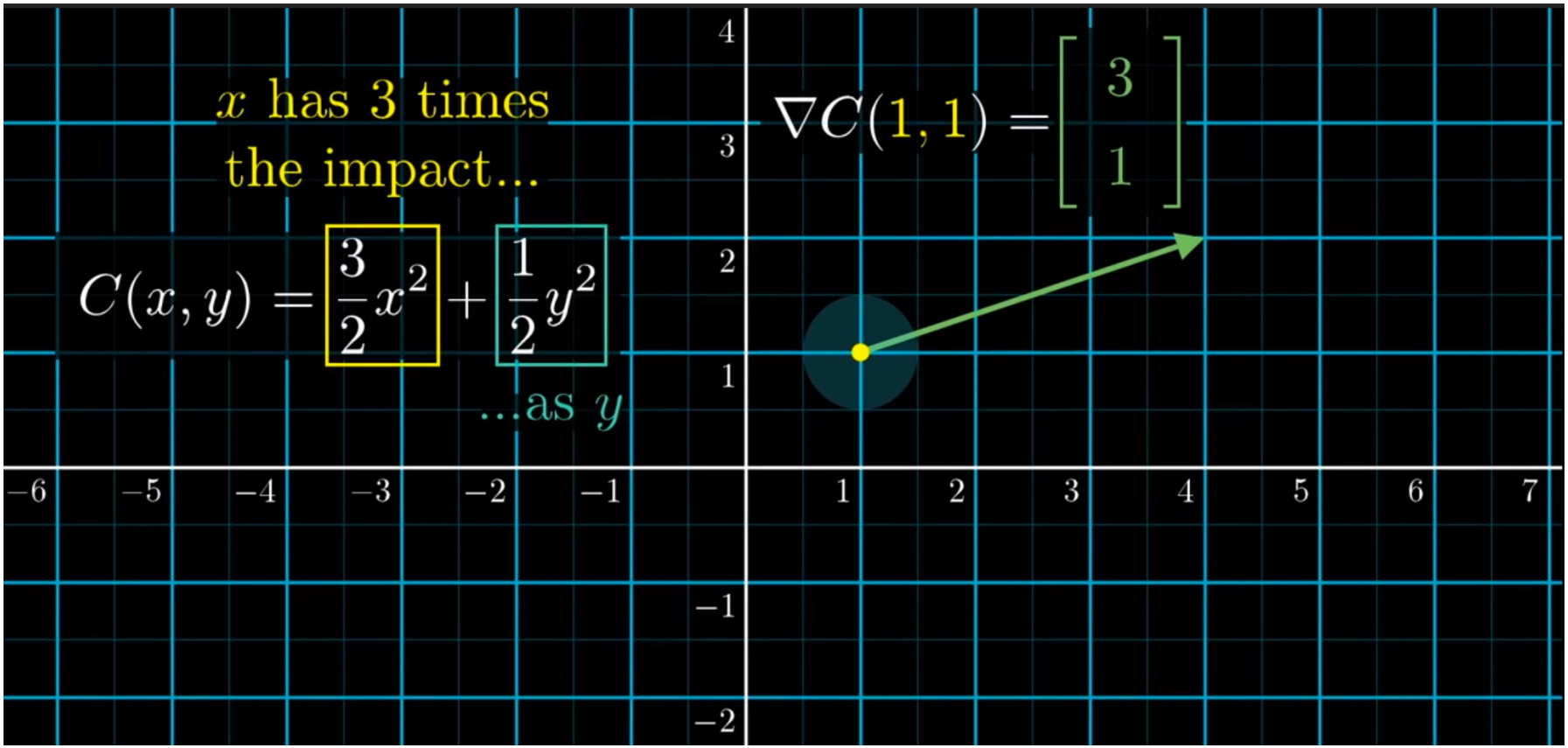
Sign of gradient says if weight should be pushed up or down



Higher weights make some neurons “more important”



x value has 3 times more importance than y value



Summary

- Neural network consists of input, (hidden) and output layers
- The goal of neural networks is to minimize a cost function:
  - Error between expected and observed value
- Neural networks have a large parameter space of weights and biases
  - Need to find a good parameter setting
- Gradient descent helps you find the (local) minimum
  - The slope of the gradient tells you in which direction to go
  - Step size is proportional to the size of the gradient