Deep Learning Basics

2: Network Structure

Francis Steen and Xinyu You
Red Hen Lab
August 2019

A. Perceptron

The building blocks of neural networks

Logic-gate neurons

- The origins of Neural Networks goes back to 1943, when Warren McCulloch, a neurophysiologist, and a young mathematician, Walter Pitts, wrote a paper on how neurons might work

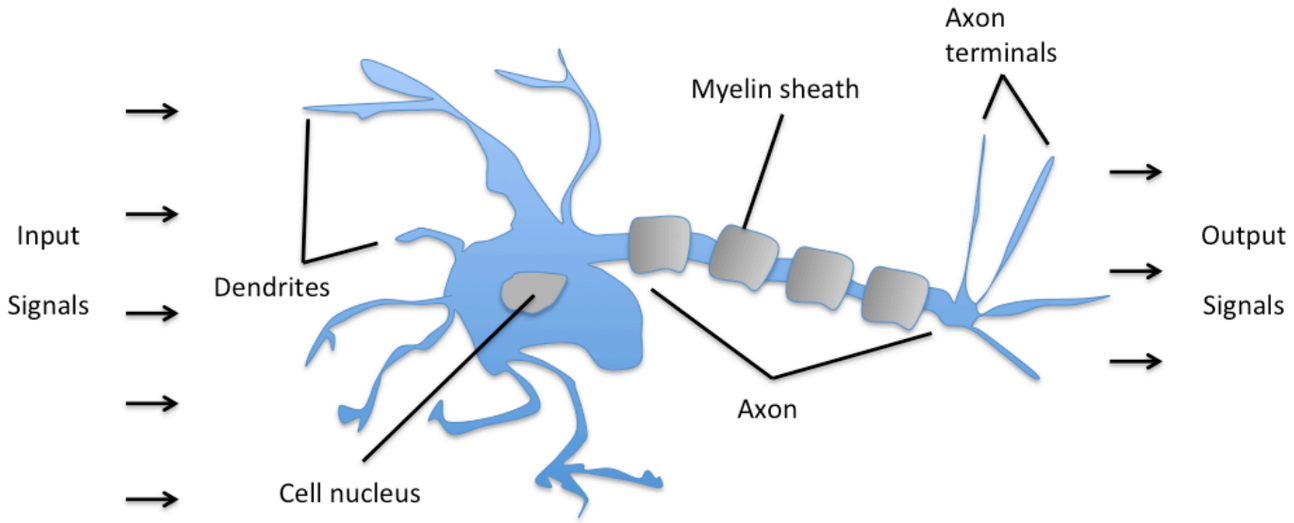
- They proposed that neurons are binary logic gates that receive multiple inputs through their dendrites and convert these to a single output

Classical computationalism

- In 1958, the neurobiologist Frank Rosenblatt extended this idea to form an artificial neuron, which he called a Perceptron

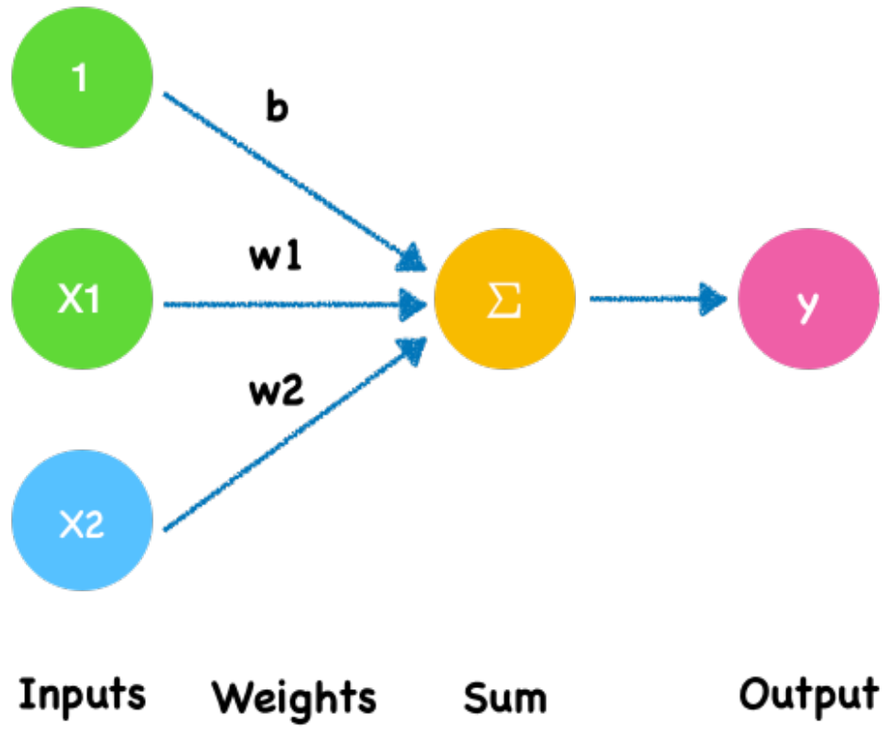
- It aimed to replicate the function of a neuron, and make it possible to process sensed information, for instance from a camera

Neuron-like perceptron



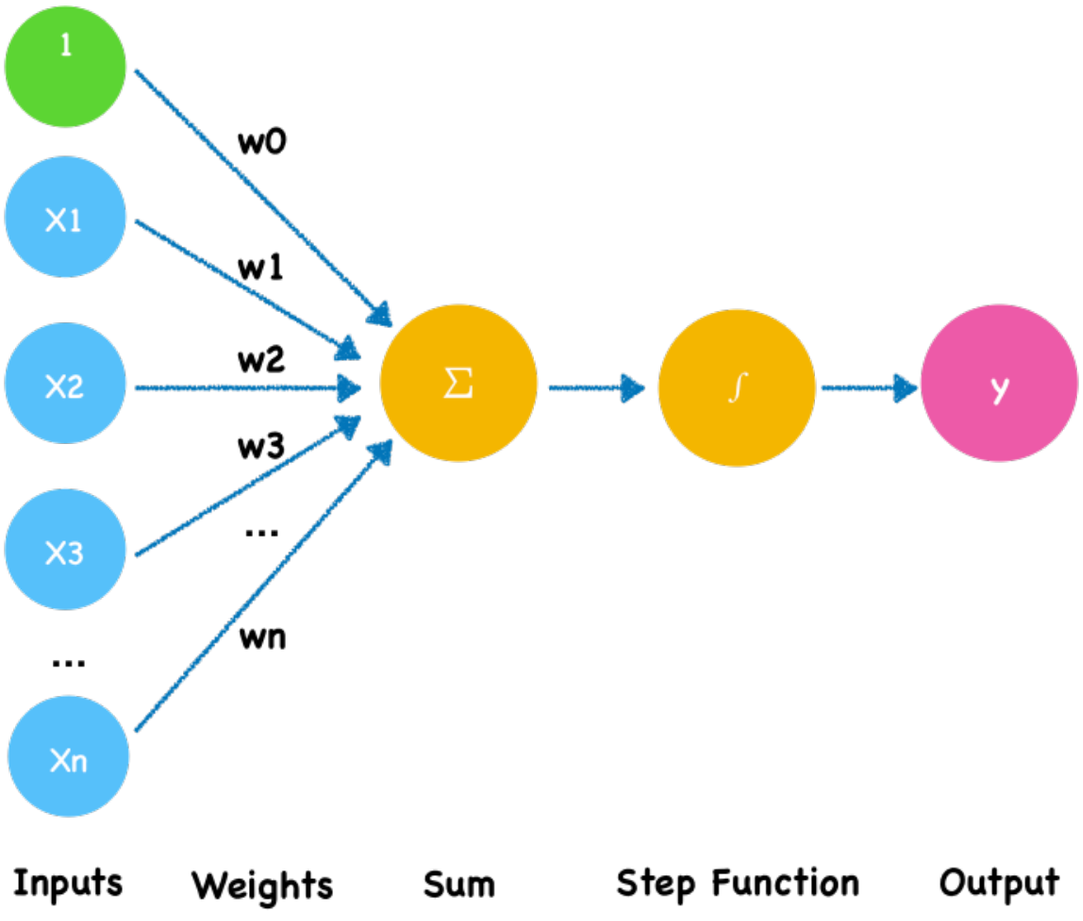
Schematic of a biological neuron.

Neuron



Perceptron

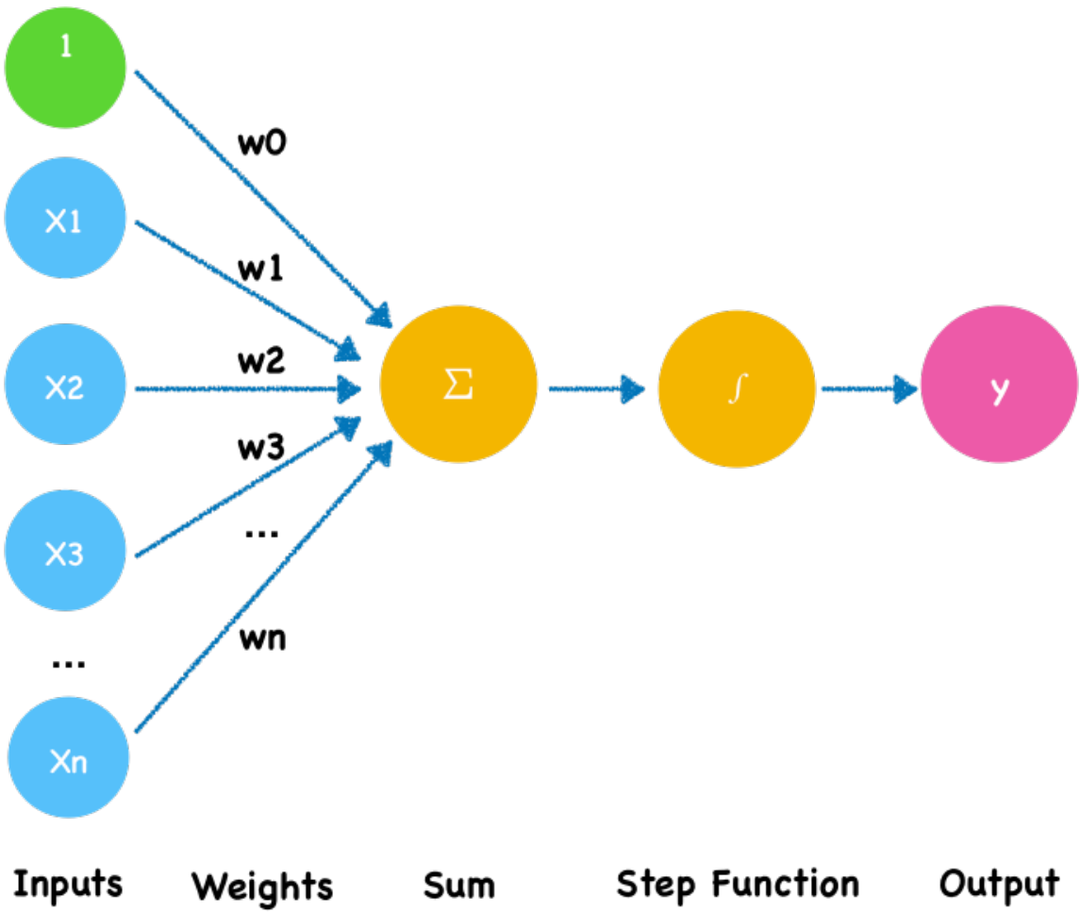
Binary perceptron



- Electrical signals -> Input
- Electrical signals amounts-> weights
- Threshold -> Step Function
- W0 is bias.

$$output = \begin{cases} 0 \sum_{i=0}^{n} w_i x_i \leq threshold \\ 1 \sum_{i=0}^{n} w_i x_i > threshold \end{cases}$$

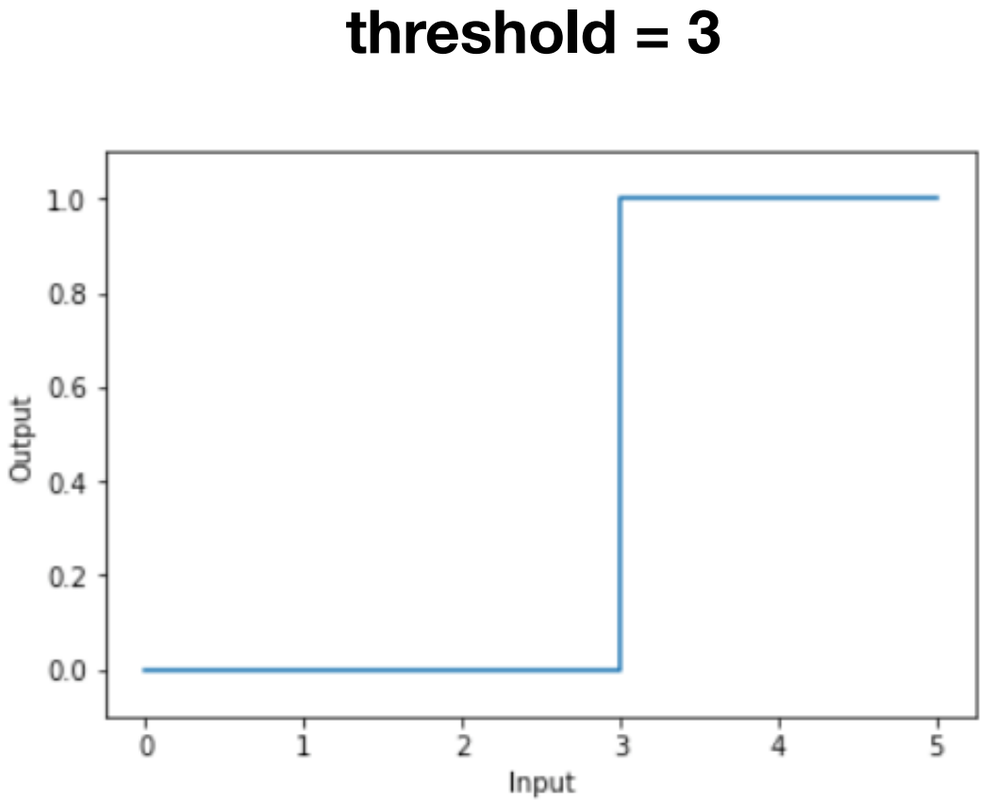
Binary perceptron



$$output = \begin{cases} 0 \sum_{i=0}^{n} w_i x_i \leq threshold \\ 1 \sum_{i=0}^{n} w_i x_i > threshold \end{cases}$$

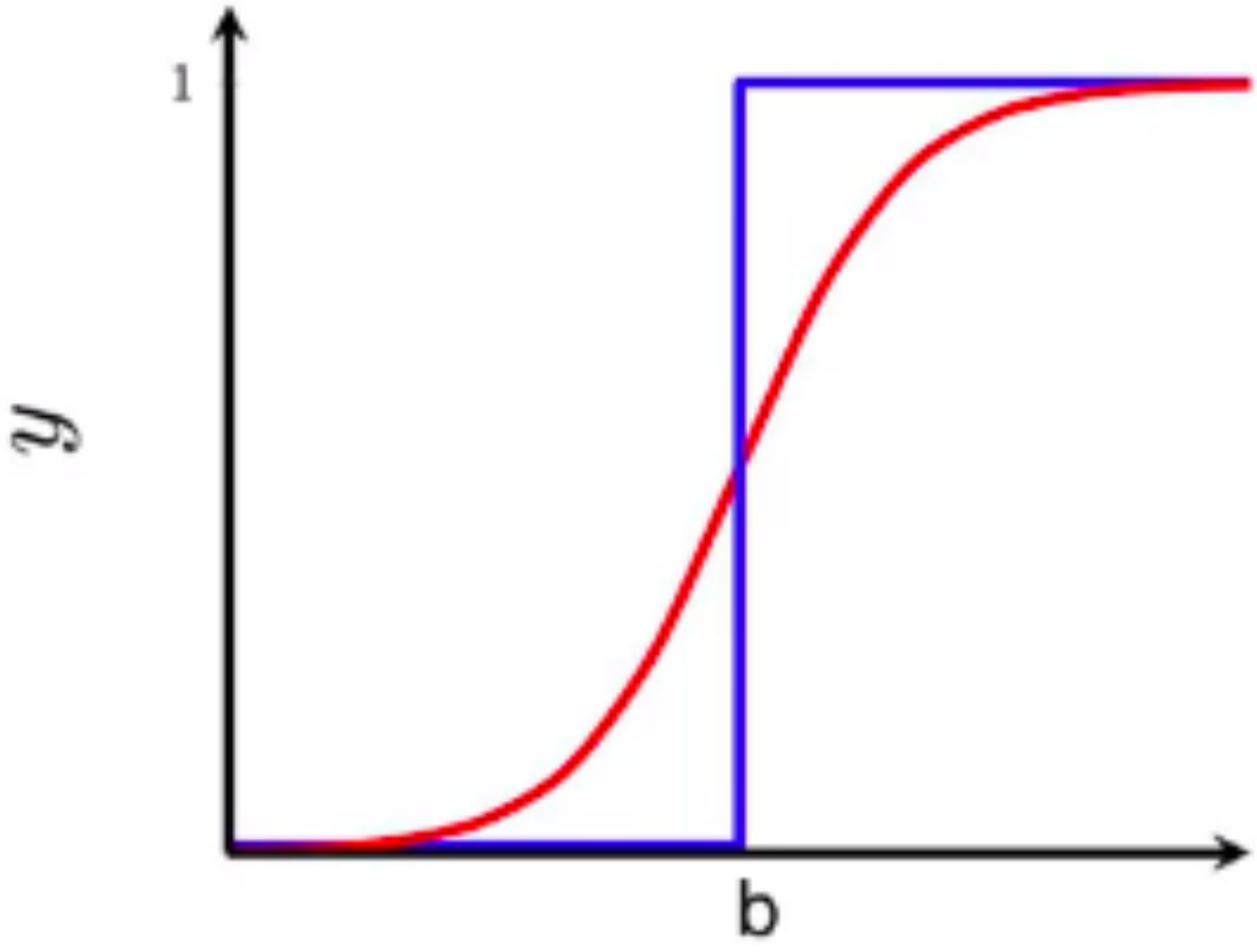
- Here W0 is a bias.
- It’s a parameter that adjusts how easily a neuron is activated.
- It's like wearing flat or high-heeled shoes to achieve a certain height.
- For example, you want to look like 1.8m in height. You can take calcium or doing stretch. What bias does here is like shoes. To reach the goal, wearing high heels is significantly easier than wearing flat shoes.

Step Function



$$output = \begin{cases} 0 \sum_{i=0}^{n} w_i x_i \leq threshold \\ 1 \sum_{i=0}^{n} w_i x_i > threshold \end{cases}$$

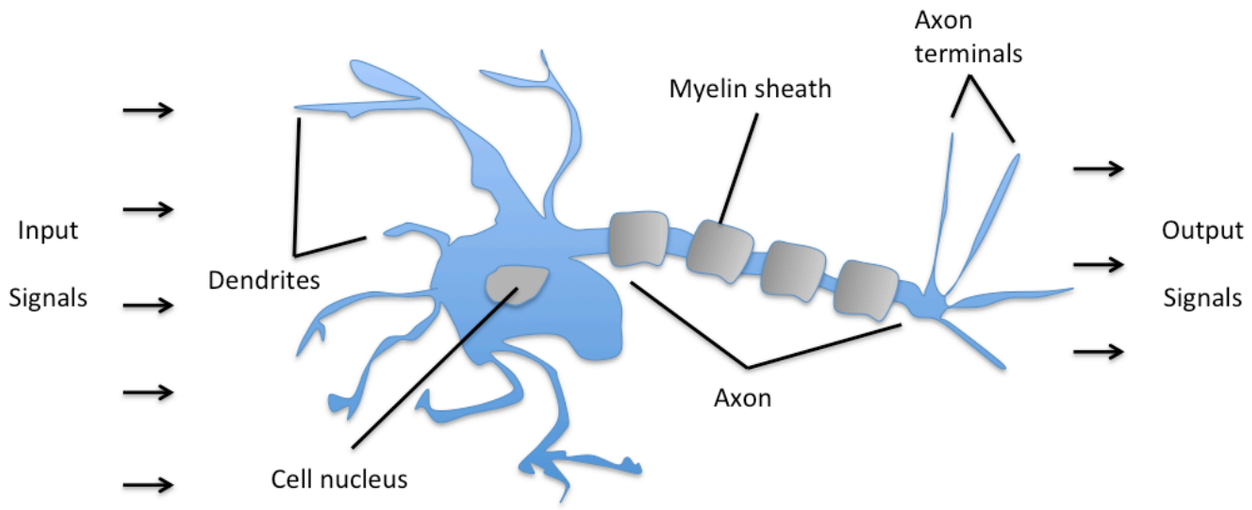
Sigmoid neuron



- $\sum_{i=1}^{n} w_i x_i$

Neurons are more powerful

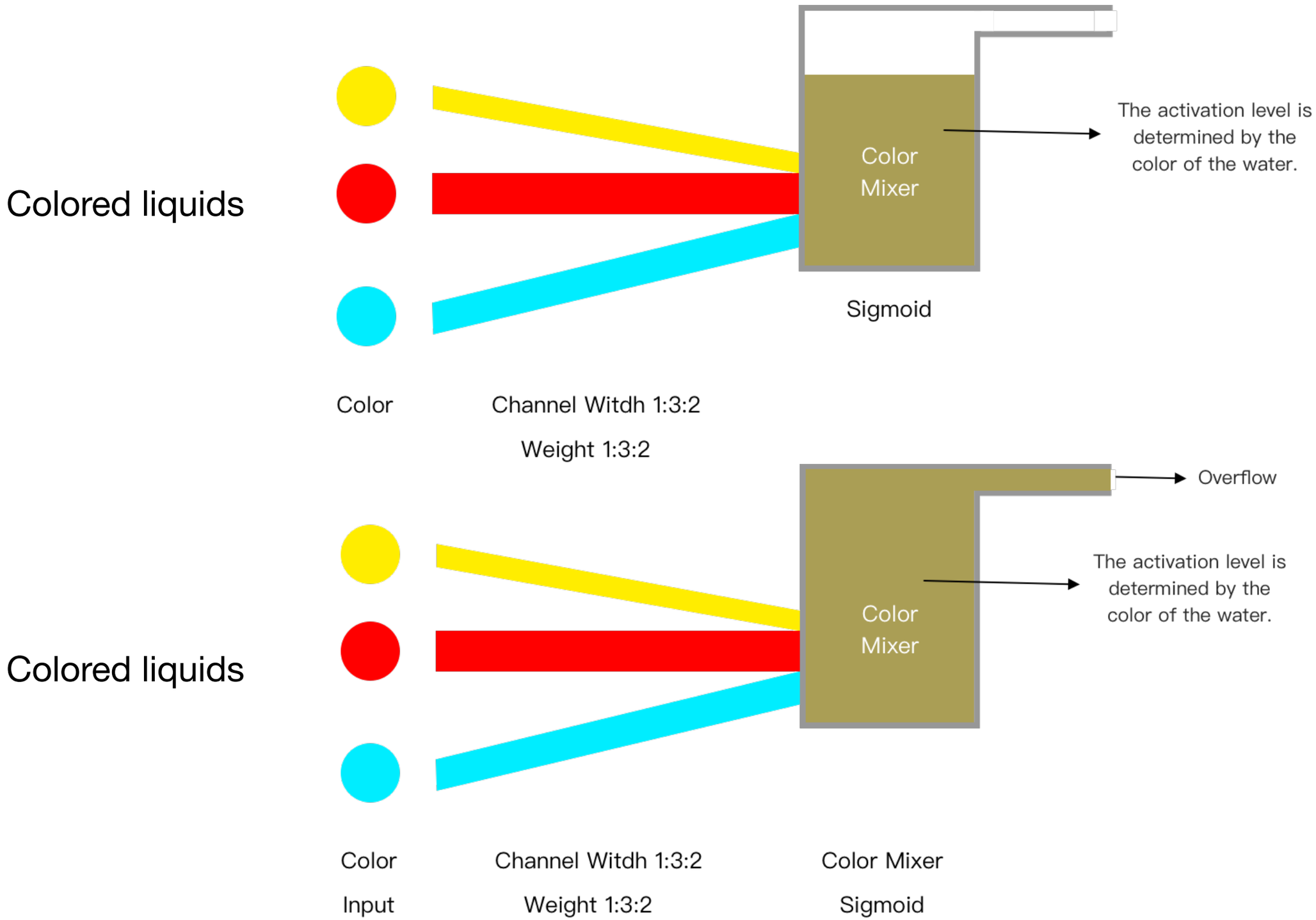
A door to new forms of AI



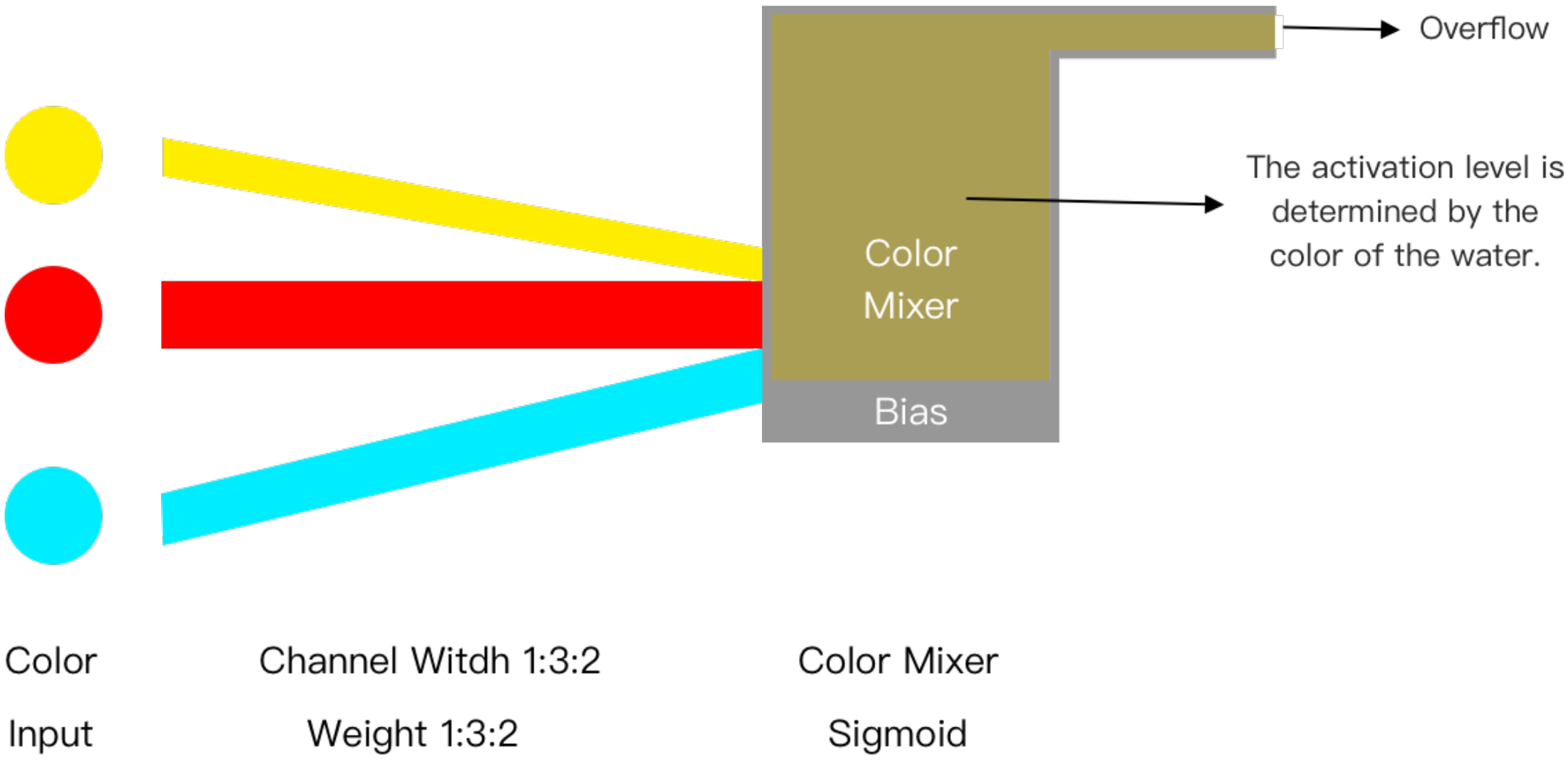
Schematic of a biological neuron.

- Real neurons are far more complex than the perceptron model
- Neuroscience studies suggest a powerful **information processing capacity** within a single neuron, possibly even face recognition
- The ability of single neurons to perform complex tasks indicates that the neuron contains **far more internal structure** than the perceptron model suggests
- A deeper understanding of how neurons process information can lead to new forms of AI

Color Factory



Color Factory

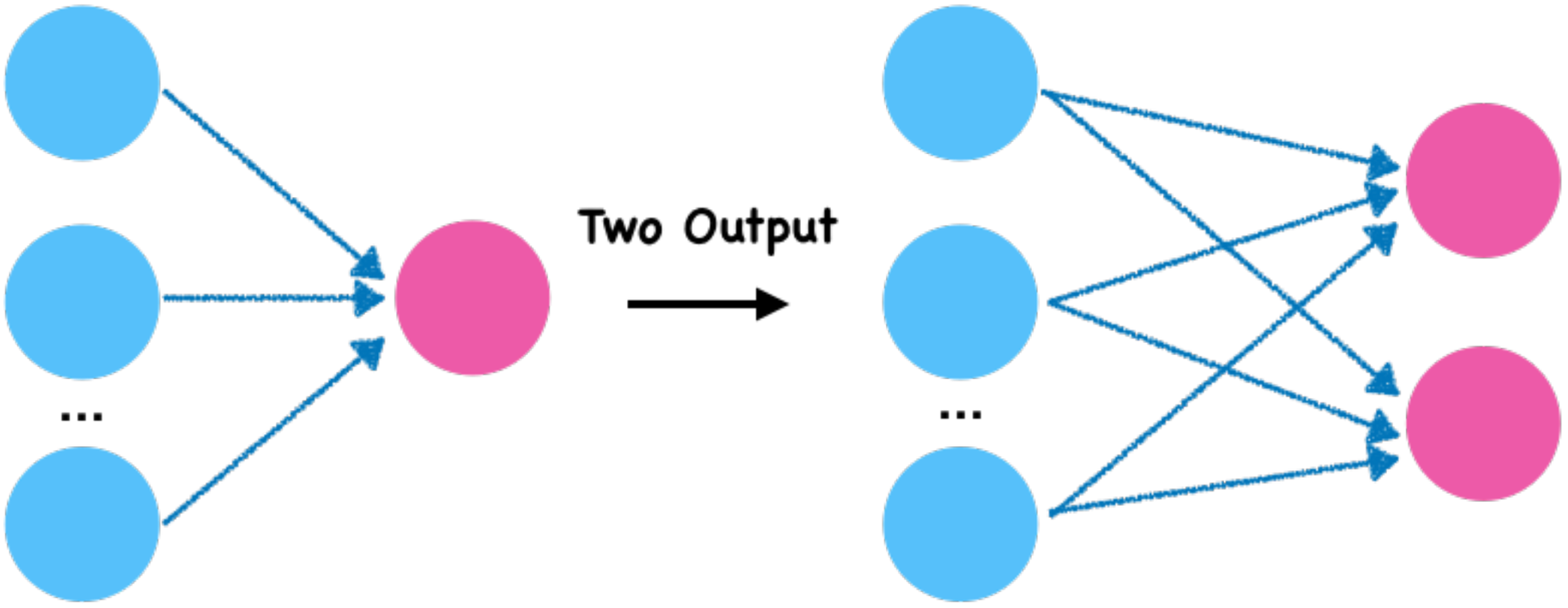


With larger bias, it will be easier to overflow the mixer.

B. Neural Network

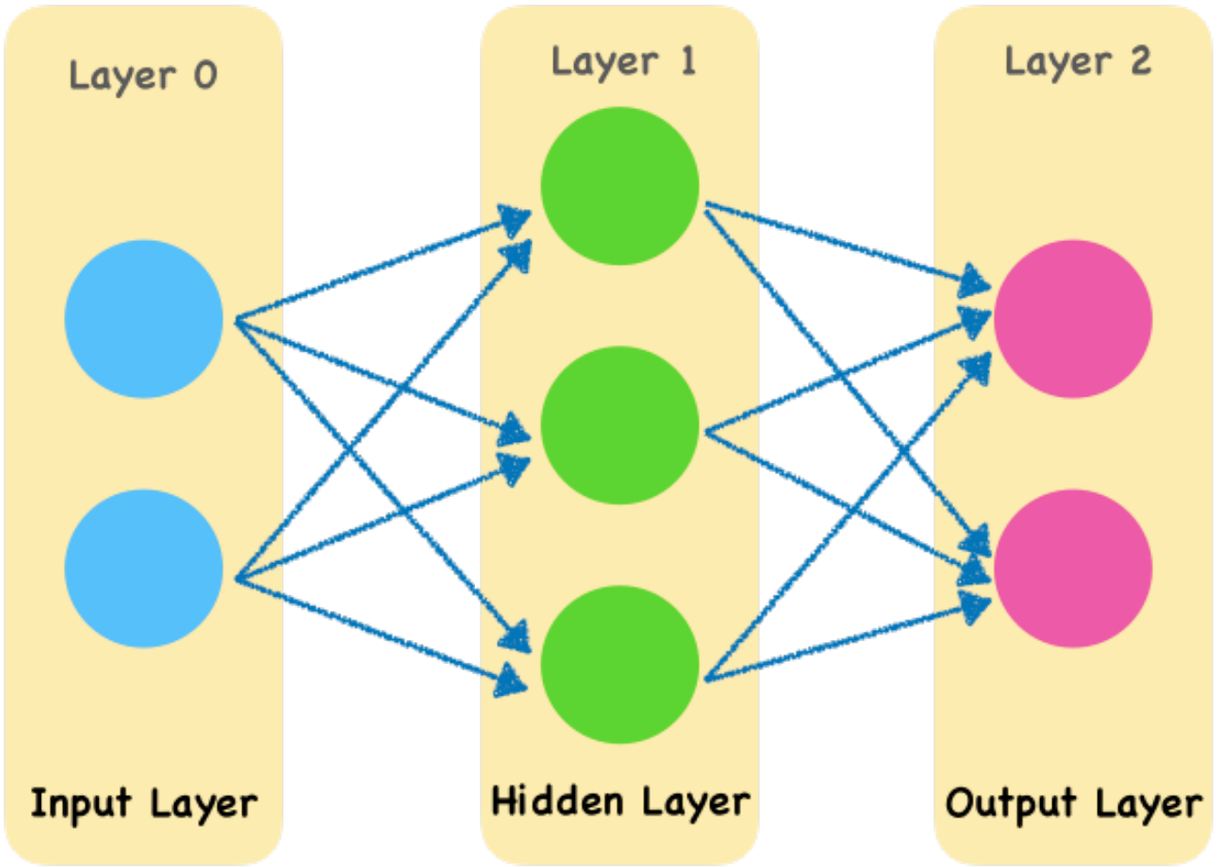
Multilayer networks of perceptrons

Perceptron networks



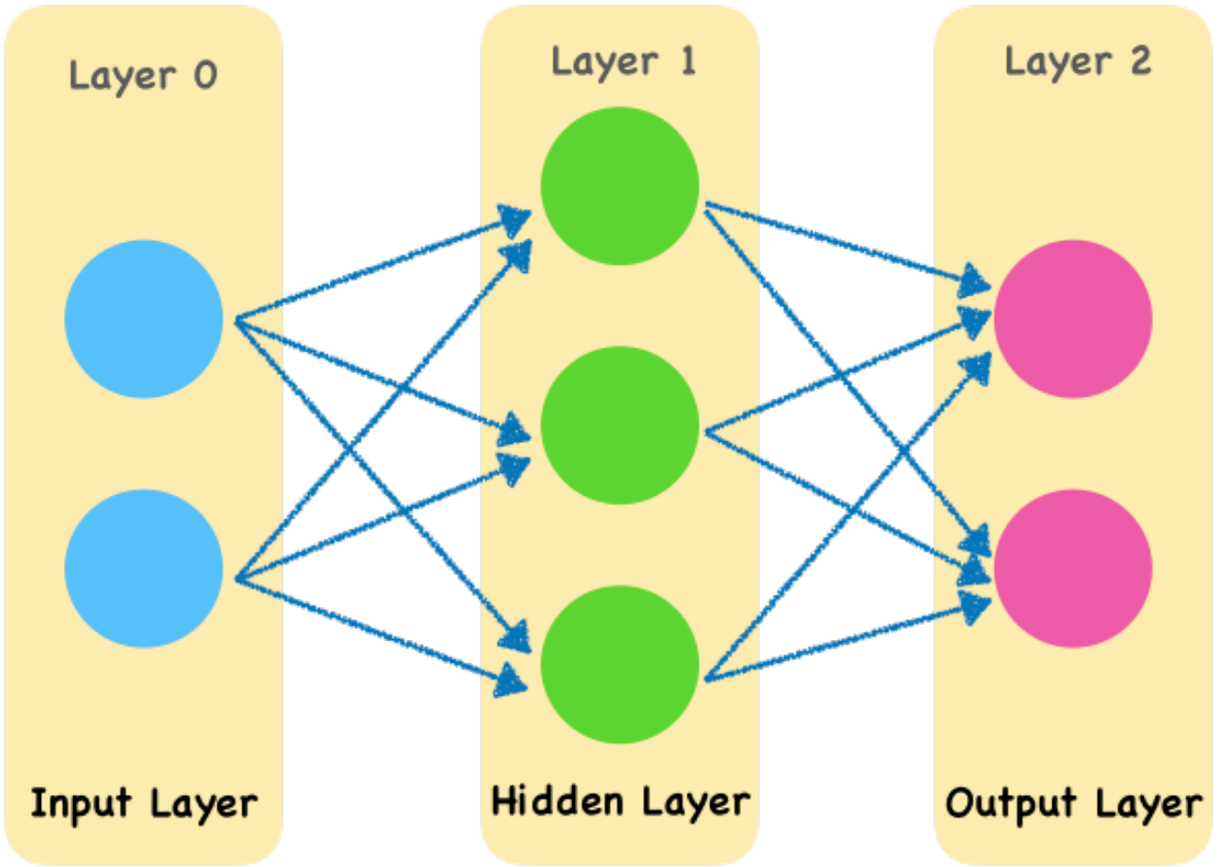
Many-to-one and many-to-many connections

Multilayer perceptron



- Multilayer perceptron also named deep feedforward networks.
- It adds hidden layers to the previous perceptron.

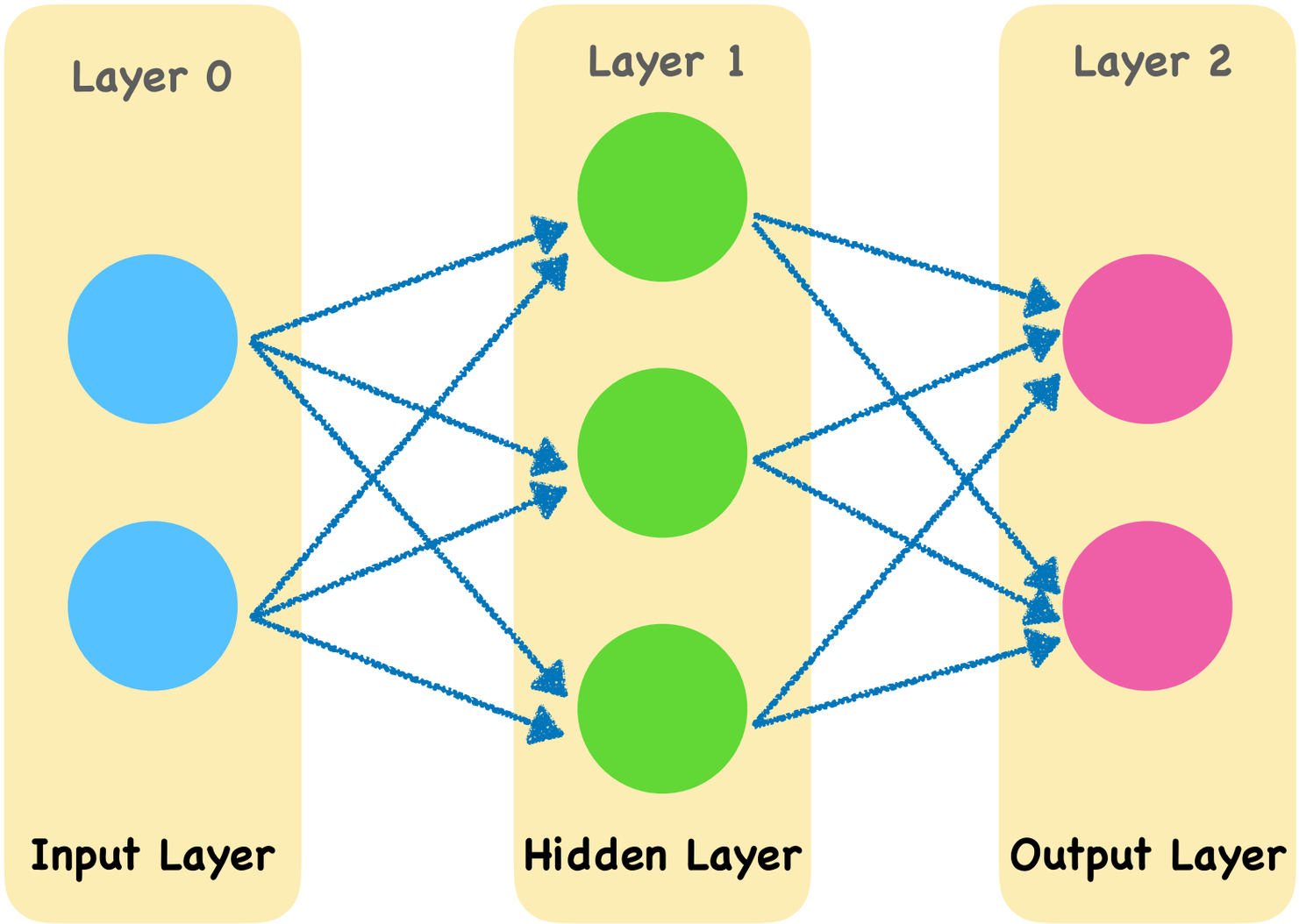
Multilayer perceptron



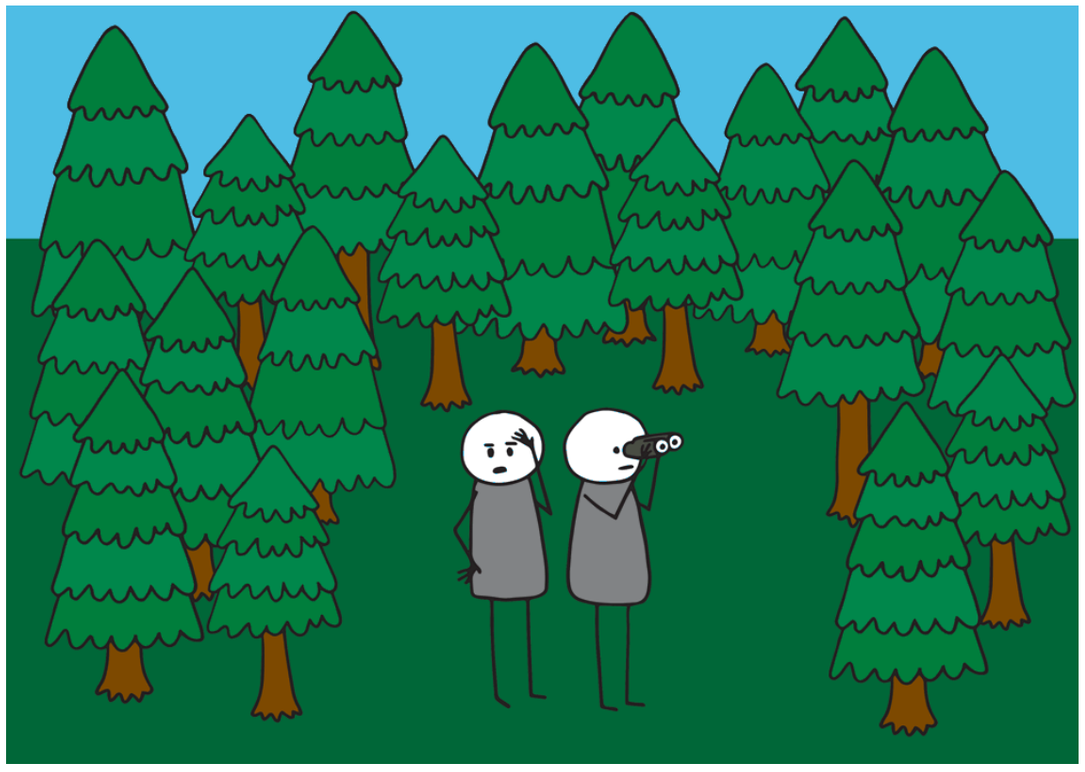
- The leftmost layer 0 is the **input layer.**
- The rightmost layer 2 is the **output layer.**
- The middle layer 1 is called **hidden layer.**
- Here the input layer and output layer have 2 nodes and the hidden layer has 3 nodes.
- Although there are three layers, only **two of them has weight**, so we call this network as **a two-layer network**.

Multilayer perceptron

Animation Process



Why is the middle layer called a hidden layer?

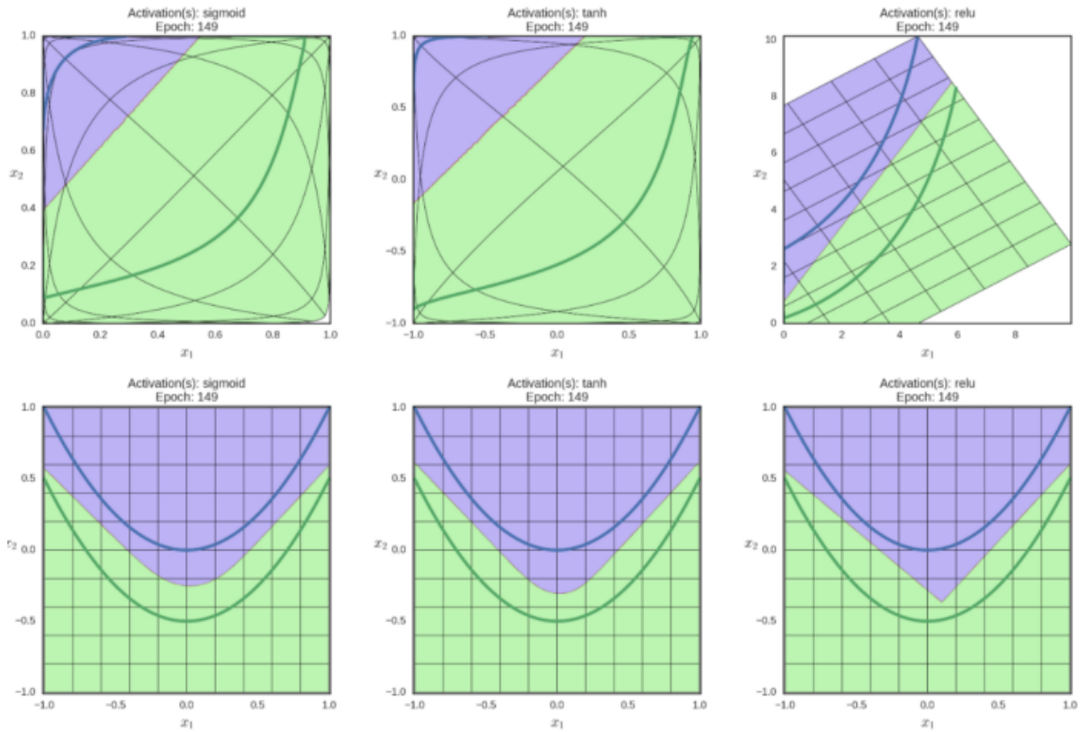


The data of the input layer and output layer are strictly observable, while the data value of the hidden layer is learned by the model.

Therefore, we cannot see them in advance. They are "hidden" for us.

Activation functions

Space wizard



What is an activation function?

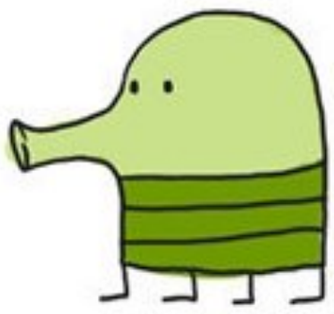
We call the function that takes all the inputs to transform an output the **activation function**

Commonly used activation functions

Step Function

The definition of step function is:

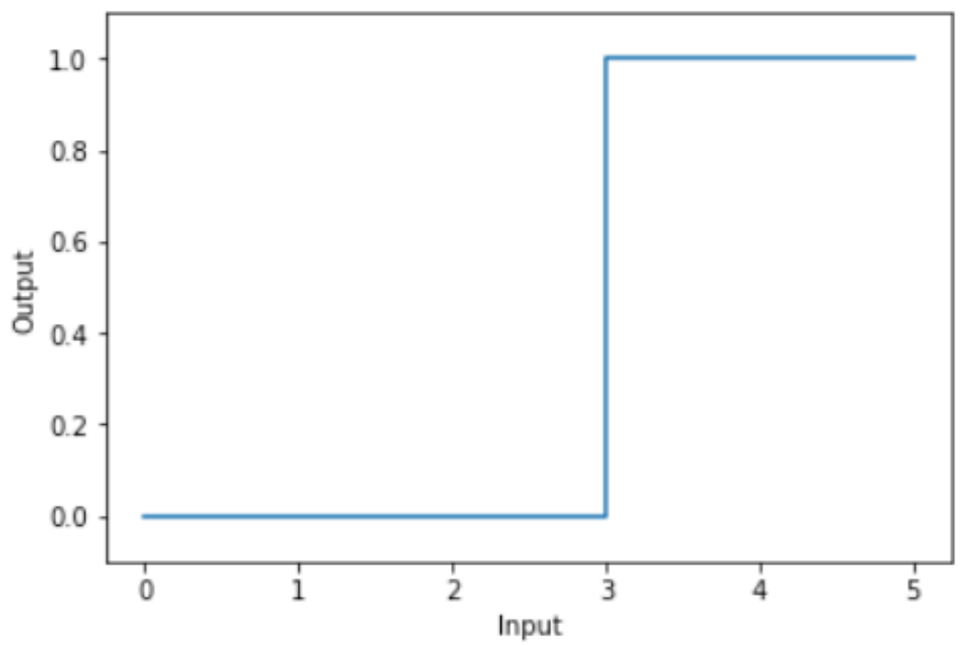
$$h(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$



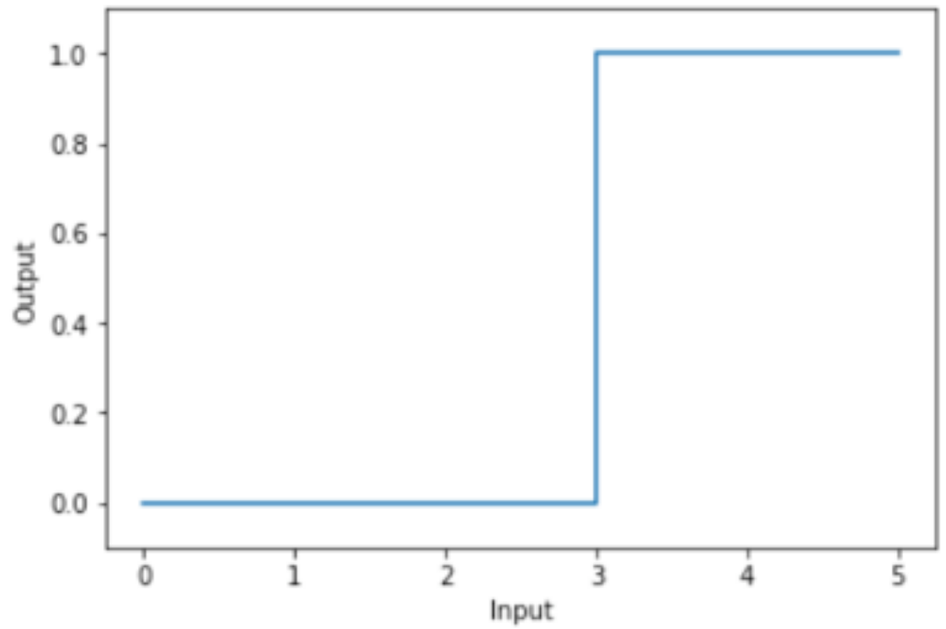
Step is a guy who likes to bounce around and often has a poor performance

```python
def step_func(x):
    # for each element in x, determine weather it positive
    # return the result as int array type
    return np.array(x > 0, dtype=np.int)
```

```python
import matplotlib.pyplot as plt
%matplotlib inline

plt.xlabel("Input")
plt.ylabel("Output")

plt.plot([0,3,3,5],[0,0,1,1])
plt.ylim(-0.1,1.1)

plt.show()
```



Step Function



Step function works like a leaky shishi-odoshi（竹筒敲石）. It's a type of water fountain used in Japanese gardens. It consists of a segmented tube, usually of bamboo, pivoted to one side of its balance point. At rest, its heavier end is down and resting against a rock.

竹筒敲石是日本的一种庭院设施。支点架起竹筒，一端下方置石，另一端切口上翘。在切口上滴水， 水积多后该端下垂，水流出，另一端翘起，之后又因重力而落下，击石发出响声。

Commonly used activation functions

Sigmoid Function

The definition of sigmoid function is:

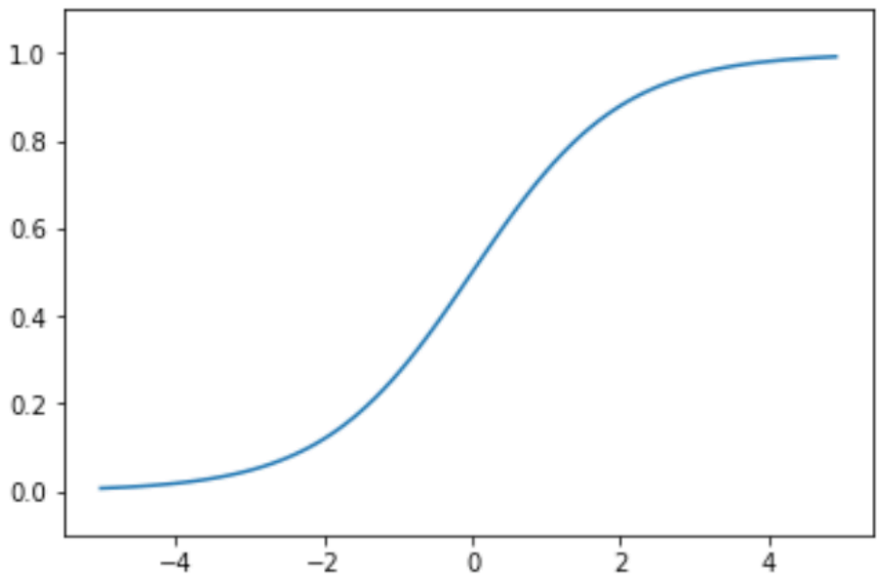
$$h(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid, a grandmother,
the most senior of the activate functions.

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Sigmoid function is a S-shaped function.

```python
X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
plt.show()
```



Commonly used activation functions

ReLU Function

The definition of ReLU(rectified linear unit) function is:

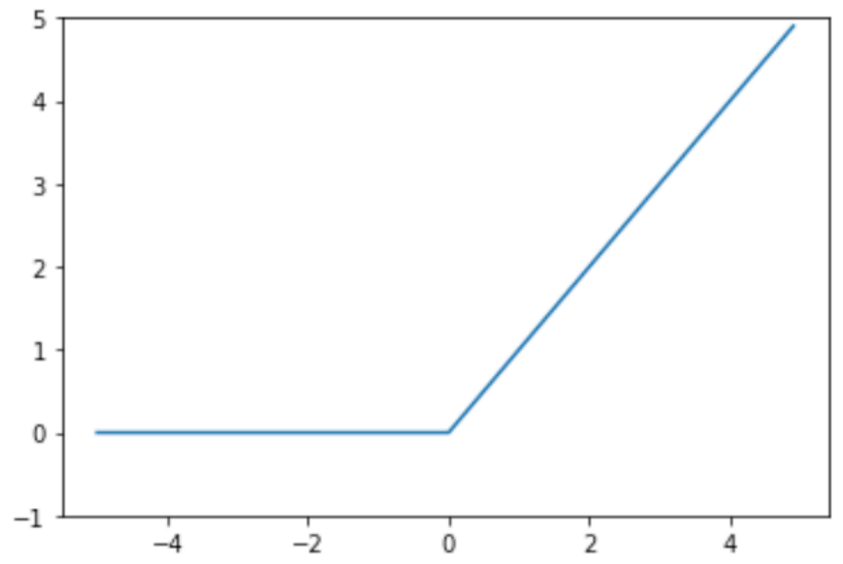
$$h(x) = max(x,0)$$

ReLU is a gatekeeper.
Muggles are shut out.

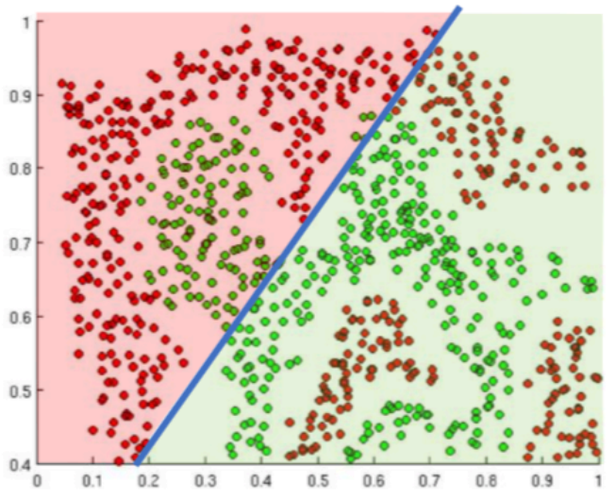
```python
import numpy as np
def relu(x):
    # return the maxium number bwtween 0 an each element of vector x
    return np.maximum(0, x)
```

Let's plot it.

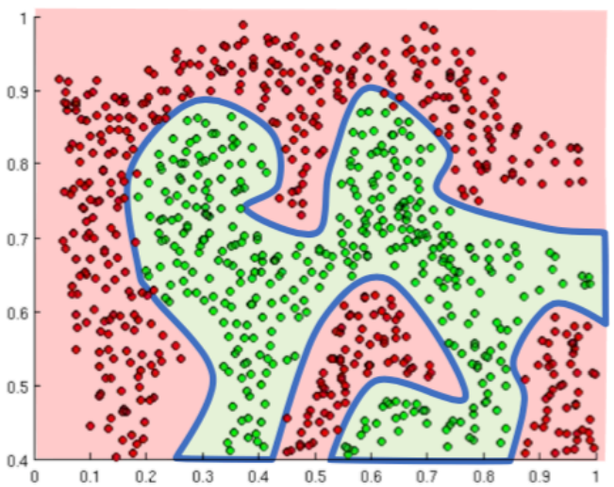
```python
x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)

plt.plot(x, y)
plt.ylim(-1.0, 5)
plt.show()
```



Why must the activation function be non-linear?



Linear Activation functions produce linear decisions no matter the network size

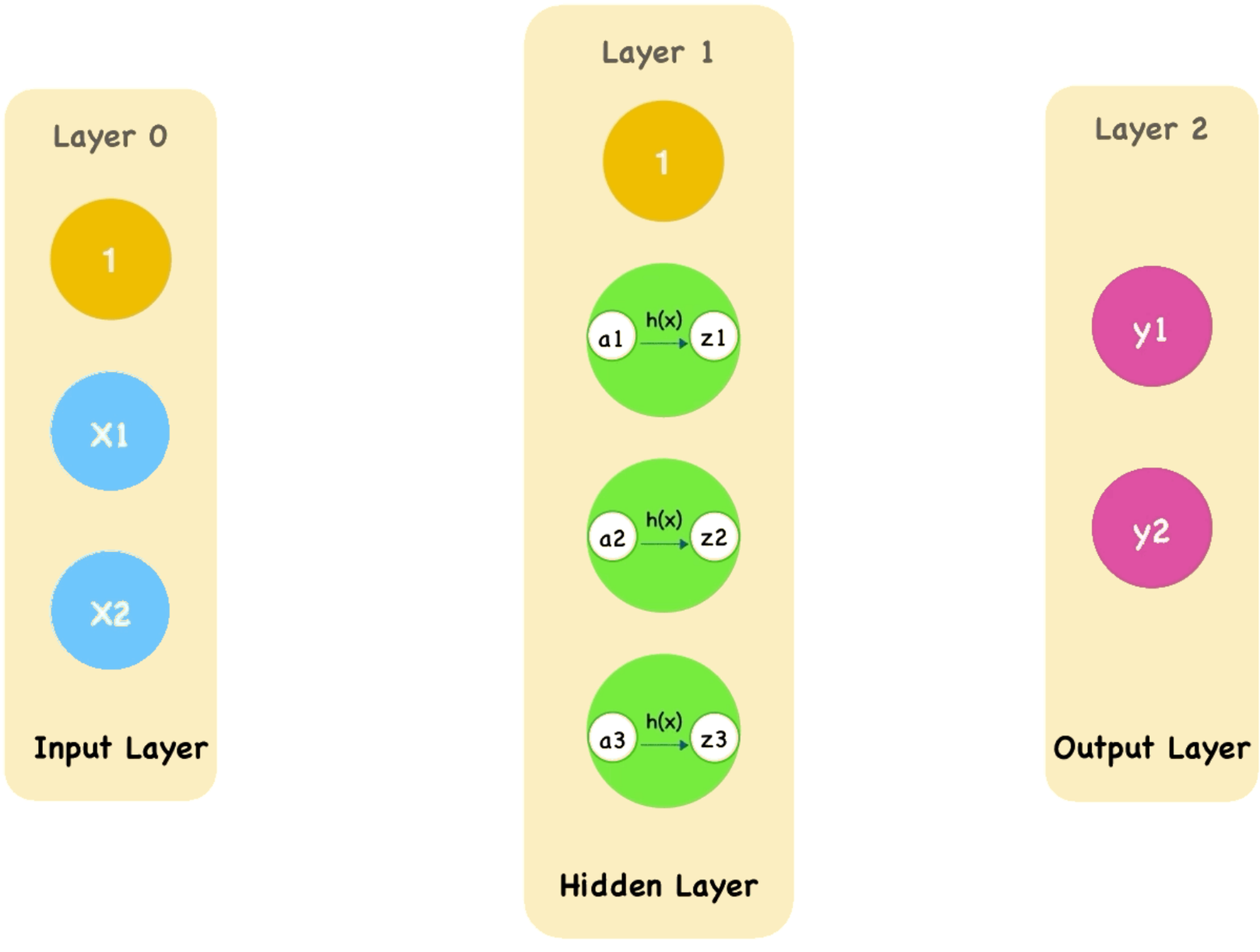


Non-linearities allow us to approximate arbitrarily complex functions

- All the activation functions we introduced are non-linear functions
- Why? Generally speaking, the purpose of activation function is to introduce non-linearities into the network

Forward Propagation

This animation shows how the single pass through the network with an activate function h(x):



C. Network Structure

Forward propagation by weights and biases

Example: Handwritten Digits Recognition

- The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples.
- The digits have been size-normalized and centered in a fixed-size image.

28

28

$$28 \times 28 = 784$$

Pixel

0.57

28

28

$28 \times 28 = 784$

Pixel

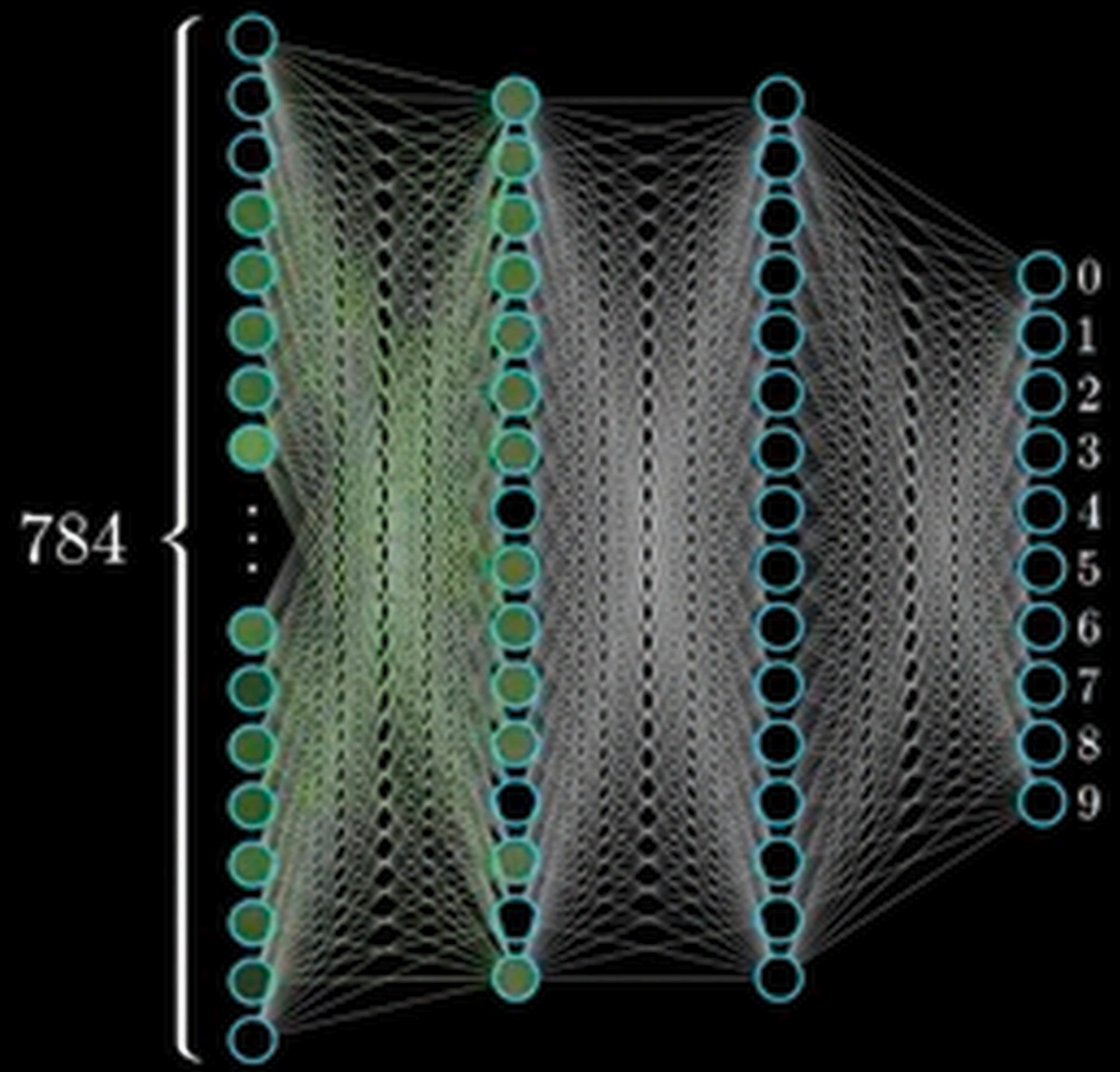
0.58

“Activation”

784

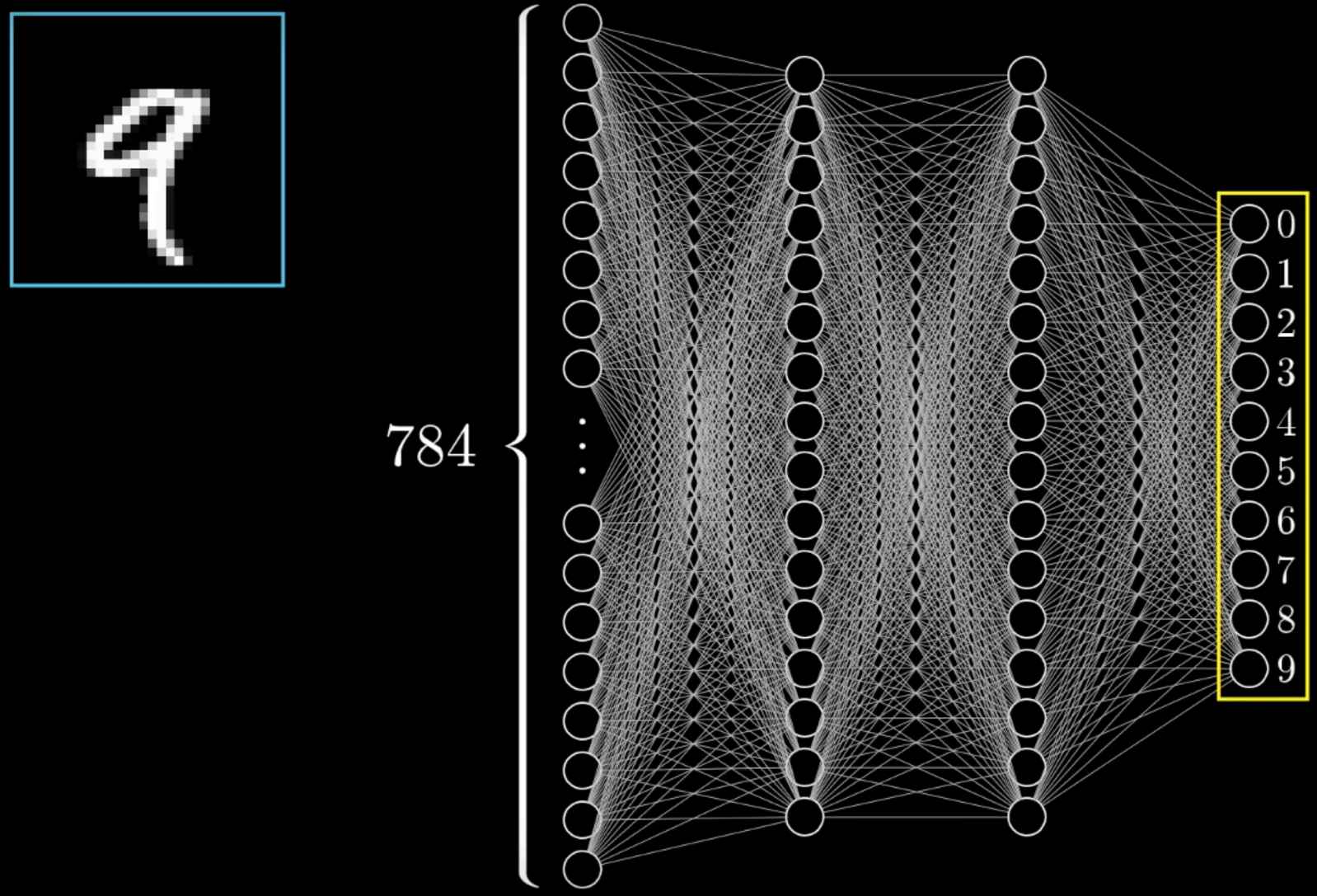
Plain vanilla

(aka "multilayer perceptron")



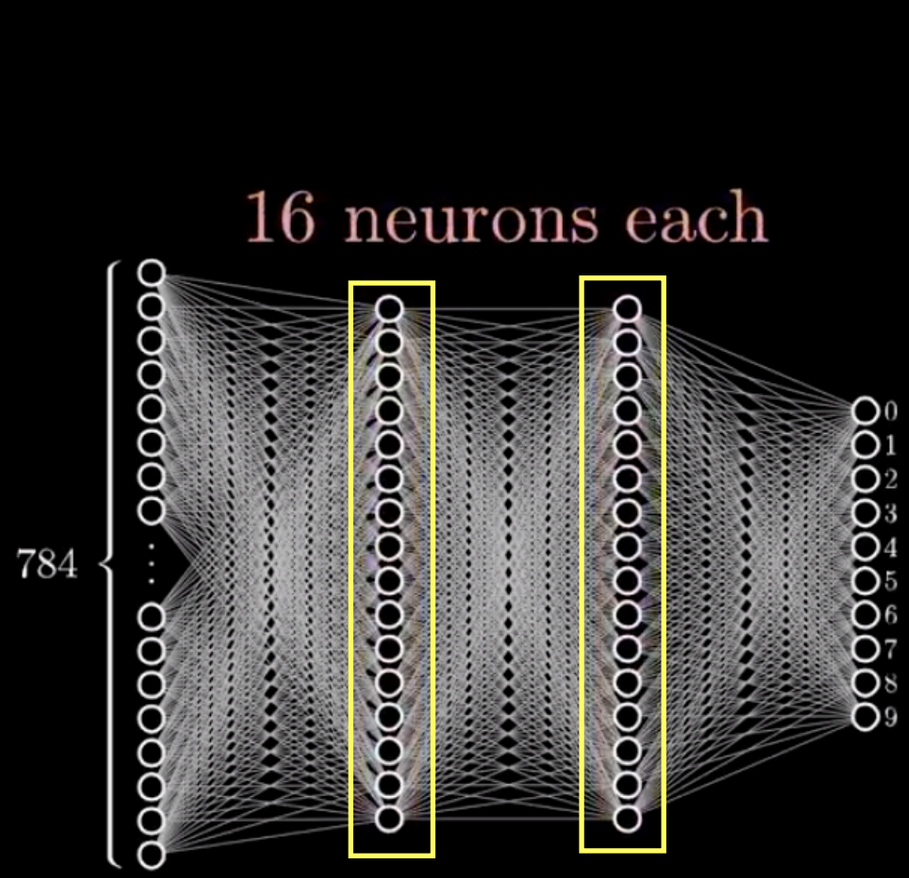
Example: Handwritten Digits Recognition

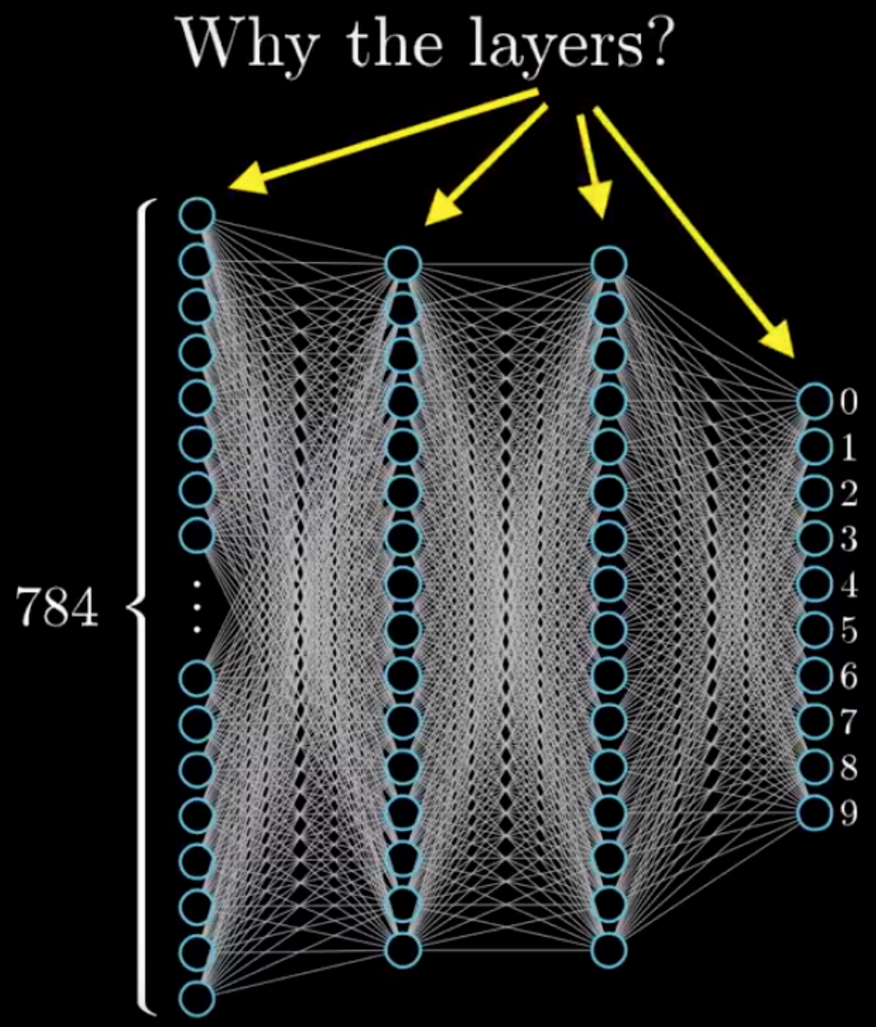
Output Layer



Example: Handwritten Digits Recognition

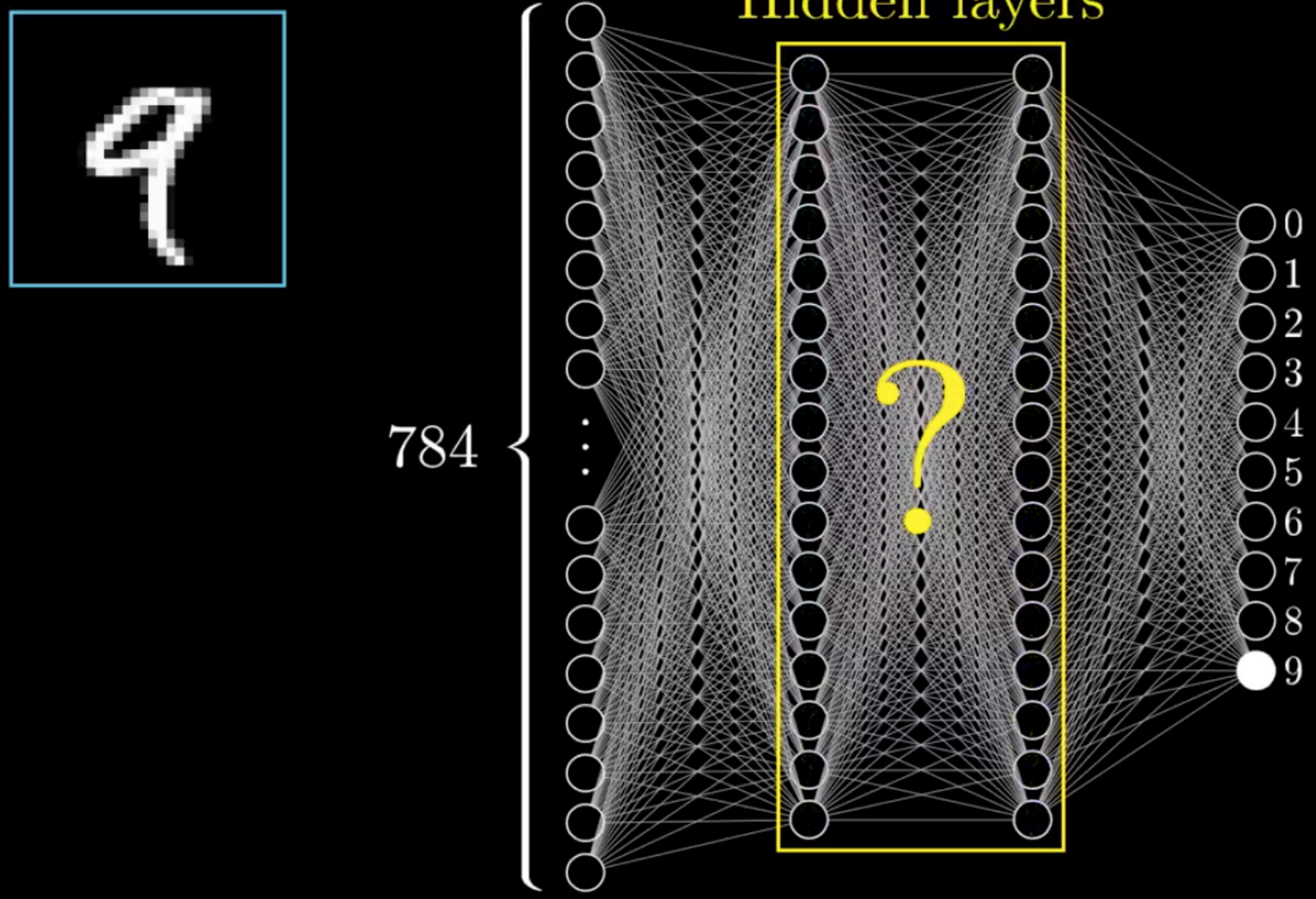
Why the Layers?





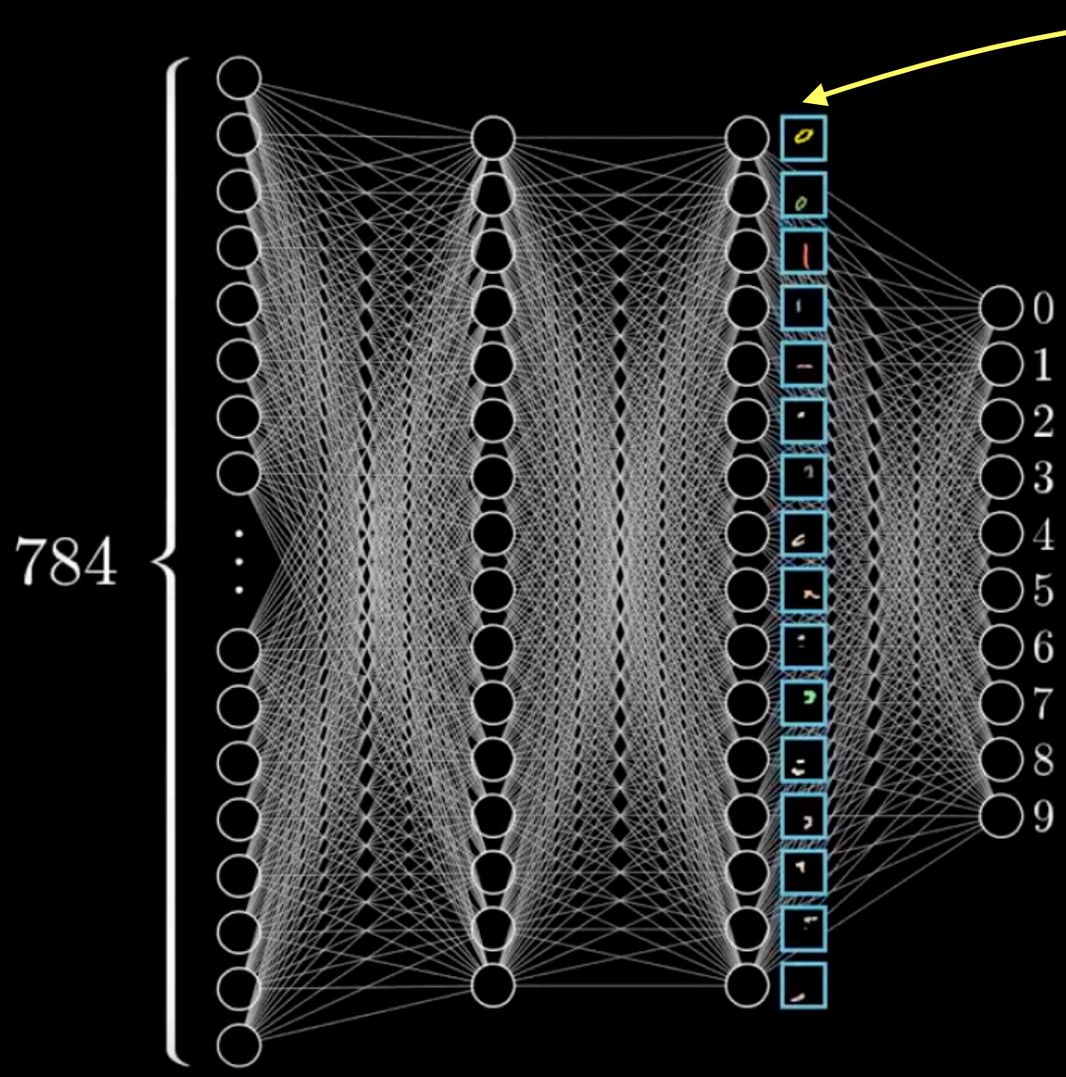
Example: Handwritten Digits Recognition

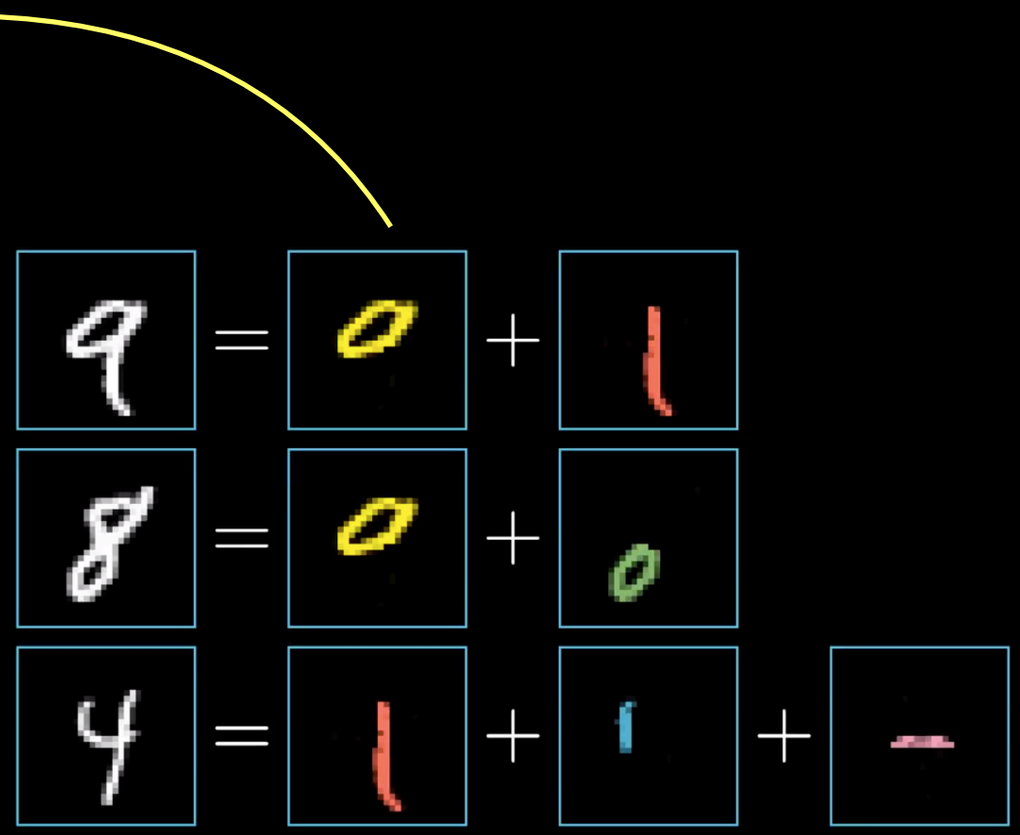
Hidden Layers



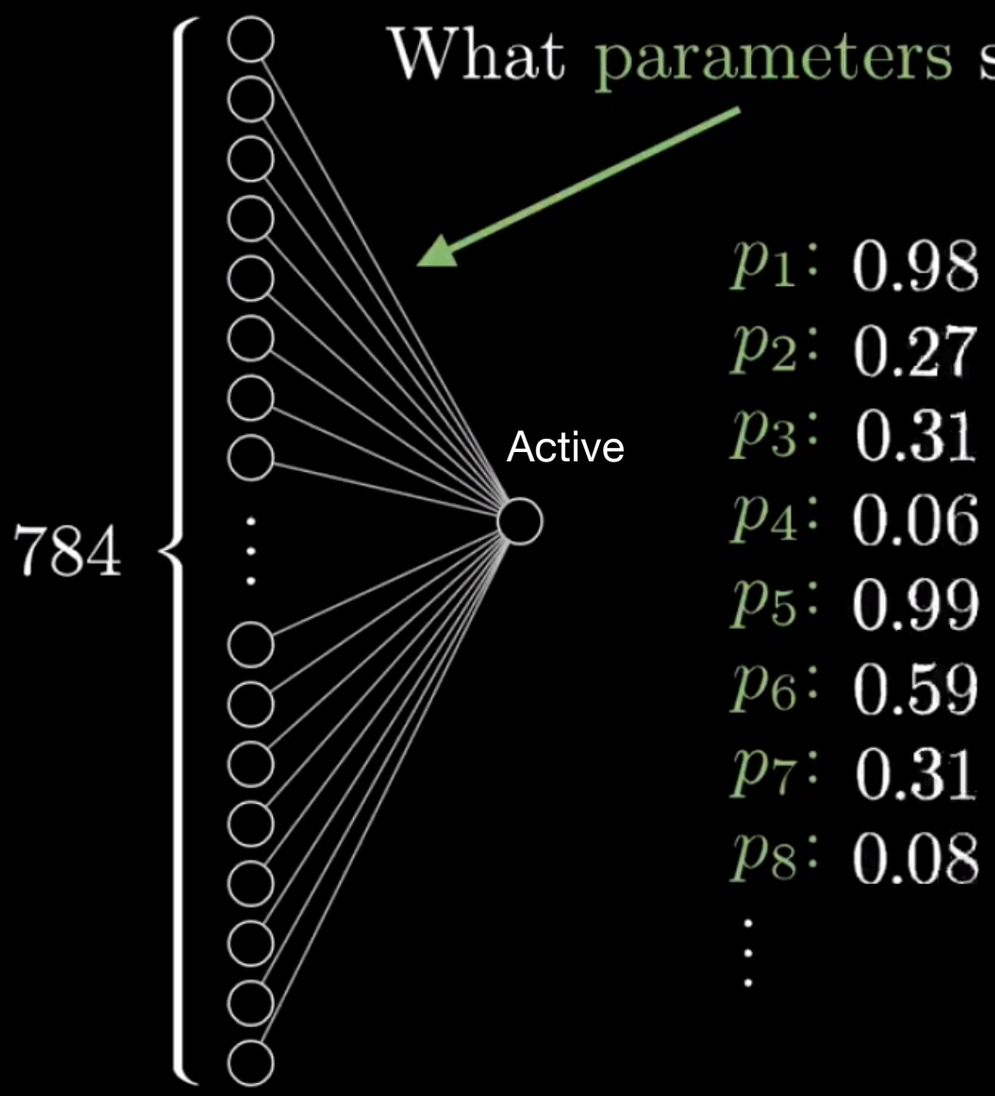
Example: Handwritten Digits Recognition

Why the Layers?

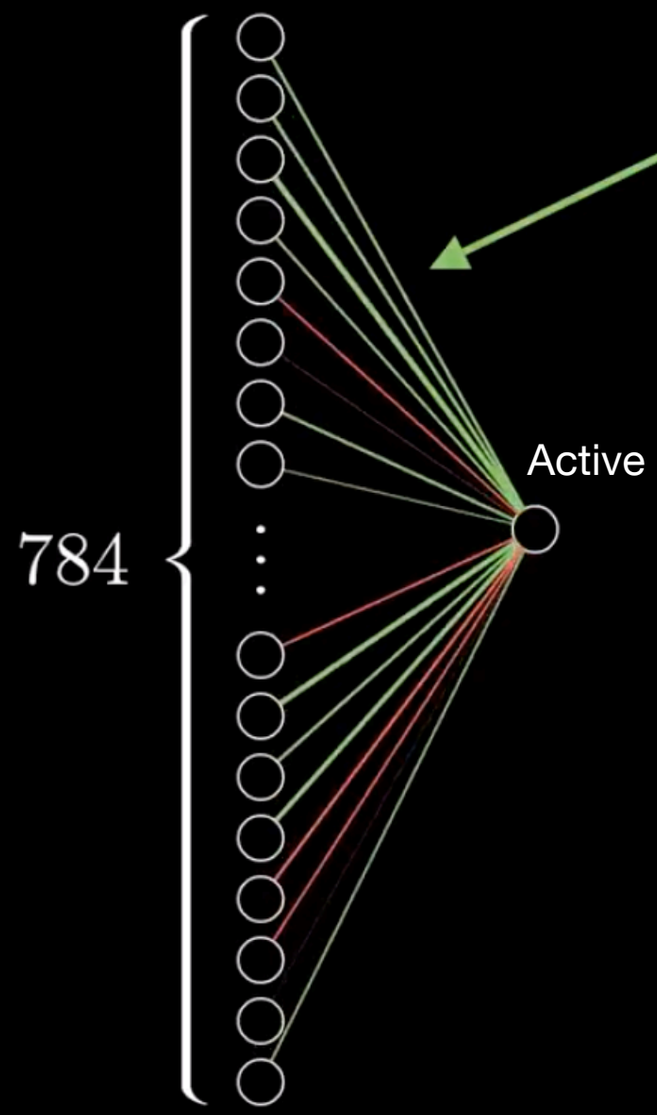




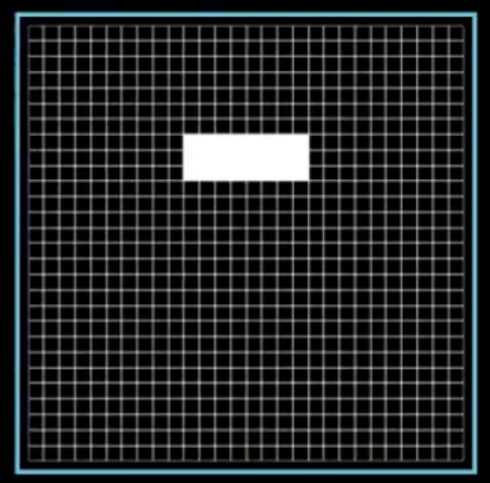
Example: Handwritten Digits Recognition



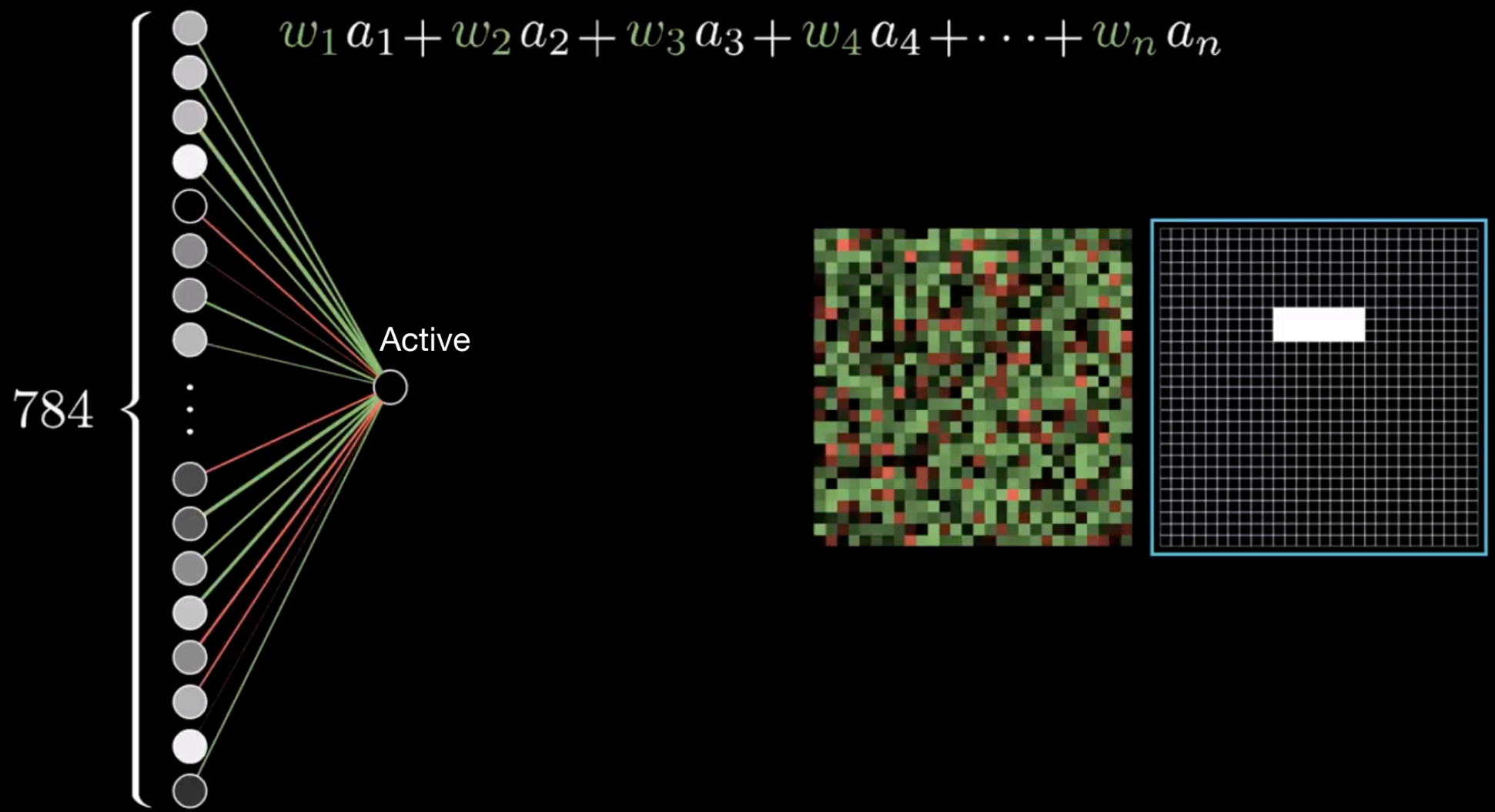
Example: Handwritten Digits Recognition



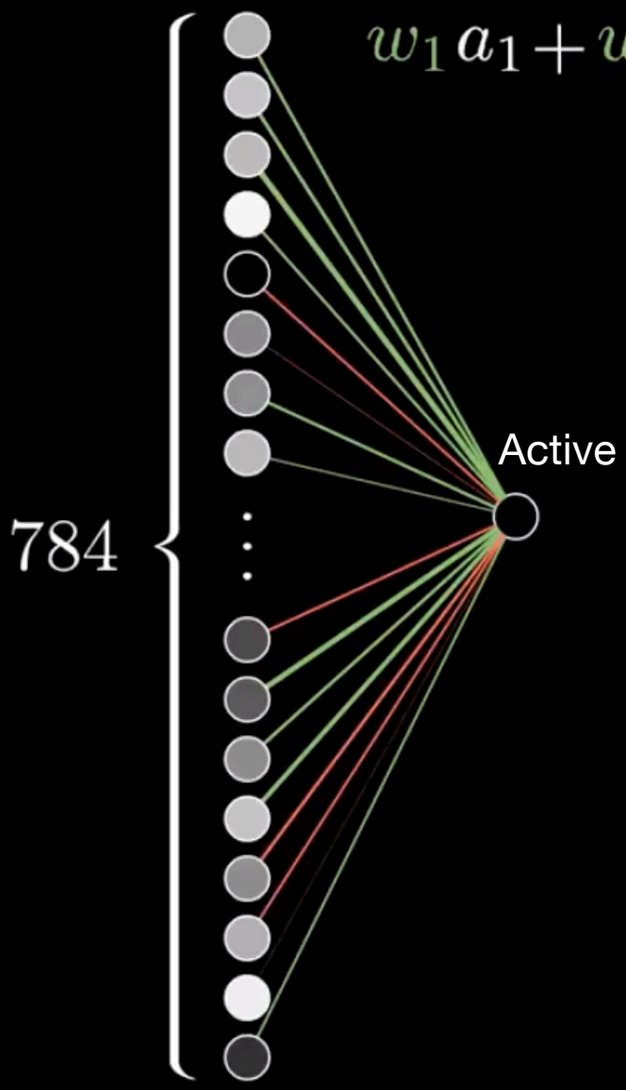
w_1: 2.07
w_2: 2.31
w_3: 3.64
w_4: 1.87
w_5: −1.51
w_6: −0.43
w_7: 2.01
w_8: 1.07
⋮

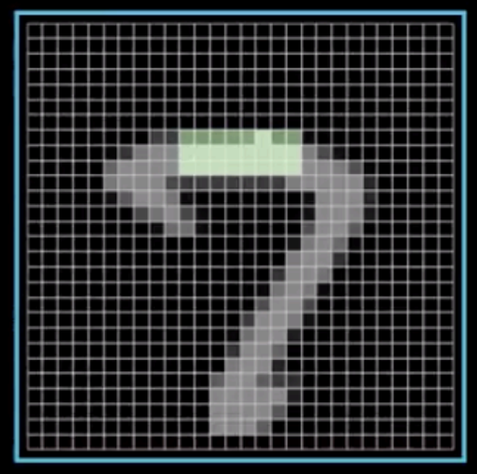


Example: Handwritten Digits Recognition



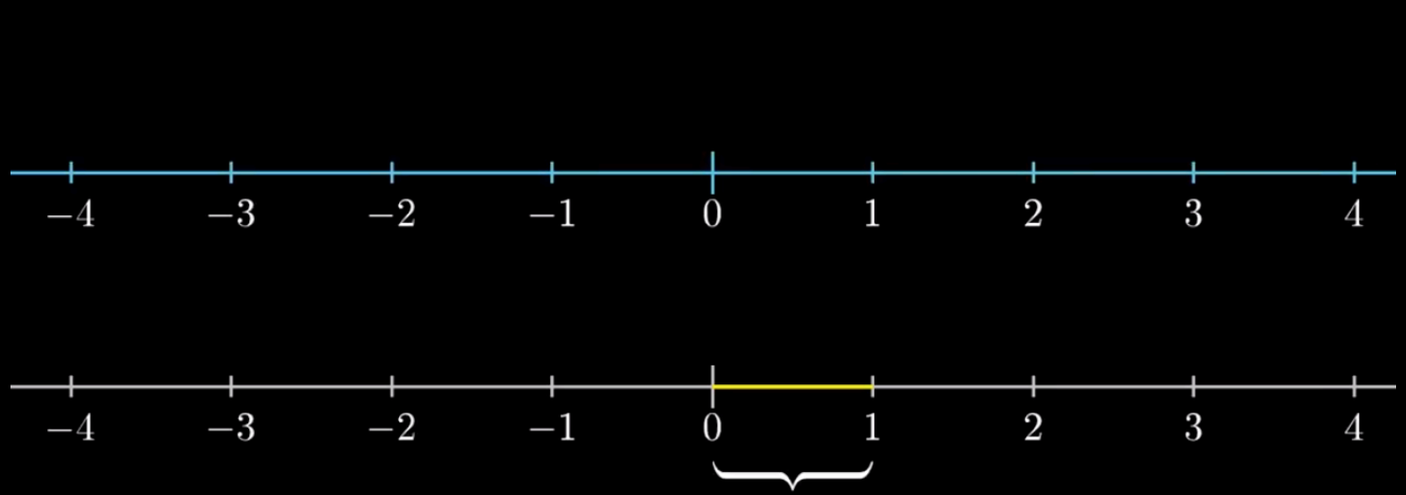
Example: Handwritten Digits Recognition

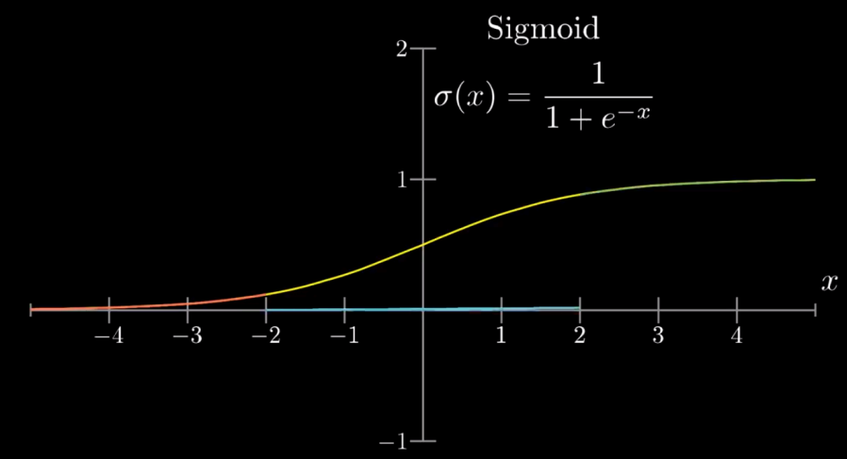




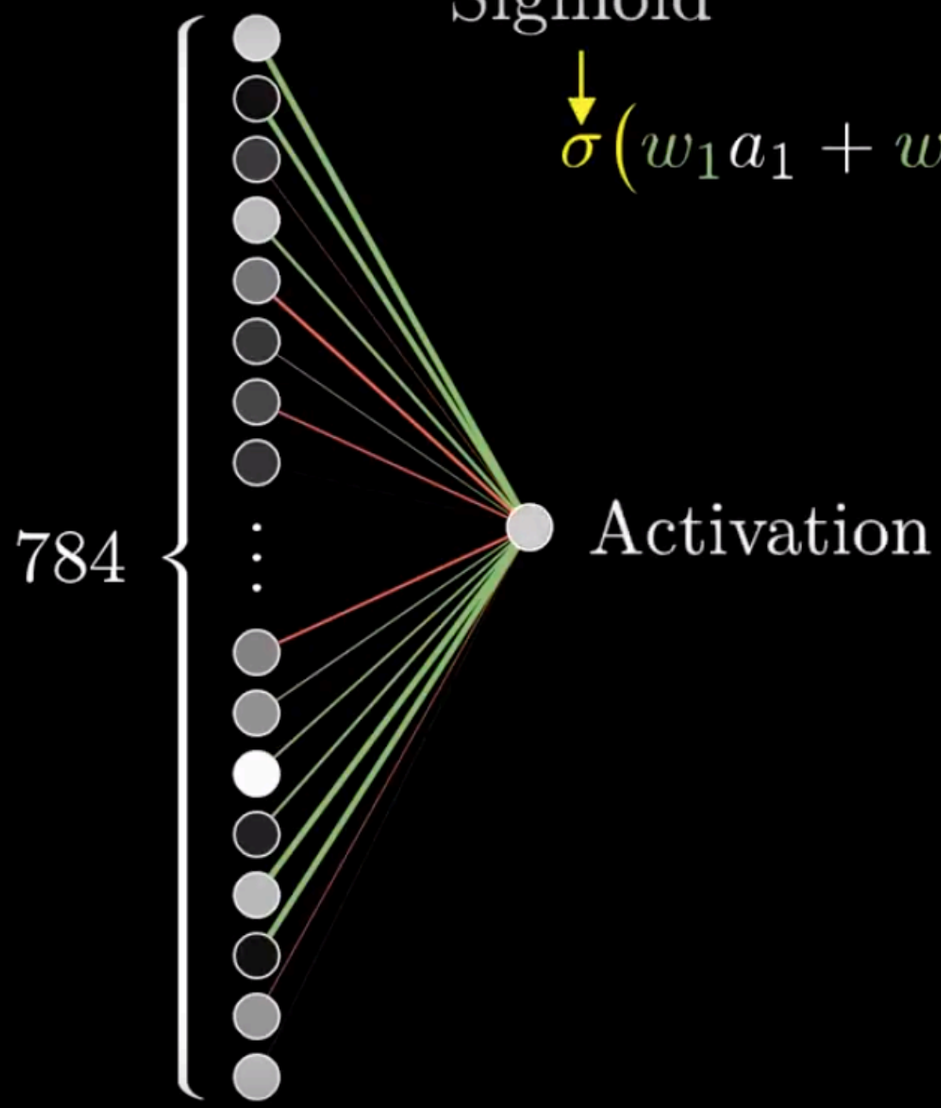
Example: Handwritten Digits Recognition

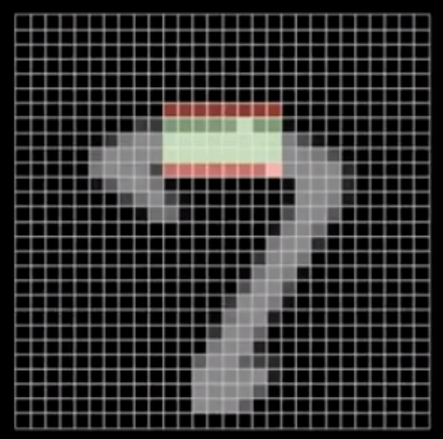
$$w_1 a_1 + w_2 a_2 + w_3 a_3 + w_4 a_4 + \cdots + w_n a_n$$



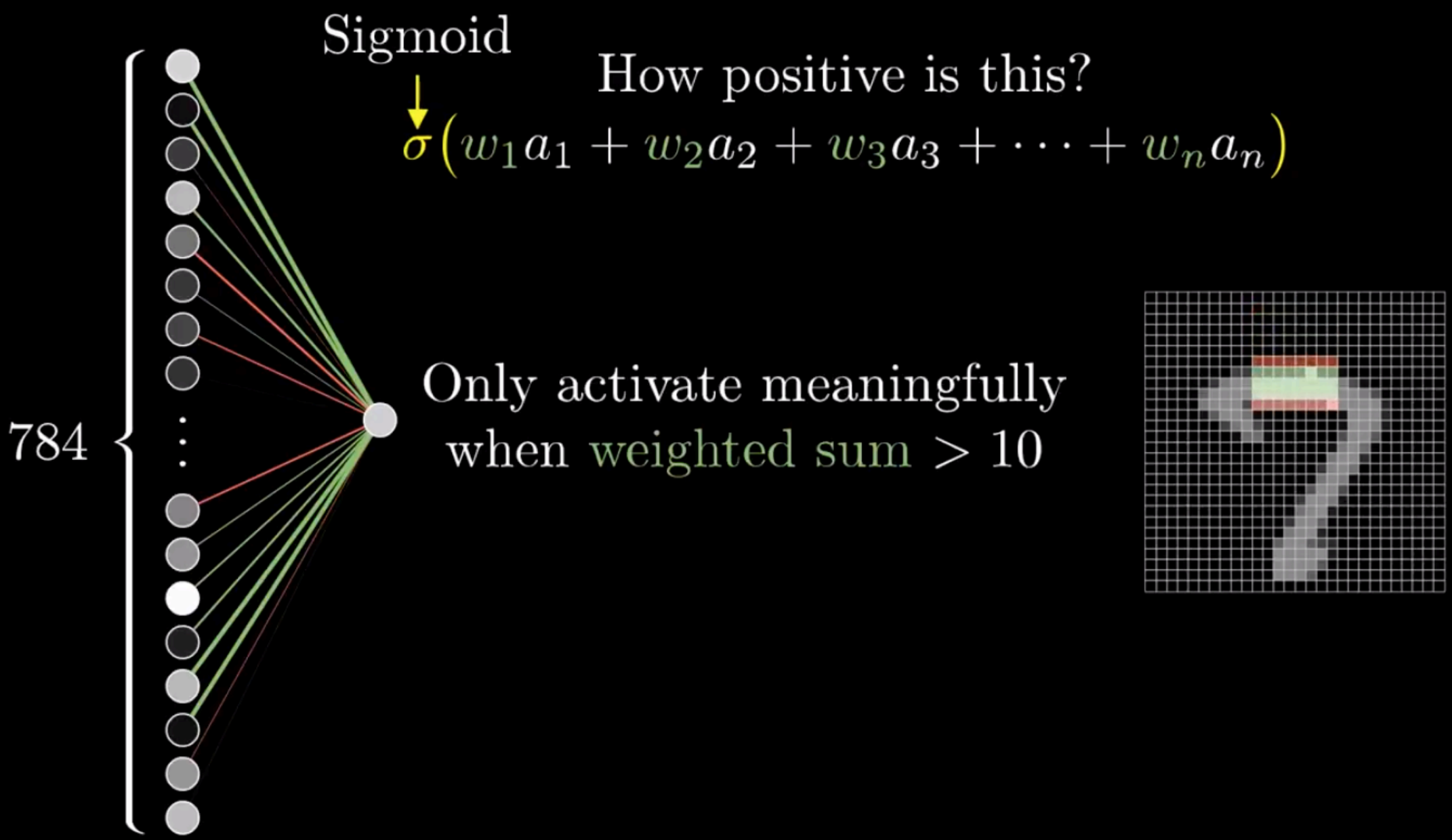


Example: Handwritten Digits Recognition

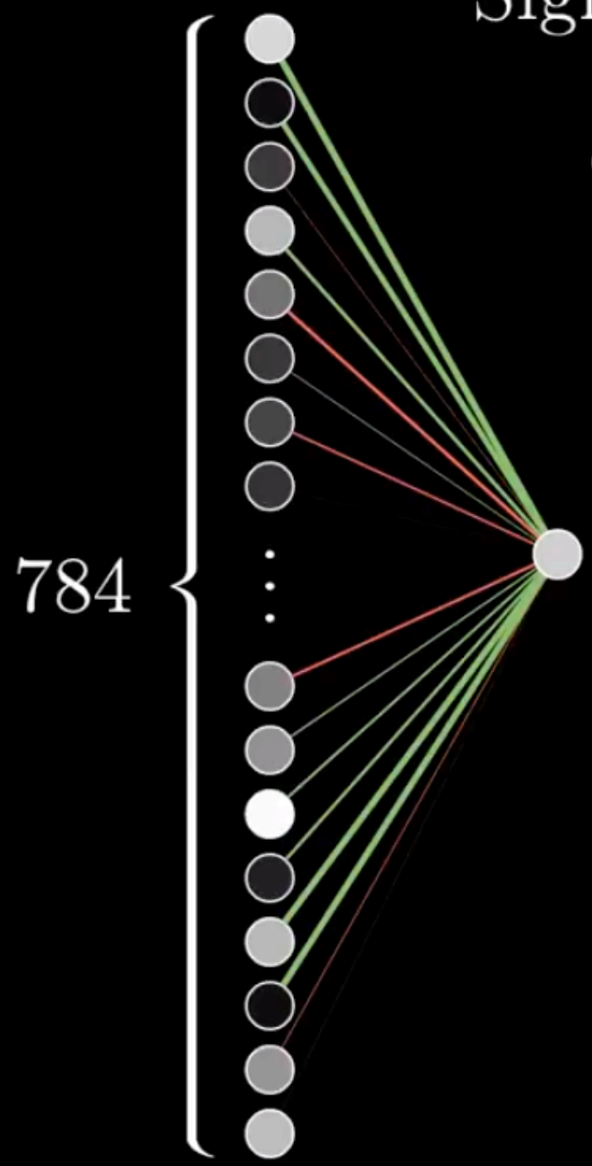


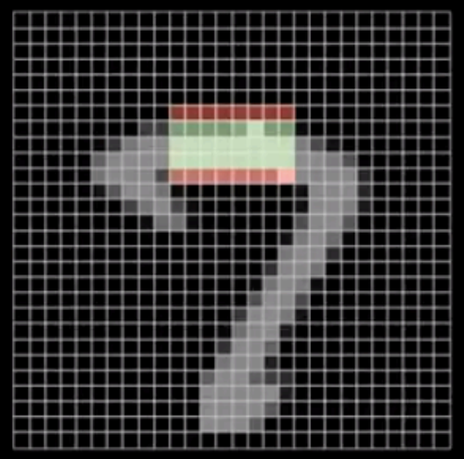


Example: Handwritten Digits Recognition

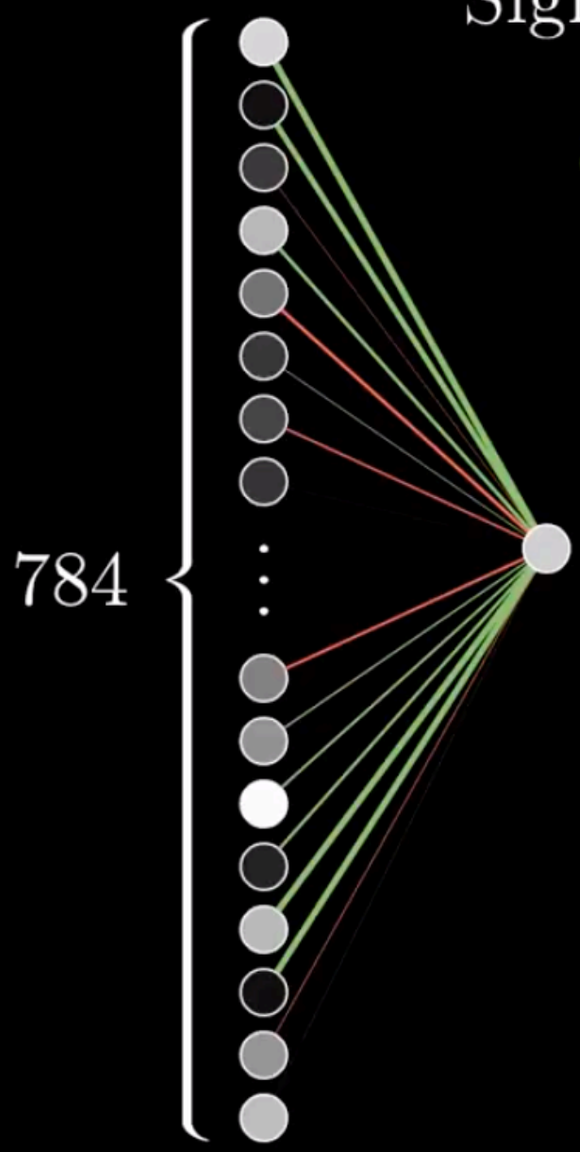


Example: Handwritten Digits Recognition





Example: Handwritten Digits Recognition



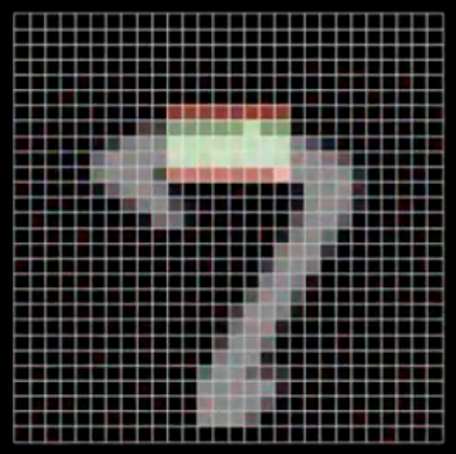
Sigmoid

How positive is this?

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

"bias"

Only activate meaningfully when weighted sum > 10

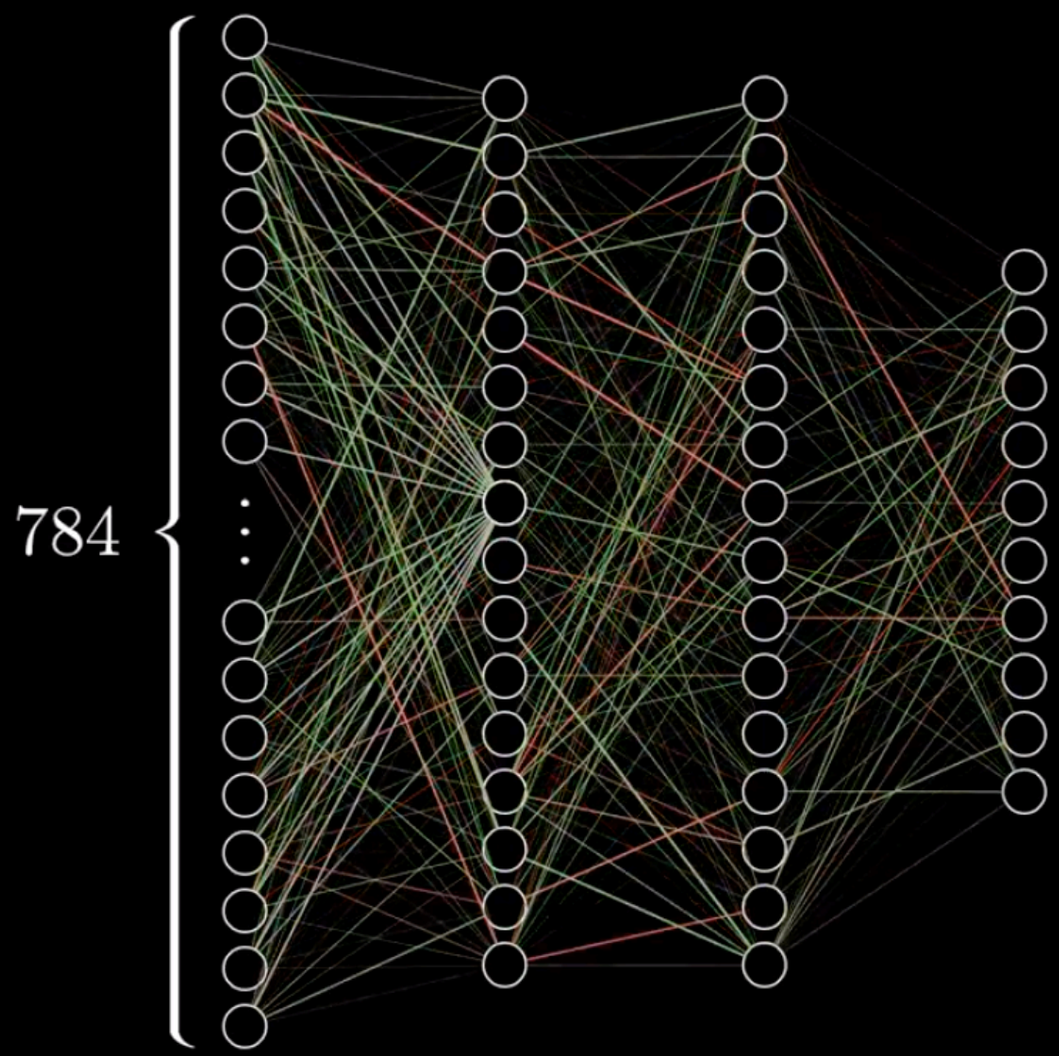


784

0
1
2
3
4
5
6
7
8
9

Example: Handwritten Digits Recognition

Learning

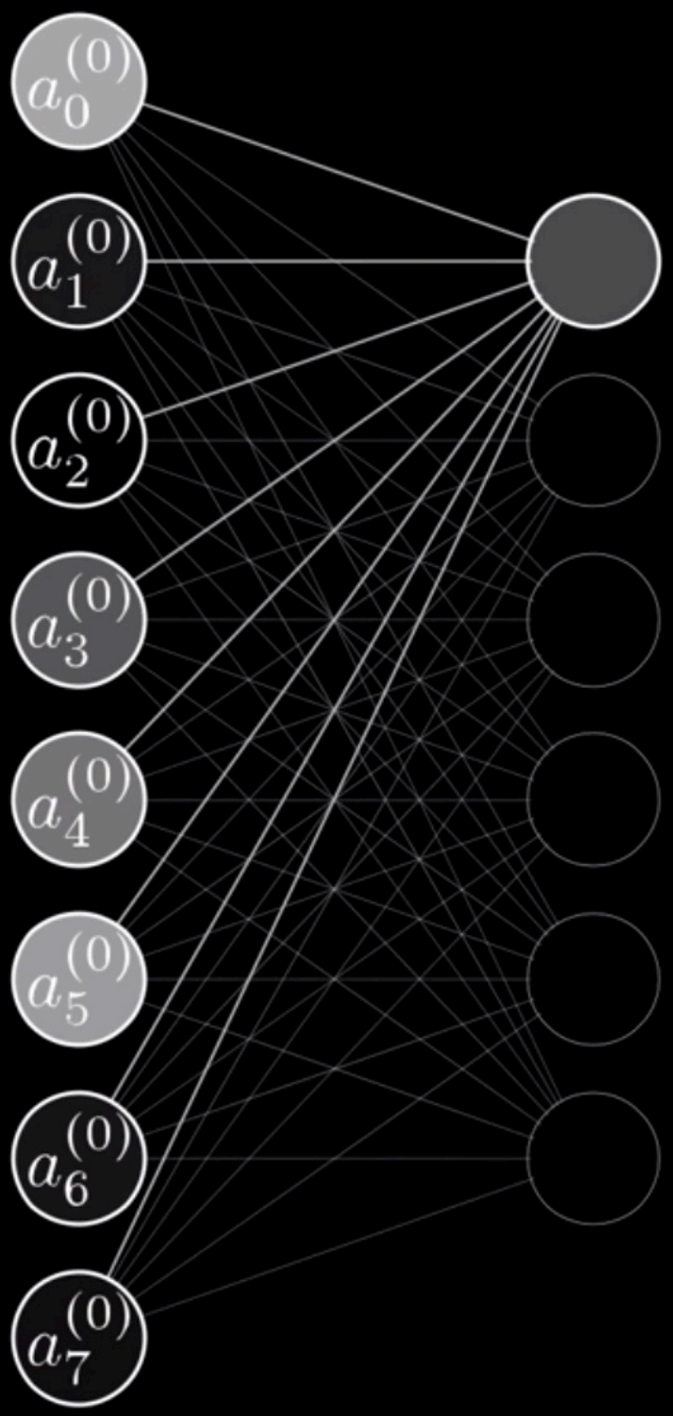


$$784\times16+16\times16+16\times10$$
weights

$$16+16+10$$
biases

$$13{,}002$$

Learning $\rightarrow$ Finding the right weights and biases

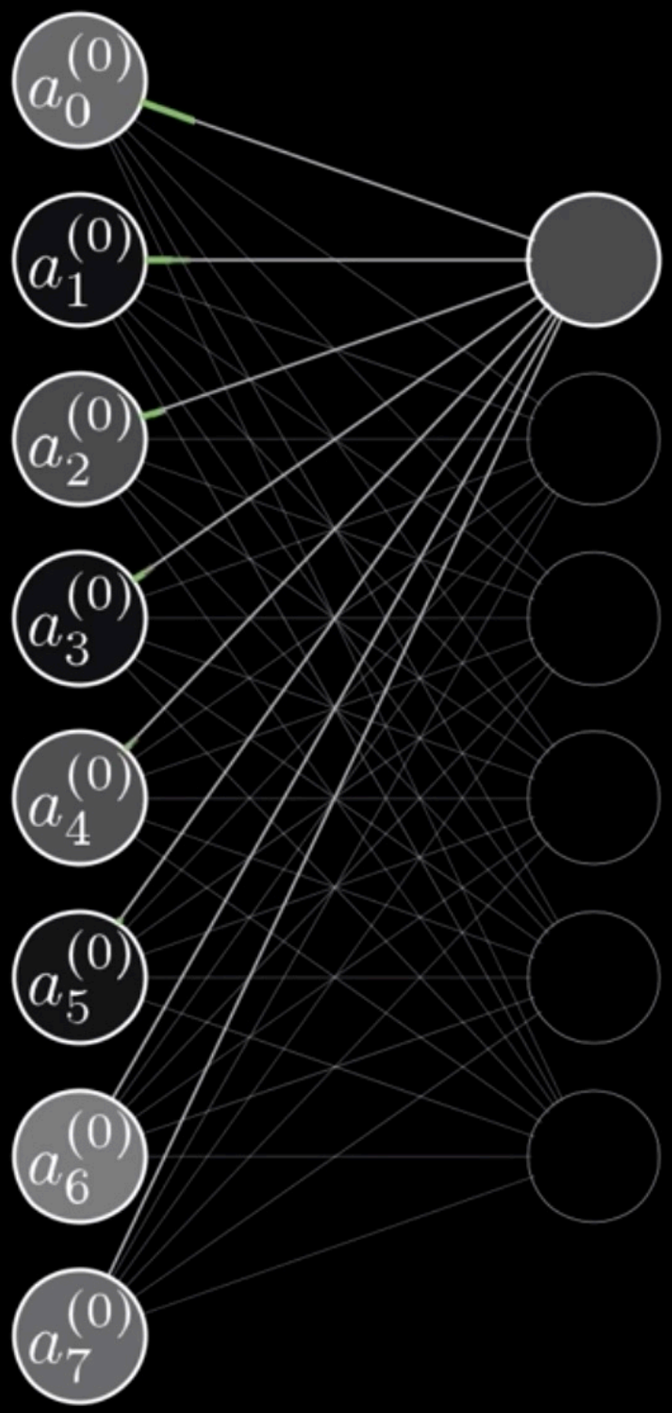


Sigmoid

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

Bias

$$\begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix}$$

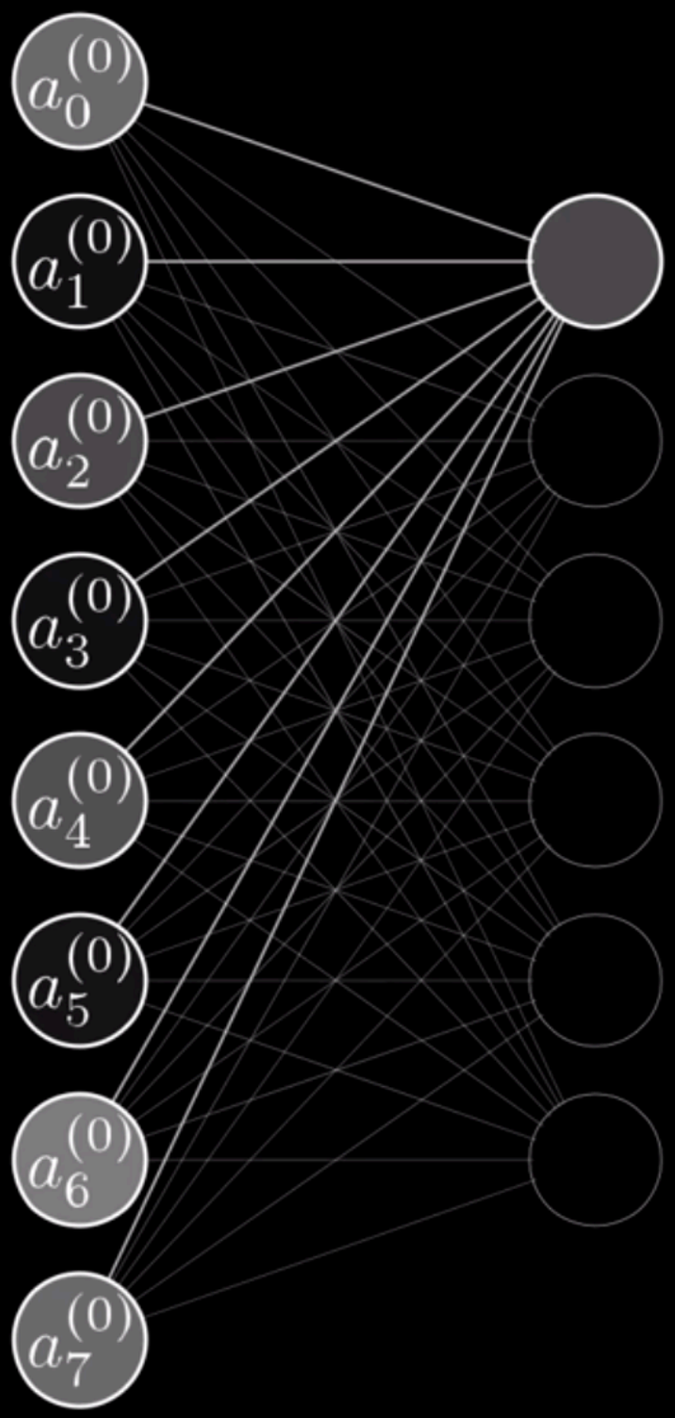


Sigmoid

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

Bias

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



Sigmoid

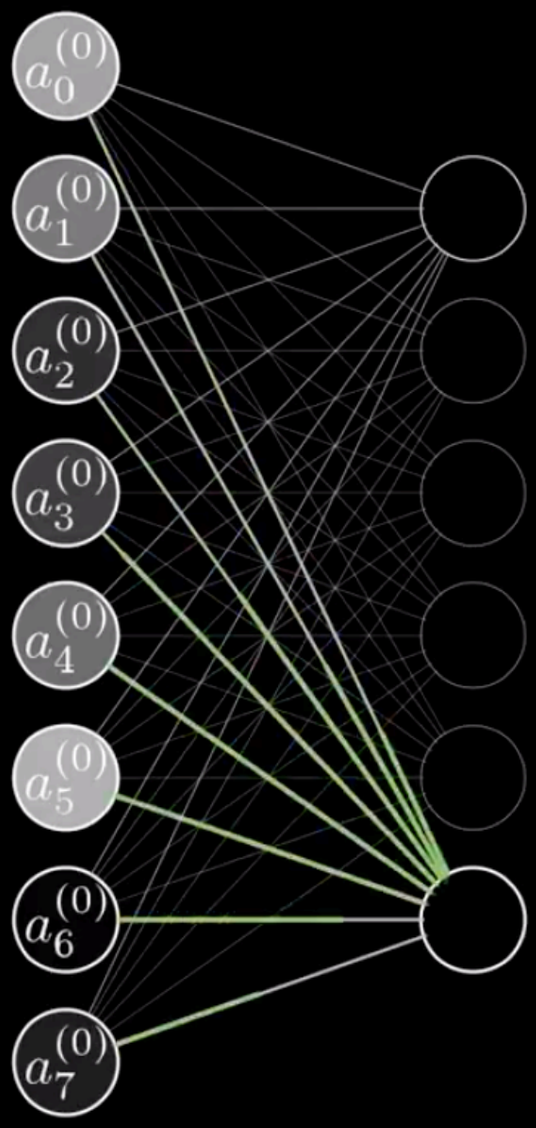
$$a_0^{(1)} = \sigma\left(w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)} + b_0 \right)$$

Bias

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

Example: Handwritten Digits Recognition

Matrix



Sigmoid

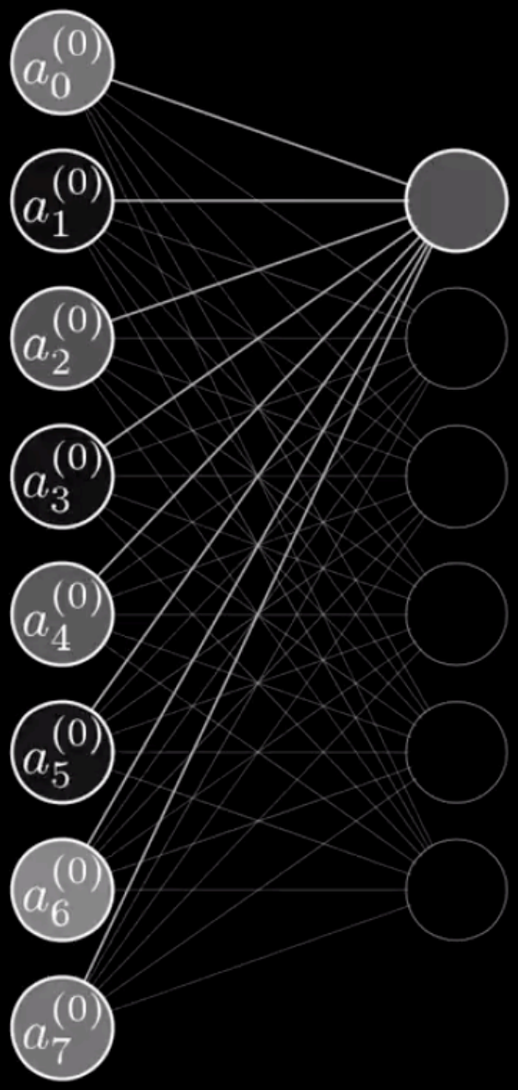
$$a_0^{(1)} = \sigma\left(w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)} + b_0 \right)$$

Bias

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

Example: Handwritten Digits Recognition

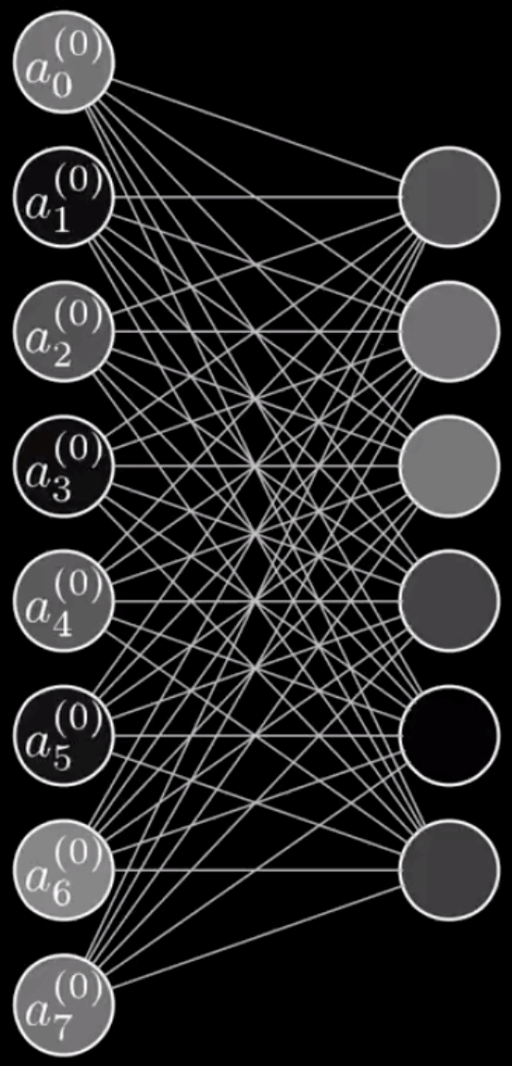
Matrix



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

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

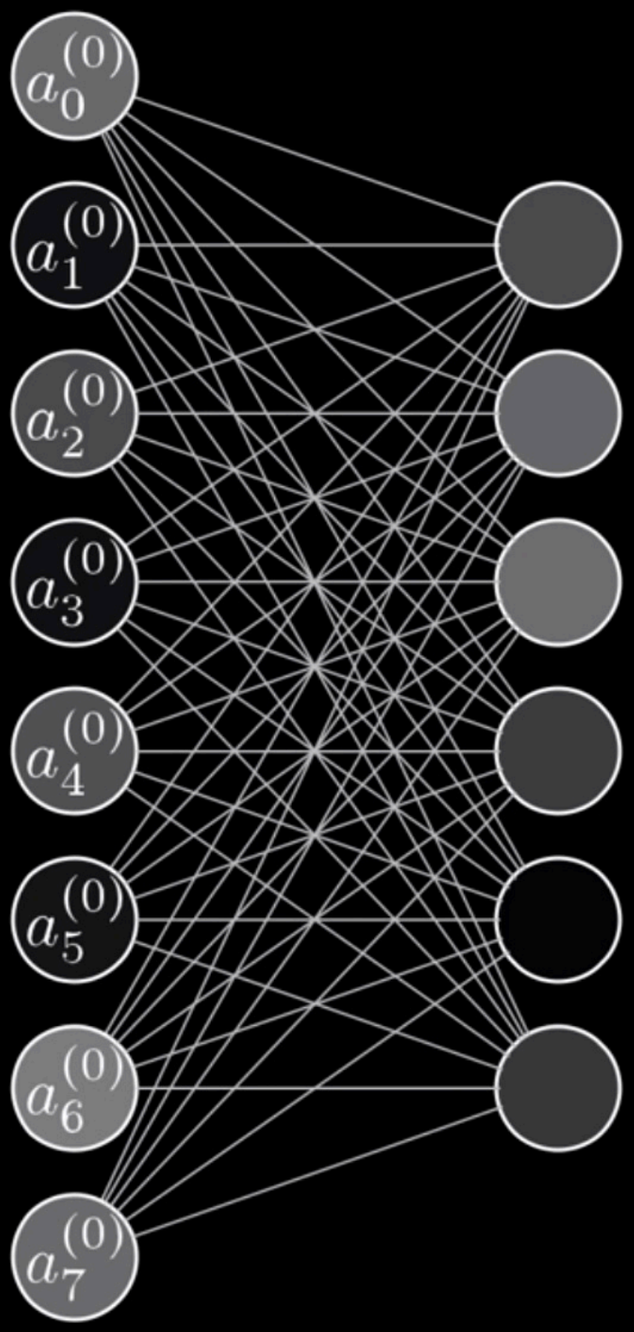
Matrix



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

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

Feedforward function



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

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

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

END

Thank you