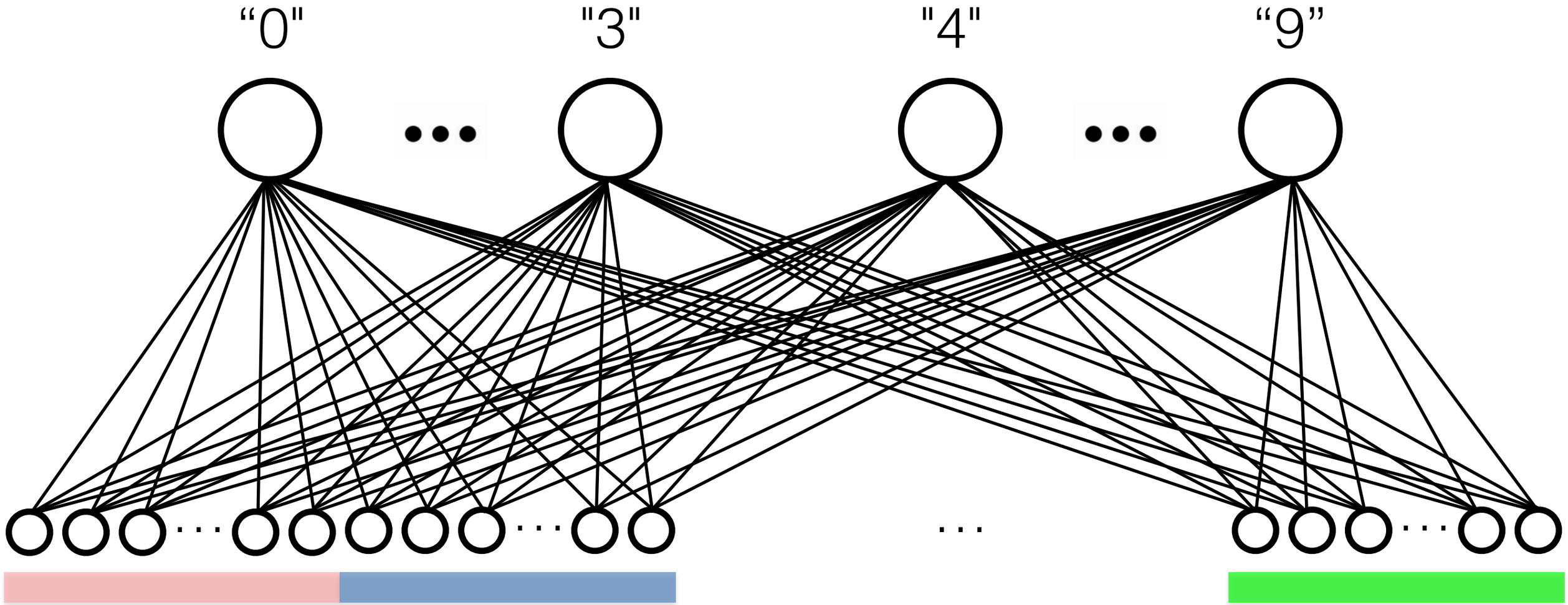
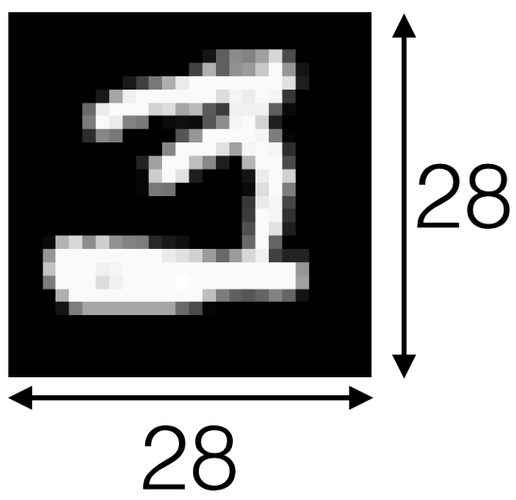
on Brightspace

Homework 3 (on Brightspace, in today's slides)
Due Thurs Sep 22
start with `Homework3.ipynb` on Brightspace

Questions about Homework 3?

`Homework3.pdf`
`Homework3.ipynb`


“0"
"3"
"4"
“9”


28
28

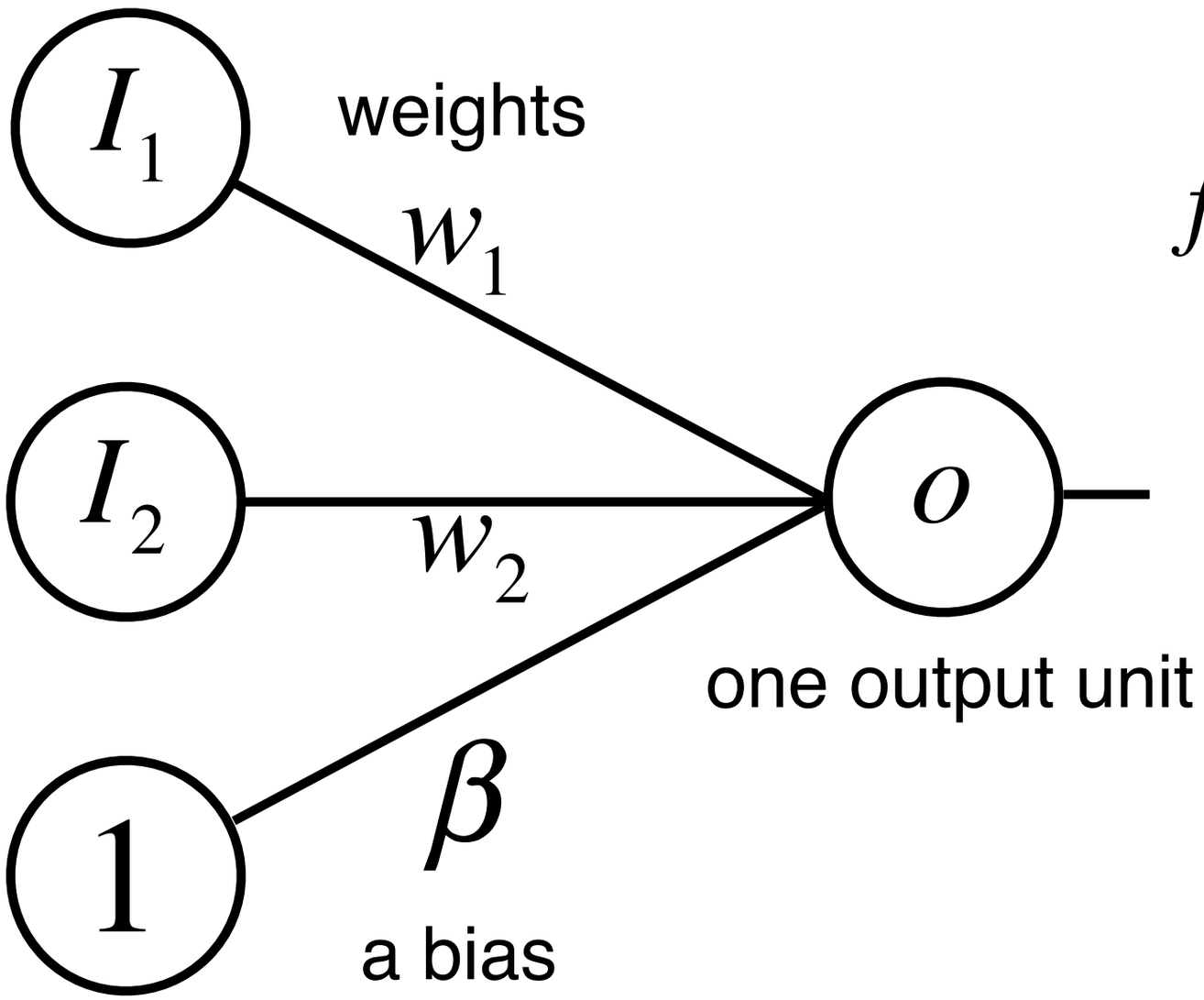
Computations by the Simplest Neural Network

computations by the simplest neural network

activation function

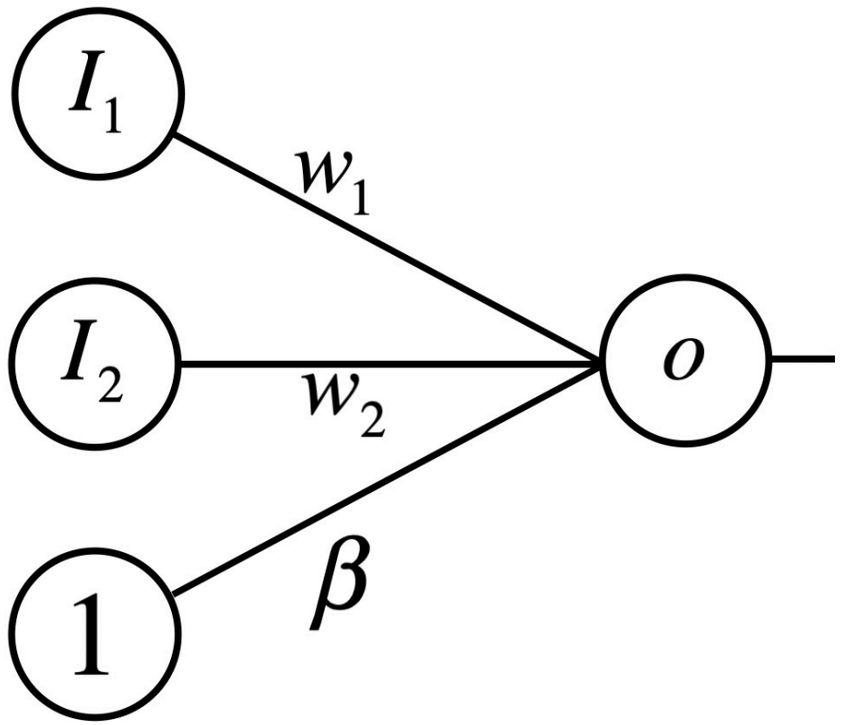
$$o = f\left(\sum_i I_i w_i + \beta\right)$$

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

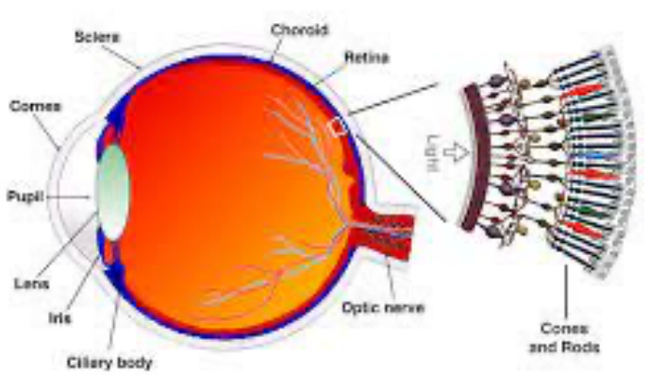


computations by the simplest neural network

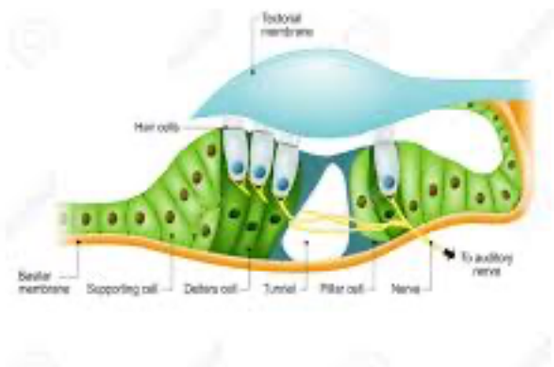
what are the inputs?



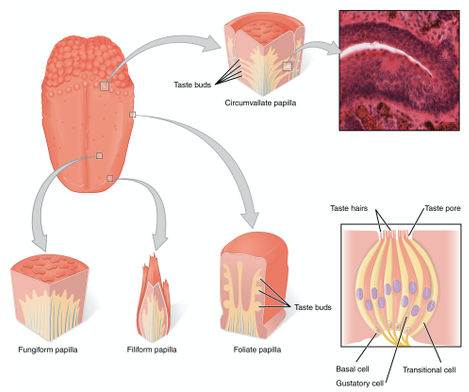
in a real brain, they'd be sensory receptors



rods and cones
retina



hair cells
cochlea

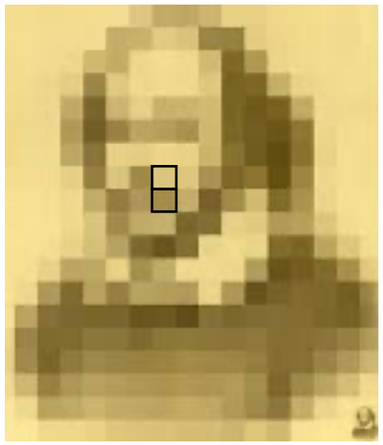


taste receptors
tongue

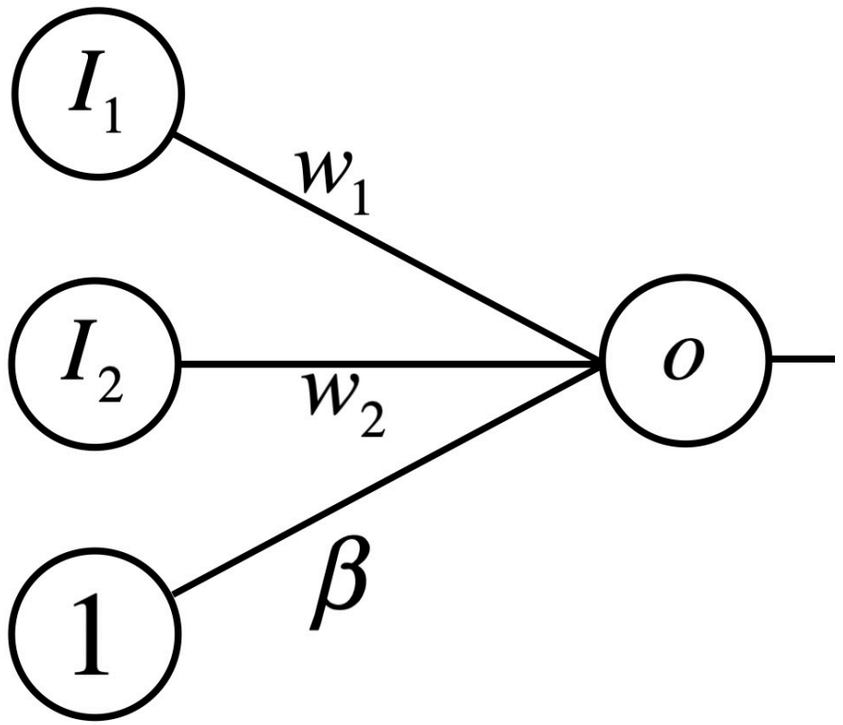
computations by the simplest neural network

what are the inputs?

pixelated image



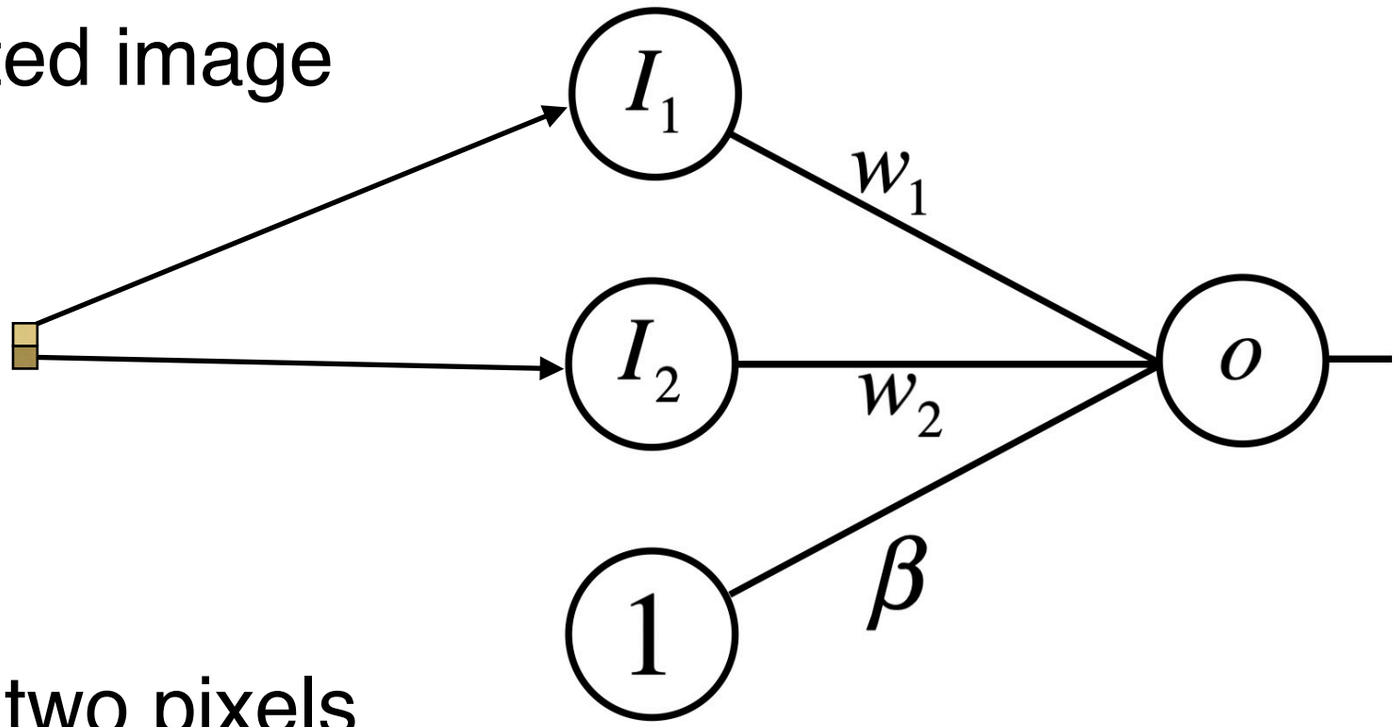
of only two pixels



computations by the simplest neural network

what are the inputs?

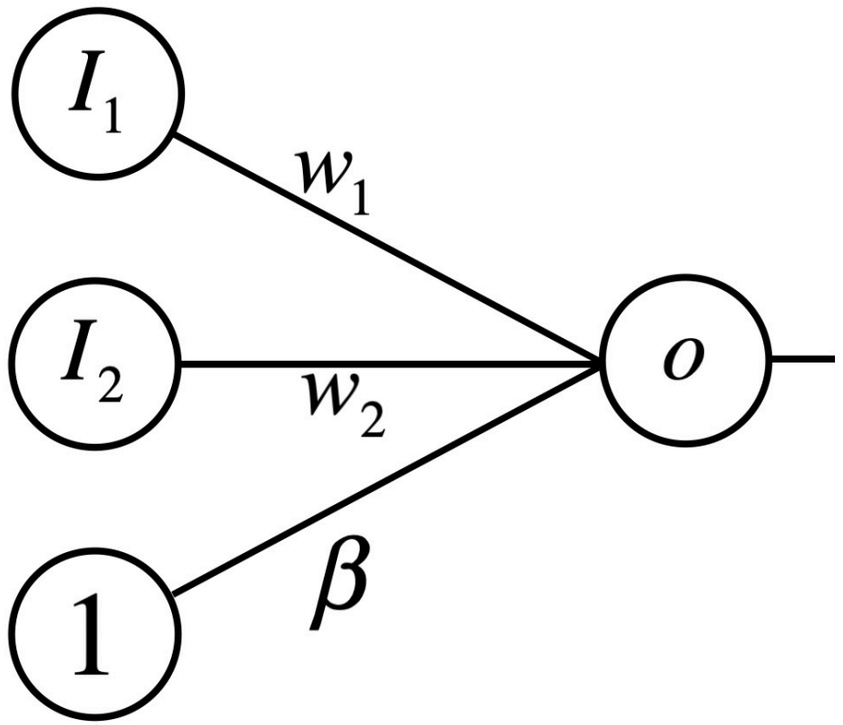
pixelated image



of only two pixels

computations by the simplest neural network

what are the inputs?

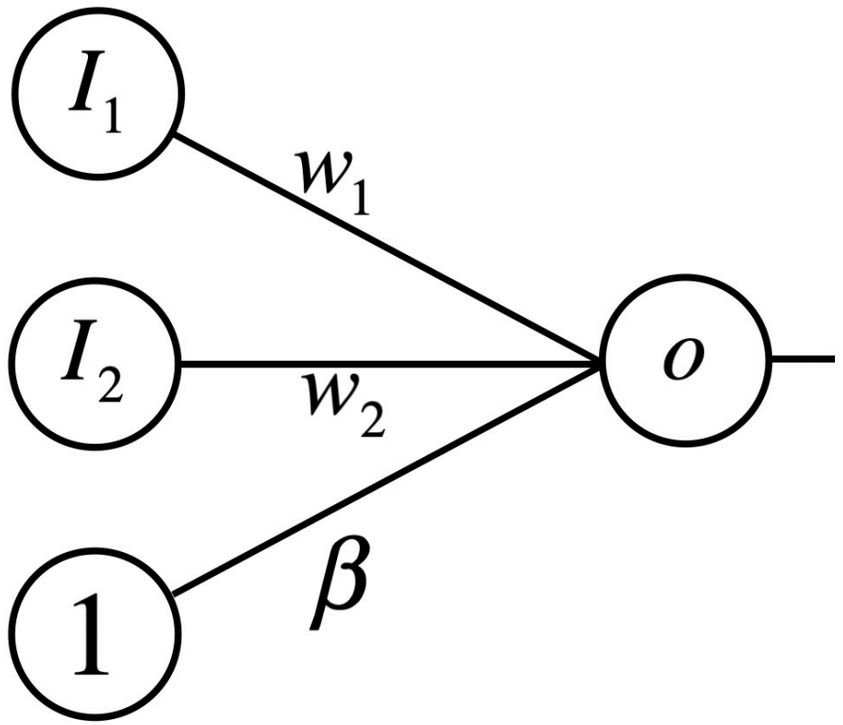


in a data science application, each input unit could represent:

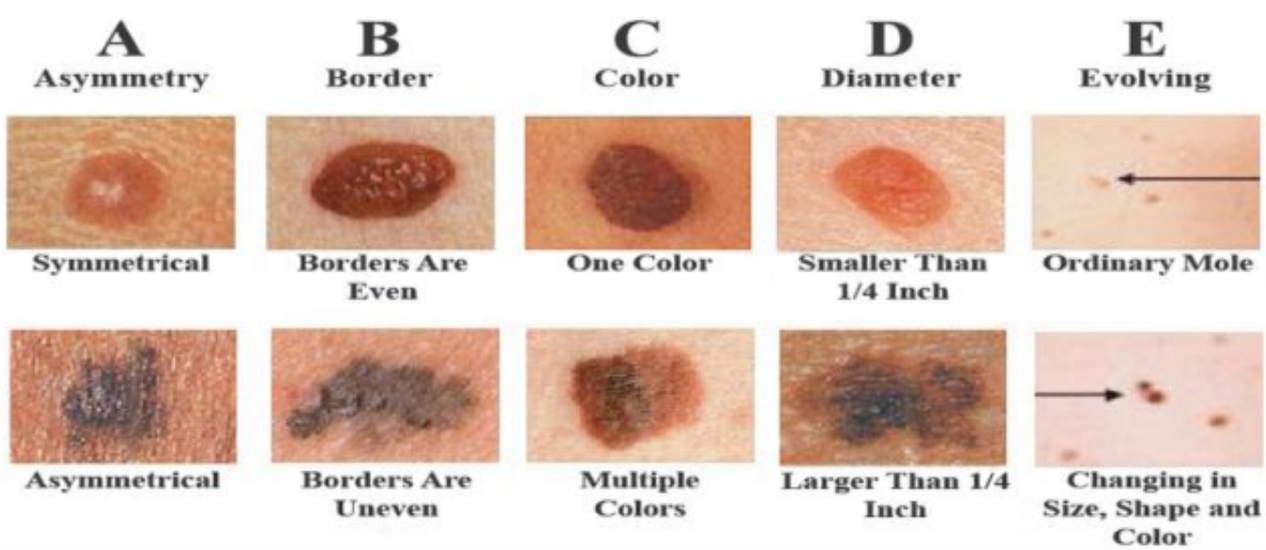
- presence or absence of a particular word in a text (0 or 1)
- the measure of a particular market indicator (continuous value)
- whether someone watched a particular movie or not (0 or 1)
- their rating of each movie they watched (continuous value)
- the activity of a particular voxel in an fMRI scan (using neural networ
  as data science tools to analyze neural data)

computations by the simplest neural network

what are the inputs?



now, let's keep things more abstract

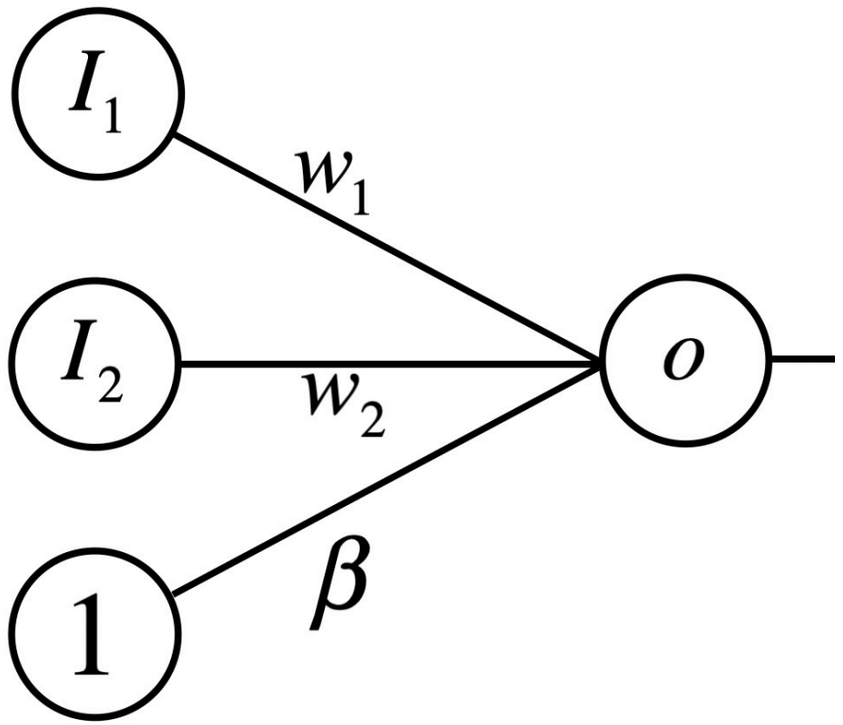


computations by the simplest neural network

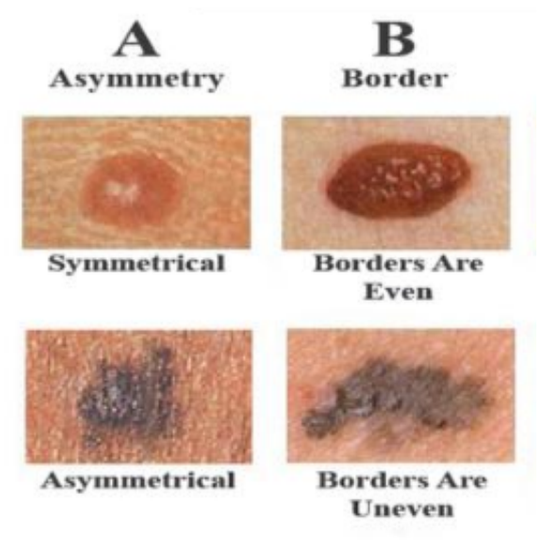
what are the inputs?

asymmetry
1=yes, 0=no

uneven border
1=yes, 0=no



we're not modeling how "asymmetry" or "border type" are recognized from an image (in the middle of cognitive processing)

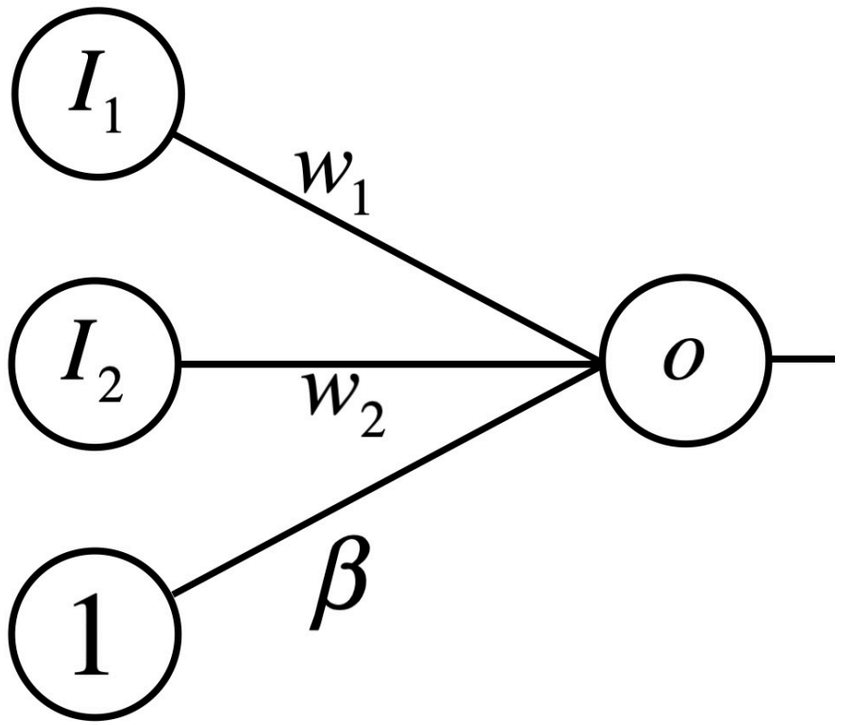


computations by the simplest neural network

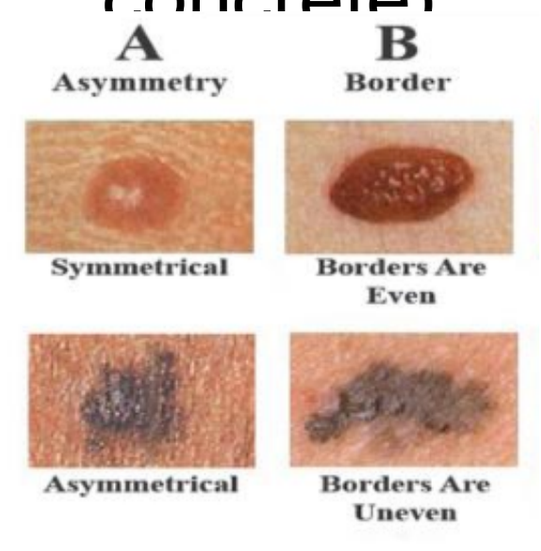
what are the inputs?

asymmetry
1=yes, 0=no

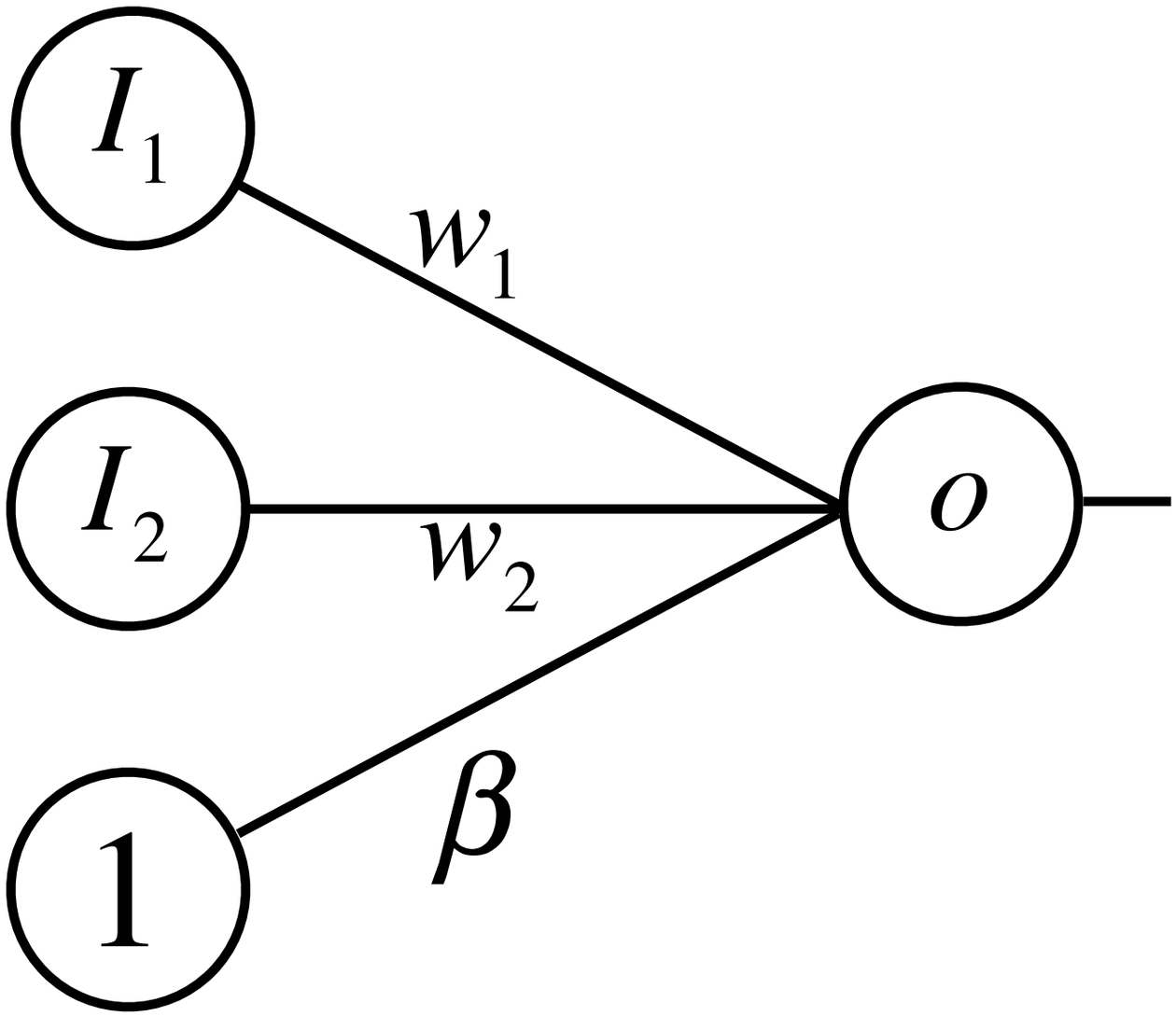
uneven border
1=yes, 0=no



and nothing about any of these examples have anything to do with real skin cancer diagnosis (just using these to make things concrete)



computations by the simplest neural network

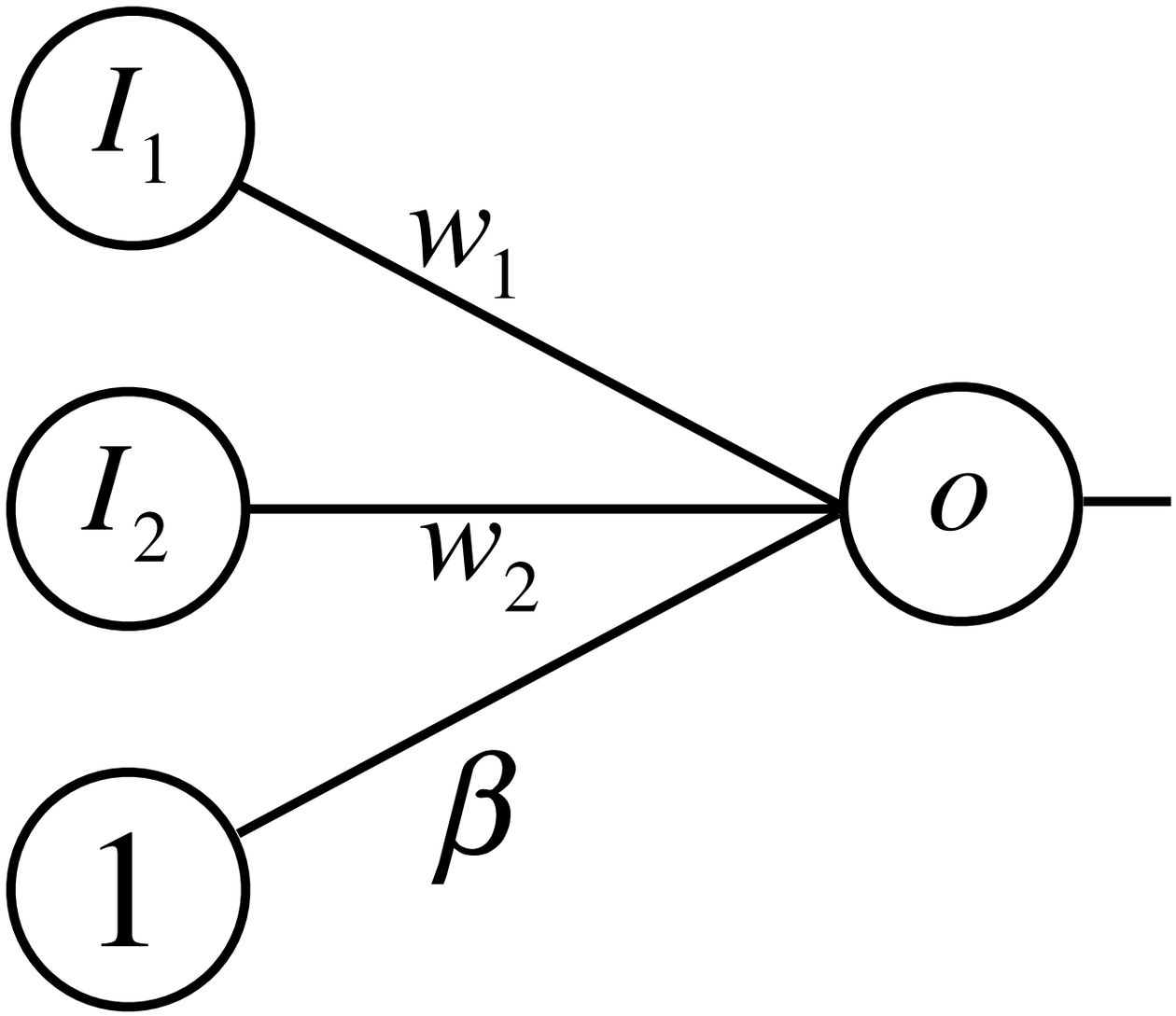


asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	0
0	1	0
1	1	1

what is this
logical operation?

(if 1=True and 0=False)

computations by the simplest neural network



asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	0
0	1	0
1	1	1

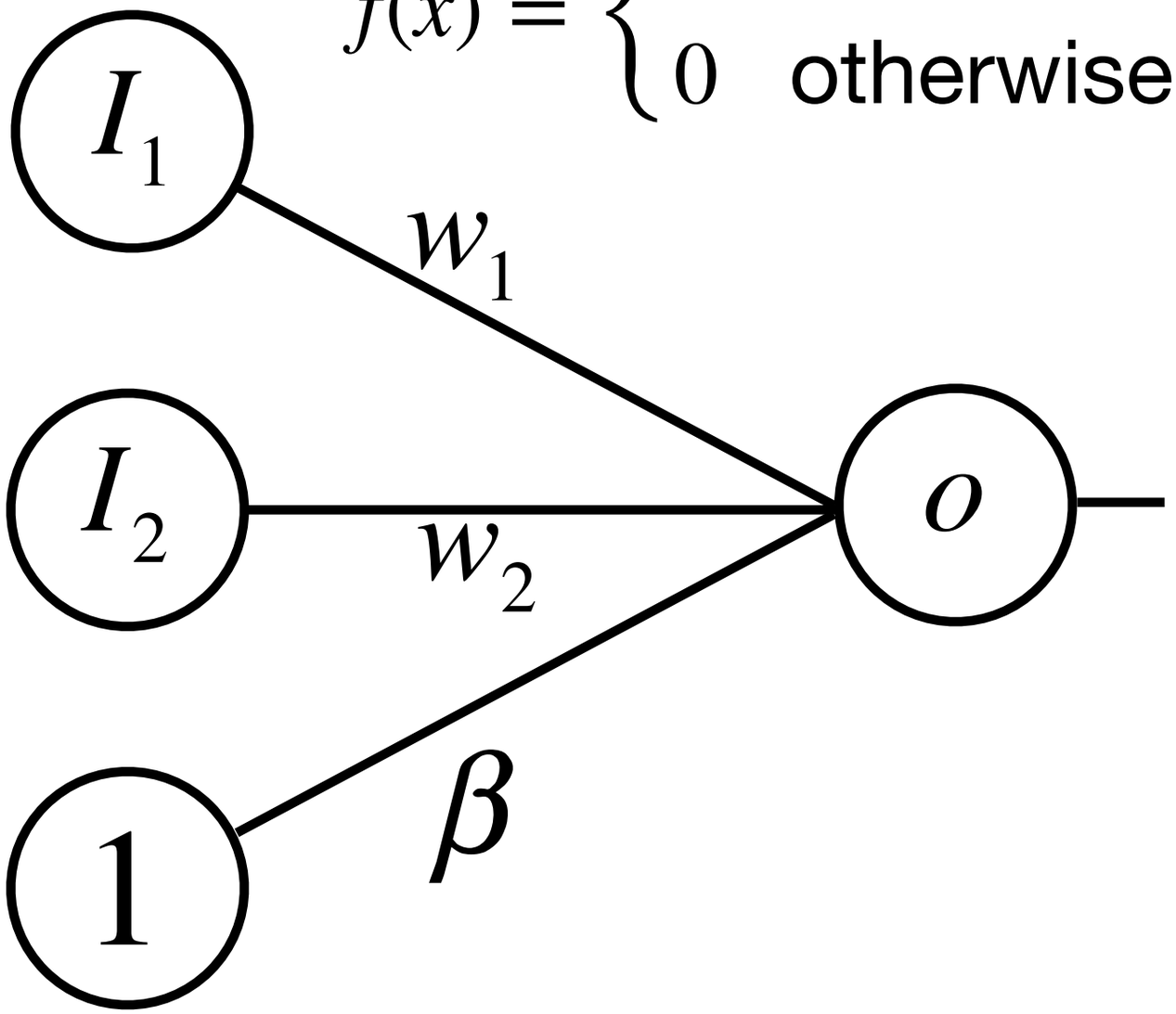
logical **AND**

computations by the simplest neural network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	0
0	1	0
1	1	1

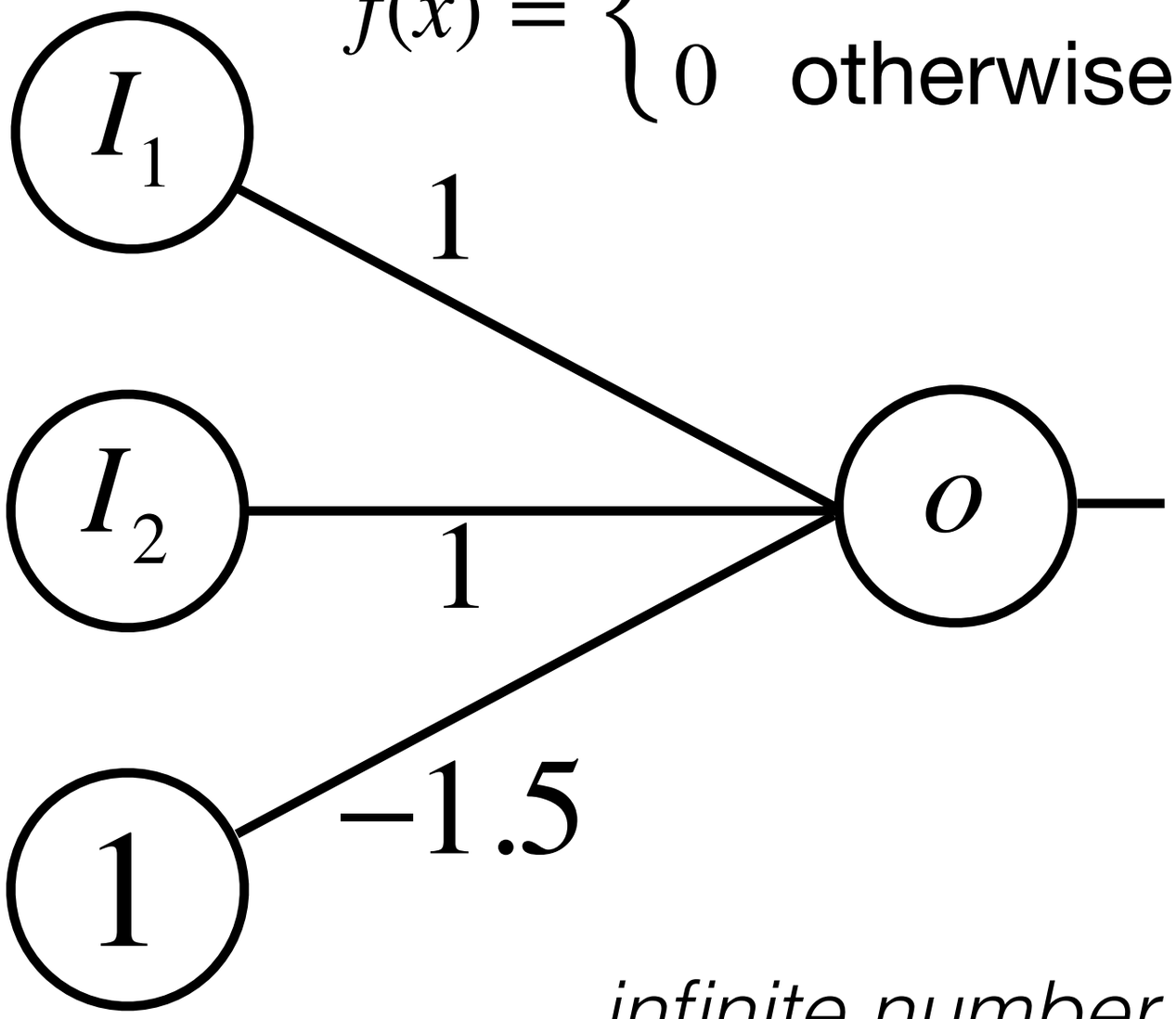
logical **AND**

computations by the simplest neural network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	0
0	1	0
1	1	1

logical **AND**

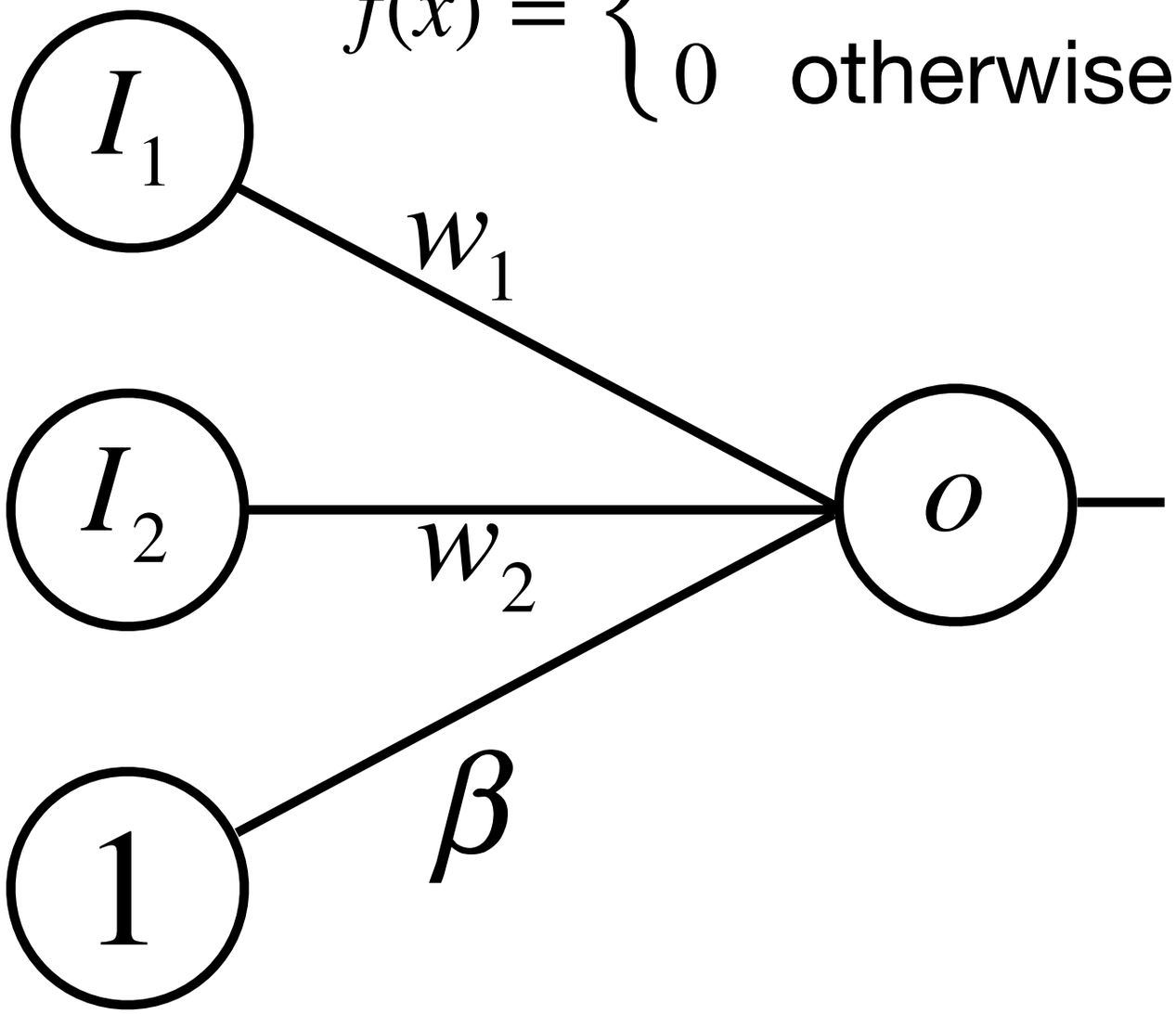
infinite number of solutions

computations by the simplest neural network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	1
0	1	1
1	1	1

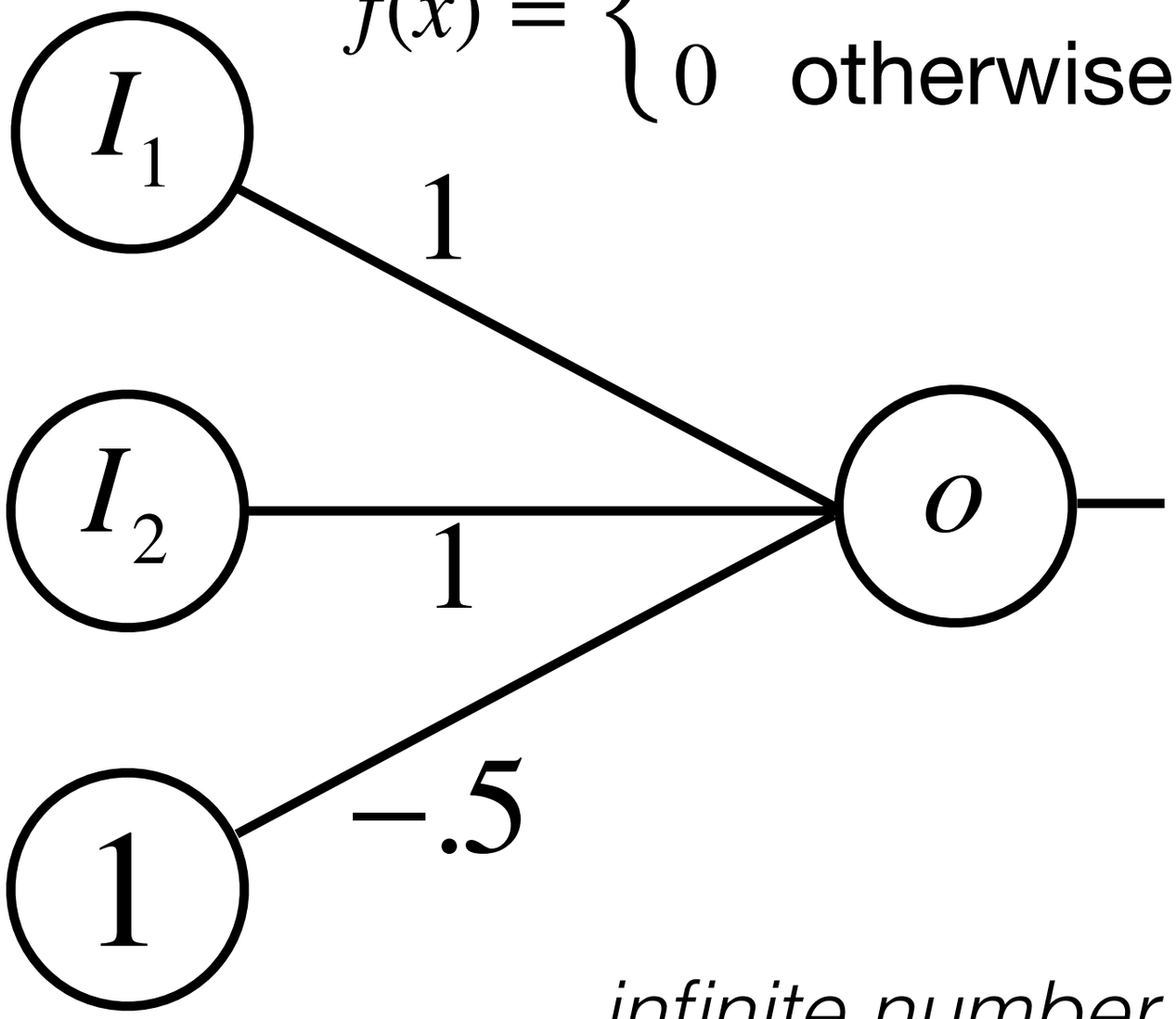
what is this logical operation?

computations by the simplest neural network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	1
0	1	1
1	1	1

logical **OR**

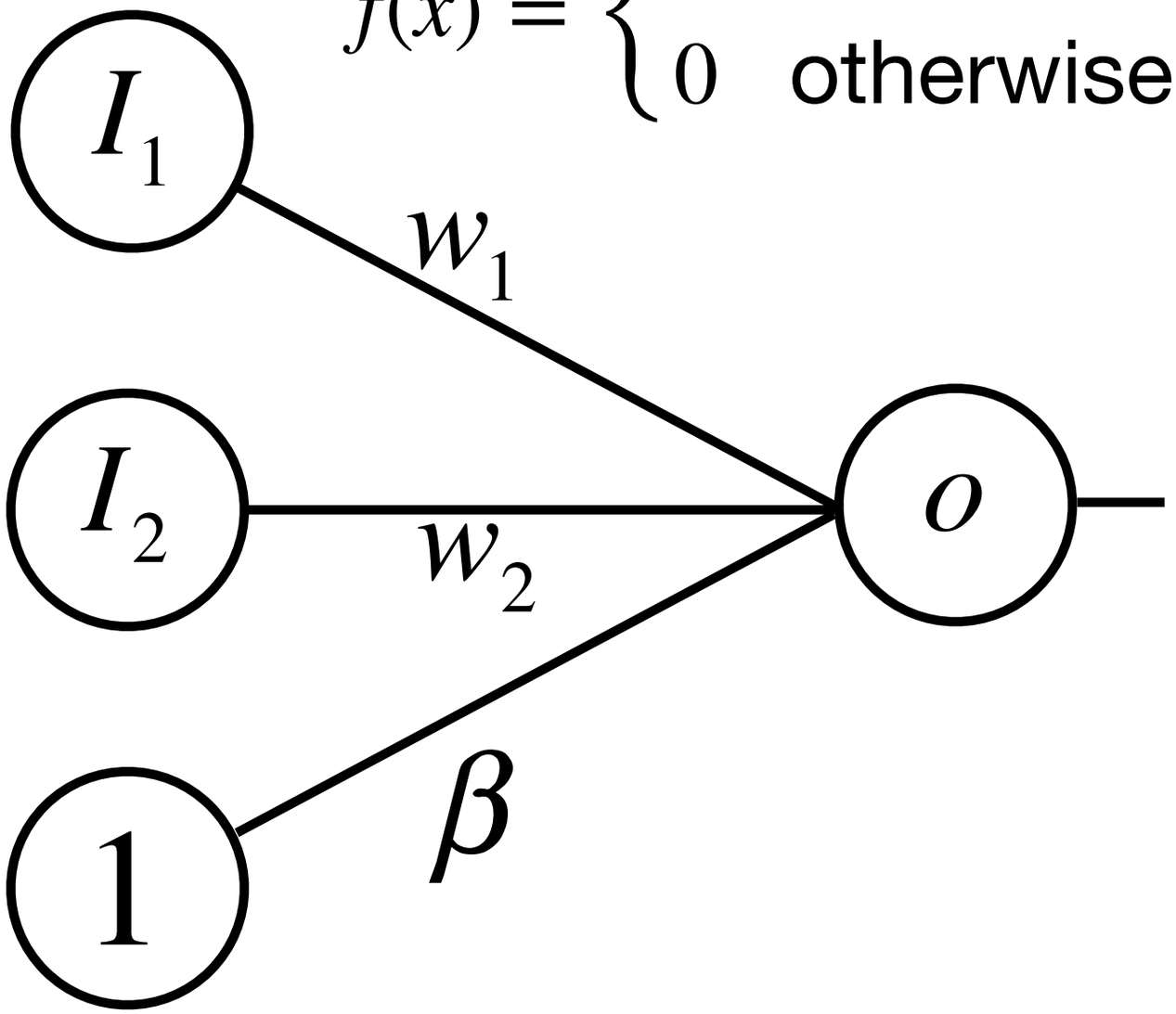
infinite number of solutions

computations by the simplest neural network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	1
0	1	1
1	1	0

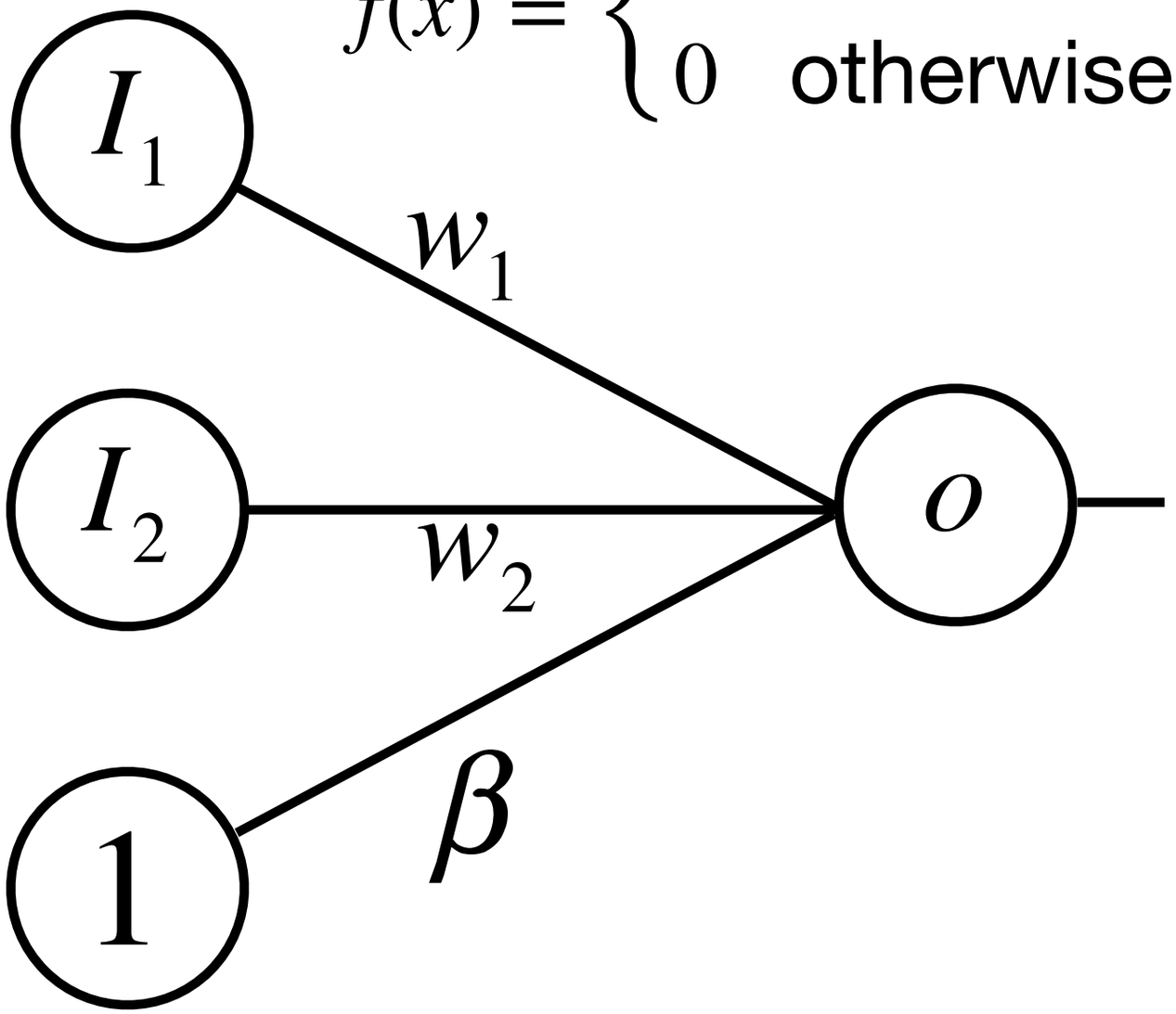
what is this logical operation?

computations by the simplest neural network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	1
0	1	1
1	1	0

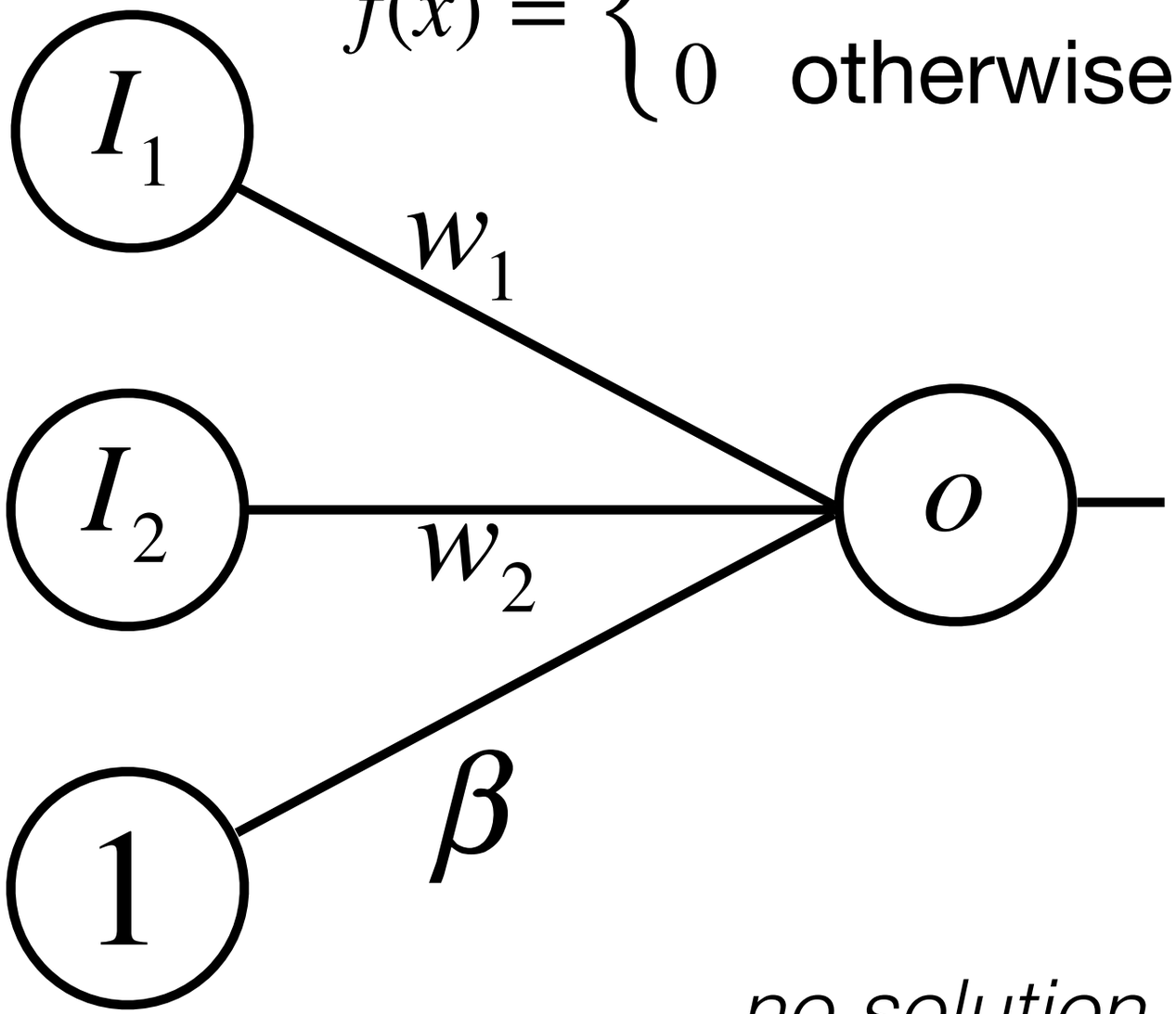
logical **XOR**
(exclusive OR)

computations by the simplest neural network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

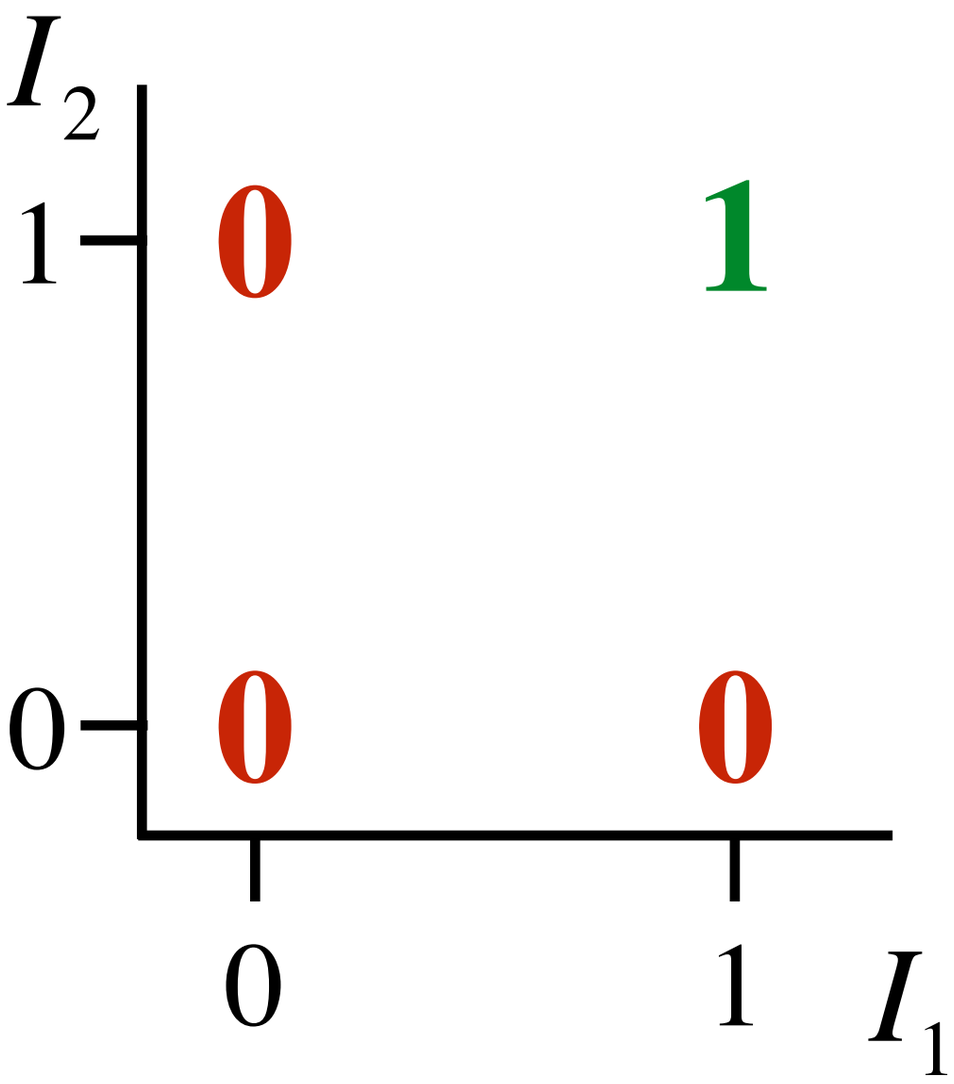


asymmetry 1=yes 0=no I_1	uneven 1=yes 0=no I_2	cancer 1=yes 0=no o
0	0	0
1	0	1
0	1	1
1	1	0

logical **XOR**
(exclusive OR)

no solution exists!!!

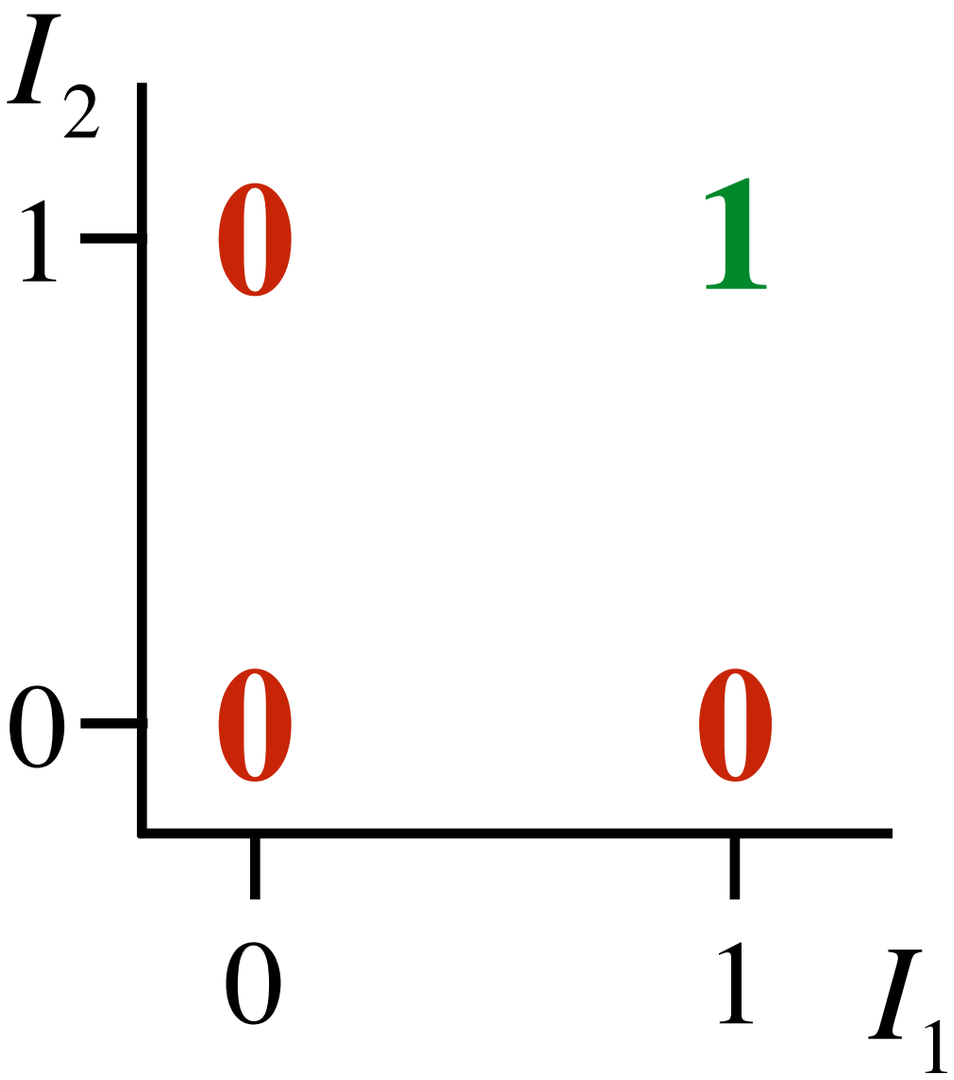
computations by the simplest neural network

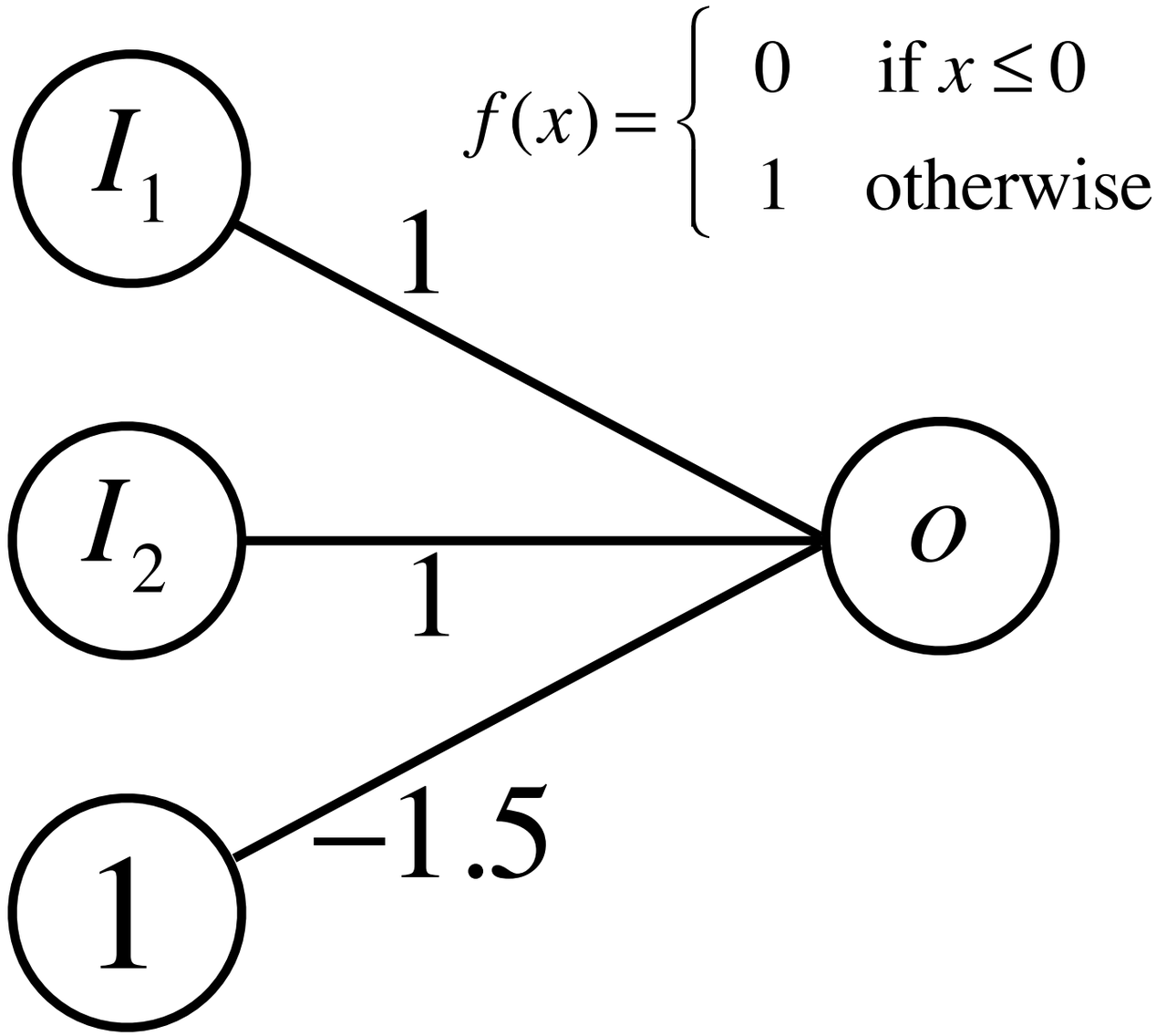


I_1	I_2	o
0	0	**0**
1	0	**0**
0	1	**0**
1	1	**1**

logical
AND

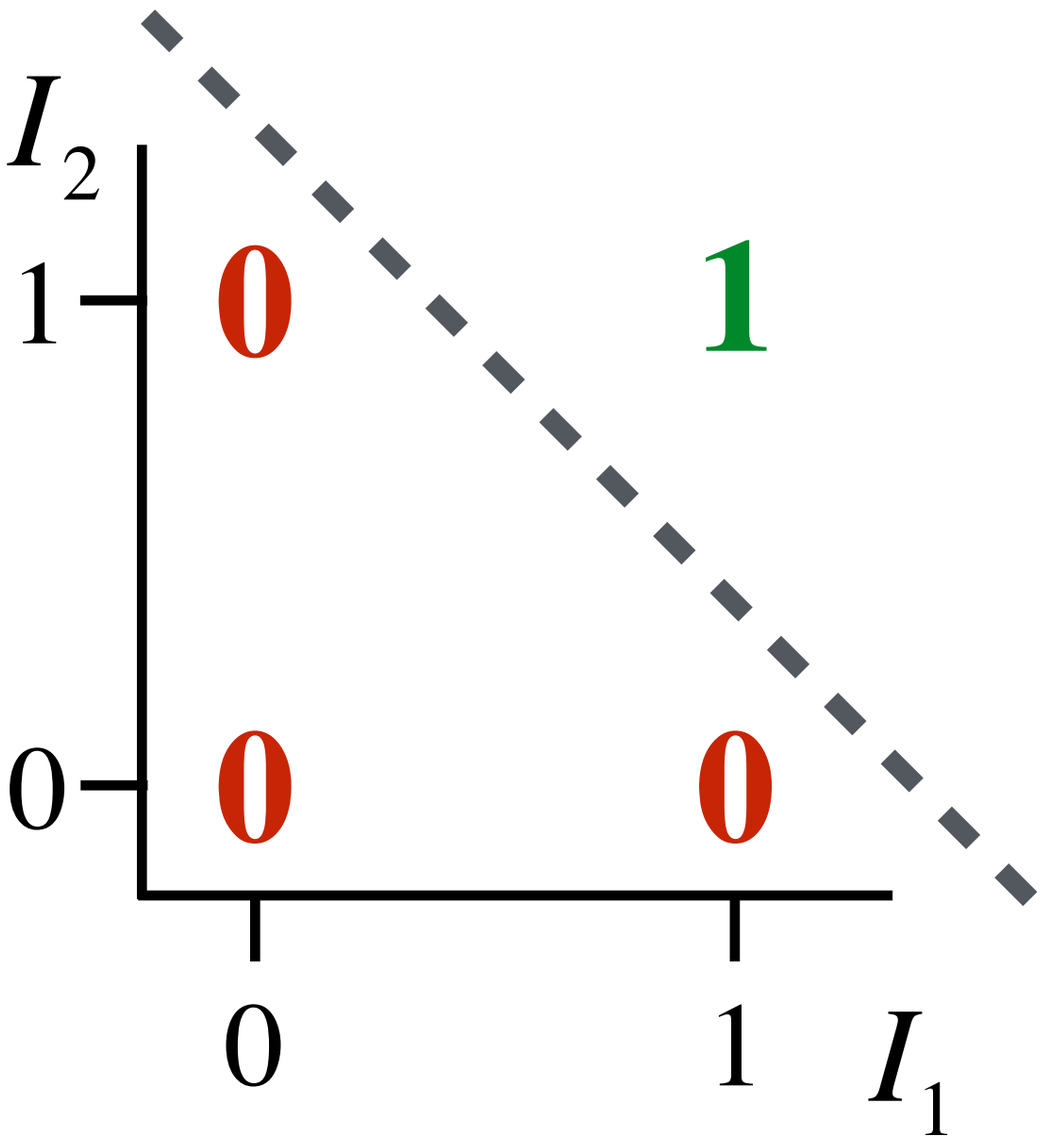
computations by the simplest neural network

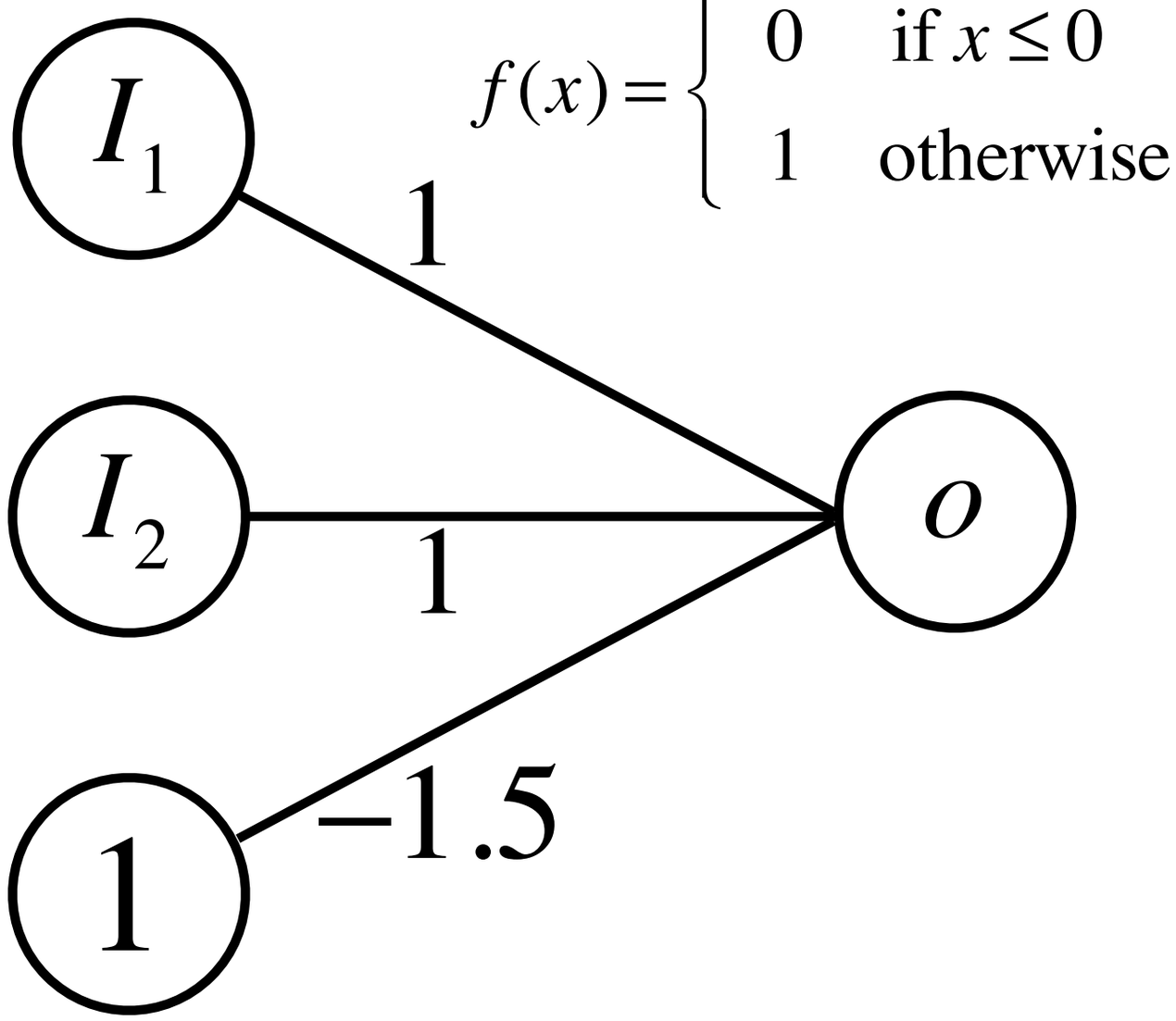




$$1 \times I_1 + 1 \times I_2 + (-1.5) \times 1 = 0$$

computations by the simplest neural network

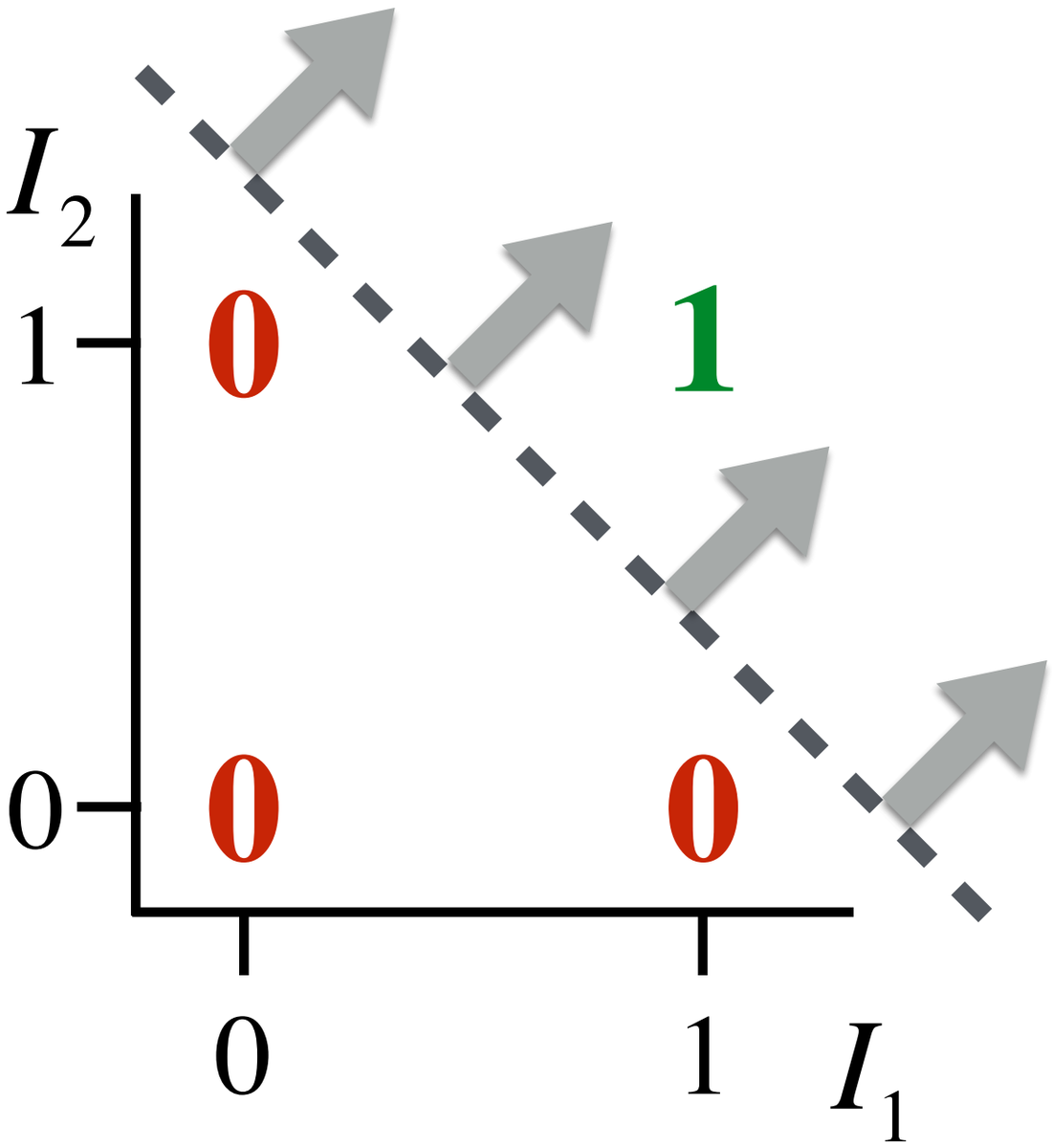


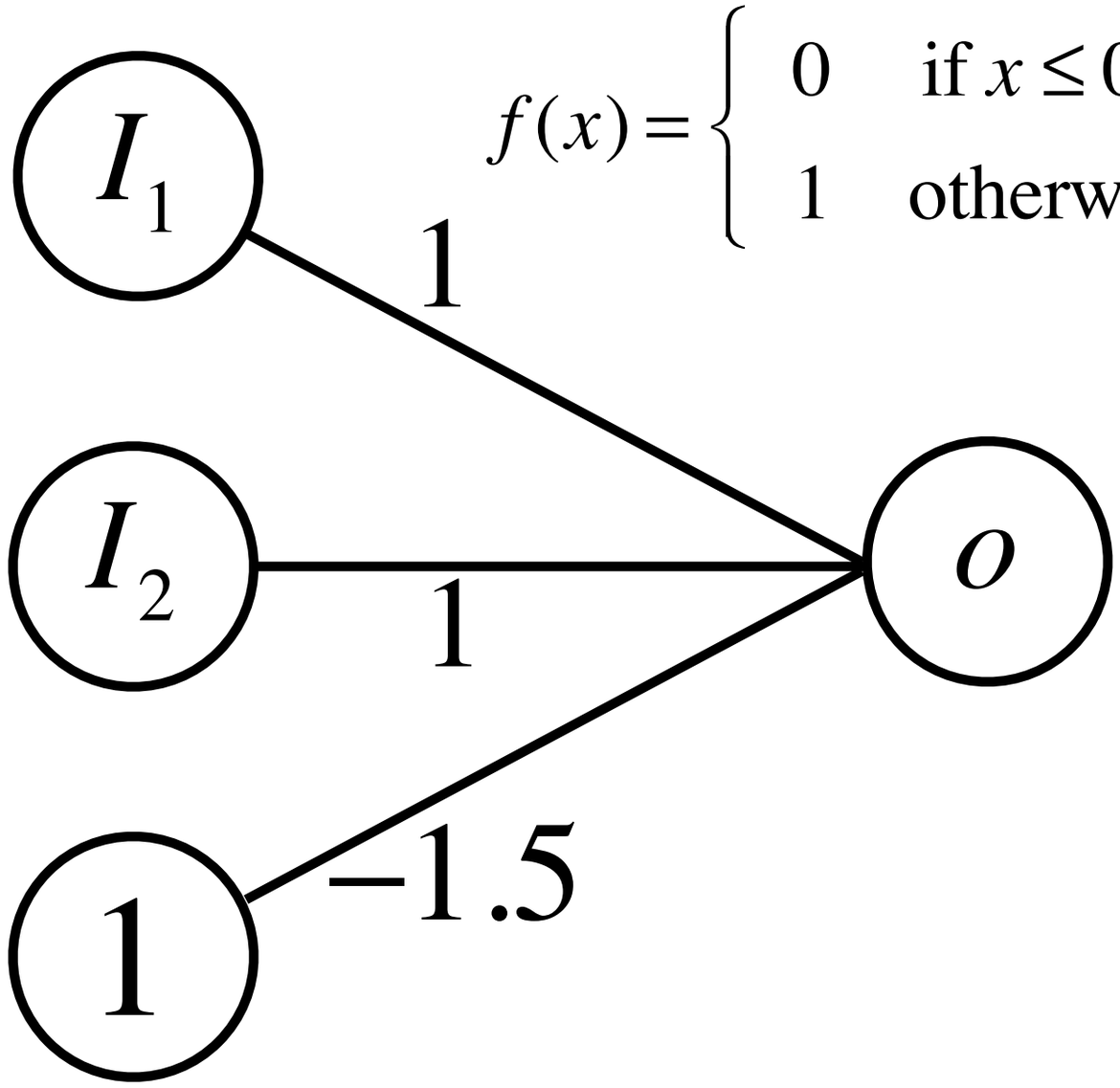


$$I_1 + I_2 - 1.5 = 0$$

$$I_2 = -I_1 + 1.5$$

computations by the simplest neural network



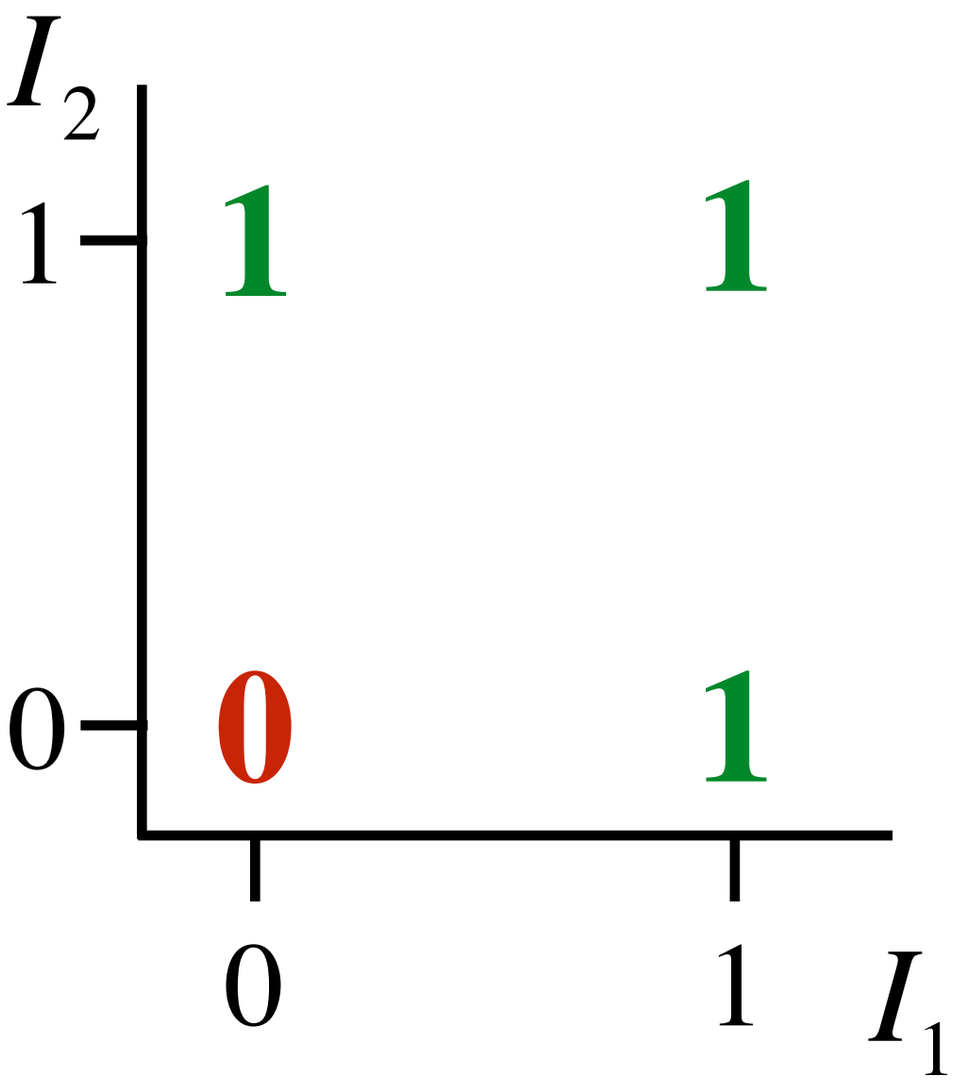


$$f(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{otherwise} \end{cases}$$

$$I_1 + I_2 - 1.5 = 0$$

$$I_2 = -I_1 + 1.5$$

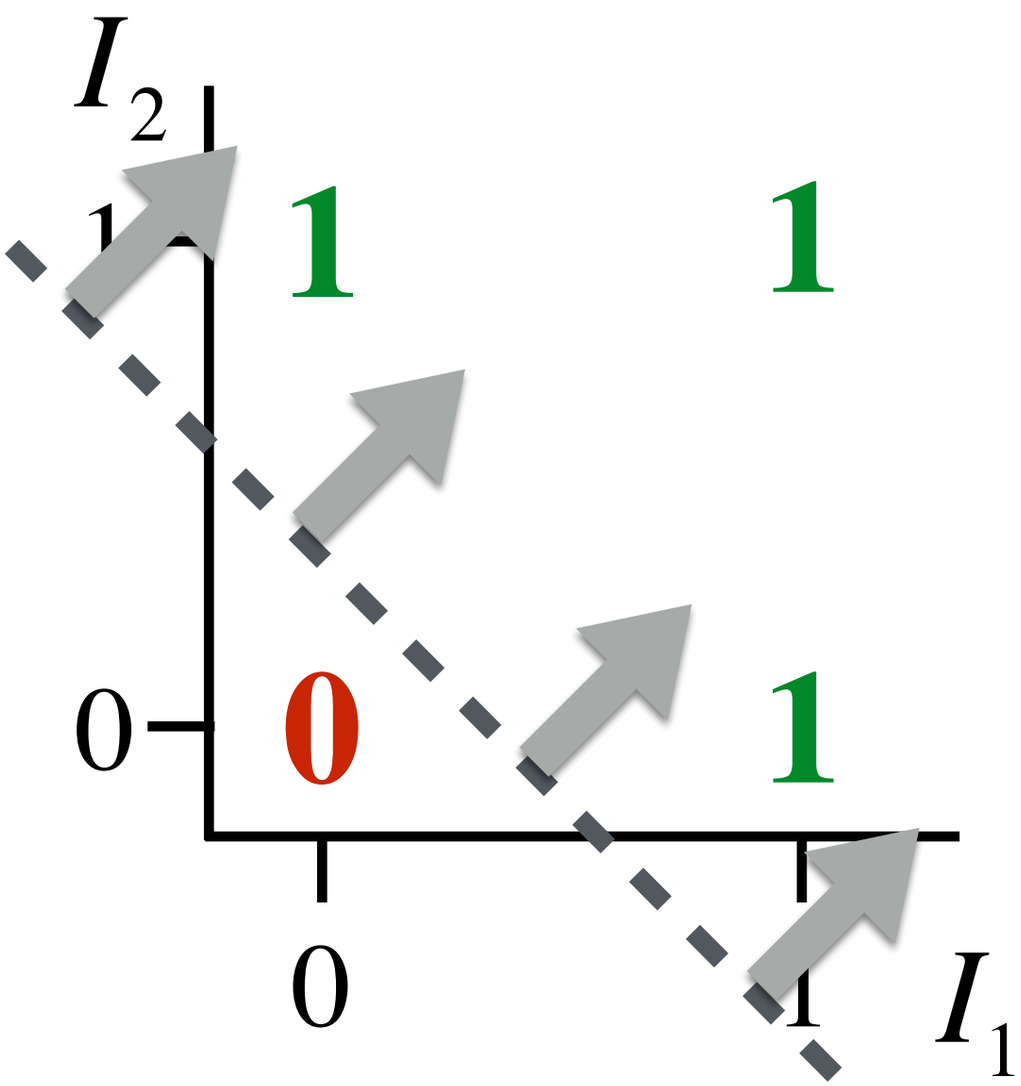
computations by the simplest neural network

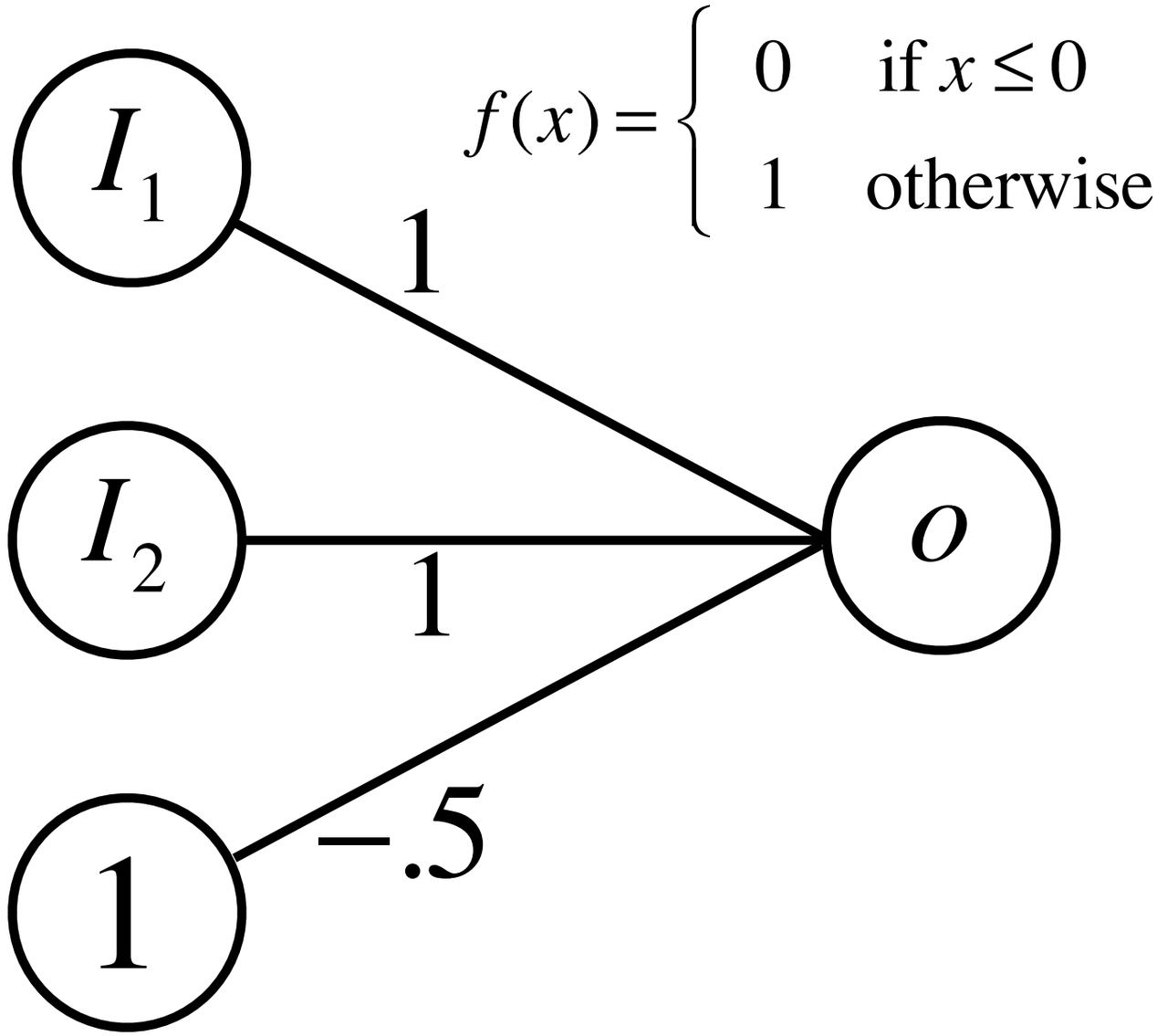


I_1	I_2	o
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**1**

logical
OR

computations by the simplest neural network

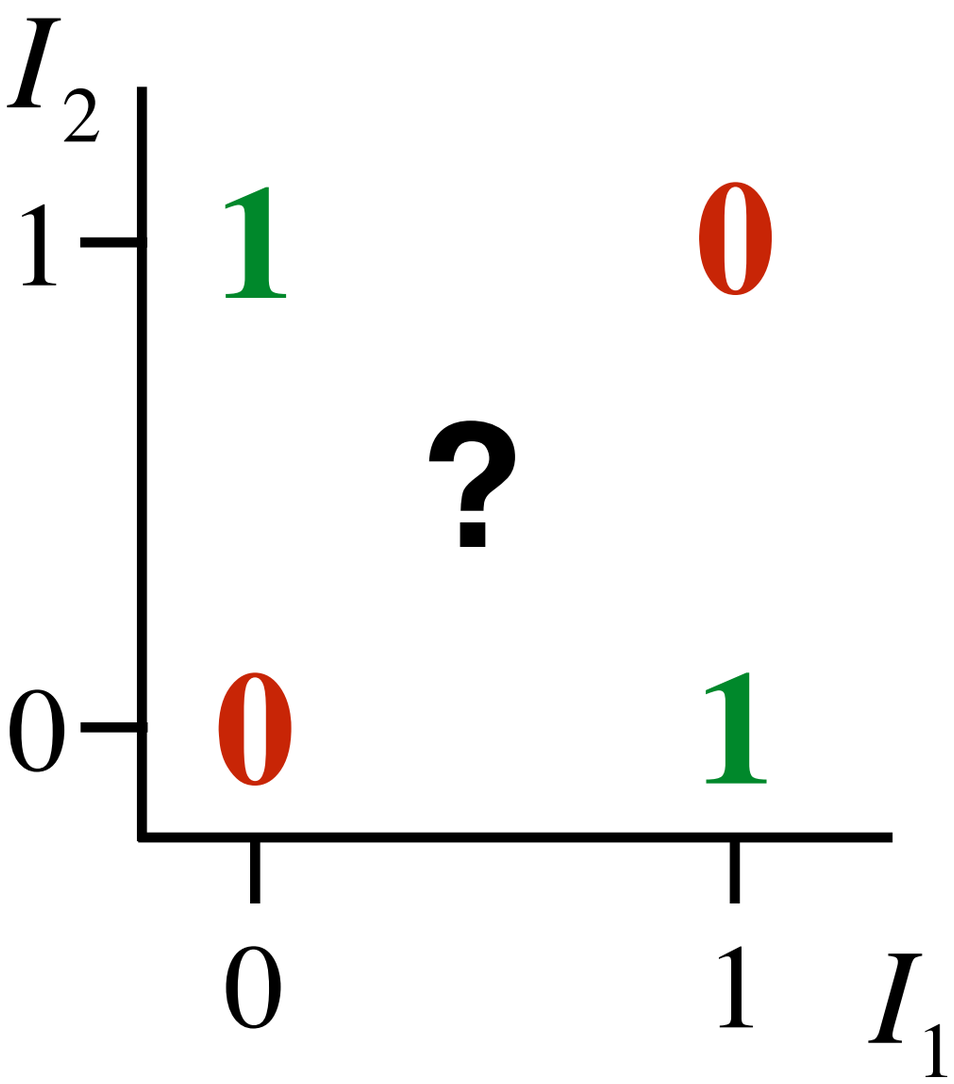




$$1 \times I_1 + 1 \times I_2 - .5 = 0$$

$$I_2 = -I_1 + .5$$

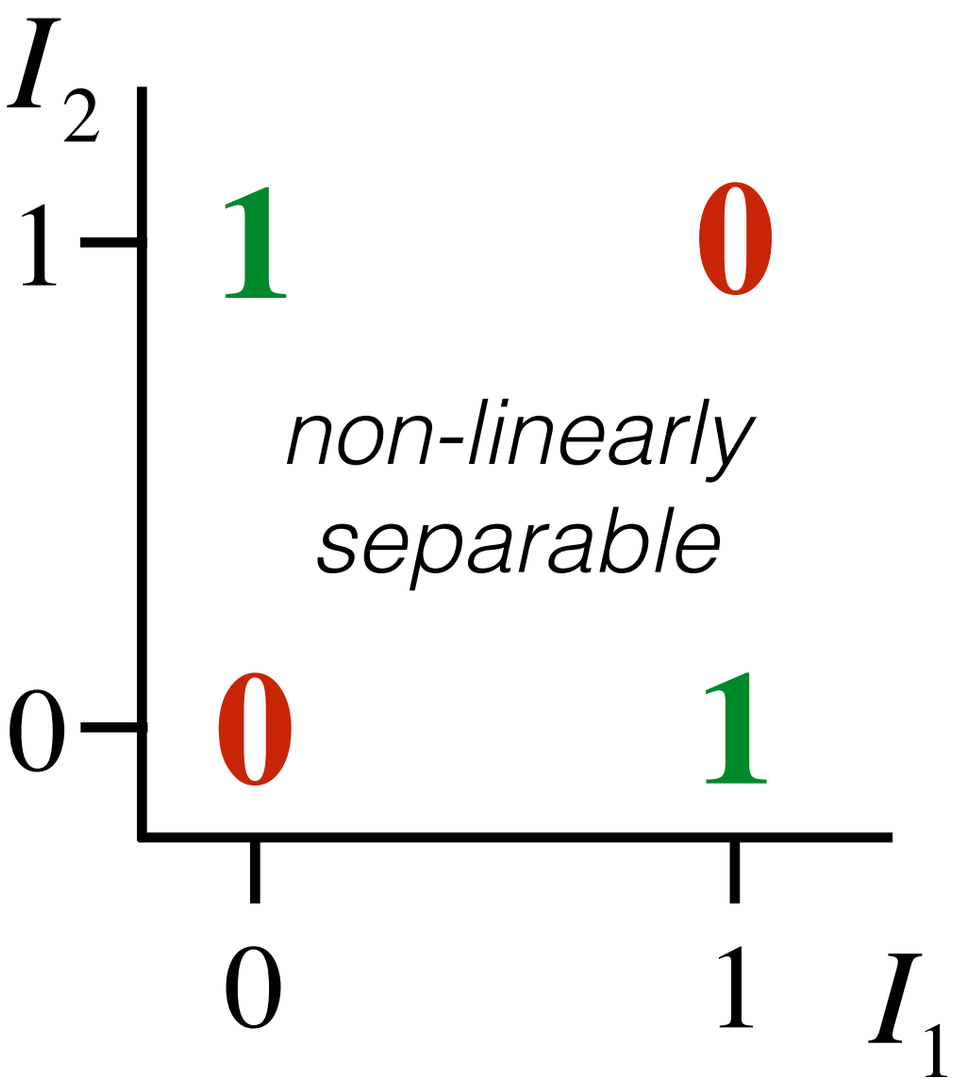
computations by the simplest neural network



I_1	I_2	o
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**0**

logical XOR

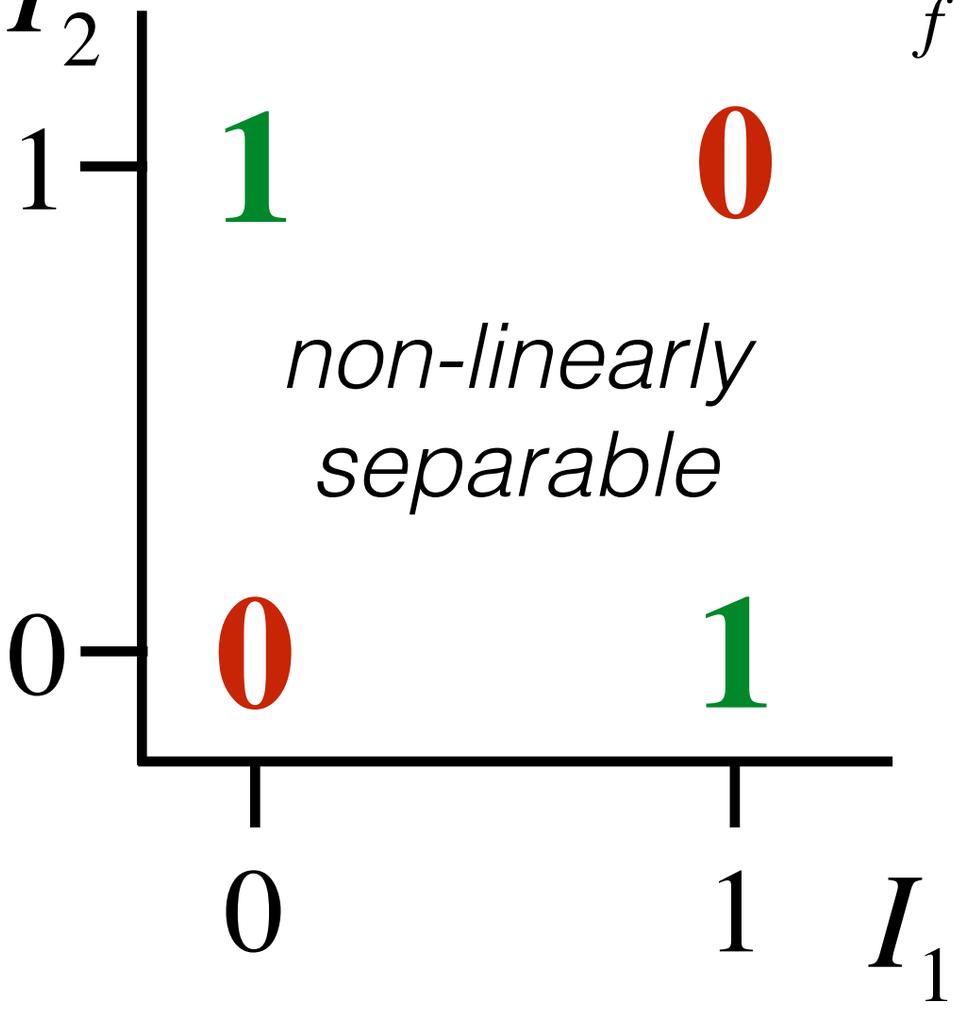
computations by the simplest neural network



I_1	I_2	o
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**0**

logical
XOR

computations by the simplest neural network



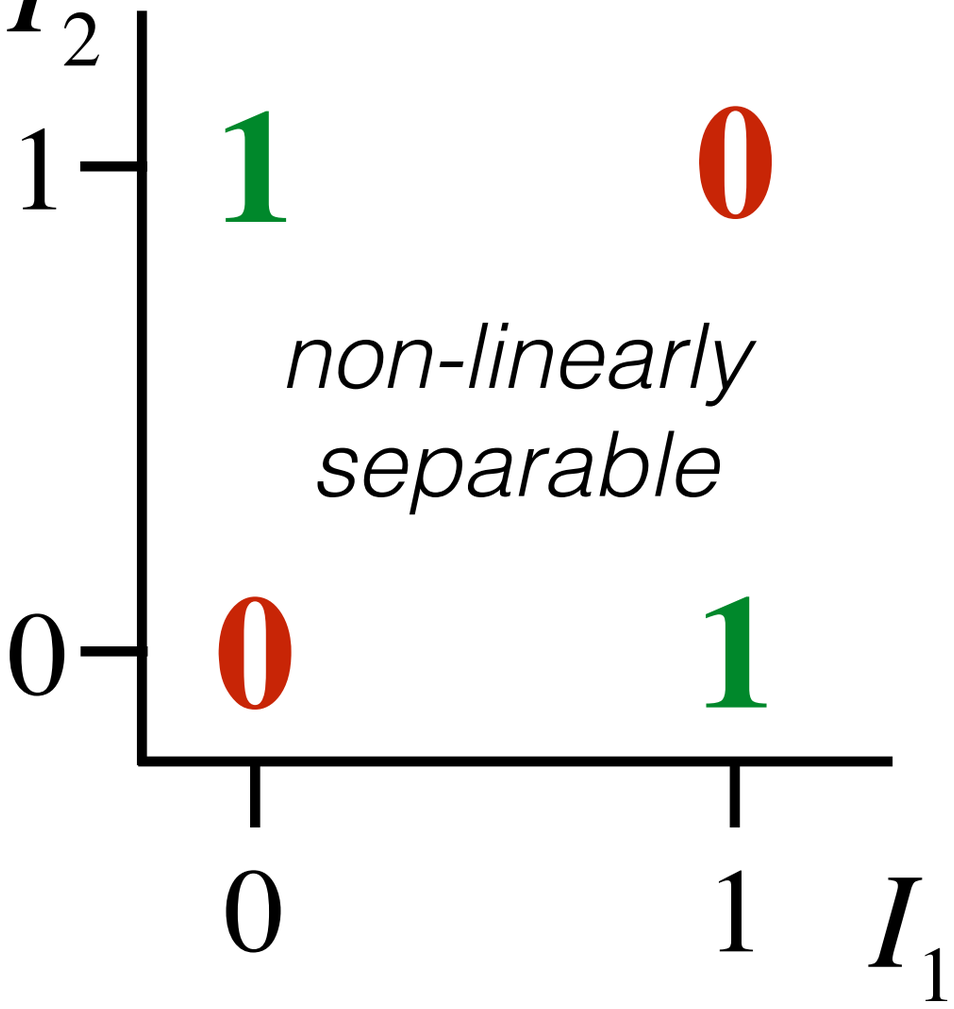
assuming a step function

$$f(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{otherwise} \end{cases}$$

I_1	I_2	o
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**0**

logical
XOR

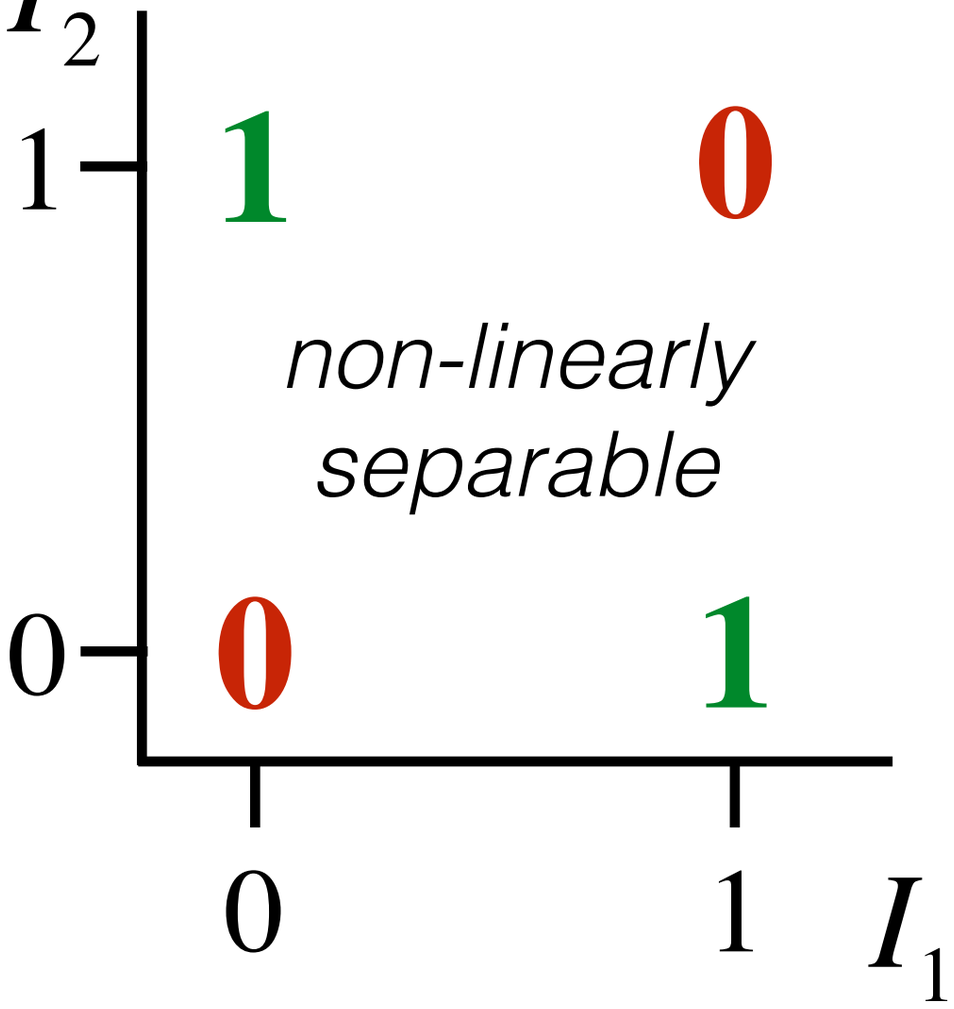
computations by the simplest neural network



I_1	I_2	o
$w_1 \times 0 + w_2 \times 0 + \beta < 0$		
1	0	**1**
0	1	**1**
1	1	**0**

logical
XOR

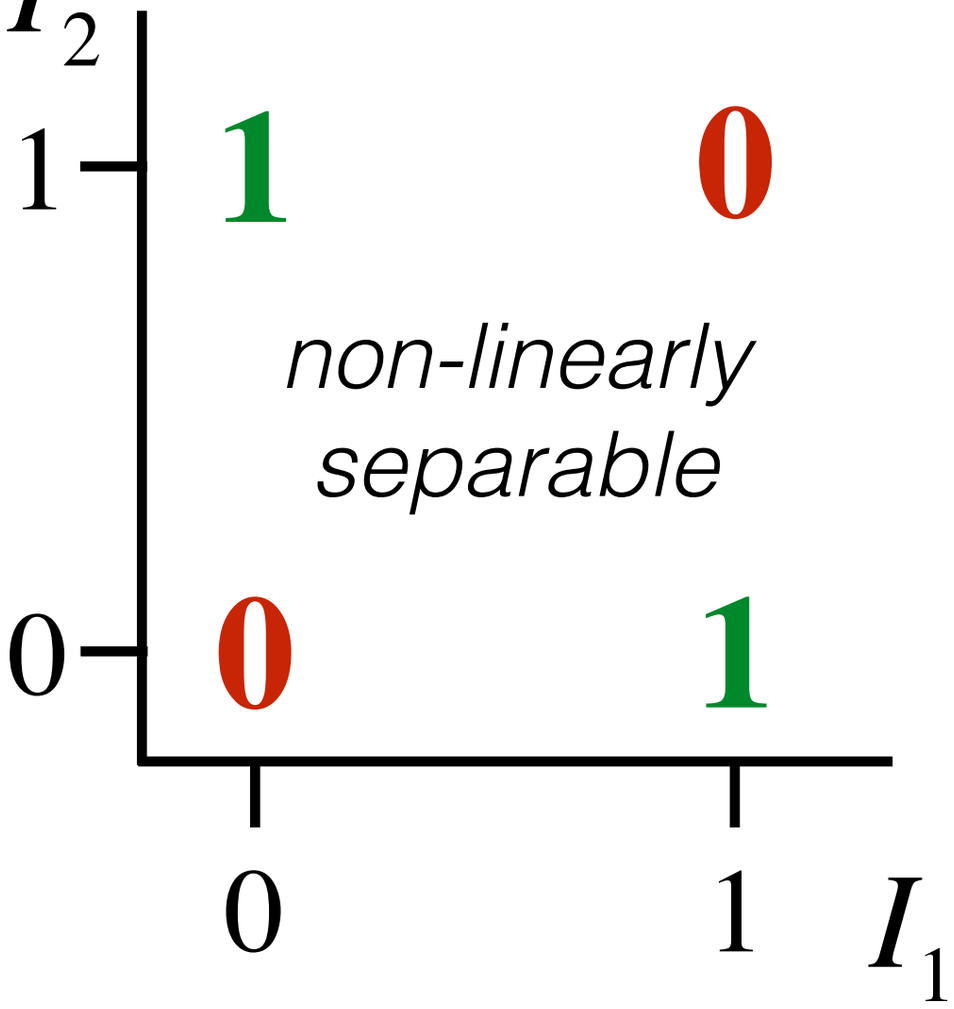
computations by the simplest neural network



I_1	I_2	o
$w_1 \times 0 + w_2 \times 0 + \beta < 0$		
$w_1 \times 1 + w_2 \times 0 + \beta \geq 0$		
0	1	**1**
1	1	**0**

logical
XOR

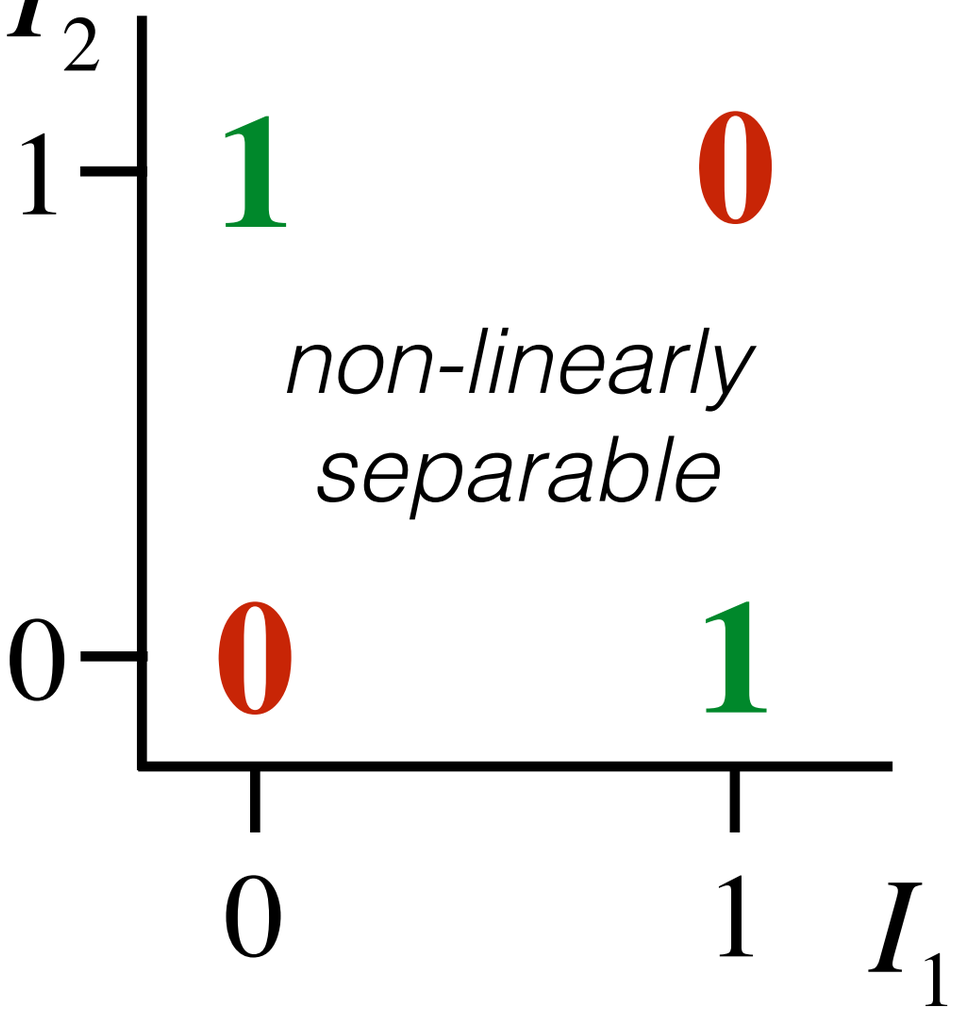
computations by the simplest neural network



I_1	I_2	o
		$w_1 \times 0 + w_2 \times 0 + \beta < 0$
		$w_1 \times 1 + w_2 \times 0 + \beta \geq 0$
		$w_1 \times 0 + w_2 \times 1 + \beta \geq 0$
1	1	0

logical
XOR

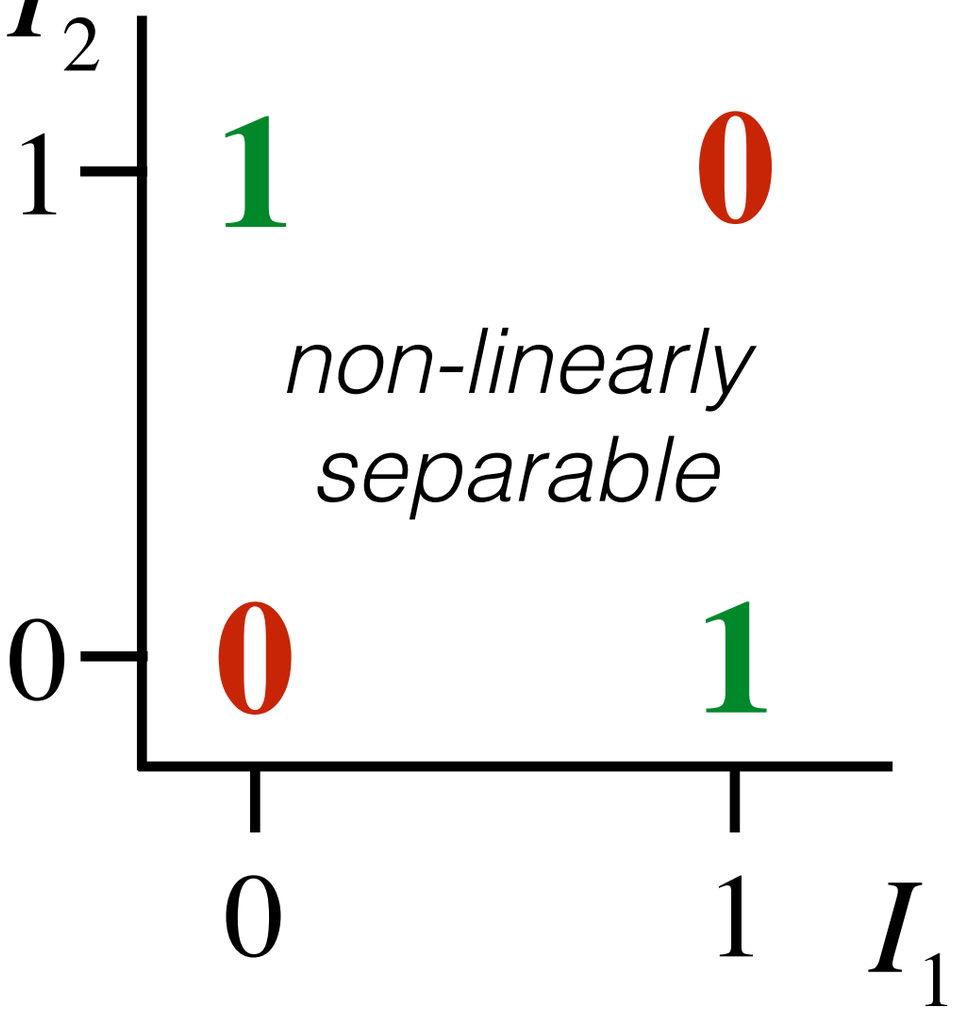
computations by the simplest neural network



I_1	I_2	o
0	0	$w_1 \times 0 + w_2 \times 0 + \beta < 0$
1	0	$w_1 \times 1 + w_2 \times 0 + \beta \geq 0$
0	1	$w_1 \times 0 + w_2 \times 1 + \beta \geq 0$
1	1	$w_1 \times 1 + w_2 \times 1 + \beta < 0$

logical
XOR

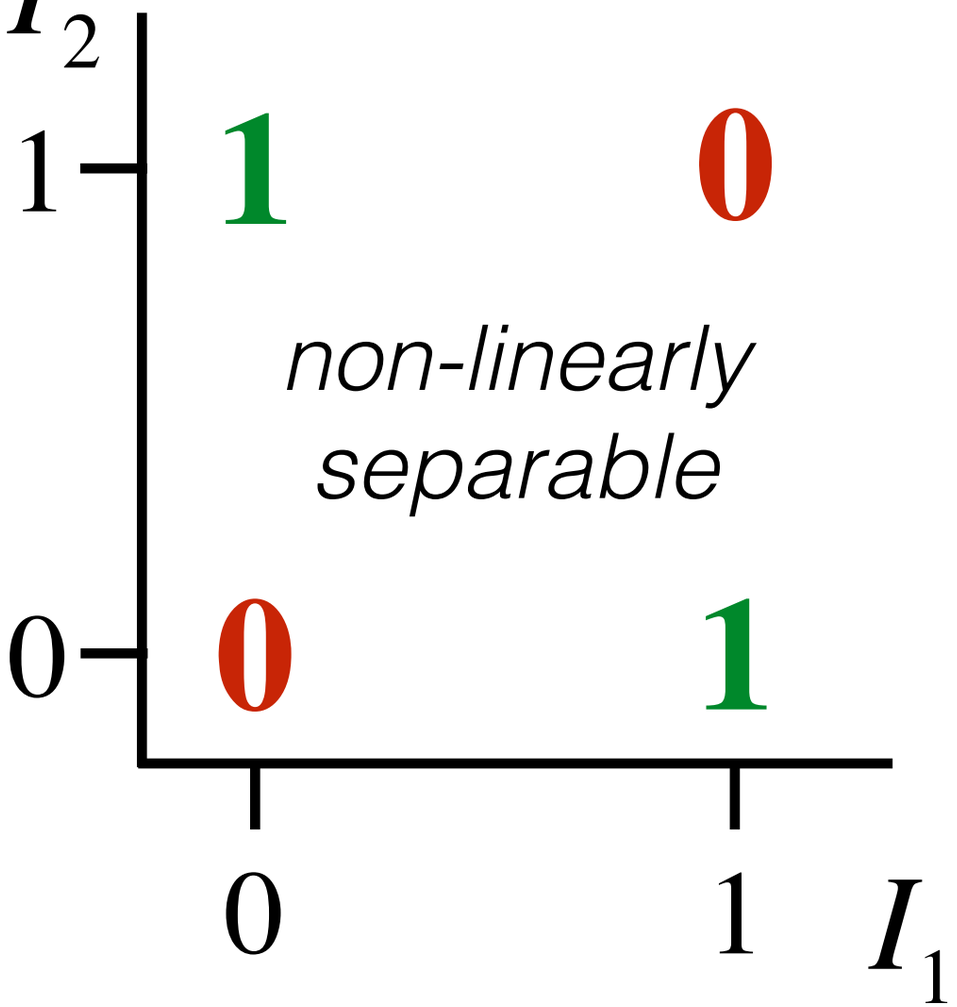
computations by the simplest neural network



I_1	I_2	o
		$\beta < 0$
		$w_1 + \beta \geq 0$
		$w_2 + \beta \geq 0$
		$w_1 + w_2 + \beta < 0$

mutually
contradictory
(*convince yourself*)

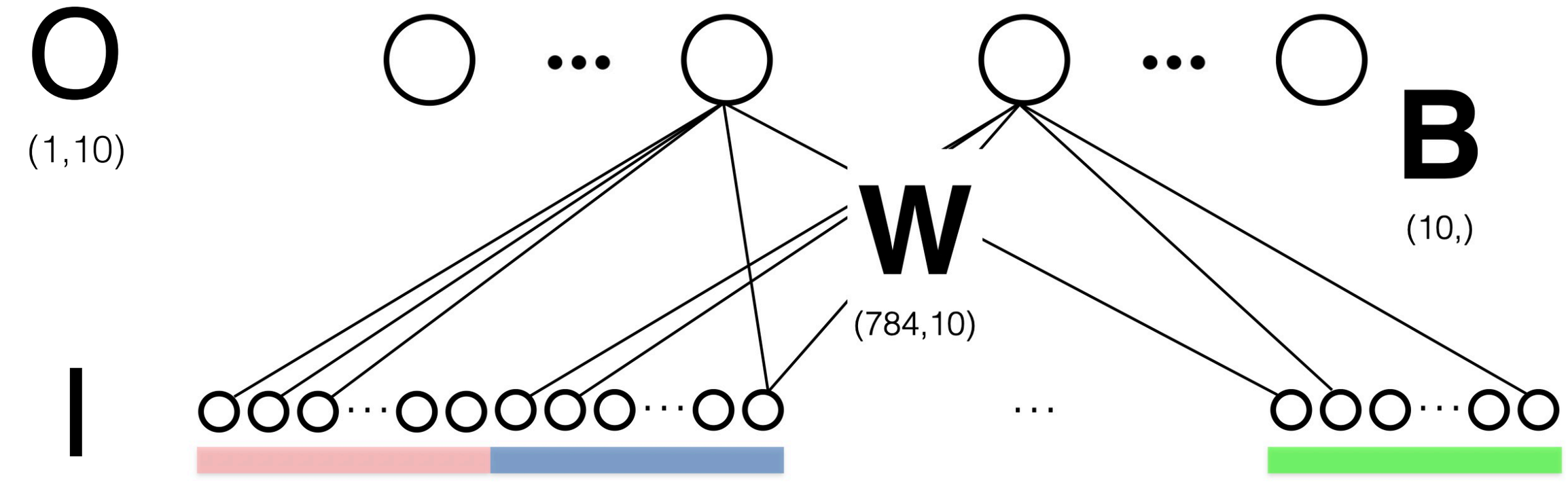
computations by the simplest neural network



I_1	I_2	o
		$\beta < 0$
		$w_1 + \beta \geq 0$
		$w_2 + \beta \geq 0$
		$w_1 + w_2 + \beta < 0$

neural networks with multiple layers can solve this!!!
(as we'll see a couple weeks from now)

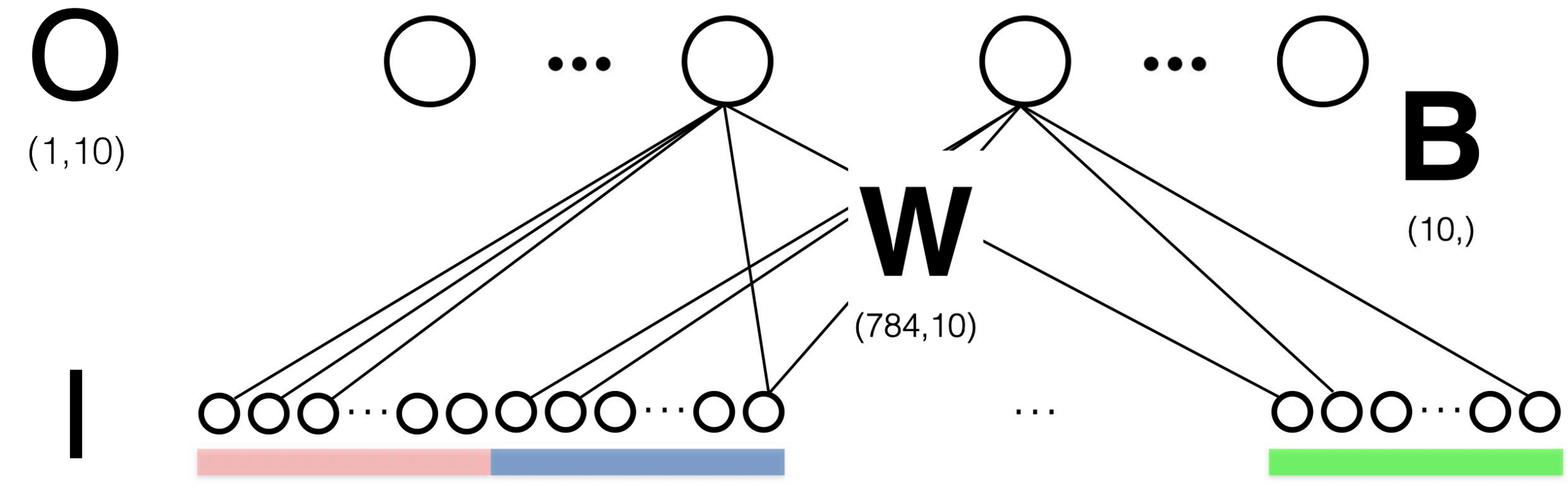
using matrix algebra with larger neural networks



$$O_{(1,10)} = f(\ I_{(1,784)} \times W_{(784,10)} + B_{(10,)}\ )$$

activation function

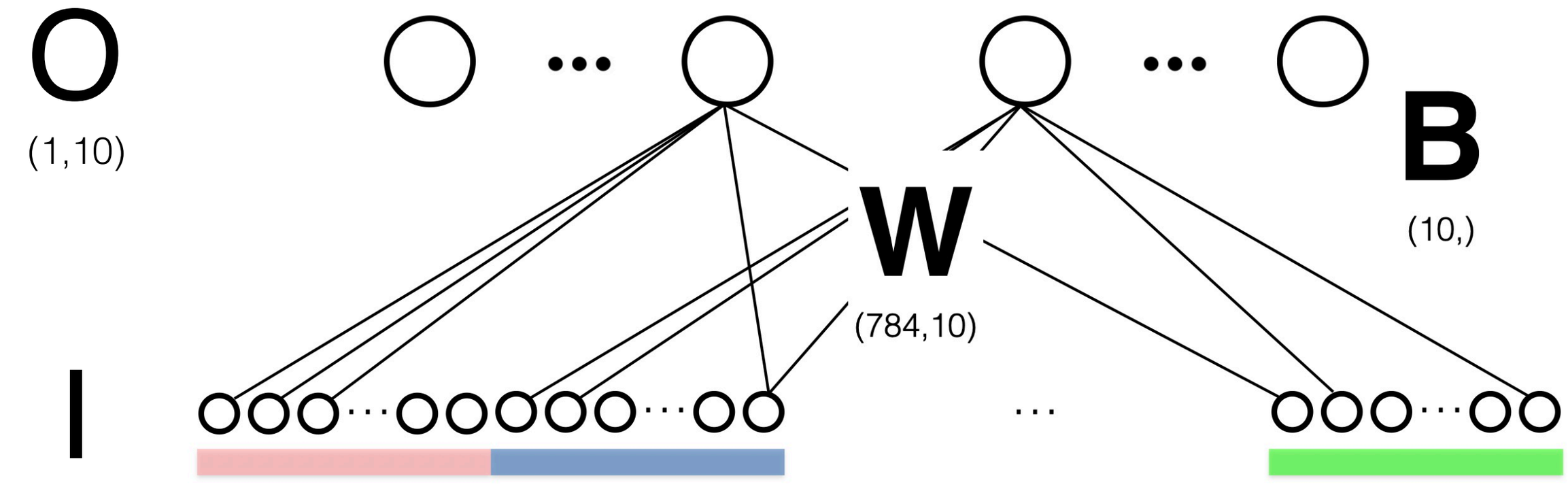
using matrix algebra with larger neural networks



$$\underset{(1,10)}{O} = f(\underset{(1,784)}{I} \times \underset{(784,10)}{W} + \underset{(1,10)}{B})$$

vector vector matrix vector

using matrix algebra with larger neural networks



vector vector matrix vector

$$O_{(1,N)} = f(\ I_{(1,M)} \times W_{(M,N)} + B_{(1,N)}\ )$$

M = # input nodes

N = # output nodes

using matrix algebra with larger neural networks

1 row
N columns
o_j

1 row
M columns
I_i

M rows
N columns
w_{ij}

1 row
N columns
β_j

$$[_{(1,N)}] = f([_{(1,M)}] \times \begin{bmatrix} \\ _{(M,N)} \\ \\ \end{bmatrix} + [_{(1,N)}])$$

$$O = I \times W + B$$

vector vector matrix vector

$$O_{(1,N)} = f(I_{(1,M)} \times W_{(M,N)} + B_{(1,N)})$$

M = # input nodes
N = # output nodes

using matrix algebra with larger neural networks

1 row N columns	1 row M columns	M rows N columns	1 row N columns
o_j	I_i	w_{ij}	β_j

$$[_{(1,N)}] = f([_{(1,M)}] \times [_{(M,N)}] + [_{(1,N)}])$$

$$O = f(I \times W + B)$$

vector	vector	matrix	vector
O	I	W	B
(1,N)	(1,M)	(M,N)	(1,N)

M = # input nodes
N = # output nodes

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \end{bmatrix}$$

1xM MxN 1xN

w_{ij} is weight
from input i_i
to output o_j

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \end{bmatrix}$$

1xM MxN 1xN

w_{ij} *is weight from input* i_i *to output* o_j

$$x_j = \sum_{k}^{M} i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \end{bmatrix}$$

1xM MxN 1xN

w_{ij} *is weight from input* i_i *to output* o_j

$$x_j = \sum_{k}^{M} i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \end{bmatrix}$$

1xM MxN 1xN

w_{ij} *is weight from input* i_i *to output* o_j

$$x_j = \sum_{k}^{M} i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_k^M i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_k^M i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_k^M i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_k^M i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_k^M i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_k^M i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_k^M i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_{k}^{M} i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

each row represents a different input pattern (like evaluating many MNIST digits at once)

$$x_j = \sum_k^M i_k w_{kj}$$

matrix multiplication

$$\begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{P1} & i_{P2} & \cdots & i_{PM} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \cdots & x_{PN} \end{bmatrix}$$

PxM MxN PxN

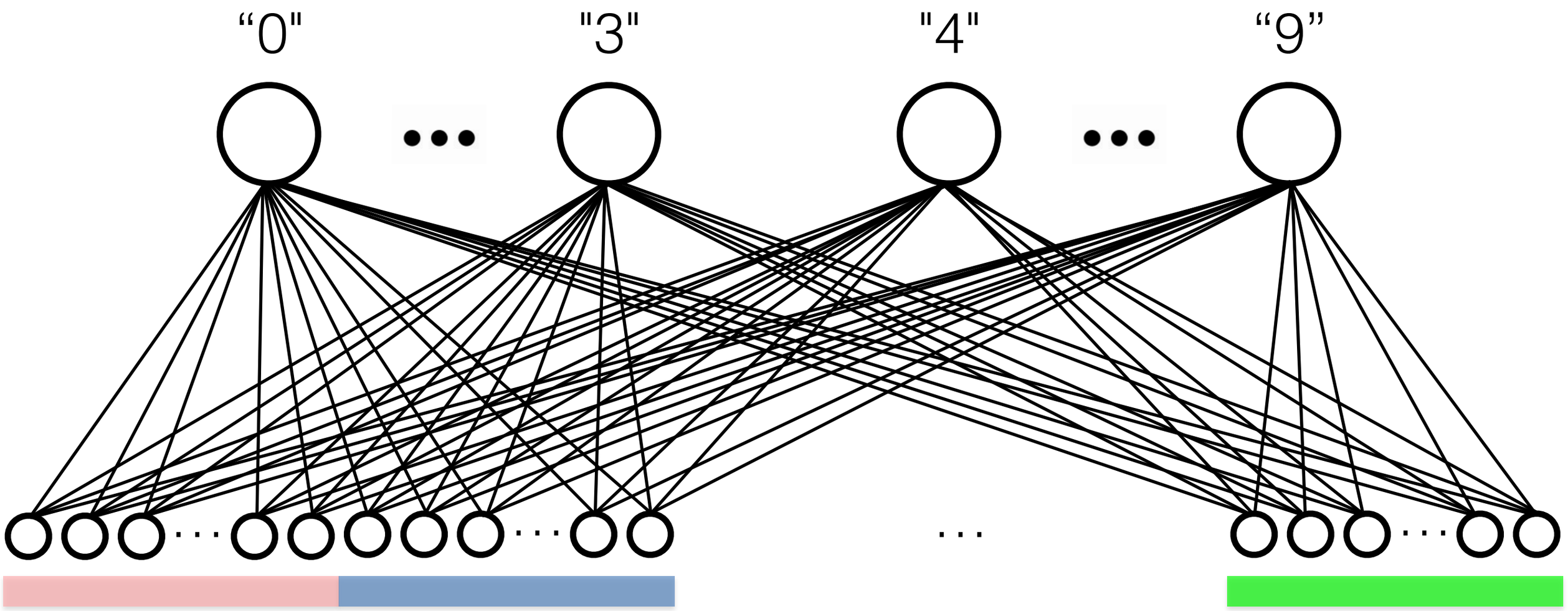
each row represents a different input pattern (like evaluating many MNIST digits at once)

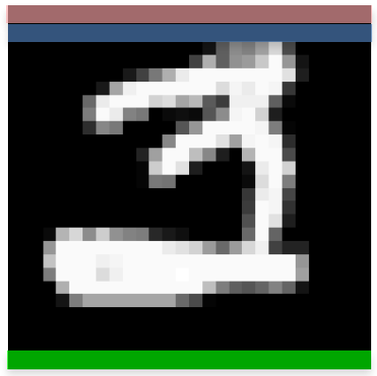
$$x_j = \sum_{k}^{M} i_k w_{kj}$$

Learning in Neural Networks
(Training Neural Networks)

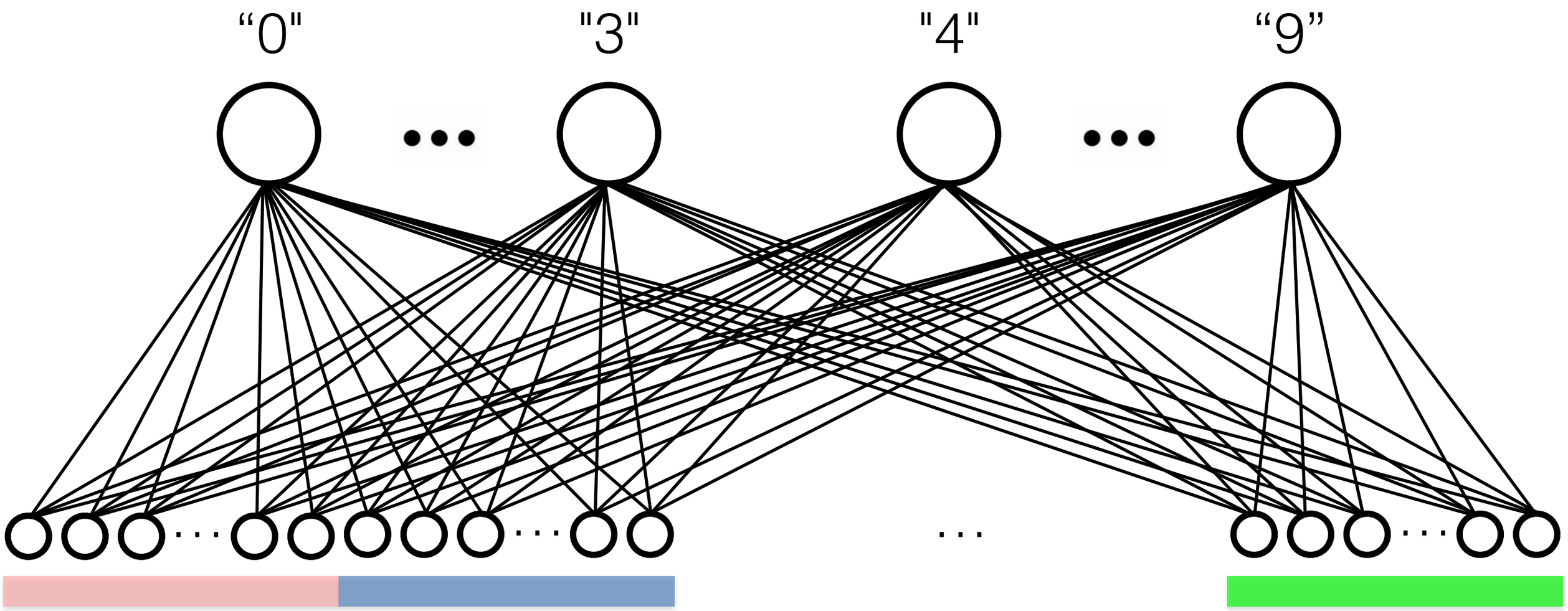
(supervised) learning involves gradually adjusting the weights and biases to produce the correct response

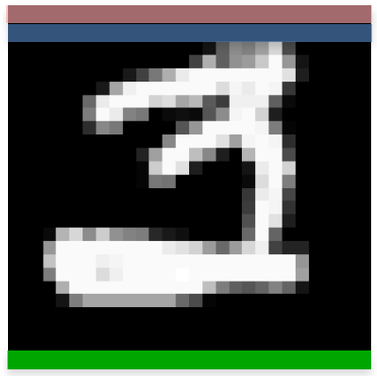
(given by a "teacher")



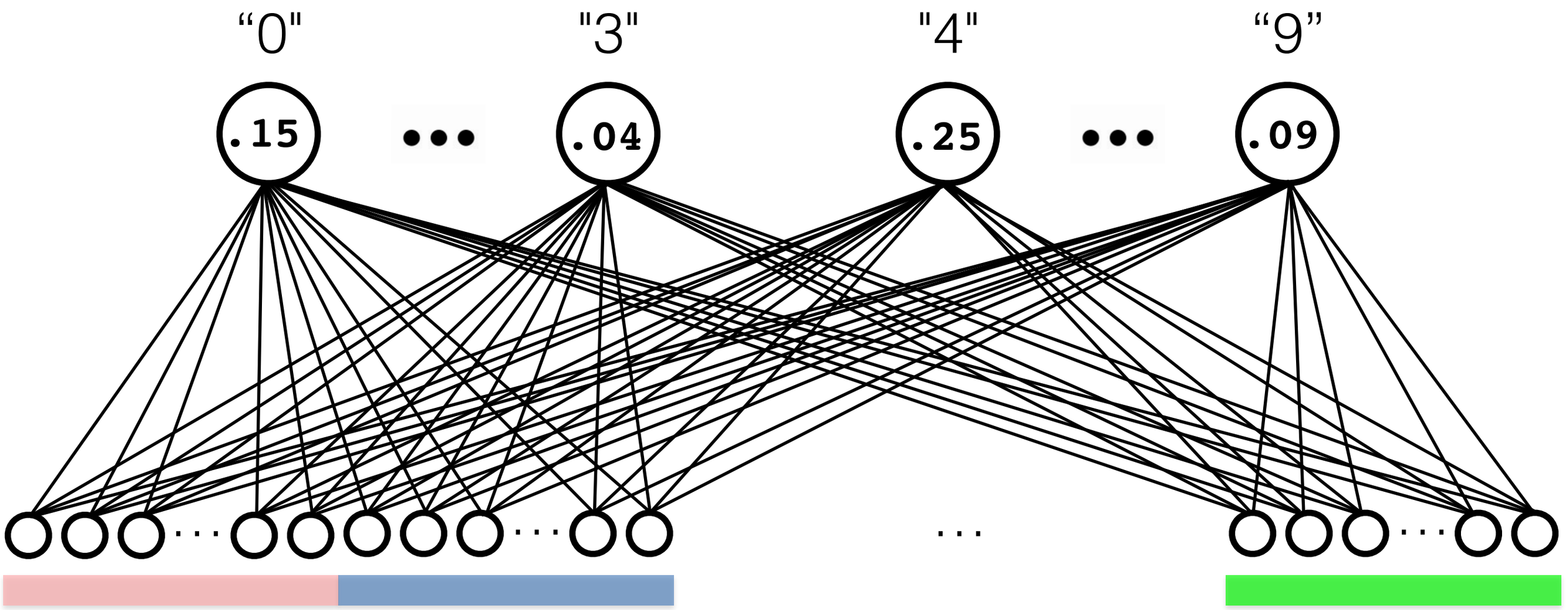


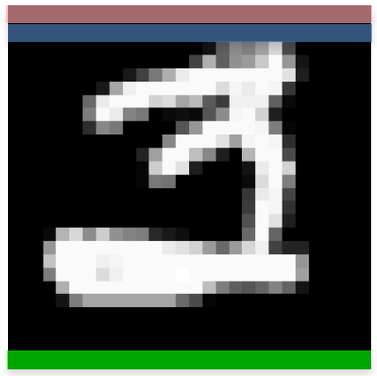
weights and biases are usually initialized to <u>small random values</u> (in many cases, learning algorithms will not work if they are not initialized this way)



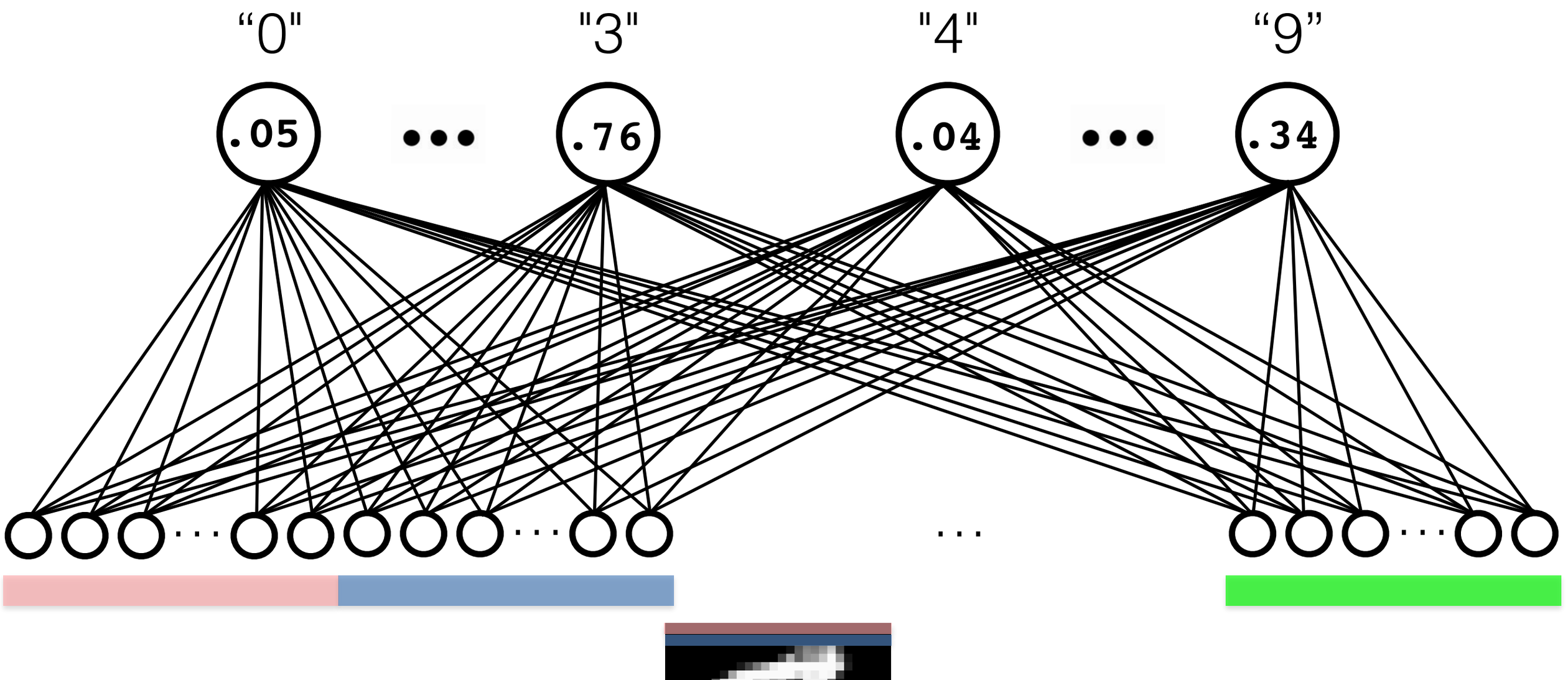


at the start of learning, given a particular input, the outputs will be far from the correct answer
(just random weights and biases)

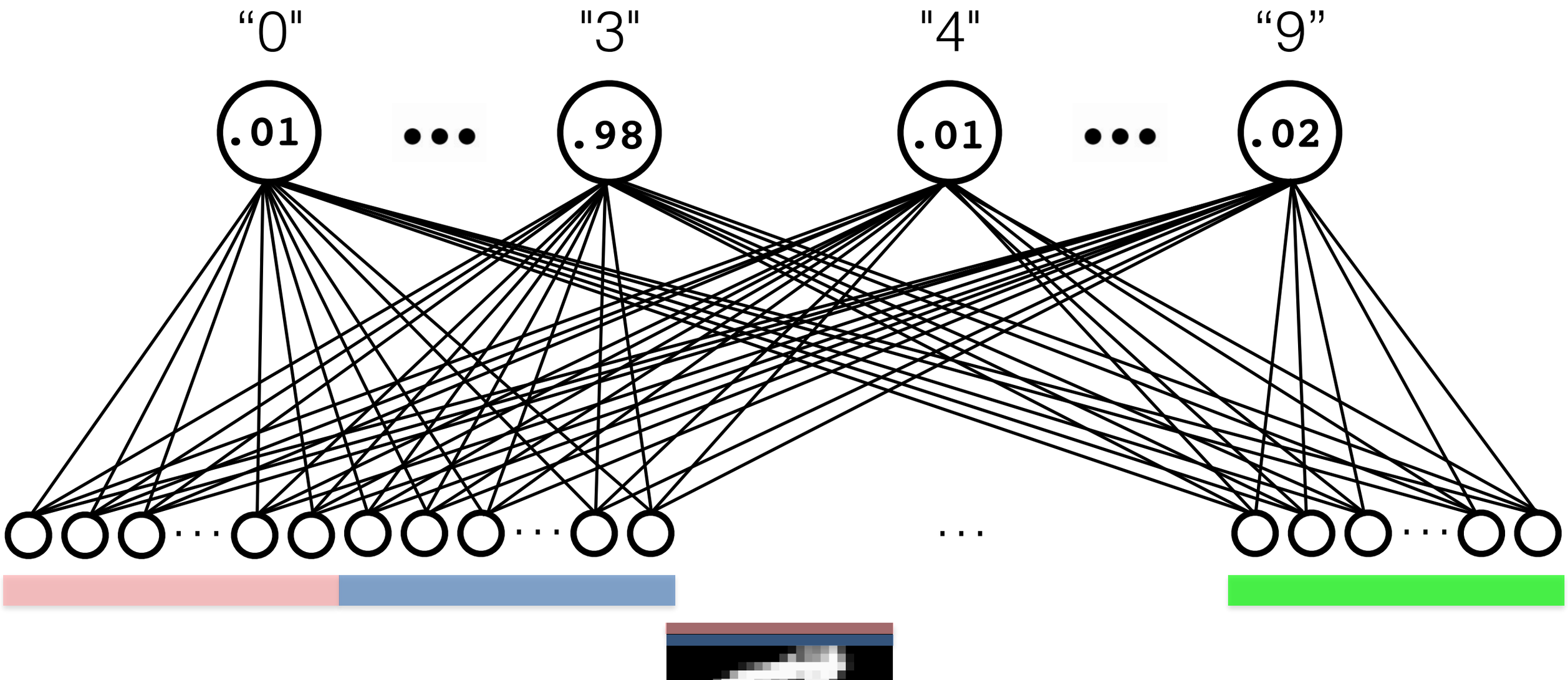




over the course of learning, if a solution is possible, the weights and biases will be gradually adjusted to produce responses closer to the correct value

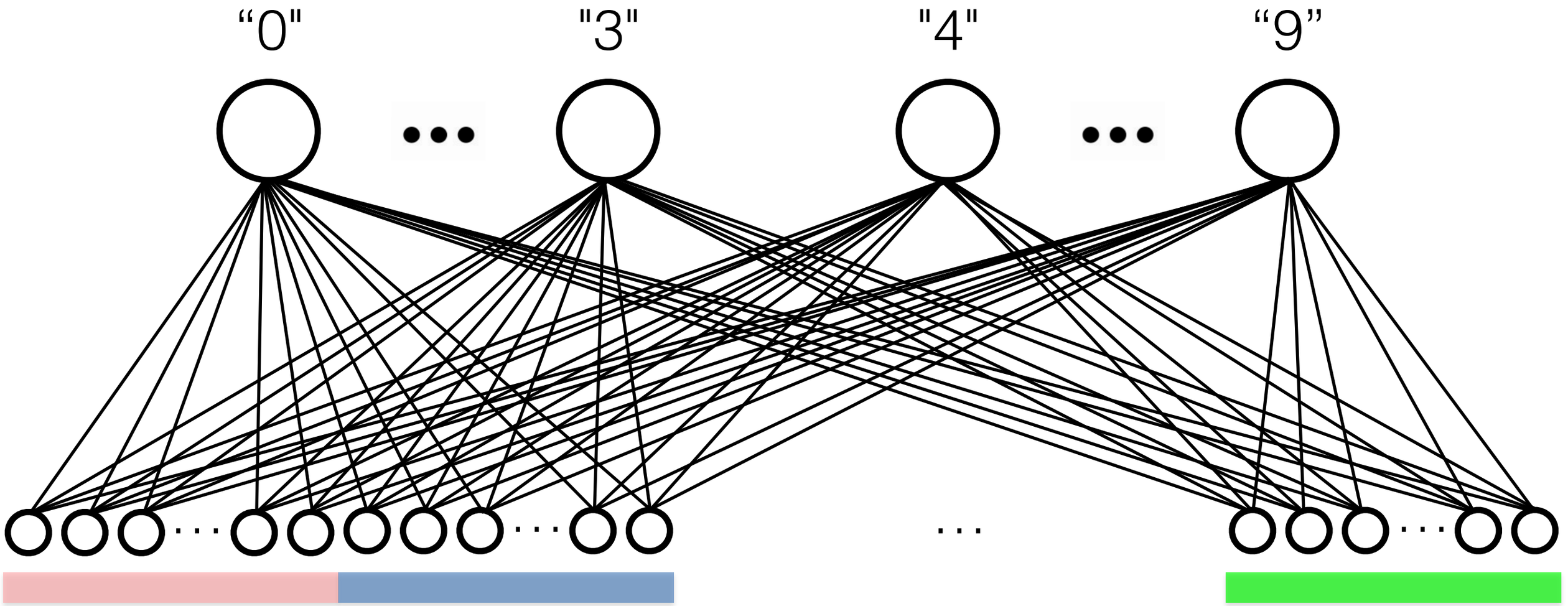


the "best" network will be close to perfect
(MNIST digits cannot be learned anywhere near perfectly using just a single-layer neural network)

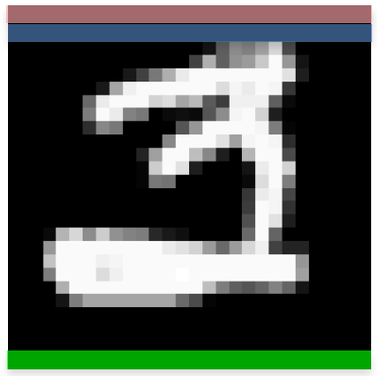


for each training pattern (MNIST digit in this case) there is a "teacher" value - weights are adjusted a bit to move the output gradually towards the teacher value

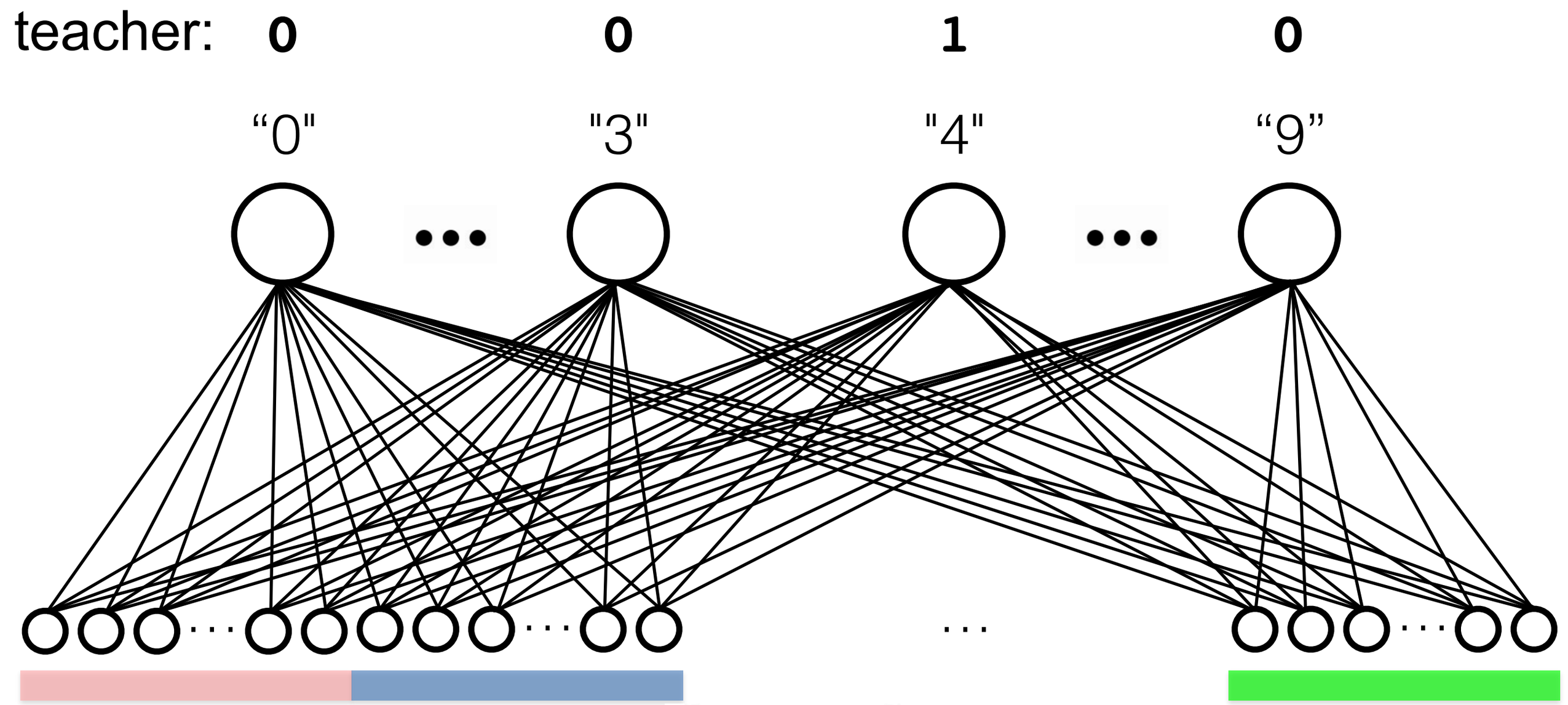
teacher: **0** **1** **0** **0**



training pattern:

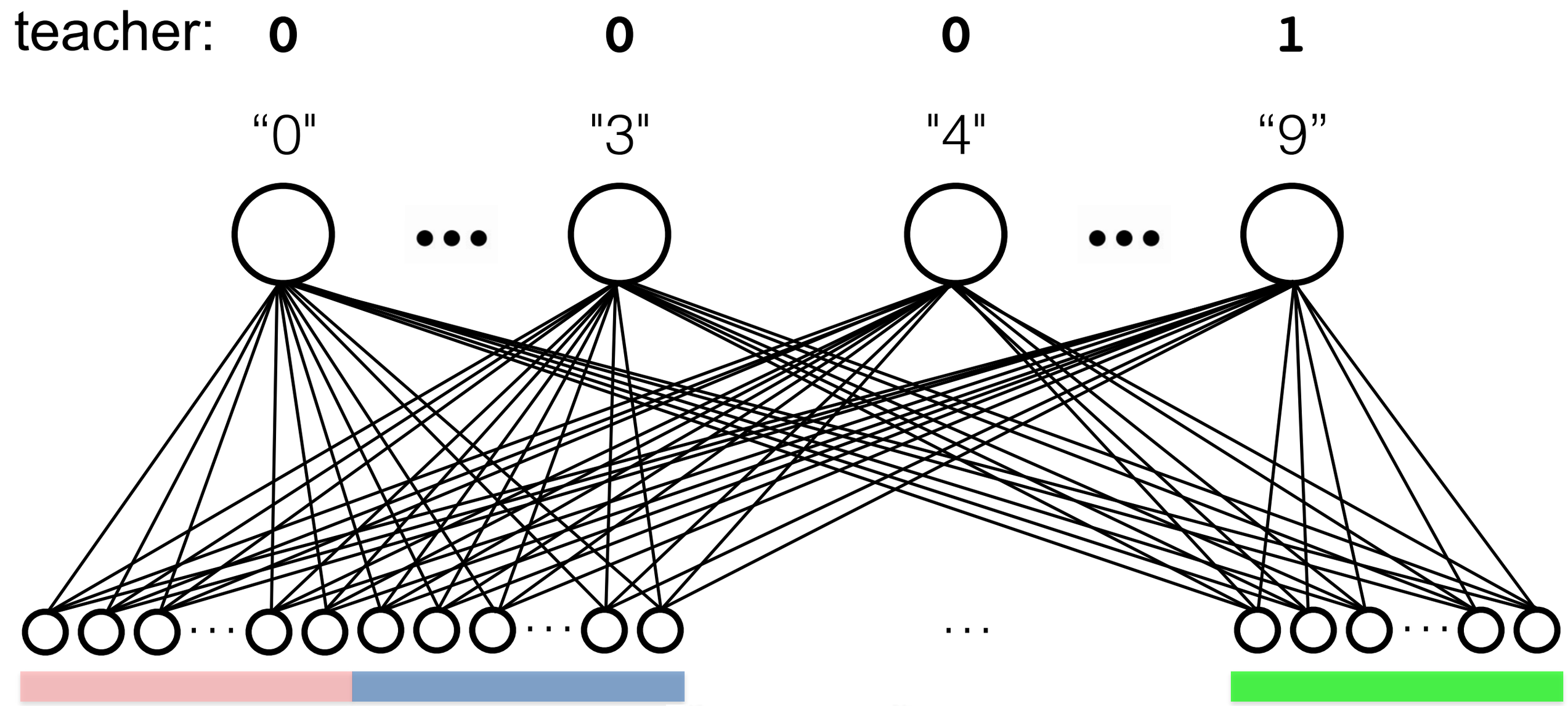


for each training pattern (MNIST digit in this case) there is a "teacher" value - weights are adjusted a bit to move the output gradually towards the teacher value



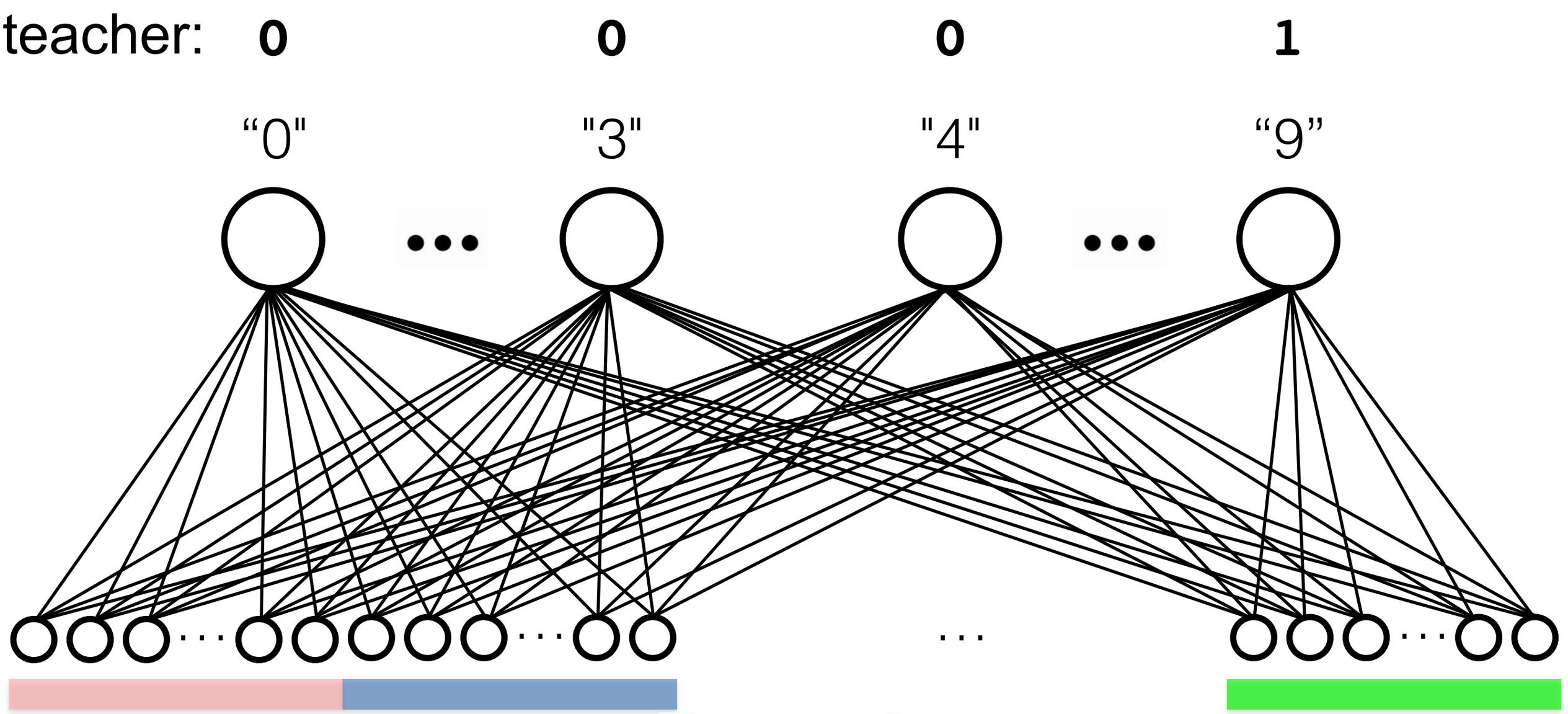
training pattern:

for each training pattern (MNIST digit in this case) there is a "teacher" value - weights are adjusted a bit to move the output gradually towards the teacher value



training pattern:

how are the weights (and biases) adjusted?
how does learning take place?



training pattern:

How Neural Networks Learn

(adjust weights and biases)

Upcoming Lectures

- Hebbian Learning
- Error-Driven Learning (single-layer networks)
- Error-Driven Learning (multi-layer networks)
- using Keras and Tensorflow

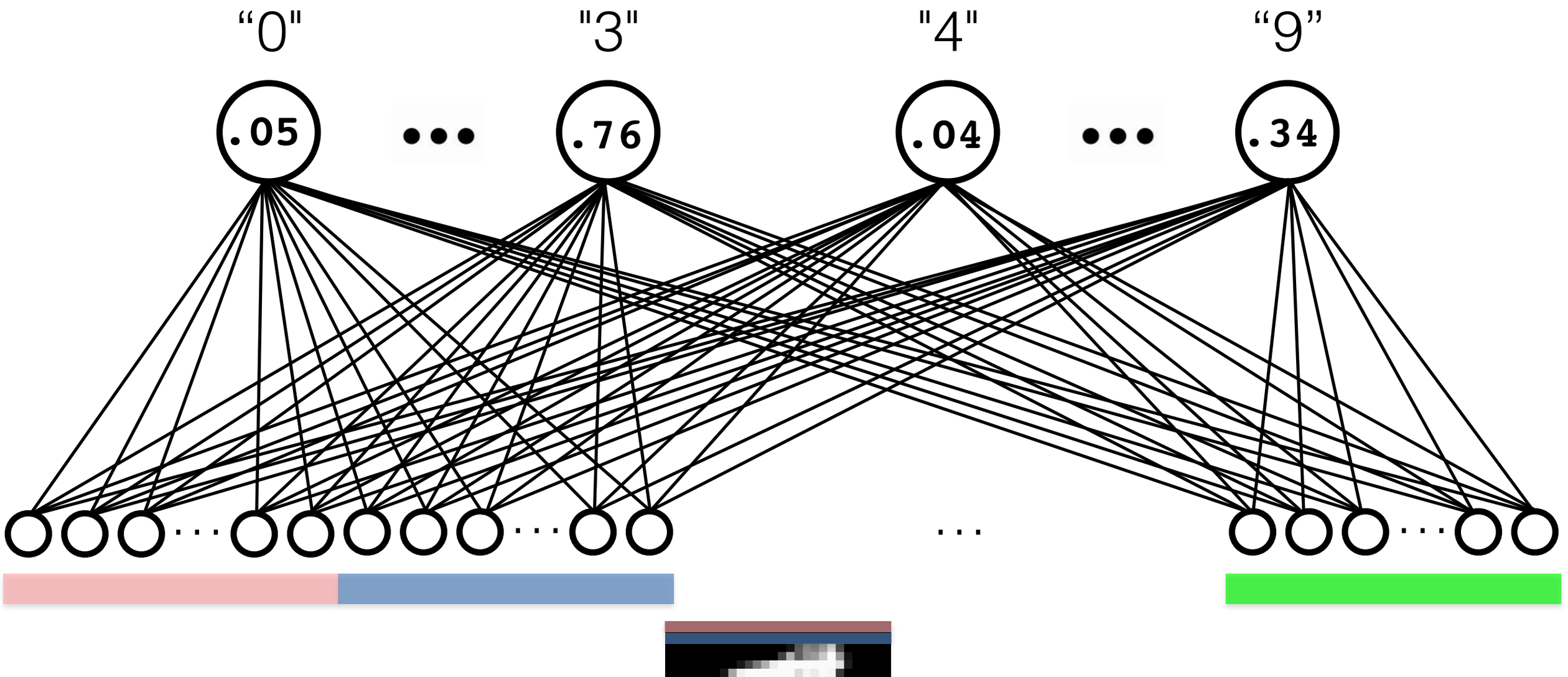
Hebbian Learning

an early (classic) theory of learning
in neural networks

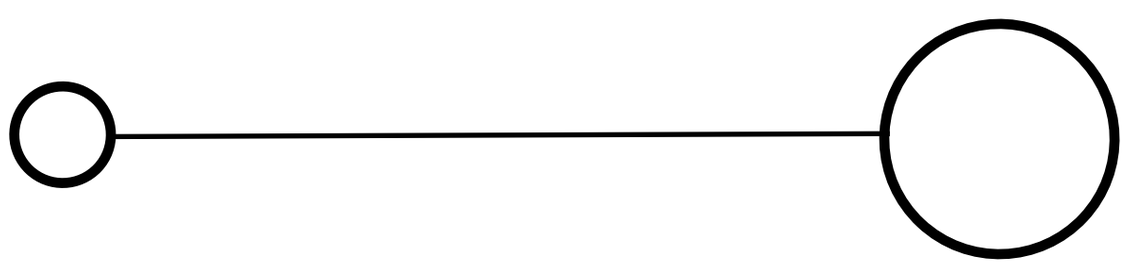
Donald O. Hebb
(1904-1985)

Hebbian Learning

let's take a look at one weight



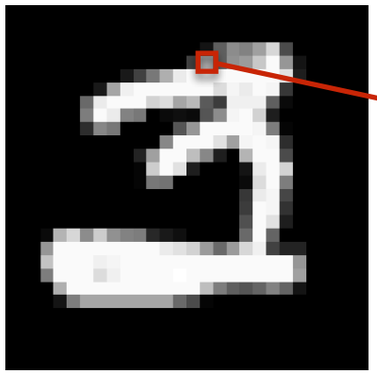
Hebbian Learning

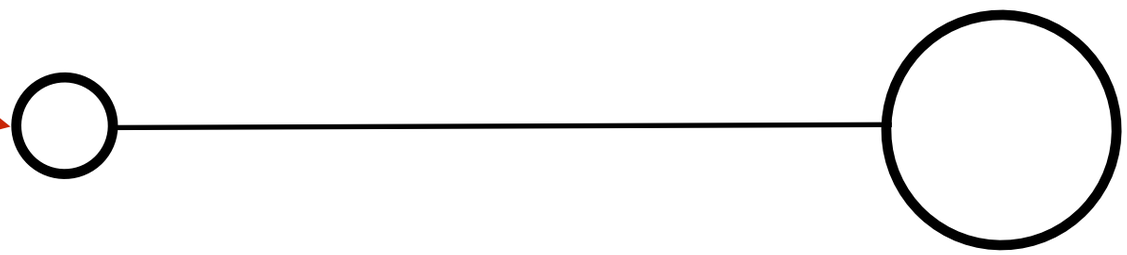


$i_i \quad w_{ij} \quad o_j$

Hebbian Learning

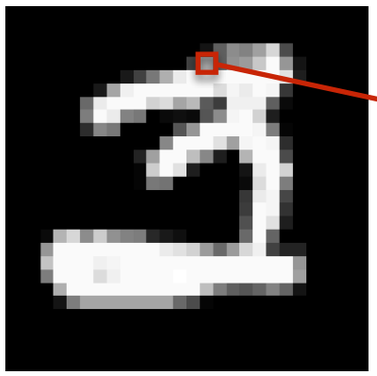
input comes from
training patterns



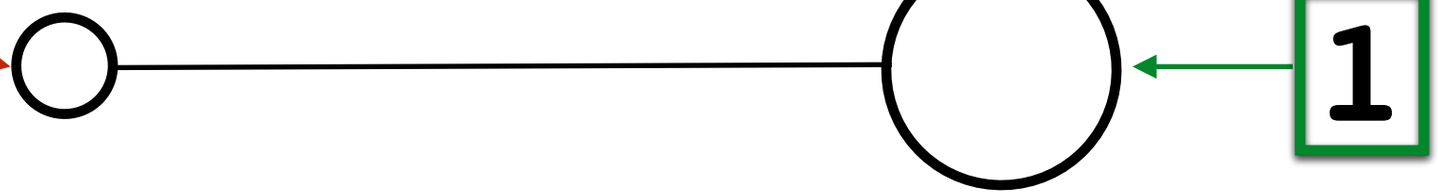


i_i w_{ij} o_j

Hebbian Learning



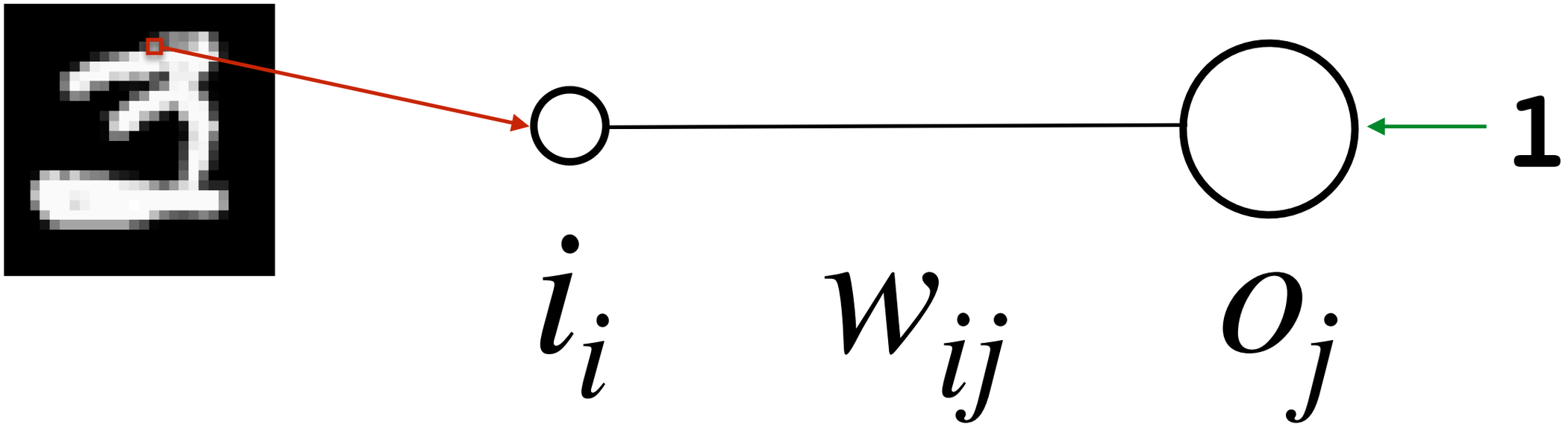
during learning, "teacher"
clamped onto output



i_i w_{ij} o_j

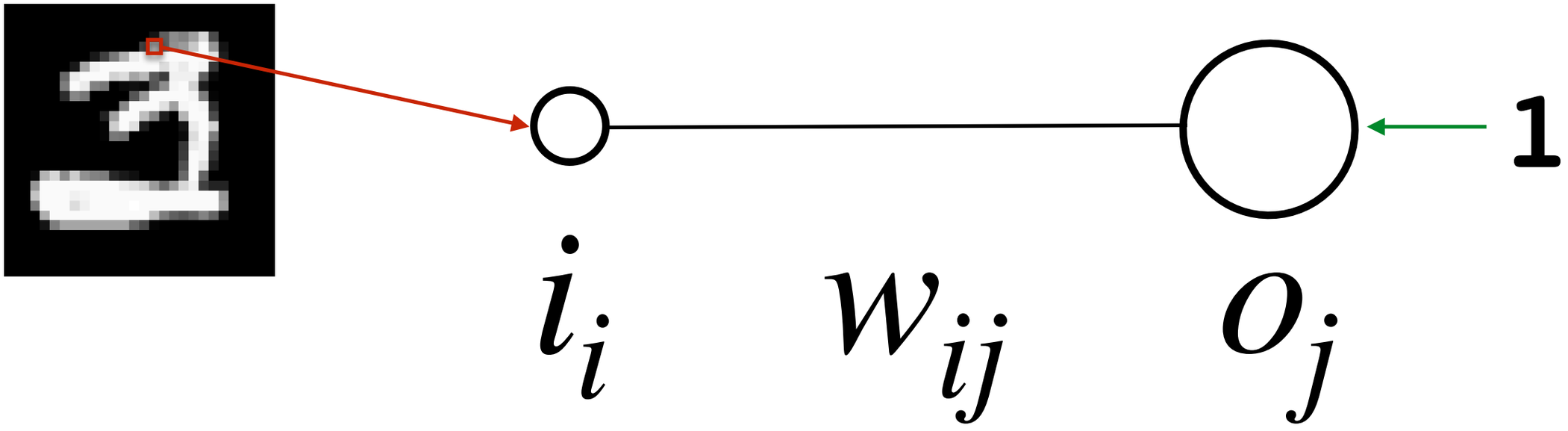
Hebbian Learning

neurons that fire together wire together



Hebbian Learning

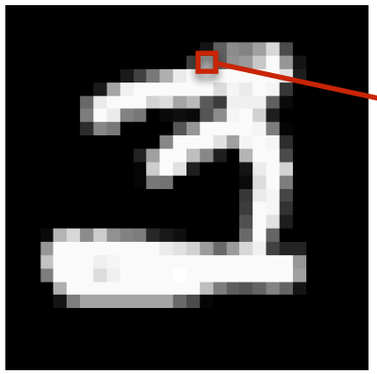
neurons that fire together wire together

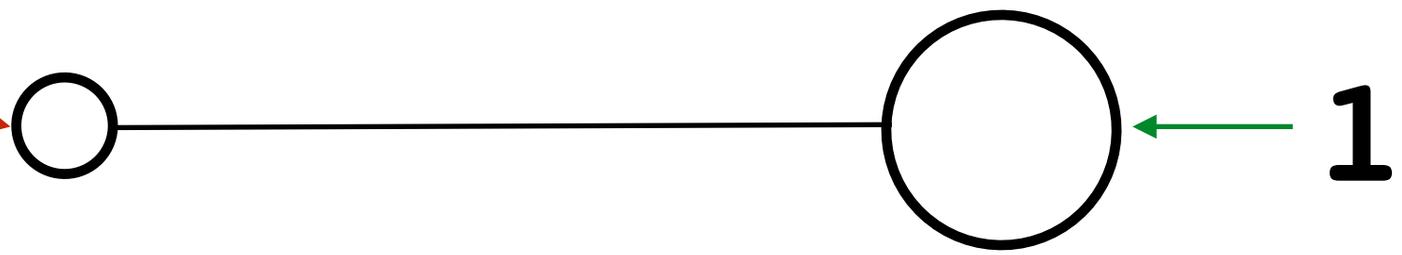


$$\Delta w_{ij} = \lambda \, i_i \, o_j$$

Hebbian Learning

neurons that fire together wire together





$i_i \quad w_{ij} \quad o_j$

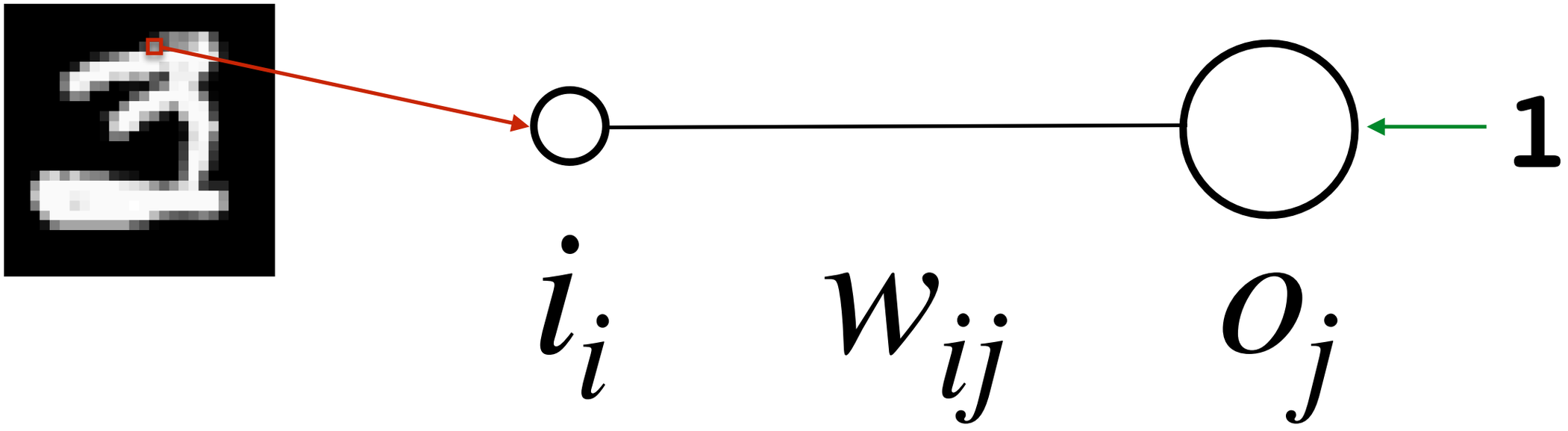
learning rate

often small, like λ=.01

$$\Delta w_{ij} = \lambda i_i o_j$$

Hebbian Learning

neurons that fire together wire together

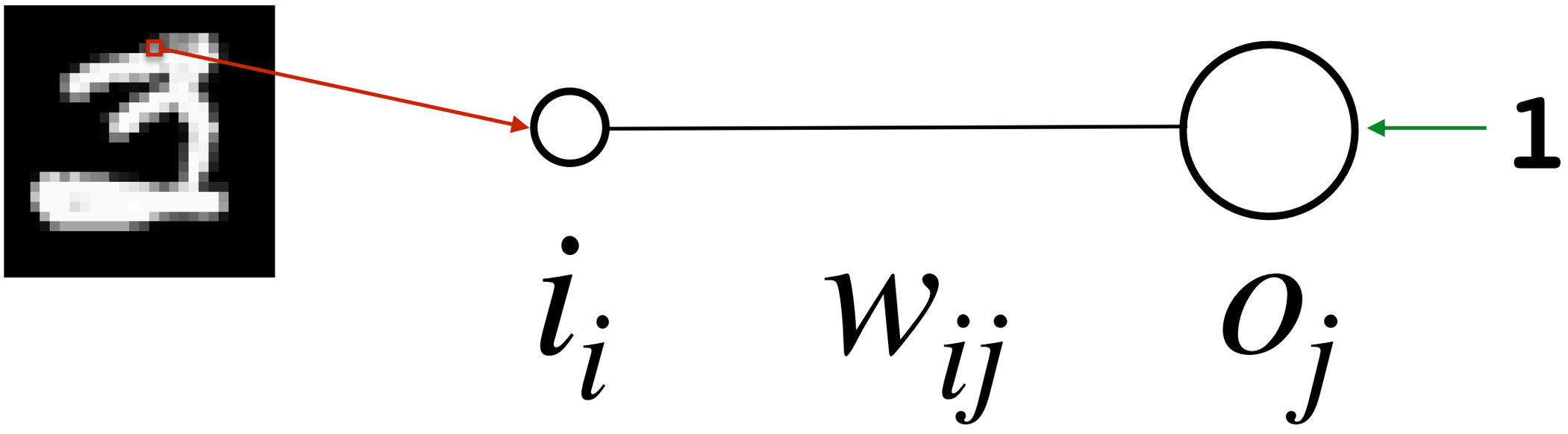


weight after training pattern N — weight before training pattern N

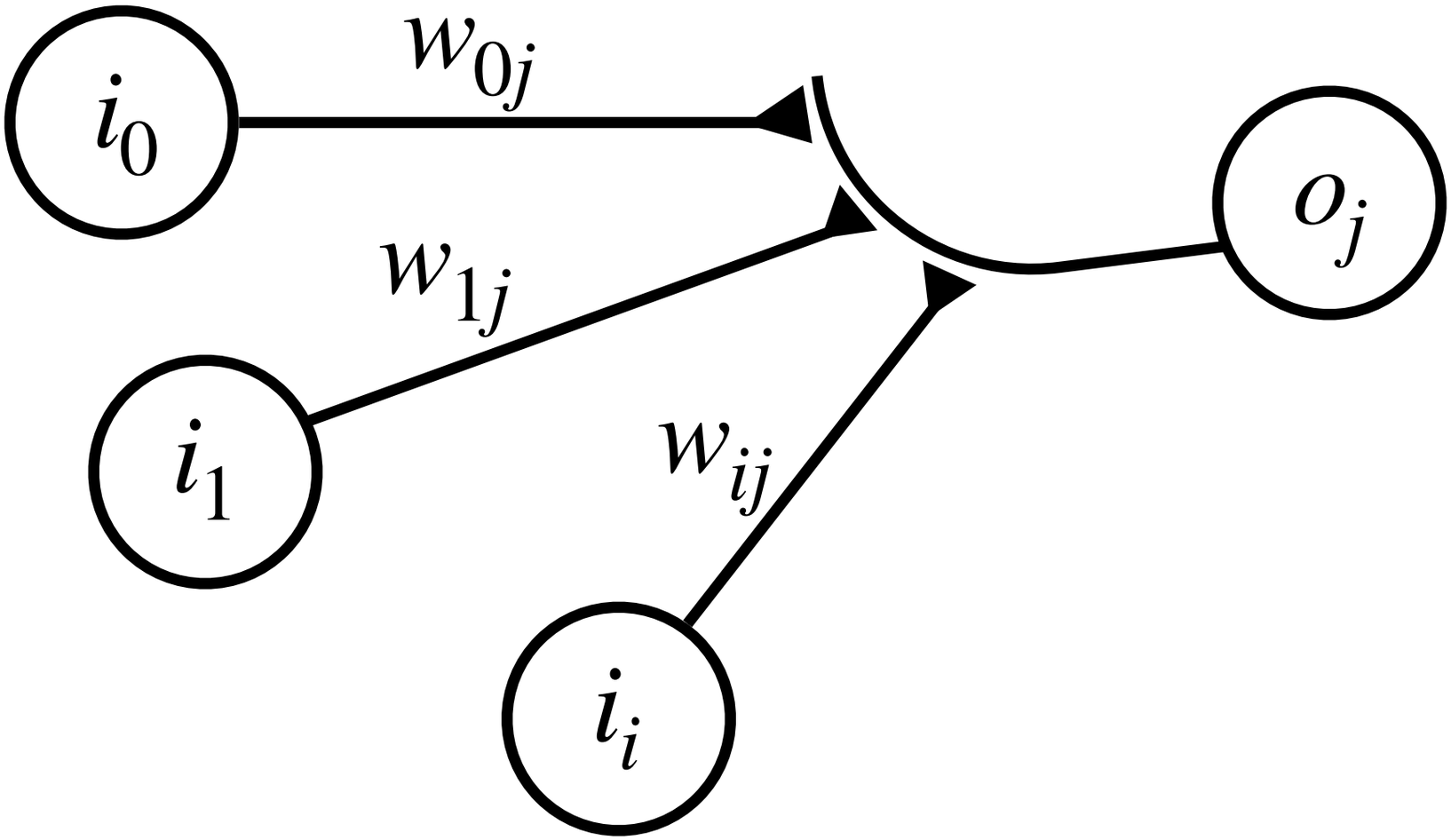
$$w_{ij}^{(N)} = w_{ij}^{(N-1)} + \Delta w_{ij} = w_{ij}^{(N-1)} + \lambda\, i_i\, o_j$$

Hebbian Learning

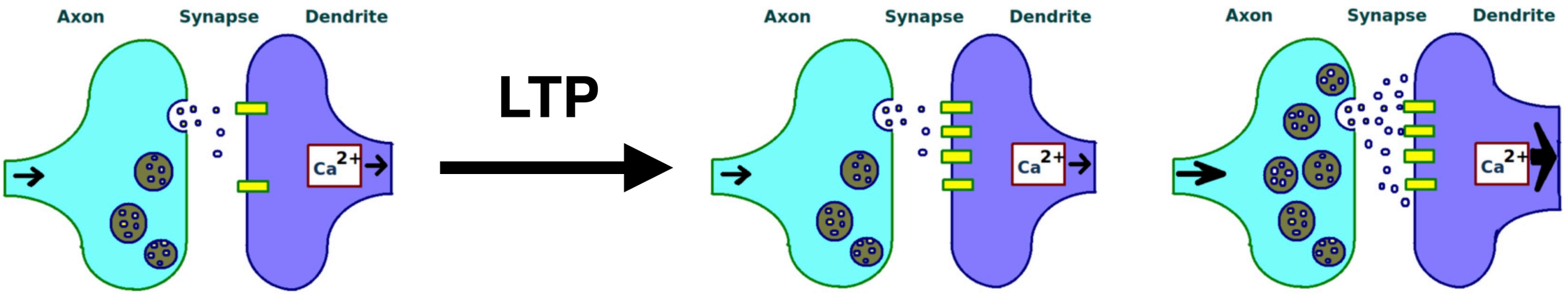
if both units positive, then Δw positive
if both units negative, then Δw positive
if one unit zero, then Δw zero
if units different signs, then Δw negative



$$w_{ij}^{(N)} = w_{ij}^{(N-1)} + \Delta w_{ij} = w_{ij}^{(N-1)} + \lambda\, i_i\, o_j$$



neural evidence for (Hebbian) **long-term potentiation** (LTP)



more receptors
post-synaptic neuron

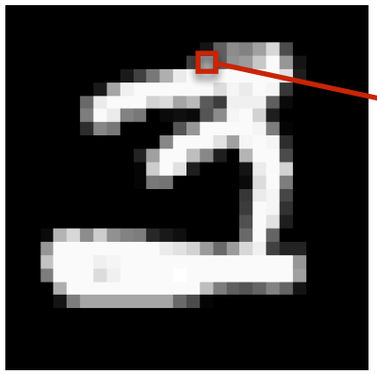
more neurotransmitters
pre-synaptic neuron

Hebbian Learning

we'll see this in code for Homework 4

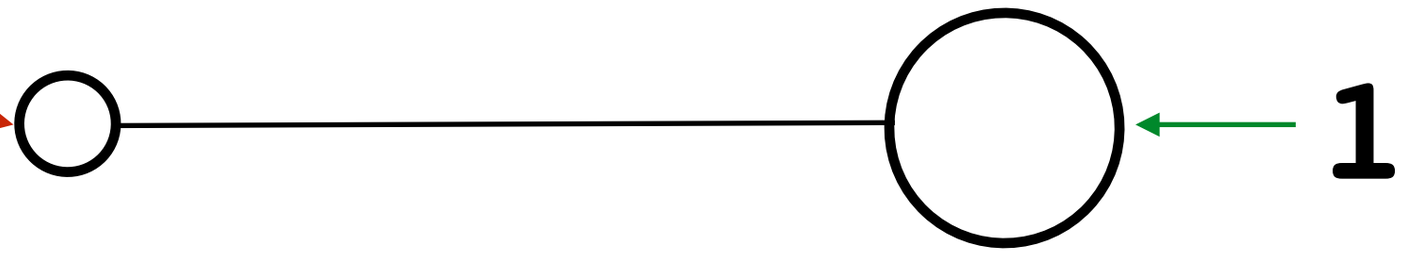
in part because of this, training patterns often rescaled to have negative and positive values, e.g., [-1, 1]

if one unit zero, then Δw zero



pixel values
[0, 255]

[−1, +1]



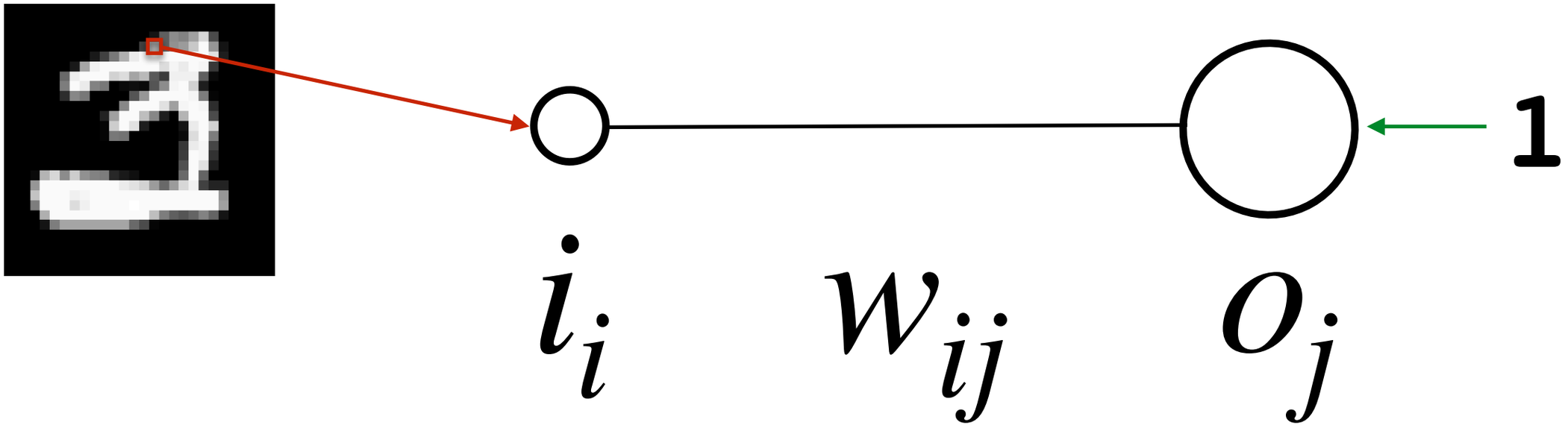
$i_i \quad w_{ij} \quad o_j$

$$w_{ij}^{(N)} = w_{ij}^{(N-1)} + \Delta w_{ij} = w_{ij}^{(N-1)} + \lambda\, i_i\, o_j$$

Hebbian Learning

weights are unbounded, they can grow to positive or negative infinity

if units same sign, then Δw positive
if units different signs, then Δw negative

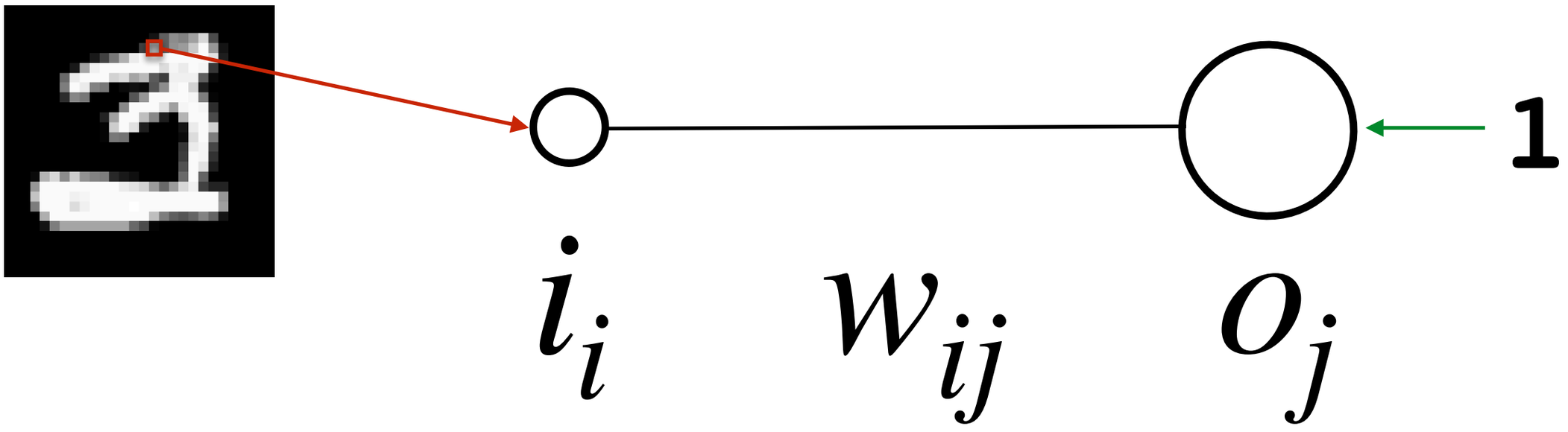


$$w_{ij}^{(N)} = w_{ij}^{(N-1)} + \Delta w_{ij} = w_{ij}^{(N-1)} + \lambda\, i_i\, o_j$$

Oja's Rule

Oja, E. (1982). A simplified neuron model as principal components analysis. *Journal of Mathematical Biology*, 15, 267-273.

Wikipedia Entry for Oja's Rule: https://en.wikipedia.org/wiki/Oja%27s_rule

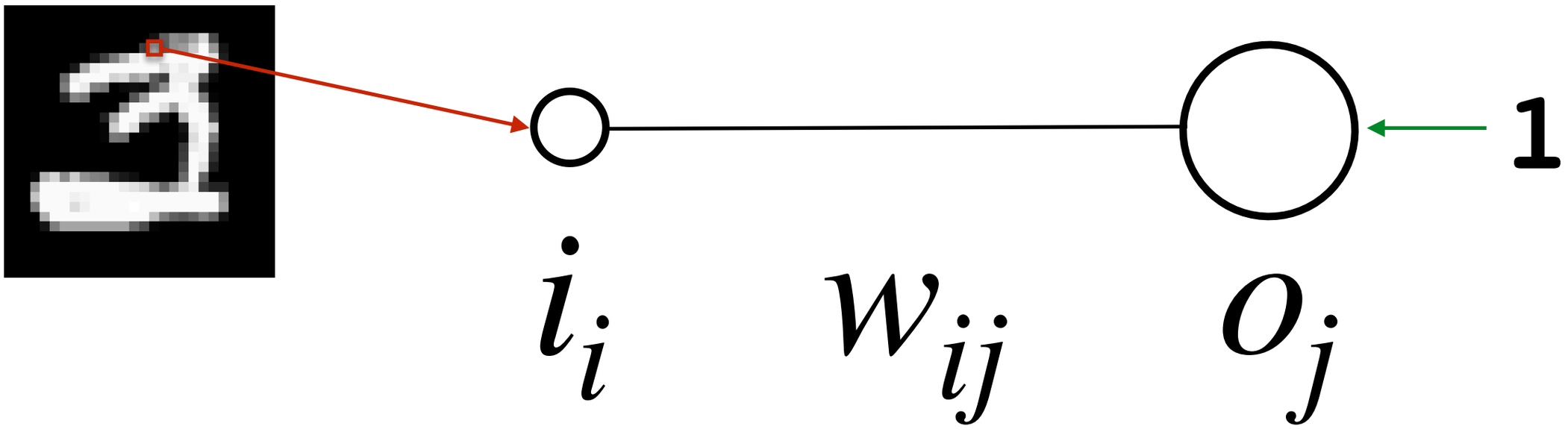


$$\Delta w_{ij} = \lambda o_j (i_i - o_j w_{ij})$$

Oja's Rule

Oja, E. (1982). A simplified neuron model as principal components analysis. *Journal of Mathematical Biology*, 15, 267-273.

Wikipedia Entry for Oja's Rule: https://en.wikipedia.org/wiki/Oja%27s_rule



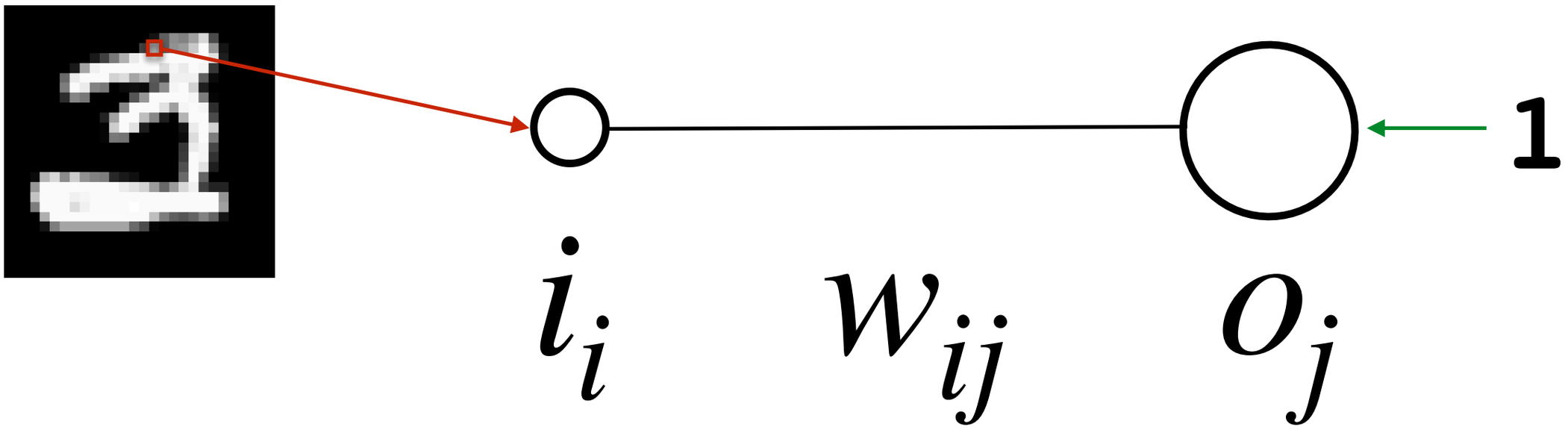
rearrange terms

$$\Delta w_{ij} = \lambda(o_j i_i - o_j^2 w_{ij})$$

Oja's Rule

Oja, E. (1982). A simplified neuron model as principal components analysis. *Journal of Mathematical Biology*, 15, 267-273.

Wikipedia Entry for Oja's Rule: https://en.wikipedia.org/wiki/Oja%27s_rule

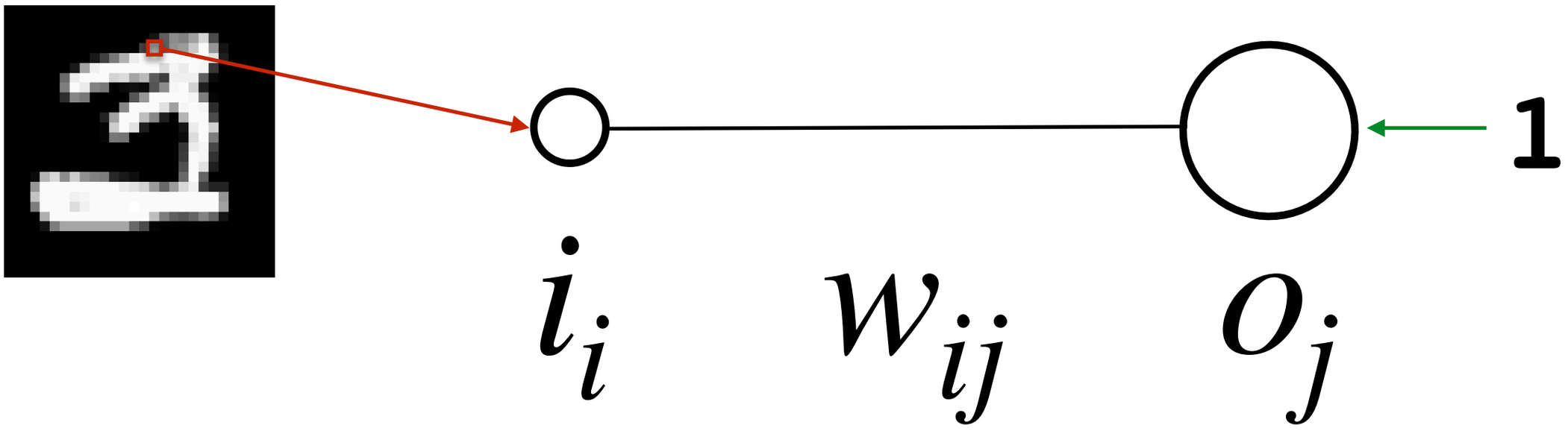


$$\Delta w_{ij} = \lambda(\overbrace{o_j\, i_i}^{\textit{Hebbian learning}} - \overbrace{o_j^2\, w_{ij}}^{\textit{pulls weights towards zero}})$$

Oja's Rule

Oja, E. (1982). A simplified neuron model as principal components analysis. *Journal of Mathematical Biology*, 15, 267-273.

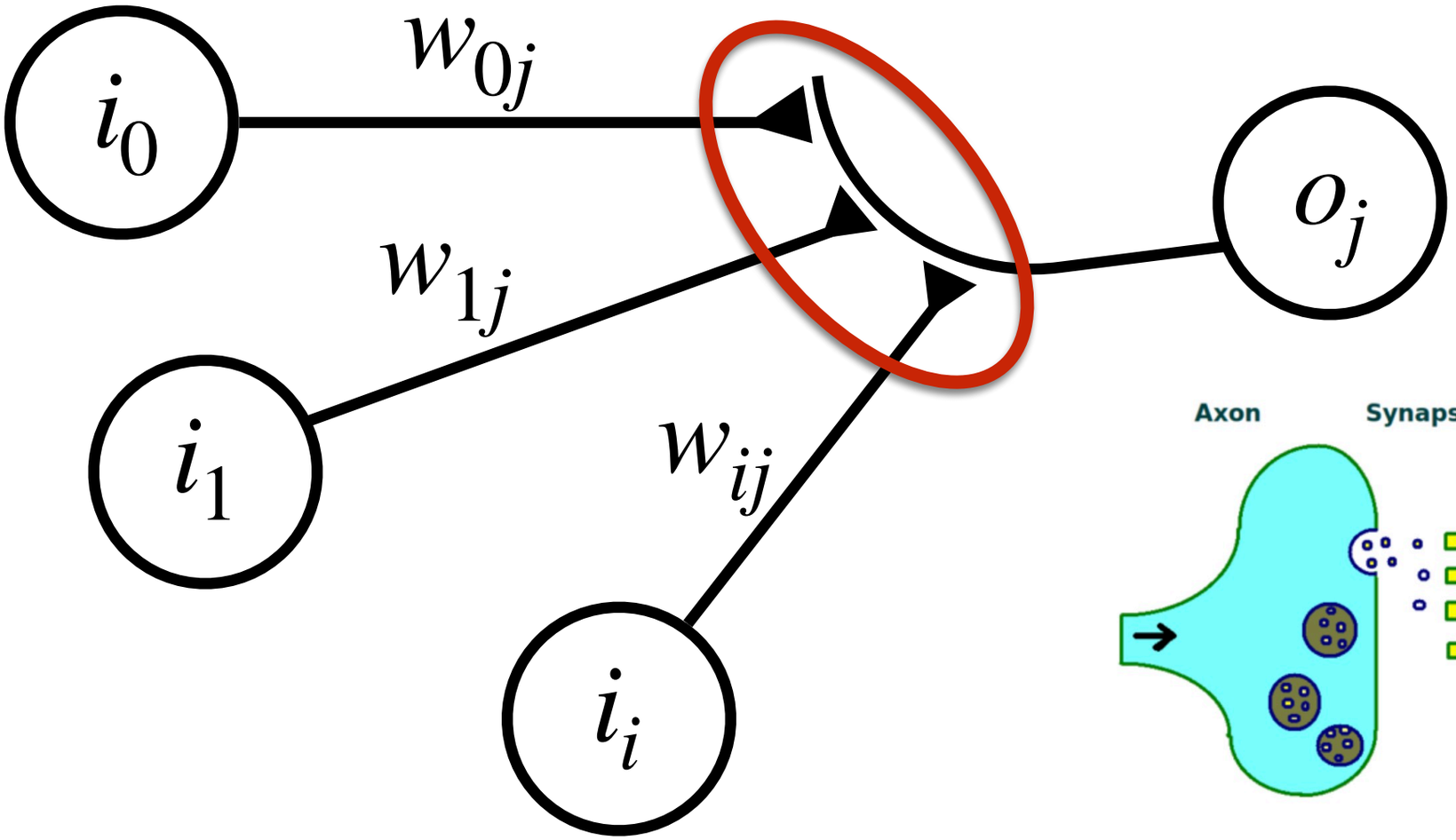
Wikipedia Entry for Oja's Rule: https://en.wikipedia.org/wiki/Oja%27s_rule



Oja's Rule is equivalent (in the limit) to two steps:

(1) $\Delta w_{ij} = \lambda\, i_i\, o_j$ apply Hebbian Learning to all weights given a pattern

(2) normalize weights after weight updates - but which weights?

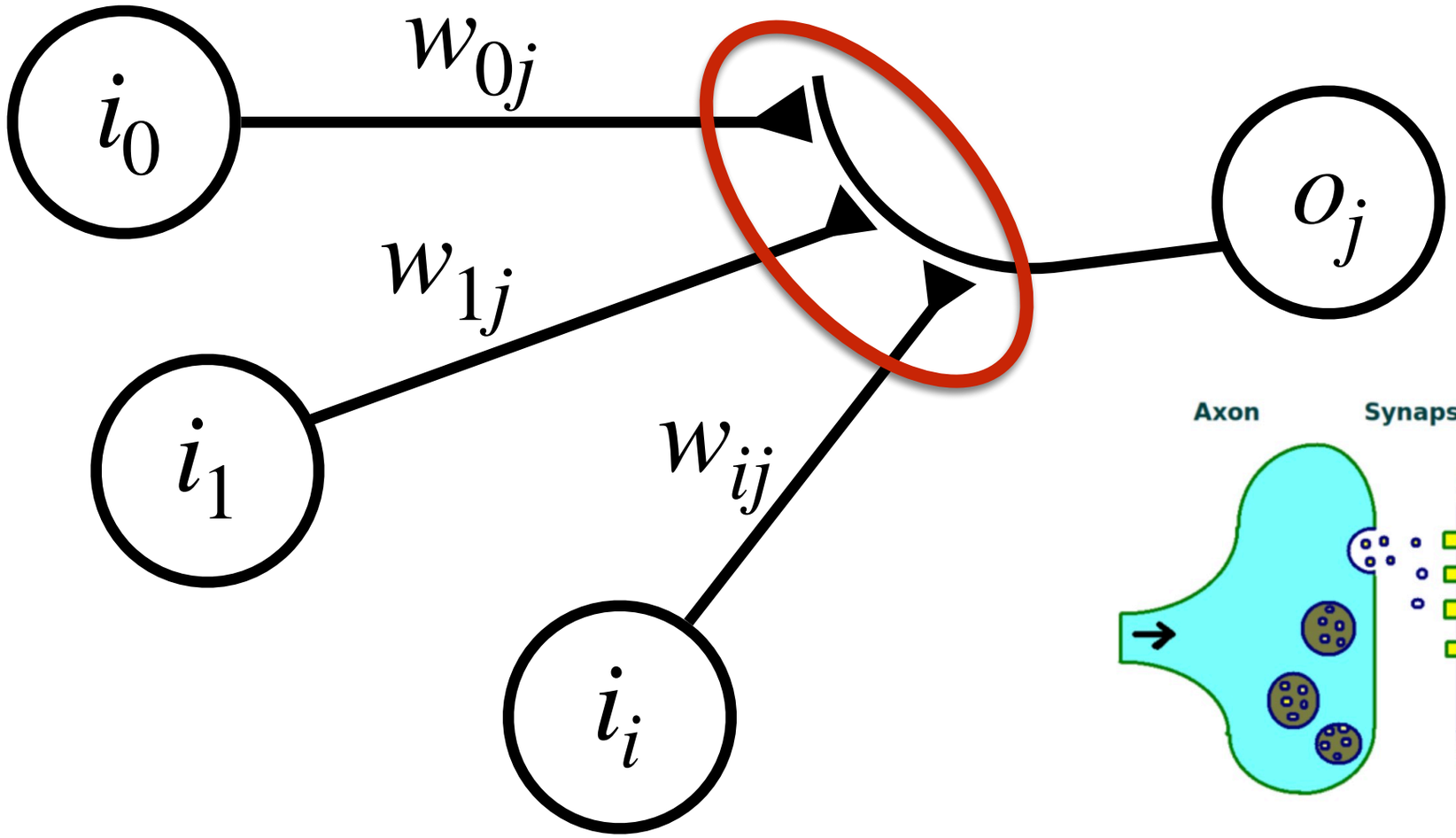


Oja's Rule is equivalent (in the limit) to two steps:

(1) $\Delta w_{ij} = \lambda\, i_i\, o_j$ apply Hebbian Learning to all weights given a pattern

(2) normalize weights after weight updates - but which weights?

normalize the vector (make it have length 1) of weights inputting into the same unit (limited synaptic resources)

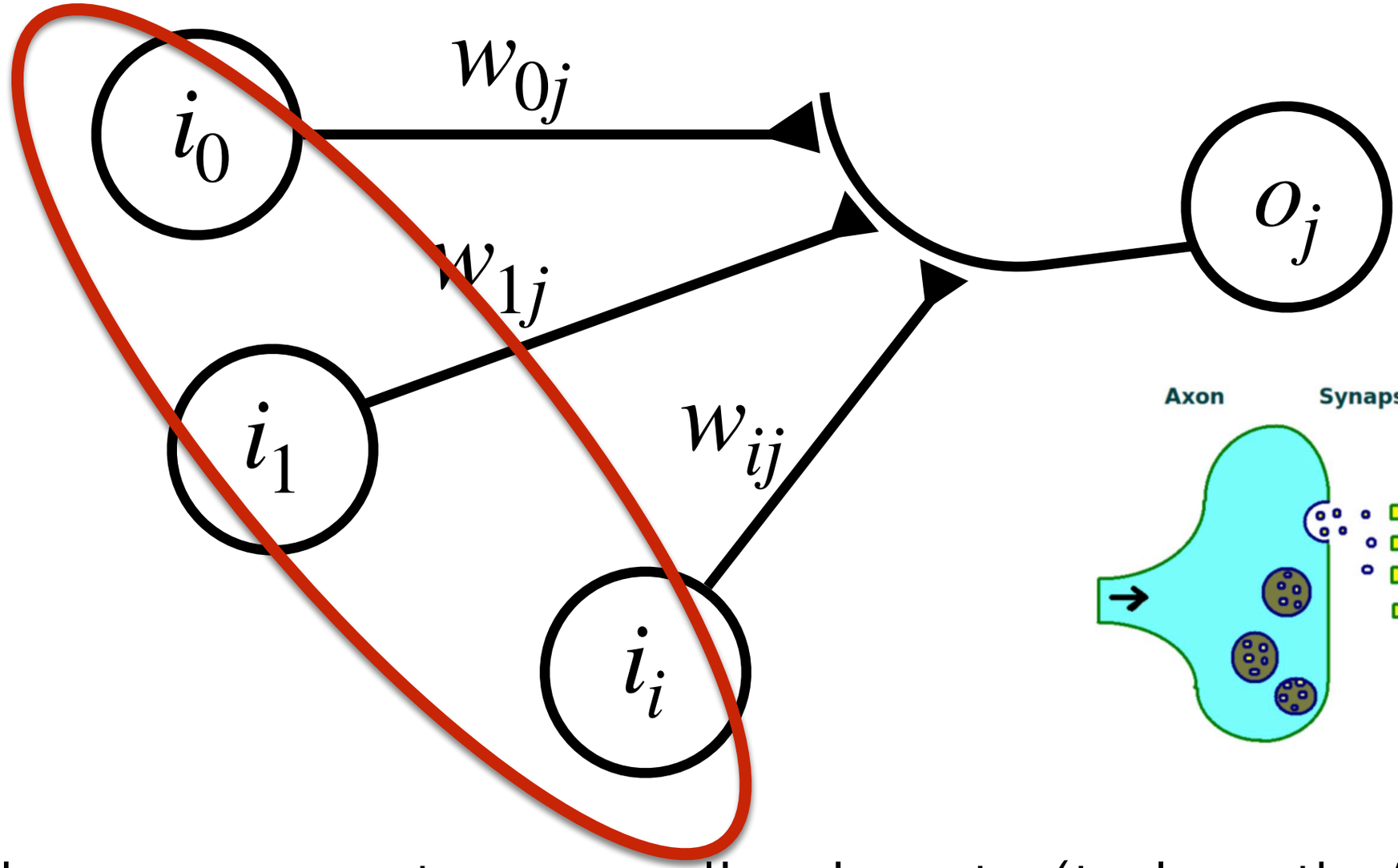


Oja's Rule is equivalent (in the limit) to two steps:

(1) $\Delta w_{ij} = \lambda\, i_i\, o_j$ apply Hebbian Learning to all weights given a pattern

(2) normalize weights after weight updates - but which weights?

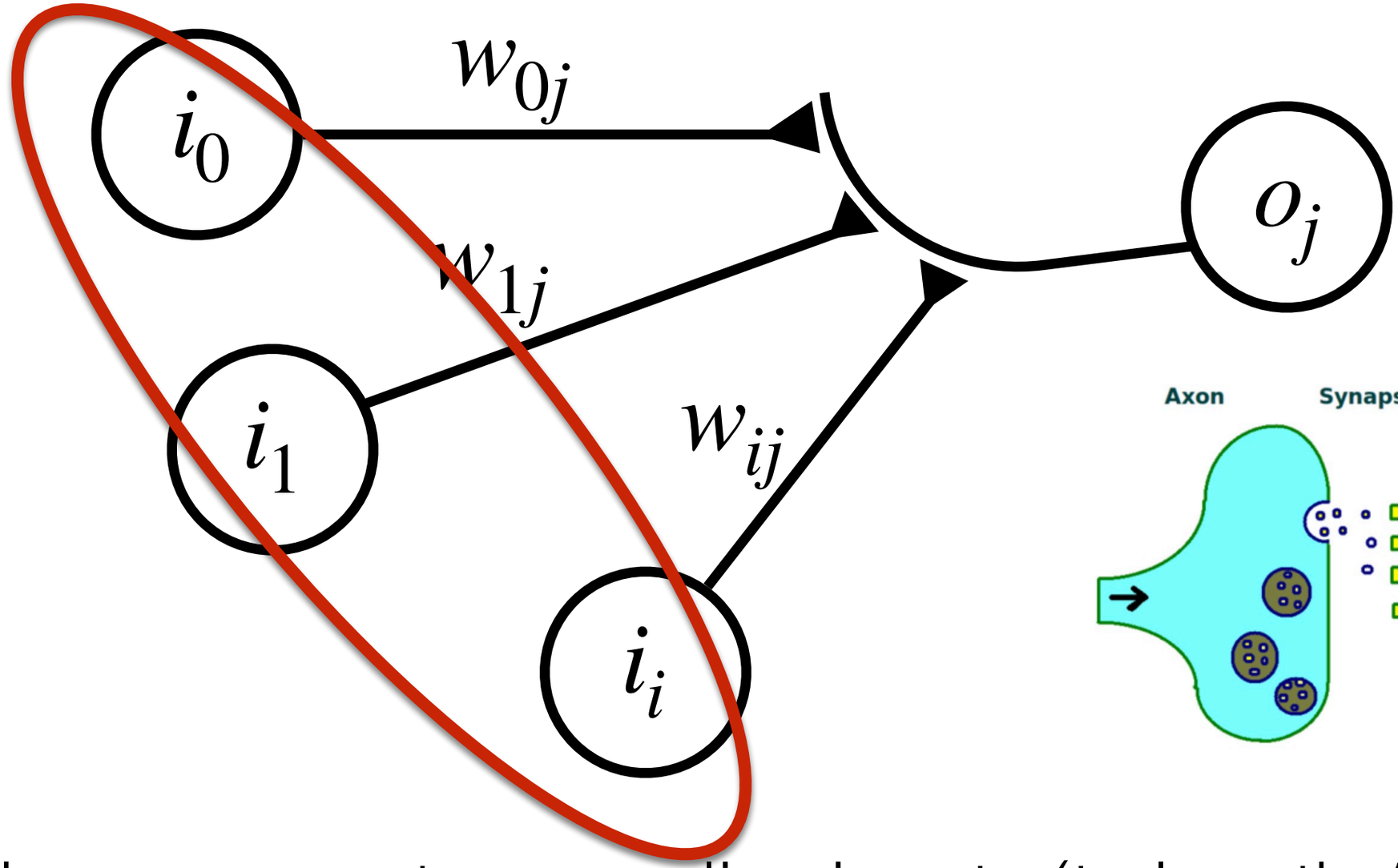
vector of weights to output unit j $\mathbf{w_j} = \dfrac{\mathbf{w_j}}{\|\mathbf{w_j}\|}$ norm of vector } result is vector of length 1



also common to normalize inputs (to length 1)

$$\mathbf{i} = \frac{\mathbf{i}}{\|\mathbf{i}\|}$$

divisive normalization is common in the brain (canonical neural computation)

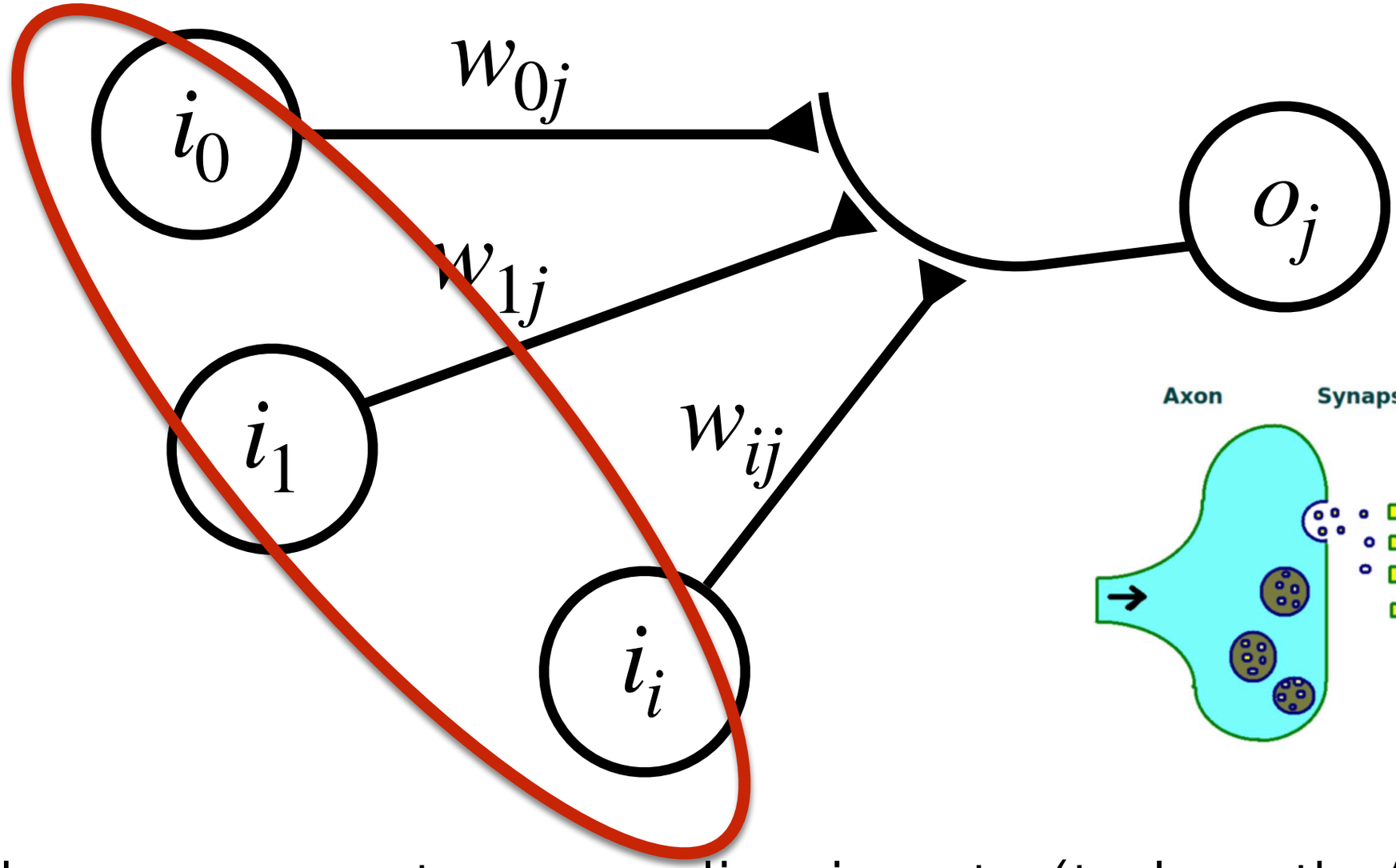


also common to normalize inputs (to length 1)

$$\mathbf{i} = \frac{\mathbf{i}}{\|\mathbf{i}\|}$$

with the inputs AND weights normalized

$$o_j = \sum i_i w_{ij} = \mathbf{i} \cdot \mathbf{w_j} = \|\mathbf{i}\| \, \|\mathbf{w_j}\| cos(\theta)$$

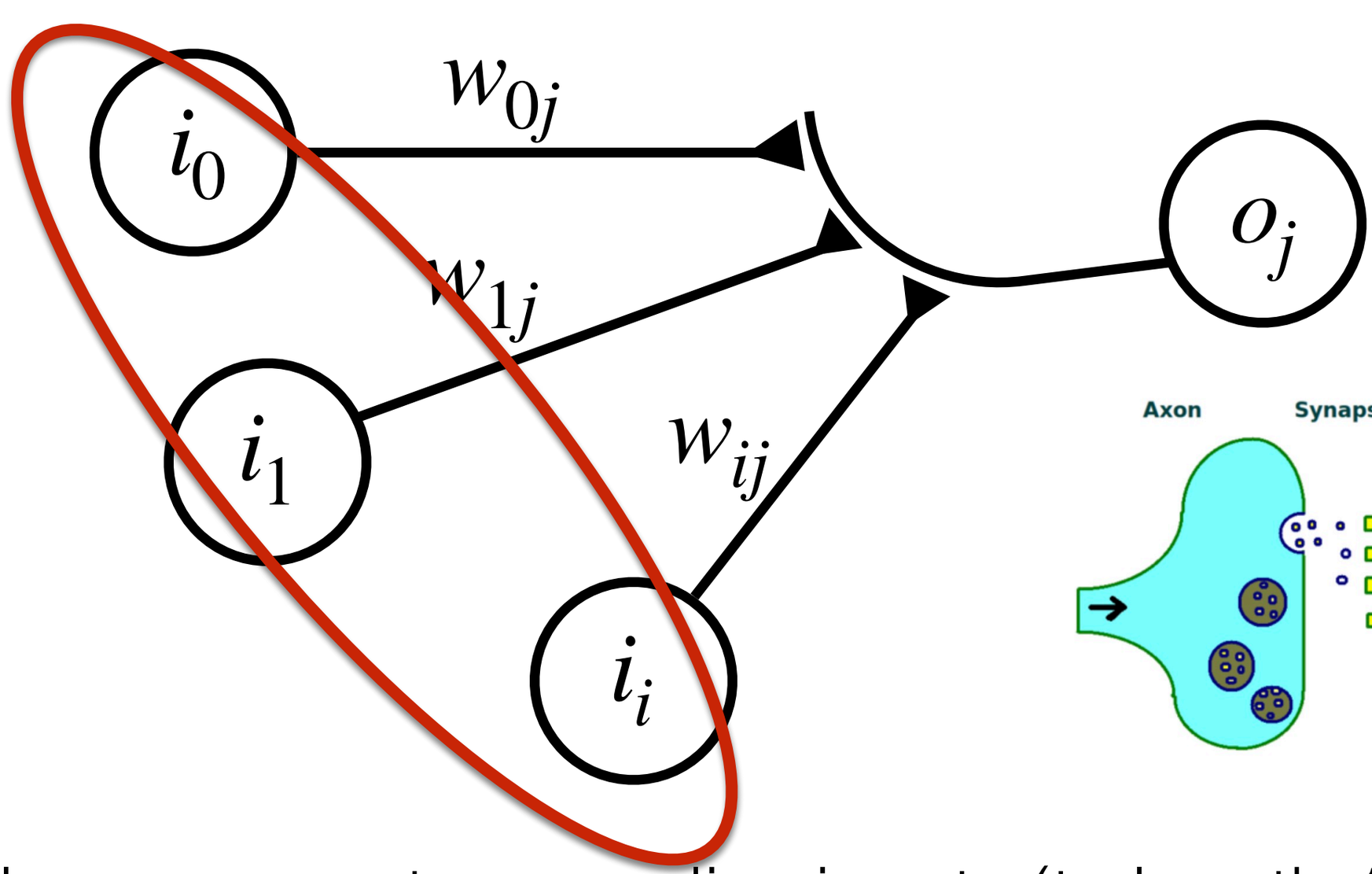


also common to normalize inputs (to length 1)

$$\mathbf{i} = \frac{\mathbf{i}}{\|\mathbf{i}\|}$$

with the inputs AND weights normalized

$$o_j = \sum i_i w_{ij} = \mathbf{i} \cdot \mathbf{w_j} = \cancel{\|\mathbf{i}\|}^{1} \cancel{\|\mathbf{w_j}\|}^{1} cos(\theta)$$



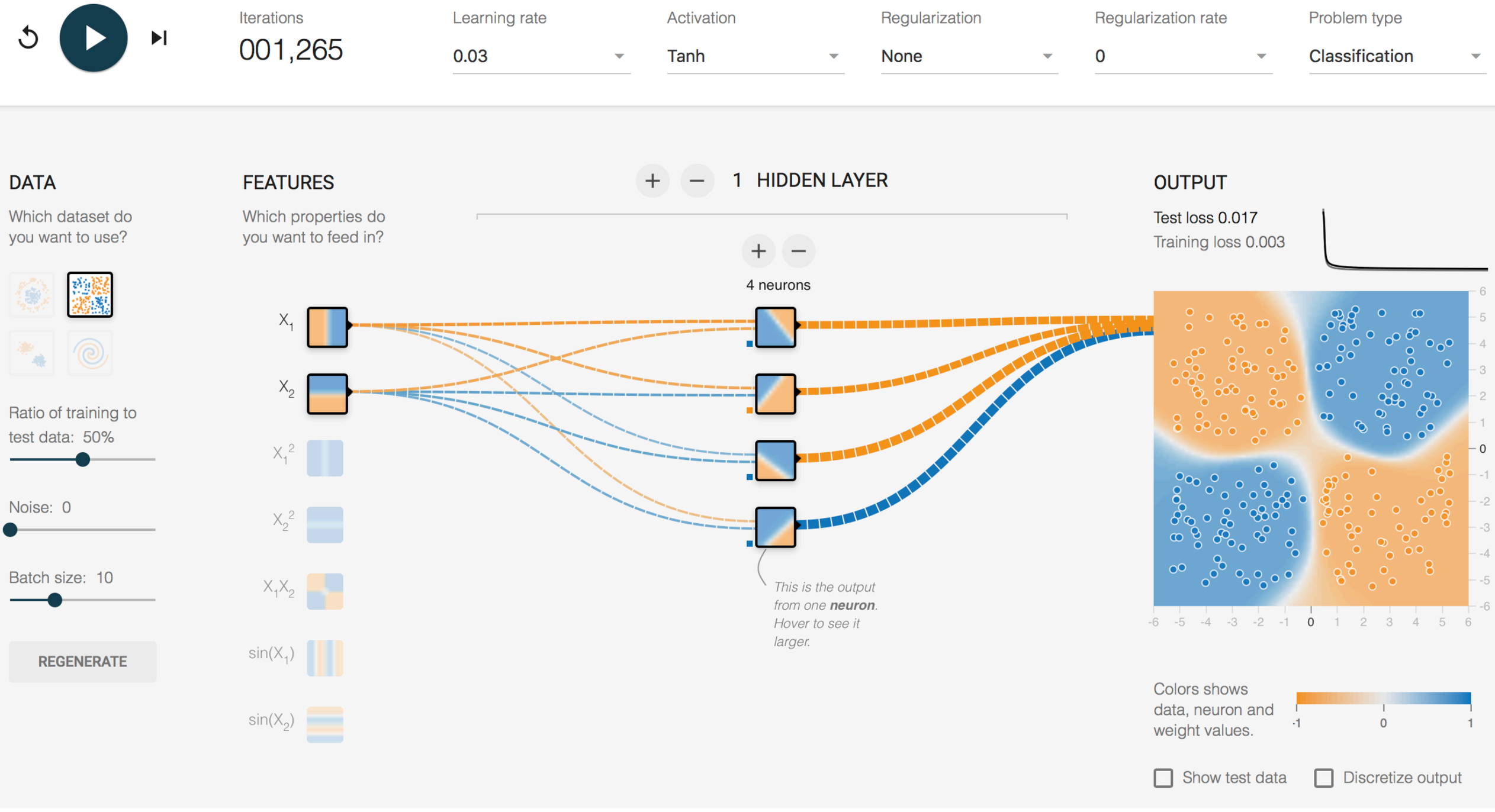
also common to normalize inputs (to length 1)

$$\mathbf{i} = \frac{\mathbf{i}}{\|\mathbf{i}\|}$$

with the inputs AND weights normalized

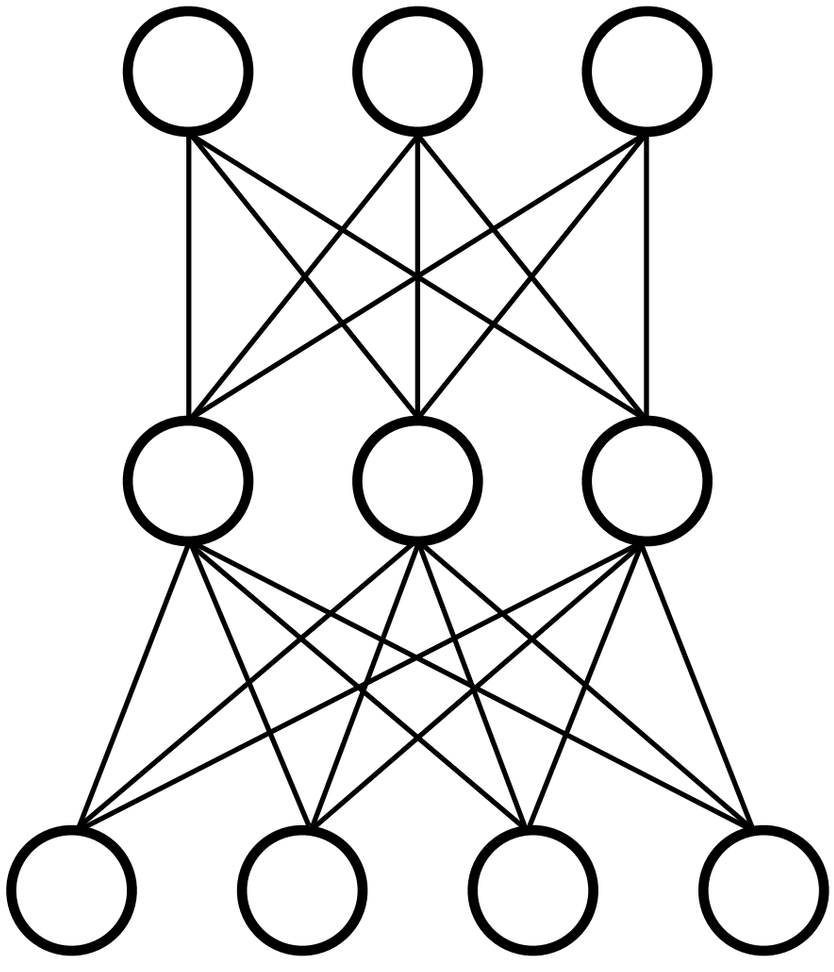
$$o_j = \sum i_i w_{ij} = \mathbf{i} \cdot \mathbf{w_j} = cos(\theta)$$

http://playground.tensorflow.org/

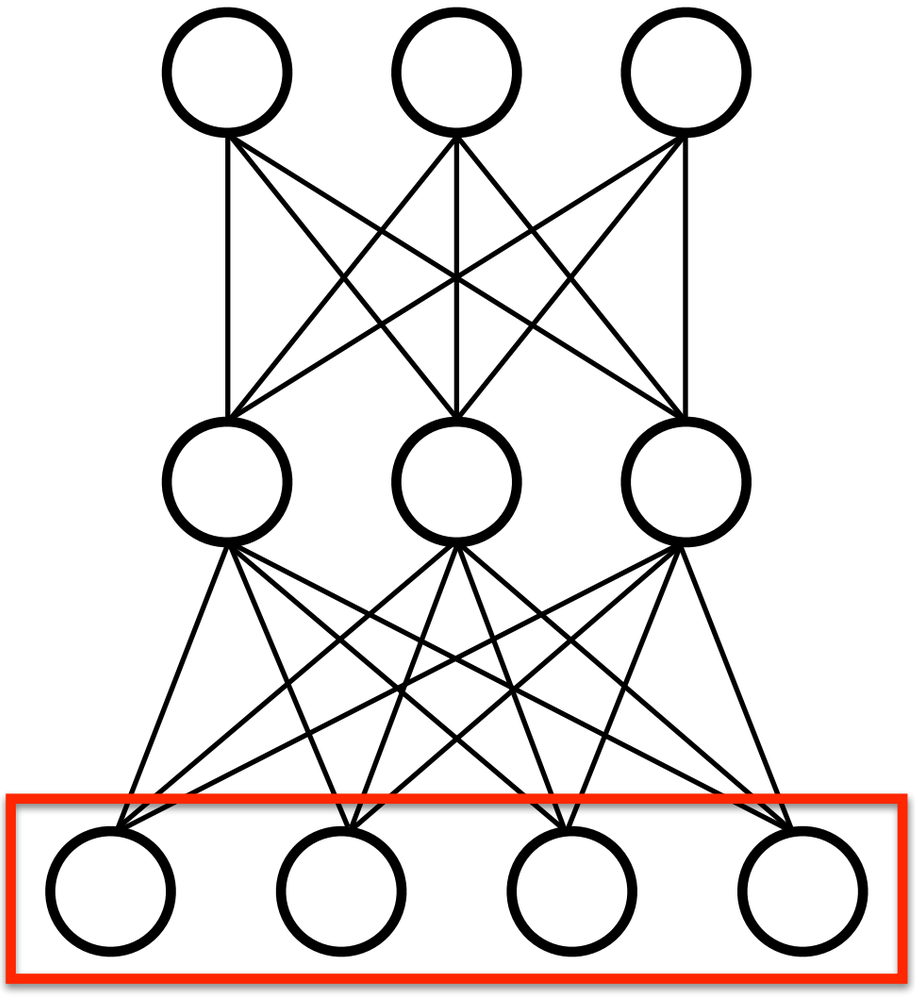


exploring how neural networks learn

examples of modeling with neural networks

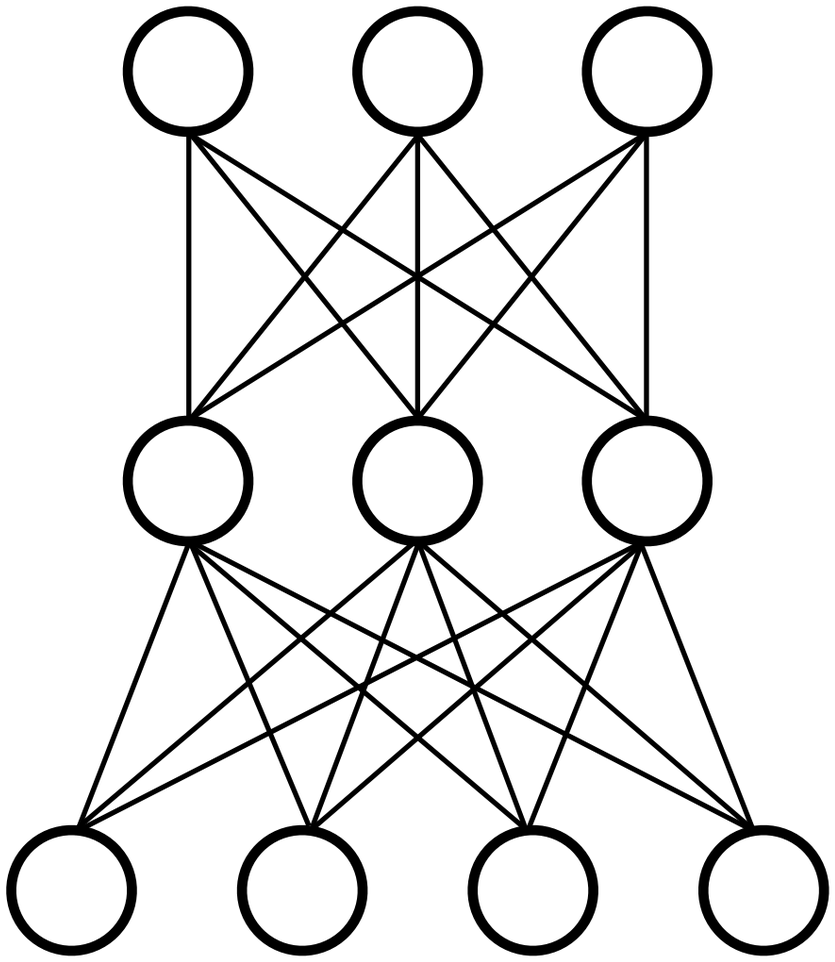


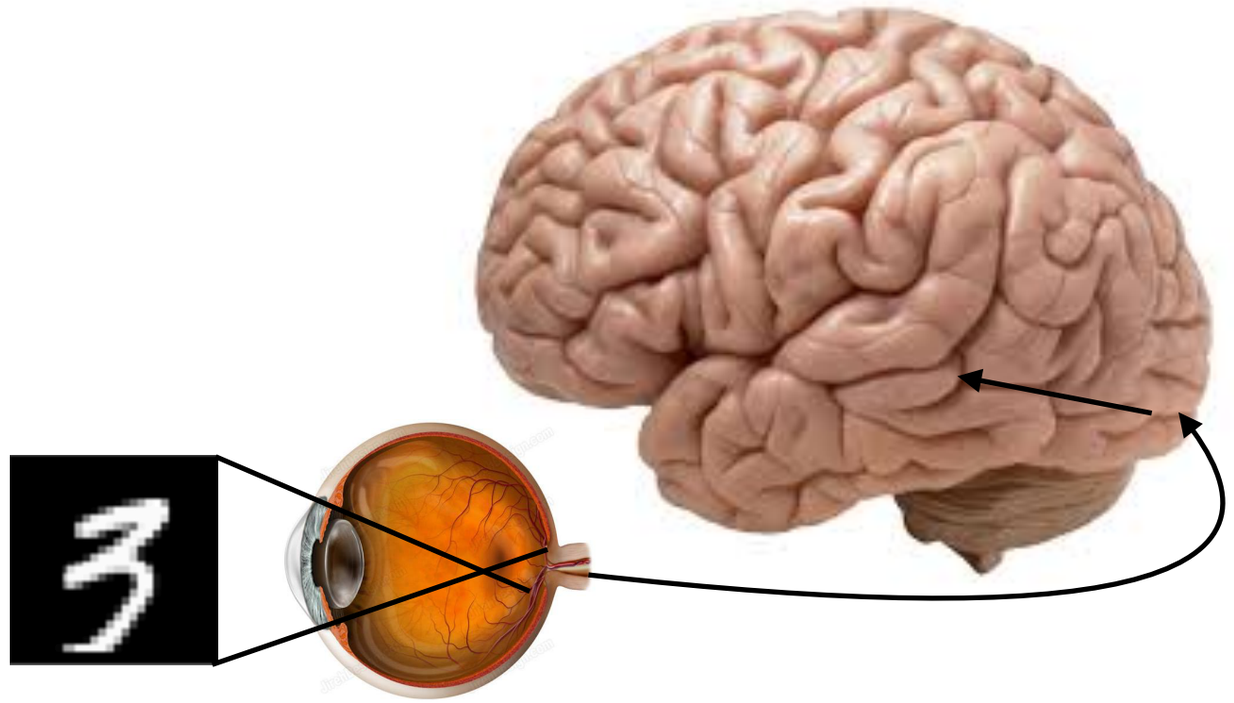
examples of modeling with neural networks



first, what the inputs could correspond to

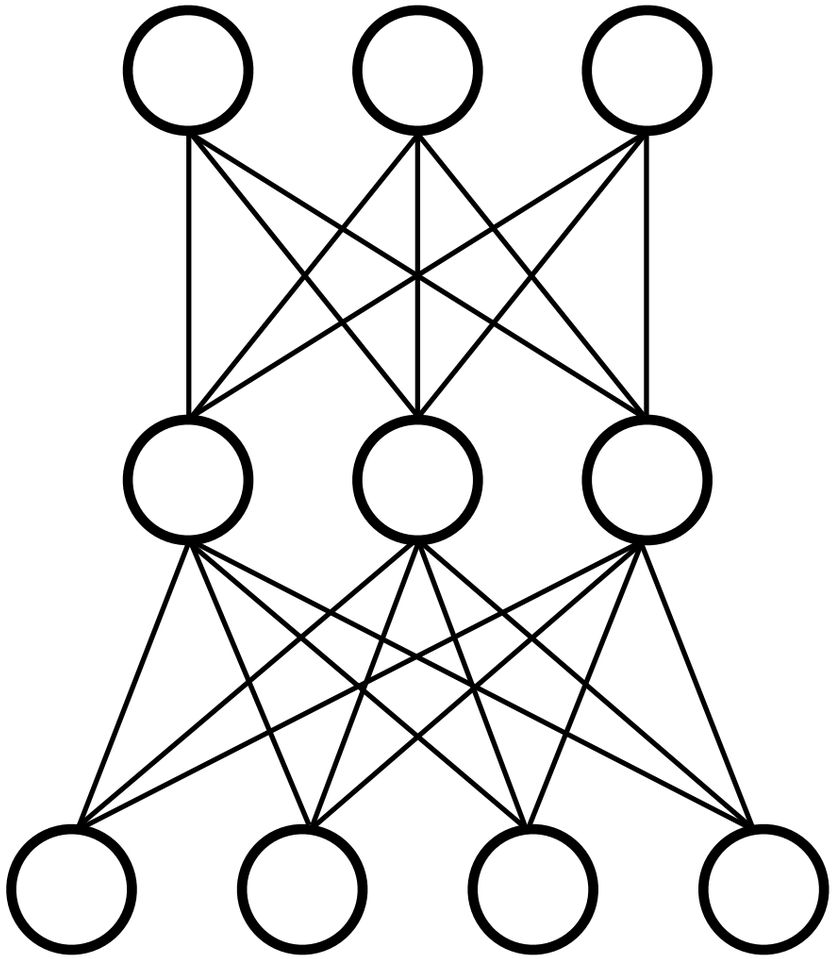
examples of modeling with neural networks

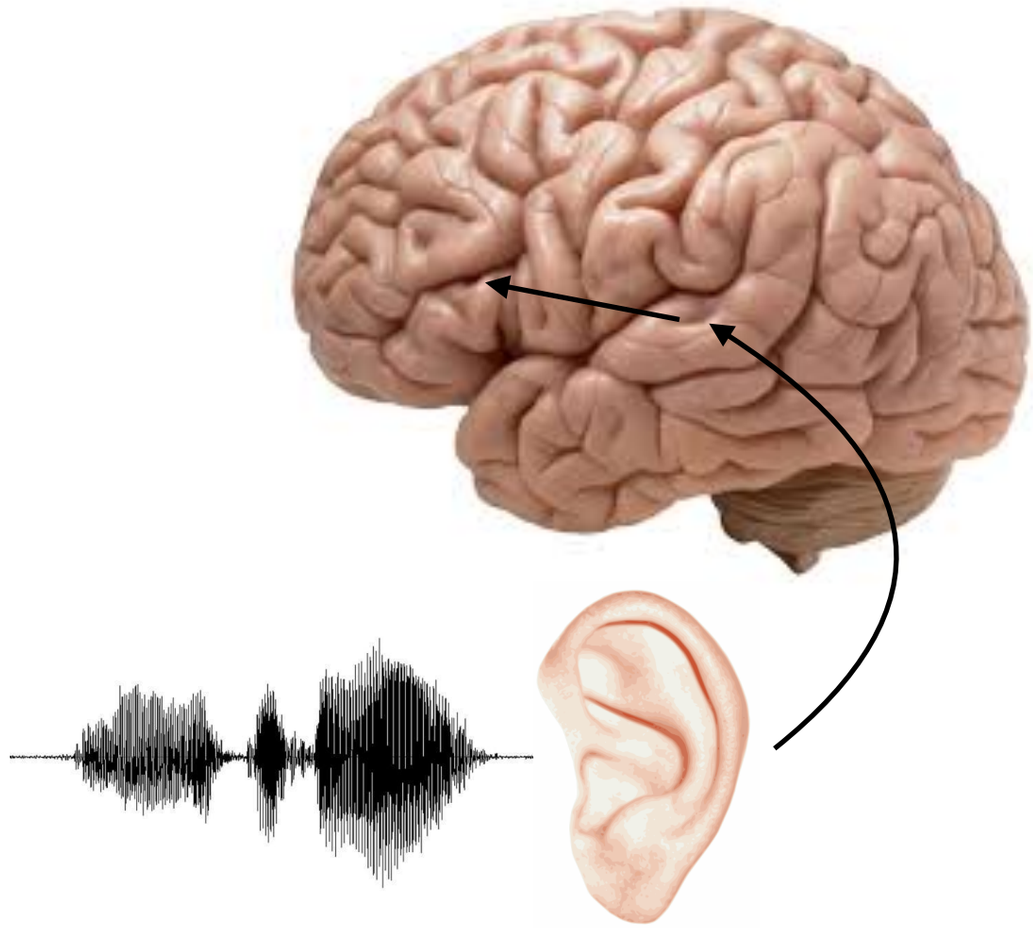


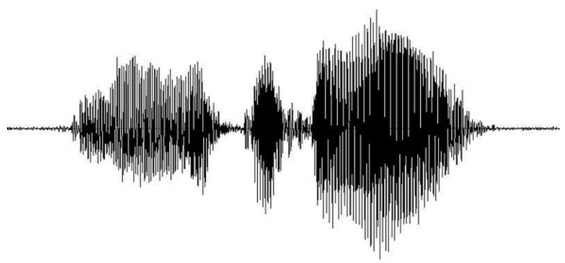


“inputs” to neural network have a close connection to the “inputs” to the organism

examples of modeling with neural networks

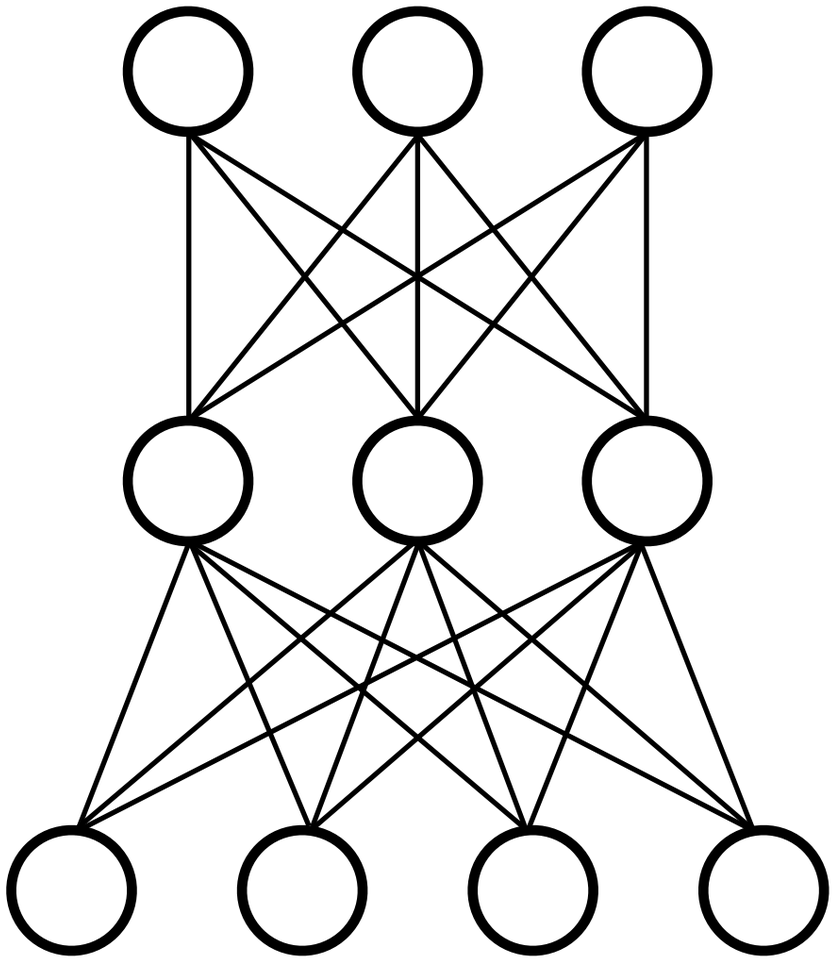






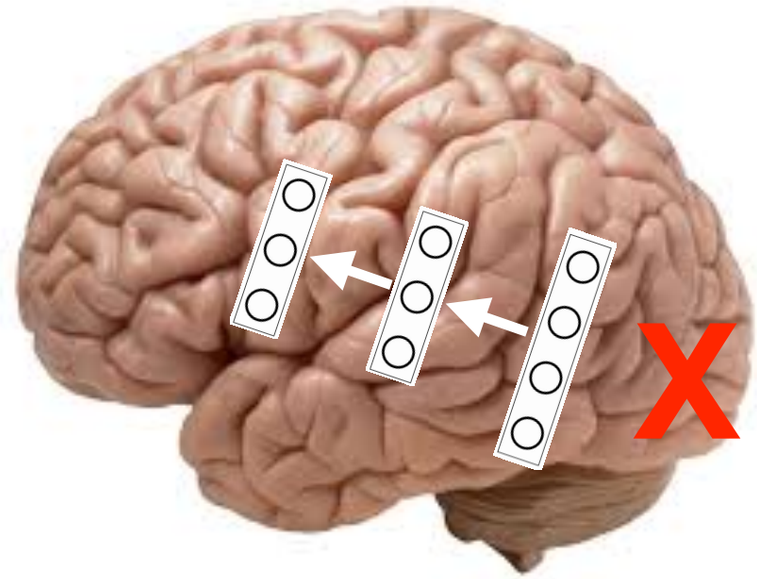
"inputs" to neural network have a close connection to the "inputs" to the organism

examples of modeling with neural networks



“g” “r” “i” “n”

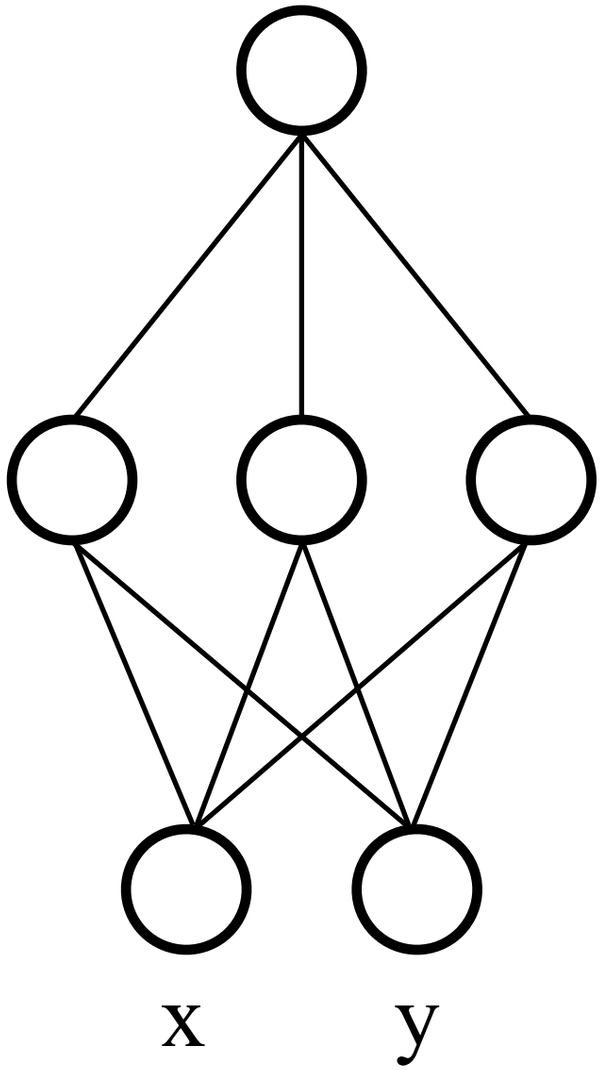
not modeling visual transformation
from retina to a letter representation,
we’re starting with the letter representation

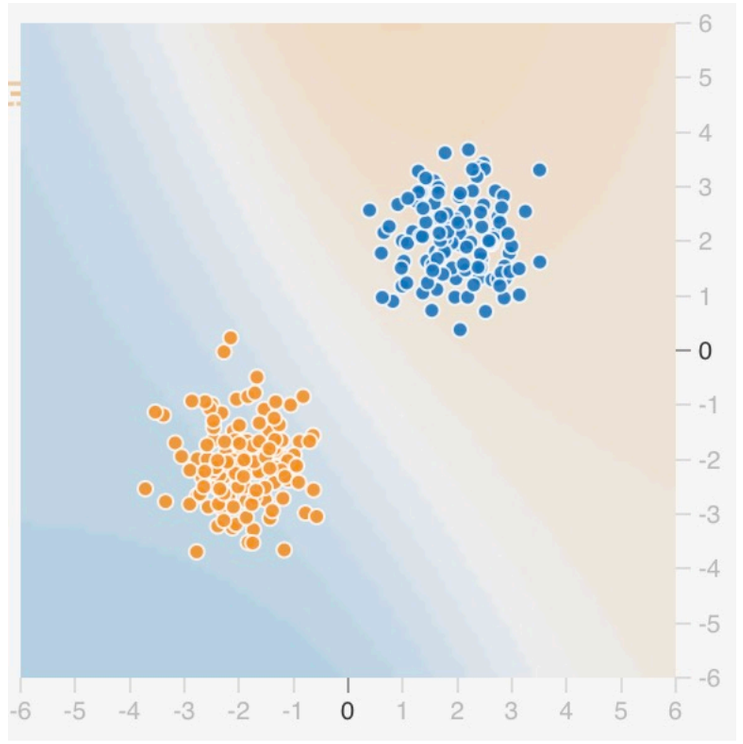


“inputs” to neural network
can also be more abstract

examples of modeling with neural networks

http://playground.tensorflow.org

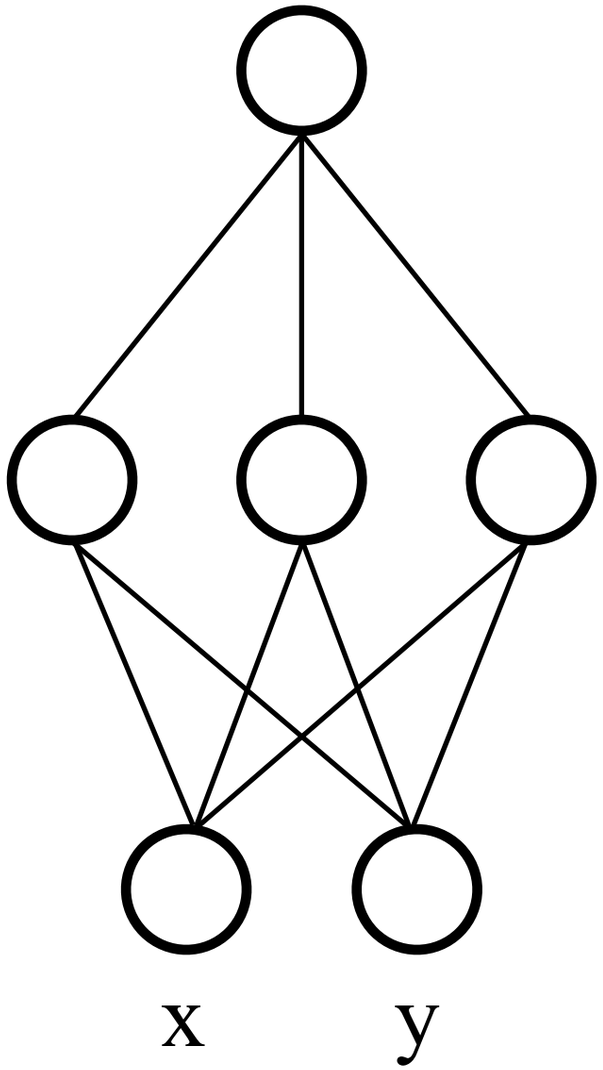


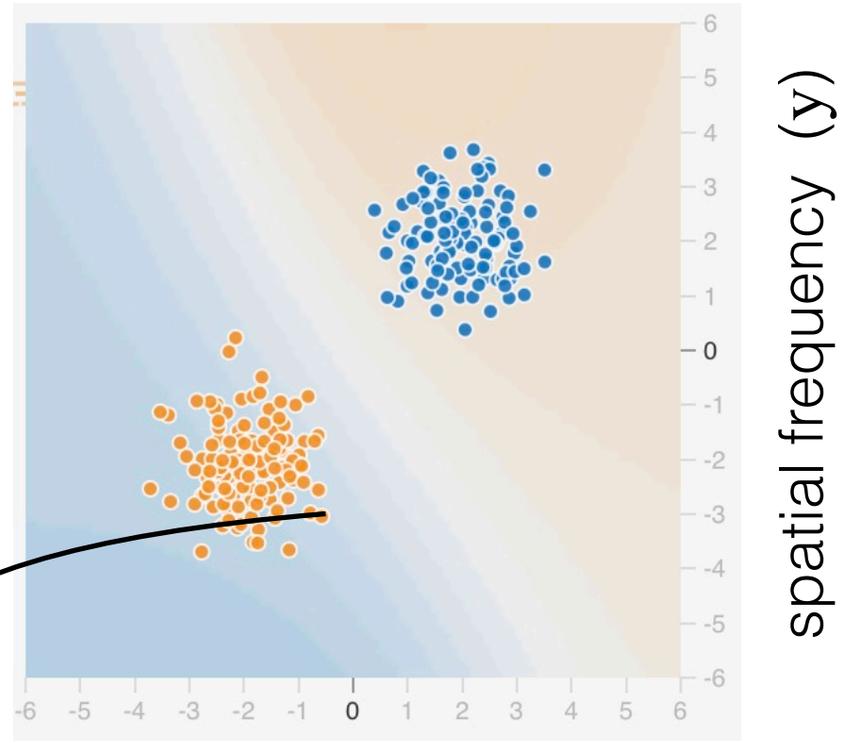


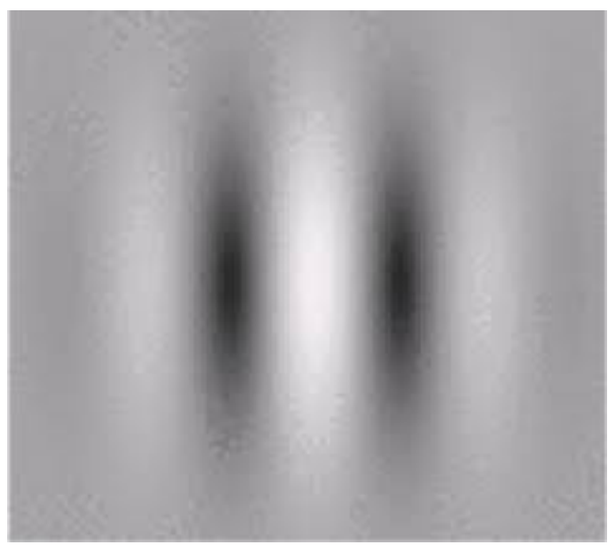
dimension 1 (x)

what do the x and y inputs correspond to?

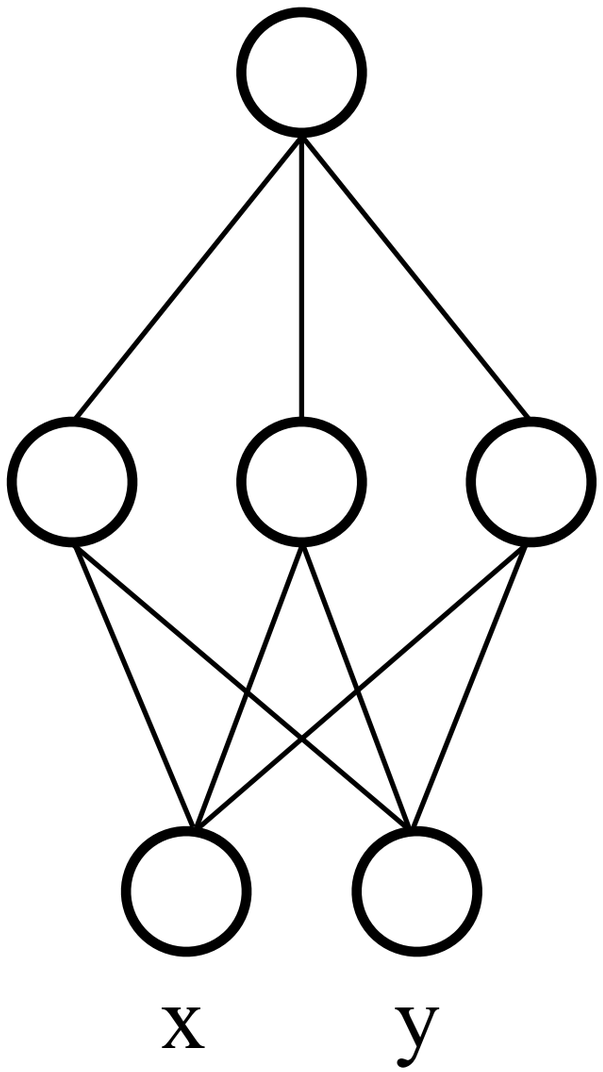
examples of modeling with neural networks

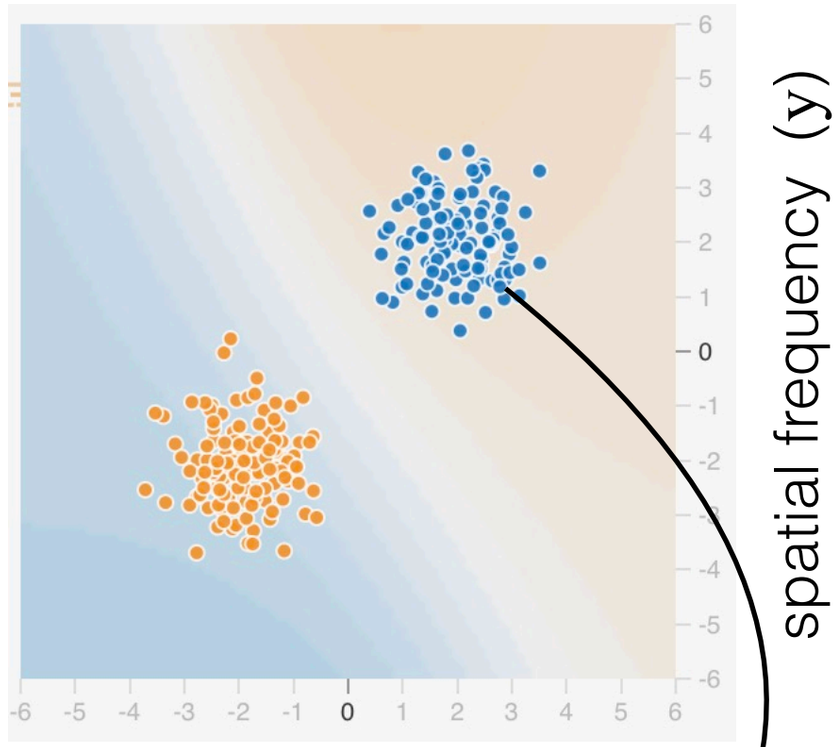


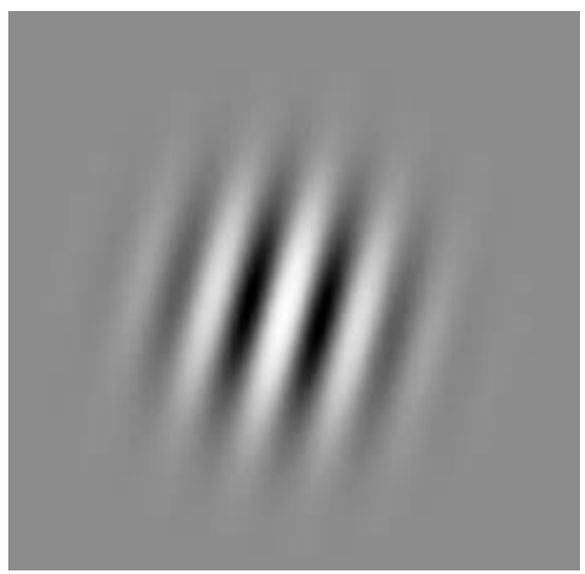




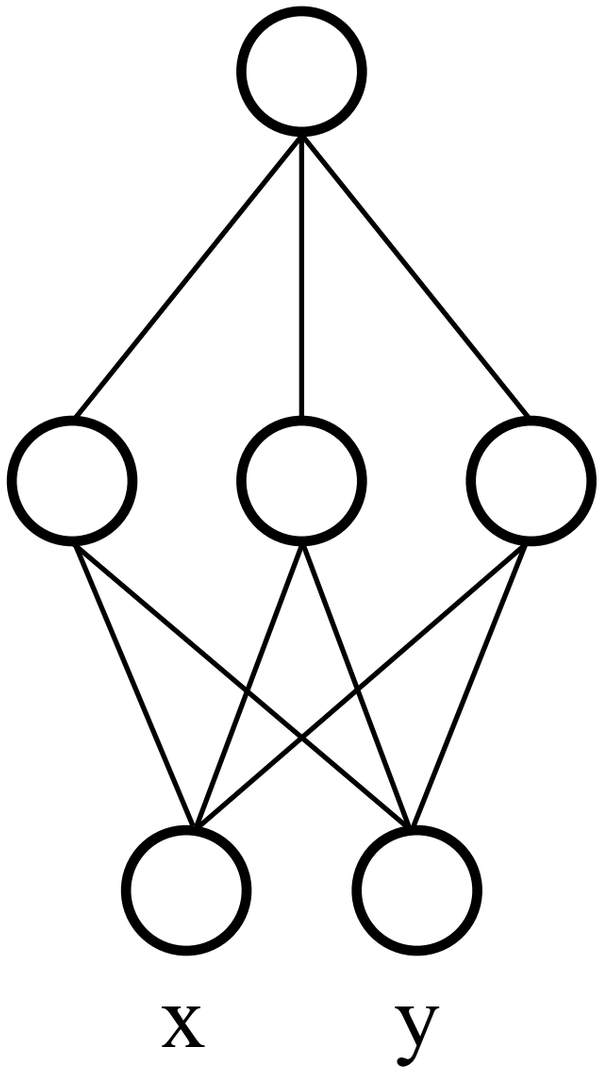
examples of modeling with neural networks

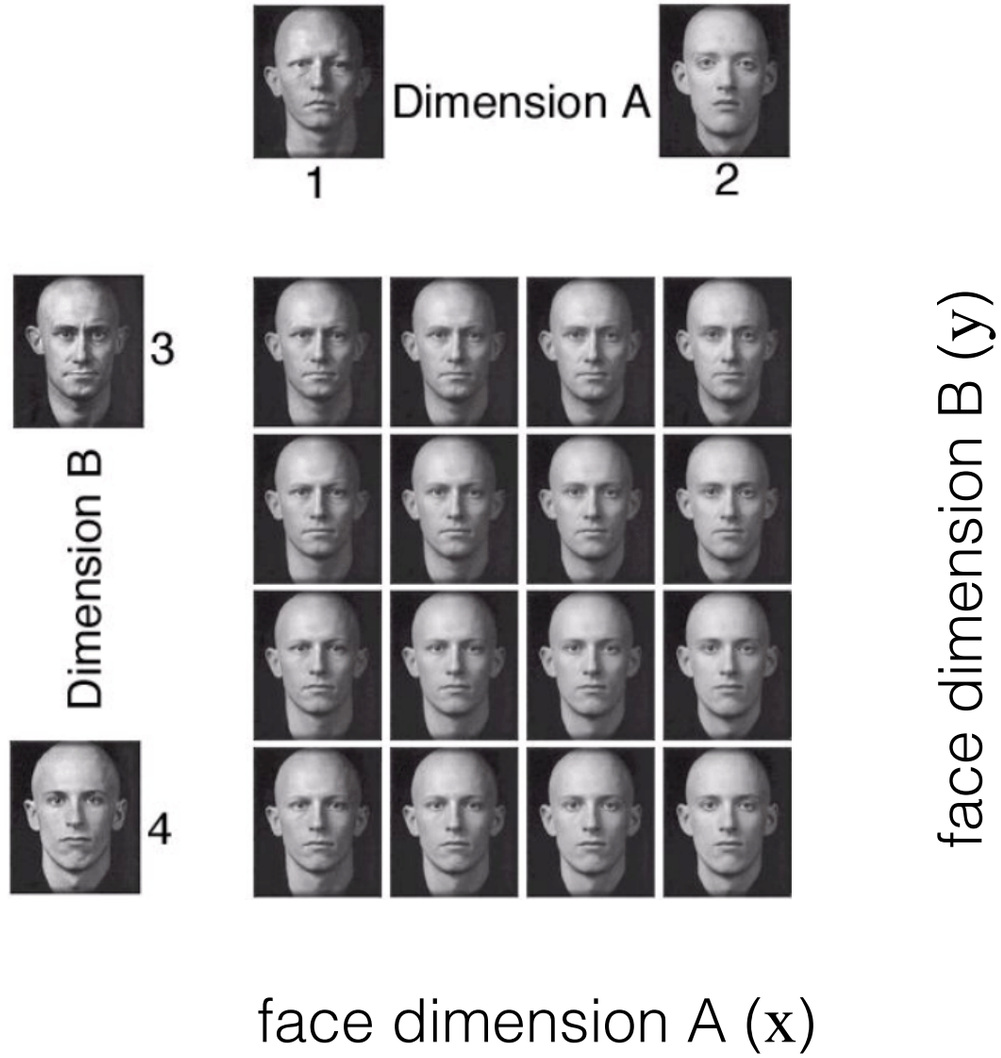




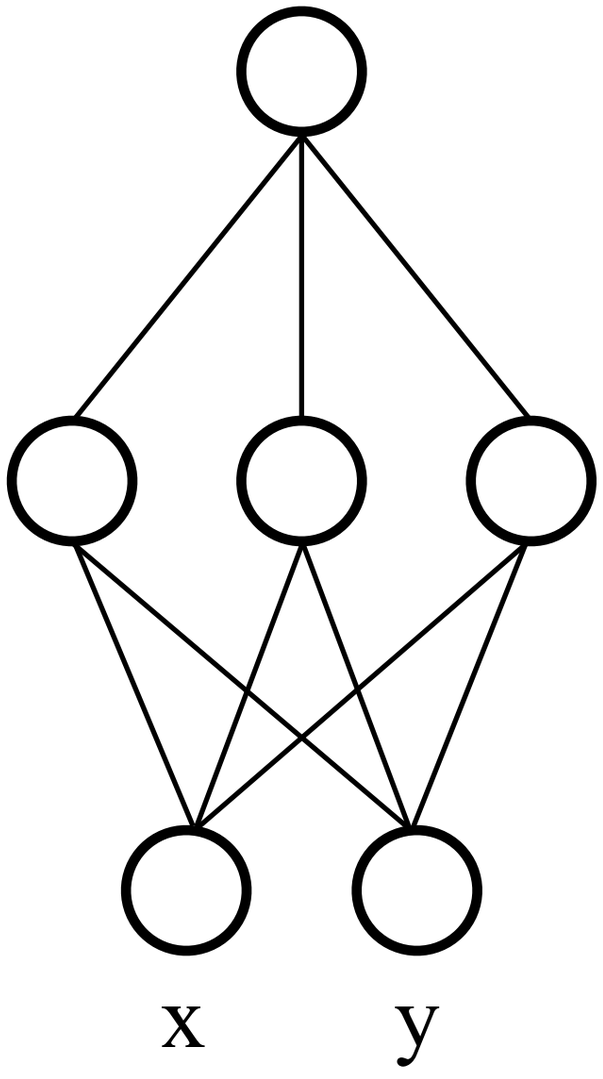


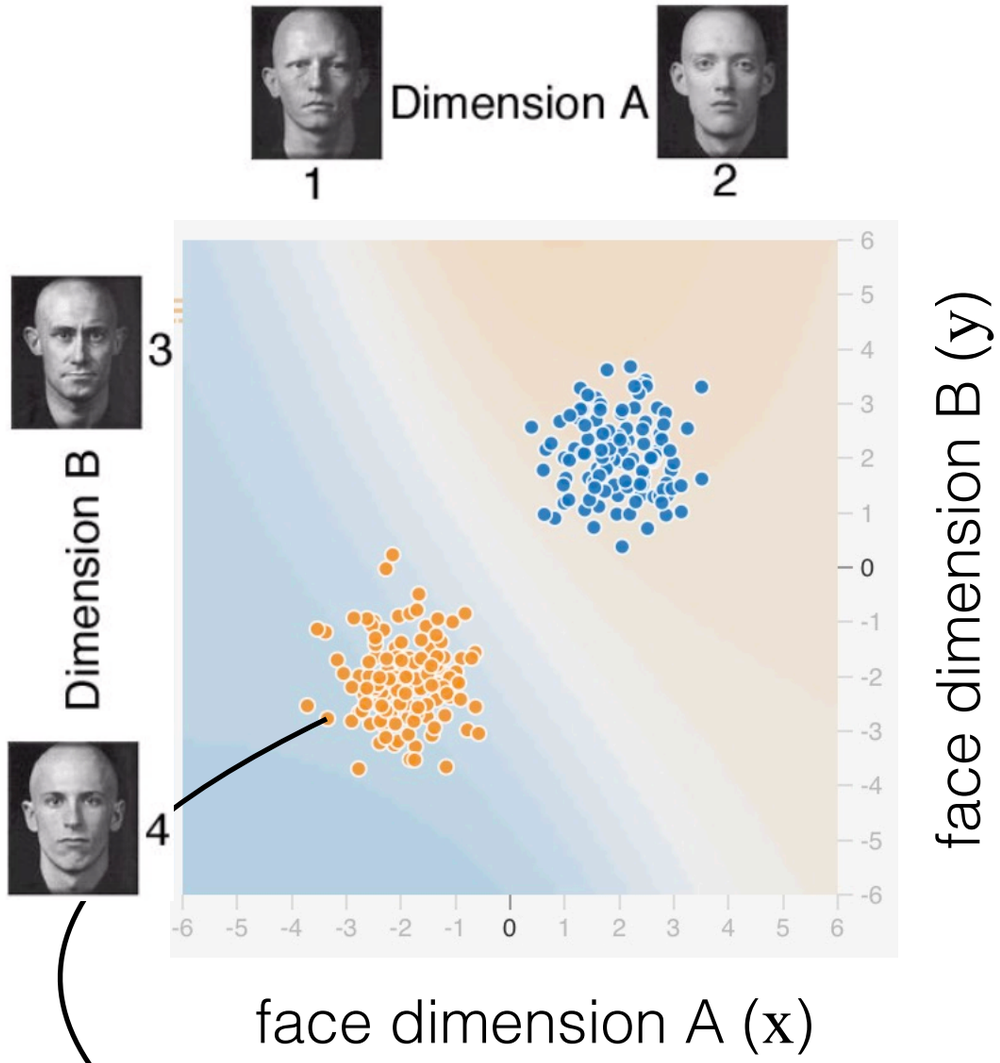
examples of modeling with neural networks

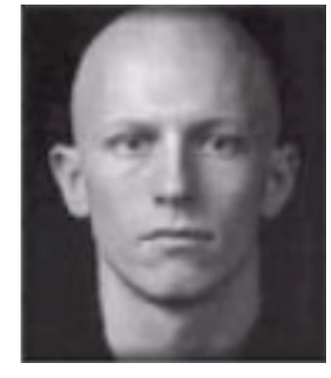




examples of modeling with neural networks







examples of modeling with neural networks