Implementing ANNs with TensorFlow

Session 02 - Perceptron & MLP

Organizational Stuff

- Courses “Neuroinformatics” and “Machine Learning” are not a prerequisite!
- Go to Sahar’s Tutorial on Wednesday (12:00 - 14:00) to learn how to use Colab!
- Afterwards write me an Email, whether you want to do your homework submissions via **Stud.IP or Colab!**

Agenda

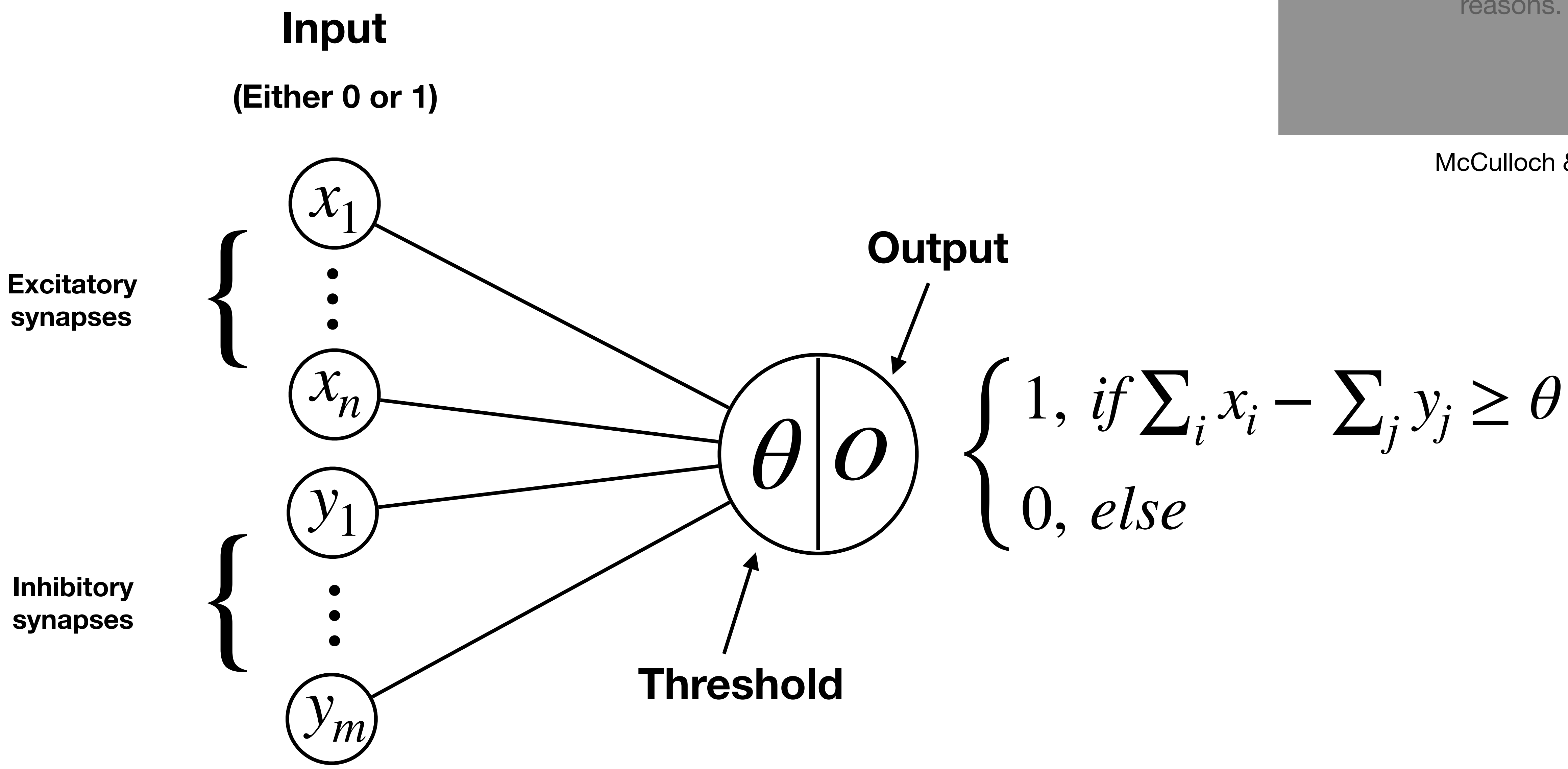
1. McCulloch-Pitts Model
2. Perceptron
3. Perceptron Learning
4. Multi-Layer Perceptron

McCulloch-Pitts

McCulloch-Pitts Neuron

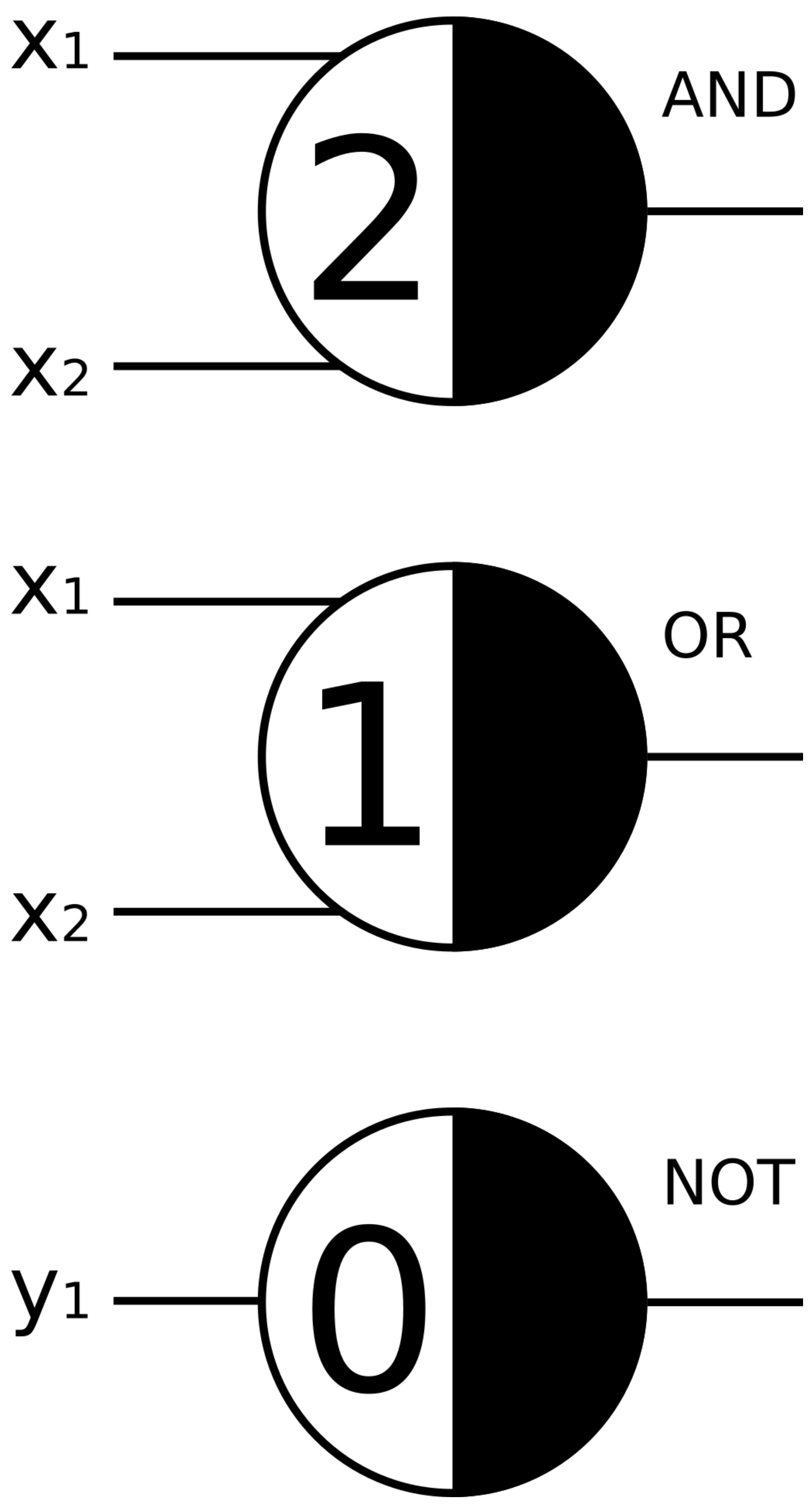
Not available due to copyright reasons.

McCulloch & Pitts



$$\begin{cases} 1, \text{ } if \sum_i x_i - \sum_j y_j \geq \theta \\ 0, \text{ } else \end{cases}$$

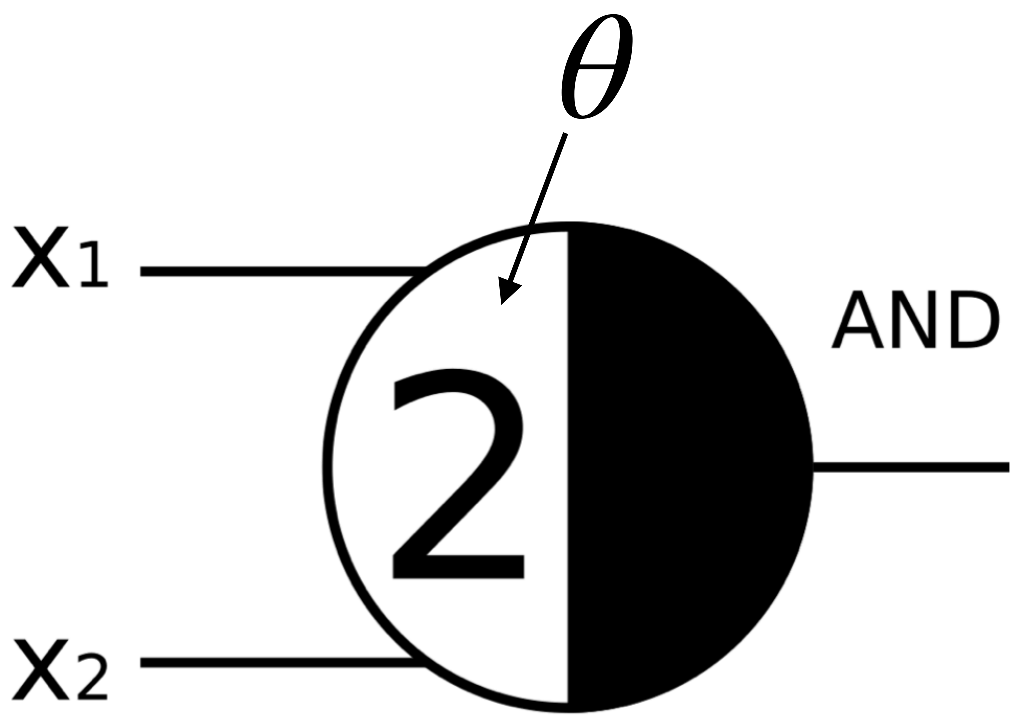
Example: Logical Gates



Logical AND Gate

Logical AND

x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1



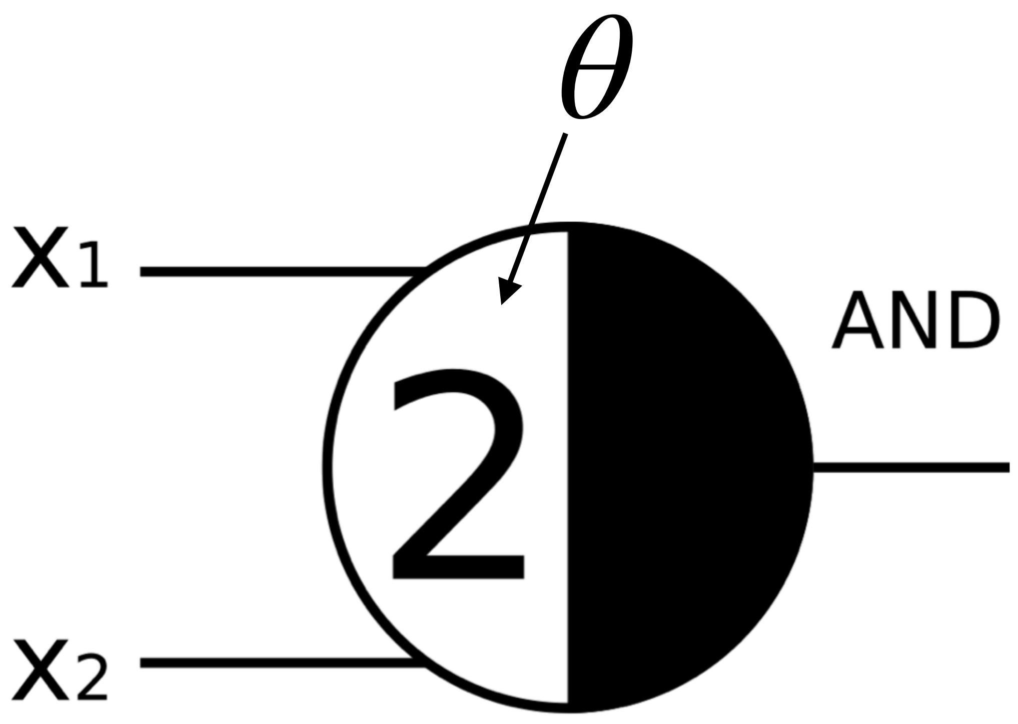
$$o = \begin{cases} 1, \; if \sum_i x_i \geq \theta \\ 0, \; else \end{cases}$$

$$0 + 0 < 2 \rightarrow o = 0$$
$$1 + 0 < 2 \rightarrow o = 0$$
$$0 + 1 < 2 \rightarrow o = 0$$
$$1 + 1 \geq 2 \rightarrow o = 1$$

Logical AND Gate

Logical AND

x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1



$$o = \begin{cases} 1, & if \sum_i x_i \geq \theta \\ 0, & else \end{cases}$$

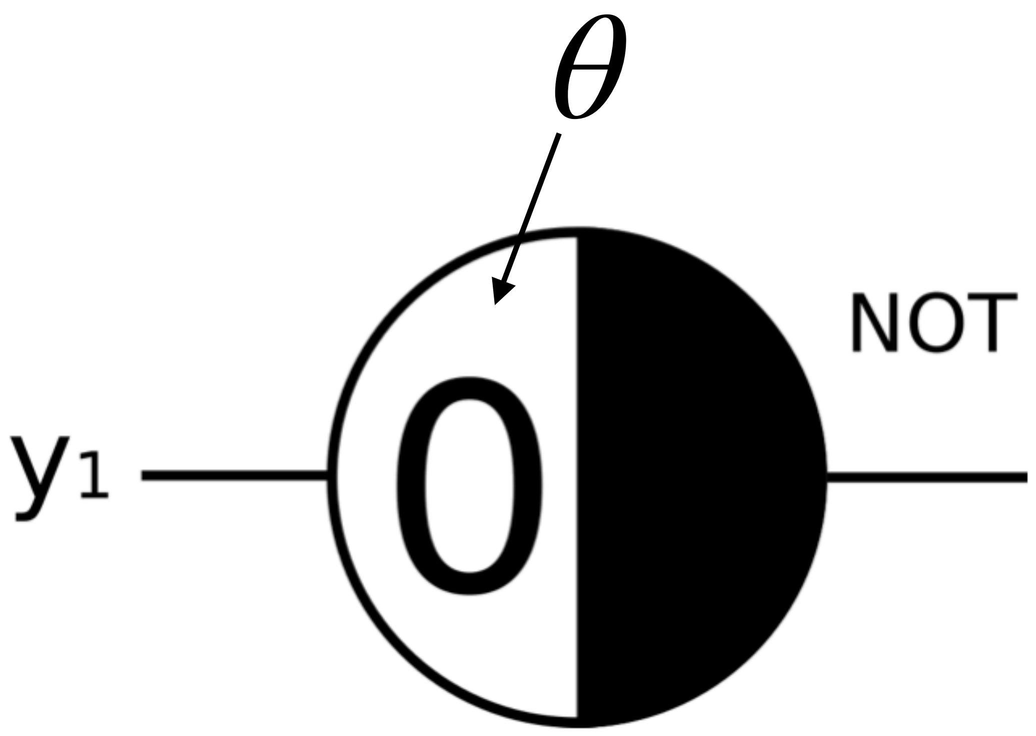
$$0 + 0 < 2 \rightarrow o = \boxed{0}$$
$$1 + 0 < 2 \rightarrow o = \boxed{0}$$
$$0 + 1 < 2 \rightarrow o = \boxed{0}$$
$$1 + 1 \geq 2 \rightarrow o = \boxed{1}$$

✓

Logical NO Gate

Logical NO

y_1	$\neg y_1$
0	1
1	0



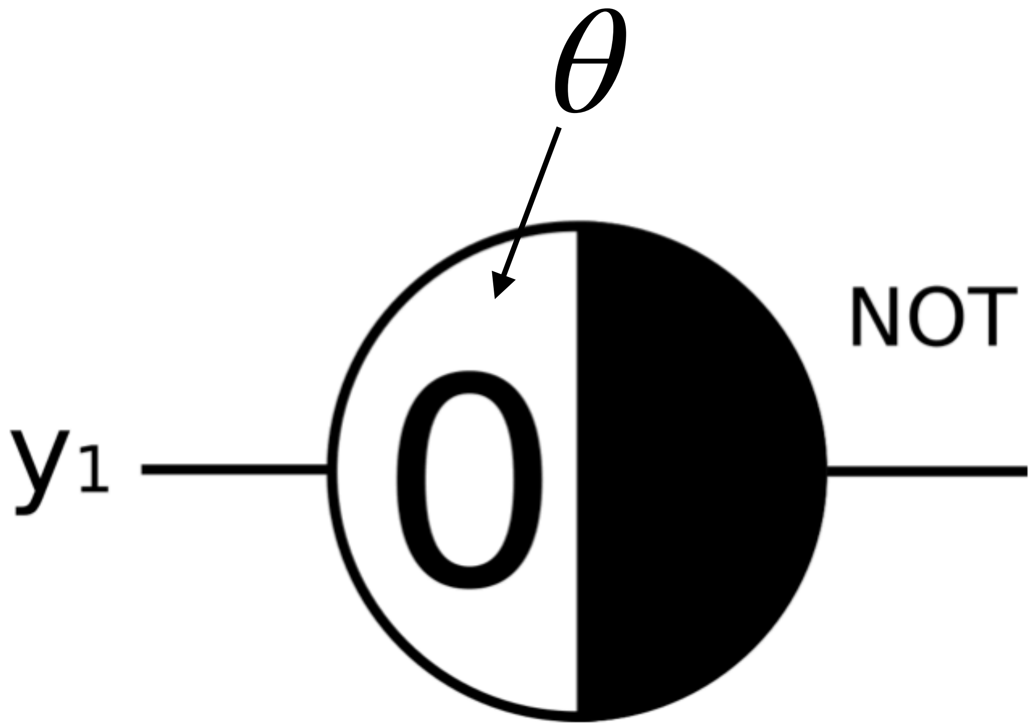
$$o = \begin{cases} 1, \; if - \sum_j y_j \geq \theta \\ 0, \; else \end{cases}$$

$$-0 \geq 0 \rightarrow o = 1$$
$$-1 < 0 \rightarrow o = 0$$

Logical NO Gate

Logical NO

y_1	$\neg y_1$
0	1
1	0

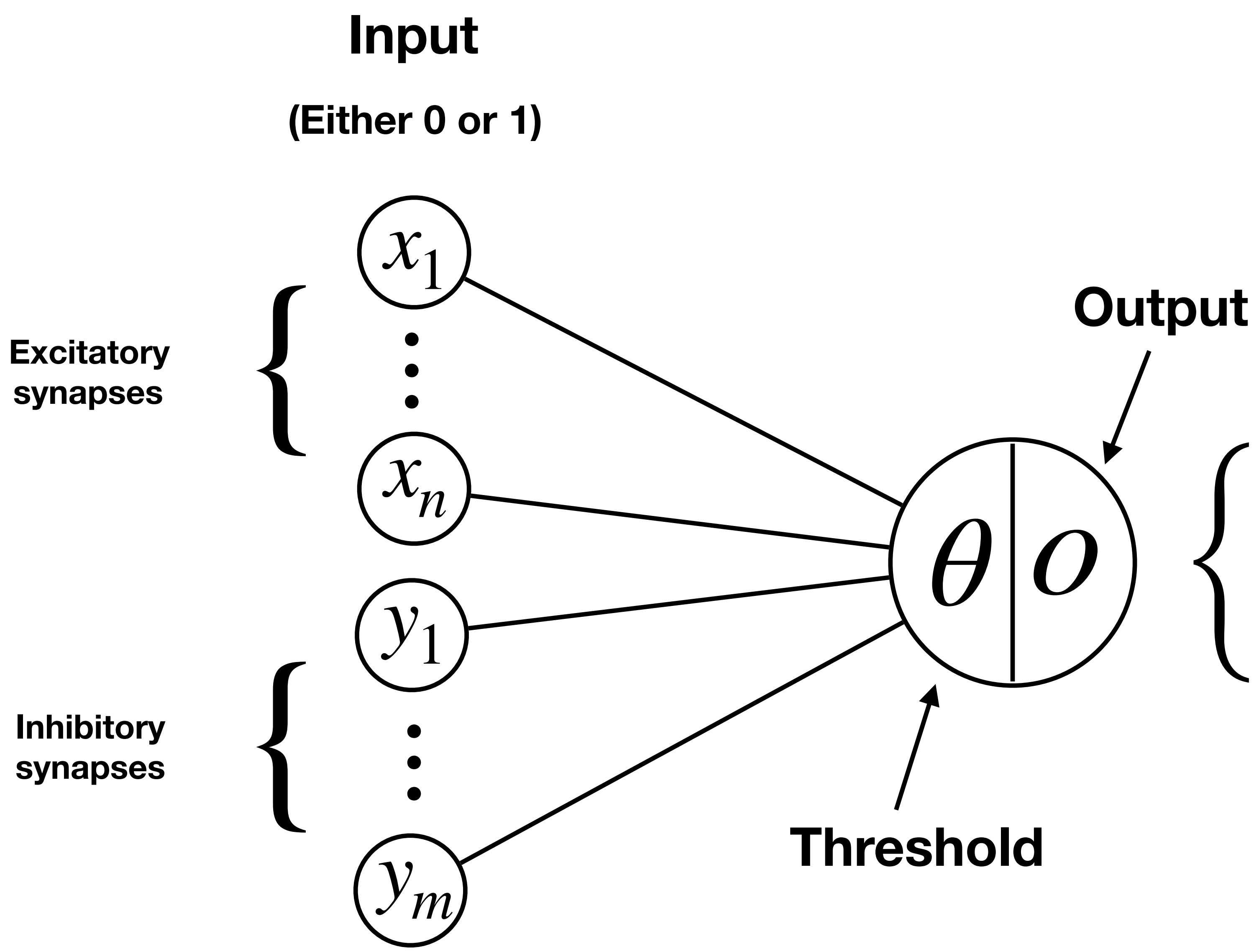


$$o = \begin{cases} 1, \ if - \sum_j y_j \geq \theta \\ 0, \ else \end{cases}$$

$$\begin{aligned} -0 \geq 0 \rightarrow o &= \boxed{1} \\ -1 < 0 \rightarrow o &= \boxed{0} \end{aligned}$$

✓

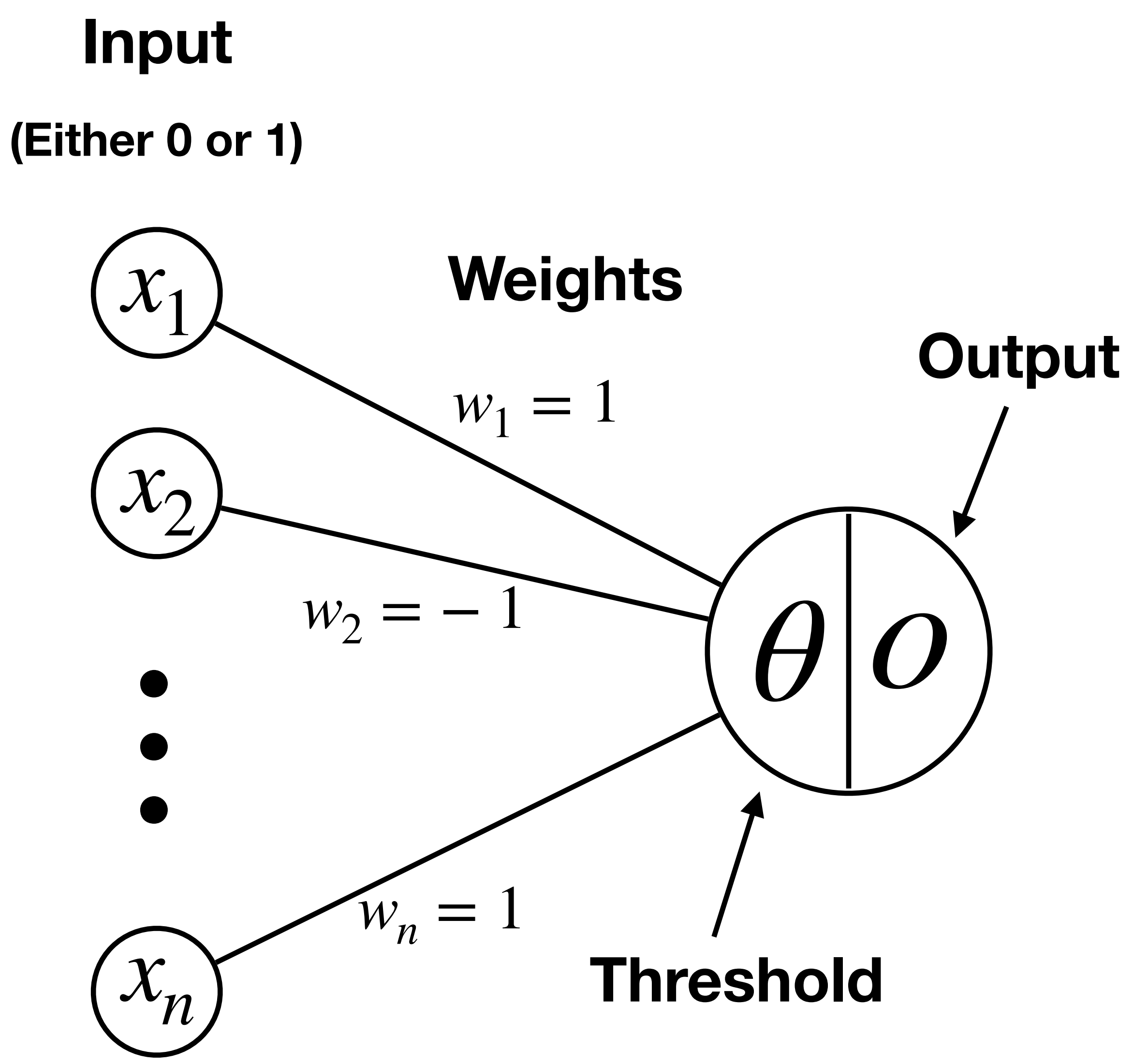
A Different Perspective



$$\begin{cases} 1, \; if \sum_i x_i - \sum_j y_j \geq \theta \\ 0, \; else \end{cases}$$

What if instead of excitatory and inhibitory synapses, we just use one kind of synapse and modulate their influence via a synaptic weight?

A Different Perspective



$$\begin{cases} 1, \ if \sum_i w_i x_i \geq \theta \\ 0, \ else \end{cases}$$

Allows us to make this beautiful simplification.

Perceptron

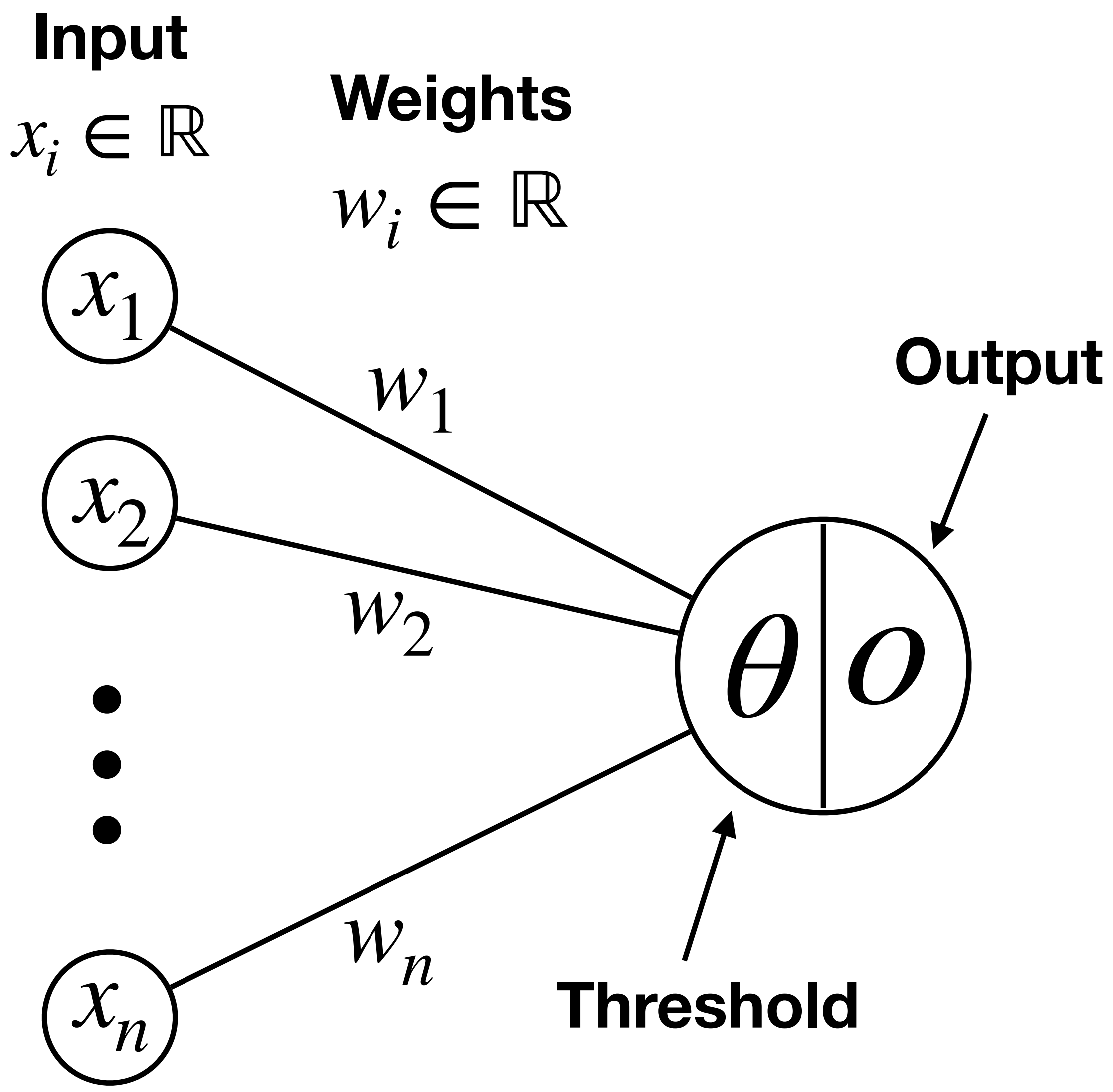
Perceptron

[w1]

Frank Rosenblatt

An even more general model for neural information processing! [fr]

Allows for real-valued inputs and weights!

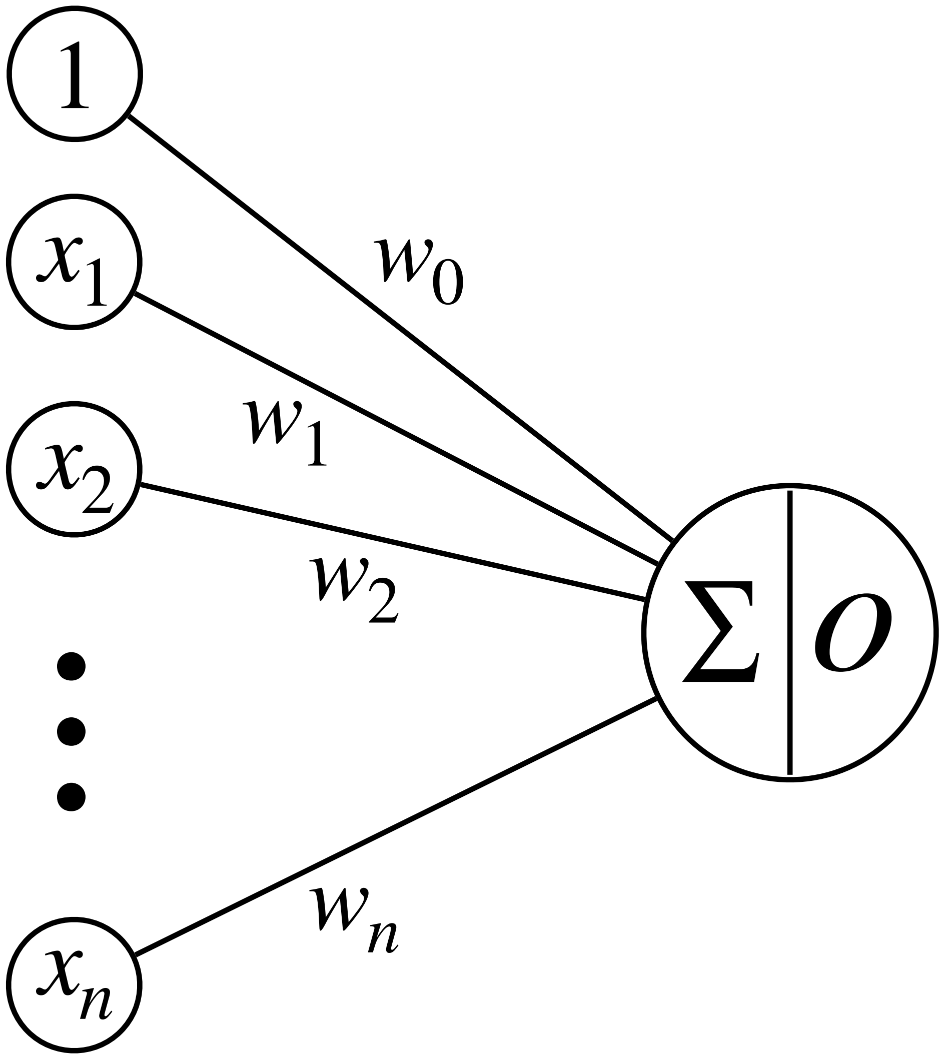


$$\begin{cases} 1, \; if \sum_i w_i x_i \geq \theta \\ 0, \; else \end{cases}$$

Perceptron

Now we turn the threshold into a bias!

$$w_1x_1 + \cdots + w_nx_n \geq \theta$$
$$\Leftrightarrow w_1x_1 + \cdots + w_nx_n \boxed{-\theta} \geq 0$$
$$\Leftrightarrow w_1x_1 + \cdots + w_nx_n \boxed{+ w_0x_0} \geq 0 \ (w_0 = -\theta, x_0 = 1)$$



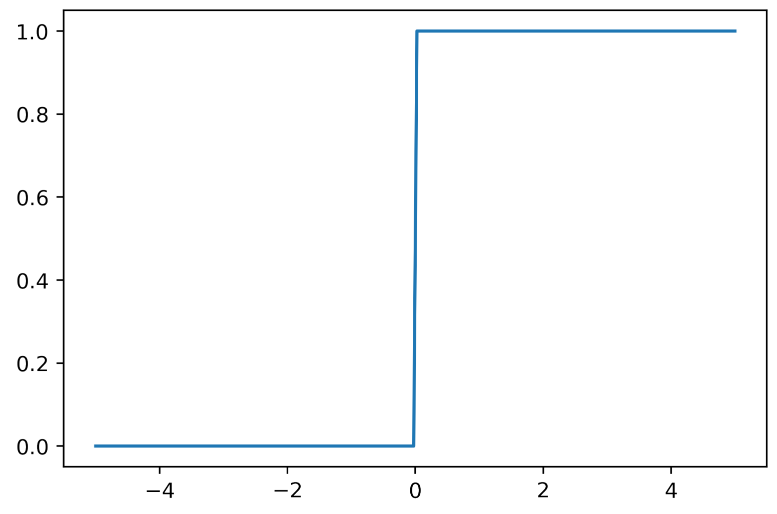
$$\begin{cases} 1, \ if \sum_i w_ix_i \geq 0 \\ 0, \ else \end{cases}$$

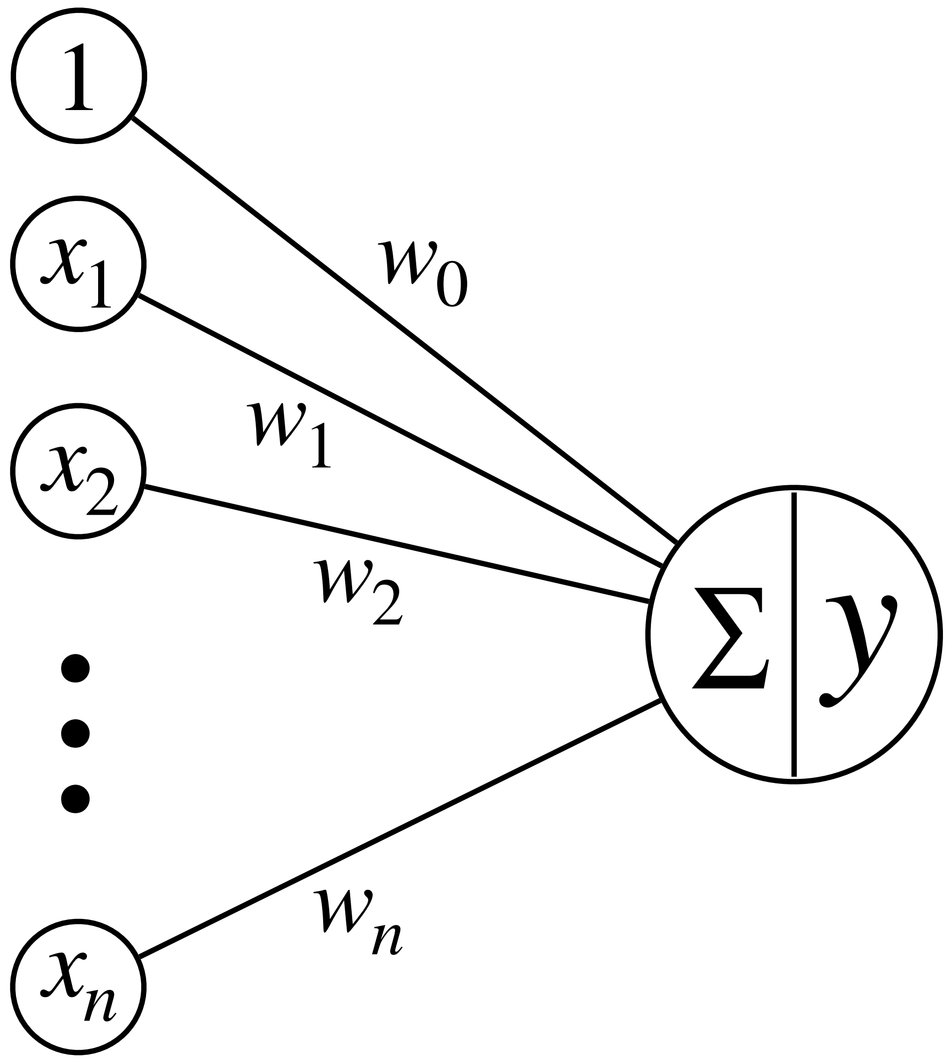
Allows us to make this even more beautiful simplification.

Perceptron

Also this allows us to introduce the notion of an <u>activation function.</u>

Heaviside step function





$$y = \sigma(\sum_i w_i x_i) = \begin{cases} 1, \; if \sum_i w_i x_i \geq 0 \\ 0, \; else \end{cases}$$

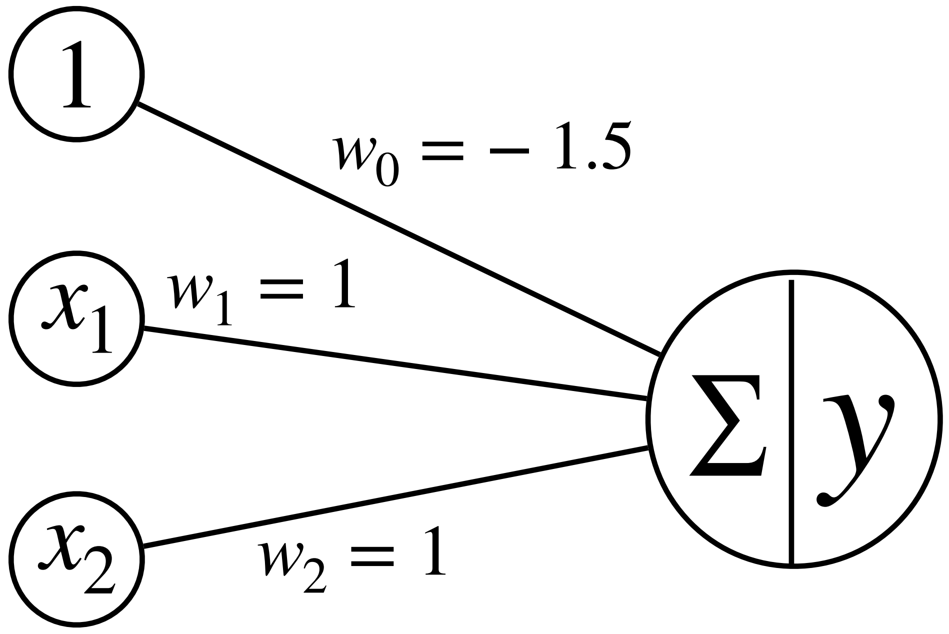
Example: Logical Gates

Perceptrons can equally be used to implement logical gates!

Logical AND

x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1

E.g.



$$1 * 0 + 1 * 0 - 1.5 < 0 \rightarrow y = 0$$
$$1 * 1 + 1 * 0 - 1.5 < 0 \rightarrow y = 0$$
$$1 * 0 + 1 * 1 - 1.5 < 0 \rightarrow y = 0$$
$$1 * 1 + 1 * 1 - 1.5 \geq 0 \rightarrow y = 1$$

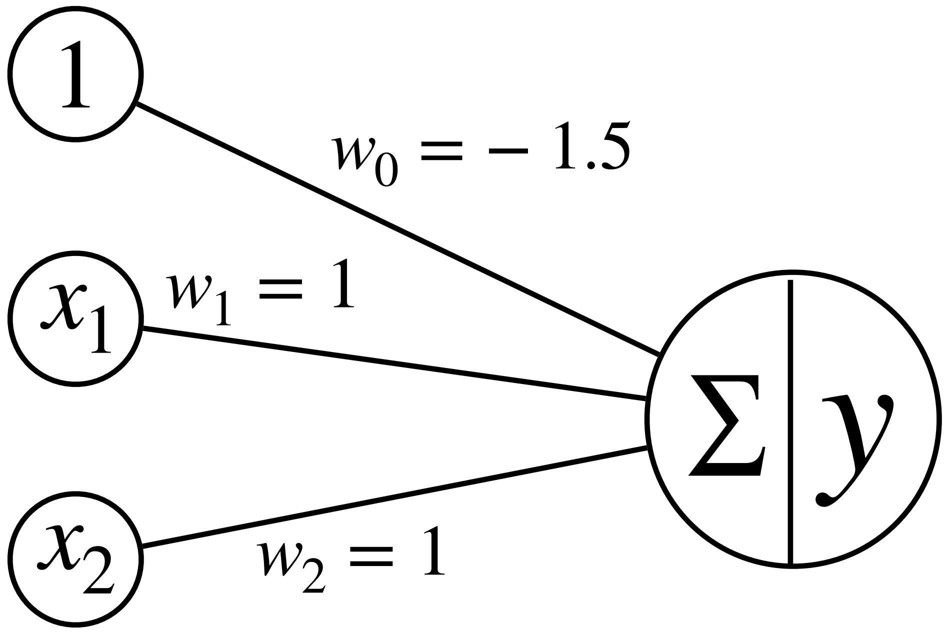
Example: Logical Gates

Perceptrons can equally be used to implement logical gates!

E.g.

Logical AND

x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1



$$1 * 0 + 1 * 0 - 1.5 < 0 \rightarrow y = \boxed{0}$$
$$1 * 1 + 1 * 0 - 1.5 < 0 \rightarrow y = \boxed{0}$$
$$1 * 0 + 1 * 1 - 1.5 < 0 \rightarrow y = \boxed{0}$$
$$1 * 1 + 1 * 1 - 1.5 \geq 0 \rightarrow y = \boxed{1}$$

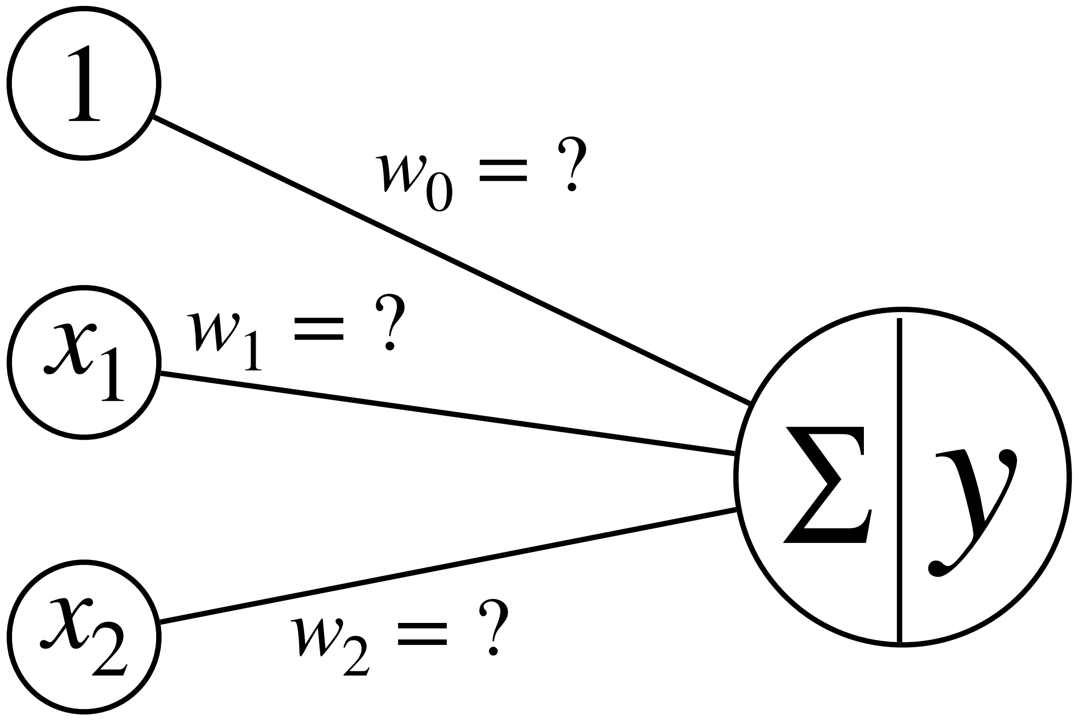
✓

Exercise: XOR

Find out the weights to implement an XOR gate.
If you know the answer please let your fellow students work it out themselves!

Logical XOR

x_1	x_2	$x_1 \oplus x_2$
0	0	0
1	0	1
0	1	1
1	1	0



$$y = \begin{cases} 1, & if \sum_i w_i x_i \geq 0 \\ 0, & else \end{cases}$$

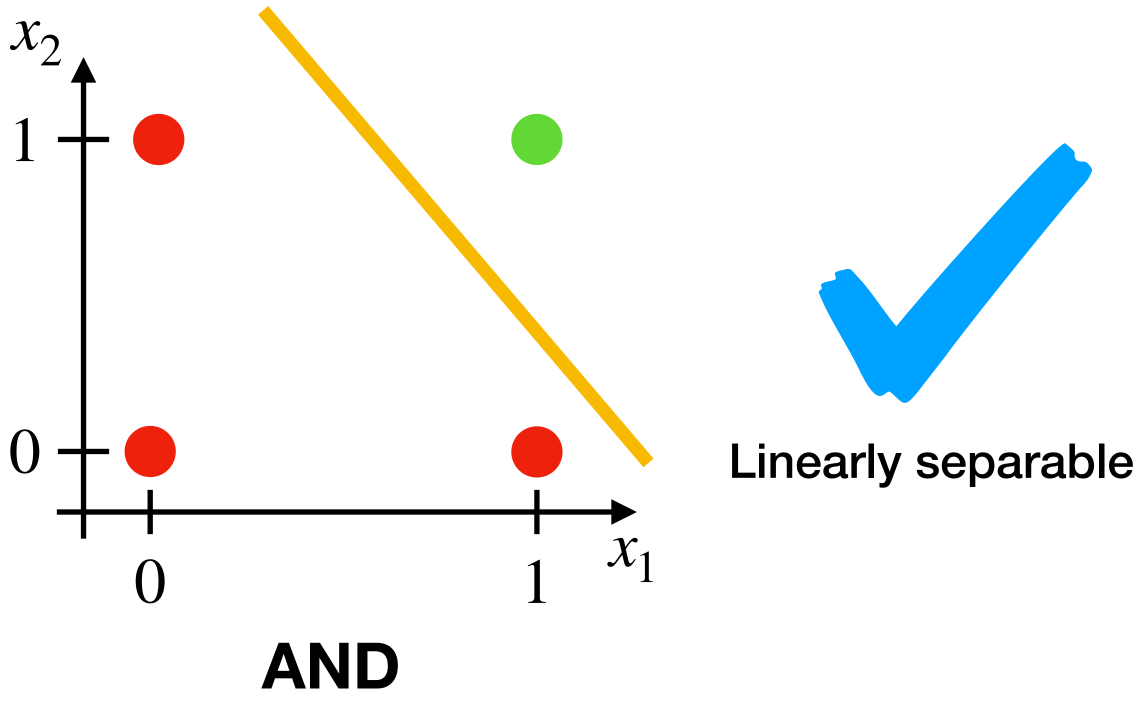
5 minutes! Do not skip to the next slide!

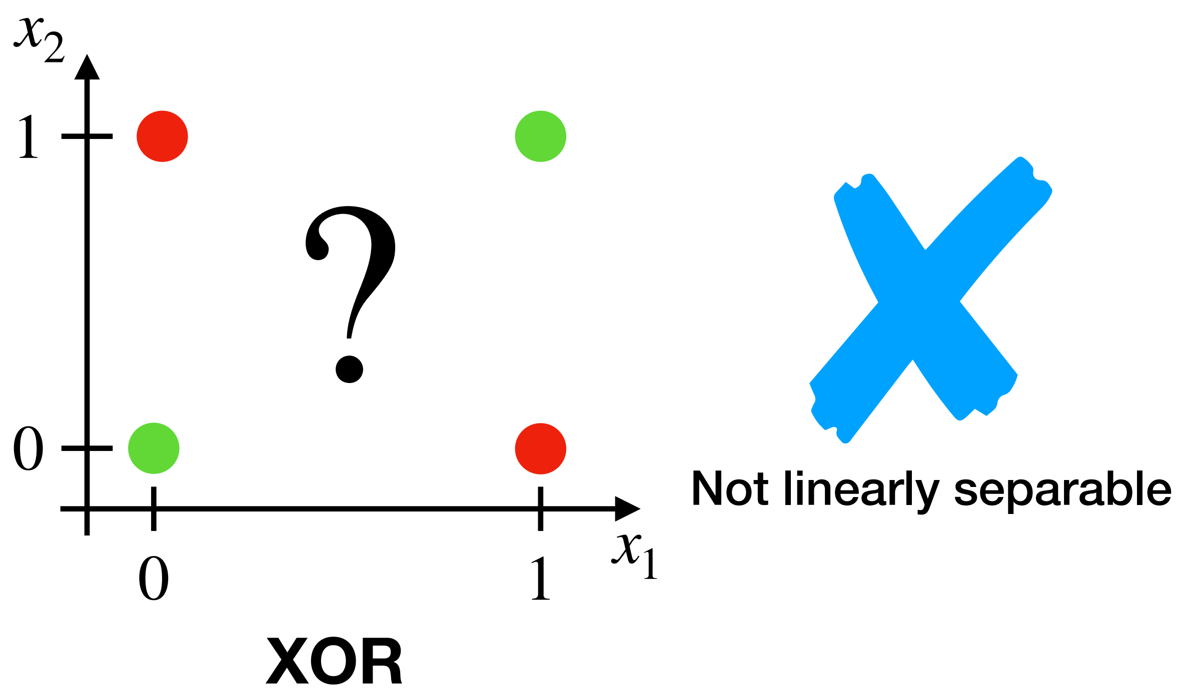
Exercise: XOR

- The answer is that a perceptron is not able to implement an XOR gate.
- Famously published in *Perceptrons: An introduction to computational geometry* (Minsky & Papert) [mp]
- Why? Simply put, because a perceptron can only solve linear problems.

Marvin Minsky & Seymour Papert

[w2, w3]





Perceptron

- Why can the perceptron only solve linear problems?
- A perceptron implements the following equation:

$$w_0 + w_1 x_1 + \ldots + w_n x_n = 0$$

(asking whether it's larger or smaller than 0)

- Let's simplify for three inputs!

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 = 0$$

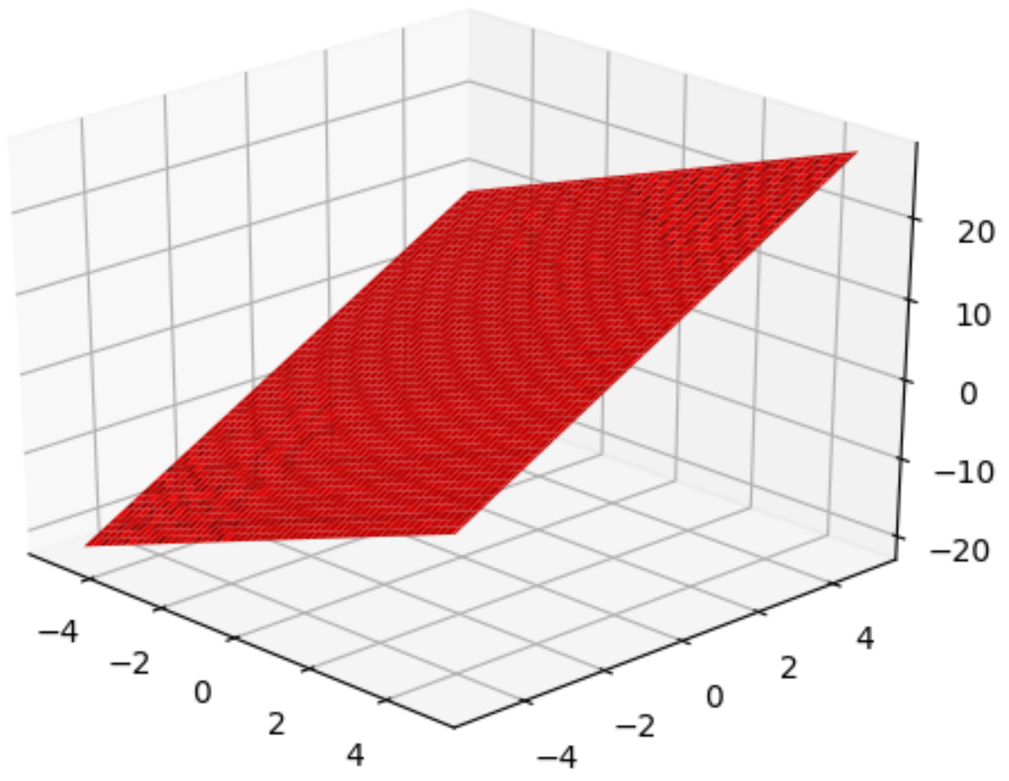
$$\boxed{w_1 x_1 + w_2 x_2 + w_3 x_3 = -w_0}$$

Does this remember you of anything? Think high-school math!

Hyperplanes

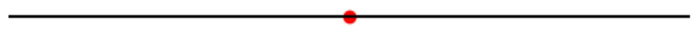
Maybe in this form? $ax + by + cz = d$

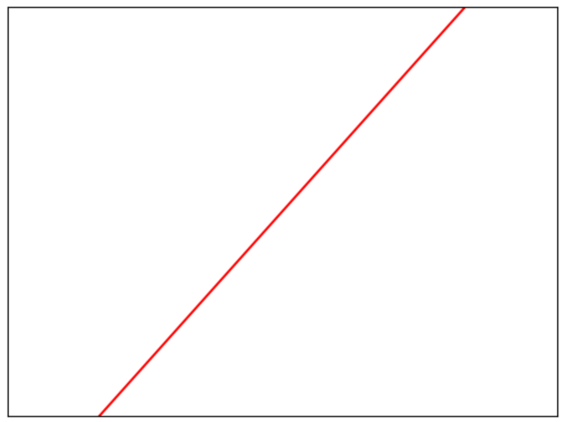
This equation is used to define a 2D plane in a 3D space!

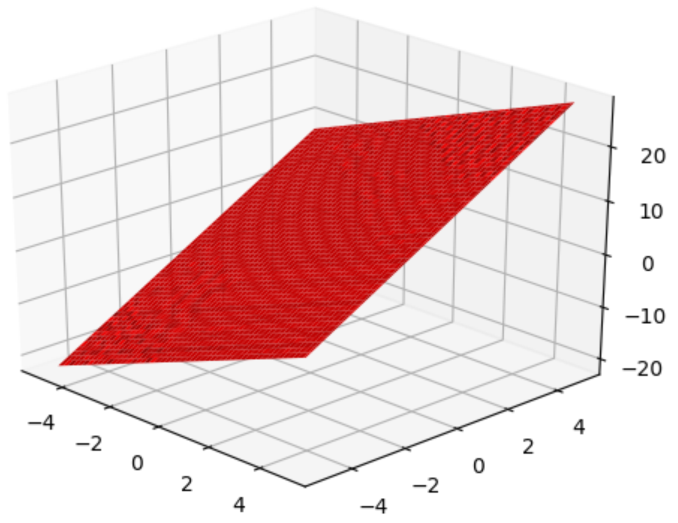


Hyperplanes

In general an equation of the form $w_0 + w_1 x_1 + \ldots + w_n x_n = 0$ **always implements a <u>hyperplane</u> in $\mathbb{R}^n$. A hyperplane is a plane, which has one dimension less than the space it is embedded in and therefore linearly separates the space into two parts.**





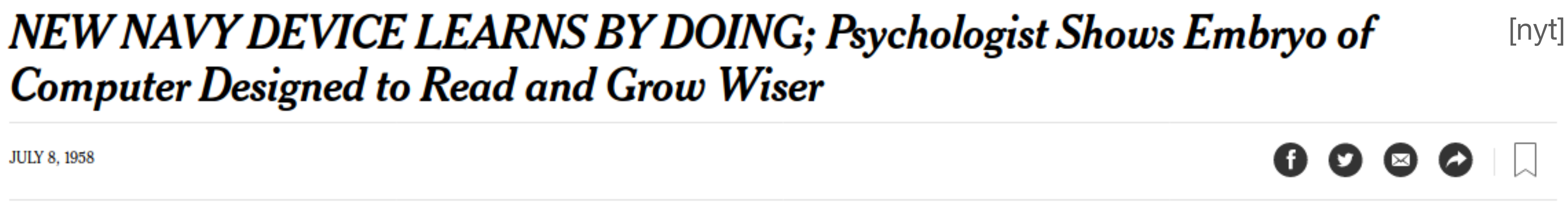


● ● ●

Perceptron Learning

Perceptron Learning

- The great deal about the perceptron was, that it was able to learn completely autonomous from given data.
- Many thought that it was the big breakthrough for AI!



NEW NAVY DEVICE LEARNS BY DOING; Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

JULY 8, 1958

[nyt]

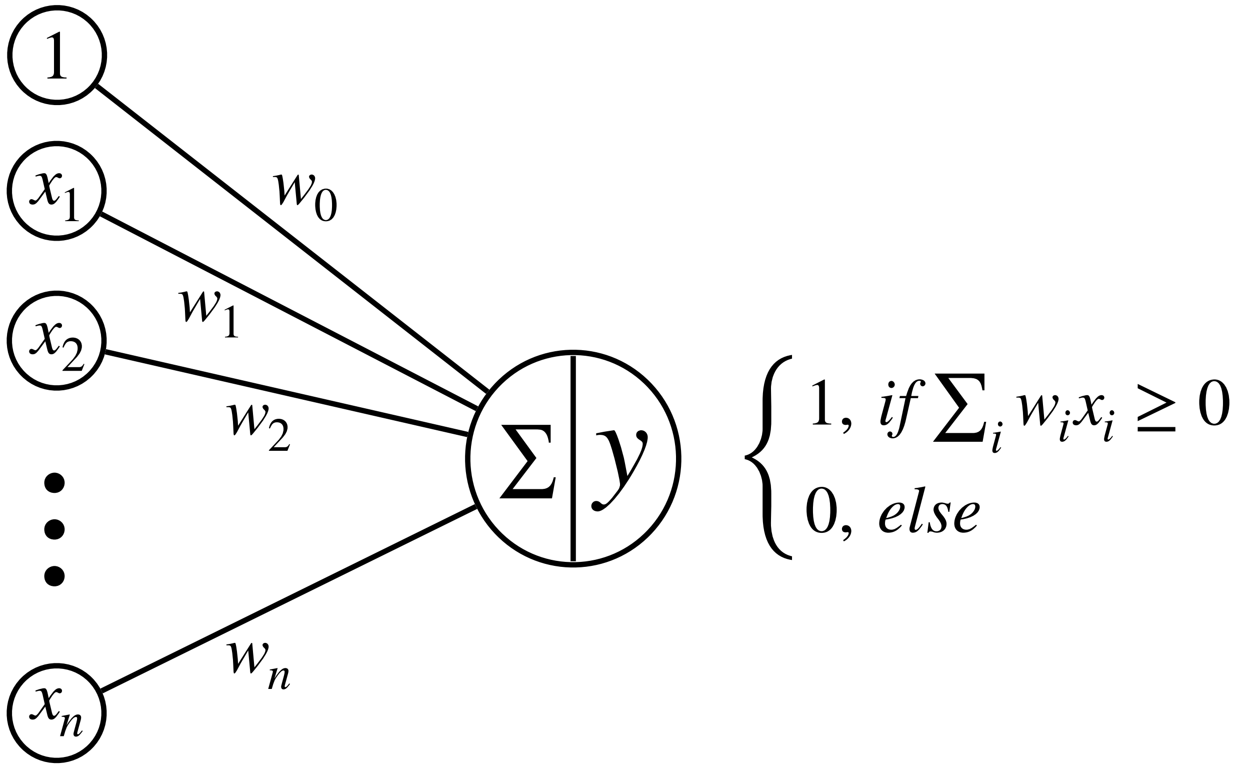
“The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence … Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers”

Perceptron Learning Rule

- It is based on one simple learning rule.
- Given a data sample $(\overrightarrow{x}, t)$ consisting of input $\overrightarrow{x}$ and label t, the update is defined by

$$w_i^{new} = w_i^{old} + \Delta w_i$$

$$\Delta w_i = \underset{\textbf{Learning rate}}{\alpha} * \underset{\textbf{Error}}{(t - y)} * x_i$$



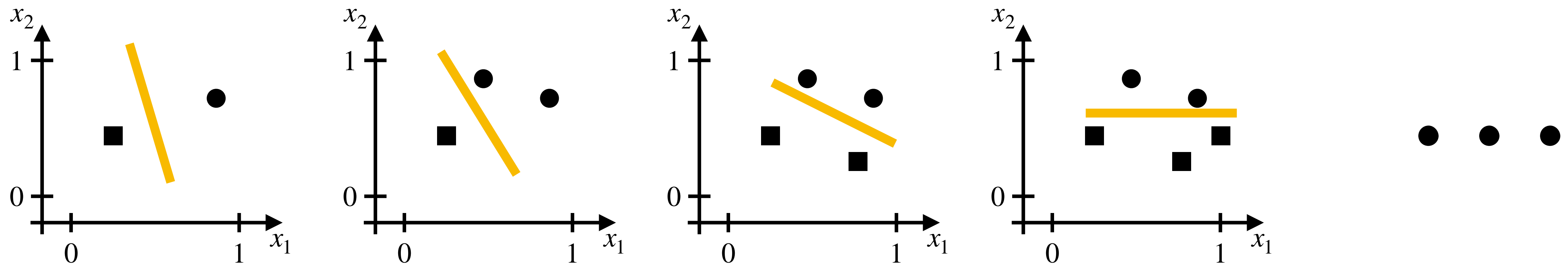
1
x_1
x_2
x_n
w_0
w_1
w_2
w_n
Σ
y
$\begin{cases} 1, \; if \sum_i w_i x_i \geq 0 \\ 0, \; else \end{cases}$

Perceptron Learning Rule

- This simple learning rule allows the perceptron to slowly adapt its hyperplane to separate the one class from the other.

$$w_i^{new} = w_i^{old} + \Delta w_i$$

$$\Delta w_i = \alpha * (t - y) * x_i$$



Multi-Layer Perceptron

Exercise: Solving XOR with Perceptrons

- Research on perceptron died because they were not able to solve non-linear problems as XOR.
- But maybe there is a way nevertheless?

Hint: $x_1 \oplus x_2 \Leftrightarrow (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

5 minutes! Do not skip to the next slide!

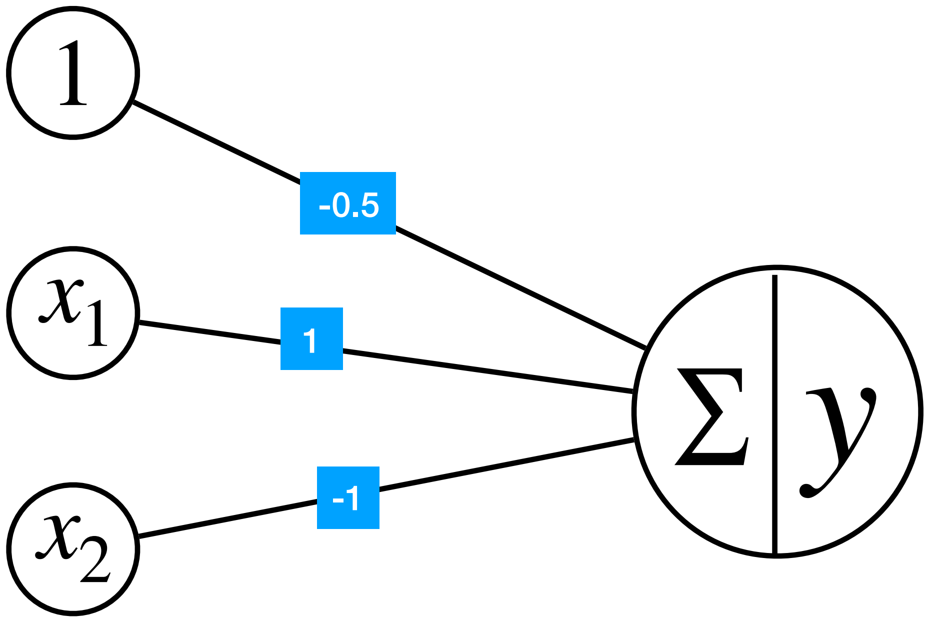
Exercise: Solving XOR with Perceptrons

- Indeed you can solve it by introducing multiple stacked perceptrons, a so called multi-layer perceptron.

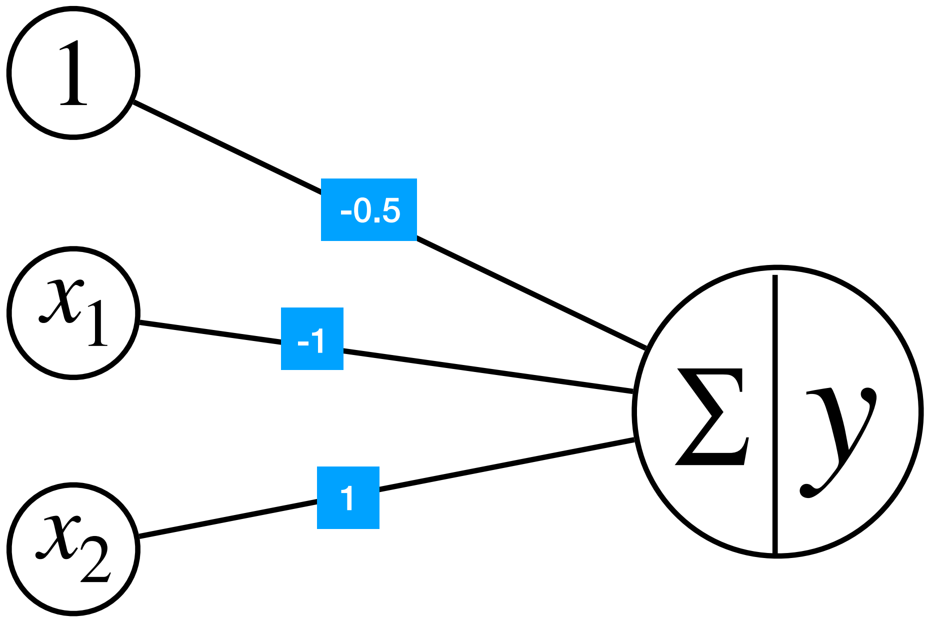
$$x_1 \oplus x_2 \Leftrightarrow (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

- First we implement the both AND gates.

$\boxed{x_1 \wedge \neg x_2}$



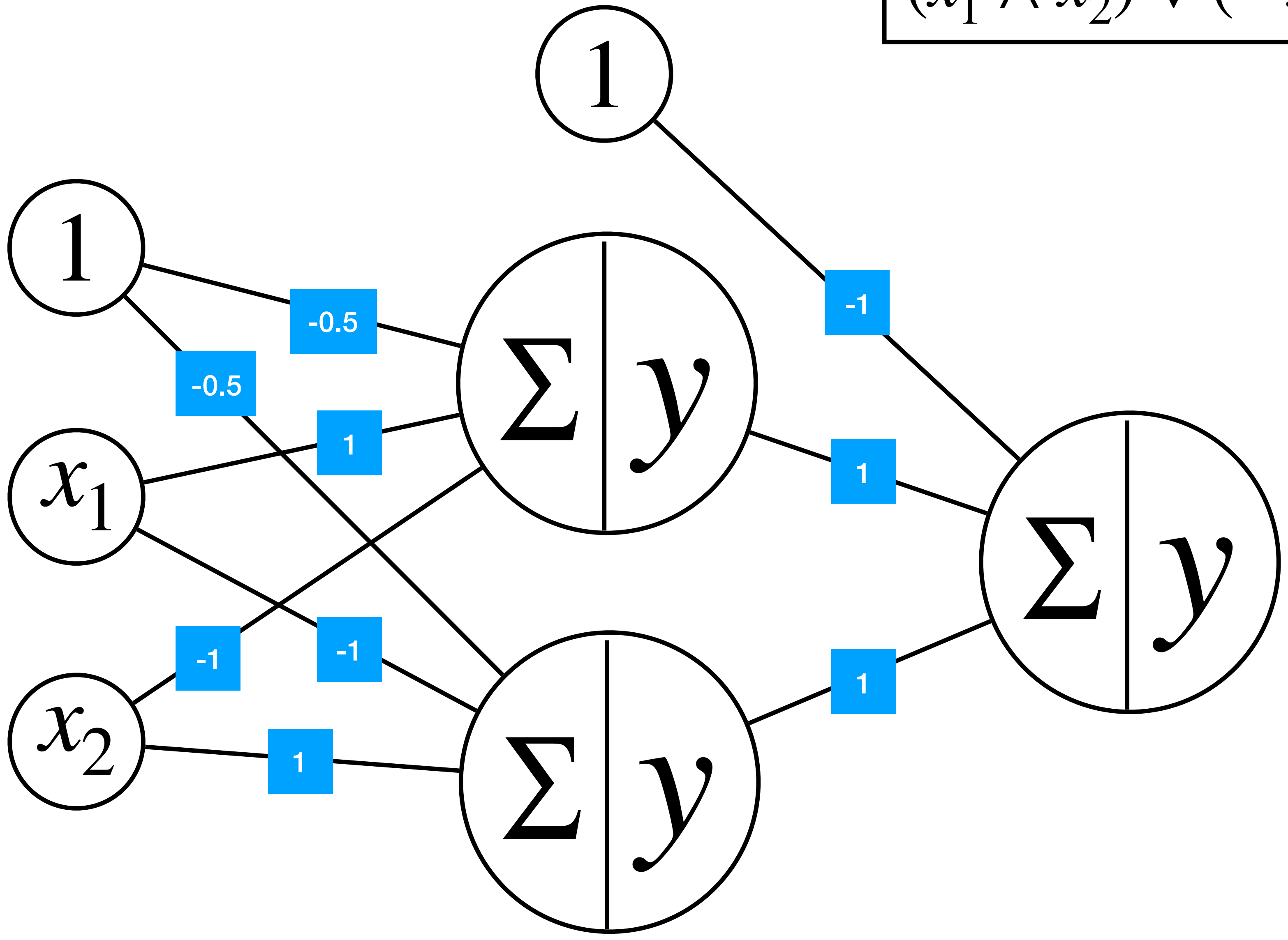
$\boxed{\neg x_1 \wedge x_2}$



Exercise: Solving XOR with Perceptrons

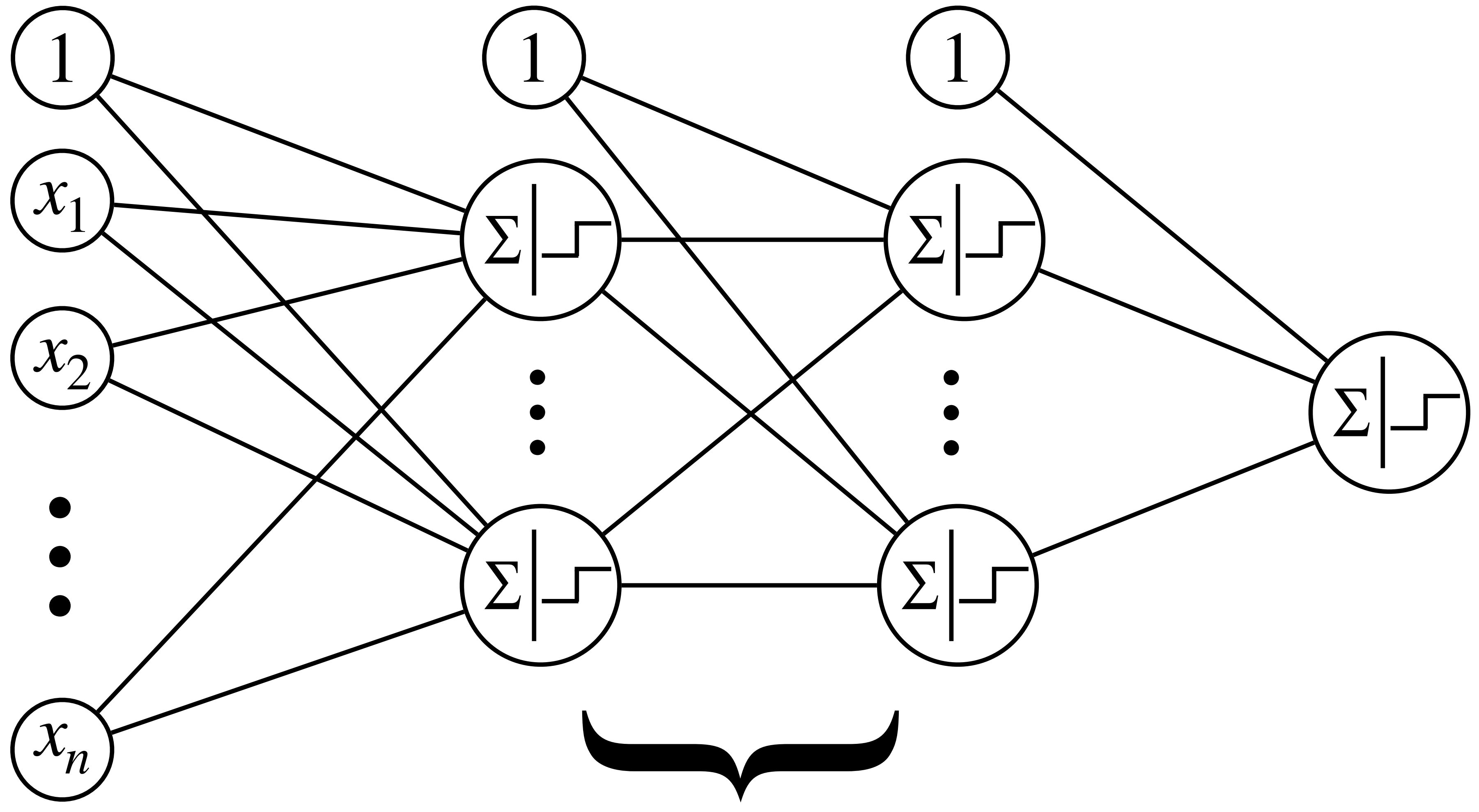
And now we connect the outputs with a new perceptron implementing an OR gate.

$$\boxed{(x_1 \land x_2) \lor (\neg x_1 \land \neg x_2)}$$



Multi-Layer Perceptron

Stacking multiple perceptrons onto one another gives us a so called multi-layer perceptron (MLP).

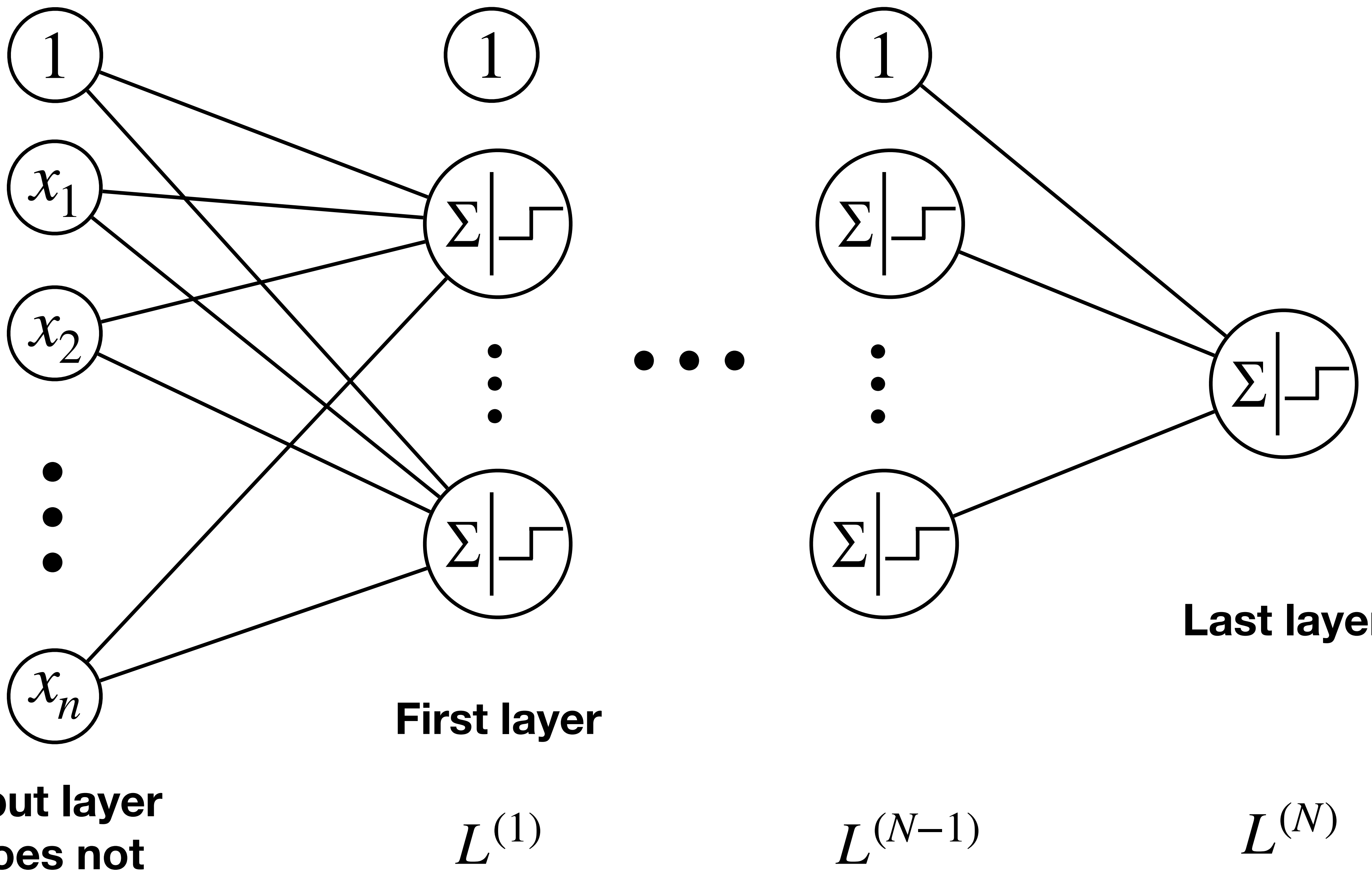


Input layer **Hidden layers** **Output layer**

MLP Notations

Layers

Notations in ANNs are not fixed and this is only one particular choice!



Input layer does not count!

First layer

$L^{(1)}$

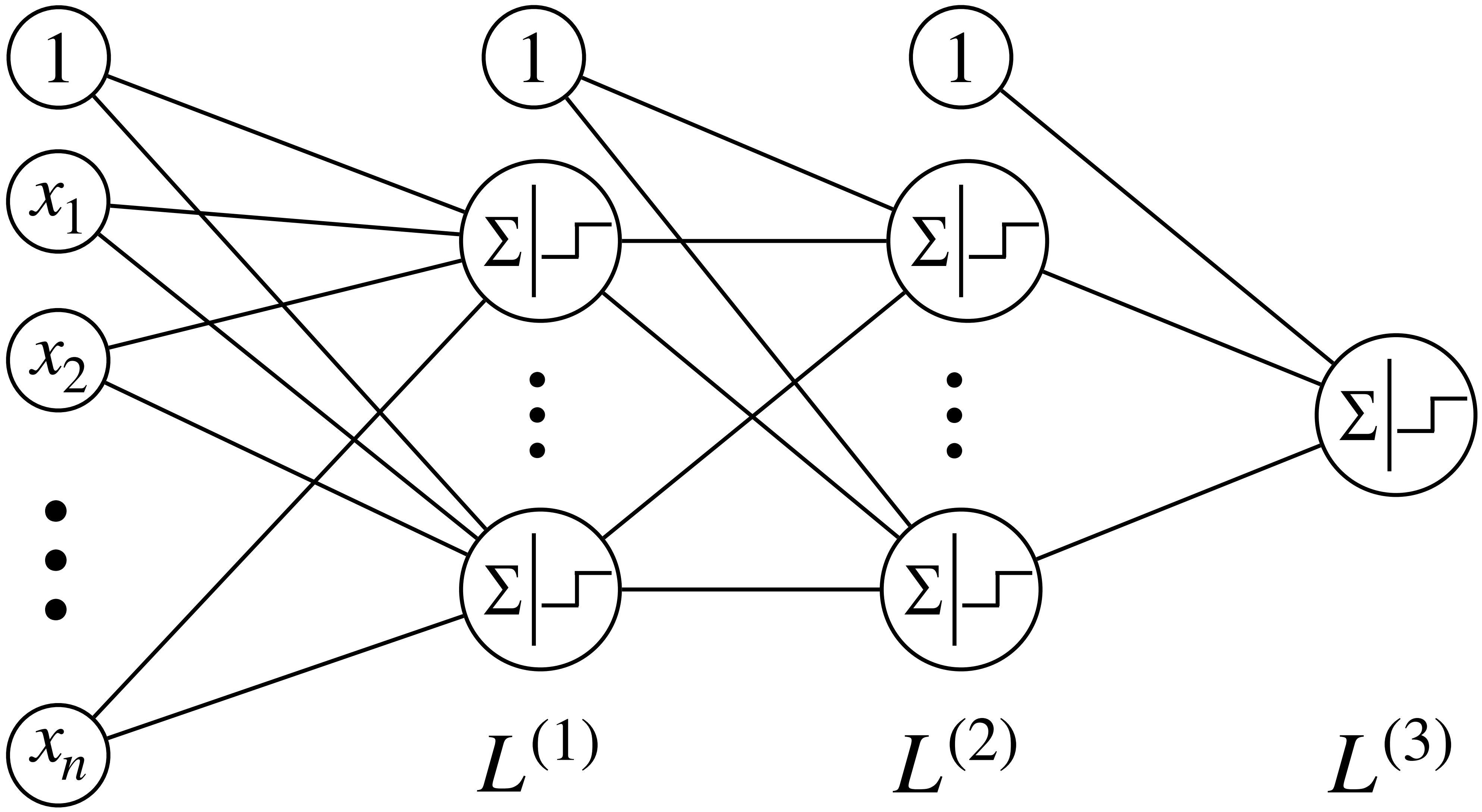
$L^{(N-1)}$

Last layer

$L^{(N)}$

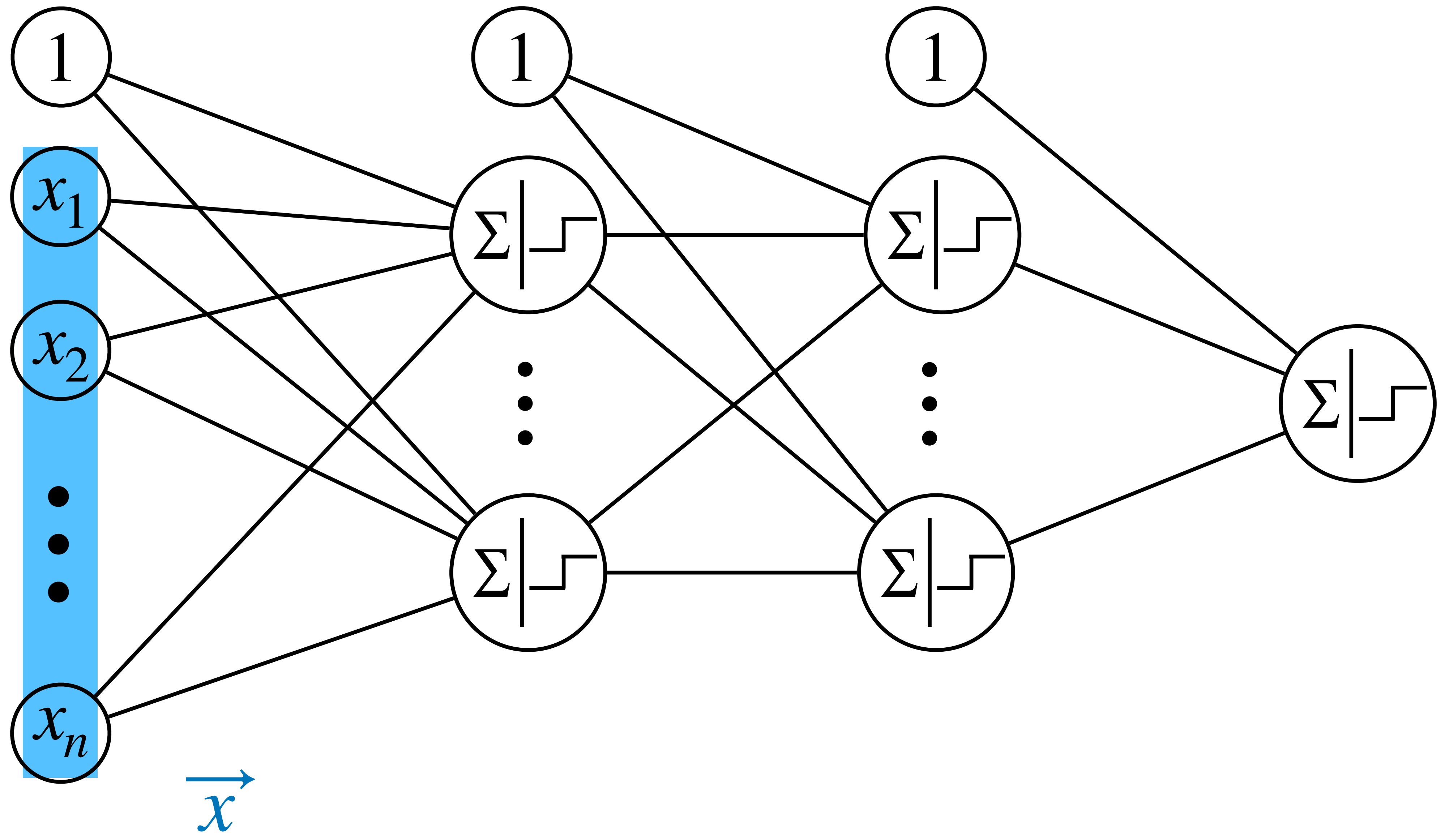
Layers

A network with three layers.



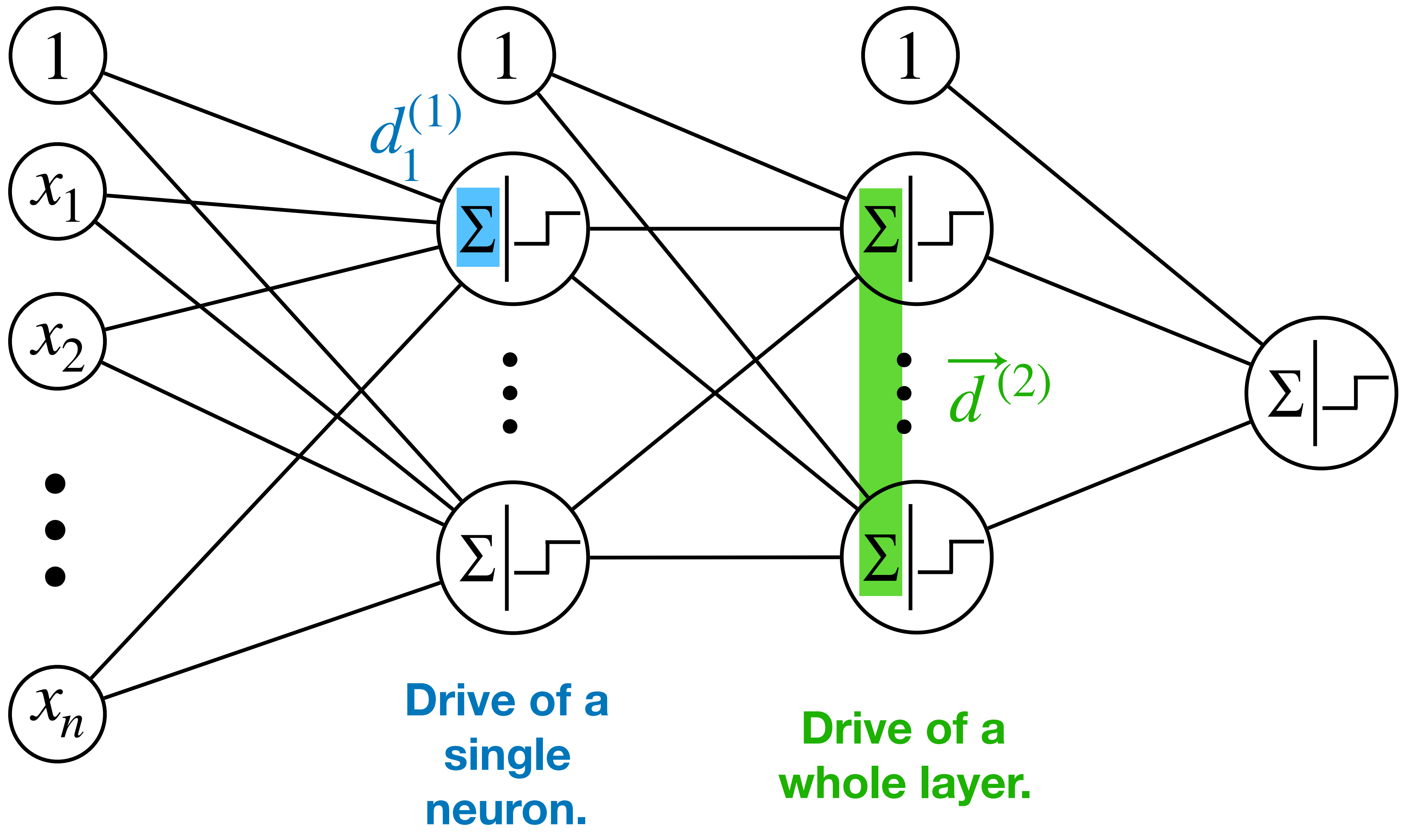
Input

The input to the network is called the <u>input</u>.



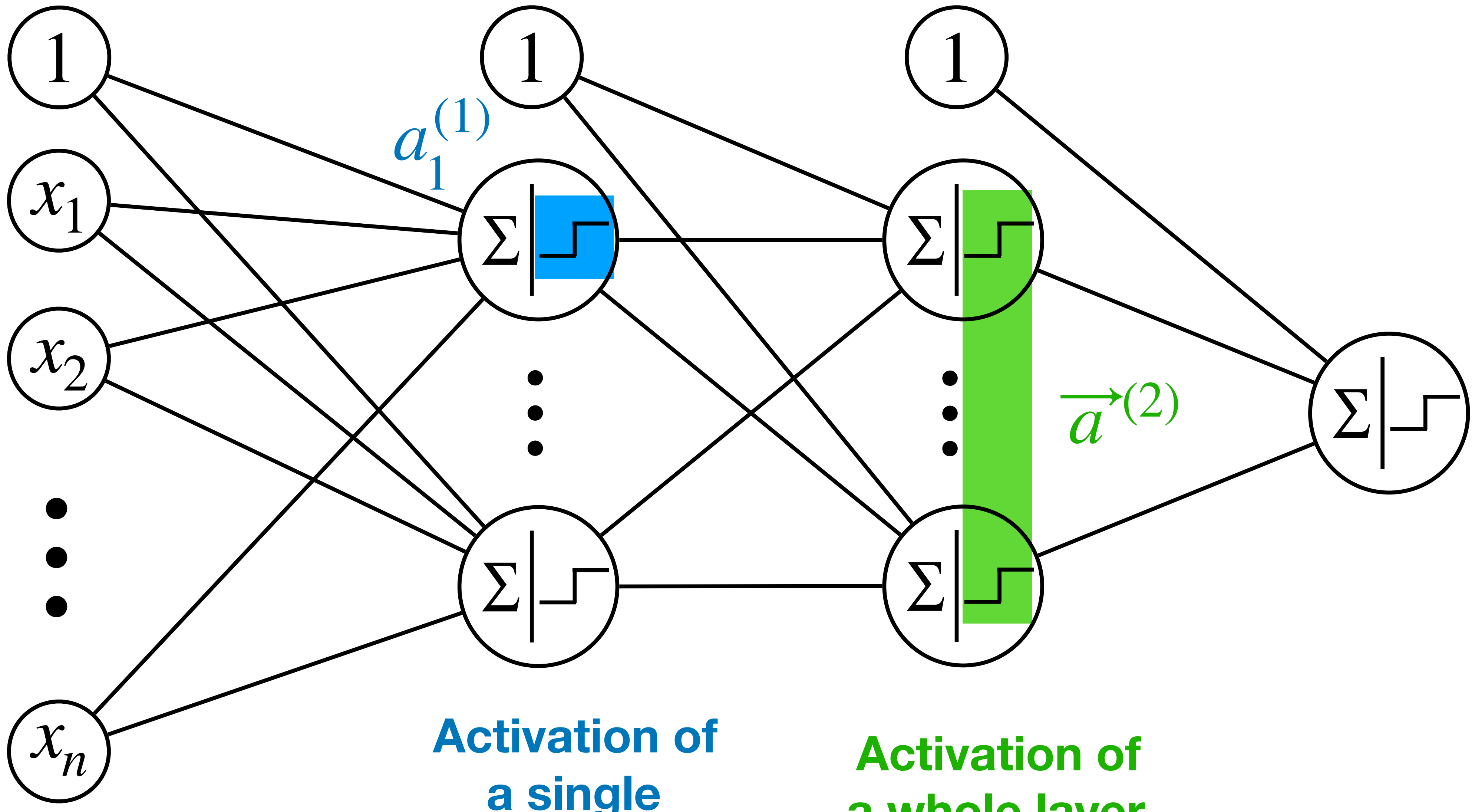
Drive

The input that a neuron receives is called the neuron's drive.



Activation

The output of a neuron in a hidden layer is called the neuron's <u>activation</u>.

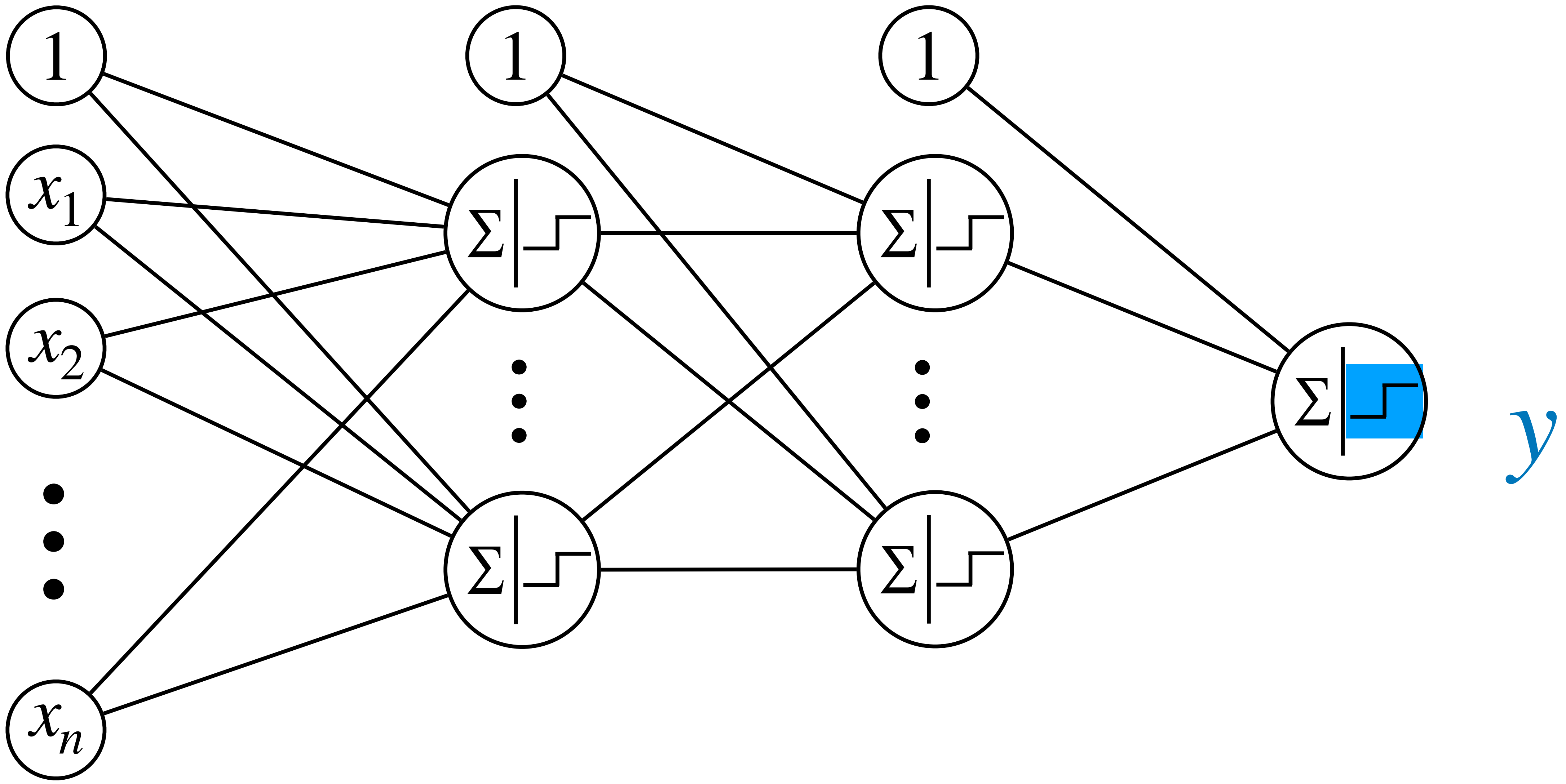


Activation of a single neuron.

Activation of a whole layer.

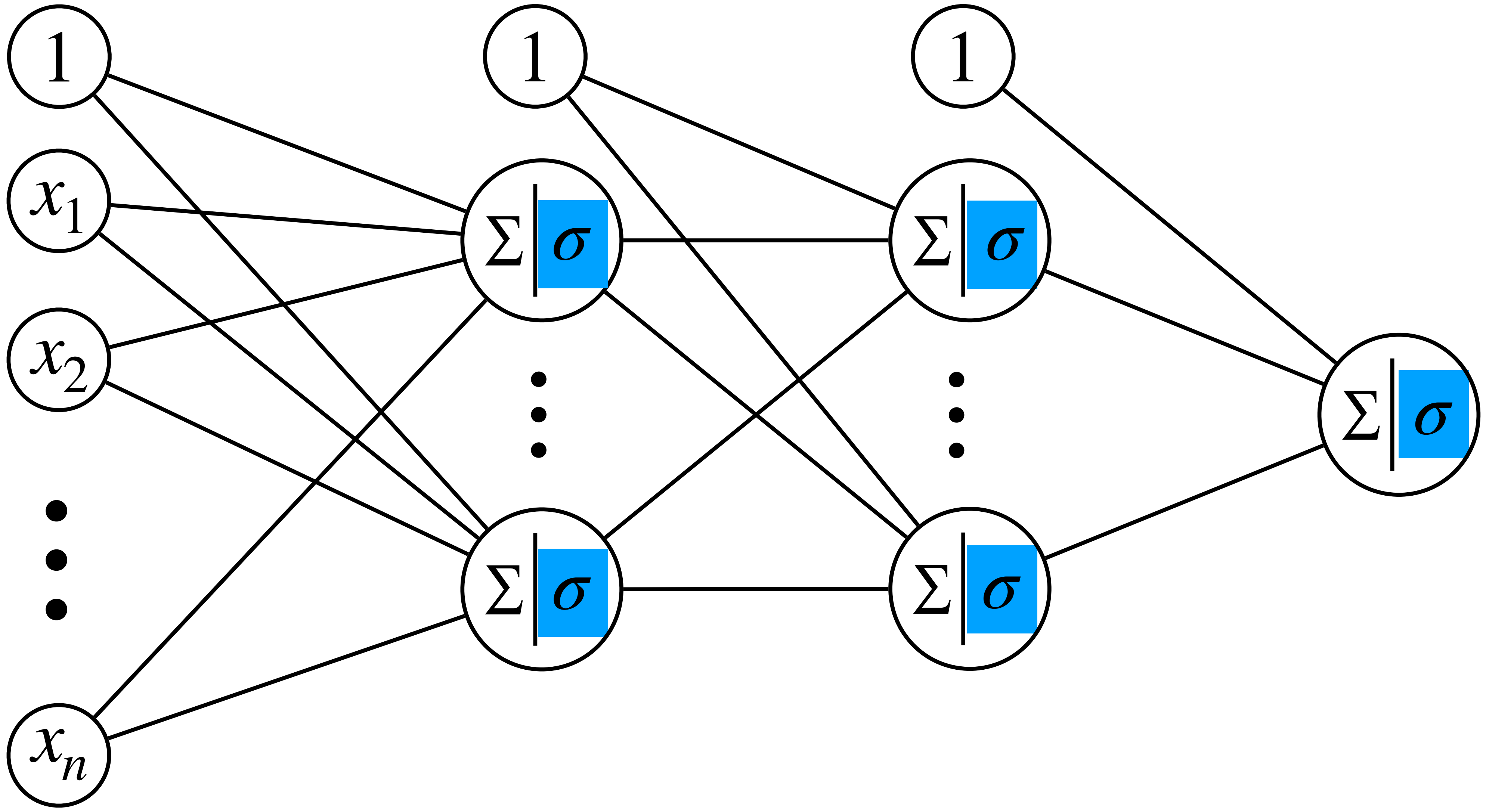
Output

The activation of the neurons in the last layer is called the network's output.



Activation Function

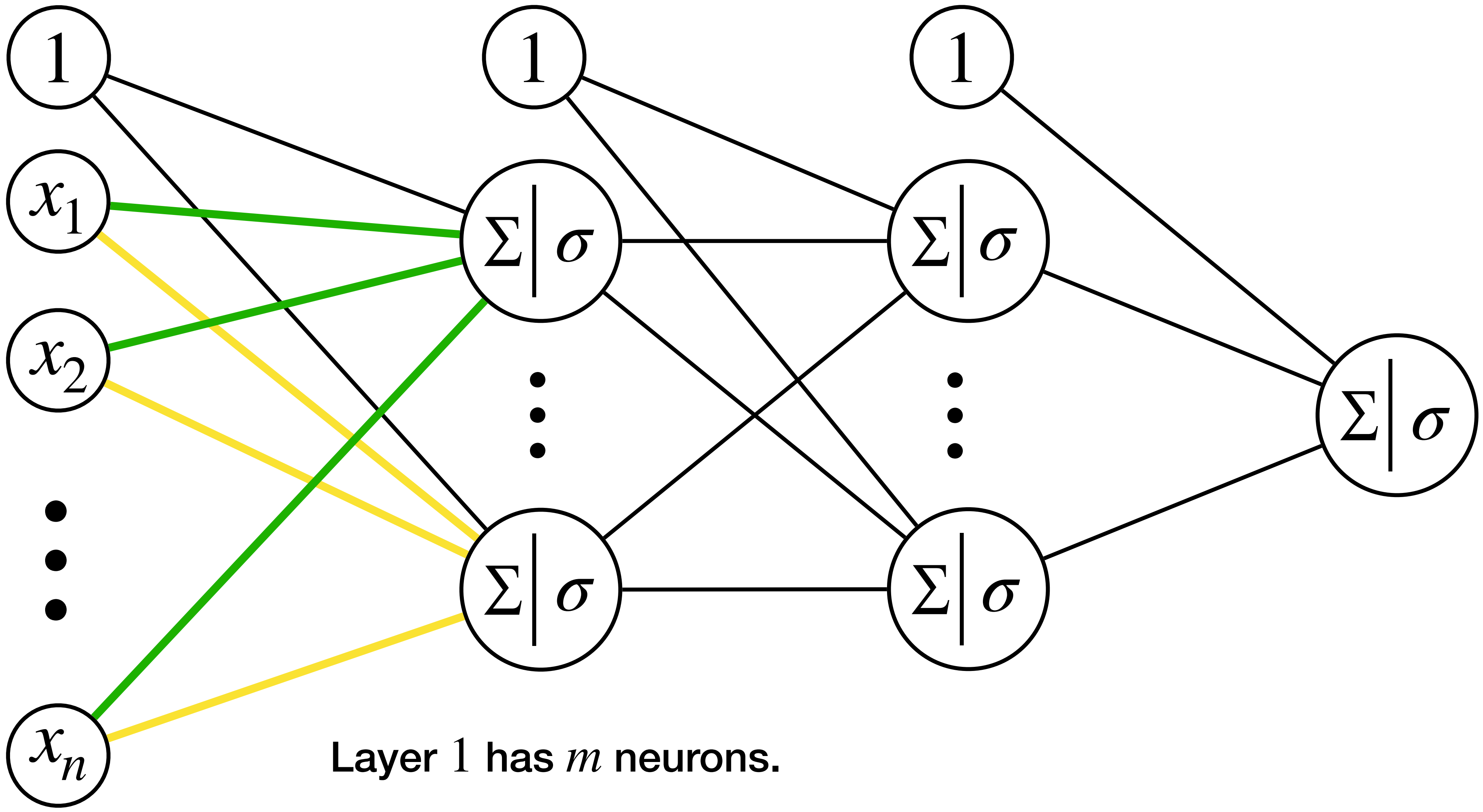
The activation function can be different from the step function, thus we generalize the notation to $\sigma(x)$.



Weights

The weights of a layer are formalized as a <u>weight matrix</u>.

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \cdots & w_{1n}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \cdots & w_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}^{(1)} & w_{m2}^{(1)} & \cdots & w_{mn}^{(1)} \end{pmatrix}$$



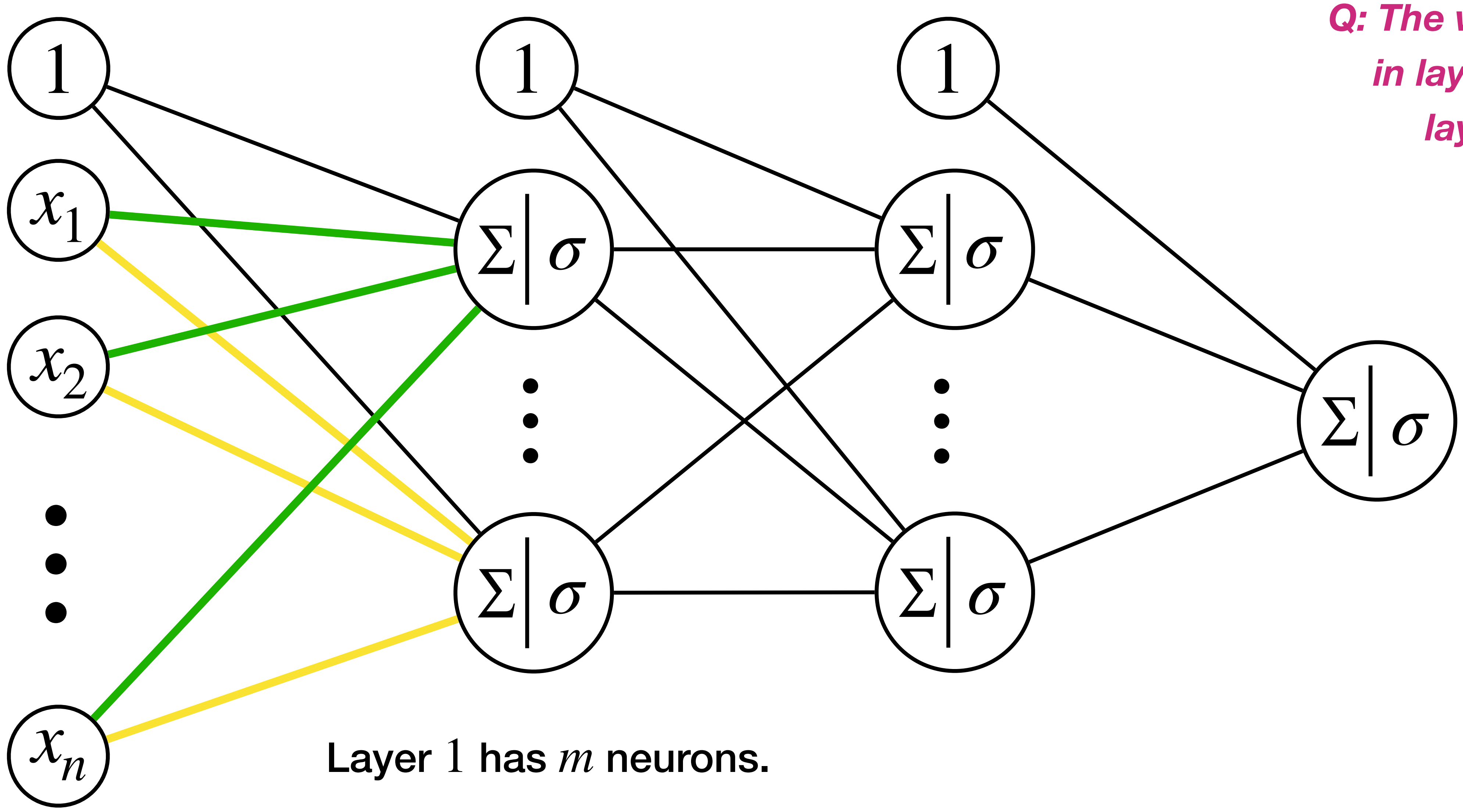
Layer 1 has m neurons.

Weights

The weights of a layer are formalized as a <u>weight matrix</u>.

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \cdots & w_{1n}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \cdots & w_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}^{(1)} & w_{m2}^{(1)} & \cdots & w_{mn}^{(1)} \end{pmatrix}$$

Q: The weight from neuron j in layer 1 to neuron i in layer 2 is called ?

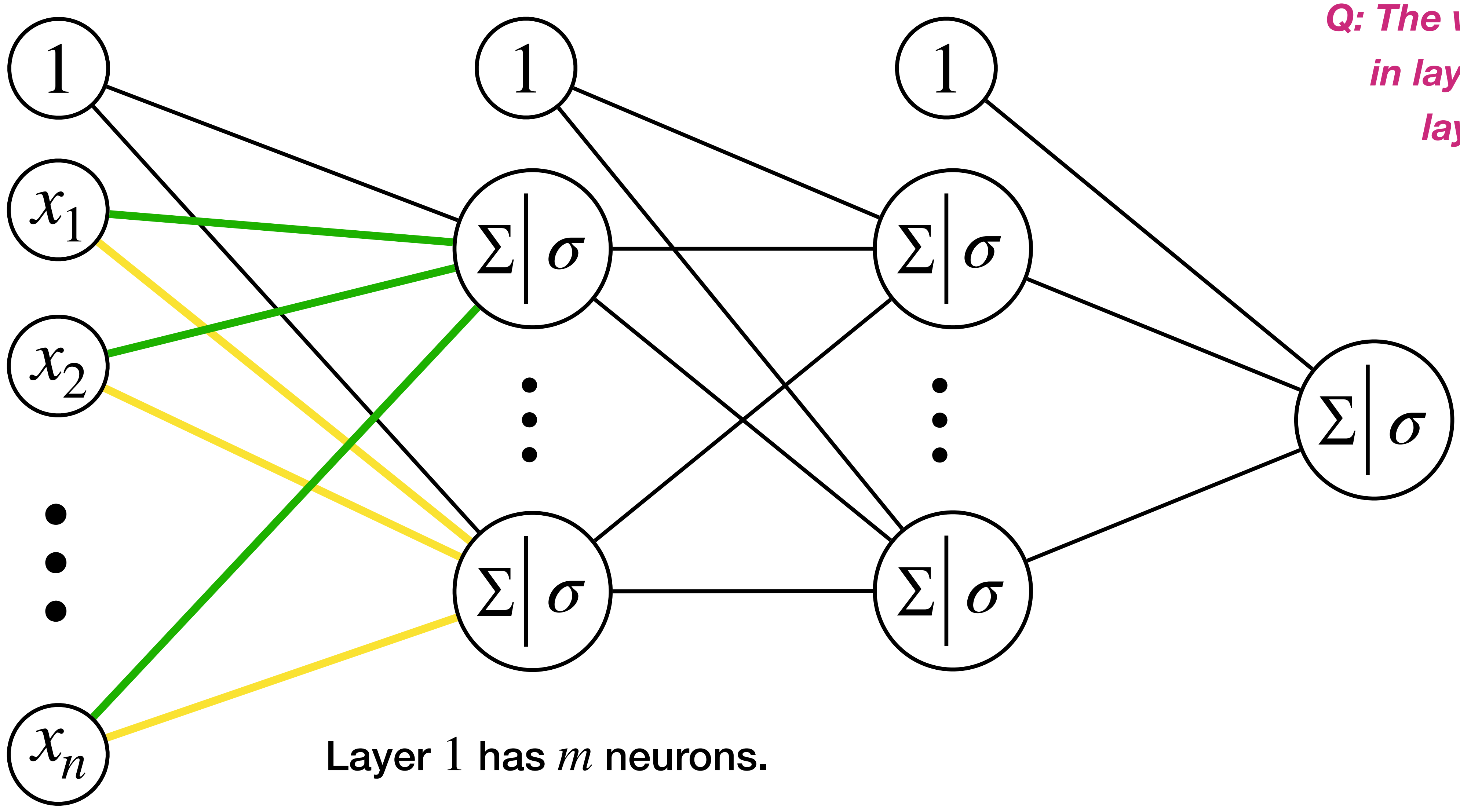


Layer 1 has m neurons.

Weights

The weights of a layer are formalized as a <u>weight matrix</u>.

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \cdots & w_{1n}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \cdots & w_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}^{(1)} & w_{m2}^{(1)} & \cdots & w_{mn}^{(1)} \end{pmatrix}$$



Layer 1 has m neurons.

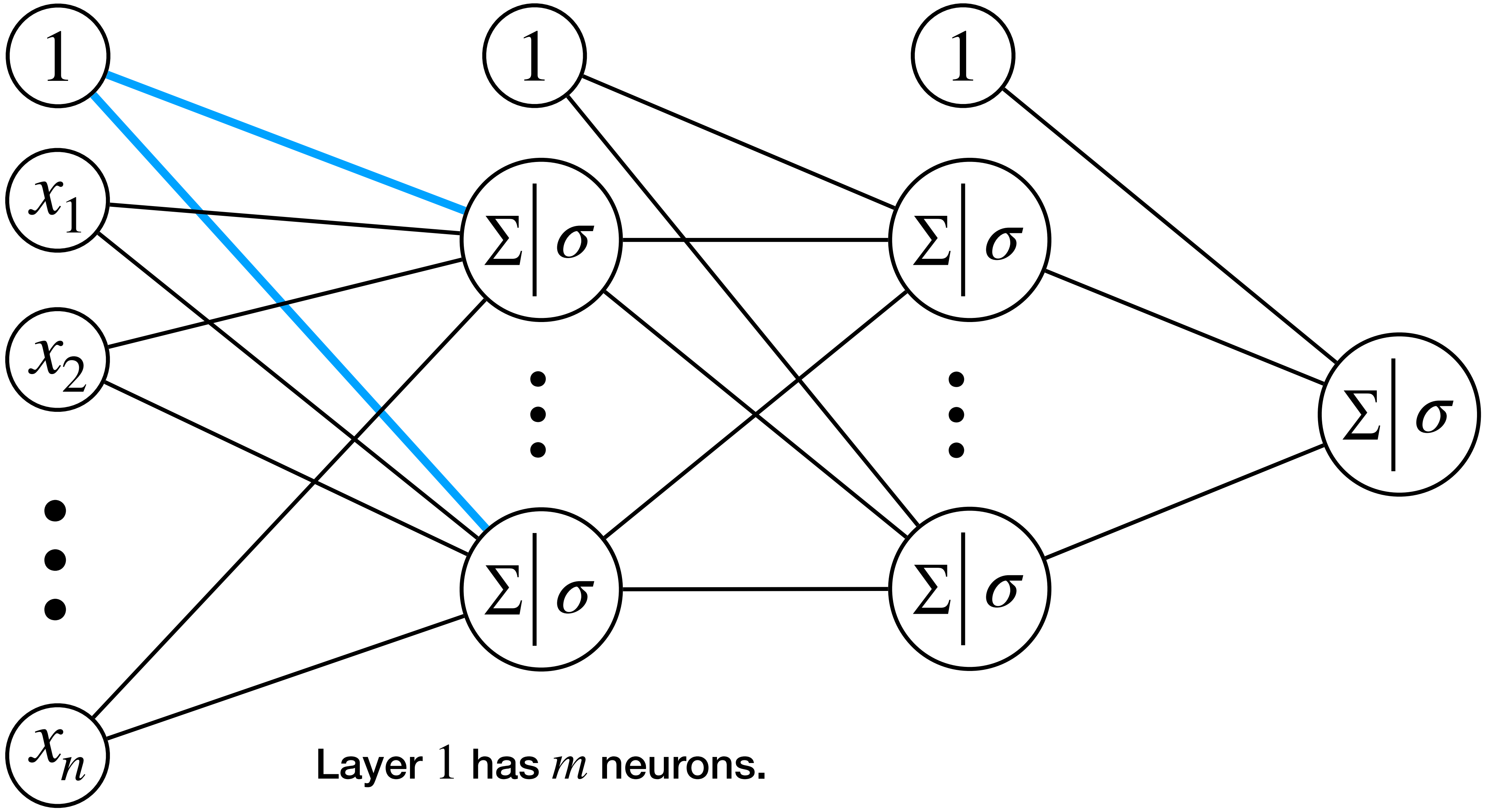
Q: The weight from neuron j in layer 1 to neuron i in layer 2 is called ?

$$w_{ij}^{(2)}$$

Bias Weights

The weights that model the threshold are called the biases or bias weights.

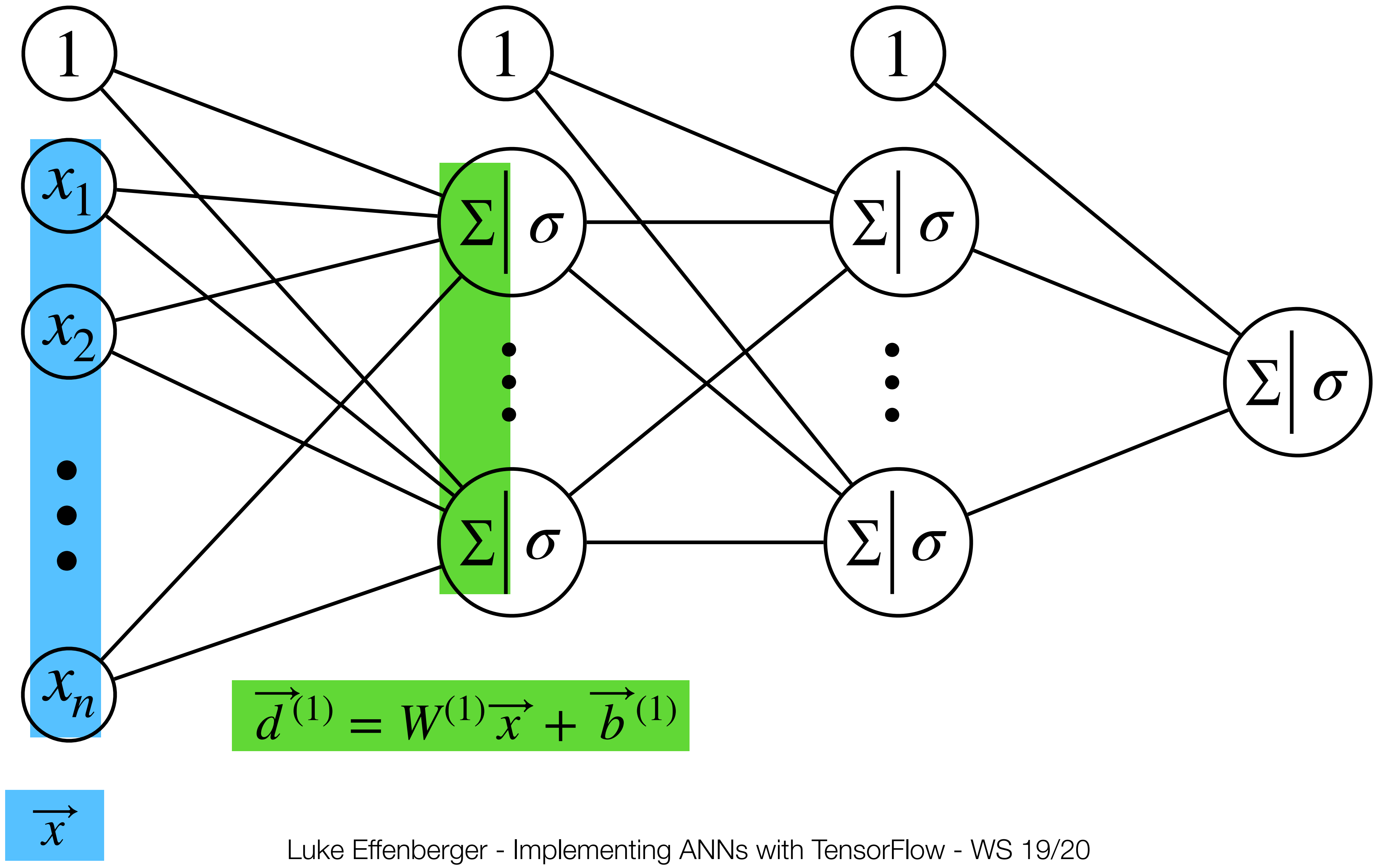
$$\vec{b}^{(1)} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ b_m^{(1)} \end{pmatrix}$$



Layer 1 has m neurons.

Forward Step

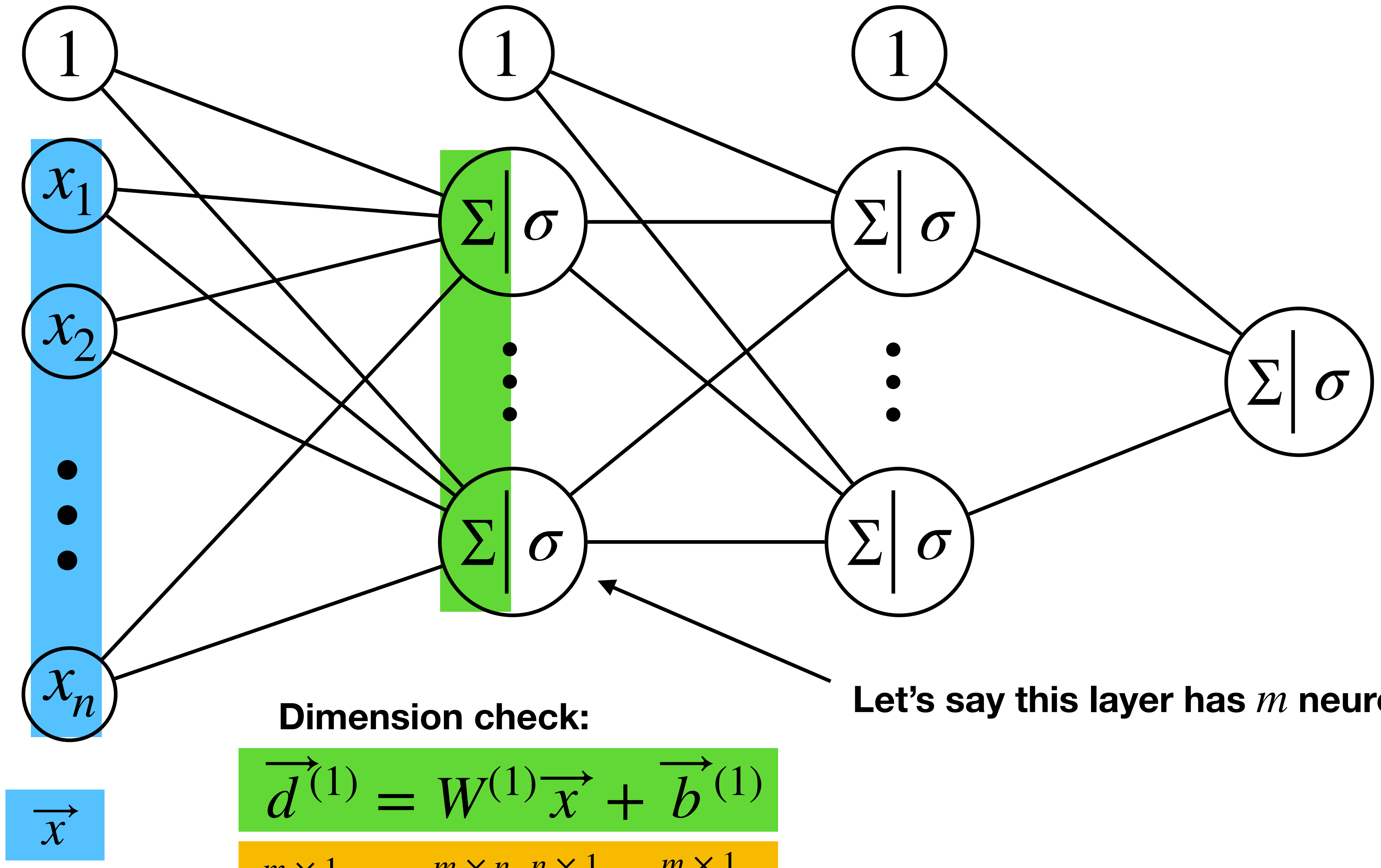
- The forward step describes how the input is processed throughout the whole network.
- We can calculate the activation of all perceptrons in a hidden layer by using matrix multiplication and vector addition.



$$\vec{a}^{(1)} = W^{(1)}\vec{x} + \vec{b}^{(1)}$$

Dimension Check

For understanding how the information is processed it is helpful to have a look of the dimensions of representations and weights.

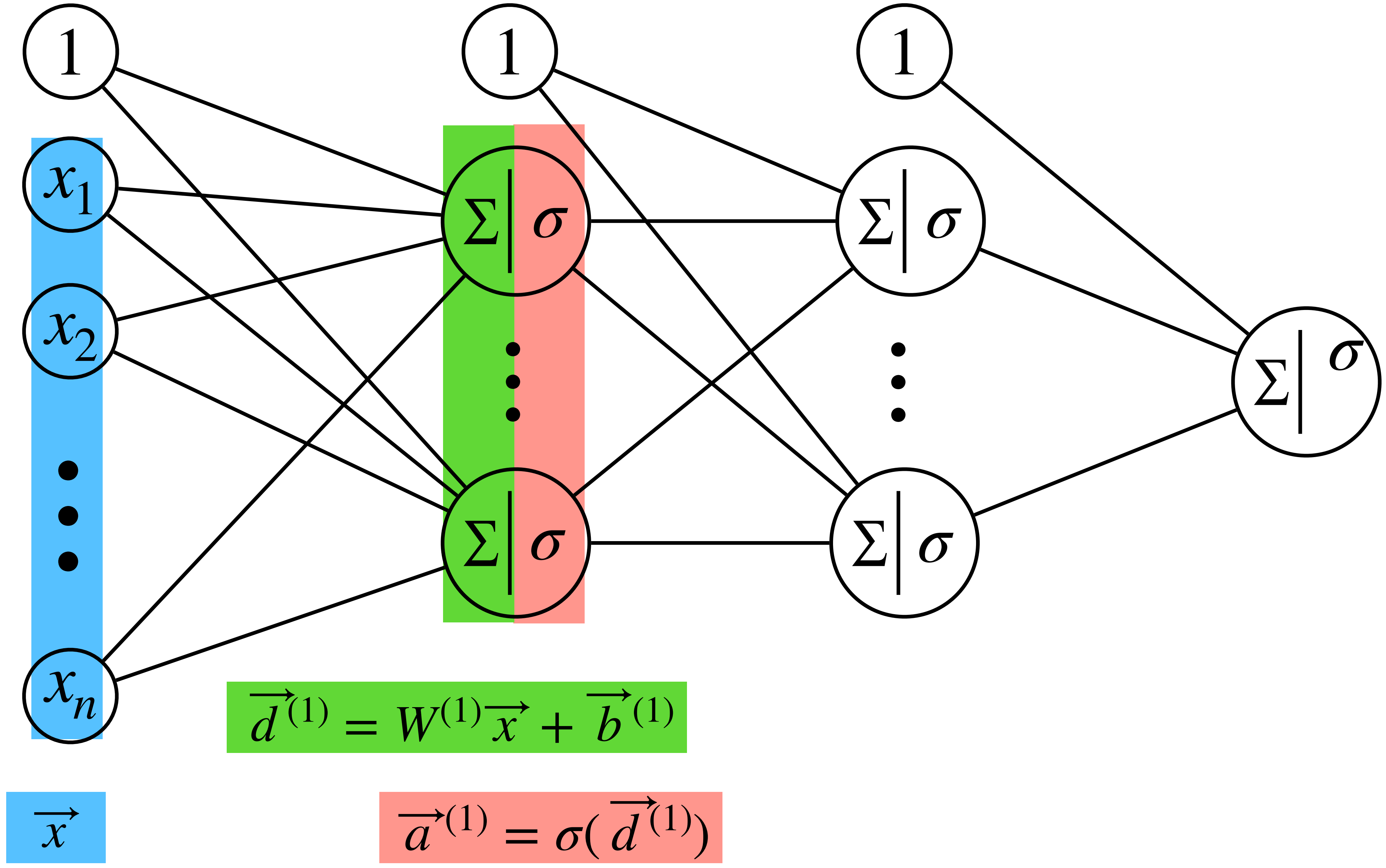


Dimension check:

$$\underbrace{\vec{d}^{(1)}}_{m \times 1} = \underbrace{W^{(1)}}_{m \times n}\underbrace{\vec{x}}_{n \times 1} + \underbrace{\vec{b}^{(1)}}_{m \times 1}$$

Forward Step

One forward step describes how the input is processed throughout the whole network.

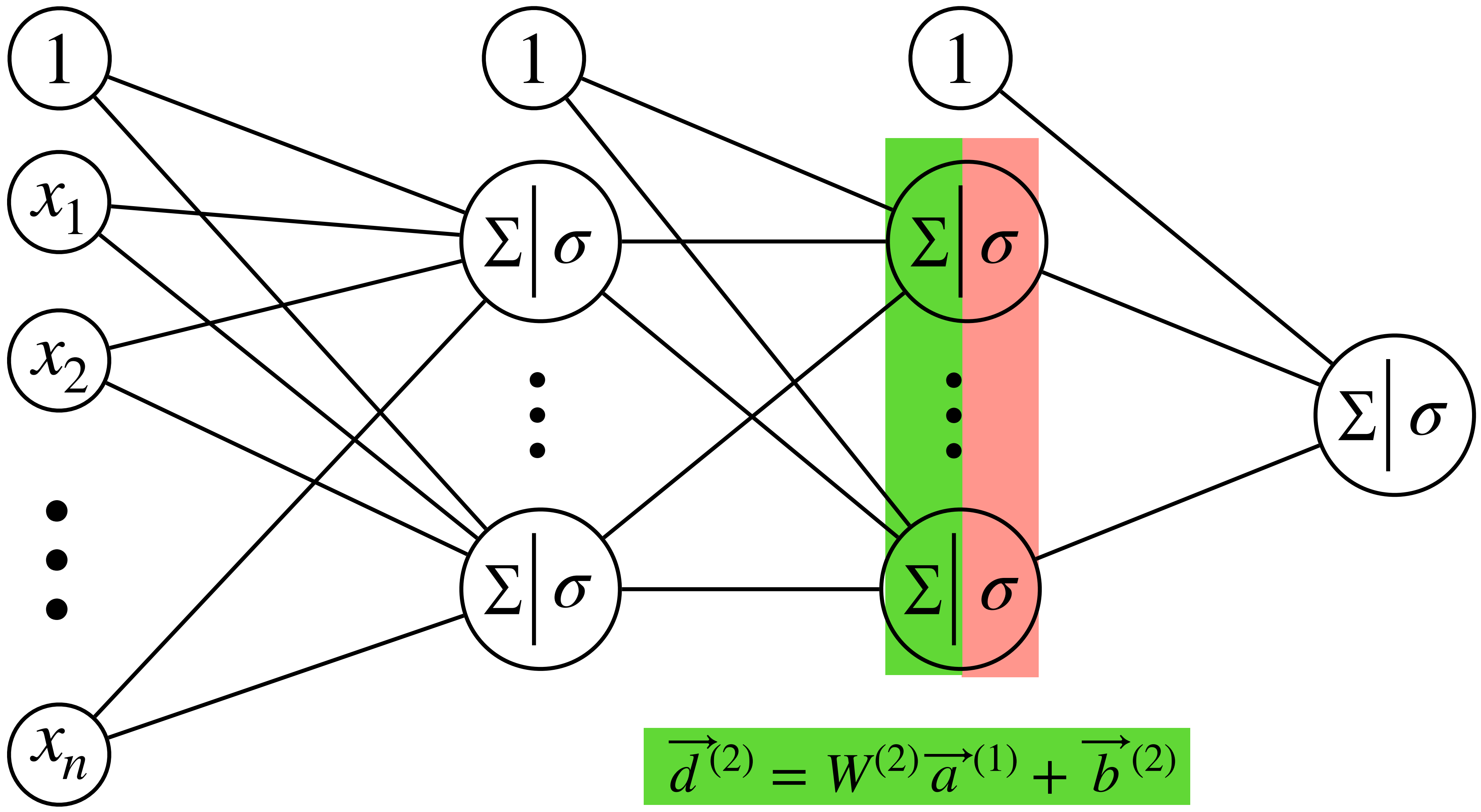


$$\vec{d}^{(1)} = W^{(1)}\vec{x} + \vec{b}^{(1)}$$

$$\vec{a}^{(1)} = \sigma(\vec{d}^{(1)})$$

Forward Step

One forward step describes how the input is processed throughout the whole network.

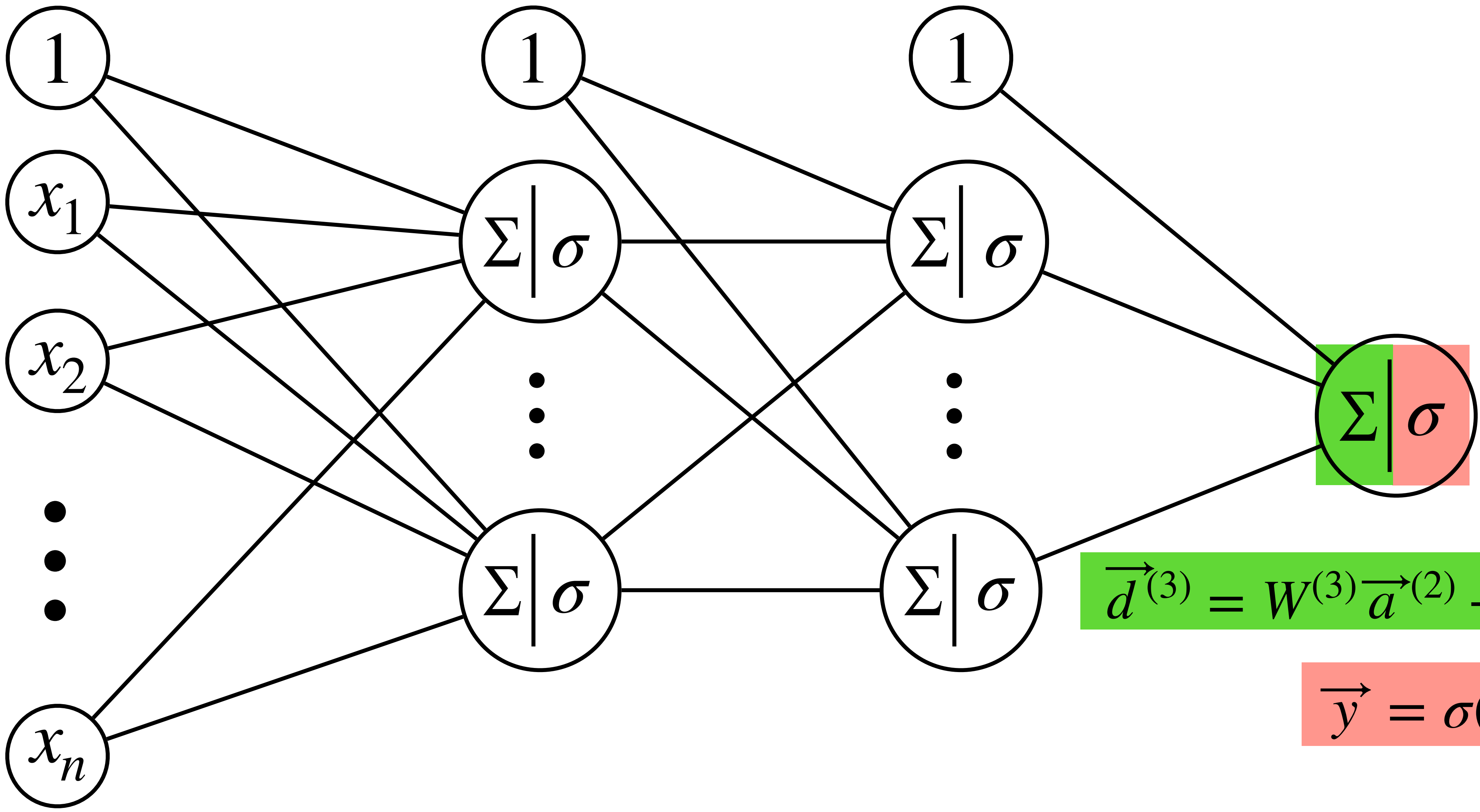


$$\vec{d}^{(2)} = W^{(2)}\vec{a}^{(1)} + \vec{b}^{(2)}$$

$$\vec{a}^{(2)} = \sigma(\vec{d}^{(2)})$$

Forward Step

One <u>forward step</u> describes how the input is processed throughout the whole network.

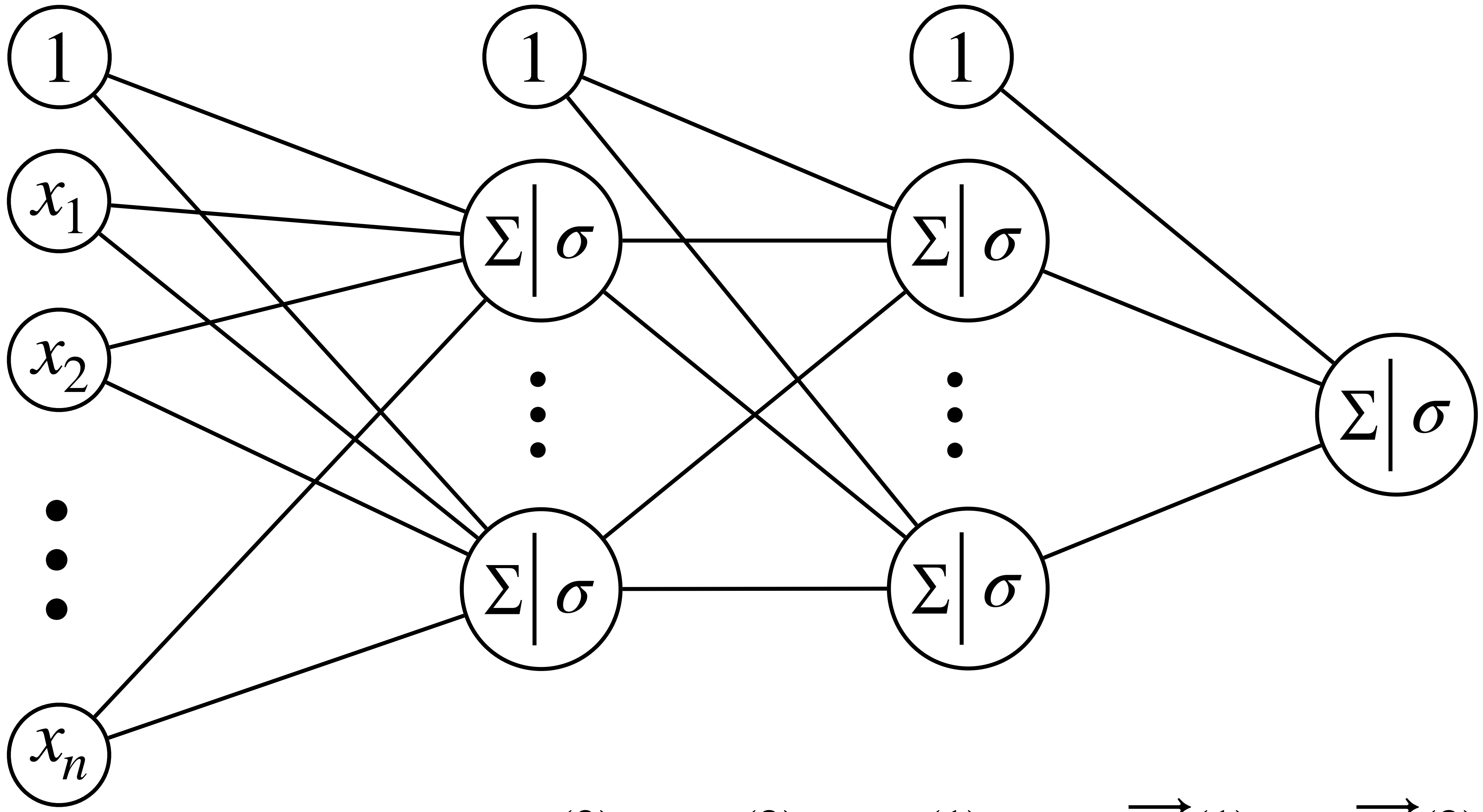


$$\vec{d}^{(3)} = W^{(3)}\vec{a}^{(2)} + \vec{b}^{(3)}$$

$$\vec{y} = \sigma(\vec{d}^{(3)})$$

Network Function

The network can be described by the function it implements, although we usually don't do this:



$$\vec{y} = \sigma(W^{(3)}\sigma(W^{(2)}\sigma(W^{(1)}\vec{x} + \vec{b}^{(1)}) + \vec{b}^{(2)}) + \vec{b}^{(3)})$$

Any questions?

Conclusion & Outlook

Conclusion

- We saw how a simple neuron model (McCulloch-Pitts) is able to implement simple logical gates.
- We discussed an extension of this model (Perceptron), which is able to learn autonomously from data.
- We found that it is not able to implement more complex functions (like XOR), because it can only learn a linear separation.
- Lastly we showed that a stacked multi-layer perceptron (MLP) is able to implement these more complex problems and formalized what a forward step in an MLP looks like.

Outlook

- The problem now is that we can’t use the perceptron learning rule to train a multi-layer perceptron.
- Although Frank Rosenblatt showed that an MLP can solve more complex problems he couldn’t came up with a way to train it.
- It was not before 1982 that the famous Backpropagation algorithm was invented that allows learning in multi-layer perceptrons.
- This will be the topic for the next lecture.

See you next week!

Resources

[w1] WeeKee at English Wikipedia (https://commons.wikimedia.org/wiki/File:Rosenblatt_21.jpg)

[w2] Rama at English Wikipedia (https://commons.wikimedia.org/wiki/File:Marvin_Minsky_at_OLPCb.jpg)

[w3] Rodrigo Mesquita at English Wikipedia (https://commons.wikimedia.org/wiki/File:Seymour_Papert.png)

[nyt] https://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html

[fr] F. Rosenblatt., "The perceptron: A probabilistic model for information storage and organization in the brain" *Psychological Review 65 (6): 386–408,* 1958.

[mp] M. Minsky, S. Papert, "Perceptrons: An Introduction to Computational Geometry" *MIT Press,* 1969.