Fundamentals of Neural Networks Part 1

Dan A. Rosa De Jesús
Ph.D. Student in Electrical and Computer Engineering
University of Texas at El Paso

April 25, 2020

danrosa@miners.utep.edu

Agenda

01. Overview

History, why neural networks and why now?

02. Neural Networks

Formal definitions, how are neural networks used, and types of learning.

03. Some Mathematics

Typical activation functions and why are they used for?, and the relevance of bias.

04. McCulloch-Pitts

Examples of neural network design for logic functions: AND, OR, and NOT

05. Data Representation

Relevance of data representation when training neural networks.

06. Hebb Learning Rule

Neural network design through the Hebb learning rule and examples for logic functions: AND and OR.

- **The 1940s: The Beginning of Neural Nets**
  - Warren McCulloch and Walter Pitts **designed the first neural network** combining many artificial neurons.
  - Donald Hebb **designed the first learning law for artificial neural networks**.
- **The 1950s and 1960s: The Golden Age of Neural Networks**
  - Frank Rosenblatt **developed a new class of artificial neural networks called perceptrons**.

SOME HISTORY *(Cont.)*

- **The 1970s: The Quiet Years**
  - Despite the demonstration of the limitations of perceptrons (i.e., single-layer nets), **research on neural networks continued**.
  - Many researchers in the field began to publish their work during the 1970s.
- **The 1980s: Renewed Enthusiasm**
  - **Backpropagation**
    - A method for propagating information about errors at the output units **back to the hidden units** had been discovered in the previous decade but had not gained wide publicity.

SOME HISTORY *(Cont.)*

- **Hardware implementation**
  - Another reason for renewed interest in neural networks is **improved computational capabilities**.

- As modern **computers become ever more powerful**, challenges arise in using machines effectively for tasks that are **relatively simple for humans**.

- For example: **Distinguishing** between **dogs** and **cats.**

- Traditional, sequential, logic-based computing **excels in many areas** but has been **less successful for other types of problems**.

- This **calls for new approaches** to **solving these problems** effectively.

One of those approaches includes the **neural network**.

WHY NEURAL NETWORKS AND WHY NOW? *(Cont.)*

- **Several factors** are responsible for the **recent interest in neural networks**:
  - High-speed digital computers.
  - Specialized hardware.
  - Parallel computing.
  - New sophisticated neural network architectures.

- However, **limitations** encountered in the inherently sequential nature of traditional computing **have motivated** new directions for neural network research.

- An **artificial neural network** is an **information-processing system** based on **generalizations of mathematical models** of **neural biology**:
  - Information processing occurs at **simple elements** called **neurons**.
  - Signals pass between neurons over **connection links**.
  - The connection links have **associated weights** that multiply the signals.
  - The **neurons apply activation functions** to the sum of weighted input signals **to determine their respective output signals**.

Traditional Computing (Sequence of Instructions)

Neural Network Computing (Data Driven)

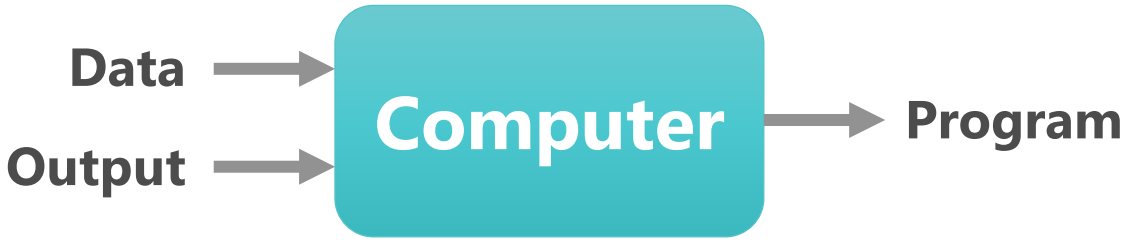


Figure 1. Traditional computing vs. neural network computing in terms of the computer's inputs and outputs.

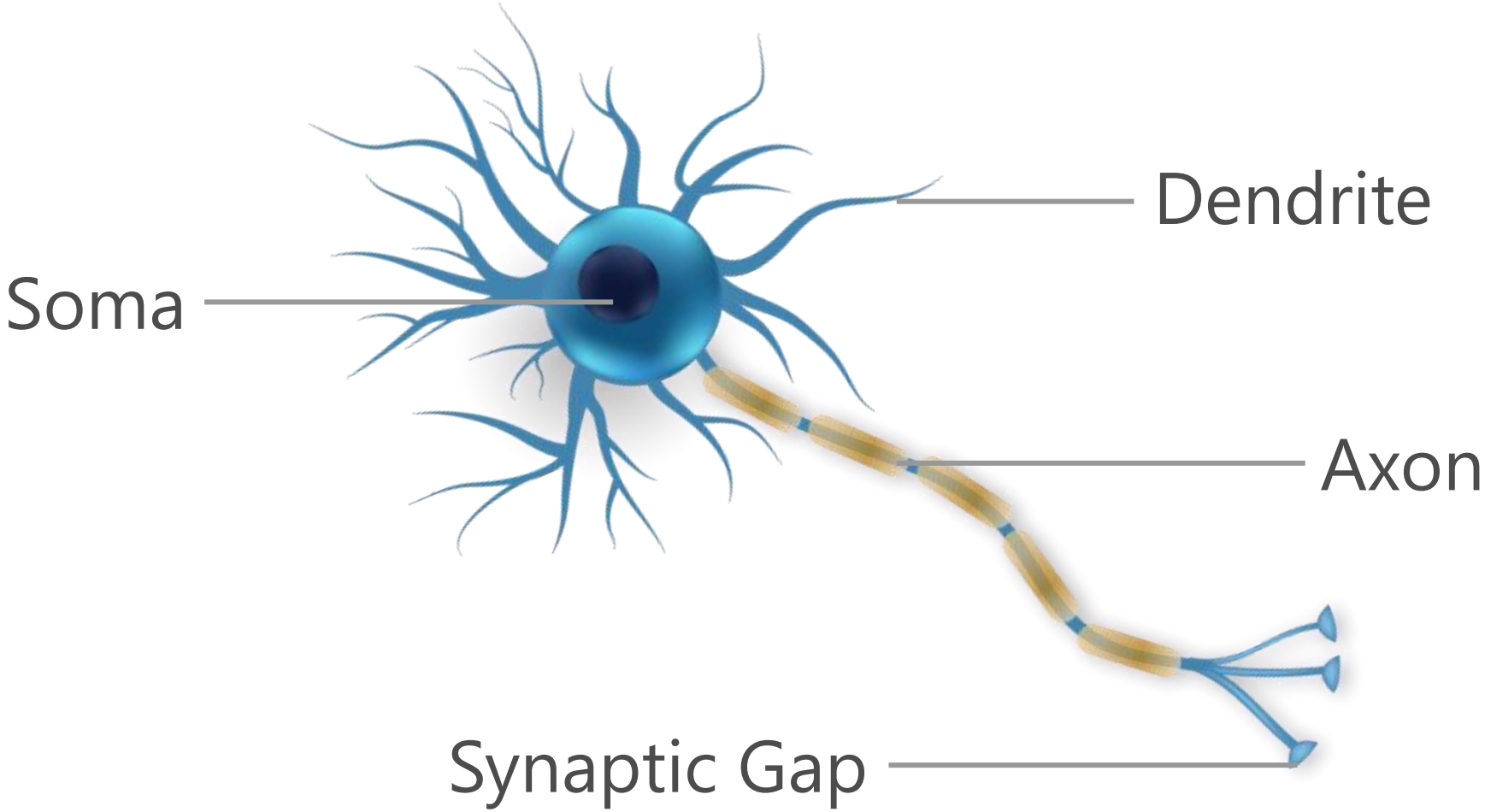
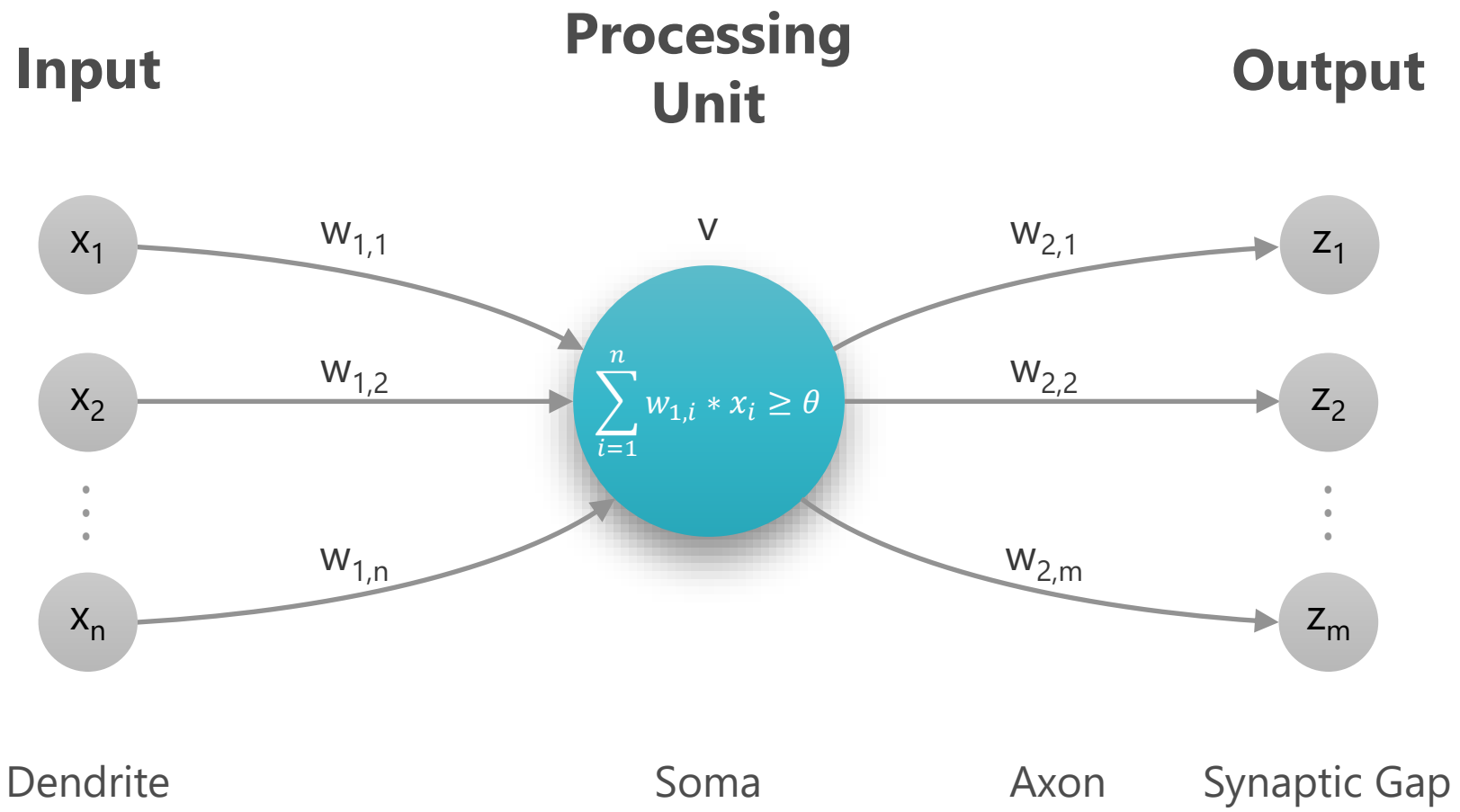


Figure 2. The biological neuron model.

THE DEFINITION OF NEURAL NETWORKS *(Cont.)*



x and ***z*** are vectors

$\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ *are matrices*

$w_{1,n} \in \boldsymbol{W_1}$ and $w_{2,m} \in \boldsymbol{W_2}$

$$\sum_{i=1}^{n} w_{1,i} * x_i \geq \theta$$

$$\boldsymbol{W_1} \cdot \boldsymbol{x} \geq \theta$$

If the condition is met, output = 1. Otherwise, output = 0.

Figure 3. Artificial Neuron Model (McCulloch-Pitts).

WHERE ARE NEURAL NETWORKS BEING USED

- **Signal Processing**
  - Noise reduction
- **Pattern Recognition**
  - Terrain classification
- **Medicine**
  - Protein 3D structure prediction
- **Speech Production and Recognition**
- **Business**
  - Fraud detection

- One of the **fundamental attributes** of how neural networks operate include their **architecture**.

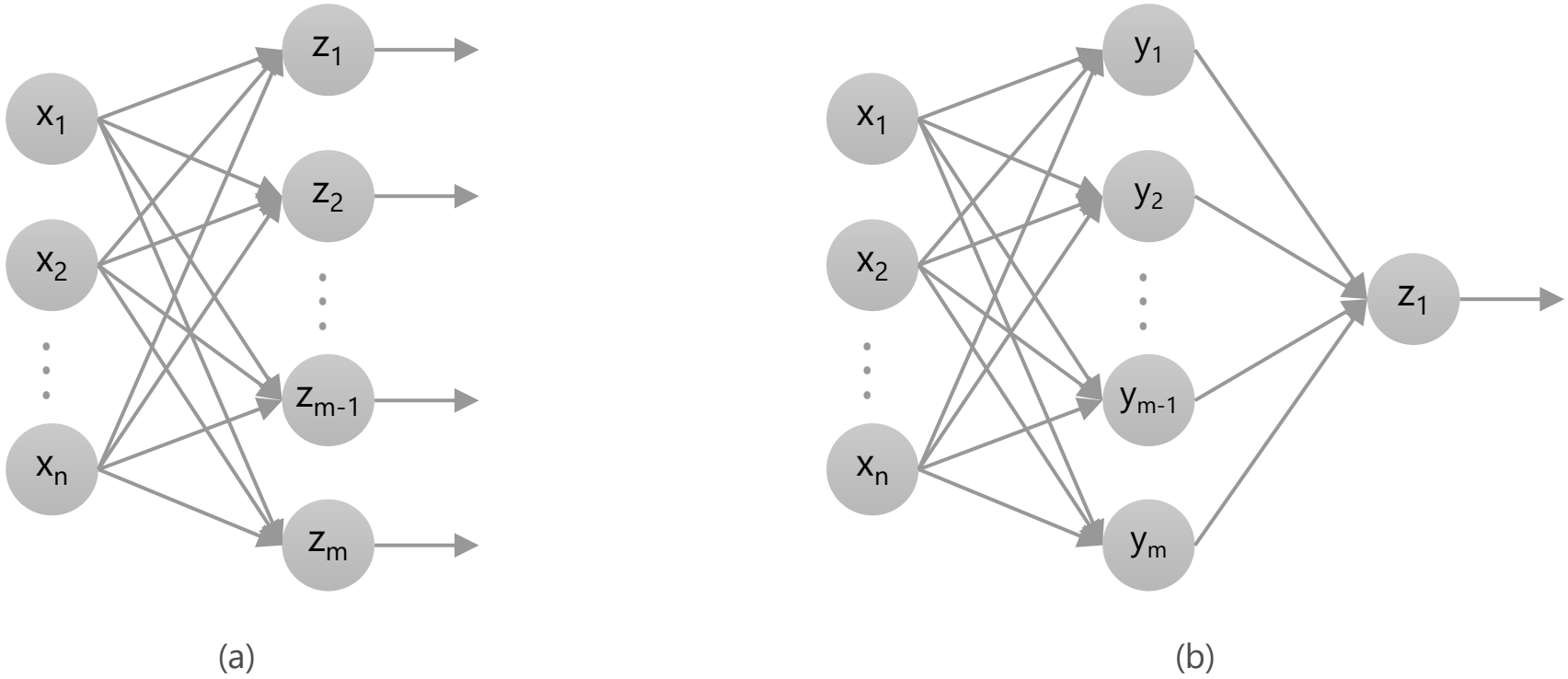


Figure 4. Single-layer neural network (a) and multi-layer neural network (b).

HOW ARE NEURAL NETWORKS USED? *(Cont.)*

- In addition to the architecture, the **methods for setting the values of the weights** are important distinguishing characteristics of different neural nets.
- Many of the tasks that neural networks can perform fall into the areas of **mapping**, **clustering**, and **constrained optimization**.
- There is some ambiguity in the labeling of training methods as **supervised** or **unsupervised**.

- **Supervised Learning**
  - **Training** involves by presenting a **sequence of training samples**, or patterns, each with **associated target outputs**.

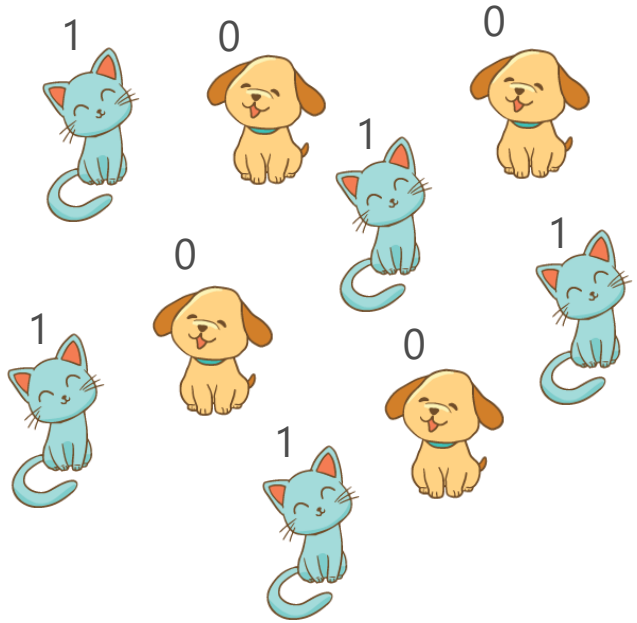


Figure 5. Animal samples with their respective labels: 0 (dog) and 1 (cat).

- **Unsupervised Learning**
  - Self-organizing neural networks **group similar input vectors together without the use of training data** to specify to which group each vector belongs.

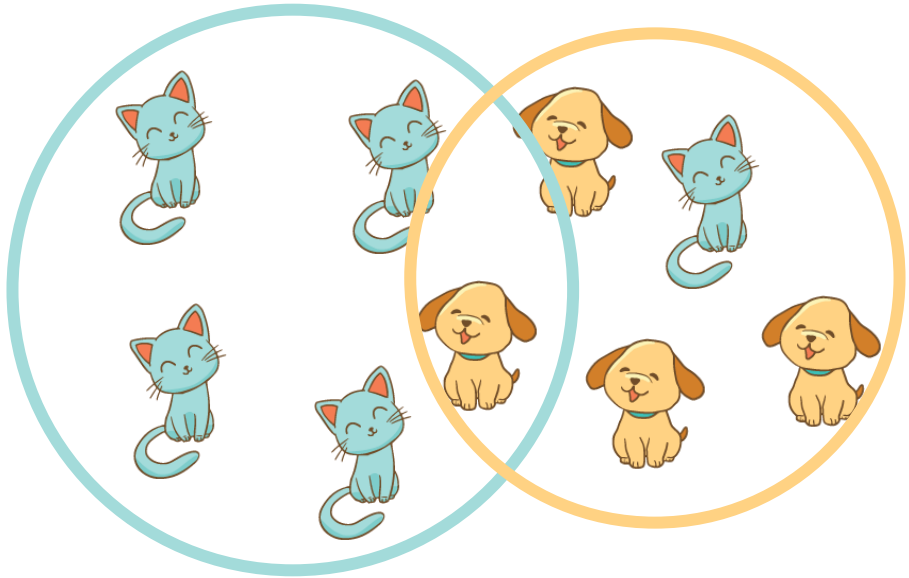


Figure 6. Animal samples in their respective clusters: dogs (left) and cats (right).

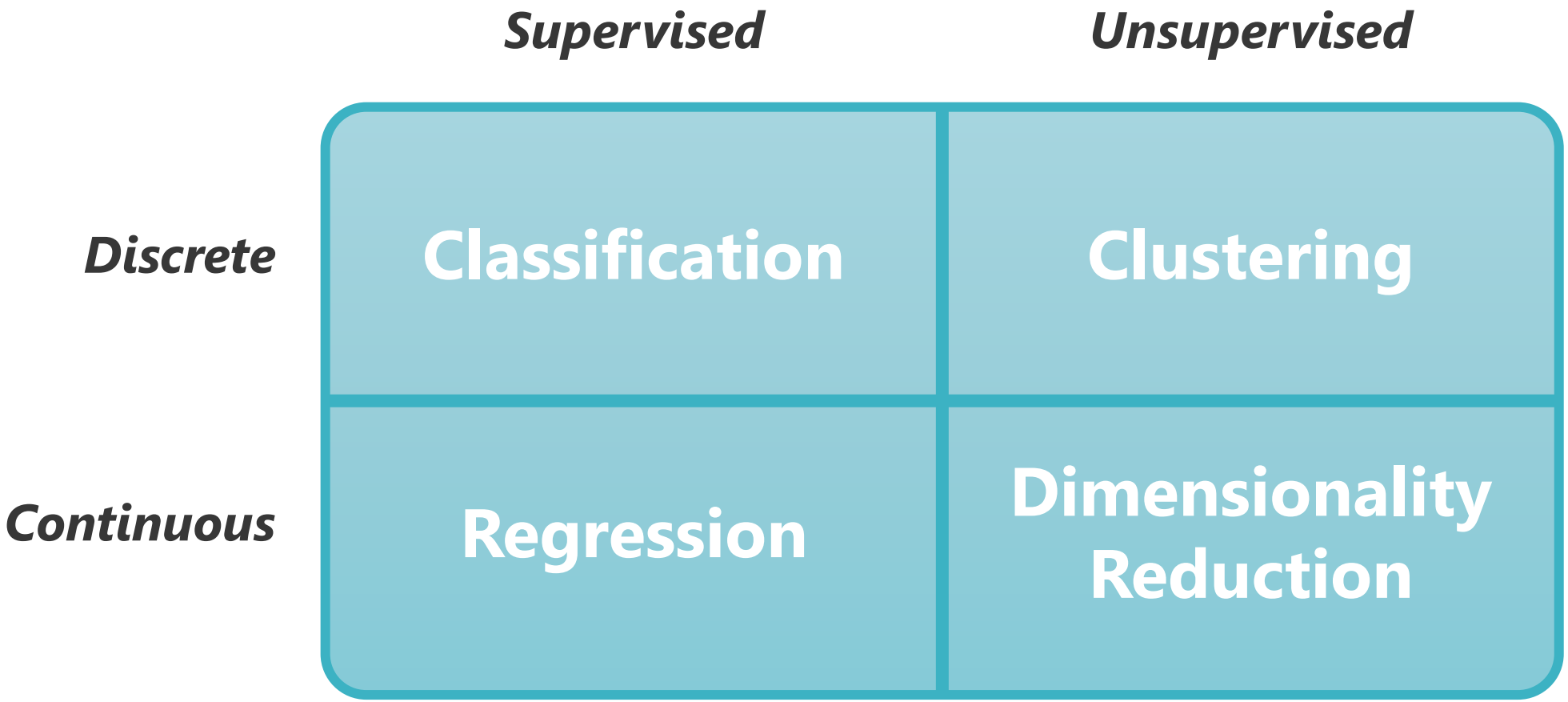


Figure 7. Neural network tasks based on the type of problem and learning.

- As mentioned before, the basic operation of an artificial neuron involves summing its weighted input signal and applying an output or **activation function**.

$$Output = f_{activation}\left(v = \sum_{i=1}^{n} w_{1,i} * x_i\right)$$

ACTIVATION FUNCTIONS *(Cont.)*

- Some **common activations** include:
  - Identity function:

$$f(v) = v, \qquad \forall v$$

  - Binary step function:

$$f(v) = \begin{cases} 0, & v < \theta \\ 1, & v \geq \theta \end{cases}$$

  - Sigmoid function (S function):

$$f(v) = \frac{1}{1 + e^{-\sigma v}}, \qquad \forall v$$

ACTIVATION FUNCTIONS *(Cont.)*

- Brief example:

$$f(v) = \frac{1}{1 + e^{-\sigma v}}, \qquad \forall v$$

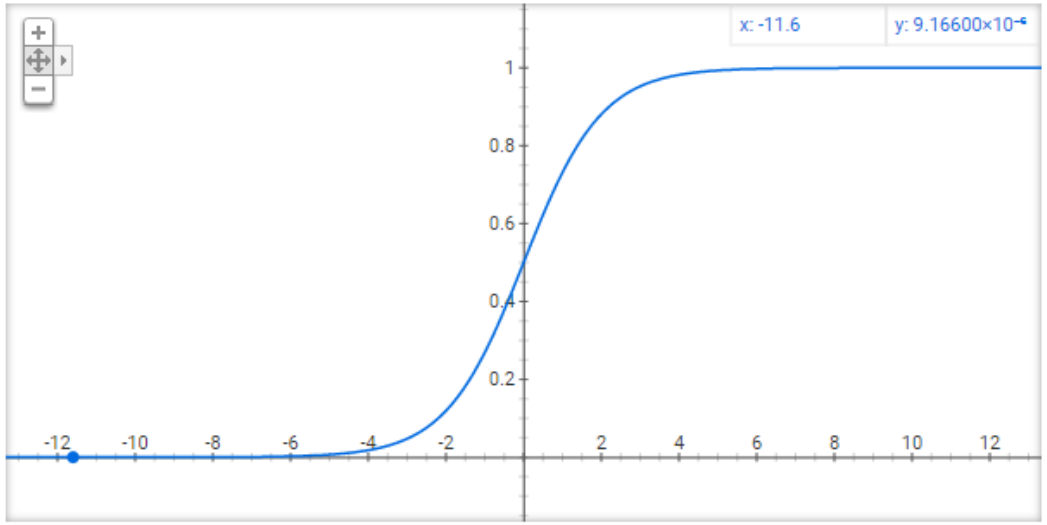
- Let us suppose we are classifying samples into two classes: 0 and 1. After summing the weighted input signals of two samples, we obtain $\boldsymbol{v_1 = -11.6}$ and $\boldsymbol{v_2 = 10}$, respectively. Compute the output of a certain neuron for $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$. **Consider $\sigma = 1$.**

- Brief example (Solution)

$$f(v_1 = -11.6) = \frac{1}{1 + e^{11.6}} = 0.000009166 \approx 0 \textbf{ (class 0)}$$

$$f(v_2 = \ \ 10.0) = \frac{1}{1 + e^{-10}} = 0.999954602 \approx 1 \textbf{ (class 1)}$$

- Brief example (Solution)



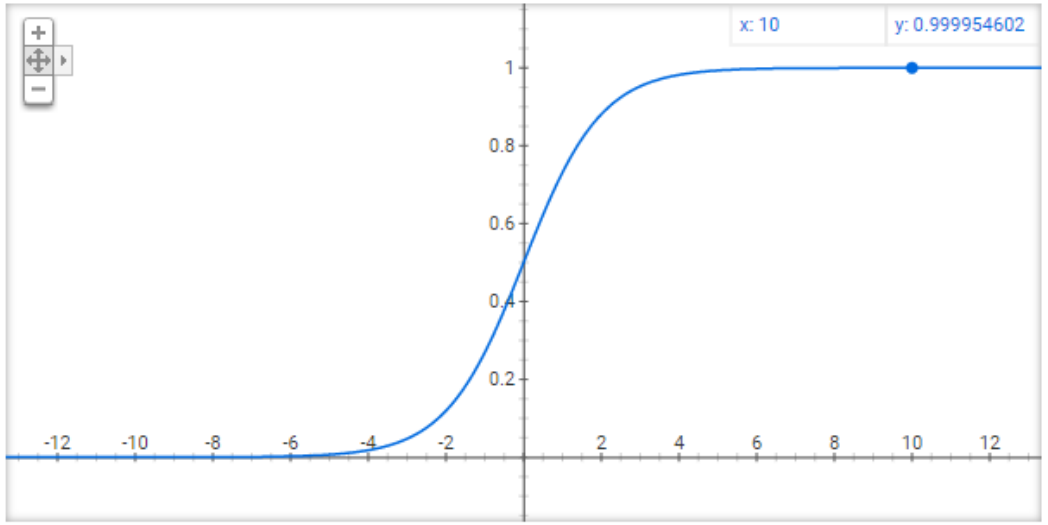


Figure 8. Sigmoid activation function with $\mathbf{v_1 = -11.6}$ (left) and $\mathbf{v_2 = 10}$ (right).

Has anyone noticed the **issue** with this model?

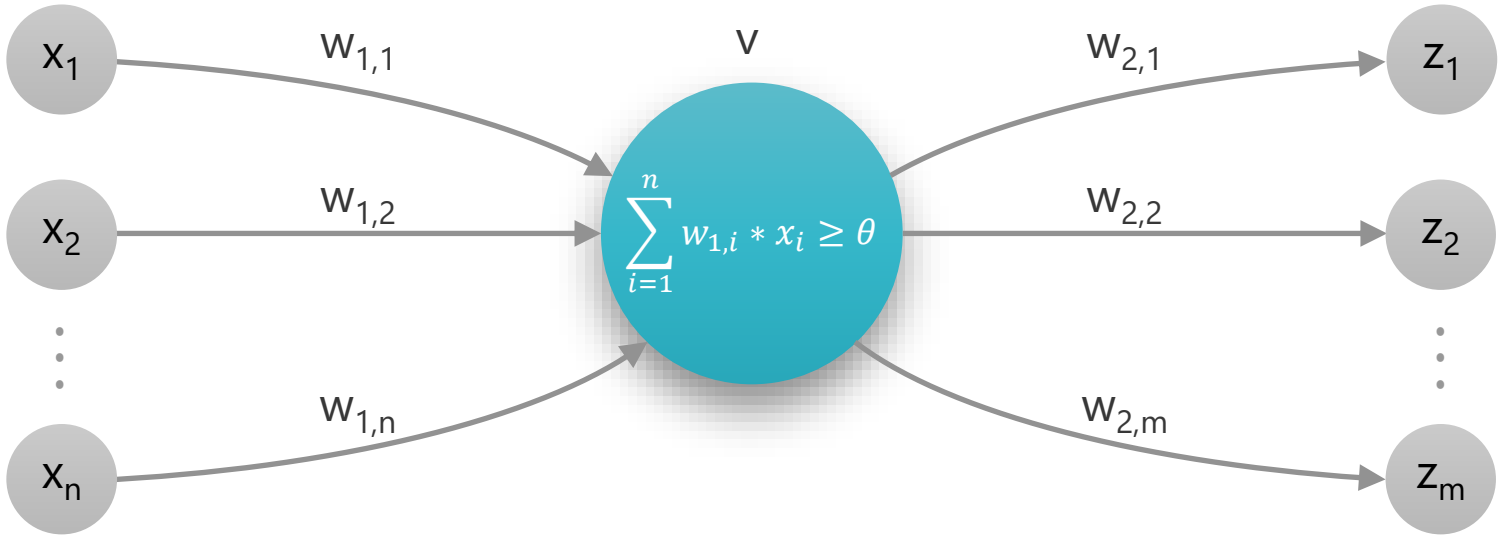


Figure 9. Artificial Neuron Model (McCulloch-Pitts).

TROUBLE IN PARADISE *(Cont.)*

- Let us consider the following scenario:

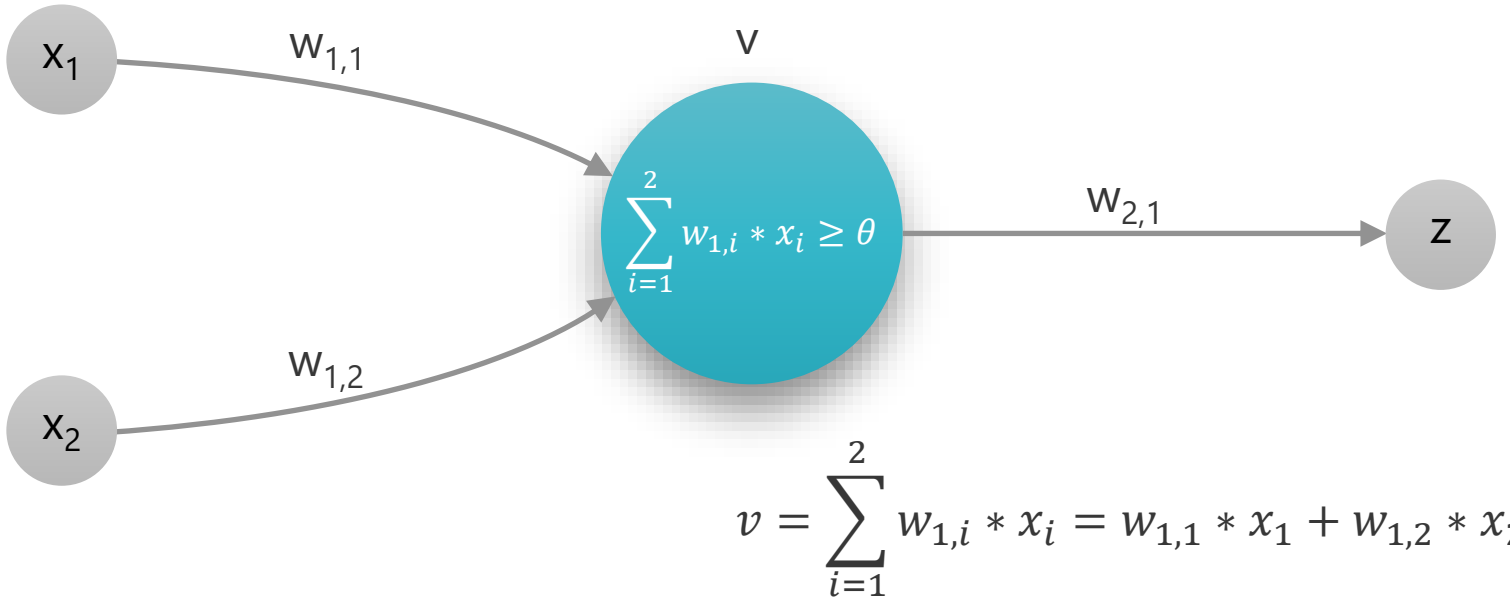


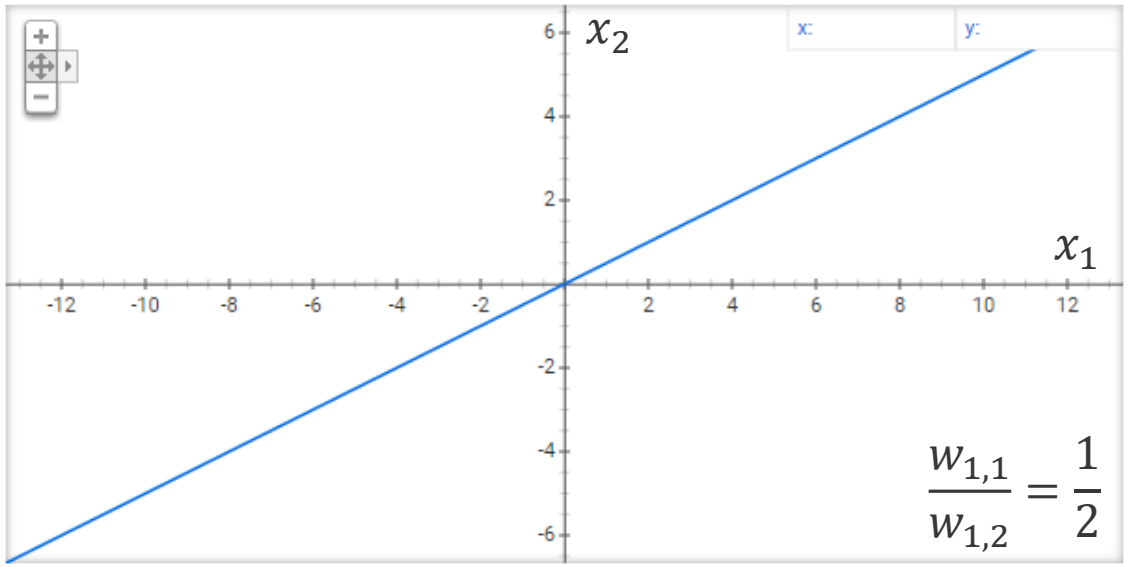
Figure 10. The artificial neuron model with two inputs and one output.

$$v = \sum_{i=1}^{2} w_{1,i} * x_i = w_{1,1} * x_1 + w_{1,2} * x_2$$

$$= w_{1,1} * x_1 + w_{1,2} * x_2 = 0 \quad \Rightarrow \quad x_2 = \frac{w_{1,1}}{w_{1,2}} x_1$$

No bias

The sign is negligible as it is embedded into the quotient.



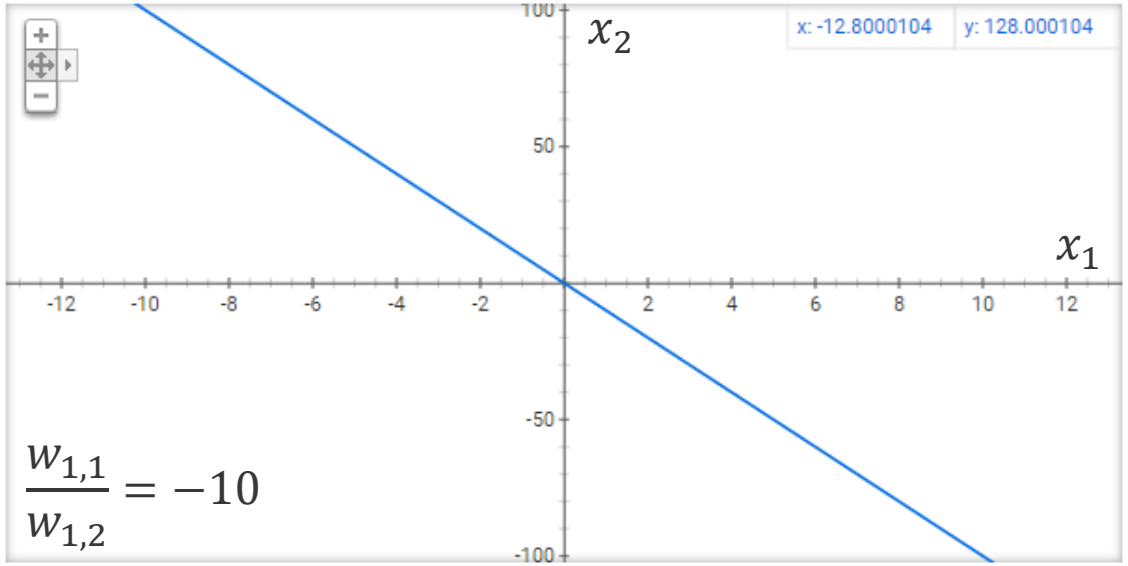
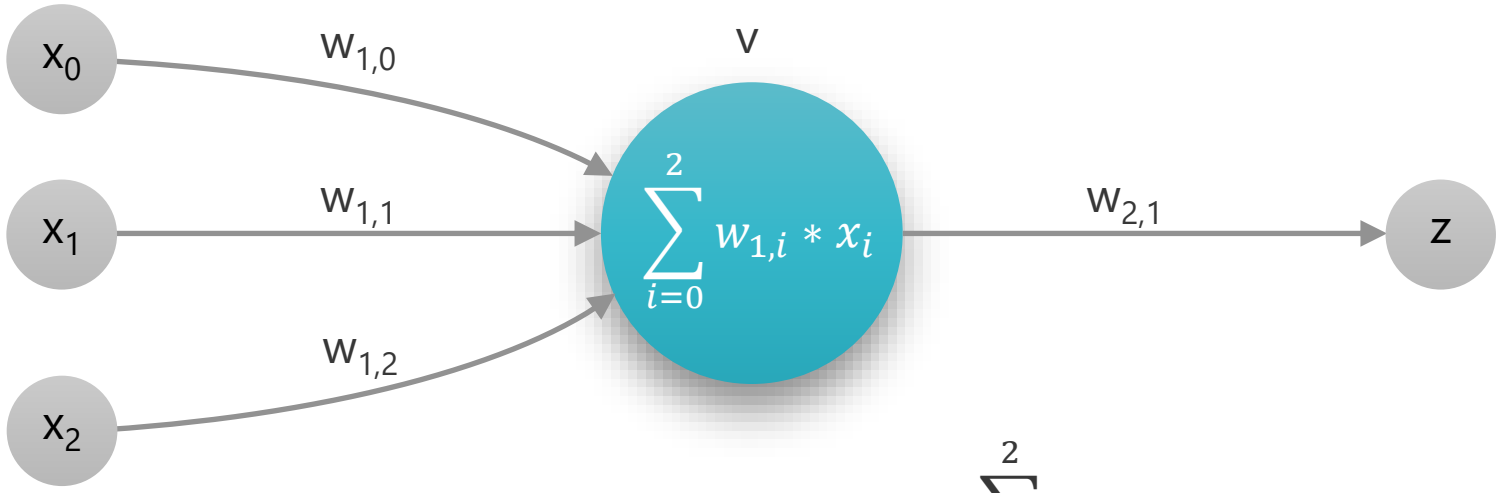


Figure 11. Graphical representation of v **without the bias**.

- Let us consider the following scenario:



$$v = w_{1,0} * x_0 + \sum_{i=1}^{2} w_{1,i} * x_i = w_{1,0} * x_0 + w_{1,1} * x_1 + w_{1,2} * x_2$$

$$x_0 = 1$$

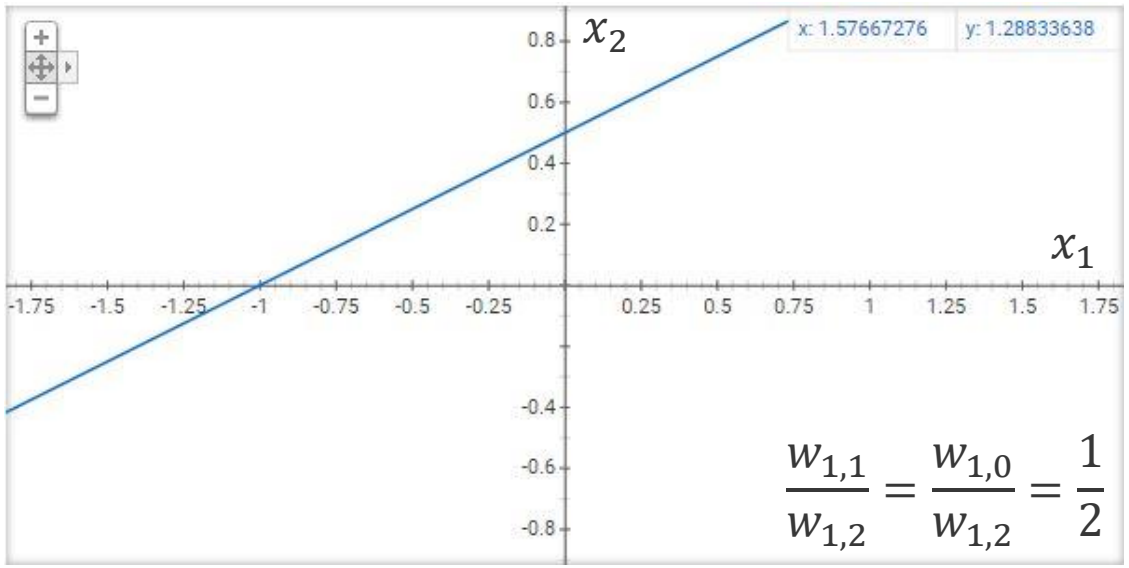
Figure 12. The artificial neuron model with three inputs, **including the bias**, and one output.

THE BIAS *(Cont.)*

$$v = w_{1,0} * x_0 + \sum_{i=1}^{2} w_{1,i} * x_i = w_{1,0} * x_0 + w_{1,1} * x_1 + w_{1,2} * x_2, \qquad x_0 = 1$$

$$= w_{1,1} * x_1 + w_{1,2} * x_2 = w_{1,0} \Rightarrow \quad x_2 = \frac{w_{1,1}}{w_{1,2}} x_1 + \frac{w_{1,0}}{w_{1,2}}$$

Bias



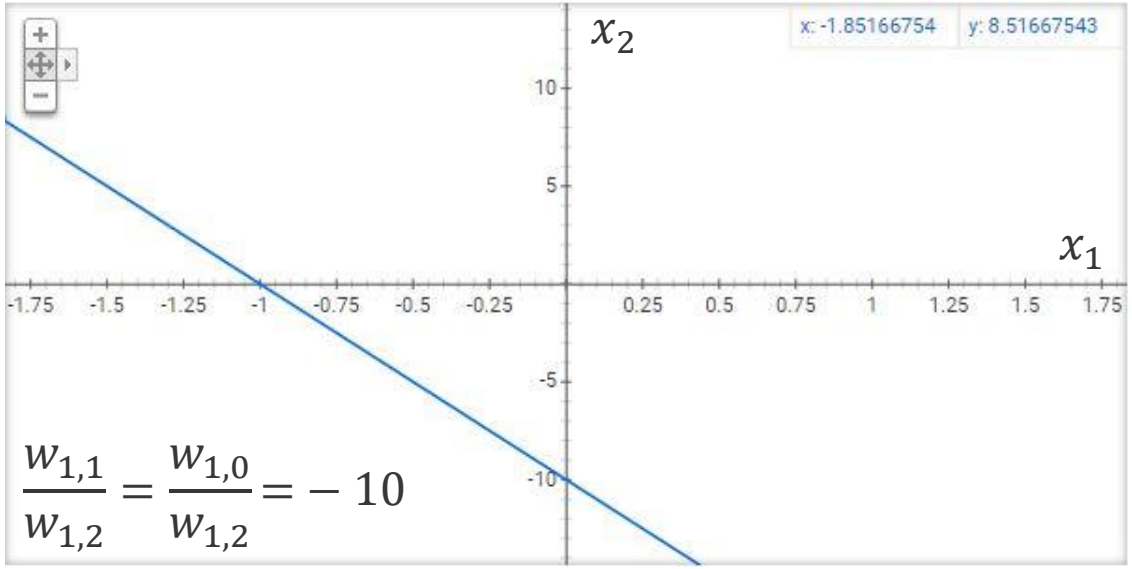


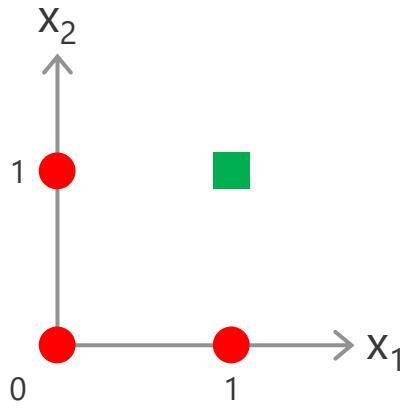
Figure 13. Graphical representation of v **with the bias**.

RECESS

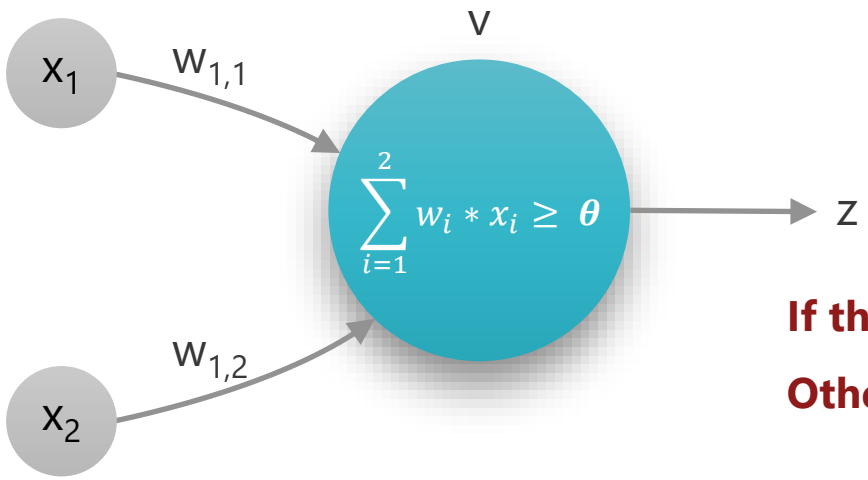
10 MINUTES

- The **AND** Function

x_1	x_2	z
0	0	0
0	1	0
1	0	0
1	1	1



Goal: To find $\mathbf{w_{1,1}}$, $\mathbf{w_{1,2}}$, and $\boldsymbol{\theta}$ such that the response z is the AND function.

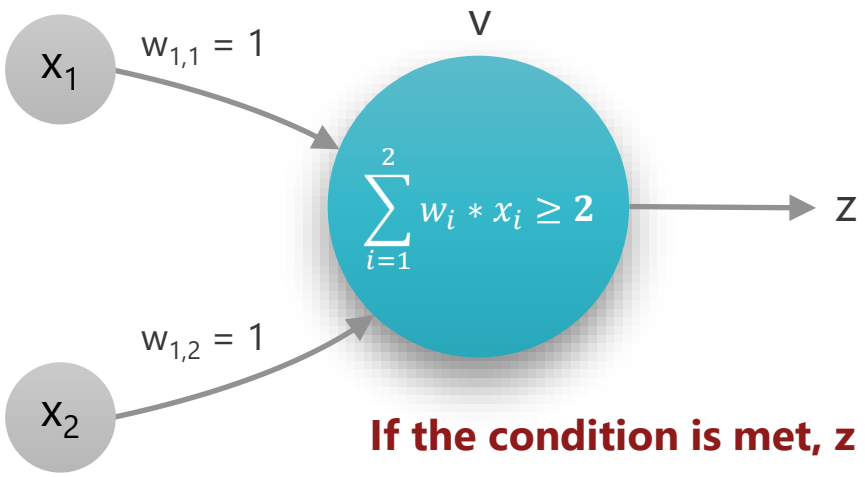


If the condition is met, z = 1.

Otherwise, z = 0.

SOME EXAMPLES: McCULLOCH-PITTS ARTIFICIAL NEURON *(Cont.)*

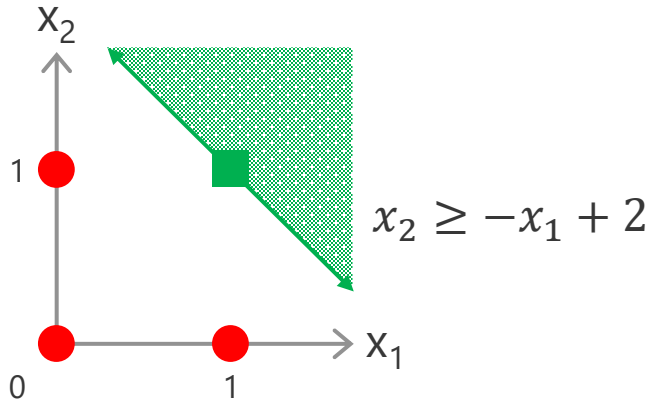
- Let us set $\mathbf{w_{1,1}} = \mathbf{w_{1,2}} = 1$ and $\theta = 2$.



If the condition is met, z = 1.

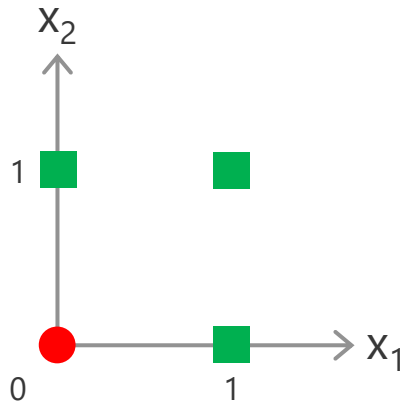
Otherwise, z = 0.

x_1	x_2	$v = \sum$	$z = \sum \geq 2$
0	0	0	False = 0
0	1	1	False = 0
1	0	1	False = 0
1	1	2	True = 1

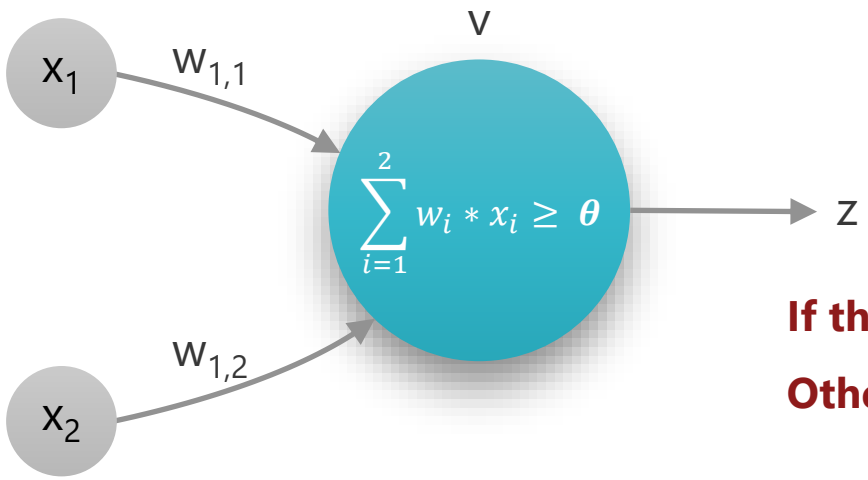


- The **OR** Function

x_1	x_2	z
0	0	0
0	1	1
1	0	1
1	1	1



Goal: To find $\mathbf{w_{1,1}}$, $\mathbf{w_{1,2}}$, and $\boldsymbol{\theta}$ such that the response z is the OR function.

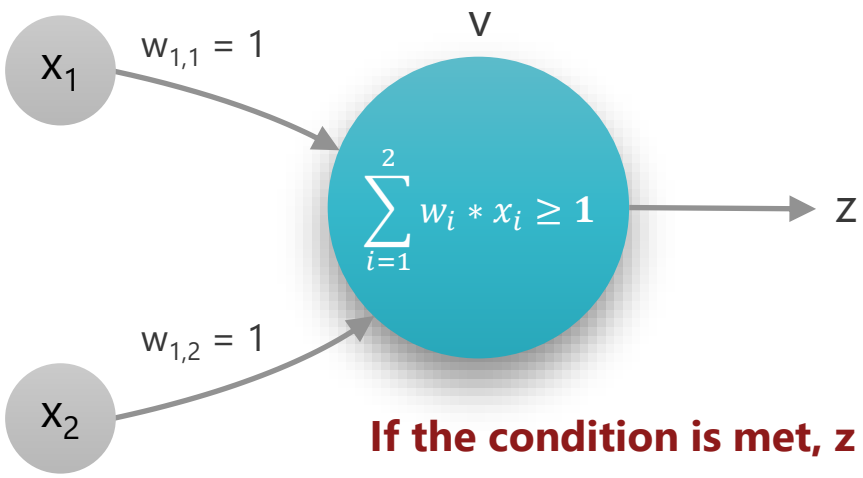


If the condition is met, z = 1.

Otherwise, z = 0.

SOME EXAMPLES: McCULLOCH-PITTS ARTIFICIAL NEURON *(Cont.)*

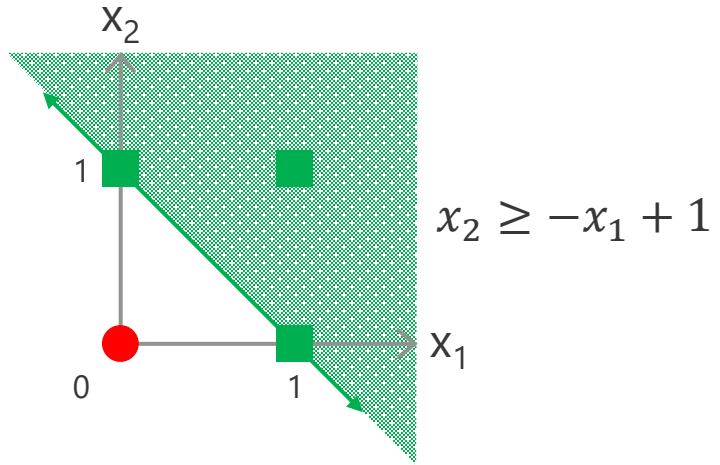
- Let us set $\mathbf{w_{1,1}} = \mathbf{w_{1,2}} = 1$ and $\theta = 1$.



If the condition is met, z = 1.

Otherwise, z = 0.

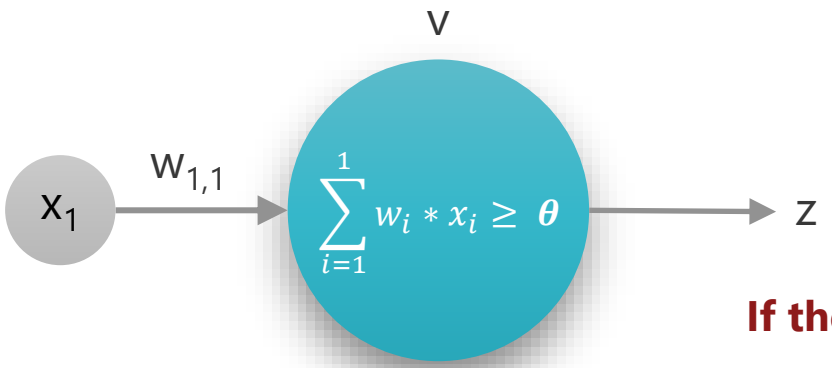
x_1	x_2	$v = \sum$	$z = \sum \geq 2$
0	0	0	False = 0
0	1	1	True = 1
1	0	1	True = 1
1	1	2	True = 1



- The **NOT** Function

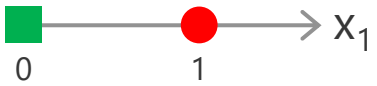
x_1	z
0	1
1	0

Goal: To find $\mathbf{w_{1,1}}$, $\mathbf{w_{1,2}}$, and $\boldsymbol{\theta}$ such that the response z is the NOT function.



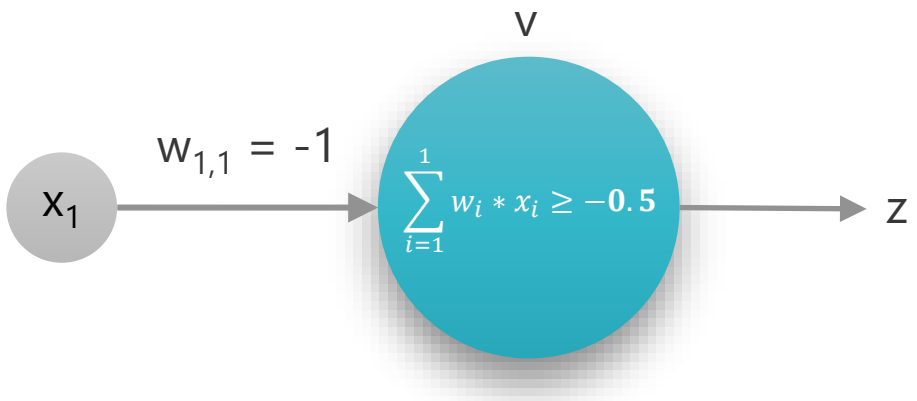
If the condition is met, z = 1.

Otherwise, z = 0.



SOME EXAMPLES: McCULLOCH-PITTS ARTIFICIAL NEURON *(Cont.)*

- Let us set $\mathbf{w_{1,1}}$ = -1 and θ = -0.5.

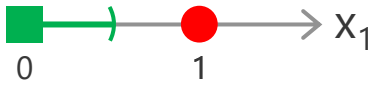


If the condition is met, z = 1.

Otherwise, z = 0.

x_1	$v = \sum$	$z = \sum \geq 2$
0	0	True = 1
1	1	False = 0

$x_1 < 0.5$



- Many early neural network models **used binary representation**, although in most cases **it can be modified to bipolar form**.
  - $0 \Rightarrow -1$
  - $1 \Rightarrow 1$
- The form of the data may change the problem from **one that can be solved** by a simple neural network **to one that cannot**.

- The earliest and simplest learning rule for a neural network is generally known as **the Hebb learning rule**.
- Hebb proposed that **learning occurs by modification of the synapse strengths** (weights) in a manner such that if two interconnected neurons are both "on" at the same time, then **the weight** between those neurons **should be increased**.
- If the data is **represented in bipolar form**, it is easy to express the desired weight update as:

$$w_i^{new} = w_i^{old} + x_i * z$$

- **Hebb Learning Rule Algorithm:**

Step 0. Initialize all weights:

$$w_i = 0, i \in [0, n].$$

Step 1. For each input training vector and target output pair, (s, t):

Step 2. Set activations for input units:

$$x_i = s_i, i \in [0, n].$$

Step 3. Set activations for output unit:

$$z = t, i \in [0, n].$$

Step 4. Adjust the weights:

$$w_i^{new} = w_i^{old} + x_i * z \in [0, n].$$

- Training a network using the Hebb learning rule for the AND function:

Can you anticipate what will happen during training when using binary inputs and targets?

Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * z$	$x_2 * z$	$x_3 * z$	$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$

- Training a network using the Hebb learning rule for the AND function:

Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * \mathrm{z}$	$x_2 * \mathrm{z}$	$x_3 * \mathrm{z}$	$w_{1,1}^{old} + x_1 * \mathrm{z}$	$w_{1,2}^{old} + x_2 * \mathrm{z}$	$w_{1,0}^{old} + b * \mathrm{z}$
0	0	1	0	0	0	0	0	0	0

HEBB NET *(Cont.)*

- Training a network using the Hebb learning rule for the AND function:

Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * z$	$x_2 * z$	$x_3 * z$	$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
0	0	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0

- Training a network using the Hebb learning rule for the AND function:

Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * z$	$x_2 * z$	$x_3 * z$	$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
0	0	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0

- Training a network using the Hebb learning rule for the AND function:

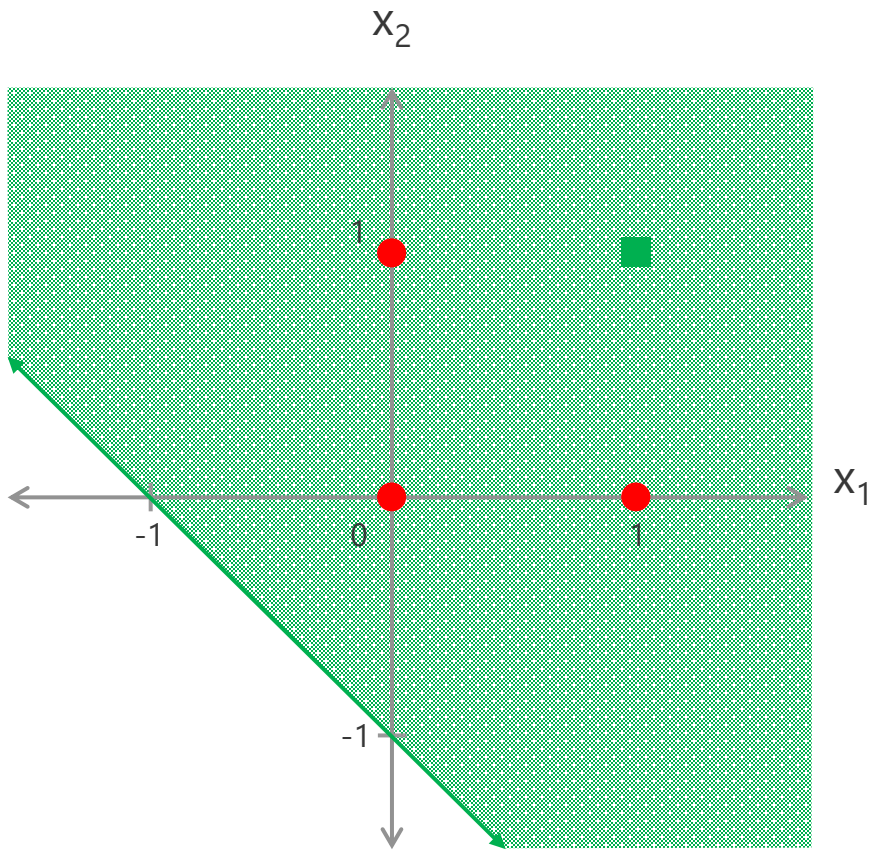
Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * \mathrm{z}$	$x_2 * \mathrm{z}$	$x_3 * \mathrm{z}$	$w_{1,1}^{old} + x_1 * \mathrm{z}$	$w_{1,2}^{old} + x_2 * \mathrm{z}$	$w_{1,0}^{old} + b * \mathrm{z}$
0	0	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1

- Training a network using the Hebb learning rule for the AND function.

Wrong solution. Why?

Weights		
$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
0	0	0
0	0	0
0	0	0
1	**1**	**1**

$$x_2 \geq -x_1 - 1$$



- Training a network using the Hebb learning rule for the AND function:

Let is give it a second try by using binary inputs and bipolar targets?

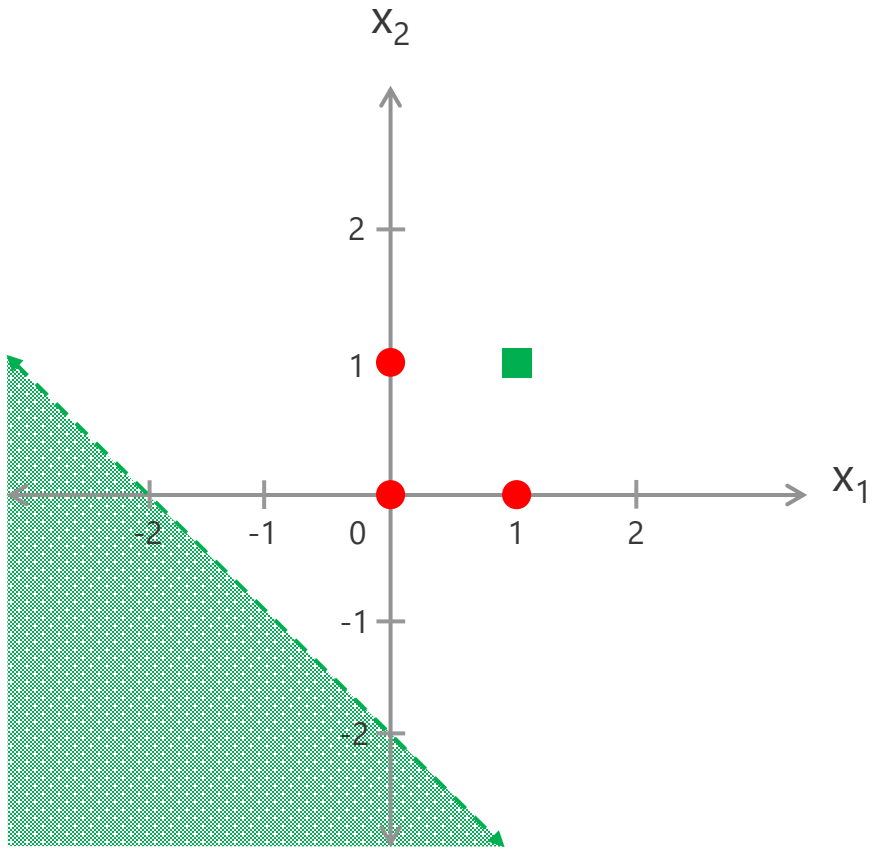
Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * \mathrm{z}$	$x_2 * \mathrm{z}$	$x_3 * \mathrm{z}$	$w_{1,1}^{old} + x_1 * \mathrm{z}$	$w_{1,2}^{old} + x_2 * \mathrm{z}$	$w_{1,0}^{old} + b * \mathrm{z}$
0	0	1	-1	0	0	-1	0	0	-1
0	1	1	-1	0	-1	-1	0	-1	-2
1	0	1	-1	-1	0	-1	-1	-1	-3
1	1	1	1	-1	-1	-1	-2	-2	-4

- Training a network using the Hebb learning rule for the AND function.

The result is even worst!

Weights		
$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
0	0	-1
0	-1	-2
-1	-1	-3
-2	**-2**	**-4**

$$x_2 < -x_1 - 2$$



HEBB NET *(Cont.)*

- Training a network using the Hebb learning rule for the AND function:

Let is give it a third try by using bipolar inputs and targets?

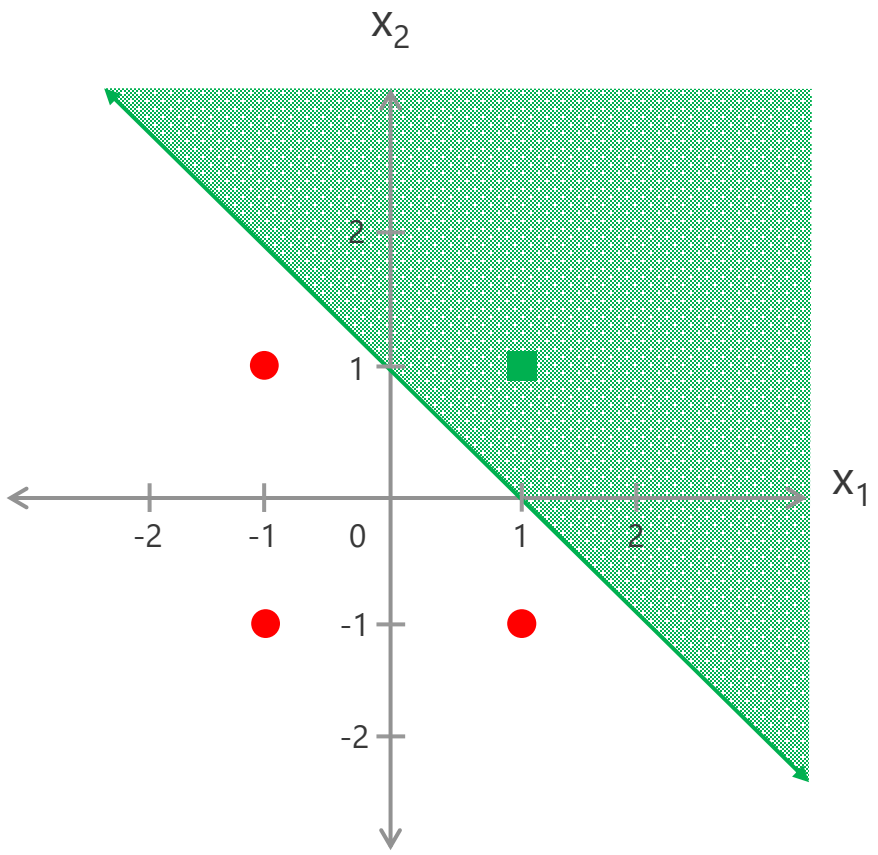
Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * z$	$x_2 * z$	$x_3 * z$	$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
-1	-1	1	-1	1	1	-1	1	1	-1
-1	1	1	-1	1	-1	-1	2	0	-2
1	-1	1	-1	-1	1	-1	1	1	-3
1	1	1	1	1	1	1	2	2	-2

- Training a network using the Hebb learning rule for the AND function:

At last, we obtained a correct answer.

Weights		
$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
1	1	-1
2	0	-2
1	1	-3
2	**2**	**-2**

$$x_2 \geq -x_1 + 1$$



- Training a network using the Hebb learning rule for the AND function:

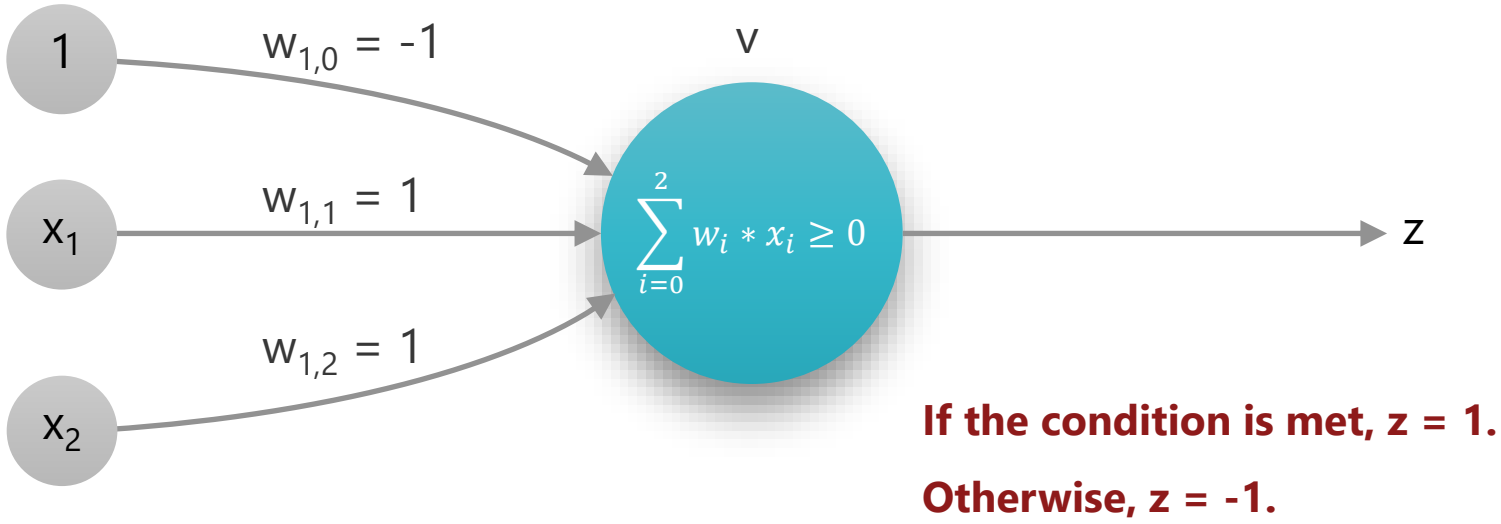


Figure 13. Hebb net for the AND function with bipolar inputs and targets.

- Training a network using the Hebb learning rule for the OR function:

Using bipolar inputs and targets.

Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * z$	$x_2 * z$	$x_3 * z$	$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
-1	-1	1	-1						
-1	1	1	1						
1	-1	1	1						
1	1	1	1						

- Training a network using the Hebb learning rule for the OR function:

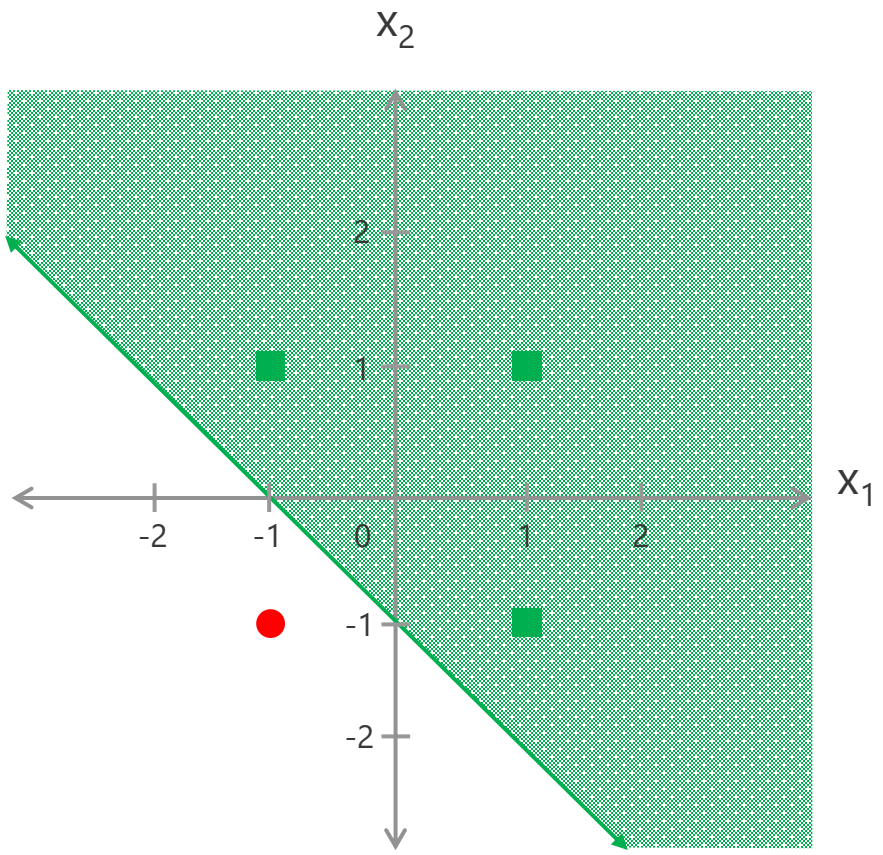
Using bipolar inputs and targets.

Input			Target	Weight Changes			Weights		
-	-	-	-	-	-	-	0	0	0
x_1	x_2	b	z	$x_1 * z$	$x_2 * z$	$x_3 * z$	$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
-1	-1	1	-1	1	1	-1	1	1	-1
-1	1	1	1	-1	1	1	0	2	0
1	-1	1	1	1	-1	1	1	1	1
1	1	1	1	1	1	1	2	2	2

- Training a network using the Hebb learning rule for the OR function:

Weights		
$w_{1,1}^{old} + x_1 * z$	$w_{1,2}^{old} + x_2 * z$	$w_{1,0}^{old} + b * z$
1	1	-1
0	2	0
1	1	1
2	2	2

$$x_2 \geq -x_1 - 1$$



- Training a network using the Hebb learning rule for the OR function:

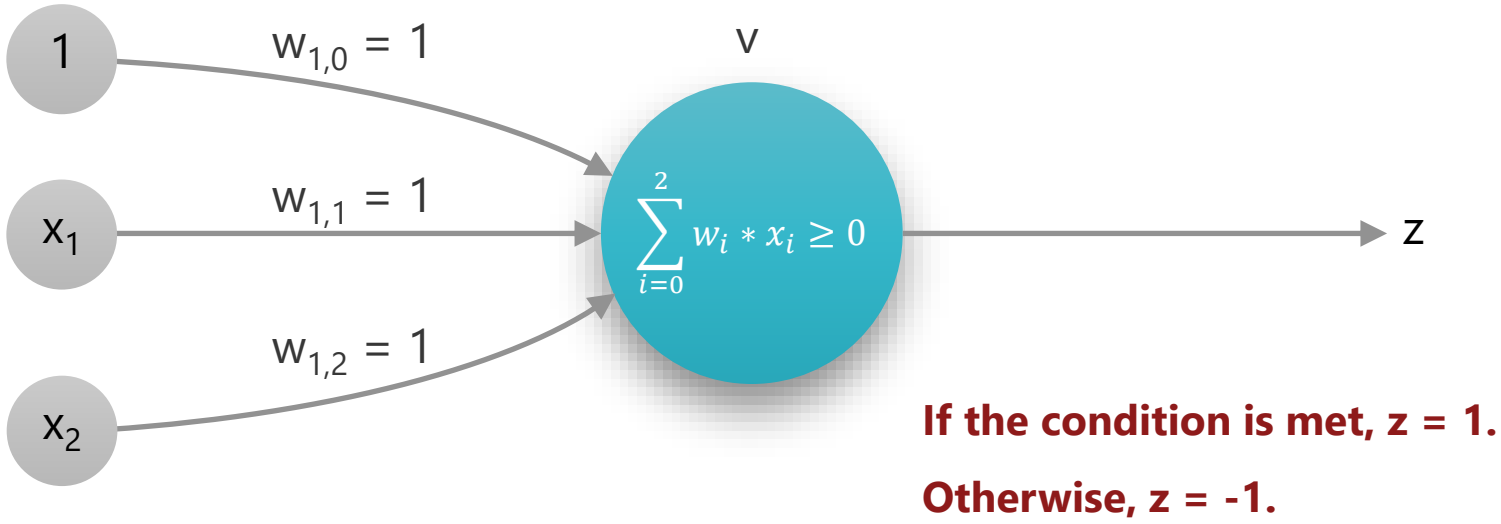


Figure 14. Hebb Net for the OR function with bipolar inputs and targets.

WHAT'S NEXT?

- In our **next session** we will:
  - Discuss the differences between the **McCulloch-Pitts** and **Perceptron** neuron models.
  - Discuss the limitations of the **Hebb learning rule**.
  - Introduce and discuss the **perceptron learning rule** which includes a new parameter, the **learning rate**.
  - **Implement** the perceptron learning rule in **Python**.

THANK **YOU**!