From the First Artificial Neuron to Multilayer Perceptron

Lecture_03 Data Mining and Text Analytics
Postgraduate Programme in Artificial Intelligence for Business and Society
IULM University

A.Y. 2025/2026
Professor Alessandro Bruno

Seeking Patterns

Back in the 1950s, ethologists discerned some patterns in the behaviour of animals.

"Newly hatched ducklings exhibit the ability to tell apart the properties of the things they see moving around them. It turns out that ducklings can imprint not just on the first living creature they see moving, but on inanimate thigs as well.

Mullard ducklings can imprint on relational concepts embodied by the objects around them.

"If upon birth the ducklings see two moving red objects, they will later follow two objects of the same colour (even if those latter objects are blue, not red)".

They also exhibit the ability to discern dissimilarity. If the first moving objects they see are, for example, a cube and a rectangular prism, they will recognise that the objects have different shapes. They will later follow two objects that are different in shape (a pyramid and a cone) while they will ignore two objects having the same shape.

Seeking Patterns

Newly hatched ducklings detect patterns in what they see (similarity and dissimilarity) with the briefest exposure to sensory stimuli.

They can act upon sensory stimuli once they detect similarity and/or dissimilarity.

Today's AI is way far from implementing such tasks. However, it does have something in common with the ducklings.

That's the ability to pick out and learn about patterns in data

When **Frank Rosenblatt** invented the **Perceptron** in the late 1950s, that represented a breakthrough in the AI landscape as it was the first outstanding "brain-inspired" algorithm that could learn about patterns in data simply by examining the data.

Perceptron's roots

Perceptron's roots

- "The perceptron's roots lie in a paper published in **1943** by an unlikely combination of a **philosophically minded neuroscientist** in his mid-forties and a **homeless teenager**."
- Warren **McCulloch** was an American neurophysiologist trained in philosophy, psychology, and medicine.
- During the 1930s, he worked on neuroanatomy, creating maps of the connectivity of parts of monkey brains. Afterward, he focused on the "logic of the brain".
- The work of mathematicians and philosophers, such as Alan Turing, Alfred North Whitehead, and Bertrand Russell suggested ties and connections between computation and logic.

Perceptron's roots

Here is an example of a logical proposition - "If P is true AND Q is true, then S is true"

The assertion was that all computaion could be reduced to such logic.

Here is the big question that prompted Warren McCulloch's research at that time: "If the brain is a computational device, as many think it is, how does it implement such logic?"

Perceptron's roots

McCulloch met Walter **Pitts** (a prodigiously talented teenager) in the 1940s at the University of Illinois.

Walter Pitts was a "runaway" from a family that could not appreciate his genius.

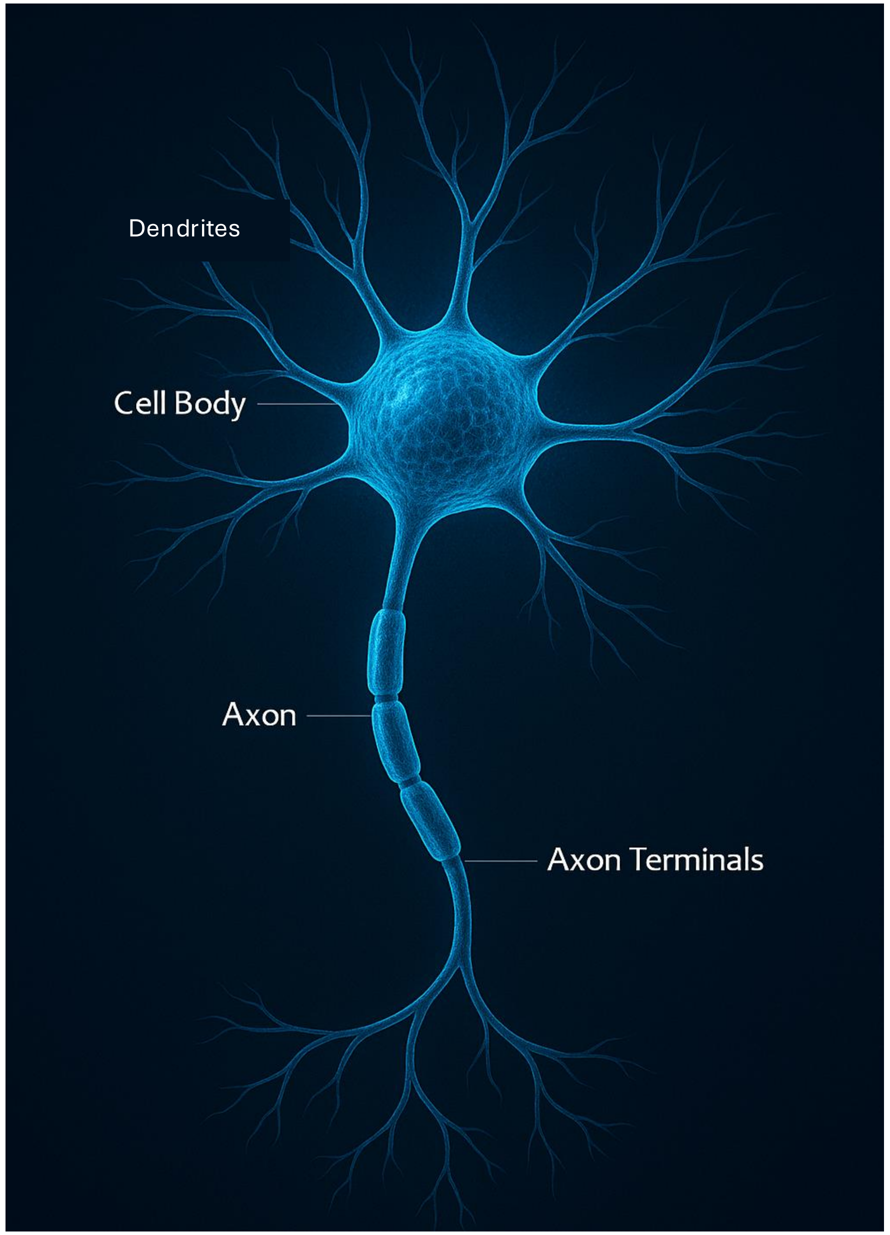
McCulloch and his wife, Rook, gave Walter Pitts a home and followed endless evenings discussing how the brain worked.

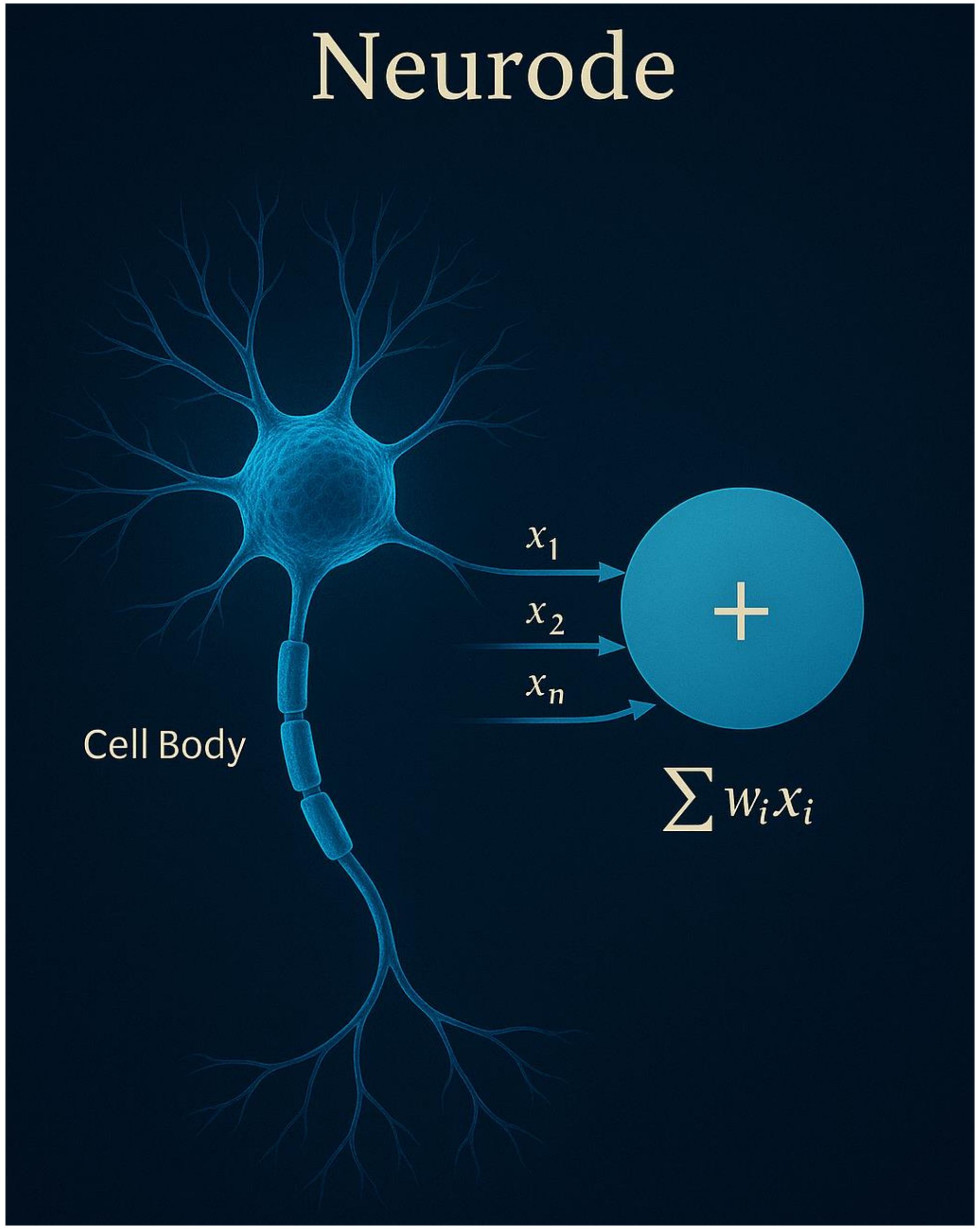
In 1943, a paper co-authored by McCulloch and Pitts and titled "A Logical Calculus of the Ideas Immanent in Nervous Activity" was published.

In that work both authors proposed a simple model of a biological neuron.

Perceptron's roots

- Let's begin with a **biological neuron**:
  - The neuro's cell body receives inputs via its treelike projections (**DENDRITES**).
  - The cell body performs some computation on the inputs.
  - Based on the computation results, the cell body may send signals (**spiking signals**) along another longer projection called **AXON**.
  - The mentioned signal travels through **Axon terminals** to communicate with **neighbouring neurons**.





Perceptron's roots

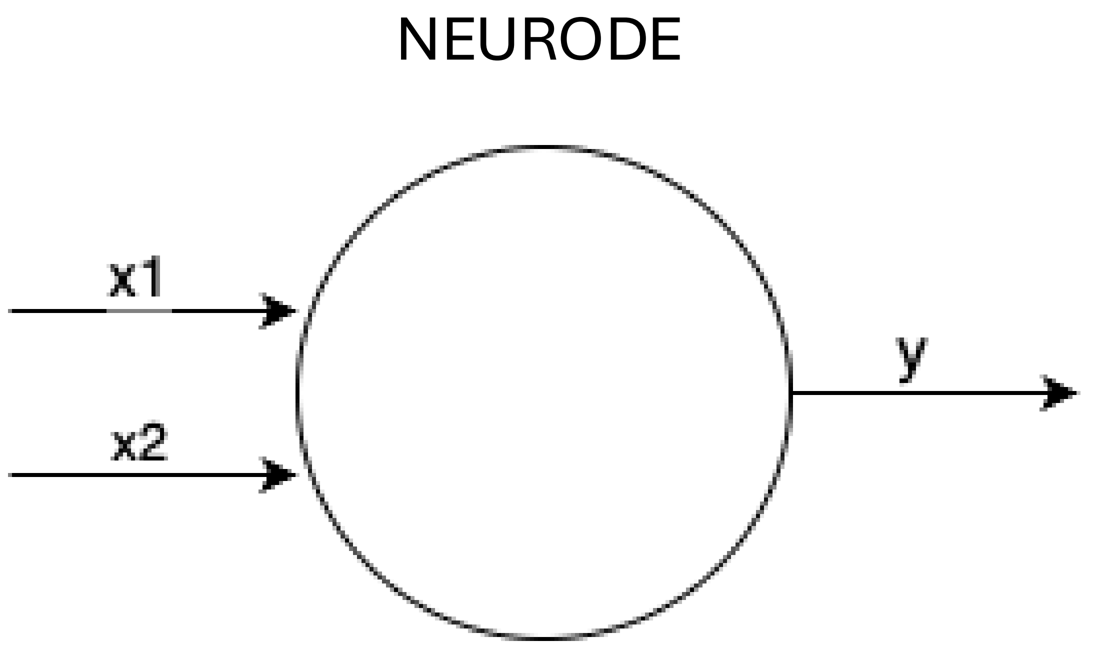
- **McCulloch and Pitts** turned out with a simple computational model, an artificial neuron.
- The artificial neuron, also called **NEURODE (neuron + node)**, could implement basic BOOLEAN logical operations such as AND, OR, NOT, and so on, which are the building blocks of the digital computation.

Zooming in on Neurode

How does a Neurode work?

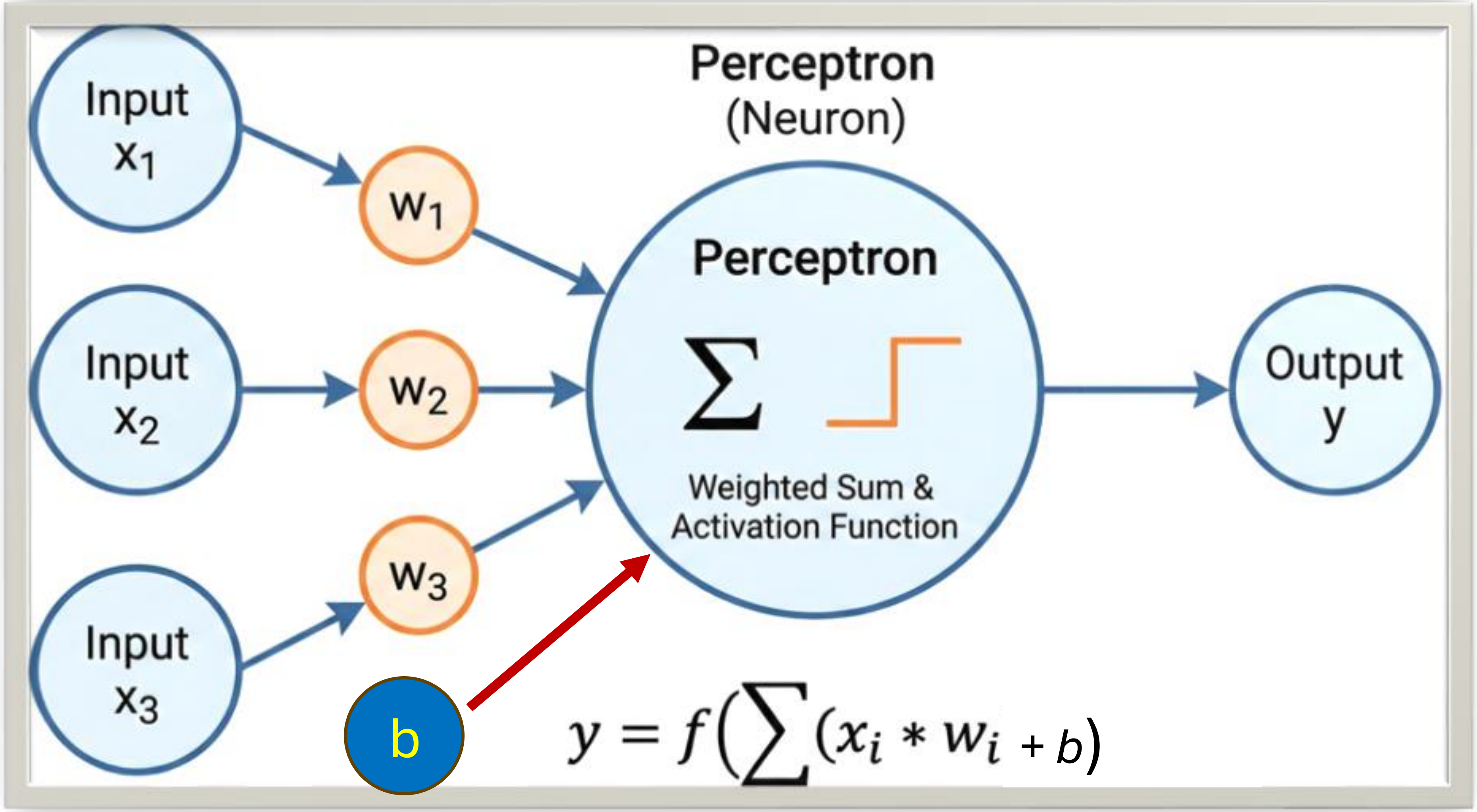
- The NEURODE performs a simple computation:
  - **Given the** Inputs: **$x1, x2 \in \{0, 1\}$**
  - **Sum the inputs**.
  - **Compare** the sum to a threshold (θ).
  - **Output**: y equals 1 if the sum of x1 and x2 is greater than the threshold, 0 otherwise.
  - $$y = \left\{ \begin{array}{ll} 1 & \text{if } (x1 + x2) \geq \theta \\ 0 & \text{else} \end{array} \right\}$$
- This mechanism can implement basic Boolean logic operations: **AND, OR, NOT**, forming fundamental building blocks of digital computation.

Graphical Representation of Neurode



y = x1 + x2

Moving on to Perceptron - Rosenblatt



$$y = f\left(\sum (x_i * w_i + b)\right)$$

How does Perceptron differ from Neurode?

As noticeable in the previous slide, new elements make their way into Perceptron with respect to Neurode.

What are **w**$_i$ and **b**?

We will get there in a minute.

Firstly, it is worth diving into Rosenblatt's intuition and the way he tackled "learning" processes.

Rosenblatt's Intuition

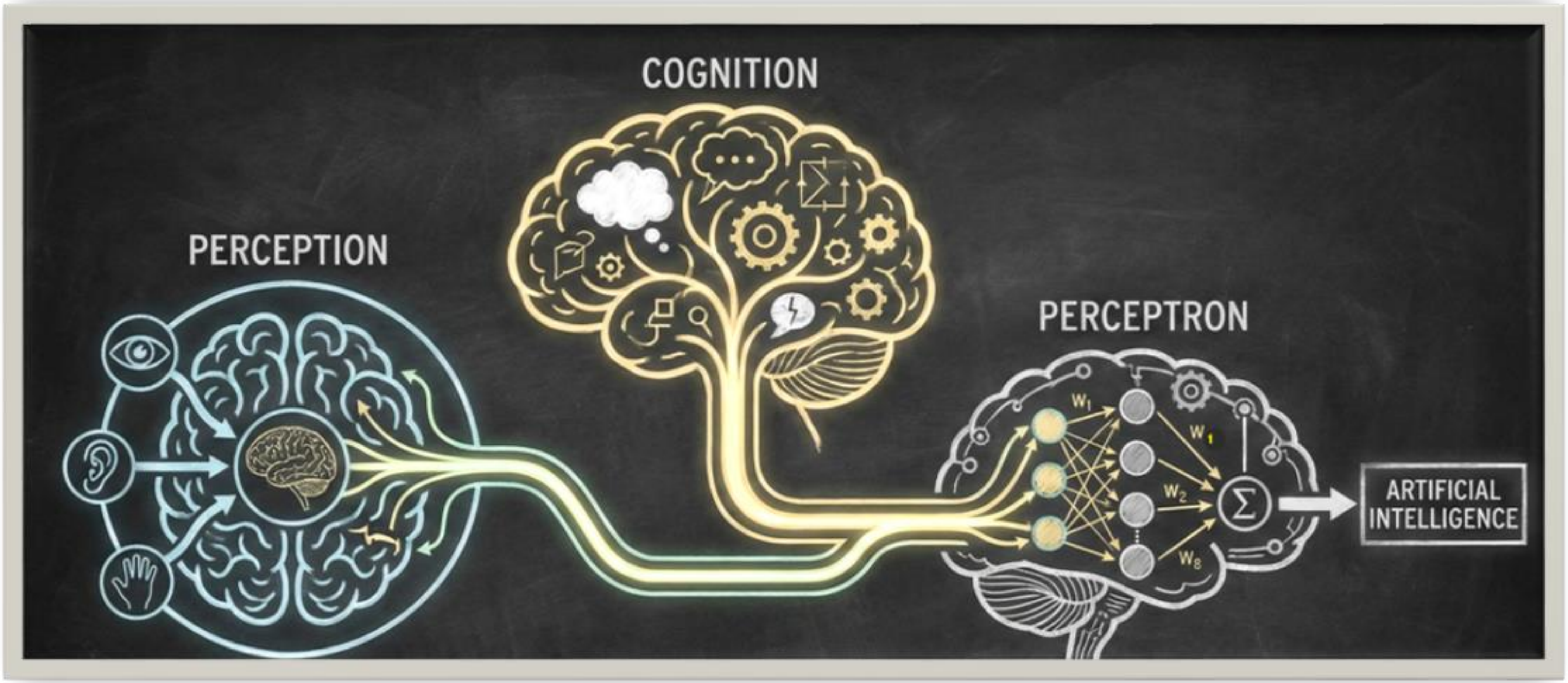
- **Rosenblatt** looked up to **McCulloch** and **Pitts**' contribution (Neurode) and extended their breakthrough into AI by extending a computational model into something new:

- ***"artificial neurons that reconfigure as they learn, embodying information in the strengths of their connections".***
- *In the summer of 1958, the editor of the Cornell Aeronautical Laboratory's Research Trends magazine devoted to Rosenblatt.*
- ***The article was titled "The Design of an Intelligent Automation: Introducing the Perceptron – a Machine that Senses, Recognises, Remembers, and Responds like the Human Mind".***

Rosenblatt and Perceptron

- *Rosenblatt chose Perceptron to describe his work, although it became one of his great regrets, as the word sounded way too much like a machine.*
- *By "**perceptron**" Rosenblatt meant a class of models of the nervous system*

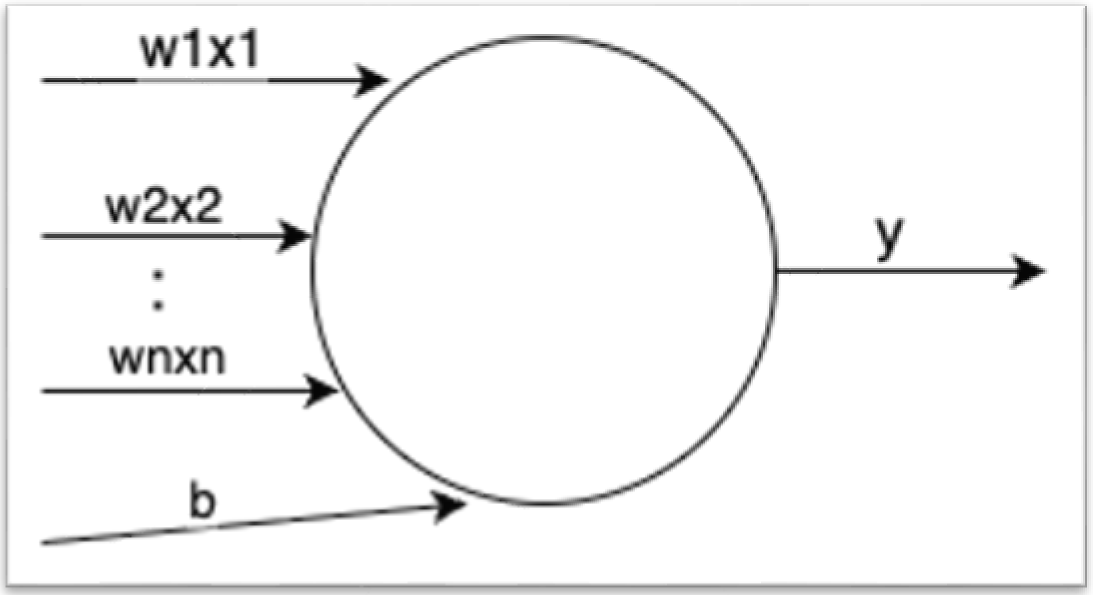
Perceptron


COGNITION
PERCEPTION
PERCEPTRON
w_1
w_1
w_2
w_8
Σ
ARTIFICIAL INTELLIGENCE

Perceptron (1958)

- Rosenblatt was a psychologist and did not have hardware computational resource to test his theory
- Therefore, he borrowed time on an **IBM 704** from the Cornell Aeronautical Laboratory
- IBM 704 is a **5-ton, room size, behemoth** (in simple terms, a gigantic computer)
- Have look at the whole IBM 700 series of computers
  - https://www.ibm.com/history/700 (IBM (n.d.))
  - https://www.youtube.com/watch?v=DKaVvv15Heo (YouTube video, 6 June 2024)

Overall Perceptron Architecture

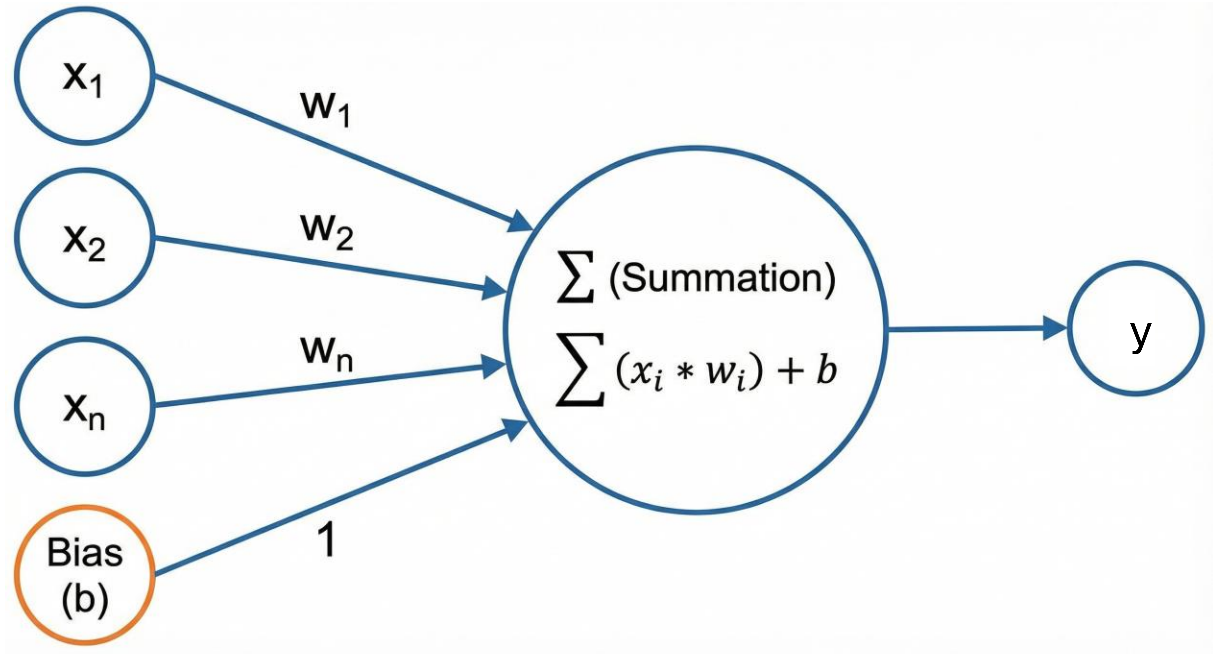


- Each **input (x_i)** is multiplied by its corresponding **weight (w_i)**. You may also notice an extra input **b**.

- The computation carried out by the perceptron goes like this:
- **sum = w1x1 + w2x2 + … + wnxn + b**
- **If sum > 0 y = 1**
- **Else y= -1**

Overall Perceptron Architecture

- The output is either **1** or **–1** (instead of 0 and 1 as for the McCulloch and Pitts' Neurode)
- **b** is an extra parameter accounting for **BIAS**
- Crucially, unlike with the Neurode, the Perceptron can learn the correct value for the **weights** and the **bias** for solving some problem.
- Hold on!
- How does it work?



Overall Perceptron Architecture and Learning process

- Imagine a simple **perceptron** that decides whether **someone** would **enjoy** a **pizza** based on two inputs: **spiciness** level (**x1**) and **cheese** amount (**x2**). We'll use a threshold of 0 and train the perceptron to predict pizza enjoyment.
- **Initial setup:**
- **Weights**: **w1** (**spiciness**) = 0.5, **w2** (**cheese**) = 0.5
- **Bias**: -1 (arbitrarily chosen)
- Threshold activation function:
  - If (w1*x1* + *w2*x2 + bias) > 0, output = 1 (will enjoy pizza)
  - If (w1*x1* + *w2*x2 + bias) ≤ 0, output = -1 (will not enjoy pizza)

Perceptron Learning Process

- **Training examples:**
- **Given the starting weight values (w1 = 0.5, and w2 = 0.5) and the arbitrarily assigned value for bias set to -1, let's work out some examples:**
- Training Example (a):
- Mild spicy (x1 = 2), lots of cheese (x2 = 3), target (+1, the training example is labelled as "enjoy the pizza") Let's use those values in the numerical expression
  y=w1x1 + w2x2 + b
- **y = 0.5 * 2 + 0.5 * 3 - 1 = 2.5 - 1 = 1.5**
- **1.5 > 0** → (Correct Prediction) → Actually enjoys pizza (output= +1)

Perceptron Learning Process

- Training Example (b)
- Very spicy (5), little cheese (1), Label "dislikes pizza" (target = -1)
- Using the given values with the threshold activation function:
- y = 0.5 * 5 + 0.5 * 1 – 1 = 2.5 + 0.5 - 1 = 2
- The predicted value is 2 > 0 meaning that the output is (+1) "likes pizza". There is a mismatch between the training target (-1) and the predicted value (+1).
- The **learning** algorithm will **adjust weights** based on **prediction errors**, **gradually** finding the right **balance** to correctly classify pizza preferences.
- Each **misclassification triggers** a **weight update** that moves the decision boundary closer to perfectly separating "**enjoy**" and "**don't enjoy**" pizza scenarios.

Rolling down training process

Let's recap from Training Example (b).

Training Example (b):

- Inputs: x1 = 5 (very spicy), x2 = 1 (little cheese)
- Target = -1 (dislikes pizza)
- Step 1: Compute weighted sum y = w1x1 + w2x2 + b y = (0.5 * 5) + (0.5 * 1) - 1 = 2.5 + 0.5 - 1 = 2

Step 2: Activation

- Since y = 2 > 0, the current weights **predict** +1 (**enjoys pizza**)
- But the **actual target** is -1 (**dislikes pizza**)
- **This means we have a misclassification**

Rolling down training process

Step 3: Weight Update

- **Learning rate (let's use α = 0.1)**
- **Update rule: Δwi = α * (target - prediction) * xi**
- **Δw1** = 0.1 * (-1 - (+1)) * 5 = 0.1 * (-2) * 5 = **-1**
- **Δw2** = 0.1 * (-1 - (+1)) * 1 = 0.1 * (-2) * 1 = **-0.2**
- **Δbias** = 0.1 * (-1 - (+1)) = **-0.2**

New weights after update:

- w1 = 0.5 - 1 = -0.5
- w2 = 0.5 - 0.2 = 0.3
- bias = -1 - 0.2 = -1.2

The training process keeps going on until some requirements are met (see the next slide).

When does the training process stop?

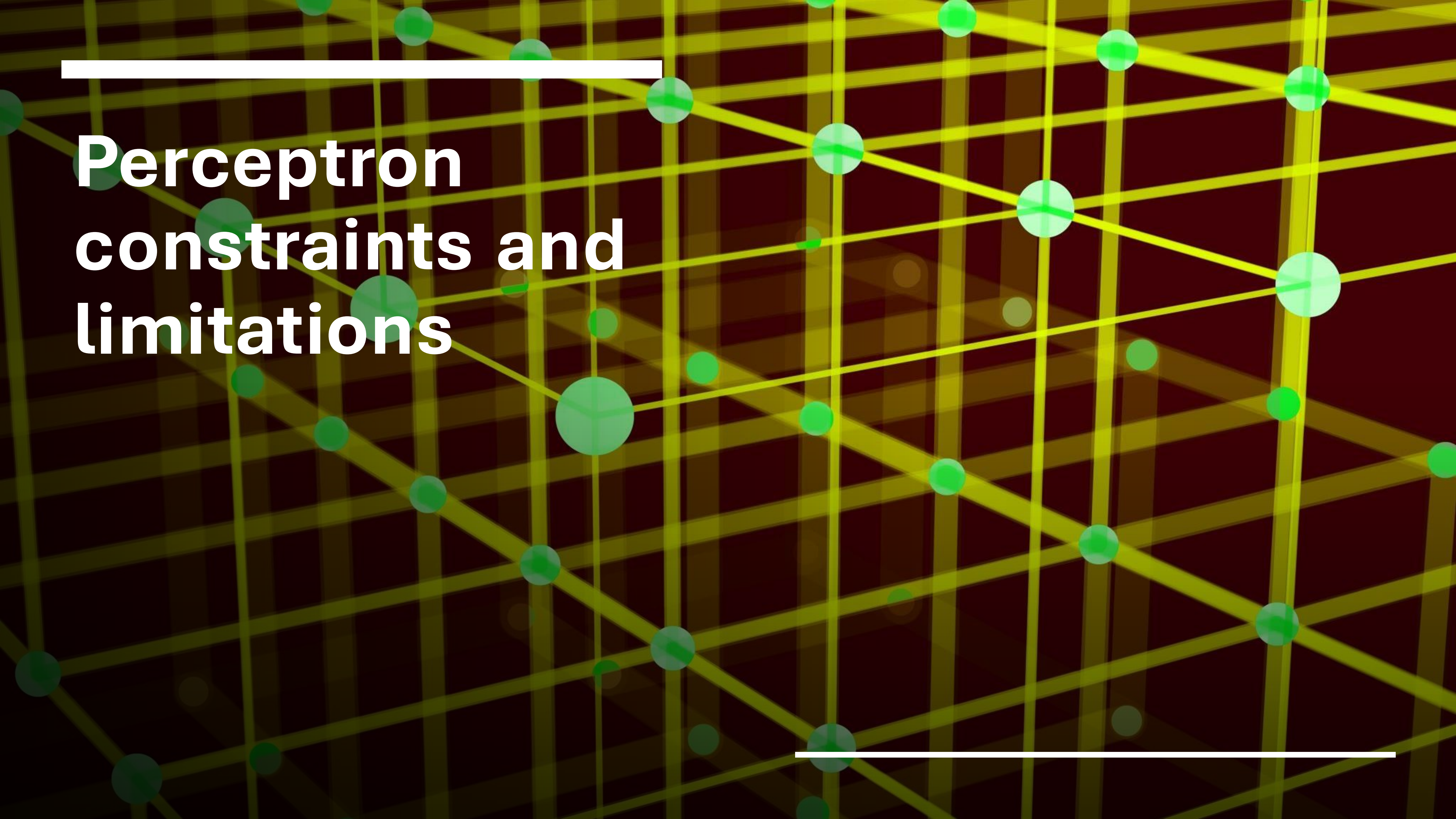
The Perceptron training typically stops under these conditions:

Convergence Criteria:

- All training examples are correctly classified
- No weight updates are needed in a complete pass through the training data

Practical Stopping Mechanisms:

- Maximum iteration limit
- No weight changes beyond a small threshold in an entire epoch
- No misclassifications in the last complete pass



Perceptron constraints and limitations

Linearly Separable Data

Linearly separable data refers to a **set of data points** that can be perfectly divided into two distinct classes by drawing a single **straight line (in 2D)** or a **hyperplane (in higher dimensions)**.

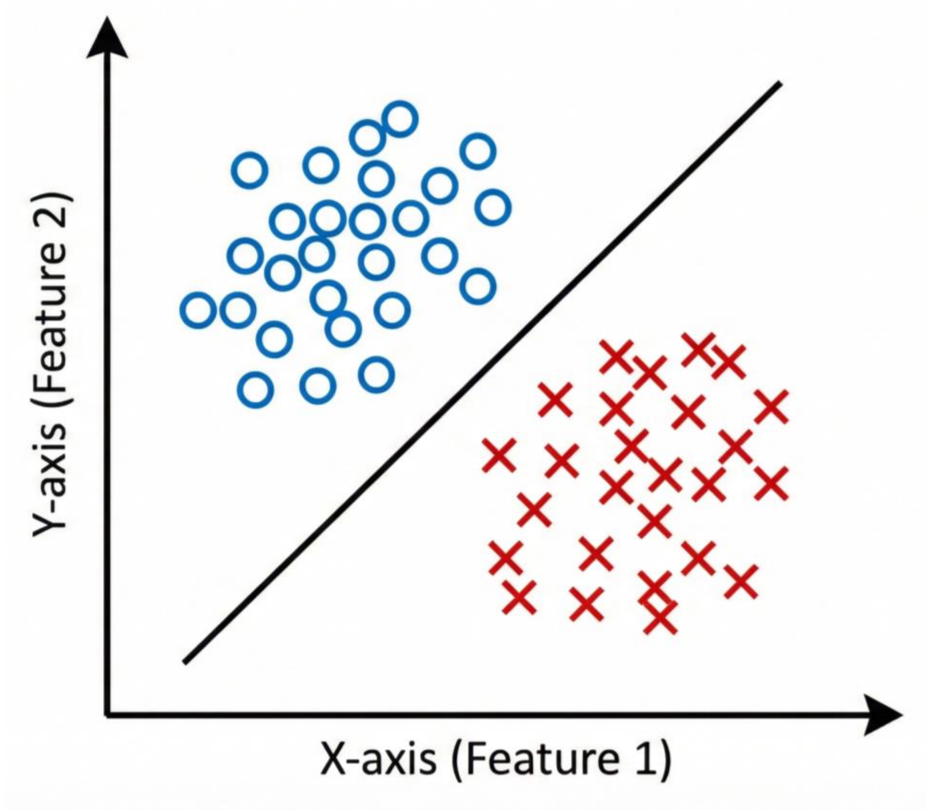
In the context of the **Perceptron algorithm**, linear separability is a crucial concept because it determines the algorithm's ability to find a perfect decision boundary.

When data is linearly separable, the **Perceptron learning algorithm is guaranteed to converge**, meaning it will find a set of weights that can correctly classify all training examples after a finite number of iterations.

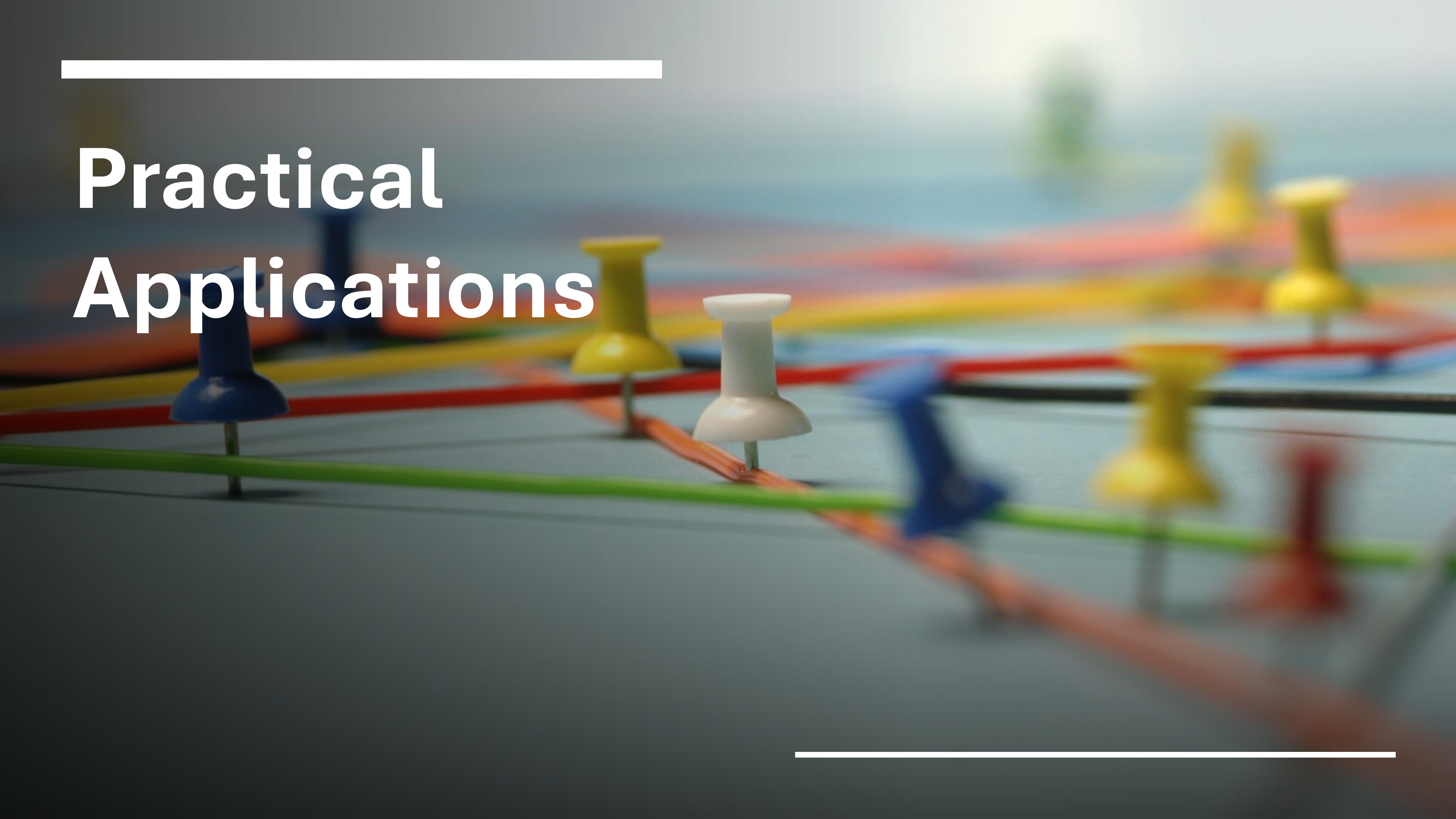
Conversely, if the data is **not linearly separable**, the **Perceptron will not converge** and will continue to make classification errors indefinitely.

Linearly Separable Data

- Figure out the following scenario:
  - Some data points are represented upon two features (A and B).
  - That might be, for instance, the representation of customers on geographic location (A) and expenditure (B) feature
  - If "customers" data points can be divided by a single straight line (as in the diagram on the right side), that means they are linearly separable.



Practical Applications



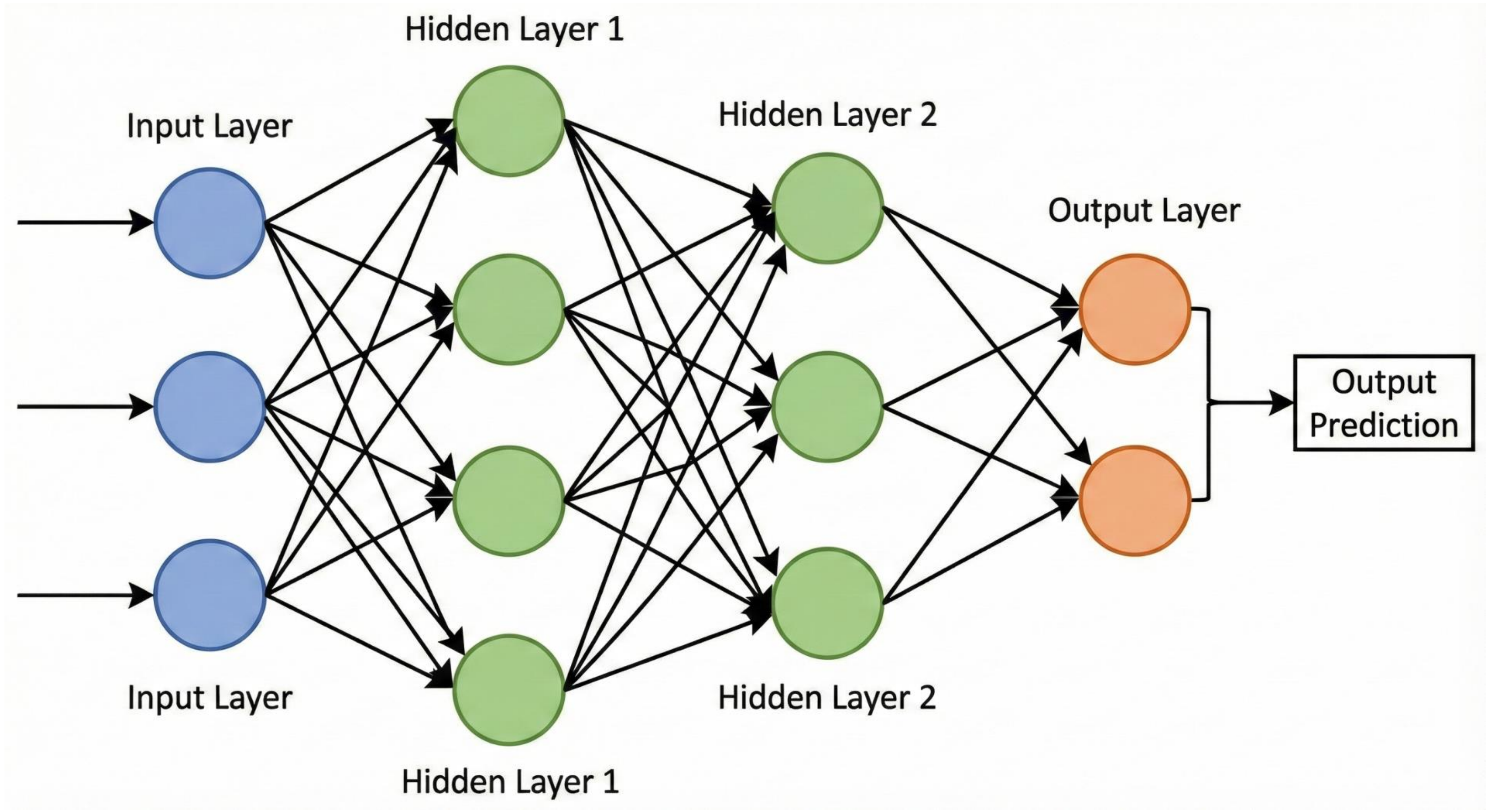
A practical perceptron application (spam, non-spam classification)

Check out the Python project at the link below:

https://github.com/alessandrobruno10/perceptron

Set up a new Python virtual environment on your laptop and run it to assess performances.

MultiLayer Perceptron



MultiLayer Perceptron (MLPerceptron)

- Multilayer Perceptron (MLP) represents a neural architecture that relies upon the primary concepts as in Perceptron.
- Main differences:
  - **Multiple layers: input layer, hidden layer(s), output layer**
  - Introduces **non-linear activation functions** (e.g., **sigmoid**, **ReLU**)
  - Can create complex, non-linear decision boundaries
  - Learns **hierarchical feature representations**

MLP can model intricate relationships between inputs and outputs. They rely upon hidden layers to learn higher level features (starting from raw data, it can learn how to represent highly representative features from data).

Learning Mechanism: Uses **backpropagation** to update **weights across all layers (getting all the way back from output to intermediate layers).**

Hidden Layers

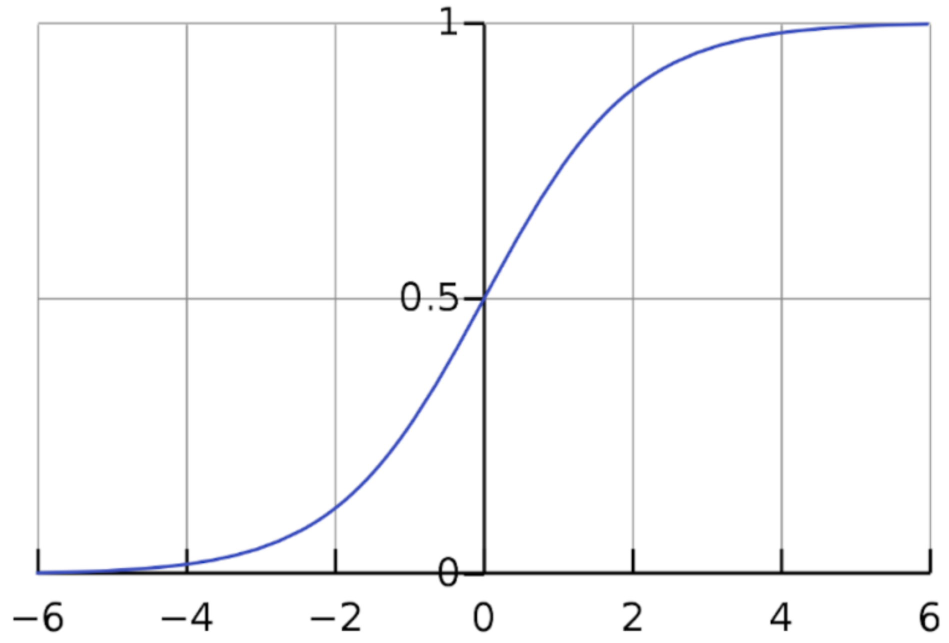
- Mathematically, hidden layers transform the input space into a more separable feature space where complex classifications become possible.
- Let's address **Hidden Layers** by tackling **Functional Purposes**, **Computational Mechanisms**, and **Key Characteristics**.
- **Functional Purposes:**
    - They represent Intermediate layers **between input and output** layers
    - They **transform input features** through **weighted linear combinations**
    - They apply **non-linear activation functions** to introduce **complexity** (so that even non-linearly separable data can be handled)

Hidden Layers

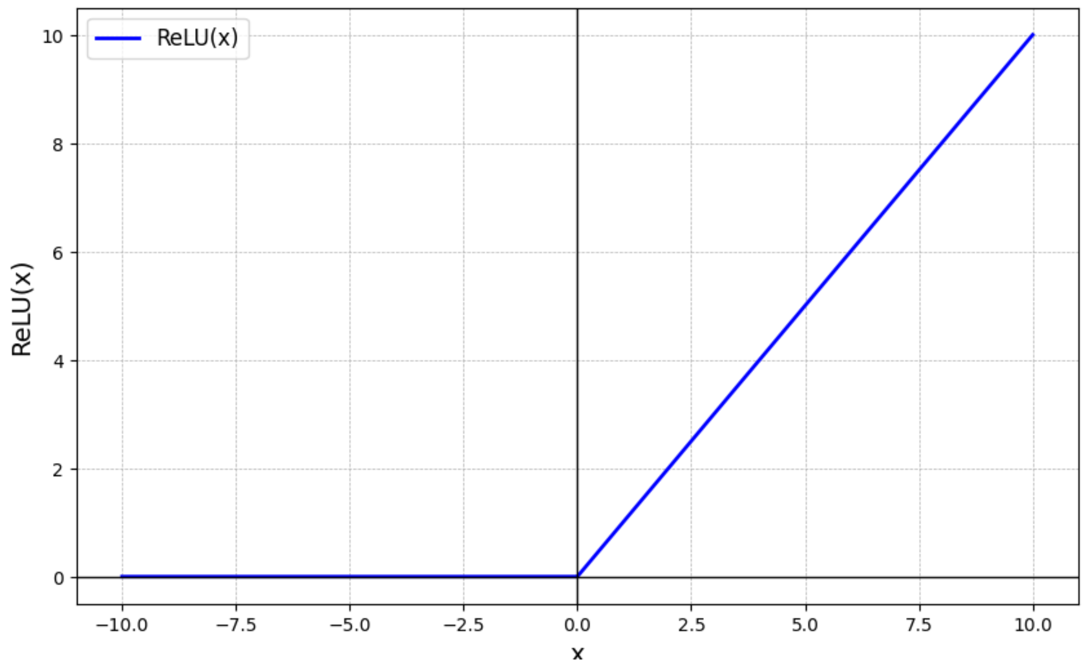
- **Computational Mechanisms:**
  - They receive weighted inputs from previous layer
  - They apply activation function (e.g., ReLU – Rectified Linear Unit, Sigmoid)
  - They generate **transformed features** passed to subsequent layers (as you go along with layers, you find higher level features).
- **Key Characteristics:**
  - **Not directly observable** from outside the network (**black box**)
  - **Number of neurons** determines representational capacity
  - Weights in hidden layer learn intermediate feature representations
  - Non-linear activation functions enable modelling of complex, non-linear relationships

Sigmoid Function & ReLU

Sigmoid



ReLU



Both introduce **nonlinearities** to the architecture allowing it to handle with non-linearly separable data

Why do Perceptron and MLPerceptron matter today?

- Perceptron allows us to learn some of the most meaningful concepts in AI:
  - Training Process
  - Weighted sum of inputs + Bias
  - Learning rate
  - Prediction Error (difference between classification and prediction outputs)
  - Linearly Separable Data
- MultiLayer Perceptron overcomes Perceptron due to nonlinearities it adds to the learning process with activation functions and hidden layers.
  - It is currently broadly adopted due its relatively low computational cost, compared with deep learning methods
  - It can infer knowledge from plenty scenarios as in reality data are rich of nonlinearities.