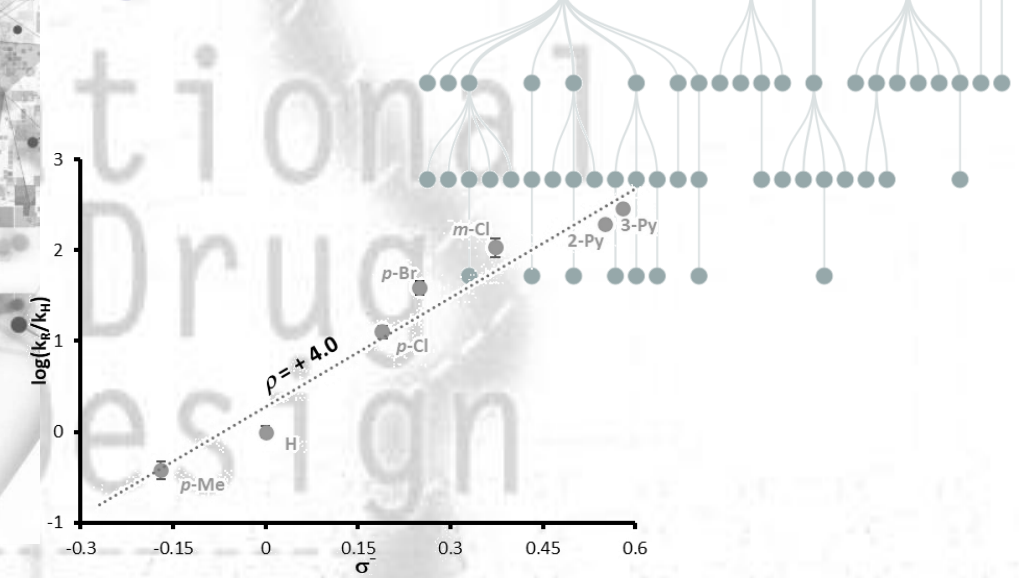
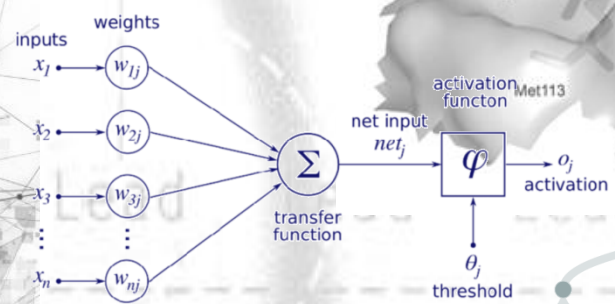
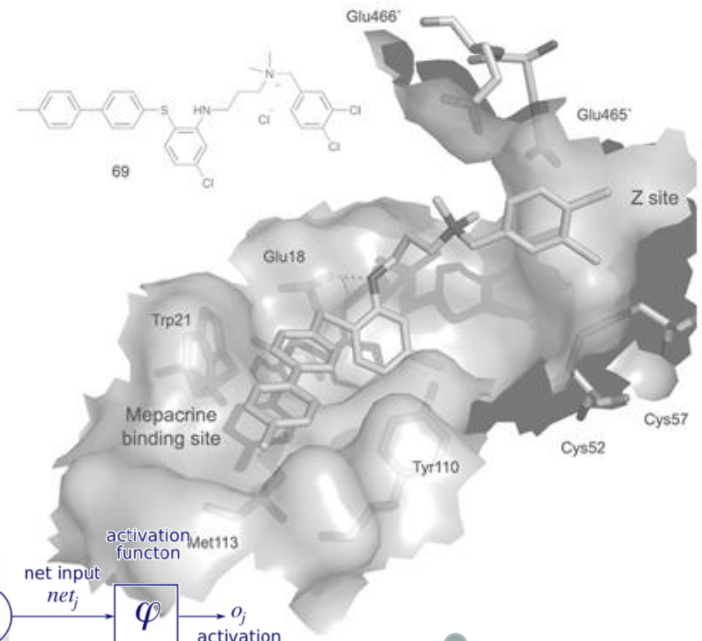


Introduzione ai metodi di Intelligenza Artificiale...



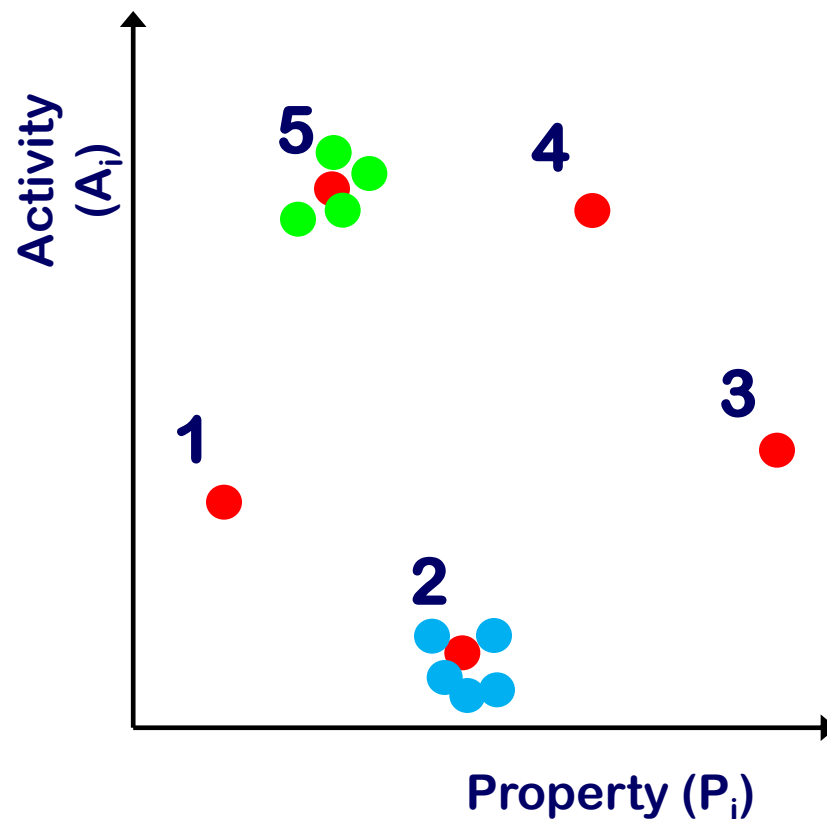
Stefano Ricci

Artificial Neural Networks (ANN):





(Q)SAR: follow me in this wonderful experience





We can start from here...



What is it?

It is easy to understand there is not a mathematical function that can answer to this question...



We can approach this problem in this way..



EXPERIENCE



We can approach this problem in this way..

MULTIPLE EXPERIENCE



SUPERVISOR

**CATEGORY
(APPLE)**



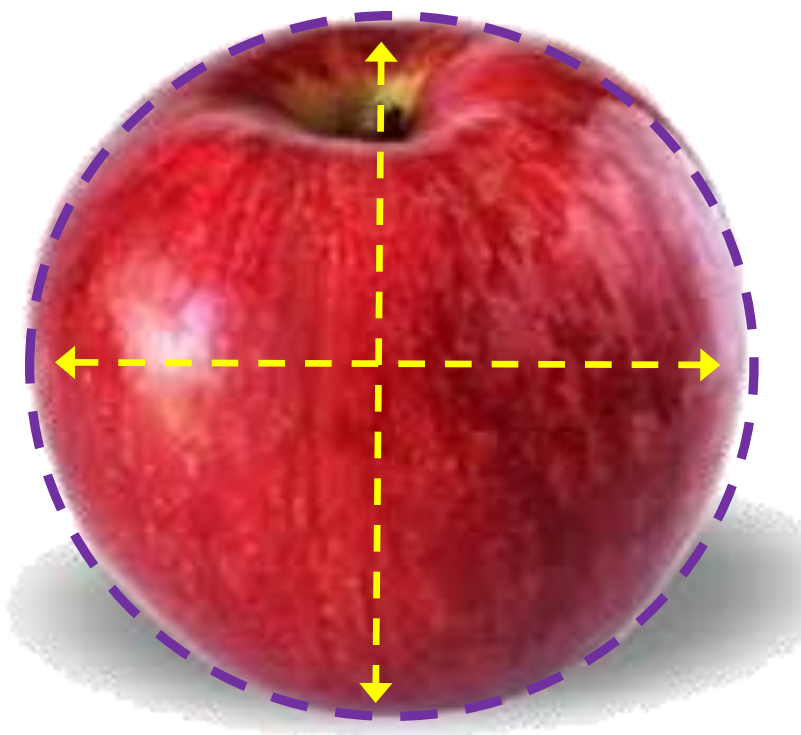
Why we can identify a category?



**CATEGORY
(APPLE)**



Back to the first example...

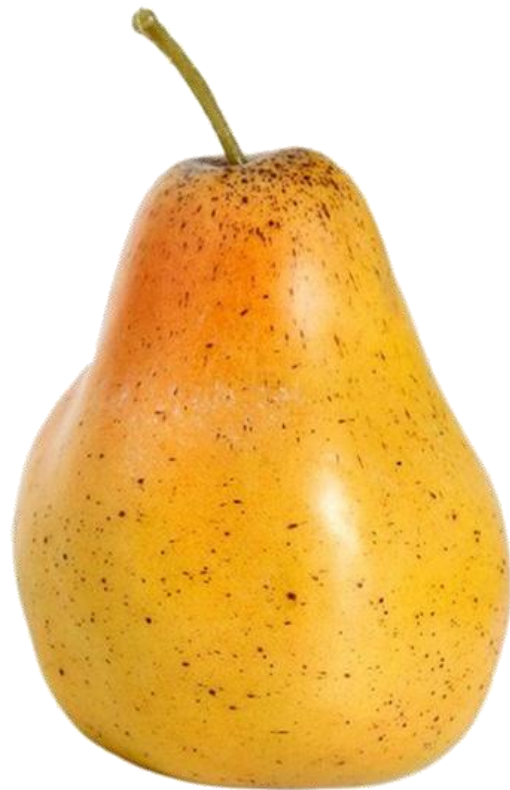


Geometrical Properties
Color

...



And now it is very easy to answer
this question: *is this an apple?*





Again..

MULTIPLE EXPERIENCE



SUPERVISOR



**CATEGORY
(PEAR)**

And this...





A bit of nomenclature:

ARTIFICIAL INTELLIGENCE

The ability of a computer program or a machine to think like humans do.

MACHINE LEARNING

Subfield of AI giving machines the skills to learn from examples without being explicitly programmed.

Examples: Fraud detection, marketing personalization, email classification



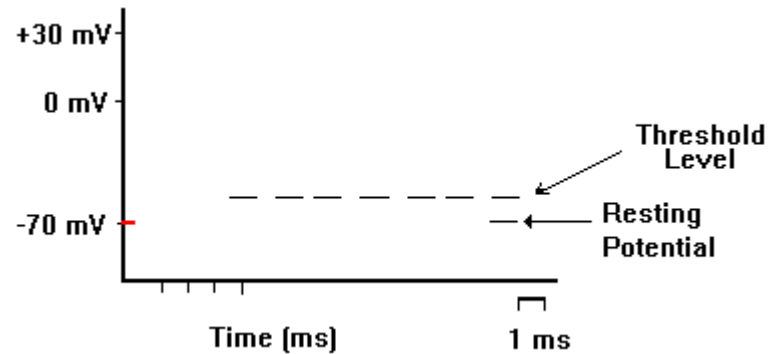
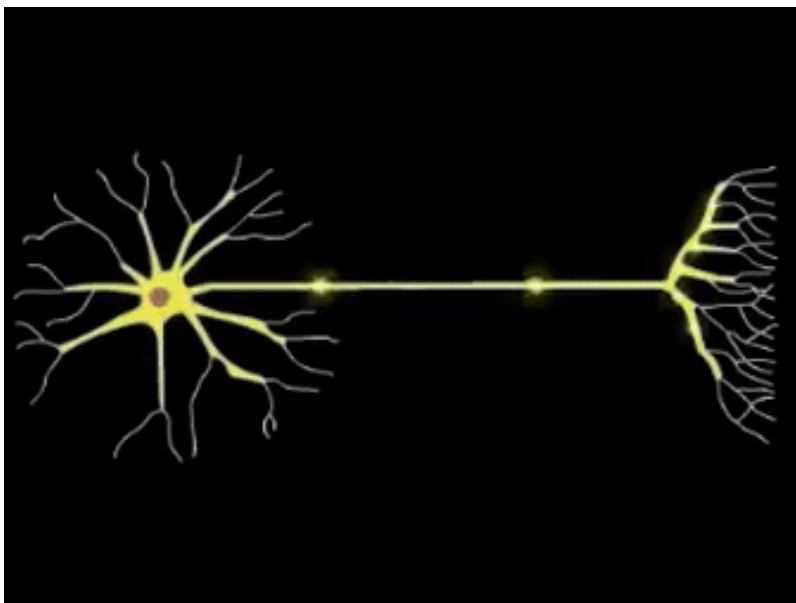
DEEP LEARNING

Specialized machine learning technique enabling machines to train themselves to perform tasks.

Examples: Image classification, vehicle detection, sentiment analysis



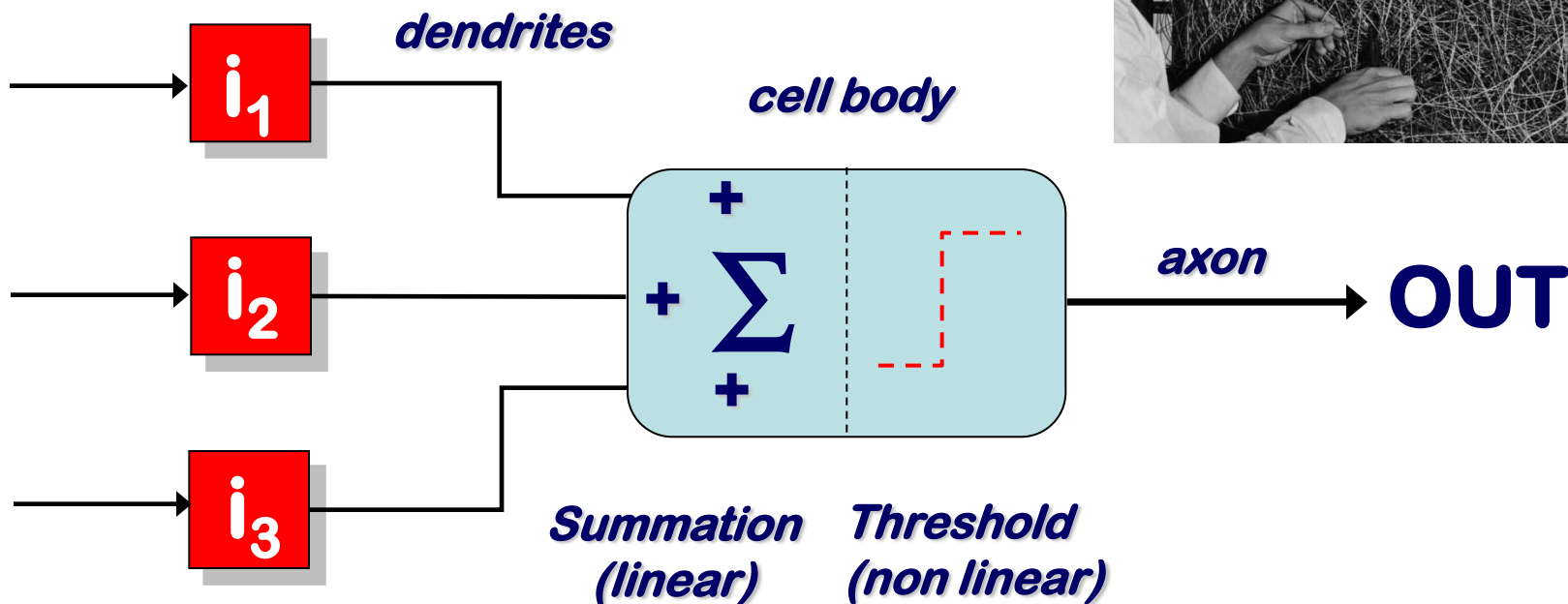
Do you remember the structure of neurons?



From human neurons to artificial neurons...



PERCEPTRON



The perceptron-based is the oldest neural network, created by **Frank Rosenblatt in 1958.**

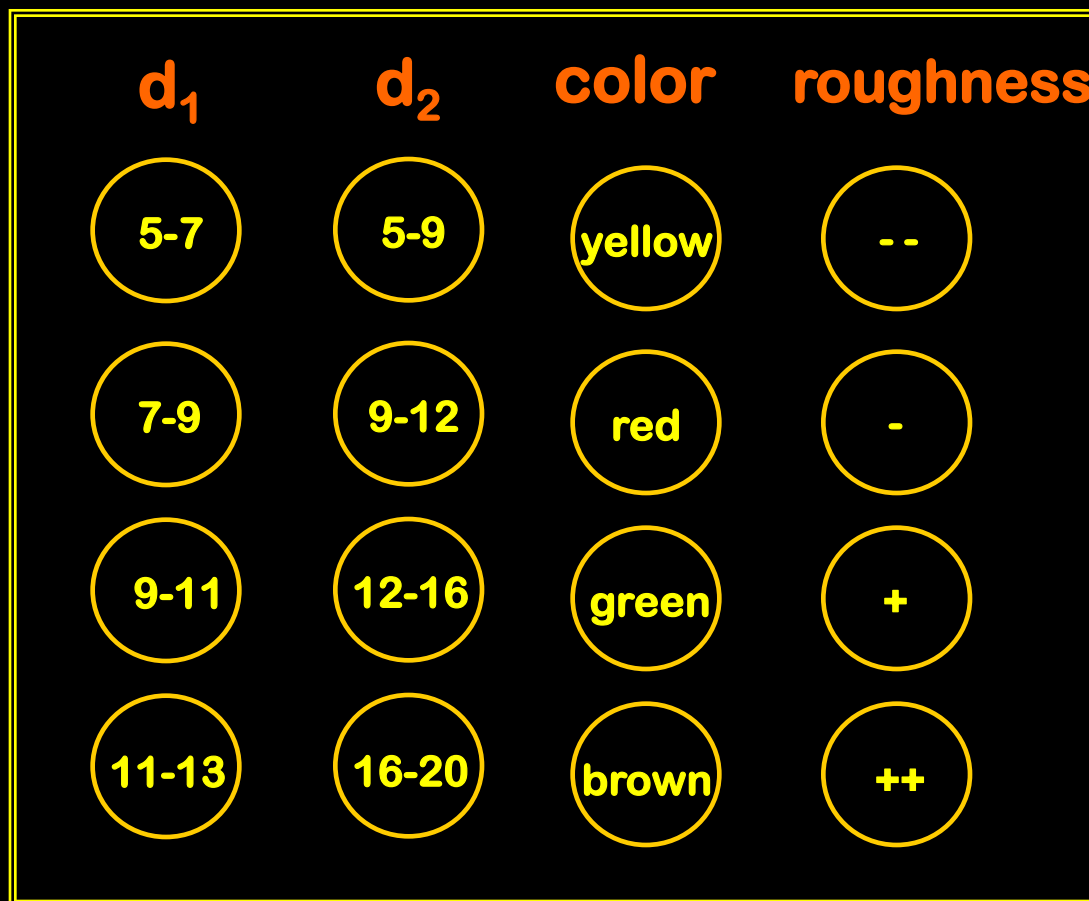
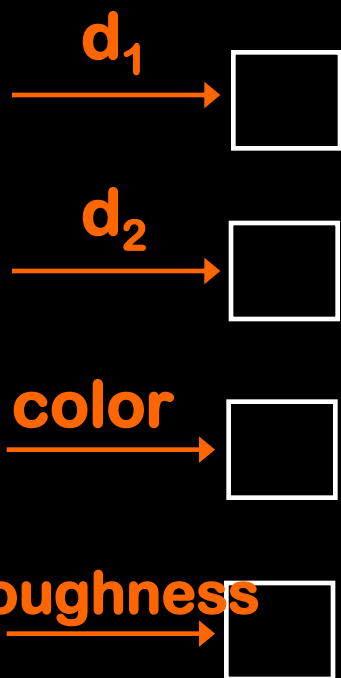
Artificial Neural Networks (ANN):

Logical structure of an ANN

INPUT

NEURAL HIDDEN

OUTPUT



Artificial Neural Networks (ANN):

Phase 1 – Learning.

INPUT

NEURAL HIDDEN

OUTPUT



d_1

5

d_2

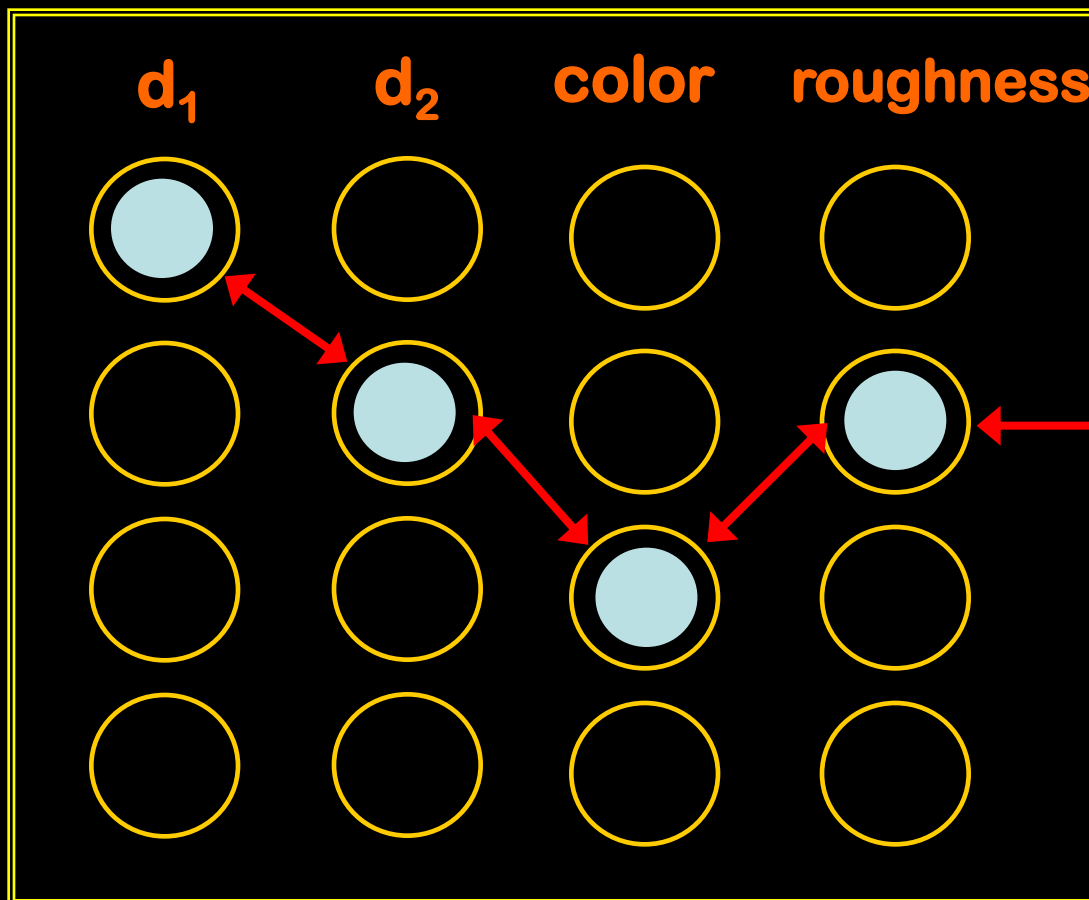
6

color

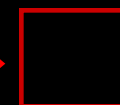


roughness

-



Apple



Artificial Neural Networks (ANN):

Phase 1 – Learning.

INPUT

NEURAL HIDDEN

OUTPUT



d_1

5

d_2

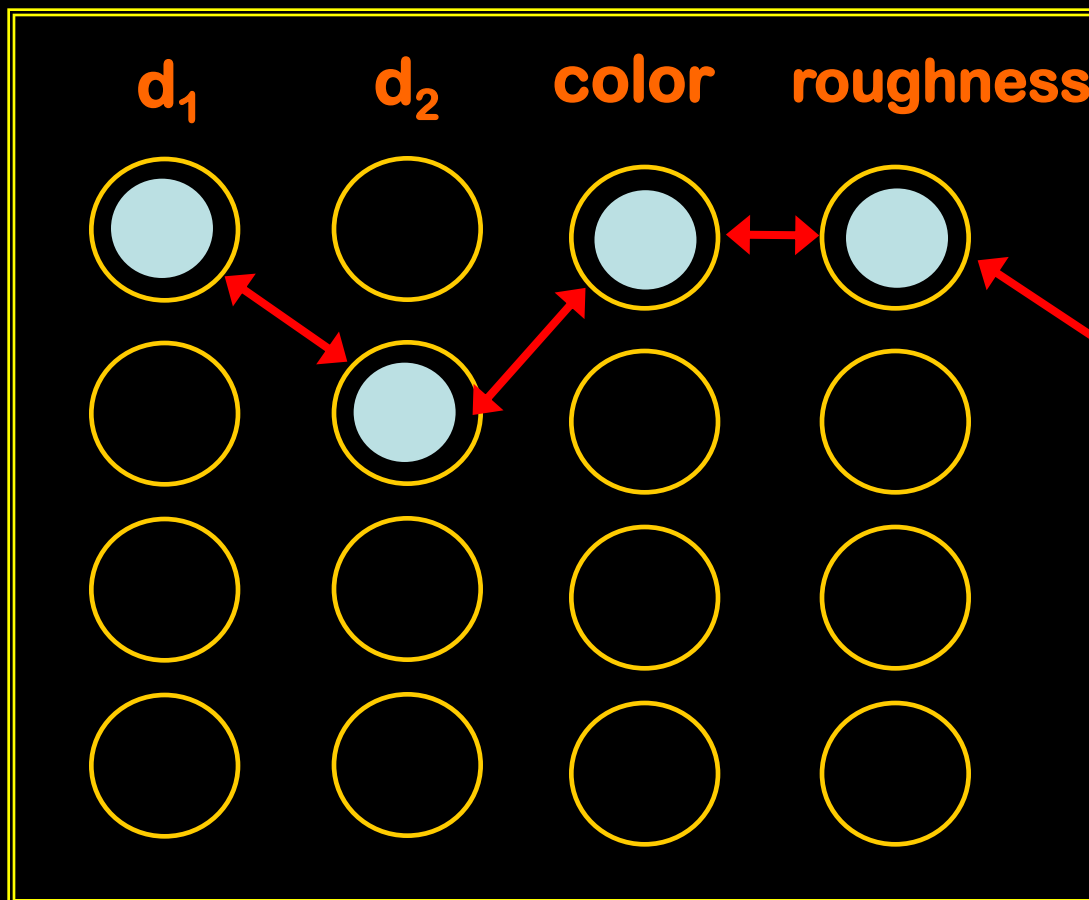
7

color

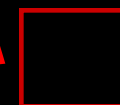


roughness

--



Apple



Artificial Neural Networks (ANN):

Phase 1 – Learning.

INPUT

NEURAL HIDDEN

OUTPUT



d_1

9

d_2

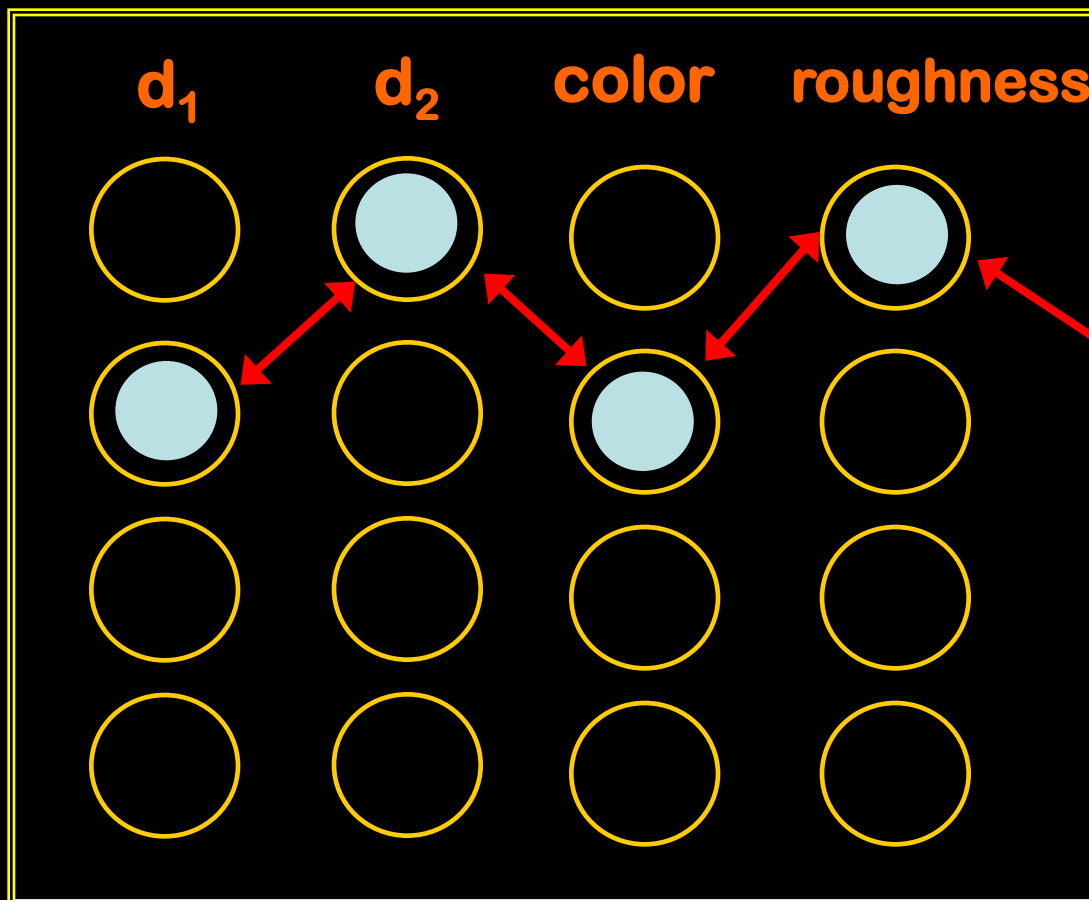
10

color



roughness

--



Apple



Artificial Neural Networks (ANN):

Phase 1 – Learning.

INPUT

NEURAL HIDDEN

OUTPUT



d_1

9

d_2

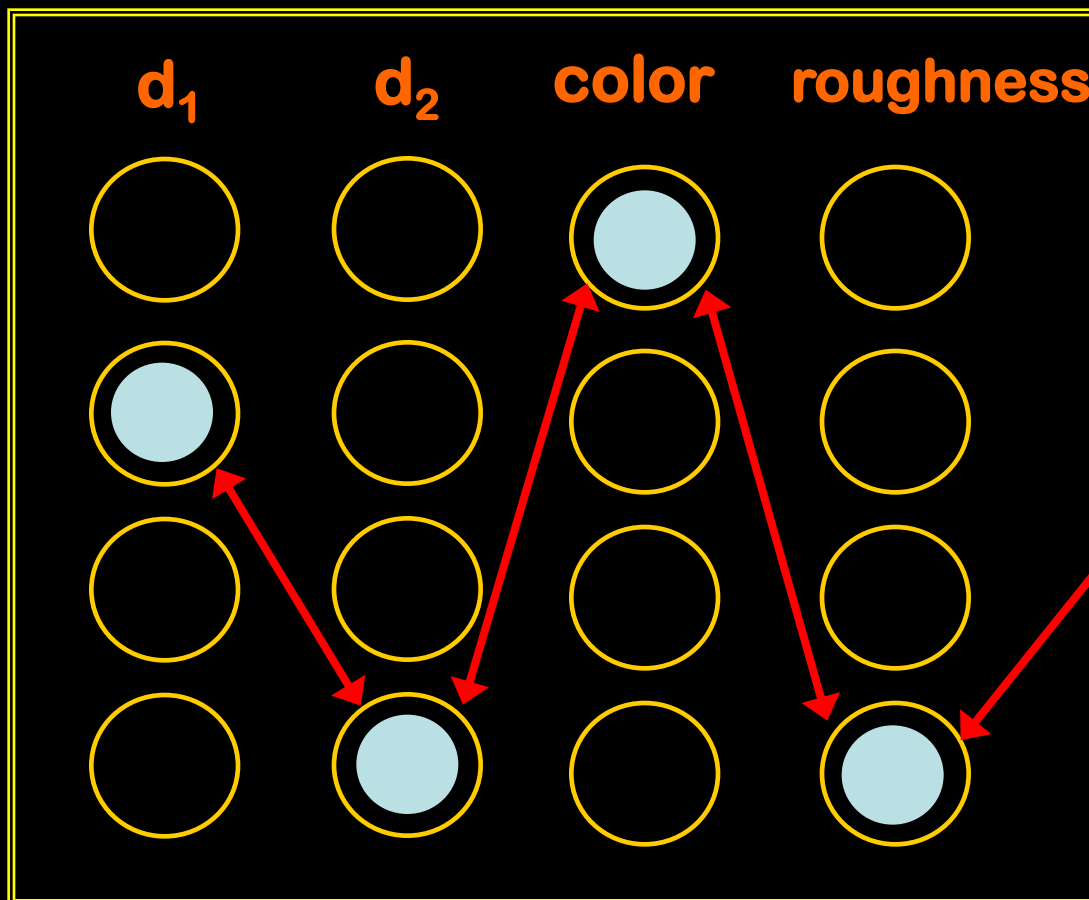
10

color



roughness

++



Not an
apple

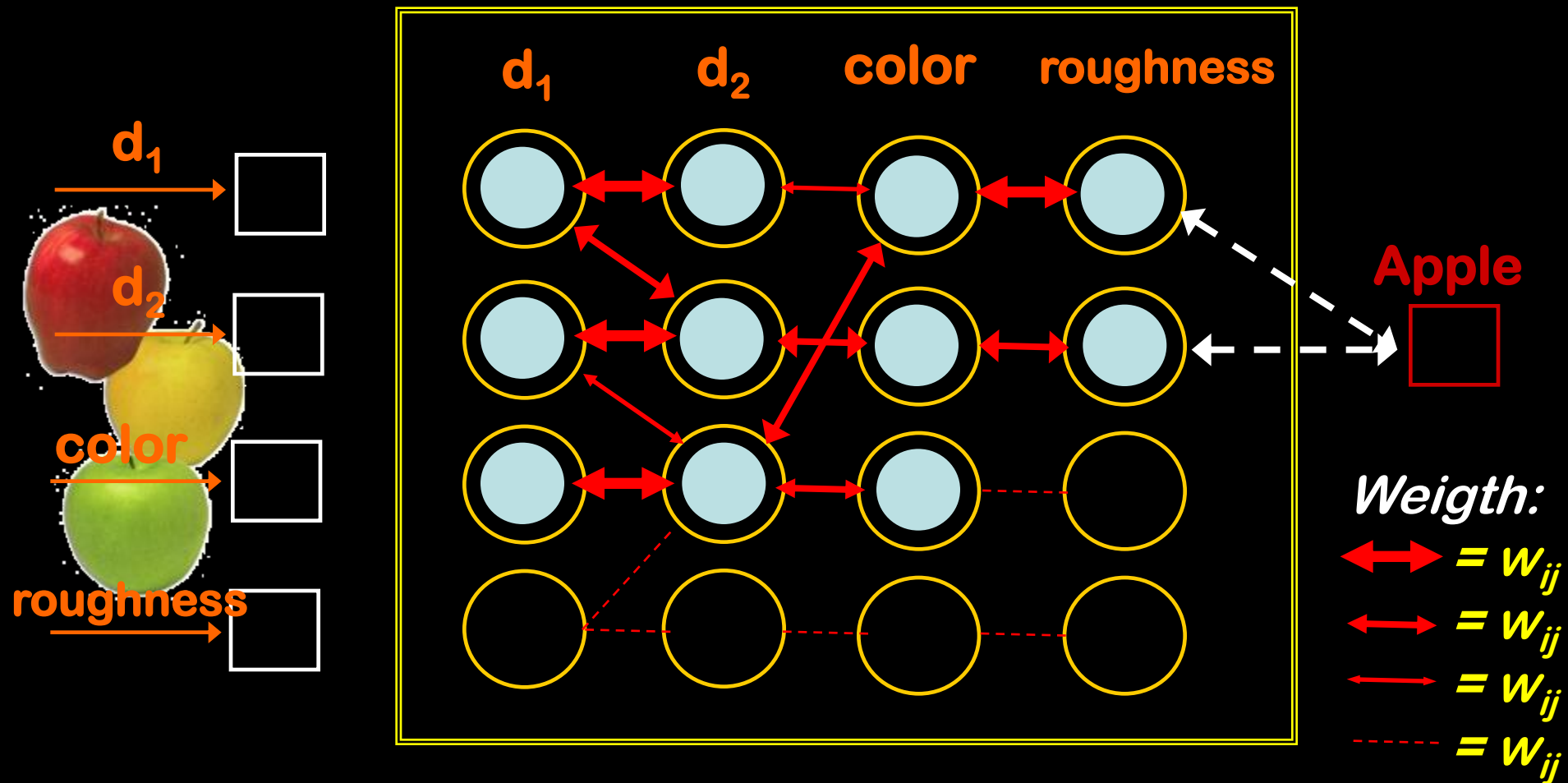
Artificial Neural Networks (ANN):

Phase 1 – Learning.

INPUT

NEURAL HIDDEN

OUTPUT



Artificial Neural Networks (ANN):

Phase 2 – Recognition.

INPUT



d_1

5

d_2

6

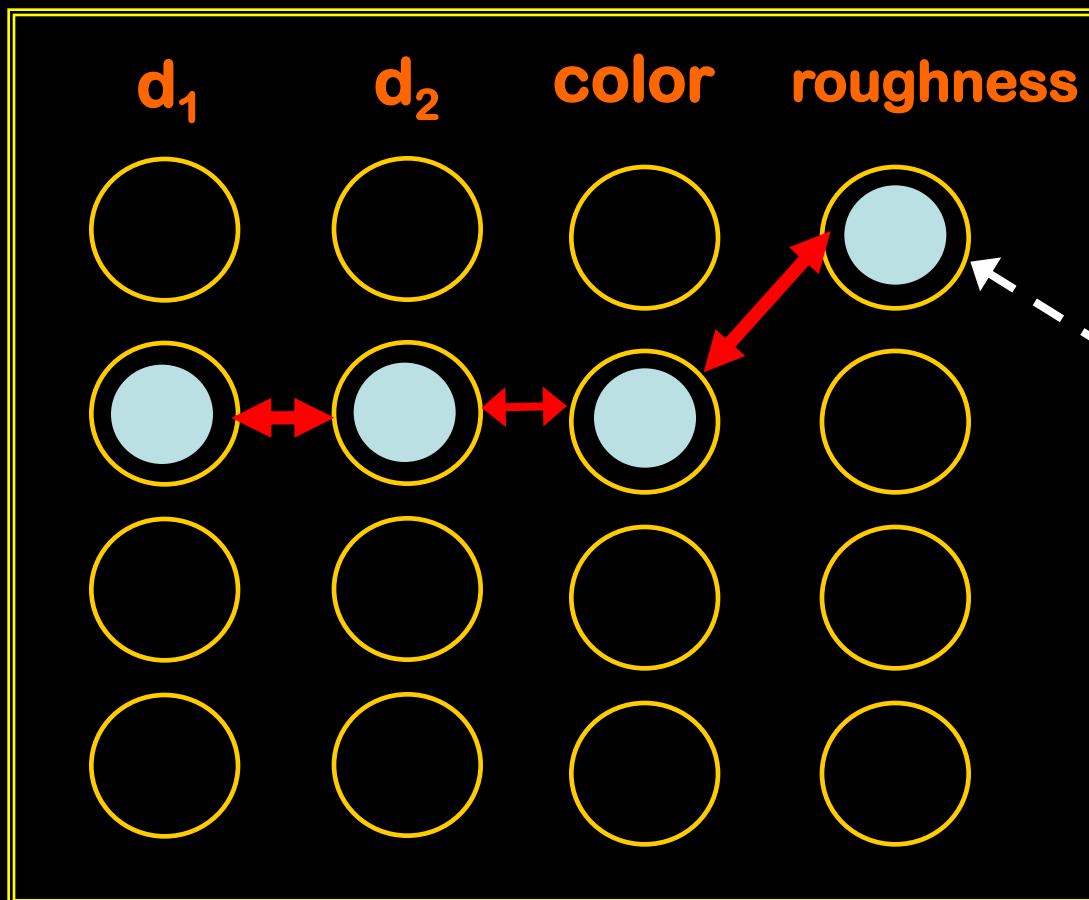
color



roughness

--

NEURAL HIDDEN



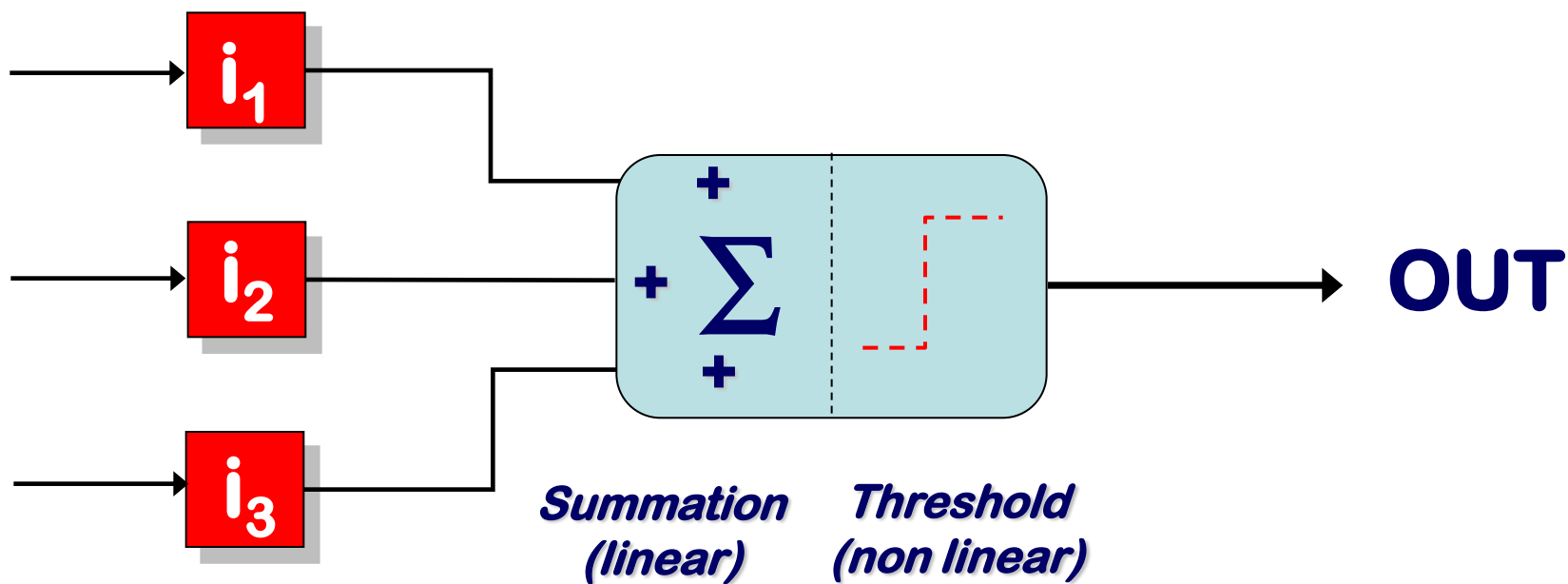
OUTPUT

Apple





Back to our *perceptron*...





Introduction to the simple ANN - the *linear perceptron* :

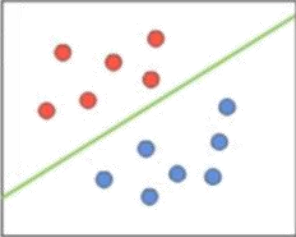
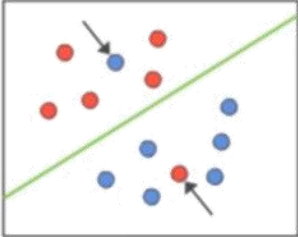
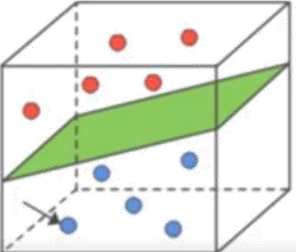
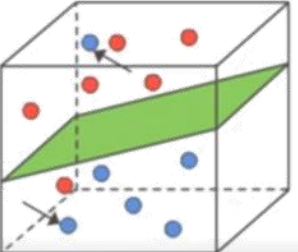
The Linear Perceptron algorithm is one of the earliest algorithms developed in the field of machine learning. It is a simple linear classifier used for binary classification tasks.

The goal of the Linear Perceptron is to adjust its weights through an iterative process, in order to correctly classify different samples into two distinct classes.



Introduction to the simple ANN - the *linear perceptron* :

Before starting: understanding whether a space is *linearly separable* means establishing whether there exists a straight line (in 2D), a plane (in 3D) or more generally a hyperplane that can separate two sets of points without overlapping. As an example:

| | Linear separable | Linear non-separable |
|----|---|---|
| 2D |  |  |
| 3D |  |  |



Introduction to the simple ANN - the *linear perceptron* :

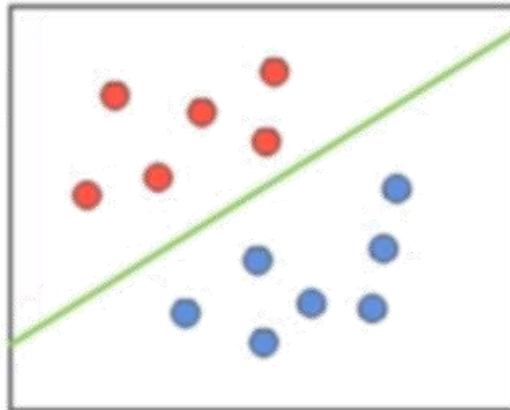
Formal definition: Two sets of points (e.g. classes 0 and 1) are linearly separable if there exists a vector w and a bias b such that:

for all points x of class 1, $w \cdot x + b > 0$

for all points x of class 0, $w \cdot x + b < 0$

2D

x_2



x_1

Remember this equation:

$$y = mx + q$$

It is equivalent to:

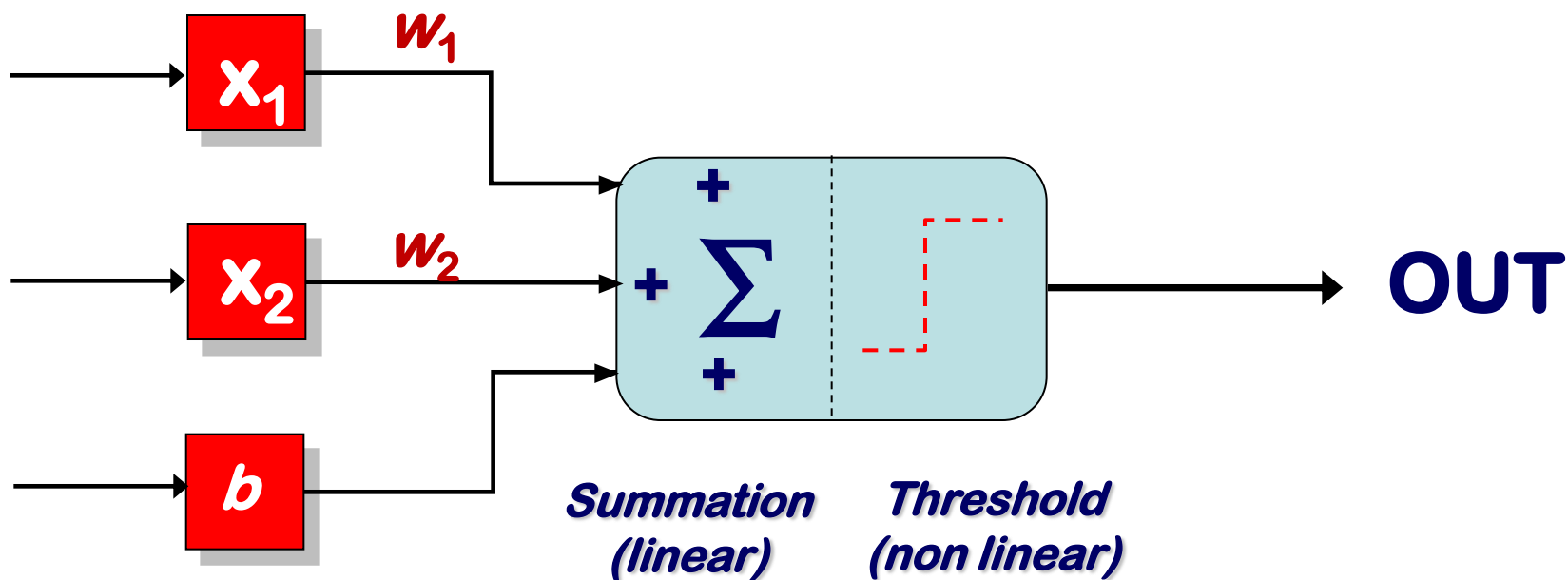
$$w_2 x_2 = w_1 x_1 + b^*$$

* when b is equal to zero all the lines must have the intercept at the origin of the axes (0,0), when b is different from zero we can explore all the lines that can best separate the points in the 2D space!



Introduction to the simple ANN - the *linear perceptron*:

The linear perceptron: w_1 , w_2 and b are defined *parameters* of the perceptron.



$$w_1x_1 + w_2x_2 + b$$

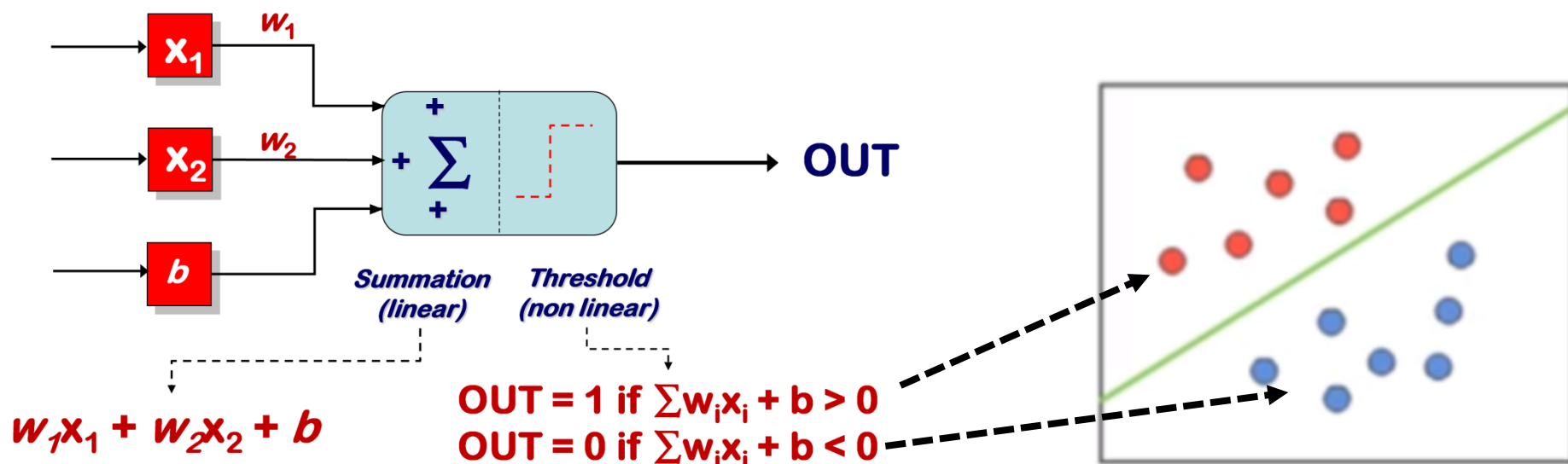
$$\text{OUT} = 1 \text{ if } \sum w_i x_i + b > 0$$

$$\text{OUT} = 0 \text{ if } \sum w_i x_i + b < 0$$



Introduction to the simple ANN - the *linear perceptron*:

Finally:



Our problem now is, given a set of data, to verify whether this space is linearly separable, *i.e.* to find the parameters of the perceptron (in this case w_1 , w_2 and b) that accurately classifies the data set.



A simple example of the application of a *linear perceptron* in medchem:

A concrete and simple example of a linear perceptron applied in a context inspired by medicinal chemistry, for example to classify a molecule as *active* or *inactive* on a certain biological target, using a few molecular descriptors.

Imagine having a dataset with two molecular descriptors for each molecule:

$$\begin{aligned}x_1 &= \log P \text{ (lipophilicity)} \\x_2 &= \text{MW (molecular weight)}\end{aligned}$$

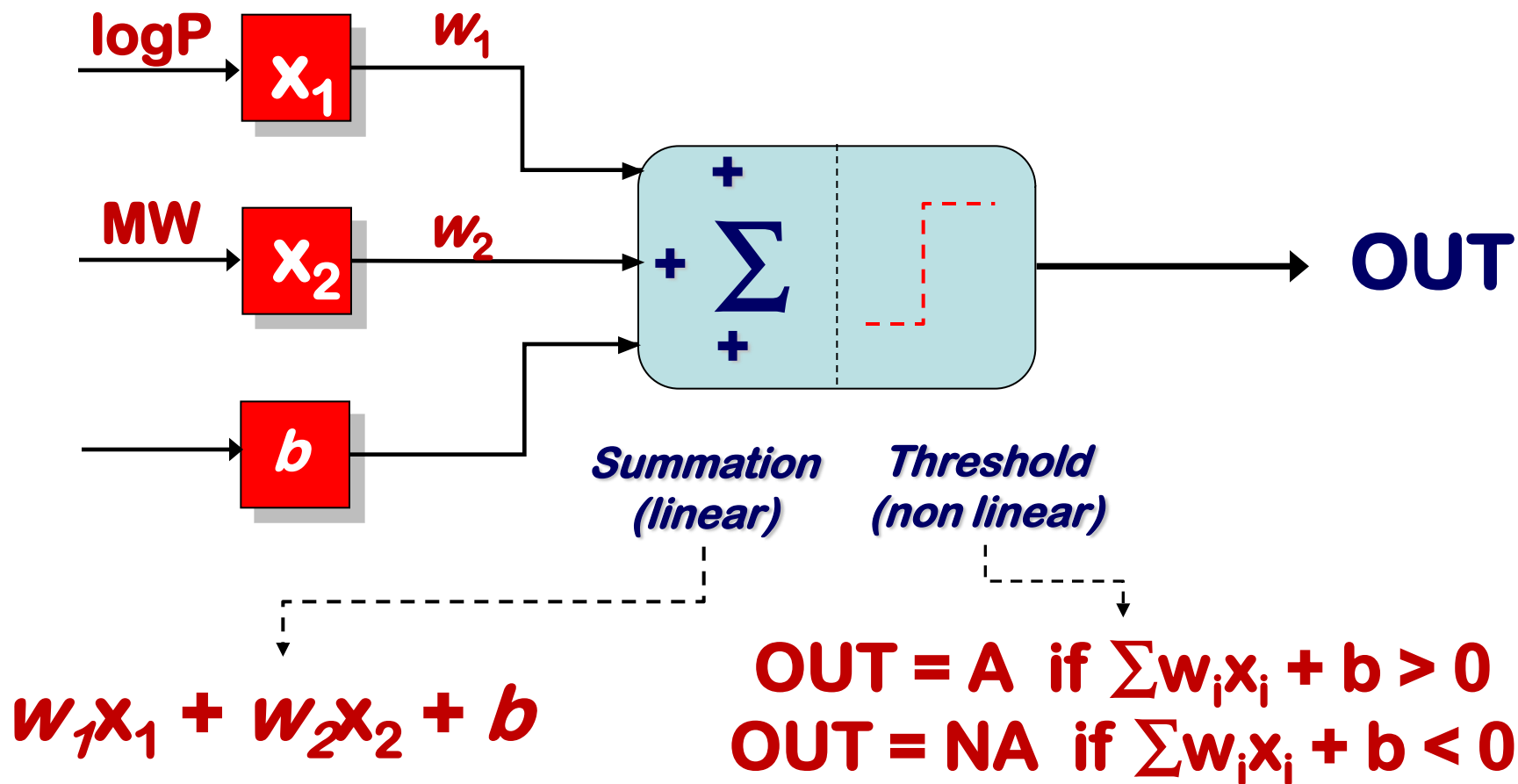
And an associated label:

$$\begin{aligned}\text{OUT} &= 1 \text{ if the molecule is } \textit{active} \\ \text{OUT} &= 0 \text{ if it is } \textit{inactive}\end{aligned}$$



Introduction to the simple ANN - the *linear perceptron*:

Here is our simple linear perceptron:





A simple example of the application of a *linear perceptron* in medchem:

The model of our linear perceptron has the form:

$$\text{OUT} = \text{step} (w_1x_1 + w_2x_2 + b)$$

where:

w_1 and w_2 are the weights and b is the bias (threshold) to learn;

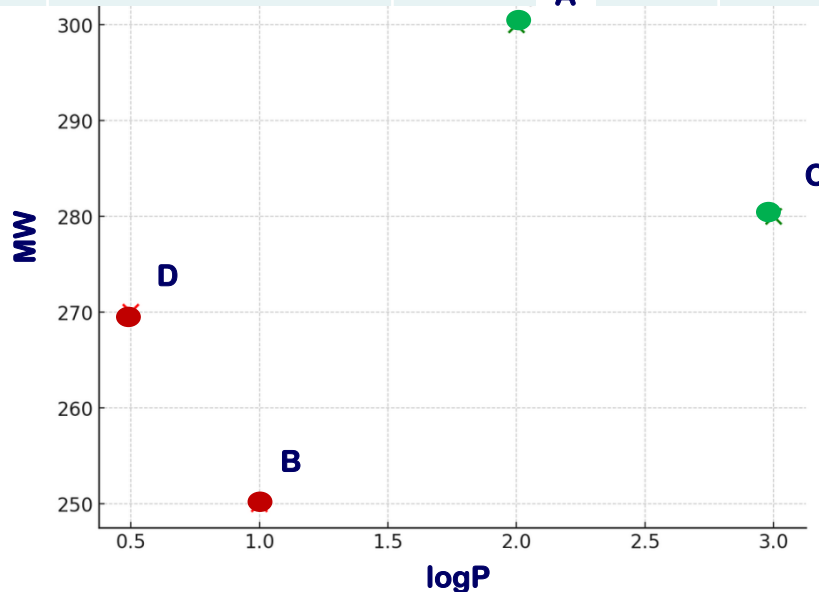
$\text{step}()$ is a function that returns **A** if the argument is ≥ 0 , otherwise **NA**.



A simple example of the application of a *linear perceptron* in medchem:

Consider this four drug candidates:

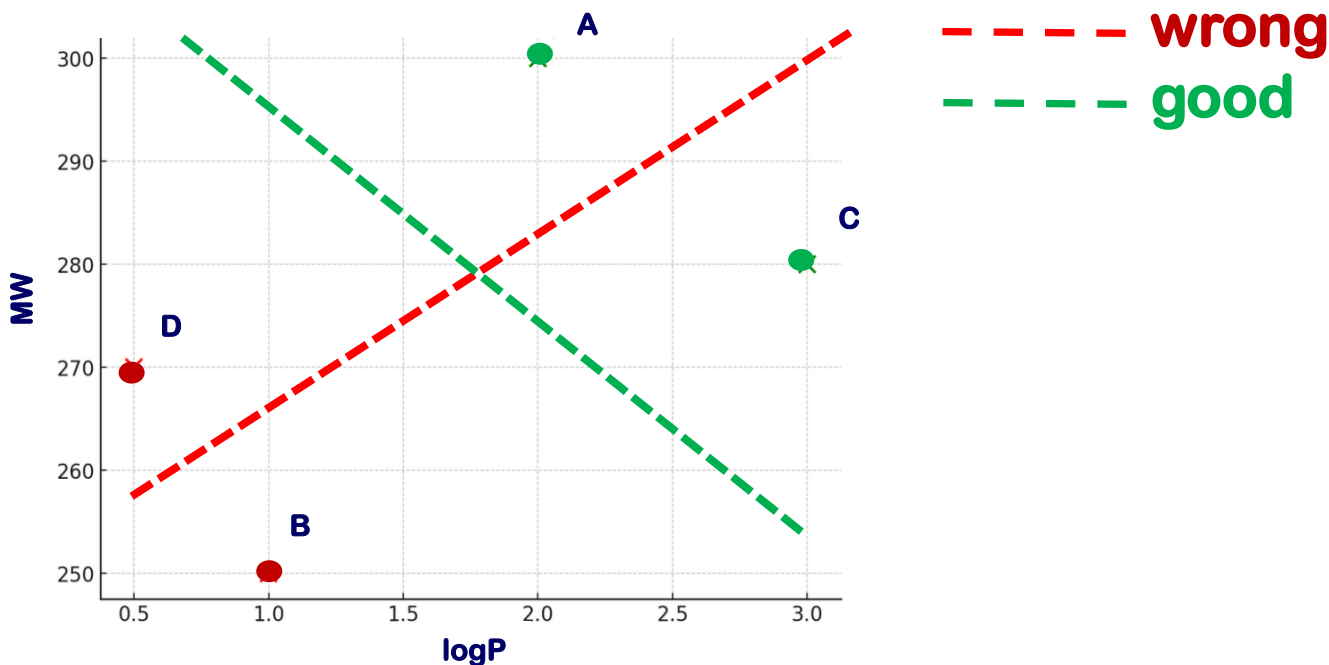
| # | logP (x_1) | MW (x_2) | Activity (OUT) |
|---|----------------|--------------|----------------|
| A | 2.0 | 300 | A |
| B | 1.0 | 250 | NA |
| C | 3.0 | 280 | A |
| D | 0.5 | 270 | NA |





A simple example of the application of a *linear perceptron* in medchem:

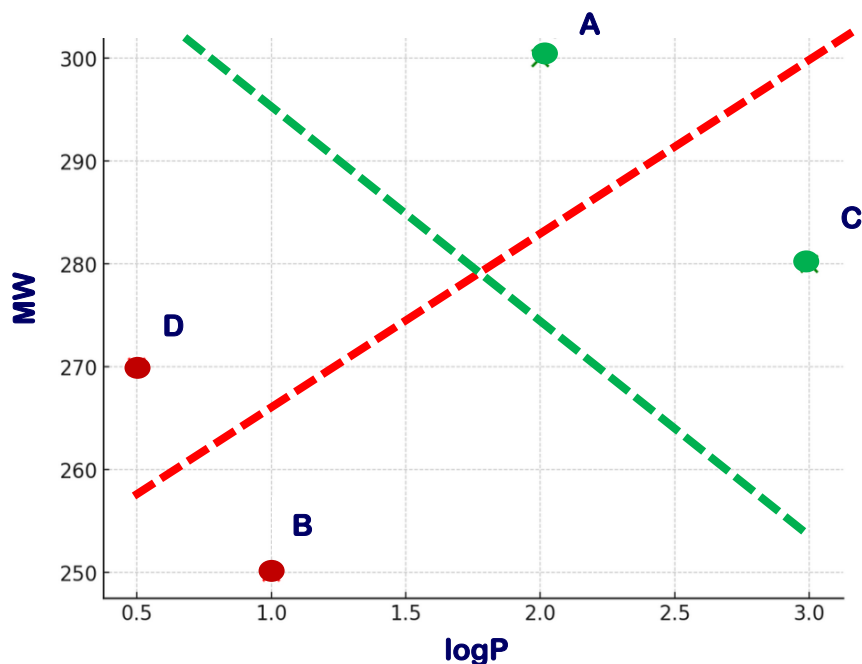
From a graphical point of view, we have to find the line (w_1 , w_2 and b) that are able to classified accurately the four drug candidates:





A simple example of the application of a *linear perceptron* in medchem:

From a graphical point of view, we have to find the line (w_1 , w_2 and b) that are able to classified accurately the four drug candidates:



--- wrong
--- good

$$\text{OUT} = \text{step} (w_1 x_1 + w_2 x_2 + b)$$

We have to determine the values of w_1 , w_2 and b that minimize the error in OUT



A simple example of the application of a *linear perceptron* in medchem:

The minimum number of points needed to have a significant linear separation depends on:

The dimensionality of the space (i.e. the number of descriptors used)

The distribution of the data

The degree of generalization desired (i.e. how well the model also fits new data)



A simple example of the application of a *linear perceptron* in medchem:

To linearly separate two classes in a n -dimensional space, you need at least $n+1$ non-aligned points (i.e. not all on the same side or line/plane), but:

Small dataset: at least 20–50 points (10–25 per class) to start seeing if there is useful linear separability.

Robust dataset for machine learning: at least 100–1000+ points, ideally distributed representatively across molecular space.



Limitation of the application of a *linear perceptron*:

A single perceptron (i.e., a linear classifier) can only separate linearly separable data.

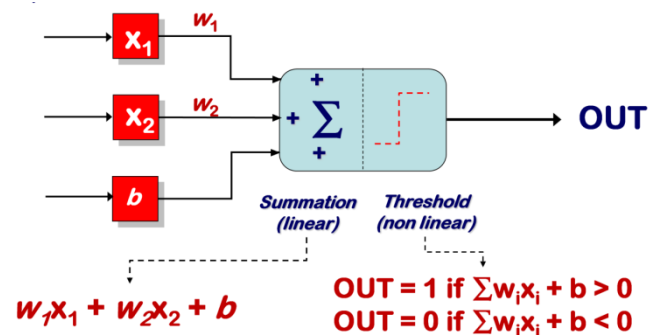
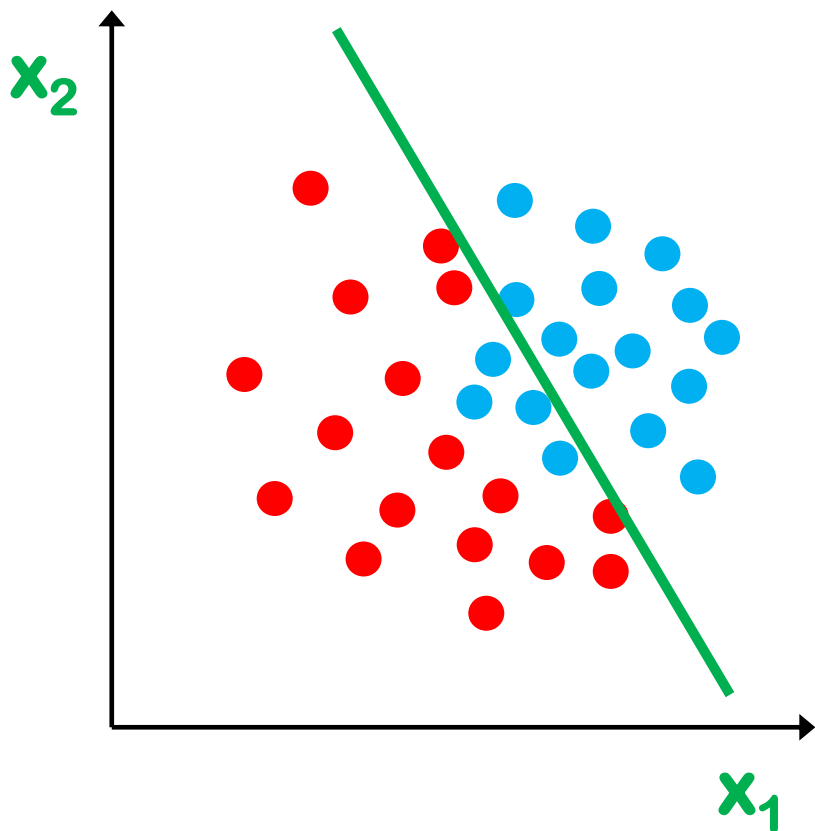
Problem: If the data is not separable by a line or a plane, the perceptron fails.

Solution: When you connect multiple perceptrons together - especially in multiple layers - you get a *Multi-Layer perceptron (MLP) neural network*.



Introduction to a *Multi-Layer perceptron (MLP)* neural network :

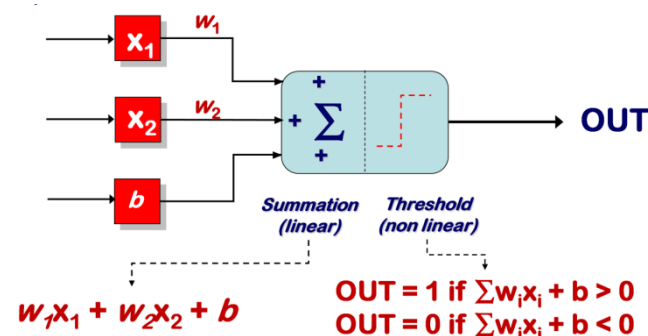
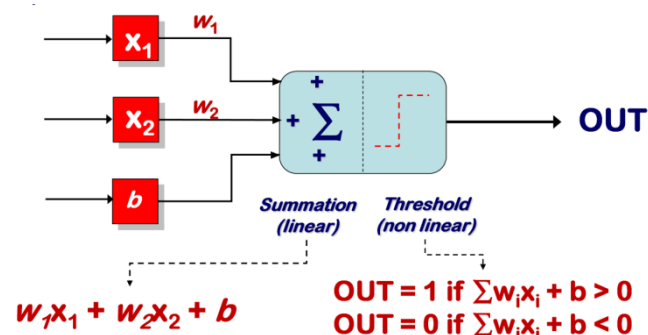
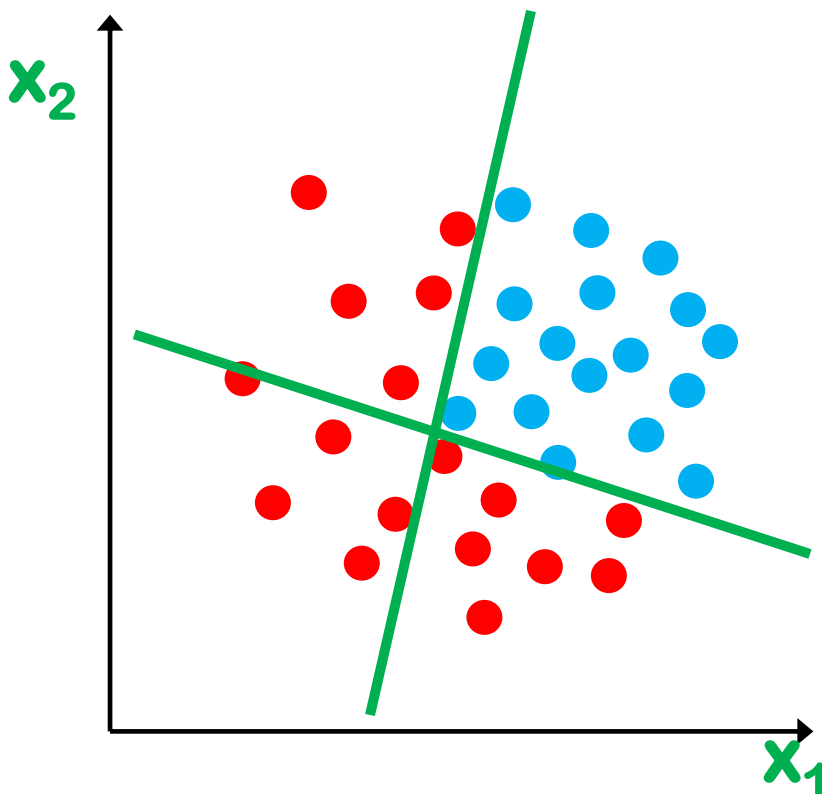
Problem: If the data is not separable by a line or a plane, the perceptron fails.





Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Solution...



Two different single layer perceptron with their decision boundary on non linear datasets.



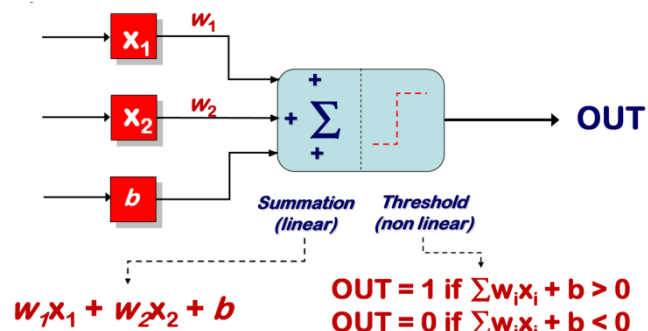
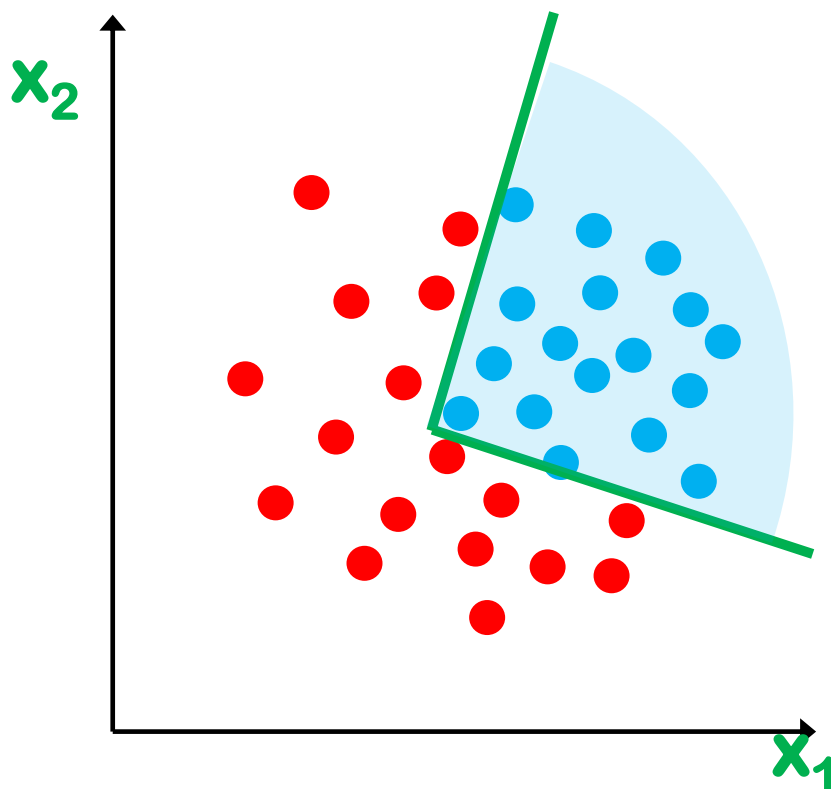
Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Solution: To overcome this limitation, the multilayer perceptron (MLP) was introduced. MLP networks contain multiple layers, including hidden layers, which introduce non-linear transformations through activation functions. This allows the network to capture complex non-linear relationships between input features and output classes. By incorporating multiple layers and non-linear activation functions, MLP networks can build decision boundaries that can flexibly adapt to non-linear data patterns.

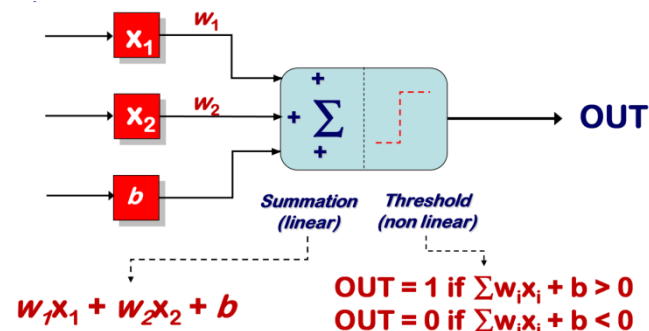


Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Solution...



+



Superimposing and smoothening the linear decision boundary of single layer perceptron to form non linear decision boundary as MLP



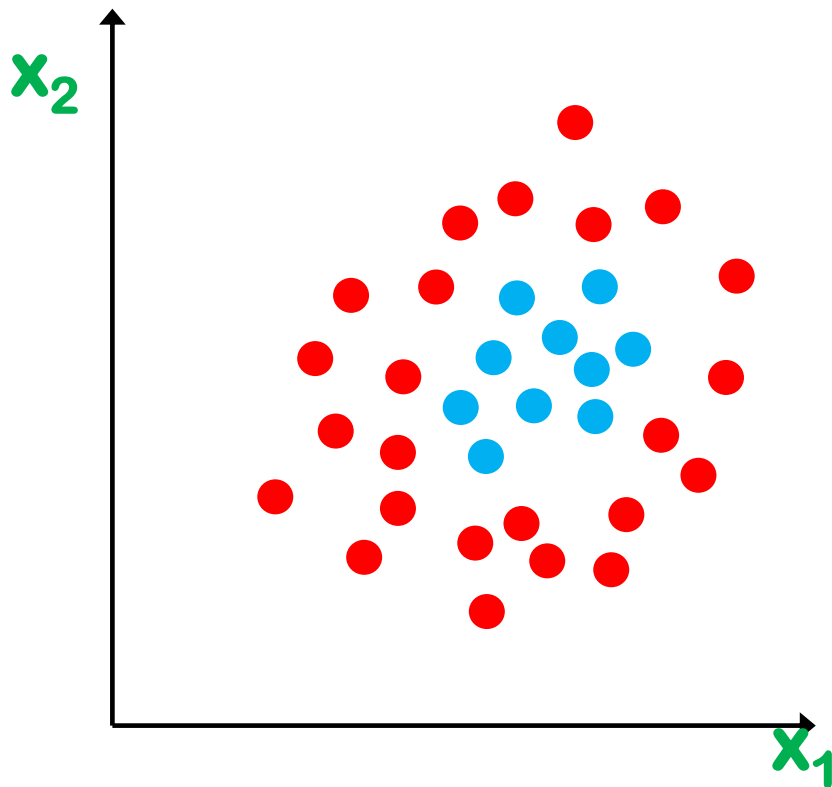
Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Solution: The ability of MLP networks to capture non-linear decision boundaries has greatly expanded the possibilities of machine learning. It enables us to tackle a wide range of complex tasks, such as image recognition, natural language processing, and speech recognition. MLP networks have demonstrated their effectiveness in handling data with intricate non-linear relationships, offering superior accuracy and predictive power compared to single layer perceptrons.



Introduction to a *Multi-Layer perceptron (MLP)* neural network :

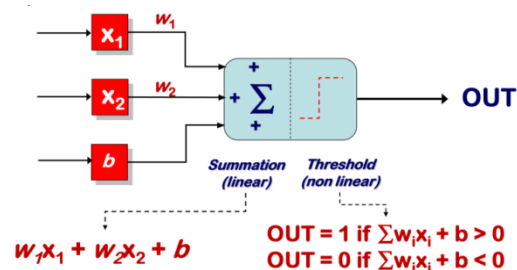
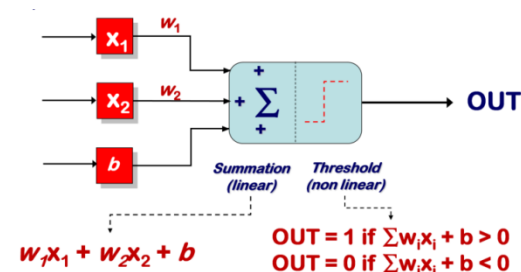
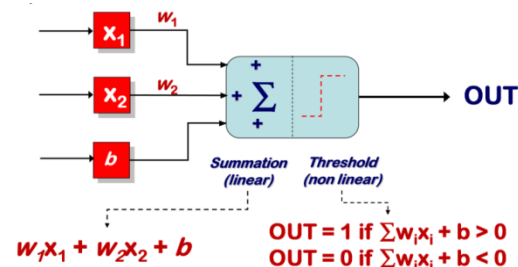
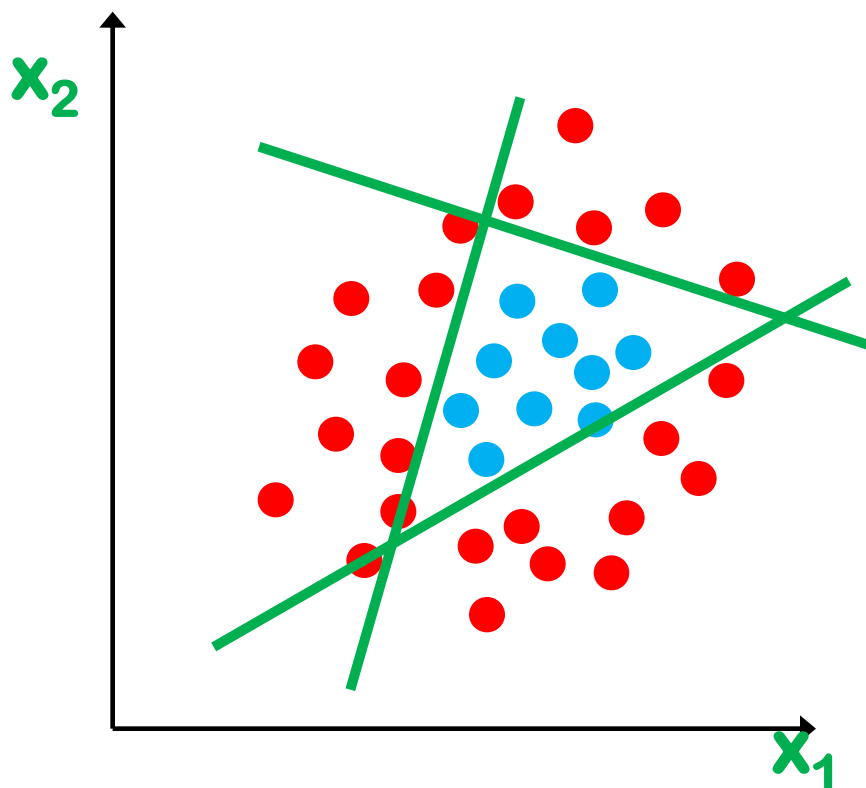
Solution...





Introduction to a *Multi-Layer* perceptron (MLP) neural network :

Solution...

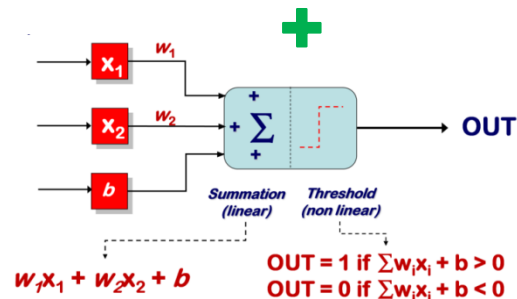
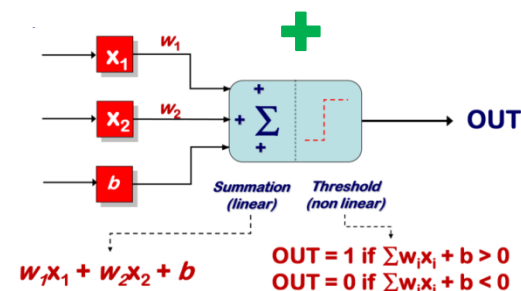
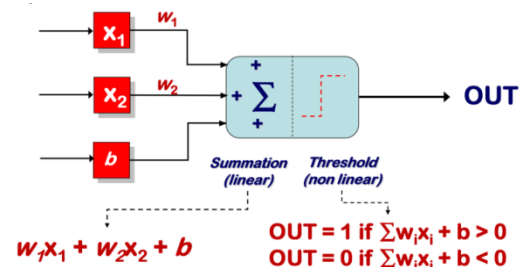
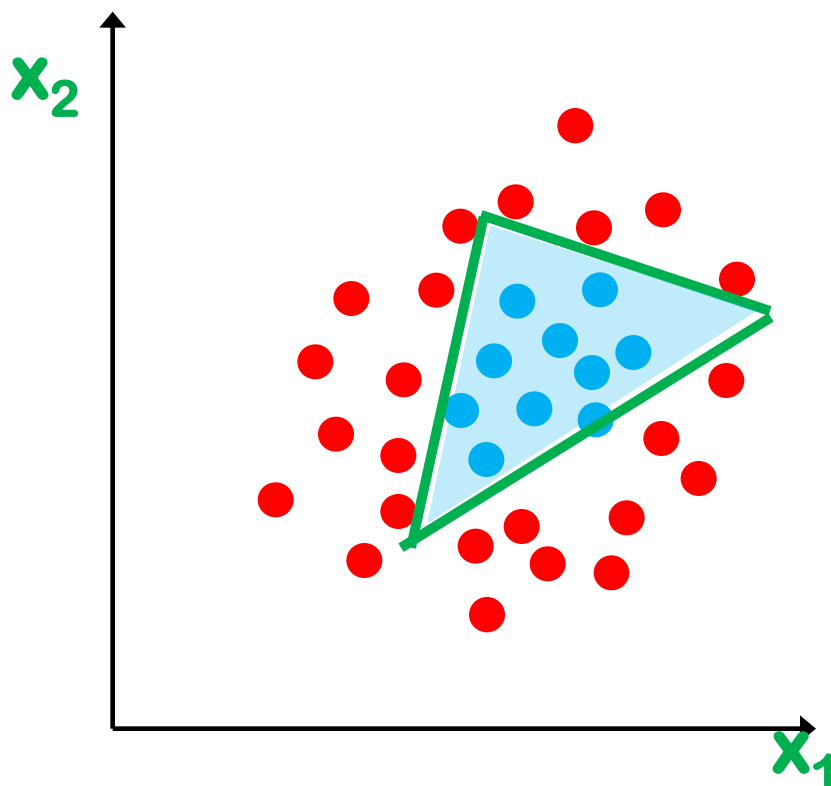


Three different single layer perceptron with their decision boundary on non linear datasets.



Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Solution...



Superimposing and smoothening the linear decision boundary of single layer perceptron to form non linear decision boundary as MLP



Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Solution: Increasing Nodes in the Hidden Layer

Adding more nodes/neurons to the hidden layer increases the network's capacity to learn complex patterns and relationships in the data.

With more nodes, the network can represent a larger set of non-linear transformations, allowing it to capture intricate patterns.



Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Solution: Changing Input from 2D to 3D

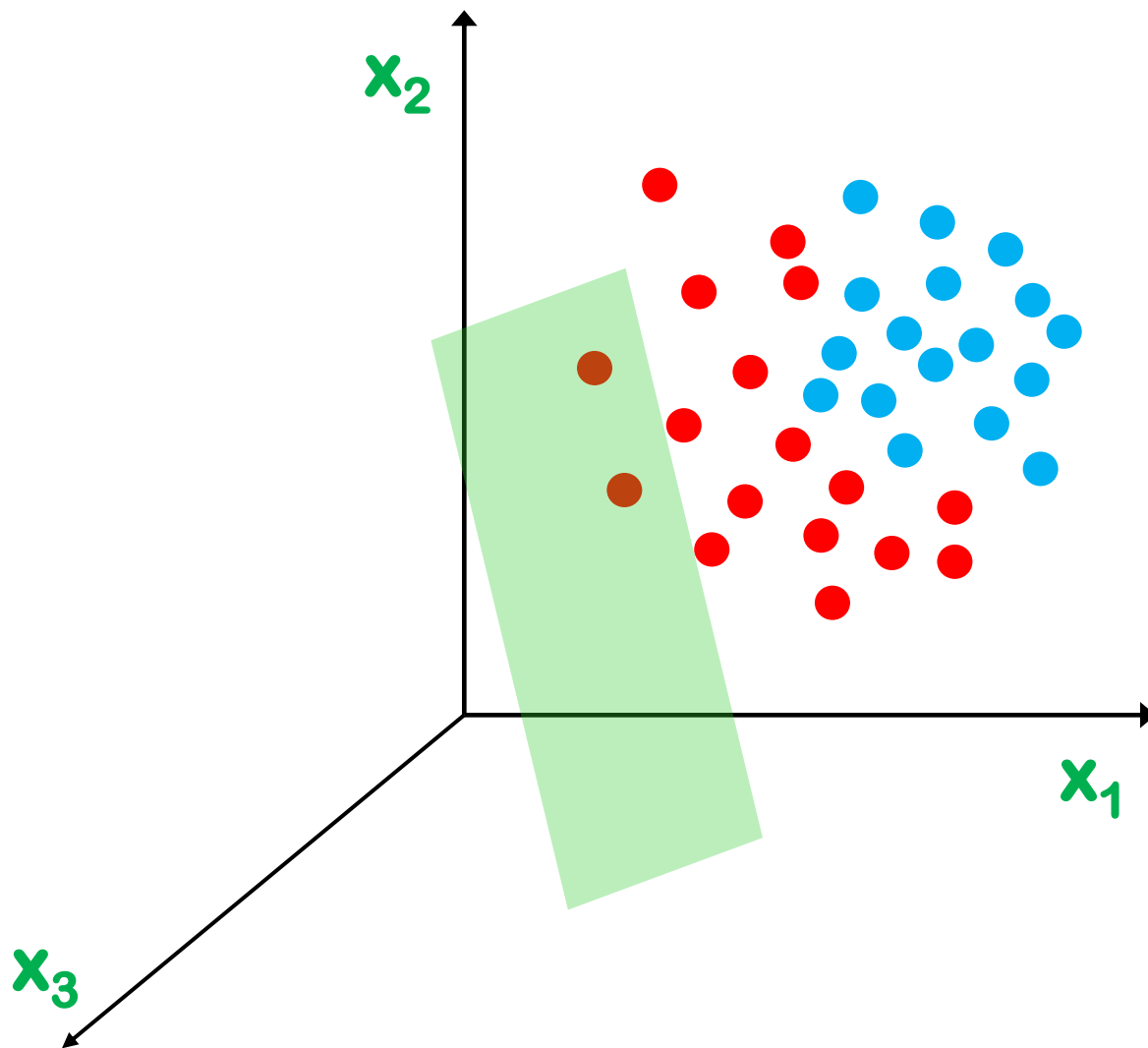
In some cases, the input data may not be effectively represented in a 2D space.

By transforming the input to a higher-dimensional space, such as converting from 2D to 3D, the network gains more expressive power to capture non-linear patterns.

Additional dimensions can reveal hidden correlations and relationships that were not apparent in the original 2D representation.

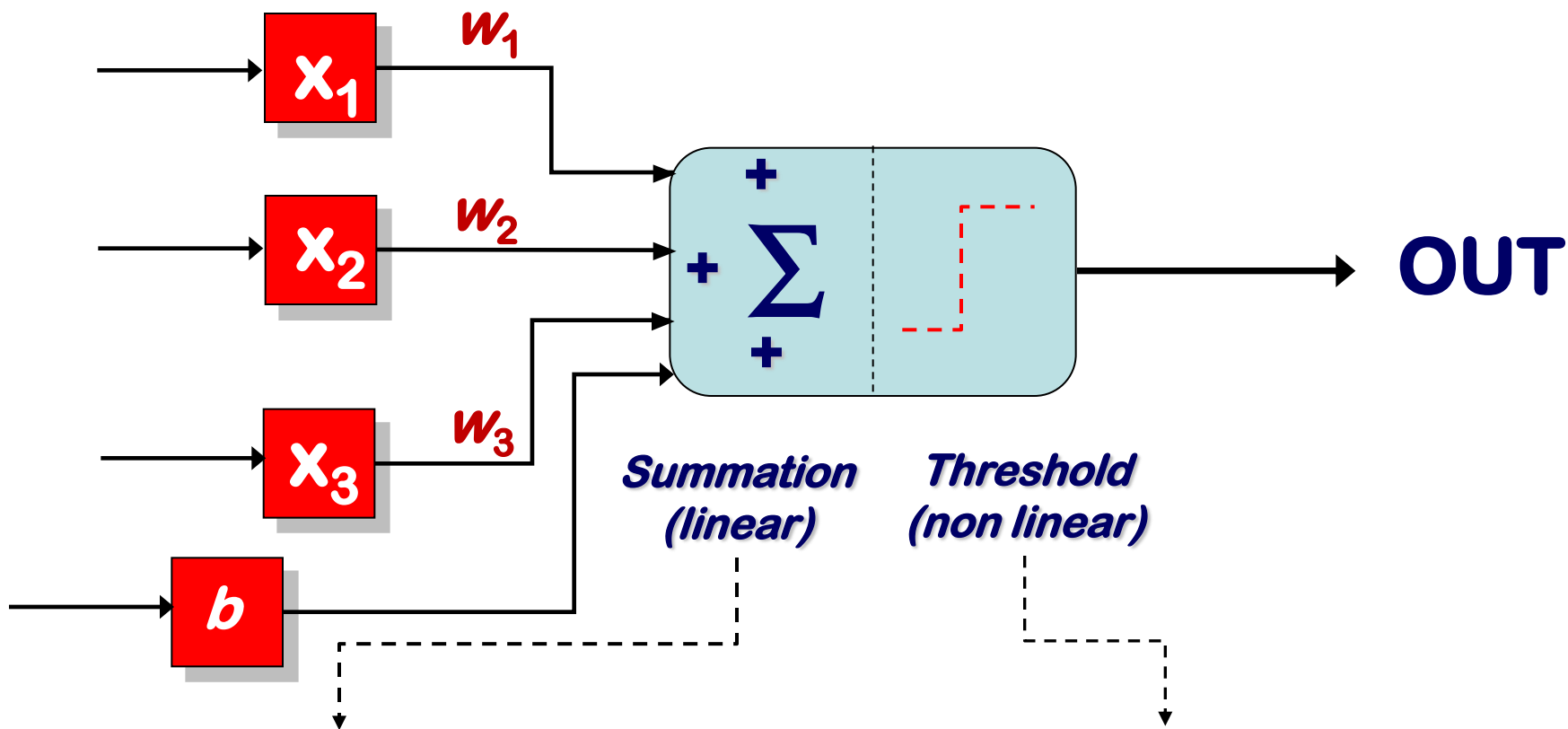


Introduction to a *Multi-Layer perceptron (MLP)* neural network :





Introduction to the simple ANN - the *linear perceptron*:

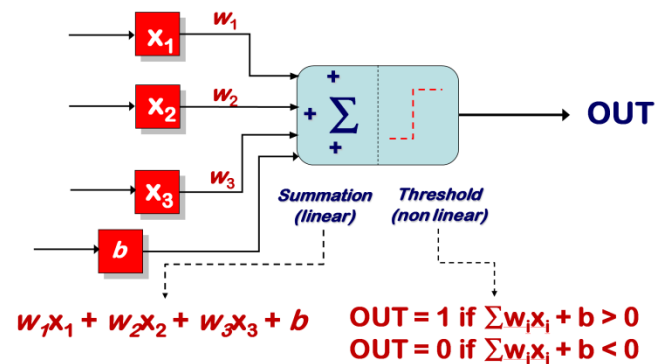
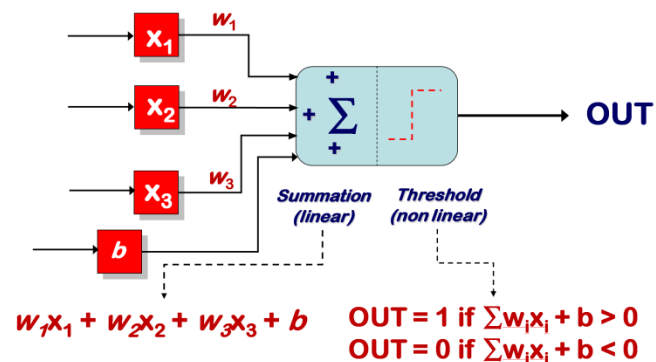
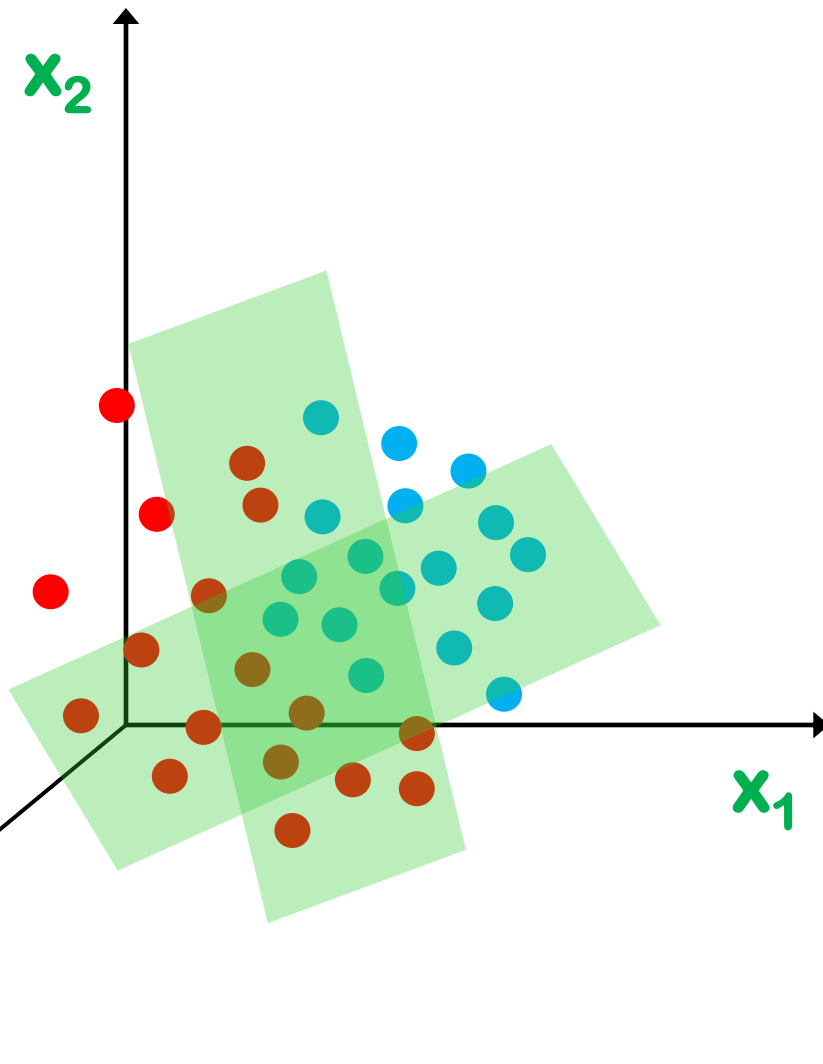


$$w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$\begin{aligned} \text{OUT} &= 1 \text{ if } \sum w_i x_i + b > 0 \\ \text{OUT} &= 0 \text{ if } \sum w_i x_i + b < 0 \end{aligned}$$



Introduction to a *Multi-Layer perceptron (MLP)* neural network :





Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Solution: Increasing Hidden Layers

Introducing additional hidden layers creates a deeper network architecture, commonly known as deep neural networks.

Deep networks have a hierarchical structure that enables them to learn and represent increasingly abstract and complex features in the data.

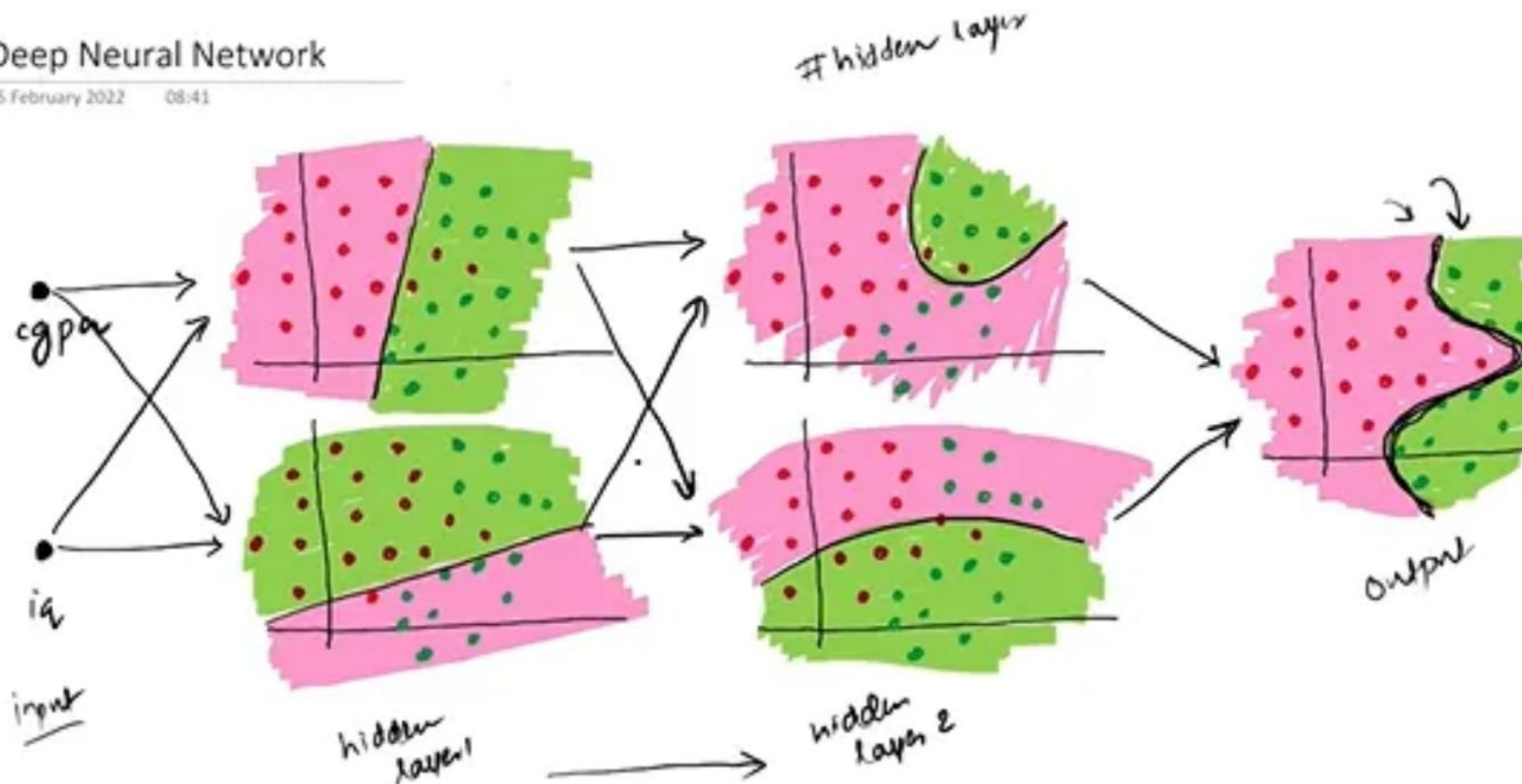
Each hidden layer extracts higher-level representations of the input, allowing the network to capture non-linear relationships more effectively.



Introduction to a *Multi-Layer perceptron (MLP)* neural network :

Deep Neural Network

25 February 2022 08:41





Introduction to the simple ANN - the *linear perceptron*: LOSS FUNCTION

A *loss function* helps a neural network to determine how wrong its predictions are, based on which the optimizer takes steps to minimize the error.

The term loss refers to the error in the prediction of a neural network. A loss function, therefore, is a function that calculates the loss for a certain prediction. The loss function is required by the learning algorithm (or optimizer) in order to decide what steps it should take to minimize the loss.



Introduction to the simple ANN - the *linear perceptron*: LOSS FUNCTION

Binary Cross-Entropy (log Loss): the most common for binary classification

What does log-loss conceptually mean? Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in case of binary classification). The more the predicted probability diverges from the actual value, the higher is the log-loss value.



Introduction to the simple ANN - the *linear perceptron*: LOSS FUNCTION

How is log-loss value calculated?

$$\text{Logloss}_i = -[y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

where i is the given observation/record,

y is the actual/true value,

p is the prediction probability,

and \ln refers to the natural logarithm (logarithmic value using base of e , $e \approx 2.718$) of a number.

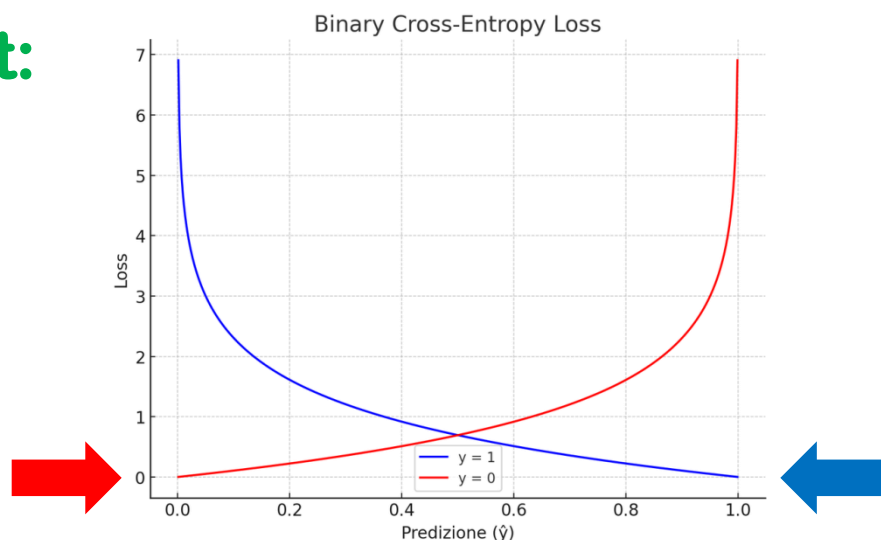


Introduction to the simple ANN - the *linear perceptron*: LOSS FUNCTION

How is log-loss value calculated?

$$\text{Logloss}_i = -[y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

Let's interpret it:



If the model predicts well ($p \approx y$), the loss is low;
If it predicts poorly (p far from y), the loss is high;
The function is convex: good for gradient descent optimization.



Introduction to the simple ANN - the *linear perceptron*: LOSS FUNCTION

How is log-loss value calculated?

$$\text{Logloss}_i = -[y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

What is it for?

During model training, the loss is calculated on the training data. **The weights w_i and the bias b are updated to minimize this loss** (with algorithms such as gradient descent). *The process is repeated until the loss is sufficiently low.*



Introduction to the simple ANN - the *linear perceptron*: LOSS FUNCTION

How is log-loss score of a model calculated?

As shown above, log-loss value is calculated for each observation based on observation's actual value (y) and prediction probability (p). In order to evaluate a model and summarize its skill, *log-loss score of the classification model is reported as average of log-losses of all the observations/predictions.*

$$\text{Logloss} = \frac{1}{N} \sum_{i=1}^N \text{logloss}_i$$

$$\text{Logloss} = -\frac{1}{N} \sum_{i=1}^N [y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

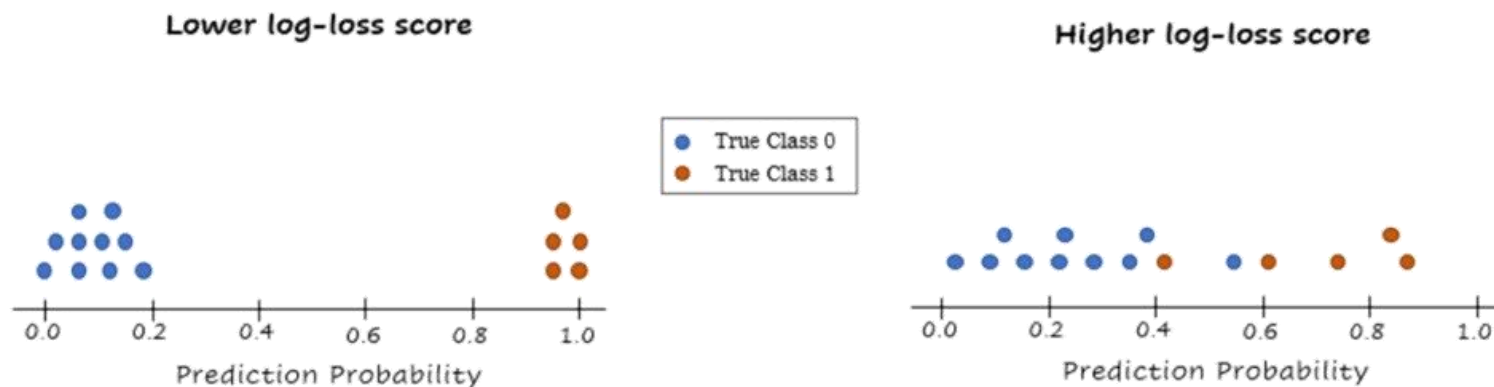
where N is the number of observations.



Introduction to the simple ANN - the *linear perceptron*: LOSS FUNCTION

How is log-loss score of a model calculated?

A model with perfect skill has a log-loss score of 0. In other words, the model predicts each observation's probability as the actual value.

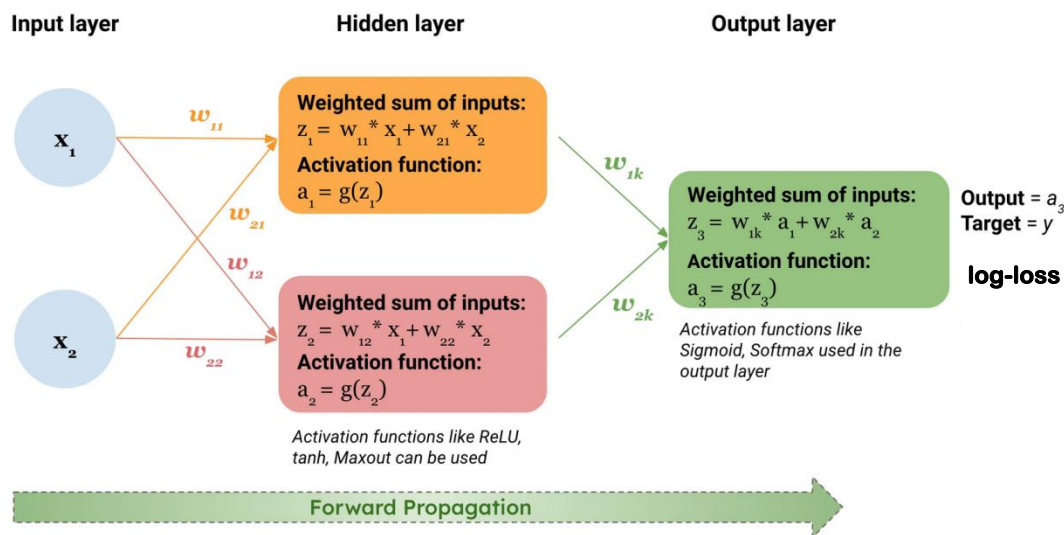


Note: A model with lower log-loss score is better than the one with higher log-loss score, provided both the models are applied to the same distribution of dataset. *We cannot compare log-loss scores of two models applied on two different datasets.*



Introduction to a *Multi-Layer perceptron (MLP)* neural network : FORWARD PROPAGATION

The process from the input layer through the hidden layers to the output layer is called forward propagation. In each layer, the aforementioned steps (weighted sum, bias addition, activation function) are applied to compute the layer's output. In an MLP, information flows in one direction, from the input layer through the hidden layers to the output layer. There are no feedback loops or recurrent connections, hence the name feedforward architecture.

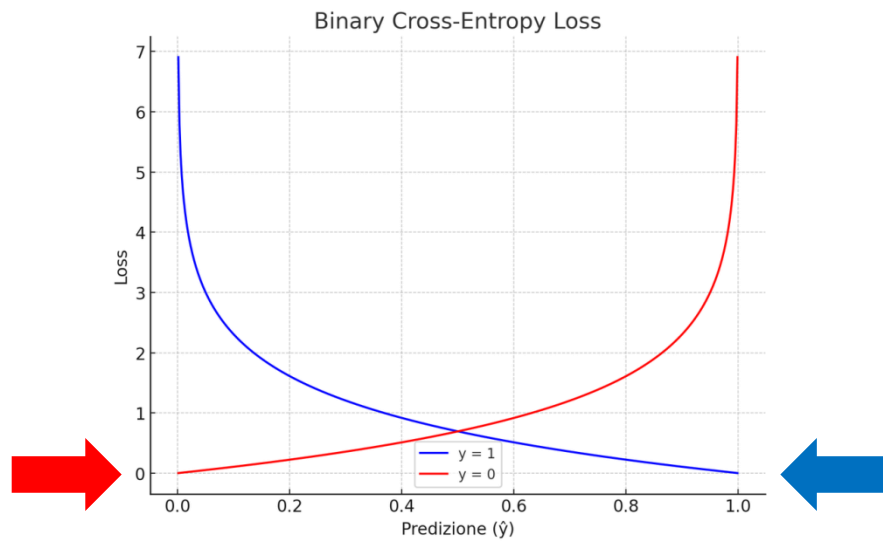




Introduction to a *Multi-Layer perceptron (MLP)* neural network : LOSS FUNCTION

Remember the loss function and in particular the log-loss?






$$\text{Logloss}_i = -[y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

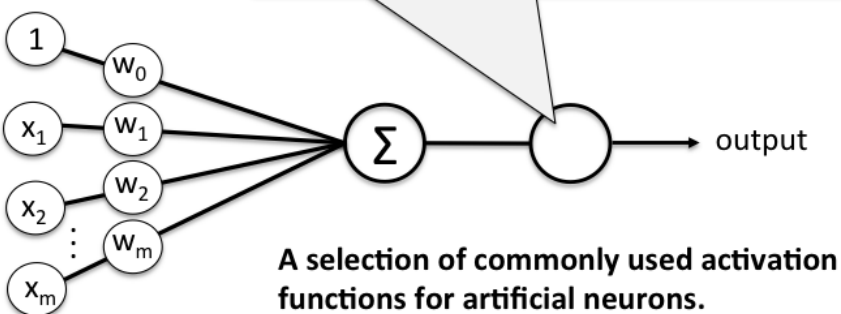


Also in MLP, during model training, the loss is calculated on the training data. **The weights w_i and the bias b are updated to minimize this loss** (with algorithms such as gradient descent). **The process is repeated until the loss is sufficiently low.**



Introduction to a *Multi-Layer perceptron (MLP)* neural network : **ACTIVATION FUNCTIONS**

| | | |
|---|------------------------------|--|
|  | Unit step | $g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise.} \end{cases}$ |
|  | | $g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise.} \end{cases}$ |
|  | Linear | $g(z) = z$ |
|  | Logistic (sigmoid) | $g(z) = 1 / (1 + \exp(-z))$ |
|  | Hyperbolic tangent (sigmoid) | $g(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}$ |
| ... | | |



An *activation function* is a crucial element in neural networks that allows the network to learn and recognize complex patterns in data. It is responsible for transforming the input data into an output value, enabling the network to make predictions or decisions. The choice of activation function is important as it can affect the network's ability to capture information and prevent the loss of input data during forward propagation and the vanishing of gradients during backward propagation. Commonly used activation functions include rectified linear units (ReLU), leaky rectified linear units (LeakyReLU), logistic sigmoid, SoftMax, tangent-Sigmoid, and hyperbolic tangent.



Introduction to a *Multi-Layer perceptron (MLP)* neural network : ACTIVATION FUNCTIONS

Activation function properties:

Non-linear: This is required to introduce non-linearity in the model.

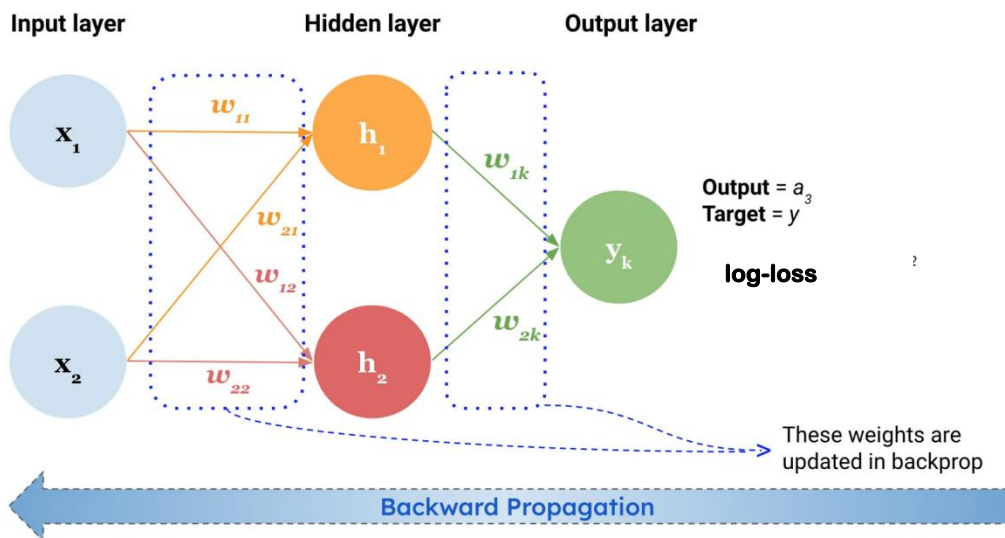
Monotonic: A function that is either entirely non-increasing or non-decreasing.

Differentiable: Deep learning algorithms update their weights via an algorithm called back propagation. This algorithm can work when the activation function used is differentiable. ie its derivatives can be calculated.



Introduction to a *Multi-Layer perceptron (MLP)* neural network : **BACK PROPAGATION and LEARNING**

Its goal is to reduce the difference between the model's predicted output and the actual output by adjusting the **weights** and **biases** in the network.





Introduction to a *Multi-Layer perceptron (MLP)* neural network : APPLICATIONS

MLPs are **universal function approximators**, i.e. they are capable of approximating any continuous function to a desired level of accuracy, given enough hidden neurons and appropriate training. This property makes them powerful tools for solving a wide range of problems including:

- Classification such as sentiment analysis, fraud detection
- Regression such as score estimation
- NLP tasks such as machine translation
- Anomaly Detection
- Speech Recognition in virtual assistant systems such as Siri, Alexa
- Computer Vision for object identification, image segmentation
- Data analytics and data visualization



Introduction to a *Multi-Layer perceptron (MLP)* neural network : APPLICATIONS IN MEDCHEM

Recommended hidden layers

Simple problem (linear QSAR, ADMET) **1 hidden layer**

Intermediate problem (nonlinearity, multi-task) **2–3 hidden layers**

Complex problem (many features, deep learning) **3–6 hidden layers**

Very deep models (with GNN/Transformer) **>6 (but only if necessary!)**



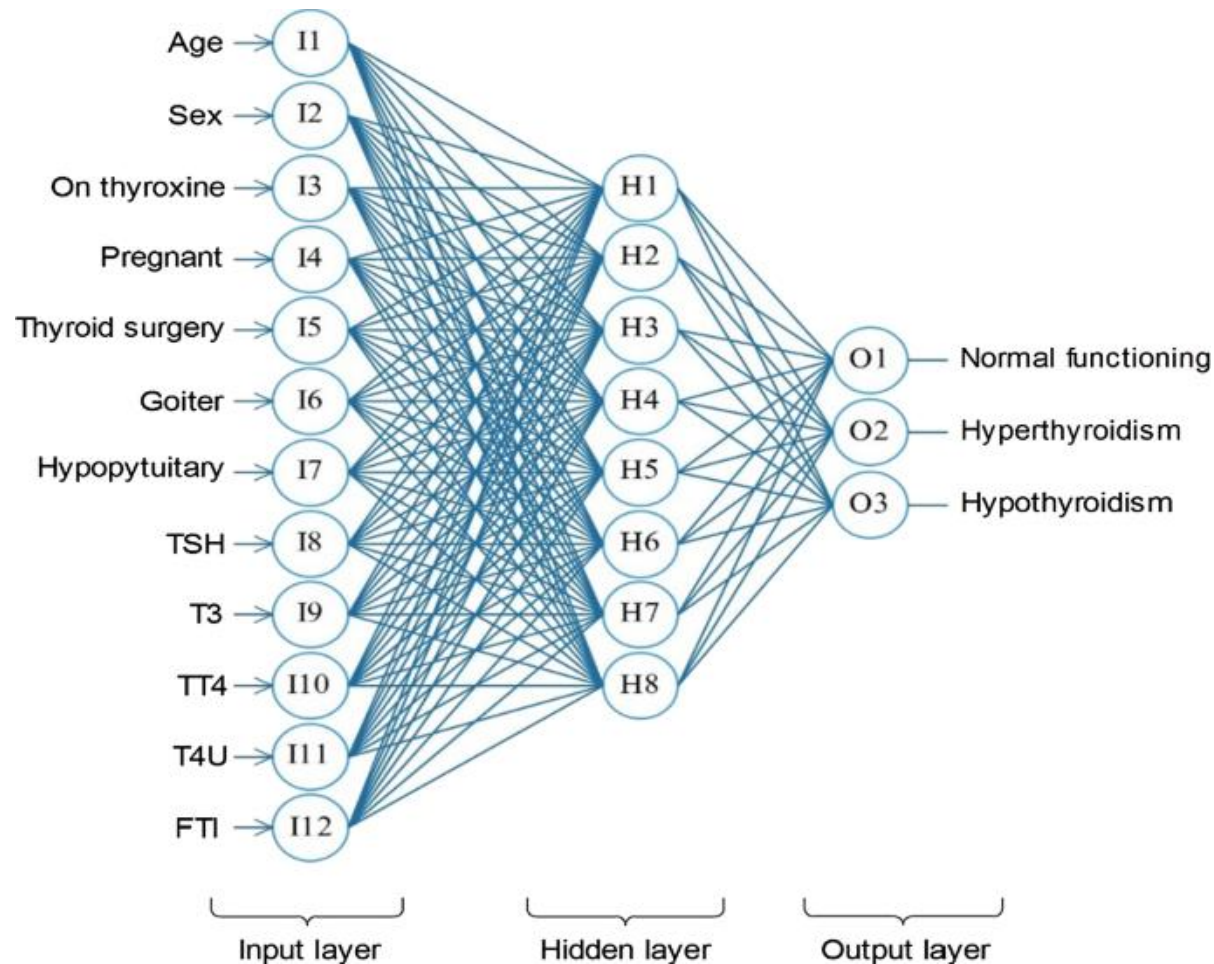
Introduction to a *Multi-Layer perceptron (MLP)* neural network : APPLICATIONS IN MEDCHEM

Recommended neurons per layer

| <i>Task type</i> | <i>Recommended maximum neurons (per layer)</i> |
|---------------------------------------|---|
| Small datasets (e.g. <1000 molecules) | < 128 |
| Medium (e.g. 5,000–50,000 molecules) | 128–512 |
| Large (e.g. >100,000 molecules) | 512–2048 sometimes up to 4096 |
| Using GPU + deep learning | higher (e.g. 8192) but beware of overfitting |



Introduction to a *Multi-Layer perceptron (MLP)* neural network : APPLICATIONS IN MEDCHEM



Hosseinzadeh, M., Ahmed, O.H., Ghafour, M.Y. *et al.* A multiple multilayer perceptron neural network with an adaptive learning algorithm for thyroid disease diagnosis in the internet of medical things. *J Supercomput* 77, 3616–3637 (2021)