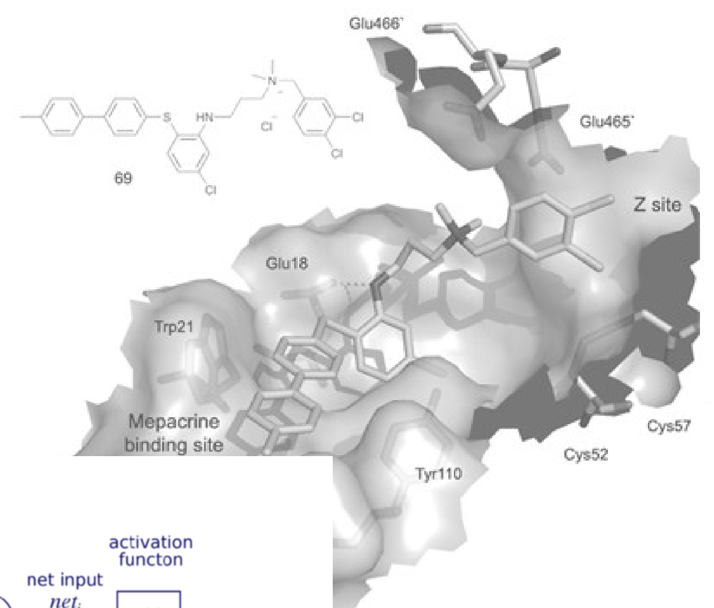
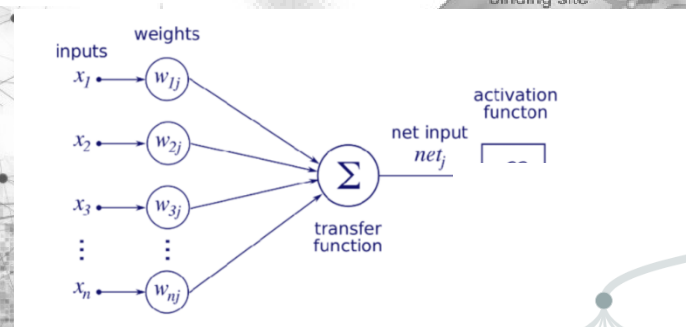
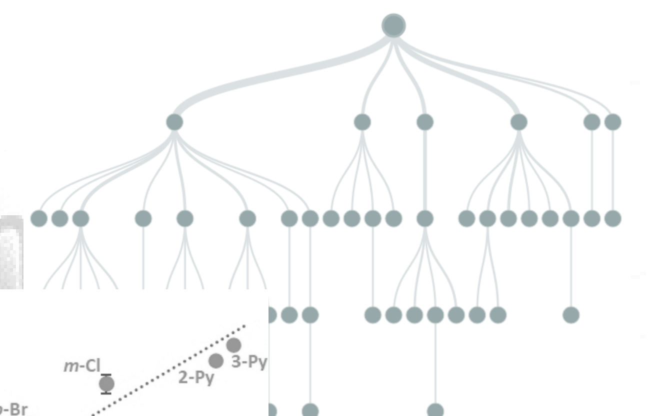
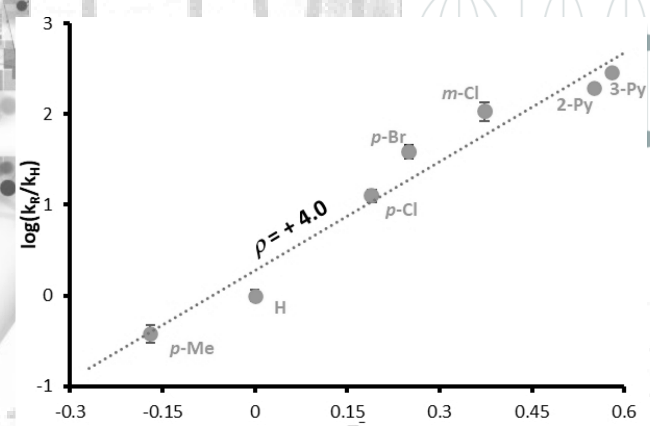
Introduzione ai metodi di Intelligenza Artificiale...



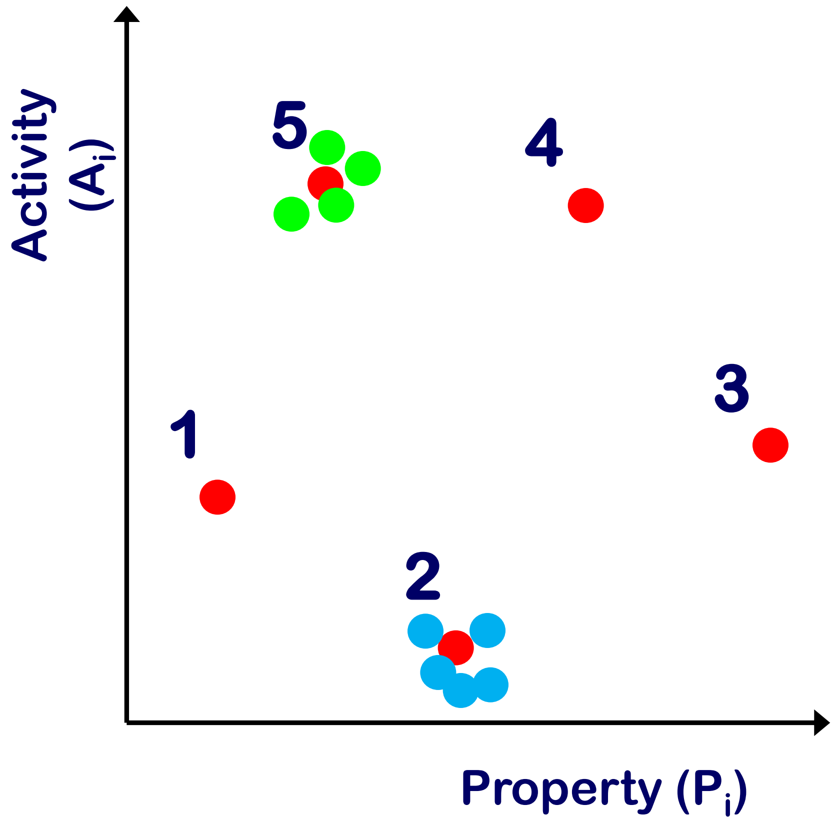






Artificial Neural Networks (ANN):

(Q)SAR: follow me in this wonderful experience



We can start from here…

What is it?

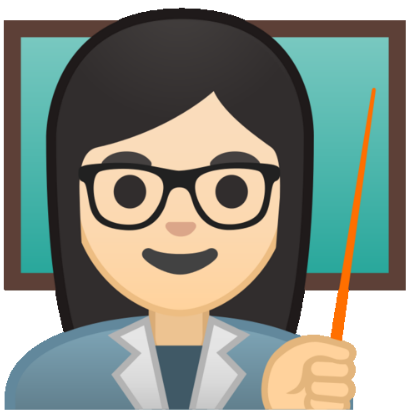
It is easy to understand there is not a mathematical function that can answer to this question…

We can approach this problem in this way..

EXPERIENCE

We can approach this problem in this way..

MULTIPLE EXPERIENCE



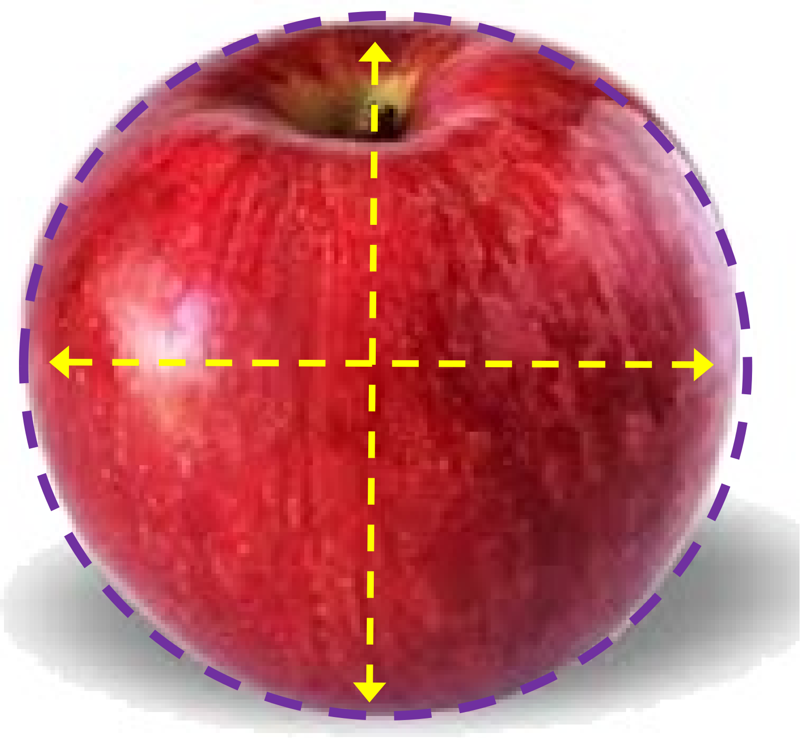
SUPERVISOR

CATEGORY (APPLE)

Why we can identify a category?

CATEGORY (APPLE)

Back to the first example…

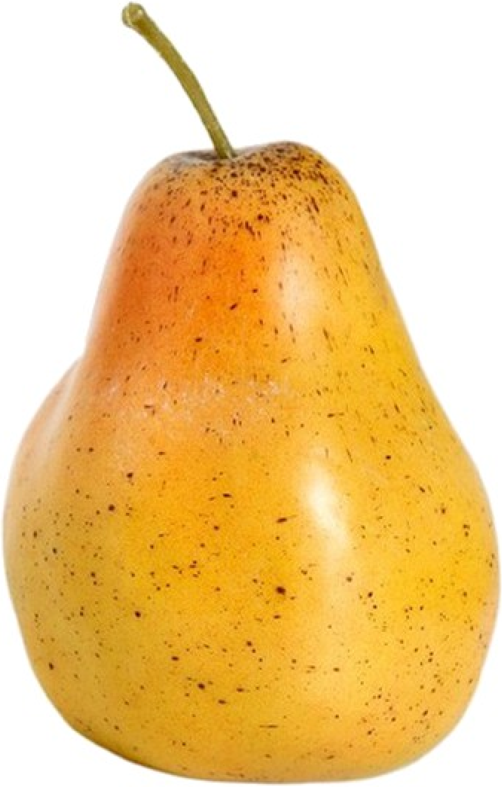


Geometrical Properties

Color

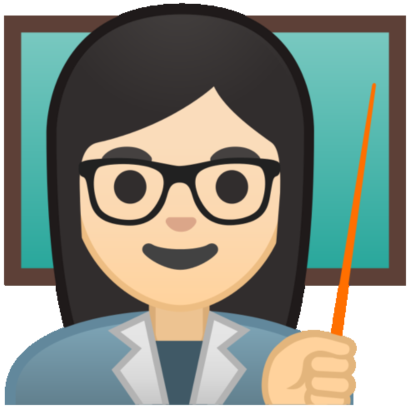
…

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

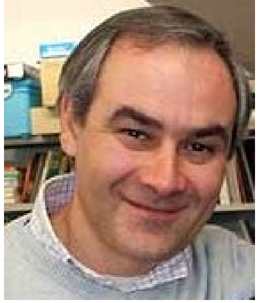
Subfield of AI giving machines the skills to learn from examples without being explicitly programmed.

Examples: Fraud detection, marketing personalization, email classification

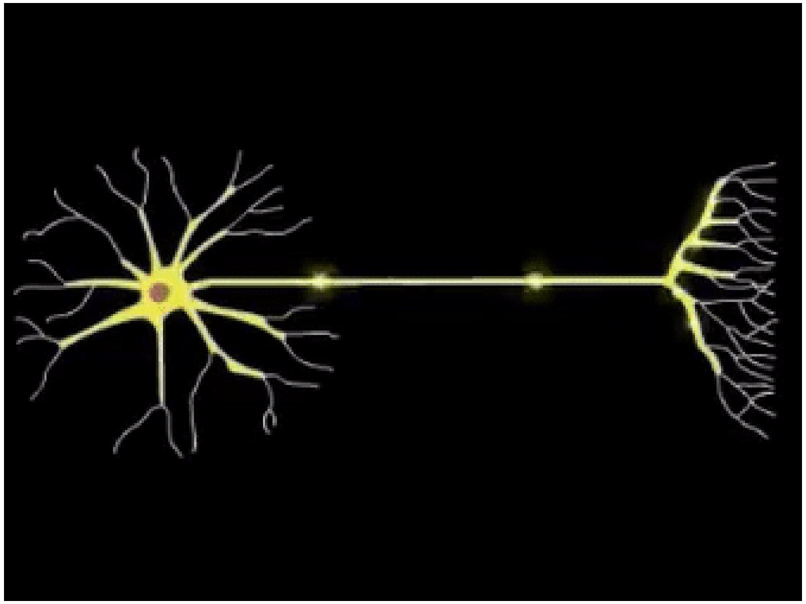
DEEP LEARNING

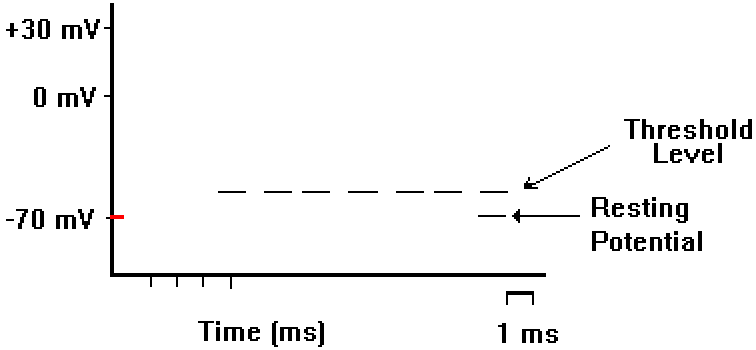
Specialized machine learning technique enabling machines to train themselves to perform tasks.

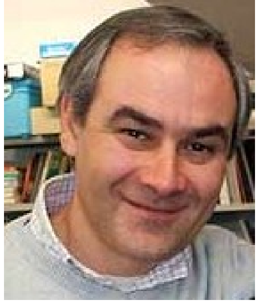
Examples: Image classification, vehicle detection, sentiment analysis



Do you remember the structure of neurons?

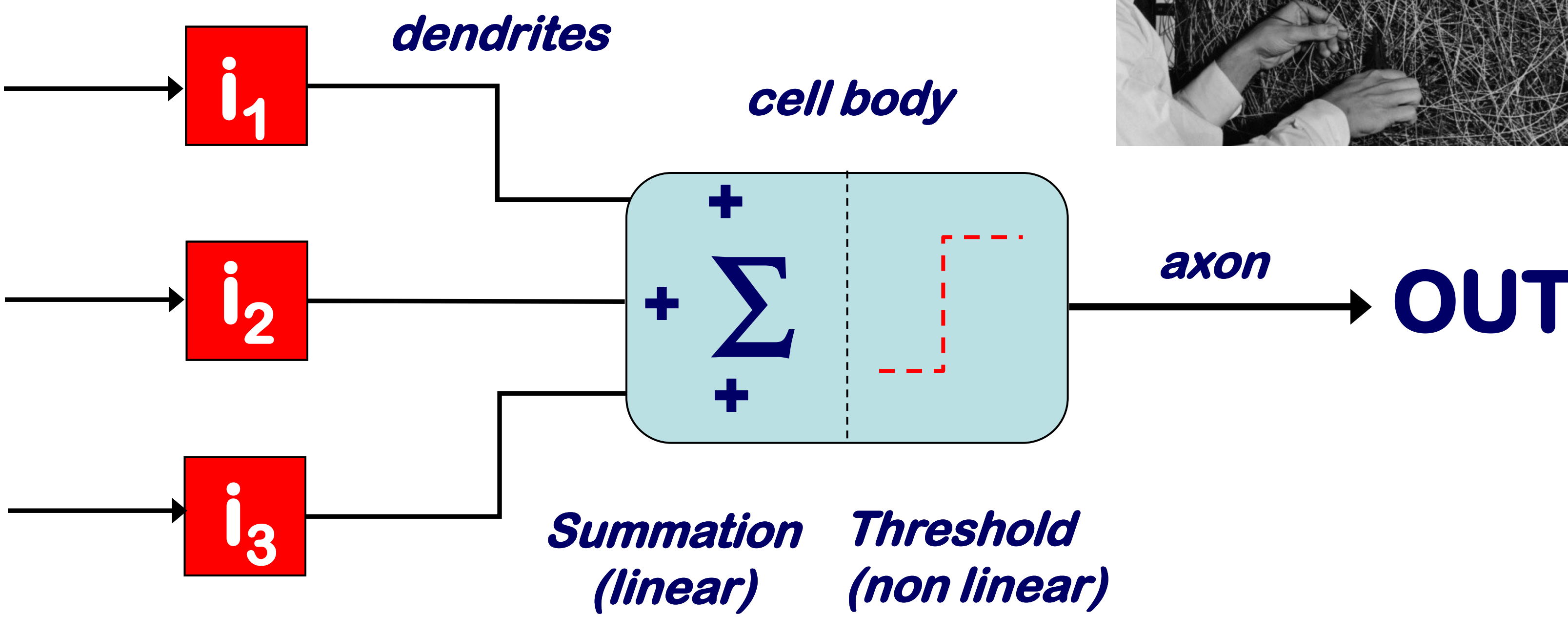






From human neurons to artificial neurons…

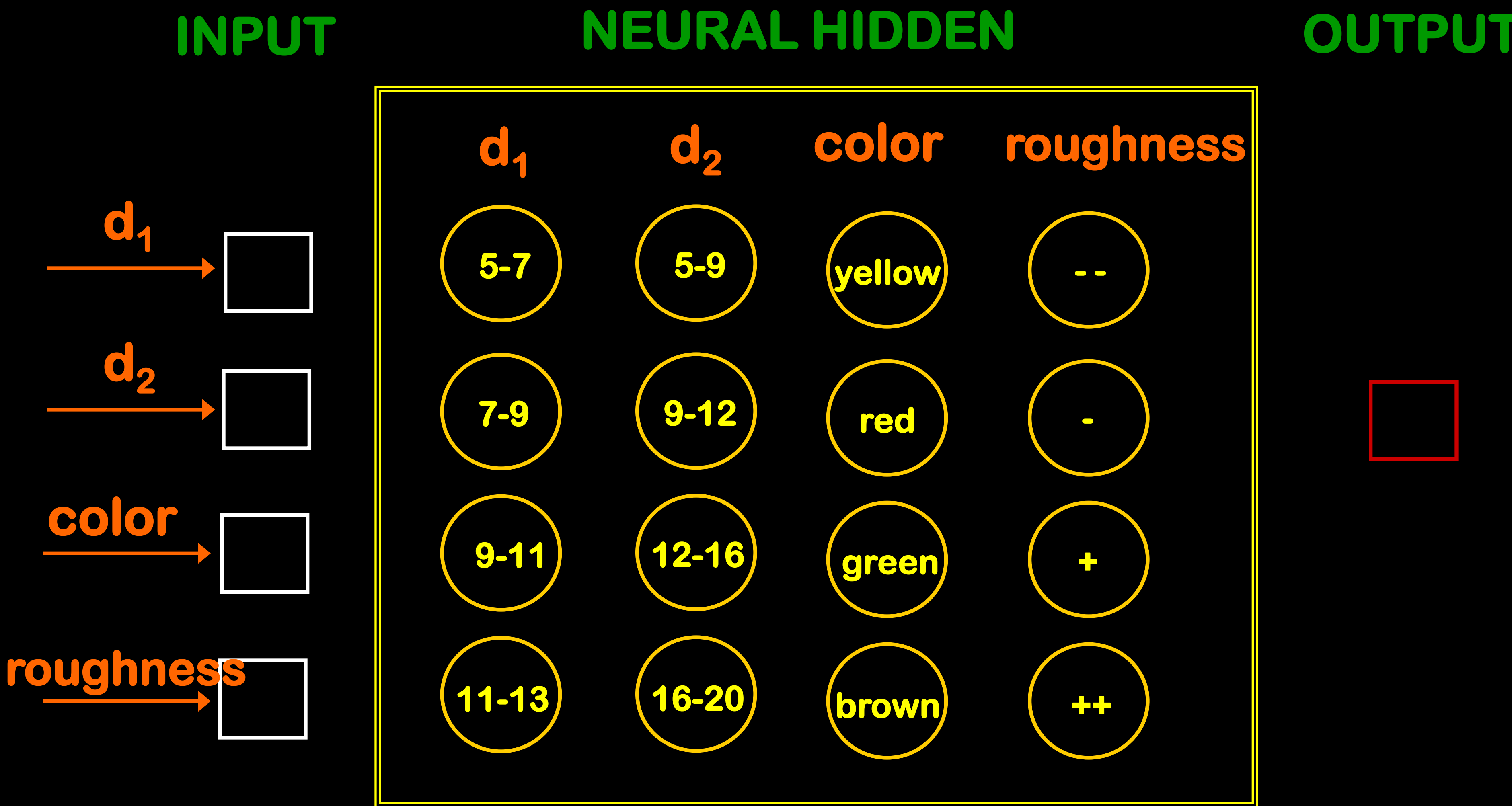
PERCEPTRON



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

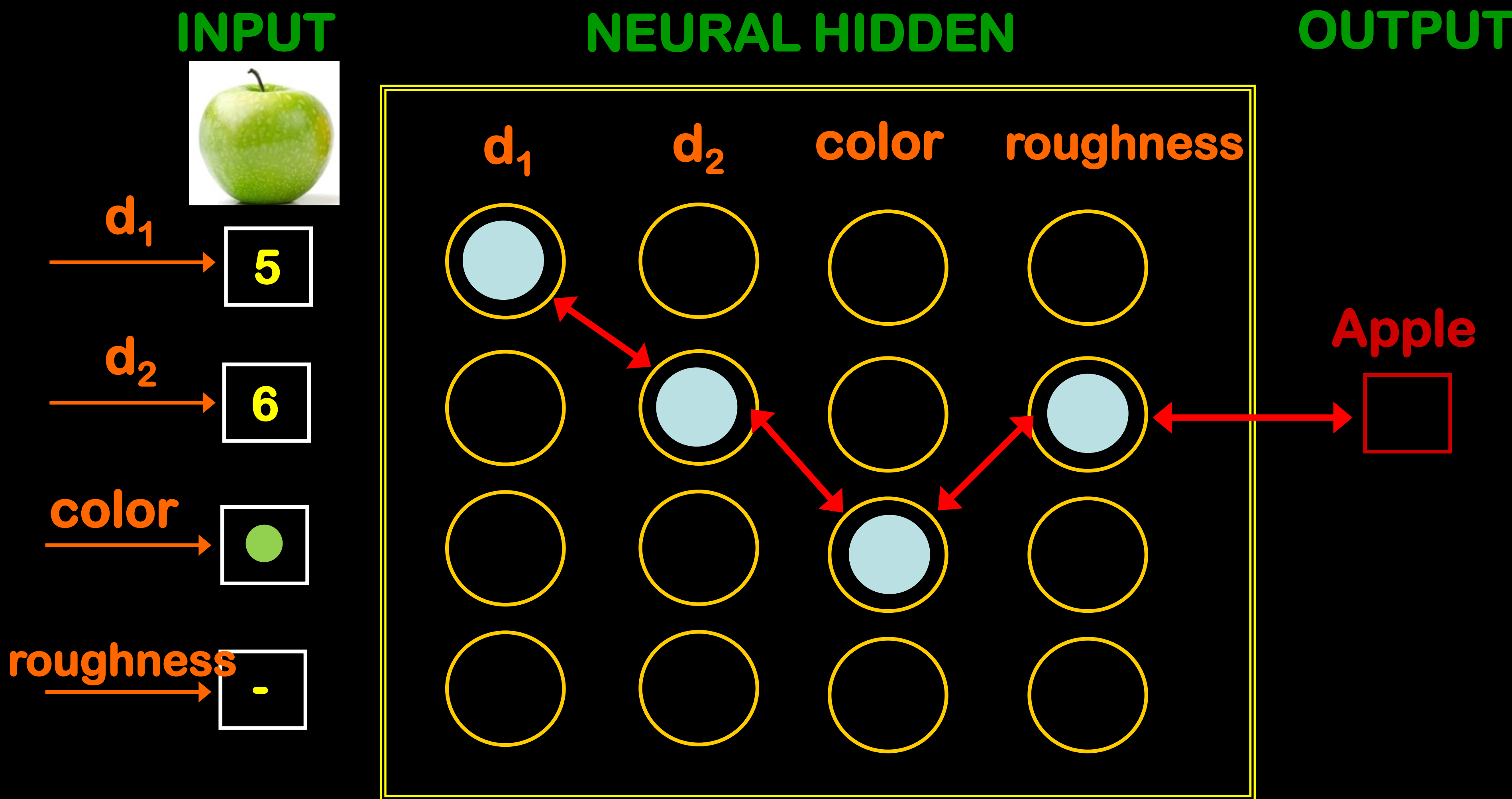
Artificial Neural Networks (ANN):

Logical structure of an ANN



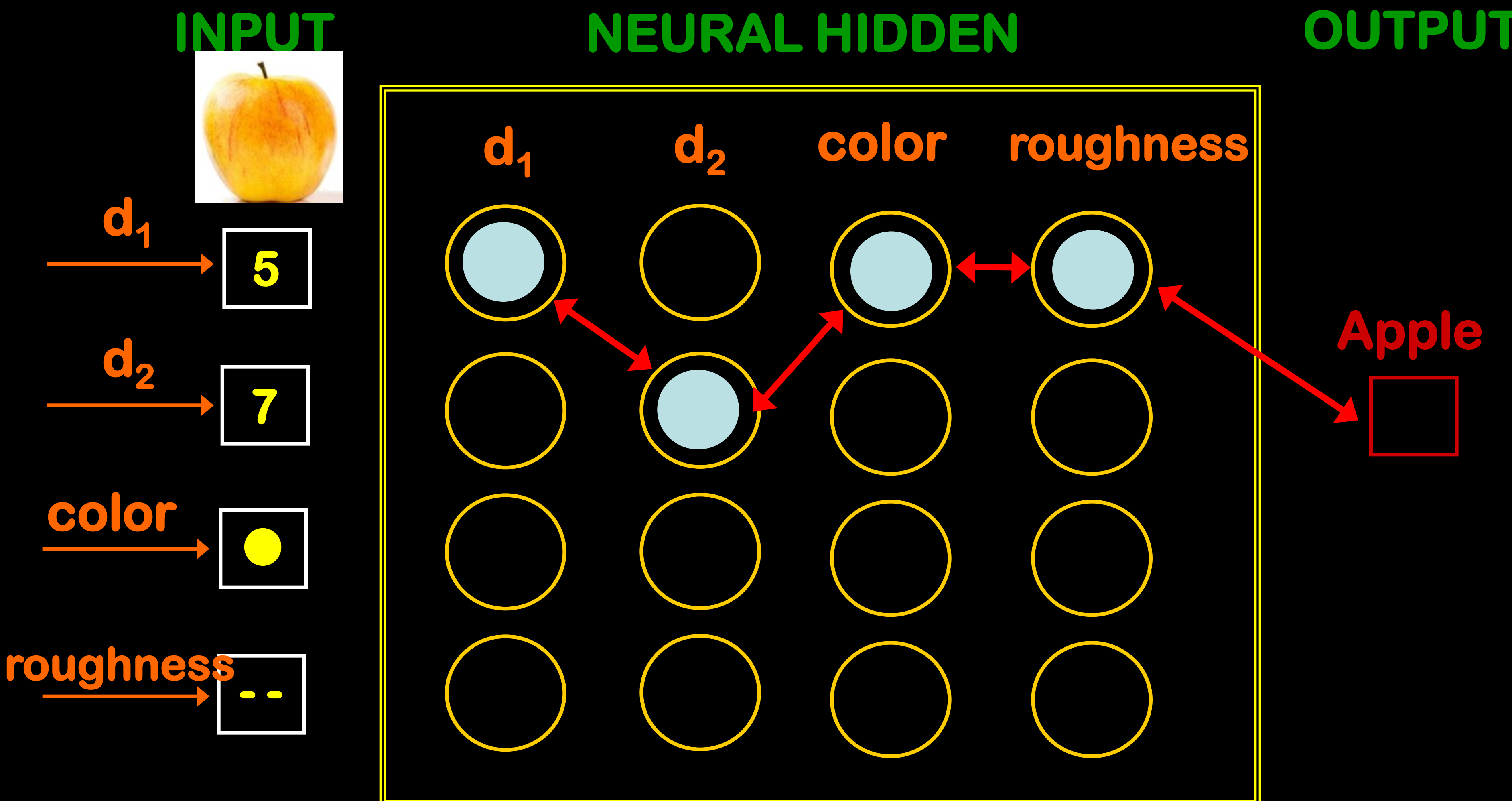
Artificial Neural Networks (ANN):

Phase 1 – Learning.



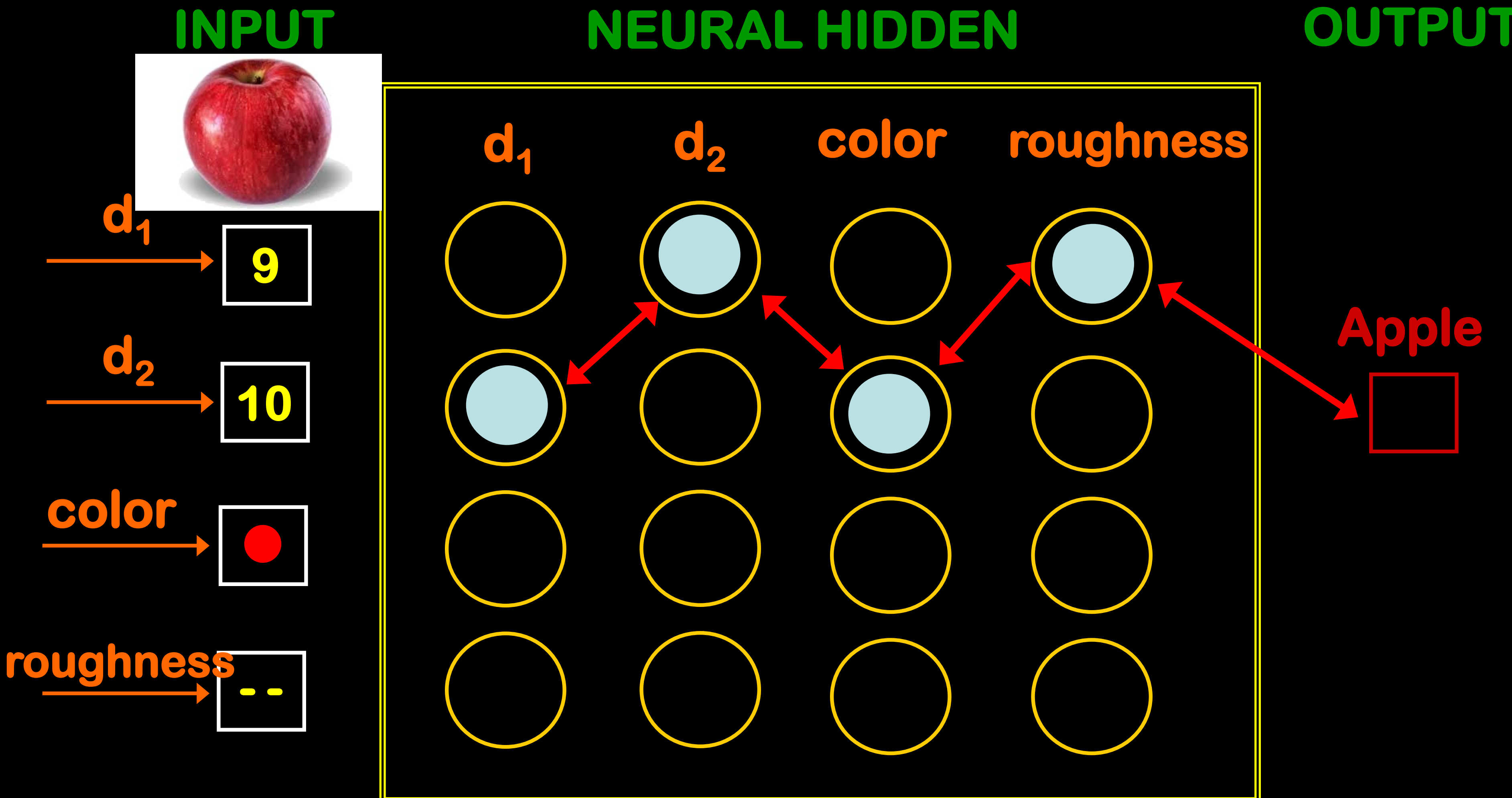
Artificial Neural Networks (ANN):

Phase 1 – Learning.



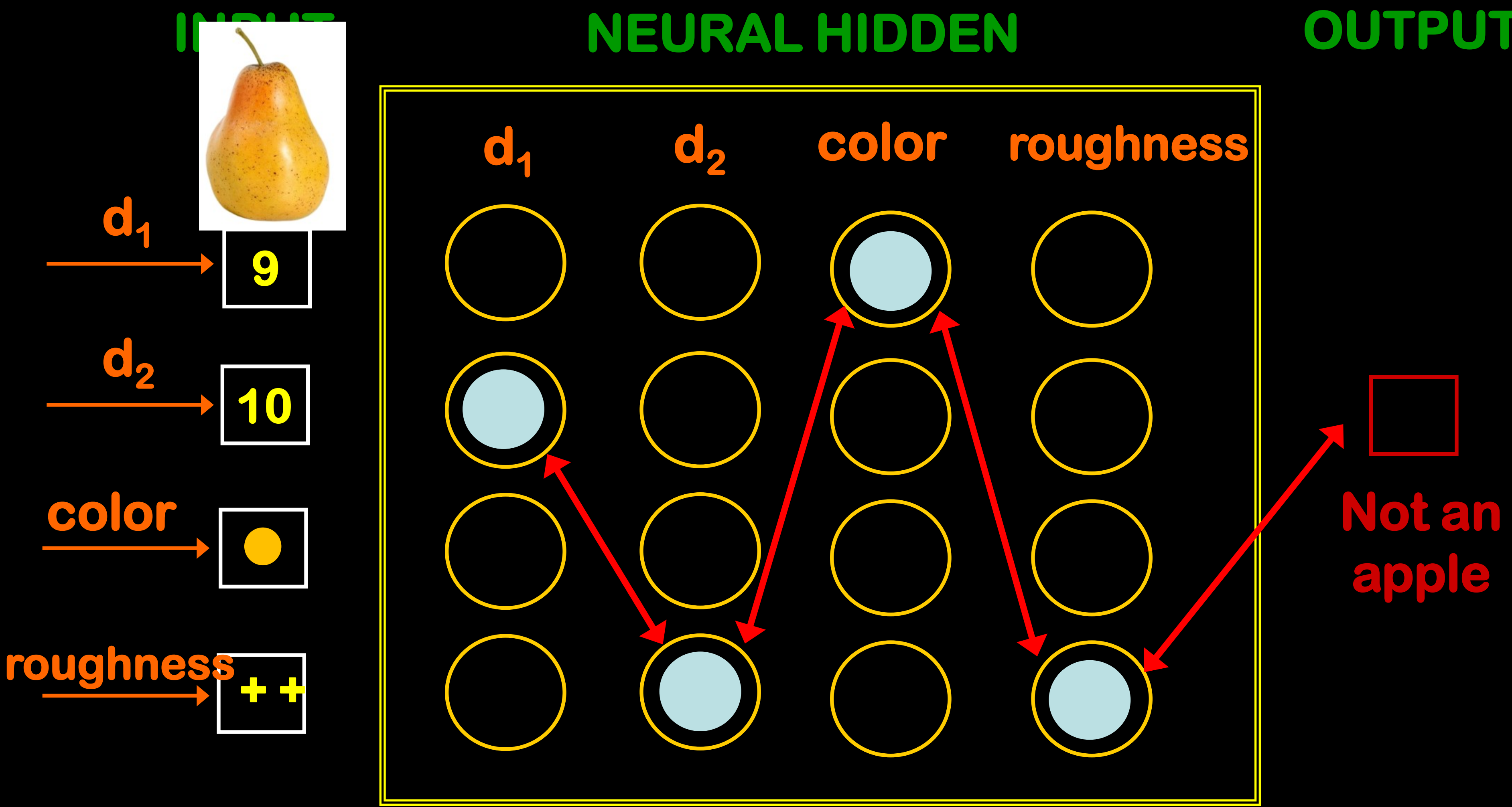
Artificial Neural Networks (ANN):

Phase 1 – Learning.



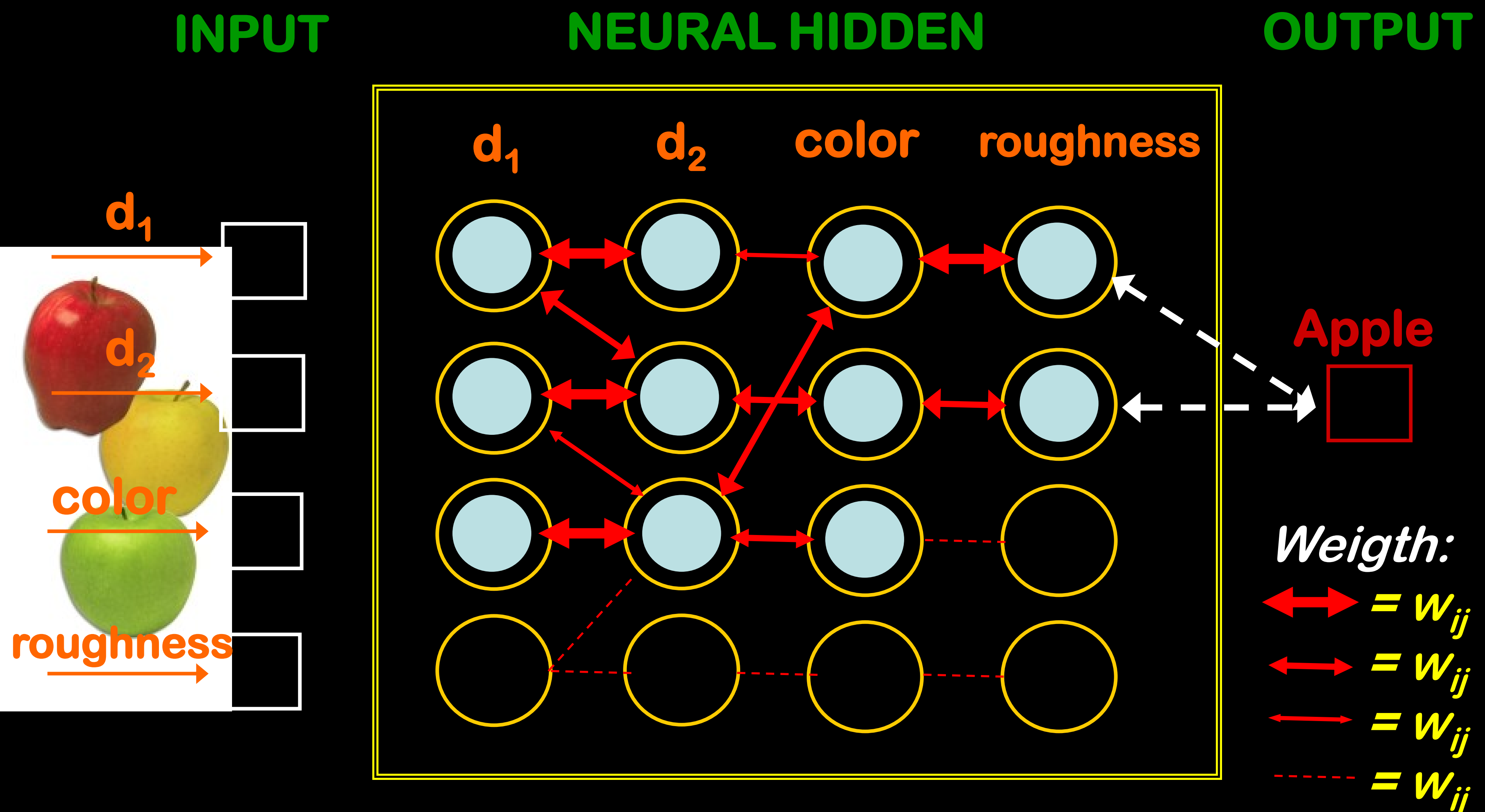
Artificial Neural Networks (ANN):

Phase 1 – Learning.



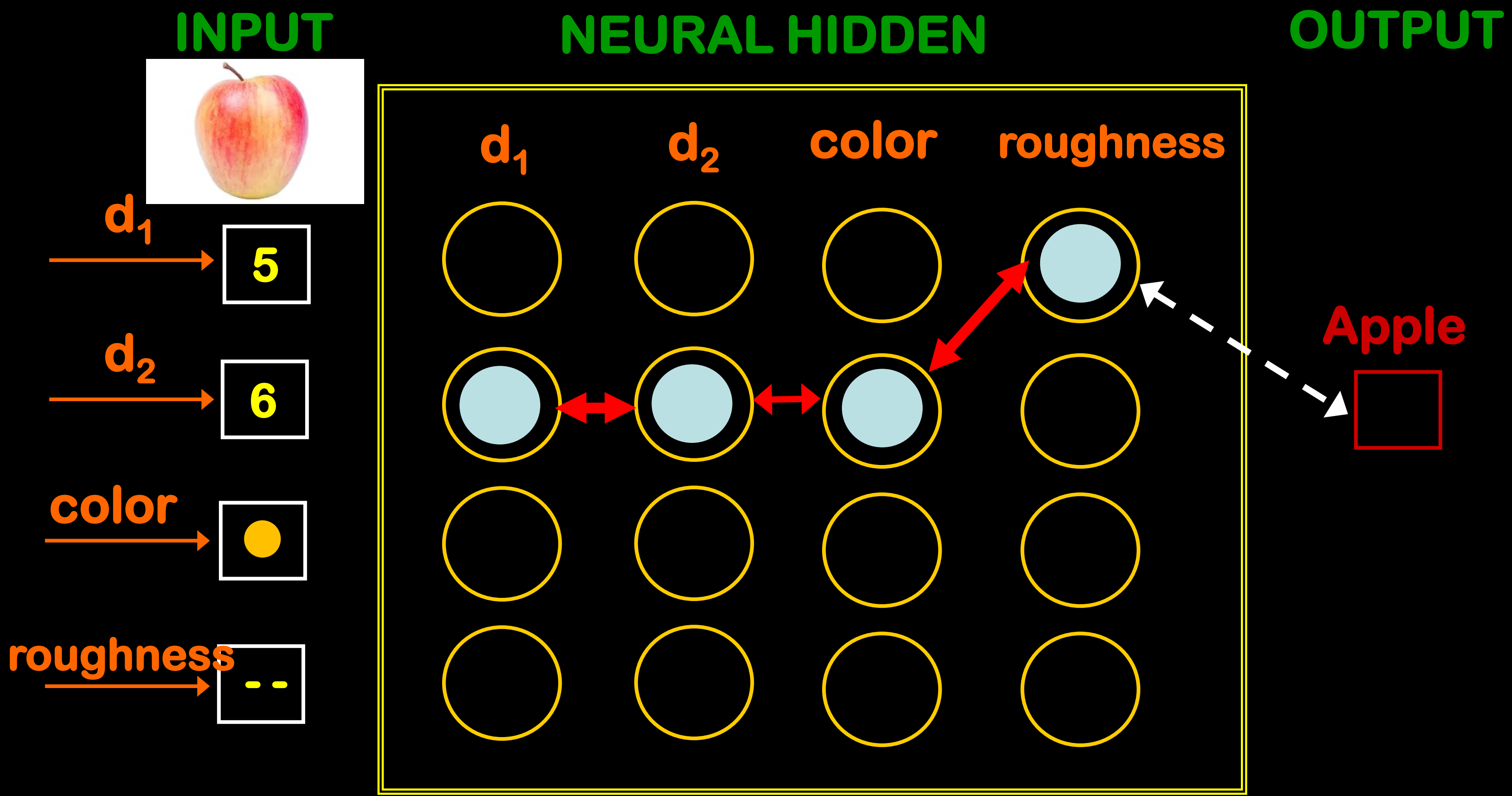
Artificial Neural Networks (ANN):

Phase 1 – Learning.

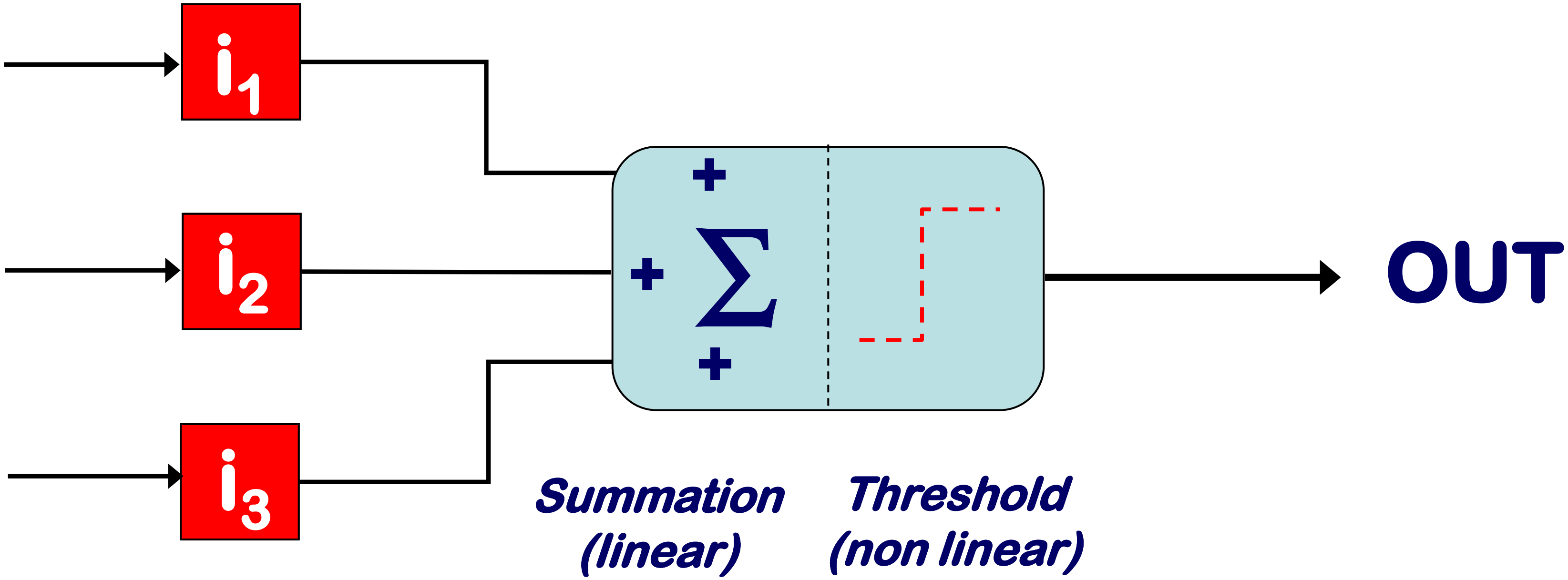


Artificial Neural Networks (ANN):

Phase 2 – Recognition.



Back to our *percepton…*



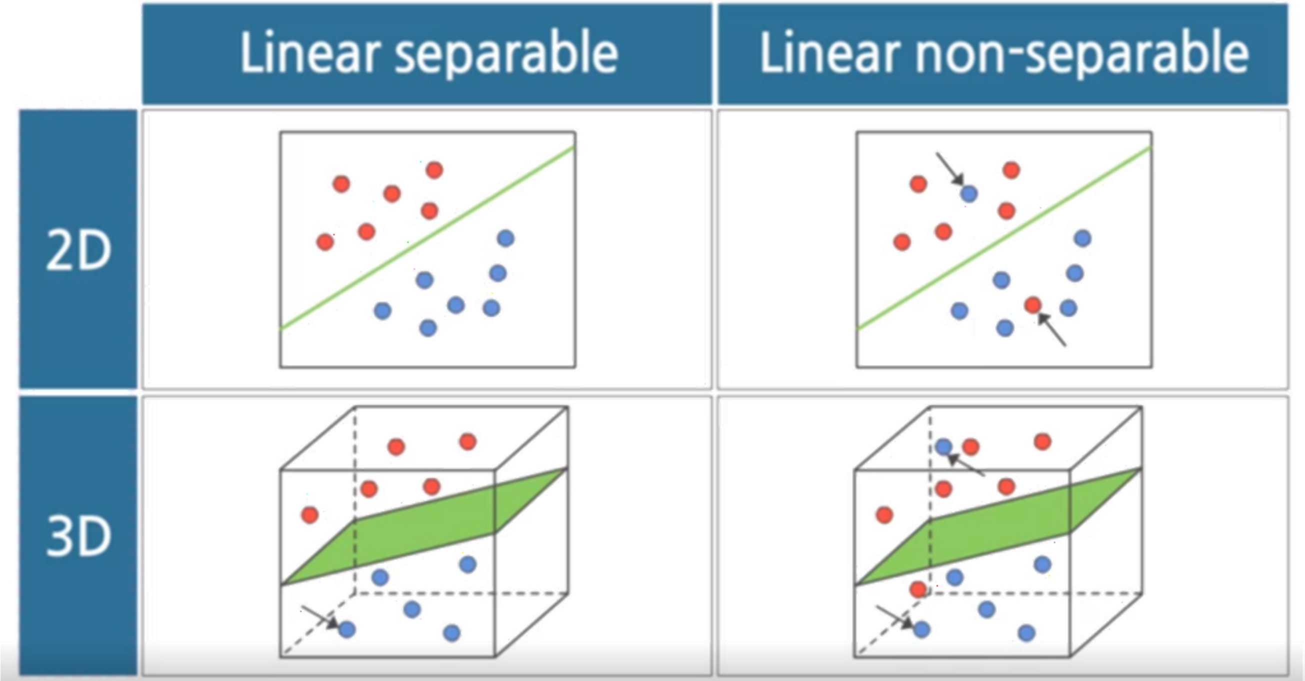
Introduction to the simple ANN - the *linear perceptron* :

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*:

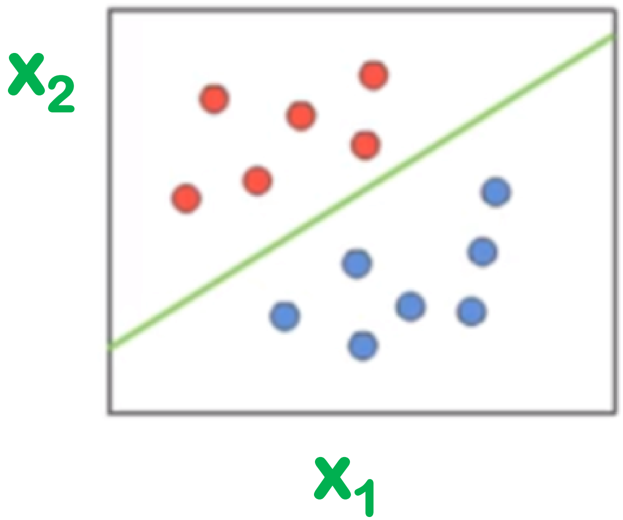
Before starting: understanding whether a space is ***linearly separable*** **means establishing whether there exists a straight line (in 2D), a plane (in 3D) or more generally a hyperplane that can separate two sets of points without overlapping. As un example:**



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, $w \cdot x + b < 0$



Remember this equation:

$$y = mx + q$$

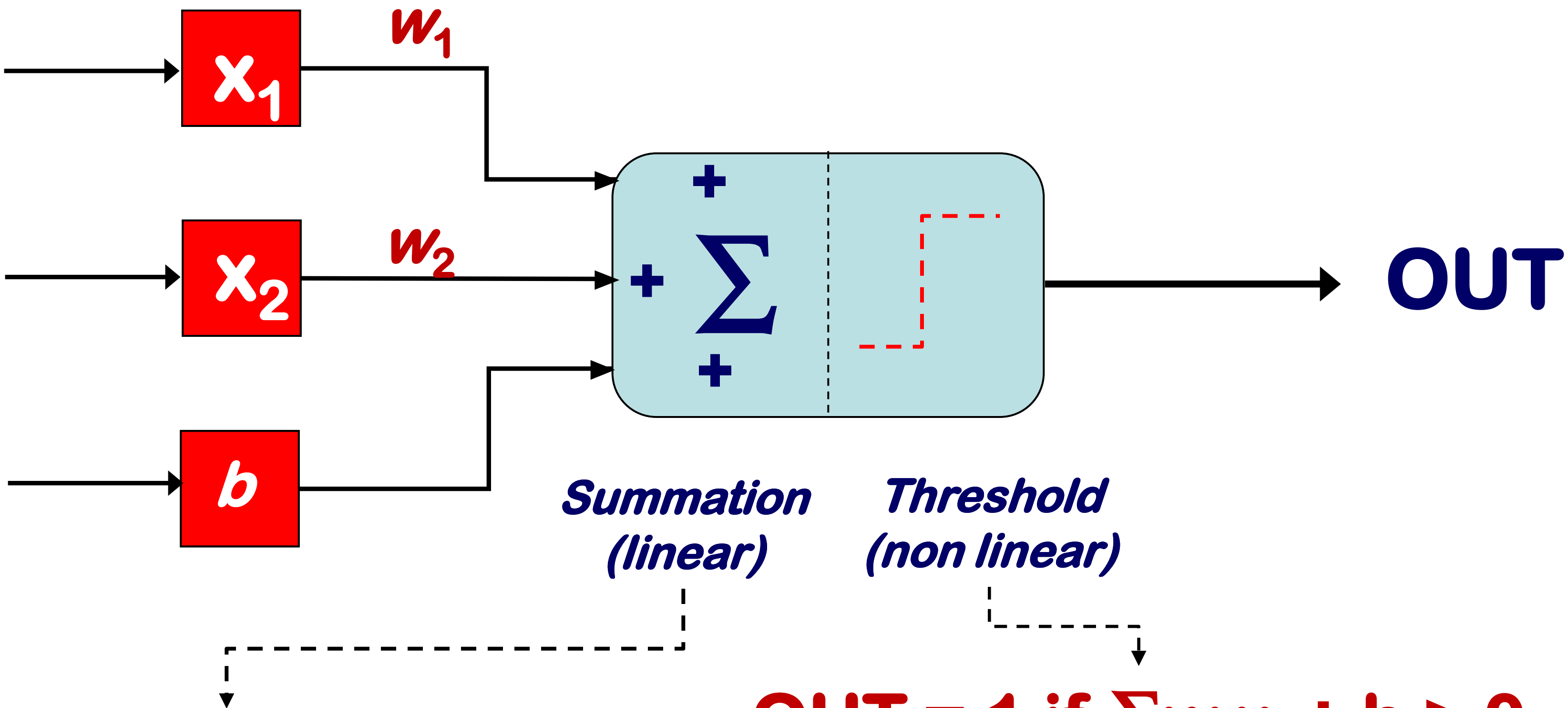
It is equivalent to:

$$x_2 = wx_1 + b^*$$

* 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.

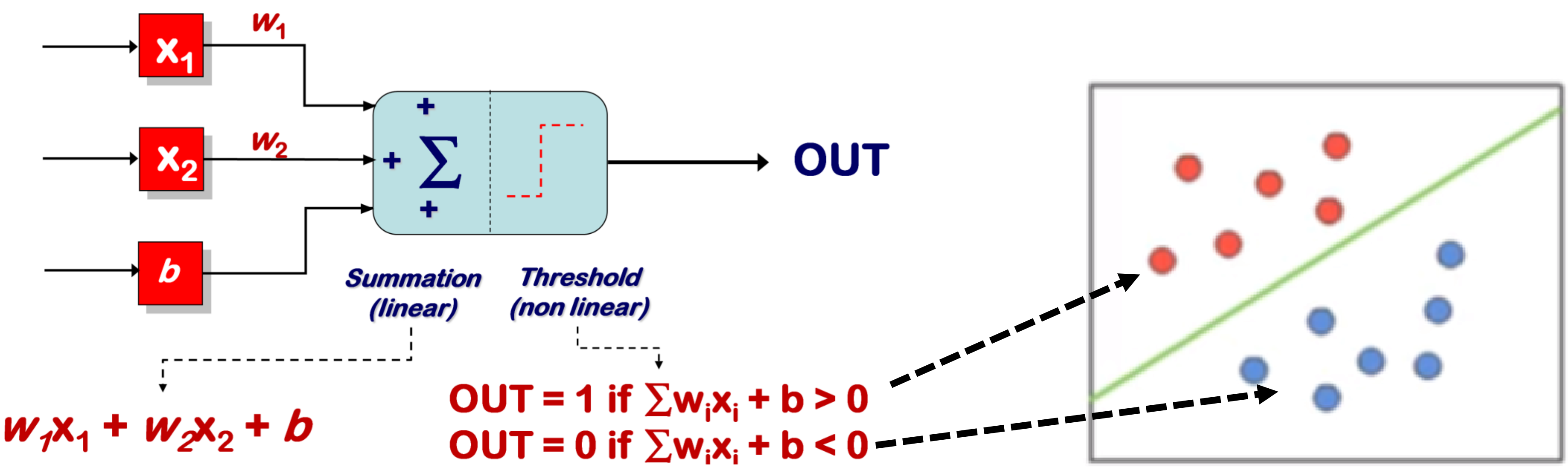


$$w_1x_1 + w_2x_2 + b$$

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

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:

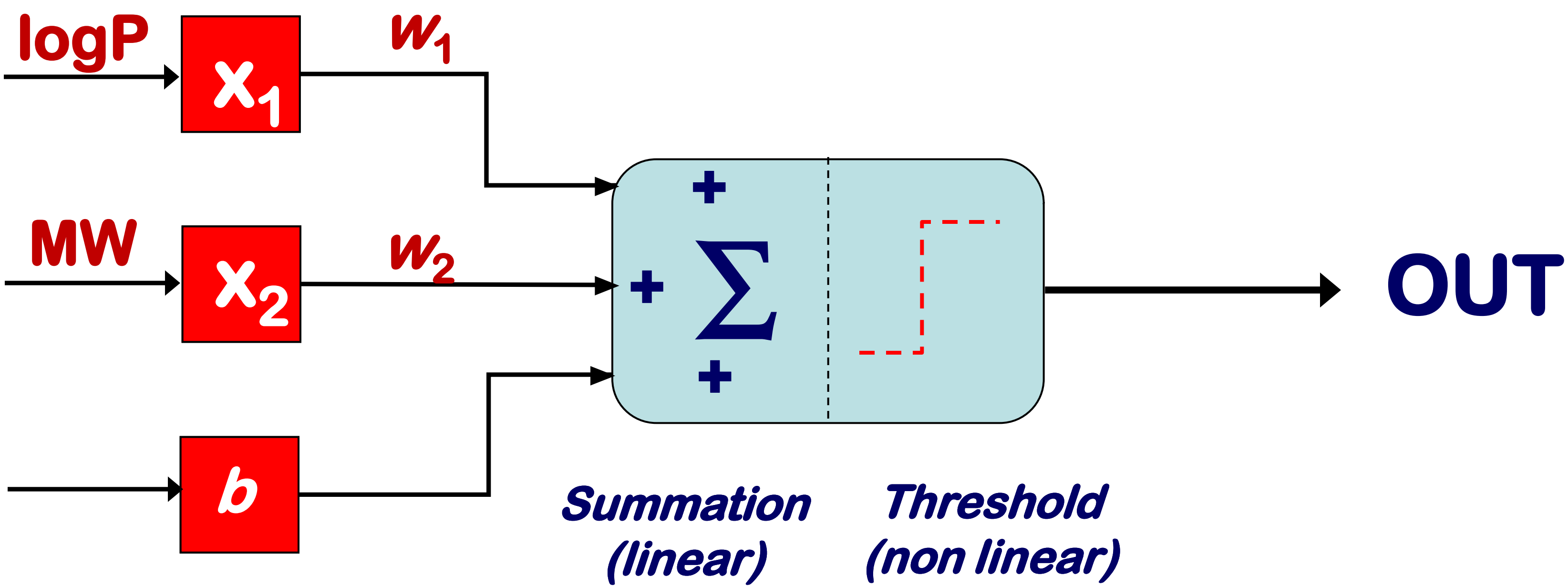
x_1 = logP (lipophilicity)
x_2 = MW (molecular weight)

And an associated label:

OUT = 1 if the molecule is *active*
OUT = 0 if it is *inactive*

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

Here is our simple linear perceptron:



$$w_1 x_1 + w_2 x_2 + b$$

$$OUT = A \;\; \text{if} \; \sum w_i x_i + b > 0$$
$$OUT = NA \;\; \text{if} \; \sum w_i x_i + b < 0$$

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

The model of our linear perceptron has the form:

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

where:

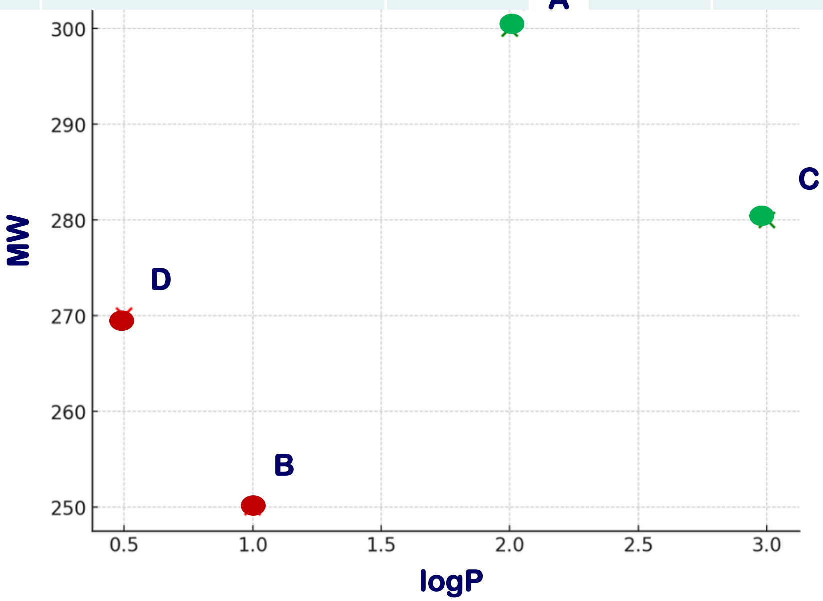
w_1 and w_2 are the weights and b is the bias (threshold) to learn;

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:

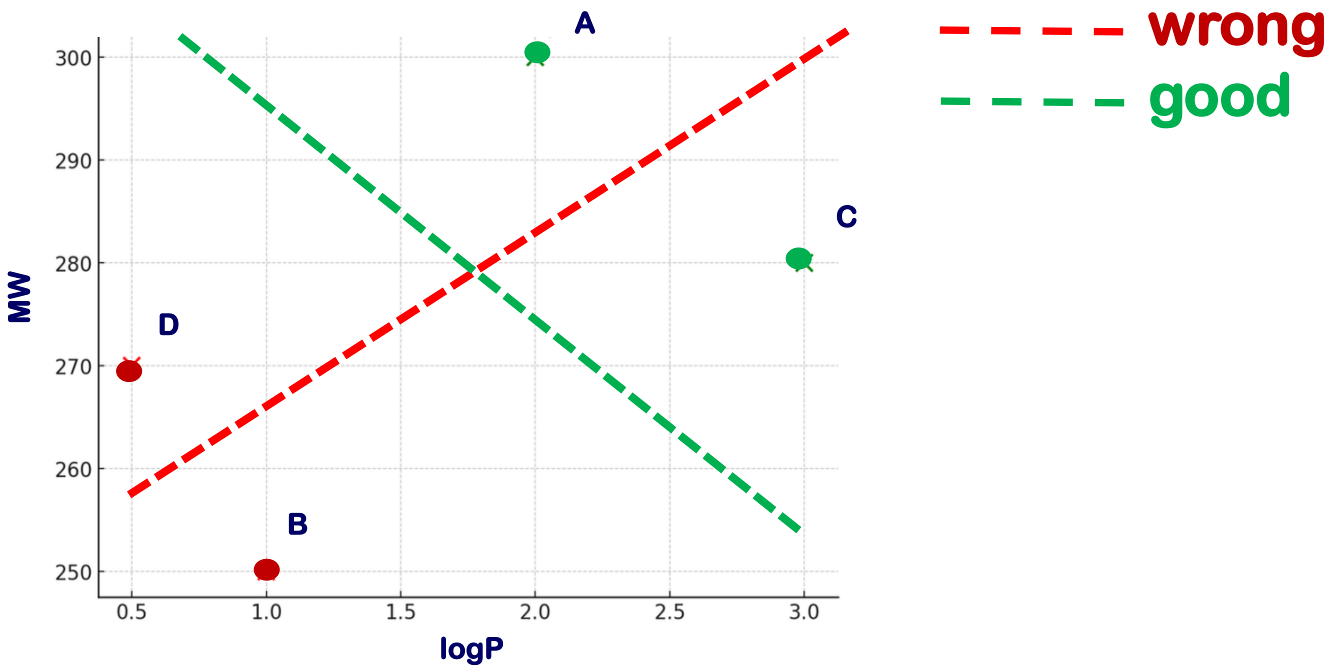
Consider this four drug candidates:

#	logP (x_1)	MW (x_2)	Activity (OUT)
A	2.0	300	A
B	1.0	250	NA
C	3.0	280	A
D	[illegible]	[illegible]	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:

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 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?

$$Logloss_i = -[y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)]$$

where *i* is the given observation/record,

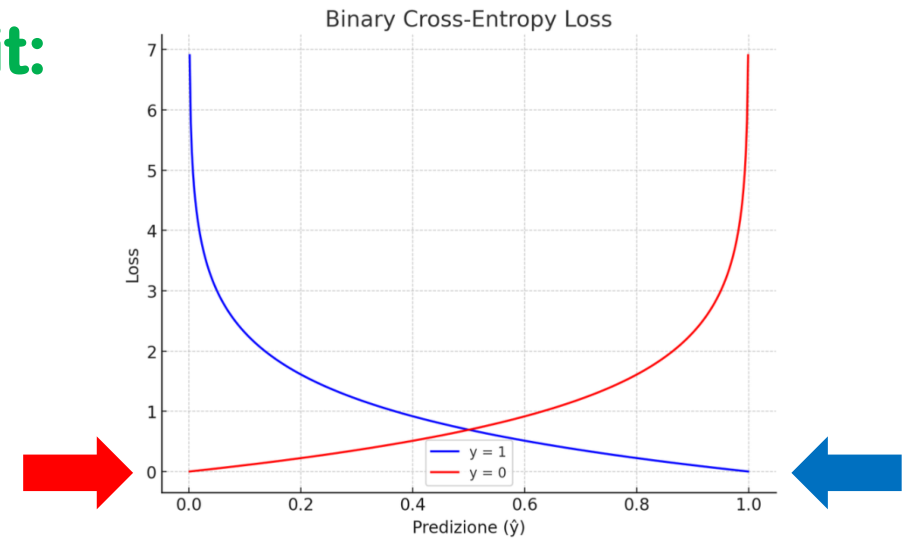
***y* is the actual/true value,**
***p* is the prediction probability,**
and *ln* refers to the natural logarithm (logarithmic value using base of *e*, *e*≈2.718) of a number.

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

How is log-loss value calculated?

$$Logloss_i = -[y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)]$$

Let's interpret it:



If the model predicts well (p≈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?

$$Logloss_i = -[y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)]$$

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.*

$$Logloss = \frac{1}{N}\sum_{i=1}^{N} logloss_i$$

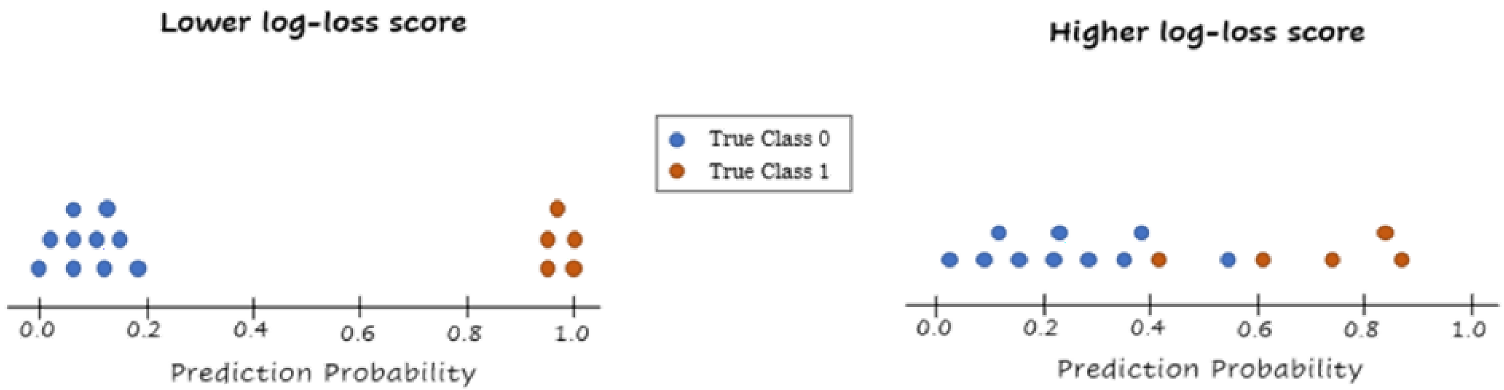
$$Logloss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)]$$

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?

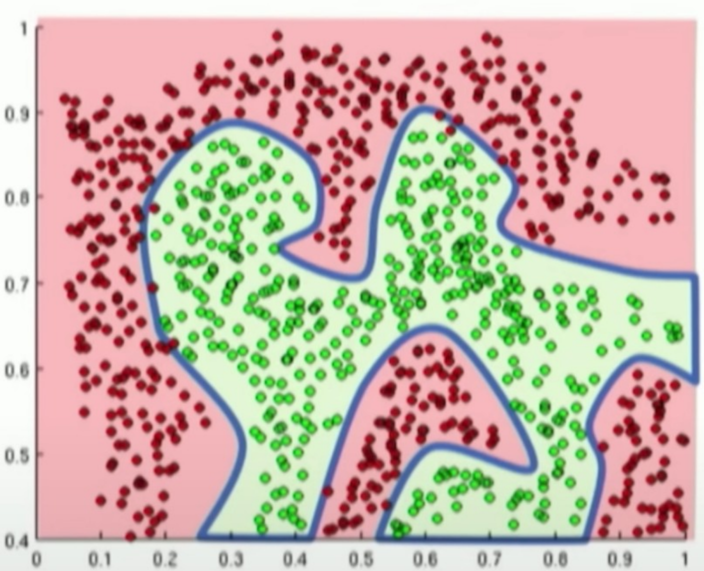
A model with perfect skill has a log-loss score of 0. In other words, the model predicts each observation's probability as the actual value.



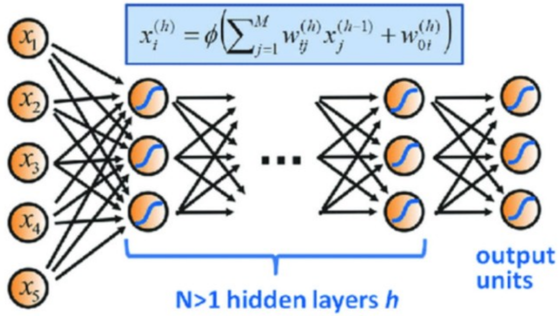
Note: A model with lower log-loss score is better than the one with higher log-loss score, provided both the models are applied to the same distribution of dataset. *We cannot compare log-loss scores of two models applied on two different datasets.*

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

A Multilayer Perceptron (MLP) is one of the simplest and most common neural network architectures used in machine learning. It is a feedforward artificial neural network consisting of multiple layers of interconnected neurons, including an **input layer**, one or more **hidden layers**, and an **output layer**.
MLPs are capable of learning complex and non-linear relationships in data (as shown in pic below), especially when they have multiple hidden layers and non-linear activation functions.



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



Depth and Width of Hidden Layers

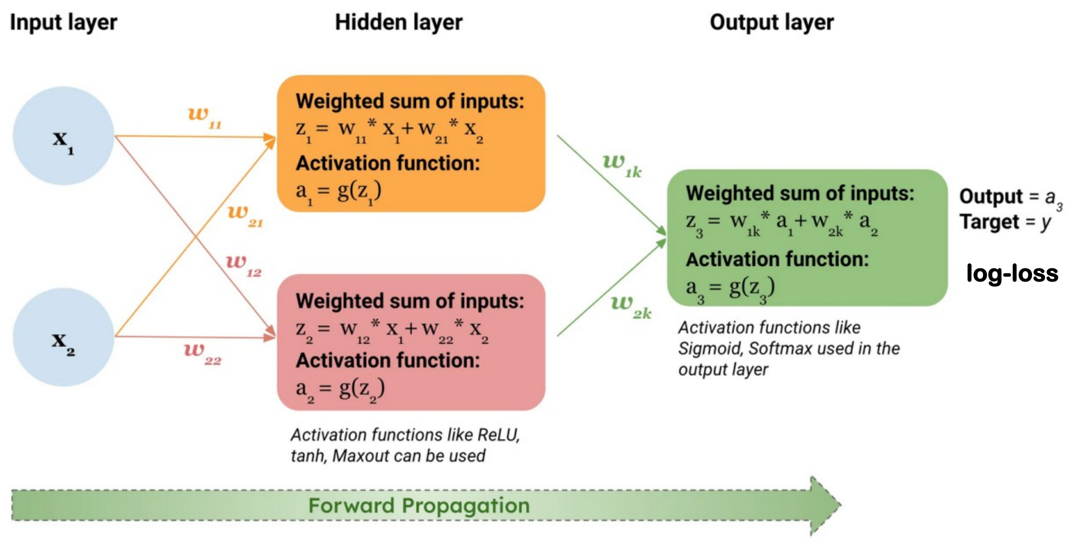
The number of hidden layers and the number of neurons in each hidden layer define the architecture of a neural network.

The depth of a network refers to the number of hidden layers it contains, while the width refers to the number of neurons in each hidden layer.

Deeper networks with more hidden layers can learn more complex representations, but they also require more data and computational power to train. Conversely, wider networks with more neurons can capture more information about the input data but may also lead to overfitting if not managed properly.

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

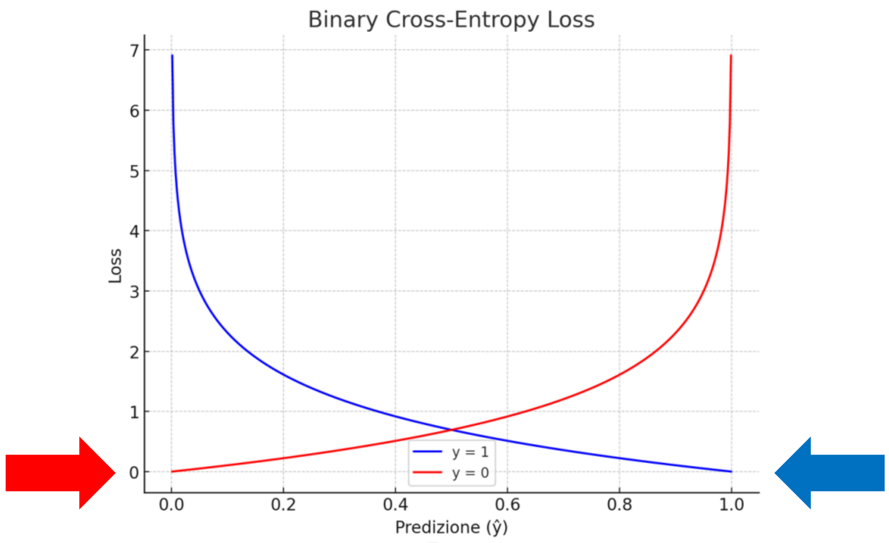
The process from the input layer through the hidden layers to the output layer is called forward propagation. In each layer, the aforementioned steps (weighted sum, bias addition, activation function) are applied to compute the layer's output. In an MLP, information flows in one direction, from the input 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

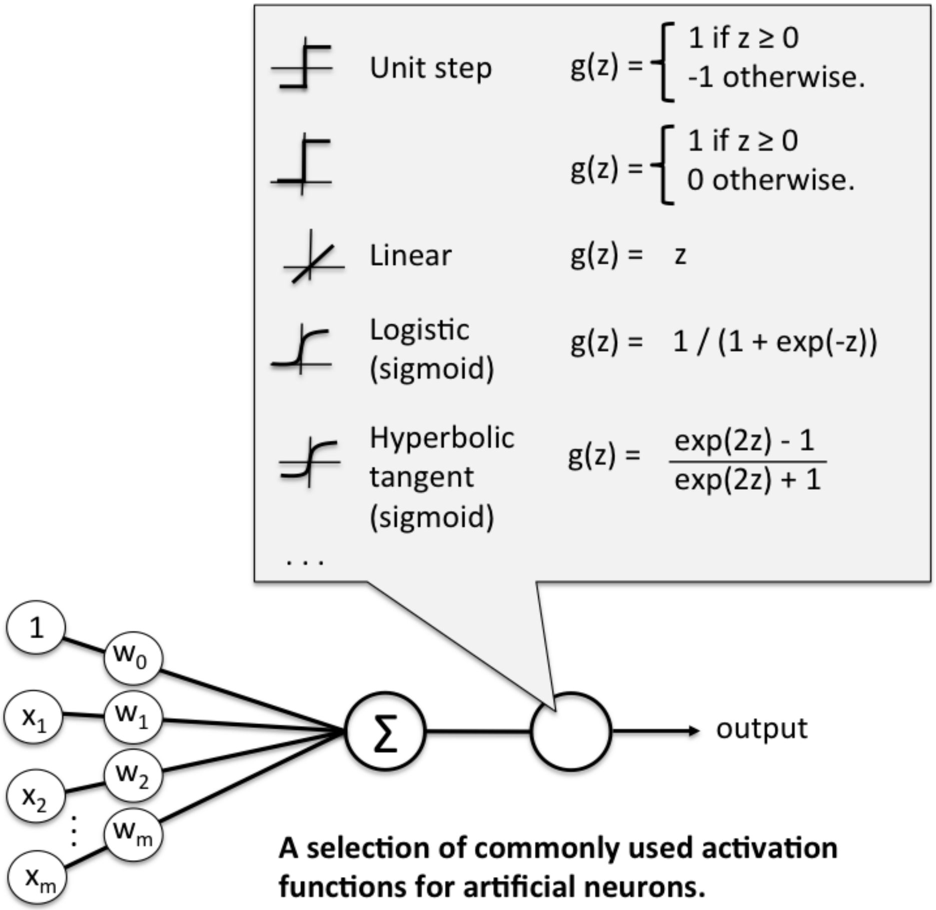
Remember the loss function and in particulat the log-loss?

$$Logloss_i = -[y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)]$$



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



A selection of commonly used activation functions for artificial neurons.

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:

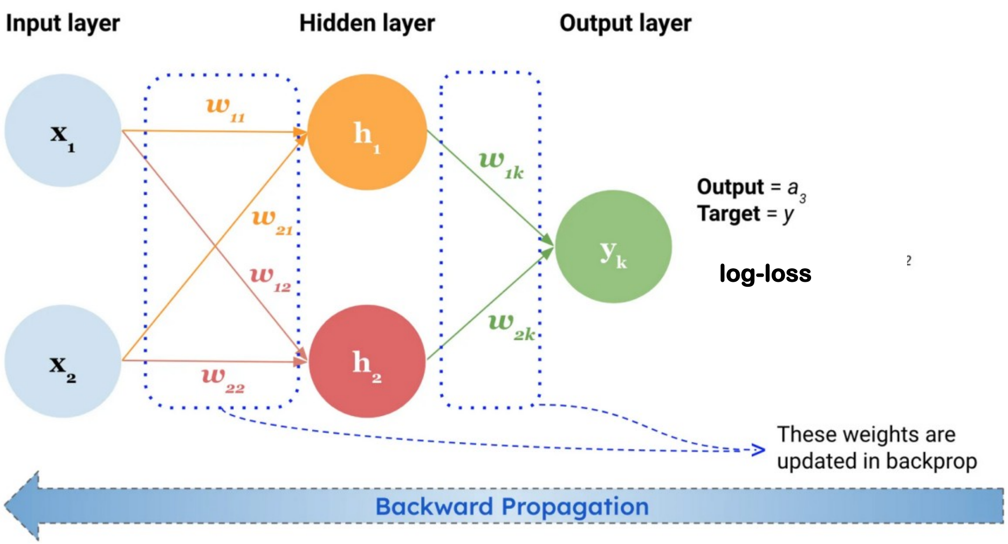
Non-linear: This is required to introduce non-linearity in the model.

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:

- Classication 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

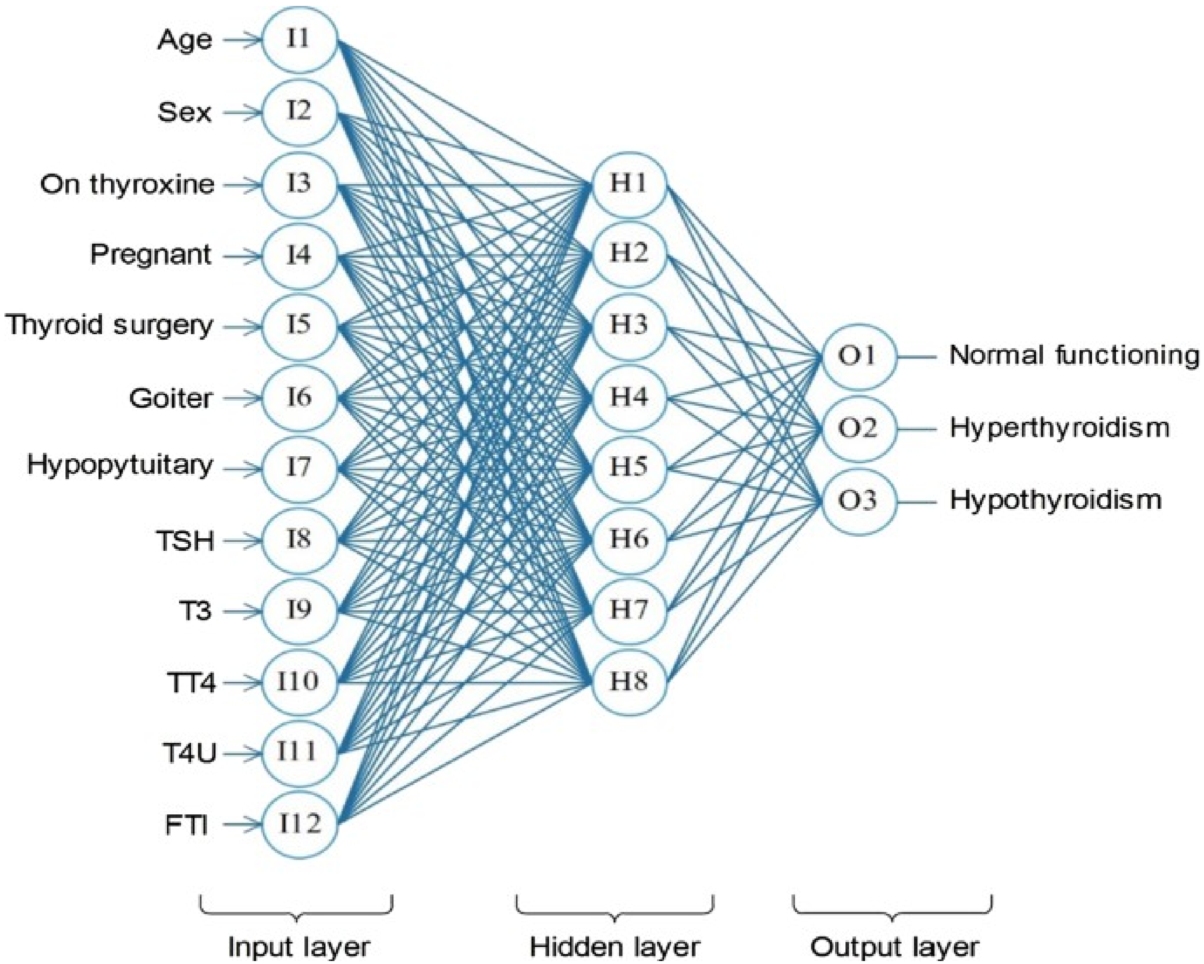
Very deep models (with GNN/Transformer) >6 (but only if necessary!)

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

Recommended neurons per layer

Task type	*Recommended maximum neurons (per layer)*
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)

https://www.youtube.com/watch?v=eMlx5fFNoYc

credits: https://botpenguin.com/glossary/transformers

https://www.aibutsimple.com/p/transformers-and-the-attention-mechanism

https://www.youtube.com/watch?v=IHZwWFHWa-w