CHAPTER IV

Machine Learning & Classification

Definition

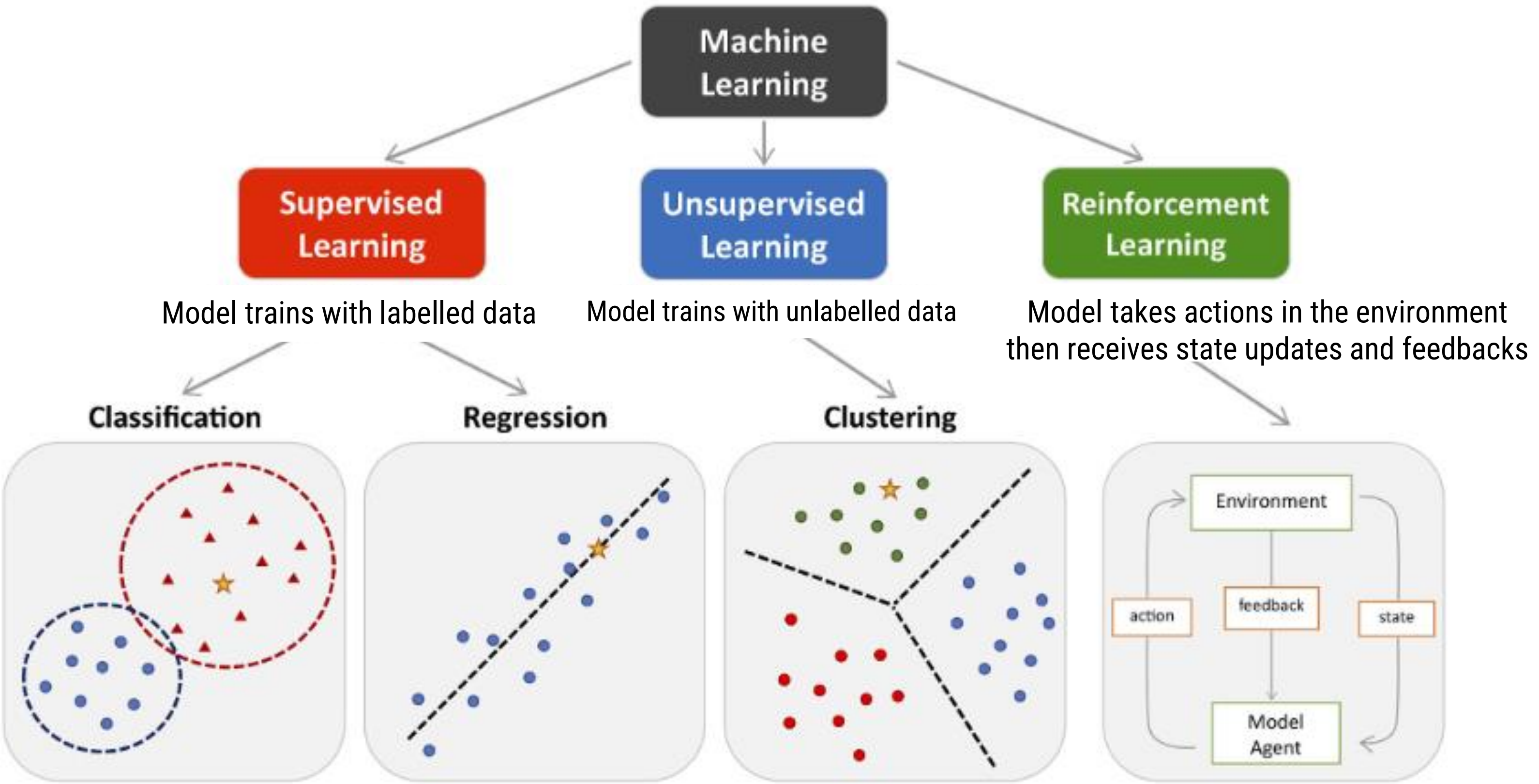
« Field of study that gives computers the ability to learn without being explicitly programmed »
Arthur Samuel, 1959

- An agent **learns** if it improves its performance on future tasks with **experience.**

- Machine learning refers to the development, analysis and implementation of methods that allow a machine to evolve through a **learning process**.

Types of Machine Learning

Learning algorithms can be categorized according to the type of learning they use:



Types of Machine Learning

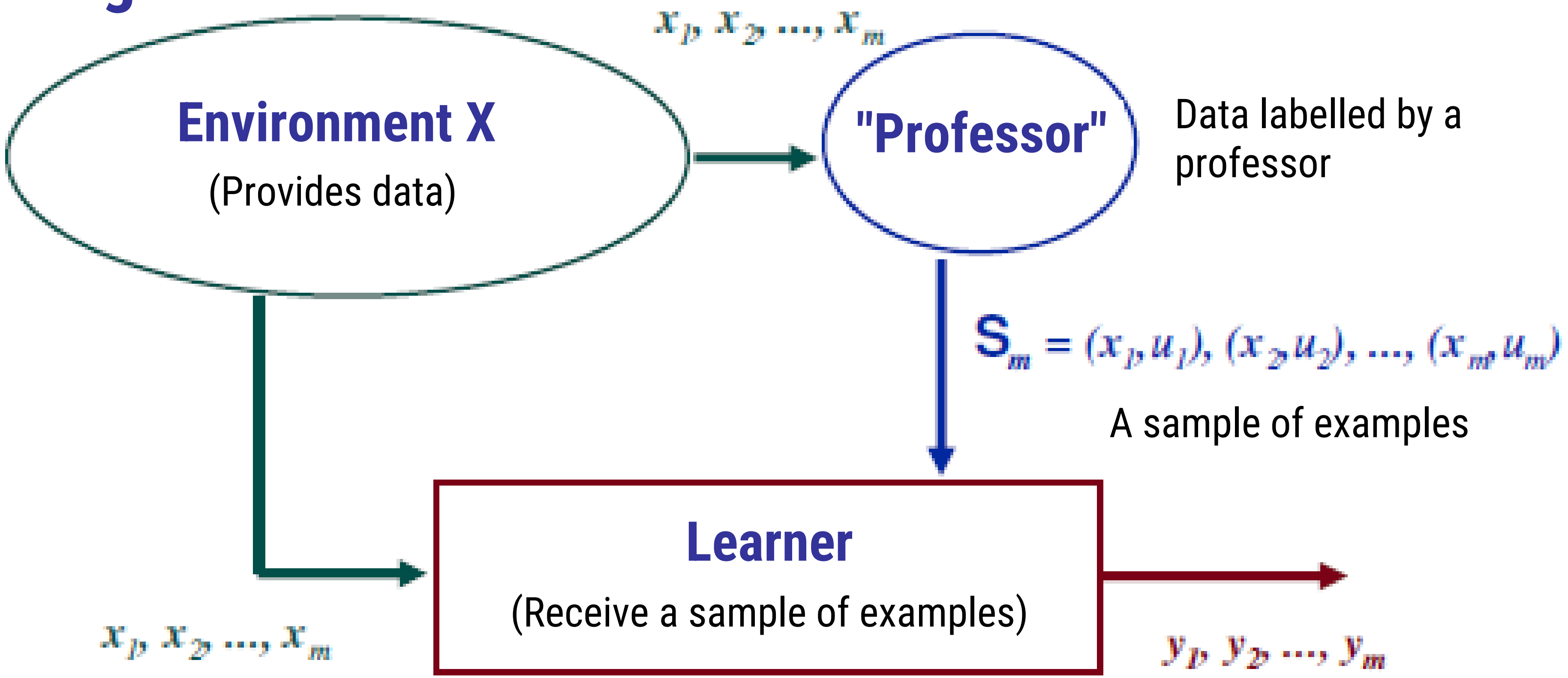
Supervised Learning

- An **expert** is employed to correctly **label** examples.
- The **learner** must then **find** or approximate the function which allows the **correct label** to be assigned to these examples.

- **Example:** character recognition using a set of pairs: (image, character identity)

Types of Machine Learning

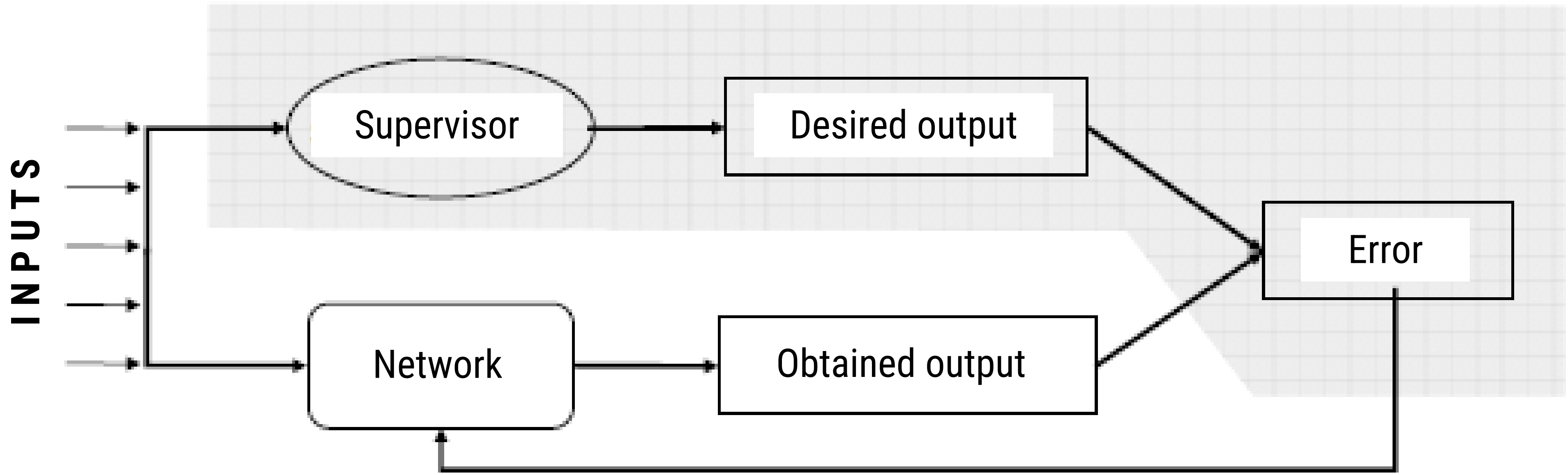
Supervised Learning



Approximate as better as possible the desired output for each observed input

Types of Machine Learning

Supervised Learning

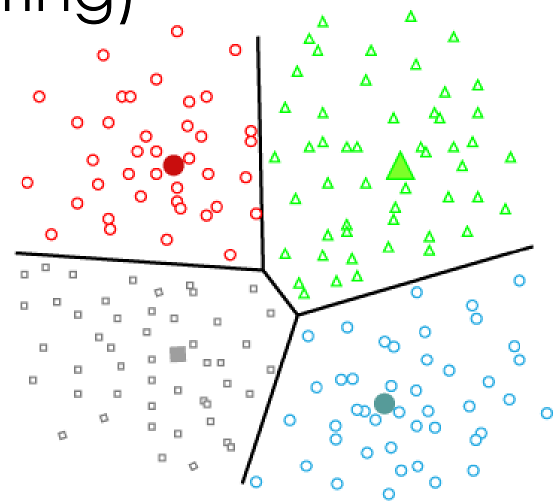


Types of Machine Learning

Unsupervised Learning

- No expert is available.
- The algorithm must discover the data structure itself .

- **Example:** identify different topics of news articles by grouping similar articles together (clustering)

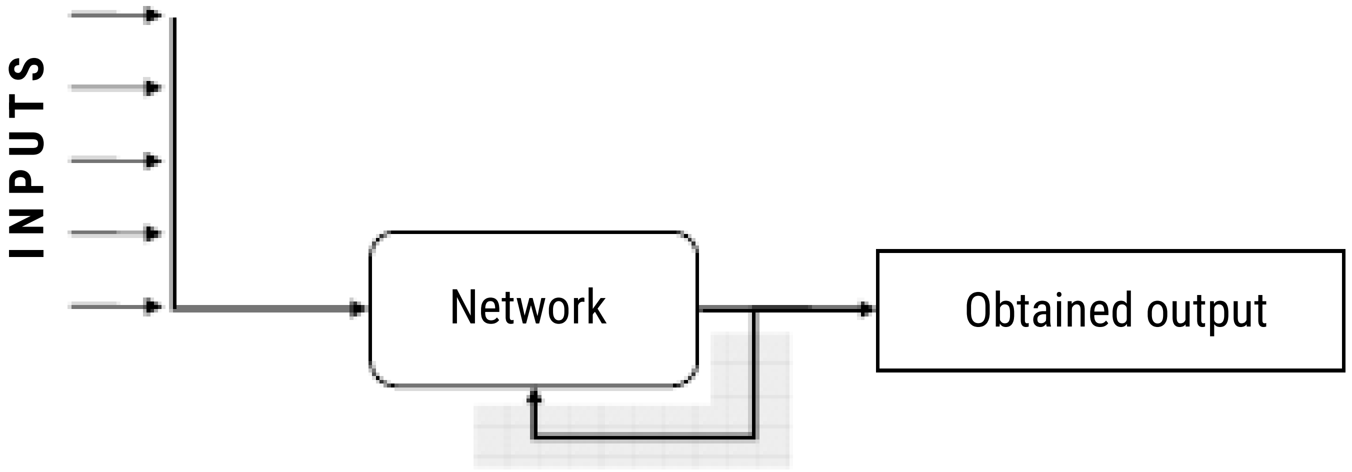


Types of Machine Learning

Unsupervised Learning

From the unlabelled training sample $S = \{(x_i)\}_{1,m}$ We look for underlying regularities:

- In the form of a function
- In the form of complex model



Types of Machine Learning

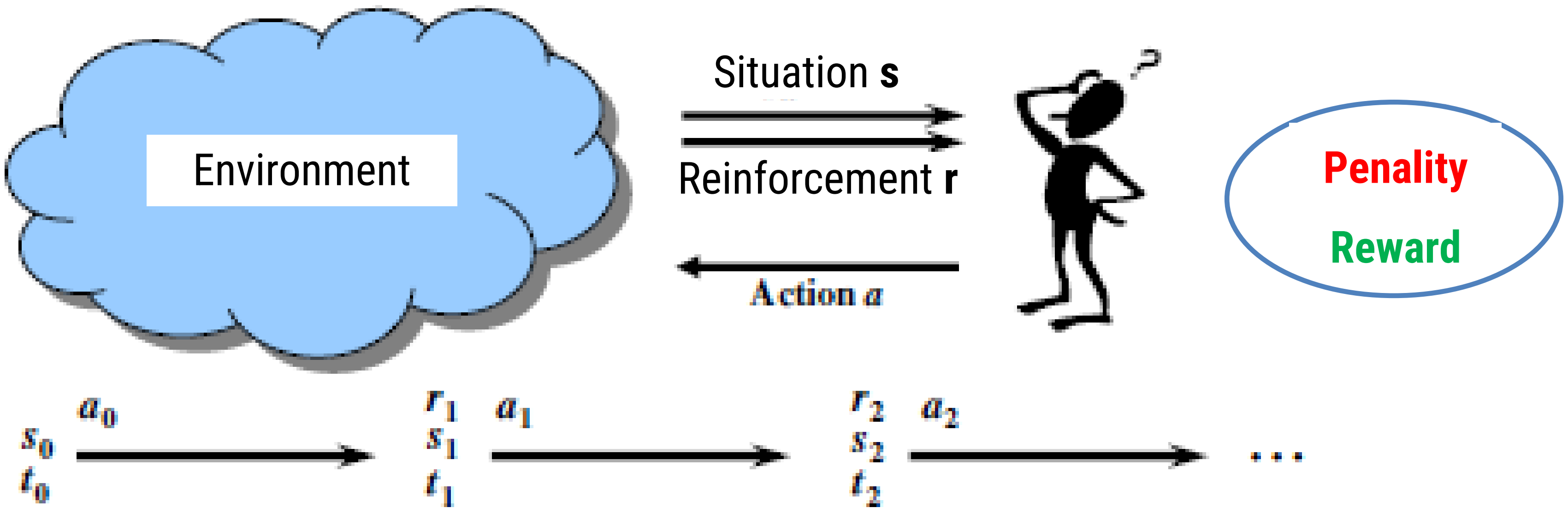
Reinforcement Learning

- The algorithm learns a behavior given an observation.
- The action of the algorithm on the environment produces feedback value that guides the learning algorithm.

- Example: giving rewards for a model who plays chess:
  - +1 if the model wins a game
  - -1 if the model loses a game

Types of Machine Learning

Reinforcement Learning



- The agent learns to approximate an **optimal behavioral strategy** through repetitive interactions with the environment
- Decisions are made sequentially at discrete time intervals

Algorithms

- K-Nearest Neighbors (K-NN)
- Artificial Neural Networks
- Bayesian methods
- Hidden Markov Models (HMM)
- Genetic algorithms
- Decision trees
- Random forests
- Support Vector Machine (SVM)
- Linear regression
- …

K-Nearest Neighbors (K-NN)

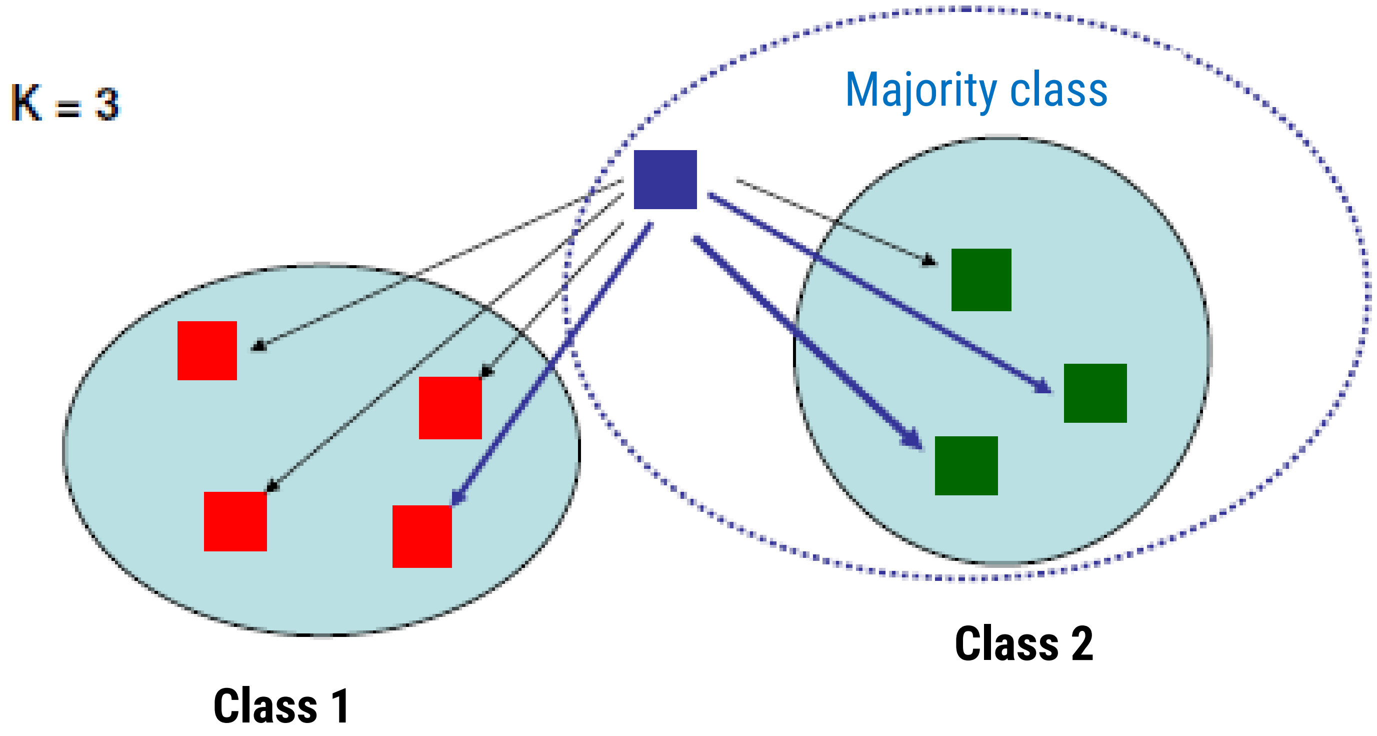
Principle

- We have a **training database** made up of **m** “input-output” pairs.

- In order to estimate the output associated with a new input **x**, the method consists of taking into account the **k** training samples whose input is closest to the new input **x**, according to a distance to be defined.

K-Nearest Neighbors (K-NN)

We will retain the most represented class among the **k** outputs associated with the **k** inputs closest to the new input **x**.



K-Nearest Neighbors (K-NN)

Example : Character recognition

e or o ?

- Training sample (100 leaning examples per class)

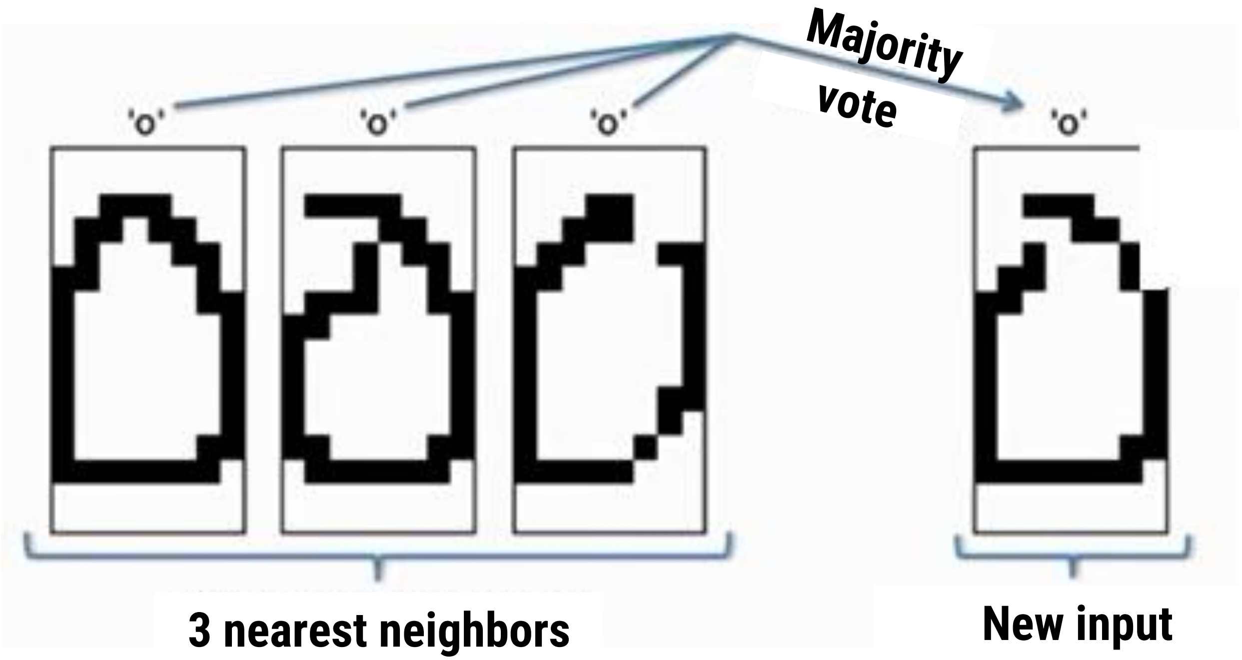
Class: e

Class: o

K-Nearest Neighbors (K-NN)

Example : Character recognition

e or o ?

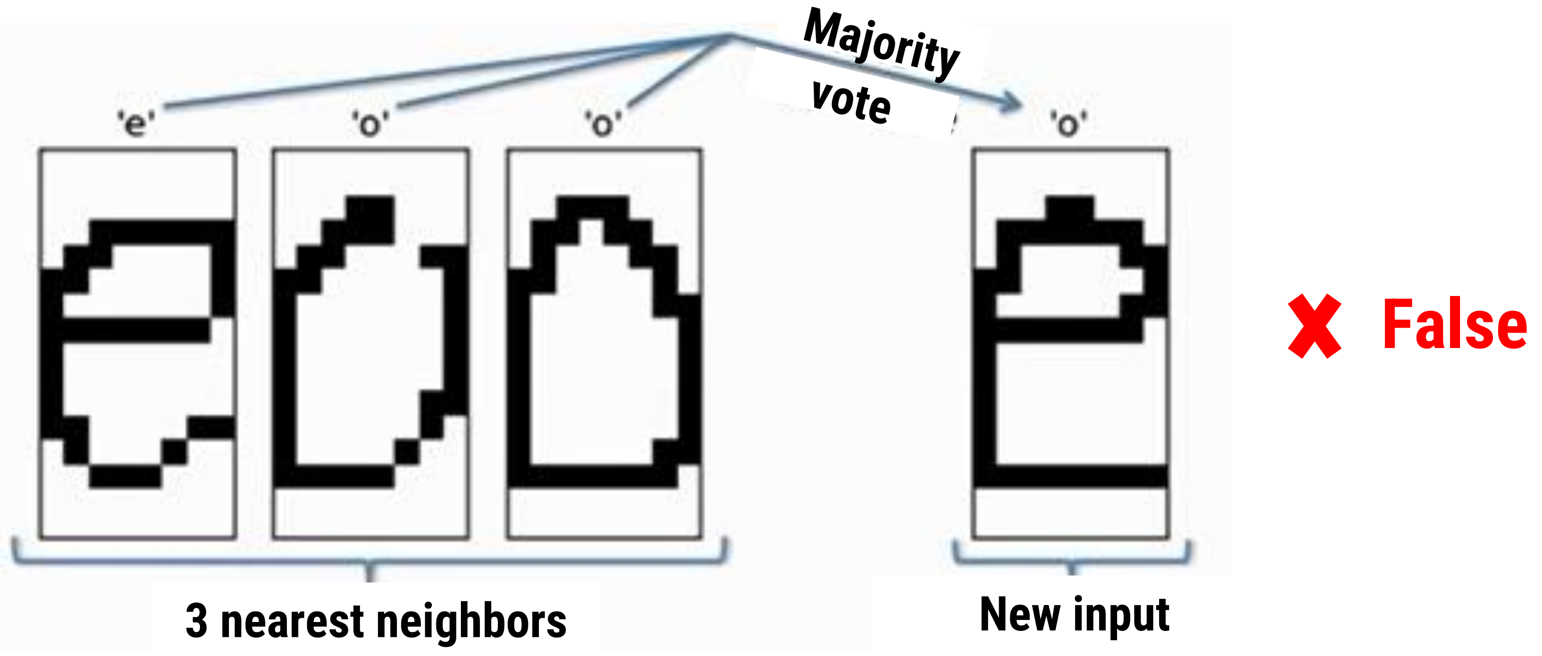


✓ True

K-Nearest Neighbors (K-NN)

Example : Character recognition

e or o ?

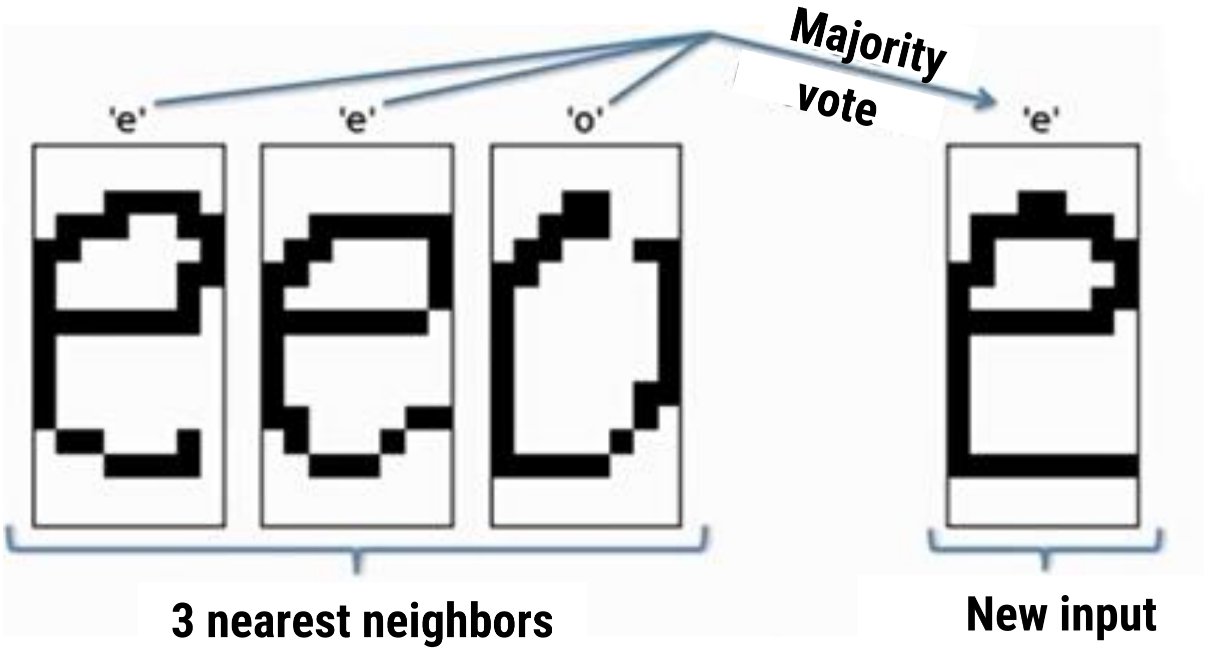


K-Nearest Neighbors (K-NN)

Example : Character recognition

e or o ?

If we add 200 examples per class



✓ True

K-Nearest Neighbors (K-NN)

Algorithm

- **Parameter :** A number **K** of neighbors
- **Data :** a sample of **m** examples and their **classes**
  - The **class** of an example **X** is **c(X)**
- **Input :** a record **Y**
- Determine the **k** closest examples to **Y** by calculating distances
- Combine the classes of these **k** examples into a class **c**
- **Output :** the class of **Y** is **c(Y)=c**

K-Nearest Neighbors (K-NN)

Distance

- The choice of distance is essential to the proper functioning of this method
- The basic distances allow to obtain satisfactory results
- **Distance properties:**
  - $d(A,A) = 0$
  - $d(A,B) = d(B,A)$
  - $d(A,B) \leq d(A,C) + d(B,C)$

K-Nearest Neighbors (K-NN)

Distance calculation

- $d(x,y) = |x-y|$
- $d(x,y) = |x-y|/d_{max}$, where d_{max} is the maximum distance between two numbers in the considered domain

Examples of distances

- Binary data : 0 or 1.

  We consider d(0,0)=d(1,1)=0 and d(0,1)=d(1,0)=1.

- Enumerative data :

  The distance is 0 if the values are equal and 1 otherwise.

- Ordered enumerative data : they can be considered as enumerative values but we can also define a distance using the order relation.

  - **Example:** If a field takes the values A, B, C, D and E, we can define the distance by considering 5 points of the interval [0,1] with a distance of 0.25 between two successive points, we then have d (A,B)=0.25; d(A,C)=0.5; ...

K-Nearest Neighbors (K-NN)

Euclidian distance

Consider **$X = (x_1, \ldots, x_n)$** and **$Y = (y_1, \ldots, y_n)$** two examples, the **euclidian distance** between **X** and **Y** is:

$$D(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

K-Nearest Neighbors (K-NN)

Exercise 1 (3 Nearest Neighbors)

Customer	Age	Income	Fidelity
Ahmed	35	35k	No
Khadidja	22	50k	Yes
Fatima	63	200k	No
Abdellah	59	170k	No
Safia	25	40k	Yes
Abderrahmane	37	50k	?

Determine the class of Abderrahmane (Loyal or not) **?**

K-Nearest Neighbors (K-NN)

Exercise 1 (3 Nearest Neighbors)

Customer	Age	Income	Loyal	Distance with Abderrahmane
Ahmed	35	35k	No	D(Abderrahmane,Ahmed)=Sqrt[$(35-37)^2+(35-50)^2$]=15.13
Khadidja	22	50k	Yes	D(Abderrahmane,Khadidja)=Sqrt[$(22-37)^2+(50-50)^2$]=15
Fatima	63	200k	No	D(Abderrahmane,Fatima)=Sqrt[$(63-37)^2+(200-50)^2$]=152.23
Abdellah	59	170k	No	D(Abderrahmane,Abdellah)=Sqrt[$(59-37)^2+(170-50)^2$]=122
Safia	25	40k	Yes	D(Abderrahmane,Safia)=Sqrt[$(25-37)^2+(40-50)^2$]=15.62
Abderrahmane	37	50m	Yes	Majority Class

K-Nearest Neighbors (K-NN)

Exercise 2 (3 Nearest Neighbors)

We consider a training database made up of 5 « input-output » pairs:

(Abdellah, Succeeded), (Ahmed, Succeeded), (Khaled, Deferred), (Salim, Deferred) et (Salah, Succeeded).

For each students, We have 4 grades in 4 different subjects:

- Abdellah :14, 12, 8,12.
- Ahmed :12, 12, 6, 10.
- Khaled : 8, 9, 9, 1.
- Salim : 15, 11, 3, 5.
- Salah : 12, 9, 14, 11.

We now have a new entry "Karim" Who has the following grades: 9,14,15 and 6.

By using the k nearest neighbors method (k =3) and choosing the Euclidean distance, Determine the class of Karim**?**.

K-Nearest Neighbors (K-NN)

Exercise 1 (3 Nearest Neighbors)

Student	Grades	Class	Distances
Abdellah	14, 12, 8, 12	Succeed.	D(Abdellah,Karim)=SQRT[$(14-9)^2+(12-14)^2+(8-15)^2+(12-6)^2$] =SQRT[25+4+49+36]=SQRT(114)=10.67
Ahmed	12, 12, 6 et 10	Succeed.	D(Ahmed,Karim)=SQRT[$(12-9)^2+(12-14)^2+(6-15)^2+(10-6)^2$] =SQRT[9+4+81+16]=SQRT(110)=**10.48**
Khaled	8, 9, 9, 1	Deferred	D(Khaled,Karim)=SQRT[$(8-9)^2+(9-14)^2+(9-15)^2+(1-6)^2$] =SQRT[1+25+36+25]=SQRT(87)=**9.32**
Salim	15, 11, 3, 5	Deferred	D(Salim,Karim)=SQRT[$(15-9)^2+(11-14)^2+(3-15)^2+(5-6)^2$] =SQRT[36+9+144+1]=SQRT(190)=13.78
Salah	12, 9, 14, 11	Succeed.	D(Salah,Karim)=SQRT[$(12-9)^2+(9-14)^2+(14-15)^2+(11-6)^2$] =SQRT[9+25+1+25]=SQRT(60)=**7.74**
Karim	9, 14, 15, 6	**Succeed.**	

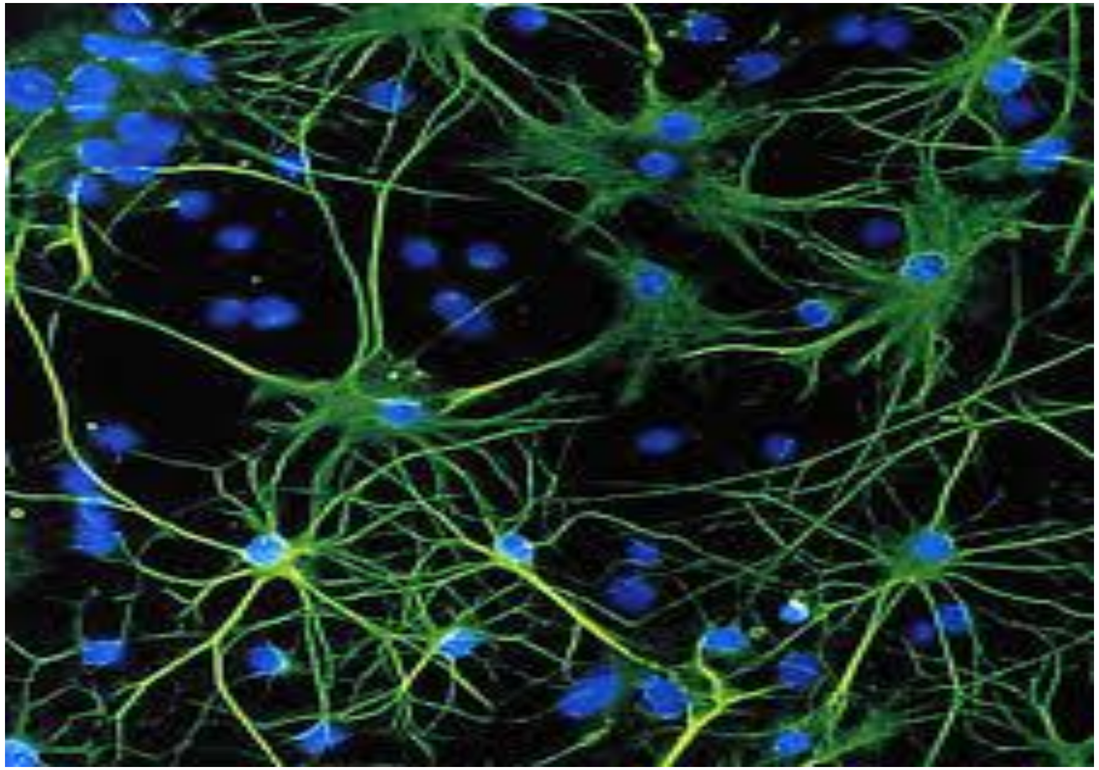
Artificial Neural Networks (ANN)

- **McCulloch et Pitts (1943) :** Birth of connectionism.
- **Rosenblatt (1957) :** First operational model (Perceptron).
  - Neural Network inspired by visual system
  - Learn some logical functions

Artificial Neural Networks (ANN)

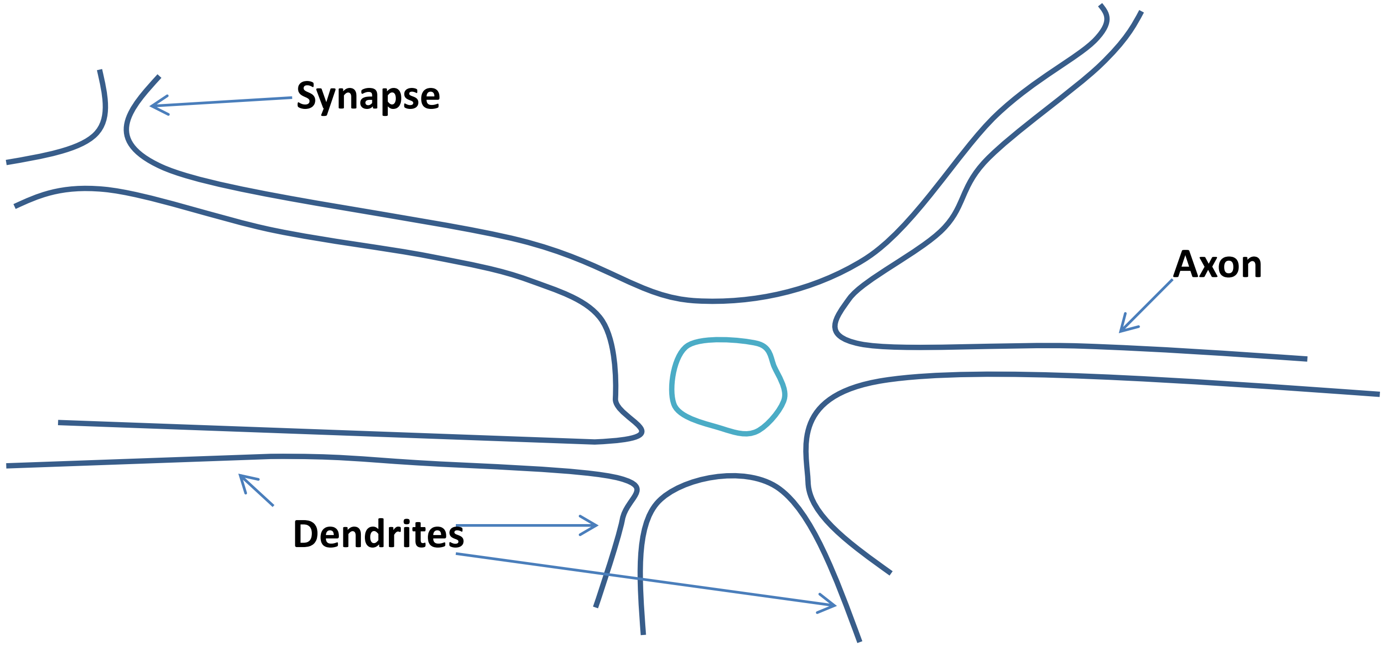
BIOLOGICAL FOUNDATIONS

- **BRAIN**
  - Control center of perception, decision and action.
  - 10^{13} neurons, each of them is connected to 1000 other neurons.



Artificial Neural Networks (ANN)

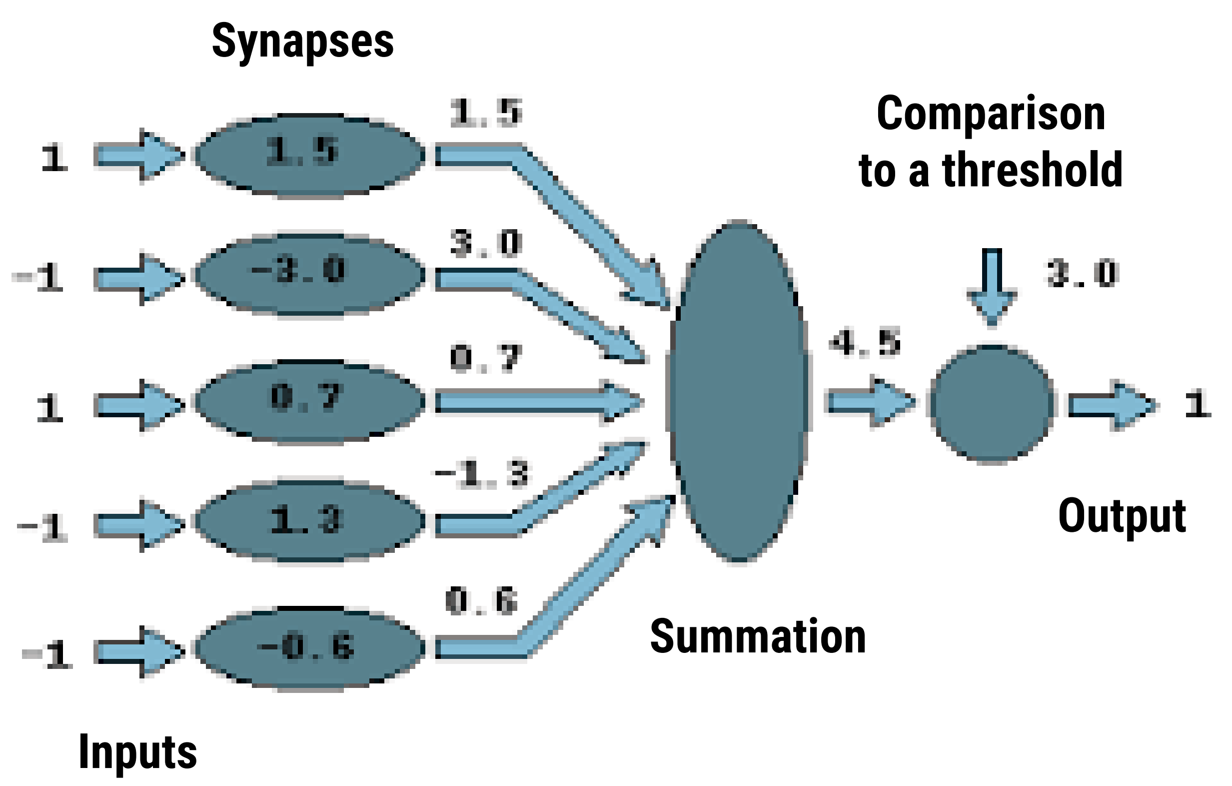
BIOLOGICAL NEURON



- The neuron receives impulses (information) from neighboring neurons via the **Dendrites**
- Perform a **summation** of these pulses
- Distribution of the calculated activity to neighboring neurons via the **Axon**
- **Synapse** : contact between nerve fibers, quantitative role in transmission

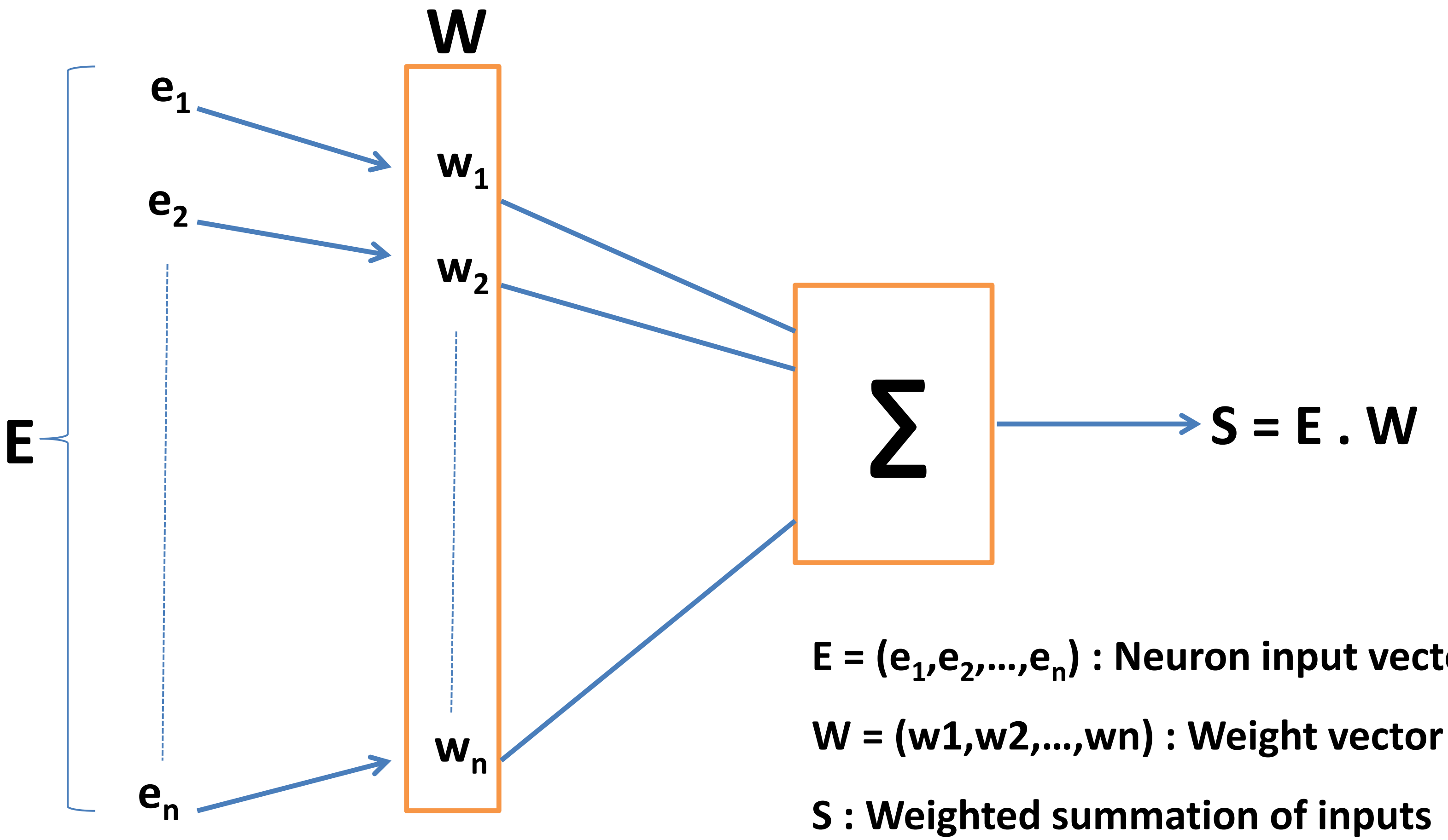
Artificial Neural Networks (ANN)

ARTIFICAL NEURON



Artificial Neural Networks (ANN)

FORMAL NEURON



E = (e_1,e_2,…,e_n) : Neuron input vector

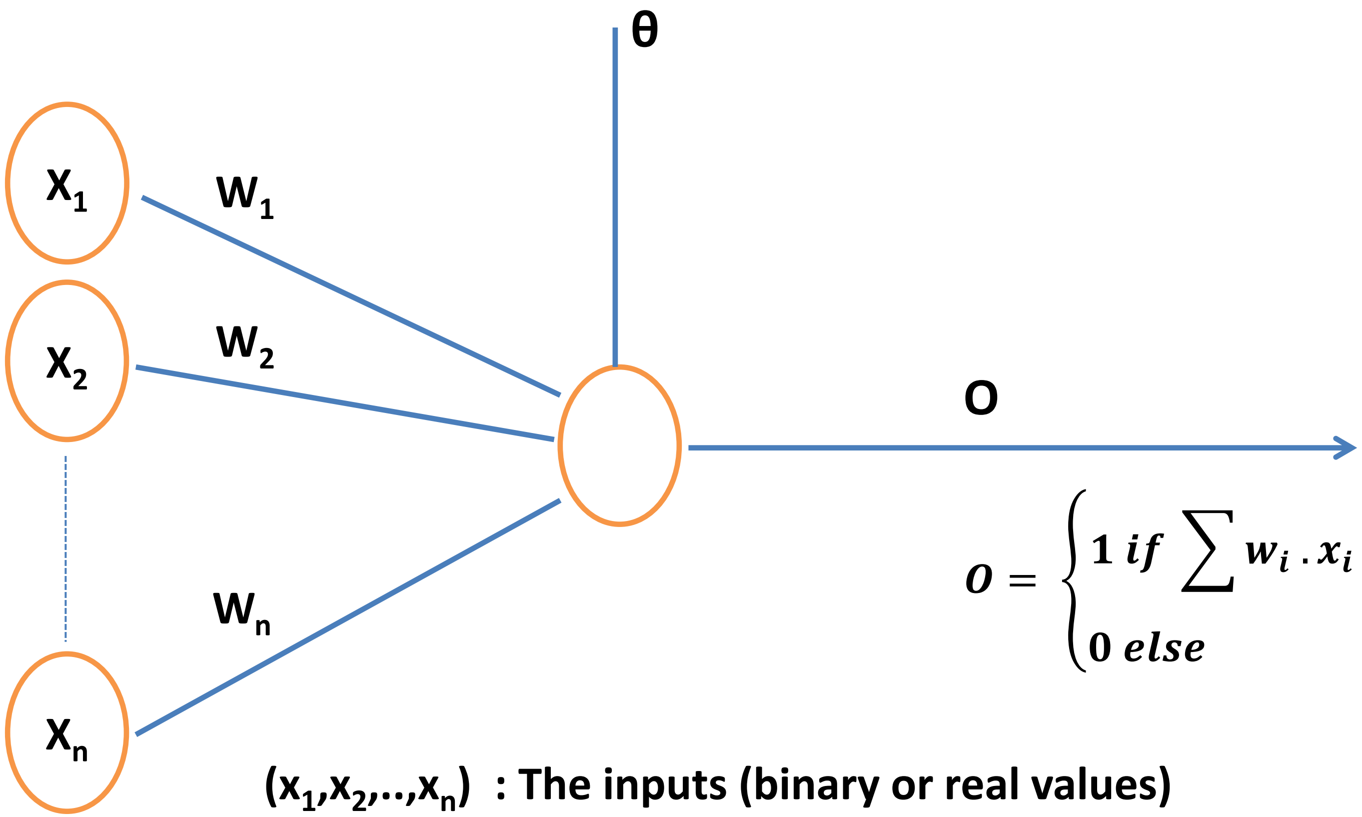
W = (w1,w2,…,wn) : Weight vector

S : Weighted summation of inputs

S = $\sum_{i=1}^{n} e_i \, . \, w_i$

Artificial Neural Networks (ANN)

PERCEPTRON MODEL: Basic diagram



$$O = \begin{cases} 1 \; if \; \sum w_i \,.\, x_i > \theta \\ 0 \; else \end{cases}$$

$(x_1, x_2, .., x_n)$: The inputs (binary or real values)

$(w_1, w_2, .., w_n)$: Vector of weights (Synaptic Coefficients)

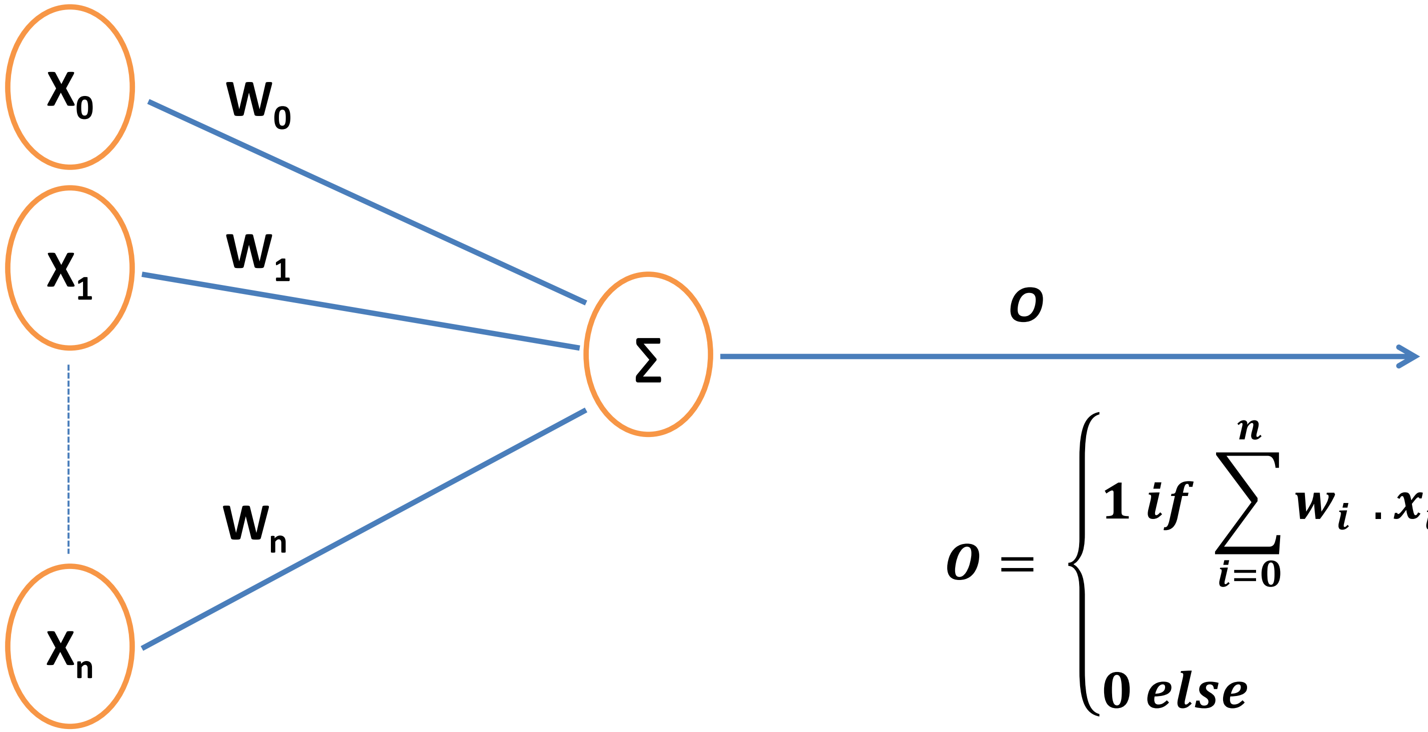
θ : Threshold (~Bias)

O : Output

Artificial Neural Networks (ANN)

PERCEPTRON MODEL: Simplified diagram

Replace the threshold **θ** with an additional input $\mathbf{x_0}$ which <u>always takes the value 1</u>, its input is associated with a coefficient $\mathbf{w_0}$



$$O = \begin{cases} 1 \; if \displaystyle\sum_{i=0}^{n} w_i \, . x_i > 0 \\ 0 \; else \end{cases}$$

PERCEPTRON MODEL

ERROR-CORRECTION LEARNING ALGORITHM

Given a training sample **S** of $\mathbf{R^n} \times \{0,1\}$ or of $\{0,1\}^n \times \{0,1\}$

- A set of examples whose descriptions are on **n real** or **binary** attributes and the result is a **binary** class.
- Find an algorithm which infers from **S**, a sample which correctly classifies the elements of **S** accordingly to their descriptions

Artificial Neural Networks (ANN)

ERROR-CORRECTION LEARNING ALGORITHM

PROCEDURE

- Initialize the $\mathbf{w}_i$ weights of the perceptron to **arbitrary** values.
- Each time we present a new example, we **adjust** the **weights** depending on whether the perceptron has correctly classified the example or not.
- Stop when all examples have been presented without modification of any weight.

ERROR-CORRECTION LEARNING ALGORITHM

NOTATION

- We note $\vec{x}$ a description which will be an element of the sample. The **i**th component of $\vec{x}$ will be denoted by $\mathbf{x_i}$.

- A sample **S** will therefore be a set of pairs ($\vec{x}$,**c**) where **c** is the class of $\vec{x}$.

- Noting that : $\mathbf{x_0=1}$ for which We do associate $\mathbf{w_0}$

Artificial Neural Networks (ANN)

ERROR-CORRECTION LEARNING ALGORITHM

Input : a sample S of R^n x {0,1} or of $\{0,1\}^n$ x {0,1}
Begin
- Random initialization of weights w_i $(w_0, w_1, .., w_n)$.
Repeat
-Take an example ($\vec{x}$,**c**) of **S**
-Calculate the output **O** of the perceptron for the input $\vec{x}$
-Update the weights (Adjustment)
For i := 0 to n
do
$w_i \leftarrow w_i + (c\text{-}o) \times x_i$
EndFor
EndRepeat
End

ERROR-CORRECTION LEARNING ALGORITHM

Example 1

- We want to build a perceptron that calculates the logical AND using an error-correction learning algorithm.
- We have as sample S={(00,0),(01,0),(10,0),(11,1)}.
- The initial weights are: -1, 1, 1.
- Stopping criterion: presentation of all examples in the sample.

➢ Reproduce the execution trace of the algorithm in a table?

Artificial Neural Networks (ANN)

ERROR-CORRECTION LEARNING ALGORITHM

Example 1

$$O = \begin{cases} 1 \; if \sum w_i . x_i > 0 \\ 0 \; else \end{cases}$$ and $$w_i \leftarrow wi + (c-o) \times xi$$

Iteration	W_0	W_1	W_2	X_0	X_1	X_2	$\sum w_i.x_i$	O	C	W_0	W_1	W_2
1	-1	1	1	1	0	0	-1	0	0	-1	1	1
2	-1	1	1	1	0	1	0	0	0	-1	1	1
3	-1	1	1	1	1	0	0	0	0	-1	1	1
4	-1	1	1	1	1	1	1	1	1	-1	1	1

We notice a stabilization of the weights from the first iteration. We then say that the perceptron was able to learn the calculation of the logical AND

ERROR-CORRECTION LEARNING ALGORITHM

Example 2

- We want to build a perceptron that calculates the logical XOR using an error-correction learning algorithm.
- The initial weights are: -1, 1, 1.
- Stopping criterion: presentation of all examples in the sample.

$$O = \begin{cases} 1 \; if \sum w_i \,.\, x_i > 0 \\ 0 \; else \end{cases} \quad \text{and} \quad w_i \leftarrow wi + (c-o) \times xi$$

- Give the input sample structure (pairs (input, output))
- Reproduce the execution trace of the algorithm in a table?
- What can We deduce**?**

ERROR-CORRECTION LEARNING ALGORITHM

Example 2

We have as a sample S={(00,0),(01,1),(10,1),(11,0)}.

Iteration	W_0	W_1	W_2	X_0	X_1	X_2	$\sum w_i.x_i$	O	C	W_0	W_1	W_2
1	-1	1	1	1	0	0	-1	0	0	-1	1	1
2	-1	1	1	1	0	1	0	0	1	0	1	2
3	0	1	2	1	1	0	1	1	1	0	1	2
4	0	1	2	1	1	1	3	1	0	-1	0	1

We deduce that the perceptron has not sufficiently learned the calculation of the XOR, we must repeat the operation (iterations) until we have a stabilization of the Wi weights W_i.

ERROR-CORRECTION LEARNING ALGORITHM

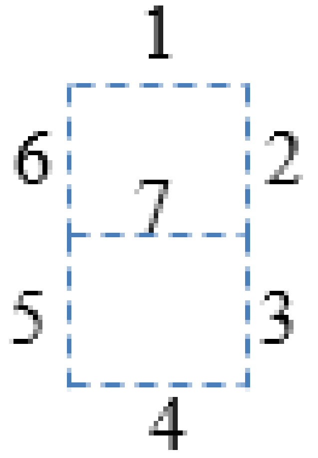
Exemple 3

We want to build a perceptron that recognizes whether a digit is even or odd. The expected result is therefore:

1 if the digit is odd (1, 3, 5, 7, 9).

0 if the digit is even (0, 2, 4, 6, 8).

The input digit is represented by a system of 7 LEDs. A LED is a segment that can be turned on (represented by 1) or off (represented by 0). The 7 LEDs are numbered as illustrated in the following figure:



$$O = \begin{cases} 1 \; if \sum w_i \,.\, x_i > 0 \\ 0 \; else \end{cases} \quad \text{and} \quad w_i \leftarrow wi + (c-o) \times xi$$

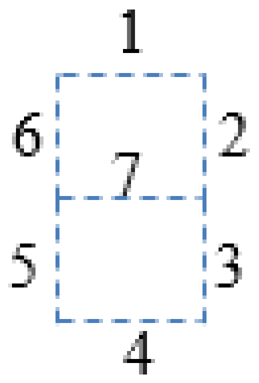
- Using the error-correction learning algorithm and choosing as:
  - Stopping criterion: the introduction of all examples in the sample.
  - Initial weights: 2, 1, 0, 1, 0, -1, 1, 1.

➢ Represent the execution trace of the algorithm in a table. Comment on the result?

Artificial Neural Networks (ANN)

ERROR-CORRECTION LEARNING ALGORITHM

Exemple 3



- Input : 0,1,2,3,4,5,6,7,8,9 (in binary values relatively to the 7 LEDs)
- Weights W_i: 2, 1, 0, 1, 0, -1, 1, 1
- X0 = 1
- The sample S will be a set of pairs ($\vec{x}$,c) where c is the class of $\vec{x}$

S = {(11111110,0), (10110000,1), (11101101,0), (11111001,1), (10110011,0), (11011011,1), (10011111,0), (11110000,1), (11111111,0), (11111011,1)}

Etape	Poids								Entrée pour chaque exemple								$\sum w_i x_i$	Sortie	Classe	Poids mis à jour							
	W0	W1	W2	W3	W4	W5	W6	W7	X0	X1	X2	X3	X4	X5	X6	X7		O	C	W0	W1	W2	W3	W4	W5	W6	W7
1	2	1	0	1	0	-1	1	1	1	1	1	1	1	1	1	0	4	1	0	1	0	-1	0	-1	-2	0	1
2	1	0	-1	0	-1	-2	0	1	1	0	1	1	0	0	0	0	0	0	1	2	0	0	1	-1	-2	0	1
3	2	0	0	1	-1	-2	0	1	1	1	1	0	1	1	0	1	0	0	0	2	0	0	1	-1	-2	0	1
4	2	0	0	1	-1	-2	0	1	1	1	1	1	1	0	0	1	3	1	1	2	0	0	1	-1	-2	0	1
5	2	0	0	1	-1	-2	0	1	1	0	1	1	0	0	1	1	4	1	0	1	0	-1	0	-1	-2	-1	0
6	1	0	-1	0	-1	-2	-1	0	1	1	0	1	1	0	1	1	-1	0	1	2	1	-1	1	0	-2	0	1
7	2	1	-1	1	0	-2	0	1	1	1	0	1	1	1	1	1	3	1	0	1	0	-1	0	-1	-3	-1	0
8	1	0	-1	0	-1	-3	-1	0	1	1	1	1	0	0	0	0	0	0	1	2	1	0	1	-1	-3	-1	0
9	2	1	0	1	-1	-3	-1	0	1	1	1	1	1	1	1	1	-1	0	0	2	1	0	1	-1	-3	-1	0
10	2	1	0	1	-1	-3	-1	0	1	1	1	1	1	0	1	1	2	1	1	2	1	0	1	-1	-3	-1	0

Comment on the result: We notice that from the 8th step there is a stabilization of weights. We then say that the perceptron has started to learn the parity calculation of the ten digits

LES RESEAUX DE NEURONES

ALGORITHME D'APPRENTISSAGE PAR CORRECTION D'ERREUR

Exercice

Soit à modéliser la classification des couleurs avec un réseau de neurone artificiel. On dispose d'un échantillon d'apprentissage composé de 4 exemples partagés en deux classes de couleurs ROUGE et BLEUE. Chaque exemple est décrit par sa représentation RVB (voir Table). On souhaite appliquer un modèle de perceptron avec un algorithme d'apprentissage par correction d'erreur pour déterminer la classe de chaque exemple.

- Décrivez les différents paramètres utilisés par le modèle **?**
- Déroulez l'algorithme sur un tableau, en prenant comme critère d'arrêt : Nombre d'itérations = 2.

Commentez les résultats **?**

	R(Rouge)	V(Vert)	B(Bleu)
ROUGE	255	0	0
	248	80	68
BLEUE	0	0	255
	67	15	210

$$O = \begin{cases} 1 \ si \ \sum_{i=0}^{n} w_i \, . x_i > 0 \\ 0 \ sinon \end{cases}$$

$$\mathbf{Wi = Wi + (C\text{-}O) \times Xi}$$

LES RESEAUX DE NEURONES

ALGORITHME D'APPRENTISSAGE PAR CORRECTION D'ERREUR

Exercice

Paramètres :

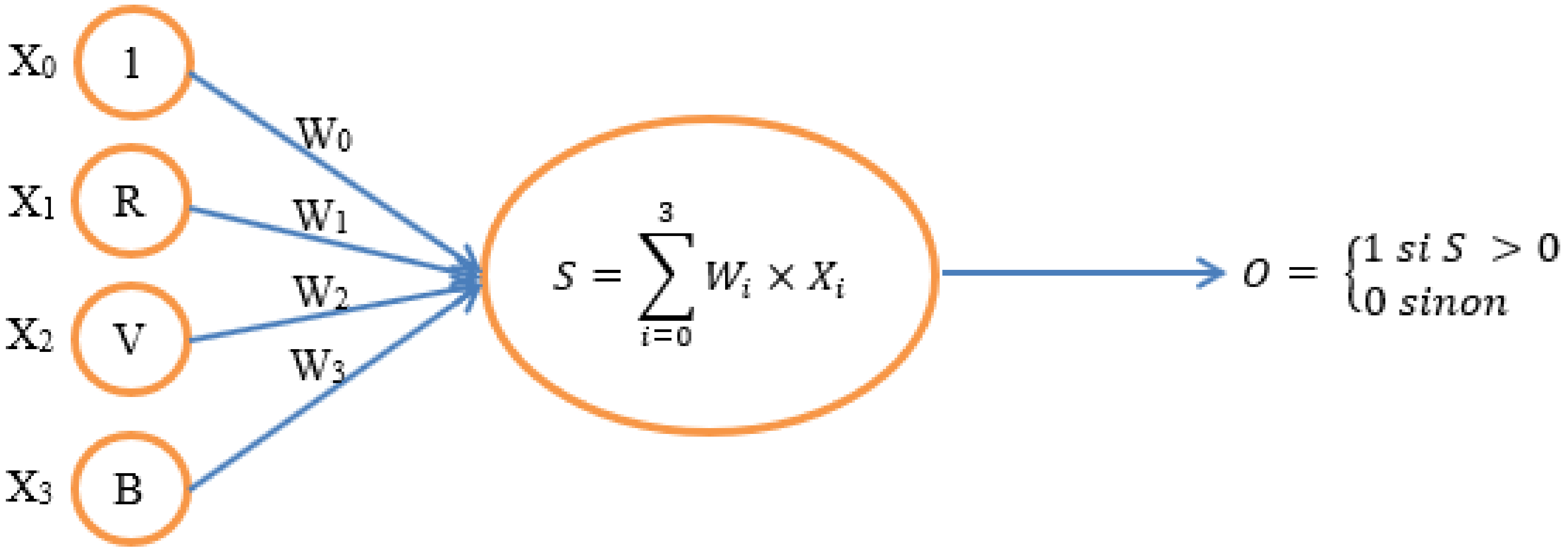
- Les entrées : X_0 = 1, X_1 = R, X_2 = V, X_3 = B (les valeurs RVB de chaque exemple)
- Les poids initiaux : W_0 = -1, W_1 = 2, W_2 = 1, W_3 = -2 (valeurs réelles quelconques)
- Les classes : ROUGE (1), BLEUE (0) (on associe une valeur binaire à chaque classe)
- La sommation $S = \sum_{i=0}^{3} W_i \times X_i$
- La sortie $O = \begin{cases} 1 \; si \; S > 0 \\ 0 \; sinon \end{cases}$
- Mise à jour des poids : $W_i = W_i + (C - O) \times X_i$
- Le vecteur d'entrée aura donc la forme (1.R.V.B, classe)
- Le premier échantillon d'entrée aura la forme : (1.255.0.0,1), (1.248.80.68,1), (1.0.0.255,0), (1.67.15.210,0)

LES RESEAUX DE NEURONES

ALGORITHME D'APPRENTISSAGE PAR CORRECTION D'ERREUR

Exercice

Schéma du modèle :



LES RESEAUX DE NEURONES

ALGORITHME D’APPRENTISSAGE PAR CORRECTION D’ERREUR

Exercice

Déroulement (nb-itérations = 2):

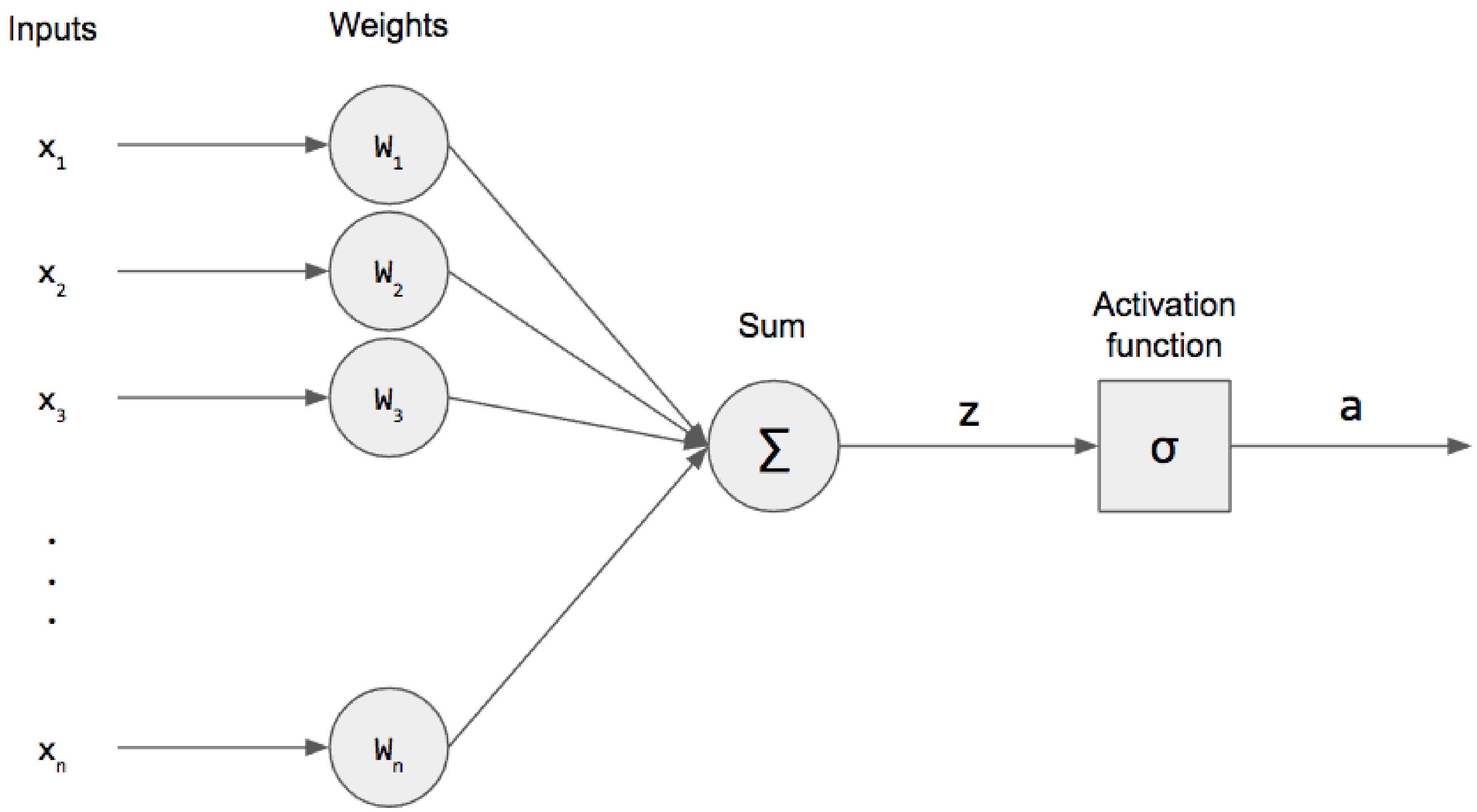
Etapes		W_0	W_1	W_2	W_3	X_0	X_1	X_2	X_3	$\sum w_i.x_i$	O	C	W_0	W_1	W_2	W_3
Itération 1	Exemple 1	-1	2	1	-2	1	255	0	0	509	1	1	-1	2	1	-2
	Exemple 2	-1	2	1	-2	1	248	80	68	475	1	1	-1	2	1	-2
	Exemple 3	-1	2	1	-2	1	0	0	255	-511	0	0	-1	2	1	-2
	Exemple 4	-1	2	1	-2	1	67	15	210	-272	0	0	-1	2	1	-2
Itération 2	Exemple 1	-1	2	1	-2	1	255	0	0	509	1	1	-1	2	1	-2
	Exemple 2	-1	2	1	-2	1	248	80	68	475	1	1	-1	2	1	-2
	Exemple 3	-1	2	1	-2	1	0	0	255	-511	0	0	-1	2	1	-2
	Exemple 4	-1	2	1	-2	1	67	15	210	-272	0	0	-1	2	1	-2

- Interprétation :

On remarque une stabilisation des poids dès la première itération. On dit alors, que le perceptron ne trouve pas de difficulté pour apprendre à classer les exemples présentés dans l’échantillon d’entrée. On peut dire aussi que le vecteur des poids initiaux a été bien choisi.

Artificial Neural Networks (ANN)

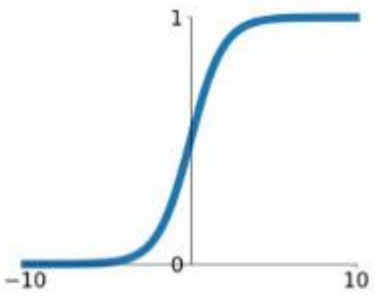
Perceptron model



Activation functions

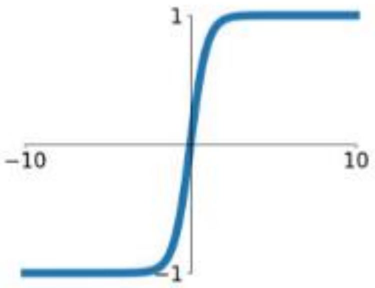
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



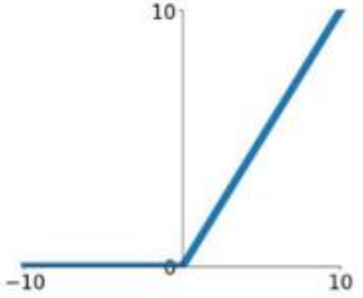
tanh

$\tanh(x)$



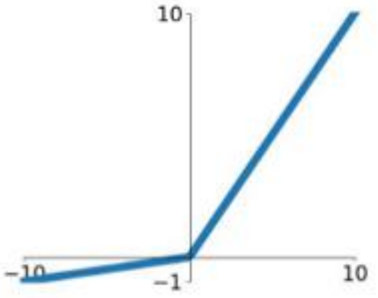
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

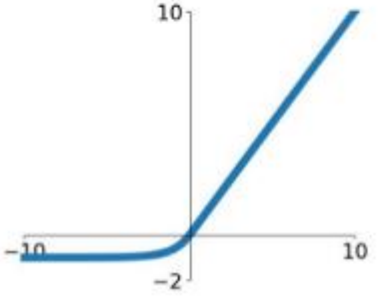


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

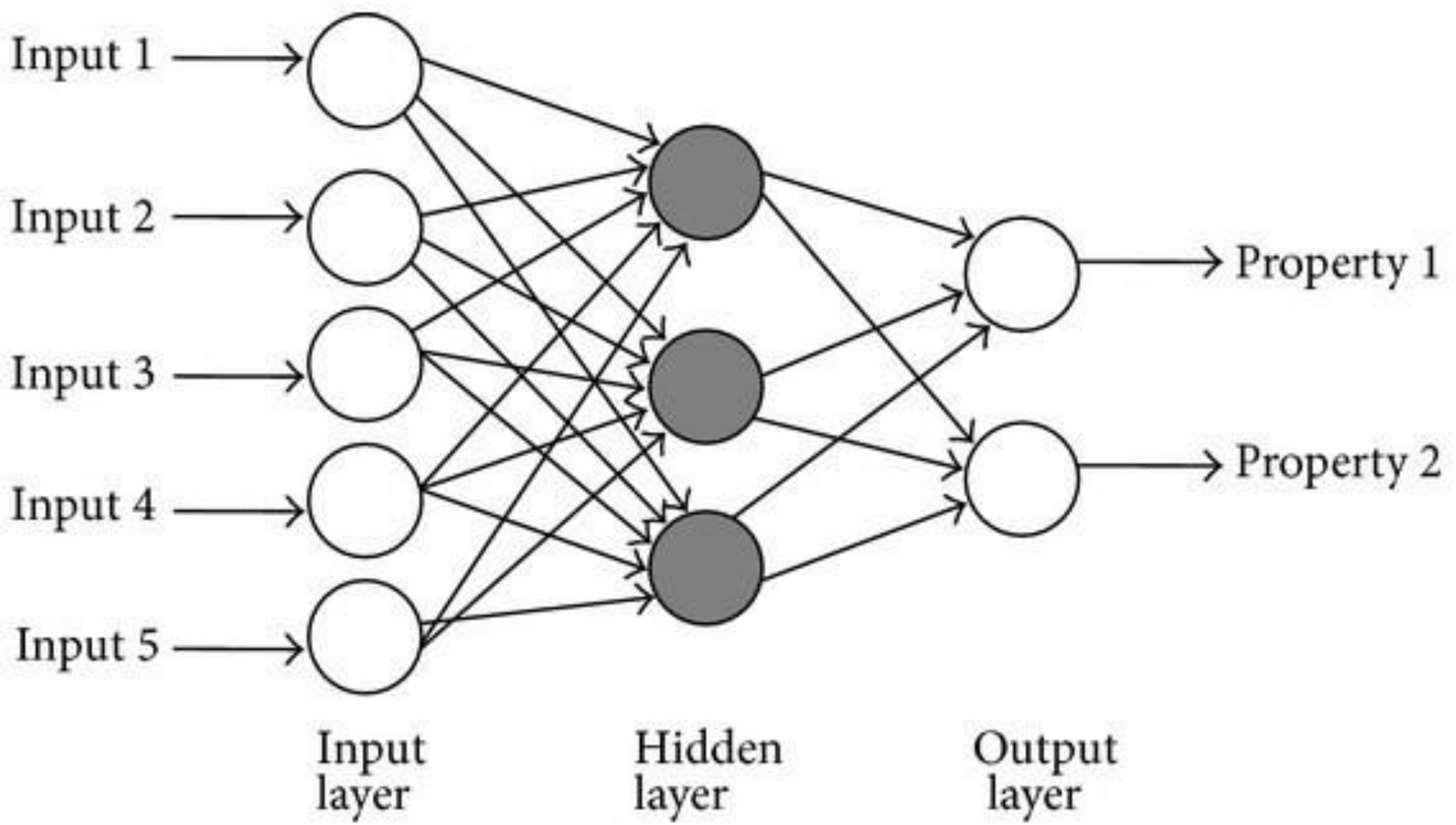
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



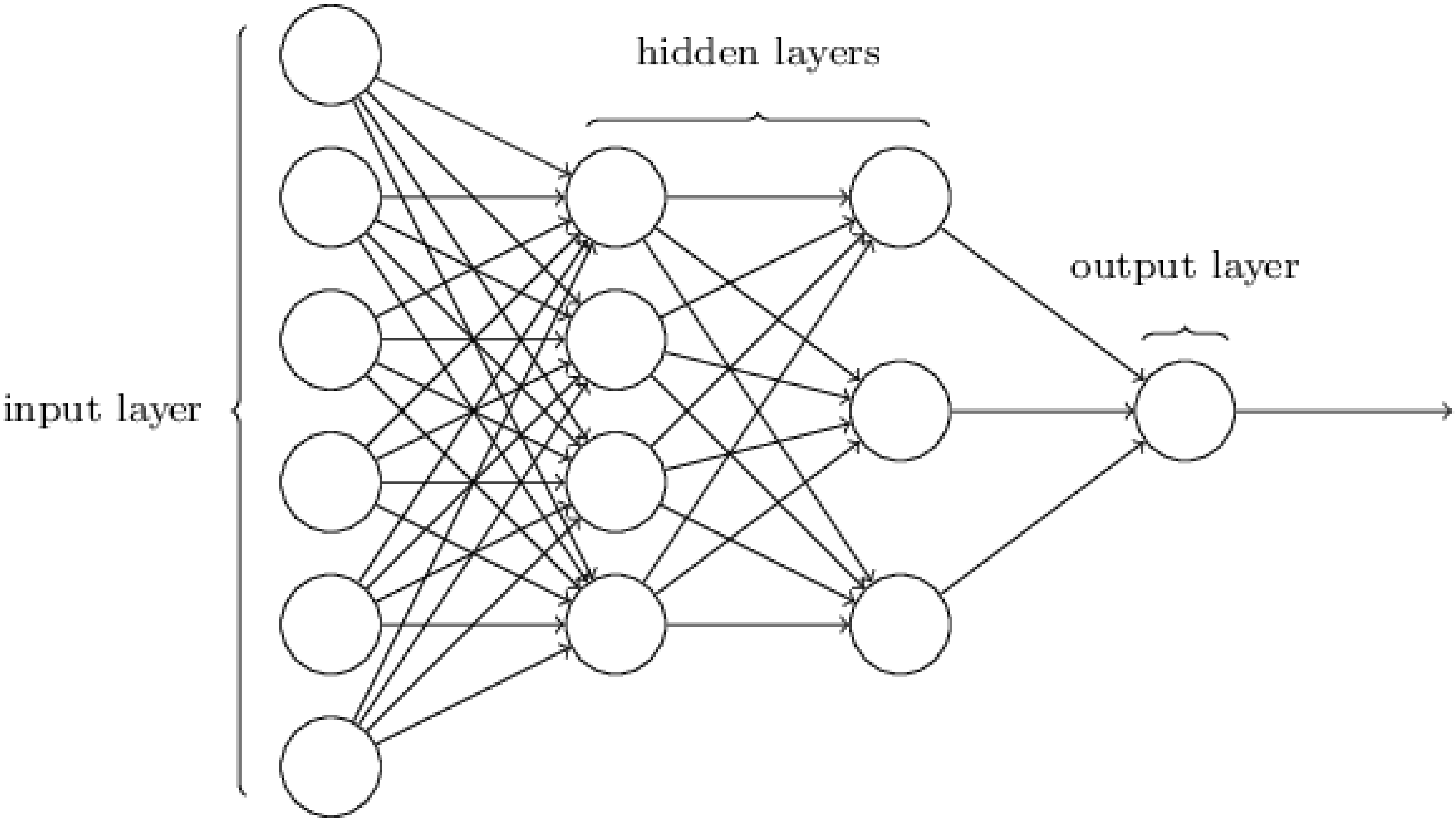
Artificial Neural Networks (ANN)

Perceptron model (with one hidden layer)



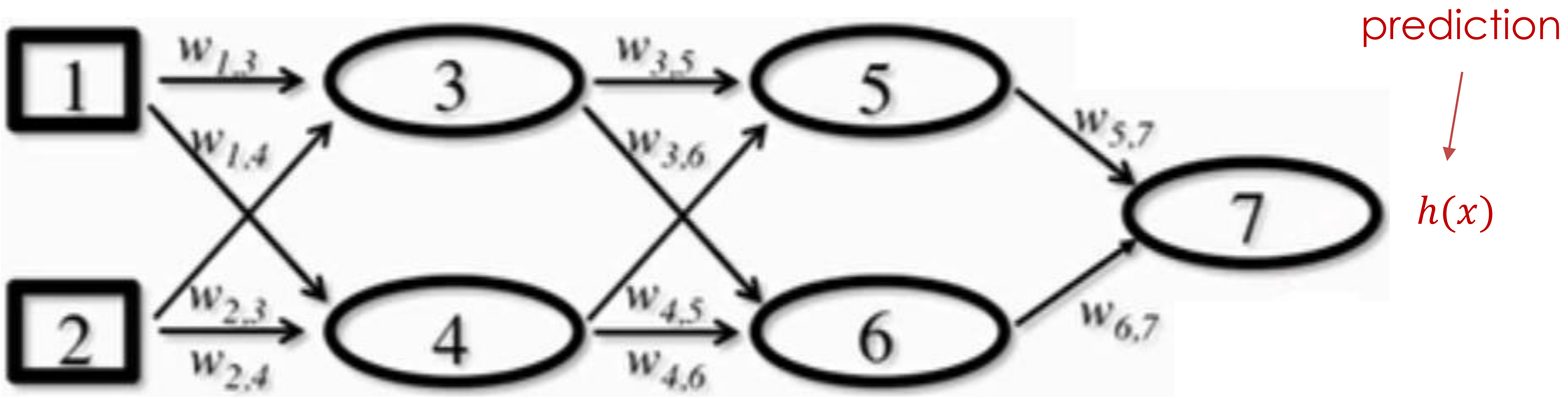
Artificial Neural Networks (ANN)

Multilayer Perceptron (MLP)



Artificial Neural Networks (ANN)

Backpropagation Algorithm



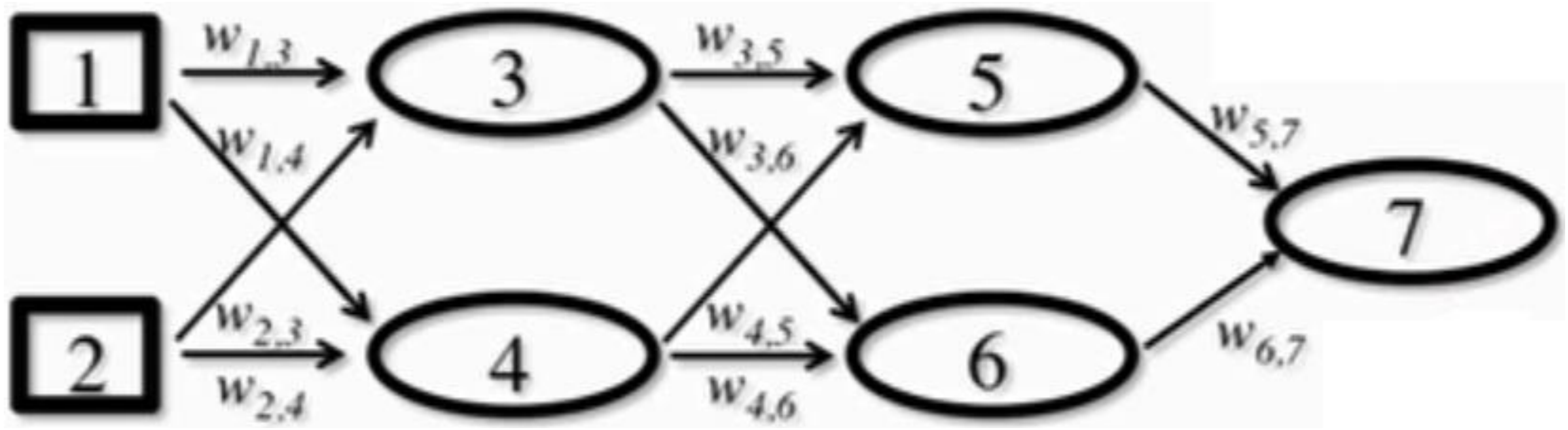
On note :

- a_j l'activité du neurone j
- in_j l'activité du neurone avant l'application de la fonction d'activation
  - $a_j = \sigma(in_j) = \sigma(\sum_i w_{i,j} a_i)$
- $h(x)$ la prédiction

LES RESEAUX DE NEURONES

Algorithme d’apprentissage par rétropropagation

Propagation avant

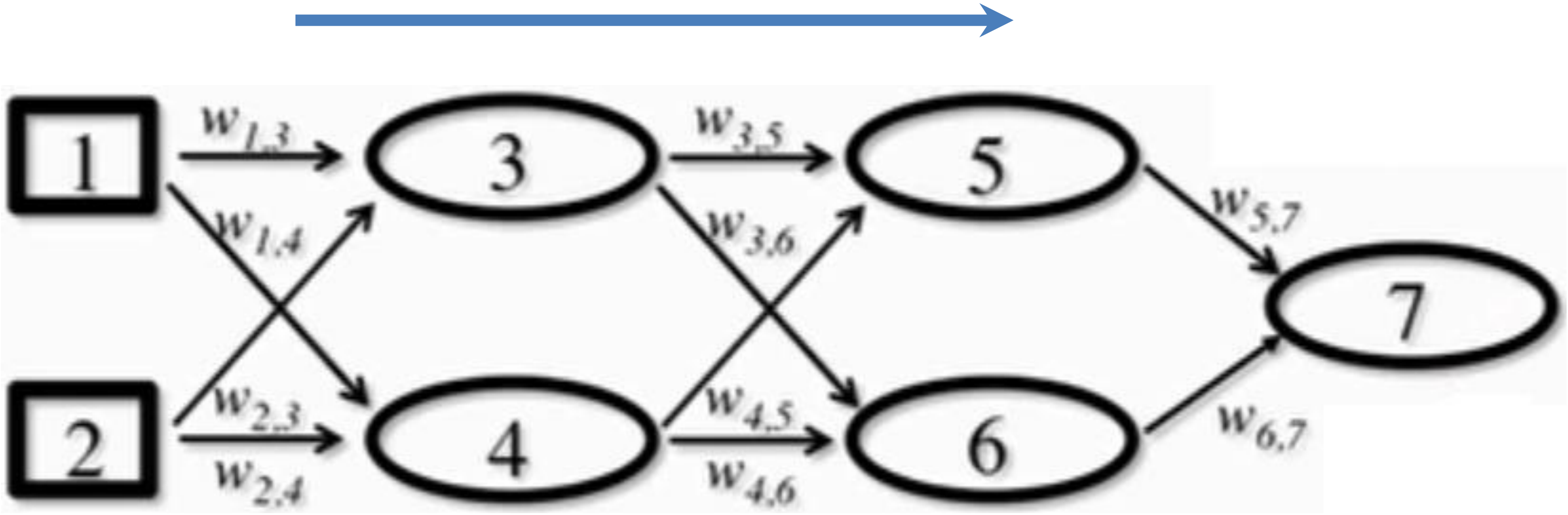


Propagation arrière

LES RESEAUX DE NEURONES

Algorithme d’apprentissage par rétropropagation

Propagation avant

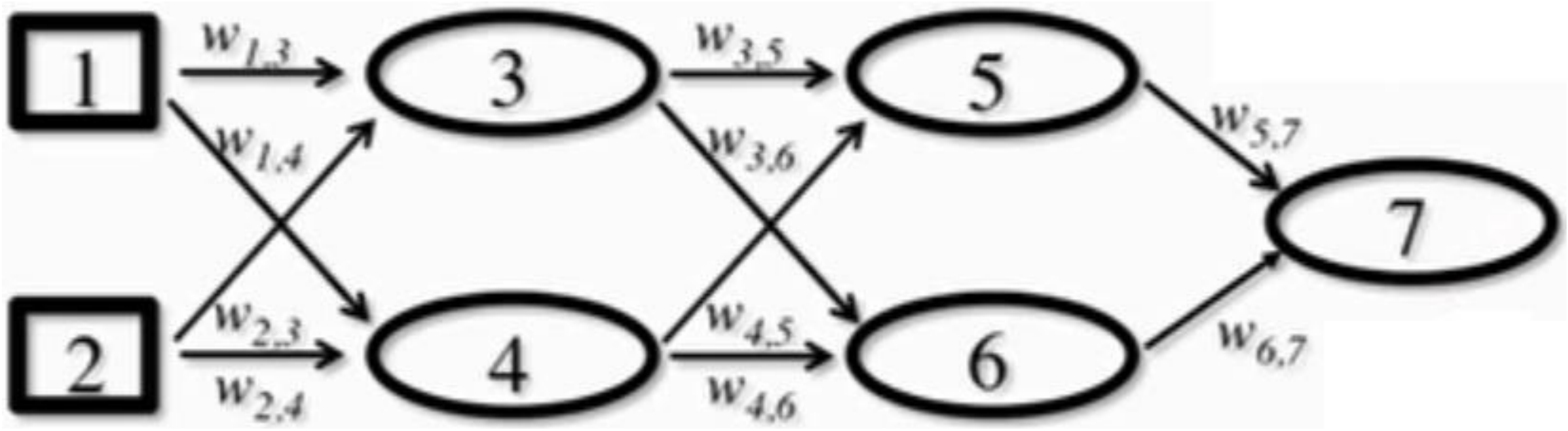


$$a_k = \sigma\left(\sum_j w_{j,k} a_j\right)$$

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation

Mise à jour des poids



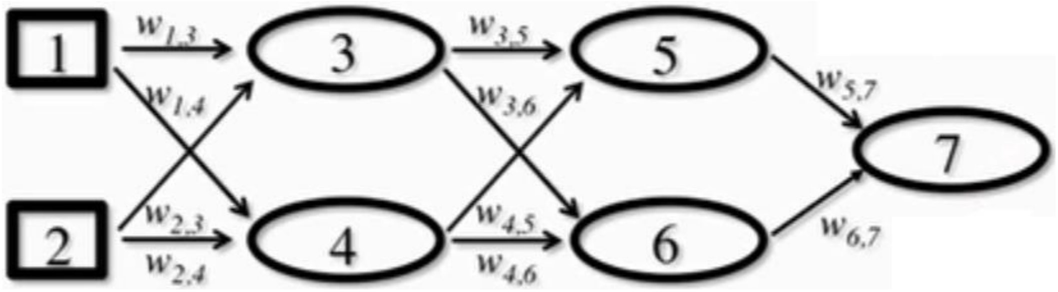
$$w_{i,j} = w_{i,j} - \alpha \frac{\partial}{\partial w_{i,j}} Loss(y_t, h_w(x_t))$$

- α : taux d'apprentissage
- $Loss$: taux de perte associée à la prédiction courante

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation

Mise à jour des poids (simplification)



$$w_{i,j} = w_{i,j} - \alpha \underbrace{\frac{\partial}{\partial in_j} Loss(y_t, h_w(x_t))}_{-\Delta[j]} \underbrace{\frac{\partial}{\partial w_{i,j}} in_j}_{a_i}$$

- Réécriture de la règle de mise à jour :

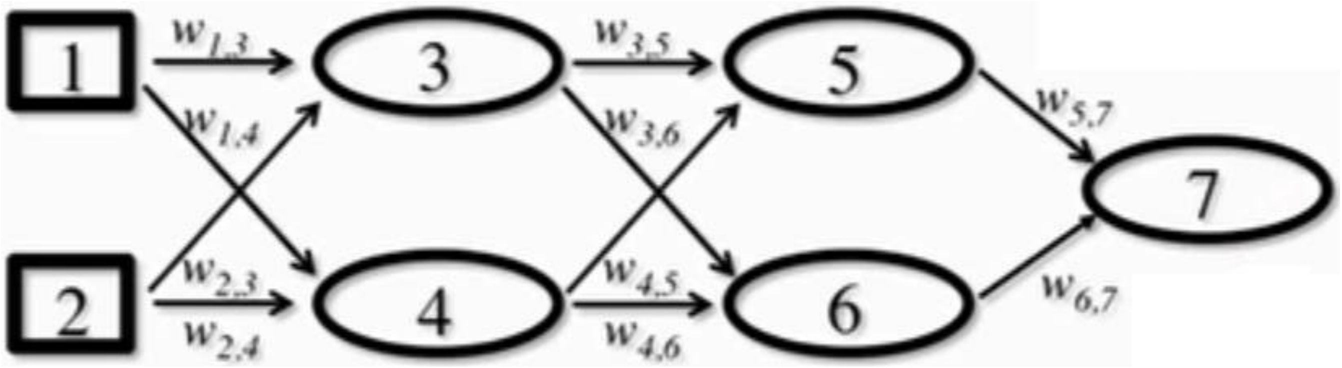
$$w_{i,j} = w_{i,j} + \alpha\, a_i \Delta[j]$$

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation

Mise à jour des poids (simplification)

Propagation arrière



- **Simplification de** $\Delta[j]$:

$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

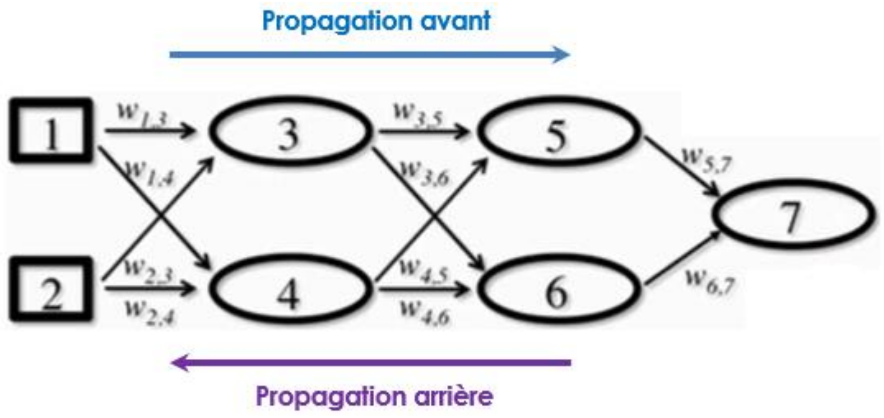
function BACK-PROP-LEARNING(examples,network) **returns** a neural network
Inputs: examples, a set of examples, each with input vector $\vec{x}$ and output vector y
network, a multilayer network with L layers, weights $w_{i\ j}$, activation function g
Local variables: Δ, a vector of errors, indexed by network node
for each weight $w_{i\ j}$ in network **do**
$w_{i\ j} \leftarrow \ a\ small\ random\ number$
repeat
for each example (x, y) **in** examples **do**
/* Propagate the inputs forward to compute the outputs */
for each node i in the input layer **do**
$a_i \leftarrow \ x_i$
for $l \ = \ 2$ to L do
for each node j in layer l do
$in_i \ \leftarrow \ \sum_i w_{i,j}\, a_i$
$a_j \ \leftarrow g(in_j)$
/* Propagate deltas backward from output layer to input layer */
for each node j in the output layer l do
$\Delta[j] \leftarrow \ y_j - a_j \qquad (= -\partial Loss/\partial in_j)$
for $l \ = L - 1$ to 1 do
for each node i in layer l do
$\Delta[i] = g(in_i)(1 - g(in_i)) \sum_j w_{i,j} \Delta[j]$
/* Update every weight in network using deltas */
for each weight $w_{i,j}$ in network do
$w_{i,j} \ \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return network

$$g = sigmoid$$

$$g = sigmoid$$

LES RESEAUX DE NEURONES

Algorithme d'apprentissage par rétropropagation

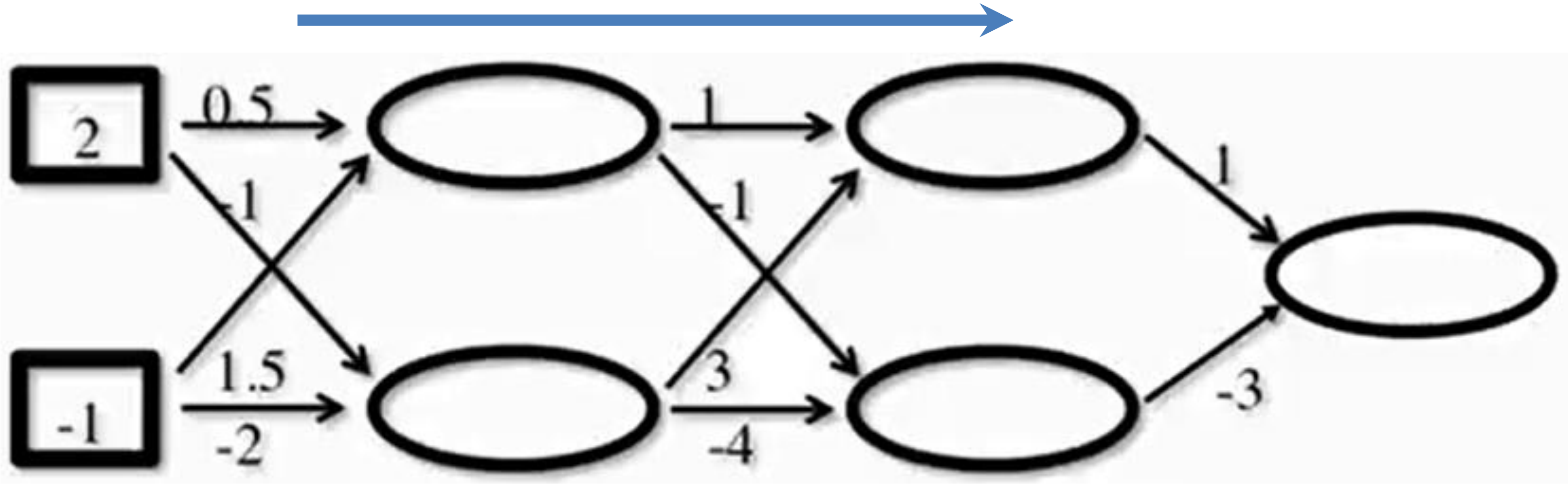


- **Propagation avant :** $a_k = \sigma(\sum_j w_{j,k} a_j)$
- **Mise à jour des poids :** $w_{i,j} = w_{i,j} + \alpha\, a_i \Delta[j]$
- **Ecart de la perte :** $\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1]$, $y = 1$, $\alpha = 0.1$

Propagation avant

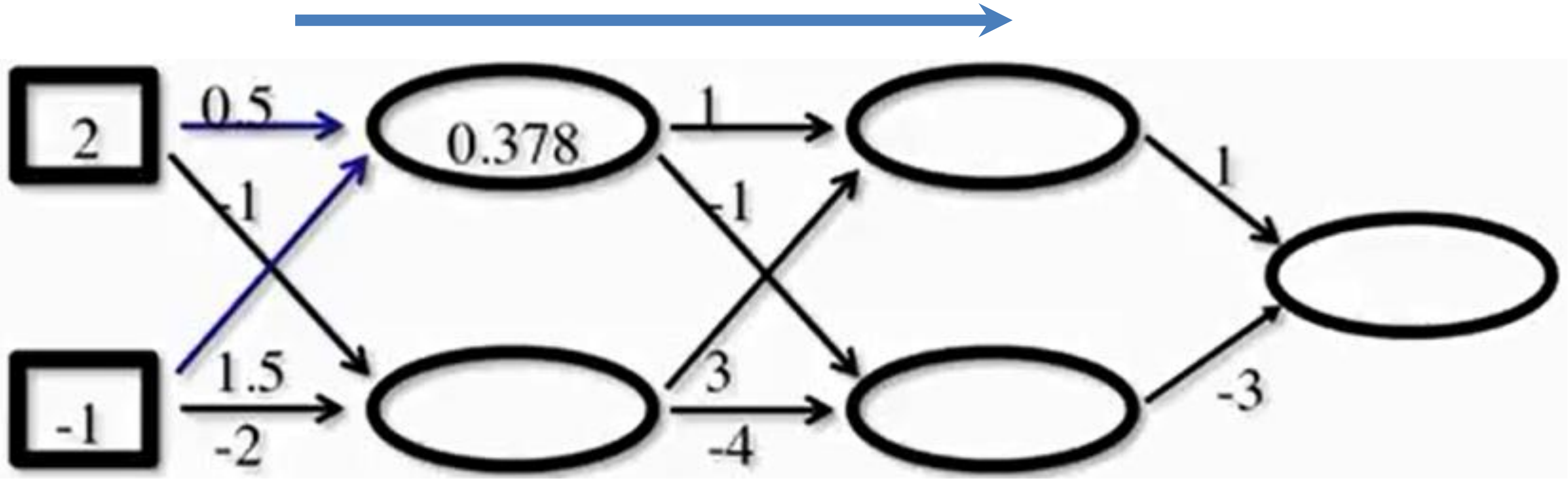


$$a_k = \sigma\left(\sum_j w_{j,k} a_j\right)$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1]$, $y = 1$, $\alpha = 0.1$

Propagation avant

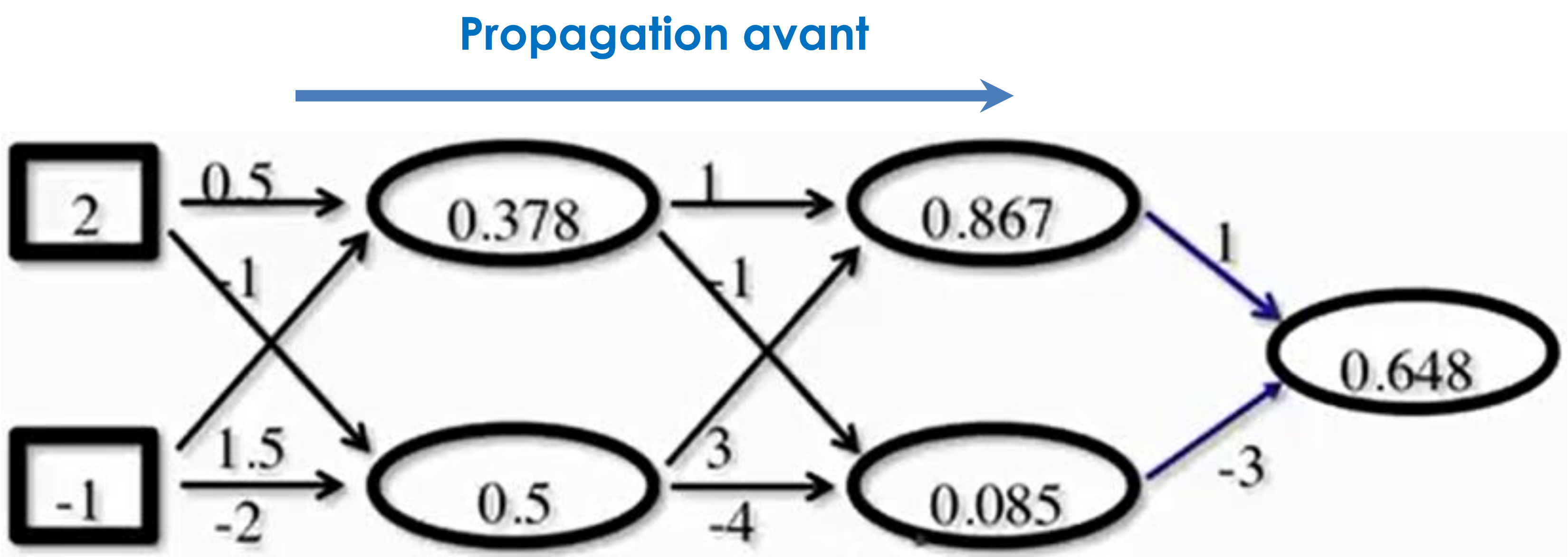


$$a_k = \sigma\left(\sum_j w_{j,k} a_j\right)$$

$$a_3 = \sigma(0.5 \times 2 + 1.5 \times -1) = \sigma(-0.5) = 0.378$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1]$, $y = 1$, $\alpha = 0.1$



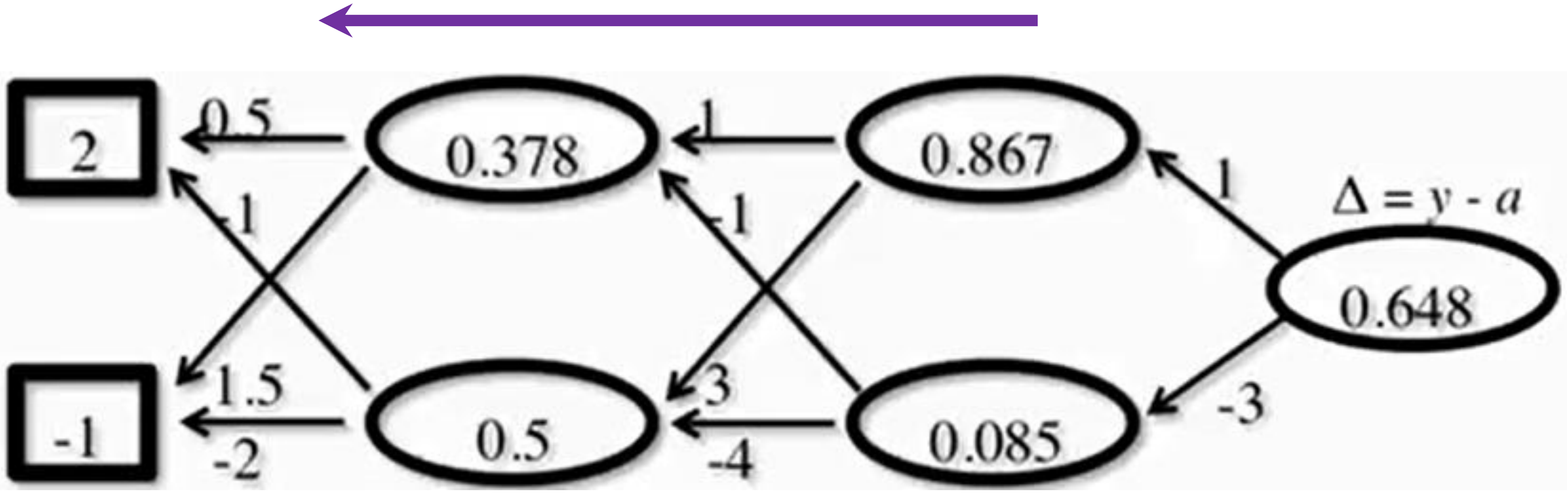
$$a_k = \sigma(\sum_j w_{j,k} a_j)$$

$$a_7 = \sigma(1 \times 0.867 + (-3) \times 0.085) = \sigma(0.612) = 0.648$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation arrière

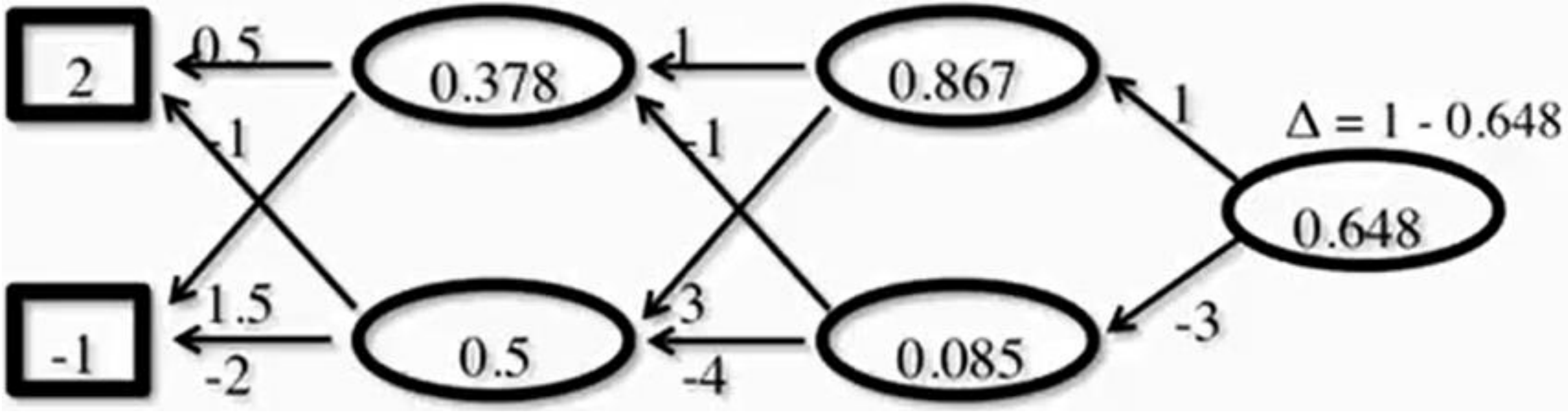


$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1]$, $y = 1$, $\alpha = 0.1$

Propagation arrière

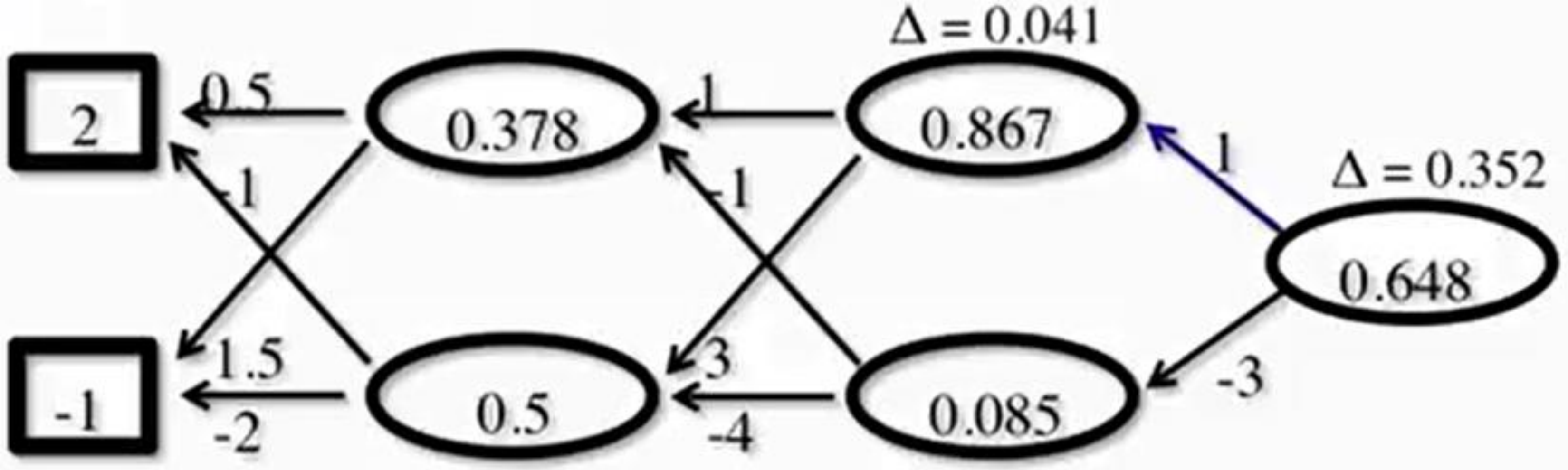


$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1]$, $y = 1$, $\alpha = 0.1$

Propagation arrière



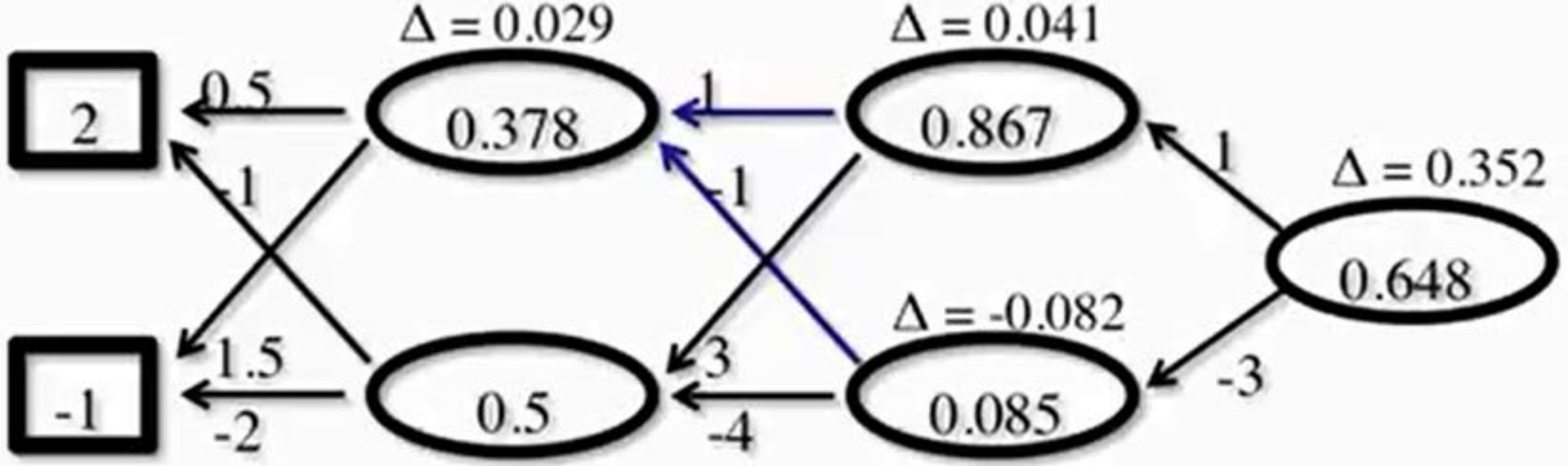
$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

$$\Delta[5] = 0.867 \times (1 - 0.867) \times 1 \times 0.352 = 0.041$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation arrière



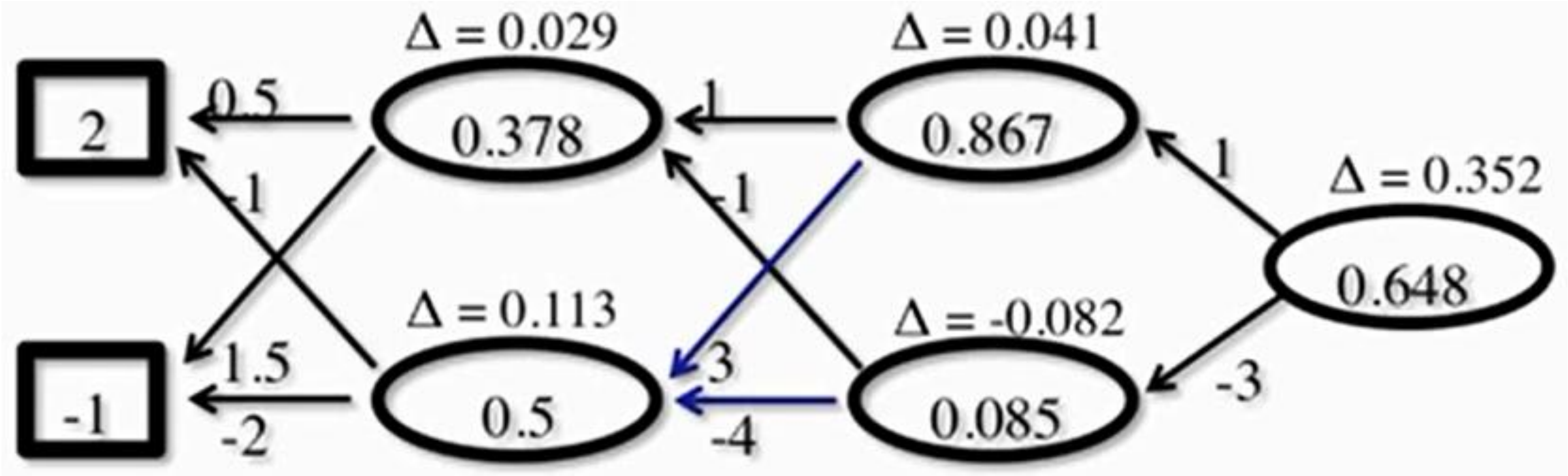
$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

$$\Delta[3] = 0.378 \times (1 - 0.378) \times (1 \times 0.041 + (-1) \times (-0.082)) = 0.029$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Propagation arrière



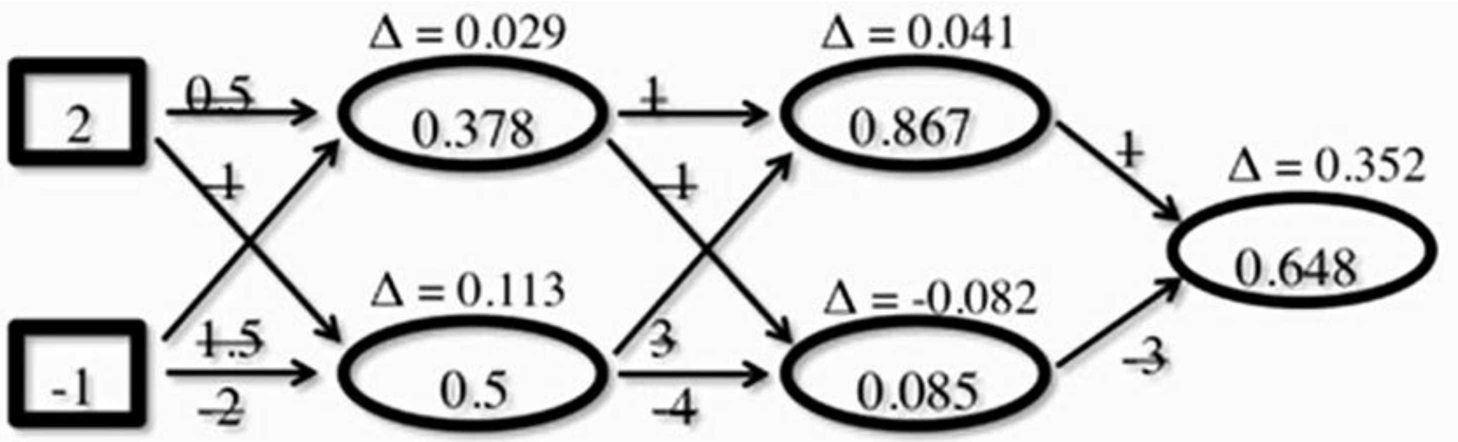
$$\Delta[j] = \sigma(in_j)(1 - \sigma(in_j)) \sum_k w_{j,k} \Delta[k]$$

$$\Delta[4] = 0.5 \times (1 - 0.5) \times (3 \times 0.041 + (-4) \times (-0.082)) = 0.113$$

Algorithme d'apprentissage par rétropropagation

Exemple : $x = [2, -1], \quad y = 1, \quad \alpha = 0.1$

Mise à jour des poids



$$w_{i,j} = w_{i,j} + \alpha\, a_i \Delta[j]$$

$w_{1,3} \leftarrow 0.5 + 0.1 * 2 * 0.029 = 0.506$

$w_{1,4} \leftarrow -1 + 0.1 * 2 * 0.113 = -0.977$

$w_{2,3} \leftarrow 1.5 + 0.1 * -1 * 0.029 = 1.497$

$w_{2,4} \leftarrow -2 + 0.1 * -1 * 0.113 = -2.011$

$w_{3,5} \leftarrow 1 + 0.1 * 0.378 * 0.041 = 1.002$

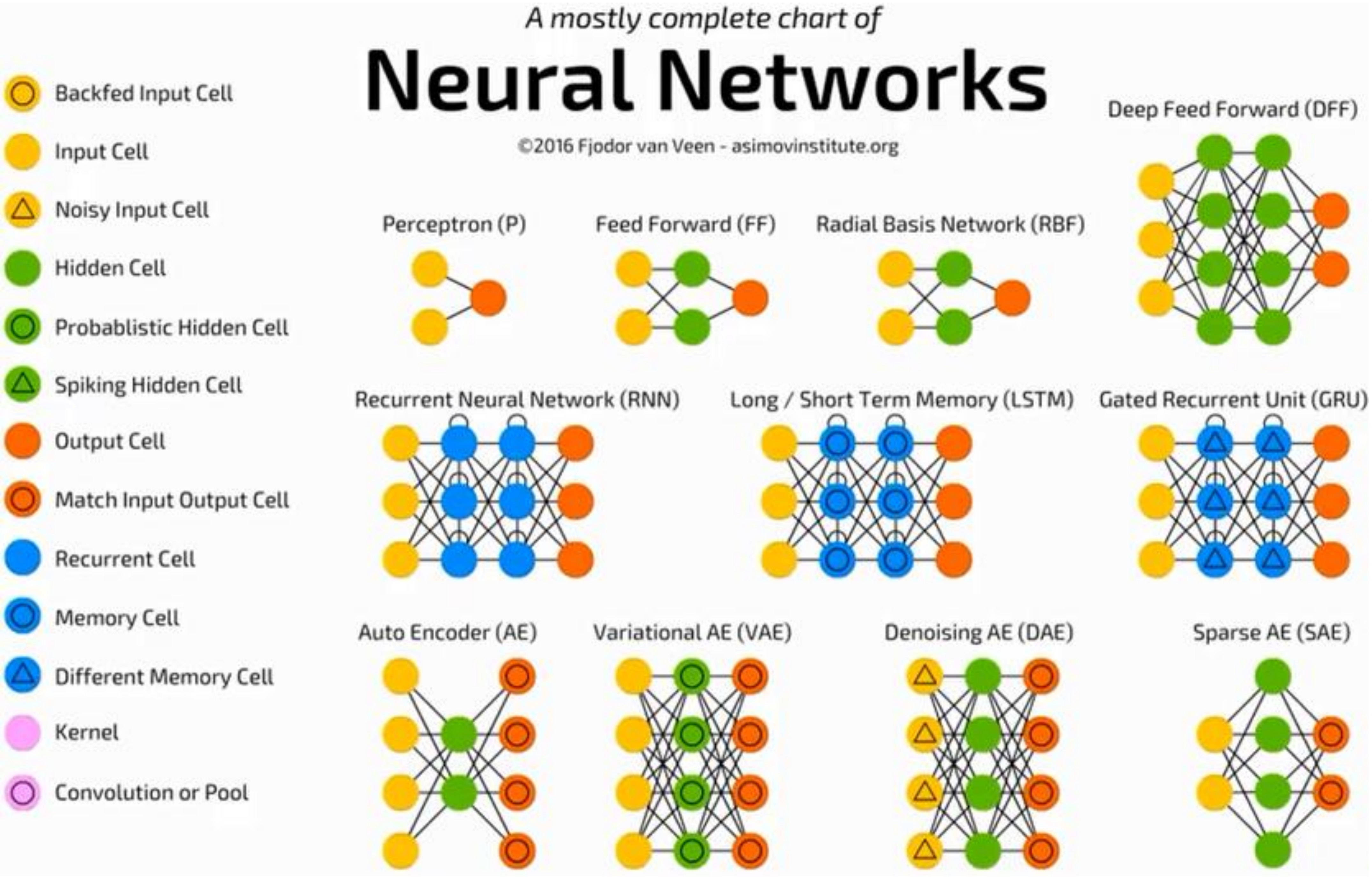
$w_{3,6} \leftarrow -1 + 0.1 * 0.378 * -0.082 = -1.003$

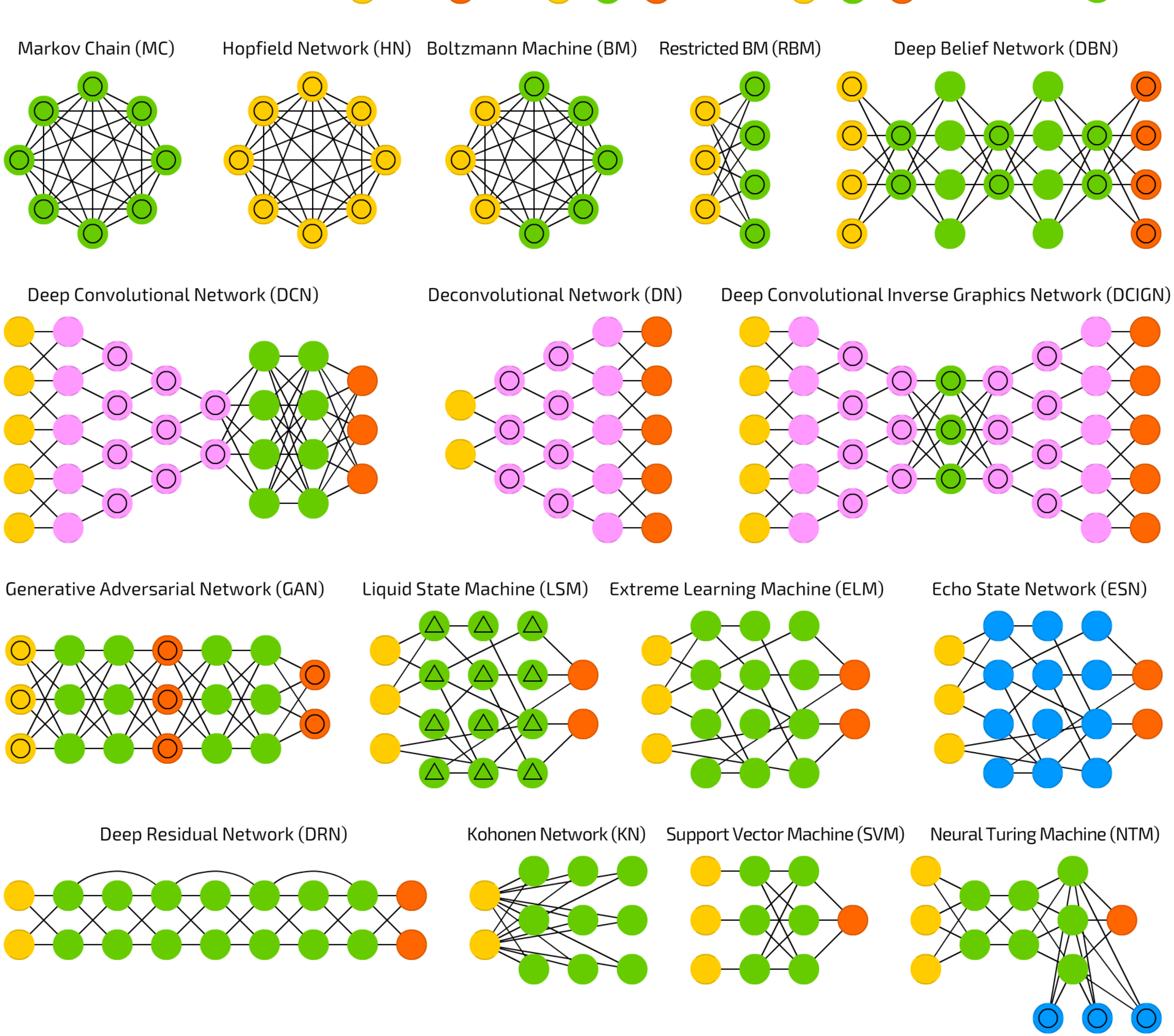
$w_{4,5} \leftarrow 3 + 0.1 * 0.5 * 0.041 = 3.002$

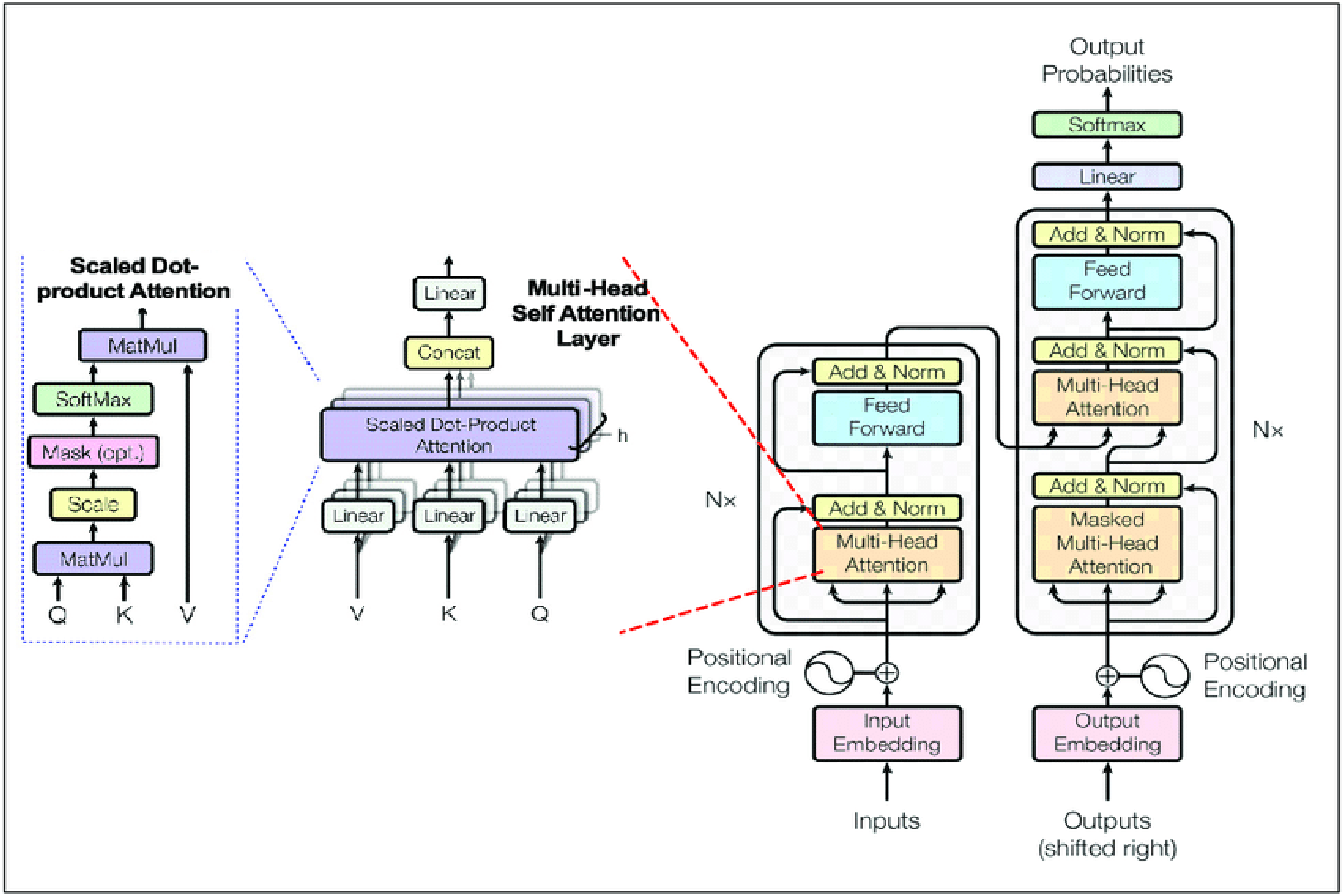
$w_{4,6} \leftarrow -4 + 0.1 * 0.5 * -0.082 = -4.004$

$w_{5,7} \leftarrow 1 + 0.1 * 0.867 * 0.352 = 1.031$

$w_{6,7} \leftarrow -3 + 0.1 * 0.085 * 0.352 = -2.997$


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)


Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)



Transformer architecture

Presentations planning

N°	Topic	Team	Date
12	Text To Speech	Aliouat - Mokrane - Khlidj	24/10/2022
6	Autonomous car	Mokrani - Kerrouche	24/10/2022
8	Internet of Things	Said Errahmani – Bendjilali - Mezouaghi	31/10/2022
3	Speech recognition	Ramla M. – Zorgane	31/10/2022
7	Web of Things	Hachemi – Bouamama - Boumelal	07/11/2022
1	Artificial life	Cherchali – Belkacem - Messous	07/11/2022
4	Computer vision	Ferrah – Frid - Drani	14/11/2022
2	Virtual reality	Sekkal - Belhenniche	14/11/2022
11	Handwritting recognition	Benameur - Kridi	21/11/2022
10	Bio-informatics	Hamadouche - Tabouche	21/11/2022
5	Augmented reality	Ramla R. – Rahmani – Ahmed Zitouni	28/11/2022
9	Robotics	Benbrik & Nourine & Rezkallah	28/11/2022
13	Part-Of-Speech Tagging	Ratta & Chaouch	05/12/2022
14	Data Science	Bachir	05/12/2022