

Introduction to Neural Networks

Martin Vielvoye - 2021

Motivation

**Il est difficile d'inventer une telle machine sans
s'inspirer de l'un des ordinateurs les plus
performants existant...**


Circuit et Cerveau

Le cerveau a toujours été une inspiration pour l'ordinateur.

► Alan Turing

- ◎ Imagine et définit l'ordinateur universel
- ◎ The Turing Game
- ◎ Qu'est-ce que représente la pensée et les limites de la computation

► John von Neumann

- ◎ Réalise le premier ordinateur programmable (5 Ko)
- ◎ *"The Computer & the Brain"* et *"Turing's Cathedral"* 

► Frank Rosenblatt

- ◎ Psychologue spécialiste de l'apprentissage et du cerveau
- ◎ Invente le modèle du *Perceptron*

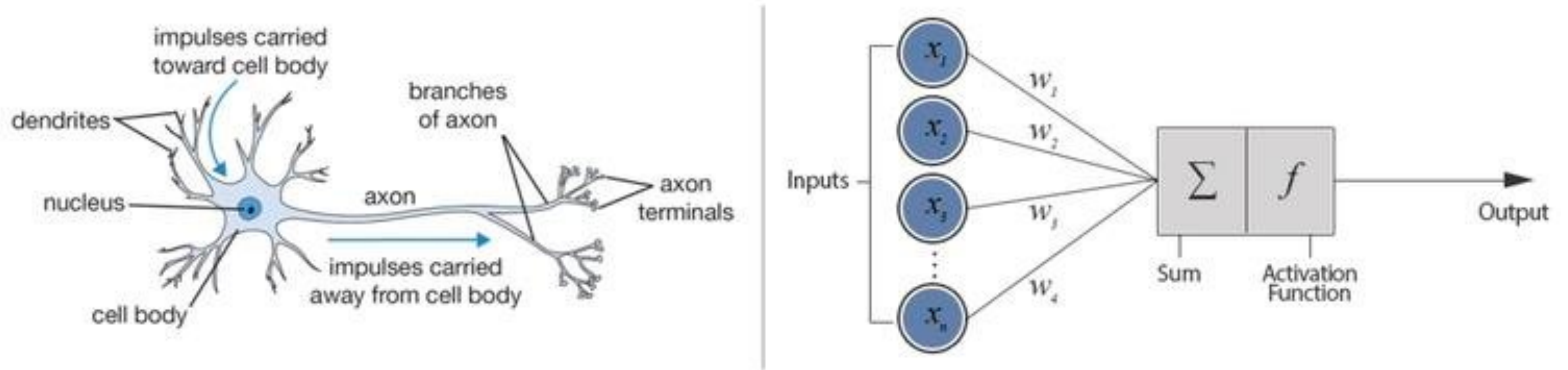
Introduction au Perceptron

Distinguishing between numbers that *mean* things and numbers that *do* things.

Distinguer les nombres qui *veulent dire* et les nombres qui *font*.

Perceptron et B-Neurone

Biological Neuron versus Artificial Neural Network

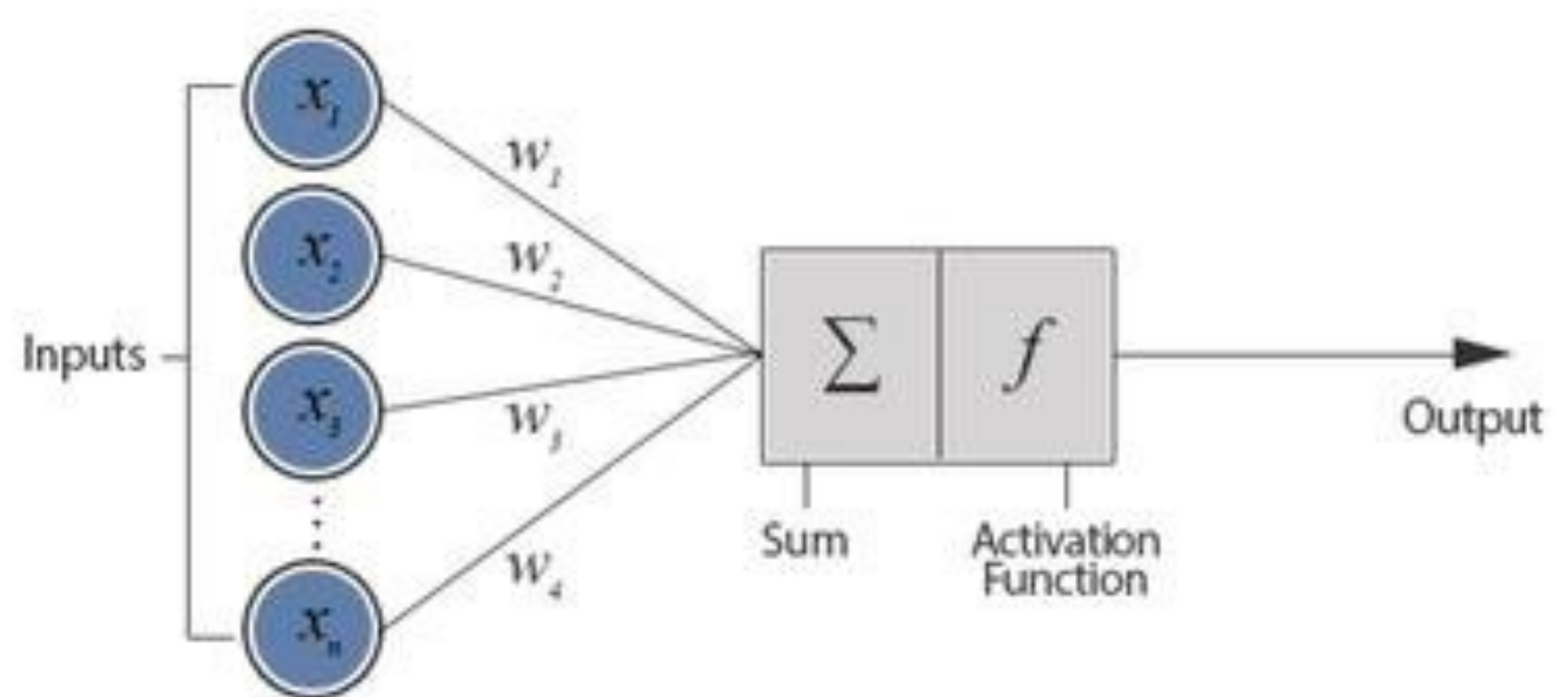


Inventé par Dr. Rosenblatt en **1957**

Perceptron et Mathématiques

- Réunir l'influence de plusieurs variables à une dimension unique.
- Très similaire à la régression linéaire, avec une fonction d'activation en plus.
- La fonction d'activation est une fonction qui contraint la somme entre $[0, 1]$.

$$f\left(\sum_{i=1}^n x_i w_i\right)$$



Perceptron et Régression Linéaire

- On retrouve une structure similaire à la regression linéaire à qui on applique une fonction d'activation :
 - Les variables/features
 - Un poids par feature
 - Un poids neutre optionnel w_0

$$f(w_0 + \sum_{i=1}^n x_i w_i)$$

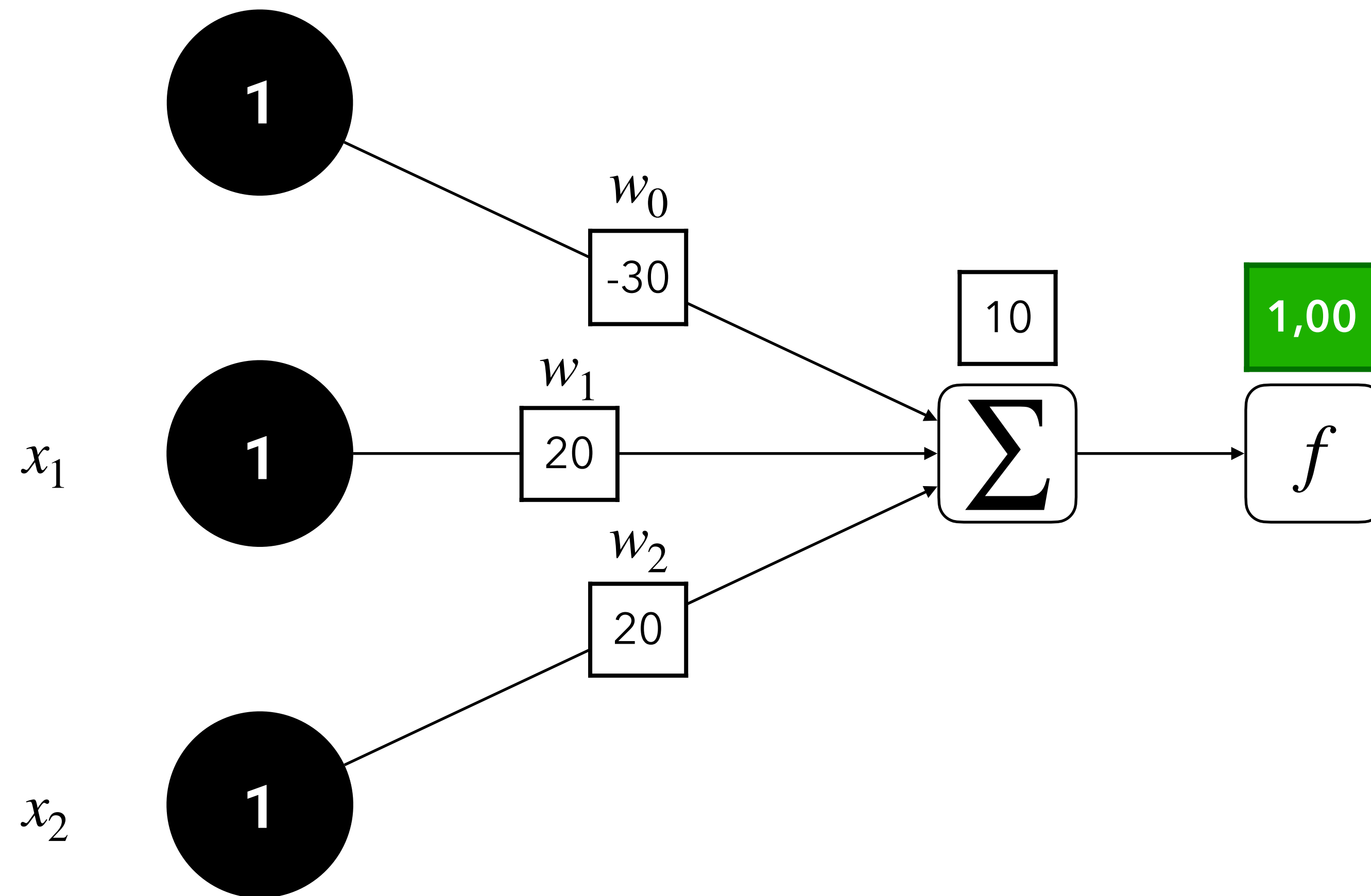
Perceptron

$$\begin{aligned} h_{\theta}(X) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \\ &= \theta_0 + \sum_{i=1}^n \theta_n x_n \end{aligned}$$

Régression Linéaire

Perceptron et Logique

Créer une porte logique grâce au perceptron



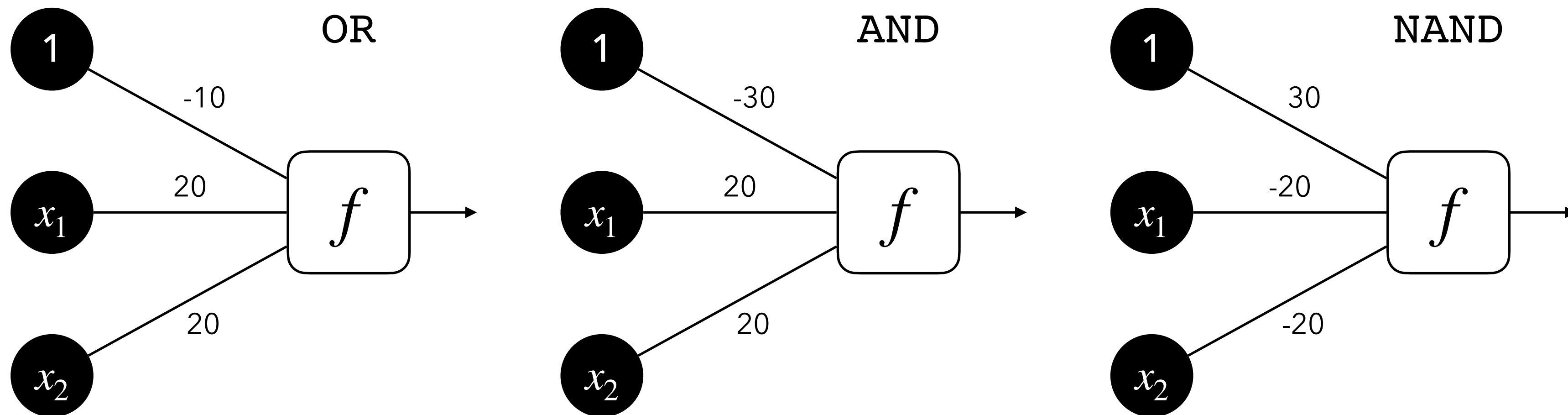
x_1	x_2	??
0	0	0
0	1	0
1	0	1
1	1	0

Perceptron et Logique

En prenant $f(z) = \frac{1}{1 + e^{-z}}$ qui est la fonction sigmoïde tel que

For $z \geq 10$, $f(z) \rightarrow 1$
For $z \leq -10$, $f(z) \rightarrow 0$

En prenant des inputs de valeurs **0** ou **1**,
On peut construire 3 types d'opérateurs logique

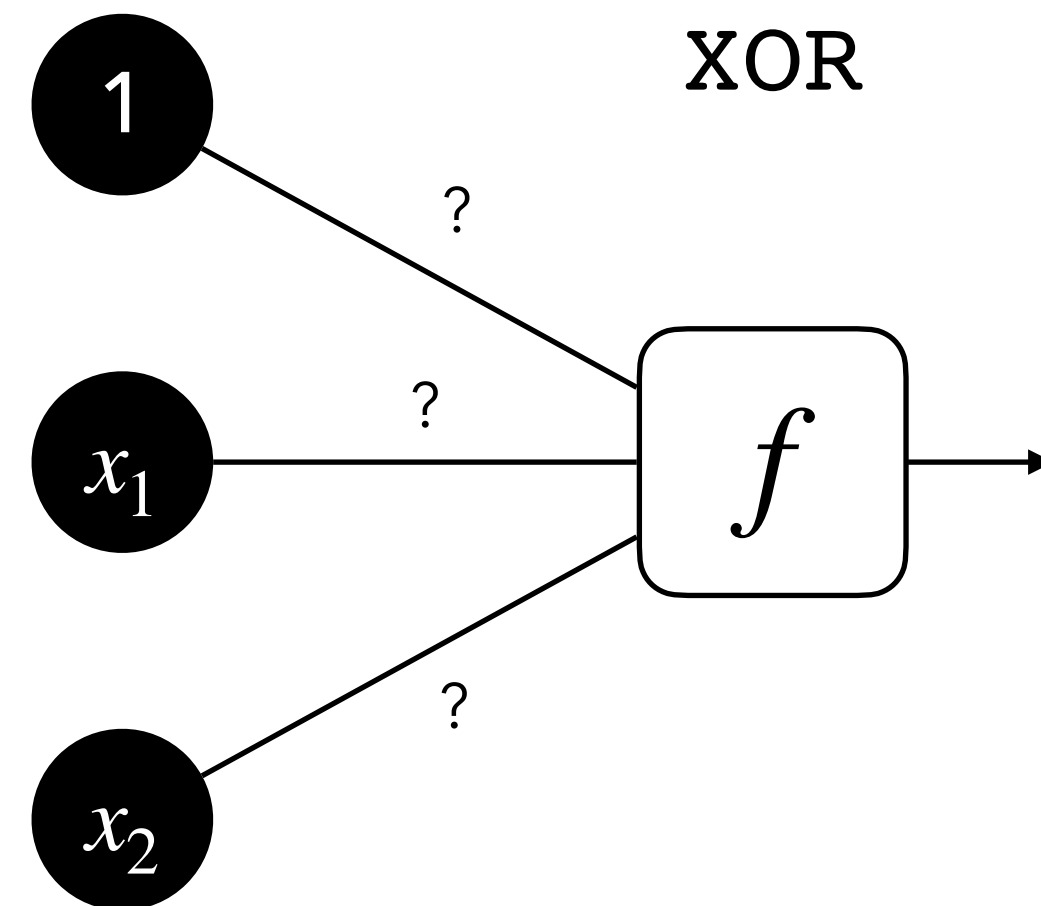


x_1	x_2	OR	AND	NAND
0	0	0	0	1
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Perceptron et Logique

Mais... impossible de construire la porte XOR avec un perceptron...

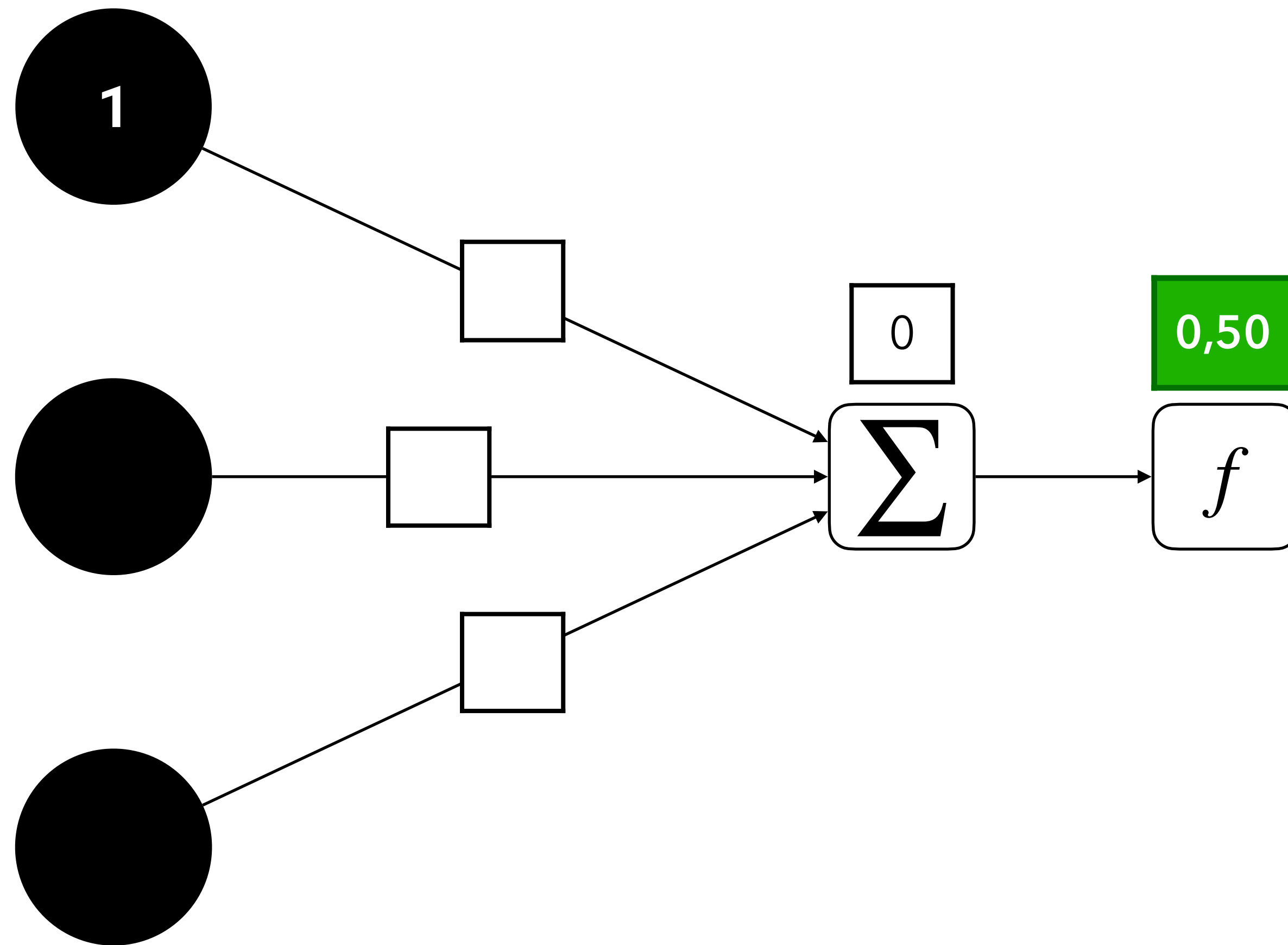
XOR ne peut pas être obtenu avec une opération linéaire



x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Perceptron et Logique

XOR ne peut pas être obtenu avec une opération linéaire

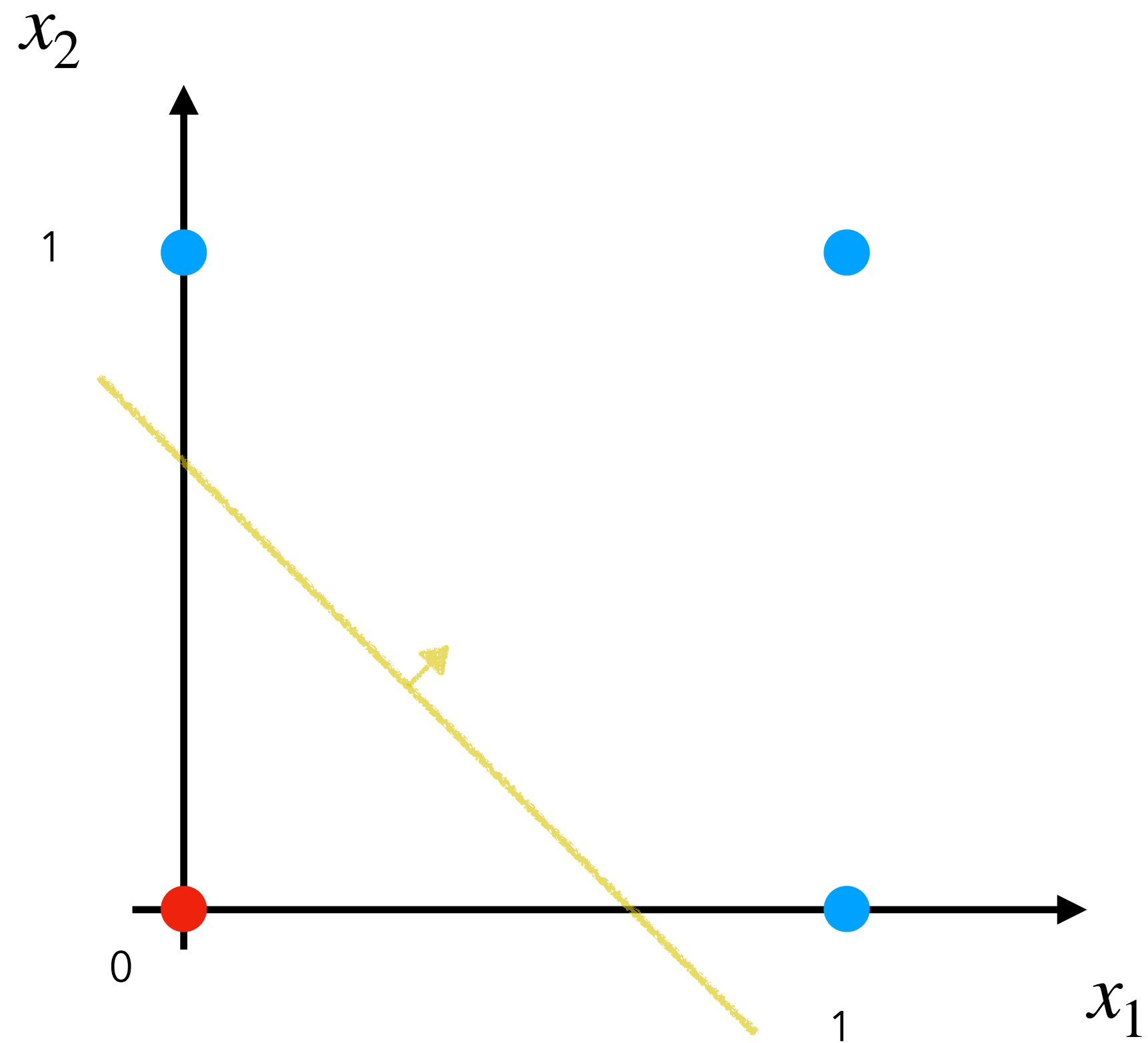


x_1	x_2	Gate
0	0	
0	1	
1	0	
1	1	

Perceptron et Logique

Intuition

OR

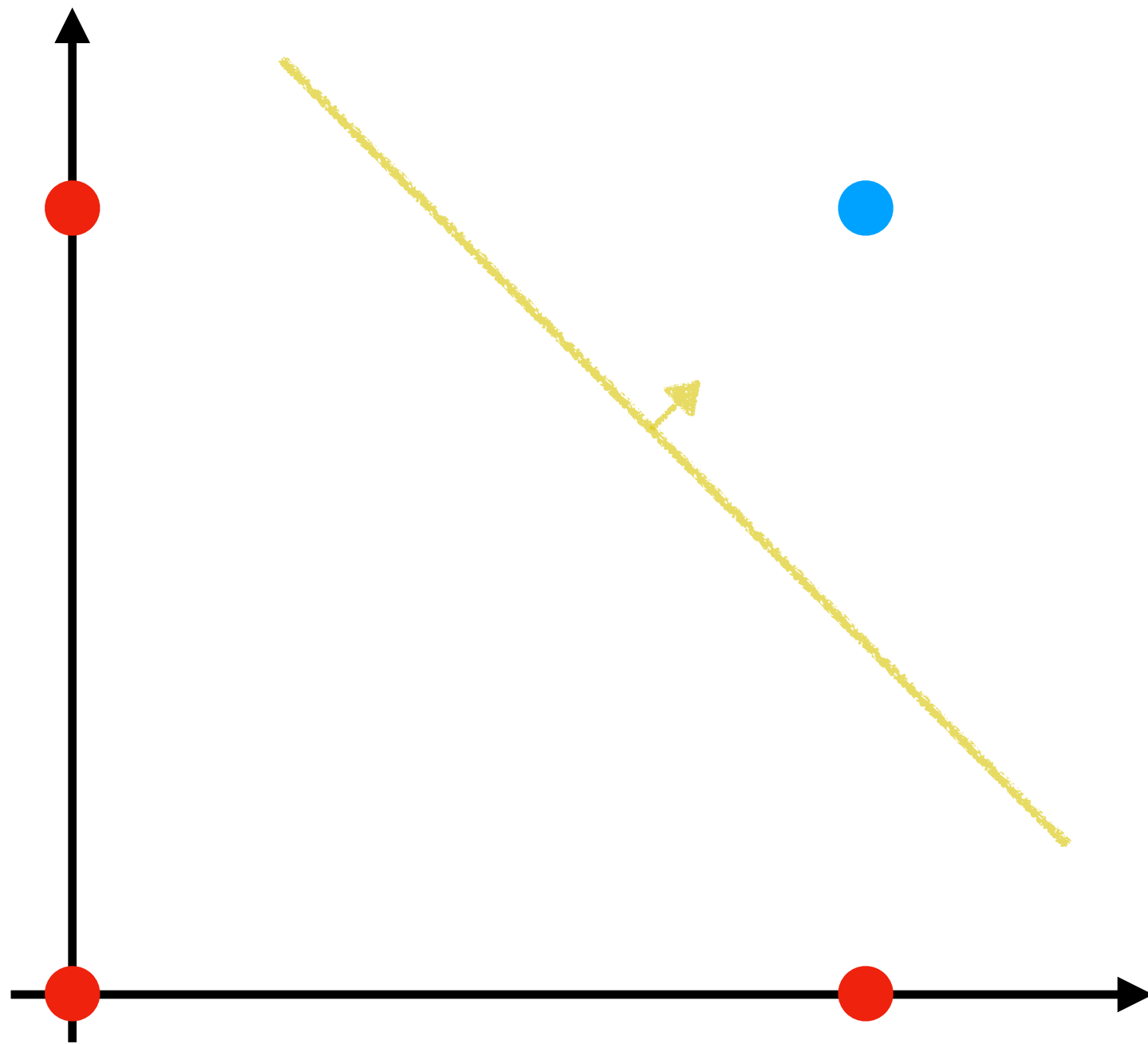


x_1	x_2	OR	AND	XOR
0	0	0	0	0
0	1	1	1	1
1	0	1	0	1
1	1	1	1	0

Perceptron et Logique

Intuition

AND

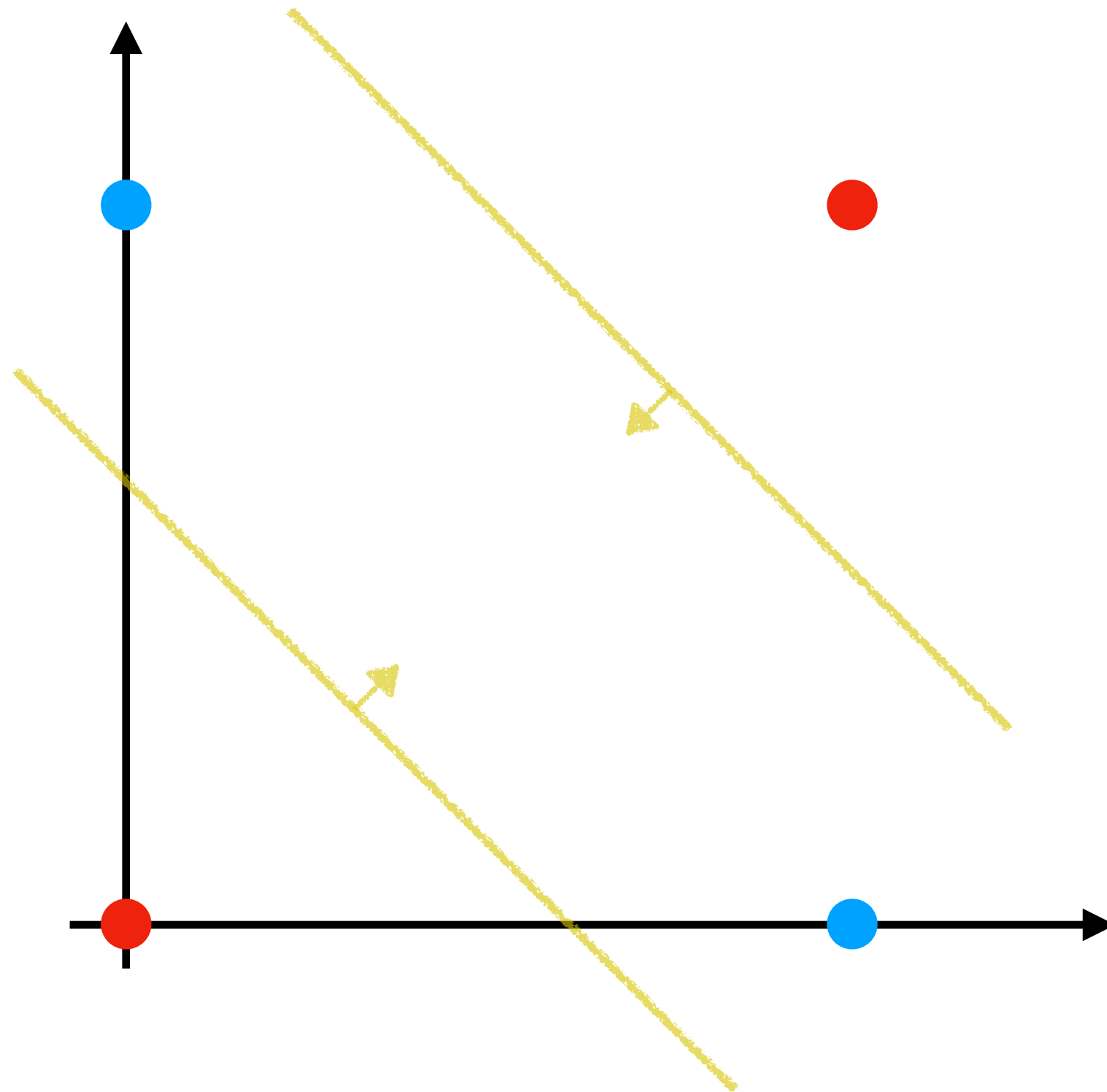


x_1	x_2	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Perceptron et Logique

Intuition

XOR

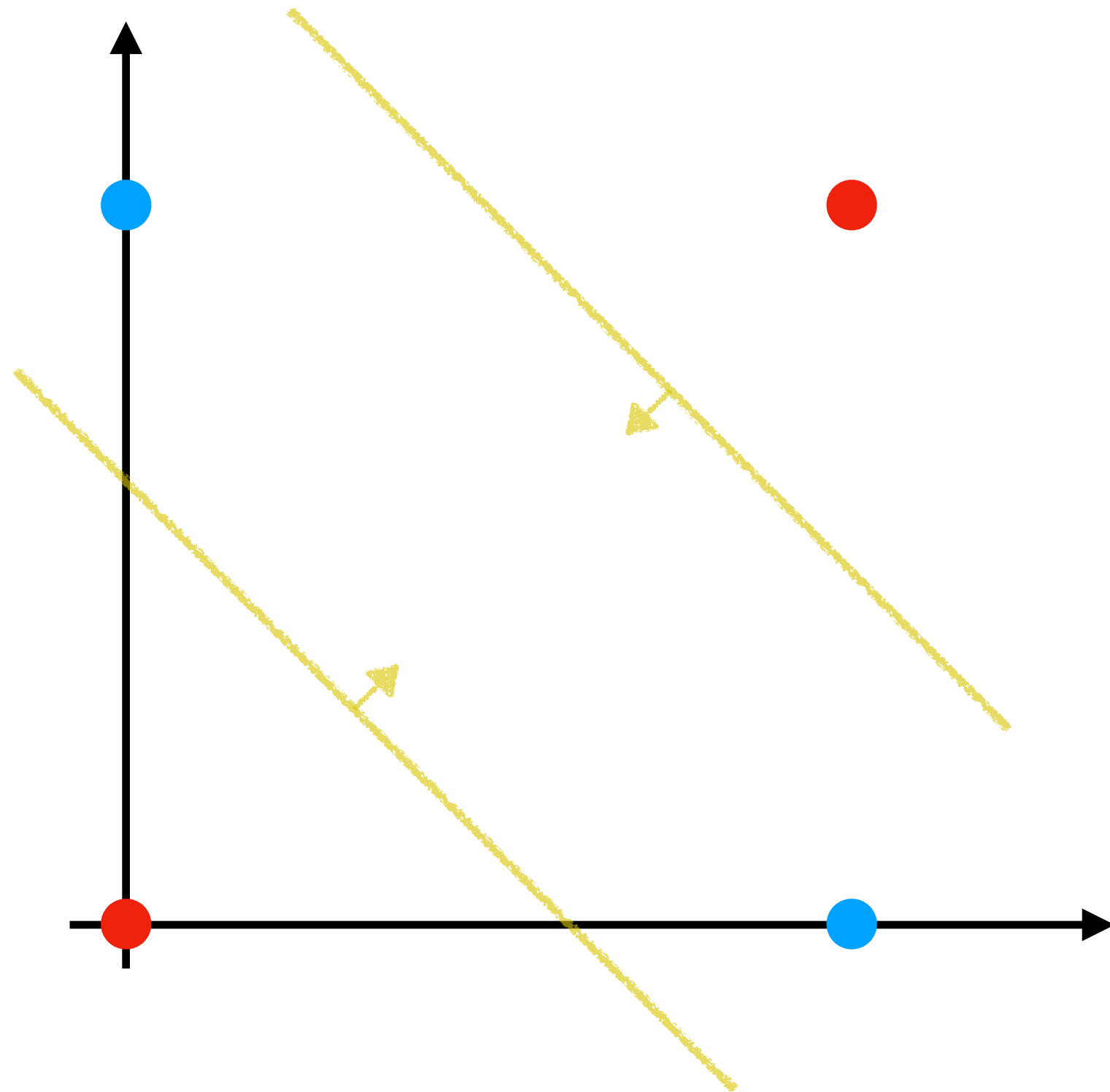


x_1	x_2	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Perceptron et Logique

Intuition: Impossible de séparer avec une seule ligne !

XOR

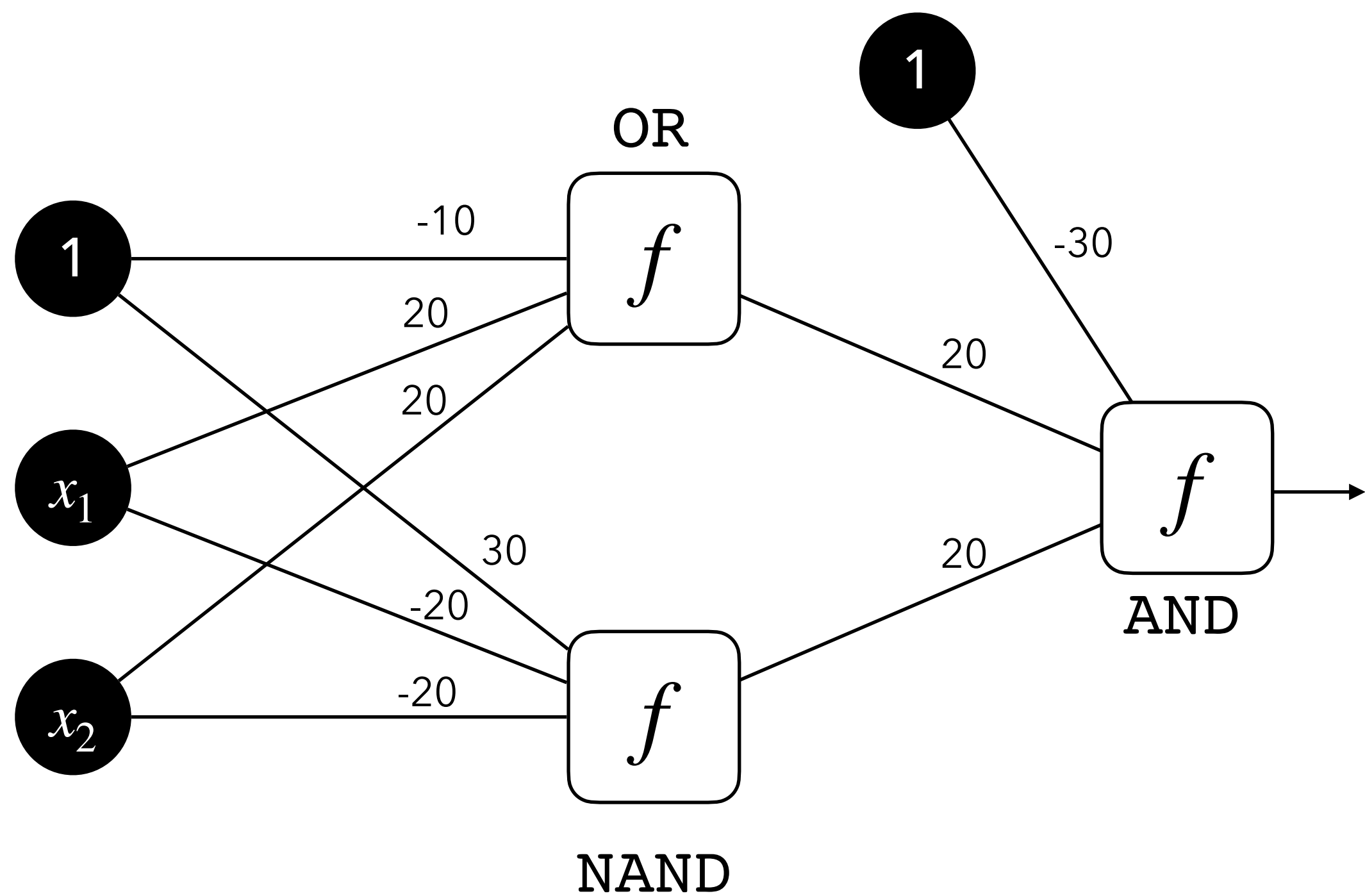


x_1	x_2	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Perceptron et Logique

Mais possible avec plusieurs perceptrons !

Rajoutons alors une deuxième ligne...



		OR	NAND	AND
0	0	0	1	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

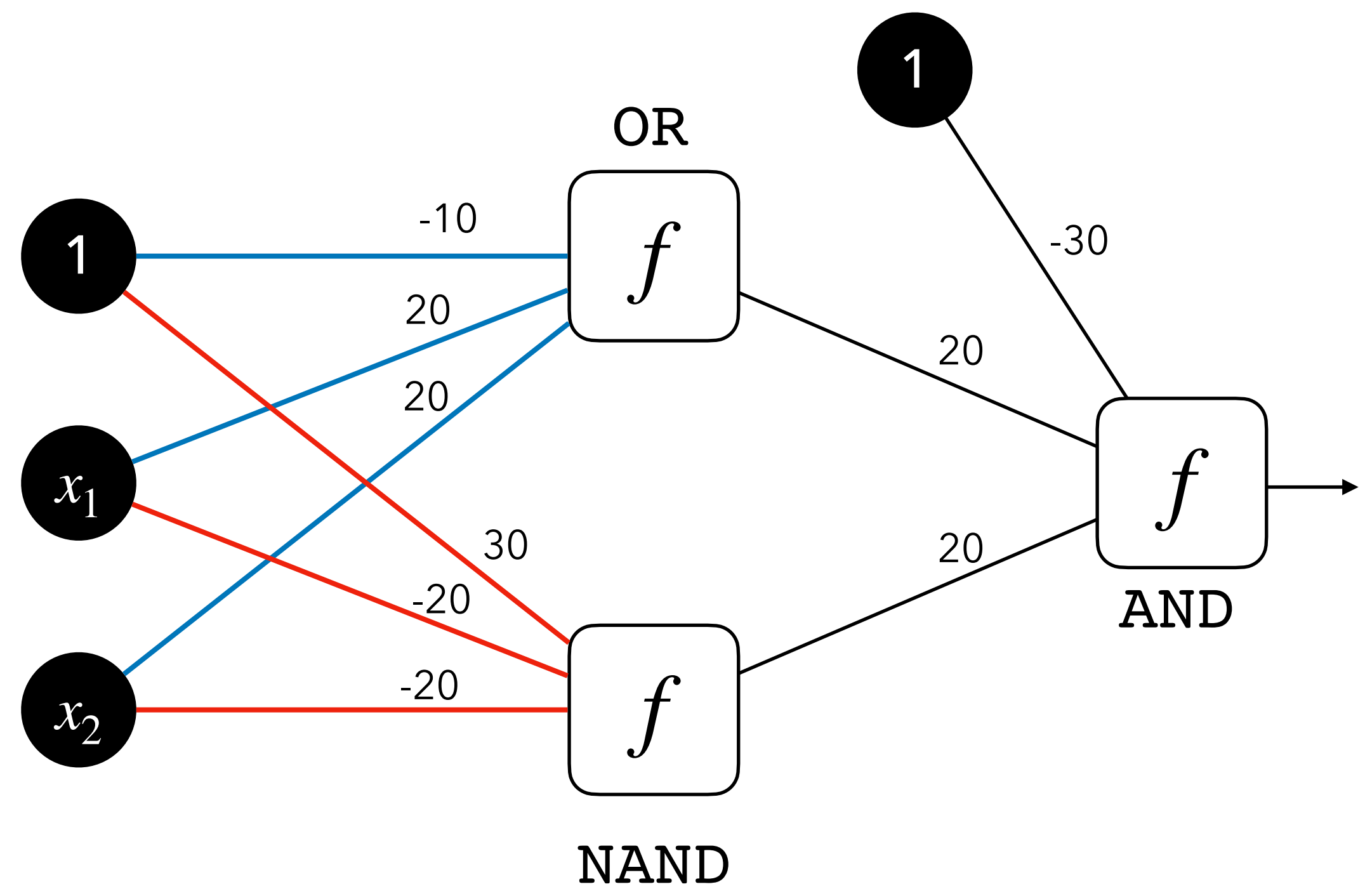
x_1	x_2	OR	AND
		NAND	
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Perceptron et Logique

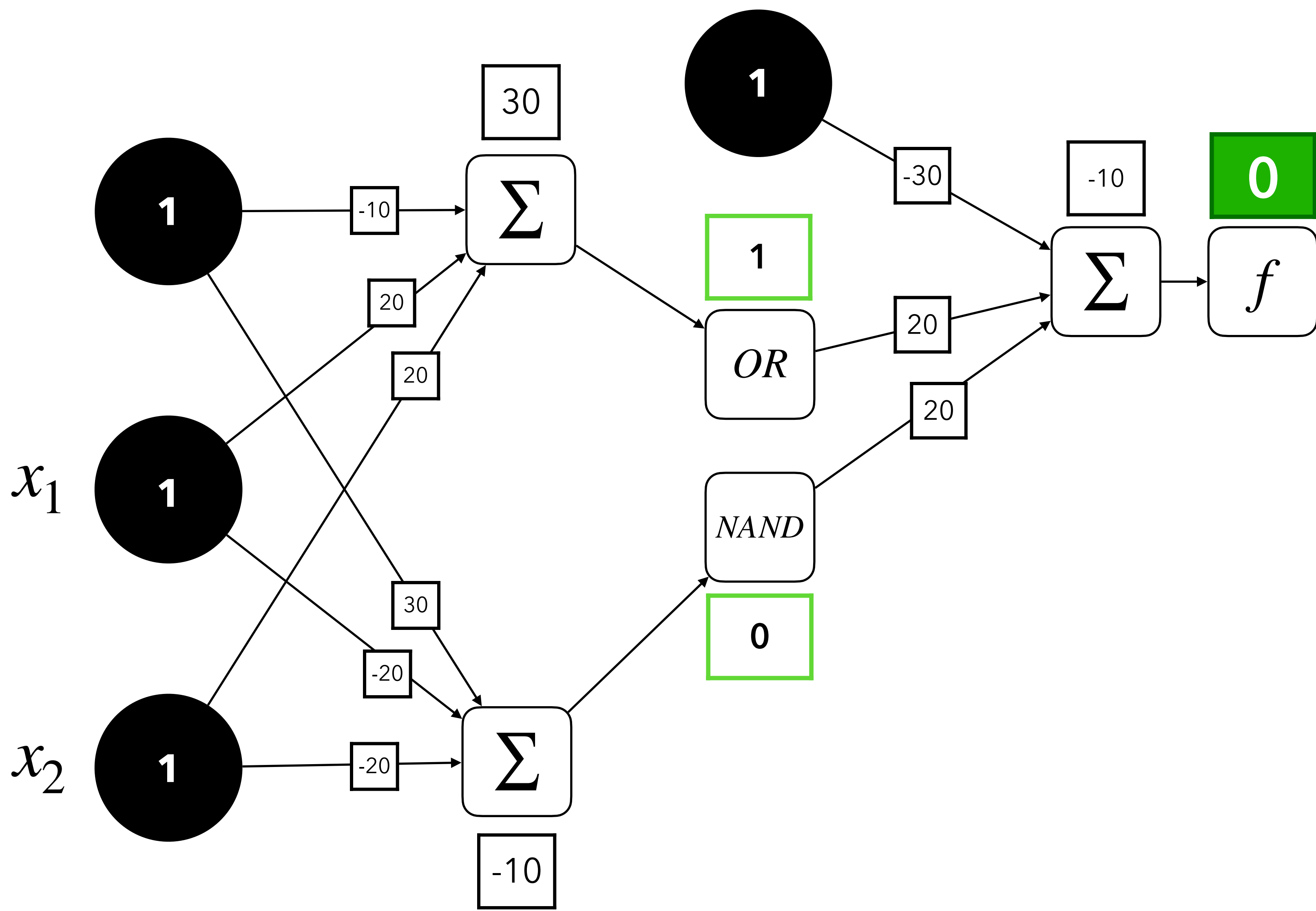
Mais possible avec plusieurs perceptrons !

Rajoutons alors une deuxième ligne...

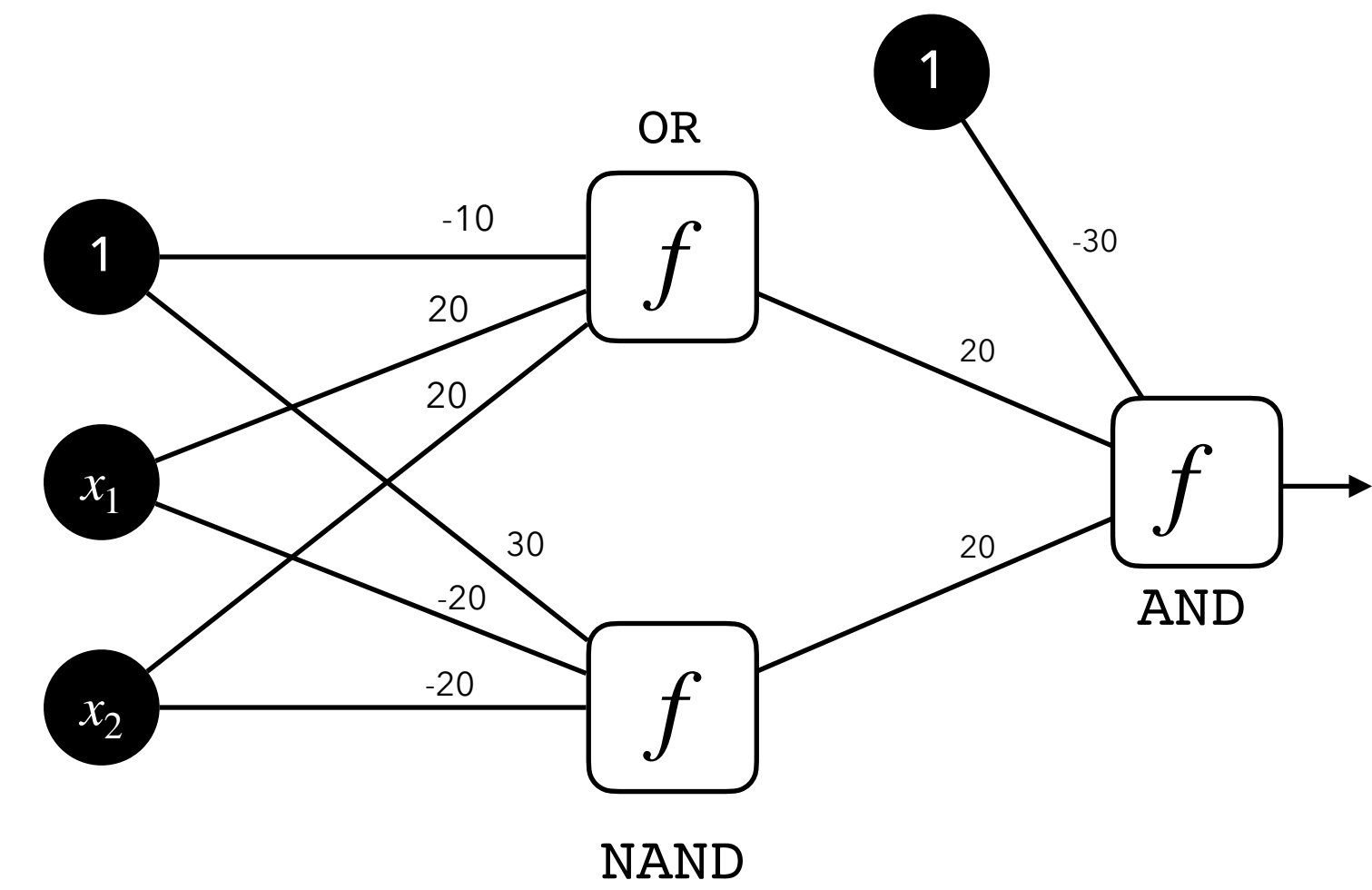
		OR	NAND	AND
0	0	0	1	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1



x_1	x_2	OR	AND
		NAND	
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

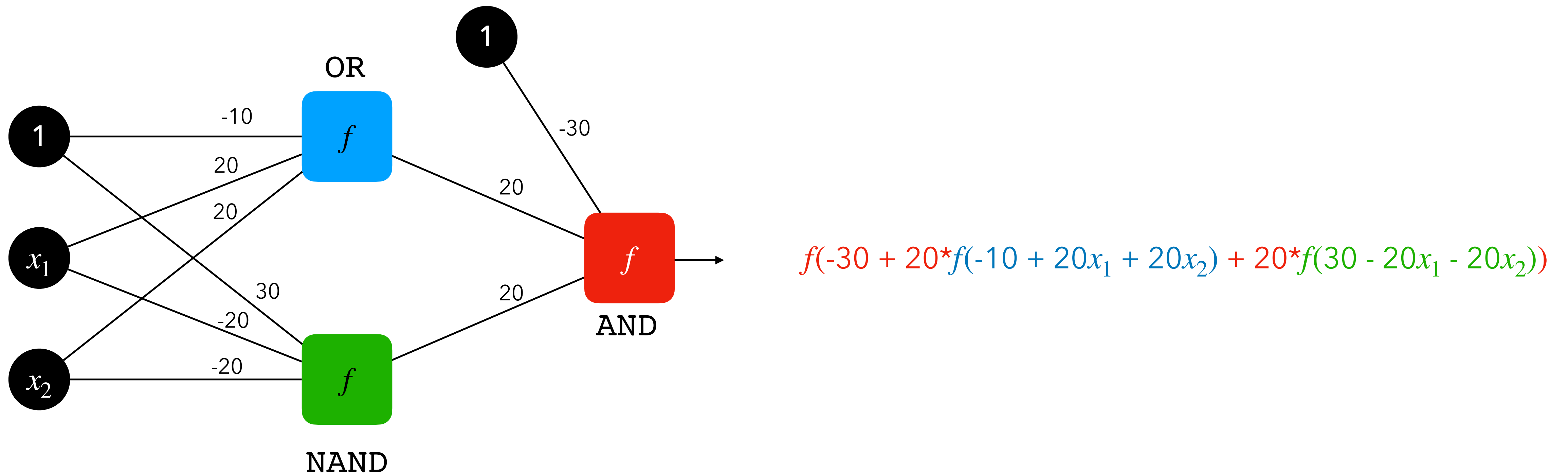


x_1	x_2	OR	AND
		NAND	
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0



Perceptron et Logique

- En manipulant les opérateurs logique dans l'ordre qu'on veut, on peut reproduire n'importe quel opérateurs.



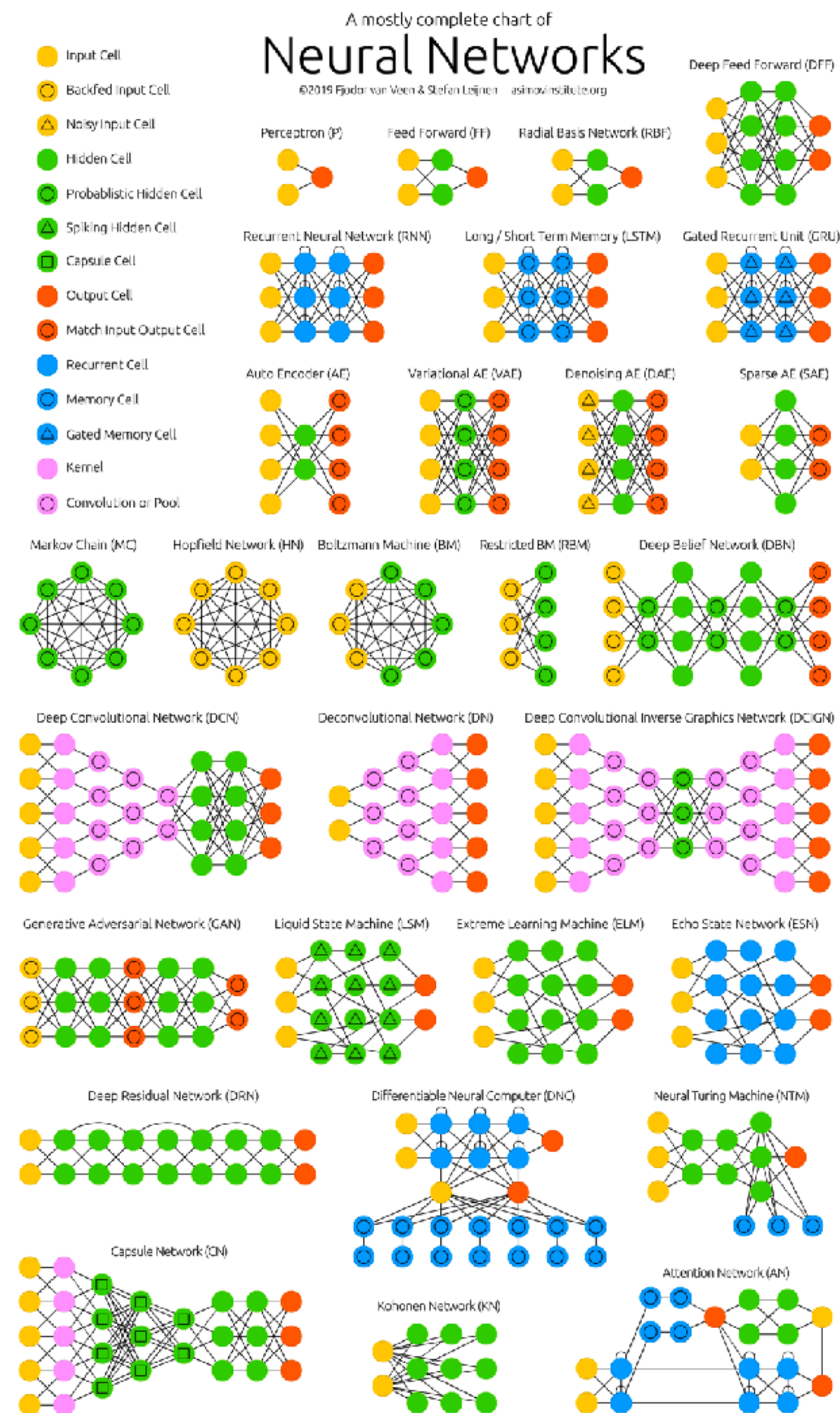
Neural Networks

Créer assez de complexité pour pouvoir tout modéliser

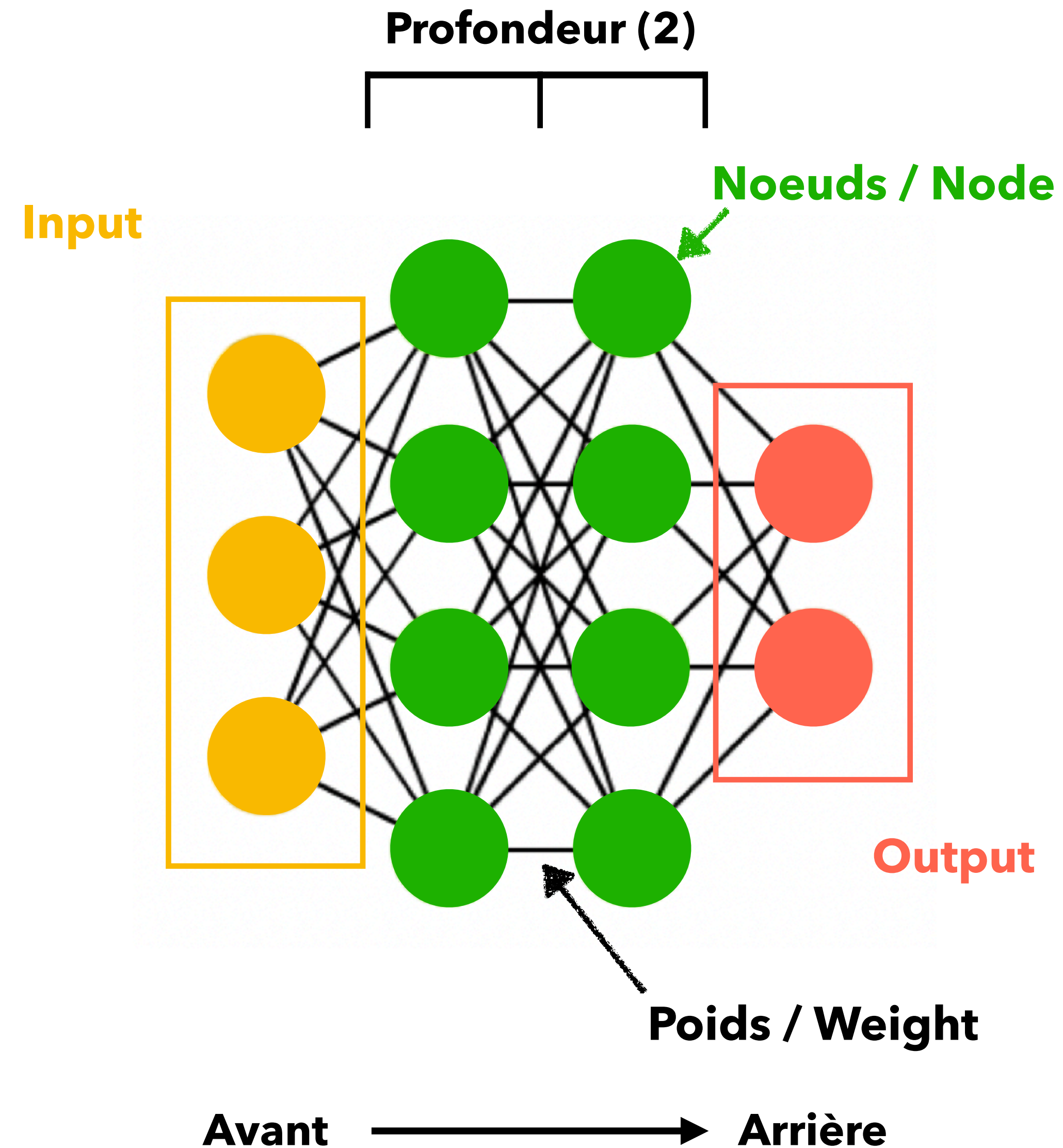
The Neural Network Zoo

- Depuis la montée en popularité du machine learning, la famille des Réseaux Neuronaux à fortement grandi (même si beaucoup ont pas mal d'années d'existences).
- Beaucoup de ces familles de réseaux sont présents et/ou inspirées par les réseaux neuronaux du cerveau.
- Une description de chaque famille est présent dans le lien.

<https://www.asimovinstitute.org/neural-network-zoo/>



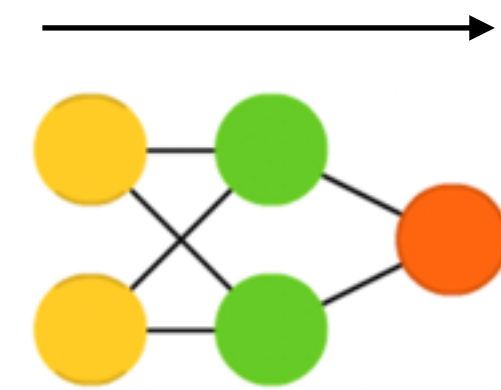
Anatomies d'un réseau neuronale



Types de NN (Neural Networks)

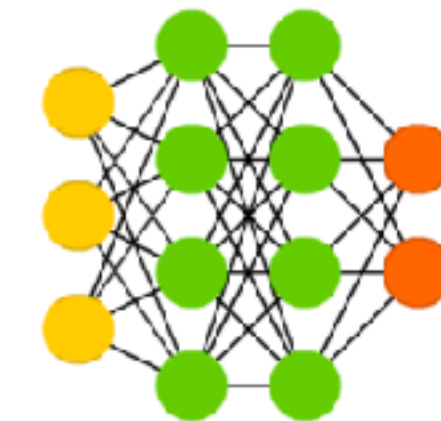
- Feed forward NN (FFNN)

- L'information va de l'avant à l'arrière, sans boucles.



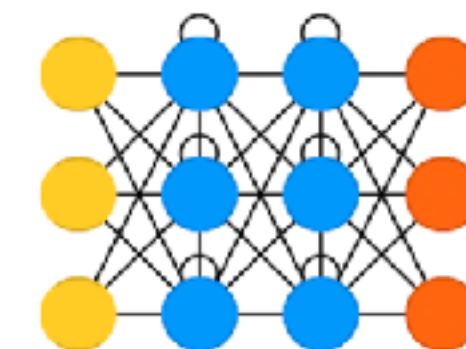
- Deep Feed Forward (DFF)

- Aussi connu sous le nom "*Deep Neural Network*" ou le terme "*Deep Learning*".
- Un FFNN avec de nombreuses couches et de noeuds.



- Recurrent NN (RNN)

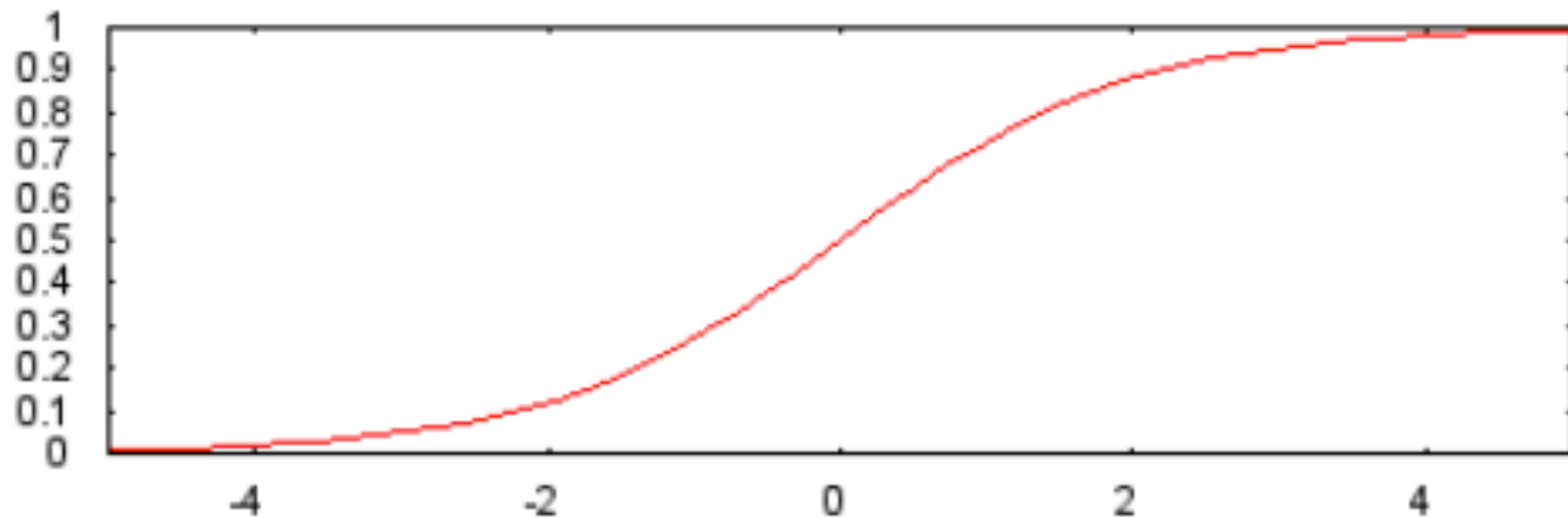
- Même famille que les DFF mais où certains noeuds ont un 'état'.
- On inclut certaines features précédentes à chaque passe rendant l'ordre des features important.



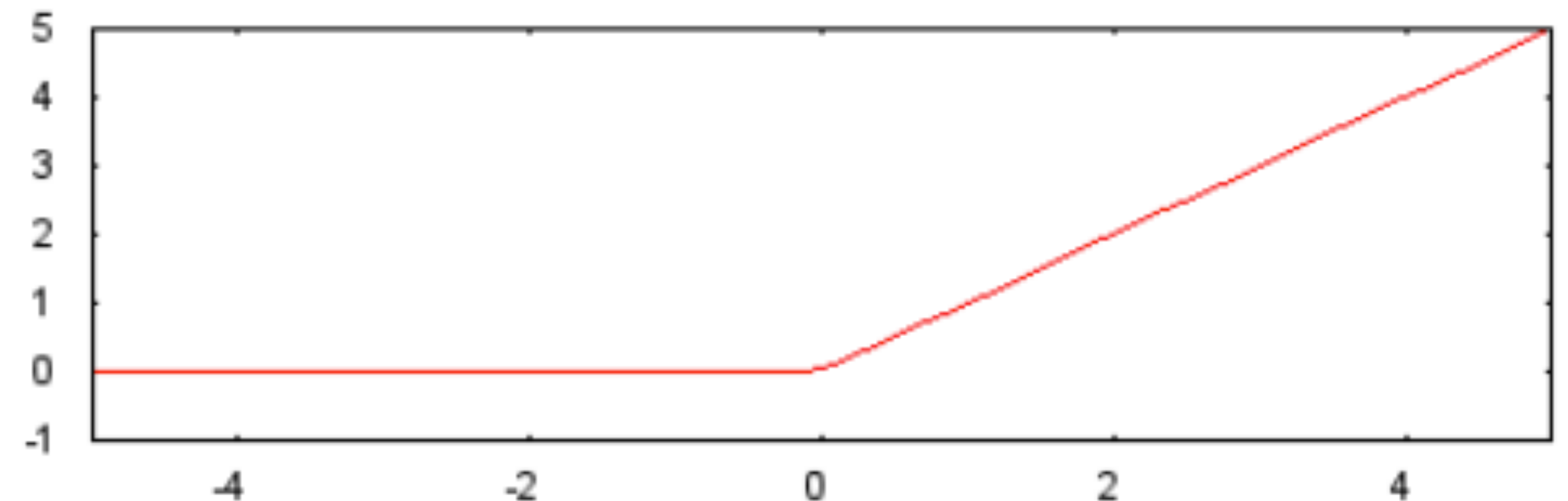
Fonction d'activation

- **Chaque noeuds** possède une fonction d'activation, "activant" ou non la sommation des poids. Il en existe beaucoup mais voici deux fonctions très populaire.

Fonction logistique



Fonction ReLu



Cost Function

- Très similaire à la régression, on calcule le "coût" ou l'erreur en prenant la distance Euclidienne entre la prédiction et l'objectif. L'équation dépend de la fonction d'activation et du type d'objectif.

Régression Linéaire

$$J = \frac{1}{2m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

J : le coût
 \hat{y} : la prédiction
 y : l'objectif
 m : le nombre de valeurs total

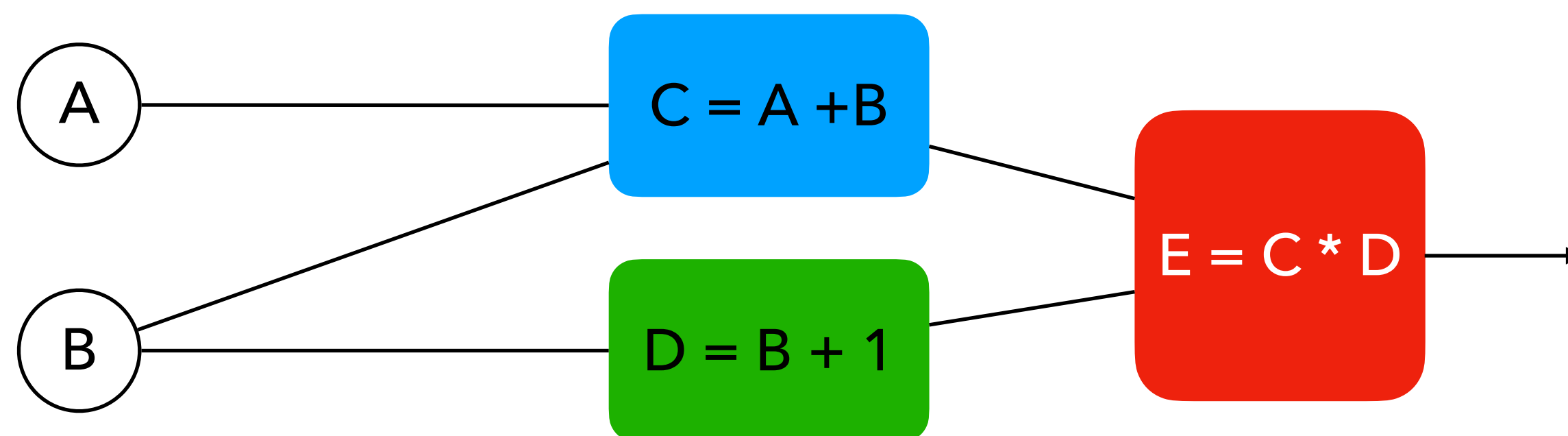
Régression Logistique

$$J = -\frac{1}{m} \sum_{i=1}^m [y^i \log(h_{\theta}(x^i)) + (1 - y^i) \log(1 - h_{\theta}(x^i))]$$

J : le cout
 y : l'objectif
 m : le nombre de valeurs

Gradient Descent

- Le Gradient Descent permet de trouver à travers une dérivée du coût, une direction pour réduire le coût.
- Mais dans le cas du machine learning, le nombre et la complexité de relation des poids rend une dérivée brute trop coûteuse (d'un point de vue de complexité algorithmique : relation factorielle)



Exemple avec un tout petit réseau neuronale

$$E = C * D$$

$$E = (A + B) * (B + 1)$$

Comment un changement dans B modifie E ?
Un changement dans B affecte-t-il tout les
chemin jusque E ?

Back Propagation

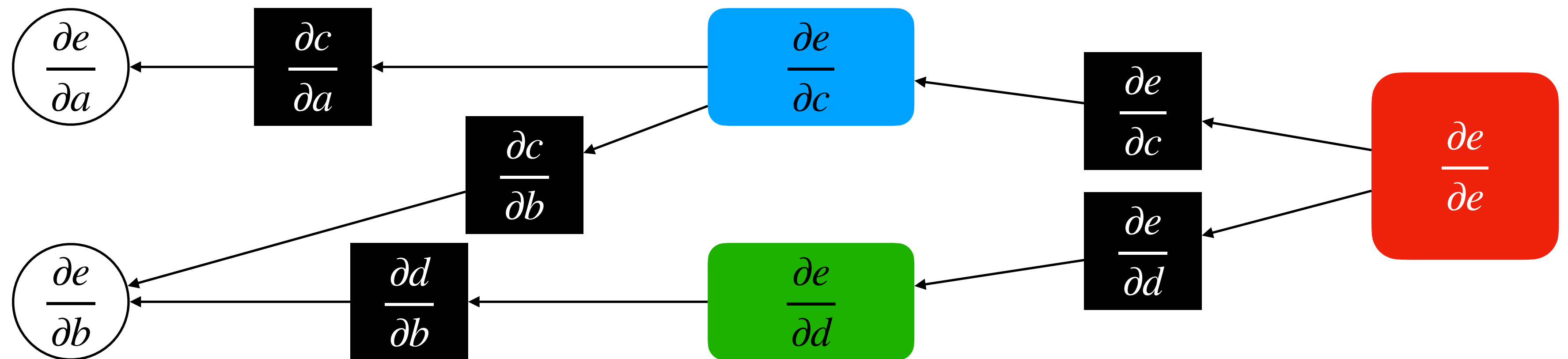
La propagation de l'erreur

- On calcule la dérivée à chaque noeuds et poids en commençant par la fin et on remonte jusqu'à l'input.
- Quand on dérive par rapport à un poids, on ne prend en compte que le poids précédent.
- En une seule passe on trouve toutes les dérivées que l'on veut.
(Complexité algorithmique linéaire)

Chain Rule

$$\frac{\partial e}{\partial c} * \frac{\partial c}{\partial a} = \frac{\partial e}{\partial a}$$

$$\frac{\partial e}{\partial c} * \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} * \frac{\partial d}{\partial b} = \frac{\partial e}{\partial b}$$



Back Propagation + Gradient Descent

- On arrive à relier le résultat finale par rapport à chaque input.
- On calcule le Gradient Descent à chaque poids pour réduire le coût et une fois qu'on a trouvé toutes les valeurs, on modifie tout les poids d'un coup.

Chain Rule

$$\frac{\partial e}{\partial c} * \frac{\partial c}{\partial a} = \frac{\partial e}{\partial a}$$

$$\frac{\partial e}{\partial c} * \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} * \frac{\partial d}{\partial b} = \frac{\partial e}{\partial b}$$

