



Présentation

Les réseaux de neurones artificiels

Encadré par :

Mr. NEMICHE Mohammed

Module :

Sciences des données

Présenté par :

- AJIDAD Nouhayla
- OUGNI Imane
- RIFI Maroua
- EL AGRI Meryem

- EL MTARKATI Mustapha
- RAJA ALLAH Hamza
- LAAMACHE Hajar

Situation de départ

Le cerveau humain à la naissance est une feuille vierge prête à être remplie par l'expérience.

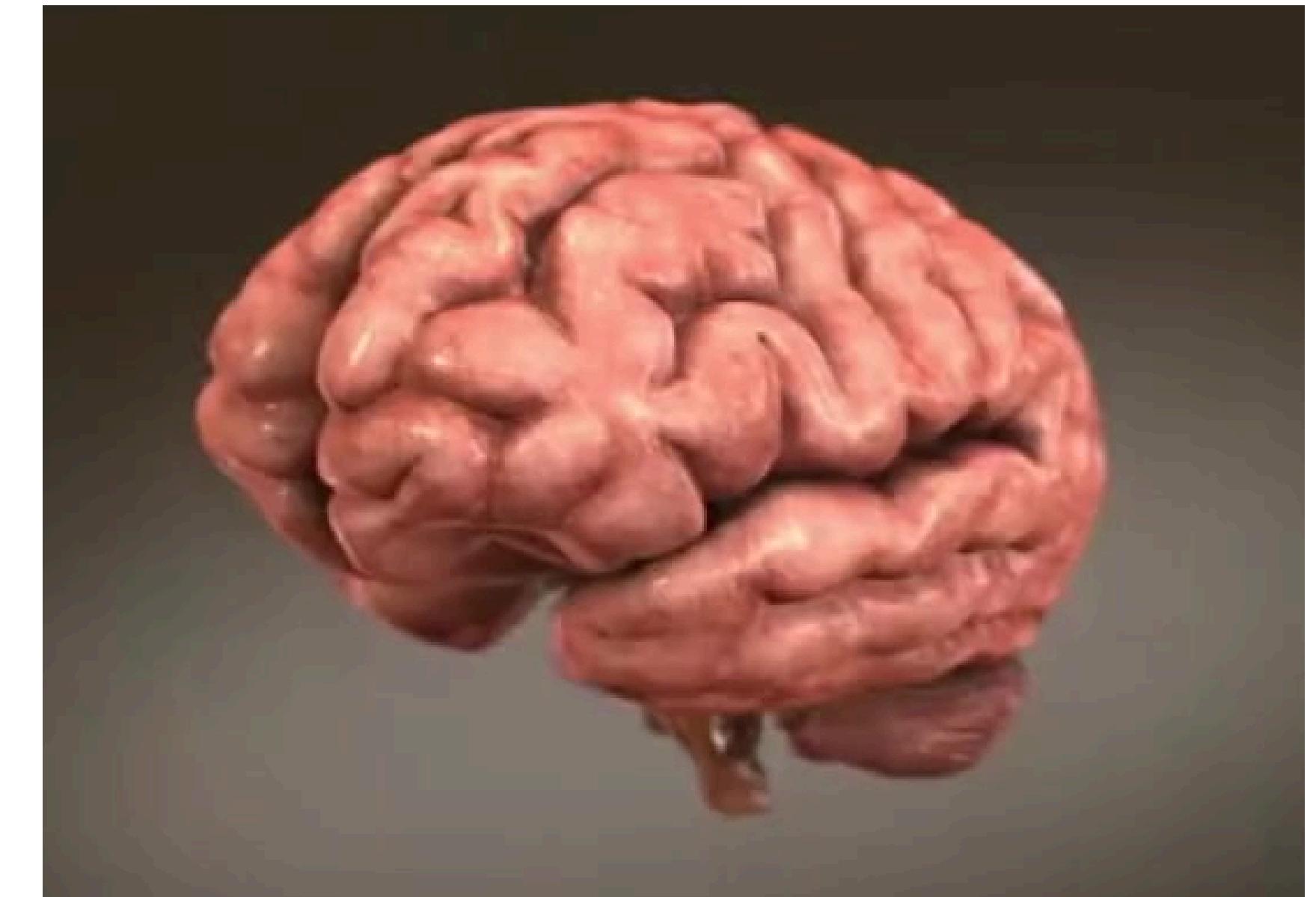




Un nourrisson, dès ses premiers jours, perçoit et interprète les stimuli autour de lui : le visage familier d'une mère, une voix douce et réconfortante. Ces interactions initiales forment les bases de son apprentissage et de sa compréhension du monde.



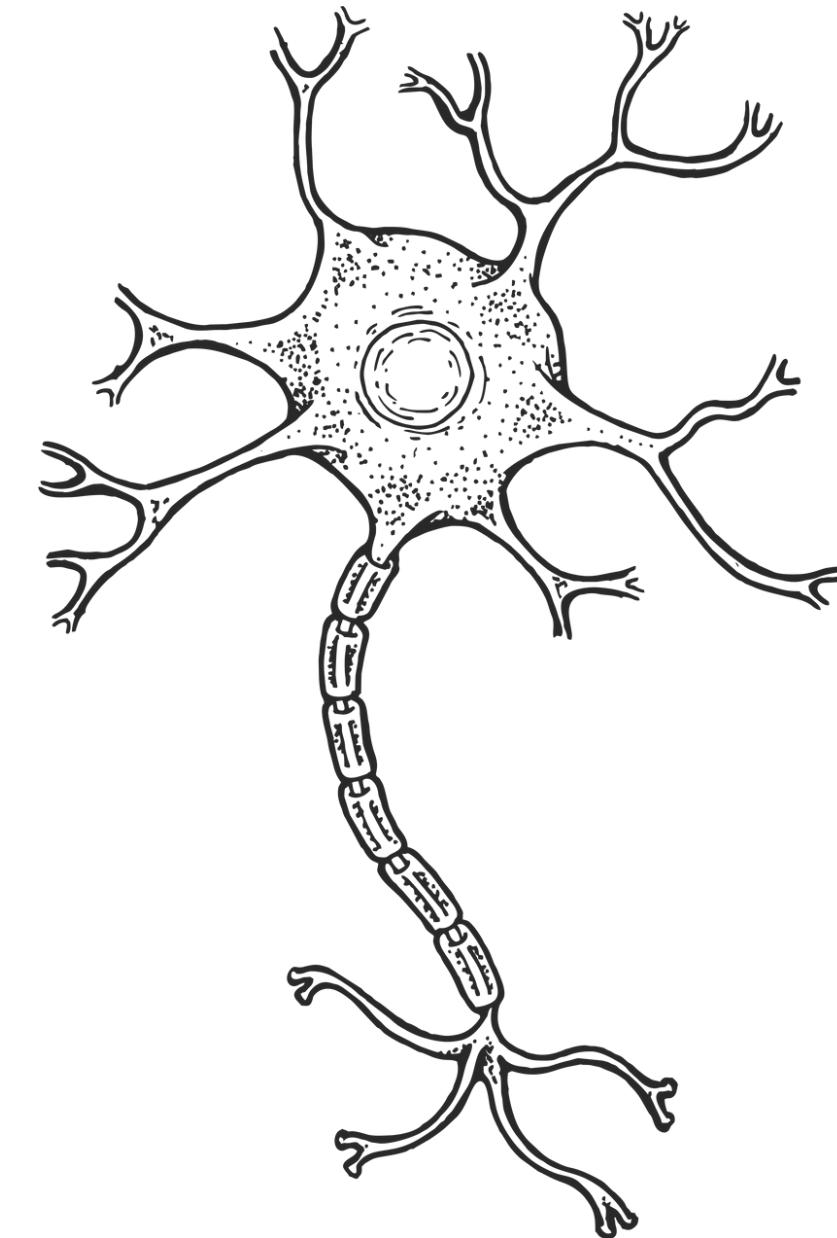
Les yeux et les oreilles captent des images et des sons, transformés en signaux électriques et envoyés au cerveau pour analyse : le cortex visuel pour les visages, le cortex auditif pour les voix.



Avec chaque répétition, les données sont consolidées, les liens renforcés, et le modèle neuronal devient plus précis.

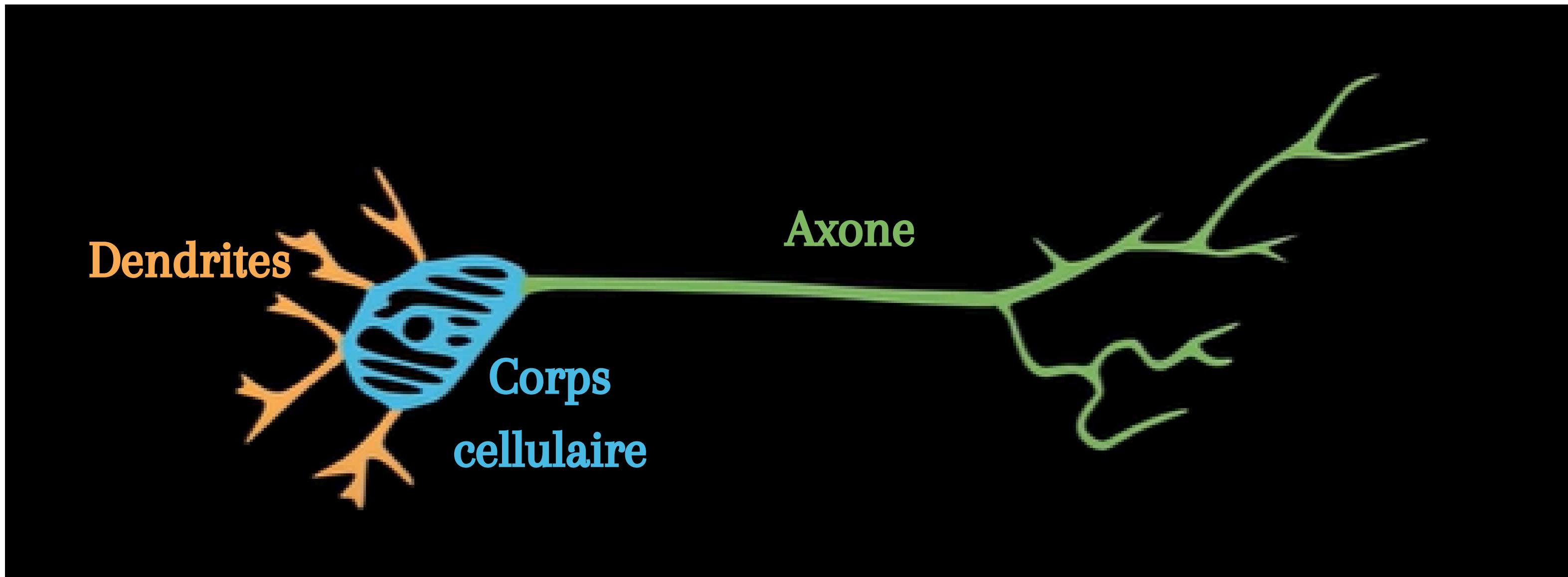


Le cerveau humain est un vaste réseau : des milliards de neurones reliés par des synapses travaillent ensemble, créant une incroyable dynamique d'interactions.



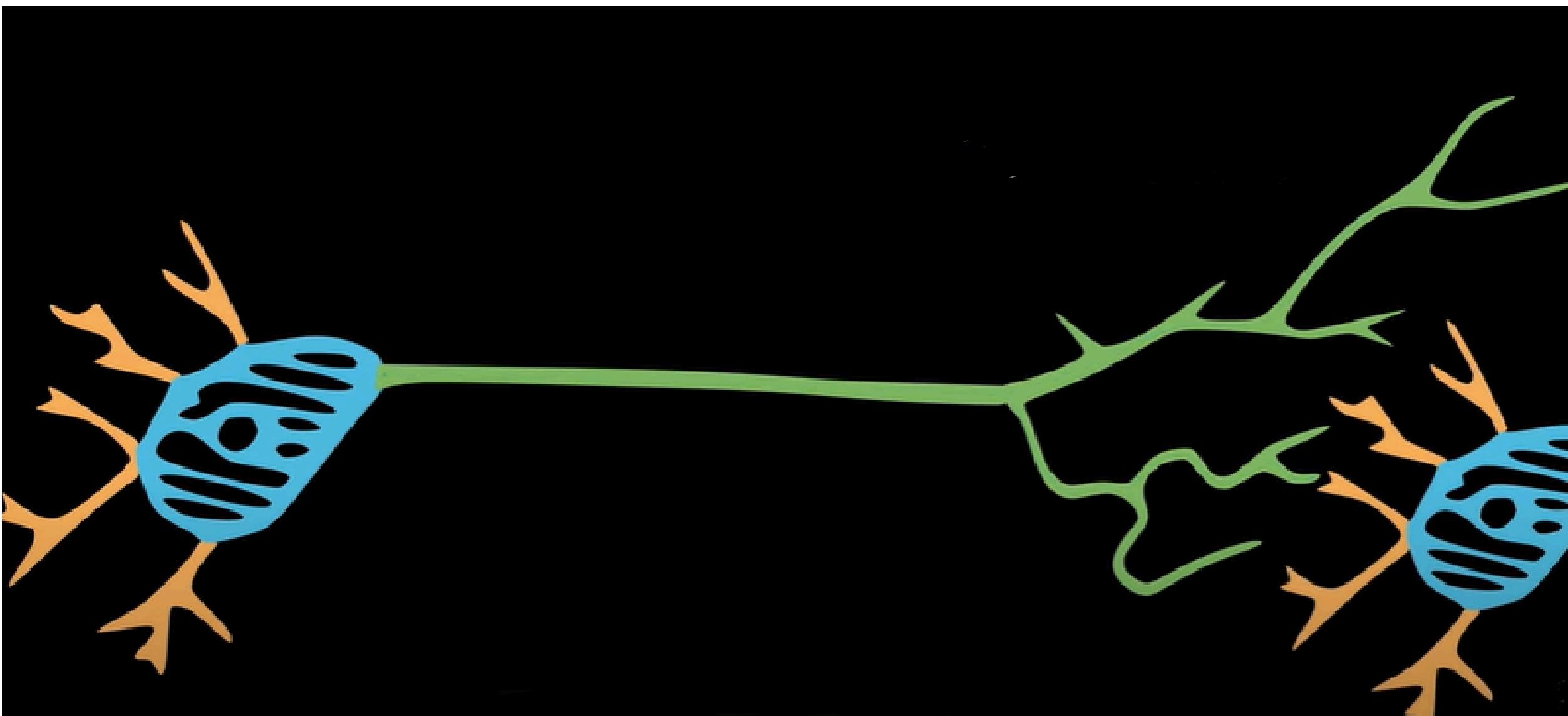
Structure d'un neurone biologique

Un neurone est une cellule nerveuse a pour rôle de faire circuler les informations, pour cela il est capable de traiter et transmettre les informations



Transmission des Informations

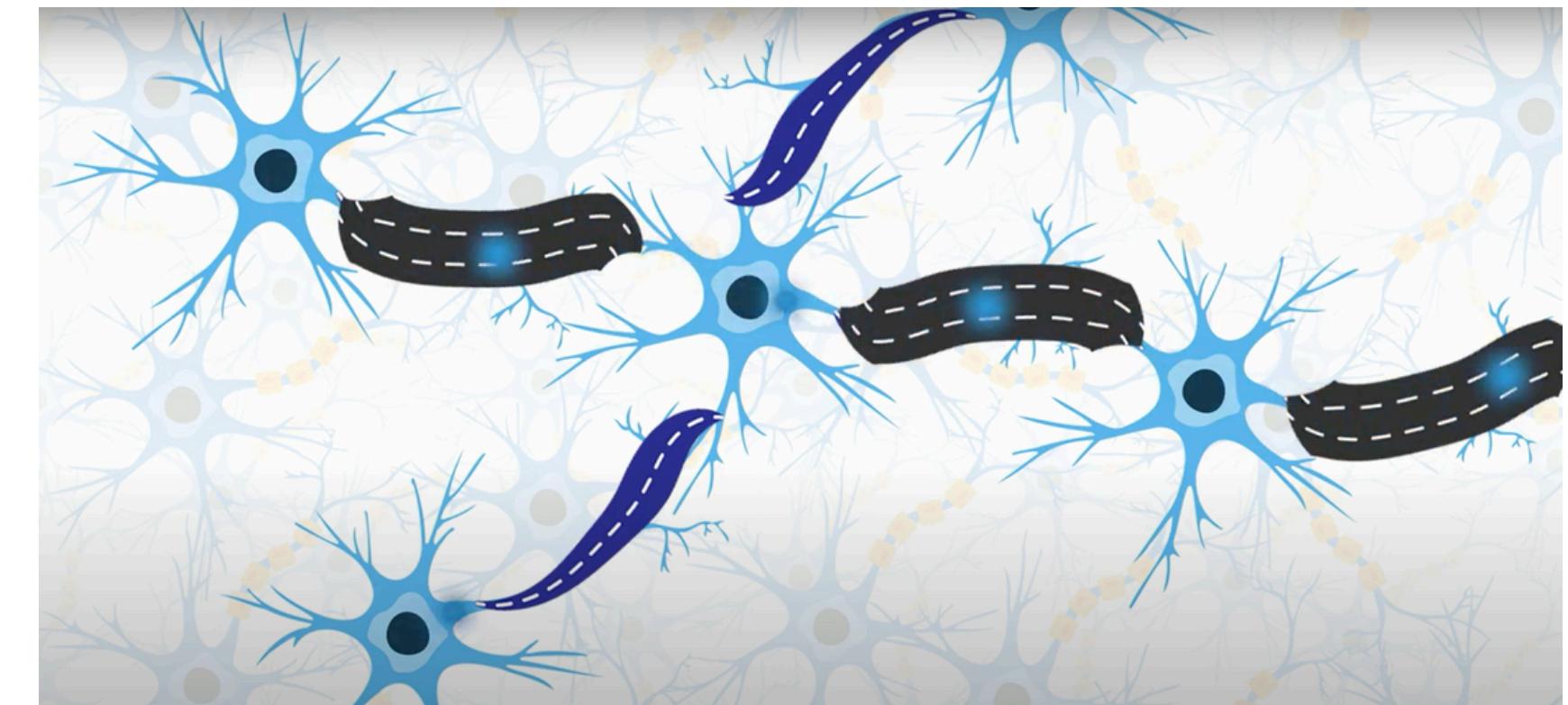
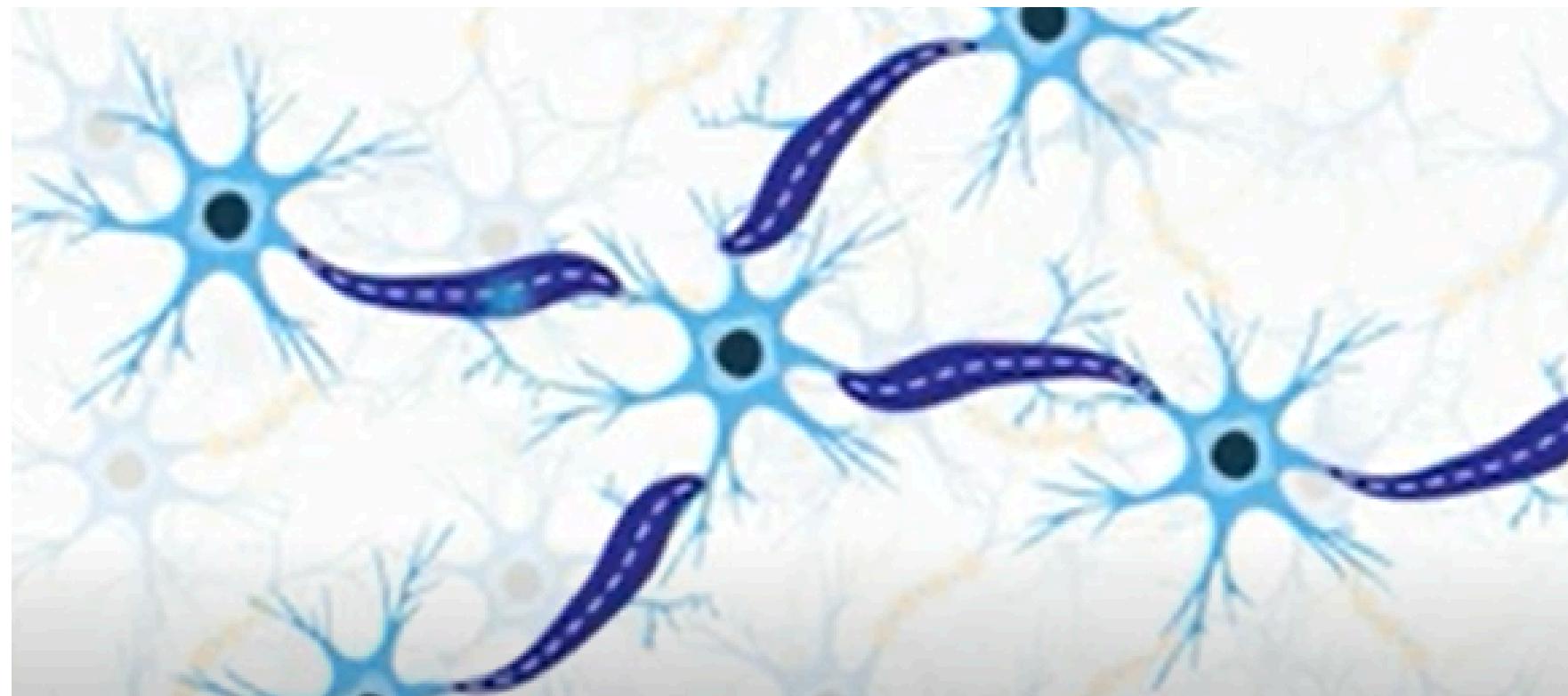
Les neurones communiquent entre eux grâce à des impulsions électriques et chimiques qui passent par les synapses. Cette transmission rapide permet au cerveau de traiter et d'analyser de grandes quantités d'informations en temps réel.



Apprentissage et connexions

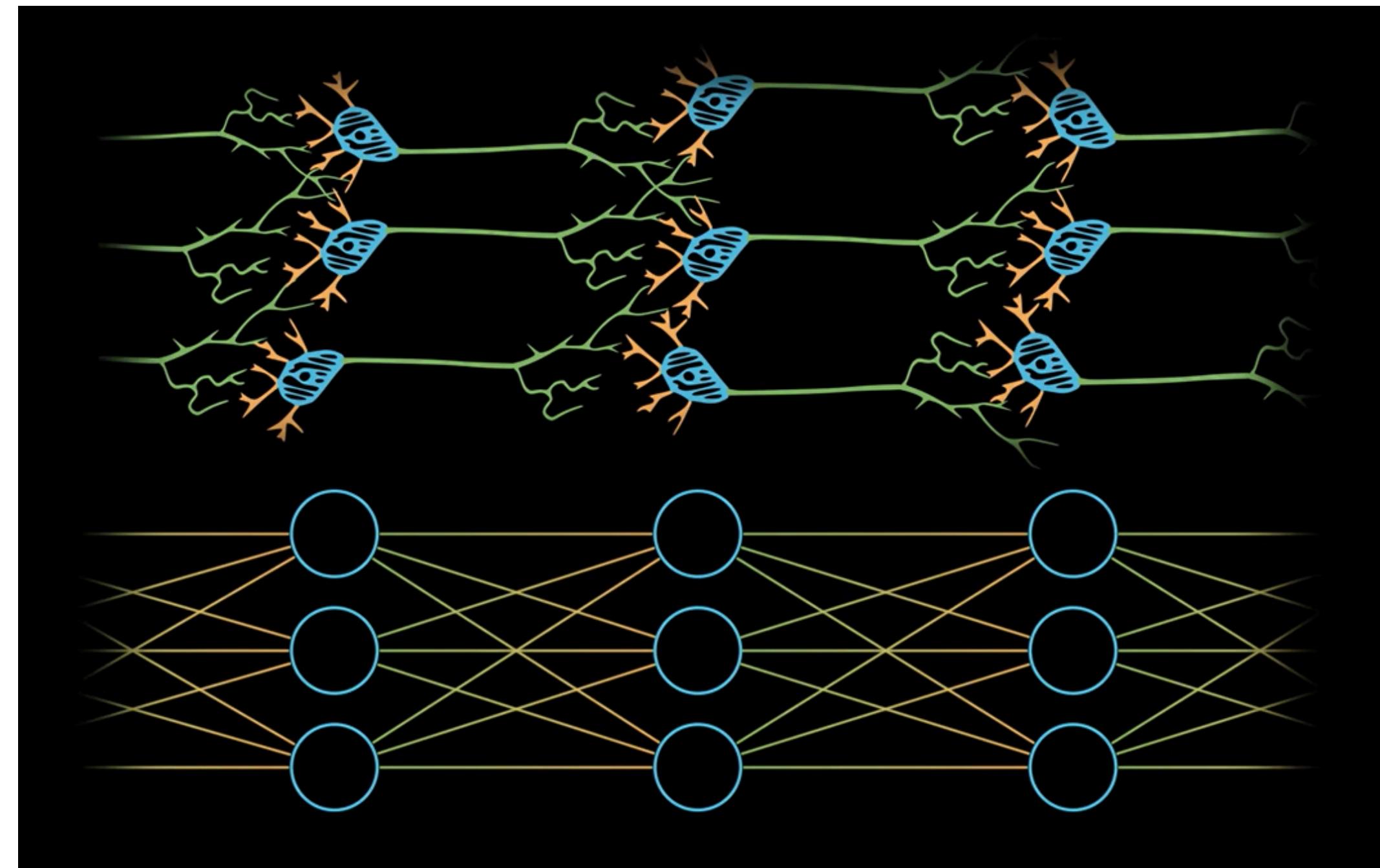
Les neurones se renforcent grâce à un processus appelé **la plasticité synaptique**, par lequel les connexions deviennent plus solides à mesure que les stimuli sont répétés.

Plus les stimuli sont répétés, plus les connexions deviennent solides.



Neurones biologiques Vs Neurones artificiels

fonctionnent sur le même principe fondamental : l'apprentissage par connexions, mais l'un est un produit de la nature, tandis que l'autre est une création humaine pour imiter cette merveille du cerveau.



Les réseaux de neurones artificiels

Définition des RNA

Histoire

Domaines d'application

Pourquoi les RNA?

Types des RNA

Perceptron simple

Perceptron multicouche

CNN

**IA/deep...

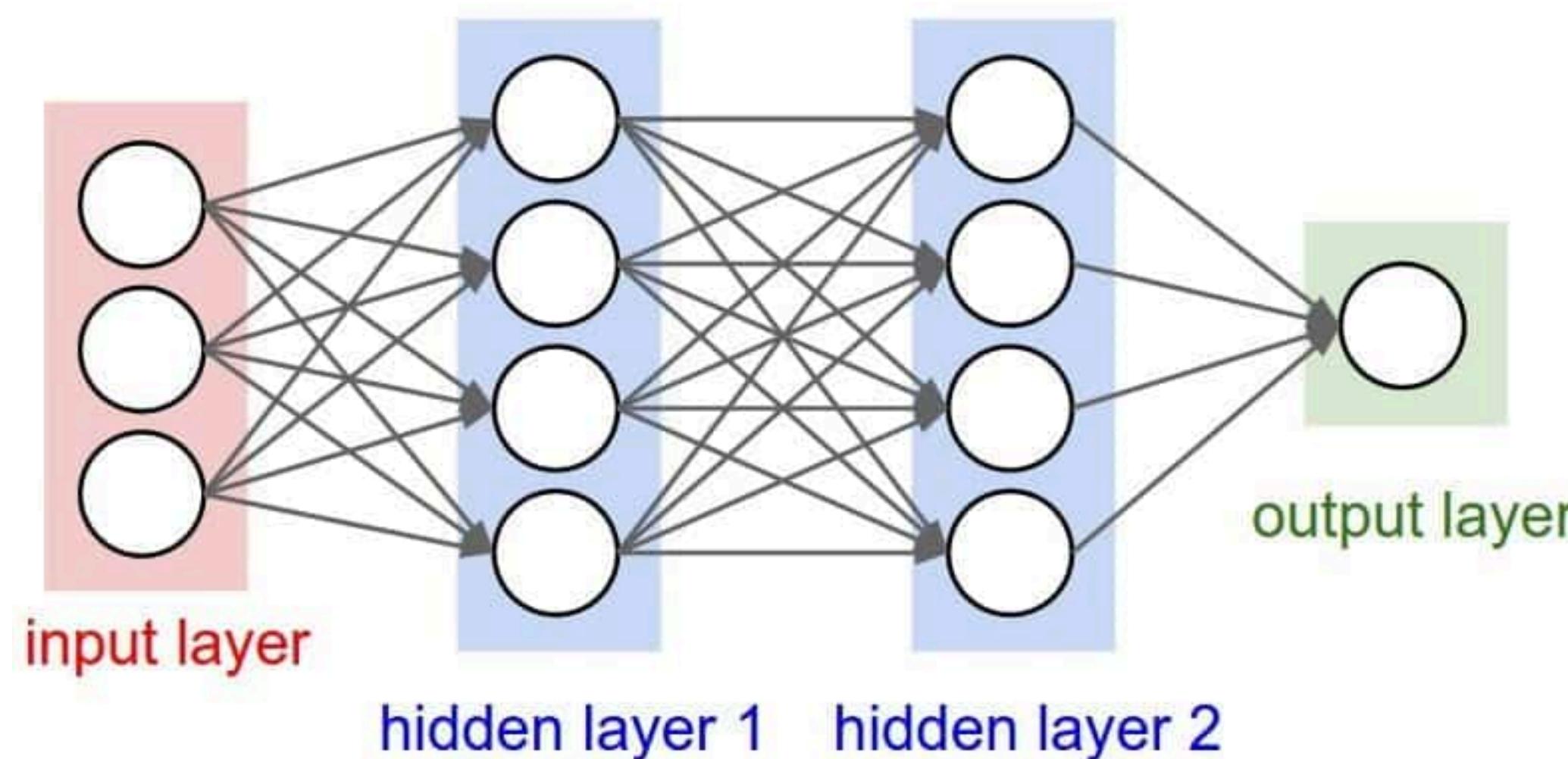
TP

Conclusion

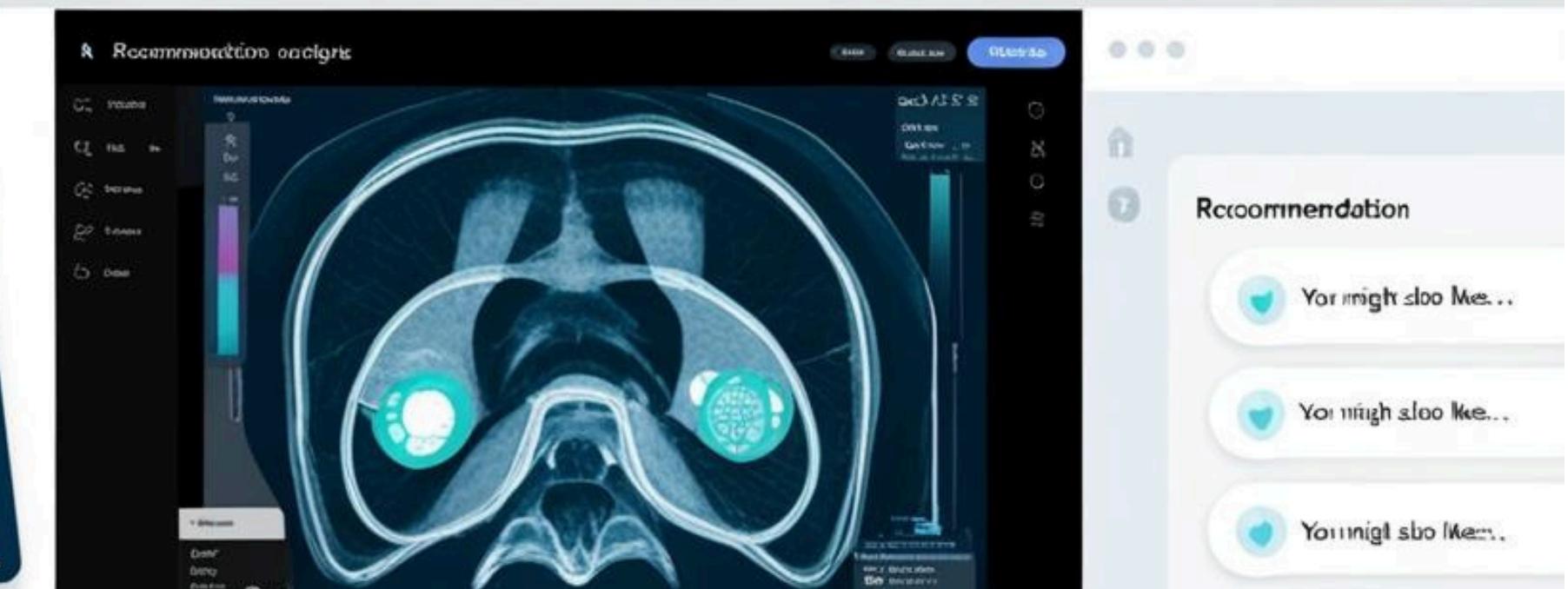
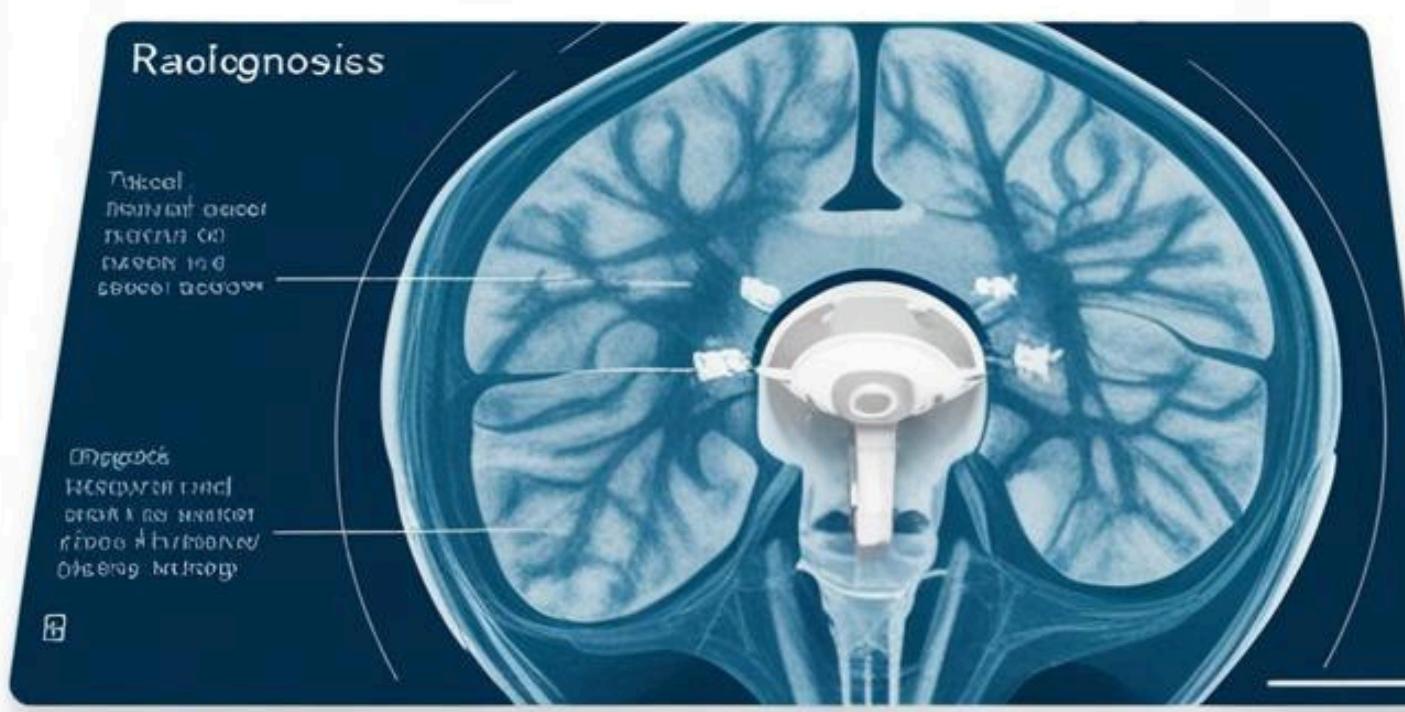
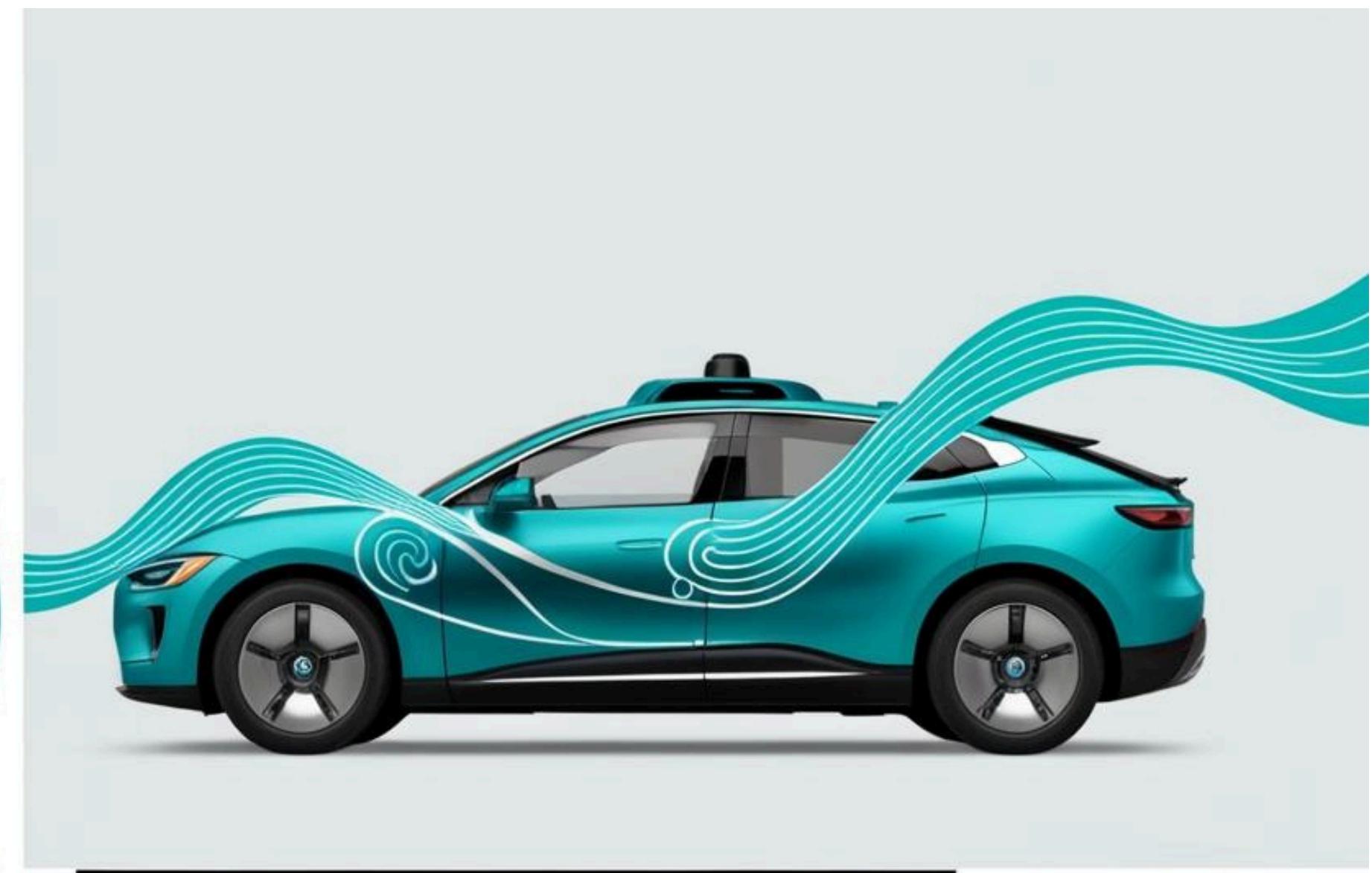


Définition

Un réseau neuronal artificiel est un modèle de calcul utilisé pour analyser des données et identifier des relations complexes en les traitant à travers plusieurs couches de nœuds interconnectés.



Domaines d'applications :

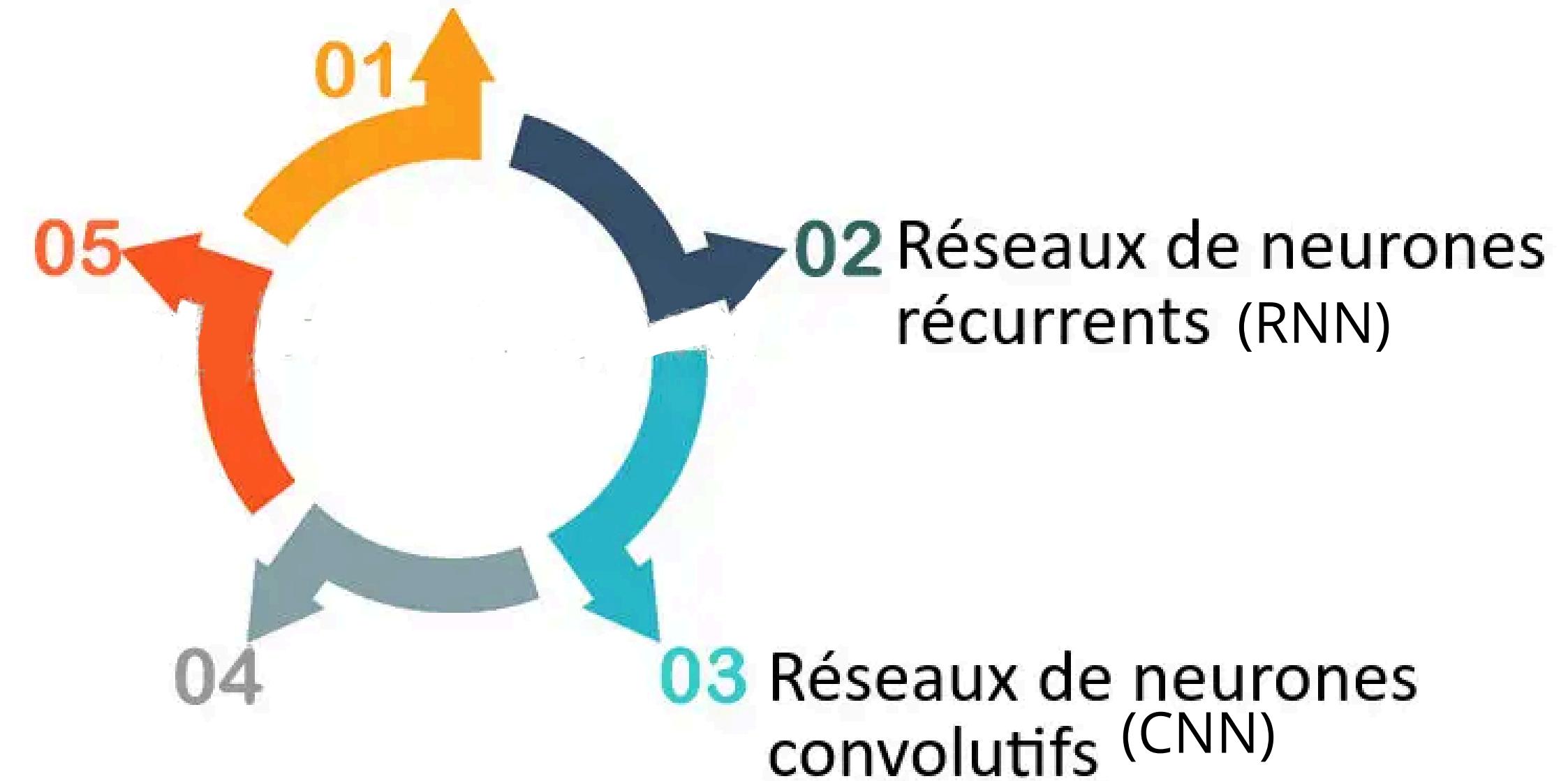


Types de réseaux de neurones :

Réseaux de neurones à réaction (Feedforward)

Réseaux de mémoire à long terme et à court terme (LSTM)

Réseaux contradictoires génératifs (GAN)



Avantages :

Performances élevées : Excellents résultats dans des tâches complexes .

Apprentissage automatique : Capables d'apprendre directement à partir des données.

Applications universelles ;

Défis :

Complexité et coût : Entraînement long et coûteux en ressources.

Explicabilité : Modèles souvent perçus comme des ‘boîtes noires’.

Besoin en données : Requiert de grandes quantités de données annotées.

Histoire

1948

L'étude des modèles neuronaux
biologiques

1949

Un modèle mathématique des réseaux
de neurones biologiques.

1957

Le premier réseau de neurones
artificiels fonctionnel : le perceptron

Histoire

1982

Un des premiers modèles de réseaux de neurones artificiels récurrents

1986

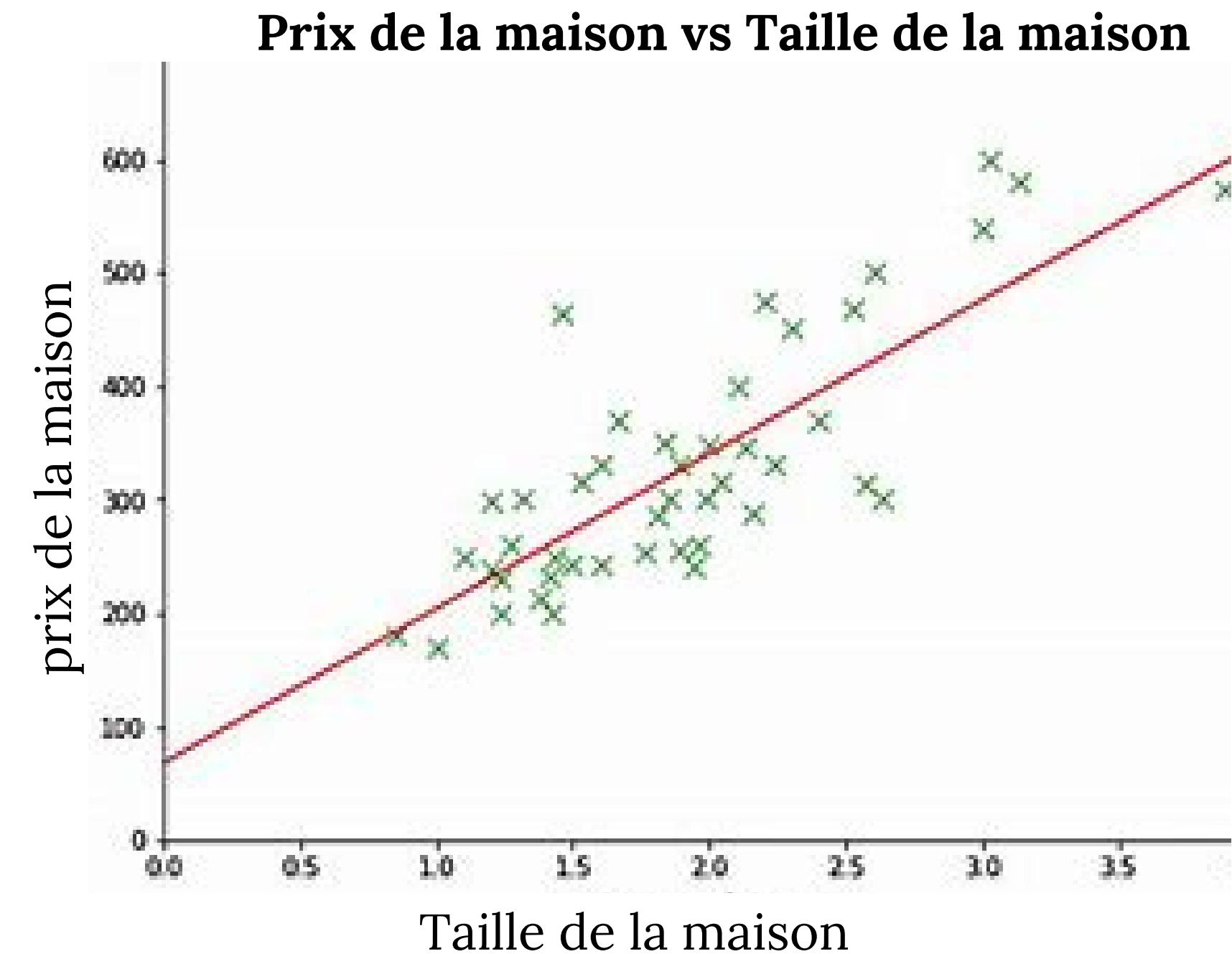
Le perceptron à couches multiples est introduit.

Après les années
2000

Les développements liés aux réseaux de neurones profonds

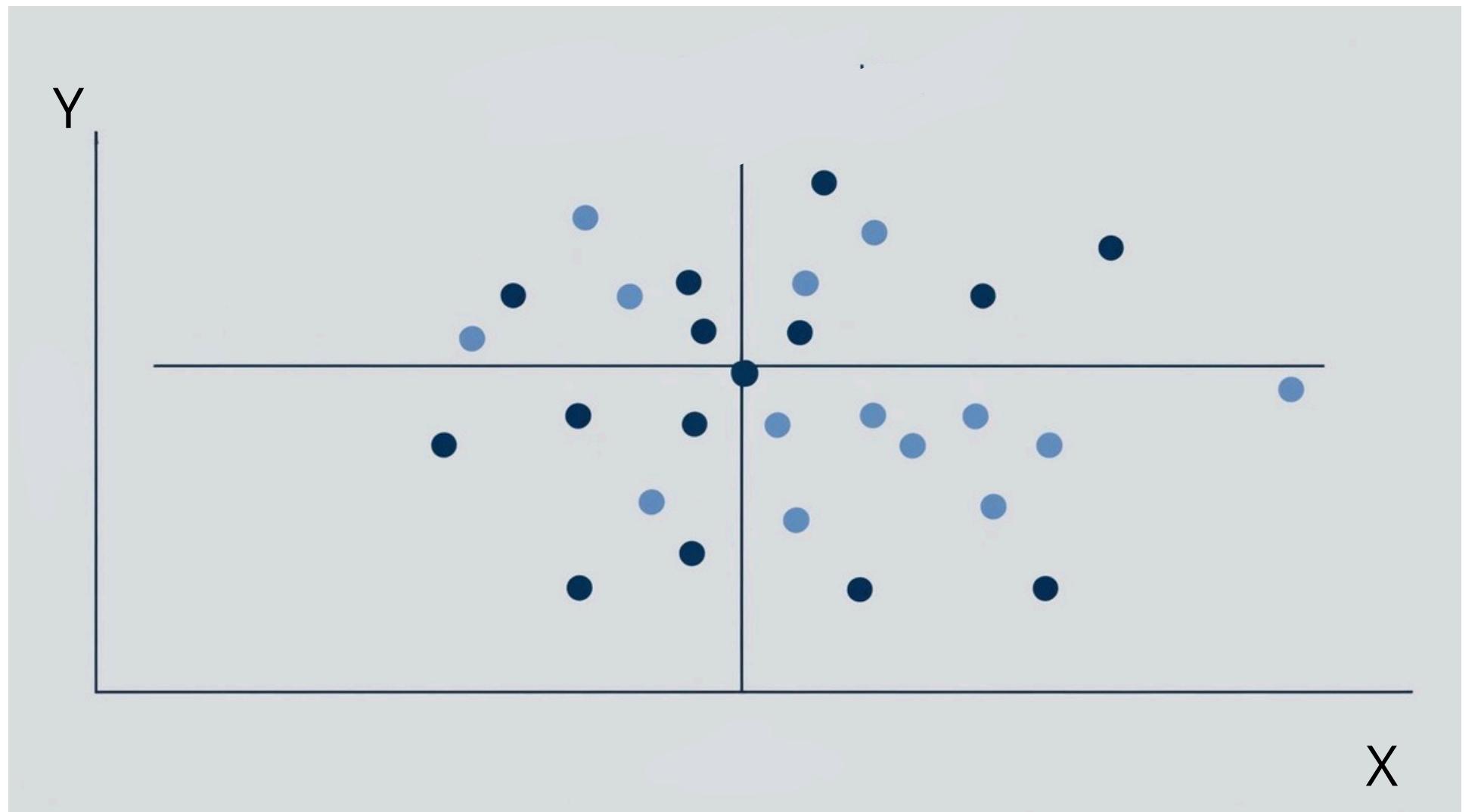
Régression Linéaire

Définition : une technique d'analyse de données qui prédit la valeur de données inconnues en utilisant une autre valeur de données apparentée et connue.



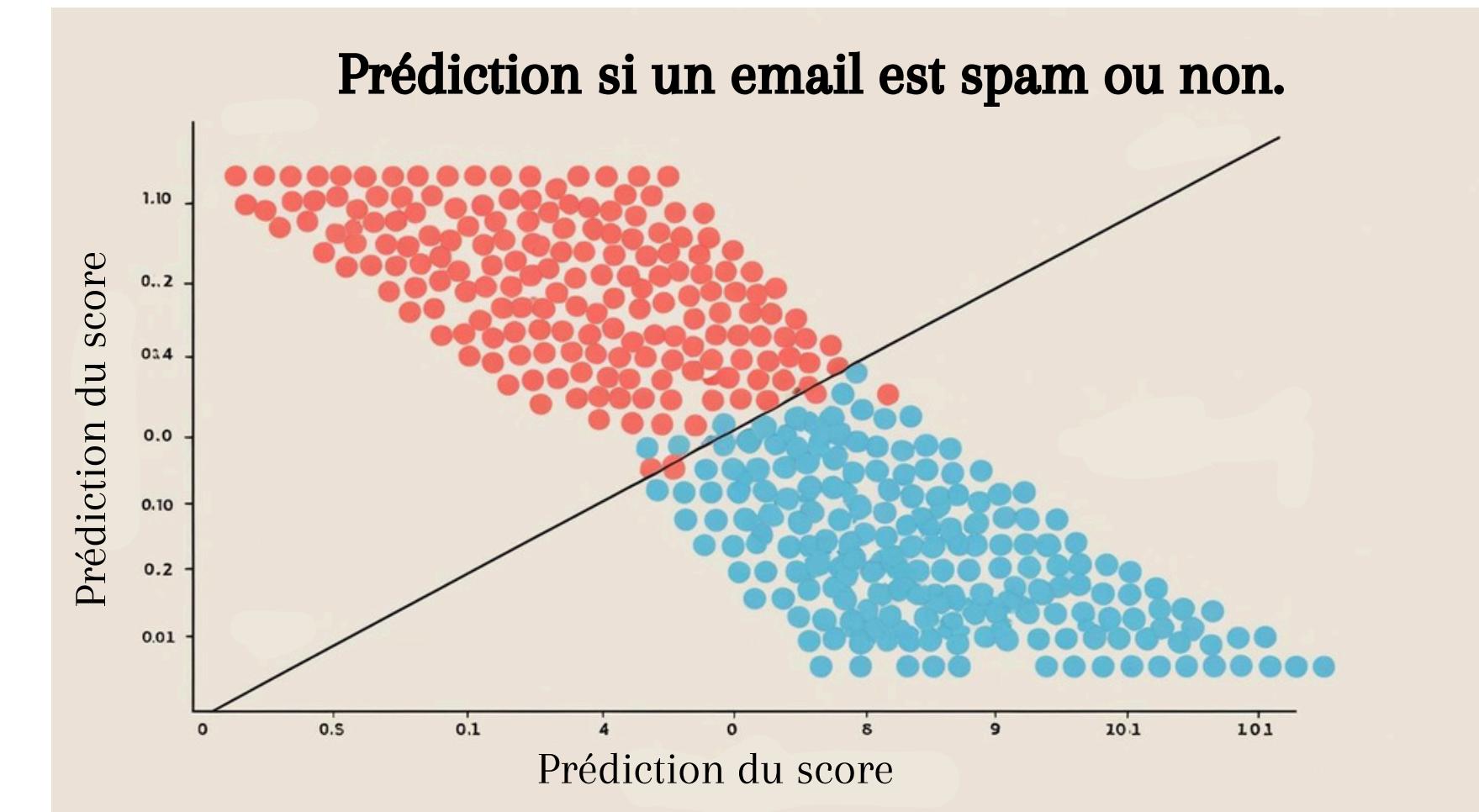
Limites principales

- Modélise uniquement des relations linéaires.
- Ne capture pas les relations complexes ou non linéaires.
- Sensible aux valeurs extrêmes (outliers).



Régression Logistique

Définition : Un modèle utilisé pour classifier les données en deux catégories.

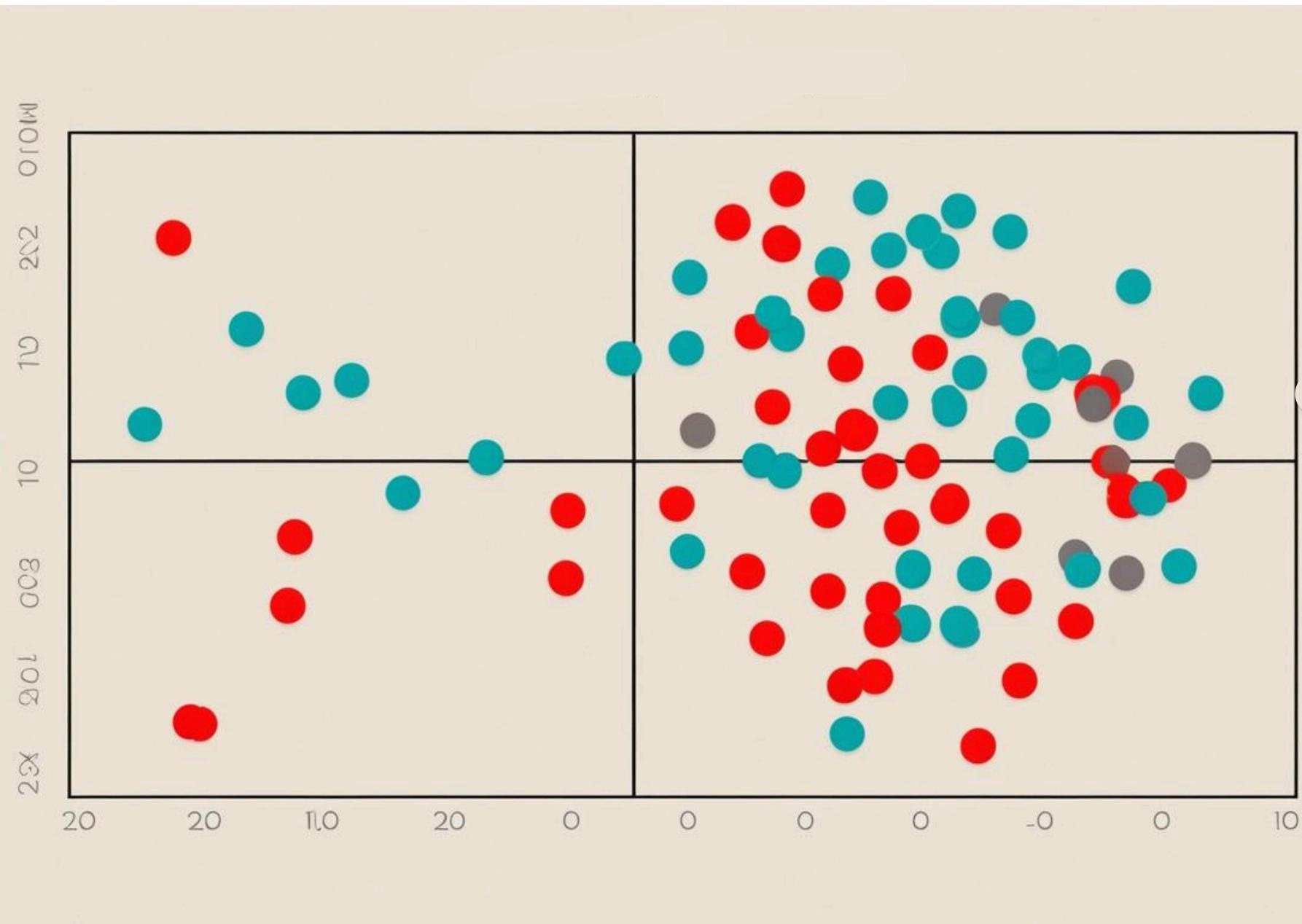


Limites principales

Gère uniquement des classifications binaires (ou via des extensions complexes).

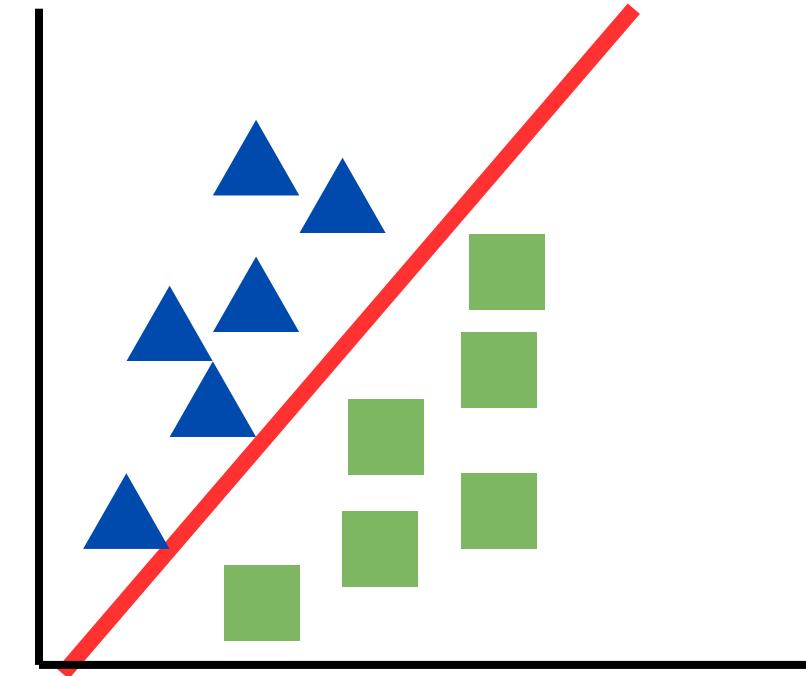
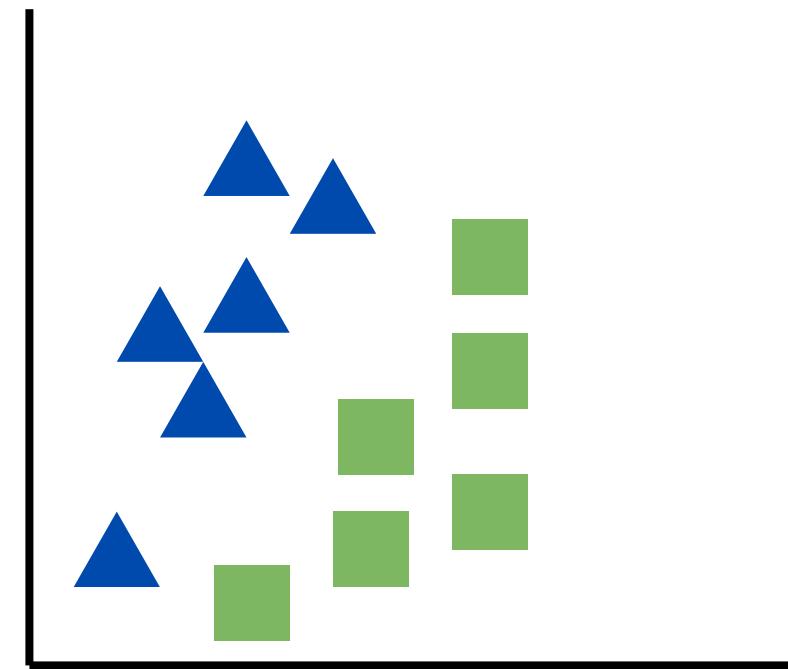
Inefficace pour des relations non linéaires complexes entre les données.

Ne fonctionne pas bien avec des volumes de données très élevés ou des données complexes (images, sons).



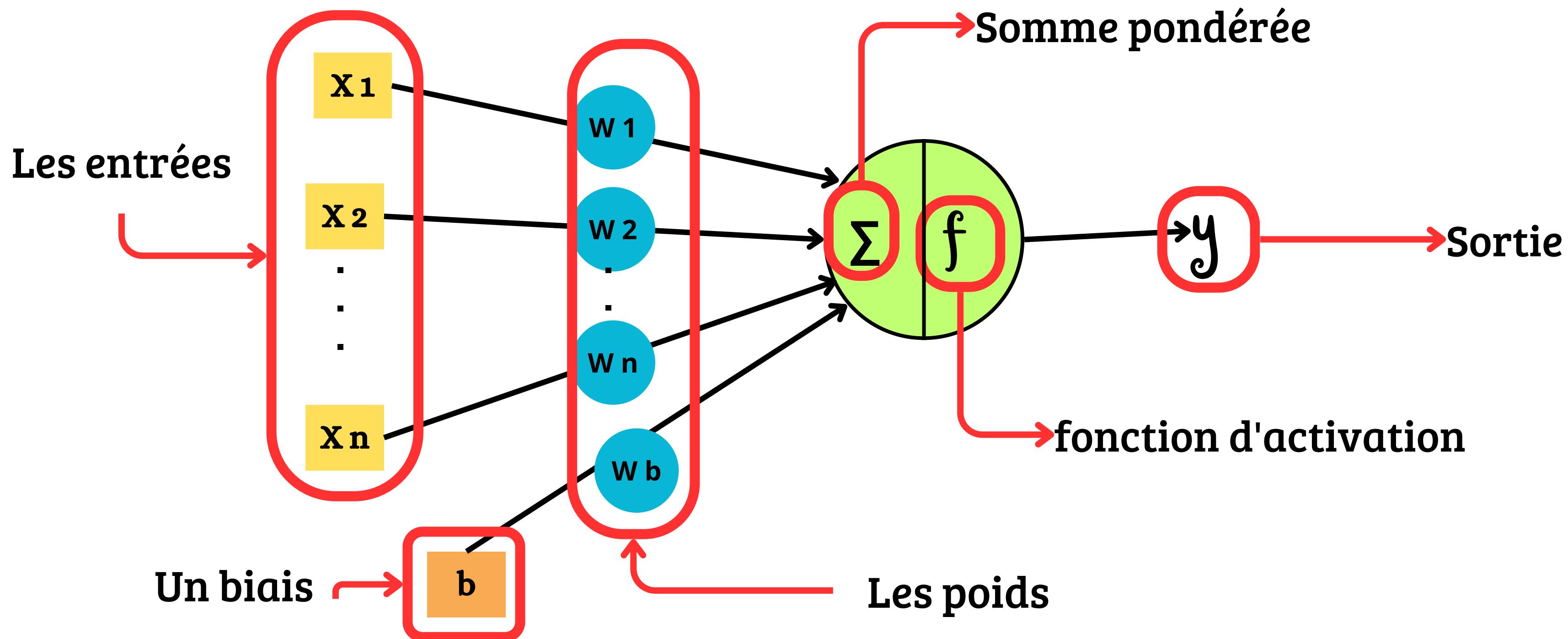
perceptron simple

Le perceptron est un modèle de base d'un neurone artificiel, capable de classer des données en les séparant par une ligne droite, en fonction d'un ensemble de poids et d'une fonction d'activation simple.

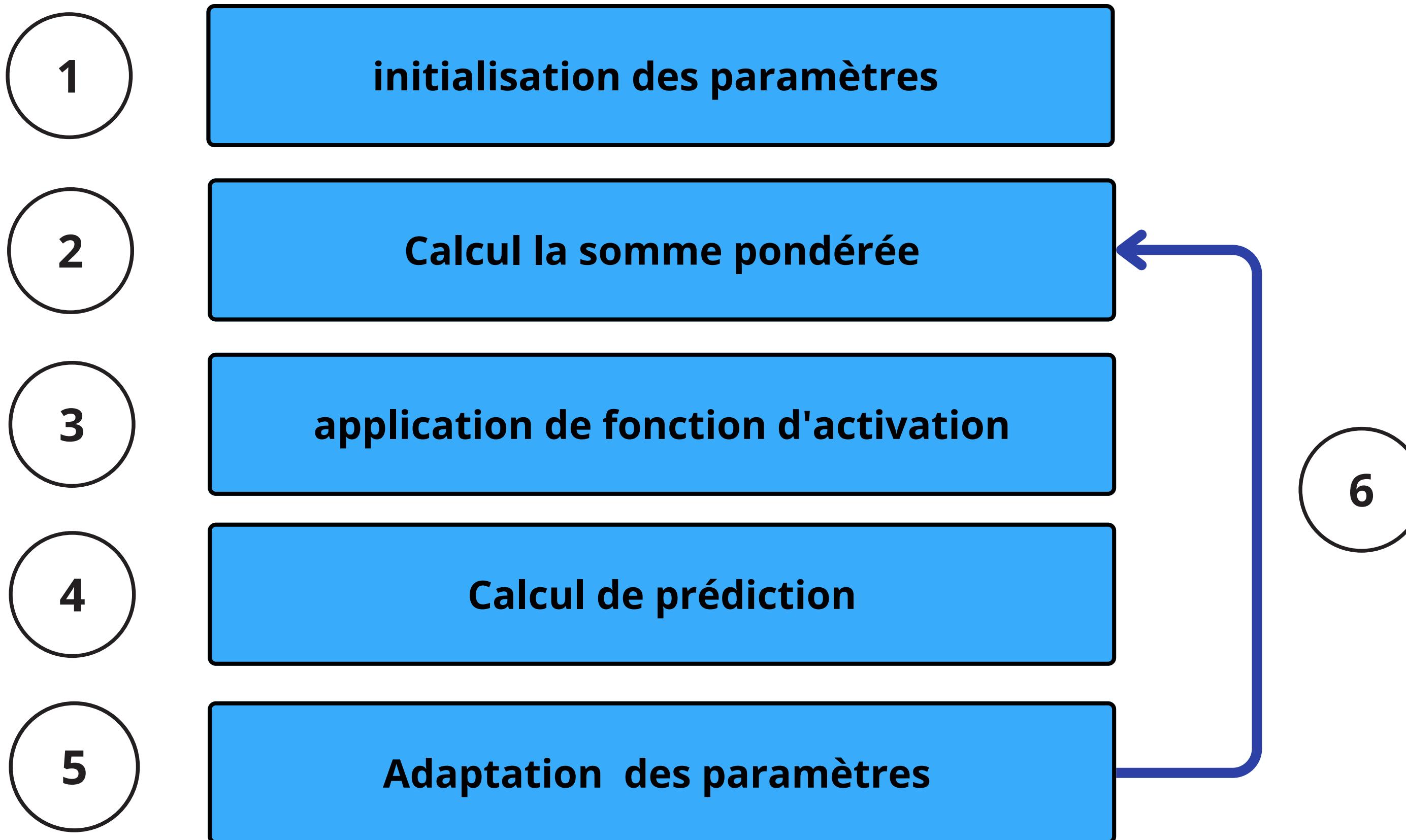


perceptron simple

- Structure de base

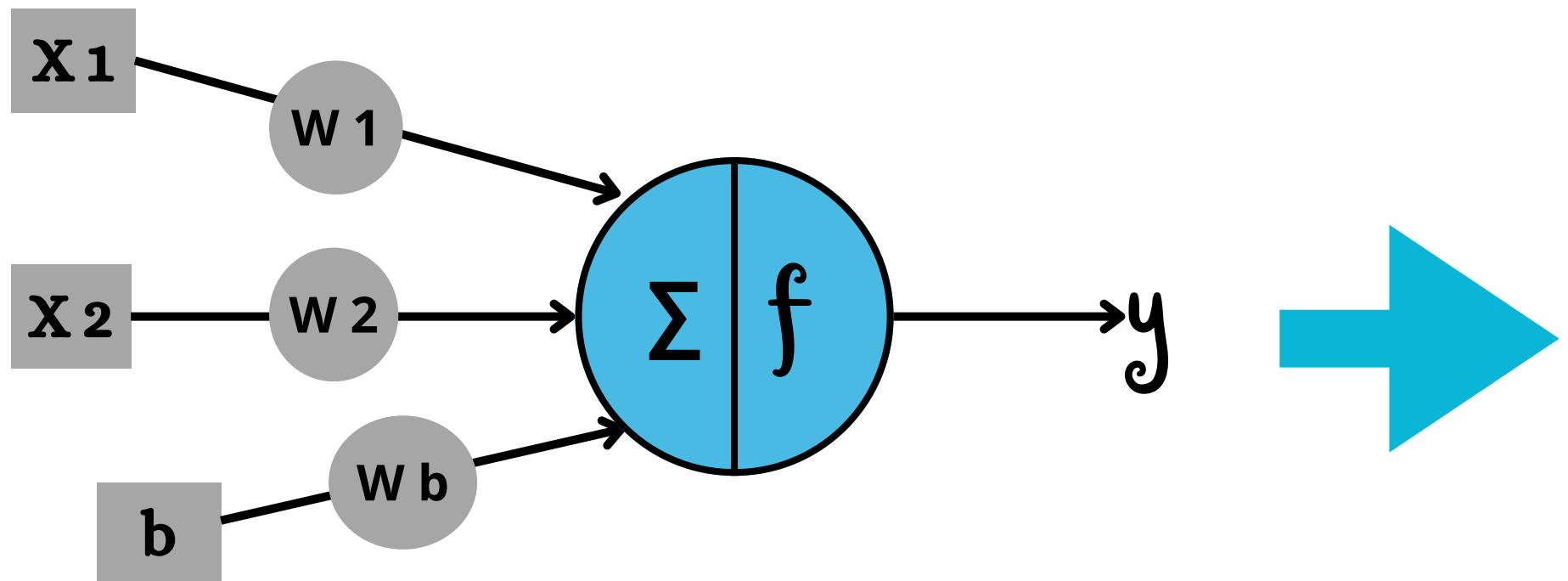


Fonctionnement de perceptron



Etape 2: la Somme Pondérée

$$z = \sum_{i=1}^n w_i x_i + b w_b$$



$$z = w_1 x_1 + w_2 x_2 + b w_b$$

Etape 3: fonction d'activation

Les fonctions d'activation sont des outils mathématiques qui transforment la somme pondérée en une sortie compréhensible pour le modèle. Elles décident si un neurone doit être activé, ajoutant des non-linéarités pour résoudre des problèmes complexes.

fonction seuil

fonction sigmoid

fonction tanh

fonction softmax

fonction ReLU

Etape 3: fonction d'activation

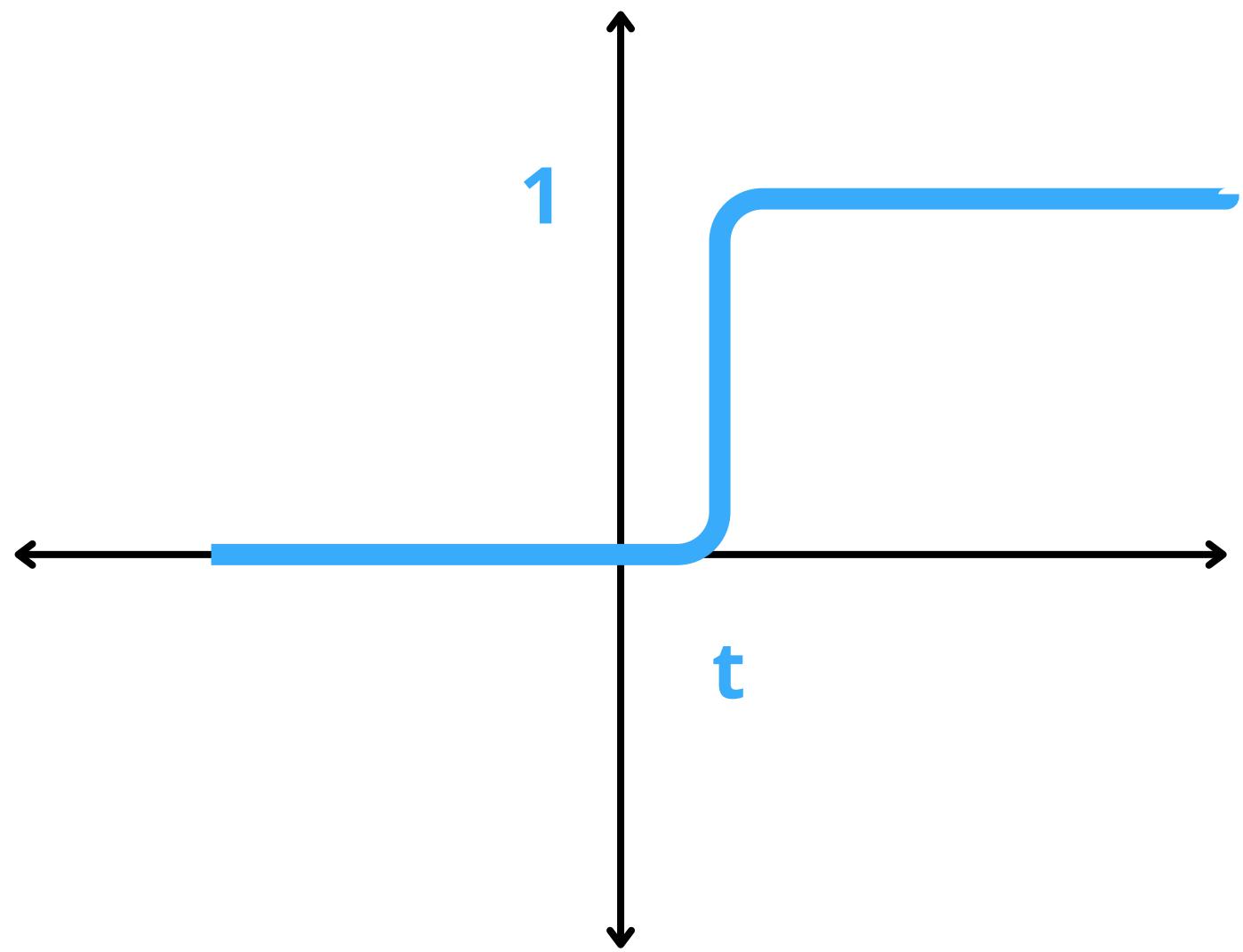
fonction seuil - step function

- $\text{step}(z)=1 \text{ si } z \geq t$
- $\text{step}(z)=0 \text{ si } z < t$

exemple: $t=5$

$Z=-1 \text{ step}(Z) = 0$

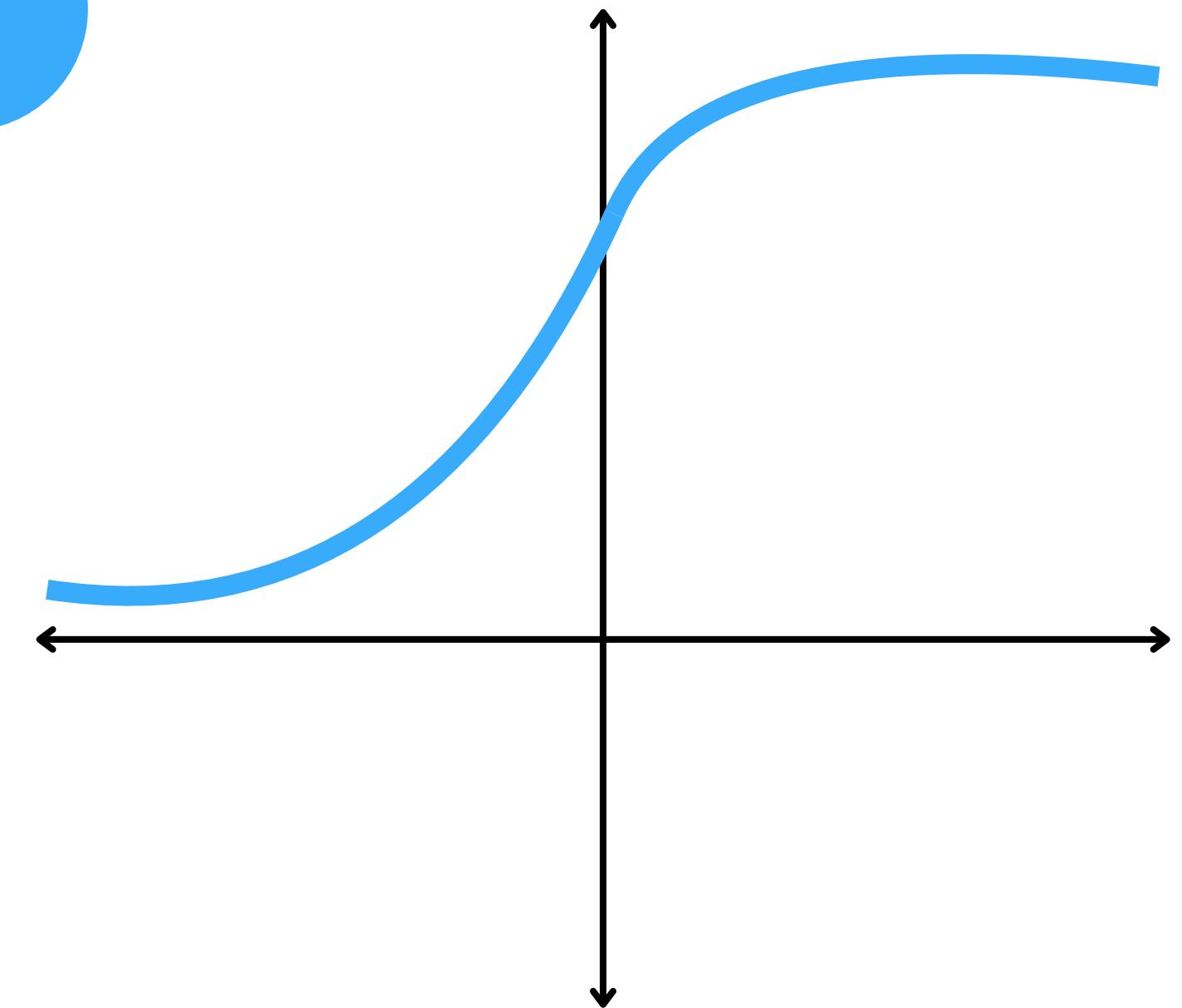
$Z=5 \text{ step}(Z) = 1$



Etape 3: fonction d'activation

fonction sigmoid

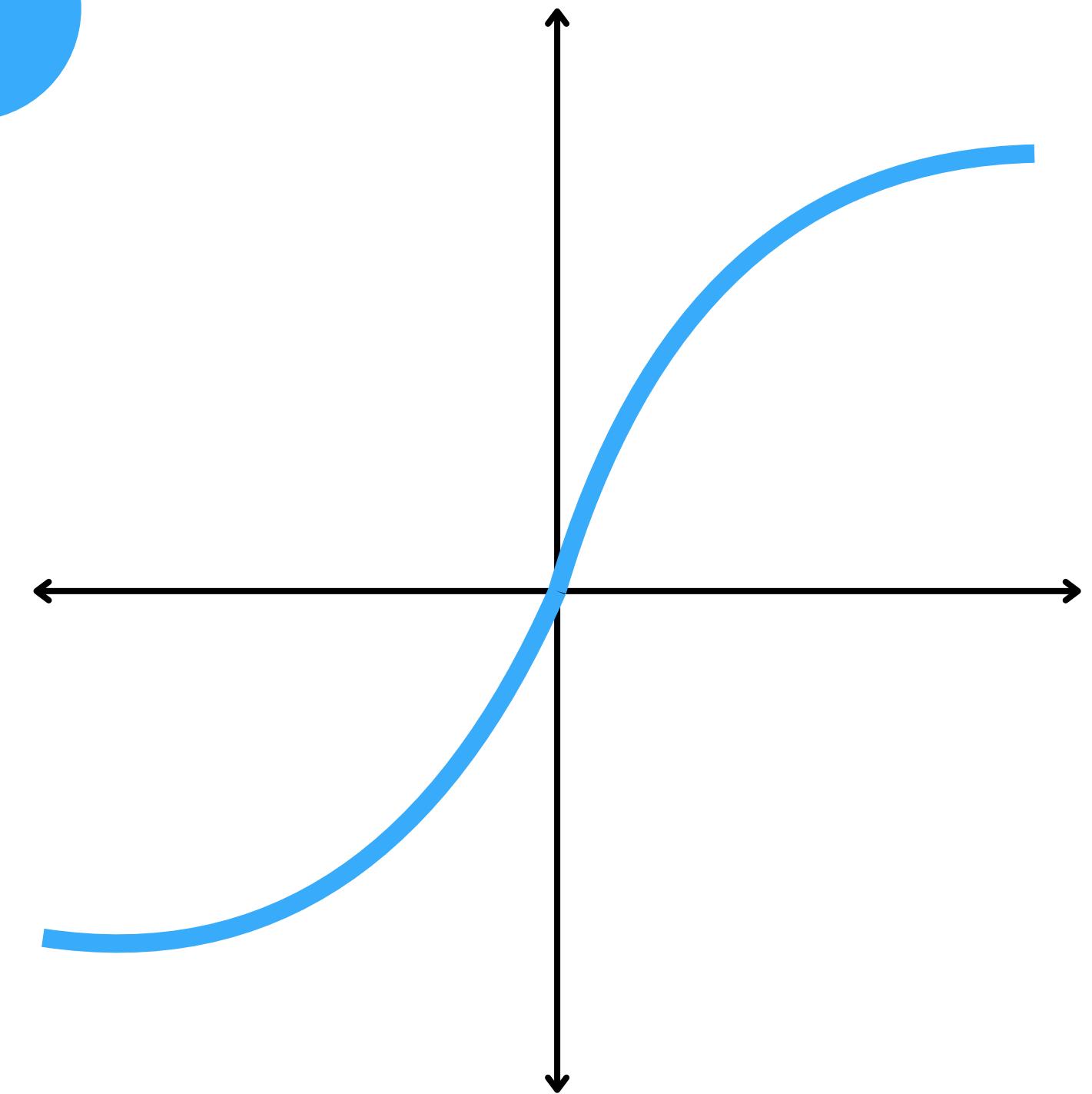
$$\text{sigm}(z) = \frac{1}{1 + e^{-z}}$$



Etape 3: fonction d'activation

fonction tanh

$$\tanh(z) = \frac{2}{1 + e^{-2z}} - 1$$



Etape 3: fonction d'activation

fonction softmax

$$\text{softmax}(z) = \frac{e^{zi}}{\sum_{j=1}^n e^{zj}}$$

$$Z1=2 \quad Z2=1 \quad Z3=0.1$$

- $e^{Z1} = e^2 \approx 7.389$
- $e^{Z2} = e^1 \approx 2.718$
- $e^{Z3} = e^{0.1} \approx 1.105$
- $\sum_{j=1}^3 e^{zj} = 7.389 + 2.718 + 1.105 = 11.212$

- $\text{softmax}(z1) = \frac{e^{z1}}{\sum_{j=1}^3 e^{zj}} = \frac{7.389}{11.212} \approx 0.659$
- $\text{softmax}(z2) \approx 0.243$
- $\text{softmax}(z3) \approx 0.098$

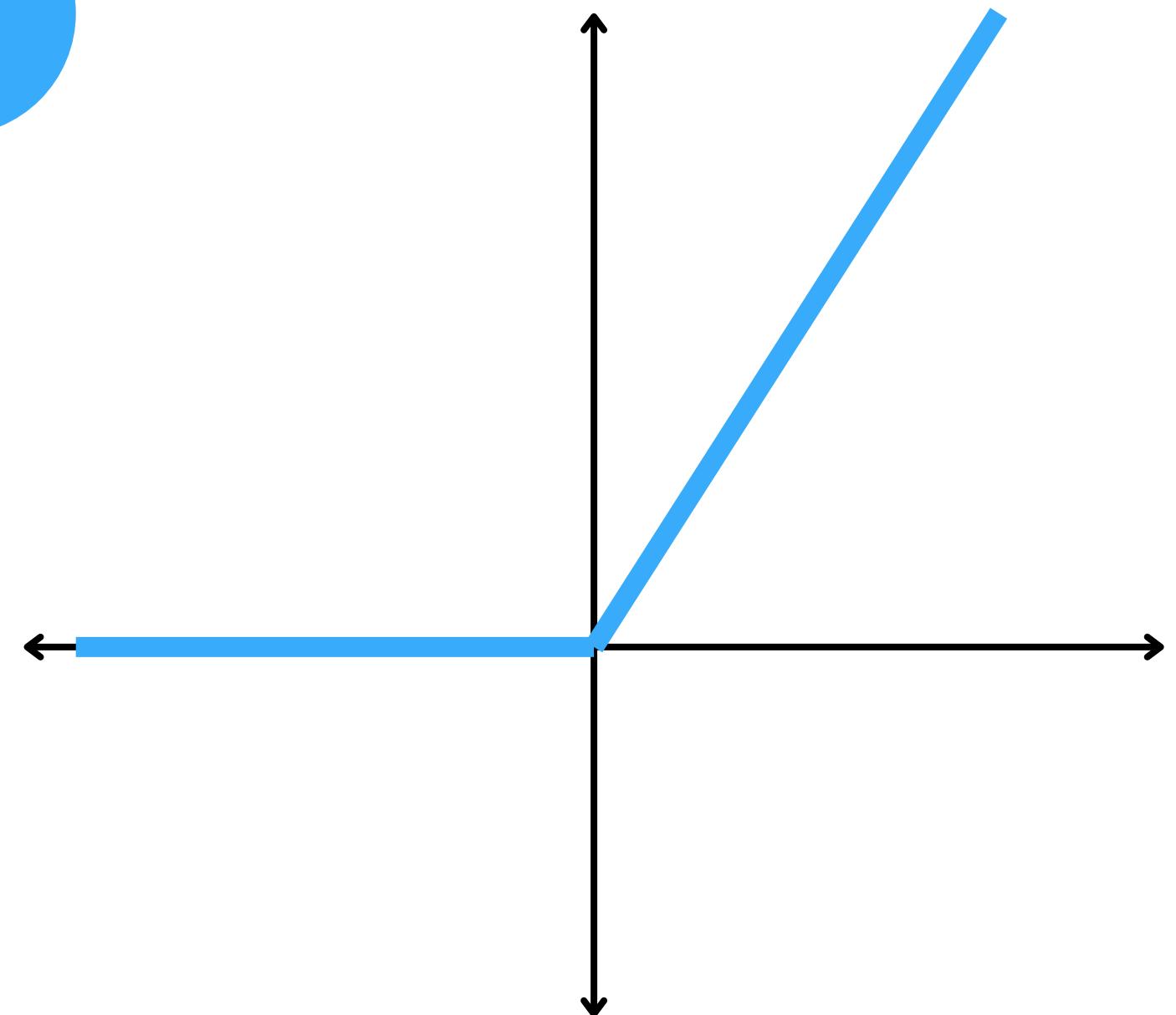
Etape 3: fonction d'activation

fonction ReLU

$$\text{Relu}(Z) = \max(0, Z) \begin{cases} \cdot 0 \text{ si } Z \leq 0 \\ \cdot z \text{ si } Z > 0 \end{cases}$$

exemple: $Z=-1$ $\text{ReLU}(Z) = 0$

$Z=10$ $\text{ReLU}(Z) = 10$



Etape 4: Prédiction

- **calculer l'erreur de prédiction**

$$E = y_r - y_p$$

{
 y_r: la valeur réelle (ou attendue) de la sortie.
y_p: la valeur prédictée par le perceptron.

E = 0 Sortie correcte.

E ≠ 0 Erreur, ajustement nécessaire.

Etape 5: Apprentissage

- **Étape d'optimisation**

ajuster les poids

$$W_i = W_i + (\eta * X_i * E)$$

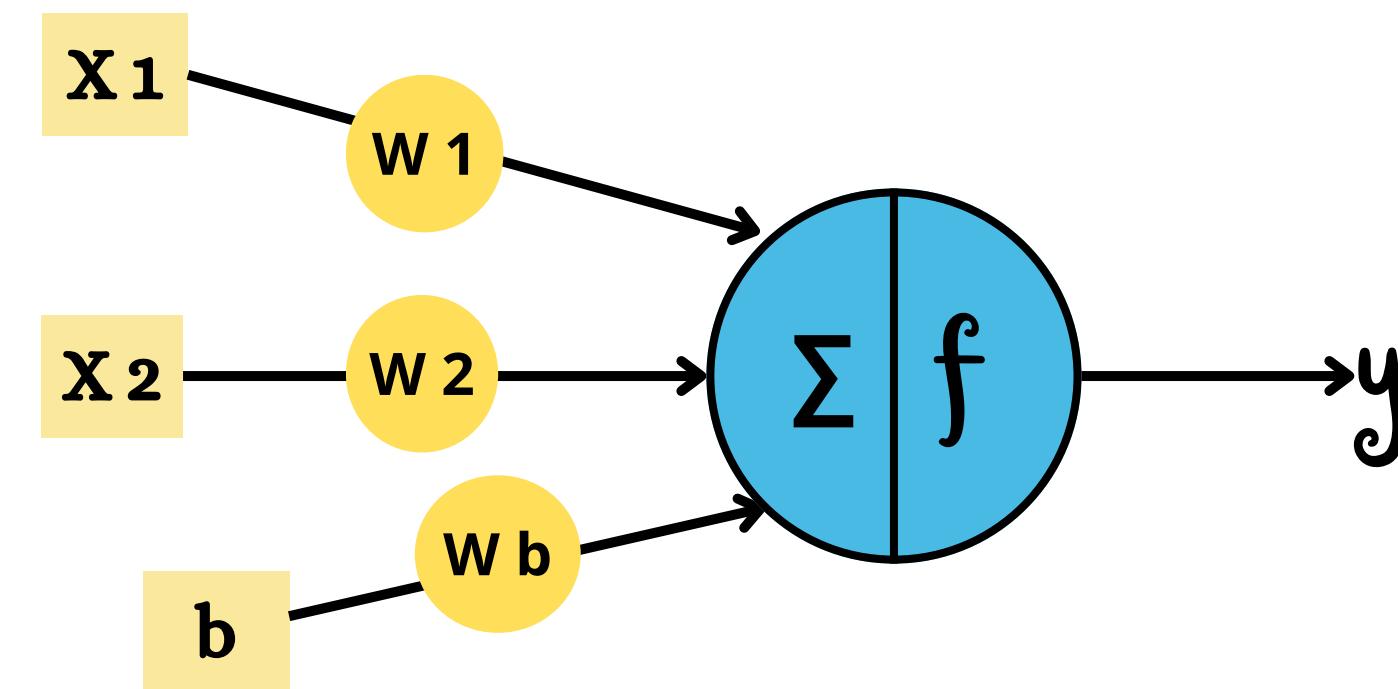
$$W_b = W_b + (\eta * b * E)$$

{

- η** : taux d'apprentissage (learning rate).
- X_i** : entrée associée au poids w_i
- E** : l'erreur

Exemple d'Algorithme d'apprentissage de perceptron simple

x₁	x₂	y
0	0	0
0	1	1
1	0	1
1	1	1



Exemple d'Algorithme d'apprentissage de perceptron simple

X1	X2	b	W1	W2	Wb	Z	t = f(z)	y
0	0	1	0.1	0.2	-0.2	-0.2	0	0
0	1	1						1
1	0	1						1

$$\bullet Z = X_1 * W_1 + X_2 * W_2 + b * W_b = 0 * 0,1 + 0 * 0,2 + 1 * -0,2 = -0,2$$

$$\bullet Z < t \quad \text{Step}(-0,2) = 0$$

t = 0,1

Exemple d'Algorithme d'apprentissage de perceptron simple

X1	X2	b	W1	W2	Wb	Z	t= f(z)	y
0	0	1	0.1	0.2	-0.2	-0.2	0	0
0	1	1	0.1	0.2	-0.2	0	0	1
1	0	1						1
								1

- $W_1 := W_1 + \eta * X_1 * E \rightarrow W_1 := 0.1 + 0.1 * 0 * 1 \rightarrow 0.1$
- $W_2 := W_2 + \eta * X_2 * E \rightarrow W_2 := 0.2 + 0.1 * 1 * 1 \rightarrow 0.3$
- $W_b := W_b + \eta * b * E \rightarrow W_b := -0.2 + 0.1 * 1 * 1 \rightarrow -0.1$

Exemple d'Algorithme d'apprentissage de perceptron simple

X1	X2	b	W1	W2	Wb	Z	t= f(z)	y
0	0	1	0.1	0.2	-0.2	-0.2	0	0
0	1	1	0.1	0.2	0.2	-0.2	0	1
1	0	1	0.1	0.3	-0.1	0	0	1
1	1	1						

- $W1 := W1 + \eta * X1 * E \rightarrow W1 := 0.1 + 0.1 * 1 * 1 \rightarrow 0.2$
- $W2 := W2 + \eta * X2 * E \rightarrow W2 := 0.3 + 0.1 * 0 * 1 \rightarrow 0.3$
- $Wb := Wb + \eta * b * E \rightarrow Wb := -0.1 + 0.1 * 1 * 1 \rightarrow 0$

$\rightarrow E = 1$

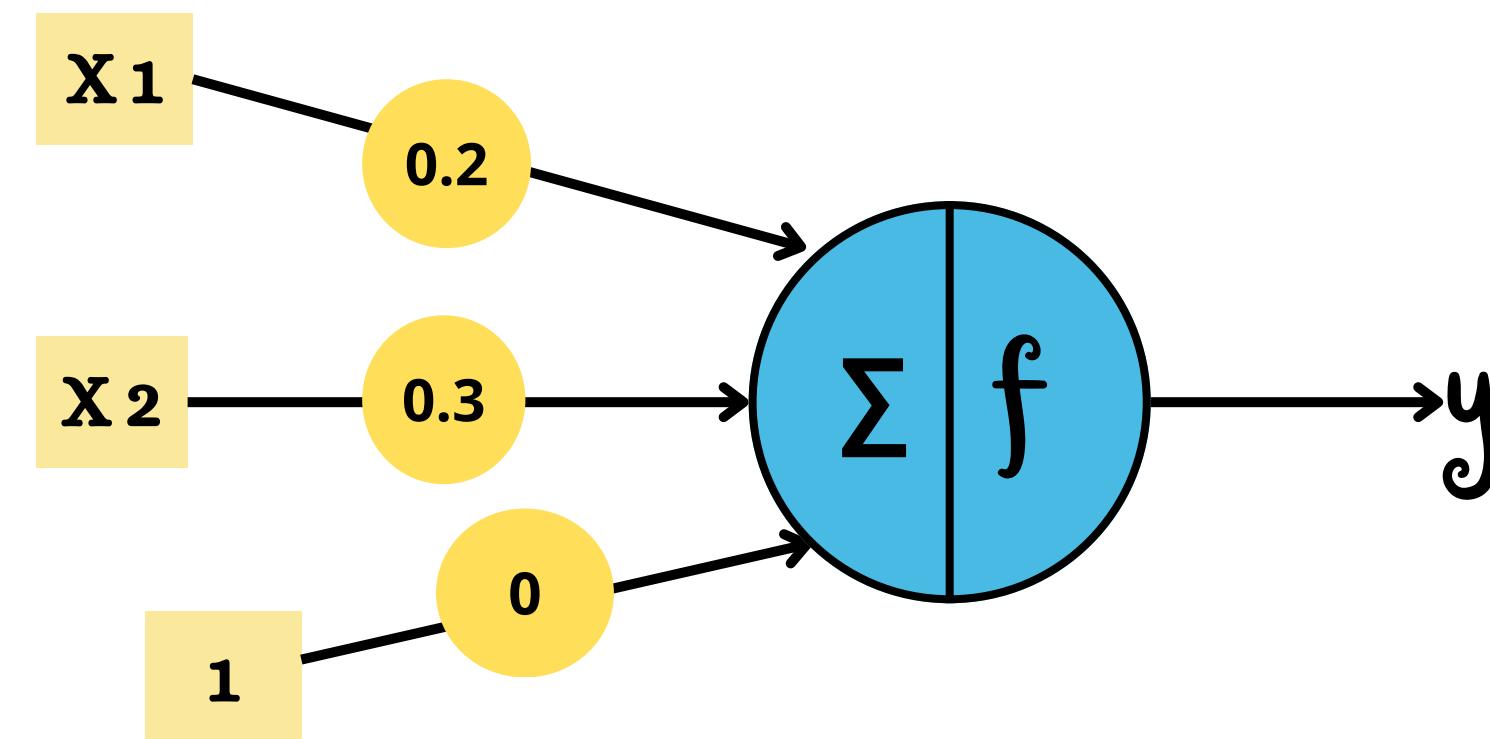
Exemple d'Algorithme d'apprentissage de perceptron simple

x_1	x_2	b	w_1	w_2	w_b	z	$t = f(z)$	y	
0	0	1	0.1	0.2	-0.2	-0.2	0	0	$\rightarrow E = 0$
0	1	1	0.1	0.2	0.2	-0.2	0	1	$\rightarrow E = 1$
1	0	1	0.1	0.3	-0.1	0	0	1	$\rightarrow E = 1$
1	1	1	0.2	0.3	0	0.5	1	1	$\rightarrow E = 0$

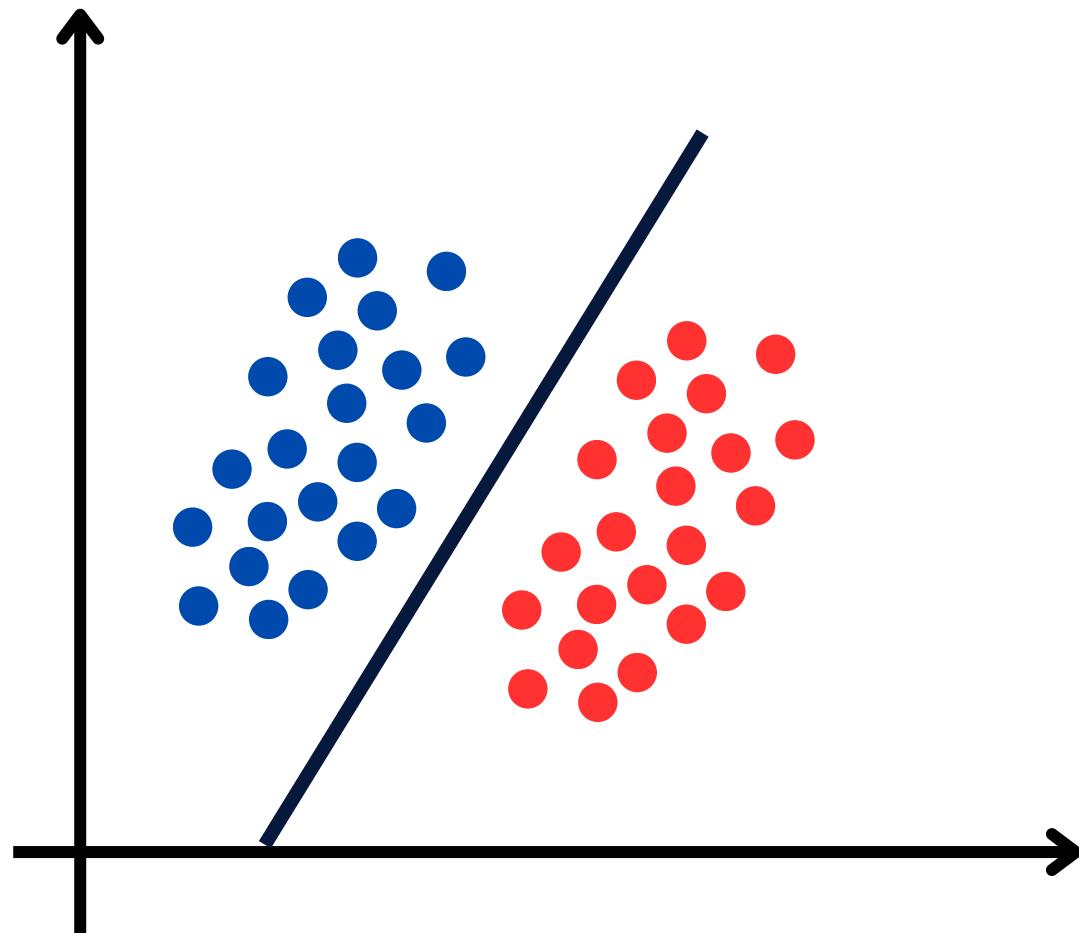
Exemple d'Algorithme d'apprentissage de perceptron simple

X1	X2	b	W1	W2	Wb	Z	t= f(z)	y
0	0	1	0.2	0.3	0	0	0	0
0	1	1	0.2	0.3	0	0.3	1	1
1	0	1	0.2	0.3	0	0.2	1	1
1	1	1	0.2	0.3	0	0.5	1	1

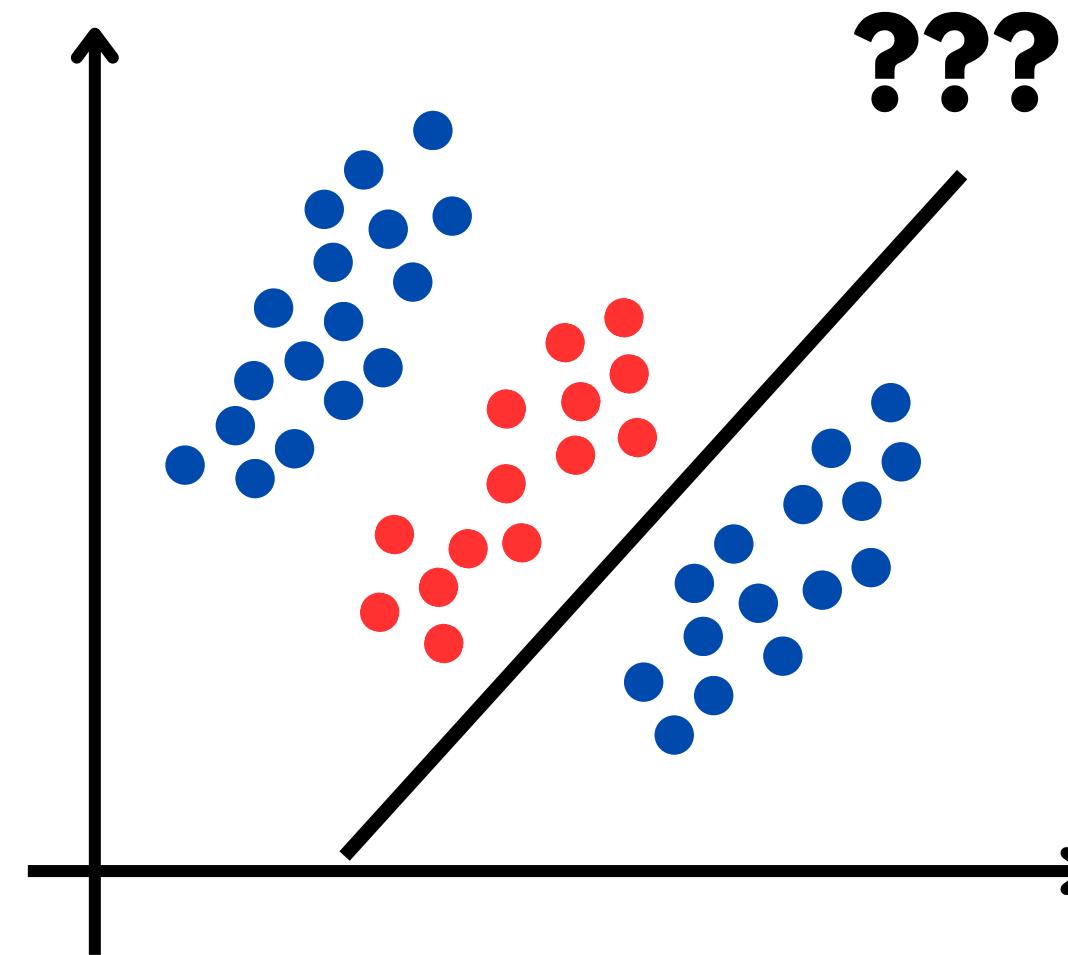
Exemple d'Algorithme d'apprentissage de perceptron simple



Les limites du perceptron simple

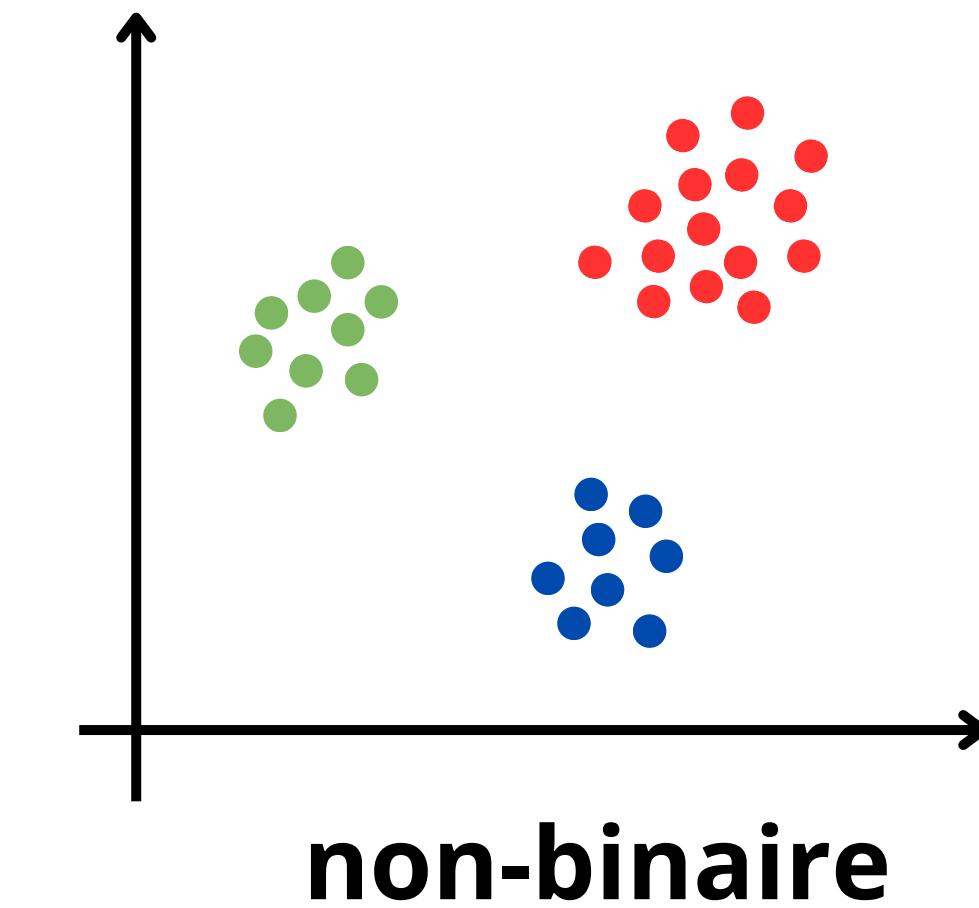
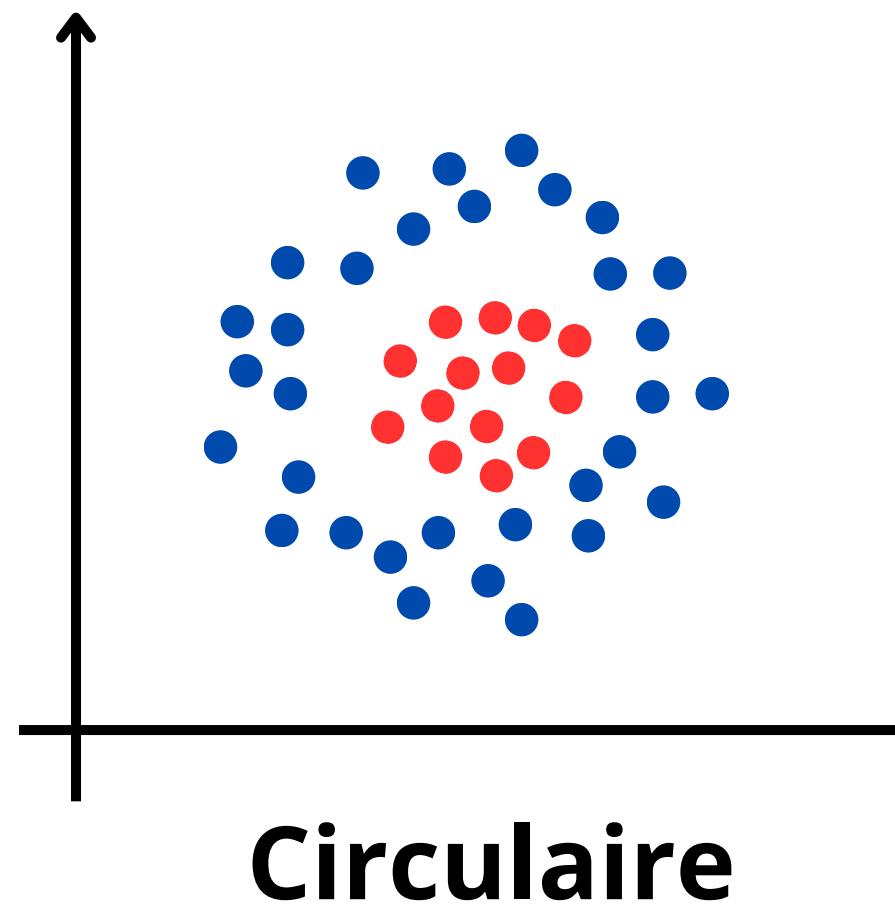
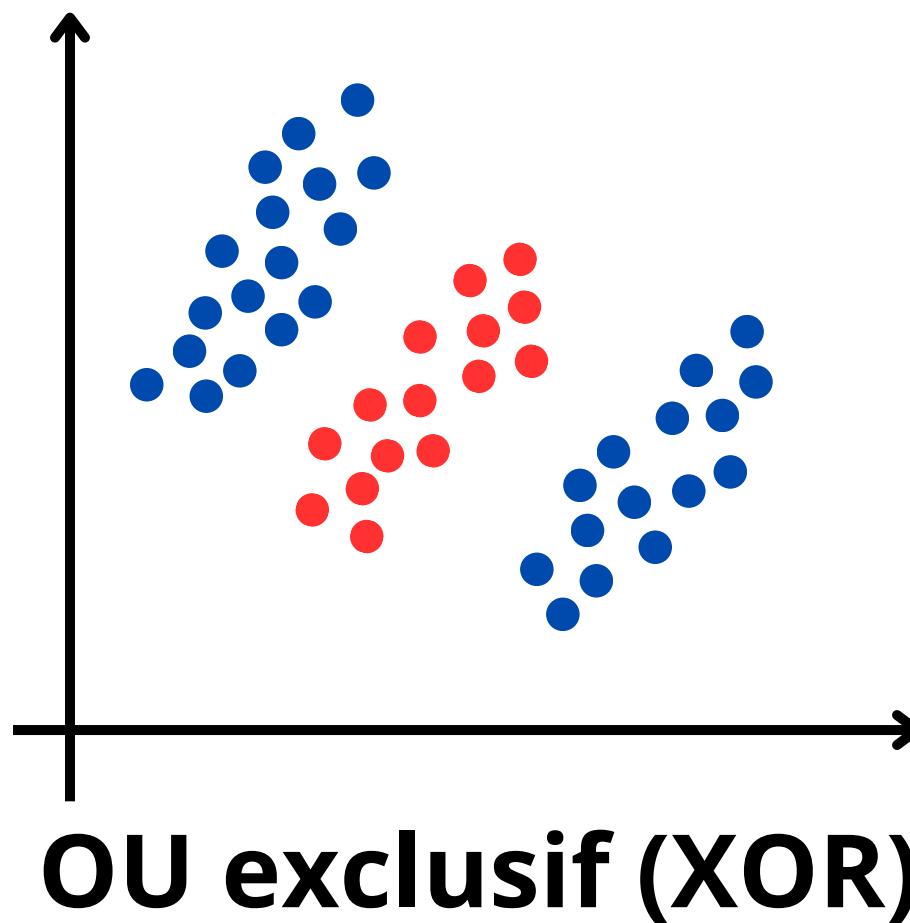


Problème Résolu



Problème non-Résolu

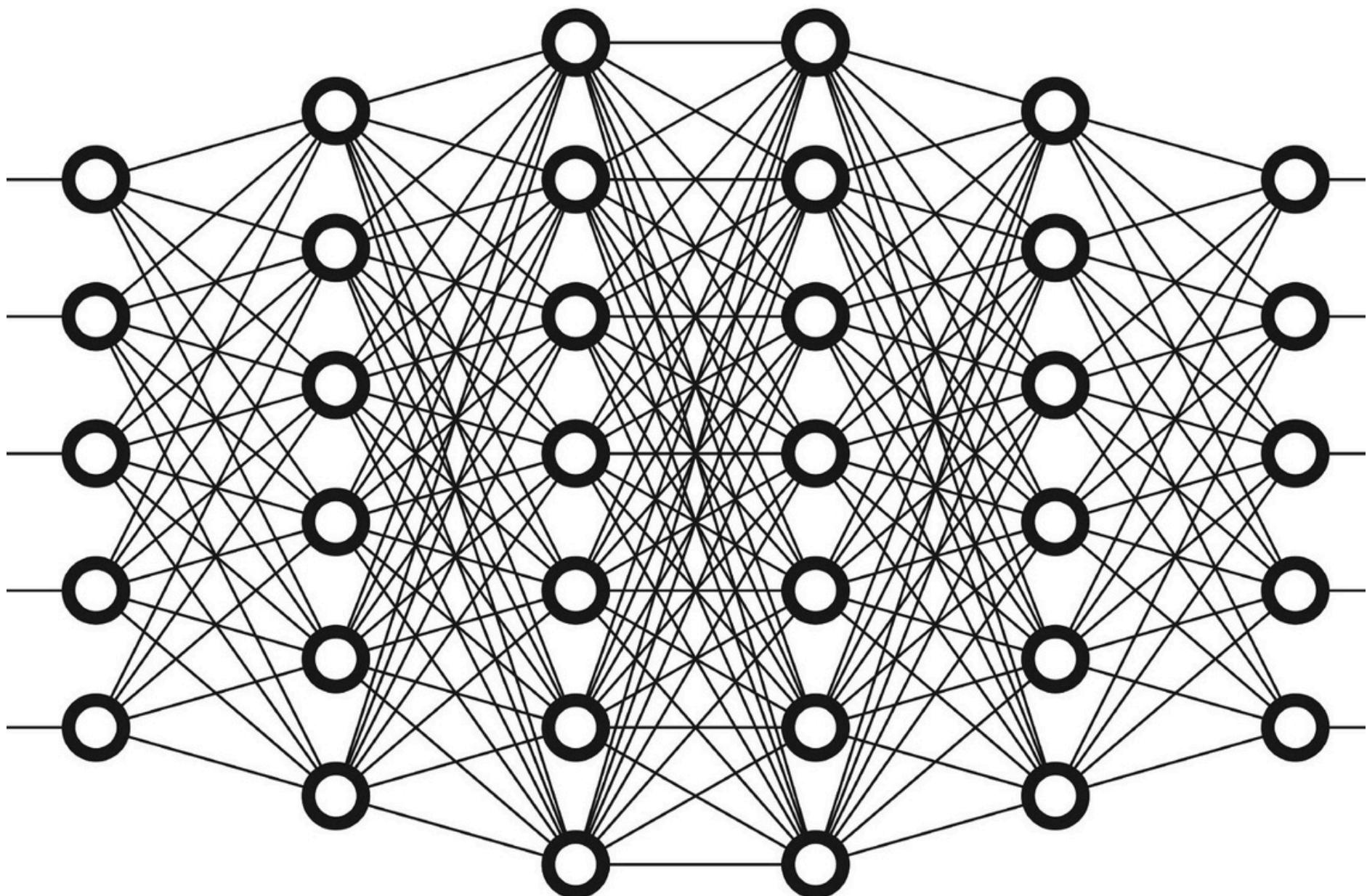
Les limites du perceptron simple



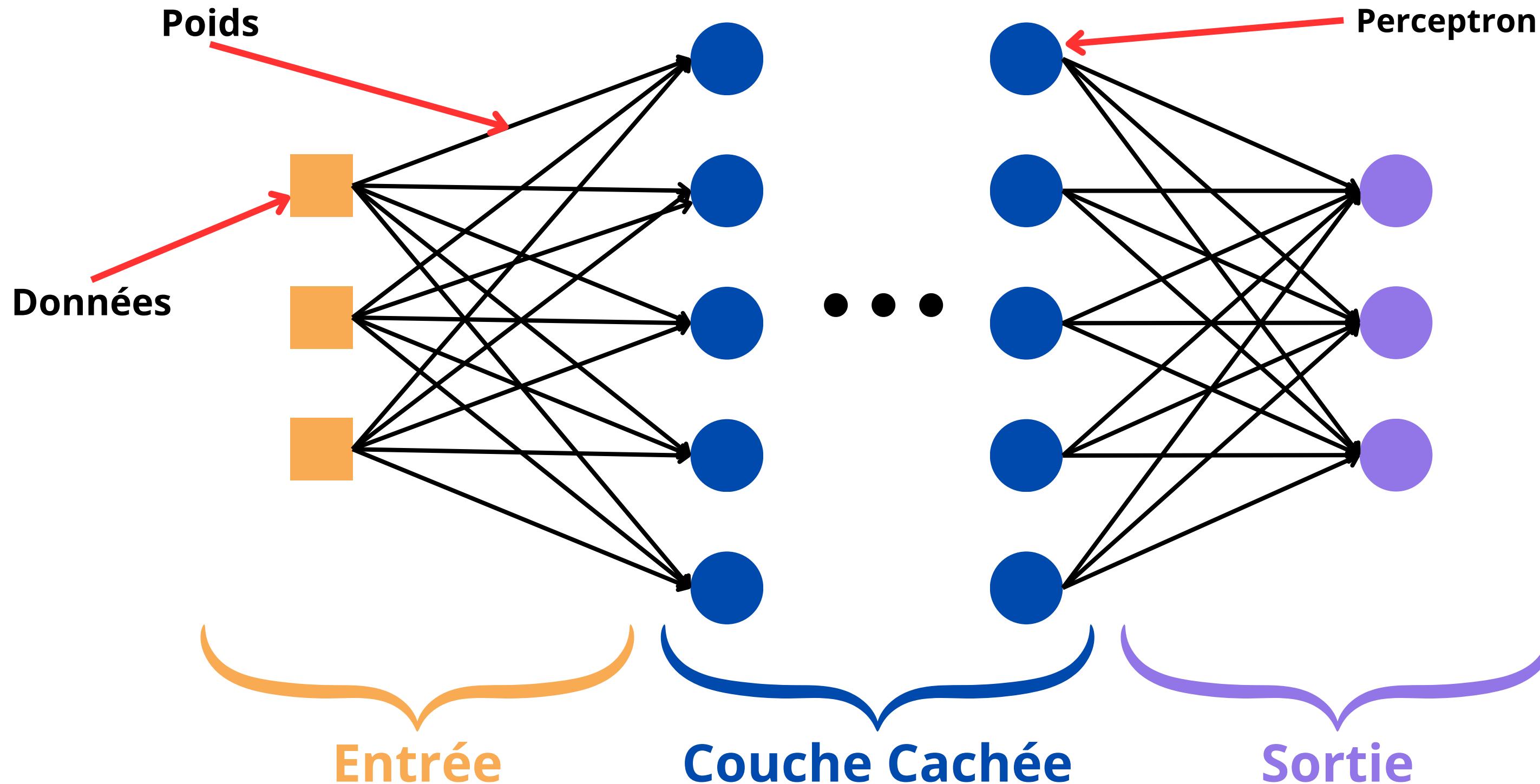
La solution: perceptron multicouche

Perceptron multicouche :

Un perceptron multicouche est un réseau de neurones composé de plusieurs couches de perceptrons, permettant de résoudre des problèmes non linéaires en apprenant des représentations complexes des données à travers ces couches.

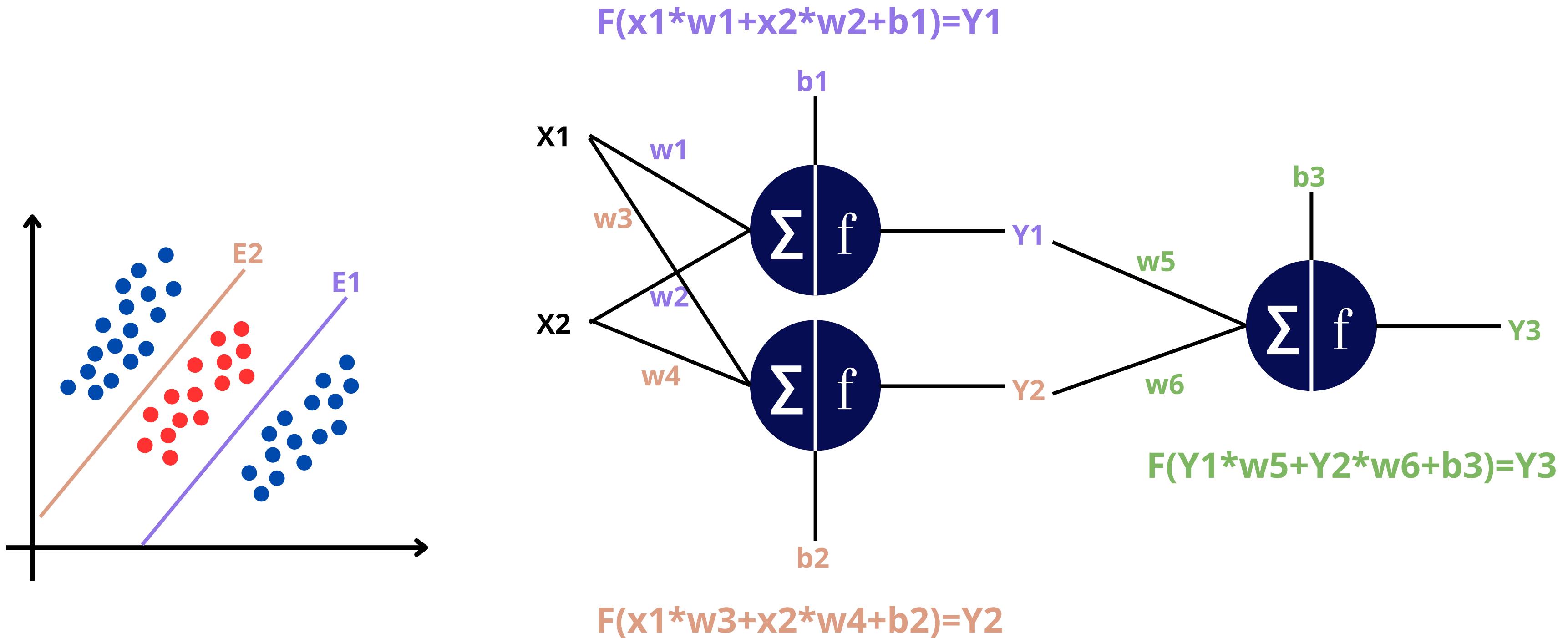


La structure générale

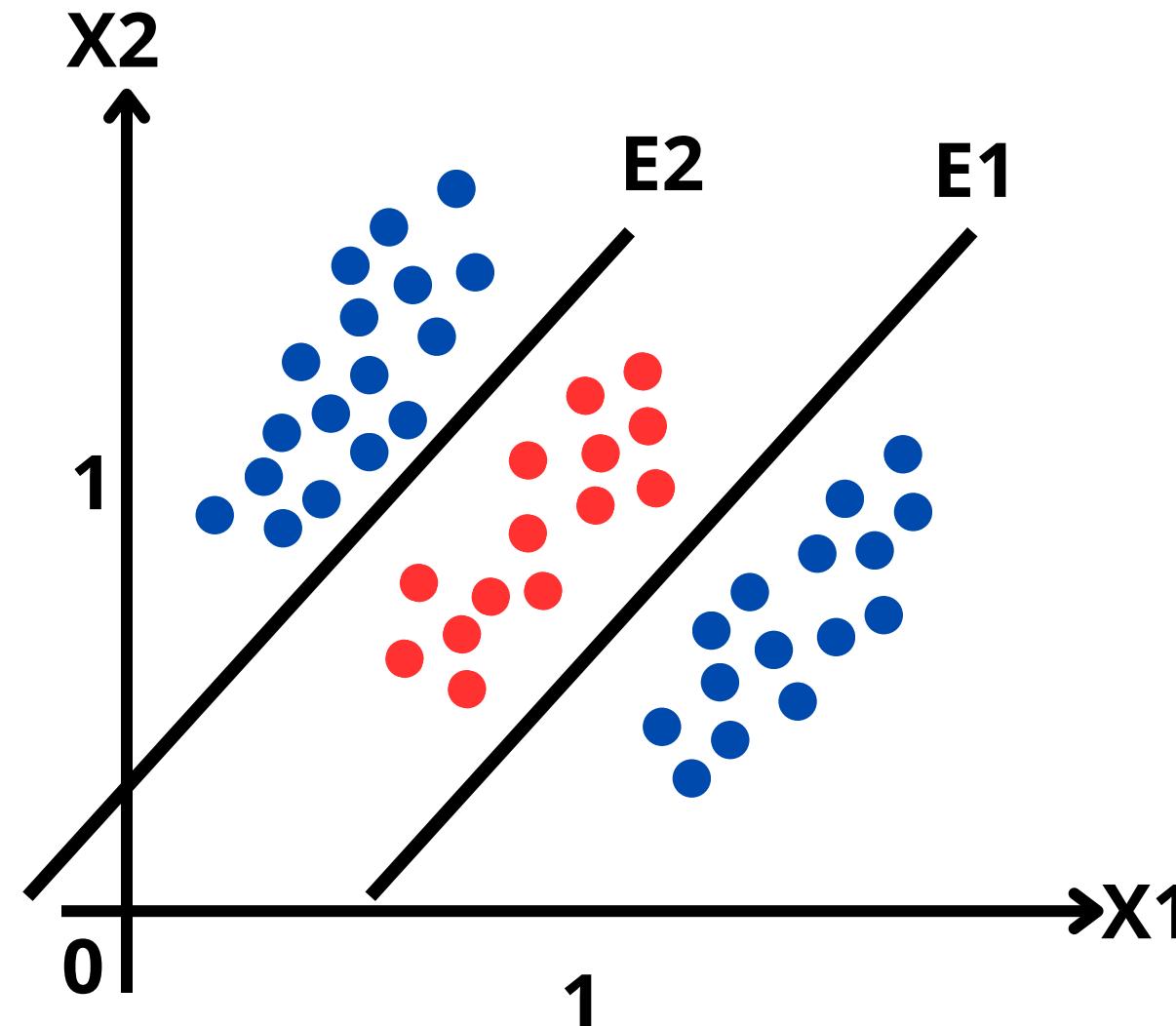


Propagation avant (Forward propagation)

Le processus par lequel les données d'entrée passent à travers les différentes couches d'un réseau de neurones pour produire une sortie.



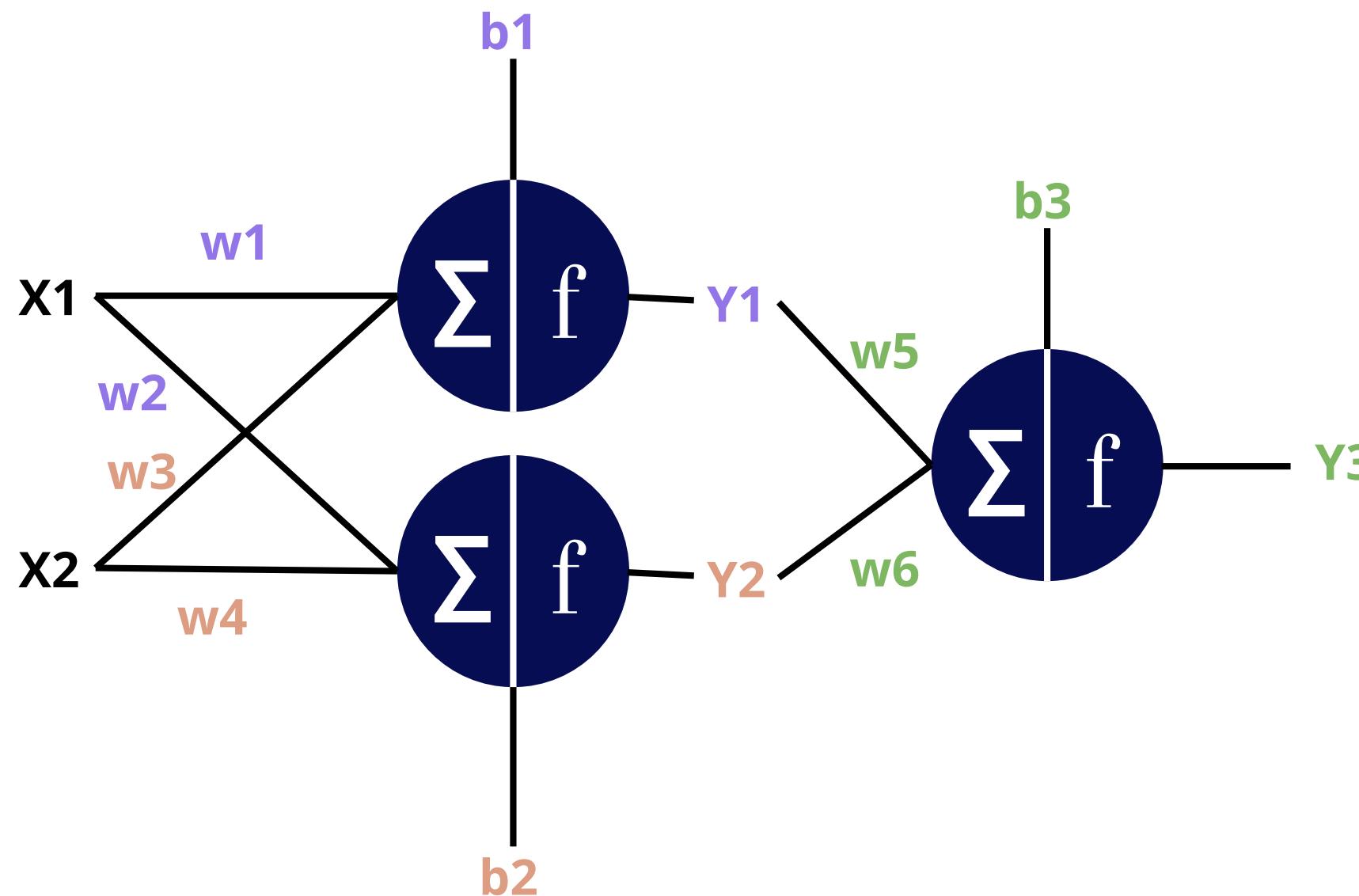
Exemple Forward propagation



X_1	X_2	Y_1	Y_2	Y_3
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

● = 0 ● = 1

Comment trouver les bon poids et biais



Poids	Biais
$w_1: ????$	$b_1: ????$
$w_2: ????$	$b_2: ????$
$w_3: ????$	$b_3: ????$
$w_4: ????$	
$w_5: ????$	
$w_6: ????$	

Comment trouver les bon poids et biais

**Bon Poids
et Biais**

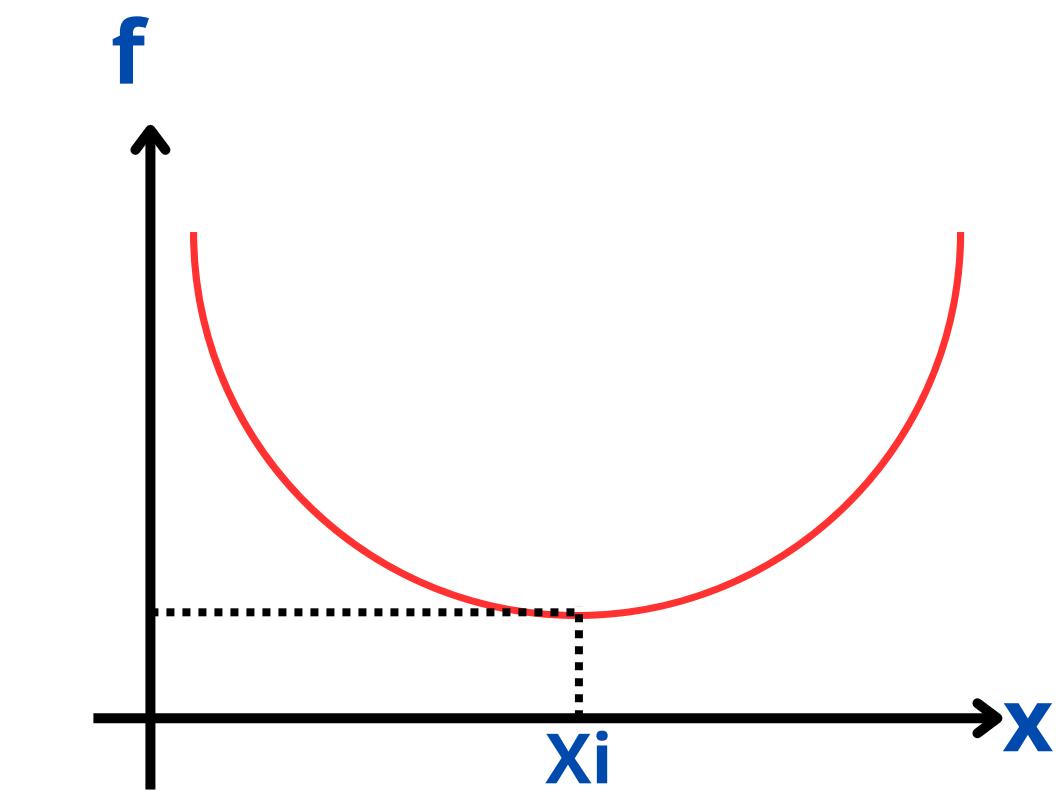
=

**Erreur
minimal**

=

**Fonction
coût minimal**

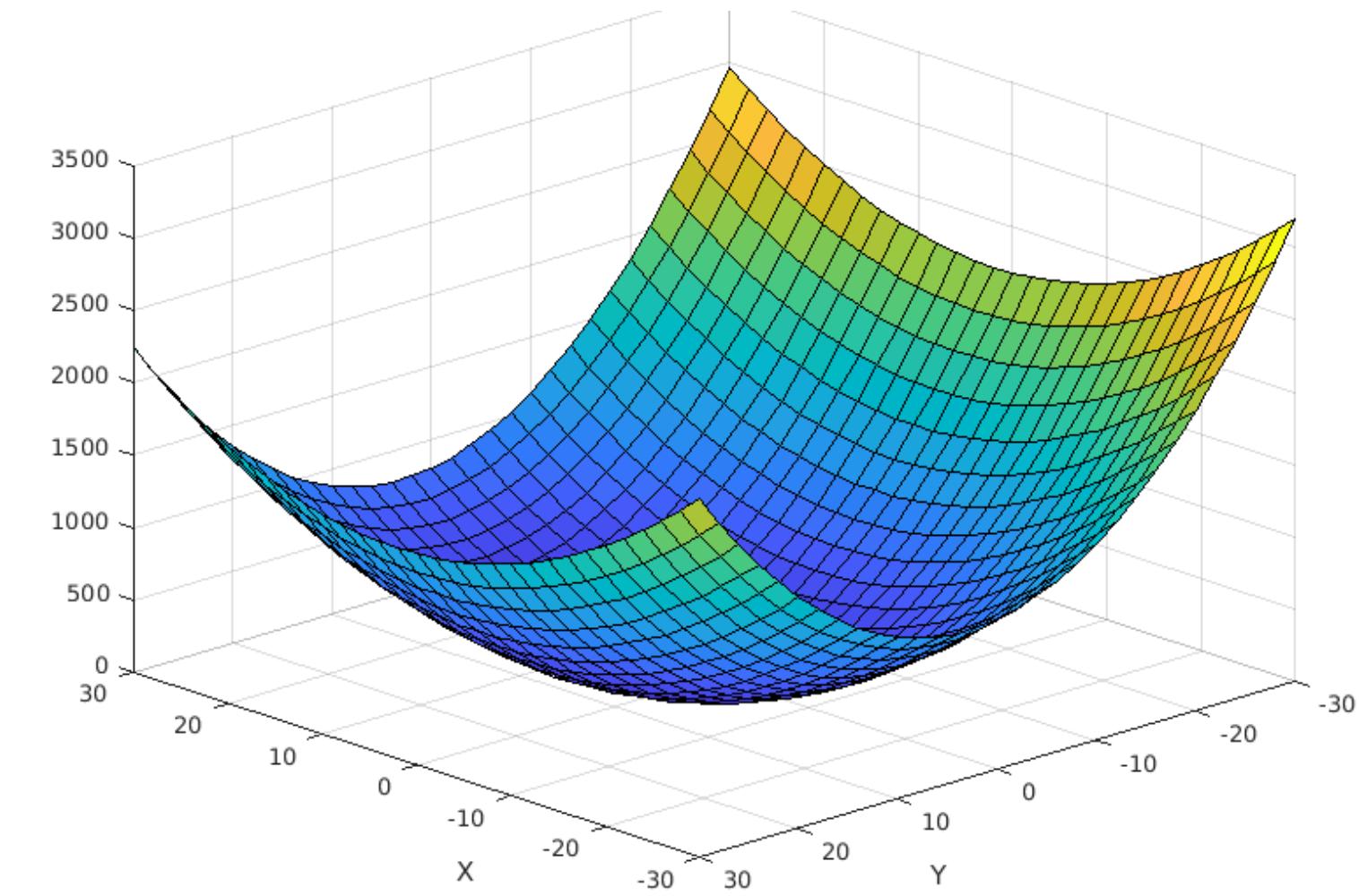
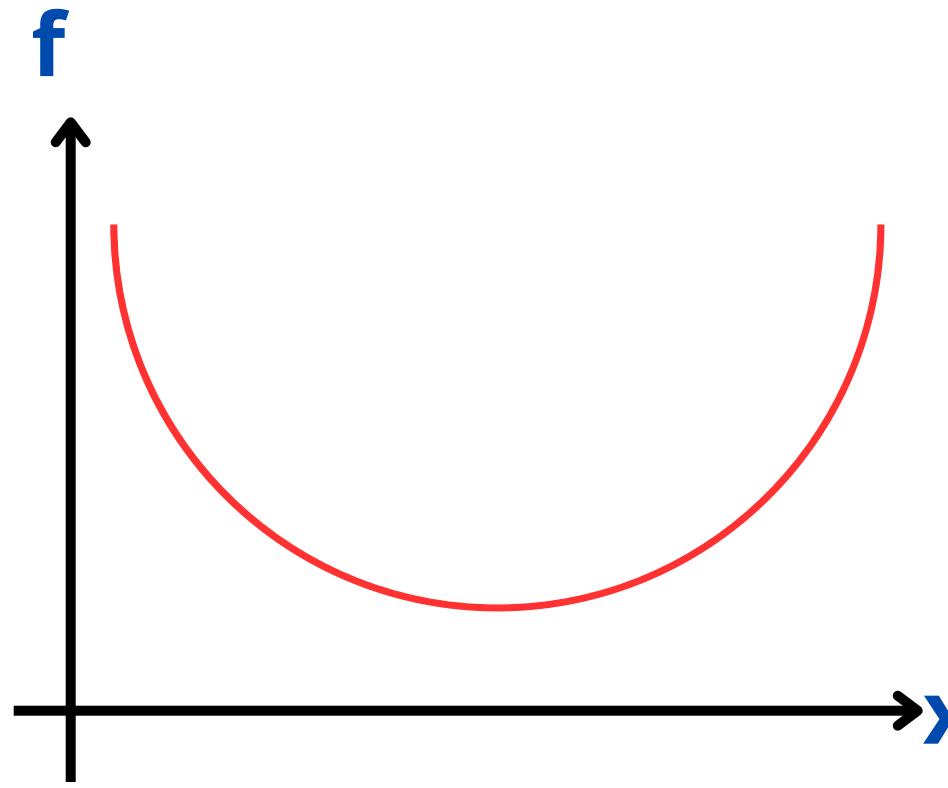
=





Descente de gradient

La descente de gradient est un algorithme d'optimisation itératif utilisé pour trouver les minima d'une fonction. Dans le contexte de l'apprentissage automatique, elle est utilisée pour minimiser la fonction de coût d'un modèle en ajustant les paramètres (poids et biais) afin d'améliorer ses performances.

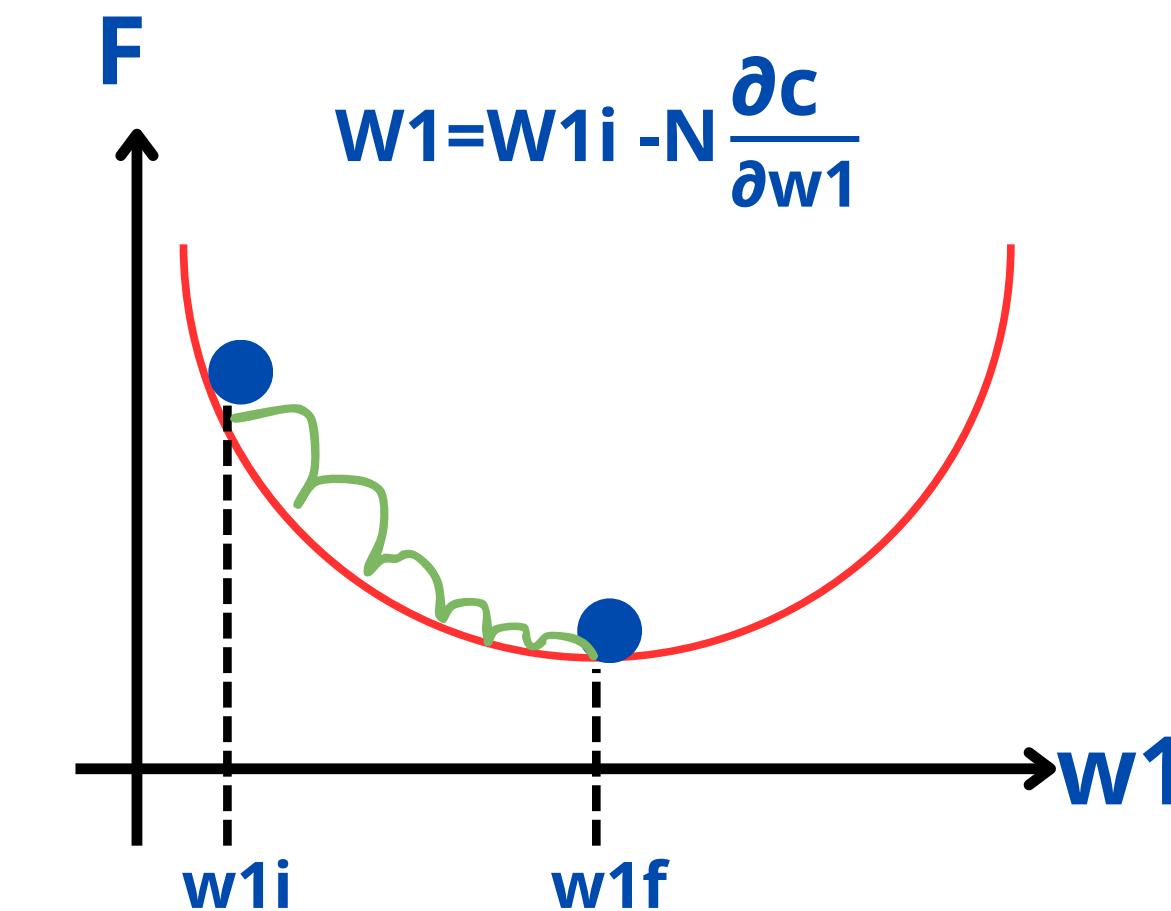
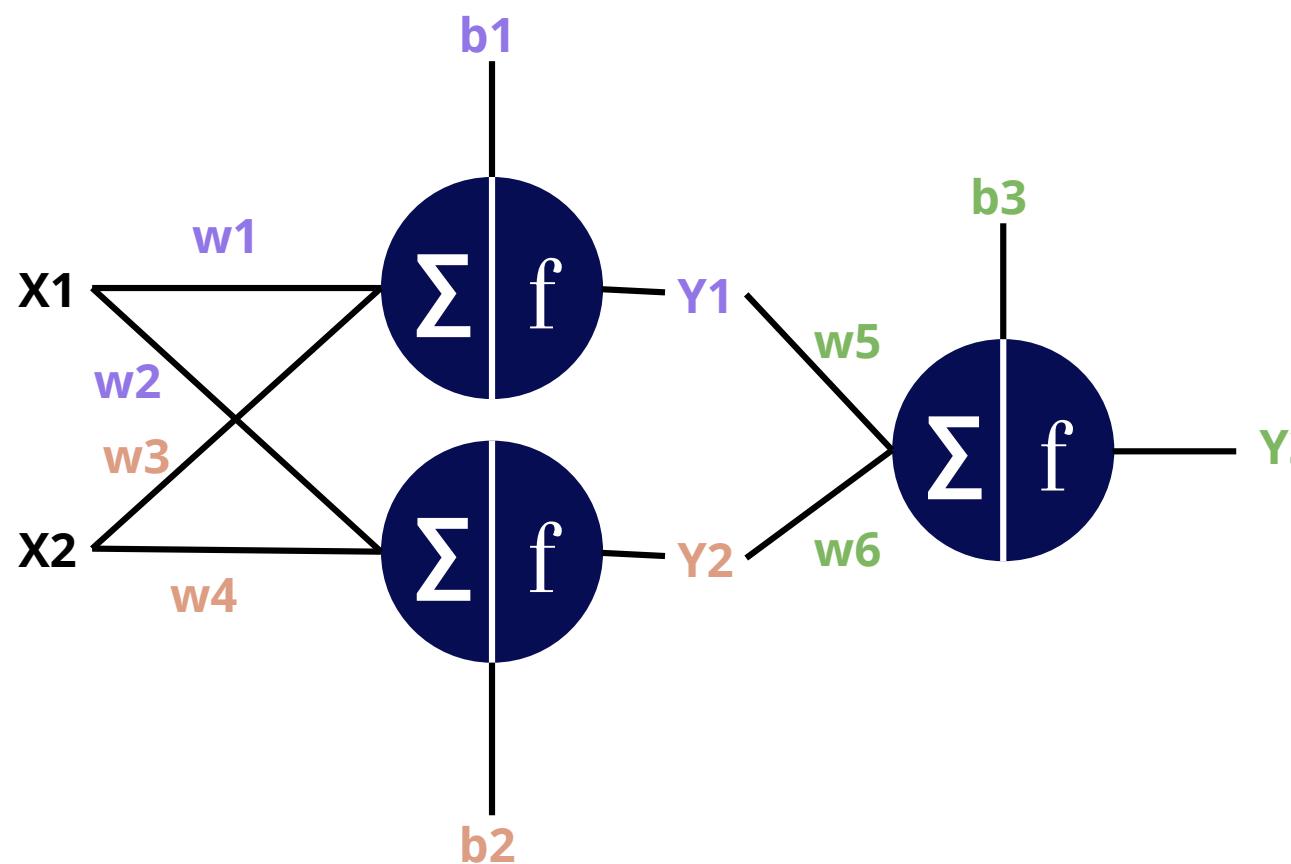


Descente de gradient

Fonction de coût

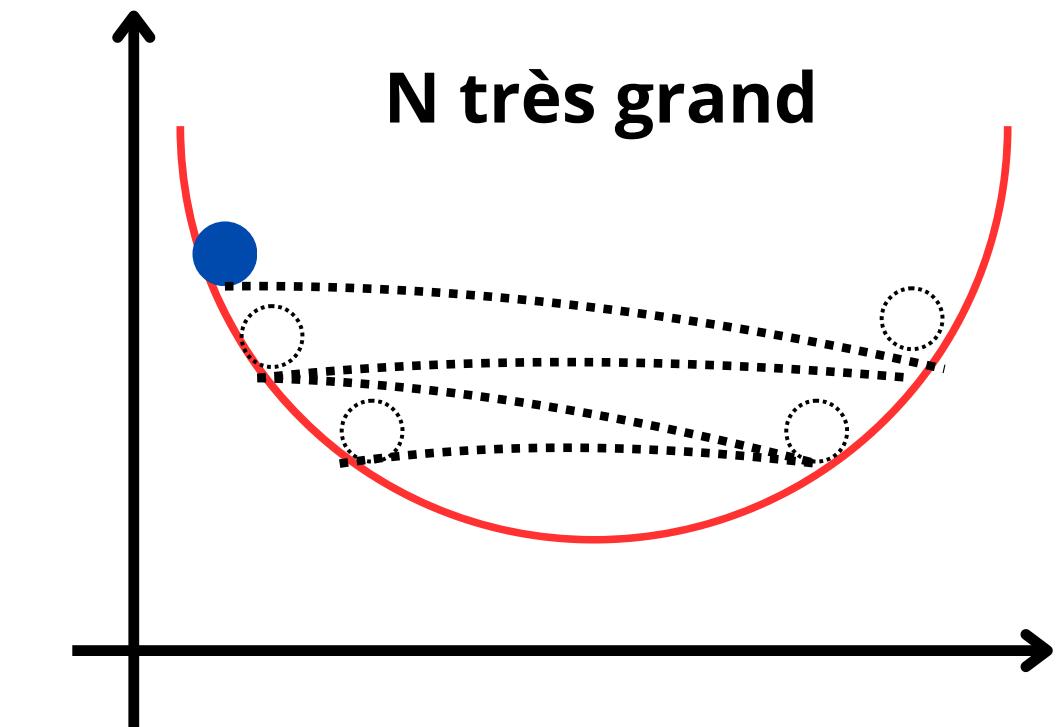
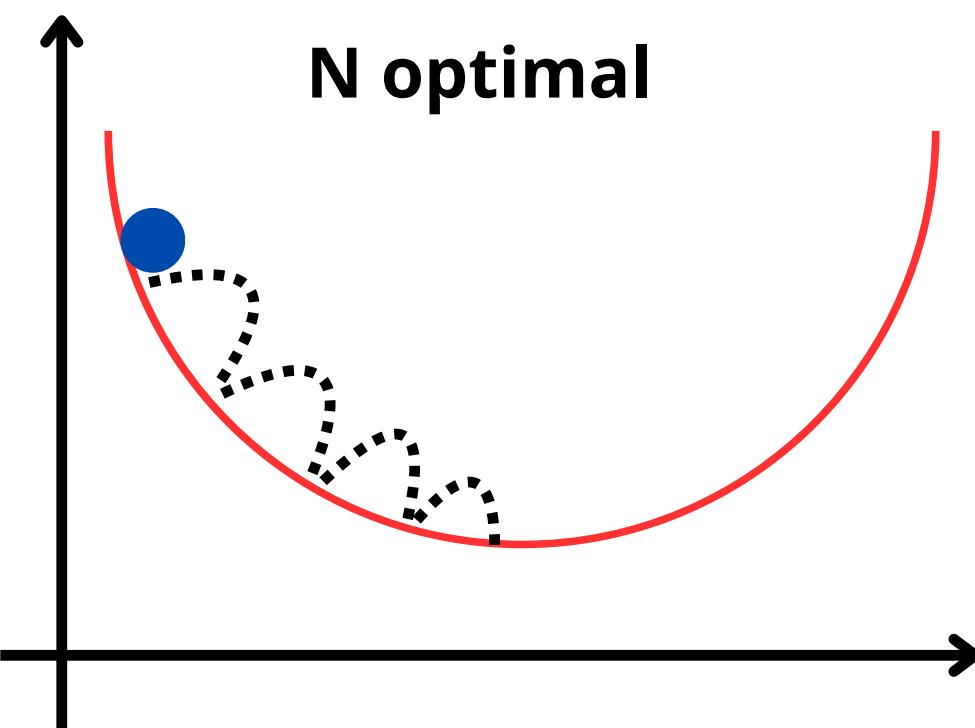
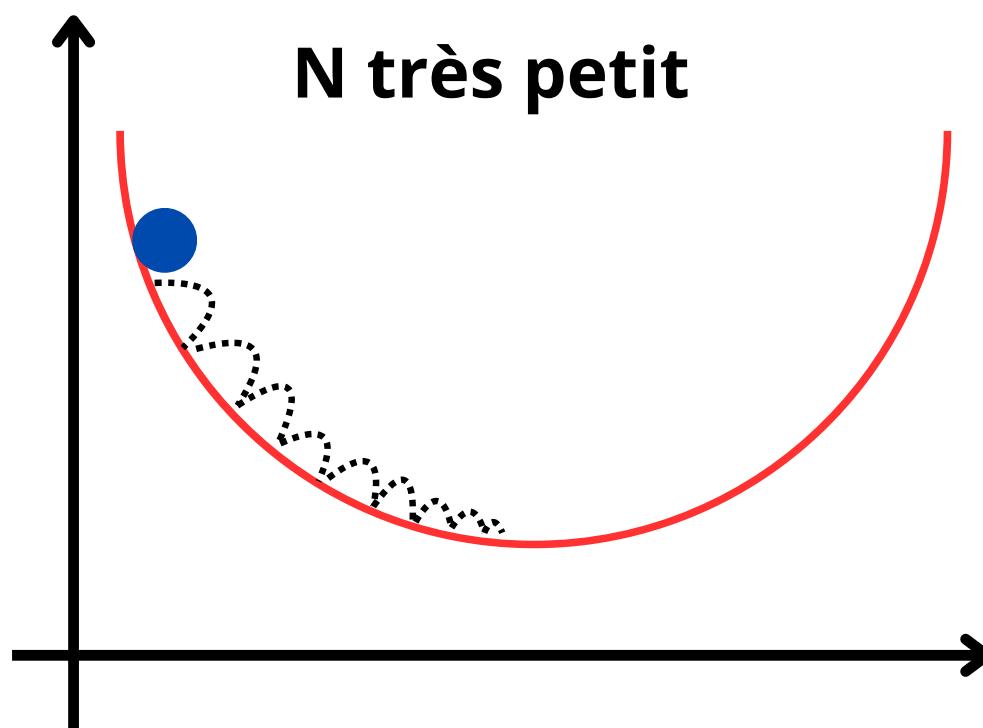
$$F(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

$$\nabla F = \left[\frac{\partial f}{\partial w_1} \frac{\partial f}{\partial w_2} \frac{\partial f}{\partial w_3} \frac{\partial f}{\partial w_4} \frac{\partial f}{\partial w_5} \frac{\partial f}{\partial w_6} \frac{\partial f}{\partial b_1} \frac{\partial f}{\partial b_2} \frac{\partial f}{\partial b_3} \right]$$

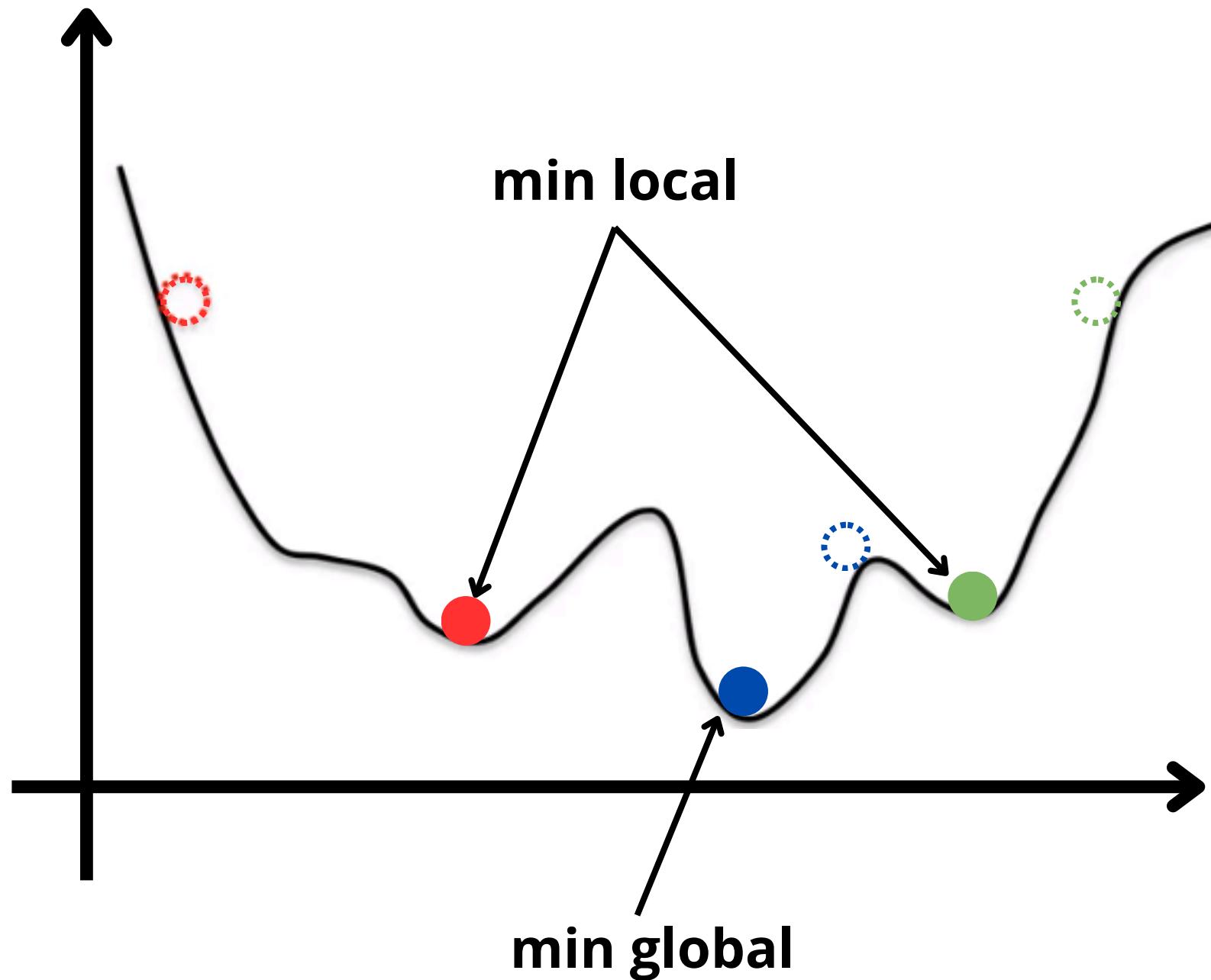


Le taux d'apprentissage

$$w_1 = w_{1i} - N \frac{\partial c}{\partial w_1}$$

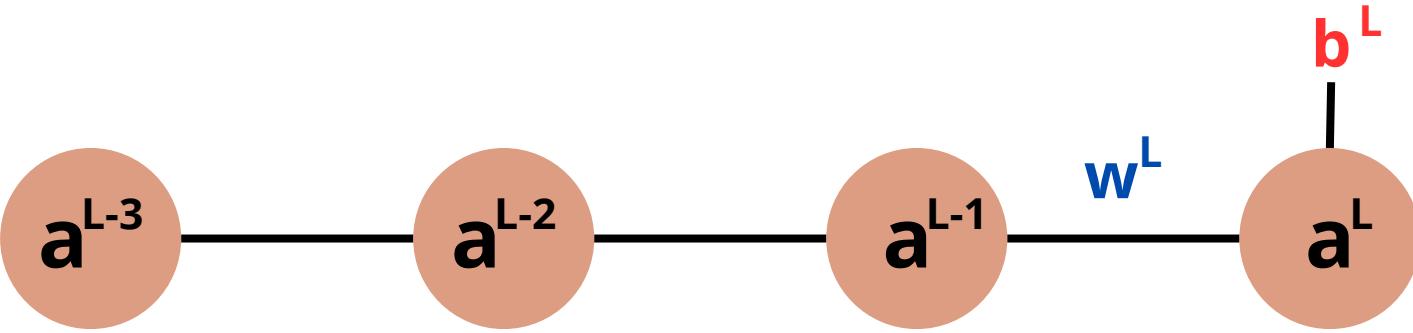
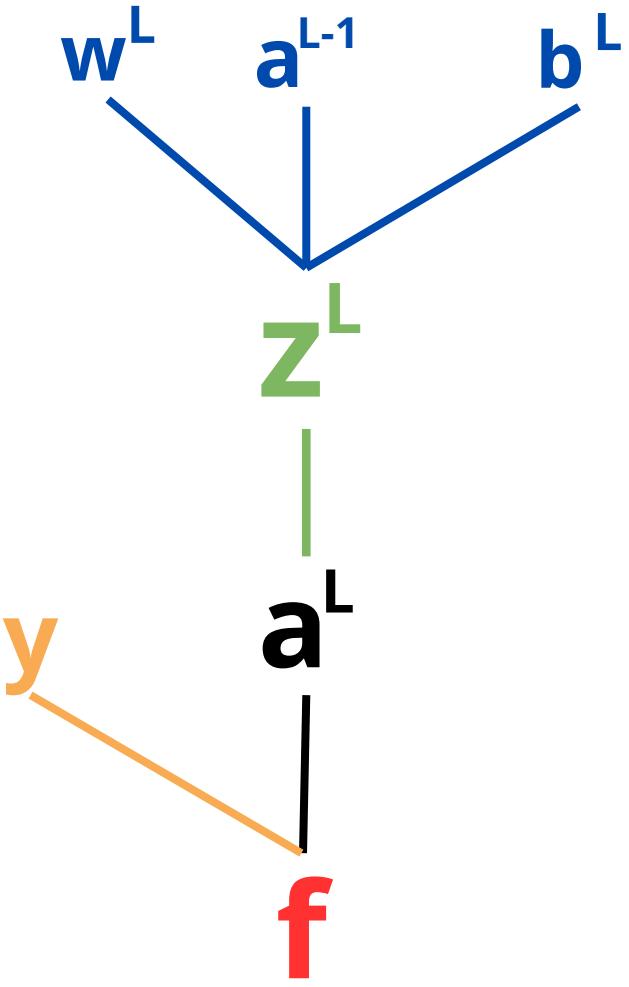


les limites de descente de gradient



La solution:

répéter l'apprentissage avec des paramètres aléatoire



$$\frac{\partial f}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial f}{\partial a^L} = a^{L-1} \sigma'(z^L) 2(a^L - y)$$

$$\frac{\partial z^L}{\partial w^L} = a^{L-1}$$

$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$$

$$\frac{\partial f}{\partial a^L} = 2(a^L - y)$$

$$\frac{\partial f}{\partial b^L} = \frac{\partial z^L}{\partial b^L} \frac{\partial a^L}{\partial z^L} \frac{\partial f}{\partial a^L} = \sigma'(z^L) 2(a^L - y)$$

$$f = (a^L - y)^2$$

$$a^L = \sigma(w^L a^{L-1} + b^L)$$

$$z^L = w^L a^{L-1} + b^L$$

$$a^L = \sigma(z^L)$$

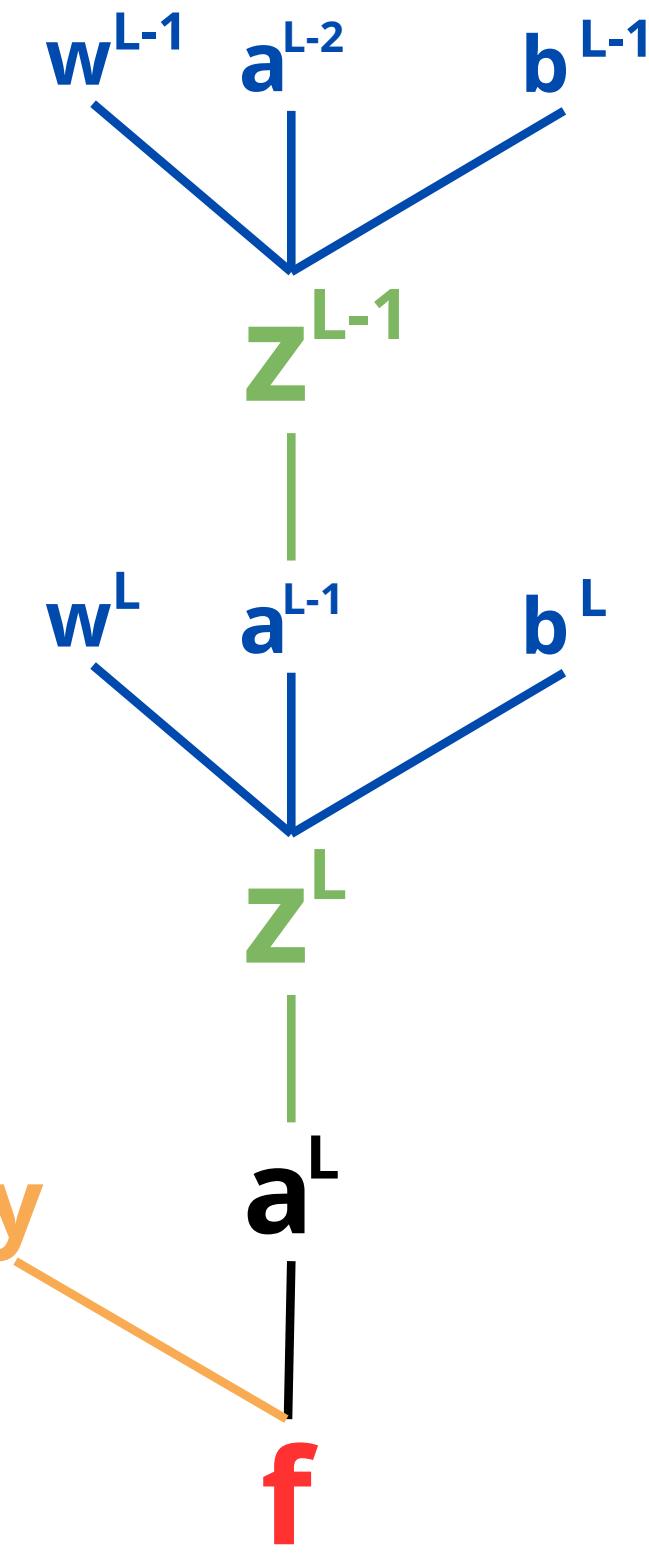


Diagram illustrating a neural network chain across layers $L-3$ to L . Inputs a^{L-3} and b^{L-1} lead to a^{L-2} . a^{L-2} and b^{L-1} lead to a^{L-1} . a^{L-1} and b^L lead to a^L . The final output f is produced from a^L .

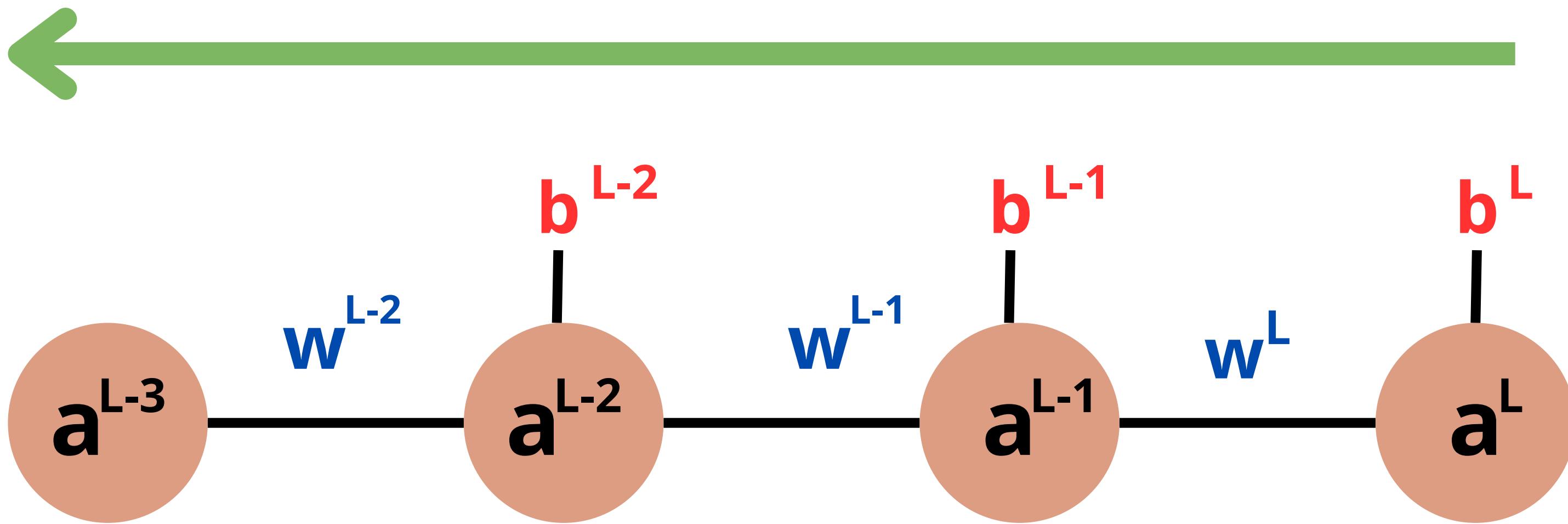
$$\frac{\partial \mathbf{f}}{\partial \mathbf{w}^L} = \frac{\partial \mathbf{z}^L}{\partial \mathbf{w}^L} \underbrace{\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} \frac{\partial \mathbf{f}}{\partial \mathbf{a}^L}}_{\delta^L}$$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{w}^{L-1}} = \frac{\partial \mathbf{z}^{L-1}}{\partial \mathbf{w}^{L-1}} \underbrace{\frac{\partial \mathbf{a}^{L-1}}{\partial \mathbf{z}^{L-1}} \frac{\partial \mathbf{z}^L}{\partial \mathbf{a}^{L-1}} \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} \frac{\partial \mathbf{f}}{\partial \mathbf{a}^L}}_{\delta^{L-1}}$$

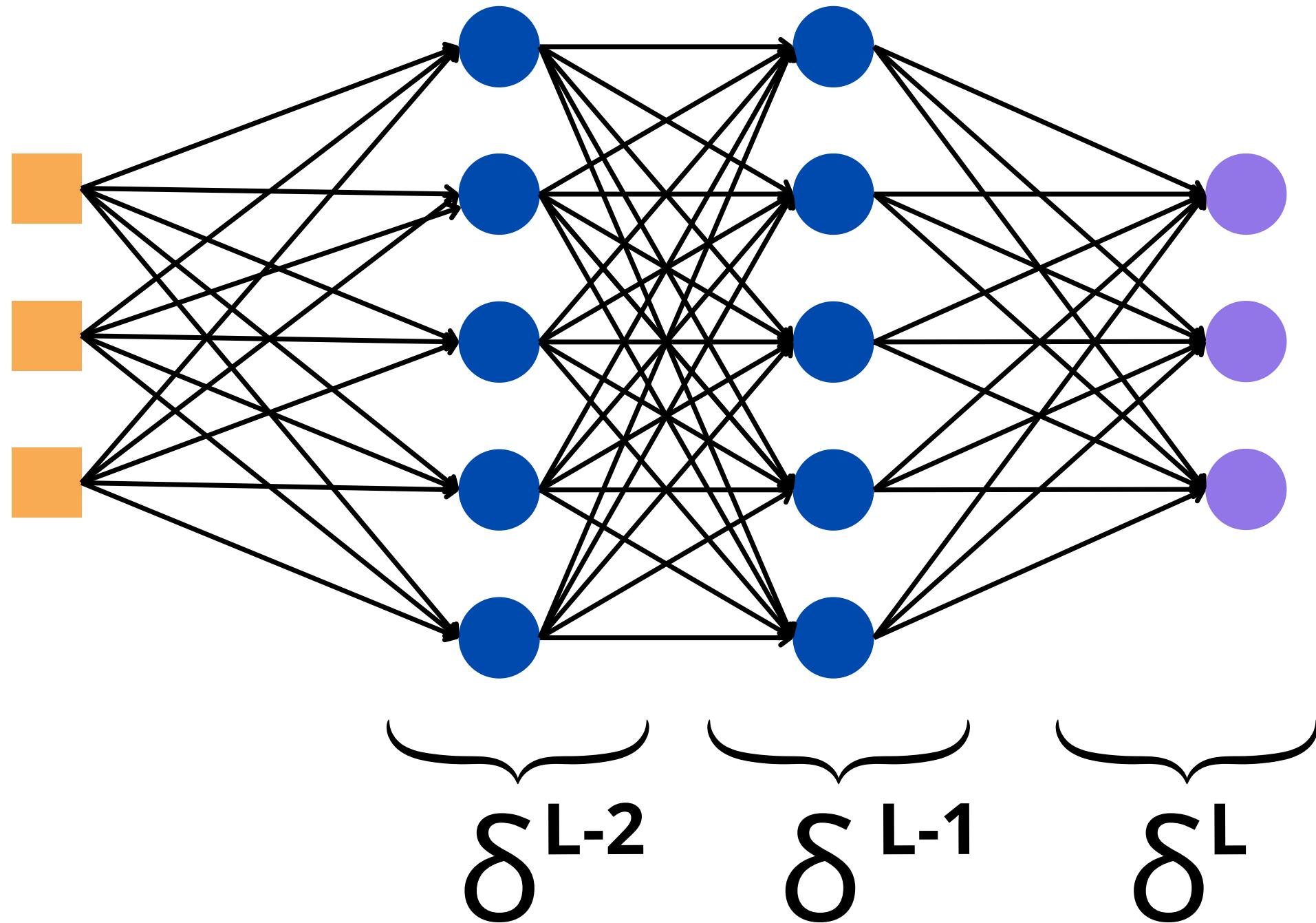
$$\frac{\partial \mathbf{f}}{\partial \mathbf{w}^{L-1}} = \frac{\partial \mathbf{z}^{L-1}}{\partial \mathbf{w}^{L-1}} \delta^{L-1} ; \quad \delta^{L-1} = \frac{\partial \mathbf{a}^{L-1}}{\partial \mathbf{z}^{L-1}} \frac{\partial \mathbf{z}^L}{\partial \mathbf{a}^{L-1}} \delta^L$$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{w}^{L-i}} = \frac{\partial \mathbf{z}^{L-i}}{\partial \mathbf{w}^{L-i}} \delta^{L-i} ; \quad \delta^{L-i} = \frac{\partial \mathbf{a}^{L-i}}{\partial \mathbf{z}^{L-i}} \frac{\partial \mathbf{z}^{L-i+1}}{\partial \mathbf{a}^{L-i}} \delta^{L-i+1}$$

Back-propagation

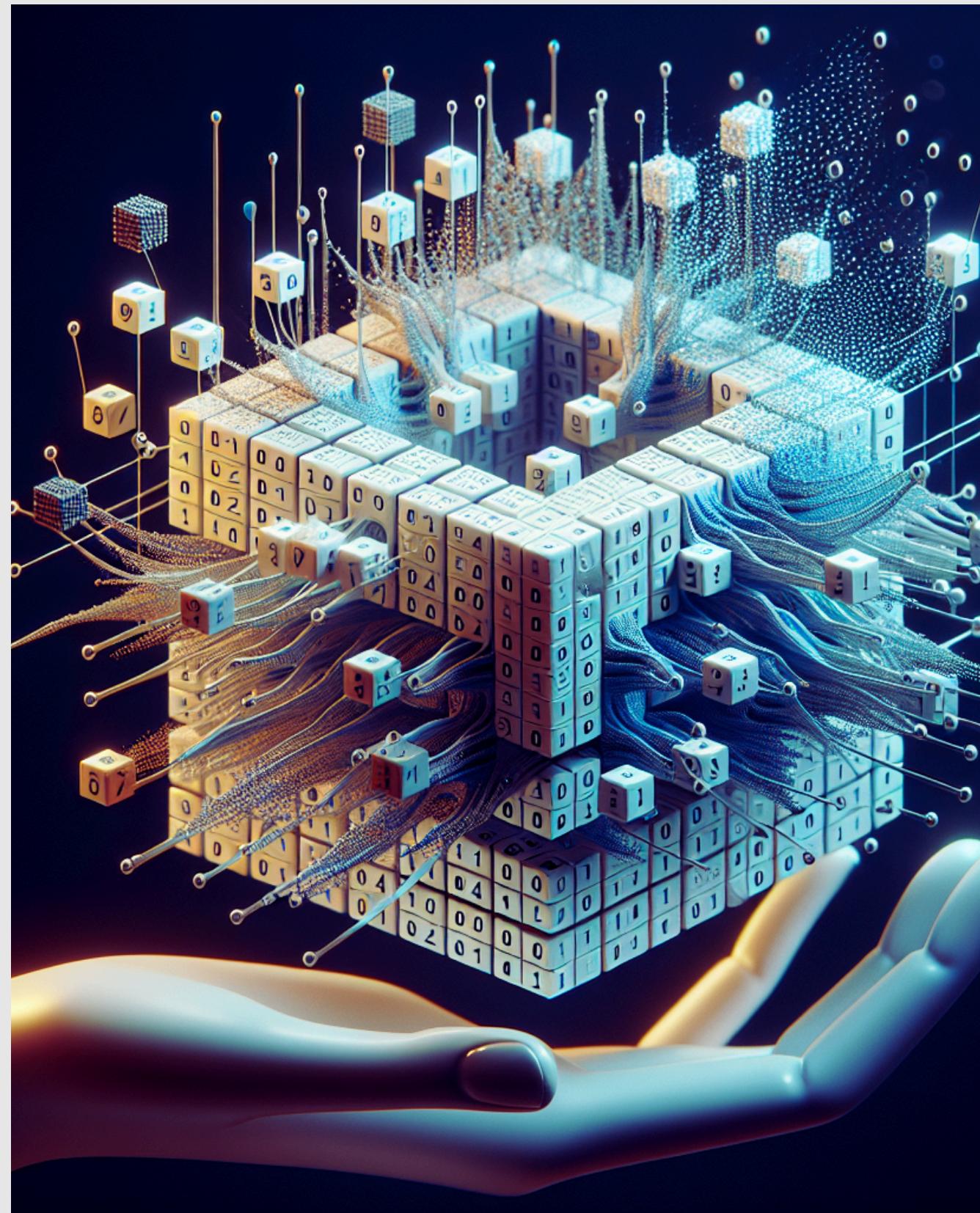


Apprentissage



- Choisir les fonctions d'activation
- Choisir la fonction de coût
- Calculer les Deltas
- Ajuster les paramètres

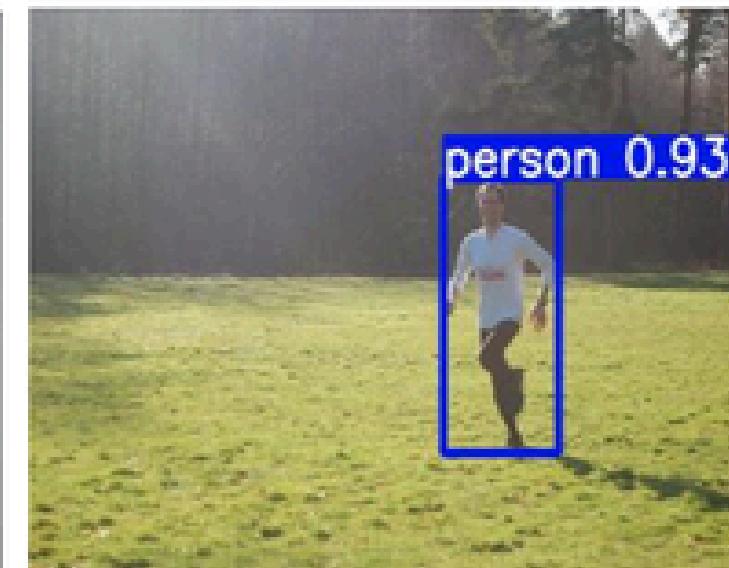
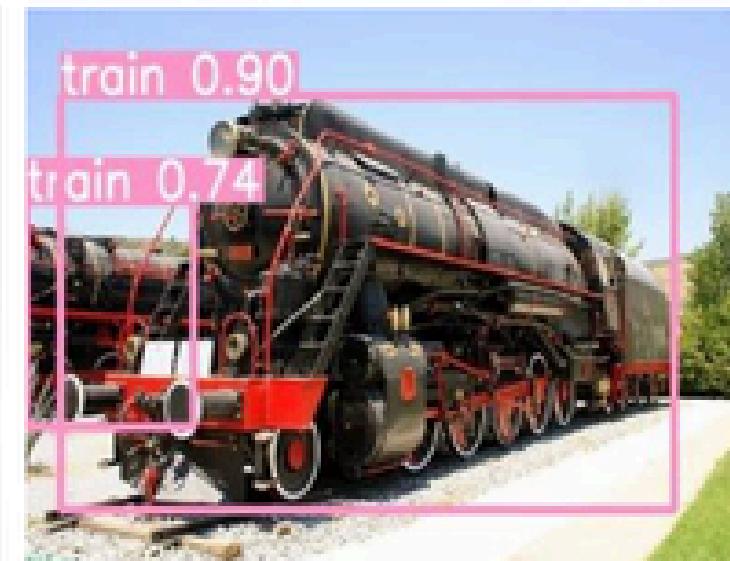
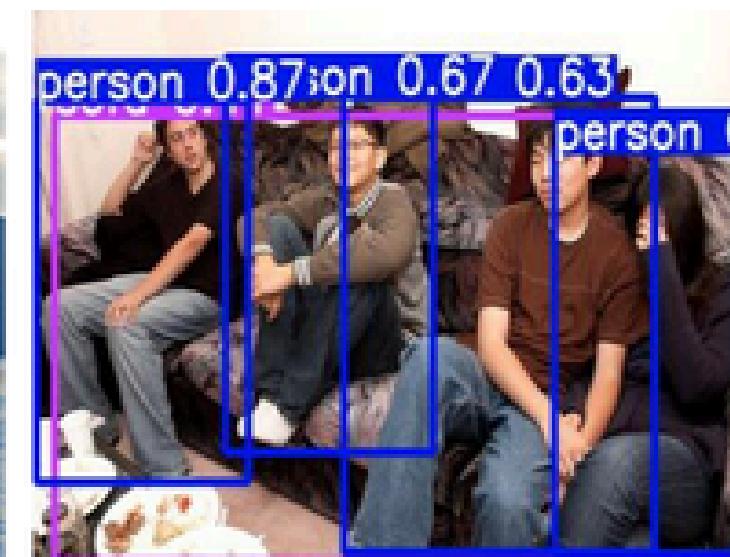
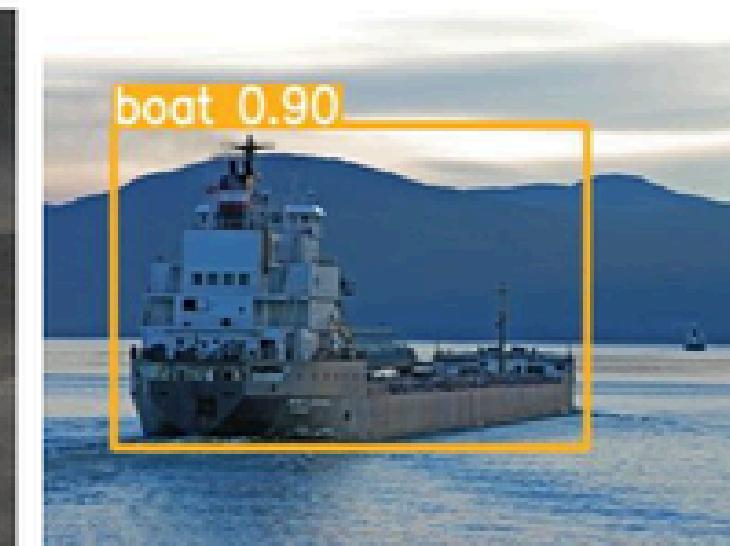
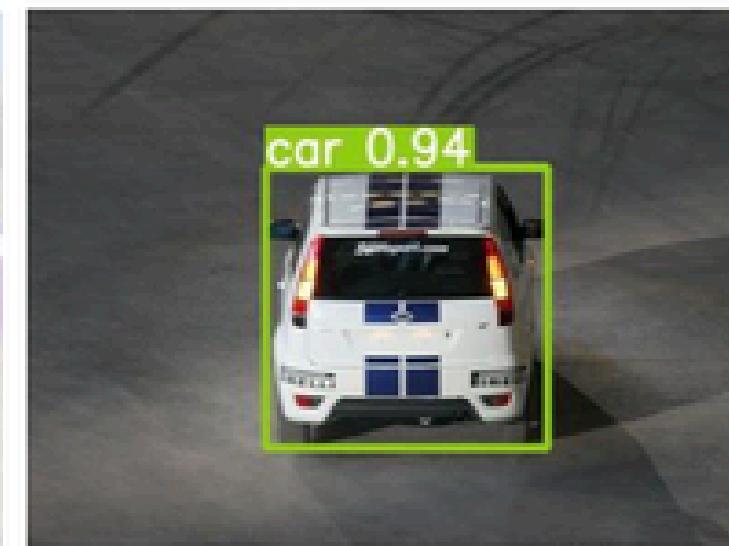
LES RESEAUX DE NEURONES CONVOLUTIFS



Définition:

Les réseaux de neurones convolutifs (CNN) sont un type de réseaux de neurones artificiels conçus spécifiquement pour traiter et analyser des images. Ils sont largement utilisés dans des tâches telles que la reconnaissance d'images et la détection d'objets grâce à leur capacité à extraire automatiquement des caractéristiques visuelles.

Fonctionnement du CNN



(a) small object

(b) medium object

(c) large object

(d) multi-object

Exemple d'analyse d'image dans un CNN

CNN traduit une image en données exploitables pour détecter et reconnaître des motifs visuels

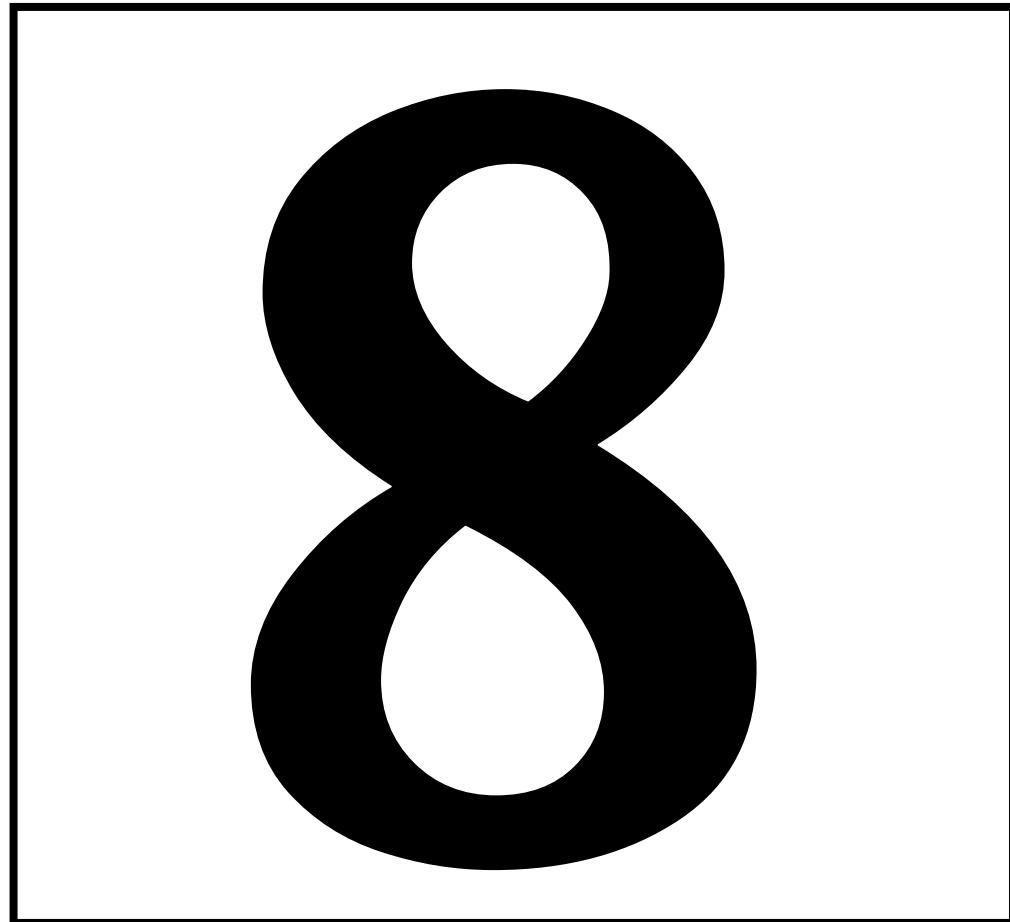
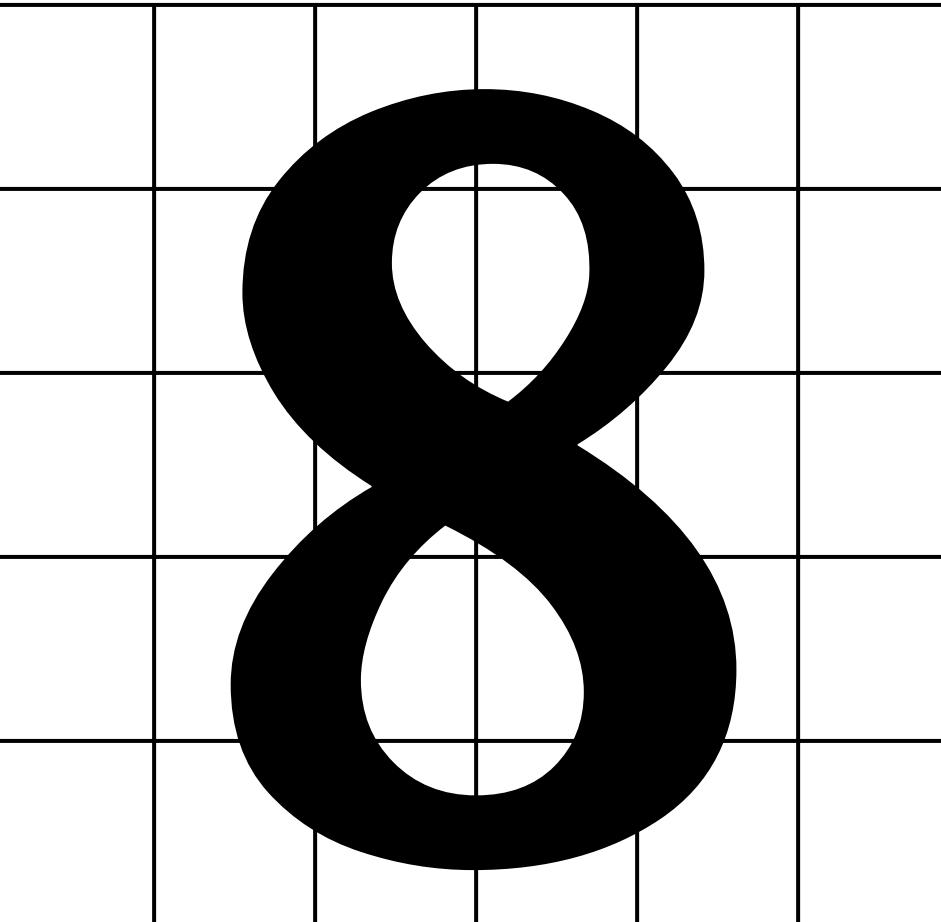


Image réelle du chiffre 8



Représentation sous forme de matrice

0	0	1	1	0	0
0	1	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	0	1	1	0	0

Représentation binaire

Couche de convolution :

Une couche de convolution contient un certain nombre de filtres qui effectuent une opération de convolution.

- Chaque image est considérée comme une matrice de valeurs de pixels.
 - Considérez l'image suivante de taille 5x5 dont les valeurs des pixels sont uniquement 0 et 1.

Image en pixels

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

(f x f)

Caractéristique convoluée :
(Matrice résultante après convolution)

4	3	4
2	4	3
2	3	4

$$(n - f + 1 \times n - f + 1)$$

$$y = \sum (\text{image} \times \text{filtre})$$

Image en pixels

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
(Matrice résultante après convolution)

4		

Image en pixels

1	1 x_1	1 x_0	0 x_1	0
0	1 x_0	1 x_1	1 x_0	0
0	0 x_1	1 x_0	1 x_1	1
0	0	1	1	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
(Matrice résultante après convolution)

4	3	

Image en pixels

1	1	x1 1	0 x0	0 x1
0	1	x0 1	x1 1	0 x0
0	0	x1 1	x0 1	x1 1
0	0	1	1	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
(Matrice résultante après convolution)

4	3	4

Image en pixels

1	1	1	0	0
0 x1	1 x0	x1 1	1	0
x0 0	0 x1	1 x0	1	1
x1 0	x0 0	1 x1	1	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
(Matrice résultante après convolution)

4	3	4
2		

Image en pixels

1	1	1	0	0
0	1_{x1}	1_{x0}	1_{x1}	0
0	0_{x0}	$x11$	1_{x0}	1
0	0_{x1}	1_{x0}	1_{x1}	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
(Matrice résultante après convolution)

4	3	4
2	4	

Image en pixels

1	1	1	0	0
0	1	1_{x1}	1_{x0}	0_{x1}
0	0	1_{x0}	1_{x1}	1_{x0}
0	0	1_{x1}	1_{x0}	0_{x1}
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
 (Matrice résultante après convolution)

4	3	4
2	4	3

Image en pixels

1	1	1	0	0
0	1	1	1	0
x1 0	x0 0	x1 1	1	1
0 x0	0 x1	1 x0	1	0
x1 0	x0 1	1 x1	0	0

Filtre

1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
(Matrice résultante après convolution)

4	3	4
2	4	3
2		

Image en pixels

1	1	1	0	0
0	1	1	1	0
0	0 x_1	1 x_0	1 x_1	1
0	x_0 0	1 x_1	1 x_0	0
0	1 x_1	1 x_0	0 x_1	0

Filtre

1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
(Matrice résultante après convolution)

4	3	4
2	4	3
2	3	

Image en pixels

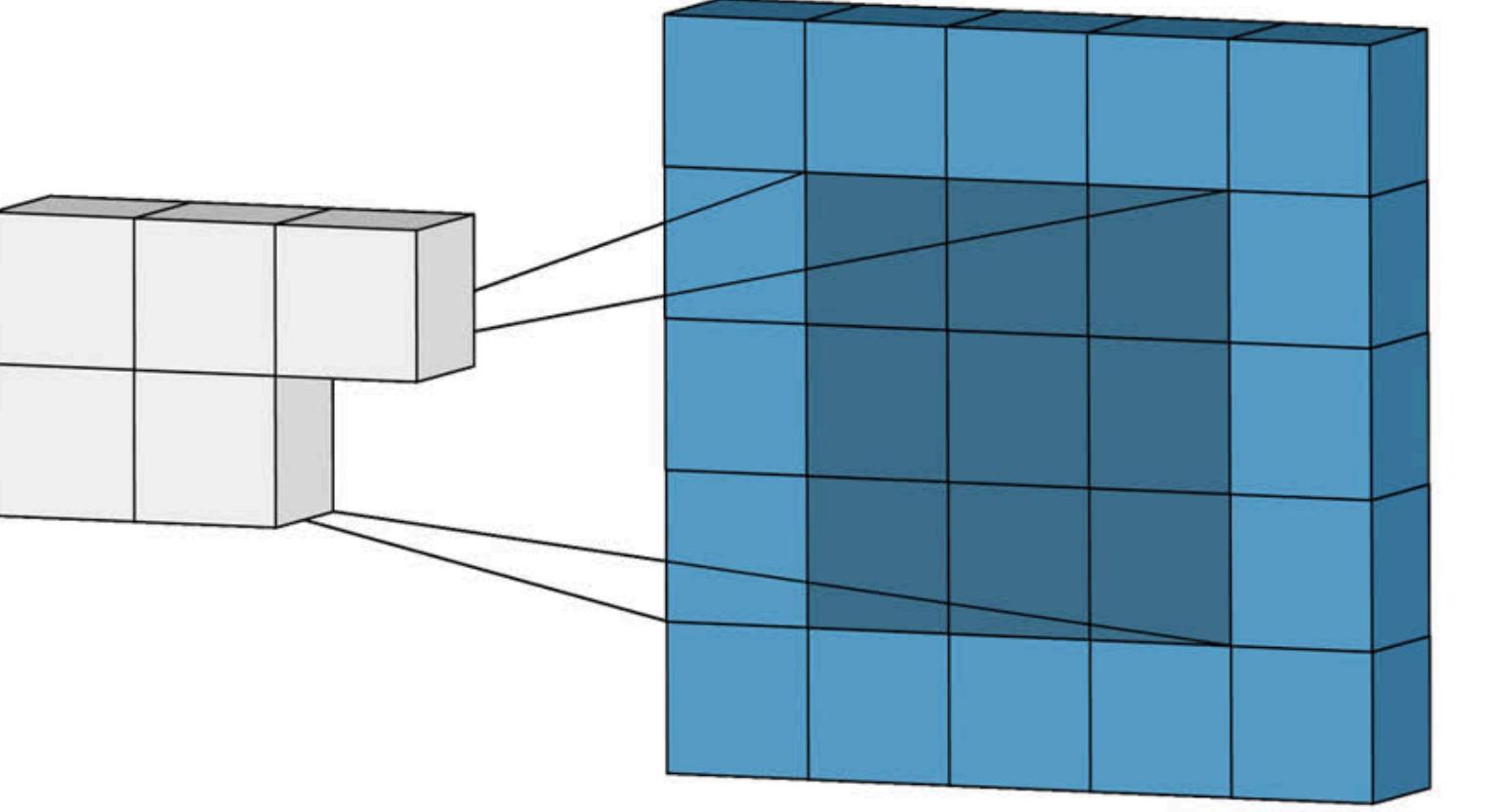
1	1	1	0	0
0	1	1	1	0
0	0	1 <i>x1</i>	1 <i>x0</i>	1 <i>x1</i>
0	0	1 <i>x0</i>	1 <i>x1</i>	0 <i>x0</i>
0	1	1 <i>x1</i>	0 <i>x0</i>	0 <i>x1</i>

Filtre

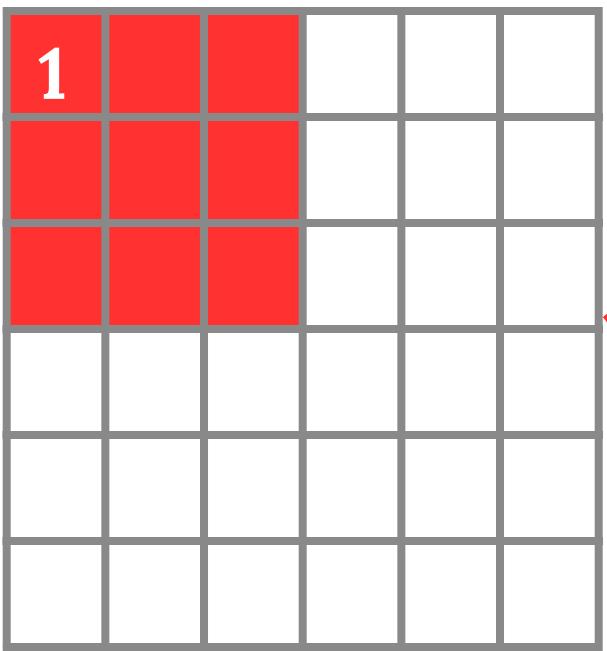
1	0	1
0	1	0
1	0	1

Caractéristique convoluée :
(Matrice résultante après convolution)

4	3	4
2	4	3
2	3	4

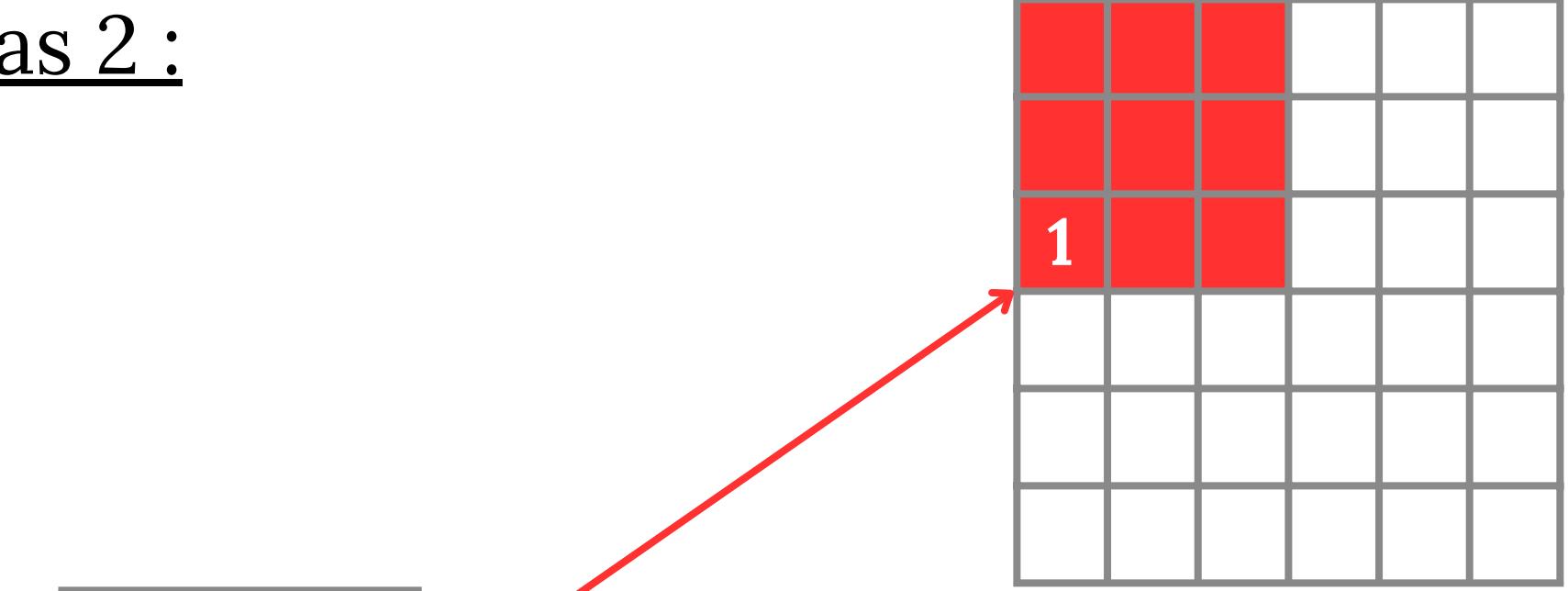


Cas 1 :

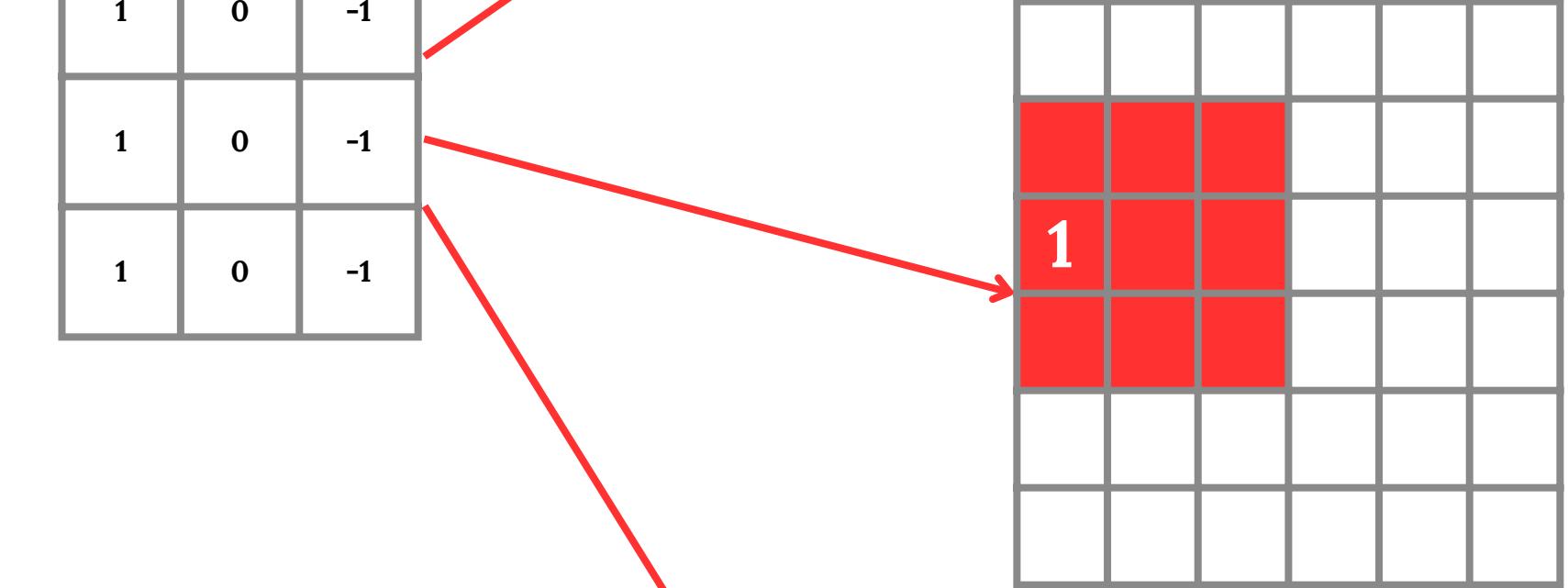


1	0	-1
1	0	-1
1	0	-1

1	0	-1
1	0	-1
1	0	-1

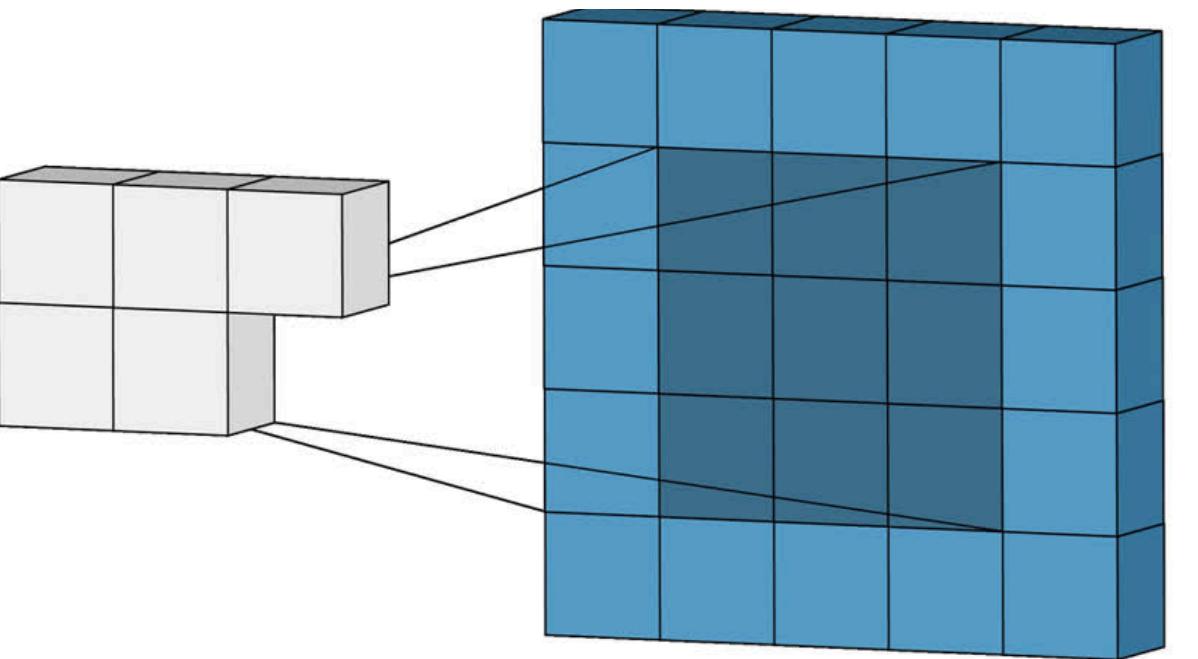


1	0	-1
1	0	-1
1	0	-1

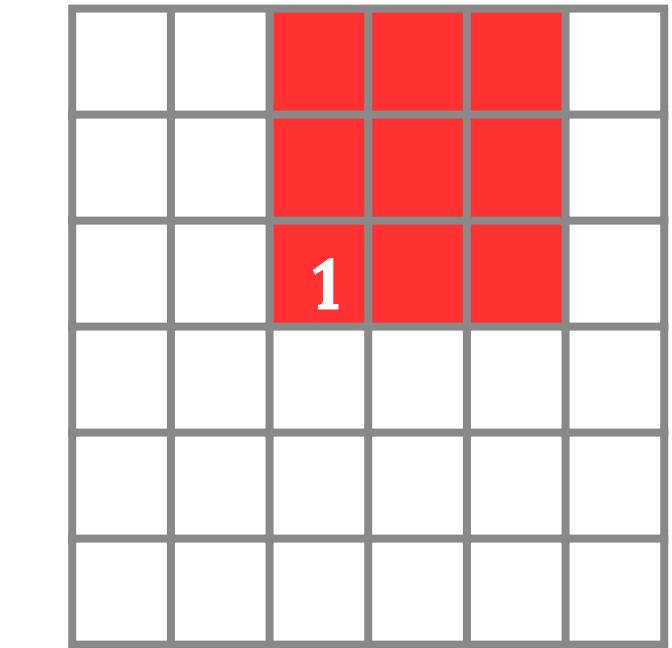
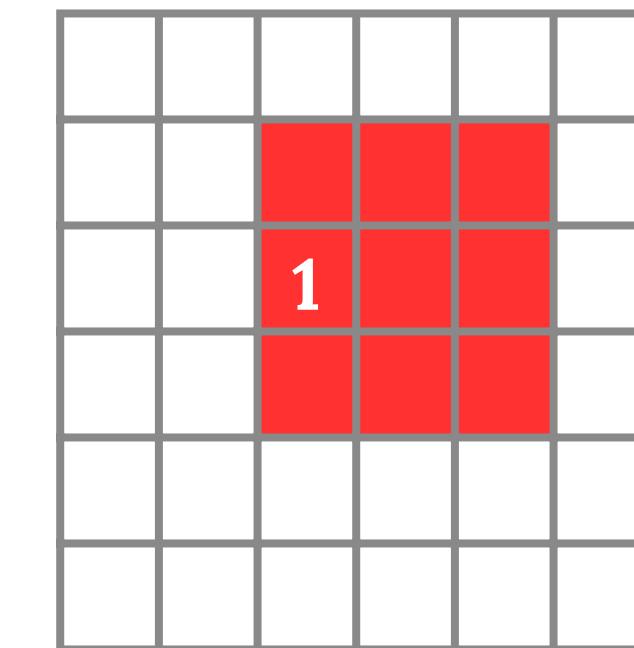
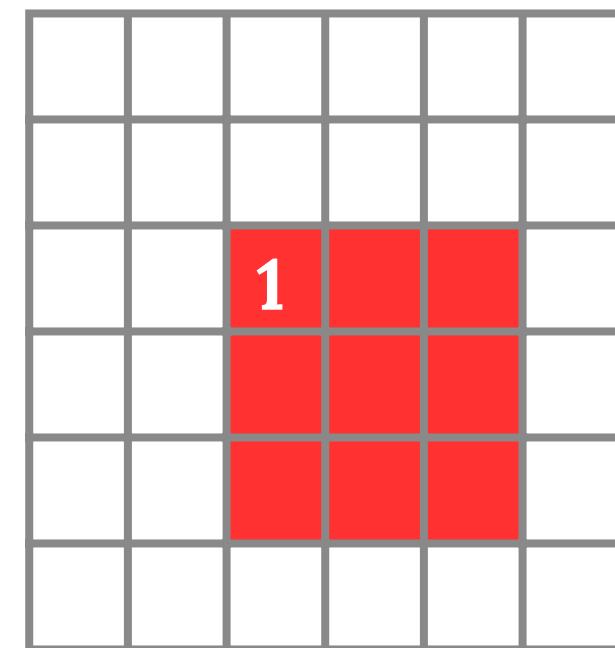
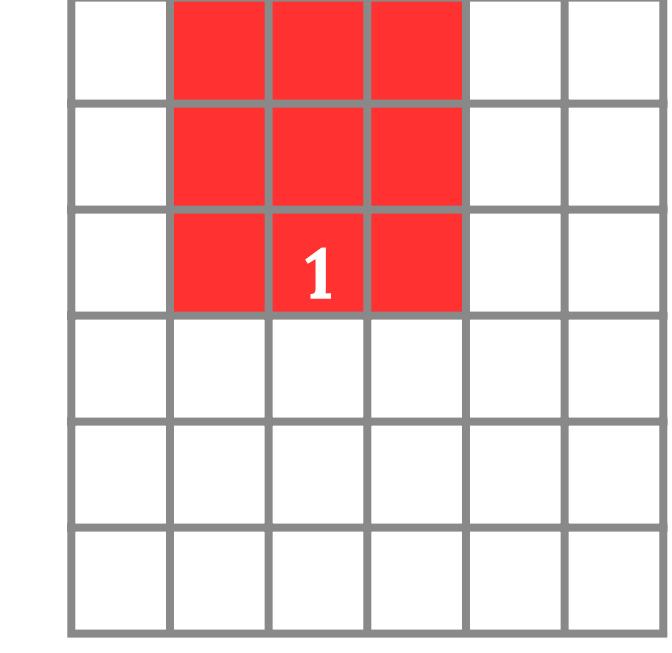
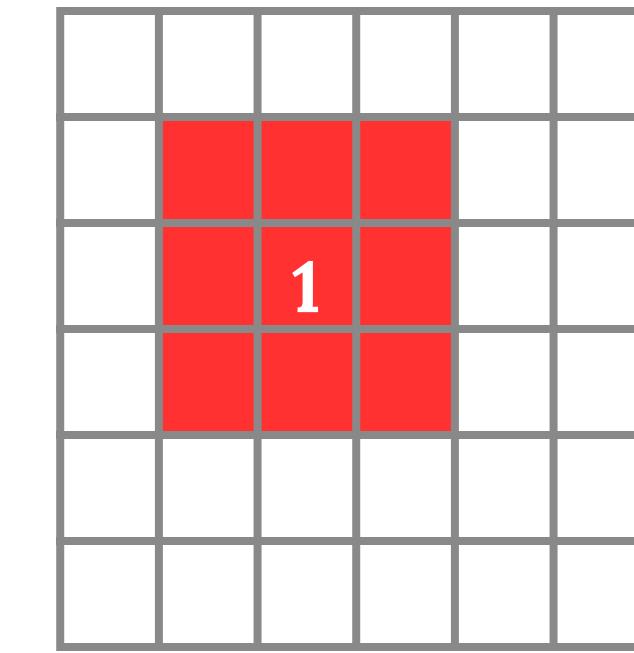
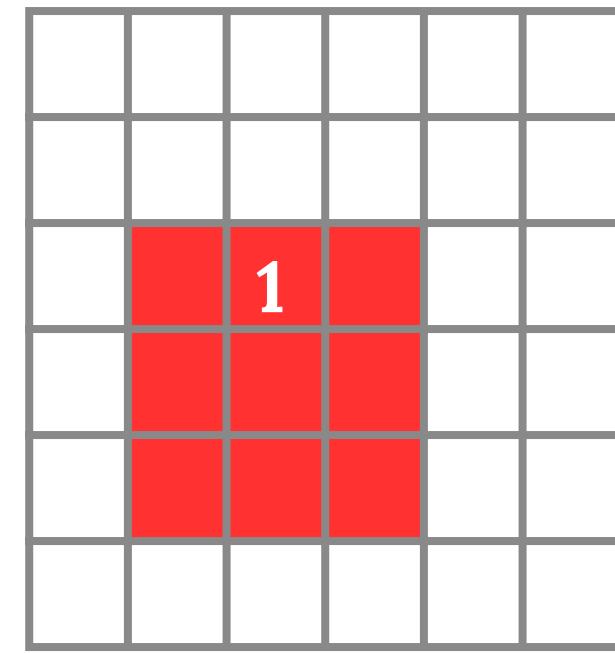
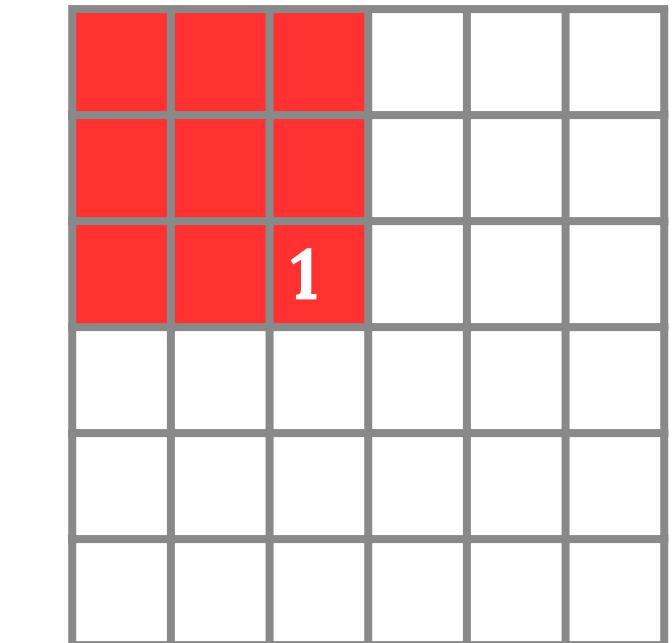
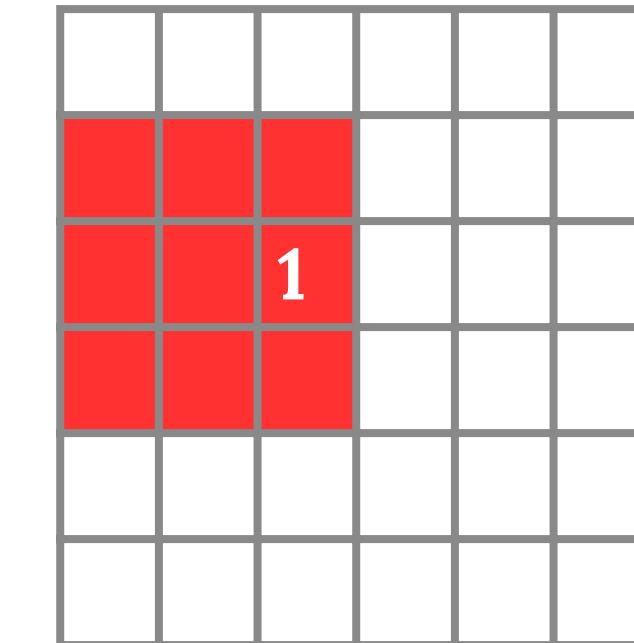
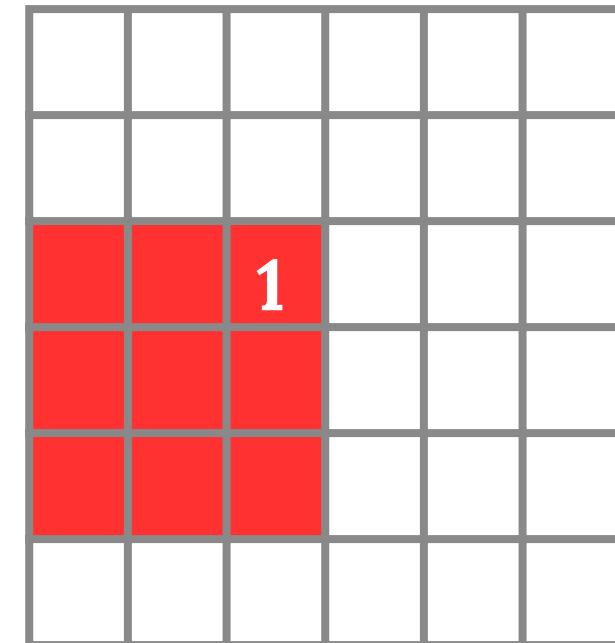
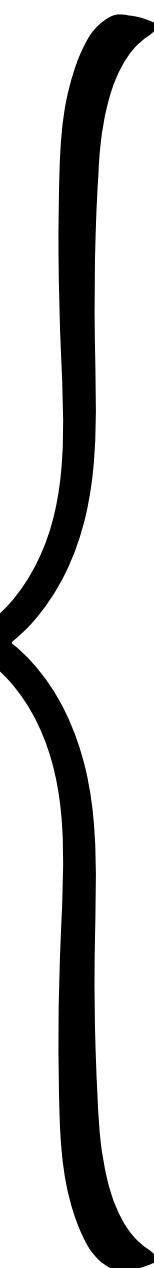
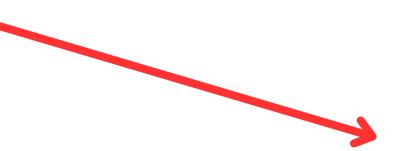


1	0	-1
1	0	-1
1	0	-1

Cas 3 :



1	0	-1
1	0	-1
1	0	-1

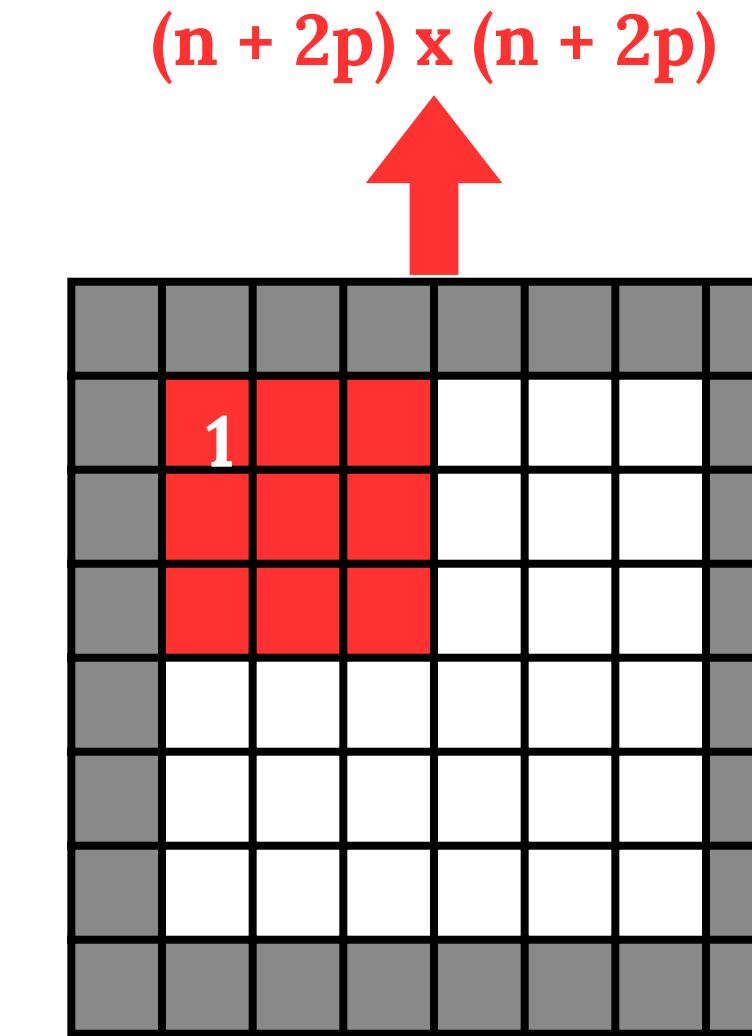


Padding

$n \times n$						
4	7	8	1	2	3	
4	3	1	3	5	1	
2	5	1	3	2	7	
8	0	2	4	7	2	
2	4	2	3	1	1	
5	7	5	4	2	3	

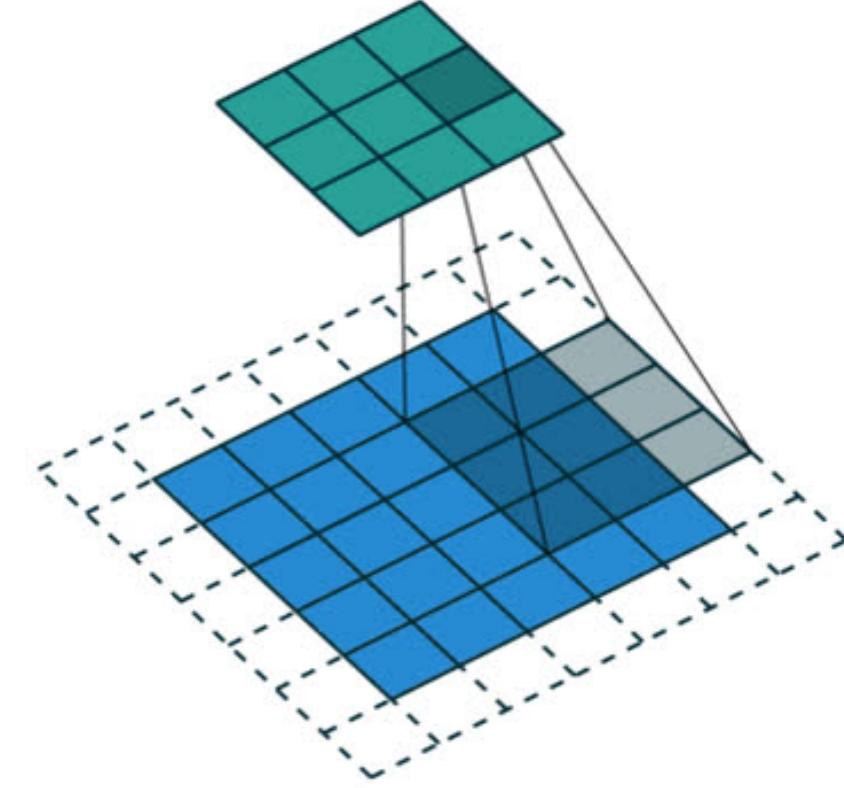
$$\begin{matrix} * & \begin{matrix} f \times f \\ 3 \times 3 \end{matrix} & \begin{matrix} f \times f \\ 3 \times 3 \end{matrix} \\ & \downarrow & \downarrow \end{matrix}$$

6×6 $n - f + 1 \times n - f + 1$

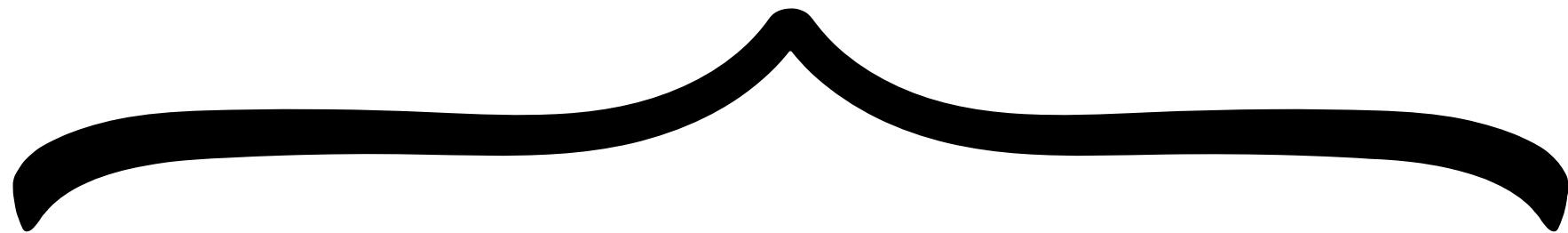


$$\begin{matrix} * & \begin{matrix} f \times f \\ 3 \times 3 \end{matrix} & \begin{matrix} f \times f \\ 3 \times 3 \end{matrix} \\ & \downarrow & \downarrow \end{matrix}$$

$(n + 2p - f + 1) \times (n + 2p - f + 1)$



Types de Padding



Valid

Ce type de padding n'ajoute pas de zéros autour de l'image d'entrée. La convolution est effectuée uniquement sur les parties valides de l'image, ce qui réduit les dimensions spatiales de la sortie.

Same

Ce type de padding ajoute des zéros autour de l'image d'entrée de manière à ce que la sortie de la couche de convolution ait la même taille spatiale (largeur et hauteur) que l'entrée.

Padding_Same

$$(n + 2p) \times (n + 2p) * f \times f \rightarrow n \times n$$

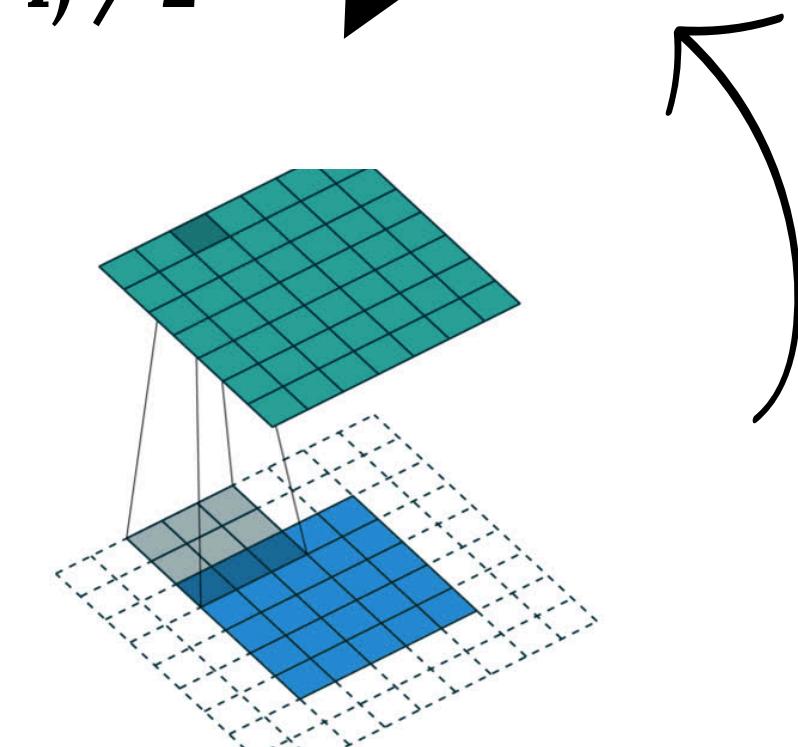
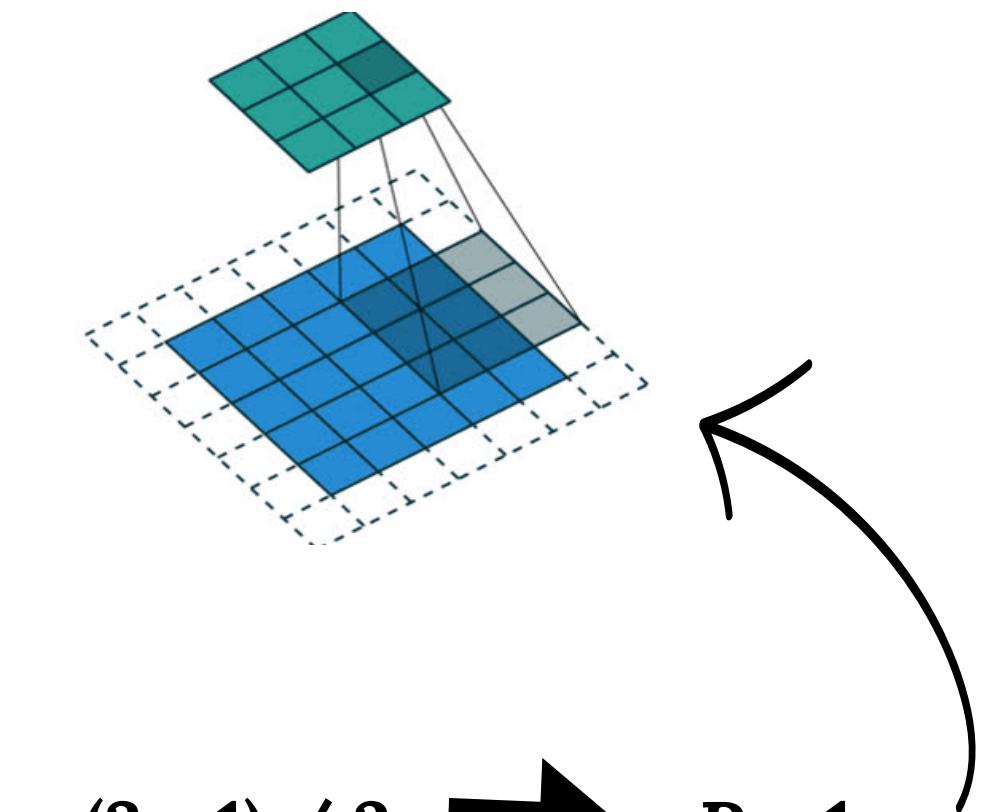
$$(n + 2p - f + 1) \times (n + 2p - f + 1)$$

n

$$n + 2p - f + 1 = n \quad \rightarrow \quad p = (f - 1) / 2$$

3 x 3 filtre → $p = (3 - 1) / 2$ → **P = 1**

5 x 5 filtre → $p = (5 - 1) / 2$ → **P = 2**



Stride

```
x = Conv2D(96,(11,11),padding='valid',activation='relu', strides=(4,4), name='block1_conv1')(img_input)
```

```
x = MaxPool2D((2,2),strides=(2,2),padding='valid', name='block2_pool')(x)
```

1	1	1	0	0
0	1x1	1x0	1x1	0
0	0x0	1x1	1x0	1
0	0x1	1x0	1x1	0
0	1	1	0	0

4	3	4
2	4	

- Le stride détermine de combien de pixels le noyau se déplace horizontalement et verticalement après chaque application de la convolution.
- Par défaut, la valeur du stride est généralement 1, ce qui signifie que le noyau de convolution se déplace d'un pixel à chaque étape.

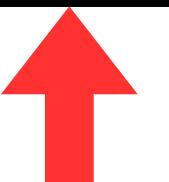
Exemple de stride

4	7	5	1	2	3	3
4	7	5	1	2	3	3
4	3	1	3	5	1	1
2	5	1	3	2	7	7
8	0	2	4	7	2	2
2	4	2	3	1	1	1
5	7	5	4	2	3	3

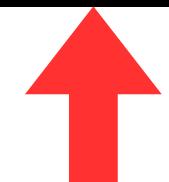
n x n

7 x 7

S = 2



p = 0



```
x = MaxPool2D((2,2),strides=(2,2),padding='valid', name='block2_pool')(x)
```

*

1	0	-1
1	0	-1
1	0	-1

=

3 x 3

f x f
3 x 3

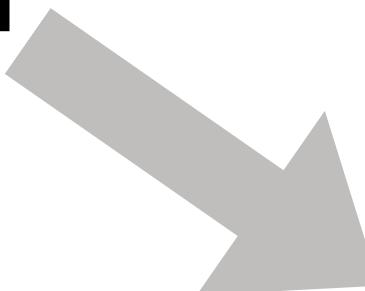
$$\frac{n + 2p - f}{S} + 1 = 3$$

$$\frac{n + 2p - f}{S} + 1 = 3$$

Fonction d'activation

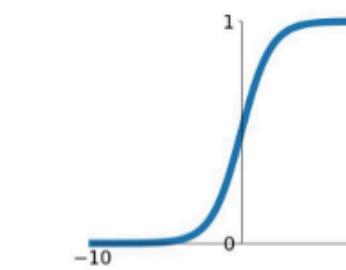
Après l'opération de convolution a extrait des caractéristiques importantes de l'image, il est essentiel de sélectionner et d'introduire de la non-linéarité dans le réseau. Cela se fait à l'aide d'une **fonction d'activation**.

En CNN



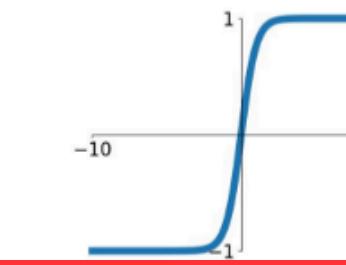
Sigmoid

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



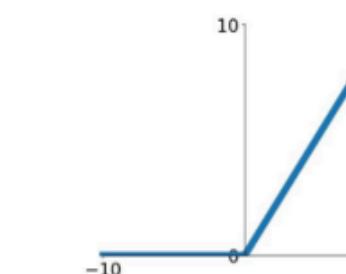
tanh

$$\tanh(x)$$



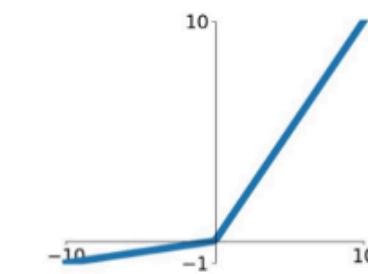
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

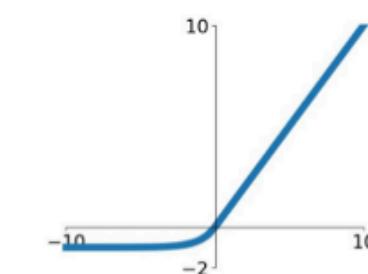


Maxout

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

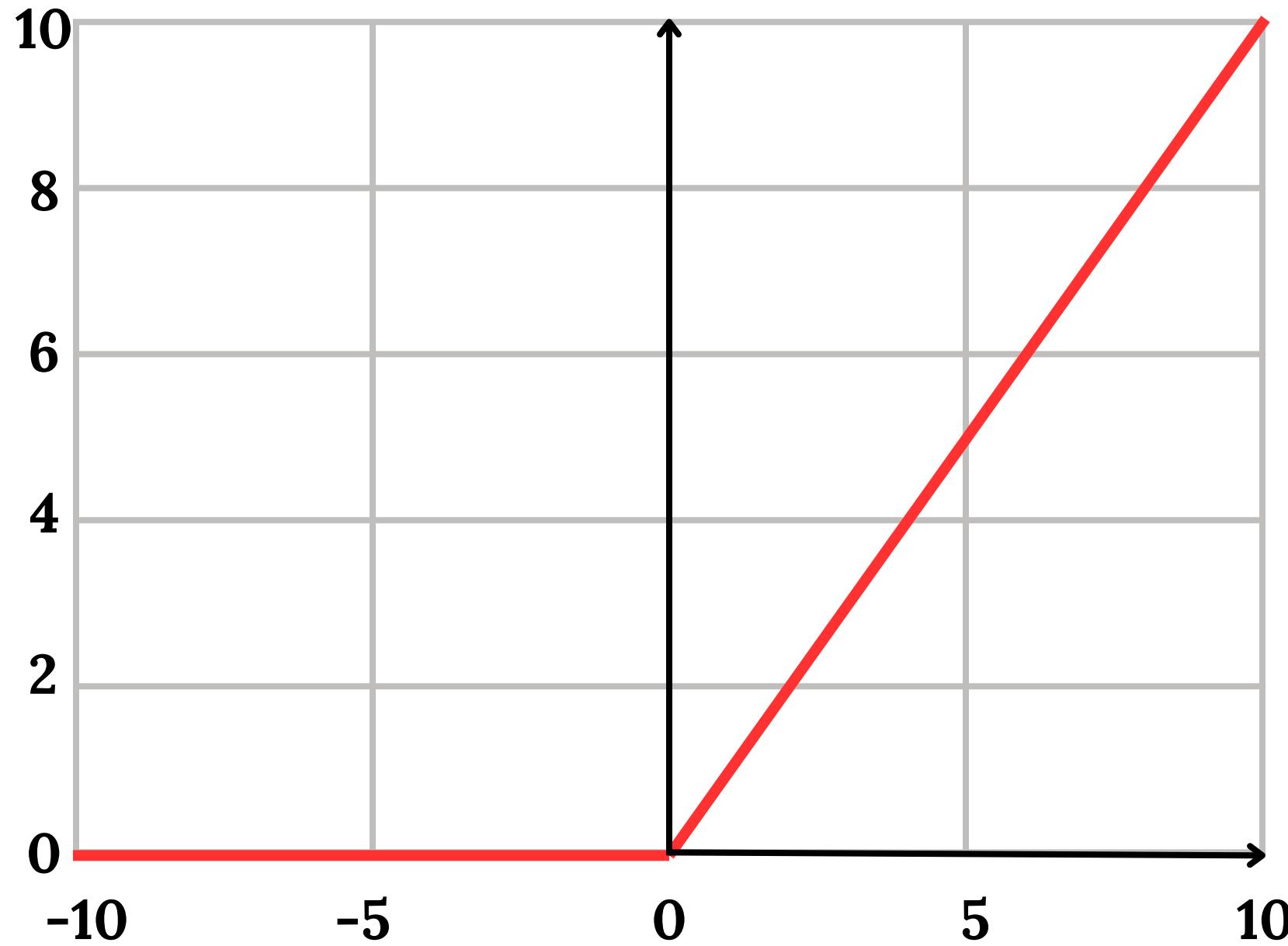
ELU

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



ReLU

$$R(z) = \max(0, z)$$



Exemple :

9	3	5	-8
-6	2	-3	1
1	3	4	1
3	-4	5	1



9	3	5	8
6	2	3	1
1	3	4	1
3	4	5	1

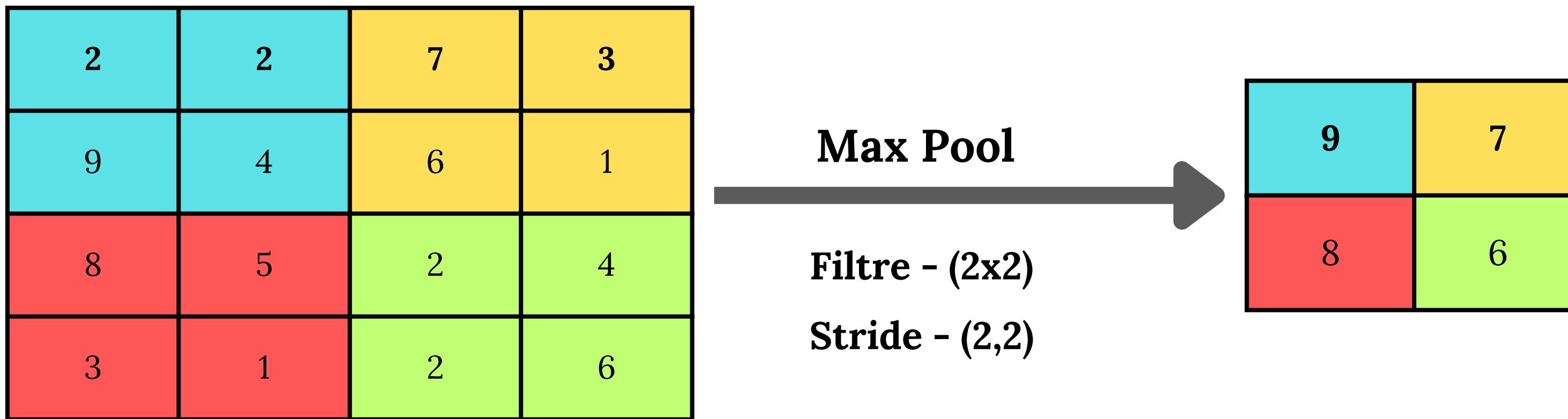
Couche ReLU

ReLU filtre les informations en éliminant les valeurs négatives et en conservant uniquement les caractéristiques pertinentes pour la tâche d'apprentissage.

Couche de Pooling

Les couches de pooling réduisent la taille des données pour simplifier le réseau tout en gardant les informations importantes. Elles éliminent les détails inutiles et rendent le modèle plus efficace. Les principaux types sont le **Max Pooling**, **l'Average Pooling** et le **Sum Pooling**.

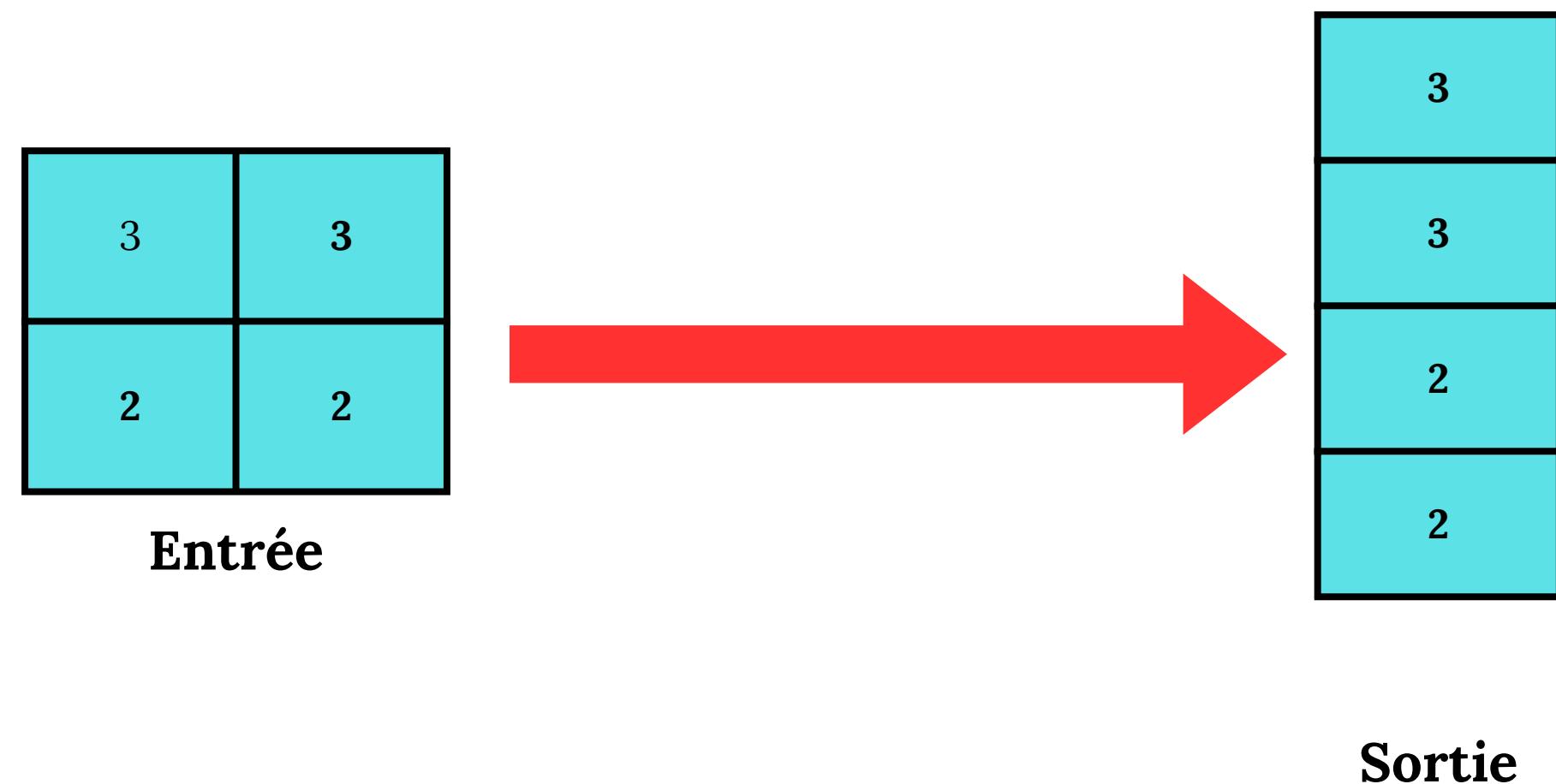
Max Pooling :



Couche Flatten (Aplatissement)

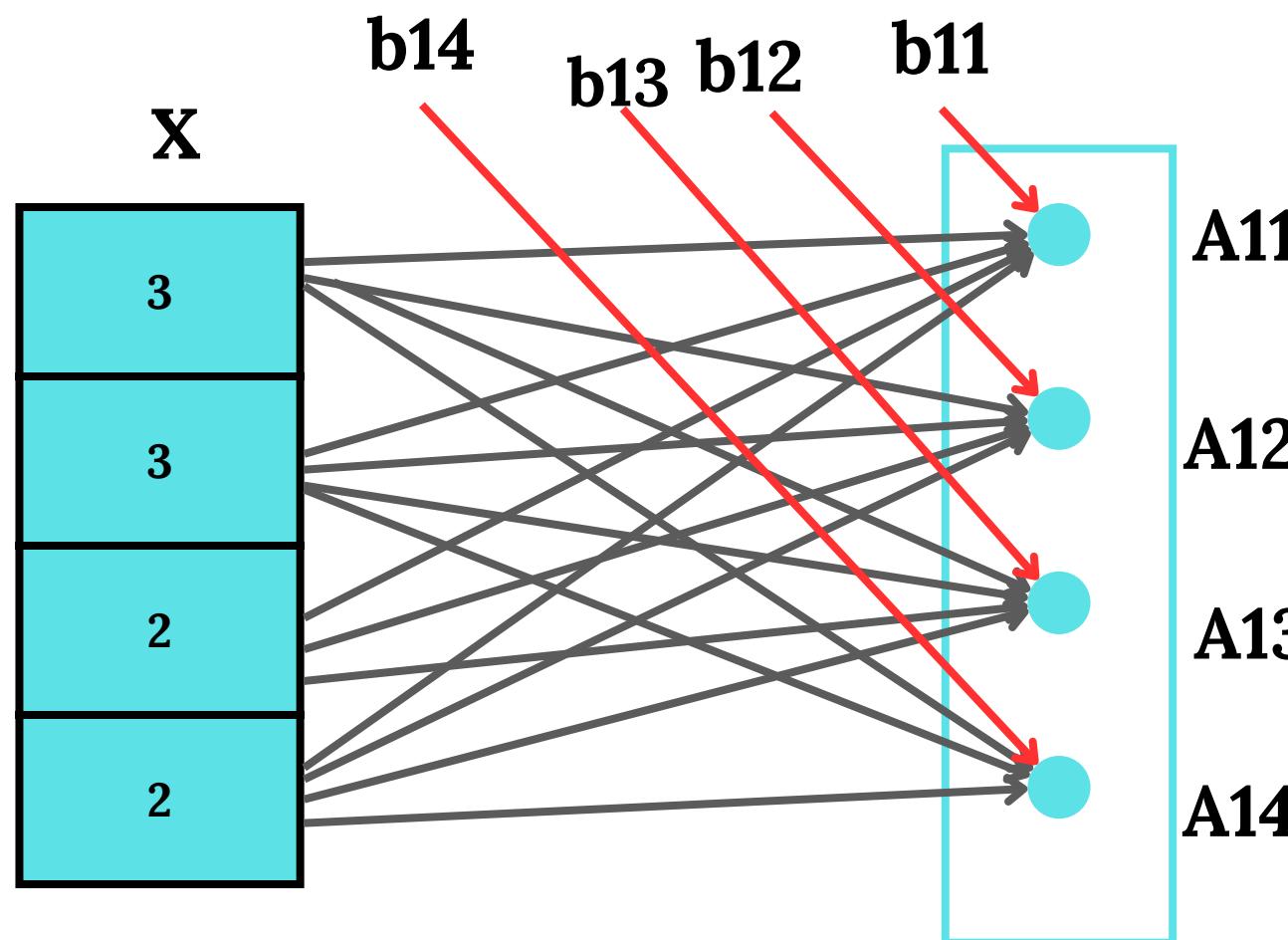
La matrice des caractéristiques sera convertie en un vecteur (x_1, x_2, x_3, \dots).

Exemple :



Couche fully-connected

Une couche fully-connected (complètement connectée) est une couche d'un réseau de neurones convolutifs où chaque neurone est connecté à tous les neurones de la couche précédente. Elle combine les entrées par une multiplication matricielle avec des poids et ajoute un biais, puis applique une fonction d'activation, comme ReLU.



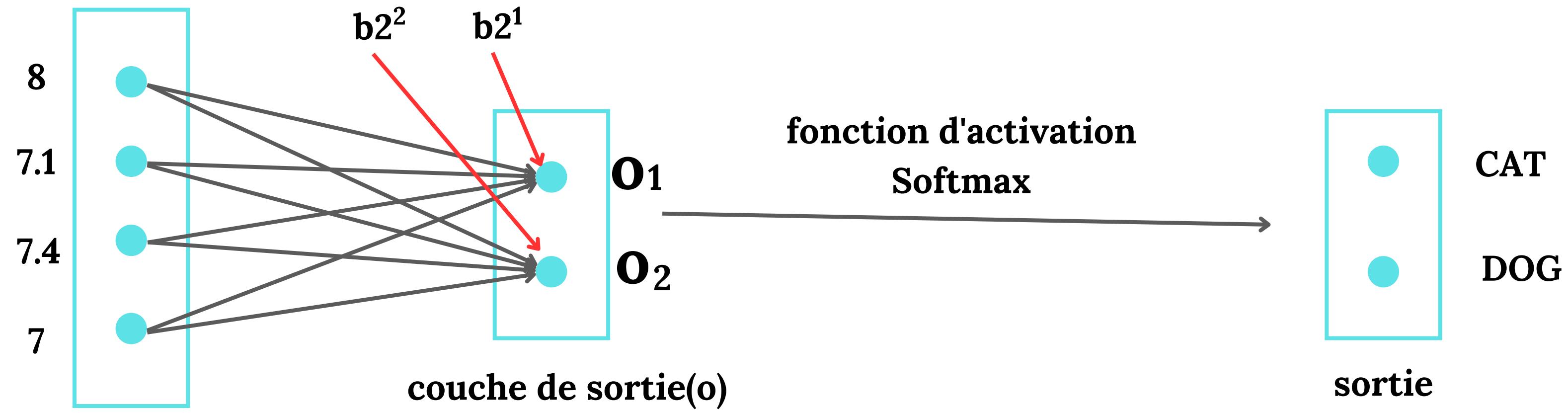
$$A_1 = W_1 X + b_1$$

$$\begin{bmatrix} A_{11} \\ A_{12} \\ A_{13} \\ A_{14} \end{bmatrix} = \begin{bmatrix} 1.0 & 1.0 & 0.2 & 0.8 \\ 1.0 & 0.5 & 0.5 & 0.8 \\ 0.8 & 1.0 & 0.2 & 0.8 \\ 0.5 & 0.5 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 8 \\ 7.1 \\ 7.4 \\ 7 \end{bmatrix}$$

$$\text{ReLU} \begin{bmatrix} 8 \\ 7.1 \\ 7.4 \\ 7 \end{bmatrix} = \begin{bmatrix} 8 \\ 7.1 \\ 7.4 \\ 7 \end{bmatrix}$$

couche fully-connected & La fonction d'activation Softmax

La couche fully-connected combine les sorties des couches précédentes avec des poids et des biais, produisant des scores bruts (O_1 et O_2). La fonction d'activation Softmax convertit ces scores en probabilités normalisées pour classer les données en différentes catégories, comme "CAT" ou "DOG".



La fonction d'activation Softmax

$$O = W_2 A_1 + b_2$$

$$\begin{bmatrix} O_1 \\ O_2 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.1 & 0.5 & 0.5 \\ 0.1 & 0.5 & 0.1 & 0.2 \end{bmatrix} \begin{bmatrix} 8 \\ 7.1 \\ 7.4 \\ 7 \end{bmatrix}$$

$$+ \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 9.61 \\ 6.49 \end{bmatrix}$$

$$\sigma(O_i) = \frac{e^{O_i}}{\sum e^{O_i}}$$

$$\sigma(O_1) = \frac{e^{O_1}}{e^{O_1} + e^{O_2}}$$

$$= \frac{14913.17}{14913.17 + 658.52} = 0.95$$

$$\sigma(O_2) = \frac{e^{O_2}}{e^{O_1} + e^{O_2}} = \frac{658.52}{14913.17 + 658.52}$$

$$= 0.04$$

Exemple pratique des réseaux de neurones artificiels

Outils utilisés



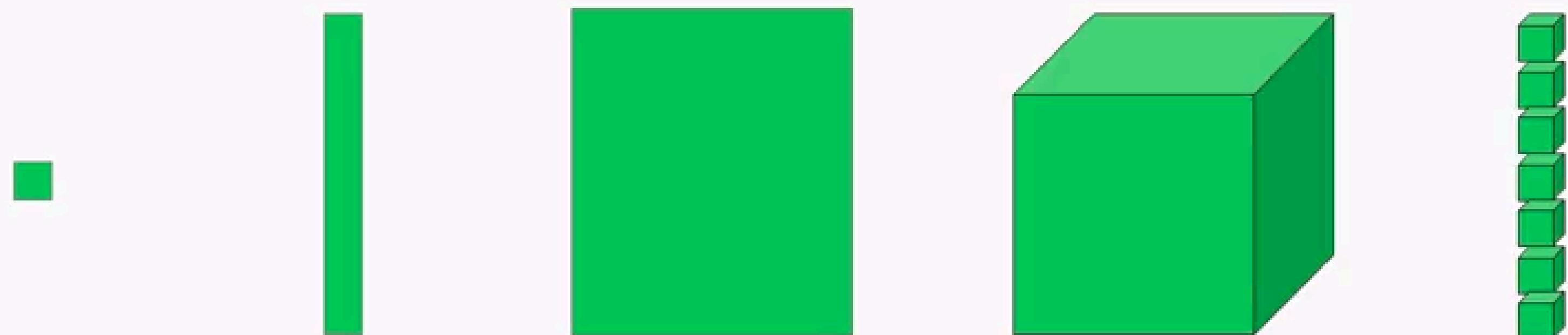
TensorFlow est une bibliothèque open source développée par Google, spécialement conçue pour le machine learning et le deep learning. Elle permet de créer, d'entraîner et de déployer des modèles de manière efficace.



Keras est une API de haut niveau pour TensorFlow, conçue pour simplifier la création et l'entraînement de modèles de deep learning

Notion de “Tenseur”

Un tenseur est un table de données à N dimensions



Rank 0
Tensor
scalar

Rank 1
Tensor
vector

Rank 2
Tensor
matrix

Rank 3
Tensor

Rank 4
Tensor