

Deep Learning

Apprentissage Profond



Dr. Sana Hamdi

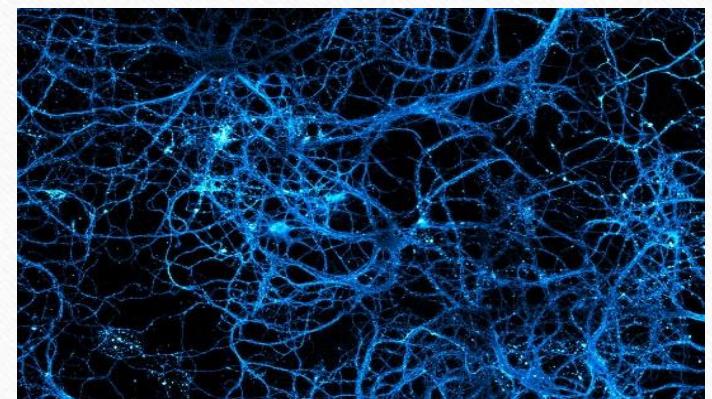
sana.hamdi@fst.utm.tn

Introduction



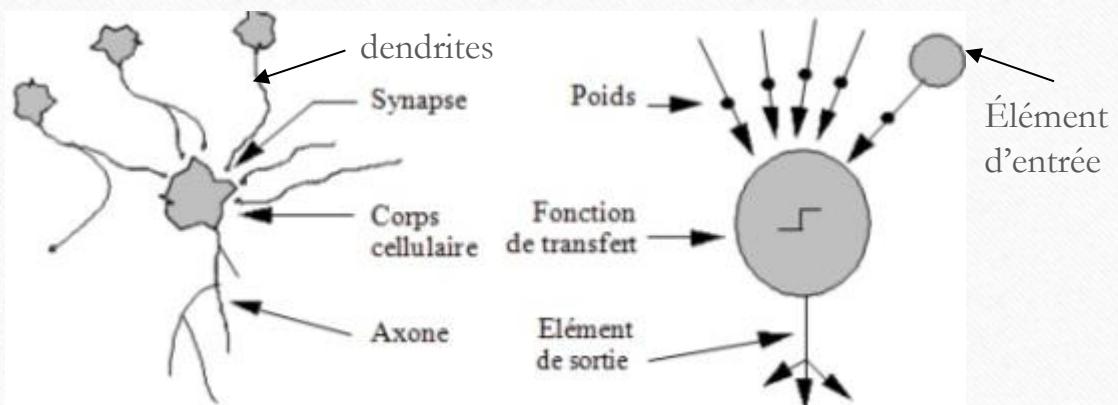
Introduction

- Une approche inspirée du traitement de l'information effectué par le cerveau
- Applications:
 - ✓ statistiques : analyse de données / prévision / classification
 - ✓ robotique : contrôle et guidage de robots ou de véhicules autonomes
 - ✓ imagerie / reconnaissance de formes (images ou signaux)
 - ✓ traduction automatique,...



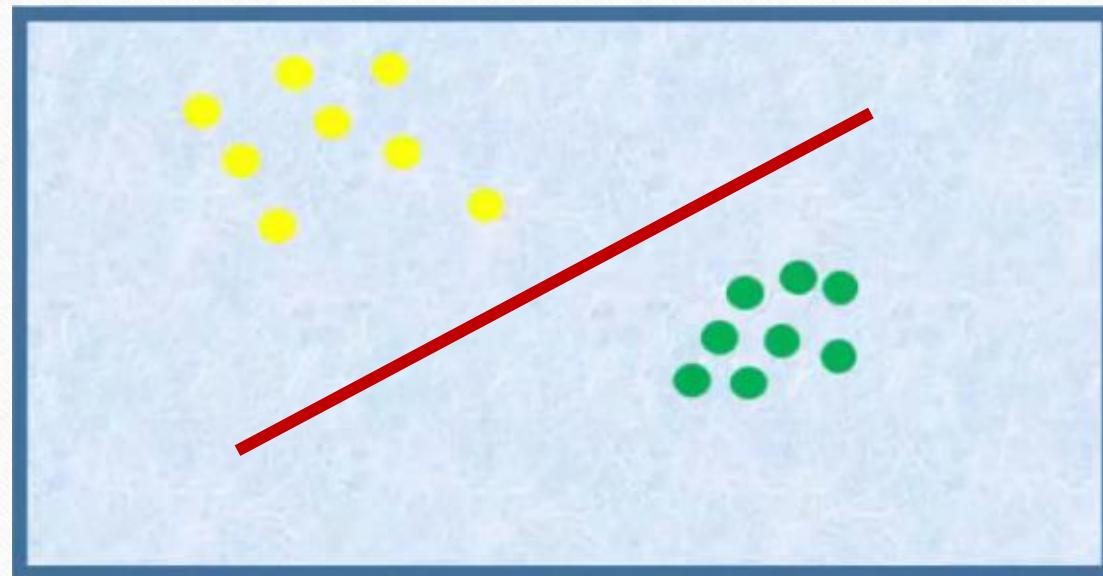
Modèle inspiré du modèle biologique

- Nous disposons d'une dizaine de milliards de neurones à l'intérieur de notre cerveau
- Les neurones reçoivent des signaux (impulsions électriques) par les **dendrites** et envoient l'information par les **axones**.
- Les contacts entre deux neurones (entre axone et dendrite) se font par l'intermédiaire des **synapses**.



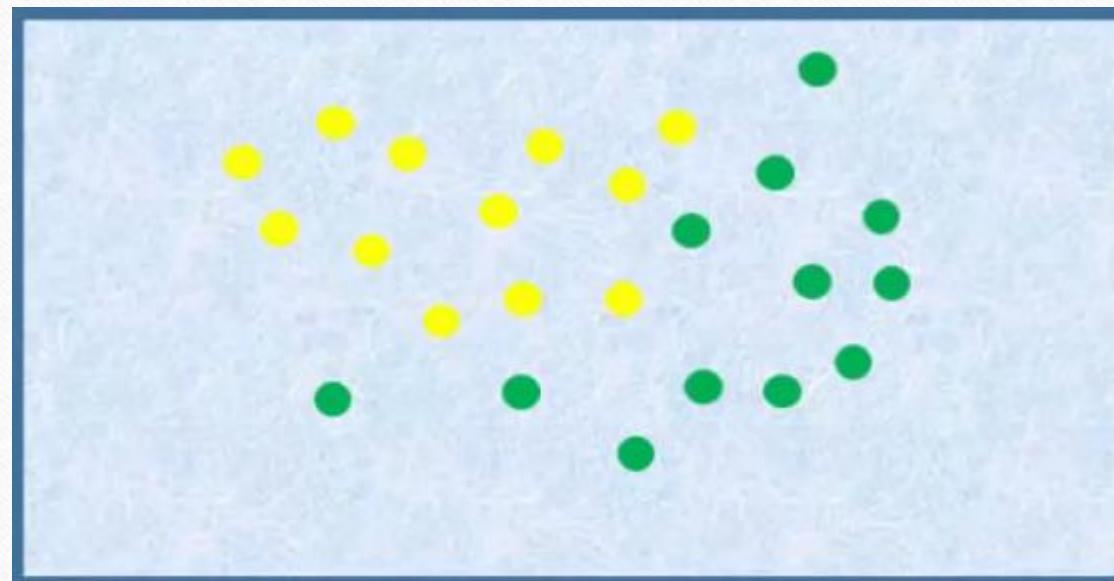
Le réseau de neurones et la classification

Classificateur linéaire



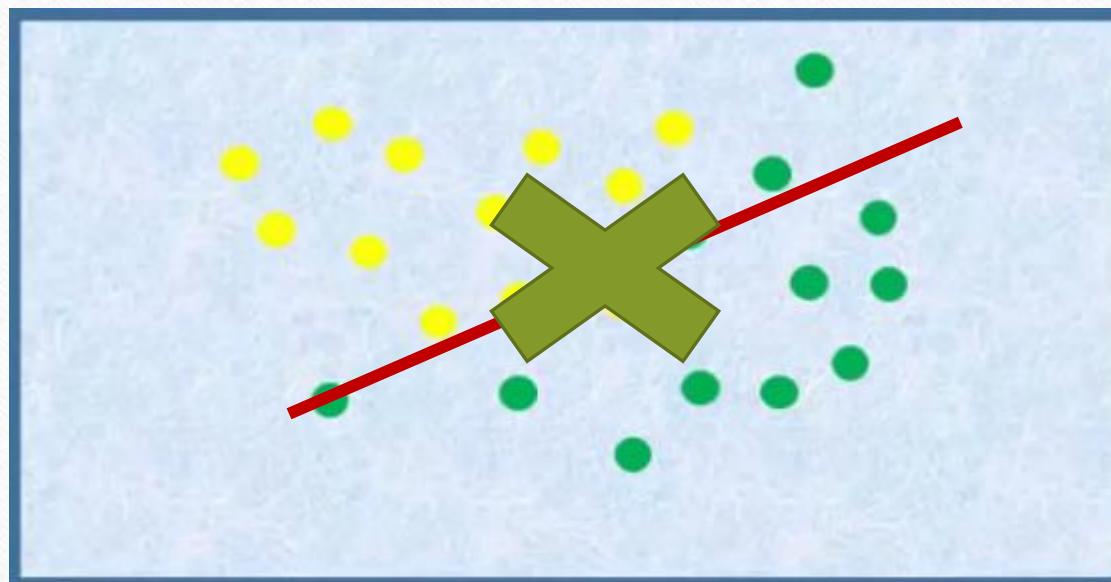
Le réseau de neurones et la classification

Classificateur linéaire et données complexes



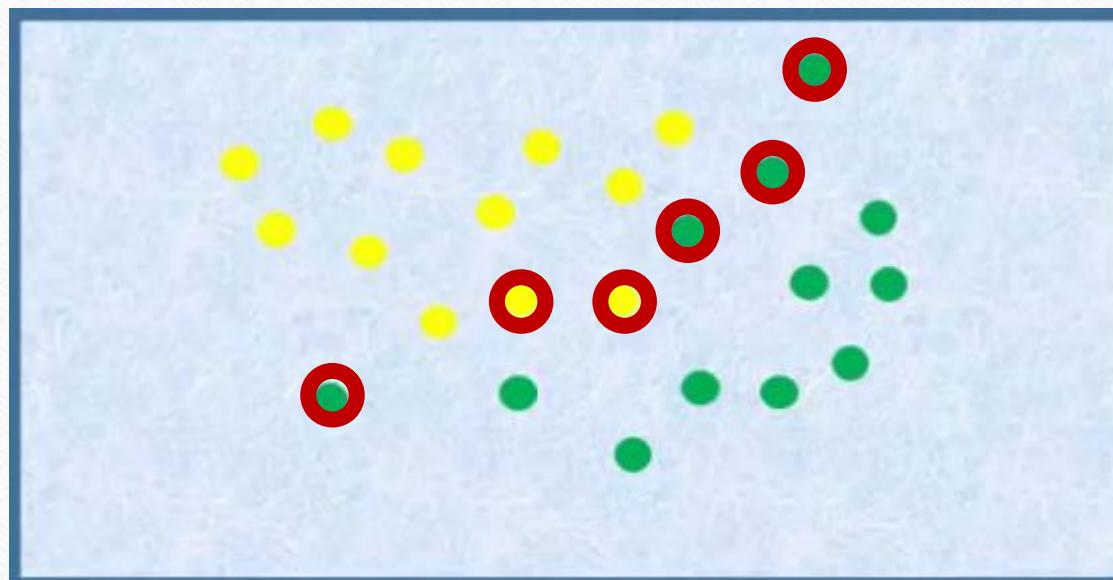
Le réseau de neurones et la classification

Classificateur linéaire et données complexes



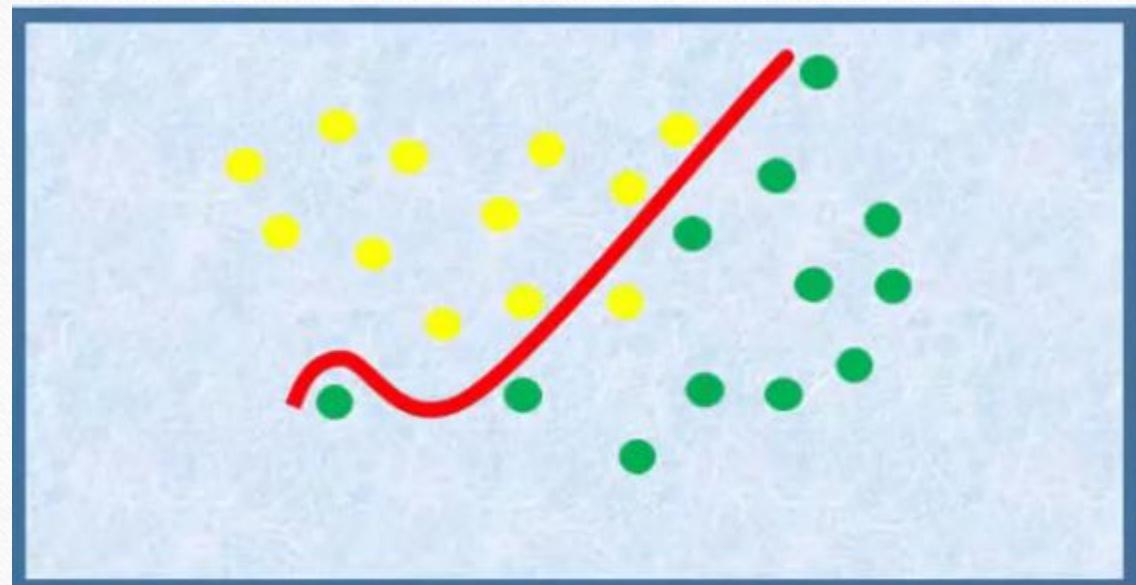
Le réseau de neurones et la classification

Non résolu linéairement



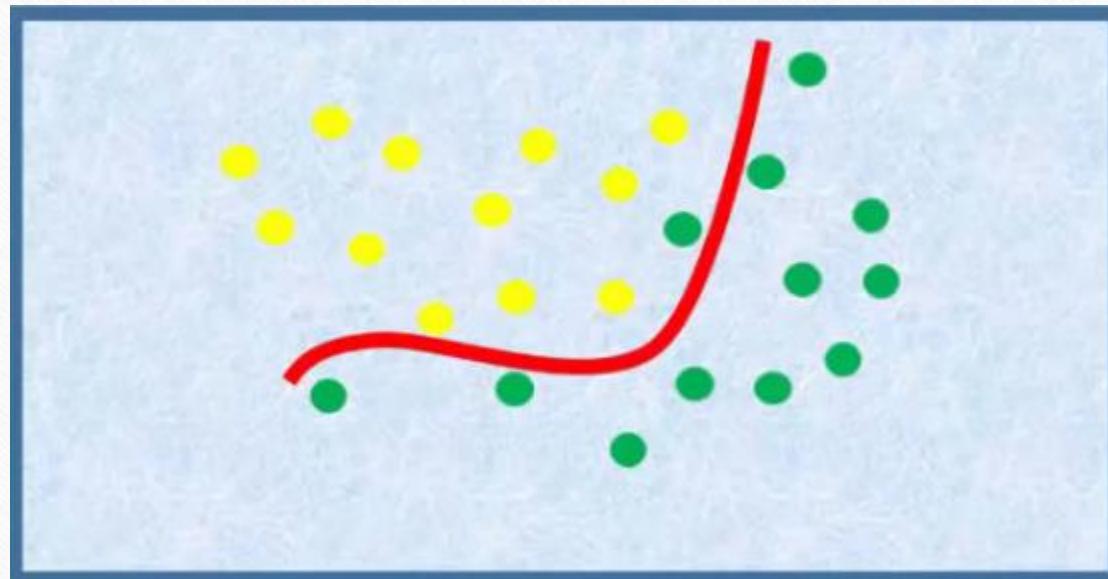
Le réseau de neurones et la classification

Classificateur non linéaire → limite de décision initiale → Apprentissage



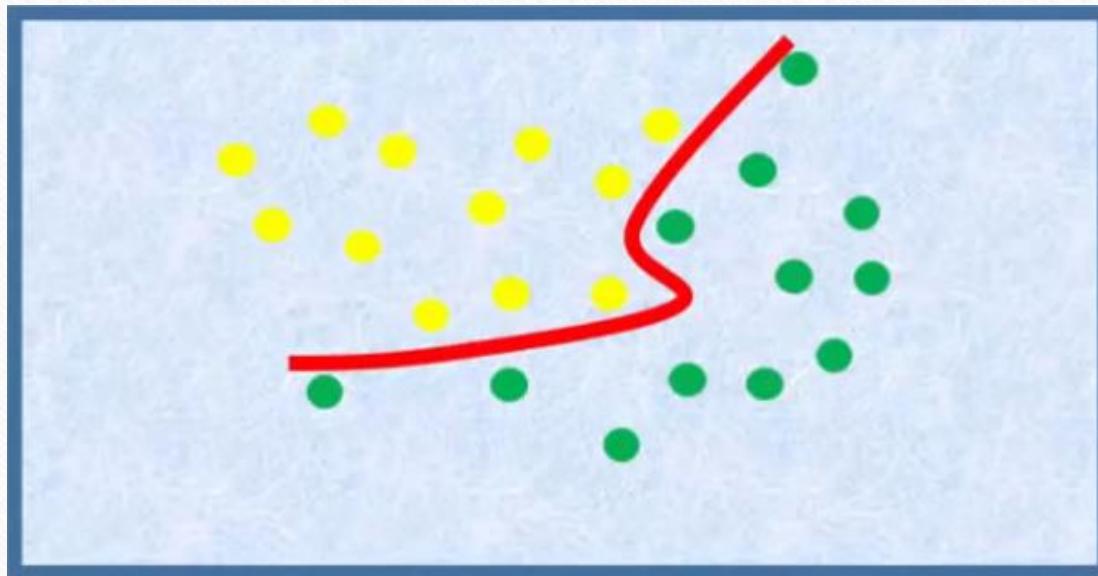
Le réseau de neurones et la classification

Classificateur non linéaire → limite de décision initiale → Apprentissage

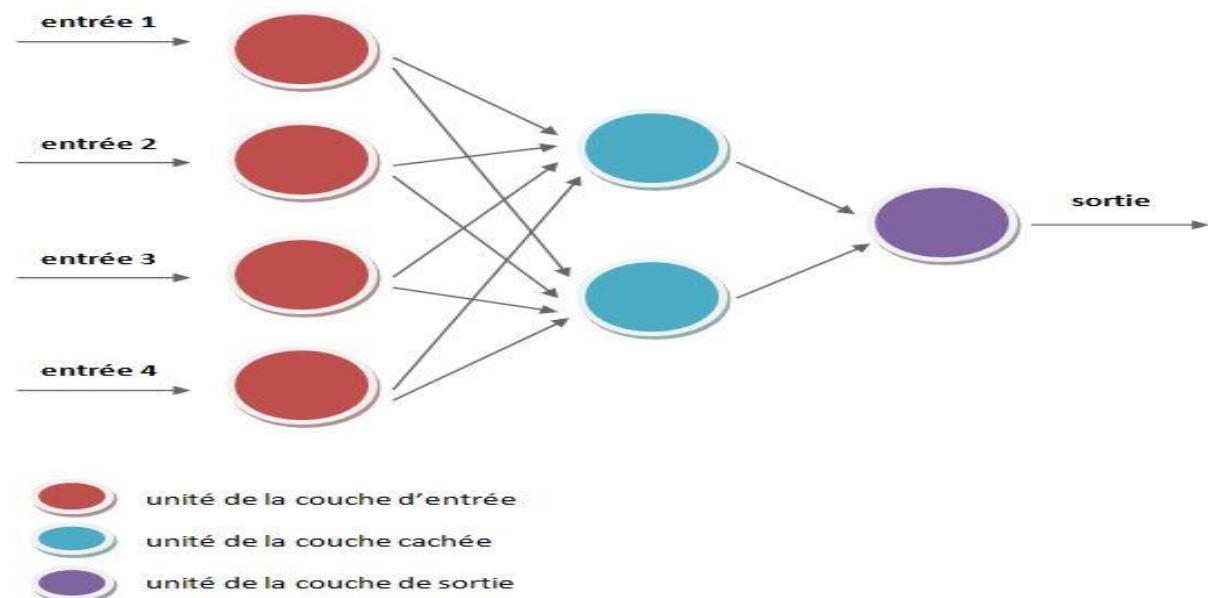


Le réseau de neurones et la classification

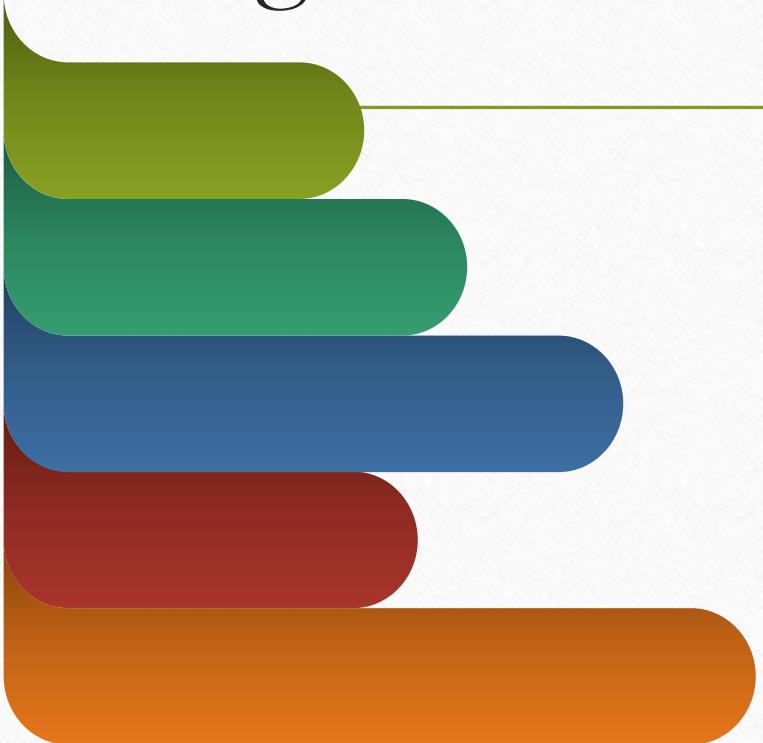
Classificateur non linéaire → limite de décision initiale → Apprentissage



Architecture du réseau de neurones



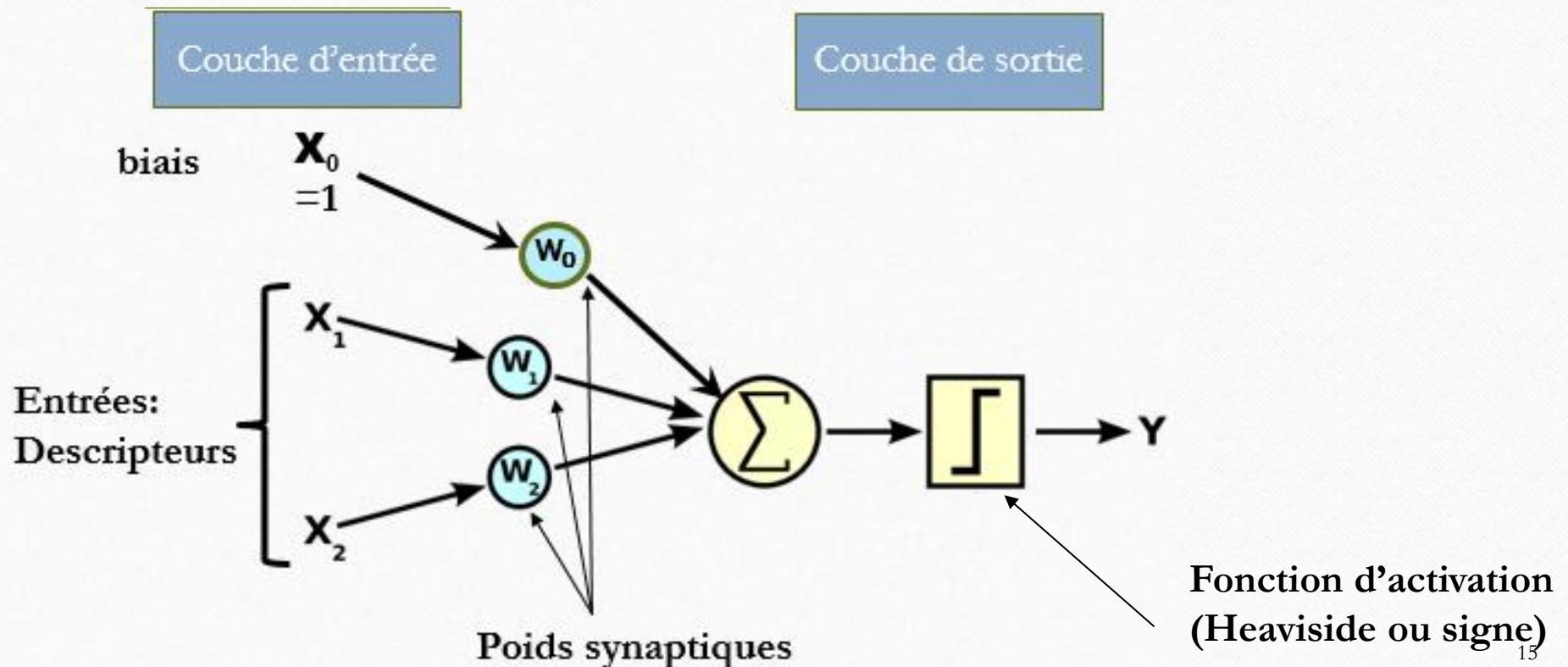
L'algorithme de perceptron



L'algorithme de perceptron

- Le perceptron est le modèle le plus simple de réseaux de neurones
- C'est un classificateur linéaire
- Le perceptron ne possède pas de couches cachées.
→ Une couche en entrée et une couche en sortie.

L'architecture de perceptron



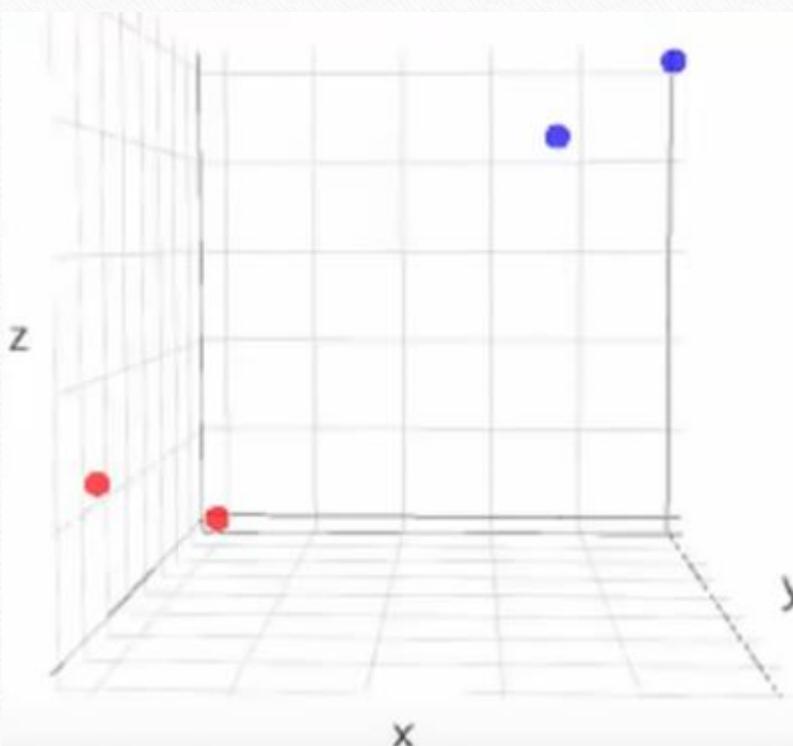
L'algorithme de perceptron

1. Mélanger aléatoirement les observations
2. Initialiser aléatoirement les poids synaptiques
3. Faire passer les observations une à une
 - Calculer l'erreur de prédiction pour l'observation
 - Mettre à jour les poids synaptiques
4. Jusqu'à convergence du processus

L'algorithme de perceptron

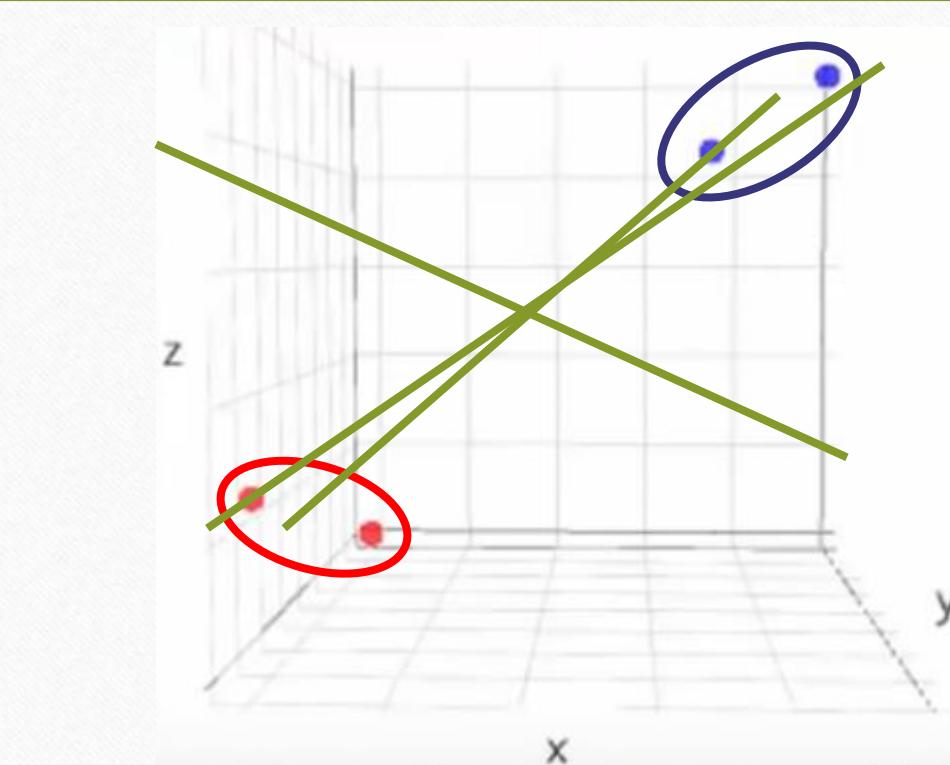
- L'algorithme d'apprentissage doit adapter la valeur des paramètres (c'est-à-dire les poids et le biais) de façon que la sortie prévue $f_W(X)$ soit la bonne réponse sur les données d'entraînement.
- **Algorithme du Perceptron:**
 1. Pour chaque paire $(X_t, Y_t) \in D$
 - a. Calculer $f_W(X_t)$
 - b. Si $Y_t \neq f_W(X_t)$ alors $W_i \leftarrow W_i + \eta(Y_t - f_W(X_t)) X_{t,i} \quad \forall i$ (mise à jour des poids et du biais)
 2. Retourner à 1 jusqu'à l'atteinte d'un critère d'arrêt (nb max d'itérations est atteint ou l'erreur = 0)
 - La mise à jour des poids est appelée la règle d'apprentissage du perceptron.
 - Le multiplicateur η est appelé le taux d'apprentissage.

Exemple de classification



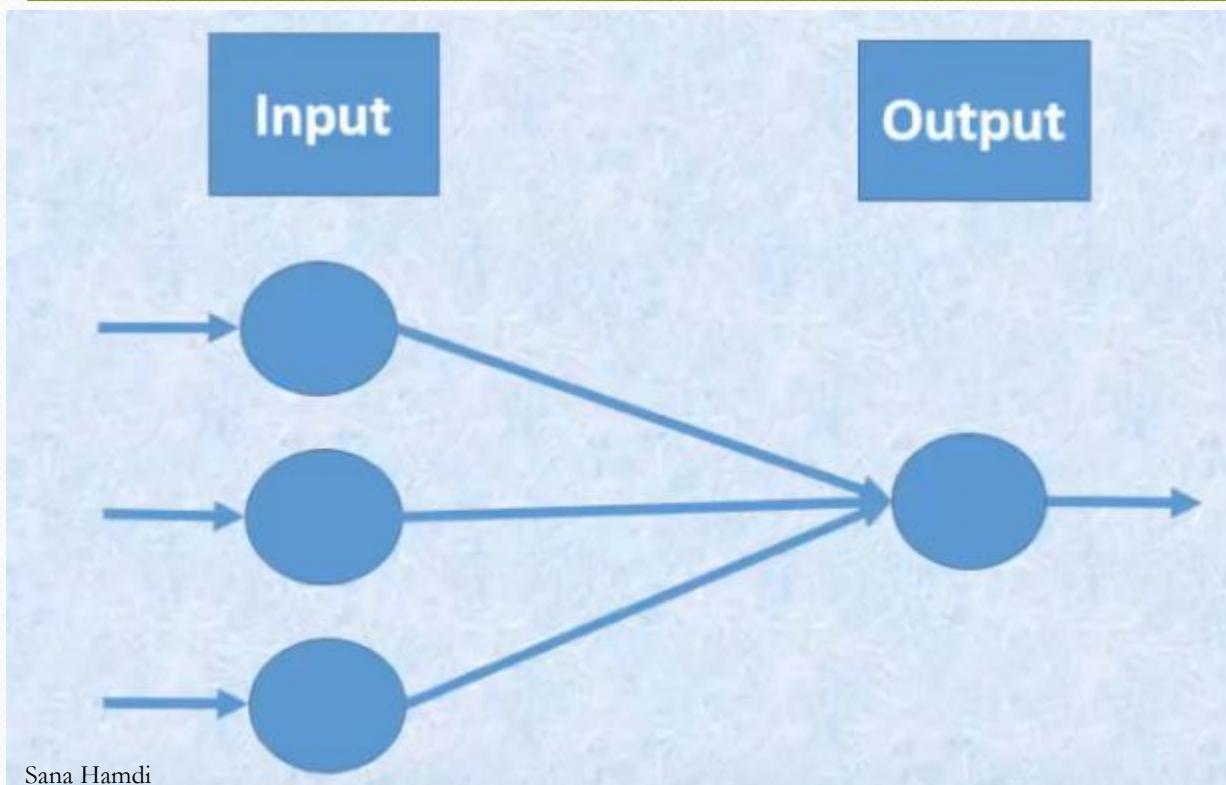
R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU

Exemple de classification

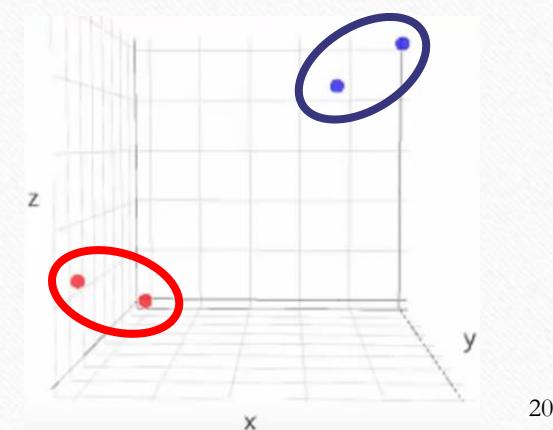


R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU

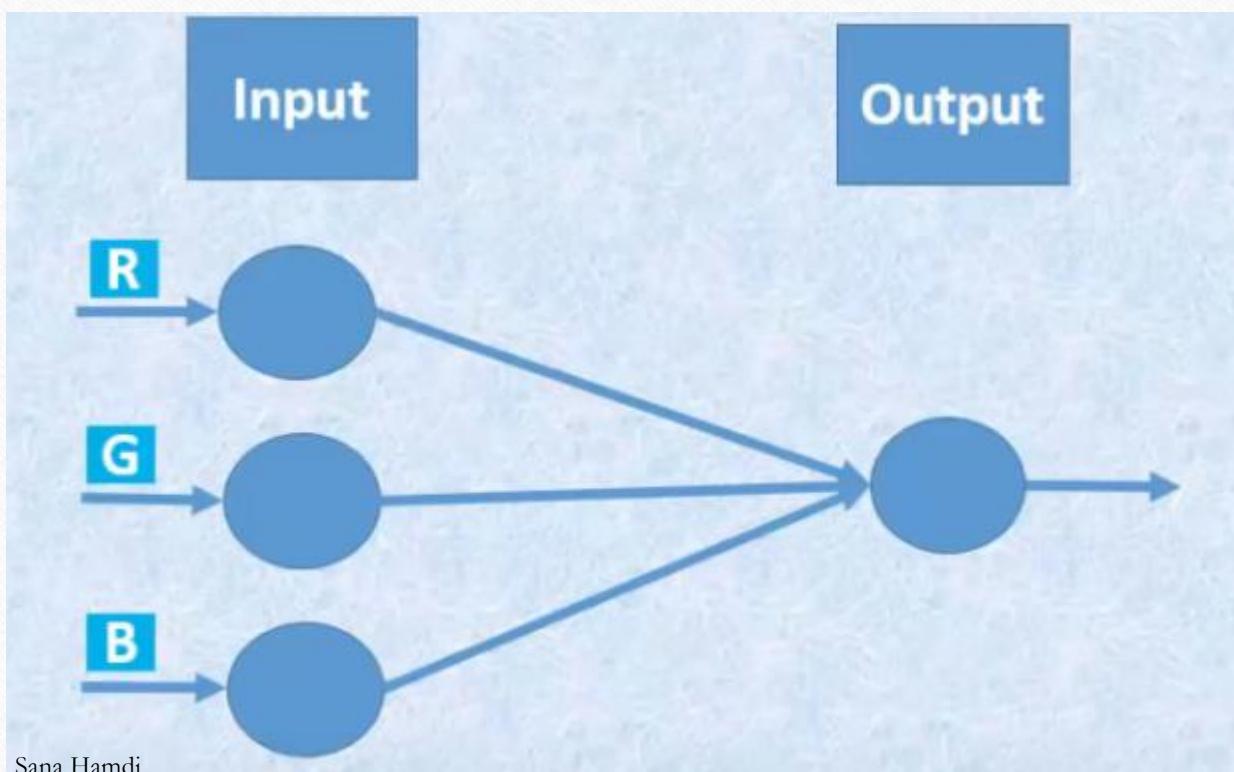
Exemple de classification



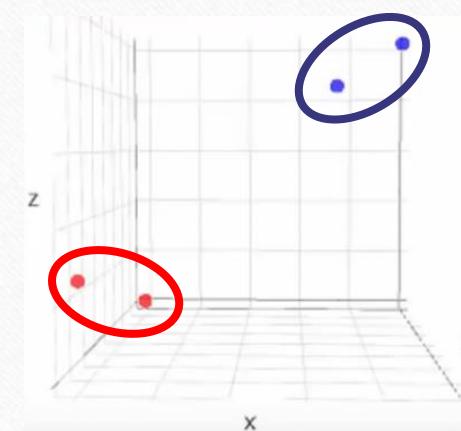
R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU



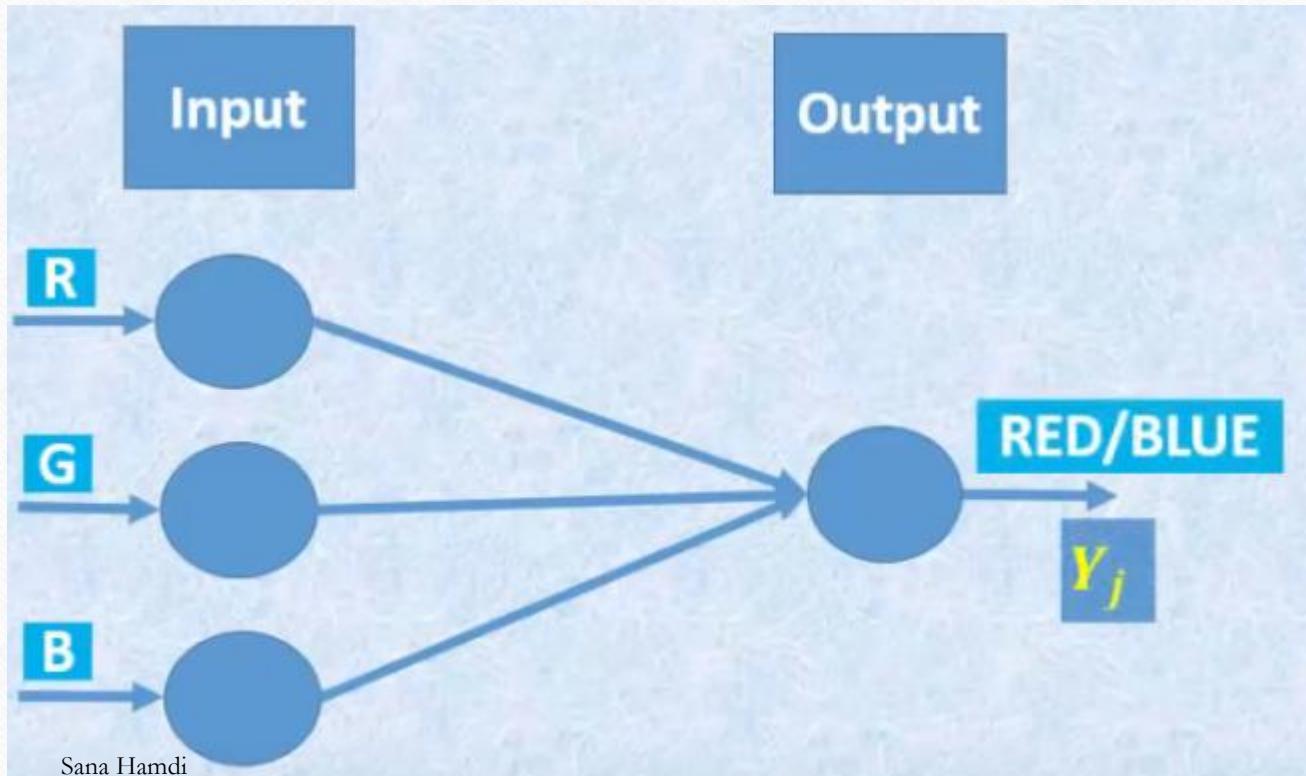
Couche d'entrée



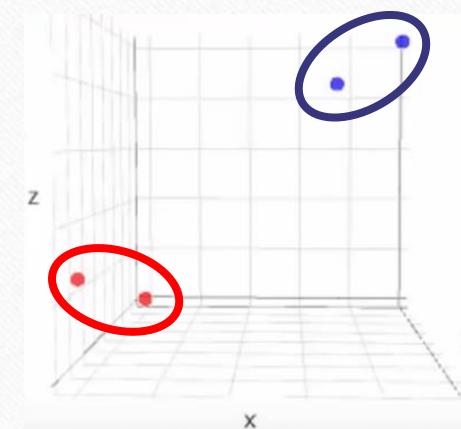
R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU



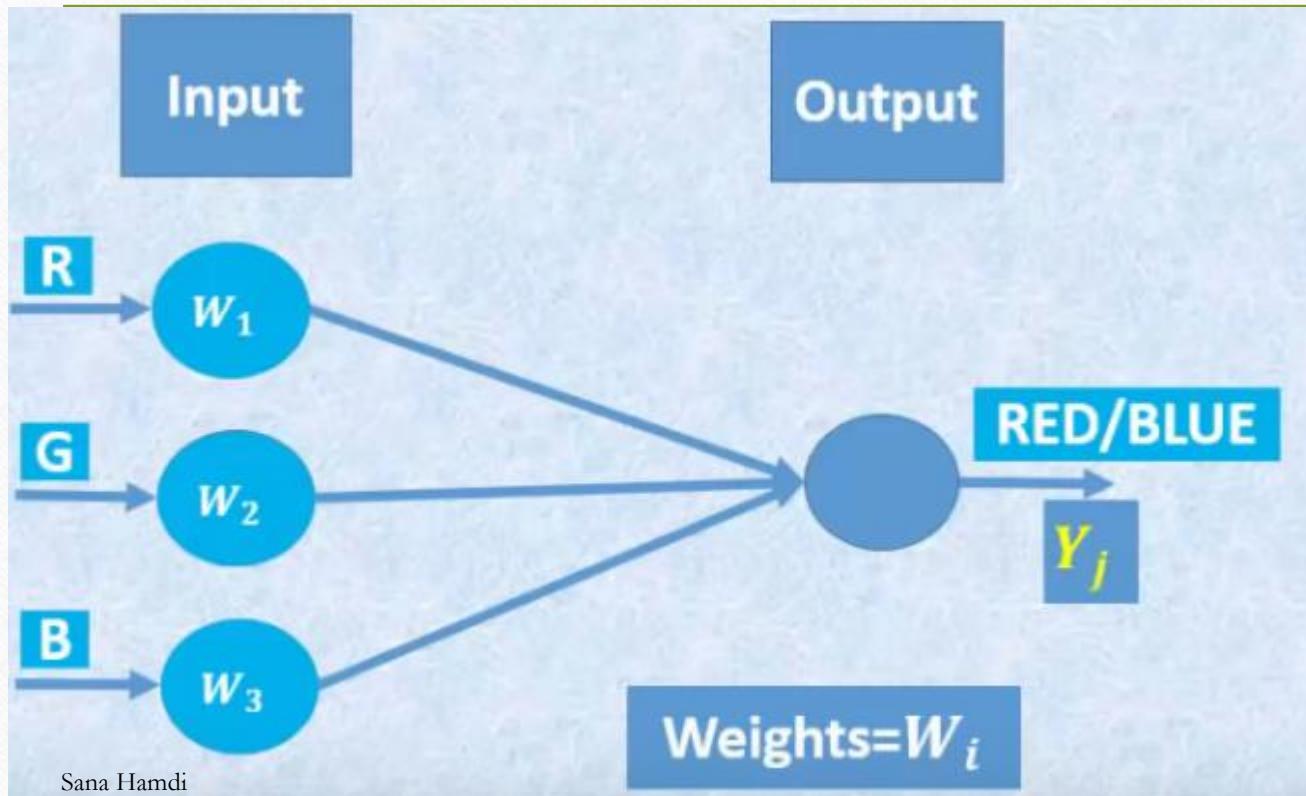
Couche de sortie



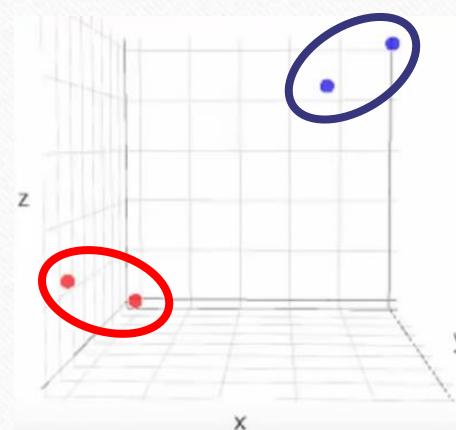
R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU



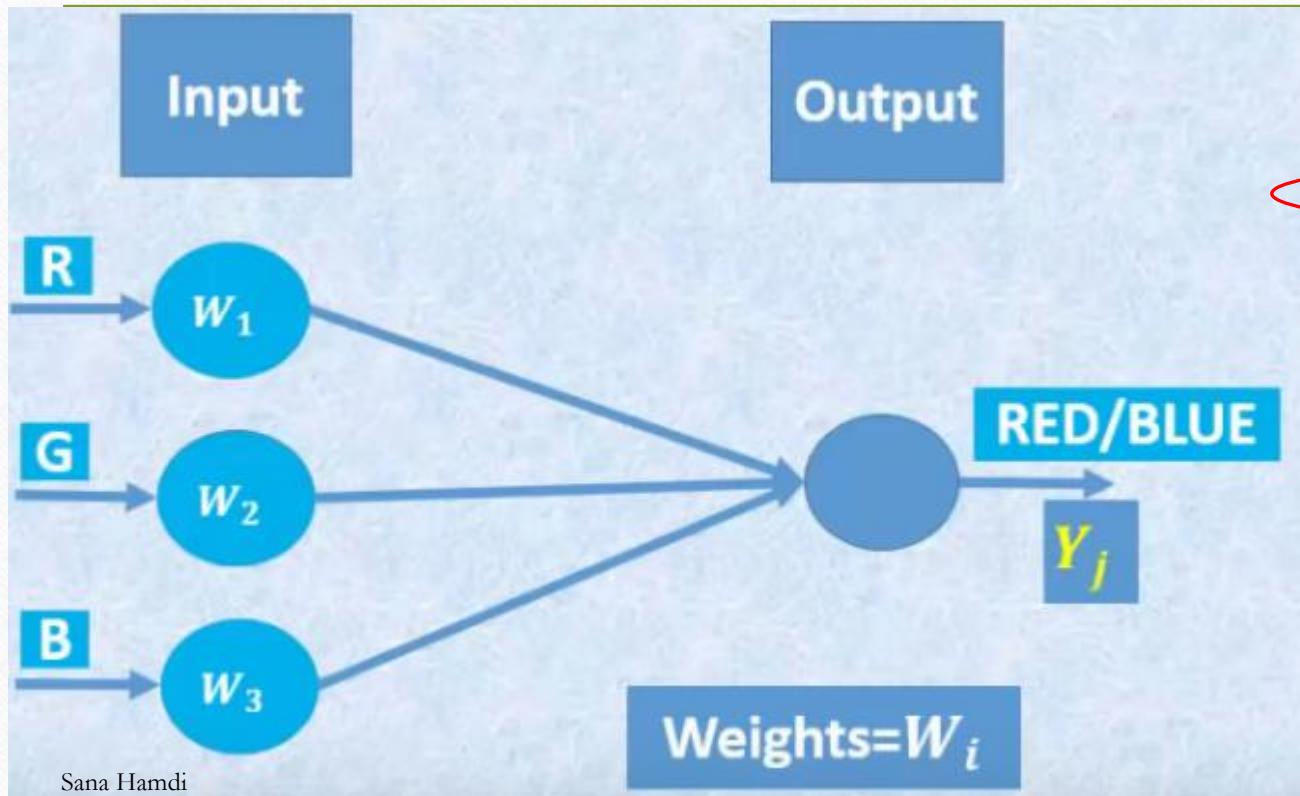
Les poids



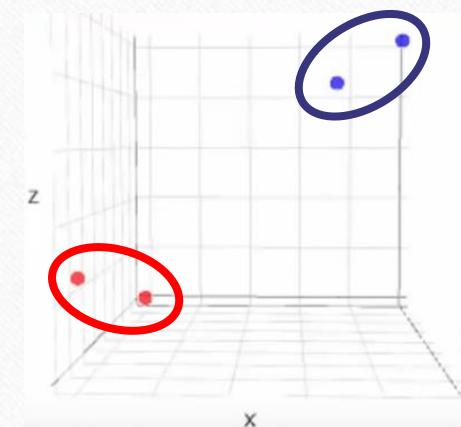
R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU



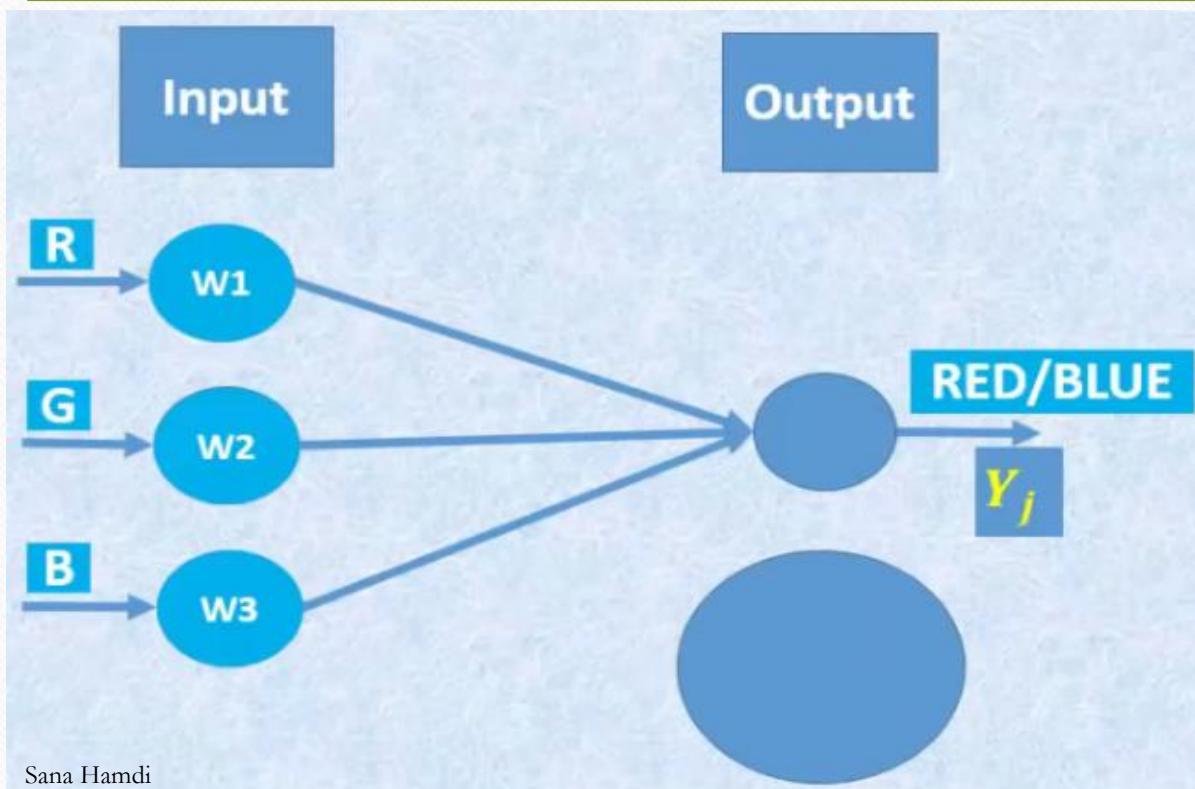
Les poids



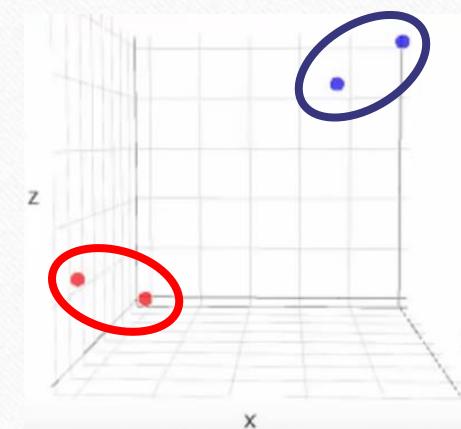
R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE ?
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU



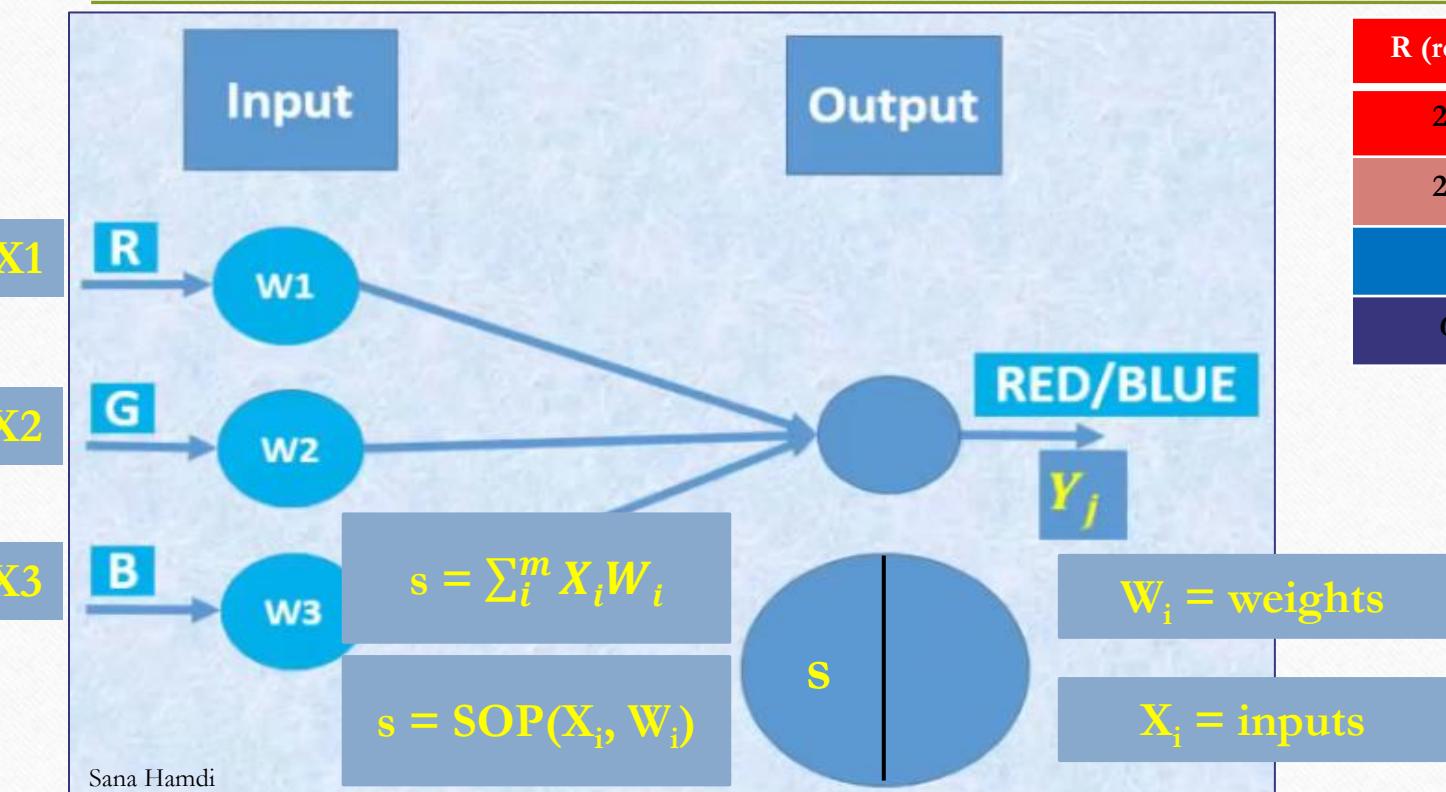
Composants du nœud de la sortie



R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE ?
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU

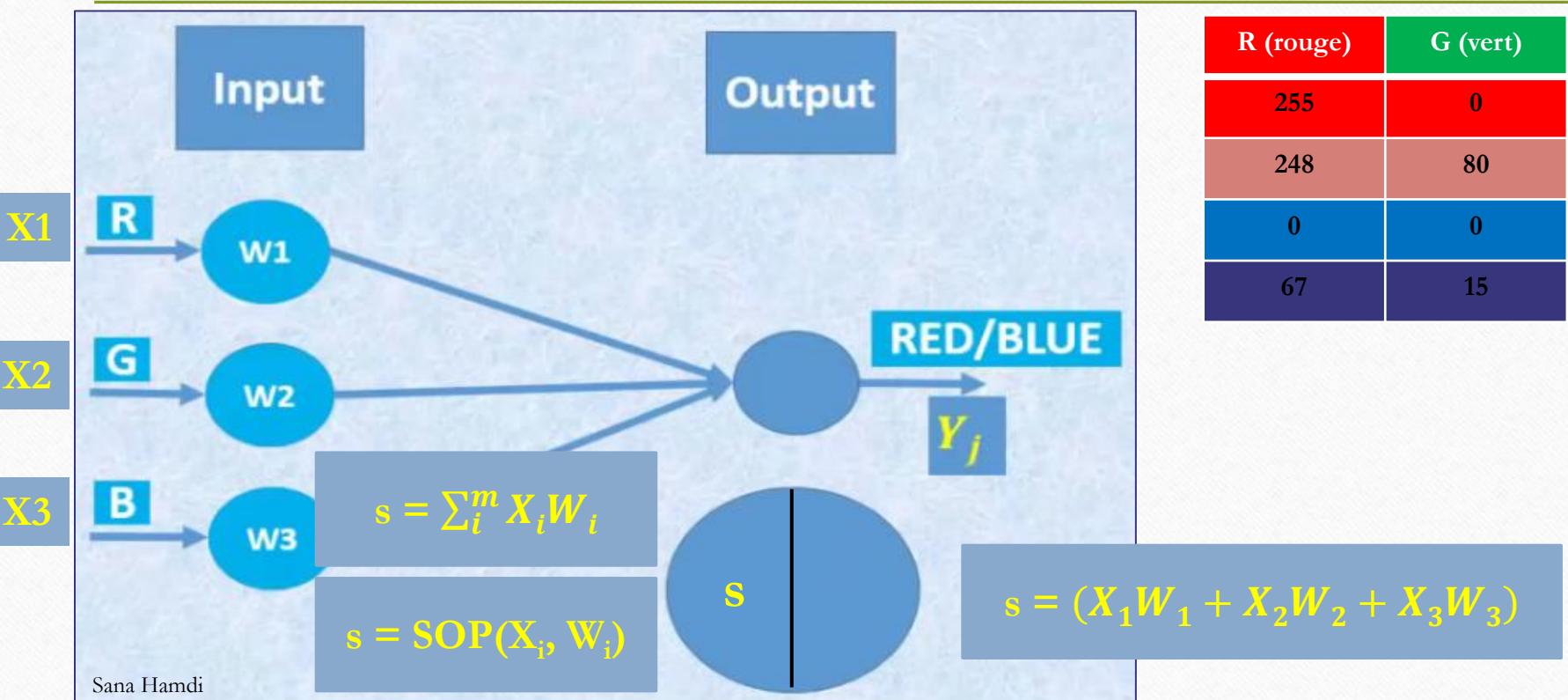


Composants du nœud de la sortie: somme pondérée



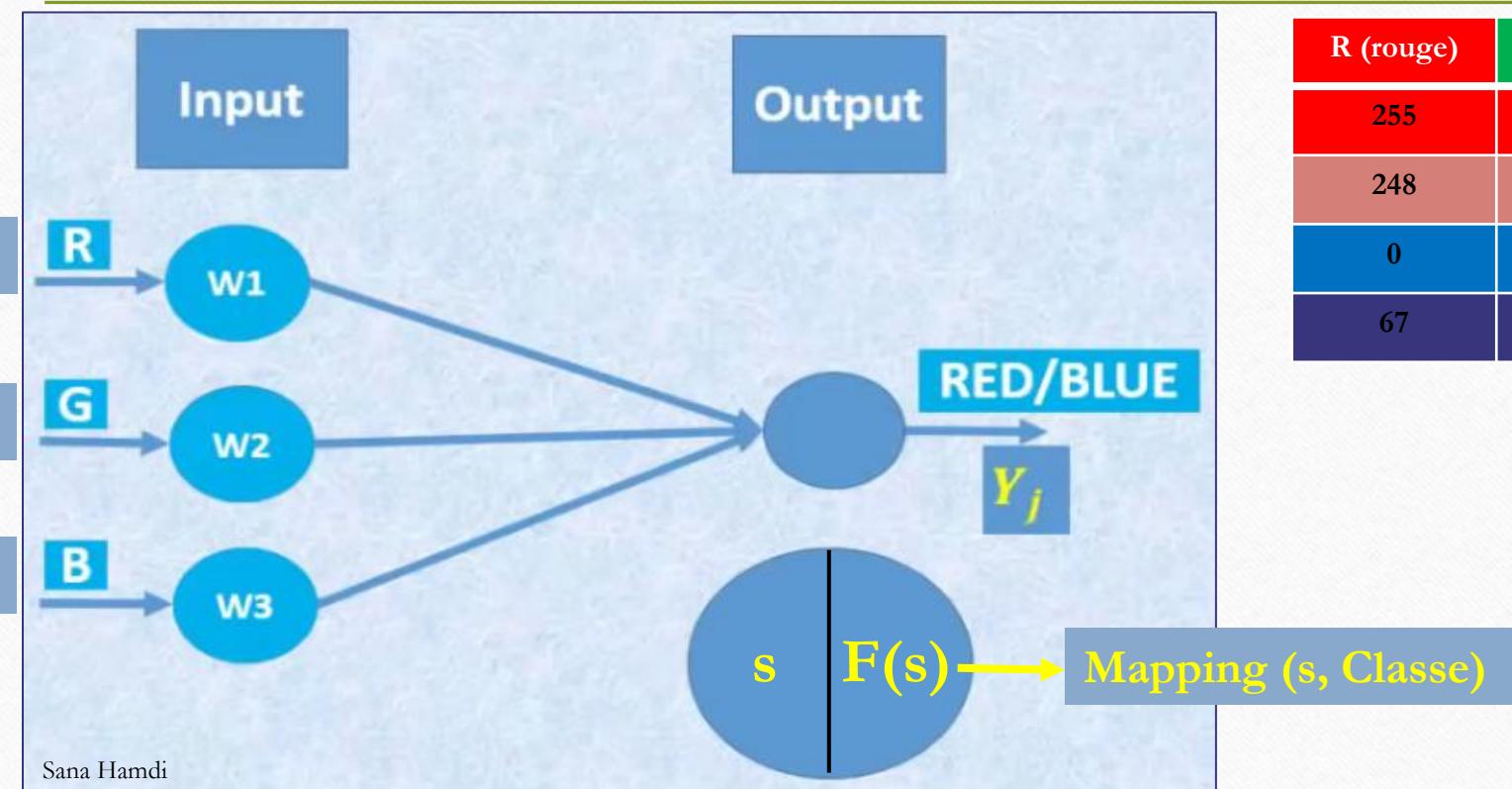
R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU

Composants du nœud de la sortie: fonction d'activation



R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU

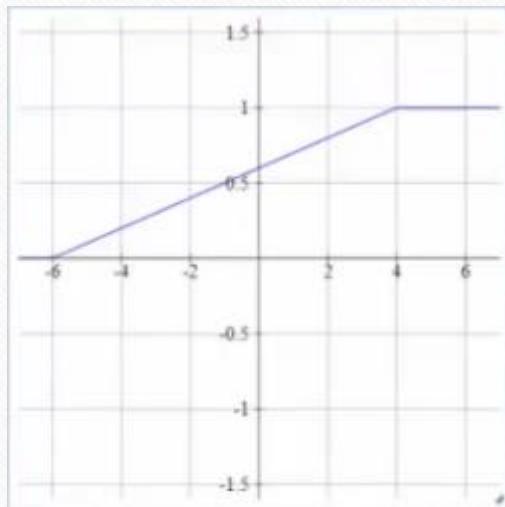
Composants du nœud de la sortie: fonction d'activation



R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU

Fonctions d'activation

La fonction linéaire par morceaux



$$f(x) = \max(0, x) + \sum_{s=1}^S a_s^s \max(0, -x + b_s^s)$$

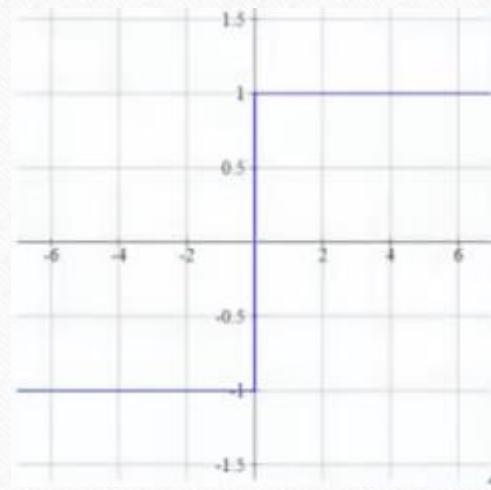
Sana Hamdi

La fonction sigmoïde



$$a_j^i = \sigma(z_j^i) = \frac{1}{1 + \exp(-z_j^i)}$$

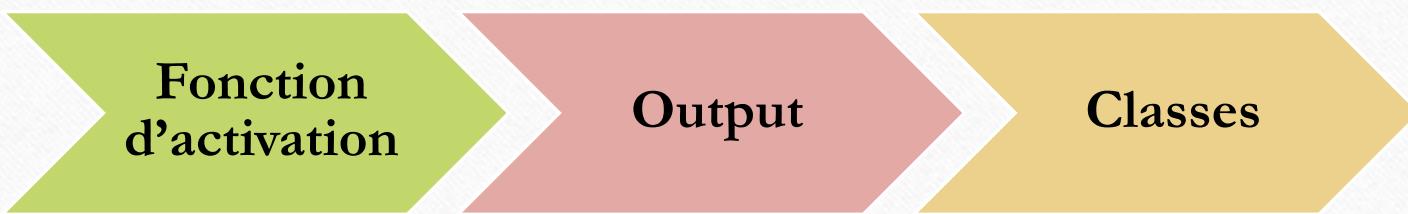
La fonction signe



$$a_j^i = \sigma(z_j^i) = \begin{cases} -1 & \text{if } z_j^i < 0 \\ 1 & \text{if } z_j^i > 0 \end{cases}$$

Fonction d'activation

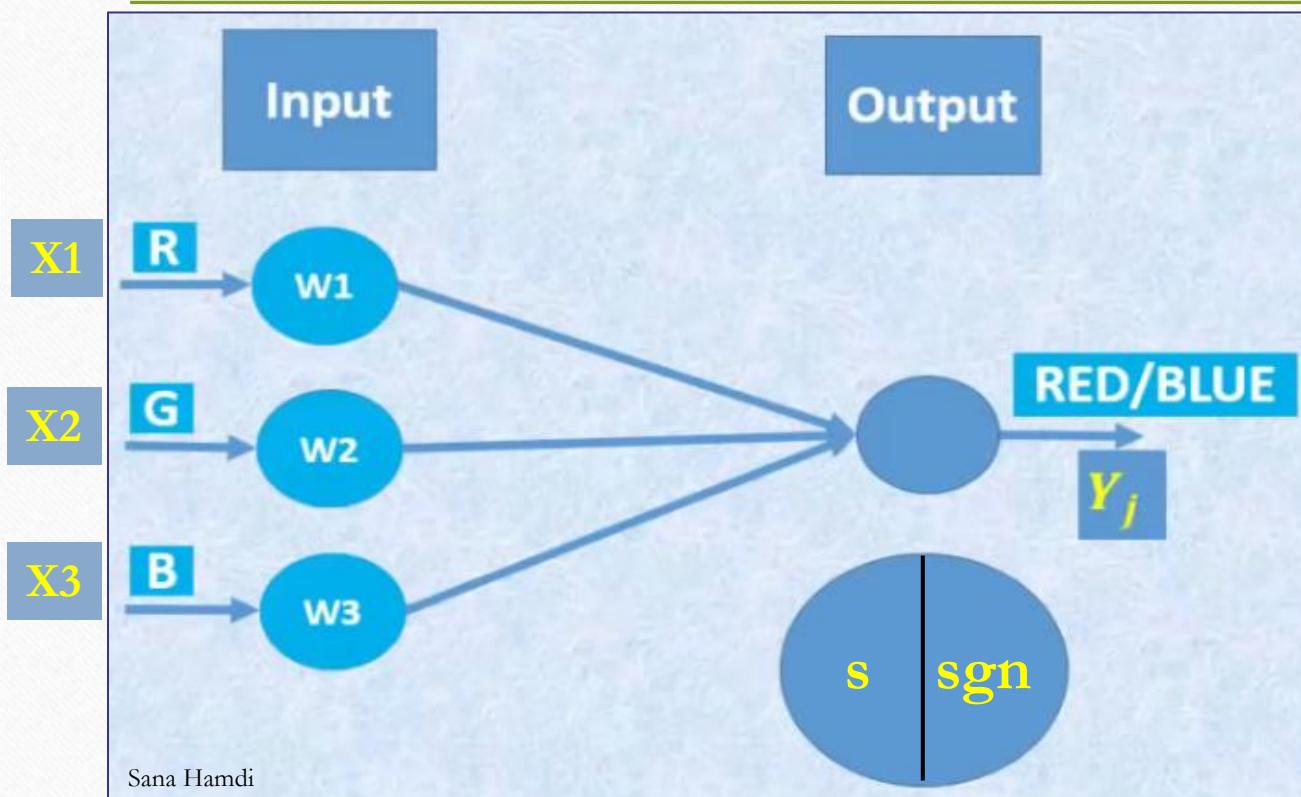
- Quelle fonction d'activation à choisir?



- Dans notre cas nous avons **deux** classes (ROUGE et BLEU)
 - La fonction d'activation à utiliser donne seulement 2 outputs.
 - **La fonction de signe**

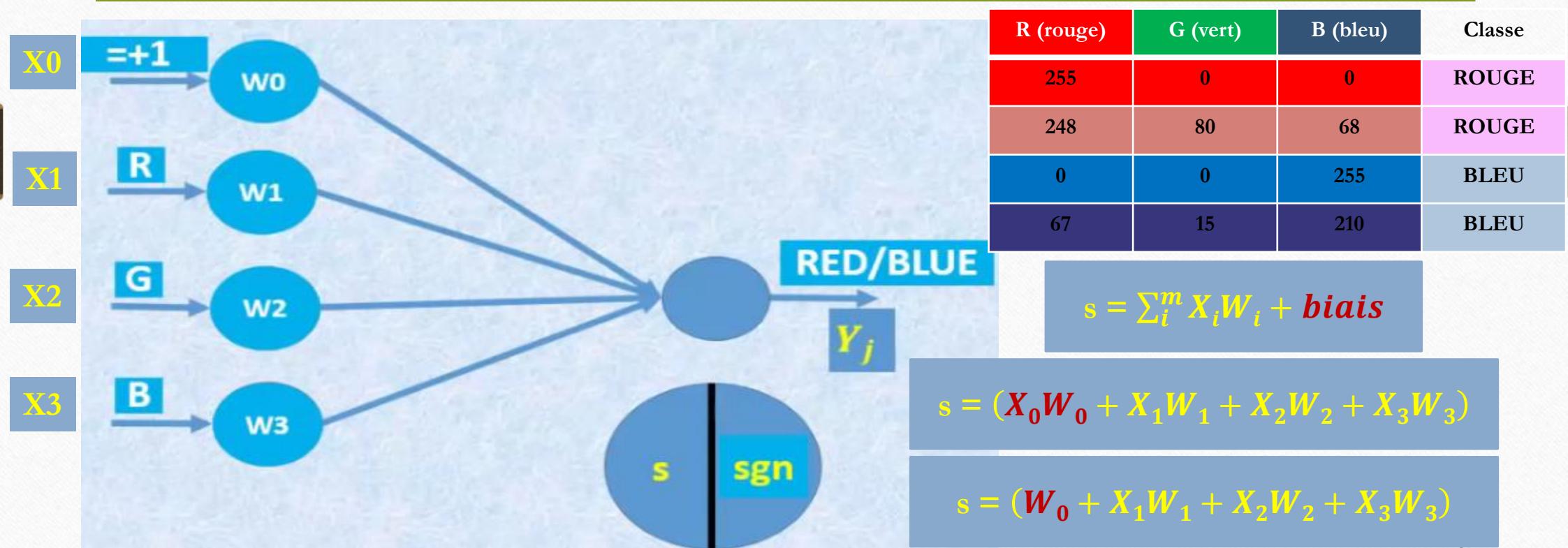
$$a_j^i = \sigma(z_j^i) = \begin{cases} -1 & \text{if } z_j^i < 0 \\ 1 & \text{if } z_j^i > 0 \end{cases}$$

Composants du nœud de la sortie

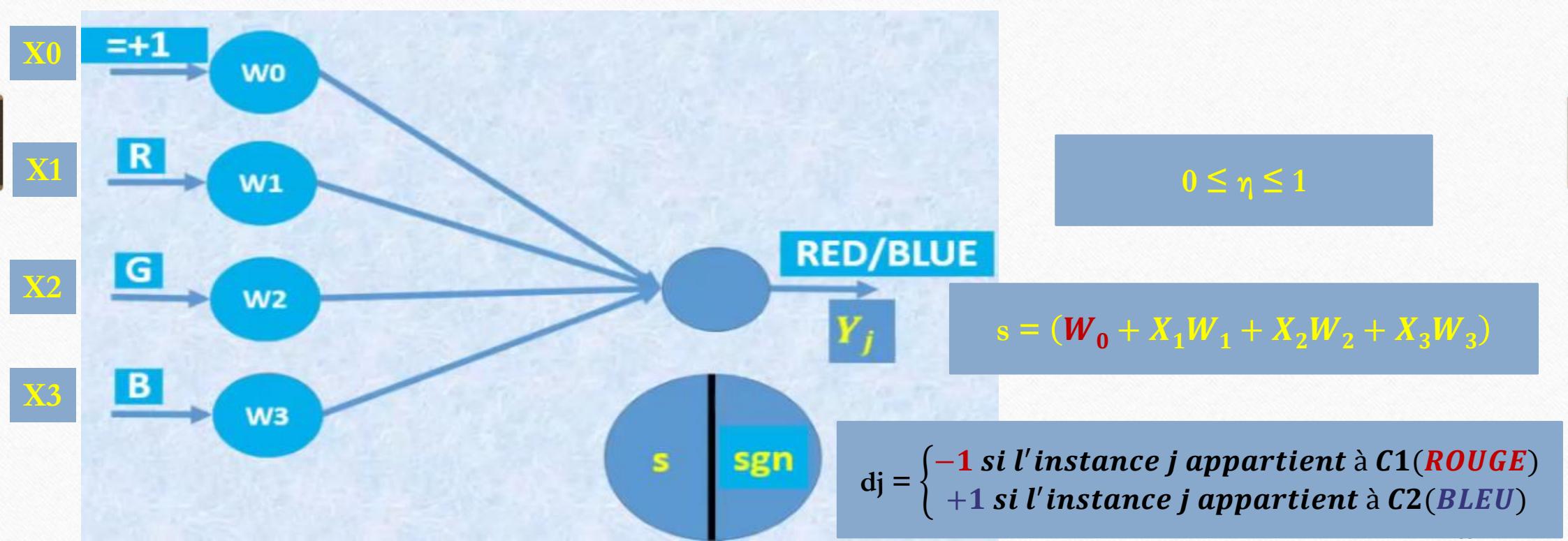


R (rouge)	G (vert)	B (bleu)	Classe
255	0	0	ROUGE
248	80	68	ROUGE
0	0	255	BLEU
67	15	210	BLEU

Le biais



La classe désirée



L'algorithme de perceptron: étapes d'apprentissage

1. Initialiser aléatoirement les poids synaptiques
 2. Passer les observations une à une
 3. Calculer la somme des produits
 4. Calculer la réponse de la fonction d'activation
 5. Mise à jour des poids
-
- ```
graph TD; 1[1. Initialiser aléatoirement les poids synaptiques] --> 2[2. Passer les observations une à une]; 2 --> 3[3. Calculer la somme des produits]; 3 --> 4[4. Calculer la réponse de la fonction d'activation]; 4 --> 5[5. Mise à jour des poids]; 5 --> 3
```

Jusqu'à convergence

# Mise à jour des poids

---

- Si la classe prédite  $\mathbf{Y}_j$  de l'instance  $j$ , à l'itération (étape de calcul)  $n$ , est différente de celle désirée  $\mathbf{d}_j$ , alors les poids doivent être mis à jour selon cette équation:

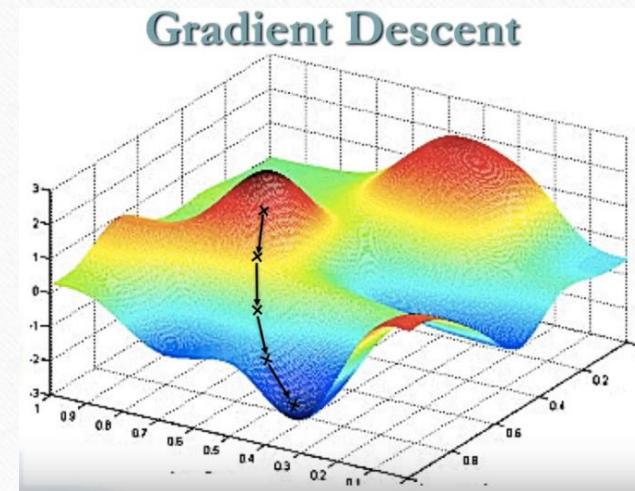
$$\mathbf{W}_i(n + 1) = \mathbf{W}_i(n) + \eta [\mathbf{d}(n) - \mathbf{Y}(n)] \mathbf{X}_i(n)$$

Avec

$i = [0, \dots, m]$  et  $m$  est le nombre d'entrées (attributs)

# Taux d'apprentissage

- Le taux d'apprentissage est un hyper-paramètre qui présente la rapidité de la réponse du réseau aux changements qu'il subisse.
- Il contrôle à quel point nous ajustons les poids de notre réseau en fonction de la descente du gradient.
- $0 \leq \eta \leq 1$



# Itération 0

---

- Pour chaque itération, les paramètres du réseau doivent être connus.
- Paramètres à l'itération  $n = 0$ :

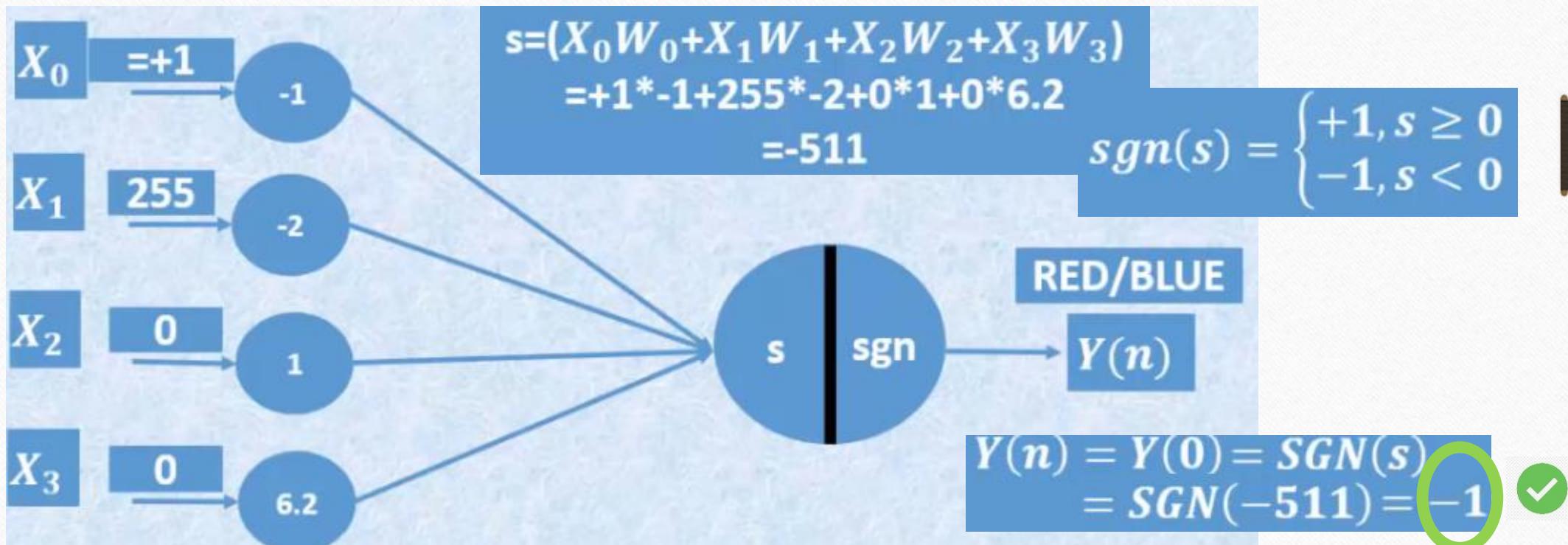
$$\eta=0,001$$

$$X(0)=[X_0, X_1, X_2, X_3]=[+1, 255, 0, 0]$$

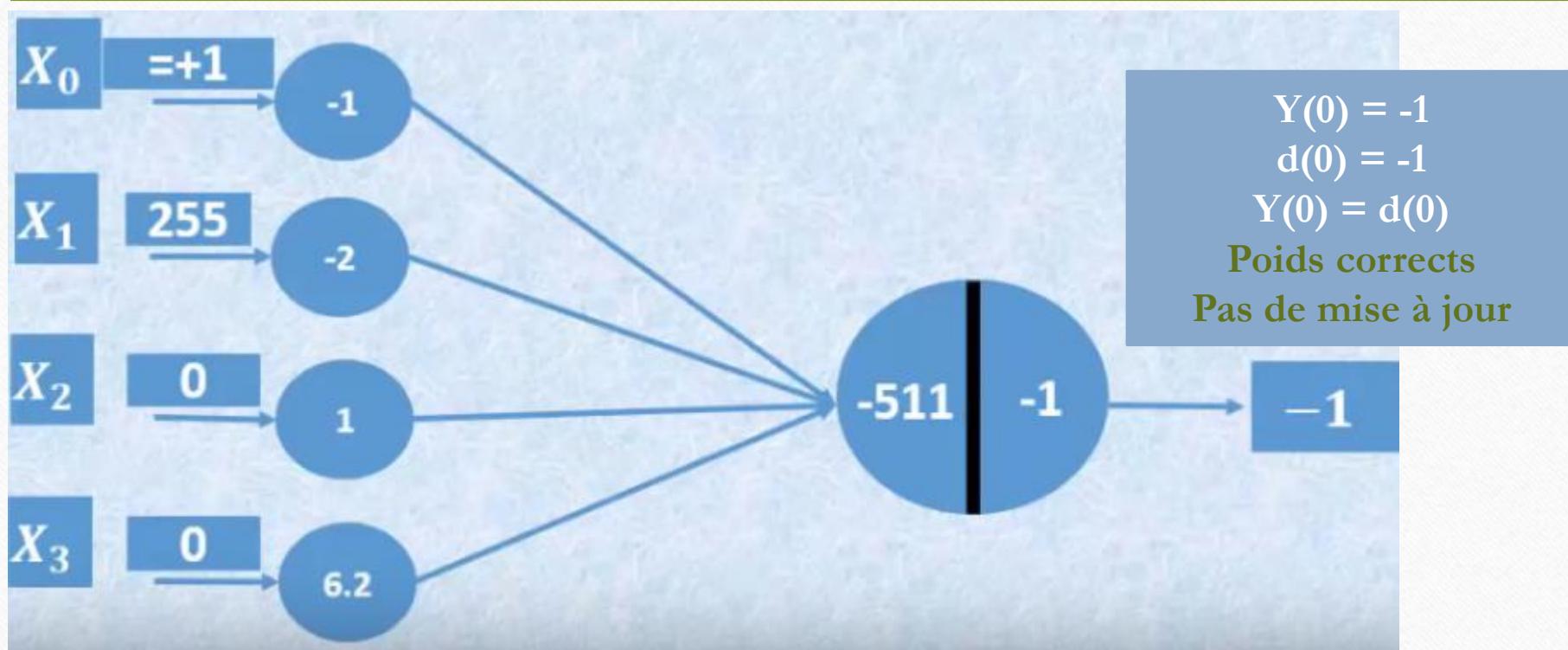
$$W(0) = [W_0, W_1, W_2, W_3]=[-1, -2, 1, 6.2]$$

$$d(0)=-1$$

# Itération 0



# Itération 0



# Itération 1

---

- Paramètres à l'itération  $n = 1$ :

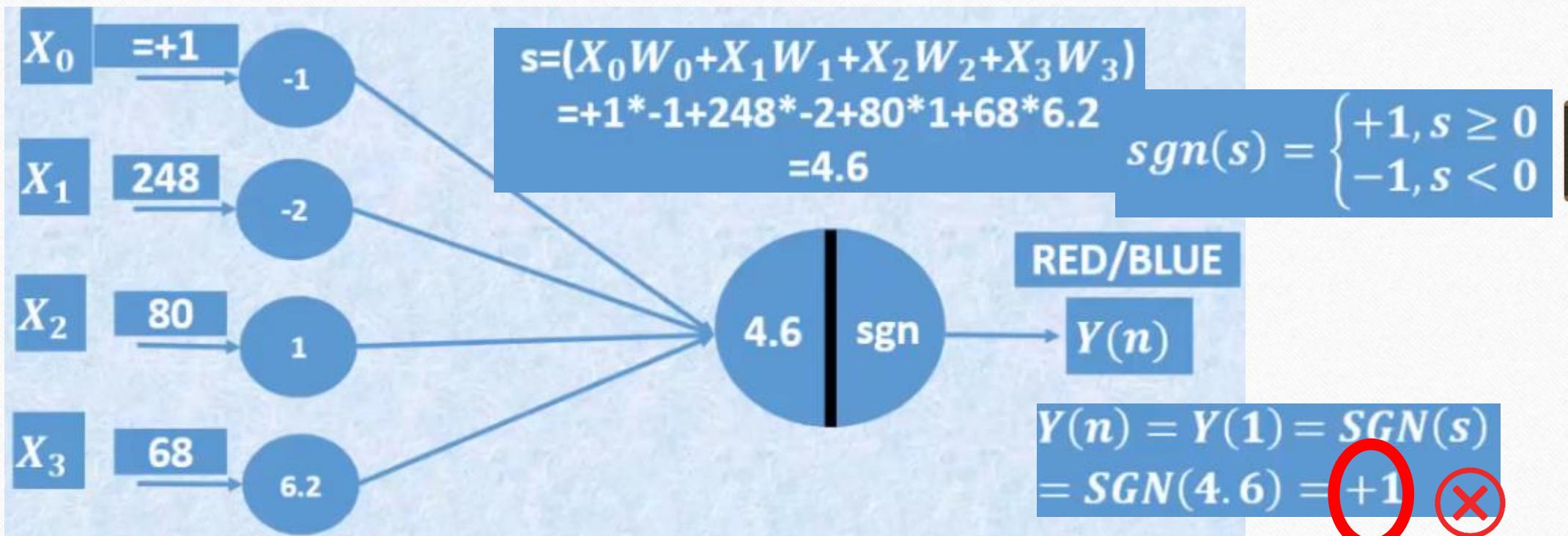
$$\eta=0,001$$

$$X(1)=[+1, 248, 80, 68]$$

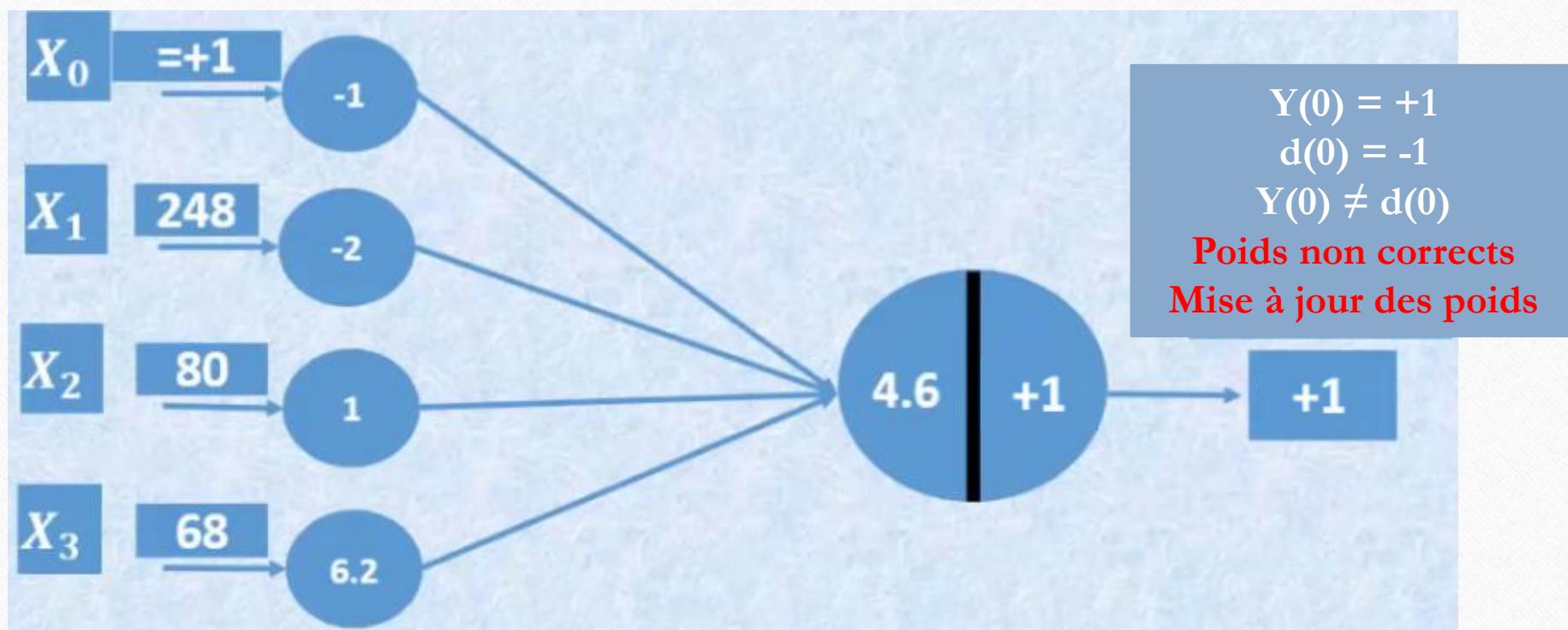
$$W(1)=[-1, -2, 1, 6.2]$$

$$d(0)=-1$$

# Itération 1



# Itération 1



# Mise à jour des poids pour l'itération 2

- Selon cette équation:

$$W_i(n+1) = W_i(n) + \eta [d(n) - Y(n)] X_i(n)$$

- Avec  $n = 1$ , on aura:

$$W(2) = W(1) + \eta [d(1) - Y(1)] X(1)$$

$$W(2) = [-1, -2, 1, 6.2] + 0.001[-1 - (+1)][+1, 248, 80, 68]$$

$$W(2) = [-1, -2, 1, 6.2] + 0.001[-2][+1, 248, 80, 68]$$

$$W(2) = [-1, -2, 1, 6.2] + [-0.002][+1, 248, 80, 68]$$

$$W(2) = [-1, -2, 1, 6.2] + [-0.002, -0.496, -0.16, -0.136]$$

$$W(2) = [-1.002, -2.496, 0.84, 6.064]$$

# Itération 2

---

- Paramètres à l'itération n = 2:

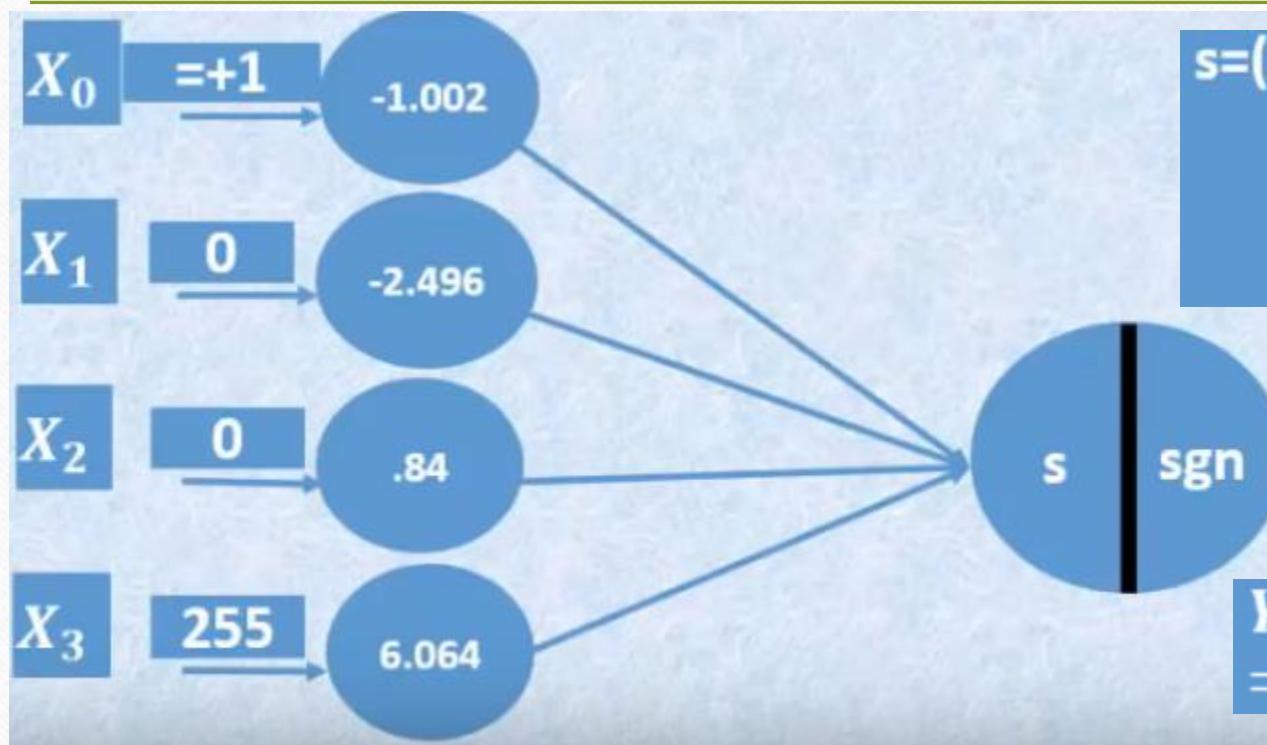
$$\eta=0,001$$

$$X(2)=[+1, 0, 0, 255]$$

$$W(2) = [-1.002, -2.496, 0.84, 6.064]$$

$$d(0)=+1$$

## Itération 2



$$\begin{aligned}s &= (X_0 W_0 + X_1 W_1 + X_2 W_2 + X_3 W_3) \\&= +1 * -1.002 + 0 * -2.496 + 0 * .84 + 255 * 6.064 \\&= 1545.32\end{aligned}$$

$Y(0) = +1$   
 $d(0) = +1$   
Poids corrects  
Pas de mise à jour

$$\begin{aligned}Y(n) &= Y(2) = SGN(s) \\&= SGN(1545.32) = +1\end{aligned}$$



# Itération 3

---

- Paramètres à l'itération n = 3:

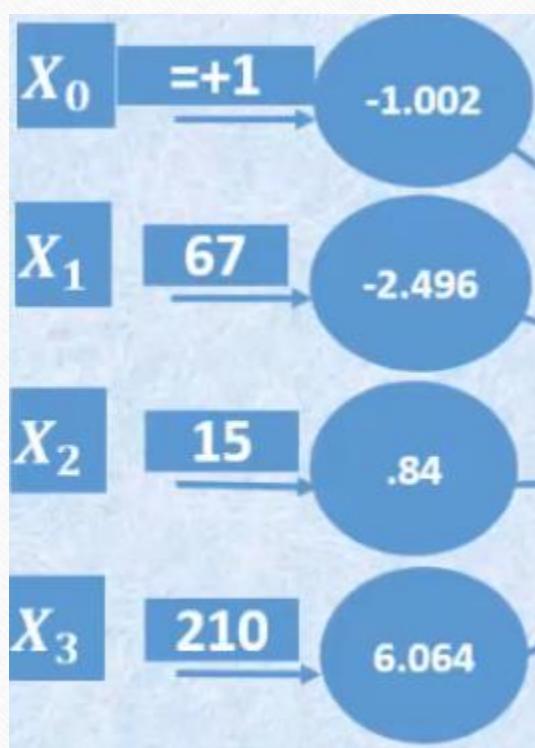
$$\eta=0,001$$

$$X(3)=[+1, 67, 15, 210]$$

$$W(3) = [-1.002, -2.496, 0.84, 6.064]$$

$$d(0)=+1$$

# Itération 3



$$\begin{aligned}s &= (X_0 W_0 + X_1 W_1 + X_2 W_2 + X_3 W_3) \\&= +1 * -1.002 + 67 * -2.496 \\&\quad + 15 * .84 + 210 * 6.064 \\&= 1349.542\end{aligned}$$

$Y(0) = +1$   
 $d(0) = +1$   
Poids corrects  
Pas de mise à jour

$$\begin{aligned}Y(n) &= Y(3) = \text{SGN}(s) \\&= \text{SGN}(1349.542) = +1\end{aligned}$$

✓

# Itération 4

---

- On n'a pas encore terminé
- On doit entraîner le modèle avec les 2 premières instances en utilisant les nouvelles valeurs des poids.
- Paramètres à l'itération  $n = 4$ :

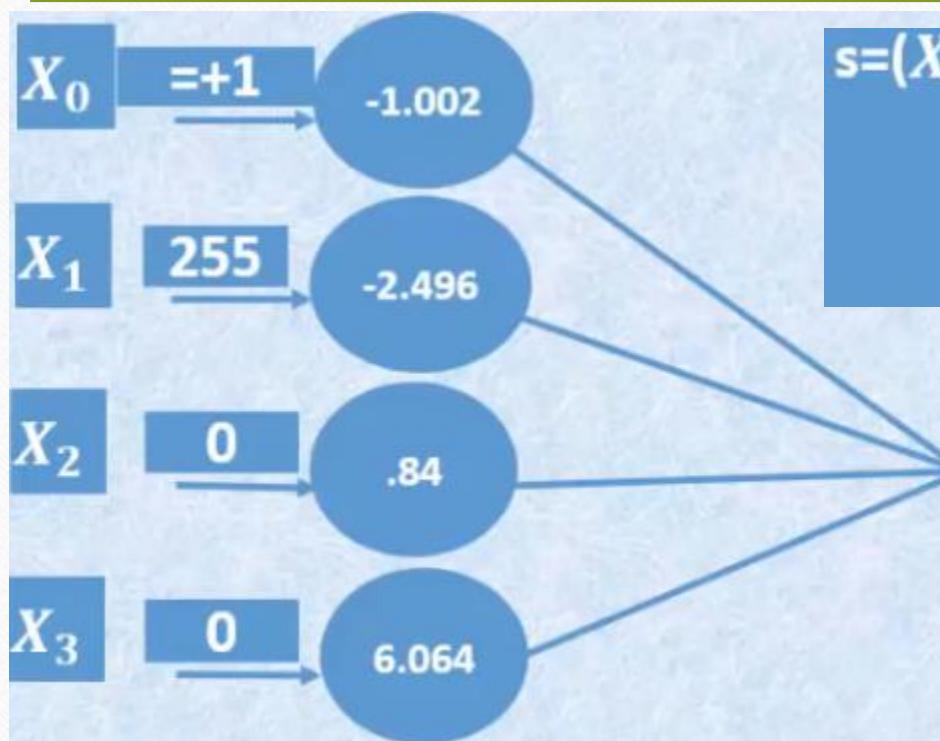
$$\eta=0,001$$

$$X(4)=[+1, 255, 0, 0]$$

$$W(4) = [-1.002, -2.496, 0.84, 6.064]$$

$$d(0)=-1$$

## Itération 4



$$\begin{aligned}s &= (X_0 W_0 + X_1 W_1 + X_2 W_2 + X_3 W_3) \\&= +1 * -1.002 + 255 * -2.496 + 0 * .84 + 0 * 6.064 \\&= -637.482\end{aligned}$$

$Y(0) = -1$   
 $d(0) = -1$   
Poids corrects  
Pas de mise à jour

$$\begin{aligned}Y(n) &= Y(4) \\&= SGN(-637.482) = -1\end{aligned}$$



# Itération 5

---

- Paramètres à l'itération n = 5:

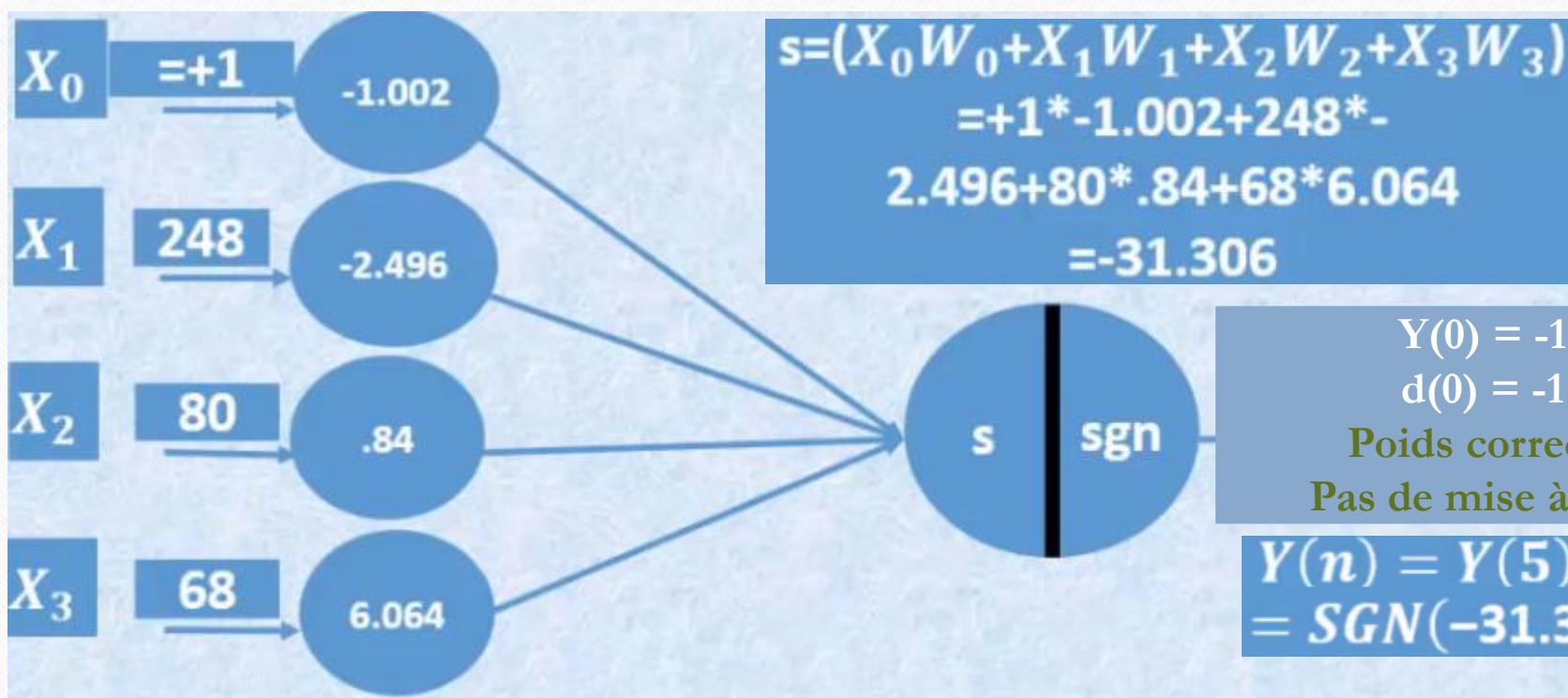
$$\eta=0,001$$

$$X(5)=[+1, 248, 80, 68]$$

$$W(5) = [-1.002, -2.496, 0.84, 6.064]$$

$$d(0)=-1$$

# Itération 5



# Prédiction

---

- Après entraîner notre réseau de neurones avec toutes les entrées en utilisant les poids mis à jour, on peut conclure que ces poids sont justes pour ce réseau et ce modèle d'apprentissage.
- Après la phase d'apprentissage, on peut utiliser le modèle pour prédire la classe d'une nouvelle instance,
- Quelle est la classe (ROUGE ou BLEU) de l'instance :

**R=150, G=100, B=180**

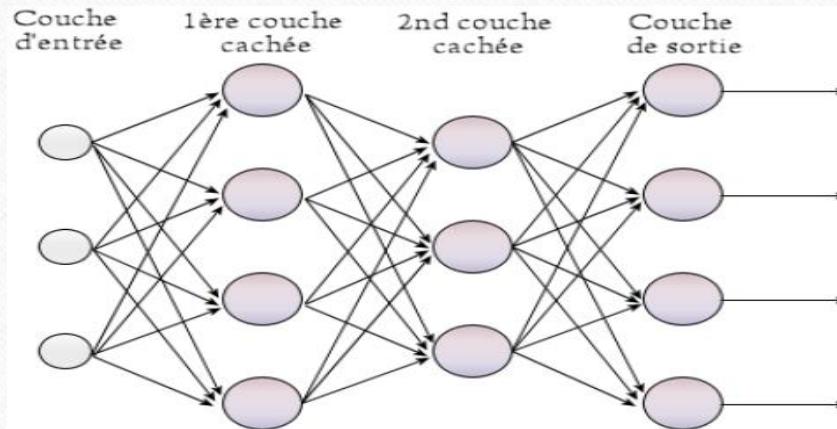
# Perceptron multicouche



# Introduction

---

- La première version du perceptron était **mono-couche**
  - Non capable de résoudre des problèmes non linéaires
- Perceptron **multicouche**: Ajouter des couches cachées pour résoudre les problèmes non linéaires



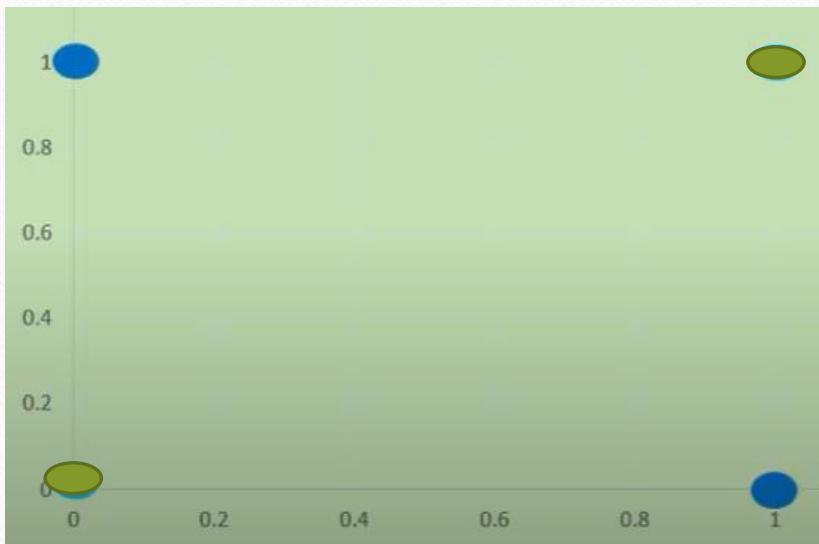
# Exemple de classification: problème logique XOR

---

- Résolution d'un problème logique de type XOR

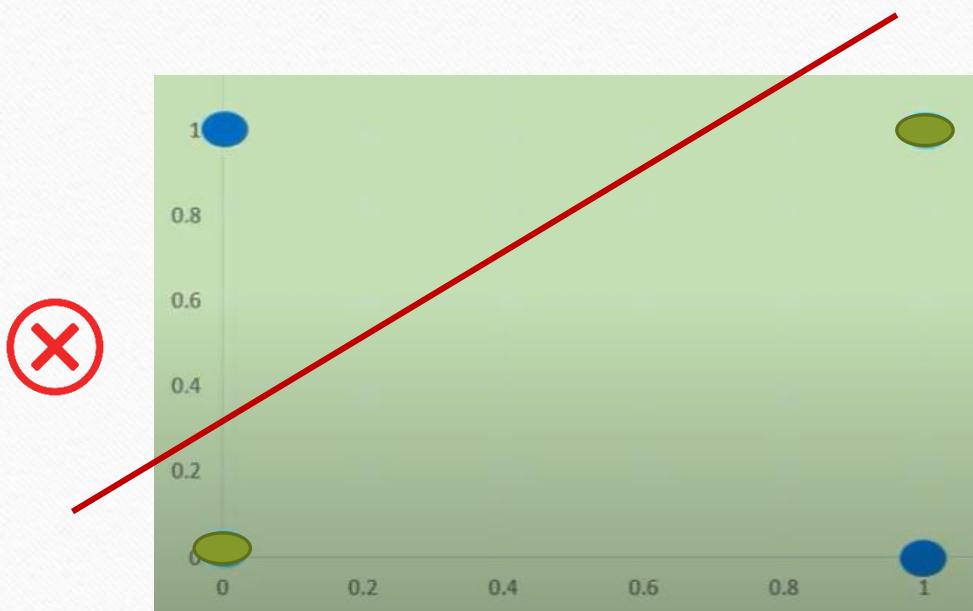
| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

# Exemple de classification: Résolution non linéaire



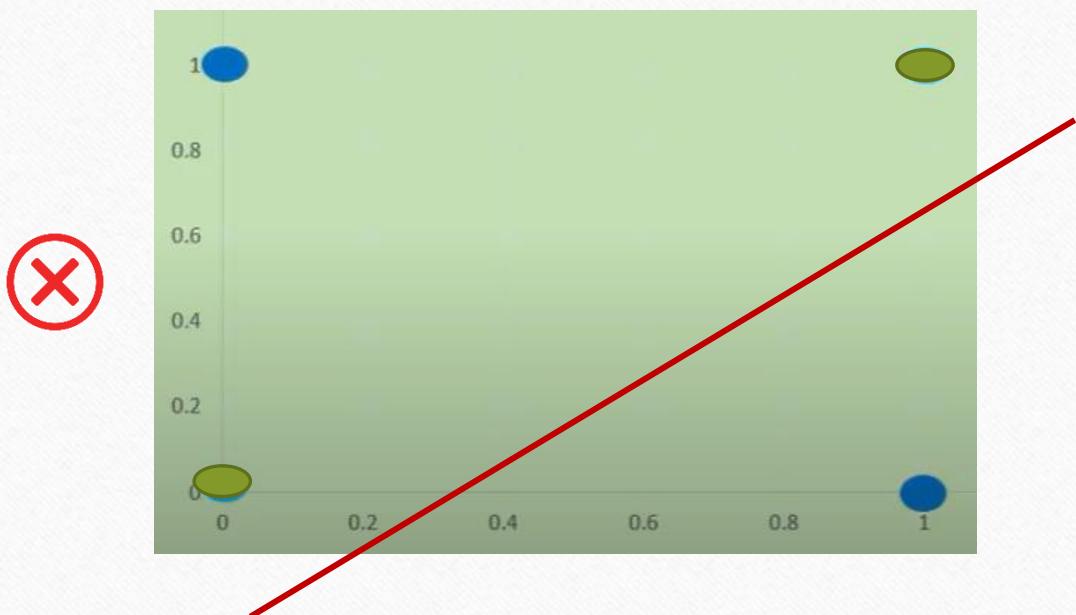
| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

# Exemple de classification: Résolution non linéaire



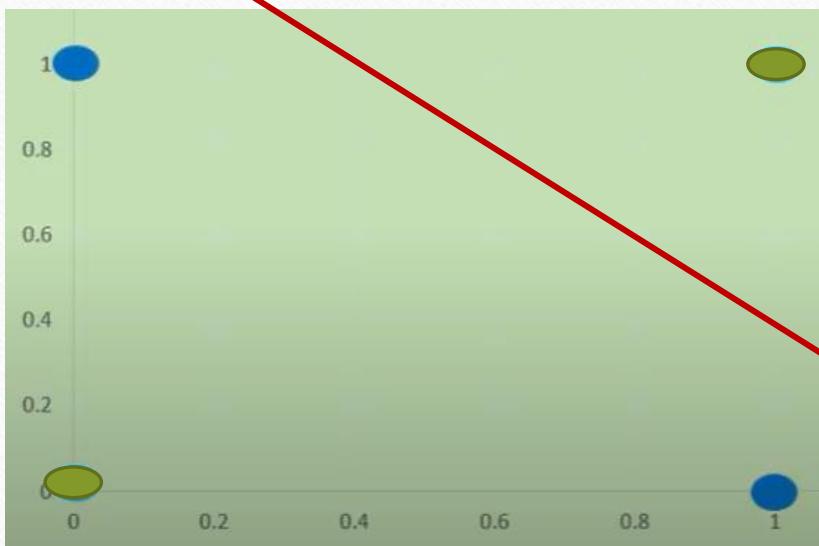
| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

# Exemple de classification: Résolution non linéaire



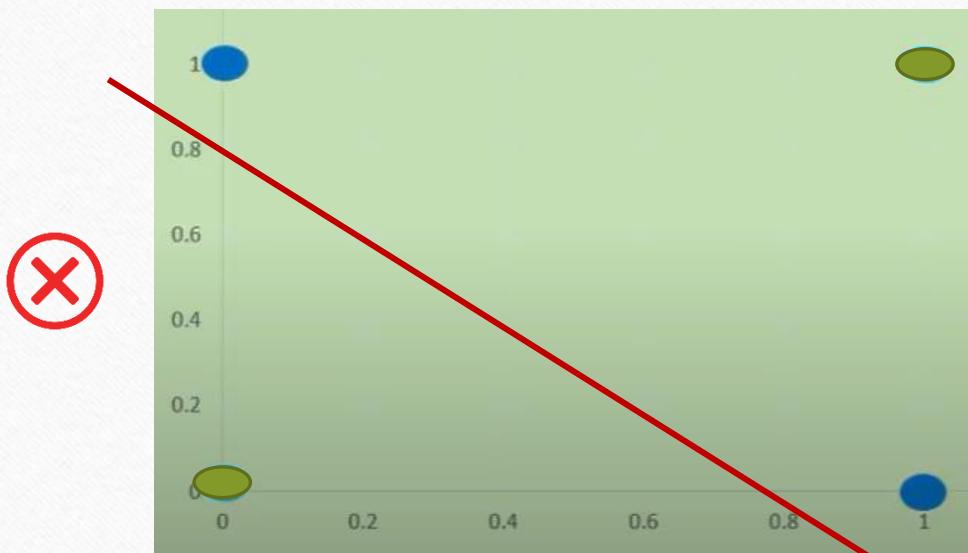
| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

# Exemple de classification: Résolution non linéaire



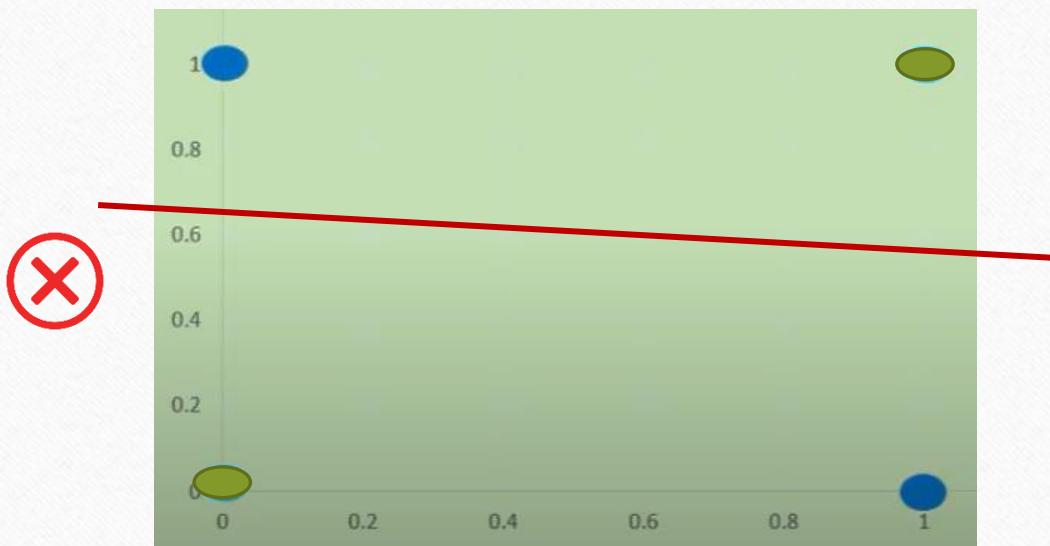
| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

# Exemple de classification: Résolution non linéaire



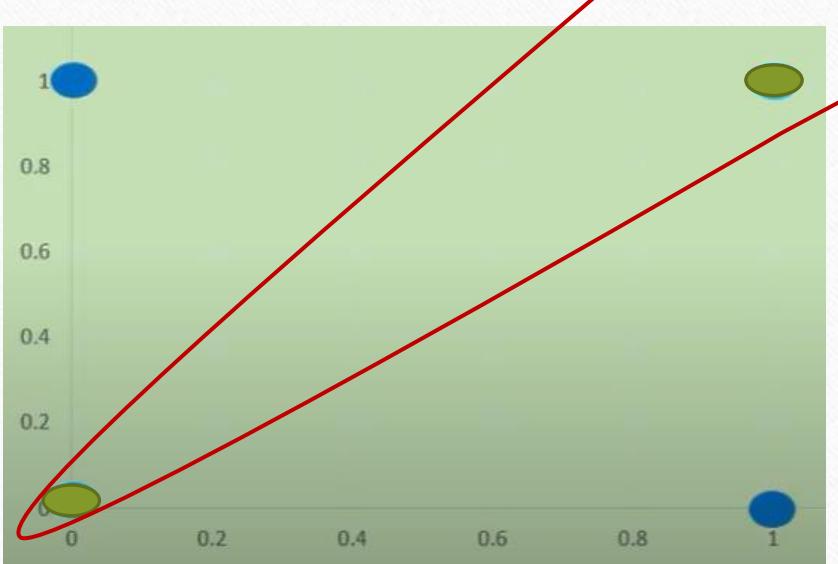
| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

# Exemple de classification: Résolution non linéaire



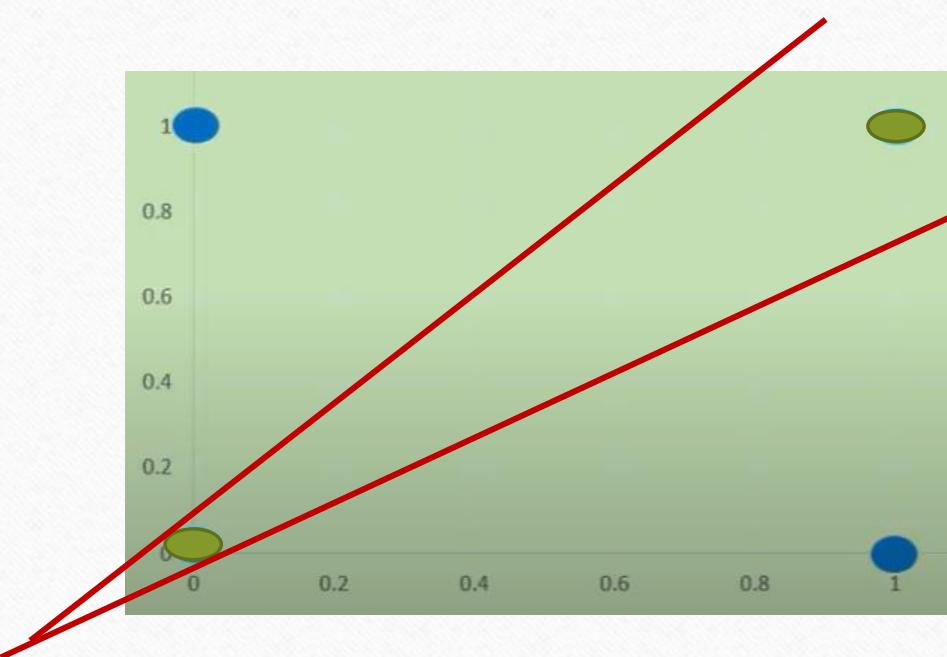
| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

# Exemple de classification: Résolution non linéaire

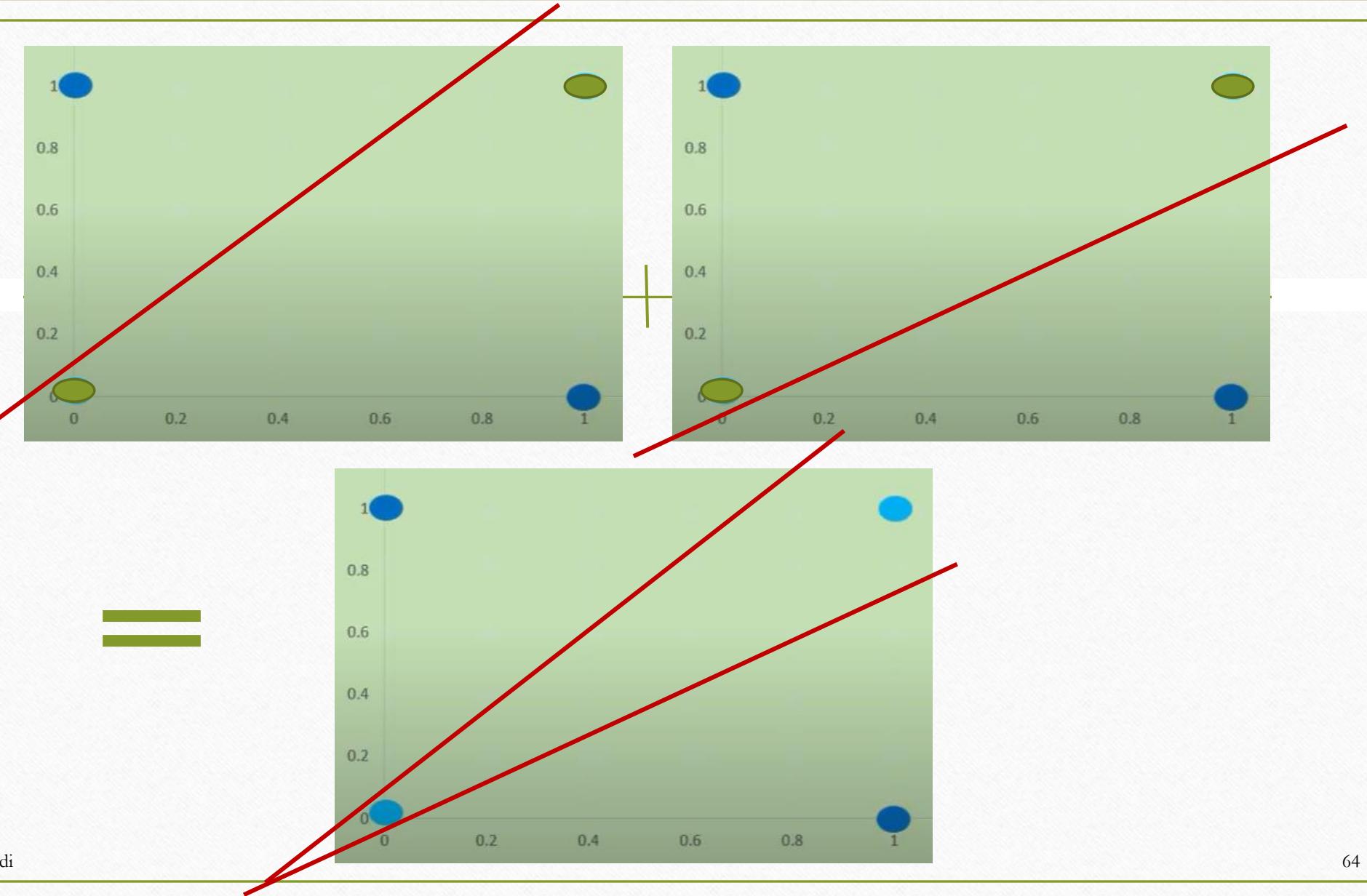


| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

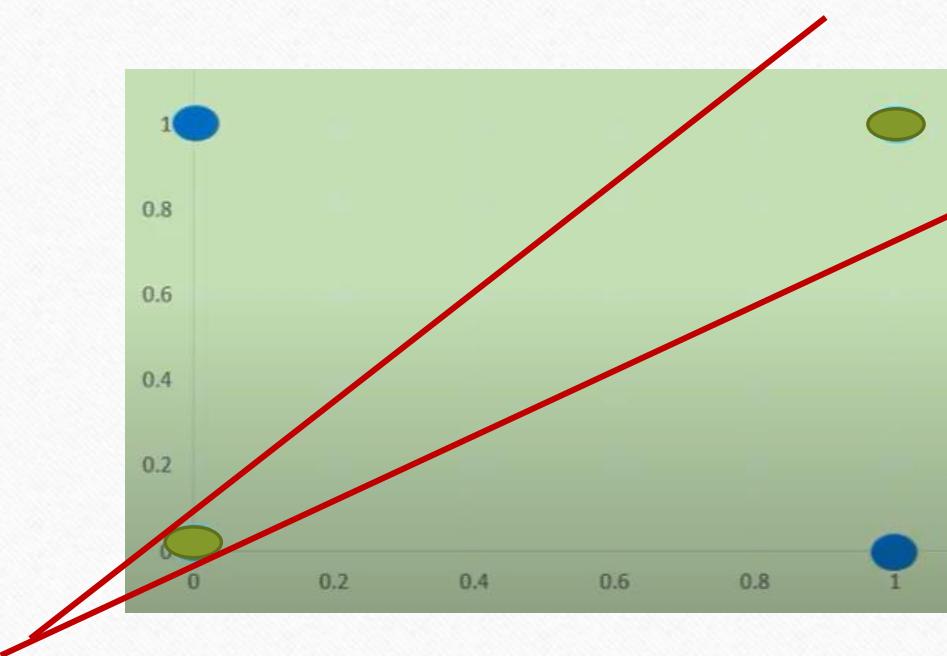
# Exemple de classification



| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

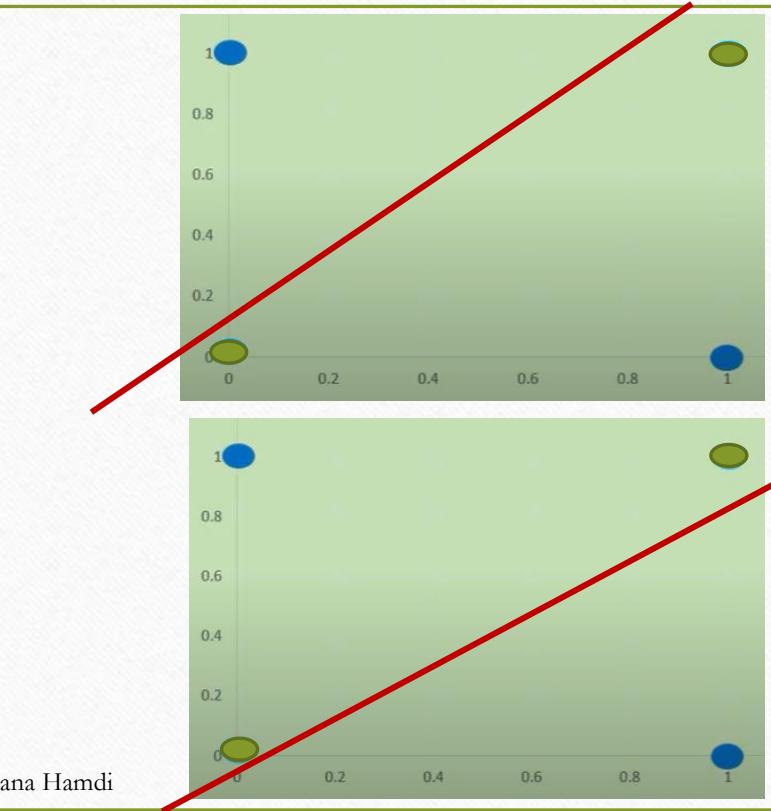


# Exemple de classification

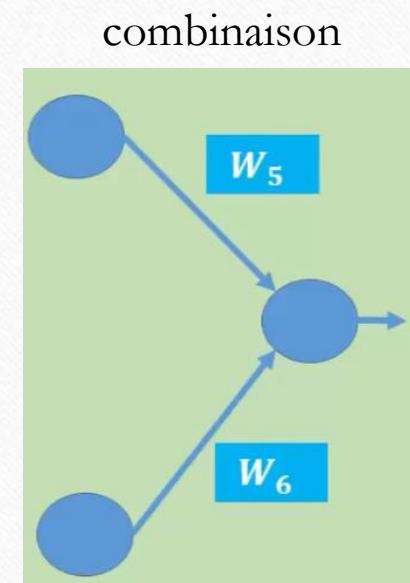
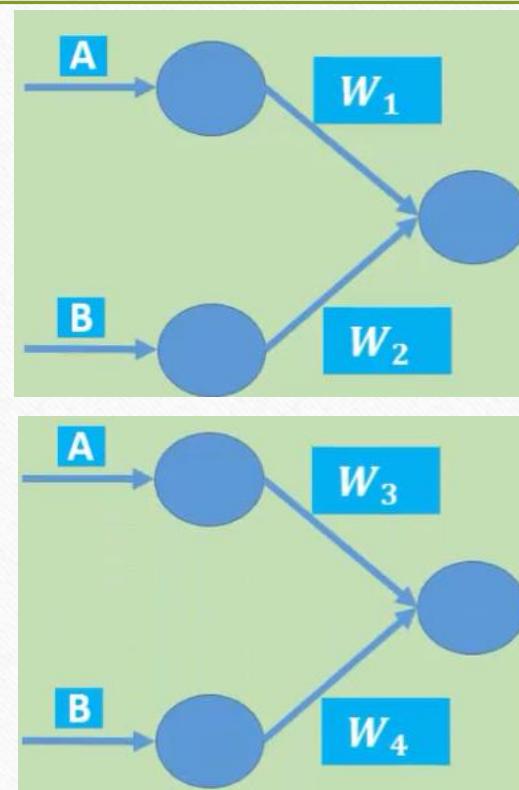


| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

# Exemple de classification: Architecture

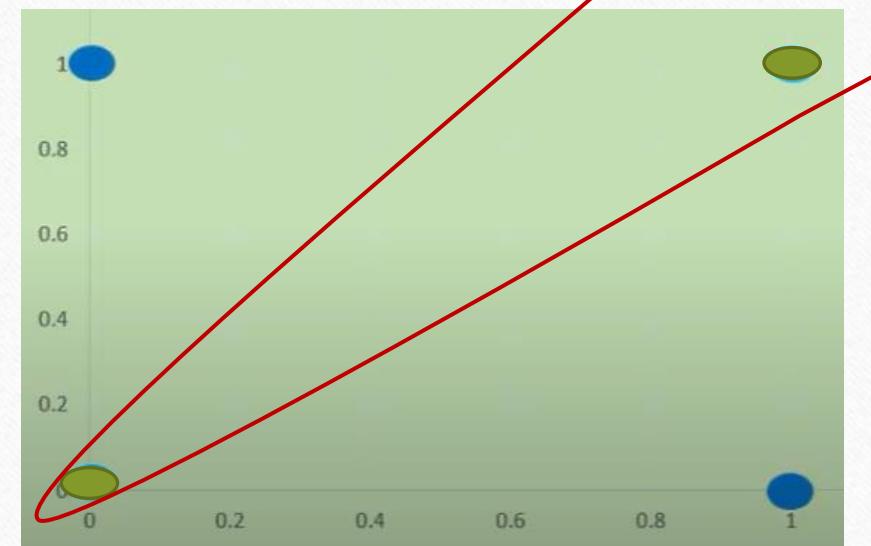
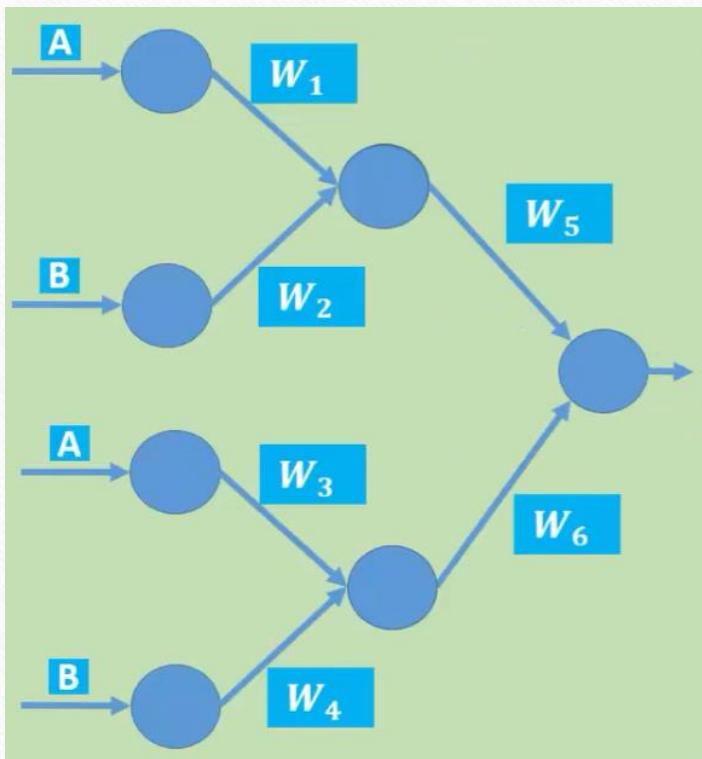


Sana Hamdi

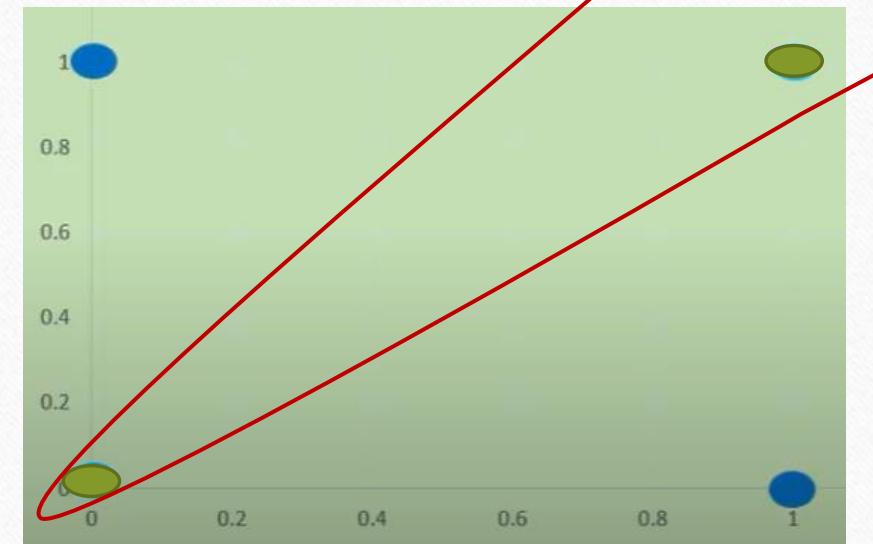
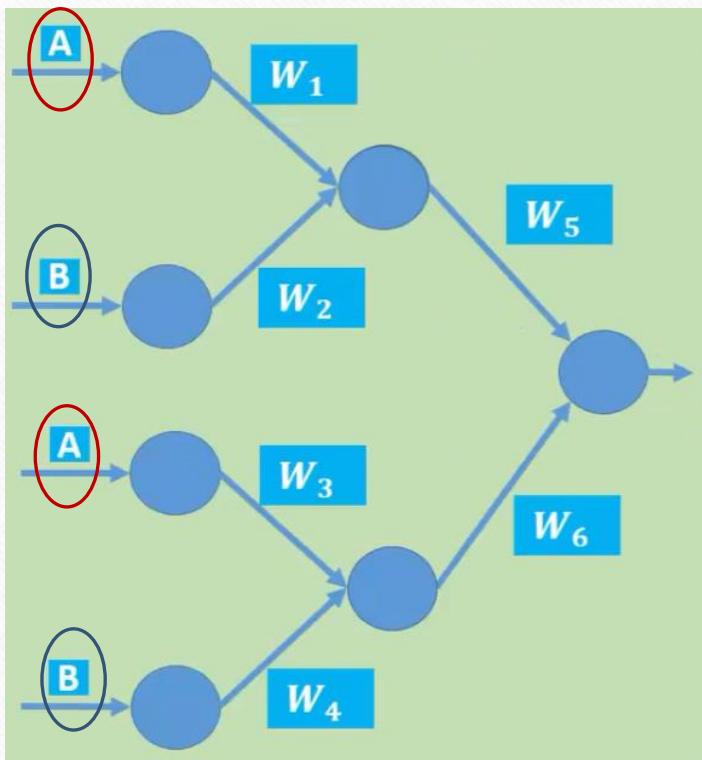


combinaison

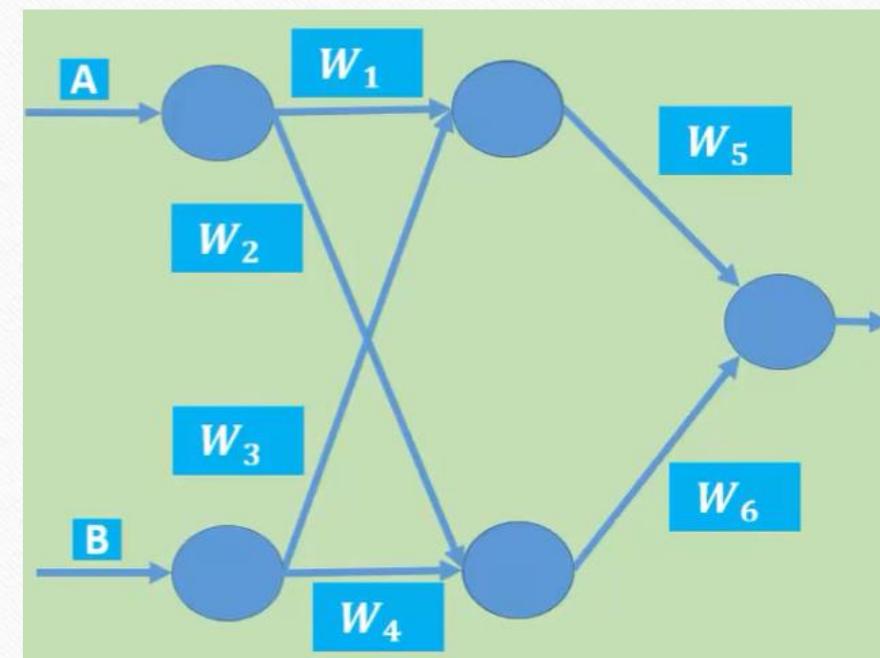
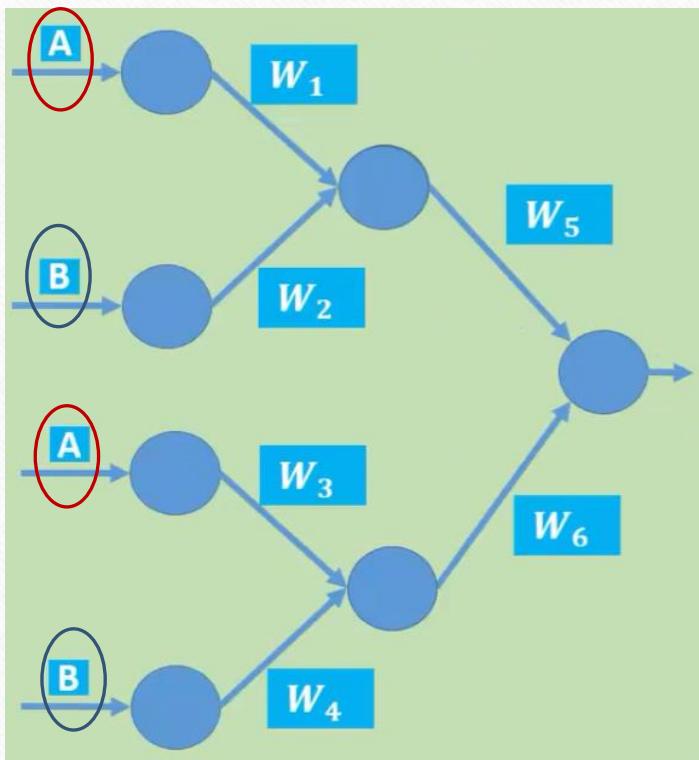
# Exemple de classification: Architecture



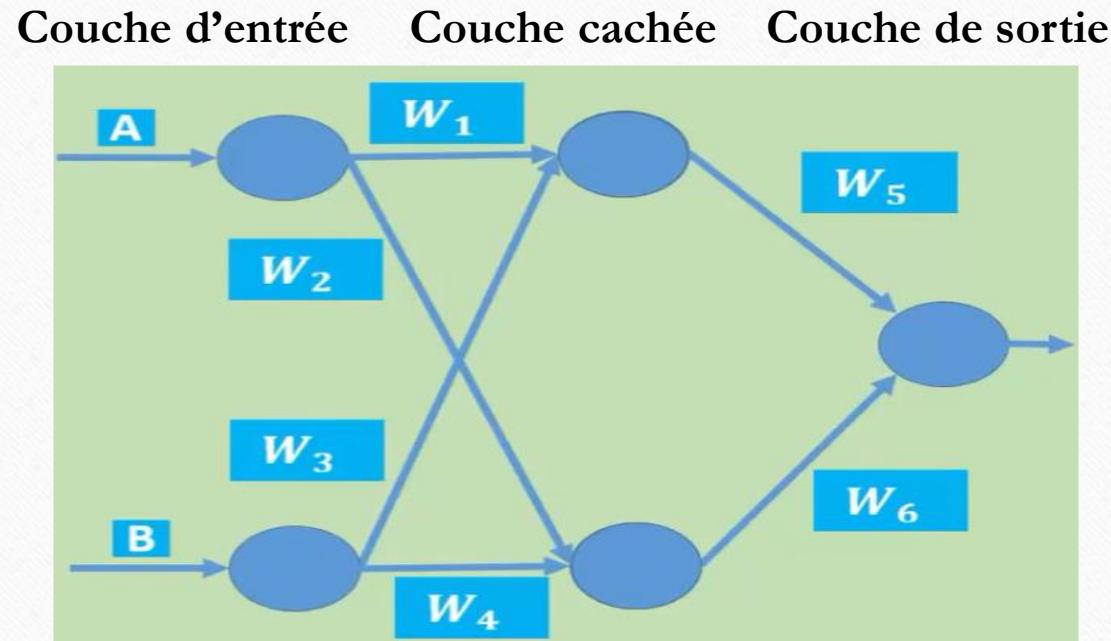
# Exemple de classification: Architecture



# Exemple de classification: Architecture



# Exemple de classification: Architecture



Les neurones d'une couche sont reliés à la totalité des neurones des couches adjacentes,

# Etapes d'apprentissage

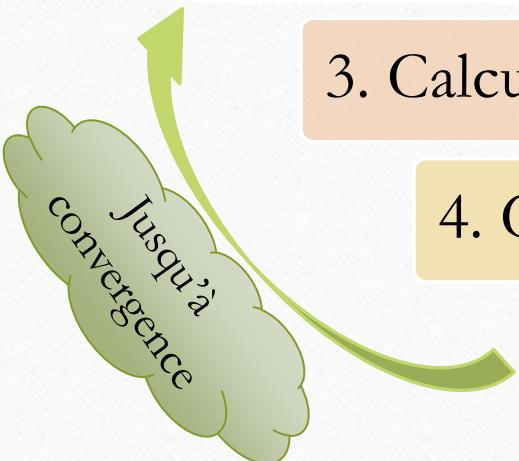
1. Initialiser les poids synaptiques

2. Passer les observations une à une

3. Calculer la somme des produits

4. Calculer la réponse de la fonction d'activation

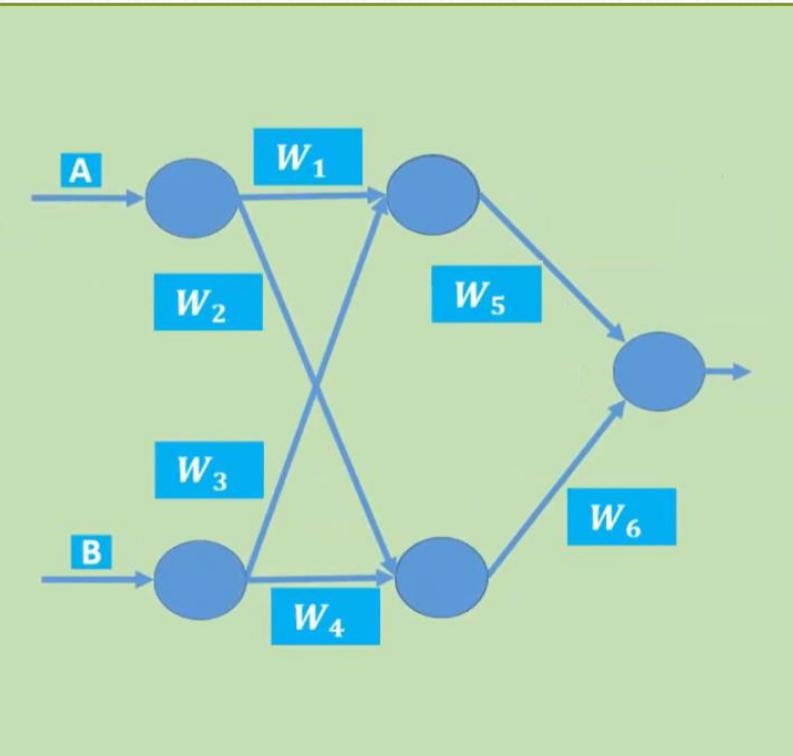
5. Mise à jour des poids



# Exemple de classification: Apprentissage

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

- Pour chaque itération, les paramètres du réseau doivent être connus.



# Exemple de classification: Apprentissage

- Paramètres à l'itération  $n = 0$ :

$$\eta=0,001$$

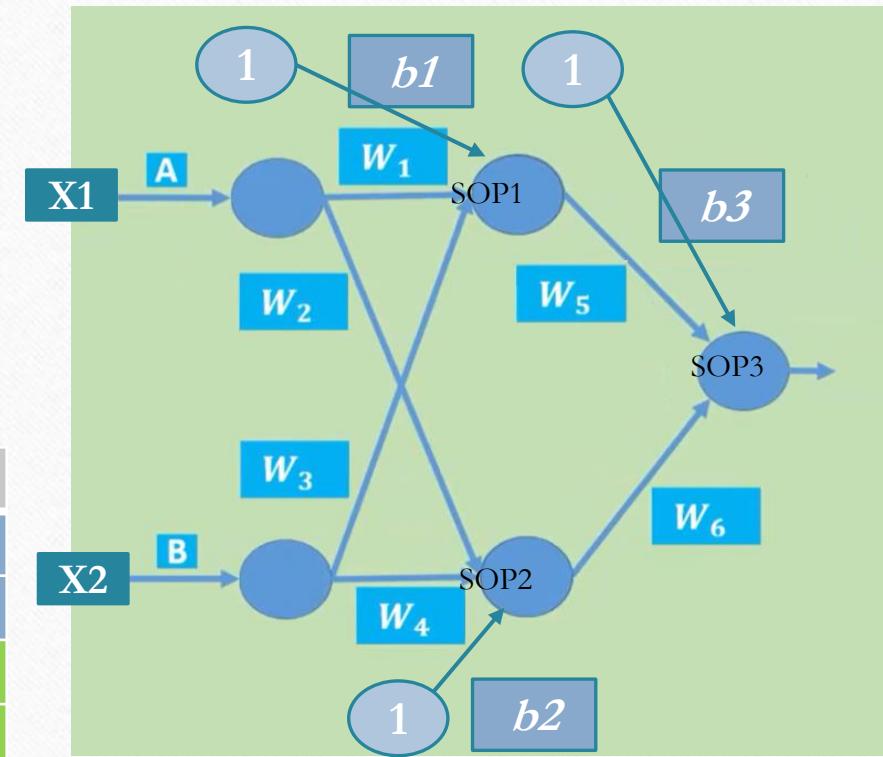
$$X(0)=[1,1,1,X_1,X_2]=[1,1,1, 1,0]$$

$$\begin{aligned}W(0) &= [b1,b2,b3,W_1,W_2,W_3,W_4,W_5,W_6] \\&=[-1.5, -0.5, -0.5, 1,1,1,1,-2, 1]\end{aligned}$$

$$d(0)=1$$

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

Sana Hamdi



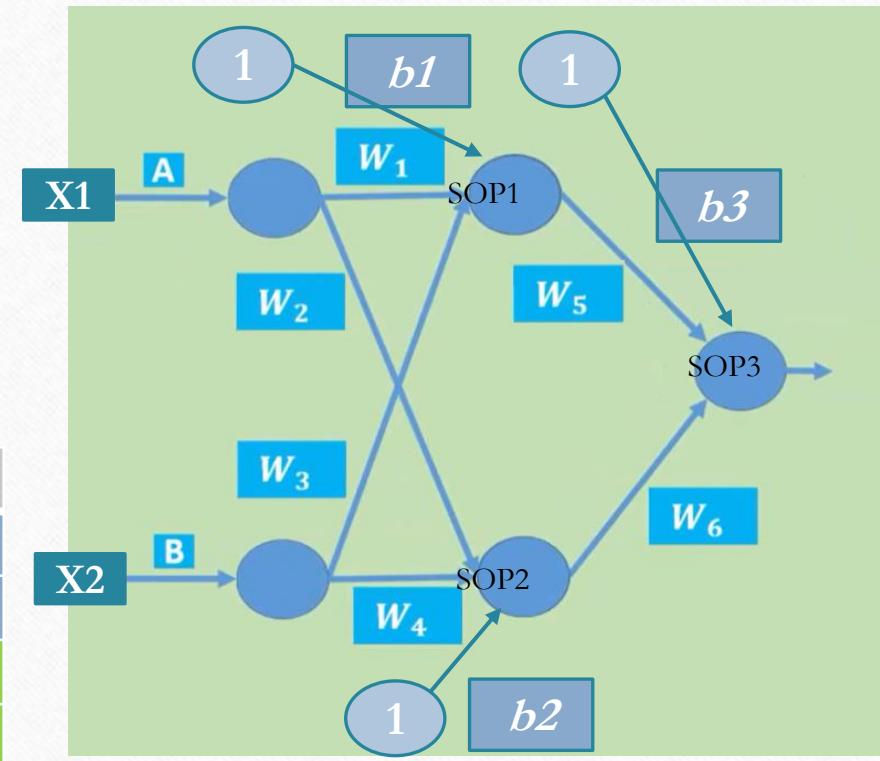
# Exemple de classification: Apprentissage

- l'itération  $n = 0$ :

$$\begin{aligned} \text{SOP1} &= (1 * b1 + X1 * W1 + X2 * W3) \\ &= -1.5 + 1 + 0 = \textcolor{red}{-0.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1 * b2 + X1 * W2 + X2 * W4) \\ &= -0.5 + 1 + 0 = \textcolor{red}{0.5} \end{aligned}$$

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |



# Exemple de classification: Apprentissage

- l'itération n = 0:

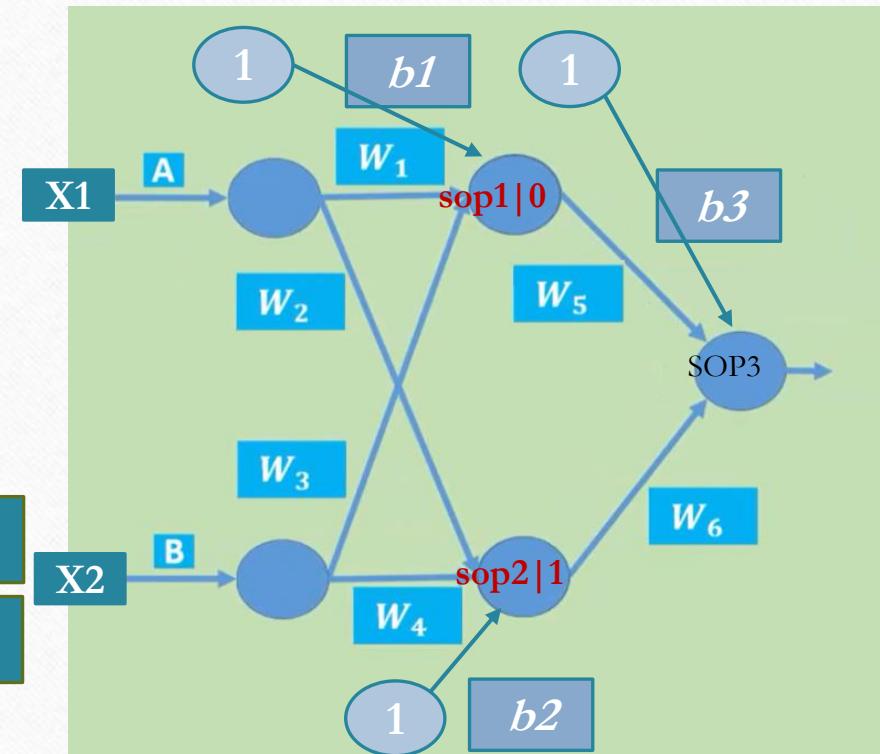
$$\begin{aligned} \text{SOP1} &= (1*b1 + X1*W1 + X2*W3) \\ &= -1.5 + 1 + 0 = \textcolor{red}{-0.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1*b2 + X1*W2 + X2*W4) \\ &= -0.5 + 1 + 0 = \textcolor{red}{0.5} \end{aligned}$$

la fonction de Heaviside

$$H(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0. \end{cases}$$

|                                                              |
|--------------------------------------------------------------|
| $Y_1 = Y(\text{SOP1}) = H(\text{SOP1}) = \textcolor{red}{0}$ |
| $Y_2 = Y(\text{SOP2}) = H(\text{SOP2}) = \textcolor{red}{1}$ |



# Exemple de classification: Apprentissage

- l'itération n = 0:

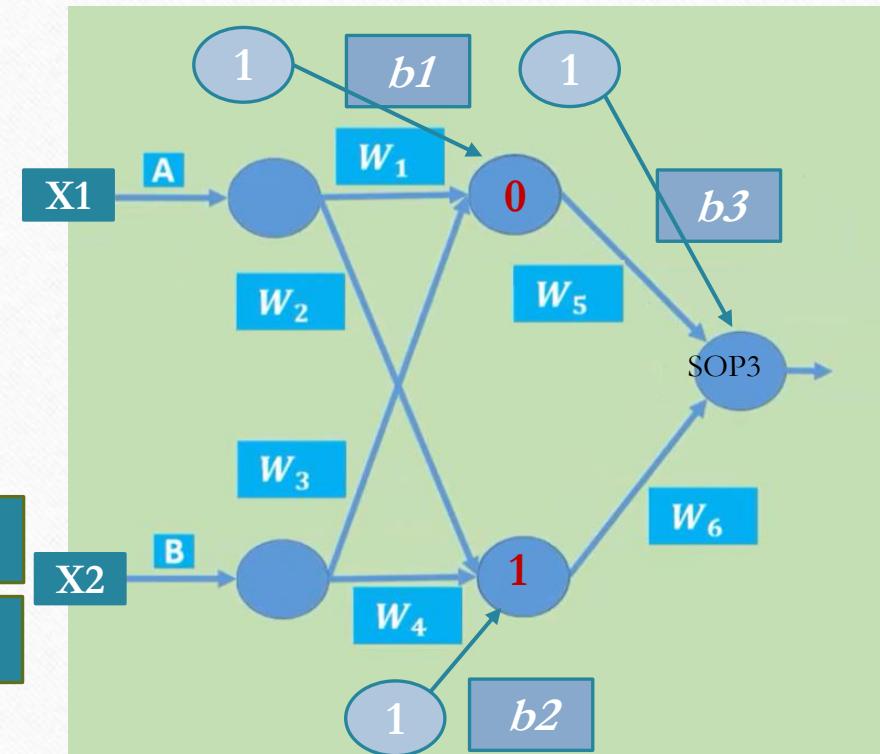
$$\begin{aligned} \text{SOP1} &= (1*b1 + X1*W1 + X2*W3) \\ &= -1.5 + 1 + 0 = \textcolor{red}{-0.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1*b2 + X1*W2 + X2*W4) \\ &= -0.5 + 1 + 0 = \textcolor{red}{0.5} \end{aligned}$$

la fonction de Heaviside

$$H(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0. \end{cases}$$

|                                                              |
|--------------------------------------------------------------|
| $Y_1 = Y(\text{SOP1}) = H(\text{SOP1}) = \textcolor{red}{0}$ |
| $Y_2 = Y(\text{SOP2}) = H(\text{SOP2}) = \textcolor{red}{1}$ |

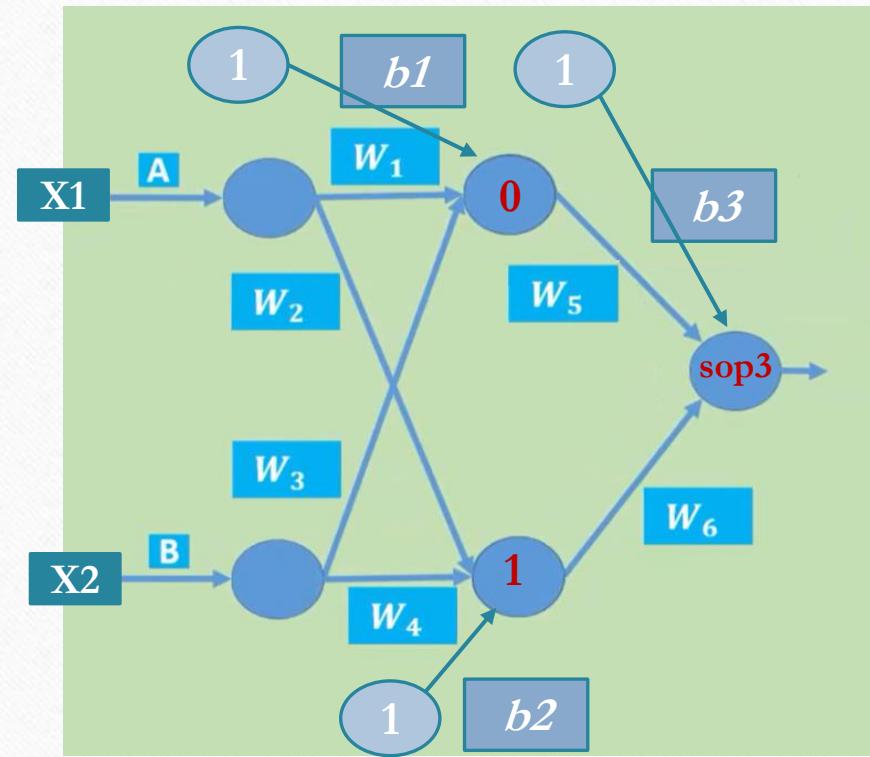


# Exemple de classification: Apprentissage

- l'itération n = 0:

$$\begin{aligned} \text{SOP3} &= (1 * b3 + X1 * W5 + X2 * W6) \\ &= (1 * -0.5) + (0 * -2) + (1 * 1) = \mathbf{0.5} \end{aligned}$$

$$Y = H(\text{SOP3}) = \mathbf{1}$$



# Exemple de classification: Apprentissage

- l'itération  $n = 0$ :

$$\text{SOP3} = (1 * b3 + X1 * W5 + X2 * W6) \\ = (1 * -0.5) + (0 * -2) + (1 * 1) = \mathbf{0.5}$$

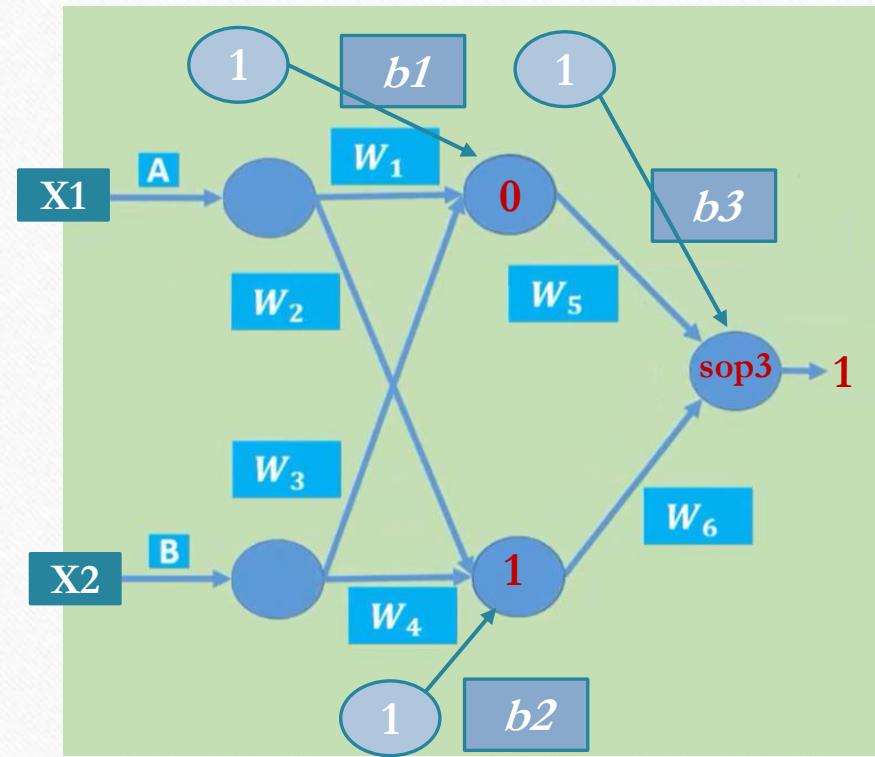
$$Y = H(\text{SOP3}) = \mathbf{1}$$

$$Y(0) = 1$$

$$d(0) = 1$$

Poids corrects

Pas de mise à jour



# Exemple de classification: Apprentissage

- Paramètres à l'itération n = 1:

$$\eta=0,001$$

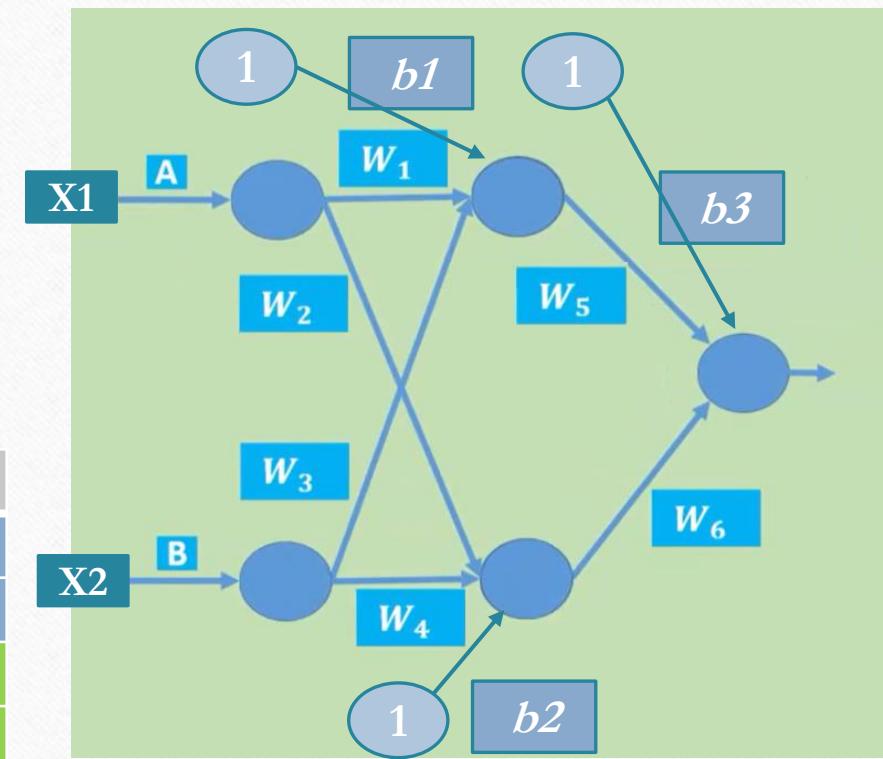
$$X(1)=[1,1,1,X_1,X_2]=[1,1,1, 0,1]$$

$$\begin{aligned}W(1) &= [b1,b2,b3,W_1,W_2,W_3,W_4,W_5,W_6] \\&=[-1.5, -0.5, -0.5, 1,1,1,1,-2, 1]\end{aligned}$$

$$d(1)=1$$

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

Sana Hamdi



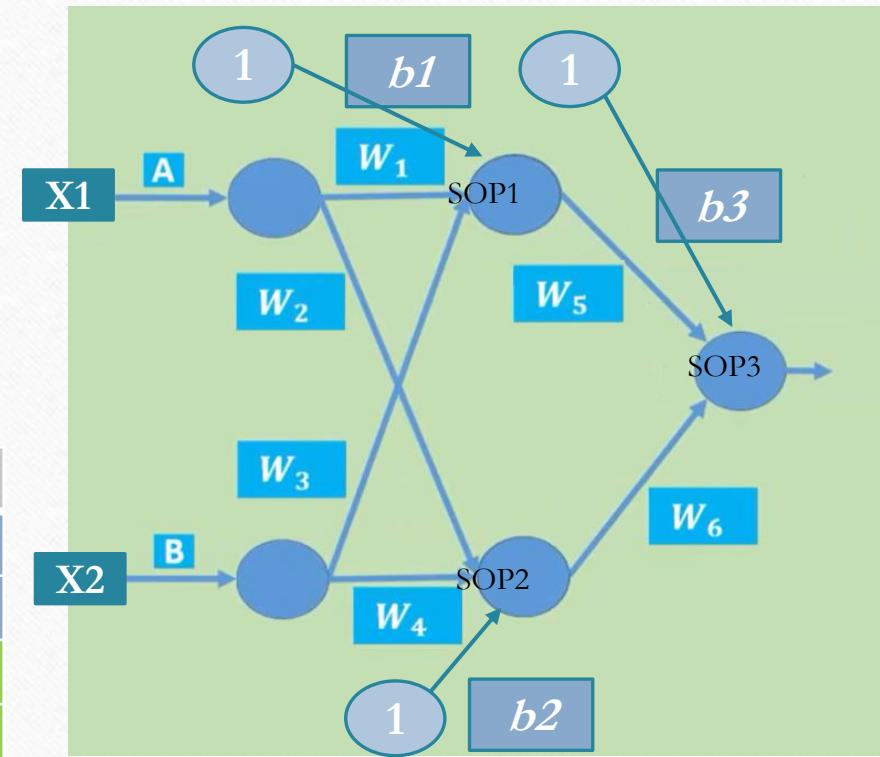
# Exemple de classification: Apprentissage

- l'itération n = 1:

$$\begin{aligned} \text{SOP1} &= (1*b1 + X1*W1 + X2*W3) \\ &= -1.5 + 0 + 1 = \textcolor{red}{-0.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1*b2 + X1*W2 + X2*W4) \\ &= -0.5 + 0 + 1 = \textcolor{red}{0.5} \end{aligned}$$

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |



# Exemple de classification: Apprentissage

- l'itération n = 1:

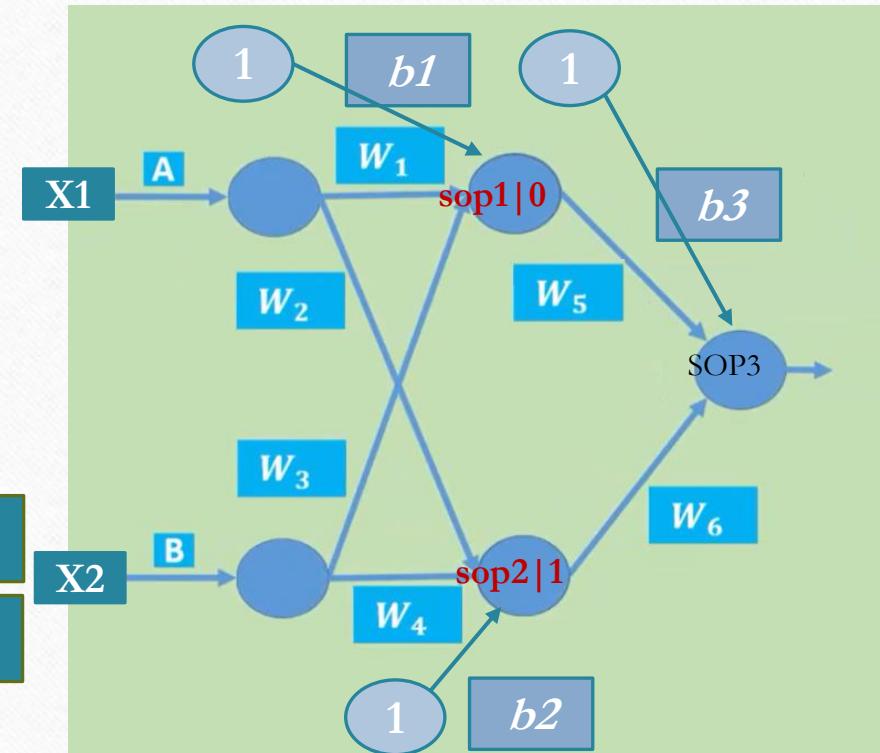
$$\begin{aligned} \text{SOP1} &= (1*b1 + X1*W1 + X2*W3) \\ &= -1.5 + 0 + 1 = \textcolor{red}{-0.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1*b2 + X1*W2 + X2*W4) \\ &= -0.5 + 0 + 1 = \textcolor{red}{0.5} \end{aligned}$$

la fonction de Heaviside

$$H(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0. \end{cases}$$

|                                                              |
|--------------------------------------------------------------|
| $Y_1 = Y(\text{SOP1}) = H(\text{SOP1}) = \textcolor{red}{0}$ |
| $Y_2 = Y(\text{SOP2}) = H(\text{SOP2}) = \textcolor{red}{1}$ |



# Exemple de classification: Apprentissage

- l'itération n = 1:

$$\text{SOP3} = (1 * b3 + X1 * W5 + X2 * W6) \\ = (1 * -0.5) + (0 * -2) + (1 * 1) = \mathbf{0.5}$$

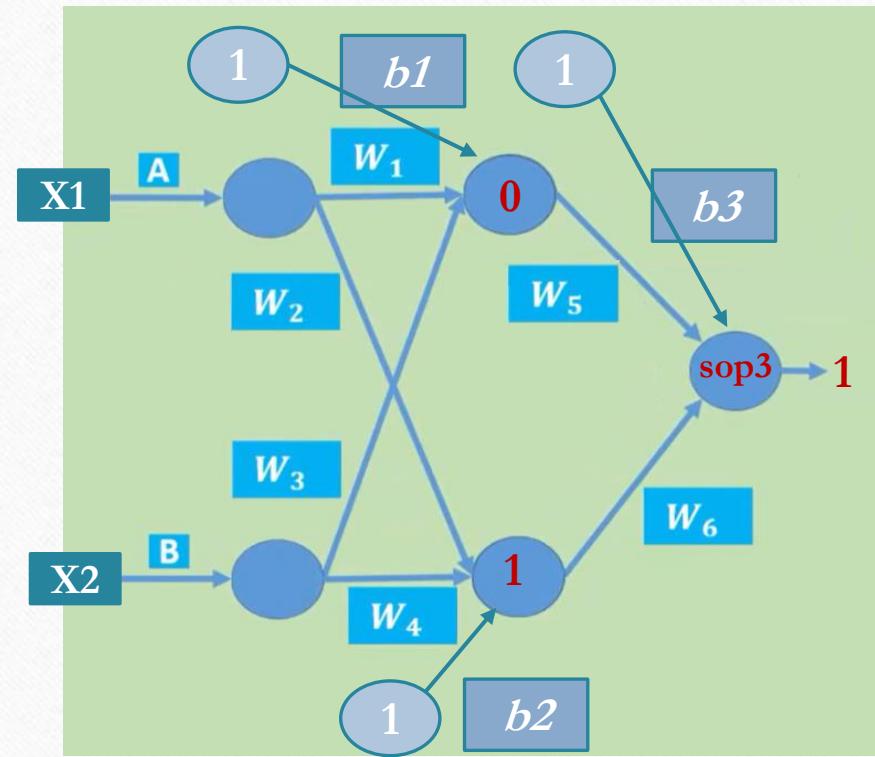
$$Y = H(\text{SOP3}) = \mathbf{1}$$

$$Y(1) = 1$$

$$d(1) = 1$$

Poids corrects

Pas de mise à jour



# Exemple de classification: Apprentissage

- Paramètres à l'itération  $n = 2$ :

$$\eta=0,001$$

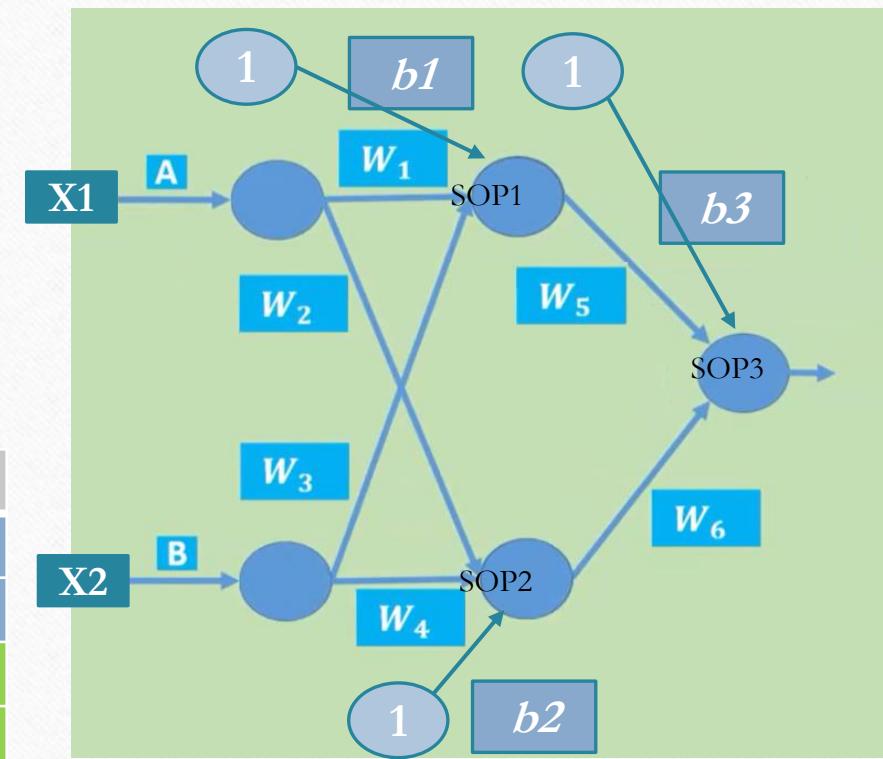
$$X(2)=[1,1,1,X_1,X_2]=[1,1,1, 0,0]$$

$$\begin{aligned}W(2) &= [b1,b2,b3,W_1,W_2,W_3,W_4,W_5,W_6] \\&=[-1.5, -0.5, -0.5, 1,1,1,1,-2, 1]\end{aligned}$$

$$d(2)=0$$

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

Sana Hamdi



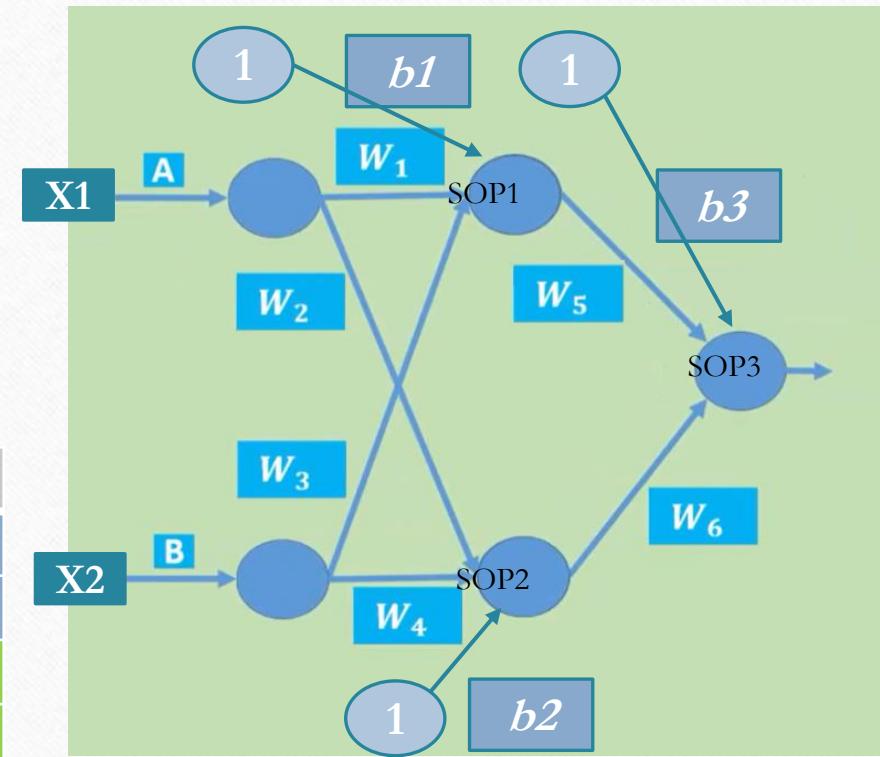
# Exemple de classification: Apprentissage

- l'itération n = 2:

$$\begin{aligned} \text{SOP1} &= (1*b1 + X1*W1 + X2*W3) \\ &= -1.5 + 0 + 0 = \textcolor{red}{-1.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1*b2 + X1*W2 + X2*W4) \\ &= -0.5 + 0 + 0 = \textcolor{red}{-0.5} \end{aligned}$$

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |



# Exemple de classification: Apprentissage

- l'itération n = 2:

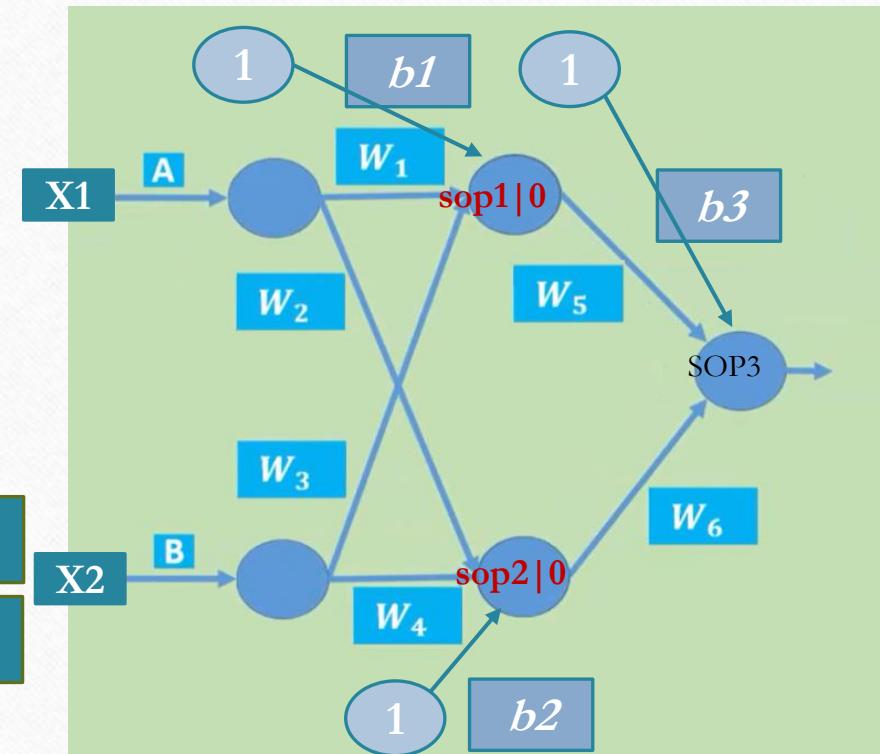
$$\begin{aligned} \text{SOP1} &= (1*b1 + X1*W1 + X2*W3) \\ &= -1.5 + 0 + 0 = \textcolor{red}{-1.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1*b2 + X1*W2 + X2*W4) \\ &= -0.5 + 0 + 0 = \textcolor{red}{-0.5} \end{aligned}$$

la fonction de Heaviside

$$H(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0. \end{cases}$$

|                                                              |
|--------------------------------------------------------------|
| $Y_1 = Y(\text{SOP1}) = H(\text{SOP1}) = \textcolor{red}{0}$ |
| $Y_2 = Y(\text{SOP2}) = H(\text{SOP2}) = \textcolor{red}{0}$ |



# Exemple de classification: Apprentissage

- l'itération n = 2:

$$\text{SOP3} = (1 \cdot b3 + X1 \cdot W5 + X2 \cdot W6) \\ = (1 \cdot -0.5) + (0 \cdot -2) + (0 \cdot 1) = \mathbf{-0.5}$$

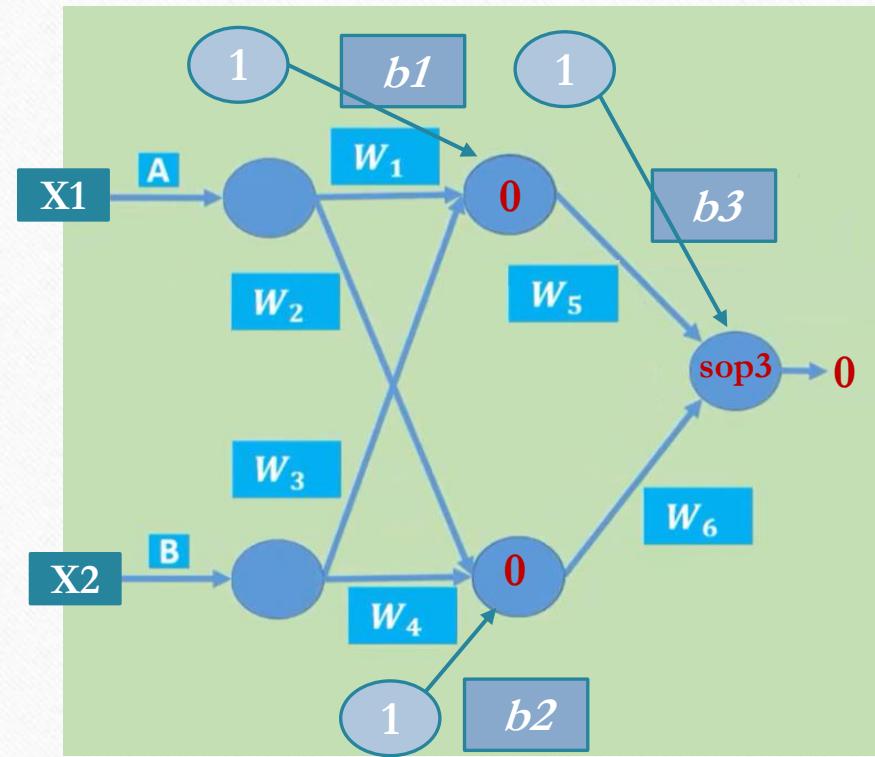
$$Y = H(\text{SOP3}) = \mathbf{0}$$

$$Y(2) = 0$$

$$d(2) = 0$$

Poids corrects

Pas de mise à jour



# Exemple de classification: Apprentissage

- Paramètres à l'itération n = 3:

$$\eta=0,001$$

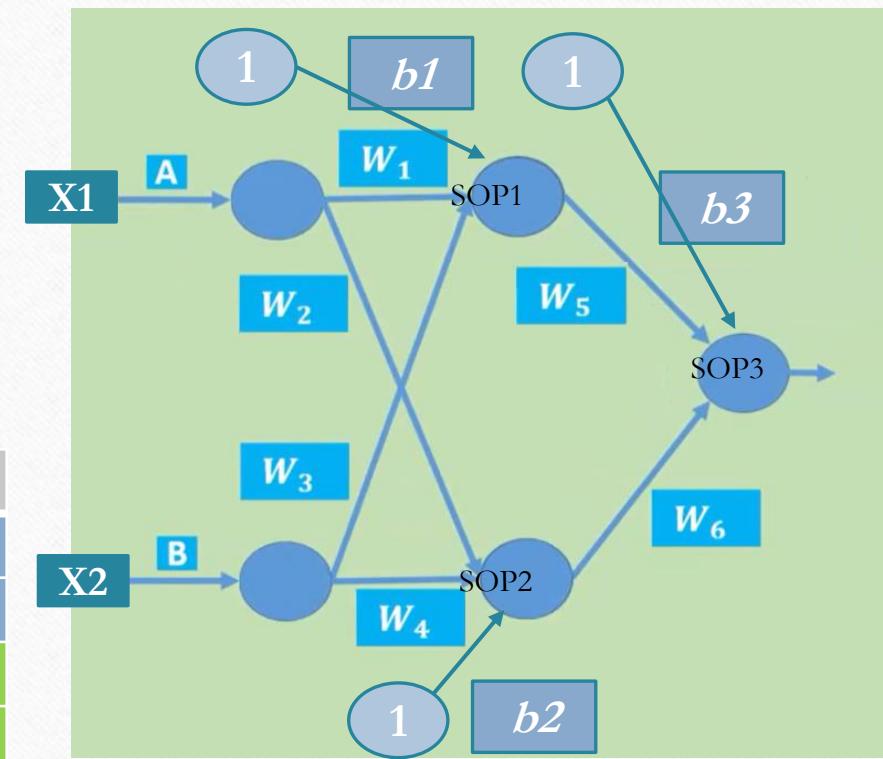
$$X(3)=[1,1,1,X_1,X_2]=[1,1,1, 1,1]$$

$$\begin{aligned}W(3) &= [b1,b2,b3,W_1,W_2,W_3,W_4,W_5,W_6] \\&=[-1.5, -0.5, -0.5, 1,1,1,1,-2, 1]\end{aligned}$$

$$d(3)=0$$

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |

Sana Hamdi



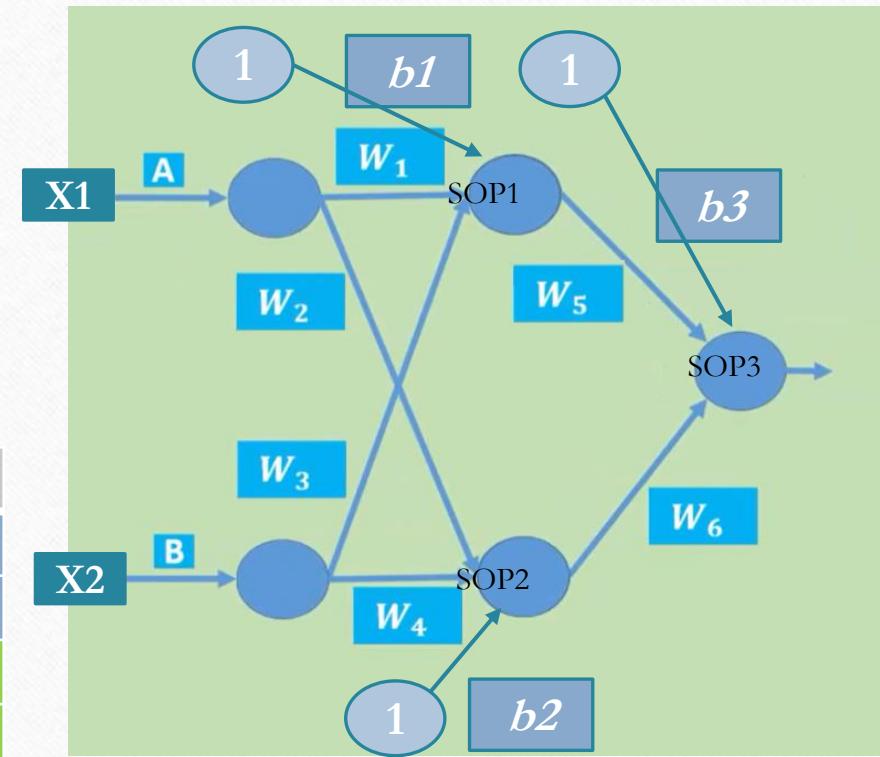
# Exemple de classification: Apprentissage

- l'itération n = 3:

$$\begin{aligned} \text{SOP1} &= (1*b1 + X1*W1 + X2*W3) \\ &= -1.5 + 1 + 1 = \textcolor{red}{0.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1*b2 + X1*W2 + X2*W4) \\ &= -0.5 + 1 + 1 = \textcolor{red}{1.5} \end{aligned}$$

| A | B | Classe |
|---|---|--------|
| 1 | 0 | 1      |
| 0 | 1 | 1      |
| 0 | 0 | 0      |
| 1 | 1 | 0      |



# Exemple de classification: Apprentissage

- l'itération n = 3:

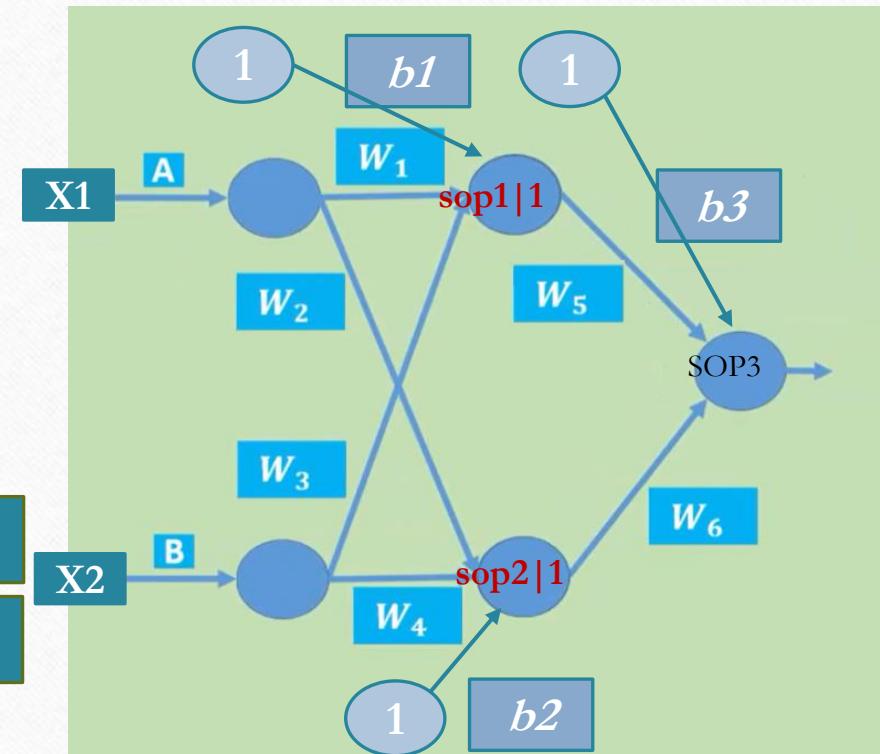
$$\begin{aligned} \text{SOP1} &= (1*b1 + X1*W1 + X2*W3) \\ &= -1.5 + 1 + 1 = \textcolor{red}{0.5} \end{aligned}$$

$$\begin{aligned} \text{SOP2} &= (1*b2 + X1*W2 + X2*W4) \\ &= -0.5 + 1 + 1 = \textcolor{red}{1.5} \end{aligned}$$

la fonction de Heaviside

$$H(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0. \end{cases}$$

|                                                       |
|-------------------------------------------------------|
| $Y1=Y(\text{SOP1})=H(\text{SOP1})=\textcolor{red}{1}$ |
| $Y2=Y(\text{SOP2})=H(\text{SOP2})=\textcolor{red}{1}$ |



# Exemple de classification: Apprentissage

- l'itération n = 3:

$$\begin{aligned} \text{SOP3} &= (1*b3 + X1*W5 + X2*W6) \\ &= (1*-0.5) + (1*-2) + (1*1) = \mathbf{-1.5} \end{aligned}$$

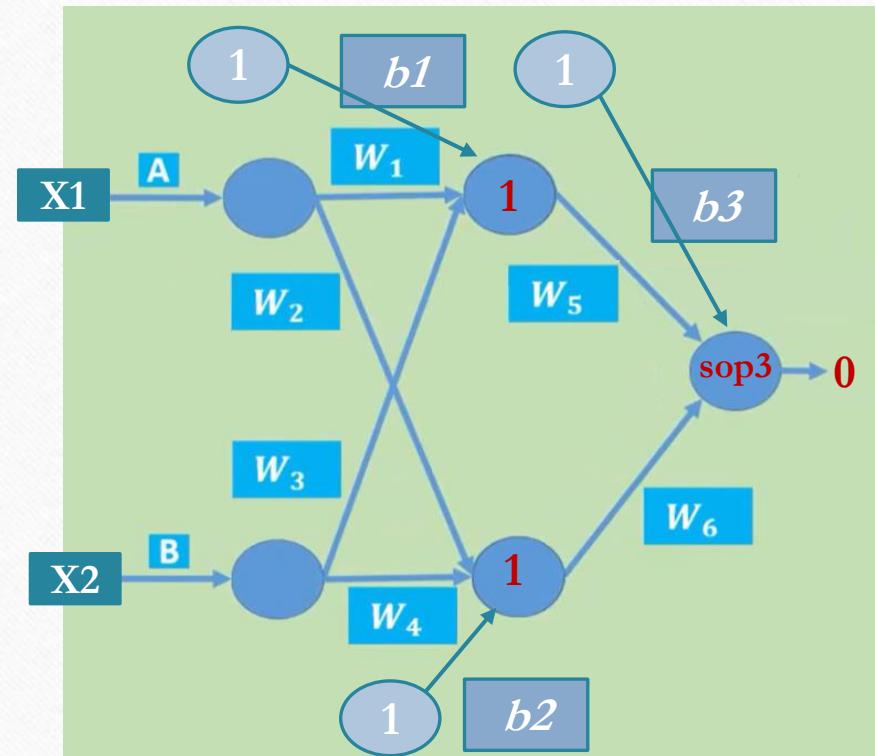
$$Y = H(\text{SOP3}) = \mathbf{0}$$

$$Y(3) = 0$$

$$d(3) = 0$$

Poids corrects

Pas de mise à jour



# Exemple de classification: Apprentissage

---

- Les poids considérés par ce modèle sont:

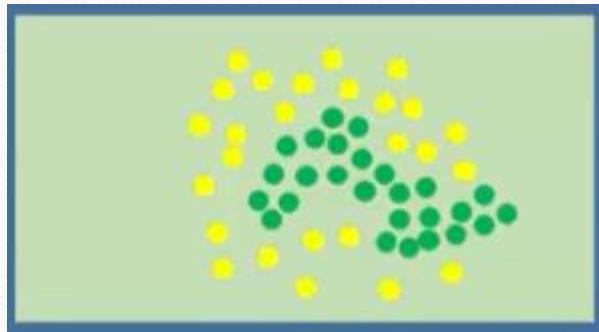
$$\mathbf{W} = [-1.5, -0.5, -0.5, 1, 1, 1, 1, -2, 1]$$

# Réseaux de neurones profonds

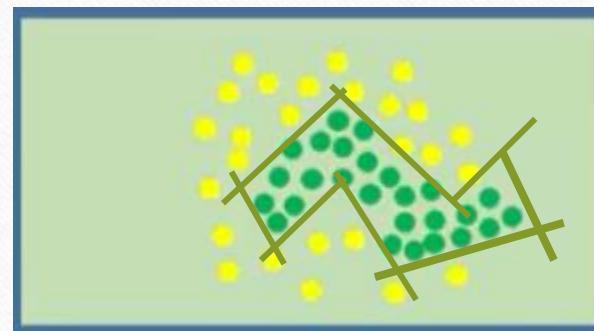
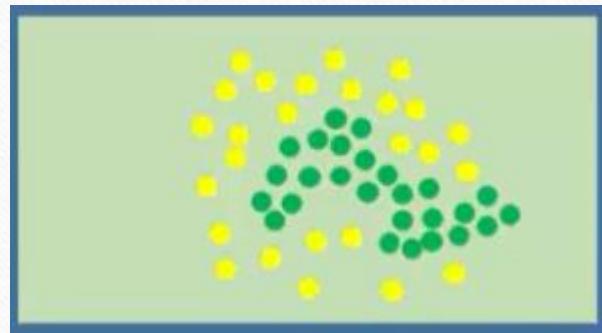


# Fonctions Complexes

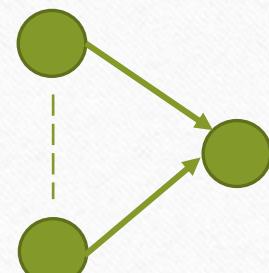
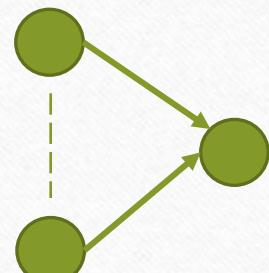
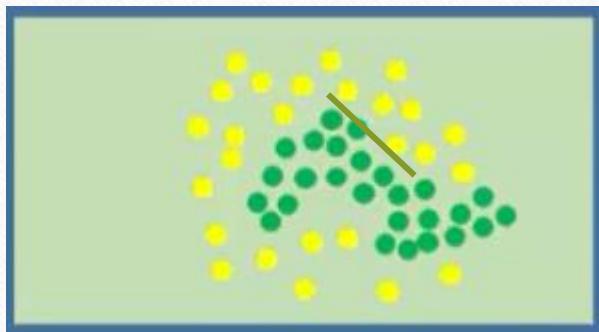
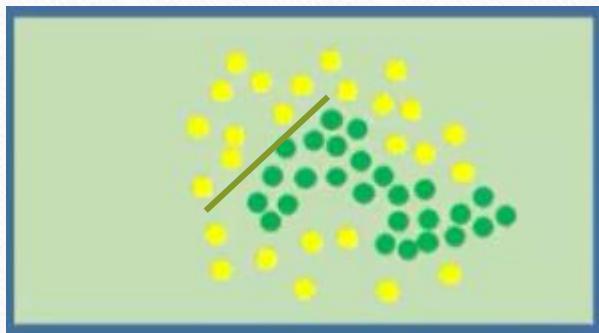
---



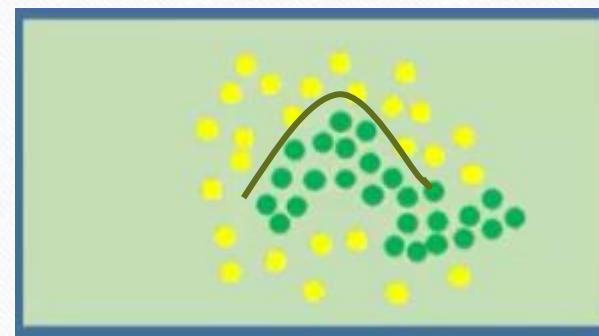
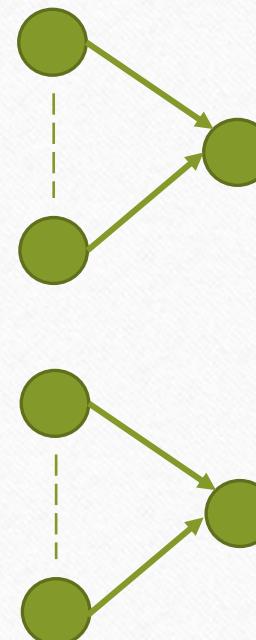
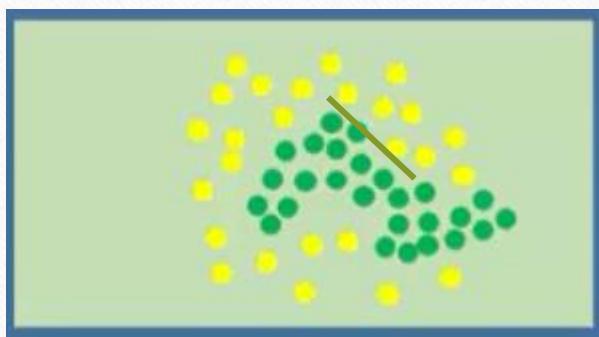
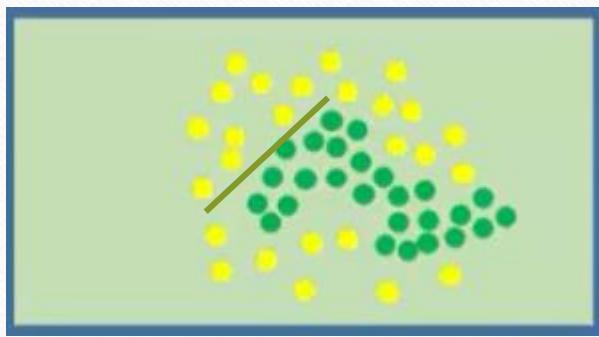
# Fonctions Complexes



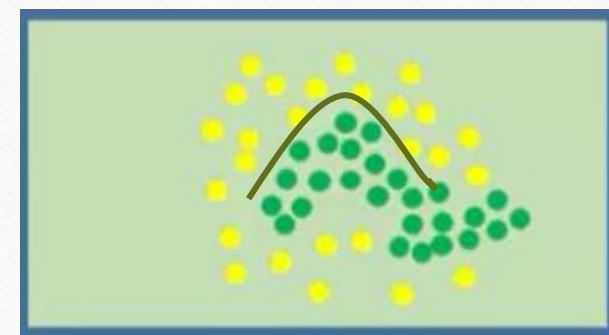
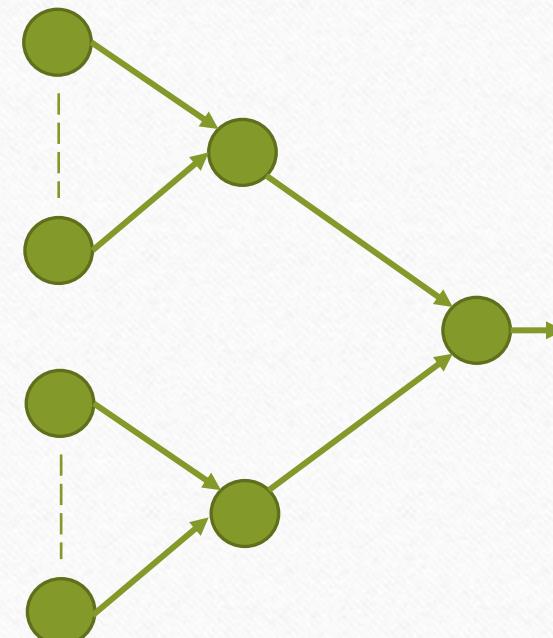
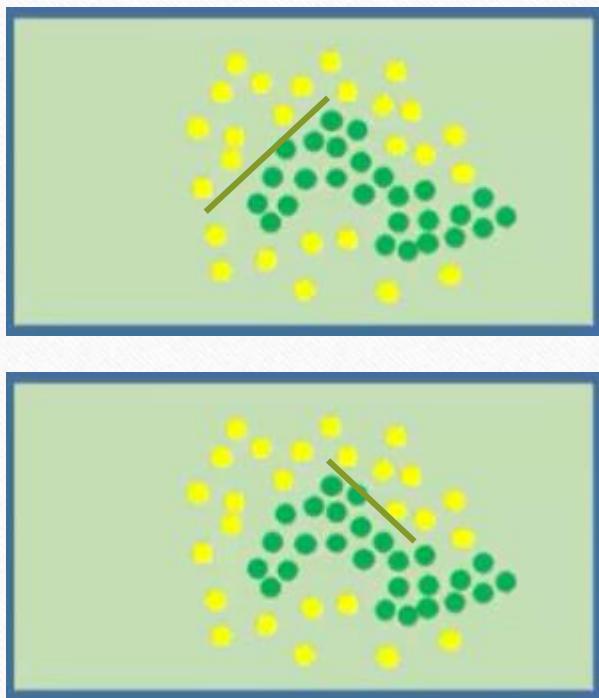
# Fonctions Complexes



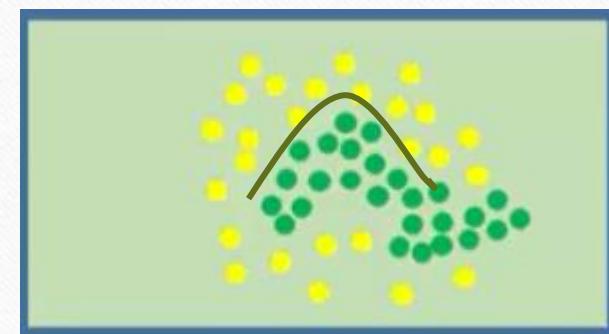
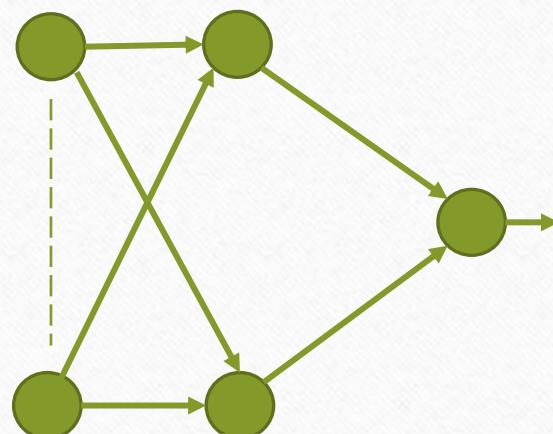
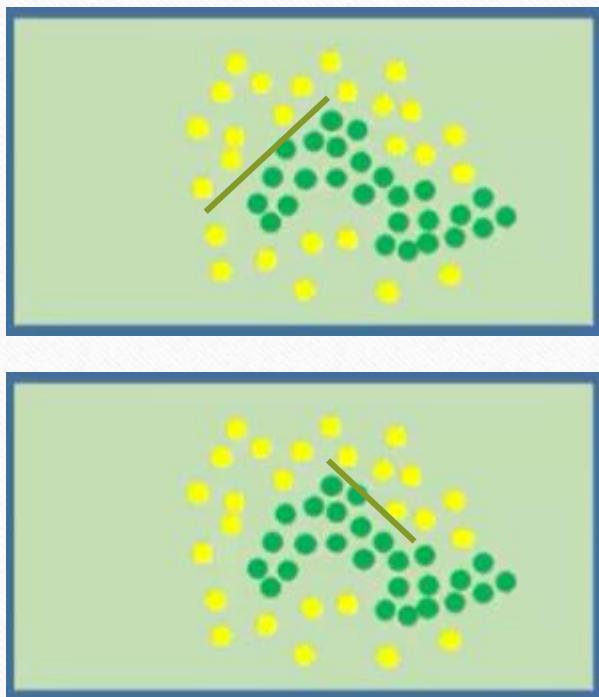
# Fonctions Complexes



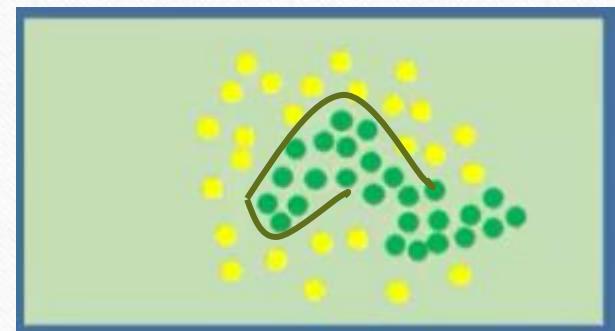
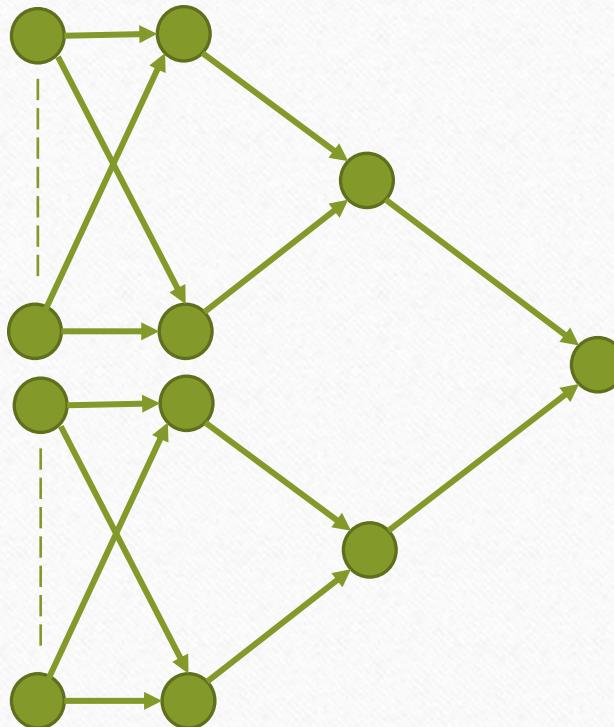
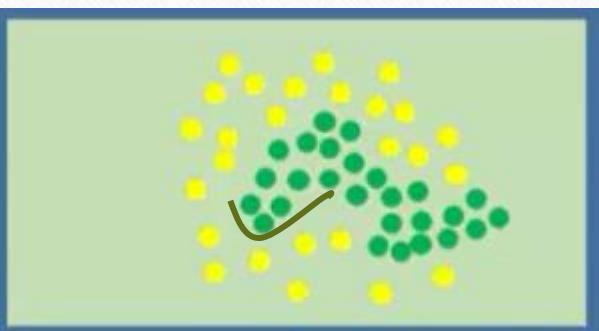
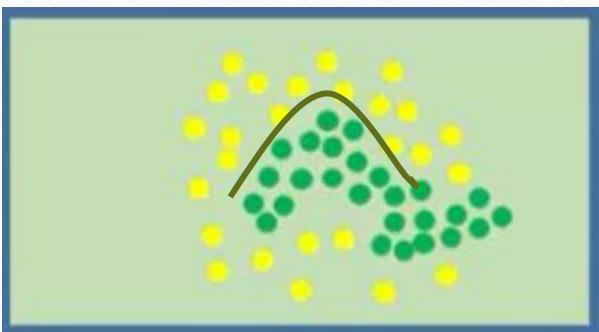
# Fonctions Complexes



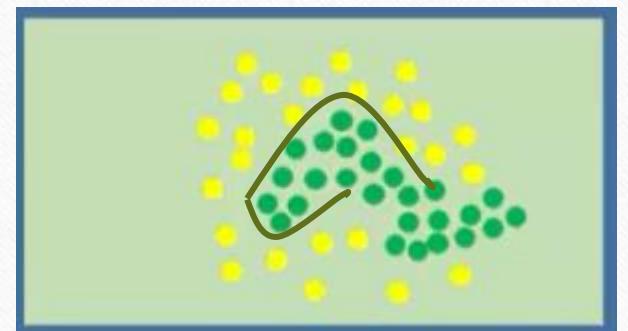
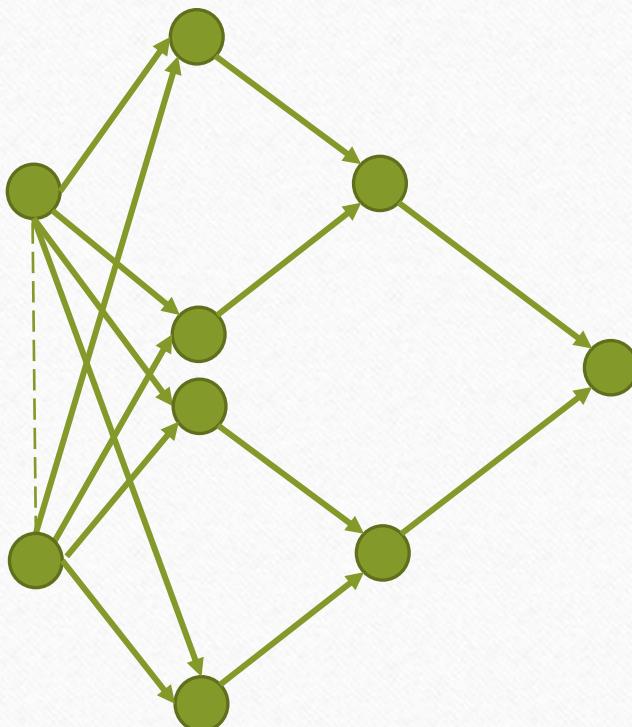
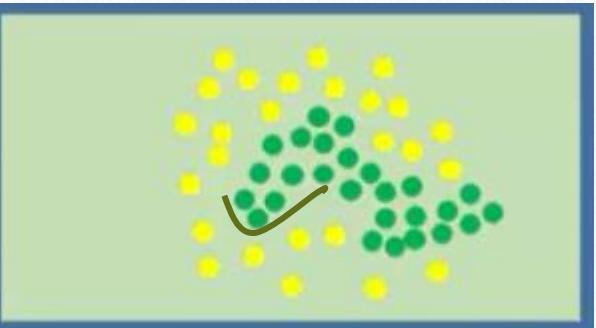
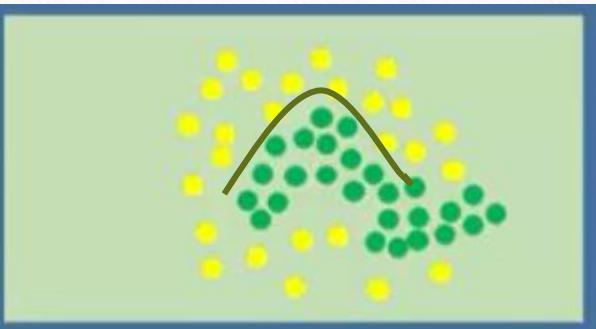
# Fonctions Complexes



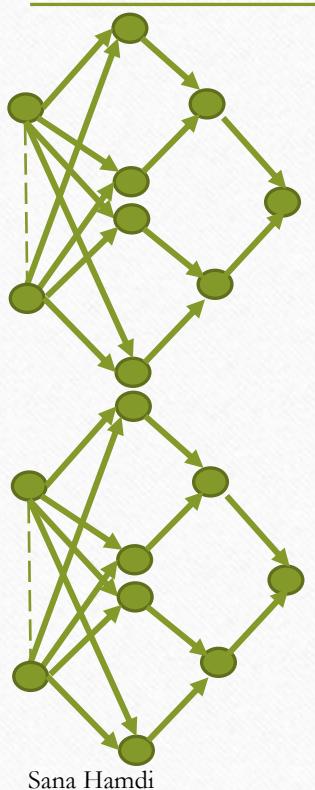
# Fonctions Complexes



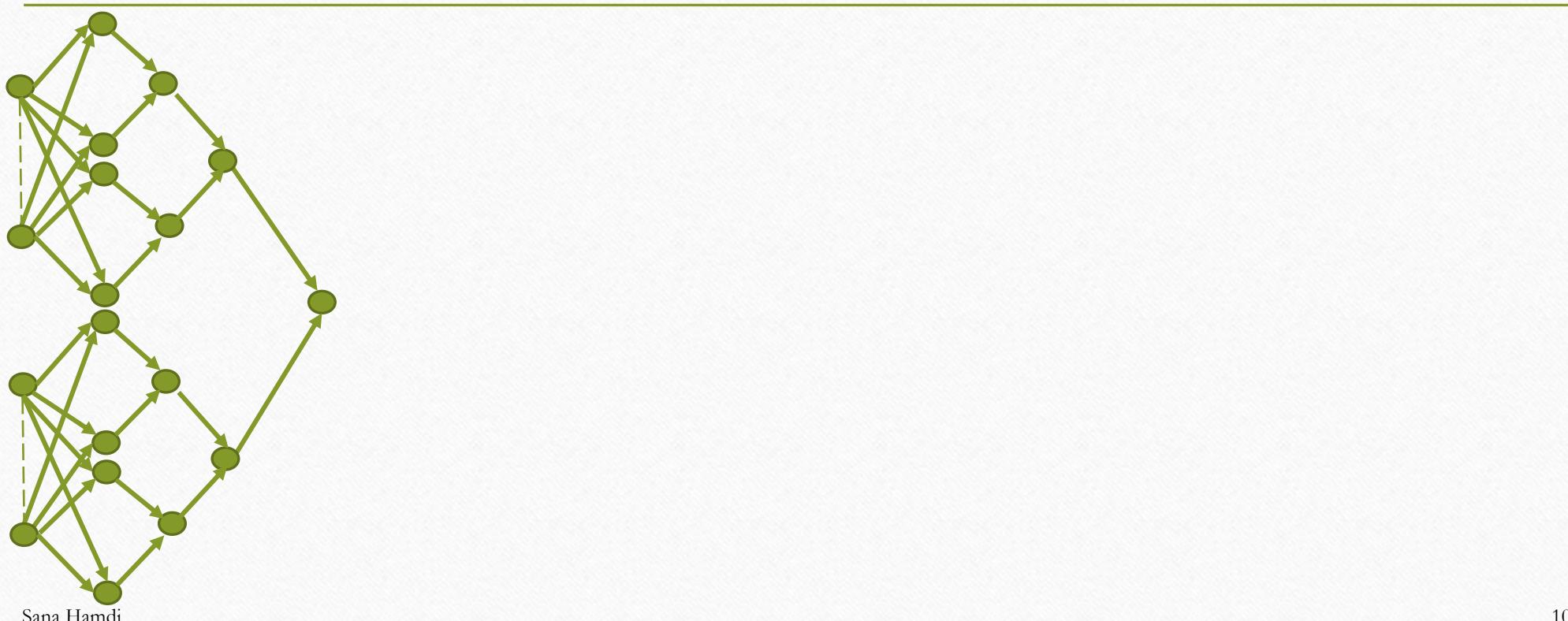
# Fonctions Complexes



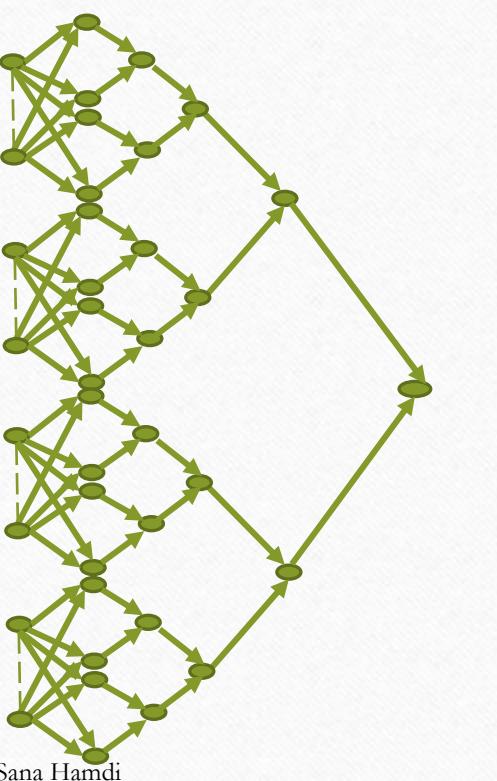
# Réseaux de neurones profonds



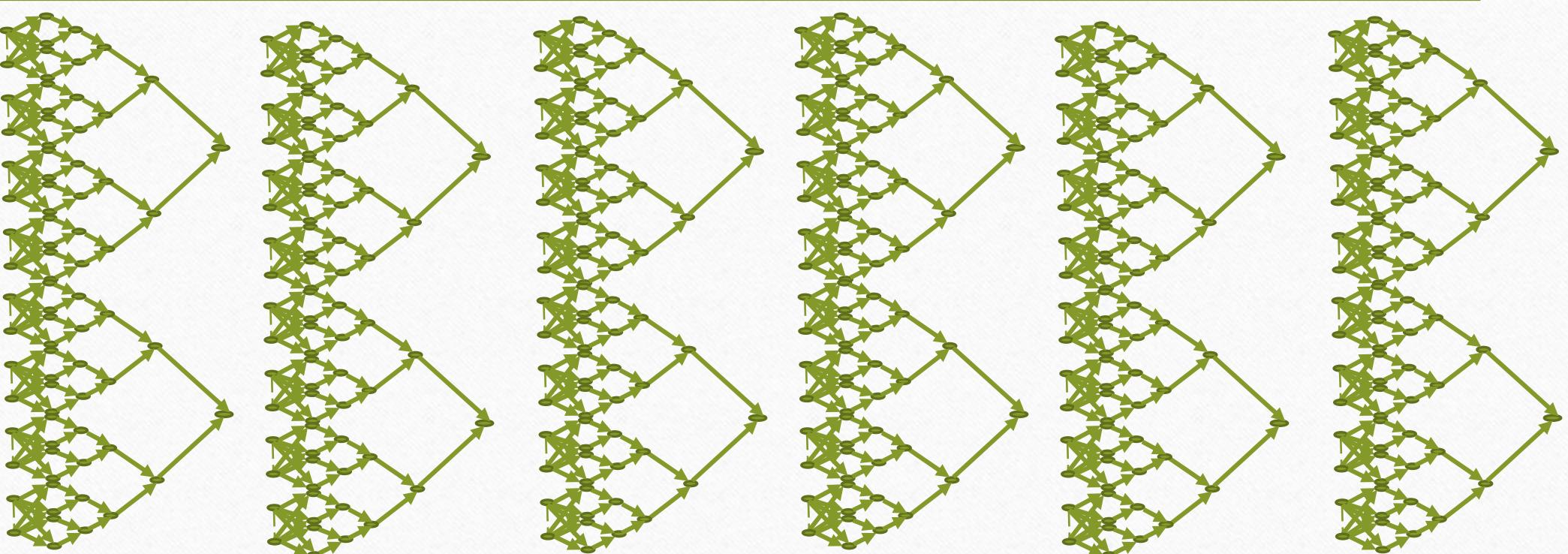
# Réseaux de neurones profonds



# Réseaux de neurones profonds



# Réseaux de neurones profonds



# La rétro-propagation



# Apprendre puis mettre à jour

- L'algorithme de rétro-propagation est utilisé pour mettre à jour les poids quand ils ne sont pas capables de faire les prédictions correctes.
- Nous devons apprendre le modèle avant la rétro-propagation.

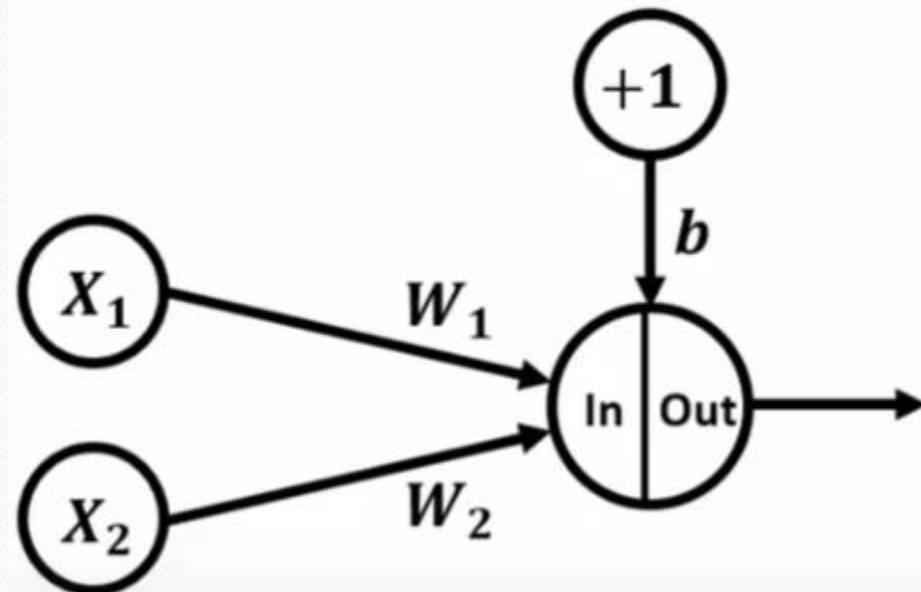


# Exemple d'apprentissage

| X1  | X2  | Output |
|-----|-----|--------|
| 0.1 | 0.3 | 0.03   |

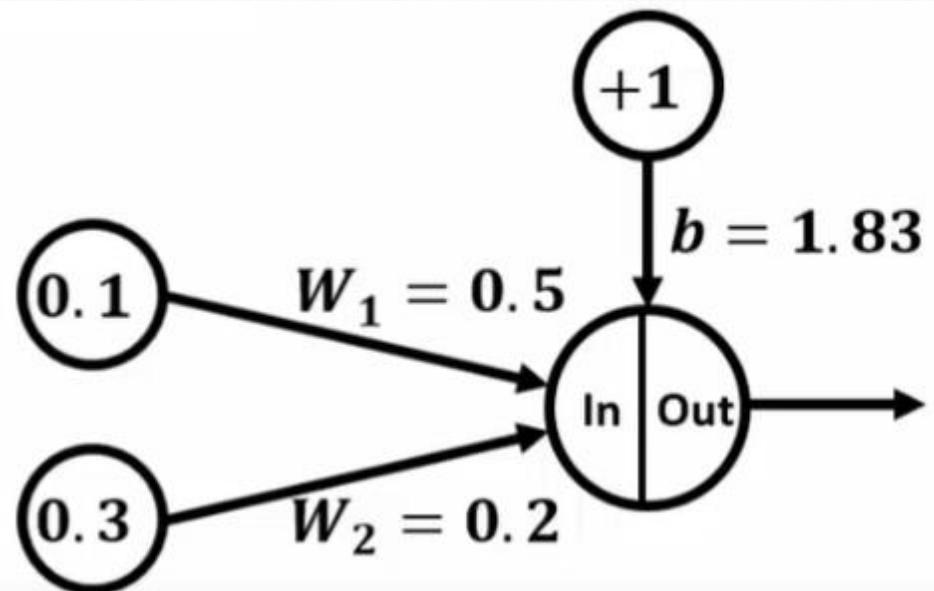
- Les poids initiaux sont:

| W1  | W2  | b    |
|-----|-----|------|
| 0.5 | 0.2 | 1.83 |



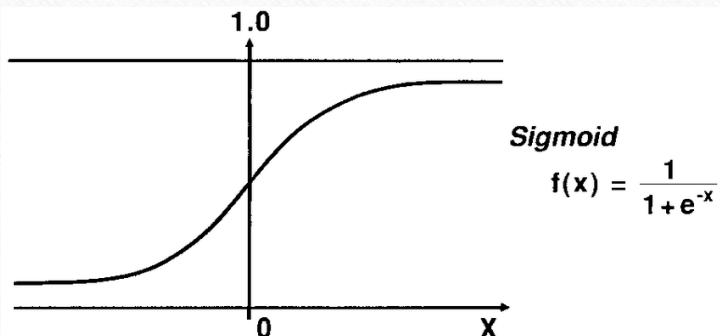
# Exemple d'apprentissage: Somme des produits

- SOP = S  
 $= X_1 * W_1 + X_2 * W_2 + 1 * b$   
 $= 0.1 * 0.5 + 0.3 * 0.2 + 1.83$   
 $= \textcolor{red}{1.94}$



# Exemple d'apprentissage: Fonction d'activation

- Nous allons utiliser la fonction Sigmoïde comme fonction d'activation:



- En se basant sur le calcul de la SOP, l'output est:

$$\text{Output} = \frac{1}{1+e^{-1.94}} = \frac{1}{1+0.144} = \mathbf{0.874}$$

# Fonction d'erreur

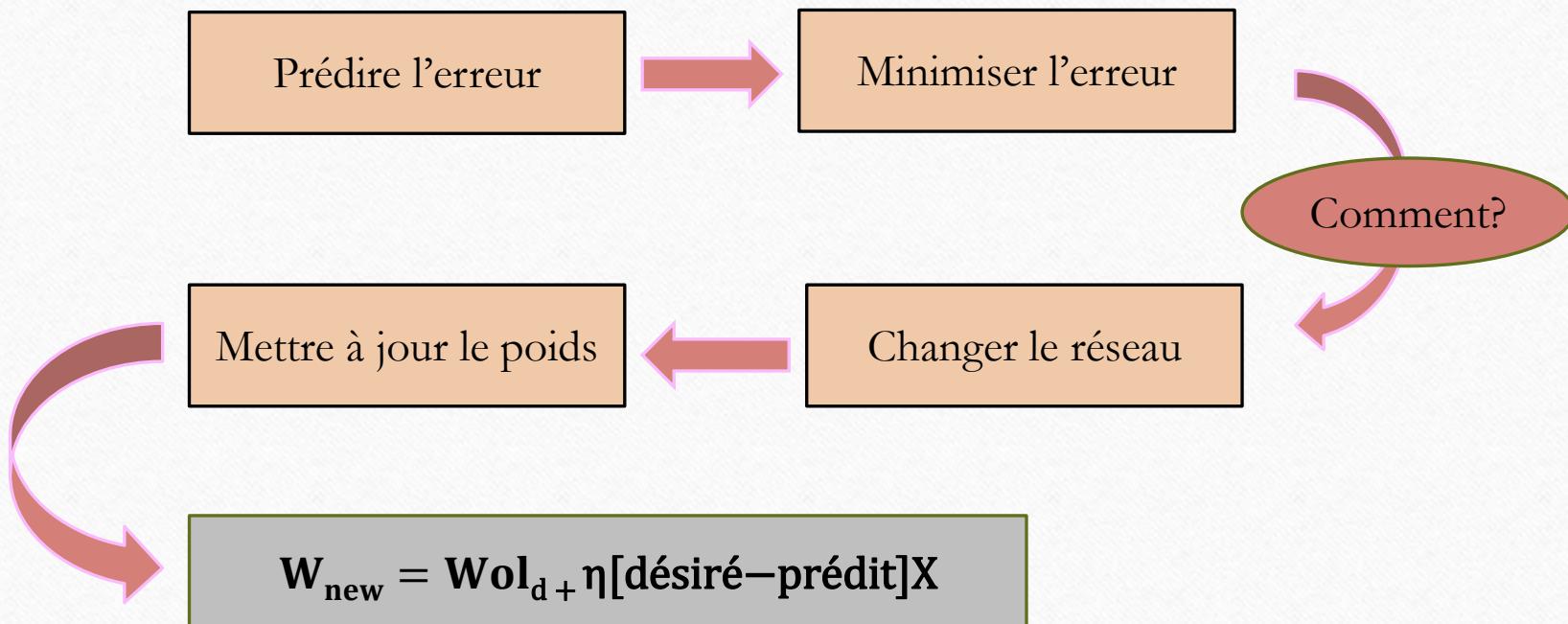
---

- Nous pouvons utiliser la fonction d'erreur quadratique moyenne:

$$E = \frac{1}{2} (Désiré - prédit)^2$$

- $E = \frac{1}{2} (0.03 - 0.874)^2 = \frac{1}{2} * 0.713 = \mathbf{0.357}$

# Fonction d'erreur



# Pourquoi la rétro-propagation?

---



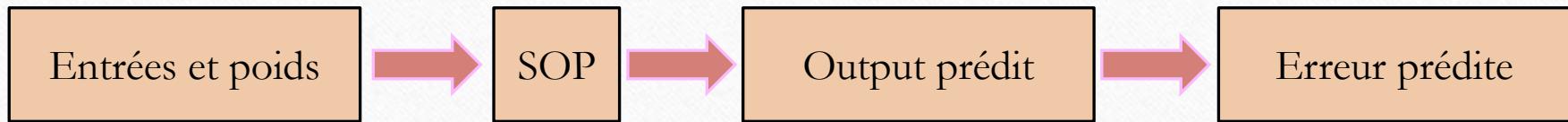
- Pourquoi les nouveaux poids sont meilleurs que les anciens poids?
- Quelle est l'effet de chaque poids sur la fonction d'erreur?
- Comment augmenter ou diminuer les valeurs des poids affecte la fonction d'erreur?

***La rétropropagation répond à toutes ces questions***

# Forward VS Backward

---

- **Forward**



- **Backward**



# Les dérivés

---

- Soit l'exemple simple suivant:  $Y=ax^2 + b$
- Comment répondre à cette question: **Quel est le changement de Y par rapport à un petit changement de x?**
- La réponse c'est l'utilisation des dérivés:  $\frac{dy}{dx} \rightarrow$  l'effet d'un petit changement de x sur Y.

# Fonction d'erreur et dérivés

---

- Changement de l'erreur par rapport au changement des poids :  $\frac{dE}{dW} = \nabla E(w_1, w_2)$

$$E = \frac{1}{2} (Désiré - prédit)^2 = \frac{1}{2} \left( Désiré - \frac{1}{1 + e^{-S}} \right)^2$$

$$E = \frac{1}{2} \left( Désiré - \frac{1}{1 + e^{-(x_1 * w_1 + x_2 * w_2 + b)}} \right)^2$$

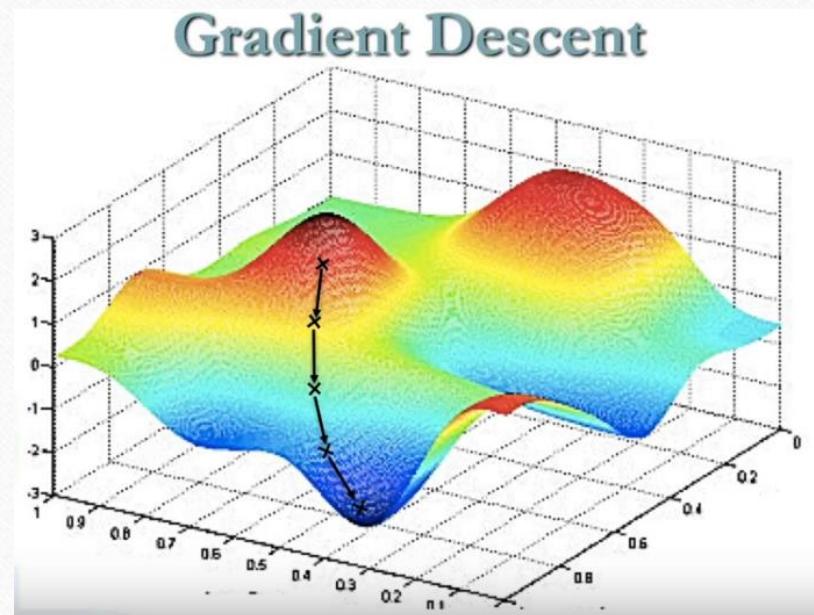
# Descente de gradient

- Objectif: tendre  $\frac{dE}{dW}$  vers 0:  $\nabla E(w_1, w_2) = \nabla E(\vec{w}) = \vec{0}$
- Etapes: 0. Initialiser  $\vec{w}$

1. Calculer  $\nabla E(w_1, w_2) = \begin{bmatrix} \frac{dE}{dw_1} \\ \frac{dE}{dw_2} \end{bmatrix}$

2. Mettre à jour  $w_{i\text{new}} = w_{i\text{old}} - \eta \frac{dE}{dw_{i\text{old}}}$

- Jusqu'à  $\nabla E(\vec{w}) = \vec{0}$



# Fonction d'erreur et dérivés



$$E = \frac{1}{2} (Désiré - prédict)^2$$

$$\frac{1}{1+e^{-s}}$$

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

w<sub>1</sub> et w<sub>2</sub>

$$\frac{dE}{dw} = \frac{d}{dw} \left( \frac{1}{2} (Désiré - \frac{1}{1+e^{-x_1 * w_1 + x_2 * w_2 + b}})^2 \right) \rightarrow \text{Calcul Compliqué} \rightarrow \text{Règle de dérivation en chaîne}$$

# Règle de dérivation en chaîne



$$E = \frac{1}{2} (Désiré - prédict)^2$$

$$\frac{1}{1+e^{-s}}$$

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

w<sub>1</sub> et w<sub>2</sub>

$$\frac{dE}{dw_1} \frac{dE}{dw_2} =$$

$$\frac{dE}{d\text{prédict}}$$

\*

$$\frac{d\text{prédict}}{ds}$$

\*

$$\frac{ds}{dW_1} \frac{ds}{dW_2}$$

# Règle de dérivation en chaîne

---

$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial \text{Predicted}} * \frac{\partial \text{Predicted}}{\partial s} * \frac{\partial s}{\partial W_1}$$

$$\frac{\partial E}{\partial W_2} = \frac{\partial E}{\partial \text{Predicted}} * \frac{\partial \text{Predicted}}{\partial s} * \frac{\partial s}{\partial W_2}$$

## Règle de dérivation en chaîne

$$\frac{\partial E}{\partial \text{Predicted}} = \frac{\partial}{\partial \text{Predicted}} \left( \frac{1}{2} (\text{desired} - \text{predicted})^2 \right) \\ = -(\text{desired} - \text{predicted})$$

$$\frac{\partial E}{\partial \text{Predicted}} = \text{predicted} - \text{desired}$$

$$\frac{\partial E}{\partial \text{Predicted}} = \text{predicted} - \text{desired} = 0.874 - 0.03$$

$$\frac{\partial E}{\partial \text{Predicted}} = 0.844$$

# Règle de dérivation en chaîne

$$\frac{\partial \text{Predicted}}{\partial s} = \frac{\partial}{\partial s} \left( \frac{1}{1 + e^{-s}} \right)$$

$$\frac{\partial \text{Predicted}}{\partial s} = \frac{1}{1 + e^{-s}} \left( 1 - \frac{1}{1 + e^{-s}} \right)$$

$$\begin{aligned}\frac{\partial \text{Predicted}}{\partial s} &= \frac{1}{1 + e^{-s}} \left( 1 - \frac{1}{1 + e^{-s}} \right) = \frac{1}{1 + e^{-1.94}} \left( 1 - \frac{1}{1 + e^{-1.94}} \right) \\&= \frac{1}{1 + 0.144} \left( 1 - \frac{1}{1 + 0.144} \right) \\&= \frac{1}{1.144} \left( 1 - \frac{1}{1.144} \right) \\&= 0.874 \left( 1 - 0.874 \right) \\&= 0.874(0.126)\end{aligned}$$

$$\frac{\partial \text{Predicted}}{\partial s} = 0.11$$

# Règle de dérivation en chaîne

---

$$\begin{aligned}\frac{\partial s}{\partial W_1} &= \frac{\partial}{\partial W_1} (X_1 * W_1 + X_2 * W_2 + b) \\&= \mathbf{1} * X_1 * (W_1)^{(\mathbf{1}-\mathbf{1})} + \mathbf{0} + \mathbf{0} \\&= X_1 * (W_1)^{(\mathbf{0})} \\&= X_1(\mathbf{1})\end{aligned}$$

$$\frac{\partial s}{\partial W_1} = X_1$$

$$\frac{\partial s}{\partial W_1} = \mathbf{0.1}$$

# Règle de dérivation en chaîne

---

$$\frac{\partial s}{\partial W_2} = \frac{\partial}{\partial W_2} (X_1 * W_1 + X_2 * W_2 + b)$$

$$= \mathbf{0} + \mathbf{1} * X_2 * (W_2)^{(1-1)} + \mathbf{0}$$

$$= X_2 * (W_2)^{(0)}$$

$$= X_2(\mathbf{1})$$

$$\frac{\partial s}{\partial W_2} = X_2$$

$$\frac{\partial s}{\partial W_2} = X_2$$

$$\frac{\partial s}{\partial W_2} = \mathbf{0.3}$$

# Dérivés partielles

- Nous pouvons calculer maintenant les dérivées partielles de E par rapport à  $w_1$  et  $w_2$  :

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial Predicted} * \frac{\partial Predicted}{\partial s} * \frac{\partial s}{\partial w_1}$$
$$\frac{\partial E}{\partial w_1} = 0.844 * 0.11 * 0.1$$

$$\frac{\partial E}{\partial w_1} = 0.01$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial Predicted} * \frac{\partial Predicted}{\partial s} * \frac{\partial s}{\partial w_2}$$
$$\frac{\partial E}{\partial w_2} = 0.844 * 0.11 * 0.3$$

$$\frac{\partial E}{\partial w_2} = 0.028$$

# Interprétation

- Deux informations importantes à partir des dérivées calculées:  $\frac{\partial E}{\partial W_1} = 0.01$      $\frac{\partial E}{\partial W_2} = 0.03$

## Derivative Sign

Increasing/decreasing weight increases/decreases error.

Increasing/decreasing weight decreases/increases error.

## Derivative Magnitude

Positive

Increasing/decreasing weight by P increases/decreases error by P\*MAG.

Negative

Increasing/decreasing weight by P decreases/increases error by P\*MAG.

- Dans notre exemple, les dérivées sont positives, donc pour diminuer l'erreur, on diminue les poids.

# Mise à jour des poids

- +0,,doivent être mis à jour comme suit:  $w_{i\text{new}} = w_{i\text{old}} - \eta \frac{dE}{dw_{i\text{old}}}$

Updating  $W_1$

$$W_{1\text{new}} = W_1 - \eta * \frac{\partial E}{\partial W_1}$$
$$= 0.5 - 0.01 * 0.01$$

$$\color{red}W_{1\text{new}} = 0.49991$$

Updating  $W_2$

$$W_{2\text{new}} = W_2 - \eta * \frac{\partial E}{\partial W_2}$$
$$= 0.2 - 0.01 * 0.028$$

$$\color{red}W_{2\text{new}} = 0.1997$$

- Faire la même chose pour b et le mettre à jour,
- Continuer la mise à jour de b selon les dérivées et réapprendre le modèle jusqu'au avoir une erreur acceptable (qui tend vers 0)

# La rétro-propagation avec des couches cachées

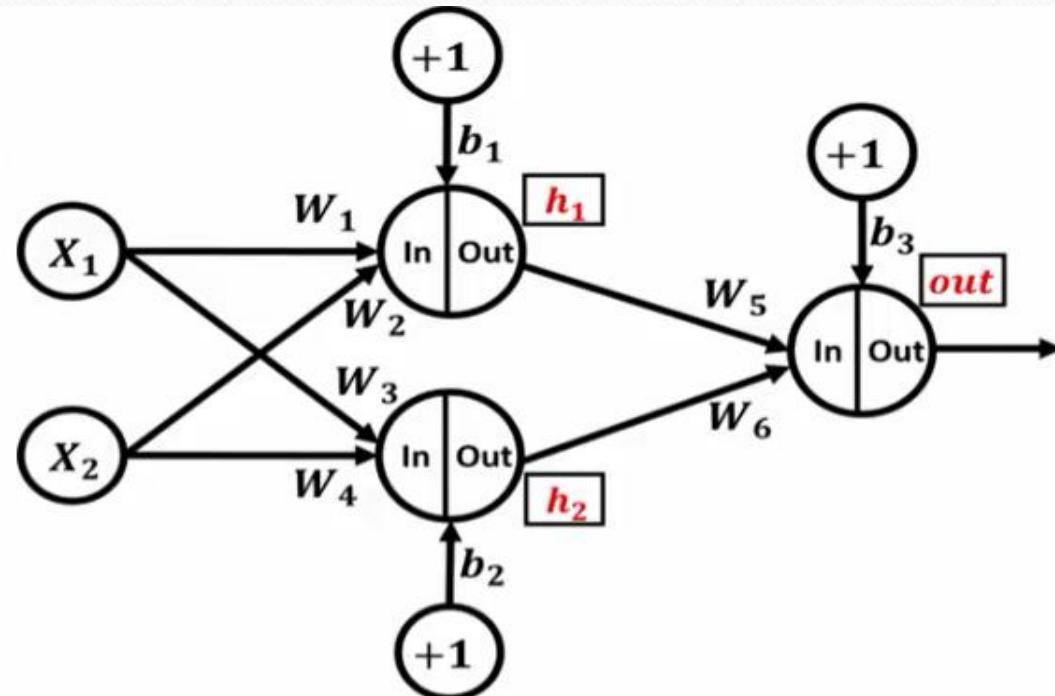


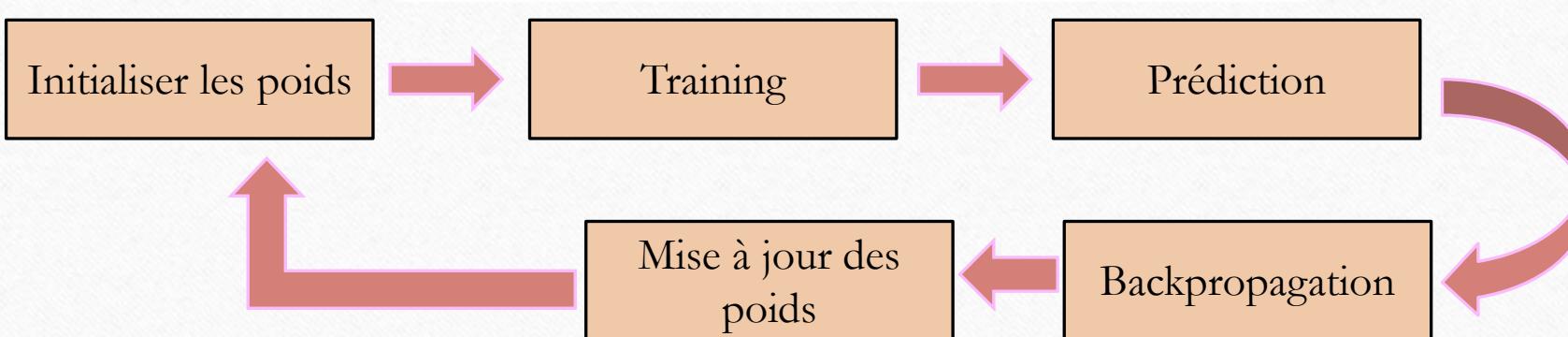
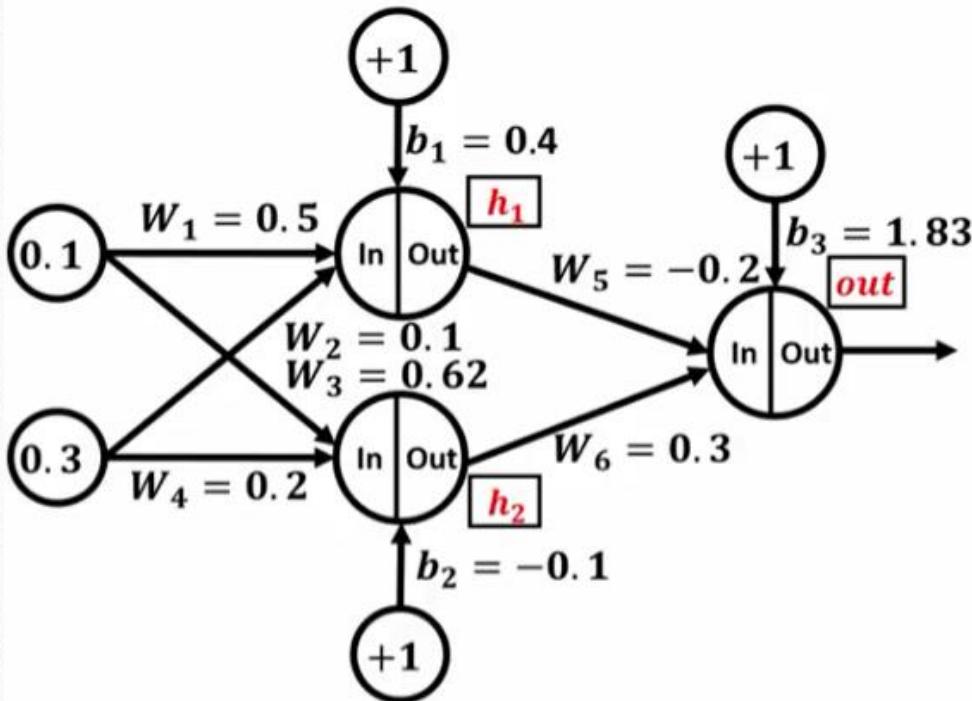
## Training Data

| $X_1$ | $X_2$ | Output |
|-------|-------|--------|
| 0.1   | 0.3   | 0.03   |

## Initial Weights

| $W_1$ | $W_2$ | $W_3$ | $W_4$ | $W_5$ | $W_6$ | $b_1$ | $b_2$ | $b_3$ |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0.5   | 0.1   | 0.62  | 0.2   | -0.2  | 0.3   | 0.4   | -0.1  | 1.83  |





# Forward pass: neurones de la couche cachée

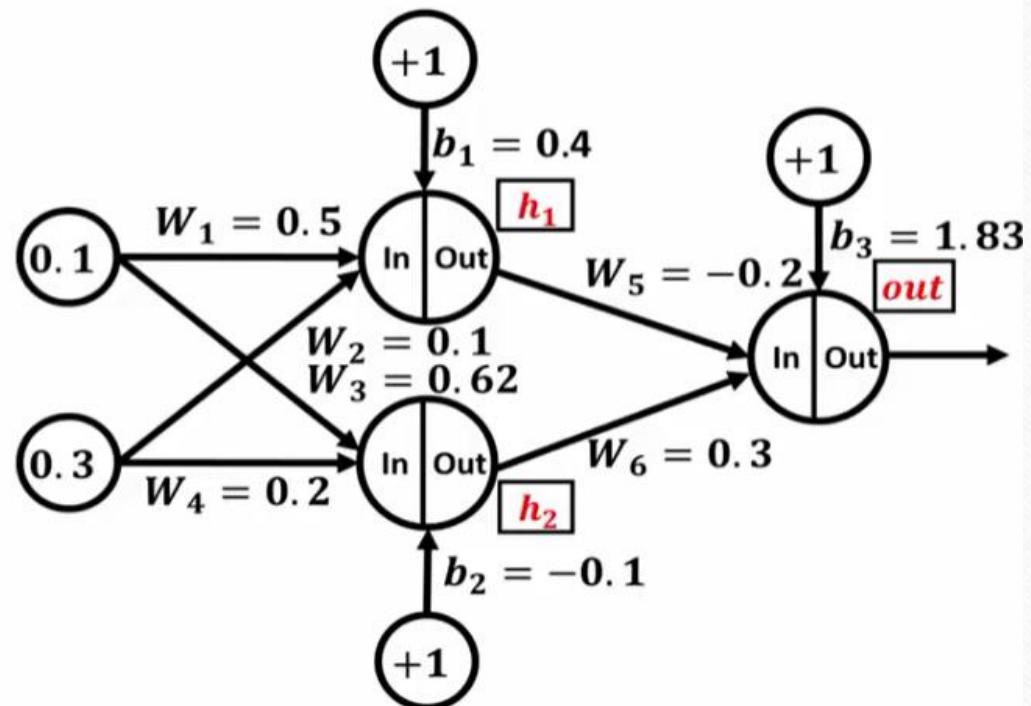
In

Out

$h_1$

$$h_{1in} = X_1 * W_1 + X_2 * W_2 + b_1 \\ = 0.1 * 0.5 + 0.3 * 0.1 + 0.4 \\ h_{1in} = 0.48$$

$$h_{1out} = \frac{1}{1 + e^{-h_{1in}}} \\ = \frac{1}{1 + e^{-0.48}} \\ h_{1out} = 0.618$$



# Forward pass: neurones de la couche cachée

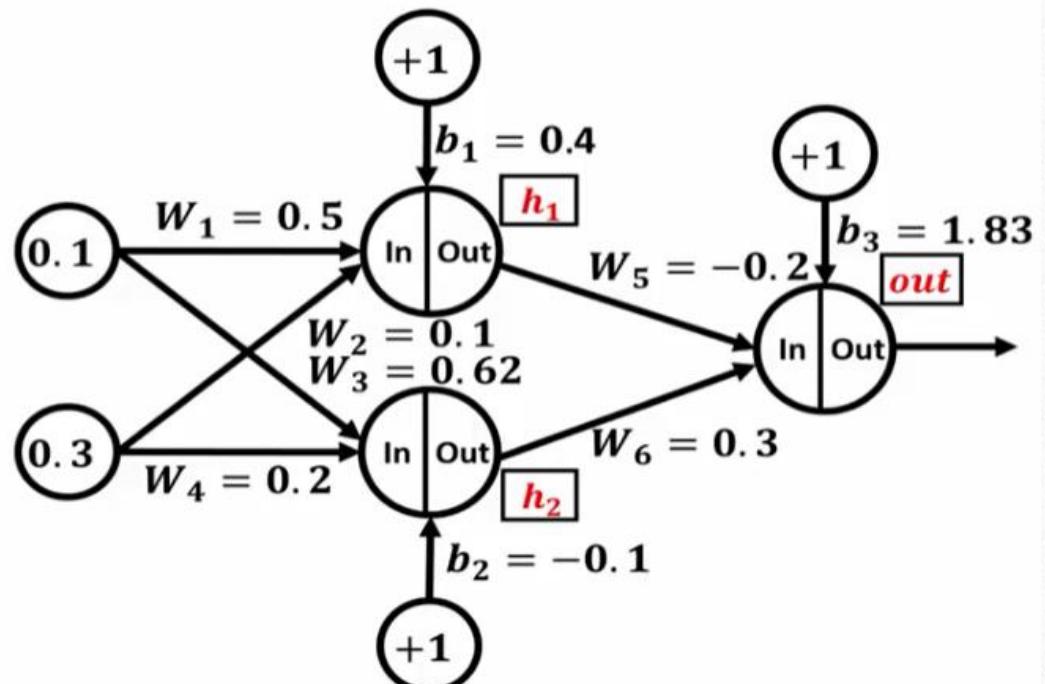
In  
Out

**$h_2$**

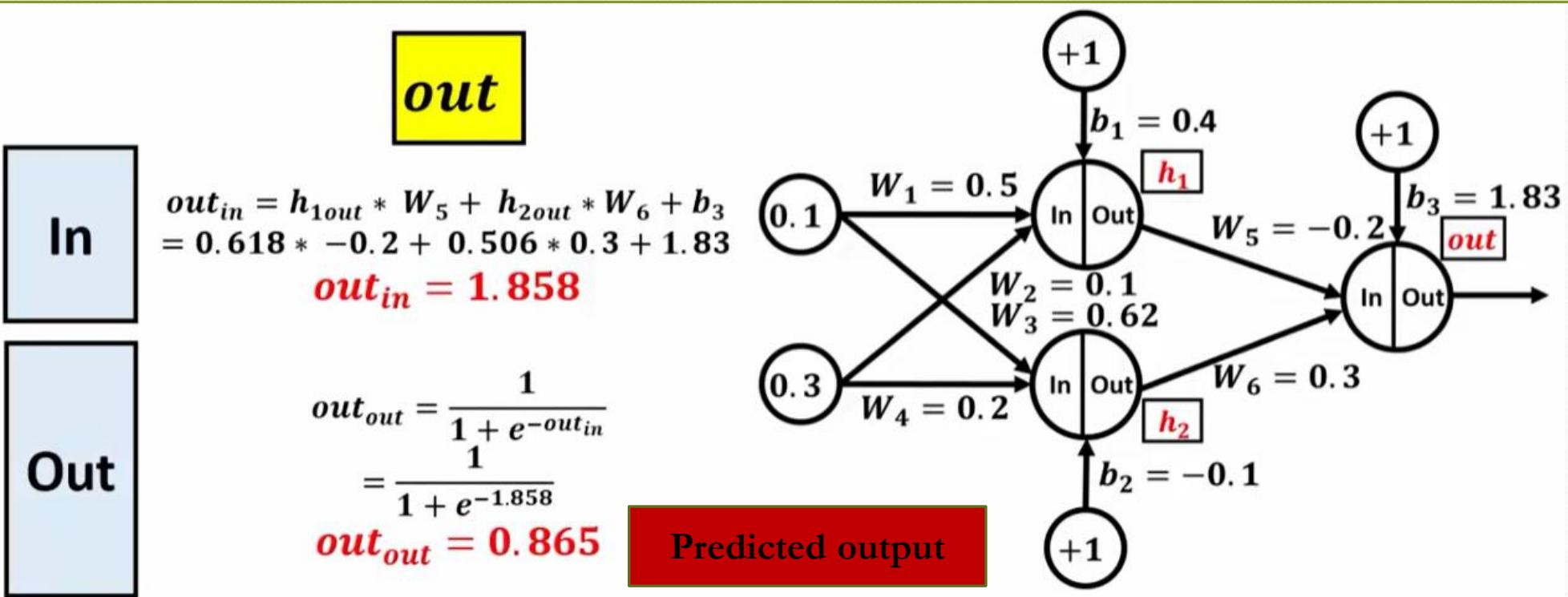
$$h_{2in} = X_1 * W_3 + X_2 * W_4 + b_2 \\ = 0.1 * 0.62 + 0.3 * 0.2 - 0.1 \\ h_{2in} = 0.022$$

In  
Out

$$h_{2out} = \frac{1}{1 + e^{-h_{2in}}} \\ = \frac{1}{1 + e^{-0.022}} \\ h_{2out} = 0.506$$



# Forward pass: neurones de la sortie



# Forward pass: Erreur de prédiction

---

*desired = 0.03*

*Predicted = out<sub>out</sub> = 0.865*

$$E = \frac{1}{2}(\text{desired} - \text{out}_{\text{out}})^2$$

$$= \frac{1}{2}(0.03 - 0.865)^2$$

$$\mathbf{E = 0.349}$$

# Forward pass: Erreur de prédiction

*desired = 0.03*

*Predicted = out<sub>out</sub> = 0.865*

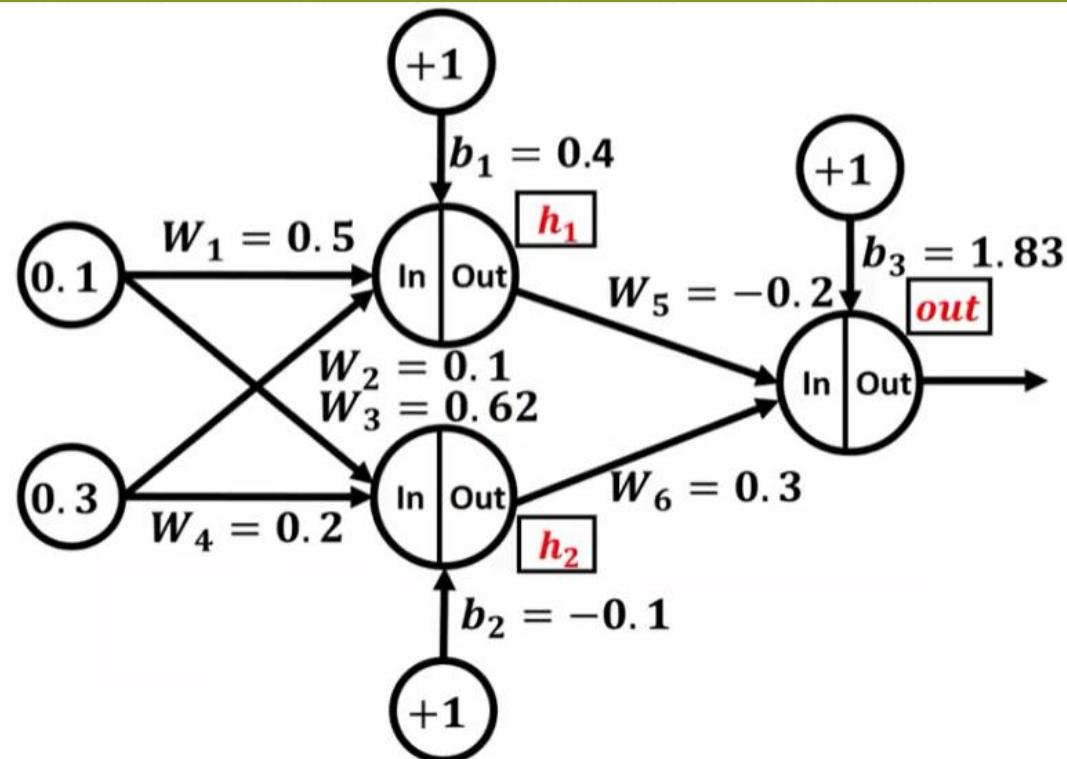
$$E = \frac{1}{2}(\text{desired} - \text{out}_{\text{out}})^2$$

$$= \frac{1}{2}(0.03 - 0.865)^2$$

$$E = 0.349$$

$$\frac{\partial E}{\partial W_1}, \frac{\partial E}{\partial W_2}, \frac{\partial E}{\partial W_3}, \frac{\partial E}{\partial W_4}, \frac{\partial E}{\partial W_5}, \frac{\partial E}{\partial W_6}$$

# Dérivées partielles



$$\frac{\partial E}{\partial W_5} = \frac{\partial E}{\partial \text{out}_{out}} * \frac{\partial \text{out}_{out}}{\partial \text{out}_{in}} * \frac{\partial \text{out}_{in}}{\partial W_5}$$

### Partial Derivative

$$\begin{aligned}\frac{\partial E}{\partial \text{out}_{out}} &= \frac{\partial}{\partial \text{out}_{out}} \left( \frac{1}{2} (\text{desired} - \text{out}_{out})^2 \right) \\ &= 2 * \frac{1}{2} (\text{desired} - \text{out}_{out})^{2-1} * (0 - 1) \\ &= \text{desired} - \text{out}_{out} * (-1) \\ \frac{\partial E}{\partial \text{out}_{out}} &= \text{out}_{out} - \text{desired}\end{aligned}$$

### Substitution

$$\begin{aligned}\frac{\partial E}{\partial \text{out}_{out}} &= \text{out}_{out} - \text{desired} = 0.865 - 0.03 \\ \frac{\partial E}{\partial \text{out}_{out}} &= 0.835\end{aligned}$$

$$\frac{\partial E}{\partial W_5} = \frac{\partial E}{\partial \text{out}_{out}} * \frac{\partial \text{out}_{out}}{\partial \text{out}_{in}} * \frac{\partial \text{out}_{in}}{\partial W_5}$$

## Partial Derivative

$$\frac{\partial \text{out}_{out}}{\partial \text{out}_{in}} = \frac{\partial}{\partial \text{out}_{in}} \left( \frac{1}{1 + e^{-\text{out}_{in}}} \right)$$

$$\frac{\partial \text{out}_{out}}{\partial \text{out}_{in}} = \left( \frac{1}{1 + e^{-\text{out}_{in}}} \right) \left( 1 - \frac{1}{1 + e^{-\text{out}_{in}}} \right)$$

$$\frac{\partial \text{out}_{out}}{\partial \text{out}_{in}} = \left( \frac{1}{1 + e^{-1.858}} \right) \left( 1 - \frac{1}{1 + e^{-1.858}} \right)$$

$$= \left( \frac{1}{1,158} \right) \left( 1 - \frac{1}{1,158} \right)$$

$$= (0,87)(1 - 0,87) = (0,87)(0,13)$$

$$\frac{\partial \text{out}_{out}}{\partial \text{out}_{in}} = 0,113$$

## Substitution

$$\frac{\partial E}{\partial W_5} = \frac{\partial E}{\partial out_{out}} * \frac{\partial out_{out}}{\partial out_{in}} * \frac{\partial out_{in}}{\partial W_5}$$

$$\frac{\partial E}{\partial out_{out}} = 0.835$$

$$\frac{\partial out_{out}}{\partial out_{in}} = 0,113$$

$$\frac{\partial out_{in}}{\partial W_5} = 0.618$$

$$\frac{\partial E}{\partial W_5} = 0.835 * 0,113 * 0.618$$

$$\frac{\partial E}{\partial W_5} = 0,058$$

$$\frac{\partial E}{\partial W_6} = \frac{\partial E}{\partial out_{out}} * \frac{\partial out_{out}}{\partial out_{in}} * \frac{\partial out_{in}}{\partial W_6}$$

$$\frac{\partial E}{\partial out_{out}} = 0.835$$

$$\frac{\partial out_{out}}{\partial out_{in}} = 0,113$$

$$\frac{\partial out_{in}}{\partial W_6} = 0.506$$

$$\frac{\partial E}{\partial W_6} = 0.835 * 0,113 * 0.506$$

$$\frac{\partial E}{\partial W_6} = 0.047$$

$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial out_{out}} * \frac{\partial out_{out}}{\partial out_{in}} * \frac{\partial out_{in}}{\partial h1_{out}} * \frac{\partial h1_{out}}{\partial h1_{in}} * \frac{\partial h1_{in}}{\partial W_1}$$

$$\frac{\partial E}{\partial out_{out}} = 0.835 \quad \frac{\partial out_{out}}{\partial out_{in}} = 0,113 \quad \frac{\partial out_{in}}{\partial h1_{out}} = -0.2 \quad \frac{\partial h2_{out}}{\partial h2_{in}} = 0.236 \quad \frac{\partial h1_{in}}{\partial W_1} = 0.1$$

$$\frac{\partial E}{\partial W_1} = 0.835 * 0,113 * -0.2 * 0.236 * 0.1$$

$$\frac{\partial E}{\partial W_1} = -0.001$$

$$W_{1new} = W_1 - \eta * \frac{\partial E}{\partial W_1} = 0.5 - 0.01 * -0.001 = 0.50001$$

$$W_{2new} = W_2 - \eta * \frac{\partial E}{\partial W_2} = 0.1 - 0.01 * -0.003 = 0.10003$$

$$W_{3new} = W_3 - \eta * \frac{\partial E}{\partial W_3} = 0.62 - 0.01 * 0.009 = 0.61991$$

$$W_{4new} = W_4 - \eta * \frac{\partial E}{\partial W_4} = 0.2 - 0.01 * 0.003 = 0.1997$$

$$W_{5new} = W_5 - \eta * \frac{\partial E}{\partial W_5} = -0.2 - 0.01 * 0,058 = -0,20058$$

$$W_{6new} = W_6 - \eta * \frac{\partial E}{\partial W_6} = 0.3 - 0.01 * 0,047 = 0.29903$$

- Continuer la mise à jour de b1, b2 et b3 selon les dérivées et re-apprendre le modèle jusqu'au avoir une erreur acceptable (qui tend vers 0)

# La fonction d'erreur pour les réseaux de neurones



# Notations

---

- Les données  $x^{(i)}$  sont décrites par les attributs:  $F_1, \dots, F_p$ , et la classe  $y^{(i)}$
- Données d'apprentissage:  $(x^{(i)}, y^{(i)}), i=1, \dots, m$  tels que  $x \in R^p$ ,  $y \in \{0, 1\}$ ,
- $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
- $x^{(i)} = (f_1^{(i)}, \dots, f_p^{(i)})$
- Les poids synaptiques des attributs:  $w \in R^p$ ,  $b \in R$
- La somme des produits:  $z^{(i)} = x^{(i)} w^T + b$
- La classe prédite en appliquant la sigmoïde:  $\hat{y}^{(i)} = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

# Fonction d'erreur

---

Ayant  $\{x^{(1)}, y^{(2)}, x^{(1)}, y^{(2)}, \dots, x^{(m)}, y^{(m)}\}$ , nous voulons  $\hat{y}^{(i)} = y^{(i)}$

- ~~La fonction de perte:  $L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2$~~ : n'est pas très utilisé maintenant
- Fonction de perte:  $L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)}))$
- Fonction d'erreur:  $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$

# Fonction d'erreur: explication intuitive

---

Ayant  $\{x^{(1)}, y^{(1)}, x^{(2)}, y^{(2)}, \dots, x^{(m)}, y^{(m)}\}$ , nous voulons  $\hat{y}^{(i)} = y^{(i)}$

- Fonction de perte:  $L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)}))$
- La fonction de perte logarithmique (log-vraisemblance) pénalise davantage les prédictions incorrectes avec une forte confiance (par exemple, prédire une haute probabilité pour une classe incorrecte), ce qui aide à ajuster les paramètres du modèle pour maximiser la précision des probabilités prédictives.

# Fonction d'erreur: explication intuitive

---

Ayant  $\{x^{(1)}, y^{(1)}, x^{(2)}, y^{(2)}, \dots, x^{(m)}, y^{(m)}\}$ , nous voulons  $\hat{y}^{(i)} = y^{(i)}$

- Fonction de perte:  $L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)}))$
- Si  $y = 1$  alors:  $L(\hat{y}^{(i)}, y^{(i)}) = -(\log(\hat{y}^{(i)}))$ :

→ on veut que  $\log(\hat{y}^{(i)})$  soit le plus élevé possible c.à.d  $\hat{y}^{(i)}$  le plus élevé possible et puisque  $\hat{y}^{(i)} = \sigma(z^{(i)})$  alors  $\hat{y}^{(i)}$  ne peut pas dépasser 1

# Fonction d'erreur: explication intuitive

Ayant  $\{x^{(1)}, y^{(1)}, x^{(2)}, y^{(2)}, \dots, x^{(m)}, y^{(m)}\}$ , nous voulons  $\hat{y}^{(i)} = y^{(i)}$

- Fonction de perte:  $L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)}))$
- Si  $y = 1$  alors:  $L(\hat{y}^{(i)}, y^{(i)}) = -(\log(\hat{y}^{(i)}))$ :

→ on veut que  $\log(\hat{y}^{(i)})$  soit le plus élevé possible c.à.d  $\hat{y}^{(i)}$  le plus élevé possible et puisque  $\hat{y}^{(i)} = \sigma(z^{(i)})$  alors  $\hat{y}^{(i)}$  ne peut pas dépasser **1**

- Si  $y = 0$  alors  $L(\hat{y}^{(i)}, y^{(i)}) = -(\log(1-\hat{y}^{(i)}))$ :

→ on veut que  $\log(1-\hat{y}^{(i)})$  soit le plus élevé possible c.à.d  $1 - \hat{y}^{(i)}$  le plus élevé possible et alors  $\hat{y}^{(i)}$  le plus petit possible et puisque  $\hat{y}^{(i)} = \sigma(z^{(i)})$  alors  $\hat{y}^{(i)}$  ne peut pas diminuer de **0**

# Probabilités associées à $y$

---

- Dans un problème de classification binaire, on cherche à prédire une variable  $y$  qui prend deux valeurs :
  - $y = 1$  pour la classe positive
  - $y = 0$  pour la classe négative
- L'objectif est de déterminer, pour une observation  $x$ , la probabilité que  $y$  soit égale à 1 ou 0.

# Probabilités associées à y

---

1. Probabilité d'appartenance à la classe **positive** :

- On note cette probabilité  $P(y = 1 \mid x)$ , souvent appelée  $\hat{y}$ .
- $P(y = 1 \mid x) = \hat{y}$

2. Probabilité d'appartenance à la classe **négative** :

- $P(y = 0 \mid x) = 1 - \hat{y}$

Les deux probabilités sont exclusives et leur somme est égale à 1.

# Fonction d'erreur: démonstration

---

- Nous avons généralement:
  - Si  $y = 1$  :  $P(y=1/x) = \hat{y}$
  - Si  $y = 0$  :  $P(y=0/x) = 1 - \hat{y}$

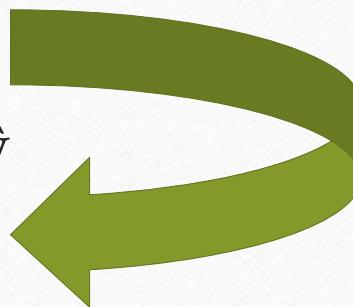
# Fonction d'erreur: démonstration

- Nous avons généralement:

- Si  $y = 1 : P(y=1/x) = \hat{y}$

- Si  $y = 0 : P(y=0/x) = 1 - \hat{y}$

$$P(y/x) = \hat{y}^y (1-\hat{y})^{1-y}$$



# Fonction d'erreur: démonstration

- Nous avons généralement:
  - Si  $y = 1$  :  $P(y=1/x) = \hat{y}$  : probabilité de la classe positive
  - Si  $y = 0$  :  $P(y=0/x) = 1 - \hat{y}$  : probabilité de la classe négative
- Pour récupérer l'erreur, on utilise la fonction de log
  - $\text{-log}(P(y/x)) = -(\log(\hat{y}^y (1-\hat{y})^{1-y})) = -(\log(\hat{y}^y) + \log((1-\hat{y})^{1-y}))$   
 $= -(y\log(\hat{y}) + (1-y)\log((1-\hat{y})))$

# Fonctions d'erreur

---

3 types de classification:

- Classification binaire : deux classes exclusives  
→ Perte d'entropie croisée binaire.
- Classement multi-classes : plus de deux classes exclusives  
→ Perte d'entropie croisée catégorielle
- Classement multi-labels : juste des classes non exclusives  
→ Perte d'entropie croisée binaire.

# Entropie croisée binaire

---

$$L(\hat{y}, y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

où :  $y$  est la valeur réelle (1 si l'étiquette est présente, 0 sinon),

$\hat{y}$  est la probabilité prédite par le modèle pour cette étiquette.

La perte d'entropie croisée binaire est idéale pour :

- **Classification multi-labels** : Chaque classe est évaluée indépendamment, donc le modèle peut attribuer plusieurs étiquettes correctes (ou incorrectes) à chaque exemple sans contrainte d'exclusivité entre les classes.
- **Classification binaire** : Dans ce cas, elle est utilisée pour évaluer une seule classe (positive ou négative).

# Entropie croisée catégorique

---

$$L(\hat{y}, y) = - \sum_{i=1}^C y_i * \log(\hat{y}_i)$$

- où :
- $y_i$  est 1 si la classe réelle est la classe  $i$ , et 0 sinon,
- $\hat{y}_i$  est la probabilité prédite par le modèle pour la classe  $i$ .

# Entropie croisée catégorique

---

$$L(\hat{y}, y) = - \sum_{i=1}^C y_i * \log(\hat{y}_i)$$

- La perte d'entropie croisée catégorielle est particulièrement adaptée à la **Classification multi-classes** : Chaque exemple est classé dans une seule catégorie exclusive parmi plusieurs (par exemple, classifier des images en différentes espèces d'animaux).
- **Réseaux de neurones** : Souvent utilisée en sortie des modèles avec une couche de sortie softmax, où la somme des probabilités pour chaque classe est de 1.

# Entropie croisée catégorique: fonctionnement

---

- Pour chaque exemple, la perte d'entropie croisée catégorielle se concentre uniquement sur la probabilité de la classe réelle (celle où  $y_i = 1$ ).
- Si le modèle prédit une haute probabilité pour la classe correcte, la perte sera faible, ce qui indique une bonne performance.
- Si le modèle assigne une faible probabilité à la classe correcte, la perte sera élevée, pénalisant le modèle pour cette erreur.

# Entropie croisée catégorique: exemple

---

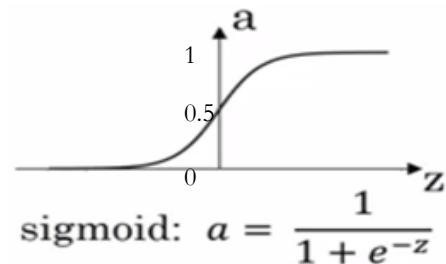
- Supposons un modèle qui doit classer une image en trois catégories : "chat", "chien", ou "oiseau". Si l'image est réellement celle d'un chat, alors :  
 $y=[1,0,0]$  (indiquant "chat" comme classe réelle).
- Si le modèle prédit les probabilités  $\hat{y}=[0.8,0.15,0.05]$ , **la perte d'entropie croisée sera faible**, car il a prédit une forte probabilité pour la classe correcte.
- En revanche, si le modèle prédit  $\hat{y}=[0.3,0.4,0.3]$ , **la perte sera élevée**, car la classe correcte ("chat") n'a pas été prédite avec une forte probabilité.

# Les fonctions d'activation dans les RN



# La fonction Sigmoïde

Sigmoïde prend une entrée et effectue les opérations suivantes:

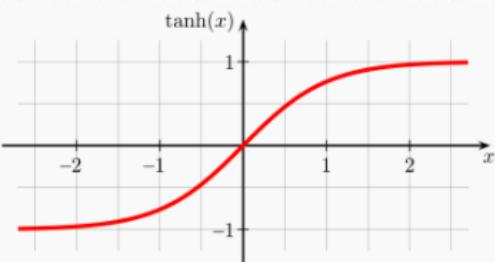


- Pour la plupart des entrées négatives, sigmoïde transformera l'entrée en un nombre très proche de 0.
- Pour la plupart des entrées positives, sigmoïde transformera l'entrée en un nombre très proche de 1.
- Pour les entrées relativement proches de 0, sigmoïde transformera l'entrée en un certain nombre entre 0 et 1.

# La fonction tangente hyperbolique

Notée tanh, sa formule est:

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

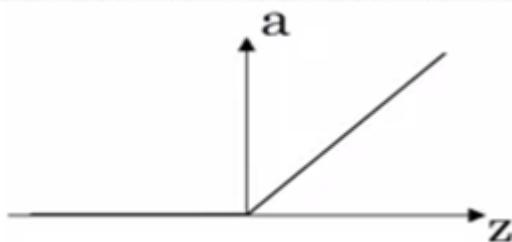


- Pour la plupart des entrées négatives, tanh transformera l'entrée en un nombre très proche de -1.
- Pour la plupart des entrées positives, tanh transformera l'entrée en un nombre très proche de 1.
- Pour les entrées relativement proches de 0, tanh transformera l'entrée en un certain nombre entre 0 et 1.
- Si l'entrée est très élevée ou très faible, alors le gradient (la dérivée, la pente) devient très faible (proche de 0) et cela **peut ralentir la descente de gradient (Vanishing Problem)**.

# L'unité linéaire rectifiée

La fonction Relu est:

$$\text{ReLU}(z) = \max(0, z)$$

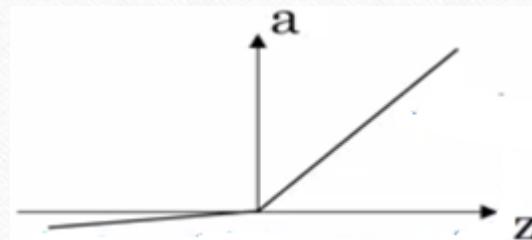


- Pour les entrées négatives, Relu transformera l'entrée en 0.
- Pour les entrées positives, Relu renvoie l'entrée.
- Un désavantage de la fonction ReLU est que sa dérivée devient nulle lorsque l'entrée est négative ce qui peut empêcher la rétropagation de gradient

# La fonction *Leaky ReLU*

- *Leaky ReLU* est définie par :

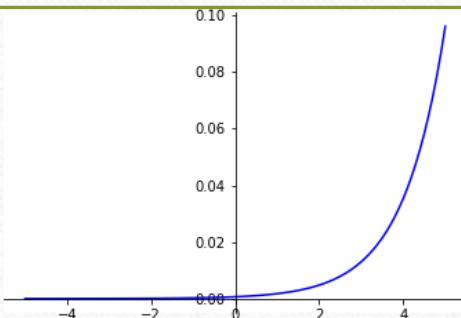
$$f(z) = \max(\epsilon z, z)$$



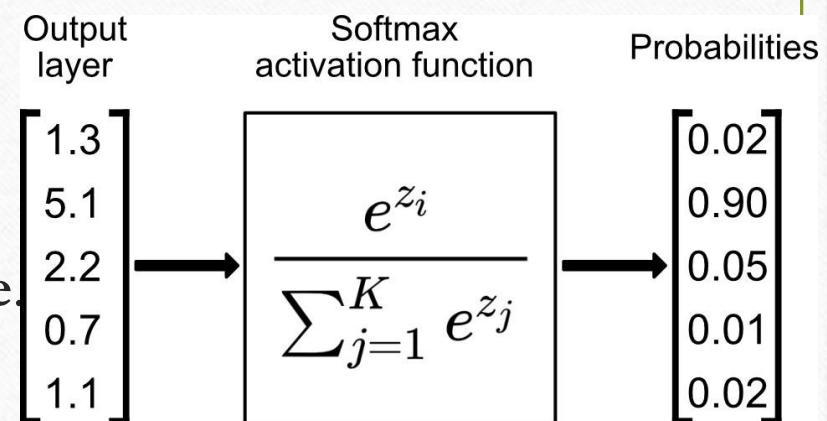
- Le paramètre  $\epsilon$  est un réel strictement positif et inférieur à 1. La dérivée est alors égale à  $\epsilon$  lorsque  $z$  est strictement négatif, ce qui permet de conserver la mise à jour des poids.
- l'avantage de la ReLU et de la leaky ReLU est que pour une grande plage de  $z$ , la dérivée de la fonction d'activation, est très différente de 0. Et donc, en pratique, en utilisant la fonction d'activation ReLU, votre réseau de neurones va souvent apprendre beaucoup plus rapidement qu'avec une tanh ou avec la fonction d'activation sigmoïde.

# La fonction SoftMax

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$



- La fonction **Softmax** permet elle de **transformer** un **vecteur réel** en **vecteur de probabilité**.
- On l'utilise **souvent** dans la **couche finale** d'un modèle de **classification**, notamment pour les **problèmes multi-classe**.



# Les fonctions de propagation



# Fonctions de propagation avant

---

- Pour la couche  $l$ :

- Entrée:  $a^{[l-1]}$
  - Sortie:  $a^{[l]}$

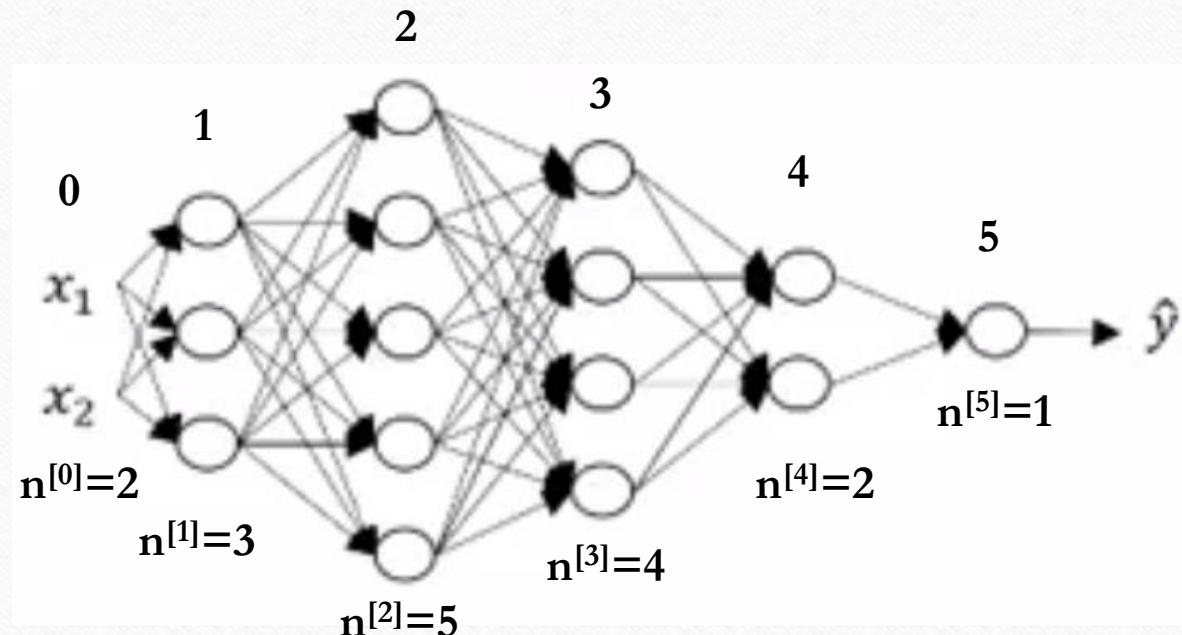
$$a^{[l]} = g^{[l]}(Z^{[l]})$$

$$Z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$

# Propagation avant

$$L = \#\text{layers} = 5$$

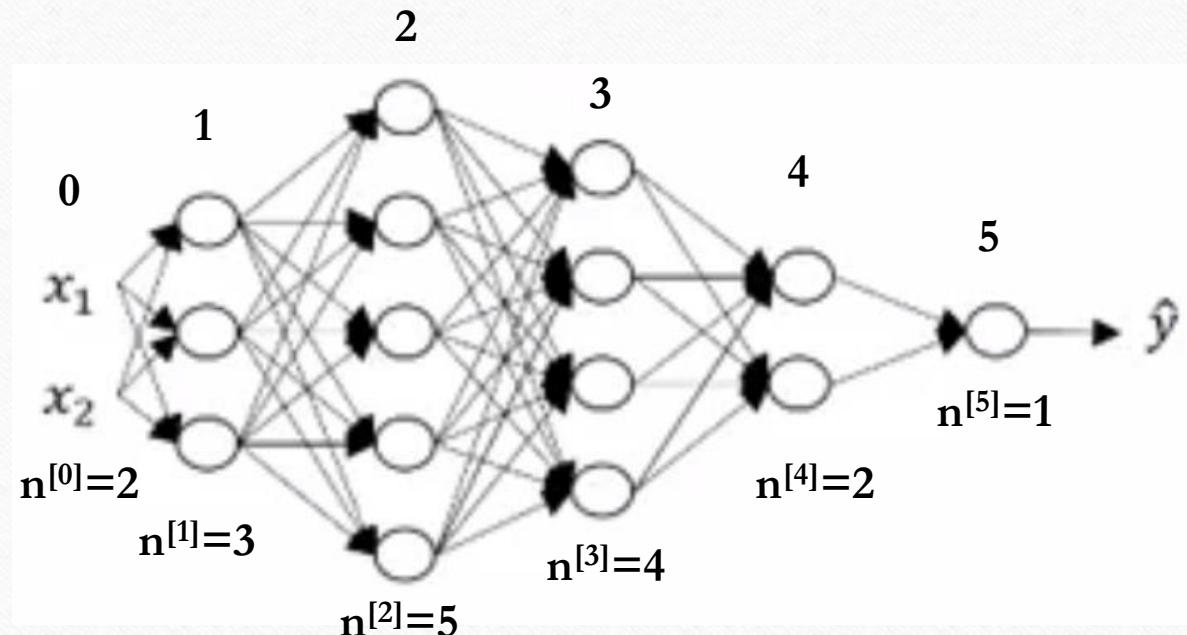
- $Z^{[1]} = W^{[1]} x + b^{[1]}$
- $a^{[1]} = g^{[1]}(Z^{[1]})$
- $W^{[1]}: (n^{[1]}, n^{[0]}): (3, 2)$
- $x: (2, 1)$
- $Z^{[1]}: (3, 1): (3, 2) * (2, 1)$
- $a^{[1]}: (3, 1)$



# Propagation avant

$$L = \# \text{layers} = 5$$

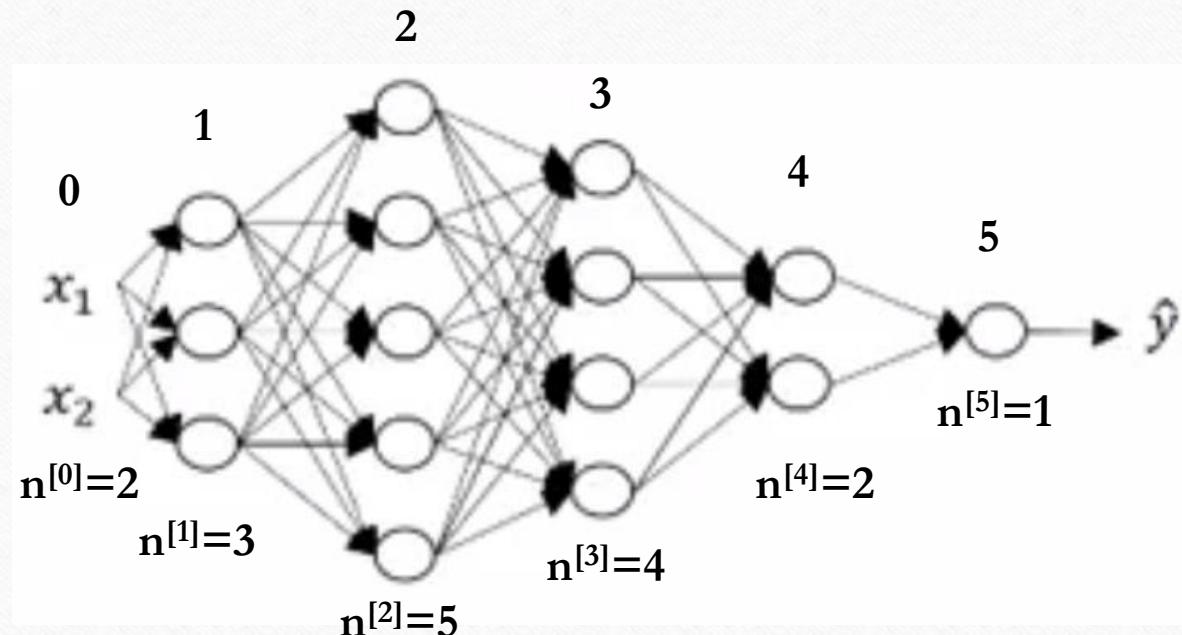
- $Z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$
- $a^{[2]} = g^{[2]}(Z^{[2]})$
- $W^{[2]}: (n^{[2]}, n^{[1]}): (5, 3)$
- $a^{[1]}: (3, 1)$
- $Z^{[2]}: (5, 1): (5, 3) * (3, 1)$
- $a^{[2]}: (5, 1)$



# Propagation avant

$$L = \# \text{layers} = 5$$

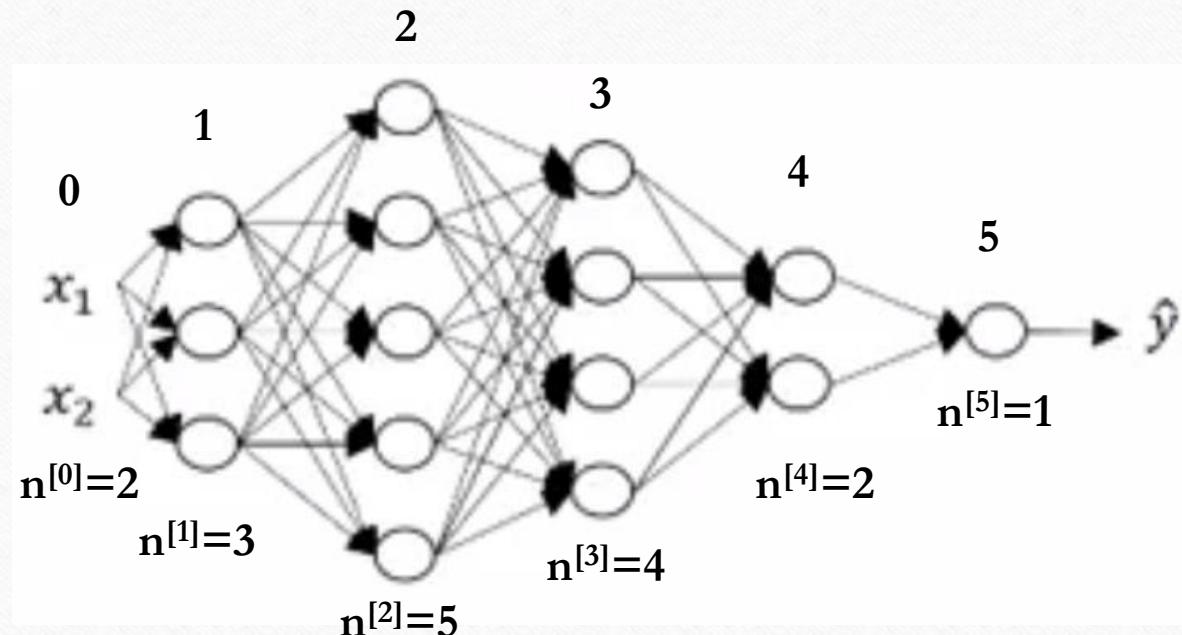
- $Z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}$
- $a^{[3]} = g^{[3]}(Z^{[3]})$
- $W^{[3]}: (n^{[3]}, n^{[2]}): (4, 5)$
- $a^{[2]}: (5, 1)$
- $Z^{[3]}: (4, 1): (4, 5) * (5, 1)$
- $a^{[3]}: (4, 1)$



# Propagation avant

$$L = \#\text{layers} = 5$$

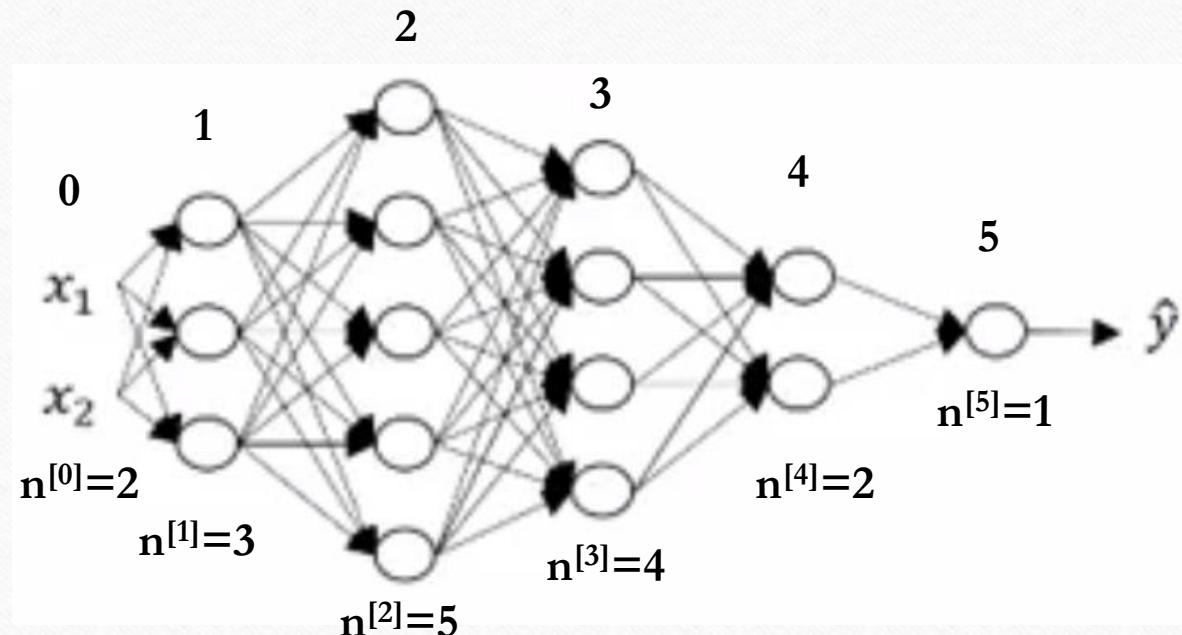
- $Z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}$
- $a^{[4]} = g^{[4]}(Z^{[4]})$
- $W^{[4]}: (n^{[4]}, n^{[3]}): (2, 4)$
- $a^{[3]}: (4, 1)$
- $Z^{[4]}: (2, 1): (2, 4) * (4, 1)$
- $a^{[4]}: (2, 1)$



# Propagation avant

$$L = \#\text{layers} = 5$$

- $Z^{[5]} = W^{[5]} a^{[4]} + b^{[5]}$
- $a^{[5]} = g^{[5]}(Z^{[5]})$
- $W^{[5]}$ : ?
- $a^{[4]}$ :  $(2, 1)$
- $Z^{[5]}$ : ?
- $a^{[5]}$ : ?



# Fonctions de rétro-propagation

---

- Pour la couche  $l$ :  $w^{[l]}, b^{[l]}$
- Entrée:  $da^{[l]}$  :  $dL/da^{[l]}$
- Sortie:  $da^{[l-1]}, dw^{[l]}, db^{[l]}$  ( $dL/da^{[l-1]}, dL/dw^{[l]}, dL/db^{[l]}$ )

$$dz^{[l]} = da^{[l]} * g'(z^{[l]})$$

$$dw^{[l]} = dz^{[l]} a^{[l-1]}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = w^{[l]} * dz^{[l]}$$

$$dz^{[l-1]} = w^{[l]} * dz^{[l]} * g'(z^{[l-1]})$$

- Mise à jour:  $w^{[l]} = w^{[l]} - \eta dw^{[l]}, b^{[l]} = b^{[l]} - \eta db^{[l]}$

# Sommaire de la descente de gradient

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]} \quad \# \text{layers} = 2 \\ \# \text{samples} = 1$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

#layers = 2

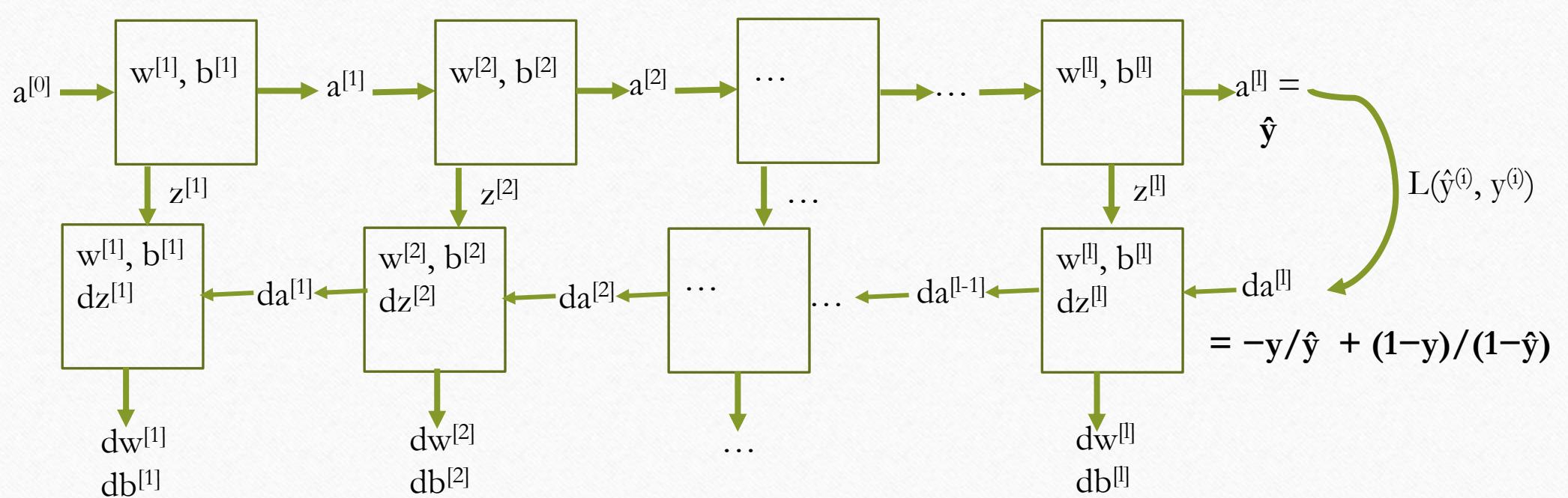
#samples = m

$$\mathcal{J}(w) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(y_i, \hat{y}_i)$$

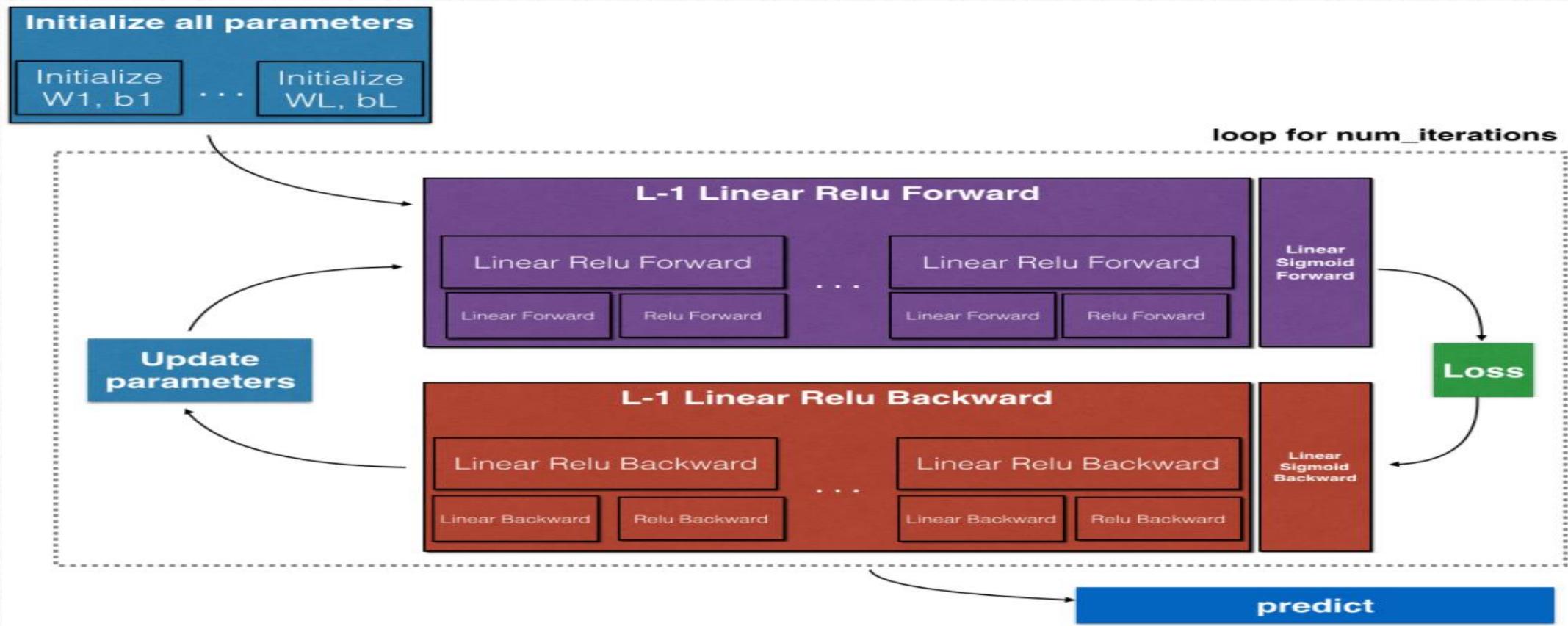
12:34

Andrew Ng

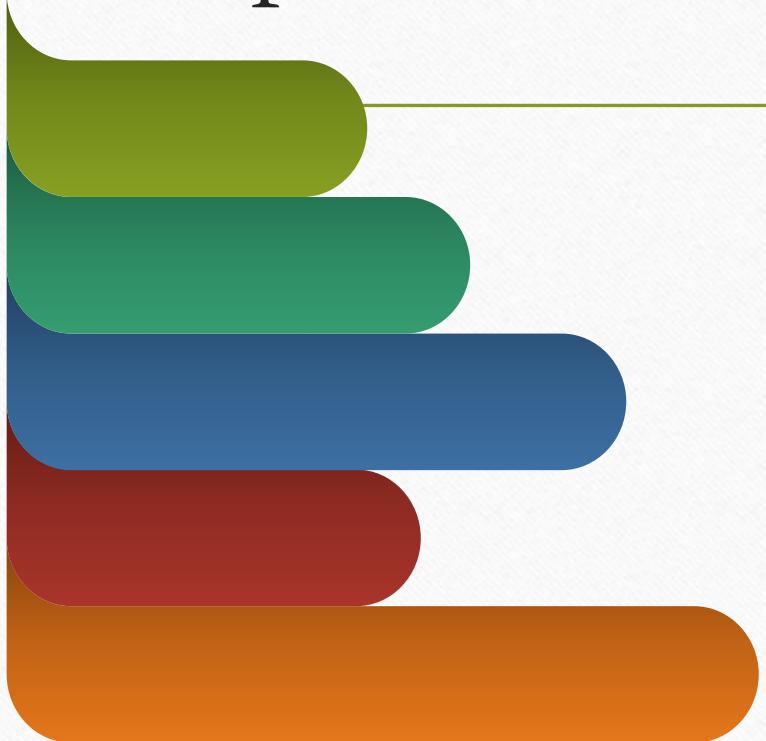
# Revue



# Revue



# Compromis biais-variance



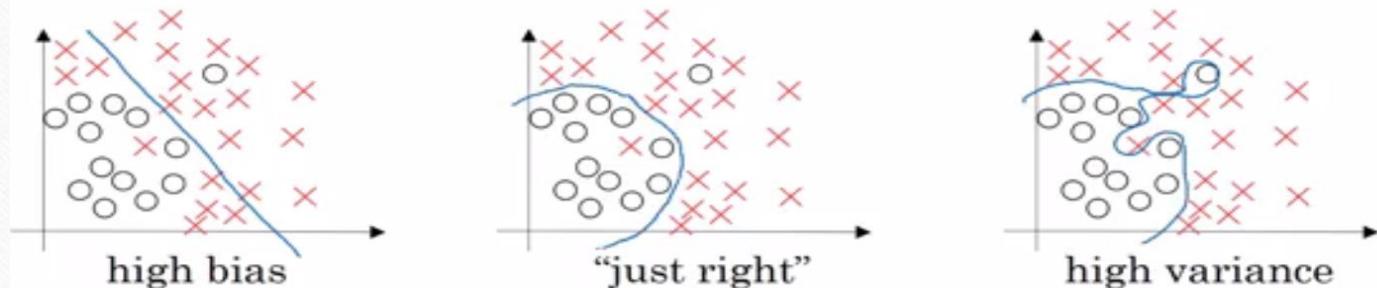
# Sur-apprentissage vs Sous-apprentissage

---

- **Sur-apprentissage** : (le terme anglais est overfitting) quand un modèle a trop appris les particularités de chacun des exemples fournis en exemple. Il présente alors un taux de succès très important sur les données d'apprentissage (pouvant atteindre jusqu'à 100%), mais se généralise mal (performance moins bonnes sur les données de test).
- **Sous-apprentissage** : (le terme anglais est underfitting) un algorithme qui n'apprend pas suffisamment de la phase d'apprentissage et qui manque de relations pertinentes entre les données en entrée et les sorties prévues (mauvaise performance sur le training set)

# Biais vs Variance

- Le biais est l'erreur provenant d'hypothèses erronées dans l'algorithme d'apprentissage. Un biais élevé peut être lié à un sous-apprentissage.
- La variance est l'erreur due à la sensibilité aux petites fluctuations de l'échantillon d'apprentissage. Une variance élevée peut entraîner un sur-apprentissage, c'est-à-dire modéliser le bruit aléatoire des données d'apprentissage plutôt que les sorties prévues.

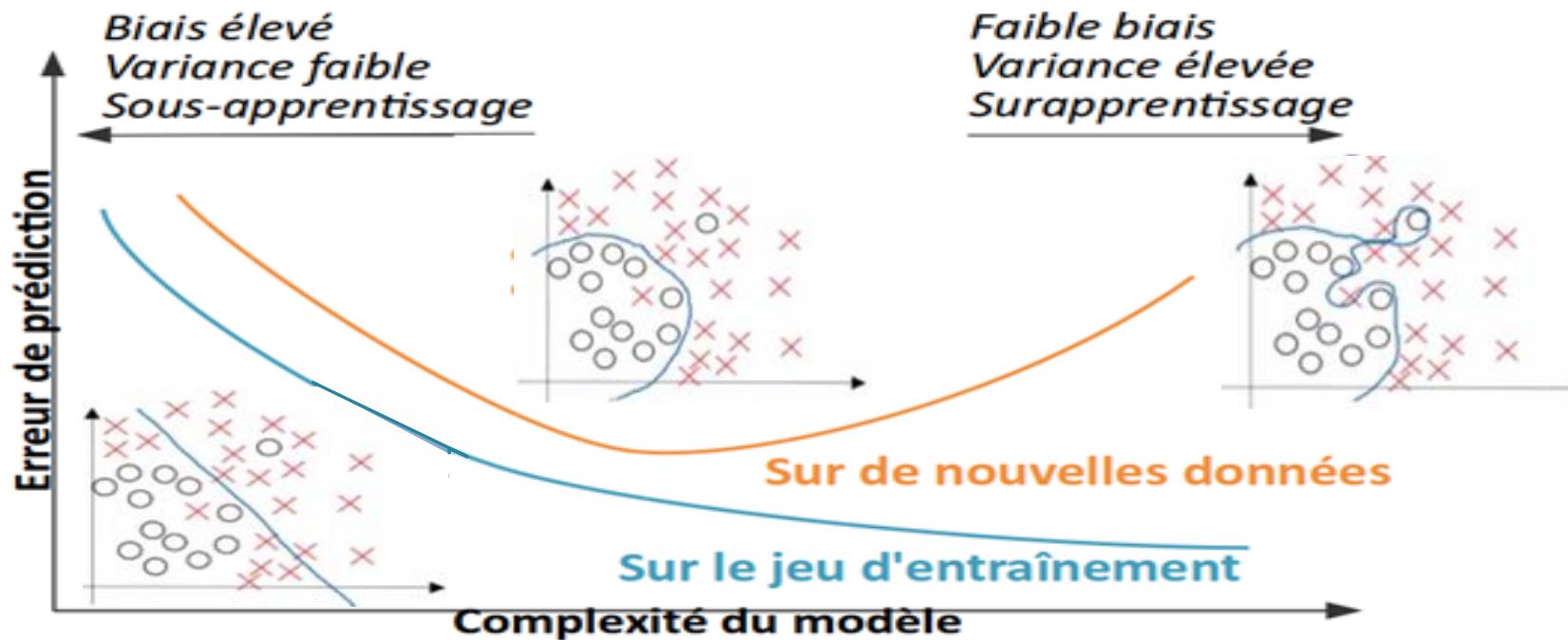


# Compromis biais/variance

---

- Le principe de **compromis entre biais et variance** est une des problématiques à laquelle vous serez confrontés lors de votre travail quotidien !
- En utilisant un modèle comportant une **trop grande complexité**, dit "**à haute variance**", on peut mal capturer le phénomène sous-jacent et devenir trop dépendant aux données d'apprentissage et aux petites fluctuations aléatoires, non représentatives du phénomène.
- A contrario, il ne faut pas choisir un modèle **trop "simple"** qui biaise le résultat et ne parvient pas à capturer toute la complexité du phénomène.

# Compromis biais/variance



# Compromis biais/variance

---

## Biais élevé

- Modéliser un réseau plus grand (plus de couches cachées ou plus d'unités cachées)
- Entrainer le modèle plus longtemps
- Modifier l'architecture du RN

## Variance élevée

- Ajout de données supplémentaires à l'ensemble d'apprentissage
- Régularisation
- Modifier l'architecture du RN

# Réduire le sur-apprentissage

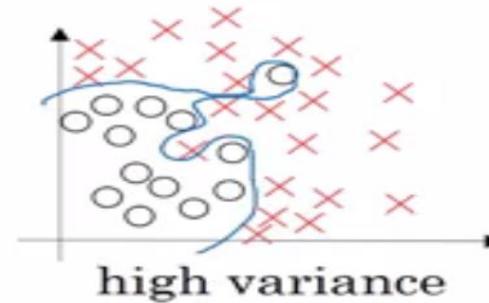
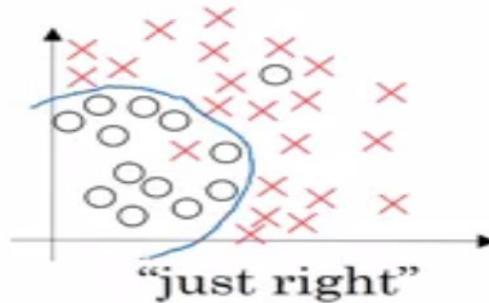
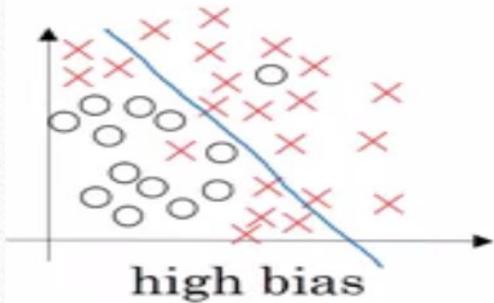


# Régularisation

---

- La régularisation est une technique qui permet de réduire le sur-apprentissage ou de réduire la variance de notre réseau en pénalisant la complexité.
- L'idée est que certaines complexités de notre modèle peuvent rendre notre modèle peu susceptible de se généraliser correctement, même si le modèle correspond aux données d'apprentissage.

# Régularisation



Andrew Ng

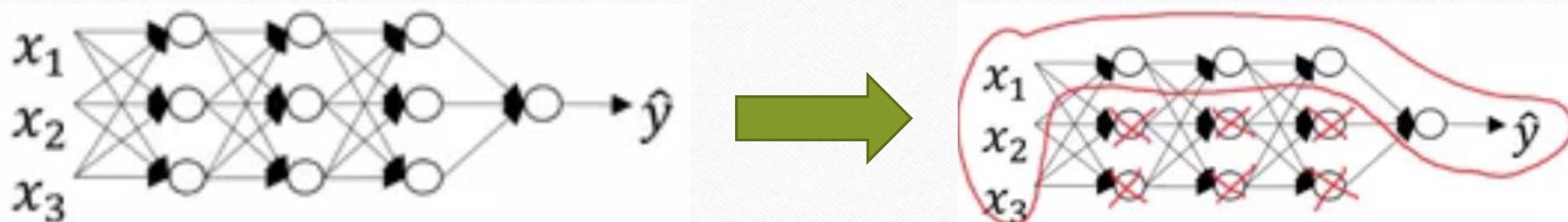
- → Si nous ajoutons une régularisation à notre modèle, nous échangeons essentiellement une partie de la capacité de notre modèle à bien s'adapter aux données d'entraînement contre la possibilité de mieux généraliser le modèle à des données qu'il n'a jamais vues auparavant.

# Régularisations L1 et L2

- Mettre en œuvre la régularisation, c'est simplement ajouter un terme à notre fonction d'erreur qui pénalise les gros poids (les plus importants)
- Deux techniques de régularisation connus: L1 et L2
- L1:  $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \sum_{j=1}^n (\|w^{[j]}\|) \frac{\lambda}{m}$
- L2:  $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \sum_{j=1}^n (\|w^{[j]}\|^2) \frac{\lambda}{2m}$ 
  - n= nombre de couches
  - m= nombre d'exemples
  - $w^{[j]}$ =la matrice de poids de la j<sup>ème</sup> couche
  - $\lambda$  = le paramètre de régularisation

# Régularisation L1

- L1:  $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \sum_{j=1}^n (\|w^{[j]}\|) \frac{\lambda}{m}$
- La plupart des poids W auront comme valeur 0 → Un réseau plus simple et rapide



- Le problème qu'on va se débarrasser de plusieurs entrées → un réseau moins précis.

# Régularisation L2

---

- La régularisation la plus courante est la régularisation L2.
- L2:  $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \sum_{j=1}^n (\|w^{[j]}\|^2) \frac{\lambda}{2m}$

# Régularisation L2 et rétro-propagation

---

Avant la régularisation:  $dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]}$

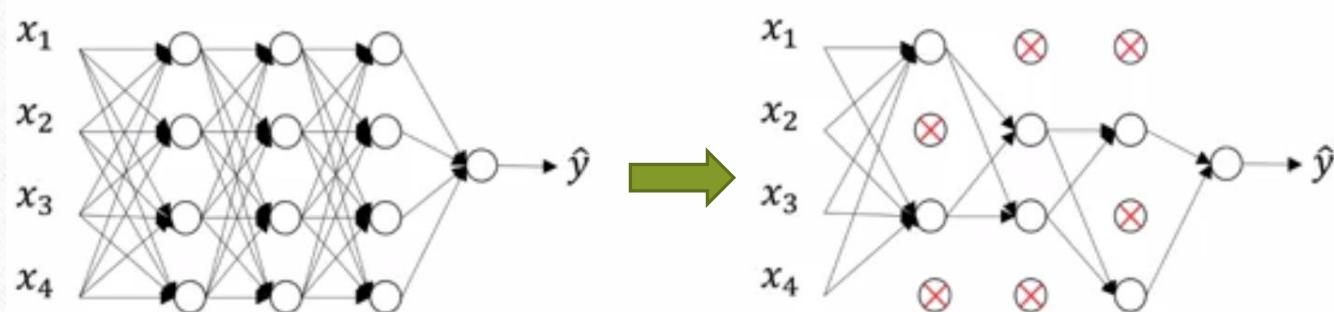
→ Mise à jour des poids:  $W^{[l]} = W^{[l]} - \eta dW^{[l]}$

Après la régularisation:  $dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]} + W^{[l]} \frac{\lambda}{m}$

→ Mise à jour des poids: 
$$\begin{aligned} W^{[l]} &= W^{[l]} - \eta \left( \frac{1}{m} dZ^{[l]} A^{[l-1]} + W^{[l]} \frac{\lambda}{m} \right) \\ &= W^{[l]} - \eta \frac{1}{m} dZ^{[l]} A^{[l-1]} - \eta W^{[l]} \frac{\lambda}{m} \\ &= W^{[l]} \left( 1 - \eta \frac{\lambda}{m} \right) - \eta \frac{1}{m} dZ^{[l]} A^{[l-1]} \end{aligned}$$

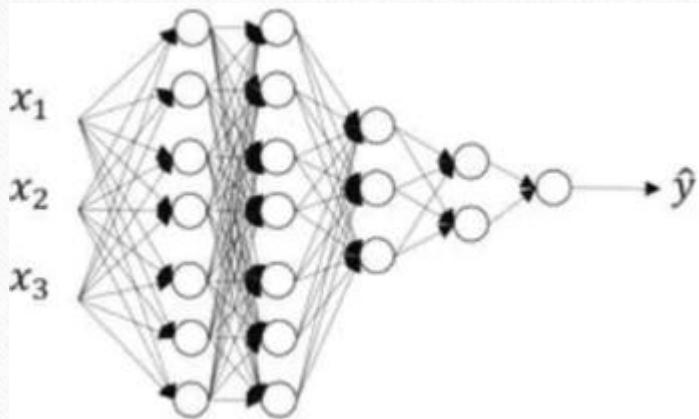
# La technique Drop-out

- Cette technique ignorera au hasard un sous-ensemble de neurones dans une couche donnée pendant l'apprentissage, c'est-à-dire qu'il supprime les nœuds de la couche. D'où le nom « drop-out ». Cela empêchera ces neurones abandonnés de participer à la production d'une prédiction sur les données.



Exemple Drop-out (50%)

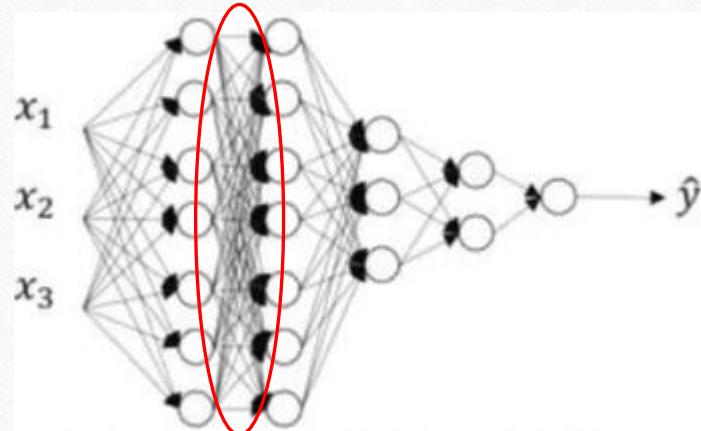
# La technique Drop-out



|       |    |
|-------|----|
| (3x7) | w1 |
| (7x7) | w2 |
| (7x3) | w3 |
| (3x2) | w4 |
| (2x1) | w5 |

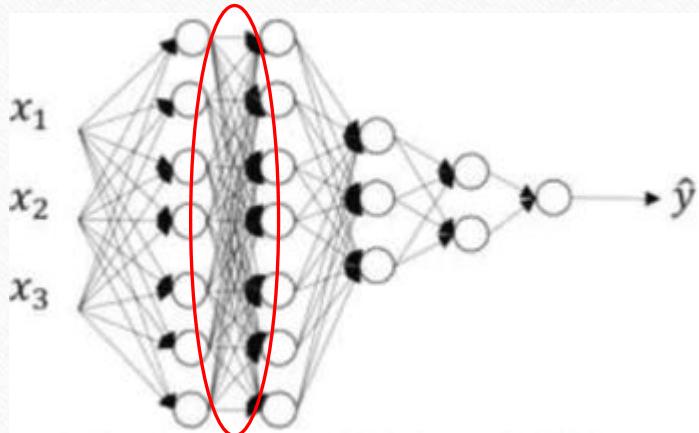
- Le problème de sur-apprentissage provient des couches ayant un nombre élevé des poids.

# La technique Drop-out



|       |    |
|-------|----|
| (3x7) | w1 |
| (7x7) | w2 |
| (7x3) | w3 |
| (3x2) | w4 |
| (2x1) | w5 |

# La technique Drop-out



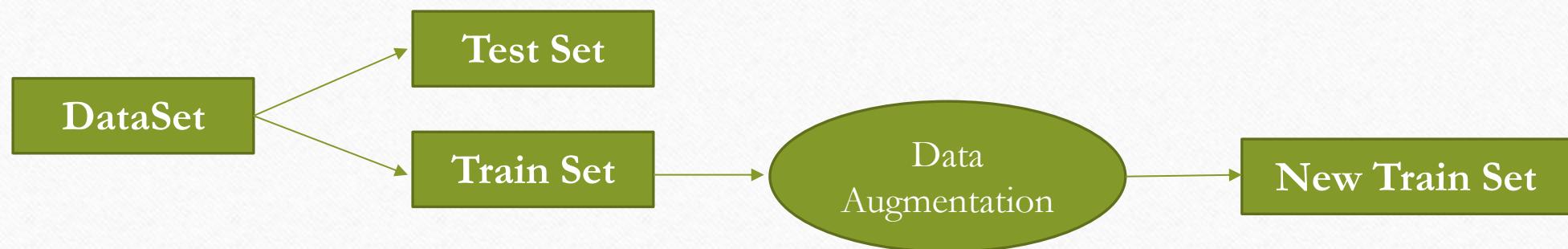
|       |    |
|-------|----|
| (3x7) | w1 |
| (7x7) | w2 |
| (7x3) | w3 |
| (3x2) | w4 |
| (2x1) | w5 |



|     |       |    |
|-----|-------|----|
| 0.7 | (3x7) | w1 |
| 0.5 | (7x7) | w2 |
| 0.7 | (7x3) | w3 |
| 1   | (3x2) | w4 |
| 1   | (2x1) | w5 |

# Augmentation de données

Il s'agit du processus de création de données supplémentaires en modifiant raisonnablement les données de notre ensemble d'apprentissage



# Augmentation de données (Computer vision)

- Pour les données image, nous pouvons effectuer des modifications en appliquant:
  - ✓ Recadrage
  - ✓ Rotation
  - ✓ Retournement
  - ✓ Zooming

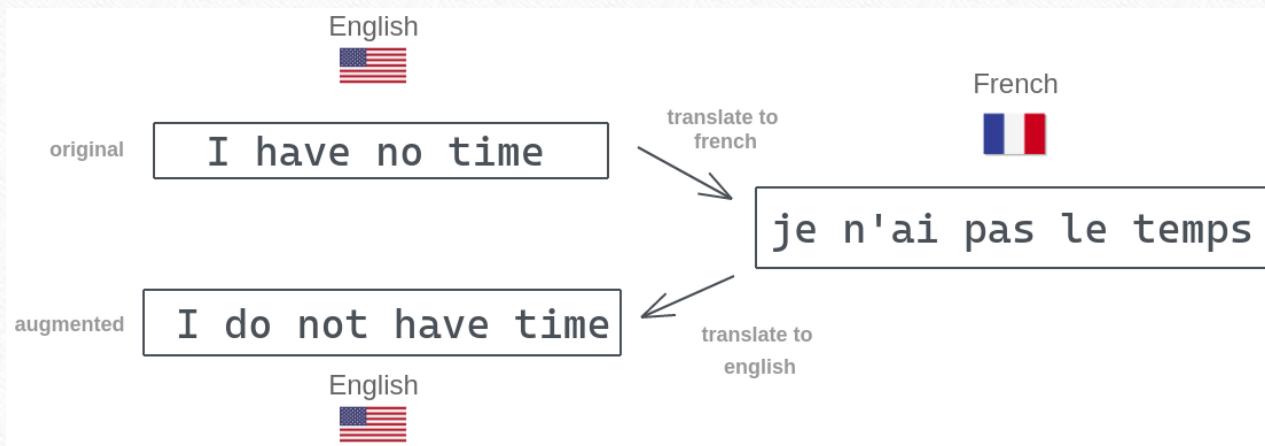


Image Source

Retournement horizontal

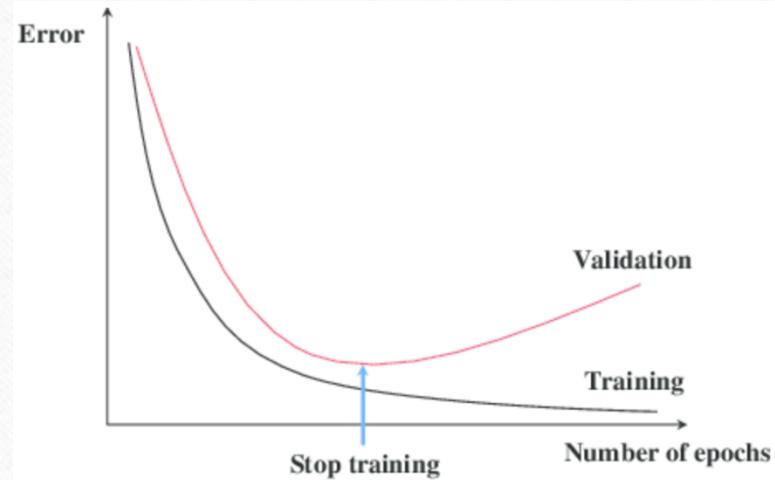
# Augmentation de données (NLP)

- . Pour les données textuelles, nous pouvons effectuer des modifications en appliquant:
  - ✓ Traduction inversée
  - ✓ Remplacement de synonymes  
(Word embedding, dictionnaires)
  - ✓ Insertion aléatoire
  - ✓ Echange aléatoire
  - ✓ Suppression aléatoire
  - ✓ Mélanger les phrases
  - ✓ transformer (forme active → forme passive), etc.

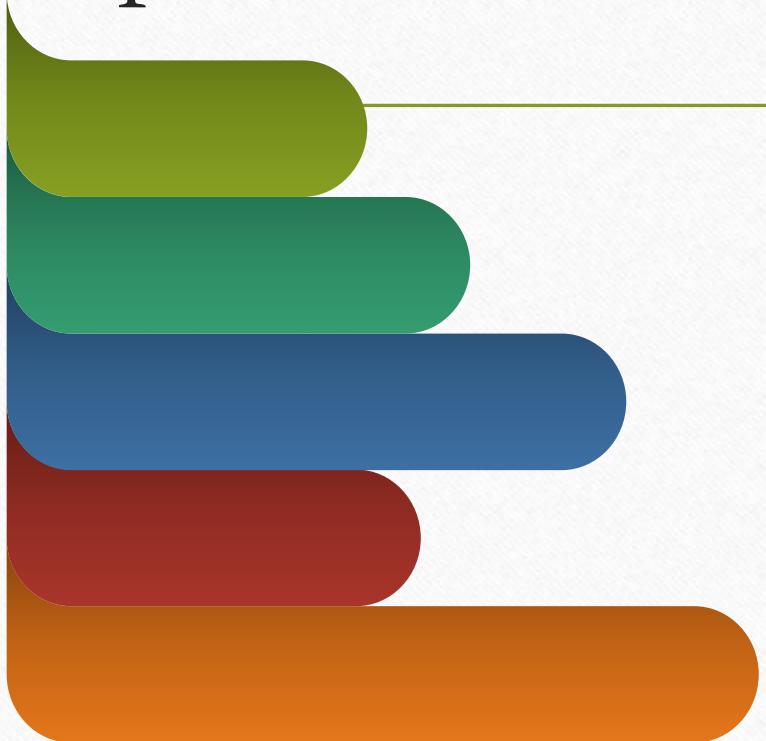


# arrêt précoce (early stopping)

- La méthode la plus simple pour éviter le sur-apprentissage consiste à arrêter la descente de gradient lorsque que la fonction d'erreur calculée sur l'ensemble de validation commence à augmenter et que la fonction d'erreur calculée sur l'ensemble d'apprentissage continue de descendre.

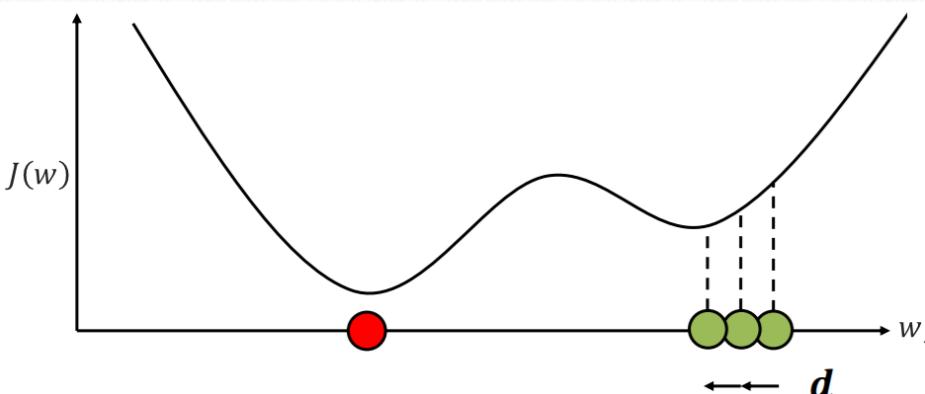


# Optimisation



# Descente du gradient par batch

- Fonction d'erreur:  $J(w) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$
  - $d = \frac{dJ(w)}{dw}$
  - Mise à jour :  $w = w - \eta * d$ 
    - + Facile à comprendre.
    - + Facile à programmer.
- Trop sensible au minimum/maximum locaux pour être utilisés en pratique.



# Descente du gradient par batch

---

## Autres inconvénients

- Traiter tous les exemples d'entraînement  $(x^{(i)}, y^{(i)})$ , donne une seule mise-à-jour → processus **lent**
- Pas de MAJ entre  $(x^{(i)}, y^{(i)})$ , et  $(x^{(i+j)}, y^{(i+j)})$  :

Erreur semblable → gradient semblable → redondance

# Descente du gradient stochastique

---

- pour améliorer la descente du gradient, on sélectionne aléatoirement une seule instance en entrée pour estimer l'erreur et le gradient.
- Cette sélection étant faite aléatoirement à **chaque itération**, on qualifie le gradient de stochastique : fruit du hasard.
  - + Les mises à jour sont plus fréquentes (le réseau évolue régulièrement). Dans la pratique, diminue le temps de convergence.
  - Du bruit apparaît dans le gradient et par conséquent dans la courbe de l'erreur aussi.

# Descente de gradient des mini-batchs

---

- Une mini-batch de taille  $p \rightarrow$  une MAJ fréquente
- **Algorithme:**
  1. Échantillonnage:  $(x^{(i)}, y^{(i)}) \quad 1 \leq i \leq p.$
  2. Gradient:  $\mathbf{d} = \widehat{\mathbf{g}} = \frac{1}{p} \sum_{i=1}^p d L(\hat{y}^{(i)}, y^{(i)}) / dw$
  3. Mise à jour :  $w = w - \eta * d$

# Descente de gradient des mini-batchs: Avantages

---

1.  $\hat{g}$  est la version bruité du gradient

Bruit aide à sortir des minimum / maximum locaux.

2. MAJ entre  $(x^{(i)}, y^{(i)})$ , et  $(x^{(i+p)}, y^{(i+p)})$

→ Diminue la redondance si erreur semblable.

→ Convergence possible avant même de voir tous les exemples.

3. Traitement parallèle des batchs.

4. Optimisation hardware quand  $m = 2^k$

# Descente de gradient des mini-batchs: Inconvénients

---

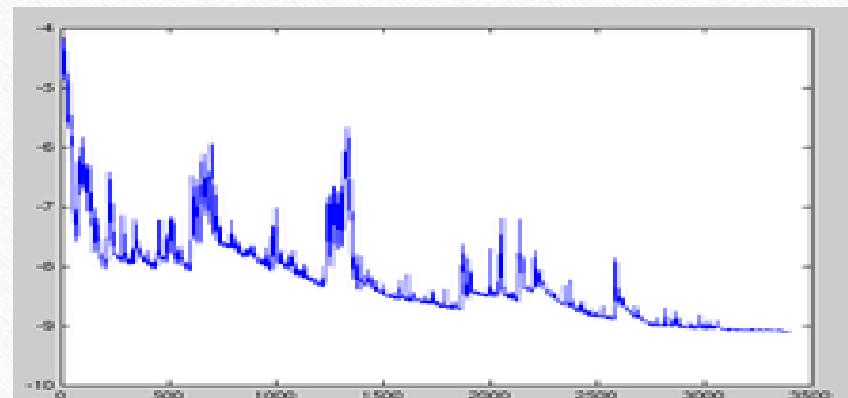
- Variance → fluctuation
- Taux d'apprentissage a une grande influence:

Lorsque l'on approche d'un optimum, la convergence est plus hasardeuse comparée à la méthode classique de descente de gradient (des lots différents).

On peut corriger cela en réduisant le learning rate.

*trop grand* peut faire diverger l'algorithme tandis qu'un taux

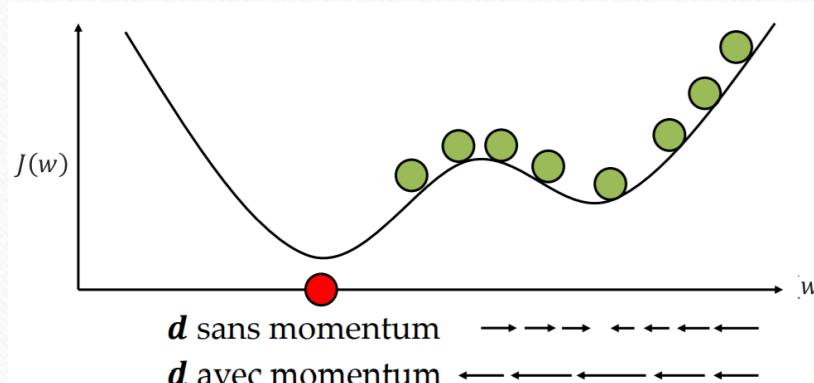
*trop petit* ralentit la convergence.



# Descente de gradient avec Momentum

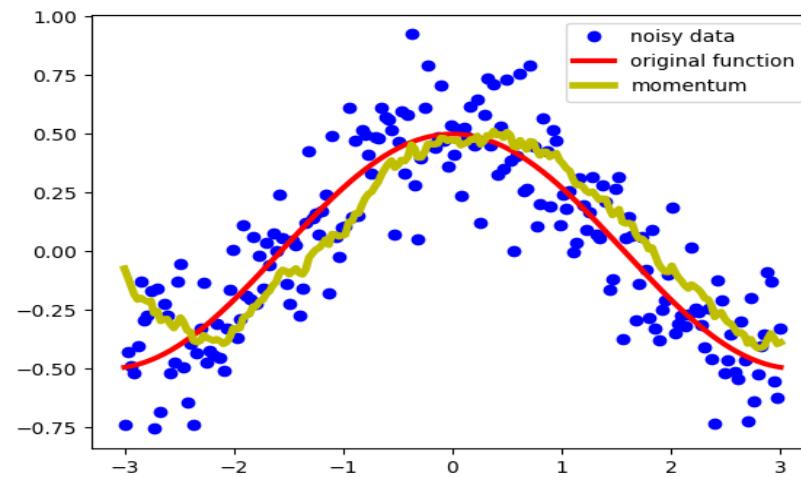
**Concept:** Ajouter à la descente de gradient une dynamique Newtonienne

On ajoute une technique à la méthode SGD pour réduire son bruit élevé. On accélère ainsi la convergence vers la direction pertinente et on réduit la fluctuation provenant des directions non pertinentes.



# Descente de gradient avec Momentum

- Cette technique consiste à pondérer le gradient courant par la pondération des gradients précédents afin de lisser des changements de direction trop importants. Pour cela, on utilise un nouvel hyperparamètre appelé **le momentum**.



# Descente de gradient avec Momentum: Moyennes pondérées de manière exponentielle

- Vélocité  $v \in \mathbb{R}^D$

Estimation des directions précédentes.

- Facteur d'oubli  $\beta \in [0, 1[$

$\beta \rightarrow 0 \equiv$  SGD standard

(Valeur suggérée:  $\beta = 0.9$ )

- Les moyennes pondérées exponentiellement de  $1/(1-\beta)$  points

$$v_t \leftarrow \beta v_{t-1} + (1 - \beta) s_t$$

- $v_t$ : Ceci représente le terme de quantité de mouvement à l'itération  $t$ . C'est un vecteur qui accumule les gradients passés.
- $\beta$ : Il s'agit du coefficient de quantité de mouvement, un hyperparamètre généralement compris entre 0 et 1. Il contrôle la contribution des gradients passés au terme de quantité de mouvement actuel. Une valeur plus élevée de  $\beta$  signifie que plus de poids est accordé aux gradients passés.
- $s_t$ : Ceci représente le gradient actuel (ou son estimation stochastique) à l'itération  $t$ . Il est généralement calculé à l'aide d'un sous-ensemble de données d'entraînement (un mini-lot)

# Descente de gradient avec Momentum: Moyennes pondérées de manière exponentielle

- Vélocité  $v \in \mathbb{R}^D$

Estimation des directions précédentes.

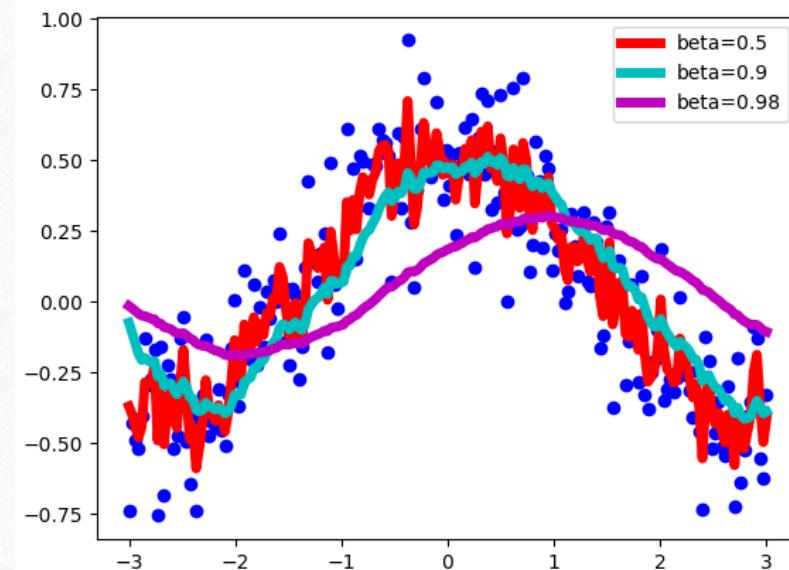
- Facteur d'oubli  $\beta \in [0, 1[$

$\beta \rightarrow 0 \equiv$  SGD standard

(Valeur suggérée:  $\beta = 0.9$ )

- Les moyennes pondérées exponentiellement de  $1/(1-\beta)$  points

$$v_t \leftarrow \beta v_{t-1} + (1 - \beta) S_t$$



*momentum - données provenant de moyennes pondérées de manière exponentielle.*

# Descente de gradient avec Momentum

---

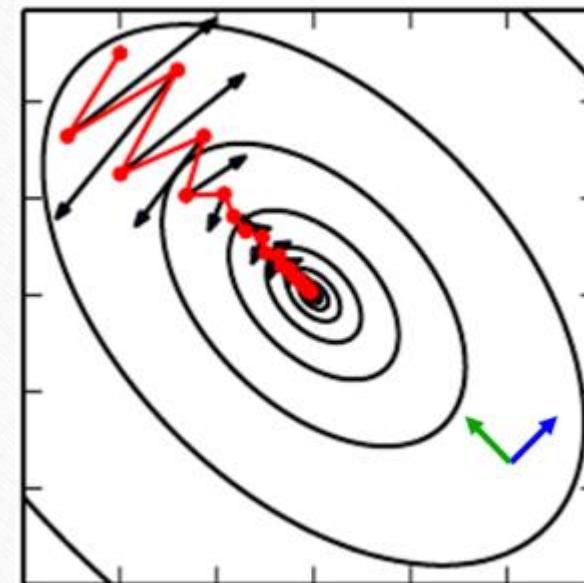
- **Algorithme:**
1. Échantillonnage :  $(x^{(i)}, y^{(i)}) \quad 1 \leq i \leq p.$
  2. Gradient :  $d = \hat{g} = \frac{1}{p} \sum_{i=1}^p d L(\hat{y}^{(i)}, y^{(i)}) / dw$
  3. Vélocité (momentum) :  $v_t \leftarrow \beta v_{t-1} + (1 - \beta) * d$
  4. Mise à jour :  $w = w - \eta v_t$

# Descente de gradient avec Momentum: Avantages

---

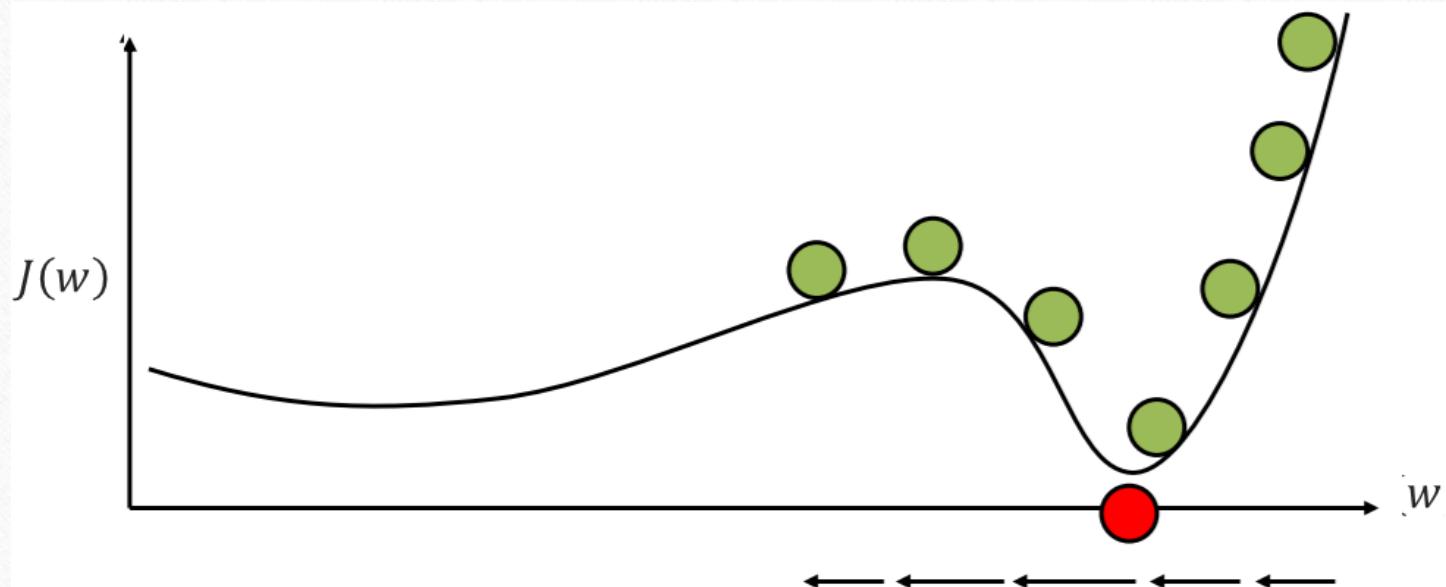
- Momentum aide quand il y a plusieurs optimums locaux
- Les MAJ de  $w$  tendent à s'aligner.

**rouge** = avec momentum  
**noir** = sans momentum



# Descente de gradient avec Momentum: Problème

- Peut survoler les min / max locaux. Mais, peut dépasser le min global.



# RMSprop

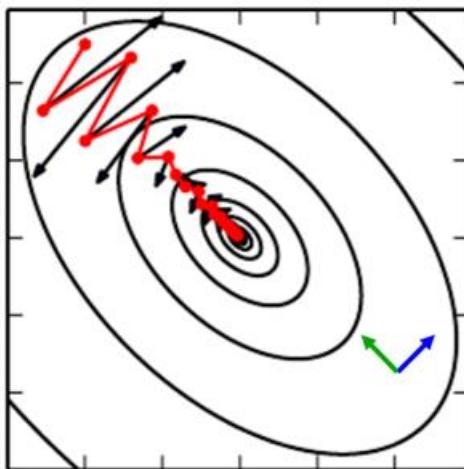
---

- **Algorithme:**

1. Échantillonnage :  $(x^{(i)}, y^{(i)}) \quad 1 \leq i \leq p.$
2. Gradient :  $d = \hat{g} = \frac{1}{p} \sum_{i=1}^p d L(\hat{y}^{(i)}, y^{(i)}) / dw$
3. Accumulateur :  $S_t \leftarrow \beta S_{t-1} + (1 - \beta) * d^2$
4. Mise à jour :  $w = w - \eta \frac{d}{\sqrt{S_t}}$

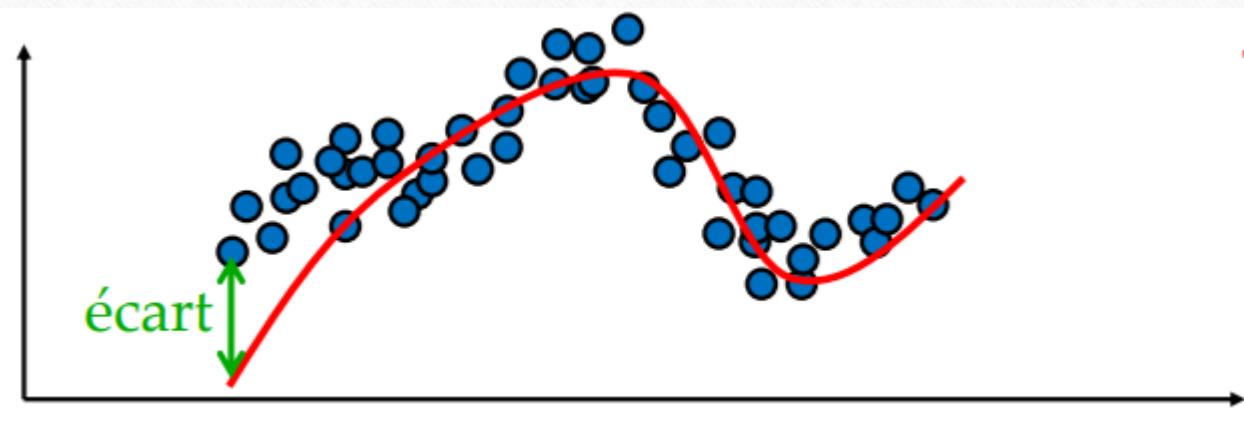
# RMSprop: Avantages

- Si  $S_t$  croit alors diminuer la rapidité de mise à jour (oscillations verticales).
- Si  $S_t$  décroît alors croître la rapidité de mise à jour (oscillations horizontales).



# RMSprop: Problème

- Cet estimateur a un large biais positif au début de la descente du gradient,



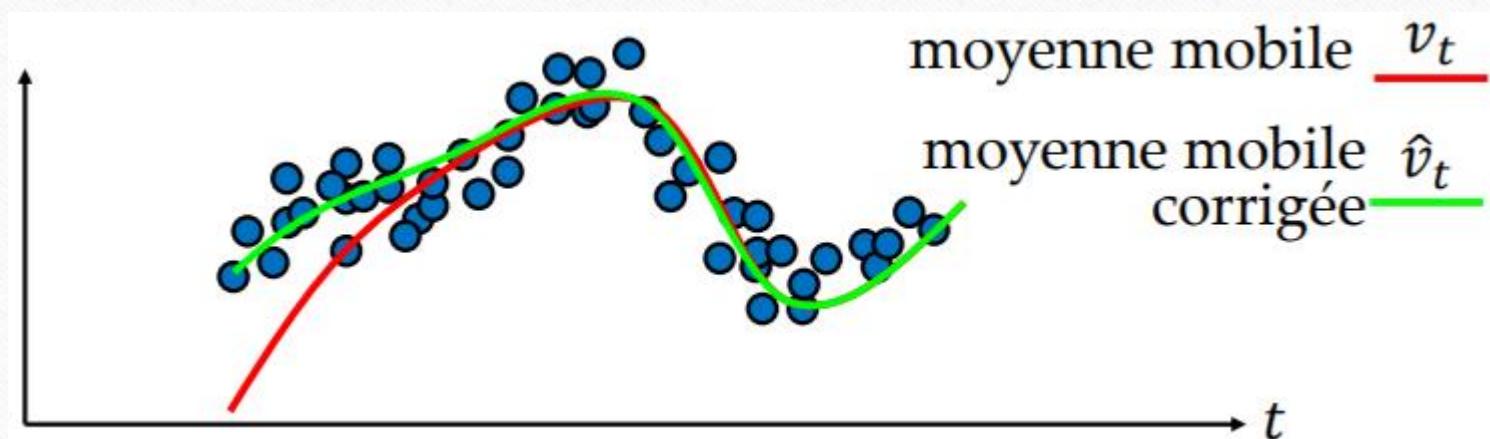
# Algorithme d'optimisation Adam

---

- Solution pour résoudre les inconvénients de RMSprop + momentum
- Méthode :
  - Calculer une estimation des premier et second moments avec une moyenne mobile exponentielle à taux  $\beta_1, \beta_2 \in [0, 1[$ .
    - Valeurs suggérées:  $\beta_1 = 0.9$  et  $\beta_2 = 0.999$
  - Corriger les biais des moments.
  - Le premier moment normalisé par le second moment donne la direction de MAJ.

# Algorithme d'optimisation Adam

- Correction du biais de départ



# Algorithme d'optimisation Adam

---

- **Algorithme:**

1. Échantillonnage :  $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \quad 1 \leq i \leq p.$
2. Gradient :  $\mathbf{d} = \widehat{\mathbf{g}} = \frac{1}{p} \sum_{i=1}^p \mathbf{d} L(\widehat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) / d\mathbf{w}$
3. 1er moment:  $\mathbf{v} \leftarrow \beta_1 \mathbf{v} + (1 - \beta_1) * \mathbf{d}$
4. 2ème moment:  $S \leftarrow \beta_2 S + (1 - \beta_2) * \mathbf{d}^2$
5. Correction biais 1er moment:  $\mathbf{v} \leftarrow \frac{\mathbf{v}}{1 - (\beta_1)^t}$
6. Correction biais 2ème moment:  $S \leftarrow \frac{S}{1 - (\beta_2)^t}$
7. Mise à jour :  $\mathbf{w} = \mathbf{w} - \eta \frac{\mathbf{v}}{\sqrt{S} + \epsilon}$

# Algorithme d'optimisation Adam

---

- Choix des hyper-paramètres:
  1.  $\eta$  à régler
  2.  $\beta_1 = 0,9$
  3.  $\beta_2 = 0,99$
  4.  $\varepsilon = 10^{-8}$

Merci pour votre attention !

Dr. Sana Hamdi

Maitre Assistante en Informatique à l'INSAT

Membre du Laboratoire LIPAH (FST-Tunisie)  
et du Laboratoire SAMOVAR (Telecom SudParis-France)