Réseaux de neurones artificiels

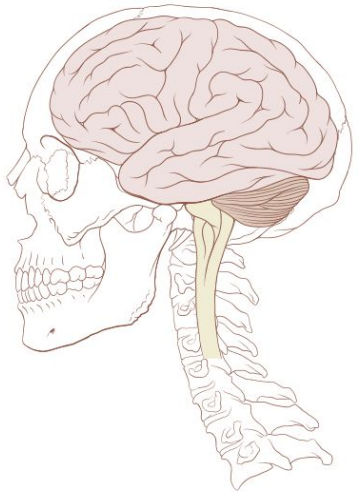
Amira Barhoumi

amira.barhoumi@univ-grenoble-alpes.fr

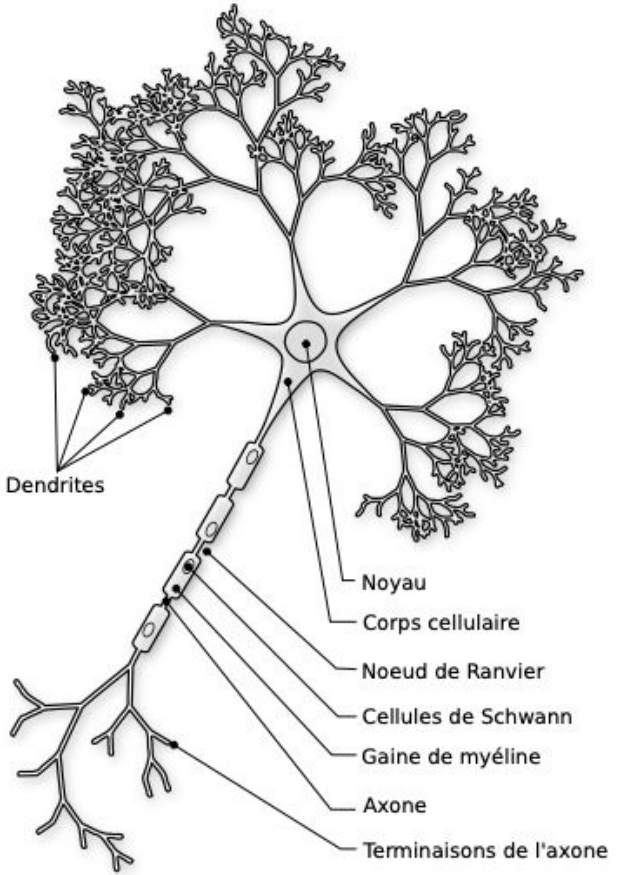
Année universitaire : 2025-2026

Introduction

- Cerveau humain
  - environ 100 milliards de neurones
  - au moyenne 10 milles connexions par neurone
  - 1mm^3 de cortex contient 1 milliard de connexions

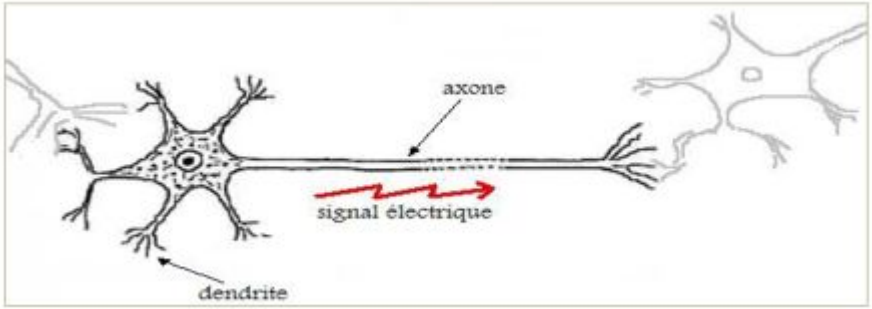


- **Neurone biologique**
  - Cellule capable de transmettre des informations à d'autres neurones via des connexions (synapses)
  - Il existe différents types de neurones avec des fonctionnements différents (sensoriel, moteur, inter-neurones, etc)
  - Les neurones sont interconnectés et forment un réseau

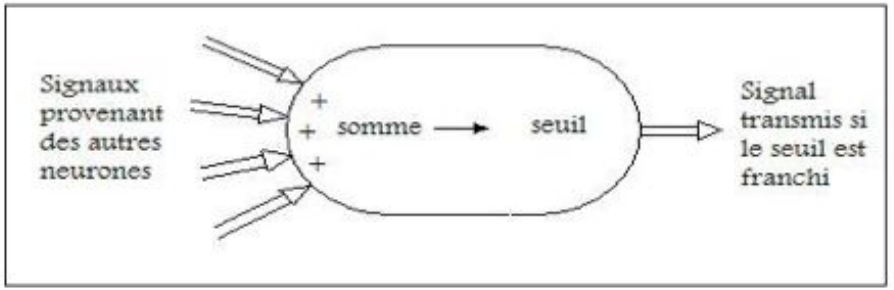


- **Neurone formel**
  - Une fonction algébrique bornée dont la valeur dépend de paramètres appelés poids (ou coefficients). Les variables de cette fonction sont habituellement appelées entrées (inputs), et la valeur de la fonction est appelée sortie (output).
  - Les neurones artificiels les plus fréquemment utilisés sont ceux dont la fonction est calculée en deux étapes :
    - calcul de la somme des entrées pondérées par les poids du neurone
    - calcul d'une fonction d'activation (fonction de transfert) pour déterminer l'output du neurone

Introduction



Neurone biologique



Neurone artificiel

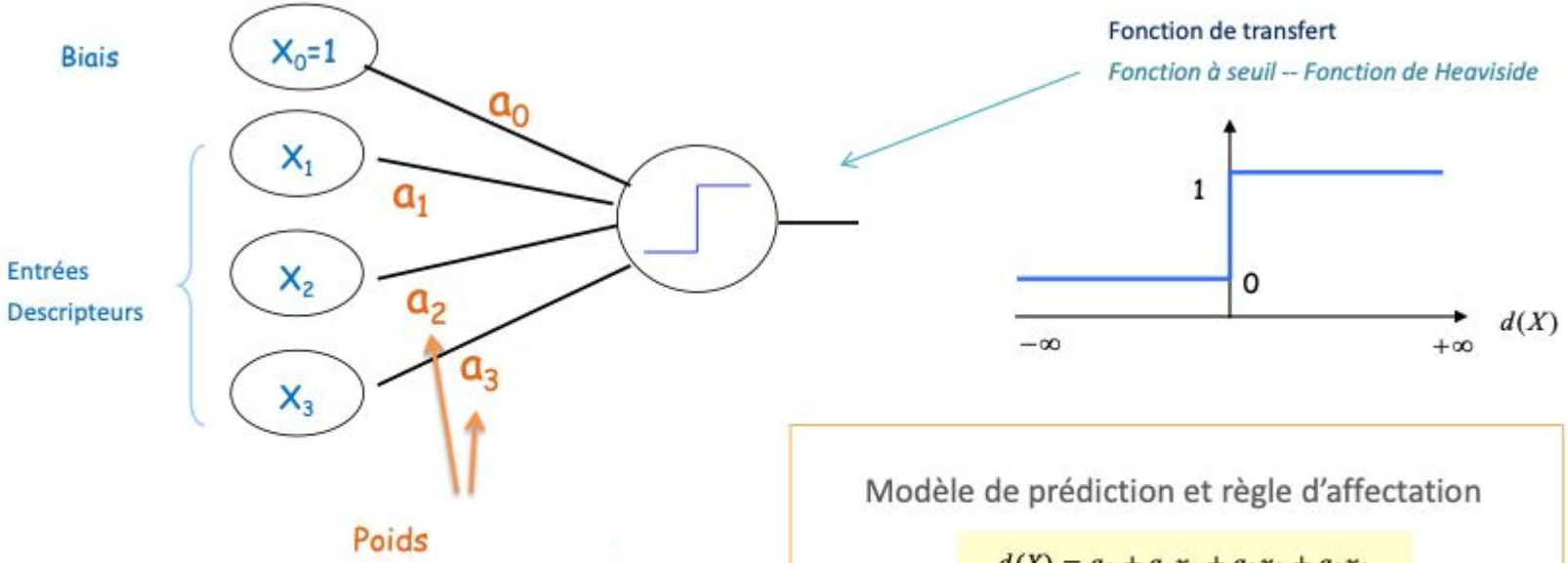
Transmission de l'information et apprentissage (étapes clés) :

- Réception de l'information (signal)
- Traitement + activation par chaque neurone
- Transmission aux autres neurones (si seuil franchi)
- À long terme : renforcement de certains liens → Apprentissage

Introduction

- Historique :
  - Mac Culloch et Pitts (1943) : définition d'un neurone formel
  - Loi de Hebb (1949)
  - Rosenblatt (1958) Widrow et Hoff : perceptron
  - Minsky et Papert (1969) : limites des perceptrons
  - Rumelhart - Mc Clelland (1980), Werbos - Le Cun : perceptron multi-couches, mécanismes d'apprentissage automatique (rétro-propagation du gradient)

- **Neurone formel**

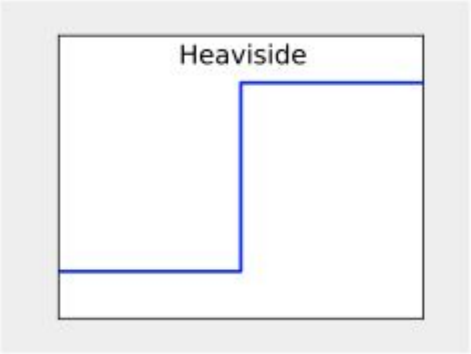


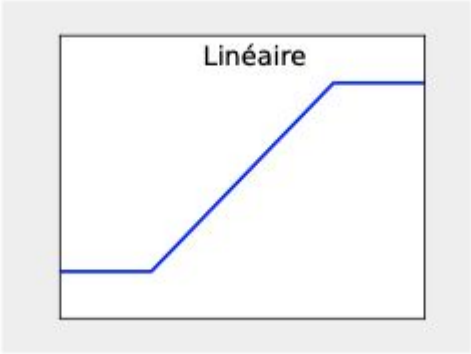
Modèle de prédiction et règle d'affectation

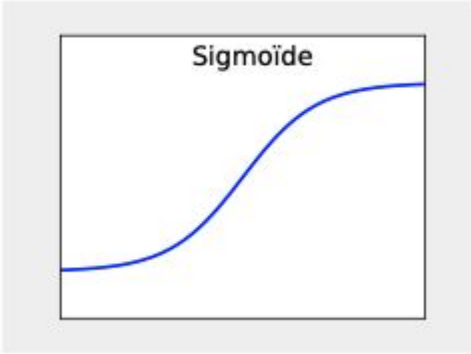
$$d(X) = a_0 + a_1x_1 + a_2x_2 + a_3x_3$$

Si $d(X) > 0$ Alors $Y = 1$ Sinon $Y = 0$

- **Fonctions d'activation**







Heaviside (seuil θ)

- Si $x < \theta$ alors f(x) $= 0$
- Si $x \geq \theta$ alors f(x) $= 1$

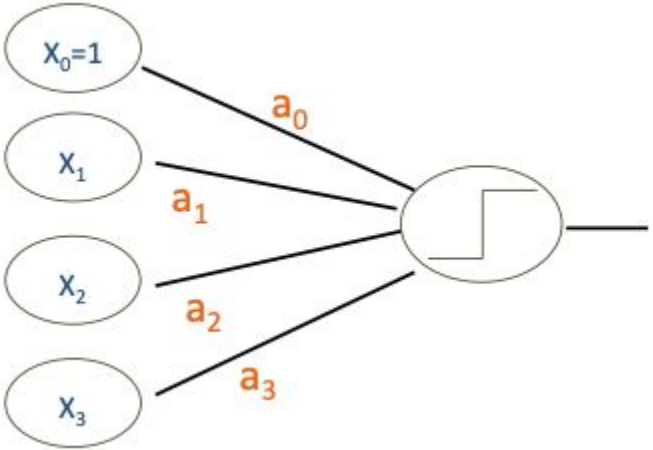
Linéaire (seuil θ_1, θ_2)

- Si $x < \theta_1$ alors f(x) $= 0$
- Si $x > \theta_2$ alors f(x) $= 1$
- Si $\theta_1 \leq x \leq \theta_2$ alors f(x) $= x$

Sigmoïde / Tangente hyperbolique

- $f(x) = \frac{1}{1+\exp(-x)}$
- $f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$

- Apprentissage à partir des données :



Comment calculer les poids à partir du corpus d'apprentissage $\{(x_1, x_2, x_3, y)\}$

Quel critère optimisé? → minimiser l'erreur de prédiction
Comment procéder à l'optimisation? → principe de l'incrémentalité (online)

Perceptron

- Un neurone formel prend des entrées $X = (x_1, x_2, ..., x_n)$ auxquelles sont associées des poids $W = (\omega_1, \omega_2, ..., \omega_n)$ qui reflètent l'importance de l'information véhiculée x_i où $1 <= i <= n$.

- Ce neurone prend en entrée également un biais (b) ⇒ ajouter de la flexibilité au réseau en agissant sur la position de la frontière de décision

- Le neurone fournit en sortie y qui correspond à l'application d'une fonction d'activation φ sur la somme des vecteurs d'entrée X pondérés par le vecteur de poids W et le biais

$$y = \varphi(W.X + b)$$
$$= \varphi(\sum_{i=1}^{n} \omega_i x_i + b)$$

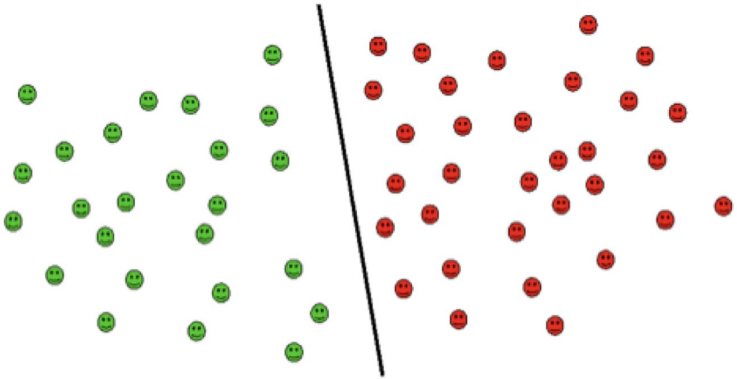
Perceptron

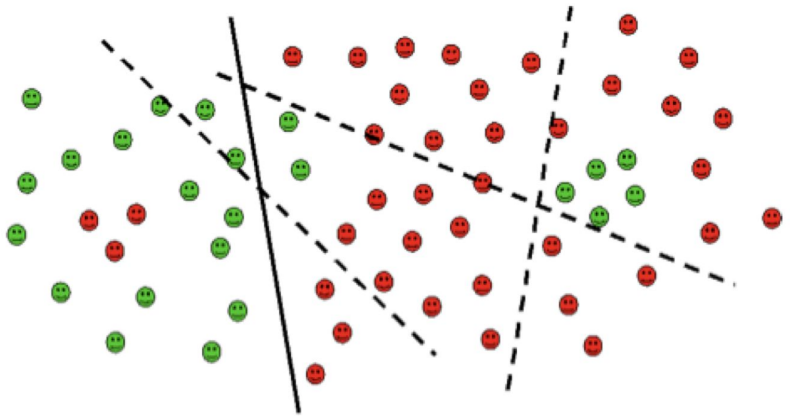
- Apprendre les poids d'un neurone (formel) quand les exemples sont linéairement séparables

- Méthode d'apprentissage itérative :
  - On choisit une droite D au hasard
  - Si D permet une classification parfaite, alors arrêt d'apprentissage
  - Sinon, tant que D fait des erreurs de classification
    - évaluer la qualité de la séparation que D permet de réaliser
    - mettre à jour les coefficients de D pour améliorer la séparation (règle delta, règle du perceptron)

Perceptron

- Données linéairement séparables : tous les points associés peuvent être séparés correctement par une frontière linéaire (hyperplan)



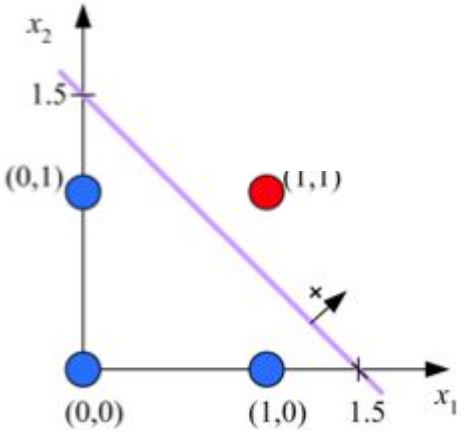


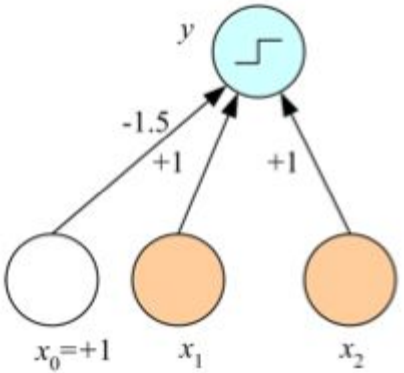
pas de séparation linéaire

- **Apprentissage de la fonction booléenne ET**

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Données





- **Apprentissage de la fonction booléenne ET**

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Données

Principales étapes :

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

Une observation peut passer plusieurs fois !

- **Apprentissage de la fonction booléenne ET**

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Données

Initialisation aléatoire des poids : $a_0 = 0.1; a_1 = 0.2; a_2 = 0.05$

Frontière :

$0.1 + 0.2x_1 + 0.05x_2 = 0 \Leftrightarrow x2 = -4.0x_1 - 2.0$

Règle de mise à jour des poids

Pour chaque individu que l'on fait passer

(Principe de l'incrémentalité)

$$a_j \leftarrow a_j + \Delta a_j$$

S'assurer que les variables sont sur la même échelle

avec

Force du signal

$$\Delta a_j = \eta\ (y - \hat{y})\ x_j$$

Taux d'apprentissage

Détermine l'amplitude de l'apprentissage

Quelle est la bonne valeur ?

Trop petit → lenteur de convergence

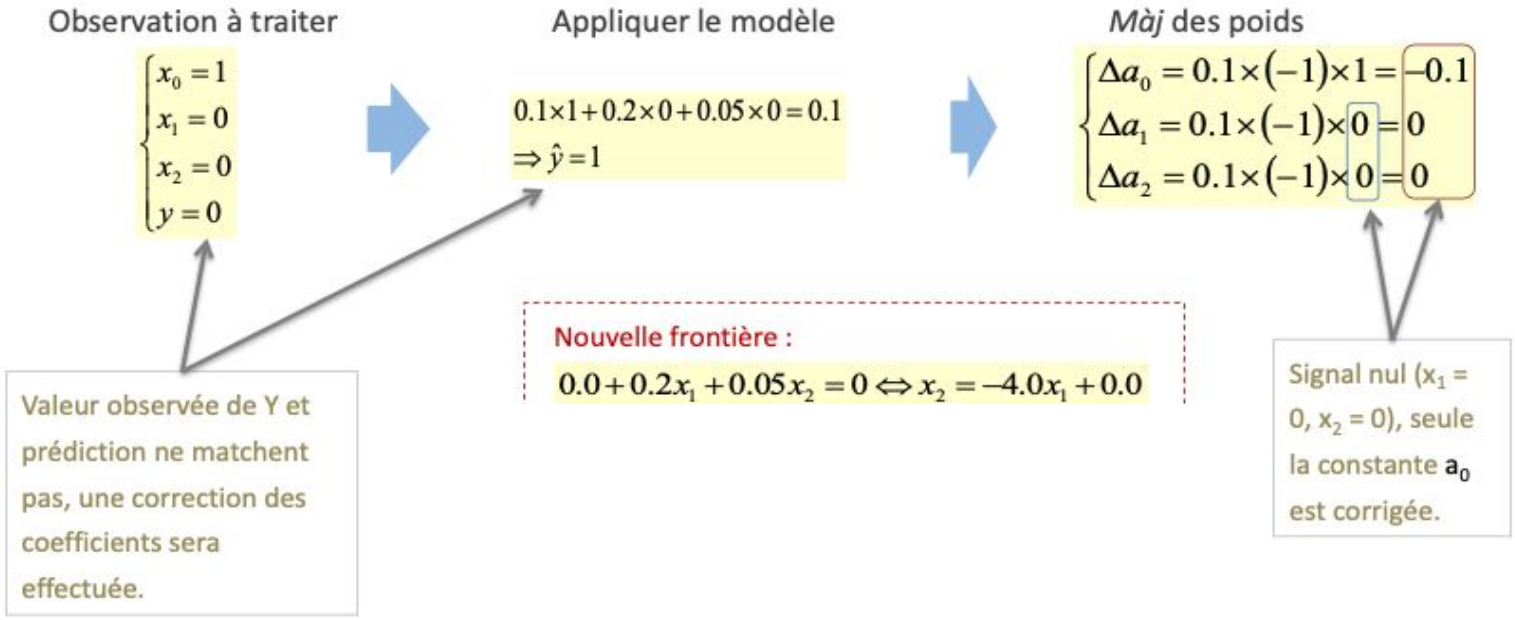
Trop grand → oscillation

En général autour de 0.05 ~ 0.15 (0.1 dans notre exemple)

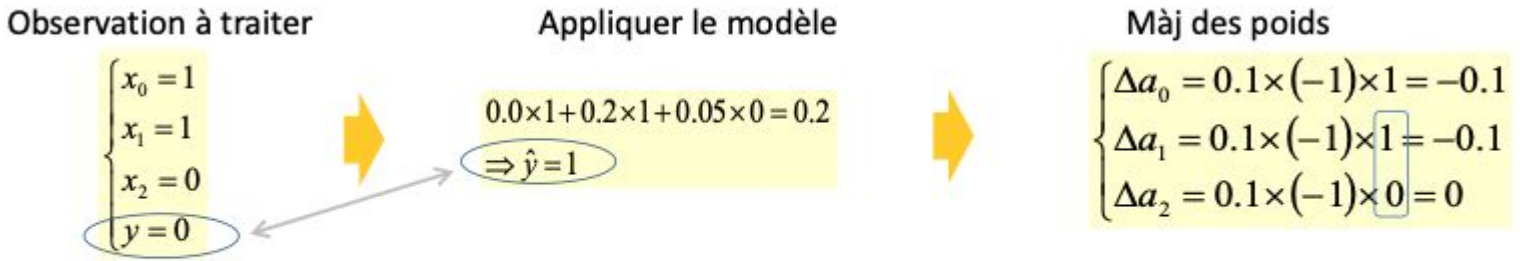
Erreur

Détermine s'il faut réagir (corriger) ou non

- **Apprentissage de la fonction booléenne ET**



- **Apprentissage de la fonction booléenne ET**



Nouvelle frontière :

$$-0.1 + 0.1x_1 + 0.05x_2 = 0 \Leftrightarrow x_2 = -2.0x_1 + 2.0$$

- **Apprentissage de la fonction booléenne ET**

Observation à traiter

$$\begin{cases} x_0 = 1 \\ x_1 = 0 \\ x_2 = 1 \\ y = 0 \end{cases}$$

Appliquer le modèle

$$-0.1\times 1 + 0.1\times 0 + 0.05\times 1 = -0.05$$

$$\Rightarrow \hat{y} = 0$$

Màj des poids

$$\begin{cases} \Delta a_0 = 0.1\times(0)\times 1 = 0 \\ \Delta a_1 = 0.1\times(0)\times 0 = 0 \\ \Delta a_2 = 0.1\times(0)\times 1 = 0 \end{cases}$$

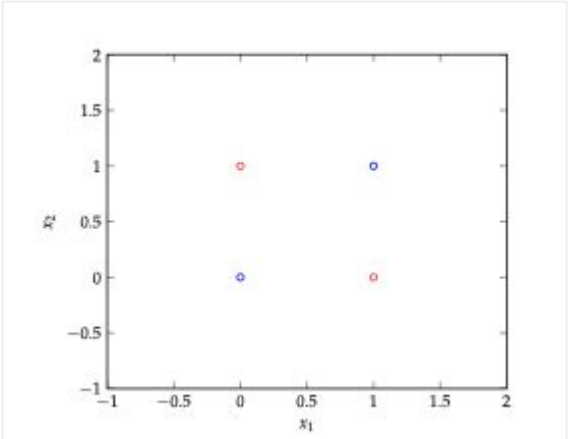
Frontière inchangée :

$$-0.1 + 0.1x_1 + 0.05x_2 = 0 \Leftrightarrow x_2 = -2.0x_1 + 2.0$$

Exemple historique du XOR

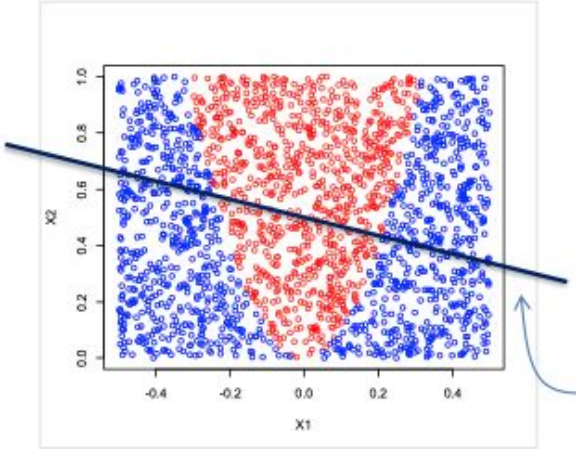
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

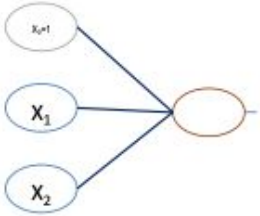
Données



Non séparable linéairement
(Minsky & Papert, 1969)

Un perceptron simple ne sait traiter que les problèmes linéairement séparables.



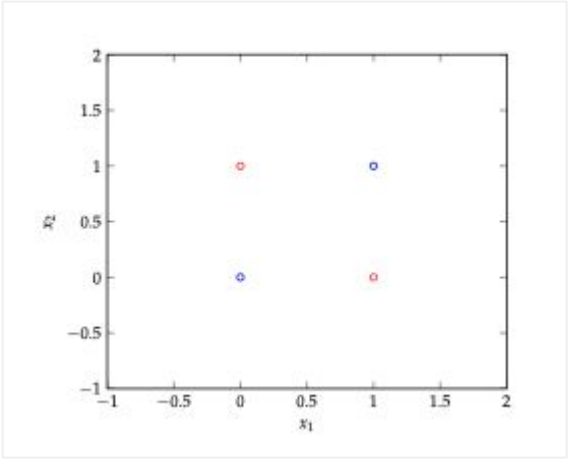


Trouver une droite de séparation efficace n'est pas possible.

Exemple historique du XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Données

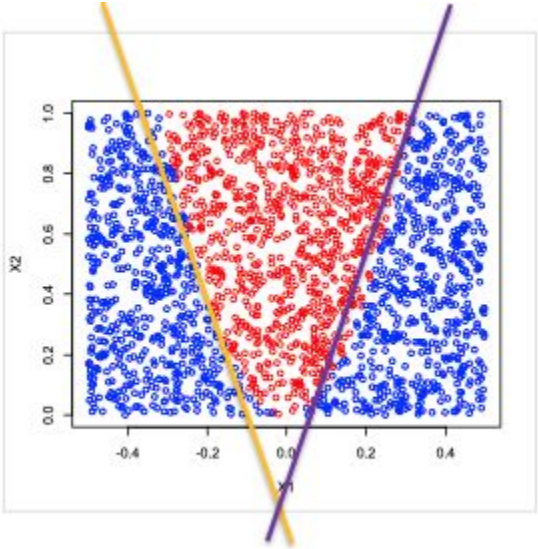


Non séparable linéairement

(Minsky & Papert, 1969)

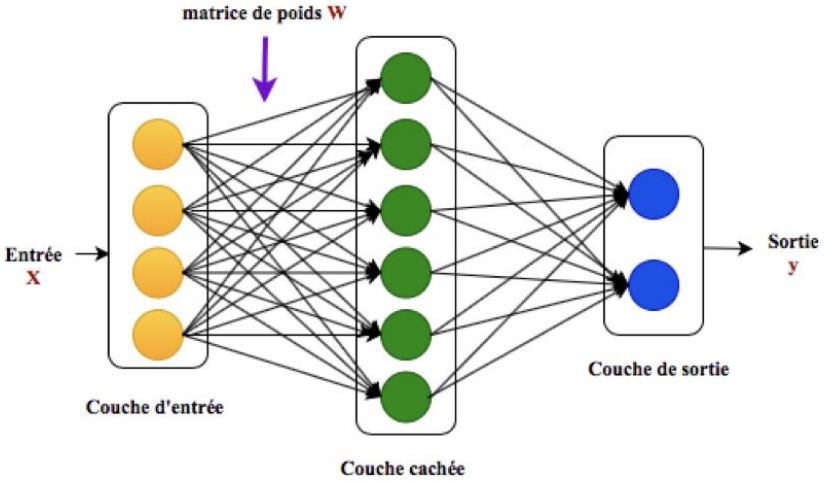
- On ne peut pas séparer les deux classes à l'aide d'une droite
- On pourrait appliquer une transformation f aux données de manière à les rendre linéairement séparables
- L'application de la transformation permet d'obtenir une nouvelle représentation des données
- La fonction de transformation f ne peut pas être linéaire

Perceptron multicouches (MLP)



Perceptron Multicouche (PMC)

Une combinaison de séparateurs linéaires permet de produire un séparateur global non-linéaire (Rumelhart, 1986).



- MLP propage l'information de la couche d'entrée aux couches cachées et enfin la couche de sortie.
- Le score le plus élevé sera retenu et la classe correspondante sera prédite.

Perceptron multicouches (MLP)

- Apprentissage par rétropropagation du gradient :
  - paramètres du réseaux : matrices de poids et biais
  - Fonction d'activation : fonction sigmoïde pour la classification binaire sinon fonction softmax
  - Minimiser la différence entre l'hypothèse et la référence ⇒ fonction de coût (*loss* function)
  - Fonction de coût non linéaire

exemple: - Erreur quadratique moyenne (*Mean square error*) $L = \frac{1}{n}\sum_{i=1}^{n}(s_i - y_i)^2$

- Entropie croisée (*Cross Entropy*) $L = \frac{1}{n}\sum_{i=1}^{n}(s_i log(y_i) - (1 - s_i) log(1 - y_i))$

Perceptron multicouches (MLP)

- Apprentissage par rétropropagation du gradient :
  - Phase “propagation avant” (*forward pass*) :
    - calculer la fonction de coût **L** entre les hypothèses et les références
  - Phase “propagation arrière” (*backword pass*) :
    - rétro-propager la dérivée partielle du coût par rapport aux poids W du réseau
    - mettre à jour en fonction de cette fonction cette dérivée partielle

$$W' = W - \Delta W$$

$$\Delta W = -\eta \frac{\partial L}{\partial W}$$

η = taux d’apprentissage (*learning rate*) : un hyper-paramètre déterminant la vitesse de convergence d’apprentissage

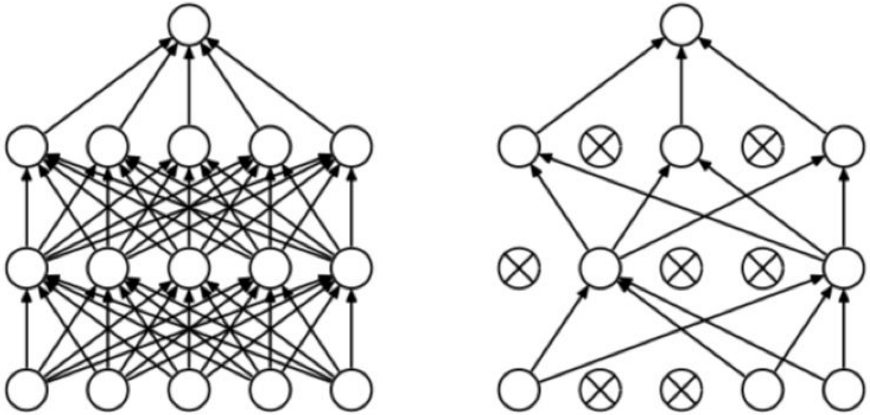
Perceptron multicouches (MLP)

- Apprentissage par rétropropagation du gradient :
  - Mélanger les exemples d'apprentissage
    - convergence lente si beaucoup d'exemples consécutifs de la même classe
    - mélanger aléatoirement pour éviter la corrélation entre les exemples consécutifs
  - Taux d'apprentissage (*learning rate*) : un hyper-paramètre déterminant la vitesse de convergence d'apprentissage
    - positif
    - fixe ou variable (au fil des itérations/époques)
  - Trois stratégies de rétro-propagation :
    - rétro-propagation stochastique : mettre à jour les poids à chaque présentation d'un exemple d'apprentissage
    - rétro-propagation par lot : mettre à jour les poids selon la moyenne de la fonction de coût sur tous les exemples d'apprentissage qui constituent un lot
    - rétro-propagation par mini-lot

Perceptron multicouches (MLP)

- Apprentissage par rétropropagation du gradient :
  - Une itération est une mise à jour des poids
  - Une époque est l'application de l'algorithme sur tous les exemples d'apprentissage
  - Nombre d'époques : un hyper-paramètre garantissant la convergence (L nulle)
    - positif
    - assez grand pour converger ⇒ temps d'apprentissage lent + éventuellement un sur-apprentissage
    - patience en nombre d'époque conditionne l'arrêt de l'apprentissage : si le système ne s'améliore pas sur un corpus de validation en un nombre d'époques égal à la patience, alors il arrête l'apprentissage. Le compteur de la patience est réinitialisé après chaque amélioration.

- Apprentissage par rétropropagation du gradient :
  - Méthodes pour éviter le sur-apprentissage :
    - **Régularisation** : L_1 (somme des valeurs absolues des poids) et L_2 (somme des carrés des poids)
    - **Dropout** : désactiver temporairement une partie des neurones et leur connexions pendant l'apprentissage



(a) Standard Neural Net

(b) After applying dropout.