APPRENTISSAGE AUTOMATIQUE

Chapitre 1:

Réseaux de Neurones Artificiels

Dr Faiza Titouna

Master1 IAM

INTRODUCTION

- L'apprentissage automatique (en anglais : machine learning) est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d' « apprendre » à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés[1] .
- Pour appliquer une méthode d'apprentissage, il est important de dispenser d'un échantillon d'exemples contenant un nombre important d'instances de données étiquetées ou non selon le mode d'emploi,
- Il existe plusieurs types d'algorithmes d'apprentissage:
  - Apprentissage supervisé : les classes de chaque exemple sont prédéterminées. Le processus se passe en deux phases. Lors de la première phase (d'*apprentissage*), il s'agit de déterminer un modèle à partir des données étiquetées. La seconde phase (de *test*) consiste à prédire l'étiquette d'une nouvelle donnée, connaissant le modèle préalablement appris.
  - Apprentissage non supervisé: On dispose d'exemples non étiquetés et le nombre de classes et leur nature n'ont pas été prédéterminées, on parle alors d'apprentissage non supervisé ou *clustering* en anglais. L'algorithme doit découvrir la structure plus ou moins *cachée* des données. Ces dernières sont ciblées selon leurs attributs disponibles, pour les classer en groupes *homogènes* d'exemples. La similarité est généralement calculée selon une fonction de distance entre paires d'exemples.
  - Apprentissage par renforcement: l'algorithme apprend un comportement selon une observation. L'action de l'algorithme sur l'environnement produit une valeur de retour qui guide l'algorithme d'apprentissage. L'algorithme de Q-learning est un exemple classique.

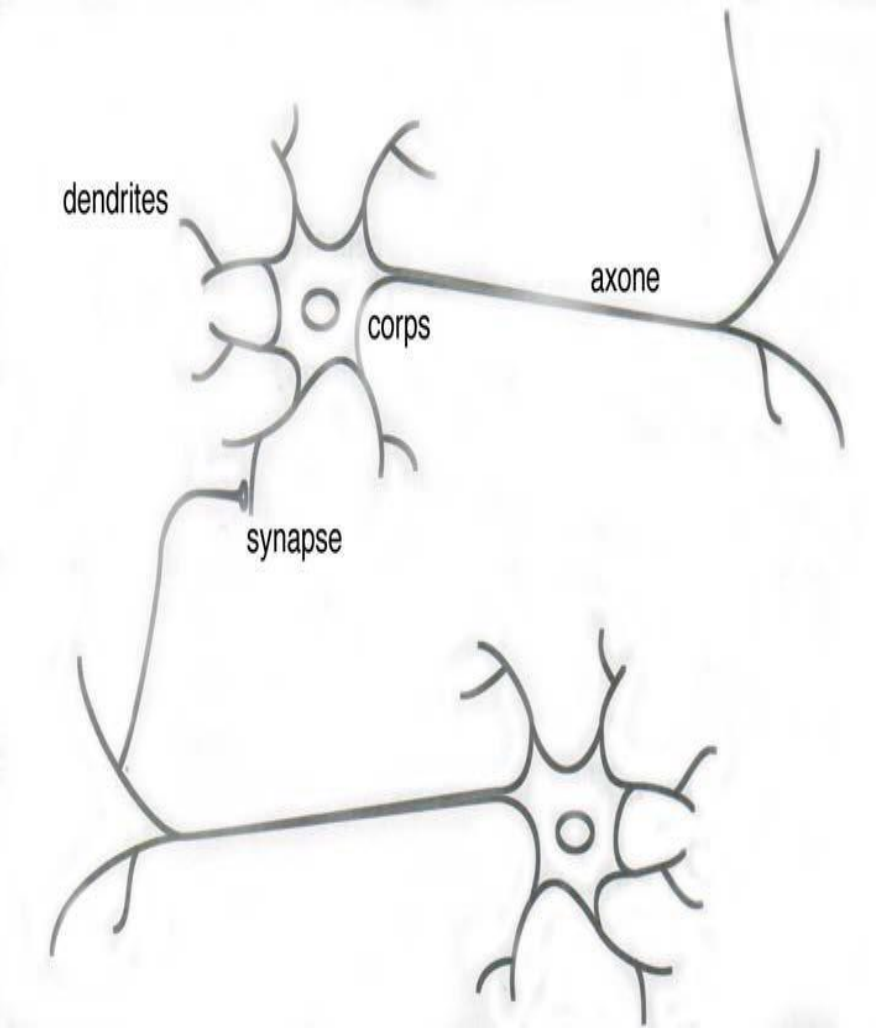
Historique

- 1943, premier modèle de neurones formels par deux bio-physiciens de Chicago : McCulloch & Pitts
- 1949, règle de Hebb : formulation du mécanisme d'apprentissage, sous la forme d'une règle de modification des connexions synaptiques.
- 1958, premier réseau de neurone artificiel (RNA), le Perceptron de Rosenblatt :
  - Une couche `perceptive' de neurones d'entrée
  - Une couche `décisionnelle' de neurones de sortie
  - Ce réseau parvient a apprendre à identifier des formes simples et à calculer certaines fonctions logiques.
- 1969, critique par Minsky & Papert : impossibilité de résoudre le problème de XOR (Ou exclusif)
- 1982 : Hopefield propose les RNA recurrents (`feed-back') :
  - Modélisation de processus mnésiques
  - Werbos initie les réseaux multicouche (MLP : Multi-layer Perceptron) avec retro-propagation des erreurs.
- 1986, Rumelhart propose et finalise les MLP,

Domaine d'applications

- Traitement des images
- Identification des signatures
- Reconnaissance des caractères (dactylos ou manuscrits)
- Reconnaissance de la parole
- Reconnaissance de signaux acoustiques (bruits sous-marins, …)
- Extraction d'un signal du bruit
- Robotique (apprentissage de tâches)
- Aide à la décision (domaine médical, bancaire, management, …)

Neurone biologique



Le système nerveux est composé de 10^{12} neurones interconnectés. Bien qu'il existe une grande diversité de neurones, ils fonctionnent tous sur le même schéma.

Ils se décomposent en trois régions principales :

- Le corps cellulaire
- Les dendrites
- L 'axone

Neurone biologique

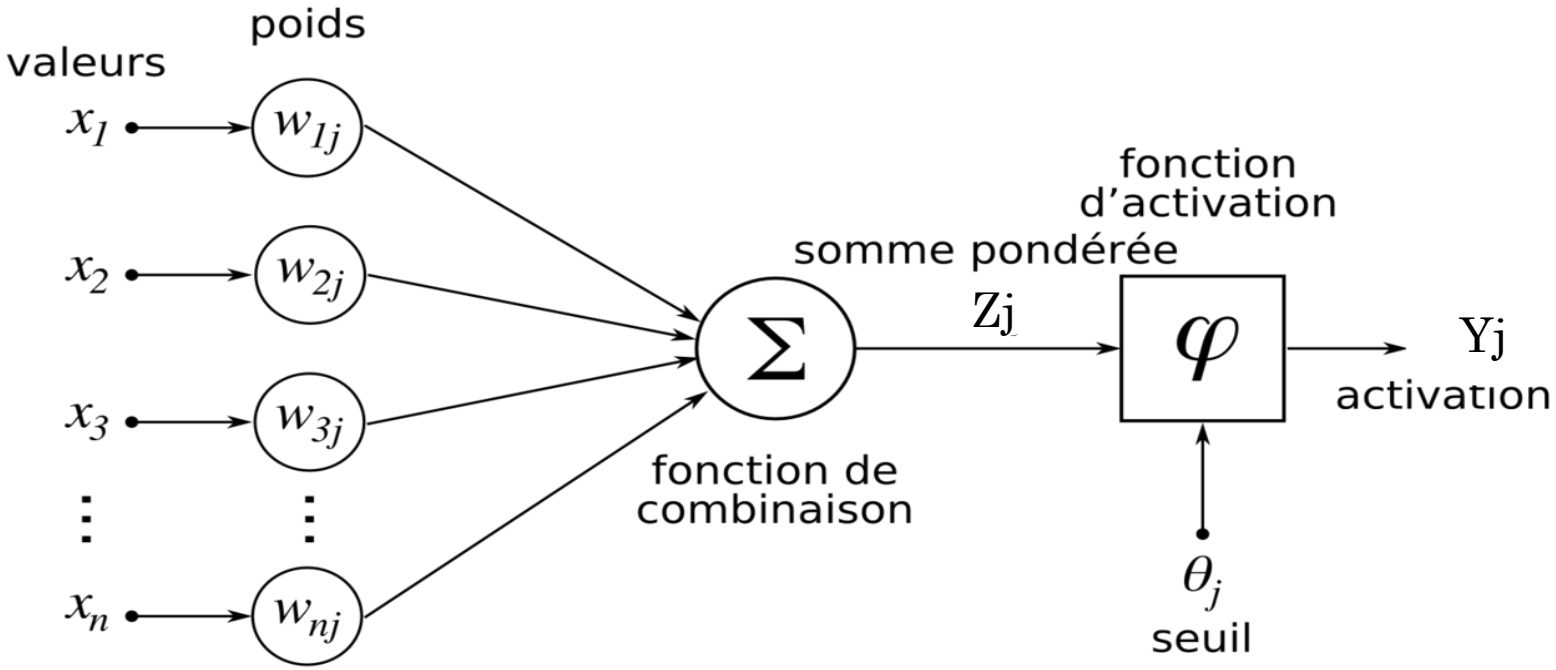
L'influx nerveux est assimilable à un signal électrique se propageant comme ceci :

- Les dendrites reçoivent l'influx nerveux d 'autres neurones.
- Le neurone évalue l'ensemble de la stimulation reçue.
- Si elle est suffisante, il est excité : il transmet un signal (0/1) le long de l 'axone.
- L'excitation est propagée jusqu'aux autres neurones qui y sont connectés via les synapses

NEURONE FORMEL

Soit un réseau de neurones formel avec n éléments dans la couche d'entrée, composée de n synapses appliquant un coefficient multiplicateur au signaux reçus. Une sommation, puis une comparaison a un seuil (θ) sont effectuées.

La Fonction de Transfert (fonction d'activation) détermine l'état du neurone (en sortie)



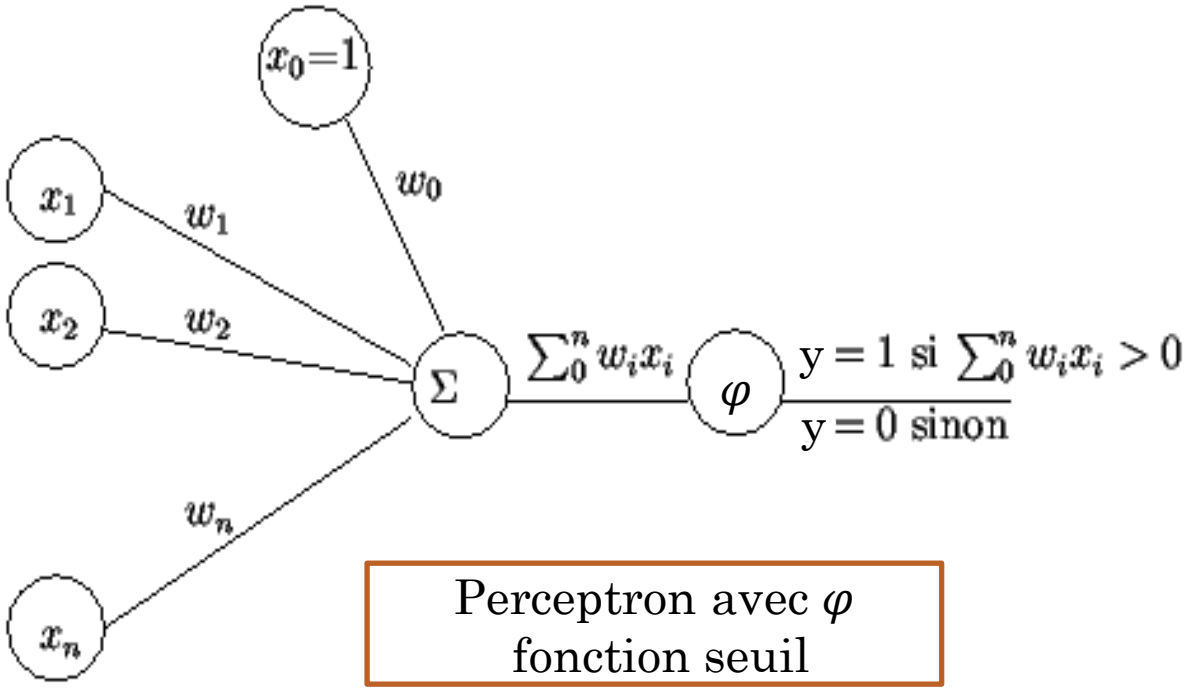
Que peut-on faire avec les réseaux de neurones ?

- Classification (ou discrimination): la variable expliquée (de sortie) est une variable nominale, chaque observation possède une classe
  - Exple: Une région d'une image représente un visage ou non ? Ici la classe est binaire (oui /non)
- Régression: la variable expliquée est une variable quantitative ($\in R$)
  - Exple: Quel sera le taux de personnes contaminées du coronavirus en Algérie dans un mois? Ici la classe est une valeur réelle

Perceptron

ARCHITECTURE D'UN PERCEPTRON

Le perceptron est formée d'une première **couche** d'**unités** (neurones) qui sont associées aux données : chaque unité correspond à une des variables d'entrées. Plus une **unité de biais** qui est toujours **activée** (égale à1). Ces unités sont reliées à une seule et unique **unité de sortie** . $y = \varphi(\sum_0^n w_i \cdot x_i)$ où φ fonction d'activation



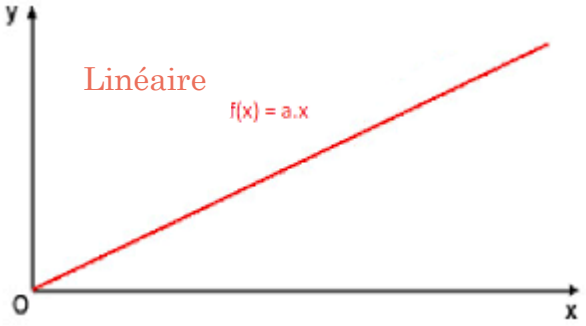
Perceptron avec φ fonction seuil

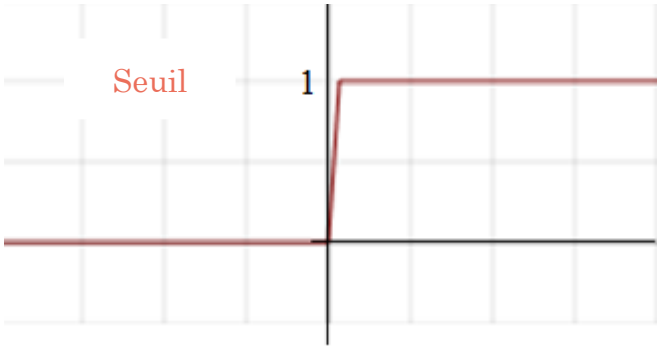
FONCTIONS D'ACTIVATIONS

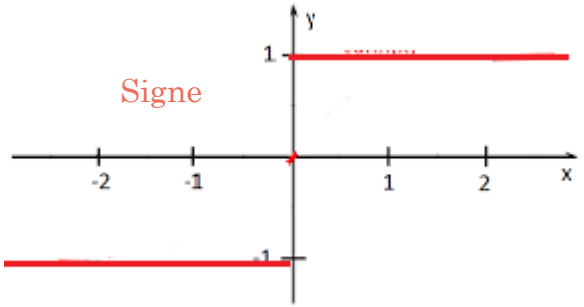
- La fonction d'activation opère une transformation d'une combinaison affine des signaux d'entrée, w_0, terme constant, étant appelé le biais du neurone et noté généralement par b. Cette combinaison affine est déterminée par un vecteur de poids $[w_0, w_1, \cdots, w_n]$ associé à chaque neurone et dont les valeurs sont estimées dans la phase d'apprentissage,

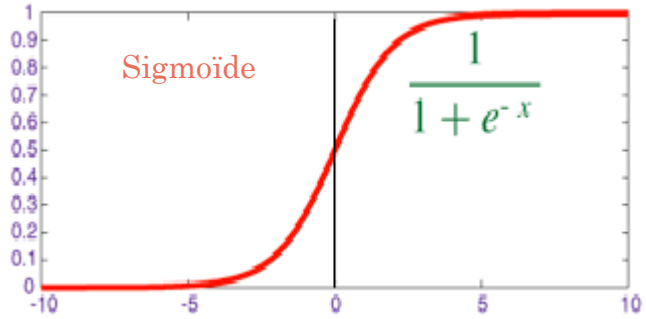
- Les fonctions d'activations les plus utilisées sont :
  - Linéaire : $\varphi(x) = a\,x$,
  - Seuil : $\varphi(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$
  - Signe : $\varphi(x) = \begin{cases} -1 & x < 0 \\ +1, & x \geq 0 \end{cases}$
  - Sigmoïde : $\varphi(x) = 1/(1 + e^{-x})$
  - Tanh : $\varphi(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

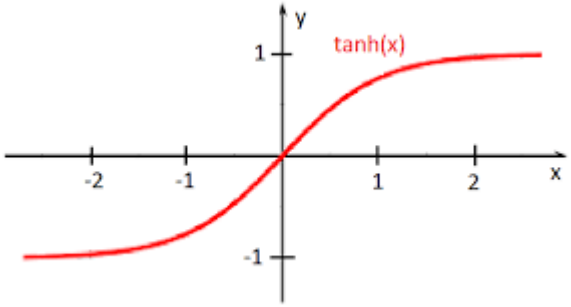
Fonctions d'activations









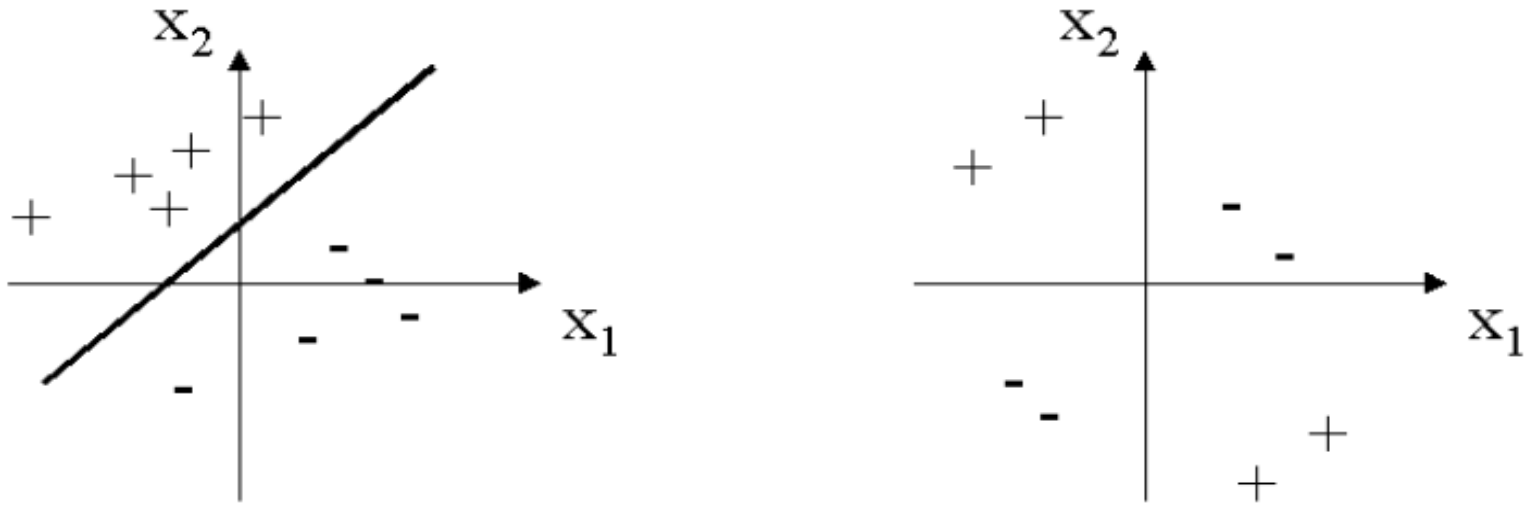


Séparation linéaire

Théorème : un perceptron à n entrées divise l'espace des entrées $\mathbb{R}^n$ en deux sous-espaces délimités par un hyperplan. Réciproquement, tout sous-ensemble linéairement séparable peut être discriminé par un perceptron,

Définition : Une surface de décision (frontière de décision) est une droite (1Dimension) ou plan (2D) ou hyperplan (> 3D),

Séparation linéaire



(a) (b)

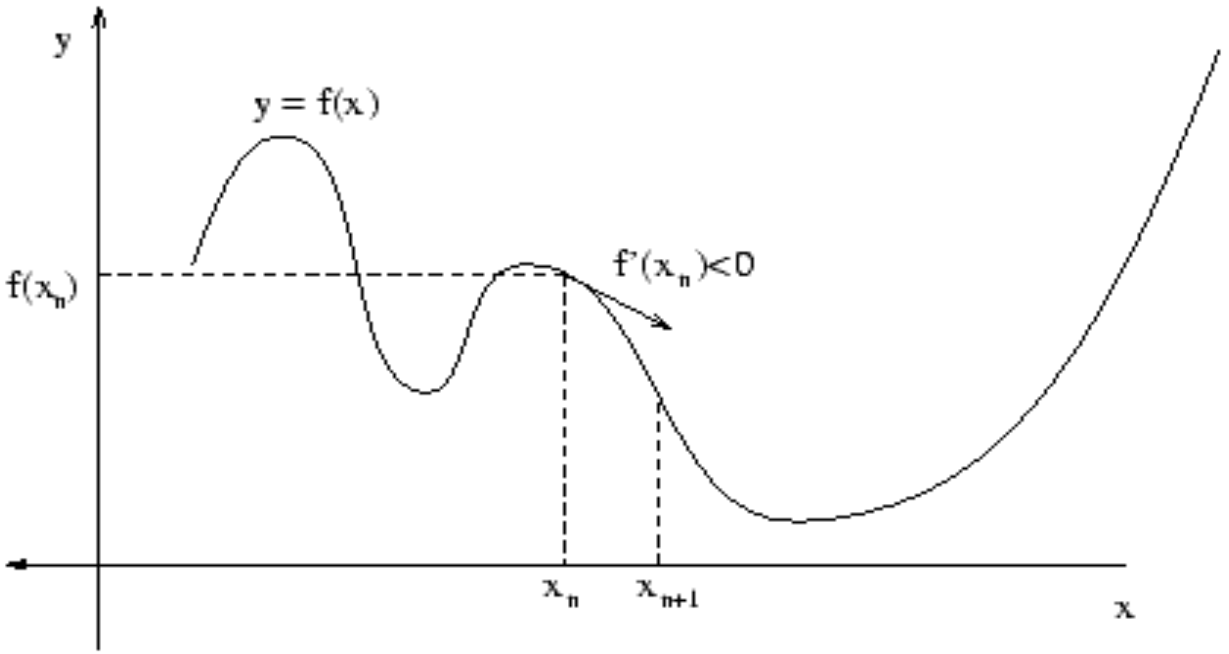
- Dans (a) les données sont séparées linéairement par une droite (fonction linéaire) en 2 classes,
- Dans (b) il faut au moins 2 droites pour séparer les données en 2 classes donc elles sont non linéairement séparables

MÉTHODE DE DESCENTE DU GRADIENT

Soit f une fonction d'une variable réelle à valeurs réelles, suffisamment dérivable dont on recherche un minimum. La méthode du gradient construit une suite qui doit s'approcher du minimum. Pour cela, on part d'une valeur quelconque x_0 et l'on construit la suite récurrente :

$$x_{n+1} = x_n + \Delta x_n \quad \text{où} \quad \Delta x_n = -\alpha f'(x_n)$$

avec α paramètre de convergence



Inconvénients

On remarque que x_{n+1} est d'autant plus éloigné de x_n que la pente de la courbe en est grande. On peut décider d'arrêter l'itération lorsque cette pente est suffisamment faible.

Les inconvénients bien connus de cette méthode sont :

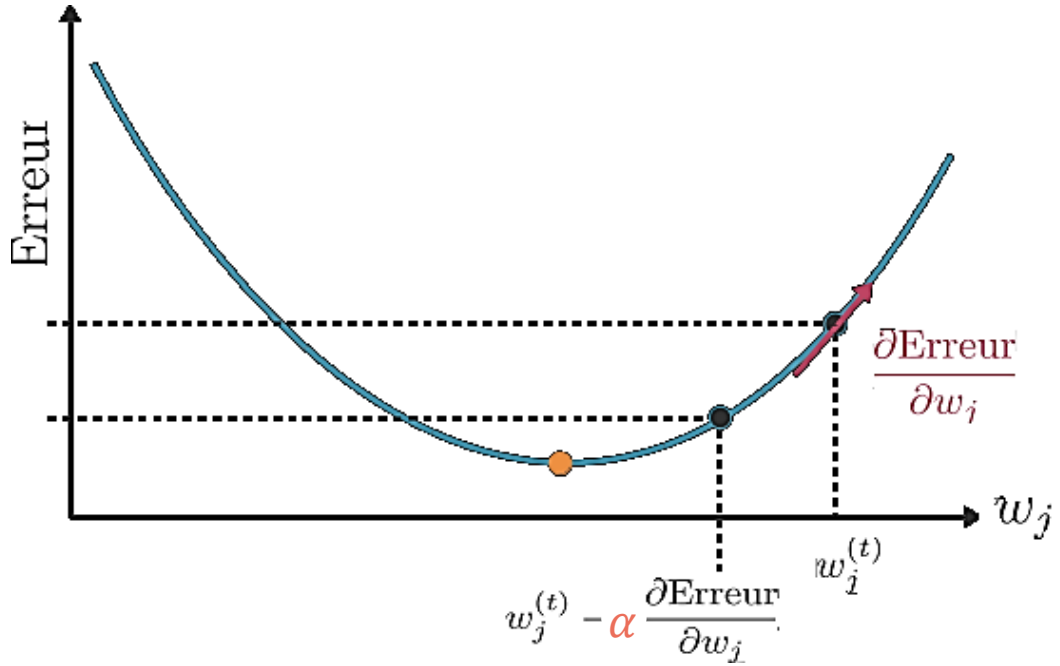
1. le choix de α est empirique,
2. si α est trop petit, le nombre d'itérations peut être très élevé,
3. si α est trop grand, les valeurs de la suite risquent d'osciller autour du minimum sans converger,
4. rien ne garantit que le minimum trouvé est un minimum global.

APPRENTISSAGE PAR DESCENTE DE GRADIENT

L'entraînement d'un perceptron est donc un processus itératif. Après chaque observation, nous allons ajuster les poids de connexion de sorte à réduire l'erreur de prédiction. Pour cela, nous allons utiliser l'algorithme du gradient : le gradient nous donne la direction de la plus grande variation d'une fonction (dans notre cas, la fonction d'erreur), pour trouver le minimum de cette fonction il faut se déplacer dans la direction opposée au gradient. (Lorsque la fonction est minimisée localement, son gradient est égal à 0.)

APPRENTISSAGE PAR DESCENTE DE GRADIENT

- L'erreur d'un perceptron P défini par w=(w_0,..,w_n) sur un échantillon d'apprentissage d'exemples S= (x^t, r^t) est définie en utilisant la fonction d'erreur quadratique : $E = \frac{1}{2} \cdot \sum_S (r^t - y)^2$ où y est la sortie calculée pour un exemple donné
- $\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w_i} = -(r^t - y) \cdot x_i$
- Un déplacement dans le sens opposé au gradient (flèche rose) rapproche w_j de la valeur minimisant l'erreur (point orange).



ALGORITHME D'APPRENTISSAGE

- Données: Base d'exemples (Base_Ex) ou échantillon d'entrainement (X, Y) où le vecteur x = $\{x_1, \cdots, x_n\}$ définit par un ensemble d'attributs caractérisant les exemples , le vecteur r représente la classe de chaque exemple.
- Initialisation : α = le taux d'apprentissage ; W = $\{w_1, \cdots, w_n\}$; $biais = w_0$
- tanque test-arrêt non atteint faire
  - tant que ¬ fin Base_Ex faire
  - prenons un exemple : (x^t, r^t)
  - calculons sa sortie y = $\varphi(w_0 + \sum_1^n w_i\, x_i)$
  - erreur = r^t - y
  - Si erreur ≠ 0 alors
  - Mise à jour des poids: ${w_i}^{t+1} = {w_i}^t + \alpha \cdot \Delta w^t$ où $\cdot\, \Delta w^t =$ erreur. x_i
  - Fin tq

  Fin tq

- Cette procédure est répétée jusqu'à ce que toute la base d'exemples soit correctement classée ou approximativement bien classée.

EXEMPLE

X1	X2	Classe
+1	+1	+
+1,5	-0,5	+
+2	-2	_
-2	-1	_
-2	+1	+
-1	-1,5	_

- Echantillon des exemples du tableau est représenté sur la figure ci-contre.
- Valeurs initiales:

  α = 0.2 , le vecteur poids

  $w = (w_0, w_1, w_2) = (0,\ 1,\ 0.5)$

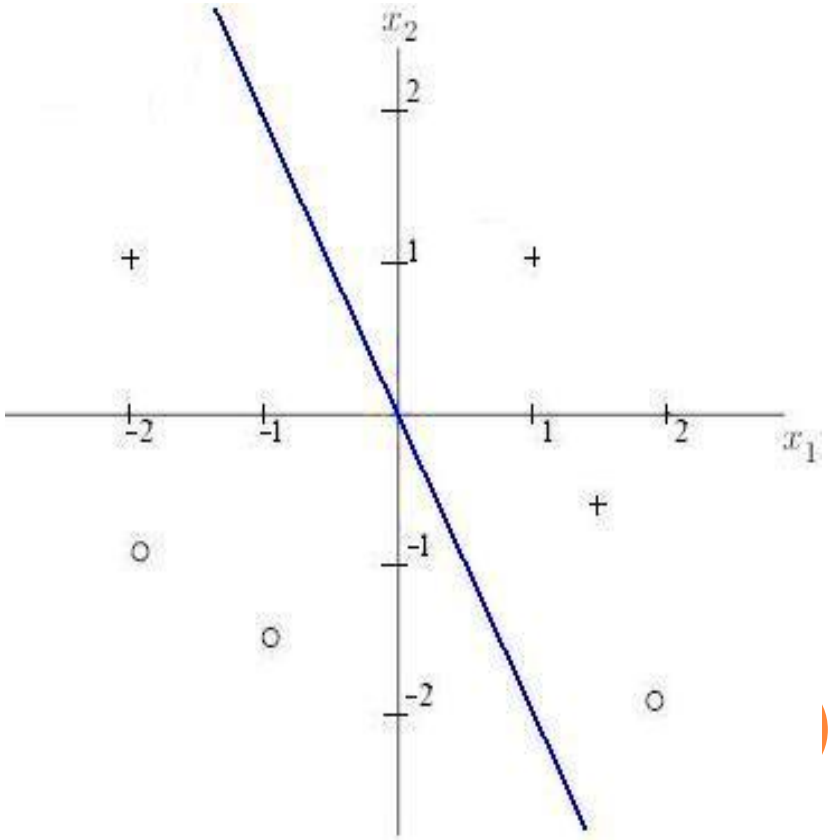
- La sortie du perceptron est :

$$y = \varphi(a) = \varphi(\sum w_i \cdot x_i) = \begin{cases} 1 & si \quad a \geq 0 \\ 0 & sinon \end{cases}$$

- L'équation de la frontière de décision est ainsi donnée par l'équation de la droite:

  $w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0 \leftrightarrow$

  $1x_1 + 0.5x_2 = 0 \leftrightarrow \boldsymbol{x_2 = -2x_1}$



Suite de l'exemple

- Prenons le point P1(+1, +1) et calculons la sortie

  y = $\varphi(w_0 + w_1 x_1 + w_2 x_2) =$

  φ(0+1*1+0,5*1) = φ(1,5) =1

→ classe positive

→ Donc le point P1 est correctement classé

→ Pas de mise à jour

- La sortie associée au point P2(+2,-2) est 1

→ le point P2 est donc mal classé

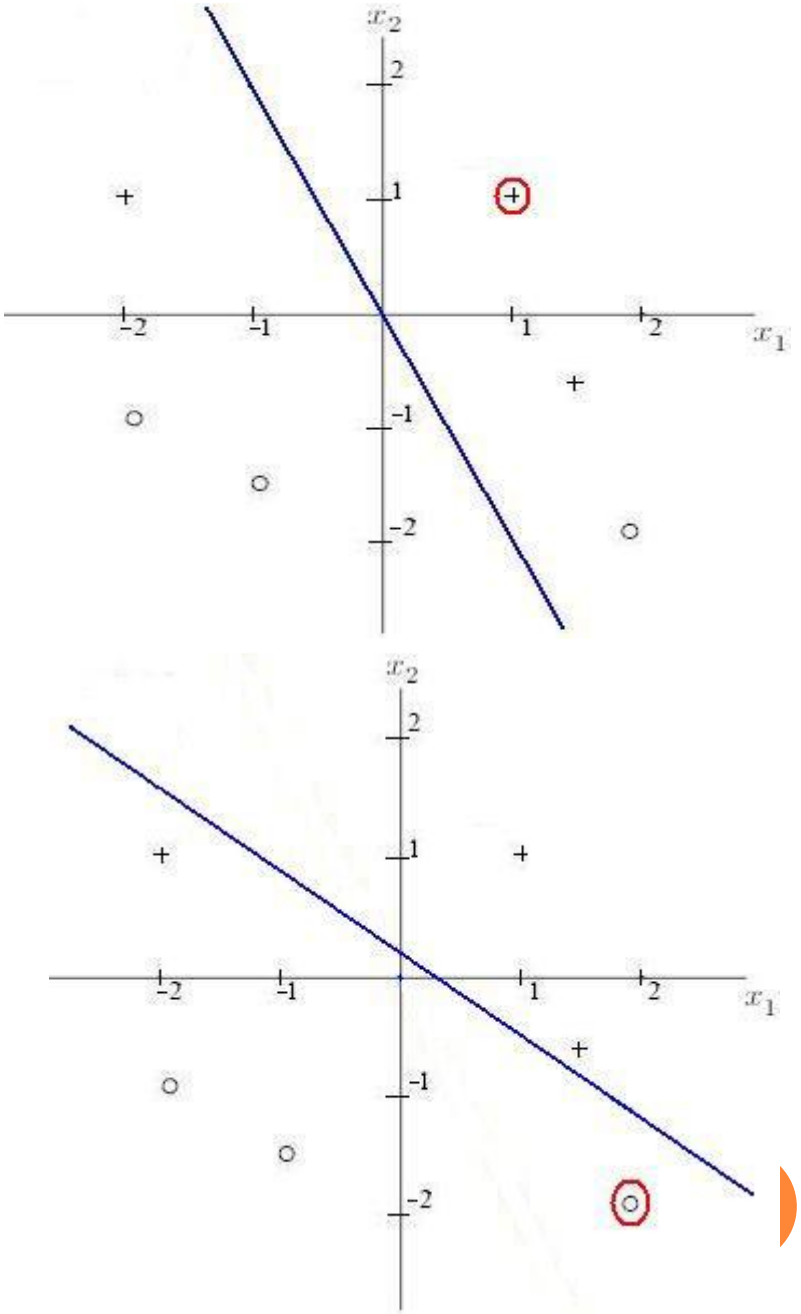
→ Mise à jour des poids

→ $w0 = w0 - 0.2 * 1 = -0,2$

$w1 = w1 - 0.2 * 2 = 0.6$

$w2 = w2 - 0.2 * (-2) = 0.9$

- Modification de la frontière de décision



Suite de l'exemple

- Le point P3(-1, -1.5) est bien classé donc pas de mise à jour
- Le point P4(-2, -1) est bien classé donc pas de mise à jour
- Le point P5(-2, 1) est mal classé donc mise à jour des poids:

→ $w0 = w0 + 0.2*1 = 0$

$w1 = w1 + 0.2*(-2) = 0.2$

$w2 = w2 + 0.2*1 = 1.1$

→ Modification de la droite

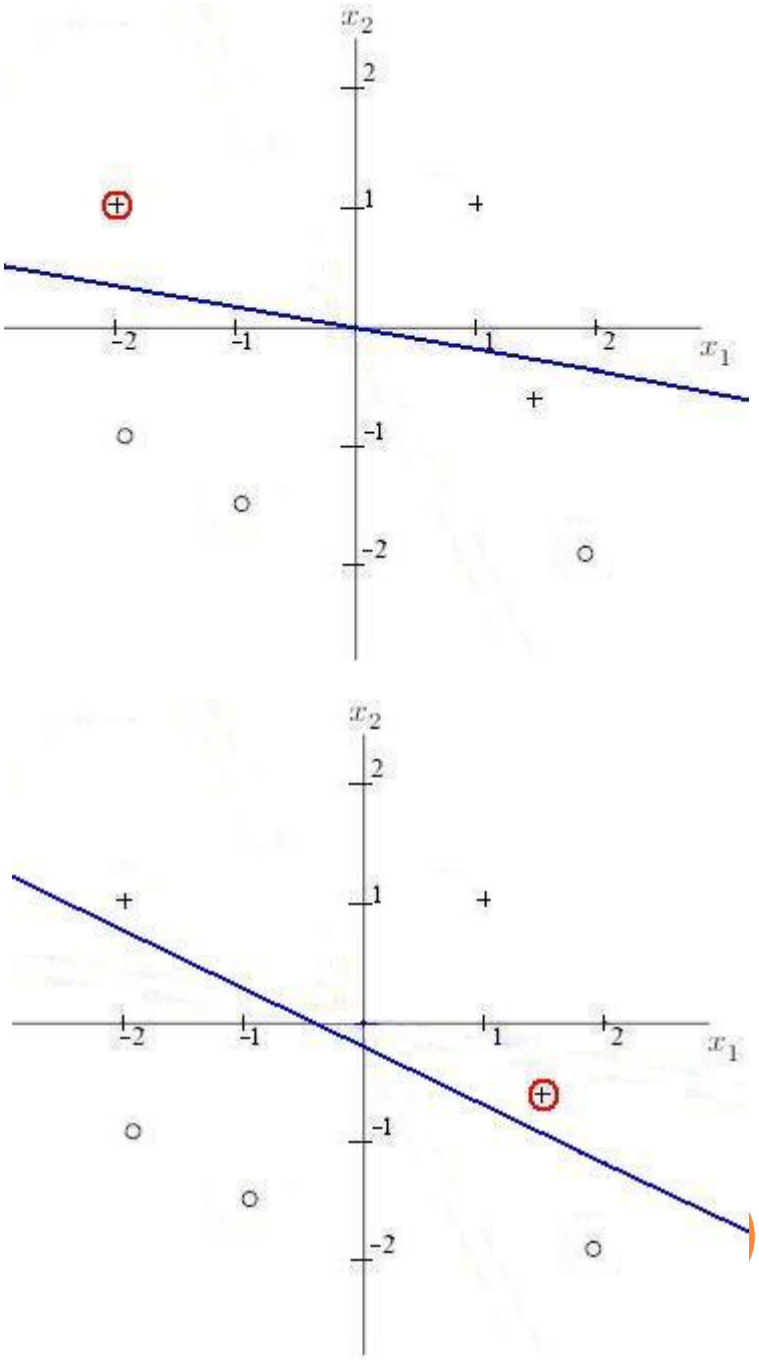
- Le point P6(+1.5, -0.5) est mal classé donc mise à jour des poids:

→ $w0 = w0 + 0.2*1$

$w1 = w1 + 0.2*1.5$

$w2 = w2 + 0.2*(-0.5)$

→ Modification de la droite



Simulation de la fonction ET logique par apprentissage

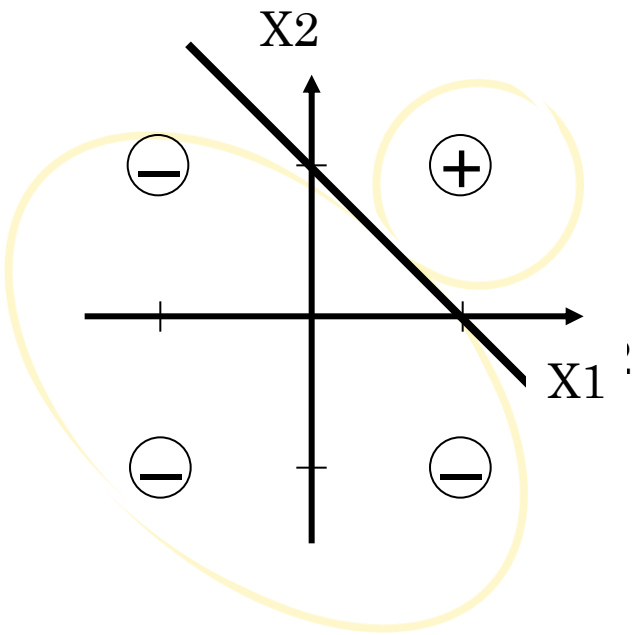
Dans le cas de la fonction ET la séparation des deux classes se fait par une ligne droite:

- W1.X1 + W2.X2 + W0 = 0

Deux classes (deux réponses):

- C1 (S = +1):
  - Pour l'entrée: (+1;+1)
- C2 (S = -1):
  - Pour les entrées:

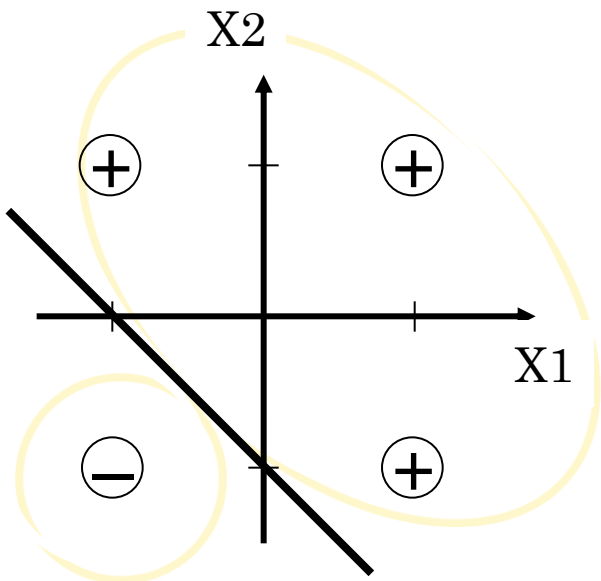
{(-1;-1); (+1;-1); (-1;+1)}



« ET » logique

Simulation de la fonction OU logique par apprentissage

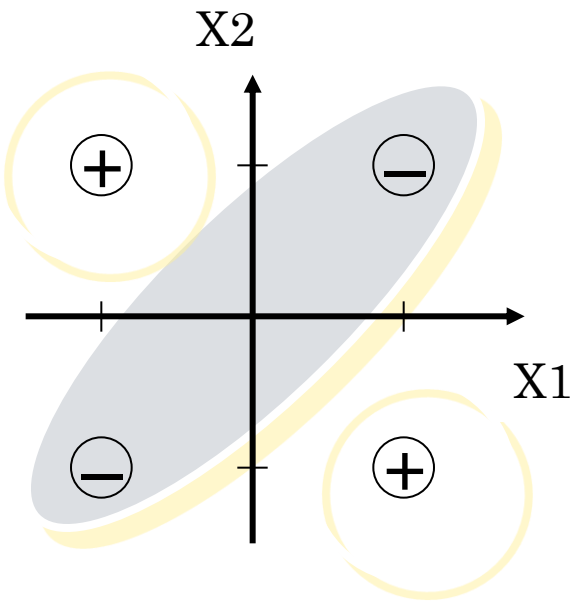
- Dans le cas de la fonction OU, une droite permet toujours la séparation des deux classes.
- Pour deux classes :
  - C1 (S = +1):
    - {(+1;+1); (+1;-1);(-1;+1)}
  - C2 (S = -1):
    - (-1;-1)



« OU » logique

Limites du perceptron : la fonction logique OU exclusif (XOR)

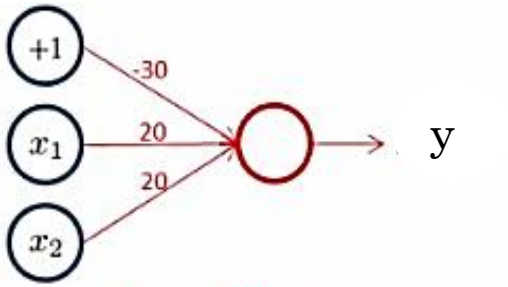
- Dans le cas de la fonction **OU-EXCLUSIF**, la séparation des deux classes ne peut se faire par une droite mais par une courbe.
  - C1 (S = +1):
    - {(-1;+1); (+1;-1)}
  - C2 (S = -1):
    - {(+1;+1); (-1;-1)}



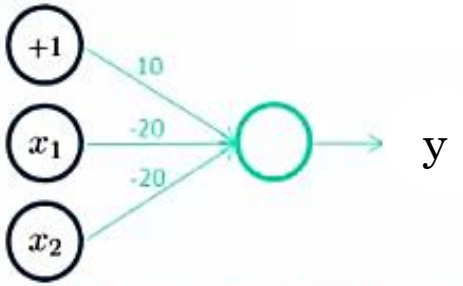
« XOR » logique

Problème du XOR (ou exclusif)

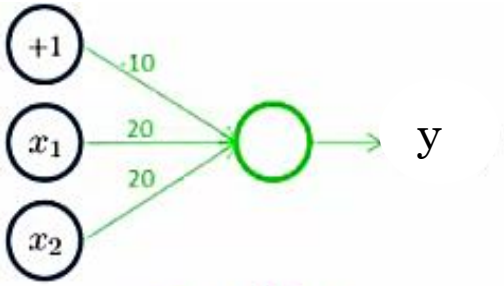
Le perceptron est incapable de distinguer les modèles **non linéairement séparables** [Minsky 69] c'est le cas du XOR



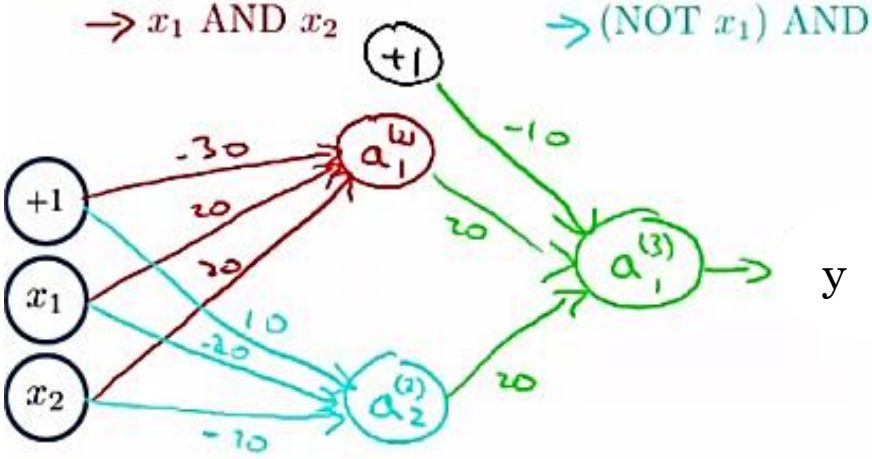
→ x_1 AND x_2



→ (NOT x_1) AND (NOT x_2)



→ x_1 OR x_2



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	y
→ 0	0	0	1	1 ←
0	1	0	0	0
1	0	0	0	0
→ 1	1	1	0	1 ←

On remarque que dans le cas du XOR, il a suffit de rajouter une couche de neurones pour modéliser cette fonction booléenne.

Perceptron multicouches

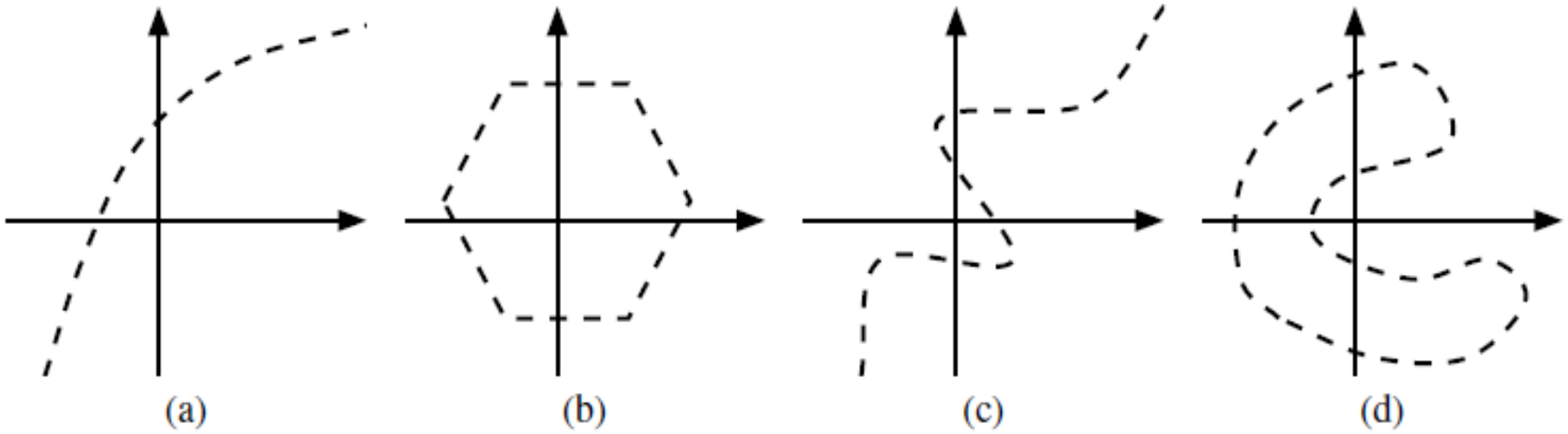
Introduction

- Un perceptron linéaire est bien adapté pour des échantillons linéairement séparables. Cependant, dans la plupart des problèmes réels, cette condition n'est pas réalisée. Un perceptron linéaire est constitué d'un seul neurone. On s'est rendu compte qu'en combinant plusieurs neurones le pouvoir de calcul était augmenté.
- La notion de perceptron multi-couches (PMC) a été ainsi définie. On considère une couche d'entrée qui correspond aux variables d'entrées, une couche de sorties, et un certain nombre de couches intermédiaires. Les liens n'existent qu'entre les unités d'une couche avec les unités de la couche suivante.

Topologie des réseaux

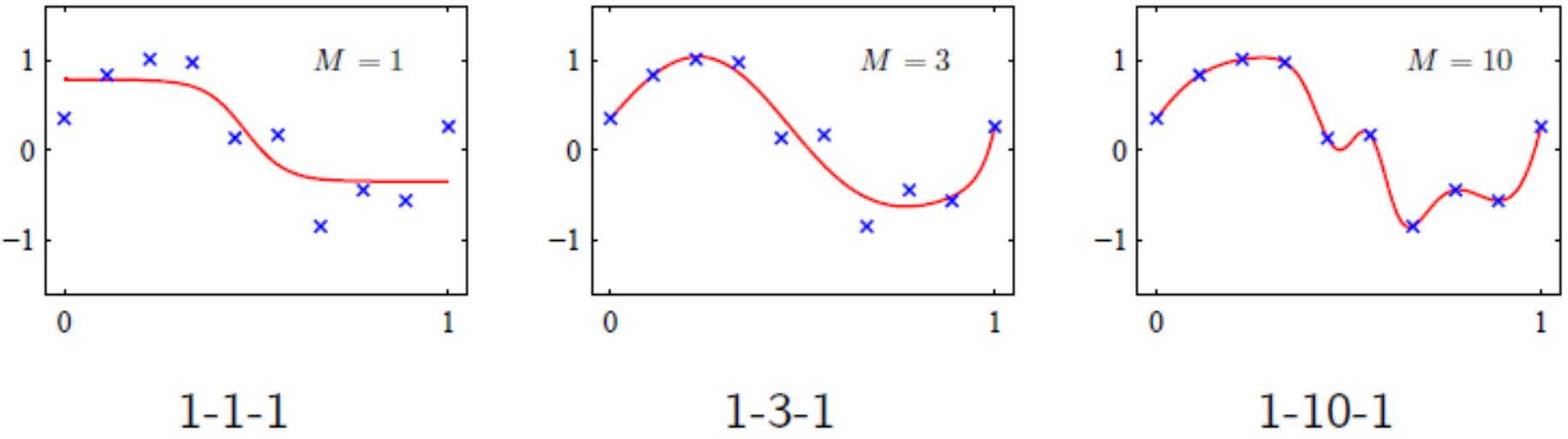
- Selon la topologie de réseau utilisé, différentes frontières de décisions sont possibles
  - Réseau avec une couche cachée et une couche de sortie : frontières convexes
  - Deux couches cachées ou plus : frontières concaves
    - Le réseau de neurones est alors un approximateur universel
- Nombre de poids (donc de neurones) détermine directement la complexité du classifieur
  - Détermination de la bonne topologie est souvent ad hoc, par essais et erreurs

Quelques formes de frontières de décision



Exemples de frontières de décision : (a) convexe ouverte ; (b) convexe fermée ; (c) concave ouverte ; et (d) concave fermée

Nombre de neurones sur la couche cachée

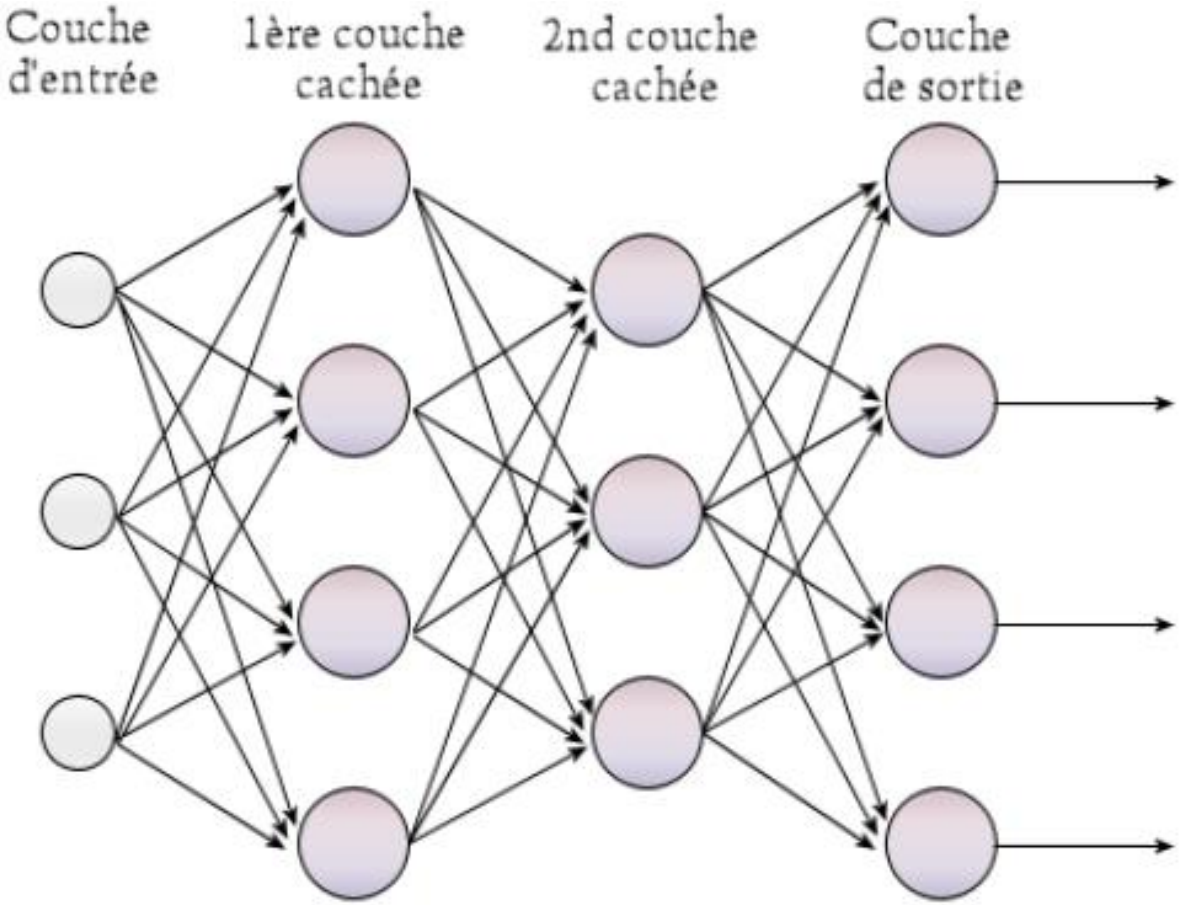


1-1-1 1-3-1 1-10-1

Tiré de *C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.*

1Couche d'entrée - Couche(s) cachée(s) - 1Couche de sortie

Architecture d'un PMC



Exemple d'un PMC à 2 couches cachées

FONCTIONNEMENT D'UN PMC

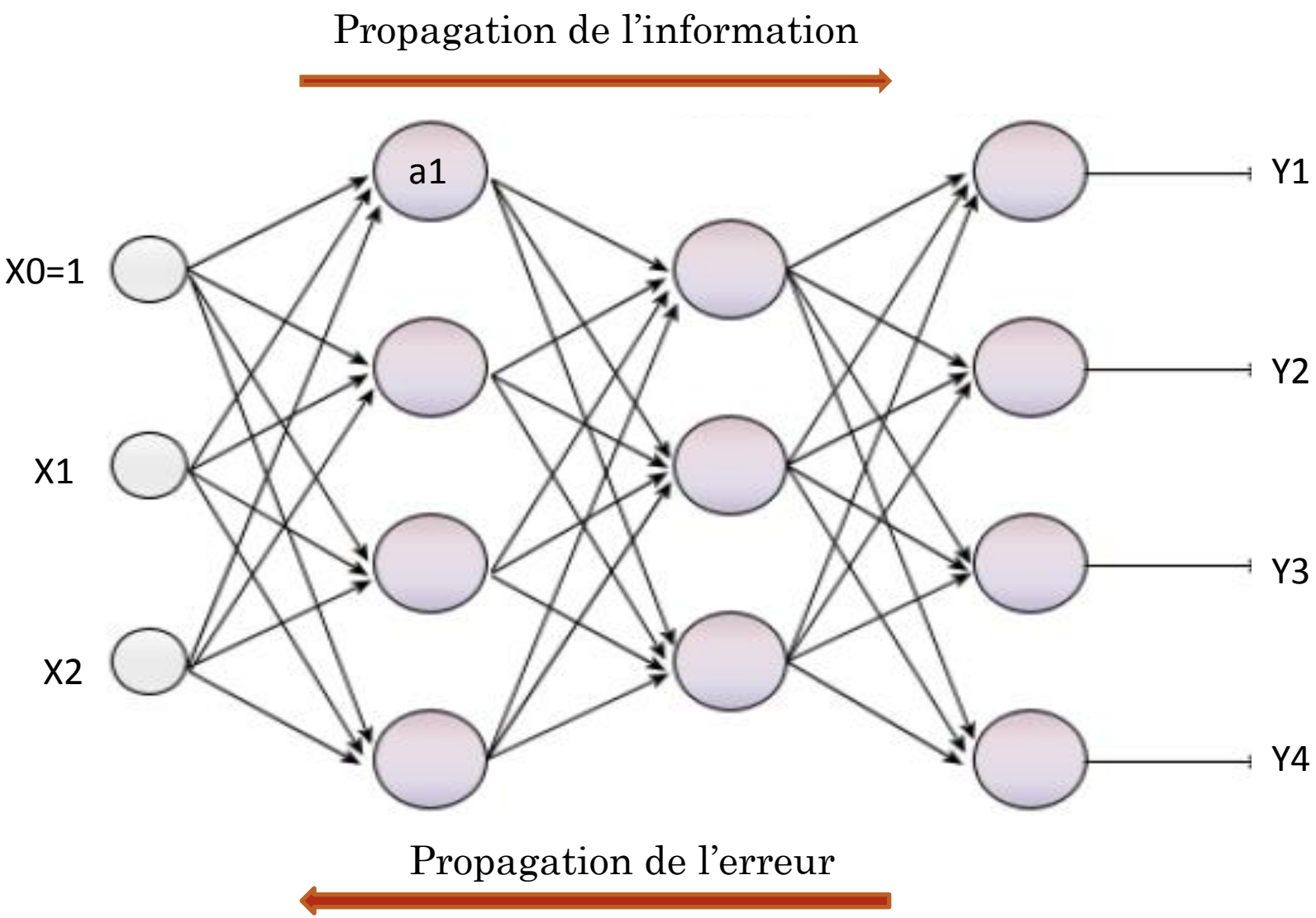
Le fonctionnement d'un PMC est décrit par deux phases:

- Phase1 (Forward Pass) : elle consiste à propager l'information arrivée, de la couche d'entrée vers la couche de sortie, par activation du réseau. La fonction d'activation utilisée au niveau de chaque neurone du réseau est la fonction Sigmoïde,

$$y = \frac{1}{1+ e^{-(w_0+\sum_i w_{ji} \cdot x_i)}}$$

- Phase2 (Backward Pass): elle consiste à propager l'erreur obtenue à travers le réseau, de la couche de sortie vers la couche d'entrée, par descente du gradient dans le sens inverse de la propagation des activations. Cette phase est appelée rétro-propagation (Backpropagation) de l'erreur.

FONCTIONNEMENT D'UN PMC



Rétro-propagation des erreurs (gradient)

Le principe de l'algorithme est, comme dans le cas du perceptron linéaire, de minimiser une fonction d'erreur

- Apprentissage du perceptron multicouche: déterminer les poids de tous les neurones $w_0, w_1 \cdots, w_n$

- Rétro propagation des erreurs : est une méthode pour calculer le gradient de l'erreur pour chaque neurone d'un réseau de neurones de la dernière couche vers la première.
  - Apprentissage par descente du gradient
  - Couche de sortie : correction guidée par l'erreur entre les sorties désirées et obtenues
  - Couches cachées : correction selon l'influence du neurone sur l'erreur dans la couche de sortie

Calcul de l'erreur

- Valeur y_j^t du neurone j pour la donnée $\mathbf{x}^t$

$$y_j^t = \varphi(a_j^t) = \varphi\left(\sum_{i=1}^{R} w_{j,i} y_i^t + w_{j,0}\right)$$

  - φ : fonction d'activation du neurone
  - $a_j^t = \sum_{i=1}^{R} w_{j,i} y_i^t + w_{j,0}$: sommation pondérée des entrées du neurone
  - $w_{j,i}$: poids du lien connectant le neurone j au neurone i de la couche précédente
  - $w_{j,0}$: biais du neurone j
  - y_i^t : sortie du neurone i de la couche précédente pour la donnée $\mathbf{x}^t$
  - R : nombre de neurones sur la couche précédente

Calcul de l'erreur : Forme générale

- Un ensemble de données $\mathcal{X} = \{\mathbf{x}^t, \mathbf{r}^t\}_{t=1}^N$, avec $\mathbf{r}^t = [r_1^t \; r_2^t \; \ldots \; r_K^t]^T$, où $r_j^t = 1$ si $\mathbf{x}^t \in C_j$, autrement $r_j^t = 0$
- Erreur observée pour donnée $\mathbf{x}^t$ sur neurone j de la couche de sortie

$$e_j^t = r_j^t - y_j^t$$

- Erreur quadratique observée pour donnée $\mathbf{x}^t$ sur les K neurones de la couche de sortie (un neurone par classe)

$$E^t = \frac{1}{2} \sum_{j=1}^{K} (e_j^t)^2$$

- Erreur quadratique moyenne observée pour les données du jeu $\mathcal{X}$

$$E = \frac{1}{N} \sum_{t=1}^{N} E^t$$

Calcul de l'erreur

- Correction des poids par descente du gradient de l'erreur quadratique moyenne

$$\Delta w_{j,i} = -\eta \frac{\partial E}{\partial w_{j,i}} = -\frac{\eta}{N} \sum_{t=1}^{N} \frac{\partial E^t}{\partial w_{j,i}}$$

- L'erreur du neurone j dépend des neurones de la couche précédente
  - Développement en utilisant la règle du chaînage des dérivées $(\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x})$

$$\frac{\partial E^t}{\partial w_{j,i}} = \frac{\partial E^t}{\partial e_j^t}\frac{\partial e_j^t}{\partial y_j^t}\frac{\partial y_j^t}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{j,i}}$$

$$\frac{\partial E^t}{\partial w_{j,0}} = \frac{\partial E^t}{\partial e_j^t}\frac{\partial e_j^t}{\partial y_j^t}\frac{\partial y_j^t}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{j,0}}$$

Développement de l'erreur

- Développement avec fonction d'activation sigmoïde ($y_j^t = \frac{1}{1+\exp(-a_j^t)}$)

$$
\begin{aligned}
\frac{\partial E^t}{\partial e_j^t} &= \frac{\partial}{\partial e_j^t}\frac{1}{2}\sum_{l=1}^{K}(e_l^t)^2 = e_j^t \\
\frac{\partial e_j^t}{\partial y_j^t} &= \frac{\partial}{\partial y_j^t} r_j^t - y_j^t = -1 \\
\frac{\partial y_j^t}{\partial a_j^t} &= \frac{\partial}{\partial a_j^t}\frac{1}{1+\exp(-a_j^t)} = \frac{exp(-a_j^t)}{[1+exp(-a_j^t)]^2} \\
&= \frac{1}{1+exp(-a_j^t)}\frac{exp(-a_j^t)+1-1}{1+exp(-a_j^t)} = y_j^t(1-y_j^t) \\
\frac{\partial a_j^t}{\partial w_{j,i}} &= \frac{\partial}{\partial w_{j,i}}\sum_{l=1}^{R} w_{j,l}y_l^t + w_{j,0} = y_i^t \\
\frac{\partial a_j^t}{\partial w_{j,0}} &= \frac{\partial}{\partial w_{j,0}}\sum_{l=1}^{R} w_{j,l}y_l^t + w_{j,0} = 1
\end{aligned}
$$

Mise à jour des valeurs des poids

- Apprentissage des poids de la couche de sortie

$$\begin{aligned}\Delta w_{j,i} &= -\frac{\eta}{N}\sum_{t=1}^{N}\frac{\partial E^t}{\partial w_{j,i}} = -\frac{\eta}{N}\sum_{t=1}^{N}\frac{\partial E^t}{\partial e_j^t}\frac{\partial e_j^t}{\partial y_j^t}\frac{\partial y_j^t}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{j,i}} \\ &= \frac{\eta}{N}\sum_{t=1}^{N} e_j^t y_j^t (1-y_j^t) y_i^t\end{aligned}$$

- Apprentissage des biais de la couche de sortie

$$\begin{aligned}\Delta w_{j,0} &= -\frac{\eta}{N}\sum_{t=1}^{N}\frac{\partial E^t}{\partial w_{j,0}} = -\frac{\eta}{N}\sum_{t=1}^{N}\frac{\partial E^t}{\partial e_j^t}\frac{\partial e_j^t}{\partial y_j^t}\frac{\partial y_j^t}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{j,0}} \\ &= \frac{\eta}{N}\sum_{t=1}^{N} e_j^t y_j^t (1-y_j^t)\end{aligned}$$

Règle delta

- Poser un delta δ_j^t, qui corresponds au *gradient local* du neurone j pour la donnée $\mathbf{x}^t$

$$\begin{aligned} \delta_j^t &= e_j^t y_j^t (1 - y_j^t) \\ \Delta w_{j,i} &= \frac{\eta}{N} \sum_{t=1}^{N} \delta_j^t y_i^t \\ \Delta w_{j,0} &= \frac{\eta}{N} \sum_{t=1}^{N} \delta_j^t \end{aligned}$$

- Formulation utile pour correction de l'erreur sur les couches cachées

Calcul de l'erreur:

au niveau de la couche cachée

- Gradient de l'erreur pour les couches cachées

$$\frac{\partial E^t}{\partial w_{j,i}} = \frac{\partial E^t}{\partial y_j^t} \frac{\partial y_j^t}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{j,i}}$$

- Seul $\frac{\partial E^t}{\partial y_j^t}$ change, $\frac{\partial y_j^t}{\partial a_j^t}$ et $\frac{\partial a_j^t}{\partial w_{j,i}}$ sont les mêmes que sur la couche de sortie
  - Erreur pour un neurone de la couche cachée dépend de l'erreur des neurones k de la couche suivante (rétropropagation des erreurs)

$$E^t = \frac{1}{2} \sum_k (e_k^t)^2$$

$$\frac{\partial E^t}{\partial y_j^t} = \frac{\partial}{\partial y_j^t} \frac{1}{2} \sum_k (e_k^t)^2 = \sum_k e_k^t \frac{\partial e_k^t}{\partial y_j^t}$$

Calcul de l'erreur:
AU niveau de la couche cachée

$$
\begin{aligned}
\frac{\partial E^t}{\partial y_j^t} &= \frac{\partial}{\partial y_j^t}\frac{1}{2}\sum_k (e_k^t)^2 = \sum_k e_k^t \frac{\partial e_k^t}{\partial y_j^t} \\
&= \sum_k e_k^t \frac{\partial e_k^t}{\partial a_k^t}\frac{\partial a_k^t}{\partial y_j^t} \\
&= \sum_k e_k^t \frac{\partial (r_k^t - y_k^t)}{\partial a_k^t}\frac{\partial \left(\sum_l w_{k,l} y_l^t + w_{k,0}\right)}{\partial y_j^t} \\
&= \sum_k e_k^t [-y_k^t(1-y_k^t)] w_{k,j} \\
\delta_k^t &= e_k^t [y_k^t(1-y_k^t)] \\
\frac{\partial E^t}{\partial y_j^t} &= -\sum_k \delta_k^t w_{k,j}
\end{aligned}
$$

Calcul de l'erreur:
au niveau de la couche cachée

- Correction de l'erreur correspondante

$$\frac{\partial E^t}{\partial w_{j,i}} = \frac{\partial E^t}{\partial y_j^t}\frac{\partial y_j^t}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{j,i}}$$

$$= -\left[\sum_k \delta_k^t w_{k,j}\right] y_j^t(1-y_j^t)y_i^t$$

$$\delta_j^t = y_j^t(1-y_j^t)\sum_k \delta_k^t w_{k,j}$$

$$\Delta w_{j,i} = -\eta\frac{\partial E}{\partial w_{j,i}} = -\frac{\eta}{N}\sum_{t=1}^{N}\frac{\partial E^t}{\partial w_{j,i}} = \frac{\eta}{N}\sum_{t=1}^{N}\delta_j^t y_i^t$$

$$\Delta w_{j,0} = -\eta\frac{\partial E}{\partial w_{j,0}} = -\frac{\eta}{N}\sum_{t=1}^{N}\frac{\partial E^t}{\partial w_{j,0}} = \frac{\eta}{N}\sum_{t=1}^{N}\delta_j^t$$

Deux types d'apprentissage

1. Apprentissage par « *batch* »:
   - Guidé par l'erreur quadratique moyenne ($E = \frac{1}{N} \sum_t E^t$)
   - Mise à jour (Correction) des poids une fois à chaque époque (itération), en calculant l'erreur pour tout le jeu (échantillon) de données d'apprentissage
   - Stabilité et Convergence de l'algorithme
2. Apprentissage « *online* »
   - Correction des poids pour chaque présentation de données, donc N corrections de poids par époque
   - Guidé par l'erreur quadratique de chaque donnée (Et)
   - Requiert la permutation de l'ordre de traitement à chaque époque pour éviter les mauvaises séquences
   - Risque d'instabilité et non convergence

ALGORITHME DE RÉTRO-PROPAGATION

Soient les données d'entrainement ${x_i}^t$ et les sorties désirées $\tilde{r}j^t$

Initialiser les poids et biais aléatoirement, $w_{i,j} \in [-0{,}5, 0{,}5]$

Tant que le critère d'arrêt n'est pas atteint, répéter :

1. Calculer les sorties observées en propageant les données vers l'avant
2. Calculer les erreurs observées sur la couche de sortie

$$e_j^t = \tilde{r}_j^t - y_j^t, \quad j = 1, \ldots, K, \quad t = 1, \ldots, N$$

3. Ajuster les poids et biais en rétropropageant l'erreur observée

$$w_{j,i} = w_{j,i} + \Delta w_{j,i} = w_{j,i} + \frac{\eta}{N} \sum_t \delta_j^t y_i^t$$

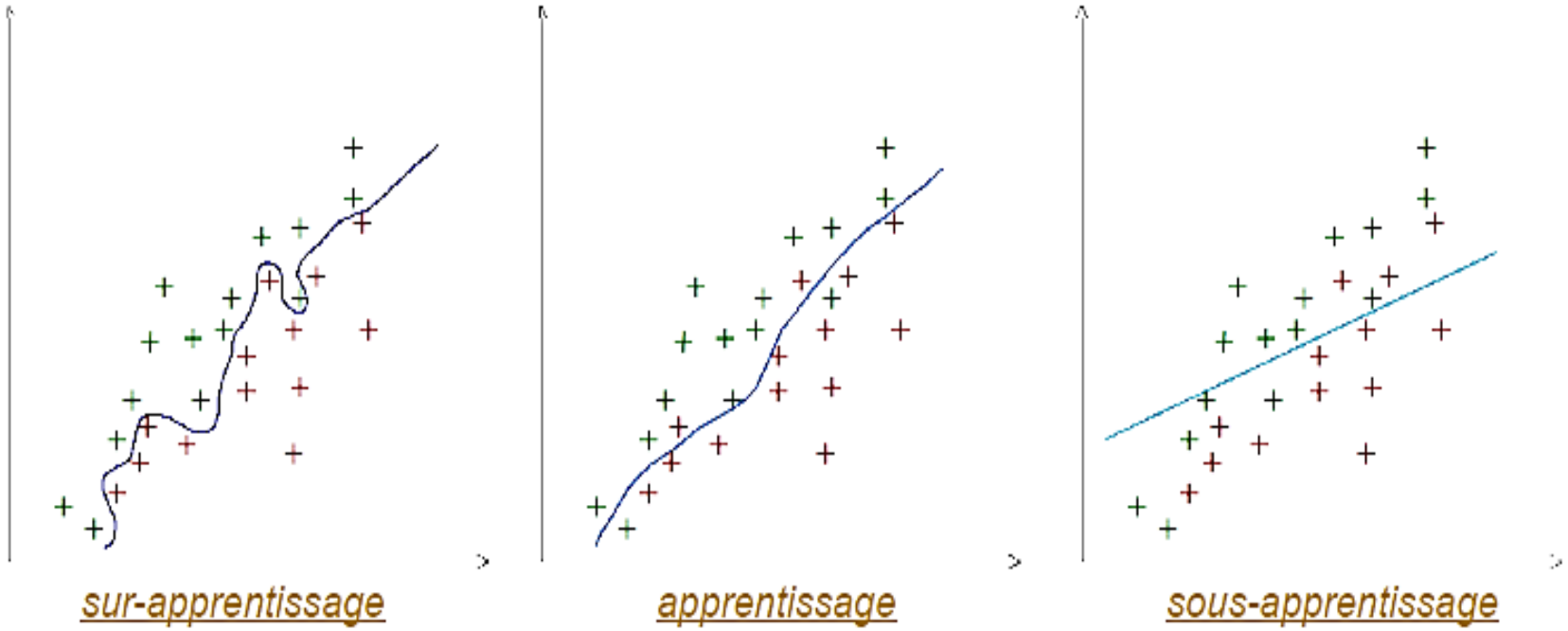
$$w_{j,0} = w_{j,0} + \Delta w_{j,0} = w_{j,0} + \frac{\eta}{N} \sum_t \delta_j^t$$

où le gradient local est défini par :

$$\delta_j^t = \begin{cases} e_j^t y_j^t (1 - y_j^t) & \text{si } j \in \text{couche de sortie} \\ y_j^t (1 - y_j^t) \sum_k \delta_k^t w_{k,j} & \text{si } j \in \text{couche cachée} \end{cases}$$

Sur-apprentissage (overfitting)

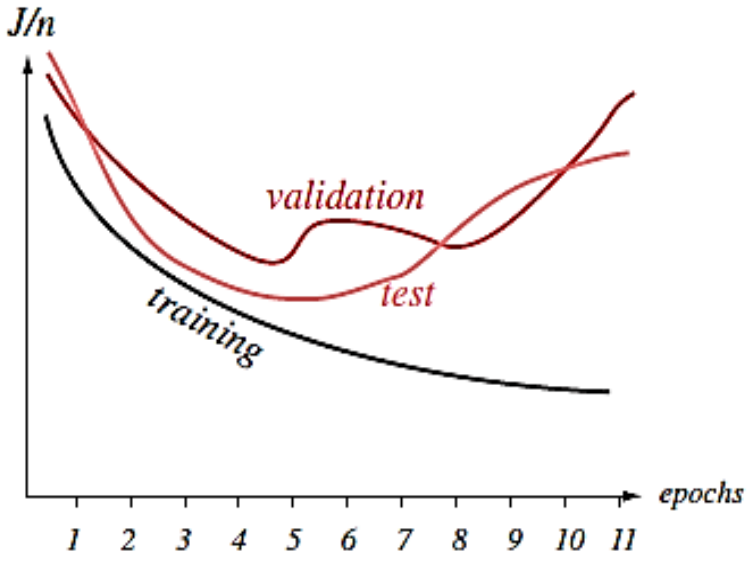
- Le sur-apprentissage (overfitting) a lieu lorsque le modèle est trop conforme aux données d'apprentissage et donc potentiellement incorrect sur de nouvelles données.



- Augmenter la taille des données d'apprentissage limite les risques de sur apprentissage et diminue la complexité du modèle.

Sur-apprentissage (overfitting)

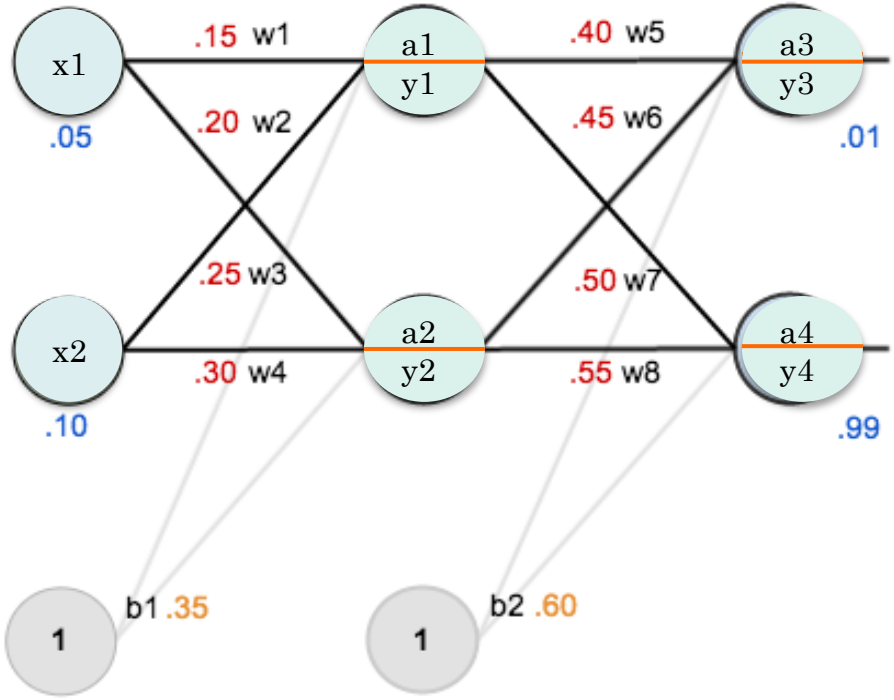
- Nombre d'époques : facteur déterminant pour le surapprentissage
- Critère d'arrêt : lorsque l'erreur sur l'ensemble de validation augmente (généralisation)
- Requiert utilisation d'une partie des données de l'ensemble pour la validation



Tiré de *R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, Wiley Interscience, 2001.*

Exemple

- Soit un réseau de neurones composé de 2 entrées, 1 couche cachée (2 neurones) et 2 sorties, plus les biais associés aux neurones de la couche cachée et la couche de sortie.
- Sur la figure ci-dessous sont indiquées les valeurs des poids initiales, des biais, l'exemple d'entrainement (x1, x2) = (0.05, 0.1) et les sorties désirées (r3, r4) = (0.01, 0.99).



Suite Exemple

Forward Pass

- Calcul de la sortie de chaque neurone de la couche cachée:
  - $y_1 = \varphi(a_1) = \varphi(w_1 x_1 + w_2 x_2 + b_1) =$
    $\varphi(0.15 \times 0.05 + 0.2 \times 0.1 + 0.35) = \varphi(0.3775) = \frac{1}{1+e^{-0.3775}} = 0.5932$
  - $y_2 = \varphi(a_2) = 0.5968$
- Calcul de la sortie de chaque neurone de la couche de sortie:
  - $y_3 = \varphi(a_3) = \varphi(w_5 y_1 + w_6 y_2 + b_2) =$
    $\varphi(0.4 \times 0.5932 + 0.45 \times 0.5968 + 0.6) = \varphi(1.1076) = \frac{1}{1+e^{-1.1076}} =$
    0.7516
  - $y_4 = \varphi(a_4) = 0.7732$
- Calcul de l'erreur totale:
  - $e_1 = r_3 - y_3 = 0.01 - 0.7516$
  - $e_2 = r_4 - y_4 = 0.99 - 0.7732$
  - $E_t = \frac{1}{2}({e_1}^2 + {e_2}^2)$

Suite Exemple

Backward Pass : calcul des erreurs au niveau des couches de sortie et cachée

- **Couche sortie**: nous voulons savoir dans quelle mesure un changement en w_5 affecte l'erreur totale $\frac{\partial E_t}{\partial w_5}$,

$$\frac{\partial E_t}{\partial w_5} = \frac{\partial E_t}{\partial y_3} \cdot \frac{\partial y_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial w_5} = \delta_3 \cdot \frac{\partial a_3}{\partial w_5} ,$$

développons chaque terme de cette expression

$$E_t = \frac{1}{2}({e_1}^2 + {e_2}^2) = \frac{1}{2}((r_3 - y_3)^2 + (r_4 - y_4)^2)$$

$$\frac{\partial E_t}{\partial y_3} = -(r_3 - y_3) = -(0.01 - 0.7516) = 0.7416$$

$$Comme \quad y_3 = \varphi(a_3) = \frac{1}{1+e^{-a_3}} \quad alors \quad \frac{\partial y_3}{\partial a_3} = y_3(1 - y_3) = 0.1867$$

$$Aussi, \quad a_3 = (w_5 y_1 + w_6 y_2 + b_2) \quad alors \quad \frac{\partial a_3}{\partial w_5} = y_1 = 0.5932$$

$$w'_5 = w_5 - \alpha . \frac{\partial E_t}{\partial w_5} = w_5 + \alpha . \Delta w_5$$

$$\frac{\partial E_t}{\partial w_5} = 0.7416 \times 0.1867 \times 0.5932 = 0.0821$$

Mise à jour du poids w_5 : $w'_5 = 0.4 - 0.5 \times 0.0821 = 0.35895$

Suite Exemple

- Même processus de calcul est répété pour w_6, w_7, w_8 *et* b_2.
- **Couche cachée**: continuons le processus de retro-propagation de l'erreur afin de calculer les nouvelles valeurs des poids w_1, w_2, w_3, w_4 *et* b_1. Calcuons l'erreur par rapport à w_1:

$$\rightarrow \frac{\partial E_t}{\partial w_1} = \frac{\partial E_t}{\partial y_1} \cdot \frac{\partial y_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

Or $\frac{\partial E_t}{\partial y_1} = \frac{\partial E_t}{\partial y_3} \cdot \frac{\partial y_3}{\partial y_1} + \frac{\partial E_t}{\partial y_4} \cdot \frac{\partial y_4}{\partial y_1}$

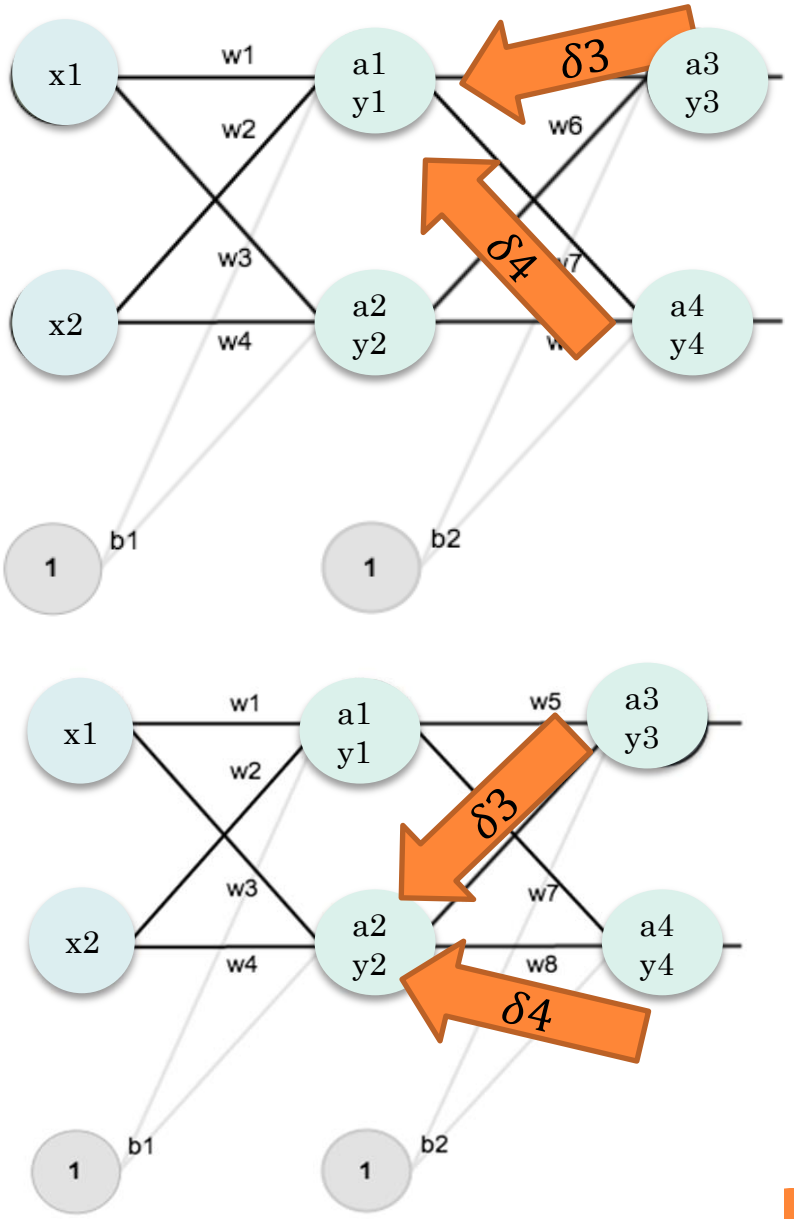
$$\frac{\partial E_t}{\partial w_1} = \sum_k \frac{\partial E_t}{\partial y_k} \cdot \frac{\partial y_k}{\partial y_1} \cdot \frac{\partial y_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial E_t}{\partial w_1} = \sum_k \frac{\partial E_t}{\partial y_k} \cdot \frac{\partial y_k}{\partial a_k} \cdot \frac{\partial a_k}{\partial y_1} \cdot \frac{\partial y_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial E_t}{\partial w_1} = \sum_k \delta_k \cdot \frac{\partial a_k}{\partial y_1} \cdot \frac{\partial y_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial E_t}{\partial w_1} = y_1(1 - y_1) \cdot x_1(\delta_3 \cdot w_5 + \delta_4 \cdot w_7)$$

$$= \delta_1 \cdot x_1$$



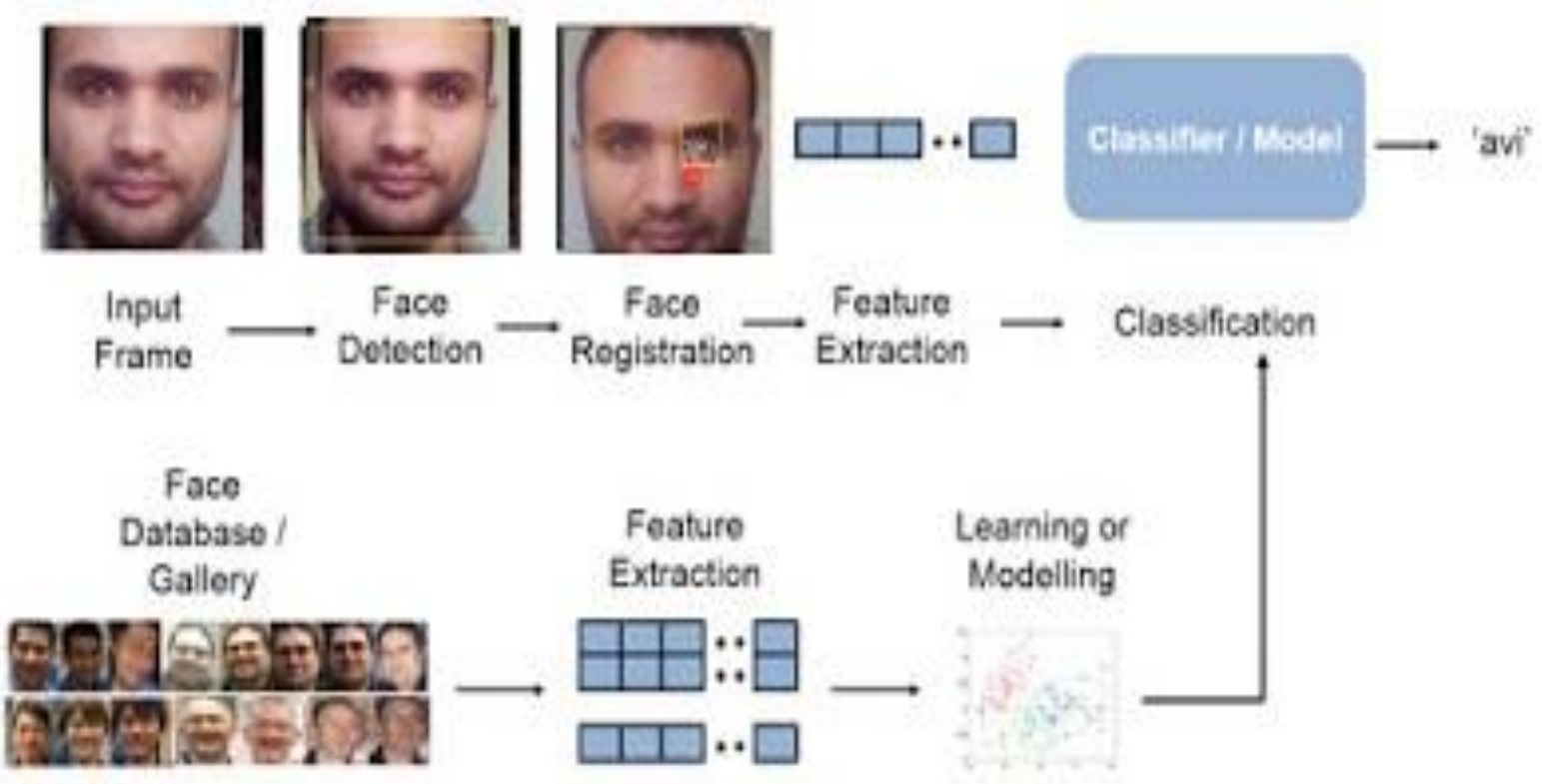
Suite Exemple

- $\delta_1 = 0.5932(1 - 0.5932) \times (0.1384 \times 0.4 - 0.038 \times 0.5) = 0.008774$
- Mise à jour du poids : $w'_1 = 0.15 - 0.5 \times 0.008774 \times 0.05 = 0.14978$
- $w'_2 = w_2 - \alpha . \delta_1 . x_2$
- Ce processus de calcul sera répété pour les valeurs des poids w_3, w_4 et b_1 avec δ_2.

- Tant que le critère d'arrêt (la moyenne des erreurs au carré ; appelée MSE : mean square error ; est inférieure à un seuil préétablie) n'est pas atteint, continuer le processus de calcul jusqu'à convergence.

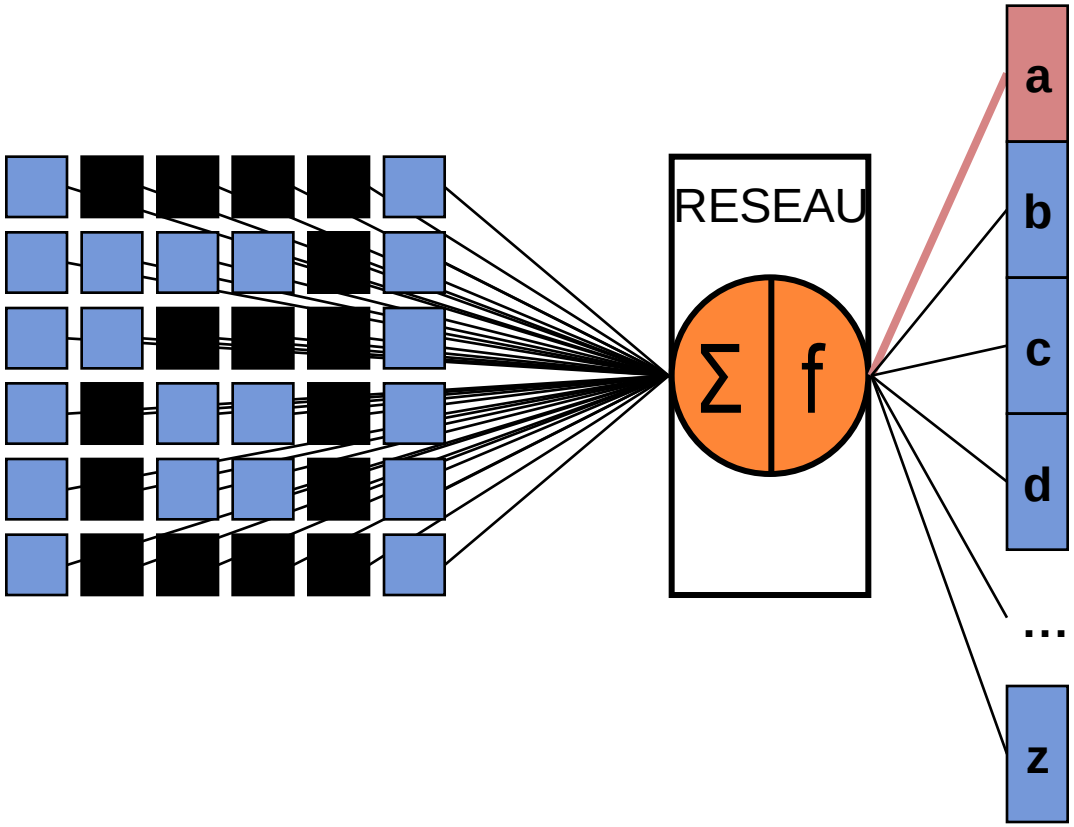
RECONNAISSANCE FACIALE

- La reconnaissance faciale est une phase importante dans de nombreux systèmes biométriques, de sécurité et de surveillance, ainsi que dans des systèmes d'indexation d'image et de vidéo. La détection de l'oeil. Initialement, l'algorithme a besoin de beaucoup d'images positives (images des yeux) et d'images négatives (images sans yeux) pour former le classifieur.



RECONNAISSANCE DE CARACTÈRES

La réponse attendue est le "a". Un seul et unique vecteur de sortie doit être activé.



Métriques de performance d'un modèle d'apprentissage automatique

Il est nécessaire d'**évaluer** n'importe quel algorithme d'apprentissage sur son jeu de données (Dataset). Parmi les métriques d'évaluation d'un modèle, nous distinguons:

- **Matrice de Confusion pour une classification binaire:**

Classe actuelle / Classe prédite	POSITIVE	NEGATIVE
POSITIVE	True Positive (TP)	False Positive (FP)
NEGATIVE	False Negative (FN)	True Negative (TN)

- **Rappel (Recall)** = $\frac{TP}{TP+FN}$; **Précision** = $\frac{TP}{TP+FP}$
- **Spécificité** = $\frac{TN}{TN+FP}$; **Accuracy** = $\frac{TP+TN}{TP+TN+FP+FN}$ (nombre de prédictions qui sont correctes)