Fouille de données - Réseaux de neurones

G. Forestier

Fouille de données

C. Wemmert, S. Lebre

Introduction

Caractéristiques de l'architecture du cerveau humain:

- une architecture massivement parallèle
- un mode de calcul et une mémoire distribués
- une capacité d'apprentissage
- une capacité de généralisation
- une capacité d'adaptation
- une résistance aux pannes
- une faible consommation énergétique

Neurones :

- $\approx 10^{11}$ neurones dans le cerveau humain
- $\approx 10^{4}$ connexions (synapses + axones) / neurones

Introduction

- Dans le cerveau, les neurones sont reliés entre eux par l'intermédiaire d'axones et de dendrites qui permettent de véhiculer des messages depuis un neurone vers un autre.
- Les dendrites représentent les entrées du neurone et son axone sa sortie.
- Un neurone émet un signal en fonction des signaux qui lui proviennent des autres neurones.
- On observe une intégration des signaux reçus au cours du temps $\rightarrow$ sommation des signaux.
- Quand cette somme dépasse un certain seuil, le neurone émet à son tour un signal électrique.

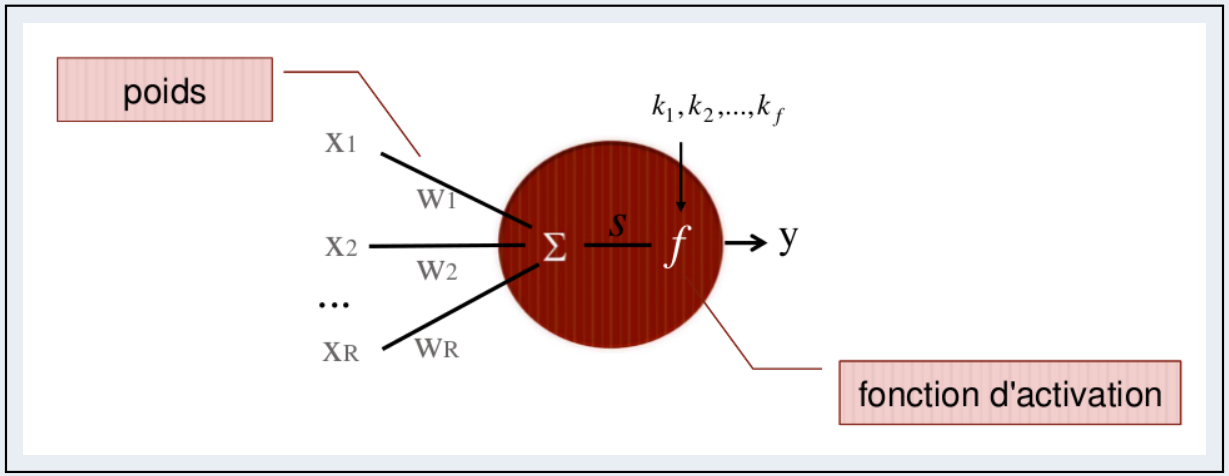
Applications

- Aérospatial : pilotage automatique, simulation du vol...
- Automobile : système de guidage automatique,...
- Défense : guidage de missile, suivi de cible, reconnaissance du visage, radar, sonar, lidar, traitement du signal, compression de données, suppression du bruit...
- Electronique : prédiction de la séquence d'un code, vision machine, synthétiseur vocal, modèle non linéaire,...
- Finance : Prévision du coût de la vie
- Secteur médical : Analyse EEC et ECG
- Télécommunications : Compression de données ...

Historique

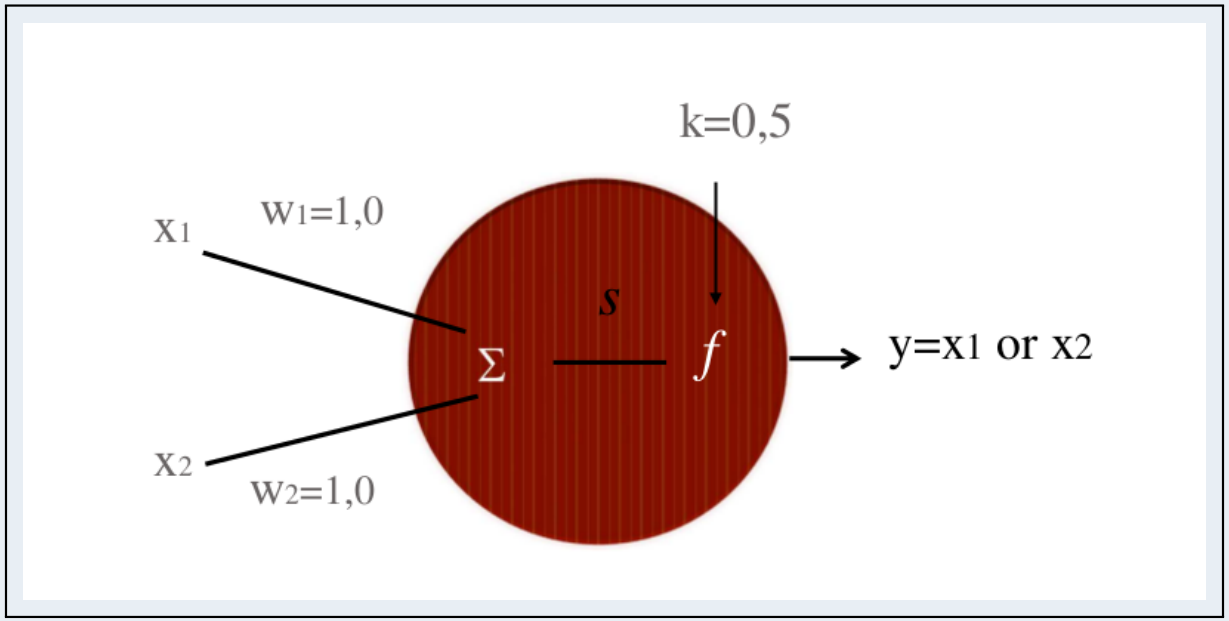
- 1943
  - Modèle de McCulloch et Pitts : premier modèle de neurone formel
- 1949
  - Règle de Hebb : apprentissage par renforcement du couplage
  - synaptique
- 1960
  - Rosenblatt : perceptron et théorème de convergence
  - Minsky et Papert : limites du perceptron mono-couche
- 1980
  - Modèle de Hopefield : réseaux bouclés
  - Perceptron Multi-Couche en 1985
  - Werbos : rétropropagation dans des perceptrons multi-couches
  - Kohonen
  - Gaz neuronaux croissants
- 1998
  - Convolutional neural network (CNN) - Deep Learning

Neurone formel



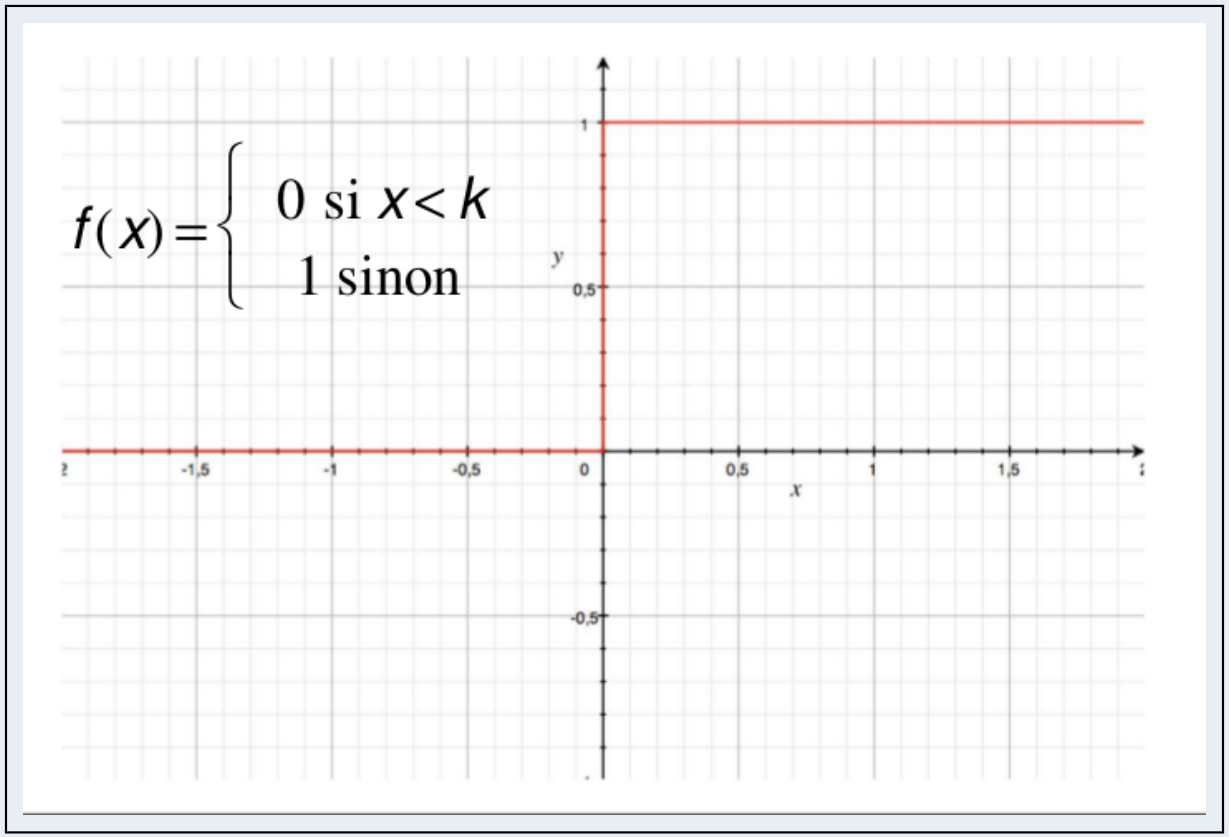
- Interprétation : $y = f(\sum w_i.x_j, (k_1, k_2, \ldots, k_j))$

Exemple : neurone représentant OR



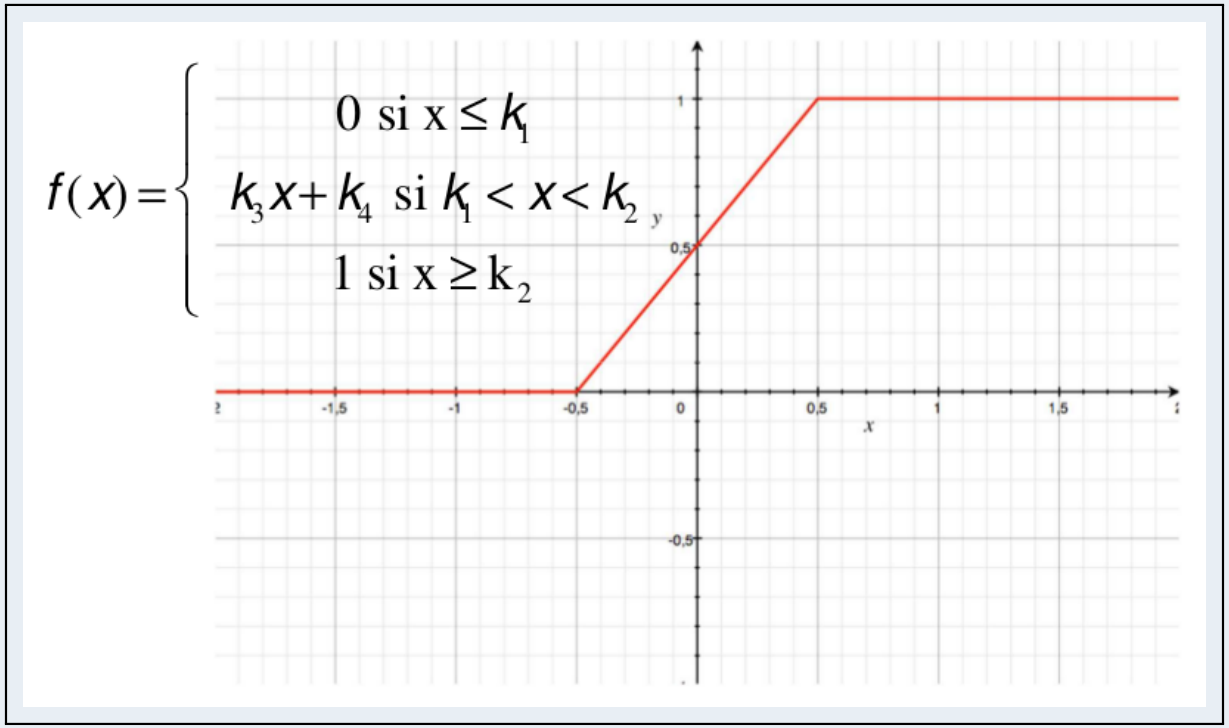
Fonction d'activation

- Fonction à seuil



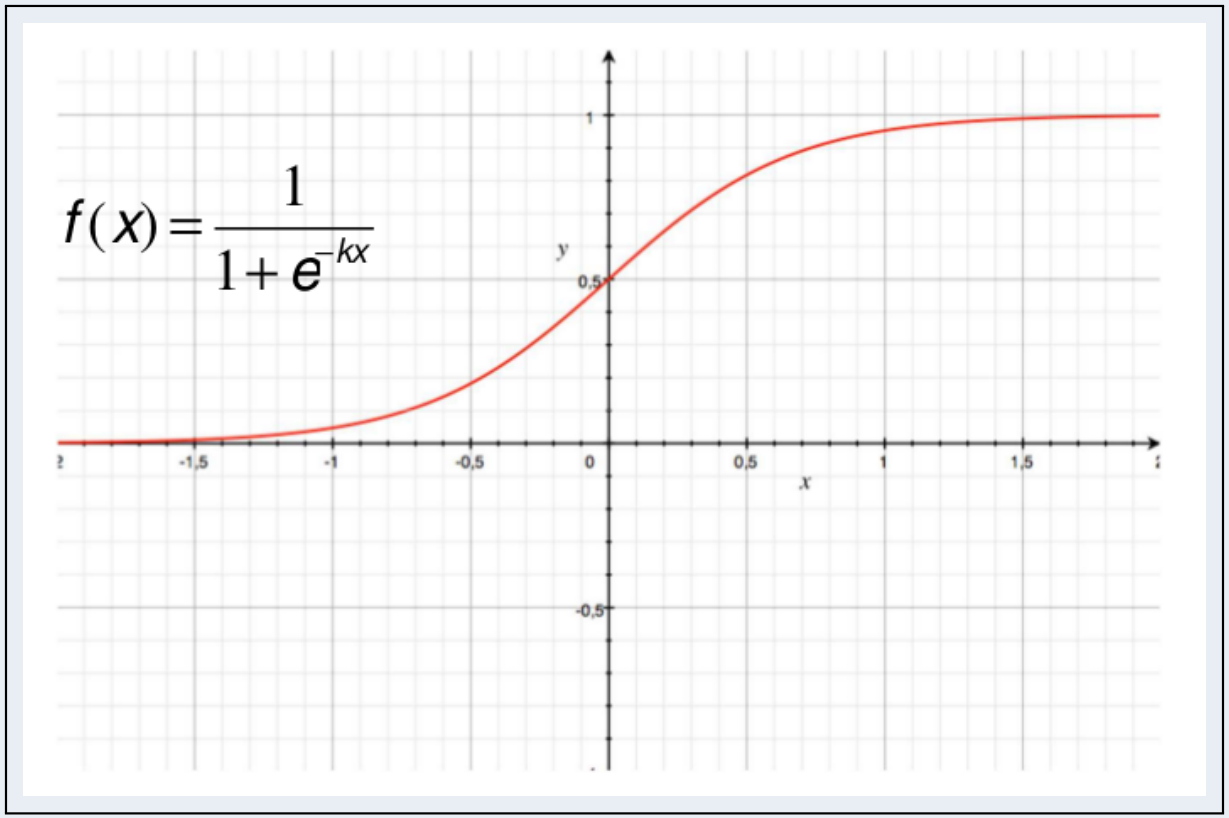
Fonction d'activation

- Fonction linéaire seuillée



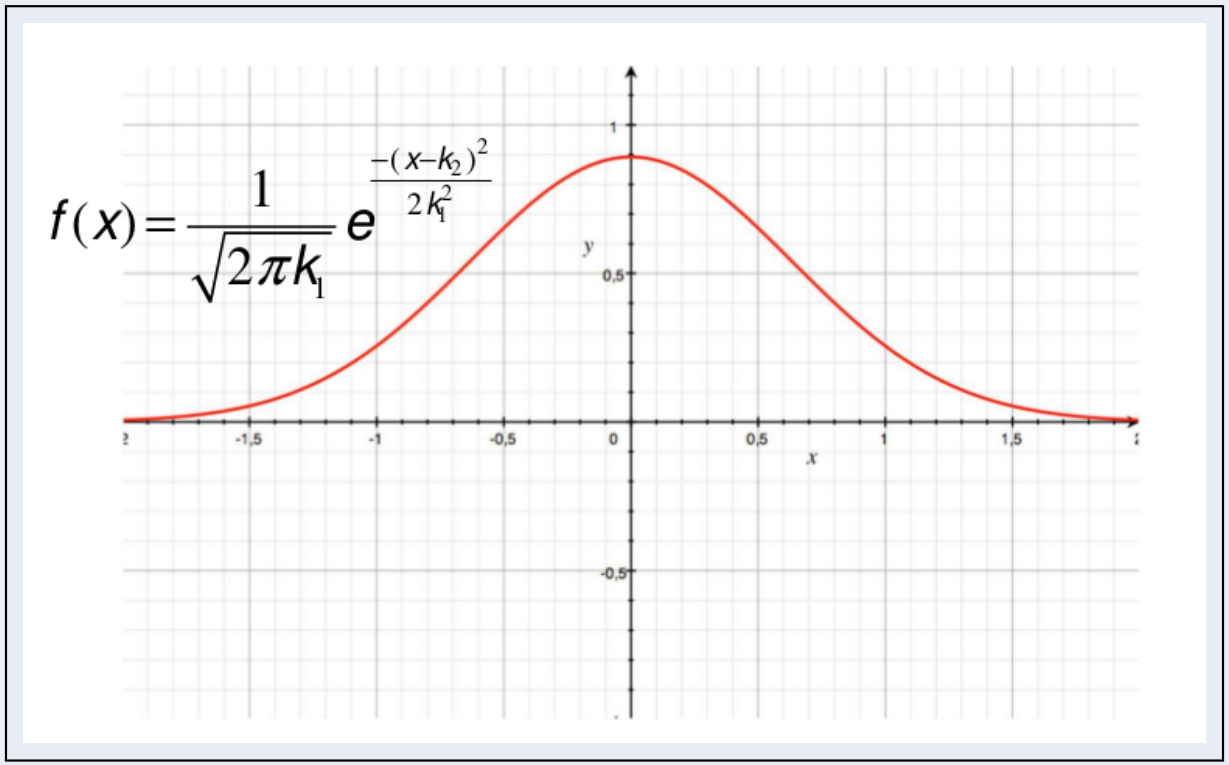
Fonction d'activation

- Fonction sigmoïde



Fonction d'activation

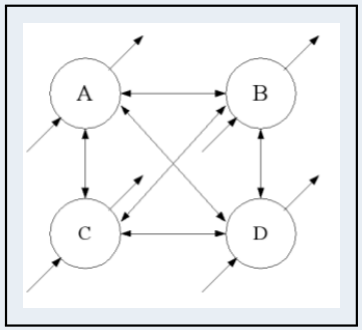
- Fonction gaussienne



Architecture d'un réseau de neurones

Deux grandes familles :

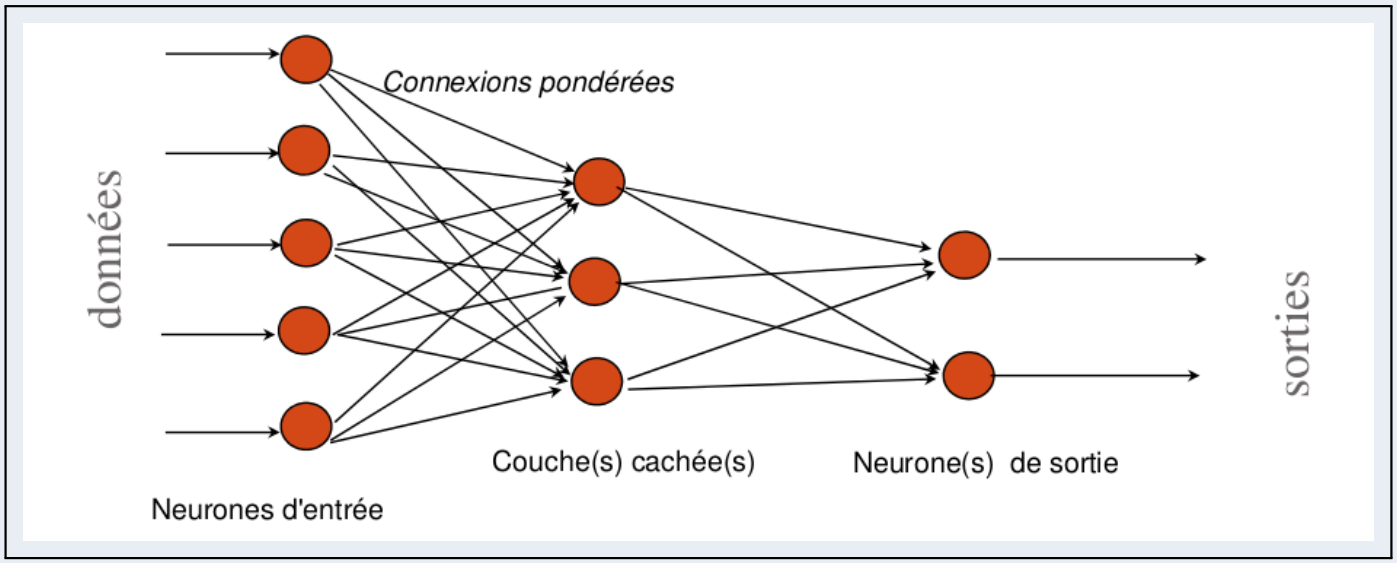
1. Réseaux bouclés :
   - Exemple : réseau de Hopfield
   - Problème de stabilité et d'apprentissage



2. Réseaux non bouclés (réseaux à couches) :
   - Sans couche cachée
   - Avec couches cachées

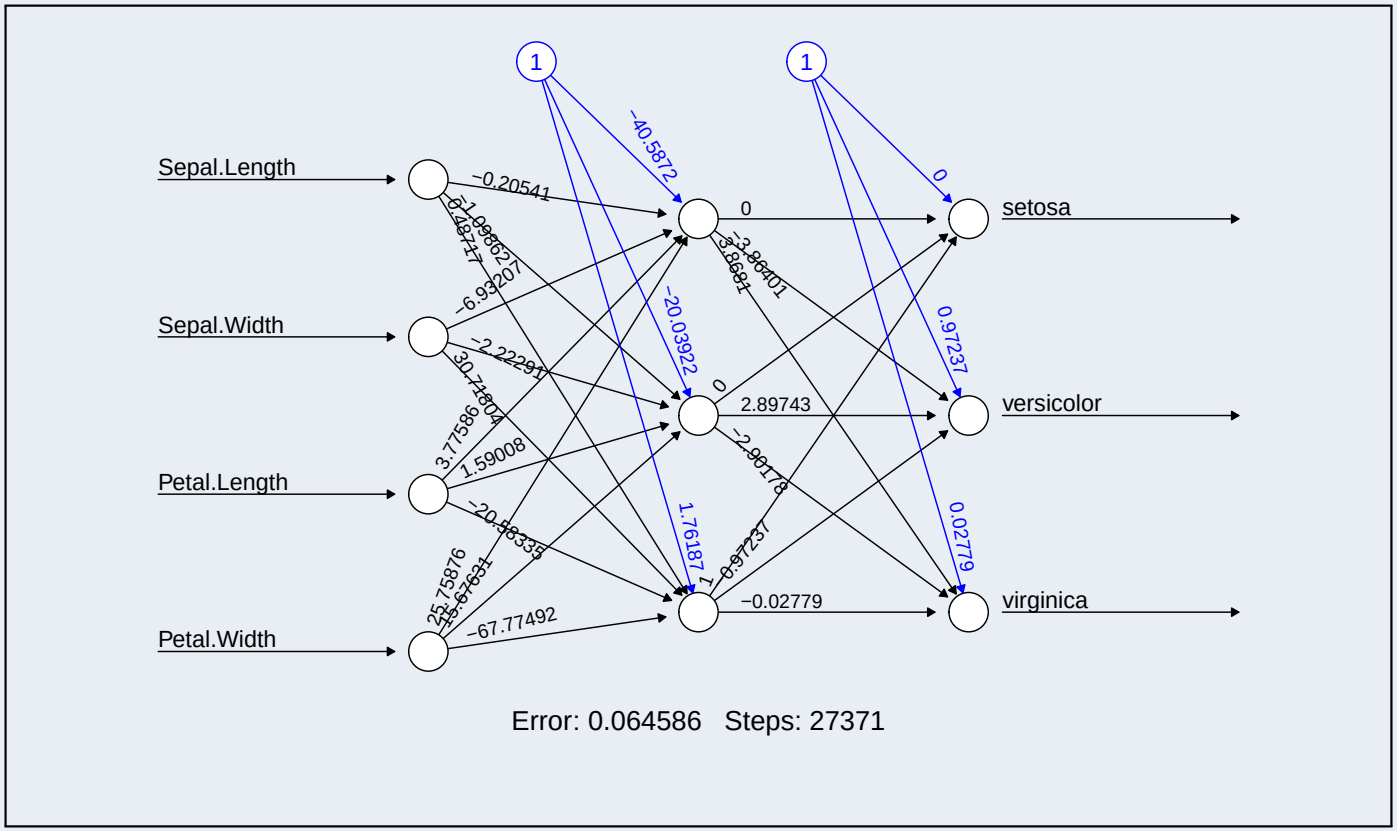
Réseaux de neurones

- Exemple : perceptron multi-couches



Réseaux de neurones

- Exemple : perceptron multi-couches



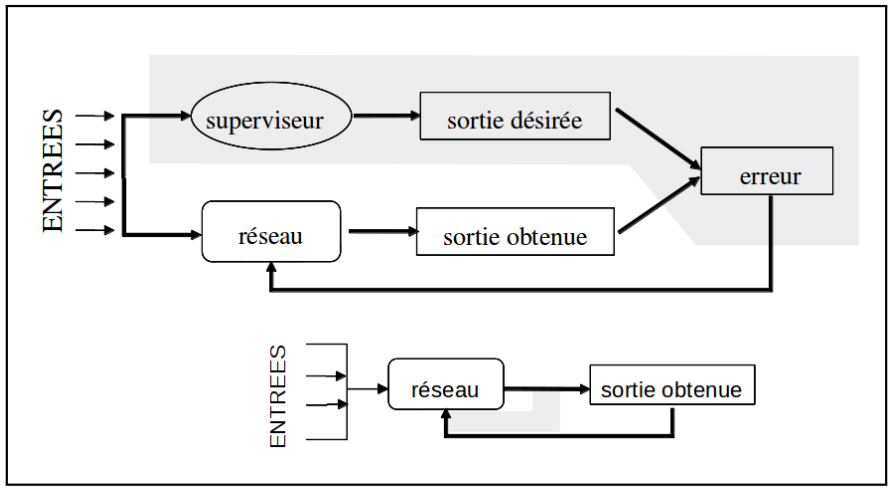
Apprentissage

Apprentissage :

- phase du développement durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré.

Deux grandes classes d'algorithmes d'apprentissage :

- apprentissage supervisé
- apprentissage non supervisé



Apprentissage

Apprentissage "classique"

- Pour chaque donnée
  1. On présente la donnée en entrée
  2. On propage les activations jusqu'aux cellules de sortie
  3. On examine toutes les cellules de sortie
  4. On compare le résultat à celui espéré
  5. On mémorise l'erreur

si l'erreur n'est pas nulle :

- On rétro-propage cette différence (erreur)
- On corrige des poids pour minimiser cette erreur :

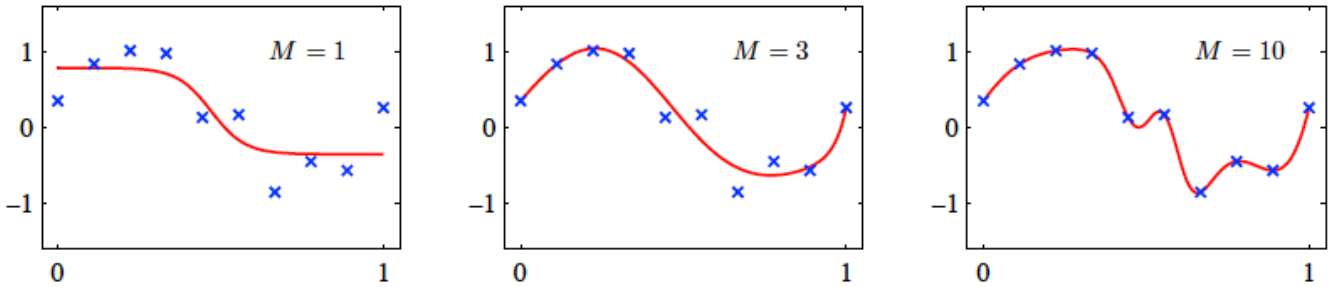
$$w(t+1) = w(t) + \Delta w(t)$$

où $\Delta w(t)$ dépend de l'erreur

Choix de l'architecture

Pas de méthode automatique pour choisir l'architecture du réseau

- On peut procéder à des essais avec un modèle simple (une couche cachée) et un modèle plus complexe (2 ou 3 couches cachées)
- Méthode de construction dynamique de réseaux



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

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

Choix des paramètres

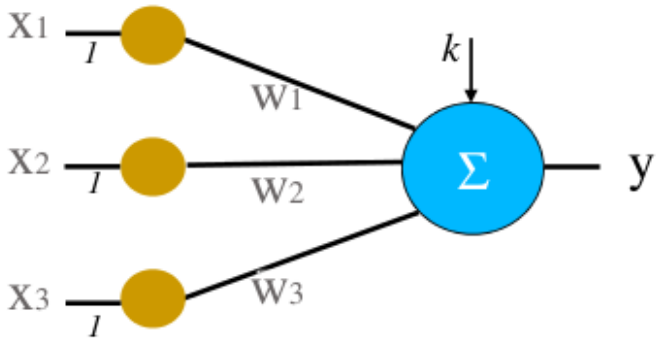
En pratique :

- On choisit une fonction de calcul et une fonction d'activation
- On choisit une architecture :
  1. Nombre d'entrées
  2. Nombre de sorties
  3. Nombre de couches internes
  4. Nombre de neurones de chacune des couches internes
- On choisit une fonction d'erreur
- On définit un critère d'arrêt

Perceptron

Réseau à propagation avant :

- composé de neurones linéaires à seuil avec une couche d'entrée et une de sortie entièrement interconnectées
- les neurones de la couche d'entrée transmettent simplement l'entrée



Perceptron

Réseau à propagation avant :

- composé de neurones linéaires à seuil avec une couche d'entrée et une de sortie entièrement interconnectées

Remarque :

$$y = \begin{cases} 1 & \text{si } \sum_{j=1}^{n} w_j x_j \geq k \\ 0 & \text{sinon} \end{cases}$$

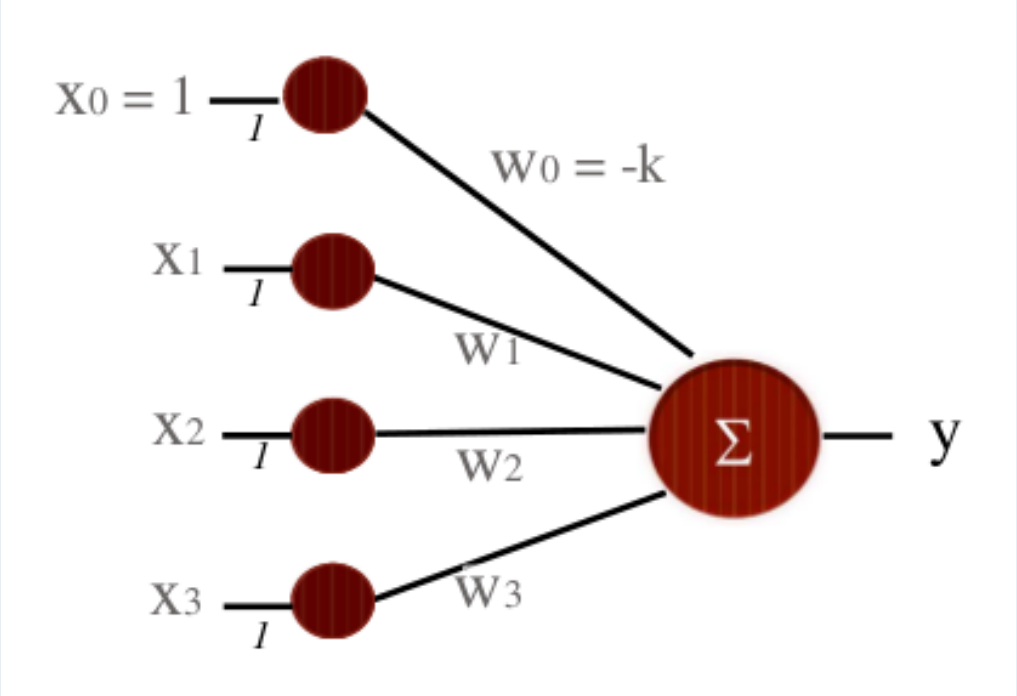
Peut s'écrire :

$$y = \begin{cases} 1 & \text{si } (\sum_{j=1}^{n} w_j x_j - k) \geq 0 \\ 0 & \text{sinon} \end{cases}$$

Si on crée une entrée virtuelle x_0 égale à 1 et que le poids affecté à cette entrée est l'opposé du seuil $w_0 = -k$

$$y = \begin{cases} 1 & \text{si } (\sum_{j=1}^{n} w_j x_j + w_0) \geq 0 \\ 0 & \text{sinon} \end{cases}$$

Perceptron



remarque : noter que x_0 vaut 1

Perceptron

Ainsi :

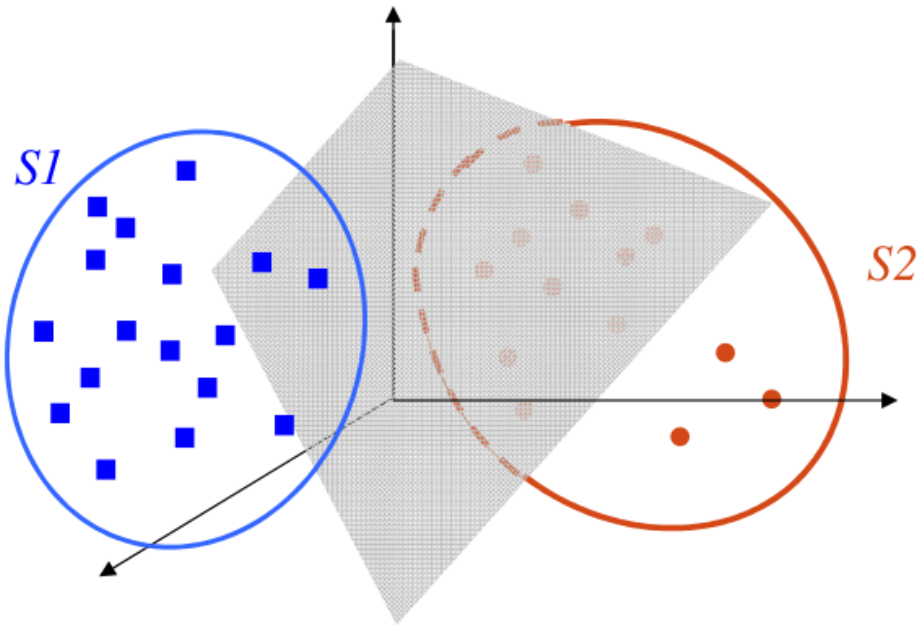
$$y = \begin{cases} 1 & \text{si } \left(\sum_{j=1}^{n} w_j x_j\right) \geq 0 \\ 0 & \text{sinon} \end{cases}$$

Ce qui est l'équation d'un **hyperplan**

Perceptron

Exemple : apprentissage à 2 classes S1 et S2

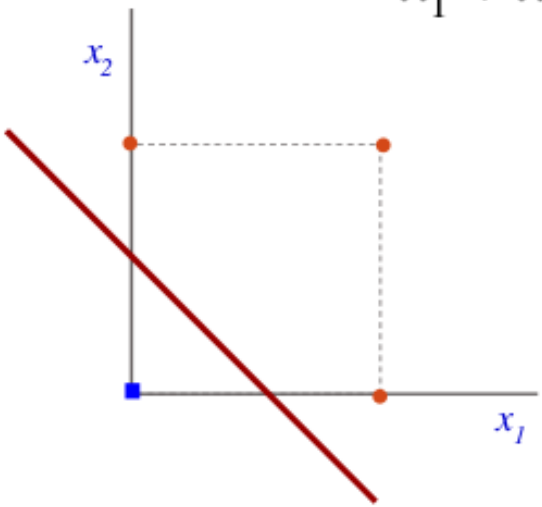
- Ces deux ensembles sont linéairement séparables s'il existe un hyperplan dans l'espace des données tel que S1 et S2 soient situés de part et d'autre de cet hyperplan

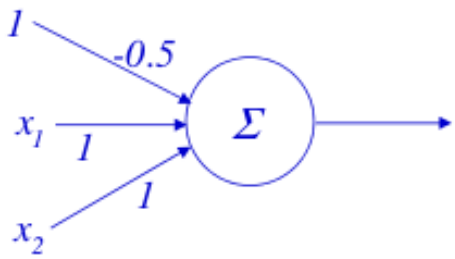


Perceptron

- Exemple en 2D : fonction booléenne OU

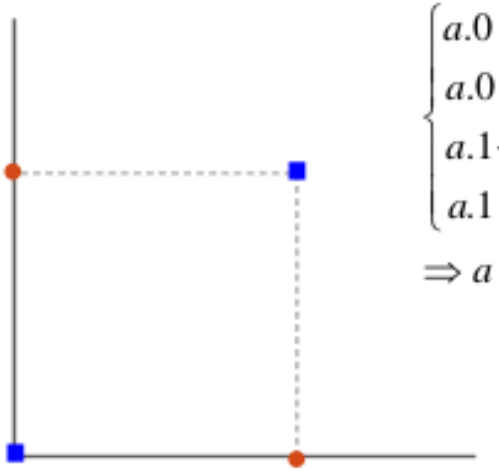
$$x_1 + x_2 - 0.5 = 0$$





Perceptron

- Fonction booléenne XOR

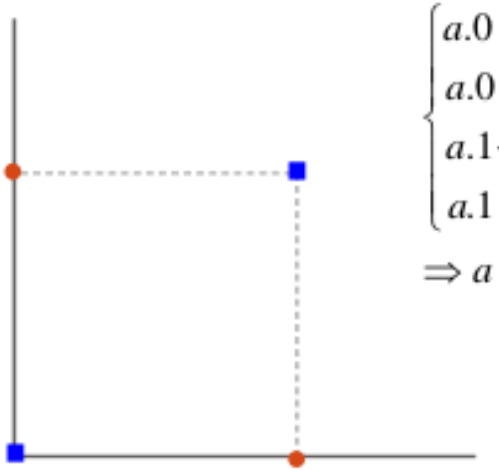


$$\begin{cases} a.0+b.0+c \leq 0 \\ a.0+b.1+c > 0 \\ a.1+b.0+c > 0 \\ a.1+b.1+c \leq 0 \end{cases}$$

$$\Rightarrow a+b+2c \leq 0 \text{ et } a+b+2c > 0$$

Perceptron

- Fonction booléenne XOR



$$\begin{cases} a.0+b.0+c \leq 0 \\ a.0+b.1+c > 0 \\ a.1+b.0+c > 0 \\ a.1+b.1+c \leq 0 \end{cases}$$

$$\Rightarrow a+b+2c \leq 0 \text{ et } a+b+2c > 0$$

=> contradiction

Perceptron

Apprentissage

- Minimisation de la fonction d'erreur
- Apprentissage dans l'esprit de la règle de Hebb : ajouter à chaque connexion quelque chose de proportionnel à l'entrée et à la sortie.
- Apprentissage seulement si erreur de classification

Algorithme :

1. On initialise le perceptron avec des valeurs aléatoires
2. Les exemples sont présentés à la suite : on ajuste les poids suivant que le perceptron classe correctement ou non l'exemple

Perceptron et descente du gradient

Apprentissage de Widrow-Hoff (ou règle Adaline, ou delta règle)

- Les poids sont modifiés directement après chaque exemple
- La convergence est garantie si ϵ est "bien" choisi

L'algorithme :

1. On initialise les poids aléatoirement
2. Répéter (passer tous les exemples)
   - Pour chaque exemple, calculer : $w_j = w_j + \Delta w_j$
   - Ajuster les poids par : $\Delta w_j = \epsilon(c_i - o_i)(x_i)_j$
3. Jusqu'à stabilisation.

Perceptron : exemple

Exemple avec une règle d'apprentissage simplifiée ($\epsilon = 1$)

- $\Delta w_j = w_j + (c_i - o_i)(x_i)_j$

Initialisation des poids à 0

	X1	X2	X3	X4	c
E1	0	0	0	1	**0**
E2	0	1	1	1	**0**
E3	1	1	0	1	**1**
E4	0	0	1	0	**0**
E5	1	0	1	1	**1**

Perceptron : exemple

	X0	X1	X2	X3	X4	w0	w1	w2	w3	w4	c	o	c-o	w0	w1	w2	w3	w4
E1	1	0	0	0	1	0	0	0	0	0	0	1	-1	-1	0	0	0	-1
E2	1	0	1	1	1	-1	0	0	0	-1	0	0	0	-1	0	0	0	-1
E3	1	1	1	0	1	-1	0	0	0	-1	1	0	1	0	1	1	0	0
E4	1	0	0	1	0	0	1	1	0	0	0	1	-1	-1	1	1	-1	0
E5	1	1	0	1	1	-1	1	1	-1	0	1	0	1	0	2	1	0	1

Perceptron : exemple

	X0	X1	X2	X3	X4	w0	w1	w2	w3	w4	c	o	c-o	w0	w1	w2	w3	w4
E1	1	0	0	0	1	0	2	1	0	1	0	1	-1	-1	2	1	0	0
E2	1	0	1	1	1	-1	2	1	0	0	0	1	-1	-2	2	0	-1	-1
E3	1	1	1	0	1	-2	2	0	-1	0	1	0	1	-1	3	1	-1	0
E4	1	0	0	1	0	-1	3	1	-1	0	0	0	0	-1	3	1	-1	0
E5	1	1	0	1	1	-1	3	1	-1	0	1	0	1	-1	3	1	-1	0

Perceptron : exemple

	X0	X1	X2	X3	X4	w0	w1	w2	w3	w4	c	o	c-o	w0	w1	w2	w3	w4
E1	1	0	0	0	1	-1	3	1	-1	0	0	0	0	-1	3	1	-1	0
E2	1	0	1	1	1	-1	3	1	-1	0	0	0	0	-1	3	1	-1	0
E3	1	1	1	0	1	-1	3	1	-1	0	1	1	0	-1	3	1	-1	0
E4	1	0	0	1	0	-1	3	1	-1	0	0	0	0	-1	3	1	-1	0
E5	1	1	0	1	1	-1	3	1	-1	0	1	1	0	-1	3	1	-1	0

Perceptron : exemple

	X0	X1	X2	X3	X4	w0	w1	w2	w3	w4	c	o	c-o	w0	w1	w2	w3	w4
E1	1	0	0	0	1	-1	3	1	-1	0	0	0	0	-1	3	1	-1	0

Le réseau se stabilise

Perceptron : exemple

- Phase de test du réseau

	X1	X2	X3	X4	c
T1	1	0	1	0	**1**
T2	0	1	1	0	**0**
T3	1	0	0	0	**0**
T4	1	1	1	0	**1**
T5	1	0	1	1	**1**

Perceptron : exemple

- Phase de test du réseau

	X1	X2	X3	X4	c	o
T1	1	0	1	0	**1**	**1**
T2	0	1	1	0	**0**	**0**
T3	1	0	0	0	**0**	**1**
T4	1	1	1	0	**1**	**1**
T5	1	0	1	1	**1**	**1**

Précision =(1/2+3/3)/2
= 75 %
Rappel = (1/1+3/4)/2
=88 %

Perceptron

Et si l'échantillon d'entrée n'est pas linéairement séparable ?

- l'algorithme ne convergera pas vers un taux d'erreur nul et il n'existe aucun moyen de le savoir.
- par contre, si les poids prennent deux fois les mêmes valeurs sans que le perceptron ait appris alors que tous les exemples ont été présentés signifie que l'échantillon n'est pas séparable.

Perceptron : Conclusion

Apprentissage par perceptron

- techniques de séparation linéaire
- méthodes non paramétriques : pas d'hypothèse sur les données

Les échantillons de moins de $2n$ exemples sont linéairement séparables lorsque n est le nombre de variables

- Une classification correcte d'un petit échantillon n'a donc aucune valeur prédictive
- Par contre, lorsque l'on travaille sur suffisamment de données et que le problème s'y prête, on constate empiriquement que le perceptron appris par un des algorithmes précédents a un bon pouvoir prédictif

Perceptron : Conclusion

La plupart des problèmes d'apprentissage ne peuvent pas être résolus par des méthodes aussi simples :

- il est rare que les exemples se répartissent "naturellement" de part et d'autre d'un hyperplan.

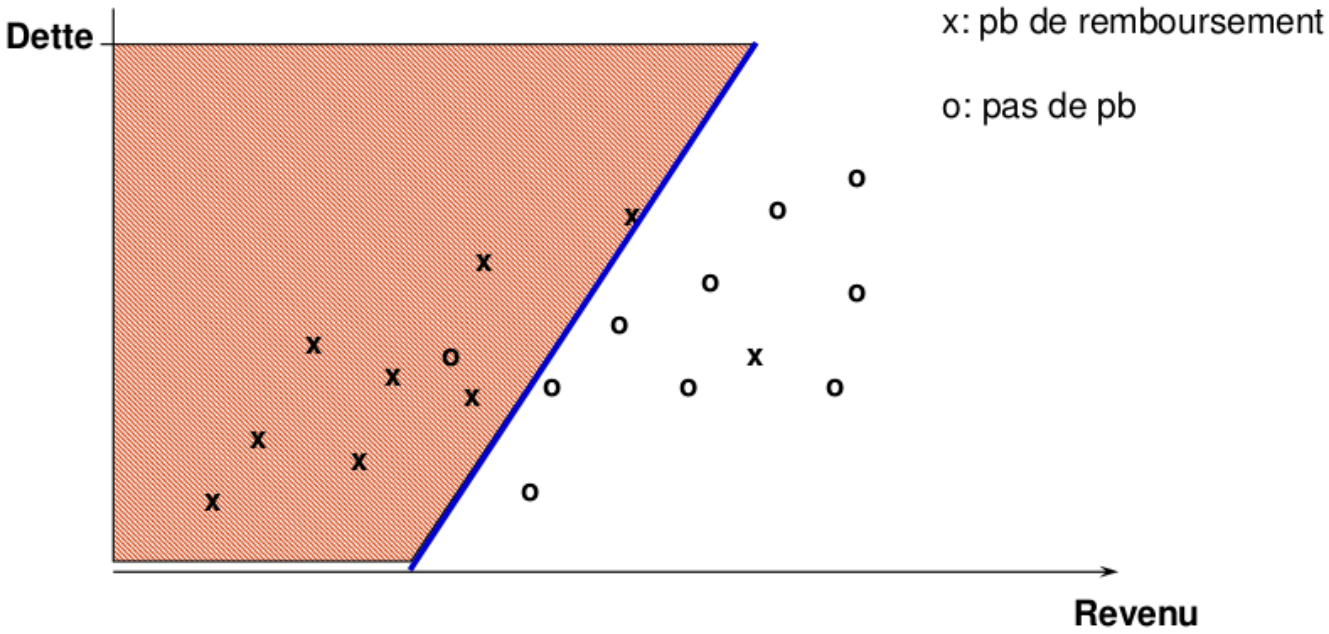
Il faut :

- soit mettre au point des séparateurs non-linéaires,
- soit complexifier l'espace de représentation de manière à linéariser le problème initial.

$\rightarrow$ C'est ce que allons étudier maintenant.

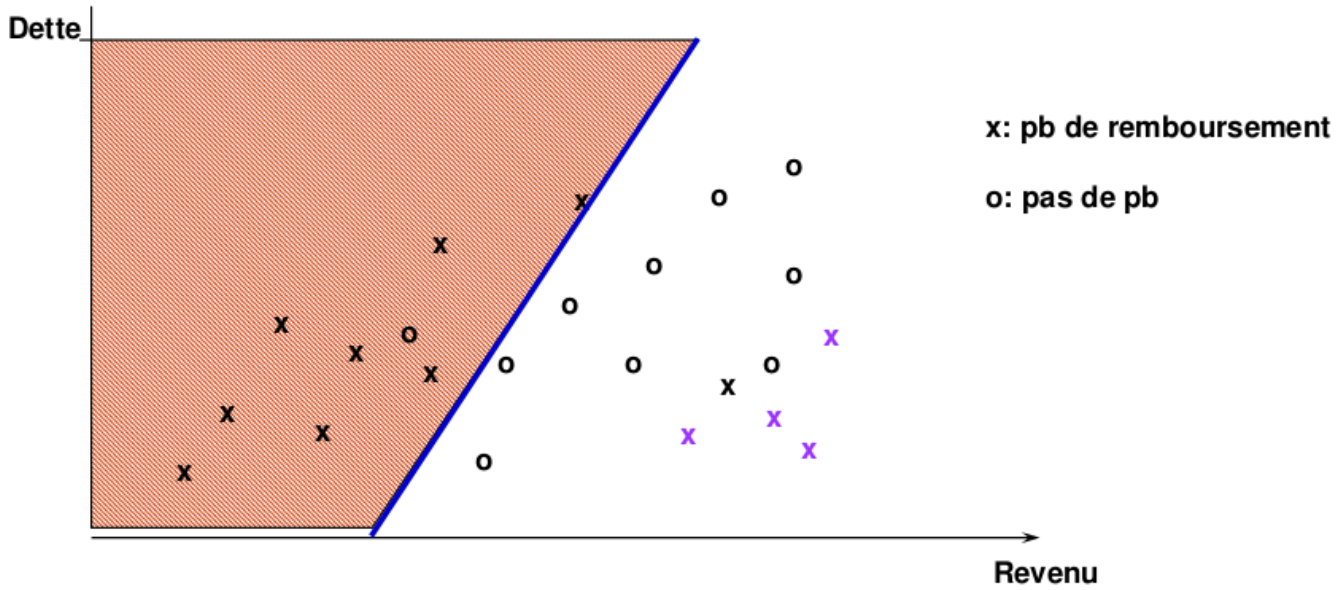
Perceptron multi-couches

- Séparation linéaire



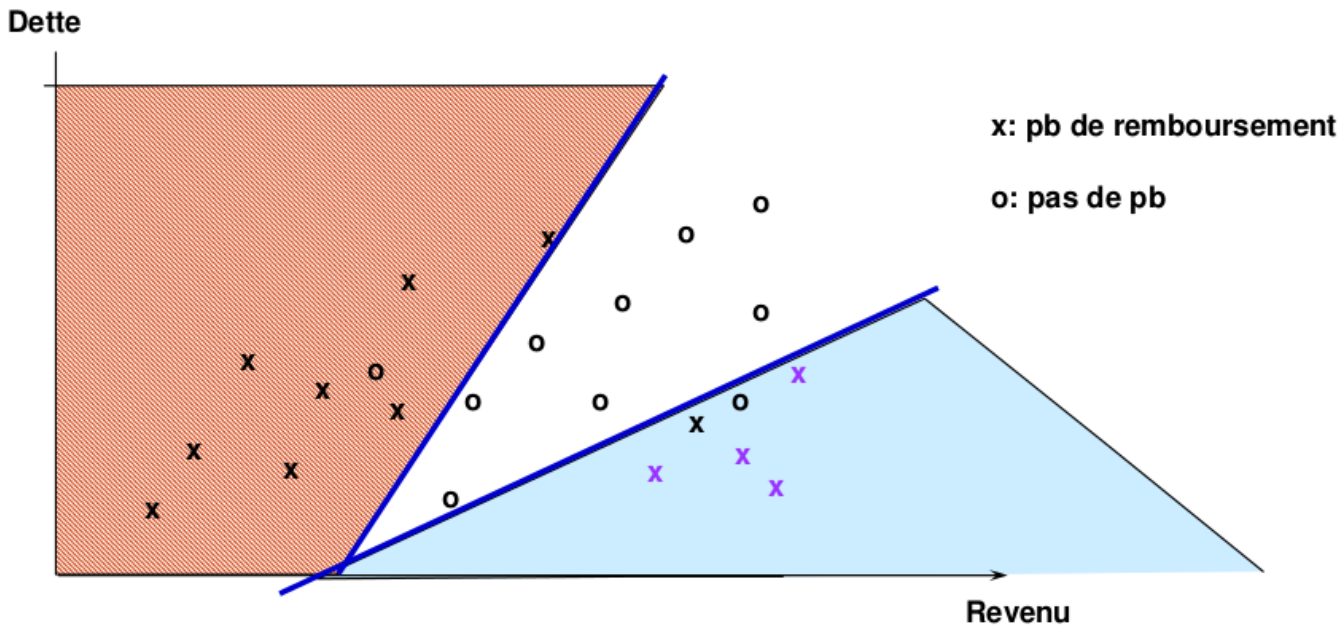
Perceptron multi-couches

- On rajoute des exemples



Perceptron multi-couches

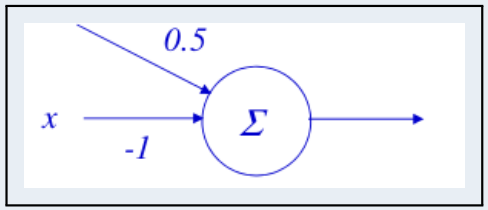
- Problème : comment faire la différence entre les différents 'x' ?



Perceptron multi-couches

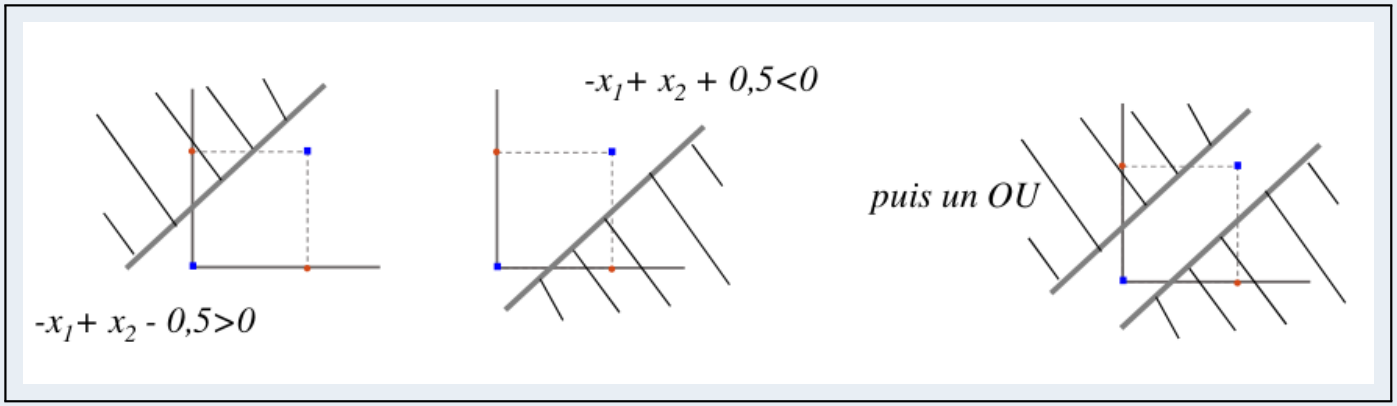
Revenons à la fonction booléenne XOR(x,y)

- 1ère solution
  - On la décompose en : $(\overline{\overline{x}.y})(\overline{x.\overline{y}})$
  - Il suffit alors de savoir coder la fonction NON :

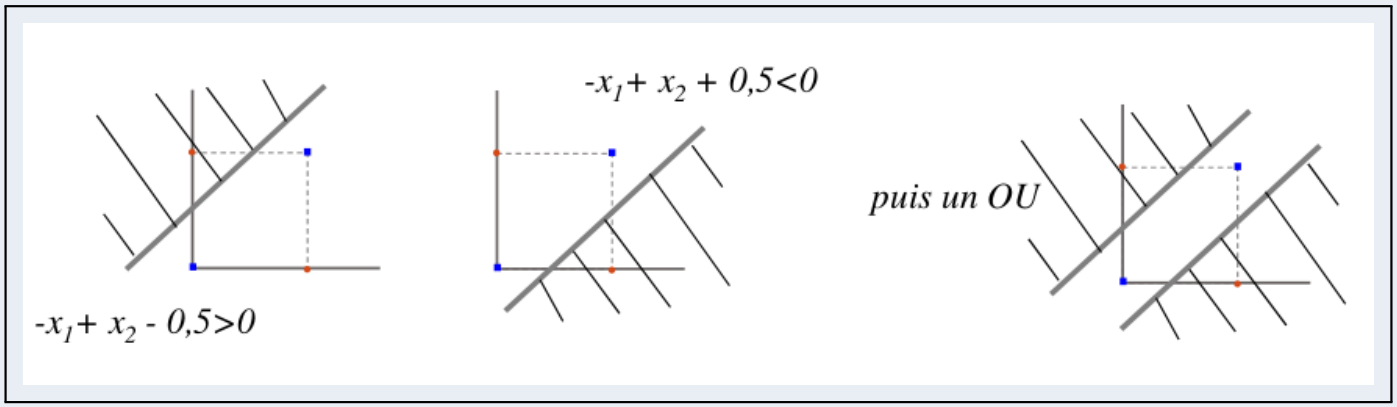


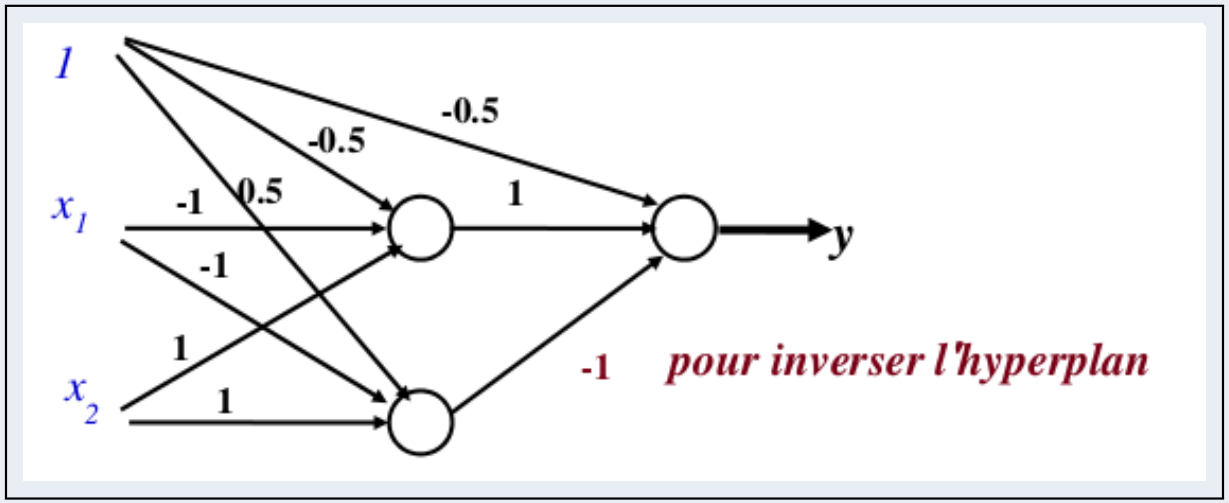
  - Puis on crée un réseau assez complexe ....
- 2ème solution :
  - On analyse la fonction en terme d'hyperplan

Perceptron multi-couches



Perceptron multi-couches





Perceptron multi-couches

Toute fonction booléenne peut être calculée par un PMC linéaire à seuil comprenant une seule couche cachée.

- la couche cachée pourra contenir jusqu'à $2n$ neurones (où n est la taille de la rétine), ce qui est **inacceptable** en pratique.

Pour pouvoir utiliser les réseaux multicouches, il faut :

- une méthode permettant de choisir une architecture de réseau : combien de couches ? combien de neurones par couche ?
- un algorithme d'apprentissage des poids des connexions.

Perceptron multi-couches

Apprentissage de l'architecture (sujet de recherche actif)

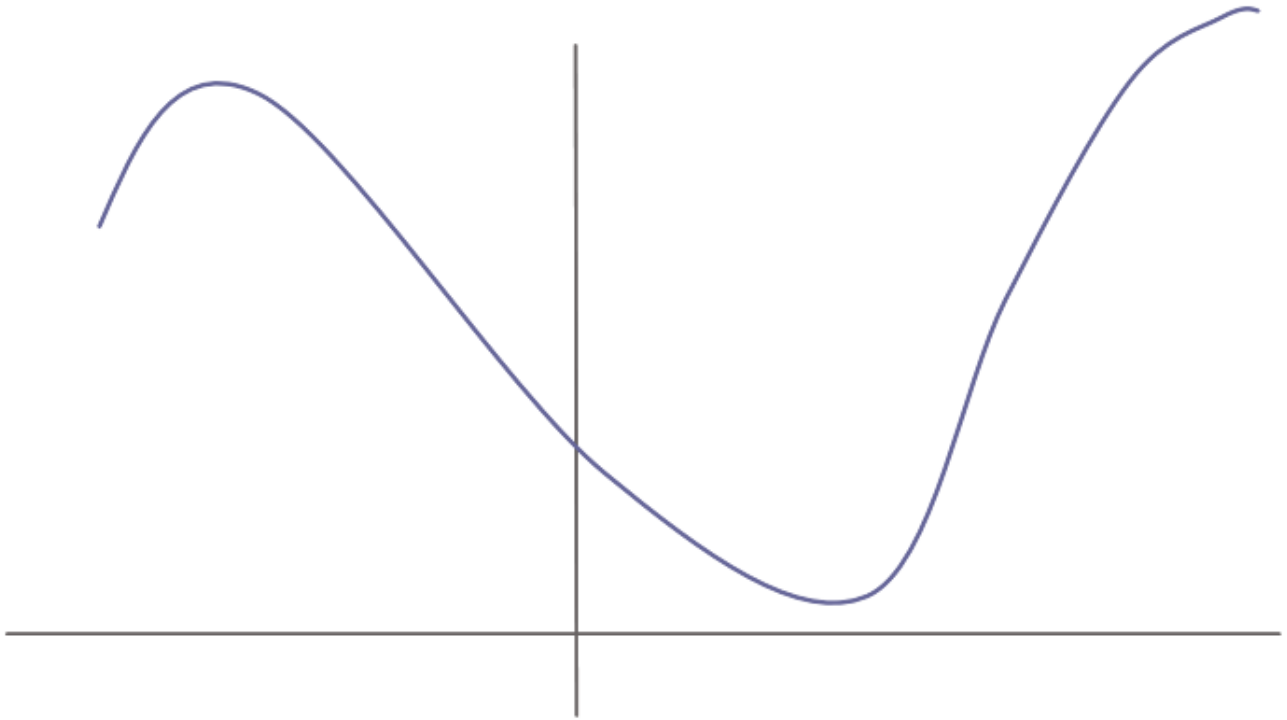
- algorithmes d'apprentissage auto-constructifs
- apprentissage de l'échantillon avec un réseau courant,
- modification du réseau courant, (ajout/suppression de neurones et/ou de couches) en cas d'échec de l'apprentissage.

Apprentissage des poids :

- algorithme de rétropropagation du gradient

Descente du gradient

- Soit une fonction f à minimiser.

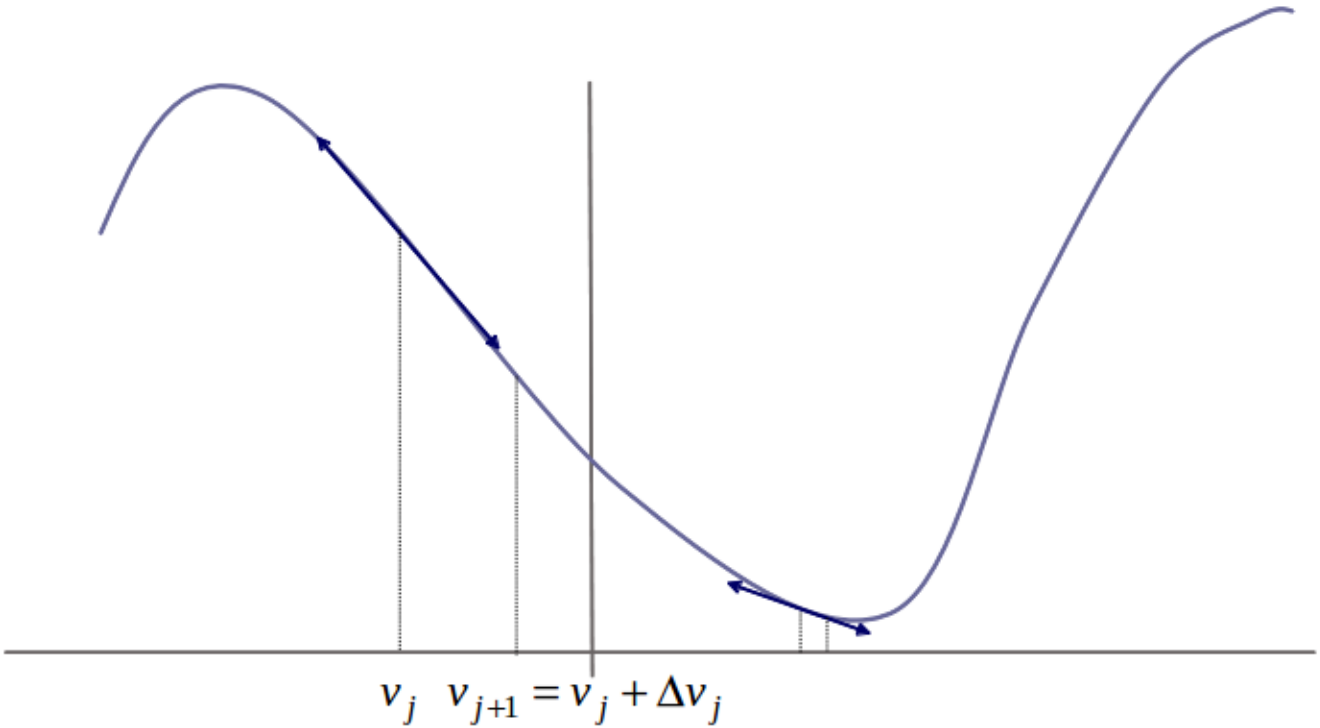


Descente du gradient

- Soit une fonction f à minimiser.
- **Idée** : On génère une suite de valeurs $v_{j+1} = v_j + \Delta v_j$ de façon à ce que cette suite converge vers le minimum local $f(v_{j+1}) < f(v_j)$
- Or f est décroissante autour d'un minimum. Sa dérivée est nulle en ce minimum : la pente de cette dérivée est de plus en plus petite.
- On va utiliser cette propriété pour guider la génération de la suite des valeurs
- Rappel : Dans notre cas nous recherchons les meilleures valeurs $w_1, ..., w_n$

Descente du gradient

- Soit une fonction f à minimiser.



Descente du gradient

Principe de la descente du gradient : $v_j = v_j + \Delta v_j$:

- On génère une suite de valeurs $f(v_{j+1}) < v_j$ de façon à ce que cette suite converge vers le minimum local
- Soit $\Delta v_j = -\epsilon f'(v_j)$ avec ϵ "bien" choisi

$$f(v_{j+1}) = f(v_j - \epsilon f'(v_j)) \approx f(v_j) - \epsilon(f'(v_j))^2 < f(v_j)$$

(théorème des approximations finies).

- on se rapproche à chaque itération du minimum local
- Rappel : Dans notre cas nous recherchons les meilleures valeurs $w_1, ..., w_n$: $w_j = w_j + \Delta w_j$ *avec* $\Delta w_j = -\epsilon \times \frac{\partial(E)}{\partial w_j}$

Descente du gradient

La pente est nulle pour un maximum (ou minimum) local.

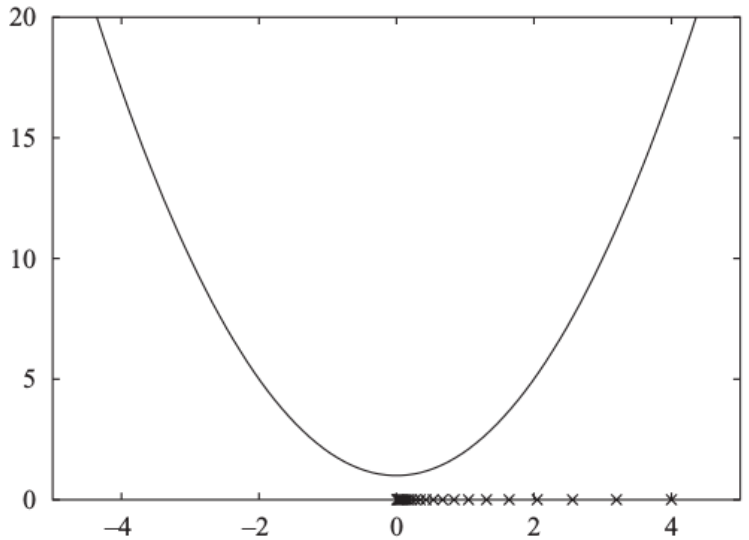
- On peut s'arrêter lorsque cette pente est suffisamment faible

Inconvénients de cette méthode :

1. Choix de ϵ empirique : si ϵ est trop petit, le nombre d'itérations peut-être élevé, s'il est trop grand risque d'oscillations sans convergence
2. Sans information sur la fonction à minimiser, rien ne garantit que le minimum trouvé soit un minimum global

Exemple

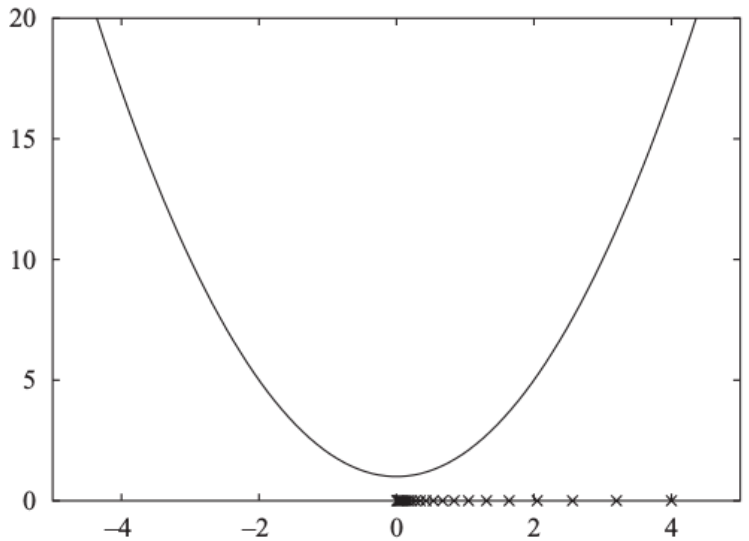
- Fonction d'erreur : $w^2 + 1$
- Dérivée de la fonction d'erreur : $2w$
- Si la dérivée est négative la courbe est décroissante vers la droite, s'il est positive, elle est décroissante vers la gauche



source : Data Mining: Practical Machine Learning Tools and Techniques

Exemple

- La valeur de la dérivée indique *l'importance* de la décroissance
- ϵ permet de contrôler la force de l'influence de la dérivée
- Avec $\epsilon = 0.1$, $w = 4$, la dérivée vaut 8, multiplié par 0.1 et soustrait à 4, on obtient 3.2
- Si on répète on obtient 3.2, puis 2.56, puis 2.048, etc.



source : Data Mining: Practical Machine Learning Tools and Techniques

Perceptron et descente du gradient

- Cas du percepton : Il faut minimiser l'erreur quadratique d'apprentissage par rapport aux poids $w = (w_1, ...w_j, ..., w_n)$.
- Soit N le nombre de données d'apprentissage : $X = x_1 x_2 ... x_N$.

- Rappel on cherche à calculer : $w_j = w_j + \Delta w_j \; avec \; \Delta w_j = -\varepsilon \times \frac{\partial(E)}{\partial w_j}$

$$\frac{\partial(E)}{\partial w} = \frac{\partial}{\partial w} . \frac{1}{2} \sum_{i=1}^{N} (c_i - o_i)^2 = \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{N} \frac{\partial}{\partial w_j} (c_i - o_i)^2$$

d'où $$\frac{\partial(E)}{\partial w} = \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{N} 2 \times (c_i - o_i) \frac{\partial}{\partial w_j} (c_i - o_i) = \sum_{j=1}^{n} \sum_{i=1}^{N} (c_i - o_i) \frac{\partial}{\partial w_j} (c_i - w.x_i)$$

d'où différents $$\frac{\partial(E)}{\partial w} = -\sum_{j=1}^{n} \sum_{i=1}^{N} (c_i - o_i)(x_i)_j$$

car tous les poids de j ont une dérivée nulle.

Perceptron : descente du gradient

- On peut extraire :

$$\Delta w_j = -\epsilon \times \frac{\partial(E)}{\partial w_j} = -\epsilon \times \sum_{i=1}^{N} (c_i - o_i)(x_i)_j$$

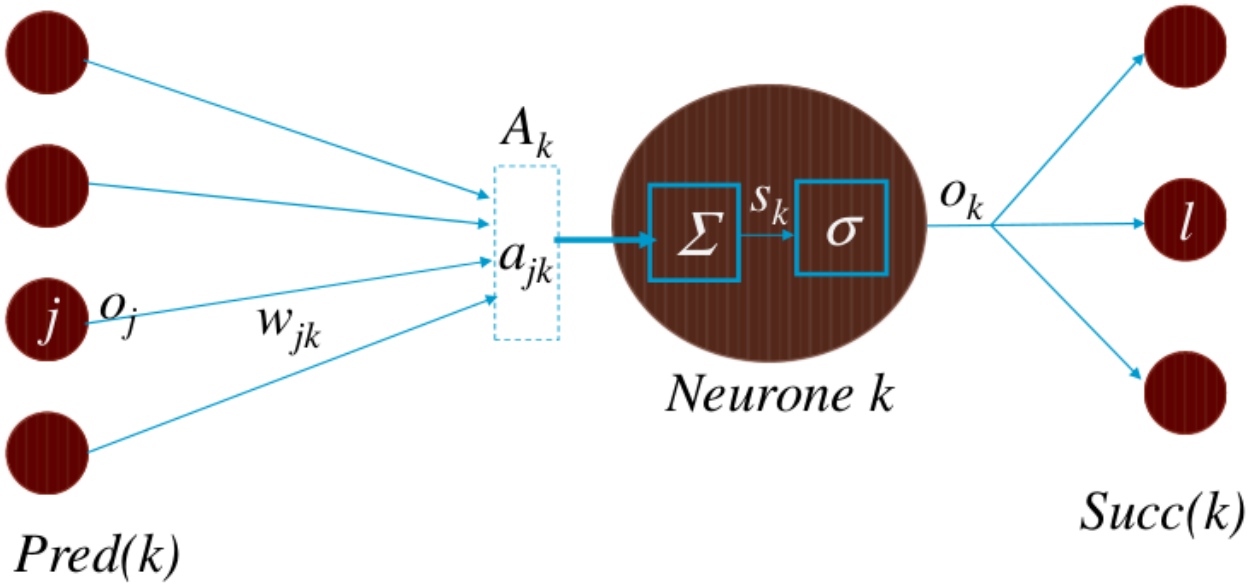
- On retrouve la formule précédente (corresp. à $\epsilon = 1$)

La fonction d'erreur quadratique étant un paraboloïde, elle ne possède qu'un minimum.

- L'algorithme est assuré de converger même si l'échantillon n'est pas linéairement séparable avec néanmoins risque d'oscillation si ϵ est trop grand : on général, on fait décroître ϵ au cours du temps.

Rétropropagation du gradient

- Cas général : la j-ième entrée a_{jk} du neurone k correspond à la sortie du j-ième neurone en entrée



Rétropropagation du gradient

Principe de l'algorithme de rétropropagation

- Pour chaque neurone, en fonction d'une sortie "désirée" E_k :
  1. modifier les poids en entrée
  2. calculer la sortie désirée de chacun des neurones en entrées
  3. rétropropager ces sorties "désirées" sur les neurones de la couche précédente

Posons E_{kl} la sortie désirée calculée pour le k-ième neurone en entrée du neurone l alors :

$$E_k = \frac{1}{ns_k} \sum_{l \in Succ(k)} E_{kl}$$

où ns_k est le nombre de successeurs du neurone k

Rétropropagation du gradient

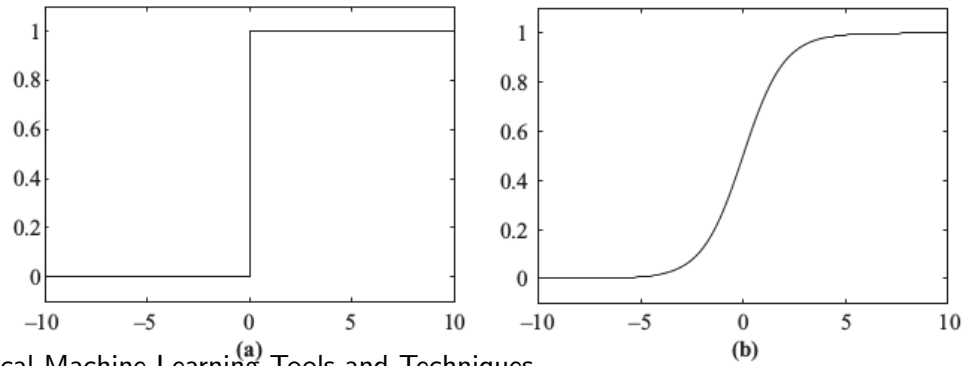
Cas général : basé sur la méthode du gradient

- Rappel - Il faut calculer : $\Delta w_{jk} = -\epsilon \times \frac{\partial(E_k)}{\partial w_{jk}}$

où w_{jk} est le poids associé au lien entre le neurone j et k

La fonction de sortie d'un neurone doit être dérivable :

- impossible de considérer un perceptron linéaire à seuil.
- L'activation d'un neurone sera une sigmoïde



source : Data Mining: Practical Machine Learning Tools and Techniques

Rétropropagation du gradient

- Une fonction sigmoïde de paramètre $k > 0$ est définie par : $\sigma_k(s) = \frac{e^{ks}}{1+e^{ks}}$
- Nous prendrons $k = 1$ dans la suite $\sigma_k(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$
- Cette fonction est facilement dérivable :

$$\sigma'(s) = \frac{e^s}{(1+e^s)^2} = \sigma(s)(1-\sigma_s)$$

Cette propriété est importante car cette dérivée va être utilisée dans la règle d'apprentissage

- La sortie d'un neurone k sera donc :

$$y = o_k = \frac{1}{1+e^{-s}}$$

où $s = \sum w_j.a_{kj}$

Rétropropagation du gradient

Cas général : basé sur la méthode du gradient

- Il faut donc calculer $\Delta w_{jk} = -\epsilon \times \frac{\partial E_k}{\partial w_{jk}}$
- Comme w_{jk} ne peut influencer la sortie du neurone k qu'à travers le calcul de la quantité e_k :

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial e_k} \frac{\partial e_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial e_k} a_{jk}$$

il faut calculer $\frac{\partial E_k}{\partial e_k}$

Rétropropagation du gradient

Calcul de $\frac{\partial E_k}{\partial e_k}$

- la quantité e_k ne peut influencer la sortie du réseau que par le calcul de o_i, d'où

$$\frac{\partial E_k}{\partial e_k} = \frac{\partial E_k}{\partial o_k} \frac{\partial o_k}{\partial e_k}$$

- Pour faire ce calcul (et modifier les poids en conséquence), deux cas sont à étudier :
  1. le neurone $k, 1 \leq k \leq P$ fait partie de la couche de sortie
  2. le neurone récepteur fait partie d'une couche cachée

Rétropropagation du gradient

1. Si k est un neurone de sortie

- la quantité e_i ne peut influencer la sortie du réseau que par le calcul de o_i.

$$\frac{\partial E_k}{\partial e_k} = \frac{\partial E_k}{\partial o_k}\frac{\partial o_k}{\partial e_k}$$

- Calculons les deux dérivées partielles

$$\frac{\partial E_k}{\partial o_k} = \frac{1}{2}\frac{\partial}{\partial o_k}(c_k - o_k)^2 = -(c_k - o_k)$$

où c_k est le résultat attendu sur la sortie k pour la donnée présentée

$$\frac{\partial o_k}{\partial e_k} = \frac{\partial \sigma(e_k)}{\partial e_k} = \sigma(e_k)(1 - \sigma(e_k)) = o_k(1 - o_k)$$

$$\boxed{\frac{\partial E_k}{\partial e_k} = -(c_k - o_k)o_k(1 - o_k)}$$

Rétropropagation du gradient

2. Si i est un neurone d'une couche cachée

- la quantité e_k va influencer le réseau à travers tous les neurones reliés à la sortie du neurone k.

$$\frac{\partial E_k}{\partial e_k} = \sum_{l \in Succ(k)} \frac{\partial E_{kl}}{\partial e_l} \frac{\partial e_l}{\partial e_k} = \sum_{l \in Succ(k)} \frac{\partial E_{kl}}{\partial e_l} \frac{\partial e_l}{\partial o_k} \frac{\partial o_k}{\partial e_k}$$

$$\frac{\partial E_k}{\partial e_k} = \sum_{l \in Succ(k)} \frac{\partial E_{kl}}{\partial e_l} \times w_{kl} \times o_k(1 - o_k)$$

- Il suffit de connaître les sorties désirées sur les neurones connectés au neurone k pour savoir comment sa sortie désirée varie en fonction de ses entrées

Rétropropagation du gradient

En résumé, on peut calculer :

$$\delta w_{jk} = -\epsilon \times \frac{\partial E_k}{\partial w_{jk}} = -\epsilon \frac{\partial E_k}{\partial e_k} \frac{\partial e_k}{\partial w_{jk}} = -\epsilon \frac{\partial E_k}{\partial e_k} a_{jk}$$

posons $\delta_k = \partial \frac{E_k}{\partial e_k}$

1. Pour un neurone de sortie $\delta_k = -(c_k - o_k)o_k(1 - o_k)$ avec c_k est le résultat attendu sur la sortie k pour la donnée présentée
2. Pour un neurone interne : $\delta_k = o_k(1 - o_k) \times \sum_{l \in Succ(k)} \delta_l w_{kl}$

Rétropropagation du gradient

L'algorithme :

- Initialisation aléatoirement tous les poids
- Pour chaque entrée X :
  1. Calculer $O = (r_1, r_2, ...r_p, ..., r_P)$ la sortie du réseau
  2. Pour tout neurone de sortie p : $\delta p = r_p(1 - r_p)(c_p - r_p)$ où c_p est le résultat attendu pour X sur la sortie p
  3. Pour chaque couche C de $q - 1$ à 1 :
  4. Pour chaque neurone k de la couche C :

$$\delta_k = o_k(1 - o_k) \sum_{l \in Succ(k)} \delta_l w_{kl}$$

  5. Pour chaque lien j en entrée de k : $w_{jk} = w_{jk} - \epsilon \delta_k a_{jk}$

Conclusion sur la rétropropagation

- Donne de bons résultats pratiques.
- Problème des minimums locaux
  - une variante couramment utilisée consiste à pondérer la modification des poids en fonction du nombre d'itérations déjà effectuées.
  - Soit $\alpha \in [0, 1[$ appelé moment (*momentum*) et t un compteur du nombre d'itérations de la boucle principale, la règle de modification des poids devient :

$$w_{ij} = w_{ij} + \Delta w_{ij}(t) \quad \text{avec} \quad \Delta w_{ij}(t) = \epsilon \delta_i x_{ij} + \alpha \Delta w_{ij}(t-1)$$

- L'intérêt de cette règle est de prendre en compte les modifications antérieures des poids afin d'éviter des oscillations perpétuelles.
- Améliore aussi la vitesse de convergence

Conclusion sur la rétropropagation

Critère d'arrêt ?

- Non précisé dans l'algorithme.
- Problème de sur-spécialisation si on se base uniquement sur l'erreur observée $\rightarrow$ utiliser un ensemble test

Choix des "bonnes" valeurs pour les paramètres ϵ et α.

- On découpe l'ensemble d'apprentissage en un ensemble d'apprentissage, un ensemble de validation et un ensemble test.
- Lors de l'apprentissage, on arrête périodiquement l'apprentissage, on estime l'erreur réelle sur l'ensemble de validation, on met à jour
- Les paramètres ϵ et α en fonction de la variation de cette erreur estimée.
- L'ensemble test sert à estimer l'erreur réelle à la fin de l'apprentissage.

Conclusion sur la rétropropagation

- Mode de présentation des exemples : répartition équivalente
- Efficacité en apprentissage
  - En O(nb de poids) pour chaque passe d'apprentissage,
  - Il faut typiquement plusieurs centaines de passes et recommencer plusieurs dizaines de fois un apprentissage avec différentes initialisations des poids
- Quel architecture pour un problème donné ?
  - Fait par l'expérience.
  - Méthodes autoconstructives : Ajouts/suppressions de neurones au cours de l'apprentissage risque de sur-spécialisation.
  - Architecture choisie à l'aide d'algorithmes génétiques.
- Efficacité des PMC en reconnaissance : possibilité de temps réel

Perceptron multi-couches : Conclusion

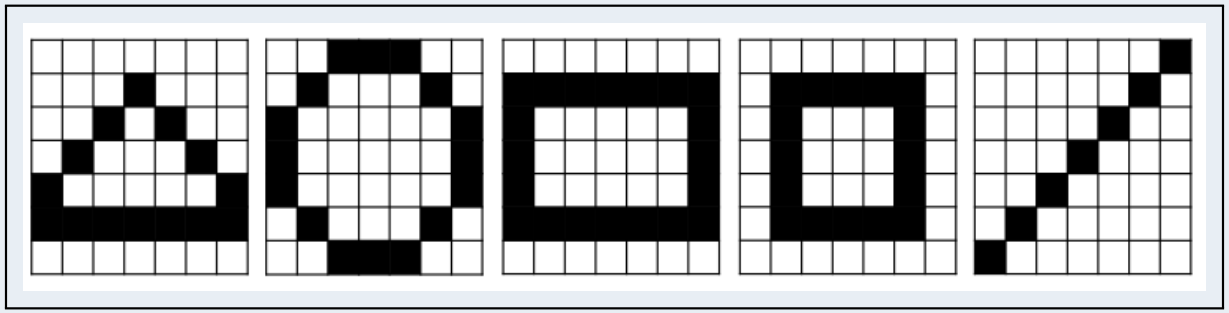
Puissance de représentation des réseaux multi- couches ?

- Toute fonction booléenne peut être calculée par un PMC linéaire à seuil à une seule couche cachée pouvant être de taille exponentielle.

Ce résultat a été généralisé par Hornik en 1989 :

- La plupart des fonctions numériques peuvent être approximées avec une précision arbitraire par des réseaux à une seule couche cachée.
- Mais cette couche cachée peut être démesurément grande
- Ce théorème est essentiellement un résultat théorique sur l'expressivité des réseaux multi-couches.

Reconnaissance de formes géométriques



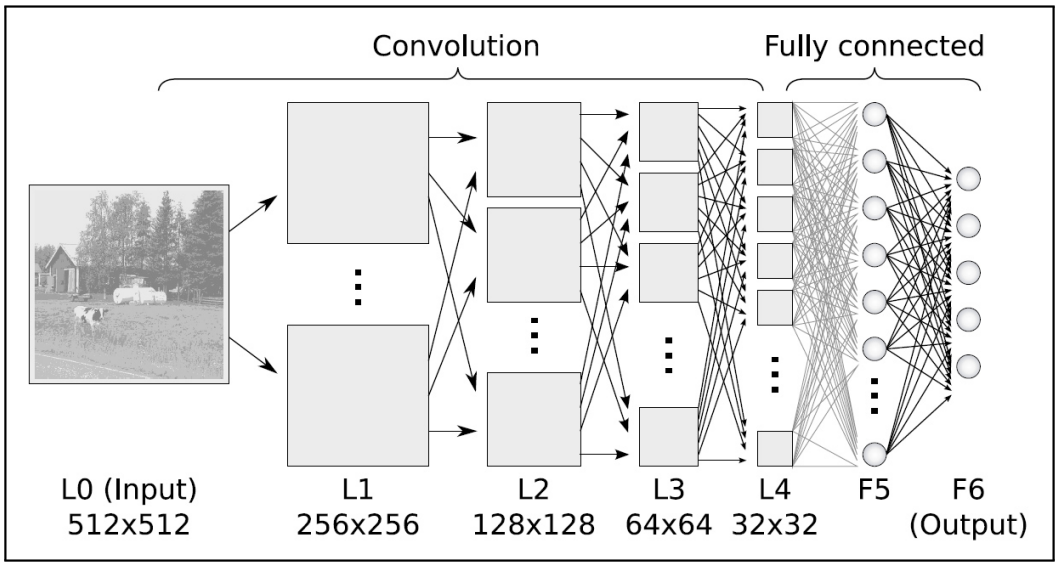
- Les formes sont représentées par des matrices binaires de 7 x 7 pixels, pouvant être allumés (valeur 1) ou éteints (valeur 0). Ceci aboutit naturellement à 49 neurones d'entrée.
- Les formes reconnues sont codées sur 5 bits, un pour chaque forme possible. Un bit est à 1 si la forme correspondante est reconnue, ou à 0 sinon.

Deep Learning

- un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires
- en fort développement depuis 2010, basé sur les réseaux de neurones
- très utilisé en analyse d'images (Google, Facebook, etc.)

Convolutional neural network

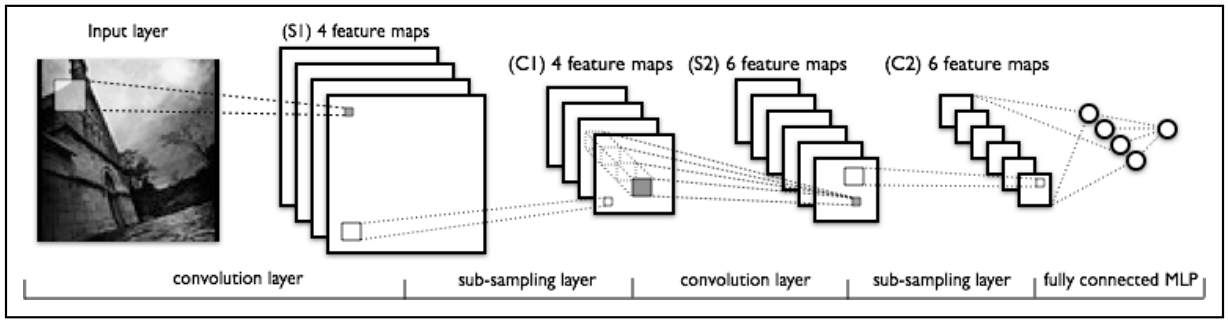
- CNN : *Convolutional neural network*
- L'objectif est d'apprendre des hiérarchies de descripteurs où les descripteurs d'un niveau supérieur sont construits à partir des informations du niveau inférieur
- Combinaison de plusieurs couches de traitements non-linéaires



source : `http://www.ais.uni-bonn.de/`

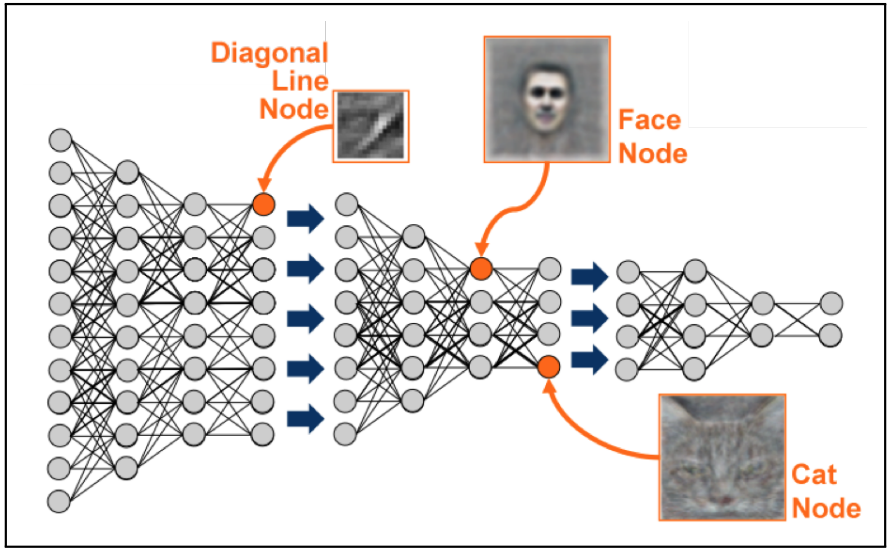
Convolutional neural network

- CNN : *Convolutional neural network*
- L'objectif est d'apprendre des hiérarchies de descripteurs où les descripteurs d'un niveau supérieur sont construits à partir des informations du niveau inférieur
- Combinaison de plusieurs couches de traitements non-linéaires



Convolutional neural network

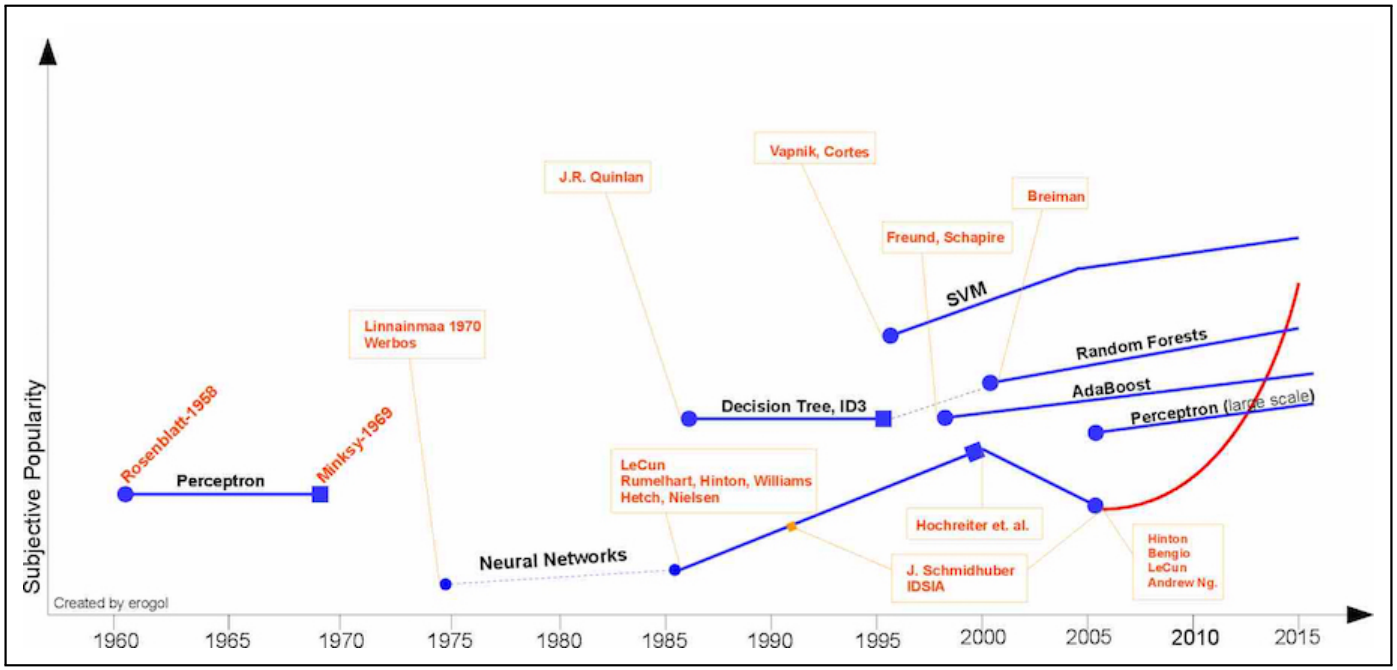
- CNN : *Convolutional neural network*
- L'objectif est d'apprendre des hiérarchies de descripteurs où les descripteurs d'un niveau supérieur sont construits à partir des informations du niveau inférieur
- Combinaison de plusieurs couches de traitements non-linéaires



source : `http://scyfer.nl/`

Convolutional neural network

- développement rapide grâce aux GPU
- nombreuses librairies implémentant les CNN disponibles



source : `http://www.kdnuggets.com`