Apprentissage artificiel

4h CM, 4h TD, 8h TP

- généralités sur l'apprentissage artificiel

- une technique : les réseaux de neurones

www.u-picardie.fr/~furst/apprentissage.php

Principe de l'apprentissage artificiel (machine learning) : à partir de données exemples (numériques et/ou symboliques), le programme doit apprendre un « modèle » des données.

Le modèle est une généralisation des données et permet, par la suite, de prendre des décisions ou faire des calculs.

Exemples :

- *on a des données décrivant des historiques de parcours d'étudiant en licence informatique (notes, type de bac, assiduité, ...), on veut obtenir un modèle prédictif de la note finale de master*
- *on veut obtenir un programme qui reconnaît des chiffres manuscrits*
- *on veut apprendre à un robot à se déplacer dans des environnements variés et imprévisibles*

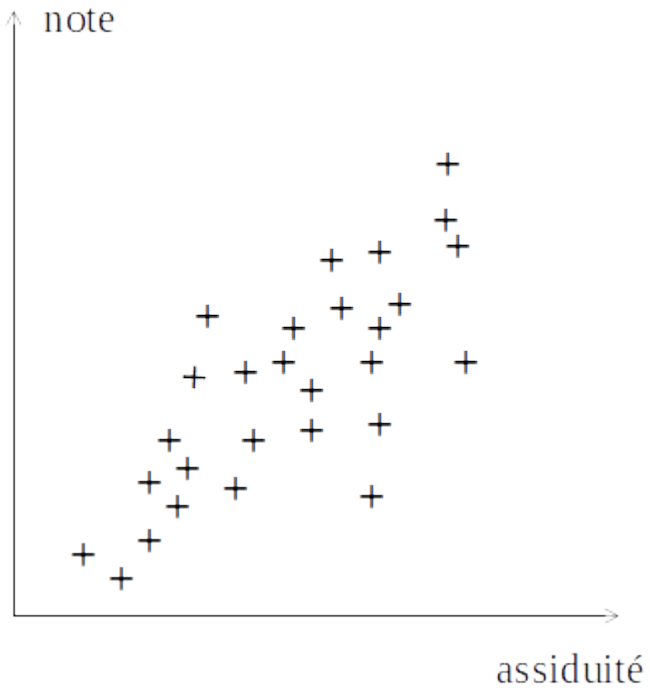
L'histoire et l'objectif de l'apprentissage artificiel en font une branche de l'intelligence artificielle.

Les outils et méthodes de l'apprentissage artificiel sont principalement issus des statistiques et de l'optimisation.

Classifier : découper l'ensemble des items d'apprentissage en catégories.

Le partitionnement (clustering) vise à découper les données en sous-ensembles ayant des propriétés communes (notion de distance/similarité).

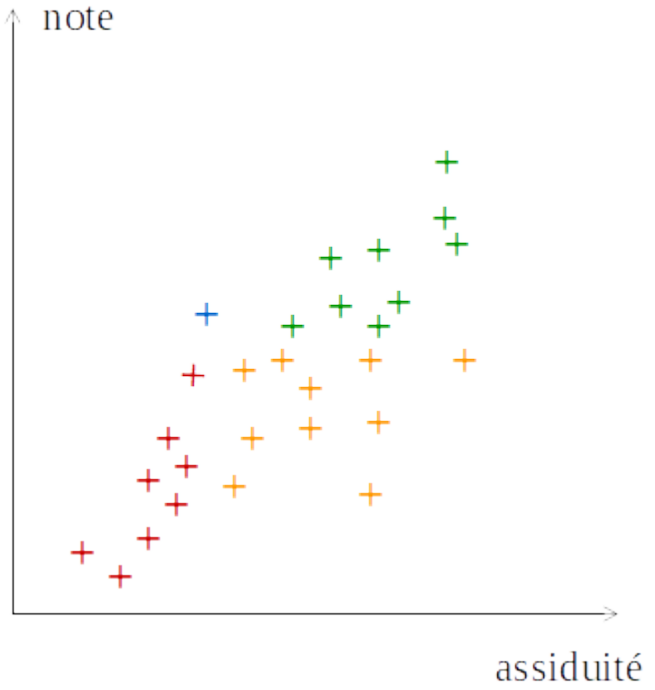
Exemple : classer les étudiants selon leurs notes et leur assiduité



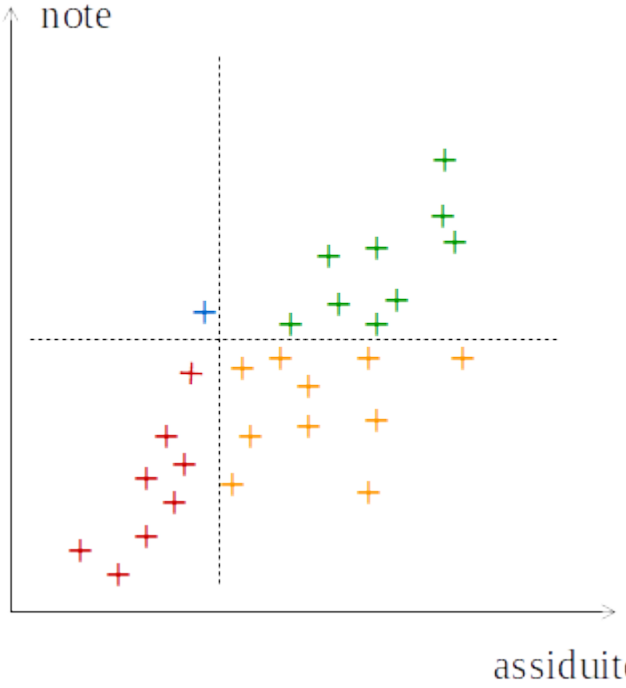
1- Définir les classes :

classe 1 : étudiants modèles (assidu, bonnes notes)
classe 2 : cancres (sèche les cours, mauvaises notes)
classe 3 : étudiants à aider (assidu, mauvaises notes)
classe 4 : génies (sèche les cours, bonnes notes)

2- Étiqueter les données



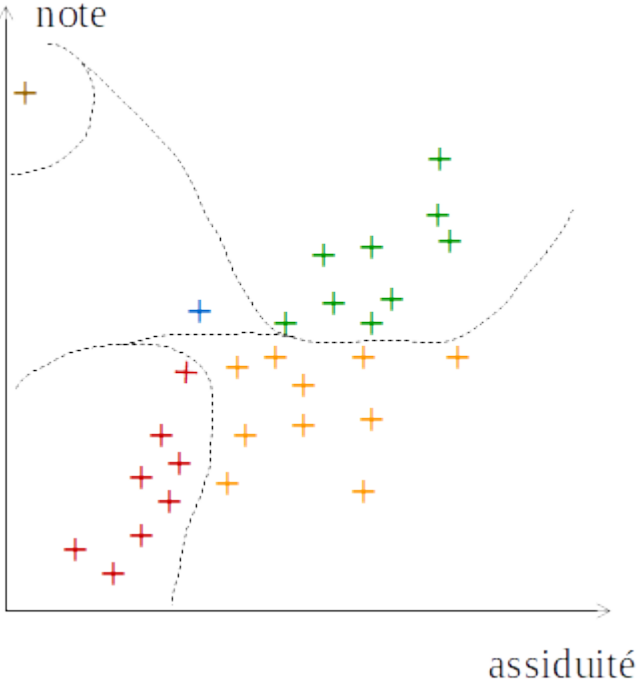
3- Généraliser en trouvant une règle automatique d'étiquetage



classe 1 : étudiants modèles (assidu, bonnes notes)
classe 2 : cancres (sèche les cours, mauvaises notes)
classe 3 : étudiants à aider (assidu, mauvaises notes)
classe 4 : génies (sèche les cours, bonnes notes)

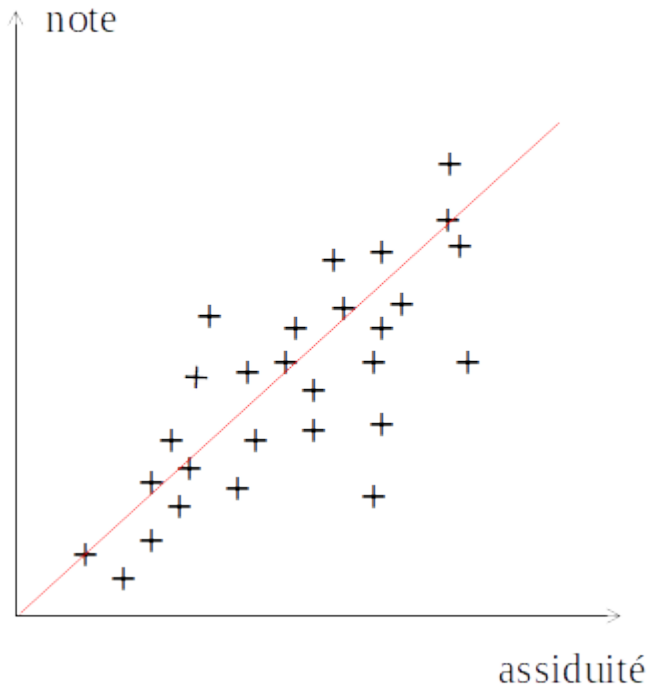
La règle peut être très complexe :

- *on peut aussi aider les étudiants très assidus mais ayant des notes correctes*
- *on peut avoir une 5e classe d'étudiants tricheurs*
- *les étudiants peuvent être décrits par bien plus que deux variables*

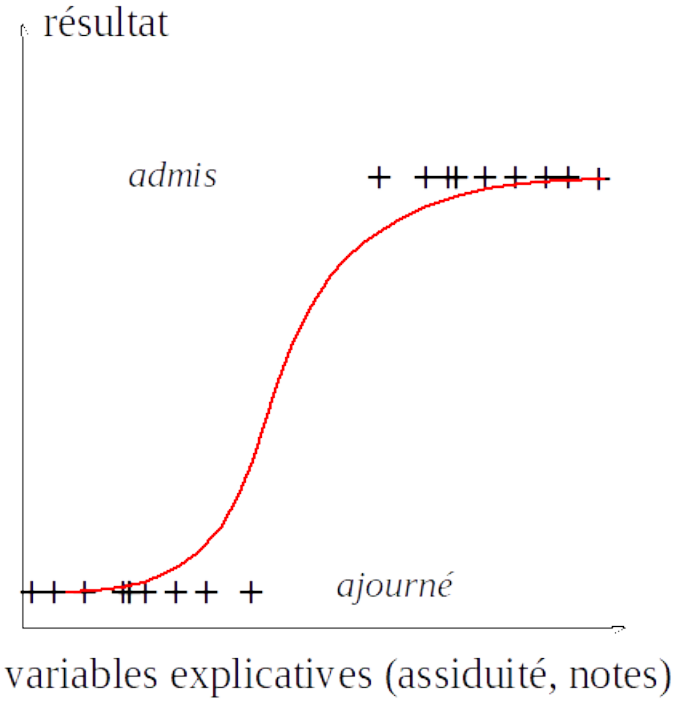


Régression : obtenir une relation entre variables numériques (continues)

<u>Exemple</u> : trouver une régression linéaire entre assiduité et notes d'étudiants



Dans le cas d'un résultat binaire on utilise une régression logistique



Les fonctions logistiques sont de la forme $\frac{\alpha}{1+\beta e^{-\lambda x}}$ avec α et λ positifs

L'apprentissage supervisé : les données exemples sont étiquetées

Exemple : reconnaître un visage sur des photos

L'apprentissage non supervisé : les données exemples ne sont pas étiquetées, les classes doivent être déterminées par l'algorithme

Exemple : déterminer les profils utilisateurs d'une application massivement utilisée

L'apprentissage semi-supervisé : certaines données exemples sont étiquetées mais pas toutes

Exemple : les données sont très massives et il est impossible de tout étiqueter

L'apprentissage par renforcement : les données exemples ne sont pas étiquetées, mais on dispose d'une fonction d'évaluation du modèle appris

Exemple : apprendre à un robot à se déplacer sur terrain accidenté

1- préparer les données
- choisir les données exemples (les données de validation)
- supervisé ou semi-supervisé : étiqueter les exemples/contre-exemples

2- déterminer le modèle et la méthode d'apprentissage les plus adaptés
- choisir un modèle
- paramétrer le modèle

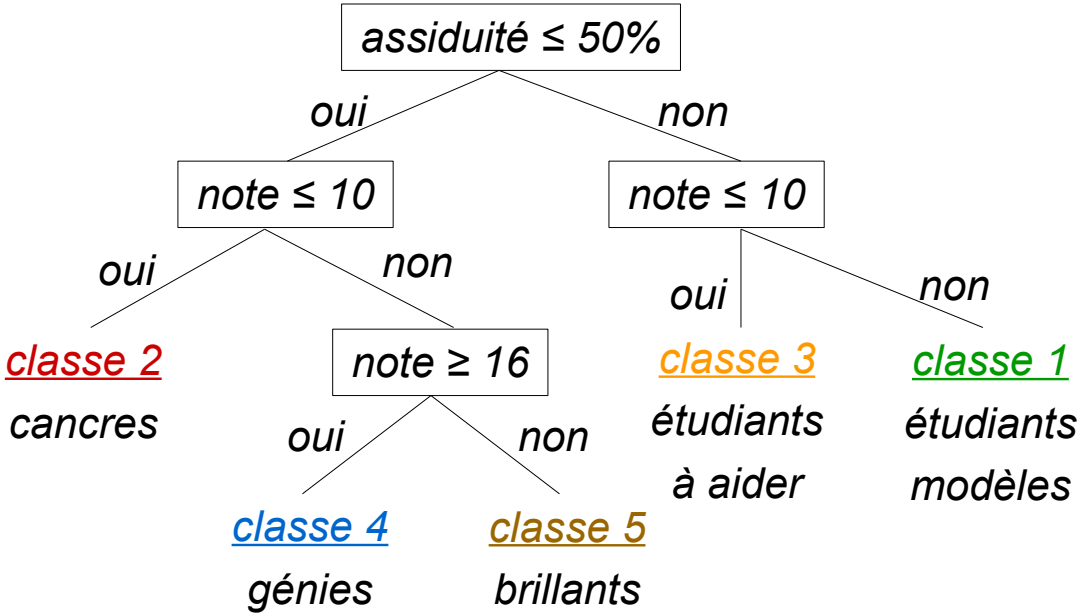
3- déterminer les paramètres généraux d'apprentissage
- renforcement : trouver une fonction d'évaluation
- fixer les critères de réussite et de performance (taux de validation, ...)

4- faire apprendre
- ça peut être long ...

5- tester et valider
- souvent, repartir en 3 ou en 1

L'apprentissage peut être réalisé une fois, à l'initialisation du programme (offline learning) ou bien de façon continue (online learning)

Un arbre de décision (Morgan & Sonquist, 1963) est un arbre binaire dont chaque noeud porte un test sur une valeur.



Il peut servir à la classification mais aussi à la régression (les feuilles portent alors une valeur numérique réelle)

Pour chaque noeud il faut choisir le test qui produit la « meilleure » découpe des données : attribut à tester et seuil de test.

Mesures de la qualité de la découpe : indice de Gini, entropie de Shannon, ...

Indice de Gini :i le test sépare les données D en deux parties c_1 et c_2, la valeur de l'indice est $1-p(c_1)^2-p(c_2)^2$ avec p(c) la proportion de c dans D

Une mesure doit être maximale si elle coupe les données en 2 parts égales (*p(c)=0.5*), minimale si une des parts est vide (*p(c)=0*).

Une mesure est d'autant meilleure qu'elle produit un arbre dont les feuilles sont hétérogènes (classes non « pures »).

Exemple d'arbre de décision efficace : le jeu Akinator qui devine à quoi vous pensez

Premier indicateur : taux de bonnes réponses sur les données exemples

- il n'est pas forcément de 100%
- les données exemples peuvent être biaisées (pas représentatives)

Deuxième indicateur : taux de bonnes réponses sur des données test

- souvent 10% des données initiales sont réservées à la validation, 90% conservées pour l'apprentissage

Validation croisée : on découpe les données en n parties P_1, ..., P_n

- on apprend sur toutes les parties sauf P_1, on teste sur P_1
- ...
- on apprend sur toutes les parties sauf P_n, on teste sur P_n
- le taux de réussite est la moyenne quadratique de toutes les erreurs

faux positifs : cas prédits comme positif, mais qui ne le sont pas

Exemple : le modèle indique qu'un cas relève d'une classe, mais ce n'est pas le cas

faux négatifs : cas prédits comme négatif, mais qui ne le sont pas

Exemple : le modèle indique qu'un cas ne relève pas d'une classe, alors que c'est le cas

On parle aussi de vrai positif et de vrai négatif.

Matrice de confusion :

réalité \ prédiction	positif	négatif
positif	VP	FN
négatif	FP	VN

Rappel (ou sensibilité) : probabilité d'une prédiction positive dans un cas positif

Rappel = VP/(VP+FN)

Spécificité : probabilité d'une prédiction négative dans un cas négatif

Spécificité = VN/(VN+FP)

Précision : probabilité d'un cas positif si la prédiction est positive

Précision = VP/(VP+FP)

Exactitude : taux de bonnes prédictions

Exactitude = (VP+VN)/(VP+FP+VN+FN)

Pour l'apprentissage de plusieurs classes, on généralise :

- rappel = somme des rappels de chaque classe/nombre de classes
- précision = somme des précision de chaque classe/nombre de classes

Exemple : 50 cas positifs, 50 cas négatifs

réalité \ prédiction	positif	négatif
positif	42	8
négatif	25	25

Rappel = 42/(42+8) = 84%

Spécificité = 25/(25+25) = 50%

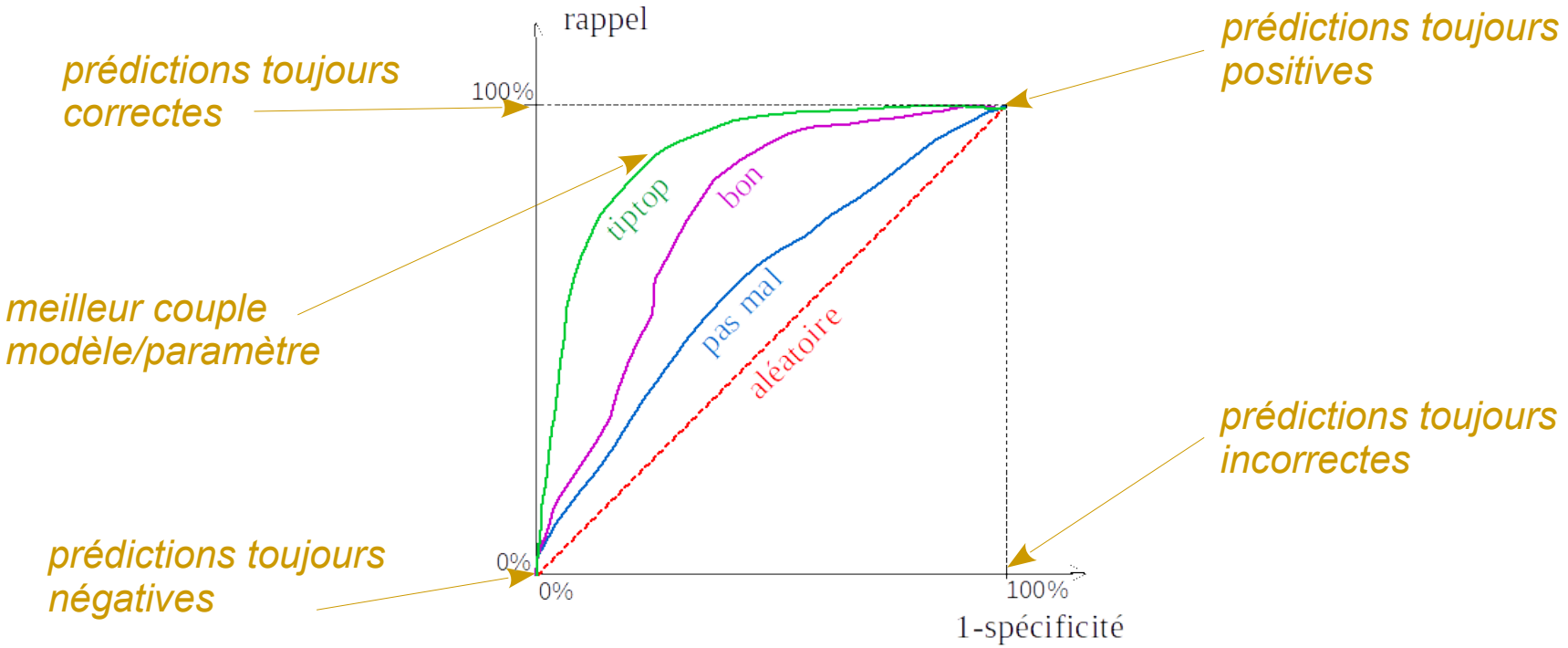
Précision = 42/(42+25) = 62.7%

Exactitude = (42+25)/(42+25+8+25) = 33.5%

Un bon rappel seul n'est pas signe de qualité : il suffit que le modèle renvoie systématiquement un résultat positif. Un bon modèle doit être à la fois sensible et spécifique.

Un modèle peut être meilleur qu'un autre en précision, mais pas en rappel, ou l'inverse.

La courbe ROC (Receiver Operating Characteristic) permet d'analyser l'influence d'un paramètre du modèle : on fait varier le paramètre et on trace la courbe du rappel en fonction de 1-spécificité (taux de vrais positifs en fonction du taux de faux positifs).



Un modèle donnant des résultats aléatoires suit la bissectrice, plus la courbe s'en écarte, meilleur est le modèle. La valeur idéale du paramètre correspond au point le plus proche du coin en haut à gauche.

La qualité d'une régression s'évalue à partir d'une fonction de coût qu'il faut minimiser

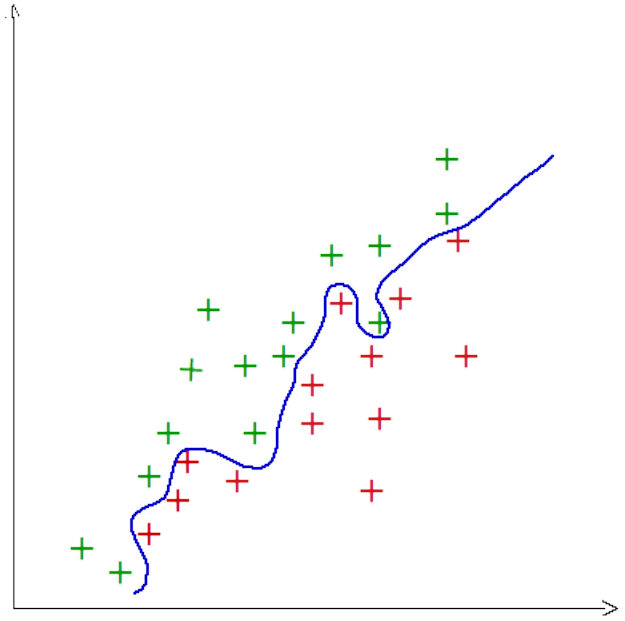
p_i : les valeurs réelles (des exemples) à prédire

q_i : les valeurs prédites

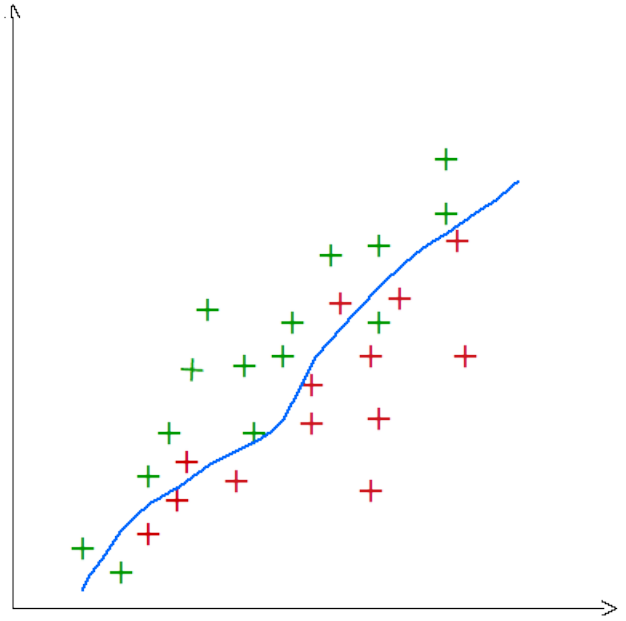
Les moindres carrés (la plus utilisée) : $\sum_i (p_i - q_i)^2$

L'entropie croisée : $\sum_i (p_i \times \log(q_i))^2$

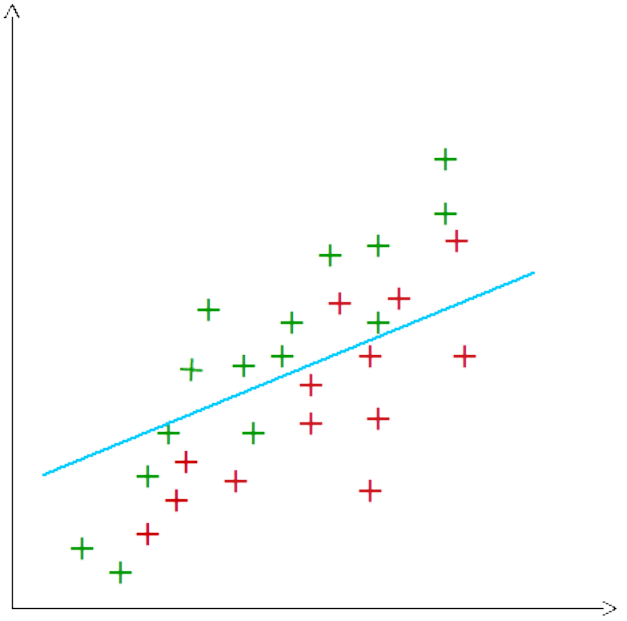
Le sur-apprentissage (overfitting) a lieu lorsque le modèle est trop conforme aux données d'apprentissage et donc potentiellement incorrect sur de nouvelles données.



sur-apprentissage
variance forte
biais (erreur) faible



apprentissage



sous-apprentissage
variance faible
biais (erreur) fort

Il faut trouver un compromis biais/variance.

Augmenter la taille des données d'apprentissage limite les risques de sur-apprentissage, diminuer la complexité du modèle aussi.

De nombreux modèles sont possibles :

- arbres de décisions, forêts aléatoires
- machines à vecteur de support
- modèles bayésiens
- réseaux de neurones formels
- ...

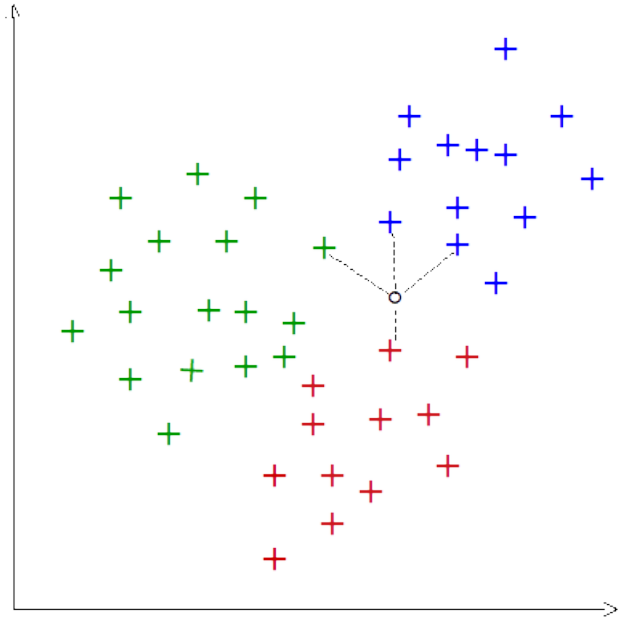
Le choix du modèle dépend du type de données à traiter, de l'objectif opérationnel, des goûts de l'informaticien, ...

Le choix d'un modèle ne conditionne pas forcément le choix de la méthode d'apprentissage.

Exemple : on peut utiliser un réseau de neurones mais réaliser l'apprentissage par descente de gradient ou par algorithme génétique.

Méthode des K plus proches voisins (KNN) (Fix & Hodges, 1951) :

- on stocke les exemples entrée-sortie d'apprentissage
- pour une nouvelle entrée, on détermine les k plus proches entrées les plus proches du nouveau cas selon une distance prédéfinie
- la sortie du nouveau cas est la moyenne des sorties des k voisins



Cette méthode se généralise dans le raisonnement à partir de cas (RAPC).

Forêts aléatoires (Breiman & Cutler, 2001) :

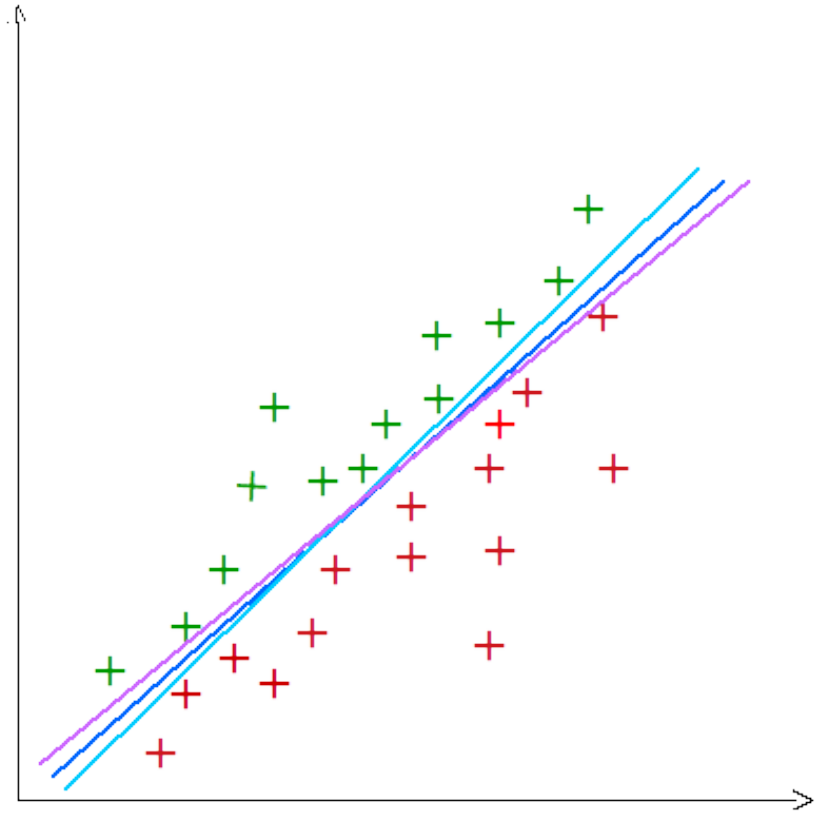
- on construit plusieurs arbres de décisions sur des sous-parties des données d'apprentissage sélectionnées aléatoirement et sur un sous-ensemble des attributs
- la sortie pour une nouvelle entrée est une moyenne des sorties obtenues sur chaque arbre

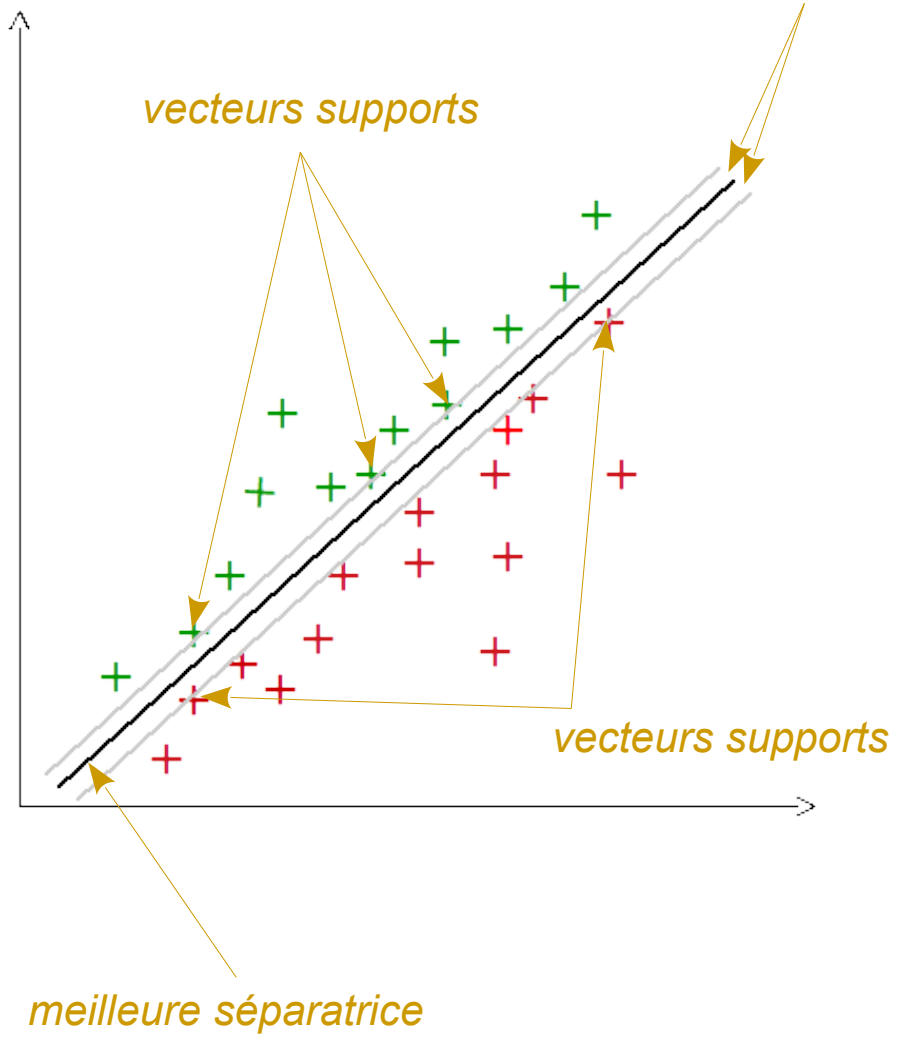
Avantages par rapport à un modèle à un seul arbre :

- apprentissage plus rapide (plusieurs petits arbres au lieu d'un gros)
- limite le risque de sur-apprentissage
- limite les biais introduits par certains attributs

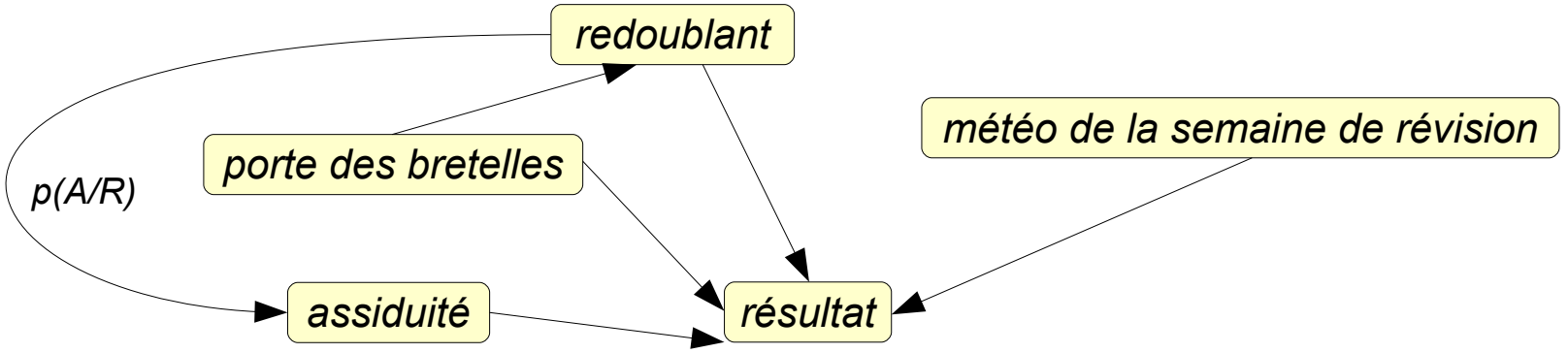
Inconvénient : on perd la lisibilité du modèle (problème de la boite noire)

Machine à vecteurs supports (SVM) (Vapnik, 1963) : des résultats théoriques montrent que la meilleure séparatrice (qui maximise la capacité de discrimination du modèle) est celle qui maximise les marges avec les données.





Les réseaux bayésiens sont des graphes modélisant des relations causales probabilistes entre les variables explicatives, selon le schéma du théorème de Bayes : p(A/B) = p(A)*p(B/A)/p(B)



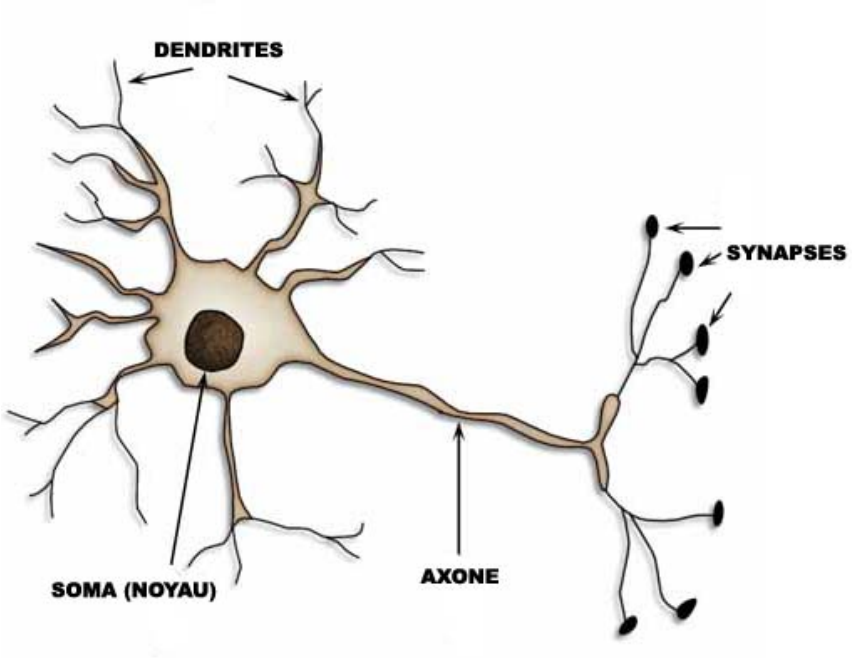
p(A/R)	faible	moyenne	correcte	forte
oui	0.8	0.15	0.04	0.01
non	0.5	0.35	0.1	0.05

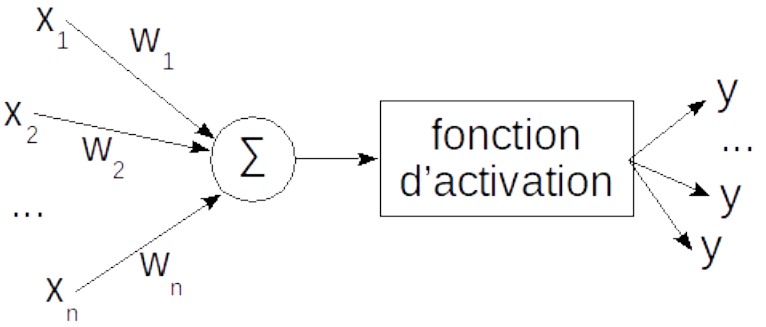
Les variables sont a priori discrètes (mais pas forcément).

Le réseau et l'algorithme d'apprentissage peut être plus on moins complexes selon la nature des données, que les variables sont indépendantes ou pas, etc.

Un neurone formel (McCulloch & Pitts, 1943) modélise un neurone biologique.

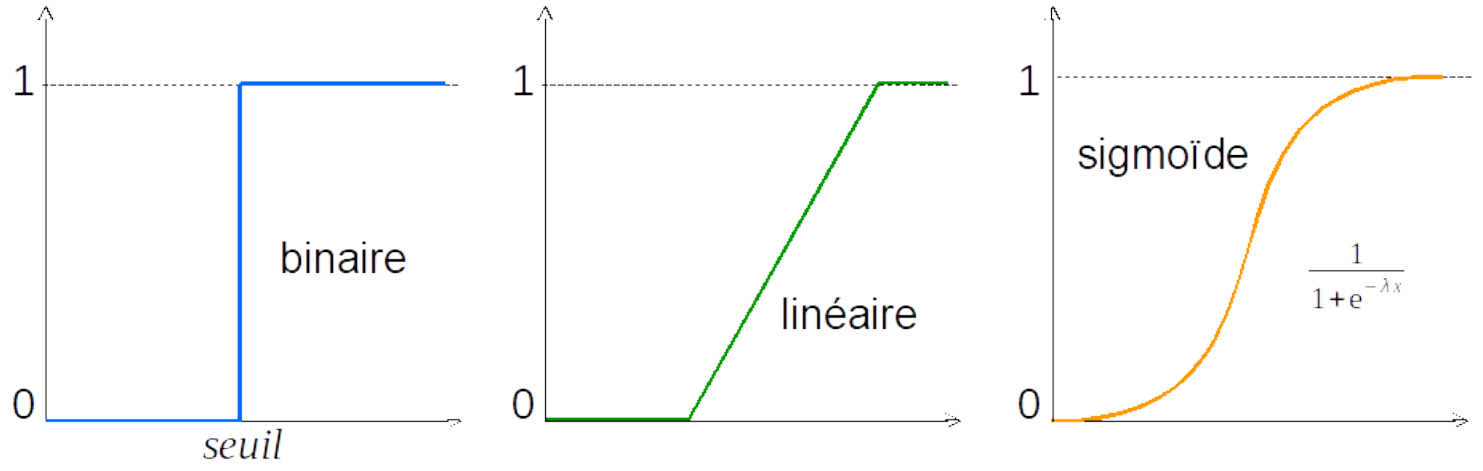
Les entrées sont multipliées par des poids synaptiques, puis sommées, avant qu'une fonction d'activation calcule la sortie.



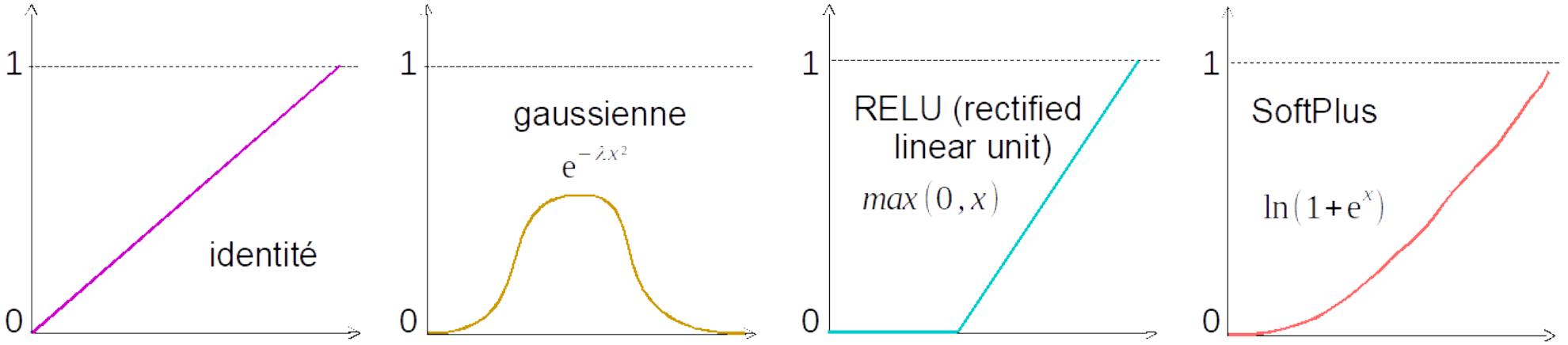


Les sorties sont généralement normalisées entre 0 et 1. Les poids peuvent être négatifs (inhibition).

Les fonctions d'activation modélisent généralement le comportement des neurones biologiques :

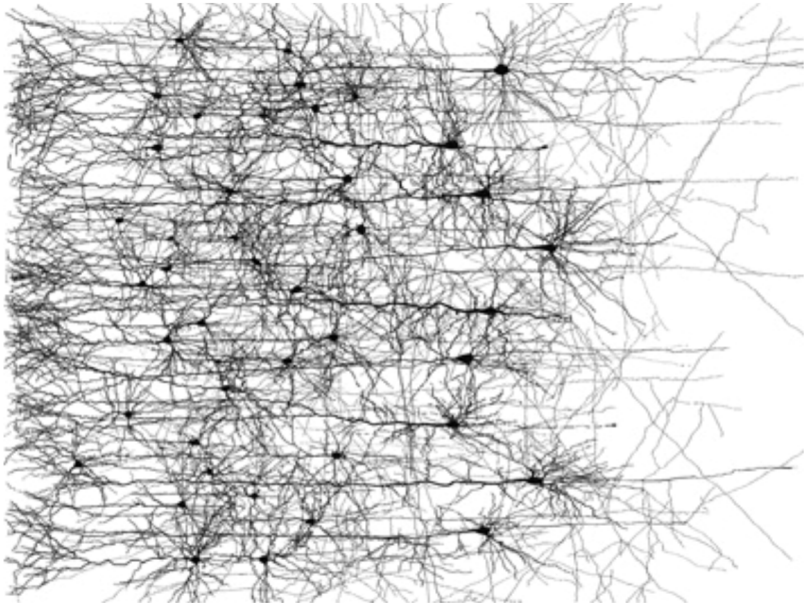


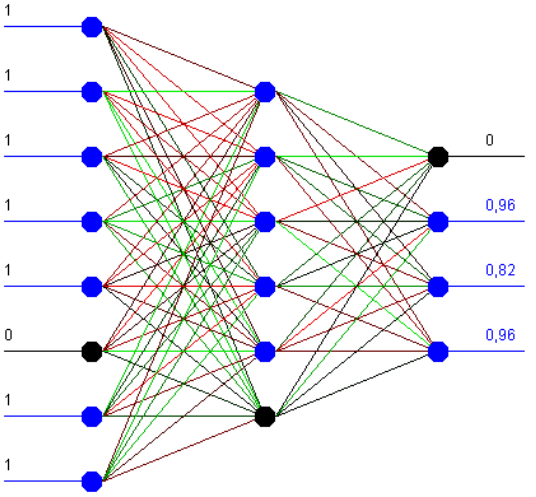
On utilise de plus en plus des fonctions qui ne sont plus conformes au fonctionnement des neurones biologiques :



Un réseau de neurones formels est un ensemble de neurones organisés en couches.

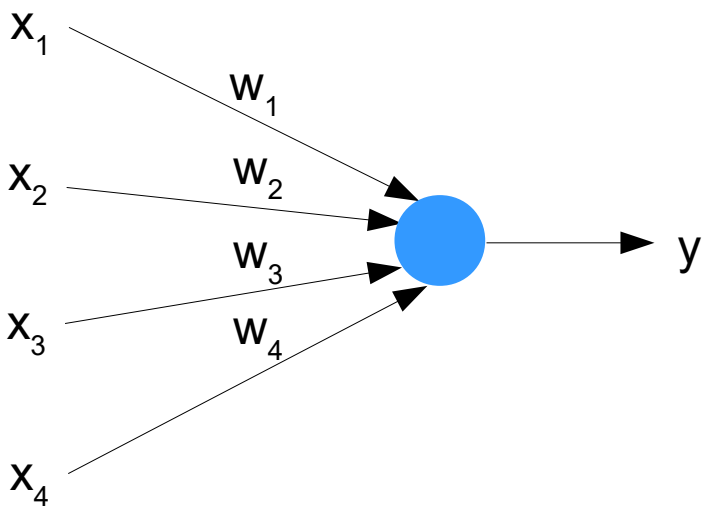
Chaque sortie d'un neurone d'une couche est dirigée sur tous les neurones de la couche suivante.





L'apprentissage dans un réseau consiste à modifier les poids synaptiques pour diminuer l'erreur en sortie du réseau (Hebb, 1949).

Le Perceptron (Rosenblatt, 1957) est un réseau monocouche. Historiquement, il ne possède qu'un seul neurone à seuil.



$$\begin{cases} y = 1 \text{ si } \sum_i x_i w_i > \theta \\ y = 0 \text{ sinon} \end{cases}$$

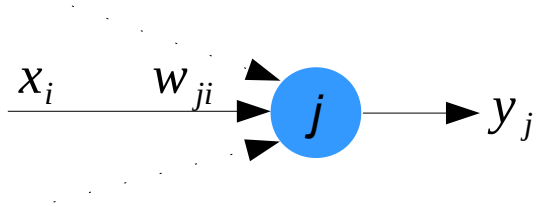
Exemple : un perceptron qui calcule la fonction OU (inclusif) avec les poids W = {1,1,1,1} et un seuil θ = 0

On veut faire apprendre à un perceptron à *n* entrées et *m* sorties l'exemple :

$$X^0=\{x_1^0,\ldots,x_n^0\},Y^0=\{y_1^0,\ldots,y_m^0\}$$

w_{ji} le poids de l'entrée *i* du neurone *j*

y_j la sortie calculée du neurone *j*



Règle de Hebb (1949) : plus un neurone décharge sur le suivant, plus le neurone suivant est facilement activable

$$\Delta w_{ji}=w_{ji}^{t+1}-w_{ji}^{t}=\alpha x_i$$

α est le taux d'apprentissage (positif).

Amélioration de Rosenblatt : on tient compte de l'erreur en sortie

$$\boxed{\Delta w_{ji}=w_{ji}^{t+1}-w_{ji}^{t}=\alpha\left(y_j^0-y_j\right)x_i}$$

Pour les perceptrons à fonction binaire, la convergence est garantie si et seulement si les exemples d'apprentissage sont linéairement séparables (Minsky & Papert, 1969).

Table du AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

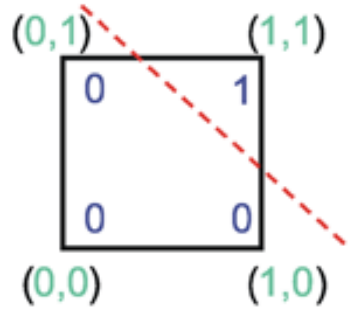
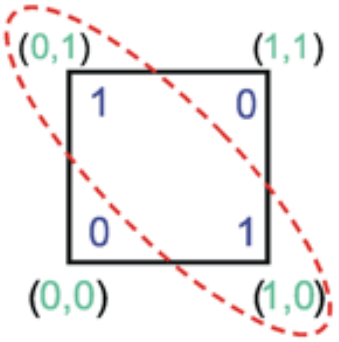


Table du XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Les perceptrons à fonctions binaires sont des classifieurs linéaires

On veut faire apprendre à un réseau à *n* entrées et *m* sorties l'exemple :

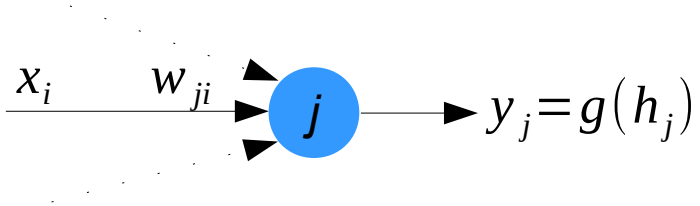
$$X^0=\{x_1^0,\ldots,x_n^0\}, Y^0=\{y_1^0,\ldots,y_m^0\}$$

w_{ji} le poids de l'entrée *i* du neurone *j*

y_j la sortie calculée du neurone *j*

g la fonction d'activation des neurones

$$h_j=\sum_k x_k w_{jk}$$

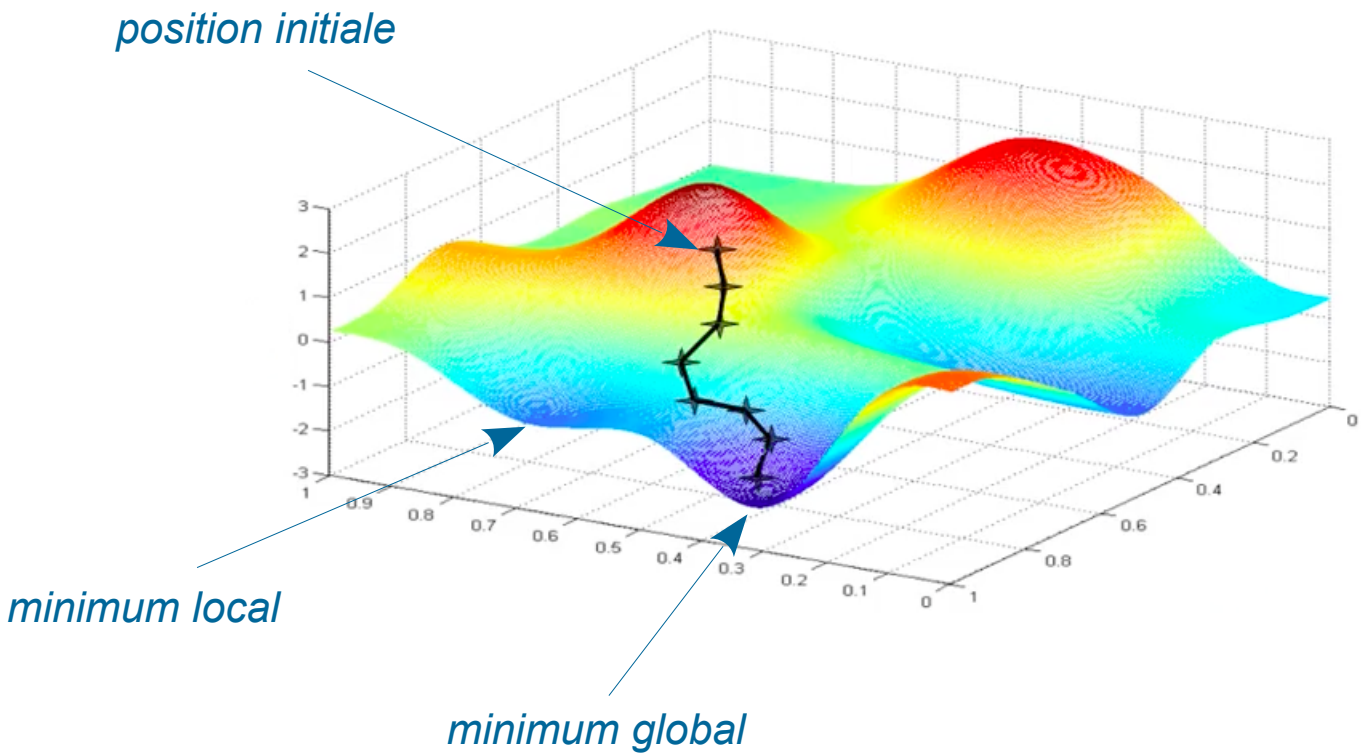


On calcule la sortie *Y* en fonction de X^0 et on modifie les poids pour réduire l'erreur quadratique *E* entre la sortie calculée et la sortie attendue

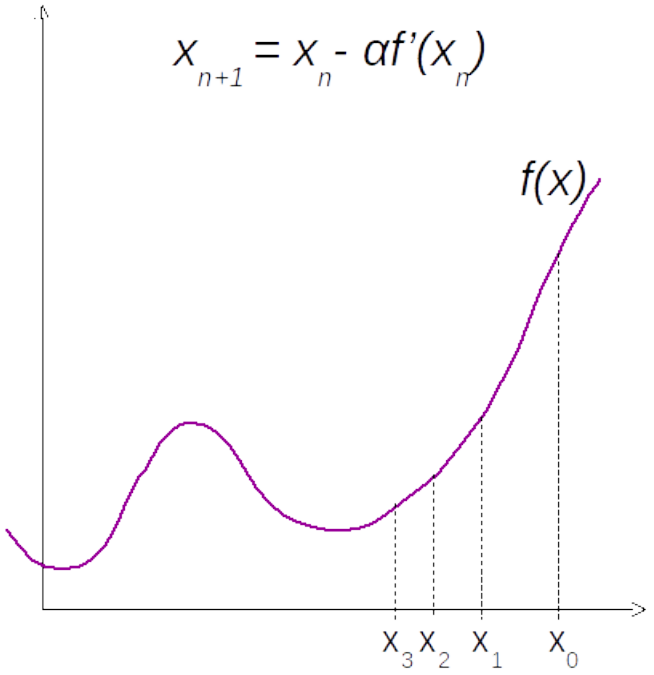
$$E=\frac{1}{2}\sum_j (y_j^0-y_j)^2$$

L'erreur en fonction des poids forme une surface dans un espace de grande dimension (le nombre de poids + 1).

On veut se déplacer vers un endroit où l'erreur est (la) plus faible.



Pour trouver un minimum d'une fonction à partir d'un point, on calcule une suite de points en suivant la pente descendante :



La fonction doit être dérivable

α est le pas de descente (taux d'apprentissage)

Un pas faible ralentit la convergence

Un pas élevé peut empêcher la convergence

La convergence n'est pas garantie et dépend fortement du point initial

On peut tomber dans un minimum local

Calcul de $\Delta w_{ji} = w_{ji}^{t} - w_{ji}^{t-1}$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial\left(\frac{1}{2}(y_j^0 - y_j)^2\right)}{\partial w_{ji}} = \frac{\partial\left(\frac{1}{2}(y_j^0 - y_j)^2\right)}{\partial y_j}\frac{\partial y_j}{\partial w_{ji}} = -(y_j^0 - y_j)\frac{\partial y_j}{\partial w_{ji}}$$

$$\frac{\partial E}{\partial w_{ji}} = -(y_j^0 - y_j)\frac{\partial y_j}{\partial h_j}\frac{\partial h_j}{\partial w_{ji}} = -(y_j^0 - y_j)g'(h_j)\frac{\partial\left(\sum_k x_k w_{jk}\right)}{\partial w_{ji}} = -(y_j^0 - y_j)g'(h_j)x_i$$

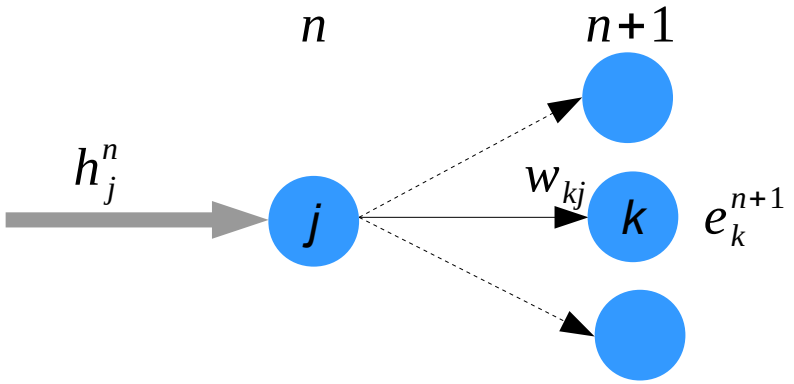
$$\boxed{\Delta w_{ji} = \alpha(y_j^0 - y_j)g'(h_j)x_i}$$ pour la couche de sortie avec α le taux d'apprentissage

Le gradient est le vecteur des dérivées partielles : $\nabla E = \left[\frac{\partial E}{\partial W_{1i}}, \frac{\partial E}{\partial W_{2i}}, \ldots\right]$

Dans un réseau multicouche, il faut également mettre à jour les poids des couches intermédiaires (couches cachées).

L'erreur sur la couche *n* est calculée à partir de l'erreur sur la couche *n+1* en rétropropageant le gradient (Werbos, 1982).

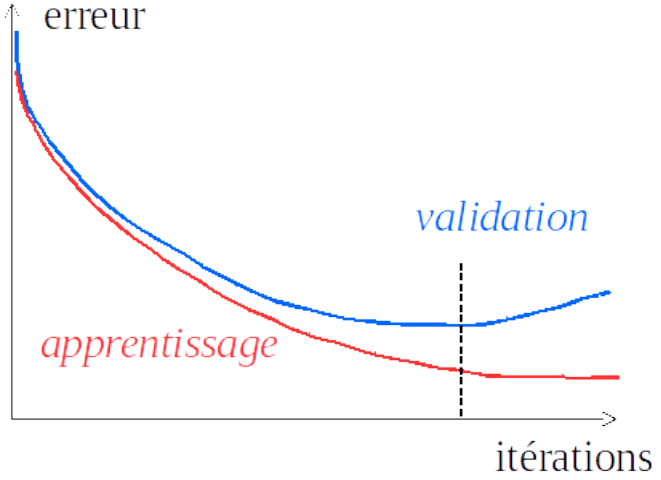
$$e_j^n = g'^n(h_j^n) \sum_k w_{kj} e_k^{n+1}$$

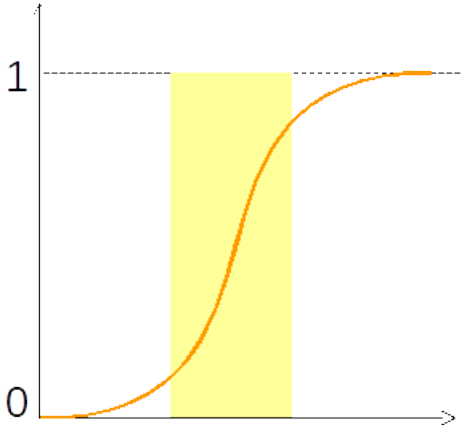


Le calcul des poids est généralisé pour chaque couche : $\Delta w_{ji}^n = \alpha e_j^n x_i^{n-1}$

On réitère la rétropropagation tant que l'erreur en sortie est supérieure à un seuil arbitraire.

Il faut parfois mieux accepter une erreur d'apprentissage plus élevée pour limiter l'erreur en validation (sur-apprentissage).





Les valeurs initiales des poids doivent être faibles (dans [-0.01,0.01]) pour se positionner sur la zone la plus pentue (rapidité d'apprentissage).

Nécessité de sortir des minimum locaux : réinitialisation des poids, …

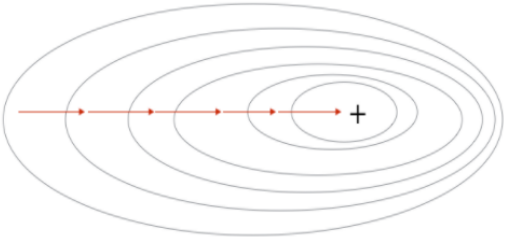
Le taux d'apprentissage peut varier (élevé au départ pour la rapidité, faible à la fin pour la précision).

Descente de gradient en mode batch (hors ligne) :

1- on calcule les erreurs et les gradients sur tous les exemples

2- on modifie les poids

3- on repart en 1 jusqu'à diminuer suffisamment l'erreur en sortie



▲ convergence garantie si les données le permettent
▲ peu de mises à jour des poids, donc moins de calculs
▲ plus de stabilité dans l'apprentissage

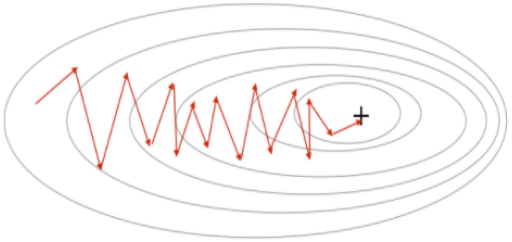
▼ souvent impraticable sur des données de grande taille
▼ risque important de tomber dans des minimums locaux

Descente de gradient stochastique (inline) :

1- on choisit un exemple

2- on calcule l'erreur pour cet exemple, puis le gradient et on modifie les poids

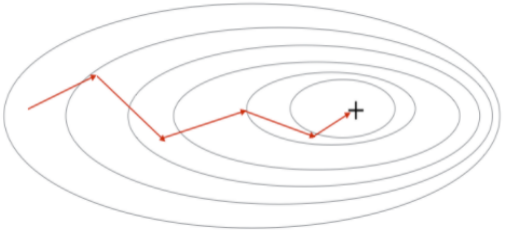
3- on recommence en 1 jusqu'à diminuer suffisamment l'erreur en sortie



▲ modifications fréquentes des poids, apprentissage a priori plus rapide
▲ moins de risque de tomber dans des minimums locaux

▼ convergence non garantie
▼ beaucoup de calculs pour mettre souvent les poids à jour

Descente de gradient par mini-lots (mini-batch) : mode inline sur des groupes d'exemples



Les réseaux sans bouclage (feed-forward) sont des approximateurs universels et parcimonieux :

- un réseau à une couche cachée de neurones sigmoïdes peut approximer toute fonction continue et bornée avec une précision arbitraire (Cybenko, 1989)
- à nombre égal de paramètres, un tel réseau approxime plus de fonctions qu'un polynôme (Sussman, 1992)

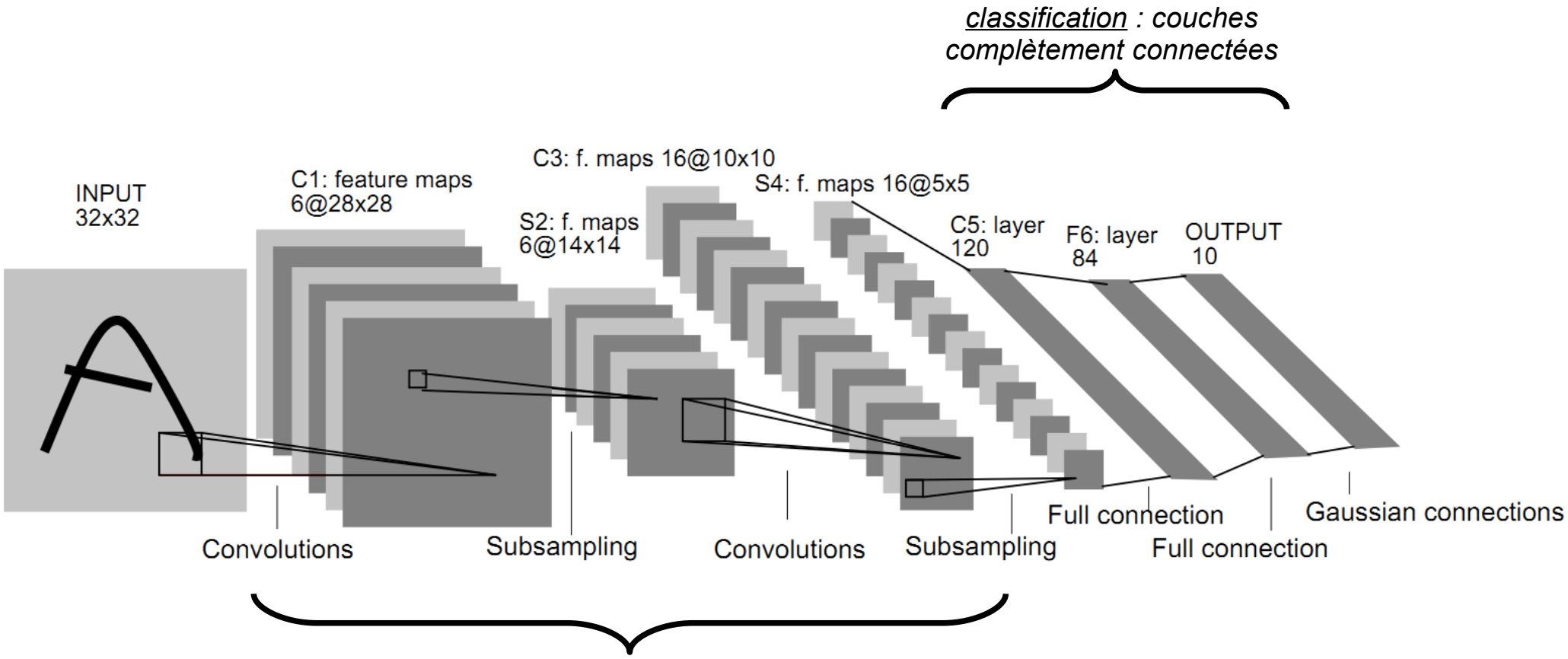
► combien de neurones cachés sont nécessaires ?

► un réseau modulaire parait plus intéressant qu'un gros pâté de neurones

Autour des années 2000 apparait l'apprentissage profond (deep learning) :

- augmenter le nombre de couches pour baisser le nombre de neurones
- spécialisation des couches
- ne pas tout connecter
- pré-apprentissage de chaque couche
- réutilisation des couches (transfer learning)
- nouvelles méthodes d'apprentissage
- ...

Les réseaux de neurones convolutifs (CNN) sont inventés pour l'analyse d'image (LeCun, 1989). Ils sont maintenant appliqués à l'analyse de texte, de la parole, ...



Alternance de couches :

- convolution (couches partageant les mêmes poids et qui se chevauchent)

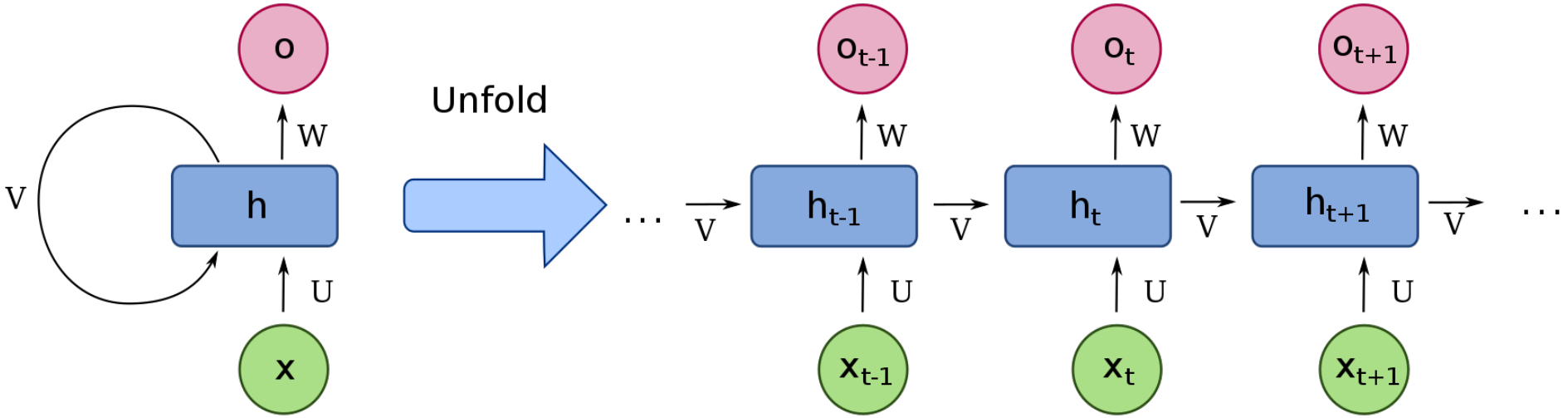
- max pooling (réduction du nombre de neurones par extraction du maximum)

- correction RELU (pour mettre les valeurs négatives à 0)

Les réseaux récurrents (RNN) servent à apprendre des données organisées en séquences : mots dans une phrase, images dans une vidéo, …

La classification d'une entrée dépend de celle des entrées précédentes.

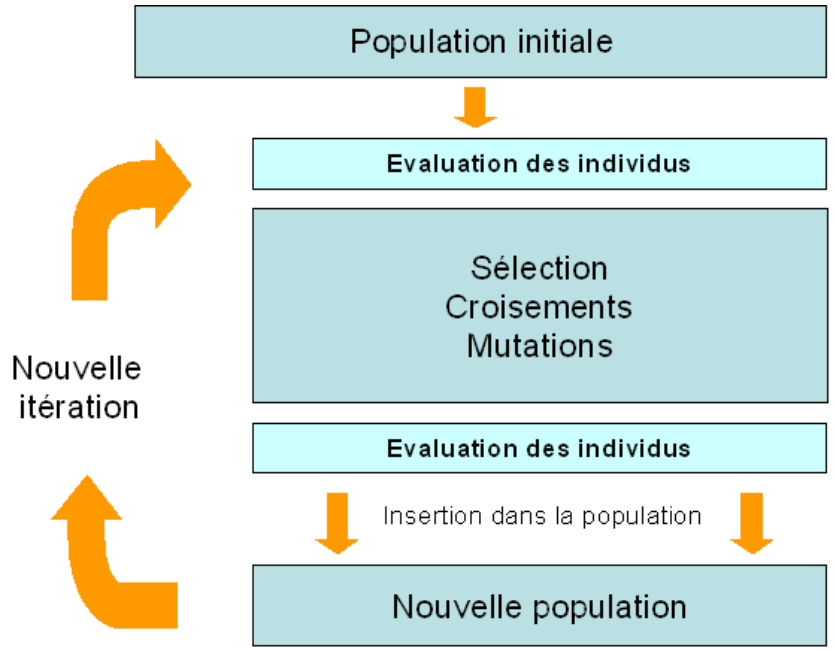
L'apprentissage se fait par rétropropagation du gradient une fois le réseau déplié.



Source de l'image : Wikipédia

L'apprentissage dans les ANN peut se faire par algorithme génétique.

La structure même du réseau peut être apprise, pas seulement les poids.



- les modèles obtenus par apprentissage sont souvent d'une faible qualité
- l'apprentissage prend souvent énormément de temps
- un apprentissage de qualité demande beaucoup de données de qualité

Exemples où l'apprentissage artificiel n'est pas adapté :

- une régression linéaire entre deux variables est simple à réaliser avec des techniques statistiques conventionnelles

- on peut faire produire par apprentissage du code pour réaliser des opérations de base de l'arithmétique, mais le code produit est très complexe et ne donne pas toujours le bon résultat

Exemples où l'apprentissage artificiel est adapté voire indispensable :

- fouille de données massives (data mining)

- reconnaissance de la parole

- analyse d'image (reconnaissance de l'écriture manuscrite, de visage, diagnostic médical, ...)

- analyse prédictive (économie, domaine juridique, météorologie, panne de machine, ...)

L'apprentissage artificiel est en plein essor, de nombreuses bibliothèques open-source, orientées GPU, sont développées :

- TensorFlow (Google)
- CNTK (Microsoft)
- Torch (issu de Facebook)
- Theano (université de Montréal)
- MXNet (Apache)
- Caffe (University of Berkeley)

Il existe aussi des front-end pour ces API :

- Keras (fonctionne avec Theano, TensorFlow, CNTK)
- pyTorch (fonctionne avec Torch)

Ces outils optimisent l'utilisation des GPU

Les API spécialisées sont de plus ne plus nombreuses (reconnaissance de la parole, ...)