Chapitre 1 : *Introduction à l'Intelligence Artificielle*

I. Définitions et caractérisations de l'IA : plusieurs définitions de l'IA existent :

1. Définitions reposant sur le concept d'Intelligence :

L'IA est la partie de l'informatique consacrée à la conception de systèmes informatiques intelligents. **E. Feigenbaum**.

L'IA est l'étude des concepts qui permettent de rendre les machines intelligentes. **Winston**.

L'IA est la partie de l'informatique consacrée à l'automatisation de comportements intelligents. **Lugger & Stubbleeld (1993)**.

2. Définition liée à la compréhension des facultés humaines

L'étude des facultés mentales à travers l'utilisation de modèles informatiques. **Charmiak & McDermott (1985)**.

3. Définitions prenant l'humain comme référence

L'IA est la science de programmer les ordinateurs pour qu'ils réalisent des tâches qui nécessitent de l'intelligence lorsqu'elles sont réalisées par des êtres humains. **Marvin Minsky**.

Des systèmes qui démontrent des capacités comparables au raisonnement humain pour améliorer la qualité de vie et améliorer la compétitivité économique. **Japan-Singapore AI Centre**.

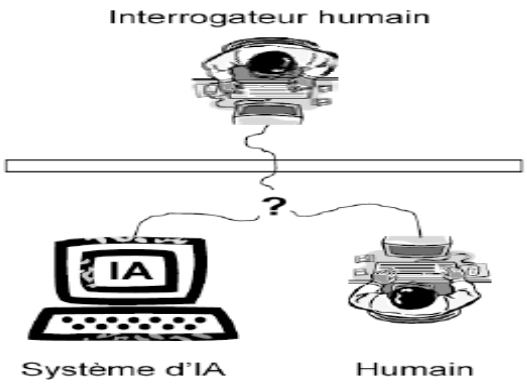
4. Le test de Turing

Article de 1950 : Computing Machinery and Intelligence. Idée : ne pas définir l'IA mais la tester.

Protocole : un individu communique à l'aide d'un terminal d'ordinateur avec un interlocuteur invisible. Il doit décider si l'interlocuteur est un être humain ou un système d'IA imitant un être humain.

Construire un cerveau artificiel en imitant le comportement humain, les "états de pensée" étant équivalents aux instructions de la machine.

vision informatique : imiter le comportement, pas le fonctionnement.



5. Définitions liées à la difficulté des problèmes visés

- L'automatisation d'activités que nous associons à la pensée humaine, comme la prise de décision, la résolution de problème ou l'apprentissage. **Bellman (1978)**.
- L'étude de comment programmer les ordinateurs pour qu'ils réalisent des tâches pour lesquelles les êtres humains sont actuellement meilleurs. **Rich & Knight (1991)**.
- L'IA commence là où l'informatique classique s'arrête : tout problème pour lequel il n'existe pas d'algorithme connu ou raisonnable permettant de le résoudre relève a priori de l'IA. **Jean-Louis Laurière**.

On peut résumer ces définitions en 2 :

➢ **Définition 1 de l'Intelligence Artificielle**

• L'IA est une branche de l'informatique
– Un système d'IA **agit** comme un humain
– Il possède un comportement intelligent, il peut :
  • Communiquer
  • Enregistrer et exploiter les informations
  •Utiliser les informations pour répondre à des questions et tirer des conclusions

➢ **Définition 2 de l'Intelligence Artificielle**

• L'IA est une science cognitive
– Un système d'IA **raisonne** comme un humain
– Comprendre comment voir, apprendre, mémoriser, raisonner
– Modéliser les connaissances et le raisonnement
– Fournir des modèles et en tester le comportement par des techniques de psychologie cognitive, en le comparant au comportement humain

II. Historique de l'IA :

- Naissance de l'IA (1943-1955)
- 1952 à 1969 : période euphorique, on annonce
  - le remplacement des experts humains par des systèmes experts,
  - compréhension du langage naturel
- 1966 à 1974 : on comprend la difficulté de la tâche
  - le remplacement des experts humains par des systèmes experts : échec, raisonnement ok, manque la connaissance, goulot d'étranglement de l'IA
  - compréhension du langage naturel : échec, trop difficile, impossible ?
- 1988 à 1993, hiver de l'IA : ralentissement des recherches en IA à cause des promesses non tenues
- Les années 90 à présent : l'avènement d'Internet a donné naissance à plusieurs disciplines :

➢ Les moteurs de recherche integrent des concepts d'IA tels que :
➢ Data maining
➢ Ontologie
➢ E-learning
➢ Réalité virtuelle
➢

III. Les domaines de l'intelligence artificielle

1. Les **systèmes experts** : un système expert est un logiciel capable de simuler le comportement d'un expert humain effectuant une tâche précise. Il s'agit là d'un domaine où le succès de l'intelligence artificielle est incontestable et cela est sans doute dû au caractère très ciblé de l'activité que l'on demande de simuler. Le logiciel MYCIN (1974) par exemple est capable de fournir un très bon diagnostic dans la mesure où il est spécialisé dans certains types de leucémies.
2. Le **calcul formel** (opposé au calcul numérique) traite des expressions symboliques. Par exemple, calculer la valeur d'une fonction réelle en un point est du calcul numérique alors que calculer la dérivée d'une fonction numérique est du calcul formel. Les logiciels actuellement proposés sur le marché (MATHEMATICA, MAPLE, ...) sont capables d'effectuer tous les calculs formels.
3. La **représentation des connaissances** : si l'on veut qu'un logiciel soit capable de manipuler des connaissances, il faut savoir les représenter symboliquement. C'est là un des secteurs les plus importants de la recherche en intelligence artificielle.
4. La **simulation du raisonnement humain**. Les hommes sont capables de raisonner sur des systèmes incomplets, incertains et même contradictoires. On tente de mettre au point des logiques qui formalisent de tels modes de raisonnement (logiques modales, temporelles, floue, non monotones, etc.).
5. Le **traitement du langage naturel**. Qu'il s'agisse de traduire un texte dans une autre langue ou de le résumer, le problème crucial à résoudre est celui de sa compréhension. On commence à trouver des traducteurs automatiques d'autant meilleurs qu'ils traitent d'un domaine spécialisé.
6. La **résolution de problèmes** : représentation, analyse et résolution de problèmes concrets. Les jeux fournissent une bonne illustration de ce domaine : le champion du monde de Backgammon est un programme depuis quelques années déjà et c'est aussi le cas pour le jeu d'échecs. Le jeu de Go résiste beaucoup plus aux efforts des programmeurs de jeux.
7. La **reconnaissance de la parole** : les progrès sont beaucoup plus lents qu'on ne l'imaginait mais constants. On est encore loin de pouvoir produire un logiciel capable de reconnaître les paroles d'un locuteur quelconque et cela essentiellement parce que la compréhension d'un mot, d'une phrase requiert beaucoup d'informations extra-langagières (le contexte, la connaissance du monde dans lequel nous vivons interviennent de manière fondamentale). Un Dictaphone automatique a malgré tout été proposé dans le commerce en 1994 mais il ne fonctionne que si le locuteur sépare chacun des mots et n'effectue aucune liaison.
8. La **reconnaissance de l'écriture** : même la reconnaissance de l'écriture dactylographiée n'est pas un problème facile (bien qu'on commence à trouver sur le marché des logiciels très performants). L'écriture manuscrite pose des problèmes.
9. La **reconnaissance des visages** : longtemps considéré comme un des problèmes les plus difficiles de l'intelligence artificielle, il semble que l'on obtienne des résultats intéressants en utilisant des réseaux neuronaux.
10. La **robotique**. Il y a déjà longtemps que des robots industriels ont fait leur apparition dans les usines. On appelle robot de la première génération, ceux qui sont capables d'exécuter une série de mouvements préenregistrés. Un robot de la deuxième génération est doté de moyens de perception visuelle lui permettant de prendre certaines décisions. Un robot de la troisième génération, objet des recherches actuelles, doit acquérir une plus grande autonomie comme se déplacer dans un environnement inconnu.

11. **L'apprentissage**. On a compris très tôt qu'un logiciel devrait avoir des capacités d'apprentissage autonome pour pouvoir être véritablement qualifié d'intelligent.
12. **Les réseaux neuronaux**. Un réseau de neurones formels est un modèle rudimentaire du cerveau humain, chaque cellule neuronale étant décrite comme une fonction à seuil possédant une sortie et dont les entrées sont reliées à d'autres neurones. Il est pourtant possible d'effectuer des tâches à l'aide de tels réseaux (la reconnaissance des formes et en particulier des visages en étant l'exemple le plus frappant).
13. ***Agents intelligents***
14. ***Data mining***
15. ***Réalité virtuelle***

L'intelligence Artificielle s'intéresse principalement à la résolution de problèmes, généralement complexes, à l'aide de connaissances générales sur le domaine considéré. Deux questions importantes se retrouvent au cœur de cette activité : Comment formaliser ces connaissances générales ? Comment concevoir des systèmes capables de les exploiter automatiquement pour résoudre les problèmes qui nous intéressent.

IV. Caractéristiques de l'IA :

Les principales composantes d'un système d'IA doivent être :

- Les bases de connaissances
- Le raisonnement : les mécanismes d'exploitation des connaissances

Les connaissances : différentes formes du savoir
• Objets du monde, propriétés de ces objets
• Classifications
• Règles heuristiques de savoir-faire
• Connaissances de bon sens
• Métaconnaissances

Pour exploiter des connaissances
• Il faut choisir un formalisme de représentation des connaissances
• Ce formalisme doit permettre de désigner une unité de connaissance par un trait de son contenu
• Il doit permettre au raisonnement de produire de nouvelles connaissances

Quelques types de raisonnement en IA
• Raisonnement formel
• Raisonnement procédural
• Raisonnement par analogie
• Raisonnement par généralisation et abstraction

Difficultés de l'IA :

• Des tâches qui nous paraissent compliquées sont facilement automatisables car conscientes
• Des tâches qui nous paraissent simples sont mal comprises et difficiles à modéliser

Chapitre II :

Introduction à l'apprentissage en I A

1. Définition :

L'apprentissage est un domaine de l'intelligence Artificielle.

L'apprentissage est l'acquisition de nouveaux savoirs ou savoir-faire, c'est-à-dire le processus d'acquisition de connaissances, compétences, attitudes ou valeurs, par l'étude, l'expérience ou l'enseignement

Exemples de tâches :

On veut répondre automatiquement à des questions comme :

- le patient aura-t-il un accident cardio-vasculaire ?
- la molécule que je désire commercialiser est-elle cancérigène ?
- qui est l'auteur de cette page HTML?
- cette phrase est-elle grammaticalement correcte ?
- quelle sera la taille de cet enfant à l'âge adulte ?

Ne pas écrire des programmes qui répondent à ces questions... mais les découvrir automatiquement, par apprentissage (observation d'exemples et de contre-exemples).
En vue de prédire (classer de nouveaux exemples), on ne peut donc pas apprendre par cœur !

2. Catégories d'apprentissage :

- Apprentissage supervisé : est une technique d'apprentissage automatique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'apprentissage contenant des *« exemples »* (en général des cas déjà traités et validés).
- Apprentissage non supervisé : Cette méthode se distingue de l'apprentissage supervisé par le fait qu'il n'y a pas de sortie *a priori*. Dans l'apprentissage non-supervisé il y a en entrée un ensemble de données collectées. Ensuite le programme traite ces données comme des variables aléatoires et construit un modèle de *« densités jointes »* pour cet ensemble de données.
- Apprentissage par renforcement : L'**apprentissage par renforcement** fait référence à une classe de problèmes d'apprentissage automatique, dont le but est d'apprendre, à partir d'expériences, ce qu'il convient de faire en différentes situations, de façon à optimiser une récompense numérique au cours du temps.

  Un paradigme classique pour présenter les problèmes d'apprentissage par renforcement consiste à considérer un agent autonome, plongé au sein d'un environnement, et qui doit prendre des décisions en fonction de son état courant. En retour, l'environnement procure à l'agent une récompense, qui peut être positive ou négative.

L'agent cherche, au travers d'expériences itérées, un comportement décisionnel (appelé *stratégie* ou *politique*, et qui est une fonction associant à l'état courant l'action à exécuter) optimal, en ce sens qu'il maximise la somme des récompenses au cours du temps.

3. Apprentissage à partir d'exemples supervisé :

1. Principe :

1 On dispose d'un (grand) nombre d'exemples.
2 On cherche une structure (des régularités) dans ces exemples, pour obtenir un modèle.
3 On se sert de ce modèle pour :

- classer un nouvel exemple.
- comprendre la structure des exemples.
- Prévoir une valeur numérique pour un nouvel exemple.

2. Structure des exemples

Définition1 :

Un attribut est un champ servant à définir les exemples. Les attributs peuvent être :

→ Discrets.
→ Continus.
→ Textes.

Définition 2 :

Un exemple est défini par une suite de valeurs d'attributs.
Tous les exemples d'un même ensemble ont le même nombre de valeurs d'attributs.
Un exemple peut aussi posséder une classe (discrète ou continue).
F. De

Exemples *d'exemples :*

37 374 LES NOMBRES PAIRES DE 5 CHIFFRES
87438
53738

3. Types de tâches

→ Trouver la classe d'un exemple : **classification**.
→ Trouver la valeur numérique de la classe de l'exemple (ou une valeur pas trop lointaine) : **prédiction/régression.**
→ Regrouper les exemples similaires : **segmentation ou clustering**.

4. ***la classification supervisée*** consiste à inférer à partir d'un échantillon d'exemples classés une procédure de classification. Les systèmes d'apprentissage peuvent être basés :
   - sur des hypothèses probabilistes (classifieur naïf de Bayes, méthodes paramétriques) ;
   - sur des notions de proximité (plus proches voisins, noyaux de Parzen) ;
   - sur des recherches dans des espaces d'hypothèses (arbres de décision, réseaux de neurones).

Chapitre III

Arbres de décisions

1. Définition :

Un **arbre de décision** permet de classer un objet à l'aide de **questions** : chaque nœud de l'arbre représente une question, chaque lien est une réponse à la question, et chaque feuille est une classe.

Un arbre de décision est un arbre orienté dont :

- Les nœuds internes sont étiquetés par un test applicable à tout individu, généralement sur un attribut de description.
- Les arcs contiennent les résultats du test.
- Les feuilles sont étiquetées par une classe.

- Une feuille est repérable par sa position : la liste (unique) des valeurs des arcs qui permettent d'y accéder.
- Un arbre de décision est donc un classifieur organisé de manière arborescente.
- Ce classifieur a une traduction immédiate en terme de règles de décision, mutuellement exclusives et ordonnées (si ... alors ... sinon ...).

Exemple *La population est constituée d'un ensemble de patients. Il y a deux classes : malade et bien portant. Les descriptions sont faites avec les deux attributs : Température qui est un attribut à valeurs décimales et gorge irritée qui est un attribut logique. On considère l'arbre de décision de la figure 1.*

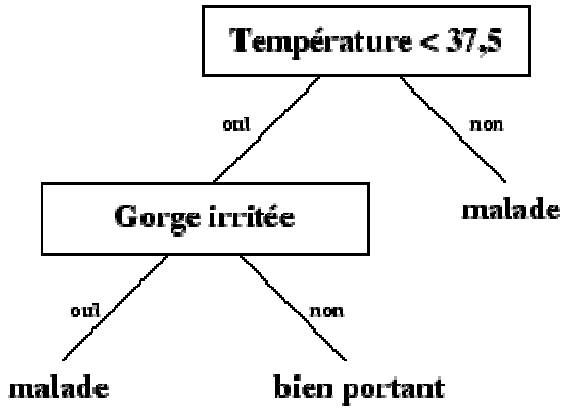


Figure 1

Un *arbre de décision* est un arbre au sens informatique du terme. On rappelle que les nœuds d'un arbre sont repérés par des positions. Si on note le mot vide par ε, les positions pour l'arbre de la figure 1 sont :

- ε étiquetée par le test `Température<37,5`,
- 1 étiquetée par le test `Gorge irritée`,
- 2 étiquetée par la feuille `malade`,
- 21 étiquetée par la feuille `malade`,

- 22 étiquetée par la feuille `bien portant`.
- Les nœuds internes sont appelés *nœuds de décision*. Un nœud de décision est étiqueté par un *test* qui peut être appliqué à toute description d'un individu de la population. En général, chaque test examine la valeur d'un unique attribut de l'espace des descriptions.

- Les réponses possibles au test correspondent aux labels des arcs issus de ce nœud. Dans le cas de nœuds de décision binaires, les labels des arcs sont omis et, par convention, l'arc gauche correspond à une réponse positive au test.

- Les *feuilles* sont étiquetées par une classe appelée *classe par défaut*.

Un arbre de décision est la représentation graphique d'une procédure de classification. En effet, à toute description complète est associée une seule feuille de l'arbre de décision. Cette association est définie en commençant à la racine de l'arbre et en descendant dans l'arbre selon les réponses aux tests qui étiquettent les nœuds internes. La classe associée est alors la classe par défaut associée à la feuille qui correspond à la description. La procédure de classification obtenue a une traduction immédiate en termes de règles de décision. Les systèmes de règles obtenus sont particuliers car l'ordre dans lequel on examine les attributs est fixé et les règles de décision sont mutuellement exclusives.

Exemple 2 : *Soit l'arbre de décision de la figure 1. Un patient ayant une température de 39 et ayant la gorge non irritée sera classé comme malade par cet arbre. La traduction de cet arbre en règles de décision est :*

- *SI* `Température<37,5` *ET* `gorge irritée` *ALORS* `malade`
- *SI* `Température<37,5` *ET NON(*`gorge irritée`*) ALORS* `bien portant`
- *SI NON(*`Température<37,5`*) ALORS* `malade`

Exemple 3 *On considère l'arbre de décision de la figure 1. De plus, on dispose d'un échantillon de 200 patients. On sait que 100 sont malades et 100 sont bien portants, la répartition entre les deux classes M (pour malade) et S (pour bien portant) est donnée par :*

	gorge irritée	*gorge non irritée*
température < 37,5	*(6 S, 37 M)*	*(91 S, 1 M)*
température ≥37,5	*(2 S, 21 M)*	*(1 S, 41 M)*

2. Construction d'un arbre de décision à partir d'un échantillon de données

● **Algorithmes en deux étapes :**

1. Construction d'un petit arbre de décision compatible
2. Elagage de l'arbre

● **Première étape :**

- Idée principale: Diviser récursivement et le plus efficacement possible l'échantillon d'apprentissage par des tests définis à l'aide des attributs jusqu'à obtenir des sous-échantillons ne contenant (presque) que des exemples appartenant à une même classe.

- Méthodes de construction Top-Down, gloutonnes et récursives.

- On a besoin de trois opérateurs permettant de :
  - Décider si un nœud est terminal
  - Si un nœud n'est pas terminal, lui associer un test
  - Si un nœud est terminal, lui affecter une classe

Algorithme d'apprentissage générique :

entrée : langage de description ; échantillon *S*
début
Initialiser à l'arbre vide ; la racine est le nœud courant
répéter
Décider si le nœud courant est terminal
Si le nœud est terminal **alors**
Affecter une classe
sinon
Sélectionner un test et créer le sous-arbre
FinSi
Passer au nœud suivant non exploré s'il en existe
Jusqu'à obtenir un arbre de décision
fin

construire-arbre(X)
SI tous les points de X sont de même classe ALORS
Créer une feuille associée à cette classe
SINON
- choisir (selon critère) le *meilleur couple (attribut; test)* pour créer un nœud
- ce test sépare X en 2 parties Xg et Xd
- construire-arbre(Xg)
- construire-arbre(Xd)

Les trois opérateurs (en général)

1. Un nœud est terminal lorsque :
   - (presque) tous les exemples correspondant à ce nœud sont dans la même classe,
   - il n'y a plus d'attribut non utilisé dans la branche correspondante,
2. On attribue à un nœud terminal la classe majoritaire (en cas de conflit, on peut choisir la classe majoritaire dans l'échantillon, ou en choisir une au hasard),
3. On sélectionne le test qui fait le plus progressé la classification des données d'apprentissage.

On mesure cette progression par deux paramètres

- **l'*indice de Gini*** utilisé dans l'algorithme CART « **Classification And Regression Tree** »

- **la notion d'*entropie*** utilisée dans l'algorithme C4.5 (Entropie de Shannon).

Algorithme de CART :

Indice de Gini

- Soit **S** l'échantillon et **S1, S2, ... Sk** sa partition suivant les classes de l'attribut du test.

$$\mathbf{Gini(S) = \Sigma_i\ |Si|/|S|*(1-|Si|/|S|)}$$

- **Exemple** : supposons **k = 2** et soit **x = |S1|/|S|**. On a **Gini(S) = 2x(1-x)** Cette fonction (idem pour k quelconque) :
  - a des valeurs dans [0,1]
  - S'annule pour x = 0 et x= 1
  - est maximale pour x = ½
  - Plus l'indice de Gini est bas, plus le nœud est pur

Gain et sélection du test

- Soit p la position courante de l'arbre en construction et T un test. On définit :

$$\mathbf{Gainf\ (p,T) = f(Sp) - \Sigma j\ Pj * f(Spj)}$$

Où

→ **Sp** est l'échantillon associé à **p** et

→ **P**j est la proportion des éléments de **Sp** qui satisfont la **j-ème** branche de **T**.

→ **f = Gini**

- Maximiser le gain revient à **minimiser Σj Pj * f(Spj)**
- Gain maximal : l'attribut permet de classer correctement toutes les données
- Gain nul : données sont aussi mal classées après le test qu'avant
- Sélectionner l'attribut dont le gain est maximum correspond à une stratégie gloutonne : rechercher le test faisant le plus progresser la classification.

Processus d'élagage de CART

- Création de l'arbre maximum. Toutes les feuilles des extrémités sont pures
- Élagages successifs de l'arbre
- Retient l'arbre élagué pour lequel le taux d'erreur estimé mesuré sur un échantillon test est le plus bas possible

Soit *t* l'arbre obtenu à la première étape. Pour élaguer, on utilise l'ensemble test *T*. On suppose, en effet, que l'erreur apparente sur *T* est une bonne estimation de l'erreur réelle.

Un élagué de *t* est obtenu en remplaçant un sous-arbre de *t* par une feuille. Nous décrivons maintenant la phase d'élagage qui prend pour entrée l'ensemble d'apprentissage *S*, l'arbre *t* produit et un ensemble test *T*.

a. **construction de la suite des arbres.** On construit une suite $t_{0..},t_i,...,t_p$ tel que t_0 soit l'arbre obtenu à la fin de la phase d'expansion, pour tout *i*, t_{i+1} est un élagué de t_i et le dernier arbre de la suite t_p est réduit à une feuille. Il nous faut définir le procédé de construction de t_{i+1} à partir de t_i. Pour toute position *p* de t_i, on note u_p le sous-arbre de t_i en position *p*. On calcule la quantité

$$g(p) = \frac{\Delta_{app}(p)}{|u_p|-1},$$

où $\Delta_{app}(p)$ est la variation d'erreur apparente mesurée sur l'ensemble d'apprentissage *S* lorsqu'on élague *t* en position *p* et $|u_p|$ est la taille de u_p. On peut remarquer que

$$\Delta_{app}(p) = \frac{MC(p)-MC(u_p)}{N(p)},$$

où $N(p)$ est le cardinal de l'ensemble des exemples de *S* associé à la position *p* de t_i, $MC(p)$ est le nombre d'éléments de *S* mal classés à la position *p* lorsqu'on élague t_i en position *p* et $MC(u_p)$ est le nombre d'éléments de *S* associés à la position *p* de t_i mal classés par u_p.

On considère alors la position *p* pour laquelle $g(p)$ est minimale et t_{i+1} est l'élagué de t_i en position *p*.

b. **choix final.** On calcule pour chaque arbre t_i de la suite construite au point précédent l'erreur apparente sur l'ensemble Test *T*. Cette valeur est prise comme estimation de l'erreur réelle. On retourne donc l'arbre qui minimise l'erreur apparente sur *T*.

Algorithme C4.0 L'algorithme C4.0 utilise la fonction Entropie définie par :

Soit S l'échantillon et S1, S2, ... Sk sa partition suivant les classes de l'attribut du test.

$$\mathbf{E(S) = \Sigma i\ |Si|/|S|* \log_2(|Si|/|S|)} \quad \mathbf{(2)}$$

Exemple : supposons k = 2 et soit x = |S1|/|S|. On a $E(S) = -x\log_2(x) - (1-x)\log_2(1-x)$.

$$\mathbf{Gainf\ (p,T) = f(Sp) - \Sigma j\ Pj * f(Spj)} \quad \mathbf{(3)}$$

f :entropie

Cette fonction de *x* prend ses valeurs dans l'intervalle [0,1], a son minimum pour *x*=0 et *x*=1 qui vaut 0 et a son maximum pour *x*=1/2 qui vaut 1

- Phase d'expansion : On dispose en entrée d'un ensemble d'apprentissage *S*. On utilise la fonction entropie définie par l'équation 2.

    1. **Décider si un nœud est terminal**. Un nœud *p* est terminal si tous les éléments associés à ce nœud sont dans une même classe ou si aucun test n'a pu être sélectionné.
    2. **Sélectionner un test à associer à un nœud**. À chaque étape, dans l'ensemble des tests disponibles, ne peuvent être envisagés que les tests pour lesquels il existe au moins deux branches ayant au moins deux éléments (cette valeur par défaut peut être modifiée). Si aucun test ne satisfait cette condition alors le nœud est terminal. Soit *p* une position, on choisit alors le test *test* qui maximise le gain défini dans l'équation 3 en utilisant la fonction entropie définie dans l'équation 2 pour mesurer le degré de mélange.

    3. **Affecter une classe à une feuille**. On attribue la classe majoritaire. S'il n'y a aucun exemple on attribue la classe majoritaire du nœud père.

- Phase d'élagage : L'idée est d'élaguer en estimant l'erreur : si la somme (pondérée) des erreurs des fils est supérieure à celle du père, alors on élague.

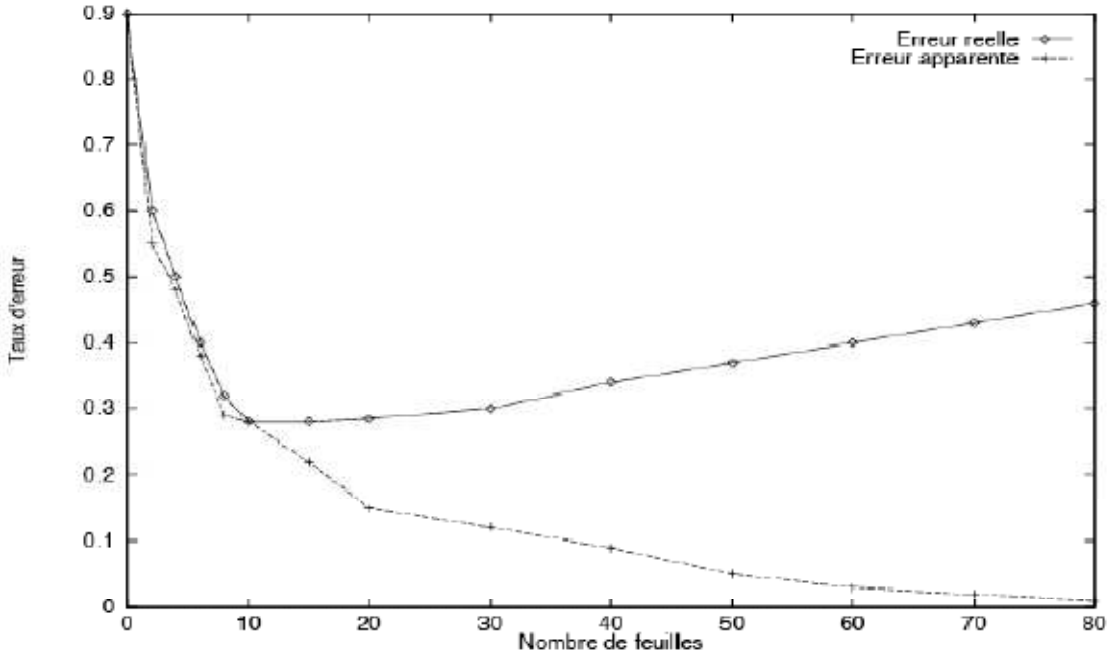
3. Avantage des arbres de décision

1. Facile à comprendre et à utiliser.
2. Nombre de tests limités par le nombre d'attributs (de *questions*).
3. **Construction** efficace (mais technique) à l'aide d'apprentissage par optimisation (pour obtenir un arbre **petit** et **« correct »**).

4. Problèmes des arbres de décisions

1. Instabilité : Un des inconvénients principaux des méthodes d'apprentissage par arbres de décision est leur *instabilité*. Sur des données réelles, un attribut est choisi plutôt qu'un autre se joue à peu de chose. Or le choix d'un attribut-test, surtout s'il est près de la racine, influence grandement le reste de la construction. Ces algorithmes ont une *variance importante.*

2. Problème de sur apprentissage : un arbre peut avoir une erreur apparente nulle mais une erreur réelle importante, c'est-à-dire être bien adapté à l'échantillon mais avoir un pouvoir de prédiction faible.



Remarques :

- Pour des attributs à plusieurs réponses (couleurs...), on peut soit traiter toutes les réponses à la fois, soit revenir à des questions oui/non.
- Pour les attributs continus (numériques), on peut tester d'éventuels « seuils » dans les valeurs (exemple pour un poids : moins de 50 g, plus de 500 g, etc...).

Réseaux de neurones

I. Introduction :

L'être humain peut traiter rapidement un très grand nombre d'informations. Ce traitement est effectué grâce au cerveau et plus précisément le neurone. Le cerveau humain est caractérisé par :

- Robustesse et tolérance aux fautes et pannes.
- Flexibilité : peut s'adapter facilement par apprentissage.
- Pouvoir de traitement d'information floue, probabiliste, bruitée ou inconsistante.
- Massivement parallèle.
- Taille réduite et faible consommation d'énergie.

Structure d'un neurone biologique :

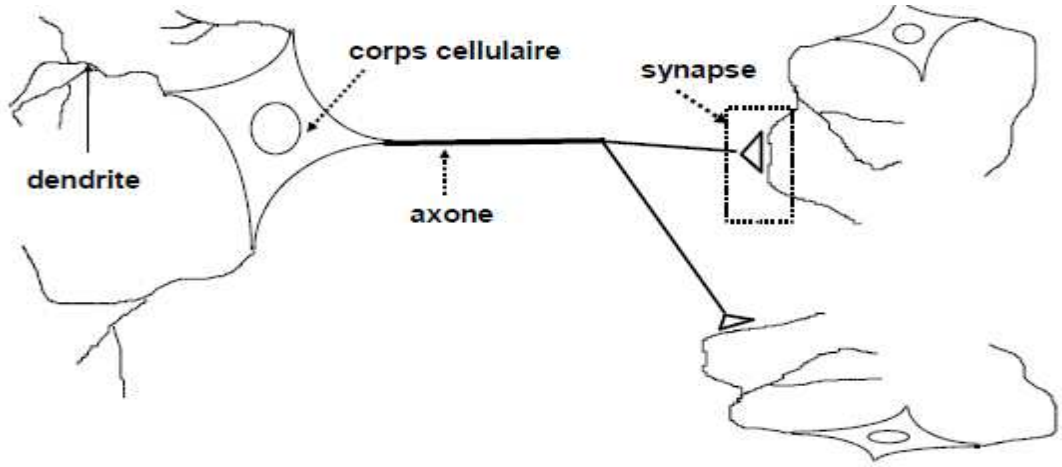


Figure 1 Neurone biologique

- Les dendrites : représentent les entrées du neurone
- Le corps cellulaire : traite l'information parvenue à travers les dendrites
- L'axone : l'information traitée est envoyée à travers l'axone. Sortie du neurone
- Synapse : jonction entre 2 neurones. Synapses excitatrices / synapses inhibitrices

II. Le neurone Formel :

La première étude de neurone artificiel a été réalisée par le neuropsychiatre McCulloch et le logicien Pitts, qui s'inspirent des travaux sur les neurones biologiques et proposèrent en 1943 le modèle suivant :

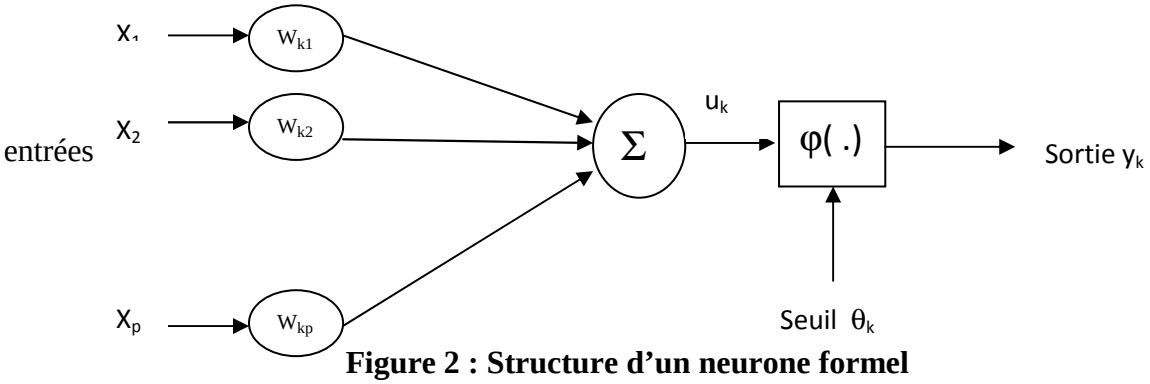


Figure 2 : Structure d'un neurone formel

Ce modèle est mathématiquement décrit par 2 équations :

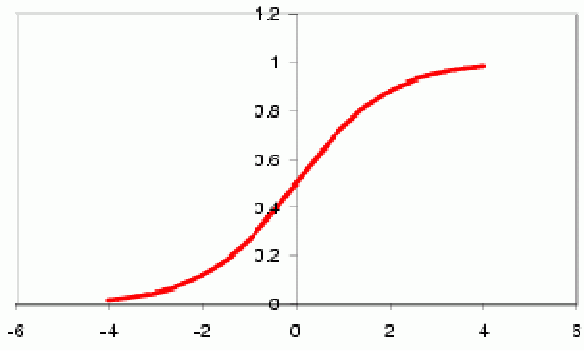
$$u_k = \sum_{j=1}^{p} w_{kj}\ x_j \quad \text{et} \quad y_k = \varphi(u_k - \theta_k)$$

- $x_1, x_2, \ldots x_p$: sont les entrées
- $w_{k1}, \ldots w_{kp}$: sont les poids synaptiques du neurone k ;
- u_k : est la sortie de l'unité de sommation
- θ_k : est le seuil
- $\varphi(.)$: est la fonction d'activation
- y_k : est le signal de sortie de neurone k.

La fonction d'activation (ou transfert) calcule la valeur de l'état du neurone qui sera transmise aux neurones avals. Il existe de nombreuses formes possibles pour cette fonction.

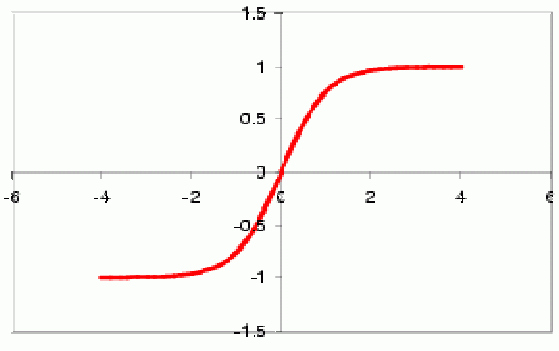
- **La fonction sigmoïde :**

Y = F(X) = 1/(1 + exp(-d*X))

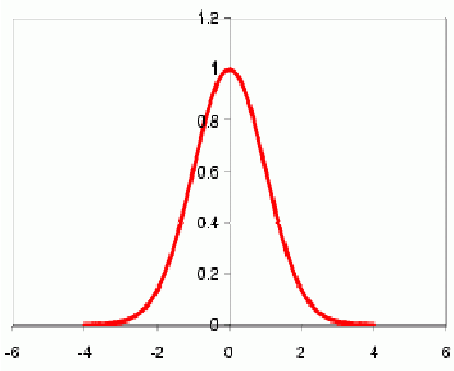


La tangente hyperbolique :

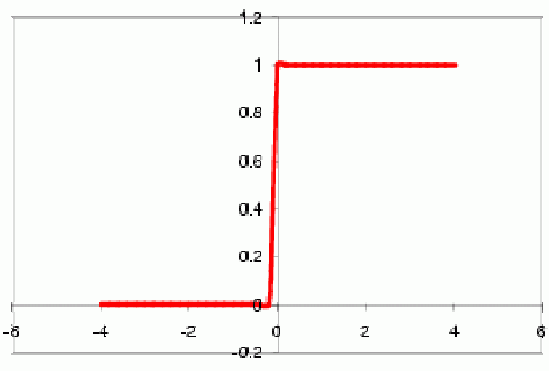
Y = 2 / (1 + exp(-2 * X)) - 1



- **La fonction Gaussienne :** Y = exp(-(X^2)/2)



Une fonction à seuil : Y = 0 si X<0 et Y=1 si X>0



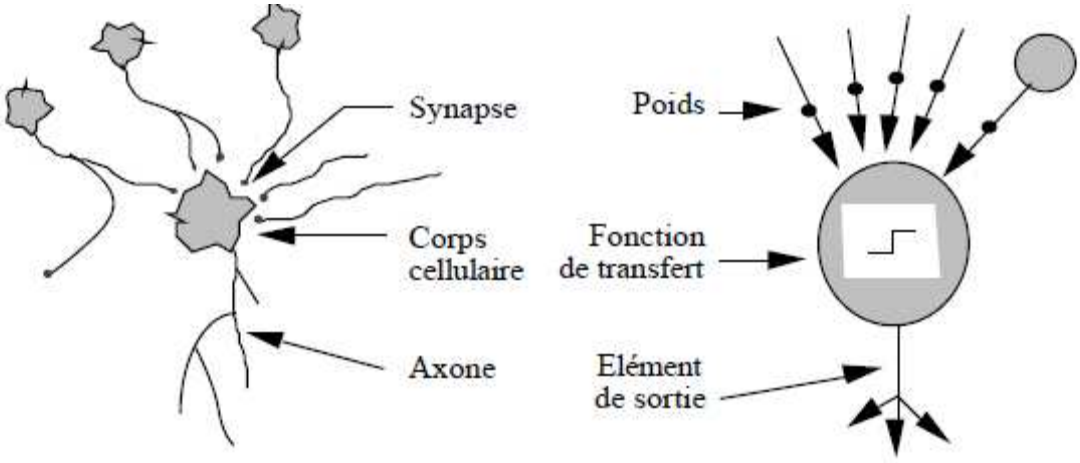


Figure 3 : correspondance entre neurone biologique et neurone artificiel

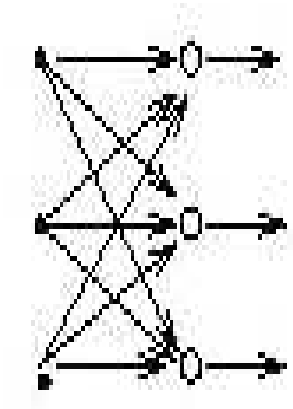
En résumé, un neurone formel réalise simplement une somme pondérée de ces entrées, ajoute un seuil à cette somme, et fait passer le résultat par une fonction de transfert pour obtenir sa sortie.

III. Les réseaux de neurones :

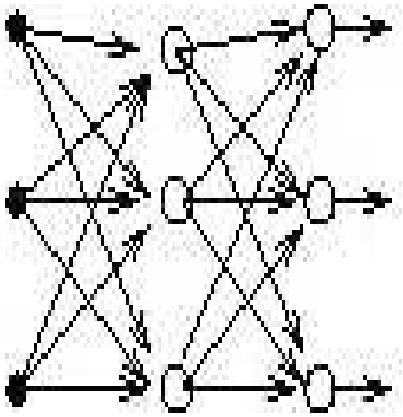
1. **Définition :**
   Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires (neurones) fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie sur la base des informations qu'il reçoit.

2. **Topologie des réseaux de neurones :**
   Il existe 2 types d'architectures :

   1. **les réseaux à couches (réseaux non récurrents, Acyclique, non bouclé, feedforward)** sont des réseaux de neurones dans lesquels l'information se propage couche par couche sans retour en arrière possible

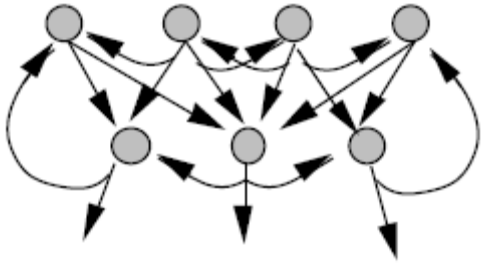


Réseau acyclique à une couche



Réseau acyclique multicouches

2. **les réseaux récurrents** sont des réseaux de neurones dans lesquels il y a une liaison vers l'arrière. Les connexions de ces réseaux forment des boucles. Ainsi la fonction d'activation peut circuler le long de ces boucles et affecter le réseau pendant une période arbitrairement longue. Pour cette raison les comportements des réseaux récurrents sont potentiellement plus complexes que ceux des réseaux à couches.

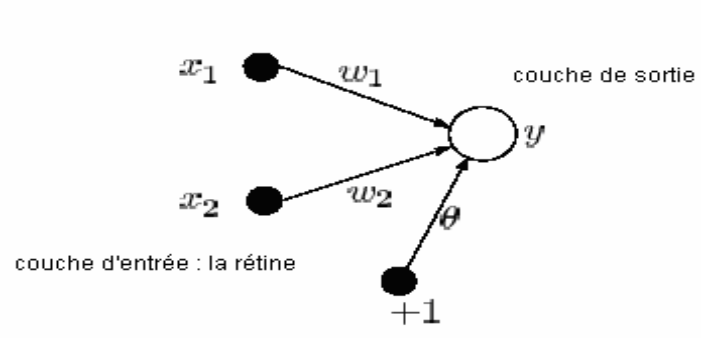


IV. Les grands types de réseaux

Il existe plusieurs modèles de réseaux. Nous citons quelques uns :

- **Le Perceptron monocouche :** inventé par Rosenblatt(1958). C'est un modèle très simple. Il ne dispose que deux couches :

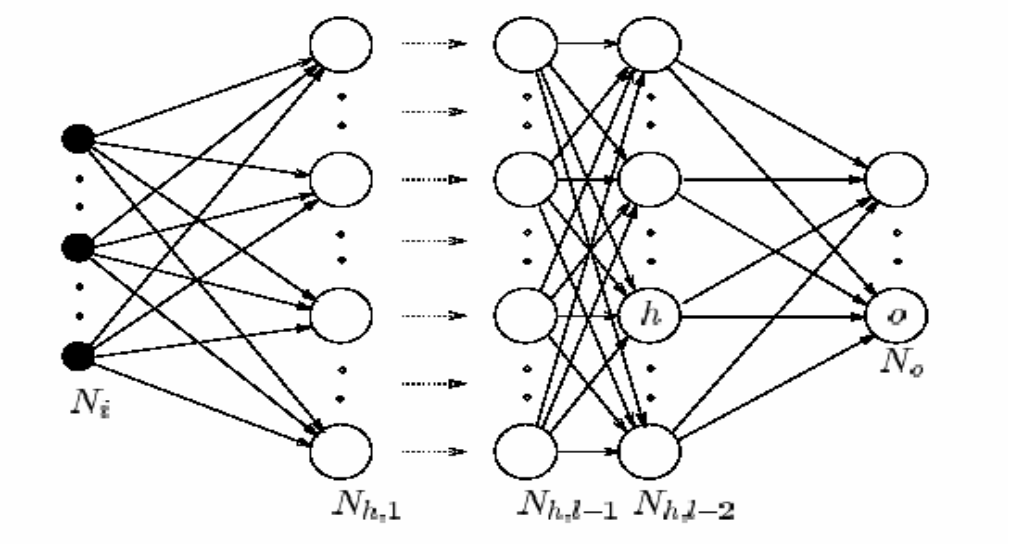
  _ une couche d'entrée qui s'appelle la rétine et qui est une aire sensorielle ;
  _ une couche de sortie qui donne la réponse correspondante à la simulation présentée à l'entrée.
  Le fonctionnement est le suivant : une donnée est présentée au réseau en activant la rétine. L'activation se propage vers la couche de sortie où on peut noter la réponse du système.
  La figure suivante montre la structure de Perceptron monocouche, avec une sortie *y*



- **Les Perceptrons Multicouches :** est un modèle d'une plus grande capacité de calcul. Sa structure est composée d'une couche d'entrée, une couche de sortie, interprétée comme la réponse du réseau et d'une ou plusieurs couches intermédiaires dites « couches cachées ». Un neurone d'une couche inférieure ne peut être relié qu'à des neurones des couches suivantes. Il suit un apprentissage supervisé et utilise une règle d'apprentissage par rétro propagation. En général, les neurones du Perceptron multicouche sont animés par

une fonction d'activation non linéaire (au moins dans une des couches). Les choix classiques pour cette fonction sont :

- la fonction tangente hyperbolique
- la fonction sigmoïde



- **Les Réseaux de Hopfield**
  Ces réseaux sont des réseaux récursifs, un peu plus complexes que les perceptrons multicouches. Chaque cellule est connectée à toutes les autres et les changements de valeurs de cellules s'enchainent en cascade jusqu'à un état stable. Ces réseaux sont bien adaptés à la reconnaissance de formes.

V. Applications des réseaux de neurones :

Les réseaux de neurones ont de nombreuses applications possibles. En voici quelques exemples:

- Approximation de fonctions: les fonctions trop compliquées peuvent être approximées, grâce au réseau, par une somme de fonctions plus simples comme des polynômes ou des sigmoïdes.

- Optimisation de trajectoires: en intégrant les paramètres tels que le vent, les conditions extérieures, ... On peut, par exemple, déterminer quelle est la meilleure trajectoire pour un avion,une fusée...

- Reconnaissance: un réseau peut servir à reconnaître des caractères. Cela est déjà utilisé à la Poste pour lire les codes postaux, ou même dans certaines banques pour lire les chèques. Il est aussi possible de retrouver la prononciation des mots à partir d'un texte.

- Prévision: on utilise de plus en plus les réseaux pour faire des prévisions en marketing (prédiction de comportement, de possibilité de vente d'un produit, ...) ou pour

le trafic routier... Mais les prévisions en météo ou en bourse sont trop compliquées à réaliser.

- Contrôle : on peut contrôler les produits dans une industrie.

- Robotique: certains robots sont dotés de réseaux de neurones. Des entreprises japonaises se vantent déjà de leur utilisation, même pour des produits électroménagers ou informatiques.

Les réseaux de neurones ont encore beaucoup d'autres applications possibles. En général, cela se résume en un problème d'optimisation.

VI. Les étapes de la conception d'un réseau de neurones :

Voici chronologiquement les quatre grandes étapes qui doivent guider la création d'un réseau de neurones.

- **Choix et préparation des échantillons :** Le processus d'élaboration d'un réseau de neurones commence toujours par le choix et la préparation des échantillons de données. Comme dans les cas d'analyse de données, cette étape est cruciale et va aider le concepteur à déterminer le type de réseau le plus approprié pour résoudre sont problème. La façon dont se présente l'échantillon conditionne : le type de réseau, le nombre de cellules d'entrée, le nombre de cellules de sortie et la façon dont il faudra mener l'apprentissage, les tests et la validation.

- **Elaboration de la structure du réseau :** La structure du réseau dépend étroitement du type des échantillons. Il faut d'abord choisir le type de réseau : un perceptron standard, un réseau de Hopfield, un réseau à décalage temporel (TDNN), un réseau de Kohonen, un ARTMAP etc... Dans le cas du perceptron par exemple, il faudra aussi choisir le nombre de neurones dans la couche cachée. Plusieurs méthodes existent et on peut par exemple prendre une moyenne du nombre de neurones d'entrée et de sortie, mais rien de vaut de tester toutes les possibilités et de choisir celle qui offre les meilleurs résultats.

- **Apprentissage :** L'apprentissage consiste tout d'abord à calculer les pondérations optimales des différentes liaisons, en utilisant un échantillon. La méthode la plus utilisée est la *rétropropagation* : on entre des valeurs dans les cellules d'entrée et en fonction de l'erreur obtenue en sortie (le *delta*), on corrige les poids accordés aux pondérations. C'est un cycle qui est répété jusqu'à ce que la courbe d'erreurs du réseau ne soit croissante (il faut bien prendre garde ne pas sur-entrainer un réseau de neurones qui deviendra alors moins performant).

- **Validation et Tests :** Alors que les tests concernent la vérification des performances d'un réseau de neurones hors échantillon et sa capacité de généralisation, la validation est parfois utilisée lors de l'apprentissage. Une fois le réseau calculé, il faut toujours procéder à des tests afin de vérifier que notre réseau réagit correctement.

Le Perceptron Multicouches (MLP)

I. Perceptron Simple :

Le perceptron est un modèle de réseau de neurones avec algorithme d'apprentissage créé par Frank Rosenblatt en 1958.

1. Définition du Perceptron

Un perceptron linéaire à seuil (figure1) prend en entrée n valeurs x_1,..., x_n et calcule une sortie o. Un perceptron est défini par la donnée de n+1 constantes : les coefficients synaptiques w_1, ..., w_n et le seuil (ou le biais). La sortie o est calculée par la formule :

$$o = \begin{cases} 1 \; si \; \sum_{i=1}^{n} w_i x_i > \theta \\ 0 \qquad sinon \end{cases}$$

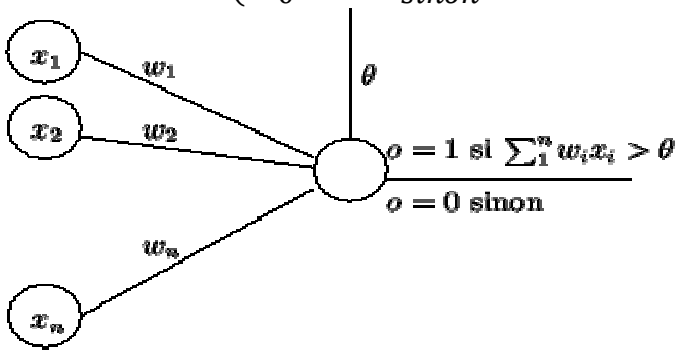


Figure 1 : Le perceptron avec seuil

- Les entrées x_1,..., x_n peuvent être à valeurs dans {0,1} ou réelles, les poids peuvent être entiers ou réels. Une variante très utilisée de ce modèle est de considérer une fonction de sortie prenant ses valeurs dans {-1,1} plutôt que dans {0,1}. Il existe également des modèles pour lesquels le calcul de la sortie est probabiliste.

- Le seuil peut être remplacé par une entrée supplémentaire x_0 qui prend toujours la valeur d'entrée x_0=1. À cette entrée est associé un coefficient synaptique w_0 (figure 2). On peut décomposer le calcul de la sortie o en un premier calcul de la quantité : $a=\Sigma (w_i \; x_i)$ appelée potentiel post-synaptique ou l'entrée totale suivi d'une application d'une fonction d'activation sur cette entrée totale. La fonction d'activation est la fonction de Heaviside définie par :

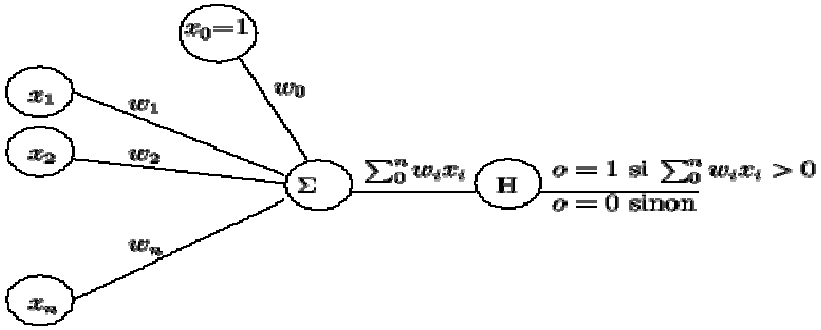


Figure 2 : Le perceptron avec entrée supplémentaire

L'équivalence entre le modèle avec seuil et le modèle avec entrée supplémentaire à 1 est immédiate : le coefficient w_0 est l'opposé du seuil.

Exemple :
Un perceptron qui calcule le OU logique avec les deux versions : seuil ou entrée supplémentaire est présenté dans la figure 3 :

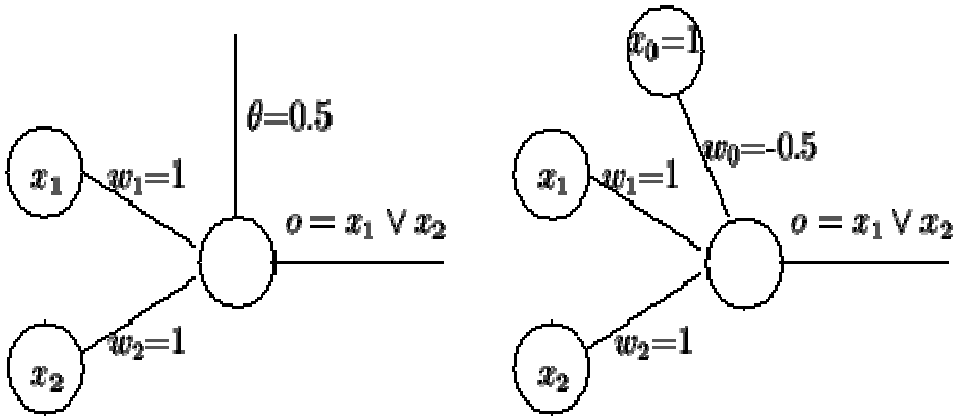


Figure 3 : perceptrons qui calculent le OU

Remarque :
On voit que quelques uns des traits principaux des neurones réels ont été retenus dans la définition du perceptron : les entrées modélisent les dendrites, les impulsions en entrée sont pondérées par les coefficients synaptiques et l'impulsion émise, c'est-à-dire la sortie, obéit à un effet de seuil (pas d'impulsion si l'entrée totale est trop faible).

2. Apprentissage du Perceptron :

Algorithme de Widrow-Hoff:

Entrée : un échantillon d'apprentissage *S (données_exemples, sorties_désirées)*
Initialisation aléatoire des poids w_i pour *i* entre 1 et *n par de petites valeurs*
Répéter jusqu'à (tous les exemples présentés et aucune modification des poids)
Prendre un exemple ($\vec{x}$, *d*) dans *S* ($\vec{x}$ =(x_1, ..x_n , d :sortie désirée)
Calculer la sortie *o* du perceptron pour l'entrée $\vec{x}$
Si différence entre sortie désirée et calculée **Alors** /* Mise à jour des poids */
Pour *i* de 1 à *n*

$$\Delta w_i = \eta\,(d-o)x_i$$
$$w_i \leftarrow w_i + \Delta w_i$$

Finpour
FinSi
finRépéter

Sortie : Un perceptron *P* défini par (w_0, w_1, ..., w_n)

Remarque : le taux d'apprentissage est un paramètre fixé par le concepteur.

$$w_i \leftarrow w_i + \Delta w_i$$

nouveau poids *ancien poids* *incrément*

$$\Delta w_i = \eta (d - o) x_i$$

Incrément *taux d'apprentissage* *valeur désirée* *valeur de sortie* *entrée*

II. Perceptron multicouches (Multi Layer Percetron) :

1. Définition :

Un perceptron multicouches est un réseau orienté de neurones :

- La première couche C_0 est la rétine composée des neurones d'entrée reliés au monde extérieur, ils correspondent aux n variables d'entrée qu'ils passent sans modification
- Les couches C_1, ..., C_{q-1} sont les couches cachées ;
- la couche C_q correspond aux sorties du système
- Les neurones de 2 couches adjacentes sont complètement connectés (les entrées de la couche i sont les sorties de la couche i-1)
- Les neurones sont reliés entre eux par des connexions pondérées, la fonction de transformation est non linéaire dérivable (sigmoïde)
- L'apprentissage est supervisé et réalisé par l'algorithme de rétro-propagation du gradient.

/

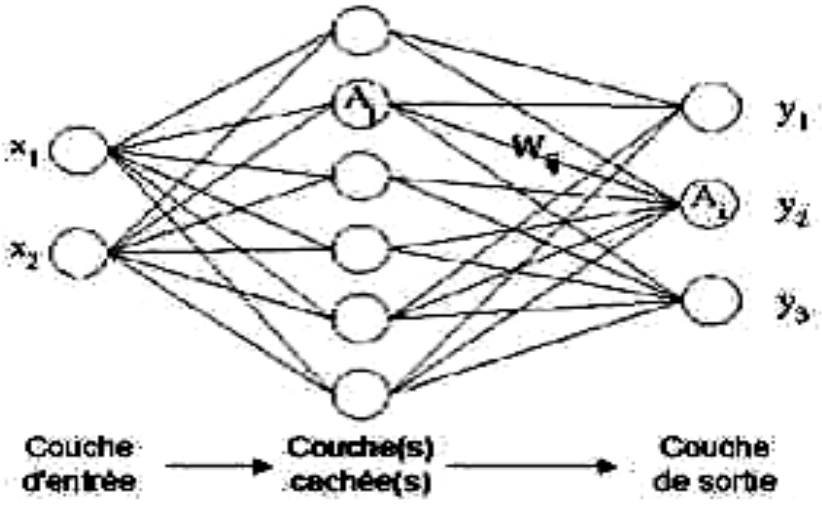


Figure 4 : perceptron multicouches

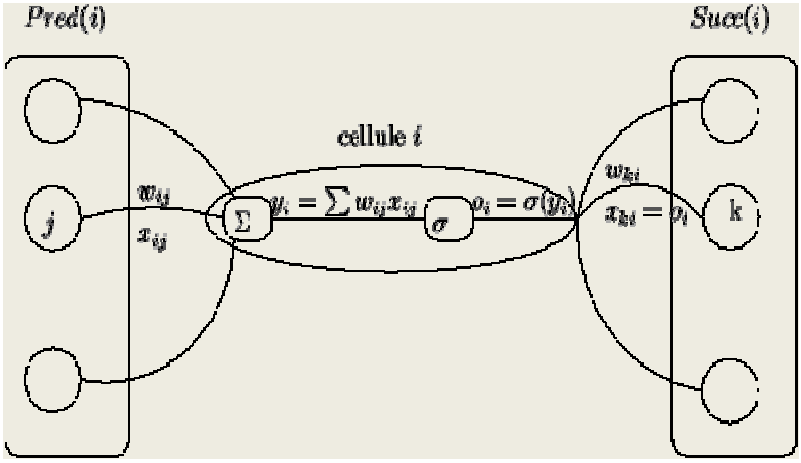


Figure 5 : représentation d'une cellule du réseau

2. Apprentissage dans un MLP :

La rétro-propagation du gradient est l'une des méthodes les plus utilisées pour l'apprentissage dans un MLP, elle est basée sur la règle « du gradient de l'erreur ».

L'objectif de cet algorithme est de minimiser l'erreur quadratique :

$$\boldsymbol{E[w] = \frac{1}{2}\sum_{e}(d_e - o_e)^2}$$

d_e : la sortie désirée pour l'exemple e **o_e : la sortie calculée**

a. Principe de la rétro-propagation :

On présente au réseau des entrées et on lui demande de modifier sa pondération pour retrouver la sortie désirée :

- 1ère Etape : on propage vers l'avant les entrées jusqu'à obtenir les sorties calculées.
- 2ème Etape : on compare la sortie calculée avec la sortie réelle connue. On modifie les poids de telle sorte qu'à la prochaine itération l'erreur entre la sortie calculée et la sortie connue soit minimisée.
- 3ème Etape : on rétro propage l'erreur vers l'arrière jusqu'à la couche d'entrée tout en modifiant les poids.
- On répète ce processus sur tous les exemples jusqu'à ce que la durée d'apprentissage soit écoulée.

b. Algorithme de rétropropagation du gradient :

Entrée : un échantillon *S*, η(taux d'apprentissage)

un PMC avec 3 couches : couche d'entrée *I*, couche cachée H
une couche de sortie *K*, *n* cellules.
Initialisation aléatoire des poids w_i dans [-0.5, 0.5] pour *i* entre 1 et *n*

Répéter

Prendre un exemple ($\vec{x}$, $\vec{d}$) de *S* et calculer

- *Pour chaque unité cachée h :* $o_h = \sigma \sum_i w_{hi} x_i$ avec $\sigma(x) = \frac{1}{1+e^{-x}}$
- *Pour chaque unité de sortie k :* $o_k = \sigma \sum_k w_{kh} x_h$ *avec* $x_h = o_h$

- - calcul des erreurs δ_i (les gradients d'erreurs) par rétro-propagation

Pour toute cellule de sortie *k* : $\delta_k \leftarrow o_k(1-o_k)(d_k-o_{ik})$ **finPour**

Pour chaque cellule *h* de la couche cachée

$\delta_h = o_h(1-o_h) \sum_k \delta_k w_{kh}$

finPour

- - mise à jour des poids

Pour tout poids $w_{ij} \neg w_{ij} + \eta \delta_i x_{ij}$ **finPour**

finRépéter

Sortie : Un PMC défini par la structure initiale choisie et les w_{ij}

***Remarque* :**

Dans le cas d'une fonction d'activation *f* autre que la fonction sigmoïde le calcul des gradients d'erreurs devient comme suit :

Pour toute cellule de sortie *k* : $\delta_k \leftarrow f'(a_k)(d_k-o_k)$ **finPour** *f'* est la dérivée de *f*

Pour chaque cellule *h* de la couche cachée

$\delta_h = f'(a_h) \sum_k \delta_k w_{kh}$ tel que k successeur (h) **FinPour**

Remarques :

- la règle de modification des poids pour le perceptron linéaire est : $w_i \rightarrow w_i + \eta (d-o) x_i$. Dans le cas du PMC, cette règle est : $w_{ij} \rightarrow w_{ij} + \eta \delta_i x_{ij}$ .Ces deux règles sont très similaires, le terme d'erreur ***d-o*** est remplacé par un terme plus compliqué δ_i **,**
- pour une cellule i de sortie, la quantité δ_i correspond à l'erreur usuelle (d_i-o_i) multipliée par la dérivée de la fonction sigmoïde,
- pour une cellule i interne, le calcul δ_i de dépend de la somme pondérée des erreurs des

cellules de la couche suivante,

- δ_i est obtenue par descente du gradient de la dérivée partielle de E par rapport à w_i, $\frac{\partial E(w)}{\partial w_i}$
- après présentation de l'entrée $x^{\rightarrow}$ et calcul de la sortie $o^{\rightarrow}$, le calcul des erreurs δ_i sera effectué de la couche de sortie vers la couche d'entrée.
- 2 façons d'appliquer l'algorithme de rétro-propagation :
  → **« batch »** : mise à jour des poids après la présentation de tous les exemples, calculs et stockage plus lourds si trop d'exemples
  → **« séquentiel »** : (on-line, stochastique) mise à jour des poids après chaque exemple, besoin de tirer l'exemple au hasard et problèmes de convergence

3. Applications :

De nombreuses applications de l'algorithme de rétro-propagation du gradient ont été réalisées. Parmi les plus souvent citées,: *NetTalk* de Sejnowski

NetTalk : est un réseau qui a appris à transformer un texte (en anglais) en une suite de phonèmes correspondant à sa lecture. Couplé en entrée à un scanner et à un OCR (Optical Character Recognition) et en sortie à un synthétiseur de paroles, ce réseau est donc capable de lire un texte à haute voix.

Description de l'architecture du réseau :

- la couche d'entrée comprend 7 groupes de 29 neurones. Chaque groupe correspond à un caractère codé directement (non compressé). Les 7 caractères en entrée forment un contexte local de trois caractères entourant de part et d'autre un caractère central. Par exemple, le caractère 'c' se prononce différemment dans 'cygne' et dans 'carte'. Les ambiguïtés les plus courantes semblent pouvoir être levés avec un tel contexte.
- la couche cachée contient 80 neurones
- la couche de sortie comprend 26 neurones servant à coder les caractéristiques des phonèmes : la *zone vibratoire* (labiale, dentale, ...), le *type* de phonème (arrêt, nasale, fricative, ...), la *hauteur* des voyelles, la *ponctuation* (silence, pause, élision, arrêt net), l'*accentuation*, ...

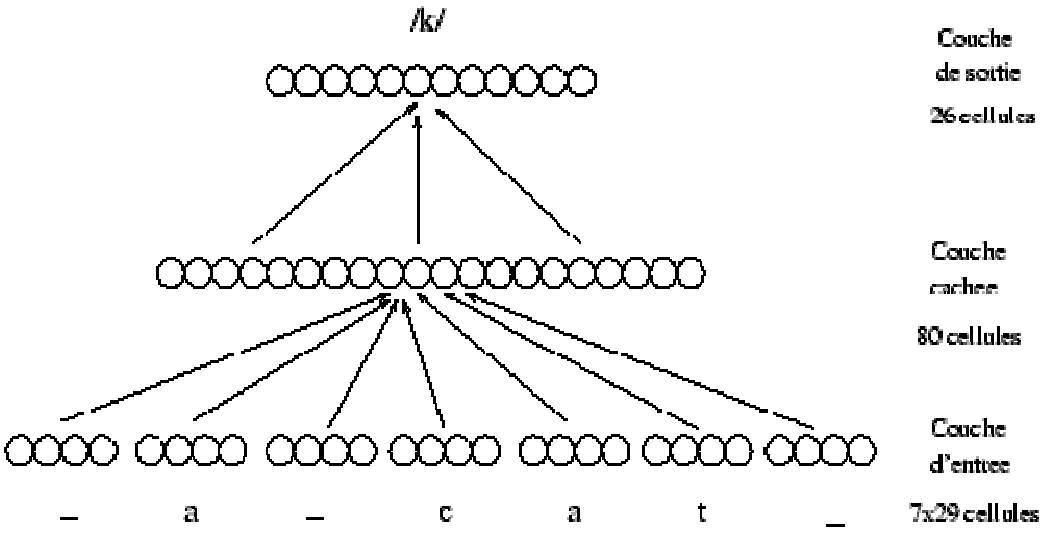


Figure 5 : NetTalk

III. Conclusion :

Algorithme de rétro-propagation:

- Bon prédicteur
- Risque de surapprentissage
- Choix des paramètres (nb neurones, nb couches, taux d'apprentissage) laissés au concepteur