2. APPRENTISSAGE AUTOMATIQUE

Dans ce chapitre, nous allons présenter un aperçu des différents outils d'apprentissage automatique, parmi lesquels on distingue deux axes principaux de recherche : les approches symboliques et les approches connexionnistes (sub-symboliques). Nous allons tout d'abord proposer une brève introduction sur l'importance et sur le rôle de ces algorithmes d'apprentissage automatique dans le domaine de *l'Intelligence Artificielle* (I.A.), des *Systèmes Experts* et des *Systèmes Intelligents* [KAN 92]. Ensuite, nous allons présenter plus en détail les systèmes d'apprentissage symbolique et les systèmes d'apprentissage connexionniste, pour aboutir au concept de système d'acquisition de connaissances hybride. Nous nous intéresserons plus particulièrement aux Systèmes Hybrides Neuro-Symboliques (S.H.N.S. [ORS 95]).

2.1 LES SYSTEMES EXPERTS ET L'ACQUISITION DE CONNAISSANCES

Le terme Intelligence Artificielle (I.A.) a été introduit par McCarthy en 1956 pendant une conférence au Dartmouth College, et depuis, ce terme a été retenu pour représenter le domaine. L'I.A. est directement liée aux concepts de Systèmes à bases de Connaissances, Systèmes Experts, Systèmes Intelligents, Acquisition de Connaissances, Apprentissage Automatique, parmi d'autres sujets d'étude et d'application pratique. (voir le Tableau 1 - Applications de L'I.A.). Dans cette thèse, nous allons nous concentrer plutôt sur le sujet IV indiqué dans le tableau ci-dessous, avec des études complémentaires à propos des items II et V.

Domaines de l'I.A. : Sujets d'étude	**Exemples d'Applications Pratiques**
I - Vision par Ordinateur : Reconnaissance d'Images, Analyse d'Images et Traitement de Signaux	Reconnaissance de formes, classification d'images, reconnaissance de textes, conduite autonome de véhicules.
II - Robotique Intelligente et Evolution Artificielle	Systèmes réactifs, robotique autonome, robots intelligents, contrôle et planification de trajectoires, *animats*.
III - Traitement du Langage Naturel	Traduction automatique, reconnaissance de la parole, analyse automatique de textes, indexation et recherche automatique d'informations.

IV - Raisonnement Automatique et Acquisition de Connaissances : Systèmes Experts, Systèmes Intelligents et Systèmes Hybrides, Apprentissage Automatique, Représentation de Connaissances	Systèmes intelligents pour l'exploration de données (*data mining, knowledge data discovery-KDD*), systèmes d'aide au diagnostic (e.g. médical, industriel), systèmes pour la classification et/ou la prévision, systèmes d'aide à la décision, systèmes de raisonnement fondé sur des cas (*CBR*), systèmes à base de connaissances (*KBS*).
V - Modélisation Cognitive	Proposition, analyse et validation de modèles du comportement cognitif humain (modèles fondés sur des études psychologiques).
VI - I.A. Distribuée et Systèmes Multi-Agents	Modélisation et conception de systèmes fondés sur des interactions avec multiples modules, agents coopératifs et collaboratifs (e.g. modèles sociaux, prise de décisions en groupe, comportements collectifs en robotique, applications militaires), théorie des jeux.
VII - Logique Formelle	Logiques pour l'I.A. (e.g. logique temporelle), mécanisation des logiques, langages d'I.A. (e.g. Prolog), preuve automatique de théorèmes.

Tableau 1 - Applications de L'I.A. (Liste non exhaustive)

Le terme d'Intelligence Artificielle est un peu provocateur, puisque le concept d'un agent ou processus intelligent n'est pas très bien définie : il n'existe pas de vrai consensus sur la frontière entre ce qui est intelligent et ce qui ne l'est pas, ni au sujet de l'évaluation de l'intelligence d'un système.

D'une façon générale, on peut dire que, " le but de l'I.A. est de construire des systèmes qui puissent exhiber un comportement intelligent et réaliser des tâches complexes avec un niveau de compétence qui est équivalent, sinon supérieur, à celui exhibé par des experts humains " [NIK 97]. Quand les recherches en I.A. ont démarré, un grand effort a été déployé dans la recherche d'un outil capable de résoudre n'importe quel type de problème (les systèmes nommés "General Problem Solvers - GPS"). Ce type d'approche s'est avéré trop ambitieux, car on n'est pas capable de créer un seul et unique système qui puisse traiter en même temps toute une large gamme de problèmes complexes. En conséquence de cet échec, un changement de cap est survenu. Les chercheurs ont conclu que des systèmes spécialisés utilisés dans la résolution de certains problèmes pourraient être développés plus facilement, et quand même être appliqués dans différents domaines (les connaissances sur le problème changent , mais les mécanismes de base restent à peu près les mêmes). A la place de chercher une seule et unique solution générale omnipotente de résolution de problèmes, il vaut mieux alors disposer d'outils spécialisés dans la résolution de tâches plus spécifiques.

Les systèmes spécialisés dans la solution de problèmes plus spécifiques sont donc apparus. Ces systèmes ont étés nommés des *systèmes experts* [HAY 83]. Ce type de système appartient à la classe des *systèmes à bases de connaissances (Knowledge-Based Systems - KBS)*.

2.1.1 Les Systèmes Experts

Feigenbaum [FEI 79] a défini un système expert comme étant " un logiciel intelligent qui utilise des connaissances et un procédé d'inférence pour résoudre des problèmes. Ces problèmes sont assez difficiles à résoudre et requièrent une expertise humaine significative pour arriver à une solution. Les connaissances d'un système expert sont composées de faits et de heuristiques ". Cette définition est d'une façon générale encore valable, comme on pourra le constater dans la suite de ce chapitre. Pourtant, elle est un peu dépassée quand on fait référence aux *Systèmes Experts de Deuxième Génération* [NIK 97, DAV 93]. Ce deuxième type de système expert est plus développé et possède des propriétés telles que l'intégration de différentes méthodes de raisonnement et l'automatisation de l'acquisition de connaissances. Comme on le verra dans la suite, notre étude a été dirigée plutôt vers ce type particulier de système expert.

Donc, les systèmes experts sont faits pour résoudre des problèmes dans un domaine spécifique d'expertise et ne sont pas des outils généraux de résolution de problèmes. Par exemple, un système expert ne va probablement jamais résoudre des problèmes d'un domaine aussi large que le domaine médical, mais peut toutefois s'adresser parfaitement à des sous-domaines spécifiques du domaine médical. Cela a été démontré par des systèmes comme MYCIN [SHO 76] et DENDRAL [LIN 80], des systèmes experts très connus utilisés pour le diagnostic de maladies infectieuses sanguines et pour l'identification de structures moléculaires respectivement. Actuellement, les systèmes experts couvrent un large domaine d'application [DUR 93, HAY 83], dans lequel on peut distinguer deux principales applications : les systèmes d'aide à la décision et les systèmes d'aide au diagnostic. Ces systèmes doivent être conçus pour *aider* l'homme dans la réalisation de tâches difficiles et ne doivent jamais être utilisés pour le *remplacer* complètement.

Les systèmes experts ont des caractéristiques particulières qui les distinguent des autres systèmes informatiques. Parmi ces caractéristiques, on trouve des éléments tels que : l'architecture particulière de ses modules ; la codification des connaissances dans une base de connaissances ; la séparation des connaissances de la partie de contrôle ; la capacité à raisonner sur des données incomplètes, inexactes et floues (sur certains systèmes experts) ; les outils d'acquisition de connaissances ; la possibilité d'explication des résultats. Ces capacités sont principalement dues au fait que ces systèmes sont capables de dériver des solutions à partir d'une heuristique, plutôt que par les approches algorithmiques employées dans les systèmes informatiques conventionnels.

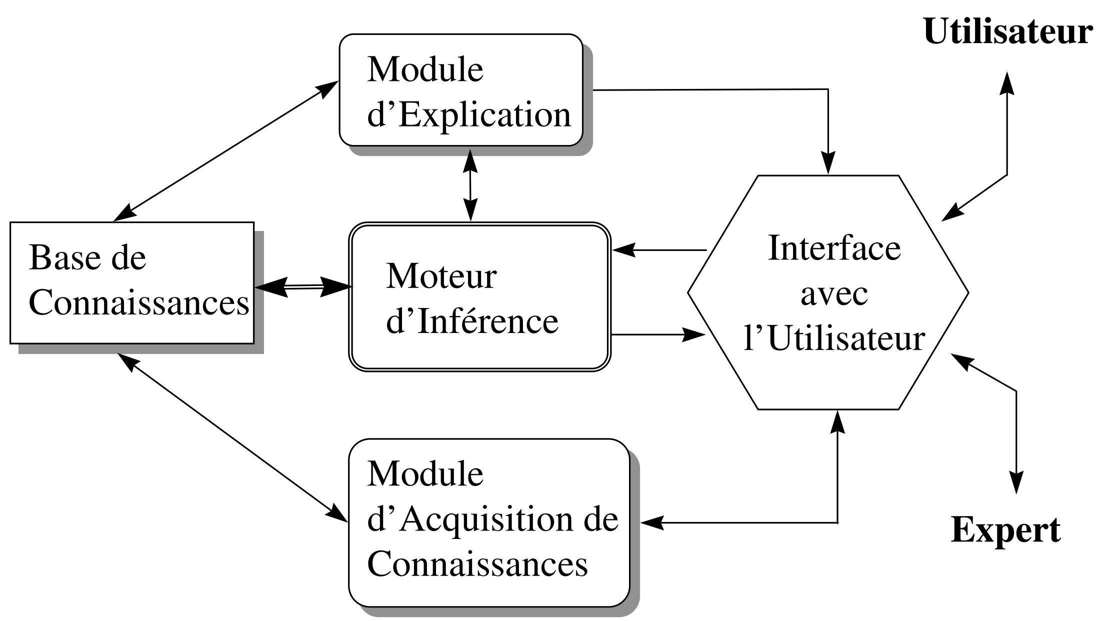


Figure 1 - L'Architecture d'un Système Expert

Les systèmes experts sont en général composés d'une architecture à cinq modules (voir Figure 1 - L'Architecture d'un Système Expert) :

1. ***Base de Connaissances*** : la base de connaissances contient les connaissances spécifiques du domaine qui sont utilisées pour résoudre un certain problème. Elle est composée

usuellement par des règles et des faits. Nous allons aborder plus en détail les aspects liés à ce module dans la section 2.1.2;

2. ***Moteur d'Inférence*** : le moteur d'inférence permet de manipuler les données stockées dans la base de connaissances de façon à pouvoir résoudre les problèmes posés au système. Le moteur d'inférence reste séparé des connaissances du domaine et généralement il n'est pas dépendant de son application à un problème particulier. Cela ne veut pas dire que l'on a un seul type de moteur d'inférence, car chaque formalisme de représentation possède ses propres techniques d'inférence et donc, le mécanisme d'inférence reste directement lié au type de représentation des connaissances employé. Un moteur d'inférence est typiquement fondé sur une règle d'inférence (permet de réaliser un raisonnement et de déduire de nouvelles connaissances à partir des connaissances de la base) et une stratégie de recherche [HAY 83, NIK 97];
3. ***Module d'acquisition de connaissances*** : ce module permet de créer, ajouter et maintenir les connaissances nécessaires pour la résolution d'un problème dans un domaine d'application. L'acquisition de connaissances peut être faite par explicitation de connaissances d'un domaine avec l'aide d'un ingénieur de connaissances et de l'expert, ou à travers des processus semi-automatiques ou totalement automatiques d'acquisition de connaissances. Ces méthodes automatiques sont connues comme *méthodes d'apprentissage automatique* ("machine learning methods"). La Figure 2 présente un schéma simplifié du processus d'acquisition de connaissances. Dans la section 2.1.3, nous allons présenter plus en détails les méthodes d'acquisition de connaissances;
4. ***Module d'explication*** : ce module permet à l'utilisateur de connaître les processus et le raisonnement qui ont amené le système à donner une certaine réponse. De cette façon, il est possible à l'utilisateur de poser des questions et interagir avec le système afin de comprendre les réponses fournies. Le système est capable de *justifier* ses réponses.
5. ***Interface avec l'utilisateur*** : l'interface permet l'échange d'informations entre l'utilisateur, l'expert, l'ingénieur de connaissances et les différents modules du système expert.

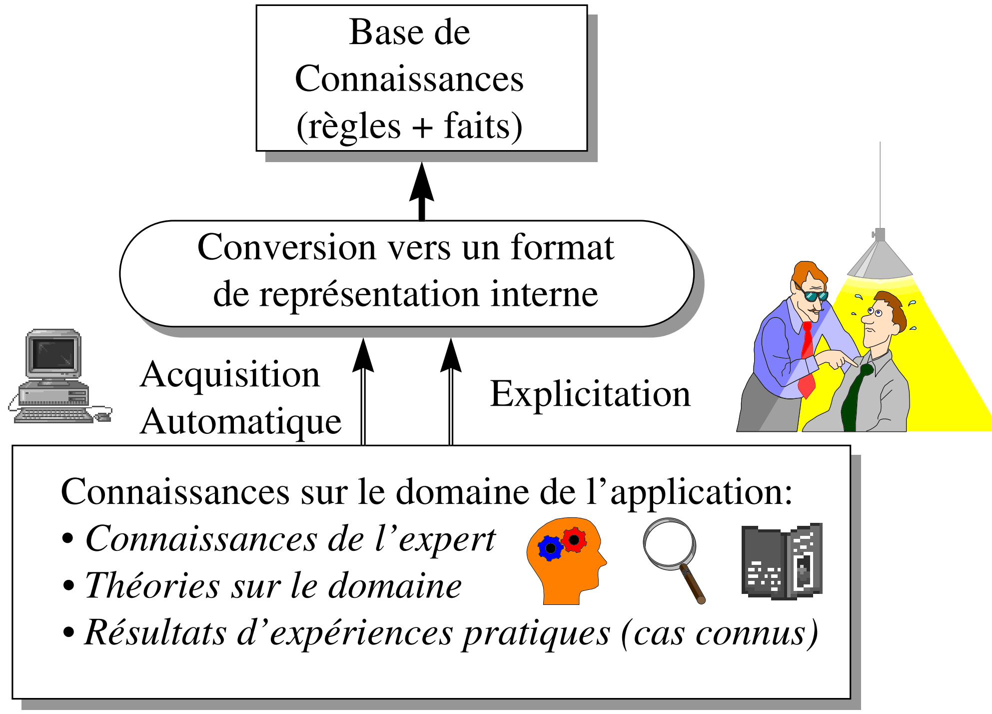


Figure 2 - Acquisition de Connaissances

Les systèmes experts permettent d'automatiser certains aspects du raisonnement humain, en utilisant des méthodes d'inférence telles que l'induction, la déduction et le raisonnement par analogie. Ces méthodes nous permettent d'inférer des propositions en utilisant des règles d'inférence (e.g. modus ponens, modus tollens, double négation, règle de la chaîne, résolution) [NIL 80, STE 93]. Certains types de moteurs d'inférence sont très connus, comme : la méthode de chaînage avant et la méthode de chaînage arrière.

Dans les méthodes de chaînage avant, c'est à partir des prémisses (faits) qu'on arrive aux conclusions. Nous avons utilisé dans le développement de nos recherches un système exploitant ce type de méthode d'inférence. Ce système s'appelle le langage **CLIPS** (*C Language Integrated Production System*) [GIA 93, CUL 93], et il sert à la construction de systèmes experts. CLIPS a été développé par le *Software Technology Branch* de la NASA aux U.S.A. Il inclut un moteur d'inférence par chaînage avant fondé sur l'algorithme *Rete* [FOR 82]. Nous l'avons utilisé pour implémenter le module symbolique de notre système hybride INSS (voir Section 4.1).

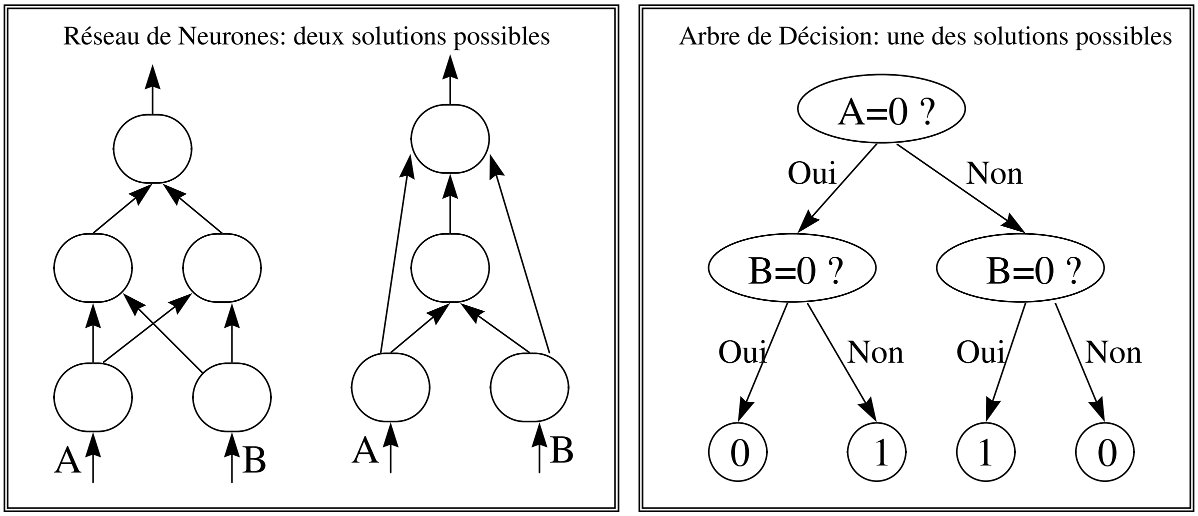
La méthode de chaînage arrière part des conclusions. Elle cherche à trouver, à partir des conclusions, les prémisses nécessaires à leur déclenchement. Un autre exemple d'outil informatique utilisé pour la construction de systèmes experts est le langage Prolog (Programmation en Logique) [COL 85, BRA 90]. Ce langage utilise une méthode de chaînage arrière dans l'implémentation du moteur d'inférence par une méthode de résolution. Ce type de méthode automatique de raisonnement, employé par les langages comme Prolog, repose sur l'idée que "tout ce qui n'est pas vrai est faux et tout ce qui n'est pas faux est vrai". C'est pour cela que de tels systèmes ont besoin d'une théorie (base de règles et faits) qui soit à la fois correcte et complète sur un domaine.

En conclusion, les systèmes experts utilisent des méthodes de raisonnement automatique. Ils nécessitent un moteur d'inférence spécifique et une base de connaissances. Cette base doit être composée par des structures de données bien adaptées au traitement automatique des informations et au moteur d'inférence employé par le système.

2.1.2 La Représentation des Connaissances

De nombreuses activités "intelligentes" de l'être humain, aussi bien dans sa vie quotidienne que dans son activité professionnelle, reposent sur l'exploitation d'une masse importante d'informations, de faits, d'expériences et de connaissances plus ou moins spécifiques d'un domaine particulier. De ce fait, une partie importante des travaux de recherche et de développement en I.A. concerne la conception de formalismes permettant la mise en oeuvre de systèmes à bases de connaissances, et plus spécifiquement l'étude des différentes façons de spécifier et de créer les bases de connaissances. Nous allons nous concentrer ici sur les problèmes de la représentation des connaissances et de leur façon de s'adapter aux différents systèmes de raisonnement par ordinateur.

La représentation de connaissances est un sujet délicat à aborder car il n'existe pas un unique formalisme de représentation de connaissances accepté par tous. Les connaissances sur un domaine peuvent être exprimées de plusieurs façons. Par exemple, dans la Figure 3 nous présentons les différentes façons de représenter les connaissances sur un problème très spécifique : la fonction logique *Ou Exclusif* (XOR). Bien que la fonction XOR soit une fonction logique très simple, cette figure montre qu'il existe déjà différentes façons de la représenter.



Exemples

A	B	Xor
0	0	0
0	1	1
1	0	1
1	1	0

Si ((A ou B) et Non(A et B))
Alors Xor=1 Sinon Xor=0

Règle de Production

$$XOR = (A.\bar{B}) + (\bar{A}.B)$$

Formule Logique

Figure 3 - Différentes façons de représenter les connaissances sur la fonction logique XOR

La Figure 3 permet de montrer qu'on peut coder les connaissances à propos d'un sujet particulier de différentes façons. L'important est donc qu'on puisse représenter les connaissances de façon à pouvoir mieux les exploiter par la suite. Ce processus de conversion des connaissances sur un sujet dans un format particulier est ce que l'on appelle le "*processus de représentation de connaissances*". Une fois codifiées, elles pourront être manipulées et exploitées par des outils informatiques.

Nous montrons, dans la Figure 2, que les connaissances utilisées dans un système expert appliqué à un domaine spécifique peuvent être obtenues à partir de différentes sources, à savoir : l'explicitation directe par l'expert de ses connaissances ; l'interrogation de l'expert de façon à *extraire* ses connaissances; l'analyse et la représentation des connaissances contenues dans de livres, manuels, etc.; l'observation des résultats obtenus dans des expériences pratiques; et aussi, à travers l'emploi d'outils d'analyse, de traitement et de manipulation automatique de connaissances qui permettent d'induire ou de déduire de nouvelles connaissances.

Nous allons tout d'abord classer selon leur source les connaissances en deux grands groupes principaux :

1. **Connaissances Empiriques :** les connaissances empiriques, c'est-à-dire les connaissances expérimentales, sont représentées par l'ensemble des cas pratiques observés sur un sujet (ensemble d'exemples). Ce sont des connaissances "*pures*" qui n'ont pas été traitées, analysées ou modifiées. Ces connaissances représentent les résultats d'expériences ou les exemples de cas pratiques ; elles n'ont pas encore subi de transformations en vue d'obtenir une théorie plus générale sur le domaine. On peut dire que ce sont des connaissances de *bas niveau*. Ce type de connaissances peut être utilisé dans les systèmes experts à travers l'application de processus de raisonnement par induction (obtention de connaissances de plus haut niveau à partir des connaissances empiriques de base - la généralisation), ou à travers des processus de raisonnement par analogie (rapprochement des cas nouveaux aux cas anciens connus les plus proches - la similarité).

2. **Connaissances Théoriques :** les connaissances théoriques modélisent les connaissances sur un sujet à l'aide d'une théorie correspondant au problème posé. Elles sont des connaissances "*traitées*" qui ont été obtenues à partir de l'analyse des connaissances de base. Ce type de connaissances représente une généralisation du savoir. Ce sont des connaissances dites de *haut niveau*. Elles sont habituellement représentées par des structures symboliques, telles que les règles de production, les modèles mathématiques, les réseaux sémantiques, les objets structurés, les propositions.

Donc, les connaissances dont on peut disposer sur un domaine constituent une combinaison des connaissances empiriques et des connaissances théoriques disponibles. *Il faut souligner que* *ces deux types de connaissances forment deux ensembles ayant une intersection, mais qui peuvent aussi être totalement complémentaires* [TOW 91]. Il faut préciser aussi qu'habituellement les connaissances sur un domaine ne sont ni parfaitement correctes, ni complètes. Par conséquent, on doit envisager d'exploiter au maximum les connaissances disponibles tout en tirant profit de ces deux types de connaissances.

Les différents systèmes, à savoir les systèmes experts de première génération ou les systèmes d'apprentissage symbolique tels que les réseaux connexionnistes, les arbres d'induction ou les systèmes de raisonnement fondé sur des cas (voir la section suivante sur l'acquisition de connaissances), ne permettent pas en général de bien exploiter ces deux types de connaissances. Un des principaux buts de notre recherche dans cette thèse est de proposer des nouvelles solutions pour pouvoir travailler à la fois sur ces deux types de connaissances.

Les connaissances empiriques et théoriques doivent être converties dans une représentation compatible avec les outils informatiques de raisonnement automatique afin de pouvoir être exploitées. Parmi les différentes approches de représentation de connaissances, on trouve deux grands groupes opposés : *les approches connexionnistes* et *les approches symboliques*.

2.1.2.1 Approches Connexionnistes

Les approches connexionnistes, aussi dénommées approches sub-symboliques, sont inspirées des études de l'architecture du cerveau. La représentation des connaissances dans les réseaux connexionnistes est fondée, avec beaucoup de simplification, sur les structures de notre cerveau telles que les neurones et leurs connexions. Dans cette approche, toutes les connaissances sont représentées par des liaisons entre les unités (neurones artificiels) et leurs poids synaptiques (valeurs) associés. Les *Réseaux Connexionnistes* sont aussi appelés *Réseaux d'Automates*, *Réseaux Neuronaux Artificiels* (RNA) ou *Réseaux Neuromimétiques*. Même si l'étude des réseaux connexionnistes est encore jeune, on peut trouver des nombreux ouvrages décrivant le sujet [AMY 96, JOD 94a, JOD 94b, AMA 95, ADB 94, HER 94, NAD 93, BOU 91]. La Figure 4 montre un exemple de réseau connexionniste.

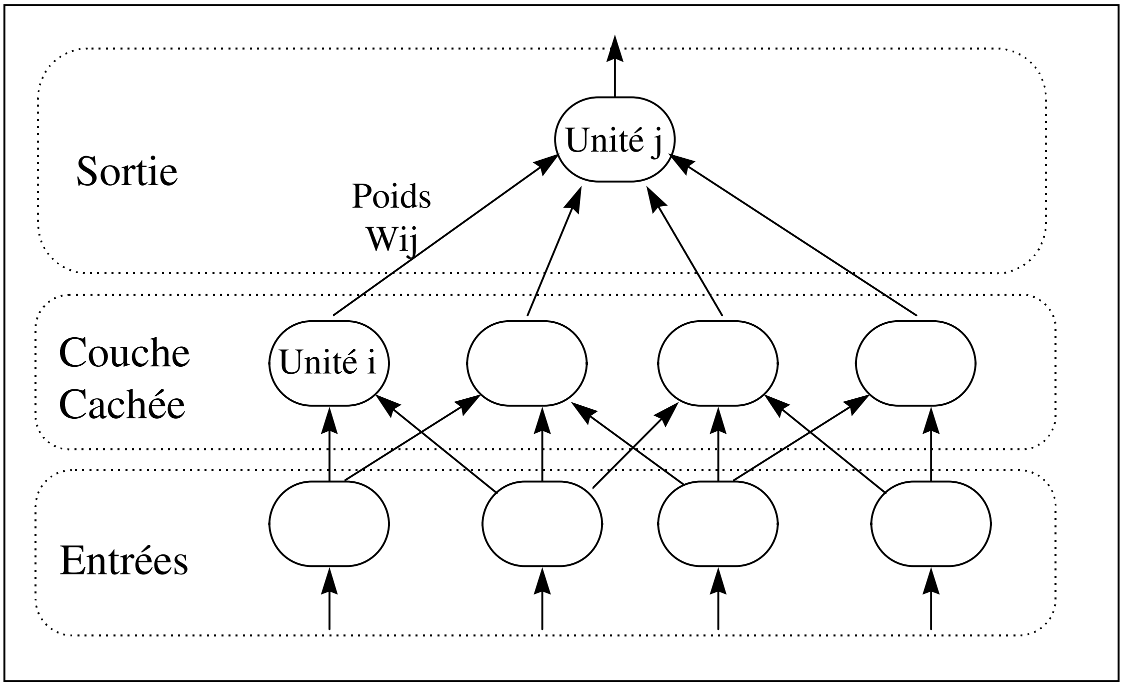


Figure 4 - Les Réseaux Connexionnistes

Les réseaux connexionnistes utilisent des méthodes d'apprentissage à partir d'exemples qui leur permettent d'ajuster leurs poids synaptiques. La modification des poids est faite de façon à modifier le comportement résultant de l'activation du réseau ; on cherche ainsi à induire (généraliser) des connaissances à partir d'une base d'apprentissage. Une bonne définition des réseaux connexionnistes est donnée par Giacometti [GIA 92] : il décrit ces réseaux comme capables de représenter le "savoir-faire" (comportement) dans un domaine d'application. Le "savoir-faire", qui représente les connaissances pratiques, apparaît en opposition au "savoir-que", qui lui représente les connaissances théoriques.

Lorsque les réseaux connexionnistes ont leurs connaissances codées sur la structure d'interconnexion et les poids des connexions, il est très difficile d'interpréter directement cette forme de codage des connaissances. Les connaissances ne forment qu'un ensemble de valeurs numériques décrivant les connexions et par conséquent le comportement d'activation du réseau. Dans la section 2.3, nous allons étudier les réseaux connexionnistes avec plus de détails.

2.1.2.2 Approches Symboliques

Les approches symboliques sont employées pour permettre de représenter les connaissances de type "savoir-que" [GIA 92]. Dans ce type d'approche, on va représenter les connaissances par des propositions qui vont exprimer le savoir d'un expert. Les propositions peuvent être représentées de différentes façons, dont la plus générale est le langage naturel. Donc, il s'agit de représenter les connaissances (savoir de l'expert) exprimées dans le langage naturel, par des propositions ou par des formules construites selon une syntaxe précise. Cette forme d'expression du savoir a été fortement influencée par la logique mathématique.

Le "savoir-que" d'un expert d'un domaine donné sera ainsi formalisé par un ensemble de propositions de même forme syntaxique, pouvant être ensuite manipulées selon des règles de la logique classique (règles d'inférence). Les connaissances représentées à travers des approches symboliques sont facilement interprétables étant donné le codage utilisé qui est assez proche du langage naturel humain. Nous allons nous concentrer ici sur les différents modes de représentation de connaissances proposés dans le cadre des approches de l'I.A., à savoir :

a) *Description de Cas Pratiques*
b) *Règles de Production et Formules Logiques*
c) *Réseaux Sémantiques*
d) *Objets Structurés et Frames*
e) *Méta-Connaissances*

2.1.2.2.a Description de Cas Pratiques

C'est la forme de représentation de connaissances la plus simple. Les connaissances sont décrites par une liste d'attributs et ses respectives valeurs associées, tel qu'on les a observée dans une expérience pratique. Le simple fait de sélectionner quelques attributs spécifiques et d'associer une classification au cas peut donner l'origine à une information plus structuré comme les règles de production. Exemple : Température = 36.5 °C, Age = 38 ans, Pression Artérielle = 80, ...

2.1.2.2.b Règles de Production et Formules Logiques

Les propositions, originellement en langage naturel, doivent être converties dans une syntaxe bien précise, pouvant donner ainsi l'origine à des règles de production et des formules logiques. Une règle de production est un quantum de connaissance, déclaratif et autonome, de la forme générale :

***Si** [conditions] **Alors** [conclusion]*

La partie *conditions* (antécédent) est constituée d'une formule logique qui doit être vérifiée pour que la règle s'applique. La partie *conclusion* (conséquent) correspond habituellement au déclenchement d'une action ou d'une autre règle. Les formules peuvent être des types : atomique, moléculaire ou généralisé [GIA 92] :

• Formules atomiques : utilisent des symboles atomiques et des relations unaires, binaires, ternaires, etc. Exemples : Froide(neige); Couleur(neige, blanche); Lancer(Philippe, neige, Denise);

• Formules moléculaires : peuvent être définies à partir des formules atomiques combinés à l'aide de connecteurs logiques tels que '$\wedge$' (ET logique), '$\vee$' (OU logique), '$\neg$' (NON logique). Exemple : Froide(neige) $\wedge$ Couleur(neige,blanche);

• Formules généralisées : peuvent être construites à partir des formules moléculaires. Il s'agit tout d'abord de substituer à des constantes (symboles atomiques) d'une formule moléculaire par des variables. Nous obtenons ainsi des fonctions sur les formules. Ces fonctions sont représentées par des quantificateurs tels que '$\exists$x' (il existe au moins une valeur possible pour la variable x), '$\forall$x' (pour toutes les valeurs possibles que la variable x peut assumer). Exemple : ($\exists$x)(Couleur(x,blanche))

Les règles de production sont classées selon leur pouvoir d'expressivité en règles d'ordre 0, d'ordre 0^+, ou de premier ordre (ordre 1). Les règles d'ordre 0 contiennent uniquement des formules atomiques ou moléculaires telles que nous les avons définies. Les règles d'ordre 0^+ contiennent en plus des formules du type '$x \in I$' (où 'x' est une variable décrivant une situation ou un événement, et 'I' est un ensemble de valeurs possibles, c.-à-d. un intervalle auquel x peut s'appliquer). Les règles de premier ordre sont des règles qui peuvent contenir des formules généralisées et par conséquence des quantificateurs. Ce dernier type de règle est aussi appelé de prédicat. Un type particulier de prédicat de premier ordre simplifié, nommé clause de Horn, est utilisé par des langages comme le Prolog.

Les règles de production sont ensuite traitées par les moteurs d'inférence. Ceux-ci permettent d'appliquer des méthodes de raisonnement formel déductif en utilisant les règles disponibles. Le calcul propositionnel est une des méthodes (système formel) qui permet de raisonner sur des formules moléculaires. Le calcul des prédicats à son tour est un système formel de raisonnement sur des prédicats (formules généralisées). Une description plus détaillée de ces méthodes de raisonnement formel déductif est proposée par Giacometti dans sa thèse [GIA 92].

Enfin, des règles de production et des méthodes de raisonnement automatique plus particulières ont été proposées afin d'augmenter le pouvoir de représentation et d'inférence des systèmes experts. On peut citer comme exemples [REY 97] :

- Les règles floues (e.g. règles de Mamdani ou de Sugeno [JAN 96]).

  Exemple : **Si** x $\in$ Fuzzy_Set_I **Alors** y = a.x + b ('a' et 'b' sont des valeurs constantes)

- Les règles obtenues à partir des réseaux connexionnistes (e.g. règles M-of-N [TOW 91]).

  Exemple : **Si** *2-parmi-4* (F_1(a), F_2(b),F_3(c),F_4(d))

  Alors *conclusion_est_vraie* **Sinon** *conclusion_est_fausse*

- Les règles bayésiennes (probabilistes) et les coefficients de vraisemblance (facteurs de certitude)

  Exemple : **Si** *possède _propriété_X* ($exemple_i$)

  Alors *probabilité_d'appartenir_à_classe_A* ($exemple_i$) = Valeur1 et

  probabilité_de_non_appartenir_à_classe_A ($exemple_i$) = Valeur2

- Les règles d'inférence graduelle.

  Exemple : (**Si**) plus *X* est un oiseau typique (**Alors**) plus on est certain que *X* vole

- Les règles de "haut niveau" (règles de modulation ou de contexte)

  Exemple : **Si** *Xi* est plus grand par rapport à *Xj* **Alors** moins important sera *Xj*

Ces règles de production jouent sur l'incertitude, sur les types de données utilisées (données discrètes ou continues), sur les relations entre données et valeurs constantes (appartenance à un intervalle, supérieur, inférieur), sur les relations entre une donnée et une autre et même sur la *force* des relations. Nous reviendrons sur ce sujet des règles de production particulières (règles de *haut niveau*) dans la section 5.3.

2.1.2.2.c Réseaux Sémantiques

Un réseau sémantique est un graphe étiqueté dans lequel les noeuds représentent des objets ou des concepts, et les arcs des relations entre eux. Issus des travaux de psychologie cognitive sur l'organisation de la mémoire (mécanismes d'association), les réseaux sémantiques ont donné lieu à de nombreux systèmes et langages de représentation des connaissances, comme par exemple : NETL et KL_ONE [FIN 79]. Ces réseaux permettent notamment d'exprimer une hiérarchie ou une taxonomie de concepts et les relations de proximité entre concepts [HAT 90, GIA 92]. Les réseaux sémantiques constituent donc un moyen de structuration des bases de connaissances.

Grâce au mécanisme d'héritage de propriétés, un noeud d'un réseau sémantique peut hériter des propriétés des noeuds qui le subsument (c.-à-d. qui sont plus hauts que lui dans la hiérarchie). Ce mécanisme constitue un mode économique de raisonnement, permettant notamment à un concept particulier de posséder automatiquement les propriétés des concepts plus généraux dont il est issu. On retrouve ce mécanisme dans les représentations par objets, actuellement beaucoup plus utilisées que les réseaux sémantiques.

2.1.2.2.d Objets Structurés et Frames

Les représentations à base d'objets structurés, telles que les *schémas, les frames, les scripts* ou *les scénarios*, permettent de regrouper ensemble des connaissances relatives à un objet, un concept, une situation ou un événement. A partir des travaux de psychologie cognitive sur l'organisation de la mémoire chez l'homme, la notion de schéma a été proposée comme modèle de représentation prototypique d'expériences passées mises à profit pour résoudre un problème nouveau. Ce concept à été repris en I.A. sous la forme de *frames*, proposés par Minsky [MIN 75] et de *scripts* ou *scénarios*, introduits par Schank [SCH 77]. Un exemple de langage basé sur les *frames* est le langage Shirka développé au LIFIA [REC 89].

< frame vin :
sorte de : boisson
appellation : (domaine : AOC/VDQ/vin de pays)
(défaut : vin de pays)
(si besoin : demander appellation)
nature : (domaine : sec/demi-sec/doux/liquoreux/corsé)
degré alcool : (intervalle : 9 à 15)
(défaut : 12)
producteur : (si besoin : trouver nom sur étiquette)
robe : ... >

Figure 5 - *Frame* décrivant le concept de vin

Frames et scénarios sont des entités de granularité importante regroupant de façon structurée l'ensemble des connaissances relatives à un objet, un concept ou une situation typique. Un *frame* est composé d'un ensemble d'attributs (*slots*) correspondant aux diverses notions relatives au concept représenté. A titre d'exemple, la Figure 5 donne une description possible du concept de vin [HAT 89].

2.1.2.2.e Méta-Connaissances

Une forme importante de connaissances est la méta-connaissance ou connaissance sur la connaissance [PIT 90]. Elle correspond au recul que l'on prend par rapport à un certain domaine d'activité et représente de ce fait souvent une part notable de l'expertise humaine. Elle intervient souvent, car elle est liée à la façon d'utiliser un ensemble de connaissances, aux stratégies de raisonnement et aussi à l'acquisition de nouvelles connaissances. Ce type de connaissances peut être représenté par des structures comme celles décrites dans les items ci-dessus ; cependant nous pensons qu'il mérite d'être classé dans un groupe à part, du fait de ses caractéristiques très particulières.

2.1.3 L'Acquisition de Connaissances

Le processus du développement d'un Système Expert se compose d'une étape d'acquisition de connaissances appelée Ingénierie de Connaissances (*Knowledge Engineering*). Elle est réalisée par l'ingénieur de connaissances, le responsable de l'obtention, de la vérification, de la validation et de la structuration des connaissances. Nous discuterons de la vérification et la validation de connaissances dans la section 4.5.

L'acquisition de connaissances est un processus de rassemblement des connaissances nécessaires à la résolution d'un problème lié à une application spécifique. Ce processus inclut la codification (structuration) des connaissances dans un formalisme adapté aux outils informatiques utilisés. Il y a plusieurs façons de coder les connaissances afin de créer la Base de Connaissances, comme on l'a vu dans la section 2.1.2. En ce qui concerne l'obtention des connaissances qui vont être placées dans cette base, on peut diviser les processus d'acquisition de connaissances en deux grands groupes : obtention par explicitation de connaissances (*Knowledge Elicitation*) et obtention par apprentissage automatique ou semi-automatique (*Machine Learning*).

L'explicitation de connaissances est réalisée par l'ingénieur de connaissances, qui va disposer de différents moyens pour la réalisation de cette tâche. Les connaissances sont obtenues à partir d'interviews avec des experts et d'informations disponibles dans des livres, manuels, rapports, etc. Cette approche est très intéressante, puisqu'on profite des connaissances déjà acquises et bien élaborées. Malheureusement, on constate un certain nombre d'inconvénients :

- Le processus d'extraction de connaissances d'un expert est extrêmement ennuyeux, il prend beaucoup de temps et sa mise en place présente de sérieuses difficultés ;

- Les experts humains utilisent souvent des activités mentales d'un niveau "*subcognitif*" pour résoudre les problèmes : ils ne savent pas verbaliser ce type de connaissances ;
- Parfois les experts ne sont même pas conscients d'avoir utilisé certaines connaissances dans la résolution d'un problème et par conséquent ne les explicitent pas.

L'explicitation de connaissances par des ingénieurs de connaissances et des experts a un problème très connu des systèmes experts : il s'agit du *goulot-d'étranglement du processus d'acquisition de connaissances (The Knowledge Acquisition Bootleneck)* [NIK 97, HAY 93]. Ce processus d'acquisition de connaissances, qui est indispensable au bon fonctionnement d'un système expert, est aussi le responsable des plus grands problèmes qui surviennent lors de la construction d'un tel système. Par conséquent, les chercheurs du domaine de l'I.A. ont beaucoup investi dans l'étude et implémentation de systèmes d'acquisition automatique de connaissances. Les recherches sur l'acquisition automatique de connaissances sont à l'origine des méthodes d'apprentissage automatique, l'un des domaines de l'I.A. en plus grande expansion actuellement.

L'apprentissage automatique est une partie très important de l'Intelligence Artificielle et doit être une des principales caractéristiques des systèmes intelligents. Une des meilleures définitions de l'apprentissage automatique est celle donné par H. Simon : "L'apprentissage dans un système est indiqué par les changements qu'il subit. Ces changements sont adaptatifs dans le sens où ils rendent possible au système de réaliser une même tâche, ou des tâches tirées d'une même population, d'une façon plus efficace et plus efficiente la prochaine fois qu'elle sera réalisée" [SIM 83]. Donc, l'apprentissage automatique se fait par des outils qui permettent d'acquérir, élargir et améliorer les connaissances disponibles au système. En général, l'apprentissage implique des processus d'adaptation et de modification des structures de contrôle et/ou de représentation de connaissances du système en question.

Etant donné que ces outils cherchent à implementer des méthodes d'apprentissage automatique, le problème du goulot-d'étranglement de l'acquisition de connaissances peut être réduit, voire parfois complètement résolu. Par conséquent, l'apprentissage automatique se présente comme une alternative prometteuse pour améliorer le processus d'acquisition de connaissances. Les systèmes experts de première génération, du fait des problèmes liés à l'acquisition de connaissances, ont évolué vers des systèmes dits de deuxième génération qui ont intégré, parmi d'autres techniques, des outils d'apprentissage automatique [NIK 97].

Les différentes méthodes d'apprentissage automatique sont classées en deux groupes : les méthodes d'apprentissage empirique (*Empirical Learning*) et les méthodes d'apprentissage fondées sur l'explication (*Explanation Based Learning*) [TOW 91].

2.1.3.1 Méthodes d'Apprentissage Empirique

Les méthodes d'apprentissage empirique sont fondées sur l'acquisition de connaissances à partir d'exemples. Elles incluent aussi des méthodes connues sous les noms d'*Instance based learning* ou *Exemplar based learning*. On peut citer comme exemples de méthodes d'apprentissage empirique les techniques suivantes : le raisonnement fondé sur des cas, les méthodes de construction de concepts et prototypes, les réseaux de neurones artificiels, les arbres de décision, les algorithmes génétiques d'induction de règles ou les méthodes du type ILP. Ces méthodes se divisent entre les méthodes d'apprentissage par analogie et les méthodes d'apprentissage par induction.

2.1.3.1.a Apprentissage par Analogie

Les approches fondées sur l'analogie essayent de faire le transfert des connaissances sur une tâche bien connue vers une autre moins connue. Ainsi, il est possible d'apprendre des nouveaux concepts ou de dériver des nouvelles solutions à partir de concepts et solutions similaires connues. Ainsi, deux notions deviennent très importants dans la définition de l'apprentissage par analogie : le *transfert* et la *similarité*. Ce type d'approche est aussi dénommée apprentissage fondé sur la similarité.

Les approches empiriques fondées sur l'analogie, telles que les systèmes de raisonnement fondé sur des cas (CBR), peuvent être considérées plutôt comme des méthodes de raisonnement que comme des méthodes d'apprentissage. Certains de ces systèmes implementent uniquement des fonctions de calcul de similarité, sans utiliser de mécanismes d'adaptation.

2.1.3.1.b Apprentissage par Induction

L'apprentissage par induction reste toujours une des principales méthodes étudiées dans le domaine de l'apprentissage automatique. Dans cette approche, on cherche à acquérir des règles générales qui représentent les connaissances obtenues à partir d'exemples. Les règles ainsi obtenues peuvent être représentées d'une façon explicite (facilement interprétables) ou d'une façon implicite avec un codage qui n'est pas toujours facile à interpréter. L'algorithme d'apprentissage par induction reçoit un ensemble d'exemples d'apprentissage et doit produire des règles de classification, permettant de classer les nouveaux exemples. Le processus d'apprentissage cherche a créer une représentation plus générale des exemples, selon une *méthode de généralisation de connaissances*. Cet type de méthodes est aussi appelé apprentissage de concepts ou bien acquisition de concepts. Parmi les approches d'apprentissage empirique par induction les plus connues, on trouve les réseaux de neurones artificiels et les arbres de décision.

L'algorithme d'apprentissage par induction peut fonctionner de façon *supervisée* ou *non-supervisée* :

- Apprentissage supervisé (*supervised learning*) : les exemples d'apprentissage sont étiquetés afin d'identifier la classe à laquelle ils appartiennent. Le but de l'algorithme de classification est de correctement classifier les nouveaux exemples dans les classes définies dans la phase d'apprentissage.

- Apprentissage non-supervisé (*unsupervised learning, clustering, discovery*) : l'algorithme d'apprentissage cherche à trouver des régularités dans une collection d'exemples, puisque dans ce type d'apprentissage on ne connaît pas la classe à laquelle les exemples d'apprentissage appartiennent. Une technique employée consiste à implémenter des algorithmes pour rapprocher les exemples les plus similaires et éloigner ceux qui ont le moins de caractéristiques communes. Ces groupes d'exemples similaires sont parfois appelés des *prototypes*.

2.1.3.2 Méthodes d'Apprentissage fondées sur l'Explication

Les méthodes inductives, telles que les réseaux de neurones ou les arbres de décision, ont besoin d'un nombre significatif d'exemples pour pouvoir bien généraliser les connaissances (induire des règles ou des concepts). Ceci restreint les possibilités d'application de ces méthodes, puisqu'on n'a pas toujours une base d'exemples assez grande et complète sur le domaine traité. Les méthodes d'apprentissage fondées sur l'explication (*Explanation-Based Methods - EBL*) [MIT 93, NIL 97] utilisent des connaissances préexistantes et un raisonnement déductif pour augmenter l'information fourni par des ensembles d'exemples. Ces méthodes sont connues sous le nom d'apprentissage par analyse (*Analytical Learning*) [MIT 97].

Dans les méthodes EBL, les connaissances sont dérivées à partir d'un simple cas par explication des raisons pour lesquelles il représente un exemple du concept appris. La méthode EBL utilise les connaissances préexistantes pour analyser, ou expliquer, comment chaque exemple observé lors de l'apprentissage satisfait les concepts existants. Ensuite, cette explication est utilisée pour différencier les attributs pertinents de l'exemple d'apprentissage de ceux qui ne le sont pas. De cette façon, l'exemple pourra être généralisé par un raisonnement logique, à la place des raisonnements statistiques souvent utilisés par les autres méthodes. Ces méthodes servent donc à améliorer les performances du système, grâce à des traitements qui rendent l'utilisation des connaissances plus efficace. *Les méthodes d'apprentissage fondées sur l'explication considèrent que les connaissances préexistantes sont à la fois correctes et complètes.*

2.1.3.3 Considérations Finales sur l'Acquisition de Connaissances

Nous avons vu dans cette section (2.1.3) les méthodes d'acquisition de connaissances classées selon les techniques d'obtention des connaissance (explicitation ou apprentissage automatique) et nous avons présenté les différentes classes de méthodes d'apprentissage automatique selon le type prédominant de connaissances employé dans l'apprentissage : les méthodes empiriques (exemples) et les méthodes d'explication (théorie). Dans les sections suivantes, nous allons présenter certaines méthodes d'apprentissage automatique de façon plus détaillée. Elles sont divisées en deux grands groupes selon le type de représentation des connaissances utilisé : les approches symboliques et les approches connexionnistes.

2.2 APPRENTISSAGE AUTOMATIQUE - APPROCHES SYMBOLIQUES

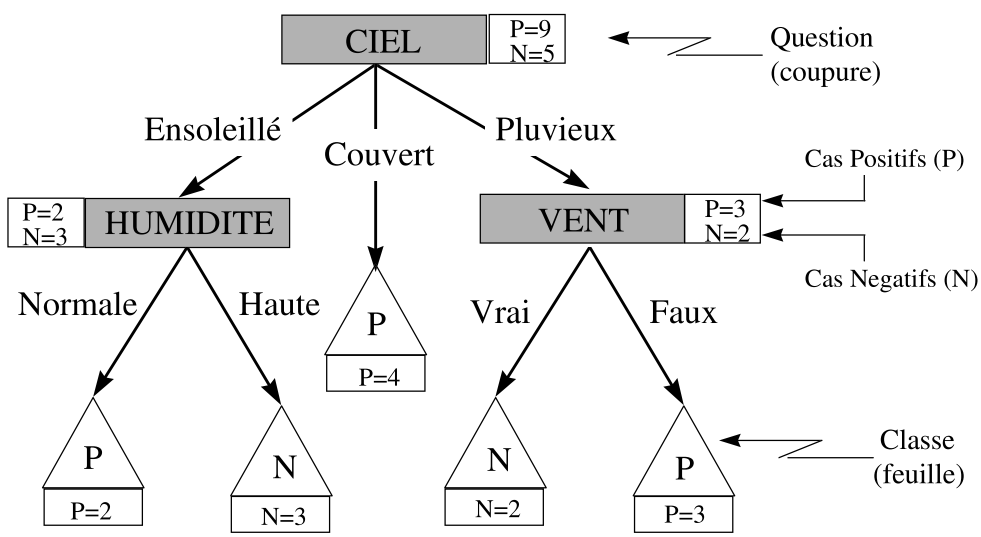
Nous allons présenter dans cette section des méthodes d'apprentissage automatique symbolique, très connues dans le domaine de l'I.A. Notre but est de mieux connaître les algorithmes, leurs avantages ainsi que leurs problèmes particuliers, afin de pouvoir proposer par la suite des solutions plus performantes. Dans la section 2.4, nous allons montrer que ces différentes méthodes peuvent être combinées afin de donner naissance à des systèmes hybrides. Nous allons tout d'abord nous concentrer sur trois méthodes principales : les arbres de décision, les algorithmes génétiques d'induction de règles et les systèmes de type CBR.

2.2.1 Arbres de Décision

Les arbres de décision sont composés d'une structure hiérarchique en forme d'arbre. Cette structure est construite grâce à des méthodes d'apprentissage par induction à partir d'exemples. L'arbre ainsi obtenu représente une fonction qui fait la classification d'exemples, en s'appuyant sur les connaissances induites à partir d'une base d'apprentissage. En raison de cela, ils sont aussi appelés arbres d'induction (*Induction Decision Trees*). Une définition un peu plus formelle des arbres de décision est la suivante : un arbre de décision est un graphe orienté, sans cycles, dont les noeuds portent une question, les arcs des réponses, et les feuilles des conclusions, ou des classes terminales.

2.2.1.1 L'Apprentissage dans les Arbres de Décision

Traditionnellement, un arbre de décision se construit à partir d'un ensemble d'apprentissage par raffinements de sa structure. Un ensemble de questions sur les attributs est construit afin de partitionner l'ensemble d'apprentissage en sous-ensembles qui deviennent de plus en plus petits jusqu'à ne contenir à la fin que des observations relatives à une seule classe. Les résultats des tests forment les branches de l'arbre et chaque sous-ensemble en forme les feuilles. Le classement d'un nouvel exemple se fait en parcourant un chemin qui part de la racine pour aboutir à une feuille : l'exemple appartient à la classe qui correspond aux exemples de la feuille. La Figure 6 donne un exemple d'arbre de décision pour le classement d'un ensemble de cas, avec un test d'appartenance à une classe. Dans ce cas particulier, les cas dits *positifs* sont ceux qui appartiennent à la classe et les cas dits *négatifs* sont ceux qui n'y appartiennent pas.



IF ((CIEL=Ensoleillé ***Et*** HUMIDITE=Normale) ***Ou***
(CIEL=Couvert) ***Ou***
(CIEL=Pluvieux et VENT=Faux))
ALORS Classe = P

Figure 6 - Exemple d'Arbre de Décision Simple

L'apprentissage est fait à partir d'une base d'exemples qui possèdent un certain nombre d'attributs significatifs, e.g. la température, le vent, l'humidité, etc. Chaque exemple associe des valeurs particulières à chaque attribut, et comme cette méthode est une *méthode d'apprentissage supervisé*, chaque exemple est associé à une classe particulière. Nous présentons ci-dessous une base d'exemples afin de mieux expliquer le processus de construction d'un arbre (ce tableau est extrait de l'article de Quinlan [QUI 86]).

NUMERO	*CIEL*	*TEMPERATURE*	*HUMIDITE*	*VENT*	*CLASSE*
1	ensoleillé	élevé	forte	non	N
2	ensoleillé	élevé	forte	oui	N
3	couvert	élevé	forte	non	P
4	pluvieux	moyenne	forte	non	P
5	pluvieux	basse	normale	non	P
6	pluvieux	basse	normale	oui	N
7	couvert	basse	normale	oui	P
8	ensoleillé	moyenne	forte	non	N
9	ensoleillé	basse	normale	non	P
10	pluvieux	moyenne	normale	non	P
11	ensoleillé	moyenne	normale	oui	P
12	couvert	moyenne	forte	oui	P
13	couvert	élevé	normale	non	P
14	pluvieux	moyenne	forte	oui	N

Tableau 2 - Ensemble d'Apprentissage : Conditions Météorologiques

Le principe de construction des arbres est le suivant : on choisit un attribut parmi les attributs non sélectionnés, et on crée un noeud portant un test sur cet attribut. Pour chaque classe d'équivalence ainsi induite, on opère le traitement suivant : si tous les exemples de cette classe d'équivalence appartiennent à la même classe (les classes météorologiques décrites dans le tableau ci-dessus), alors on crée une feuille correspondante à cette classe, reliée au test précédent par un arc étiqueté par la valeur de l'attribut correspondant ; si tous les exemples de la classe d'équivalence considérée ne sont pas dans la même classe, alors on réitère ce processus en enlevant l'attribut précédemment considéré des attributs à sélectionner.

Par exemple, on peut construire de la manière suivante l'arbre de la Figure 6 : le premier attribut sélectionné est le 'ciel'; cet attribut sépare la base en trois ensembles : "ciel ensoleillé", "ciel pluvieux" et "ciel couvert". Ensuite, pour les exemples qui ont l'attribut "ciel ensoleillé", on prend un autre attribut qui va permettre de les distinguer, par exemple l'humidité. Une fois que tous les exemples qui ont l'attribut "humidité forte" appartiennent à la même classe et qu'il en est de même avec ceux qui ont l'attribut "humidité normale", on peut arrêter le processus de division des branches de l'arbre à ce niveau. Il faut alors terminer la construction de l'arbre pour les autres branches qui n'ont pas encore été traités jusqu'à la fin.

On peut remarquer que l'ordre dans lequel ces attributs sont sélectionnés est primordial pour la construction de l'arbre. On peut avoir différents arbres résultant de l'apprentissage d'une même base d'exemple, comme le montre la Figure 7. En effet, si l'on commence par prendre la 'température', alors l'arbre considéré sera plus grand. On suppose que les arbres qui généralisent le mieux sont les arbres de petite taille, puisque ils vont tout d'abord sélectionner les décisions plus importantes et qui séparent le plus grand nombre de cas différents. De plus, la préférence pour la construction d'arbres plus simples est une application du principe du "rasoir d'Occam" [BLU 87].

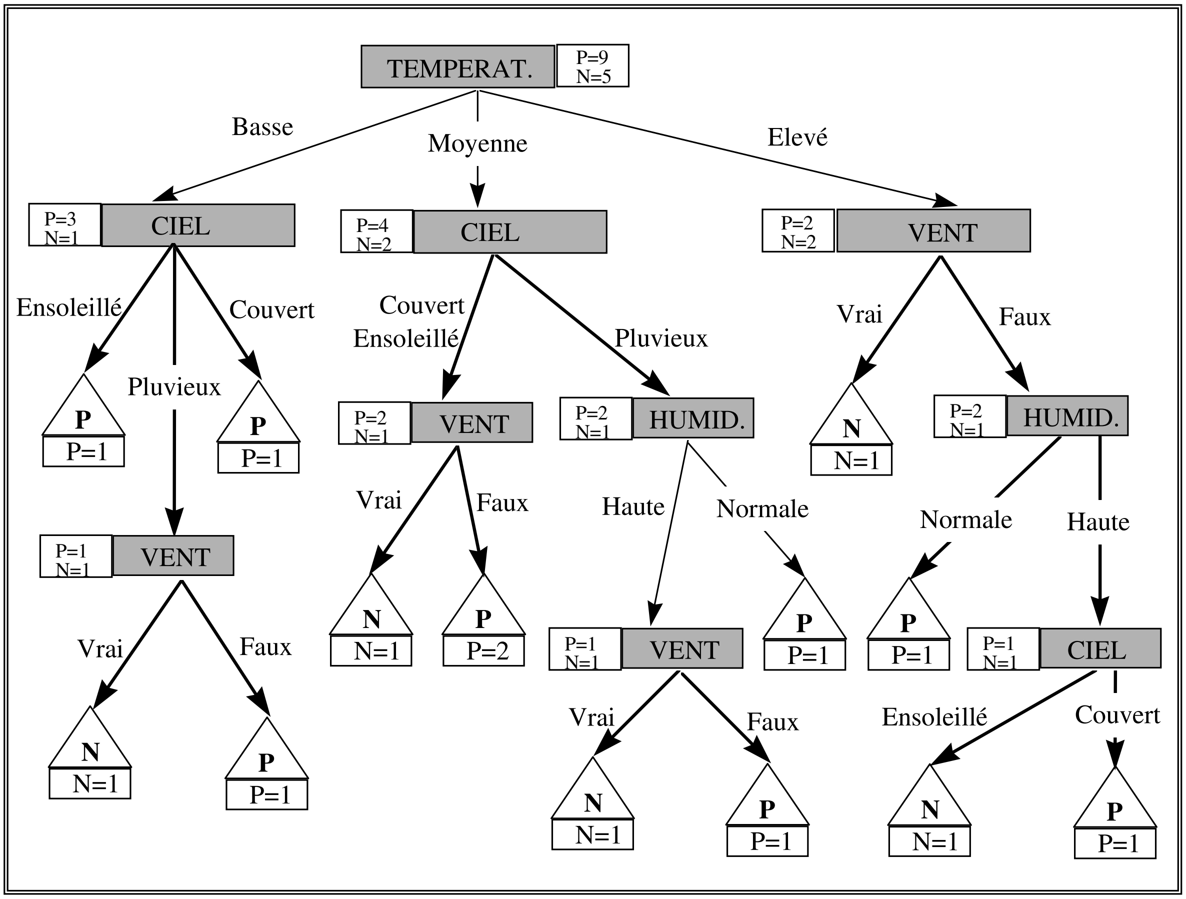


Figure 7 - Exemple d'Arbre de Décision Complexe

Quinlan a proposé une méthode pour traiter ce problème du choix de l'attribut. Il utilise une technique fondée sur la théorie de l'information de Shannon [QUI 86]. L'idée consiste à appliquer une méthode qui permet de maximiser le gain d'information apporté par chaque test.

2.2.1.2 Les Systèmes fondés sur les Arbres de Décision

Parmi les méthodes d'apprentissage fondées sur les arbres de décision les plus connues, on trouve l'algorithme ID3 [QUI 79, QUI 86] et la méthode CART [BRE 84]. Ces systèmes sont assez semblables ; ils ont été développés par deux groupes de recherche séparés et presque à la même époque. La principale différence entre ces deux systèmes réside dans le choix de la mesure utilisée pour la sélection des attributs pendant la construction de l'arbre. Cette mesure est généralement fondée sur la théorie de l'information de Shannon (entropie et gain d'information - utilisées dans ID3).

La méthode ID3 est à l'origine de plusieurs autres systèmes. L'un des plus connus est le système C4.5, développé plus récemment par Quinlan [QUI 93]. Les logiciels qui implementent ce système peuvent être obtenus avec l'ouvrage de Quinlan, ce qui a fait de C4.5 une référence de base dans les études sur les arbres de décision. A l'origine du système C4.5 se trouve la volonté de résoudre les différents problèmes rencontrés dans son prédécesseur, l'algorithme ID3. Ces problèmes sont assez caractéristiques des arbres de décision en général, à savoir :

A) Les premiers systèmes ont été conçus pour traiter des attributs avec des valeurs nominales et discrètes (*variables qualitatives*) et ne pouvaient pas traiter des attributs avec des valeurs continues (*variables quantitatives*). Il manquait des méthodes bien adaptées pour trouver les bonnes questions à poser sur les valeurs continues et qui seront postérieurement associées aux noeuds de l'arbre ;

B) Le choix du meilleur attribut qui va diviser les exemples présente lui aussi certains problèmes. Comment peut-on nous assurer que la méthode employée va nous amener à la construction de l'arbre le plus simple possible ? Malheureusement, on peut pas être sûr de l'obtenir, puisque ce type de problème devient intraitable d'un point de vue calculatoire [SHA 90]. De plus, il n'est pas difficile de trouver des exemples pour lesquels une variable a un très grand pouvoir de discrimination, mais n'aide pas beaucoup à la solution du problème (e.g.: l'âge ou le sexe d'une personne permet de discriminer les gens d'une façon très efficiente ; toutefois est-elle une bonne variable par rapport à tous les types de problèmes de classification ?) ;

C) Parfois on ne connaît pas toutes les valeurs de chacun des attributs considérés pour la classification. Le fait que l'on doive travailler sur des domaines représentés par des information incomplètes ouvre la problématique du traitement des *attributs avec valeurs manquantes* ;

D) Les arbres de décision qui résultent de l'algorithme décrit ci-dessus souffrent du problème de surapprentissage (*overtrainning/ovefitting*). Le fait que la construction de l'arbre de décision ne s'arrête que lorsque tous les exemples de la feuille appartiennent à une même classe, suppose des ensembles d'apprentissage contenant uniquement des exemples "parfaits" (ils doivent être corrects et doivent bien représenter tout l'espace des entrées). L'arbre de décision finit par trop se spécialiser dans les exemples d'apprentissage, ce qui peut poser de graves problèmes au niveau de la généralisation aux nouveaux cas. Souvent, les bases d'apprentissage sont incomplètes et ont des exemples incorrects, et donc, cette méthode d'apprentissage ne fonctionnera pas correctement ;

E) Sur un arbre de décision, les connaissances restent "cryptées". Bien qu'il s'agisse d'une représentation symbolique des connaissances, elle peut être parfois un peu difficile à lire et à interpréter ;

F) Les arbres de décision, tels que ID3, doivent être complètement reconstruits pour prendre en compte un nouvel exemple ajouté à la base d'apprentissage. Ce type d'arbre n'est pas incrémental du point de vue de l'acquisition de données, même si sa structure est construite d'une façon incrémentale (par ajout de noeuds). Il faut recommencer tout l'apprentissage pour prendre en compte un nouvel exemple ;

G) Malgré le fait de que ces méthodes cherchent à construire des arbres simples, certaines branches peuvent être répliquées. La structure en arbre n'est pas toujours la forme la plus simple et économique pour représenter les informations. Les graphes de décision restent une alternative à la structuration en forme d'arbres ;

H) Les arbres de décision, comme leur nom l'indique, servent à obtenir une classification des donnés et ont donc une ou plusieurs sorties binaires. On ne peut pas avoir une sortie qui représente une valeur continue. Ce type d'arbres de décision n'est pas utilisable pour l'approximation de fonctions (régression) ;

I) Enfin, ces arbres de décision n'exploitent pas de méthodes permettant l'utilisation des connaissances théoriques disponibles sur le problème. Dans leur forme initiale, les arbres d'induction travaillent uniquement sur les connaissances empiriques.

Plusieurs systèmes fondés sur les arbres de décision ont pris en compte ces différentes remarques décrites ci-dessus, en essayant d'apporter des solutions plus performantes :

- **C4.5** : c'est un système dérivé de l'ID3. Il présente des propositions pour traiter et améliorer les items A (discrétisation des variables quantitatives), B (prise en compte des coûts associés au choix de chaque attribut), C (prise en compte par la fonction de sélection d'attributs), D (élagage de l'arbre à la fin du processus d'apprentissage, à partir d'un ensemble d'exemples de test de généralisation) et E (explicitation de règles symboliques du type Si-Alors à partir des arbres de décision) [QUI 93, QUI 96];
- **Assistant Professional** : il utilise une autre fonction de choix d'attributs qui permet l'élagage de l'arbre de décision (*post-pruning*), et implemente une méthode de binarisation des attributs avec des valeurs continues [CES 87, THR 91];
- **ID5R** : Il s'agit d'une méthode de construction incrémentale d'arbres de décision qui permet d'apprendre des nouveaux exemples sans avoir besoin de recommencer tout l'apprentissage [UTG 89];
- **ITI** : système d'apprentissage incrémental d'arbres de décision, développé à partir de l'ID5R. Il apporte des améliorations par rapport à l'utilisation de variables continues et de valeurs manquantes, et il permet aussi d'élaguer les arbres (*virtual pruning process*) [UTG 95];
- **IDL** : Il s'agit d'un algorithme de construction incrémental d'arbres de décision fondé sur la méthode ID5R [VAN 90, THR 91];
- **PRISM** : méthode d'apprentissage fondée sur l'algorithme ID3 de Quinlan. Il se distingue d'ID3 par le type de méthode employé pour la sélection des attributs discriminants [CEN 87, THR 91];
- **SIPINA** : système de construction de graphes de décision. Il utilise une fonction particulière de sélection d'attributs, implemente une méthode de discrétisation de

variables continues, implemente une méthode qui évite le surappentissage et permet aussi l'explicitation de règles représentées dans les graphes de décision [ZIG 92, ZIG 96b, ZIG 96c].

2.2.1.3 Discussion sur les Arbres de Décision

Les arbres de décisions sont largement utilisés, car ce type d'approche est assez stable et donne des bons résultats dans un grand nombre d'applications. Cependant, dans certains types d'applications, les arbres de décision donnent des résultats décevants. Les principaux problèmes sont ceux énumérés dans la section précédente en relation avec l'algorithme ID3.

Nous avons présenté toute une série de problèmes liées aux algorithmes comme l'ID3. Plusieurs systèmes ont été conçus afin de résoudre ou de minimiser ces problèmes. Cependant, certains aspects sont directement liés à la façon dont les arbres de décision sont construits et à leurs propriétés. Les techniques que nous avons décrites apportent des améliorations, mais les problèmes de base demeurent, à savoir :

- La difficulté de manipulation de variables quantitatives (valeurs continues). La discrétisation des données reste une solution à considérer, mais elle ne résout pas tous les problèmes de traitement d'informations non-symboliques ;

- Le choix des attributs à considérer dans les divisions (noeuds de l'arbre) ne garantit pas une solution toujours parfaite. De plus, la méthode la plus utilisée, fondée sur la théorie de l'information de Shannon, impose de fortes contraintes aux méthodes incrémentales (on est obligé de tout reconstruire) ;

- Les méthodes incrémentales, telles que l'ID5R et l'ITI, restent des méthodes assez intéressantes à étudier, mais elles imposent d'autres contraintes au niveau de la construction de l'arbre. Dans ces deux cas, le processus d'élagage semble être difficile à implementer et la complexité de la tâche de construction et de maintenance des arbres augmente ;

- Les arbres de décision n'utilisent pas de connaissances théoriques disponibles sur le problème posé. Ils exploitent uniquement des connaissances empiriques.

Enfin, la méthode de représentation des connaissances utilisée dans les arbres de décision est très dépendante du type des questions associées aux noeuds. Nous verrons dans la section 5.3 que les arbres de décision (du moins ceux qu'on a présenté ici) ne sont pas capables de traiter correctement des problèmes qui mettent en relation directe deux variables avec des valeurs continues.

2.2.2 Les Algorithmes Génétiques

La notion d'Algorithme Génétique (AG) a été introduite par John H. Holland en 1975 [HOL 75] et a été considérablement développée au cours des années quatre-vingt (cf. les ouvrages [DAV 91, MIC 92, GOL 94]). Plus récemment, une autre voie de recherche, la programmation génétique [KOZ 92], a été dérivée des AG. Nous allons ici nous focaliser sur les algorithmes génétiques et leur application à l'apprentissage de règles par induction à partir d'une base d'exemples.

2.2.2.1 Principes de Base

Les algorithmes génétiques sont des algorithmes d'exploration fondés sur les mécanismes de la sélection naturelle et de la génétique. Ils utilisent à la fois les principes de survie des structures les mieux adaptées, et de modification pseudo-aléatoire d'informations, pour former un algorithme d'exploration qui possède des caractéristiques intéressantes. Les AG ont une capacité de s'adapter et de se transformer sans avoir à imposer beaucoup de restrictions sur la recherche d'une solution, ce qui leur permet de mieux explorer le problème et de trouver des solutions originales.

Le principe de l'algorithme est simple et repose sur un codage des problèmes et de leurs solutions sous la forme de chaînes d'éléments de base. Les chaîne peuvent être rompues entre chaque élément de base, à l'image des chromosomes, qui eux constituent de véritables listes de caractéristiques d'un individu. Le codage prend habituellement la forme d'une chaîne binaire très structurée, de longueur fixe ou variable selon le type de problème. On génère tout d'abord une *population* de solutions potentielles à un problème donné, sous la forme de telles chînes, puis on sélectionne, au moyen d'une mesure d'ajustement, les éléments de la population qui satisfont au mieux les contraintes (ou caractéristiques) de la solution recherchée. *Des opérateurs génétiques* sont ensuite appliqués à cette population de manière à obtenir une nouvelle population, possédant, dans son ensemble, de meilleures solutions que la précédente génération. On réitère le processus sur plusieurs générations, jusqu'à l'obtention d'une génération de solutions qui satisfait les critères de qualité imposés, et qui est beaucoup plus adapté au traitement du problème en question. Il suffit alors de choisir parmi la population la meilleure solution.

Les AG mettent en œuvre des mécanismes assez simples pour l'évolution vers la solution. Ils ne font que des copies et des échanges de morceaux de chaînes, en plus des opérations d'évaluation et de sélection des éléments d'une population. Les manipulations des chaînes sont faites en utilisant trois opérateurs basiques : la *reproduction*, le *croisement* et la *mutation*.

2.2.2.2 Induction de Règles à partir d'Exemples

L'utilisation des algorithmes génétiques dans les systèmes d'apprentissage automatique s'est largement développé à partir des années quatre-vingt. Plusieurs applications des AG à la construction de classifieurs ont été déjà expérimentées avec des bons résultats. La première application a été réalisée par celui qui est à l'origine même des AG, J. Holland [HOL 86, HOL 87]. Plusieurs autres expérimentations ont fait suite à ces premiers travaux [DAV 91, GOL 89, DEJ 88, DEJ 90, DEJ 93b, GRE 93, EIC 96].

Pour pouvoir réaliser un apprentissage de règles avec des AG, il est nécessaire d'avoir une représentation interne de l'espace de recherche. Le codage utilisé par les AG est donc très important vis-à-vis de l'implémentation des outils d'apprentissage automatique. Le choix d'un type de codage spécifique doit essayer de garder les propriétés des AG, sans imposer des restrictions qui peuvent nuire ou restreindre la convergence de l'algorithme vers une bonne solution.

Deux approches assez connues pour générer automatiquement un classifieur à partir d'un AG sont : l'approche de Pitt et l'approche de Michigan [DEJ 93a]. L'approche de Pitt, développée initialement à l'Université de Pittsburgh, postule qu'un élément (chromosome) correspond à une base de règles et qu'une population de chromosomes est un ensemble de bases de règles. L'autre approche, celle de Michigan, à été développée à l'Université de Michigan par l'équipe de J. Holland. Cette approche postule qu'un chromosome représente une règle et qu'une population correspond donc à une base de règles. L'algorithme, au cours de ses itérations cherche à construire et améliorer une seule base de connaissances.

2.2.2.3 Discussion sur les Algorithmes Génétiques

Les algorithmes génétiques se présentent comme une approche très intéressante. Ils sont capables d'exploiter l'espace de recherche sans avoir à imposer de restrictions (*biais*) liées au type d'algorithme choisi, ce qui n'est pas toujours le cas des autres algorithmes d'apprentissage automatique. Par contre, les choix faits au niveau du codage employé, du type de règles manipulées, ou du type de méthodes de sélection et de transformation utilisées, vont beaucoup influencer l'apprentissage. Une des tâches les plus difficiles va être la définition de ces choix d'implémentation des AG. Ce type de problème est d'autant plus délicat qu'il n'existe pas de méthodologie générale à suivre.

La difficulté du choix d'un codage et d'une bonne fonction de sélection et adaptation font des AG une technique très particulière qui doit être bien maîtrisée pour donner des bons résultats. Dans le cas contraire, ces algorithmes n'arriveront pas à trouver de bonnes solutions aux problèmes posés. Le manque d'une garantie de convergence vers une bonne solution s'agit de l'un des principaux désavantages des AG. De plus, les AG sont des algorithmes très lourds à exécuter, d'où l'intérêt d'exploiter le parallélisme dans ce type d'approche.

Comme d'autres algorithmes d'optimisation, ils connaissent les problèmes liés à la convergence et au blocage dans des *minima locaux*. Par contre, si les fonctions de mutation, de reproduction et de croisement sont bien définies, les AG peuvent constituer une solution assez intéressante pour s'échapper des minima locaux. Un autre avantage des AG est le fait qu'ils admettent l'introduction de connaissances théoriques préexistantes dans la population initiale à traiter. Donc, cette population initiale nous permet d'avoir un point de départ pour l'apprentissage qui peut être plus proche de la solution finale recherché.

Pour conclure cette discussion, il nous reste à souligner que la majorité des AG sont des approches qui manipulent des informations purement symboliques, d'où la difficulté à travailler avec des variables continues et des données approximatives. C'est le type de représentation des règles symboliques qui va définir le pouvoir d'un tel algorithme d'apprentissage. On trouve dans la littérature des travaux qui cherchent à surmonter ces limitations en utilisant des règles plus robustes, comme le système DELVAUX, qui utilise des règles bayésiennes [EIC 96].

2.2.3 Le Raisonnement Fondé sur des Cas - CBR

Le raisonnement fondé sur des cas (CBR) recouvre un ensemble de méthodes de résolution de problèmes qui exploite les expériences passées, plutôt que les connaissances générales d'un niveau supérieur, telles que les règles de production. Les CBR sont des méthodes d'apprentissage totalement fondées sur les connaissances empiriques, au lieu de faire usage des connaissances théoriques d'un domaine.

2.2.3.1 Principes de Base

Un système CBR est capable d'utiliser la connaissance spécifique contenue dans son expérience passée pour résoudre les nouveaux problèmes. Cet expérience est représentée normalement sous la forme de cas. Ces cas, qui ont été corrigés et assignés par l'expert aux classes auxquelles ils appartiennent, constituent ainsi la mémoire d'un système CBR.

Quand un nouveau problème est présenté à un système CBR, il va *se rappeler* des cas passés stockés dans son mémoire, similaires au problème courant. Ensuite, le système adapte la meilleure solution mémorisée et la transfère au problème actuel. Le cas nouveau, qui a été traité par le système et reconnu, peut être à son tour mémorisé et donc ajouté comme une nouvelle expérience du système [MAL 96, KOL 93].

En général, un système de raisonnement fondé sur des cas contient les phases suivantes :

1. *Remémoration* des cas les plus similaires par rapport au cas posé en question;
2. *Réutilisation* de la connaissance du(ou des) cas remémore(s) pour la résolution du problème;
3. *Révision* de la solution donné afin de la valider;
4. *Mémorisation* de cette nouvelle expérience, pour une utilisation future.

2.2.3.2 Discussion sur le Raisonnement Fondé sur des Cas

Les systèmes de raisonnement fondé sur des cas font appel à des algorithmes d'apprentissage dits *paresseux* (*lazy learning* [AHA 97]). Ils diffèrent des autres algorithmes par le fait qu'ils retardent le traitement des informations, c'est-à-dire qu'ils ont un faible coût calculatoire pendant la phase d'apprentissage et des calculs plus intenses pendant la phase de test (reconnaissance, *recall*). Les systèmes CBR n'essaient pas de traiter les exemples fournis, alors que les autres algorithmes d'apprentissage font une compilation des exemples d'apprentissage et les remplacent par des abstractions concises (e.g.: arbres de décision, ensembles de règles).

Les CBR sont des systèmes purement fondés sur les connaissances empiriques et ne permettent pas l'utilisation de connaissances théoriques. Donc, on ne peut pas profiter des connaissances disponibles sur le domaine d'application sauf si celles-ci sont représentées par des cas pratiques. Toutefois, on trouve dans la littérature des méthodes alternatives que permettent de contourner ce problème d'une manière assez simpliste : la solution consiste en transformer les règles symboliques en exemples que les représentent (prototypes des concepts théoriques) [ALT 97,ORS 95].

Une des particularités de ce type d'approche est le rôle essentiel joué par la fonction de similarité employée. Elle va déterminer le bon fonctionnement du système, car il est nécessaire de bien retrouver les bons exemples proches de l'exemple traité. La fonction de remémoration la plus simple et une des plus utilisées est la méthode des K-Plus-Poches-Voisins (*K-PPV* ou *K-Nearest-Neighbor*), avec une mesure de similarité fondée sur la distance euclidienne. Ce type de mesure peut bien fonctionner avec des variables quantitatives, mais est mal adapté quand il s'agit de mesurer la distance entre deux valeurs d'une variable qualitative (e.g. peut-on mesurer la distance entre 'lundi' et 'jeudi', ou entre 'bleu' et 'rouge' ?).

L'une des avantages d'une telle approche est le traitement assez simple des cas particuliers et des exceptions, ce qui n'est pas toujours évident pour un système d'apprentissage inductif. Enfin, ce type de système n'est pas très performant en ce qui concerne l'explication de la solution trouvée puisqu'au mieux il va donner la liste des cas les plus semblables au cas présenté sans faire une analyse plus profonde de la solution fournie.

2.3 APPRENTISSAGE AUTOMATIQUE - APPROCHES CONNEXIONNISTES

Les réseaux connexionnistes sont des assemblages fortement connectés d'unités de calcul, les *neurones formels*. Ces derniers ont pour origine un modèle du *neurone biologique*, dont ils ne retiennent d'ailleurs qu'une vision fort simplifiée (voir Figure 8 et Figure 9). Le neurone, comme toute cellule, est composé d'un corps (ou *soma*), qui contient son noyau et où se déroulent les activités propres à sa vie cellulaire. Cependant, il est aussi doté d'un *axone* et de *dendrites*, structures spécialisées dans la communication avec les autres neurones. Cette communication entre cellules nerveuses s'effectue via des impulsions nerveuses. Les impulsions sont générées à l'extrémité somatique de l'axone et vont vers les terminaisons axonales. Là, elles affecteront tous les neurones reliés au neurone générateur, par l'intermédiaire de jonctions entre les terminaisons axonales et les autres cellules. Cette jonction est appelée *synapse*.

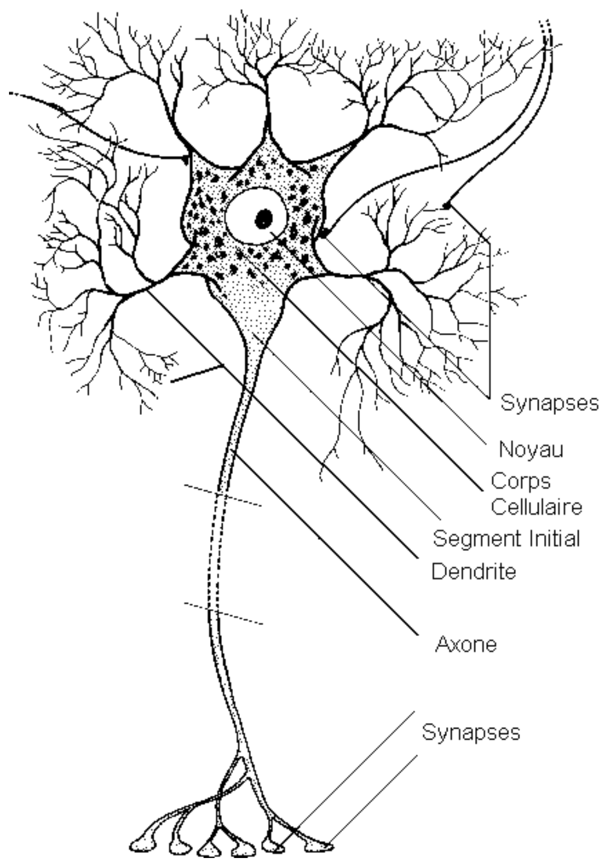


Figure 8- Exemple de Neurone Biologique

Cet héritage de la neurobiologie forme une composante importante de l'étude des réseaux connexionnistes, et le souci de maintenir une certaine correspondance avec le système nerveux humain a animé et continue à animer une part importante des recherches dans ce domaine. Malgré cet héritage, l'essentiel des travaux d'aujourd'hui ont pour objet les réseaux de neurones formels et non son corrélât neurobiologique. Vu comme des systèmes de calcul, les réseaux de neurones possèdent plusieurs propriétés qui les rendent intéressants d'un point de vue théorique, et fort utiles en pratique. C'est cette approche - plus technique que biologique - qui est adoptée pour nous dans cette thèse.

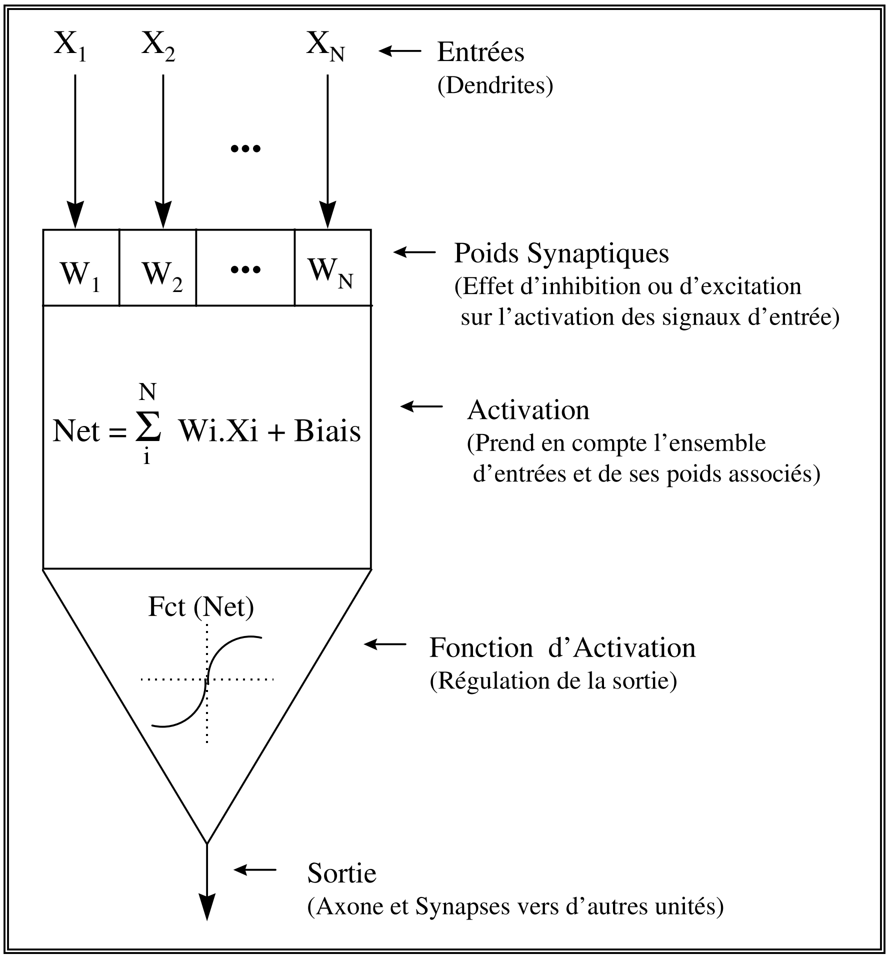


Figure 9- Exemple de Neurone Formel

Un réseau connexionniste est constitué par un graphe orienté et pondéré. Les noeuds de ce graphe sont des automates simples nommés *neurones formels* ou tout simplement *unités du réseau*. Les unités sont dotées d'un *état interne*, que l'on appelle *état d'activation*. Les unités peuvent propager son état d'activation aux autres unités du graphe en passant par des arcs pondérés appelés *connexions*, *liens synaptiques* ou bien *poids synaptiques*. La règle qui détermine l'activation d'un neurone en fonction de l'influence venue de ses entrées et de leurs poids respectifs s'appelle *règle d'activation* ou *fonction d'activation*. Les *changements* apportés aux valeurs *des poids synaptiques* ou dans la structure d'interconnexion des unités du réseau sont responsables des changements de comportement d'activation de ce réseau, ce qui lui permet d'*apprendre* un nouveau comportement. Ainsi, le réseau est capable d'établir des associations entrée-sortie (stimulus et réponse) afin de bien résoudre un problème. La méthode utilisée pour modifier le comportement d'un réseau s'appelle règle d'apprentissage.

La grande quantité de modèles de réseaux connexionnistes rend difficile leur descritpion exhaustive. Le lecteur pourra se référer aux nombreux ouvrages sur le sujet pour une description générale des réseaux de neurones. En particulier, l'ouvrage édité par E. Fiesler et R. Beale, intitulé "Handbook of Neural Computation" [FIE 97] complétera les différents points abordés ici (voir aussi les références citées dans la section 2.1.2.1). Nous allons plutôt approfondir ci-dessous les différences entre les modèles, afin de mieux comprendre les choix qui ont été faits dans cette thèse.

2.3.1 Classification et Propriétés

La grande quantité de modèles connexionnistes existants [SAR 97, JOD 94b] nous oblige à tout d'abord les classer selon leurs différentes proprietés, afin de mieux comprendre les avantages et les inconvénients du choix d'un modèle plutôt que l'autre. Il n'existe pas une seule façon de classer les modèles existants, mais plusieurs, selon les attributs choisis, tels que : le type d'apprentissage, l'architecture d'interconnexion des unités du réseau, la forme pour traiter et représenter les données, etc. Une description plus detaillée des réseaux connexionnistes et de leur classification est donnée par E. Fiesler [FIE 97].

2.3.1.1 Apprentissage Connexionniste

L'apprentissage est en général un processus graduel, itératif, où les poids du réseau sont modifiés plusieurs fois selon une règle d'apprentissage avant d'atteindre leurs valeurs finales. Les approches d'apprentissage connexionniste peuvent être reparties en trois grandes classes, selon le degré de contrôle donné à l'utilisateur :

- *Apprentissage Supervisé* : l'utilisateur dispose d'un comportement de référence précis qu'il désire faire apprendre au réseau. Le réseau est donc capable de mesurer la différence entre son comportement actuel et le comportement de référence, et de corriger ses poids de façon à réduire cette erreur. L'apprentissage supervisé utilise des connaissances empiriques, habituellement représentées par des ensembles d'exemples étiquetés par la classe à laquelle ils appartiennent.
- *Apprentissage Semi-Supervisé* : l'utilisateur ne possède que des indications imprécises (par exemple, échec/succès du réseau) sur le comportement final désiré. Les techniques d'apprentissage semi-supervisé sont aussi appelées apprentissage par renforcement (*reinforcement learning*). En effet, on dispose souvent tout au plus d'une évaluation qualitative du comportement du système.
- *Apprentissage non-supervisé* : les poids du réseau sont modifiés en fonction de critères internes comme la coactivation des neurones. Les comportements résultants de ces apprentissages sont en général comparables à des techniques d'analyse de données. On les appelle aussi *auto-organisation*.

L'apprentissage connexionniste nécessite en général une grande quantité de données, que l'on regroupe dans un *ensemble d'exemples d'apprentissage*. Selon la technique d'apprentissage utilisée, d'autres ensembles de données sont aussi employés, notamment pour mesurer la validité de la solution trouvée par le réseau. On appelle ces données supplémentaires de *ensembles d'exemples de test* ou de *généralisation*. On appelle généralisation la capacité du réseau à réagir correctement lorsqu'on lui présente des entrées qui n'ont pas été vues lors de l'apprentissage. Dans la pratique, ce sont les *capacités de généralisation* d'un réseau connexionniste qui vont établir les possibilités de son application à différents problèmes.

Un réseau peut trop se spécialiser à un ensemble d'apprentissage. Ce type de comportement d'apprentissage va probablement mener à un problème de mauvaise généralisation, ou surapprentissage (*overfitting*). Une façon d'éviter le surapprentissage est l'utilisation de la procédure de la validation croisée (*cross-validation*). Cette procédure consiste en diviser les données en deux ensembles distincts : un ensemble d'apprentissage et un ensemble de test. L'ensemble de test nous permet de contrôler l'erreur de généralisation ; ainsi, on peut déterminer le bon moment pour arrêter le processus d'apprentissage. La Figure 10 montre deux courbes où sont représentées les erreurs de sortie d'un réseau par rapport aux ensembles d'apprentissage et de test.

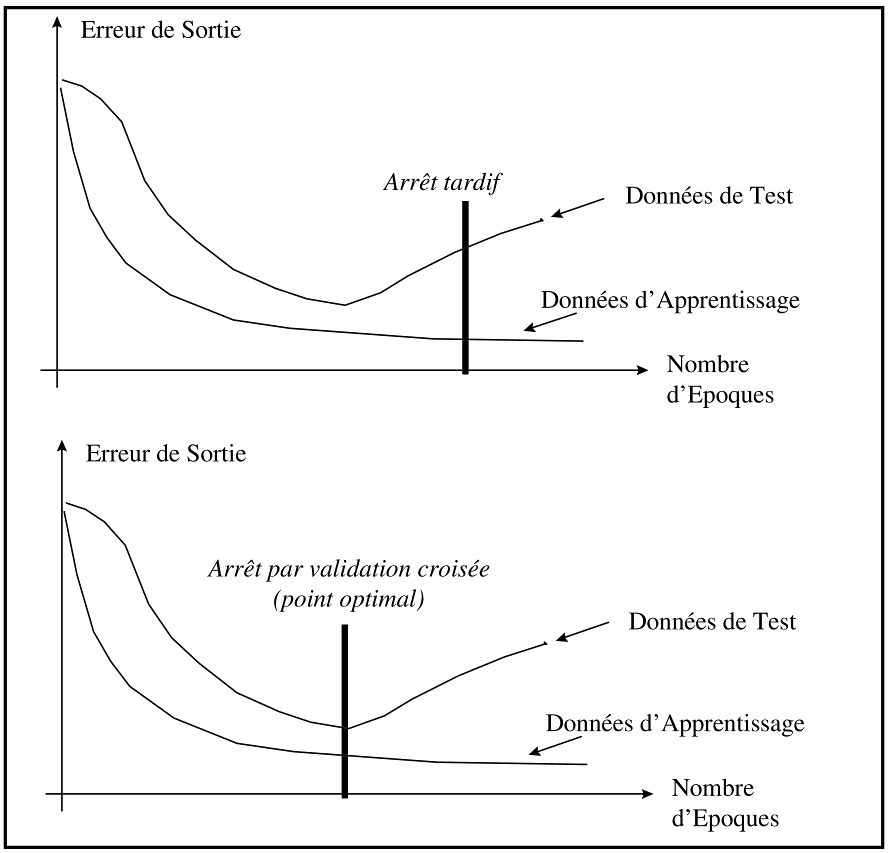


Figure 10- Apprentissage : Erreur dans l'ensemble d'apprentissage et de test

L'apprentissage d'un ensemble de données peut être réalisé de différentes façons, selon la manière dont le réseau est alimenté par les données :

• *Apprentissage Instantané* : l'ensemble de données d'apprentissage est analysé une seule fois et permet d'établir les poids du réseau d'une seul coup. Ce mode d'apprentissage requiert un seul passage sur les données et il est aussi appelé *one-shot-learning*. Ce n'est pas une méthode très utilisée.

• *Apprentissage par Paquets* : l'ensemble des données d'apprentissage est présenté au réseau plusieurs fois, de façon à optimiser les poids du réseau et minimiser l'erreur en sortie. Chaque présentation de l'ensemble complet de données d'apprentissage est appelée une époque (*epoch*). L'algorithme d'apprentissage réduit petit à petit l'erreur de sortie à chaque présentation des données, et l'état du réseau doit converger vers la solution du problème. On peut aussi manipuler l'ordre des exemples de l'ensemble d'apprentissage, ce qui peut avoir des conséquences sur l'évolution de l'apprentissage (cf. l'apprentissage actif). Cette méthode est connue aussi comme *batch-learning* et constitue l'un des types d'apprentissage les plus utilisés.

• *Apprentissage Continu* : l'algorithme d'apprentissage prend en compte continuellement les exemples qui lui arrivent (*continuous/on-line learning*). Cette méthode est aussi appelée apprentissage incrémental (par rapport aux données), mais nous allons utiliser plutôt le terme apprentissage continu pour éviter toute confusion avec l'apprentissage incrémental dans le cas où des unités sont ajoutées à la structure du réseau. Un des principaux problèmes de l'apprentissage continu est la difficulté de trouver un bon compromis entre adaptation et stabilité, ce qui peut amener le réseau à 'oublier' les données apprises (trop d'adaptation) ou bien être incapable de s'adapter aux nouvelles données qui arrivent (trop de stabilité).

• *Apprentissage Actif* : cette approche est basée sur l'idée que l'algorithme d'apprentissage peut lui aussi imposer des choix par rapport aux données à utiliser. L'apprentissage passe d'un comportement passif, où le réseau ne peut pas intervenir sur les données qui lui arrivent, à un comportement actif, où le réseau peut déterminer quelles données vont être prises en compte et aussi dans quel ordre elles doivent être considérées. Il s'agit d'une voie de recherche relativement récente.

L'apprentissage peut être aussi implementé par différentes méthodes, selon le type de *règle d'apprentissage* sélectionné. Les règles d'apprentissage les plus utilisées sont [JOD 94b, SIM 90, CAU 92] :

- Les méthodes de *correction d'erreur*, telles que la descente de gradient sur une surface d'erreur. Exemples : Adaline, Perceptron, Rétro-propagation, Cascade-Correlation;

- Les méthodes d'apprentissage par *renforcement*. Exemples : AHC, ARC;
- Les méthodes d'apprentissage par *compétition* ou par *auto-organisation*. Exemples : Kohonen Feature Map, ART1;
- Les méthodes d'apprentissage par création de *prototypes* ou de *noyaux*. Exemples : RBF, ART1, ARN2;
- Les méthodes d'apprentissage basées sur des *mémoires associatives* (auto-associatives ou hétéro-associatives). Exemples : Modèle de Hopfield, BAM;
- Les méthodes *d'apprentissage temporel* (réseaux récurrents). Exemples : SRN, BPTT, RTRL.

Il existe des méthodes qui peuvent appartenir à plus d'une classe au même temps. Par exemple, les réseaux avec apprentissage du type ARN2 incluent dans cette approche des techniques d'apprentissage non-supervisé, supervisé, par compétition et aussi par création de prototypes. Dans cette thèse, nous allons nous focaliser plutôt sur les méthodes d'apprentissage supervisé fondées sur la descente du gradient. Le système que nous avons développé dans le cadre de cette thèse utilise un algorithme dérivé de la Rétro-Propagation - le Cascade-Correlation (CasCor) [FAH 90]. Nous allons étudier plus en détails les algorithmes d'apprentissage de type Rétro-Propagation dans la section 2.3.2.

2.3.1.2 Type des Unités

Les unités d'un réseau (neurones formels) peuvent être de différents types, selon la fonction interne utilisée pour calculer l'activation. Les principales différences résident dans le type de fonction d'activation employé (e.g.: linéaire, sigmoïde exponentielle - asymétrique, sigmoïde tangentielle - symétrique, gaussienne) [JOD 94a, JOD 94b], et dans la fonctionnalité des unités (unités à prototypes, unités du type Perceptron). Selon la fonctionnalité des unités nous pouvons avoir deux types majeurs de réseaux :

• *Réseaux à Prototypes* : ce type de réseau utilise des unités qui servent à représenter des prototypes d'exemples appris : les unités ont une représentation interne regroupant les caractéristiques typiques d'un ensemble d'exemples [CHE 97, MAL 96b, ORS 95, GIA 92]. Les réseaux à prototypes ont normalement une apprentissage non-supervisé, mais ils peuvent aussi avoir un apprentissage supervisé (avec un ou plusieurs prototypes associés à chaque classe). Une des avantages de ce type de réseau est la possibilité de faire un apprentissage continu (incrémental), puisqu'il n'est pas très difficile de concevoir des méthodes qui permettent d'augmenter le réseau par ajout de nouveaux prototypes. Les prototypes sont aussi appelés des noyaux.

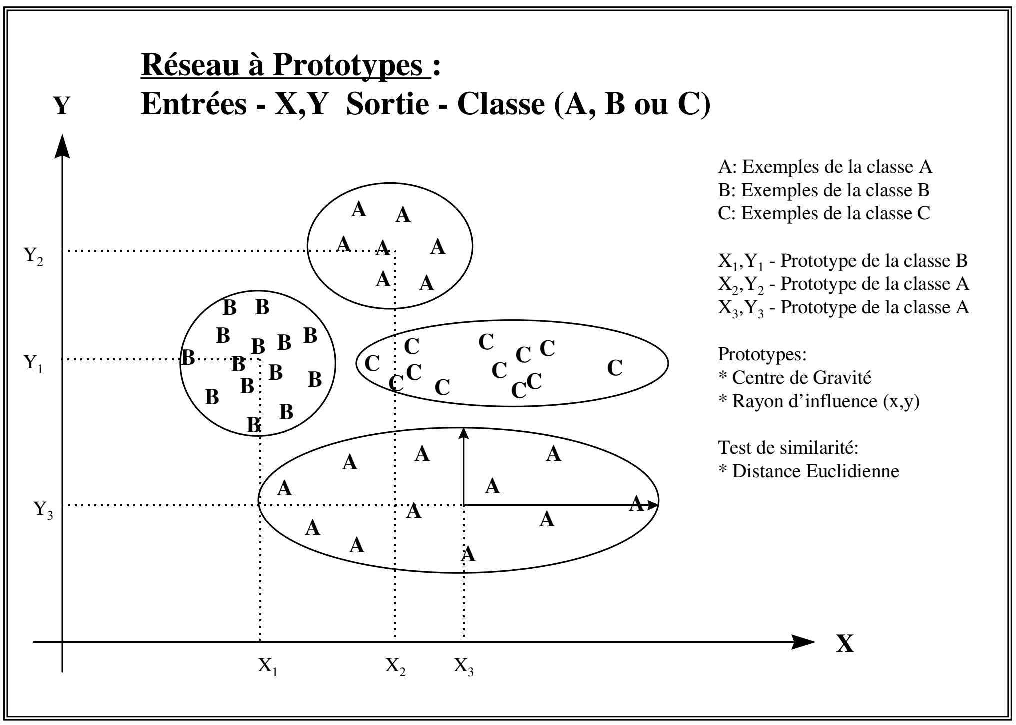


Figure 11 - Prototypes d'un réseaux à deux entrées

•*Réseaux de type "Perceptron"* : les unités du type "Perceptron" ont été proposées par Frank Rosenblatt en 1958. Il s'agit d'un des modèles d'unités les plus utilisés actuellement. Il est à l'origine de plusieurs autres réseaux de neurones avec apprentissage supervisé par correction d'erreur. Le modèle du Perceptron Multi-Couche (PMC) est devenu très connu, tout en étant associé à la règle d'apprentissage de la Rétro-Propagation [JOD 94b, WID 90, RUM 86b]. Nous allons étudier ce type de réseau dans la section 2.3.2.

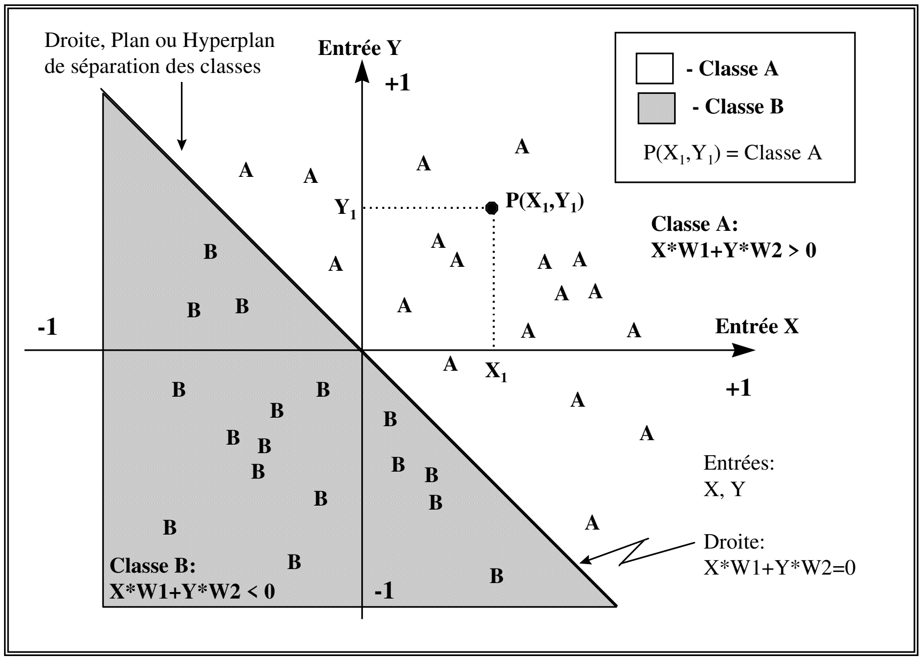


Figure 12 - Séparation des classes par un Perceptron à deux entrées

2.3.1.3 Type d'Architecture du Réseau

La façon d'interconnecter les unités d'un réseau va nous permettre d'obtenir différentes architectures. La Figure 13 montre des exemples de différentes façons d'interconnecter les unités. Les architectures les plus importantes sont :

- *Réseaux à une seule couche* : les unités sont toutes sur le même niveau. Dans ce type d'architecture, les unités sont connectées directement aux entrées et sont aussi les sorties du réseau. Les réseaux à une seule couche ont normalement des connexions latérales (entre les neurones d'une même couche). Un exemple de ce type d'architecture sont les réseaux du type "Kohonen Feature Map".

• *Réseaux à couches unidirectionnels* : les unités sont organisées en plusieurs niveaux bien définis, que l'on appelle des couches. Chaque unité d'une couche reçoit ses entrées à partir de la couche précédente et envoie ses sorties vers la couche suivante (*feed-forward nets*). La Figure 13(a) montre un exemple de réseaux à trois couches. Cette architecture à trois couches (entrée, couche cachée et sortie) est très utilisée dans la pratique. Le modèle du PMC [JOD 94b, WID 90] se compose en général d'une architecture de ce type, c'est-à-dire avec une seule couche cachée (*hidden layer*), mais rien n'empêche d'avoir plus d'une seule couche cachée dans ce modèle. Un autre type d'interconnexions dans les réseaux à couches sont les raccourcis (*short-cuts*) qui permettent de lier des unités en passant à travers des niveaux. Ainsi, on peut 'sauter' d'une couche à l'autre avec les raccourcis, à condition de ne pas créer une boucle (voir Figure 13(b)).

• *Réseaux récurrents* : les réseaux récurrents [SZI 95, JOD 94b, LAB 93] peuvent avoir une ou plusieurs couches, mais leur particularité est la présence d'interconnexions depuis la sortie d'une unité vers une autre unité de la même couche ou d'une couche inférieure. Ce type d'interconnexions permet de modéliser des aspects temporaux et des comportements dynamiques, où la sortie d'une unité dépend de son état antérieur. Les boucles internes rendent ce type de réseaux instable, ce qui nous oblige à utiliser des algorithmes plus spécifiques (et usuallement plus complexes) pour l'apprentissage. Un type particulier de réseau récurrent sont les réseaux totalement connectés, tels que le modèle de Hopfield représenté dans la Figure 13(d).

•*Réseaux d'ordre supérieur* : les unités de ce type permettent la connexion directe entre deux entrées ou plus, avant d'appliquer la fonction de calcul de l'activation de l'unité [FIE 94a]. Ce type de réseau sert à modéliser les synapses de modulation, c'est-à-dire quand une entrée peut moduler (agir sur) le signal provenant d'une autre entrée. Un modèle particulier de réseaux d'ordre supérieur (*high-order neural net*) est le réseau *Sigma-Pi* proposé par Rumelhart [RUM 86a], et représenté dans la Figure 13(e).

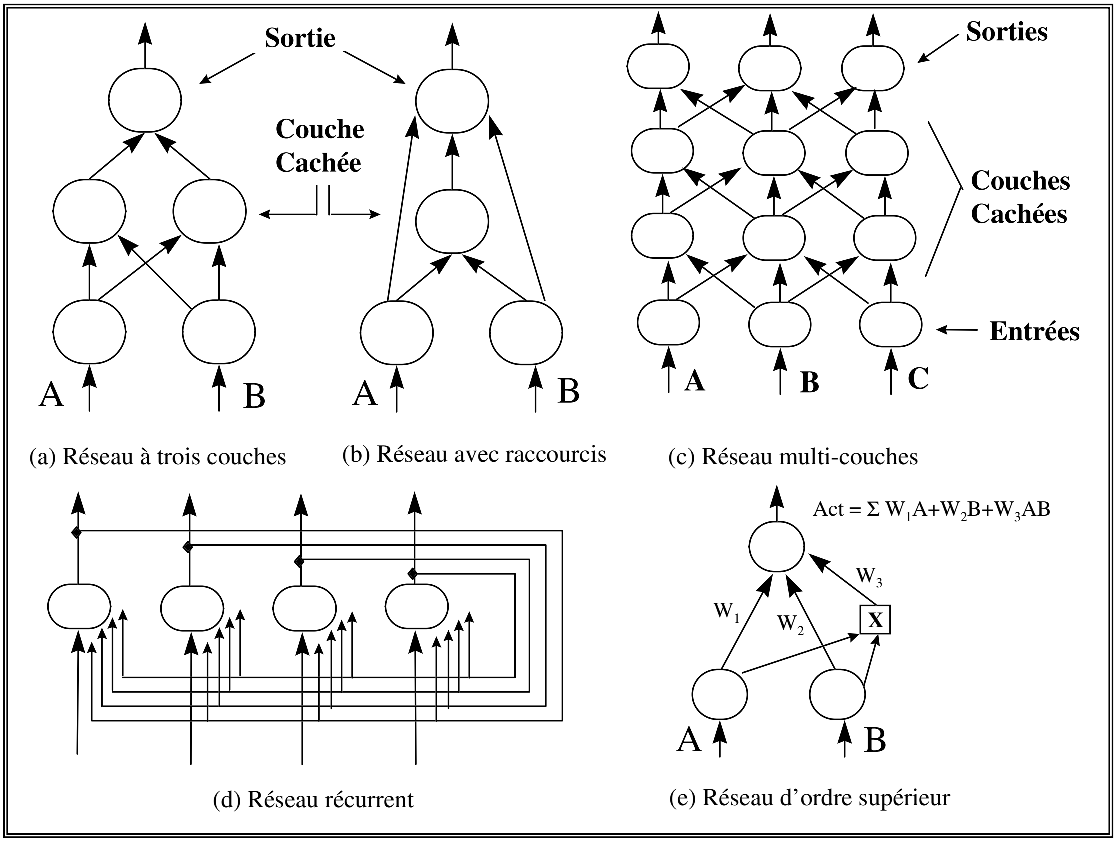


Figure 13 - Architectures d'interconnexion des unités d'un réseau

L'architecture d'un réseau peut être aussi classée selon son évolution au cours de l'apprentissage et de son utilisation. En fonction de ce critère, les réseaux sont divisés dans les groupes suivants :

• *Structure statique* : le réseau a sa structure définie avant l'apprentissage. La quantité de neurones ainsi que sa structure d'interconnexions ne changent pas. Les seuls changements subis par le réseau se situent au niveau des poids synaptiques, modifiés au cours de l'apprentissage. Ce type de modèle pose une difficulté : la détermination du bon nombre de neurones et interconnexions à utiliser. Un réseau avec peu d'unités et d'interconnexions risque de ne pas pouvoir apprendre à résoudre correctement un problème, et un réseau avec trop d'unités peut avoir des problèmes de convergence, de complexité et de généralisation [JOD 94a, FLE 94]. Il n'existe pas de méthode optimale qui permet de spécifier automatiquement le nombre exact d'unités à employer pour bien résoudre un problème donné. Les réseaux PMC à Rétro-Propagation classiques sont des réseaux statiques.

• *Structure dynamique* : les réseaux ayant une structure dynamique sont des réseaux dont le nombre d'unités et d'interconnexions peut varier. Ils sont aussi appelés réseaux ontogéniques [FIE 94b]. Les changements dans la structure du réseau peuvent être de type *génératif* (*incrémental*) ou de type *destructif* (*élagage*). Le choix entre ces deux méthodes est controversé : faut-il commencer petit et agrandir ou faut-il commencer grand et réduire ensuite ? [ELM 93]. D'un point de vue calculatoire, commencer petit et ajouter des unités et des interconnexions au fur et à mesure de l'apprentissage est plus performant que faire le travail d'apprentissage sur un grand réseau pour en détruire certaines parties ensuite. Une seule chose semble être acceptée par la plupart des chercheurs : les réseaux ontogéniques sont une des meilleures méthodes dont on dispose pour choisir une bonne architecture pour un réseau et pour bien résoudre un problème donné. Les réseaux CasCor, que nous avons utilisé dans cette thèse, sont des réseaux dynamiques du type incrémental.

Le dernier point relevant des architectures concerne la *modularité des réseaux* [AMY 96]. Les réseaux neuronaux peuvent avoir des architectures modulaires : ils peuvent être constitués par des blocs plus ou moins dépendants les uns des autres. On peut avoir des différentes approches pour intégrer et faire coopérer les modules d'un réseau. Une première approche consiste à décomposer le problème et obtenir ainsi des modules spécialisés pour chaque sous-problème. Dans cette approche, si l'on prend une tache de classification en plusieurs classes, on peut considérer que l'identification de chaque classe va pouvoir être traitée par des modules séparés. L'autre approche, plus difficile à mettre en place, est celle où les différents modules vont coopérer pour aboutir à une solution. Dans cette approche, on ne va pas imposer de tâches particulières à des modules pré-spécifiés. La modularité est un problème relatif au choix de l'architecture d'un réseau, mais peut être aussi lié au choix du partitionnement des données. Finalement, la modularité peut devenir un aspect très important à prendre en compte selon la complexité du problème abordé. Un étude plus profonde sur ce sujet a été développée par S. Rouzier [ROU 98].

2.3.1.4 Type d'Application du Réseau

Les réseaux peuvent être appliqués à plusieurs types de tâches, telles que : la reconnaissance de formes (e.g. reconnaissance de visages), la classification, la transformation de données (e.g. compression), la prédiction (e.g. prédiction de séries temporelles), le contrôle de processus et l'approximation de fonctions. Toutes ces tâches peuvent être regroupés en deux groupes principaux selon le type des sorties fournies par le réseau et le comportement qui est recherché. Ces deux groupes principaux d'applications sont :

- *Réseaux pour l'Approximation de Fonctions* : ce type de réseau doit avoir une sortie continue et sera employé pour l'approximation exacte (*interpolation*) ou pour l'approximation approchée d'une fonction représentée par les données d'apprentissage [CHE 97]. Ce type de réseau est capable d'apprendre une fonction de transformation (ou d'association) des valeurs d'entrée vers les valeurs de sortie. Cette fonction acquise par le réseau permet de prédire les sorties étant données les entrées. On appelle ce type de problème, un problème de *régression* [BIS 97]. En général, les fonctions représentées sont des fonctions avec des variables d'entrée et de sortie continues.

- *Réseaux pour la Classification* : ce type de réseau doit attribuer des classes (valeur de sortie non-continue) aux exemples qui lui sont fournis. La classification est un cas particulier de l'approximation de fonctions où la valeur de sortie est discrète et appartient à un ensemble limité de classes. Cet ensemble de classes peut être connu d'avance dans le cas de l'apprentissage supervisé. Un réseau adapté à la classification doit avoir des sorties discrètes ou implementer des méthodes de discrétisation des sorties (e.g. application d'un seuil de discrimination).

Il est un peu délicat de classer précisément tous les différents modèles de réseaux de neurones dans une seule de ces deux classes. La plupart des modèles peuvent être adaptés pour être utilisés dans l'un ou l'autre type d'application. Mais certains modèles sont plus adaptés à une tache de classification que d'approximation et vice versa.

2.3.2 Le modèle du Perceptron Multi-Couches

Nous allons faire ici une brève description des réseaux de type Perceptrons Multi-Couches (PMC) et des processus d'activation et d'apprentissage couramment utilisés avec ce type de modèle. Les réseaux PMC sont constitués par des unités organisées en couches : une couche d'entrée, une ou plusieurs couches cachées, et une couche de sortie (voir Figure 13 (a,b,c)). Les unités de la couche d'entrée reçoivent les signaux provenants de l'environnement. Ces signaux représentent les informations traitées par le réseau, telles que : la description de l'état du système, les valeurs des attributs associés aux variables d'entrée, les mesures obtenues à partir de capteurs externes ou les faits qui composent les prémisses d'un système expert. La couche de sortie rend disponible le résultat de l'activation du réseau à l'environnement extérieur. Enfin, les unités cachées, comme leur nom l'indique, n'ont pas d'interaction directe avec l'environnement.

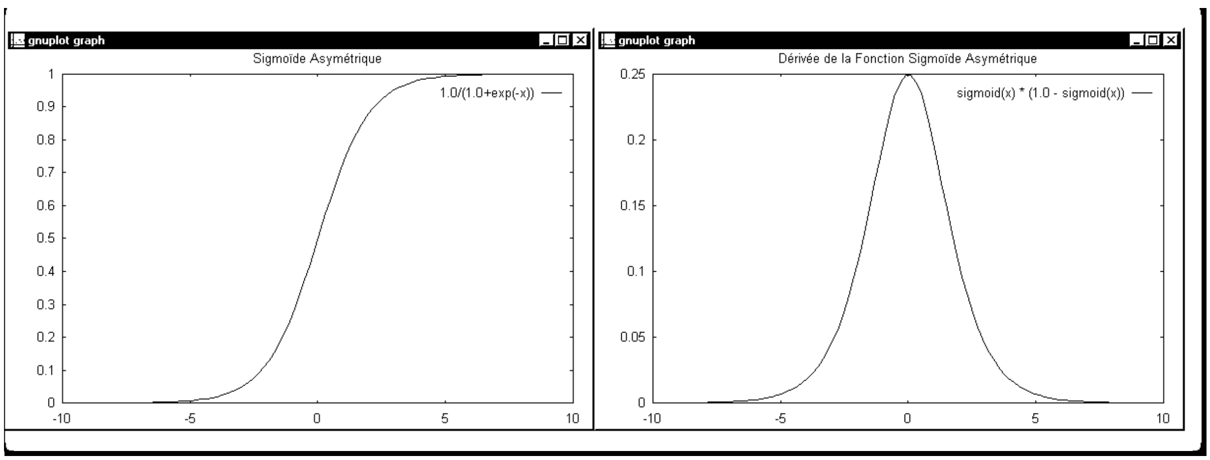


Figure 14- Sigmoïde Asymétrique

Les unités sont connectées selon une architecture sans boucles (connexions unidirectionnelles), mais on peut avoir des connexions inter-couches du type "raccourcis". Les réseaux PMC travaillent sur deux modes opératoires distincts : en mode de rappel (activation du réseau/consultation) et en mode d'apprentissage (adaptation). L'activation des réseaux PMC est réalisée par la propagation des signaux à partir des unités d'entrée vers les sorties. Ces unités reçoivent en entrée des signaux provenant soit des entrées externes soit des sorties des autres unités du réseau auxquelles elles sont connectées. Chaque unité génère une valeur de sortie qui est envoyée vers les unités auxquelles elle est connecté ou vers la sortie externe. La sortie d'une unité du réseau est obtenue à partir des équations suivantes :

Somme pondérée : $Si = \sum_{j=1}^{N} Wij.\, Xj + \Theta i$ (Eq. 1)

Fonction d'activation (sigmoïde asymétrique) : $Fa(x) = \dfrac{1}{1+e^{-x}}$ (Eq. 2)

Fonction d'activation (tangente hyperbolique symétrique) : $Fa(x) = \tanh(x)$ (Eq. 3)

En utilisant la fonction d'activation du type sigmoïde asymétrique on obtient :

Activation de l'unité : $Ai = Fa(Si) = \dfrac{1}{1+e^{-(\sum Wij.\, Xj + \Theta i)}}$ (Eq. 4)

avec :

Xj : Valeur de l'entrée 'j' de l'unité en question. Cette valeur peut être originaire d'une entrée du réseau ou de la sortie d'une autre unité.

Wij : Poids associé à l'entrée 'j' de l'unité 'i'.

Θi : Seuil de l'unité 'i'. Il peut être considéré comme un poids synaptique associé à une entrée toujours égale à 1. (biais).

N : Nombre d'entrées de l'unité en question.

Si : Valeur cumulée des entrées pondérées par les poids synaptiques de l'unité 'i'.

Ai : Valeur obtenue à la sortie de l'unité 'i' après activation.

Fa(x) : Fonction d'activation, en général la sigmoïde (voir Figure 14).

Les réseaux PMC utilisent une méthode d'apprentissage de type supervisé. L'apprentissage est réalisé par un algorithme de minimisation sur l'erreur estimé à partir des sorties du réseau. Cet algorithme réalise une descente du gradient de la surface d'erreur (voir Figure 15). L'emploi de cette méthode permet d'entraîner les unités de sortie du réseau, où l'erreur peut être obtenu directement par simple différence entre la valeur de sortie désirée et la valeur obtenue par le réseau ; mais il permet aussi d'entraîner les unités cachées grâce à une technique de propagation en arrière de l'erreur à travers les couches du réseau, d'où le nom de Rétro-Propagation" (*Back-Propagation Learning*).

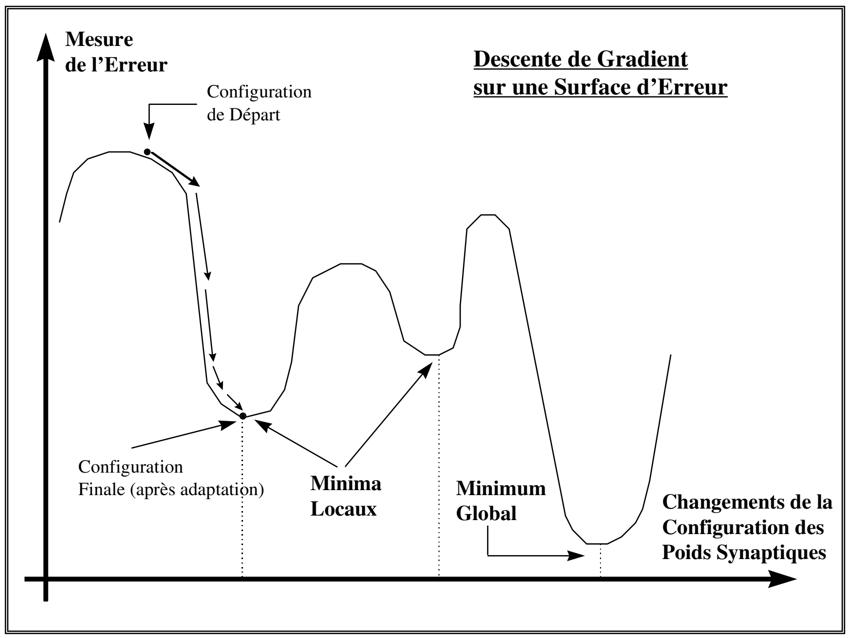


Figure 15 - Descente de gradient sur une surface d'erreur et minima locaux

Les Perceptrons ont été créés par Rosenblatt [ROS 59] il y a longtemps, mais les réseaux de ce type sont restés limités à des architectures avec deux couches seulement. Pourtant, certains problèmes ne pouvaient pas être traités par un simple réseau à deux couches (comme l'apprentissage de la fonction logique "Ou Exclusif" - XOR). C'est seulement à partir des années 80 que des règles d'apprentissage pour les réseaux à plus de deux couches ont été proposées. L'algorithme d'apprentissage de la Rétro-Propagation de l'erreur [RUM 86, PAR 85, LEC 88] est la méthode qui a permis de résoudre le problème de l'attribution du blâme (*credit assignment*) concernant la répartition de l'erreur observée parmi les unités de la couché cachée des réseaux PMC. Cette méthode est en effet assez proche de la règle delta de Widrow et Hoff [WID 90]. Elle sert à minimiser l'erreur quadratique entre la sorties du réseau et la sortie désirée, et adapter les poids des unités cachées du réseau.

L'algorithme d'apprentissage par minimisation de l'erreur quadratique (descente du gradient) est dérivé des formules qui suivent :

Erreur quadratique : $E = \frac{1}{2}\sum_i (Di - Ai)^2$ (Eq. 5)

Adaptation des poids synaptiques par descente du gradient : $\Delta Wij = -\alpha \frac{\partial E}{\partial Wij}$ (Eq. 6)

Calcul du gradient de l'erreur par rapport au poids synaptique :

Unité de sortie avec une sortie linéaire (Ai = Si)

$\frac{\partial E}{\partial Wij} = \frac{\partial E}{\partial Ai}\frac{\partial Ai}{\partial Wij} = \delta i\,\mathrm{Xj}$ (Eq. 7)

$\frac{\partial E}{\partial Ai} = -\,(\mathrm{Di} - \mathrm{Ai}) = \delta i$ (Eq. 8)

$\frac{\partial Ai}{\partial Wij} = \mathrm{Xj}$ (Eq. 9)

$\Delta Wij = -\alpha.\delta i.Xj = \alpha.(\mathrm{Di} - \mathrm{Ai}).\mathrm{Xj}$ (Eq. 10)

Unité de sortie avec une fonction d'activation non-linéaire (Ai = Fa(Si))

$\frac{\partial E}{\partial Wij} = \frac{\partial E}{\partial Si}\frac{\partial Si}{\partial Wij} = \delta i\,\mathrm{Xj}$ (Eq. 11)

$\frac{\partial E}{\partial Si} = \frac{\partial E}{\partial Ai}\frac{\partial Ai}{\partial Si} = -\,(\mathrm{Di} - \mathrm{Ai}).\mathrm{Fa}'(\mathrm{Si}) = \delta i$ (Eq. 12)

$\frac{\partial E}{\partial Ai} = -\,(\mathrm{Di} - \mathrm{Ai})$ (Eq. 13)

$$\frac{\partial Ai}{\partial Si} = \text{Fa'(Si)} \qquad \text{(Eq. 14)}$$

$$\frac{\partial Si}{\partial Wij} = \text{Xj} \qquad \text{(Eq. 15)}$$

$$Fa(x) = \frac{1}{1+e^{-x}} \quad \therefore \quad \text{Fa'(x) = Fa(x).(1 - Fa(x))}$$

$$\text{Fa'(Si) = Fa(Si).(1 - Fa(Si)) = Ai.(1 - Ai)} \qquad \text{(Eq. 16)}$$

$$\text{Fa(x) = tanh (x)} \quad \therefore \quad \text{Fa'(x) = (1 - Fa(x).Fa(x))}$$

$$\text{Fa'(Si) = (1 - Fa(Si).Fa(Si)) = (1 - Ai.Ai)} \qquad \text{(Eq. 17)}$$

$$\Delta Wij = -\alpha.\delta i.Xj = \alpha\text{.(Di - Ai).Fa'(Si).Xj} \qquad \text{(Eq. 18)}$$

Unité cachée avec une fonction d'activation non-linéaire (Ai = Fa(Si))

$$\frac{\partial E}{\partial Wij} = \frac{\partial E}{\partial Si}\frac{\partial Si}{\partial Wij} = \delta i\,\text{Xj} \qquad \text{(Eq. 19)}$$

$$\frac{\partial E}{\partial Si} = \frac{\partial E}{\partial Ai}\frac{\partial Ai}{\partial Si} = \text{- Fa'(Si).}\sum_k \delta kWki = \delta i \qquad \text{(Eq. 20)}$$

$$\frac{\partial E}{\partial Ai} = \sum_k \frac{\partial E}{\partial Sk}\frac{\partial Sk}{\partial Ai} = \sum_k \frac{\partial E}{\partial Sk}\frac{\partial}{\partial Ai}\sum_h WhkAh = \ldots$$

$$\ldots = \sum_k \frac{\partial E}{\partial Sk}Wki = -\sum_k \delta kWki \qquad \text{(Eq. 21)}$$

$$\frac{\partial Ai}{\partial Si} = \text{Fa'(Si)} \qquad \text{(Eq. 22)}$$

$$\frac{\partial Si}{\partial Wij} = \text{Xj} \qquad \text{(Eq. 23)}$$

$$\Delta Wij = -\alpha.\delta i.Xj = \alpha\text{.Xj.Fa'(Si).}\sum_k \delta kWki \qquad \text{(Eq. 24)}$$

Les notations utilisées sont les suivantes :

Fa'(x) : Dérivée de la fonction d'activation. Nous avons présenté deux exemples de fonctions d'activation Fa(x) et ses dérivées : la sigmoïde asymétrique et la sigmoïde symétrique (tanh).

Di : Sortie désirée de l'unité 'i', provenant de la base d'apprentissage.

Si : Valeur cumulée des entrées pondérées par les poids synaptiques de l'unité 'i'.

Ai : Valeur obtenue à la sortie de l'unité 'i' après activation.

Xj : Valeur de l'entrée 'j' de l'unité en question.

E : Mesure de l'erreur (fonction d'erreur quadratique).

Wij : Poids synaptique associé à l'entrée 'j' de l'unité 'i'

Wki : Poids associé à l'entrée 'i' de l'unité 'k'. L'unité 'k' se trouve connectée à la sortie de l'unité 'i' (couche précédente).

δi : Représente l'erreur de l'unité 'i'.

δk : Représente l'erreur de l'unité 'k'. L'unité 'k' se trouve dans la couche suivante après 'i'.

ΔWij : Valeur du changement effectué sur le poids Wij.

α : Valeur constante qui sert à contrôler la vitesse de convergence de l'apprentissage (pas d'apprentissage - *learning rate*). Cette valeur est comprise dans l'intervalle [0..1].

L'algorithme de la Rétro-Propagation peut être amélioré par l'ajout d'un terme d'inertie dans le calcul du ΔWij :

$$\Delta Wij(t) = -\alpha . \delta i . Xj + \beta . \Delta Wij(t-1) \qquad \text{(Eq. 25)}$$

Où nous avons :

$\Delta Wij(t)$: Valeur du changement effectué sur le poids Wij à l'instant 't'.

$\Delta Wij(t-1)$: Valeur du changement effectué sur le poids Wij à l'instant 't-1'.

β : Valeur constante qui sert à contrôler l'inertie (*momentum*) du changement des poids. Cette valeur est comprise dans l'intervalle [0..1].

On obtient ainsi les équations finales qui représentent l'adaptation des poids d'un réseau PMC par Rétro-Propagation, avec des unités de type sigmoïde asymétrique :

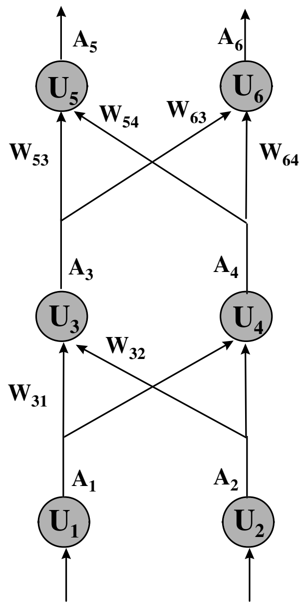
Wij = Wij + $\Delta Wij(t)$ (Eq. 26)

$\Delta Wij(t) = \alpha$.(Di - Ai).(Ai.(1-Ai)).Xj + $\beta.\Delta Wij(t-1)$ (Unité de sortie)

$\Delta Wij(t) = \alpha . \sum_k \delta k Wki$.(Ai.(1-Ai)).Xj + $\beta.\Delta Wij(t-1)$ (Unité de la couche cachée)

En résumé, le processus d'apprentissage consiste à présenter au réseau un exemple avec sa valeur de sortie désirée. L'activation du réseau va produire des valeurs de sortie qui sont ensuite utilisées pour le calcul de l'erreur sur la couche de sortie. Cette erreur des unités de sortie est ensuite propagée aux couches antérieures en utilisant les équations que nous venons de décrire (voir Figure 16). Une fois que toutes les unités ont été traitées et qu'on dispose de toutes les valeurs de $\Delta Wij(t)$, on effectue l'adaptation des poids selon l'Equation 26. Les seuils associés à chacune des unités (Θi) sont traités comme des poids connectés à des entrées qui ont une valeur constante toujours égal à 1. Ainsi, les équations d'adaptation des poids décrites ci-dessus sont appliquées exactement de la même façon pour l'adaptation des seuils des unités du réseaux.

L'adaptation des poids avec l'algorithme de la Rétro-Propagation peut être effectuée après la présentation de chaque exemple, mais on peut aussi faire le cumul des $\Delta Wij(t)$ pendant la présentation de toute la base d'exemples d'apprentissage et réaliser la mise à jour des poids seulement à la fin de cette période. Cette dernière méthode s'appelle apprentissage par paquets ou par époques.



$A_5 = Fa\,(W_{53}.A_3 + W_{54}.A_4 + \theta_5)$
$A_3 = Fa\,(W_{31}.A_1 + W_{32}.A_2 + \theta_3)$

$\Delta W_{53} = -\alpha.\delta_5.A_3 = -\alpha.[(D_5\text{-}A_5).(A_5.(1\text{-}A_5))].A_3$

$\Delta W_{54} = -\alpha.\delta_5.A_4 = -\alpha.[(D_5\text{-}A_5).(A_5.(1\text{-}A_5))].A_4$

$\Delta W_{63} = -\alpha.\delta_6.A_3 = -\alpha.[(D_6\text{-}A_6).(A_6.(1\text{-}A_6))].A_3$

$\Delta W_{64} = -\alpha.\delta_6.A_4 = -\alpha.[(D_6\text{-}A_6).(A_6.(1\text{-}A_6))].A_4$

$\Delta W_{31} = -\alpha.\delta_3.A_1 = -\alpha.[(\delta_5.W_{53} + \delta_6.W_{63}).(A_3\,.(1\text{-}A_3))].A_1$

δ_i : Erreur de l'unité 'i'
α : Valeur constante (0.0 =< α =< 1.0)
D_i : Sortie désirée de l'unité 'i'
A_i : Sortie de l'activation de l'unité 'i'
W_{ij} : Poids de l'entrée 'j' de l'unité 'i'
θ_i : Seuil de l'unité 'i'
Fa(x) : Fonction d'activation (sigmoïde)

Figure 16 - Détail du processus de Rétro-Propagation

2.3.3 Discussion sur les Réseaux Connexionnistes

Avant d'analyser les propriétés des réseaux connexionnistes, nous allons préciser ici un certain nombre de points importants relatifs à notre étude. Notre intérêt pour les réseaux réside essentiellement dans leur utilisation dans des systèmes intelligents tels que les systèmes experts. Pour réaliser ce type de systèmes, nous avons choisi d'utiliser des réseaux avec un apprentissage supervisé et nous nous sommes orientés vers des tâches de classification (plutôt que des tâches d'approximation de fonctions ou de modélisation de comportements dynamiques). Nous avons choisi d'étudier plus en détail les réseaux construits à partir d'unités à prototypes et les réseaux de type Perceptrons (PMC), car la plupart des systèmes experts connexionnistes sont dérivés de l'un de ces deux types de réseaux. Il est utile de rappeller que dans les systèmes experts connexionnistes, l'activation du réseau joue le rôle d'un moteur d'inférence d'un système expert classique, alors que ses connexions et ses poids synaptiques jouent le rôle des connaissances.

Les réseaux connexionnistes, en particulier ceux employés pour la construction de systèmes intelligents, possèdent les avantages suivants :

- *Connaissances Empiriques* : l'apprentissage à partir d'exemples (méthode d'apprentissage empirique) se fait d'une façon assez simple et permet d'obtenir de bons résultats par rapport aux autres techniques d'apprentissage automatique;

- *Dégradation progressive* : les réponses données par les réseaux se dégradent progressivement en présence de bruit dans les entrées. Les réseaux permettent de bien généraliser les connaissances présentes dans la base d'apprentissage et sont moins sensibles aux perturbations que les systèmes symboliques. Le fait de travailler sur une représentation 'numérique' des connaissances rend les réseaux plus adaptés pour manipuler des données quantitatives (valeurs continues). Les réseaux de neurones sont moins vulnérables aux données approximatives et à la présence de données incorrectes dans la base d'apprentissage;

- *Parallélisme massif* : les réseaux sont composés d'un ensemble d'unités de traitement de l'information qui peuvent opérer en parallèle. Bien que la plupart des implémentations des réseaux connexionnistes soient réalisées sur des simulateurs séquentiels, il est possible de faire des implémentations (logicielles ou matérielles) exploitant la possibilité d'activer simultanément les unités. La plupart des implémentations des réseaux de neurones peuvent être facilement converties d'une version séquentielle vers une version parallèle.

Les réseaux connexionnistes présentent aussi un certain nombre d'inconvénients, tels que :

- *Architecture et paramètres* : il n'existe pas de méthode automatique pour choisir la meilleure architecture possible pour un problème donné. Il est assez difficile de trouver la bonne topologie du réseau ainsi que les bons paramètres de réglage de l'algorithme d'apprentissage. L'évolution du processus d'apprentissage est très influencée par ces deux éléments (l'architecture du réseau et les paramètres de réglage) et dépend beaucoup du type de problème traité. Le simple fait de changer la base d'apprentissage utilisée, peut nous obliger à reconfigurer le réseau en entier;

- *Initialisation et codage* : les algorithmes d'apprentissage connexionniste sont en général très dépendants de l'état initial du réseau (initialisation aléatoire des poids) et de la configuration de la base d'apprentissage. Un mauvais choix des poids employés pour initialiser le réseau, de la méthode de codage des données, ou même de l'ordre des données, peut bloquer l'apprentissage ou poser des problèmes pour la convergence du réseau vers une bonne solution;

- *Boîte noire* : les connaissances acquises par le réseau sont codées par l'ensemble des valeurs des poids synaptiques ainsi que par la façon dont les unités sont interconnectées. Il est très difficile pour un être humain de les interpréter directement. Les réseaux connexionnistes sont des boîtes noires, où les connaissances restent enfermés et sont inintelligibles pour l'utilisateur ou pour l'expert. Un réseau ne peut pas expliquer le raisonnement qui l'a amené à une solution spécifique;

- *Connaissances théoriques* : les réseaux classiques ne permettent pas de profiter des connaissances théoriques disponibles sur le domaine du problème traité. Comme les arbres de décision, ils sont dédiés à la manipulation de connaissances empiriques. Une façon simpliste de profiter des connaissances théoriques consiste à convertir des règles en exemples (prototypes). Cependant, cette méthode ne garantit pas que ces exemples vont être bien représentés dans les connaissances du réseau à la fin de l'apprentissage, car nous sommes obligés à passer pour une phase d'apprentissage où se mélangent sans distinction des connaissances empiriques avec des connaissances théoriques codées par des exemples.

Cette liste n'est pas exhaustive (e.g. voir [ORS 95]), mais elle nous permet d'avoir une idée des principaux problèmes des réseaux. En particulier, la méthode d'apprentissage des PMC par Rétro-Propagation présente certains inconvénients, à savoir [FAH 88] :

- *Paralysie de l'apprentissage* : les réseaux PMC qui utilisent la Rétro-Propagation, en raison de la façon dont les poids sont adaptés, ont tendance à ne plus modifier les poids une fois que la sortie de l'unité est assez proche de 0 ou de 1 (pour la sigmoïde asymétrique), ou assez proche de -1 ou de +1 (pour la sigmoïde symétrique). Si l'on observe la fonction de calcul d'erreur, on constate que l'erreur d'une unité est calculée à partir de la multiplication de la dérivée de la fonction d'activation par la différence entre la sortie désirée et la sortie réelle, ou par une valeur proportionnelle à l'erreur obtenue dans la couche suivante. La dérivée de la fonction d'activation donne des valeurs très faibles (proches de zéro) quand, en entrée de cette fonction, on a des valeurs proches des extrémités de la fonction (très négatives ou très positives, voir Figure 14). La valeur de la dérivée de la sigmoïde étant très proche à zéro, l'erreur calculée est elle aussi proche de zéro et par conséquent les poids ne vont plus évoluer. Ceci permet de donner au réseau une certaine stabilité au cours de l'apprentissage, mais peut paralyser l'adaptation des unités du réseau (*flat-spot*);

- *Instabilité et oubli catastrophique* : inversement au problème de la paralysie, les réseaux souffrent du problème de l'instabilité et de l'oubli catastrophique. Les réseaux minimisent l'erreur d'une façon non-coordonnée : chaque unité essaye de réduire l'erreur le plus possible. Le problème de cette approche est que les unités peuvent modifier excessivement leur poids et perdre les connaissances déjà acquises. Cela peut arriver dans deux circonstances : les unités changent beaucoup "d'avis" pendant l'apprentissage (*moving target problem*) ; le réseau perd des connaissances déjà acquises quand on change la base d'apprentissage (oubli catastrophique). Ainsi, dans certains cas, le réseau n'apprend plus de nouvelles connaissances ou détruit presqu'entièrement les connaissances déjà acquises;

- *Vitesse de convergence et paramètres de réglage de l'algorithme* : dans la plupart des cas, la vitesse de convergence de l'algorithme de la Rétro-Propagation est très faible. Les changements des poids doivent être faits très lentement (avec des pas très petits), pour ne pas risquer de ne pas converger vers une solution. L'algorithme de Rétro-Propagation est doté de deux paramètres α et β, qui contrôlent le pas d'apprentissage (*vitesse*) et l'inertie (*momentum*) respectivement. Ces deux paramètres permettent d'accélérer le processus d'apprentissage, mais il faut les régler précisément pour obtenir de bons résultats. Ils sont essentiels dans le processus d'apprentissage, mais posent du même coup le problème de l'estimation des valeurs à leur donner. De plus, les valeurs de α et β sont très liées à la base d'apprentissage employé et doivent être reconfigurés pour chaque nouvelle application. Une valeur de α ou de β mal choisie peut réduire à néant l'effort d'apprentissage.

Enfin, la Rétro-Propagation n'est pas un algorithme incrémental, ni au niveau de la structure du réseau, ni au niveau de la base de données. L'architecture du réseau est statique et reste difficile à définir par rapport à un problème spécifique donné. Malgré tous ces problèmes, l'algorithme de la Rétro-Propagation reste une des méthodes les plus utilisées pour l'apprentissage des réseaux connexionnistes. Par ailleurs, conscients de ces points faibles, plusieurs chercheurs ont essayé de résoudre ces différents problèmes [SCH 94, SCH 93]. On peut mettre en évidence certaines techniques améliorées d'apprentissage : la méthode **RPROP** [RIE 93] (paramètres adaptés au cours de l'apprentissage), la méthode **QuickProp** [FAH 88] (une descente du gradient de deuxième ordre), la méthode du **Gradient Conjugué** - *Scaled Conjugated Gradient* [MOL 90] (une autre technique de deuxième ordre), ou enfin la méthode **CasCor** - *Cascade-Correlation* [FAH 90] et les autres techniques ontogéniques d'apprentissage [FIE 94] (évolution des poids et aussi de l'architecture du réseau pendant l'apprentissage).

Une question souvent soulevée est la suivante : "les réseaux de neurones ne sont-ils qu'une simple réalisation de méthodes déjà connues, à savoir les méthodes statistiques d'analyse de données ? ". Une discussion intéressante sur ce sujet est proposée par B. Orsier [ORS 95] qui conclut que les réseaux restent une approche très intéressante, même s'ils ont des points communs avec d'autres approches, car ils peuvent toujours apporter des contributions au domaine de l'analyse de données. Les réseaux possèdent des propriétés qui leur permettent de surpasser certaines méthodes classiques d'analyse de données, dans certaines situations. Avant d'affirmer qu'une méthode surpasse complètement une autre, il faut bien avoir à l'esprit qu'une méthode peut toujours être meilleure ou aussi performante que toutes les autres pour un cas particulier et unique d'application (on trouve toujours *le cas parfait* pour une méthode), mais il semble impossible d'avoir une seule méthode qui soit plus performante par rapport à un grand ensemble d'applications différentes. Une solution possible semble être la combinaison de différentes méthodes afin d'obtenir des solutions composées plus performantes.

2.4 VERS LES SOLUTIONS HYBRIDES

Nous avons présenté, dans les sections précédentes, les différents méthodes d'apprentissage automatique ainsi que leurs propriétés et limitations. Etant donnés les problèmes inhérents à chacune des différentes approches, les chercheurs essayent de trouver de nouvelles solutions plus performantes. Les solutions hybrides se présentent comme une tendance dans les recherches développées sur les systèmes intelligents [KAN 92, MIC 94, ORS 95, LAL 96, HIL 93, HIL 94]. Les systèmes experts de deuxième génération sont le symbole de cette recherche de solutions hybrides, car on y trouve des systèmes conçus pour permettre l'intégration de différents modèles de raisonnement et d'apprentissage [NIK 97, DAV 93]. Dans cette section, nous allons tout d'abord dresser une brève liste présentant des exemples connus d'hybridation de systèmes puis discuterons sur un type de solution hybride en particulier : l'intégration des approches symbolique et sub-symbolique (connexionniste).

2.4.1 Systèmes et Méthodes Hybrides d'Apprentissage Automatique

Le concept de système hybride ou de méthode hybride est très large. Ce groupe d'applications inclut toute méthode qui intègre deux approches différentes pour la solution d'un problème. Nous allons nous concentrer sur les approches hybrides d'apprentissage automatique, c.-à-d. les différents systèmes et méthodes résultants de la combinaison des méthodes de base présentées dans ce document. Notre idée n'est pas de dresser une liste complète de toutes les possibilités de combinaison d'approches différentes, mais seulement de donner un aperçu de ce qui est fait actuellement en termes de systèmes et méthodes hybrides. Voici quelques exemples d'approches hybrides appliqués à l'apprentissage automatique ou à la conception de systèmes intelligents :

- **Systèmes Symbolique-Flous** : les systèmes symbolique-flous intègrent la logique floue (*fuzzy-logic*) et les systèmes à base de connaissances (*KBS*). Ce type de système est en réalité une extension des systèmes à base de connaissances, dans lesquels on ajoute la possibilité de représenter des règles floues et de les manipuler à travers des mécanismes d'inférence floue. Un exemple de ce type de systèmes est le *Fuzzy-CLIPS* [IIT 96], une version du langage CLIPS adaptée au traitement de la logique floue. Un autre système connu, plus ancien, est le *FLIE* [VES 93].

- **Systèmes Symbolique-Génétiques** : ces systèmes sont normalement composés d'un module génétique responsable de l'acquisition de connaissances à partir des données (apprentissage), et d'un module symbolique responsable du moteur d'inférence symbolique (raisonnement). Des exemples de ce type de systèmes sont les systèmes *DELVAUX* [EIC 96], *COGIN* [GRE 93] et *GABIL* [DEJ 93b].

• **Systèmes Génético-Flous** : une des possibilités d'application des algorithmes génétiques à la logique floue consiste à les utiliser pour l'optimisation des ensembles flous. Un exemple de systèmes génético-flous de ce type est celui décrit par A. Homaifar [HOM 95].

• **Systèmes Neuro-Génétiques** : la plupart des systèmes neuro-génétiques sont conçus afin de contourner le problème relatif au choix de l'architecture du réseau. Les réseaux qui relèvent de ce type d'approche sont aussi appelés réseaux évolutifs (*evolutinary ANN* [POR 97, PAT 97, BAL 95, FLO 94, NOL 94]). L'idée de base repose sur l'utilisation des algorithmes génétiques pour faire évoluer et sélectionner les architectures de RNA plus performantes par rapport à une application donnée. On trouve aussi dans la littérature d'autres types moins répandus d'hybridation neuro-génétique, tels que : adaptation des poids synaptiques réalisée par un algorithme génétique [POT 92], adaptation des paramètres de vitesse de convergence et d'inertie (α et β) par un algorithme génétique [SAL 96].

• **Systèmes Neuro-Symboliques:** ce type de système hybride est sans doute le plus étudié parmi toutes les approches hybrides que nous avons décrites. Les systèmes hybrides neuro-symboliques peuvent être encore divisés en sous-groupe :

- ♦ **Systèmes Neuro-Flous** : les systèmes hybrides neuro-flous sont une des catégories de systèmes hybrides les plus développées car la logique floue et les réseaux connexionnistes ont beaucoup de points en commun. On trouve dans la littérature un grand nombre de références sur ce type de système [JAN 96, KRZ 97, KAS 95, GON 96, NAU 95b, HAY 94, ALC 93, COX 92, POS 92, ROC 92]. Les systèmes neuro-flous sont principallement de trois types : systèmes qui intègrent des règles floues dans des réseaux (le modèle RBF étant un des plus utilisés) ; les systèmes qui font l'extraction de règles floues à partir des réseaux ; et les systèmes qui implementent des neurones flous. On peut citer comme exemple les systèmes suivantes : *FuzzyARTMAP* [CAR 92], *FUN* [SUL 93], *FuzzyCOPE* [KIL 97, KAS 96], *NEFCON-NEFCLASS-NEFPROX* [NAU 97], *ANFIS* [JAN 93] et *FUNEGEN* [HAL 94].

- **Systèmes Neuro-IDT** : les systèmes hybrides qui combinent des réseaux de neurones avec des arbres de décision (IDT) relèvent de deux approches principales : la construction et initialisation d'un réseau à partir d'un arbre de décision [LEE 95, SAH 95], ou l'extraction d'un arbre de décision à partir d'un réseau après la fin de son apprentissage [BOZ 97b, CRA 96a, CRA 96b, MED 95]. L'insertion d'un arbre de décision dans un réseau sert à simplifier le choix de son architecture et de ses poids initiaux. Cette approche apporte une solution alternative au problème de la construction des réseaux. L'autre approche (extraction d'arbres de décision) adresse le problème du manque d'une représentation symbolique des réseaux. L'extraction d'un arbre de décision va permettre de obtenir des explications sur le raisonnement employé pour la solution d'un problème. Ainsi, l'arbre de décision obtenu à partir d'un réseau permet d'entrer dans la "boîte noire" et de mieux comprendre son fonctionnement. En plus de ces deux approches, on trouve aussi dans la littérature des approches vraiment mixtes, où les noeuds de l'arbre de décision peuvent être remplacés par des unités d'un réseau connexionniste [ALC 93, UTG 88].

- **Systèmes Neuro-CBR** : l'intégration d'un réseau connexionniste et d'un système de raisonnent fondé sur des cas permet d'induire des connaissances de plus haut niveau, tout en gardant certaines des avantages des systèmes CBR. Un exemple de ce type d'approche est le système *ProBis* [MAL 96a, MAL 96b]. Ce système permet d'obtenir un réseau avec des prototypes que représentent les cas appris. Il permet aussi de garder les cas atypiques et frontières dans la mémoire du système CBR. De cette façon, on peut profiter au même temps d'une représentation à base de prototypes (cas généralisés) et d'une représentation à base de cas (cas particuliers).

- **Systèmes Neuro-KBS ou SHNS** : les systèmes qui integrent des réseaux connexionnistes et des systèmes à base de connaissances (KBS) sont couramment appelés systèmes hybrides neuro-symboliques (SHNS). Nous allons utiliser plutôt ce dernier terme pour faire référence à tous les systèmes qui permettent d'intégrer des règles symboliques (connaissances symboliques) avec des réseaux connexionnistes (connaissances sub-symboliques). Dernièrement, un très grand nombre de publications a été consacré à ce sujet [SUN 97, NIK 97, MEM 97, HIL 96b, AND 96, SUN 95a, GAL 95, HAL 95, GOO 95, HON 95, MED 95, MED 94, HON 94, FU 94, SUN 94, HIL 94b, BOO 93, KAN 92, DIN 92, MIN 90, et, MAH 96, LAL 96, ORS 95, GIA 92, TOW 91].

On trouve actuellement un bon nombre d'exemples de systèmes SHNS, tels que :

- Le système *SYNHESYS* - KBS + RNA à Prototypes [GIA 92, ORS 95];
- Les réseaux *KBANN* - Règles symboliques + RNA type PMC [TOW 91]
- Le système *INSS* - KBS + RNA type PMC [OSO 95a, OSO 97, AMY 97];
- L'algorithme *EBNN* - EBL + RNA [MIT 97];
- Les réseaux *KBCNN* - Règles symboliques + RNA type PMC [FU 94];
- Le système *SCANDAL* - KBS + RNA type PMC [STU 96];
- Le système *RAPTURE* - Règles + RNA type CFBP [MAH 96, MAH 93];
- Le système *MACIE* - Système expert basé sur les RNA à PMC [GAL 93, GAL 88];
- Le réseau *RUBICON* - Base de règles codés dans RNA (sans apprentissage) [SAM 92];
- Le système *CONSYDERR* - Traitement basé sur des règles et sur la similarité [SUN 95a];
- Le système *CORE* - Interaction entre deux modules, KBS et RNA [KAS 90].

Nous avons fait le choix d'orienter les travaux réalisés dans cette thèse sur ce dernier type de systèmes : les systèmes hybrides neuro-symboliques - SHNS. Notre idée est d'utiliser l'intégration des approches symbolique et connexionniste afin de profiter de la complémentarité existante entre ces deux types d'approches. Dans la section suivante, nous allons présenter certaines des caractéristiques que nous avons souhaité obtenir à travers la réalisation de ce type d'intégration.

2.4.2 Vers les Systèmes Hybrides Neuro-Symboliques

Dans ce chapitre, nous avons présenté différentes méthodes d'apprentissage automatique. Nous avons discuté sur ses particularités, ses propriétés et ses limitations. Il est clair que chacune des approches, considérée isolément, reste assez limitée et impose toujours des restrictions au niveau de l'apprentissage. Afin d'exploiter les avantages particulièrs de chaque approche, symbolique et connexionniste, et de surmonter ses limitations, nous avons décidé d'utiliser une approche hybride. Notre but est d'arriver à des systèmes experts basés sur une approche d'acquisition de connaissances qui satisfasse les items suivants :

• Possibilité d'**exploiter toutes les connaissances disponibles** sur le problème traité, c'est-à dire exploiter les **connaissances théoriques** tout aussi bien que les **connaissances empiriques.** Le système d'apprentissage automatique doit être capable de traiter des bases de règles et d'exemples;

• Possibilité de traiter des problèmes où les connaissances ne sont ni complètes ni tout à fait correctes. Les connaissances disponibles sur un problème sont la plupart du temps des **connaissances imparfaites** et des **connaissances qui évoluent** au cours du temps;

• Possibilité de traiter les **données symboliques** tout autant que les **données numériques**. Les connaissances sont représentées par des informations qui contiennent des variables discrètes et des variables continues. Un système intelligent capable de manipuler des données continues, doit aussi être capable de **manipuler des informations approximatives**;

• Possibilité de représenter des connaissances de bas niveau et de haut niveau. Les connaissances doivent être représentées de telle manière qu'on puisse travailler aussi bien sur un **niveau symbolique** que sur un **niveau sub-symbolique**. Le formalisme de **représentation des connaissances** doit être assez **robuste** pour permettre de coder différentes types de relations existant entre les informations.

En conclusion, nous allons nous diriger vers les systèmes hybrides pour que l'on puisse créer des systèmes experts plus robustes. En particulier, les systèmes hybrides basés sur une approche neuro-symbolique se présentent comme des méthodes d'acquisition de connaissances très intéressantes. La complémentarité existante entre ces deux types d'approches (voir section 3.1), nous amène donc à la création de solutions hybrides neuro-symboliques, qui vont nous permettre de résoudre une grande partie des problèmes que nous avons posés jusqu'ici. Dans le chapitre suivant, nous allons présenter plus en détails les SHNS et ses propriétés, ainsi que nous allons donner des exemples de SHNS qui se proposent à résoudre certains des problèmes qui ont été abordés tout au long de ce chapitre.