UNIVERSITÉ D'ARTOIS

Treillis de Galois et réseaux de neurones : une approche constructive d'architecture des réseaux de neurones

THÈSE

présentée et soutenue publiquement le ...

en vue de l'obtention du

Doctorat de l'Université d'Artois et de l'Université de Yaoundé I

(Spécialité Informatique)

par

TSOPZE Norbert

Composition du jury

Université d'Artois - Faculté des Sciences Jean Perrin - Centre de Recherche en Informatique de Lens – CNRS FRE 2499

rue Jean Souvraz, S.P. 18 F-62307, Lens - Cedex France

Université de Yaoundé I - Faculté des Sciences - Département d'Informatique

BP 812 Yaoundé - Cameroun

Je dédie cette thèse
à ma maman et toute ma famille,
à tous les victimes des catastrophes
et aussi à tous les humanitaires.

Table des matières

Table des figures

Notations

(x, y) : exemple à apprendre.

x : vecteur contenant les valeurs prises par les attributs décrivant un exemple.

x_i : composant numéro i du vecteur x.

y : étiquette de la classe de l'exemple, aussi valeur attendue du réseau.

y_i est le composant numéro i du vecteur y.

N_{obj} : nombre total des exemples.

N_{cl} : nombre de total de classes, taille du vecteur y.

N_{att} : nombre d'attributs utilisés pour chaque exemple, taille du vecteur x.

N_{iter} : nombre d'itérations à faire au cours de l'apprentissage.

ψ : fonction représentant la classe des objets.

ψ^* : approximation de ψ.

Ψ : ensemble total des fonctions ψ^*.

$l(x, y)$: fonction de coût entre x et y.

$d\psi$: dérivée de la fonction ψ.

φ : fonction d'activation des neurones ; et φ' la fonction dérivée de φ.

E : erreur en sortie d'un neurone.

E_i erreur en sortie du neurone i.

E_i^l : erreur en sortie du neurone i de la couche l.

e_i : état interne du neurone i.

W : vecteur de poids de connexion.

w_i poids de connexion de la connexion i.

$w_{i,j}$ connexion entre le neurone i et le neurone j.

θ : seuil d'activation du neurone ; θ_i seuil d'activation du neurone i.

C : contexte formel.

O : ensemble des objets du contexte formel, O_1 sous-ensemble de O ;

A : ensemble des attributs du contexte formel, A_1 sous-ensemble de A.

f (resp. g) : application qui associe à un ensemble d'objets (resp. attributs) l'ensemble d'attributs (resp. objets) qui vérifient ces objets (resp. attributs).

c : concept formel.

cl : attribut dénotant la classe ;

cl_i classe numéro i.

H : nombre maximal de couches.

L : ensemble ordonné de concepts formels.

NN : réseau de neurones.

n : étiquette du neurone de sortie pour lequel les règles sont extraites.

β_p (resp. β_n) : nombre des connexions de poids positifs.

P_p (resp. S_n) : ensemble constitué des sous-ensembles de connexions positives (resp. négatives).

P (resp. Q) : élément de S_p (resp. S_n).

U : ensemble des connexions en entrée d'un neurone.

V_0 : sous-ensemble de U.

V : vecteur binaire permettant de représenter un sous-ensemble de U.

RNA : Réseau de Neurones Artificiels.

CLANN : Concept Lattice-based Artificial Neural Network.

MCLANN : Multiclass Concept Lattice-based Artificial Neural Network.

Introduction générale

Contexte

Les applications informatiques et Internet produisent des masses de données. L'exploitation de ces données est appelée fouille de données [HK01, CM10, KZ02] et vise à y extraire de nouvelles connaissances. L'exploitation des ces connaissances aide les décideurs à faire des choix sur l'orientation de la politique de l'entreprise. Les connaissances extraites peuvent être sous forme de corrélation entre attributs, ou de fonction permettant de prédire un résultat inconnu et fournissant une aide à la prise de décision. La classification supervisée [HK01, DSM$^+$02] est l'une des applications majeures de la fouille de données. Elle consiste à construire des fonctions ou modèles (appelés classifieurs) à partir des données permettant de prédire des classes pour les futures données. Par exemple, les clients d'une banque peuvent être classés entre les "*bons*" clients (à qui le crédit a été accordé) et les "*mauvais*" clients (à qui le crédit a été refusé). Le classifieur définit à partir d'un ensemble de clients connus comme *bons* ou *mauvais*, un modèle qui aidera le gestionnaire à prédire si un futur client est *bon* ou *mauvais*.

La littérature présente un nombre important de techniques pour résoudre les problèmes de classification supervisée. Dans [HK01], les auteurs proposent les critères de comparaisons des classifieurs. Ces critères sont :

1. La capacité du modèle à prédire correctement la classe d'un objet (*généralisation*).
2. Le coût algorithmique de construction et d'utilisation du modèle (*vitesse*).
3. L'habilité du modèle à bien se comporter en présence des données bruitées et des valeurs manquantes (*robustesse*).
4. Le comportement du classifieur face aux données volumineuses (*passage à l'échelle*).
5. Le niveau de compréhension des résultats obtenus du modèle (*interprétabilité*).

Plusieurs autres travaux de recherche [TP90] comparent les classifieurs ; mais la majorité des travaux se focalisent sur les deux premiers critères [NB07, PYH00] alors que les critères énumérés dans [HK01] permettent d'analyser plusieurs aspects des modèles.

Les réseaux de neurones ou réseaux de neurones artificiels (RNA) [Hay99, DN93, MR91, Roj96], ou encore réseaux connexionnistes [Ben06] sont une technique largement utilisée pour résoudre les problèmes de classification. Un RNA est un système qui adapte son comportement en fonction des données qu'il reçoit dans le but de bien approximer la fonction présentée par ces données. Ces éléments sont des neurones reliés entre eux par des liens de connexion et qui communiquent entre eux à travers ces liens. Les RNA sont utilisés dans diverses applications [DSM+02, HKP91] : en classification [Sch94, DSM+02, HKP91], en reconnaissance des formes [Sch94], en reconnaissance des caractères manuscrits [MKS+90, Gos96], en économie [Sha02], en banque [Wit99], en biologie [TR07], dans l'approximation des fonctions [Cyb89], à la compression d'images [KMN08]... Des études théoriques et expérimentales faites sur les RNA montrent que parmi les critères cités dans [HK01], ils satisfont la généralisation (travaux de schmidt [Sch94], de Parekh [PYH97a], de Craven [Cra96] ; ils sont traités d'approximateurs universels [HSW89, Bar93]), la robustesse [HKP91, FS92, Roj96], relativement la vitesse et le passage à l'échelle. Ils présentent un bon comportement en absence de connaissances du domaine.

Les RNA restent limitées quant au critère d'*interprétabilité* [ADT95] ; il est difficile pour l'utilisateur d'expliquer comment le résultat mis en sortie du RNA a été calculé.

Problématique

Les limites des RNA au critère d'interprétabilité est l'une des causes majeures qui empêchent son utilisation dans certains domaines [ADT95, Cra96, DSZ04]. Ils sont considérés comme des "boites noires" [HK01, DSM+02] parce qu'il est très difficile pour un concepteur de justifier son résultat face à une situation. Alors, pour ce dernier il n'est pas facile de trouver une interprétation à des résultats obtenus en sortie du RNA. Cette interprétation peut permettre de découvrir certaines corrélations entre les attributs et être utilisées dans d'autres systèmes.

Les travaux de recherche [ADT95, Boz95, SL00, DGBG01, KM05] proposent de décrire le comportement du RNA par un ensemble de règles extraites de ce dernier. Mais il se pose alors un problème du choix [Dar01, CS97] et du nombre souvent très élevé des règles [CS97].

Au problème d'interprétabilité des RNA s'ajoute la difficulté de trouver une architecture adéquate et optimale pour la résolution d'un problème donné. En effet, il n'existe aucune démarche méthodique permettant aux concepteurs de définir une architecture optimale du réseau de neurones pour la résolution d'un problème donné [CM10, HK01]. Face à un problème, l'utilisation des réseaux de neurones suscite beaucoup de questions auxquelles le concepteur doit apporter une réponse : *Quel est le nombre de couches du réseau ? Quel est le nombre de neurones par couche ? Comment les connecter ? Comment initialiser les poids de connexions ?* Les

réponses à ces questions concourent à l'élaboration de l'architecture du réseau de neurones.

Quelques travaux de recherche proposent des solutions pour la résolution aux problèmes de définition d'architecture de RNA [FL90, Gal90, Fre92b, TS94, YPH96, PYH97a, YPH99, CO02]. Dans ces travaux, nous distinguons deux philosophies majeures : Une première philosophie [TS94] propose de définir l'architecture du réseau en utilisant un ensemble de connaissances (représentées sous forme de règles) fournies sur le domaine du problème à résoudre. Une deuxième philosophie [PYH97b] propose de définir cette architecture à partir des données ; de manière incrémentale par ajout de nouveaux neurones au réseau jusqu'à l'obtention des performances désirées. La première philosophie construit une architecture transparente à l'utilisateur car ce dernier peut expliquer la présence de chaque neurone comme représentant une variable de l'ensemble des règles et chaque connexion entre deux neurones comme une relation entre deux variables de l'ensembles de règles, mais s'avère impossible à l'utiliser quand on ne dispose pas de connaissances apriori. La deuxième philosophie est mieux adaptée quand le concepteur ne dispose pas de connaissances, mais aucune justification ne peut être donnée à la structure du RNA qu'elle construit.

Les questions que nous abordons dans cette thèse sont les suivantes : "*Quelles sont les méthodes proposées pour construire un RNA permettant de résoudre un problème de classification ? Comment construire un réseau de neurones transparent à l'utilisateur en l'absence de connaissances apriori ? Comment améliorer l'interprétabilité des RNA construits ?*"

Contribution

Cette thèse contribue à la résolution des problèmes suscités : aider les concepteurs à définir une architecture de RNA pour résoudre un problème et contribuer à l'amélioration de l'interprétabilité des RNA par l'extraction de connaissances incorporées dans le réseau. Les méthodes que nous développerons seront appliquées à la résolution des problèmes de classification supervisée.

Notre contribution se présente sous quatre points :

1. Une étude comparative théorique et expérimentale des méthodes de construction des architectures de RNA pour la résolution des problèmes de classification [TMNT07b]. Cette contribution présente une étude que nous avons faite sur les méthodes de construction d'architecture de RNA présentées par la littérature. Les expériences ont été faites sur les données prises sur la base UCI [NHBM98].
2. Une proposition d'une méthode de construction des architectures de RNA à partir des treillis de Galois [Bir67, BM70, GR99] pour la la classification à deux classes. Nous avons ainsi défini les modèles CLANN (Concept Lattice-based Artificial Neural Network)

[TMNT07c, TMNT07a, TMNT08]. Nous apportons alors une aide au concepteur pour la construction d'une architecture en l'absence de connaissances apriori. Nous verrons ensuite que l'architecture ainsi construite est transparente comme celle obtenue de la première philosophie de construction des RNA. Cette méthode a l'avantage de construire le RNA compréhensible en absence de connaissances apriori.

3. Une proposition d'une méthode de construction des architectures de RNA à partir des treillis de Galois pour le cas multi-classes. La méthode CLANN étant limitée aux problèmes à deux classes, nous avons développé MCLANN (Multiclass Concept Lattice-based Artificial Neural Network) [MNTT08, MNTT09] permettant d'utiliser les treillis de Galois pour construire les RNA capables de classer en N_{cl} classes ($N_{cl} \geq 2$).
4. Une proposition d'une nouvelle approche d'extraction de règles à partir des RNA. Cette approche est appelée approche des MaxSubsets [Tso09]. Elle permet d'extraire des ensembles de règles sous les deux formats proposés par les chercheurs. Ce qui n'était pas le cas (à notre connaissance) pour des algorithmes connus.
5. Nous proposons aussi une explication du résultat émis en sortie par le RNA. Pour un exemple placé en entrée au RNA, nous expliquons la sortie obtenue du RNA par la combinaison des différents neurones cachés actifs.

Plan

Cette thèse est organisée de la manière suivante :

Le chapitre 1 : Généralités - préliminaires

Ce chapitre présente les notions préliminaires permettant de comprendre le problème de classification et la technique des RNA. Nous y présenterons dans la première partie le problème de classification supervisée ; le processus général de résolution et quelques méthodes de résolution basées sur d'autres techniques que les RNA. Dans la deuxième partie, nous présenterons les notions de base sur les réseaux de neurones et les algorithmes d'apprentissage permettant de modifier ses paramètres à partir des données pour approximer une fonction.

Le chapitre 2 : Construction des réseaux de neurones multicouches

Dans ce chapitre, nous faisons une revue de littérature des algorithmes proposés pour la construction des réseaux de neurones. Nous présentons les méthodes appartenant aux deux philosophies de construction de RNA. Une étude comparative théorique et expérimentale de ces méthodes est aussi présentée.

Le chapitre 3 : Contribution de treillis de Galois dans la construction de réseaux de neu-

rones

Ce chapitre introduit les treillis de Galois et sa contribution à la construction des réseaux de neurones. Nous y présentons les deux systèmes CLANN et MCLANN que nous avons conçus pour définir la structure des réseaux de neurones. Des résultats expérimentaux obtenus de l'expérimentation des deux systèmes sont aussi présentés. Le chapitre se termine par une comparaison de CLANN et de MCLANN à d'autres méthodes.

Le chapitre 4 : Règles dans les réseaux de neurones

Ce chapitre aborde l'interprétabilité des RNA. Nous commencerons par la présentation des méthodes d'extraction de règles à partir des RNA. Ensuite, la méthode des MaxSubsets que nous avons développée est aussi présentée. Ce chapitre s'achève par l'extraction des règles dans les réseaux construits avec l'approche CLANN et MCLANN et l'explication des résultats émis en sortie de ces réseaux.

En conclusion, nous résumerons les résultats de cette thèse. Et nous présentons aussi quelques perspectives pour ce travail.

Chapitre 1

Préliminaires - Généralités

Sommaire

1.1 Introduction

Le mot 'Classification' (en français) désigne à la fois la classification supervisée (classification en Anglais) et la classification non supervisée (Clustering en Anglais) [MNN05]. Elle est dite automatique lorsqu'elle est faite par une machine, c'est-à-dire qu'elle est réalisée par un programme informatique. La classification est supervisée si les données sont fournies sous forme de couple (x, y) où x est le vecteur d'attributs et chacun de ses composants a pris une valeur permettant de décrire un objet ou une situation, et y est l'étiquette de la classe de l'objet x. Le système de manière automatique construit un modèle capable d'associer la classe y au vecteur x pour chaque couple (x, y). Quant à la classification non supervisée, les valeurs du composant y des vecteurs ne sont pas fournies ; le système construit un modèle qui de manière automatique regroupe les objets similaires par catégorie. Dans cette thèse, nous nous intéresserons à la classification automatique supervisée. Nous utiliserons le mot *classification* pour parler de *classification automatique supervisée*.

Nous pouvons aussi définir la classification comme étant le processus de recherche d'un ensemble de modèles (ou fonctions) qui décrivent et distinguent les classes d'objets [DSM$^+$02, HK01] ; ces modèles construits à partir d'un ensemble d'objets de classe connue seront capables de prédire la classe des autres objets.

Un processus de classification comprend quatre étapes : le prétraitement éventuel, la construction du modèle, l'évaluation du modèle construit et son utilisation. Le modèle défini peut être représenté sous diverses formes : les réseaux de neurones, les arbres de décisions, les k-plus proches voisins, les réseaux bayésiens, les séparateurs à vaste marge (SVM), etc. Ce chapitre a pour objet la pésentation de la classification supervisée, des méthodes de résolution des problèmes de classification supervisée et des généralités sur les réseaux de neurones artificiels.

La prochaine section présentera la classification supervisée, le fonctionnement et l'évaluation d'un système de classification supervisée. Dans la troisième section, nous présenterons les méthodes de classification. Les réseaux de neurones feront l'objet de la quatrième section, ses concepts y seront présentés.

1.2 Classification automatique

Nous traiterons la classification comme la tâche de fouille de données qui consiste à prédire avec une certaine précision la classe y de chaque exemple décrit par x.

1.2.1 Le problème de classification

Le problème de la classification se pose de la manière suivante : étant donné un ensemble d'exemples ou d'observations $o_1, o_2, ..., o_{N_{obj}}$, décrits par les attributs $x_1, x_2, ..., x_{N_{att}}, y$ (où y est un attribut particulier appelé *classe*), comment trouver une fonction (ou un ensemble de fonctions) ψ telle que $\psi(x_1, x_2, ..., x_{N_{att}}) = y$ pour tout objet ?

Définition 1 *Un **exemple** est une description d'une situation donnée, d'un phénomène ou d'un objet.*

Un exemple est décrit par un ensemble d'attributs et chaque attribut ayant pris une valeur précise. Ces valeurs décrivent une situation donnée du phénomène étudié. Il peut être aussi vu comme un vecteur d'attributs, chaque composant du vecteur ayant pris une valeur. Un exemple est aussi appelé objet ou vecteur d'entrée. Ces exemples sont décrits par des attributs qui peuvent être numériques, symboliques, multimédia,... Le type permet de définir les différentes opérations qui s'appliquent sur l'attribut. L'ensemble des attributs constitue l'ensemble des variables d'entrée aux algorithmes de traitement de ces données.

Exemple 1 *Le tableau 1.1 est un ensemble de données décrivant les animaux dont les noms figurent à la colonne 'Animal'. Chaque animal est décrit avec les attributs 'Taille', 'Poids', 'Nombre' et 'déplacement'. L'attribut 'Taille' mesure la taille de l'animal en mètres, 'Poids' son poids en kilogrammes, 'Nombre' est le nombre de Pattes qu'il possède et 'Déplacement' la façon avec laquelle l'animal peut se déplacer. Ces animaux sont classés suivant le type représenté par la colonne 'Type'.*

Un exemple d'objet à classer est 'Chien'. Le vecteur x permettant de le décrire est (0,5 ; 30 ; 4 ; marche) et y sa classe est 'terrestre'.

A partir de cet exemple de données, les biologistes peuvent classer ces animaux comme 'terrestre', 'aérien' ou 'aquatique'. La classification automatique recherche alors le modèle capable de classer automatiquement chaque animal décrit par ces attributs avec une erreur minimale sur l'ensemble.

Animal	taille	Poids	Nombre	déplacement	Type
Moineau	0,15	0,25	2	Vol, marche	aérien
Chien	0,5	24	4	marche	terrestre
Mouton	0.5	35	4	marche	terrestre
Vache	1,5	150	4	marche	terrestre
Aigle	1	6.5	2	vol,marche	aérien
Hippopotame	3.5	350	4	nage, marche	aquatique
Maquereau	0,3	2	0	nage	aquatique
Homme	1.7	75	2	marche	terrestre

TAB. 1.1 – Exemple de données prélèvées sur les animaux.

La **classification** est une opération très pratiquée par l'Homme. Elle permet à ce dernier de juger, de catégoriser, de sélectionner... Ses applications sont multiples :

- En marketing, classer les clients peut permettre de prédire quels sont les potentiels intéressés d'un nouveau produit et aider alors à la définition d'une politique commerciale orientée vers ces clients.
- En informatique, classer les problèmes en problèmes NP-Complets, NP-Difficiles et Polynomiaux permet de prédire le coût algorithmique de leurs résolutions.
- En météorologie, classer les régions en vigilance verte, jaune, rouge ou orange permet d'attirer l'attention des habitants et des services de secours sur les probables risques.
- En biologie, classer les êtres vivants en espèces permet de connaître dans quel milieu naturel ils vivent et de prévenir les risques de leur disparution liés à la dégradation de cet espace.

Nous schématisons le processus de classification par la figure 1.1. Les données saisies par les opérateurs ou les utilisateurs sont intégrées et enregistrées dans des supports de stockage. Ensuite ces données peuvent éventuellement subir un prétraitement avant d'être utilisées pour initialiser, apprendre et tester les modèles. Les mesures obtenues de l'apprentissage et de l'évaluation du modèle permettent de connaître le degré de confiance à accorder au modèle.

1.2.2 Etapes d'un processus de classification

La classification est un processus allant du prétraitement des données à la proposition d'un modèle à l'utilisateur et l'évaluation du modèle proposé. Nous proposons de découper ce processus en quatre étapes : le prétraitement éventuel des données, la construction du modèle (définition, apprentissage et la validation), l'évaluation du modèle et l'utilisation du modèle

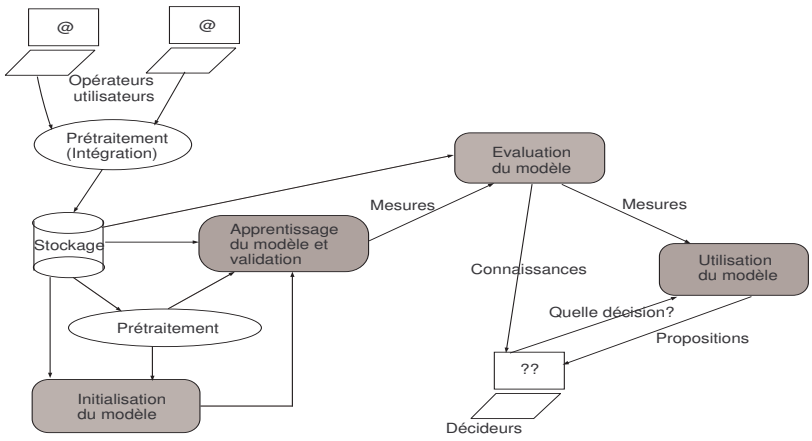

FIG. 1.1 – Différentes étapes du processus de classification.

construit.

a- Le prétraitement

Les données ne se présentent pas toujours comme le souhaitent les utilisateurs. Le problème est surtout lié à : la présence des bruits (erreur introduite dans la mesure d'une variable) dans les données, la présence de valeur inconnue pour certains attributs (données manquantes), l'inconsistance des données, l'incohérence des données et le volume élevé des données. Le but du prétraitement [HK01, KZ02] est d'améliorer la qualité des données et par conséquent les résultats des algorithmes qui utilisent ces données ; d'améliorer l'efficacité des algorithmes et la facilité de traitement par ces algorithmes. Le prétraitement regroupe [CM10] :

- L'intégration des données : ce processus permet de regrouper les données provenant de plusieurs sources et d'uniformiser le format.
- Le nettoyage : il s'agit de supprimer les bruits et de corriger les inconsistances.
- La transformation des données : elle consiste à mettre les données sous un type plus approprié aux algorithmes de traitement. Elle intègre les opérations de :
  * normalisation : elle consiste à transformer les valeurs d'un attribut afin que celles-ci puissent appartenir à un domaine.
  * binarisation : elle consiste à transformer les données de type numérique (réel ou entier) ou symbolique (caractères ou énuméré) en données binaires.
  * discrètisation : elle consiste à transformer les données continues d'un attribut en valeurs discrètes.
  * projection : elle consiste à ajouter à chaque exemple un attribut supplémentaire (appelée projection) dont la valeur est calculée comme les sommes des carrés des autres

attributs [PYH95, PYH97a]. Le but est de éviter la dépendance du modèle à l'attribut dominant.

– La réduction des données : elle consiste à éliminer les redondances dans les données, regroupe ces données et fournit les moyens permettant de sélectionner les attributs pertinents et d'éliminer les attributs non pertinents.
– Le traitement des valeurs manquantes : elle consiste à remplacer les valeurs non renseignées dans les exemples.

Nous utilisons ces méthodes pour apprêter les données utilisées au cours de l'expérimentation des différents algorithmes. Les effets liés au prétraitement ne sont pas traités dans le cadre de cette thèse car notre intérêt porte sur la construction et l'évaluation du modèle.

Dans un processus de classification, après le prétraitement, l'ensemble de données est généralement séparé en trois sous-ensembles : un sous-ensemble d'apprentissage, un ensemble de validation et un sous-ensemble de test. Les deux premiers sous-ensembles sont utilisés pour la construction du modèle alors que le troisième est utilisé pour l'évaluer.

b- Construction du modèle

La construction du modèle de classification est la phase principale d'une tâche de classification. Pour la classification avec les RNA, nous pouvons décomposer cette étape en trois phases successives : l'initialisation du modèle, l'apprentissage du modèle et la validation.

L'initialisation du modèle permet de lui donner une architecture initiale. Cette tâche varie suivant les techniques utilisées pour construire les classifieurs. Ces techniques seront présentées dans la section 1.3. Le modèle construit peut alors être appris pour prédire les objets de l'ensemble d'apprentissage.

Phase d'apprentissage

En science cognitive, l'apprentissage est un processus qui doit faire de telle sorte qu'un individu réalise de manière autonome une tache donnée [DSM$^+$02]. Elle est intuitive et claire pour les Hommes et les animaux. Cette définition correspond à l'apprentissage humain ou animal.

Au niveau machine, on parle d'apprentissage automatique ou artificiel, elle permet de rechercher les paramètres convenables d'un modèle décrivant une réalité à partir des exemples. L'apprentissage automatique est défini dans [CM10] comme la recherche de l'hypothèse que vérifient les données avec une erreur minimale. Nous utiliserons l'algorithme d'apprentissage défini de la manière suivante :

Définition 2 *L'algorithme d'apprentissage est un algorithme qui permet d'ajuster les paramètres d'une ou plusieurs fonctions pour décrire un phénomène avec une précision désirée ; cet ajustement utilise les observations décrivant certaines situations du phénomène [DSM+02].*

Un problème d'apprentissage met en relation trois composants [HK01, CM10] :

1. L'environnement. Il est représenté par les objets x et de leurs classes y. Ces objets sont des exemples à apprendre et sont tirés suivant une distribution de probabilité inconnue. Les exemples (x, y) décrivent un phénomène ; ce dernier est une fonction inconnue ψ telle que $\psi(x) = y$; pour tous les couples (x, y).
2. L'apprenant. C'est le composant qui doit être capable d'établir à partir des différents exemples (x, y) une fonction ψ^* telle que $\psi^*(x) = y$ qui approxime au mieux ψ. On parle de l'apprentissage humain quand l'apprenant est un Homme et d'apprentissage automatique ou artificiel quand il est un programme exécutant un algorithme.
3. L'oracle ou le superviseur. Il compare pour chaque exemple décrit par x la réponse de l'apprenant à celle attendue y. Il permet alors d'évaluer le niveau de l'apprenant.

On distingue généralement trois formes d'apprentissage :

1. Lorsque ce processus est fait à partir des données (sans utiliser connaissance sur le problème), on parle d'apprentissage à partir des instances ou d'apprentissage empirique. Les données (x, y) sont présentées à l'apprenant et ce dernier déduit intuitivement la règle décrivant les valeurs de y en fonction de x.
2. L'apprentissage peut aussi se faire à partir des connaissances : on dispose d'un ensemble de connaissances ou des règles décrivant les objets d'une classe et l'apprenant utilise ces règles pour définir le modèle.
3. On parle d'apprentissage hybride lorsque les deux techniques précédentes sont combinées. Un ensemble de règles est d'abord présenté à l'apprenant et il utilise l'ensemble de données présenté ensuite pour raffiner ces règles.

L'apprentissage est un problème d'optimisation qui cherche à minimiser l'écart entre la fonction choisie par l'apprenant et celle que ce dernier aurait dû choisir. En effet l'ensemble des fonctions possibles représente l'ensemble des hypothèses [CM10]. Formellement l'apprenant doit trouver une fonction ψ^* qui minimise le risque réel R_{reel} et donc vérifie l'équation 1.1.

$$\psi^* = ArgMin_{\psi \in \Psi} R_{reel}(\psi) \tag{1.1}$$

Le risque réel d'une hypothèse ψ est l'espérance mathématique du coût de choisir y parmi le hypothèses. Il est calculé suivant la formule 1.2.

$$R_{reel}(\psi) = \int_{X \times Y} l(x, \psi(x)) d\psi(x, x) \tag{1.2}$$

où $l(x, y)$ est le coût de prendre $y = \psi(x)$.

Or il est très difficile de connaître le risque réel lié à chaque hypothèse ; pour cela, on utilise le risque empirique R_{emp} à la place du risque réel. Le risque empirique est la perte moyenne mesurée sur l'échantillon. Il est calculé par la formule 1.3.

$$R_{emp}(\psi) = \frac{1}{m} \sum_{x \in X} l(x, \psi(x)) \tag{1.3}$$

Phase de validation

Dans un problème de classification, évaluer le risque empirique se ramène à calculer une fonction de la différence entre ce que trouve le modèle et ce qu'il aurait dû trouver. Dans notre cas, nous évaluons plutôt le taux d'exemples bien classés ou taux de validation. Si $N_{vc} = \sum_{i=1}^{N_v} |y_i - \psi(x_i)|$ (la fonction de coût $l(x, \psi(x)) = |y - \psi(x)|$) représente la fraction des N_v exemples de validation bien classés alors l'erreur de validation E_v et le taux de validation T_v sont calculés respectivement par les formules 1.4 et 1.5. N_{vc} compte le nombre d'exemples bien classés, les sorties sont binaires (0 et 1).

$$E_v = \frac{N_v - N_{vc}}{N_v} \tag{1.4}$$

$$T_v = \frac{N_{vc}}{N_v} = 1 - E_v \tag{1.5}$$

Cette phase permet d'évaluer les performances du processus d'apprentissage, de vérifier s'il converge, d'arrêter l'apprentissage et de prévenir les risques de surapprentissage.

Définition 3 *On dit que l'apprentissage* ***converge*** *lorsque l'erreur de validation décroît au cours du processus et tend vers une valeur minimale fixée.*

La validation utilise l'ensemble de validation pour évaluer l'erreur de validation et l'ensemble d'apprentissage pour évaluer le risque empirique lié au modèle en cours. La recherche des meilleurs paramètres alterne apprentissage et validation pendant plusieurs itérations jusqu'à ce que le risque empirique devienne infiniment petite (inférieure à un seuil fixé par l'utilisateur). Le surapprentissage survient quand le risque empirique décroît alors l'erreur de validation s'accroît.

Dans nos expériences, la validation est faite avec l'ensemble d'apprentissage. Les données sont divisées en deux sous-ensembles disjoints : l'ensemble d'apprentissage (utilisé pour l'apprentissage et la validation) et l'ensemble de test. Nous utiliserons 'apprentissage' pour parler de l'apprentissage et de la validation.

c- Phase de test

Pendant cette phase, le modèle (ψ^*) appris est utilisé pour prédire des classes aux objets de l'ensemble de test. Le test permet d'évaluer le système sur les données inconnues du modèle.

La mesure

Parmi les mesures utilisées pour évaluer les modèles d'apprentissage, nous avons choisi le taux de généralisation. Il permet de tester la capacité du modèle à généraliser sur les exemples de test, et de mesurer le degré de confiance à accorder au modèle.

Le taux de généralisation (noté $Taux$) est le rapport entre le nombre d'exemples bien classés par le modèle et le nombre total d'exemples présentés au modèle. Il peut être aussi appelé *Taux de classification* et est calculé suivant la formule 1.6.

$$Taux = \frac{N_{tc}}{N_t} \times 100\% \tag{1.6}$$

où N_{tc} est le nombre d'exemples bien classés par le modèle et N_t le nombre total d'exemples de l'ensemble de test.

Techniques d'évaluation

Plusieurs techniques utilisées pour diviser l'ensemble de données en ensemble d'apprentissage et de test sont proposées dans la littérature [HK01, KZ02]. Parmi ces techniques, nous pouvons citer :

1. La validation croisée d'ordre k. Elle consiste à partitionner les exemples en k sous-ensembles $S_1, S_2, ..., S_k$ disjoints de taille presque égale ; à l'itération i, le sous-ensemble S_i est utilisé pour faire le test du modèle construit et appris avec les $k-1$ autres sous-ensembles. Le taux de généralisation du système est calculé comme la moyenne des taux de généralisation obtenus au cours des k itérations. Et le modèle proposé à l'utilisateur est celui qui a eu les meilleurs performances (taux de généralisation, temps de calcul, ...)
   Une variante de la validation croisée est le "leave-one-out" où k est égal à la cardinalité de l'ensemble d'apprentissage.
2. Le "holdout". Contrairement à la validation croisée, l'ensemble des exemples est divisée aléatoirement en deux parties ; une partie permet de faire l'apprentissage du modèle et l'autre partie est utilisée pour tester le modèle. Le taux de généralisation du modèle est celle obtenue sur l'ensemble de test.
3. La resubstitution. Le même ensemble est utilisé pour construire et pour tester le modèle. Elle est utilisée lorsque la taille de l'ensemble d'apprentissage est assez réduite.

Le lecteur peut se référer à [HK01] pour plus de détails sur les méthodes d'évaluation des systèmes de classification.

Nous présentons à la suite des étapes de classification, les techniques utilisées pour construire les modèles de classification. Nous comparerons les résultats expérimentaux de ces techniques aux résultats obtenus des RNA.

1.3 Méthodes de classification supervisée

Après le prétraitement, diverses méthodes peuvent être utilisées pour traiter ces données afin de proposer de l'aide aux décideurs. Les méthodes de classification diffèrent suivant la fonction construite pour classer les exemples et la manière de la construire. En classification, nous distinguons : les plus proches voisins, les treillis de Galois, les arbres de décision, les réseaux bayésiens, les SVM, les réseaux de neurones ... Ces méthodes sont très utilisées dans la résolution des problèmes de classification supervisée ; nous comparerons les résultats expérimentaux des modèles neuronaux à ceux des modèles basées sur ces méthodes.

1.3.1 Les plus proches voisins

La méthode des k-plus proches voisins k-NN [CH67] est une méthode d'apprentissage à partir des instances basée sur la notion de voisinage. Elle consiste à prendre en compte les k échantillons d'apprentissage les plus proches du nouvel exemple (à classer).

L'algorithme d'apprentissage consiste à stocker les exemples de l'ensemble d'apprentissage. Pendant la phase de test, les distances entre les exemples stockés et l'exemple à classer sont calculées et les k plus proches de ses voisins (exemples) sont sélectionnés.

La classe d'un exemple inconnu est celle majoritaire (la classe à laquelle le plus grand nombre d'exemples appartient) dans les k plus proches voisins sélectionnés. L'inconvénient est que la classe la plus fréquente a tendance à dominer.

Le choix de k dépend des données. Il peut se faire par des méthodes heuristiques telles que la validation croisée ou des algorithmes génétiques.

Le fait que les exemples soient stockés en mémoire (pour la phase d'apprentissage) limite cette approche et le rend applicable seulement sur les données de faible taille. En plus de cette

limite, cette approche perd sa qualité en présence des données bruitées.

1.3.2 Les treillis de Galois

La classification à l'aide des treillis de Galois [BM70, Bir40, Bir67] consiste à organiser sous forme de treillis les hypothèses résultant de la phase d'abstraction des données. Les treillis de Galois permettent de représenter des classes non disjointes sous-jacentes à un sous - ensemble d'objets décrits par un ensemble d'attributs [MNN05]. Ces classes aussi appelées concepts formels représentent une idée générale que l'on a d'un objet. Un concept est défini formellement par une extension (sous-ensemble d'objets) et une intension (sous-ensemble d'attributs).

L'apprentissage à l'aide des treillis de Galois consiste à rechercher des concepts dans un espace des hypothèses. Plusieurs approches permettant de faire de la classification à l'aide des treillis de Galois sont étudiées et présentées dans [FFNMN04, MNN05] :

- L'approche incrémentale. Le treillis est mis à jour par ajout de nouveaux objets : des connexions redondantes sont supprimées, de nouveaux nœuds sont ajoutées. Cette approche est à la base des systèmes de classifications GRAND et GALOIS.
- L'approche descendante. On utilise les mesures fréquence, validité, cohérence,...pour sélectionner certains concepts du pseudo-treillis. LEGAL est un exemple de systèmes de classification basée sur cette approche.
- Générer des règles qui pourront être utilisées pour catégoriser un nouvel exemple. Tel est le cas pour le système RULEARNER.

La phase d'apprentissage continue par induire les règles de classification que les nouveaux exemples devront vérifier pour se voir affecter une classe.

Le classement se fait aussi de plusieurs façons :

- Vote majoritaire. L'exemple est affecté à la classe la plus probable en s'appuyant sur les hypothèses vérifiées par cet exemple.
- Utiliser les mesures de similarité entre le nouvel exemple et les concepts sélectionnés. Cette mesure se définit comme le nombre de propriétés du concept vérifiées par l'objet.

La complexité asymptotique exponentielle en temps de construction du treillis et le choix des concepts à utiliser dans la phase de classement sont les inconvénients majeurs de cette technique. Ses avantages sont multiples : extraire des règles permettant de décrire chaque classe, trouver les corrélations cachées entre les attributs et visualiser des exemples et les attributs qu'ils vérifient ; l'utilisateur peut facilement introduire des mesures pour sélectionner certains concepts.

1.3.3 Les arbres de décision

Il s'agit d'une représentation graphique du processus de classification [Mic93, Qui86, Qui93]. C'est un outil d'aide à la décision et à l'exploration des données. Dans un arbre de décision, les correspondances suivantes sont établies :

- Le nœud interne (et la racine) représente un test. Le test consiste à comparer les valeurs des attributs d'un exemple à ceux affectés au nœud. La racine est un nœud de test.
- Une branche est une réponse au test. Les différentes réponses au test (différentes valeurs que peuvent prendre l'attribut du nœud) sont utilisées pour étiquetter les branches.
- Les feuilles représentent les étiquettes des classes.

La construction de l'arbre suit une approche descendante DPR (Diviser-Pour-Régner). Ce processus est décrit par les étapes suivantes :

1. Initialement un nœud représentant tous les exemples est introduit dans l'arbre.

2. Si tous les exemples appartiennent à une seule classe, ce nœud devient une feuille et est étiquettée par cette classe.

3. Sinon l'algorithme utilise une heuristique pour déterminer l'attribut qui sépare au mieux les exemples dans les classes. L'attribut sélectionné sert d'attribut de comparaison du test pour le nœud :

   - Une branche est créée pour chaque valeur de l'attribut et l'ensemble d'apprentissage partitionné suivant ces valeurs.
   - Le même processus est repris récursivement dans chaque partition.

Le classement d'un nouvel exemple se fait par le test de ses attributs à l'arbre. Ceci permet d'établir un chemin allant de la racine à une feuille ; on lui affecte alors la classe qui étiquette cette feuille.

Les inconvénients des arbres de décision sont le fait que la construction de l'arbre devient très complexe en présence de grands volumes de données ; le fait que les données continues doivent être discretisées avant d'être utilisées et les mauvaises performances en présence des données bruitées. L'avantage des arbres de décision est la possibilité d'obtenir des règles de classification qui régissent son processus de classification. Les règles 'si condition alors conclusion' sont ainsi obtenues en décrivant les différents chemins allant de la racine à toutes les feuilles de l'arbre. La classe correspondante à la feuille est la conclusion de la règle alors que le reste du chemin formera la prémisse.

1.3.4 Les réseaux bayésiens

Cette technique [Pea86, Pea00] de classification est basée sur le calcul de probabilités. Elle consiste à déterminer la probabilité pour qu'un exemple appartienne à une classe en utilisant le théorème de Bayes [Ber58]. Pour un problème de classification, cette approche consiste à déterminer les probabilités $P(cl_i/x)$ où x est un exemple, cl_i une classe à laquelle x peut appartenir. La probabilité conditionnelle à postériori de cl_i sachant x est calculée par la formule 1.7.

$$P(cl_i/x) = \frac{P(x/cl_i)P(cl_i)}{P(x)} \tag{1.7}$$

$P(cl_i)$ et $P(x/cl_i)$ sont respectivement la probabilité de l'hypothèse cl_i et celle du vecteur x sachant cette hypothèse ; elles sont estimées sur les données d'apprentissage.

Le classement d'un nouvel exemple est fait en estimant les probabilités $P(cl_i/x)$ pour chaque classe cl_i. La classe ayant la plus grande probabilité est celle de x.

On distingue deux sortes de classifieurs bayésiens : les classifieurs naïfs qui construisent les modèles en supposant qu'il n'y a aucune dépendance entre les attributs et les réseaux de croyance qui considèrent une relation causale entre les attributs.

Les inconvénients des réseaux bayésiens sont la difficulté à calculer toutes les probabilités dans le réseau et la fiabilité du réseau. Ses avantages sont la représentation de connaissances incertaines et le raisonnement à partir d'informations incomplètes.

1.3.5 Les SVM - Support Vector Machine ou Séparateurs à Vaste Marge

Ils sont aussi appelés séparateurs linéaires [FH95, Vap82, Vap95]. Pour un problème consistant à classer les objets en deux catégories cl_1 et cl_2, on suppose qu'en les représentant dans un espace de dimension $N_{cl} + 1$, on peut trouver une surface qui permet de séparer les objets de cl_1 de ceux de cl_2. En effet les SVM cherchent à trouver une formule que peuvent vérifier les exemples d'une classe (cl_1 par exemple). Cette formule est une combinaison linéaire des attributs. Elle permet de séparer l'espace de représentations en deux sous espaces tels que certains objets appartiendront au sous espace de signe + (pour la classe cl_1) et d'autres au sous espace de signe - (pour la classe cl_2).

L'apprentissage consiste donc à rechercher les coefficients w_i de l'équation 1.8

$$si\ x \in cl_1\ alors\ \sum_i w_i x_i \geqslant 0\ et\ x \in cl_2\ alors\ \sum_i w_i x_i \leqslant 0 \tag{1.8}$$

Le classement d'un exemple x' sera de déterminer le signe de la quantité $\Sigma_i w_i x'_i$. Il se verra affecter la classe cl_1 si cette quantité est supérieure à 0 et cl_2 sinon.

Les SVM sont capables de modéliser les phénomènes non linéaires avec une bonne précision mais présentent comme inconvénients le temps de calcul trop long, l'opacité du modèle et le risque de surapprentissage.

1.3.6 Les réseaux de neurones

Un réseau de neurones est un ensemble d'unités de traitement interconnectées capables d'échanger les informations entre elles et avec l'environnement extérieur. Chaque connexion porte un poids. Un réseau de neurones [DN93] dispose des unités qui reçoivent des données des utilisateurs (ce sont les unités d'entrée) et des unités qui fournissent des résultats aux utilisateurs (ce sont les unités de sortie). L'apprentissage des réseaux de neurones consiste à utiliser les données d'apprentissage pour modifier les poids que portent les connexions dans le but de réduire les erreurs entre les sorties attendues et les sorties obtenues du réseau.

Pendant la phase de classement, le réseau obtenu à la phase d'apprentissage est utilisé pour prédire les résultats sur les exemples placés à ses entrées [Sch94]. Ces résultats sont observés sur les unités de sortie.

Les RNA sont capables de bien modéliser les phénomènes non linéaires et d'approximer avec une certaine précision n'importe quelle fonction [BH88, Cyb89, HSW89], mais le temps d'apprentissage est très élevé et le modèle construit n'est généralement pas interprétable.

Les études expérimentales sur les RNA [PYH95, PYH97a] montrent que sur les données utilisées, ils fournissent des résultats satisfaisants. Nous allons présenter dans la prochaine section la technique des RNA, les concepts utilisés et son mode de fonctionnement.

1.4 Les réseaux de neurones artificiels (RNA)

Le vocabulaire utilisé pour désigner les concepts dans le domaine des réseaux de neurones artificiels est inspiré de la biologie. Nous allons commencer par la présentation de ces concepts de biologie avant de les utiliser de manière métaphorique dans le domaine des RNA.

1.4.1 Notions de biologie

Définition 4 *Un **neurone** est la cellule constituante de l'unité fonctionnelle du cerveau. Il est responsable de l'émission et de la propagation du message nerveux [Roj96].*

On compte des milliards de neurones dans le corps humain. En général, les neurones ont le même fonctionnement chez tous les animaux : réception, intégration et transmission de l'information. Il assure la transmission des signaux nerveux entre les organes de sens et le cerveau. Les différents états d'un neurone sont :

1. L'état excité. Dans cet état, il transmet et propage, en fonction des informations qu'il reçoit, des signaux électriques.

2. L'état inhibé. Le neurone est inactif et son activité est ralentie.

Les parties du neurone sont : le noyau, l'axone, les dendrites et les synapses. L'information circule entre les neurones sous forme de signaux appelés influx nerveux. L'influx nerveux se transmet sous forme électrique. Le corps cellulaire reçoit de l'influx en provenance des dendrites et réalise leur intégration. Quand cet influx atteint un certain seuil, le corps cellulaire produit un potentiel d'action qu'il diffuse le long de l'axone grâce à des canaux à ions. A l'extrémité de l'axone, l'influx est transmis aux dendrites des neurones qui lui sont reliés via des synapses. Le poids synaptique définit alors la capacité d'un synapse à influencer le neurone récepteur.

L'apprentissage est alors basé sur le mécanisme d'auto-organisation des poids synaptiques et des liaisons neuronales. L'ensemble des neurones constitue le système nerveux. Le cerveau est l'organe central de coordination et de supervision des activités du corps humain.

Les réseaux de neurones artificiels [DSM$^+$02, FB97, HKP91, MR91, Roj96] ou réseaux connexionnistes [Ben06] (qu'on pourra encore appelés modèles neuronaux) ont été appliqués dans des domaines divers : en classification [Sch94, DSM$^+$02, HKP91], en reconnaissance des formes [Sch94], en reconnaissance des caractères manuscrits [MKS$^+$90, Gos96], en économie [Sha02], en banque [Wit99], en biologie [TR07], dans l'approximation des fonctions [Cyb89]... Ils sont capables de représenter les dépendances non-linéaires de haut niveau entre des variables explicatives. Nous nous intéresserons dans la suite à l'utilisation des RNA pour la résolution des problèmes de classification.

1.4.2 Historique

Le neurone a été formalisé pour la première fois par les chercheurs McCulloh et Pitts en 1943 [MA43]. Par cette formalisation, ces deux chercheurs veulent démontrer que le cerveau est équivalent à une machine de Turing.

En 1949, D. Hebb présente dans son ouvrage intitulé “The Organization of Behavior”[Heb49] une règle d’apprentissage d’un réseau de neurones, de nombreux modèles neuronaux aujourd’hui s’inspirent encore de la règle de D. Hebb.

En 1958, F. Rosenblatt développe le modèle du perceptron [Ros58]. C’est un réseau de neurones inspiré du système visuel. Il possède deux couches de neurones : une couche de perception appelée *rétine* et une couche liée à la prise de décision. C’est le premier système artificiel capable d’apprendre par expérience. Dans la même période, le modèle de l’Adaline (ADAptive LINear Element) [WH60] a été présenté par B. Widrow et M. E. Hoff. Ce modèle sera par la suite le modèle de base des réseaux multicouches.

En 1969, M. Minsky et S. Papert [MP69] publient une critique des propriétés du Perceptron. Cette critique va avoir une grande incidence négative sur la recherche dans ce domaine. Cette recherche va fortement diminuer jusqu’en 1972, où T. Kohonen présente ses travaux sur les mémoires associatives et propose des applications à la reconnaissance de formes.

C’est en 1982 que J. Hopfield présente son étude d’un réseau complètement bouclé, dont il analyse la dynamique. La même année, T. Kohonen propose le modèle SOM (Self Organized Map) [Koh82] pour le traitement des cartes.

En 1985 et 1986, l’algorithme de rétropropagation du gradient est mis au point de manière indépendante par les chercheurs Parker [Par85], Rumelhart [RHW86a] et Lecun [Cun85]. Cette découverte va relancer la recherche sur les réseaux de neurones. Ils seront appliqués dans beaucoup de domaines. Ils connaîtront un grand essor au cours des années 90. Plusieurs autres variantes de l’algorithme du perceptron et de celui de la rétropropagation du gradient seront inventées, ainsi que d’autres modèles. Et depuis deux décennies, les recherches s’intéressent aussi à la dynamique des réseaux de neurones [Ndo05, CTT92], la construction des architectures des réseaux de neurones et à ce que le réseau a appris.

1.4.3 Concepts

Neurone formel

Définition 5 *Un **neurone formel** est une représentation mathématique et informatique du neurone biologique.*

Un neurone formel (ou unité neuronale) reçoit en entrée un vecteur d'attributs (entrées) et met en sortie une valeur (sortie). Ces entrées et cette sortie correspondent respectivement aux dendrites et au cône d'émergence (axone et synapses) du neurone biologique. Les entrées peuvent provenir de l'environnement ou des autres neurones.

On distingue deux types d'unités neuronales [Ben06, Ned98, YPH99] :

1. Unité produit scalaire : ce sont des unités basées sur des automates à seuil. L'état interne est une fonction du produit scalaire entre le vecteur d'entrée et le vecteur des poids de connexion.
2. Unité distance : ce sont des unités basées sur le calcul de distance entre le vecteur d'entrée et le vecteur de poids de connexion. L'état interne est calculé comme une fonction de la distance entre ces deux vecteurs.

Un neurone formel basé sur le système visuel peut être présenté comme le schéma de la figure 1.2.

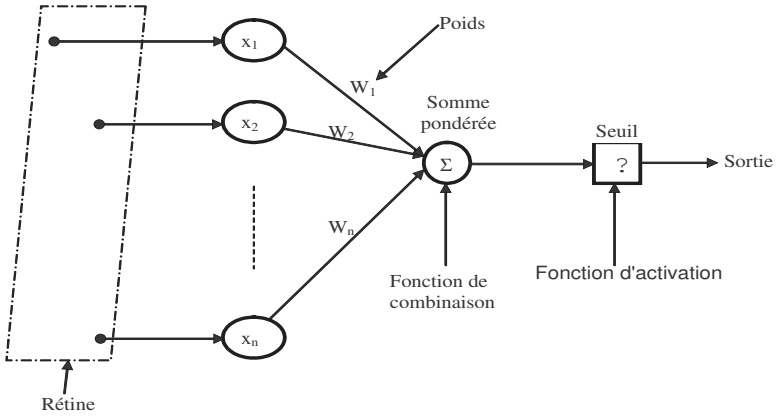

FIG. 1.2 – Neurone formel. La rétine reçoit de l'information et la transmet aux cellules x_i ; les informations des différentes cellules x_i sont ensuite combinées et transférées à la cellule $\sum$.

Couche

Définition 6 *Une **couche** est un ensemble de neurones n'ayant pas de connexions entre eux et ayant presque les mêmes neurones connectées à leurs entrées ou connectés en sortie presque aux mêmes neurones.*

Les activités des neurones d'une même couche se font généralement de manière simultanée. On distingue trois sortes de couches : la couche d'entrée composée des neurones qui reçoivent l'information de l'extérieur, la couche de sortie composée des neurones qui émettent les résultats vers l'extérieur et la couche cachée composée des neurones qui font des traitements intermédiaires. Un RNA peut ne pas avoir de couche cachée ou en avoir plusieurs.

Etat interne

Définition 7 *L'**état interne** e_i du neurone i est un nombre réel associé à ce neurone et qui représente l'information qu'il détient.*

L'état interne permet de connaître le contenu de la mémoire du neurone. Il contient l'information détenue par le neurone à un instant donné.

Poids de connexion

Définition 8 *Le **poids de connexion** est un nombre réel $w_{i,j}$ associé à la connexion entre le neurone i et le neurone j. Il est aussi appelé poids synaptique.*

Le poids de connexion permet de mesurer l'influence du neurone j sur le neurone i. Ce nombre matérialise la liaison entre l'axone du neurone i et le synapse du neurone j. La liaison est excitatrice si son poids de connexion est positif et inhibitrice s'il est négatif.

Fonction de combinaison ou de sommation

Définition 9 *La **fonction de combinaison** d'un neurone est une fonction définie sur les différentes entrées de ce neurone et qui émet comme résultat un réel. Ce réel est appelée **Potentiel membranaire** du neurone.*

La **fonction de combinaison** définit comment les entrées du neurone participent à son activité. Elle définit aussi le type du neurone. Elle permet d'intégrer l'information en provenance de différentes entrées. Parmi les fonctions de combinaison nous pouvons citer [Ned98, YPH99] :

1. Le produit scalaire (pour le neurone produit scalaire). L'activité du neurone est une fonction du produit scalaire du vecteur des entrées x et du vecteur des poids de connexion W. Formellement si A_i est cette activité alors sa valeur est calculée par l'équation 1.9.

$$A_i = \sum_j w_{ij} x_j \tag{1.9}$$

2. La distance (pour le neurone distance). L'activité du neurone est calculée comme étant la distance entre le vecteur des poids de connexion et le vecteur des entrées. Formellement cette activité entre les vecteurs W et x peut être calculée par la formule 1.10 (distance euclidienne).

$$A_i = \sqrt{\sum_j |w_{ij} - x_j|^2} \tag{1.10}$$

Fonction de transfert

Définition 10 *La fonction de transfert d'un neurone est la fonction qui calcule l'état de ce neurone à partir de son potentiel membranaire.*

Encore appelé **fonction d'activation**, elle permet d'introduire la non-linéarité dans le fonctionnement du neurone [DJ99, HKP91]. Une étude de ces fonctions est présentée dans [DJ99]. Parmi les fonctions d'activation, nous présentons les suivantes :

– La fonction sigmoïde (ou logistique) dont la formule est donnée par la formule 1.11 et présentée par la figure 1.3 (a).

$$\varphi(x) = \frac{1}{1 + e^x} \tag{1.11}$$

– La fonction tangente hyperbolique donnée par la formule 1.12 et représentée graphiquement par la figure 1.3 (b).

$$\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.12}$$

– La fonction de Heaviside est une fonction calculée par intervalle. Elle est calculée par la formule 1.13 et représentée graphiquement par la figure 1.3 (c).

$$\varphi(x) = 1 \; si \; x \geq 0 \; et \; 0 \; sinon. \tag{1.13}$$

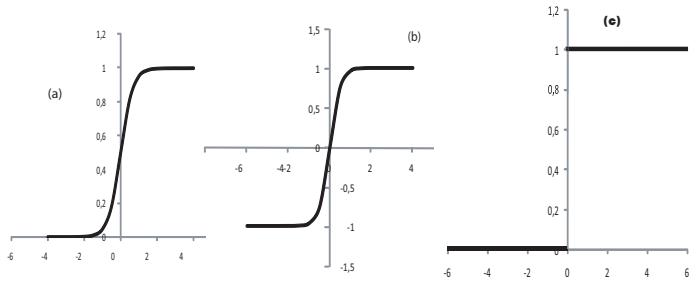

FIG. 1.3 – Représentation graphique de la fonction sigmoïde (a), de la fonction tangente hyperbolique (b) et la fonction de Heaviside (c).

Les fonctions sigmoïde et tangente hyperbolique permettent d'introduire la non linéarité entre les neurones et leurs entrées alors la fonction de Heaviside permet de rendre les sorties binaires [DJ99]. Les fonctions sigmoïde et tangente hyperbolique sont aussi différentiables et continues. Le lecteur peut se referer à [DJ99] pour d'autres fonctions d'activations. Il est également démontré dans [LS93] qu'un RNA avec une fonction d'activation continue autre qu'un polynôme peut approximer n'importe quelle fonction.

Le réseau de neurones

Définition 11 *Mathématiquement, un modèle neuronal ou* ***réseau de neurones artificiels*** *est défini comme un graphe valué orienté, constitué d'un ensemble d'unités (neurones), chaque unité réalisant des calculs élémentaires, structuré (éventuellement) en couches successives capables d'échanger des informations au moyen des connexions qui les relient [Ben06].*

Les caractéristiques d'un RNA sont :
- Une architecture décrivant le nombre de neurones et leurs interconnexions ;
- Pour chaque neurone i :
  1. **Un état interne** e_i défini sur chacun
  2. **La fonction de sommation** Σ qui définit le type du nœud et permet de calculer son potentiel membranaire A_i.
  3. **Le seuil d'activation** θ_i associé à chaque nœud i. Il permet de marquer une limite entre l'activité excitatrice et l'activité inhibitrice.
  4. **La fonction de transfert** $\varphi_i(A_i - \theta_i)$ définie sur tous les neurones.
  5. **Un vecteur de poids de connexion** W où w_{ik} associé à chaque lien entre le nœud i et le nœud k.

– Un pas d'apprentissage η, un nombre strictement positif inférieur à 1 permettant de contrôler la convergence de l'apprentissage.

La circulation de l'information (ou encore la propagation du signal) dans le RNA se fait des neurones d'entrée vers les neurones de sortie (et éventuellement des sorties vers les entrées) de la manière suivante : un neurone i calcule son potentiel membranaire A_i en fonction de ses entrées ; puis applique la fonction de transfert à la différence entre son potentiel A_i et son seuil θ_i. Le résultat obtenu de cette fonction est son état interne qu'il peut transférer aux neurones connectés à sa sortie. Ce processus est répété jusqu'aux neurones de sortie.

1.4.4 Les modèles de réseaux de neurones

Il existe plusieurs modèles de RNA [Ben06, DSM$^+$02, Roj96] : le perceptron, les RNA multicouches feedforward ou Perceptron multicouches, les réseaux récurrents [Ben06, HKP91], les réseaux à fonction radiale [DSM$^+$02, Ben06, HKP91], les cartes auto-organisationnelles [Koh82], les réseaux ART (Adaptive Raisonance Theory) [HKP91], ... Les détails sur ces modèles peuvent être trouvés dans [Ben06, Roj96, HKP91, FS92]. Ces modèles peuvent être regroupés en deux [DSM$^+$02] : les réseaux non bouclés (perceptron, perceptron multicouches, les cartes auto-organisationnelles) et les réseaux bouclés (réseaux récurrents, réseau à fonction radiale, les réseaux ART). Le modèle auquel nous nous intéressons est le modèle des RNA non bouclés.

Le Perceptron

Ce modèle a été créé par Rosenblatt en 1958 [Ros58] et connu comme le premier réseau de neurones. Graphiquement, il est représenté par la figure 1.2. Contrairement au modèle figé de McCulloh et Pitts, le modèle de Rosenblatt est capable d'apprendre. Cette version (figure 1.2) du perceptron est formée de trois éléments principaux :

1. la rétine dont le rôle est la réception des stimulus, des informations en provenance de l'environnement.
2. les unités d'association ou de combinaison connectées en entrée à la rétine et en sortie aux unités de décision. Leur rôle est de faire la somme des signaux d'entrée (rétine) et de la transmettre aux cellules de décison.
3. les cellules de décision qui reçoivent des signaux des cellules d'association et mettent en sortie des décisions du réseau. En général pour un perceptron simple, cette couche n'a qu'une seule unité.

Dans le modèle de la figure 1.2, seules les connexions entre les cellules d'association et les cellules de décision sont pondérées. Pour simplifier, rien que les cellules d'association et de décision ainsi que les connexions entre elles sont représentées sur le schéma du perceptron (figure 1.4). Les sorties du réseau sont binaires (0,1) ou bipolaires (-1,1).

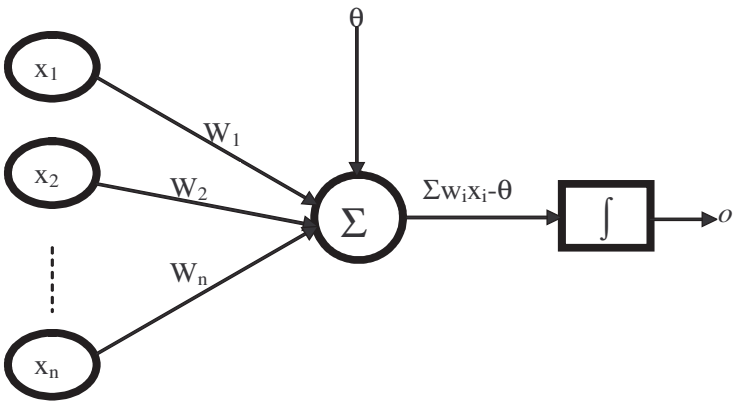

FIG. 1.4 – Schéma simplifié d'un neurone formel. La rétine est supprimée ; on suppose que les informations sont déjà intégrées et peuvent être reçues directement par les cellules x_i.

La sortie o du perceptron est définie par l'équation 1.14 où φ sa fonction d'activation est un sigmoïde.

$$o = \varphi(\sum_i w_i x_i - \theta) \tag{1.14}$$

θ est le seuil d'activation du neurone o et w_i est le poids de connexion entre l'entrée i et le neurone o.

Le perceptron ne réalise qu'une partition des vecteurs d'entrée en deux classes. La frontière entre les éléments de deux classes est définie par le plan d'équation 1.15.

$$\sum_i w_i x_i - \theta = 0 \tag{1.15}$$

Un perceptron est capable de trouver l'équation de la ligne séparatrice pour les fonctions logiques ET et OU représentées respectivement par les figures 1.5 (a) et 1.5 (b). Pour la fonction ET, le point de coordonnées (1,1) a pour sortie Vrai (1) se retrouve seul d'un coté de la ligne alors que les points de coordonnées (0,0), (0,1) et (1,0) sont de l'autre coté. Et pour la fonction OU les points (0,1) ; (1,0) ; (1,1) ont pour sortie Vrai (1) sont d'un coté de la droite alors que le point (0,0) est de l'autre coté. La ligne séparatrice permet de séparer l'ensemble en deux demi-plans de telle sorte que les points ayant pour classe 1 se trouve exclusivement d'un coté et les points ayant pour classe 0 de l'autre coté. Ces problèmes sont dits linéairement séparables.

Définition 12 *Deux ensembles de points S_- et S_+ d'un espace de dimension n sont linéairement séparables s'il existe des réels $w_1, w_2, ..., w_{n+1}$ tel que $\forall (x_1, x_2, ..., x_n) \in S_- \sum_i^n w_i x_i < w_{n+1}$ et $\forall (x_1, x_2, ..., x_n) \in S_+ \sum_i^n w_i x_i \geq w_{n+1}$ [Roj96].*

La séparabilité linéaire d'un problème est la possibilité de trouver un hyperplan défini par l'équation 1.15 (avec $\theta = w_{n+1}$) permettant de diviser l'ensemble d'apprentissage en deux parties S_+ et S_- de telle sorte que les exemples de classe + appartiennent exclusivement à S_+ et ceux de la classe - à S_-. Les fonctions ET et OU sont linéairement séparables et peuvent être traitées par un perceptron.

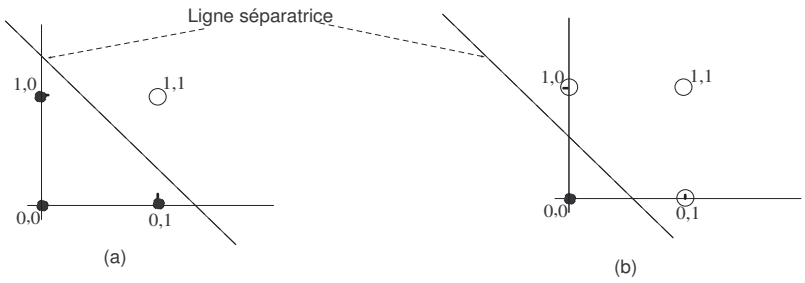

FIG. 1.5 – Représentation graphique des fonctions logiques ET (a) et OU (b).

Théorème 1 *Si un ensemble d'exemples est linéairement séparable, alors l'apprentissage du perceptron converge vers une solution correcte en un nombre fini d'itérations [Ben06, HKP91].*

La démonstration de ce théorème peut être trouvée dans [Ben06, HKP91]. Le perceptron ne converge pas quand le problème n'est pas linéairement séparable. Minsky et Papert [MP69] ont pris le cas de la fonction $OU\ exclusif (XOR)$ dont la représentation graphique est la figure 1.6 pour montrer cette limite du perceptron. En effet, il n'existe pas de droite permettant de séparer les exemples en deux demi-plans contenant chacune exclusivement les éléments d'une classe comme dans le cas des fonctions ET et OU.

La fonction XOR n'est pas linéairement séparable. Par conséquent le perceptron n'est pas capable de résoudre cette classe de problème. Cette limite a poussé les chercheurs à penser à ajouter une nouvelle couche entre l'entrée et la sortie : d'où les RNA multicouches.

Réseau de neurones multicouches feed-forward ou Perceptron Multicouches (PMC)

Il s'agit d'un réseau de neurones qui en plus de la couche d'entrée et de la couche de sortie possède des couches intermédiaires. Ces couches sont dites cachées (internes ou inter-

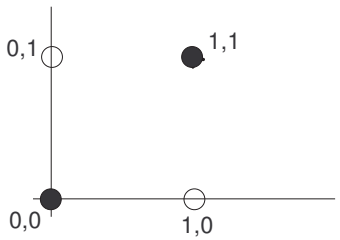

FIG. 1.6 – Représentation graphique de la fonction logique OU exclusif entre deux variables. Les couples (x, y) représentent les valeurs en entrée ; XOR(0,0)=0 ; XOR(0,1)=1 ; XOR(1,0)=1 et XOR(1,1)=0.

médiaires). Les liens de connexion n'existent en général qu'entre les neurones d'une couche et ceux des couches adjacentes. Les couches cachées ont pour rôle de faire les calculs intermédiaires. L'utilisation des couches cachées permet de définir des surfaces de décision non linéaires assurant une classification correcte [Ren06]. Elles augmentent les performances du réseau dans le cas où les exemples à apprendre ne sont pas linéairement séparables (cas de la fonction OU-Exclusif).

L'ajout des couches cachées donne aux RNA la capacité à approximer n'importe quelle fonction. Les RNA multicouches sont considérés comme des approximateurs universels [HSW89, Bar93] :

Théorème 2 *Toute fonction bornée suffisamment régulière peut être approchée uniformément avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant la même fonction de transfert et un neurone en sortie linéaire [HSW89].*

Ce théorème démontré dans [HSW89] montre l'importance de la couche cachée. L'objectif des PMC est de dépasser la limite du perceptron liée à la non séparabilité linéaire des exemples du perceptron. L'architecture des PMC vérifie les propriétés suivantes :

1. Les neurones sont repartis de manière exclusive dans les couches que nous notons dans l'ordre (de l'entrée vers la sortie) $0, 1, ..., H$.
2. La couche 0 est la couche d'entrée et comporte autant de neurones que le nombre de variables d'entrée. Les couches $1, 2, ..., H - 1$ sont les couches cachées.
3. La couche H est composée des neurones de sortie (ou de décision).
4. Les entrées d'une unité de la couche i sont des sorties des unités de la couche $i - 1$.

Le schéma 1.7 est un exemple de RNA multicouches ayant une couche cachée. Les unités a, b, c, d forment la couche d'entrée. La couche de sortie quant à elle est composée des neurones cl_1, cl_2, cl_3. Les unités h_1, h_2, h_3, h_4 constituent la couche cachée. Ce réseau peut être appris pour classer les données du tableau 1.1. La classe *aérien* sera traitée comme cl_1, la classe $terrestre$ comme cl_2 et la classe $aquatique$ comme cl_3. L'attribut a désignera la taille de l'animal, l'attribut b son poids, l'attribut c le nombre de pattes et l'attribut d son mode de déplacement.

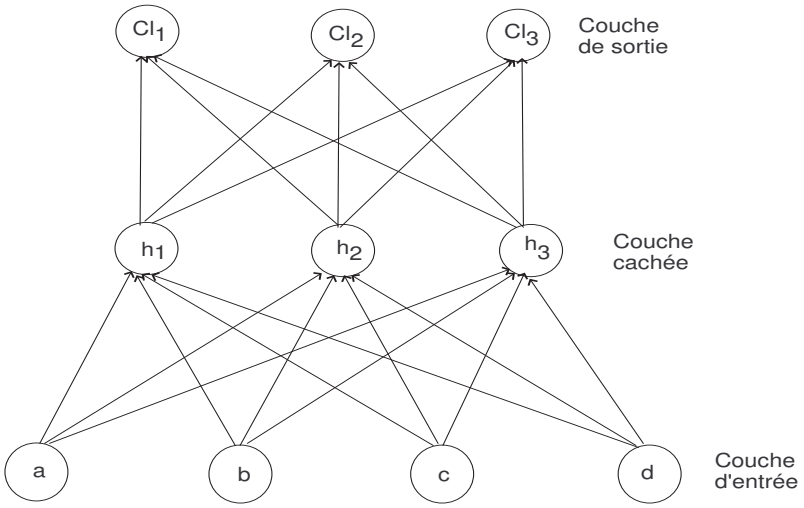

FIG. 1.7 – Exemple de réseau de neurones multicouches.

1.4.5 Apprentissage des RNA

L'apprentissage des RNA peut être supervisé ou non supervisé. L'apprentissage supervisé consiste à modifier les poids de connexion dans le but de minimiser l'erreur totale en sortie du réseau. Le processus guide le RNA à converger vers un point connu. Au cours de l'apprentissage non supervisé, le point de convergence n'est pas connu, le RNA recherche ce point et y converge. Les réseaux que nous construisons sont appris de manière supervisée ; Le mot 'apprentissage' sera utilisé pour parler d'apprentissage supervisé.

Définition 13 *L'**apprentissage d'un réseau de neurones** est le mécanisme par lequel les paramètres libres d'un réseau de neurones sont adaptés à travers un processus de stimulation par l'environnement dans lequel le réseau est intégré [Hay99].*

Dans cette définition de Haykin [Hay99], les paramètres libres font référence aux poids de connexion entre les neurones et aux différents seuils associés à chaque neurone. L'apprentissage consiste alors à modifier les poids de connexion dans le but de réduire les erreurs produites par le neurone. Les erreurs sont calculées en comparant le résultat attendu à celui pris à la sortie du réseau. Lorsque la topologie du réseau est bien définie, les poids de connexion du réseau sont généralement modifiés par l'algorithme de rétropropagation de l'erreur [RHW86a, RHW86b]. Celui-ci produit des bons résultats lorsque l'architecture est appropriée [CM10]. Des outils tels que WEKA [WF05] et SNNS (Stuttgart Neural Network Simulator)[1] intègrent l'algorithme de rétropropagation de l'erreur et apprennent à l'aide de cet algorithme les réseaux de neurones créés par les utilisateurs.

L'algorithme de rétropropagation de l'erreur est plus adaptée aux réseaux multicouches. Pour des réseaux monocouches (ayant une couche d'entrée et une couche de sortie), l'apprentissage est souvent réalisé par l'algorithme d'apprentissage du perceptron (ou de correction d'erreur) ou par une de ses variantes. Nous allons dans la suite présenter quelques algorithmes utilisés pour l'apprentissage des RNA monocouches et des RNA multicouches.

L'algorithme d'apprentissage du perceptron

Un RNA ayant une unité en sortie peut être appris pour partitionner l'ensemble des exemples S en S_+ et S_-. L'algorithme du perceptron détermine les poids W tel que pour un vecteur d'entrée X appartenant à S_+, $WX \geq 0$, et pour un vecteur X appartenant à S_-, $WX < 0$. La sortie o d'une unité de sortie (utilisant la règle du perceptron) est égale à une fonction de la somme des apports des N_{att} unités connectées en entrée à cette unité :

$$o = \varphi(\sum_{j=0}^{N_{att}} w_j x_j - \theta) \tag{1.16}$$

La règle de modification des poids dans l'algorithme du perceptron (équation 1.17).

$$w_j = w_j + \eta(y - o)x_j. \tag{1.17}$$

où η est le pas d'apprentissage. L'algorithme 1 présente un pseudo-code de l'algorithme d'apprentissage du perceptron.

Cet algorithme trouve les poids W appropriés pour définir le demi-plan dans la classification

[1] simulateur des réseaux de neurones disponibles à l'adresse Internet : http ://www-ra.informatik.uni-tuebingen.de/SNNS/

Algorithme 1 Algorithme du perceptron (correction d'erreur)

Entrées: Un ensemble S d'exemples.

Sorties: Un ensemble de poids capables de mieux classer les exemples de S.

1: Initialisation aléatoire des poids w_i pour toute entrée i ;
2: **Répéter**
3: Présenter un exemple en entrée ;
4: Calculer la sortie o du perceptron pour l'entrée x comme indiquée à l'équation 1.16 ;
5: Modifier les poids de connexion (équation 1.17) ;
6: **jusqu'à** ce qu'il y ait convergence ou atteinte du nombre maximal d'itérations

à l'aide d'un perceptron ; il produit de bons résultats lorsque l'ensemble S est linéairement séparable, il ne converge que quand le problème n'est pas linéairement séparable.

Algorithme Pocket with ratchet

C'est une des variantes de l'algorithme du perceptron présentée dans [Gal90]. Il a pour objectif de rechercher les poids de connexion qui classent correctement la plus grande partie de l'ensemble des exemples d'apprentissage même quand cet ensemble n'est pas linéairement séparable. Il exécute pendant plusieurs itérations l'algorithme du perceptron (algorithme 1) sur les exemples d'apprentissage. L'idée de "Pocket with ratchet modification" est de garder "en poche" les poids qui classent correctement la plus grande partie des données avant d'itérer l'exécution de la règle de modification de l'algorithme d'apprentissage du perceptron (équation 1.17). Initialement, dans cet algorithme tous les poids de connexion (ceux de perceptron et ceux en poche) ont une valeur nulle. Quand les poids obtenus de l'exécution de cette règle de modification à l'itération courante classent mieux les exemples que les poids retenus "en poche", les poids gardés en poche sont remplacés par ceux obtenus pendant l'itération. L'algorithme produit comme résultat les poids gardés en poche.

L'algorithme 2 présente le pseudo-code de l'algorithme 'pocket with ratchet modification'. Plusieurs autres variantes de l'algorithme d'apprentissage du perceptron peuvent être trouvées dans la littérature [PYH99]. On peut citer parmi ces variantes, Barycenric Correction algorithm [Pou95] et Thermal Perceptron algorithm [Fre92a]. Une étude comparative de ces algorithmes présentée dans [PYH99] affirme que ces algorithmes ont presque le même comportement et que 'Pocket with rachet modification' est plus simple que les autres.

Algorithme 2 Algorithme pocket with ratchet modification

Entrées: Ensemble d'exemples d'apprentissage et N_{iter} le nombre maximum d'itérations.

Sorties: Les poids de connexion entre le neurone en sortie et les entrées capables de bien classer la grande partie des exemples.

1: Initialiser les poids du perceptron et ceux gardés en poche à zéro ;
2: Choisir aléatoirement un exemple ;
3: Si cet exemple est bien classé par les poids du perceptron alors augmenter le nombre d'exemples bien classés par ce dernier ;
4: Si ce nombre est supérieur à celui des poids gardés alors tester les poids du perceptron sur tous les exemples ;
5: **Si** les poids du perceptron classent mieux que les poids gardés en poche **alors**
6: Remplacer les poids gardés en poche par ceux du perceptron ;
7: Garder aussi son taux de classification ;
8: **finsi**
9: **Si** tous les exemples sont bien classés **alors**
10: Retourner les poids gardés en poche
11: **sinon**
12: Exécuter la règle de modification des poids du perceptron ;
13: **finsi**
14: Si pas de convergence et le nombre d'itérations $Niter$ n'est pas encore atteint alors aller à l'étape 2.

Algorithme de rétropropagation de l'erreur (ou du gradient)

Cet algorithme (que nous allons souvent appeler rétropropagation) consiste à propager d'abord le signal des entrées vers les sorties et ensuite à corriger les poids de connexion par rétropropagation (propager le signal des sorties vers les entrées) des erreurs [Cun85, Par85, RHW86a]. Il est en fait fondé sur la supposition selon laquelle les erreurs obtenues en sortie est la somme des erreurs produites à chaque couche du réseau.

Le but de la rétropropagation est de trouver la combinaison des poids de connexion qui minimise la fonction de l'erreur (risque empirique). Cette fonction est calculée par la formule 1.18.

$$E = \frac{1}{2}\sum_{i=1}^{N} |y_i - o_i|^2 \tag{1.18}$$

Pour minimiser cette fonction (équation 1.18), il faut calculer son gradient ; et pour cela, cette fonction doit être continue et dérivable. Ceci implique que la fonction d'activation φ des neurones ne peut être la fonction de Heaviside (fonction 1.13). Les fonctions les plus utilisées sont la fonction sigmoïde (fonction 1.11) et la tangente hyperbolique (fonction 1.12). Ce gradient $\triangle E$ est calculé par la formule 1.19.

$$\triangle E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l}) \tag{1.19}$$

En effet pour chaque vecteur d'entrée x_i, l'algorithme de rétropropagation exécute les étapes suivantes :

1. Propagation avant. L'exemple x_i est placée en entrée du réseau et le signal est propagé des entrées jusqu'à la sortie. Pour chaque neurone de la couche l (cachée ou de sortie) d'état interne e_j^l, son activation est calculée par $e_j^l = \varphi(A_j^l)$, où A_j^l est le potentiel membranaire calculé par la formule 1.16.
2. Calcul de l'erreur en sortie. Après des déductions faites à partir de la formule 1.19, (qui peuvent être trouvées dans [Roj96]), l'erreur en sortie est donnée par la formule 1.20.

$$E_i^H = \varphi'(A_i^H)(y_i - o_i) \tag{1.20}$$

La grandeur A_i^H est le potentiel membranaire du neurone i de la couche de sortie et calculée la formule 1.16. La fonction φ' est la dérivée de la fonction d'activation. La sortie o_i du neurone est obtenue de l'application de la fonction d'activation φ à ce potentiel membranaire ($o_i = \varphi(A_i^H)$). y_i et o_i sont respectivement la valeur attendue pour l'exemple x_i et la valeur obtenue du réseau après application du signal de x_i en entrée.

3. Rétropropagation (propagation arrière) de l'erreur. L'erreur E_j^{l-1} du neurone j de la couche $l-1$ connecté en sortie aux neurones i de la couche l est donnée par l'équation 1.21.

$$E_j^{l-1} = \varphi'(A_j^{l-1}) \sum_i w_{i,j} E_i^l \tag{1.21}$$

4. Mise à jour des poids. Les poids de connexion qui contribuent à engendrer une erreur importante seront aussi modifiés de façon plus significative que les autres. La grandeur $\triangle w_{i,j}^l$ de mise à jour à ajouter à ce poids est donnée par la formule 1.22.

$$\triangle w_{i,j}^l = \eta E_i^l x_j^{l-1} \tag{1.22}$$

où η une grandeur comprise entre 0 et 1 est le pas d'apprentissage. Les poids sont alors mis à jour par l'équation 1.23.

$$w_{i,j}^l = w_{i,j}^l + \triangle w_{i,j}^l \tag{1.23}$$

L'algorithme 3 est un pseudo-code de l'algorithme de rétropropagation de l'erreur [RHW86a, RHW86b]. Les versions détaillées ainsi que les démonstrations des différentes formules de cet algorithme peuvent être trouvées dans [RHW86a, HKP91, MR91, Roj96].

Algorithme 3 Algorithme de rétropropagation de l'erreur

Entrées: Un ensemble S d'exemples, un RNA avec les poids initiaux.

Sorties: Un RNA avec les poids capables de mieux classer les exemples de l'ensemble S.

1: **Répéter**
2: Présenter un exemple x_i en entrée.
3: Propager le signal des entrées jusqu'à la sortie.
4: Comparer la valeur obtenue $o_i = \varphi(A_i^H)$ en sortie à la valeur attendue y_i.
5: Calculer l'erreur de la couche de sortie en utilisant la formule 1.20.
6: Propager l'erreur jusqu'à la couche d'entrée en utilisant l'équation 1.21.
7: Mettre à jour les poids par la formule 1.23.
8: **jusqu'à** ce qu'il y ait convergence ou atteinte du nombre maximal d'itérations

Cet algorithme est intégré dans beaucoup d'outils de fouille de données comme WEKA ou SNNS. Une des difficultés est qu'il peut ne pas converger ; dans cette situation, il s'arrête lorsque le nombre maximal d'itérations est atteint.

La rétropropagation souffre aussi :

– Des problèmes de minima locaux [HKP91, Roj96]. Pour résoudre cela, il est conseillé de considérer la grandeur à ajouter comme une fonction de la grandeur précédemment ajoutée. Le calcul de $\triangle w$ sera fait par l'équation 1.24.

$$\triangle w_{i,j}^h(t) = \eta E_i^h x_j^{h-1} + \alpha \triangle w_{i,j}^h(t-1) \tag{1.24}$$

où α est un nombre relativement petit appelé moment.

– De lenteur. En effet une petite modification d'un poids affecte relativement le gradient de l'erreur observée sur les autres poids ; en plus la fonction associée à chaque poids est quadratique. Pour cela une variante appelée "QuickProp" a été proposée par Fahlman [Fah88]. Il nous est impossible de juger cette version car les études comparatives sont rares.

Principe du "Winner Takes All" (WTA)

Le Winner Takes All est utilisé dans la situation où un seul neurone parmi les neurones d'une couche donnée doit être actif pour un exemple en entrée [PYH97a]. Il n'est généralement pas considéré comme un algorithme d'apprentissage, mais plutôt comme une méthode d'activation des neurones. Ce neurone (actif) est appelé unité gagnante et les autres neurones les perdants. Tout se passe comme si les neurones de la couche étaient en compétition et qu'il n'y aura qu'un seul vainqueur. Soit x un exemple en entrée, l'unité i gagnante (cette unité représente la classe à laquelle x appartient) est celle qui a le plus grand potentiel membranaire $\sum w_{i,j}x_j$, c'est-à-dire $\sum w_{i,j}x_j > \sum w_{i',j}x_j$ pour tout $i' \neq i$. L'algorithme d'apprentissage modifie les poids de telle sorte que seule l'unité i sera active chaque fois que le vecteur x sera présenté en entrée. Ce principe est pratique en classification. En effet à chaque exemple est associée une seule classe, l'apprentissage doit rendre le réseau capable d'activer une seule sortie (correspondante à sa classe) pour un exemple donné. Le principe de WTA aide alors à trouver le neurone vainqueur et ce dernier représente alors la classe de l'exemple [PYH95, PYH97a].

Complexité des algorithmes d'apprentissage

Nous supposons comme dans [PYH99] où on considère un réseau à une couche permettant en N_{cl} classes N_{obj} objets décrits par N_{att} attributs et que l'apprentissage doit s'arrêter après N_{iter} itérations. Dans cette situation, la couche de sortie a N_{cl} neurones. De manière asymptotique et dans cette condition, les algorithmes Perceptron et 'Pocket with rachet modification' ont la même complexité en temps $O(N_{obj}N_{att}N_{cl}N_{iter})$ et la même en espace $O(N_{obj}N_{att}N_{cl})$.

Nous évaluons la complexité de l'algorithme de retropropagation de l'erreur en fonction du nombre de connexions (N_{con}). Un exemple est par itération au cours de l'apprentissage est en $O(2N_{con})$ correspondant une propagation et une rétropropagation. Pour N_{iter} et N_{obj} exemples cette complexité est en $O(N_{con}N_{iter}N_{obj})$.

Dans des cas d'utilisation, les algorithmes Perceptron et Pocket with rachet modification sont utilisés pour un apprentissage local (pour un neurone) alors que l'algorithme de rétropropagation de l'erreur est utilisé pour un apprentissage global (pour un RNA).

1.5 Conclusion

Ce chapitre a été consacré aux généralités sur la classification et les réseaux de neurones artificiels. Nous y avons présenté le problème de classification supervisée, les étapes de classification et quelques techniques utilisées pour le résoudre. Parmi ces techniques, nous avons brièvement présenté les arbres de décision, les treillis de Galois, les réseaux bayésiens, les SVM, etc.

Nous avons également présenté dans ce chapitre les réseaux de neurones, les notions utilisées dans le domaine des réseaux de neurones et quelques algorithmes d'apprentissage. Plusieurs raisons justifient l'utilisation des réseaux de neurones [PYH95, PYH97a, DFM08] : la possibilité de parallélisation, la tolérance aux bruits, la capacité à pouvoir approximer toute fonction d'un espace de dimension n vers un espace de dimension m (où n et m sont respectivement les tailles du vecteur d'entrée et de sortie) [HSW89], le bon comportement du modèle même dans la situation où on a une faible quantité de données, le bon apprentissage avec peu de connaissances du domaine.

L'utilisation des RNA pour la résolution d'un problème donné suscite néanmoins beaucoup de questions : *Quel est le nombre de couches à utiliser ? Quel est le nombre de neurones pour chaque couche ? Comment connecter ces neurones ?...* Les réponses proposées à ces questions par des recherches seront abordées dans le prochain chapitre. La littérature présente des travaux de recherche proposant des algorithmes pour construire l'architecture d'un RNA. Dans le prochain chapitre, nous allons présenter les algorithmes proposés dans la littérature pour définir cette architecture dans le cadre de la résolution des problèmes de classification.

Chapitre 2

Construction des réseaux de neurones multicouches

Sommaire

2.1 Introduction

Dans le chapitre précédent, nous avons présenté le problème de classification supervisée et les méthodes de résolution. Nous avons aussi présenté la méthode des réseaux de neurones artificiels (RNA). Les domaines d'application des réseaux de neurones sont multiples [DSM+02, Ben06] : biologie moléculaire (analyse des séquences d'ADN [TS94]), traitement d'images, reconnaissance des caractères, économie, génie logiciel, etc. Mais la question suivante est toujours posée pour l'utilisation des RNA [BH88] : "*Quelle est la taille du RNA pour une bonne généralisation ?*". Nous allons aborder dans ce chapitre la construction des réseaux de neurones pour la résolution des problèmes de classification.

L'utilisateur des RNA doit trouver des réponses aux questions suivantes : *quel est le nombre de couches à utiliser ? quel est le nombre de neurones pour chaque couche ? comment connecter ces neurones ?...* Les réponses à toutes ces questions aident à définir l'architecture (ou topologie) du RNA à utiliser. Le problème d'architecture des RNA est un problème ouvert [HK01, CM10] ; en effet, il n'existe pas une démarche méthodique permettant de trouver l'architecture optimale du RNA pour la résolution d'un problème donné.

Les résultats théoriques sur la recherche d'architecture de RNA montrent que trois couches suffisent pour la résolution des problèmes ayant des données continues ou discrètes [Bar93, Cyb89, HKP91, HSW89, LS93] ; mais ces travaux ne proposent pas une méthode pour trouver cette architecture. Dans la pratique, pour les problèmes de classification supervisée, les utilisateurs définissent l'architecture du RNA feed-forward à trois couches (entrée, sortie, cachée) de manière ad'hoc : la couche d'entrée avec autant de neurones que d'attributs décrivant les exemples, la couche de sortie avec autant de neurones que de classes d'exemples et la couche cachée que le nombre de neurones est relativement lié à la connaissance de l'utilisateur. Certains outils définissent ce nombre de manière ad'hoc comme étant la moyenne entre le nombre de neurones en entrée et le nombre de neurones en sortie ; c'est le cas du modèle par défaut construit par la plateforme de fouille de données WEKA [WF05]. Dans le but d'avoir une architecture optimale, d'autres travaux [Rak03, YB00] proposent d'optimiser l'architecture du RNA par une sélection de variables réduisant ainsi le nombre de neurones dans la couche d'entrée.

Les travaux de recherche pour la résolution du problème d'architecture des RNA peuvent être classés en trois grandes catégories :

1. Méthodes statiques : définir d'abord l'architecture du RNA et ensuite l'apprendre à partir des exemples. Parmi ces approches, on peut citer KBANN [TS94]. Elle consiste à utiliser les connaissances du domaine du problème pour définir une architecture. Le

RNA construit est ensuite appris à l'aide de l'algorithme de rétropropagation de l'erreur [RHW86a].

2. Méthodes dynamiques : définir l'architecture du RNA et l'apprendre simultanément. Cette construction faite à partir des exemples est itérée jusqu'à ce que le réseau devienne capable de bien classer les exemples. Elle initialise le réseau avec une couche d'entrée et une couche de sortie, et ajoute à chaque étape un neurone (resp. une couche) dans la couche cachée (resp. réseau) pour corriger les erreurs produites par le réseau à l'étape précédente. Cette approche apporte une solution en l'absence de connaissances. Parmi les méthodes dynamiques, nous pouvons citer MTiling [YPH96], MTower [PYH00], MUpstart [Fre92b, PYH00], Distal [YPH99],...
3. Méthodes évolutives [CO02, SM02]. Il s'agit des algorithmes inspirés de la technique de l'évolution des espèces pour rechercher la structure du réseau. Une population initiale composée de plusieurs RNA est générée. Ces RNA vont évoluer vers des nouvelles architectures en subissant des opérations génétiques comme le croisement et la mutation.

Des études sur les deux premières catégories de ces méthodes de recherche d'architecture de réseaux de neurones sont présentées dans [PYH00, PYH97a, TMNT07b]. Dans ce chapitre, nous allons présenter les différentes méthodes, puis proposer une étude théorique et expérimentale de ces méthodes.

La suite de ce chapitre sera organisée de la manière suivante : nous commencerons par la présentation des critères sur lesquels nous évaluerons les différents algorithmes de recherche d'architecture de RNA. La troisième section montrera comment à partir d'un ensemble de règles obtenir l'architecture d'un réseau de neurones : c'est la méthode KBANN. Les méthodes dynamiques des réseaux de neurones multicouches feront l'objet de la quatrième section. Dans la cinquième section, nous ferons une évaluation des algorithmes sur les critères définis et nous terminerons par les résultats expérimentaux de ces algorithmes sur des données de la base UCI [NHBM98].

2.2 Critères d'évaluation des algorithmes de construction des RNA

La recherche d'architecture d'un réseau de neurones est un processus algorithmique qui conduit à la définition du nombre de couches, du nombre de neurones par couche et du mode d'interconnexion des différents neurones. Les algorithmes de construction d'architectures neuronales diffèrent les uns des autres en fonction de l'architecture du système produit, du type de

données traitées, de l'algorithme d'apprentissage utilisé et des complexités en temps et en espace mémoire. L'apprentissage d'un réseau de neurones par l'algorithme par rétropropagation nécessite une architecture précise et appropriée du réseau [CM10].

Les critères d'évaluation des algorithmes de construction d'un réseau de neurones sont donc :

- Type de données.
  L'utilisation des réseaux de neurones peut être assimilée à des multiplications matricielles. La matrice des poids $w_{i,j}$ est multipliée par le vecteur x_j. Cela implique que les réseaux de neurones peuvent traiter les données numériques sans transformation. Cependant certains problèmes présentent des données symboliques, booléennes, etc ; le traitement ces données nécessite un prétraitement dans le but de normalisation, de discrétisation, de projection ou de binarisation de ces données.
- Algorithmes d'apprentissage.
  Dans les méthodes dynamiques, l'unité ajoutée à l'étape i est apprise pour corriger partiellement ou totalement les erreurs produites par le réseau construit à l'étape $i-1$. Divers algorithmes ont été développés pour cette opération. Ces algorithmes utilisés sont la correction d'erreur (perceptron) ou ses variantes ("Pocket with ratchet modification" [Gal90], "Barycentric Correction" [Pou95], "Thermal Perceptron" [Fre92a]). Ces algorithmes ne peuvent être utilisés que dans le cadre d'un apprentissage local (limité à un neurone) ; ils sont adaptés aux méthodes dynamiques alors que l'algorithme de rétropropation de l'erreur [RHW86a] est plus adapté aux RNA construits avec des méthodes statiques.
- Structure du réseau final et interprétation.
  Pour gagner en temps et en espace mémoire, l'idéal serait pour un problème donné de disposer d'un réseau optimal, ayant un nombre minimum d'unités et de couches et ayant un taux de généralisation acceptable. Il a été démontré qu'un réseau à deux couches cachées ayant suffisamment de neurones est capable d'approximer toute fonction ψ [Cyb89, Bar93]. La structure du réseau a un impact non négligeable sur les temps d'apprentissage du réseau, de classement des exemples et sur sa capacité à généraliser.
  Une structure de RNA où aucune sémentique n'est associée aux éléments rend les résultats du réseau non interprétables, et justifie le refus d'utiliser les RNA dans les domaines où l'interprétation est importante [ADT95].
- Espace mémoire nécessaire.
  La taille mémoire occupée et la gestion de celle-ci lors d'un processus d'apprentissage sont des facteurs importants pour lequel le processus devra trouver une bonne efficacité surtout en présence des données volumineuses. Ce critère permet de juger quelle est la taille maximale de données que la méthode peut traiter sur une machine donnée.

Elménts de l'ensemble des règles	Concepts du Réseau de neurones
Conclusions finales	Neurones de sortie
Conclusions intermédiaires	Unités cachées
Hypothèses	Neurones d'entrée
Lien de dépendance	Lien connexion

TAB. 2.1 – Correspondance entre les éléments de l'ensemble des règles et les éléments du RNA équivalent.

- Temps d'exécution.
  En fonction du nombre d'attributs, du nombre d'exemples et du nombre de classes, le temps d'exécution permet de juger l'efficacité de chaque méthode.
- Nombre de classes en sortie.
  C'est une caractéristique importante d'un modèle de classification. Certains modèles traitent les données à deux classes. D'autres plus élaborés permettent de traiter plusieurs classes.
- Capacité à généraliser
  Elle permet d'évaluer la confiance à accorder au modèle construit.

Nous allons présenter dans les sections suivantes les algorithmes de construction d'architecture de RNA. Nous allons les regrouper suivant les méthodes statiques, les méthodes dynamiques et les autres méthodes.

2.3 Méthode statique : la méthode KBANN

Dans cette section, nous présenterons la méthode KBANN (Knowledge Based Artificial Neural Network). KBANN [TS94] est à notre connaissance l'unique méthode statique qui a été developpée en combinant l'apprentissage empirique et un ensemble de connaissances. Elle construit une architecture à partir des connaissances (représentées par un ensemble de règles d'inférence) du domaine. Les règles sont présentées sous forme de clauses de Horn et sans cycle. Le réseau est dérivé à partir de la hiérarchie des termes. Cette hiérarchie est obtenue à partir des correspondances présentées dans le tableau 2.1.

Les différentes étapes de construction du RNA à partir de l'ensemble des règles sont :

1. Reécriture. Cette étape transforme les règles dans le but de faire ressortir les types de nœuds dans la hiérarchie entre les termes et de faciliter la transformation en RNA. Au

cours de cette phase, les règles ayant une même conclusion sont regroupées. Si un groupe comporte plusieurs règles, alors ces règles sont reécrites de la manière suivante : la conclusion du groupe reste la conclusion des nouvelles règles créées ayant pour antécédents des termes créés, un terme est créé par règle dans le groupe. Ces termes nouvellement créés constituent les conclusions des autres règles qui auront pour antécédents les antécédents du groupe. Le nœud correspondant à la conclusion est un nœud $'OU'$ ou disjonctif et les autres nœuds sont des nœuds $'ET'$ ou conjonctif.

2. Transformation. Au cours de cette phase, l'ensemble des règles est transformé en RNA. Les poids de connexion sont également initialisés. Chaque terme dans l'ensemble des règles devient alors un neurone. Les liens de précédence entre ces termes deviennent les connexions entre les neurones correspondants. Cette étape dispose de manière hiérarchique les règles en commençant par la conclusion finale, suivie des conclusions intermédiaires. Le processus s'arrête quand tous les termes sont placés dans la hiérarchie. Les correspondances du tableau 2.1 sont utilisées pour transformer cette hiérarchie en RNA.
3. La numérotation ou l'étiquettage. Les étiquettes sont attribuées à chaque unité neuronale du RNA formé à l'étape précédente.
4. Ajout des nouveaux neurones cachés. D'après les auteurs [TS94] aucune garantie n'assure que le RNA obtenu à l'étape 2 est fidèle à l'ensemble des règles. Pour cela, ces nouveaux neurones permettent d'étendre le vocabulaire du RNA afin de couvrir la sémantique des règles.
5. Ajout des neurones dans la couche d'entrée. Certains attributs peuvent ne pas figurer dans la liste des termes ; cette étape permet d'introduire les neurones représentant ces attributs.
6. Ajout des liens de connexions. Les connexions sont ajoutées entre les unités de la couche i et celles de la couche $i-1$ si aucun lien n'existait entre ces unités. Les poids de connexion de ces nouveaux liens sont initialisés à 0.
7. Perturbation du réseau. Un facteur de petite valeur est ajouté aux poids de connexion pour perturber le réseau.

Etant donnés p le nombre de connexions positives (variables non complémentées) en entrée du neurone, et w un nombre positif choisi aléatoirement, les poids de connexion des liens entre les neurones sont initialisés de la manière suivante :

* w entre les neurones i et j si le terme correspondant au neurone j était représenté sous forme directe dans la prémisse d'une règle ayant pour conclusion le terme correspondant au neurone i.
* $-w$ entre les neurones i et j si le terme correspondant au neurone j était représenté sous forme complémentée dans la prémisse d'une règle ayant pour conclusion le terme correspondant au neurone i.

* Les seuils d'activation sont $(p-1)w/2$ pour les neurones correspondant à la conclusion des règles conjonctives (ET) et $w/2$ pour les neurones correspondant à des conclusions des règles disjonctives (OU).

Le réseau ainsi obtenu est appris par l'algorithme de rétropropagation du gradient. L'algorithme 4 présente un pseudo-code de la méthode KBANN permettant d'obtenir le RNA à partir des connaissances.

Algorithme 4 Algorithme KBANN

Entrées: Un ensemble de règles décrivant le domaine et un ensemble S d'exemples.

Sorties: Un RNA capable de bien classer les éléments de S.

1: Reécriture ;
2: Transformation ;
3: Numérotation : Remplacement des termes par des unités neuronales ;
4: Ajout des unités cachées ;
5: Ajout des unités en entrée ;
6: Ajout des liens ;
7: Perturbation par ajout d'un nombre proche de zéro ;
8: Apprentissage du réseau par rétropropagation.

Illustration 1 *Un exemple de construction d'un RNA avec la méthode KBANN est décrite à la figure 2.1. Dans cette figure les étapes de cette méthode sont résumées. La première colonne présente un ensemble de règles représentant le domaine ; la deuxième est un ensemble de règles équivalentes à la première mais reécrites et la troisième est l'architecture du RNA obtenu à partir de ces règles.*

Complexité

La complexité (étapes 1 à 7) en temps et en espace mémoire de KBANN est linéaire en nombre de termes de l'ensemble de règles, en supposant que chaque terme représente un attribut alors cette complexité est en $O(N_{att})$ (pour N_{att} attributs décrivant les exemples). A la complexité de construction du réseau s'ajoute celle de l'apprentissage.

Limites

L'utilisation de KBANN nécessite deux ensembles de données en entrée : l'ensemble des règles et l'ensemble des exemples. Il est alors très difficile d'utiliser cette approche lorsque l'on ne dispose pas de règles apriori. D'autres techniques ont été developpées pour construire les réseaux de neurones en l'absence de règles. Nous allons présenter dans la section suivante les méthodes de cette catégorie.

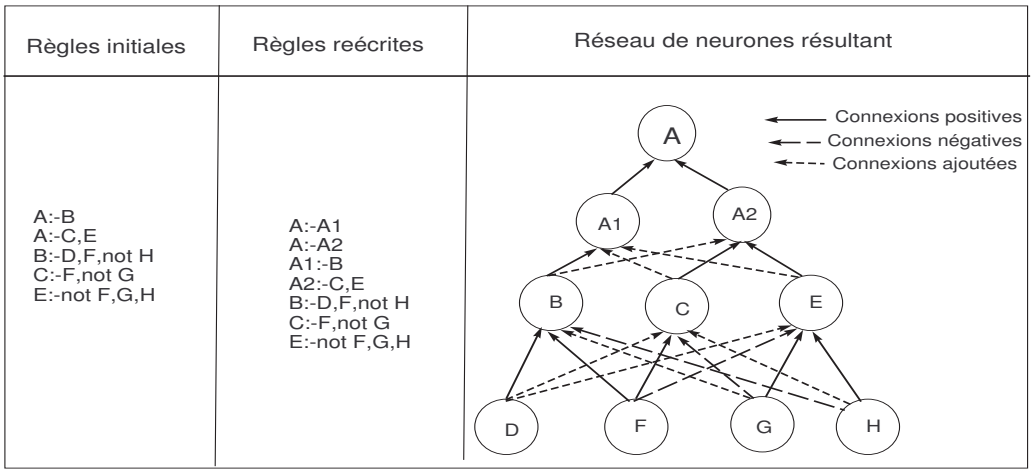

FIG. 2.1 – Exemple de construction de RNA avec la méthode KBANN

2.4 Méthodes dynamiques de construction des RNA multicouches

Ces algorithmes initialisent le réseau de neurones et le font évoluer en y ajoutant des neurones jusqu'à ce que le RNA converge. Nous présentons dans cette section les algorithmes MTiling [YPH96], MTower [Gal90, PYH97a], MUpstart [PYH97b], Perceptron-Cascade [PYH00] et Distal [YPH99]. Une démonstration de la convergence (sous condition que l'ensemble d'apprentissage ne contient pas de données contradictoires) de ces algorithmes est présentée dans [PYH00] et une évaluation empirique dans [NB07]. Une étude théorique et expérimentale de ces algorithmes est également présentée dans [TMNT07b].

Pour l'utilisation des méthodes MTiling, MTower, MUptart et Cascade-Correlation, les auteurs conseillent d'ajouter un attribut à chaque exemple, cet attribut représente la projection du vecteur exemple. La projection est définie comme la somme des carrés de tous les autres attributs ($\sum_i x_i^2$). Le but de ce prétraitement est de réduire l'influence des attributs ayant des valeurs élevées.

Les exemples d'architecture que nous présentons pour illustrer chaque algorithme sont des cas d'architecture pouvant être obtenues des données de la table 1.1. Dans un cadre général, nous supposerons qu'il y a N_{obj} exemples utilisés pour l'apprentissage décrits par N_{att} d'attributs et que ces exemples appartiennent à N_{cl} classes. Nous supposerons aussi que l'utilisateur

choisira H couches maximum et N_{iter} itérations pour l'algorithme d'apprentissage utilisé.

Pour illustrer ces méthodes avec l'exemple des données du tableau 1.1, la classe *aérien* sera traitée comme cl_1, la classe $terrestre$ comme cl_2 et la classe $aquatique$ comme cl_3. L'attribut a désignera la taille des animaux, l'attribut b son poids, l'attribut c le nombre de pattes et l'attribut d son mode de déplacement.

2.4.1 La méthode MTiling

MTiling [YPH96] est une extention de l'algorithme Tiling [MN88] à la résolution des problèmes à plusieurs classes. La méthode Tiling [MN88] construit un réseau de neurones multi-couches dans lequel les unités d'un niveau (couche) supérieur reçoivent des unités du niveau inférieur (immédiat) comme entrées. Mais Tiling est limité aux problèmes à deux classes. MTiling l'étend à N_{cl} en remplaçant dans chaque couche le neurone représentant la classe par N_{cl} neurones chacun représentant une classe. Dans chaque couche cachée, on distingue les neurones maîtres et les neurones auxiliaires. Les neurones maîtres permettent de représenter les classes. Leur nombre dans chaque couche est égal au nombre de classes des exemples. Les neurones auxiliaires sont associés aux neurones maîtres pour corriger les erreurs faites par ces derniers. Pour chaque neurone maître, le nombre de neurones auxiliaires ajoutés est égale au nombre de classes des exemples qu'il a mal classés.

L'algorithme 5 décrit la méthode MTiling. Le réseau est initialisé à deux couches : une couche d'entrée ayant autant de neurones que d'attributs (auxquels une entrée correspondante à la projection a été ajoutée) dans les exemples et une couche de sortie ayant autant de neurones que de classes. Si le réseau existant ne converge pas, la procédure augmente les neurones (neurones auxiliaires) à la couche de sortie courante (ligne 10), elle augmente également une nouvelle couche de N_{cl} neurones au réseau et connecte les entrées de cette couche aux sorties des unités de la couche de sortie ; cette nouvelle couche devient la couche de sortie (ligne 15). Les poids de connexion des neurones ajoutés sont appris individuellement ou en groupe par une variante de l'algorithme d'apprentissage du perceptron.

Les conditions suivantes permettent d'arrêter la modification du réseau existant :

1. le réseau a convergé et classe correctement les exemples d'apprentissage ;
2. le nombre maximum de couches spécifié par l'utilisateur a été atteint
3. la couche (ou le neurone) ajoutée dégrade les performances du réseau existant.

Les stratégies ont été développées pour élaguer le RNA construit par MTiling. Elles consistent à détecter et supprimer les neurones dits 'morts' (qui ne change pas de valeur en sortie quelque

soit l'exemple en entrée), les neurones redondants et les neurones corrélés et de les supprimer du RNA. Deux neurones sont dits redondants s'ils ont toujours la même valeur en sortie quelque soit l'exemple en entrée et corrélés si le coefficient de corrélation calculée entre leurs sorties est élevée en valeur absolue.

La couche de sortie fonctionne suivant le principe de WTA afin d'assurer qu'une seule classe soit active pour un exemple donné.

Algorithme 5 Un pseudo-code de l'algorithme MTiling

Entrées: Un ensemble S d'exemples et H le nombre maximal de couches cachées.

Sorties: Un RNA capable de bien classer les éléments de S

1: Commencer avec un réseau de deux couches complètement connectées donc la première (couche d'entrée) est constituée de $N_{att} + 1$ neurones et la seconde (couche de sortie) de N_{cl} unités, tous les N_{cl} neurones sont les neurones maîtres.
2: Apprendre les poids du réseau ;
3: **Si** les performances du réseau sont acceptables **alors**
4: Arrêt de l'algorithme.
5: **sinon**
6: **Si** les performances de la couche courante ne sont pas satisfaisantes **alors**
7: **Répéter**
8: Identifier le neurone ayant fait le plus d'erreurs ;
9: Déterminer l'ensemble des exemples ayant été mal classés par le neurone identifié ;
10: Ajouter k $(1 \leq k \leq N_{cl})$ neurones auxiliaires ; k est le nombre de classes représentées dans cet ensemble ;
11: Apprendre les poids de connexion des neurones ajoutés ;
12: **jusqu'à** une bonne classification ou à ce que les N_{cl} neurones 'maitres' aient des neurones 'auxilliaires'.
13: **finsi**
14: **finsi**
14: Ajouter aussi une couche de N_{cl} neurones 'maîtres' au réseau et connecter leurs entrées aux neurones de la couche de sortie du RNA courant ;
15: Apprendre les poids de connexion des N_{cl} neurones maîtres connectés aux neurones (comme nouvelle couche de sortie) de la couche précédente et aller à l'étape 2.

Complexité

Dans le pire des cas c'est-à-dire le RNA construit a H couches cachées et qu'après l'apprentissage de chaque neurone ajouté, le nombre de classes des exemples mal classés est égal à N_{cl} : à chaque neurone maître, sont associés N_{cl} neurones auxiliaires ; ceci dans le but de corriger les mauvais classements faits sur les exemples des diffiréntes classes. Chaque couche cachée aura N_{cl} neurones maîtres et N_{cl}^2 neurones auxiliaires, soit $N_{cl} + N_{cl}^2$ neurones au total. Donc le RNA aura $H(N_{cl} + N_{cl}^2)$ neurones cachés. A ces couches cachées devront s'ajouter une couche d'entrée de $(N_{att} + 1)$ neurones et une couche de sortie de N_{cl} neurones. Le réseau final aura alors $(N_{att} + 1) + H(N_{cl}^2 + N_{cl}) + N_{cl}$ neurones.

Pour évaluer le temps de construction du réseau, nous allons nous intéresser aux instructions d'apprentissage (instructions 2 et 15) car le test à l'instruction 3 n'est qu'une propagation en une itération. En supposant que l'apprentissage des neurones ajoutés est l'opération fondamentale de cet algorithme, la complexité en temps $H(N_{cl} + N_{cl}^2)((N_{att} + 1)N_{cl}N_{obj}N_{iter})$. Le temps de construction est en $O(HN_{att}N_{cl}^3N_{obj}N_{iter})$.

Illustration 2 *Les schémas 2.2 et 2.3 présentent une construction d'un exemple de réseau de neurones par la méthode MTiling pour l'apprentissage des données du tableau 1.1. La figure 2.2 (a) présente le RNA initial. La figure 2.2 (b) est le RNA après une itération, MTiling ayant ajouté une couche de N_{cl} neurones maîtres ($N_{cl} = 3$) et N_{cl} neurones auxilliaires. En supposant que le neurone maître (RNA initial (figure 2.2 (a))) cl_1 ait fait des erreurs dans la classification des objets de classe cl_2 et cl_3, et que le neurone maître cl_3 ait aussi fait des erreurs dans la classification des objets de classe cl_1, alors des neurones auxilliares respectifs 12, 13 et 31 sont ajoutés pour corriger ces erreurs et une nouvelle couche (nouvelle couche de sortie) est aussi ajoutée.*

Si après l'apprentissage du nouveau réseau, les neurones cl_1 et cl_2 ne font pas d'erreurs, mais le neurone cl_3 fait des erreurs pour classer des objets de la classe cl_1 alors un neurone auxilliaire (31 de la figure 2.3) est ajouté et une nouvelle couche est également au RNA. Si $H = 2$ alors le RNA final peut être comme le réseau de la figure 2.3.

2.4.2 La méthode MTower

Cette méthode (MTower [PYH97a]) construit le réseau sous forme de 'tour' comme la méthode originale Tower [Gal90]. Tower [Gal90] était limité aux problèmes à deux classes. MTower est son extension aux problèmes à plusieurs classes. Le réseau initial est formé de deux

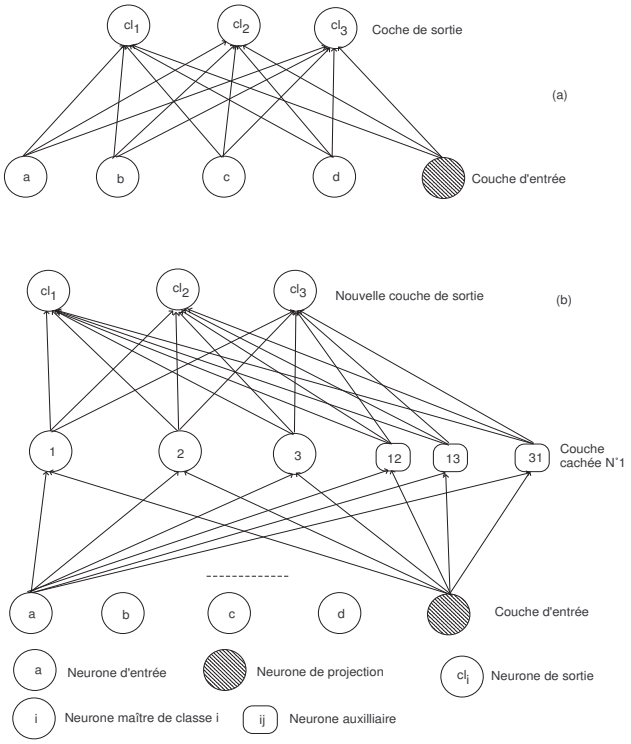

FIG. 2.2 – Initialisation du RNA (a) et ajout d'une couche par la méthode MTiling (b). Les neurones 1, 2 et 3 sont des neurones représentant respectivement les classes 1, 2 et 3 à l'initialisation.

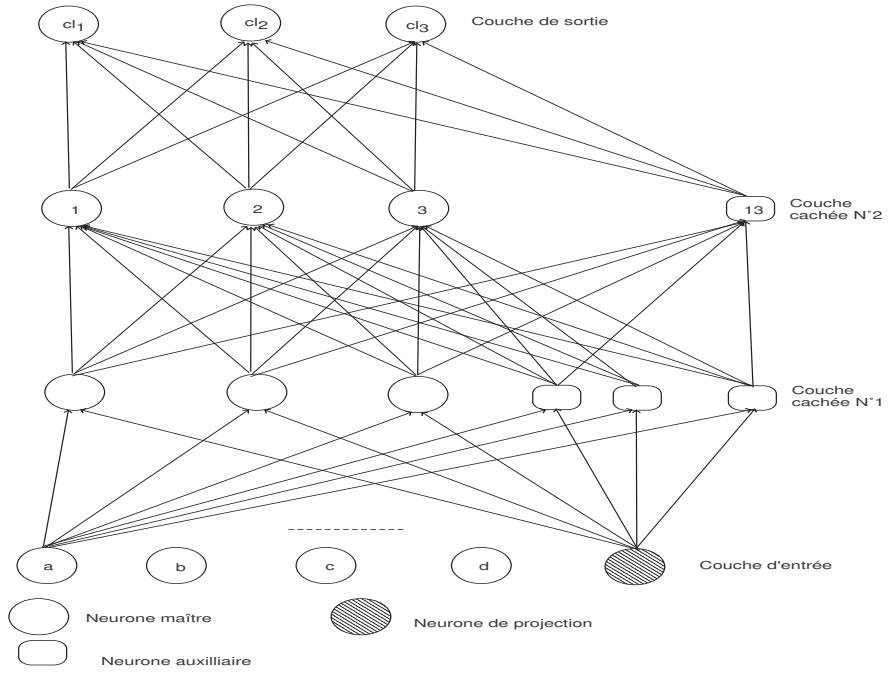

FIG. 2.3 – Architecture d'un réseau de neurones construit par l'algorithme MTiling

couches : la couche d'entrée ayant autant de neurones que d'attributs dans chaque exemple, une couche de sortie dont le nombre de neurones est égal au nombre de classes. Le RNA est construit en ajoutant successivement les couches de N_{cl} unités neuronales au réseau. Ces unités sont apprises par l'une des variantes de l'algorithme d'apprentissage du perceptron. La couche ajoutée est complètement connectée en entrée à la couche de sortie et à la couche d'entrée ; cette couche devient ainsi la nouvelle couche de sortie. La modification du réseau est répétée jusqu'à ce que le réseau converge ou à ce que le nombre maximal H de couches cachées soit atteint. L'architecture du réseau ainsi construit est telle que : les neurones au sommet (sortie) sont connectés à tous les neurones d'entrée ; un neurone de la couche k reçoit l'information de tous les neurones de la couche $k-1$ (couche immédiatement connectés à la couche k). La couche de sortie fonctionne aussi suivant le principe du WTA comme dans le cas de l'algorithme MTiling.

Les conditions d'arrêt de l'algorithme sont les suivantes :
- Le RNA produit des bonnes performances en classant les exemples d'apprentissage ;
- Les neurones ajoutés dans la couche diminue ou n'augmente pas les performances du réseau existant ;
- Le nombre maximum de couches cachées spécifiées par l'utilisateur est atteint.

L'algorithme 6 est le pseudo-code de la méthode de construction MTower des RNA.

Algorithme 6 Algorithme MTower

Entrées: Un ensemble S d'exemples et H le nombre maximal de couches cachées.

Sorties: Un RNA capable de bien classer les éléments de S

1: Initialiser le RNA avec une architecture de deux couches : une couche de sortie de N_{cl} neurones et une couche d'entrée de $N_{att}+1$ neurones.
2: Apprendre le réseau et tester la convergence du réseau ;
3: Si le RNA converge alors arrêt de l'algorithme ;
4: Ajouter une nouvelle couche de N_{cl} neurones au RNA et et connecter ces neurones aux $N_{att}+1$ neurones en entrée ; cette couche devient la nouvelle couche de sortie ;
5: Connecter complètement en entrée la nouvelle couche à l'ancienne couche de sortie du RNA ;
6: Apprendre les poids associés aux neurones ajoutés par une variante du perceptron ;
7: Répéter les étapes 2, 4, 5 et 6 jusqu'à obtention des performances acceptables.

Complexité

En supposant que le réseau ne converge pas et que l'utilisateur a spécifié H couches cachées au maximum, le réseau construit devrait avoir la structure suivante : une couche de sortie de N_{cl}

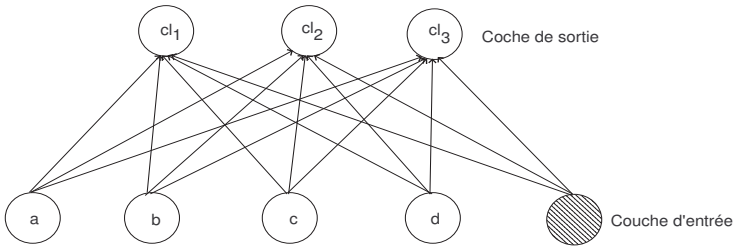

FIG. 2.4 – Architecture d'un réseau de neurones construit initialisée par l'algorithme MTower.

neurones ; une couche d'entrée de $N_{att} + 1$ neurones et H couches cachées de N_{cl} neurones chacune. Le réseau aura au total $(N_{att} + 1) + (H + 1)N_{cl}$ neurones.

Pour évaluer le temps de construction du RNA par la méthode MTower, nous allons nous intéresser aux instructions d'apprentissage (instructions 2 et 6) car les durées d'exécution des autres instructions sont dominées par les temps des instructions d'apprentissage. L'instruction 2 peut se faire en $O(H(N_{att}+1)N_{cl}^2 N_{obj} N_{iter})$. L'étape 6 pourra se dérouler en $(N_{att}+1)N_{cl} \times N_{obj} \times N_{iter}$. En conclusion, la complexité de MTower est en $O(H N_{cl}^2 N_{obj} N_{att} N_{iter})$.

Illustration 3 *La figure 2.4 présente une architecture initialisée par MTower pour la classification des données du tableau 1.1. Supposons que ce RNA (figure 2.4) fasse des erreurs alors une nouvelle couche de N_{cl} neurones (3 pour ce cas) est ajoutée. Le RNA résultant de cette modification est présenté par la figure 2.5. Si le nouveau RNA fait encore des erreurs après l'apprentissage, une nouvelle couche de N_{cl} est également ajoutée ; la figure 2.6 présente le RNA résultant. Dans cette figure (pour le cas de MTower), les connexions notées (u) et (v) ne sont pas ajoutées. Si le nombre maximum de couches cachées est 2 ($H = 2$) alors la construction du RNA s'arrête.*

2.4.3 La méthode MPyramid

MPyramid [PYH00] construit le RNA de la même façon que MTower. La différence entre les deux méthodes réside sur le fait qu'une couche k ajoutée par MPyramid est complètement connectée à toutes les couches i ($1 \leq i < k$) alors que pour MTower cette couche n'est que connectée à la couche immédiatement inférieure et à la couche d'entrée. Sa complexité est semblable à celle de MTower.

Illustration 4 *La figure 2.6 est un exemple de RNA construit par la méthode Mpyramid pour les données du tableau 1.1. Le nombre de couches cachées choisi est deux. Le RNA est initialisé*

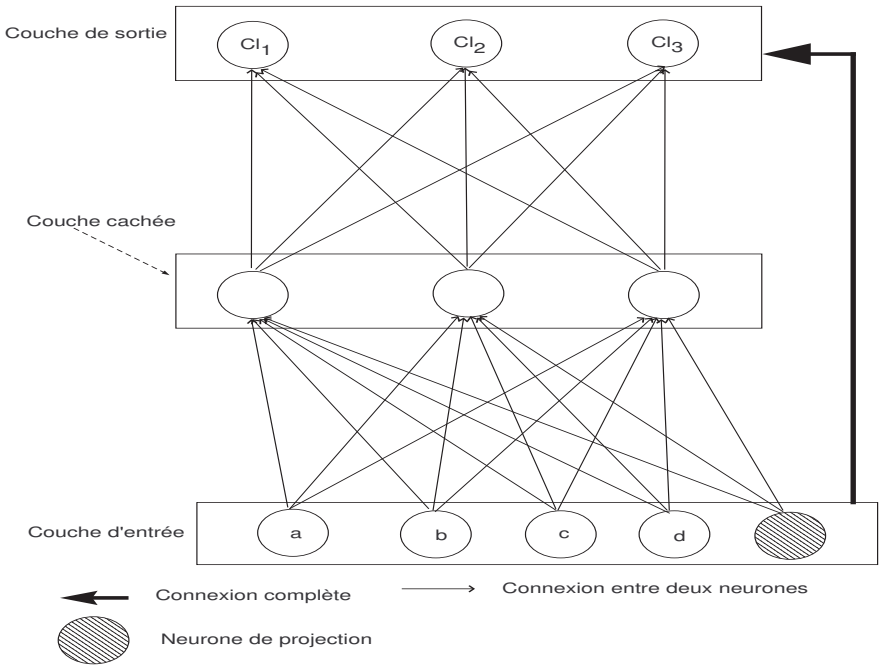

FIG. 2.5 – Architecture d'un réseau de neurones construit par l'algorithme MTower si le nombre de couche maximum est 1.

comme le réseau de la figure 2.4. En cas d'erreur une nouvelle couche est ajoutée au RNA et le RNA se présente comme la figure 2.5. Si ce nouveau RNA fait encore des erreurs alors une autre couche de N_{cl} neurones est encore ajoutée et le RNA sera comme celui de la figure 2.6. Contrairement au RNA construit par MTower, les connexions notées (u) et (v) sont ajoutées. Et si $H = 2$ alors le processus s'arrête.

2.4.4 La méthode MUpstart

MUpstart [PYH97b] est une version de l'algorithme Upstart [Fre92b] pour la classification multi classes. Comme Upstart, cet algorithme est basé sur une correction d'erreur produite par le réseau courant ; l'algorithme MUpstart construit un réseau à partir des exemples du problème un réseau sous forme d'arbre binaire. Ce réseau est initialisé à deux couches : une couche d'entrée de $N_{att} + 1$ unités et une couche de sortie de N_{cl} unités. Pendant la construction du réseau, Upstart distingue deux sortes d'erreurs pour un neurone du réseau : "Wrongly on" et "Wrongly off". En cas d'erreur, un fils gauche G est augmenté au RNA et connecté au nœud si cette erreur est "Wrongly on" ou un fils droit D si cette erreur est "Wrongly off". On parle de "Wrongly on" (resp. "Wrongly off") si $o = 1$ et $y = 0$ (resp. $o = 0$ et $y = 1$) où o est la sortie obtenue du réseau et y la sortie attendue. Le neurone ajouté est appris par une variante du perceptron

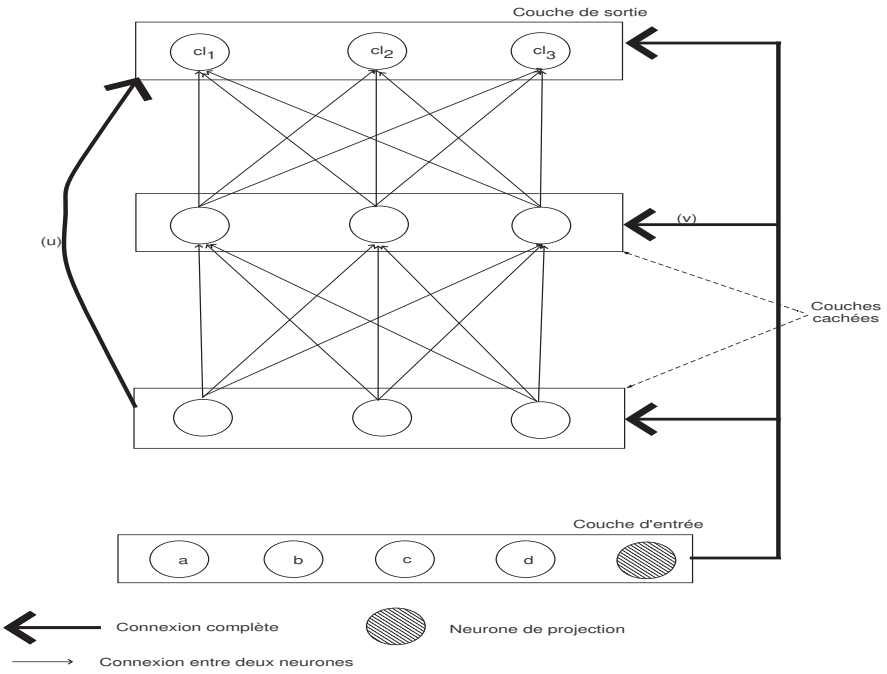

FIG. 2.6 – Architecture d'un réseau de neurones construit par l'algorithme Mpyramid. Le nombre maximum de couches cachées est 3.

et connecté à tous les neurones de la couche supérieure. La couche de sortie fonctionne suivant le principe du WTA. L'apprentissage des unités G et D se fait avec comme sortie attendue les données par le tableau 2.2 où y (resp. o) désigne la sortie attendue (resp. obtenue) ; T_G (resp. T_D) est la sortie attendue pour l'apprentissage de G et D en cas de wrongly on et de wrongly off respectivement).

L'algorithme 8 présente le pseudo-code de la méthode MUpstart de construction de RNA. Il utilise le tableau 2.2 pour orienter l'apprentissage des neurones ajoutés.

o	y	T_G	T_D
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

TAB. 2.2 – Tableau présentant les sorties attendues des unités ajoutées. y et o sont respectivement la sortie obtenue et la sortie attendue pour l'exemple.

Algorithme 7 Algorithme MUpstart

Entrées: Un ensemble S d'exemples et H le nombre maximal de couches cachées.

Sorties: Un RNA capable de bien classer les éléments de S

1: Initialiser le RNA à deux couches : N_{cl} unités en sortie et $(N_{att}+1)$ unités en entrée ;
2: Apprendre les N_{cl} unités en sortie ;
3: **Si** tous les exemples ne sont pas bien classés **alors**
4: **Répéter**
5: Déterminer le neurone de la couche de sortie ayant fait plus d'erreurs ;
6: Ajouter l'unité G (resp. D) dans le cas de Wrongly on (resp. Wrongly off) ;
7: Construire l'ensemble d'apprentissage de cette unité ;
8: Réapprendre les poids de connexion de cette unité neuronale et geler ces poids ;
9: Connecter l'unité à toutes les sorties et réapprendre ces poids ;
10: **jusqu'à** le réseau converge ou que le nombre maximal de neurones dans la couche cachée soit atteint.
11: **finsi**

Complexité

Le RNA construit a une architecture de : $N_{att}+1$ neurones dans la couche d'entrée et $N_{cl}H$ neurones dans la couche cachée. Si chaque neurone (de la couche de sortie) provoque au moins un erreur 'Wrongly on' et une erreur 'Wrongly off', alors deux neurones seront ajoutés dans la couche cachée pour corriger ces erreurs. La première couche cachée aura $2N_{cl}$ neurones. Donc au pire des cas, le nombre de neurones ajoutés est doublé à chaque correction. Cette structure aura $(N_{att}+1)+N_{cl}+2^H N_{cl}$ neurones.

Au cours de l'exécution de l'algorithme 8, l'instruction 2 se déroulera en $N_{cl}(N_{att}+1)N_{obj}\times N_{iter}$. L'apprentissage (instruction 8) du neurone ajouté se fait en $N_{obj}(Natt+1)N_{iter}$. Le réapprentissage du réseau pourra être majoré par $((N_{att}+1)N_{iter}N_{cl}N_{obj}$. Donc MUpstart a une complexité asymptotique en $O(2^H N_{cl}^2 N_{att}N_{obj}N_{iter})$.

Illustration 5 *Les figures 2.7 et 2.8 illustrent la construction d'un RNA par la méthode MUpstart. Elles montrent l'insertion d'un neurone fils (droit). Ce RNA est initialisé comme dans le cadre de MTower (figure 2.4). Supposons que le neurone de sortie cl_1 fait une erreur de type 'wrongly on' et une erreur de type 'wrongly off' alors les deux neurones G_1 et D_1 sont ajoutés et connectés à cl_1. Supposons aussi que le neurone de sortie cl_2 ait fait un 'wrongly on' alors le neurone D_2 est ajouté au RNA et connecté aussi à cl_2 ; la figure 2.7 illustre ces modifications.*

Supposons qu'après le test du RNA de la figure 2.7, les neurones G_1 et D_1 font respecti-

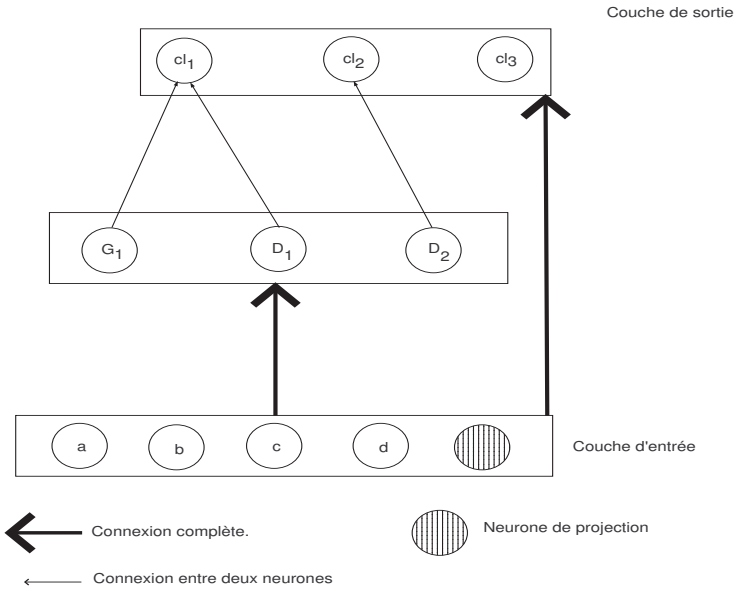

FIG. 2.7 – Architecture d'un réseau de neurones construit par l'algorithme MUptart. les neurones cl_1 a fait deux erreurs, le neurone cl_2 une erreur et le neurone cl_3 aucune erreur.

vement un 'wrongly on' et un 'wrongly off', les neurones G_{12} et D_{21} sont également ajoutés au RNA pour corriger ces erreurs. Si le nombre maximum de couches est 2 alors l'algorithme s'arrête et le RNA construit est celui de la figure 2.8

2.4.5 La méthode Cascade-Correlation

La méthode Perceptron-Cascade [PYH00] est une extension de la méthode Cascade-Correlation [FL90] aux problèmes multi-classes. Perceptron-Cascade construit un réseau de neurones par ajout successif des neurones dans la couche cachée. Le réseau initial comporte deux couches : la couche d'entrée ayant autant de neurones que d'attributs dans les exemples et la couche de sortie ayant autant de neurones que de classes. Ces deux couches sont complètement connectées à l'initialisation et les poids de connexion peuvent être modifiés. Les neurones de la couche de sortie calculent une combinaison linéaire de leurs entrées pendant le fonctionnement du réseau.

Les neurones ajoutés sont complètement connectés en entrée à tous les neurones de la couche d'entrée et aux neurones des autres couches cachées ; en sortie à tous les neurones de la couche de sortie. Le processus se passe comme si à chaque itération, une nouvelle couche d'un neurone est ajoutée au réseau. Leurs poids de connexion sont gelés (ne pourront pas être modifiés) après l'insertion. Le réseau est appris à nouveau (mais seuls les poids de connexion

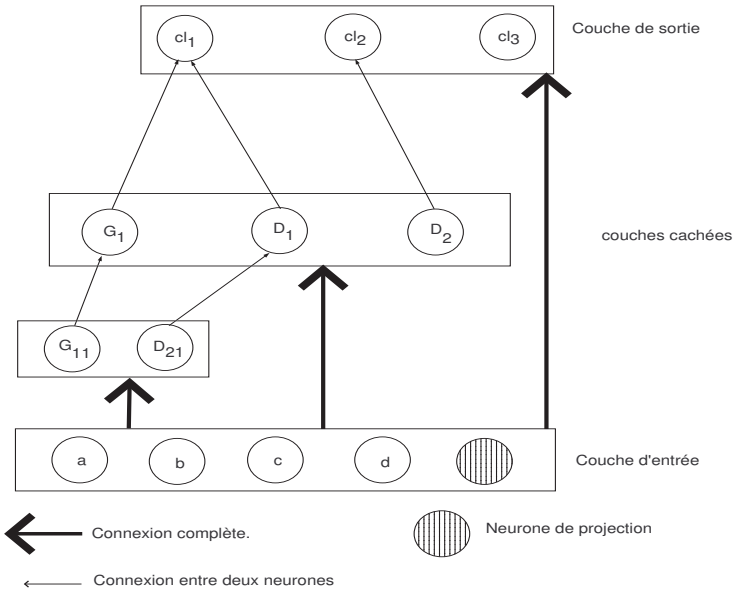

FIG. 2.8 – Architecture d'un réseau de neurones construit par l'algorithme MUptart.

de la couche de sortie sont modifiés). Si ses performances après cet apprentissage sont satisfaisantes alors l'algorithme s'arrête. Sinon un nouveau neurone est ajouté. L'algorithme est répété jusqu'à satisfaction des performances (ou atteinte d'un nombre maximum de couches cachées spécifiées par l'utilisateur).

Complexité

Cascade est similaire à MUpstart ; la différence réside sur le fait le neurone ajouté dans Cascade est connecté à tous les neurones des couches inférieures alors dans MUpstart ce neurone est connecté à la couche d'entrée. Donc Cascade a la même complexité que MUpstart : $O(2^H N_{cl}^2 N_{att} N_{obj} N_{iter})$.

Illustration 6 *La figure 2.9 est un exemple de RNA construit avec la méthode Cascade. Il est initialisé comme la figure 2.4. Les neurones sont ajoutés comme dans le cas de MUpstart mais ces neurones ajoutés sont complètement connectés aux neurones des couches supérieures.*

2.4.6 La méthode Distal

Cette méthode calcule d'abord les distances entre exemples, pour chaque exemple, Distal trie ces distances par ordre croissant et les stocke dans une matrice D. La méthode Dis-

Algorithme 8 Algorithme Cascade

Entrées: Un ensemble S d'exemples et H le nombre maximal de couches cachées.

Sorties: Un RNA capable de bien classer les éléments de S

1: Initialiser le RNA à deux couches : N_{cl} unités en sortie et $N_{att} + 1$ unités en entrée ;
2: Apprendre les N_{cl} unités en sortie ;
3: **Si** tous les exemples ne sont pas bien classés **alors**
4: Ajouter une couche au RNA ;
5: Choisir de manière aléatoire un neurone de sortie qui a provoqué au moins une erreur ;
6: Ajouter un neurone dans la couche immédiatement inférieure à la couche de sortie ;
7: Construire l'ensemble d'apprentissage de cette unité et l'apprendre ses poids de connexion ;
8: Geler les poids de connexion du nouveau neurone ;
9: Connecter l'unité à toutes les sorties et réapprendre ces poids.
10: **finsi**

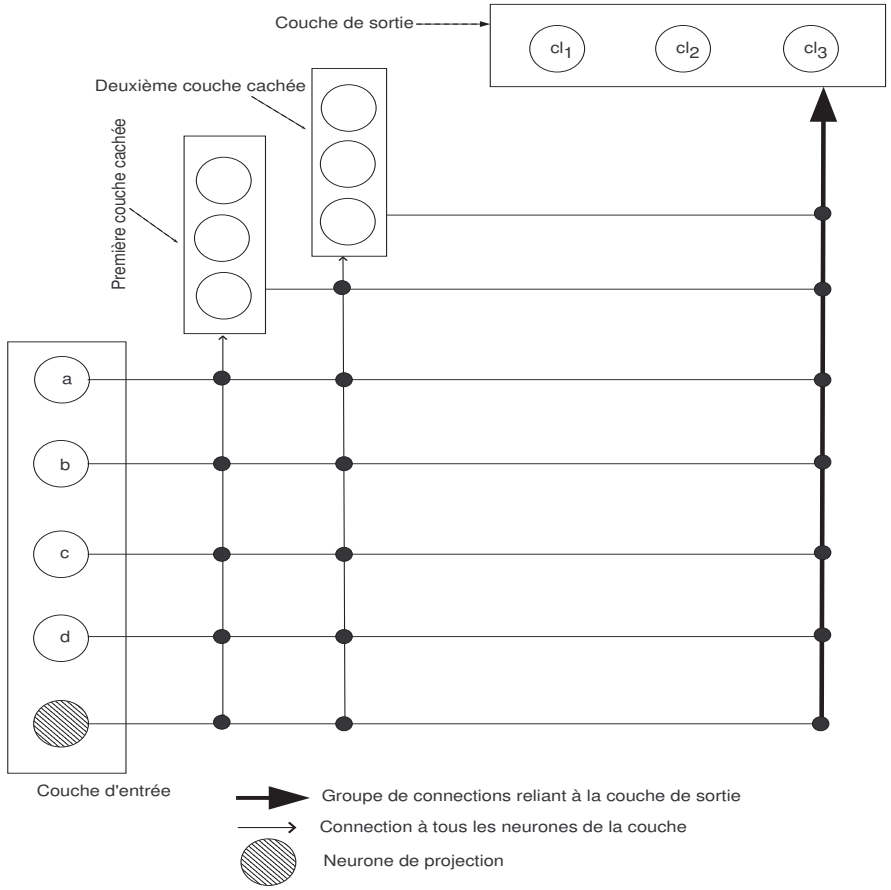

FIG. 2.9 – Architecture d'un réseau de neurones construit par l'algorithme Perceptron Cascade.

tal [YPH99] crée alors un réseau à trois couches dont une couche d'entrée (resp. de sortie) ayant N_{att} (resp. N_{cl}) unités à partir des données d'apprentissage sans aucune connaissance du domaine. Contrairement aux algorithmes précédents, les neurones dans Distal sont des unités distances.

La création de la couche cachée se fait de la manière suivante : Un neurone est ajouté pour reconnaître k exemples tel que k soit le nombre d'exemples de la classe ayant plus d'éléments consécutifs dans une ligne de la matrice D appartenant à la même classe (étape 4). Les neurones de la couche cachée dans Distal sont à seuil sphérique (neurone actif si $\theta_{low} \leq distance(W, X) \leq \theta_{high}$ et inactif sinon ; $distance(W, X)$ est la distance entre les poids de connexion W du neurone et le vecteur d'entrée X). Les poids de connexion W entre l'unité ajoutée et les unités d'entrée sont les valeurs des attributs de l'exemple à partir duquel le plus grand groupe a été trouvé. Les poids de connexion entre le neurone ajouté et un neurone i de la couche de sortie sont initialisés à 1 si les exemples du groupe appartiennent à la classe représentée par le neurone i et à zéro sinon. A l'étape suivante, tous les poids de connexion entre la couche cachée et celle de sortie sont multipliés par deux. Les unités de la couche de sortie fonctionnent suivant le principe de WTA (Winner Take All).

L'algorithme 9 est le pseudo-code de l'algorithme Distal. Il suppose que les distances entre les objets sont calculées et stockées dans la matrice D. Les autres objets utilisés dans l'algorithme sont $\theta_{low}, \theta_{high}$ qui signifient respectivement les seuils inférieur et supérieur du neurone ajouté.

Complexité

Si nous supposons qu'aucun regroupement n'est possible, c'est à dire chaque groupe contient un seul exemple. Alors le réseau construit aura la structure suivante : une couche d'entrée de N_{att} neurones, une couche de sortie de N_{cl} neurones et une couche cachée de N_{obj} neurones.

D'après les auteurs, ce calcul de distances entre N_{obj} exemples peut se faire en $O(N_{obj}^2)$, une opération supplémentaire faite en prétraitement. Les opérations de tri pour un exemple en $O(N_{obj} \ln N_{obj})$ et de recherche (ligne 4) de la plus grande plage en $O(N_{obj})$; l'algorithme a une complexité asymptotique en $O(N_{obj}^2)$.

Il existe dans [YPH99] des procédés permettant de calculer les distances sur les données symboliques, ce qui donne à Distal la faculté de traiter des données symboliques. Mais, il se pose le problème du choix de la distance. Les auteurs de Distal suggèrent d'utiliser les connaissances du domaine pour faire ce choix.

Illustration 7 *La figure 2.10 présente un exemple de réseau de neurones obtenu par la méthode*

Algorithme 9 Algorithme Distal

Entrées: Un ensemble S d'exemples.

Sorties: Un RNA capable de bien classer les éléments de S.

1: Initialiser le RNA avec une couche d'entrée et une couche de sortie ;
2: **Tant que** l'ensemble S n'est pas vide **faire**
3: Doubler les poids entre la couche cachée et la sortie ;
4: Identifier la ligne k de D qui permet d'exclure de S le plus grand nombre de exemples appartenant à la même classe (où D est la matrice des distances) de la manière suivante :
5: **Pour** toute ligne r de la matrice D **faire**
6: Identifier le plus groupe C_r d'exemples consécutifs appartenant à la même classe ;
7: Sélectionner $k = argmaxC_r$ et affecter à S_k le groupe d'exemples correspondants ;
8: Soit S_k est l'ensemble des exemples appartenant à cette plage $d^k_{low} = D[k, i_k], d^k_{high} = D[k, j_k]$.
9: **fin pour**
10: Ajouter un neurone à la couche cachée, le connecter complètement à la couche d'entrée et affecter à ses poids de connexion les composants de l'exemple X_k et ses seuils d'activation $\theta_{low} = d^k_{low},\ \theta_{high} = d^k_{high}$;
11: Supprimer S_k de S ;
12: Connecter le nouveau neurone caché à toutes les sorties $W_{i,h} = 1$ (si les exemples supprimés appartiennent à la classe i) et $W_{i',h} = 0$ où $i \neq i'$.
13: **fin tantque**

Distal. L'ensemble d'apprentissage du schéma (a) est partionné par l'algorithme pour former la couche cachée du schéma (b).

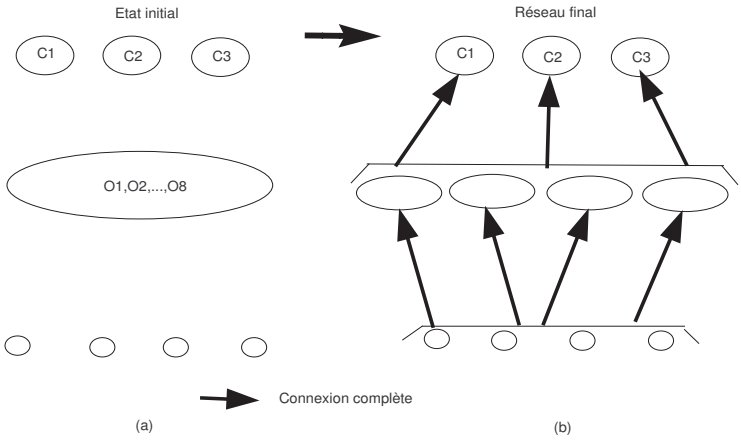

FIG. 2.10 – Architecture d'un réseau de neurones construit par l'algorithme Distal. Initialement, la couche cachée est vide ; Distal partitionne les exemples et représente chaque partition par un neurone (partie (b)).

2.4.7 Limites des approches dynamiques

La principale limite de ces approches est l'opacité des RNA construits. L'utilisateur n'a aucun moyen de justifier l'architecture du réseau construit et de contrôler ses résultats. Aucune sémantique n'est associée aux neurones ajoutés, ce qui rend difficile l'interprétabilité des RNA construits par ces approches. Un autre problème concerne la paramètrage : choisir le nombre maximum de couches cachées, l'algorithme d'apprentissage, le normbre d'itérations au cours de l'apprentissage, la distance entre les exemples et le coût mémoire. Le choix de ces paramètres n'est pas facile et il n'existe aucun moyen permettant de guider ce choix.

2.5 Autres approches

Il existe d'autres méthodes de recherche d'architecture de réseau de neurones. Parmi ces méthodes, peuvent être classées les méthodes évolutionnaires [CO02, VJJ97], les méthodes ad'hoc, ...

2.5.1 Les méthodes évolutionnaires

Elles sont inspirées de la théorie de l'évolution naturelle des espèces [CO02, SM02, VJJ97]. Pour la construction des réseaux de neurones, l'espace de recherche est constitué de toutes les topologies possibles et de leur fonction de fitness. Les opérations génétiques de mutation et de croisement permettent de faire évoluer ces réseaux vers une bonne architecture. Les réseaux engendrés sont appris et leurs fitness évalués. L'évaluation du fitness doit prendre en considération l'erreur totale, la vitesse de convergence et la complexité du réseau. On peut décider d'encoder l'ensemble des caractéristiques du réseau par une matrice d'incidence permettant de symboliser une connexion entre deux neurones. L'opération de mutation va alors consister à modifier un composant de cette matrice. Le croisement consistera à permuter les lignes de la matrice d'incidence de deux parents pour engendrer deux descendants.

Les mutations peuvent modifier les poids de connexion ou la structure du réseau. La mutation des poids consiste à l'ajout d'une grandeur $\pm\varepsilon$. La mutation de structure peut s'opérer de deux manières :

1. Ajout de connexion par mutation. Une nouvelle connexion est ajoutée entre deux neurones (qui n'étaient pas connectés) et son poids initialisé à 1.
2. Ajout d'un neurone par mutation. Une connexion existante entre deux neurones i et j est supprimée et un nouveau neurone est placé à sa place. Ce neurone est alors connecté en entrée à j et en sortie à i. Les poids de connexion ont la mème valeur que le poids de la connexion supprimée.

Il est très difficile d'estimer la complexité des approches évolutives parce qu'elle est étroitement liée aux opérations génétiques (croisement et mutation). Si la population initiale est de taille importante, alors l'exécution d'une méthode évolutive nécessitera aussi beaucoup de temps et d'espace mémoire. Un autre problème est la difficulté à l'utilisateur d'interpréter les résultats obtenus du RNA construit.

2.5.2 Les méthodes ad'hoc

Elles consistent à construire un réseau à trois couches : une couche d'entrée ayant autant de neurones que d'attributs (N_{att} attributs) dans les exemples, une couche de sortie ayant autant de neurones que de classes (N_{cl} classes) dans les exemples et une couche cachée dont le nombre de neurones est une fonction des nombres de neurones en entrée et en sortie. Les fonctions généralement utilisées sont :

1. La règle empirique de Baum-Haussler [Ren06, BH88]. Le nombre optimal de neurones

cachée N_h est doit vérifier la règle suivante :

$$N_h \leq \frac{N_{obj} \times E}{N_{att} \times N_{cl}} \tag{2.1}$$

où N_h est le nombre de neurones de la couche cachée, N_{obj} le nombre d'exemples d'apprentissage et E l'erreur relativement acceptable.

2. La méthode Ad'hoc simple. Le nombre de neurones de la couche cachée est calculé comme étant la moyenne entre le nombre de neurones de la couche d'entrée et le nombre de neurones de la couche de sortie.

$$N_h = \frac{N_{att} + Ncl}{2} \tag{2.2}$$

Le RNA construit par les méthodes ad'hoc ont $N_{att} + N_h + N_{cl}$ neurones. Le temps de contruction du RNA est en $O(max(N_{att}, N_{cl}))$. Les avantages de la méthode ad'hoc sont la facilité de paramètrage et la complexité linéaire de construction du RNA. Mais le RNA construit est opaque à l'utilisateur.

Les méthodes de construction des RNA diffèrent suivant plusieurs critères. Nous allons dans la section suivante discuter sur ces critères.

2.6 Discussion - Analyse des algorithmes

Nous présentons dans cette section, une évaluation des algorithmes de construction de RNA suivant le type de données, la structure du réseau, etc.

Types de données

Les bases de données stockent plusieurs types d'informations (numériques, symboliques, multimédia ...). Il est important pour ces algorithmes de pouvoir traiter différents formats de données. Les algorithmes cités précédemment ne traitent que les données numériques à l'exception de Distal et de KBANN. KBANN définit la structure du RNA à partir des connaissances, ce qui lui donne la possibilité de traiter les données symboliques ; par ailleurs il est aussi possible de définir les distances sur différents types de données et permettre à Distal de traiter les données de ce type. Le traitement des données de type non numérique nécessite une phase de prétraitement. Comme opération supplémentaire, l'utilisation des méthodes dynamiques nécessitent l'opération de projection (somme des carrés de tous les autres attributs ($\sum_i x_i^2$)) et pour Distal le calcul des distances. Une autre caractéristique commune à ces algorithmes est qu'il n'existe pas de stratégie permettant de traiter les données manquantes.

Structure du réseau final et interprétation

La complexité de classement d'un exemple après l'apprentissage est essentiellement fonction de la structure du réseau. Cette complexité dépend du nombre de couches et du nombre de neurones par couche. Les algorithmes construisent les réseaux multicouches, la différence est surtout perceptible à l'organisation de la couche cachée. Pour un problème de classification, le réseau de neurones nécessaire à la résolution peut avoir une seule couche cachée (où chaque unité sera apprise pour classer un seul exemple) comme il est démontré dans [Bar93, Cyb89, HSW89]. Il est préférable d'avoir moins de couches afin de réduire le temps de propagation lors du classement, ce qui est une qualité de l'algorithme Distal. Suivant le nombre de couches du RNA construit, nous pouvons grouper ces algorithmes en deux ensembles : les algorithmes qui construisent un RNA ayant plusieurs couches cachées comme KBANN, MTiling, MTower, MUpstart, MPyramid et Perceptron-Cascade ; ce qui ne concorde pas avec les résultats théoriques ; et les algorithmes qui construisent un RNA ayant une couche cachée comme Distal.

A l'exception de KBANN, le RNA construit par ces algorithmes est vu comme une boîte noire par l'utilisateur, car il lui est impossible de justifier son architecture et d'interpréter ses résultats. Avec un RNA construit par KBANN, après l'apprentissage, l'utilisateur peut extraire un ensemble de règles qui lui permettra d'expliquer les résultats du réseau. Le nombre de régles extraites d'un RNA de grande taille est aussi élevé et rend l'exploitation de ces règles plus difficile.

L'espace mémoire nécessaire

L'algorithme Distal nécessite plus d'espace mémoire pour les opérations de distances. Les autres méthodes MUpstart, MTiling,... n'ayant généralement pas d'opérations supplémentaires coûteuses sur les données en mémoire, stockent plutôt une grande structure du réseau. Mais vu la structure du RNA construit par MTiling et comparé aux RNA construits par les autres méthodes, théoriquement, MTiling, Musptart et Perceptron Cascade ont une complexité mémoire plus élevée que les autres méthodes constructives ; et la méthode ad'hoc a une faible compléxité. En comparant aux RNA construits par les approches évolutives, ces dernières pourront exiger un espace mémoire de taille plus grande (pour le stockage de la population initiale des réseaux).

Le tableau 2.3 présente les caractéristiques du réseau de neurones obtenu des algorithmes étudiés. Dans ce tableau, la colonne "#Cachés" présente le nombre de couches cachées ; "#Total" le nombre total de neurones et "Mémoire" la complexité mémoire de l'algorithme. Cette

Algorithmes	#Cachées	#Total	Mémoire
KBANN	N_{att}	N_{att}	N_{att}
Distal	1	N_{obj}	N_{obj}
MUpstart	H	$2^H N_{cl}$	$2^H N_{cl}^2$
MTiling	H	$H N_{cl}^2$	$H N_{cl}$
MTower	H	$H N_{cl}$	$H N_{cl}$
P. Cascade	H	$2^H N_{cl}$	$2^H N_{cl}^2$
Ad'hoc	1	$max(N_{cl}, N_{att})$	$max(N_{cl}, N_{att})$

TAB. 2.3 – Tableau comparatif des caractéristiques des RNA construits. #Cachés indique le nombre de couches cachées et #Total le nombre total de neurones. Pour la méthode KBANN, le nombre de couches cachées dépend de la hiérarchie des règles.

complexité est exprimée en fonction du nombre de neurones dans le RNA construit.

La complexité en temps

Les complexités de l'algorithme KBANN et de la méthode Ad'hoc ne prennent pas en compte l'apprentissage du réseau construit ; elles ne représentent que le temps de recherche de la structure du réseau. La complexité des autres méthodes est calculée en supposant le pire des cas : pour Distal, chaque neurone dans la couche cachée ne représente qu'un seul exemple. Pour MTiling et les autres, la condition d'arrêt est l'atteinte du nombre maximum de couches cachées. Le tableau 2.4 présente la complexité en temps des algorithmes de construction de RNA.

Algorithmes	Temps en construction	Temps en Classement
KBANN	N_{att}	N_{att}
Distal	N_{obj}^2	N_{obj}
MTiling	$H N_{att} N_{cl}^3 N_{obj} N_{iter}$	$H N_{cl}^2$
MTower	$H N_{att} N_{cl}^2 N_{obj} N_{iter}$	$H N_{cl}$
MUpstart	$2^H N_{cl}^2 N_{att} N_{obj} N_{iter}$	$2^H N_{cl}$
P. Cascade	$2^H N_{cl}^2 N_{att} N_{obj} N_{iter}$	$2^H N_{cl}$
Ad'hoc	$max(N_{cl}, N_{att})$	$max(N_{cl}, N_{att})$

TAB. 2.4 – Complexité en temps (de construction et de classement) et en espace mémoire des méthodes de construction de RNA.

Algorithme d'apprentissage

L'algorithme du perceptron peut être utilisé si les exemples sont linéairement séparables (mais il est difficile de savoir pour un ensemble d'exemples si ceux-ci sont linéairement séparables). L'utilisation de ce critère permet de classer les algorithmes en deux groupes : les algorithmes qui déterminent les poids par apprentissage (MUpstart, MTiling, MTower, KBANN, méthode ad'hoc) et ceux qui ne le font pas (Distal).

Parmi les algorithmes qui font de l'apprentissage, nous pouvons aussi les classer en deux groupes : les méthodes dynamiques et les méthodes statiques. Les méthodes dynamiques (MTiling, MUpstart, MTower, Mpyramid et Cascade) apprennent les poids de connexion par une variante du perceptron à chaque mise à jour du RNA alors que les méthodes statiques (KBANN) construisent le RNA et l'apprenent avec l'algorithme de rétropropagation de l'erreur le RNA sans modifier son architecture.

La capacité de généralisation

Il est montré dans [YPH99] que Distal assure une erreur de validation nulle. Mais si le regroupement des exemples ne se fait pas alors Distal conduira à un réseau trivial [PYH97a] qui consiste à avoir autant de neurones que d'exemples dans la couche cachée.

La convergence à une erreur nulle des méthodes MTiling, MTower, MUpstart et Perceptron-Cascade est démontrée dans [PYH95, PYH97a]. Cette convergence est soumise à la condition selon laquelle l'ensemble d'apprentissage ne doit pas avoir des exemples contradictoires. Un exemple x est dit contradictoire s'il appartient à plusieurs classes, c'est-à-dire que l'on peut avoir des couples (x, y_1), (x, y_2) avec $y_1 \neq y_2$ dans l'ensemble d'apprentissage. Mais d'après les auteurs, au cas où il existerait des contradictions les performances de ces méthodes peuvent être plus mauvaises que celle d'une simple couche de neurones fonctionnant comme des perceptrons. La capacité de généralisation des RNA construits par des méthodes évolutives est fortement liée à la capacité de la méthode à trouver l'architecture appropriée ainsi des poids corrects. La généralisation des RNA construits par des méthodes ad'hoc dépend de la capacité de l'algorithme de rétropropagation de l'erreur à trouver des poids approprié tout en évitant les minimas locaux.

Les taux de généralisation des autres algorithmes dépendent de la représentativité des données d'apprentissage. Théoriquement, il n'est pas possible d'évaluer le taux d'apprentissage de ces algorithmes car les données de test sont différentes des données d'apprentissage. Nous al-

lons dans la section suivante faire une évaluation expérimentale du taux de généralisation des algorithmes.

2.7 Expérimentations

Ces algorithmes ont été testés sur les données de la base UCI [NHBM98]. L'algorithme KBANN n'a pas été expérimenté à cause de l'absence des règles décrivant les différents domaines des données de UCI. Ces expériences ont été faites par validation croisée d'ordre 10. Dans nos expériences, la distance choisie entre exemples pour Distal est la distance euclidienne parce qu'elle nous semble plus adaptée et que nous ne disposons d'aucune connaissance permettant d'orienter notre choix comme conseillent les auteurs [YPH99]. Les unités ajoutées au cours de la construction par les méthodes MUpstart, MTiling et les autres méthodes dynamiques ont été appris par l'algorithme 'Pocket Perceptron with Ratchet Modification' avec 500 itérations. Dans le but d'avoir pour ces algorithmes un RNA avec le même nombre de couches, nous avons choisi pour les algorithmes MTiling, MUpstart, Perceptron-Cascade (Cascade), MTower et Mpyramid, un nombre maximal de 10 couches et des taux d'apprentissage de 70% et 95%. Ces paramètres sont les valeurs par défaut choisies par les auteurs. Nous nous intéressons pendant ces expérimentations au taux de généralisation, au temps et au nombre de neurones de chaque méthode.

2.7.1 Données

Le tableau 2.5 présente les jeux de données utilisés pour les expériences. Dans ce tableau "Données" est le nom du jeu de données ; #Exemples est le nombre total d'exemples du jeu de données ; #Attr est le nombre d'attributs décrivant les exemples ; #Class le nombre de classes dans les exemples. Les attributs de ces données sont numériques et aucun prétraitement n'est effectué ; ces données ne possèdent pas des valeurs manquantes.

Ces données présentent plusieurs variabilités : un grand nombre d'exemples avec un nombre peu élévé d'attributs et deux classes en sortie, un grand nombre d'exemples avec beaucoup d'attributs et plusieurs classes en sortie,... Cette variabilité permet de juger le comportement de ces algorithmes dans diverses situations. Les détails sur ces données peuvent être trouvées dans [NHBM98].

Données	#Exemples	#Attr	#Class
Australian	690	14	2
Balance	625	4	3
Image-Segmentation (Image)	2310	19	7
Ionosphere	351	34	2
Iris plants	150	4	3
Letter recognition (Letter)	20 000	16	26
Libras-movement (Libras)	360	90	15
Lymphography (Lympho)	148	18	4
Magico4	19 020	10	2
Optical-digits (Optical)	5620	64	10
Shuttle	14500	9	7
Wine	178	13	2

TAB. 2.5 – Description des données utilisées pour les expérimentations.

2.7.2 Résultats

Nous présenterons dans cette section les taux de généralisation obtenus de ces algorithmes. Dans les tableaux 2.6 et 2.7, les taux de généralisation sont obtenus en imposant comme contrainte de convergence, des taux d'apprentissage respectifs de 95% et 70%. Dans les tableaux présentant ces résultats, le symbole '*' indique que l'algorithme n'a pas convergé.

Le tableau 2.6 présente les résultats obtenus au cours de l'expérimentation de ces algorithmes sur les données du tableau 2.5 avec un taux d'apprentissage de 95%. Dans les conditions expérimentales que nous avons décrites, on note l'échec des algorithmes MTiling sur le jeu de données Magic04, de MUpstart sur les données Letter, magic04 et Optical, de Distal sur les données Letter et Magic04. Les algorithmes MTiling et MUpstart ont atteint le nombre maximal de couches sans converger alors l'exécution de Distal a été interrompue après 24 heures. Les échecs de MTiling et MUpstart sont dûs au fait que le taux d'apprentissage souhaité est trop contraignant ; car avec ce taux à 70% MUsptart classe à 95,75% les données 'Opdigits' ; à 70% de taux d'appprentissage souhaité, MTiling classe les données 'Magic04' à 80.81% et MUpstart classe respectivement les données 'Letter' et 'Magic04' à 70,60% et à 77,97%. Ces résultats permettent d'envisager de réduire le taux d'apprentissage lorsque le nombre des exemples est élevé (Letter, Optical, Magic04) ; et de garder ce taux élevé pour les données de taille faible (le cas de 'Libras' où 70% Cascade classe correctement 13,89% des exemples alors qu'à 95%, il classe correctement 56,65%). Nous utiliserons les résultats du tableau 2.6 dans les comparai-

Données	MTiling	MTower	MUpstart	Cascade	Mpyramid	Distal
Australian	81,16	71,01	79,71	83,37	76,71	82,36
Balance	88,71	82,26	91,54	93,55	82,26	72,40
Image	92,64	93,94	96,10	94,81	94,82	91,18
Ionosphere	85,71	94,29	85,71	94,29	85,82	82,65
Iris plants	100	100	100	92,09	86,67	90,67
Libras	83,22	70,56	86,23	56,65	78,62	76,25
Lympho	92,86	78,56	79,66	78,87	78,58	72,57
Shuttle	99,38	97,59	96,83	94,55	96,07	89,65
Wine	88,24	82,22	100	76,47	75,09	91,9
Moyenne	**90,21**	**85,60**	**90,64**	**84,96**	**83,85**	**83,29**
Optical	96,81	98,40	*	96,26	96,65	80,05
Letter	72,55	76,80	*	72,45	78,45	*
Magic04	*	77,29	*	80,55	77,55	*
Moyenne générale	-	**85,27**	-	**84,52**	**83,93**	-

TAB. 2.6 – Taux de généralisation obtenus des méthodes dynamiques sur les données du tableau 2.5 avec le taux d'apprentissage souhaité de 95%.

sons parce que non seulement ces résultats sont meilleurs (suivant la moyenne) que les résultats du tableau 2.7, mais aussi parce qu'ils ont obtenus en utilisant les paramètres par défaut définis par les auteurs.

Le comportement des algorithmes MTiling, MTower, MUptart, Cascade et MPyramid est presque similaire (en cas de convergence) à l'exception des données Libras où Mpyramid présente un très mauvais taux par rapport aux autres.

En comparant les moyennes de taux de généralisation en cas de succès, nous pouvons classer les algorithmes du premier au dernier dans l'ordre suivant : MUpstart, MTiling, MTower, Cascade, MPyramid et Distal. Les succès de MUpstart et de MTiling sont étroitement liés au taux d'apprentissage souhaité, un taux très faible peuvent conduire à une mauvaise généralisation alors qu'un taux d'apprentissage élevé peut conduire à la non convergence.

Le tableau 2.8 présente les temps moyen d'exécution des différentes méthodes sur les données du tableau 2.5. L'histogramme 2.11 présente une comparaison logarithmique de ce temps. On constate que pour ces données , le temps d'exécution des méthodes MTiling, MPyramid et Distal sont dans un cas général plus élevés que le temps des autres méthodes. Le temps de Distal augmente lorsque le nombre d'exemples devient grand. Cela est dû au fait que les distances

Données	MTiling	MTower	MUpstart	Cascade	Mpyramid
Australian	78,26	75,36	78,26	73,91	84,06
Balance	93,55	91,94	98,39	95,16	95,16
Image	95,24	80,00	95,67	93,07	95,24
Ionosphere	88,57	87,45	77,14	82,26	80,00
Iris plants	86,67	93,33	30,00	100	80,00
Libras	47,22	36,11	38,89	13,89	55,56
Lympho	85,71	78,57	85,71	78,57	71,43
Shuttle	94,55	96,07	94,26	96,07	96,34
Wine	100	100	88,24	100	82,35
Moyenne	**84,98**	**81,96**	**74,70**	**80,54**	**83,05**
Optical	95,55	96,44	95,75	95,91	96,62
Letter	66,23	67,75	70,60	72,25	75,05
Magic04	80,81	77,39	77,97	70,50	78,79
Moyenne générale	**84,41**	**81,72**	**76,47**	**80,93**	**82,59**

TAB. 2.7 – Taux de généralisation obtenus des méthodes dynamiques sur les données du tableau 2.5 avec le taux d'apprentissage souhaité de 70%.

sont calculées entre les exemples.

Le tableau 2.9 présente le nombre de victoires de la méthode i (représentée à la ligne i) sur la méthode j (représentée à la colonne j) sur les données du tableau 2.5. De ce tableau, on observe que MUpstart a plus de victoires que les autres méthodes alors que Distal est le dernier.

Le tableau 2.10 présente une comparaison expérimentale des résultats des implémentations de la plate forme de fouille de données WEKA [WF05] des méthodes :

- MLP, la méthode ad'hoc de construction des RNA.
- IB1, Une méthode de classification à base des instances utilisant l'approche des plus proches voisins ;
- PART et J48, deux méthodes à base des arbres de décisions.
- BayesNet, une méthode bayesienne.
- SMO, une méthode à base de SVM (Séparateurs à Vaste Marge).

Sur les données utilisées pour ces expériences, la moyenne de la méthode MLP est la plus élévée et la méthode bayésienne BayesNet la dernière. En comparant ces moyennes à celles du tableau 2.6, les méthodes MLP, MUpstart et MTiling sont respectivement première, deuxième et troisième. Les méthodes Distal et MPyramid sont classées après toutes les méthodes. Les

Données	MTiling	MTower	MUpstart	Cascade	Mpyramid	Distal
Australian	158,31	15,42	75,38	20,05	49,75	98,79
Balance	6,20	12,51	1,42	17,35	2,07	10,3
Image	11,20	6,24	2,24	25,13	18,32	68,04
Ionosphere	36,50	27,44	22,07	2,37	0,77	1,02
Iris plants	0,35	0,39	0,32	0,38	0,33	0,2
Libras	1,76	1,21	1,75	1,34	1,5	1,3
Lympho	2,89	4,41	3,32	2,12	3,02	1.70
Shuttle	1200,1	650,64	198,03	806,82	61,39	1352,4
Wine	24,08	8,53	1,46	8,06	0,84	0,2
Optical	161,05	107,79	*	200,71	126,04	201,0
Letter	264,79	250,03	*	2025,8	408,19	*
Magic04	*	248,03	*	337,01	1245	*

TAB. 2.8 – Temps moyen en secondes d'exécution des méthodes dynamiques obtenus par la validation croisée d'ordre 10.

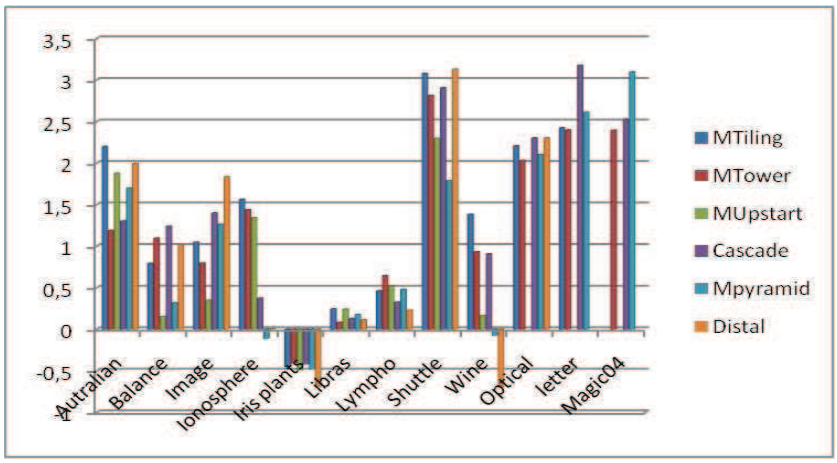

FIG. 2.11 – Comparaison logarithmique des temps d'exécution des différents algorithmes sur les données 2.5.

Données	MTiling	MTower	MUpstart	Cascade	MPyramid	Distal
MTiling	+	6	5	7	8	9
MTower	5	+	5	6	5	9
MUpstart	5	6	+	6	8	8
Cascade	4	4	6	+	9	10
Mpyramid	4	6	4	2	+	9
Distal	3	3	2	2	3	+

TAB. 2.9 – Tableau des victoires de la méthode i (ligne) sur la méthode j (colonne) données du tableau 2.5

.

Données	MLP	IB1	PART	J48	BayesNet	SMO
Australian	84,92	80,00	83,62	85,21	84,92	85,5
Balance	97,6	66,72	77,6	63,2	91,36	91,04
Image	96,36	97,14	97,79	96,79	91,60	93,03
Ionosphere	91,17	86,32	91,74	91,45	89,46	88,60
Iris plants	97,33	95,33	94,0	96,0	92,67	96,0
Libras	81,39	85,83	70,55	69,72	60,55	75,0
Lympho	81,08	75,67	81,75	75,0	79,05	86,48
Shuttle	99,67	99,83	99,88	99,88	99,37	96,0
Wine	97,75	94,94	92,69	93,26	98,87	98,31
Moyenne	**91,92**	**86,86**	**87,73**	**85,61**	**87,54**	**90,0**
Magic04	85,87	80,93	85,46	85,05	72,67	79,15
Letter	98,01	95,91	89,05	87,89	74,31	82,42
Optical	98,39	98,66	92,43	90,23	92,32	98,34
Moyenne	**92,41**	**88,01**	**88,02**	**86,09**	**85,74**	**89,22**

TAB. 2.10 – Comparaison des taux de généralisation obtenus sur les données du tableau 3.3 par d'autres méthodes de classification

Données	MUptart	MTiling	MLP	IB1	PART	J48	BayesNet	SMO
MUpstart	-	5	3	5	4	4	6	6
MTiling	5	-	3	4	5	5	6	4
MLP	9	9	-	8	8	8	10	9
IB1	7	4	4	-	5	7	7	6
PART	8	7	4	7	-	7	9	5
J48	8	7	4	5	4	-	8	5
BayesNet	6	6	1	5	3	4	-	4
SMO	6	8	3	6	7	6	8	-

TAB. 2.11 – Tableau des victoires de la méthode i (ligne) par rapport à la méthode j (colonne) sur les données du tableau 2.5.

autres méthodes que neuronales occupent des rangs intermédiaires.

Les résultats présentés dans le tableau 2.11 comparent les victoires des méthodes dynamiques à celles des autres méthodes. Elles montrent le nombre de victoires de l'algorithme i sur l'algorithme j. MLP a plus de victoires que toutes les autres et BayesNet en a le plus petit nombre.

Les résultats des méthodes dynamiques dans certaines situations peuvent être justifiés par le choix des paramètres utilisés pour construire le réseau. L'augmentation du nombre de couches cachée, le taux d'apprentissage souhaité ou le choix d'un autre algorithme d'apprentissage pourrait augmenter leurs performances. Dans les cas particuliers de MTiling, de MUpstart et de Distal, qui échouent quand la taille des données est relativement grande (avec un taux d'apprentissage élevé), les échecs sont probablement liés à la taille de la matrice des distances calculées (dans le cas de Distal) et à la taille du RNA (dans les cas de MTiling et de MUpstart).

2.8 Conclusion

La détermination d'une architecture d'un réseau de neurones pour la résolution d'un problème reste un problème ouvert. Dans ce chapitre, nous avons présenté quelques méthodes pour la résolution de ce problème. Le choix de la méthode appropriée peut dépendre des connaissances que l'utilisateur a du problème.

Si l'utilisateur dispose des connaissances sous forme de règles d'inférence, la méthode

KBANN est envisageable pour la construction du réseau. Non seulement la complexité de KBANN est linéaire en nombre de variables mais aussi le réseau construit est compréhensible. Il peut justifier la présence de chaque neurone par le fait que celui-ci représente une variable dans l'ensemble des règles et les connexions par les liens de précédence entre ces variables.

En l'absence de règles, le choix devrait être porté sur l'une des méthodes dynamiques ou ad'hoc. Vu les résultats expérimentaux sur les données utilisées, Mupstart et MTiling se comportent mieux que les autres méthodes mais ne convergent si le taux d'apprentissage désiré par l'utilisateur est très contraignant.

Dans ce chapitre, l'analyse théorique des algorithmes de construction de RNA a été aussi présentée. De ces analyses, on peut noter que théoriquement, ces algorithmes ont un comportement similaire. Quelque soit la méthode de construction des réseaux de neurones, le choix des paramètres reste toujours très problématique. Pour la méthode Distal, il faut trouver l'influence du choix de calcul de distance entre exemples ou entre attributs ; pour les méthodes MUpstart, MTiling, MTower, MPyramid, Cascade correlation, quelle est l'influence de l'algorithme d'apprentissage des unités ajoutées et comment choisir le nombre maximal de couches cachées ? De plus il n'est pas possible d'extraire la connaissance apprise après l'apprentissage. La méthode KBANN ne pose pas de problèmes de paramètres et construit un réseau interprétable ; par contre nécessite des connaissances du domaine.

Avec les méthodes présentées dans ce chapitre, l'utilisateur est toujours confronté à des problèmes : le réseau est vu comme une boîte noire, il doit disposer des connaissances apriori. Il faut alors penser à une approche de construction de RNA interprétable sans connaissances apriori. Nous allons dans le prochain chapitre présenter comment définir l'architecture du RNA à partir des treillis de Galois.

Chapitre 3

Construction des réseaux de neurones à partir des treillis de Galois

Sommaire

3.1 Introduction

Dans le chapitre précédent, nous avons présenté un ensemble de méthodes permettant de construire l'architecture des réseaux de neurones pour résoudre un problème de classification. Malheureusement, ces méthodes présentent quelques limites :

- Pour les approches qui définissent cette architecture à partir d'un ensemble de connaissances pour définir la structure, le RNA construit est compréhensible. Mais l'utilisation de ces approches nécessitent des connaissances apriori sur le domaine. Ces méthodes ne peuvent pas être utilisées si les concepteurs ne disposent pas de ces connaissances apriori.
- Pour les approches qui construisent le RNA à partir des données, le réseau obtenu est présenté comme une 'boîte noire', donc incompréhensible. En effet il n'est pas possible d'expliquer comment le réseau calcule les solutions proposées lorsqu'il est construit par ces méthodes. De plus aucune justification ne peut être donnée à la présence d'un neurone ou d'une connexion dans le réseau.

Dans le but de dépasser ces limites, ce chapitre présente un rapprochement entre les treillis de Galois et les réseaux de neurones. Il est en fait question dans ce chapitre de répondre à la question : *"Comment la théorie des treillis de Galois peut-elle aider à construire le RNA en évitant le problème de l'interprétabilité des modèles neuronaux lorsque l'utilisateur ne dispose pas des connaissances apriori ?"*.

Nous présentons dans ce chapitre une nouvelle approche basée sur les treillis de Galois (ou treillis de concepts) pour la définition d'une architecture de RNA multicouches. Un treillis de Galois permet de représenter les données sous forme intension / extension où intension est un ensemble d'attributs et extension un ensemble d'objets. Il est construit à partir des données sans aucune connaissance apriori. Le graphe de Hasse du treillis de Galois permet de représenter graphiquement la relation de spécialisation / généralisation établie entre les éléments du treillis. Chaque élément du treillis peut être considéré comme un neurone, et la relation de généralisation / spécialisation permet alors de définir les connexions entre les neurones des différentes couches du RNA. Dans [MNTT08, MNTT09, TMNT07a], les modèles CLANN ("Concept Lattice-based Artificial Neural Network") et MCLANN (MultiClass CLANN) basés sur les treillis de Galois sont proposés pour répondre aux questions du choix d'architecture d'un RNA pour la résolution d'un problème donné. Les avantages de ces approches sont :

– la capacité de construire un réseau ayant une architecture compréhensible sans connaissance apriori et capable de bien classer les données ; les treillis de Galois ont été utilisés pour l'extraction de règles d'implication [GD86] et d'association [BBR03, PBTL99] et en classification supervisée [FFNMN04, MNN05]. Ils peuvent présenter un double atout pour la recherche de cette architecture interprétable, et pour une meilleure classification des données.
– la possibilité de justifier la présence des différents neurones et des liens de connexion du réseau ; en effet chaque neurone correspond à un concept formel formé d'un ensemble d'objets et d'un ensemble d'attributs ; et chaque lien de connexion du RNA matérialise une relation entre un sous-concept et un super-concept.

En outre l'effet combinatoire du treillis de Galois est ici limité dans CLANN par la génération d'un sup-demi-treillis, en utilisant des approches de construction par niveau, contraintes par des heuristiques de fréquence ou support, ou simplement de niveau (ou profondeur) du demi-treillis. CLANN est limité aux ensembles de données ayant seulement deux classes. L'extension de CLANN à MCLANN apporte alors une solution pour les cas où on est en présence de plusieurs classes.

Ce chapitre est organisé de la manière suivante : la section suivante introduira les treillis de Galois et les contraintes utilisées pour sélectionner les concepts. Les sections 3 et 4 décriront respectivement les approches CLANN et MCLANN, le processus de construction du demi-treillis et la transformation de ce demi-treillis en RNA. Nous terminerons par l'étude expérimentale que nous avons menée sur les données de la base UCI [NHBM98].

3.2 Treillis de Galois

Les treillis de concepts formels [BM70, Bir40, Bir67] (ou treillis de Galois) sont une structure mathématique permettant de représenter les classes non disjointes sous-jacentes à un ensemble d'objets (exemples, instances, tuples ou observations) décrits à partir d'un ensemble d'attributs (propriétés, descripteurs ou items) [MNN05]. Ces classes sont aussi appelées concepts formels ou nœuds du treillis. Chaque nœud est composé d'un ensemble d'objets ayant en commun un ensemble de propriètés (ou attributs). Ils permettent de visualiser les données et de les représenter graphiquement. Les treillis de Galois ont été étudiés de manière approfondie dans le cadre de l'analyse formelle des conceptss (AFC [GR99, Wil82]). La notion centrale de l'AFC est la dualité encore appelée connexion de Galois. Cette dualité est observée entre deux types d'objets mis en relation ; par exemple entre objets et attributs, entre document et termes, etc. Le

treillis de Galois permet alors de représenter les classes non disjointes entre ces objets et ces attributs. Nous commencerons cette partie par quelques définitions.

3.2.1 Généralités

Nous commencerons cette section par les notions de la théorie des ensembles et des relations. Ces deux notions sont fondamentales en analyse formelle des concepts.

Ensembles et relations

Définition 14 *Un **ensemble** est un groupe d'objets (éléments) ayant en commun certaines caractéristiques, et manipulés par les mêmes opérations.*

Les entiers, les réels ou les chaînes de caractères sont des exemples d'ensemble de données. Un ensemble peut être défini en extension (liste exhaustive de ses éléments) ou en compréhension (sous forme d'une formule que doivent vérifier ses éléments). Un ensemble est dit fini (resp. infini) s'il a un nombre fini (resp. infini) d'éléments. Nous ne manipulerons que des ensembles finis définis en extension[2]. Les liens peuvent être établis entre les éléments d'un ou plusieurs ensembles. Ces liens définissent la relation entre les éléments de ces ensembles.

Définition 15 *Une **relation** est une mise en correspondance des éléments d'un ou de plusieurs ensembles. Formellement une relation R définie sur n ensembles E_i est une partie de produit cartésien $X(E_i) = E_1 \times E_2 \times ... \times E_n$; ($R \subseteq X(E_i)$).*

Quand une relation ne met en correspondance que les éléments de deux ensembles ($n = 2$), on parle de **relation binaire**. Alors la relation binaire R définie entre les éléments de E_1 et E_2 sera formellement définie comme un sous-ensemble du produit cartésien de E_1 et de E_2 ($R \subseteq E_1 \times E_2$; $R = \{(x, y)/x \in E_1 \; et \; y \in E_2\}$). On dira que R est définie sur $E_1 \times E_2$. R est une **application** si à chaque élément de E_1 est associé au plus un élément de E_2.

Définition 16 *Une relation R définie sur $E \times E$ est une **relation d'ordre** [DP02, Dom02] si elle vérifie les propriétés suivantes :*

1. *Réflexivité : $\forall x \in E \; (x, x) \in R$;*

[2] nous écrirons $abcd$ pour parler de l'ensemble $\{a, b, c, d\}$

2. *Antisymétrie :* $\forall x, y \in E, (x, y) \in R \, et \, (y, x) \in R \Longrightarrow x = y$;
3. *Transitivité :* $\forall x, y, z \in E, (x, y) \in R \, et \, (y, z) \in R \Longrightarrow (x, z) \in R$

L'ensemble E muni d'une relation d'ordre sera noté $(E, \leqslant)$; et on dira que E est un ensemble ordonné. Pour tout élément x de E, on peut alors définir ses **majorants** par $[x) = \{y \in E / x \leqslant y\}$ et ses **minorants** par $(x] = \{y \in E / y \leqslant x\}$.

La restriction de la relation d'ordre définie sur E à un sous-ensemble E_1 de E fait de ce sous-ensemble E_1 un **sous-ensemble ordonné** de E.

Définition 17 *Soit $(E, \leqslant)$ un ensemble ordonné et E_1 un sous-ensemble ordonné non vide de E. On dit qu'un élément $x \in E$ est un **majorant** (resp. **minorant**) de E_1 si pour tout $x_1 \in E_1$ on a $x_1 \leqslant x$ (resp. $x \leqslant x_1$).*

Si x est un majorant (resp. minorant) de E_1 et que pour tout autre majorant (resp. minorant) y de E_1, on a $x \leqslant y$ (resp. $y \leqslant x$) alors x est appelé **supremum** ou borne supérieure (resp. **infimum** ou borne inférieure) de E_1 [DP02, Dom02].

Définition 18 *Un **treillis** est un ensemble ordonné $(E, \leqslant)$ tels que pour toute paire d'éléments $x, y \in E$, il existe un infimum et un supremum pour x et y. On définit aussi un treillis comme un ensemble muni d'un ordre partiel dans lequel chaque paire d'éléments a un unique supremum et un unique infimum [Dom02].*

Définition 19 *Un demi-treillis est un ensemble partiellement ordonné et fermé soit sur le supremum, soit sur l'infimum [DP02, GR99, Wil82].*

Un sup-demi-treillis (resp. inf-demi-treillis) est alors un ensemble ordonné $(E, \leqslant)$ où toute paire d'éléments x et y de E admet un supremum (resp. infimum).

Analyse formelle des concepts - AFC

En AFC, l'adjectif 'formel' est utilisé pour qualifier les notions manipulées. L'adjectif formel permet de faire la différence entre les concepts extraits dans le cadre du AFC et les concepts philosophiques qui sont utilisés par les Hommes pour s'exprimer et communiquer. De ce fait, les éléments des types d'objets mis en relation en AFC sont appelés 'objets formels' et 'attributs

formels'. Dans les applications, les objets formels sont des objets et les attributs formels leurs caractéristiques ou attributs. L'ensemble constitué des objets formels, des attributs formels et de la relation entre objets et attributs est appelé 'contexte formel'. Dans la suite, nous allons omettre l'adjectif 'formel' pour parler de ces notions (on écrira par exemple 'objet' au lieu de 'objet formel').

Définition 20 *Un **contexte formel** est un triplet $C = (O, A, R)$ où O est un ensemble fini non vide d'objets, A un ensemble fini non vide d'attributs et R une relation binaire entre les éléments de O et ceux de A. Formellement $R \subseteq O \times A; R = \{(o, a) \,/ o \in O \text{ et } a \in A\}$.*

Un contexte C peut être représenté comme une matrice binaire M où les lignes sont étiquetées par les éléments de O (voir tableau 3.1), les colonnes par les éléments de A et $M_{i,j} = 1 \iff (o, a) \in R$ et 0 sinon ; ($o \in O$ et $a \in A$). Il peut être aussi représenté comme une base de transactions, une ligne (transaction) i est la collection des attributs que vérife l'objet o_i ; A (resp. O) est encore appelé ensemble des items (resp. transactions).

Soit le contexte formel $C = (O, A, R)$. Soit l'application f de l'ensemble des parties de O (ensemble de tous les sous-ensembles de O) noté par 2^O dans l'ensemble des parties de A noté 2^A. L'application f associe à un ensemble des objets $O_1 \subseteq O$, l'ensemble des attributs $a \in A$ communs à tous les objets de O_1.

$$f : 2^O \longrightarrow 2^A$$
$$f(O_1) = \{a \in A \,/\, \forall o \in O_1 \,,\, (o, a) \in R\}.$$

Soit g une application de l'ensemble des parties de A dans l'ensemble des parties de O. Elle associe à un ensemble d'attributs $A_1 \subseteq A$, l'ensemble d'objets $o \in O$ communs à tous les attributs $a \in A_1$.

$$g : 2^A \longrightarrow 2^O$$
$$g(A_1) = \{o \in O \,/ \forall a \in A_1 \,,\, (o, a) \in R\}.$$

Définition 21 *Le couple d'applications (f, g) définit une **correspondance de Galois** entre l'ensemble des parties de O et l'ensemble des parties de A. Les applications $f \circ g$ et $g \circ f$ sont appelées les opérateurs de fermeture de la correspondance de Galois [Bir67].*

Définition 22 *On dit que A_1 est **fermé** si et seulement si $A_1 = f \circ g(A_1)$. $f \circ g(A_1)$ (resp. $g \circ f(O_1)$) définit une relation de fermeture sur A (resp. O).*

Propriétés :

Soit $\phi(x)$ une relation de fermeture définie sur un ensemble E et $\leq$ une relation d'ordre définie sur $E \times E$. Les propriétés d'une relation ϕ sont :

1. Idempotence : $\forall x \in E,\ \phi(\phi(x)) = \phi(x)$.
2. Isotonie : $\forall x, y \in E; x \leq y \Longrightarrow \phi(x) \leq \phi(y)$.
3. Extensivité : $\forall x \in E, x \leq \phi(x)$.

Définition 23 *Soient $O_1 \subseteq O$ et $A_1 \subseteq A$, la paire (O_1, A_1) est un **concept formel** ssi $f(O_1) = A_1$ et $g(A_1) = O_1$.*

A_1 (resp. O_1) est l'intension (resp. l'extension) du concept. Les éléments O_1 et A_1 composant le couple dans un concept formel sont des fermés. Un concept formel est aussi appelé hyper-rectangle ou rectangle maximal [DP02, Dom02, Pol98].

Définition 24 *Soient (O_1, A_1) et (O_2, A_2) deux concepts, et $\leq$ une relation définie sur l'ensemble des concepts extraits du contexte par : $(O_1, A_1) \leq (O_2, A_2)$ si $O_1 \subseteq O_2\,(A_1 \supseteq A_2)$. (O_1, A_1) est le **successeur** de (O_2, A_2) et (O_2, A_2) le **prédécesseur** de (O_1, A_1) s'il n'existe aucun autre concept entre eux. La relation $\leq$ définit une relation d'ordre sur l'ensemble L des concepts [DP02, GR99, Wil82].*

Définition 25 *L'ensemble L des concepts formels muni de cette relation d'ordre est un **treillis de Galois**.*

Cette relation d'ordre est aussi appelée relation successeur / prédécesseur entre les concepts. Le concept (O_1, A_1) est appelé sous-concept de (O_2, A_2) et (O_2, A_2) super-concept de (O_1, A_1).

Définition 26 *Le **graphe de Hasse** est la représentation graphique d'une relation d'ordre.*

Une représentation graphique de la relation de précédence entre les concepts où il existe un lien entre deux concepts c_1 et c_2 si et seulement si c_1 est un successeur de c_2, est un graphe de Hasse. Dans cette représentation, les nœuds sont des concepts et il existe un lien entre deux nœuds s'il existe la relation successeur / prédécesseur entre ces concepts.

Exemple 2 *Le tableau 3.1 est un exemple de contexte formel. Ce contexte est obtenu de l'exemple 1.1 du chapitre 1 après le prétraitement suivant :*

Objets	a	b	c	d	e	f	Classe
1	1	1	1	1	1	0	aérien
2	1	1	0	1	0	0	terrestre
3	1	0	0	1	0	0	terrestre
4	0	0	0	1	0	0	terrestre
5	0	1	1	1	1	0	aérien
6	0	0	0	1	0	1	aquatique
7	1	1	1	0	0	1	aquatique
8	0	0	1	1	0	0	terrestre

TAB. 3.1 – Exemple de contexte formel présenté sous forme d'une matrice binaire. A chaque objet est associée sa classe.

– *a=1 si taille* $\leq$ *1 et 0 sinon ;*
– *b=1 si poids* $\leq$ *40 et 0 sinon ;*
– *c=1 si le nombre de pattes est inférieur ou égal à 2 et 0 sinon ;*
– *d=1 si l'animal peut marcher et 0 sinon ;*
– *e=1 si l'animal peut voler et 0 sinon ;*
– *f=1 si l'animal peut nager et 0 sinon.*

Dans ce contexte la relation I est définie entre $O = \{1, 2, 3, 4, 5, 6, 7, 8\}$ *et* $A = \{a, b, c, d, e, f\}$.

Le treillis de Galois construit de ce contexte est présenté la figure 3.1.

Le couple $(125, bd)$ *est un concept formel ;* bd *est son intention et 1345 son extension. (15,*$bcde$*) est un sous-concept de* $(125, bd)$ *et réciproquement* $(125, bd)$ *est un super-concept de* $(15, bcde)$*. Par contre* $(15, bcd)$ *n'est pas un concept formel car* bcd *n'est pas fermé ; le fermé de* bcd *est* $bcde$*.*

3.2.2 Algorithmes de construction des treillis de Galois

Plusieurs algorithmes sont présentés dans la litérature pour la construction des treillis de Galois. Ces algorithmes peuvent être regroupés en deux groupes : les algorithmes qui génèrent simplement les concepts et les algorithmes qui en plus de la génération des concepts construisent le graphe de Hasse du treillis [Gue91, KO02]. Parmi les algorithmes étudiés [FMN04, KO02], l'algorithme de Bordat en plus de la liste des concepts construit le graphe de Hasse de la relation d'ordre entre les concepts.

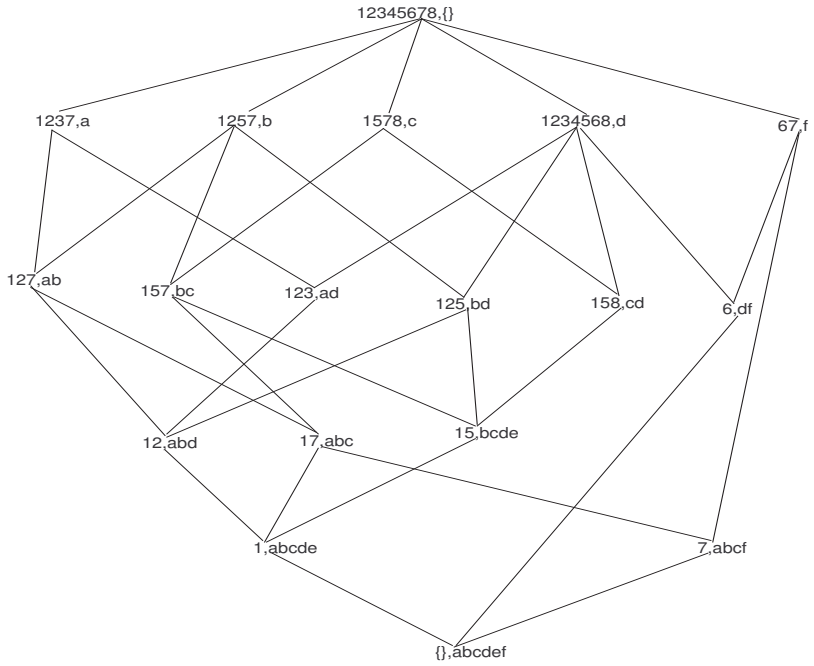

FIG. 3.1 – Treillis de Galois construit à partir du contexte du tableau 3.1. La colonne 'Classe' n'est pas prise en compte.

L'algorithme de Bordat [Bor86] engendre pour un concept (O_1, A_1) les concepts (O'_1, A'_1) qu'il couvre. La relation de couverture permet ainsi d'établir la relation sous-concept/super-concept entre les concepts. (O_1, A_1) couvre (O'_1, A'_1) si (O'_1, A'_1) est successeur de (O_1, A_1) sans intermédiaire. Le treillis est construit à partir du concept $(g(\{\}), \{\})$. La liste est parcourue séquentiellement et pour chaque concept de la liste, l'algorithme construit les concepts qu'il couvre et les insère dans la liste s'ils ne sont pas déjà dans la liste. Il s'arrête quand le parcours de la liste finit sans faire d'insertion de nouveaux concepts.

Pour éviter l'effet combinatoire inhérente à la structure des treillis de Galois, plusieurs travaux [MN93, MNN05] proposent des méthodes permettant de sélectionner les concepts et de réduire le temps et l'espace mémoire nécessaire pour l'exécution de l'algorithme.

3.2.3 Heuristiques ou contraintes de sélection de concepts

Cette section présente les contraintes qui peuvent être utilisées pour sélectionner les concepts.

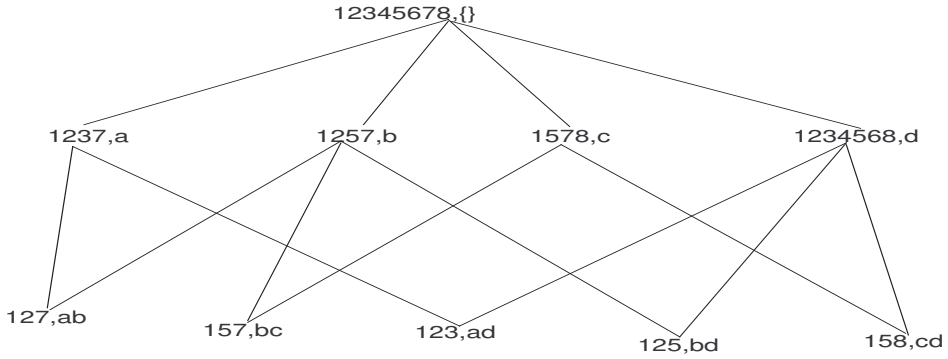

FIG. 3.2 – Demi-treillis de Galois (a) construit à partir des exemples en appliquant la contrainte de sélection 'Fréquence=30%' du tableau 3.1.

Fréquence relative

La fréquence relative s d'un concept (O_1, A_1) est le rapport entre la cardinalité de son extension et la cardinalité de l'ensemble des objets ($s = |O_1| \,/\, |O|$).

Définition 27 *Soit α appelé aussi fréquence minimale ou $minsupp$ un pourcentage (ou un ratio) spécifié par l'utilisateur. Formellement, le concept (O_1, A_1) est fréquent ssi sa fréquence relative est supérieure à α ($|O_1| \,/\, |O| \geq \alpha$).*

La fréquence est une contrainte anti-monotone qui va permettre d'élaguer le treillis et de réduire la complexité de construction. Le mot 'fréquence' sera utilisée dans la suite pour parler de fréquence relative.

Exemple 3 *La figure 3.2 présente le demi-treillis formé des concepts du treillis de la figure 3.1 vérifiant la contrainte 'Fréquence minimale = 30%'.*

Cohérence du concept

Soient α et β deux pourcentages (ou ratios) spécifiés par l'utilisateur. L'ensemble des données d'apprentissage O est divisé en deux sous-ensembles (exemples (O^+) et contre exemples (O^-)). Les contraintes suivantes [MN93, MNN05] sont appliquées pour sélectionner les concepts à conserver dans le demi-treillis : validité et quasi-cohérence.

Définition 28 *Un concept (O_1, A_1) est valide si A_1 est vérifié par au moins un certain nombre d'objets, formellement (O_1, A_1) est valide si $|O_1| \geq \alpha$ avec $0 < \alpha \leq |O^+|$.*

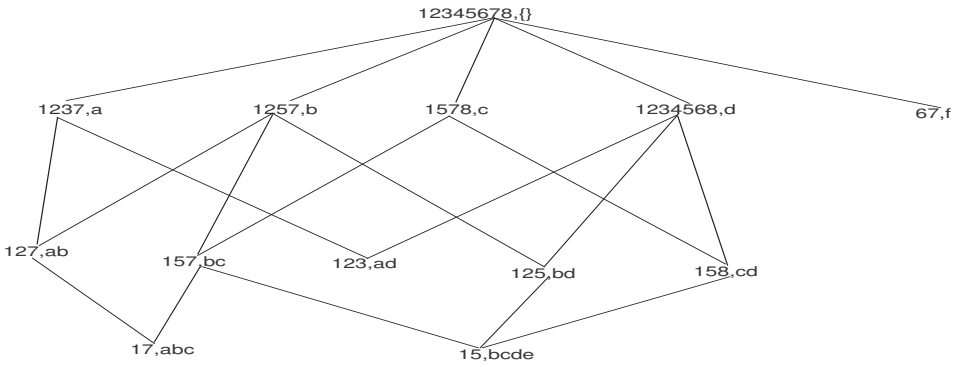

FIG. 3.3 – Demi-treillis de Galois (a) construit à partir des exemples en appliquant comme contrainte de cohérence : '$\alpha = 50\%;\ \beta = 60\%$'.

Définition 29 *Un concept (O_1, A_1) est quasi-cohérent s'il est valide et que sa description reconnaît peu de contre exemples ($\left|O_1^+\right| \geq \alpha$ et $\left|O_1^-\right| \leq \beta$).*

Nous dirons qu'un concept est *cohérent* s'il est valide et quasi-cohérent.

Exemple 4 *Supposons que les classes aérien et $aquatique$ forment la classe + et la classe $terrestre$ la classe -. La figure 3.3 présente un demi-treillis obtenu à partir du treillis 3.1 ; la contrainte appliquée pour sélectionner les concepts est la pertinence avec les valeurs $\alpha = 50\%$ et $\beta = 60\%$.*

Profondeur ou hauteur du treillis

La profondeur d'un concept est définie comme le nombre d'arêtes entre le supremum et ce concept. Cette contrainte est facilement applicable lorsque les concepts sont générés par niveau. L'utilisateur peut l'utiliser pour limiter le nombre de couches du réseau. Pour un réseau de k couches ($k > 0$), il suffit de construire un demi-treillis de profondeur k ($k \leq min(N_{obj}, N_{att})$). Cette méthode de sélection a aussi pour avantage que le demi-treillis aura toujours au moins un concept.

Exemple 5 *La figure 3.4 présente un exemple de RNA obtenu à partir du tableau 3.1. Le demi-treillis (figure 3.4) est construit en appliquant la contrainte 'Profondeur du treillis = 2'.*

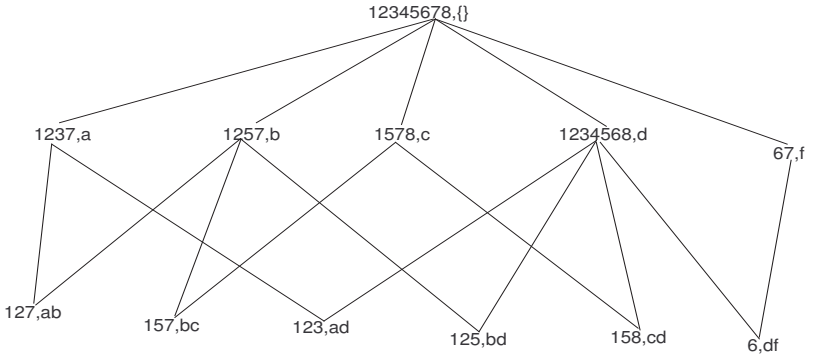

FIG. 3.4 – Demi-treillis Galois construit à partir du treillis 3.1, la profondeur de demi-treillis choisi est 2.

3.2.4 Usage des treillis de Galois en fouille de données

Les treillis de Galois sont utilisés en fouille de données pour les tâches d'extraction de règles d'association [PBTL99, BYHMN06, BBR03], et de classification supervisée [FFNMN04, MNN05]. De nombreux travaux décrits dans [BYGMN09] sont basés sur le treillis de Galois pour l'extraction de bases génériques de règles [GD86], ensemble minimal à partir duquel les autres règles peuvent être calculées.

Les treillis de Galois sont aussi utilisés pour l'organisation des données et l'interprétation [Pol98]. Leur construction a été étendue aux données symboliques [MDNST08, KAMN10] et aux données floues [BYJ01].

De nos jours, des rapprochements se font plus en plus entre les treillis de Galois et les RNA : utiliser un RNA pour calculer les fermés d'un contexte [Rud07]. Dans cet article [Rud07], l'auteur montre comment obtenir les fermés d'un contexte binaire à partir d'un réseau de neurones. A l'inverse de [Rud07], nous allons montrer comment définir le réseau de neurones à partir d'un treillis de Galois. Nous présentons l'utilisation des treillis de Galois à la recherche d'architecture de réseau de neurones, à travers les systèmes CLANN (Concept Lattices-based Artificial Neural Network) et MCLANN (Multi-category CLANN) qui construisent l'architecture du RNA à partir de la structure d'un treillis de Galois.

3.3 Construction des réseaux de neurones à partir des treillis de Galois

Les interprétations de ces sommets et de leurs interconnexions du graphe de Hasse représentant la relation d'ordre entre eux ont été bien étudiées dans la littérature [PBTL99, BYHMN06, BBR03, Pol98, GR99]. Nous utilisons cette structure (de graphe) dans ce travail pour définir la structure du réseau de neurones dont l'apprentissage permettra de classer les données. Pour résoudre le problème lié à l'effet combinatoire du treillis, nous utiliserons les contraintes de fréquence, de validité et de profondeur pour sélectionner les concepts.

La méthode que nous présentons diffère des méthodes précédentes (comme KBANN [TS94]) par le fait que nous ne disposons pas de connaissances apriori. Elle diffère des méthodes dynamiques par la possibilité de justifier la présence de chaque noeud du réseau. La présence d'un nœud peut être justifiée par le fait que les exemples représentés par le nœud en question ont en commun un certain nombre d'attributs ; les connexions entre deux nœuds sont justifiées par la relation de précédence entre ces deux concepts. En plus notre approche est statique comme KBANN, puisque l'apprentissage du réseau est fait sans modification de sa structure.

Nous commencerons par la présentation générale du modèle et ensuite nous présenterons les deux versions de cette approche : la version CLANN (permettant de construire les RNA pour faire de la classification à deux classes et la version MCLANN pour traiter les données en plusieurs classes ($N_{cl} \geq 2$).

3.3.1 Présentation générale

CLANN définit l'architecture d'un réseau de neurones basée sur les treillis de Galois pour les problèmes à deux classes et MCLANN (Multi-class CLANN) est l'extension de CLANN à la résolution des problèmes à plusieurs classes. Ces approches consistent à construire le modèle du classifieur en trois étapes comme illustre la figure 3.5 : (1) la construction d'un demi-treillis à partir des données, (2) la transformation du demi-treillis en architecture initiale du réseau de neurones et (3) l'apprentissage du réseau obtenu. Nous allons nous intéresser aux deux premières étapes. L'apprentissage est fait avec l'algorithme de rétropropagation de l'erreur [RHW86a].

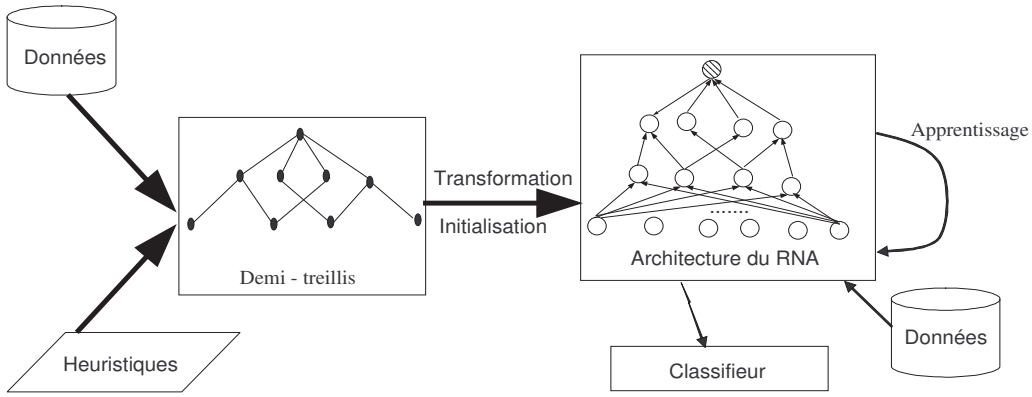

FIG. 3.5 – Architecture générale de CLANN et de MCLANN

3.3.2 Construction du demi-treillis

Nous utilisons dans notre approche une version modifiée de l'algorithme de Bordat [Bor86]. La modification apportée consiste à sélectionner au cours de la construction du treillis les concepts qui vérifient la contrainte appliquée. L'utilisation de l'algorithme de Bordat est justifiée par le fait qu'en plus du calcul des concepts par niveau [KO02], il construit également le graphe de Hasse entre ces concepts, et de ce fait il est facile d'y introduire des contraintes pour sélectionner les concepts, et réduire l'espace de recherche.

L'algorithme 10 présente le pseudo-code d'une version modifiée de l'algorithme de Bordat. Il construit à partir d'un contexte binaire C le demi-treillis de concepts L. Cet algorithme est utilisé pour construire le demi-treillis à partir duquel nous déduisons l'architecture du réseau.

Algorithme 10 Algorithme de construction du demi-treillis

Entrées: Un contexte binaire C et une contrainte pour sélectionner les concepts.

Sorties: Le demi-treillis de concepts (concepts extraits du contexte C) et le graphe représentant la relation d'ordre entre ces concepts

1: Initialiser la liste L des concepts à $(O, \{\})$ ($L \leftarrow \{(O, \{\})\}$) ;
2: **Répéter**
3: **Pour** chaque concept dans L dont les successeurs ne sont pas encore calculés **faire**
4: Calculer ses successeurs ;
5: Ajouter les successeurs à L s'ils vérifient **la contrainte spécifiée** ;
6: **fin pour**
7: **jusqu'à** ce qu'il n'y ait plus de concept vérifiant la contrainte spécifiée ;
8: Retourner L.

Proposition 1 *L'algorithme 10 est complet et correct.*

Preuve

Les ensembles A et O du contexte C sont finis et non vides ; l'intension (resp. extension) d'un concept est un sous-ensemble de A (resp. O) : $|L| \leq 2^{min(|A|,|O|)}$. Comme le calcul des successeurs est lié aux concepts existants dans la liste L alors au pire des cas le concept infimum (qui n'a pas de successeur) permettra de mettre fin à l'algorithme.

La complétude de cet algorithme est celle de l'algorithme de Bordat. En effet, comme l'algorithme de Bordat est complet, nous allons traiter cette complétude en fonction des contraintes utilisées.

A une étape i du calcul des successeurs :

- Si la contrainte est la fréquence, il est montré dans [AS94] que le successeur d'un concept non fréquent n'est pas fréquent et que le prédécesseur d'un concept fréquent est fréquent. L étant initialisé au supremum (fréquence = 100%) contient un concept fréquent à l'initialisation. Puisque le test de fréquence est effectué avant l'insertion d'un concept dans L, tous les concepts conservés sont fréquents.
- Si la contrainte est la profondeur k, les concepts de profondeur i sont les successeurs des concepts de profondeur $i-1$. Il est alors impossible de générer les concepts de profondeur $k+1$ car la génération s'arrête à la profondeur k.
- Si la contrainte est la cohérence : les concepts retenus dans L fréquents et quasi-consistents. Puisque L ne contient que les concepts fréquents lorsque la contrainte est la fréquence ; le test de la quasi-cohérence permet d'éliminer les concepts fréquents et non quasi-cohérents.

A la fin de l'algorithme 10, le demi-treillis contient tous les concepts du contexte C qui vérifient la contrainte spécifiée.

■

Cet algorithme modifié de Bordat est utilisé par les méthodes CLANN et MCLANN pour construire le demi-treillis. Ce demi-treillis est ensuite utilisé pour définir l'architecture initiale du RNA.

3.4 La méthode CLANN

Dans la construction du demi-treillis, CLANN choisit parmi les deux classes une à discriminer ; cette classe est traitée comme la classe positive (+). Le demi-treillis est construit en utilisant les objets de cette classe. Nous expliquons dans cette section comment le demi-treillis est transformée en RNA.

3.4.1 Passage du demi-treillis au RNA

Cette étape consiste à la transformation du demi-treillis en la structure du RNA équivalent. Les différentes couches (entrée, cachée et sortie) du RNA seront construites. Le passage du demi-treillis au réseau de neurones artificiel suit le processus suivant :

1. Le concept supremum du demi-treillis devient la sortie du réseau de neurones. En effet tous les objets du contexte appartiennent à l'extension de ce concept.
2. Tous les autres concepts forment les neurones de la couche cachée. Ces concepts sont formés de plusieurs objets et de plusieurs attributs. Ces concepts ne peuvent être ni en entrée car une entrée est réservée à un seul attribut, ni à la sortie car tous les objets ne seront pas représentés.
3. Une nouvelle couche est créée ayant autant de neurones que le nombre d'attributs décrivant les exemples ; cette nouvelle couche est la couche d'entrée du réseau de neurones. Dans cette couche, chaque neurone correspond à un attribut.
4. Les connexions entre neurones n_1 et n_2 correspondant aux concepts c_1 et c_2 sont celles du graphe de Hasse entre c_1 et c_2. Cette conservation apporte une justification aux liens de connexions établies entre neurones dans le RNA. La couche d'entrée est complètement connectée aux unités de la couche cachée n'ayant pas de successeurs. Cette connexion complète est faite afin de permettre aux attributs faiblement représentés (vérifiés par peu d'objets) et qui pourront n'appartenir à aucune intension des concepts sélectionnés de participer au calcul de résultat du RNA.

L'algorithme 11 est le pseudo-code qui présente le passage du demi-treillis au RNA correspondant. Il reçoit le graphe de Hasse d'un demi-treillis L et retourne le RNA NN correspondant.

Proposition 2 *L'algorithme 11 se termine toujours et transforme le demi-treillis de la structure L en réseau de neurones NN en temps fini. Le réseau de neurones NN a $|L| + N_{att}$ neurones.*

Algorithme 11 Passage du demi-treillis au réseau de neurones

Entrées: L Un demi-treillis construit à partir des exemples.

Sorties: NN Architecture du RNA obtenue du demi-treillis L.

1: **Pour** tout concept $c \in L$ **faire**
2: Ajouter un neurone correspondant à c au réseau NN avec pour prédécesseurs et successeurs les neurones correspondant respectivement à ses prédécesseurs et successeurs c dans L ;
3: Si l'ensemble des successeurs de c est vide alors marquer le neurone ajouté comme "dernier neurone caché" ;
4: **fin pour**
5: Créer une nouvelle couche de N_{att} neurones et connecter tous les neurones de cette couche aux neurones marqués "dernier neurone caché" dans NN ;
6: Initialiser les poids de connexions du réseau de neurones NN. Cette nouvelle couche est la couche d'entrée ;
7: Retourner NN.

Preuve

Nous allons vérifier qu'à tout instant d'exécution de cet algorithme, on a toujours $|NN| \leq |L| + N_{att}$ et à la fin de l'algorithme $|NN| = |L| + N_{att}$:

Initialement NN est vide donc $|NN| \leq |L| + N_{att}$.

Pendant l'exécution de la boucle Pour, remarquons que tout neurone inséré dans NN correspond à un et un seul concept c_i de L avec les mêmes successeurs et prédécesseurs. A la sortie de cette boucle une bijection est établie entre les concepts de L et les neurones de NN ; donc $|NN| = |L| \leq |L| + N_{att}$. A la fin de l'instruction 5, $|NN| = |L| + N_{att}$ avec l'ajout de N_{att} neurones de la couche d'entrée. Donc l'algorithme 11 transforme correctement le demi-treillis L en RNA NN.

■

3.4.2 Initialisation des poids de connexion et des seuils

Les poids de connexion sont définis de la manière suivante :

1. Entre les unités des couches internes, les poids de connexion sont initialisés à 1. Le but est de transmettre le signal des neurones correspondant aux sous - concepts aux neurones correspondant aux super - concepts sans l'inhiber.

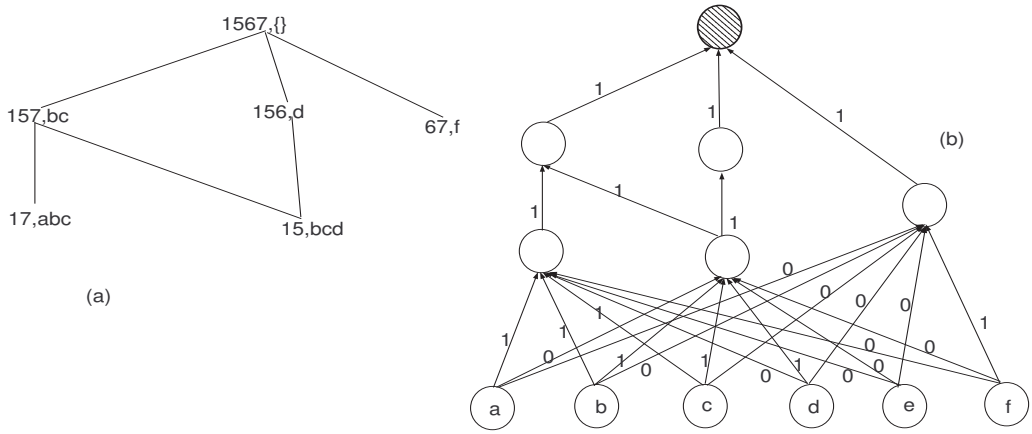

FIG. 3.6 – Demi-treillis de fréquence 50% et architecture initiale équivalente.

2. Entre la couche cachée et la sortie, les poids de connexion sont aussi initialisés à 1.
3. Et entre la couche d'entrée et la couche intermédiaire, les poids sont initialisés comme suit : 1 si l'attribut correspondant au neurone en entrée appartient à l'intension du concept correspondant au neurone auquel cette entrée est connectée et 0 dans la cas contraire. En effet lorsqu'un attribut n'appartient pas à l'intension d'un concept, initialement, il n'a aucun effet (ni d'excitation, ni d'inhibition) sur ce concept.

Illustration 8 *Les figures 3.6 (a) et 3.6 (b) présentent respectivement le demi-treillis construit avec la contrainte fréquence = 50% et le RNA équivalent. Le demi-treillis est construit à partir du tableau 3.1 en considérant les exemples de classe 'aérien' et 'aquatique' comme objets de classe + et les exemples de classe 'terrestre' comme objets de classe -. Une couche ayant six neurones y a été ajoutée. Cette couche est la couche d'entrée.*

Les figures 3.7 (a) et 3.7 (b) présentent respectivement un demi-treillis et l'architecture initiale du RNA équivalent. Ils sont construits dans les mêmes considérations que celles de la figure 3.6, mais la contrainte est profondeur = 2.

La méthode CLANN ainsi décrite permet de construire un RNA pour séparer les données en deux classes. On y introduit facilement les contraintes (fréquence, cohérence et profondeur). Mais CLANN est limitée aux problèmes à deux classes. Pour résoudre les problèmes à plusieurs classes, une solution serait de construire plusieurs modèles CLANN pour discriminer chaque classe. Cette solution non seulement sera lourde en nombre de RNA construits mais aussi on ne pourra profiter des relations qui existent entre les classes, puisque les exemples de classes différentes peuvent avoir des attributs en commun. Pour apporter une solution, CLANN a été étendue MCLANN.

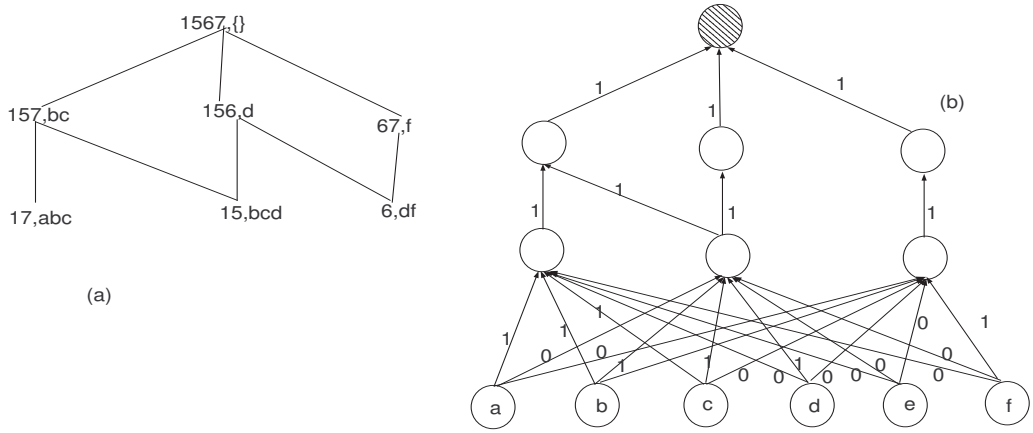

FIG. 3.7 – Demi-treillis de profondeur 2 et architecture initiale équivalente.

3.5 MCLANN : Multiclass Concept Lattice-based Artificial Neural Network

Le modèle précédent n'est donc pas approprié pour une utilisation généralisée. Afin d'utiliser les treillis de Galois dans un cadre plus général, nous avons proposé MCLANN (Multi-class Concept-based Artificial Neural Networks) [MNTT08, MNTT09]. Comme son prédécesseur CLANN, MCLANN définit une architecture du RNA en deux phases :

1. phase 1 : construction de demi-treillis à partir des données représentant le contexte.
2. phase 2 : transformation du demi-treillis en une architecture du réseau de neurones.

Vu les avantages que présente l'algorithme de Bordat, MCLANN utilise la version modifiée comme CLANN pour la construction du demi-treillis. MCLANN utilise aussi les contraintes pour élaguer le treillis et pour réduire son coût de construction. La construction du demi-treillis ne tient pas compte des classes des exemples.

3.5.1 Passage du demi-treillis au RNA

La différence fondamentale entre MCLANN et CLANN réside sur la nature du RNA obtenu à la fin du processus. En effet CLANN construit un RNA ayant une seule sortie alors que MCLANN construit un RNA ayant plusieurs sorties. Cette différence se trouve dans la façon de transformer le demi-treillis en RNA. Le supremum du demi-treillis est remplacé par une couche de N_{cl} neurones, N_{cl} étant le nombre de classes dans les exemples. Ainsi donc les relations entre les classes représentées dans les concepts sont conservées. La construction du demi-treillis est

faite par l'algorithme 10.

L'intégration de la couche de sortie est différente de celle de CLANN. Puisque cette couche est constituée de N_{cl} neurones ; chaque neurone de cette couche représente une classe. Les neurones correspondants aux concepts successeurs du suprenum vont être marqués pour être connectés aux N_{cl} neurones de la couche de sortie.

L'algorithme 12 présente le pseudo-code de l'algorithme de transformation du demi-treillis L en RNA NN par la méthode MCLANN.

Algorithme 12 Passage du demi-treillis au réseau de neurones de la méthode MCLANN

Entrées: L Un demi-treillis construit à partir des exemples.

Sorties: NN Architecture du RNA obtenue du demi-treillis L.

1: Supprimer le suppremum de L ;
2: **Pour** tout concept $c \in L$ **faire**
3: Si l'ensemble des prédécesseurs de c est vide alors ajouter un neurone et marquer le comme "dernier neurone caché" ;
4: Si l'ensemble des successeurs de c est vide alors ajouter un neurone et marquer le comme "premier neurone caché" ; connecter ce neurone aux neurones correspondant à ses prédécesseurs.
5: Sinon c devient un neurone et ajouter au réseau NN avec pour prédécesseurs et successeurs les neurones équivalents respectivement à ses prédécesseurs et successeurs dans L.
6: Finsi
7: **fin pour**
8: Créer une nouvelle couche de N_{att} neurones et connecter tous les neurones de cette couche aux neurones marqués "premier neurone caché" dans NN. Cette couche est la couche d'entrée.
9: Créer une nouvelle couche de N_{cl} neurones et connecter tous les neurones de cette couche aux neurones marqués "dernier neurone caché" dans NN. Cette nouvelle couche est la couche de sortie ;
10: Initialiser les poids de connexions du réseau de neurones NN ;
11: Retourner NN.

Proposition 3 *L'algorithme 12 se termine toujours et transforme le demi-treillis de la structure L en réseau de neurones NN en temps fini. Le réseau de neurones NN a $|L| + N_{att} + N_{cl} + 1$ neurones.*

Preuve

La preuve de cette proposition est la même que celle de l'algorithme 11.

■

Pour tous les neurones, le seuil d'activation est initialisé à zéro. Les poids de connexion sont initialisés de la manière suivante :

- Les poids de connexion entre deux neurones dérivés directement du demi-treillis sont initialisés à 1.
- Entre la couche d'entrée et la couche cachée, ils sont définis comme suit : 1 si l'attribut représenté par le neurone en entrée appartient à l'intension du concept associé au nœud de la couche cachée et 0 sinon.
- Entre la couche de sortie et la couche cachée, les poids sont également initialisés à 1.

3.5.2 Contraintes pour élaguer le demi - treillis

MCLANN utilise les mêmes contraintes que CLANN pour élaguer le treillis à l'exception de la cohérence du concept. Cette exception est dûe au fait qu'il est difficile de choisir parmi plusieurs classes une et la considérer comme positive (ou négative). Il se posera alors un problème de représentativité : soit le nombre des positifs est très inférieur au nombre des négatifs, soit l'inverse. Donc dans le cadre de MCLANN, les contraintes utilisées pour construire le demi-treillis sont la fréquence et la profondeur du treillis. Ces deux contraintes peuvent aussi être combinées dans cette tâche.

Illustration 9 *Les RNA initiaux équivalents aux demi-treillis des figures 3.2 et 3.4 sont présentés respectivement dans les figures 3.8 et 3.9.*

3.5.3 Complexité

Comme CLANN et MCLANN utilise une version modifiée de l'algorithme de Bordat [Bor86] pour construire le demi-treillis, la complexité de cet algorithme est en $O(2^{min(N_{obj},N_{att})})$. La construction du demi-treillis par CLANN et MCLANN est bornée par celle de l'algorithme de Bordat. Cette complexité au pire des cas est obtenue quand le demi-treillis contient tous les concepts ; la fréquence minimale spécifiée par l'utilisateur est 0% ou la profondeur spécifiée est supérieure à $min(N_{obj}, N_{att})$.

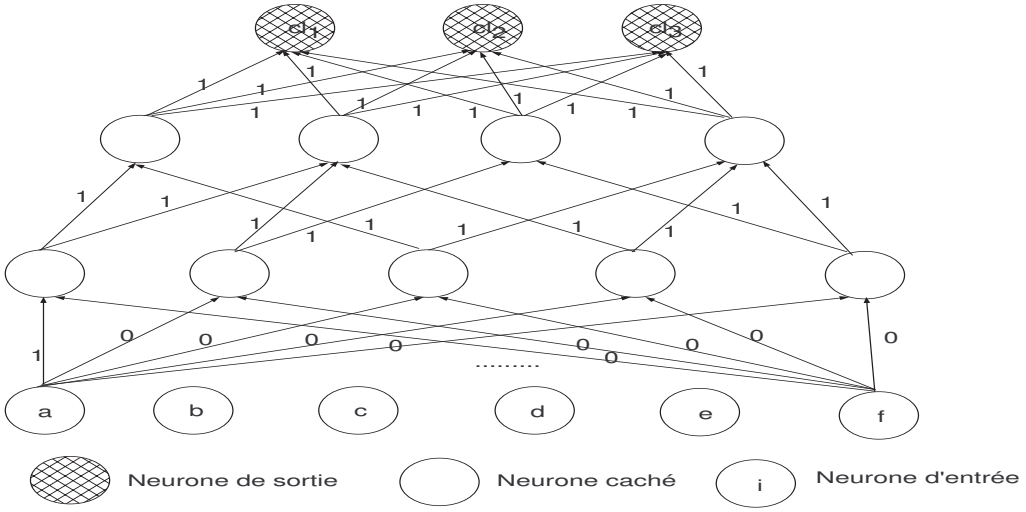

FIG. 3.8 – Architecture initiale du RNA équivalent au demi-treillis de la figure 3.2.

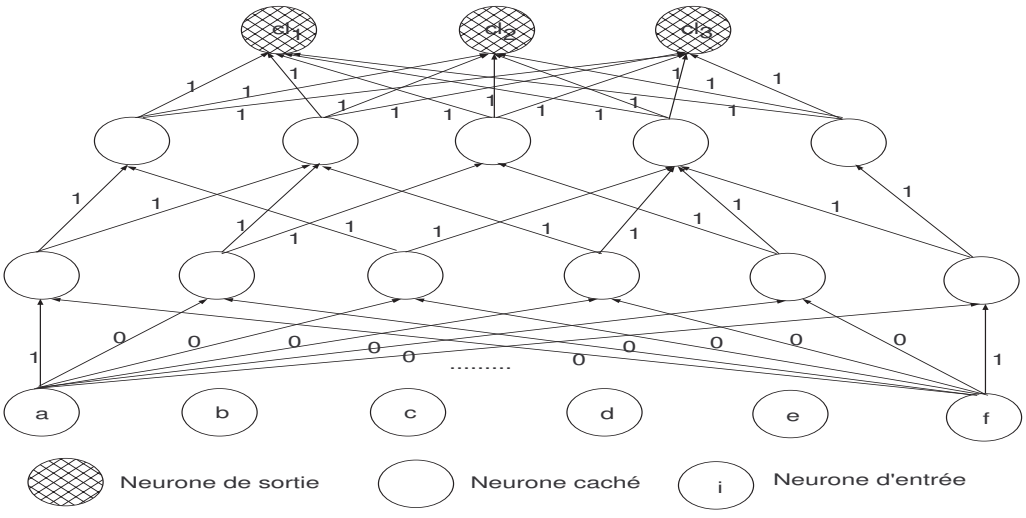

FIG. 3.9 – Architecture initiale du RNA équivalent au demi-treillis de la figure 3.4.

Algorithmes	Temps en construction	Temps en Classement
MCLANN	$2^{min(N_{obj},N_{att})}$	$2^{min(N_{obj},N_{att})}$
KBANN	N_{att}	N_{att}
Distal	N_{obj}^2	N_{obj}
MTiling	$HN_{att}N_{cl}^3N_{obj}N_{iter}$	HN_{cl}^2
MTower	$HN_{att}N_{cl}^2N_{obj}N_{iter}$	HN_{cl}
MUpstart	$2^HN_{cl}^2N_{att}N_{obj}N_{iter}$	2^HN_{cl}
P. Cascade	$2^HN_{cl}^2N_{att}N_{obj}N_{iter}$	2^HN_{cl}
Ad'hoc	$max(N_{cl}, N_{att})$	$max(N_{cl}, N_{att})$

TAB. 3.2 – Complexité en temps (de construction et de classement) et en espace mémoire des méthodes de construction de RNA.

La transformation du demi-treillis en RNA est linéaire en nombre de concepts sélectionnés ; elle est aussi bornée par la complexité de l'algorithme de Bordat. Au temps de construction du RNA, il faut ajouter le temps d'apprentissage.

Comparer à la complexité des autres algorithmes, CLANN et MCLANN ont théoriquement la complexité la plus élevée. La complexité théorique de MCLANN est exponentielle mais l'introduction des contraintes permet de réduire considérablement son temps d'exécution.

Le tableau 3.2 présente les complexités en temps de construction et en temps de classement des différents algorithmes de construction d'architecture des RNA.

Nous allons dans la section suivante expérimenter ces deux approches. Les données que nous utilisons suivent d'abord une étape de prétraitement avant l'exécution des algorithmes CLANN et MCLANN.

3.6 Expérimentations

Pour tester l'aspect pratique des approches CLANN et MCLANN, nous les avons expérimentés sur les données du base UCI [NHBM98]. Le tableau 3.3 présente les caractéristiques de ces données. Dans ce tableau, la colonne "Données" est le nom du jeu de données, #Taille le nombre total des exemples, #Classes le nombre de classes, #Nominaux le nombre initial d'attributs nominaux (ou symbliques) dans chaque exemple, #Binaires le nombre d'attributs obtenus après la binarisation. La binarisation a été faite par la procédure 'Filters.NominalToBinary' de

Données	#Taille	#Classes	#Nominaux	#Binaires
Balance	625	3	4	20
Car	1728	4	6	21
Chess	3 196	2	36	38
Lympho	148	4	19	51
Monks1	432	2	6	15
Monks2	432	2	6	15
Monks3	432	2	6	15
Spect	231	2	22	22
Soybean	384	19	35	151
Tic	958	2	9	27

TAB. 3.3 – Description des données expérimentales

la plate forme de fouille de données WEKA [WF05]. La diversité de ces données (de 24 à 3196 exemples ; de 2 à 19 classes) nous permet de confronter CLANN et MCLANN dans diverses situations. Les exemples de ces données n'ont pas de données manquantes.

Ces données sont décrites comme suit :

1. Balance Scale Weight (notée Balance) est un jeu de données obtenu des expériences en psychologie. Il est composé de 625 exemples décrits avec quatre attributs nominaux. Les exemples sont repartis en trois classes L, B et R suivant respectivement les pourcentages 46,08%, 7,84% et 46,08%.
2. Car Evaluation (notée Car) est un ensemble de données obtenues d'une évaluation des véhicules. Il contient 1728 exemples décrits par 6 attributs. Les données sont réparties dans les quatre classes proportionnellement de la manière suivante : la classe 'unacc' 70.023%, la classe 'acc' 22.22%, la classe 'good' 3.993% et la classe 'v-good' 3.762%.
3. Chess End-Game – King+Rook versus King+Pawn (Chess) est un ensemble de données obtenu du jeu d'échecs et composé de 3196 exemples décrits par 36 attributs booléens. Ils sont répartis en deux classes 'win' représentée par 52% des instances et de 'nowin' 48%.
4. Lymphography (Lympho), un ensemble de données obtenues de la médécine est composé de 148 exemples et 19 attributs. Ces attributs sont décrits par un ensemble de 2 à 10 valeurs discrètes. Ces exemples sont repartis en quatres classes 'normal find' 1.35%, 'metastases' 54,73%, 'malign lymph' 41,21% et 'fibrosis' 2,70% respectivement.
5. The Monk's Problems (Monks1, Monks2 et Monks3) est un ensemble de données décrivant les expériences en robotique. Il est composé de 432 exemples décrits par six attributs

à classer en deux classes. Suivant les propriétés de ces attributs, on distingue trois problèmes 'Monk's' : 'Monks1', 'Monks2' et 'Monks3'.

6. Large Soybean (Soybean) provient de la biologie. Cet ensemble contient 384 exemples décrits par 35 attributs. Ces données sont reparties en 19 classes. Ces exemples sont réparties entre ces classes, allant d'un exemple à 40 par classe.
7. SPECT heart (Spect), un ensemble de 267 exemples issus de la médécine. Ces données classées en deux groupes de 20,60% et de 79,40% respectivement sont décrites par des attributs binaires.
8. Tic-Tac-Toe Endgame (Tic), un ensemble de données décrivant le jeu tic-tac-toe. Il est composé de 958 exemples décrits par 9 attributs, chaque attribut appartenant à l'ensemble {x,o,b}. Les données sont réparties en deux classes 'positive' 65,30% et 'negative' 34,70%.

Pour expérimenter CLANN et MCLANN, le nombre d'itérations dans l'algorithme de rétropropagation est 500, c'est-à-dire que chaque exemple est présenté 500 fois au cours de l'apprentissage. Ce nombre est choisi par l'utilisateur, à notre connaissance, il n'existe pas de méthode permettant de l'estimer. La fonction d'activation des neurones est la fonction sigmoïde : $f(x) = \frac{1}{1+\exp(\lambda x)}$. La fonction sigmoïde est continue et il a été démontré [LS93] que les RNA multicouches ayant une fonction d'activation non polynomiale peuvent approximer avec une certaine précision toute fonction.

CLANN a été expérimentée sur les données à deux classes alors que MCLANN a testé sur toutes les données. L'évaluation du taux de généralisation s'est faite par validation croisée d'ordre 10 pour toutes les données. Nous présentons successivement les résultats de CLANN, ensuite les résultats de MCLANN et enfin une comparaison aux résultats des autres méthodes.

3.6.1 Résultats de CLANN

Les tableaux 3.4 et 3.5 présentent les taux de classification de CLANN sur les données Balance, Chess, Monks1, Monks2, Monks3, Spect et Tic-tac-toe (Tic). Dans ces tableaux CL20, CL25 et CL30 représentent respectivement les exécutions de CLANN avec les contraintes fréquence minimum 20, 25 et 30 ; CL1 et CL2 sont des versions avec les profondeurs respectives 1 et 2 ;

CLANN classe les données Chess, Monks1, Monks2 et Monks3 ; ses résultats sur les données Spect et Tic ne sont pas aussi bien. La particularité des données Chess, Monks1, Monks2 et Monks3 est la répartition presque équitable des données entre les deux classes.

Données	Fréquence			Profondeur	
	CL20	CL25	CL30	CL1	CL2
Chess	99,46±0,29	99,37±0,2	99,12±0,7	89,21±3,14	99,28±0,21
Monks1	99,25±0,80	99,97±1,98	99,37±0,51	100	100
Monks2	100	100	100	100	100
Monks3	100	100	100	100	100
Spect	66,08±2,47	66,52±0,96	61,74±3,03	60,51±1,1	63,94±3,78
Tic	69,47±1,67	70,42±1,13	69,15±1,1	69,58±1,2	69,05±0,86

TAB. 3.4 – Taux de généralisation (± écart-type) de CLANN sur les données du tableau 3.3 en utilisant comme contraintes la fréquence ou la profondeur.

Données	$\alpha = 20$		$\alpha = 25$		$\alpha = 30$	
	$\beta = 40$	$\beta = 50$	$\beta = 40$	$\beta = 50$	$\beta = 40$	$\beta = 50$
Chess	99,15±0,19	99,24±0,04	99,22±0,07	99,28±0,07	98,38±0,04	97,75±0,3
Monks1	100	99,97±0,98	98,37±1,1	100	99,21±0,62	100
Monks2	100	100	100	100	100	100
Monks3	100	100	100	100	100	100
Spect	69,57±5,5	66,95±1,92	64,78±5,63	63,48±2,2	68,70±3,85	65,65±0,14
Tic	66,47±0,3	68,84±1,8	74,31±0,87	69,05±0,53	68,32±1,36	67,68±0,10

TAB. 3.5 – Taux de généralisation (± écart-type) de CLANN sur les données du tableau 3.3 en utilisant comme contraintes la cohérence.

Données	Fréquence			Profondeur	
	MCL20	MCL25	MCL30	MCL1	MCL2
Balance	99,51±0,15	-	-	99.25±0,25	98,55±0,05
Car	99,82±0,05	99,94±0,01	99,76±0,07	99,65±0,80	99,24±0,23
Chess	99,15±0,17	93,19±1,51	98,81±0,70	99,21±0,14	77,21±2,21
Lympho	97,85±1,58	97,85±1,58	97,86±0,68	97,85±0,86	97,85±0,88
Monks1	81,62±0,80	96,97±1,98	98,37±0,51	100	100
Monks2	71,86±1,40	95,11±1,54	100	100	100
Monks3	100	100	100	100	100
Spect	55,22±3,71	60,87±0,02	66,09±3,03	66,51±4,54	70,43±5,78
Soybean	+	+	+	99,37±0,20	97,79±1,22
Tic	58,00±8,6	99,37±0,13	99,15±0,27	99,37±0,2	69,89±0,86

TAB. 3.6 – Taux de généralisation sur les données du tableau 3.3 en utilisant comme contraintes la fréquence ou la profondeur.

3.6.2 Résultats de MCLANN

Les taux de généralisation sont présentés dans les tableaux 3.6 et 3.7. Le symbole '-' signifie que le contexte est de très faible densité et dans la construction du demi-treillis, aucun concept n'a vérifié la contrainte ; par conséquent MCLANN n'a pas construit de RNA. Le symbole '+' signifie que l'expérience a été arrêtée après 12 heures alors que les 10 itérations n'étaient pas achevées.

Le tableau 3.6 présente les taux de classification des versions de MCLANN. Les versions de MCLANN présentées sont : MCL20, MCL25 et MCL30 pour MCLANN dont le demi-treillis a été construit avec pour supports minimas respectifs 20, 25 et 30. MCL1 et MCL2 représentent les taux de généralisation obtenus en utilisant comme contraintes la hauteur ; les valeurs sont respectivement 1 et 2. Le tableau 3.7 présente les résultats obtenus en combinant les deux contraintes hauteur et fréquence ; un demi-treillis doit avoir au maximum la hauteur spécifiée et les concepts sélectionnés doivent avoir au moins la fréquence spécifiée.

Des tableaux 3.6 et 3.7, on constate que dans la majorité des cas (chess, monks1, Monks2, Spect et Tic) les performances du RNA deviennent mauvaises quand le demi treillis construit a plus de concepts. Les résultats de MCLANN (MCL1) avec pour contrainte la profondeur 1 sont en général meilleurs que ceux des autres modèles de MCLANN. En plus, les temps de classification de MCL1 est plus faible comparé aux autres ; un autre avantage est que la densité du contexte n'influence pas sur la présence des concepts dans le demi-treillis comme c'est le cas de MCL25 et MCL30 pour 'Balance'. Nous allons dans la suite comparer les résultats de

Données	profondeur=1			Profondeur=2		
	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$
	MCL1-20	MCL1-25	MCL1-30	MCL2-20	MCL2-25	MCL2-30
Balance	99,35±0,20	-	-	99,51±0,15	-	-
Car	99,88±0,03	99,82±0,05	99,76±0,47	99,94±0,01	99,94±0,16	99,76±0,07
Chess	99,37±00	99,37±0,09	77,93±0,23	77,37±3,68	77,93±1,05	76,49±0,09
Lympho	86,42±2,48	98,57±0,45	97,86±0,68	97,85±1,58	97,86±0,68	96,68±0,68
Monks1	100	100	100	83,25±2,80	90,23±7,94	98,37±0,51
Monks2	73,49±0,30	94,19±1,83	100	73,48±0,29	94,18±1,83	100
Monks3	100	100	100	100	100	100
Spect	65,22±4,22	63,04±2,06	71,74±2,06	68,70±0,27	64,78±0,13	65,66±3,98
Soybean	99,71±0,3	99,26±0,23	99,26±0,32	99,21±0,25	98,38±0,41	98,38±0,51
Tic	99,26±0,24	99,16±0,27	99,15±0,22	62,42±2,43	99,26±0,10	99,26±0,23

TAB. 3.7 – Taux de généralisation obtenus sur les données du tableau 3.3 en combinant les contraintes fréquence (α) et profondeur.

MCL-1 aux résultats des autres techniques de construction de RNA.

les tableaux 3.8 et 3.9 présentent le nombre total de neurones que possèdent les différents RNA construits par MCLANN. Avec la contrainte *profondeur=1*, MCLANN construit un RNA ayant peu de neurones et dans un cas général avec la contrainte *fréquence=20%* le RNA a beaucoup de neurones. La densité du contexte a une influence sur le demi-treillis ; avec la contrainte fréquence, il peut arriver qu'aucun concept ne la vérifie et MCLANN s'arrête ; c'est le cas avec les données 'Balance'.

Les tableaux 3.10 et 3.11 présentent le temps (en secondes) d'exécution de MCLANN au cours des expériences. Ce temps est fortement lié à la capacité de l'algorithme de rétropropagation à converger.

MCL1 construit un RNA ayant une couche cachée comme Distal [YPH99], mais les deux méthodes sont différentes : avec Distal chaque neurone de la couche cachée représente un sous-ensemble d'exemples d'apprentissage et l'intersection de ces sous-ensembles est vide ; dans le cadre MCL1, ces neurones représentent les concepts formels et les intersections des extensions ne sont pas vides. L'activation des neurones dans Distal se calcule comme une distance entre ses poids de connexion et l'exemple en entrée. Dans MCL1 cette activation se calcule comme une combinaison linéaire du vecteur représenté par l'exemple en entrée et les coefficients sont les poids de connexion. Enfin les poids de connexion dans Distal sont les valeurs des attributs de l'exemple qui regroupe mieux les autres exemples alors que MCL1, ces poids sont obtenus par apprentissage. Nous allons dans la suite comparer les résultats de MCL1 aux résultats des

Données	Fréquence			Profondeur	
	MCL20	MCL25	MCL30	MCL1	MCL2
Balance	34,7±1,49	-	-	43,0	193
Car	46	40,6±0,52	34	46	229
Chess	234,3±2,63	159,6±1,73	111,4±1,73	66	256,7±0,67
Lympho	149,6±28,6	179,7±6,27	138,5±8,99	227,9±18,1	229,7±1,4
Monks1	32	29,7±0,48	26,9±0,31	31	121
Monks2	32	30	27	31	121
Monks3	32	28,7±3,09	27	31	121
Spect	151,4±6,9	92,5±4,79	68,5±2,84	43	194,1±8,77
Soybean	+	+	+	234,7±0,8	567,2±19,22
Tic	57,7±2,48	50	46	55	378

TAB. 3.8 – Nombre (moyen) total de neurones que possède chaque RNA construit en utilisant comme contraintes la fréquence (α) et la profondeur pour classer les données du tableau 3.3.

Données	profondeur=1			Profondeur=2		
	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$
	MCL1-20	MCL1-25	MCL1-30	MCL2-20	MCL2-25	MCL2-30
Balance	34,3±1,41	-	-	33,3±0,67	-	-
Car	46	40,7±0,67	34	46	40,1±1,20	34
Chess	81,2±2,30	53	53	13	103	
Lympho	28,5±0,53	70,1±1,5	68,7±1,5	131,9±5,1	111,4±5,66	99,6±4,38
Monks1	31	29,3±0,48	27	32	29,8±0,78	27
Monks2	32	29,8±0,78	27	32	29,8±0,78	27
Monks3	31	29,5±0,53	29	31	29,5±0,97	27
Spect	42,5±0,53	42	38,2±0,42	103,2±6,37	72,7±1,95	59,2±1,75
Soybean	207,7±0,82	203,8±0,63	189,9±0,74	253±2	329,2±5,76	291,1±4,01
Tic	54	50	46	55,4±0,84	50	46

TAB. 3.9 – Nombre (moyen) total de neurones que possède chaque RNA construit pour classer les données du tableau 3.3 en combinant les contraintes fréquence (α) et la profondeur.

Données	Fréquence			Profondeur	
	MCL20	MCL25	MCL30	MCL1	MCL2
Balance	15,3±0,67	-	-	15,0±0,5	41,9±12,96
Car	50,3±0,48	43,3±0,5	26,2±0,42	58,56±3,6	436,33±230
Chess	1653,8±41,7	787,8±34,4	390±15,05	172±0,91	308,9±35,85
Lympho	58,1±14,06	23,3±2,4	12,8±2,0	14,4±19,9	24,4±25,65
Monks1	3,5±0,53	4,8±1,35	5,2±0,42	14,7±17,69	52,4±4,57
Monks2	3,8±0,42	4,3±1,59	5,01±0,1	8,7±0,48	52,9±2,8
Monks3	3,5±0,53	4,6±1,58	5,2±0,42	9,6±0,7	51,1±0,31
Spect	151,4±6,9	92,5±4,79	68,5±2,84	43	194,1±8,77
Soybean	+	+	+	345,4±4,08	564,19±810,9
Tic	24,4±10,54	30,9±0,32	29,7±1,25	39,1±1,79	169,2±182,7

TAB. 3.10 – Temps moyen (en secondes) d'exécution des différents algorithmes pour classer les données du tableau 3.3.

Données	profondeur=1			Profondeur=2		
	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$
	MCL1-20	MCL1-25	MCL1-30	MCL2-20	MCL2-25	MCL2-30
Balance	9,5±0,97	-	-	9,7±0,47	-	-
Car	45,7±1,58	40±1,7	24±4	50±0,47	37,9±2,45	24,5±4,74
Chess	92,8±4,66	99±0,67	52,5±3,44	87,6±6,93	68,3±5,75	
Lympho	2,2±0,41	6,2±0,63	59±0,74	12,3±10,17	16,2±3445	5,4±5,48
Monks1	6,1±0,1	5,8±0,42	5,4±0,51	1,2±0,42	4,5±2,01	5,1±0,32
Monks2	4,3±5	3,2±0,42	4,8±1,40	5,2±0,42	4,5±2,01	5,1±0,32
Monks3	6,5±0,71	6±0,47	5,8±0,42	6,7±0,48	4,6±1,73	5,2±0,42
Spect	5,7±0,48	5,6±0,51	4,7±0,48	23,2±6,44	15,3±20,75	6,1±3,18
Soybean	192,3±3,4	164,5±6,0	142,6±4,03	413,87±11,0	270,4±314,01	222,3±216,6
Tic	34,9±0,32	28,7±0,95	24,3±048	8,8±11,68	30,9±0,32	54,2±0,42

TAB. 3.11 – Temps d'exécution (en secondes) des modèles construits en combinant les contraintes pour classer les données du tableau 3.3 en combinant les contraintes fréquence (α) et profondeur.

Données	CL1	MCL1	KBANN
Monks1	99,25±0,80	100	96,75±1,2
Monks2	100	100	99,05±0,9
Monks3	100	100	100

TAB. 3.12 – Comparaison des résultats de CL1, MCL1 aux résultats de KBANN sur les données Monks.

autres techniques de construction de RNA.

Résultats de KBANN sur les données Monk's

Le tableau 3.12 présente une comparaison des taux de généralisation des RNA construits avec la méthode CLANN, MCLANN et KBANN sur les données Monks1, Monks2 et Monks3. Les règles utilisées pour construire le RNA de KBANN ont été prises dans [Tea91, Ned98]. Nous ne disposons pas des règles pour les autres données. Les résultats de KBANN sont comparables à ceux de MCLANN et CLANN construits avec la contrainte profondeur 1.

3.6.3 Comparaison expérimentale

Cette comparaison est faite avec les méthodes MTiling, MTower, MUpstart, Perceptron-Cascade (Cascade), MPyramid et Distal. Comme dans le chapitre précédent, pour ces méthodes, l'algorithme d'apprentissage choisi est la variante 'Pocket Perceptron with rachet modification' [Gal90, PYH99] de l'algorithme d'apprentissage du perceptron avec 500 itérations. Nous avons choisi 10 comme nombre maximum de couches cachées. Pour l'expérimentation de Distal la distance euclidienne a été calculée entre les exemples ; nous ne disposons d'aucune connaissance pour le choix d'une mesure de distance appropriée comme indiquent les auteurs de cet algorithme.

Le tableau 3.13 présente les résultats des différentes méthodes dynamiques sur les données du tableau 3.3. Le symbole '*' signifie que l'algorithme a atteint le nombre maximum de couches cachées sans converger. MCL1 est pratiquement meilleur que tous les autres algorithmes sur tous les jeux de données à l'exception de 'Spect' où MTower est meilleur. Distal présente en général le plus mauvais résultat. Ceci est peut-être lié au fait qu'il assure un taux d'apprentissage de 100% sur l'ensemble d'apprentissage et qu'il regroupe les exemples en groupes disjoints appartenant exclusivement à une classe. Les exemples de test de ces données

Données	MCL1	MTiling	MTower	MUpstart	Cascade	Mpyramid	Distal
Balance	99.25	98,39	100	98,41	96,77	100	58,49
Chess	99,21	95,92	96,87	98,43	94,98	97,49	60,59
Lympho	97,85	92,86	92,63	96,84	92,86	85,71	67,41
Monks1	100	88,37	100	100	100	100	93,23
Monks2	100	72,09	62,79	65,12	65,81	63,49	90,98
Monks3	100	100	100	100	100	100	93,67
Tic	99,37	98,95	100	100	95,79	97,89	53,24
Moyenne	**99,38**	**92,37**	**93,18**	**94,11**	**92,31**	**92,08**	**74,95**
Spect	66,51	*	69,57	47,83	65,22	60,87	55,91
Car	99,65	*	95,93	*	*	91,86	77,28
Soybean	99,37	*	*	*	*	*	*
Moyenne générale	**96,42**	**-**	**-**	**-**	**-**	**-**	**-**

TAB. 3.13 – Comparaison des taux de généralisation obtenus sur les données du tableau 3.3.

d'expérimentations devront être plus proches des groupes appartenant à des classes différentes que la leur. les échecs de MTiling, de MUpstart sur les données 'car' et 'Spect' sont dûs à la valeur désirée du taux d'apprentissage à satisfaire au cours de l'apprentissage. En fait en réduisant ce taux à 70%, MTiling classe les données 'Spect' à 56,52% ; et en le réduisant à 85%, MTiling classe les données 'Car' à 88,05%, MUpstart à 91,28% et Cascade à 90,12% Pour les données 'Soybean', les RNA construits par les algorithmes MTower,Cascade et MPyramid atteignent le nombre maximum de couches cachées sans converger. MCL1 obtient un bon taux de classification sur les données 'Soybean' alors que les autres échouent probablement à cause du nombre de classes. Un autre avantage de MCLANN par rapport aux autres est que ces méthodes construisent des réseaux opaques aux utilisateurs alors tous les neurones de MCLANN peuvent être interprètés comme un ensemble d'attributs (intension du concept) partagés par un ensemble d'objets, et une connexion est un lien d'inclusion entre ces ensembles d'attributs.

Le tableau 3.14 compare les victoires entre les méthodes du tableau 3.13. Il s'agit du nombre de cas où le taux de classification de la méthode de la ligne i est supérieur au taux de généralisation de la méthode de la colonne j. Distal a la moyenne la plus faible alors que MCLANN a la plus élevée. Mtower est la meilleure méthode dynamique sur les données du tableau 3.3.

Le tableau 3.15 présente les temps moyens d'exécution (obtenu en validation croisée) en secondes des différentes méthodes sur les données du tableau 3.3. Pour la méthode MCL1, ce temps représente le temps de construction du demi-treillis et le temps d'apprentissage. Pour les méthodes MTiling, MTower et MUpstart, Perceptron-Cascade et MPyramid, il est le temps obtenu quand le processus de construction dynamique du RNA s'arrête. Et pour Distal, il est

i \j	MCL1	MTiling	MTower	MUpstart	Cascade	MPyramid	Distal
MCL1	*	9	5	7	8	7	10
MTiling	0	*	2	1	4	3	5
MTower	3	6	*	3	6	4	8
MUpstart	1	6	3	*	4	4	7
Cascade	0	2	2	2	*	3	7
MPyramid	1	6	2	4	5	*	8
Distal	0	4	1	7	2	1	*

TAB. 3.14 – Table des victoires : Comparaison des nombres de victoires entre la méthode i (ligne) et la méthode j (colonne).

Données	MCL1	MTiling	MTower	MUpstart	Cascade	MPyramid	Distal
Balance	15,0	1,06	0,76	1,01	*0,42*	1,01	**15,3**
Chess	172,8	8,96	9,79	7,27	*4,33*	6,85	**862,42**
Lympho	14,4	**16,8**	7,2	3,4	3,1	6,9	*2,6*
Monks1	14,7	**68,46**	*0,42*	1,33	1,76	1,32	5,05
Monks2	8,7	**965,77**	74,29	106,56	65,99	64,31	*6,01*
Monks3	**9,6**	*0,01*	*0,01*	*0,01*	*0,01*	*0,01*	5,01
Tic	39,1	1,47	1,46	1,89	*1,00*	1,14	**55,25**
Spect	5,9	*	1,48	**7,59**	3,63	1,14	*0*
Car	58,56	*	*23.53*	*	*	21,85	**314,02**
Soybean	**345,4**	*	*147,18*	*	*	147,18	314,6

TAB. 3.15 – Comparaison des temps d'exécution (en secondes) des méthodes neuronales. Pour chaque jeu de données, le temps maximal est écrit en gras et le minimal en italique.

la somme du temps utilisé pour calculer les distances et du temps de construction de la couche cachée du RNA. Distal a un temps d'exécution faible quand la taille des données est également faible, mais son temps d'exécution devient très élevé lorsque le nombre d'exemples devient plus grand (cas de 'Chess'). MTower est meilleur en temps d'exécution que MCL1 et tous les autres. Mais MCL1 a convergé plus vite que MTower sur les données 'Car' et 'Monks2'. Pour toutes ces méthodes, le temps de test du RNA construit est pratiquement nul.

Le tableau 3.16 compare le nombre de neurones des RNA construits par chaque méthode. Ce nombre est le nombre des neurones de toutes les couches (entrée, sortie et cachée). Les RNA construits par Distal et MCL1 ont dans la majorité des cas plus de neurones que les RNA construits par les autres algorithmes. Le nombre de neurones cachés dans les RNA construit par Distal est fonction du nombre de partitions des exemples. MCL1 construit un RNA dont la couche cachée a autant de neurones que de concepts successeurs du supremum du demi-treillis.

Données	MCL1	MTiling	MTower	MUpstart	Cascade	MPyramid	Distal
Balance	43	29	26	*23*	*23*	*23*	**56**
Chess	66	50	*42*	52	54	*42*	**143**
Lympho	73	71	*67*	**96**	83	*67*	83
Monks1	26,9	64	19	*18*	20	19	**93**
Monks2	27	*17*	19	65	24	23	**97**
Monks3	27	*16*	*16*	*16*	*16*	*16*	**81**
Tic	**55,1**	37	*29*	33	29	34	55
Spect	**43,5**	*	*26*	35	29	27	41
Car	46,14	*	*37*	*	*	*37*	**201**
Soybean	234,7	*	**341**	*	*	**341**	*23*

TAB. 3.16 – Comparaison du nombre de neurones. Pour chaque jeu de données, le maximum est écrit en gras et le minimum en italique.

Le nombre de successeurs du supremum est borné par le nombre N_{att} d'attributs. MTower construit les RNA ayant moins de neurones que les autres méthodes ; en effet les différentes couches cachées ont N_{cl} neurones alors que les méthodes MTiling et MUpstart ont plus N_{cl} neurones dans les couches cachées (nombre de neurones auxillaires).

Le tableau 3.17 compare les résultats de MCLANN à ceux des autres méthodes de classification : MLP, IB1, PART, J48, BayesNet,et SMO. Sur les données binarisées du tableau 3.3, les méthodes neuronales (MCL1 et MLP) sont meilleures que les autres. La ligne 'Moyenne' est la moyenne des taux de généralisation obtenue sur les données de la première partie du tableau et la ligne 'Moyenne générale' est la moyenne des taux de généralisation de l'ensemble des données. Sur ces donnnés (tableau 3.3), la méthode bayésienne BayesNet a les mauvais résultats. Les méthodes arborescentes (PART et J48) et les SVM ont des résultats moyens, mais les méthodes PART et J48 ont pour avantage qu'un ensemble de règles décrivant le processus de classification peut être extrait de l'arbre de décision.

Les algorithmes de recherche d'architecture de RNA diffèrent les uns des autres par plusieurs facteurs : la taille du RNA construit que nous pouvons évaluer en fonction du type de données traitées, du nombre de neurones, de la durée de construction du RNA, de l'espace mémoire nécessaire pour son exécution, ...

Données	MCL1	MLP	IB1	PART	J48	bayesNet	SMO
Balance	99,25	97,6	66,72	83,52	76,96	77,60	91,04
Car	99,65	99,53	77,25	96,93	97,87	85,01	93,75
Chess	99,21	99,53	90,27	99,09	99,43	87,89	95,65
Lympho	97,85	81,75	80,40	79,72	74,32	82,43	83,10
Monks1	100	100	100	97,45	98,61	75	75
Monks2	100	100	53,56	95,60	94,67	59,95	67,13
Monks3	100	100	100	100	100	97,22	100
Moyenne	**99,38**	**96,89**	**81,89**	**92,76**	**91,03**	**78,36**	**87,18**
Spect	66,51	70,04	66,67	64,41	70,41	69,29	70,78
Soybean	99,37	93,41	88,85	91,06	91,94	90,62	93,99
Tic	99,37	96,86	81,63	93,95	93,21	68,48	98,33
Moyenne générale	**96,42**	**94,60**	**80,65**	**90,41**	**89,86**	**79,29**	**86,90**

TAB. 3.17 – Comparaison des taux de généralisation obtenus sur les données du tableau 3.3

3.6.4 Discussion

Types de données

Les algorithmes MTiling, MTower, MUpstart etc. traitent les données numériques sans prétraitement et KBANN traite les données symboliques. CLANN et MCLANN ne traitent que les données binaires, leurs utilisations nécessitent au préalable une binarisation.

Structure du réseau final et interprétation

La complexité de classement d'un vecteur n'appartenant pas à l'ensemble des exemples utilisés pour l'apprentissage est essentiellement fonction de la structure du réseau. Cette classification coûte plus en temps pour un réseau qui a plus de couches. Les algorithmes cités ci-dessus construisent les réseaux multicouches, la différence est surtout perceptible à l'organisation de la couche cachée. Pour un problème de classification, le réseau de neurones nécessaire à la résolution peut avoir une seule couche cachée (où chaque unité sera apprise pour classer un seul exemple). Les résultats de MCLANN avec pour contrainte $profondeur = 1$ tendent à corroborer avec les résultats théoriques [Cyb89, LS93] qui démontrent qu'un RNA à une couche cachée est capable d'approximer n'importe quelle fonction.

Le RNA construit par la méthode MCLANN est interprétable, comme chaque nœud correspond à un concept formel, il est possible d'exploiter la structure du demi-treillis et d'y extraire

les règles de décision compréhensibles.

L'espace mémoire nécessaire

Asymptotiquement ces algorithmes ont le même comportement. Cependant l'algorithme Distal nécessite plus d'espace mémoire pour les opérations de distances. Les autres méthodes MCLANN, MUpstart, MTiling,... n'ayant généralement pas d'opérations supplémentaires coûteuses sur les données en mémoire, stockent plutôt une grande structure du réseau. CLANN et MCLANN ont une complexité mémoire exponentielle, mais l'introduction des contraintes a réduit l'espace mémoire utilisé pour son exécution et cet espace est comparable à celui occupé par les autres méthodes neuronales pour la classification des données.

Algorithme d'apprentissage

La complexité des algorithmes utilisés pour apprendre les unités neuronales ajoutées par des méthodes dynamiques est présenté dans [PYH99] et est en $O(N_{iter}N_{obj}N_{att}N_{cl})$. Ces algorithmes ne peuvent être utilisés que pour l'apprentissage des poids de connexion des neurones ajoutés au réseau dans le cadre des algorithmes MTiling, MTower, MUpstart, Cascade et MPyramid. Ils sont pas adaptés au réseau construit par les méthodes KBANN ou MCLANN car le RNA construit par ces méthodes est composé de couches cachées et l'apprentissage est fait après la construction. Le RNA construit par les méthodes KBANN ou MCLANN est appris par l'algorithme de rétropropagation de l'erreur, qui est plus approprié aux réseaux ayant des couches cachées.

La capacité de généralisation

Les taux de généralisation obtenus des expériences par validation croisée sont comparables. La méthode Distal assure théoriquement un taux de validation de 100% si la méthode de validation est la resubstitution ; ce qui l'expose au problème de surapprentissage et peut conduire à des mauvaises performances (comme sur les données Spect, Chess et Tic).

Les taux de généralisation des autres algorithmes dépendent de la représentativité des données d'apprentissage. Même si MCLANN présente de meilleurs résultats en généralisation, il n'est pas supérieur à toutes les autres méthodes quelque soit les données, ce qui vient corroborer le théorème du 'No Free Lunch' [WM97]. Il ressort de notre expérimentation que la méthode

Données	MLP	IB1	PART	J48	bayesNet	SMO
Balance	97,92	66,72	77,28	64,48	91,36	90,24
Car	99,54	77,25	95,76	92,36	85,70	93,75
Chess	99,37	90,27	99,09	99,43	87,89	95,65
Lympho	81,76	75,68	82,43	76,35	85,13	83,10
Monks1	100	79,86	100	100	97,22	100
Monks2	100	55,32	71,53	67,13	66,44	67,13
Monks3	100	72,45	100	96,53	75	75
Moyenne	**96,94**	**73,94**	**89,44**	**85,18**	**84,11**	**86,41**
Spect	70,03	66,67	64,41	70,41	69,29	70,79
Soybean	92,53	91,51	92,53	93,41	90,63	95,17
Tic	96,89	81,63	94,47	85,07	69,42	98,33
Moyenne générale	**84,48**	**79,94**	**83,80**	**82,96**	**76,45**	**88,10**

TAB. 3.18 – Taux de généralisation obtenus sur les données du tableau 3.3 sans prétraitement

MCLANN a de très bonnes performances en généralisation, et se prête donc comme une alternative efficace à la recherche d'architecture de RNA.

Effets de la binarisation

Le tableau 3.18 présente les taux de classification des implémentations des méthodes MLP, IB1, PART, J48, bayesNet et SMO sur les données sans prétraitement. Le but est de vérifier les effets de la binarisation sur les taux de classification de ces méthodes. En général, les moyennes obtenues sur les données binarisées sont meilleures que celles obtenues des données non binarisées. De même les résultats des méthodes dynamiques (tableau 3.13) sur les données 'Lympho' et 'Balance' binarisées sont meilleurs que ceux obtenus sur ces données non binarisées (tableau 2.6).

3.7 Conclusion

Dans ce chapitre, nous avons présenté des approches basées sur les treillis de Galois pour la construction des réseaux de neurones interprétables : la méthode CLANN conçue pour les problèmes à deux classes et la méthode MCLANN pour la résolution des problèmes à plusieurs classes. Les deux approches contruisent un demi-treillis à partir des données, puis transforment ce demi-treillis et architecture du RNA, et enfin elles utilisent l'algorithme de rétropropagation

de l'erreur pour rechercher les poids de connexion du réseau.

Une expérimentation de ces approches a montré leurs comportements sur quelques problèmes de classification. L'évaluation des deux systèmes (CLANN et MCLANN) menée sur des données prises dans la base UCI [NHBM98] a montré des résultats acceptables et a montré son efficacité en classification supervisée après comparaison avec des méthodes standards sur les données du tableau 3.3. En comparant les taux de classification à ceux des autres techniques (neuronales ou pas) de classification, MCLANN se classe parmi les premiers sur les données utilisées. Dans cette comparaison, les RNA construits par MCLANN ont plus de neurones dans la majorité des cas que les RNA construits par d'autres approches. Théoriquement, l'approche CLANN et MCLANN ont une complexité exponentielle, mais l'introduction des heuristiques permet de réduire considérablement le temps d'éxecution sans perdre son efficacité dans la classification.

Outre le bon comportement dans les problèmes de classification, CLANN et MCLANN construisent des réseaux interprétables (une interprétation peut être donnée à chaque neurone et à chaque connexion). Dans le prochain chapitre nous allons nous focaliser à l'extraction des règles des réseaux de neurones.

Chapitre 4

Les règles dans les réseaux de neurones

Sommaire

4.1 Introduction

Dans les chapitres précédents, nous avons montré comment construire les réseaux de neurones et les entraîner. Dans ce chapitre, nous allons nous intéresser à l'extraction des connaissances contenues dans le RNA après l'apprentissage, exprimées sous la forme des règles.

Certes les résultats des réseaux de neurones en classification sont très appréciables ; mais ceux-ci n'ont pas toujours les faveurs des utilisateurs des domaines comme la médécine et le nucléaire [ADT95]. L'une des raisons les plus crédibles est l'incapacité d'expliquer comment les décisions fournies en sortie du réseau sont prises [DSZ04] :

> *"La prédiction par des modèles opaques est irraisonnable parce qu'il n'existe aucun moyen de les contrôler et de les tester sur des domaines proches de celui du problème à résoudre"*.

Une solution à ce problème est d'associer au RNA un module permettant d'extraire des règles pour expliquer son fonctionnement car un ensemble de règles est plus compréhensible qu'une collection de poids de connexion et de liens. Ces règles permettent aussi d'accroître la confiance au modèle et de découvrir certaines corrélations entre les entrées [ADT95]. Ce processus suppose que le neurone a pour valeur en sortie VRAI ou 1 (le neurone est actif) et FAUX ou 0 (le neurone est inactif). L'extraction des règles se focalise plus à la recherche des prémisses de ces règles. Les étiquettes des différents neurones sont utilisées comme différentes conclusions. Pour cela, il est important que le RNA soit compréhensible et interprétable. De ce fait il est alors très difficile de trouver des interprétations aux RNA construits par des méthodes dynamiques, évolutives ou ad'hoc. Nous utiliserons l'expression 'extraction des règles' dans ce chapitre, mais en fait nous ne ferons que l'extraction des prémisses ; car la conclusion est connue comme étant l'étiquette du neurone pour lequel les règles sont extraites.

Les articles [ADT95, Dar01] présentent des principales contributions au sujet de l'extraction des règles à partir d'un RNA. Plusieurs algorithmes d'extraction de règles dans les RNA y sont présentés. Un schéma de classification de ces algorithmes est présenté dans [ADT95] ; ce schéma est basé sur quatre aspects : la forme et la qualité des règles extraites, la nécessité d'un algorithme spécifique d'apprentissage, la complexité de l'algorithme d'extraction des règles et la compréhensibilité du RNA.

Trois approches principales pour extraire les règles sont présentées [ADT95, Dar01] :

1. L'approche pédagogique. Le réseau est vu comme une "boîte noire", l'algorithme produit toutes les différentes combinaisons des entrées qui peuvent rendre la sortie active.
2. L'approche décompositionnelle. Pour chaque nœud de la couche interne ou de la couche de sortie, la méthode extrait les différentes combinaisons de liens de connexion pouvant le rendre actif.
3. L'approche hybride. Elle combine les deux approches précédentes.

Nous nous intéressons à l'approche décompositionnelle. Plusieurs algorithmes [DGBG01, Cra96, CS94, TS93] d'extraction de règles sont présentés dans la littérature et les règles extraites sont présentées sous deux formats :

1. '**si (condition) alors conclusion**'. Où *condition* est une forme normale conjonctive des variables d'entrée du neurone et *conclusion* l'étiquette d'un neurone. Toutes les différentes combinaisons des liens de connexion qui peuvent rendre le neurone actif sont produites. Dans la suite nous utiliserons aussi l'expression 'si alors' pour parler de cette forme de règles.
2. **'si (plusieurs) parmi N alors conclusion'** où N un ensemble de conditions (ou de connexions en entrée) et si un nombre positif non nul m des conditions de N est vérifié alors la conclusion est aussi *VRAI*. Nous utiliserons l'expression 'm parmi N' pour désigner cette forme de règles [CS97, Set01].

Les approches existantes sont limitées à l'extraction exclusive d'un format de règles. L'objet de ce chapitre est de présenter le processus d'extraction de règles à partir d'un RNA, les approches utilisées et d'expliquer les résultats proposés par le RNA pour un exemple donné. Nous présentons aussi une nouvelle approche que nous avons développée. Cette approche appelée $MaxSubsets$ [Tso09] qui suivant le choix de l'utilisateur permet d'extraire l'une des formes est également présentée dans ce chapitre. Sa particularité est d'extraire une structure intermédiaire à partir de laquelle les règles 'si alors' ou les règles 'm parmi N' peuvent être dérivées.

Les règles contenues dans les RNA construits avec la méthode MCLANN seront aussi extraites dans ce chapitre. Comme chaque neurone dans ces RNA correspond à un ensemble d'attributs, alors l'activation du neurone peut être expliquée par la combinaison de ces attributs. Nous proposerons une méthode expliquant pour un exemple donné comment la sortie du RNA a été calculée en recherchant les différents neurones activés.

Ce chapitre est structuré de la manière suivante : la prochaine section présentera le processus d'extraction de règles à partir des RNA. Nous poursuivrons dans la troisième section par la pré-

sentation des méthodes existantes d'extraction de règles. Dans la quatrième section, l'approche des Maxsubsets que nous avons développée sera présentée. La cinquième section aura pour objet l'extraction des règles à partir d'un modèle MCLANN et nous terminerons ce chapitre par la méthode qui permet d'expliquer le résultat en sortie du RNA à propos d'un exemple placé en entrée.

4.2 Extraction des règles à partir d'un RNA

Les connaissances incorporées dans un RNA sont représentées par sa topologie, sa fonction de transfert, ses poids de connexion et les seuils d'activation. L'objectif de l'extraction des règles est de présenter de manière explicite pour chaque neurone (de la couche cachée ou de la couche de sortie), les différentes combinaisons de entrées x_i pouvant rendre ce neurone actif suivant l'équation 4.1 ; où w_i est le poids de connexion du lien de l'entrée i, x_i est la valeur en sortie du neurone d'entrée i et θ est le seuil d'activation du neurone y ; la fonction f est la fonction d'activation.

$$y = f(\sum_{i}^{n} w_i x_i - \theta) \tag{4.1}$$

4.2.1 Définitions et propositions

Définition 30 *Une règle '**si condition alors conclusion**' extraite d'un réseau de neurones est une implication logique entre une partie des entrées d'un neurone (représentant la condition) et sa sortie (représentant la conclusion) indiquant que si condition (un sous-ensemble des entrées) est vérifiée alors la sortie du neurone prendra la valeur VRAI.*

Définition 31 *Une règle '**si plusieurs parmi N alors conclusion**' extraite d'un RNA est une implication logique entre les entrées d'un neurone (représentant la condition) et sa sortie (représentant la conclusion) indiquant qu'il existe un entier positif m, $1 \leq m \leq |N|$ tel que si au moins m conditions de l'ensemble N des entrées du neurone sont vérifiées alors la sortie du neurone prendra la valeur VRAI.*

La forme générale d'une règle '*m parmi N*' est 'si $\bigwedge_i (m_i \; parmi \; N_i)$ alors conclusion' ; pour chaque sous-ensemble N_i des variables (éléments) d'entrées du neurone, si au moins m_i entrées sont vérifiées alors la sortie du neurone est VRAI.

Proposition 4 *Soient* $r_1 = x \wedge y_1$ *et* $r_2 = x \wedge y_2$ *deux règles* m *parmi* N*, si* $y_1 = (1\ parmi\ S') \wedge ((m-1)\ parmi\ S)$ *et* $y_2 = m\ parmi\ S$ *avec* $|S'| = 1$ *alors la règle* $r = x \wedge y$ *avec* $y = m\ parmi (S \cup S')$ *généralise les règles* r_1 *et* r_2*.*

Preuve

Pour que la règle r_1 soit vérifiée, il faut que m variables soient vérifiées, S' et un sous-ensemble de cardinalité $(m-1)$ de S soient vérifiés simultanément ; et pour que la règle r_2 soit vérifiée, il faut qu'un sous-ensemble de m éléments de S soit vérifié. Les règles r_1 ou r_2 sont vérifiées si m éléments de $S \cup S'$ sont vérifiés. Or m parmi $(S \cup S') = r$; donc les règles r_1 et r_2 peuvent être remplacées par r.

■

Illustration 10 *Soient les deux règles* $r_1 =' 1\ parmi\ \{a\} + 3\ parmi\ \{b,c,d,e,f\}'$ *et* $r_2 =' 4\ parmi\ \{b,c,d,e,f\}'$ *suivantes obtenues d'un neurone quelconque. La règle* r_1 *est incluse dans la règle* r_2 *alors la règle* $r =' 4\ parmi\ \{a,b,c,d,e,f\}'$ *peut remplacer* r_1 *et* r_2*.*

Proposition 5 *Soit* $R = \{r_1, r_2, ..., r_p\}$ *un ensemble de règles 'si alors', supposons que les règles* $r_i = x \wedge y_i$ *ont le même nombre* m *de variables et* N_i *l'ensemble des variables de la partie* y_i *et* $N_0 = \bigcap_i N_i$*, si* $p = C^{m-|N_0|}_{|\bigcup_i N_i - N_0|}$ *alors la règle 'm parmi N'* $r = x \wedge y$ *où* $y = |N_0|$ *parmi* $N_0 \wedge (m - |N_0|)$ *parmi* $(\bigcup_i N_i - N_0)$ *est équivalente à l'ensemble des règles* R*.*

Preuve

Si le nombre de règles est $p = C^{m-|N_0|}_{|\bigcup_i N_i|}$, ceci implique la partie y_i de toutes les règles est formée de N_0 combinée à $m - |N_0|$ éléments pris dans le complémentaire de N_0 dans l'ensemble des variables.

En développant la règle '($|N_0|$ parmi N_0) $\wedge$ (($m - |N_0|$) parmi ($\bigcup_i N_i - N_0$))', on trouvera les règles 'si alors' formées de N_0 combinées à un sous-ensemble de $(m - |N_0|)$ éléments choisis par les autres variables.

■

Illustration 11 *Soient* $r_1 = 3$ *parmi* $\{a,b,c\}$ *;* $r_2 = 3$ *parmi* $\{a,c,d\}$ *et* $r_3 = 3$ *parmi* $\{b,c,d\}$ *alors* $N_0 = \{c\}$ *et* $r = 1$ *parmi* $\{c\} \wedge\ 2\ parmi\ \{a,b,d\}$

Théorème 3 *Les règles 'si alors' et les règles 'm parmi N' extraites du même neurone sont équivalentes.*

Preuve

Soit $r =' \bigwedge_i (m_i \; parmi \; N_i) \; alors \; y'$ une règle '*m parmi N*' et supposons que $r = m \; parmi \; N$ alors $\forall N' \subseteq N$ tel que $|N'| \geq m$ on peut construire une règle 'si alors' *si FND(N') alors y* où FND est une forme normale disjonctive des éléments de N'. L'ensemble des règles *si alors* ainsi construites correspond à la règle r. Il suffit de trouver tous les sous-ensembles de N de cardinalité au moins égale à m.

Soit $R = \{r_1, s_2, ..., s_p\}$ un ensemble de règles 'si alors'. Supposons que les règles r_i ont le même nombre m de variables et N_i l'ensemble des variables de la règle r_i. Nous pouvons utiliser les propositions 4 et 5 pour transformer cet ensemble de règles 'si alors' en un ensemble de règles 'm parmi N'.

■

Exemple 6 *Considérons le neurone de la figure 4.1 où le neurone Y a pour équation $y = f(4.5x_1 + 3.5x_2 + 2.5x_3 + 2x_4 - 1.55x_5 - 5x_6 - 6)$ avec $f(x) = sign(x) = 0$ si $x < 0$ et 1 sinon. Supposons que $x_i \in \{0, 1\}$ (VRAI=1 et FALSE=0).*

L'extraction des règles pour ce neurone va consister à produire les règles suivantes :

- *Pour les règles 'si alors' :*
  $y \Leftarrow \bar{x}_6, x_1, x_2$;
  $y \Leftarrow \bar{x}_6, x_1, x_3, x_4$;
  $y \Leftarrow \bar{x}_6, x_1, x_3, \bar{x}_5$;
  $y \Leftarrow \bar{x}_6, x_2, x_3, x_4$;
  $y \Leftarrow \bar{x}_6, x_2, x_3, \bar{x}_5$;
  $y \Leftarrow x_1, x_2, x_3, x_4$.
- *Pour les règles 'm parmi N' :*
  $y \Leftarrow 3 \; parmi \; \{\bar{x}_6, x_1, x_2\}$;
  $y \Leftarrow 1 \; parmi \; \{\bar{x}_6\} \wedge 2 \; parmi \; \{x_1, x_2, x_3\} \wedge 1 \; parmi \; \{x_4, \bar{x}_5\}$;
  $y \Leftarrow 4 \; parmi \; \{x_1, x_2, x_3, x_4\}$

Où $y \Leftarrow x_i$ signifie y est une conclusion de la règle et la combinaison x_i la prémisse ; $\bar{x}$ est la négation de la variable x (complémentaire de la variable x) ; $\wedge$ est l'opérateur booléen ET.

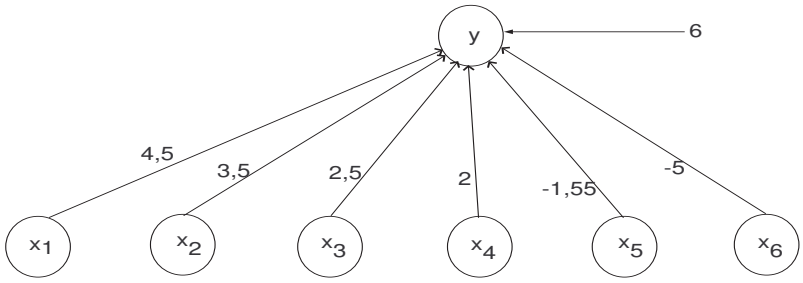

FIG. 4.1 – Exemple de neurone avec six connexions et $\theta = 6$

4.2.2 Importance des règles extraites du RNA

Les règles extraites d'un RNA sont d'une importance capitale. Elles permettent de remplir les objectifs suivants :

1. **Capacité d'expliquer les décisions ou les résultats produits à un utilisateur.**
   En Intelligence Artificielle, l'explication consiste à expliciter la structure interne afin de la rendre utile pour le raisonnement et la compréhension des résultats. Il s'agit de répondre à la question "*Comment les décisions sont prises ?*" Cette explication permet de vérifier la logique interne du système et de donner à l'utilisateur novice la possibilté d'avoir un aperçu sur le problème étudié [Gal88].
2. **Extension de l'utilisation des RNA à des domaines "critiques".**
   L'extraction des règles permet de doter les RNA d'une capacité à expliquer les résultats qu'il produit. A cet effet, ce processus permet d'utiliser les RNA dans les domaines où l'explication des résultats est capitale. Les règles permettront à l'utilisateur d'avoir une idée du résultat attendu du RNA quand il lui présentera un exemple [Thr95].
3. **Test logiciel et débogage des composants neuronaux d'un logiciel.**
   La validation des règles extraites peut permettre de comparer les résultats fournis par le RNA aux spécifications mentionnées dans le cahier de charges du logiciel intégrant un composant neuronal.
4. **Amélioration de la généralisation par le RNA.**
   Un RNA est généralement vu comme une boîte noire ; à cet effet, il est difficile de déterminer quand la généralisation peut échouer. Les règles extraites peuvent permettre d'anticiper la décision du RNA et de prédire les circonstances dans lesquelles la généralisation peut échouer.
5. **Exploration des données et induction de la théorie scientifique.**
   En effet, les RNA sont capables de découvrir les possibles relations non linéaires entre

les attributs. L'extraction des règles permet alors de déduire une théorie sur ces relations entre attributs.

6. **Production des règles pour des systèmes symboliques d'intelligence artificielle.** Pour les domaines où on ne dispose pas de règles, les utilisateurs peuvent se servir des RNA pour obtenir les règles ou pour les raffiner.

4.2.3 Processus d'extraction des règles

Le processus d'extraction des règles (suivant l'approche décompositionnelle) à partir d'un RNA suit deux phases :

1. décomposition du RNA : le réseau est décomposé en différents neurones qui le composent.

2. extraction (proprement dite) des règles. Elle considère individuellement chaque neurone (de la couche cachée et de la couche de sortie) et produit les différentes combinaisons des entrées de ce neurone capables de le rendre actif.

Les règles obtenues peuvent être validées ou testées sur un ensemble d'exemples. La décomposition du RNA suppose que la structure de ce dernier est compréhensible. Une structure d'un RNA est compréhensible si une sémantique est associée à chaque neurone et à chaque connexion.

Exemple 7 *La figure 4.2 est un exemple de RNA à partir duquel les deux neurones présentés dans la figure 4.3 ont été extraits par décomposition. Dans la figure 4.3, nous présentons deux neurones* cl_2 *(figure 4.3 (a)) de la couche de sortie et* d *(figure 4.3(b)) de la couche cachée obtenue en décomposant le RNA initial. Les étiquettes des différents neurones (des couches cachées) sont des intensions des concepts correspondants.*

Après la décomposition du RNA, les algorithmes sont appliqués à chaque neurone pour extraire l'ensemble de règles décrivant son comportement en fonction de ses entrées. Dans la prochaine section nous présenterons les approches décompositionnelles existantes d'extraction de règles à partir d'un RNA.

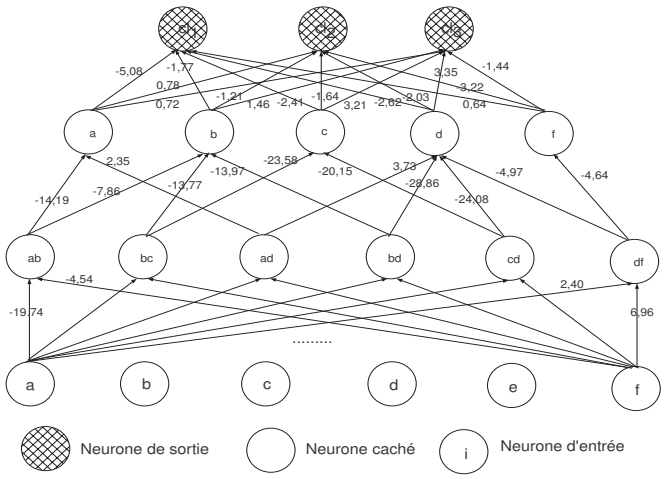

FIG. 4.2 – Exemple de RNA à partir duquel on peut extraire les règles.

4.3 Approches existantes d'extraction de règles

L'extraction de règles à partir d'un RNA est une tâche très complexe. Plusieurs algorithmes [ADT95, Boz95, Dar01, LJ97] sont proposés dans la litérature pour résoudre ce problème. Ces algorithmes peuvent être regroupés en deux classes : les algorithmes qui produisent des règles sous la forme '*si alors*' dont le plus connu est l'algorithme 'Subset' [Fu91] et récemment la méthode coalition-opposition [BHME07, Bad09] ; et les algorithmes qui produisent les algorithmes sous la forme 'm parmi N' [Set01] ; à l'origine, cette nouvelle forme de règles était extraite par l'algorithme '*MofN*', mais plusieurs variantes ont été mises au point. D'autres recherches [KM05, KM06] se focalisent sur l'extraction des règles floues à partir d'un RNA.

4.3.1 Approche Subset

Principe

Cette approche consiste pour un neurone y à générer et tester les sous-ensembles de liens de connexions en entrée de y et de conserver les prémisses qui activent y. L'approche *Subset* [Fu91] partitionne l'ensemble de liens de connexion en deux groupes : le groupe des liens ayant les poids positifs (ensemble des positifs) S_p et le groupe des liens ayant les poids négatifs (ensemble des négatifs) S_n. Les sous-ensembles positifs capables d'activer le neurone y sont d'abord extraits. Pour chaque sous-ensemble positif P extrait, l'algorithme extrait aussi les

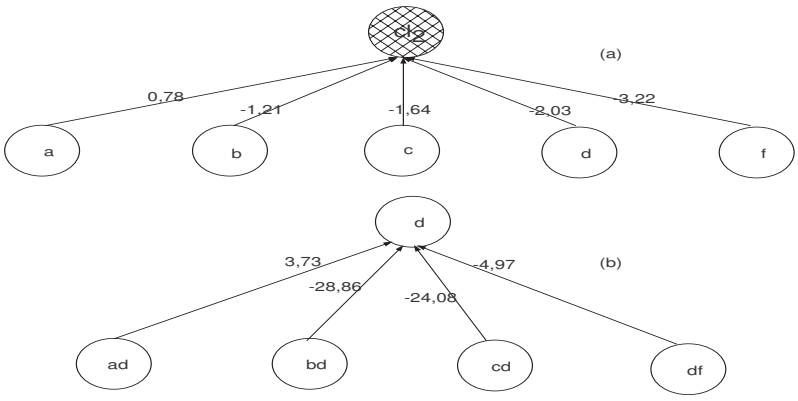

FIG. 4.3 – Deux neurones obtenus par décomposition du RNA de la figure 4.2. Le neurone cl_2 est un neurone de sortie dans le RNA (figure 4.2 avec en entrées a, b, c, d et f ; le neurone d est un neurone de la couche cachée et qui a pour entrées ad, bd, cd et df.

sous-ensembles négatifs Q qui vérifient les contraintes suivantes :
C_1 **:** la somme des poids (en valeur absolue) de Q ajoutées au seuil d'activation reste inférieure à la somme des poids du sous-ensemble positif considéré.
C_2 **:** la somme des poids de P et ceux du complément de Q ($S_n - Q$) est supérieure au seuil.

Le complément de chaque sous-ensemble négatif extrait est ajouté à ce sous-ensemble positif pour constituer la prémisse d'une règle ayant pour conclusion y.

Les règles extraites sont sous la forme "*si (prémisse) alors conclusion*". L'algorithme 13 présente la version d'une approche Subset développée dans [Ned98]. Il reçoit en entrée un neurone, ses entrées ainsi que les poids de connexion respectifs, puis retourne la liste des règles extraites de ce neurone.

Complexité

A cause du fait qu'il génère les sous-ensembles et les teste, sa complexité est en $O(2^n)$, n est le nombre de connexions.

Limites

L'approche *Subset* souffre de quelques défaillances :

1. Sa complexité est exponentielle. En effet le coût de recherche des sous-ensembles est

Algorithme 13 pseudo-code de l'algorithme Subset d'extraction des règles

Entrées: Un réseau de neurones, ses entrées et les poids de connexion.

Sorties: Pour chaque neurone (caché ou de sortie) y, fournit l'ensemble des règles ayant pour conclusion y.

1: **Pour tout** neurone de la couche cachée ou de la couche de sortie **faire**
2: Affecter à S_p l'ensemble des poids positifs de l'unité ;
3: Affecter à S_n l'ensemble des poids négatifs de l'unité ;
4: **Pour tout** tout sous-ensemble P de S_p dont la somme des poids est supérieur au seuil **faire**
5: Trouver les sous-ensembles Q de S_n qui vérifient les deux propriétés C_1 et C_2 ;
6: Pour chaque sous-ensemble Q, former la règle : "Si P et $\bar{Q}$ alors y".
7: **fin pour**
8: **fin pour**

exponentiel car il s'agit de tester les éléments d'un ensemble de parties d'un ensemble. Pour résoudre ce problème, certains auteurs [Fu91] proposent de limiter le nombre d'antécédents pour chaque règle produite. Ce problème réduit son utilisation à des réseaux de petite taille.

2. Le nombre de règles extraites est très élèvé. A cause de cette taille très élèvée de l'ensemble des règles, l'exploitation est difficile.
3. Une autre difficulté est qu'elle est limité à l'extraction exclusive des règles sous le format 'si alors'.

Afin de résoudre du nombre élevé de règles extraites, l'approche coalition-opposition introduite dans [BHME07] propose un ensemble minimal de conditions. A cet ensemble, les autres prémisses peuvent être déduites.

4.3.2 Approche Coalition-Opposition

Cette approche consiste à extraire pour un neurone des coalitions minimales et des oppositions minimales. Les coalitions sont des sous-ensembles de connexions capables d'activer le neurone et les oppositions sont des sous-ensembles capables de le désactiver. Elles sont minimales si aucun de leurs sous-ensembles ne sont pas capables d'activer ou de désactiver le neurone.

Définition 32 *La forme positive d'un neurone est obtenue en multipliant les poids de connexion*

négatifs par -1 et en complémentant les variables correspondantes.

De manière similaire, la forme négative d'un neurone est obtenue en multipliant les poids de connexion positifs par -1 et en complémentant les variables correspondantes.

Soient U l'ensemble des entrées d'un neurone y, et V un sous-ensemble de U, l'activité minimale A_{min} de V est calculée à partir de la forme positive de y par la formule 4.2 et l'activité maximale A_{max} à partir de la forme négative de y par la formule 4.3.

$$A_{min}(V) = \sum_{i \in V} w_i - \sum_{i \in U/V} |w_i| \tag{4.2}$$

$$A_{max}(V) = \sum_{i \in V} w_i + \sum_{i \in U/V} |w_i| \tag{4.3}$$

Définition 33 *V est une coalition si $A_{min}(V) \geq \theta$ et une opposition si $A_{max} < \theta$.*

V est une coalition maximale si aucun de ses sous-ensembles n'est une coalition ; V est une opposition minimale si aucun de ses sous-ensembles n'est une opposition.

L'approche coalition-opposition recherche les coalitions minimales à partir d'un arbre. A partir de la forme positive, l'arbre est construit de telle sorte que les différentes branches constituent les différentes coalitions. Pour la construction de cet arbre, les insertions de nouvelles variables se font par ordre croissant des poids de connexion ; c'est-à-dire le poids de connexion d'une variable correspondante à un nœud de l'arbre est toujours supérieure ou égale au poids de ses fils. L'élagage se fait dès que l'ensemble des variables correspondant à l'ensemble des nœuds constituant le chemin allant de la racine jusqu'au nœud nouvellement inséré forme une coalition. Les oppositions sont extraites de la manière en utilisant la forme négative.

Les coalitions forment ainsi l'ensemble des prémisses pour les règles extraites du neurone considéré. La règle pour le neurone y s'écrit comme la forme normale conjonctive de la liste des coalitions. Pour exprimer la sortie du neurone y en fonction des variables d'entrées du RNA, de manière itérative, chaque variable complémentée est remplacée par la forme normale conjonctive de ses oppositions et chaque variable non complémentée par la forme normale conjonctive de ses coalitions.

Le pseudo-code de l'approche coalition-opposition est présenté par l'algorithme 14. La correction et la complétude de cet algorithme sont démontrées dans [BHME07].

Algorithme 14 pseudo-code de l'algorithme Coalition-opposition d'extraction des règles

Entrées: Un réseau de neurones, ses entrées et les poids de connexion.
Sorties: Pour chaque neurone y, fournit l'ensemble des règles ayant pour conclusion y.

1: **Pour tout** neurone de la couche cachée ou de la couche de sortie **faire**
2: Calculer la forme positive de y ;
3: Calculer la forme négative de y ;
4: Extraire les coalitions de y ;
5: Extraire toutes les oppositions de y ;
6: Former une règle pour y avec la forme normale conjonctive de ses coalitions ;
7: **fin pour**
8: Exprimer la règle de y en fonction des entrèes du RNA.

Complexité

L'approche coalition-opposition recherche les sous-ensembles de connexions d'un neurone ; si le neurone a n connexions alors sa complexité est bornée par 2^n.

Limites

La complexité exponentielle de cette approche est sa principale limite. Mais elle est aussi limitée à la recherche des règles 'si alors' et sa complexité mémoire est aussi élevée due à la construction de l'arbre.

Le nombre de règles 'si alors' étant toujours élevé, les chercheurs [TS93, Set01] ont proposé le format de représentation 'm parmi N' qui permet de représenter plusieurs règles en une seule. L'approche MofN [TS93, Ned98, Set01] extrait les règles écrites dans ce format.

4.3.3 Approche MofN

Cette approche extrait les règles sous forme résumée car plusieurs prémisses sont compressées en une seule sous la forme *'m parmi les éléments de l'ensemble N'*. D'après les auteurs [TS93, Set01], la complexité est aussi réduite grâce au prétraitement appliqué sur le réseau avant l'extraction des règles proprement dite.

Principe

L'idée de l'algorithme *MofN* [Cra96] est qu'un lien de connexion unique n'a pas d'importance alors qu'un groupe d'antécédents forme une classe dans laquelle chaque lien de connexion a la même importance. Cette équivalence a pour conséquence que l'intérêt de l'algorithme se porte aux groupes de connexion sans se soucier d'une connexion particulière. Les différentes étapes d'extraction sont :

1. **Le partitionnement**. La première étape de MofN est de créer les classes d'équivalence. Les différents liens de connexions sont rangés par groupes similaires (classes d'équivalentes). Plusieurs techniques de partitionnement sont utilisées pour l'extraction des règles à partir d'un RNA. Ces techniques vont des algorithmes de partitionnement classique comme Kmeans [Mac67, SL00] aux approches génétiques [HE06, KA09].
2. **Le calcul des moyennes**. Après la formation des groupes, l'étape suivante consiste à calculer pour chaque classe la moyenne des poids de connexion. Et désormais les poids de connexion de toutes les connexions de cette classe sont égaux à cette moyenne.
3. **L'élagage**. La procédure identifie les classes moins influentes et les supprime [Set97]. Une classe est jugée moins influente si elle n'a pas d'effet ou si son effet est négligeable sur l'activité de ce neurone (qu'il soit actif ou pas).
4. **Le réapprentissage**. Cette étape consiste à refaire l'apprentissage du réseau élagué à l'étape précédente. Mais cet apprentissage ne modifie que les seuils d'activation des neurones.
5. **L'extraction des règles**. Cette étape forme les règles qui décrivent exactement le réseau. Ces règles sont obtenues en transformant directement les seuils d'activation et les poids de connexion entrainant la conclusion à la valeur vrai.
6. **La simplification**. L'algorithme simplifie la règle en produisant les différentes combinaisons des antécédents dont la somme dépasse le seuil.

   L'algorithme 15 présente un pseudo-code l'algorithme MofN [TS93] d'extraction des règles à partir d'un réseau de neurones.

Complexité

Une estimation de la complexité de cet algorithme est présenté dans [TS93]. Cette compléxité n'est pas précise car l'étape de réapprentissage n'est pas prise en compte. De ce calcul, les différentes étapes sont exécutées de la manière suivante :

– Formation des groupes (étape 2) : en $O(ul^2)$ où u est le nombre de neurones dans le réseau et l le nombre moyen de liens par neurone.

Algorithme 15 Pseudo-code de l'algorithme MofN

Entrées: Un réseau de neurones.

Sorties: Pour chaque neurone y, fournit l'ensemble des règles ayant pour conclusion y.

1: **Pour** tout neurone de la couche cachée ou de la couche de sortie **faire**
2: Former les groupes de connexions similaires ;
3: Calculer la moyenne des poids de connexion de chaque groupe et l'affecter comme poids de connexion du groupe ;
4: Eliminer les groupes qui n'affectent pas significativement l'activité du neurone ;
5: Conserver tous les poids de connexion constants, réapprendre le réseau en modifiant seulement les seuils d'activation par l'algorithme de rétropropagation.
6: **fin pour**
7: Former pour chaque neurone une règle simple. Cette règle est constituée de seuil et d'antécédents de connexion des groupes restants ;
8: Simplifier les règles (si possible) en éliminant les poids et seuils superflus. Produire les différentes combinaisons des antécédents dont la somme dépasse le seuil d'activation.

– Suppression des liens moins influents (étape 4) : en $O(n \times u \times l)$; où n est le nombre d'exemples dans l'ensemble d'apprentissage.
– Les autres étapes : en $O(ul)$.

En conclusion, d'après [TS93], cette complexité est en $O(max(ul^2, n \times u \times l))$.

Limites

La technique MofN permet d'obtenir un nombre réduit de règles. Mais le prétraitement pose un certain nombre de problèmes :

1. Choix de l'algorithme de partitionnement et de ses paramètres. En fait le résultat du partitionnement dépend de l'initialisation de l'algorithme.
2. Suppression des connexions qui paraissent moins influentes alors qu'elles contribuent à la décision finale.
3. Aucune assurance sur la convergence du processus de réapprentissage. S'il n'y a pas convergence dans cette étape alors le réseau à partir duquel les règles seront extraites ne sera pas capable de bien classer les objets.

La forme 'm parmi N' permet de réduire le nombre de règles et donc de réduire aussi l'espace mémoire nécessaire pour le stockage de ces règles. Mais les problèmes liés au prétraitement limitent l'approche. Par contre, pour une bonne explication, nous pensons qu'une forme

plus explicite comme les règles 'si alors' serait plus appropriée. Nous proposons donc une nouvelle approche appelé approche des MaxSubsets qui permet d'extraire les deux formes de règles 'if then' et 'm parmi N'. Cette approche permet d'obtenir l'une et l'autre forme de règles en limitant les effets liés aux limites des méthodes précédentes.

L'approche des MaxSubsets extrait une structure à partir de laquelle les prémisses de taille minimale peuvent être déduites comme dans l'approche coalition-opposition. Elle étend l'utilisation de la structure construite à l'extraction des règles 'm parmi N'. Pour une gestion efficace de la mémoire, l'approche des MaxSubsets utilise un vecteur binaire pour rechercher et représenter les conditions.

4.4 Nouvelle approche : MaxSubsets

Nous venons de présenter les approches existantes d'extraction des règles à partir d'un réseau de neurones. Les algorithmes existants procèdent en deux phases : la phase éventuelle de prétraitement (suppression des connexions redondantes ou moins influentes, et réapprentissage) et la phase d'extraction de règles. Les problèmes liés au prétraitement sont multiples : difficulté de choix de la forme de règles et du paramètrage des algorithmes de partitionnement, aucune garantie sur la réussite du réapprentissage et que le réseau obtenu sera fidèle au réseau original [Dar01]. Pour éviter ces problèmes, aucun prétraitement n'est appliqué au réseau avant l'extraction de règles dans la nouvelle approche (appelée approche des MaxSubsets [Tso09]) que nous présentons. Nous utilisons dans cette approche la définition de la forme positive d'un neurone comme définie dans l'approche coalition-opposition [BHME07] et nous supposerons que les entrées sont binaires (0,1).

Définition 34 *Soient y un neurone, U l'ensemble des connexions du neurone y, V un sous-ensemble de U et $\bar{V}$ le sous-ensemble du complémentaire de V dans U constitué des éléments complémentés dans la forme positive de U ; l'activité de y sous l'effet de V est calculée par la formule 4.4 où V_+ est l'ensemble des éléments non complémentés de V.*

$$A(V) = \sum_{i \in V_+} w_i - \sum_{i \in \bar{V}} w_i \tag{4.4}$$

Exemple 8 *La forme positive du neurone cl_3 de la figure 4.1 est $\{(5; \bar{x}_6), (4,5; x_1), (3,5; x_2), (2,5; x_3), (2; x_4), (1,55; \bar{x}_5)\}$. L'activité de y sous l'effet de l'ensemble de connexions $V =$*

$\{\bar{x}_6, x_1, x_2\}$ *est* $A(V) = (4,5+3,5) - 1,55 = 6,45$.
Et sous l'effet de l'ensemble de connexions $V' = \{x_2, x_3, \bar{x}_5\}$ *cette activité est calculée comme suit :* $V'_+ = \{x_2, x_3\}; \bar{V'} = \{\bar{x_5}\}$ *et* $A(V') = (3,5+2,5) - 5 = 1$

Définition 35 *Soit y un neurone ayant pour seuil d'activation θ et U l'ensemble des entrées de y, un sous-ensemble V de U est un* ***Subset*** *si $A(V) \geq \theta$. V est maximal (****MaxSubset****) si $\forall V' \subset V \; A(V') < \theta$*

Exemple 9 *Le vecteur V de l'exemple 8 est un MaxSubset alors V' du même exemple ne l'est pas.*

Un MaxSubset peut être vu comme une coalition minimale [BHME07], mais toutes les entrées sont positives ou nulles.

L'approche des *MaxSubset* [Tso09] consiste à extraire pour un neurone donné une structure intermédiaire appelée liste des $MaxSubsets$ (Maximum Subsets) à partir de laquelle il sera possible d'extraire les règles 'si alors' et les règles 'm parmi N'. Un MaxSubset pour un neurone donné est un antécédent capable de rendre ce neurone actif ; tous les sous-ensembles d'un MaxSubset rendent le neurone en question inactif et tous ses super ensembles activent le neurone. Les heuristiques consistant à limiter l'espace de recherche aux sous-ensembles dont la cardinalité sera bornée par des valeurs $kmin$ et $kmax$ (que nous déterminerons) sont introduites dans l'algorithme. Nous montrons comment générer les règles 'si alors' et les règles 'm parmi N' à partir de la liste de MaxSubsets.

4.4.1 Extraction des MaxSubsets

La recherche des MaxSubsets implique la production de sous ensembles de cardinalité bornée par $kmin$ et $kmax$ et sélectionner ceux qui sont capables d'activer le neurone. Les différentes étapes de cette recherche sont les suivantes :

1. Calculer la forme positive du neurone.
2. Trier par ordre décroissant les connexions suivant leurs poids.
3. Calculer $kmin$ et $kmax$.
4. Extraire l'ensemble des MaxSubsets dans la partition délimitée par $kmin$ et $kmax$.

La forme positive se calcule comme dans le cadre de l'approche coalition-opposition : *si* $w_i < 0$ *alors complémenter l'entrée* i *et considérer plutôt* $|w_i|$ *sinon pas de changement.*

Le tri des connexions consiste à utiliser le résultat de l'étape précédente et à classer par ordre décroissant les différentes connexions suivant leur poids.

Corrolaire 1 *Le tri par ordre décroissant des connexions implique :* $A(V) \geq A(next(V))$

Preuve
Soient V un vecteur caractéristique et $V' = next(V)$, on sait qu'il existe seulement deux indices l et $l+1$ telles que (V_l=1, $V'_l = 0$), ($V_{l+1} = 0, V'_{l+1} = 1$) et $\forall i$, $i < l$ et $i > l+1$, $V_i = V'_i$. Les activités du neurone sont calculées comme suit :
$A(V) = \sum_{i \in V_+} w_i - \sum_{i \in \bar{V}} w_i = a + w_l - w_{l+1}$ et $A(V') = \sum_{i \in V'_+} w_i - \sum_{i \in \bar{V'}} w_i = a - w_l + w_{l+1}$; a est la contribution de la partie commune entre V et V'.
$A(V) - A(V') = 2w_l - 2w_{l+1}$, or comme le vecteur W est trié par ordre décroissant alors $w_l \geq w_{l+1} \Longrightarrow A(V) \geq A(V')$.

■

Comme $kmin$ et $kmax$ représentent respectivement le nombre minimum et maximum de connexions pouvant activer le neurone et que le tri est par ordre décroissant, les $kmax$ connexions sont regroupées à la fin du vecteur alors que les $kmin$ sont regroupés au début du vecteur.

L'étape à laquelle nous allons nous intéresser est celle de l'extraction des MaxSubsets.

Enumération des Maxsubsets

La structure de représentation des sous-ensembles est le vecteur binaire que nous appelons *vecteur caractéristique* du sous-ensemble.

Définition 36 *Le* ***vecteur caractéristique*** V *d'un ensemble* $E_1 \subseteq E$ *est un vecteur binaire de taille* $|E|$ *et tel que* $v_i = 1$ *(composant à la position* i*) si l'élement à la position* i *appartient à* E_1 *et 0 sinon.*

Nous pouvons alors représenter un sous-ensemble de variables d'entrée d'un neurone par son vecteur caractéristique. Cette représentation binaire nous permet de réduire l'espace mémoire nécessaire pour le stockage des MaxSubsets et facilite d'y appliquer les heuristiques pour élaguer l'espace de recherche..

Définition 37 *Un vecteur caractéristique V représentant un sous-ensemble de connexion d'entrée d'un neurone ayant pour seuil d'activation θ est dit* ***valide*** *si $A(V) \geq \theta$.*

Un MaxSubset est un vecteur caractéristique valide. Nous utiliserons les fonctions $next()$ et $new\_next()$ dans cet algorithme pour rechercher les vecteurs.

Soient e^i, le vecteur V de taille n tel que : $V_i = 1$ et $\forall j \neq i,\ V_j = 0$; l_V un entier inférieur ou égal à n tel que $V_{l_V} = 1$ et $\forall i > l_V, V_i = 0$.

Définition 38 *Soit V le vecteur caractéristique d'un ensemble E, $V' = next_i(V)$ est définie de la manière suivante :*

$$next_i : U \longrightarrow U$$
$$V \longrightarrow V - e^i + e^{i+1}$$

Nous noterons $next_{l_V} = next(V)$.

Exemple 10 *Le sous-ensemble de connexion $\{\bar{x}_6, x_1, x_2\}$ a pour vecteur caractéristique $V = 111000$. Nous avons $l_V = 3$, donc $e^3 = 001000$ et $e^4 = 000100$. L'image de V par la fonction next() est $V' = 111000 - 001000 + 000100 = 110100$.*

Théorème 4 *Soit V un vecteur caractéristique et $V' = next(V)$, si V n'est pas valide alors V' ne l'est pas aussi.*

Preuve

Puisque les poids sont triés par ordre décroissant, la somme des connexions appartenant à V est supérieure ou égale à celle de V' ($A(V) \geq A(V')$).
V n'est pas valide $\Longrightarrow A(V) < \theta$.
Or $A(V) \geq A(V') \Longrightarrow A(V') < \theta \Longrightarrow V'$ n'est pas valide.

■

Ce théorème est utile pour l'élagage de l'espace de recherche des MaxSubsets. Alors dès qu'un vecteur devient non valide, tous les candidats obtenus par application de la fonction $next()$ sont ignorés.

Si k est égal à $|E|$ alors son image à travers la fonction $next()$ n'existe. Dans ce cas la fonction $new\_next()$ est utilisée. La fonction $new\_next()$ permet de trouver le successeur V' (valide) de V de même cardinalité.

Définition 39 *Soit V un vecteur caractéristique, $V' = new\_next(V)$ est calculé de la manière suivante : soient k_1 un entier positif tel que $V_{k_1-1} = 0$, $\forall i\ (k_1 \leq i \leq l_V), V_i = 1$: s'il existe $k_0, k_0 < k_1 - 1$, $V_{k_0} = 1$ et $\forall i\ (k_0 < i < k_1),\ V_i = 0$ alors*

$$new\_next : U \longrightarrow U$$
$$V \longrightarrow (V - e^{k_0} - \sum_{(k_1 \leq i \leq l_V)} e^i) + \sum_{(k_0+1 \leq i \leq (l_V - k_1 + k_0 + 2))} e^i$$

Exemple 11 *Soit le sous-ensemble $\{\bar{x}_6, x_1, x_3, x_4\}$; son vecteur caractéristique est V=110110. Son image par la fonction new_next se calcule de la manière suivante :$k_0 = 2$; $k_1 = 4$ et $l_V = 5$. $V'_1 = V_1 = 1$; $V'_{k_0} = V'_2 = 0$; $V'_3 = V'_4 = V'_5 = 1$; $V'_6 = 0$; donc $V' = new\_next(V) = 101110$.*

Etant donné un entier positif k et un neurone y ayant pour entrées U, les fonctions $next()$ et $new\_next()$ permettent de calculer tous les sous-ensembles V valides de U de cardinalité k. Pour rechercher les sous-ensembles valides de U de cardinalité supérieure à k, les générateurs de cardinalité k permettent d'éviter de tester les sous-ensembles valides de U de cardinalité $k+1$ ayant un sous-ensemble de cardinalité k valide.

Définition 40 *Un vecteur caractéristique V est appelé **générateur** si V est valide et $next(V)$ n'est pas valide ou n'existe pas.*

Exemple 12 *Les vecteurs caractéristiques 111000, 110110 et 101110 sont des générateurs.*

Soient G un générateur, la fonction *générer(G)* permet de calculer le vecteur V valide et ayant plus d'éléments que G. Cette fonction permet de trouver le sous-ensemble de connexions à ajouter à $next(G)$ ou à $new\_next(G)$ (s'il n'est pas valide) pour obtenir un vecteur caractéristique valide.

Définition 41 *Soit V un générateur, le vecteur caractéristique V'=générer(V) est calculée de la manière suivante :*

1. *Si $l_V < n - 1$ alors $V' = V - e^{l_V} + \sum_{i=1}^{t} e^{l_V + i}$*

2. *sinon : soient* k_1 *un entier positif tel que* $V_{k_1-1} = 0$, $\forall i\ (k_1 \leq i \leq l_V), V_i = 1$ *: s'il existe* k_0 *tel que* $k_0 < k_1 - 1$, $V_{k_0} = 1$ *et* $\forall i (k_0 < i < k_1), V_i = 0$ *alors*
   $V' = V - e^{k_0} - \sum_{k_1 \leq i \leq l_V} e^i + \sum_{(k_0 < i \leq (l_V - k_1 + k_0 + t)} e^i$
   t *est le nombre d'éléments qu'il faut ajouter pour que* V' *devienne valide. Il est obtenu en ajoutant successivement les éléments à* V'.

Exemple 13 *Soit le vecteur V=111000, V'=générer(V) se calcule de la manière suivante :* $V_3 = 1$ *et pour tout* i, $3 < i \leq 6$, $V_i = 0$ *; on aura donc* $V'_1 = V_1 = 1$, $V'_2 = V_2 = 1$, $V'_3 = 0$, $V'_4 = V'_5 = 1$, $V'_6 = 0$ *avec t=1 ;* $V' = 110110$. *Comme V' est valide alors générer(V)=V'.*

Les générateurs permettent aussi d'élaguer l'espace de recherche. En effet, les générateurs de cardinalité k constituent les racines de recherche des MaxSubsets candidats de cardinalité $k + t$. Les générateurs sont formées des MaxSubsets qui permettront d'obtenir les règles 'm parmi N'. La recherche des MaxSubsets et des générateurs pour un neurone donné est présentée par l'algorithme 16.

Algorithme d'extraction des MaxSubsets

Les MaxSubsets sont générés en commençant par le vecteur caractéristique de $kmin$ éléments, les $kmin$ premières positions du vecteur ont pour valeur 1 et le reste du vecteur 0. L'algorithme utilise les fonctions $next()$ et $new\_next()$ pour rechercher les autres MaxSubsets de cardinalité $kmin$. Les générateurs de cardinalité i permettent de rechercher les MaxSubsets de cardinalité $i + 1$. L'algorithme s'arrête quand les générateurs de cardinalité $kmax$ sont trouvés.

Cet algorithme utilise les notions définies précédemment pour produire les MaxSubsets associés à un neurone y. Il retourne deux résultats : LM la liste des MaxSubsets et LG celle des générateurs.

Proposition 6 *L'algorithme 16 est complet et correct.*

Preuve
Soient $V, V' \in LM$, supposons que $V \subseteq V'$.
Soit également V_1 le premier vecteur caractéristique de cardinalité $kmin$ alors on sait qu'il n'existe pas $V' \in \{V_2, V_3, ..., V_k\}$ avec $V_2 = next(V_1)$, $V_3 = next(V_2)$,...,$V_k = next(V_{k-1})$.
Soit $V = V_k$ et $V' = new\_next(V)$, il existe k_0 tel que $V_{k_0} = 1$ et $V'_{k_0} = 1$, en plus $|V| = |V'|$, donc $V' \nsubseteq V$ et $V \nsubseteq V'$. Notons que la fonction $new\_next()$ ne peut pas être appliquée aux

Algorithme 16 Recherche des MaxSubsets et des générateurs

Entrées: Un neurone donné y, ses poids de connexion en entrée et son seuil d'activation.

Sorties: Liste des MaxSubsets LM et celle des générateurs LG extraits de ce neurone.

1: Calculer la forme positive de y ;
2: Trier les connexions par ordre décroissant ;
3: Calculer $kmin$ et $kmax$;
4: Initialiser le vecteur V avec les $kmin$ premiers composants à 1 et le reste à 0 ;
5: Initialiser la liste des MaxSubsets et la liste des générateurs LG à l'ensemble vide ;
6: **Répéter**
7: Insérer V dans LM ;
8: Si ($next(V)$) n'est pas valide ou n'existe pas alors ajouter V à LG et remplacer V par $new\_next(V)$;
9: Sinon calculer $next(V)$ et l'affecter à V ;
10: **jusqu'à** V ne soit plus valide.
11: **Tant que** tous les générateurs de LG ne sont pas traités **faire**
12: Soit G le prochain générateur dans l'ordre d'insertion, calculer *V=générer(G)* ;
13: **Si** V existe **alors**
14: **Répéter**
15: Insérer V dans LM ;
16: Si $next(V)$ n'est pas valide ou n'existe pas alors insérer V dans LG et remplacer V par $new\_next(V)$;
17: Sinon remplacer V par $next(V)$;
18: **jusqu'à** V ne soit plus valide
19: **finsi**
20: **fin tantque**

vecteurs V_1, V_2,..., V_{k-1} parce que l'application de $next()$ à ces vecteurs produit un résultat. De même, en appliquant la fonction *générer* à V (V' =*générer*(V), il existe k_0 tel que $V_{k_0} = 1$ et $V'_{k_0} = 1$, $V \not\subseteq V'$. Donc il n'existe pas $V, V' \in LM$, tels que $V \subseteq V'$. L'algorithme est correct.

Supposons qu'il existe $V \notin LM$, V est un MaxSubset : $kmin \leq |V| \leq kmax$. Soit V_1 le premier MaxSubset généré ($|V_1| = kmin$). V est obtenu à partir de V_1 par application des fonctions *next()*, *new_next()* et *générer()* :

1. $\exists V_2, ..., V_k$ avec $V_2 = next(V_1)$, $V_3 = next(V_2)$,...,$V = next(V_k)$.
2. $\exists V'_2, ..., V'_k$ avec $V'_2 = new\_next(V_k), V'_3 = next(V'_2), ..., V = next(V'_k)$.
3. $\exists G_k$ (un générateur) tel que V'_2=générer(G_k), $V'_3 = next(V'_2)$,...,$V = next(V'_k)$.

Comme V est valide alors il appartient à LM ; donc l'algorithme est complet.

■

L'algorithme ne produit que les ensembles de connexions permettant de décrire le comportement du neurone. Puisque $kmin$ et $kmax$ sont finis alors le nombre de sous-ensembles de cardinalité comprise entre les deux nombres est aussi fini. Or $kmin$ (resp. $kmax$) est le nombre minimum (resp.maximum) d'entrées nécessaires pour activer la sortie du neurone. Sachant qu'un MaxSubset ne décrit qu'une situation minimale permettant d'activer le neurone, l'extraction de tous les MaxSubsets permet alors de trouver toutes les situations dans lesquelles le neurone est actif.

A partir de la liste des MaxSubsets extraits du neurone y, nous pouvons écrire une règle sous la forme normale conjonctive des MaxSubsets pour y.

Illustration 12 *Pour le neurone présenté en figure 4.1, après les deux premières étapes de l'algorithme (rendre positives les connexions et les trier), les entrées seront étiquettées et ordonnées de la manière suivante :* $\bar{x_6}, x_1, x_2, x_3, x_4, \bar{x_5}$.

$kmin$ *et* $kmax$ *sont respectivement 3 (car* $A(\bar{x_6}, x_1, x_2)$*=4,5+3,5+1,55=6,45>6) et 5 (car* $A(x_1, x_2, x_3, x_4, \bar{x_5})$*=4,5+3,5+2,5+2-5=7,5>6) et le premier vecteur caractéristique est V=111000 (étape 4) correspondant au sous-ensemble* $\{\bar{x_6}, x_1, x_2\}$.
next(V) n'est pas valide, donc V est inséré dans la liste des générateurs (étape 8), il est aussi inséré dans la liste des MaxSubsets. L'algorithme calcule alors V = générer(111000) = 110110 (étape 12) et l'insére dans la liste des MaxSubsets (étape 15). L'algorithme calcule alors

next(110110) = 110101 ; comme 110101 est valide alors il est inséré dans la liste de MaxSubsets. next(110101) n'existe pas donc il est inséré dans la liste des générateurs et new_next(110101) = 110011 est calculé. Comme 110011 est valide , il est inséré dans la liste des MaxSubsets et next(110011) n'existe pas, il est inséré parmi les générateurs. L'algorithme exécutera ces étapes jusqu'à ce que V soit égal à 011110 ; next(011110) = 011101 n'est pas valide et new_next(011110) n'existe pas. Aucun autre MaxSubset ne sera plus trouvé avec les autres générateurs et à la fin de l'algorithme les résultats seront :

- *Comme générateurs : 111000, 110101, 110011, 101101,011110.*
- *Comme MaxSubsets : 111000, 110110, 110101, 110011, 101110, 101101, 011110.*

A partir des structures intermédiaires obtenues de l'algorithme 16, les règles peuvent être calculées.

4.4.2 Obtention des règles à partir de la liste des MaxSubsets et de la liste de générateurs

A partir de la liste des MaxSubsets et celle des générateurs ainsi calculées, nous allons montrer comment l'utilisateur peut déduire les règles 'if then' ou 'm parmi N'.

Calcul des règles 'si alors'

La liste des MaxSubsets est la forme minimale des règles 'si alors' que nous pouvons calculer pour un neurone donné car un sous-ensemble d'un MaxSubset n'est pas valide, et ne peut donc pas être un prémisse ; et tous les super ensembles d'un MaxSubset forment des prémisses. Les MaxSubsets peuvent alors être écrits comme des prémisses des règles 'si alors' extraites du neurone.

Malgré cette forme minimale que présente la liste des MaxSubsets, leur nombre peut être aussi très élevé et occupent plus d'espace mémoire. Le théorème 5 montre que la liste des générateurs est une structure de petite taille à partir de laquelle tous les MaxSubsets peuvent être déduits.

Théorème 5 *Si V est un MaxSubset extrait d'un neurone alors il existe un générateur G extrait du même neurone à partir duquel V peut déduit.*

Preuve
Soient $V_0, V_1, ..., V_m$ des MaxSubsets tels que $V_1 = next(V_0); V_2 = next(V_1); ...; V_m = next(v_{m-1})$ et $next(V_m)$ n'est pas valide alors $G = V_m$ est le générateur à partir duquel tous les MaxSubsets $V_0, V_1, ..., V_m$ peuvent être déduits. Pour faire cette déduction, il faut procèder récursivement à partir de V_m et calculer V_{i-1} tel que $next(V_{i-1}) = V_i$.

Dans la liste des MaxSubsets calculée par l'algorithme 16, les MaxSubsets stockés entre deux générateurs G_1 et G_2, et tel qu'aucun générateur ne soit aussi entre G_1 et G_2 sont représentés par la liste des générateurs LG par G_2.

■

Calcul des règles 'm parmi N'

Nous allons présenter dans cette sous-section comment obtenir les règles '*m parmi N*' à partir de la liste des générateurs.

Théorème 6 *La liste des règles 'm parmi N' extraites pour un neurone y est fonction de la liste des générateurs extraits de ce neurone.*

Preuve
En effet, un générateur représente plusieurs MaxSubsets et peut s'écrire sous la forme 'm parmi N' résumant les MaxSubsets qui peuvent être calculés à partir de ce générateur et réduire leur nombre en utilisant les propositions 4 et 5 de la section 4.2.

■

Pour obtenir un ensemble de règles de taille plus faible, nous proposons l'algorithme 18. Cet algorithme regroupe les générateurs tels que chaque groupe soit formé des générateurs ayant la même cardinalité (même nombre de positions ayant le valeur 1). Aucun traitement n'est réalisé ici car dans l'algorithme 16, la liste des générateurs LG est générée de telle sorte que les générateurs soient classés par ordre croissant de cardinalité. Pour chaque groupe obtenu, l'algorithme procède par deux étapes pour produire les règles 'm parmi N :

1. Si la distance (de Hamming) entre le générateur courant et le prochain dans la même partition vaut 2, alors seul le deuxième générateur est considéré et le premier ignoré. La distance de Hamming entre deux sous-ensembles dans le même groupe ne peut pas être

impaire (car ils ont le même nombre de positions ayant la valeur 1). Quand cette distance devient supérieure à 2, la règle est stockée dans la liste des règles et le processus continue jusqu'à ce que tous les générateurs de la partition soient traités. L'algorithme 18 présente le pseudo-code de cette étape. Il utilise la procédure $Interpret()$ qui transforme chaque générateur en règle 'm parmi N' équivalente (voir l'algorithme 17). Dans ces algorithmes, l'objet V est le vecteur caractéristique booléen (V_i est son i^e composant). Une règle MofN est composée d'un ensemble de pair (m, N) où m est un entier positif et représente le nombre minimum d'éléments (conditions) de N qui doivent être vérifiées pour que la conclusion prennent la valeur booléenne 1.

2. Cette étape revisite les règles obtenues de chaque partition dans l'étape précédente. Les propositions 4 et 5 sont appliquées au résultats de l'étape 1 pour supprimer les redondances dans les règles et réduire leur nombre.

Algorithme 17 Interpret(V) //Transformation d'un générateur en une règle MofN

Entrées: Un générateur présenté sous forme d'un vecteur binaire V

Sorties: Une règle MofN r correspondante au vecteur V.

1: Initialiser l'indice k à 1 (début du vecteur) ;
2: **Répéter**
3: A partir de k, affecter à N_0 les variables correspondantes à la suite consécutive de bits ayant la valeur 0 ;
4: Ajouter à k la cardinalité de N_0 ;
5: A partir de k, affecter à N_1 les variables correspondantes à la suite consécutive de bits ayant la valeur 1 ;
6: Ajouter à k la cardinalité de N_1 ;
7: Former le composant avec $N = N_0 \cup N_1$ et $m = |N_1|$;
8: Insérer le composant formé dans r si N_1 n'est pas vide
9: **jusqu'à** $N_1 = \emptyset$
10: Retourner r.

Complexité

Supposons qu'il existe n connexions en entrée du neurone considéré. Cette complexité peut être bornée par la taille de la structure $LG = \sum_{i=kmin}^{kmax} \binom{n}{i}$ où $\binom{n}{i} = \frac{n!}{i!(n-i)!}$. Notons aussi qu'il est impossible d'avoir $kmin = 1$ et $kmax = n$ parce que dans le cas où $kmin = 1$, il existera au moins un lien capable d'activer la sortie, par conséquent la cardinalité de l'ensemble des connexions restantes deviendra au maximum $n - 1$. Mais comme cette complexité est facilement exprimable en fonction du résultat (ensemble des MaxSubsets), nous allons la calculer en fonction de la taille de la structure LM :

Algorithme 18 Production des règles m parmi N

Entrées: La liste des générateurs LG ordonnée comme à la sortie de l'algorithme 16

Sorties: Un ensemble R de règles 'm parmi N'.

1: Initialiser i à 1 ;
2: **Tant que** la liste des générateurs n'est pas vide **faire**
3: Tant que $distance(G_i, G_{i+1}) = 2$ faire incrémenter i ;
4: Insérer $Interpret(G_i)$ dans R ;
5: **fin tantque**
6: Appliquer les propositions 4 et 5 à l'ensemble des règles de la section 4.2 ;
7: Retourner R.

1. **Complexité mémoire**
   Un vecteur caractéristique est de taille n bits. Donc les structures de données LM et LG seront respectivement de taille $n \times |LM|$ et $n \times |LG|$. Et comme $LG \subseteq LM$, cette complexité est bornée en $O(n \times |LM|)$.

2. **Complexité en temps**
   Ce temps peut être fonction du temps de construction de LM et de LG. Mais les fonction $new\_next()$ et ***générer()*** s'exécutent en $2 \times n$ parce qu'elles peuventt parcourir le vecteur en aller et retour. Dans l'algorithme 16, le temps de construction des structures LM et LG peut être exprimé en fonction de LM uniquement car $LG \subseteq LM$. Sachant que le temps de recherche d'un élément de la structure est de $2n$ alors le temps d'exécution de cette fonction est bornée par $2n \times |LM|$. La complexité de construction des structures LM et LG est en $O(n \times |LM|)$

4.4.3 Expérimentations

Nous allons tester l'approche des MaxSubsets sur un réseau de neurones appris pour classer les données monks prises dans la base UCI. Le RNA est construit suivant les approches MCLANN [MNTT09] et KBANN [TS94].

Données

Le problème Monks a été utilisé par Thrun et associés [Tea91] pour tester plusieurs modèles d'apprentissage. Le problème Monk's provient de la robotique. Les robots sont décrits avec six attributs ; le tableau 4.1 décrit ces attributs.

Attribut	Valeurs
forme-tête	ronde, carrée, octogone
forme-corps	ronde, carrée, octogone
est-souriant	oui, non
tenir	épée, ballon, drapeau
couleur-veste	rouge, jaune, verte, bleue
a-une-cravate	oui, non

TAB. 4.1 – Les attributs et leurs valeurs du problème Monk's

Trois problèmes de classification sont concernés dans Monk's :

1. Monk1 : si (*forme-tête* = *forme-corps* ou $couleur - veste = rouge$) alors '*Monk1*'.
2. Monk2 : Si exactement deux parmi les six attributs ont la première valeur alors '*Monk2*'.
3. Monk3 : Si ($forme - corps \neq octogone$ et $couleur - veste \neq bleue$) ou ($tenir = peetcouleur - veste = verte$) alors '*Monk3*'.

Ces données sont binarisées avant d'être utilisées pour l'apprentissage du réseau. Les attributs binaires issus du prétraitement sont :

- at_1 =1 si forme-tête = ronde et 0 sinon.
- at_2 = 1 si forme-tête = carrée et 0 sinon ;
- at_3 = 1 si forme-tête = octogone et 0 sinon ;
- at_4 = 1 si forme-corps = ronde et 0 sinon ;
- at_5 = 1 si forme-corps = carrée et 0 sinon ;
- at_6 = 1 si forme-corps = octagone et 0 sinon ;
- at_7 = 1 est-souriant = oui et 0 sinon ;
- at_8 = 1 si tenir = épée et 0 sinon ;
- at_9 = 1 si tenir = ballon et 0 sinon ;
- at_{10} = 1 si tenir = drapeau et 0 sinon ;
- at_{11} = 1 si couleur-veste = rouge et 0 sinon
- at_{12} = 1 si couleur-veste = jaune ;
- at_{13} = 1 si couleur-veste = verte et 0 sinon ;
- at_{14} = 1 si couleur-veste = bleue et 0 sinon ;
- at_{15} = 1 si a-une-cravate = oui et 0 sinon.

Les RNA construits pour le traitement des trois problèmes par la méthode MCLANN et par la méthode KBANN sont présentés dans les tableaux 4.2 et 4.3. Dans ces tableaux, la colonne 'Problème' désigne le problème, '#Entrées' est le nombre de neurones de la couche d'entrée, '#Cachées' le nombre de neurones cachés et '#Sortie' le nombre de neurone en sortie du réseau.

Problème	#Entrées	#Cachés	#Sortie
Monks1	15	12+3	1
Monks2	15	13	1
Monks3	15	13+2	1

TAB. 4.2 – Architecture des RNA utilisés pour l'apprentissage des problèmes Monk's avec la méthode MCLANN

Problème	#Entrées	#Cachés	#Sortie
Monks1	15	4	1
Monks2	15	15	1
Monks3	15	7	1

TAB. 4.3 – Architecture des RNA utilisés pour l'apprentissage des problèmes Monk's et construit grâce à l'approche KBANN

Dans ces expérimentations, nous allons nous intéresser au nombre de règles extraites pour chaque neurone, et au temps nécessaire pour réaliser l'extraction de ces règles.

Résultats

Les tableaux 4.4, 4.5, 4.6 et 4.7 présentent les résultats statistiques obtenus pendant l'extraction des règles des différents neurones présentés dans le tableau 4.2. Dans ces tableaux, $Neurone$ indique le numéro du neurone, $\#Max$ est le nombre de MaxSubsets extraits de ce neurone, $\#Gen$ est le nombre de générateurs, $\#MofN$ est le nombre de règles 'm parmi N' obtenu du neurone ; $Tgen$ et $TMofN$ sont respectivement le temps d'extraction des générateurs (et des MaxSubsets) et celui de la production des règles 'm parmi N' à partir de la liste des générateurs obtenus. Le rapport entre le nombre de générateurs et le nombre de MaxSubsets est représenté à la colonne 'Gen/Max' et le rapport entre le nombre de règles 'm parmi N' et le nombre de générateurs à la colonne 'MofN/Gen'.

Le nombre de générateurs est considérablement plus faible que celui des MaxSubsets. Dans la majorité des cas (à l'exception des neurones ayant une seule entrée), le nombre de générateurs est deux fois plus petit que le nombre de MaxSubsets (compris entre 32 et 56%). Le nombre de

Neurone	#Max	#Gen	#MofN	TGen	TMofN	Gen/Max	MofN/Gen
1	130	51	45	0	0	0,39	0,88
2	242	113	89	1	0	0,46	0,79
3	313	137	83	0	0	0,44	0,60
4	189	93	49	0	1	0,49	0,53
5	324	144	89	0	0	0,44	0,62
6	1	1	1	0	0	1	1
7	1	1	1	0	0	1	1
8	195	85	47	0	0	0,43	0,55
9	89	42	24	0	0	0,47	0,57
10	1	1	1	0	0	1	1
11	396	183	94	1	0	0,46	0,51
12	199	112	62	0	0	0,56	0,55
13	151	57	43	0	0	0,38	0,75
14	318	154	74	0	1	0,48	0,48
15	1	1	1	0	0	1	1
16	1	1	1	0	0	1	1
17	1	1	1	0	0	1	1
18	350	114	53	0	0	0,32	0,46

TAB. 4.4 – Etude statistique de l'application de l'approche des MaxSubsets sur le réseau appris pour le problème Monk1

règles 'm parmi N' représente aussi dans la majorité des cas 40% du nombre des générateurs. Ce rapport montre le gain en espace mémoire qu'on aura à utiliser les générateurs (pour la sauvegarde) que les MaxSubsets.

La complexité théorique en temps de l'approche des MaxSubsets est exponentielle. Cependant le temps d'exécution sur les données Monk's est presque nulle. Ces données (Monk1, Monk2, Monk3) classées à l'aide de l'approche arborescente $PART$ ont permis d'obtenir respectivement 11, 19 et 8 règles. La version de PART utilisée est celle implémentée dans la plate forme Weka [WF05]. Le nombre de règles obtenues de la méthode arborescente PART est plus reduit (comparés au tableaux 4.4, 4.5, 4.6, 4.7) car il s'agit simplement de décrire tous les chemins allant de la racine à chacune de ses feuilles. Par contre, pour le cas des RNA l'extraction des règles implique la description de toutes les combinaisons de poids capables d'activer le neurone.

Unit	#Max	#Gen	#MofN	TGen	TMofN	Gen/Max	MofN/Gen
1	139	59	40	1	0	0,42	0,68
2	175	72	43	0	0	0,41	0,60
3	178	67	50	0	0	0,38	0,75
4	151	49	26	0	0	0,32	0,53
5	6	3	1	0	0	0,5	0,33

TAB. 4.5 – Etude statistique de l'application de l'approche des MaxSubsets sur le réseau construit par l'approche KBANN et appris pour le problème Monk1

Unit	#Max	#Gen	#MofN	TGen	TMofN	Gen/Max	MofN/Gen
1	244	111	54	0	0	0,45	0,49
2	14	9	3	0	0	0,64	0,33
3	718	417	134	1	0	0,58	0,32
4	287	107	85	0	1	0,37	0,79
5	298	101	80	0	1	0,34	0,79
6	770	373	128	1	0	0,48	0,34
7	905	426	182	1	0	0,47	0,43
8	175	58	39	0	0	0,33	0,67
9	83	45	28	0	0	0,54	0,62
10	487	223	107	0	1	0,45	0,48
11	110	49	29	0	0	0,45	0,59
12	239	82	43	0	0	0,34	0,52
13	253	148	64	0	1	0,58	0,43
14	92	42	33	0	0	0,45	0,79
15	432	210	82	0	0	0,49	0,39
16	41	8	5	0	0	0,20	0,63

TAB. 4.6 – Etude statistique de l'application de l'approche des MaxSubsets sur le réseau appris pour le problème Monk2. RNA construit par l'approche MCLANN

Unit	#Max	#Gen	#MofN	TGen	TMofN	Gen/Max	MofN/Gen
1	437	188	121	0	0	0,43	0,64
2	147	55	47	0	0	0,37	0,85
3	3368	146	111	0	0	0,40	0,76
4	259	97	61	0	0	0,37	0,63
5	229	90	83	1	0	0,39	0,92
6	188	77	53	0	0	0,41	0,69
7	234	93	71	0	0	0,40	0,76
8	16	13	5	0	0	0,81	0,38

TAB. 4.7 – Etude statistique de l'application de l'approche des MaxSubsets sur le réseau construit à l'aide de la méthode KBANN et appris pour le problème Monk3

Discussion

Dans cette section, nous allons comparer théoriquement l'approche des MaxSubsets aux précédentes approches [Fu91, CS94, Set01].

Les approches Subset, coaliton-opposition et MaxSubsets recherchent les sous-ensembles de connexions d'entrées capables de rendre active la sortie du neurone. Ce fait rend sa complexité exponentielle ($O(2^n)$).

Malgré le fait que les auteurs de MofN [TS93] montrent que sa complexité est linéaire ou quadratique suivant les facteurs, les étapes de partitionnement, d'élagage et de réapprentissage présentent chacune des inconvénients : suppression des attributs qui suivant l'algorithme d'élagage semble moins pertinent alors qu'il participe à la décision finale, aucune garantie concernant le succès du réapprentissage,... Son résultat est conditionné par le succès de ces opérations (élagage et réapprentissage). En plus il n'extrait que les règles 'm parmi N'.

L'approche des MaxSubsets malgré une complexité théorique qui est exponentielle, présente plusieurs avantages : capacité à produire les deux formes de règles, aucun prétraitement (comme l'approche MofN) n'est effectué sur les entrées et la gestion très efficace de l'espace mémoire. Ce qui permet de vérifier facilement l'équivalence entre les règles (quelque soit la forme) et le réseau initial.

Nous venons de présenter une nouvelle approche d'extraction de règles à partir d'un RNA, une approche permettant d'extraire suivant le choix de l'utilisateur un format de règles parmi les formats 'si alors' et 'm parmi N'. Nous allons présenter dans la section suivante à l'extraction

des règles dans les RNA construits avec la méthode MCLANN.

4.5 Extraction des règles à partir du modèle MCLANN

Nous allons dans cette partie présenter comment interpréter la structure du réseau construit avec la méthode MCLANN présenté dans le chapitre précédent.

4.5.1 Treillis de Galois et extraction des règles

Plusieurs travaux [AS94, PBTL99, BYGMN09] proposent les algorithmes basés sur les treillis de Galois d'extraction des règles d'association à partir des données. Une règle d'association est définie comme une implication entre deux sous-ensemble d'attributs. Nous supposons que les données sont représentées par le contexte formel $C = (O, A, R)$ et que la correspondance de Galois (f, g) est établie entre les éléments de O et de A.

Définition 42 *Soient A_1 et A_2 deux sous-ensembles d'attributs, $A_2 \Leftarrow A_1$ avec $A_1 \cap A_2 = \emptyset$ définie une règle basée sur $A_1 \cup A_2$, A_1 et A_2 sont appelés respectivement prémisse et conclusion de la règle.*

Définition 43 *Le support d'un sous-ensemble d'attributs $A_1 \subseteq A$ noté $support(A_1)$ est $|g(A_1)| \,/\, |O|$*
Le support de la règle $r =' A_2 \Leftarrow A_1'$ noté $support(r)$ est le support de $A_1 \cup A_2$.
La confiance $conf(r)$ de la règle $r =' A_2 \Leftarrow A_1'$ est le rapport $support(A_1 \cup A_2)/support(A_1)$.

Définition 44 *Soient α $(0 < \alpha \leq 1)$ et β $(0 < \beta \leq 1)$ deux nombres fixés par l'utilisateur, une règle r est valide si $support(r) \geq \alpha$ et $conf(r) \geq \beta$.*

La tâche d'extraction des règles consiste à extraire les règles valides. Une décomposition de ce problème proposée dans [AS94] est :

1. Extraction des sous-ensembles fréquents d'attributs.
2. Génération des règles d'association valides basées sur les ensembles fréquents extraits.

L'extraction des ensembles fréquents est faite en construisant un demi-treillis de Galois et par génération successive des successeurs. Sachant que le successeur d'un ensemble non fréquent n'est pas fréquent [AS94], la génération s'arrête quand l'algorithme ne trouve plus de concepts fréquents.

Définition 45 *Soit r une règle, r est une règle exacte si $conf(r) = 1$ et approximative sinon [PBTL99].*

Soit c un concept formel du demi-treillis, les règles exactes sont les implications $Y \Leftarrow X$ où $X \cup Y$ est égale à l'intension de c.

Soient c_1 et c_2 deux concepts formels tels que c_2 est le successeur de c_1, une **règle approximative** est définie comme suit : "$intent(c_2) \Leftarrow intent(c_1)/intent(c_2)$, où $intent(c_i)$ dénote l'intension du concept c_i et la confiance de r est calculée comme le rapport entre les cardinalités : $|extent(c_2)|/(|extent(c_1)|$. Où $|extent(c_i)|$ est la cardinalité de l'extension du concept c_i.

Exemple 14 *A partir du treillis de la figure 3.1 construit avec $'\alpha = 5\%'$, nous pouvons extraire les règles exactes $'e \Leftarrow bcd'$ et $'bcd \Leftarrow e'$ de support 2/8. Nous pouvons aussi extraire la règle approximative $'b \Leftarrow d'$ qui a pour support 3/8 et pour confiance 3/4.*

4.5.2 Application aux RNA construits avec la méthode MCLANN

Les RNA construits avec les méthodes MCLANN et CLANN sont basés sur la structure des demi-treillis ; il est possible d'y extraire un ensemble de règles. Deux sortes de règles peuvent être extraites de cette structure : les règles classiques (si alors ou m parmi N) et les règles approximatives (comme dans les tâches d'extraction des connaissances [PBTL99]). Les règles 'if then' et 'm parmi N' ayant été étudiées par la section précédente, nous allons nous focaliser aux règles approximatives [PBTL99] dans cette section.

Nous transposons cette définition de règles approximatives dans le contexte des RNA. Ces règles présentent plusieurs avantages :

1. Ces règles définissent exactement comment chaque connexion influence l'activation du neurone ;
2. Le nombre de règles est égal au nombre de connexions ; le temps d'extraction est donc linéaire.
3. Il est plus facile de vérifier quel est l'apport d'une entrée sur la décision finale.
4. Dans le cas d'un raffinement de la base des règles, on peut utiliser les règles approximatives pour trier les règles de la base.

Dans les tâches d'extraction des règles à partir des données, la confiance est définie à partir des données. Dans le cadre des RNA, nous allons calculer une grandeur correspondante à

la confiance pour un lien de connexion et nous allons l'appeler $influence$ notée $infl_j$ pour l'influence du lien de connexion j. Cette formule permet de déterminer l'effet exact de chaque entrée du neurone i.

Définition 46 *Etant donné un neurone i, l'influence $infl_j$ d'une connexion j est calculée par l'équation 4.5.*

$$infl_j = 100 * \frac{w_{ij}}{\sum_k |w_{ik}|} \tag{4.5}$$

En utilisant les structures de demi-treillis construits par CLANN et MCLANN les neurones sont étiquettés par les intensions des concepts correspondant. Ces intensions peuvent être plus significatives que chaque attribut pris de manière unique. Cette façon de procéder est importante parce qu'elle permet d'expliquer cette décision juste en regardant la combinaison des intensions de ses successeurs.

Par exemple, soit un neurone représentant un concept c de la première couche cachée et qui a pour successeurs les neurones représentant les concepts ayant pour intension $intent_i$ et connectés à c_i avec un poids w_i. Si $|w_i| \ll |w_j|$ alors nous pouvons expliquer une classe cl par les effets collectifs des attributs $intent_i$.

Illustration 13 *Pour illustrer l'extraction de ce nouveau type de règles, nous allons extraire les règles approximatives à partir du RNA de la figure 4.2. A partir de ce RNA, les règles sont les implications et leurs influences suivantes :*

- *Couche de sortie :*
  1. $cl_1 \leftarrow a, -39,92\%$ *;* $cl_1 \leftarrow b, -13,91\%$ *;*$cl_1 \leftarrow c, -18,93\%$ *;* $cl_1 \leftarrow d, -22,21\%$ *;* $cl_1 \leftarrow f, 5,03\%$;
  2. $cl_2 \leftarrow a, 8,80\%$ *;* $cl_2 \leftarrow b, -13,59\%$ *;* $cl_2 \leftarrow c, -18,49\%$ *;* $cl_2 \leftarrow d, -22,90\%$ *;* $cl_2 \leftarrow f, -36,22\%$ *;*
  3. $cl_3 \leftarrow a, 7,06\%$ *;* $cl_2 \leftarrow b, 14,37\%$ *;* $cl_2 \leftarrow c, 31,53\%$ *;* $cl_3 \leftarrow d, 32,88\%$ *;* $cl_3 \leftarrow f, -14,16\%$ *;*
- *Deuxième couche cachée :*
  1. $a \leftarrow b, -85,86\%$ *;* $a \leftarrow d, 14,18\%$ *;*
  2. $b \leftarrow a, -22,08\%$ *;* $b \leftarrow c, -38,69\%$ *;* $b \leftarrow d, -39,23\%$ *;*
  3. $c \leftarrow b, -53,92\%$ *;* $c \leftarrow d, -46,08\%$ *;*

4. $d \leftarrow a, 6,05\%$; $d \leftarrow b, -46,82\%$; $d \leftarrow c, -39,07\%$; $d \leftarrow f, -8,061\%$;
5. $f \leftarrow d, -100\%$;

– *Première couche cachée, nous n'avons pas supprimé l'intersection de la conclusion et de la prémisse parce que la prémisse désigne une entrée. En supprimant on perdra la sémantique de la règle.*

1. $ab \leftarrow a, -17,02\%$; $ab \leftarrow b, 14,27\%$; $ab \leftarrow c, 1,23\%$; $ab \leftarrow d, -2,13\%$; $ab \leftarrow e, 61,46\%$; $ab \leftarrow f, -3,89\%$;
2. $ad \leftarrow a, 5,19\%$; $ad \leftarrow b, -19,71\%$; $ad \leftarrow c, -12,59\%$; $ad \leftarrow d, -1,94\%$; $ad \leftarrow e, -45,74\%$; $ad \leftarrow f, -14,82\%$;
3. $bc \leftarrow a, -5,83\%$; $bc \leftarrow b, 13,70\%$; $bc \leftarrow c, 3,72\%$; $bc \leftarrow d, -11,91\%$; $bc \leftarrow e, 36,03\%$; $bc \leftarrow f, 28,81\%$;
4. $bd \leftarrow a, -5,71\%$; $bd \leftarrow b, 13,67\%$; $bd \leftarrow c, 2,01\%$; $bd \leftarrow d, -12,58\%$; $bd \leftarrow e, 36,35\%$; $bd \leftarrow f, 29,72\%$;
5. $cd \leftarrow a, -4,45\%$; $cd \leftarrow b, 14,10\%$; $cd \leftarrow c, 5,36\%$; $cd \leftarrow d, -9,25\%$; $cd \leftarrow e, 35,73\%$; $cd \leftarrow f, 31,11\%$;
6. $df \leftarrow a, 10,06\%$; $df \leftarrow b, -8,94\%$; $df \leftarrow c, -8,65\%$; $df \leftarrow d, -7,53\%$; $df \leftarrow e, -35,60\%$; $df \leftarrow f, 29,22\%$.

A partir de ces règles, nous constatons que les neurones de la première couche cachée peuvent être influencés positivement ou négativement par n'importe quelle entrée. Cette influence (dûe à la modification des poids par apprentissage) peut être indépendante de l'appartenance de l'attribut à son intension. C'est le cas de l'attribut e qui bien que n'appartenant à aucune intension a la plus grande influence sur tous les neurones de la première couche cachée.

De même dans la deuxième couche, on observe aussi la domination de certaines connexions où les neurones correspondants aux concepts ayant pour intension bd et cd dominent les neurones correspondants aux concepts ayant pour intension ad et df. C'est le cas des entrées de nœud ayant pour intension d.

Nous avons présenté dans cette section comment l'activation d'un neurone est influencée par ses entrées. Cela nous permet de connaître comment le sous-ensemble d'attributs correspondant à l'intension du concept représenté par un neurone participe à la sortie du RNA. Nous allons dans la section suivante proposer comment expliquer un résultat en sortie du RNA lorsqu'un exemple donné est placé en entrée.

4.6 Explication des résultats du RNA

Dans cette section, nous expliquons comment le résultat du RNA a été calculé en recherchant le chemin constitué des neurones actifs allant de la sortie aux entrées. Il est question pour nous de trouver la relation entre les règles approximatives contenues dans la structure du demi-treillis et les différentes activations des neurones du RNA pour un exemple. Nous pouvons justifier le résultat proposé par le RNA à propos de cet exemple par la combinaison des différents neurones actifs (comme illustre le schéma 4.4). Nous nous intéressons aux règles approximatives parce qu'elles sont celles qui existent entre un concept successeur et un de ses prédécesseurs car les règles exactes ne sont qu'entre les attributs de l'intension d'un concept.

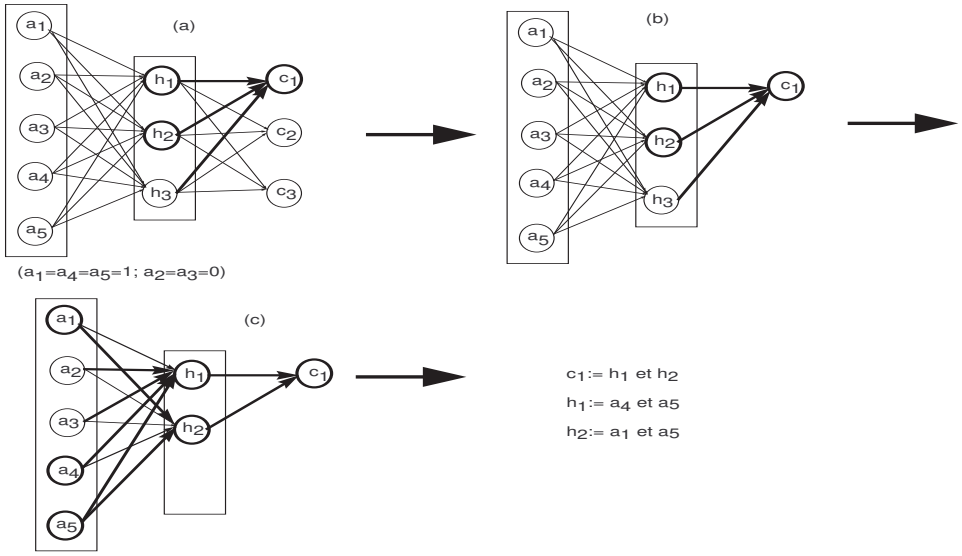

FIG. 4.4 – Processus d'explication d'un résultat en sortie du RNA.

Pour cela, nous allons observer leurs successeurs et connaître lesquels s'activent et influencent l'affection d'une classe à un objet donné. L'explication d'un résultat suit les étapes suivantes :

1. Positionner l'exemple à l'entrée du RNA (figure 4.4(a)) ;
2. Calculer la sortie du réseau (figure 4.4(a)) ;
3. Observer le chemin allant de la sortie active vers les entrées ; ce chemin est constitué des neurones actifs. Dans les schémas 4.4 (b) et (c), ces neurones (actifs) ainsi que les connexions excitatrices sont marqués en gras. Nous pouvons alors ignorer les neurones inactifs et expliquer l'activation d'un neurone juste par des entrées actives.

4. Expliquer le résultat ; elle se fait en partant de la sortie et expliquant l'activité d'un neurone par la combinaison des ses entrées actives.

L'algorithme 19 présente le pseudo-code de la procédure d'explication d'un résultat du RNA.

Algorithme 19 Processus d'explication d'un résultat calculé par le RNA

Entrées: Un RNA et un exemple

Sorties: Une explication du résultat de RNA obtenu sur l'exemple.

1: Positionner l'exemple en entrée du RNA ;
2: Propager le signal dans le RNA et calculer la sortie du RNA ;
3: Pour chaque couche allant de la sortie aux entrées, sélectionner les neurones actifs ;
4: Expliquer le résultat ;

Illustration 14 *Soit un RNA construit par la méthode MCLANN (en choisissant comme contrainte la hauteur du demi-treillis 1) pour apprendre les données présentées dans le contexte du tableau 3.1. En faisant de l'apprentissage de ce RNA jusqu'à 0% de taux d'erreur, puis en classant les mêmes exemples, les observations faites sur les neurones sont décrites dans le tableau 4.8. La colonne "Objet" est le numéro de l'objet, la colonne "Sortie" est sa classe (obtenue du RNA car ce dernier a été appris pour ne pas faire d'erreur sur ces données), "Caché" l'intension du concept correspondant au neurone activé dans la couche cachée (pour ce cas, il y a une seule couche cachée car la contrainte appliquée lors de la construction du demi-treillis est la hauteur 1) et "Entrées" les neurones d'entrée (attributs) qui ont la valeur 1. De ce tableau, nous pouvons faire les observations suivantes :*

- *Les exemples 1 et 5 appartenant à la classe cl_1 (aérien) activent (lorsqu'ils sont placés en entrée du RNA) seulement le neurone correspondant au concept ayant pour intension b (i.e poids≤40). Ces objets ont en commun les attributs b, c, d, e ; mais la propagation des ces attributs ne permet pas d'activer les neurones correspondants aux concepts ayant les intensions c, d. On peut alors penser que l'exemple est classé cl_1 parce que le neurone b s'est activé. De manière litérale, cela signirait : "ces animaux sont classés 'aérien' parce que leurs poids sont inférieurs à 40 kilogrammes".*
- *Les exemples 6 et 7 de la classe cl_2 (aquatique) activent le neurone correspondant au concept ayant pour intension f (l'animal peut nager). Ce qui corrobore avec les attributs, puisque ces objets n'ont en commun que l'attribut f. Nous pouvons expliquer de manière litérale par : "ces animaux sont classés 'aquatique' parce qu'ils peuvent nager".*
- *Les exemples 3 et 4 de la classe cl_3 (terrestre) activent les neurones correspondant aux concepts d'intension b (poids≤40) alors qu'en commun, ils n'ont que l'attribut d. Egale-*

Objet	Sortie	Caché	Entrées
1	cl_1	b	a,b,c,d,e
2	cl_3	b	a,b,d
3	cl_3	d	a,d
4	cl_3	d	d
5	cl_1	b	b,c,d,e
6	cl_2	f	d,f
7	cl_2	f	a,b,c,f
8	cl_3	-	c,d

TAB. 4.8 – Observations faites sur le RNA appris pour classer les données du tableau 3.1 ; le RNA a une couche cachée.

ment, l'interprétation de cette observation peut être : "ces animaux sont classés 'terrestre' parce qu'ils peuvent marcher".

– *L'exemple 2 apporte une contradiction sur les observations faites au sujet des exemples 1 et 5 car il active le même neurone b. Mais les apports des autres neurones cachés l'orientent vers la classe cl_3.*

Illustration 15 *Soit le RNA de la figure 4.2 construit (avec comme contrainte, hauteur du demi-treillis égale à 2) pour apprendre les données présentées dans le contexte du tableau 3.1. L'apprentissage de ce RNA s'est fait jusqu'à 0% de taux d'erreur ; en classant les mêmes données, les observations faites sont décrites dans le tableau 4.9.*

La première observation est que pour tous les objets présentés en entrée, les neurones de la première couche restent inactifs à l'exception d'un seul cas (objet 3). Mais les valeurs internes de ces neurones sont plus proches de zéro, et non nulle car la fonction d'activation est un sigmoïde. Les observations faites sur l'activité des neurones de la deuxième couche cachée sont :

– *Les objets 1 et 5 appartenant à la classe cl_1 activent les mêmes neurones : neurones correspondant aux concepts ayant pour intension ab,bc,bd,cd. L'activation de ces neurones de la couche cachée 2 est très fortement influencée par le neurone d'entrée e.*
– *Les objets 6 et 7 appartenant à la classe cl_2 activent également les mêmes neurones : ces neurones correspondent aux concepts ayant pour intension bc,bd,cd,df. L'activation de ces neurones est aussi très fortement influencée par le neurone d'entrée f.*

Objet	Sortie	Caché 2	Caché 1	Entrées
1	cl_1	-	ab,bc,bd,cd	a,b,c,d,e
2	cl_3	-	cd	a,b,d
3	cl_3	a,d	ad,df	a,d
4	cl_3	-	-	d
5	cl_1	-	ab,bc,bd,cd	b,c,d,e
6	cl_2	-	bc,bd,cd,df	d,f
7	cl_2	-	bc,bd,cd,df	a,b,c,f
8	cl_3	-	-	c,d

TAB. 4.9 – Observations faites sur le RNA appris pour classer les données du tableau 3.1 ; le RNA a deux couches cachées.

- *Les objets 4 et 8 de la classe cl_3 n'activent aucun neurone de la couche cachée. Ces deux objets ont en commun la variable d'entrée d (variable commune à tous les objets de la classe cl_3.*
- *Les activations des neurones faites par les objets 2 et 3 de la classe cl_3 sont totalement différentes.*

En sélectionnant (avec une restriction aux classes cl_1 et cl_2) les intensions des concepts correspondant aux neurones qui se sont activés exclusivement pour une classe donnée, on déduit :

- $cl_1 \leftarrow ab$ *ce qui peut se traduire litéralement de la manière suivante : "ces animaux sont classés 'aérien' parce qu'ils a une taille inférieure à un mètre et un poids inférieur à 40 kilogrammes".*
- $cl_2 \leftarrow df$ *; la traduction est : "ces animaux sont classés 'aquatique' parce qu'ils peuvent marcher et nager".*

Pour un exemple à classer, le chemin allant du neurone actif en sortie vers les entrées en passant par les neurones cachés expliquent le résultat que le RNA propose pour cet exemple. Le résultat peut être expliqué par une combinaison linéaire des neurones actifs de la couche k, ou de continuer à la couche $k-1$. Cette explication permet aux utilisateurs de comprendre comment ce résultat a été calculé et pouvoir donner une interprétation au comportement du RNA.

4.7 Conclusion

Nous avons dans ce dernier chapitre abordé l'aspect explicatif des réseaux de neurones. Cet aspect est réalisé par l'extraction des règles à partir du RNA après apprentissage. Nous avons présenté les approches principales présentées dans la littérature pour effectuer cette tâche. Ces approches sont Subset et coalition-opposition (pour l'extraction des règles 'si alors') et MofN (pour l'extraction des règles 'm parmi N').

La nouvelle méthode (appelée approche des MaxSubsets) que nous avons développée a été aussi présentée. Cette approche présente plusieurs avantages : capacité de produire les deux formes de règles, sans aucun prétraitement. Nous avons aussi montré comment extraire les règles à partir des RNA construits par MCLANN. Nous avons enfin proposé une méthode pour expliquer le résultat mis en sortie du RNA pour un exemple donné.

Le choix du format des règles extraites d'un RNA est toujours problématique. La suite sera d'étudier ces formats, de mener d'autres expériences sur les différents formats afin de trouver les conditions d'utilisation et les conditions de validation de chacun.

Conclusion et perspectives

Conclusion

L'utilisation des réseaux de neurones artificiels suscite deux problèmes majeurs :

1. La définition de son architecture. Elle implique choisir certains paramètres : nombres de couches, nombre de neurones par couche, connexions entre les neurones, etc.
2. L'interprétabilité du RNA. Elle consiste à faciliter la compréhension du calcul des résultats. Les RNA sont souvent traités de 'boîte noire'.

Les objectifs principaux de ce travail étaient de trouver un moyen pour construire les RNA interprétables pour la résolution des problèmes de classification en absence de connaissances apriori et de contribuer à l'interprétation du réseau construit.

Pour réaliser ces objectifs, nous avons d'abord fait une présentation des différentes méthodes de classification : les arbres de décision, les k-plus proches voisins, les réseaux bayésiens et les SVM. Dans l'optique de présenter de manière détaillée la technique des RNA, nous avons défini les concepts utilisés dans ce domaine. Les algorithmes d'apprentissage du perceptron, Pocket with rachet modification (pour l'apprentissage des unités neuronales) et l'algorithme de rétropropagation de l'erreur (pour l'apprentissage des réseaux multicouches) ont été aussi décrits.

Nous avons ensuite présenté notre contribution à la recherche des architectures des réseaux de neurones pour la résolution des problèmes de classification supervisée. Cette contribution a été faite sur plusieurs points :

1. Une étude des méthodes existantes de construction ;
2. Une proposition d'une approche (CLANN) de construction à partir des treillis de Galois des RNA pour la résolution d'un problème à deux classes ;

3. Une extension de la méthode précédente aux problèmes à plusieurs classes (MCLANN) ;
4. Une proposition d'une approche d'extraction des règles à partir des réseaux de neurones ; nous avons aussi montré comment expliquer le résultat en sortie du RNA concernant un exemple.

Dans la première contribution, nous avons étudié et classé les méthodes de construction des RNA en plusieurs catégories :

– les méthodes statiques qui construisent le RNA, puis fait son apprentissage sans modifier son architecture. Parmi ces méthodes, KBANN qui définit cette architecture à partir d'un ensemble de connaissances ; la méthode ad'hoc qui construit un RNA à trois couches et dont le nombre de neurones cachées est une fonction des nombres de neurones en sortie et en entrée. Les RNA ainsi construits sont appris par l'algorithme de rétropropagation de l'erreur.
– les méthodes dynamiques qui de manière incrémentale ajoutent les neurones (ou les couches) à un RNA existant et apprennent ses poids afin de le rendre capable de bien classer les exemples ; dans cette catégorie, les méthodes MTiling, MTower, MUpstart, Perceptron-Cascade, MPyramid construisent les réseaux ayant plusieurs couches cachées alors la méthode Distal contruit les réseaux ayant une couche cachée.
– les méthodes génétiques qui recherchent l'architecture en faisant évoluer une population de RNA par des opérations génétiques de croisement et de mutation.

Nous avons proposé une étude théorique et expérimentale de ces méthodes. A la fin nous avons comparé les résultats expérimentaux de ces méthodes à ceux des autres méthodes. Les limites de ces approches (absence de connaissances, opacité du RNA construit) ont été relévées.

Dans le but de dépasser ces limites, nous avons proposé dans le troisième chapitre deux méthodes de construction des RNA à partir des treillis de Galois : CLANN et MCLANN. CLANN est utilisé pour la construction des RNA capables de résoudre les problèmes à deux classes, il s'agit de la deuxième contribution de ce travail. Pour des problèmes à plusieurs classes, MCLANN a été proposé. Cette troisième contribution étend la méthode CLANN à la résolution des problèmes à plusieurs classes. Ces deux approches construisent d'abord un demi-treillis de Galois à partir des exemples. Les contraintes sont introduites à la construction de ce demi-treillis pour réduire le temps et l'espace mémoire nécessaire. Ce demi-treillis de Galois est ensuite transformé en architecture de RNA et le réseau obtenu est appris par rétropropagation. L'architecture de RNA obtenu est interprétable, chaque neurone correspondant à un ensemble d'objets ayant en commun un ensemble d'attributs (concept formel) et chaque lien de connexion entre les neurones correspond à la relation de précédence entre les concepts. Les approches CLANN et MCLANN ainsi proposées construisent des RNA interprétables en absence de connaissances.

Elles peuvent donc être utilisés quand l'utilisateur ne dispose pas de connaissances apriori.

Les connaissances incoporées dans le réseau ont fait l'objet de la quatrième contribution de ce travail. Nous avons étudié les algorithmes d'extraction des règles à partir d'un RNA présents dans la littérature. Face aux limites que présentaient ces algorithmes, nous avons proposé une nouvelle approche d'extraction de règles appelée "approche des MaxSubsets". Elle consiste pour chaque neurone à extraire une structure intermédiaire à partir de laquelle les règles sous forme "si alors" et sous forme "m parmi N" peuvent être déduites. Cette nouvelle méthode permet d'extraire les règles "si alors" et "m parmi N" sans appliquer de prétraitements. Dans ce chapitre, nous avons aussi montrer comment expliquer les résultats que le RNA (construit par MCLANN) met en sortie pour un exemple placé en entrée. Cette explication considère pour un exemple classé par le RNA justifie la classe prédite par le réseau comme résultant de la combinaison des différents neurones actifs.

Perspectives

A l'issue de ce travail, les perspectives suivantes peuvent être envisagées :

1. Nous allons étendre l'approche MCLANN aux données multivaluées et aux données complexes. En effet CLANN et MCLANN construisent des demi-treillis à des contextes binaires. Ils nécessitent alors une phase de prétraitement consistant à binariser les données avant de les utiliser ; alors que pour le même problème les exemples peuvent présenter les attributs binaires, symboliques, numériques ou autres. De nombreux travaux [MDNST08, KAMN10] proposent les algorithmes de construction de treillis de Galois à partir des contextes multivalués.
2. CLANN et MCLANN ont été testés sur les données disponibles sur internet. Il serait aussi important de tester leurs comportements sur des problèmes réels. L'aide d'un expert du domaine pourra aider à sélectionner et à valider les règles extraites d'un RNA construits par ces approches, à justifier l'explication que nous donnons au classement d'un exemple.
3. La connectivité des neurones dans les modèles construits par CLANN et MCLANN est celle obtenue du demi-treillis ; afin de définir l'influence des autres neurones que ceux correspondants aux successeurs dans la couche sous jacente, une connexion complète entre les neurones adjacentes pourra être étudiée.
4. L'approche des MaxSubsets pour l'extraction des règles n'a pas été testée à des réseaux de grande taille. Une étude expérimentale et comparative aux méthodes existantes pourra être menée afin de dégager quelles sont les conditions favorables à son utilisation. De même une validation des règles extraites par cette technique devra être étudiée afin de définir les conditions d'une bonne utilisation et quel format de règles conseillé aux utilisateurs.
5. Enfin, il reste à explorer l'application des treillis de Galois à la construction des architectures de RNA pour la résolution des problèmes de classification non supervisée. En effet les treillis de Galois ont été appliqués en classification non supervisée [DC02]. On peut alors explorer son utilité dans l'initialisation de la structure d'un modèle de Kohonen.

Annexe

Le programme exécutable de MCLANN (pour l'environnement Linux) peut être téléchargé à l'URL :"http ://www.isima.fr/ mephu/FILES/MCLANN/".

Le dossier décompressé contient deux sous dossiers : mclanncv pour la version de MCLANN s'exécutant en validation croisée d'ordre 10 et mclann pour une version simple de MCLANN.

Le dossier mclanncv

Ce dossier contient les trois versions suivantes du programme :

- Mclanncv_level : la contrainte de sélection des concepts est la hauteur.
- Mclanncv_freq : la contrainte de sélection des concepts est la fréquence.
- Mclanncv_freqlev : la contrainte de sélection des concepts est la combinaison de la fréquence et de la hauteur.

Paramètres

Pour exécuter mclanncv, l'utilisateur devrait entrer les paramètres suivants :

1. Nombre d'objets : il s'agit du nombre total d'exemples.
2. Nombre d'attributs : nombre d'attributs utilisés pour décrire chaque exemple. Il s'agit du nombre de colonnes décrivant les exemples.
3. Nombre de classes : nombre de classes en sortie. C'est le nombre de colonnes de données utilisées pour représenter le vecteur de classe.
4. Valeur de la contrainte : pour la contrainte hauteur, taper le nombre correspondant et pour la contrainte fréquence, taper le pourcentage.
5. Nombre d'itérations au cours de l'apprentissage : il s'agit du nombre d'itérations à faire au cours de l'exécution de l'algorithme de rétropropagation de l'erreur.
6. Nom du fichier de données : c'est le nom du fichier contenant tous les exemples. C'est ce fichier qui est divisé pour la validation croisée.
7. Nom du fichier d'apprentissage : c'est le nom du fichier qui contiendra la partie des données réservée à l'apprentissage.

8. Nom du fichier de test : c'est le nom du fichier qui contiendra la partie des données réservée au test.
9. Nom du fichier résultat : c'est le fichier dans lequel les résultats seront enregistrés.
10. Nom du fichier des erreurs : c'est le fichier dans lequel les erreurs de test pendant l'apprentissage sont enregistrées.

Ces paramètres sont saisis l'un après l'autre au clavier. Après la saisie du dernier parmètre, MCLANN est lancé et peut durer quelques minutes (en fonction des données). Après cette exécution, les résultats sont affichés à l'écran et enregistrés également dans le fichier des résultats.

A l'exception du fichier de données, le contenu des autres fichiers est remplacé pendant l'exécution s'ils existent déjà.

Le dossier mclann

Le dossier mclann contient les mêmes progammes que mclanncv. A l'exception du fichier de données, les paramètres sont les mêmes. Mais pour exécuter MCLANN, les fichiers d'apprentissage et de test doivent exister et contenir respectivement les exemples d'apprentissage et les exemples de test. Si les fichiers de résultat et des erreurs existent déjà, leurs contenus seront remplacés.

Pendant l'exécution, l'évolution de l'erreur d'apprentissage est affichée à l"ecran et à la fin les résultats du réseau sur les exemples de test sont affichés.

Résultats

Les résultats sont obtenus à chaque itération de la validation croisée et à la fin de la validation croisée.

1. A la fin de chaque itération de la validation croisée, le taux de généralisation de cette itération est calculé et inséré dans le fichier résultat. De même les temps respectifs de construction, d'apprentissage et test du modèle sont inclus dans ce fichier.
2. A la fin de la validation croisée, le taux général de classification est aussi calculé et inséré dans le fichier résultat. Chaque ligne du fichier résultat contient l'intitulé du paramètre résultat suivi de sa valeur.

Format du fichier des données

Chaque ligne représentant un exemple dans ce fichier est formé d'un vecteur binaire (0, 1). Les valeurs sont séparées par l'espace. La première partie de la ligne décrit l'exemple et la deuxième partie sa classe. Par exemple, l'exemple (x=0001101 ; y=010) s'écrira 0 0 0 1 1 0 1 0 1 0.

Index

Bibliographie

[ADT95] R. ANDREWS, J. DIEDERICH, et A. TICKLE. « Survey and critique of techniques for extracting rules from trained artificial neural networks ». *Knowledge-Based Systems*, 8(6) :373–389, 1995.

[AS94] R. AGRAWAL et R SRIKANT. « Fast algorithms fo mining association rules in large databases ». Dans *Proc VLDB conference*, pages 478–499, 1994.

[Bad09] S. BADER. « Extracting Propositional Rules from Feed-forward Neural Networks by Means of Binary Decision Diagrams ». Dans Artur S. d'Avila GARCEZ et Pascal HITZLER, éditeurs, *Proceedings of the 5th International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'09, held at IJCAI-09*, pages 22 – 27, Pasadena, USA, JUL 2009.

[Bar93] A. BARRON. « Universal Approximation bounds for superposition of a sigmoidal function ». *IEEE Transactions on Information Theory*, 39 :930–945, 1993.

[BBR03] J. F. BOULICAUT, A. BYKOWSKI, et C. RIGOTTI. « Free-sets : a condensed representation of boolean data for the approximation of frequency queries ». *Data Mining and Knowledge Discovery*, 7(1) :5–22, 2003.

[Ben06] Y. BENNANI. *Apprentissage Connexionniste*. Hermès science, 2006.

[Ber58] G. A BERNARD. « Studies in the History of Probability and Statistics : IX. Thomas Bayes's Essay Towards Solving a Problem in the Doctrine of Chances ». *Biometrika (biographical remarks)*, 45 :293–295, 1958.

[BH88] E. B. BAUM et D. HAUSSLER. « What size net gives valid generalization ? ». *Neural Computation*, 1 :151–160, 1988.

[BHME07] S. BADER, S. HOLLDOBLER, et V. MAYER-EICHBERGER. « Extracting Propositional Rules from Feed-forward Neural Networks : A New Decompositional Approach ». Dans CEUR WORKSHOP, éditeur, *IJCAI-07 Workshop on Neural-Symbolic Learning and Reasoning, NeSy'07*, volume 230, 2007.

[Bir40] G. BIRKHOFF. *Lattice Theory,1st edn*. American Mathematical Society, Providence, RI., 1940.

[Bir67] G. BIRKHOFF. *Lattice Theory,3rd edn*. American Mathematical Society, Providence, RI., 1967.

[BM70] M. BARBUT et B. MONJADET. *Ordre et Classification*. Hachette Paris, 1970.

[Bor86] J. BORDAT. « Calcul pratique du treillis de Galois d'une correspondance ». *Mathématiques, Informatiques et Sciences Humaines*, 24 :31–47, 1986.

[Boz95] O. BOZ. « Knowledge integration and rules Extraction in neural networks ». Rapport technique, EECS Lehigh University, 1995.

[BYGMN09] S. BEN YAHIA, G. GASMI, et E. MEPHU NGUIFO. « A new generic basis of factual and implicative association rules ». *Intelligent Data Analysis (IDA)*, 13 :633–656, 2009.

[BYHMN06] S. BEN YAHIA, T. HAMROUNI, et E. MEPHU NGUIFO. « Frequent closed itemset based algorithms : A thorough structural and analytical survey ». *In acm sigkdd explorations*, pages 93–104, 2006.

[BYJ01] S. BEN YAHIA et A. JAOUA. « Discovering knowledge from fuzzy concept lattice ». *In A. Kandel, M. Last, and H. Bunke, editors, Data mining and Computational Intelligence, Studies in Fuzziness and Soft Computing*, 68 :167–190, 2001.

[CH67] T. M. COVER et P. E. HART. « Nierest neigbor pattern classification ». *IEEE Transactions On Information Theory*, 13(1) :21–27, 1967.

[CM10] A. CORNUEJOLS et L. MICLET. *Apprentissage Artificiel : Concepts et algorithmes*. Eyrolles, 2ieme édition, 2010.

[CO02] D. CURRAN et C. O'RIORDAN. « Applying Evolutionary Computation to designing Neural Networks : A study of the State of the art ». Rapport technique, 2002.

[Cra96] M. W. CRAVEN. *Extracting comprehensible models from trained neural networks*. University of Wisconsin, PHD Thesis, 1996.

[CS94] M. W. CRAVEN et J. W. SHAVLIK. « Using sampling and Queries to extract rules from trained neural networks ». Dans *11th International Conference on Machine Learning*, volume 1, 1994.

[CS97] M. W. CRAVEN et J. W. SHAVLIK. « Using neural networks for data mining ». *Future Generation Computer Systems*, 13(2-3) :211–229, 1997.

[CTT92] M. COSNARD, M. TCHUENTE, et G. TINDO. « Sequences generated by neuronal automata with memory ». *Complex Systems*, 6 :13–20, 1992.

[Cun85] Y. Le CUN. « Une Procédure d'Apprentissage pour Réseau à Seuil Asymétrique ». Dans *Cognitiva 85 : A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, pages 599–604, 1985.

[Cyb89] G. CYBENKO. « Approximation by superpositions of a sigmoidal function ». *Mathematics of Control, Signal and Systems*, 2 :303–314, 1989.

[Dar01] A. DARBARI. « Rule extraction from trained ANN : A Survey ». Rapport technique, Institute of Artificial Intelligence, Dept. of Computer Science, TU Dresden, Germany, 2001.

[DC02] N. DURAND et B. CREMILLEUX. « ECCLAT : a New Approach of Clusters Discovery in Categorical Data ». Dans *In the 22nd Int. Conf. on Knowledge Based Systems and Applied Arti Intelligence (ES'02*, pages 177–190. Springer, 2002.

[DFM08] R. DELOGU, A. FANNI, et A. MONTISCI. « Geometrical synthesis of MLP neural networks ». *Neurocomputing*, 71 :919–930, 2008.

[DGBG01] A. S. D'AVILA GARCEZ, K. BRODA, et D. M. GABBAY. « Symbolic knowledge extraction from trained neural networks : a sound approach ». *Artificial Intelligence*, 125 :155–207, 2001.

[DJ99] W. DUCH et N. JANKOWSKI. « Survey of neural tranfer functions ». *Neural computing Surveys*, 2(1-2) :163–212, 1999.

[DN93] E. DAVALO et P. NAIM. *Des réseaux de Neurones*. Eyreolles, 1993.

[Dom02] F. A. DOMENACH. *Structures latticielles, correspondances de Galois, contraintes et classification symbolique*. Thèse de Doctorat, Université de Paris 1 Panthéon - Sorbonne, 2002.

[DP02] B.A. DAVEY et H. A PRIESLEY. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, 2nd édition, 2002.

[DSM+02] G. DREYFUS, M. SAMUELIDES, J. M. MARTINEZ, M. B. GORDON, F. BADRAN, S. THIRIA, et L. HÉRAULT. *Réseaux de Neurones : Méthodologie et applications*. Eyrolles, 2002.

[DSZ04] W. DUCH, R. SETIONO, et J. Z. ZURADA. « Computational intelligence methods of rule-based data understanding ». *Proc of the IEEE*, 92(5) :771–805, 2004.

[EFP08] D. ENDRES, P. FÖLDIÁK, et U. PRISS. « Application of Formal Concept Analysis to Neural Decoding ». Dans SPRINGER-VERLAG, éditeur, *Proceeding of the Concepts Lattice and Applications CLA'2008, LNCS*, 2008.

[Fah88] S. E. FAHLMAN. « Faster-learning variations on Back-propagation : an ampirical study ». Dans *the 1988 Connectionist Models Summer school*, 1988.

[FB97] E. FIESLER et R. BEALE. *Handbook of Neural Computation*. Iop Publishing and Oxford University Press, 1997.

[FFNMN04] H. FU, H. FU, P. NJIWOUA, et E. MEPHU NGUIFO. « Comparative study of FCA-based supervised classification algorithms ». Dans *proceeding of 2nd ICFCA, Sydney*, pages 313–320, 2004.

[FH95] E. FIX et J.L. HODGES. « Discriminatory analysis, nonparametric discrination : Consistency properties ». *The VLDB Journal*, pages 478–489, 1995.

[FL90] S. FAHLMAN et C. LEBIERE. « The Cascade Correlation Procedure Learning Architecture ». *Neural Information systems Touretzky*, 2 :524–532, 1990.

[FMN04] H. FU et E. MEPHU NGUIFO. « Etude et conception d'algorithmes de génération de concepts formels ». *Ingénierie des Systèmes d'Information(Revue ISI)*, 9(3-4) :109–132, sep 2004.

[Fre92a] M. FREAN. « A Thermal Perceptron Learning Rule ». *Neural computation*, 4 :946–957, 1992.

[Fre92b] M. FREAN. « The Upstart algorithm : A method for constructing and training feed forward neural networks ». *Neural computation*, 4 :198–209, 1992.

[FS92] J. A. FREEMAN et D. M. SKAPURA. *Neural Networks : Algorithms, Applications and Programming Techniques*. Addison Wesley, 1992.

[Fu91] L. FU. « Rule Learning by Searching on Adapted Nets ». Dans *Proceedings of the 9th National Conference on Artificial. Intelligence, Anaheim, CA : AAAI Press*, pages 590–595, 1991.

[Gal88] S. GALLANT. « Connectionist expert systems ». *Communication of the ACM*, 31(2) :235–247, 1988.

[Gal90] S. GALLANT. « Perceptron - Based learning Algorithms ». *IEEE Transactions On Neural networks*, 1 :179–191, 1990.

[GD86] J. L. GUIGUES et V. DUQUENNE. « Familles minimales d'implications informatives resultant d'un tableau de données binaires ». *Mathématiques et sciences sociales*, 95 :5–18, 1986.

[Gos96] B. GOSSELIN. « Multilayer Perceptrons Combination Applied to Handwritten Character Recognition ». *Neural Processing Letters*, 3 :3–10, 1996.

[GR99] B. GANTER et R.WILLE. *Formal Concepts Analysis : Mathematical foundations*. Springer - Verlag, 1999.

[Gue91] A. GUENOCHE. « Construction des treillis de Galois d'une relation binaire ». *Mathématiques et Sciences humaines*, 109 :35–47, 1991.

[Hay99] S. HAYKIN. *Neural networks. A comprehensive foundations*. Upper Saddle River, Prentice Hall, 1999.

[HE06] E. R. HRUSCHKA et N. F. F. EBECKEN. « Extracting rules from multilayer perceptrons in classification problems : A clustering-based approach ». *Neural Computation*, 70(1-3) :348–397, 2006.

[Heb49] D. O. HEBB. *The Organization of Behavior*. Lawrance Erlbaum Associates, 1949.

[HK01] J. HAN et M. KAMBER. *Datamining : Concepts and Techniques*. Morgan Kauffman Publishers, 2nde édition, 2001.

[HKP91] J. HERTZ, A. KROGH, et R. G. PALMER. *Introduction to the theory of neural computation*. Lecture Notes, Santa Fe Institute, Addison Wesley Publishing, 1991.

[HSW89] K. HORNIK, M. STINCHCOMBE, et H. WHITE. « Multilayer feedforward networks are universal approximators ». *Neural Networks*, 2 :359–366, 1989.

[KA09] H. KAHRAMANLI et N. ALLAHVERDIA. « Extracting rules for classification problems : AIS based approach ». *Expert Systems with Applications*, 36(7) :10494–10502, 2009.

[KAMN10] M. KAYTOUE, Z. ASSAGHIR, N. MESSAI, et A. NAPOLI. « Classification de données numériques par treillis de concepts basée sur une similarité symbolique/numérique ». Dans *SFC'2010*, 2010.

[KM05] E. KOLMAN et M. MARGALIOT. « Are Artificial Neural Networks White Boxes ? ». *IEEE Transaction Neural Network*, 16(4) :844–852, 2005.

[KM06] E. KOLMAN et M. MARGALIOT. « Extracting Symbolic Knowledge from Recurrent Neural Networks - A Fuzzy Logic Approach ». *Fuzzy sets and systems*, 2006.

[KMN08] S. KOUAMO, P. MELATAGIA, et R. NDOUNDAM. « Compression d'images avec les RNA. Cas de l'algorithme de rétro propagation du gradient de l'erreur ». Dans *actes du Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées (CARI'08)*, pages 127–134, 2008.

[KO02] S. O. KUZNETSOV et A. OBIEDKOV. « Comparing performance of Algorithms for generating concepts lattices ». Dans methods Special Issue on Concept Lattice-based THEORY et tools for Knowledge Discovery in DATABASES 14(2/3), éditeurs, *Journal of Experimental and Theoretical Artificial intelligence (JETAI)*, pages 198–216, 2002.

[Koh82] T. KOHONEN. « Self-organized formation of topolgically correct fiture maps ». *Biological Cybernetics*, 43 :59–69, 1982.

[KZ02] W. KLÖSGEN et J. M. ZYTKOW. *Handbook of Datamining and knowledge discovery*. Oxford University Press, 2002.

[Lan95] P. LANGLEY. *Elements of Machine Learning*. Morgan Kaufman, Palo Alto, 1995.

[LCZ09] D. LIU, T. CHANG, et Y. ZHANG. « A Constructive Algorithm for Feedforward Neural Networks With Incremental Training ». *Journal of Computer Science*, 5(11) :843–848, 2009.

[LJ97] T. LOFSTROM et U. JOHANSSON. « Predicting the benefict of rule extraction : A novel component of data mining ». *Human IT*, 7(3) :78–108, 1997.

[LS93] M. LESHNO et S. SCHOCKEN. « Multilayer feedforward networks with a nonpolynomial activation function can approximate any function ». *Neural Networks*, 6 :861–867, 1993.

[MA43] W. S. MCCULLOH et W. APITTS. « A logical calculs of the ideas imminent in nervous activity ». *Bullintin of mathematics biophysics*, 5 :115–133, 1943.

[Mac67] J. B. MACQUEEN. « Some Methods for classification and Analysis of Multivariate Observations ». *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1 :281–297, 1967.

[MDNST08] N. MESSAI, M. D. DEVIGNES, A. NAPOLI, et M. SMAIL-TABBONE. « Many-Valued Concept Lattices for Conceptual Clustering and Information Retrieval ». Dans *18th European Conference on Artificial Intelligence ECAI'08*, pages 127–131, 2008.

[MGR90] M. MARCHAND, M. GOLEA, et P. RUJAN. « A convergence theorem for sequential learning in two layers perceptron ». *Europhysics Letters*, 11(6) :487–492, 1990.

[Mic93] R. S. MICHALSKI. *A theory and methodoly of inductive learning, Readings in Knowledge Acquisition and Learning : Automating the construction and Improvement of expert systems*. B.G Buchanan and Wikins (Eds.), San Mateo CA, M. Kaufmann, 1993.

[MKS+90] O. MATAN, R. K. KIANG, C. E. STENARD, B. BOSER, J. S. DENKER, D. HENDERSON, R. E. HOWARD, W. HUBBARD, L. D. JACKEL, et Y. LECUN. « Handwritten character recognition using neural network architectures ». Dans *Proc. of the 4th US Postal Service Advanced Technology Conference, Washington D.C.*, pages 1003–1011, 1990.

[MN88] M. MEZARD et J. NADAL.. « Learning feed forward network : the tiling algorithm ». *J. Phys A : Math Gen.*, 22 :2191–2203, 1988.

[MN93] E. MEPHU NGUIFO. « Une nouvelle approche basée sur les treillis de Galois pour l'apprentissage de concepts ». Dans *Mathématiques et sciences humaines*, volume 123, pages 19–38, 1993.

[MN01] E. MEPHU NGUIFO. *Extraction des connaissances basée sur les treillis de Galois : Méthodes et applications*. Thèse de HDR, Université d'Artois, 2001.

[MNN05] E. MEPHU NGUIFO et P. NJIWOUA. « Treillis de concepts et classification supervisée ». Dans *RSTI-TSI*, volume 24 (4), pages 449–488, 2005.

[MNTT08] E. MEPHU NGUIFO, N. TSOPZÉ, et G. TINDO. « M-CLANN : Multi-class Concept Lattice-Based Artificial Neural Network for Supervised Classification ». Dans *proc. of the International Conference on Artificial Neural Networks - ICANN 2008, LNCS 5164/2008*, pages 812–821, 2008.

[MNTT09] E. MEPHU NGUIFO, N. TSOPZÉ, et G. TINDO. « *Multiclass Concept Lattice-Based Artificial Neural Networks* », volume 258 de *Studies in Computational Intelligence*, pages 103–121. Springer Berlin / Heidelberg, nov 2009.

[MP69] M. MINSKY et S. PAPERT. *Perceptrons*. MIT Press, Cambridge MA, 1969.

[MR91] B. MÜLLER et J. REINHARDT. *Neural Networks, An introduction*. Springer Verlag, 1991.

[NB07] M. C. NICOLETTI et J. R. BERTINI. « An Empirical evaluation of constructive neural networks algorithms in classifciation tasks ». *Int. Journal of Innovative Computing and applications*, 1(1) :2–13, 2007.

[Ndo05] R. NDOUNDAM. *Contribution à l'étude de la dynamique des réseaux d'automates*. Université de Yaoundé 1, Thèse d'état, 2005.

[Ned98] T. NEDJARI. *Réseaux de neurones artificiels et connaissances symboliques : Insertion, Raffinement et Extraction*. Université de Paris 13, Thèse de Doctorat, 1998.

[NHBM98] D. J. NEWMANN, S. HETTICH, C. L. BLAKE, et C. J. MERZ. *(UCI)Repository of machine learning databases*. Dept. Inform. Comput. Sci., Univ. California, Irvine, CA, http ://www.ics.uci.edu/AI/ML/MLDBRepository.html, 1998.

[NT03] R. NDOUNDAM et M. TCHUENTE. « Exponential transient length generated by a neuronal recurrence equation ». *Theor. Comput. Sci.*, 306(1-3) :513–533, 2003.

[Par85] D. PARKER. « Learning Logic ». Tech report tr-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, 1985. Describes the package natbib.

[PBTL99] N. PASQUIER, Y. BASTIDE, R. TAOUIL, et L. LAKHAL. « Efficient mining of association rules using closed itemset lattices ». *Journal of Information Systems*, 24(1) :25–46, 1999.

[Pea86] J. PEARL. « Fusion, propagation, and structuring in belief networks ». *Artificial Intelligence*, 29 :241–286, 1986.

[Pea00] J. PEARL. *Causality : Models, Reasoning, and Inference*. Cambridge University Press, 2000.

[Pol98] G. POLAILLON. *Organisation et interprétation par les treillis de Galois de données de type multivalué, intervalle ou histogramme*. Thèse de Doctorat, Université de Paris IX-Dauphine, 1998.

[Pou95] H. POULARD. « Barycentric correction procedure : A fast method of learning threshold units ». Dans *Proc. WCNN 95 Washington, D.C.*, pages 710–713, 1995.

[PYH95] R. PAREKH, J. YANG, et V. HONAVAR. « Constructive Neural Networks Learning Algorithms for Multi-Category Classification ». Tech report isu cs tr 95-15, Department of Computer Science Lowa State University, 1995. Describes the package natbib.

[PYH97a] R. PAREKH, J. YANG, et V. HONAVAR. « Constructive Neural Network learning Algorithm for Multi-Category Real Valued Pattern Classification ». Rapport de recherche, Department of Computer Science Lowa State University, 1997.

[PYH97b] R. PAREKH, R. YANG, et V. HONAVAR. « Mupstart-A constructive Neural Network Learning Algorithm for multi-Category Patterns Classification ». Dans *Proceedings of the IEEE/INNS International Conference on Neural Networks INNS'97*, pages 81–106, 1997.

[PYH99] R. PAREKH, J. YANG, et V. HONAVAR. « Comparison of performance of Variants of Single-layer Perceptron Algorithms on Non-separable Datasets ». *Neural, Parallel, and Scientific Computations*, 3, 1999.

[PYH00] R. PAREKH, J. YANG, et V. HONAVAR. « Constructive Neural-Network Learning Algorithms for Pattern Classification ». *IEEE Transactions on Neural Networks*, 11(2) :436–451, 2000.

[Qui86] J. R. QUINLAN. « Induction of decision trees ». *Machine Learning*, pages 81–106, 1986.

[Qui93] J. R. QUINLAN. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[Rak03] A. RAKOTOMAMONJY. « Variable Selection Using SVM-based Criteria ». *Journal of Machine Learning Research*, 3 :1357–1370, 2003.

[Ren06] J. P. RENNARD. *Réseaux de neurones*. Vuibert, 2006.

[RHW86a] D. E. RUMELHART, G. E. HINTON, et R. J. WILLIAMS. « Learning internal representations by error propagation ». *In Parallel Distributed Processing, Cambridge, MA : MIT Press*, 1 :318–362, 1986.

[RHW86b] D. E. RUMELHART, G. E. HINTON, et R. J. WILLIAMS. « Learning representations by back-propagating errors ». *Nature*, 323 :533–536, 1986.

[Roj96] R. ROJAS. *Neural networks A systematic introduction*. Springer Verlag, 1996.

[Ros58] F. ROSENBLATT. « The Perceptron : a probabilistic model for information storage and organization in the brain ». *Psychological Review*, 6(65) :386–408, 1958.

[Rud07] S. RUDOLPH. « Using FCA for Encoding of Closure Operators into Neural Networks ». Dans SPRINGER-VERLAG, éditeur, *In Uta Priss, Simon Polovina, Richard Hill, Conceptual Structures : Knowledge Architectures for Smart Applications, ICCS 2007, LNCS*, volume 4604, pages 321–332, 2007.

[Sch94] F. W. SCHMIDT. *Neural Pattern Classiying Systems, Theory and experiments with trainable classifiers*. Thesis, Techniche Universiteit Deif, 1994.

[Set97] R. SETIONO. « Extracting rules from neural networks by pruning and hidden-unit splitting ». *Neural Computation*, 9(1) :205–225, 1997.

[Set01] R. SETIONO. « Extracting M-of-N rules from trained neural networks ». *IEEE Transactions on Neural Networks*, 11 :306–312, 2001.

[Sha02] Y. SHACHMUROVE. « Applying Artificial Neural Networks to Business, Economics and Finance ». Penn CARESS Working Papers 5ecbb5c20d3d547f357aa1306, Penn Economics Department, August 2002.

[SL00] R. SETIONO et W. K. LEOW. « FERNN : An algorithm for fast extraction of rules from neural networks ». *Applied Intelligence*, 12(1-2) :15–25, 2000.

[SM02] K. O. STANLEY et R. MIIKKULAINEN. « Evolving Neural Networks through Augmenting Topologies ». *Evolutionary Computation*, 10(2) :99–127, 2002.

[Tea91] S. B. Thrun et AL.. « The Monk's problems : A performance comparison of different learning algorithms ». Rapport technique CMU-CS-91-197, 1991.

[Thr95] S. THRUN. « Extracting rules from artificial neural networks with distributed representations ». *Advances in Neural information Processing Systems, the MIT Press, Cambridge MA*, 7 :16–21, 1995.

[TMNT07a] N. TSOPZÉ, E. MEPHU NGUIFO, et G. TINDO. « CLANN : Concept Lattice-based Artificial Neural Network for Supervised Classification ». Dans Peter W.

EKLUND, Jean DIATTA, et Michel LIQUIERE, éditeurs, *Proceedings of the Fifth International Conference on Concept Lattices and Their Applications, CLA 2007, Montpellier, France, October 24-26, 2007*, volume 331 de *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[TMNT07b] N. TSOPZÉ, E. MEPHU NGUIFO, et G. TINDO. « Etude des algorithmes de construction des réseaux de neurones ». *Révue des Nouvelles Technologies de l'Information, extraction et gestion des connaissances, EGC'2007*, pages 9–20, 2007.

[TMNT07c] N. TSOPZÉ, E. MEPHU NGUIFO, et G. TINDO. « Une approche basée sur les treillis de Galois pour la construction des réseaux de neurones ». Dans *Actes des des 11'iemes rencontres de la Société Française de la Classification SFC'07, Editeurs Hudry O., Charon I. Hébrail G.*, pages 186–189, 2007.

[TMNT08] N. TSOPZÉ, E. MEPHU NGUIFO, et G. TINDO. « Une approche basée sur les treillis de Galois pour le construction des réseaux de neurones ». Dans *actes du Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées (CARI'08)*, pages 399–407, 2008.

[TP90] A. C. TSOI et R. A. PEARSON. « Comparison of three classification techniques, CART, C4.5 and multi-layer perceptrons ». Dans *Proc. of the 1990 conference on Advances in neural information processing systems 3*, pages 963–969, 1990.

[TR07] C. R. TOSH et G. D. RUXTON. « Introduction. The use of artificial neural networks to study perception in animals ». *Philosophical Transaction of the Royal Society B*, 362 :337–338, 2007.

[TS92] G. TOWELL et J. W. SHAVLIK. « Interpretation of Artificial Neural Networks : Mapping Knowledge-Based Neural Networks into Rules ». *Advances in Neural Information Processing Systems, San Mateo, CA, Morgan Kaufmann*, 4 :977–984, 1992.

[TS93] G. G. TOWELL et J. W. SHAVLIK. « The Extraction of Refined Rules from Knowledge-Based Neural Networks ». *Machine Learning*, pages 71–101, 1993.

[TS94] G. G. TOWELL et J. W. SHAVLIK. « Knowledge-based artificial neural networks ». *Artificial Intelligence*, 70(1-2) :119–165, 1994.

[Tso09] N. TSOPZÉ. « Une généralisation de l' approche décompositionnelle d'extraction de règles d'un réseau de neurones artificiels ». Dans *Actes de la 11ème conférence d'apprentissage (CAP'09),Eds. Bennani Y. et Rouveirol C., Hammamet (Tunisie)*, pages 375–378, 2009.

[Vap82] V. VAPNIK. *Estimation of dependences based on empirical data*. Springer Verlag, 1982.

[Vap95] V. VAPNIK. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[VJJ97] E. VONK, L. C. JAIN, et R. P. JOHNSON. *Automatic generation of neural network architecture using evolutionary computation*. World Scientific Pub Co Inc, 1997.

[WF05] I. H. WITTEN et E. FRANK. *Data Mining : Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005.

[WH60] B. WIDROW et M. E. HOFF. « Adaptive switching circuits ». *Ire Wescon Convention*, 4 :96–104, 1960.

[Wil82] R. WILLE. « Restructuring Lattice Theory : An Approach Based on Hierarchies of Concepts ». *I. Rival(Ed) Ordered Sets*, 95 :445–470, 1982.

[Wit99] D. WITKOWSKA. « Applying artificial neural networks to bank-decision simulations ». *International Advances in Economic Research*, 5(3) :350–368, 1999.

[WM97] D. H. WOLPERT et W. G. MACREADY. « No Free Lunch Theorems for Optimization ». *IEEE Transactions on Evolutionary Computation*, 1 :67–82, 1997.

[YB00] M. YACOUB et Y. BENNANI. « Features Selection and Architecture Optimization in Connectionist Systems ». *International Journal of Neural Systems*, 10(5) :379–395, 2000.

[YPH96] J. YANG, R. PAREKH, et V. HONAVAR. « Mtiling-A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification ». Dans *Proceedings of the World Congress on Neural Networks*, pages 182–187, 1996.

[YPH99] J. YANG, R. PAREKH, et V. HONAVAR. « Distal : An Inter-pattern Distance-based Constructive Learning Algorithm ». *Intelligent Data Analysis*, 3 :55–73, 1999.

Résumé

Les réseaux de neurones artificiels connaissent des succès dans plusieurs domaines. Mais les utilisateurs des réseaux de neurones sont souvent confrontés aux problèmes de définition de son architecture et d'interprétabilité de ses résultats. Plusieurs travaux ont essayé d'apporter une solution à ces problèmes. Pour les problèmes d'architecture, certains auteurs proposent de déduire cette architecture à partir d'un ensemble de connaissances décrivant le domaine du problème et d'autres proposent d'ajouter de manière incrémentale les neurones à un réseau ayant une taille initiale minimale. Les solutions proposées pour le problème d'interprétabilité des résultats consistent à extraire un ensemble de règles décrivant le fonctionnement du réseau. Cette thèse contribue à la résolution de ces deux problèmes. Nous nous limitons à l'utilisation des réseaux de neurones dans la résolution des problèmes de classification.

Nous présentons dans cette thèse un état de l'art des méthodes existantes de recherche d'architecture de réseaux de neurones : une étude théorique et expérimentale est aussi faite. De cette étude, nous observons comme limites de ces méthodes la disponibilité absolue des connaissances pour construire un réseau interprétable et la construction des réseaux difficiles à interpréter en absence de connaissances. En alternative, nous proposons une méthode appelée CLANN (Concept Lattice-based Artificial Neural network) basée les treillis de Galois qui construit un demi-treillis à partir des données et déduire de ce demi-treillis l'architacture du réseau. CLANN étant limitée à la résolution des problèmes à deux classes, nous proposons MCLANN permettant d'étendre cette méthodes de recherche d'architecture des réseaux de neurones aux problèmes à plusieurs classes.

Nous proposons aussi une méthode appelée 'Approche des MaxSubsets' pour l'extraction des règles à partir d'un réseau de neurones. La particularité de cette méthode est la possibilité d'extraire les deux formats de règles ('si alors' et 'm parmi N') à partir d'une structure que nous construisons. Nous proposons aussi une façon d'expliquer le résultat calculé par le réseau construit par la méthode MCLANN au sujet d'un exemple.

Mots-clés: Réseau de neurones, Apprentissage, Architecture des réseau de neurones, extraction des règles, Treillis de Galois.

Abstract

The artificial neural networks are successfully applied in many applications. But the users are confronted with two problems : defining the architecture of the neural network able to solve their problems and interpreting the network result. Many research works propose some solutions about these problems : to find out the architecture of the network, some authors propose to use the problem domain theory and deduct the network architecture and some others propose to dynamically add neurons in the existing networks until satisfaction. For the interpretability problem, solutions consist to extract rules which describe the network behaviour after training. The contributions of this thesis concern these problems. The thesis are limited to the use of the artificial neural networks in solving the classification problem.

In this thesis, we present a state of art of the existing methods of finding the neural network architecture : we present a theoritical and experimental study of these methods. From this study, we observe some limits : difficulty to use some method when the knowledges are not available ; and the network is seem as 'black box' when using other methods. We a new method called CLANN (Concept Lattice-based Artificial Neural Network) which builds from the training data a semi concepts lattice and translates this semi lattice into the network architecture. As CLANN is limited to the two classes problems, we propose MCLANN which extends CLANN to many classes problems.

A new method of rules extraction called 'MaxSubsets Approach' is also presented in this thesis. Its particularity is the possibility of extracting the two kind of rules (If then and M-of-N) from an internal structure. We describe how to explain the MCLANN built network result about some inputs.

Keywords: Artificial neural network, machine learning, Neural network architecture, rules extraction from trained artificial neural network, Concepts lattice.