TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

La complexité et la dynamique de beaucoup de problèmes du monde réel, particulièrement en ingénierie et industrie, exigent des méthodes et des outils sophistiquées pour construire des systèmes d'information intelligents. Tels systèmes devraient être capables d'apprendre et traiter différents types de données et connaissance par interaction avec l'environnement d'une façon incrémentale. Sept exigences majeures des systèmes intelligents présents sont discutées dans [KAS 01a]. Elles concernent l'apprentissage rapide, l'apprentissage adaptatif incrémental en-ligne, organisation de structure ouverte, mémorisation de l'information, interaction active, acquisition de la connaissance et auto-amélioration, et l'apprentissage spatial et temporel.

Les systèmes vont fonctionner dans un environnement caractérisé principalement par les incertitudes, le changement rapide et l'imprécision. L'intelligence des systèmes est leur capacité de s'adapter aux nouvelles situations [BOU 07].

Le soft computing est un consortium de méthodologies qui fournissent une fondation pour la conception et le déploiement de systèmes intelligents. Contrairement au hard computing traditionnel, le soft computing vise à une adaptation au monde réel caractérisé principalement par l'imprécision, l'incertitude et la vérité partielle. Exploiter la tolérance à ces caractéristiques pour accomplir une souplesse, une robustesse, un coût de solution bas et un meilleur rapport avec la réalité est le principe directeur du soft computing [ZAD 94, ZAD 97].

L'idée de base du soft computing, intégrant la théorie des ensembles flous, les réseaux de neurones artificiels et les algorithmes évolutionnaires, est d'utiliser l'hybridation de ces méthodes intelligentes pour profiter de la capacité de l'apprentissage des réseaux de neurones, la capacité de gérer l'incertitude des ensembles flous et le potentiel de recherche des AEs dont l'objectif d'imiter les processus de raisonnement et de prise de décision d'un être humain. Une raison pour le succès du soft computing est la synergie dérivée de ses composants. La création des systèmes hybrides basés sur une intégration de ces technologies fournit un raisonnement complémentaire et méthodes de recherche qui permettent de développer des outils adaptatifs et résoudre des problèmes complexes [BON 01].

Le problème d'adaptabilité des systèmes intelligents a attiré l'attention des chercheurs récemment. Cela a mené pendant les dernières années à la formation du domaine des systèmes intelligents adaptatifs.

Dans les situations dynamiques, les systèmes vont forcément fonctionner dans un environnement qui se change dans le temps [MAU 05], où des nouvelles données, de nature différente à celles utilisées pour l'apprentissage, peuvent y arriver, et introduire des nouveaux concepts ou modifier les concepts existants, ce qui exige – par contraste aux méthodes d'apprentissage statiques – des systèmes qui peuvent suivre les transitions progressives des concepts et se changer graduellement pour s'adapter à la dynamique de leur environnement.

L'adaptation est le processus des changements structurels et fonctionnels d'un système pour améliorer sa performance dans un environnement changeant [KAS 07].

L'apprentissage adaptatif vise à résoudre le dilemme bien connu de stabilité/plasticité qui signifie que le système doit être assez stable pour retenir les modèles appris à partir de données

observées précédemment, en étant assez flexible pour apprendre de nouveaux modèles à partir de nouvelles données. Un progrès significatif dans l'apprentissage adaptatif a été accompli dû à la théorie de la résonance adaptative (Adaptive Resonance Theory ART) et ses différents modèles qui incluent des modèles non supervisés (ART1, ART2, FuzzyART) et des versions supervisées (ARTMAP, FuzzyARTMAP) [CAR 87, CAR 90, CAR 91a, CAR 91b, CAR 92].

L'apprentissage en ligne, l'apprentissage incrémental et l'apprentissage life-long sont des méthodes typiques de l'apprentissage adaptatif [KAS 07]. Tout ces types d'apprentissage cherchent à implémenter des systèmes capables d'apprendre des nouvelles informations à partir de nouvelles données sans faire référence aux données précédemment utilisées ni oublier la connaissance apprise à partir desquelles, c.-à-d. la capacité du système à mettre à jour sa connaissance, sa structure et ses fonctionnalités uniquement avec les nouvelles données sans devoir reprendre l'apprentissage avec les données déjà utilisées. Ceci doit s'effectuer sans oublier la connaissance apprise précédemment dans le but de répondre au mieux au dilemme de stabilité/plasticité expliqué au dessus. La différence entre l'apprentissage en-ligne, l'apprentissage incrémental et l'apprentissage life-long est dans leur façon de traiter les nouvelles informations.

Le fait de devoir traiter les exemples sur le moment s'appelle l'apprentissage en-ligne [ORS 07]. Il est concerné d'apprendre des instances individuelles reçues à partir de l'environnement pendant le fonctionnement du système, et ces instances peuvent exister seulement en court temps puisque, à cause de l'interaction continuelle entre le système et l'environnement, la suite de données est sans cesse croissante et il n'est pas question de tous mémoriser. Cela signifie qu'une étape d'apprentissage aura lieu à chaque présentation d'une forme d'entraînement tirée aléatoirement (usuellement en temps réel c.-à-d. le temps de traitement doit être court et limité) [CHE 07, KAS 01a, ORS 07, SU 06]. Récemment, plusieurs algorithmes d'apprentissage en-ligne ont été proposés (voir les références de [SU 06]).

L'apprentissage incrémental concentre sur le traitement séquentiel de données [YAL 07] (le jeu de données est fourni en séquence) sans contrainte de temps [ORS 07], on parle alors de nouvel ensemble d'apprentissage plutôt que de nouvelle donnée. Pour chaque séquence de données, le concept à apprendre est supposé fixe, on déclenche l'apprentissage sur cette séquence et on doit retenir seulement une structure de connaissance dans la mémoire c.-à-d. mémoriser exactement une définition pour chaque concept.

L'apprentissage life-long, également appelé apprentissage continuel, est une extension de l'apprentissage en-ligne et l'apprentissage incrémental mais exige des méthodes plus sophistiquées. Il est concerné de la capacité d'un système d'apprendre à partir de données venant continuellement dans un environnement changeant tout au long de sa durée de vie [HAM 01, KAM 07, KAS 07]. La suite de données à apprendre est donc infinie. Dans l'apprentissage life-long, le système apprend un ensemble ouvert de données influencé par le bruit et autres facteurs, et le système préserve la connaissance apprise précédemment dans un environnement changeant tant qu'elle ne contredit pas la tâche courante [KAM 07].

C'est dans le cadre de l'apprentissage incrémental que se place notre travail. Notre objectif est de concevoir un système à apprentissage hybride incrémental qui peut s'adapter à un environnement dynamique. Pour atteindre cet objectif, nous sommes passés par trois étapes reflétées par la structuration de ce mémoire. En effet, outre la partie introductive et la conclusion générale, ce document est structuré en trois chapitres.

Dans le premier chapitre, nous nous intéressons aux systèmes intelligents rassemblés sous la dénomination du "soft computing". Ce terme désigne l'imitation du style humain de raisonnement

et de prise de décision afin d'affronter l'imprécision et l'incertitude impliquées dans les problèmes complexes. Le soft-computing intègre les ensembles flous, les réseaux de neurones artificiels et les algorithmes évolutionnaires pour créer des systèmes hybrides intelligents puissants. Nous présentons, dans ce chapitre, les notions fondamentales des méthodes de base du soft computing. Nous étudions ensuite plusieurs intégrations entre ces méthodes prises deux à deux puis toutes les trois, nous avons constaté la richesse des approches hybrides du soft-computing, ce qui nous a encouragés à envisager l'utilisation de l'une de ces méthodes dans le cadre de notre travail.

Nous abordons, dans le second chapitre, la notion d'apprentissage adaptatif incrémental qui fait référence au processus d'accumulation et de gestion de connaissances dans le temps. Il est très approprié pour les tâches d'apprentissage dans lesquelles les ensembles de données d'entraînement deviennent disponibles pendant une longue période de temps. Nous présentons le principe, les caractéristiques ainsi que les types de l'apprentissage incrémental. Une grande partie du chapitre est consacrée à la présentation des méthodes et algorithmes que nous avons étudiés dans le domaine de l'apprentissage incrémental.

Le système proposé est présenté dans le troisième chapitre, c'est un système adaptatif à apprentissage incrémental conçu en utilisant une hybridation d'approches issues du soft computing dans le domaine applicatif de la classification d'images. C'est un système de classification d'images capable de s'adapter de manière incrémentale aux changements de l'environnement. L'approche suggérée consiste à apprendre des règles de classification par un réseau de neurones flou, puis optimiser ces règles par les algorithmes génétiques.

CHAPITRE 1

LES SYSTEMES EVOLUTIFS INTELLIGENTS EN SOFT COMPUTING

Les systèmes intelligents sont rassemblés sous la dénomination du "soft computing". Ce terme désigne l'imitation du style humain de raisonnement et de prise de décision afin d'affronter l'imprécision et l'incertitude impliquées dans les problèmes complexes [TUN 05]. Le soft-computing intègre les ensembles flous, les réseaux de neurones artificiels et les algorithmes évolutionnaires pour créer des systèmes hybrides intelligents puissants.

1.1. Introduction

L'intelligence computationnelle et le soft computing sont deux termes inventés pour spécifier le champ d'étude des systèmes intelligents qui essaient de modéliser le comportement intelligent chez l'être humain en utilisant des idées de la biologie et en considérant la définition et l'utilisation de l'incertitude dans le calcul [GUD 97]. Ces deux termes sont utilisés parfois comme synonymes, cependant, l'intelligence computationnelle comprend toutes les techniques qui peuvent aider dans la modélisation et la description des systèmes complexes d'inférence et de prise de décision, y compris les techniques du Soft Computing [TET 01]. Le terme intelligence computationnelle, comme défini par Zadeh, est la combinaison du soft computing et du traitement numérique [DOT 01]. Les approches telles que le raisonnement symbolique, les systèmes de représentation de connaissance par des règles logiques non floues et le traitement automatique de langue sont des techniques de l'intelligence computationnelle mais ne se qualifient pas comme des approches du Soft Computing [TET 01]. On peut donc dire que le Soft Computing est un sous-domaine de l'intelligence computationnelle.

Bien qu'en hard computing[1], l'imprécision et l'incertitude soient des propriétés indésirables, en soft computing, la tolérance à l'imprécision et l'incertitude est exploitée pour accomplir une souplesse, coût inférieur, haut Quotient d'Intelligence de la Machine (MIQ) et économie de communication [ZIL 01]. Cependant, il est très efficace lorsqu'il est appliqué aux problèmes du monde réel qui ne sont pas capables d'être résolus par le hard computing.

Les méthodologies de base du soft computing sont la théorie des ensembles flous, les réseaux de neurones artificiels et les algorithmes évolutionnaires. Chacune de ces méthodes a des propriétés caractéristiques (telles que la capacité d'apprendre, l'explicabilité des décisions) qui la rendent convenable pour des problèmes particuliers mais pas pour les autres. Pour profiter des avantages de plusieurs méthodes, le soft computing s'intéresse à leur intégration qui fournit une fondation pour la conception et le déploiement de systèmes intelligents.

[1] Le hard computing classique est définie comme l'antipode du soft computing, c.-à-d., une collection hétérogène de méthodes de calcul traditionnelles [OVA 06]. Il est basé sur la logique binaire et l'analyse numérique, et caractérisé par la précision et la certitude.

Dans ce chapitre, nous présentons d'abord brièvement les bases de chacune des approches isolée du soft computing pour permettre une meilleure compréhension de leur intégration.

1.2. Systèmes Intelligents

L'intelligence fait référence aux mentalités humaines et au processus de cognition. Elle implique la capacité de comprendre, raisonner, apprendre et mémoriser [SIN 00].

Bien qu'on soit encore loin de réaliser une machine qui soit aussi intelligente qu'un être humain, nous ne pouvons pas ignorer le fait que les gens exigent aujourd'hui des systèmes qui peuvent compléter leurs capacités physiques ou cognitives. Cette demande mène à la création des systèmes intelligents [TAY 00]. Ces systèmes doivent imiter les principales caractéristiques de l'intelligence de l'être humain telles que la représentation de connaissance, la formation de concept, le raisonnement et l'adaptation [KAS 07].

Un système intelligent est défini dans [KAS 07] comme étant :

Un système d'information qui manifeste des caractéristiques d'intelligence, telles que l'apprentissage, la généralisation, le raisonnement, l'adaptation ou la découverte de connaissances, et applique celles-ci aux tâches complexes telles que la prise de décision, le contrôle adaptatif, la reconnaissance des formes, le traitement de la parole, de l'image et de l'information multimodale, etc.

1.3. Soft Computing

Le terme Soft Computing a été inventé originairement en 1993 par Zadeh L., l'inventeur des ensembles flous et de la logique floue. La définition originale de Zadeh L. du soft computing est [ZAD 94] :

Le soft computing est une collection de méthodologies qui visent à exploiter la tolérance à l'imprécision, l'incertitude et la vérité partielle pour aboutir à une certaine souplesse, robustesse et diminution du coût de la solution. Les principaux composants du soft computing sont la logique floue, les réseaux de neurones artificiels, et le calcul probabiliste. Le soft computing joue un rôle de plus en plus important dans beaucoup de domaines d'application, y compris l'ingénierie de logiciel. Le modèle de base pour le soft computing est l'esprit humain.

En 1997, Lotfi Zadeh a ajouté les algorithmes génétiques comme un autre composant du Soft Computing [ZAD 97].

Le soft computing fournit une occasion attirante pour représenter l'ambiguïté dans la pensée humaine avec l'incertitude de la vie réelle. Il cause un changement de paradigme dans les domaines des sciences et de l'ingénierie puisqu'il peut résoudre des problèmes qui n'ont pas pu être résolus par les méthodes analytiques traditionnelles. De plus, il produit une représentation riche de connaissance (symbole et modèle), acquisition adaptative de connaissance (par apprentissage automatique à partir de données et en interviewant des experts) et traitement adaptatif de connaissances (inférence en interfaçant entre la connaissance symbolique et modèle), qui permettent aux systèmes intelligents d'être construits à moindre coût [haut quotient d'intelligence de la machine (HMIQ)] [DOT 01].

Il est difficile de trouver une définition nette de la signification et des limites des disciplines impliquées en Soft Computing. Le Soft Computing est un terme général couvrant plusieurs méthodologies qui, contrairement aux algorithmes conventionnels, sont tolérantes à l'imprécision, l'incertitude et la vérité partielle. Les techniques du Soft Computing ne souffrent pas de la fragilité et l'inflexibilité des approches algorithmiques standards. En conséquence, elles offrent une certaine adaptabilité, autrement dit, elles peuvent suivre un environnement changeant du problème, et donc s'adaptent bien aux problèmes d'intérêt qui n'admettent pas des solutions qui sont fixées une fois pour toutes [TET 01].

1.4. Notions Fondamentales des Méthodes de Base du Soft Computing

Plusieurs méthodes telles que les ensembles bruts et les réseaux probabilistes peuvent être qualifiées de type Soft Computing. Cependant, nous nous sommes volontairement limités dans ce chapitre aux approches qui concernent la logique floue, les algorithmes évolutionnaires et les réseaux de neurones artificiels et leurs différentes combinaisons.

1.4.1. Réseaux de Neurones Artificiels (Artificial Neural Networks) RNAs

Une caractéristique clé des réseaux de neurones artificiels est leur capacité d'approximer des fonctions non linéaires arbitraires. Les RNAs peuvent donc être efficaces dans l'implémentation des systèmes intelligents capables d'apprendre et exécuter des tâches cognitives supérieures qui exigent de prendre des décisions fortement non linéaires seulement par interaction avec un système et en employant les données sur son comportement [KAR 04].

1.4.1.1. Inspiration biologique

Les réseaux de neurones artificiels sont des systèmes informatiques connexionnistes inspirés de l'architecture biologique des neurones dans le cerveau humain [KAR 04]. Les neurones biologiques sont des petites unités cellulaires interconnectés qui forment le système nerveux humain. Chaque neurone a trois composants structurels principaux : les dendrites, le corps cellulaire qui renferme le noyau, et l'axone. La communication entre ces neurones s'effectue via des impulsions nerveuses. Un neurone reçoit des signaux électriques à partir d'autres neurones par l'intermédiaire de ses dendrites, traite ces signaux dans le corps cellulaire et délivre le signal résultant aux neurones voisins par l'intermédiaire de son axone. À l'autre extrémité de l'axone, des zones de contact appelés synapses permettent la transmission du signal aux neurones voisins. Un diagramme schématique représentant ce concept est présenté dans Fig 1.1 [KON 00].

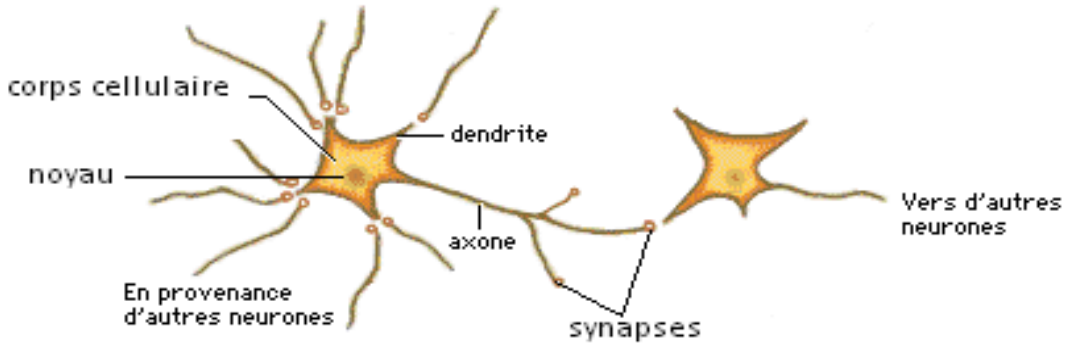


Figure 1.1 : Une représentation simplifiée des neurones biologiques

1.4.1.2. Architecture

Un Réseau de Neurones Artificiel RNA consiste en un ensemble d'éléments de traitement appelés neurones artificiels, qui représentent l'analogue électrique des neurones biologiques (Fig 1.2), habituellement organisés en couches et interconnectés par des poids synaptiques formant ainsi des structures de traitement distribuées parallèlement (Fig 1.3).

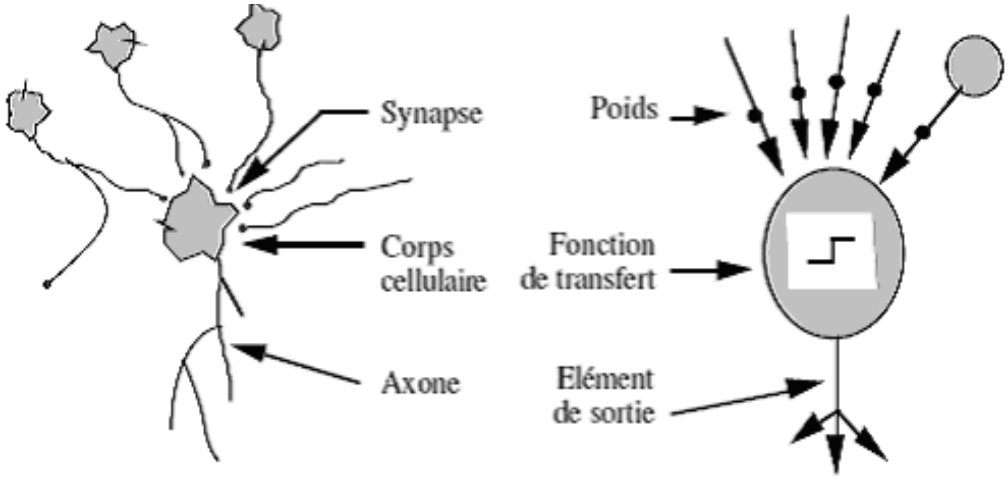


Figure 1.2 : Correspondance entre neurone biologique et neurone artificiel

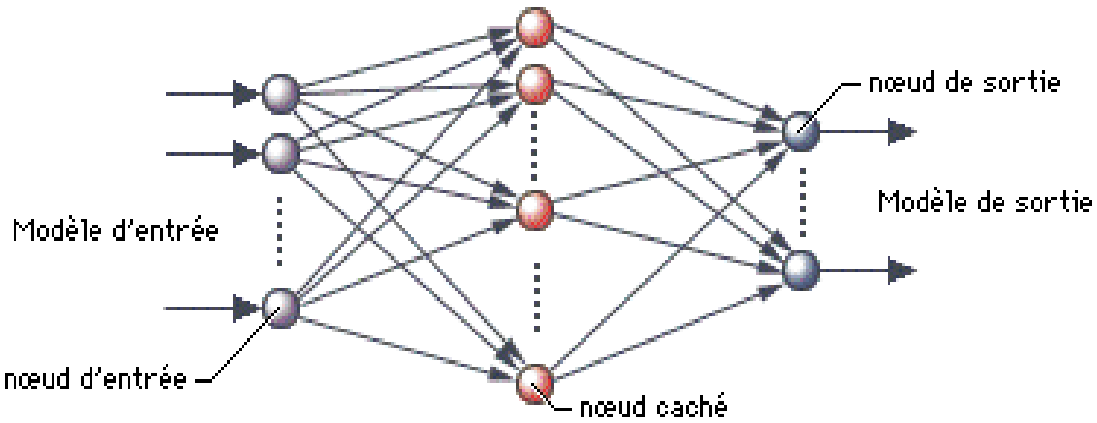


Figure 1.3 : Un réseau de neurones artificiel

Entre les nœuds d'entrée et de sortie figurent de nombreux autres nœuds, appelés pour cette raison nœuds cachés. Un réseau de neurones artificiel peut comprendre plusieurs niveaux de nœuds cachés.

Un RNA peut être décrit comme un graphe orienté et pondéré dans lequel chaque nœud i exécute une fonction de transfert (fonction d'activation), f_i, de la forme,

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij}\, x_j - \theta_i\right) \qquad (1)$$

où y_i est la sortie du nœud i, x_j est la j ème entrée au nœud, w_{ij} est le poids synaptique de la connexion entre les nœuds i et j. θ_i est le seuil (ou biais) du nœud (Fig 1.4) [XIN 99].

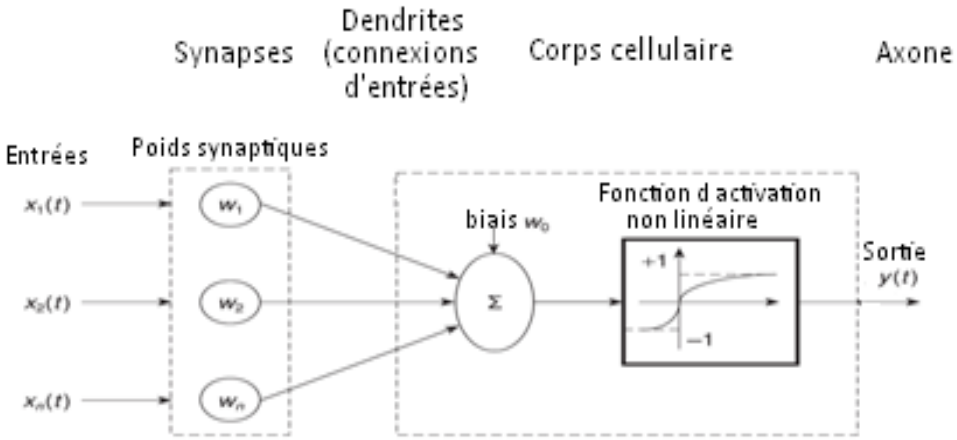


Figure 1.4 : Neurone Artificiel – Analogue électrique du neurone biologique [KAR 04]

1.4.1.3. Critères de distinction entre RNAs

Les différents types de RNAs sont distingués par trois critères dont le choix dépend principalement du type de problème considéré :

- Le type de neurone.
- La topologie du réseau.
- La règle d'apprentissage associée au réseau.

Le type de neurone

Le type de neurone se détermine par la forme de sa fonction d'activation. Cette forme varie selon le problème considéré et l'emplacement du neurone dans une couche donnée. Elle peut être une correspondance linéaire mais en général c'est une fonction non linéaire dont les plus utilisées sont : la fonction sigmoïde, la fonction signe, la fonction booléenne, la fonction tangente hyperbolique et la fonction Gaussienne.

La topologie du réseau

La topologie d'un RNA correspond au classement de ses neurones et leurs interconnexions de la couche d'entrée à la couche de sortie du réseau. De ce fait, elle concerne trois choix :

- Le nombre de couches cachées (dépend de la complexité du problème considéré).
- Le nombre de neurones dans chaque couche : couche d'entrée, couche cachée et couche de sortie (dépend du nombre de caractéristiques et de classes).
- La connectivité : Les RNAs peuvent être divisés selon leur connectivité en deux classes feedforward et récurrents. Un RNA est feedforward si chaque neurone est relié aux neurones de la couche ultérieure par des connexions unidirectionnelles et il n'y a aucune connexion d'un neurone vers lui-même ou vers un autre neurone de la même couche ou d'une couche précédente. Le réseau perceptron multicouche et le réseau RBF sont parmi les réseaux les plus connus qui utilisent la topologie feedforward. Contrairement à ce type, les réseaux récurrents permettent des connexions d'un neurone vers lui-même ou vers un autre neurone de la même couche ou d'une couche précédente. Plusieurs réseaux de neurones bien connus ont été conçus en se basant sur la topologie récurrente telle que le réseau Hopfield.

La règle d'apprentissage associée au réseau

L'apprentissage des RNAs est accompli typiquement en utilisant les exemples. L'essence d'un algorithme d'apprentissage est la règle d'apprentissage, c'est une règle de mise à jour des poids qui détermine comment les poids de connexion sont changés. L'apprentissage peut être divisé en gros en trois types : l'apprentissage non supervisé, l'apprentissage supervisé, et l'apprentissage par renforcement :

- Apprentissage non supervisé : il est basé uniquement sur les corrélations entre les données d'entrée. Aucune information sur la "sortie correcte" n'est disponible pour l'apprentissage. Le système suit des directives prédéfinies pour découvrir des propriétés collectives émergentes et organiser les données en groupes ou catégories. Les poids de connexion du réseau sont alors ajustés par une sorte de compétition entre les nœuds de la couche de sortie où le neurone gagnant sera celui avec la plus grande valeur. Parmi les règles d'apprentissage non supervisé : la règle de Hebb, l'apprentissage compétitif (quantification vectorielle LVQ) et les cartes caractéristiques auto-organisatrices [KAR 04, XIN 99].
- Apprentissage supervisé : dans lequel un ensemble d'exemples dont les sorties sont connues a priori est présenté au système. Pendant le processus d'apprentissage, les sorties du RNA sont comparées avec les sorties désirées. Une règle d'apprentissage appropriée (en général la règle de descente de gradient) utilise l'erreur entre la sortie obtenue et la sortie désirée pour ajuster itérativement les poids de connexion afin de minimiser une fonction d'erreur telle que l'erreur carrée moyenne totale. Parmi les règles d'apprentissage supervisé : l'algorithme de rétro-propagation, l'algorithme du Perceptron, la règle LMS et la règle Delta [KAR 04, XIN 99].
- Apprentissage par renforcement : il est basé seulement sur l'information que la sortie obtenue soit correcte ou pas. Dans le cas d'une sortie correcte, les connexions correspondantes qui mènent à cette sortie sont renforcées, autrement elles sont affaiblies. Parmi les stratégies utilisées pour implémenter un algorithme d'apprentissage renforcé sont : la comparaison renforcée, la critique heuristique adaptative, le Q apprentissage et le policy only scheme [KAR 04, XIN 99].

1.4.2. Algorithmes Evolutionnaires (Evolutionary Algorithms) AEs

Les algorithmes évolutionnaires sont une famille d'algorithmes dits bio-inspirés, car s'inspirant de la théorie de l'évolution pour résoudre des problèmes divers. Ils font ainsi évoluer un ensemble de solutions à un problème donné, dans l'optique de trouver les meilleurs résultats. Ce sont des algorithmes stochastiques, car ils utilisent itérativement des processus aléatoires.

La grande majorité de ces méthodes sont utilisées pour résoudre des problèmes d'optimisation, elles sont en cela des méta-heuristiques, bien que le cadre général ne soit pas nécessairement dédié aux algorithmes d'optimisation au sens strict.

1.4.2.1. Description générale

Les algorithmes évolutionnaires constituent une classe générale de méthodes de recherche et d'optimisation stochastiques inspirées des idées et principes de sélection naturelle introduits par Charles Darwin au 19ème siècle. Ils imitent les processus biologiques permettant aux populations des organismes de s'adapter à leur environnement. Ses approches sont : les stratégies d'évolution, les algorithmes génétiques, la programmation évolutionnaire et la programmation

génétique [TET 01, XIN 99, CHA 04]. Une caractéristique importante de toutes ces méthodes est leur stratégie de recherche basée-population [XIN 99]. Une population est l'ensemble d'individus qui représentent les solutions potentielles pour le problème considéré codées sous forme de chromosomes. Chaque chromosome est une séquence ordonnée de gènes dont chacun code un paramètre du problème sous forme d'un caractère binaire ou un nombre réel [CHA 04]. Les individus de la population rivalisent et échangent de l'information afin d'exécuter certaines tâches [XIN 99]. Une fonction objective évalue l'optimalité des individus et associe à chacun sa fitness qui représente la valeur de sa qualité. L'opérateur standard de la sélection est la sélection de la Roue de la Roulette qui assigne à chaque solution de la progéniture une probabilité de survie en proportion à sa fitness. Les survivants deviennent les prochains parents de la génération [CHA 04].

L'idée générale d'un algorithme évolutionnaire est de maintenir une population d'individus pour un problème donné, et la fait évoluer en fonction de la fitness des individus et en appliquant itérativement un ensemble d'opérateurs de recherche stochastiques (mutation, croisement et sélection), qui sont imitations directes de leurs équivalents génétiques, jusqu'à ce que certains critères de terminaison soient satisfaits. Chaque itération est appelée génération [TET 01]. La structure générale d'un AE est présentée dans Fig 1.5.

1. Générer la population initiale $G(0)$ aléatoirement, et mettre i = 0;
2. RÉPÉTER
   (a) Évaluer chaque individu dans la population;
   (b) Sélectionner des parents à partir de $G(i)$ en se basant sur leur fitness dans $G(i)$;
   (c) Appliquer des opérateurs de recherche aux parents et produire les fils qui forment G (i+1);
   (d) i = i + 1;
3. JUSQU'À CE QUE 'critère de terminaison' est satisfait

Figure 1.5 : Structure générale d'un algorithme évolutionnaire [XIN 99]

La population initiale peut être soit un échantillon aléatoire de l'espace d'individus soit un ensemble construit avec des individus trouvés par des procédures de recherche locales simples, si celles-ci sont disponibles [TET 01].

La mutation perturbe aléatoirement un individu; le croisement décompose deux individus distincts et ensuite recombine leurs parties pour former des nouveaux individus (il assure par cela l'échange d'information entre les individus); et la sélection duplique les individus les plus aptes trouvés dans une population à un taux proportionnel à leur qualité relative [TET 01].

Le processus résultant a tendance à trouver, en suffisamment de temps, des solutions globalement optimales au problème d'une façon très similaire à celle des populations des organismes de la nature qui ont tendance à s'adapter à leur environnement [TET 01].

1.4.2.2. Différentes approches évolutionnaires

Comme nous l'avons annoncé dans la section 2.1., les différentes approches des algorithmes évolutionnaires sont les stratégies d'évolution, les algorithmes génétiques, la programmation évolutionnaire et la programmation génétique.

Stratégies évolutionnaires

Les stratégies évolutionnaires (ES) ont été développées initialement par Rechenberg en 1973 comme une méthode de résolution des problèmes d'optimisation continus. Elles font utilisation d'une population de dimension égale à un avec une seule opération génétique, c.-à-d., la mutation. Schwefel en 1981 a utilisé la forme à valeur réelles pour représenter les chromosomes et a comparé la performance de la SE avec les techniques d'optimisation les plus traditionnelles. De plus, il a étendu la dimension de la population à plus de un et a introduit d'autres opérateurs tels que la sélection et le croisement [KAR 04].

Algorithmes génétiques

Les AGs représentent la forme la plus populaire des AEs. Ils ont été proposés par Holland en 1975 [CHA 04]. Ils accomplissent l'optimisation dans le domaine binaire pour imiter le processus évolutionnaire qui a lieu au niveau génétique.

Les techniques basées sur les AGs possèdent deux caractéristiques attirantes [KAR 04] :

- Par leur nature de recherche discrète, ils pourraient être appliqués facilement aux fonctions continues aussi bien qu'aux fonctions discontinues.
- De plus, bien que les AGs ne puissent pas fournir la solution mathématiquement exacte pour un problème d'optimisation donné, ils devancent habituellement les techniques basées-gradient dans l'obtention des optima globaux et d'où évitent d'être piégés dans les locaux.

Programmation évolutionnaire

Les techniques de la programmation évolutionnaire sont développées par Fogel à la fin des années 1980s. Dans la programmation évolutionnaire, les chromosomes sont représentés sous forme de machines à état fini (MEFs). L'objectif est d'optimiser ces MEFs afin de fournir une représentation significative de comportement basé sur l'interpolation du symbole. Typiquement, la programmation évolutionnaire n'utilise pas d'opérateurs de croisement, le principal processus est l'opération de mutation qui est utilisée pour créer aléatoirement la nouvelle population. Il existe cinq opérateurs possibles de mutation [KAR 04] :

- Modifier un symbole de sortie.
- Modifier une transition d'état.
- Ajouter un état.
- Supprimer un état.
- Changer l'état initial.

Un de ces opérateurs de mutation est sélectionné en se basant sur une distribution de mutation (habituellement une distribution de probabilité). En outre, plus qu'un opérateur de mutation peut être appliqué à un parent.

Programmation génétique

La programmation génétique applique essentiellement un AG pour évoluer des programmes informatiques logiciels ou matériels [SAA 08]. Les individus de la population sont les arbres d'analyse de programmes informatiques où les valeurs de fitness sont mesurées en les exécutant [KAR 04]. Un programme génétique peut être considéré comme un sous-ensemble d'algorithmes génétiques avec la différence demeure dans la représentation des solutions. Un PG est habituellement implémenté en utilisant les quatre étapes suivantes :

- Initialiser la population d'individus (arbres d'analyse de programmes informatiques) de fonctions et terminaux. Ces fonctions et terminaux sont l'alphabet des programmes informatiques.
- Exécuter chaque programme et lui assigner une valeur de fitness selon son comportement pour résoudre le problème.
- Générer la nouvelle population des programmes informatiques comme suit:
    a) Copier le programme informatique qui a la meilleure valeur de fitness.
    b) Créer un nouveau programme informatique en utilisant l'opérateur de mutation.
    c) Appliquer un opérateur du croisement pour créer de nouveaux programmes informatiques.
- Le programme avec la plus grande valeur de fitness est considéré le résultat de la programmation génétique.

Le point de la mutation est choisi aléatoirement. La mutation dans la programmation génétique a deux types :

- Dans le premier, une fonction peut remplacer seulement une fonction et un terminal peut remplacer seulement un terminal.
- Dans le deuxième type, un sous arbre entier de la population de l'arbre d'analyse peut remplacer un autre sous arbre.

Dans l'opération du croisement, deux programmes parentaux sont choisis de façon probabiliste à partir de la population en se basant sur leurs valeurs de fitness. Après que le point de croisement a été déterminé aléatoirement pour les deux chromosomes, un sous arbre du premier est substitué avec un sous arbre du second. En programmation génétique, l'opération du croisement est un processus essentiel pour introduire la diversité à la population.

1.4.3. Logique Floue (Fuzzy Logic) LF

La logique floue est une extension de la logique classique qui permet la modélisation des imperfections des données et se rapproche dans une certaine mesure de la flexibilité du raisonnement humain.

En logique classique, les décisions sont binaires : soit vraies, soient fausses. C'est sur ce point que la logique floue va se distinguer de la logique classique. En logique floue, une décision peut être à la fois vraie et fausse en même temps, avec un certain degré d'appartenance à chacune de ces deux croyances.

1.4.3.1. Logique Floue

La logique floue a été développée en 1965 par Lotfi Zadeh. Elle est née de la constatation que la logique binaire qui tient compte seulement de deux états représentés par 0 et 1 (un élément est soit complètement à l'intérieur ou complètement à l'extérieur d'un ensemble) ne permet pas une

flexibilité suffisante pour modéliser la plupart des problèmes du monde réel caractérisés principalement par l'incertitude, l'imprécision, le vague et la vérité partielle. Elle s'appuie sur la théorie mathématique des ensembles flous.

Grâce à l'utilisation de niveaux d'appartenance qui se trouvent entre 0 et 1, la théorie des ensembles flous qui est à la base de la logique floue et des systèmes flous permet une extension et généralisation de la théorie des ensembles classiques pour la prise en compte d'ensembles définis de façon imprécise. Elle se réduit effectivement à la théorie des ensembles classiques dans le cas où les fonctions d'appartenance considérées prennent des valeurs binaires. [ZAD 65] montre un modèle complet de propriétés et de définitions formelles de cette théorie élaboré par son inventeur Lotfi Zadeh.

La logique floue est basée sur trois concepts principaux :

- Les ensembles flous,
- Les variables linguistiques, et
- Les distributions de possibilité.

Ensembles Flous

Le concept d'ensembles flous a été proposé en premier par Zadeh [ZAD 65] comme une méthode pour modéliser l'incertitude dans le raisonnement humain. Un ensemble flou est déterminé uniquement par sa fonction d'appartenance et il est associé à un terme linguistique.

Soit X un ensemble. Un ensemble flou A de X est caractérisé par une fonction d'appartenance μ_A qui associe à chaque élément x de X une valeur réelle dans l'intervalle [0, 1] représentant le degré d'appartenance de x dans A [ZAD 65] :

$$\mu_A : X \rightarrow [0, 1]$$

Les fonctions mathématiques différentes peuvent être utilisées pour représenter μ_A. A la fois les fonctions discrètes et continues sont appropriées pour ce but. Elle peut être par exemple triangulaire, trapézoïdale, gaussienne ou sigmoïdale. Un expert humain doit décider quelle fonction sera employée pour le problème spécifique.

Variables linguistiques

L'expression « variable linguistique » a été introduite par Zadeh et signifie une variable dont les valeurs sont des termes linguistiques en langage naturel ou artificiel. Par exemple, la taille d'un objet est une variable linguistique dont la valeur peut être « petite », « moyenne », et « grande ». Une variable linguistique est donc une variable prenant ses valeurs dans un ensemble de termes linguistiques dont chacun correspond à un sous-ensemble flous. La fonction d'appartenance de l'ensemble flou assure le traitement numérique des variables linguistiques.

Distributions de possibilité

Quand un ensemble flou est assigné à une variable linguistique, il impose une contrainte élastique, appelée une distribution de possibilité, sur les valeurs possibles de la variable.

La figure suivante montre différentes distributions prédéfinies de possibilité :

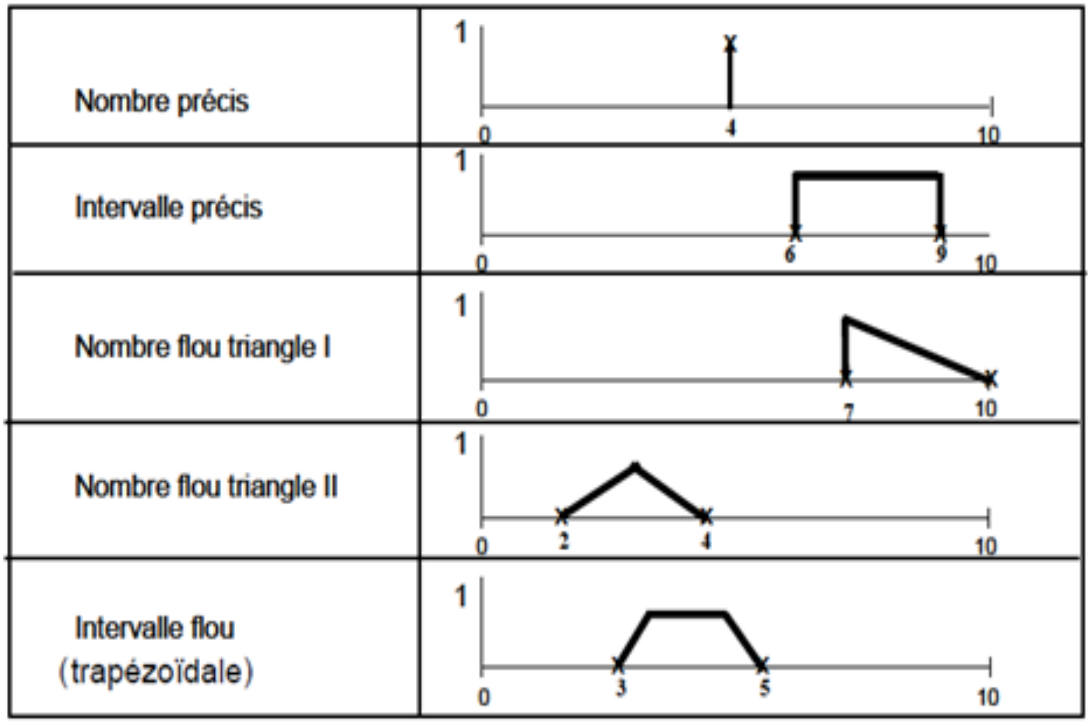


Figure 1.6 : Différentes distributions de possibilité

1.4.3.2. Systèmes flous

L'utilisation de la logique floue est précieuse dans les problèmes de machines intelligentes à cause de son incorporation du raisonnement approximatif qui est l'objectif des systèmes flous basés sur les règles floues et l'inférence floue. Un système flou est défini par trois composants principaux : les variables d'entrée et de sortie floues définies par leurs valeurs floues, un ensemble de règles floues et un mécanisme d'inférence flou. Le secret pour le succès des systèmes flous est qu'ils sont faciles à implémenter, faciles à maintenir, faciles à comprendre, robustes et pas chers.

Le raisonnement approximatif signifie le raisonnement sur des propositions imprécises [ZIL 01] dites propositions floues, c.-à-d., les propositions qui contiennent des variables floues et des valeurs floues, par exemple, "la température est élevée", "la hauteur est petite". Les valeurs de vérité pour les propositions floues ne sont pas VRAI / FAUX seulement, comme c'est le cas dans la logique propositionnelle booléenne, mais inclut toute la graduation entre deux valeurs extrêmes. Le raisonnement approximatif est un mécanisme d'inférence pour dériver des conclusions à partir d'un ensemble de règles floues et une ou plusieurs conditions.

Comme montré dans Fig 1.7, le fonctionnement d'un système flou se décompose en trois étapes distinctes :

- Fuzzification,
- Inférence floue, et
- Défuzzification.

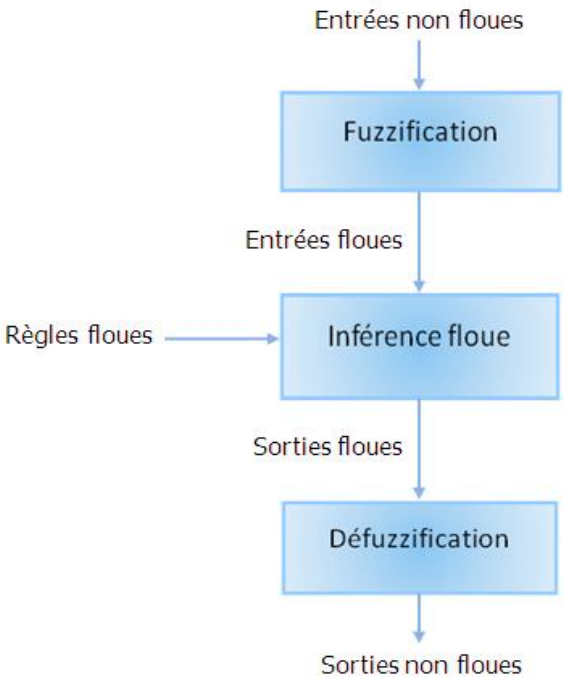


Figure 1.7 : Fonctionnement d'un Système Flou

Fuzzification

La fuzzification consiste à transformer les entrées non floues en sous-ensembles flous. Elle est effectuée en utilisant les fonctions d'appartenance définies dans le système pour déterminer le degré d'appartenance de chaque variable d'entrée à chaque sous-ensemble. Ces fonctions d'appartenances utilisent en général entre trois et sept sous-ensembles flous pour chaque variable.

Inférence floue

Les degrés d'appartenance de chaque variable à chaque sous-ensemble flou permettent d'appliquer les règles floues. Ces dernières traitent les valeurs floues comme, par exemple, "élevé", "froid", "très bas", etc. Ces concepts flous sont représentés habituellement par leurs fonctions d'appartenance.

Une règle floue peut être considérée comme une structure syntactique de la forme:

SI antécédent ALORS conséquent;

Bien que le conséquent d'une règle basée sur la logique booléenne peut être seulement une valeur vraie ou fausse, le conséquent d'une règle floue peut supposer une valeur qui se trouve entre 0 et 1 en se basant sur la notion de fonction d'appartenance [SAA 08].

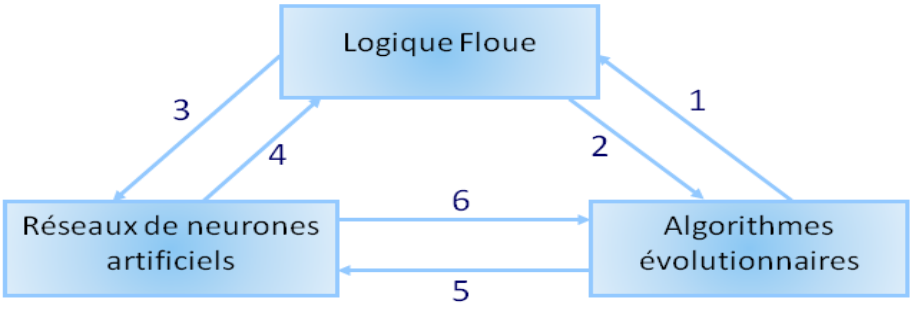
Défuzzification

Cette étape s'effectue toujours à l'aide des fonctions d'appartenance. Elle consiste à transformer les sous-ensembles flous en sorties non floues pour déterminer les valeurs précises qui représentent le mieux les distributions de possibilité. A partir des degrés d'appartenance, on obtient autant de valeurs qu'il y a d'états. Pour déterminer la valeur précise à utiliser, on doit

utiliser une méthode de défuzzification. Parmi les méthodes de défuzzification les plus répandues : la méthode du centre de gravité, la méthode du maximum (premier maximum, dernier maximum, centre maximum). Le choix de la méthode appropriée dépend des propriétés de l'application.

1.5. Intégration de méthodes du soft-computing

Cette section s'intéresse aux combinaisons entre les RNAs, les AEs et la LF dont l'intérêt est la conception des systèmes hybrides intelligents basés sur ces méthodes. La figure 1.8 offre une illustration de ce qu'est le soft computing en termes d'outils intelligents où chaque connexion constitue un type de combinaison.



1 : Systèmes flous évolutionnaires
2 : Algorithmes évolutionnaires flous
3 : Réseaux de neurones flous
4 : Systèmes flous neuronaux
5 : Réseaux de neurones évolutionnaires
6 : Algorithmes évolutionnaires neuronaux

Figure 1.8 : Connexions du soft computing entre la logique floue, les réseaux de neurones et les algorithmes génétiques [MUS 98].

Le soft computing n'est pas un mélange, mais plutôt, c'est une association dans laquelle chacun des éléments contribue une méthodologie distincte pour résoudre des problèmes dans son domaine. Les contributions principales de la logique floue sont le traitement de l'imprécision, le raisonnement approximatif, la granulation floue de l'information et le calcul avec les mots. Celles des réseaux de neurones sont l'identification du système, l'apprentissage et l'adaptation. Et enfin, les algorithmes génétiques contribuent à la recherche aléatoire systématisée, le réglage et l'optimisation. Dans cette perspective, les principales méthodologies du SC sont complémentaires plutôt que compétitives. Pour cette raison, il est fréquemment avantageux de les utiliser en combinaison plutôt qu'exclusivement, pour mener aux "systèmes intelligents hybrides".

Ces méthodologies obtiennent leur inspiration à partir de paradigmes différents, ce sont des systèmes nerveux biologiques pour les réseaux de neurones artificiels, la sélection dans les populations naturelles pour les algorithmes évolutionnaires, et le raisonnement approximatif pour les systèmes flous [TET 01].

D'un point de vue fondations et sources d'inspiration, ces trois éléments principaux du Soft Computing ne partagent pas les mêmes caractéristiques et ont des préoccupations différentes.

Cependant, d'un point de vue pragmatique et orienté-objectif, il y a une compatibilité sous-jacente d'intentions dans leur approche commune pour la résolution de problèmes qui est suffisante pour justifier de traiter le Soft Computing comme une discipline de résolution de problèmes c.à.d. on peut remarquer une complémentarité qui justifie leur intégration. La logique floue est une façon formellement solide d'inclure et de traiter l'imprécision dans les données et le raisonnement. Les réseaux de neurones artificiels admettent aussi un degré d'imprécision d'un type différent et aux niveaux différents : les données utilisées pour apprendre les réseaux peuvent être bruitées jusqu'à un certain point sans affecter l'apprentissage trop. En plus, un réseau de neurones appris devrait être capable de fonctionner en présence de la variabilité des données et le pouvoir parfait de discrimination n'est pas exigé. Les algorithmes évolutionnaires, une fois un codage convenable pour les solutions candidates a été trouvé, marchent sans connaissance explicite de la structure du problème et ont tendance à trouver des solutions plutôt satisfaisantes qu'optimales.

1.5.1. Systèmes flous neuronaux

Ce type d'intégration consiste à doter le système flou des capacités d'apprentissage neurales pour apprendre des fonctions d'appartenance ou des règles pour un système flou donné.

Les algorithmes neuronaux adaptent les systèmes flous :

- hors-ligne (les algorithmes neuronaux apprennent les fonctions d'appartenance ou les règles ou les deux, une fois pour toutes) ;
- en-ligne (les algorithmes neuraux sont utilisés pour adapter les fonctions d'appartenance ou les règles du système flou ou les deux, quand le système fonctionne) ;

Types

- Mémoire associative adaptatives.
- Cartes caractéristiques auto-organisatrices.
- Apprentissage d'ensembles flous pour les systèmes de type Sugéno.
- Fuzzy ART et Fuzzy ARTmap.

1.5.2. Réseaux de neurones flous

Dans la fuzzification des réseaux de neurones, le flou peut être introduit aux niveaux différents : le niveau de poids, le niveau de fonction de transfert ou le niveau algorithme d'apprentissage.

1.5.3. Systèmes flous évolutionnaires

Les systèmes flous à base de règles incluent des règles pour diriger le processus de décision et des fonctions d'appartenance pour convertir les termes linguistiques en valeurs numériques précises nécessitées par l'ordinateur. L'ensemble de règles peut être accumulé à partir de la connaissance et l'expérience d'un expert humain, et les fonctions d'appartenance sont choisies par le développeur du système pour représenter la conception de l'expert humain des termes linguistiques. Les AEs peuvent être utilisés pour générer des règles floues et régler les fonctions d'appartenance des ensembles flous.

Une des méthodes hybrides les plus fréquemment décrites consiste à l'utilisation des algorithmes évolutionnaires pour l'optimisation des systèmes flous.

Lors de la conception d'un système flou, nous sommes confrontés à la tâche de sélectionner une base de règles appropriée. Elle prend du temps, de l'expérience et de la connaissance de l'expert.

Ce processus peut être automatisé grâce aux algorithmes évolutionnaires, la flexibilité duquel et l'indépendance du problème résolu est utilisée aux moments suivants de la conception d'un système flou [RUT 08] :

- Lorsque nous avons un système flou à notre disposition et nous nous efforçons d'améliorer l'efficacité de son fonctionnement. Les algorithmes évolutionnaires peuvent être utilisés pour le réglage d'une fonction d'appartenance, c'est à dire changer sa location ou sa forme.
- Quand un ensemble de fonctions d'appartenance des termes linguistiques est défini, nous pouvons générer une base de règles au moyen de méthodes évolutives. Trois approches pour résoudre ce problème sont utilisées - l'approche de Michigan, l'approche de Pittsburgh et l'apprentissage itératif de règle, qui se distinguent par la méthode de codage et de construction d'une base de règles.

Il y a plusieurs façons de concevoir un système flou en utilisant les AEs, selon le degré auquel le système flou dépend de l'évolution.

- La structure totale du système flou peut être évidente dès le début, et tout ce qui est exigé d'un AE est le réglage de certains paramètres.
- Tout à partir de la structure aux paramètres du système flou peut être l'objet de la recherche évolutionnaire.

1.5.3.1. Réglage de Fonction d'Appartenance

La sélection de fonctions d'appartenance performantes est la phase la plus prenante de temps de développement du système flou. Les changements dans les fonctions d'appartenance changent la performance du système parce qu'elles déterminent la contribution que chaque règle fait pour le choix de la sortie du système. Donc, la performance d'un système flou est directement liée au choix des fonctions d'appartenance. Pour cette raison les AEs étaient en premier lieu appliqués à la détermination et au réglage de la fonction d'appartenance. Cela réduit le temps de développement requis pour concevoir des contrôleurs flous.

Le problème central dans la conception d'un algorithme évolutionnaire pour le réglage de fonction d'appartenance est le codage. Cela est passé selon les suppositions qui sont faites a priori sur les formes des fonctions d'appartenance.

Plusieurs fonctions populaires peuvent être distinguées [RUT 08] :

- triangle isocèle - les fonctions sont codées en un chromosome au moyen de deux points extrêmes du triangle: a et b.
- triangle asymétrique – location et forme du triangle asymétrique est codée au moyen de trois paramètres: a, b et c. En comparaison avec le triangle isocèle, la location du sommet central est de plus définie. Au lieu des coordonnées des deux points extrêmes, nous pouvons aussi donner les distances au point central.
- trapézoïde - un trapézoïde est caractérisé par quatre points, mais il faut rappeler, comme dans le cas des triangles, que les points: a, b, c et d, qui représentent le trapézoïde, doivent remplir la condition a <b <c <d.
- également d'autres fonctions peuvent être distinguées, par exemple la fonction gaussienne qui peut être décrite au moyen des deux paramètres: le centre $\bar{x}$ et la largeur σ. Les autres fonctions utilisées incluent la fonction radiale et la fonction sigmoïde (pour les ensembles flous extrêmes qui représentent les valeurs des variables linguistiques).

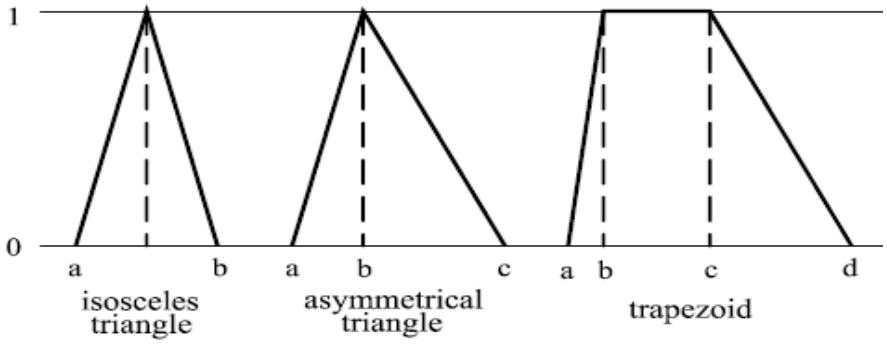


Figure 1.9 : Différentes fonctions d'appartenance avec des points caractéristiques [RUT 08]

Après que la représentation appropriée des ensembles flous dans le chromosome a été sélectionnée, l'algorithme évolutionnaire fonctionne sur la population d'individus (de chromosomes contenant des formes codées de fonctions d'appartenance du système flou) selon le cycle évolutionnaire qui peut comprendre les étapes suivantes :

1. Coder chacun des individus de la population consiste à recréer l'ensemble de fonctions d'appartenance et construire un système flou approprié. La base de règle est prédéfinie.
2. Le fonctionnement d'un système flou est évalué sur la base de la différence (erreur) entre les réponses du système et les valeurs désirées. Cette erreur définit la fitness de l'individu.
3. La reproduction des individus et l'application des opérateurs génétiques. Des techniques spécifiques dépendent de l'AE choisi, comme AG et SEs, sont caractérisées par différents mécanismes de sélection et de recombinaison.
4. Si le critère d'arrêt n'est pas atteint, nous passons au point 1.

1.5.3.2. Evolution des Règles

Il y a des plusieurs stratégies de codage appropriées pour la conception évolutionnaire de règles pour les systèmes flous à base de règles telles que le codage tabellaire (par une matrice), un gène dans le génotype pour chaque règle possible, ID de règle, …etc.

Nous pouvons distinguer trois méthodes d'utilisation des AEs dans le processus de génération de règles du système floue : l'approche de Michigan, l'approche de Pittsburgh et le processus d'apprentissage itératif [RUT 08].

L'approche de Michigan

Une caractéristique de cette approche est que des règles particulières sont codées dans des chromosomes séparés. Elle utilise le concept dit de systèmes classifieur. Chacun des individus représente une règle codée. Tous ou seulement une partie des chromosomes de la population sont considérés comme la base de règles qui est recherchée.

Approche de Pittsburgh

L'approche de Pittsburgh est une méthode de codage qui correspond plutôt au fonctionnement des algorithmes évolutionnaires. La base de règles entière est codée en un chromosome. Ainsi, la solution recherchée est trouvée dans le meilleur individu adapté.

Une caractéristique de cette méthode est de placer toute la solution en un seul chromosome. Chaque individu représente un ensemble complet de règles. Les individus concurrencent les uns

avec les autres, les faibles meurent, les forts survivent et se reproduisent. Cela se produit sur la base des opérateurs de sélection et de croisement et mutation. Il est possible de garder l'équilibre entre l'exploration de nouvelles solutions et l'exploitation de la meilleure solution. Le fonctionnement de l'approche de Pittsburgh ne diffère pas du fonctionnement de l'AG classique. Malheureusement, il a des défauts, aussi bien. Le chromosome codant toutes les règles est beaucoup plus grand que dans la méthode de Michigan, qui prolonge la durée de fonctionnement de l'algorithme évolutionnaire, de plus la chance de trouver une solution correcte également diminue. Dans le plus simple algorithme de rechercher une base de règles floue, nous devons déterminer a priori le nombre maximal de règles. L'ensemble de toutes les règles du système flou est stocké sous la forme d'une chaîne de longueur constante avec la division en sous-chaînes dans lesquelles des règles particulières sont codées.

Apprentissage itératif de règle

La troisième approche pour la recherche d'une base de règles est celle dite apprentissage itératif de règle. Elle combine les meilleures caractéristiques des approches de Michigan et de Pittsburgh. Le concept de coder une règle par chromosome a été utilisé ici, mais la création de la base entière se fait graduellement. La base de règle finale est constituée des meilleurs individus étant le résultat de fonctionnement des activations ultérieures de l'algorithme évolutionnaire.

L'apprentissage itératif de règle a été développé comme une tentative de combiner les meilleures caractéristiques des approches de Michigan et de Pittsburgh décrites précédemment. Cette méthode est caractérisée par le fait que chaque chromosome dans la population représente une seule règle, comme dans l'approche de Michigan. En revanche, ce qui est caractéristique de l'approche de Pittsburgh est le fait que seulement le meilleur individu de la population est choisi. La base de règles complète peut être obtenue en répétant l'algorithme évolutionnaire plusieurs fois, ajoutant à chaque fois la meilleure règle à la base de règles jusqu'à ce que la solution entière soit trouvée. L'algorithme de génération de la base de règles basé sur l'apprentissage itératif de règle peut être présenté comme suit :

1. Appliquer un algorithme évolutionnaire pour trouver une règle. Comme la fitness du chromosome codant une règle nous pouvons prendre par exemple le nombre de données d'apprentissage classées correctement ou la simplicité de règle.
2. Attacher la règle à l'ensemble final de règles.
3. Évaluer l'efficacité de l'ensemble des règles, particulièrement en prenant en considération la règle trouvée récemment.
4. Si l'ensemble des règles générées est satisfaisant pour le problème résolu, finir le fonctionnement de cet algorithme. Sinon, passer au point 1.

La génération des règles dans les méthodes de Pittsburgh et de Michigan a consisté à obtenir simultanément la base de règles entière. Dans l'apprentissage itératif de règles, des règles particulières se développent indépendamment les unes des autres sans aucune information sur celles générées précédemment. Il peut en résulter dans la répétition de règles ou de leur exclusion mutuelle. Par conséquent, après que le fonctionnement de l'algorithme est terminé, des procédures de simplification de la base de règle sont souvent appliquées. Cependant, déjà lorsque l'algorithme est en fonctionnement, nous pouvons empêcher des règles identiques d'apparaître. La méthode est basée sur l'enlèvement de ces données de la séquence d'apprentissage qu'elles ont utilisé pour le but de l'apprentissage correct des règles trouvées préalablement.

1.5.4. Réseaux de neurones évolutionnaires

Un réseau de neurones évolutionnaire est un réseau de neurones avec certains composants génétiques. Ce type de combinaison mène aux systèmes intelligents évolutifs. Cette section étudie différents types d'utilisation des algorithmes évolutionnaires (AEs) dans les RNAs y compris utiliser les AEs pour évoluer les poids de connexion (déterminer l'ensemble optimal des poids de connexion), architectures (déterminer la topologie optimale du réseau), règles d'apprentissage, et caractéristiques d'entrée du RNA.

L'utilisation des AEs pour évoluer les RNAs peut surmonter le problème du choix empirique des valeurs des paramètres qui peuvent affecter la performance d'un RNA sur un problème donné, tel que la topologie du réseau, les poids de connexion, les détails des règles d'apprentissage et de la fonction d'activation, et les ensembles de données d'apprentissage. Ils peuvent être utilisés pour accomplir plusieurs tâches tel que l'apprentissage des poids de connexion, la conception de l'architecture, l'adaptation de la règle d'apprentissage, la sélection de caractéristiques d'entrée, l'initialisation des poids de connexion, l'extraction de règle à partir de RNA, …etc.

Dans cette section on va étudier comment utiliser les AEs pour évoluer les poids de connexion, les architectures, les règles d'apprentissage, et les caractéristiques d'entrée du RNA.

1.5.4.1. Evolution des poids de connexion

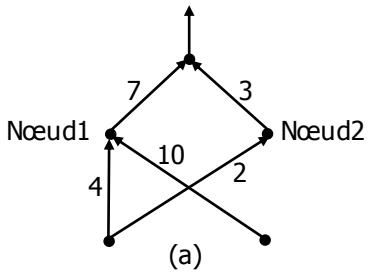
L'évolution des poids de connexion peut surmonter les défauts des algorithmes d'entraînement basés-descente-gradient [TET 01]. Elle constitue une approche adaptative et globale à l'apprentissage et consiste à formuler le processus d'entraînement comme étant l'évolution des poids de connexion dans l'environnement déterminé par l'architecture et la tâche d'apprentissage. Son objectif est de trouver globalement un ensemble optimal de poids de connexion pour un RNA avec une architecture fixe en utilisant les AEs [XIN 99].

L'évolution des poids de connexion dans les RNAs consiste en deux phases majeures [XIN 99] :

- La première phase est de décider la représentation des poids de connexion, c.-à-d., soit sous forme de chaînes de bits ou pas ; et l'AE utilisé pour évoluer les poids du RNA.
- La deuxième est le processus évolutionnaire simulé par un AE, dans lequel les opérateurs de recherche tels que le croisement et la mutation doivent être décidés conjointement avec le schème de représentation.

Des représentations (codage) différentes et opérateurs de recherche différents peuvent mener à des performances d'entraînement tout à fait différentes.

Le chromosome représentant le RNA peut être une liste de poids [TET 01]. Dans le codage binaire, chaque poids de connexion est représenté comme une chaîne de caractères binaires à être décodée en valeur réelle. Cependant, tant que le nombre de connexions augmente, la longueur de la chaîne de caractères augmente à une dimension qui ralentit sévèrement le processus évolutionnaire et ensuite la recherche. Pour alléger ce problème, le codage par nombres réels peut être utilisé pour représenter directement les valeurs des poids de connexion du RNA, c.-à-d., un nombre réel par poids de connexion. Fig 1.10 donne un exemple des deux schèmes de représentation.



(a)

0100 1010 0010 0000 0111 0011

(b)

(4.0, 10.0, 2.0, 0.0, 7.0, 3.0)

(c)

Figure 1.10 : (a) Un RNA avec poids de connexion montré; (b) Une représentation binaire des poids, supposant que chaque poids est représenté par quatre bits, le poids 0000 n'indique aucune connexion entre deux nœuds. (c) Une représentation en nombres réels des poids

Le processus évolutionnaire entier peut être décrit par le pseudo-code suivant :

génération = 0
Assigner des vecteurs de poids aléatoires à la population initiale des RNAs
tant que la condition de terminaison n'est pas satisfait **faire**
    génération = génération + 1
    Décoder chaque individu dans la génération courante en un ensemble de poids de connexion et construire le RNA correspondant avec ces poids.
    Calculer la fitness de chaque RNA sur les données de test
    Sélectionner des parents (RNAs) pour la reproduction en se basant sur leur fitness.
    Appliquer des opérateurs de recherche, tel que le croisement et/ou la mutation, aux parents (RNAs sélectionnés) pour générer les fils qui forment la prochaine génération.
fin tant que

L'évolution s'arrête lorsque la fitness est plus grande qu'une valeur prédéfinie (c.-à-d., l'erreur de l'apprentissage est plus petite qu'une certaine valeur).

La fitness d'un RNA est évaluée en calculant l'erreur cumulée sur les données d'entraînement (par certaine variante d'apprentissage de rétro-propagation), plus l'erreur est élevée plus la fitness est basse. Cependant, il est parfois utile d'inclure d'autres critères tels que la complexité du réseau, le temps d'entraînement, …etc.

La mutation consiste à choisir aléatoirement un neurone non d'entrée et changer son poids par une valeur aléatoire. Le croisement prend deux chaînes de caractères de vecteur poids parent et construit un seul enfant en sélectionnant aléatoirement un des poids du parent pour chaque connexion d'unité non d'entrée à partir de la couche précédente. Les opérateurs de croisement ont tendance à détruire les détecteurs de caractéristiques utiles évolués, pour surmonter ce problème il est utile de mettre les poids de connexion du même nœud ensemble dans la représentation de l'individu. Les chromosomes à valeurs réelles exigent des opérateurs de recherche spéciaux parce que le croisement et la mutation binaires traditionnels ne peuvent plus être appliqués directement à des vecteurs réels.

1.5.4.2. Evolution des architectures

L'architecture d'un RNA inclut sa structure topologique, c.-à-d. la connectivité, et la fonction de transfert de chaque nœud dans le RNA, et détermine la capacité du traitement de l'information du RNA [XIN 99]. Etant donné une tâche d'apprentissage, un RNA avec seulement peu de connexions et nœuds linéaires ne peut pas être capable d'exécuter la tâche dû à sa capacité limitée, alors qu'un RNA avec un grand nombre de connexions et nœuds non linéaires peut sur-apprend le bruit dans les données d'entraînement et échoue à avoir une bonne capacité de généralisation. L'objectif de l'évolution des architectures est de trouver une architecture optimale du RNA pour la tâche considérée, c.-à-d. déterminer le nombre approprié de nœuds et leurs interconnexions.

Similairement à l'évolution des poids de connexion, deux phases majeures sont impliquées dans l'évolution des architectures :

- Décider le schème de représentation des architectures, c.-à-d. codage direct ou indirect, et l'AE utilisé pour évoluer les architectures RNA.
- Le processus évolutionnaire simulé par un AE.

La représentation et les opérateurs de recherche utilisés dans les AEs sont deux choix importants dans l'évolution des architectures. Il est montré que les AEs qui comptent sur les opérateurs de croisement ne fonctionnent pas très bien dans la recherche d'une architecture du RNA optimale. Raisons et résultats empiriques sont donnés dans [XIN 99].

Deux schèmes de représentation existent pour coder les architectures RNAs : le schème de codage direct dans lequel tous les détails d'une architecture RNA sont codés dans le chromosome, et le schème de codage indirect dans lequel une description condensée du réseau est évoluée c.-à-d. seulement les paramètres les plus importants d'une architecture, tel que le nombre de couches cachées et nœuds cachés dans chaque couche sont codés et les autres détails sur l'architecture sont spécifiés par quelques règles développementales ou par un processus d'apprentissage.

Dans le codage direct, la topologie de connexion est représentée par une matrice d'adjacence A : N x N, où N est le nombre de nœuds, a_{ij} = 1 signifie qu'il y a une connexion entre les unités i et j et a_{ij} = 0 signifie qu'il n y a aucune connexion. Un individu dans une population d'architectures est la chaîne de caractères résultant de la concaténation de lignes consécutives de la matrice. Ce codage est utile seulement pour les petites architectures.

Plusieurs chercheurs ont concentré leurs efforts sur les techniques de développement ou d'augmentation des réseaux de neurones (codages grammaticaux), plutôt que chercher une description complète du réseau au niveau individuel. Fig 1.11 montre un exemple d'une grammaire de génération de graphe. En appliquant successivement les productions à partir d'un symbole de début donné on finit avec un tableau binaire qui peut être interprété comme la matrice d'adjacence d'un graphe dirigé représentant un réseau de neurones.

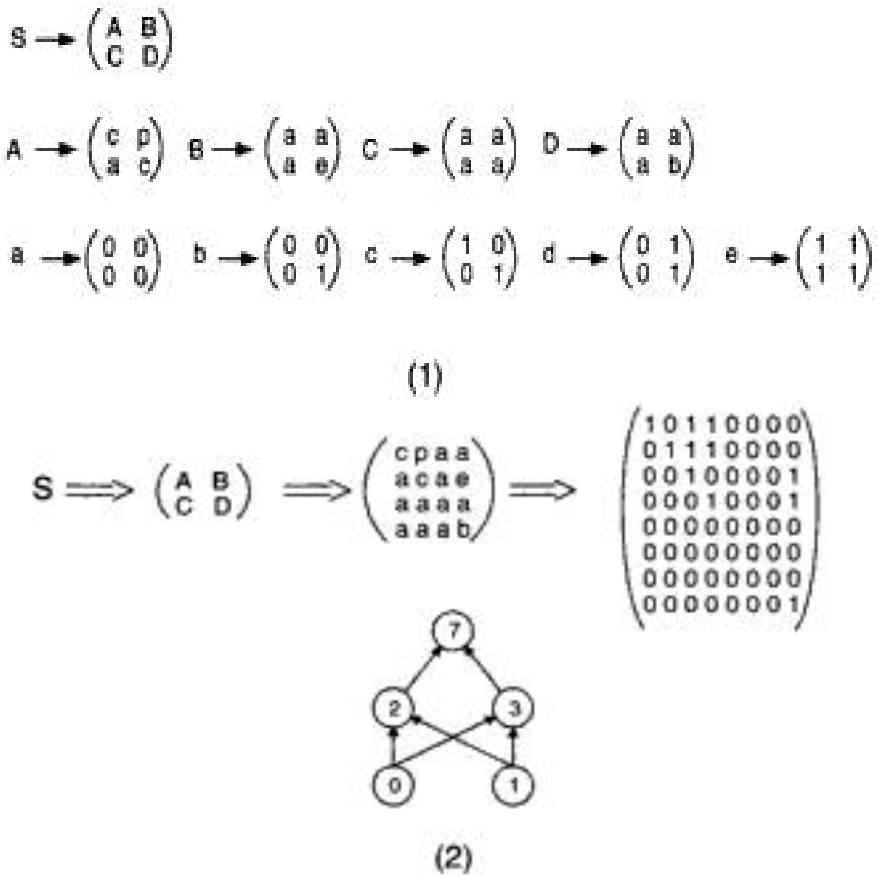


Figure 1. 11 : Exemple d'utilisation des règles de production de génération de graphe pour le développement d'un réseau feedforward ou-exclusif. La grammaire est représentée dans la partie (1). La partie (2) de la figure montre l'application consécutive des règles de la grammaire commençant par le symbole S et produisant un réseau pour le calcul de la fonction Booléenne ou-exclusif.

L'algorithme évolutionnaire fonctionne selon le pseudo-code suivant:

génération = 0
Initialiser la population d'individus (chromosomes représentant les architectures)
tant que la condition de terminaison n'est pas satisfait **faire**
génération = génération + 1
Décoder chaque individu dans la génération courante en une architecture. Si le schème de codage indirect est utilisé, les détails supplémentaires de l'architecture sont spécifiés par quelques règles développementales ou par un processus d'apprentissage.
Calculer la fitness de chaque architecture en l'entraînant sur les données de test
Sélectionner des parents (architectures RNAs) pour la reproduction en se basant sur leur fitness.
Appliquer des opérateurs de recherche, tel que le croisement et/ou la mutation, aux parents (architectures RNAs sélectionnées) pour générer les fils qui forment la prochaine génération.
fin tant que

Le cycle s'arrête quand un RNA satisfaisant est trouvé.

La fitness d'un RNA est évaluée en calculant l'erreur par certaine variante d'apprentissage de rétro-propagation, plus l'erreur est élevée plus la fitness est basse. Des critères tels que la complexité du réseau et le temps d'entraînement peuvent être ajoutés.

Bien que la plupart des travaux traitent l'évolution de la structure topologique d'une architecture, certains chercheurs ont traité l'évolution des fonctions de transfert des nœuds (voir [XIN 99]).

L'évolution des architectures sans toute information du poids a des difficultés dans l'évaluation correcte de la fitness. Un problème majeur avec l'évolution des architectures sans poids de connexion est l'évaluation bruitée de la fitness. Une façon d'alléger ce problème est d'évoluer les architectures RNA et les poids de connexion simultanément. Dans ce cas, chaque individu dans une population est un RNA complètement spécifié avec l'information du poids complète. Séparer l'évolution des architectures et des poids de connexion peut rendre l'évaluation de fitness inexacte et tromper l'évolution. L'évolution simultanée des architectures et poids de connexion RNA généralement produit des meilleurs résultats.

1.5.4.3. Evolution des règles d'apprentissage

Quand il y a peu de connaissance sur l'architecture la plus appropriée pour un problème donné, il devient très utile de fournir au RNA la capacité d'adapter automatiquement sa règle d'apprentissage selon son architecture et la tâche à exécuter plutôt qu'elle soit conçue et fixée manuellement. L'évolution peut être utilisée pour permettre à un RNA d'adapter sa règle d'apprentissage à son environnement dont l'objectif est de trouver une règle d'apprentissage optimale pour la tâche considérée c.-à-d. améliorer la capacité d'apprentissage par évolution.

La recherche sur l'évolution des règles d'apprentissage est importante pour :

- fournir une façon automatique d'optimiser les règles d'apprentissage,
- modéliser la relation entre l'apprentissage et l'évolution,
- modéliser le processus créatif de nouvelles règles d'apprentissage.

L'ajustement adaptatif des paramètres de l'algorithme de rétro-propagation (tel que le taux et la vitesse de l'apprentissage) par l'évolution pourrait être considéré comme la première tentative de l'évolution des règles d'apprentissage. Cependant, l'évolution de paramètres algorithmiques ne touche presque pas la partie fondamentale de l'algorithme d'entraînement, c.-à-d., sa règle d'apprentissage ou règle de mise à jour des poids. Pour améliorer grandement l'adaptabilité d'un RNA à un environnement dynamique, l'évolution des règles d'apprentissage doit travailler sur le comportement dynamique du RNA.

Un schème de représentation qui peut être utilisé pour coder le comportement dynamique d'une règle d'apprentissage en chromosomes statiques est inexistant. Les contraintes doivent être mises sur le type de comportements dynamiques, c.-à-d., la forme basique des règles d'apprentissage qui sont évoluées.

Une règle d'apprentissage est supposée être une fonction linéaire de variables locales (tel que l'activation du nœud d'entrée, l'activation du nœud de sortie, le poids de connexion courant, etc.)

et leurs produits, et elle est la même pour toutes les connexions dans un RNA. Une règle d'apprentissage peut être décrite par la fonction

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1,i_2,\ldots,i_k=1}^{n} (\theta_{i_1,i_2,\ldots,i_k} \prod_{j=1}^{k} x_{i_j} (t-1)$$

où t est temps, Δw est le changement du poids, x_1, x_2, …, x_n sont des variables locales, et les θ's sont des coefficients réels qui seront déterminés par évolution. En d'autres termes, l'évolution des règles d'apprentissage dans ce cas est équivalente à l'évolution des vecteurs réels θ's. Différents θ's déterminent des règles d'apprentissage différentes. Dû à un grand nombre de termes possibles dans l'équation, qui rendraient l'évolution très lente et irréaliste, seulement peu de termes ont été utilisés dans la pratique selon certaine connaissance biologique ou heuristique.

Il y a trois questions majeures impliquées dans l'évolution des règles d'apprentissage:

1. la détermination d'un sous-ensemble de termes décrits dans l'équation,
2. la représentation de leurs coefficients à valeurs réels en chromosomes, et
3. l'AE utilisé pour évoluer ces chromosomes.

L'algorithme évolutionnaire fonctionne selon le pseudo-code suivant:

génération = 0
Initialiser la population d'individus (chromosomes représentant les règles)
tant que la condition de terminaison n'est pas satisfait **faire**
    génération = génération + 1
    Décoder chaque individu dans la génération courante en une règle d'apprentissage.
    Construire un ensemble de RNAs avec des architectures et des poids de connexion initiaux générés aléatoirement, et les apprendre en utilisant la règle d'apprentissage décodée.
    Calculer la fitness de chaque individu (règle d'apprentissage codée) selon le résultat moyen de l'apprentissage.
    Sélectionner des parents (règles d'apprentissage) à partir de la génération courante selon leur fitness.
    Appliquer des opérateurs de recherche aux parents pour générer les fils qui forment la prochaine génération.
fin tant que

Le processus itératif s'arrête lorsque la population converge ou un nombre maximal prédéfini d'itérations a été atteint.

Si une règle d'apprentissage optimale pour des architectures RNA différentes est à être évoluée, l'évaluation de fitness devrait être basée sur le résultat moyen d'entraînement à partir des architectures RNA différentes afin d'éviter de sur-apprendre une architecture particulière.

Des règles d'apprentissage efficaces peuvent être évoluées à partir de règles générées aléatoirement. À partir d'une population de règles générées aléatoirement, et avec un codage approprié en chromosomes, Chalmers était capable d'évoluer la règle connue de delta et certaines de ses variantes.

1.5.4.4. Evolution des caractéristiques d'entrée

Un grand nombre d'entrées à un RNA augmentent sa taille et donc exigent plus de données d'entraînement et temps d'entraînement plus longs afin d'accomplir une capacité de généralisation raisonnable. Un prétraitement est exigé souvent pour réduire le nombre d'entrées à un RNA. Plusieurs techniques de réduction de dimension y compris l'analyse en composante principale ont été utilisées pour ce but.

Le problème de trouver un ensemble optimal de caractéristiques d'entrée à un RNA peut être formulé comme un problème de recherche. Etant donné un grand ensemble d'entrées potentielles, nous voulons trouver un sous-ensemble optimal ayant le moins nombre de caractéristiques mais la performance du RNA utilisant ce sous-ensemble n'est pas pire que celle du RNA l'ensemble d'entrée entier. Les AEs ont été utilisés pour effectuer telle recherche efficacement.

Dans l'évolution de caractéristiques d'entrée, chaque individu dans la population représente un ensemble de toutes les entrées possibles. Cela peut être implémenté en utilisant un chromosome binaire dont la longueur est la même que le nombre total de caractéristiques d'entrée. Chaque bit dans le chromosome correspond à une caractéristique. "1" indique la présence d'une caractéristique, pendant que "0" indiquent l'absence de la caractéristique. L'évaluation d'un individu est faite en entraînant un RNA avec ces entrées et en utilisant le résultat pour calculer sa valeur de fitness. L'architecture RNA est souvent fixée [XIN 99].

Essentiellement, dans un codage AE du problème, chaque individu dans la population représente une partie des données d'entrée. Le RNA est entraîné avec ces vecteurs et le résultat est une partie de la fitness de l'individu [TET 01].

Les algorithmes évolutionnaires peuvent être utilisés efficacement pour :

- Découvrir automatiquement des caractéristiques importantes à partir de toutes les entrées possibles [XIN 99].
- Découvrir des nouveaux exemples d'entraînement [XIN 99].
- La réduction de dimension de l'ensemble de données d'entrée sans une perte en performance [TET 01].
- La sélection de caractéristiques d'entrées à un RNA. Sinon : l'utilisation de l'ensemble de données entier diminue l'efficacité en ralentissant l'entraînement et peut mener à des pauvres capacités de généralisation [TET 01].
- Le partitionnement de données d'entrée d'un réseau dans un ensemble d'entraînement et de validation. Ce partitionnement est presque toujours fait arbitrairement, bien qu'il puisse influencer la performance du réseau d'une manière significative [TET 01].

[XIN 99] décrit une structure générale de réseaux de neurones évolutifs, qui fournit une base commune pour comparer des modèles différents de RNAs évolutifs.

1.5.5. Algorithmes évolutionnaires neuronaux

Les algorithmes évolutionnaires ont été utilisés avec succès pour optimiser plusieurs paramètres de contrôle. Cependant, il est très consommant du temps et coûteux d'obtenir des valeurs de fitness pour certains problèmes de contrôle comme il est irréaliste d'exécuter un système réel pour chaque combinaison de paramètres de contrôle. Afin d'aller autour de ce problème et rendre

l'évolution plus efficace, les valeurs de fitness sont souvent approximées plutôt que calculées exactement. Les RNAs sont souvent utilisés pour modéliser et approximer un système de contrôle réel dû à leurs bonnes capacités de généralisation. L'entrée à tel RNAs sera un ensemble de paramètres de contrôle. La sortie sera la sortie du système de contrôle à partir de laquelle une évaluation du système entier peut être obtenue facilement. Lorsqu'un AE est utilisé pour rechercher un ensemble optimal de paramètres de contrôle, le RNA sera utilisé dans l'évaluation de fitness plutôt que le système de contrôle réel [XIN 99].

Cette combinaison des RNAs et AEs a deux avantages dans évoluer des systèmes de contrôle. En premier, l'évaluation de fitness basée sur les systèmes de contrôle réels est remplacée par une évaluation rapide de fitness basée sur les RNAs. En deuxième, cette combinaison fournit une évolution sans risque des systèmes de contrôle. Les AEs sont des algorithmes stochastiques. Il est possible que certains paramètres du contrôle pauvres puissent être générés dans le processus évolutionnaire. Ces paramètres pourraient endommager un système de contrôle réel. Si nous utilisons les RNAs pour estimer la fitness, nous n'avons pas besoin d'utiliser le système réel et donc nous pouvons éviter des dommages pour le système réel. Cependant, comment réussir cette approche de combinaison sera dépend en grande partie sur comment les RNAs apprennent et généralisent.

1.5.6. Algorithmes évolutionnaires flous

Dans cette section, nous décrivons l'application de la logique floue pour la construction des AEs.

1.5.6.1. Codage flou pour les algorithmes évolutionnaires

Exemples de techniques de codage flou pour l'AE sont le codage flou des paramètres de l'AE et le codage flou des chromosomes. Ces méthodes utilisent un mappage intermédiaire entre les chaînes de caractères génétiques et les paramètres de l'espace de recherche.

Le codage flou est une méthode indirecte de codage. Il fournit la valeur d'un paramètre sur la base du nombre optimal des ensembles flous sélectionnés et leurs degrés d'appartenance. Il représente la connaissance associée à chaque paramètre. On partitionne chaque paramètre en ensembles flous et, à différentes parties du partitionnement flou, des fonctions d'appartenance de formes différentes sont sélectionnées. L'AE optimise les fonctions d'appartenance et le nombre d'ensembles flous, alors que la valeur véritable du paramètre est obtenue par défuzzification. Le codage floue, avec une combinaison appropriée de fonctions d'appartenance, est capable de trouver des paramètres mieux optimisés que, à la fois, l'AE utilisant des méthodes de codage binaire et la technique de descente de gradient pour l'apprentissage des paramètres des RNAs.

Dans le codage flou, chaque paramètre est codé en deux sections. Dans la première section, les ensembles flous associés à chaque paramètre sont codés en bits dont chacun représente l'ensemble flou sélectionné correspondant. Par exemple, dans le but d'optimiser un système avec deux paramètres X_1 et X_2, nous pouvons donner un partitionnement flou complet de l'intervalle de chaque paramètre comme trois ensembles flous A_i, i = 1, 2, 3. La deuxième section présente les degrés d'appartenance correspondants μ_{Ai} (x_j), évalués à x_j. Le chromosome correspondant utilisant le codage flou est montré dans la Fig 1.12.

Binary coded fuzzy sets						Degrees of membership					
$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mu_{\mathcal{A}_1}(x_1)$	$\mu_{\mathcal{A}_2}(x_1)$	$\mu_{\mathcal{A}_3}(x_1)$	$\mu_{\mathcal{A}_1}(x_2)$	$\mu_{\mathcal{A}_2}(x_2)$	$\mu_{\mathcal{A}_3}(x_2)$
x_1			x_2			x_1			x_2		

Figure 1.12 : La représentation du codage floue d'un chromosome pour un problème d'optimisation à deux paramètres. La première section contient les ensembles flous codés binaire, et la deuxième section contient les degrés d'appartenance correspondants. A_i, i = 1, 2, 3, sont les partitions floues des paramètres

1.5.6.2 Réglage adaptatif de paramètres en utilisant la logique floue

Les AEs adaptatifs ajustent dynamiquement les paramètres de contrôle sélectionnés ou les opérateurs génétiques durant l'évolution. Leur objectif est d'offrir le comportement d'exploration et d'exploitation le plus approprié afin d'éviter le problème de convergence prématurée et améliorer les résultats finals.

L'AG paramétrique dynamique et l'AG adaptatif flou (FAGA) sont des AGs dont les paramètres de contrôle sont ajustés en utilisant la logique floue. L'idée principale est d'utiliser un contrôleur flou avec des entrées comme toute combinaison de mesures de performance actuelles et des paramètres de contrôle actuels de l'AG, et les sorties comme les nouveaux paramètres de contrôle de l'AG.

1.5.7. Systèmes neuro-flous évolutionnaires

Les paramètres d'un réseau de neurones flou peuvent être appris en utilisant les AEs. Tels systèmes sont appelés systèmes neuro-flous évolutifs.

Dans les systèmes neuro-flous, l'apprentissage des paramètres emploie généralement la méthode de descente de gradient, qui peut converger vers un minimum local. Un algorithme d'optimisation global tel que les AEs peut remplacer la technique de descente de gradient pour les systèmes d'entraînement neuro-flous. Les AEs peuvent optimiser à la fois l'architecture et les paramètres des systèmes neuro-flou. En d'autres termes, les AEs sont utilisées pour évoluer à la fois les règles floues et leur fonctions d'appartenance et poids de connexion respectifs.

Le réseau de neurones flou a l'architecture d'une représentation standard à deux niveaux OR/AND des fonctions booléennes de symboles. Le réseau de neurones flou peut être entraîné en utilisant une procédure en trois phases. L'optimisation architecturale est réalisée en utilisant la programmation génétique, alors que les paramètres qui s'ensuivent sont optimisés par l'apprentissage basé gradient. Une collection d'ensembles flous est d'abord sélectionnée, et reste inchangée durant les phases successives de développement du modèle. La programmation génétique est appliquée pour chercher une architecture optimale du modèle flou. Après que l'architecture est sélectionnée, le réseau est sujet à un certain raffinement paramétrique en optimisant les poids en utilisant l'apprentissage basé gradient.

Le système génétique flou neuronal (FuGeNeSys) et l'extracteur de règle floue génétique (GEFREX) sont des modèles synergiques de la logique floue, les réseaux de neurones, et l'AG. Ils sont les deux des méthodes générales pour l'apprentissage supervisé flou des systèmes MIMO. Le modèle flou de Mamdani est utilisé, et la méthode de défuzzification moyenne pondérée est

appliquée. Chaque individu de la population est constituée d'un ensemble de Nr règles, avec chaque règle comprenant n entrées (antécédents) et m sorties (conséquents). L'AG fine-grained est utilisé, dans lequel les individus sont généralement placés sur une grille planaire. A la fois, les sélections globales et locales sont utilisées. La sélection globale est proportionnelle à la fitness et identifie le centre de la sous-population (deme), alors que la sélection locale utilise la même méthode de sélection, mais seulement dans la sous-population (deme). Un opérateur de hill-climbing démarre chaque fois qu'un individu est généré avec une valeur de fitness plus élevée que celle la meilleure obtenue jusqu'à maintenant. Les deux méthodes utilisent différentes techniques de codage génétique et fonctions de fitness.

Dans le GEFREX, le codage génétique ne concerne que les prémisses des règles floues. Ce codage est un mélange de parties à virgule flottante et parties binaires, où les codes de partie à virgule flottante et les valeurs significatives des fonctions d'appartenance et la partie binaire sont consacrés à la sélection de caractéristiques. Le nombre d'entrées n dépend des domaines d'application, alors que le nombre de Nr règles est spécifié par l'utilisateur du GEFREX. Au maximum Nr × n antécédents sont requis. La partie binaire nécessite seulement des bits, qui sont utilisés lorsque l'option de sélection de caractéristiques est activée durant la phase d'apprentissage. Le GEFREX est également capable de détecter les caractéristiques importantes sur demande durant la phase d'apprentissage. Les conséquences optimales peuvent être obtenues en utilisant SVD. Différents opérateurs de croisement et de mutation sont utilisés pour les gènes binaires et les gènes codés réels.

Le réseau flou adaptatif du contrôle de l'apprentissage (FALCON) est un système neuro-flou à cinq couches. Le FALCON-GA est algorithme d'apprentissage hybride en trois phases pour l'apprentissage de structure/paramètre, le fuzzy ART pour le clustering des données d'entraînement supervisé, l'AG pour trouver des bonnes règles floues, et le rétro-propagation pour régler les fonctions d'appartenance entrée/sortie.

Les systèmes existants pour cette classe sont : FuGeNeSys, GEFREX, FALCON-GA, DENFIS, EFuNN, ECOS, …etc.

1.6. Présentation de quelques systèmes hybrides

L'objectif de cette section est de présenter quelques systèmes hybrides de soft computing. Pour chacun d'eux, nous mettons l'accent sur les techniques utilisées.

1.6.1. Le système EPNet [XIN 99]

Yao et Liu ont développé un système automatique EPNet basé sur la PE pour l'évolution simultanée des architectures RNA et poids de connexion. EPNet n'utilise pas d'opérateurs de croisement. Il compte sur plusieurs opérateurs de mutation pour modifier les architectures et les poids. L'évolution comportementale (c.-à-d., fonctionnelle), évolution plutôt génétique, est accentuée dans EPNet. Plusieurs techniques ont été adoptées pour maintenir le lien comportemental entre un parent et ses fils. Fig 1.13 montre la structure principale d'EPNet.

EPNet utilise la sélection basée-rang et cinq mutations: entraînement hybride, suppression de nœud, suppression de connexion, addition de connexion et addition de nœud. L'entraînement hybride est la seule mutation dans EPNet qui modifie les poids du RNA. Il est basé sur un algorithme de rétro-propagation modifié (MBP) avec un taux d'apprentissage adaptatif et recuit

simulé. Les quatre autres mutations sont utilisées pour augmenter et élaguer les nœuds cachés et les connexions.

Le nombre d'époques utilisé par MBP pour entraîner chaque RNA dans une population est défini par deux paramètres spécifiés par l'utilisateur. Il n'y a aucune garantie qu'un RNA convergera à un même optimum local après ces époques. D'où ce processus d'entraînement est appelé l'entraînement partiel. Il est utilisé à combler le retard comportemental entre un parent et son fils.

Les cinq mutations sont tentées séquentiellement. Si une mutation mène à un meilleur fils, elle est considérée comme réussie. Aucune mutation supplémentaire ne sera appliquée. Autrement la prochaine mutation est tentée. La motivation derrière ordonner les mutations est d'encourager l'évolution des RNAs compacts sans sacrifier la généralisation. Un ensemble de validation est utilisé dans EPNet pour mesurer la fitness d'un individu.

EPNet a été testé largement sur plusieurs problèmes d'évaluation et été accompli des résultats excellents, y compris problèmes de parité de taille de quatre à huit, le problème de deux-spirale, le problème de cancer de sein, le problème de diabète, le problème de maladie de cœur, le problème thyroïde, le problème de carte de crédit australien, le problème de prédiction de série du temps du Mackey-Glass, etc. RNAs très compacts avec bonne capacité de généralisation ont été évolués.

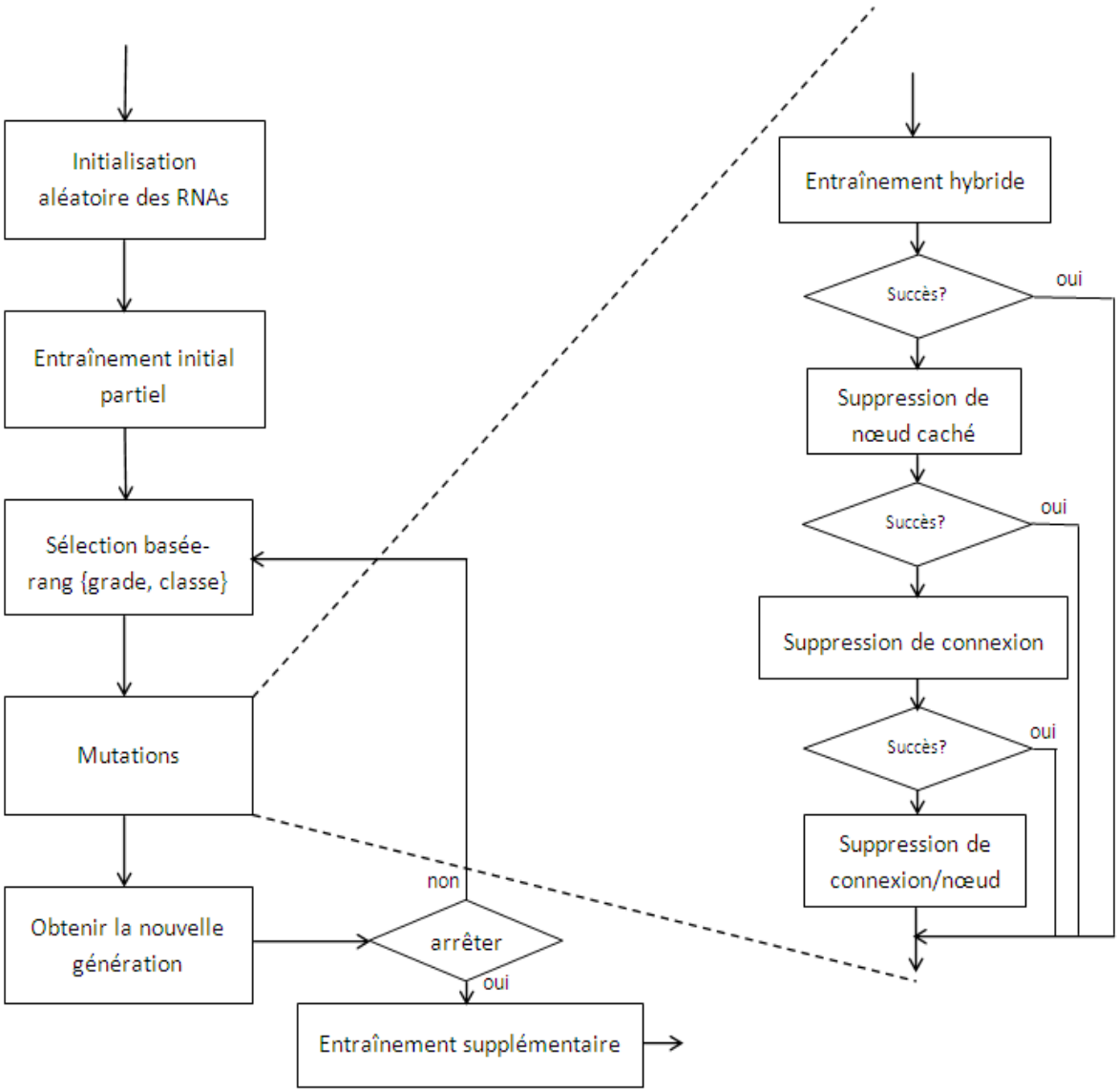


Figure 1.13 : La structure principale d'EPNet [XIN 99]

1.6.2. Le système NEFPROX [TET 01]

Dans NEFPROX (NEuro Fuzzy function apPROXimator), un système neuro-flous est vu comme un réseau feedforward de trois-couche.

Il n'y a pas de cycles dans le réseau et aucunes connexions n'existent entre la couche n et la couche n + j, avec j > 1. La première couche représente des variables d'entrée, la couche cachée représente les règles floues et la troisième couche représente les variables de sortie. Les unités cachées et de sortie dans ce réseau utilisent des t-normes et des t-conorms comme fonctions d'agrégation, d'une manière similaire à ce que nous avons vu dans les sections précédentes. Les ensembles flous sont codés comme des poids de connexion flous et des entrées floues. Le réseau entier est capable d'apprendre et fournit un chemin d'inférence flou. Le résultat final devrait être interprétable comme un système de règles linguistiques.

Le problème devant être résolu est celui d'approximer une fonction continue inconnue en utilisant un système flou étant donné un ensemble d'échantillons de données. Il y a une preuve d'existence que les systèmes flous sont capables d'approximation de fonction universelle. Cependant, construire réellement une telle approximation pour un problème donné exige la spécification des paramètres sous la forme de fonctions d'appartenance et d'une base de règle. Cette identification peut être faite par connaissance antérieure, essai et erreur, ou par certaine méthodologie d'apprentissage automatique. NEFPROX code les paramètres du problème dans le réseau et utilise un algorithme d'apprentissage supervisé dérivé à partir de la théorie de réseau de neurones afin de conduire le mappage vers des solutions satisfaisantes. L'avantage de l'approche floue sur un réseau de neurones standard est que, pendant que le dernier est une boîte noire, le système flou peut être interprété en termes de règles et donc a un pouvoir plus descriptif.

Le système NEFPROX est un réseau de trois-couche avec les caractéristiques suivantes:

- Les unités d'entrée sont étiquetées x_1, x_2, …, x_n. Les unités de règle cachées sont appelées R_1, R_2, …, R_k et les unités de sortie sont dénotées comme y_1, y_2, …, y_m.
- Chaque connexion est pondérée avec un ensemble flou et est étiquetée avec un terme linguistique.
- Toutes les connexions venant de la même unité d'entrée et ayant la même étiquette sont pondérées par le même poids commun, qui est appelé un poids partagé. La même chose vaut pour les connexions qui mènent à la même unité de sortie.
- Il n'y a aucune paire de règles avec des antécédents identiques.

D'après ces définitions, il est possible d'interpréter un système NEFPROX comme un système flou dans lequel chaque unité cachée signifie une règle floue si-alors. Les poids partagés sont exigés pour que chaque valeur linguistique avoir une interprétation unique. Si ce n'était pas le cas, il serait possible pour les poids flous représentant des termes linguistiques identiques d'évoluer différemment durant l'apprentissage, menant aux fonctions d'appartenance individuelles différentes pour ses variables d'antécédents et de conclusions qui veulent à leur tour empêchent l'interprétation appropriée de la base de règle floue. Fig 1.14 représente graphiquement la structure d'un système NEFPROX.

L'apprentissage dans NEFPROX est basé sur l'entraînement supervisé et emplois une variation conceptuelle de rétro-propagation standards dans les RNAs puisque nous sommes dans une structure où les ensembles d'entrée/sortie inconnus existent habituellement. La différence avec l'algorithme standard est que la méthode pour déterminer les erreurs et les rétro-propager en arrière et pour effectuer des modifications locales de poids n'est pas basée sur la descente de gradient.

Puisque les règles floues sont utilisées dans NEFPROX pour approximer la fonction inconnue, la connaissance pré-existante peut être utilisée au début en initialisant le système avec les règles déjà connues, s'il y en a. Les règles restantes doivent être trouvées dans le processus d'apprentissage. Si rien n'est connu sur le problème, le système commence sans unités cachées, qui représentent des règles, et les apprend de manière incrémentale.

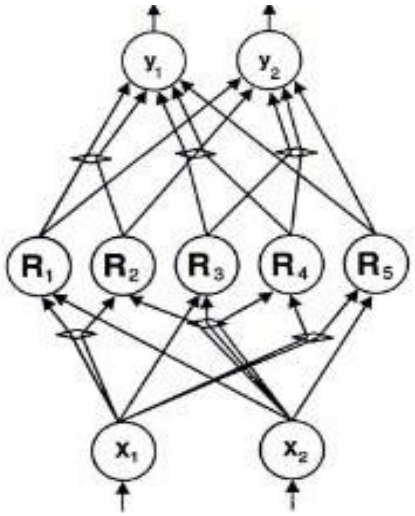


Figure 1.14 : Architecture schématique d'un système NEFPROX. Les connexions passant par les petits losanges sont liées c.-à-d., elles partagent le même poids

Les fonctions d'appartenance triangulaires simples sont utilisées pour les ensembles flous bien que d'autres formes soient aussi admissibles. Au début du processus d'apprentissage les partitions floues pour chaque variable d'entrée sont spécifiées. Les ensembles flous pour les variables de sortie sont créés durant l'apprentissage et une procédure de défuzzification est utilisée aux nœuds de sortie pour comparer les valeurs calculées et observées. Etant donné un ensemble de modèles d'entraînement $\{s_1, t_1, \ldots, s_r, t_r\}$ où $s \in \mathbb{R}^n$ est un modèle d'entrée et $t \in \mathbb{R}^m$ la sortie désirée, l'algorithme d'apprentissage a deux parties : une partie d'apprentissage-structure et une partie d'apprentissage-paramètre. Une description simplifiée de l'algorithme est la suivante :

Algorithme d'Apprentissage de Structure

1. Sélectionner le prochain modèle d'entraînement (s, t) à partir de l'ensemble d'entraînement.
2. Pour chaque unité d'entrée x_i trouver la fonction d'appartenance $\mu_{ji}^{(i)}$ tel qui

$$\mu_{ji}^{(i)}(s_i) = \max_{j \in \{1,\ldots,p_i\}}\{\mu_j^{(i)}(s_i)\}.$$

3. S'il n'y a aucune règle R avec les poids $W(x_1,R) = \mu_{ji}^{(1)}, \ldots, W(x_n,R) = \mu_{jn}^{(1)}$ alors créer le nœud et le connecter à tous les nœuds de sortie.
4. Pour chaque connexion à partir du nouveau nœud de règle aux nœuds de sortie trouver un poids flou approprié $v_{ji}^{(i)}$ en utilisant les fonctions d'appartenance assignées aux unités de sortie y_i tel que $v_{ji}^{(i)}(t_i) = \max_{j \in \{1,\ldots,q_i\}}\{v_j^{(i)}(t_i)\}$ et $v_j^{(i)}(t_y) \geq 0.5$. Si l'ensemble flou n'est pas défini alors créer un nouvel ensemble flou $v_{new}^{(i)}(t_i)$ pour la variable de sortie y_i et mettre $W(R, y_i) = v_{new}^{(i)}$.
5. S'il n y a plus de modèles d'entraînement alors arrêter la création de règle; autrement aller à 1.
6. Évaluer la base de règle et changer les conclusions de règle si approprié.

La partie d'apprentissage supervisé adapte les ensembles flous associés aux poids de connexion selon le schéma suivant :

Algorithme d'Apprentissage de Paramètre

1. Sélectionner le prochain modèle d'entraînement (s, t) à partir de l'ensemble d'entraînement et le présenter à la couche d'entrée.
2. Propager le modèle en avant (forward) à travers la couche cachée et laisser les unités de sortie déterminent le vecteur de sortie o.
3. Pour chaque unité de sortie y_i déterminer l'erreur $\delta_{yi} = t_i - o_{yi}$.
4. Pour chaque unité de règle R avec sortie $o_R > 0$ faire ;
    - Mettre à jour les paramètres des ensembles flous W (R, y_i) en utilisant un paramètre du taux d'apprentissage $\sigma > 0$.
    - Déterminer le changement $\delta_R = o_R(1 - o_R).\sum_{y \in \text{output layer}} (2W(R, y)(t_i) - 1).|\delta_y|$.
    - Mettre à jour les paramètres des ensembles flous W (x, R) en utilisant δ_R et σ pour calculer les variations.
5. Si une passe à travers l'ensemble d'entraînement a été complétée et le critère de convergence est rencontré alors arrêter; autrement aller à étape 1.

Deux approches neuro-floues apparentées sont NEFCON et NEFCLASS qui sont utilisés, respectivement, pour les applications de contrôle et pour les problèmes de classification. NEFCON est similaire à NEFPROX mais a seulement une variable de sortie et le réseau est entraîné par l'apprentissage par renforcement en utilisant une mesure d'erreur floue basée-règle comme un signal de renforcement. NEFCLASS voit la classification de forme comme un cas spécial de l'approximation de fonction et utilise l'apprentissage supervisé d'une manière similaire à NEFPROX pour apprendre des règles de classification.

1.6.3. Le système ANFIS

ANFIS signifie système d'inférence flou adaptatif basé-réseau (Adaptive Network-based Fuzzy Inference System) et est un système neuro-flou qui peut identifier les paramètres en utilisant des méthodes d'apprentissage supervisé. ANFIS peut être considéré comme une représentation en réseau de systèmes flous de type-Sugeno avec des capacités d'apprentissage. ANFIS est similaire dans l'esprit à NEFPROX mais, en ce qui concerne le dernier, l'apprentissage se passe dans un réseau de structure fixe et il exige des fonctions différentiables. L'architecture hétérogène du réseau ANFIS est constituée par plusieurs couches de nœuds qui ont la même fonction pour une couche donnée mais sont différent d'une couche à l'autre. Par exemple, considérer le système d'inférence flou avec deux entrées x et y et une seule sortie z. Pour un modèle Sugeno ordre-premier, un ensemble de règle utilisant une combinaison linéaire des entrées peut être exprimée comme :

$$\text{SI x est } A_1 \text{ ET y est } B_2 \text{ ALORS } f_1 = p_1x + q_1y + r_1$$

$$\text{SI x est } A_2 \text{ ET y est } B_2 \text{ ALORS } f_2 = p_2x + q_2y + r_2$$

Le mécanisme de raisonnement pour ce modèle est:

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \overline{w}_1 + \overline{w}_2$$

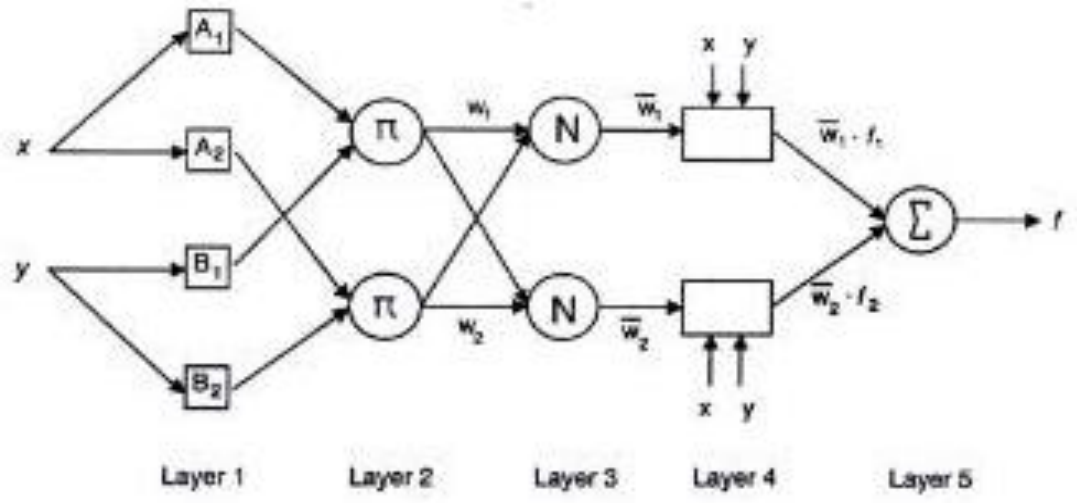


Figure 1.15 : Architecture ANFIS correspondant à un modèle Sugeno flou premier-ordre de deux-entrées avec deux règles

L'architecture du réseau ANFIS correspondant à ce modèle Sugeno est montrée dans Fig 1.15. Les couches dans le réseau sont constituées par nœuds ayant la même fonction pour une couche donnée. Les fonctionnalités des couches sont comme suit :

Couche 1: Dénoter par $O_{l,\,i}$ la sortie de nœud i dans la couche l, chaque nœud dans la couche l est une unité adaptative avec sortie donnée par:

$$O_{1,i} = \mu_{A_i}(x), \qquad i = 1, 2$$
$$O_{1,i} = \mu_{B_{i-2}}(x), \quad i = 3, 4$$

où x et y sont les valeurs d'entrée au nœud et A_i ou B_{i-2} sont les ensembles flous associés au nœud. En d'autres termes, chaque nœud dans cette couche produit les classes d'appartenance de la partie prémisse. Les fonctions d'appartenance pour A_i et B_i peuvent être toute fonction d'appartenance paramétrée appropriée tel que triangulaire, trapézoïdal, Gaussienne ou en forme de cloche.

Couche 2: Chaque nœud dans cette couche est étiqueté Π et calcule la force de tir de chaque règle comme le produit des nouvelles entrées ou tout autre opérateur t-norme :

$$O_{2,i} = w_i = \mu_{A_i}(x)\Delta\mu_{B_i}(y), \quad i=1, 2$$

Couche 3: Chaque nœud dans cette couche est étiqueté N et il calcule le ratio de la $i^{\text{-ème}}$ force de tir de règle à la somme de toutes les forces de tir de règle :

$$O_{3,i} = \bar{w}_i = \frac{w_i}{w_1+w_2}, \quad i=1, 2$$

Couche 4: Chaque nœud dans cette couche a la fonction suivante:

$$O_{4,i} = \bar{w}_i f_i = \bar{w}_i(p_i x + q_i y + r_i)$$

où $\bar{w}_i$ est la sortie de la couche 3 et $\{p_i, q_i, r_i\}$ est l'ensemble de paramètres.

Couche 5: Il y a un seul nœud ∑ dans cette couche. Il agrège la sortie totale comme l'addition de tous les nouveaux signaux :

$$O_{5,1} = \sum_i \bar{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

Cela complète la construction du réseau qui est vue comme ayant les mêmes fonctionnalités que le modèle Sugeno équivalent.

1.7. Conclusion

Dans ce chapitre, nous nous sommes intéressés au soft-computing en commençant par ses méthodes de base : les réseaux de neurones, la logique floue et les algorithmes évolutionnaires. A travers notre étude relative à différentes intégrations entre ces méthodes prises deux à deux puis toutes les trois, nous avons constaté la richesse des approches hybrides du soft-computing, ce qui nous a encouragés à envisager l'utilisation de l'une de ces méthodes dans le cadre de notre travail.

CHAPITRE 2

L'APPRENTISSAGE ADAPTATIF INCREMENTAL

L'adaptabilité représente l'une des limitations les plus fondamentales des techniques d'apprentissage qui sont actuellement relativement efficaces dans le cas statique. De ce fait, les méthodes classiques sont souvent inefficaces pour répondre aux nouveaux besoins des applications actuelles où des flux continus de gros volumes de données sont disponibles.

Les recherches dans le domaine de l'apprentissage incrémental constituent une des tendances actuelles de l'apprentissage automatique. L'apprentissage incrémental concerne la capacité d'apprendre à partir de données reçues au fur et à mesure du temps pour s'adapter de manière incrémentale aux changements de l'environnement.

Plusieurs techniques d'apprentissage incrémental ont été proposées dans différents contextes applicatifs et pour différentes approches de classification.

2.1. Introduction

L'apprentissage incrémental fait référence au processus d'accumulation et de gestion de connaissances dans le temps. Il est très approprié pour les tâches d'apprentissage dans lesquelles les ensembles de données d'entraînement deviennent disponibles pendant une longue période de temps [CHE 07, MUR 07].

En général, il y a trois types de changements dans les problèmes de classification que leur traitement nécessite des classifieurs pouvant apprendre de manière incrémentale et s'adapter progressivement au nouvel environnement : des nouvelles données d'apprentissage peuvent être disponibles ; des nouveaux attributs peuvent être trouvés et des nouvelles classes peuvent apparaître [GUA 05]. Ces types de changement impliquent trois types d'apprentissage incrémental : apprentissage incrémental d'exemple ; apprentissage incrémental d'attribut et apprentissage incrémental de classe [LIY 08, WEN 07].

Le terme "apprentissage incrémental" a été utilisé plutôt librement dans la littérature pour indiquer divers concepts comme "augmentation et élagage incrémentaux de réseau", "apprentissage en-ligne", ou "réapprentissage des exemples mal-classifiés précédemment". En outre, beaucoup d'approches présentant quelques aspects de l'apprentissage incrémental (par exemple : l'apprentissage de nouvelle information) sont utilisées sous plusieurs noms comme "apprentissage constructif", "apprentissage life-long", "dérive de concept" et "apprentissage évolutionnaire" [POL 01].

Comment apprendre des nouvelles données sans dégrader les connaissances déjà acquises est un problème important de l'apprentissage incrémental qui s'appelle le dilemme "stabilité-plasticité" (c'est un problème spécifique à l'apprentissage adaptatif).

2.2. Définitions de l'apprentissage incrémental

Plusieurs définitions existent dans la littérature. Une des premières définitions formelles est la description de Gold de l'apprentissage dans la limite qui suppose que l'apprentissage continue indéfiniment, et l'algorithme a accès à toutes les données générées précédemment [MUH 09]. D'autres définitions ont été aussi proposées.

Définition 1 [GUA 05]

L'apprentissage incrémental est une technique d'apprentissage *ad hoc* par laquelle l'apprentissage se produit plutôt avec l'arrivée de nouvelles données qu'à partir d'un ensemble fixe de données, c.-à-d., c'est plutôt un processus continu qu'une expérience one-shot.

Définition 2 [MAU 05]

Un apprenant L est incrémental si L entre une expérience d'apprentissage à la fois, ne retraite aucune expérience précédente et retient seulement une structure de la connaissance dans la mémoire.

Définition 3. [MUH 09]

Un algorithme d'apprentissage est incrémental si, pour une séquence d'ensembles de données d'entraînement (ou instances), il produit une séquence d'hypothèses, où l'hypothèse courante décrit toutes les données qui ont été vues précédemment, mais dépend seulement des hypothèses antérieures et les données d'entraînement courantes. D'où, un algorithme d'apprentissage incrémental doit apprendre la nouvelle information, et retient la connaissance acquise précédemment, sans avoir accès aux données vues précédemment.

Définition 4 [WEN 07]

L'apprentissage incrémental est de construire une nouvelle hypothèse en utilisant seulement l'hypothèse d'avant et l'information récente.

Définition 5 [ORS 07]

Le caractère incrémental s'occupe de la manière dont la connaissance est modifiée lorsqu'on apprend un problème B après un problème A. En plus d'acquérir de la connaissance sur une tâche donnée, il faut notamment que le système soit capable :

- d'en accumuler de plus en plus, sans borne *a priori,* ou une borne très grande, de sorte que l'agent considère qu'il n'y en a virtuellement pas. Ceci suppose que l'apprenant ne doit pas avoir de borne a priori sur la taille mémoire nécessaire (ou au pire une borne très grande), et qu'il doit aussi ne pas oublier rapidement ce qu'il a appris, surtout lorsqu'il apprend d'autres tâches,
- de la généraliser, pour l'utiliser dans le plus de cas possibles,
- de la spécialiser ou la discréditer, dans le cas où elle se révélerait erronée,
- de la réutiliser dans d'autres tâches.

D'autres définition sont citées dans [ORS 07], y compris :

- Le système doit pouvoir accumuler de la connaissance et l'adapter au fil des exemples [FRI 94][2].
- Il s'agit de fournir les exemples d'un même problème les uns après les autres, sans contrainte de temps [MIT 97, RUS 03] [3, 4].
- On suppose que le concept à apprendre est fixe, mais que le jeu de données est fourni en séquence. On impose au système d'apprendre (fournir des hypothèses ou être prêt à prendre des décisions) en continu, sans avoir reçu l'ensemble des données [COR 05][5].

2.3. Le dilemme "stabilité-plasticité"

Le but de l'apprentissage incrémental est d'obtenir une quantité suffisante de connaissances en les accumulant à partir d'un grand nombre d'ensembles de données venues en différents temps. Pour accomplir ce but, les algorithmes d'apprentissage incrémental devraient avoir deux capacités d'apprentissage : stabilité et plasticité. La stabilité signifie la capacité de l'apprentissage à maintenir les connaissances obtenues précédemment, et la plasticité signifie la capacité de l'apprentissage à extraire et accumuler des nouvelles connaissances [PAR 09].

Comment acquérir des nouvelles connaissances à partir de nouvelles données d'apprentissage en retenant les connaissances apprises précédemment est un problème important pour l'apprentissage incrémental qui s'appelle le dilemme "stabilité-plasticité". Un classifieur strictement stable va retenir les connaissances existantes sans pouvoir apprendre de nouvelles informations, tandis qu'un classifieur strictement plastique va pouvoir apprendre de nouvelles informations mais au détriment de celles précédemment apprises. Tandis qu'une perte graduelle peut être inévitable, une perte soudaine et complète des connaissances acquises précédemment devrait être évitée [MUH 09]. La relation dilemmatique entre la stabilité et la plasticité nécessite de proposer le meilleur compromis "stabilité-plasticité" c.-à-d. d'apprendre des nouvelles informations en retenant autant que possible les connaissances précédentes.

2.4. Principe général d'un algorithme d'apprentissage incrémental

Un algorithme d'apprentissage incrémental offre à un système la capacité d'apprendre à partir de nouvelles informations au fur et à mesure qu'elles deviennent disponibles [SEI 05]. Il peut être exécuté en cinq étapes [MIS 06] :

1. Apprendre des règles à partir de l'ensemble d'exemples d'apprentissage ;
2. Stocker les nouvelles règles et abandonner les exemples les plus anciens ;
3. Utiliser les règles apprises pour prédire et naviguer ;
4. Quand des nouveaux exemples arrivent, apprendre des nouvelles règles en utilisant les anciennes règles et les nouvelles instances ;
5. Aller à l'étape 2.

---

[2] B. Fritzke, *Fast learning with incremental RBF networks,* Neural Processing Letters, 1994.
[3] T. M. Mitchell, *Machine Learning,* McGraw-Hill, 1997.
[4] S. Russell et P. Norvig, *Artificial Intelligence : A Modern Approach (Second Edition),* Prentice-Hall, 2003.
[5] A. Cornuéjols, *Apprentissage et circulation d'information,* Habilitation à diriger les recherches, Université de Paris, 2005.

Un algorithme d'apprentissage incrémental est défini dans [POL 01] comme répondant aux critères suivants :

- Il doit être capable d'apprendre des informations supplémentaires à partir de nouvelles données (plasticité).
- Il doit ne pas nécessiter l'accès aux données d'origine utilisées pour apprendre le classifieur existant).
- Il doit préserver les connaissances acquises précédemment (il ne devrait pas souffrir d'oublie catastrophique : stabilité).
- Il doit pouvoir s'adapter aux nouvelles classes qui peuvent être introduites aves les nouvelles données.

Ces quatre points s'appliquent pour tout problème général de l'apprentissage incrémental. La propriété 3 est particulièrement pertinente pour fournir une haute qualité de résultats [GRA 08]. De même, d'autres caractéristiques peuvent compléter ces 4 points. Il s'agit en particulier de la capacité d'oublier ou supprimer des connaissances devenues obsolètes, ou de restructurer de manière autonome les connaissances acquises par des opérations de regroupement ou d'éclatement [PRU 06] ; d'apprendre les données en une seule passe [KAS 01a, YAL 07] ou de limiter la mémoire et le temps de traitement [YAL 07].

Les algorithmes d'apprentissage incrémental mettent des nouveaux défis aux chercheurs: (1) comment mémoriser la connaissance apprise pour d'autres mises à jour sans enregistrer chaque cas déjà appris; (2) comment éviter (ou garder) les effets de l'ordre dans lequel les cas ont été appris; (3) comment concevoir un algorithme à mise à jour rapide; (4) comment rendre les résultats de l'apprentissage interprétables aux être humain [WU 07].

Lors de la conception d'un système d'apprentissage incrémental, les principes suivants devraient être considérés [WU 07] :

- Le coût de mise à jour de la méthode doit être petit.
- La méthode devrait accepter des cas comme entrée décrite par tout mélange de variables symboliques et numériques, parfois variables continues.
- La méthode devrait être capable de gérer des classes multiples aussi bien que deux classes.
- La méthode devrait être assez forte pour gérer des données bruitées et des données contradictoires.
- La méthode devrait prendre la possibilité que les données entre les catégories soient déséquilibrées en considération.
- Capable de gérer certains problèmes avec relations fortes parmi plusieurs attributs.

2.5. Caractéristiques d'un système d'apprentissage incrémental

Un système d'apprentissage incrémental met à jour ses hypothèses, quand c'est nécessaire, suite aux données d'apprentissage récemment disponibles sans réexaminer les anciennes données [CHE 07, MUR 07]. Les trois suppositions les plus importantes caractérisant un système d'apprentissage incrémental sont [MAU 05] :

a) il doit être capable d'utiliser les connaissances acquises à n'importe quelle étape de l'apprentissage ;

b) l'incorporation d'expériences dans la mémoire au cours l'apprentissage doit être efficace computationnellement (la révision de théorie doit être efficace dans l'ajustement de nouvelles observations arrivées) et ;
c) le processus d'apprentissage ne devrait pas faire des demandes déraisonnables d'espace, de sorte que les besoins en mémoire augmentent en fonction de l'expérience traitable (les besoins en mémoire doivent ne pas dépendre de la taille d'entraînement).

2.6. Types d'apprentissage incrémental

Un système d'apprentissage incrémental est paramétré par trois familles de paramètres qui peuvent être modifiés pendant l'apprentissage [VIL 07, VIL 09, CHA 02] :

- Les paramètres de structure, par exemple, le nombre de neurones ou de connexions d'un réseau de neurones.
- Les paramètres d'apprentissage, par exemple, le pas.
- Les paramètres de complexité des données qui peuvent représenter toute mesure de complexité des données d'entraînement.

L'analyse de l'apprentissage biologique a mené à comprendre que l'apprentissage incrémental peut être de trois types, chacun d'eux modifie une des familles de paramètres définis ci-dessus :

- Apprentissage incrémental de structure : La structure du système est modifiée pendant l'apprentissage. Exemple : les méthodes d'augmentation et d'élagage de réseau.
- Apprentissage incrémental de paramètres d'apprentissage : les paramètres d'apprentissage sont adaptés pendant l'apprentissage.
- Apprentissage incrémental de données : l'ensemble de données ou sa complexité sont augmentés en étapes pendant l'apprentissage.

Dans de nombreux cas un mélange de ces trois types d'apprentissage incrémental peut avoir lieu. L'apprentissage incrémental dans fuzzy ARTMAP est une combinaison de l'apprentissage incrémental de structure et de données. Comme des données de classes différentes sont ajoutées au système ou plus de données sur les classes existantes sont ajoutées, la structure du système s'adapte en créant un plus grand nombre de catégories devant permettre de correspondre les étiquettes de sortie. L'apprentissage incrémental dans Learn++ est un apprentissage incrémental de données parce que comme des nouvelles données sont ajoutées, de nouveaux classifieurs sont construits pour être ajoutés au système existant.

2.7. Avantages de l'apprentissage incrémental

1. L'apprentissage incrémental fournit les moyens pour maintenir efficacement des modèles de classe exacts et à jour lorsqu'une nouvelle donnée d'apprentissage devient disponible [GRA 08].
2. Une telle stratégie d'apprentissage est à la fois économique spatialement et temporellement, parce qu'elle enlève le besoin de stocker et de retraiter les anciennes instances [CHE 07, MUR 07].
3. La complexité computationnelle basse requise pour mettre à jour un classifieur. En effet, le stockage temporaire des nouvelles données est requis seulement durant

l'apprentissage et l'apprentissage est effectué seulement avec les nouvelles données [GRA 08].

4. L'apprentissage incrémental peut fournir un outil puissant dans les applications centré-humain où un expert du domaine est appelé à proposer le nouvel ensemble de données pour concevoir et mettre à jour progressivement un système de classification quand un environnement complexe apparaît [GRA 08].

2.8. Méthodes/Algorithmes d'apprentissage incrémental

Les classifieurs conventionnels ne sont pas appropriés à l'apprentissage incrémental ; soit parce qu'ils sont stables tel que le Perceptron Multi-Couches PMC, les réseaux à fonction radiale de base (RBF) ou les machines à vecteur de support SVM ; soit parce qu'ils ont une haute plasticité et ne peuvent pas retenir les connaissances acquises précédemment sans avoir accès aux anciennes données tel que la méthode des K-Plus Proches Voisins. Une complication supplémentaire survient si les données supplémentaires introduisent des nouvelles classes du concept ce qui représente un changement particulièrement hostile et brusque dans la distribution principale de données.

Une approche pragmatique souvent utilisée pour apprendre à partir de nouvelles données est d'ignorer le classifieur existant et réapprendre un nouveau en utilisant toutes les données accumulées précédemment. Cette approche résulte en un *oubli catastrophique,* qui signifie une perte de toutes les connaissances apprises précédemment. En plus de violer la définition formelle de l'apprentissage incrémental, cette approche est aussi indésirable si le réapprentissage est couteux computationellement ou financièrement et ne peut même pas être faisable si les données originales ne sont pas disponibles.

La recherche sur les algorithmes constructifs et destructifs représente un effort vers la conception automatique des architectures. En gros, un algorithme constructif commence avec un réseau minimal (réseau avec un nombre minimal de couches cachées, nœuds et connexions) et ajoute des nouvelles couches, nouveaux nœuds et nouvelles connexions quand c'est nécessaire pendant l'entraînement bien qu'un algorithme destructif fait le contraire, c.-à-d., commence avec le réseau maximal et supprime les couches, nœuds et connexions inutiles pendant l'entraînement.

2.8.1. Les architectures ART

Les architectures ART issues des travaux de S. Grossberg et G. Carpenter sont basées sur une théorie inspirée de la biologie nommée "Théorie de la Résonance Adaptative". Cette théorie est le résultat d'une tentative pour comprendre à quel point les systèmes biologiques sont capables de maintenir la plasticité durant toute leur vie, sans compromettre la stabilité des modèles précédemment construits. Les modèles construits en utilisant cette théorie protègent les connaissances acquises en rendant l'apprentissage conditionnel. Pour ce faire, ils distinguent la mémoire à long terme, contenue dans les connexions et modifiée par apprentissage, de la mémoire à court terme, mémoire instable contenue dans les neurones. L'élément clé pour gérer le passage d'un mode "stable" à un mode "plastique" dans les modèles ART est le contrôle de la correspondance partielle entre les vecteurs déjà appris et les nouveaux vecteurs de caractéristique en utilisant un paramètre nommé *facteur de vigilance* qui contrôle le degré de désadaptation que le système peut tolérer entre les nouveaux modèles et les modèles appris [KAS 07].

Le principe de la théorie de la résonance adaptative est comme suit :

Le réseau possède un réservoir de neurones de sortie qui ne sont utilisés que si nécessaire. Un neurone sera soit recruté soit libre. On appelle vecteur prototype le vecteur poids d'un neurone recruté, et patron la représentation x de la donnée à apprendre. Un patron et un prototype "résonnent" lorsqu'ils sont suffisamment proches. La notion de similarité est contrôlée via un paramètre de vigilance ρ qui détermine une similarité seuil au dessous de laquelle un patron et un prototype ne résonnent pas. Lorsqu'un patron n'est pas assez proche des vecteurs prototypes existants, une nouvelle catégorie est créée et un neurone libre y est assigné avec comme vecteur prototype le patron correspondant. S'il n'existe plus de neurone libre dans le réservoir, le patron ne provoque pas de réponse du réseau et la donnée n'est pas apprise.

Le paramètre de vigilance ρ est défini par l'utilisateur du système à l'aide d'une valeur comprise entre 0 et 1 déterminant le nombre de catégories reconnues. La valeur choisie influe grandement sur la qualité ainsi que la taille des classes à obtenir. Plus la valeur est proche de 1, plus les classes sont nombreuses et plus la classification est stricte. Par contre, plus la valeur est proche de zéro, plus les classes obtenues seront volumineuses mais peu nombreuses. En pratique, la valeur de ce paramètre est définie par l'utilisateur par essai et erreur.

Les méthodes ART fonctionnent comme suit :

- Au début l'ensemble des vecteurs prototypes est déterminé.
- Si un nouveau patron est introduit afin d'être classé, le système le compare à l'ensemble des vecteurs prototypes existants.
- Le système identifie le vecteur prototype le plus proche du patron d'entrée et calcule la distance entre le vecteur prototype sélectionné et le patron d'entrée grâce à une fonction G différente selon le type de réseau ART utilisé.
- Si le patron d'entrée est suffisamment similaire au vecteur prototype (selon le seuil de vigilance ρ), le patron d'entrée est inséré dans la classe décrite par le vecteur prototype sélectionné et le vecteur prototype est ajusté.
- Un prototype existant est modifié seulement si le patron courant est suffisamment similaire à l'individu moyen de ce prototype, sinon, ce patron est ajouté comme vecteur prototype aux prototypes existants.

Ce mécanisme est repris à partir de la troisième étape avec chaque nouveau patron.

Une description complète des équations régissant le modèle ART peut être trouvée dans [CAR 87, GRO 88, GRO 91].

2.8.1.1. Le modèle ART1

Le modèle ART1 [CAR 87] est un algorithme d'apprentissage non supervisé qui possède la particularité de prendre en entrée un ensemble de vecteurs binaires et créer à partir desquels des regroupements sur la base de certains critères de similarité.

Fig 2.1. montre un diagramme d'une architecture ART1 simple qui consiste en deux couches de neurones : une couche d'entrée qui comprend les neurones représentant les caractéristiques d'entrée, et une couche de sortie qui comprend les neurones représentant le concept de sortie. Les connexions b_{ij} de chaque entrée i à chaque sortie j et les connexions t_{ji} des sorties en arrière aux entrées sont montrées dans la figure. Chacun des neurones de sortie a une connexion excitatrice forte à lui-même et une connexion inhibitrice forte à chacun des autres neurones de sortie. Les poids des connexions t_{ji} représentent les modèles prototypes appris et les poids des connexions b_{ij} représentent un plan pour les nouvelles entrées devant être accommodées dans le réseau.

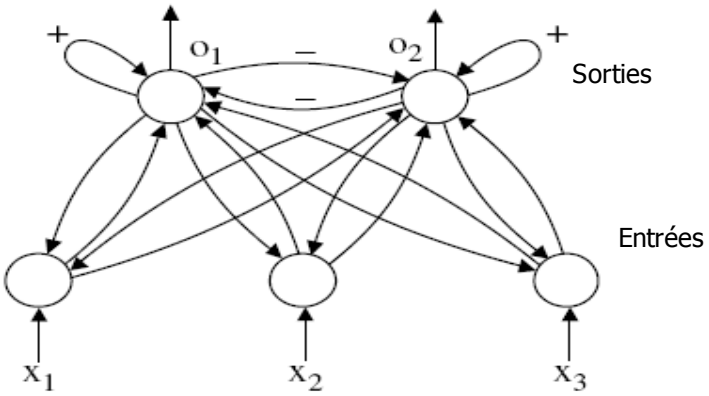


Figure 2.1 : Un diagramme schématique d'ART1 [KAS 07]

Les modèles, associés à un neurone de sortie j, sont représentés collectivement par le vecteur poids de ce neurone t_j (vecteur prototype). La réaction du neurone j à un nouveau vecteur d'entrée particulier est définie par un autre vecteur poids b_j.

L'algorithme d'apprentissage ART1 pour les entrées et les sorties binaires est donnée au-dessous. Il consiste en deux phases majeures. La première présente le modèle d'entrée et calcule les valeurs d'activation des neurones de sortie. Le neurone gagnant est défini. La deuxième phase est pour calculer la désadaptation entre le modèle d'entrée et le modèle associé actuellement au neurone gagnant. Si la désadaptation est au-dessous d'un seuil (paramètre de vigilance) ce modèle est mis à jour pour accommoder le nouveau neurone. Mais si la désadaptation est au-dessus du seuil, la procédure continue soit à trouver un autre neurone de sortie, soit à créer un nouveau neurone.

Algorithme ART1

1. Les coefficients des poids sont initialisés:

   $t_{ij}(0):1$, $b_{ij}:=1/(1+n)$, pour chaque i=1,2,...,n; j=1,2,...m

2. Un coefficient de similarité r, prétendu un facteur de vigilance, est défini, 0<=r<=1. Plus la valeur de r est grande, les modèles les plus similaires devraient être dans l'ordre d'activer le même neurone de sortie représentant une catégorie, une classe, ou un concept.
3. TANT QUE (il y a des vecteurs d'entrés) FAIRE
   a) un nouveau vecteur d'entrée x(t) est acheminé au moment t, $x = (x_1,x_2,...,x_n)(t)$
   b) les sorties sont calculées:
      $O_j=b_{ij}(t).x_i(t)$, for j=1,2..., m
   c) une sortie O_{j^*} avec la plus haute valeur est définie;
   d) la similarité du modèle d'entré associé à j^* est définie :
      SI (nombre de "1" dans l'interaction du vecteur x(t) et $t_{j^*}(t)$) divisé par le nombre de "1" dans x(t) est plus grand que la vigilance r) ALORS ALLER À (f)

      SINON
   e) la sortie j^* est abandonnée et la procédure revient à (b) afin de calculer une autre sortie devant être associée à x(t);
   f) le modèle x(t) est associé au vecteur $t_{j^*}(t)$, par conséquent le modèle $t_{j^*}(t)$ est changé en utilisant son interaction avec x(t):

$t_{ij}^*(t+1) := t_{ij}^*(t).x_i(t)$, pour i=1,2,...,n

g) les poids b_{ij} sont changés :

$b_{ij}^*(t+1) := b_{ij}^*(t) + t_{ij}^*(t).x_i/(0.5 + t_{ij}^*(t).\ x_i(t))$

2.8.1.2. Le modèle ARTMAP

ARTMAP est une méthode d'apprentissage supervisé qui combine deux réseaux ART, respectivement nommés ARTa et ARTb, reliés entre eux par l'intermédiaire d'un *map field* tel qu'illustré à Fig 2.2. pour effectuer un apprentissage supervisé. Ces cartes permettent de prédire les associations pouvant exister entre les catégories construites par les deux modules ART [CAR 91a, CAR 92]. Dans ce cas, on peut apprendre la topologie existant entre les formes présentées. Cependant, cette topologie ne peut être apprise qu'a posteriori (une fois que les catégories ont été crées dans les deux modules ART). Les informations de topologie ne peuvent donc pas être utilisées comme moyen de généralisation a priori.

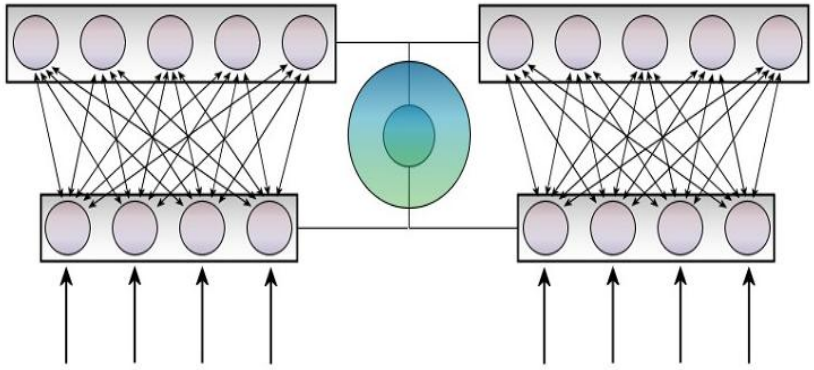


Figure 2.2 : Structure d'un réseau ARTMAP

Différents modèles ART

L'architecture ART1 a été développée dans plusieurs versions. ART2 accepte des entrées continues en incorporant aux mécanismes de ART1 différentes opérations complexes de normalisation. SimplifiedART est également une implémentation en valeurs continues de ART1. Quant à ART3, il développe ART2 davantage en lui ajoutant un nouveau mécanisme de réinitialisation bio-inspiré. Une cinquième architecture nommée fuzzyART possède la relative simplicité de ART1 tout en offrant la capacité de ART2 à traiter des entrées continues. Ces cinq architectures utilisent toutes un processus d'apprentissage non supervisé.

Les versions supervisées de l'architecture ART sont basées sur le modèle ARTMAP. Le fuzzy ARTMAP [CAR 92] est une extension de ARTMAP lorsque les neurones d'entrée représentent des degrés d'appartenance dans l'intervalle [0,1]. Il utilise des opérateurs flous MIN et MAX pour calculer l'intersection et l'union entre le modèle d'entrée floue x et les vecteurs poids à valeurs continues t. Le modèle GaussianARTmap [WIL 96] est plus performant et plus résistant au bruit que ARTMAP, grâce à l'utilisation des Gaussiennes comme fonction de choix et fonction de correspondance des modules ART.

Nous allons présenter dans le tableau suivant les modèles issus de la théorie de résonance adaptative en commençant par les versions non supervisés. Ces modèles diffèrent par le type d'apprentissage utilisé (supervisé, non supervisé) et le type de patrons d'entrées (binaires, analogiques).

Nom du modèle ART	Auteur, année et référence	Type d'apprentissage	Types de patrons d'entrées
ART1	Carpenter et Grossberg; 1987; [CAR 87]	Non supervisé	Patrons d'entrées binaires
SimplifiedART	Baraldi et Alpaydin ; 1998, [BAR 98]	Non supervisé	Patrons d'entrées analogiques (continues)
ART2	Carpenter et Grossberg; 1987; [CAR 87]	Non supervisé	Patrons d'entrées analogiques ou binaires
ART3	Carpenter et Grossberg ; 1990 ; [CAR 90]	Non supervisé	Patrons d'entrées analogiques ou binaires
FuzzyART	Carpenter, Grossberg et Rosen; 1991; [CAR 91b]	Non supervisé	Incorporation des notions de la logique floue
ARTmap	Carpenter et Grossberg; 1991; [CAR 91a]	Supervisé	Patrons d'entrées binaires ou analogiques selon le type de réseaux ART qui le constitue ART1 ou ART2 respectivement
FuzzyARTmap	Carpenter, Grossberg, Markuzon, Reynolds et Rosen; 1992; [CAR 92]	Supervisé	Comme ARTMAP mais les ARTs sont remplacés par des FuzzyARTs
GaussianARTmap	Williamson ; 1996 ; [WIL 96]	Supervisé	Comme ARTMAP, mais la fonction de choix et la fonction de correspondance des modules ART sont définies comme des Gaussiennes.

Tableau 2.1 : Particularités des modèles du réseau de neurones ART

L'inconvénient des méthodes ART est leur sensibilité à la sélection du seuil de similarité, au niveau de bruit dans les données d'apprentissage, et à l'ordre dans lequel les données sont présentées.

2.8.2. PBIL : Population-Based Incremental Learning

Baluja a développé l'algorithme PBIL basé sur l'exploration d'un espace de recherche en utilisant un vecteur de probabilité [BAL 94]. PBIL est à l'origine inspiré de l'apprentissage compétitif et est conçu pour des problèmes binaires. La version continue utilise une distribution gaussienne à base d'un produit de densités univariantes et indépendantes, ainsi que plusieurs adaptations pour l'évolution du vecteur de variance.

PBIL est un algorithme qui intègre de façon efficace les caractéristiques des algorithmes génétiques avec celles de l'apprentissage compétitif. Cette combinaison particulière résulte en un outil puissant qui est considérablement plus simple que l'AG, et dépasse les capacités d'un AG en termes de rapidité et de précision.

PBIL construit à chaque itération un ensemble de m solution x, à l'aide d'un vecteur de probabilité p(x) qui représente un prototype de la population. La mise à jour de p(x) est guidée par un coefficient d'apprentissage α qui contrôle l'amplitude des changements. La règle de mise à jour est inspirée de l'apprentissage compétitif pour l'apprentissage des cartes caractéristiques de Kohonen.

L'algorithme PBIL peut être décrit comme suit [KAR 04] :

Initialiser le vecteur de probabilité, P
REPETER
Générer des échantillons :
    Générer un vecteur d'échantillon en fonction de probabilités dans P
    Évaluer le vecteur d'échantillon
Trouver le vecteur correspondant à l'évaluation maximale
    (max ≡ meilleur échantillon)
Mettre à jour le vecteur de probabilité selon la règle suivante :
    $P_i = P_i*(1 - LR) + \max_i*(LR)$
    où LR représente le taux d'apprentissage
Muter le vecteur de probabilité
    si (random (0, 1] < probabilité de mutation)
    $P_i = P_i*(1 - Mut\_Shift) + rand(0, 1]*(Mut\_Shift)$
où Mut_Shift est la quantité de mutation affectant le vecteur de probabilité

2.8.3. ILFN : Incremental Learning Fuzzy Neural Network

ILFN est un classifieur implémentant une théorie hybride de réseau de neurone et ensembles flous [YEN 99].

2.8.3.1. L'architecture ILFN

L'architecture du réseau du classifieur ILFN est distinguée par deux mode différents: un mode d'apprentissage (montré dans Fig 2.3.) et un mode exploitation (montré dans Fig 2.4.). Les deux modes ont des différences seulement dans le module contrôleur et le module d'étiquetage cible. Le mode d'apprentissage utilise le schème d'apprentissage supervisé exigeant les paires d'entrée et cible des formes pour construire les prototypes du système. En revanche, le mode d'exploitation utilise l'algorithme d'apprentissage non supervisé pour déterminer la classe cible pour une forme d'entrée donnée. Lorsque le système détecte des nouvelles catégories, il utilise le module d'étiquetage cible pour assigner les cibles correspondants aux futures formes d'entrée. Les cibles qui sont assignées aux nouveaux prototypes sont significativement différentes des cibles existantes dans le module cible.

Le système ILFN a quatre couches: une couche d'entrée, une couche cachée, une couche de sortie et une couche de décision, comme montré dans Fig 2.3. et Fig 2.4.. Généralement, le système est composé de deux sous-systèmes : un sous-système d'entrée et un sous-système de cible. Chaque sous-système a trois couches : une couche d'entrée, une couche cachée et une couche de sortie. La couche cachée des deux sous-systèmes d'entrée et de cible sont liées ensemble par un module contrôleur qui est utilisé pour contrôler les neurones croissants dans la couche cachée. Chaque couche de sortie des deux sous-systèmes consiste en deux modules. La couche de sortie du sous-système d'entrée consiste en un module d'élagage et un module d'appartenance, pendant que la couche de sortie du sous-système de cible consiste en un module

d'élagage et un module de cible. Le module d'appartenance du sous-système d'entrée et le module cible du sous-système de cible sont mis à jour en simultanément avec leur nombre de neurones contrôlé par les modules d'élagage. La sortie du classifieur est liée ensemble par une couche de décision.

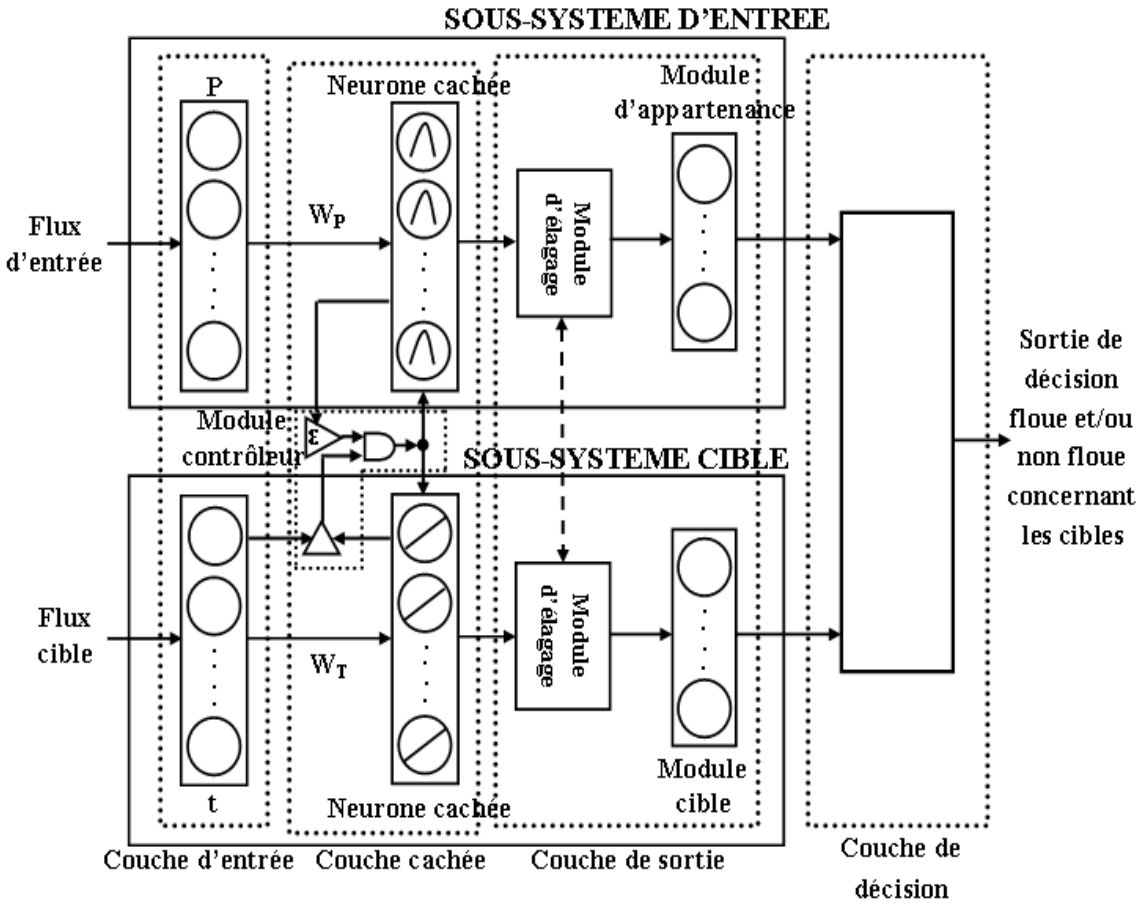


Figure 2.3 : Architecture du réseau du classifieur ILFN dans le mode d'apprentissage [YEN 99]

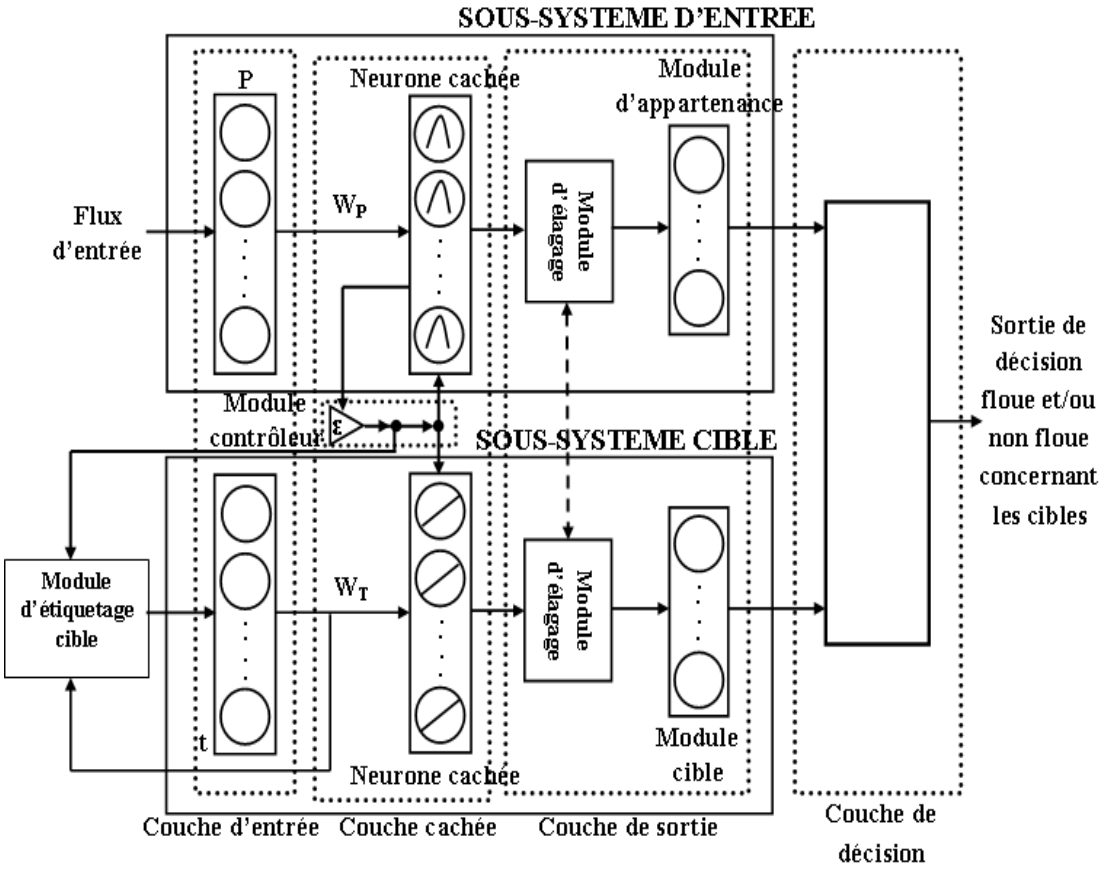


Figure 2.4 : Architecture de réseau du classifieur ILFN dans le mode d'exploitation [YEN 99]

2.8.3.2. L'algorithme ILFN

Étape 1: Mettre le paramètre seuil défini par l'utilisateur (ε), la déviation standard initiale σ_0 et le nombre maximal de modèles permis à inclure dans chaque cluster qui est utilisé dans le mode d'exploitation.

Étape 2: Lire dans le premier modèle d'entrée

- Utiliser le premier modèle d'entrée pour installer le premier prototype (ou moyenne) à W_p.
- Mettre le nombre de modèles pour le premier nœud à être 1.
- Mettre la déviation standard égale à la déviation standard initiale σ_0.
- Mettre un nouveau neurone à W_T en utilisant la première cible t pour être la cible correspondante du prototype dans W_p.

Étape 3: Lire dans le prochain exemple d'apprentissage avec une entrée et un modèle cible.

Étape 4: Mesurer la distance Euclidienne entre l'entrée p et le prototype W_p.

Étape 5: Calculer les valeurs d'appartenance pour chaque nœud en utilisant la fonction de base radiale de type Gaussien.

Étape 6: Assigner des valeurs d'appartenance à chaque nœud. Le modèle d'entrée courant a des degrés différents pour chaque nœud ou sous-classe. Pour chaque classe, sélectionner la valeur d'appartenance maximale à partir de chaque sous-classe pour représenter le degré de similarité en ce qui concerne cette classe.

Étape 7: Identifier la plus grande appartenance en utilisant l'opérateur floue OU.

Étape 8: Pour le nœud gagnant,

S'il y a cible correspondante (c.-à-d., il est dans le mode d'apprentissage),

1. si le gagnant est plus grand que ε et la cible t est la même valeur que W_T au nœud gagnant alors la mise à jour de poids W_p, la déviation standard et le nombre de modèles appartenant à ce nœud.
2. si 1) n'est pas satisfait, alors:
   - Mettre un nouveau centre du nœud pour W_p en utilisant le modèle d'entrée p.
   - Mettre le nombre de modèles pour le nouveau nœud à être 1.
   - Mettre la déviation standard initiale au nouveau nœud.
   - Ajouter un nouveau neurone à W_T, en utilisant la nouvelle cible t comme la cible correspondante d'un nouveau prototype dans W_p.

S'il n'y a aucune cible correspondante (c.-à-d., il est dans le mode d'exploitation),

1. si le gagnant est plus grand que ε et le nombre de modèles est moins que le nombre maximal de modèles permis, alors mettre à jour le poids W_p, la déviation standard et le nombre de modèles qui appartenant à ce nœud. Identifier la classe de sortie qui est stockée dans W_T au même index du nœud gagnant de W_P.
2. si le gagnant est plus petit que ε alors

Mettre un nouveau centre du nœud W_P en utilisant le modèle d'entrée p.

- Mettre le nombre de modèles pour le nouveau nœud à être 1.
- Mettre la déviation standard initiale au nouveau nœud.
- Ajouter un nouveau neurone à W_T et affecter une nouvelle cible comme la cible correspondante d'un nouveau prototype dans W_P.
-

Étape 9: S'il n'y a plus de modèles d'entrée, alors arrêter.
Autrement, aller à étape 3.

ILFN fournit les capacités d'apprentissage incrémental en une-passe, en ligne, temps réel. Le classifieur ILFN utilise une fonction Gaussienne à base radiale pour structurer les bornes de décision douces et dures. ILFN emploie un schème d'apprentissage hybride supervisé et non supervisé pour générer ses prototypes. Le réseau est un classifieur auto-organisé avec la capacité d'apprendre de manière adaptative des nouveaux prototypes sans oublier les existants [YEN 99], [YEN 01].

2.8.4. Machine à vecteurs support (SVM) incrémentale

Les machines à vecteurs supports (SVM) sont des systèmes de classification très puissants qui vont déterminer la séparatrice maximisant la marge entre les classes. Cette marge est définie comme la distance entre la séparatrice et les individus les plus proches, appelés vecteurs supports.

Les premiers algorithmes d'apprentissage incrémental de machine à vecteurs supports ont été proposés dans [SYE 99] et [RÜP 01]. Cette approche vise à gérer l'ensemble de vecteurs supports de manière incrémentale. Lorsqu'une nouvelle donnée d'apprentissage est mal classée ou à

l'intérieur de la marge, la séparatrice est recalculée à partir des vecteurs supports et de ce nouveau point.

Le principal inconvénient des SVM incrémentales est leur difficulté à gérer l'ajout de nouvelles classes. Une classe récemment ajoutée ne possèdera que peu d'individus, et en maximisant la marge, une SVM aura tendance à isoler ces quelques points sans vraiment généraliser la représentation de cette nouvelle classe.

[BOU 09] propose un algorithme permettant de mettre à jour une SVM de manière incrémentale, mais aussi de manière décrémentale. Cette méthode permet d'apprendre à partir de nouvelles données, mais également d'oublier ce qui a été appris à partir de certaines anciennes données lorsqu'elles deviennent obsolètes.

2.8.5. Learn++

Learn++ [POL 01] a été inspiré de l'algorithme AdaBoost[6] (adaptive boosting). Les deux génèrent un ensemble de classifieurs (hypothèses) faibles, qui sont obtenues en réapprenant un classifieur de base sur des distributions mises à jour stratégiquement de la base de données d'apprentissage, et les combinent à travers le vote majoritaire pondéré des classes prédites (sorties) par les hypothèses individuelles pour obtenir la règle de classification finale. La différence entre les deux est dans la règle de mise à jour de la distribution qui est optimisée dans AdaBoost pour améliorer l'exactitude du classifieur, alors que celle de Learn++ est optimisée pour l'apprentissage incrémental de nouvelles données, en particulier quand les nouvelles données introduisent des nouvelles classes.

Dans le cadre de l'apprentissage incrémental, les exemples qui ont une grande probabilité d'engendrer une erreur sont précisément ceux qui sont inconnus ou qui n'ont pas encore été utilisés pour apprendre le classifieur. Cette information était derrière le principe de base de Learn++ qui est de générer de nouveaux "classifieurs faibles" pour des portions invisibles précédemment de l'espace de caractéristiques. Chaque nouveau classifieur ajouté à l'ensemble est appris en utilisant un ensemble d'exemples prélevé d'une distribution qui assure que les exemples qui sont mal classés par l'ensemble courant de classifieurs ont une grande probabilité d'être prélevés, qui est l'objectif de la règle de mise à jour de la distribution qui donne à ces exemples des poids plus élevés.

Les grandes difficultés de cet algorithme sont : la création des sous-ensembles d'apprentissage et le choix de la règle pour la combinaison des décisions de ces classifieurs.

2.8.5.1. L'algorithme d'apprentissage Learn++

L'algorithme Learn++ proposé dans [POL 01] reçoit en entrée une distribution d'exemples d'apprentissage à partir de laquelle les sous-ensembles d'apprentissage sont choisis ; un algorithme d'apprentissage faible WeakLearn qui sera utilisé comme classifieur de base ; et le nombre d'itérations qui indique le nombre de classifieurs à être générés.

A chaque étape de l'apprentissage on considère qu'un seul ensemble d'exemples est disponible. Le système va entraîner plusieurs classifieurs sur plusieurs sous-ensembles de l'ensemble d'exemples

[6] AdaBoost : Originalement développé pour améliorer la performance de classification de classifieurs faibles. L'idée est qu'un apprenant faible qui peut à peine être un peu mieux que l'estimation aléatoire peut être transformé à un apprenant fort qui presque toujours accomplit arbitrairement un taux d'erreurs bas en utilisant une procédure appelée boosting.

disponible. Pour créer ces sous-ensembles d'apprentissage on affecte un poids à chaque exemple, ce poids servira à définir si cet exemple appartient à un sous-ensemble.

Pour la première itération, les poids $w_1(i)$ sont initialisés à 1/m, pour donner une probabilité égale à chaque exemple d'appartenir au premier sous-ensemble d'apprentissage, à moins qu'il y ait une raison suffisante d'initialiser autrement.

WeakLearn génère des hypothèses multiples en utilisant des sous-ensembles différents des données d'apprentissage S_k et chaque hypothèse apprend seulement une portion de l'espace d'entrée. Cela est accompli par mettre à jour itérativement une distribution D_t, t=1, 2, …, T_k à partir de laquelle les sous-ensembles d'apprentissage sont choisis. La distribution elle-même est obtenue en normalisant un ensemble de poids assignés à chaque exemple en se basant sur la performance de classification des classifieurs sur cet exemple.

À chaque itération t=1, 2, …, T_k, Learn++ dichotomise S_k en un sous-ensemble d'apprentissage TR_t et un sous-ensemble de test TE_t d'après la distribution courante D_t, et appelle WeakLearn à générer l'hypothèse h_t en utilisant le sous-ensemble d'apprentissage TR_t. Ensuite on calcule l'erreur ε_t de ce classifieur sur l'ensemble $S_k=TR_t+TE_t$. Si $\varepsilon_t > 1/2$, h_t est ignoré et des nouveaux TR_t et TE_t sont sélectionnés. Sinon l'erreur normalisée β_t ($0 < \beta_t < 1$) est calculé comme :

$$\beta_t = \varepsilon_t / (1 - \varepsilon_t)$$

Le fait que ε_t soit inférieure à 1/2 assure que β_t soit inférieure à 1.

Toutes les hypothèses générées dans les itérations antérieures seront ensuite combinées en utilisant le vote majoritaire pondéré. Une décision de classification est alors prise en se basant sur les sorties combinées d'hypothèses individuelles qui constituent l'hypothèse générée H_t qui est définie comme étant :

$$H_t = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} log\left(1/\beta_t\right)$$

Notant que H_t choisit la classe qui reçoit le vote total le plus haut de toutes les hypothèses. L'erreur générée faite par H_t est alors calculée comme :

$$E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i) = \sum_{i=1}^{m} D_t(i)\,[|H_t(x_i) \neq y_i|]$$

sur les exemples mal classés, où $[|\ .\ |]$ est 1 si le prédicat est vrai, et 0 autrement. Si $E_t > 1/2$, le h_t courant est abandonnée, un nouveau sous-ensemble d'apprentissage est sélectionné et une nouvelle h_t est générée. Sinon, l'erreur normalisée générée est calculée comme :

$$B_t = E_t / (1 - E_t)$$

Les poids w_t (i) sont ensuite mis à jour, pour calculer la prochaine distribution D_{t+1} qui sera utilisé pour sélectionner les prochains sous-ensembles d'apprentissage et de test, TR_{t+1} et TE_{t+1}, respectivement. La règle de mise à jour de la distribution constitue le cœur de l'algorithme, parce qu'elle permet à Learn++ d'apprendre de façon incrémentale :

$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & si\ H_t(x_i) = y_i \\ 1, & autrement \end{cases}$$

$$= w_t(i) \times B_t^{1-[|H_t(x_i)\neq y_i|]}$$

D'après cette règle, si l'exemple x_i est classé correctement par l'hypothèse générée H_t, son poids est diminué. Si x_i est mal classé, son poids de distribution est gardé inchangé. Cette règle réduit la

probabilité des exemples classés correctement d'être choisies dans TR_{t+1}, bien qu'elle augmente la probabilité des exemples mal classés pour être sélectionnés dans TR_{t+1}.

Après que T_k hypothèses soient générées pour chaque base de données D_k, l'hypothèse finale générée par cet algorithme est obtenue par le vote majoritaire pondéré de toutes les hypothèses générées :

$$H_{final} = \arg\max_{y \in Y} \sum_{k=1}^{K} \sum_{t:H_t(x)=y} log\left(1/\beta_t\right)$$

L'utilisation de cette hypothèse rend l'apprentissage incrémental possible particulièrement quand des exemples de nouvelles classes sont introduits, puisque ces exemples seront mal classés par l'hypothèse générée et forcés dans le prochain ensemble de données d'apprentissage.

Algorithme 1 : L'algorithme Learn++

Entrées : Pour chaque base de données tirée de D_k k=1, 2, …, K

- Séquence de m exemples d'entraînement $S = [(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)]$.
- Algorithme d'apprentissage faible WeakLearn.
- Entier T_k, spécifiant le nombre d'itérations.

Pour k=1, 2, …, K Faire :

Initialiser $w_1(i) = D(i) = 1/m$, $\forall i$, à moins qu'il y ait connaissance antérieure pour sélectionner autrement.

Pour $t=1, 2, \ldots, T_k$ Faire :

1. Mettre $D_t = w_t / \sum_{i=1}^{m} w_t(i)$ pour que D_t est une distribution.
2. Choisir aléatoirement les sous-ensembles d'entraînement TR_t et de test TE_t selon D_t.
3. Appeler WeakLearn, lui fournir avec TR_t.
4. Récupérer une hypothèse h_t : X→Y, et calculer l'erreur de h_t : $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$ sur $S_t = TR_t + TE_t$. Si $\varepsilon_t > 1/2$, mettre *t=t-1,* ignorer h_t et aller à étape 2. Autrement, calculer l'erreur normalisée comme $\beta_t = \varepsilon_t/(1-\varepsilon_t)$.
5. Appeler la majorité pondérée, obtenir l'hypothèse générée $H_t = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} log\left(1/\beta_t\right)$, et calculer l'erreur générée $E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i) = \sum_{i=1}^{m} D_t(i)[|H_t(x_i) \neq y_i|]$. Si $E_t > 1/2$, mettre *t=t-1,* ignorer H_t et aller à étape 2.
6. Mettre $B_t = E_t/(1-E_t)$ (l'erreur générée normalisée), et mettre à jour les poids des instances :

$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & si\ H_t(x_i) = y_i \\ 1, & autrement \end{cases}$$

$$= w_t(i) \times B_t^{1-[|H_t(x_i)\neq y_i|]}$$

Appeler la majorité pondérée sur les hypothèses combinées H_t et produire l'hypothèse finale :

$$H_{final} = \arg\max_{y \in Y} \sum_{k=1}^{K} \sum_{t:H_t(x)=y} log\left(1/\beta_t\right)$$

2.8.5.2. Propriétés importantes de Learn++

- Learn++ peut apprendre à partir de nouvelles données même lorsque les données introduisent des nouvelles classes.
- La connaissance précédente n'est pas perdue puisque tous les classifieurs précédents sont conservés et l'apprentissage incrémental est accompli à travers générer des classifieurs supplémentaires.
- Learn++ est indépendant du classifieur de base utilisé comme apprenant faible. Tout algorithme d'apprentissage faible peut servir comme classifieur de base de Learn++, en particulier, Learn++ peut être utilisé pour convertir tout classifieur supervisé, originalement incapable d'apprendre de façon incrémentale, à un qui peut apprendre de nouvelles données.
- Il est insensible à l'ordre de présentation des données d'apprentissage.
- Il est insensible aux ajustements mineurs des paramètres de l'algorithme.
- Learn++ garantit la convergence sur tout ensemble de données d'apprentissage donné, en réduisant l'erreur de classification avec chaque hypothèse ajoutée. Un théorème déclaré et prouvé dans [POL 01] lie la borne supérieure de l'erreur totale de Learn++ aux erreurs individuelles de chaque hypothèse :
  $E \leq 2^T \prod_{t=1}^{T} \sqrt{E_t.(1-E_t)}$ où E_t est l'erreur de la $t^{ème}$ hypothèse générée H_t.
- L'utilisation des apprenants forts n'est pas recommandée puisqu'il y a peu à être gagné par leur combinaison, et/ou ils peuvent mener à un sur-apprentissage des données.
- L'utilisation des apprenants faibles élimine le problème de précision de réglage et sur-apprentissage puisque chaque apprenant se rapproche seulement à peu près de la borne de décision.
- L'algorithme a deux composants clés qui peuvent être améliorés. Le premier est la sélection de l'ensemble de données d'apprentissage ultérieur qui dépend de la règle de mise à jour de la distribution. Le deuxième est la règle de combinaison d'hypothèse. Actuellement, les votes pondérés[7] sont déterminés en se basant sur les performances des hypothèses sur leur propre sous-ensemble de données d'apprentissage.

2.8.6. IGNG

L'algorithme IGNG (Incremental Growing Neural Gas) [PRU 06] est une extension de l'algorithme GNG (Incremental Growing Neural Gas) proposé dans [FRI 95]. Les deux algorithmes sont constructifs et permettent l'extraction de la topologie des données au sens non supervisé. Ils effectuent une construction incrémentale d'un graphe avec une structure et une dimension variable dans l'espace de représentation. L'ajout de nœuds dans GNG s'effectue à chaque fois qu'un nombre fixe de données sont présentées, par contre dans IGNG, les nœuds sont créés uniquement en cas de besoin, et non à étapes régulières.

Un graphe IGNG est un réseau dans lequel chaque nœud $n_i \in N$ est associé à un modèle des données qu'il représente M_{n_i}, une probabilité d'existence $P(n_i)$ et une position dans l'espace de représentation w_{n_i} ; et chaque arc $a_{n_i \to n_j} \in A$ entre deux nœuds n_i et n_j est associé à une probabilité d'existence $P(a_{n_i \to n_j})$ et représente une continuité des données entre les deux modèles

[7] L'idée du vote majoritaire pondéré est : "Si une grande majorité des hypothèses h_t est d'accord sur la classe d'un exemple particulier, alors œla peut être interprété comme l'algorithme ayant la haute confiance dans la décision finale".

associés aux deux nœuds qu'il relie (n_i et n_j). Le modèle associé à un nœud dépend essentiellement du type de représentation des données (vecteur de réels, graphe, …).

L'algorithme IGNG fonctionne selon le principe le suivant :

Au départ il n'y a pas de nœud, le premier nœud est créé lors de la présentation de la première donnée et il est initialisé à sa position dans l'espace de représentation.

A chaque fois qu'un "paquet" de données est disponible, il est présenté plusieurs fois pour effectuer l'apprentissage afin d'affiner la position de chaque nœud du graphe, chaque présentation de tout le "paquet" de données en cours d'apprentissage est appelée cycle. A chaque étape, une donnée à apprendre est tirée aléatoirement parmi les données disponibles et le graphe est adapté, jusqu'à ce qu'il n'y ait plus de donnée à apprendre. Ceci est recommencé selon le nombre de cycles qui a été prédéfini.

Lors de la présentation au réseau d'une nouvelle donnée x à apprendre, on cherche s'il existe des modèles M_{n_i} associés à des nœuds n_i tel que la probabilité que la donnée présentée appartienne à ces modèles soit supérieure à un seuil fixé au préalable θ c.à.d. $P_{M_{n_i}}(x) > \theta$, on distingue trois cas :

- Si un seul nœud s_1 vérifiant cette condition existe : on adapte sa position en le déplaçant au centre des données qui s'y projettent, et tous les arcs émanants de ce nœud voient leur probabilité d'exister diminuer.
- S'il existe plusieurs nœuds vérifiant cette condition : on crée un arc entre les deux nœuds les plus probables s_1 et s_2 en mettant sa probabilité d'existence au maximum et si cet arc existe déjà on remet sa probabilité d'existence au maximum. La probabilité d'existence associée à chaque arc est donnée par : $P\left(a_{n_i \to n_j}\right) = e^{-\frac{\gamma_{n_i \to n_j}}{\beta}}$ avec $\gamma_{n_i \to n_j}$ est l'âge de l'arc $n_i \to n_j$ et β est un paramètre à fixer a priori.
- Si aucun nœud ne vérifie cette condition : alors un nouveau nœud est créé et initialisé à la position de la donnée. La probabilité d'existence de ce nœud est initialisée au minimum pour permettre une résistance aux données bruitées et elle va augmenter au fur et à mesure que des données viennent se projeter dans ce nœud. La probabilité d'existence d'un nœud n_i est donnée par $P_{n_i} = 1 - e^{-\frac{\tau_{n_i}}{\alpha}}$ avec τ_{n_i} le nombre de fois où n_i a été considéré comme le nœud le plus probable et α est un paramètre à fixer a priori.

Enfin tout arc dont la probabilité d'existence devient inférieure ou égale à un seuil $\emptyset$ donné sera supprimé.

Les nœuds dont la probabilité d'existence devient inférieure ou égale à un seuil ψ ne seront pas supprimés mais ne seront considérés qu'en phase d'apprentissage et non en phase d'utilisation. Ces nœuds correspondent en réalité à des hypothèses de modèles de données en attente de confirmation.

L'algorithme d'apprentissage d'une seule donnée est le suivant :

Algorithme : L'algorithme IGNG

Entrée : X, un ensemble de données

Sortie : G, un graphe (IGNG)

début

pour i allant de 0 à nbcycle **faire**

pour j allant de 0 à nbdedonnées **faire**

effectuer un tirage sans remise d'une donnée x de l'ensemble X ;

$s_1 = arg \max_{n_i \in N} P_{M_{n_i}}(x)$;

$s_2 = arg \max_{n_i \neq s_1 \in N} P_{M_{n_i}}(x)$;

si $P_{M_{s_1}}(x) \leq \theta$ **alors**

$w_{n_{new}} = x$;

$P_{n_{new}} = 1 - e^{-\frac{1}{\alpha}}$;

$N = N \cup n_{new}$;

Sinon

$\tau_{s_1} = \tau_{s_1} + 1$;

$\forall n_i \in N, \Delta w_{n_i} = \begin{cases} \frac{\|x - w_{n_i}\|}{\tau_{n_i}} & si\ P_{M_{n_i}}(x) = \max_j P_{M_{n_j}}(x) \\ 0\ sinon \end{cases}$;

$\forall n_j / \exists\ a_{s_1 \to n_j}, \gamma_{s_1 \to n_j} = \gamma_{s_1 \to n_j} + 1$;

si $\exists a_{s_1 \to s_2}$ **alors**

$\gamma_{s_1 \to s_2} = 1$

sinon

si $P_{M_{s_2}}(x) > \theta$ **alors**

$\gamma_{s_1 \to s_2} = 1$

$A = A \cup \{a_{s_1 \to s_2}\}$

si $\exists a_{n_i \to n_j} \in A / P(a_{n_i \to n_j}) \leq \emptyset$ **alors**

$A = A - \{a_{n_i \to n_j}\}$

fin

Dans la phase d'utilisation du graphe, seuls les nœuds dont la probabilité d'existence est supérieure à un seuil ψ sont considérés. Ceci implique que seuls les arcs qui relient deux nœuds dont la probabilité d'existence est supérieure à ψ seront utiles dans cette phase.

2.8.7. Le Perceptron Auto-Organisé PAO (SOP : Self-Organizing Perceptron)

Le perceptron auto-organisé PAO est une architecture neuronale hybride composée d'un perceptron multicouche (PMC) combiné à une carte auto-organisatrice quelconque (par exemple, une carte de Kohonen, une carte GCS, une carte GNG ou autres). Cette carte sert à extraire les informations topologiques des données d'apprentissage pour orienter l'information dans le réseau PMC [HEB 99a, HEB 99b, HEB 99c, HEB 00, PRU 06].

L'architecture hybride du réseau PAO permet de lier l'ensemble des neurones d'une carte auto-organisatrice préalablement entraînée aux neurones cachés d'un PMC afin de partitionner ce dernier en groupes de neurones spécialisés (chacun d'entre eux spécialisé à résoudre une partie spécifique du problème). Ce partitionnement est réalisé en reliant chaque neurone de la carte auto-organisatrice à un neurone de la dernière couche cachée du PMC, tel qu'illustré à Fig 2.5. Ce lien entre les deux réseaux requiert que le nombre de neurones au sein de la carte auto-organisée

corresponde exactement au nombre de neurones situés sur la couche cachée du PMC. Le PMC et la carte auto-organisatrice sont alimentés par le même vecteur d'entrée. Le nombre de neurones de sortie du PMC doit être connu dès le début de l'apprentissage c.à.d. le nombre maximal de classes doit être fixé.

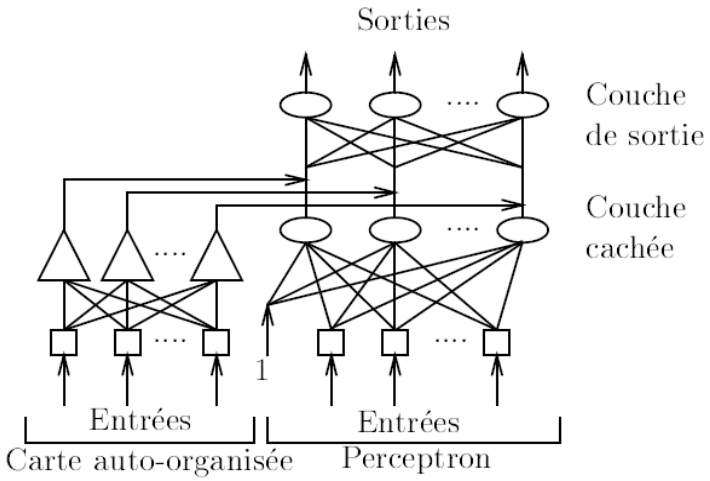


Figure 2.5 : Architecture hybride du réseau PAO

Le PAO peut fonctionner selon deux modes d'apprentissage :

- l'apprentissage passif : il est réalisé en deux phases successives. Premièrement, la carte auto-organisée est entraînée, puis l'apprentissage du PMC s'effectue selon l'algorithme de rétro-propagation des erreurs.
- l'apprentissage incrémental : le phénomène de partitionnement dynamique permet au réseau d'apprendre de façon incrémentale, c'est-à-dire qu'il peut être entraîné séquentiellement sur des ensembles de données indépendants.

Grâce à ce réseau, il est possible d'affiner le taux de reconnaissance en faisant réapprendre les classes et les données qui ont été mal apprises.

La stratégie d'apprentissage incrémental du PAO repose sur la définition de plusieurs ensembles de données de vigilance locales qui visent à limiter l'influence que pourrait avoir une nouvelle donnée sur le PAO et permettent de mettre à jour l'état des connaissances de ce dernier en fonction de l'évolution temporelle du problème posé. Grâce à la spécialisation des neurones cachés du réseau PAO, il est possible d'assigner chaque donnée de vigilance à une zone (ou une branche) bien précise du réseau PAO, d'où l'appellation de données de vigilance locales.

L'apprentissage du perceptron auto-organisé par une nouvelle donnée x commence par déterminer quel est le neurone gagnant n_1 de la carte auto-organisée. Ensuite, il faut déterminer N' l'ensemble des neurones voisins à n_1, ceux fortement activés par la donnée x, qui sont déterminés selon :

$$N' = \{n_j \in N / ||x - w_j|| \leq \delta \times \sigma_j\}$$

δ est un paramètre utilisé pour pondérer la sphère d'influence σ_j de chaque neurone n_j. On récupère ensuite tous les ensembles de vigilance associés au neurone gagnant ainsi qu'aux neurones appartenant à N'. Une nouvelle phase d'apprentissage du PAO est ensuite réalisée sur un ensemble constitué de toutes ces données et de la donnée x. On vérifie que celle-ci soit correctement apprise. Si c'est le cas, on ajoute x à l'ensemble de vigilance de n_1 et l'apprentissage

de cette donnée est terminé. Dans le cas contraire, s'il existe un neurone de la carte auto-organisée récent qui est activé par la donnée x alors on déplace ce neurone vers la donnée x. Si un tel neurone n'existe pas et que le neurone gagnant est activé par la donnée x, un nouveau neurone est créé avec comme position dans l'espace de représentation celle de la donnée x. Ce nouveau neurone est ensuite connecté au neurone n_1. Si le neurone gagnant n'est pas activé par la donnée x, alors deux nouveaux neurones sont créés et sont connectés entre eux. Un de ces neurones a la position de la donnée x et l'autre a pour position :

$$w_{new\ 2} = w_1 + \epsilon(x - w_1)$$

Les données de vigilance sont remises à jour et le réseau est entraîné à nouveau, jusqu'à ce que la donnée x soit bien apprise.

Algorithme : Algorithme d'apprentissage incrémental du PAO

Entrée : X, un ensemble d'apprentissage

début

1. Tirer au hasard une donnée d'apprentissage x associée à sa classe c et déterminer le neurone gagnant $n_1(x)$
2. Déterminer l'ensemble $N' \subset N$ des neurones voisins tel que :
$$N' = \{n_j \in N / ||x - w_j|| \leq \delta \times \sigma_j\}$$
où δ est un paramètre utilisé pour pondérer la sphère d'influence σ_j de chaque neurone n_j.
3. Entraîner le PAO avec les données :
$$X' = x \cup (\cup_{n_j \in N'} V_j),$$
V_j correspondant à l'ensemble de vigilance de n_j.
4. **si** la donnée est correctement apprise **alors**
aller à l'étape 6

   sinon

   si $\exists n_j \in N'$ tel que n_j est un neurone récent et que $\|x - w_j\| \leq \sigma_{n_j}$ **alors**

   le neurone est déplacé et sa nouvelle position est $w_j' = w_j + \epsilon(x - w_j)$.

   Sinon

   si $||x - w_1|| \leq n_1$ **alors**

   Un neurone est ajouté avec $w_{new} = x$ et ce neurone est connecté au neurone gagnant

   Sinon

   Deux neurones sont créés avec $w_{new\ 1} = x$ et $w_{new\ 2} = w_1 + \epsilon(x - w_1)$. Ces deux neurones sont reliés entre eux et disjoints de la structure existante.
5. Mettre à jour les ensembles de vigilance et ré-entraîner le réseau
6. Mettre la donnée x au sein de l'ensemble de vigilance V_1 et retourner à l'étape 1.

fin

2.8.8. AI2P : modèle d'Apprentissage Incrémental en 2 Phases

AI2P [ALM 08] est un modèle d'apprentissage incrémental à base de prototypes qui est exécuté en deux phases pour apprendre progressivement un système de reconnaissance et arriver à un apprentissage incrémental rapide et fiable.

- La première phase correspond à un apprentissage rapide et consiste à créer un nouveau prototype pour chaque nouvel exemple afin d'acquérir le maximum possible de connaissances.
- Par contre, la deuxième phase l'apprentissage consiste à adapter les connaissances existantes en modifiant les prototypes existants à chaque nouvel exemple.

Notant que les processus d'apprentissage et d'adaptation sont supervisés.

En effet, la création des prototypes pour chaque nouvel exemple est coûteuse en termes de temps et de mémoire, et donc, l'utilisation de l'adaptation devient indispensable afin d'avoir un système à la fois dynamique et léger.

Algorithme 1 : L'algorithme de AI2P

pour chaque *nouvel exemple e de la classe C* faire
  si *classe C est dans phase 1* alors
    appeler l'algorithme de création de prototypes avec *e*;
    appeler l'algorithme d'adaptation;
    si *nombre d'exemples de classe C* ≥ *N* alors
      basculer vers phase 2
    fin
  fin
  si *classe* C *est dans phase 2* alors
    appeler l'algorithme d'adaptation;
    si *e est mal-classé* alors
      nbErr[*C*] + +;
      si *nbErr*[*C*] ≥ *S* alors
        appeler l'algorithme de création de prototypes;
        nbErr[*C*] = 0;
      fin
    fin
  fin
fin

2.8.8.1. Phase 1 : Apprentissage incrémental rapide

Dans cette phase, un prototype est créé autour de chaque nouvel exemple. Ses conclusions sont initialisées à 1 pour la conclusion qui correspond à la classe du caractère, et à 0 pour toutes les autres classes. Puis tous les prototypes existants et leurs conclusions sont adaptés pour ajuster globalement le système au nouvel exemple. Le système passe de la phase 1 à la phase 2 pour une classe donnée après avoir eu *N* exemples (*N* prototypes) de cette classe.

2.8.8.2. Phase 2 : Apprentissage par adaptation

Dans cette phase, l'adaptation permet de modifier les prototypes existants pour prendre en compte les connaissances acquises par les nouveaux exemples. Un nouveau prototype est créé pour une classe donnée quand le nombre d'erreurs de reconnaissance (nbErr) pour cette classe dépasse un seuil défini (S).

2.8.9. AttributeNets

AttributeNets [WU 07] est une méthode de classification incrémentale inspirée de la structure de données nommée Graphe de Concepts (GC) [ENE 05] pour l'apprentissage incrémental. Les deux sont composées de plusieurs couches d'attributs dont chacune correspond à un attribut et les nœuds de chaque couche représentent les valeurs possibles pour l'attribut correspondant. La différence entre la structure de GC et celle de AttributeNet est que GC a une couche de classification dont chaque nœud correspond à une catégorie, alors que cette couche n'existe pas dans AttributeNets parce que chaque AttributeNet fait référence à seulement une catégorie. Concernant d'apprentissage, GC enregistre tous les cas en attachant la séquence de numéros de ces cas au nœud correspondant lorsque la valeur de l'attribut égale la valeur du nœud pour chaque couche d'attribut et pour le nœud de classification à qui le cas appartient. Au lieu de mémoriser chaque détail historique de cas (attacher la séquence de numéros de cas à chaque nœud), AttributeNets enregistre seulement l'information statistique des valeurs d'attributs des cas appris. Chaque nœud garde un compteur (degré du nœud) pour enregistrer combien de cas appartiennent à ce nœud ; pour chacun de deux nœuds, un autre compteur (degré de lien) est gardé enregistrant combien de cas appartiennent aux deux nœuds.

2.8.9.1. La structure AttributeNets

La structure AttributeNets fait référence à la combinaison de structures isomorphiques individuelles appelées AttributeNet.

Pour chaque catégorie, dans un problème de classification, une AttributeNet est construit et est composé de plusieurs couches d'attributs comprenant des nœuds d'attribut, chaque couche correspond à un attribut spécifique de cas et un nœud dans la couche correspond à une valeur spécifique de l'attribut correspondant. Ainsi, le nombre de AttributeNet individuels dans AttributeNets est égale au nombre de catégories pour le problème de classification considéré ; le nombre de couches dans chaque AttributeNet est égale au nombre d'attributs de la catégorie correspondante et le nombre de nœuds dans chaque couche est égale au nombre de valeurs que peut prendre l'attribut correspondant.

Un nœud représente une valeur spécifique d'un attribut et garde un compteur (degré du nœud) comptant le nombre de cas qui ont cette valeur pour l'attribut spécifique. Nous disons qu'un nœud A_{ij} est activé par un cas si le $i^{ème}$ attribut du cas a la valeur du nœud de A_{ij}.

Chaque couche représente un attribut spécifique de cas. Donc une couche est composée de plusieurs nœuds représentants les valeurs correspondantes à cet attribut, le nombre de nœuds dans une couche est égale au nombre de valeurs possibles pour l'attribut correspondant.

Il y a des liens entre tous deux nœuds de couches différentes. Si un cas appartient aux deux nœuds, le degré du lien entre ces deux nœuds augmente par 1. Le degré du lien initial de tous deux nœuds est 0. Le degré du lien entre tous deux nœuds de la même couche est toujours 0.

La méthode AttributeNets est basée sur deux algorithmes différents : un algorithme d'apprentissage et un algorithme de classification.

2.8.9.2. L'algorithme d'apprentissage AttributeNets

L'algorithme d'apprentissage incrémental AttributeNets est basé sur la structure appelée AttributeNets décrite dans la section précédente. Le processus d'apprentissage AttributeNets consiste à mémoriser l'information statistique des valeurs d'attribut et les relations entre tout deux de valeurs d'attributs différents, avec considération des cas de la catégorie propre de net seulement.

Algorithme 1 : (algorithme d'apprentissage AttributeNets)

Entrée : AttributeNets ($Attr_i$, $1 \leq i \leq ||Catégories||$) à être mis à jour, nouveau cas d'entraînement (*Cas*)
Sortie : AttributeNets mis à jour
Etape1 : i = *catégorieDe* (*Cas*)
Etape2 : Pour $1 \leq j \leq ||Couches||$
*degré*_nœud [j][k]++ (*degré*_nœud [j][k] est le degré du nœud$_{jk}$ qui est un nœud de la couche j de $Attr_i$ et est activé par *Cas*)
Etape3 : Pour $1 \leq j \leq ||Couches||$
Pour $1 \leq u \leq ||Couches||$
*degré*_nœud [j][k][u][v]++ (*degré*_nœud [j][k][u][v] est le degré du lien du nœud entre les nœuds activés i.e. nœud$_{jk}$ de la couche j et nœud uv de de la couche u de $Attr_i$)
Etape4 : Fin

Lorsqu'un cas d'apprentissage de la catégorie i vient, AttributeNet$_i$ est activé pendant qu'autres nets autre que la catégorie i sont simplement ignorés par ce cas. Avec AttributeNet$_i$, pour chaque attribut du cas, c.-à-d. chaque couche d'AttributeNet$_i$, nous augmentons le degré de nœud si cet attribut a la valeur identique à la valeur du nœud. Pour tous deux nœuds de couches différentes, nous augmentons le degré du lien entre ces deux nœuds par 1 si les deux nœuds sont activés par le cas.

L'AttributeNets est appris cas par cas et le résultat de l'apprentissage est indépendant de l'ordre dans lequel les cas sont appris. Quand un nouveau cas vient, nous avons besoin seulement d'augmenter le degré du nœud des nœuds et le degré du lien des liens de nœud qu'il active. Le coût de temps et de mémoire du processus d'apprentissage le sont O (n2), où n est le nombre de nœuds d'AttributeNets.

2.8.9.3. L'algorithme de classification AttributeNets

Dans cette section, l'algorithme de classification basé sur AttributeNets est donné.

Algorithme 2 : (algorithme de classification AttributeNets)

Entrée : AttributeNets ($Attr_i$, $1 \leq i \leq ||Catégories||$) appris, nouveau cas (*Cas*) avec sa catégorie inconnue
Sortie : Catégorie c du Cas
Etape1 : Pour $1 \leq i \leq ||Catégories||$
$r_i = 1$
Etape2 : Pour $1 \leq i \leq ||Catégories||$
Pour $1 \leq j \leq ||Couches||$

$r_i = r_i \times \sqrt{degré\_nœud\ [i][j] + \Delta}$

(*degré_nœud* [*i*][*j*] est la valeur du nœud activé par Cas dans la couche *j* de $Attr_i$, Δ est un petit nombre empêchant r_i d'être 0)

Etape3 : Pour 1≤ *i* ≤||*Catégories*||

Pour 1≤ *j* ≤ ||*Couches*||

Pour 1≤ *k* ≤ ||*Couches*||

$r_i = r_i \times (degré_{lien}\ [i][j][k] + \Delta)$

(*degré_lien* [*i*][*j*][*k*] est la valeur du nœud lien entre les nœuds activés de la couche *j* et la couche *k* de $Attr_i$, Δ est un petit ajustement empêchant r_i d'être 0)

Etape4 : Retourner i qui *Maximize* (r_i)

2.8.9.4. Avantages de AttributeNets

- Il est en lui-même un classifieur multi-catégorie à cause de la structure multi-nets.
- Il est exceptionnel dans l'utilisation de la mémoire et la vitesse d'adaptation qui sont d'importance vitale pour l'apprentissage incrémental, spécialement l'apprentissage en ligne.
- Les résultats de classification sont faciles à comprendre (AttributeNets pourrait générer des résultats efficaces interprétables aux êtres humains parce qu'il existe une injection entre les couches d'AttributeNets et les attributs de cas).
- Il est robuste aux données bruitées.
- Il est robuste au manque de cas d'apprentissage, il marche tout à fait bien avec seulement une petite taille de l'ensemble d'apprentissage disponible.

2.8.10. Méthode de clustering évolutif (Evolving Clustering Method ECM)

La méthode de clustering évolutif adaptatif ECM [KAS 07] est un algorithme rapide d'une seule-passe pour le clustering dynamique d'un flux de données d'entrée où il n'ya pas de nombre prédéfini de clusters. Il s'agit d'une méthode de clustering basée-distance où les centres des clusters sont représentés par des nœuds évolués dans un mode adaptatif. Pour tout tel cluster, la distance maximale *MaxDist* entre un point exemple x_i et le centre le plus proche du cluster, ne peut pas être plus grande qu'une valeur seuil *Dthr*, qui est un paramètre prédéfini de clustering. Ce paramètre affecterait le nombre de clusters évolués. La valeur seuil *Dthr* peut être réglable pendant le processus de clustering adaptatif, en fonction de certains critères d'optimisation et de réglage automatique, tels que l'erreur actuelle, le nombre de clusters, et ainsi de suite.

Le processus de clustering commence avec un ensemble vide de clusters. Quand un nouveau cluster C_j est créé, son centre de cluster Cc_j est défini et son rayon de cluster Ru_j est initialisé à zéro. Avec d'autres exemples présentés l'un après l'autre, certains clusters déjà créés seront mis à jour en changeant les positions de leurs centres et en augmentant leur rayon de cluster. Quel cluster sera mis à jour et combien il sera changé dépend de la position de l'exemple actuel de données dans l'espace d'entrée. Un cluster C_j ne sera plus mis à jour lorsque son rayon de cluster Ru_j atteint la valeur égale à la valeur seuil *Dthr*.

L'algorithme ECM

Etape 0 : Créer le premier cluster C_1 en prenant simplement la position du premier exemple du flux de données d'entrée comme étant le premier centre de cluster Cc_1, et en fixant la valeur 0 pour son rayon de cluster Ru_1.

Etape 1 : Si tous les exemples du flux de données ont été traités, le processus de clustering se termine. Sinon, l'exemple d'entrée actuel x_i est pris et la distance euclidienne normalisée D_{ij} entre cet exemple et tous les n centres de cluster déjà créés Cc_j, $D_{ij} = ||x_i - Cc_j||$, j = 1, 2, ..., n, est calculée.

Etape 2: S'il y a un cluster C_m avec un centre Cc_m, un rayon de cluster Ru_m, et une valeur de distance D_{im} tel que:

$D_{im} = ||x_i - Cc_m|| = \min\{D_{ij}\} = \min\{||x_i - Cc_j||\}$, pour j = 1, 2, … , n ; et

$D_{im} < Ru_m$

l'exemple actuel x_i est considéré comme appartenant à ce cluster. Dans ce cas, ni un nouveau cluster est créé, ni un cluster existant est mis à jour (modifié). L'algorithme retourne ensuite à Etape 1.

Sinon :

Etape 3 : Trouver un cluster C_a (avec un centre Cc_a, un rayon de cluster Ru_a, et une valeur de distance D_{ia}) qui a une valeur minimale S_{ia} :

$S_{ia} = D_{ia} + Ru_a = \min\{S_{ij}\}$, j = 1, 2, …, n.

Etape 4 : Si S_{ia} est supérieure à $2\times Dthr$, l'exemple x_i n'appartient à n'importe quel cluster existant. Un nouveau cluster est créé de la même manière que décrit dans Etape 0. L'algorithme retourne ensuite à Etape 1.

Sinon :

Etape 5: Si S_{ia} n'est pas supérieure à $2\times Dthr$, le cluster C_a est mis à jour en déplaçant son centre Cc_a et en augmentant sa valeur de rayon Ru_a. Le rayon mis à jour Ru_a^{new} est réglé pour être égal à $S_{ia}/2$ et le nouveau centre Cc_a^{new} est situé sur la ligne reliant le vecteur d'entrée x_i et l'ancien centre de cluster Cc_a, de sorte que la distance du nouveau centre Cc_a^{new} au point x_i est égal à Ru_a^{new}. L'algorithme retourne ensuite à l'étape 1.

De cette façon, la distance maximale séparant tout centre de cluster à l'exemple le plus loin qui appartient à ce cluster est maintenue inférieure à la valeur seuil *Dthr* bien que l'algorithme ne garde aucune information des exemples passés.

La fonction objective ici est très simple et elle est définie pour assurer que pour chaque exemple de données x_i il ya un centre de cluster C_j tels que la distance entre x_i et le centre de cluster Cc_j est inférieure à un seuil prédéfini *Dthr*.

Les règles évolutives de ECM incluent :

- Une règle pour la création d'un nouveau cluster
- Une règle pour la modification d'un cluster existant

2.8.11. Réseaux de neurones flous évolutifs (Evolving Fuzzy Neural Networks EFuNN)

Les réseaux de neurones flous évolutifs (EFuNNs) sont des structures connexionnistes qui peuvent être interprétées en termes de règles floues. Les EFuNNs ont des caractéristiques des systèmes basés-connaissance, de systèmes logiques, de systèmes de raisonnement à base des cas, et de systèmes connexionnistes adaptatifs.

2.8.11.1. L'architecture EFuNN

EFUNN est une structure à cinq couches (Figure 2.6). Ici, les neurones et les connexions sont créés/connectés lorsque des exemples de données sont présentés. Une couche de mémoire à court terme facultative peut être utilisée par une connexion de feedback depuis la couche de neurones de règles (également appelée de cas) (Figure 2.7). La couche de connexions de feedback pourrait être utilisée si les relations temporelles des données d'entrée doivent être mémorisées structurellement.

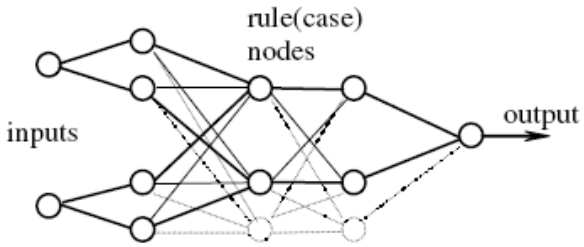


Figure 2.6 Le réseau de neurones flou évolutif EFuNN: un exemple d'un système EFuNN feed-forward standard simplifié [KAS 01b].

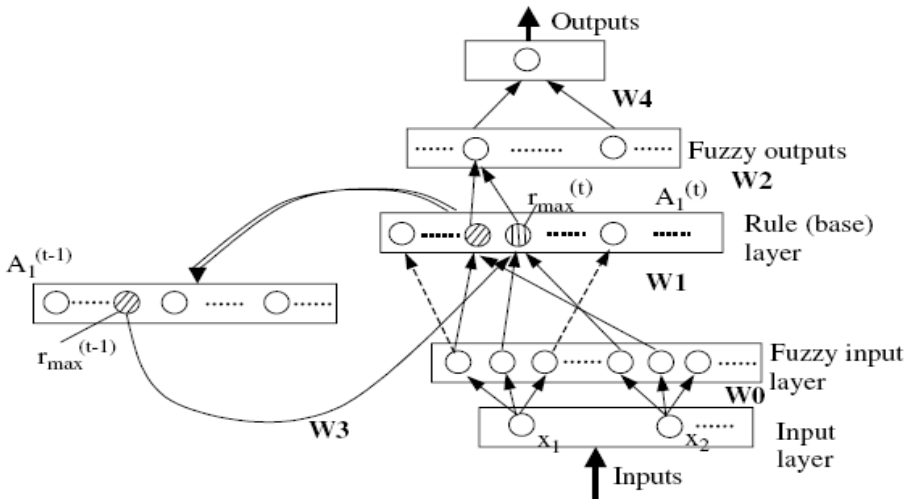


Figure 2.7 Un exemple d'EFuNN avec une mémoire à court terme réalisée en tant que connexion de rétroaction [KAS 01b].

La première couche est la couche des entrées. Les neurones de cette couche représentent les variables d'entrée.

La deuxième couche de neurones est la couche des entrées floues. Elle représente la quantification floue de chaque variable d'entrée. Différentes fonctions d'appartenance peuvent être attachées à ces neurones (triangulaire, gaussienne, etc.). Le nombre et le type de fonction d'appartenance peuvent être modifiés dynamiquement. La tâche des neurones d'entrée flous est

de transférer les valeurs d'entrée en degrés d'appartenance auxquelles elles appartiennent à la fonction d'appartenance correspondante.

La troisième couche est la couche des règles. Elle contient des neurones de règle (cas) qui évoluent à travers l'apprentissage supervisé et/ou non supervisé. Les neurones de règles représentent des prototypes (exemples, clusters) d'associations de données d'entrées-sorties qui peuvent être représentés graphiquement comme des associations d'hypersphères des espaces d'entrée floue et de sortie floue. Chaque neurone de règle r est défini par deux vecteurs de poids de connexion, W1(r) et W2(r), ce dernier étant ajusté par apprentissage supervisé en fonction de l'erreur de sortie et le premier étant ajusté par un apprentissage non supervisé basé sur une mesure de similarité dans une zone locale de l'espace du problème. Une fonction d'activation linéaire, ou une fonction gaussienne, est utilisée pour les neurones de cette couche.

La quatrième couche de neurones est la couche des sorties floues. Elle représente une quantification floue des variables de sortie, similaire à la représentation des neurones flous d'entrée. Ici, une fonction d'entrée de somme pondérée et une fonction d'activation linéaire saturée sont utilisées pour que les neurones calculent les degrés d'appartenance auxquels le vecteur de sortie associé au vecteur d'entrée présenté appartient à chacune des fonctions d'appartenance de sortie.

La cinquième couche est la couche des sorties. Elle représente les valeurs des variables de sortie. Ici, une fonction d'activation linéaire est utilisée pour calculer les valeurs définies pour les variables de sortie.

2.8.11.2. Algorithmes et règles d'apprentissage supervisé évolutif EFuNN

L'apprentissage évolutif dans EFuNN est basé sur l'une des hypothèses suivantes:

1. Aucun neurone de règle n'existe avant l'apprentissage et tous sont créés (générés) pendant le processus évolutif; ou
2. Il existe un ensemble initial de neurones de règles qui ne sont pas connectés aux neurones d'entrée et de sortie et qui se connectent à travers le processus (évolutif) d'apprentissage.

L'algorithme évolutif EFuNN présenté ci-dessous ne se différencie pas entre ces deux cas.

Chaque neurone de règle, par exemple, r_j, représente une association entre une hypersphère de l'espace d'entrée floue et une hypersphère de l'espace de sortie floue (Figure 2.8), les poids de connexion $W_1(r_j)$ représentant les coordonnées du centre de la sphère dans l'espace d'entrée floue, et $W_2(r_j)$ les coordonnées dans l'espace de sortie floue. Le rayon de l'hypersphère d'entrée d'un neurone de règle r_j est défini comme $R_j = 1\text{-}S_j$, où S_j est le paramètre de seuil de sensibilité définissant l'activation minimale du neurone de règle r_j à un nouveau vecteur d'entrée x à partir d'un nouvel exemple (x, y) afin que l'exemple soit pris en compte pour l'association avec ce neurone de règle.

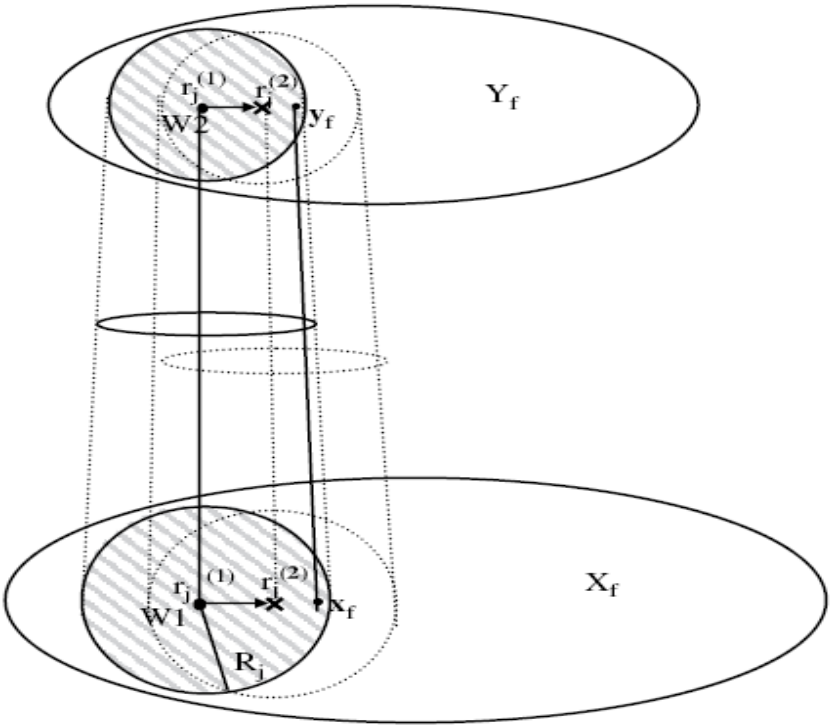


Figure 2.8 L'apprentissage adaptatif dans EFuNN: un neurone de règle représente une association de deux hypersphères de l'espace d'entrée floue et de l'espace de sortie floue; Le neurone de règle r_j 'se déplace' d'une position $r_i^{(1)}$ j à $r_j^{(2)}$ pour accommoder le nouvel exemple d'entrée-sortie (x_f , y_f) [KAS 07].

La paire de vecteurs de données d'entrée-sortie floues (x_f, y_f) sera attribuée au neurone de règle r_j si x_f tombe dans le champ réceptif de l'entrée r_j (hypersphère) et y_f tombe dans l'hypersphère du champ réactif de la sortie r_j. Ceci est assuré par deux conditions: qu'une différence floue normalisée locale entre x_f et $W1(r_j)$ est inférieure au rayon R_j, et l'erreur de sortie normalisée Err=||y-y'||/Nout est inférieure à un seuil d'erreur E. Nout est le nombre de sorties et y' est le produit par la sortie de EFuNN. Le paramètre d'erreur E définit la tolérance d'erreur du système.

Trois algorithmes d'apprentissage supervisé sont décrits ici qui diffèrent dans les formules d'ajustement de poids. Les mêmes formules sont applicables lorsque l'activation de m nœuds de règles est propagée et utilisée, ce qu'on appelle le mode « many-of-n » ou « m-of-n » pour faire court.

L'algorithme d'apprentissage EFuNN-s/u

Définir les valeurs initiales pour les paramètres du système: nombre de fonctions d'appartenance; seuil de sensibilité initiale S; seuil d'erreur E; paramètre d'agrégation N_{agg} (un certain nombre d'exemples consécutifs après lesquels une agrégation est effectuée); paramètres d'élagage OLD et *Pr*; une valeur pour m (en mode m-of-n); les seuils T_1 et T_2 pour l'extraction des règles.

Définir le premier neurone de règle pour mémoriser le premier exemple (x, y):

$$W1(r_0) = x_f \text{ et } W2(r_0) = y_f$$

Boucler sur les présentations des paires entrées-sorties (x, y)

{

Évaluer la distance floue normalisée locale D entre x_f et les connexions de neurone de règle existantes W1 :

$$D(x_f, W1) = || x_f - W1 || / || x_f + W1 ||$$

où $|| x_f - W1 ||$ désigne la somme de toutes les valeurs absolues d'un vecteur qui est obtenu après soustraction de vecteurs (ou sommation dans le cas de $|| x_f + W1 ||$) de deux vecteurs x_f et W1; / indique division.

Calculer l'activation A1 de la couche de neurone de règle. Trouver le neurone de règle le plus proche r_k (ou les m neurones de règle les plus proches en cas de mode m-of-n) au vecteur d'entrée flou x_f :

$$A1(r_k^{(t)}) = f_2(D(W1(r_k^{(t)}), x_f))$$

une fonction linéaire simple peut être utilisée pour f_2;

Si $A1(r_k) < S_k$ (seuil de sensibilité pour le neurone r_k); Créer un nouveau neurone de règle pour (x_f, y_f)

sinon

Trouver l'activation de la couche de sortie floue A2 = W2A1 et l'erreur de sortie Err = ||y-y'|| / N_{out}.

si Err >E

Créer un nouveau neurone de règle pour accommoder l'exemple actuel (x_f, y_f)

sinon

Mettre à jour $W1(r_k)$ et $W2(r_k)$ (dans le cas de l'EFuNN m-of-n mettre à jour tous les m neurones de règle avec l'activation A1 la plus élevée) :

$$W1(r_k(t+1)) = W1(r_k^{(t)}) + l_k.(x_f - W1(r_k^{(t)}))$$

$$W2(r_k(t+1)) = W2(r_k^{(t)}) + l_k.(y_f - A2).A1(r_k^{(t)})$$

où A2 est le vecteur d'activation des neurones de sortie flous dans la structure EFuNN lorsque x est présentée; A1 est l'activation du neurone de règle $r_j^{(t)}$; l_k est le taux d'apprentissage actuel du neurone de règle r_k.

Appliquer *la procédure d'agrégation* des neurones de règles après chaque groupe de N_{agg} exemples :

Pour l'agrégation de trois neurones de règles r_1, r_2 et r_3, les deux règles d'agrégation suivantes peuvent être utilisées pour calculer les nouvelles connexions de neurone de règle agrégée $W1(r_{agg})$ (les mêmes formules sont utilisées pour calculer les connexions W2):

(a) En tant que centre géométrique des trois neurones:

$$W1(r_{agg}) = (W1(r1)+W1(r2)+W1(r3))/3$$

(b) En tant que centre statistique pondéré:

$$W2(r_{agg}) = (W2(r1).Nex(r1)+W2(r2).Nex(r2)+W2(r3).Nex(r3))/Nsum$$

où

$$Nex(r_{agg}) = Nsum = Nex(r1)+Nex(r2)+Nex(r3)$$

Les trois neurones de règle agrégeront seulement si le rayon du champ réceptif du neurone agrégé est inférieur à un rayon maximal prédéfini Rmax:

$$Rr_{agg} = D(W1(r_{agg}))W1(r_j))+R_j <= Rmax,$$

r_j est le neurone de règle à partir des trois neurones qui ont une distance maximale du nouveau neurone r_{agg} et R_j est son rayon du champ réceptif.

Mettre à jour les paramètres S_k, R_k, Age(r_k), TA(r_k) pour le neurone de règle r_k :

$$Sk^{(t+1)} = Sk^{(t)} - D(W1(rk^{(t+1)}), W1(rk^{(t)}))$$

$$Rk^{(t+1)} = Rk^{(t)} + D(W1(rk^{(t+1)}), W1(rk^{(t)}))$$

Age(r_k) est calculé comme le nombre d'exemples qui ont été présentés au EFuNN après que r_k ait été créé pour la première fois;

TA(r_k), l'activation totale, est calculée comme le nombre d'exemples pour lesquels r_k a été le neurone gagnant correct.

Elaguer des neurones de règles si nécessaire, tel que défini par les paramètres d'élagage.

SI (Age(rj) > OLD) ET (l'activation totale TA(rj) est inférieure à un paramètre d'élagage Pr fois Age (rj)) ALORS élaguer le neurone de règle rj,

où Age(r_j) est calculé comme le nombre d'exemples qui ont été présentés au EFuNN après que rj ait été créé pour la première fois; OLD est une limite d'âge prédéfinie; *Pr* est un paramètre d'élagage dans l'intervalle [0,1], et l'activation totale *TA*(rj) est calculée comme le nombre d'exemples pour lesquels rj a été le neurone gagnant correct.

Extraire les règles à partir des neurones de règles

Chaque neurone de règle r_j peut être exprimé comme une règle floue, où les nombres attachés aux étiquettes floues indiquent le degré auquel les centres des hypersphères d'entrée et de sortie appartiennent à la fonction d'appartenance respective. Les degrés associés aux éléments de condition sont les poids de connexion de la matrice W1. Seulement les valeurs qui sont supérieures à un seuil T1 sont laissées dans les règles comme étant les plus importantes. Les degrés associés à la partie conclusion sont les poids de connexion de W2 qui sont supérieurs à un seuil de T2.

} Fin de la boucle principale.

L'algorithme d'apprentissage EFuNN-dp

Ceci est différent de l'EFuNN-s/u dans la formule d'ajustement des poids pour W2 :

$$W2(r_j^{(t+1)}) = W2(r_j^{(t)}) + lj.(y_f - A2)A1(r_j^{(t+1)})$$

ce qui signifie que, après la première propagation du vecteur d'entrée et le calcul de l'erreur *Err*, si les poids vont être ajustés, les poids W1 sont ajustés en premier), et ensuite le vecteur d'entrée $\boldsymbol{x}$ est propagé de nouveau à travers le neurone de règle déjà ajusté r_j à sa nouvelle position $r_j^{(t+1)}$ dans l'espace d'entrée, une nouvelle erreur *Err* est calculée et, après cela, les poids W2 du neurone de règle r_j sont ajustés. Il s'agit d'un ajustement de poids plus précis que l'ajustement dans EFuNN-s/u ce qui peut faire une différence dans l'apprentissage de séquences courtes, mais pour apprendre des séquences plus longues, il ne peut y avoir aucune différence dans les résultats obtenus par l'EFuNN-s/u plus simple et plus rapide.

L'algorithme d'apprentissage EFuNN-gd

Cet algorithme est différent de l'EFuNN-s/u dans la façon dont les connexions W1 sont ajustées, ce qui n'est plus non supervisé, mais ici, un algorithme de descente de gradient en une passe est utilisé :

$$W1(r_j^{(t+1)}) = W1(r_j^{(t)}) + lj.(x_f - W1(r_j^{(t)}))(y_f - A2)A1(r_j^{(t)})\ W2(r_j^{(t)})$$

Cette formule doit être étendue lorsque le mode m-of-n est appliqué. L'algorithme EFuNN-gd n'est plus supervisé/non supervisé et les neurones de règles ne sont plus attribués aux centres de cluster de l'espace d'entrée.

Règles d'apprentissage passif (Sleep-Learning) EFuNN

Dans un autre mode, l'apprentissage passif ou sleep learning, l'apprentissage est effectué lorsqu'il n'y a pas de modèle d'entrée présenté. Cela peut être nécessaire d'appliquer une fois qu'un apprentissage initial a été effectué. Dans ce cas, les connexions existantes qui stockent les modèles d'entrée précédemment alimentés sont utilisées comme un «écho» pour réitérer le processus d'apprentissage. Ce type d'apprentissage peut être appliqué dans le cas d'une courte présentation initiale des données, lorsque seulement une petite partie des données est apprise en mode adaptatif incrémental à une passe, puis l'apprentissage est raffinée par la méthode d'apprentissage passif lorsque le système consolide ce qu'il a appris auparavant.

Apprentissage Une-Passe Versus Multiple-Passes

La meilleure façon d'appliquer les algorithmes d'apprentissage ci-dessus est de tirer des exemples de manière aléatoire dans l'espace du problème, de les propager à travers le EFuNN et d'ajuster les poids de connexion et les neurones de règles, de modifier et d'optimiser les valeurs des paramètres, etc., jusqu'à ce que l'erreur devienne souhaitablement petite. Dans un mode d'apprentissage rapide, chaque exemple est présenté une seule fois au système. S'il est possible de présenter les exemples deux ou plusieurs fois, l'erreur peut devenir plus petite, mais cela dépend des valeurs des paramètres du EFuNN et des caractéristiques statistiques des données.

2.8.11.3. Avantages et difficultés de EFuNN

- Procédure d'apprentissage très rapide grâce au réglage local d'élément, un seul neurone de règle (ou m, dans le mode m-of-n) sera soit mis à jour ou créé pour chaque exemple de données.
- L'apprentissage d'un nouvel exemple de données ne cause pas l'oubli des anciens (stabilité).
- Des variables de nouvelles entrées et de nouvelles sorties peuvent être ajoutées pendant le processus d'apprentissage, ce qui rend le système EFuNN plus flexible pour accueillir de nouvelles informations, une fois qu'elles sont disponibles, sans tenir compte des informations déjà apprises (plasticité).
- Traitement des valeurs manquantes grâce à l'utilisation de fonctions d'appartenance et de degrés d'appartenance, ainsi que l'utilisation de la différence floue locale normalisée.

Malgré les avantages de EFuNN, il existe des difficultés lors de leur utilisation:

- Les EFuNNs sont sensibles à l'ordre dans lequel les données sont présentées et aux valeurs initiales des paramètres.
- Il existe plusieurs paramètres qui doivent être optimisés dans un mode adaptatif incrémental. Ces paramètres sont: le seuil d'erreur Err; Le nombre, la forme et le type des fonctions

d’appartenance; le type d'apprentissage; le seuil d'agrégation et le nombre d'itérations avant l'agrégation,

2.9. Conclusion

Nous avons abordé, au début de ce chapitre, la relation dilemmatique entre la stabilité et la plasticité qui nécessite d’apprendre des nouvelles informations en retenant autant que possible les connaissances précédentes.

Nous avons présenté ensuite le principe, les caractéristiques ainsi que les types de l’apprentissage incrémental.

Nous avons consacré la suite du chapitre à présenter les principales méthodes et algorithmes d’apprentissage incrémental dont l’utilisation a une grande influence sur la performance des systèmes lorsque les ensembles de données d’entraînement deviennent disponibles pendant une longue période de temps.

CHAPITRE 3

SYSTEME EVOLUTIF POUR LA CLASSIFICATION ADAPTATIVE INCREMENTALE D'IMAGES

L'objectif de ce chapitre est de présenter le système proposé qui est un système de classification d'images capable de s'adapter de manière incrémentale aux changements de l'environnement. L'approche suggérée consiste à apprendre des règles de classification par un réseau de neurones flou évolutif, puis optimiser ces règles par les algorithmes génétiques.

3.1. Introduction

La classification d'images est une tâche importante de la recherche d'information qui pourrait être considérée comme un outil d'aide pour différentes tâches d'accès à l'information. C'est un problème qui intéresse les chercheurs depuis relativement longtemps, et même s'il est certain que les applications dans ce domaine sont assez nombreuses et que des avancées importantes ont été observées depuis, les caractéristiques de l'environnement dans lequel les systèmes doivent fonctionner diminuent grandement leur efficacité. Ces caractéristiques concernent le changement rapide causé par la croissance incontestable de nombre d'images numériques sous différents formats (bmp, JPEG, GIF, PNG, …), l'incertitude, l'imprécision, le vague, …etc. Ceci exige des systèmes qui peuvent suivre les changements de l'environnement.

La tâche d'apprentissage incrémental est prometteuse parce qu'elle permet de doter les systèmes de la capacité d'adapter automatiquement leur structure et fonctionnalité pour suivre les changements de l'environnement causés par l'arrivée de nouvelles données introduisant des nouveaux concepts ou des nouvelles classes.

Ce chapitre propose une approche pour améliorer la capacité d'un classificateur à bien accomplir sa tâche dans des situations où un entraînement sur un nombre suffisant d'images n'aura pas été possible. L'objectif final de notre travail est de concevoir et implémenter un système intelligent évolutif pour la classification adaptative incrémentale d'images. Ce chapitre présente la démarche à suivre pour atteindre cet objectif.

3.2. Base d'images utilisée

La base utilisée est la base de segmentation d'images « Image Segmentation » de l'UCI prise du groupe de vision de l'université du Massachusetts en 1990 avec contributions de Carla Brodley et disponible en ligne sur le site de l'UCI : http://archive.ics.uci.edu/ml/datasets.html. Cette base est composée des caractéristiques de 2310 images provenant de 7 classes (BRICKFACE, SKY, FOLIAGE, CEMENT, WINDOW, PATH, GRASS). Ces données sont représentées par 19 attributs: region-centroid-col, region-centroid-row, region-pixel-count, short-line-density-5, short-line-density-2, vedge-mean, vegde-sd, hedge-mean, hedge-sd, intensity-mean, rawred-mean,

rawblue-mean, rawgreen-mean, exred-mean, exblue-mean, exgreen-mean, value-mean, saturation-mean, hue-mean. Les instances ont été tirées aléatoirement à partir d'une base de données de 7 images plein air.

Nous avons réparti ces données sur quatre bases, trois servent pour l'apprentissage et une pour le test dont la répartition en classes est donnée par le tableau 3.1. Dans chaque base d'apprentissage, seules cinq des sept classes sont disponibles et la base de test est la seule qui contient les sept classes. A chaque nouvelle base, le système doit prendre en compte l'apparition d'une nouvelle classe. Ce partitionnement a été fait ainsi pour démontrer la capacité du système proposé à apprendre de nouvelles classes qui apparaissent à chaque nouvelle base de données disponible.

Base	DS1	DS2	DS3	Test	Total
Brickface	82	82	83	83	330
Path	83	83	82	82	330
Window	82	83	83	82	330
Grass	83	82	82	83	330
Foliage	165	0	0	165	330
Sky	0	165	0	165	330
Cement	0	0	165	165	330
Total	495	495	495	825	2310

Tableau 3.1 : Répartition de données utilisées

3.3. Système de classification d'images proposé

Notre objectif est de concevoir un système évolutif de classification d'images qui effectue un apprentissage adaptatif incrémental des règles de classification par un réseau de neurones flou évolutif EFuNN (section 2.8.11) [Kas 01b], puis optimise ces règles par les algorithmes génétiques.

L'idée est de disposer d'un ensemble d'images préalablement étiquetés à partir duquel nous extrairons des règles de classification.

A partir des caractéristiques de chaque image, on construit un vecteur caractéristique à longueur fixe dont les composantes sont les valeurs des descripteurs de l'image de sorte à avoir dans la même position la valeur du même descripteur, c'est le vecteur descripteur de cette image.

Les données contenues dans la base utilisée sont représentées par des vecteurs composés de 19 attributs. Cette base est composée de données représentant 7 classes (brickface, sky, foliage, cement, window, path, grass).

La structure du système se construit avec l'arrivée de paquets de données au fur et à mesure de l'apprentissage incrémental.

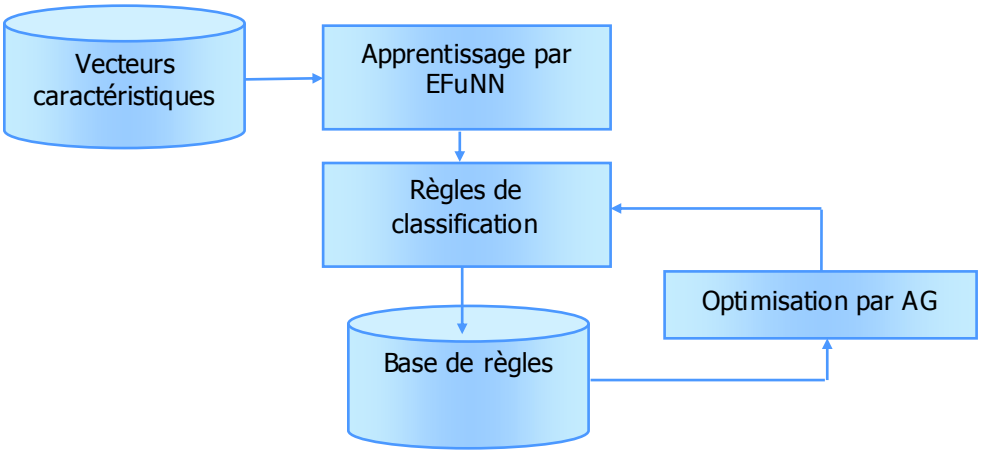


Figure 3.1 : Architecture du système proposé

3.3.1 Choix de la méthode d'apprentissage

Parmi les méthodes analysées ce sont les réseaux de neurones flous évolutifs EFuNNs qui répondent le mieux aux besoins nécessaires pour atteindre nos objectifs. EFuNN est un modèle du soft computing qui intègre les réseaux de neurones et la logique floue pour effectuer un apprentissage adaptatif incrémental en ligne et apprendre des règles de classification floues.

Les EFuNNs fonctionnent dans un espace ouvert c.à.d. le nombre d'éléments et de dimensions peut augmenter. Ils n'ont pas besoin de voir une donnée plusieurs fois pour bien apprendre (ils peuvent fonctionner en temps réel). Ils font de l'apprentissage constructif et ont une structure dynamique. Ils apprennent en mode «life-long». Ils ont une architecture qui permet d'ajouter ou d'extraire rapidement et facilement des règles floues compréhensibles pour un être humain.

La méthode utilisée dans ce mémoire pour effectuer l'apprentissage adaptatif incrémental sera le modèle EFuNN.

3.3.2 Processus d'apprentissage par EFuNN

Les valeurs suivantes pour les paramètres de EFuNN ont été définies : la valeur initiale pour le seuil de sensibilité initiale S = 0,9; le nombre de fonctions d'appartenance = 3, on utilise des fonctions d'appartenance triangulaires; le seuil d'erreur E = 0,1; un rayon maximal Rmax = 0,2; le paramètre d'agrégation N_{agg} = 3; lr1 = lr2 = 0,1; les seuils pour l'extraction des règles T1 = 0,5 et T2 = 0,5.

Les exemples *d = (x, y)* sont composés d'un vecteur d'entrées *x* et d'un vecteur de sorties *y* correspondant.

Le nombre de neurones par couche change constamment pendant l'apprentissage. Dans notre application, une fois les neurones de la première, la deuxième, la quatrième et la cinquième couche sont crée leur nombre reste inchangée parce que nous avons utilisé des vecteurs d'entrée

de longueur fixe (19 caractéristiques) et un nombre fixe de fonctions d'appartenance (trois fonctions). La troisième couche doit créer de nouveaux neurones constamment pour apprendre de nouvelles règles.

EFuNN commence son apprentissage sans aucun neurone.

Lorsque le premier exemple (x, y) est présenté au réseau, les neurones de la première couche sont créés (19 neurones représentant les 19 valeurs caractéristiques du vecteur d'entrée).

Le nombre exact de connexions de la première couche vers la deuxième couche dépend du nombre de fonctions floues d'entrées qui est choisi dans cette application égale à trois c.à.d. pour chaque neurone de la première couche, trois neurones sont créés dans la deuxième couche, ces neurones représentent les valeurs d'appartenance floues « Bas », « Moyen » et « Grand » (voir Figure 3.2). Le nombre de connexion sera donc 19 × 3 = 57. Chaque neurone de la deuxième couche est relié à un seul neurone de la première couche. La sortie d'un neurone de la deuxième couche est donc dépendante de la sortie d'un seul neurone dans la première couche.

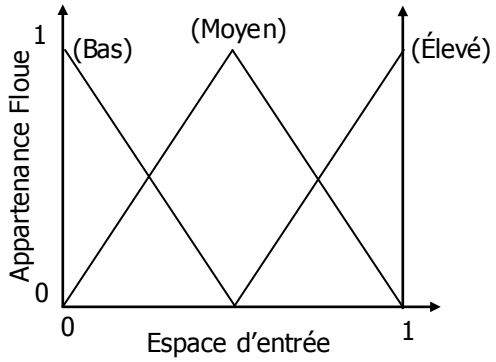


Figure 3.2 : Ensemble des trois fonctions d'appartenance floue ainsi que leur correspondance linguistique.

Les neurones de la cinquième couche sont créés (19 neurones représentant les 19 valeurs caractéristiques du vecteur de sortie).

Pour chaque neurone de la cinquième couche, trois neurones dans la quatrième couche sont créés, ces neurones représentent les valeurs d'appartenance floues « Bas », « Moyen » et « Grand ». Le nombre de connexions sera donc 57. Chaque neurone de la quatrième couche est relié à un seul neurone de la cinquième couche.

Un premier neurone est créé dans la troisième couche pour stocker l'état des neurones de la deuxième couche $W1(r_j)$ et l'état des neurones de la quatrième couche $W2(r_j)$ pour le premier exemple comme étant une règle r_j. Chaque neurone de la troisième couche est relié avec tous les neurones de la deuxième couche et avec tous les neurones de la quatrième couche.

Les connaissances d'un EFuNN sont toutes gardées dans la troisième couche sous la forme de règles conditionnelles floues, voici un exemple d'une règle floue :

> SI entrée1 est basse ET entrée2 est moyenne ET entrée3 est basse ET entrée4 est basse ET entrée5 est moyenne ET entrée6 est élevée ET entrée7 est moyenne ET entrée8 est élevée ET entrée9 est basse ET entrée10 est moyenne ET entrée11 EST élevée ET entrée12 est moyenne ET entrée13 est élevée ET entrée14 est basse ET entrée15 est moyenne ET entrée16 est moyenne ET entrée17 est moyenne ET entrée18 est élevée ET entrée19 est basse ALORS sortie est PATH.

Pour chaque présentation d'une paire entrées-sorties (x, y)

Évaluer la distance floue normalisée locale D entre x_f (f signifie flou) et les connexions de neurone de règle existantes W1 :

$$D(x_f, W1) = || x_f - W1 || / || x_f + W1 ||$$

Calculer l'activation A1 de la couche de neurone de règle. Trouver le neurone de règle le plus proche r_k au vecteur d'entrée flou x_f :

$$A1(r_k^{(t)}) = 1-(D(W1(r_k^{(t)}), x_f));$$

Si $A1(r_k) < S_k$; Créer un nouveau neurone de règle pour (x_f, y_f)

sinon

Trouver l'activation de la couche de sortie floue A2 = W2A1 et l'erreur de sortie Err = ||y-y'|| / N_{out}.

si Err >E

Créer un nouveau neurone de règle pour accommoder l'exemple actuel (x_f, y_f)

sinon

Mettre à jour $W1(r_k)$ et $W2(r_k)$:

$$W1(r_k(t+1)) = W1(r_k^{(t)}) + l_k.(x_f - W1(r_k^{(t)}))$$

$$W2(r_k(t+1)) = W2(r_k^{(t)}) + l_k.(y_f - A2).A1(r_k^{(t)})$$

$l_k = 1/Nex(r_k)$, où Nex (r_k) est le nombre d'exemples actuellement associés au neurone de règle r_k.

Appliquer *la procédure d'agrégation* des neurones de règles après chaque groupe de N_{agg} exemples (N_{agg}=3):

$$W1(r_{agg}) = (W1(r1)+W1(r2)+W1(r3))/3$$

$$W2(r_{agg}) = (W2(r1).Nex(r1)+W2(r2).Nex(r2)+W2(r3).Nex(r3))/Nsum$$

où

$$Nex(r_{agg}) = Nsum = Nex(r1)+Nex(r2)+Nex(r3)$$

Si $Rr_{agg} = D(W1(r_{agg}))W1(r_j))+R_j <= Rmax$, alors les trois neurones de règle agrégeront.

r_j est le neurone de règle à partir des trois neurones qui ont une distance maximale du nouveau neurone r_{agg} et R_j est son rayon du champ réceptif.

Mettre à jour les paramètres S_k, R_k, Age(r_k), TA(r_k) pour le neurone de règle r_k :

$$Sk^{(t+1)} = Sk^{(t)} - D(W1(rk^{(t+1)}), W1(rk^{(t)}))$$

$$Rk^{(t+1)} = Rk^{(t)} + D(W1(rk^{(t+1)}), W1(rk^{(t)}))$$

Elaguer des neurones de règles si nécessaire, tel que défini par les paramètres d'élagage.

SI (Age(rj) > OLD) ET (l'activation totale TA(rj) est inférieure à un paramètre d'élagage Pr fois Age (rj)) ALORS élaguer le neurone de règle rj,

Extraire les règles à partir des neurones de règles.

Chaque neurone de règle r_j peut être exprimé comme une règle floue.

3.3.3. Architecture générale du EFuNN obtenu

L'EFuNN obtenu est un réseau de neurones à cinq couches : une couche des entrées, une couche des entrée floues, une couche de règles (pour le stockage des règles), une couche des sorties floues et une couche des sorties (Figure 3.3).

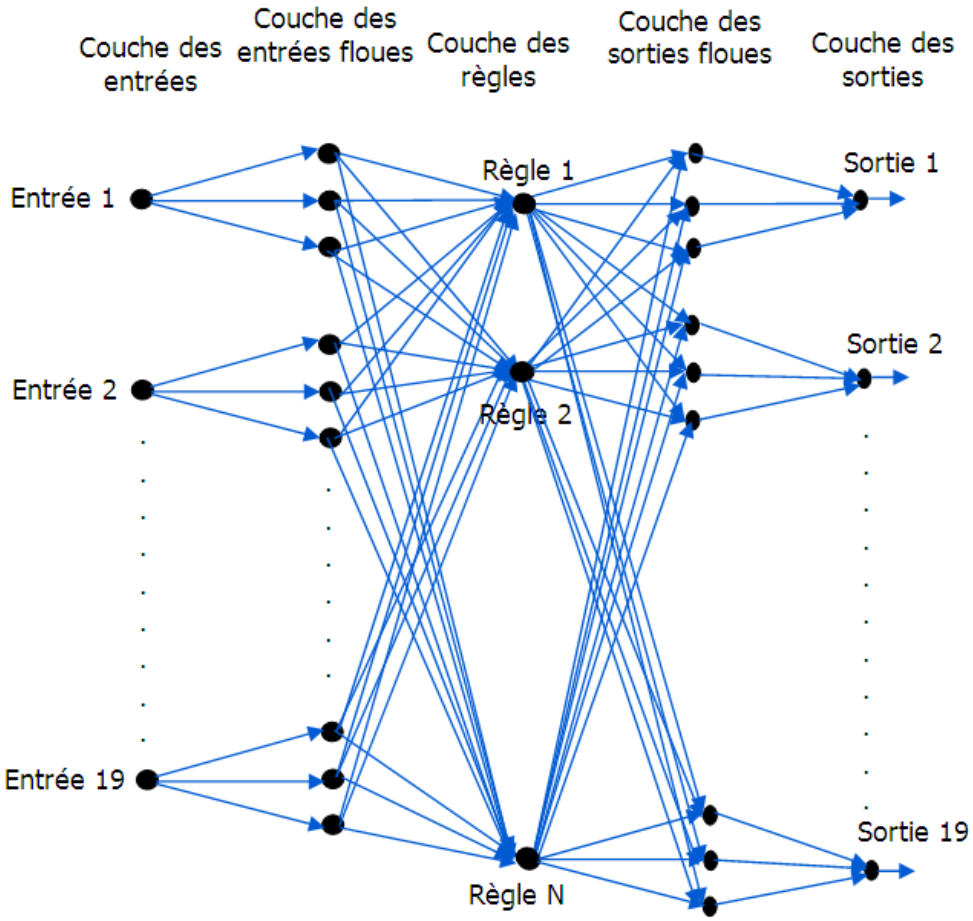


Figure 3.3 : Architecture du réseau de neurone flou évolutif EFuNN pour la classification d'images de la base« Image Segmentation » de l'UCI.

3.3.4 Optimisation de la base de règles par les algorithmes génétiques

Chaque neurone de la troisième couche de EFuNN représente une règle floue. L'ensemble des règles extraites de cette couche représente la base de règle obtenue par l'apprentissage.

Nous proposons l'optimisation de ces règles par les algorithmes génétiques comme suit :

Les règles de la base de règles sont codées dans des chromosomes séparés. Chaque chromosome représente une règle codée. L'ensemble de tous les chromosomes de la population représente la base de règles.

Chaque chromosome dans la population représente une seule règle et seulement le meilleur individu de la population est choisi. La base de règles complète peut être obtenue en répétant l'algorithme génétique plusieurs fois, ajoutant à chaque fois la meilleure règle à la base de règles

jusqu'à ce que la solution entière soit trouvée. L'algorithme d'optimisation de la base de règles par algorithme génétique basé sur l'apprentissage itératif de règle peut être présenté comme suit :

1. Appliquer l'algorithme génétique pour trouver une règle. On prend comme fitness du chromosome codant une règle le nombre de données d'apprentissage classées correctement.
2. Attacher la règle à l'ensemble final de règles.
3. Évaluer l'efficacité de l'ensemble des règles, particulièrement en prenant en considération la règle trouvée récemment.
4. Si l'ensemble des règles générées est satisfaisant pour le problème résolu, finir le fonctionnement de cet algorithme. Sinon, passer au point 1.

Dans l'optimisation des règles par l'apprentissage itératif de règles, des règles particulières se développent indépendamment les unes des autres sans aucune information sur celles générées précédemment. Il peut en résulter dans la répétition de règles ou de leur exclusion mutuelle. Par conséquent, après que le fonctionnement de l'algorithme est terminé, des procédures de simplification de la base de règle sont appliquées. Cependant, déjà lorsque l'algorithme est en fonctionnement, nous pouvons empêcher des règles identiques d'apparaître. La méthode est basée sur l'enlèvement de ces données de la séquence d'apprentissage qu'elles ont utilisé pour le but de l'apprentissage correct des règles trouvées préalablement.

3.4. Evaluation de la qualité du classifieur

Pour tester le système proposé, nous avons utilisé l'ensemble Test (voir le tableau Tab. 3.1).

On va appliquer une mesure d'évaluation sur l'ensemble d'images de test afin de s'assurer que le modèle est généralisable à d'autres images et pour mesurer la performance du classifieur.

La précision π_i pour la classe C_i est définie comme la probabilité conditionnelle qu'un exemple choisi aléatoirement dans la classe soit bien classé par le système.

Le rappel ρ mesure la largeur de l'apprentissage et correspond à la fraction des exemples pertinents, parmi ceux proposés par le classifieur.

Précision : $\pi_i = a_i / (a_i + b_i)$, soit le nombre d'assignations correctes sur le nombre total d'assignations.

Rappel : $\rho_i = a_i / (a_i + c_i)$, soit le nombre d'assignations correctes sur le nombre d'assignations qui auraient dû être faites.

Où

a_i: le nombre d'exemples correctement classés comme appartenant à la classe C_i.

b_i: le nombre d'exemples incorrectement classés comme appartenant à la classe C_i.

c_i: le nombre d'exemples incorrectement rejetés de la classe C_i.

d_i: le nombre d'exemples correctement rejetés de la classe C_i.

La précision et le rappel globaux, notés respectivement π et ρ, peuvent être calculés à travers une moyenne des résultats obtenus pour chaque classe. Le score global du système est calculé par la mesure F_β qui prend en compte les deux valeurs π et ρ. Elle est donnée par :

$$F_\beta = (\beta^2+1)\pi\rho / \beta^2\rho+\pi$$

Le paramètre β prend la valeur 1 dans notre évaluation.

3.5. Caractéristiques du système proposé

Le système proposé a les caractéristiques suivantes :

1. Il apprend des règles de classification floues de façon incrémentale.
2. Il peut expliquer à tout moment de son fonctionnement l'essence et la «connaissance» qu'il a apprit par un ensemble de règles floues.
3. Il fait de l'apprentissage constructif (augmentation et élagage de réseau) et a une structure dynamique.
4. Il n'a pas besoin de voir une donnée plusieurs fois pour bien apprendre.
5. L'approche proposée offre le traitement de l'incertitude, le vague, l'imprécision et la vérité partielle et permet l'adaptation aux changements de l'environnement.
6. La méthode proposée répond bien au dilemme « stabilité-plasticité ».
7. L'approche proposée est une méthode d'apprentissage incrémental car elle répond aux 4 critères :
    - La plasticité du système est garantie par la création incrémentale de règles de classification.
    - La stabilité du système est garantie par le stockage des règles apprises au niveau des neurones de la troisième couche.
    - Elle peut s'adapter aux nouvelles classes qui peuvent être introduites avec les nouvelles données : l'apprentissage de nouvelles classes est effectué par l'ajout de neurones dans la troisième couche.
    - Elle ne nécessite pas l'accès aux données d'origine utilisées pour apprendre le classifieur existant : la connaissance est stockée sous forme de règles.

3.6. Avantages du système proposé

- Les résultats de classification sont explicables (des règles floues).
- Procédure d'apprentissage très rapide grâce au réglage local d'élément, un seul neurone de règle sera soit mis à jour ou créé pour chaque exemple de données (surtout dans le cas où des fonctions d'activation linéaire sont utilisées).
- L'apprentissage d'un nouvel exemple de données ne cause pas l'oubli des anciens.
- Des variables de nouvelles entrées et de nouvelles sorties peuvent être ajoutées pendant le processus d'apprentissage, ce qui rend le système EFuNN plus flexible pour accueillir de nouvelles informations, une fois qu'elles sont disponibles, sans tenir compte des informations déjà apprises.
- Traitement des valeurs manquantes grâce à l'utilisation de fonctions d'appartenance et de degrés d'appartenance, ainsi que l'utilisation de la différence floue locale normalisée.

3.7. Conclusion

Nous avons présenté dans ce chapitre notre approche pour la classification adaptative incrémentale d'images basée sur un système neuro-flou évolutif. La base d'images utilisée pour l'évaluation de ce système est la base « Image Segmentation » de l'UCI Machine Learning Repository.

CONCLUSION

Le travail de recherche présenté dans ce document se situe dans le cadre général de l'apprentissage automatique. Il se concentre principalement sur les problèmes posés au niveau des systèmes évolutifs qui sont censés avoir la capacité d'intégrer (dans un système déjà entraîné) de nouvelles connaissances telles que de nouvelles données d'apprentissage ou de nouvelles classes.

L'adaptabilité ou la capacité d'évolution représente l'une des limitations les plus fondamentales des techniques d'apprentissage qui sont actuellement relativement efficaces dans le cas statique. De ce fait, les méthodes classiques sont souvent inefficaces pour répondre aux nouveaux besoins des applications actuelles où des flux continus de gros volumes de données sont disponibles.

L'apprentissage adaptatif ou incrémental concerne des problèmes complexes, dynamiques et évolutifs, avec des données de natures et d'origines différentes, hétérogènes et bruitées. Ce problème représente l'une des préoccupations majeures de la communauté de l'apprentissage automatique et constitue un champ de recherche ouvert qui a fait l'objet plusieurs types de travaux.

Nous avons envisagé d'appréhender cette problématique en étudiant l'apport de l'hybridation de paradigmes sur la capacité d'adaptabilité et d'évolution de l'apprentissage. Notre intérêt se focalise principalement sur les méthodes connexionnistes, les algorithmes génétiques et la logique floue. Nous nous sommes concentrés sur l'étude de leurs possibilités d'hybridation dans le cadre de la conception de systèmes intelligents évolutifs.

Notre objectif de départ était de concevoir un système à apprentissage hybride incrémental qui peut s'adapter à un environnement dynamique. Pour atteindre cet objectif, nous sommes d'abord intéressés à l'étude des systèmes intelligents rassemblés sous la dénomination du "soft computing" intégrant les réseaux de neurones, la logique floue et les algorithmes évolutionnaires. Nous avons étudié plusieurs intégrations possibles entre ces méthodes, ce qui nous a permis de constater la richesse des approches hybrides du soft-computing puis d'envisager l'utilisation de l'une de ces méthodes dans le cadre de notre travail.

Nous nous sommes ensuite concentrés sur la notion d'apprentissage adaptatif incrémental qui fait référence au processus d'accumulation et de gestion de connaissances dans le temps. Il est très approprié pour les tâches d'apprentissage dans lesquelles les ensembles de données d'entraînement deviennent disponibles pendant une longue période de temps. Nous présentons le principe, les caractéristiques ainsi que les types de l'apprentissage incrémental.

Dans le cadre de notre travail, nous proposons un système hybride conçu en utilisant une intégration d'approches issues du soft computing dans le domaine applicatif de la classification d'images. L'approche proposée pour la classification adaptative incrémentale d'images est basée sur un système neuro-flou évolutif. La base d'images utilisée pour l'évaluation de ce système est la base « Image Segmentation » de l'UCI Machine Learning Repository.

Il s'agit d'un système de classification d'images capable de s'adapter de manière incrémentale aux changements de l'environnement. L'approche suggérée consiste à apprendre des règles de classification par un réseau de neurones flou, puis optimiser ces règles par les algorithmes génétiques.

BIBLIOGRAPHIE

[ALM 08] Abdullah Almaksour, Harold Mouchère, Eric Anquetil, "Apprentissage incrémental et synthèse de données pour la reconnaissance de caractères manuscrits en-ligne", Actes du 8ème Colloque International Francophone sur l'Ecrit et le Document (CIFED'08), vol. 1, pp. 55 – 60, Rouan, 2008.

[BAL 94] Shumeet Baluja, "Population Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning", Carnegie Mellon University, technical report CMU-CS-94-163, Computer Science Department, Carnegie Mellon University, 1994.

[BAR 98] Andrea Baraldi, Ethem Alpaydin, "Simplified ART: A new class of ART algorithms", Technical Report TR-98-004, International Computer Science Institute, Berkeley, CA, 1998.

[BON 01] Piero P. Bonissone, Kai Goebel, "Soft Computing Applications in Equipment Maintenance and Service", Proceedings of IFSA/NAFIPS, vol. 5, pp. 2752 – 2757, 2001.

[BOU 07] Abdelhamid Bouchachia, Bogdan Gabrys, Zoheir Sahel, "Overview of some incremental learning algorithms", IEEE international conference on fuzzy systems, vol. 23 – 26, pp. 1 – 6, 2007.

[BOU 09] Khaled Boukharouba, Laurent Bako, Stéphane Lecoeuche, "Incremental and decremental multi-category classification by support vector machines", International Conference on Machine Learning and Applications ICMLA'09, IEEE, pp. 294 –300, 2009.

[CAR 87] Gail A. Carpenter, Stephen Grossberg, "ART 2: self–organization of stable category recognition codes for analog input patterns", Applied Optics, vol. 26, no. 23, pp. 4919 – 4930, 1987.

[CAR 90] Gail A. Carpenter, Stephen Grossberg, "ART 3: Hierarchical Search Using Chemical Transmitters in Self–Organizing Pattern Recognition Architectures", Neural Networks, vol. 3, pp. 129 – 152, 1990.

[CAR 91a] Gail A. Carpenter, Stephen Grossberg, John H. Reynolds, "ARTMAP: Supervised Real–Time Learning and Classification of Nonstationary Data by a Self–Organizing Neural Network", Neural Networks, vol. 4, no. 5, pp. 565 – 588, 1991.

[CAR 91b] Gail A. Carpenter, Stephen Grossberg, John H. Reynolds, "Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System", Neural Networks, vol. 4, no. 6, pp. 759 – 771. 1991.

[CAR 92] Gail A. Carpenter, Stephen Grossberg, Natalya Markuzon, John H. Reynolds, David B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps", IEEE Transactions on Neural Networks, vol. 3, no. 5, pp. 698 – 13, 1992.

[CHA 02] P.T. Chan, A.B. Rad, "Adaptation and learning of a fuzzy system by nearest neighbor clustering", Fuzzy Sets and Systems, vol. 126, no. 3, pp. 353 – 366, 2002.

[CHA 04] Zeke S. H. Chan, Nikola Kasabov, "Gene Trajectory Clustering with a Hybrid Genetic Algorithm and Expectation Maximization Method", Proceedings of International Joint Conference on Neural Networks (IJCNN 2004), IEEE Press, pp. 1669 – 1674, 2004.

[CHE 07] ZhiHang Chen, Liping Huang, Yi L. Murphey, "Incremental Learning for Text Document Classification", Proceedings of International Joint Conference on Neural Networks (IJCNN 2007), pp. 2592 – 2597, IEEE, 2007.

[COR 05] A. Cornuéjols, "Apprentissage et circulation d'information", Habilitation à diriger les recherches, Université de Paris-Sud, France, 2005.

[DOT 01] Yasuhiko Dote, Seppo J. Ovaska, "Industrial applications of soft computing: A Review", Proceedings of IEEE, vol.89, no. 9, pp. 1243 – 1265, 2001.

[ENE 05] Fabricio Enembreck, Jean–Paul Barthes, "ELA—A new Approach for Learning Agents", Autonomous Agents and Multi-Agent Systems, vol. 10, no. 3, pp. 215 – 248, 2005.

[FRI 94] B. Fritzke, "Fast learning with incremental RBF networks", Neural Processing Letters, vol.1, no.1, pp. 2 – 5, 1994.

[FRI 95] B. Fritzke, "A growing neural gas network learns topologies", Advances in Neural Information Processing Systems, vol. 7, pp. 625 – 632, 1995.

[GRA 08] Eric Granger, Jean–François Connolly, Robert Sabourin, "A Comparison of Fuzzy ARTMAP and Gaussian ARTMAP Neural Networks for Incremental Learning", IEEE International Joint Conference on Neural Networks (IJCNN 2008), pp. 3305 – 3312, 2008.

[GRO 75] Stephen Grossberg, "A neural model of attention, reinforcement, and discrimination learning", International Review of Neurobiology, vol. 18, pp. 263 – 327, 1975.

[GRO 76a] Stephen Grossberg, "Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors", Biological cybernetics, vol. 23, no. 3, pp 121 – 134, 1976.

[GRO 76b] Stephen Grossberg, "Adaptive pattern classification and universal recoding: II. Feedback, expectation, olfaction, illusions", Biological Cybernetics, vol. 23, no. 4, pp 187 –202, 1976.

[GRO 87] Stephen Grossberg, "Competitive Learning: From Interactive Activation to Adaptive Resonance", Cognitive science, vol. 11, no. 1, pp. 23 – 63, 1987.

[GRO 88] Stephen Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures", Neural Networks, vol. 1, pp. 17 – 61, 1988.

[GRO 91] Stephen Grossberg, David Somers, "Synchronized oscillations during cooperative feature linking in a cortical model of visual perception", Neural Networks, vol. 4, pp. 453 – 466, 1991.

[GUA 05] Sheng–Uei Guan, Fangming Zhu, "An Incremental Approach to Genetic–Algorithms–Based Classification", IEEE Transactions on Systems, Man, and Cybernetics– Part B: Cybernetics, vol. 35, no. 2, pp. 227 – 239, 2005.

[GUD 97] Ricardo Gudwin, Fernando Gomide, "A Computational Semiotics Approach for Soft Computing", IEEE International Conference on Systems, Man and Cybernetics, vol. 4, pp. 3981 – 3986, 1997.

[HAM 01] Fred H. Hamker, "Life–long learning Cell Structures—continuously learning without catastrophic interference", IEEE Transactions on Neural Networks, vol. 14, no. 4 – 5, pp. 551 – 573, 2001.

[HEB 99a] Jean-François Hébert, Marc Parizeau, Nadia Ghazzali, "An Hybrid Architecture for Active and Incremental Learning: The Self-Oganizing Perceptron (SOP) Network", Proceedings of the International Joint Conference on Neural Networks (IJCNN99), pp. 1646 – 1651, 1999.

[HEB 99b] Jean-François Hébert, Marc Parizeau, Nadia Ghazzali, "Cursive Characters Detection using Incremental Learning", Proceedings of the 5th International Conference on Document Analysis and Recognition (ICDAR), pp. 808 – 811, 1999.

[HEB 99c] Jean-François Hébert, Marc Parizeau, Nadia Ghazzali, "Learning to Segment Cursive Words using Isolated Characters", Proceedings of the 12th Conference on Vision Interface (VI), pp. 33 – 40, 1999.

[HEB 00] Jean-François Hébert, "Architecture neuronale Hybride pour l'apprentissage incrémental des connaissances : Application à la reconnaissance d'écriture cursive", PhD thesis, Université de Laval, Québec, Canada, Mars 2000.

[JAE 99] Jaesoo Kim, Nikola Kasabov, "HyFIS: adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems", Neural Networks vol. 12, no. 9, pp. 1301 – 1319, 1999.

[JAI 06] Sanjay Jain, Steffen Lange, Sandra Zilles, "Towards a better understanding of incremental learning", Proceedings of the 17th International Conference on Algorithmic Learning Theory (ALT'06), Lecture Notes in Artificial Intelligence, vol. 4264, pp. 169 – 183, 2006.

[KAM 07] Youki Kamiya, Toshiaki Ishii, Osamu Hasegawa, "Life–long Semi–supervised Learning: Continuation of Both Learning and Recognition", IEEE Symposium on Computational Intelligence in Image and Signal Processing, CIISP, pp. 403 – 408, 2007.

[KAR 04] Fakhreddine O. Karray, Clarence W De Silva, "Soft Computing and Intelligent Systems Design: Theory, Tools and Applications", Pearson Education, 2004.

[KAS 99a] Nikola Kasabov, Steven I. Israel, Brendon J. Woodford, "Adaptive, Evolving, Hybrid Connectionist Systems for Image Pattern Recognition", Soft computing for image processing, Studies in Fuzziness and Soft Computing, vol. 42, pp. 318 – 336, 2000.

[KAS 99b] Nikola Kasabov, "Evolving Connectionist Systems for On–line, Knowledge–based Learning: Principles and Applications", Information Science, Discussion Papers Series, No. 99/02, 1999.

[KAS 99c] Nikola Kasabov, Brendon Woodford, "Rule Insertion and Rule Extraction from Evolving Fuzzy Neural Networks: Algorithms and Applications for Building Adaptive, Intelligent Expert Systems", IEEE International Fuzzy Systems Conference Proceedings, pp. 1406 – 1411, 1999.

[KAS 01a] Nikola Kasabov, "Evolving Fuzzy Neural Networks for Supervised/Unsupervised On–Line, Knowledge–Based Learning", IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, vol. 31, no. 6, pp. 902 – 918, 2001.

[KAS 01b] Nikola Kasabov, "Evolving Fuzzy Neural Networks for On–line Knowledge Discovery", The Information Science Discussion Paper Series, 2001.

[KAS 01c] Nikola Kasabov, "On–line learning, reasoning, rule extraction and aggregation in locally optimized evolving fuzzy neural networks", Neurocomputing, vol 41, no. 1 – 4, pp. 25 – 45, 2001.

[KAS 02a] Nikola Kasabov, Qun Song, "DENFIS: Dynamic Evolving Neural–Fuzzy Inference System and Its Application for Time–Series Prediction", IEEE Transactions on Fuzzy Systems, vol. 10, no. 2, pp. 144 – 154, 2002.

[KAS 02b] Nikola Kasabov, "Evolving Connectionist Systems for Adaptive Learning and Knowledge Discovery: Methods, Tools, Applications", Proceedings of the First International IEEE Symposium on Intelligent Systems, vol.1, pp. 24 – 28, 2002.

[KAS 02c] Nikola Kasabov, "Evolving Connectionist Systems for Adaptive Learning and Knowledge Discovery: Methods, Tools, Applications", Proceedings of the 9th International Conference on Neural Information Processing ICONIP'02, vol.2, pp. 590 – 595, 2002.

[KAS 05] Nikola Kasabov, David Zhang, Paul S. Fang, "Incremental Learning in Autonomous Systems: Evolving Connectionist Systems for On–line Image and Speech Recognition", Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts, pp. 120 – 125, 2005.

[KAS 06] Nikola Kasabov, "Adaptation and interaction in dynamical systems: Modelling and rule discovery through evolving connectionist systems", Applied Soft Computing, vol. 6, no. 3, pp. 307 – 322, 2006.

[KAS 07] Nikola Kasabov, "Evolving Connectionist Systems", The Knowledge Engineering Approach (2nd edition), 2007.

[KIM 99] J. Kim, N. Kasabov, "HyFIS: adaptive neuro–fuzzy inference systems and their application to nonlinear dynamical systems", Neural Networks, vol. 12, no. 9, pp. 1301 – 1319, 1999.

[KON 00] Amit Konar, "Artificial Intelligence and Soft Computing: Behavioral and Cognitive Modeling of the Human Brain", CRC Press LLC, 2000.

[LIY 08] Tian Liye, Ben Kerong, Tu Song, Cui Lilin, "Acoustic Fault Identification of Underwater Vehicles Based on SOM/OMRBF", Proceedings of IEEE Congress on Image and Signal Processing, vol. 4, pp. 14 – 18, 2008.

[MAU 05] Nicola Di Mauro, Floriana Esposito, Stefano Ferilli, Teresa M.A. Basile, "Avoiding Order Effects in Incremental Learning", AI*IA 2005: Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 3673, pp. 110 – 121, 2005.

[MIS 06] Sigita Misina, "Incremental Learning for E-mail Classification", Proceedings of the International Conference on Computational Intelligence, Theory and Applications, vol. 38, pp. 545 – 553, 2006.

[MIT 97] Tom M. Mitchell, "Machine Learning", McGraw-Hill Science, New York, 1997.

[MUH 09] Michael D. Muhlbaier, Apostolos Topalis, Robi Polikar, "Learn++.NC: Combining Ensemble of Classifiers With Dynamically Weighted Consult–and–Vote for Efficient Incremental Learning of New Classes", IEEE Transactions on Neural Networks, vol. 20, no. 1, pp. 152 – 168, 2009.

[MUR 07] Yi L. Murphey, Zhi Hang Chen, Lee A. Feldkamp, "An incremental neural learning framework and its application to vehicle diagnostics", Applied Intelligence, vol. 28, no. 1, pp. 29 – 49, 2007.

[MUS 98] G. Muscato, "Soft Computing techniques for the control of walking robots", IEEE Computing and Control Engineering Journal, vol. 9, no. 4, pp. 193 – 200, 1998.

[NAU 00] Detlef Nauck, Rudolf Kruse, "NEFCLASS-J – A JAVA-Based Soft Computing Tool Intelligent Systems and Soft Computing", Lecture Notes in Computer Science, vol. 1804, pp. 130 – 160, 2000.

[ORS 07] Laurent Orseau, "Imitation Algorithmique : Apprentissage Incrémental En ligne de Séquences", Thèse de Doctorat, Institut National des sciences Appliquées de Rennes, Septembre 2007.

[OVA 06] Seppo J. Ovaska, Akimoto Kamiya, YangQuan Chen, "Fusion of Soft Computing and Hard Computing: Computational Structures and Characteristic Features", IEEE Transactions on Systems, Man, and Cybernetics– Part C: Applications and reviews, vol. 36, no. 3, pp. 439 – 448, 2006.

[PAR 09] Myoung Soo Park, Jin Young Choi, "Evolving Logic Networks With Real–Valued Inputs for Fast Incremental Learning", IEEE Transactions on Systems, Man, and Cybernetics– Part B: Cybernetics, vol. 39, no. 1, pp. 254 – 267, 2009.

[POL 01] Polikar R., Udpa L., Udpa S., Honavar V., "Learn++: An Incremental Learning Algorithm for Supervised Neural Networks", IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and reviews, vol. 31, no. 4, pp. 497 – 508, 2001.

[PRU 04] Yann Prudent, Abdel Ennaji, "Les K Plus Proches Classifieurs", Huitième Colloque International Francophone sur l'Ecrit et le Document, CIFED04, 2004.

[PRU 06] Yann Prudent, "Système d'apprentissage incrémental et hybride", Thèse de Doctorat, Université et INSA (Institut National des sciences Appliquées) de Rouen, Décembre 2006.

[RÜP 01] Stefan Rüping, "Incremental learning with support vector machines", Proceedings of the IEEE International Conference on Data Mining ICDM, pp. 641 – 642. 2001.

[RUS 03] S. Russell et P. Norvig, "Artificial Intelligence: A Modern Approach" (Second Edition), Prentice-Hall Series in Artificial Intelligence, 2003.

[RUT 08] Leszek Rutkowski, "Computational Intelligence: Methods and Techniques", Springer, 2008.

[SAA 08] Ashraf Saad, "An Overview of Hybrid Soft Computing Techniques for Classifier Design and Feature Selection", HIS '08 Proceedings of the 8th International Conference on Hybrid Intelligent Systems, pp. 579 – 583, 2008.

[SEI 05] Tebogo Seipone, John A. Bullinaria, "Evolving Improved Incremental Learning Schemes for Neural Network Systems", IEEE Congress on Evolutionary Computation, vol. 3, pp. 2002 – 2009, 2005.

[SIN 00] Naresh K. Sinha, Madan M. Gupta, "Soft Computing and Intelligent Systems: Theory and Applications", Academic Press Series in Engineering, 2000.

[SU 06] Mu–Chun Su, Jonathan Lee, Kuo–Lung Hsieh, "A new ARTMAP–based neural network for incremental learning", Neurocomputing, vol. 69, no. 16 – 18, pp. 2284 – 2300, 2006.

[SYE 99] Nadeem Ahmed Syed, Syed Huan, Liu Kah, Kay Sung, "Incremental learning with support vector machines", Proceedings of the Workshop on Support Vector Machines at the International Joint Conference on Artificial Intelligence (IJCAI'99), 1999.

[TAY 00] John G. Taylor, "Bringing AI and Soft Computing Together: A Neurobiological Perspective", Intelligent Systems and Soft Computing, Lecture Notes in Computer Science, vol. 1804, pp. 41 – 71, 2000.

[TET 01] A. Tettamanzi, M. Tomassini, "Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems", Springer, 2001.

[TUN 05] W.L. Tung, C. Quek, "GenSo–FDSS: a neural–fuzzy decision support system for pediatric ALL cancer subtype identification using gene expression data", Artificial Intelligence in Medicine, vol. 33, no. 1, pp. 61 – 88, 2005.

[VIL 07] Christina B. Vilakazi, Tshilidzi Marwala, "Incremental Learning and Its Application to Bushing Condition Monitoring", Advances in Neural Networks, vol. 4491, pp. 1237 – 1246, 2007.

[VIL 09] Christina B. Vilakazi, Tshilidzi Marwala, "Computational Intelligence Approach to Condition Monitoring: Incremental Learning and Its Application", Intelligent Engineering Systems and Computational Cybernetics, pp. 161 – 171, 2009.

[WEN 07] Yi–Min Wen, Bao–Liang Lu, "Incremental Learning of Support Vector Machines by Classifier Combining", Advances in knowledge discovery and data mining, Lecture Notes in Computer Science, vol. 4426, pp. 904 – 911, 2007.

[WIL 96] James R. Williamson, "Gaussian ARTMAP: A Neural Network for Fast Incremental Learning of Noisy Multidimensional Maps", Neural Networks, vol. 9, no. 5, pp. 881 – 897, 1996.

[WU 07] Hu Wu, Yongji Wang, Xiaoyong Huai, "AttributeNets: An Incremental Learning Method for Interpretable Classification", Advances in knowledge discovery and data mining, Lecture Notes in Computer Science, vol. 4426, pp. 940 – 947, 2007.

[XIN 99] Xin Yao, "Evolving Artificial Neural Networks". Proceedings of the IEEE, vol. 87, no. 9, pp. 1423 – 1447, 1999.

[YAL 07] Aycan Yalçın, Zeki Erdem, Fikret Gürgen, "Ensemble Based Incremental SVM Classifiers for Changing Environments", 22nd international symposium on Computer and information sciences, pp. 1 – 5, 2007.

[YEN 99] Gary Yen, Phayung Meesad, "Pattern Classification by an Incremental Learning Fuzzy Neural Network", Proceedings of the International Joint Conference on Neural Networks, vol. 5, pp. 3230 – 3235, 1999.

[YEN 01] Gary Yen, Phayung Meesad, "An effective neuro-fuzzy paradigm for machinery condition health monitoring", IEEE Transactions on Systems, Man, and Cybernetics, vol. 31 – 4, pp. 523 – 536, 2001.

[ZAD 65] Lotfi A. Zadeh, "Fuzzy sets", Information and Control, vol. 8, pp. 338 – 353, 1965.

[ZAD 94] Lotfi A. Zadeh, "Soft Computing and Fuzzy Logic", IEEE Software, vol. 11, pp. 48 – 56, 1994.

[ZAD 97] Lotfi A. Zadeh, "The Roles of Soft Computing and Fuzzy Logic in the Conception, Design and Deployment of Intelligent Systems", Proceedings of the Sixth IEEE International Conference on Fuzzy Systems, vol. 1, pp. 1 – 2, 1997.

[ZIL 01] Ali Zilouchian, Mo Jamshidi, "Intelligent Control Systems Using Soft Computing Methodologies", CRC Press LLC, 2001.