ECOLE SUPERIEURE DE GENIE INFORMATIQUE

MEMOIRE DE RECHERCHE

RESEAUX DE NEURONES FORMELS APPLIQUES A L'INTELLIGENCE ARTIFICIELLE ET AU JEU

Soutenu à PARIS en Septembre 2009

Août 2009
Version 1.06
Auteurs : J.BORDAS, F.TSCHIRHART

Contacts :
jeremie@jbordas.com
fabien@tschirhart.eu

Paris,

Août 2009.

1. Présentation

Ce mémoire de recherche a été réalisé dans le cadre de la dernière année de formation au sein de l'**E**cole **S**upérieure de **G**énie **I**nformatique (**ESGI**) et en vue d'obtenir le titre Expert en Systèmes et Réseaux Informatiques

L'apprentissage du travail de recherche engage la rédaction d'un mémoire de recherche, par binôme d'étudiant. Le mémoire doit répondre à une problématique et y apporter plusieurs éléments de réponse, il présente l'état de l'art dans le domaine concerné ainsi qu'une recherche personnelle et ouverte vers des études orientées et plus approfondies.

2. Remerciements

Nous tenons à remercier Mr Alain Lioret, notre directeur de recherche, pour les connaissances qu'il nous a apporté tout au long de l'année, sur le sujet que nous abordons dans le présent mémoire ainsi que pour l'orientation qu'il nous a suggéré d'entreprendre pour sa rédaction.

Nous voudrions également remercier la totalité des membres du jury pour avoir accepté d'assister et d'être le jury lors de la soutenance de ce mémoire.

3. Sommaire

Réseaux de neurones formels appliqués à l'intelligence artificielle et au jeu

4. Introduction

L'intelligence artificielle, science à l'objectif ambitieux de permettre aux systèmes informatiques d'obtenir des capacités intellectuelles comparables à celles des êtres humains, est longtemps restée une porte à peine entrouverte qui pourrait, selon plusieurs opinions **[BER 06] [CAR 04]**, nous mener vers une nouvelle ère informatique, voir, si l'on se penche sur quelques études **[CAR 04]**, vers un nouvel avenir dans un monde où la conscience ne serait plus propre à l'homme. Et si nous ignorons encore jusqu'où nous conduiront les sciences cognitives[1] et les applications que nous en ferons, peu de doutes subsistent sur le potentiel de ces disciplines et sur l'intérêt que devrait lui porter l'homme.

Aujourd'hui encore, l'intelligence artificielle reste un domaine fermé où trop peu de gens se risquent, considérant la discipline comme obscure, sans avenir proche, voir même comme étant l'alchimie du XX^{ème} siècle **[LED 92]** ou parfois, au contraire, c'est l'idée de la singularité technologique[2] qui freine les ambitions ; pourtant depuis l'apparition du terme au milieu des années 50, les progrès sur le sujet donnent de sérieuses raisons de considérer avec intérêt le potentiel et les ambitions imaginables au travers de l'intelligence artificielle, déjà abordée de multiples façons, au travers entre autre de la logique floue, des réseaux Bayésiens, du Modèle de Markov, des arbres de décisions ou encore des différentes formes d'apprentissage automatique… Parmi ces outils, les réseaux de neurones, apparus quelques années avant le domaine de l'intelligence artificielle, est un modèle de calcul originellement placé dans le domaine des statistiques, mais les méthodes d'apprentissage qu'ils emploient et leur mode de décision, s'appuyant d'avantage sur la perception des informations que sur le raisonnement logique formel, leurs ont permis de devenir l'une des technologies majeure de l'intelligence artificielle « moderne » puisque depuis les années 1990, les réseaux de neurones connaissent un succès dans de multiples secteurs, comme les statistiques et la classification bancaire ou postale, l'investissement boursier, la robotique, la cryptographie et le déchiffrage ou encore l'industrie de l'armement notamment pour le guidage des missiles de croisière ; même l'industrie du jeu vidéo a, à plusieurs reprises fait usage des réseaux de neurones dans le cadre de l'intelligence artificielle **[BAT 96, B&W 01, B&W 03]**.

Ces différents sujets qui ont servis d'étude ou d'application aux réseaux de neurones, utilisent différents modèles de réseaux. Tous ont leurs spécificités, leurs limites et leurs utilités, tous ne sont pas applicables à toutes les problématiques ; et des sujets comme ceux proposés par l'industrie du jeu vidéo, nécessitent une étude cadrée et approfondie pour déterminer le ou les modèles les plus adaptés aux problèmes soumis. C'est pourquoi les réseaux de neurones et plus généralement le connexionnisme[3], qui représentent l'une des nombreuses voies actuellement empruntées pour développer l'intelligence artificielle (ou peut être devrions nous dire, pour essayer de développer l'intelligence artificielle, tant cette notion reste pour le moment, floue et incertaine) restent une technologie complexe qui mérite d'être étudiée et approfondie en direction de l'intelligence artificielle, dont les définitions comme les objectifs restent nombreux, et pour laquelle chacun peut encore y trouver son compte et un but personnel ; ainsi, c'est sur le flan du jeu vidéo, que nous avons décider de débuter notre ascension, utilisant les réseaux de neurones comme outils et une meilleure vision de leur avenir et de leur utilisation, comme objectif.

[1] Ensemble de disciplines scientifiques dédiées à l'étude et la compréhension des mécanismes de la pensée humaine, animale ou artificielle, et plus généralement de tout système cognitif.

[2] Concept, selon lequel, à partir d'un point hypothétique de son évolution technologique, la civilisation humaine sera dépassée par les machines.

[3] Approche utilisée en sciences cognitives, neurosciences, psychologie et philosophie de l'esprit. Le connexionnisme modélise les phénomènes mentaux ou comportementaux comme des processus émergents de réseaux d'unités simples interconnectées.

5. Présentation et historique

5.1 Les réseaux de neurones formels

5.1.1 Principe de fonctionnement

Apparus à la fin des années 1950, suite aux travaux des neurologues Warren McCulloch et Walter Pitts **[MCC 59]**, les réseaux de neurones formels, représentations mathématiques et informatiques de neurones biologiques, permettent à l'instar de leurs homologues biotiques, la résolution de problèmes complexes de diverses natures : logiques, arithmétiques ou encore symboliques ; mais contrairement aux systèmes traditionnels de résolution de problème dont nous avons l'habitude d'utiliser en informatique, un réseau de neurones formels n'est pas conçu selon l'appréciation, par le développeur, du problème à résoudre, c'est le réseau de neurones qui va, à partir des informations reçues sur la problématique, établir un modèle de résolution ; pour ce faire, confronté au problème, le réseau de neurones va apprendre en mettant en œuvre le principe d'induction, à savoir, l'apprentissage par l'expérience. Ainsi, par confrontation avec des situations ponctuelles qui leurs sont soumises, ils infèrent un système de décision intégré dont le caractère générique est fonction du nombre de cas d'apprentissages rencontrés et de leur complexité par rapport à la complexité de la problématique proposée.

Le cœur de l'étude étant alors d'établir les mécanismes permettant de calculer les coefficients synaptiques, ceux la même qui permettent l'induction et donc l'apprentissage. Les coefficients synaptiques représentent ainsi le paramètre le plus important d'un réseau de neurone. La majorité des réseaux de neurones possède un algorithme d'apprentissage, ou d'entraînement, qui permet de modifier et d'adapter les coefficients synaptiques, également appelés "poids", ceci, en fonction de données présentées en entrée ; grâce à cet algorithme, le réseau de neurone va apprendre et sera par la suite capable de retrouver les valeurs qui lui avaient été soumises lors de son apprentissage. Tout l'intérêt du réseau de neurone étant alors de généraliser et de trouver des solutions à des problématiques adjacentes, et ainsi de suite.

Le réseau de neurones ne fournit pas toujours de règle facilement exploitable par un humain et reste souvent une boîte noire qui fournit une réponse quand on lui présente une donnée, mais il ne propose jamais de justification facile à interpréter.

5.1.2 Chronologie

C'est à l'ère de la cybernétique et lors de son premier mouvement qui débuta en 1942, que germa l'idée et le concept des réseaux de neurones formels ; à une époque où les chercheurs, immergés dans la genèse de ce qui deviendra ce que nous connaissons aujourd'hui de l'électronique, de l'informatique, ou encore, de la mécanique moderne, cherchent à concevoir un système, entièrement artificiel, capable de reproduire le comportement humain, ou au moins, son intelligence.

Ce sont alors les scientifiques Warren McCulloch et Walter Pitts, l'un, chercheur en neurologie et l'autre en psychologie cognitive ; qui proposèrent en 1943 le tout premier réseau de neurones artificiels, basé bien entendu, sur le principe de leurs homologues biologique et du système nerveux auquel ils sont liés. A l'image de ces derniers, le modèle proposé par McCulloch et Pitts, permet d'apprendre, de mémoriser des informations voir encore, de traiter des informations incomplètes **[BOR 07]**. Le principe de fonctionnement de ce tout premier réseau de neurone est basé sur la notion de coefficient synaptique : contrairement à ce que l'on trouve usuellement dans les diverses sciences de traitement de l'information, un réseau de neurones ne peut être conçu et utilisé à partir une suite d'instructions figées ou écrites par le concepteur au fur et à mesure de sa compréhension du problème ; le réseau de neurones se veut être capable de résoudre un problème, sans

l'intervention du concepteur, et pour y parvenir, il met en place un modèle de résolution selon les informations qui lui sont transmises, la mise en place de ce modèle de résolution s'effectue via les coefficients synaptiques, paramètre essentiel du réseau.

Mais les travaux de McCulloch et de Pitts n'ont laissés aucune information pour adapter les coefficients synaptiques et il fallu attendre 1949 et Donald Hebb, psychologue et neuropsychologue Canadien, qui donna un début de réponse grâce à ses travaux sur l'apprentissage, *The Organization of Behaviour* où Hebb propose une règle simple permettant de définir les coefficients synaptiques selon les liaisons des neurones. Cette règle, connue sous le nom de "Règle de Hebb", est encore utilisée aujourd'hui.

Moins de dix années plus tard, Franck Rosenblatt propose une application de ces différent travaux, et son réseau de neurones, le Perceptron, devient le tout premier système capable d'apprendre, y compris lorsque certaines des informations qui lui sont fournies, sont erronées. Mais en 1969, Marvin Lee Minsky chercheur en science cognitive et en intelligence artificielle, coécrit avec le mathématicien et informaticien Seymour Papert, un ouvrage mettant en avant les limites du modèle de Rosenblatt, démontrant son incapacité à résoudre des problèmes non linéaire. Ils étendirent implicitement cette limitation, à tous les réseaux de neurones existant ; tous fonctionnant plus ou moins sur le même principe que celui du Perceptron. Ce fut alors le début de sombres années pour les réseaux de neurones, toujours utilisés dans certains milieux, la recherche piétina et les fonds furent rapidement redirigés vers les autres voies de l'intelligence artificielle. Ce n'est qu'en 1982 que le physicien John Joseph Hopfield relança timidement les réseaux de neurones avec son modèle éponyme. Toutefois, son modèle ne permettant toujours pas la résolution de problèmes non linéaire, l'engouement pour les réseaux de neurones restera limité, et il faudra attendre 1984 pour obtenir un modèle de réseaux de neurones qualifié de multicouches et ne possédant pas les défauts démontrés par Minsky et Papert ; ce modèle proposé par Paul J. Werbos et mit en œuvre par David Rumelhart en 1986 repose sur la rétropropagation **[RUM 86]** du gradient[4] de l'erreur dans des systèmes à plusieurs couches.

Depuis; les réseaux de neurones connaissent un essor considérable, et la communauté gravitant autour de leur évolution, progresse sans cesse et multiplie leurs applications, notamment dans le jeu vidéo, avec une première utilisation en 1996 avec *Battlecruiser 3000AD* **[BAT 96]** et *Creatures*, puis une seconde en 2001 avec *Black & White* de Lionhead.

5.1.3 Technologie

- 5.1.3.1 Structure d'un réseau

Un réseau de neurones est constitué de plusieurs couches de neurones connectées entres elles : la couche d'entrée récolte d'abord ses informations à partir de capteurs disposés en dehors du réseau puis répercute l'information sur la couche suivante qui est elle même connectée à une autre couche, et ainsi de suite jusqu'à la couche de sortie. Chaque couche i est ainsi composée d'une quantité Ni de neurones connectés en entrée sur les Ni-1 neurones de la précédentes couche ; ces connexions entre les neurones sont souvent appelés "synapse" à l'image des synapses de notre cerveau, servant de lien entre nos neurones ; chaque synapse possède un poids ou coefficient synaptique, les valeurs de sorties des neurones de la couche précédente sont ainsi multipliés par ce poids puis additionnés aux neurones de la couche suivante ; à chaque couche une fonction de sortie est appliquée et permet de générer la sortie.

[4] Le gradient est une valeur représentant la variation d'une fonction dépendant de plusieurs paramètres par rapport à la variation de ces différents paramètres.

Un neurone lorsqu'il est activé, effectue la somme de ses entrées, préalablement pondérées par leur coefficient synaptique, et applique la fonction d'activation au résultat, pour déterminer sa valeur de sortie, qui peut alors être pondérée par un nouveau coefficient synaptique et envoyé vers un autre neurone.

Cette structure, assez simple de base, peut se complexifier en contenant par exemple des boucles et ainsi changer radicalement les possibilités du réseau.

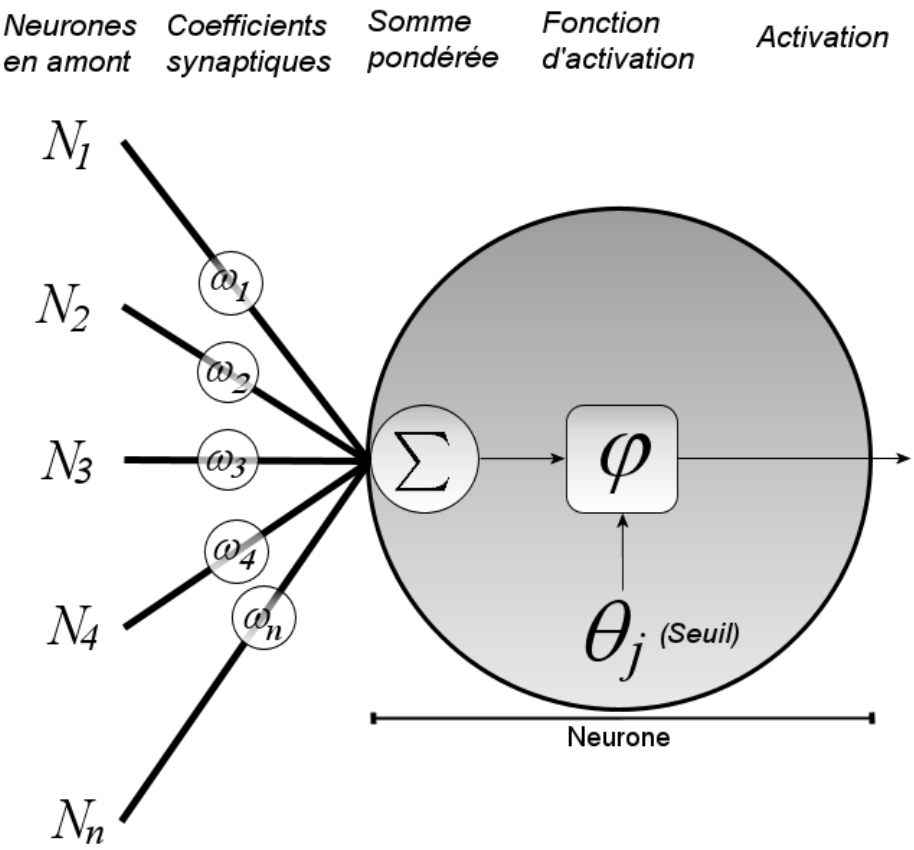


Fig1. *Structure d'un neurone artificiel*

- **5.1.3.2 Comparaison avec le cerveau humain [WIK 09]**

Le principe de fonctionnement des réseaux de neurones, à la fois proche et inspiré du cerveau humain, suggère souvent des comparaisons entre le système formel et notre machinerie biologique ; le cerveau humain comporterait prés de cent milliards de neurones **[WIL 01]**, chacun d'eux étant susceptible de recevoir des informations en provenance de centaines de milliers d'autres neurones, ce qui indique que le cerveau humain pourrait contenir plusieurs millions de milliards de synapses, soit environ 10 000 synapses par neurones. Un neurone ne pouvant émettre une information que toutes les 10ms, chacun d'eux à une fréquence de fonctionnement maximale d'environ 100 Hz, ce qui indique une cadence théorique de 10^{17} d'opérations par seconde, toutefois, les écarts entre les émissions d'un neurone sont des valeurs analogiques et les informations véhiculées le sont au travers de plusieurs impulsions, ce qui rend une éventuelle comparaison avec un ordinateur, extrêmement difficile et probablement pas fiable, surtout si l'on prend en considération le niveau d'incertitude présent autour de chacune des valeurs biologiques précitées. On peut toutefois considérer qu'un cerveau humain serait capable, dans le meilleur des cas, d'effectuer $2x10^{20}$ opérations par seconde.
L'un des plus puissant supercalculateur actuel, en 2009, le Roadrunner, du Département de l'Energie des Etats-Unis (DOE), conçu à partir de presque 6500 processeurs Dual-Core, n'est capable d'effectuer que

$2x10^{15}$ opérations par seconde, soit un PFlops[5]. Il reste donc aujourd'hui encore, un écart de puissance non négligeable entre notre cerveau et les ordinateurs les plus puissants qui soient, mais la puissance de ces derniers évolue très rapidement, puisqu'elle double environ tous les deux ans. Malgré ce différentiel de puissance, il est intéressant de simuler le fonctionnement de neurones pour résoudre des problèmes simples.

Un réseau de neurones biologique fonctionne de la façon suivante : connecté à plusieurs milliers d'autres neurones, un neurone leur transmet une information sous forme d'onde de dépolarisation, ce qui signifie qu'un neurone reçoit en entrée les signaux provenant des autres neurones, au travers de synapses, et émet lui même une information. De la même manière, dans un réseau de neurones formels, ces derniers sont connectés par des liaisons unidirectionnelle, recevant des informations des neurones précédant et en envoyant aux suivants.

Si les réseaux actuels permettent de traiter de façon très correcte des problématiques liées à la perception et ou à la classification, il ne faut pas, à partir de réseaux de neurones seuls, s'attendre à beaucoup mieux avec l'actuelle génération d'ordinateurs. En leur qualité de percepteur, certains réseaux de neurones, associés à des systèmes d'intelligence artificielle évolués, peuvent peut être apporter des résultats plus aboutis, il est d'ailleurs intéressant de remarquer que dans les organismes vivant, comme le notre, la perception des signaux, par nos différents sens, et leur exploitation par notre cerveau, s'effectue au travers d'organes et de processus distincts.

5.1.4 Applications dans l'industrie

Leur aptitude pour la classification et la généralisation a rapidement orienté l'utilisation des réseaux de neurones dans les problèmes de nature statistiques, ainsi, les applications proposant de nombreuses données pouvant être utilisée pour l'apprentissage du réseau sont particulièrement adaptée à leur utilisation, l'achat boursier, la classification de colis, la reconnaissance de caractère voir de visage, sont des utilisations parfaites pour un réseau de neurones. De la même manière, le projet NavLab qui avait pour objectif, la réalisation d'un véhicule autonome capable de circuler sans conducteur et sans commettre d'accidents sur une autoroute, est basé sur l'utilisation d'un réseau de neurones **[BER 06]**, chargé de reproduire la perception qu'un humain a, de la signalisation horizontale, présente sur la chaussée. Le principe étant, comme toujours d'apprendre cette signalisation au réseau, avant de lui en soumettre relativement proche et d'attendre ses suggestions sur la direction à prendre, par le véhicule.

Toujours sur ce principe d'induction, les banques font régulièrement appel aux réseaux de neurones pour accepter ou non une demande de prêt à partir d'un jeu de données : revenu, âge, nombre d'enfants, professions, nature du contrat, autres crédits... Le réseau, pourra à partir des caractéristiques d'un nouveau client, indiquer si il représente ou non un bon client, ceci, en généralisant à partir des cas qu'il connaît.

On retrouve ainsi des réseaux de neurones pour :

- Tout type de classification, allant des colis postaux aux relevés ADN.
- La reconnaissance de motifs, de la signalisation routière au tri des colis postaux ou pour vérifier le montant de chèques.
- Effectuer des approximations de fonctions inconnues.
- Modéliser une fonction connue mais extrêmement complexe.
- Les estimations boursières.

[5] Flops : Floating-point Operation Per Second (opérations à virgule flottante par seconde) est une mesure commune de repère pour évaluer la vitesse des microprocesseurs, il s'agit du nombre d'opérations par seconde, y compris sur des réels.

- Estimer la valeur d'une entreprise à partir de certaines caractéristiques.
- Des tentatives de prédictions des fluctuations boursières.
- L'apprentissage dont le bénéfice pourrait être utilisé par différentes formes d'intelligence artificielle.

Nous reviendrons à plusieurs reprises sur ce dernier point, qui nous intéresse tout particulièrement.

5.2 Intelligence artificielle et jeux vidéo

Un réseau de neurones, seul, n'est pas considéré comme une entité potentiellement intelligente, il n'est qu'un outil destiné à la perception ou à la généralisation. Il représente un organe qui, associé un autre organe, apte à traiter ce que perçoit, apprend et propose un réseau de neurones, détient un potentiel unique et particulièrement encourageant.

« Les réseaux de neurones sont depuis plus de 60 ans la voie d'étude et de développement la plus prometteuse pour la perception et les automatismes sensori-moteurs. »

— Hughes Bersini, *De l'intelligence humaine à l'intelligence artificielle*

Afin de mieux comprendre l'usage que nous pouvons faire des réseaux neuronaux, c'est vers l'intelligence artificielle que nous allons à présent nous tourner. Que cela soit dans sa globalité ou spécifiquement appliquée aux jeux vidéo, l'intelligence artificielle est une science complexe, récente, et particulièrement difficile à cerner.

5.2.1 Principe de fonctionnement

- 5.2.1.1 L'intelligence artificielle

L'intelligence artificielle est une science récente dont l'objectif est de construire des entités douées d'intelligence, toute la complexité résidant dans cette notion encore floue qu'est l'intelligence, faculté de penser et de se savoir pensant **[CAR 04]**, pour certains, aptitude à comprendre ce qui nous entoure au travers de la conceptualisation et du raisonnement, pour d'autres ; l'intelligence reste selon beaucoup, ce qui différencie les *Homo Sapiens* que nous sommes, aux autres animaux, et ce sont ces aptitudes que sont la pensée, le raisonnement, la prise de décision et la résolution de problèmes, que tentent de recréer les acteurs de l'intelligence artificielle.

Mais si tout le monde est loin d'être d'accord sur ce qu'est véritablement l'intelligence, il en est de même pour l'intelligence artificielle, qui génère bien des tensions et divergences d'opinion. S'agit-il de systèmes qui agissent comme les humains ? Qui pensent comme les humains ? Ou bien qui pensent ou agissent rationnellement[6] **[RUS 06]** ? Ces quatre approches de l'IA ont été suivies et le sont encore aujourd'hui, ce que l'on retiendra en premier lieu, c'est que l'approche est soit empirique, basée sur le domaine interdisciplinaire que sont les sciences cognitives ; soit rationnelle, combinant outils mathématiques et informatique pour modéliser les règles et les processus de la pensée et du comportement humain.
Cet imbroglio de points de vue et d'objectifs génère de multiples principes de fonctionnements complexes et spécifiques aux applications, toutefois, qu'ils soient développés sur l'aspect cognitif ou rationnel, la majorité des intelligences artificielles fonctionnent sur le principe de la perception, puis de la réflexion et enfin, de l'action. Cette procédure, parfois exprimée en d'autres termes, est récurrente dans le domaine de l'intelligence artificielle, ce qui varie d'un système à l'autre, d'une modélisation mathématique vers un processus plus em-

[6] Loin de nous l'idée de sous entendre que l'être humain a un comportement irrationnel, mais plutôt que la rationalité dont nous faisons tous preuve n'est pas comparable à celle d'un ordinateur, au fonctionnement purement logiciste sans éléments émotifs perturbateurs.

pirique, c'est son mode de réflexion ; c'est toute la tâche de l'IA que de concevoir ce mode de réflexion, basé sur des modèles, des objectifs, un usage particulier ou bien sur des fonctions d'apprentissage et d'induction, le mode de réflexion renferme toute l'efficacité de l'intelligence artificielle.

Ainsi, en premier lieu, le système va percevoir des informations en provenance de son environnement, ces informations, seront perçues au travers de percepts ou de capteurs ; elles seront alors traitées lors de la réflexion où l'intelligence artificielle aboutira à une décision qui sera sa valeur de sortie, à savoir, son action.

- 5.2.1.2 Intelligence artificielle et jeux vidéo

Dans les applications de type jeux vidéo, l'intelligence artificielle intervient lors de toute prise de décision effectuée par une entité dirigée par le jeu. Si cette décision doit sembler le plus réaliste possible, elle est également contrainte par les intérêts ludiques du jeu. L'objectif étant alors souvent de ne donner au joueur que l'illusion d'un comportement intelligent, sans chercher à atteindre une quelconque forme d'intelligence artificielle forte[7], pour ce faire, les développeurs font souvent usage à une intelligence artificielle scriptée, appartenant à la famille des IA « symboliques » . Cette approche, simple et sous le contrôle total du développeur, est l'exemple même de la simulation d'intelligence : la totalité des réactions du personnage non joueur[8] sont prévues et modélisées par le développeur. Toutefois, certains jeux vidéo font usage de techniques issues de l'intelligence artificielle dites « située » **[GAB 04]** fonctionnant à partir de l'expérience en situation et donc de l'induction et de l'apprentissage. Parmi eux comme cités plus tôt, *Creatures* et *Battlecruiser 3000AD* distribués en 1996, ou enfin *Black & White*, sortit en 2001. *Dungeon Keeper* peut également susciter notre intérêt, puisque selon ses développeurs (Bullfrog) dont sont issus une partie de l'équipe qui développa *Black & White*, le jeu ferait usage d'un processus appelé le « behavioral cloning » **[KEE 97]** dont l'objectif est d'apprendre à partir des agissements du joueur, ce qui est le principe même de l'intelligence artificielle située.

NB : Cette utilisation de l'intelligence artificielle située dans certains jeux vidéo est un point intéressant pour notre étude, puisque les réseaux de neurones formels font partis de cette fameuse famille des IA situées.

5.2.2 Chronologie de l'intelligence artificielle [RUS 06]

- 5.2.2.1 Les sciences cognitives : base de l'intelligence artificielle

Bien que le terme qui la désigne ne soit apparu qu'au milieu du XX^{ème} siècle, bien des disciplines, des écrits et personnes ont participé à l'éclosion de cette science, définie par John MacCarthy durant le séminaire de l'université de Dartmouth en 1956. Déjà, Aristote, au IV^{ème} siècle av. JC, affirme que les actions sont justifiées par un lien logique entre des objectifs et la connaissance du résultat des actions :

« Nous délibérons non sur les fins, mais sur les moyens. En effet, ni le médecin ne délibère pour savoir s'il doit guérir, ni l'orateur pour savoir s'il doit persuader... Mais, ayant posé en principe la fin, ils examinent comment, c'est à dire, par quels moyens, elle sera réalisée. Et s'il se révèle possible de l'obtenir par plusieurs moyens ils examinent par lequel elle le sera le plus facilement et le mieux. Si au contraire elle ne peut être accomplie que par un seul moyen, ils examinent comment elle sera obtenue par ce moyen, et ce moyen lui-même, par quel moyen on l'obtiendra, jusqu'à ce qu'ils arrivent à la première cause, [...] et ce qu'on

[7] L'intelligence artificielle forte fait référence à une machine capable non seulement de produire un comportement intelligent, mais d'éprouver l'impression d'une réelle conscience de soi, voir le chap 7.2.1.

[8] Un personnage non joueur ou PNJ, NPC en anglais (Non Playable Caracter), est une entité entièrement contrôlée par l'ordinateur et par l'intelligence artificielle qui lui est associée.

trouve en dernier lieu dans l'ordre de l'analyse, c'est ce qu'on fait en premier lieu dans l'ordre de réalisation. »
— Aristote, *Ethique à Nicomaque (Livre III, 3, 1112b)*[9]

Cet algorithme de résolution de problème proposé par Aristote et par conséquent, vieux de plusieurs millénaires, est identique à certain processus intégrés dans différents systèmes pourtant nettement plus modernes. Succédant à cette première analyse faite par Aristote, des penseurs, mathématiciens, philosophes, psychologues et plus récemment, informaticiens, ont construit plus ou moins involontairement et au fur et à mesure des siècles, les bases de ce que nous considérons aujourd'hui être « la science de l'intelligence artificielle ». Déjà, Raymond Lulle, philosophe catalan décédé au début du XIV^{ème} siècle avait imaginé une « machine à raisonner » fonctionnant au moyen de roues pivotantes. Le suivant de quelques siècles, le scientifique Wilhelm Schickard conçu la première machine à calculer en 1623. Vint ensuite la Pascaline, en 1642, machine à calculer de Blaise Pascal. Sur un aspect moins concret, René Descartes sera au XVII^{ème} siècle, le premier à affirmer qu'il faut faire une distinction entre l'esprit et la matière. Et que si notre esprit était régit par des lois uniquement physiques alors « *nous aurions autant de libre arbitre qu'une simple pierre* ». Les partisans du dualisme et du matérialisme[10] se sont alors confrontés, philosophant sur la nature physique de l'esprit et du libre arbitre.

C'est durant cette même période que le mouvement empiriste vit le jour, et apporta avec lui l'idée selon laquelle toute connaissance provient de l'expérience. C'est sur cette réflexion que le philosophe David Hume définira en 1739, ce que nous appelons aujourd'hui le principe d'induction, celui-là même que nous utilisons dans nos réseaux neuronaux artificiels. Au XX^{ème} siècle, les philosophes Carnap et Carl Hempel poursuivront cette réflexion au travers de la théorie de la confirmation où ils essayèrent de comprendre comment l'expérience peut mener à la connaissance.

Parallèlement à ces réflexions philosophiques sur l'esprit et le savoir, les mathématiciens s'interrogèrent sur les notions de calculabilité, sur les méthodes de raisonnements à partir d'informations incertaines, ou encore, sur l'établissement de règles formelles qui assurent la validité d'une conclusion. Sur ce dernier point, c'est George Boole qui au au XIX^{ème} siècle élaborera l'algèbre éponyme aujourd'hui utilisé en informatique, en électronique, en probabilité et dans bien d'autres domaines encore. Les recherches mathématiques portant sur la notion de calculabilité nous amenèrent à celle d'indécidabilité[11] **[TUR 38]**, qui induira la nécessité à découper un problème impraticable ou incalculable, en problèmes plus petits et plus simples qui pourront à leur tour être résolus **[AUT 92]**.

Enfin, les méthodes de raisonnements à partir d'informations incertaines ont pus être considérées et traitées en faisant usage des probabilités et des statistiques. C'est le mathématicien Jérôme Cardan qui au XVI^{ème} siècle, a été le premier à proposer le principe des probabilités. Durant les siècles qui suivirent, de nombreux mathématiciens étayèrent l'étude des probabilités, introduisant les méthodes statistiques.

Nous avions déjà les principes philosophiques essentiels de l'intelligence artificielle et nous venons d'y ajouter les outils mathématiques spécifiques que sont la logique, la probabilité et la calculabilité ; mais comment

[9] Traduction Gauthier & Jolif, Presses universitaire de Louvain, 1970.

[10] Le dualisme se réfère à une vision de la relation matière-esprit fondée sur l'affirmation que les phénomènes mentaux possèdent des caractéristiques qui sortent du champ de la physique, s'opposant ainsi au matérialisme qui considère que la matière construit toute réalité.

[11] L'indécidabilité, l'incalculabilité ou encore l'impraticabilité, sont des notions synonymes ayant l'objectif commun de déterminer si une fonction est calculable ou non, les études effectuées sur ces problématique ont permis de définir des procédures de simplification de ces problèmes.

prendre de bonnes décisions ? Quels sont les liens entre la pensée et les actions ? De quelle manière les informations sont elles traitées ?

La théorie de la décision d'Herbert Simon et Bernard Roy, ainsi que la théorie des jeux de Von Neumann, fournissent un vaste ensemble de procédures basées sur le gain, la conjoncture et l'état du système **[NEU 53]**. Cet ensemble de procédure sera très largement utilisé en l'intelligence artificielle. Par ailleurs Simon travaillera également sur la recherche opérationnelle, la rationalité limitée, la théorie des organisations, écrivant de nombreux ouvrages utilisés en intelligence artificielle **[PAR 06]**.

Les liens entre la « pensée » et les actions d'une intelligence artificielle, se veulent bien entendu directement inspirés du cerveau humain, et c'est à ce niveau qu'est intervenue la psychologie. Différents travaux ont servis à modéliser et à concevoir le mode de traitement des informations par une intelligence artificielle, parmi eux, *The Nature of Explanation*, de Kenneth Craik. Dans son ouvrage, Craik pose les bases des liens entre ce que nous pensons et ce que nous faisons : conversion du stimulus en représentation interne, dérivation de cette représentation en de nouvelles représentations et enfin, transformation en action.

Enfin, le traitement des informations, est un aspect qui évolue avec notre savoir sur le fonctionnement de notre cerveau. Nous savons que les différentes parties du cerveau sont affectées à des tâches précises, et nous parvenons progressivement à discerner ces tâches et ces parties, surtout avec l'utilisation de l'IRMf[12]. Toutefois, ce qui reste inexpliqué, c'est que cet ensemble de cellules, de neurones, de synapses, soit à l'origine de la pensée et de la conscience, de l'esprit **[SEA 95]**. A moins de considérer l'alternative du mysticisme qui justifie cette problématique par l'existence d'une dimension supplémentaire, au-delà de la physique, dans laquelle les esprits fonctionneraient. Mais cette approche complexifierait considérablement l'étude de l'intelligence artificielle. Par conséquent, la grande majorité des recherches et des investissements, préfèrent une démarche plus porche de la physique et des sciences dites « rationnelles ».

- **5.2.2.2 Naissance de l'intelligence artificielle et évolution**

L'association d'une partie de ce savoir, acquit au fil des époques, permis à Warren McCulloch et à Walter Pitts d'établir en 1943, le tout premier modèle de réseau de neurones. Leurs travaux sont considérés comme étant la toute première approche de l'IA dans l'histoire **[RUS 06]**. Ce n'est que 13 années plus tard, lors d'un séminaire se déroulant à Dartmouth, que John McCarthy baptisa cette nouvelle science « Intelligence Artificielle ». L'année 1956 représente ainsi, l'année de naissance de l'IA. Les premières années d'existence de l'IA engendrèrent de grandes espérances. Cette nouvelle science permit de résoudre de nombreux problèmes qui paraissaient alors insolubles[13]. En 1963, Le GPS, Global Problem Solver d'Allan Newell et d'Herbert Simon, est l'une des premières applications à succès de l'IA. Ce programme s'approchait de la démarche humaine pour résoudre les problèmes, décomposant la problématique d'une façon comparable à celle des humains.

Toutefois, les premières IA n'obtinrent d'impressionnantes performances que sur la résolution de problématiques simples. Ces performances générèrent un excès de confiance de la part des chercheurs en IA, en leurs systèmes. Mais, à partir de 1966, la quasi-totalité de ces systèmes échouèrent à chaque confrontation à un problème plus compliqué, diminuant considérablement les espérances. La raison de ces échecs réside

[12] Imagerie par Résonance Magnétique Fonctionnelle est une application de l'imagerie par résonance magnétique à l'étude du fonctionnement du cerveau.

[13] Avec le recul, nous sommes à même de relativiser sur ces succès : à cette époque, les ordinateurs étaient considérés incapables d'effectuer autre chose que des opérations arithmétiques. La finesse apportée par l'intelligence artificielle a ainsi permis de résoudre bien des problèmes.

dans le fait que les programmes ne résolvaient les problèmes qu'au travers de manipulation syntaxique et mathématique, sans réellement comprendre la problématique. Ainsi, tout problème requérrant une connaissance générale du sujet (traduction, communication, politique, facteur humain ou social), ne pouvait être résolu par ces programmes. Par ailleurs, ces même programmes résolvaient les problèmes en combinant toutes les solutions possible jusqu'à trouver la meilleure. Si cette procédure était envisageable avec les premiers problèmes, elle devint nettement plus incertaine avec des problèmes contenant beaucoup plus d'éléments et de solutions possibles.

C'est alors qu'apparaissent les premiers systèmes experts, notamment avec le programme DENDRAL développé en 1969. Ce programme permet, à partir d'informations fournies par un spectromètre, de résoudre « le problème d'inférence d'une structure moléculaire »[14]. Pour cela, le programme dispose d'une part, de méthodes de résolution combinatoires, générant toutes les structures possibles à partir des informations fournies, les comparant aux données effectives permettant enfin, de déterminer la solution ; et d'autre part, d'une large base de connaissances constituées à partir du savoir d'experts en chimie analytique. Cette base permettait alors d'effectuer des sélections durant les combinaisons et les comparaisons, diminuant considérablement le nombre de candidats possible. Cette approche combinant les opérations syntaxique et le savoir, permet de résoudre des problèmes de la même façon qu'un expert. Toutefois, elle ne peut résoudre une problématique face à laquelle un expert resterait bloqué, puisqu'elle utilise la même base de connaissances **[IGN 91]**.

C'est à partir de 1980 que l'intelligence artificielle connaît son véritable essor et son utilisation à grande échelle. De nombreuses entreprises se dotent de systèmes experts, ces derniers permettant parfois d'économiser jusqu'à 40 millions de dollars par an. Les Etats-Unis et le Japon se sont alors lancés dans une course à l'intelligence artificielle avec respectivement, les projets MCC[15] et Cinquième Génération. Mais à la fin des années 1980, l'intelligence artificielle connue une nouvelle période de stagnation, les systèmes expert n'évoluant plus, et les ambitions du MCC et du projet Cinquième Génération restant inatteignable **[CRE 93]**. Cette période est connue comme étant le second « hiver de l'intelligence artificielle », le premier étant survenu juste avant l'apparition des systèmes expert.

Beaucoup de choses se sont formalisées durant cet « hiver » : l'intelligence artificielle, souvent abordée de façon désordonnée par les chercheurs voulant essayer toutes sortes d'idées sans aucun conformisme, s'est progressivement tournée vers une approche plus scientifique. Les expérimentations s'appuyant désormais sur des tests rigoureux et les résultats étant évalués au moyen d'analyses statistiques poussées, ont transformé le domaine de l'intelligence artificielle en une science à part entière.

- 5.2.2.3 L'intelligence artificielle de nos jours

Le formalisme dont a été touché l'IA a également atteint les domaines adjacents comme l'informatique, la robotique, la vision, la linguiste, la physiologie, ou encore, les jeux vidéo. Toutefois, cette formalisation a eu tendance à fragmenter ces différents domaines, les isolant de plus en plus les uns des autres, mais l'objectif unificateur de l'intelligence artificielle permet souvent de réunir sur différents projets, des domaines qui s'étaient éloignés ; c'est le retour progressif des sciences cognitives **[VAR 97]** qui avaient disparues avec les systèmes experts et les objectifs industriels que sont la productivité et le bénéfice.

[14] Cette problématique précise sortant du cadre du mémoire, elle ne sera pas approfondie ou détaillée.
[15] Microelectronics and Computer Technology Corporation.

Depuis 1995, ce sont les agents intelligents qui font leur apparition ceci essentiellement au travers de l'Internet : les moteurs de recherche et les « bots »[16], mais de nombreux systèmes embarqués font également usage de ces agents. Ces derniers se décrivent comme des entités qui captent des informations en provenance de leur environnement, déterminent l'action à entreprendre et enfin, l'exécutent. C'est un principe de fonctionnement qui a aujourd'hui tendance à se généraliser lorsque l'on emploi l'intelligence artificielle dans un système, et c'est également le principe de fonctionnement qui nous intéresse le plus, dans ce mémoire, au travers des applications aux jeux vidéo. L'apparition et le développement de l'Internet, le renouveau des ambitions robotique, et plus récemment, l'ouverture du jeu vidéo à l'intelligence artificielle, sont trois des principaux axes de développement et d'évolution de l'intelligence artificielle ; nous reviendrons à plusieurs reprises sur le dernier de ces trois domaines.

5.2.3 Technologie de l'intelligence artificielle

Les intelligences artificielles actuelles, comme les agents intelligents, fonctionnent en trois étapes : Premièrement, elles doivent capter une information. Cette information peut provenir d'une donnée fournie par l'utilisateur, d'un paquet réseau, d'une disquette ou d'un capteur quelconque branché sur l'une des entrées du système. Cette information va générer un stimulus et s'en suivra alors la seconde étape, celle de la décision. C'est durant cette étape que s'effectue l'essentiel du travail de l'intelligence artificielle : manipulation des connaissances, combinaisons, comparaison, apprentissage, généralisation… Ceci, dans l'objectif de déterminer une action. Une fois cette action déterminée, elle peut être mémorisée ou non, selon le type d'IA. Enfin, elle est activée, c'est alors la dernière étape : l'action. Les deux activités principales d'une intelligence artificielle lors de son processus décisionnel, sont le raisonnement et l'apprentissage.

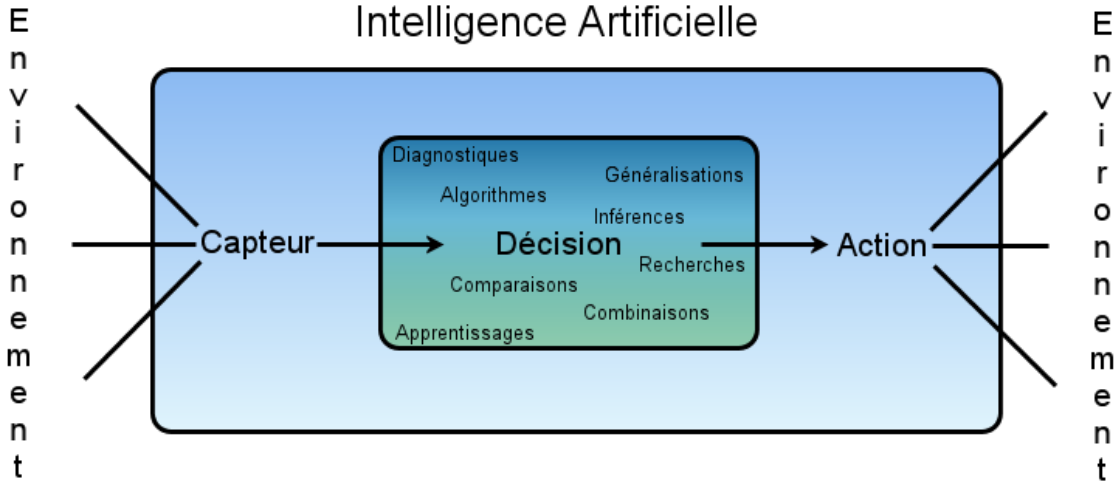


Fig2. *Principe de fonctionnement d'un agent intelligent*

- 5.2.3.1 Raisonnement : le mécanisme d'inférence

Le raisonnement, ensemble de propositions logiques cohérentes desquelles résulte une conclusion, ne peut être effectué à partir d'opérations arithmétiques. Pour manipuler leur base de connaissances, les intelligences artificielles font usage d'inférence. L'inférence permet d'établir les liens entre les connaissances.
Si on sait que les humains doivent dormir pour survivre, et que Bob est un humain, alors Bob doit dormir pour survivre. C'est une déduction. L'IA peut également fonctionner par induction : si Bob est un humain et qu'Alice l'est également. Et que Bob et Alice doivent dormir pour survivre, alors on peut induire que les humains

[16] Un bot informatique, est un programme développé pour réaliser des tâches de façon automatique ou semi-automatique (déclenchement manuel).

doivent dormir pour survivre[17]. Enfin, l'abduction permet de déterminer les prémices d'une connaissance : si Bob a besoin de dormir pour survivre et que les humains ont besoin de dormir pour survivre, alors Bob est un humain[18].

Ces trois méthodes d'inférence permettent de couvrir une grande partie des applications de l'intelligence artificielle. La déduction est surtout utilisée dans la reproduction de la pensée humaine, elle est généralement juste et est applicable à un monde ouvert où toutes les connaissances ne sont pas acquises. L'abduction est utilisée pour effectuer des diagnostiques ou pour établir des plannings, l'induction sert surtout pour les applications d'apprentissage basé sur des exemples **[FAL 09]**.

L'induction et l'abduction pouvant établir des règles erronées, on considère souvent ces deux méthodes comme étant réservées aux mondes clos, où toutes les connaissances sont acquises. Toutefois, les erreurs qui peuvent être commises par une intelligence artificielle faisant usage d'induction ou d'abduction, pourraient également l'être par un humain.

- **5.2.3.2 Apprentissage : la clef de l'intelligence**

L'apprentissage est souvent considéré comme la caractéristique principale de l'intelligence. De nombreuses études en IA ont donc été consacrées à l'apprentissage et aux méthodes qui permettent à une machine, d'étendre ses connaissances. Le principe de l'apprentissage devient alors le suivant : les informations captées ne servent pas uniquement à prendre une décision mais également à améliorer le système. Pour cette tâche, c'est l'apprentissage artificiel[19] par induction qui est aujourd'hui retenu par la communauté scientifique. Mais quelque soit le mode d'apprentissage, on discerne habituellement trois situations : l'apprentissage supervisé, l'apprentissage non supervisé et enfin, l'apprentissage par renforcement aussi connu sous le terme « reinforcement learning ».

L'apprentissage supervisé est une forme d'apprentissage assisté ; dans cette situation, le système obtient la valeur à retenir par un élément extérieur, généralement l'utilisateur. Imaginons un cas (1) où le système doit décider de tourner une clef vers la gauche ou vers la droite. Toutefois, il ignore que l'objectif est de déverrouiller la porte, il lui sera alors fourni par l'utilisateur. Une fois cet objectif en main, le système essaiera de tourner la clef et déterminera le bon sens de rotation permettant d'aboutir à l'état indiqué par le superviseur.
L'apprentissage non supervisé est identique au supervisé, à la différence qu'aucun utilisateur n'est présent pour indiquer la valeur à obtenir. Ainsi, dans le cas (1) vu précédemment, le système va tourner la clef dans un sens puis dans l'autre et déverrouiller la porte, mais sans comprendre que ce qu'il a fait est correct. Un apprentissage artificiel sans aucune supervisation ne saura jamais si ses actions sont correctes ou non. Il est intéressant d'utiliser ce type de système en compétition avec d'autres, ou de lui fournir des statistiques comme support d'apprentissage.

Enfin, l'apprentissage par renforcement qui fonctionne également sans superviseur, apprend grâce au renforcement. Ce terme signifie que le système est guidé au moyen de récompenses qui sont attribués selon ses actions. Le système cherche, au travers d'expériences itérées, une stratégie optimale, en ce sens qu'il maximise la somme des récompenses au cours du temps. L'apprentissage par renforcement est considéré

[17] Jusqu'à ce qu'une nouvelle règle vienne prouver le contraire. En effet, si Bob et Alice sont humains et que Bob et Alice aiment les fraises, cela ne signifie pas forcément que tous les humains aiment les fraises. Mais cela permet tout de même une généralisation assez proche de la réalité, la notion de goût individuel étant une problématique secondaire, c'est une analyse statistique qu'il faudrait intégrer pour valider ou non cette règle.
[18] D'avantages de connaissances permettrait de tempérer cette affirmation, si par exemple l'intelligence artificielle apprenait que les singes ont besoin de dormir, il hésiterait sur la nature de Bob, et resterait en attente de connaissances complémentaires.
[19] Apprentissage artificiel : apprentissage dans le cadre de l'amélioration des performances d'une intelligence artificielle.

comme étant le meilleure mode d'apprentissage actuellement utilisé, il est extrêmement proche au mode d'apprentissage naturel[20] et présente ainsi de nombreuses similitudes avec notre cerveau. Les chercheurs en apprentissage artificiel ont en partie redécouvert en partie ce que la nature a mit en place. La zone du cerveau qui montre des analogies avec les algorithmes d'apprentissage par renforcement s'appelle les ganglions de la base, dont une sous partie appelée la substance noire émet un neuromodulateur, appelé la dopamine. Cette dopamine renforce chimiquement les connexions synaptiques entre les neurones. Ce fonctionnement des ganglions de la base a été identifié comme existant chez l'ensemble des vertébrés **[RED 99]**. On retrouve également ces résultats en imagerie médicale chez l'homme **[ODO 04]**.

5.2.4 Applications de l'intelligence artificielle dans l'industrie

L'intelligence artificielle trouve aujourd'hui sa place dans de nombreux secteurs, qu'il s'agisse de haute technologie, de recherche ou d'applications destinées aux particuliers. La NASA faisait ainsi usage, il y a quelques années, d'un programme nommé « Remote Agent » pour la planification autonome d'un vaisseau spatial. Ce programme surveillait les opérations à bord au fur et à mesure de l'exécution de la mission. Il était capable de détecter, de diagnostiquer et de résoudre les anomalies qui survenaient **[NAS 09]**. Certains prototypes comme le projet NavLab cité plus tôt, utilise l'intelligence artificielle pour un contrôle autonome d'une machine, dans notre cas, une voiture. La médecine n'hésite pas non plus à faire usage de l'intelligence artificielle pour effectuer des diagnostiques médicaux à partir des symptômes des patients. Ou encore, HipNav est un programme qui utilise une technique de « vision artificielle » afin de simuler un modèle 3D de l'anatomie interne d'un malade. Par ailleurs, l'usage d'intelligence artificielle pour la planification logistique est devenu courant, on notera notamment l'utilisation du logiciel DART[21] qui permit la gestion automatique de la logistique et du transport de plusieurs dizaines de milliers d'unités durant la guerre du golfe Persique survenue en 1991 **[RUS 06]**.

Toujours plus efficace et performante, la robotique remplace l'homme dans nombre de ses métiers les plus dangereux. Une nouvelle fois, c'est l'intelligence artificielle qui, associée au savoir des roboticiens, permet d'effectuer ces travaux avec au moins, la même qualité qu'un homme. Les systèmes experts sont répandus à travers le monde et de nombreuses entreprises en font usage, banques, assurances, entrepreneurs, traders, agences immobilières, tous utilises des systèmes experts pour répondre automatique et rapidement à des problématiques qui leur sont régulièrement soumises par leurs clients.

- 5.2.4.1 L'intelligence artificielle dans les jeux vidéo

L'intelligence artificielle est longtemps restée un problème secondaire dans les jeux vidéo, la priorité ayant longtemps été l'aspect visuel et l'ambiance sonore. Mais les environnements de plus en plus réaliste habités par des personnages tout aussi esthétiques, poussent les studios de développement à augmenter les fonds et ressources destinés à l'IA. Afin d'éviter un contraste entre la qualité visuelle et la crédibilité du comportement des personnages, l'intelligence artificielle est devenue, depuis le début des années 2000, un élément central pour de nombreux studios de développement, soucieux de percer dans cette technologie encore récente dans ce secteur de l'industrie. Certains jeux comme Creatures ou Black & White, sont reconnus pour leur audace en terme d'intelligence artificielle. Toutefois, une majorité des jeux vidéos se contentent d'employer une intelligence artificielle scriptée où chaque action des PNJ a été prévue par le développeur. Dans ce type de jeu, l'ordinateur disposera ses unités autour du joueur, parce que le développeur lui aura indiqué, à l'avance, de procéder de cette façon ; l'ordinateur ignore qu'il est en train d'encercler le joueur, il ignore

[20] Ce sont d'ailleurs les travaux effectués sur l'apprentissage par renforcement, pour l'intelligence artificielle, qui ont guidés les neurobiologistes et les psychologues pour la compréhension du fonctionnement du cerveau **[HOU 95a]**.

[21] Dynamic Analysis and Replanning Tool, outil d'analyse et de replanification dynamique utilise par l'armée américaine.

également l'utilité de ce procédé, et en cas d'échec, il ne cherchera pas à s'améliorer. Certains procédés non évoqués dans ce mémoire, comme les algorithmes et la programmation génétique ont également été mis en œuvre dans le jeu Spore d'Electronic Arts. Enfin, sur le devant de la scène de l'intelligence artificielle ludique, le projet Milo, dirigé par l'ambitieux Peter Molyneux, qui n'écarte pas la possibilité de réussir le test de Turing avec son nouveau projet.

Fig3. *Milo – jeu vidéo de Peter Molyneux, développé par Lionhead pour Xbox360*

Quelle est la place des réseaux de neurones dans cette industrie ? Comment peuvent-ils contribuer à ce récent engouement pour l'intelligence artificielle ? C'est à ces questions que nous allons essayer de répondre dans la suite de ce mémoire.

6. Intelligence artificielle et réseaux de neurones formels, quel avenir ?

6.1 Avenir de l'intelligence artificielle [RUS 06] [CAR 04] [BER 06]

Pouvons nous aujourd'hui estimer avec plus ou moins de précision, l'avenir de l'intelligence artificielle ? Les bases de la science sont posées et reconnues, des objectifs concluant on été atteints et les différentes industries se partagent les technologies et les ambitions. Mais quelles sont ces ambitions ? Que pouvons nous nous permettre d'envisager pour cette science dont l'objectif originel semble encore très incertain et à peine palpable ? Pour dessiner une vision plus concise de l'état de l'art et de son avenir, nous faisons le choix de découper une intelligence artificielle moderne en sous éléments afin d'en décrire l'état technologique et de déterminer ce qui est alors envisageable dans un futur plus ou moins proche.

6.1.1 Etat de l'art

- **6.1.1.1 Interactions avec l'environnement**

La plupart des intelligences artificielles conçues jusqu'à aujourd'hui ont toujours eu besoin d'intervention humaine pour fournir les entrées ou pour interpréter et utiliser les sorties. Si il existe des systèmes quasiment autonomes, ils restent rares et développés pour des applications spécifiques. Ces dernières années, nous avons vu apparaître des IA plus autonomes avec leur environnement, pouvant ainsi y évoluer sans intervention humaine. Ces IA ne nécessite qu'une programmation haut niveau de l'utilisateur, elle sont donc programmables et prêtes à l'emploi. C'est dans la robotique de haute technologie que l'on trouve les meilleures avancées en terme d'interactions avec l'environnement : Sony Corporation notamment, avec le chien robot AIBO ou son humanoïde bipède Qrio qui est capable de reconnaissance vocable et faciale, de localiser des objets ou du son dans l'espace. Il peut se déplacer sur un sol mobile ou encombré et réagit pour maintenir sa stabilité si on le pousse. Il est le premier robot capable de sauter ou de courir[22]. Enfin, il peut interagir avec son environnement extérieur par synthèse vocale et soutenir une conversation simple avec un être humain **[FIE 06]**. Ces robots illustrent directement les progrès effectués en terme d'interactions avec l'environnement.

Dans l'avenir, les avancées technologiques en matière de miniaturisation et de microsystèmes électromécanique[23] devraient permettre d'intégrer un plus grand nombre de capteurs et de contrôleurs aux futures IA. Stuart Russel et Peter Norvig imaginaient notamment, en 2003, des insectes volants artificiels utilisant ces technologies. Aujourd'hui, de nombreux robots ont été développés **[HAR 09]** et sont notamment utilisés à des fins militaires et expérimentales.

- **6.1.1.2 Lecture de l'environnement**

La lecture de l'environnement est une problématique liée à la précédente. Si dans la recherche d'interactions nous considérions essentiellement des robots capables de capter ou d'émettre des informations, la lecture d'information relève de la capacité de l'IA à comprendre les informations reçues ou émises. Les progrès effectués en matière de robotique détaillés plus tôt illustrent déjà notre avancé sur ce domaine. Le projet de jeu, Milo, de la société Lionhead présente également de nombreuses aptitudes d'interactions et de lecture

[22] Déclaration controversée à cause de sa vitesse de course extrêmement faible : 2,4Km/h, Sony considère que Qrio peut courir car il passe par une phase de vol où les deux pieds du robot sont décollés du sol.

[23] Un microsystème électromécanique (également appelé MEMS) est un système comprenant des éléments mécaniques et utilisant l'électricité comme source d'énergie. L'ensemble est contenu dans une structure présentant des dimensions micrométriques.

de l'environnement, notamment lorsque des discussions entre le joueur et le personnage ou lorsque ce dernier récupère et identifie un dessin réalisé à la main, sur du papier, par le joueur **[JOY 09]**.

- **6.1.1.3 Planification des actions**

Dans une intelligence artificielle, le planificateur a pour tâche d'établir la suite d'actions à effectuer pour atteindre l'objectif déterminé. Pour ce faire, le planificateur manipule trois éléments : Une description de l'état de son environnement, un objectif et un ensemble d'actions qui peuvent être réalisées. Ces actions sont associées à des post conditions : des effets sur l'environnement, qui sont connues du planificateur.

Si il existe de nombreuses procédures de planification on retrouve essentiellement les méthodes suivantes :

- La planification linéaire, où les actions sont effectuées dans l'ordre indiqué des objectifs, cette méthode pose un problème évident puisque les objectifs ne sont pas forcément dans le bon ordre, et si ils le sont, c'est qu'il y a une intervention extérieure qui effectue la planification en lieu et place du planificateur.

- La planification non-linéaire, qui permet de résoudre la problématique précédente mais qui contient toujours une intervention extérieure chargée d'ordonner les actions.

- La planification probabiliste de l'incertitude, qui fait usage d'un processus de décision Markovien, à base d'états, d'actions et de récompenses. Cette planification est utilisée lorsque des doutes subsistent sur l'effet des actions. Le processus de décision Markovien prend en considération le fait qu'une action peut occasionner l'effet escompté, ou son inverse. Il est intéressant de noter que cette méthode est également utilisée pour l'apprentissage par renforcement, que nous avions abordé plus tôt.

- La planification en ordre partiel est une extension de la planification non-linéaire, à la différence qu'il nécessite un ensemble de contraintes d'ordonnancement, qui lui permettront de disposer les actions dans le bon ordre sans intervention extérieure.

- La planification hiérarchique, utilisée pour les problèmes complexes, qui effectue une abstraction des éléments de moindre importance afin de gérer le problème à haut niveau, avant d'y appliquer une granularité et de détailler selon la nécessité. Cette décomposition en éléments simples permet de faciliter la résolution des problèmes dont ils découlent et de déterminer une solution là où un autre planificateur échouerait.

Il existe de nombreuses autres approches de la planification, comme la planification conformante qui fonctionne sans perception et permet ainsi de traiter des informations incomplètes ou la planification continue qui génère de nouveaux objectifs durant le déroulement du plan ce qui permet un traitement temps réel des états. La planification muli-agent est utile lorsque le système est placé dans un environnement contenant d'autres agents intelligents.

Toutes ces approches possèdent leurs partisans et leurs détracteurs, et aucune ne fait l'unanimité ou n'est considérée comme étant la meilleure. Toutefois, la richesse de ces différentes démarches, leur association et leur croisement permettent d'intensifier les progrès et l'efficacité des systèmes de planification. Si de nombreux travaux doivent encore être effectués, l'apprentissage par renforcement hiérarchique, qui lie la planification hiérarchique aux modèles de Markov utilisés dans l'apprentissage par renforcement, représente l'une des voies les plus prometteuses de la planification **[RUS 06]**.

- **6.1.1.4 Utilité des actions**

L'utilité des actions est parfois considérée comme une branche de la planification, en effet, on retrouve souvent la problématique de l'action et de son rôle, lors de la création du plan d'actions. Il est toutefois préférable de considérer cette phase du raisonnement artificiel comme l'un des aspects les plus importants de la prise de décision. C'est lors cette prise de décision que l'on fait intervenir la théorie de l'utilité[24], cette dernière a pour objectif de maximiser l'utilité de chaque action. C'est principalement dans cet objectif que le choix du modèle de décision Markovien est fait ; son fonctionnement à base de récompenses permet d'orienter le choix du planificateur vers des actions plus utiles. Toutefois, la valeur de l'utilité varie selon le référentiel ; en effet, selon le mode de construction de ses fonctions d'utilités, une intelligence artificielle plongée dans un environnement complexe peut finir par s'orienter vers des actions peu souhaitables. C'est la cas notamment si l'on considère une probabilité d'incertitude sur le résultat d'une action : absence de l'effet escompté. Ce type de situation peut s'avérer catastrophique et il est parfois important d'intégrer des fonctionnalités de gestion du risque qualifiée de « risquophobes ».

Ce travail sur la préférence et l'utilité des actions s'avère aujourd'hui très lourd et très complexe à mettre en œuvre. Si la fonction de récompense utilisée en planification est très simple, la fonction d'utilité de l'action qui lui est associée est nettement plus compliquée et il reste très difficile d'intégrer à des intelligences artificielles complexes un mode décisionnel permettant d'aboutir à ce que nous attendons d'elles. Le professeur Stuart Russel et le docteur Peter Norvig suggèrent de faire usage de l'ingénierie des connaissances, notamment utilisées dans les systèmes experts avec d'intéressants résultats, pour indiquer à nos intelligence artificielles les résultats que nous attendons **[RUS 06]**.

Mais l'usage d'une telle base de connaissances risque de poser les mêmes problèmes que pour les systèmes experts, à savoir que le système ne pourra aboutir à une solution qui n'aurait été trouvée par un expert humain. Ceci est dû au fait que cette base ne peut évoluer sans l'intervention humaine. L'apprentissage peut être l'une des façons de s'affranchir de ce frein.

- **6.1.1.5 Apprentissage**

Nous l'avons vu plus tôt, il existe trois types d'apprentissages : l'apprentissage supervisé, où une entité externe assiste le système en lui indiquant si ce qu'il acquiert lui sera utile ou non. L'apprentissage non supervisé, où le système est laissé à lui-même. Enfin, l'apprentissage par renforcement, basé sur le principe de récompense positive ou négative selon les actions du système, afin d'orienter ce dernier. Cette méthode, très proche de la méthode humaine, semble être la meilleure voie en cours d'exploration. Sa variante, l'apprentissage par renforcement hiérarchique est également très prometteuse et permet d'assembler les actions d'une bibliothèque en différents niveaux : *OuvrirPorte* contiendrait alors les actions *TendreBras*, *SaisirPoignée*, *TournerPoignet* et *FléchirBras*. L'action de haut niveau *QuitterPièce* contiendrait *MarcherVersPorte*, *OuvrirPorte* et *MarcherAuTraversPorte.* Au sein de cette vaste tâche, celle de l'apprentissage est de valider et de mémoriser les micros actions. Ces dernières sont validées si elles permettent d'atteindre l'objectif déterminé, ici, d'ouvrir la porte ; le système essaie différentes façons qui échouent, et fini par trouver une méthode, il la valide et la mémorise alors. L'apprentissage par renforcement hiérarchique permet également d'utiliser des

[24] La théorie de l'utilité (ou l'utilitarisme) est une doctrine éthique qui prescrit d'agir de façon à maximiser le bien-être du plus grand nombre. Elle considère que la valeur d'une action est déterminée à partir de son utilité générale. Elle est nécessairement conséquentialisme, puisque la valeur ne peut être déterminer sans prendre en compte toutes les conséquences d'une action.

variantes pour une même action de haut niveau ; selon le type de porte ou de poignée, l'approche peut nécessiter d'être différente **[MIT 97]**.

- **6.1.1.6 Raisonnement et rationalité**

Le raisonnement reste la grande inconnue de l'avenir des intelligences artificielles. Tant que nous ignorerons si le raisonnement descend de la conscience et si cette dernière répond à des règles physiques ou mystiques, alors nous resterons incertain sur les modèles de raisonnement à implémenter. Aujourd'hui nous nous tournons de plus en plus vers des architectures hybrides, intégrant des techniques dites « système réflexe » et « système compilé ». Le premier de ces deux systèmes, très rapide, fonctionne uniquement à partir des informations captées, alors que le second, nettement plus lent, prend également en compte des informations extraites de sa base de connaissance. La fusion de ces deux systèmes permet de créer des réponses réflexes à partir de solutions établies et validée en utilisant une méthode compilée.

Si ce modèle de raisonnement est efficace, est-il pour autant satisfaisant ? La vision de l'avenir en terme de raisonnement artificiel reste extrêmement floue, et il est difficile de dire si nous sommes ou non sur la bonne voie, et bien que nous puissions relever des progrès constant et régulier, nous restons incapable de déterminer si l'objectif qu'est un raisonnement de type humain est atteignable ou non **[CAR 04]**.

Mais cet objectif est-il le bon ? L'homme a réussi à voler lorsqu'il a cessé d'imiter les oiseaux, peut être parviendra t-il, de la même façon, à concevoir une intelligence artificielle lorsqu'il cessera d'essayer de reproduire sa propre intelligence. Jusqu'alors, quatre voies ont été empruntées :

- Celle de la rationalité parfaite, qui peut être employée dans des environnements fermés et de petites dimensions mais qui en revanche nécessite beaucoup trop de performances ou de temps dans un environnement complexe. Cette voie pourra peut-être être empruntée si nous parvenons à augmenter significativement la puissance de nos machines[25].

- Celle de la rationalité calculatoire, qui utilise la théorie de la décision associé à une rationalité parfaite, mais de la même façon que pour cette dernière, le temps de calcul reste beaucoup trop long. Pour palier à ce problème, le système effectue des compromis sur la qualité de la réponse, ce qui lui permet de gagner en temps, toutefois, ces compromis ne sont pas toujours adaptés au problème et la réponse devient alors insatisfaisante.

- La rationalité limitée de Herbert Simon, fonctionne à partir du choix le plus satisfaisant, ce dernier n'est déterminé que sur la contrainte de temps : il ne cherche qu'une réponse suffisamment bonne sur le temps imparti. Le problème s'inverse alors, et c'est la qualité de la réponse « suffisamment bonne » qui ne convient pas toujours. Toutefois, cette option peut être utilisée dans de nombreux cas d'urgence, en tant que « répons réflexe ».

- Enfin, l'optimalité limitée, suggère qu'une intelligence artificielle doit tout simplement faire de son mieux selon ses ressources. L'objectif n'étant plus d'effectuer des actions optimales à l'utilité élevée, mais de concevoir des programmes optimaux ; l'homme ne devant intervenir qu'au niveau du programme et laisser ce dernier se charger des actions. L'idée sous-jacente à l'optimalité limitée est de mettre en concurrence différents programmes et d'utiliser la meilleure solution. Alors que dans des programmes basé sur la rationalité toutes les solutions établies dans un contexte identiques seront identiques, un programme à optimalité limitée pourra trouver différentes solutions.

[25] Bien plus que ce que ne prévoit la loi de Moore.

6.1.2 Dans le futur

- 6.1.2.1 Un avenir riche mais incertain

Nous l'avons dis à plusieurs reprises, beaucoup d'études sont encore à faire et de nombreuses voies restent à explorer ou à découvrir. Quelle que soit l'approche que l'on entreprend de l'intelligence artificielle, on trouvera des travaux à effectuer et des études à mener. Les applications industrielles de l'intelligence artificielle se font de plus en plus régulières et l'on peut espérer une émancipation de la science dans les années à venir.

Malgré cela, nous ignorons encore où ces études finiront par nous mener, l'objectif originel de l'intelligence artificielle est-il atteignable ? Resterons nous éternellement en périphérie de nos ambitions ? Nous sommes incapables de le dire et les succès de la discipline, bien que convaincants, ne semblent toutefois pas effleurer l'idée d'un esprit factice, d'une intelligence artificielle forte, consciente de ce qu'elle est. Si elle ne s'est pas éteinte, la perspective d'une telle IA semble s'essouffler au sein de la communauté scientifique. Les chercheurs s'intéressant plus à la réalisation de programmes fonctionnels destinés aux industriels qu'au tâtonnement, à l'incertitude et à l'expérimentation qu'implique cette voie.

Toutefois, les ambitions de certaines industries ont des chances d'orienter petit à petit les fonds et les recherches, sur cette piste entamée il y a soixante années. L'industrie robotique et les projets d'humanoïdes débutés par Honda, tel ASIMO, ou par Aldebaran Robotics, comme Nao ; suggèrent à long terme, des recherches extrêmement poussées sur la simulation de consciences. L'industrie du jeu vidéo, dont le réalisme visuel rapproche de plus en plus le virtuel du réel, suscite également un grand intérêt pour l'animation et le comportement des personnages non joueurs que les jeux contiennent. En effet, plongés dans un monde quasi identique à la réalité, ces personnages devront contribuer et participer à l'immersion du joueur dans le jeu. Cela ne serait possible qu'avec l'usage d'intelligences artificielles fortes.

- 6.1.2.2 Conjecture de la singularité technologique

Le concept de la singularité technologique, introduit par Von Neuman dans les années 1950, considère qu'il y a un point de l'histoire future, où l'homme sera intellectuellement dépassé par les machines **[ULA 58]**. Dés lors, les progrès ne seront plus l'œuvre de l'homme mais des intelligences artificielles qu'il aura conçu, ce qui sous-entend que l'activité humaine telle que nous la connaissons aujourd'hui, ne pourra plus se poursuivre. Le professeur Irvin John Good prononcera à ce sujet :

« Supposons qu'existe une machine surpassant en intelligence tout ce dont est capable un homme, aussi brillant soit-il. La conception de telles machines faisant partie des activités intellectuelles, cette machine pourrait à son tour créer des machines meilleures qu'elle-même; cela aurait sans nul doute pour effet une réaction en chaîne de développement de l'intelligence, pendant que l'intelligence humaine resterait presque sur place. Il en résulte que la machine ultra intelligente sera la dernière invention que l'homme aura besoin de faire, à condition que ladite machine soit assez docile pour constamment lui obéir. »

— Irvin John Good*, Speculations Concerning the First Ultraintelligent Machine*, (Advances in Computers, vol 6, page 33).

Les tenants de la thèse de la singularité attendent cette dernière pour le XXI^{ème} siècle. Les futurologues estimant son apparition pour les années 2030 et certains mouvements transhumanistes vers la fin de ce même siècle **[KUR 07]**. Toutefois, cette théorie ne fait pas l'unanimité et est abondamment critiquée par la commu-

nauté scientifique, considérant les partisans de cette thèse comme des croyants et non des scientifiques. Le physicien et futurologue Theodore Modis écrivit ainsi :
« Ce que je veux dire est que Kurzweil[26] et les singularistes sont impliqués dans une sorte de para-science, qui diffère des vraies sciences en termes de méthodologie et de rigueur. Ils ont tendance à négliger les pratiques scientifiques rigoureuses telles que se concentrer sur les lois naturelles, donner des définitions précises, vérifier les données méticuleusement, et estimer les incertitudes. [...] Kurzweil et les singularistes sont plus des croyants que des scientifiques. »

— Theodore Modis, *Technological Forecasting & Social Change (2006)*

Il est vrai que le concept de la singularité technologique n'est pas démontré et qu'il contient de nombreuses fragilités comme le fait que les besoins en énergie, la limite quantique ou les diverses perturbations naturelles envisageables, ne sont pas prises en considération. Toutefois, nous ne pouvons ignorer cette éventualité et ses conséquences éthiques. La création d'une intelligence artificielle supérieure à celle de l'homme aura très certainement pour conséquence de modifier le cours de notre évolution voir de l'avenir de l'espèce humaine ; mais si l'on ne peut ni empêcher le progrès ni voir dans le temps, il est à peu prés certain qu'il reste encore beaucoup à faire[27].

6.2 Objectifs des réseaux de neurones formels dans l'intelligence artificielle

6.2.1 Rappels sur les réseaux de neurones

A ce niveau de notre étude, un petit rappel de ce qui a été expliqué plus tôt permettra de mieux cerner l'usage des réseaux de neurones dans le monde de l'intelligence artificielle. Rappelons tout d'abord que si les réseaux de neurones sont souvent considérés comme étant une application de la famille de l'intelligence artificielle, ils sont avant tout un outil utilisé pour la perception et la classification. Leur utilisation au fil des années a permis de mettre en avant leur aptitude à la généralisation et à l'apprentissage.

En effet les réseaux de neurones ont la capacité d'apprendre de leurs expériences, il est par exemple possible de leur faire apprendre la forme d'une lettre comme « K », puis de lui soumettre différentes lettres manuscrites. Il sera alors capable avec un excellent taux de réussite, de retrouver tous les « K ». C'est son aptitude à la généralisation qui lui permet cette efficacité, le réseau va regarder à quoi chaque lettre perçue correspond le plus, c'est ainsi que le « K » appris plus tôt, ressortira d'avantage et sera donc sélectionné.

Toujours grâce à sa méthode de généralisation, un réseau de neurones est capable de trouver une solution alors même que les informations fournies sont en partie erronées, que cela soit lors de son apprentissage ou de sa mise en situation. Enfin, le réseau de neurones apprend lors de chacune de ses expériences, il devient donc meilleur avec le temps, à condition toutefois qu'il soit informé de la validité de ses réponses.

Ces facultés font des réseaux de neurones un excellent outil pour la perception et l'apprentissage, et si ils peuvent être utilisés indépendamment ils peuvent également être disposés en aval d'un agent intelligent[28].

[26] Raymond C. Kurzweil, auteur de plusieurs ouvrages sur la santé, l'intelligence artificielle, la prospective et la futurologie. Il est l'un des théoriciens du transhumanisme et de la singularité technologique.
[27] « *Nous ne pouvons voir qu'à une courte distance devant nous, mais nous voyons qu'il reste encore beaucoup à faire* » - Alan Turing, 1950
[28] Le terme « agent intelligent » sera à présent employé pour désigner les intelligences artificielles comprenant les parties capteur, décision et action.

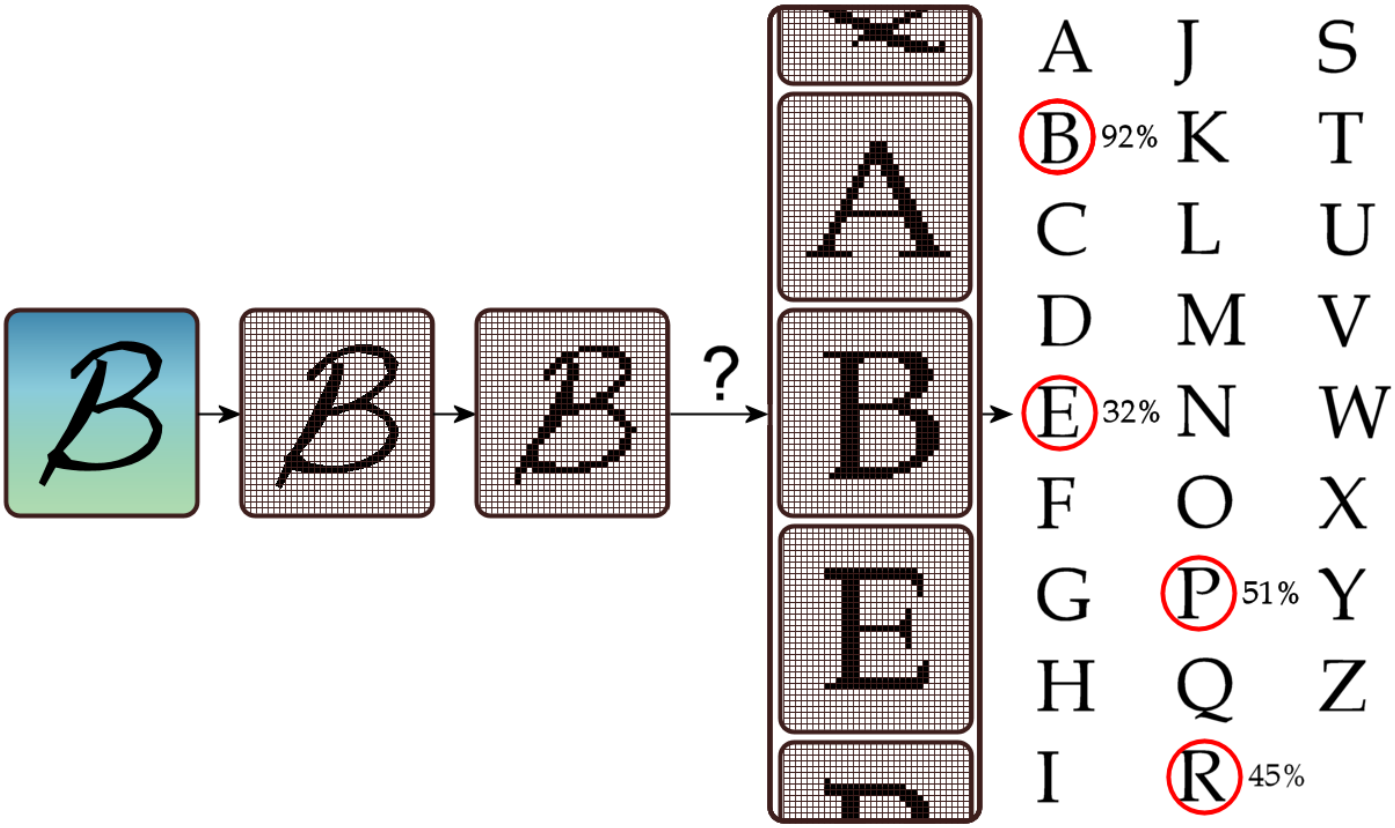


Fig4. *Principe de fonctionnement de la reconnaissance de caractères*

L'utilisation illustrée sur la figure 4 ci-dessus présente le fonctionnement d'un outil de reconnaissance de caractère[29] utilisant un réseau de neurones. Le caractère manuscrit est d'abord capté en entrée avant d'être échantillonné. Les pixels issus de l'échantillonnage sont ensuite rassemblés en groupes qui seront affectés aux différents neurones d'entrées d'un réseau. C'est à ce niveau qu'entrent en jeu les coefficients synaptiques, c'est eux qui, étalonnés lors de l'apprentissage, vont permettre d'activer ou non les différents neurones du réseau, déterminant si il s'agit plus probablement d'un « B », d'un « P » ou d'un « Y ».

6.2.2 Comment exploiter les réseaux de neurones ?

Nous savons que les réseaux de neurones possèdent deux aptitudes particulièrement utiles que sont la généralisation et l'apprentissage. La figure 3 illustre l'utilisation simultanée de la perception et de la généralisation au travers d'un outil de reconnaissance de caractères. Mais ce type d'utilisation peut-il être étendu à la reconnaissance de formes, d'environnements, de visages, de surfaces, d'ingrédients, de sons ? Tous ces éléments sont identifiables à partir d'informations qui pourraient être captée par un agent faisant usage d'un réseau de neurones, et ce dernier pourrait alors apprendre et généraliser, ou plutôt, approximer[30]. L'essentiel du problème se tournerait alors, d'une part, vers les capteurs : des capteurs fiables permettant d'obtenir une information exploitable par le réseau de neurones sont indispensables. Et d'autre part, vers le réseau de neurones lui-même : il existe en effet un vaste panel de réseaux de neurones, tous ayant leurs spécifications, leurs avantages, leurs inconvénients et leurs utilisations passées.

[29] Souvent utilisé dans la classification de colis postaux pour la lecture manuscrite du code postal.

[30] Ce terme, rarement utilisé, est pourtant parfaitement adapté, le réseau faisant une approximation entre ce qu'il capte et ce qu'il connaît, pour trouver une solution.

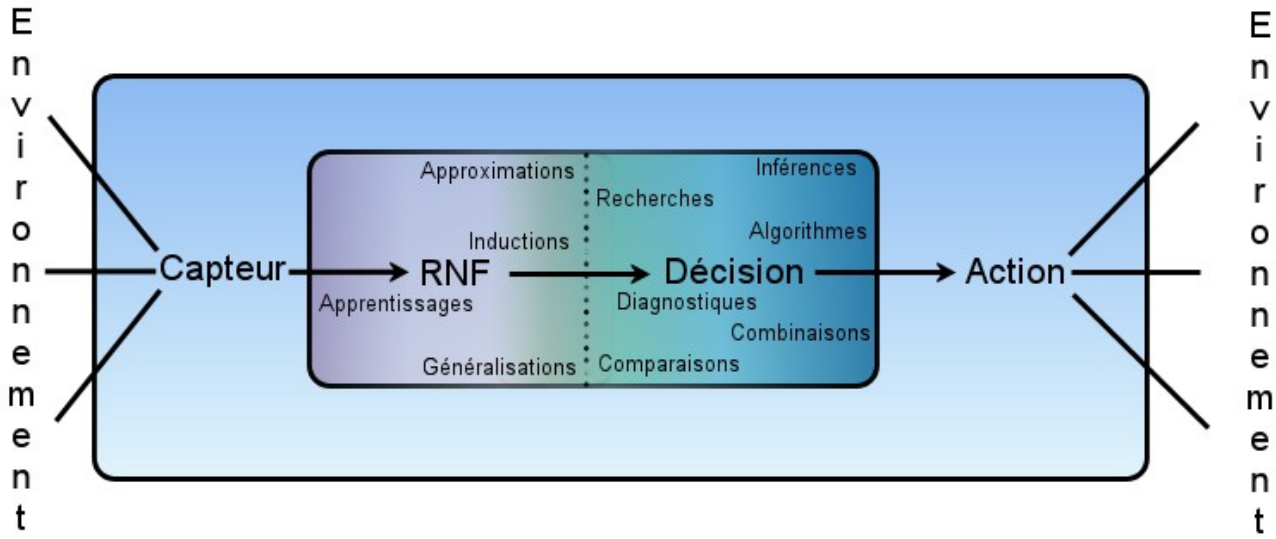


Fig5. *Principe de fonctionnement d'un agent intelligent exploitant un réseau de neurones formels*

La figure 5 illustre le principe de fonctionnement d'un agent utilisant doté d'un réseau de neurones, les informations captées sont directement envoyées au réseau qui est alors chargé de la perception et de l'approximation. L'information émise en sortie du réseau peut alors être traitée par la partie décisionnelle de l'agent avant d'être transformée en action.

7. Différentes formes

7.1 De réseaux de neurones formels

Nous avons fait un tour d'horizon des applications de l'intelligence artificielle, penchons nous à présent sur les principaux modèles de réseaux de neurones, voyons quels sont les réseaux qui ont fait leur preuves et pourquoi. Ne pouvant tous les utiliser pour nos expérimentations, cette partie de l'étude s'avère décisive pour effectuer une première sélection de réseaux.

Les réseaux de neurones diffèrent par plusieurs paramètres dont les principaux sont :

- La topologie des connexions entre les neurones du réseau.
- La fonction d'activation des neurones.
- Le mode d'apprentissage (supervisé ou non).
- L'algorithme d'apprentissage.

Il existe également des réseaux qui possèdent des particularités qui les différencient de tous les autres. Dans la présentation qui suit, les réseaux seront d'abord classés selon leur mode d'apprentissage puis selon l'algorithme d'apprentissage qu'ils utilisent.

7.1.1 Réseaux à apprentissage supervisé sans rétro propagation

- **7.1.1.1 Le Perceptron**

Le Perceptron, inventé en 1957 par Franck Rosenblatt, est considéré comme étant le réseau de neurones le plus simple. D'autre part, il est la première application reconnue du principe des réseaux neuronaux introduits par Pitts et McCulloch en 1943.

Il existe plusieurs types de Perceptron, toutefois sous sa version la plus simple, il est conçu à partir d'une seule couche constituée d'un unique neurone connecté à *n* entrées :

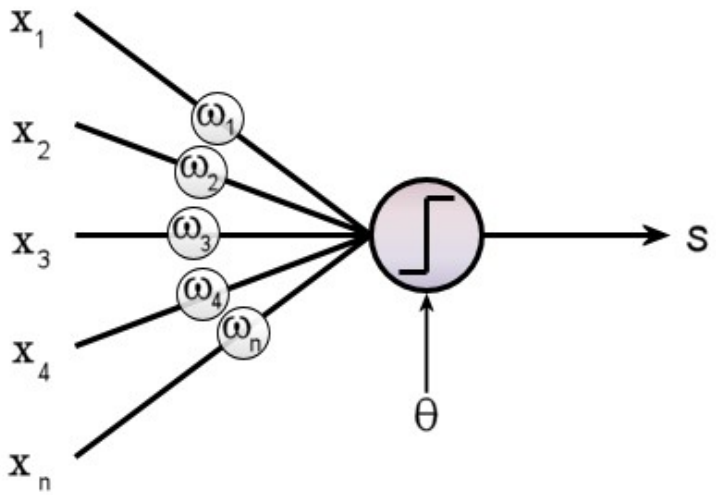


Fig6. *Perceptron simplifié à fonction d'activation de Heaviside*

Le Perceptron utilise la fonction de Heaviside[31] comme fonction d'activation, il calcule sa sortie de la façon suivante :

$$S = \begin{cases} 1 & if \sum_{i=1}^{n} \omega_i - x_i \geq \theta \\ 0 & if \sum_{i=1}^{n} \omega_i - x_i < \theta \end{cases}$$

Avant toute utilisation d'un réseau de neurone, il est nécessaire de l'entraîner, cet entraînement passe par une phase d'apprentissage. Durant cette phase d'apprentissage, l'utilisateur va soumettre des exemples au Perceptron, associé à des réponses. Imaginons par exemple que nous désirions enseigner à notre Perceptron la capacité à dissocier un chiffre pair d'un chiffre impair, nous procéderions alors en lui envoyant une série de valeurs paire et impaires, chacune associée à la réponse, ce qui permettrait au Perceptron d'ajuster ses coefficients synaptiques, ce qui est l'objectif de l'apprentissage.

Algorithme d'apprentissage :

Considérons une matrice M que l'on désire enseigner au Perceptron :

M = {(000,0),(001,1),(010,0),(0000011,1),(100,0),(101,1),(110,0),(111,1)}
Chaque chiffre pair étant noté 0, et impair, 1.

Initialisation des poids ω_i à des valeurs quelconques.

Initialisation d'un paramètre t à 0.

Répéter

Présenter chaque tableau m de la matrice M au Perceptron

Soient T la sortie désirée et S la sortie calculée par le Perceptron :

Si T ≠ S **Alors**
Modifier les poids :
$\omega_i(t+1) = \omega_i(t) + (T - S)E_i$
Avec E les entrées du réseau
Fin Si

t = t + 1

Jusqu'à ce que tous les M ai été entièrement présenté au Perceptron sans qu'il n'y ai eu de modifications de ce dernier

[31] Fonction échelon.

Si S = 0 et T = 1, alors le Perceptron n'a pas suffisament pris en compte les neurones actifs en entrée, dans ce cas : $\omega_i(t+1) = \omega_i(t) + E_i$, l'algorithme ajoute la valeur des entrées aux coefficients synaptiques. Dans le cas contraire, si S = 1 et T = 0, alors l'algorithme soustrait la valeur des entrées aux coefficients synaptiques : $\omega_i(t+1) = \omega_i(t) - E_i$.

Le tableau ci-dessous présente l'évolution de l'apprentissage d'une fonction logique OU :

t	$\omega_1(t)$	$\omega_2(t)$	$\omega_3(t)$	Entrée	S	T
0	1	1	-1	001	0	0
1	1	1	-1	011	0	1
2	1	2	0	101	1	1
3	1	2	0	111	1	1
4	1	2	0	001	0	0
5	1	2	0	011	1	1

On peut observer l'évolution des coefficients synaptiques tant que S et T ne sont pas égaux.
La principale limitation du Perceptron a été démontrée par Marvin Lee Minsky en 1969, et se situe dans son incapacité à résoudre des problèmes non linéaires. Le Perceptron ne peut donc pas résoudre de problème faisant par exemple usage d'une fonction XOR.

- **7.1.1.2 Réseau ADALINE - Adaptive Linear Neuron**

Le modèle ADALINE est identique au Perceptron, il n'en diffère que par sa fonction d'activation qui est une fonction linéaire en lieu et place de la fonction échelon utilisée sur le Perceptron. Les réseaux de neurones de type ADALINE utilisent la méthode des moindres carrés, ce qui permet de réduire considérablement les parasites reçus en entrée ; ils sont d'ailleurs employés pour la réduction du bruit en télécommunication **[VER 09]**.

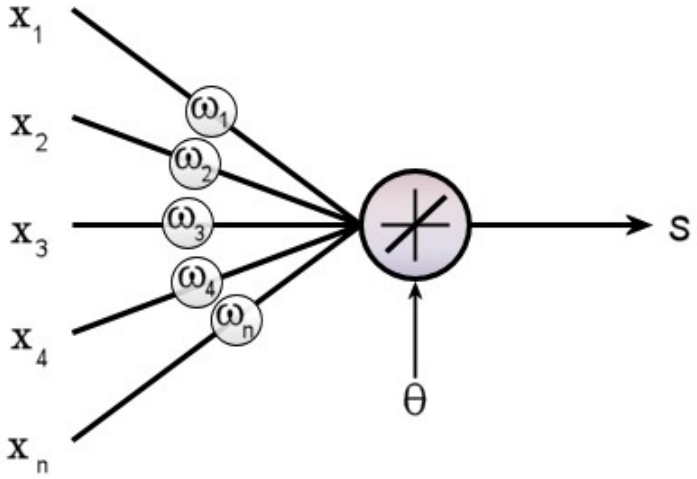


Fig7. *Réseau ADALINE à fonction d'activation linéaire*

Un réseau de type ADALINE réalise une somme pondérée des valeurs qu'il reçoit en entrée, à laquelle il rajoute la valeur de seuil θ, une fonction de transfert linéaire est ensuite utilisée pour l'activation du neurone. Lors de l'apprentissage, les poids sont modifiés en utilisant la loi de Widrow-Hoff[32].

[32] Les réseaux de neurones de type ADALINE ne sont pas adaptés aux applications que nous souhaitons en faire, nous ne nous attarderons donc pas sur leur mode de fonctionnement, toutefois, de nombreux ouvrages notamment en traitement des signaux détaillent leur fonctionnement. Nous suggérons leur consultation aux lecteurs intéressés **[MOK 98]**.

- **7.1.1.3 Associative Reward-Penalty**

Proposé en 1985, le réseau associative reward-penalty ou réseau ARP, utilise le principe de fonctionnement de l'apprentissage par renforcement. Ce qui signifie que le réseau reçoit une récompense ou une pénalité selon la justesse de son information de sortie. Toutefois, une pénalité n'indique pas la nature de l'erreur, il est donc nécessaire d'employer un réseau ayant une importante part d'aléatoire permettant de mieux scruter l'ensemble des solutions **[HAS 95]**.

Il existe plusieurs cas d'apprentissage par renforcement envisageables pour notre réseau :

- Dans le premier cas on considère que le réseau doit apprendre une sortie déterminée, selon ses entrées.

- Dans le second cas, l'environnement est stochastique[33]. Nous n'avons dans ce cas qu'une probabilité aléatoire d'obtenir la sortie attendue, et donc, un signal positif de renforcement.

- Dans le troisième et dernier cas, l'environnement de l'application est de type complexe, c'est-à-dire qu'il est soumis des règles arbitraires survenant dans le temps. Il ne permet pas de déterminer toutes les solutions proposées par le réseau méritent une récompense ou une pénalité. C'est notamment le cas dans les processus stratégique, où seule la fin de la totalité des opération permet d'aboutir à une situation positive ou négative. C'est le cas lors d'une partie d'échec, où seul l'échec et mat peut être une récompense ou une pénalité.

Un réseau ARP n'est pas capable de gérer ce dernier cas il nécessite impérativement l'utilisation d'une sortie de nature stochastique ainsi qu'une règle d'apprentissage particulière. L'architecture du réseau est choisie en fonction du problème à résoudre **[HER 91]**.

Pour construire la règle d'apprentissage, il faut calculer la différence entre la valeur de sortie ciblée γ et la valeur moyenne des sorties δ . Cette moyenne peut bien sûr être obtenue en effectuant plusieurs cycles.

Les coefficients synaptiques sont alors déterminés via la formule suivante :

$$\Delta w_{j,i} = \begin{cases} \eta^{+}[S_i^{\mu} - \langle S_i^{\mu} \rangle] x_j & \text{si } r^{\mu} = 1 \text{ (récompense)} \\ \eta^{-}[-S_i^{\mu} - \langle S_i^{\mu} \rangle] x_j & \text{si } r^{\mu} = -1 \text{ (pénalité)} \end{cases}$$

Cette formule ne permet que la mise à jour des poids synaptiques conduisant aux cellules de sortie (les dernières liaisons du réseau). Les autres poids devant être mis à jour à partir de règles adaptées à la topologie du réseau. Enfin, Selon la nature du signal de renforcement, le coefficient d'apprentissage peut être différent.

Les réseaux ARP n'ont pas de secteur d'application spécifique. Ils ne représentent en fait qu'un modèle d'apprentissage applicable à une majorité des réseaux de neurones.

- **7.1.1.4 Cascade-correlation [FAH 91]**

[33] Un environnement stochastique est un environnement qui comporte une variable aléatoire. Dans notre cas, cela signifie qu'aucune stratégie n'a été mises en place pour lier les entrées aux sorties, ou inversement.

Un réseau de neurones *Cascade-correlation* (CasCor) est une architecture à apprentissage supervisé créée par Scott Fahlman en 1991. La particularité de ce réseau réside en son aptitude à concevoir lui-même sa topologie. En effet, un réseau CasCor débute avec une architecture minimaliste et génère automatique des neurones cachés, tout en calculant leur coefficient synaptique. Chaque nouveau neurone participe alors à son tour, à l'évolution du réseau. Cette méthode permet d'éviter d'utiliser la rétropropagation **[RUM 86]** du gradient de l'erreur, qui est un processus extrêmement lent de la phase d'apprentissage.

Un réseau CasCor combine deux éléments clefs : le premier est bien entendu ce principe d'auto construction de la topologie du réseau, où les neurones sont ajoutés un par un, et où leur poids n'est plus modifié dès lors qu'ils sont apparus. Le second est l'algorithme d'apprentissage qui va concevoir ces nouveaux neurones, l'objectif étant de maximiser la corrélation entre la sortie des nouveaux neurones et le gradient de l'erreur que l'on souhaite éliminer.

Lorsqu'il vient d'être créé, un réseau CasCor est constitué d'un nombre variable d'entrées et de sorties ; chaque sortie étant connectée à un neurone. Il y a donc au départ, autant de neurones que de sorties. Le nombre d'entrées et de sorties est décidé par l'utilisateur selon l'application qu'il souhaite faire du réseau. Chaque entrée est connectée à chaque neurone de sortie. Dans la version originale du CasCor, une valeur de seuil fixée à +1 est également reliée aux neurones de sortie, ces derniers peuvent effectuer la somme pondérée des signaux reçus en entrée ou utiliser une fonction d'activation plus complexe. Ce choix revient à l'utilisateur du réseau.

Les neurones cachés sont ajoutés un par un, et chacun d'eux a ses entrées de connectées à l'ensemble des entrées du réseau ainsi qu'à la totalité des neurones déjà créés. Ses sorties sont reliées à la totalité des neurones de sorties du réseau. Le coefficient synaptique des neurones cachés est déterminé lors de leur création et ne varie plus ensuite, seuls les coefficients synaptiques des neurones de sortie sont constamment recalculés.

Chaque nouveau neurone ajouté au réseau y forme une nouvelle couche. L'algorithme d'apprentissage débute sans aucun neurone caché, seuls les neurones de sorties et leur poids sont entraînés en utilisant l'algorithme d'apprentissage de notre choix. Fahlman suggère l'utilisation de l'algorithme de Widrow-Hoff ou celui du Perceptron, tout en précisant qu'il n'est pas impératif de ce limiter à ces deux choix. L'apprentissage finira par atteindre une limite, où la variation des coefficients sera proche de zéro ; le réseau est alors lancé afin de mesurer l'erreur entre la sortie ciblée et la sortie effective. Si la performance du réseau est satisfaisante alors son apprentissage est interrompu.

Sinon, cela signifie qu'il subsiste un écart trop important entre la sortie ciblée et la sortie effective. Le réseau va essayer de réduire cet écart en ajoutant un neurone caché au réseau. Comme indiqué précédemment, ce neurone sera connecté à toute les entrées et tous les neurones cachés déjà présent dans le réseau, et ses sorties sont reliées à la totalité des neurones de sortie.

Enfin, son coefficient synaptique est fixé lors de sa création, et ne changera plus. Les trois figures suivantes illustrent le principe de création d'une topologie de réseau CasCor.

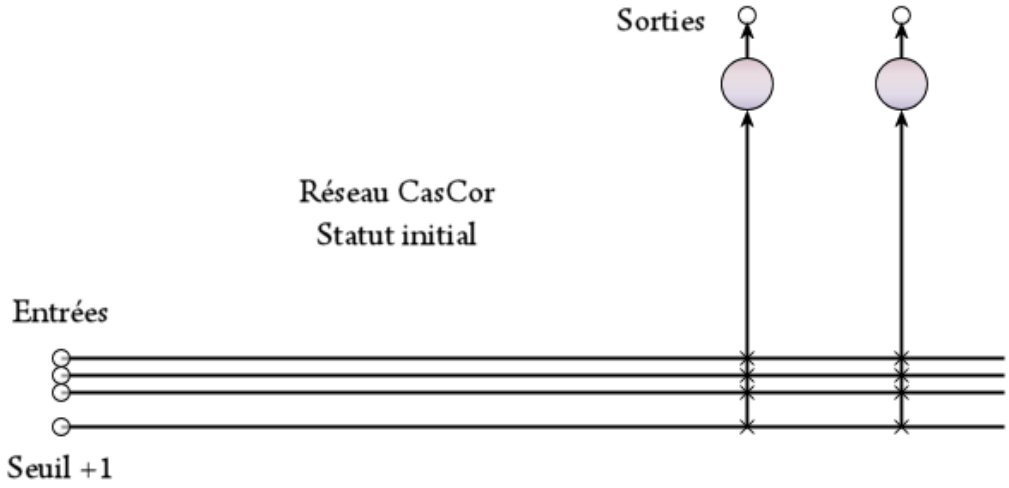


Fig8. *Réseau CasCor générique – statut initial sans neurones cachés*

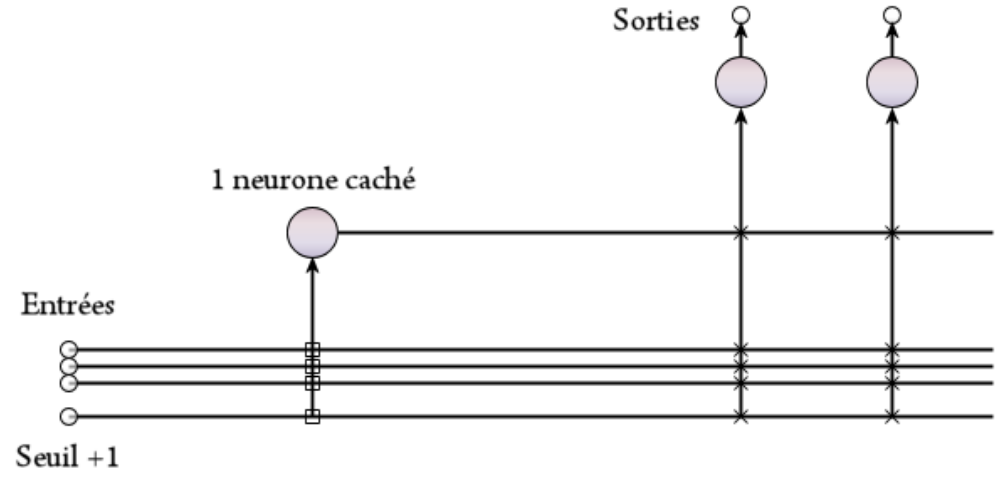


Fig9. *Réseau CasCor générique – Avec un premier neurone caché*

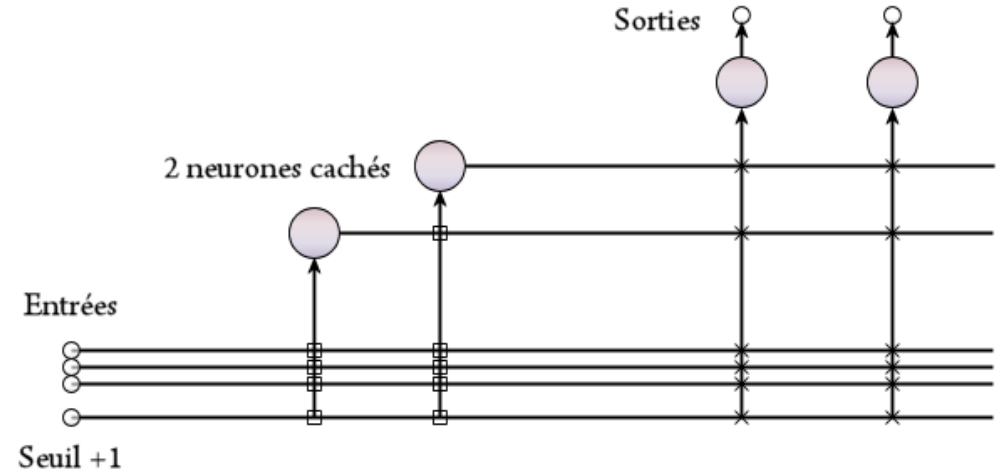


Fig10. *Réseau CasCor générique – Avec deux neurones cachés*

Les connexions marquées d'une croix représentent les coefficients synaptiques qui sont récursivement entraînés. Celles marquées d'un carré, représentent les coefficients synaptiques fixes. Voyons comment ceux-ci sont déterminés en même temps que la procédure de création d'un nouveau neurone.

Un neurone candidat est créé et est relié à la totalité des entrées, toutefois, sa sortie reste en l'air. L'algorithme d'apprentissage est alors lancé dans le but de déterminer le coefficient synaptique du neurone candidat. L'objectif étant de maximiser S, la somme de tous les neurones de sortie o de la magnitude de corrélation entre V la valeur du neurone candidat et Eo écart résiduel observé sur le neurone o.
Nous calculons S à partir de la formule suivante :

$$S = \sum_{o} \left| \sum_{p} \left(V_p - \overline{V}\right)\left(E_{p,o} - \overline{E}_o\right) \right|$$

Avec :
o la sortie du réseau où l'erreur est mesurée.
p le motif utilisé pour l'apprentissage du réseau.
$\overline{V}$ et $\overline{E}_o$ les valeurs moyennes de V et E_o établit à partir de la totalité des motifs.

Afin de maximiser S, nous devons calculer sa dérivée partielle par rapport à chacun des poids des entrées du neurone candidat ω_i :

$$\delta S / \delta \omega_i = \sum_{p,o} \sigma_o \left(E_{p,o} - \overline{E}_o\right) f_p' I_{i,p}$$

Avec :
σ_o le signe de la corrélation entre o et V, la valeur du neurone candidat.
f_p' la dérivée pour motif p de la fonction d'activation du neurone candidat.
$I_{i,p}$ est l'entrée que reçoit le candidat du neurone i pour le motif p.

Une fois la dérivée partielle de S calculée pour chacune des connexions du neurone candidat, nous pouvons procéder à la maximisation de S. Lorsque S atteint une asymptote et que sa progression est proche de zéro, la maximisation est interrompue, les coefficients synaptiques du neurone candidat sont fixés et il est alors est intégré au réseau comme neurone actif.

Si un neurone candidat possède une corrélation positive avec une erreur provenant d'un autre neurone, il va développer un coefficient synaptique négatif afin d'essayer d'annuler cette erreur. Si la corrélation est négative, alors le coefficient synaptique sera positif. Les poids d'un neurone étant différent à chaque entrée, un neurone peut développer un coefficient positif sur certaines entrées et un coefficient négatif sur d'autres.

Il est également possible d'utiliser une population de neurones candidats au lieu d'un candidat unique. Chacun d'eux étant initialisé avec des poids aléatoires. Etant donné qu'ils n'interagissent pas entre eux et qu'ils ne perturbent pas le résultat en tant que candidat, ils peuvent tous êtres entraînés en parallèle. Lorsque l'entraînement est terminé et qu'il n'y a plus de progrès observables, le meilleur candidat est activé et les autres sont supprimés. Cette méthode permet de s'assurer que le candidat ne reste pas bloqué et augmente ainsi la probabilité d'obtenir un neurone caché performant.

Tout en offrant un gain de temps considérable puisqu'il devient possible de calculer l'efficacité de plusieurs aires de poids en même temps (inutile de recalculer ce qui l'a déjà été pour le neurone candidat voisin). De plus il est possible de varier les fonctions d'activation de ces candidats. L'utilisation d'une population de candidats est probablement à privilégier en terme de gain de temps et de performances sur le réseau final.

- **7.1.1.5 Learning Vector Quantization et apprentissage compétitif [KOH 82]**

Les réseaux LVQ (Learning Vector Quantization) sont des réseaux hybrides, essentiellement utilisés pour la classification. Cette dernière ne répond pas à nos besoins liés à l'application dans les jeux vidéo, le fonctionnement détaillé des réseaux LVQ dépasse donc le cadre de ce mémoire. Toutefois, leur importance dans l'histoire des réseaux de neurones, puisqu'ils sont inspirés des réseaux SOM (Self Organizing Map) que nous aborderons en partie 7.1.3, nous impose leur présentation.

Comme il l'a été dit, les réseaux LVQ sont des réseaux hybrides à l'apprentissage partiellement supervisé. Ils font usage de l'apprentissage compétitif dont l'objectif est de permettre la classification automatique **[BOR07]** à savoir, le rassemblement d'éléments en différentes classes, selon des propriétés semblables à partir de critères choisis. Les réseaux à apprentissage compétitif fonctionnent de façon non supervisée, ce qui signifie qu'ils s'adaptent seuls à leur environnement. Nous reviendrons plus loin sur cette notion d'absence de supervision. Ce qu'il est nécessaire de savoir pour comprendre le fonctionnement des réseaux LVQ, c'est qu'en apprentissage compétitif, les neurones sont amenés à se déplacer par eux même pour correspondre aux prototypes soumis. Ainsi, les neurones d'un réseau LVQ, soumis à des prototypes, auront tendance à se regrouper auprès des prototype leur correspondant le plus ; c'est le principe de la classification. L'apprentissage compétitif obtiendra ce type de résultat :

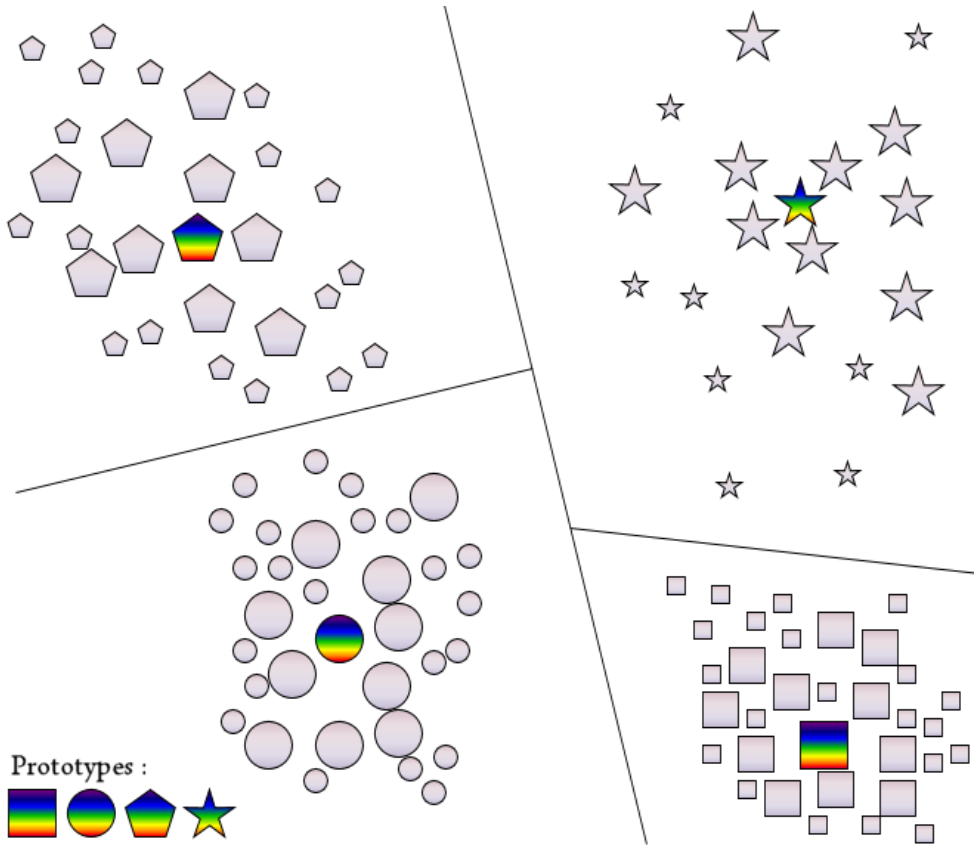


Fig11. *Classification des neurones par ressemblance*

Les neurones sont d'abords initialisés de façon aléatoires, puis, ils sont mis en compétition par rapport à chaque prototype[34] ; le meilleur des neurones est alors sélectionné et sa position est recalculée, les neurones proches du meilleur en terme de valeur (et donc de probable classification) verront leur position également re-calculée.

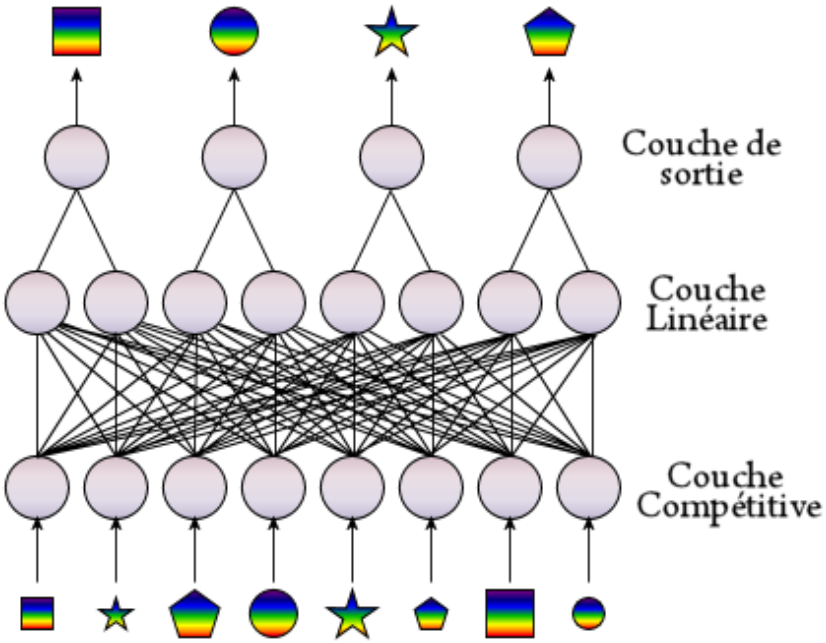


Fig12. *Présentation simplifiée d'un réseau LVQ pour la classification de formes*

La figure 15 est une représentation simplifiée d'un réseau LVQ. Si telle qu'elle est représentée ici la couche compétitive comporte autant de neurones que la couche linéaire, cela n'est pas nécessairement le cas. La couche compétitive peut comporter autant de neurones que d'éléments (dans notre cas des formes) à traiter.

C'est la couche linéaire qui prend les décisions, ses poids sont fixés avant l'apprentissage et permettent de déterminer à quelle forme appartient l'élément en entrée. La couche compétitive utilise la règle de Kohonen pour spécialiser chaque neurone afin qu'il représente un groupe bien particulier selon les points communs qui les rassemblent. L'idée étant de discrétiser l'espace de données en plusieurs zones et d'y disposer un vecteur prototype représentatif de cette zone.

Après une initialisation aléatoire des valeurs de chaque neurones on soumet une à une les données de l'environnements. Selon les valeurs des neurones, il y en a un qui répondra le mieux car sa valeur sera la plus proche de la valeur présentée. Ce neurone se verra alors modifié pour qu'il réponde encore mieux lors de la présentation d'une donnée de même nature que la précédente. On modifie alors également les neurones voisins du gagnant ce qui permet à toute la région de la carte autour du neurone gagnant de se spécialiser **[KOH 95]**.

Comme indiqué au début de cette partie, ce type de réseau fonctionnement d'une façon trés différente des autres réseaux, nous aurons l'occasion d'étudier de façon plus approfondie un réseau qualifié de carte auto-adaptative: le SOM (Self-Organizing Map).

[34] Egalement qualifié de vecteur prototype ou de « codebook vector » (par Kohonen)

- **7.1.1.6 Probabilistic Neural Network & General Regression Neural Network**

Les réseaux PNN (Probabilistic Neural Network) et GRNN (General Regression Neural Network) utilisent un principe de fonctionnement commun, employant la fonction de base radiale. Une fonction de base radiale (RBF) est une fonction ϕ symétrique autour d'un centre définit par :

$$\mu_j : \phi_j(x) = \phi\left(\left\|x - \mu_j\right\|\right)$$

Où $\|.\|$ est une norme **[BUH 03, VIE 04]**.

Les réseaux utilisant les fonctions de base radiales (RBF) sont des modèles connexionnistes simples à mettre et oeuvre et assez intelligibles, ils sont souvent utilisés pour la régression et la discrimination. D'un point de vue topologique, il s'agit de réseaux standard avec une fonction d'activation de type RBF :

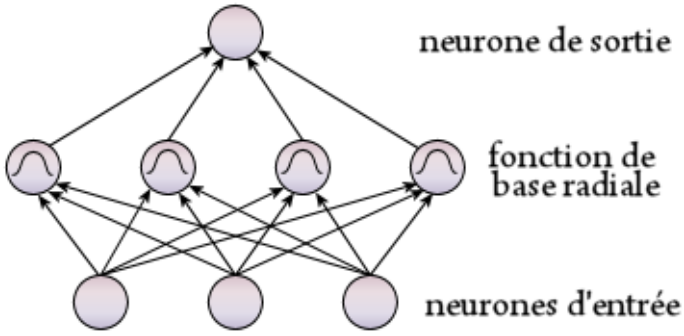


Fig13. *Réseau à fonction de base radiale*

Les réseaux PNN ou GRNN permettent d'estimer la nature d'un élément à partir de toutes les informations qu'ils peuvent obtenir de l'environnement : Imaginons un potager où se trouvent des plants de carottes, de radis, et de tomates. Chacun caractérisé sa position en X et Y et bien entendu, sa nature (carotte, radis ou tomate) :

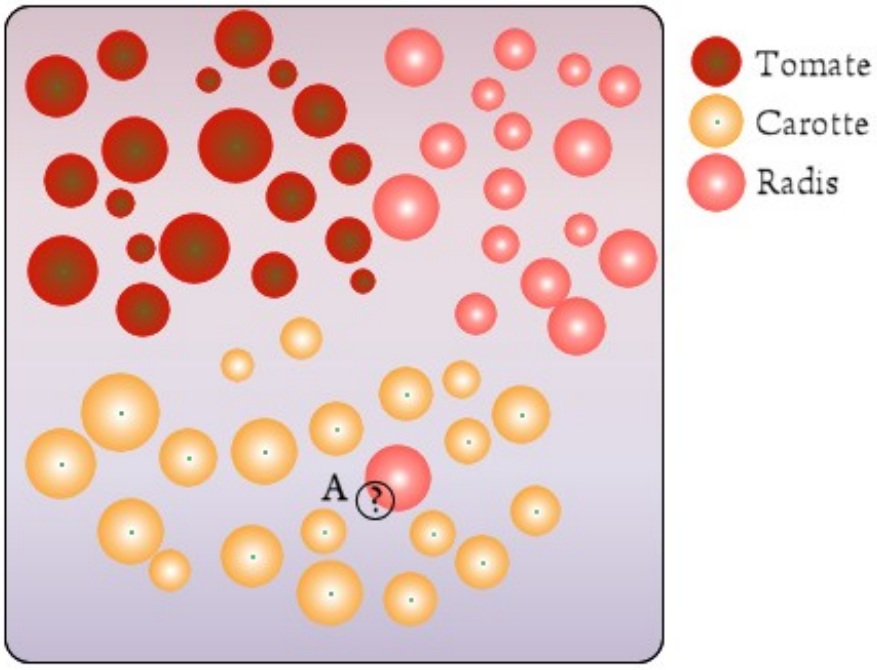


Fig14. *Exemple d'utilisation d'un PNN/GRNN*

Dans le potager représenté sur la figure 17, nous connaissons toutes les caractéristiques de chacun des plants, sauf du plant A dont nous n'avons que la position en X et Y. Dans un réseau de neurone standard, il y a de grandes chances pour que le plant A soit considéré comme étant un plan de tomate à cause de sa grande proximité avec l'un d'entre eux. Ceci malgré son positionnement au sein d'un groupe de carottes.

Un PNN prendra en considération la présence des autres carottes dans sa recherche de la nature du plant A. La distance est calculée à partir du plant A vers chacun des autres plants, et la fonction de base radiale est appliquée à ces distances afin de déterminer le poids pour chacun des plants :

$$w_i = RBF(D_{A,i})$$

Avec D la distance entre A et i, et w_i le poids de chaque lien entre A et i.

L'application de la fonction ressemble alors à ceci :

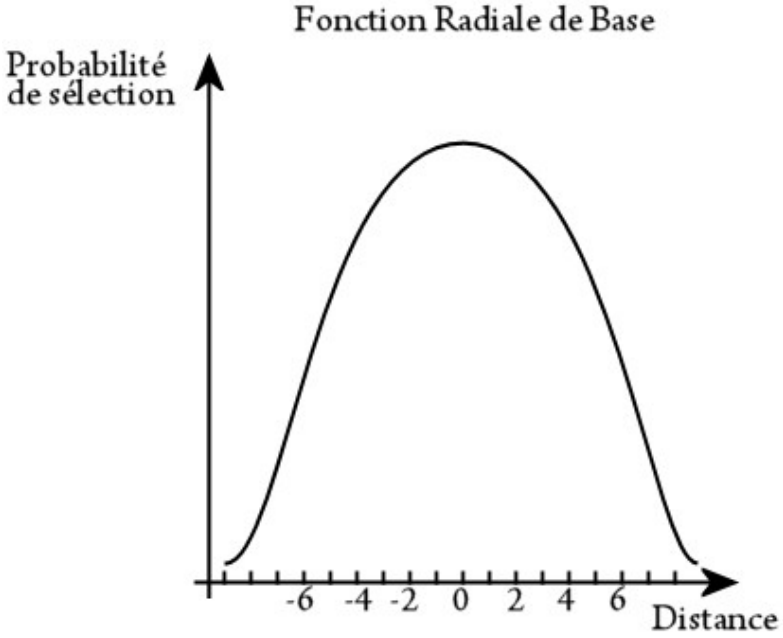


Fig15. *Fonction de Gauss – une fonction radiale de base*

En s'éloignant chaque plan diminue sa probabilité de sélection, et son poids sera moins important dans la détermination de la nature du plan A. L'un des paramètres du réseau étant bien entendu de pouvoir déterminer la sélectivité du filtre (très sélectif ou non).

L'architecture d'un PNN ou d'un GRNN se présente de la façon suivante :

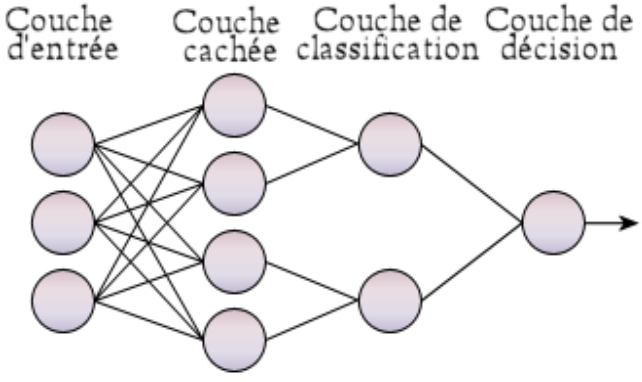


Fig16. *Architecture commune d'un PNN ou d'un GRNN*

Les PNN/GRNN possède quatre couches :

Une couche d'entrée qui possède autant de neurones que de variables par élément (dans notre cas, il y a quatres variables, qui sont la position en X, la position en Y et la nature du plant). Les informations sont mises en formes puis émises vers chacun des neurones de la couche cachée.

Une couche cachée où intervient la fonction radiale de base : après avoir calculé la distance entre le plant A un autre plant, la couche cachée applique la fonction radiale de base sur cette distance puis envoie le résultat à la couche de classification.

Une couche de classification différente si il s'agit d'un PNN ou d'un GRNN :

Dans le cas d'un PNN, il y a autant de neurones de classification que de catégories ou de motifs (dans notre cas, il s'agira de la nature du plant, nous devrions donc avoir 3 neurones dans la couche de classification). Chaque neurone fait la somme des poids des éléments de sa catégorie.

Dans le cas d'un GRNN, il n'y a que deux neurones dans la couche de classification, l'un est appelé le dénominateur de sommation, l'autre le numérateur de sommation. Le dénominateur fait la somme de tous les poids provenant des neurones cachés et le numérateur additionne les poids provenant des neurones cachés, multiplié par le poids ciblé.

Une couche de décision dont l'action est différente entre un PNN et un GRNN :

Dans le cas d'un PNN, la couche de décision compare les différents poids stockés dans chaque neurone de la couche de classification et le poids le plus élevé définit la nature de l'élément ciblé.

Pour le GRNN, la couche de decision divise la valeur accumulée par le numérateur par la valeur accumulée par le dénominateur. Le résultat devient la valeur de sortie.

- **7.1.1.7 Réseau de Hopfield [WIK 09f]**

Le réseau de Hopfield est un modèle de réseau de neurones récurrent dont les connexions sont symétriques et où mise à jour des neurones est asynchrone. Ce modèle a été conçu par le physicien John Hopfield en 1982, il est un des réseaux les plus célèbres car sa mise au point permis de relancer l'intérêt dans les réseaux de neurones qui s'était essoufflé durant les années 1970 suite à l'article de Marvin Minsky et Seymour Papert que nous avions évoqué plus tôt, dans la chronologie.

Dans ce modèle, chaque neurone est connecté à l'ensemble des autres neurones et il n'y a ainsi aucune différenciation possible entre les neurones d'entrée et ceux de sortie. Ce réseau fonctionne comme une mémoire associative non-linéaire et est capable de trouver un objet stocké à partir d'informations partielles ou bruitées de cet objet **[HOP 85]**. On retrouve l'utilisation des réseaux de Hopfield dans le stockage de connaissances mais aussi, parfois, dans la résolution de problèmes d'optimisation. Il n'est pas adapté à nos attentes en terme d'utilisation dans le cadre du jeu vidéo, mais la particularité de sa topologie et son importance dans l'histoire des réseaux de neurones, nous incite à nous pencher sur son fonctionnement.

Le réseau de Hopfield est l'un des rares réseaux de neurones à employer un mode d'apprentissage non-supervisé sans faire usage de la rétro-propagation.

Nous avons indiqué plus haut que le modèle de Hopfield est un réseau de neurones récurrent. Précisons donc qu'un réseau de neurones récurrent est un réseau où l'information est amenée à revenir en arrière : la topologie du réseau dispose d'une ou plusieurs boucles permettant à une information déjà traitée par un neurone de revenir sur celui-ci :

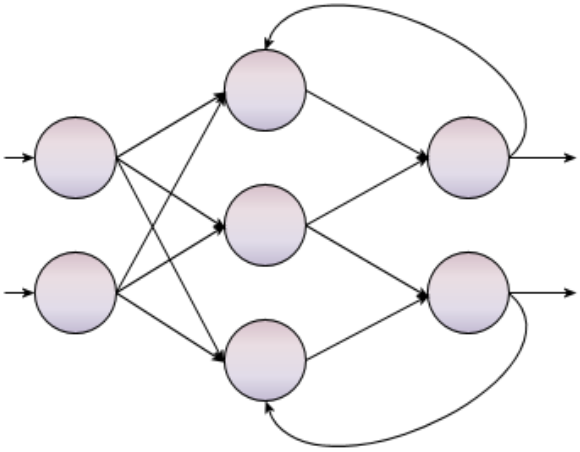


Fig17. *Exemple de réseau de neurones récurrent : les sorties bouclent sur la seconde couche*

Pour sa part, le réseau de Hopfield est généralement représenté, sous sa forme la plus simple, de la façon suivante :

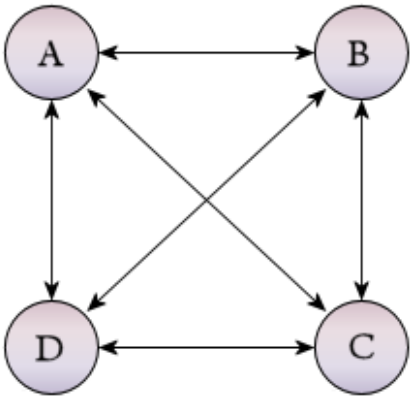


Fig18. *Réseau de Hopfield*

On remarque que les neurones sont tous connectés entre eux et que les connexions sont symétriques. La valeur d'entrée d'un neurone n se calcule alors via la formule suivante :

$$V_n = \sum_j \omega_{n,j} J_j$$

Où $\omega_{n,j}$ est le poids de la connexion entre le neurone n et le neurone j,

Et J_j l'état du neurone j, qui varie entre -1 et 1 (état binaire).

L'état du réseau peut ainsi être représenté par un mot de J bits où J est le nombre de neurones du réseau. Les réseaux de Hopfield sont séquencés par une horloge et il existe plusieurs méthodes de mise à jour des neurones du réseau :

- Le mode stochastique utilisé dans le réseau original de Hopfield où le neurone à se mettre à jour est choisi de façon aléatoire à chaque impulsion de l'horloge **[HOP 82, HOP 84]**.

- Le mode synchrone où tous les neurones sont mis à jour simultanément.

- Le mode asynchrone où les neurones sont mis à jour un à un, à chaque top de l'horloge.

Le calcul du nouvel état d'un neurone se calcule alors de la façon suivante :

$$V_n(t+1) = \begin{cases} 1 \; si \quad \sum_j \omega_{n,j} J_j > 0 \\ 0 \end{cases}$$

L'apprentissage dans un réseau d'Hopfield consiste à faire en sorte que chacune des informations à mémoriser soit :

- Un état stable du réseau.

- Un état pouvant être retrouvé à partir d'états légèrement différents ou bruités.

Pour calculer les poids, on utilise l'apprentissage hebbien, inspiré de la loi de Hebb[35] :

$$\omega_{n,j} = \frac{1}{d} \sum_{k=1}^{p} x_n^k x_j^k$$

Où $\omega_{n,j}$ est toujours le poids de la connexion entre les neurones n et j.

d est la dimension du vecteur d'entrée.

p le nombre de motif d'entraînement.

x_n^k la kième entrée du neurone n.

L'efficacité de la connexion entre un neurone n et un neurone j est donc augmentée de 1 si ces deux neurones sont dans le même état $(-1 \times -1 = 1)$ et $(1 \times 1 = 1)$ et diminuée de 1 si ces neurones sont dans des états différents $(1 \times -1 = -1)$ ou inversement.

Comme indiqué plus tôt et contrairement à la majorité des réseaux de neurones, les réseaux de Hopfield ne sont pas utilisés pour résoudre des problèmes mais servent de mémoire adressable. Le nombre de configurations possibles pour un réseau est égal à 2^J, où J est le nombre de neurones du réseau. Pour un réseau comprenant 2 neurones, il existe donc 4 configurations possibles ; Avec 3 neurones, on peut stocker 8 configurations. Le réseau de la figure 18 contient 4 neurones et peut donc stocker 16 configurations différentes.

[35] Créée par Donald Hebb en 1949 et observée dans le cerveau humain en 1973

7.1.2 Réseaux à apprentissage supervisé avec rétro propagation

La technique de rétropropagation (*Backpropagation* en anglais) est une méthode qui permet de calculer le gradient de l'erreur pour chaque neurone du réseau, de la dernière couche vers la première. On appelle souvent technique de rétropropagation du gradient l'algorithme classique de correction des erreurs basé sur le calcul du gradient grâce à la rétropropagation, mais cela n'est pas toujours le cas. La correction des erreurs peut se faire selon d'autres méthodes, comme le calcul de la dérivée seconde ou partielle que l'on emploie dans les réseaux CasCor étudiés précédemment ou encore, en évaluant la responsabilité des neurones situés en amont comme nous allons le découvrir avec les réseaux ALN. Le plus souvent, dans le cas des réseaux de neurones, la méthode de correction d'erreur agit en corrigeant de manière significative les coefficients synaptiques qui contribuent à engendrer une erreur importante tout en pondérant également les neurones générant une erreur moins conséquente.

Ces méthodes de rétropropagation du gradient firent l'objet d'études dès 1975, puis en 1985 mais ce sont les travaux de Rumelhart, Hinton & Williams en 1986 qui suscitèrent le véritable début de l'engouement pour cette méthode avec sa première application dans le Perceptron muticouche. C'est grâce à la rétropropagation du gradient que les réseaux de neurones ont de nouveau suscité l'intérêt de la communauté scientifique.

Dans le cas d'un apprentissage supervisé, des données sont présentées à l'entrée du réseau de neurones et celui-ci produit des sorties. La valeur des sorties dépend des paramètres liés à la structure du réseau : topologie, fonctions d'agrégation et d'activation ainsi que les coefficient synaptiques. Si ce sont généralement ces derniers qui sont modifiés, cela n'est pas toujours le cas : la modification peut également intervenir au niveau des fonctions d'agrégation ou d'activation, mais plus rarement sur la topologie.

Les différences entre ces sorties et les sorties ciblées forment des erreurs qui seront corrigées via la rétropropagation. La manière de quantifier cette erreur peut varier selon le type d'apprentissage employé. En appliquant récursivement cette étape, l'erreur tend à diminuer et le réseau offre ainsi une meilleure prédiction. Il se peut toutefois qu'il ne parvienne pas à échapper à un minimum local, c'est pourquoi on ajoute en général un terme d'inertie à la formule de la rétropropagation pour aider la descente de gradient à sortir de ces minimums locaux.

- **7.1.2.1 Le Perceptron multicouche [PAR 04]**

Algorithme de rétropropagation :

1. Présentation d'un motif d'entraînement au réseau.
2. Comparer la sortie du réseau avec la sortie ciblée, calculer l'erreur en sortie de chaque neurone du réseau.
3. Pour chaque neurone calculer ce que la sortie aurait dûe être ainsi qu'un facteur d'échelle permettant de définir l'augmentation ou la diminution nécessaire pour obtenir la sortie attendue. Cette information devient l'erreur locale.
4. Ajuster les coefficients synaptiques à l'erreur locale la plus basse.
5. Attribuer un blâme pour l'erreur locale à tous les neurones en amont, assignant une plus grande responsabilité aux neurones connectés avec un plus grand coefficient synaptique.
6. Recommencer à partir de l'étape 3 sur les neurones en amont en utilisant le blâme comme erreur.

Le Perceptron multicouche est un réseau orienté de neurones artificiels organisée en couches et où l'information ne circule que dans un seul sens : de la couche d'entrée vers la couche de sortie. La couche d'entrée représente toujours une couche virtuelle associée aux entrées du système et ne contient donc aucun neurone. Les couches suivantes, elles, sont constituées de neurones. Les sorties des neurones de la dernière couche correspondent toujours aux sorties du système. Un Perceptron multicouche peut posséder un nombre de couches quelconque et chaque couche peut comporter un nombre de neurones (ou d'entrées, si il s'agit de la couche d'entrée) également quelconque.

Les neurones sont reliés entre eux par des connexions pondérées. Ce sont les poids de ces connexions qui gouvernent le fonctionnement du réseau et "programment" une application de l'espace des entrées vers l'espace des sorties à l'aide d'une transformation non linéaire. La création d'un Perceptron multicouche pour résoudre un problème donné passe donc par l'inférence de la meilleure application possible telle que définie par un ensemble de données d'apprentissage constituées de paires de vecteurs d'entrées et de sorties désirées. Cette inférence peut se faire, entre autre, par un algorithme rétropropagation.

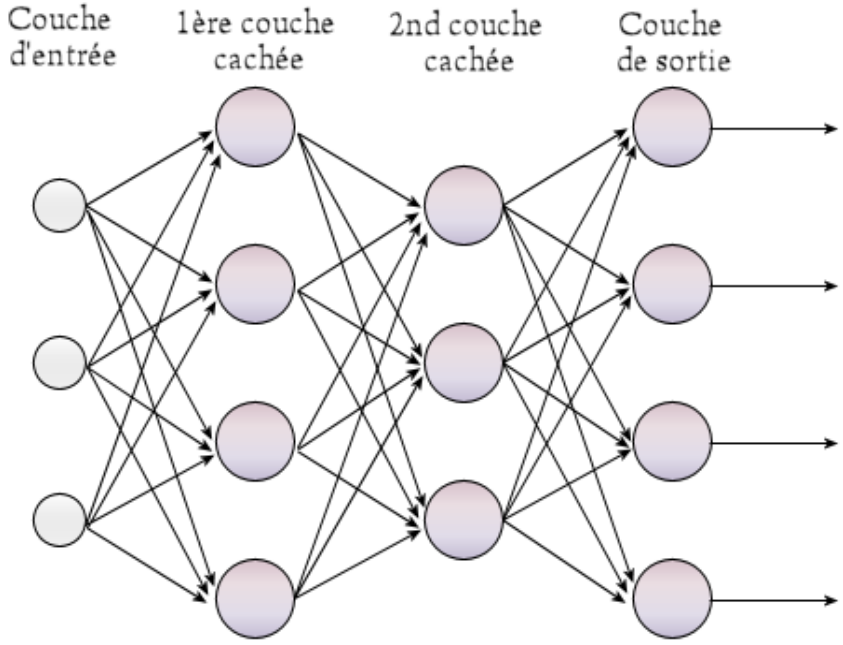


Fig19. *Perceptron multicouche à deux couches cachées*

Soit $\overrightarrow{X}(n)$ et $S(n)$, correspondant aux entrée fournies et aux sorties ciblées du réseau avec n désignant la nième donnée d'apprentissage. L'algorithme de rétropropagation consiste à mesurer l'erreur entre les sorties ciblées $S(n)$ et les sorties observées $Y(n)$ résultat de la propagation vers l'avant des informations d'entrées $\overrightarrow{X}(n)$ et à rétro propager cette erreur des sorties vers les entrées. L'algorithme de rétropropagation procède donc à l'adaptation des poids neurone par neurone en commençant par la couche de sortie :

Soit $\varsigma_i(n)$ l'erreur observée pour le neurone de sortie i et le motif d'apprentissage n :

$$\theta_i(n) = S_i(n) - Y_i(n)$$

Avec $S_j(n)$ les sorties ciblées et $Y_j(n)$ les sorties observées pour le neurone j.

L'objectif de l'algorithme est d'adapter les poids des connexions du réseau de manière à minimiser la somme des erreurs sur tous les neurones de sortie, la sortie $Y_j(n)$ du neurone j est définie par :

$$Y_j(n) = \varphi\left[v_j(n)\right] = \varphi\left[\sum_{i=0}^{r} \omega_{ji}(n) Y_i(n)\right]$$

Avec $\varphi[.]$ la fonction d'activation du neurone, $v_j(n)$ la somme pondérée des entrées du neurone j, ω_{ji} le coefficient synaptique entre le neurone i de la couche précédente et le neurone j de la couche courante et $Y_i(n)$ la sortie de ce même neurone i.

On considère que la couche précédente contient r neurones, que ω_{jo} correspond au biais du neurone j et que $Y_0(n) = -1$.

Pour corriger l'erreur observée, il faut modifier le coefficient $\omega_{ji}(n)$ dans le sens opposé au gradient de l'erreur calculé par la dérivée partielle suivante :

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)}$$

L'objectif étant d'atteindre le minimum local en modifiant le coefficient synaptique dans sens ou dans l'autre. On exprime la variation de poids par la formule suivante :

$$\Delta \omega_{ij}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ji}(n)}$$

Avec $0 \leq \eta \leq 1$ représentant le taux d'apprentissage de l'algorithme.

En utilisant la règle de chaînage des dérivées partielles[36] on obtient[37] :

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = -e_j(n) y_j(n)\left[1 - y_j(n)\right] y_i(n)$$

Ce qui, associé à la règle du « Delta » exprimant la variation de poids, donne :

$$\Delta \omega_{ji}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ji}(n)} = -\eta \ \delta_j(n) y_i(n)$$

$$\text{Avec : } \delta_j(n) = e_j(n) y_j(n)\left[1 - y_j(n)\right]$$

Ce qui correspond au gradient local de la couche de sortie. Hors, le réseau dispose encore de deux couches cachées sur lesquels la rétropropagation doit également s'effectuer, considérons donc à présent la seconde couche cachée du réseau (toutes les couchées se traitant ensuite de la même façon). L'objectif restant le même, c'est-à-dire, d'adapter les coefficients des connexions entre la couche précédente et la couche cou-

[36] Règle qui dit que $\frac{\partial f(y)}{\partial x} = \frac{\partial f(y)}{\partial y} \cdot \frac{\partial y}{\partial x}$

[37] Voir **[PAR 04]** pour la démonstration complète

rante. Les indices i et j désignent toujours respectivement un neurone de la couche précédente et un neurone de la couche courante. Enfin, l'indice k désigne un neurone de la couche suivante.

Si nous réutilisons la règle des dérivées partielles de l'erreur totale $E(n)$ nous devons réévaluer le terme de la dérivée partielle selon la sortie observée de la couche précédente :

$$\frac{\partial E(n)}{\partial y_i(n)} = \sum_{k \in c} \delta_k(n)\omega_{kj}(n)$$

Ce qui nous amène alors à formuler :

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = -y_j(n)\left[1 - y_j(n)\right]\left[\sum_{k \in c} \delta_k(n)\omega_{kj}(n)\right] y_i(n)$$

Et : $\Delta \omega_{ji}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ji}(n)} = -\eta \ \delta_j(n) y_i(n)$

Avec : $\delta_j(n) = y_j(n)\left[1 - y_j(n)\right]\sum_{k \in c} \delta_k(n)\omega_{kj}(n)$

Ces deux équations sont utilisables pour la totalité des couches cachées, quel que soit leur nombre, toutefois, pour la première couché cachée, il faut remplacer $Y_i(n)$ par $X_i(n)$.

La procédure de rétropropagation peut se résumer à la série d'étapes suivantes :

1. Initialiser tous les poids à de petites valeurs aléatoires dans l'intervalle [−0.5, 0.5].
2. Normaliser les données d'entraînement.
3. Permuter aléatoirement les données d'entraînement.
4. Pour chaque donnée d'entraînement n :
   (a) Calculer les sorties observées en propageant les entrées vers l'avant ;
   (b) Ajuster les poids en rétro propageant l'erreur observée :

$$\omega_{ji}(n) = \omega_{ji}(n-1) + \Delta\omega_{ji}(n) = \omega_{ji}(n-1) + \eta \ \delta_j(n) y_i(n)$$

   Où le gradient local est défini par (1) s'il s'agit d'une couche de sortie et par (2) si c'est une couche cachée:

$$\delta_j(n) = \begin{cases} e_j(n) y_j(n)\left[1 - y_j(n)\right] & (1) \\ y_j(n)\left[1 - y_j(n)\right]\sum_{k \in c} \delta_k(n)\omega_{kj}(n) & (2) \end{cases}$$

5. Répéter les étapes 3 et 4 jusqu'à un nombre maximum d'itérations ou jusqu'à ce que l'erreur soit en dessous d'un certain seuil.

- **7.1.2.2 Adaptive Logic Network**

Un « Réseau logique adaptatif » ou Adaptive Logic Network (ALN) est une forme particulière de Perceptron multicouches. Il peut être décrit comme un arbre binaire, où les neurones effectuent des opérations logiques.

Les entrées d'un ALN sont des variables binaires, et sont réparties en deux groupes : les entrées binaires et les entrées complémentaires (qui sont également encodée sous forme binaire) **[OLI 97]**. Chaque neurone d'entrée est connecté à deux variables binaires, provenant soit de l'ensemble dit « binaire » soit de l'ensemble dit « complémentaire », soit une de chaque **[ARM 97]**.

Ces neurones effectuent alors une opération logique de ET, OU, GAUCHE ou DROITE[38]. Cette opération est déterminée par deux compteurs binaires associés à chacun des neurones. L'incrémentation de ces compteurs représente la phase d'apprentissage d'un réseau ALN. L'objectif de cet apprentissage étant d'obtenir les portes logiques permettant au réseau d'effectuer les opérations attendues.

L'incrémentation des compteurs s'effectue lors de la réception d'un signal de « responsabilité » provenant du neurone en aval. Chaque neurone est donc responsable de l'apprentissage des deux neurones situés en amont de sa position.

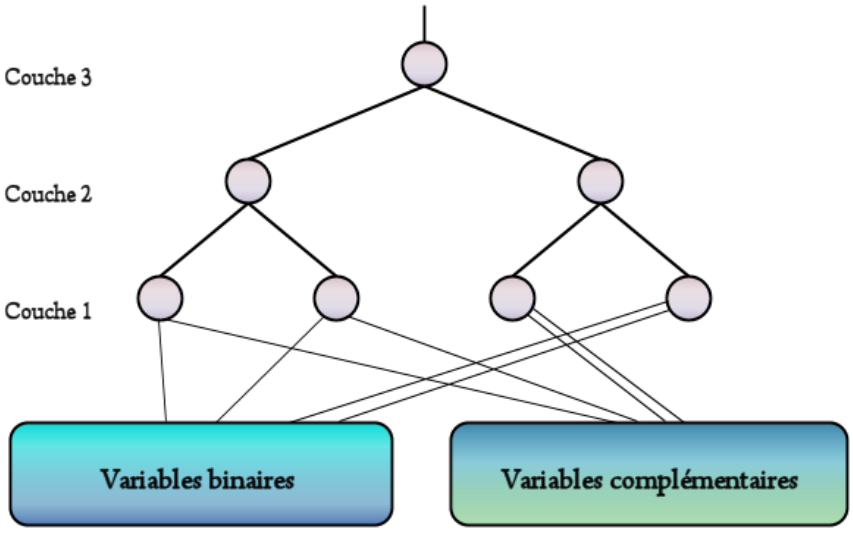


Fig20. *Adaptive Logic Network (ALN)*

Chaque neurone reçoit deux entrées. Il effectue l'opération logique et transmet le résultat au neurone suivant.

La figure 20 précise le fonctionnement d'un ALN en détaillant le fonctionnement d'un neurone. Ce dernier est divisé en deux parties. L'une qui reçoit les signaux Xg et Xd provenant des neurones précédents, y appliquant une opération logique avant d'émettre à son tour vers le neurone suivant (Signal S).

L'autre qui reçoit le signal de responsabilité Sr du neurone en aval, incrémentant ou non ses compteurs. Ces mêmes compteurs qui permettent de sélectionner l'opération logique appliquée par le neurone. C'est cette

[38] GAUCHE et DROITE permettant bien entendu de prendre directement l'information provenant du neurone de « gauche » ou du neurone de « droite ».

même partie qui émet à son tour les signaux Srg et Srd vers les neurones en amont, incrémentant ou non leurs compteurs, selon le résultat d'une partie de l'algorithme de responsabilité.

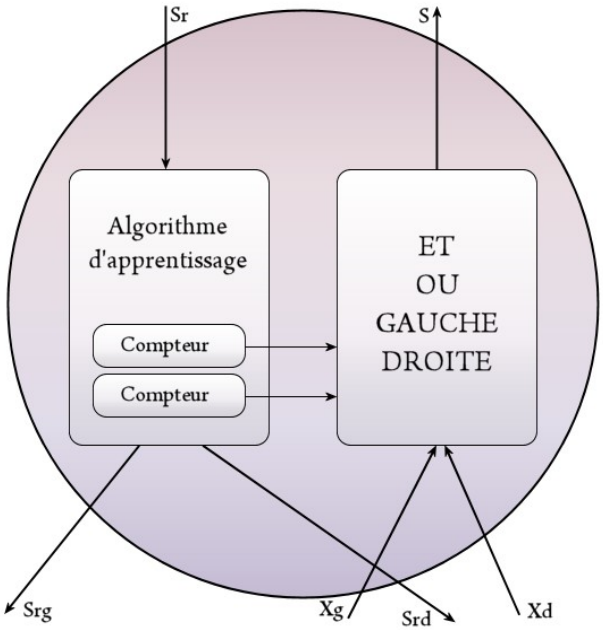


Fig21. *Zoom sur un neurone d'ALN*

Selon la correspondance entre l'information ciblée et l'information obtenue, le neurone doit évaluer via son algorithme d'apprentissage, la responsabilité des deux neurones qui le précédent. Cette opération se répercute ainsi tout au long du réseau, de sa sortie jusqu'à ses entrées. C'est une forme de rétropropagation de l'erreur, de la dernière couche du réseau, jusqu'à sa première.

C'est l'algorithme d'apprentissage qui permet de déterminer le traitement des compteurs et la responsabilité des neurones en amont. Si il existe de nombreux algorithmes d'apprentissage **[ARM 79]**, il semble que le réseau ALN soit aujourd'hui beaucoup moins utilisé, et il est difficile d'acquérir ou de retrouver ces algorithmes ; mais nous conservons le principe de fonctionnement de ces réseaux qui lie un mode d'activation simple à une méthode d'apprentissage aboutie.

- **7.1.2.3 Time Delay Neural Network**

Le « réseau de neurones par délais temporel » ou Time Delay Neural Network (TDNN), et un réseau ont l'originalité se situer sur sa méthode de traitement des informations reçues. Le TDNN utilise plusieurs séries d'informations pour n'obtenir qu'une seule sortie.

Si au premier abord cette méthode de traitement peut sembler latente, elle est en fait particulièrement intéressante car elle permet de traiter une information qui évolue dans le temps et ainsi, d'affiner sa réponse. Ce modèle a notamment été utilisé pour la reconnaissance vocale **[WAI 89]**, la reconnaissance de caractères **[SCH 95]** ou plus récemment, pour la détection de la qualité d'une vidéo **[LEC 08]**.

Pour ce dernier cas, le processus de fonctionnement d'un TDNN permet d'utiliser plusieurs images successives d'une vidéo, qui sont visuellement très proches, pour déterminer la qualité de la vidéo (compression, pixellisation…). La figure suivante illustre cette procédure de fonctionnement.

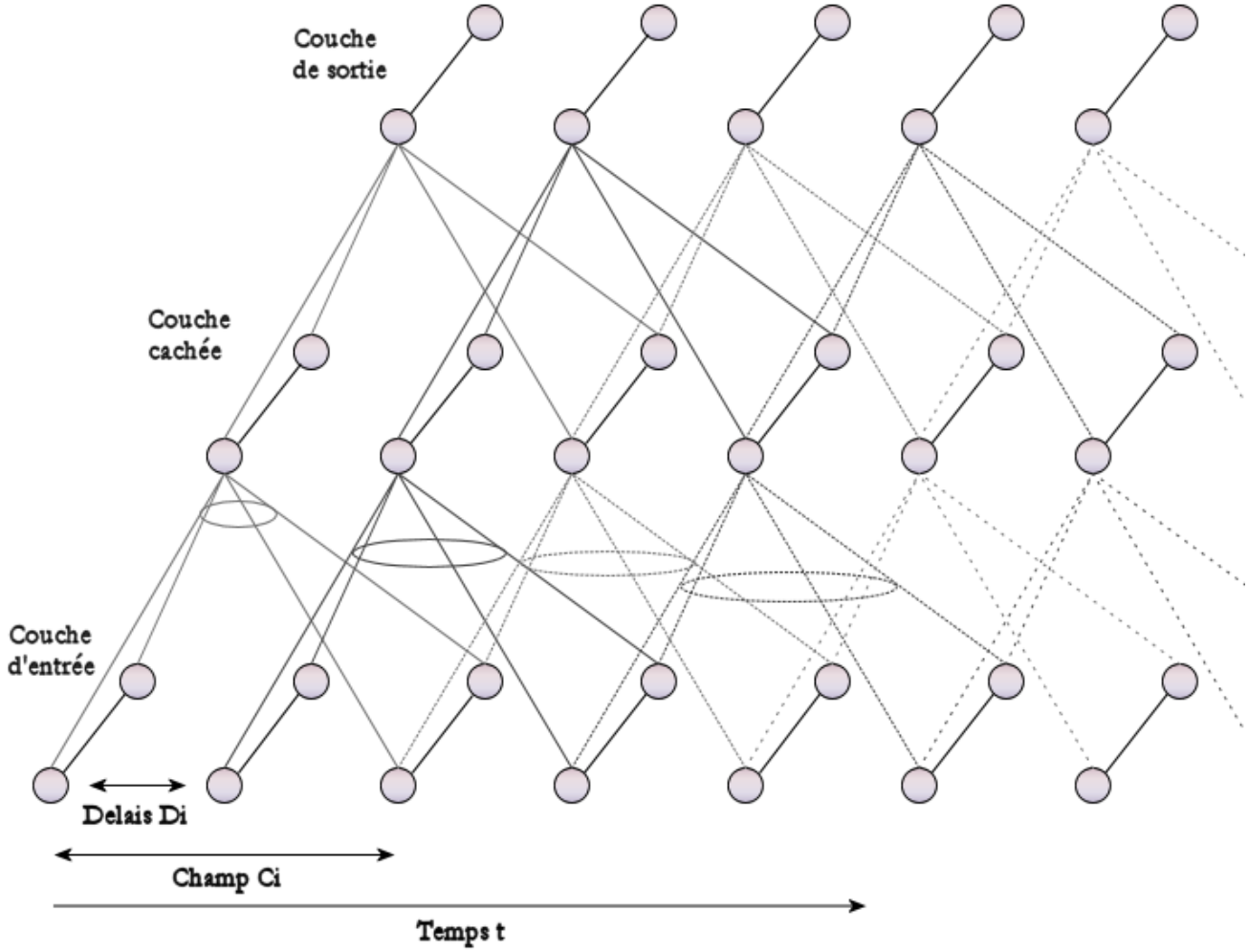


Fig22. *Time Delay Neural Network – modèle générique*

Un TDNN se caractérise ainsi par :

- Son nombre de couches.
- Son nombre de neurones de chaque couche selon la direction temporelle (Temps t).
- Le nombre de neurones de chaque couche, ici, nous avons deux neurones par couche).
- La taille du champ temporel vue par chaque couche (sauf celle d'entrée), le nombre de neurones de la couche i vu par un neurone de la couche i+1 (dans notre cas, quatre neurones).
- Le délai temporel entre chaque champ.

L'algorithme de rétropropagation utilisé pour le perceptron multi-couche et détaillé plus tôt, est généralement l'algorithme d'apprentissage utilisé pour le TDNN.

7.1.3 Réseaux à apprentissage non-supervisé avec rétro propagation

Dans les réseaux à apprentissage non supervisé, les neurones sont en compétition pour être actifs. Ils sont à sortie binaire et son généralement considérés actifs lorsque leur sortie vaut 1. Alors que dans les autres règles plusieurs sorties de neurones peuvent être actives simultanément, dans le cas de l'apprentissage compétitif, un seul neurone est actif à un instant donné. Chaque neurone de sortie est ainsi spécialisé pour « détecter » une suite de formes similaires et devient alors un détecteur de caractéristiques. La fonction d'entrée est dans ce cas, $h = \varphi - dist(\omega, x)$ où φ, ω et x sont respectivement le seuil, le poids synaptique et l'entrée.

Le neurone gagnant est celui pour lequel h est maximum. Ce qui signifie, dans le cas où les seuils sont identiques, celui dont le poids est le plus proche de l'entrée. Le neurone dont la sortie est maximale sera le vainqueur et sa sortie sera mise à 1 alors que les perdants auront leur sortie mise à 0. Un neurone apprend en déplaçant ses poids vers les valeurs des entrées qui l'activent pour augmenter ses chances de gagner. Si un neurone ne répond pas à une entrée, aucun ajustement de poids n'intervient. Si un neurone gagne, une portion des poids de toutes les entrées est redistribuée vers les poids des entrées actives. On est ainsi amené à observer un déplacement des neurones vers les valeurs d'entrées avec un surprenant résutat :

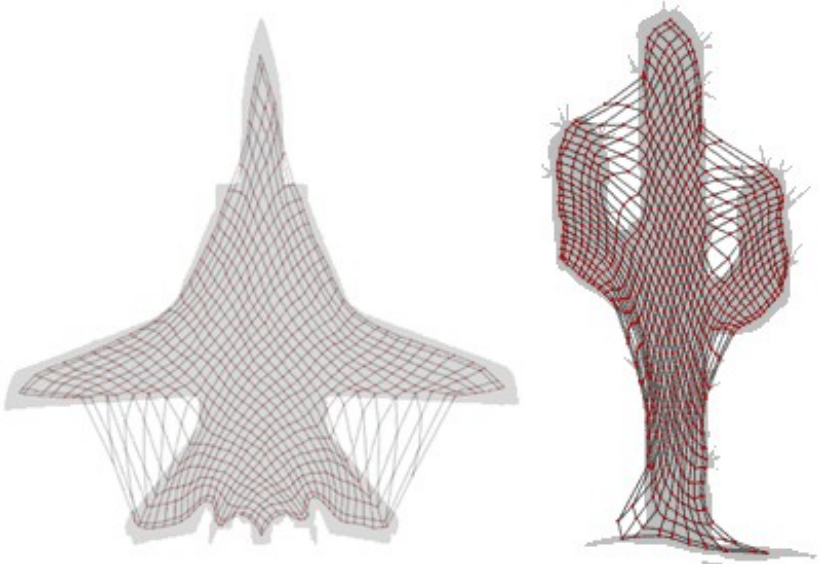


Fig23. *Réseaux de Kohonen appliqués à une silhouette de chasseur et de cactus*

- **7.1.3.1 Kohonen Self Organizing Map [KOH 82, KOH 95]**

Les cartes auto adaptatrices ou Self-Organizing Maps (SOM) sont des réseaux à apprentissage non-supervisé, développés par Teuvo Kohonen en 1984 et qui établissent une carte discrète, ordonnée topologiquement, en fonction de patterns d'entrée. Le réseau forme ainsi une toile dont chaque noeud est un neurone associé à un vecteur de poids. La correspondance entre chaque vecteur de poids est calculée pour chaque entrée. Par la suite, le vecteur de poids ayant la meilleure corrélation, ainsi que certains de ses voisins, vont être modifiés afin d'augmenter encore cette corrélation. On retrouve dans cette procédure, le principe de fonctionnement des réseaux Linear Vector Quantization, étudiés plus tôt et également développés par Kohonen.

Les cartes auto adaptatrices sont constituées :

- d'une couche d'entrée où tout motif a classer est représenté par un vecteur multidimensionnel qualifié de vecteur d'entrée. A chaque motif est affecté un neurone d'entrée.

- d'une couche de sortie également appelée couche de compétition où les neurones entrent en compétition. Les valeurs provenant des neurones d'entrées sont transmises à tous les neurones de la couche de compétition, en même temps. pour approcher des valeurs passées par les neurones d'entrées.

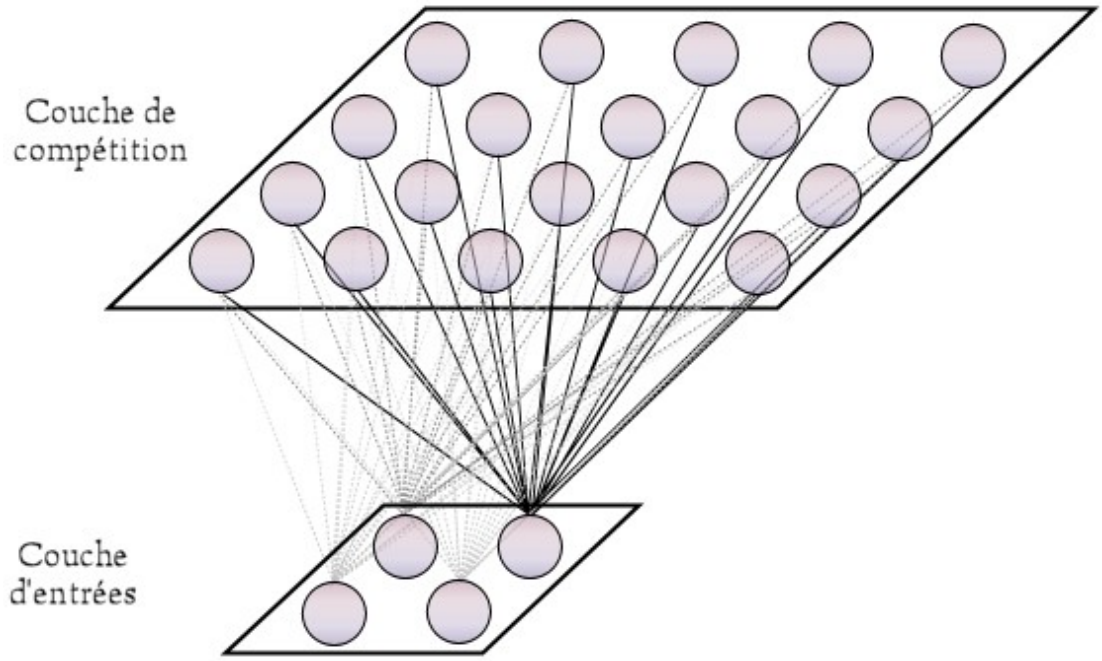


Fig24. *Self Organizing Map*

On peut constater sur la figure 24 que chaque neurone d'entrée dispose d'un lien avec la totalité des neurones de compétition, l'apprentissage se déroule alors de la façon suivante :

On soumet successivement les valeurs d'un vecteur d'entrée multidimensionnel aux neurones d'entrées, ces derniers répartissent alors l'information à l'ensemble des neurones de la carte qui entrent alors en compétition. La compétition s'effectue en calculant la distance entre le vecteur d'entrée et chacun des neurones, le neurone le plus proche emporte la compétition. Le coefficient synaptique du vainqueur est alors réévalué ainsi que celui des neurones proches de lui. On utilise pour cela la formule suivante :

$$\omega_i(t+1) = \omega_i(t) + \alpha_{ci}(t).[x_i(t) - \omega_i(t)]$$

Avec ω_i le coefficient synaptique du neurone i.

$\alpha_{ci}(t)$ le coefficient d'apprentissage : fonction linéaire décroissante, grande au début pour accélérer cette phase, plus faible par la suite afin d'affiner. Ce coefficient a une valeur qui décroît aussi pour les neurones voisins, de manière à ce qu'ils soient moins affectés que le neurone vainqueur, par la réévaluation **[SPD 09a , WIK 09e]**.

Et $[x_i(t) - \omega_i(t)]$ la distance euclidienne entre le vecteur d'entrée x_i et le coefficient ω_i à l'instant (t).

Le coefficient d'apprentissage est simplement constitué d'une fonction linéaire décroissante de type $\frac{a}{b+t}$ avec a et b des constantes, et d'une fonction impliquant la distance entre le neurone vainqueur et le neurone voisin ainsi que le temps. Afin de diminuer l'importance de la modification dans le temps et l'espace, la fonction suivante est généralement utilisée **[OGI 08]** :

$$\exp\left(\frac{-dist(c,i)}{(2.radius^2(t))}\right)$$

Ce qui donne alors pour $\alpha_{ci}(t)$:

$$\alpha_{ci}(t) = \frac{a}{b+t} \cdot \exp\left(\frac{-dist(c,i)}{\left(2.radius^2(t)\right)} \right)$$

En appliquant cette formule les neurones voisins verront donc leurs coefficients synaptiques augmenter proportionnellement à leur distance par rapport au neurone vainqueur.

Les cartes auto adaptatives peuvent être utilisées dans les applications de classification non-supervisées en rassemblant les motifs fournis en entrées selon les caractéristiques qui les rapproches : forme, couleurs, apparence générale … Il faut garder à l'esprit qu'une carte n'est pas nécessairement en deux dimensions, il est possible d'en utiliser plus et détendre ainsi les applications.

Comme il l'a été indiqué, on retrouve les cartes auto adaptatives pour le clustering[39], mais également pour la prédiction (la prévision numérique de la météorologie est une intéressante illustration de cet usage), pour la représentation de données statistiques, la classification temps réelle, ou enfin, la perception de motifs et de mouvements.

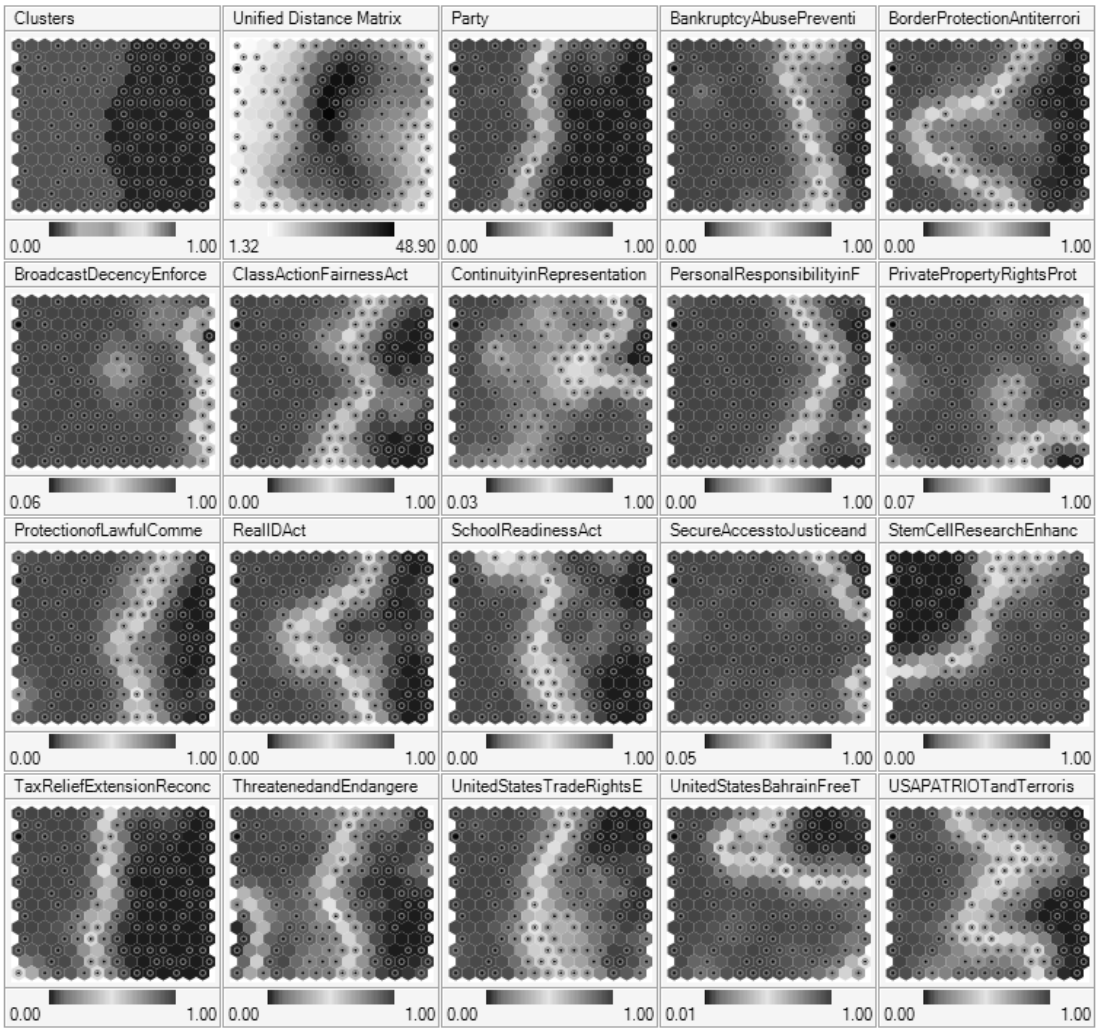


Fig25. *Clustering et représentations de données classifiées*

- **7.1.3.2 Adaptive Resonance Theory**

[39] Méthode statistique d'analyse des données qui a pour but de regrouper un ensemble de données en différents paquets homogènes, en ce sens que les données de chaque sous-ensemble partagent des caractéristiques communes.

Les réseaux ART (Adaptative Resonnance Theory) sont des réseaux à apprentissage par compétition développés par Stephen Grossberg en 1976. Ils ont été conçus spécifiquement pour traiter l'information de la même façon que le cerveau, alors que les autres réseaux de neurones n'ont de cérébral que leur nom et le principe de fonctionnement indépendant de chaque neurone, les réseaux ART se veulent traiter l'information d'une façon identique ou du moins, très proche, de celle du cerveau humain. Ils sont principalement destinés aux applications de perception et de prédiction.

Un réseau ART est constitué d'une couche de comparaison et d'une couche de reconnaissance, chacune constituée d'un nombre variable de neurones ; un paramètre de vigilance qui permet de valider ou non l'apprentissage d'une nouvelle information et enfin, un module de remise à zéro. Le paramètre de vigilance a une influence considérable sur le réseau, une vigilance élevée produit des mémoires extrêmement détaillées (des catégories plus précises et une granularité plus importante) alors qu'une vigilance plus faible résultera sur des mémoires plus générales. La couche de comparaison prend un vecteur d'entrée (un tableau unidimensionnel de valeurs) et le transfert au neurone s'y rapprochant le plus dans la couche de compétition. Ce dernier étant bien entendu celui dont la valeur du vecteur de poids se rapproche le plus de la valeur du vecteur d'entrée. Ce qui est effectué en utilisant la formule suivante **[GRO 87]** :

$$y_i = \sum_{j=1}^{n} \omega_{ij}(t) x_i$$

C'est alors qu'intervient le principe de l'inhibition latérale, qui est la capacité d'un neurone biologique à réduire ou à interrompre l'activité des neurones voisins. Dans notre réseau, chaque neurone de la couche de reconnaissance émet un signal négatif, proportionnel à son rapport au vecteur d'entrée, à l'ensemble des autres neurones de la couche de reconnaissance, les inhibant en conséquence. Permettant ainsi, à chaque neurone, de représenter la catégorie correspondant à une certaine classe de vecteur d'entrées.

Une fois le vecteur d'entrée classifié, le module de remise à zéro compare le niveau de correspondance de la reconnaissance au paramètre de vigilance. Si le seuil que représente ce paramètre est atteint, alors l'entraînement débute. Dans l'autre cas, le neurone vainqueur est inhibé jusqu'à ce qu'un nouveau vecteur d'entrée soit appliqué. L'entraînement ne débute qu'après le déroule d'une procédure de recherche. Durant cette procédure, les neurones de la couche de reconnaissance sont désactivés un par un, par le module de remise à zéro, jusqu'à ce que le seuil du paramètre de vigilance soit atteint par l'un des neurones.

L'entraînement, lui, peut se employer deux modes différent : le mode lent ou le mode rapide. Dans l'apprentissage lent, le coefficient d'équilibrage du poids du neurone par rapport au vecteur d'entrée, est calculé à partir de valeur continues et d'équations différentielles, l'ensemble est dépend du temps écoulé durant la soumission du vecteur d'entrée au réseau. L'apprentissage rapide utilise des équations polynomiales et des valeurs binaires pour calculer l'ajustement du poids à effectuer.

Si l'apprentissage rapide prend bien sûr moins temps tout en étant efficace sur de nombreuses tâche, il est biologiquement moins plausible et moins proche du probable traitement effectué par le cerveau. L'apprentissage lent lui l'est donc beaucoup plus et permet de surcroît le traitement de vecteur d'entrée variant en permanence.

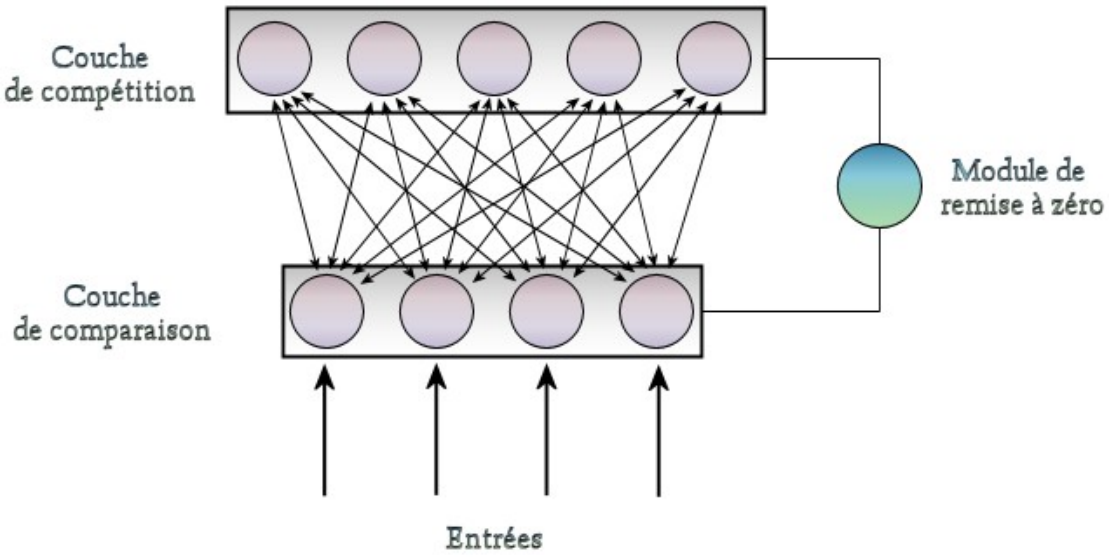


Fig26. *Structure d'un réseau ART*

Il existe plusieurs versions de réseaux ART :

- Le réseau *ART1* qui est la première version et la plus basique qui soit, elle ne travaille que sur des valeurs binaires **[GRO 03]**.
- Le réseau *ART2* qui étend la capacité du réseau en supportant des entrées continues **[GRO 87a]**.
- Le réseau *ART2-A*, une version améliorée du réseau *ART2* à la vitesse de traitement nettement supérieure, tout en maintenant des résultats d'une qualité très rarement inférieure à celle des *ART2* **[GRO 91]**.
- *ART3*, basée sur le modèle *ART2* et qui se veut simuler, au plus près, le fonctionnement des neurotransmissions en simulant des concentrations d'ions calciums et sodium dans le système d'équations, afin d'obtenir une fonctionnement plus proche de la réalité **[GRO 90]**.
- *Fuzzy ART* qui implémente la logique floue[40] à un réseau *ART*, essentiellement pour la reconnaissance de forme tout en exploitant la capacité de généralisation des réseaux de neurones **[GRO 91a]**.
- *ARTMAP* qui combine plusieurs réseaux *ART* en centralisant leur fonctionnement au sein d'un apprentissage supervisé **[GRO 91b]**.

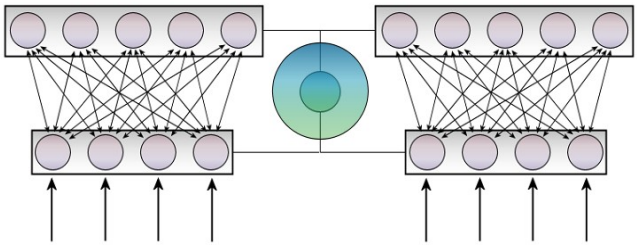


Fig26. *Structure d'un réseau ARTMAP*

- enfin, *Fuzzy ARTMAP*, qui cumule le principe précédent de l'*ARTMAP* et la logique floue **[GRO 92]**.

[40] À l'inverse de la logique booléenne, la logique floue permet à une condition d'être en un autre état que *vrai* ou *faux*. Il y a des degrés dans la vérification d'une condition.

7.2 D'Applications de l'Intelligence Artificielle aux Jeux Vidéo

La récente et fulgurante évolution de la qualité visuelle et sonore des jeux vidéo suscite depuis la fin des années 1990 un engouement à l'égard des progrès technologiques effectués par les studios de développement. Le développement et l'intégration de technologies modernes toujours plus nombreuses, comme le bump mapping[41] et ses nombreuses variantes, l'éclairage volumique, l'ombrage ou la radiosité temps réel ; exploitées par des moyens matériels de plus en plus performants, permettent aux moteurs de rendus d'offrir des environnements temps réel au réalisme parfois bluffant.

Fig27. *Crysis – Crytek Studios (2007)*

Aujourd'hui, ce réalisme est obtenu par la communion de technologies provenant de différents domaines tel que la 3D, la physique (pour la gestion de la mécanique classique comme la chute de corps, les collisions, l'inertie, les forces…), le traitement du son et de l'image, l'animation ou encore, l'électronique, base de l'ordinateur et de son systèmes de calcul. L'unification de ces domaines permet de concevoir des environnements immersifs car riches en détails et en similitudes avec notre réalité.

Toutefois, si cette immersion est rendue possible au travers de la qualité visuelle des environnements où le joueur évolue, l'intégration de personnages non joueurs, entièrement gérés par l'ordinateur, nécessite la prise en compte d'un nouveau facteur, qu'est l'intelligence artificielle. Dans des environnements ressemblants très grossièrement à notre réalité ou ne faisant que l'évoquer ; il était possible de se contenter d'une intelligence artificielle elle aussi grossière et prévi-

[41] Le bump mapping (placage de relief) est une technique permettant de donner un aspect rugueux ou nivelé à une surface plane, cette simulation permet une importante économie de performance tout en augmentant sensiblement le réalisme émanant des surfaces affectées.

sible, comme cela était le cas dans les années 1980 et durant la première moitié des années 1990. Mais aujourd'hui, une intelligence artificielle grossière voir absente n'est plus envisageable.

Des personnages à l'apparence et à l'animation comportementale et faciale étonnamment proche de la réalité ne peuvent se contenter d'une intelligence approximative et trop contrastée avec la qualité de l'environnement, à moins de sacrifier l'immersion tant recherchée. C'est pourquoi les recherches en intelligence artificielle se sont accélérées depuis le début de notre siècle, car si parfois l'utilisation de zombies et de monstres à l'intellect limité permet de justifier l'attitude extrêmement primaires des personnages non joueurs, cette alternative contente de moins en moins de joueurs, et de développeurs.

Fig28. *Heavy Rain – Quantic Dream (Sortie prévue en 2010)*

Ainsi, concevoir un jeu véritablement immersif impose aujourd'hui l'intégration d'une intelligence artificielle capable de mimer différents faciès du comportement humain. Car quel que soit le jeu que l'on souhaite développer, il ne fait nul doute que ce dernier comportera de nombreuses raisons de simuler l'attitude et les agissements d'êtres à l'intelligence équivalente à la notre.

Qu'il s'agisse d'un jeu de course de voitures, d'une simulation militaire, d'un FPS[42], d'un jeu d'aventure ou de rôle, ou d'une simulation de vie comme la série *Les Sims.* Tous engagent à un moment où à un autre, des antagonistes susceptibles d'agir au travers d'un raisonnement proche de celui d'un humain. Proche, car nous le savons, il n'est pas encore question de concevoir aujourd'hui, une intelligence artificielle forte, douée d'une conscience, et agissant de la même façon que n'importe quel être humain normal.

[42] Un First Person Shooter (jeu de tir à la première personne) est jeu vidéo de tir en 3D dans lequel le personnage doit en général éliminer des ennemis à l'aide d'une arme à feu et dans lequel l'angle de vue proposé simule le champ visuel du personnage incarné.

Et si cette ambition reste la finalité à atteindre de cette science qu'est l'intelligence artificielle, nos objectifs à court et moyen termes sont plus modestes. Bien souvent, impressionner le joueur, avertit ou non, grâce aux progrès effectués, le bluffer quelques instants ou lui donner l'envie de percer les failles d'une nouvelle forme d'intelligence artificielle, représentent les jalons de nos ambitions. Jalons identiques à ceux rencontrés depuis une petite vingtaine d'années par les développeurs de moteurs de rendus, et les artistes qui les accompagnent ; puisque qu'il y a presque quinze ans déjà, tous étaient impressionnés des progrès observables dans chaque nouveau moteur de rendu, chaque nouveau repoussant un peu plus loin le réalisme et l'immersion visuelle. Domaine qui semble si loin de celui de l'intelligence artificielle et qui pourtant possède le même objectif que de faire confondre réel et artificiel.

Ainsi, c'est ce même chemin que nous empruntons aujourd'hui et pour quelques années encore ; avec des agents intelligents en lieu et place des moteurs de rendus et la simulation d'un comportement humain pour les PNJ remplaçant l'illusion visuelle et l'immersion graphique recherchée au travers de chaque nouveau moteur de rendu. Toutefois, l'intelligence artificielle que nous rencontrons dans les jeux vidéos et les environnements virtuels peut se diviser en plusieurs axes, chacun nécessitant une approche indépendante puis, d'estimer l'intérêt que pourrait avoir un réseau de neurones dans cet axe précis.

7.2.1 Roaming et déplacements

L'aspect « Roaming » concentre toutes les notions liées aux déplacements : Où aller ? A quelle vitesse ? Quel chemin emprunter ? Comment y aller ?

Il est ici question de gérer les actions d'un PNJ désirant aller d'un point A à un point B. il est important de préciser qu'à cette étape du raisonnement, la raison de ce choix ne nous concerne plus, la décision ayant été prise en amont de cette fonction. Mais il est toutefois possible qu'un signal provenant de la fonction précédente interrompe ou change cette décision, mettant ainsi à jour les informations la partie de l'agent intelligent, pilotant les déplacements.

Cet aspect de l'intelligence artificielle est assez riche, les déplacements étant un problème majeur des jeux vidéo, ils ont été abordés à maintes reprises. Ainsi les algorithmes de pathfinding comme A star ou Dijkstra ont fait leur preuve dans leur aptitude à trouver le chemin le plus court pour atteindre un point à partir d'un autre [RUS 06]. Toutefois, pour être correctement appliqués, ces algorithmes ont besoin d'informations en provenance de leur environnement : Où sont les murs ? Les escaliers ? Un précipice ? Un ou plusieurs ennemis ? Un objet particulièrement intéressant à récupérer en chemin ?

Ces informations proviennent de la couche de perception, celle là même où un éventuel réseau de neurones serait intégré. Permettant ainsi à l'algorithme associé de posséder les informations nécessaires à son bon déroulement.

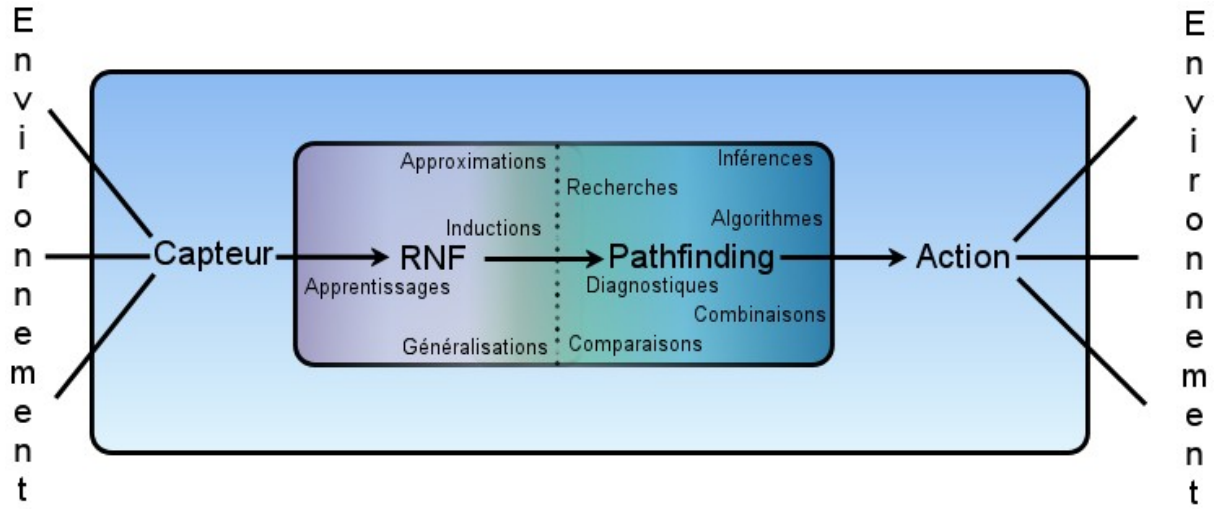


Fig29. *Agent intelligent associant un réseau de neurones a un algorithme de pathfinding*

L'intérêt pour nous serait alors d'implémenter un réseau de neurone, en « entrée » de l'agent intelligent, permettant de fournir des informations sur l'environnement à l'algorithme de pathfinding assurant le rôle décisionnel de l'agent.

Selon nos ambitions, ce réseau peut être plus ou moins complexe, pouvant aller jusqu'à percevoir des informations sonores. L'essentiel toutefois serait de percevoir et de traiter les informations visuelles exploitables par l'algorithme de pathfinding.

7.2.2 Behavior et décisions

L'aspect Behavior ou « comportement », peut être considéré comme le cœur de l'organisation d'une intelligence artificielle. C'est à ce niveau que se prennent les décisions majeures de l'entité :

« Attaquer l'ennemi ? »

« Lancer une grenade ? »

« Aller a couvert ? »

« Changer d'arme ? »

Si ces décisions peuvent paraître basiques et ne pas nécessité d'algorithme particulièrement évolué, la complexité de la tâche prend tout son sens lorsque l'on présente plus de caractéristiques :

« Vaut-il mieux prendre 4 secondes pour recharger mon arme et continuer de tirer sur mon ennemi qui se situe à environ 8 mètres et qui se rapproche à une vitesse estimée d'un mètre par seconde, ou bien courir a couvert à plus de 10 mètres de ma position actuelle pour remonter ma santé actuellement à 17% ? À moins de donner l'ordre à un de mes équipiers de me couvrir. »

Chacune des décisions envisageables implique de nombreux paramètres, risques et conséquences et certains arbres de décisions peuvent s'avérer être extrêmement complexes et y simuler un raisonnement humain n'est pas facile. Se pose par ailleurs un problème majeur de l'intelligence artificielle : souhaitons nous concevoir une intelligence proche de l'intelligence humaine et donc, susceptible de commettre les même erreurs, ou bien une intelligence à la rationalité parfaite ? Ce dilemme est compliqué, et naviguer entre ces deux objectifs est assez délicat.

Il existe de nombreuses technologies pour répondant à ce problématique sous forme d'arbre de décision. L'une des plus efficace permettant de résoudre une problématique comme celle énoncée plus haut est le réseau Bayésien [RUS 06, NAI 04].

Les réseaux Bayésiens sont des machines à calculer des probabilités conditionnelles : En fonction d'informations observées, ils calculent la probabilité d'événements non observés. Par exemple, en fonction des caractéristiques d'une situation dans le jeu, ils calculent la probabilité de réussite des différentes actions compatibles avec la situation. Ils peuvent aussi calculer la probabilité d'informations non observées, et en déduire des actions envisageables.

Les réseaux Bayésiens sont donc très adaptés pour résoudre ce type de problème, ceci de façon entièrement autonome. Toutefois, leur fonctionnement est basé sur des informations observées en provenance de l'environnement, et il pourrait être intéressant d'associer leur fonctionnement à un réseau de neurone chargé de percevoir les informations de cet environnement et de fournir des données traitées, au réseau Bayésien.

Il est également possible d'essayer de remplacer les réseaux Bayésiens par un réseau de neurones, dans l'objectif d'effectuer des comparaisons d'efficacité. Des réseaux de neurones légers ont déjà été employés avec succès dans des jeux, pour gérer l'apprentissage de PNJ [CRE 96, B&W 01].

7.2.3 Planification & stratégie

Une intelligence artificielle orientée sur la planification et la stratégie est principalement utilisée dans des jeux de type RTS (Real Time Strategy) où le joueur doit mener des batailles contre un adversaire géré par l'ordinateur. Si dans une partie d'Echec, de Dames ou de Go, on peut envisager l'utilisation de la logique combinatoire et de la rationalité pour réaliser les meilleures décisions à chaque coup, cela n'est pas possible pour un RTS. En effet, ces jeux utilisent généralement différents concepts qu'il est difficile de gérer et d'équilibrer de façon parfaitement rationnelle : la récolte de différentes ressources, la production d'énergie, la construction de bâtiments oude défenses, l'évolution des recherches pour améliorer la vitesse des unités, leur résistance, ou pour produire plus d'énergie ou obtenir de nouvelles possibilités de constructions ; le vaste panel d'unités maritimes, aériennes ou terrestres couplé aux avantages et aux défauts de chacune ; la topologie du champs de bataille, la position de l'adversaire ainsi que ses différents choix stratégiques, sans oublier le fameux « brouillard de guerre »[43] qui empêche l'ensemble des joueurs de se maintenir au courant, en perma-

[43] Le brouillard de guerre ou Fog of War, dans certains jeux de stratégie, est un voile qui obscurcit une partie de la carte et empêche le joueur de connaître les activités de l'ennemi.

nence des activités de l'adversaire. La totalité de ces actions et de ces événements génère une quantité infinie de possibilités, mêlée à une part d'aléatoire et d'inconnu, ce qui rend impossible une gestion omnisciente de la situation comme dans un jeu d'échec où chaque coup peut être prévu et où seule la puissance de calcul et la qualité de l'algorithme combinatoire permet d'obtenir la victoire.

Ainsi, ce type d'intelligence artificielle suit généralement un arbre décisionnel écrit par les développeurs. Cet arbre contient la procédure de développement de la « base » gérée par l'ordinateur : ordre de construction des bâtiments, nombre de récolteurs, ordre de recherche des technologies, construction des unités, fréquence des attaques et nombre d'unités par escouade. Ce type d'arbre, extrêmement simple, n'offre malheureusement pas une intelligence artificielle de qualité, et le joueur en perçoit rapidement les nombreuses failles : défenses inadaptées ou mal positionnées, bâtiments principaux trop exposés, attaques inutiles et sans aucune stratégie, priorités mal choisies.

Si l'utilisation de réseaux Bayésiens pourrait s'avérer intéressante, il est probable qu'elle soit également limitée ; ces derniers ayant une base de connaissance figés, ils ne peuvent apprendre en temps réel et faire ainsi évoluer leur stratégie selon les méthodes employées par leur adversaire. Toutefois, si les réseaux de neurones formels semblent être adaptés à cette problématique, leur intégration et leur utilisation pourraient s'avérer plus délicate qu'il n'y parait. En effet, les réseaux de neurones sont avant tout particulièrement efficaces pour la classification, la perception et la reconnaissance ou encore, la généralisation. Si ils sont capables d'apprendre et de changer leur choix selon leur expérience, le mode d'apprentissage supervisé ne peut pas convenir, à moins de trouver une méthode pour que le joueur supervise inconsciemment l'apprentissage de son adversaire, et un mode d'apprentissage non-supervisé ne semble pas adapté à nos besoins.

Le modèle SOM de Kohonen pourrait être utilisé en parallèle d'un outil décisionnel, pour observer l'activité de l'adversaire, plaçant un neurone sur chaque zone observable, ce dernier réagirait ou se déplacerait selon l'activité observée. Il serait alors possible de transmettre des informations précises au corps décisionnel de l'agent.

Enfin, l'utilisation d'un Perceptron multicouches pourrait également s'avérer intéressante, notamment pour percevoir l'orientation ou la vitesse de déplacement d'escouades ennemies. Il serait alors au préalable, nécessaire de correctement les entraîner.

8. Mise en application

8.1 Préparation

8.1.1 Applications testées

L'objectif de nos tests est de déterminer l'efficacité de réseaux de neurones formels appliqués à certaines utilisations de l'intelligence artificielle dans les jeux. Nous nous servirons pour cela de notre première approche effectuée précédemment :

<u>Roaming et déplacements :</u>

Test A : Perception visuelle et transmissions des informations utiles au module décisionnel constitué d'un algorithme de pathfinding quelconque.

Test B : Perception sonore et transmissions des informations utiles au module décisionnel constitué d'un algorithme de pathfinding quelconque.

Behavior et décisions :

Test C : Perception d'informations pré organisées et utilisation pour effectuer des décisions ou pour générer de nouvelles informations qui pourront être traitées par un réseau Bayésien

Planification & stratégie :

Test D : Mise en place d'un réseau de kohonen pour détecter l'activité de l'adversaire.

Test E : Utilisation d'un réseau de neurone pour percevoir le déplacement de troupes voir la vitesse de ce déplacement.

8.1.2 Méthode de test

Pour tester ces différentes applications, nous avons fait le choix d'utiliser des réseaux spécifiques à chaque test, ceci en adaptant à chaque fois, la topologie et les différentes caractéristiques du réseau.

Les motifs utilisés pour l'entraînement seront adaptés à chaque application. Il s'agira bien entendue d'informations traitées en amont de la prise en charge par le réseau. Il n'est par exemple pas question, de fournir une image rastérisée en haute résolution en entrée du réseau, cela serait beaucoup trop lourd. L'image aurait alors été traitée et adaptée.

Une fois les réseaux entraînés, nous leur soumettrons des vecteurs d'entrées plus ou moins proche de ce qui leur a été soumis durant leur entraînement et nous évaluerons l'efficacité des réseaux pour ces différentes tâches.

Nous ferons également usage des études effectuées sur l'application NeroGame (*Neuro-Evolving Robotic Operatives*) développée par le *Neural-Network Research Group* **[NNR 09]** et utilisant le moteur rtNEAT (*Real-Time Neuro-Evolution of Augmenting Topologies*). Cette application et ce moteur sont destinés aux chercheurs dans le domaine de l'intelligence artificielle et de l'utilisation de réseaux de neurones, elle permet de paramétrer un grand nombre d'éléments et s'avère particulièrement intéressante pour les trois concepts de l'IA que nous avons précédemment déterminés (déplacements, décisions et stratégies). De nombreuses

études ayant été effectuées sur ce moteur, elles nous serviront lors de l'analyse de nos résultats et de la conclusion.

Enfin, nous ferons également usage de la librairie FANNe (Fast Artificial Neural Network explorer) **[FAN 09]** qui permet de mettre facilement et rapidement en œuvre des réseaux de neurones afin d'effectuer des tests et d'obtenir des graphiques explicites et complets.

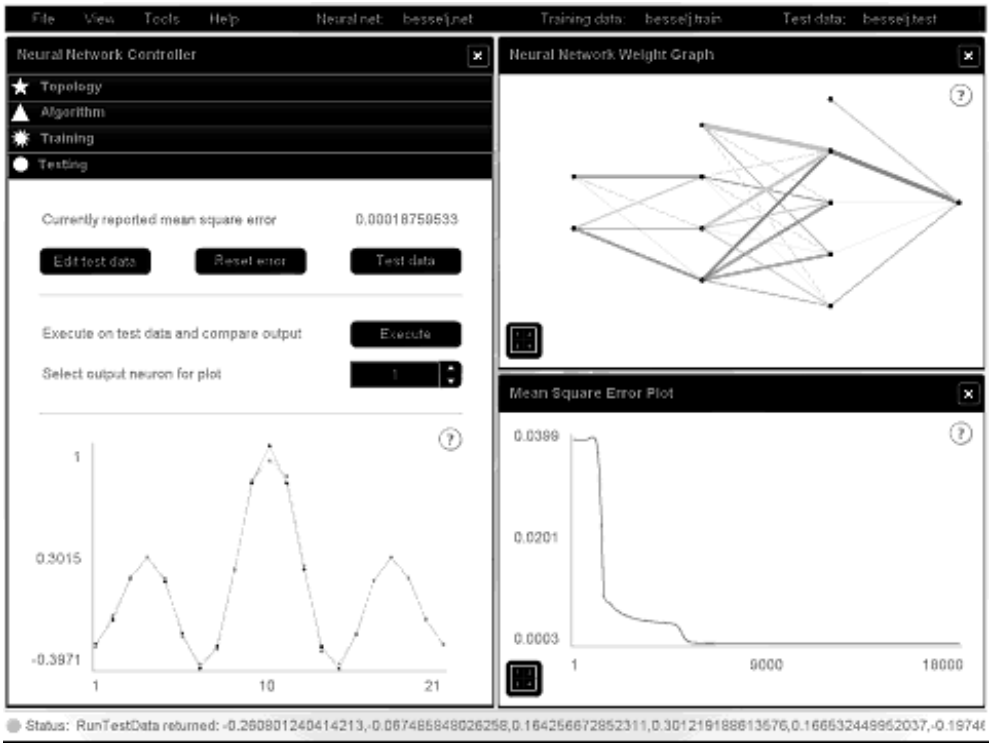


Fig30. *Fast Artificial Neural Network Explorer*

- **8.1.2.1 Test A**

Pour le test A (perception et reconnaissances d'informations) nous avons fait le choix d'utiliser un Perceptron multicouches à rétropropagation, pour sa polyvalence et l'efficacité qui lui est reconnue en terme de perception. Nous testerons plusieurs topologies et feront varier le nombre de neurones par couche, ainsi que le nombre de couches.
Nous considérerons le contexte suivant :
Le Perceptron doit être en mesure de détecter la présence d'éléments susceptible de modifier l'algorithme de pathfinding qu'il alimente. Le vecteur d'entrée disposé en entrée du Perceptron est un tableau de valeurs RGB, ces valeurs proviennent de l'image perçue par l'agent intelligent.
Toutefois, afin d'optimiser les chances du Perceptron à percevoir une information utile, cette image a été découpée en plusieurs carrés de tailles égales, ils sont ensuite successivement transmis au Perceptron. Ce dernier doit réagir sur la présence de pixels d'une couleur particulière qui est, dans notre cas, le rouge. Cependant, une image peut tout à fait contenir un pixel rouge sans pour autant que cela ne signifie la présence d'une entité à prendre en considération pour le Pathfinding.

Plusieurs traitements peuvent ainsi être envisagés en amont du Perceptron, la plus intéressante d'entre elle est probablement d'effectuer une moyenne de la valeur RGB de chaque zone. Cela permet d'une part de minimiser le nombre de traitement effectué par le Perceptron, mais également,

d'augmenter les chances de faire réagir ce dernier sur une information réellement utile pour la procédure de pathfinding.

- **8.1.2.2 Test B**

Comme pour le test précèdent, nous ferons usage d'un Perceptron multicouche à rétropropagation, il recevra en entrée les valeurs analogiques du spectre sonore capté par l'agent intelligent.

Chaque entrée recevra une information décalée dans le temps par rapport à l'entrée précédente, afin de suivre l'évolution du son dans le temps et d'en déduire si il doit être utilisé par le Pathfinding ou non.

- **8.1.2.3 Test C**

Pour le test C nous ferons usage d'un réseau ART 1 et d'un Perceptron Multicouches à rétropropagation, l'objectif étant avec ce premier réseau, de classer les décisions, et avec le second, de prendre la meilleure décision selon les choix proposés.

Dans le cadre de nos tests, ces choix seront proposés au réseau de façon numérique ou sous forme de motifs.

- **8.1.2.4 Test D**

Nous souhaitons observer l'efficacité d'un réseau de Kohonen à suivre en temps réel l'évolution de motifs qui lui sont proposés.

- **8.1.2.5 Test E**

Nous avons hésité entre un Time-Delay Neural Network et un Perceptron Multicouches, il s'avère toutefois que ce type de test à déjà été effectué avec succès avec un TDNN **[SCH 95]**, nous nous tournons donc uniquement vers le Perceptron Multicouche à rétropropagation pour effectuer ce test.

8.2 Mise en pratique

8.2.1 Test A - Déroulement et observations

Nous avons débuté notre test en choisissant un Perceptron Multicouche à rétropropagation comme réseau de neurone. Ce réseau a trois entrées, une pour chaque composante rouge, vert et bleu, une couche cachée composée de quatre neurones, et une sortie dont la valeur varie entre 0 et 1.

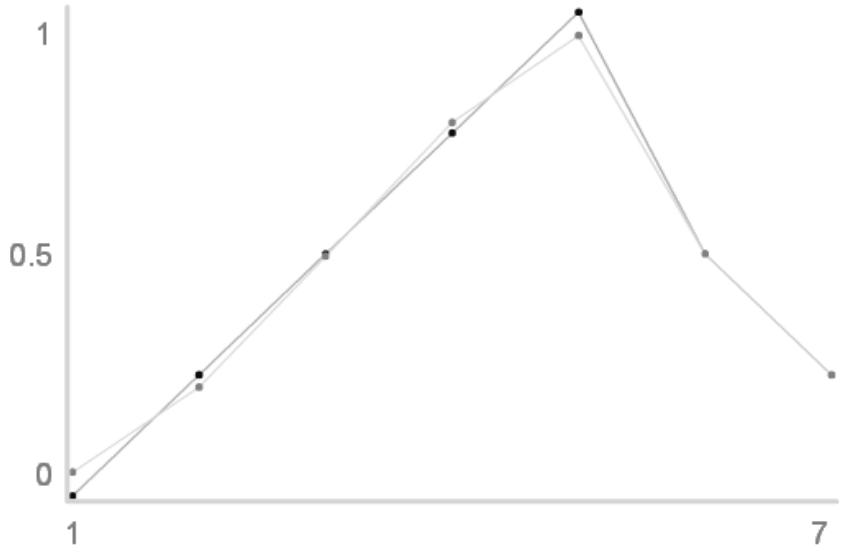
Cette valeur doit se rapprocher de 1 si la zone est majoritairement rouge, et de zéro si ce n'est pas le cas.

Nous avons commencer notre test en entraînant le réseau à détecter du rouge uniquement, sans prendre en compte les autres composantes. Puis nous avons ajouté petit à petit chacune des autres composantes en nous assurant de la réaction adaptée du réseau et de sa sortie. Une fois le réseau réagissant correctement à nos entraînements, nous lui avons soumis des cas plus complexes, notamment des cas qui ne lui avaient pas été soumis lors de l'entraînement, afin d'observer sa réaction.

8.2.2 Test A - Résultats

N.B : la courbe claire représente le résultat attendu, la courbe rouge le résultat observé. Pour les réseaux de neurones, les courbes foncées représente les poids négatifs, et les courbes claires, les poids positifs. La taille de la courbe est représentative de la valeur de ce poids (très positif ou très négatif)

Nos premiers essais en ajoutant progressivement du rouge sur les cinq premiers pixels puis en ajoutant du vert et enfin du bleu :



Nous pouvons constater une évolution correcte de la courbe.

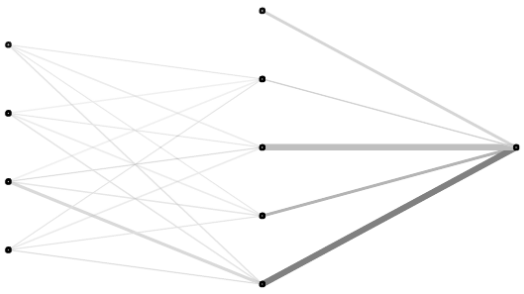
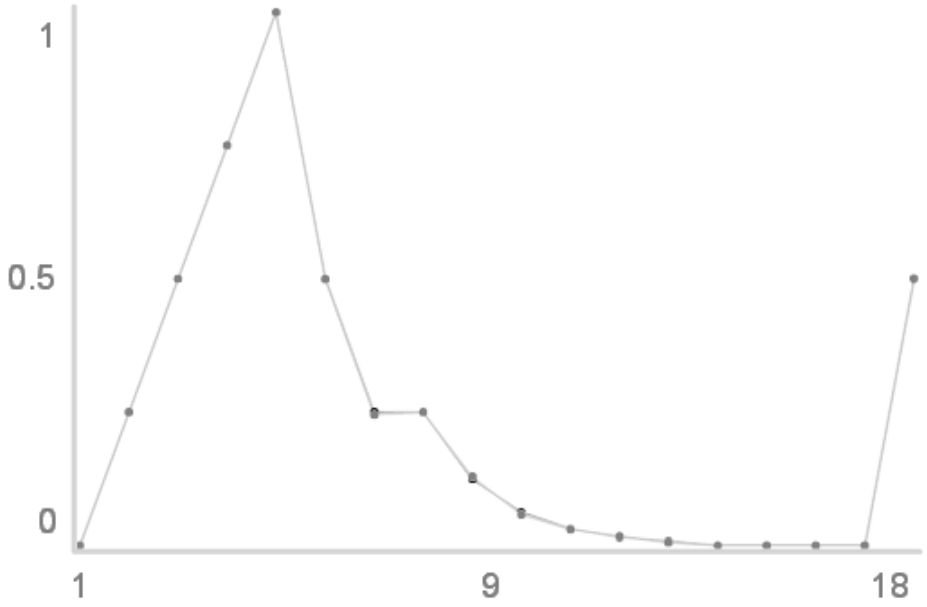


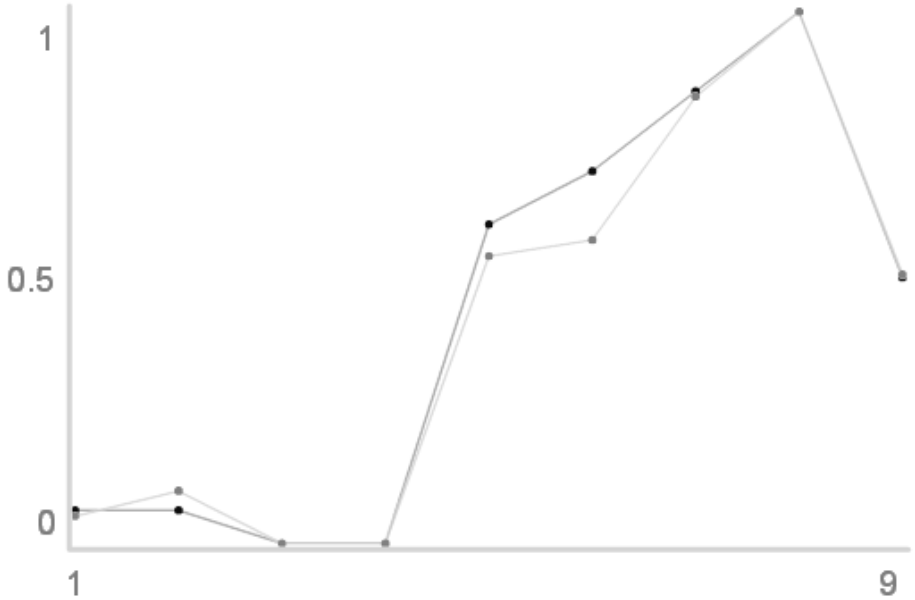
Fig31. *Le réseau utilisé pour les deux premières courbe*

Suite à ces résultats positifs, nous avons continué l'entraînement du réseau en y ajoutant d'avantage de bleu et de rouge, puis en supprimant le rouge progressivement, sur le dernier pixel, nous avons remis le rouge à son maximum (100%) et passé le bleu et le vert au quart de cette valeur (25%), ce qui nous a permis d'obtenir la courbe suivante :



La courbe évolue donc toujours correctement puisque confondue en tout point avec la courbe ciblée.

Le réseau étant parfaitement entraîné, nous avons décidé de lui soumettre des valeurs qui ne lui ont pas été soumises durant son entraînement, afin d'observer son comportement. Ce test s'avérait donc décisif :



Si nous pouvons observer de légers écarts entre les deux courbes, ces derniers restent minimes.

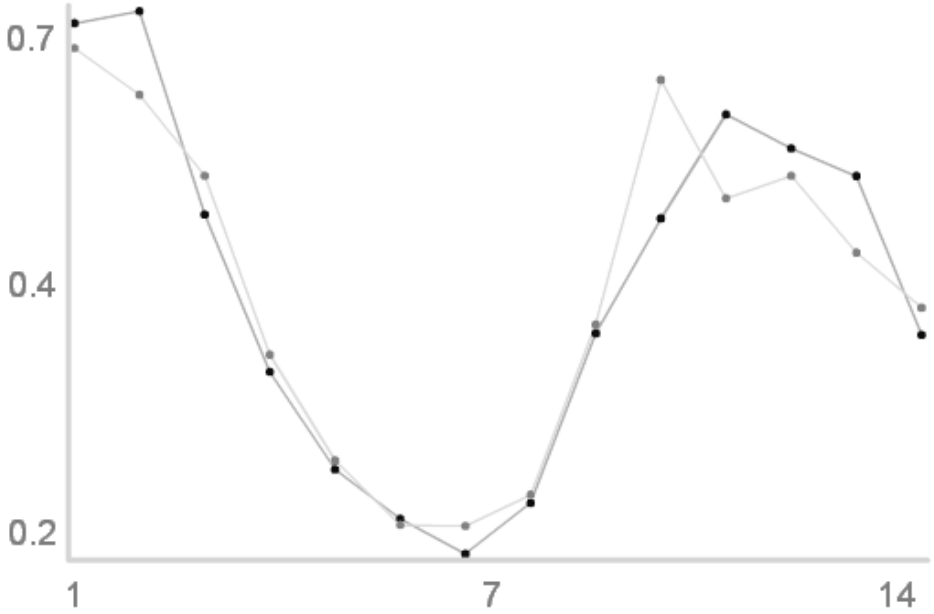
Il est intéressant de constater que la répartition des poids entre les différents neurones du réseau n'a pas énormément évoluée tout au long de l'entraînement, le réseau de la figure 31 reste parfaitement représentatif.

8.2.3 Test B - Déroulement et observations

Le test A s'étant parfaitement déroulé avec la procédure utilisée, nous avons fait le choix d'employer la même pour ce second test. Nous utiliserons toutefois un réseau à quatre entrées cette fois, afin de gérer les différents spectres audibles par le PNJ en même temps, et une couche cachée composée de cinq neurones.

8.2.4 Test B - Résultats

Après l'usage de la procédure utilisée pour le Test A, nous obtenons comme courbe finale pour le test B, la courbe suivante :



Une fois de plus, nous constatons quelques écarts, parfois important, toutefois, les deux courbes restent relativement proches, il est probable qu'avec un entraînement plus précis nous puissions limiter le nombre et la valeur de ces quelques écarts.

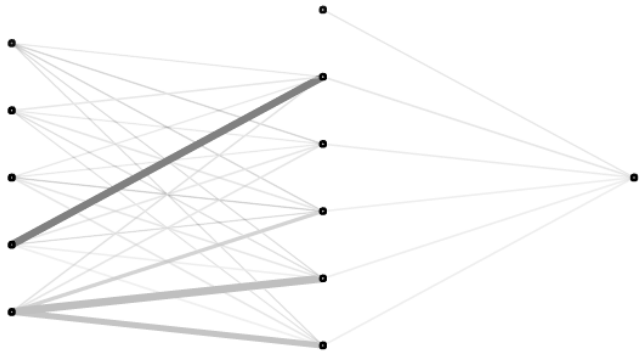


Fig32. *Réseau utilisé pour le test B*

Ce Perceptron multicouches s'avère ainsi, parfaitement correspondre à nos attentes.

8.2.5 Test C - Déroulement et observations

Si nous faisons le choix d'utiliser un réseau ART, c'est pour exploiter sa capacité de classification que nous souhaitons utiliser afin de classer les décisions. L'idée étant que les décisions sont rassemblées par groupes, représentées par des motifs. Ils sont tous successivement soumis au réseau et ce dernier attribuera à chacun l'objectif étant de voir si le réseau maintient cette attribution dans le temps ou pas. Pour le vérifier, nous lui soumettrons plusieurs fois chaque motif.

Nous avons par conséquent soumis différents motifs au réseau, en commençant par 10, pour 10 classes. Nous avons soumis ces différents motifs plusieurs fois (au moins deux fois chacun, parfois plus), et avons constaté la modification de la classification puis une stabilisation. Nous avons alors augmenté le nombre de classe et rajouté des motifs afin d'observer si la classification première serait chamboulée ou si au contraire, les nouveaux motifs se verraient attribués de nouvelles classes. Pour la seconde partie de ce test, nous restons sur notre utilisation du Perceptron Multicouches à rétropropagation. Le principe à mettre en œuvre étant essentiellement un travail d'expert : selon le entrées une sortie doit être sélectionnée.

Dans un contexte réel, un expert (dans le cadre du développement d'un agent intelligent pour un PNJ, il s'agira du développeur) est chargé d'étudier les sorties qui doivent être choisies selon le contexte présenté en entrée, tout en prenant en considération l'aspect généralisation qui sera effectué, l'idée étant d'éviter à l'expert de donner une réponse à toutes situations et de permettre à l'intelligence artificielle de déduire elle-même selon ce qu'elle aura appris. La phase d'entraînement s'avérera donc complexe à effectuer. Nous avions d'abord choisi un réseau complexe, considérant que le nombre de connexions et de poids pourraient permettre d'obtenir un plus grand nombre de sorties possible selon les valeurs d'entrées. Nous avions également disposés deux sorties à ce réseau, ainsi que quatre entrées. Toutefois, tous nos essais se sont soldés par un échec, les sorties passant rapidement toutes les deux à zéro avec un motif d'entraînement pourtant assez simple

Nous avons donc réorienté notre choix vers un réseau plus modeste composé de deux couches cachées, contenant chacune quatre neurones, une unique sortie, et quatre entrées permettant de recevoir les nombreuses informations de l'environnement.

Nos premiers essais devaient nous permettre d'obtenir des valeurs tantôt positives tantôt négatives selon les valeurs d'entrées. Rapidement, le réseau s'est bloqué n'oscillant que très peu vers les valeurs ciblées, nous avons donc revu ce choix de valeurs relatives, l'orientant vers une variation de valeur décimales : 0,001 ; 0,01 ; 0,1 ; 1.

8.2.6 Test C - Résultats

Réseau **ART** :

Lors de notre premier essai, composé de 6 motifs nous avons observé la classification suivante :

```
[   O ] -> Classe 0
[  O O] -> Classe 1
[    O] -> Classe 1
[  O O] -> Classe 2
[    O] -> Classe 1
[  O O] -> Classe 2
[    O] -> Classe 1
[ OO O] -> Classe 3
[ OO  ] -> Classe 3
[ OO O] -> Classe 4
[ OO  ] -> Classe 3
```

On peut y constater l'évolution de la classification, puis ce qui semble être une stabilisation pour chacune des classes, afin d'en être sûr, nous avons étendus le test en y ajoutant des motifs et des classes :

```
[   o ] -> Classe 0  [oo   ] -> Classe 6
[  o o] -> Classe 1  [ooo  ] -> Classe 7
[    o] -> Classe 1  [oooo ] -> Classe 8
[  o o] -> Classe 2  [ooooo] -> Classe 9
[    o] -> Classe 1  [o    ] -> Classe 5
[  o o] -> Classe 2  [ o   ] -> Classe 3
[    o] -> Classe 1  [  o  ] -> Classe 2
[ oo o] -> Classe 3  [   o ] -> Classe 0
[ oo  ] -> Classe 3  [    o] -> Classe 1
[ oo o] -> Classe 4  [  o o] -> Classe 4
[ oo  ] -> Classe 3  [ oo o] -> Classe 9
[ooo  ] -> Classe 5  [ oo  ] -> Classe 7
[oo   ] -> Classe 5  [ooo  ] -> Classe 8
[o    ] -> Classe 5  [oo   ] -> Classe 6
```

Si nous pouvons une nouvelle fois observer la stabilisation des premières classes, le réseau semble encore hésitant sur la classification de certains nouveaux motifs. Nous décidons donc d'effectuer un dernier passage en ajoutant 4 motifs que nous répéterons plusieurs fois afin de valider leur classification, tout en insérant d'anciens motifs afin d'observer d'éventuelles perturbations :

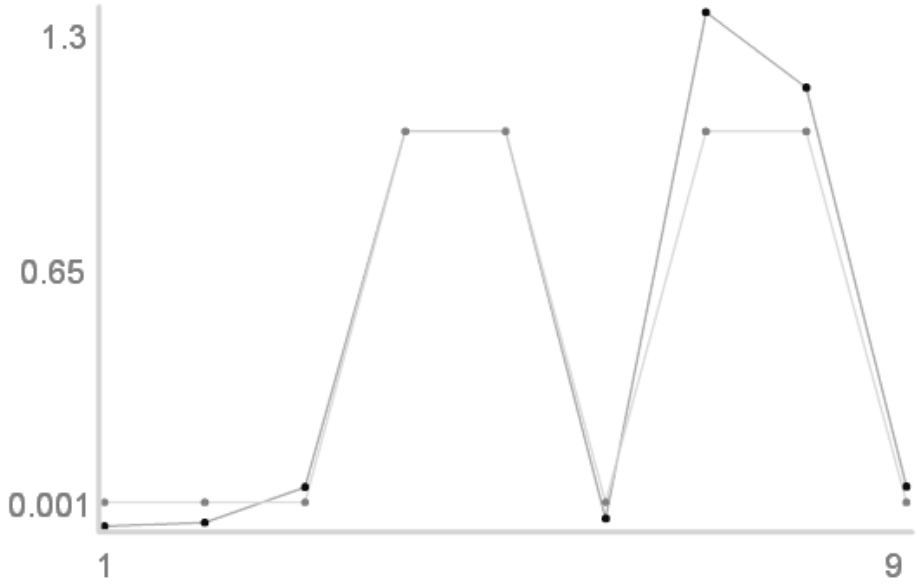
```
[   o ] -> Classe 0
[  o o] -> Classe 1
[    o] -> Classe 1
[  o o] -> Classe 2
[    o] -> Classe 1
[  o o] -> Classe 2
[    o] -> Classe 1
[ oo o] -> Classe 3
[ oo  ] -> Classe 3
[ oo o] -> Classe 4
[ oo  ] -> Classe 3
[ooo  ] -> Classe 5
[oo   ] -> Classe 5
[o    ] -> Classe 5
[oo   ] -> Classe 6
[ooo  ] -> Classe 7
[oooo ] -> Classe 8
[ooooo] -> Classe 9
[o    ] -> Classe 5
[ o   ] -> Classe 3
[  o  ] -> Classe 2
[   o ] -> Classe 0
[    o] -> Classe 1
[  o o] -> Classe 4
[ oo o] -> Classe 9
[ oo  ] -> Classe 7
[ooo  ] -> Classe 8
[oo   ] -> Classe 6
[oooo ] -> Classe 10
[ooooo] -> Classe 11
[ oo  ] -> Classe 7
[o   o] -> Classe 11
[ooo o] -> Classe 12
[ooooo] -> Classe 13
[ooo  ] -> Classe 8
[ooooo] -> Classe 13
[ oo o] -> Classe 9
[ooooo] -> Classe 13
[o   o] -> Classe 11
[oo   ] -> Classe 6
[ooo o] -> Classe 12
[oooo ] -> Classe 10
```

Ce dernier cycle est un succès puisque chaque motif est assigné à une classe stabilisée. Nous pouvons donc conclure cette partie du test sur un succès du réseau ART pour la classification de motifs représentant des données et des décisions.

Réseau **PMC** :

Comme nous l'avions indiqué plus tôt, l'objectif du Perceptron Multicouches et de permettre le traitement d'information afin de prendre des décisions, l'étape la plus importante étant donc d'entraîner correctement le réseau afin que ce dernier puisse prendre les décisions attendues grâce à son principe de généralisation.

Le premier essai effectué nous permet d'obtenir la courbe suivante.



Nous pouvons observer ci-dessous l'état du réseau et la répartition des poids au sein des différentes connexions :

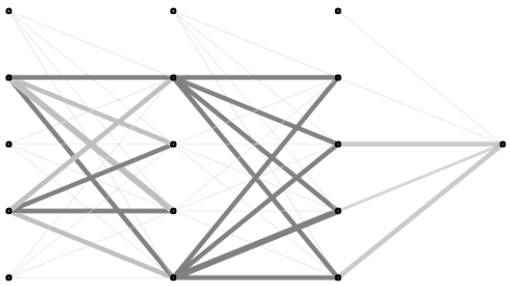
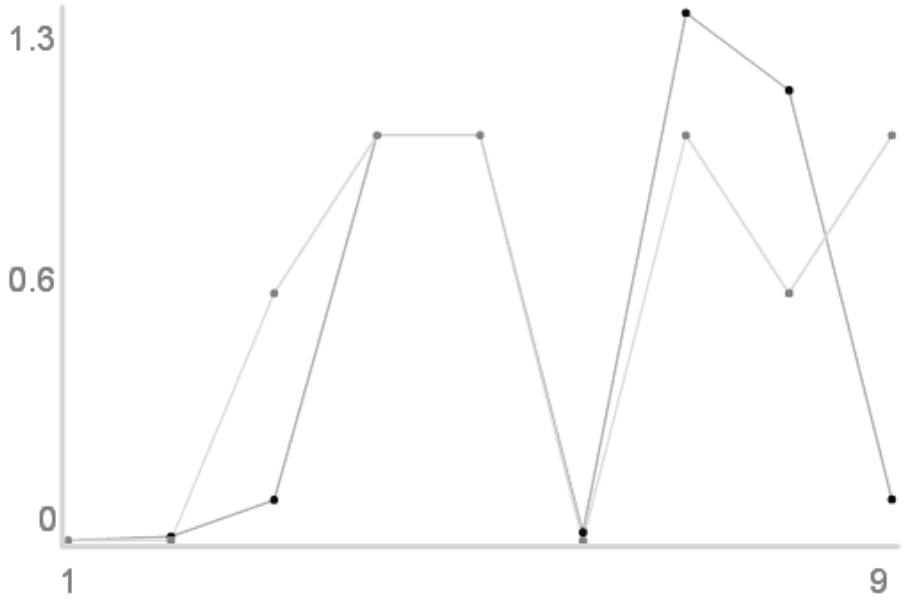


Fig33. *PMC utilisé pour la partie C – 1ère étape*

Courbe qui, si elle est intéressante, semble manquer de précisions sur les valeurs faibles. Nous décidons donc d'appuyer notre entraînement sur ces valeurs en proposant au réseau de nouveaux motifs basé sur des valeurs faibles ou précise (0,108) :



Si nous parvenons à corriger certains écarts, nous constatons également l'apparition de grosses erreurs.
Il est intéressante de remarquer l'évolution des poids du réseau, évolution que nous n'avions pas constaté lors de nos précédents test (A et B)

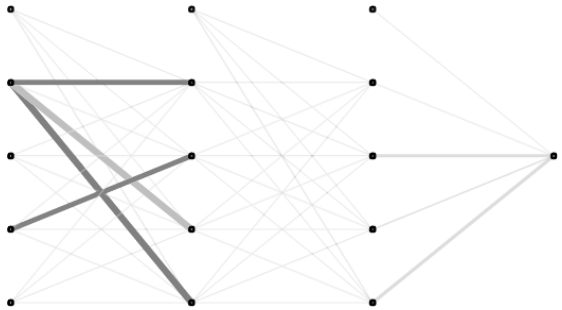
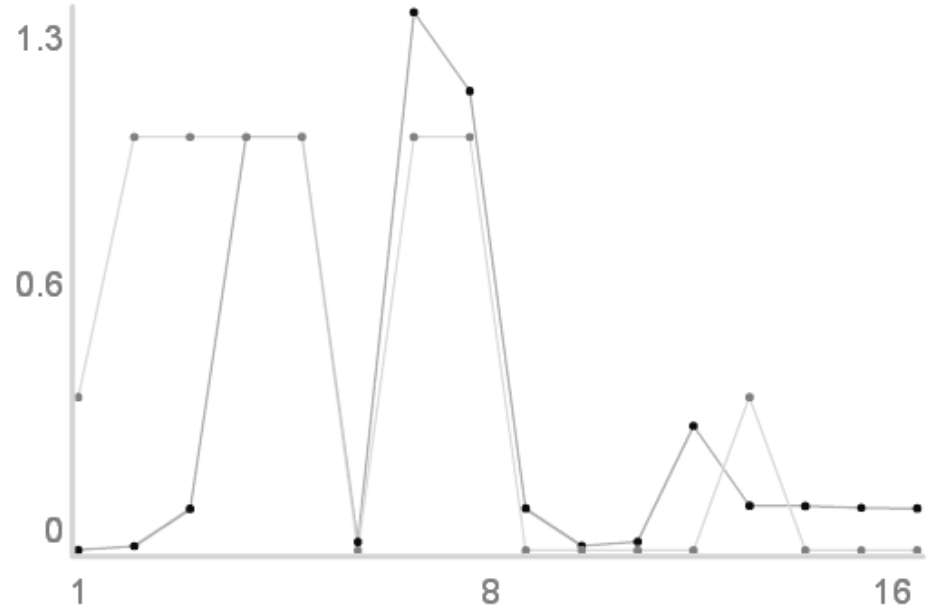


Fig34. *PMC utilisé pour la partie C – 2ème étape*

Notre nouvel essai nous permet d'obtenir la courbe suivante :



Nous constatons ainsi que plus l'entraînement est varié est complexe que de nombreuses entrées entre en jeu, plus il devient difficile d'obtenir des résultats précis et dans notre cas, utile.

Après plusieurs essais similaires non fructueux, nous décidons d'interrompre cette seconde partie du test C.

8.2.7 Test D - Déroulement et observations

Pour effectuer ce test, nous avons utilisé l'outil SOM Image (Self-Organizing Map Image) qui permet de classer un lot d'image selon leurs couleurs. Nous remarquerons en essayant l'exemple fourni avec l'outils, que ce réseau pourrait également être utilisé dans le cadre du Test A est de la détection de couleurs. Toutefois, ses phases d'apprentissages et de classification sont beaucoup trop longues pour être utilisé dans un jeu rapide tel un FPS.

Mais dans un jeu de type RTS, ce type de traitement peut tout a fait être appliqué au travers d'un thread parallèle, durant la partie ; ce qui permet à l'intelligence artificielle d'analyser la surface du champs de bataille et d'y détecter la présence d'ennemis selon l'intensité de couleurs.

Afin d'assurer l'efficacité du réseau de Kohonen, nous avons décidé d'effectuer des tests avec d'autres banques d'image et de comparer les résultats.

8.2.8 Test D – Résultats

La classification effectuée par cet outil et le modèle de Kohonen qu'il utilise, nous semble parfaite ; en recoupant nos résultats avec d'autres études **[GEN 09, JAC 02c, CIS 97]** nous sommes en mesure de confirmer l'aptitude de ce modèle pour la détection d'éléments ; son usage premier qu'est le tri est la classification l'obligeant à récupérer des informations en provenance des données qui lui sont soumises, nous pourrions tout à fait limiter l'utilisation du réseau à cette détection et y intégrer un module pour en information l'agent intelligent responsable des décisions, en aval.

8.2.9 Test E - Déroulement et observations

Pour effectuer ce test nous avions d'abord choisi d'orienter un Perceptron Multicouche vers le mode de fonctionnement d'un Time-Delay Neural Network. Toutefois, le Perceptron ne permettant pas de traiter une information dans le temps, nous avions fait le choix de lui fournir plusieurs informations temporelles en même temps ceci via plusieurs paires d'entrées (4 entrées). Essayant de cette façon d'implémenter la fonction de calcul de la longueur d'une droite ce qui permettrait d'obtenir la vitesse de déplacement (grâce au délais D entre les deux instant t et t+1 utilisés pour relever les coordonnées de l'escouade détectée via la test précédent.

Mais nos différentes tentatives pour implémenter cette fonction ont échoués. Nous avons essayé avec plusieurs réseaux :

1 Perceptron multicouche à rétropropagation avec un algorithme incrémental mettant à jour les poids entre chaque motif, ainsi qu'un algorithme biatch, mettant à jour les poids à la fin de chaque session d'entraînement uniquement.

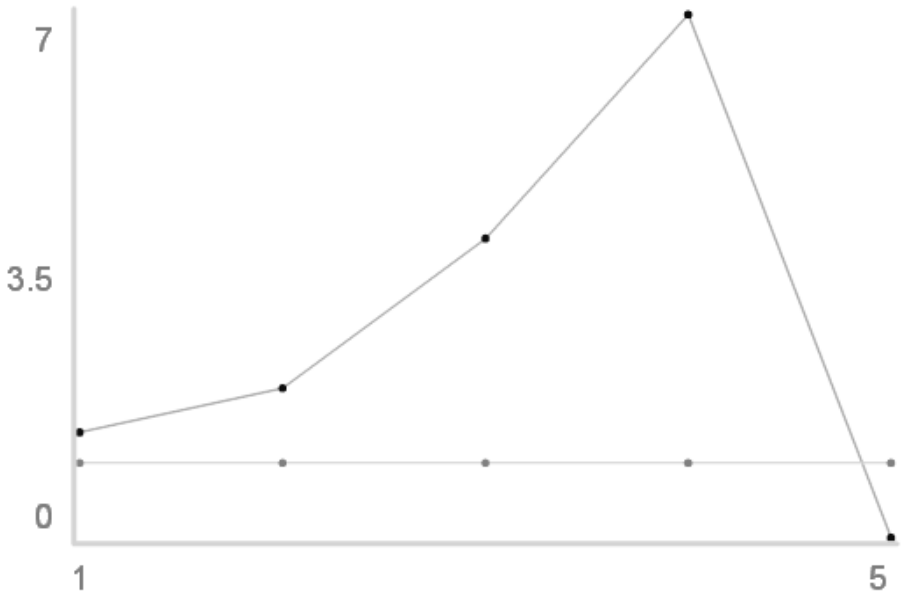
- 2 entrées, 4 neurones dans la première couche cachée, 4 neurones dans la seconde couche cachée et 1 unique neurone de sortie.

- 2 entrées, 4 neurones dans la première couche cachée, 6 neurones dans la seconde couche cachée et 1 unique neurone de sortie.

- 2 entrées, 4 neurones dans la première couche cachée, 6 neurones dans la seconde couche cachée, 4 neurones dans la troisième couche cachée et 1 unique neurone de sortie.

- 2 entrées, 4 neurones dans la couche cachée et 1 unique neurone de sortie.

8.2.10 Test E – Résultats



Pour chacun des réseaux et configurations utilisées, nous avons obtenus, dans notre tentative de calcul de la longueur d'un segment, une valeur proche de 1. L'utilisation de plusieurs réseaux en parallèle pourrait être une solution à ce problème, afin de découper l'équation en problèmes plus simples, mais dans ce cas, il faudrait trois réseaux. Enfin, à ce niveau de la problématique, il devient préférable d'employer une fonction de base implémentée dans le code pour effectuer ce calcul plutôt que de l'assigner à plusieurs réseaux neuronaux, cela serait en effet beaucoup plus léger et avec un résultat plus fiable.

Toutefois, le succès obtenu avec l'utilisation du réseau de Kohonen nous incite à penser qu'il serait probablement préférable de centraliser la détection d'une escouade et de son mouvement, uniquement avec le réseau de Kohonen : l'évolution dans t de la position des neurones sur les axes X et Y permettraient en effet de détecter les mouvements de l'adversaire, ainsi que leur vitesse et leur orientation.

8.3 Analyse des résultats

L'ensemble de nos différents tests s'avère intéressant et suggère d'intéressantes utilisations des réseaux neuronaux :

Le test A nous a permis de mettre en avant la capacité d'un Perceptron Multicouches à rétropropagation à détecter la présence d'une composante parmi les trois composantes qui lui sont soumises. Une fois correctement entraîné, le réseau peu facilement détecter si un pixel, et donc une texture ou une image dispose majoritairement d'une composante rouge, verte ou bleue (avec 3 sorties). Si l'entraînement est assez long, il peut toujours être effectué durant le développement, en temps réel, seul la détection sera nécessaire, et elle est suffisamment légère pour être intégrée à un jeu d'action.

Cette aptitude du PMC à la détection de composantes parmi d'autres a également être mise en avant avec succès lors du Test B. Il y avait en effet, peu de chances que cela fonctionne dans le premier cas mais pas dans le second. Quoi qu'il en soit, cela ouvre des portes déjà entre ouvertes, sur l'utilisation de Perceptron multicouches pour la perception d'informations bruitées. Les informations extraites d'un jeu vidéo étant souvent bruitées le PMC dispose d'une capacité intéressante qu'il serait intéressant d'exploiter.

La classification mise en avant durant le Test C, au moyen d'un réseau ART devrait, correctement utilisée, permettre d'exploiter plus facilement les informations fournies à un réseau Bayésien. Classer les informations reçues par l'environnement au moyen d'un réseau ART puis les fournir au réseau Bayésien, réseau qui dispose des même régles de classification, permettrait d'optimiser la qualité des informations perçues par le réseau et ainsi, d'augmenter sensiblement ses performances et la qualité de ses décisions.
Toutefois, notre essai avec le PMC pour remplacer un réseau Bayésien, n'est pas particulièrement prometteur, cependant, il est possible qu'en étudiant tout particulièrement la phase d'entraînement du réseau on puisse obtenir des résultats de meilleure qualité. C'est un problème qui nécessiterait une étude spécifique et approfondie.

Le modèle de Kohonen est pour nous un franc succès. Il est un réseau qui selon nous, devrait d'avantage être exploité dans les jeux vidéo. Sa remarquable capacité de classification est efficacement utilisée dans de nombreux domaines, au premier abord nous imaginons que le mode de fonctionnement des SOM serait idéal pour un RTS comme nous l'avons remarqué lors des test D et E, mais il est probable que l'on puisse exploiter les performances de ce réseau dans d'autre types de jeux et pour d'autres usages.

Notre tentative d'intégrer le calcul de la longueur d'un segment à un PMC était une erreur, ce type de calcul serait effectué de façon beaucoup plus simple au travers d'une fonction de calcul implémentée dans le code. Les capacités du réseau de neurones ne sont pas exploitées au travers de ce type d'application. Selon notre étude le modèle de Kohonen devrait suffire à gérer tous les aspects recherchés au travers des tests D et E. Une étude plus approfondie de ce réseau et de ses applications au jeu permettrait de déterminer les limites de nos ambitions.

9. Conclusion

Nous avons mis en avant différentes capacités des réseaux de neuronaux appliqués aux jeux vidéo et si nous connaissions déjà l'efficacité de ces réseaux en terme de perception et de classification, peu d'études mettent en avant leur utilisation dans le cadre vidéo ludique que sont les jeux vidéo. Ce mémoire essaie d'ouvrir des voies d'études et suggère différentes pistes à approfondir :

Nous n'avons pas trouvé matière à exploiter la totalité des réseaux de neurones présentés dans ce mémoire, réseaux qui sont parmi les principaux réseaux neuronaux, et avons ainsi focalisé nos mises en applications sur les réseaux qui semblaient les plus aptes à aboutir sur des résultats positifs.

Les réseaux ADALINE ne sont qu'une variante du Perceptron de base, et notre choix s'est naturellement tourné vers le PMC (Perceptron MultiCouches à rétropropagation), plus récent et plus efficace, en tout point, qu'un réseau ADALINE qui sont essentiellement exploités de façon matérielle en télécommunication **[VER 09]**. De la même façon, si les réseaux ALN présentent une originalité au travers de leur utilisation de portes logique en lieu et place des fonctions d'activation traditionnelles, rien ne justifie l'exploitation de cette particularité dans le cadre des jeux vidéo. L'utilisation de réseaux de Hopfield pour la reconnaissance de motifs et donc de textures était une application intéressante à mettre en œuvre, toutefois, l'efficacité de ces réseaux a déjà été démontrées à plusieurs reprises **[FAR85]**. Le principe de fonctionnement évolutif des réseaux CasCor (Cascade Correlation) est appliqué et étudié au travers du rtNEAT et du Nero Game, il n'était pas utile de reproduire ces études dans notre mémoire. Nous n'avons pas trouvé de contexte d'application aux PNN (Probabilistic Neural Network) et GRNN (General Regression Neural Network) qui bien souvent sont utilisés pour déterminer la nature d'un élément selon les informations fournies sur son environnement.

Le Perceptron Multicouches à rétropropagation a été exploité de façon prometteuse pour la détection de composantes bruitées comme des couleurs ou du son ; correctement utilisé, ce réseau peut permettre la perception d'entités avec lesquels des interactions sont envisageables qu'il s'agissent d'ennemis, de collisions, d'alliés, de déplacements ou d'objets à récupérer. Si notre approche de la stratégie et de la planification ne s'est pas montrée fructueuse, d'autres études **[KOH 08, KOH 09]** abordant le problème d'une autre façon, maintiennent cette voie. Par ailleurs, le principe de fonctionnement des réseaux CasCor où le réseau évolue pour résoudre les problèmes qui lui sont soumis, sans faire usage de la rétropropagation considérée lourde, est une voie intéressante pour effectuer l'entraînement du réseau en temps réel, pendant le déroulement du jeu ; ce concept de faire évoluer le réseau pour l'adapter à la problématique qui lui est soumise a plusieurs fois été abordée **[RIS 06, VAL 08]** par Risto Miikulainen, Professeur à l'université du Texas et membre de l'équipe de développement du *rtNEAT*. Ses études présentent le principe de l'évolution neuronale de façon prometteuse et l'avenir des jeux vidéos est peut être tourné vers le concept de l'évolution en temps réel des réseaux de neurones **[STA 05]** afin de mieux adapter ceux-ci aux environnements toujours plus complexes de nos jeux.

Nous avons mis en avant l'efficacité du réseau ART pour toutes les applications de type classification et avons ainsi suggéré une exploitation de cette efficacité dans le cadre des jeux vidéo et plus précisément de la prise de décision associée à un réseau Bayésien **[DEN 07]**, approfondir cette approche et déterminer l'efficacité effective serait une avancée intéressante sur ces domaines complexes de l'intelligence artificielle que sont la prise de décisions et la stratégie.

Les réseaux de Kohonen sont fréquemment utilisés dans les domaines autres que le jeu vidéo, notre étude présente comment l'exploiter au sein de jeux de stratégie, jeux qui aujourd'hui souffrent souvent de problématiques liées aux capacités limitées de l'intelligence artificielle. Les SOM pourraient permettre de détecter l'acti-

vité du joueur en temps réel et d'obtenir des informations utiles à la prise de décisions stratégiques et de planifications. C'est pourquoi notre étude propose différentes utilisations de ces réseaux tout en suggérant d'effectuer des études plus approfondies des SOM appliquées aux jeux vidéo en général et aux jeux de stratégie en temps réel, plus particulièrement.

Ainsi, si les réseaux de neurones formels existent depuis plus de cinquante années maintenant, nous constatons avec notre étude que leur exploitation dans le cadre des jeux vidéo reste relativement récente et succincte, nous espérons que les pistes que nous avons ouvertes avec les PMC et les SOM mèneront vers des applications effectives et efficaces de réseaux neuronaux au sein d'intelligences artificielles de personnages non joueurs. Tout en gardant à l'esprit les nombreuses et intéressantes études effectuées par l'équipe du *Neural-Network Research Group* et leur principe de neuroévolution en temps réel qui devrait également susciter de l'intérêt pour les réseaux neuronaux dans les années à venir.

10. Références

(À mettre en page et à uniformiser)

10.1 Bibliographie

[ARB 95] Arbib, Michael (1995), The Handbook of Brain Theory and Neural Networks.

[ARM 79] William Armstrong, J. Gecsei (1979), "Adaptation Algorithms for Binary Tree Networks" IEEE Trans. on Systems, Man and Cybernetics.

[ARM 97] William Armstrong (1997), Hardware requirements for fast evaluation of functions learned by Adaptive Logic Networks.

[AND 90] Andler (1990) Connexionnisme et cognition - Paru dans Revue de Synthèse, série générale CXI, nº 1-2, janv.-juin 1990, p. 95-127

[AND 00] Anderson C.W. Draper B.A. Peterson D.A., Department of Computer Science, Colorado State University, Fort Collins, CO 80523 USA, Behavioral Cloning of Student Pilots with Modular Neural Networks

[ARI 02] Arikan, O., And Forsyth, D. A. 2002. Interactive motion generation from examples. In ACM Transactions on Graphics

[AUT 92] J.-M. Autebert (1992, Calculabilité et décidabilité

[BAH 08] Erkin Bahceci (2008), transfer of evolved pattern-based heuristics in games

[BAR 04] Anna Bartkowiak (2004), Neural networks and pattern recognition lecture notes

[BER 06] Hughes Bersini (2006), De l'intelligence humaine à l'intelligence artificielle

[BIS 96] Bishop, C. 1996. Neural Network for Pattern Recognition. Cambridge University Press

[BOR 07] Pierre Borne, Mohamed Benrejeb, Joseph Haggège (2007). Les réseaux de neurones

[BLU 92] Adam Blum (1992). Neural networks in C++

[BUH 03] M. D. Buhmann (2003), Radial Basis Functions

[CAR 04] Alain Cardon (2004). Modéliser et concevoir une machine pensante : Approche de la conscience artificielle

[CAT 06] Thanh Giang Robert Mooney Christopher Peters Carol O'Sullivan, Real-Time Character Animation Techniques, Image Synthesis Group Trinity College Dublin (2006)

[CHA 07] Rodolphe Charrier, Christine Bourjot, François Charpillet (2007) Modèle connexionniste pour l'intelligence en essaim

[CRE 93] Daniel Crevier (1993), AI: The Tumultuous Search for Artificial Intelligence

[DAR 88] Darpa. (1988). DARPA Neural Network Study. AFCEA International Press

[DAY 03] Peter Dayan, L.F. Abbott. Theoretical Neuroscience. MIT Press

[DEN 07] Jean-baptiste Denis (2007), Les réseaux Bayésiens – introduction élémentaire et intuitive

[DRE 08] Gérard Dreyfus, Jean-Marc Martinez, Mannuel Samuelides, Mirta Gordon, Fouad Badran, Sylvie Thiria (2008), "Apprentissage statistique : réseaux de neurones, cartes topologiques, machines à vecteurs supports" Eyrolles.

[DSI 07] Thomas Dsilva,Roy Janik, Michael chrien, Kenneth OStanley, Risto Miikhulainen(2007) Retaininglearned behavior during real-time neuroevolution

[DSI 09] Thomas Dsilva, Risto Miikkulainen (2009) Learning dynamic obstacle avoidance for a robot arm using neuroevolution

[FAH 91] Scott E. Fahlman,Christian Lebiere (1991) The Cascade-Correlation Learning Architecture

[FAL 09] Boi Faltings, Michael Ignaz Schumacher (2009), L'intelligence Artificielle par la pratique

[FAR 85] Fahrat, N. H., Psaltis, D., Prata, A., & Paek, E. (1985). Optical implementation of the Hopfield model

[FID 08] Peggy Fidelman, Thayne Coffman, Risto Miikkulainen(2008), Detecting Motion in the Environment with a moving quadruped robot

[FIE 06] Cyril Fiévet, Philippe Bultez Adams, Axel Kahn (2006), Robots extraordinaires

[FIX 08] Jeremy Fix (2008) Mécanismes numériques et distribués de l'anticipation motrice

[FIX 08] Jeremy Fix (2008) Mécanismes numériques et distribués de l'anticipation motrice (sliders de la soutenance)

[FUK 75] Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network.

[FUK 88] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition.

[GAB 04] Robert, Gabriel (2005). MHiCS, une architecture de sélection de l'action Motivationnelle et Hiérarchique à Systèmes de Classeurs pour Personnages Non Joueurs adaptatifs"

[GOL 05] Aliza Gold (2005) Academic AI and Video Games

[GOM 08] Faustino Gomez, Jurgen Schmidhuber, Risto Miikkulainen (2008)Accelerated Neural Evolution through Cooperatibely coevolved synapses

[GUI 08] David Guiraud(2008) Modélisation du système sensori-moteur humain & neuroprothèses

[GRI 97] I. Gritz, J. K. Hahn. Genetic Programming Evolution of Controllers for 3-D Character Animation. Koza, J.R., et al. (editors), Genetic Programming 1997: Proceedings of the 2nd Annual Conference, 139-146, July 13-16 1997, Stanford University. San Francisco: Morgan Kaufmann. 1997.

[GRE 98] Grzeszczuk et al., "NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models", pp. 1-104, Department of Computer Science, University of Toronto (1998).

[GRO 03] Grossberg, Carpenter (2003), Adaptive Resonance Theory

[GRO 87] Grossberg (1987), Competitive learning: From interactive activation to adaptive resonance

[GRO 87a] Grossberg, Carpenter (1987), ART 2: Self-organization of stable category recognition codes for analog input patterns

[GRO 90] Grossberg, Carpenter (1990), ART 3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architecture

[GRO 91] Grossberg, Carpenter, Rosen (1991), ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition

[GRO 91a] Grossberg, Carpenter, Rosen (1991), Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system

[GRO 91b] Grossberg, Carpenter, Reynolds (1991), ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network

[GRO 92] Grossberg, Carpenter, Reynolds, Markuzon, Rosen (1992) Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps

[HAY 98] Simon Haykin (1998), Neural Networks : A Comprehensive Foundation

[HAT ??] Jean-Paul Haton, Modèles connexionnistes pour l'intelligence artificielle

[HAS 95] Mohammad Hassoun (1995), Fundamentals of artificials neural networks

[HEB 49] Hebb, D. O. (1949). The Organization of Behaviour. New York: Wile

[HER 91] J.Hertz, A.Krogh, R.G.Palmer (1991), Introduction to the theory of neural computation

[HOD ??] Jessica K. Hodgins, James F. O'Brien, and Robert E. Bodenheimer, Jr., Computer Animation

[HOD 07] Jessica K. Hodgins (2007), Contraint-based Motion Optimization Using a statistical Dynamic Model

[HOP 82] J.J Hopfield (1982), Neural Networks and Physical Systems with Emergent Collective Computational Abilities

[HOP 84] J.J Hopfield (1984), Neurons with graded response have collective computational properties like those of two-state neurons

[HOP 85] J.J Hopfiel, D.W. Tank (1985), Neural computation of decisions in optimization problems

[HOU 95a] Houk, J.C., Adams, J.L. & Barto, A.G. (1995), A Model of how the Basal Ganglia generate and Use Neural Signals That Predict Reinforcement.

[HOU 95b] Houk et al. (1995), Models of Information Processing in the Basal Ganglia. The MIT Press, Cambridge, MA.

[IGN 91] James Ignizio (1991), Introduction to Expert Systems

[JAC 02a] Christian Jacob(2002), Artificial Neural Network (Introduction and Perceptron learning)

[JAC 02b] Christian Jacob (2002), Feedforward Networks

[JAC 02c] Christian Jacob (2002), Self-Organizing Feature Maps

[KUR 07] Ray Kurzweil, Adeline Mesmin (2007) Humanité 2.0 - La bible du Changement

[KOH 08] Nate Kohl, Riso Miikkulainen(2008), Evolving Neural Netowrks for Fractured Domains

[KOH 09] Nate Kohl, Risto Miikkulainen(2007), Evolving neural networks for strategic decision-making problem

[KOH 82] T.Kohonen (1982), "Self-Organized Formation of Topologically Correct Feature Maps", Biological Cybernetics vol. 46, pages 59–69

[KOH 95] T.Kohonen (1995), Self-Organizing Maps

[KRO 96] B.Krose, P. Van Der Smagt (1996), An Introduction to Neural Networks - 8ème édition

[LEC 08] P.Le Callet, C.Viard-Gaudin, D.Barba (2008), A convolutional neural network approach for objective video quality assesment

[LED 92] Hubert Lederer Dreyfus (1992), Intelligence Artificielle

[LIO 04] Lioret Alain, Émergence de nouvelles esthétiques du mouvement, Paris, L'Harmattan, coll. Champs Vallon, 2004

[LIP 87] Richard P. Lippman (1987)/ An Introduction to Computing with Neural Nets

[LOC 07] Alan Lockette, Charles Chen, Risot Miikkulainen (2007), Evolving Explicit Opponent Models in Game Playing

[LYO 90] – Les entretiens de Lyon (1990) - Neural Networks : biological computers or electronic brains

[MAC 03] MacKay, David (2003). Information Theory, Inference, and Learning Algorithms

[MAE 96] George Maestri, Principles of Traditional Animation Applied to 3D Computer Animation

[MCC 43] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervousactivity.

[MCC 59] W. McCulloch / W.Pitts (1959). What the frog's eye tells the frog's brain

[MEY 02] Jean-Arcady Meyer, Agnes Guillot (2002), Vers une robotique animale

[MIK 06] Risto Miikkulainen(2006), Computational Intelligence in Games

[MIK 06] Risto Miikkulainen(2006), Creating intelligent Agents in Games

[MIN 69] M.Minsky / S.Papert (1969), Perceptrons : an introduction to computational geometry

[MIT 97] Tom M.Mitchell (1997), Machine learning

[MOK 98] Mohand Mokhtari, Michel Marie (1998), Applications de MATLAB 5 et SIMULINK 2 : Contrôle de procédés, Logique floue, Réseaux de neurones, Traitement du signal.

[NEU 53] John von Neumann, Oskar Morgenstern (1953), The Theory of Games and Economic Behavior

[ODO 04] O'Doherty J., Dayan P., Schultz J., Deichmann R., Friston K. & Dolan R. (2004), Dissociable Roles of Dorsal and Ventral Striatum in Instrumental Conditioning.

[OGI 08] Jean Marc Ogier (2008), Les cartes de Kohonen

[OLI 97] Juan P. Oliver, André Fonseca de Oliveira, Julio Pérez Acle,Roberto J. de la Vega , Rafael Canetti (1997) Implementation of Adaptive Logic Networks on an FPGA board

[PER 03] Léon Personnaz et Isabelle Rivals, Réseaux de neurones formels pour la modélisation, la commande et la classification, CNRS Editions, 2003

[PAR 04] Marc Parizeau Réseaux de neurones (2004)

[PAR 04a] Marc Parizeau (2004), Le perceptron multicouche et son algorithme de retropropagation des erreurs

[PAR 06] Claude Parthenay (2006), Herbert Simon : rationalité limitée, théorie des organisations et sciences de l'artificiel

[RED 99] Redgrave, P., Prescott, T.J. & Gurney, K. (1999); The basal ganglia: a vertebrate solution to the selection problem ?

[RUM 86] Rumelhart, D. E., Hinton, McClelland, and Williams, R. J. (1986) “Learning Internal Representations by Error Propagation” Parallel Distributed Processing: Explorations in the Microstructure of Cognition

[RUS 06] Stuart Russell, Peter Norvig (2006), Intelligence Artificielle

[SCH 06] Georges Schutz, (2006), Adaptations et applicationde modèles mixtes de réseaux de neurones à processus industriel

[SCH 08] Jacob Schrum, Risto Miikkulainen (2008), Constructing Complex NPC Behavior via Multi-Objective Neuroevolution

[SCH 95] M. Schenkel, I.Guyon, D.Henderson (1995), On-Line cursive script recognition using Time Delay Neural Networks and Hidden Markov Models - Machine vision application.

[SEA 95] John Searle, (1995) La redécouverte de l'esprit

[SIL 90] Silva, F. M., & Almeida, L. B. (1990). Speeding up backpropagation

[SIM 91] K. Sims. Artificial evolution for computer graphics. In Computer Graphics (SIGGRAPH 91 Conf. Proc.), volume 25, pages 319-328, Las Vegas, Nevada, July 1991

[SIM 94a] K. Sims. Evolving virtual creatures. In SIGGRAPH 94 Conf. Proc., pages 15-22, Orlando, Florida, July 1994

[SIM 94b] K. Sims. Evolving 3D Morphology and Behavior by Competition. In Artificial Life IV Proceedings, ed. by R. Brooks & P. Maes, MIT Press, 1994, pages 28-39

[SIT 09] Yiu Fai Sit, Risto Miikkulainen(2009) Computational Predictions on the Receptive Fields and Organization of V2 for Shape Processing

[STA 05] Kenneth OStanley, Bobby Bryant, Risto Miikkulainen(2005), Evolving Neural Network Agents in the NERO Video Game

[STA 05] Kenneth OStanley, Bobby Bryant, Risto Miikkulainen(2005), Real-time Neuroevolution in the NERO Video Game

[STA 05] Kenneth OStanley, Ryan Cornelius, Risto Miikkulainen, Tomas DSilva, Aliza Gold (2005) Real-Time learning in the NERO Video Game

[STA 06] Kenneth Ostanley,Bobby Bryant, Igor Karpov, Risto Miikkulainen(2006), Realt-Time Evolution of Neural Networks in the NERO Video Game

[SUT 92] Sutton, R. S., Barto, A., & Wilson, R. (1992). Reinforcement learning is direct adaptive optimal control

[TER 94] Demetri Terzopoulos, Xiaoyuan Tu, and Radek Grzeszczuk (1994), Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World

[TOU 92] Claude Touzet (1992), Les réseaux de neurones artificiels Introduction au connexionnisme

[TUR 38] Turing Allan (1939), Systems of logic defined by ordinals

[TUR 97] Turhan M., KOHONEN'S SELF ORGANIZING NETWORKS WITH "CONSCIENCE"

[TVE 08] Tal Tversky (2008), Motion Perception and the Scene Statistics of Motion

[ULA 58] Stanislaw Ulam (1958), Tribute to John Von Neuman

[VAL 08] Vinod Valsalam, Risto Miikkulainen(2008) Modular Neuroevolution of Multilegged Locomotion

[VAR 97] Francisco J Varela (1997), Invitation aux sciences cognitives

[VER 09] Adrien Verger (2009), Réseaux de neurones artificiels

[VIE 04] Emmanuel Viennet (2004), "Fonctions de base radiale" Apprentissage connexionniste

[WAI 89] A.Waibel, T.Hanazawa, G.Hinton, K.Shikan, K.Lang (1989), Phoneme recognition using time delay neural network - IEEE Trans on acoustic, speech and signal processing

[WIL 01] Robert W. Williams,Karl Herrup (2001), The Control of Neuron Number

[YON 05] Chern Han Yong, Kenneth OStanley, Risto Miikkulainen(2005), Incorporated Advice into Evolution of Neural Networks

[ZLA 03] Zlateva J. Todorov G., BAM (Bi-directional Associative Memory) Neural Network Simulator

10.2 Webographie

[BRE 09] http://www-inrev.univ-paris8.fr/extras/Michel-Bret/cours/ (visité au 28/04/2009)

[CIS 97] http://www.cis.hut.fi/research/som-research/worldmap.html (visité au 22/08/2009)

[FAN 09] http://leenissen.dk/fann/ (visité au 21/08/2009)

[GEN 09] http://www.generation5.org/content/2004/kohonenApplications.asp (visité au 22/08/2009)

[GAC 09] http://www.cs.unc.edu/~helser/291/Animation_using_GAs.htm (visité au 28/04/2009)

[HAR 09] http://micro.seas.harvard.edu/research.html (visité au 01/08/2009)

[JOY 09] http://www.joystiq.com/2009/06/02/taking-a-walk-with-milo-molyneuxs-project-natal-game/ (visité au 01/08/2009)

[NAS 09] http://ti.arc.nasa.gov/project/remote-agent/ (visité au 28/07/2009)

[NNR 09] http://nn.cs.utexas.edu/ (visité au 21/08/2009)

[SKE 09] http://cg.skeelogy.com/aifighter (visité au 28/04/2009)

[SPD 09a] http://www.scholarpedia.org/article/Kohonen_network (visité le 28/04/2009)

[SPD 09b] http://www.scholarpedia.org/article/Hopfield_network (visité le 28/04/2009)

[STE 04] http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html (visité au 12/09/2009 site présentant les travaux de C.Stergiou et D.Siganos)

[TRA 09] http://tralvex.com/pub/nap (visité le 28/04/2009)

[WIK 09a] http://en.wikipedia.org/wiki/Neural_network (visité au 12/09/2009)

[WIK 09b] http://fr.wikipedia.org/wiki/Perceptron (visité au 28/04/2009)

[WIK 09c] http://en.wikipedia.org/wiki/Hopfield_net (visité au 22/08/2009)

[WIK 09d] http://fr.wikipedia.org/wiki/R%C3%A9seaux_de_neurones (visité au 22/08/2009)

[WIK 09e] http://fr.wikipedia.org/wiki/Carte_auto_adaptative (visité au 22/08/2009)

[WIK 09f] http://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_de_Hopfield (visité au 22/08/2009)

10.3 Vidéos

[AEV 03] Streeter T., Artificial Evolution of Humanoid Standing Behavior http://video.google.com/videoplay?docid=-2510462304066175045

[CPH 05] Computer History Museum, Pixar - A Human Story of Computer Animation http://www.youtube.com/watch?v=YjSExqtiIyg

[GTT 07] Google Tech Talks November, 29 2007, The Next Generation of Neural Networks, http://www.youtube.com/watch?v=AyzOUbkUf3M&hl=fr

[SEG 09] Matthieu Segret, Intelligence artificielle : les réseaux de neurones
http://matthieusegret.com/ia-intelligence-artificielle-reseaux-neurones

[SIM 94] Karl Sims, Evolved Virtual Creatures, Evolution Simulation, 1994 http://www.youtube.com/watch?v=JBgG_VSP7f8

10.4 Jeux Vidéo

[BAT 96] Battlecruiser: 3000AD, 3000AD (1996), Battlecruiser: 3000AD, par Take 2 Interactive Software

[CRE 90] Creatures (1990) : jeu vidéo dans lequel des créatures évoluent dans un monde virtuel

[KEE 97] Dungeon Keeper, Bullfrog, Peter Molyneux (1997) le joueur incarne le gardien maléfique des souterrains d'un donjon (en anglais, dungeon keeper) peuplé de créatures belliqueuses soumises à ses ordres. Le jeu dispose d'un mode "Assistant" où l'ordinateur va assister le joueur dans la construction de son donjon, ceci, de différentes façons (défensif, constructeur, agressif, équilibré) et selon la façon de jouer du joueur, par ailleurs les autres gardiens du jeu semblent fonctionner à partir d'un système qualifié de "behavioral cloning" dont le principe de fonctionnement : apprendre à partir du joueur puis application, se rapproche de celui des réseaux de neurones.

[B&W 01] Black & White, Lionhead, Peter Molyneux (2001) : Jeu vidéo dans lequel le joueur incarne un dieu qui fait évoluer une créature dont le comportement change selon son éducation.

[B&W 03] Black & White 2, Lionhead, Peter Molyneux (2005) : Second opus du jeu précédent.

11. Annexes

Annexe A – Perceptron à rétro propagation

```
/******************************************************************************

                    ====================================================
        Network:    Backpropagation Network with Bias Terms and Momentum
                    ====================================================

        Application:  Time-Series Forecasting
                      Prediction of the Annual Number of Sunspots

        Author:       Karsten Kutza
        Date:         17.4.96

        Reference:    D.E. Rumelhart, G.E. Hinton, R.J. Williams
                      Learning Internal Representations by Error Propagation
                      in:
                      D.E. Rumelhart, J.L. McClelland (Eds.)
                      Parallel Distributed Processing, Volume 1
                      MIT Press, Cambridge, MA, pp. 318-362, 1986

 ******************************************************************************/



/******************************************************************************
                             D E C L A R A T I O N S
 ******************************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <math.h>


typedef int           BOOL;
typedef int           INT;
typedef double        REAL;

#define FALSE         0
#define TRUE          1
#define NOT           !
#define AND           &&
#define OR            ||

#define MIN_REAL      -HUGE_VAL
#define MAX_REAL      +HUGE_VAL
#define MIN(x,y)      ((x)<(y) ? (x) : (y))
#define MAX(x,y)      ((x)>(y) ? (x) : (y))

#define LO            0.1
#define HI            0.9
#define BIAS          1

#define sqr(x)        ((x)*(x))
```

```
typedef struct {                    /* A LAYER OF A NET:                     */
        INT           Units;        /* - number of units in this layer       */
        REAL*         Output;       /* - output of ith unit                  */
        REAL*         Error;        /* - error term of ith unit              */
        REAL**        Weight;       /* - connection weights to ith unit      */
        REAL**        WeightSave;   /* - saved weights for stopped training  */
        REAL**        dWeight;      /* - last weight deltas for momentum     */
} LAYER;

typedef struct {                    /* A NET:                                */
        LAYER**       Layer;        /* - layers of this net                  */
        LAYER*        InputLayer;   /* - input layer                         */
        LAYER*        OutputLayer;  /* - output layer                        */
        REAL          Alpha;        /* - momentum factor                     */
        REAL          Eta;          /* - learning rate                       */
        REAL          Gain;         /* - gain of sigmoid function            */
        REAL          Error;        /* - total net error                     */
} NET;


/******************************************************************************
        R A N D O M S   D R A W N   F R O M   D I S T R I B U T I O N S
 ******************************************************************************/


void InitializeRandoms()
{
  srand(4711);
}


INT RandomEqualINT(INT Low, INT High)
{
  return rand() % (High-Low+1) + Low;
}


REAL RandomEqualREAL(REAL Low, REAL High)
{
  return ((REAL) rand() / RAND_MAX) * (High-Low) + Low;
}


/******************************************************************************
             A P P L I C A T I O N - S P E C I F I C   C O D E
 ******************************************************************************/


#define NUM_LAYERS    3
#define N             30
#define M             1
INT                   Units[NUM_LAYERS] = {N, 10, M};

#define FIRST_YEAR    1700
#define NUM_YEARS     280

#define TRAIN_LWB     (N)
#define TRAIN_UPB     (179)
#define TRAIN_YEARS   (TRAIN_UPB - TRAIN_LWB + 1)
#define TEST_LWB      (180)
```

```
#define TEST_UPB      (259)
#define TEST_YEARS    (TEST_UPB - TEST_LWB + 1)
#define EVAL_LWB      (260)
#define EVAL_UPB      (NUM_YEARS - 1)
#define EVAL_YEARS    (EVAL_UPB - EVAL_LWB + 1)

REAL                  Sunspots_[NUM_YEARS];
REAL                  Sunspots [NUM_YEARS] = {

                        0.0262,  0.0575,  0.0837,  0.1203,  0.1883,  0.3033,
                        0.1517,  0.1046,  0.0523,  0.0418,  0.0157,  0.0000,
                        0.0000,  0.0105,  0.0575,  0.1412,  0.2458,  0.3295,
                        0.3138,  0.2040,  0.1464,  0.1360,  0.1151,  0.0575,
                        0.1098,  0.2092,  0.4079,  0.6381,  0.5387,  0.3818,
                        0.2458,  0.1831,  0.0575,  0.0262,  0.0837,  0.1778,
                        0.3661,  0.4236,  0.5805,  0.5282,  0.3818,  0.2092,
                        0.1046,  0.0837,  0.0262,  0.0575,  0.1151,  0.2092,
                        0.3138,  0.4231,  0.4362,  0.2495,  0.2500,  0.1606,
                        0.0638,  0.0502,  0.0534,  0.1700,  0.2489,  0.2824,
                        0.3290,  0.4493,  0.3201,  0.2359,  0.1904,  0.1093,
                        0.0596,  0.1977,  0.3651,  0.5549,  0.5272,  0.4268,
                        0.3478,  0.1820,  0.1600,  0.0366,  0.1036,  0.4838,
                        0.8075,  0.6585,  0.4435,  0.3562,  0.2014,  0.1192,
                        0.0534,  0.1260,  0.4336,  0.6904,  0.6846,  0.6177,
                        0.4702,  0.3483,  0.3138,  0.2453,  0.2144,  0.1114,
                        0.0837,  0.0335,  0.0214,  0.0356,  0.0758,  0.1778,
                        0.2354,  0.2254,  0.2484,  0.2207,  0.1470,  0.0528,
                        0.0424,  0.0131,  0.0000,  0.0073,  0.0262,  0.0638,
                        0.0727,  0.1851,  0.2395,  0.2150,  0.1574,  0.1250,
                        0.0816,  0.0345,  0.0209,  0.0094,  0.0445,  0.0868,
                        0.1898,  0.2594,  0.3358,  0.3504,  0.3708,  0.2500,
                        0.1438,  0.0445,  0.0690,  0.2976,  0.6354,  0.7233,
                        0.5397,  0.4482,  0.3379,  0.1919,  0.1266,  0.0560,
                        0.0785,  0.2097,  0.3216,  0.5152,  0.6522,  0.5036,
                        0.3483,  0.3373,  0.2829,  0.2040,  0.1077,  0.0350,
                        0.0225,  0.1187,  0.2866,  0.4906,  0.5010,  0.4038,
                        0.3091,  0.2301,  0.2458,  0.1595,  0.0853,  0.0382,
                        0.1966,  0.3870,  0.7270,  0.5816,  0.5314,  0.3462,
                        0.2338,  0.0889,  0.0591,  0.0649,  0.0178,  0.0314,
                        0.1689,  0.2840,  0.3122,  0.3332,  0.3321,  0.2730,
                        0.1328,  0.0685,  0.0356,  0.0330,  0.0371,  0.1862,
                        0.3818,  0.4451,  0.4079,  0.3347,  0.2186,  0.1370,
                        0.1396,  0.0633,  0.0497,  0.0141,  0.0262,  0.1276,
                        0.2197,  0.3321,  0.2814,  0.3243,  0.2537,  0.2296,
                        0.0973,  0.0298,  0.0188,  0.0073,  0.0502,  0.2479,
                        0.2986,  0.5434,  0.4215,  0.3326,  0.1966,  0.1365,
                        0.0743,  0.0303,  0.0873,  0.2317,  0.3342,  0.3609,
                        0.4069,  0.3394,  0.1867,  0.1109,  0.0581,  0.0298,
                        0.0455,  0.1888,  0.4168,  0.5983,  0.5732,  0.4644,
                        0.3546,  0.2484,  0.1600,  0.0853,  0.0502,  0.1736,
                        0.4843,  0.7929,  0.7128,  0.7045,  0.4388,  0.3630,
                        0.1647,  0.0727,  0.0230,  0.1987,  0.7411,  0.9947,
                        0.9665,  0.8316,  0.5873,  0.2819,  0.1961,  0.1459,
                        0.0534,  0.0790,  0.2458,  0.4906,  0.5539,  0.5518,
                        0.5465,  0.3483,  0.3603,  0.1987,  0.1804,  0.0811,
                        0.0659,  0.1428,  0.4838,  0.8127

                      };

REAL                  Mean;
REAL                  TrainError;
```

```
REAL                    TrainErrorPredictingMean;
REAL                    TestError;
REAL                    TestErrorPredictingMean;

FILE*                   f;


void NormalizeSunspots()
{
  INT  Year;
  REAL Min, Max;

  Min = MAX_REAL;
  Max = MIN_REAL;
  for (Year=0; Year<NUM_YEARS; Year++) {
    Min = MIN(Min, Sunspots[Year]);
    Max = MAX(Max, Sunspots[Year]);
  }
  Mean = 0;
  for (Year=0; Year<NUM_YEARS; Year++) {
    Sunspots_[Year] =
    Sunspots [Year] = ((Sunspots[Year]-Min) / (Max-Min)) * (HI-LO) + LO;
    Mean += Sunspots[Year] / NUM_YEARS;
  }
}


void InitializeApplication(NET* Net)
{
  INT  Year, i;
  REAL Out, Err;

  Net->Alpha = 0.5;
  Net->Eta   = 0.05;
  Net->Gain  = 1;

  NormalizeSunspots();
  TrainErrorPredictingMean = 0;
  for (Year=TRAIN_LWB; Year<=TRAIN_UPB; Year++) {
    for (i=0; i<M; i++) {
      Out = Sunspots[Year+i];
      Err = Mean - Out;
      TrainErrorPredictingMean += 0.5 * sqr(Err);
    }
  }
  TestErrorPredictingMean = 0;
  for (Year=TEST_LWB; Year<=TEST_UPB; Year++) {
    for (i=0; i<M; i++) {
      Out = Sunspots[Year+i];
      Err = Mean - Out;
      TestErrorPredictingMean += 0.5 * sqr(Err);
    }
  }
  f = fopen("BPN.txt", "w");
}


void FinalizeApplication(NET* Net)
{
  fclose(f);
}
```

```
/******************************************************************************
                        I N I T I A L I Z A T I O N
 ******************************************************************************/


void GenerateNetwork(NET* Net)
{
  INT l,i;

  Net->Layer = (LAYER**) calloc(NUM_LAYERS, sizeof(LAYER*));

  for (l=0; l<NUM_LAYERS; l++) {
    Net->Layer[l] = (LAYER*) malloc(sizeof(LAYER));

    Net->Layer[l]->Units      = Units[l];
    Net->Layer[l]->Output     = (REAL*)  calloc(Units[l]+1, sizeof(REAL));
    Net->Layer[l]->Error      = (REAL*)  calloc(Units[l]+1, sizeof(REAL));
    Net->Layer[l]->Weight     = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->WeightSave = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->dWeight    = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->Output[0]  = BIAS;

    if (l != 0) {
      for (i=1; i<=Units[l]; i++) {
        Net->Layer[l]->Weight[i]     = (REAL*) calloc(Units[l-1]+1,
sizeof(REAL));
        Net->Layer[l]->WeightSave[i] = (REAL*) calloc(Units[l-1]+1,
sizeof(REAL));
        Net->Layer[l]->dWeight[i]    = (REAL*) calloc(Units[l-1]+1,
sizeof(REAL));
      }
    }
  }
  Net->InputLayer  = Net->Layer[0];
  Net->OutputLayer = Net->Layer[NUM_LAYERS - 1];
  Net->Alpha       = 0.9;
  Net->Eta         = 0.25;
  Net->Gain        = 1;
}


void RandomWeights(NET* Net)
{
  INT l,i,j;

  for (l=1; l<NUM_LAYERS; l++) {
    for (i=1; i<=Net->Layer[l]->Units; i++) {
      for (j=0; j<=Net->Layer[l-1]->Units; j++) {
        Net->Layer[l]->Weight[i][j] = RandomEqualREAL(-0.5, 0.5);
      }
    }
  }
}


void SetInput(NET* Net, REAL* Input)
{
  INT i;
```

```
  for (i=1; i<=Net->InputLayer->Units; i++) {
    Net->InputLayer->Output[i] = Input[i-1];
  }
}


void GetOutput(NET* Net, REAL* Output)
{
  INT i;

  for (i=1; i<=Net->OutputLayer->Units; i++) {
    Output[i-1] = Net->OutputLayer->Output[i];
  }
}


/******************************************************************************
           S U P P O R T   F O R   S T O P P E D   T R A I N I N G
 ******************************************************************************/


void SaveWeights(NET* Net)
{
  INT l,i,j;

  for (l=1; l<NUM_LAYERS; l++) {
    for (i=1; i<=Net->Layer[l]->Units; i++) {
      for (j=0; j<=Net->Layer[l-1]->Units; j++) {
        Net->Layer[l]->WeightSave[i][j] = Net->Layer[l]->Weight[i][j];
      }
    }
  }
}


void RestoreWeights(NET* Net)
{
  INT l,i,j;

  for (l=1; l<NUM_LAYERS; l++) {
    for (i=1; i<=Net->Layer[l]->Units; i++) {
      for (j=0; j<=Net->Layer[l-1]->Units; j++) {
        Net->Layer[l]->Weight[i][j] = Net->Layer[l]->WeightSave[i][j];
      }
    }
  }
}


/******************************************************************************
                    P R O P A G A T I N G   S I G N A L S
 ******************************************************************************/


void PropagateLayer(NET* Net, LAYER* Lower, LAYER* Upper)
{
  INT  i,j;
  REAL Sum;

  for (i=1; i<=Upper->Units; i++) {
    Sum = 0;
```

```
    for (j=0; j<=Lower->Units; j++) {
      Sum += Upper->Weight[i][j] * Lower->Output[j];
    }
    Upper->Output[i] = 1 / (1 + exp(-Net->Gain * Sum));
  }
}


void PropagateNet(NET* Net)
{
  INT l;

  for (l=0; l<NUM_LAYERS-1; l++) {
    PropagateLayer(Net, Net->Layer[l], Net->Layer[l+1]);
  }
}


/******************************************************************************
                 B A C K P R O P A G A T I N G   E R R O R S
 ******************************************************************************/


void ComputeOutputError(NET* Net, REAL* Target)
{
  INT  i;
  REAL Out, Err;

  Net->Error = 0;
  for (i=1; i<=Net->OutputLayer->Units; i++) {
    Out = Net->OutputLayer->Output[i];
    Err = Target[i-1]-Out;
    Net->OutputLayer->Error[i] = Net->Gain * Out * (1-Out) * Err;
    Net->Error += 0.5 * sqr(Err);
  }
}


void BackpropagateLayer(NET* Net, LAYER* Upper, LAYER* Lower)
{
  INT  i,j;
  REAL Out, Err;

  for (i=1; i<=Lower->Units; i++) {
    Out = Lower->Output[i];
    Err = 0;
    for (j=1; j<=Upper->Units; j++) {
      Err += Upper->Weight[j][i] * Upper->Error[j];
    }
    Lower->Error[i] = Net->Gain * Out * (1-Out) * Err;
  }
}


void BackpropagateNet(NET* Net)
{
  INT l;

  for (l=NUM_LAYERS-1; l>1; l--) {
    BackpropagateLayer(Net, Net->Layer[l], Net->Layer[l-1]);
  }
```

```
}


void AdjustWeights(NET* Net)
{
  INT  l,i,j;
  REAL Out, Err, dWeight;

  for (l=1; l<NUM_LAYERS; l++) {
    for (i=1; i<=Net->Layer[l]->Units; i++) {
      for (j=0; j<=Net->Layer[l-1]->Units; j++) {
        Out = Net->Layer[l-1]->Output[j];
        Err = Net->Layer[l]->Error[i];
        dWeight = Net->Layer[l]->dWeight[i][j];
        Net->Layer[l]->Weight[i][j] += Net->Eta * Err * Out + Net->Alpha *
dWeight;
        Net->Layer[l]->dWeight[i][j] = Net->Eta * Err * Out;
      }
    }
  }
}


/******************************************************************************
                    S I M U L A T I N G   T H E   N E T
 ******************************************************************************/


void SimulateNet(NET* Net, REAL* Input, REAL* Output, REAL* Target, BOOL Train-
ing)
{
  SetInput(Net, Input);
  PropagateNet(Net);
  GetOutput(Net, Output);

  ComputeOutputError(Net, Target);
  if (Training) {
    BackpropagateNet(Net);
    AdjustWeights(Net);
  }
}


void TrainNet(NET* Net, INT Epochs)
{
  INT  Year, n;
  REAL Output[M];

  for (n=0; n<Epochs*TRAIN_YEARS; n++) {
    Year = RandomEqualINT(TRAIN_LWB, TRAIN_UPB);
    SimulateNet(Net, &(Sunspots[Year-N]), Output, &(Sunspots[Year]), TRUE);
  }
}


void TestNet(NET* Net)
{
  INT  Year;
  REAL Output[M];

  TrainError = 0;
```

```
  for (Year=TRAIN_LWB; Year<=TRAIN_UPB; Year++) {
    SimulateNet(Net, &(Sunspots[Year-N]), Output, &(Sunspots[Year]), FALSE);
    TrainError += Net->Error;
  }
  TestError = 0;
  for (Year=TEST_LWB; Year<=TEST_UPB; Year++) {
    SimulateNet(Net, &(Sunspots[Year-N]), Output, &(Sunspots[Year]), FALSE);
    TestError += Net->Error;
  }
  fprintf(f, "\nNMSE is %0.3f on Training Set and %0.3f on Test Set",
             TrainError / TrainErrorPredictingMean,
             TestError / TestErrorPredictingMean);
}


void EvaluateNet(NET* Net)
{
  INT  Year;
  REAL Output [M];
  REAL Output_[M];

  fprintf(f, "\n\n\n");
  fprintf(f, "Year    Sunspots    Open-Loop Prediction    Closed-Loop
Prediction\n");
  fprintf(f, "\n");
  for (Year=EVAL_LWB; Year<=EVAL_UPB; Year++) {
    SimulateNet(Net, &(Sunspots [Year-N]), Output,  &(Sunspots [Year]), FALSE);
    SimulateNet(Net, &(Sunspots_[Year-N]), Output_, &(Sunspots_[Year]), FALSE);
    Sunspots_[Year] = Output_[0];
    fprintf(f, "%d       %0.3f                   %0.3f
%0.3f\n",
               FIRST_YEAR + Year,
               Sunspots[Year],
               Output [0],
               Output_[0]);
  }
}


/*******************************************************************************
                                    M A I N
 *******************************************************************************/


void main()
{
  NET  Net;
  BOOL Stop;
  REAL MinTestError;

  InitializeRandoms();
  GenerateNetwork(&Net);
  RandomWeights(&Net);
  InitializeApplication(&Net);

  Stop = FALSE;
  MinTestError = MAX_REAL;
  do {
    TrainNet(&Net, 10);
    TestNet(&Net);
    if (TestError < MinTestError) {
```

```
      fprintf(f, " - saving Weights ...");
      MinTestError = TestError;
      SaveWeights(&Net);
    }
    else if (TestError > 1.2 * MinTestError) {
      fprintf(f, " - stopping Training and restoring Weights ...");
      Stop = TRUE;
      RestoreWeights(&Net);
    }
  } while (NOT Stop);

  TestNet(&Net);
  EvaluateNet(&Net);

  FinalizeApplication(&Net);
}
```

Annexe B – Réseau Adaptive Resonance Theory

```
/*****************************************************************************

                        ========================
        Network:        Adaptive Resonance Theory
                        ========================

        Application:    Brain Modeling
                        Stability-Plasticity Demonstration

        Author:         Karsten Kutza
        Date:           27.6.96

        Reference:      G.A. Carpenter, S. Grossberg
                        A Massively Parallel Architecture
                        for a Self-Organizing Neural Pattern Recognition Machine
                        Computer Vision, Graphics, and Image Processing, 37,
                        pp. 54-115, 1987

 *****************************************************************************/



/*****************************************************************************
                             D E C L A R A T I O N S
 *****************************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <math.h>


typedef int             BOOL;
typedef char            CHAR;
typedef int             INT;
typedef double          REAL;

#define FALSE           0
#define TRUE            1
#define NOT             !
#define AND             &&
#define OR              ||

#define MIN_REAL        -HUGE_VAL
#define MAX_REAL        +HUGE_VAL


typedef struct {                        /* A LAYER OF A NET:                    */
        INT             Units;          /* - number of units in this layer      */
        BOOL*           Output;         /* - output of ith unit                 */
        REAL**          Weight;         /* - connection weights to ith unit     */
        BOOL*           Inhibited;      /* - inhibition status of ith F2 unit   */
} LAYER;

typedef struct {                        /* A NET:                               */
        LAYER*          F1;             /* - F1 layer                           */
        LAYER*          F2;             /* - F2 layer                           */
        INT             Winner;         /* - last winner in F2 layer            */
```

```
        REAL          A1;           /* - A parameter for F1 layer           */
        REAL          B1;           /* - B parameter for F1 layer           */
        REAL          C1;           /* - C parameter for F1 layer           */
        REAL          D1;           /* - D parameter for F1 layer           */
        REAL          L;            /* - L parameter for net                */
        REAL          Rho;          /* - vigilance parameter                */
} NET;


/************************************************************************
              A P P L I C A T I O N - S P E C I F I C   C O D E
 ************************************************************************/


#define N             5
#define M             10

#define NO_WINNER     M

#define NUM_DATA      30

CHAR                  Pattern[NUM_DATA][N] = { "   O ",
                                              "  O O",
                                              "    O",
                                              "  O O",
                                              "    O",
                                              "  O O",
                                              "    O",
                                              " OO O",
                                              " OO  ",
                                              " OO O",
                                              " OO  ",
                                              "OOO  ",
                                              "OO   ",
                                              "O    ",
                                              "OO   ",
                                              "OOO  ",
                                              "OOOO ",
                                              "OOOOO",
                                              "O    ",
                                              " O   ",
                                              "  O  ",
                                              "   O ",
                                              "    O",
                                              "  O O",
                                              " OO O",
                                              " OO  ",
                                              "OOO  ",
                                              "OO   ",
                                              "OOOO ",
                                              "OOOOO"  };

BOOL                  Input [NUM_DATA][N];
BOOL                  Output[NUM_DATA][M];

FILE*                 f;


void InitializeApplication(NET* Net)
{
  INT n,i,j;
```

```
  for (n=0; n<NUM_DATA; n++) {
    for (i=0; i<N; i++) {
      Input[n][i] = (Pattern[n][i] == 'O');
    }
  }
  Net->A1  = 1;
  Net->B1  = 1.5;
  Net->C1  = 5;
  Net->D1  = 0.9;
  Net->L   = 3;
  Net->Rho = 0.9;
  for (i=0; i<Net->F1->Units; i++) {
    for (j=0; j<Net->F2->Units; j++) {
      Net->F1->Weight[i][j] = (Net->B1 - 1) / Net->D1 + 0.2;
      Net->F2->Weight[j][i] = Net->L / (Net->L - 1 + N) - 0.1;
    }
  }
  f = fopen("ART1.txt", "w");
}


void WriteInput(NET* Net, BOOL* Input)
{
  INT i;

  for (i=0; i<N; i++) {
    fprintf(f, "%c", (Input[i]) ? 'O' : ' ');
  }
  fprintf(f, " -> ");
}


void WriteOutput(NET* Net, BOOL* Output)
{
  if (Net->Winner != NO_WINNER)
    fprintf(f, "Class %i\n", Net->Winner);
  else
    fprintf(f, "new Input and all Classes exhausted\n");
}


void FinalizeApplication(NET* Net)
{
  fclose(f);
}


/******************************************************************************
                        I N I T I A L I Z A T I O N
 ******************************************************************************/


void GenerateNetwork(NET* Net)
{
  INT i;

  Net->F1 = (LAYER*) malloc(sizeof(LAYER));
  Net->F2 = (LAYER*) malloc(sizeof(LAYER));

  Net->F1->Units     = N;
```

```
  Net->F1->Output    = (BOOL*)  calloc(N, sizeof(BOOL));
  Net->F1->Weight    = (REAL**) calloc(N, sizeof(REAL*));

  Net->F2->Units     = M;
  Net->F2->Output    = (BOOL*)  calloc(M, sizeof(BOOL));
  Net->F2->Weight    = (REAL**) calloc(M, sizeof(REAL*));
  Net->F2->Inhibited = (BOOL*)  calloc(M, sizeof(BOOL));

  for (i=0; i<N; i++) {
    Net->F1->Weight[i] = (REAL*) calloc(M, sizeof(REAL));
  }
  for (i=0; i<M; i++) {
    Net->F2->Weight[i] = (REAL*) calloc(N, sizeof(REAL));
  }
}


void SetInput(NET* Net, BOOL* Input)
{
  INT  i;
  REAL Activation;

  for (i=0; i<Net->F1->Units; i++) {
    Activation = Input[i] / (1 + Net->A1 * (Input[i] + Net->B1) + Net->C1);
    Net->F1->Output[i] = (Activation > 0);
  }
}


void GetOutput(NET* Net, BOOL* Output)
{
  INT i;

  for (i=0; i<Net->F2->Units; i++) {
    Output[i] = Net->F2->Output[i];
  }
}


/******************************************************************************
                    P R O P A G A T I N G   S I G N A L S
 ******************************************************************************/


void PropagateToF2(NET* Net)
{
  INT  i,j;
  REAL Sum, MaxOut;

  MaxOut = MIN_REAL;
  Net->Winner = NO_WINNER;
  for (i=0; i<Net->F2->Units; i++) {
    if (NOT Net->F2->Inhibited[i]) {
      Sum = 0;
      for (j=0; j<Net->F1->Units; j++) {
        Sum += Net->F2->Weight[i][j] * Net->F1->Output[j];
      }
      if (Sum > MaxOut) {
        MaxOut = Sum;
        Net->Winner = i;
      }
```

```
    }
    Net->F2->Output[i] = FALSE;
  }
  if (Net->Winner != NO_WINNER)
    Net->F2->Output[Net->Winner] = TRUE;
}


void PropagateToF1(NET* Net, BOOL* Input)
{
  INT  i;
  REAL Sum, Activation;

  for (i=0; i<Net->F1->Units; i++) {
    Sum = Net->F1->Weight[i][Net->Winner] * Net->F2->Output[Net->Winner];
    Activation = (Input[i] + Net->D1 * Sum - Net->B1) /
                 (1 + Net->A1 * (Input[i] + Net->D1 * Sum) + Net->C1);
    Net->F1->Output[i] = (Activation > 0);
  }
}


/******************************************************************************
                        A D J U S T I N G   W E I G H T S
 ******************************************************************************/


REAL Magnitude(NET* Net, BOOL* Input)
{
  INT  i;
  REAL Magnitude;

  Magnitude = 0;
  for (i=0; i<Net->F1->Units; i++) {
    Magnitude += Input[i];
  }
  return Magnitude;
}


void AdjustWeights(NET* Net)
{
  INT  i;
  REAL MagnitudeInput_;

  for (i=0; i<Net->F1->Units; i++) {
    if (Net->F1->Output[i]) {
      MagnitudeInput_ = Magnitude(Net, Net->F1->Output);
      Net->F1->Weight[i][Net->Winner] = 1;
      Net->F2->Weight[Net->Winner][i] = Net->L / (Net->L - 1 + MagnitudeInput_);
    }
    else {
      Net->F1->Weight[i][Net->Winner] = 0;
      Net->F2->Weight[Net->Winner][i] = 0;
    }
  }
}


/******************************************************************************
                      S I M U L A T I N G   T H E   N E T
```

```
 ******************************************************************************/


void SimulateNet(NET* Net, BOOL* Input, BOOL* Output)
{
  INT  i;
  BOOL Resonance, Exhausted;
  REAL MagnitudeInput, MagnitudeInput_;

  WriteInput(Net, Input);
  for (i=0; i<Net->F2->Units; i++) {
    Net->F2->Inhibited[i] = FALSE;
  }
  Resonance = FALSE;
  Exhausted = FALSE;
  do {
    SetInput(Net, Input);
    PropagateToF2(Net);
    GetOutput(Net, Output);
    if (Net->Winner != NO_WINNER) {
      PropagateToF1(Net, Input);
      MagnitudeInput = Magnitude(Net, Input);
      MagnitudeInput_ = Magnitude(Net, Net->F1->Output);
      if ((MagnitudeInput_ / MagnitudeInput) < Net->Rho)
        Net->F2->Inhibited[Net->Winner] = TRUE;
      else
        Resonance = TRUE;
    }
    else Exhausted = TRUE;
  } while (NOT (Resonance OR Exhausted));
  if (Resonance)
    AdjustWeights(Net);
  WriteOutput(Net, Output);
}


/******************************************************************************
                                    M A I N
 ******************************************************************************/


void main()
{
  NET Net;
  INT n;

  GenerateNetwork(&Net);
  InitializeApplication(&Net);

  for (n=0; n<NUM_DATA; n++) {
    SimulateNet(&Net, Input[n], Output[n]);
  }

  FinalizeApplication(&Net);
}
```

Annexe C – Self-Organizing Map

```
/******************************************************************************

                    ==================
        Network:    Self-Organizing Map
                    ==================

        Application:  Control
                      Pole Balancing Problem

        Author:       Karsten Kutza
        Date:         6.6.96

        Reference:    T. Kohonen
                      Self-Organized Formation
                      of Topologically Correct Feature Maps
                      Biological Cybernetics, 43, pp. 59-69, 1982

 ******************************************************************************/



/******************************************************************************
                            D E C L A R A T I O N S
 ******************************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <math.h>


typedef int           BOOL;
typedef int           INT;
typedef double        REAL;

#define FALSE         0
#define TRUE          1
#define NOT           !
#define AND           &&
#define OR            ||

#define MIN_REAL      -HUGE_VAL
#define MAX_REAL      +HUGE_VAL
#define MIN(x,y)      ((x)<(y) ? (x) : (y))
#define MAX(x,y)      ((x)>(y) ? (x) : (y))

#define PI            (2*asin(1))
#define sqr(x)        ((x)*(x))


typedef struct {                    /* A LAYER OF A NET:                     */
        INT           Units;        /* - number of units in this layer       */
        REAL*         Output;       /* - output of ith unit                  */
        REAL**        Weight;       /* - connection weights to ith unit      */
        REAL*         StepSize;     /* - size of search steps of ith unit    */
        REAL*         dScoreMean;   /* - mean score delta of ith unit        */
} LAYER;
```

```
typedef struct {                    /* A NET:                                */
        LAYER*        InputLayer;   /* - input layer                         */
        LAYER*        KohonenLayer; /* - Kohonen layer                       */
        LAYER*        OutputLayer;  /* - output layer                        */
        INT           Winner;       /* - last winner in Kohonen layer        */
        REAL          Alpha;        /* - learning rate for Kohonen layer     */
        REAL          Alpha_;       /* - learning rate for output layer      */
        REAL          Alpha__;      /* - learning rate for step sizes        */
        REAL          Gamma;        /* - smoothing factor for score deltas   */
        REAL          Sigma;        /* - parameter for width of neighborhood */
} NET;


/******************************************************************************
       R A N D O M S   D R A W N   F R O M   D I S T R I B U T I O N S
 ******************************************************************************/


void InitializeRandoms()
{
  srand(4715);
}


REAL RandomEqualREAL(REAL Low, REAL High)
{
  return ((REAL) rand() / RAND_MAX) * (High-Low) + Low;
}


REAL RandomNormalREAL(REAL Mu, REAL Sigma)
{
  REAL x,fx;

  do {
    x = RandomEqualREAL(Mu-3*Sigma, Mu+3*Sigma);
    fx = (1 / (sqrt(2*PI)*Sigma)) * exp(-sqr(x-Mu) / (2*sqr(Sigma)));
  } while (fx < RandomEqualREAL(0, 1));
  return x;
}


/******************************************************************************
              A P P L I C A T I O N - S P E C I F I C   C O D E
 ******************************************************************************/


#define ROWS          25
#define COLS          25

#define N             2
#define C             (ROWS * COLS)
#define M             1

#define TRAIN_STEPS   10000
#define BALANCED      100

FILE*                 f;


void InitializeApplication(NET* Net)
```

```
{
  INT i;

  for (i=0; i<Net->KohonenLayer->Units; i++) {
    Net->KohonenLayer->StepSize[i] = 1;
    Net->KohonenLayer->dScoreMean[i] = 0;
  }
  f = fopen("SOM.txt", "w");
}


void WriteNet(NET* Net)
{
  INT  r,c;
  REAL x,y,z;

  fprintf(f, "\n\n\n");
  for (r=0; r<ROWS; r++) {
    for (c=0; c<COLS; c++) {
      x = Net->KohonenLayer->Weight[r*ROWS+c][0];
      y = Net->KohonenLayer->Weight[r*ROWS+c][1];
      z = Net->OutputLayer->Weight[0][r*ROWS+c];
      fprintf(f, "([%5.1f°, %5.1f°], %5.1fN)    ", x, y, z);
    }
    fprintf(f, "\n");
  }
}


void FinalizeApplication(NET* Net)
{
  fclose(f);
}


/******************************************************************************
                    S I M U L A T I N G   T H E   P O L E
 ******************************************************************************/

                                    /* SIMULATION PARAMETERS FOR THE POLE:   */
#define L             1             /* - length of pole [m]                  */
#define Mc            1             /* - mass of cart [kg]                   */
#define Mp            1             /* - mass of pole [kg]                   */
#define G             9.81          /* - acceleration due to gravity [m/s²]  */
#define T             0.1           /* - time step [s]                       */
#define STEPS         10            /* - simulation steps in one time step   */


typedef struct {                    /* STATE VARIABLES OF A POLE:            */
        REAL          x;            /* - position of cart [m]                */
        REAL          xDot;         /* - velocity of cart [m/s]              */
        REAL          w;            /* - angle of pole [°]                   */
        REAL          wDot;         /* - angular velocity of pole [°/s]      */
        REAL          F;            /* - force applied to cart               */
} POLE;                             /*   during current time step [N]        */


void InitializePole(POLE* Pole)
{
  do {
    Pole->x    = 0;
```

```
    Pole->xDot = 0;
    Pole->w    = RandomEqualREAL(-30, 30);
    Pole->wDot = 0;
    Pole->F    = 0;
  } while (Pole->w == 0);
}


void SimulatePole(POLE* Pole)
{
  INT  s;
  REAL x, xDot, xDotDot;
  REAL w, wDot, wDotDot;
  REAL F;

  x    = Pole->x;
  xDot = Pole->xDot;
  w    = (Pole->w    / 180) * PI;
  wDot = (Pole->wDot / 180) * PI;
  F    = Pole->F;
  for (s=0; s<STEPS; s++) {

    wDotDot = (G*sin(w) + cos(w) * ((-F - Mp*L*sqr(wDot)*sin(w)) / (Mc+Mp))) /
              (L * ((REAL) 4/3 - (Mp*sqr(cos(w))) / (Mc+Mp)));

    xDotDot = (F + Mp*L * (sqr(wDot)*sin(w) - wDotDot*cos(w))) / (Mc+Mp);

    x    += (T / STEPS) * xDot;
    xDot += (T / STEPS) * xDotDot;
    w    += (T / STEPS) * wDot;
    wDot += (T / STEPS) * wDotDot;
  }
  Pole->x    = x;
  Pole->xDot = xDot;
  Pole->w    = (w    / PI) * 180;
  Pole->wDot = (wDot / PI) * 180;
}


BOOL PoleStillBalanced(POLE* Pole)
{
  return (Pole->w >= -60) AND (Pole->w <= 60);
}


REAL ScoreOfPole(POLE* Pole)
{
  return -sqr(Pole->w);
}


/*****************************************************************************
                          I N I T I A L I Z A T I O N
 *****************************************************************************/


void GenerateNetwork(NET* Net)
{
  INT i;

  Net->InputLayer   = (LAYER*) malloc(sizeof(LAYER));
```

```
  Net->KohonenLayer = (LAYER*) malloc(sizeof(LAYER));
  Net->OutputLayer  = (LAYER*) malloc(sizeof(LAYER));

  Net->InputLayer->Units        = N;
  Net->InputLayer->Output       = (REAL*)  calloc(N, sizeof(REAL));

  Net->KohonenLayer->Units      = C;
  Net->KohonenLayer->Output     = (REAL*)  calloc(C, sizeof(REAL));
  Net->KohonenLayer->Weight     = (REAL**) calloc(C, sizeof(REAL*));
  Net->KohonenLayer->StepSize   = (REAL*)  calloc(C, sizeof(REAL));
  Net->KohonenLayer->dScoreMean = (REAL*)  calloc(C, sizeof(REAL));

  Net->OutputLayer->Units       = M;
  Net->OutputLayer->Output      = (REAL*)  calloc(M, sizeof(REAL));
  Net->OutputLayer->Weight      = (REAL**) calloc(M, sizeof(REAL*));

  for (i=0; i<C; i++) {
    Net->KohonenLayer->Weight[i] = (REAL*) calloc(N, sizeof(REAL));
  }
  for (i=0; i<M; i++) {
    Net->OutputLayer->Weight[i] = (REAL*) calloc(C, sizeof(REAL));
  }
}


void RandomWeights(NET* Net)
{
  INT i,j;

  for (i=0; i<Net->KohonenLayer->Units; i++) {
    for (j=0; j<Net->InputLayer->Units; j++) {
      Net->KohonenLayer->Weight[i][j] = RandomEqualREAL(-30, 30);
    }
  }
  for (i=0; i<Net->OutputLayer->Units; i++) {
    for (j=0; j<Net->KohonenLayer->Units; j++) {
      Net->OutputLayer->Weight[i][j] = 0;
    }
  }
}


void SetInput(NET* Net, REAL* Input)
{
  INT i;

  for (i=0; i<Net->InputLayer->Units; i++) {
    Net->InputLayer->Output[i] = Input[i];
  }
}


void GetOutput(NET* Net, REAL* Output)
{
  INT i;

  for (i=0; i<Net->OutputLayer->Units; i++) {
    Output[i] = Net->OutputLayer->Output[i];
  }
}
```

```
/******************************************************************************
                    P R O P A G A T I N G   S I G N A L S
 ******************************************************************************/


void PropagateToKohonen(NET* Net)
{
  INT  i,j;
  REAL Out, Weight, Sum, MinOut;

  for (i=0; i<Net->KohonenLayer->Units; i++) {
    Sum = 0;
    for (j=0; j<Net->InputLayer->Units; j++) {
      Out = Net->InputLayer->Output[j];
      Weight = Net->KohonenLayer->Weight[i][j];
      Sum += sqr(Out - Weight);
    }
    Net->KohonenLayer->Output[i] = sqrt(Sum);
  }
  MinOut = MAX_REAL;
  for (i=0; i<Net->KohonenLayer->Units; i++) {
    if (Net->KohonenLayer->Output[i] < MinOut)
      MinOut = Net->KohonenLayer->Output[Net->Winner = i];
  }
}


void PropagateToOutput(NET* Net)
{
  INT i;

  for (i=0; i<Net->OutputLayer->Units; i++) {
    Net->OutputLayer->Output[i] = Net->OutputLayer->Weight[i][Net->Winner];
  }
}


void PropagateNet(NET* Net)
{
  PropagateToKohonen(Net);
  PropagateToOutput(Net);
}


/******************************************************************************
                      T R A I N I N G   T H E   N E T
 ******************************************************************************/


REAL Neighborhood(NET* Net, INT i)
{
  INT  iRow, iCol, jRow, jCol;
  REAL Distance;

  iRow = i / COLS; jRow = Net->Winner / COLS;
  iCol = i % COLS; jCol = Net->Winner % COLS;

  Distance = sqrt(sqr(iRow-jRow) + sqr(iCol-jCol));

  return exp(-sqr(Distance) / (2*sqr(Net->Sigma)));
```

```
}


void TrainKohonen(NET* Net, REAL* Input)
{
  INT  i,j;
  REAL Out, Weight, Lambda, StepSize;

  for (i=0; i<Net->KohonenLayer->Units; i++) {
    for (j=0; j<Net->InputLayer->Units; j++) {
      Out = Input[j];
      Weight = Net->KohonenLayer->Weight[i][j];
      Lambda = Neighborhood(Net, i);
      Net->KohonenLayer->Weight[i][j] += Net->Alpha * Lambda * (Out - Weight);
    }
    StepSize = Net->KohonenLayer->StepSize[i];
    Net->KohonenLayer->StepSize[i] += Net->Alpha__ * Lambda * -StepSize;
  }
}


void TrainOutput(NET* Net, REAL* Output)
{
  INT  i,j;
  REAL Out, Weight, Lambda;

  for (i=0; i<Net->OutputLayer->Units; i++) {
    for (j=0; j<Net->KohonenLayer->Units; j++) {
      Out = Output[i];
      Weight = Net->OutputLayer->Weight[i][j];
      Lambda = Neighborhood(Net, j);
      Net->OutputLayer->Weight[i][j] += Net->Alpha_ * Lambda * (Out - Weight);
    }
  }
}


void TrainUnits(NET* Net, REAL* Input, REAL* Output)
{
  TrainKohonen(Net, Input);
  TrainOutput(Net, Output);
}


void TrainNet(NET* Net)
{
  INT  n,t;
  POLE Pole;
  REAL wOld, wNew, ScoreOld, ScoreNew, dScore, dScoreMean, StepSize;
  REAL Input[N];
  REAL Output[M];
  REAL Target[M];

  n = 0;
  while (n<TRAIN_STEPS) {
    t = 0;
    InitializePole(&Pole);
    fprintf(f, " Time     Angle     Force\n");
    fprintf(f, "%4.1fs    %5.1f°    %5.1fN\n", t * T, Pole.w, Pole.F);
    wOld = Pole.w;
    ScoreOld = ScoreOfPole(&Pole);
```

```
    SimulatePole(&Pole);
    wNew = Pole.w;
    ScoreNew = ScoreOfPole(&Pole);
    while (PoleStillBalanced(&Pole) AND (t<BALANCED)) {
      n++;
      t++;
      Net->Alpha   = 0.5 * pow(0.01, (REAL) n / TRAIN_STEPS);
      Net->Alpha_  = 0.5 * pow(0.01, (REAL) n / TRAIN_STEPS);
      Net->Alpha__ = 0.005;
      Net->Gamma   = 0.05;
      Net->Sigma   = 6.0 * pow(0.2, (REAL) n / TRAIN_STEPS);
      Input[0] = wOld;
      Input[1] = wNew;
      SetInput(Net, Input);
      PropagateNet(Net);
      GetOutput(Net, Output);
      Pole.F = Output[0];
      StepSize = Net->KohonenLayer->StepSize[Net->Winner];
      Pole.F += StepSize * RandomNormalREAL(0, 10);
      fprintf(f, "%4.1fs    %5.1f°    %5.1fN\n", t * T, Pole.w, Pole.F);
      wOld = Pole.w;
      ScoreOld = ScoreOfPole(&Pole);
      SimulatePole(&Pole);
      wNew = Pole.w;
      ScoreNew = ScoreOfPole(&Pole);
      dScore = ScoreNew - ScoreOld;
      dScoreMean = Net->KohonenLayer->dScoreMean[Net->Winner];
      if (dScore > dScoreMean) {
        Target[0] = Pole.F;
        TrainUnits(Net, Input, Target);
      }
      Net->KohonenLayer->dScoreMean[Net->Winner] += Net->Gamma * (dScore -
dScoreMean);
    }
    if (PoleStillBalanced(&Pole))
      fprintf(f, "Pole still balanced after %0.1fs ...\n\n", t * T);
    else
      fprintf(f, "Pole fallen after %0.1fs ...\n\n", (t+1) * T);
  }
}


/******************************************************************************
                                    M A I N
 ******************************************************************************/


void main()
{
  NET Net;

  InitializeRandoms();
  GenerateNetwork(&Net);
  RandomWeights(&Net);
  InitializeApplication(&Net);
  TrainNet(&Net);
  WriteNet(&Net);
  FinalizeApplication(&Net);
}
```

Annexe D – Motifs d'entraînements

Test A 1

```
5 3 1
0 0 0
0
0.25 0 0
0.25
0.5 0 0
0.5
0.75 0 0
0.75
1 0 0
1
```

Test A 2

```
11 3 1
0 0 0
0
0.250000 0 0
0.250000
0.500000 0 0
0.500000
0.750000 0 0
0.750000
1 0 0
1
1 0.250000 0
0.500000
1 0.250000 0.250000
0.250000
1 0.500000 0
0.250000
1 0.500000 0.500000
0.125000
1 0.750000 0.500000
0.062500
1 0.750000 0.750000
0.031250
```

Test A 3

```
16 3 1
0 0 0
0
0.250000 0 0
0.250000
0.500000 0 0
0.500000
0.750000 0 0
0.750000
1 0 0
1
1 0.250000 0
0.500000
1 0.250000 0.250000
0.250000
```

```
1 0.500000 0
0.250000
1 0.500000 0.500000
0.125000
1 0.750000 0.500000
0.062500
1 0.750000 0.750000
0.031250
1 1 0.750000
0.015625
1 1 1
0.007531
0.750000 1 1
0
0.500000 1 1
0
0.250000 1 1
0
```

Test A 4

```
18 3 1
0 0 0
0
0.250000 0 0
0.250000
0.500000 0 0
0.500000
0.750000 0 0
0.750000
1 0 0
1
1 0.250000 0
0.500000
1 0.250000 0.250000
0.250000
1 0.500000 0
0.250000
1 0.500000 0.500000
0.125000
1 0.750000 0.500000
0.062500
1 0.750000 0.750000
0.031250
1 1 0.750000
0.015625
1 1 1
0.007531
0.750000 1 1
0
0.500000 1 1
0
0.250000 1 1
0
0 1 1
0
1 0 0.250000
0.500000
```

Test A Full

```
9 3 1
0.500000 0.500000 0.500000
0.062500
0.250000 0.250000 0.250000
0.062500
0 0 0
0
0 0 0
0
0.750000 0.125000 0
0.600000
0.800000 0.125000 0
0.700000
1 0.125000 0
0.850000
1 0 0
1
1 0.125000 0.125000
0.500000
```

Test B

```
14 4 1
0.800000 0.581472 0.251015 0.641244
0.742893
0.742893 0.800000 0.311929 0.453553
0.754695
0.754695 0.742893 0.581472 0.374746
0.559010
0.559010 0.754695 0.800000 0.268147
0.407868
0.407868 0.559010 0.742893 0.251015
0.313832
0.313832 0.407868 0.754695 0.311929
0.266624
0.266624 0.313832 0.559010 0.581472
0.232741
0.232741 0.266624 0.407868 0.800000
0.281853
0.281853 0.232741 0.313832 0.742893
0.444797
0.444797 0.281853 0.266624 0.754695
0.555203
0.555203 0.444797 0.232741 0.559010
0.655330
0.655330 0.555203 0.281853 0.407868
0.622589
0.622589 0.655330 0.444797 0.313832
0.595939
0.595939 0.622589 0.555203 0.266624
0.443274
```

Test C 1

```
8 4 1
1 0 0 0
1
0 1 0 0
1
```

```
0 0 1 0
-1
0 0 0 1
-1
1 0 1 0
0
1 0 0 1
0
0 1 1 0
0
0 1 0 1
0
```

Test C 2

```
7 4 1
1 0 0 0
0.001000
0 1 0 0
0.010000
0 0 1 0
0.100000
0 0 0 1
1
1 0 0 1
1.001000
0 2 0 0
0.020000
2 0 3 1
1.302000
```

Test C 3

```
13 4 1
1 0 0 0
0.001000
0 1 0 0
0.010000
0 0 1 0
0.100000
0 0 0 1
1
1 0 0 1
1.001000
0 2 0 0
0.020000
2 0 3 1
1.302000
1 1 1 1
1.111000
1 0 1 0
0.101000
1 1 0 0
0.011000
1 2 0 0
0.021000
1 0 2 0
0.301000
8 0 1 0
0.108000
```

Test C 4

```
16 4 1
1 0 0 0
0.001000
0 1 0 0
0.010000
0 0 1 0
0.100000
0 0 0 1
1
1 0 0 1
1.001000
0 2 0 0
0.020000
2 0 3 1
1.302000
1 1 1 1
1.111000
1 0 1 0
0.101000
1 1 0 0
0.011000
1 2 0 0
0.021000
1 0 2 0
0.301000
8 0 1 0
0.108000
7 0 1 0
0.107000
3 0 1 0
0.103000
1 0 1 0
0.101000
```

Test E

```
4 4 1
0 0 1 1
1.410000
0 0 2 2
2
2 3 4 7
4
-3 2 4 2
7
```

Annexe E – Réseaux

Test A

```
FANN_FLO_2.1
num_layers=3
learning_rate=0.700000
connection_rate=1.000000
network_type=0
learning_momentum=0.000000
training_algorithm=0
train_error_function=1
train_stop_function=0
cascade_output_change_fraction=0.010000
quickprop_decay=-0.000100
quickprop_mu=1.750000
rprop_increase_factor=1.200000
rprop_decrease_factor=0.500000
rprop_delta_min=0.000000
rprop_delta_max=50.000000
rprop_delta_zero=0.500000
cascade_output_stagnation_epochs=12
cascade_candidate_change_fraction=0.010000
cascade_candidate_stagnation_epochs=12
cascade_max_out_epochs=150
cascade_max_cand_epochs=150
cascade_num_candidate_groups=2
bit_fail_limit=3.4999999999999980000e-001
cascade_candidate_limit=1.0000000000000000000e+003
cascade_weight_multiplier=4.0000000000000020000e-001
cascade_activation_functions_count=8
cascade_activation_functions=3 5 7 8 10 11 14 15
cascade_activation_steepnesses_count=4
cascade_activation_steepnesses=2.5000000000000000000e-001
5.0000000000000000000e-001 7.5000000000000000000e-001
1.0000000000000000000e+000
layer_sizes=4 5 2
scale_included=0
neurons (num_inputs, activation_function, activation_steepness)=(0, 0,
0.0000000000000000000e+000) (0, 0, 0.0000000000000000000e+000) (0, 0,
0.0000000000000000000e+000) (0, 0, 0.0000000000000000000e+000) (4, 4,
5.0000000000000000000e-001) (4, 4, 5.0000000000000000000e-001) (4, 4,
5.0000000000000000000e-001) (4, 4, 5.0000000000000000000e-001) (0, 0,
0.0000000000000000000e+000) (5, 4, 5.0000000000000000000e-001) (0, 0,
0.0000000000000000000e+000)
connections (connected_to_neuron, weight)=(0, -1.8848471853787057000e+000) (1,
-8.8531495459847065000e-002) (2, 1.6528320558791165000e-002) (3,
2.3225646906409509000e+000) (0, -1.1297093263718105000e+000) (1,
7.9632569545992737000e-002) (2, 4.2736817043078190000e-002) (3,
-7.1094481465872994000e-001) (0, 2.6670584421691229000e-001) (1,
-2.0928955389990733000e-002) (2, -5.5609131688015623000e-002) (3,
-2.6213193051535993000e-001) (0, 1.0870680304474463000e+000) (1,
6.6198731455187954000e-002) (2, 2.0745849918512249000e-002) (3,
7.1562961035732631000e-001) (4, -1.0857489325014487000e+001) (5,
-9.0846635135628944000e+000) (6, 1.4841035291319367000e+000) (7,
7.8730737452583783000e+000) (8, 3.8063009174013032000e+000)
```

Test C

```
FANN_FLO_2.1
num_layers=4
learning_rate=0.700000
connection_rate=1.000000
network_type=0
learning_momentum=0.000000
training_algorithm=0
train_error_function=0
train_stop_function=0
cascade_output_change_fraction=0.010000
quickprop_decay=-0.000100
quickprop_mu=1.750000
rprop_increase_factor=1.200000
rprop_decrease_factor=0.500000
rprop_delta_min=0.000000
rprop_delta_max=50.000000
rprop_delta_zero=0.500000
cascade_output_stagnation_epochs=12
cascade_candidate_change_fraction=0.010000
cascade_candidate_stagnation_epochs=12
cascade_max_out_epochs=150
cascade_max_cand_epochs=150
cascade_num_candidate_groups=2
bit_fail_limit=3.4999999999999980000e-001
cascade_candidate_limit=1.0000000000000000000e+003
cascade_weight_multiplier=4.0000000000000020000e-001
cascade_activation_functions_count=8
cascade_activation_functions=3 5 7 8 10 11 14 15
cascade_activation_steepnesses_count=4
cascade_activation_steepnesses=2.5000000000000000000e-001
5.0000000000000000000e-001 7.5000000000000000000e-001
1.0000000000000000000e+000
layer_sizes=5 5 5 2
scale_included=0
neurons (num_inputs, activation_function, activation_steepness)=(0, 0,
0.0000000000000000000e+000) (0, 0, 0.0000000000000000000e+000) (0, 0,
0.0000000000000000000e+000) (0, 0, 0.0000000000000000000e+000) (0, 0,
0.0000000000000000000e+000) (5, 4, 5.0000000000000000000e-001) (5, 4,
5.0000000000000000000e-001) (5, 4, 5.0000000000000000000e-001) (5, 4,
5.0000000000000000000e-001) (0, 0, 0.0000000000000000000e+000) (5, 4,
5.0000000000000000000e-001) (5, 4, 5.0000000000000000000e-001) (5, 4,
5.0000000000000000000e-001) (5, 4, 5.0000000000000000000e-001) (0, 0,
0.0000000000000000000e+000) (5, 4, 5.0000000000000000000e-001) (0, 0,
0.0000000000000000000e+000)
connections (connected_to_neuron, weight)=(0, 2.3895264027942176000e-002) (1,
-2.1850586263099103000e-002) (2, -8.8415528661244025000e-002) (3,
-9.9670411641454848000e-002) (4, -6.5972901373697823000e-002) (0,
3.1860352037256234000e-002) (1, 4.5434570989527856000e-002) (2,
7.2070313573931344000e-002) (3, -7.5646973783477733000e-002) (4,
6.8103028358564188000e-002) (0, -5.6457520372532599000e-002) (1,
7.1673585052394628000e-002) (2, -9.9578858905715606000e-002) (3,
4.5343018253788614000e-002) (4, 5.2368164842846454000e-003) (0,
-2.2979736670549755000e-002) (1, -9.9035646006996103000e-002) (2,
-6.9708252991858899000e-002) (3, -1.8341064726428158000e-002) (4,
8.0889893783478328000e-002) (5, -9.0936280651931156000e-002) (6,
8.9440919301523536000e-002) (7, -7.3834229615840741000e-002) (8,
6.4733887683360081000e-002) (9, 5.5865479348085501000e-002) (5,
-9.7137452619335818000e-002) (6, 7.7935792176958785000e-002) (7,
-2.7203369545986789000e-002) (8, 2.3175049173460138000e-002) (9,
```

-1.84997561350428440000e-002) (5, -6.49292001862704640000e-002) (6,
1.79260256577435940000e-002) (7, -4.89257819790509530000e-002) (8,
6.92810068917424360000e-002) (9, 7.33459483585647830000e-002) (5,
-4.97680671478519800000e-002) (6, 5.68908699883650120000e-002) (7,
1.11572267287556310000e-002) (8, -3.30566411175823300000e-002) (9,
1.99218752968590710000e-002) (10, 7.17712413038498200000e-002) (11,
6.85180674272487520000e-002) (12, -3.26110844703180190000e-002) (13,
6.54846201164218660000e-002) (14, -9.89685073341206590000e-002)