UNIVERSITE PAUL SABATIER
118, route de Narbonne
31400 Toulouse

Diplôme d'Etudes Approfondies
"représentation des connaissances
et formalisation du raisonnement"
année 1995-1996

"Un moteur de type réseau de neurones pour la révision des connaissances dans un système tuteur intelligent"

Laboratoire LGC / Equipe API
Université Paul Sabatier
50, chemin des Maraîchers
31077 Toulouse cedex

Sébastien MARCEL

Directeur de recherche : M. Guy Gouardères

Mots clés Enseignement Intelligemment Assisté par Ordinateur (EIAO)
Systèmes Tuteurs Intelligents (STI)
Réseaux de neurones (RN), Connexionnisme
Intelligence Artificielle (IA)

Résumé

La révision des connaissances dans les systèmes tuteurs intelligents (STI) est un domaine qui soulève de nombreux problèmes. Quelles connaissances vont et doivent être révisées et surtout comment ? A ce sujet, nous étudions une solution connexionniste : l'intégration d'un moteur de type réseau de neurones à un STI.

Il est à noter que nous nous plaçons dans le cadre d'un système tuteur intelligent dédié à l'apprentissage du calcul mental par l'exercice.

Remerciements

Je tiens à remercier mon directeur de recherche, Monsieur Guy Gouardères, pour m'avoir accueilli dans son laboratoire, et pour ses conseils qui m'ont guidé dans la réalisation de ce stage.

Je remercie également tous les membres de l'équipe LGC, pour leur accueil et pour leurs remarques constructives données tout au long de cette année. Merci à Christophe, Frédéric, Maodo, Marie-Françoise, Murielle et Luc.

SOMMAIRE

[1]**Ci-joint :** deux feuilles "volantes" récapitulant toutes les variables et toutes les formules utilisées dans ce rapport.

I - Introduction

Une des voies de recherches actuelles sur les systèmes de tuteurs intelligents (STI) portent sur l'intégration d'un "modèle" de l'élève. L'objectif est de simuler un élève virtuel pour essayer de prévoir le comportement de l'élève réel afin d'adapter la stratégie d'apprentissage du tuteur à celle de résolution de l'élève.

Nous nous plaçons dans le cadre d'architectures multi-agents pour les STI. Dans ces systèmes, les stratégies sont multiples et il devient difficile de mesurer l'écart entre celles de l'élève virtuel[2], celles de l'élève réel et celles de l'expert.

Une nouvelle approche, la modélisation qualitative [Blanc 96] étudie cette possibilité de changer les stratégies en combinant des techniques de planification dynamique et des techniques inspirées de la programmation sous contraintes.

Les agents des STI sont fondés sur un modèle à trois niveaux (figure I.1).

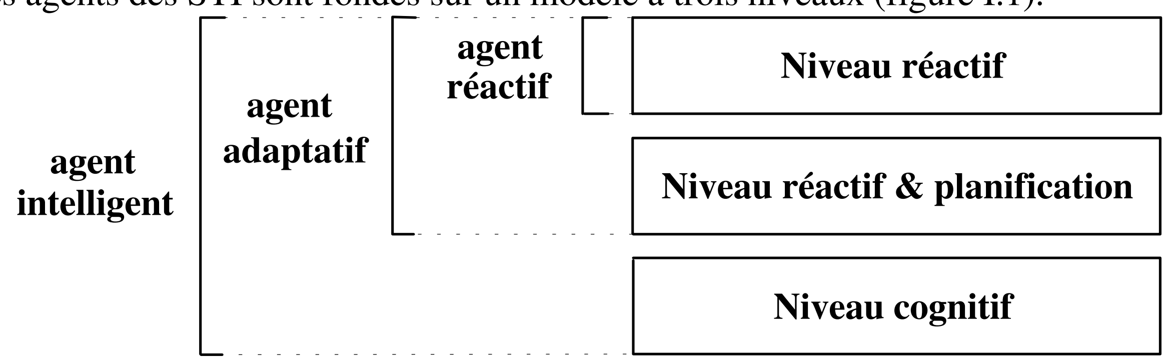


figure I.1

Le niveau cognitif caractérise les agents intelligents, comme ceux introduits par le modèle conceptuel des **actors** [Frasson 1995]. Dans ce modèle, les agents ne sont plus simplement réactifs (conditionnés par des réflexes) ou adaptatifs (recherches heuristiques), mais ils atteignent un niveau supérieur "d'intelligence", le niveau cognitif, inspiré des processus cognitifs humains (états et schémas mentaux).

En effet, les agents ne peuvent pas rester seulement réactifs ou adaptatifs. L'apprentissage de la "décision" doit et ne peut se faire qu'au niveau cognitif par l'acquisition de connaissances de base sur l'expertise du domaine.

[2]modèle, clône informatique de l'élève simulant ses préférences et son comportement.

Le cycle d'apprentissage (figure I.2) contribue à accroître ces connaissances. Ceci se traduit par la mémorisation d'exemples de base lors d'un apprentissage supervisé.

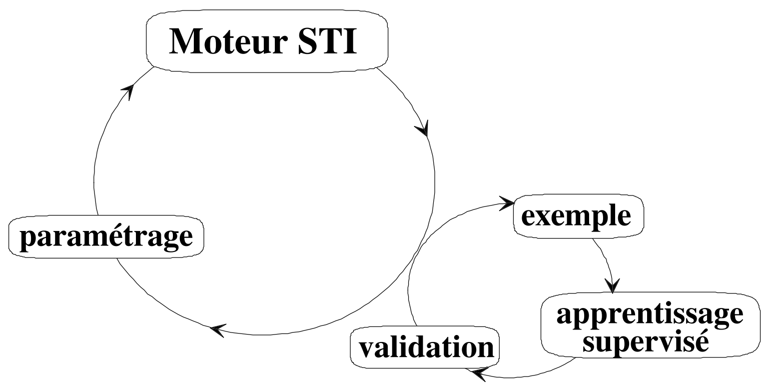


figure I.2 : **Cycle d'apprentissage**

Dans ce cycle, le moteur STI, c'est-à-dire le coeur du système tuteur, subit un apprentissage de chaque exemple. Lorsque cet apprentissage est validé, on passe à celui d'un nouvel exemple et éventuellement à un ajustement des paramètres du STI.

Actuellement dans les STI, on a des difficultés à modéliser le niveau cognitif. Les recherches s'orientent vers de nombreuses voies pour trouver des modèles convenables.

Parmi celles-ci on retrouve [Honavar 1994]:

- le calcul symbolique [Buchanan & Wilkins 1993],
- les réseaux de neurones [Carpenter & Grossberg 1991],
- la *statistical pattern recognition* [Fukunaga 1990],
- les algorithmes génétiques [Koza 92].

Les techniques traditionnelles d'Intelligence Artificielle permettent de modéliser la majeure partie des problèmes. Mais, lorsque ces derniers deviennent complexes la modélisation devient longue et coûteuse. Il peut donc être bénéfique de s'intéresser à une nouvelle technique.

Nous avons choisi l'approche connexionniste pour ses propriétés d'adaptation et d'apprentissage. En effet, un réseau de neurones est capable de modifier son comportement d'après les résultats de son expérience passée.

Organisation du rapport :

Après une présentation des systèmes tuteurs intelligents qui nous permettra de nous familiariser avec l'EIAO et son vocabulaire, nous aborderons le domaine des réseaux de neurones, où nous introduirons leurs fondements, leurs applications et leurs propriétés d'apprentissage.

Puis, nous exposerons l'architecture générale d'un STI pour l'entraînement aux techniques du calcul mental et nous justifierons l'intégration d'un réseau de neurones à cette architecture. Ceci nous donnera un aperçu du rôle du réseau dans la mesure de l'écart entre les solutions de l'élève et de l'expert.

Par la suite, nous présenterons le modèle de réseau de neurones choisi qui, selon nous, s'adapte le mieux à notre problématique. Nous le formaliserons, ainsi que les processus de traitement de l'information et les mécanismes d'apprentissage.

Enfin, nous conclurons sur les points forts et les limites mis en évidence par les tests, ainsi que sur les perspectives de recherche envisageables pour combler ces mêmes limites.

Présentation des outils de travail :

Le rapport est rédigé avec Winword 2.0 sous Windows 3.1 et sous Windows 95. La majeure partie des pré-rapports qui le composent, ont été écrits avec Amipro 3.0 ainsi la plus part des figures, converties, puis retravaillées sous MSDraw, à l'exception de celles en annexes tracées sous Excel 4.0. D'autre part, les équations sont définies à l'aide de l'éditeur d'équations de Winword 2.0.

Les implémentations sont effectuées en langage C++, et tous les programmes sources ont été compilés :

- sur compatibles PC à l'aide de Borland C++ 3.1,
- sur station HP 712/60, système HP/Ux 10.1 à l'aide du compilateur HP C++,
- sur station Sun/Sparc, système SOLARIS/Unix à l'aide du compilateur gnu C++.

Cependant, la totalité des tests sont réalisés sur la station HP.

II - Etat de l'art

1. EIAO : Enseignement Intelligemment Assisté par Ordinateur

1.1. Introduction

L'enseignement assisté par ordinateur est apparu peu de temps après l'informatique. La technologie est venu compléter l'enseignement et lui a apporté un nouvel outil, l'ordinateur. L'utilisation de l'ordinateur à des fins pédagogiques constitue le champ de l'Enseignement Assisté par Ordinateur (EAO) [Lefe 84] [Canut 90].

1.2. De l'EAO vers l'EIAO

L'EIAO hérite des concepts de l'EAO mais aussi de ceux de l'IA, des sciences de l'éducation et, plus généralement, des sciences cognitives (figure II.1.1).

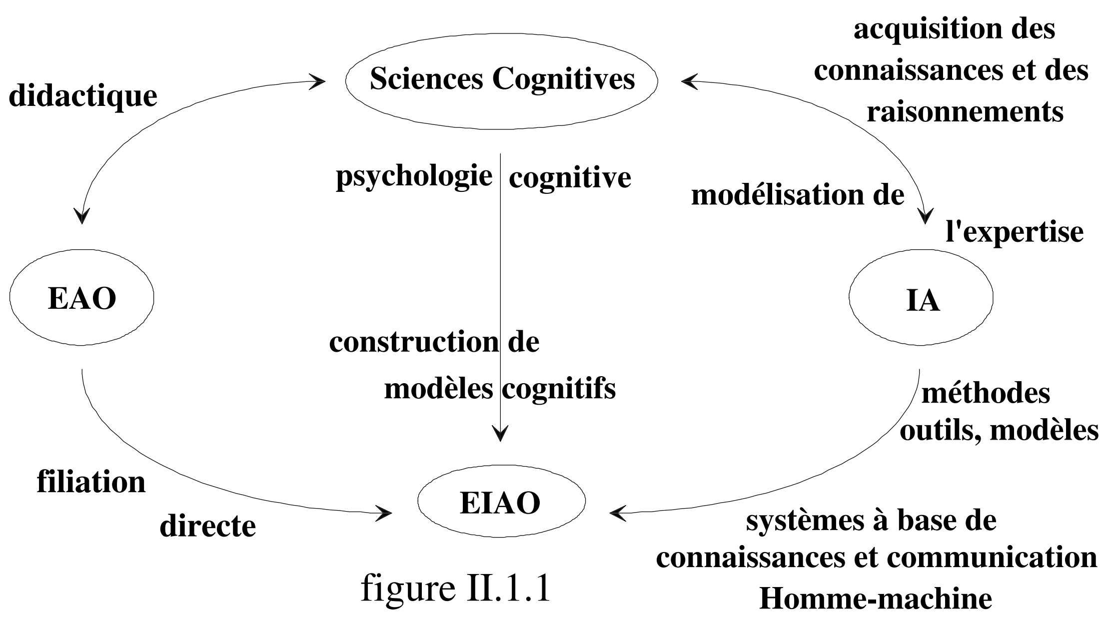


figure II.1.1

L'EIAO s'inscrit dans la filiation de l'EAO, il bénéficie des recherches menées en sciences cognitives et de celles relatives à la didactique. Il utilise des méthodes et des techniques d'IA pour la manipulation et la représentation des connaissances ainsi que pour la communication Homme-Machine. Il s'agit de faire bénéficier l'EAO des nouvelles techniques informatiques, tout en brisant le rôle central et dirigiste de la machine ainsi que la rigidité de ses échanges avec l'apprenant [Haton 92].

1.3. Architecture et composantes d'un STI classique

Au domaine de l'EAO, l'IA peut apporter [Nicaud 88]:

- la maîtrise du domaine par l'accès à des connaissances manipulables et par la capacité à résoudre des problèmes,
- l'individualisation de l'enseignement par une prise en compte du profil de l'élève et la compréhension de ses démarches,
- la flexibilité de l'interaction élève-machine au niveau du contenu et de la forme du discours à l'aide d'interfaces conviviales.

Traditionnellement, un STI comprend 4 composantes : l'expert, le tuteur, l'interface et le modèle de l'élève (figure II.1.2).

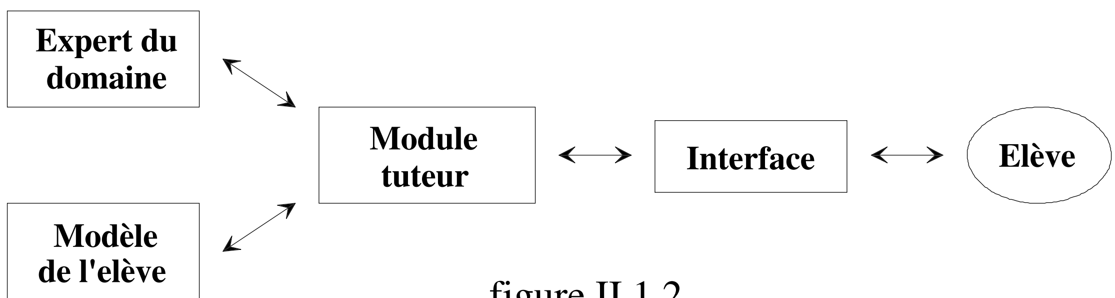


figure II.1.2

L'expert possède la maîtrise du domaine à enseigner, le tuteur surveille et dirige l'acquisition des connaissances et des raisonnements, le modèle de l'élève permet d'intégrer le comportement de ce dernier dans le STI et l'interface assure le dialogue entre l'élève et le tuteur.

1.3.1. L'expert du domaine

L'expert possède les connaissances expertes sur un domaine particulier, son travail varie généralement en fonction du domaine et du type des problèmes à enseigner. Ses connaissances servent de référence lors de l'évaluation de celles de l'élève par le module tuteur.

Des règles de production peuvent être utilisées pour tracer ou donner un raisonnement, l'ajout d'heuristiques permet d'expliquer ces raisonnements.

1.3.2. Le modèle de l'élève

Le modèle de l'élève effectue la prédiction des comportements de ce dernier et le diagnostic des causes de ses erreurs.
Le modèle doit comprendre [Nicaud 88] :

- les processus cognitifs : connaissances factuelles et opératoires,
- les stratégies méta-cognitives : capacités de raisonnement,
- les attributs psychologiques : niveau de développement, intérêt et style d'apprentissage.

Généralement, on essaie d'établir un profil de l'élève (essais, échecs) ou de modéliser ses connaissances (faits faux, règles déviées, mauvaises heuristiques).

1.3.3. Le module tuteur

Le module tuteur a pour rôle de placer l'élève dans une situation d'apprentissage optimale par rapport à l'objectif fixé et à ses connaissances. Il faut donc expliciter cette connaissance pour la rendre disponible et évaluable par le système d'EIAO envisagé. De plus, le tuteur doit intégrer l'écart entre les solutions réelles, les solutions expertes et les solutions prédites à partir du modèle de l'élève.

Dans un système **coach**, par exemple [Mengelle 95], lorsque le tuteur intervient il félicite, conseille ou réprimande l'élève en affichant un message d'information dans un écran réservé à cet usage.
Cependant, la difficulté réside toujours dans le fait de déterminer le degré de rigidité du guidage, des sollicitations (interventions) et de l'orientation (changement de stratégie pédagogique) en fonction de l'écart, tout en préservant la souplesse et la flexibilité du module tuteur.

1.3.4. L'interface avec l'élève

La qualité de l'interface est fondamentale pour faciliter les échanges avec l'élève. Aussi, généralement on utilise le multi-fenêtrage et la souris pour que la communication avec ce dernier soit la plus rapide, la plus conviviale, la plus adaptée et surtout la moins contraignante possible.

1.4. Conclusion

Les STI sont destinés à compléter l'enseignement traditionnel et non pas à le remplacer. Cependant, tous les problèmes ne sont pas résolus. Peut-on réellement faire "entrer" la pédagogie et les experts dans la machine ? Et surtout, peut-on réellement intégrer un modèle de l'élève ?

En effet, ces systèmes ont des difficultés à mesurer en temps réel l'écart entre solutions multi-sources (expert, réel, virtuel), ainsi qu'à décider comment et quand intégrer le comportement de l'élève et quelles actions entreprendre pour poursuivre l'apprentissage.

Nous pensons que les capacités d'apprentissage et de traitement de l'information en temps réel des réseaux de neurones devraient permettre de résoudre les problèmes des STI. C'est ce point que nous développerons par la suite après avoir présenté l'état de l'art sur les réseaux de neurones.

2. Les réseaux de neurones

Les neurosciences et les sciences cognitives nous proposent de nombreux modèles, notamment pour représenter les systèmes perceptifs et les systèmes réflexes. Mais, il reste encore beaucoup de travail de recherche en neurosciences et en sciences cognitives pour pouvoir modéliser les systèmes cognitifs, et enfin passer du "réflexe à la réflexion" [Bourret 91] [Davalo 89] [Renders 95].

Si un tel intérêt se porte sur les réseaux de neurones c'est grâce à leurs propriétés : parallélisme, capacité de généralisation, capacité d'adaptation et mémoire distribuée.

2.1. Introduction

Depuis l'apparition de l'informatique, les chercheurs ont tenté de reproduire un mécanisme qui est le propre de l'homme : le raisonnement et au sens le plus large du terme, la pensée [Marcel 95].

Ainsi, est née l'Intelligence Artificielle, dont les tendances de recherche se sont rapidement orientées vers l'approche symbolique, c'est-à-dire vers une représentation des connaissances et une formalisation du raisonnement sous forme de règles, inspirée de la logique mathématique.

Pourtant, dès 1943 W. McCulloch et Pitts introduisent les réseaux de neurones et donnent ainsi naissance au connexionnisme [McCulloch 43]. Puis en 1957, Rosenblatt présente le premier modèle opérationnel de réseaux de neurones : le perceptron inspiré du système visuel et capable d'apprendre à calculer certaines fonctions logiques en modifiant ses connexions synaptiques [Rosenblatt 57]. Déjà, en 1949 Hebb parle de cette propriété de plasticité synaptique en proposant une explication à l'apprentissage [Hebb 49].

Cependant, dans les années 60, les réseaux de neurones tombent dans l'oubli par un désintéressement des chercheurs et des investisseurs. Ce n'est que depuis quelques années que l'on assiste au renouveau des réseaux de neurones grâce à des chercheurs comme Hopfield qui en 1982 a montré l'analogie des réseaux de neurones avec certains systèmes physiques et a permis de leur appliquer un formalisme mieux maîtrisé [Hopfield 82].

Depuis, les recherches et les applications ont encore progressé :

1943	Mc Culloch & Pitts	Définition du neurone formel
1949	Hebb	Règles d'apprentissage pour les neurones
1957	Rosenblatt	Modèle de réseau de neurones : le perceptron
1969	Minsky & Papert	Optimisations sur les perceptrons
1982	Hopfield	Modèle en relation avec les systèmes physiques
1989	Kohonen	Auto-organisation et mémoires associatives
1992	Boughanem	Réseaux de neurones appliqués à la recherche d'informations

2.2. Fondements biologiques

Les chercheurs en intelligence artificielle sur les réseaux de neurones se sont inspirés des modèles biologiques (cerveau et neurones).

2.2.1. Le cerveau

Le cerveau est le centre de contrôle de la perception, de la décision, et de l'action. Le système nerveux traite l'information via des unités interconnectées entre elles : les neurones. Le cerveau en possède 10^{13} et chacun d'entre eux est connecté en moyenne à mille autres.

2.2.2. Le neurone

L'information circule entre les neurones sous forme d'impulsions électriques d'amplitude constante mais de fréquence variable (figure II.2.1). Cette information va être traitée, transformée et éventuellement amplifiée et propagée.

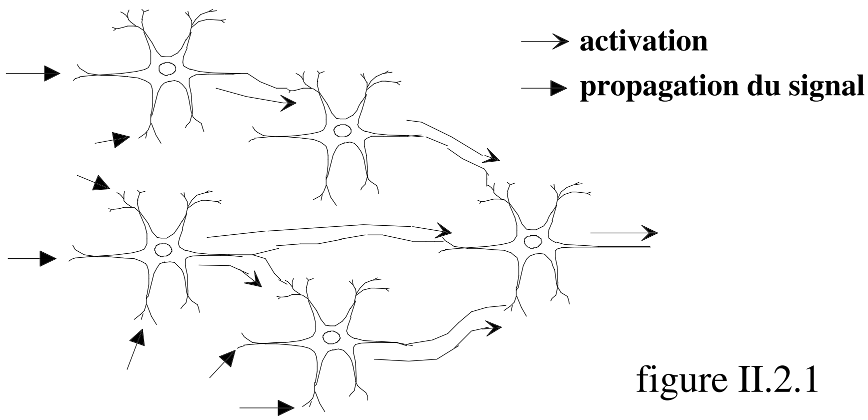


figure II.2.1

Le neurone (figure II.2.2) reçoit des impulsions des neurones afférents via les dendrites et effectue une sommation de ces impulsions (figure II.2.3). L'activité calculée est distribuée aux neurones connectés en sortie de l'axone.

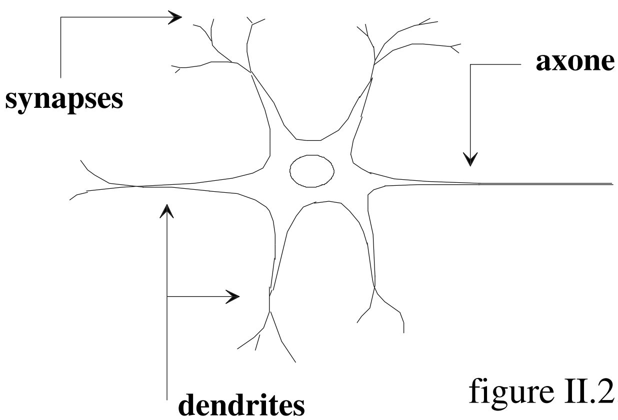


figure II.2.2

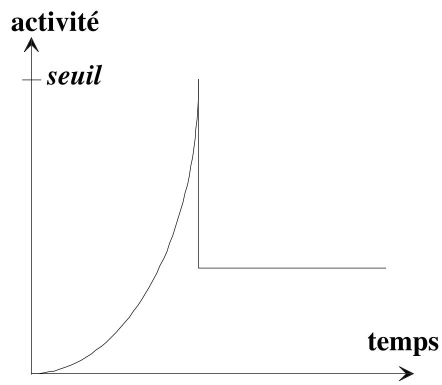


figure II.2.3

Les contacts entre les fibres nerveuses sont appelés synapses et ont un rôle quantitatif dans la transmission de l'information, elles peuvent être excitatrices ou inhibitrices.

2.3. Le neurone artificiel

Les structures biologiques et les mécanismes physiologiques des neurones sont extrêmement compliqués. Aussi, pour essayer d'en comprendre le fonctionnement, il a fallu introduire une forme simplifiée du neurone : le neurone formel.

Nous allons donc étudier le neurone formel de base, puis un neurone formel plus détaillé et plus général, le neurone formel étendu.

2.3.1. Le neurone formel de base

Le neurone formel a été présenté par Mc Culloch & Pitts en 1943, il se présente comme un opérateur de sommation dont les entrées sont pondérées (figure II.2.4).

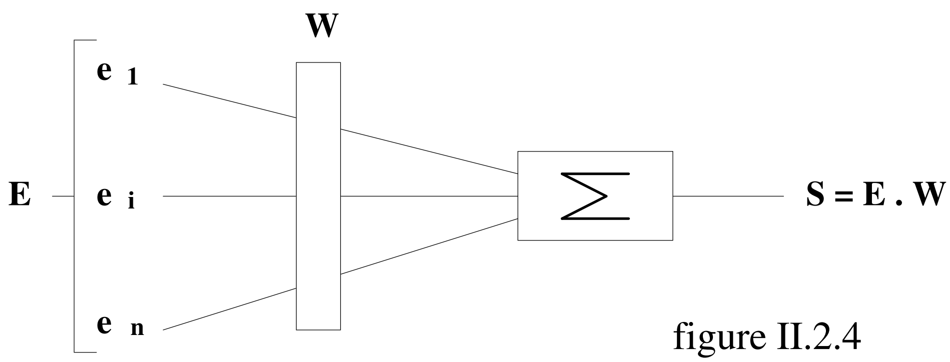


figure II.2.4

avec :

E(e1...ei...en) : vecteur d'entrée du neurone

W(w1...wi...wn) : vecteur des poids

S : sommation pondérée des entrées $S = \sum_{i=1}^{n} e_i . w_i$

La sortie S est souvent traitée par une fonction d'activation pour produire un signal de sortie.

2.3.2. Le neurone formel à fonction d'activation

Le modèle du neurone formel précédent est complété par l'association d'une fonction d'activation à la valeur S calculée. Nous pouvons distinguer [Davalo 89]:

Fonctions à seuil : - Heaviside : $F(x) = \begin{cases} 1 & si \quad x \geq 0 \\ 0 & si \quad x < 0 \end{cases}$

- signe : $F(x) = \begin{cases} +1 & si \quad x \geq 0 \\ -1 & si \quad x < 0 \end{cases}$

Fonctions sigmoïde : $F(x) = \dfrac{1}{1+e^{-\theta x}}$ ou $F(x) = \dfrac{1+e^{-\theta x}}{1+e^{+\theta x}}$

Les fonctions de type sigmoïde sont introduites par vraisemblance biologique pour prendre en compte l'importance de l'activation du neurone (figure II.2.3).

2.3.3. Nature des fonctions d'entrée, d'activation et de sortie

La nature des entrées et des sorties peut être soit binaire, soit réelle.

Par extension, le neurone formel étendu est défini ainsi (figure II.2.5):

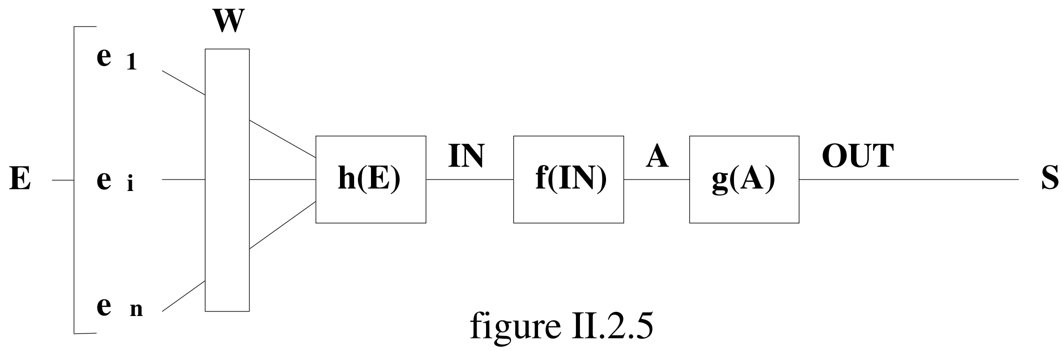


figure II.2.5

avec :

E(e1...ei...en) : vecteur d'entrée du neurone
W(w1...wi...wn) : vecteur des poids
IN = h(E) : fonction d'entrée totale (binaire, linéaire, affine ou polynomiale)
A = f(IN) : fonction d'activation (binaire à seuil, linéaire à seuil ou multi-seuil, sigmoïde, stochastique, ou toute fonction croissante et impaire).
S = OUT = g(A) : fonction de sortie (identité, ou toute fonction croissante et impaire)

2.4. Les connexions

Les connexions vont relier les neurones entre eux. **La structure des connexions** peut aller de la connectivité partielle à la connectivité totale. On associe à une connexion entre deux neurones, un **poids** qui rend compte de l'influence d'un des neurone sur l'autre. **La dynamique des états** correspond à l'évolution des états des différents neurones d'un réseau. Elle dépend à la fois des fonctions d'activation, de la structure et des poids des connexions. **La dynamique des connexions** représente le fait que les poids des connexions peuvent être modifiés par une phase appelée **l'apprentissage**. On peut assimiler cette dynamique à la plasticité synaptique.

2.5. Architectures de réseaux de neurones

L'organisation des réseaux de neurones peut se faire à partir de 2 grandes catégories d'architectures : les réseaux sans couches et les réseaux à couches.

2.5.1. Réseaux sans couches

Le réseau possède des neurones connectés entre eux, certains sont des neurones d'entrée et d'autres sont des neurones de sortie (figure II.2.6).

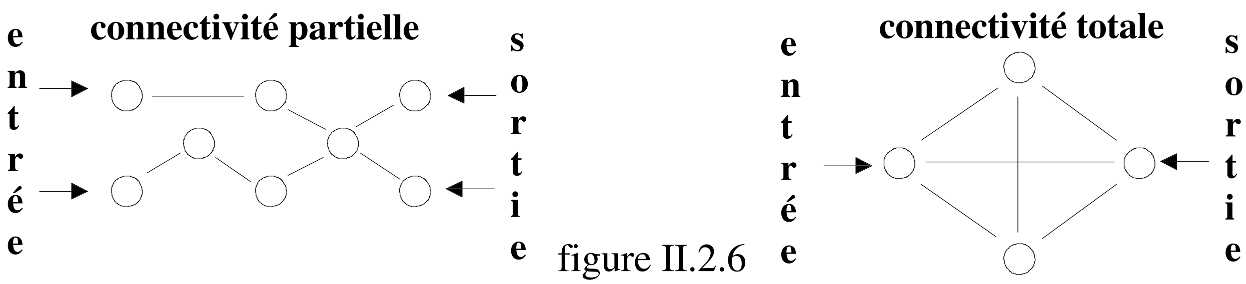


figure II.2.6

2.5.2. Réseaux à couches

La particularité de ce type de réseau est de posséder des couches de neurones indépendantes. Chaque couche a sa propre organisation, mais la connexion entre les couches est particulière. En effet, les neurones d'une couche C_i peuvent être connectés avec les neurones de la couche C_{i-1} et avec ceux de la couche C_{i+1}.
On distingue 3 types de couches : couche d'entrée, couches cachées et couche de sortie (figure II.2.7).

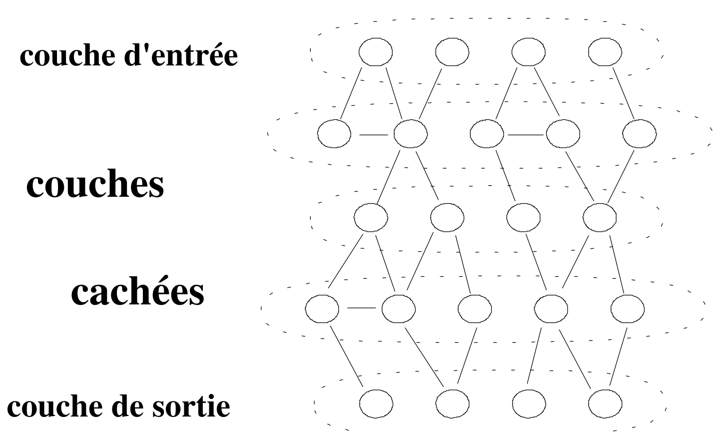


figure II.2.7

2.6. Modèles de réseaux de neurones

De nombreux chercheurs ont développé des topologies de réseaux de neurones, dont ils ont démontré les propriétés. Par la suite, ces topologies sont devenues des modèles de base dont on s'inspire pour créer de nouvelles architectures de réseau de neurones.

2.6.1. Le modèle du perceptron

Le modèle du perceptron ou le perceptron de Rosenblatt [Rosenblatt 57] est le premier modèle à couches opérationnel, mettant en oeuvre les idées de Hebb et de Mc Culloch & Pitts. Le perceptron est une machine de vision élémentaire composée de 3 couches interconnectées (figure II.2.8).

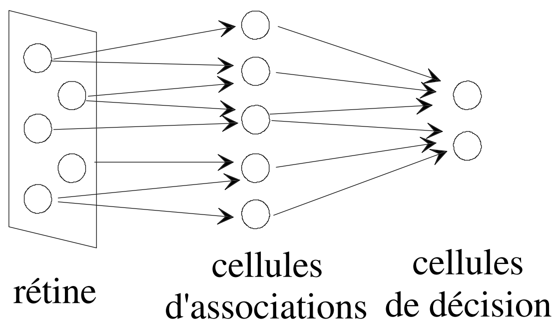


figure II.2.8

2.6.2. Le modèle de Hopfield

D'après Hopfield, le système nerveux recherche des états stables ou attracteurs dans son espace d'états. Les états voisins tendent à se rapprocher d'un état stable, ce qui autorise la correction d'erreurs et la capacité à compléter les informations manquantes [Hopfield 82]. Un réseau de Hopfield est composé de neurones binaires tous reliés entre eux, ce qui donne une connectivité totale (figure II.2.9).

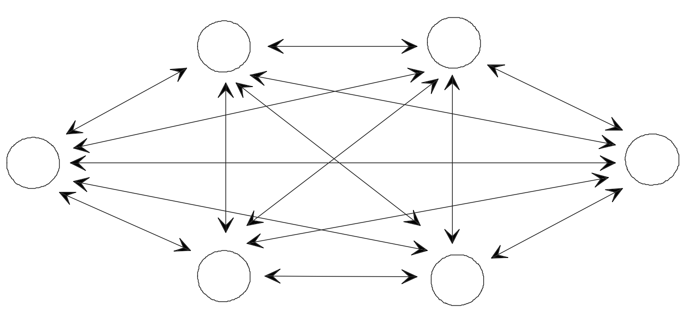


figure II.2.9

Lorsque la matrice des poids des connexions est symétrique, la stabilité du réseau est déterminée par le minimum d'une fonction d'énergie appelé **Attracteur**.

Si l'on peut démontrer la stabilité du réseau, le rappel des exemples appris n'est pas toujours réalisé. De plus, le nombre d'exemples mémorisables n'est pas très élevé.

2.6.3. Le modèle de Kohonen

Les réseaux du modèle de Kohonen [Kohonen 82] sont utilisés pour réaliser des opérations de classification et de regroupement. Ils sont composés d'une seule couche appelée couche d'auto-organisation (figure II.2.10). L'architecture tient compte des données extérieures des cellules d'entrée et des connexions internes.

Chaque neurone du réseau est connecté aux n composantes du vecteur d'entrée X. Puis, les neurones propagent le signal vers les autres neurones du réseau. Kohonen a montré qu'il existe entre les neurones émetteurs et récepteurs un mécanisme d'interaction latérale qui dépend de la distance entre les neurones concernés. Cette dépendance est caractérisée par la fonction du "chapeau mexicain" (figure II.2.11).

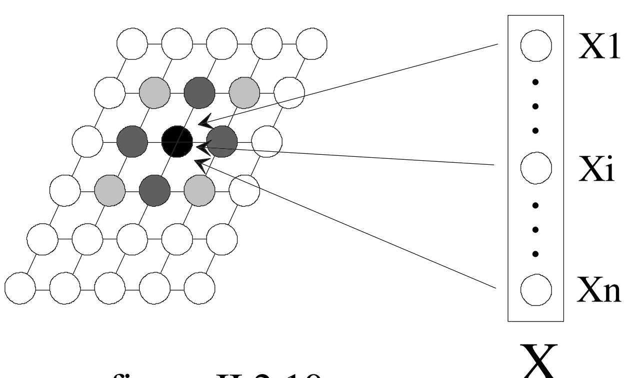


figure II.2.10

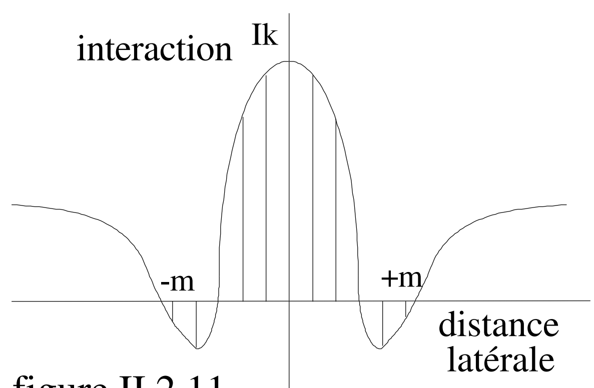


figure II.2.11

L'interprétation du comportement du réseau est la suivante :

- dans une zone latérale proche d'un neurone, les neurones qui lui sont connectés ont une action excitatrice,
- dans un voisinage plus lointain, l'action est inhibitrice,
- dans un voisinage encore plus lointain, l'action est négligeable.

La discrétisation de la fonction du "chapeau mexicain" va permettre de déterminer les poids des connexions internes entre les neurones (figure II.2.11).
Finalement, l'interaction latérale doit provoquer l'amas de neurones excités autour du neurone le plus activé par le stimulus.

2.7. Apprentissage dans les réseaux de neurones

2.7.1. Apprentissage non supervisé

2.7.1.1. Présentation

L'apprentissage modifie les connexions pour produire la sortie la plus consistante, mais il n'y a pas de sorties désirées comme dans l'apprentissage supervisé.

2.7.1.2. Règles d'apprentissage de Hebb

L'apprentissage se fait par les algorithmes suivants [Hebb 49]

$$w_{ij}^{t+1} = w_{ij}^{t} + O_i . O_j \text{ ou } w_{ij}^{t+1} = w_{ij}^{t} + (O_i^t - O_i^{t-1}).(O_j^t - O_j^{t-1})$$

avec w_{ij} : poids de la connexion entre le neurone i et le neurone j

O_i : sortie du neurone i

t : étape d'apprentissage (temps)

2.7.1.3. Règles d'apprentissage compétitif

L'apprentissage s'effectue par coopération et compétition entre les neurones, le poids des connexions évolue ainsi $W_i^{t+1} = W_i^t + \eta.(X_i - W_i^t)$ [Davalo 89]

avec W_i : vecteur poids des neurones en sortie du neurone j

X_i : vecteur d'entrée du neurone i

η : vitesse d'apprentissage

t : étape d'apprentissage

Ce genre d'apprentissage aboutit souvent à la stabilité et la convergence du réseau.

2.7.2. Apprentissage supervisé

2.7.2.1. Présentation

L'apprentissage supervisé consiste à modifier les paramètres du réseau de façon à ce que les réponses de ce dernier soient des réponses attendues. On présente une entrée et on compare la sortie obtenue à la sortie désirée puis on ajuste les poids des connexions pour minimiser la différence.

2.7.2.2. Apprentissage dans le perceptron

L'apprentissage du perceptron de Rosenblatt [Rosenblatt 57] est un apprentissage supervisé qui se fait par correction d'erreur. Il consiste à présenter au réseau une série d'exemples à apprendre, puis à minimiser l'erreur entre la sortie désirée S et la sortie effectivement obtenue Y.

L'algorithme est le suivant :

- présenter au réseau un exemple (E,S)
- calculer la sortie obtenue : $Y_j = \sum_i e_i . w_{i\,j}$
- pour chaque neurone, calculer : $w_{i\,j}^{t+1} = w_{i\,j}^{t} + \eta\ .(S_j - Y_j)$

avec $w_{i\,j}$: poids de la connexion entre le neurone i et le neurone j

S_j : sortie désirée pour le neurone j

Y_j : sortie obtenue pour le neurone j

η : coefficient représentant la vitesse d'apprentissage

t : temps

2.7.2.3. Apprentissage de Widrow-Hoff

Cet apprentissage est une extension du modèle du perceptron à sorties binaires à des sorties réelles [Widrow 60]. L'algorithme est le suivant :

Affecter des valeurs aléatoires aux connexions du réseau
Présenter un vecteur d'entrée $X(x_1, \ldots, x_n)$
Calculer les sorties $S_j = \sum_i x_i . w_{i\,j}$

Calculer l'erreur de chaque neurone $\varepsilon_j = Y_j - S_j$

Modifier le poids des neurones $w_{i\,j}^{t+1} = w_{i\,j}^{t} + \eta \;.x_i.\varepsilon_j$

avec $w_{i\,j}$: poids de la connexion entre le neurone i et le neurone j

S_j : sortie désirée pour le neurone j

Y_j : sortie obtenue pour le neurone j

η : coefficient représentant la vitesse d'apprentissage

2.7.2.4. Apprentissage par rétropropagation

L'apprentissage par rétropropagation est une généralisation de la règle de Widrow-Hoff [Widrow 60]. Le principe est de rétropropager l'erreur commise en sortie vers les couches internes du réseau (figure II.2.12).

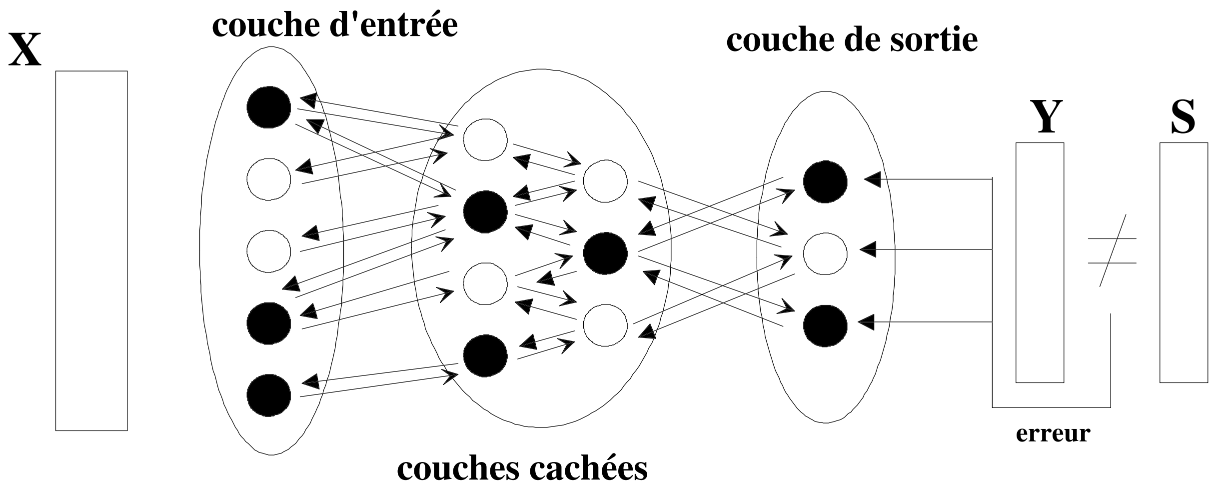


figure II.2.12

Le calcul de l'erreur commise pour le résultat à un exemple k est donné par

$$E^k(W) = \sum_{i=1}^{m} (Y_i^k - S_i^k)^2$$

avec S_j : sortie désirée pour le neurone j

Y_j : sortie obtenue pour le neurone j

E(W) : erreur quadratique

W : matrice des connexions

On effectue une modification du poids des connexions à chaque exemple k

$$w_{i\,j}^{k} = w_{i\,j}^{k-1} - e(k).d_i.O_j$$

pour la couche de sortie $d_i = 2.(S_i - Y_i).f'(I_i)$

pour les couches cachées $d_i = (\sum_h d_h.w_{i\,h}).f'(I_i)$

avec h : ensemble des neurones vers lesquels le neurone i envoi un signal

f : fonction sigmoïde, f' est sa dérivée

O_j : sortie du neurone j

I_i :entrée du neurone i,

e(k) : pas du gradient à l'étape k

2.8. Conclusion

Les systèmes à base de réseaux de neurones sont de plus en plus utilisés dans de nombreux secteurs et de nombreux domaines (reconnaissance des formes, de la parole, traitement de l'information, robotique).

Exemples d'applications de chaque modèle :

Modèles de réseaux	**Applications**
PERCEPTRON	reconnaissance de visages pour l'identification
HOPFIELD	modélisation de systèmes dynamiques (physique)
KOHONEN	approximation de données spatio-temporelles inconnues en cartographie

Ils peuvent s'appliquer dans la plupart des cas à tous les systèmes de reconnaissance et d'approximation de données, et en général dans tous les systèmes où l'apprentissage s'avère indispensable. En effet, nous avons vu que grâce à sa propriété d'apprentissage, un réseau de neurones peut changer son comportement d'après les résultats de son expérience passée.

Dans la suite de ce rapport, nous présentons l'utilité des réseaux de neurones pour l'EIAO. Leurs propriétés d'apprentissage et de traitement de l'information temps réel nous donnent les outils nécessaires pour faire reculer certaines limites des STI.

III - Réseaux de neurones et évaluation de l'écart Elève / Tuteur

1. Architecture générale d'un STI pour l'entraînement : intégration du RN

Nous nous basons sur l'architecture d'un schéma de tuteur intelligent pour l'entraînement (module interface, module tuteur, module expert, module de l'élève). Pour un exercice donné, l'élève et le module expert proposent leurs solutions.

Classiquement, le module tuteur évalue l'écart entre les deux solutions et agit en conséquence (mise à jour des connaissances, guidage, sollicitation, ...). La gestion du modèle de l'élève est effectuée souvent à partir d'un profil de référence, et certains modèles gardent l'historique des sessions antérieures. L'idée est donc, d'une part d'intégrer le profil de l'apprenant dans un module de l'élève, et d'autre part de garder un historique des actions entreprises dans un module de simulation d'un élève virtuel. Cet historique est stocké sous forme de règles de production. Il est enrichi constamment par une généralisation du comportement de l'élève durant les sessions antérieures.

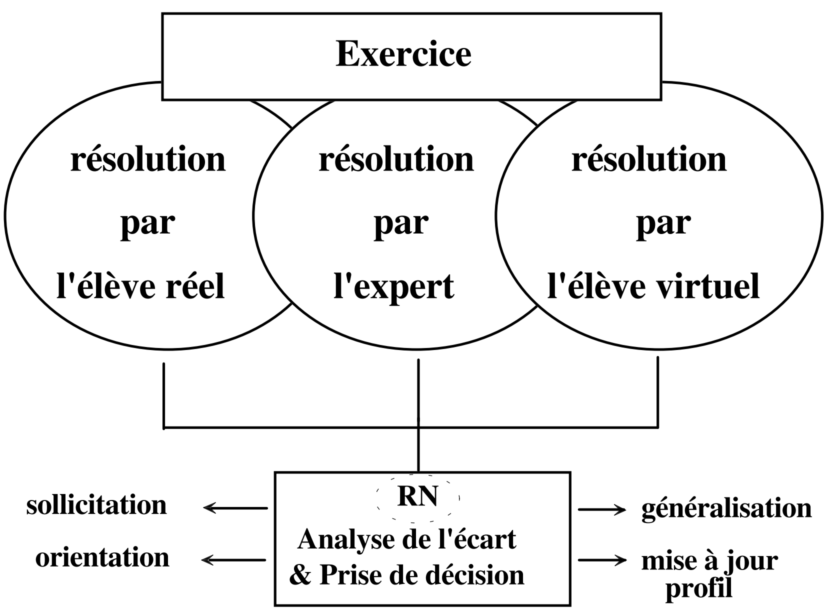


figure III.1

Les STI ont des difficultés à mesurer l'écart entre les solutions et à prendre en compte le comportement de l'élève. Nous pensons que l'intégration d'un réseau de neurones à notre architecture devrait permettre de lever ces difficultés.

Ce dernier interviendra lors de l'analyse de l'écart et de la prise de décision dans le cycle didactique (figure III.1).

Le module expert construit sa **solution experte** pour un problème donné, l'élève donne, pour le même problème, sa **solution réelle**. Le module simulateur de l'élève virtuel propose la **solution virtuelle** qu'il a inférée à partir de sa base de connaissances (figure III.2). Ces trois solutions forment les entrées du réseau de neurones.

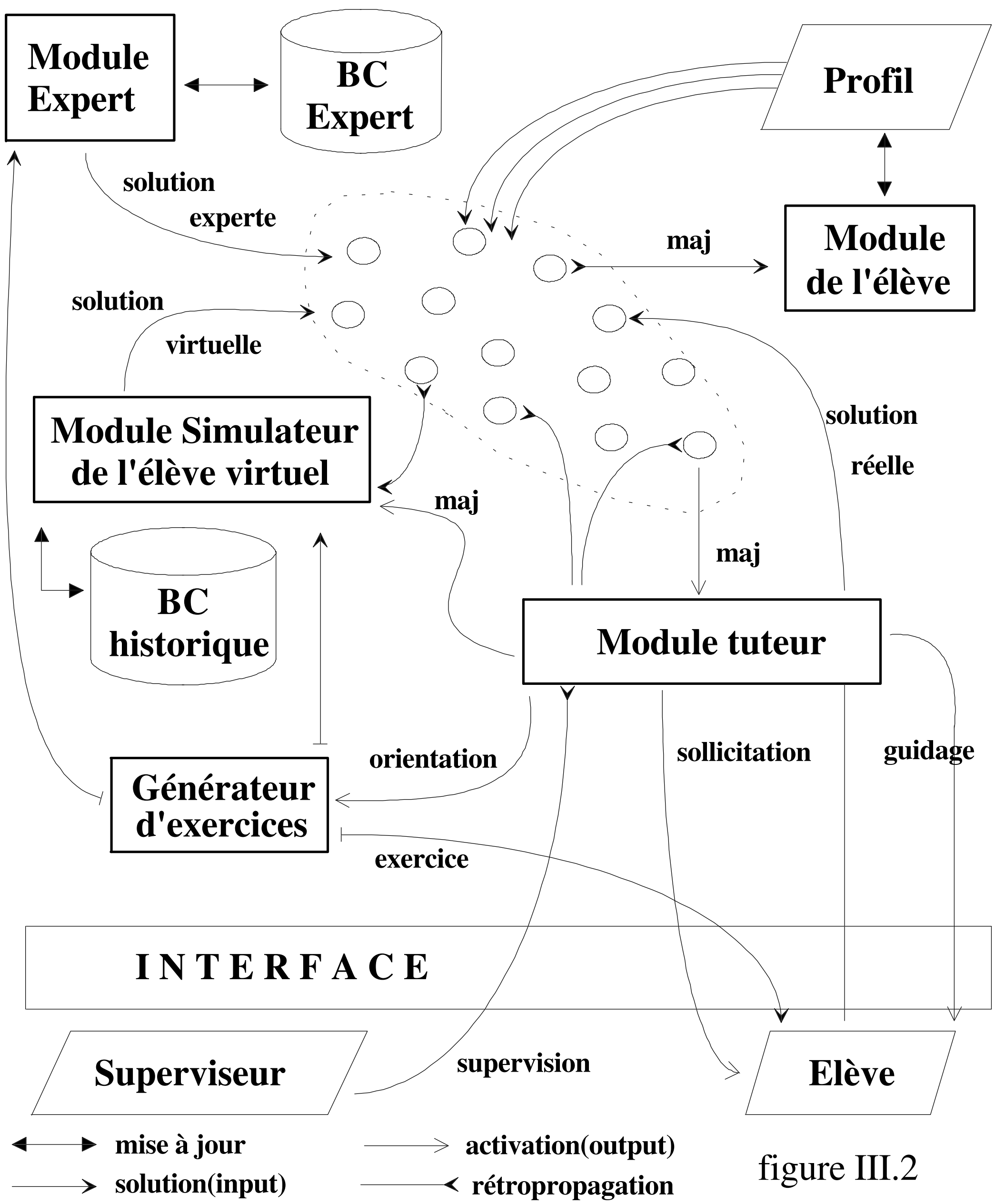


figure III.2

Le réseau est chargé de "mesurer" l'écart entre les différentes solutions. Le module de l'élève met à jour son profil. Le module tuteur analyse la sortie du réseau pour prendre la décision finale, et va par exemple guider l'élève plus ou moins fortement. Il informe également, le module simulateur de mettre à jour sa base de connaissances caractérisant le comportement de l'élève.

Nous pouvons également imaginer la possibilité de faire intervenir un nouvel acteur externe, le superviseur (méta-tuteur). Ce dernier peut être en réalité soit l'enseignant, soit le pédagogue. Il interviendra pour juger de la pertinence de l'intervention du STI dans le cas d'une sollicitation ou dans celui d'un guidage.

2. Générateur d'exercices

Le générateur d'exercices est un module qui génère aléatoirement un exercice par rapport à un niveau et à une difficulté donnée. L'exercice est ensuite proposé au module expert, au module simulateur et à l'élève pour être résolu.

3. Module Expert

Le module expert utilise les règles de sa base de connaissances pour élaborer la ou les solutions pour résoudre l'exercice proposé par le générateur. Les règles se présentent sous la forme **SI** *<conditions>* **ALORS** *<actions>*, elles permettent de décider quelles sont les actions à entreprendre (outils à utiliser) dans des conditions données.

Les outils disponibles sont définis ci-dessous sous forme procédurale :

- DDI(n,**di,r**) : Décomposer à la dizaine inférieure un entier n=di+r
  exemple : DDI(n=14,**di,r**) => di=10 et r=4
- DDS(n,**ds,r**) : Décomposer à la dizaine supérieure un entier n=ds-r
  exemple : DDS(n=37,**ds,r**) => ds=40 et r=3
- D(n,**r1,r2**) : Décomposer un entier n=r1+r2
  exemple : D(n=27,**r1,r2**) => r1=25 et r2=2
- P(**n1,n2**) : Permuter deux entiers dans l'expression
  exemple : P(**n1**=3,**n2**=17) => n1=17 et n2=3
- C(n1,n2,**r**) : Calculer la somme de deux entiers consécutifs r=n1+n2
  exemple : C(n1=4,n2=12,**r**) => r=16

Les règles de l'expertise et les outils ont été définis par un expert pédagogue pour apprendre les techniques du calcul mental sur des résolutions d'exercices du type ab+c, où a,b et c sont des entiers naturels à un chiffre [Hau Palé 94].

SI (a>0 et b<=5 et c>=5 et (b+c>=10))
ALORS DecomposerDizaineInférieure(ab, d, r),
x = Calculer(c + r),
add = Calculer(d + x)

SI (b<=5 et c>=5 et (b+c>=10))
ALORS DecomposerDizaineSupérieure(c, d, r),
x = calculer(ab + dc),
add = calculer(x - rc)

Les solutions générées forment un ensemble de séquences d'outils. Soit E_o l'ensemble des outils disponibles alors $E_o = \{DDI, DDS, D, C, P\}$.
Ainsi, les solutions de l'expert sont données par l'ensemble $\Omega_x = \{..., \Delta_i, ...\}$ des séquences $\Delta_i = \{..., O_j, ...\}$ d'outils $O_i \in E_o$. Ces séquences d'outils peuvent être de longueur quelconque.

Les règles expertes seront formalisées par $R_x = \langle \rho, C, A \rangle$
où $-1 \leq \rho \leq 1$: pertinence de la règle fixée à 1 pour l'expert
A : actions, séquence d'outils de E_o
C : prédicat, conjonction de conditions $C = \{..., \Phi_i, ...\}$
avec Φ_i condition (exemple: b+c>10)

4. Module de simulation du comportement de l'élève

Le module simulateur construit un ensemble de solutions à partir de sa base de connaissances. Cette base de connaissances est composée de règles de la forme **SI** *<conditions>* **ALORS** *<actions>*, mais elle n'est pas identique à la base de connaissances de l'expert.

Ces règles vont permettre de simuler le comportement de l'élève, elles doivent donc le refléter.

En effet, la généralisation des règles se fera par analyse et prise en compte du comportement de l'élève au cours des sessions d'apprentissage effectuées. La généralisation permettra d'intégrer de nouvelles règles et de les pondérer pour en déterminer la pertinence.

Les règles virtuelles sont formalisées par $R_v = \langle \rho , C , A \rangle$.

5. Module de l'élève : intégration du profil de l'élève

Le profil de l'élève va permettre de caractériser ce dernier sous une forme qui sera exploitable par le système durant les sessions d'apprentissage.

Il se compose de son niveau de progression (très faible, faible, médiocre, moyen, assez bien, bien, très bien) [Nicaud 88] et des caractéristiques de ses habitudes évocatives.

Le profil de l'élève se caractérise par son niveau et les scores (essais, échecs, mauvais usages, temps de réponse) [Mengelle 95] qu'il a réalisés, permettant ainsi d'évaluer l'utilisation des outils durant les sessions antérieures.

Le profil sera donc formalisé par $P = \langle N , S_{DDI} , S_{DDS} , S_D , S_P , S_C \rangle$ avec

N : niveau de progression de l'élève $N \in [1..7]$
S_o : Score pour l'outils o, avec $S_o = \langle N_e , N_c , N_m , \Delta_r \rangle$
N_e : Nombre d'essais d'utilisation de l'outils
N_c : Nombre d'échecs d'utilisation de l'outils
N_m : Nombre de mauvais usages d'utilisation de l'outils
Δ_r : Temps moyen de réponse d'utilisation de l'outils

Nous distinguons le nombre d'échecs d'un outil, lorsqu'il est mal utilisé, du nombre de mauvais usages, lorsque son utilisation n'est pas pertinente ou mal appropriée.

6. Module tuteur et interface

Le module tuteur va s'occuper de récupérer la solution de l'élève, qu'il va ensuite rediriger en entrée du réseau de neurones. Puis, il observera les sorties du réseau pour déclencher les opérations de sollicitations, de mise à jour du profil, de généralisation de règles, et d'orientation de la poursuite de la session.

L'élève développe sa solution suivant le concept de la "raisonnette". Il travaille sur un écran à l'aide de boutons ayant des fonctionnalités relatives à la résolution du problème.

Nous disposons d'une interface (Annexe 1) proposant dans une fenêtre, le problème à résoudre et dans une autre, les informations d'aide. Nous disposons également d'un écran qui propose un éventail d'outils à utiliser (permutation, addition...) ainsi qu'un pavé numérique à l'écran (pour cliquer sur les nombres).

IV - Module moteur de réseaux de neurones

1. Architecture d'un Environnement de Réseau de Neurones (ERN)

Un ERN est un environnement contenant un réseau de neurones, ainsi qu'une solution experte et une solution virtuelle. Le module moteur de réseau de neurones va permettre leur gestion par un ensemble de primitives (Moteur N++).

1.1. Processus de génération d'ERN

Les ERN sont générés à partir des solutions expertes et virtuelles, et sont évalués lors de l'acquisition de la solution réelle.

Le module expert et le module simulateur vont générer respectivement un ensemble de solutions expertes et un ensemble de solutions "virtuelles" (figure IV.1).

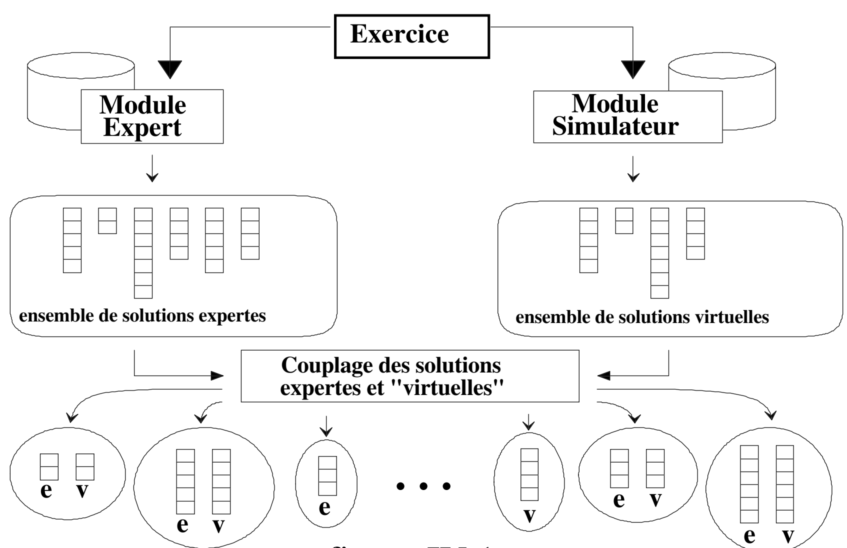


figure IV.1

Les solutions générées ne sont pas de longueur égale (cf. page 29), or pour que le réseau de neurones puisse analyser les solutions, elles doivent être de longueur identiques.

Nous allons donc créer des couples de solutions expertes et "virtuelles" de même dimension. Si une solution experte (réciproquement "virtuelle") n'a pas de solution associée "virtuelle" (respectivement experte), alors on aura un singleton solution.

L'étape suivante consiste à créer un environnement de réseau de neurones pour chaque couple ou singleton solution (figure IV.2).

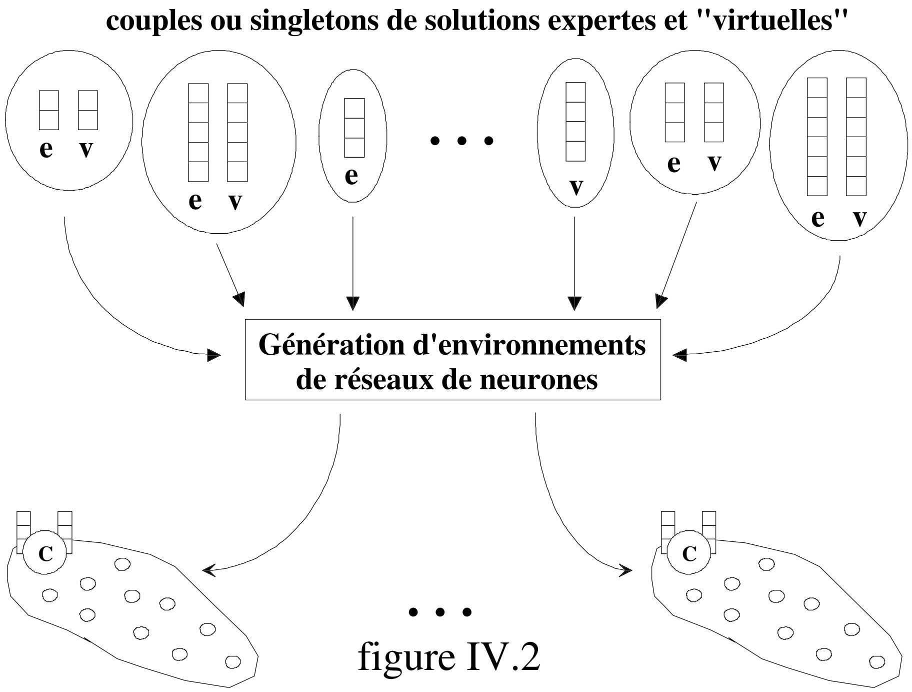


figure IV.2

Ainsi chaque environnement possédera des solutions uniques qui seront les entrées de son réseau de neurones (figure IV.3). Le contrôleur d'environnement va extraire un élément dans la file solution experte, ainsi que dans la file solution "virtuelle".

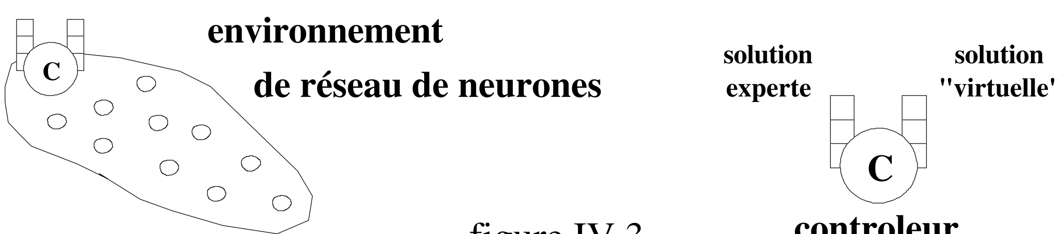


figure IV.3

Il ne reste plus qu'à effectuer l'acquisition de la solution de l'élève en temps réel, et à la distribuer à chaque environnement.

A chaque réception d'un événement[3] par le contrôleur, celui-ci reconnaît l'outil associé et extrait un élément des files solutions expertes et "virtuelles". Puis, il active les neurones des couches d'entrées (figure IV.4).

[3]clic bouton, choix d'outils.

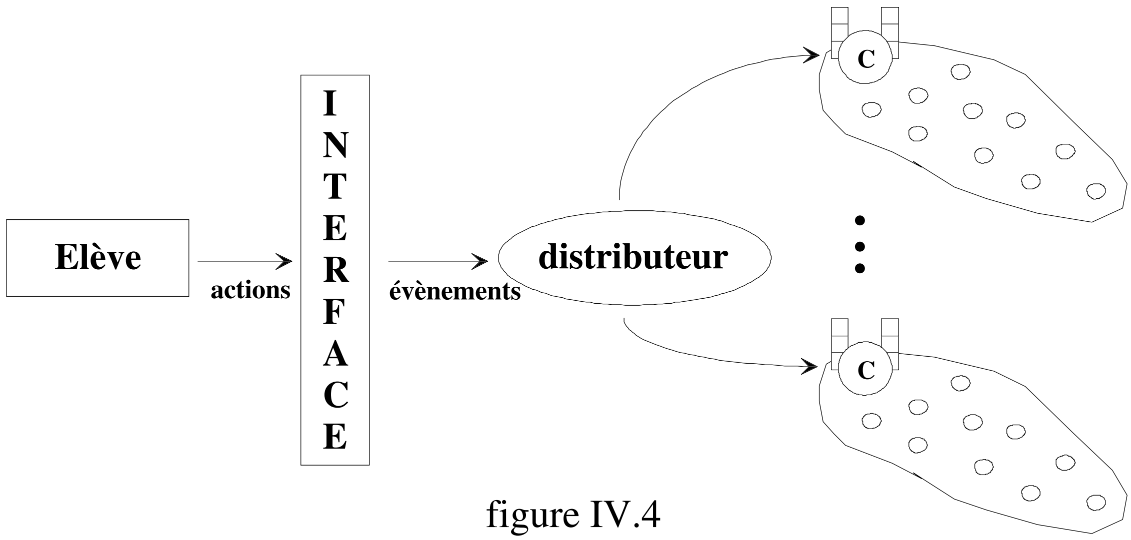


figure IV.4

Lors de la réception d'un événement par un contrôleur, si l'une des file est vide alors l'environnement est détruit. S'il ne reste plus qu'un environnement et que la solution de l'élève est de dimension supérieure à celle des solutions expertes ou "virtuelles" alors l'élève sera averti que sa solution est trop longue et donc trop complexe, et le système pourra lui suggérer de recommencer ou de revenir en arrière.

1.2. Modèle de réseau de neurones de l'ERN

Le modèle de réseau choisi doit permettre l'intégration des multiples informations d'entrées (expertes, réelles, virtuelles) et de celles de sorties (sollicitation, orientation, mise à jour du profil, guidage). L'information se propage dans le réseau des entrées vers les sorties. Elle doit être traitée et transformée pour reconnaître les entrées présentées et produire la sortie la plus convenable.

Aussi, le modèle à couche du type du perceptron nous semble le plus approprié.

Notre réseau va s'organiser sur trois niveaux (figure IV.5):

- les couches d'entrées effectuent l'acquisition des éléments des différentes solutions en intégrant le profil de l'élève,
- la couche d'association mesure l'écart entre les solutions,
- la couche de décision détermine l'action à entreprendre.

Nous disposons de trois couches d'entrées de 5 neurones, chacun correspondant aux outils utilisés par les trois acteurs (expert, virtuel et réel). Ces neurones sont binaires, et seulement un neurone par couche peut être activé au même instant.

La couche d'association possède également 5 neurones associés aux outils. Elle est chargée d'intégrer les choix d'outils des trois acteurs et de mesurer l'écart.

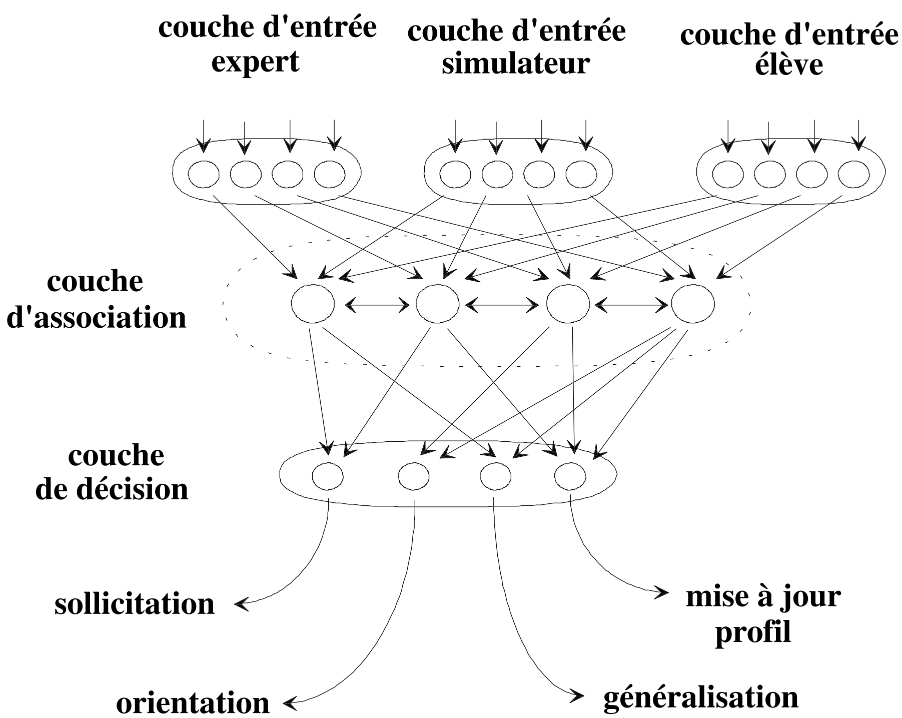


figure IV.5

La couche de décision est composée de quatre neurones auxquels on associe respectivement les concepts de sollicitation (intervention), d'orientation (changement de niveau de difficulté), de généralisation (intégration du comportement) et de mise à jour du profil (prise en compte d'une habitude évocative).

Ce modèle va nous permettre d'acquérir pas à pas les solutions et de décider en temps réel de la suite des opérations. Cependant, ce modèle est simplement réactif et il ne peut pas s'intégrer à un niveau cognitif.

En effet, nous ne savons pas déterminer explicitement la connectivité du niveau cognitif. Néanmoins, nous pensons pouvoir l'atteindre en utilisant le principe de la boîte noire. Ceci se traduit par l'insertion de couches cachées de neurones.

Ainsi, à l'initialisation du système, le nombre de couches cachées et le nombre de neurones est minimal, et la connectivité est soit aléatoire, soit maximale. La modélisation du niveau cognitif est donc liée à l'éclaircissement de la boîte noire, c'est-à-dire à la détermination d'une connectivité (figure IV.6). Nous appelons cette phase, la phase d'apprentissage (cf. page 7, figure I.2).

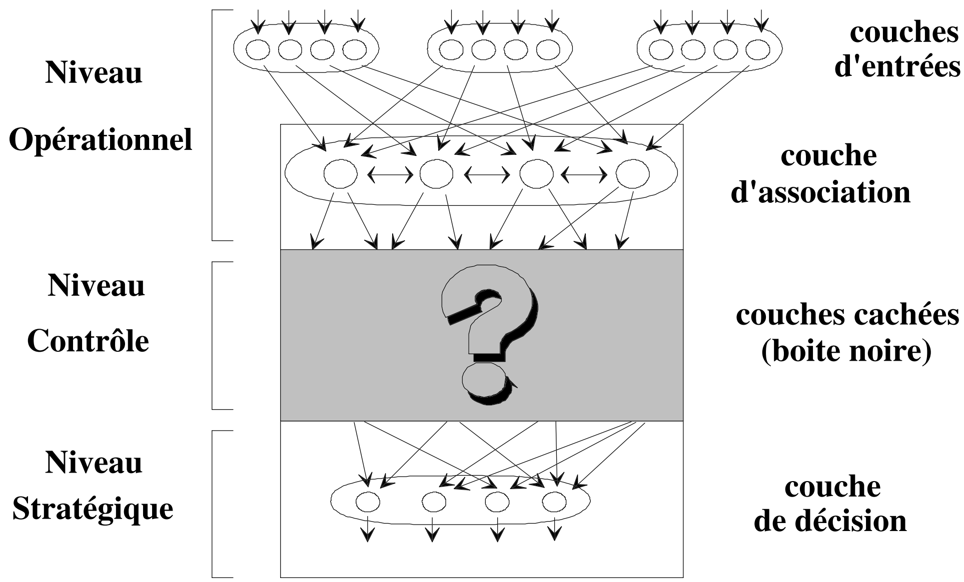


figure IV.6

Cette phase est caractérisée par un cycle dans lequel un apprentissage supervisé contribue à accroître les connaissances à partir d'exemples sur le domaine. Or, nous disposons déjà d'une expertise et nous pouvons dénombrer facilement les exemples stratégiques type. En effet, chaque acteur (expert, réel, virtuel) propose sa solution, c'est-à-dire qu'il choisit un outil parmi cinq, d'où $5^3 = 125$ exemples.

Ainsi, pour chaque exemple type on génère aléatoirement une sortie désirée. En effet, on suppose que si l'apprentissage est correct pour des sorties aléatoires alors il sera correct pour des sorties consistantes définies par un expert pédagogue.

Dans le modèle actuel (modèle à couches), les seules entrées du réseau sont déterminées par le choix des outils des différents acteurs (expert, virtuel, réel) à une même étape. Cependant, on peut envisager une extension à ce modèle en bouclant en entrée du réseau à un instant **t** les sorties obtenues au temps **t-1**. On obtiendrait ainsi un modèle récurrent à couches (figure IV.7).

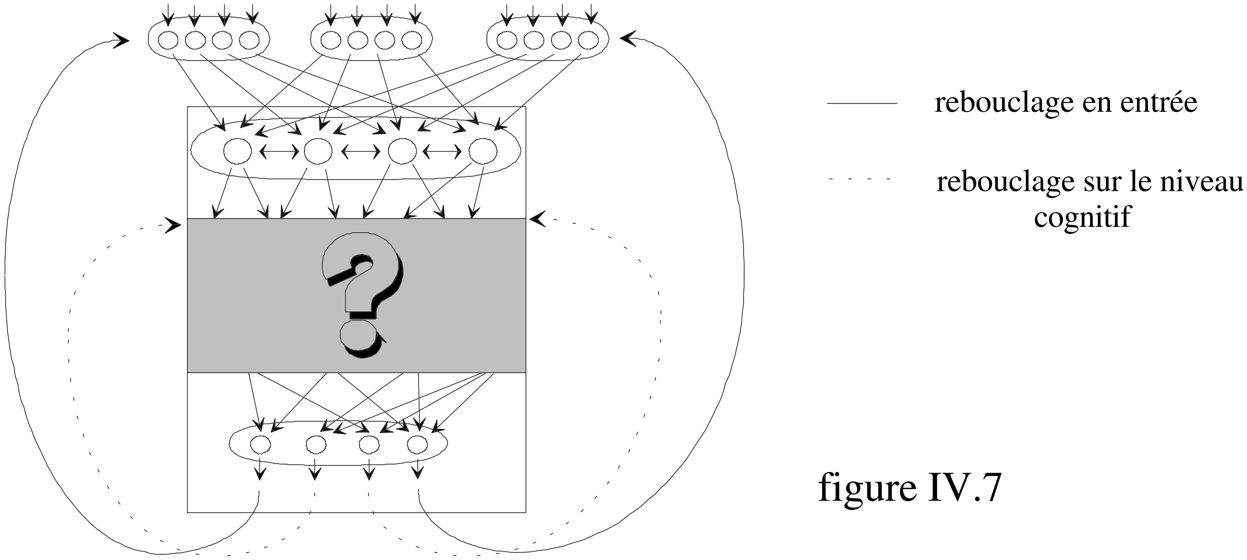


figure IV.7

Ce modèle permettrait une intégration temporelle de la stratégie au niveau cognitif. En effet, la stratégie de l'élève étant développée au fur et à mesure, l'écart mesuré par le réseau peut être parfois négligeable. Ainsi, il est possible qu'il ne soit pas toujours pertinent de prendre en compte la décision en sortie à chaque instant. Il faut donc cumuler cette sortie dans le temps pour qu'elle participe, plus tard, à une décision réellement consistante. Cependant, nous n'allons pas approfondir ce modèle récurrent qui est une extension du premier modèle, mais nous allons nous consacrer essentiellement au modèle non récurrent.

Nous procéderons donc à l'apprentissage supervisé des exemples type par une technique de rétropropagation de l'erreur. L'apprentissage de ces exemples est l'axe principal de recherche dans la suite de ce rapport. Nous essayerons de mettre en oeuvre et d'évaluer l'algorithme de rétropropagation, ainsi que d'autres techniques d'apprentissage basées sur cet algorithme.

Lors de l'utilisation du système, chacun des réseaux de neurones embarqués dans les agents continuera à s'adapter à l'élève par un apprentissage non supervisé [Hebb 49].

La finalité du réseau est d'enrichir le modèle de l'élève par un processus d'acquisition automatique de connaissances et d'améliorer la prise de décision. Le réseau décidera si le comportement de l'élève est assez caractéristique pour pouvoir l'intégrer à son modèle virtuel.

Néanmoins, avant de pouvoir exploiter l'ERN dans la phase d'utilisation, il faut éclaircir la boîte noire. Ainsi, une fois que l'apprentissage aura atteint un seuil de pertinence tolérable, on pourra fixer le modèle et évaluer la généralisation pas à pas sur des exercices en intégrant éventuellement le rebouclage des sorties du réseau.

Il est donc nécessaire de formaliser le réseau de neurones, la propagation de l'information, la rétropropagation de l'erreur, ainsi que d'autres techniques d'apprentissage.

2. Couches d'entrées

Les couches d'entrées se chargent d'intégrer les solutions expertes, réelles et virtuelles tout en prenant en compte le profil de l'élève. Cette intégration ne peut se faire que par l'explicitation du profil sous une forme numérique. Ainsi l'évaluation de l'activité de l'étudiant peut être réalisée [Frasson 92].

Il y a trois couches d'entrées, la couche d'entrée de l'expert C_x, la couche d'entrée de l'élève virtuel C_v et la couche de l'élève réel C_e. Les poids des connexions des couches d'entrées avec la couche d'association sont respectivement déterminés par les matrices C_xA, C_vA et C_eA.

Les indices des matrices correspondent respectivement aux outils de E_o.

$$C_xA=\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{bmatrix} \qquad C_vA=C_eA=\begin{bmatrix}S_1&0&0&0&0\\0&S_2&0&0&0\\0&0&S_3&0&0\\0&0&0&S_4&0\\0&0&0&0&S_5\end{bmatrix}$$

avec $S_i=S_{ii}=\left[\left(\theta_{stat}*STAT+\theta_{tps}*TPS\right)/\left(\theta_{stat}+\theta_{tps}\right)\right]*\left[N/7\right]$

$$STAT = 1 - \left[\left(o(i).N_c + o(i).N_m \right) / \left(2 * o(i).N_e \right) \right]$$
$$TPS = 1 - o(i).\Delta_r / \Delta_{\max}$$

où o(i) outils de rang i dans E_o et $i \in [1...5]$

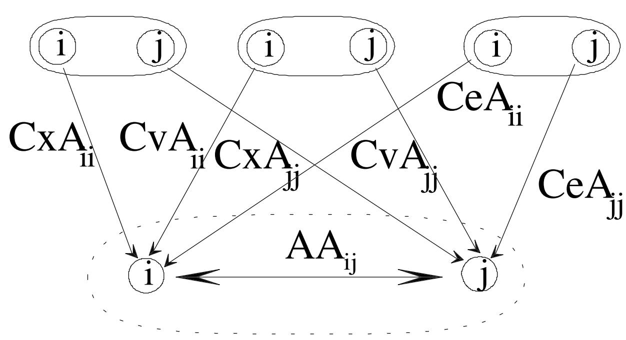
$\theta_{stat}, \theta_{tps}$ coefficients de priorité des statistiques et du temps établis par le pédagogue

Les scores sont fonction du niveau N de l'élève (cf. page 30) et de ses habitudes évocatives. En effet, les valeurs numériques STAT et TPS sont significatives de la maîtrise d'un outil au travers du nombre d'essais, d'échecs, de mauvais usages et du temps moyen de réponse. De plus, des coefficients de priorités permettront au pédagogue ou au tuteur de paramétrer, et ainsi de contrôler le niveau de progression de l'élève.

Les poids des connexions de la couche experte à la couche d'association sont à la valeur maximale 1 qui donne la plus grande importance à l'expert. Cependant, on pourra remettre en question l'expert dans le cas de l'intervention du méta-tuteur.
Les poids des connexions de la couche virtuelle et de la couche réelle avec la couche d'association sont identiques à l'initialisation du système. Ces poids pourront être modifiés par apprentissage.

3. Couche d'association

Les neurones de la couche d'association sont connectés entre eux pour permettre de mesurer l'écart entre les différentes solutions par concurrence. Les poids de ces connexions seront fixés par une formule inspirée de celle du barycentre.



Ainsi, la formule du barycentre va nous donner l'écart moyen des différences des connexions de deux neurones distincts de la couche d'association avec ceux respectifs des couches expertes, réelles et virtuelles.

La matrice obtenue est la suivante :

$$AA_{ij} = \begin{bmatrix} 0 & AA_{12} & . & . & . \\ AA_{21} & 0 & . & . & . \\ . & . & 0 & . & . \\ . & . & . & 0 & AA_{45} \\ . & . & . & AA_{54} & 0 \end{bmatrix}$$

avec $AA_{ij} = AA_{ji},\ AA_{ii} = 0$

$$AA_{ij} = \left[\alpha.Dx + \beta.Dv + \delta.De\right] / \left(\alpha + \beta + \delta\right)$$

$$Dx = \left|C_x A_{ii} - C_x A_{jj}\right|$$

$$Dv = \left|C_v A_{ii} - C_v A_{jj}\right|$$

$$De = \left|C_e A_{ii} - C_e A_{jj}\right|$$

α, β, δ : importance des solutions expertes, virtuelles et réelles

4. Couche de décision

La couche décision est composée de neurones qui vont permettre de déclencher des opérations d'orientation, de sollicitation, de généralisation et de mise à jour du profil. Les poids des connexions avec la couche cachée ne sont pas encore définis mais ceux-ci évolueront durant l'apprentissage. Nous supposons, pour cela, que la connectivité est totale à l'initialisation.

5. Fonctionnement du moteur

5.1. Formalisation

On définit un réseau de neurones par le triplet <I, H, O>

avec I = <NbI, $\left\{ \ldots, C_q^I, \ldots \right\}$> structure des couches d'entrées

H = <NbH, $\left\{ \ldots, C_q^H, \ldots \right\}$> structure des couches cachées

O = <NbO, $\left\{ \ldots, C_q^O, \ldots \right\}$> structure des couches de sorties

NbI, NbH, NbO: nombre de couches d'entrées, cachées et de sorties

C_q^X : couche de type X d'indice q avec $q \in \left[1 \ldots NbX\right] / X \in \left[I, H, O\right]$

soit C^k la structure de couche de base suivante:

$$C^k = < N, \beta, \beta', \theta, \eta, \varepsilon, \varphi, \gamma, NbIn, \left\{ \ldots, In_i^k, \ldots \right\}, NbOut, \left\{ \ldots, Out_j^k, \ldots \right\}, \Pi_{[N,N]} >$$

on a C^k: couche numéro k avec $k \in \left[1 \ldots (NbI + NbH + NbO)\right]$

N : nombre de neurones de la couche

β : seuil d'activation des neurones

β' : seuil de rétroactivation des neurones

θ : vitesse de propagation de l'information dans les neurones

η : vitesse d'apprentissage

ε : seuil de désapprentissage

φ : seuil de disparition des connexions

γ : seuil de rétropropagation

Tous les paramètres permettent d'initialiser les neurones, mais certains de ces paramètres pourront être modifiés durant les apprentissages.

NbIn : nombre de couches en entrée

$In_i^k = < C^{k'}, \Pi_{\left[N, C^{k'} :: N\right]} >$: i-ème couche d'entrée avec k>k'

NbOut : nombre de couches en sortie

$Out_i^k = < C^{k''}, \Pi_{\left[N, C^{k''} :: N\right]} >$: j-ème couche de sortie avec k<k''

$\Pi_{[n,m]}$: matrice n*m de connexions

Nous noterons **::** l'opérateur d'accessibilité à nos structures de couches et de réseaux.

La vitesse de propagation intervient dans les fonctions d'activation et de rétroactivation pour freiner ou accélérer le signal dans les neurones. Les seuils d'activation et de rétroactivation, eux, interviennent dans les fonctions de sorties.

Les fonctions suivantes sont inspirées de celles définies dans le modèle du neurone formel étendu (cf. page 18 figure II.2.5).

fonction d'entrée totale :

$h(...,e_j,...)=\sum_j w_{ij}.e_j$ somme des entrées (activations reçues par i) pondérée par les poids des connexions, où w_{ij} est le poids de la connexion entre le neurone i et le neurone j.

fonctions sigmoïdes d'activation :

$f_1(x)=1/(1+e^{-\theta x})$ ou $f_2(x)=f_1(x)-0.5$

ou $f_3(x)=(1+e^{+\theta x})/(1+e^{-\theta x})$ [Davalo 89] [Bough 95]

Il existe de nombreuses formules "sigmoïdes", aussi, lors de l'évaluation nous chercherons à déterminer une formule optimale pour notre application.

La fonction de sortie choisie est la fonction à seuil, intégrant ainsi le seuil β.

fonction de sortie : $g(x)=\begin{Bmatrix} x \; si \; x \geq \beta \\ 0 \; si \; x < \beta \end{Bmatrix}$

Le neurone i propage le signal si son activation est supérieure au seuil.

fonctions dérivées de rétroactivation : $f_1'(x)=f_2'(x)=\theta.e^{-\theta x}/(1+e^{-\theta x})^2$

ou $f_3'(x)=(e^{\theta x}-e^{-\theta x})/(1+e^{-\theta x})^2$

fonction dérivée de sortie : $g'(x)=\begin{Bmatrix} x \; si \; x \geq \beta' \\ 0 \; si \; x < \beta' \end{Bmatrix}$

Les fonctions dérivées sont indispensables pour effectuer la rétropropagation.

Les termes d'activation et de rétroactivation sont propres à l'activité interne d'un neurone. Tandis que les termes de propagation et de rétropropagation sont relatifs au sens du parcours de l'information et donc à l'activité générale du réseau. Dans la suite du modèle, on considérera la couche d'association comme la première couche cachée.

5.2. Propagation

La propagation se déroule ainsi :

- Les neurones outils de chaque couche d'entrée reçoivent leurs vecteurs d'activation tels qu'un neurone de chaque couche est activé. Ils propagent leurs activations sur la couche d'association via les connexions.
- Les neurones de la couche d'association calculent tout d'abord leurs entrées à partir des activations reçues, puis, ils calculent leurs propres activations qu'ils se propagent entre eux. Une fois que les activations de tous les neurones de la couche se sont propagées, ils recalculent leurs activations pour les propager cette fois, sur la première couche cachée.
- Les neurones des couches cachées reçoivent les activations, ils les traitent et éventuellement les propagent aux couches suivantes, dont celle de décision.
- Les neurones de la couche de décision effectuent le même travail, mais ils ne propagent pas leurs activations.

Nous présentons une entrée E au réseau qui nous donne sa sortie O^D, on note:

$E = \left\langle S_1, S_2, S_3 \right\rangle$: un exemple d'entrée

avec $S_i = \left(s_1, s_2, s_3, s_4, s_5\right)$

$$i \in [1...3], \forall k, k \in [1...5] (\exists j, j \in [1...5] / s_j = 1 \wedge s_k = 0 \wedge j \neq k)$$ [4]

$O^D = \left\langle o_1, ..., o_4 \right\rangle$: la sortie obtenue par le réseau pour cet exemple

où $0.0 \leq o_j \leq 1.0,\ j \in [1..4]$

I_i^m : résultat de la fonction d'entrée du neurone i de la couche m

[4]il n'y a qu'un seul neurone de chaque couche d'entrée qui est activé.

O_i^m : résultat de la fonction de sortie du neurone i de la couche m
$C^m::N$: nombre de neurones de la couche m
w_{ij} : poids de la connexion entre le neurone i et le neurone j

Nous avons formalisé la propagation ci-dessous :
Nous calculons la sortie de chaque neurone i sur chaque couche d'entrée k.

$$O_i^k = g(f(h(S_k))) \text{ avec } k \in [1...3]$$

Par la suite, nous effectuons la sommation des sorties O_j^{m-1} des neurones j de la couche m-1 pondérée par leurs connexions w_{ij} avec le neurone i de la couche m.

$$I_i^m = \sum_j^{C^{m-1}::N} (O_j^{m-1}.w_{ij}) \qquad O_i^m = g(f(I_i^m)) \text{ avec } m \in [1..NbH]$$

Puis, la sortie O_i^m de chaque neurone i de chaque couche cachée m est calculée.
Enfin, nous pondérons les sorties $O_j^{H::NbH}$ des neurones j de la dernière couche cachée H::NbH par le poids w_{ij}

$$I_i^D = \sum_j^{C^{H::NbH}::N} (O_j^{H::NbH}.w_{ij}) \qquad O_i^D = g(f(I_i^D))$$

et nous déterminons la sortie de chaque neurone i de la couche de sortie D.

5.3. Rétropropagation

La rétropropagation consiste tout d'abord à présenter un exemple <E,D> au réseau pour qu'il nous donne sa sortie O^D par une propagation, puis, une fois la sortie obtenue, nous rétropropagons l'écart de cette dernière avec la sortie désirée.

$D = \langle d_1, ..., d_4 \rangle$: sortie désirée pour l'exemple
où $0.0 \le d_i \le 1.0, i \in [1..4]$

La rétropropagation est représentée avec le formalisme suivant :
Nous déterminons l'erreur δ_i^D commise en sortie de la couche de sortie D pour chaque neurone i en intégrant la valeur d'entrée I_i^D obtenue lors de la précédente propagation, nous avons ainsi :

$$\delta_i^D = f'(g'(I_i^D)).(D_i - O_i^D)\,,\, i \in [1...4]$$

avec g' : dérivée de la fonction de sortie g
f' : dérivée de la fonction d'activation f

Cette erreur est rétropropagée sur les couches cachées m, $1 \leq m \leq H::NbH$, où nous effectuons la sommation des erreurs δ_j^{m+1} commises sur chaque neurone j de la couche m+1, pondérées par le poids w_{ij}^{m+1} des connexions. Puis, nous intègrons la valeur d'entrée I_i^D obtenue lors de la précédente propagation.

$$\delta_i^m = f'(g'(I_i^m)).\sum_j^{C^{m+1}::N} (w_{ij}^{m+1}.\delta_j^{m+1})$$

avec I_i^m : résultat de la fonction d'entrée du neurone i de la couche m
$C^m::N$: nombre de neurones de la couche m
w_{ij} : poids de la connexion entre le neurone i et le neurone j
H::NbH: nombre de couches cachées

5.4. Apprentissage

Nous distinguons deux types d'apprentissage dans nos réseaux de neurones.

- Apprentissage inter-neuronal

Lors de la rétropropagation on effectue une modification des poids des connexions du réseau.

Lors de l'évaluation, on effectuera des tests pour rechercher la formule optimale.

Les poids des liens seront ajustés par $w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$, on propose les différentes formules suivantes :

$$\Delta_1 w_{ij} = \eta.\delta_j^{m+1}.O_j^{m+1}$$

$$\Delta_2 w_{ij} = \eta.\delta_j^{m+1}.O_i^{m}$$

$$\Delta_3 w_{ij} = \eta.\delta_j^{m+1}.w_{ij}^{old}$$

- Apprentissage intra-neuronal

Lors de la rétropropagation on effectue une modification des seuils des neurones du réseau.

Les seuils d'activation des neurones seront modifiés par $\beta_i^{new} = \beta_i^{old} + \Delta\beta_i$

avec $\Delta\beta_i^m = \eta. \sum_{j}^{C^{m+1}::N} (w_{ij}.\delta_j^{m+1}).(O_i - \beta_i)$

Nous pensons ainsi obtenir un auto-ajustement du seuil qui va tendre vers la sortie "idéale" pour chaque neurone.

Il sera intéressant d'évaluer la qualité de l'apprentissage obtenu par chaque méthode, ainsi que celui obtenu par les deux méthodes combinées.

Cependant, nous pouvons envisager de nouvelles techniques d'apprentissage qui donneraient une modification des connexions plus restreinte et plus sélective, réduisant ainsi le désapprentissage.

6. Autres techniques d'apprentissage

Nous savons que l'apprentissage par rétropropagation a ses limites. S'il peut mémoriser parfaitement un exemple, par contre l'apprentissage de nouveaux exemples provoque le désapprentissage des précédents.

Dans le souci d'éliminer ce problème, nous proposons deux techniques:

- la fusion de réseaux de neurones (figure IV.8),
- la rétropropagation sélective dynamique.

6.1. Fusion de réseaux de neurones

Partant du constat que l'on sait faire apprendre un exemple à un réseau, nous proposons l'idée suivante. Si deux réseaux reconnaissent respectivement deux exemples distincts alors le réseau de neurones obtenu par la fusion de ces deux précédents réseaux reconnaîtra les deux exemples.

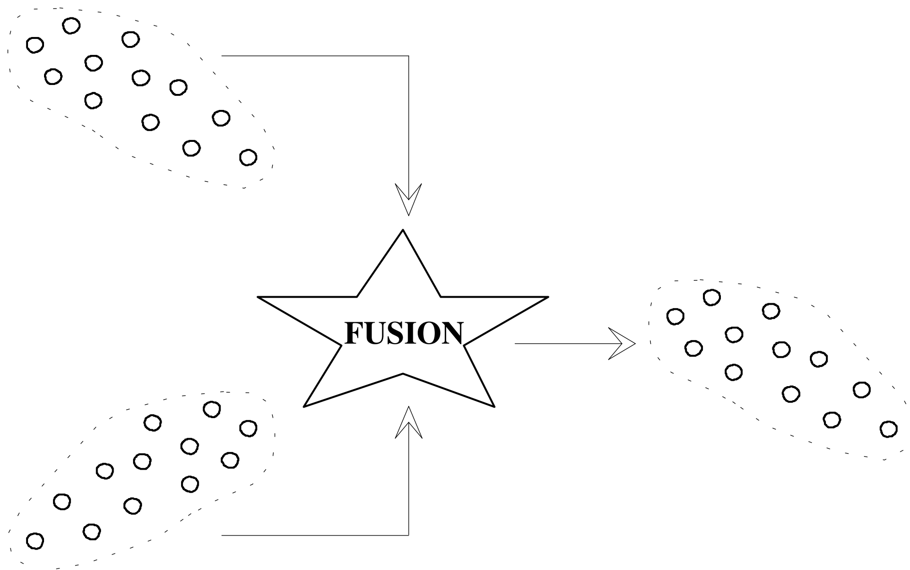


figure IV.8

Ainsi, si les expérimentations vont dans ce sens alors il semble probable que l'on puisse généraliser le processus à plus de deux exemples. La fusion provoquera une union des connexions et des neurones uniquement au niveau des couches cachées.

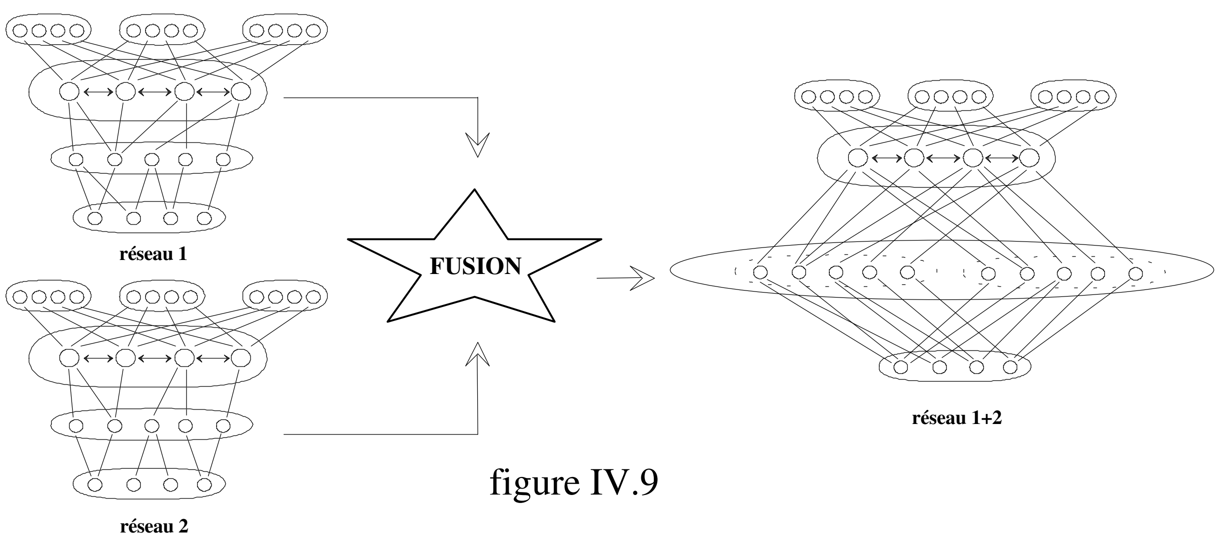


figure IV.9

Nous prenons comme convention que les connexions des couches d'entrée (niveau perceptif) sont identiques pour tous les réseaux. Ainsi, la fusion ne détermine que le niveau cognitif (figure IV.9), ceci se traduisant par l'ajout de connexions et donc l'extension des matrices des poids. On peut également envisager la possibilité de rajouter des intra-connexions inhibitrices pour empêcher la propagation par des connexions aboutissant à de mauvaises sorties.

6.2. Rétropropagation sélective dynamique

La rétropropagation sélective dynamique (RSD) repose sur le principe de la rétropropagation classique. Cependant, cette dernière a l'inconvénient de modifier toutes les connexions, y compris celles qui ne devraient pas l'être. Ceci provoque le désapprentissage.

Nous avons donc imaginé un algorithme de rétropropagation qui va modifier une connexion seulement si cela est nécessaire. Dans le cas où les connexions ne peuvent pas être modifiées, alors on rajoutera des neurones et donc de nouvelles connexions.

Il faut donc mettre en oeuvre un mécanisme de verrouillage des connexions. Pour cela, nous allons utiliser le seuil γ de rétropropagation et le seuil ε de désapprentissage de chaque neurone (cf. page 41).

L'algorithme est le suivant :
lors de la remontée de l'erreur par un neurone j à un neurone i via une connexion,

- si l'erreur est faible, la connexion est modifiée,
- si l'erreur est nulle, la connexion n'est pas modifiée,
- si l'erreur est trop forte et risque de provoquer un désapprentissage, on crée un nouveau neurone k qui va hériter de certaines des connexions du neurone i.

En réalité, à chaque réception d'une erreur nulle via une connexion, le seuil de désapprentissage, initialement fixé à 1, est progressivement abaissé. Lors de l'ajout d'un neurone, le seuil de rétropropagation devient nul et empêche définitivement la progression de l'erreur. Les connexions sont donc verrouillées et ne peuvent pas être modifiées.

Soit ε le seuil de désapprentissage du neurone i, initialement à 1
γ le seuil de rétropropagation du neurone i, initialement à 1
e^t l'erreur commise en j et rétropropagée en i au temps t

$$si \quad (\left|e^t\right| < \varepsilon) \wedge (\gamma \neq 0.0) \qquad w_{ij}^{new} = w_{ij}^{old} + \eta . e^t . w_{ij}^{old}$$

$$si \quad (\left|e^t\right| > \varepsilon) \wedge (\gamma \neq 0.0) \qquad w_{kj} = w_{ij} + \eta . e^t . w_{ij}, \quad \gamma = 0.0$$

$$si \quad e^t = 0.0 \quad \varepsilon = \min(e^{t-1}, \varepsilon)$$

L'héritage des connexions du nouveau neurone k par le neurone i peut être soit totale, soit partiel. Cependant, afin de faciliter les tests, nous effectuerons un héritage total (figure IV.10).

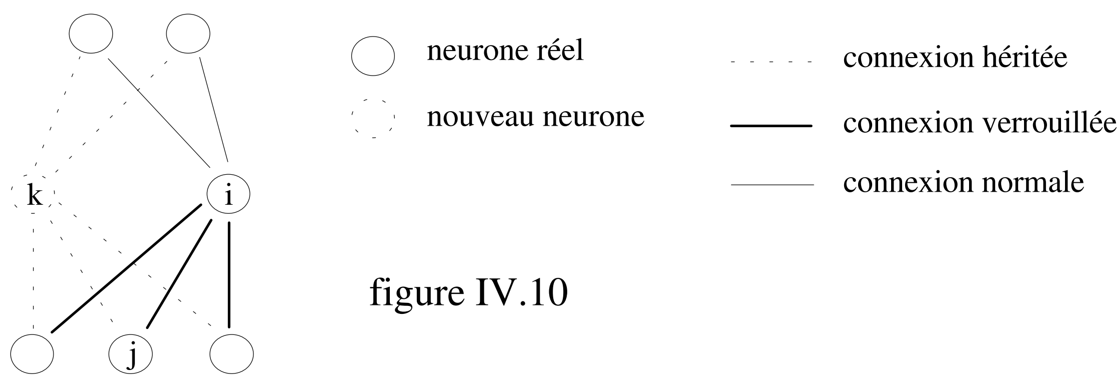


figure IV.10

Cet algorithme démarre avec un nombre minimal de neurones cachés qui vont voir leur nombre croître. Si les techniques de fusion et de RSD fonctionnent, il serait intéressant de comparer les modèles obtenus. Nous pourrons également utiliser simplement le mécanisme de verrouillage des connexions de la RSD lors de la fusion et lors de la rétropropagation classique. Ainsi, nous pourrons observer si nous obtenons de meilleurs résultats.

7. Optimisations

Afin d'accroître le rapidité de génération d'un réseau de neurones nous n'allons pas explicitement créer les connexions entre neurones. En effet, un neurone ne contiendra pas d'informations explicites (listes chaînées d'adresses) sur ses connexions, ainsi on ne perdra pas de temps à les créer.

Par contre, un neurone ne connaîtra que sa couche. A partir d'une couche, on pourra accéder à la ou aux couches précédentes et suivantes. Une couche peut contenir les adresses de ses neurones, les adresses des couches suivantes et précédentes, ainsi que les matrices des connexions avec ces dernières (Annexe 4).

De plus, pour accélérer la propagation du signal dans le réseau et donc l'apprentissage, nous allons doter les neurones de la faculté soit d'attendre, soit de ne pas attendre toutes ses activations. En effet, on remarque que la plupart des activations d'entrées sont nulles, d'où certains neurones vont attendre et propager une activation nulle ce qui est une perte de temps.

Pour résoudre ce problème, chaque neurone va cumuler ses entrées et calculer son état. Lorsque l'état dépassera le seuil alors on propagera le signal.

8. Expérimentations

Le moteur N++ est représenté à l'aide d'un formalisme objet (Annexe 2). Il est spécifié et implémenté en C++ (Annexe 3). Dans le but d'établir les connexions optimales qui rendent le comportement du réseau pertinent, c'est-à-dire transformer la "boîte noire" en "boîte claire", il faut pouvoir évaluer ce réseau.

Ainsi, nous mesurons l'apprentissage par le taux de rappel (***tr***). Ce dernier donne le rapport en pourcentage entre les sorties obtenues par le réseau et les sorties désirées. Nous proposons la formule suivante : $tr = \left[1 - \left(\sqrt{|d_i - o_i|}\right)/4\right] * 100$

avec d_i : sortie désirée pour le neurone i

o_i : sortie obtenue pour le neurone i

Cette formule nous donne un taux de rappel de 80% à 100% pour un exemple correctement appris et un taux inférieur à 80% pour un exemple qui ne l'est pas.

Nous avons expérimenté nos réseaux de neurones sur une base aléatoire d'exemples avec certaines configurations de paramètres (seuils, vitesse d'apprentissage, ...) fixés au plus bas pour qu'ils aient le moins d'influence. Ainsi, en évaluant tout d'abord les performances d'un réseau avec des paramètres neutres, nous pourrons, ultérieurement, analyser les influences de ces derniers.

Le réseau apprend itérativement chaque exemple suivant l'algorithme :

pour chaque exemple <E,D>
propager en entrée E et récupérer en sortie O
mesurer le taux de rappel
tant que le taux de rappel n'atteint pas **90%**
rétropropager l'erreur E - O
propager en entrée E et récupérer en sortie O
mesurer le taux de rappel

Cependant, le danger est de désapprendre les précédents exemples. Nous allons donc mesurer le taux de rappel de chaque exemple après l'apprentissage.
La mesure de l'apprentissage se fait par la comparaison de deux taux de rappels :

- **apprentissage :** l'apprentissage d'un exemple est stoppé et le dernier taux de rappel est préservé lorsque le taux demandé est atteint (par exemple 96%) ou lorsque le temps autorisé est dépassé (180 ou 300 secondes).
- **évaluation :** une fois tous les exemples appris, on évalue successivement le taux de rappel de chaque exemple pour mesurer le désapprentissage.

De plus, lors de tous les tests, nous avons relevé les durées d'apprentissage et les vitesses de convergence. La mesure de la durée d'apprentissage indique le temps en secondes nécessaire à l'apprentissage de chaque exemple. La mesure de la vitesse de convergence donne le nombre d'itérations (rétropropagations) par seconde nécessaire à l'apprentissage d'un exemple.

Tout d'abord, nous montrons qu'un réseau de neurones peut apprendre un exemple quelconque par la technique de rétropropagation de l'erreur. Lors de l'apprentissage d'un exemple, le taux de rappel progresse à chaque itération (Annexe 5.1). Durant ces mêmes itérations, l'erreur converge vers 0 montrant ainsi la réduction de celle-ci (Annexe 5.2). Cette réduction provoque l'ajustement des connexions, ce qui caractérise l'apprentissage de l'exemple (Annexe 5.3).

Le travail d'expérimentation consiste ensuite à déterminer les formules d'activation et d'apprentissage les mieux adaptées. Pour cela, nous construisons un réseau de neurones sur le modèle de l'ERN. Ce réseau comprend une couche cachée avec un neurone. Puis, nous allons le dupliquer autant de fois qu'il y a d'exemples à apprendre, c'est-à-dire 125 fois.

Chacun de ces 125 réseaux va apprendre un exemple distinct. Ces tests ont été effectués tout d'abord avec les trois formules d'activation f1, f2, f3 (cf. page 42). Puis, ils ont été poursuivis sur les formules d'apprentissage Δ_1, Δ_2, Δ_3 (cf. page 46). Les résultats nous donnent de meilleurs taux de rappel pour la formule d'activation f1 (Annexe 5.4) et la formule d'apprentissage Δ_1 (Annexe 5.5). Aussi, par la suite, tous les tests ont été réalisés à l'aide de ces deux dernières formules, ainsi que sur 10 exemples pour réduire la quantité des apprentissages et les temps de réponses.

Les séries de tests qui suivent, mettent en évidence les capacités et les limites de la rétropropagation totale[5], de la fusion et de la rétropropagation sélective dynamique.
L'objectif à atteindre est de déterminer un seul réseau reconnaissant plusieurs exemples. On ne peut donc pas tolérer d'avoir plusieurs réseaux qui reconnaissent seulement un exemple. Aussi, la rétropropagation totale et la rétropropagation sélective dynamique s'effectuent sur un réseau et plusieurs exemples alors que la fusion s'applique sur plusieurs réseaux pour n'en donner qu'un. De plus, lors des tests sur la rétropropagation totale et sur la fusion nous distinguons, un apprentissage facultatif du seuil. En effet, le seuil contribue à l'apprentissage car il intervient dans la propagation du signal. Ainsi, lorsque l'on observe son ajustement (Annexe 5.6), nous constatons qu'il converge vers un état stable.

Dans un premier temps, nous évaluons l'algorithme de rétropropagation totale. Nous savons qu'un neurone caché suffit à un réseau de neurones pour apprendre un exemple. Ainsi, nous supposons que 10 neurones cachés suffisent à l'apprentissage de 10 exemples par rétropropagation totale. Cependant, cet algorithme met en évidence le désapprentissage (Annexe 5.7) des exemples au profit du dernier.
L'intégration du seuil (Annexe 5.8) ne réduit pas ce désapprentissage mais elle diminue la vitesse de convergence ce qui n'est pas négligeable.

Dans la série suivante, nous observerons les résultats obtenus par la fusion de réseaux d'un neurone caché. Le taux atteint durant l'apprentissage est supérieur et plus constant que celui obtenu précédemment (Annexe 5.9). Pourtant, bien que la fusion soit moins coûteuse en temps, le désapprentissage est identique, et cela, même en effectuant l'ajustement du seuil (Annexe 5.10).

---

[5] apprentissage par rétropropagation de tous les exemples sur un seul et même réseau.

Puis, nous dotons la fusion du mécanisme de verrouillage de la rétropropagation sélective dynamique. Nous observons alors que la fusion sur deux réseaux s'effectue correctement et mémorise les deux exemples (Annexes 5.11 et 5.12.). Pourtant, dès le troisième, la fusion perd son efficacité (Annexes 5.11.). Nous cherchons actuellement une méthode de fusion des réseaux pour éliminer ce problème.

Enfin, nous intégrons le mécanisme de verrouillage à la rétropropagation totale (Annexe 5.13). Ainsi, nous espérions limiter la modification des connexions. Pourtant, bien que le taux de rappel d'apprentissage soit retrouvé lors de l'évaluation, ce dernier n'est pas assez élevé pour être pris en compte.

La rétropropagation sélective dynamique est actuellement en cours de tests et n'a donc pas été expérimentée. Cependant, les résultats sont encourageants puisqu'ils montrent qu'il est possible de faire apprendre à un réseau de neurones deux exemples. Nous espérons obtenir de meilleurs résultats en affinant nos techniques et en optimisant les paramètres qui, pour nos tests, étaient neutres.

Aussi, nos recherches s'orientent vers la fusion et la rétropropagation sélective dynamique. Nous envisageons également la possibilité de faire intervenir des techniques complémentaires pour ajuster les nombreux paramètres.

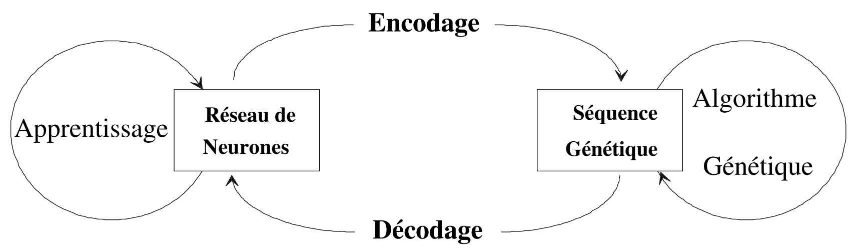
V - Conclusion & Perspectives

Nous avons étudié et conçu une architecture de STI intégrant un réseau de neurones. Ainsi, un moteur de réseau de neurones pour un STI dédié à l'apprentissage des techniques du calcul mental a été développé. Ce projet est donc double. D'une part, il a montré les apports possibles des réseaux de neurones aux STI pour la mesure de l'écart entre les stratégies élève/tuteur. D'autre part, il a permis d'approfondir des techniques d'apprentissage sur les réseaux de neurones (fusion et RSD).

Cependant, on s'aperçoit du phénomène de désapprentissage provoqué par la rétropropagation. Ceci un problème majeur dans les réseaux de neurones. En effet, lorsque la rétropropagation est totale, elle modifie toutes les connexions sans distinction. Les tests montrent qu'il vaut donc mieux envisager de mettre en oeuvre soit l'algorithme de rétropropagation sélective dynamique qui sélectionnera les connexions à modifier pour minimiser le désapprentissage, soit le mécanisme de fusion de réseaux de neurones. Pourtant, il reste à déterminer de nombreux paramètres pour optimiser ces techniques.

Aussi, pour effectuer la recherche des meilleurs paramètres, on pourrait imaginer une approche évolutionniste où un algorithme génétique optimiserait des réseaux de neurones. C'est d'ailleurs une des nouvelles voies de recherche [Bala 95] qui laisse entrevoir de nombreuses applications dans le domaine des sciences cognitives.

En effet, la modélisation et l'implémentation des systèmes à base de réseaux de neurones est "relativement" facile. Par contre, l'optimisation visant à accroître la vitesse d'apprentissage et de convergence se révèle trop longue, trop complexe et trop coûteuse de par les nombreux paramètres à manipuler. Ce problème trouverait sa solution dans une approche "génétique" dont l'objectif serait d'encoder et d'optimiser des réseaux de neurones par les algorithmes génétiques (figure ci-dessous).



Le principe est de coder le réseau de neurones (phénotype) en une séquence génétique (génotype) [Bala 95]. L'algorithme génétique va optimiser ce génotype à l'aide des opérateurs de sélection, de mutation et de croisement. Le réseau de neurones une fois décodé devra voir ses capacités accrues et notamment ses vitesses de convergence et d'apprentissage. Ceci est un thème de recherche vaste et ne fait pas l'objet de ce rapport, mais il pourrait en être le prolongement.

Le moteur est toujours en cours de test pour optimiser les différents paramètres, réduire la durée d'apprentissage, minimiser le désapprentissage et évaluer la capacité de généralisation. Une fois le moteur opérationnel, on effectuera la modélisation d'un ERN en intégrant les outils et des profils réels.

VI - Bibliographie & autres lectures

[Bala 95] Balakrishnan, K., & Honavar, V. Evolutionary Design of Neural Architectures - A preliminary Taxonomy and Guide to Literature. Tech. rept. CS TR95-01. Department of Computer Science, Iowa State University, Ames.

[Blanc 96] Blanc F., Une utilisation de la modélisation qualitative pour la planification pédagogique. ITS'96, Montréal, Canada, juin 1996 (*à paraître*).

[Bouch 92] Boucheron S., Théorie de l'apprentissage. Hermès, Paris 1992.

[Bough 92] Boughanem M., Les systèmes de recherche d'informations : d'un modèle classique à un modèle connexionniste. Doctorat de l'Université Paul Sabatier, Toulouse III, 1992.

[Bough 95] Boughanem M., MercureO2 : Un Système de Recherche d'Informations Orienté Objet basé sur l'Approche Connexionniste. Rapport de recherche interne MSI 95 - 05. Université de Limoges, décembre 1995.

[Bourret 91] Bourret P., Reggia J., Samuelides M., Réseaux neuronaux, une approche connexionniste de l'intelligence artificielle. Tekneau, Toulouse, 1991.

[Buchanan & Wilkins 93] Buchanan B.G., Wilkins D.C., Readings in Knowledge Acquisition and Learning. San Mateo, California : Morgan Kaufmann 1993

[Canut 90] Canut M.F., Spécification formelle de systèmes d'enseignement intelligemment assisté par ordinateur pour l'atelier de génie didacticiel intégré. Doctorat de l'Université Paul Sabatier, Toulouse III, 1990.

[Carpenter & Grossberg 91] Carpenter G., Grossberg S., Pattern recognition by Self-Organizing Neural Networks. Cambridge, Massachusetts, MIT Press 1991.

[Davalo 89] Davalo E., Naïm P., Des réseaux de neurones. Editions EYROLLES 1989.

[Frasson 92] Frasson C., Ramazani D., Prédiction du niveau d'acquisition des connaissances dans la modélisation de l'étudiant. Springer-Verlag, ITS'92, Montréal, Canada, juin 1992.

[Frasson 95] Frasson C., Mengelle T., Aïmeur E., Gouardères G., An Actor-based Architecture for Intelligent Tutoring Systems. ITS'96, Montréal, Canada, juin 1996 (*à paraître*).

[Fukunaga 90] Fukunaga K., Introduction to Statistical Pattern Recognition. New York, Academic Press 1990.

[Grossberg 82] Grossberg S., Competitive learning from interactive activation to adaptative resonance. Cognitive Science, 11, pp 23-63, 1982

[Hebb 49] Hebb D.O., The organization behaviour. J.Wiley & Sons, New York 1949.

[Hopfield 82] Hopfield J., Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proceedings of the National Academy of Science, USA, 79, pp 2554-2558, 1982.

[Haton 92] Haton M.C., l'ordinateur "pédagogue". La Recherche, vol 3, pp 1014-1022, sept.1992.

[Haton 91] Haton J-P, Bouzid N., Charpillet F., Le raisonnement en intelligence artificielle, Modèles, techniques et architectures pour les systèmes à base de connaissances. InterEditions, Paris, 1991.

[Hau Palé 94] Hau Palé S., Etude d'un EIAO pour l'apprentissage des techniques de bases du calcul mental. DEA de l'Université Paul Sabatier, Toulouse III, juin 1994.

[Honovar 94] Honavar V., Toward Learning Systems That Integrate Different Strategies and Representations. In Symbol Processors and Connectionist Networks for Artificial Intelligence and Cognitive Modelling: Steps towards Principled Integration, Honavar, V. & Uhr, L. (Ed), New York, Academic Press.

[Kohonen 89] Kohonen T., Self-organization and associative memory. 3rd edition Berlin, Springer-Verlag, 1989.

[Koza 92] Koza J., Genetic Programming. Boston, Massachusetts: MIT Press 1992.

[Le Cun 85] Le Cun Y., Modèles connexionnistes de l'apprentissage. Proceeding of Cognitiva 85, CESTA AFCET, juin 1985.

[Fefe 84] Lefèvre J.M., "Guide pratique de l'EIAO", Eds Cedic Nathan, 1984.

[Marcel 95] Marcel S., Introduction aux Réseaux de Neurones : Fondements, Modèles, Apprentissages et Applications. Support de cours, Faculté des Sciences de Limoges 1995.

[McCulloch 43] Mc Culloch W.S., Pitts, A logical calculus of the ides immanent in nervous activity. Bulletin of Mathematical Biophysics, 5, 115-133, 1943.

[Minsky 69] Minsky M., Papert S., Perceptrons. MIT Press, Cambridge MA, 1969.

[Mengelle 95] Mengelle T., Etude d'une architecture d'environnements d'apprentissages basés sur le concept de préceptorat avisé. Doctorat de l'Université Paul Sabatier, Toulouse III, 1995.

[Nicaud 88] Nicaud J.F, Vivet M., Les tuteurs intelligents: réalisations et tendances de recherches. Numéro spécial TSI "Application de l'informatique à la formation". Dunod-Afcet, vol 7/1, pp 21-45, 1988.

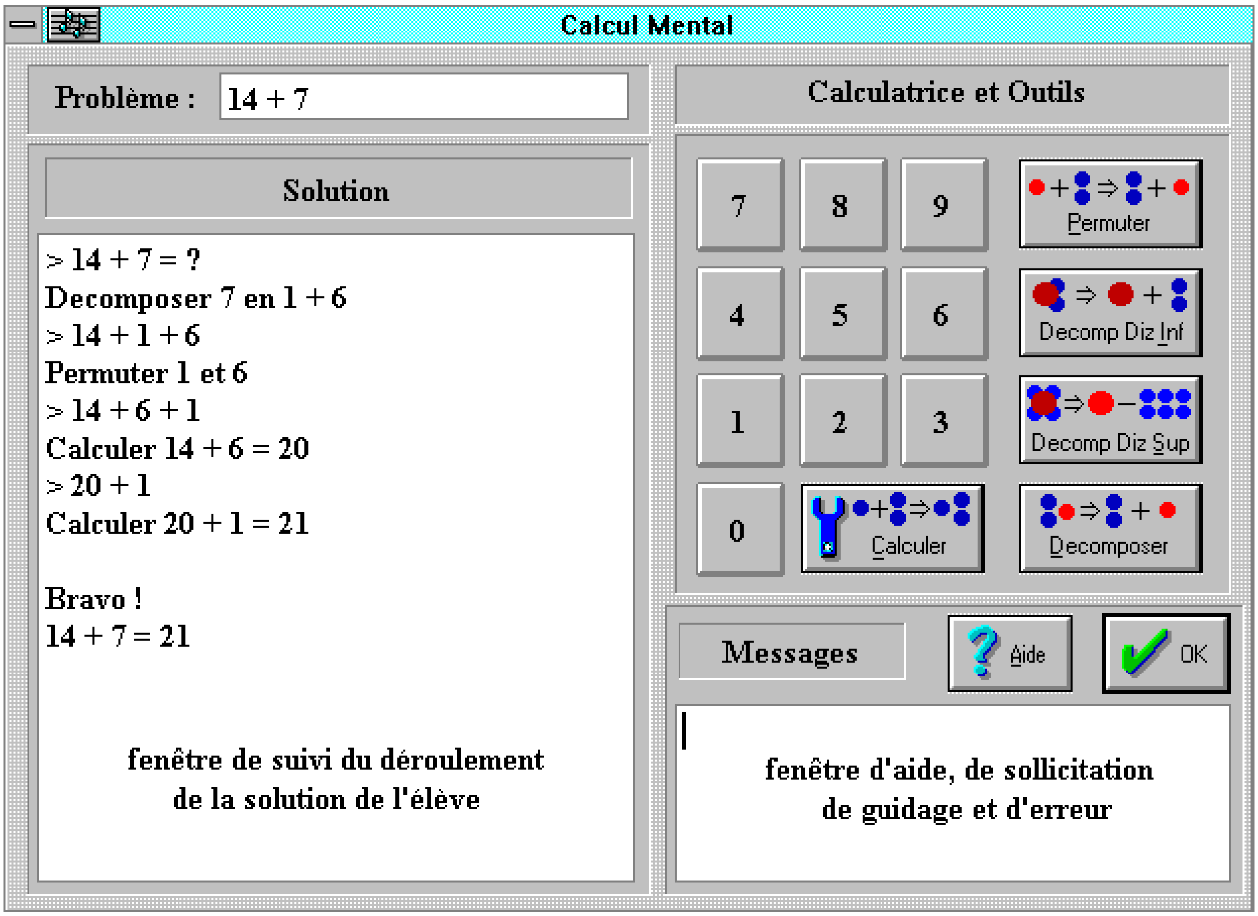
[Rosenblatt 57] Rosenblatt F., The perceptron: a perceiving and recognizing automation. Project Para, Cornell Aeronautical Lab. Report 85-460-1, 1957.

[Renders 95] Renders J.M., Algorithmes génétiques et réseaux de neurones. Hermès, Paris 1995.

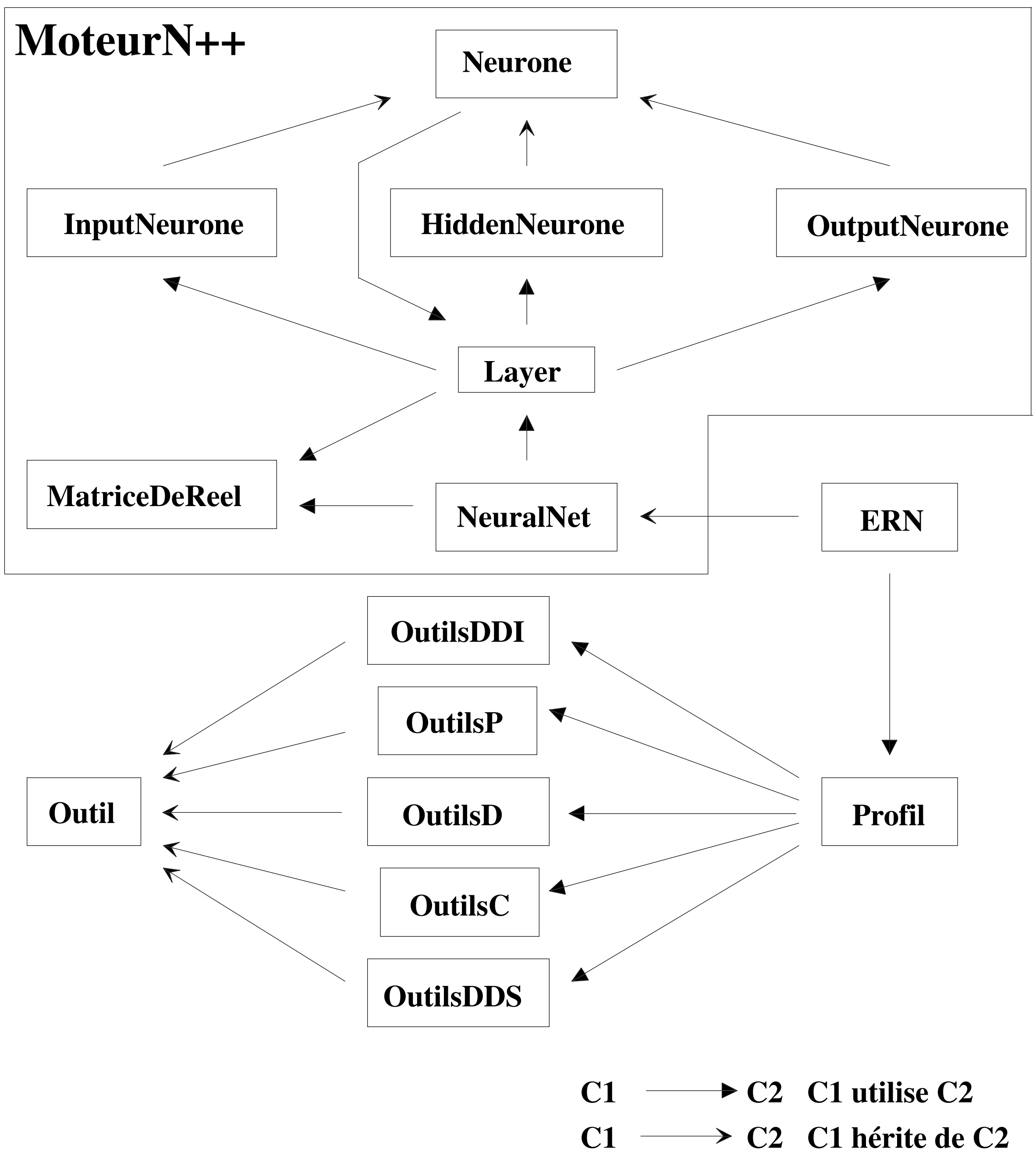
[Widrow 60] Widrow, Hoff M.E., Adaptative switching circuits. Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4, 96-104, 1960.

VII - Annexes

1. Interface d'acquisition de la solution de l'élève



2. Modèle conceptuel Objet



3. Spécification du moteur N++

```
#pragma COPYRIGHT "MARCEL Sebastien"
#pragma COPYRIGHT_DATE "1995-1996"

// Copyright MARCEL Sebastien 1995-1996
// Specification de l'objet MatriceDeReel : MatriceDeReel.h

#ifndef MATRICE_DE_REEL

#define MATRICE_DE_REEL

class MatriceDeReel
{
public:
	// Constructeur 1
	MatriceDeReel(ulong, ulong, double =0.0);
		// Taille maximale de la matrice
		// Valeur de l'initialisation

	// Constructeur 2
	MatriceDeReel(FILE *);
		// Fichier

	// Destructeur
	~MatriceDeReel();

	// Modificateur
	Booleen Set(ulong, ulong, double);
		// indices
		// nouvelle valeur

	// Accesseur
	Booleen Get(ulong, ulong, double &);
		// indices
		// valeur recuperee

	// Recuperation de la taille de la matrice
	ulong GetNbrLignes() { return i_max; };
	ulong GetNbrColonnes() { return j_max; };

	// Sauvegarde de la matrice
	Booleen Save(FILE *);

	Booleen Existe();

private:
};

#endif
```

```
#pragma COPYRIGHT "MARCEL Sebastien"
#pragma COPYRIGHT_DATE "1995-1996"

// Copyright MARCEL Sebastien 1995-1996
// Specification de l'objet NeuralNet : NeuralNet.h

#ifndef NEURALNET

#define NEURALNET

struct SLayer;

class NeuralNet
{
public:
	// Constructeur 1
	NeuralNet(char *, uint=1, uint=1, uint=1);
		// Nom du fichier de sauvegarde
		// Nombre de couches d'entrees
		// Nombre de couches cachees
		// Nombre de couches de sorties

	// Destructeur
	~NeuralNet();

	// Fixe les parametres d'une couche : ETAPE 1
	Booleen SetLayer(LayerNeuronType, uint, SDefLayer);
		// Type de la couche
		// Numero de la couche
		// Parametres de la couche

	// Fixe les connexions inter-couches : ETAPE 2
	Booleen SetInterLayerConnexions(LayerNeuronType, uint, LayerNeuronType, uint, MatriceDeReel *);
		// Type de la couche In
		// Numero de la couche In
		// Type de la couche Out
		// Numero de la couche Out
		// Matrice des connexions inter-couche In-Out

	// Propagation
	Booleen Propagation(MatriceDeReel &, MatriceDeReel &, uint=0, Booleen=FAUX);
		// Matrice [i,j] d'activation des neurones j des couches i d'entrees
		// Matrice [i,j] des etats des neurones j des couches i de sorties
		// Nbr de cycles pour la propagation intra-couche
		// si VRAI alors Auto-Apprentissage sinon rien

	// Retro-propagation
	Booleen RetroPropagation(MatriceDeReel &, MatriceDeReel &, Booleen=FAUX);
		// Matrice [i,j] des etats des neurones j desires aux couches i de sortie
		// Matrice [i,j] des etats des neurones j retropropages aux couches i d'entree
		// Effectuer le retro-seuillage
```

```
	// Calculer le taux de rappel
	double TauxRappel(MatriceDeReel, MatriceDeReel);
		// Sortie obtenue
		// Sortie desiree

	// Renommer le reseau de neurones
	void Rename(char *);
		// Nouveau nom

	// Sauvegarde le reseau
	Booleen Save();

	// Charge le reseau
	Booleen Load();

	// Initialise le reseau
	Booleen InitNet();

private:
	// Etat du reseau
	Booleen NetState;

	struct SLayer
	{
		Layer *AdresseCouche;
	};

	// Recuperer l'adresse d'une couche et sa taille
	SLayer* SelectLayer(LayerNeuronType, uint&);
		// Type de couche
		// Nombre de couches de ce type

	// Nombre de couches
	uint NbrInputLayer, NbrHiddenLayer, NbrOutputLayer;

	// Vecteur des adresses des couches
	SLayer *InputLayer, *HiddenLayer, *OutputLayer;

	// Nombre d'inter-connexions
	uint NbrInterConnexions;
	struct SListMatriceDeConnexions
	{
		Layer *LayerIn, *LayerOut;
		MatriceDeReel *AdresseMatrice;
		SListMatriceDeConnexions *Suivant;
	};

	// Liste des matrices de connexions
	SListMatriceDeConnexions *ListeMatriceDeConnexions;

	// Nom du fichier
	char *NomRN;
};
#endif
```

```
#pragma COPYRIGHT "MARCEL Sebastien"
#pragma COPYRIGHT_DATE "1995-1996"

// Copyright MARCEL Sebastien 1995-1996
// Specification de l'objet Layer : Layer.h

#ifndef LAYER

#define LAYER

class Layer
{
	friend class NeuralNet;
	friend class Neurone;
	friend class InputNeurone;
	friend class HiddenNeurone;
	friend class OutputNeurone;
public:
	// Constructeur
	Layer(LayerNeuronType, uint, MatriceDeReel *, uint, uint, uint);
		// Type de la couche
		// Numero de la couche
		// Matrice des intra-connexions
		// Nombre de neurones de la couche
		// Nombre de couches In
		// Nombre de couches Out

	// Destructeur
	~Layer();

	// Fixer les parametres
	Booleen SetParam(double, double, double, double);
		// Seuils
		// Vitesse d'apprentissage
		// Vitesse de propagation

	// Obtenir le nombre de neurones de la couche
	uint GetNbrNeurone();

	// Fixer les inter-connexions
	Booleen SetInterConnexions(LayerNeuronType, Layer *, MatriceDeReel *);
		// Type d'interconnexion
		// Adresse de la couche a interconnecter
		// Matrice des inter-connexions


	// Recupere le nombre d'input d'un neurone
	uint GetNbrInputNeurone(ulong);
		// Identificateur du neurone

	// Recupere le nombre d'output d'un neurone
	uint GetNbrOutputNeurone(ulong);
		// Identificateur du neurone
```

```
	// Afficher la couche
	void Afficher();

	// Sauvegarder les caracteristiques de la couche
	Booleen Save(FILE *);

	// Initialisation des neurones de la couche
	Booleen InitLayer();

private:
	// Numero de la couche
	uint IdLayer;

	// Etat de la couche
	Booleen LayerState;

	// Caracteristique de la couche
	SDefLayer sLay;

	// Type de la couche
	LayerNeuronType LayerType;

	struct SInterLayer
	{
		Layer *LayerConnect;
		MatriceDeReel *InterConnexions;
	};

	// Nombre de couches In et de couches Out
	uint NumLayerIn, NumLayerOut;

	// Vecteur des couches d'entrees et de sorties
	SInterLayer *InterInputConnexions, *InterOutputConnexions;

	struct SNeurone { Neurone *AdresseNeurone; };

	// Vecteur des neurones de la couche
	SNeurone *NeuroneLayer;
};

#endif
```

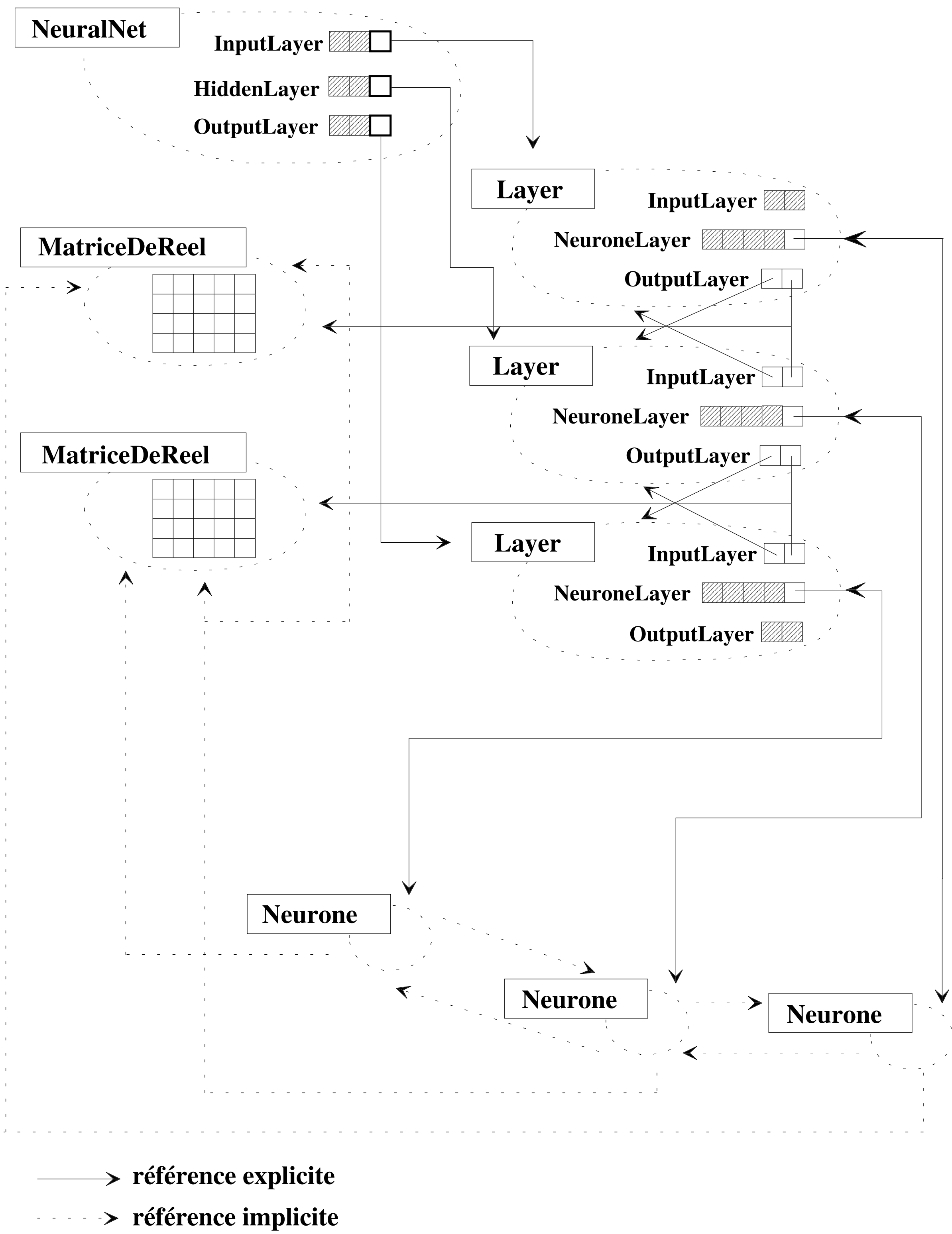
```
#pragma COPYRIGHT "MARCEL Sebastien"
#pragma COPYRIGHT_DATE "1995-1996"

// Copyright MARCEL Sebastien 1995-1996
// Specification de l'objet Neurone : Neurone.h

#ifndef NEURONE

#define NEURONE

//#define WAIT
#define RETRO_LEARNING

class Neurone
{
	friend InputNeurone;
	friend HiddenNeurone;
	friend OutputNeurone;
	friend NeuralNet;
public:
	// Constructeur
	Neurone(LayerNeuronType, ulong, double, double, double, double, Layer *);
		// Type du neurone
		// Identificateur
		// Seuils
		// Vitesse d'apprentissage
		// Vitesse de propagation
		// Adresse de la couche

	// Destructeur
	~Neurone();

	// Reception en entree de l'activation d'un neurone
	virtual Booleen ReceptionInput(double, double, Booleen=FAUX, uint=0) { return FAUX; };
		// Activation recue
		// Poids de la connexion

	// Reception d'une erreur retro-propagee
	virtual Booleen ReceptionErreur(double, double, ulong, Layer *, Booleen) { return FAUX; };
		// Erreur recue
		// Poids de la connexion
		// Identificateur du neurone qui renvoie l'erreur
		// Adresse de la couche du neurone qui envoie l'erreur
		// Effectuer le retro-seuillage

	// Activation des neurones en sortie
	Booleen ActiverOutput(uint);

	// Retro-propager l'erreur
	Booleen Retropropager(Booleen);

	// Afficher les composantes du neurone
	void Afficher();
```

```
    // Election du neurone
    void SetFirst() { bIsFirst=VRAI; };

    // Initialisation de l'etat du neurone
    void Init();

protected:
    // Nombre d'activations attendues
    uint GetNbrInput();

    // Nombre de neurones a activer
    uint GetNbrOutput();

    // Apprentissage par la regle du delta
    double DeltaRule(double, double, double);
        // Erreur retro-propagee
        // Poids d'une connexion

    // Fonction d'entree
    double h_input(double, double);
        // Activation
        // Poids

    // Fonction d'activation
    double f_activate(double);

    // Fonction de sortie
    double g_output(double);

    // Fonction derivee de la fonction f
    double f_derivee(double);

    // Fonction derivee de la fonction g
    double g_derivee(double);

    // Identificateur
    ulong Identificateur;

    // Etat d'election
    Booleen bIsFirst;

    // Seuil d'activation
    double BetaSeuil;

    // Seuil de retro-activation
    double BetaRetro;

    // Vitesse d'apprentissage et de propagation
    double EtaSpeed, TetaSpeed;

    // Valeur d'entree
    double I;
    double OldI;
```

```
        // Valeur d'activation
        double A;

        // Valeur de sortie
        double O;
        double OldO;

        // Valeur de l'erreur locale
        double E;

        // Nombre d'activations recues
        uint NbrActivationsRecues;

        // Nombre d'erreurs recues
        uint NbrErreursRecues;

        // Adresse de la couche du neurone
        Layer *HostLayer;

        // Type du neurone
        LayerNeuronType NeuroneType;
};

#endif


#pragma COPYRIGHT "MARCEL Sebastien"
#pragma COPYRIGHT_DATE "1995-1996"

// Copyright MARCEL Sebastien 1995-1996
// Specification de l'objet InputNeurone : InputNeurone.h

#ifndef INPUT_NEURONE

#define INPUT_NEURONE

class InputNeurone : protected Neurone
{
public:
        // Constructeur
        InputNeurone(ulong, double, double, double, double, Layer *);
                // Identificateur
                // Seuils
                // Vitesse d'apprentissage
                // Vitesse de propagation
                // Adresse de la couche

private:
};

#endif
```

```
#pragma COPYRIGHT "MARCEL Sebastien"
#pragma COPYRIGHT_DATE "1995-1996"

// Copyright MARCEL Sebastien 1995-1996
// Specification de l'objet HiddenNeurone : HiddenNeurone.h

#ifndef HIDDEN_NEURONE

#define HIDDEN_NEURONE

class HiddenNeurone : protected Neurone
{
public:
    // Constructeur
    HiddenNeurone(ulong, double, double, double, double, Layer *);
        // Identificateur
        // Seuils
        // Vitesse d'apprentissage
        // Vitesse de propagation
        // Adresse de la couche

private:
};

#endif

#pragma COPYRIGHT "MARCEL Sebastien"
#pragma COPYRIGHT_DATE "1995-1996"

// Copyright MARCEL Sebastien 1995-1996
// Specification de l'objet OutputNeurone : OutputNeurone.h

#ifndef OUTPUT_NEURONE

#define OUTPUT_NEURONE

class OutputNeurone : protected Neurone
{
public:
    // Constructeur
    OutputNeurone(ulong, double, double, double, double, Layer *);
        // Identificateur
        // Seuils
        // Vitesse d'apprentissage
        // Vitesse de propagation
        // Adresse de la couche

private:
};

#endif
```

4. Architecture interne du moteur N++



5.1. Mesure du taux de rappel

5.2. Mesure de l'erreur

5.3. Mesure des poids des connexions

5.4. Comparaison des formules d'activation

5.5. Comparaison des formules d'apprentissage

5.6. Mesure du seuil d'activation

5.7. Rétropropagation totale sans le seuil

5.8. Rétropropagation totale avec le seuil

5.9. Fusion sans le seuil

5.10. Fusion avec le seuil

5.11. Fusion avec seuil et verrouillage

5.12. Fusions sur 2 exemples

5.13. Rétropropagation totale avec seuil et verrouillage

Profil de l'élève $P=\left\langle N, S_{DDI}, S_{DDS}, S_D, S_P, S_C \right\rangle$

N : niveau de progression de l'élève

S_o : Score pour l'outils o, avec $S_o=\left\langle N_e, N_c, N_m, \Delta_r \right\rangle$

où N_e: Nombre d'essais d'utilisation de l'outils

N_c: Nombre d'échecs d'utilisation de l'outils

N_m: Nombre de mauvais usages d'utilisation de l'outils

Δ_r: Temps moyen de réponse d'utilisation de l'outils

Couche de l'expert C_x **/ Couche de l'élève virtuel** C_v**/ Couche de l'élève réel** C_e

$$C_x A=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad C_v A = C_e A = \begin{bmatrix} S_1 & 0 & 0 & 0 & 0 \\ 0 & S_2 & 0 & 0 & 0 \\ 0 & 0 & S_3 & 0 & 0 \\ 0 & 0 & 0 & S_4 & 0 \\ 0 & 0 & 0 & 0 & S_5 \end{bmatrix}$$

avec $S_i = S_{ii} = \left[\left(\theta_{stat} * STAT + \theta_{tps} * TPS \right) / \left(\theta_{stat} + \theta_{tps} \right) \right] * [N/7]$

$$STAT = 1 - \left[\left(o(i).N_c + o(i).N_m \right) / \left(2 * o(i).N_e \right) \right] \qquad TPS = 1 - o(i).\Delta_r / \Delta_{\max}$$

où o(i) outils de rang i dans E_o et $i \in [1 \ldots 5]$

$\theta_{stat}, \theta_{tps}$: coefficients de priorité des statistiques et du temps

Couche d'association A

$$AA_{ij} = \begin{bmatrix} 0 & AA_{12} & . & . & . \\ AA_{21} & 0 & . & . & . \\ . & . & 0 & . & . \\ . & . & . & 0 & AA_{45} \\ . & . & . & AA_{54} & 0 \end{bmatrix}$$

avec $AA_{ij} = AA_{ji},\ AA_{ii} = 0$

$AA_{ij} = [\alpha.Dx + \beta.Dv + \delta.De] / (\alpha + \beta + \delta)$

$$Dx = \left| C_x A_{ii} - C_x A_{jj} \right| \; Dv = \left| C_v A_{ii} - C_v A_{jj} \right| \; De = \left| C_e A_{ii} - C_e A_{jj} \right|$$

α : importance de la solution de l'expert fixée à 1

β : importance de la solution virtuelle

δ : importance de la solution de l'élève

Formalisation d'un réseau de neurones à couches : triplet <I, H, O>

avec I = $<NbI, \left\{...,C_q^I,...\right\}>$, H = $<NbH, \left\{...,C_q^H,...\right\}>$, O = $<NbO, \left\{...,C_q^O,...\right\}>$

NbI, NbH, NbO: nombre de couches d'entrées, cachées et de sorties

C_q^X : couche de type X d'indice q avec $q \in \left[1...NbX\right] / X \in \left[I,H,O\right]$

$$C^k = < N, \beta, \beta', \theta, \eta, \varepsilon, \varphi, \gamma, NbIn, \left\{..., In_i^k, ...\right\}, NbOut, \left\{..., Out_j^k, ...\right\}, \Pi_{[N,N]} >$$

C^k: couche numéro k avec $k \in \left[1...(NbI + NbH + NbO)\right]$

N : nombre de neurones de la couche

β : seuil d'activation des neurones

β' : seuil de rétro-activation des neurones

θ : vitesse de propagation de l'information dans les neurones

η : vitesse d'apprentissage

ε : seuil de désapprentissage

φ : seuil de disparition des connexions

γ : seuil de rétropropagation

NbIn, NbOut : nombre de couches en entrée et en sortie

$In_i^k = < C^{k'}, \Pi_{\left[N, C^{k'}.N\right]} >$: i-ème couche d'entrée avec k>k'

$Out_i^k = < C^{k''}, \Pi_{\left[N, C^{k''}.N\right]} >$: j-ème couche de sortie avec k<k''

$\Pi_{[n,m]}$: matrice n*m de connexions

Fonctions d'entrée (h), d'activation (f) et de sortie (g)

$$h(...,e_j,...) = \sum_j w_{ij} * e_j$$

$$f_1(x) = 1/(1+e^{-\theta x}), f_2(x) = f_1(x) - 0.5, f_3(x) = (1+e^{+\theta x})/(1+e^{-\theta x})$$

$$f_1'(x) = f_2'(x) = \theta.e^{-\theta x}/(1+e^{-\theta x})^2 \;,\; f_3'(x) = (e^{\theta x} - e^{-\theta x})/(1+e^{-\theta x})^2$$

$$g(x) = \begin{Bmatrix} x \; si \; x \geq \beta \\ 0 \; si \; x < \beta \end{Bmatrix} \quad g'(x) = \begin{Bmatrix} x \; si \; x \geq \beta' \\ 0 \; si \; x < \beta' \end{Bmatrix}$$