المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Ministère de l'Enseignement Superieur et de la

Recherche Scientifique

Ecole Nationale Polytechnique

DER Génie Electrique et Informatique

Thèse de Magister

en Electronique
Option : Acquisition et Traitement de l'Information.

présentée par :
LAIDI KAMEL Ingénieur d'Etat en Electronique de l'ENP

Thème

Implémentation Orientée Hardware des Réseaux de Neurones Artificiels : Application à la Navigation d'un Robot Mobile Autonome

Soutenue devant le jury :

M.MEHENNI	Maitre de Conférences	à l'ENP	Président
A.FARAH	Professeur	à l'ENP	Rapporteur
F.BOUDJEMA	Maitre de Conférences	à l'ENP	Examinateur
C.LARBES	PhD, Enseignant	à l'ENP	Examinateur
R.SADOUN	Maitre de Conférences	à l'ENP	Examinateur

Septembre1998

1 0, Avenue Hassan Badi, El Harrach, Alger (16200), Tel : (02) 52. 53. 01/03, Fax : (02) 52. 29. 73.

Avant-propos

Ce travail est effectué au sein du laboratoire des Techniques Digitales et Systèmes (TDS) au Département d'Electronique de l'Ecole Nationale Polytechnique sous la direction du Professeur A.FARAH, à qui je présente l'expression de ma profonde gratitude pour tous ses conseils et ses encouragements .

Je remercie vivement Monsieur M.Mehenni, Maitre de Conférences à L'ENP, de l'honneur qu'il ma fait en acceptant de juger ce travail et de présider le jury de thèse.

Je suis très reconnaissant également aux Messieurs F.Boudjema, Maitre de conférences, C.Larbès PhD, enseignant et R.Sadoun Chargé de Cours pour leur participation au jury de cette thèse, à qui nous exprimons nos sincères remerciements

Je tiens à remercier tous les enseignants qui ont contribué à notre formation en graduation et en post-graduation.

Toute notre reconnaissance va au personnel du centre de documentation ainsi que du centre de calcul

Enfin, je remercie tous mes amis pour leurs soutiens de prés ou de loin.

DEDICACES

A ma chère mère,
A mon cher père,
A tous mes frères et sœurs.
A toute ma famille,
A tous mes amis.

Je dédie ce travail.

Kamel

SOMMAIRE.

INTRODUCTION GENERALE

La neurobiologie a découvert que la puissance de traitement dans les couches du cerveau humain est due au grand nombre d'unités de traitement identiques(neurones), liés entre eux par des longueurs variables en terme de réseau de poids synaptiques. Cette puissance de traitement neuronal est la cause de l'apparition d'un large espace de recherche dans le domaine des réseaux de neurones artificiels (*RNAs*) incluant la modélisation neurale, algorithmes, architectures, implémentations et applications.

La réponse et les caractéristiques des modèles neuronaux sont primitivement simulés sur des ordinateurs vectoriels, des stations de travail, des coprocesseurs spéciaux ou des réseaux de transputers. L'inconvénient fondamental de chaque simulateur est que le parallélisme spatio-temporel dans le traitement d'information qui est inhérent aux réseaux de neurones est entièrement ou partiellement perdu. Le temps de calcul du réseau simulé spécialement pour des larges associations de neurones croit avec l'ordre de dimensions à un point où l'acquisition rapide du savoir-faire neural est gênée ou devient impossible [1].

Une réduction appréciable dans le temps de calcul et ainsi que la manipulation des grandes tâches ou celles qui doivent être exécutées en temps réel deviennent possibles avec une implémentation hardware dans un circuit spécialisé *ASIP(Application Specific Instruction Processor)*. A part le temps de calcul le plus court possible, l'*ASIP* neural offre un volume structurel très petit, qui peut être implémenté pour la même tâche. Cet aspect est spécialement important lorsque le processeur neural est à incorporer dans des terminaux pour une communication homme-machine ou robots mobiles.

Mais, si l'implémentation "hardware" offre ces avantages, elle nécessite des méthodes et des techniques modernes pour contourner les problèmes de conception et de réalisation traditionnelles, surtout en ce qui concerne la justesse, la fiabilité et la maintenabilité de la conception ; ce qui a poussé les développeurs du "hardware" à réfléchir aux nouvelles méthodes et techniques flexibles et efficaces permettant de développer des circuits à très large échelle d'intégration.

L'évolution rapide de la technologie et spécialement les outils de C.A.O. ont apporté de nouvelles techniques de développement du matériel tout en utilisant la modélisation et la simulation logicielle avant de passer à son implémentation matérielle.

Dans ce contexte, en exploitant la fertilisation réciproque et la similarité existante entre le matériel et le logiciel, la technique de la modélisation orientée objet constitue un nouvel outil dans le domaine, vue qu'elle permet une modélisation software des composants hardware tout en utilisant les propriétés inhérentes de : modularité des objets, encapsulation, héritage, polymorphisme et la notion de types de données définis par l'utilisateur (ou classes)[2][3].

Dans le cadre de cette thèse de magister, nous nous sommes intéressés à l'implémentation digitale d'un *RNA* pour la navigation d'un robot mobile autonome en utilisant :

- Les réseaux systoliques pour l'architecture.
- La technique de programmation orientée objet (T.P.O.O) pour la simulation de la conception et la validation des résultats.

Le travail se compose des chapitres suivants :

-Le premier chapitre est consacré aux fondements théoriques des réseaux de neurones, leurs différents modèles ainsi que les algorithmes d'apprentissage associés à chaque modèle.

-Dans le deuxième chapitre, nous considérons deux approches possibles d'implémentation hardware digitale des réseaux de neurones artificiels: l'approche par commutateurs programmables et l'approche par réseaux systoliques. Cette dernière sera en définitive retenue, au troisième chapitre, pour notre application.

-Le troisième chapitre présentera, le développement sous forme d'architecture systolique d'un réseau de neurones utilisant l'algorithme de la retro-propagation du gradient (*RPG*) appliqué à la navigation d'un robot mobile autonome *(RMA)* dans un environnement partiellement structuré.

-Le quatrième chapitre représente modélisation orientée objet de l'architecture systolique développée dans le chapitre précédent, en utilisant le C++ comme code de simulation.

-Le cinquième chapitre est consacré aux résultats obtenus par simulation graphique de l'environnement d'application ainsi que leurs interprétations.

Enfin, une conclusion générale résume les performances de cette technique et les objectifs atteints.

Chapitre 1.

RESEAUX DE NEURONES.

1.1. INTRODUCTION

Les modèles connexionistes neuronaux sont des systèmes qui tentent de simuler les mécanismes de traitement de l'information ayant lieu dans le cerveau humain.

Une importante caractéristique des réseaux de neurones est leur capacité à apprendre. L'apprentissage va permettre au réseau de modifier sa structure interne (poids synaptiques) pour s'adapter à son environnement. En effet, nous n'avons pas besoin d'une formulation rigoureuse d'un problème donné pour le résoudre, tout ce dont nous avons besoin est une collection d'exemples représentative de la fonction désirée. Le réseau s'adapte ensuite pour reproduire les sorties souhaitées quand un exemple d'entrée est présenté.

Une autre caractéristique des réseaux de neurones est leur capacité à traiter des données bruitées ou incomplètes par rapport aux méthodes algorithmiques traditionnelles. Un réseau de neurones, par sa structure composée de neurones ou cellules interconnectées entre elles par des liens synaptiques, est capable d'apprendre en modifiant ces liens. En plus, il est doté de parallélisme massif dans son fonctionnement [4].

Il existe aujourd'hui une panoplie de réseaux qui se distinguent par leur architecture et/ou par la loi d'apprentissage qui régie le réseau.

Dans ce chapitre nous présentons une synthèse des aspects descriptifs, algorithmiques et caractéristiques des modèles neuronaux les plus connus.

1.2. NEURONE BIOLOGIQUE

Le cerveau humain est composé de milliards de cellules nerveuses appelées neurones et c'est l'activité électrochimique de ces cellules qui reflète l'accomplissement des tâches par le cerveau. La description la plus classique des neurones est la suivante [4][5]:

"Chaque neurone possède une arborescence de fibres entrantes appelées dendrites, un élément de traitement appelé corps cellulaire et une arborescence de fibres sortantes appelée axone(voir Fig.1.1). La connexion de l'axone d'un neurone aux dendrites ou au corps cellulaire d'un autre neurone est appelée synapse".

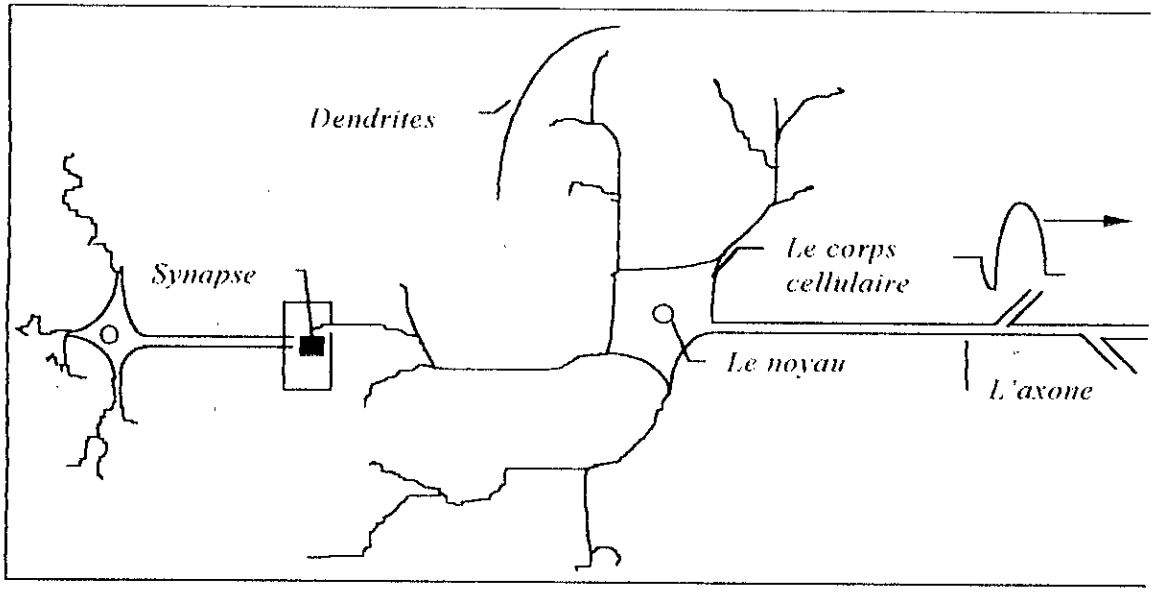


Fig.1.1. Schéma simplifié d'un neurone biologique.

L'information circule sous forme de trains d'impulsions électriques. Le neurone, qui reçoit des informations depuis ses dendrites, effectue approximativement l'intégration (la somme) de ces trains d'impulsions. Si le potentiel somatique dépasse un certain seuil, il émet un signal bref que l'on appelle signal d'activation (voir Fig.1.2), ce dernier se propage sans amortissement le long de l'axone et des ramifications[6].

La sortie du neurone dépend d'une fonction de transfert. Les principales sont représentées sur la figure suivante [4][8].

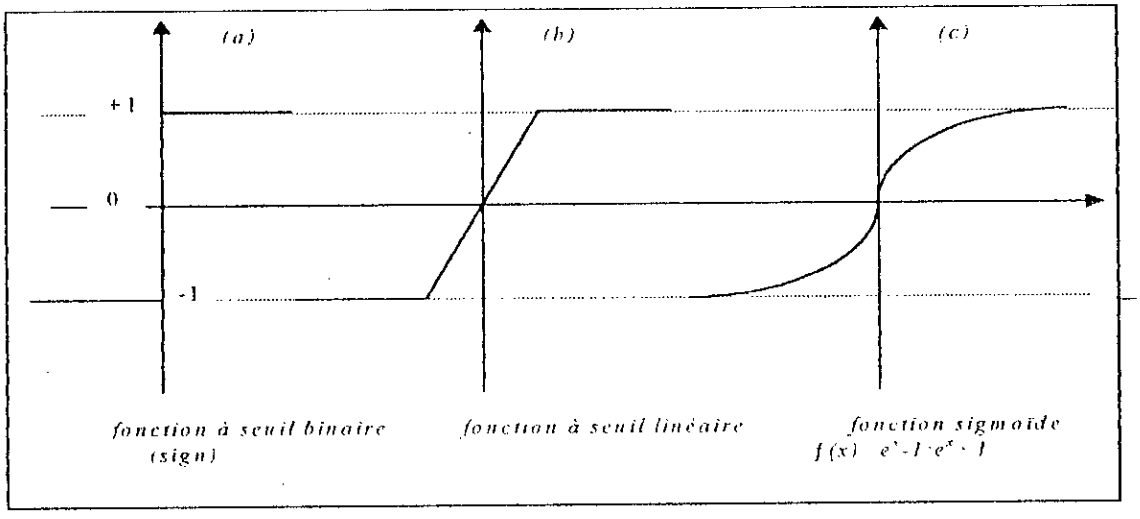


Fig.1.4. Les principales fonctions d'activation.

1.4. MODELISATION GENERALE D'UN RESEAU DE NEURONES

Un réseau de neurones est un ensemble de neurones formels (systèmes élémentaires) interconnectés et évoluant dans le temps par interactions réciproques. Pour décrire un tel réseau il nous faut donc décrire :

- le comportement de chaque neurone,
- l'interaction entre neurones.

1.4.1. DYNAMIQUE DES ETATS

La dynamique des états s'intéresse à l'évolution des états des différents neurones d'un réseau. Elle cherche l'existence d'états stables ou de cycles stables que ce soit pour des cellules particulières, pour des groupes de cellules ou pour le réseau tout entier[6].

Ce qui revient à étudier pour chaque réseau :

- la fonction d'activation des neurones,
- la structure et poids des connexions.

1.4.2. DYNAMIQUE DES CONNEXIONS

Le poids w_{ij} de la connexion entre les neurones *(i,j)* pondère le signal transmis, est peut être:

- excitateur (w_{ij} >0)
- inhibiteur (w_{ij} <0).

La dynamique des connexions ne s'intéresse qu'aux réseaux dont les poids des connexions sont évolutifs[6].

1.4.3. FONCTIONNEMENT

Une fois l'architecture et la dynamique du réseau sont choisies, le réseau va subir à son entrée les vecteurs de forme à apprendre (phase d'apprentissage). L'algorithme d'apprentissage détermine la façon d'ajuster les poids du réseau pour obtenir la sortie désirée pour un vecteur de forme donné. La phase suivante est appelée phase d'utilisation ou de rappel. Elle consiste à présenter au réseau des vecteurs autres que ceux qui ont contribués à son apprentissage (des vecteurs généralement bruités ou incomplets). Pendant cette phase le réseau va réagir selon les connaissances acquises durant la phase d'apprentissage.

1.5. TOPOLOGIES DES RESEAUX DE NEURONES

Plusieurs topologies ont été proposées. Pour les distinguer, nous pouvons les classer en deux grandes catégories, les réseaux non-récurrents et les réseaux récurrents[5].

1.5.1. RESEAUX NON-RECURRENTS

Ceux sont des réseaux où les unités (neurones) sont organisées en couches successives. Ils ne présentent donc pas de boucle de rétroaction (feed-back). Nous distinguons les modèles suivants.

1.5.1.1. LE PERCEPTRON

Le perceptron est le premier modèle opérationnel de réseaux de neurones. Il est basé sur le neurone de *Mc Culloch* et *Pitts* et décrit un algorithme capable de modifier les poids en fonction des valeurs qu'on souhaite lui faire apprendre. Le terme perceptron couvre maintenant une classe de modèles beaucoup plus grande que le perceptron élémentaire d'origine, proposé par *Rosemblatt* en 1957. Il est composé de trois types de cellules différentes[9].

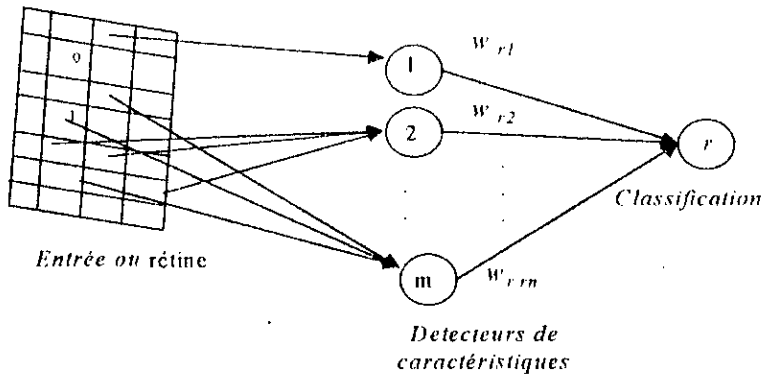


Fig.1.5. Schéma d'un perceptron élémentaire avec m nœuds détecteurs de caractéristiques et un nœud de réponse r.

• La première couche, appelée rétine, contient les cellules d'entrées. Chaque cellule se contente de recopier la valeur qu'elle reçoit de l'extérieur.

• La seconde couche est composée de m cellules dites associatives ou détecteurs de caractéristiques. Chaque cellule a des connexions entrantes pouvant parvenir de toutes ou d'une partie des cellules de la rétine avec connexion sortante reliée au nœud réponse.

• La dernière couche, composée d'un seul nœud r à espace d'états binaire ($s_i = 1$ *ou* -1) est caractérisée par un seuil θ_r et des forces de connexions indiquées par les poids w_{ri}. La décision du noeud-réponse r est prise suivant le seuillage de la somme pondérée des entrées.

$$U_r = \sum w_{ri} \cdot s_i \ . \qquad (1.1)$$

Le noeud r rend 1 si et seulement si $s_i \geq \theta_r$ et -1 sinon.

Les perceptrons élémentaires sont des machines à règle de décision linéaire, partageant l'espace des vecteurs en deux classes séparées, par un hyperplan de l'espace à m dimensions. Ils ne sont donc capables d'apprendre que les classifications linéairement séparables[6].

1.5.1.2. RESEAUX NON LINEAIRES

Appelés aussi Mémoires Associatives Linéaires, ce modèle simple a été introduit par *Anderson* en 1977 et *Kohonen* en 1986. Les neurones sont des éléments dont la caractéristique entrée/sortie est linéaire. Le réseau a une structure complètement interconnectée. Ainsi, les N

entrées du réseau sont reliées de manière indépendante aux P sorties, comme cela est montré sur la Figure.1.6 .

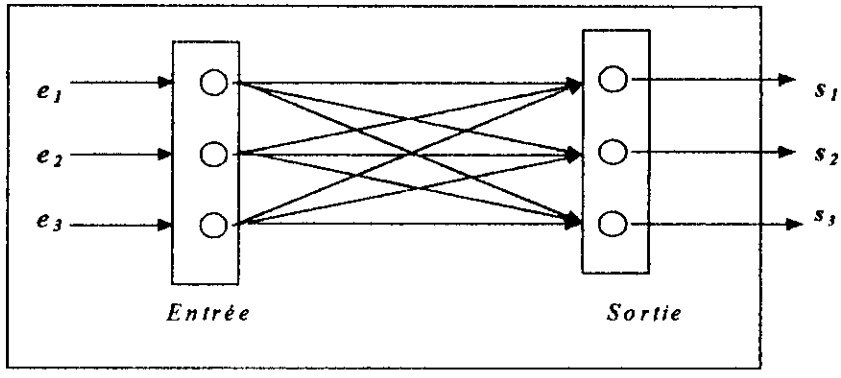


Fig.1.6. ***Réseau à deux couches et à circulation de l'information dirigée vers l'avant*** .[4]

Chaque entrée e_i est reliée à toutes les sorties S_j par les poids W_{ij}. Les entrées, les sorties et les poids ont des valeurs quelconques; à priori réelles.

1.5.1.3. ADALINE / MADALINE

Le terme Adaline (*ADAptive LINear Element*) ne désigne pas en fait un modèle de réseau, mais un type de cellules. Ce sont des cellules dont la fonction de transfert (de transition) est l'identité. Un Adaline est composé d'un seul neurone et d'une couche d'interconnexion de N poids (N entrées) et d'une sortie comme il est montré dans la Figure.1.7.

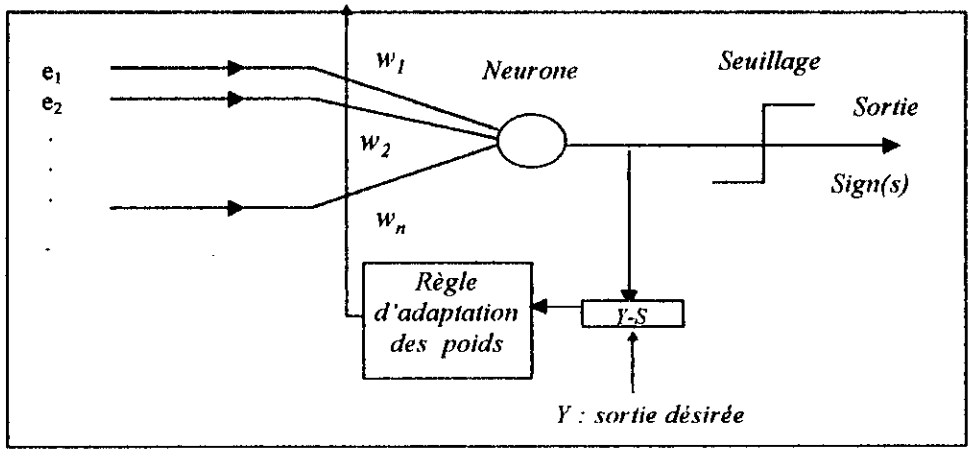


Fig.1.7. ***Schéma global d'une Adaline*** [4][5].

Le terme Madaline (Multiple Adaline) signifie l'association de plusieurs Adalines en réseau à une seule couche d'interconnexions.

Il a été rapidement apparu aux chercheurs que pour dépasser les limitations de la séparation linéaire, il était nécessaire d'ajouter des couches intermédiaires.

1.5.1.4. RESEAUX MULTICOUCHES A RETRO-PROPAGATION DU GRADIENT

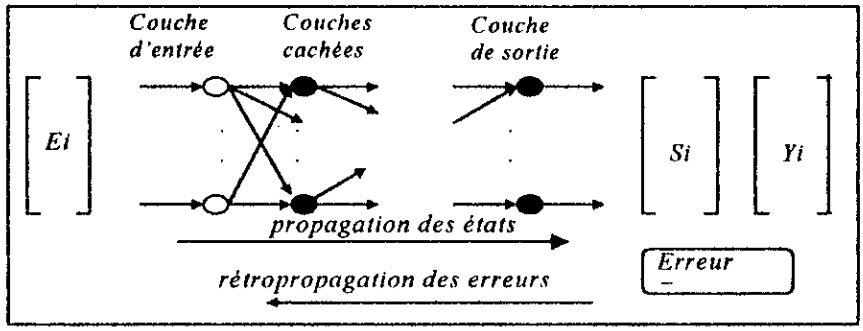


Fig.1.8. ***Réseau multicouches.***

Avec ce type de réseaux, la dynamique de mise à jour est implicite. L'état des unités est donné fonctionnellement par l'état des unités d'entrées.

Le principe de l'algorithme d'apprentissage, désigné couramment par "*back-propagation*", est que de même que nous sommes capables de propager un signal provenant de la couche d'entrée vers la couche de sortie, nous pouvons, en suivant le chemin inverse, rétropropager l'erreur commise en sortie vers les couches internes. Ainsi nous donnons lieu à une modification de chaque poids. Nous cherchons à minimiser l'erreur quadratique commise sur l'ensemble des exemples, considérée comme une fonction des poids, par une approximation d'une descente de gradient. Cette erreur est donnée par l'équation suivante[4][8]:

$$E_{rr}(W) = \sum_{i=1}^{m} (Y_i - S_i)^2 \qquad (1.2)$$

Où :

$Y_i = (y_1, y_2, ..., y_m)$ *est la sortie désirée,*

$S_i = (s_1, s_2, ..., s_m)$ *est la sortie obtenue ,*

$E_{rr}(W)$ *: l'erreur quadratique,*

W : *est la matrice des connexions .*

La règle de modification des poids à la présentation numéro k de l'exemple E est:

$$w_{ij}^{(k)} = w_{ij}^{(k-1)} - \eta(k).d_j.s_i \qquad (1.3)$$

Où : dj est calculé de proche en proche de la couche de sortie à la couche d 'entrée, d'où le nom de la rétropropagation :

$$d_j = \begin{cases} 2(Y_j - S_j)f'(U_j) & : \textit{pour les cellules de la couche de sortie,} \\ \sum_h d_h . w_{jh} . f'(U_j) & : \textit{pour les cellules des couches cachées.} \end{cases} \qquad (1.4)$$

Où : *h : est l'ensemble de neurones vers lequel le neurone j envoie un signal,*

f : est la fonction sigmoïde, f ' sa dérivée,

S_j *: la sortie du neurone j,*

U_j *: l'entrée du neurone j,* $U_j = \sum_i w_{ij} . s_i$.

$\eta(k)$: le pas du gradient à l'étape k.

Néanmoins, beaucoup de questions restent en suspens concernant cet algorithme notamment en ce qui concerne la méthode de conception d'un réseau (en terme de nombre de couches et de nombre de cellules par couche) pour résoudre un problème donné, la vitesse de convergence et la méthode qui permet de choisir l'ordre de présentation des exemples au réseau.

1.5.2. RESEAUX RECURRENTS

Les réseaux récurrents sont des réseaux dynamiques dont les graphes d'interconnexions contiennent des circuits et qui de ce fait ont un fonctionnement plus complexe que celui des réseaux non récurrents.

Parmi les modèles récurrents les plus connus nous pouvons citer:

1.5.2.1. MODELE DE *HOPFIELD*

Un réseau de *Hopfield* est composé de n cellules interconnectées les unes avec les autres.

Hopfield a montré qu'un tel réseau convergeait toujours vers un état stable si :

- la matrice des poids W était symétrique: $w_{ij} = w_{ji} \ \forall \ i \neq j$,
- les cellules étaient mises à jour selon une dynamique séquentielle aléatoire (toutes les cellules étant équiprobables).

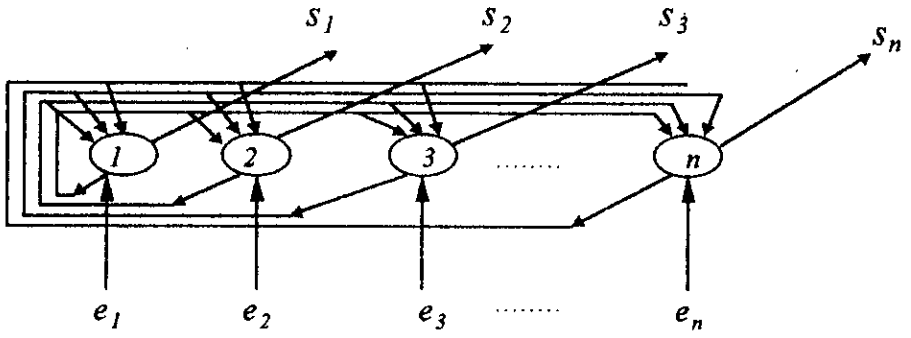


Fig.1.9. ***Structure d'un réseau de Hopfield.***

Ceci tient du fait que l'état du réseau à un instant t peut être caractérisé par une fonction réelle $H(t)$ appelée énergie. Cette fonction admet un minimum absolu et qui ne peut que décroître entre deux itérations du réseau. L'équation $H(t)$ est comme suite:

$$H(t) = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \cdot s_i(t) \cdot s_j(t) + 2\sum_{i=1}^{n} \theta_i \cdot s_i(t) \qquad (1.5)$$

θ_i : est le seuil du neurone i,

w_{ij} : est le poids symétrique entre le neurone i et le neurone j,

$S_i(t)$: l'état du neurone i, (-1 ou 1).

Un réseau de *Hopfield* va donc converger, en mode séquentiel aléatoire, quelque soit son état initial ($s_i(t=0)$) et les états stables qu'il pourra atteindre correspondront aux minimums de H. Tout le problème consiste donc à trouver une matrice W symétrique qui rendra la fonction H minimale pour les formes de l'échantillon d'apprentissage.

Pour ajouter un état stable au réseau, il suffit à priori de modifier W_{ij} par la règle de *Hebb*. Cependant ce réseau ne permet pas de mémoriser un trop grand nombre d'exemples soit 0.14 fois le nombre de neurones de ce dernier.[5]

En 1984, *Hopfield* a montré que les résultats obtenus sur les réseaux à activité binaire (La sortie d'un neurone est binaire) pouvaient être étendus aux réseaux à activité continue (La sortie d'un neurone est continue).

1.5.3. RESEAUX A AUTO-ORGANISATION ET COMPETITION

Un caractère commun à tous les réseaux que nous avons vus jusqu'à maintenant est l'aspect de collaboration des cellules du réseau afin de reconstituer un vecteur à la sortie. Chaque cellule aide à déterminer un fragment de la frontière de décision.

Contrairement; dans les réseaux à auto-organisation, chaque cellule présente un concept et entre en compétition avec toutes les autres cellules pour déterminer le concept le plus proche à celui présenté à l'entrée. Un des modèles présentant cette caractéristique est le réseau de *Kohonen*[4].

1.5.3.1.RESEAU DE KOHONEN

Les réseaux de *Kohonen*, connus sous le nom de cartes topologiques, sont utilisés pour réaliser les opérations de classification et de regroupement. Ce réseau est composé de deux couches (Voir Fig.1.10).

La première couche est composée de M cellules d'entrée, la seconde couche est appelée couche d'auto-organisation de N cellules.

Cette architecture tient compte à la fois des données extérieures qui proviennent des cellules d'entrée, et des connexions internes au réseau.

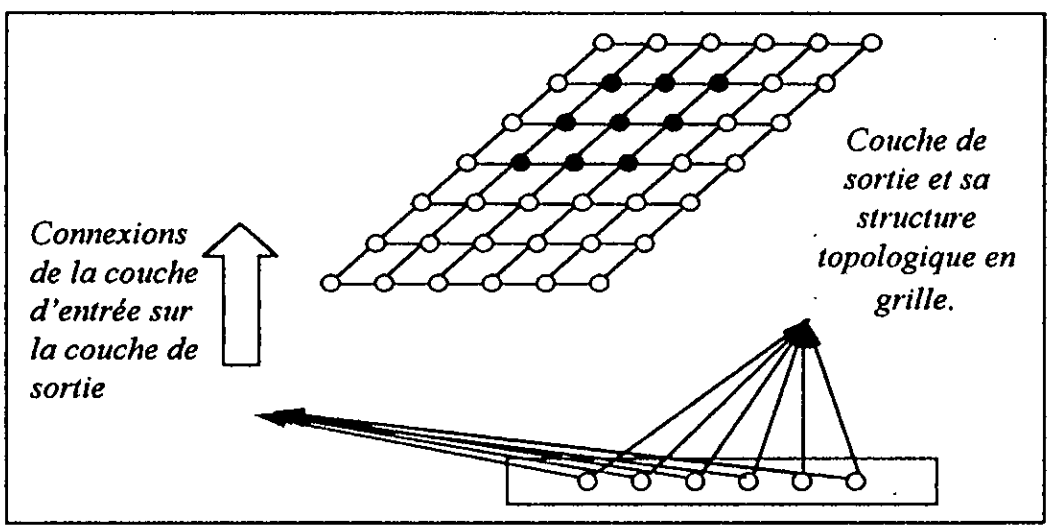


Fig.1.10. Architecture d'une carte topologique.

Les poids synaptiques reliants les neurones de la couche d'entrée à un neurone de la couche de sortie définissent les coordonnées dans l'espace des entrées du prototype représenté par le neurone prototype (de la couche de sortie) dont il se rapproche le plus.

Les unités de sorties sont reliées par des liens pondérés par w_{ij} (valeur du lien entre les neurones i et j du réseau).

Chaque neurone de la couche de sortie reçoit les N composantes du vecteur d'entrée, puis propage le signal vers les autres cellules du réseau selon un mécanisme d'interaction latérale qui dépend de la distance entre les neurones concernés; cette dépendance est représentée par la fonction dite du "chapeau mexicain " [5].

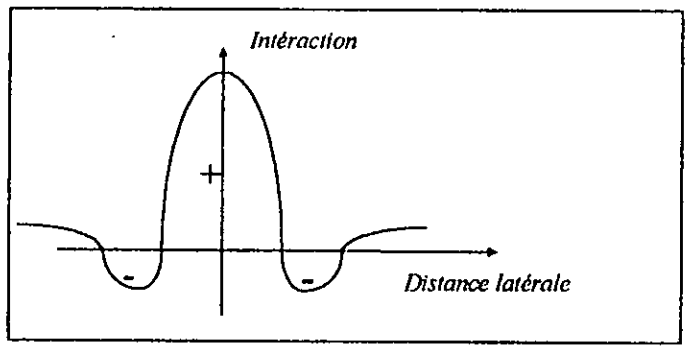


Fig.1.11. ***La fonction du chapeau mexicain.***

Cette courbe s'interprète comme suit:

1. dans une zone latérale proche du neurone, les neurones qui lui sont connectés ont une activité excitatrice,
2. dans un voisinage plus lointain, leur activité est inhibitrice,
3. l'action des neurones les plus lointains est négligeable.

L'interaction latérale entre les neurones doit provoquer la formation d'amas de neurones excités autour du neurone le plus stimulé par le signal d'entrée. Les autres neurones se stabilisent dans un état d'activation faible.

1.6. APPRENTISSAGE

Une des caractéristiques les plus intéressantes des réseaux de neurones est leur capacité à apprendre. L'apprentissage va permettre au réseau de modifier sa structure interne (poids synaptiques) pour s'adapter à son environnement.

1.6.1. PRINCIPE

Dans le cadre des réseaux de neurones, un réseau est définit par son graphe de connexions et la fonction d'activation de chaque neurone.

A chaque choix de coefficients synaptiques (poids de connexions) correspond alors un système, et c'est dans l'ensemble de tous ces systèmes que l'on se propose de trouver celui résolvant au mieux le problème.

Pour pouvoir évaluer un système particulier, nous effectuons pour cela une suite d'expériences permettant d'observer le comportement du réseau. Une expérience consiste à présenter un exemple d'entrée au système dont la réponse est recueillie à sa sortie.
Le réseau est ainsi évalué par la valeur prise par une fonction d'erreur. Le problème d'apprentissage consiste alors à trouver un réseau minimisant cette fonction d'erreur[4][5][7].

1.6.2. **MODES D'APPRENTISSAGE**[4][5][6].

Nous distinguons trois modes d'apprentissage: l'apprentissage supervisé, non supervisé et renforcé.

1.6.2.1. APPRENTISSAGE SUPERVISE.

Dans ce mode, un professeur qui connaît parfaitement la sortie désirée ou correcte guide le réseau en lui apprenant à chaque étape le bon résultat. Donc l'apprentissage ici consiste à comparer le résultat obtenu avec le résultat désiré, puis à ajuster les poids des connexions pour minimiser la différence entre les deux.

1.6.2.2. APPRENTISSAGE NON SUPERVISE.

Dans l'apprentissage non supervisé, le réseau modifie ses paramètres en tenant compte seulement des informations locales. Ces méthodes n'ont pas besoins de sorties désirées préétablies. Les réseaux utilisant cette technique sont appelés réseaux à dynamique autonome et sont considérés comme des détecteurs de régularité, car le réseau apprend en détectant les régularités dans la structure des vecteurs d'entrée et produit la sortie la plus satisfaisante.

1.6.2.3. APPRENTISSAGE RENFORCE.

Il est utilisé quand une information en retour sur la qualité de la performance est fournie, mais que la conduite souhaitée du réseau n'est pas complètement spécifiée par un professeur.
Donc l'apprentissage est moins dirigé que l'apprentissage supervisé. Contrairement à l'apprentissage non supervisé où aucun signal de retour n'est donné, le réseau à apprentissage renforcé peut utiliser le signal de renforcement pour trouver les poids les plus désirables quand c'est nécessaire.

1.6.3. REGLES D'APPRENTISSAGE .

La méthode d'ajustement des poids du réseau pendant l'apprentissage peut être choisie dans une gamme variée d'algorithmes, dont les plus connus sont cités ci dessous.

1.6.3.1. LA REGLE DE *HEBB* (OU REGLE DE CORRELATION).

L'apprentissage de *Hebb* est le premier mécanisme d'évolution des synapses proposé, par *D.O.Hebb* en 1949. Une interprétation de cette règle pour les réseaux de neurones est la suivante: Nous considérons que si deux neurones connectés entre eux sont activés en même temps, la connexion qui les relie doit être renforcée, dans le cas contraire elle n'est pas modifiée[4].

Sa formalisation est :

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + e.s_i.s_j \qquad (1.6)$$

Où :

$w_{ij}^{(t)}$ *: est le poids de la connexion reliant les neurones i et j à l'instant t*

e : nombre generalement compris entre 0 *et* 1,

t : est l'étape d'apprentissage.

1.6.3.2. LA REGLE DE *WIDROW-HOFF.*

Nous avons vu, que le perceptron est limité à des sorties binaires, *Widrow* et *Hoff* ont étendu l'algorithme d'apprentissage du perceptron en considérant une fonction de sortie continue [4][6].

Le principe est de :

- calculer l'erreur quadratique selon la formule suivante :

$$E(W) = \sum_{j=1}^{m} (Y_j - S_j)^2 \qquad (1.7)$$

Avec :

$S_j = \sum_i e_i . w_{ij}$

m : le nombre des unités de sortie ,

Y_j *: est la sortie désirée pour le neurone j.*

- minimiser cette erreur en modifiant le poids de chaque neurone suivant la règle :

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta . e_i (Y_j - S_j) \qquad (1.8)$$

1.6.3.3. LA REGLE DELTA

L'apprentissage selon cette règle consiste à calculer la réponse s_r, à la comparer avec la réponse correcte y_r, puis à actualiser les poids. Le cycle est répété jusqu'à ce que le réseau classe correctement tous les motifs d'entrée.

Soit $\delta_r = y_r - s_r$ le signal d'erreur et η est une constante. Ces poids sont incrémentés des quantités respectives $\Delta w_{ri} = \eta . \delta_r . s_i$; d'où la règle de modification des poids[7] :

$$w_{ri}^{(t+1)} = w_{ri}^{(t)} + \eta . \delta_r . s_i \qquad (1.9)$$

1.6.3.4. L'ALGORITHME DE LA RETRO-PROPAGATION DU GRADIENT .

Cet algorithme utilise la méthode du gradient pour minimiser l'erreur commise par un réseau multicouches, en utilisant une formule de récurrence permettant de calculer les signaux d'erreur des unités d'une couche à partir de ceux de la couche suivante. Le calcul s'effectue par couches successives en allant des unités de sortie vers les unités d'entrée. De plus, le calcul est local pour la topologie du réseau. En effet, le calcul du signal d'erreur d'une unité n'utilise que des informations relatives aux unités voisines et les poids des connexions les reliant.

Nous trouvons ainsi une solution au problème des unités cachées soulevé par *Minsky* et *Papert* en 1969 [1].

A la présentation de P exemples, la formulation de la règle de modification des poids w_{ij} est:

$$\Delta w_{ij} = -\eta \frac{1}{p} \sum_{k=1}^{p} \frac{\partial Err^{(k)}(w)}{\partial w_{ij}} \qquad (1.10)$$

Où :

$Err^{(k)}(w)$: *est l'erreur commise sur l'exemple k.*

1.6.3.5. APPRENTISSAGE COMPETITIF

L'apprentissage compétitif se fait par coopération et compétition entre les neurones. Les algorithmes d'apprentissage compétitif tentent à minimiser une fonction de distorsion entre un vecteur d'entrée E_i et un vecteur de poids W_{ei}.

La règle de modification des poids est donnée par l'équation suivante :

$$w_i^{(t+1)} = w_i^{(t)} + \eta(E_i - w_i^{(t)}) \qquad (1.11)$$

où :

w_i *: représente le vecteur poids,*

η *: est une constante qui contrôle la vitesse d'apprentissage,*

t *: représente le temps d'apprentissage.*

Pour finir, un algorithme d'apprentissage doit théoriquement se terminer lorsque un minimum d'erreur entre la sortie désirée et la sortie obtenue est atteint. Cependant, cette stratégie porte deux inconvénients:

1. elle risque de conduire à des temps d'apprentissage inutilement longs.
2. elle peut être nuisible aux capacités de généralisation du réseau (phénomène de surapprentissage)[6].

Il est préférable d'utiliser une mesure de la qualité de classification atteinte. Il s'agit de prendre un ensemble d'exemples distinct de l'ensemble d'apprentissage, puis évaluer le taux de réponse du réseau.

1.7. PROPRIETES ET LIMITES D'UTILISATION DES RNAs

1.7.1. PROPRIETES DES RNAs [4][6]

L'intérêt porté aujourd'hui aux réseaux de neurones tient sa justification dans les quelques propriétés suivantes :

1. La capacité d'adaptation

La capacité d'adaptation se manifeste dans les réseaux de neurones par la capacité d'apprentissage qui permet au réseau de tenir compte de nouvelles contraintes ou de nouvelles données. Cette capacité présente un intérêt déterminant pour les problèmes évolutifs.

2. La capacité de généralisation

La capacité de généralisation se traduit par la capacité d'un système à apprendre et à retrouver, à partir d'un ensemble d'exemples, des règles qui permettent de résoudre un problème.

3. Le parallélisme

Cette notion se situe à la base de l'architecture des réseaux de neurones considérés comme un ensemble d'entités élémentaires qui travaillent simultanément. Le parallélisme permet une rapidité de calcul supérieure mais exige de penser et de poser différemment les problèmes à résoudre.

4.La mémoire distribuée

Dans les réseaux de neurones, la " mémoire d'un fait " correspond à une carte d'activation des neurones. Cette carte est en quelque sorte un codage du fait mémorisé. L'intérêt de cette distribution de la mémoire sur plusieurs entités est la résistance au bruit.

1.7.2. LIMITES D'UTILISATION DES RNAs

Ces limites sont tout d'abord d'ordre technique et sont dues aux difficultés que nous rencontrons pour utiliser le parallélisme inhérent aux réseaux de neurones. Ainsi, la plupart des réseaux sont simulés sur des machines séquentielles, ce qui entraîne des temps de calculs importants. L'implémentation hardware des *RNAs* sur des circuits spécialisés tente de contourner cette limitation.

De plus, l'un des principaux reproches fait aux réseaux de neurones tient de leur incapacité à expliquer les résultats qu'ils fournissent. Les réseaux se présentent comme des boîtes noires dont les règles de fonctionnement sont inconnues. La qualité de leurs performances ne peut être mesurée que par des méthodes statistiques, ce qui amène une certaine méfiance de la part des utilisateurs potentiels[4].

1.8. CONCLUSION

Un réseau de neurones est un ensemble de cellules interconnectées par des liens ajustables. Cette propriété lui permet de trouver une représentation interne d'un problème donné. Le réseau, grâce à cette représentation, est capable de reproduire la sortie appropriée pour une entrée qui lui est présentée.

Les propriétés des réseaux de neurones ont permis à ces derniers de trouver une large utilisation dans divers domaines d'application tels que le traitement de l'information, la reconnaissance de formes, le traitement de signal, la robotique, l'optimisation, etc...

Leur implémentation digitale sur des circuits spécialisés massivement parallèles est un des axes de recherche actuels. Notre contribution est décrite dans les chapitres suivants.

Chapitre 2.

ARCHITECTURE DIGITALE DES RNAs

2.1 INTRODUCTION

Les réseaux de neurones artificiels(*RNAs*) sont des réseaux parallèles massivement interconnectés des éléments simples et leur organisation hiérarchique est destinée à interagir avec les objets du monde réel de la même manière que le système nerveux biologique.

Il y' a deux aspects importants dans la conception architecturale des circuits *VLSI* ; une est la complexité fonctionnelle de chaque Processeur Elémentaire (*PE*), l'autre est la structure d'interconnexion du système du réseau, ainsi que deux aspects correspondant à l'architecture des *RNAs*, qui sont respectivement les unités de traitement et la forme de connectivité des poids.

L'intérêt principal dans un algorithme orienté vers la conception d'un processeur de réseau est : Etant donné un algorithme, comment un processeur réseau est systématiquement dérivé. En se basant sur cette méthodologie systématique d'implémentation, quel est le type de conception spécialement facile à dériver[1].

Les implémentations analogiques et digitales des réseaux de neurones constituent une partie de recherche dans le domaine.

Les réseaux de neurones analogiques doivent être utilisés dans des étages de prétraitement pour la compression et l'interprétation initiale, vu que la vitesse des données dans les applications réelles dépassent la capacité des réseaux de neurones digitaux (Ex : la vision). D'autre part, les étages de prétraitement nécessitent toujours moins de précision, ce qui est convenable pour les réseaux analogiques. Enfin, assumant une réduction (données initiales) était faite, les étages de prétraitement appellent à une implémentation digitale, vu les exigences de flexibilité, où on peut exploiter son avantage de partitionner de nombreux problèmes et d'implémenter différents réseaux avec le même matériel [9].

Dans ce chapitre, nous présentons deux méthodes d'implémentation hardware digitale des *RNAs* à partir desquelles un choix sera fait quant à la manière d'implémenter ces réseaux, dans notre cas.

2.2. OPPORTUNITÉS DE L'IMPLÉMENTATION DIGITALE

L'approche matérielle se heurte à certains problèmes, particulièrement pour une réalisation analogique :

- Un stockage non volatil des poids analogiques donne une grande densité synaptique, mais peut être souvent non suffisamment programmable.
- La conception d'une synapse, la taille du réseau et le degré de résolution analogique sont dépendants l'un de l'autre.
- Le problème majeur de la conception avec les circuits analogiques est la relation de la précision avec l'espace du chip.
- L'espace minimal du chip est aussi influencé par des facteurs comme le bruit et la consommation du courant .
- La précision limitée dans la graduation des poids (réalisés par exemple en formes de résistances ohmiques), veut dire d'une part qu'une précision de calcul limitée dans l'implémentation analogique et par conséquent influe sur le nombre et la complexité des vecteurs de formes que peuvent être traités par le réseau analogique. D'autre part la précision limitée dans le traitement d'information par les réseaux de neurones analogique veut dire que l'information doit être codée d'une façon floue ou redondante.

- Si un algorithme d'apprentissage (avec ses multiples itérations) est à implémenter en circuits analogiques, il est nécessaire de s'assurer à priori si l'application voulue peut être complètement implementée. Par conséquent, c'est la tâche de l'utilisateur concerné par l'analyse et la modélisation du problème de caractériser ces tâches de traitement de vecteurs de formes, dans lesquelles les dérivations des valeurs actuelles des poids par rapport aux autres precalculés peuvent être tolérées.

Tous ces problèmes de conception analogique font appel à une conception digitale, flexible, précise, programmable etc[9][10]. Ce qui donne des avantages clairs :

- L'analyse du problème orienté à une application et la modélisation des caractéristiques du réseau peuvent être faites indépendamment de la conception du circuit.
- Par simulation on peut augmenter la précision de calcul (longueur du mot) pour chaque bloc de fonction avant de passer à son implémentation. La technologie et l'architecture déterminent, seulement, combien de neurones peuvent être intégrés dans le chip et à quel moment ils sont activés.
- C'est la tâche du concepteur d'implémenter les algorithmes selon l'état de l'art dans le traitement du signal digital [10].

• La grande résolution des poids, qui est facile à implémenter dans l'architecture digitale, permet plus d'une pondération individuelle des signaux d'entrée que dans le cas analogique. En plus, la conception digitale peut supporter facilement la longueur du mot nécessaire pour implémenter l'algorithme d'apprentissage[1][9].

• A l'inverse de l'implémentation analogique, qui souffre de résoudre des problèmes qui nécessitent un nombre de neurones plus grand que la taille physique du réseau de neurones, vu l'inexistence d'une méthode pour stocker les valeurs intermédiaires, reprogrammer le réseau, combiner les résultats pour plusieurs passages et la difficulté d'implémenter hiérarchiquement les réseaux interconnectés pour résoudre les grands problèmes. L'implémentation digitale offre une technologie moderne, une grande précision et une grande échelle d'intégration.

• En utilisant la logique digitale et les mémoires, il est plus facile de partitionner un grand problème afin d'être résolus par une petite implémentation (en terme de matériel). En utilisant la même logique et des mémoires. Une implémentation digitale peut réaliser plus d'un réseau et combine les résultats d'une façon hiérarchique pour résoudre un grand nombre de problèmes.

Les principaux inconvénients de l'implémentation digitale *VLSI* est l'espace de silicium nécessaire, relativement une vitesse et un coût d'interconnexion des unités de traitement faibles.

Plusieurs considérations nécessaires au traitement parallèle et à l'implémentation digitale de l'architecture du réseau sont résumées ci-dessous [9][10].

- Les approches de convergence, de mise à jour (synchrone) de la dynamique du système dans la phase de relaxation doivent être examinées.
- La conception architecturale doit assurer un traitement dans les deux phases qui partagent la même configuration du réseau (stockage et matériel de traitement). Ceci n'accélère pas l'apprentissage en temps réel, mais évite de recharger les poids synaptiques pour la phase de relaxation.
- La conception digitale doit identifier une technique propre et arithmétique pour calculer efficacement les opérations nécessaires pour les deux phases.
- La conception architecturale *VLSI* doit assurer un calcul intensif et en pipeline et encore augmenter ses possibilités de communication.
- L'implémentation digitale doit fournir une grande flexibilité, donc une architecture programmable de but général est préférée pour implémenter une large variété d'algorithmes de réseaux de neurones dans les deux phases.

2.3. ARCHITECTURE PAR COMMUTATEURS PROGRAMMABLES

Cette approche constitue une architecture distribuée et synchrone. Un processeur de base est associé. Il est capable d'exécuter en autonomie toutes les opérations de relaxation et d'apprentissage. La circulation de données est implementée par des techniques de décalage[11].

- Les données sont stockées dans des mémoires spécialisées.
- Le traitement des données se fait par un processeur neuronal.

2.3.1. CHOIX ARCHITECTURAL DE BASE

• Le processeur neuronal

Dans cette approche une entité physique est associée avec chaque neurone, c'est un processeur à application spécifique(*ASIP*). On suppose que le réseau de neurones est constitué d'une seule couche (réseau de *Hopfield*) ou de plusieurs couches (réseau multicouches).

Chaque neurone est connecté à tous les neurones de la couche précédente dans le réseau multicouches ou à tous les autres neurones dans le cas du réseau de *Hopfield* [11].

Dans la phase de relaxation le processeur neuronal effectue les opérations classiques montrées dans la figure.2.1.

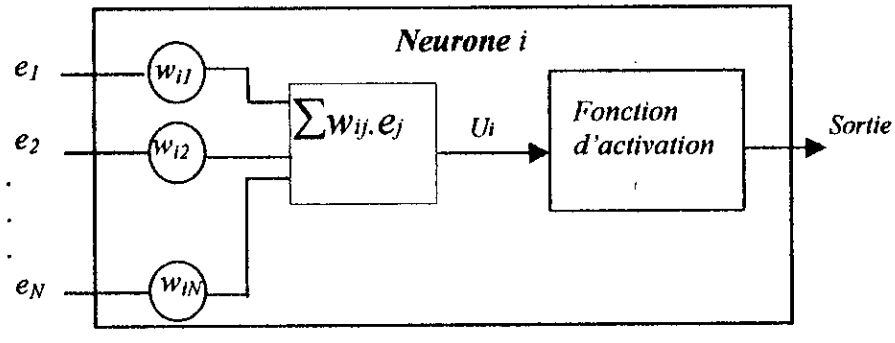


Fig.2.1. La fonction réalisée par le neurone dans la phase de relaxation.

• Connexion entre les neurones et l'architecture globale.

Le grand inconvénient, qui émerge lors de l'implémentation des réseaux de neurones, c'est lorsqu'on réalise un nombre important de connexions entre les neurones. Pratiquement des connexions virtuelles sont implementées pour remplacer les connexions spatiales. Ce qui veut dire que si le processeur ne veut pas être connecté à un grand nombre de voisins, l'état de ces voisins

circule dans des registres à décalage et le processeur prend au bon moment les valeurs des états de ses prédécesseurs.

L'architecture globale souhaitée est de construire des blocs élémentaires, qui sont des réseaux de processeurs neuraux à deux dimensions (2D). Un bloc correspond à une couche ou à une portion de couche dans les réseaux multicouches ou à une portion du réseau entier de *Hopfield*.

Dans le bloc, le processeur est organisé en des rangs entre deux bus. Ces bus sont faits de segments de bus qui peuvent être connectés par des commutateurs programmables lorsqu'ils sont nécessaires (utiles) dans une phase de calcul.

Pour détailler l'architecture globale, Nous traitons un exemple de 8 processeurs réalisé par un réseau de 2x4. Les architectures complètes seront construites comme un assemblage de blocs de base. Ces 8 processeurs constituent une couche dans les réseaux multicouches ou une portion de couche dans le réseau de *Hopfield*.

En premier lieu, nous allons voir comment se fait le calcul des potentiels donnés par les équations (2.1).

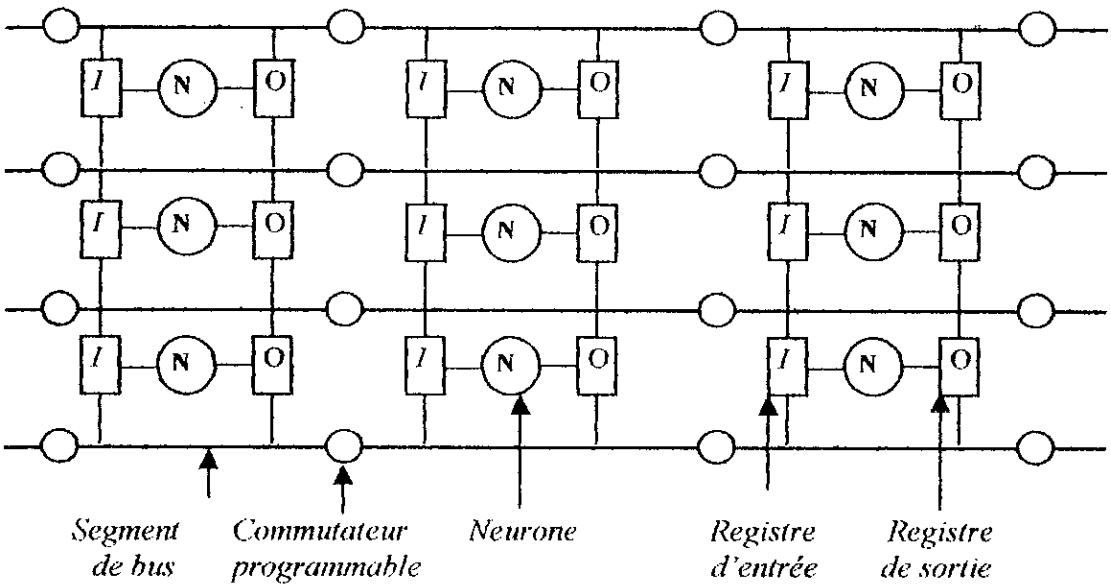


*Fig.2.2. **Architecture globale «vue partiale»***

Dans la première étape de calcul, les segments de bus sont connectés comme le montre la figure.2.3. Les commutateurs de la deuxième colonne sont ouverts, l'état du neurone *N1* est disponible sur le segment de bus du deuxième rang pour les deux neurones *N1* et *N2*. L'état du neurone *N3* est similairement disponible sur le prochain segment de bus et la même chose pour *N5* et *N7*. Les calculs exécutés pendant cette étape sont donnés par la figure.2.4.[11]

Dans la deuxième étape de calcul, un décalage circulaire est fait dans la première couche comme il est indiqué dans la figure.2.5.

$$U_1 = w_{11}.e_1 + w_{12}.e_2 + \ldots\ldots + w_{18}.e_8$$
$$U_2 = w_{21}.e_1 + w_{22}.e_2 + \ldots\ldots + w_{28}.e_8$$
$$\vdots \qquad (2.1)$$
$$U_7 = w_{71}.e_1 + w_{72}.e_2 + \ldots\ldots + w_{78}.e_8$$
$$U_8 = w_{81}.e_1 + w_{82}.e_2 + \ldots\ldots + w_{88}.e_8$$

L'état du neurone *N1* a été décalé sur le prochain segment de bus et il est disponible pour tous les neurones *N5* et *N6*. Il est clair que 4 décalages élémentaires " un décalage circulaire complet" est nécessaire pour exécuter la moitié du calcul complet.

Deux décalages circulaires complets (8 décalages élémentaires) sont nécessaires pour accomplir le calcul. Une fois le calcul est fait, le contrôleur initialise le calcul de la fonction d'activation.[11]

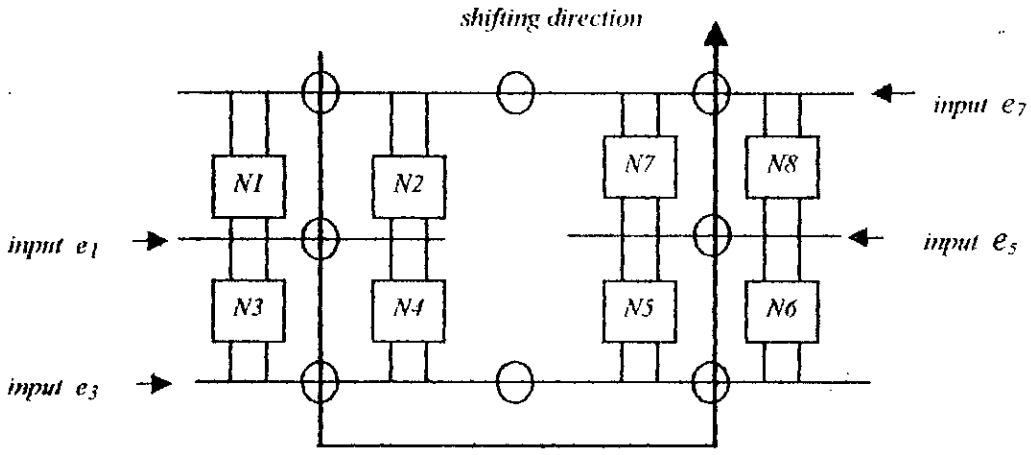


Fig.2.3. ***Bloc de 8 neurones***

$$U_1 = \boxed{w_{11}.e_1} + w_{12}.e_2 + \ldots\ldots + w_{18}.e_8$$
$$U_2 = \boxed{w_{21}.e_1} + w_{22}.e_2 + \ldots\ldots + w_{28}.e_8$$
$$U_3 = w_{31}.e_1 + w_{32}.e_2 + \boxed{w_{33}.e_3} + \ldots\ldots + w_{38}.e_8$$
$$U_4 = w_{41}.e_1 + w_{42}.e_2 + \boxed{w_{43}.e_3} + \ldots\ldots + w_{48}.e_8$$
$$\vdots$$
$$U_7 = w_{71}.e_1 + w_{72}.e_2 + \ldots\ldots + \boxed{w_{77}.e_7} + w_{78}.e_8$$
$$U_8 = w_{81}.e_1 + w_{82}.e_2 + \ldots\ldots + \boxed{w_{87}.e_7} + w_{88}.e_8$$

Fig.2.4. ***Opérations exécutées dans la première étape.***

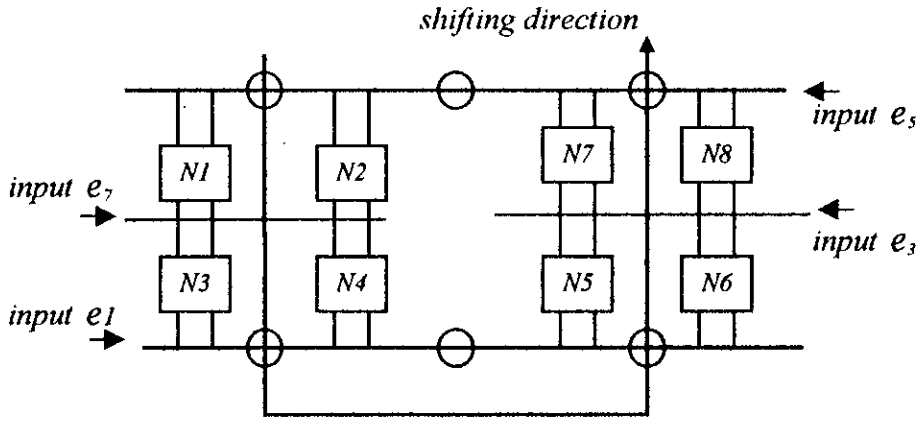


Fig.2.5. ***La deuxième étape de calcul.***

2.3.2. ARCHITECTURE DETAILLEE

L'architecture est une architecture synchrone distribuée. Chaque processeur neuronal a un contrôle autonome et il est capable d'exécuter toutes les opérations du calcul neuronal (potentiel et fonction d'activation, phases de relaxation et d'apprentissage).

Le processeur neuronal contient un ensemble de registres d'identifications contenant les informations nécessaires sur l'architecture globale du réseau. Un chargement initial de ces registres adaptera le réseau au calcul voulu. Les mémoires locales sont des RAMs pour implementer le chip dans les deux phases de relaxation et d'apprentissage et une EPROM pour les chips optimisés, spécifiques à l'utilisation pour une application choisie. Pour un circuit ASIC optimisé pour la phase d'utilisation, la technologie des mémoires ROMs doit être utilisée pour les registres d'identification et les coefficients de la mémoire. Le premier prototype réalisé utilise 64 x 8 bits (*RAM*) [11].

Cette approche est complètement différente des autres approches basées sur des dispositifs de multiplication puissants laissant à l'extérieur d'autres fonctions.

2.3.2.1. STRUCTURE DU NEURONE [11]

Le processeur neuronal est constitué de :

1. sept (7) registres d'identification (*IDR*)
   - *IDR1* contient le nombre de bits codant l'état,
   - *IDR2* contient le nombre de rangés dans le bloc multiplié par 2,
   - *IDR3* contient le nombre de coefficients stockés dans la mémoire,

• *IDR4* contient la direction du décalage des entrées.

Par exemple : Si *IDR4=0*, le décalage se fait vers le bas.

Si *IDR4=1*, le décalage se fait vers le haut.

• *IDR5, IDR6, IDR7*, chacun contient le nombre nécessaire de décalages pour former le résultat de chaque multiplication.

2. Une mémoire locale(*RAM*) contenant les coefficients $\{w_{ij}\}$.
3. Un chemin de données contient des opérateurs(en multiplications et additions / soustractions).
4. Un contrôleur, contrôle le calcul du potentiel, l'état du neurone et le transfert de données avec les autres neurones.
5. Un registre d'entrée, mémorise et décale la valeur de l'entrée.
6. Un registre de sortie, mémorise l'état du neurone.
7. Trois compteurs *C1, C2, C3.*

C1 est initialement chargé avec la valeur du registre *IDR1* et décrémenté pendant l'opération de multiplication.

C2 est utilisé dans chaque état de décalage, initialement il est chargé par une des valeurs des registres *IDR2, IDR5, IDR6, IDR7* et décrémenté pendant le décalage.

C3 est initialement chargé par la valeur du registre *IDR3* et décrémenté après chaque cycle de lecture de la mémoire.

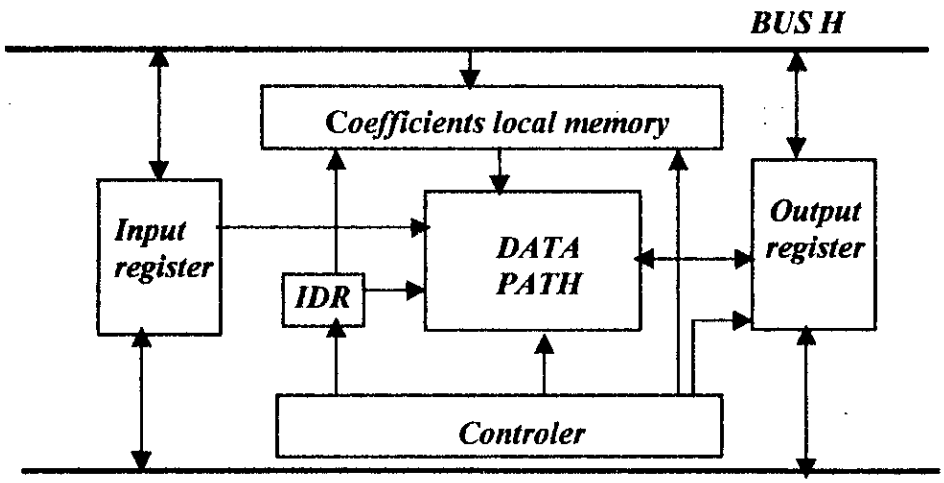


Fig 2.6. Exemple d'architecture d'un processeur neuronal

2.3.2.2. ALGORITHME UTILISE POUR LE CALCUL DU POTENTIEL.

Le chemin de données utilisé dans la phase de relaxation calcule le potentiel du neurone donné par $\sum w_{ij}.e_j$. Où $\{w_{ij}\}$ et $\{e_j\}$ sont respectivement les coefficients et les états d'entrée.

Tous les deux, coefficients et états d'entrée, sont représentés en complément à deux (*C2*) et codés sur *IDR1* bits. Le chemin de données est constitué de deux opérateurs, le multiplieur et l'additionneur / soustracteur.

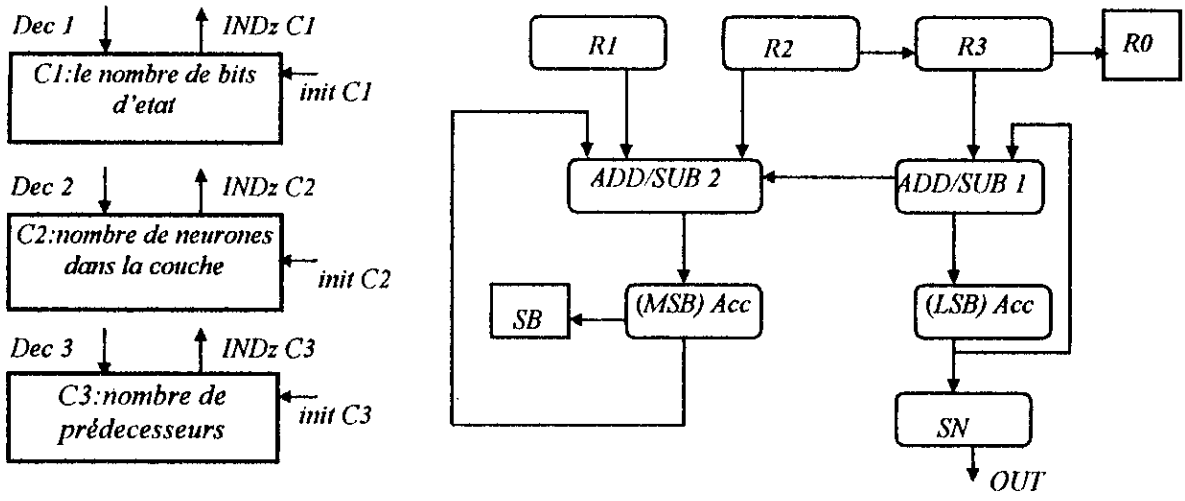


Fig.2.7. ***Structure du chemin de données.***

2.3.2.3. ALGORITHMES UTILISES POUR LA PHASE D'APPRENTISSAGE.

Différents algorithmes existants sont utilisés pour la phase d'apprentissage des *RNAs,* comme on peut faire appel à un superviseur ou non.

Dans cette approche, l'implémentation d'un algorithme d'apprentissage très populaire est examinée, c'est l'algorithme de la rétro-propagation de l'erreur (un réseau de taille constante et un apprentissage supervisé), pour lequel le processeur neuronal élabore ses poids synaptiques successifs par la comparaison de la sortie à un ensemble d'entraînement spécifique de sorties désirées par un superviseur.

L'algorithme de la rétro-propagation de l'erreur peut être décrit par les étapes suivantes.

Etape 1 : Initialisation des poids.

Etape 2 : Présentation des entrées et des sorties désirées.

Etape 3 : Calcul des sorties actuelles.

Etape 4 : Vérification des critères de minimisation.

- Si le critère est satisfais, l'apprentissage est fini.
- Sinon, l'apprentissage n'est pas complété et aller à l'étape 5.

Etape 5 : Mise à jour des poids en commençant par la dernière couche.

Etape 6 : Répéter en allant à l'étape 2.

Le critère est la minimisation de l'erreur quadratique moyenne entre la sortie désirée et la sortie calculée du neurone dans la dernière couche.

• Mise à jour des poids :

On définit l'erreur δ_j entre la sortie désirée et la sortie calculée (S_o) du neurone j.

On appelle l'erreur pondérée du neurone j induit par le neurone i, l'expression $w_{ij}.\ \delta_j$.

Si nous considérons que (K est le nombre de neurones dans la couche) le neurone i a K successeurs dans la prochaine couche, nous appelons l'erreur globale :

$$\Delta_i = \sum_{j=1}^{K} w_{ij}.\delta_j \qquad (2.2)$$

La mise à jour des poids est faite selon la formule suivante :

$$w_{ij}(t+1) = w_{ij}(t) + \eta.x_i.f'(x_i) \qquad (2.3)$$

Où η est une constante, x_i représente la sortie du neurone i et f' est la dérivée de la fonction d'activation f.

• Implémentation de l'algorithme

Nous détaillons dans ce paragraphe comment l'erreur quadratique moyenne et l'erreur pondérée globale peuvent être calculées sur l'architecture présentée précédemment.

On suppose que le réseau est constitué de L couches. La deuxième couche est constituée de N_L neurones, pliés en m colonnes de ($C1$ à Cm) et R rangs.

- Chaque neurone dans la dernière couche calcule son erreur quadratique moyenne.
- Deux colonnes $C1$ et $Cm/2+1$ accumulent respectivement les erreurs quadratiques des colonnes $C1$ à $Cm/2$ et de $Cm/2+1$ à Cm.
- Un neurone arbitraire, par exemple le neurone $N1$, accumulera l'erreur quadratique moyenne finale. Donc, les erreurs partielles circulent de la colonne $Cm/2+1$ au neurone $N1$.
- Le neurone $N1$ compare le résultat de cette somme S à la valeur V fixée par le superviseur.
- Si S est inférieur ou égal à V, le prototype présenté est appris par le réseau.
- Si S est plus grand que V, une nouvelle mise à jour des poids est nécessaire.

Nous illustrons ça par un exemple ayant 12 neurones dans la dernière couche, organisé en 4 colonnes et 3 rangs. [Exemple empreinté à la référence 11]

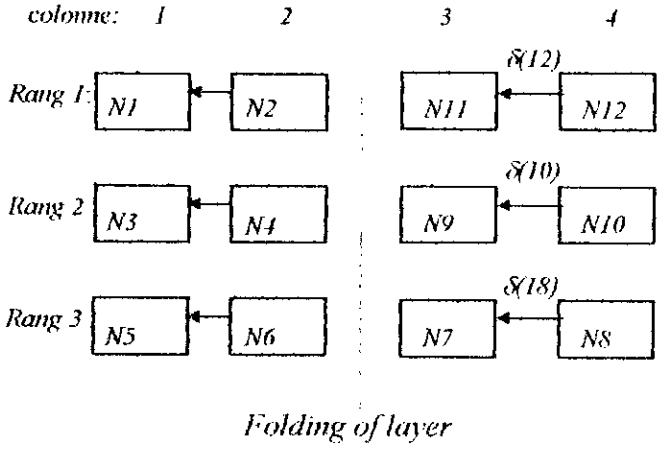


Fig 2.8.a. ***Somme des erreurs de C1 et C2, C3 et C4***

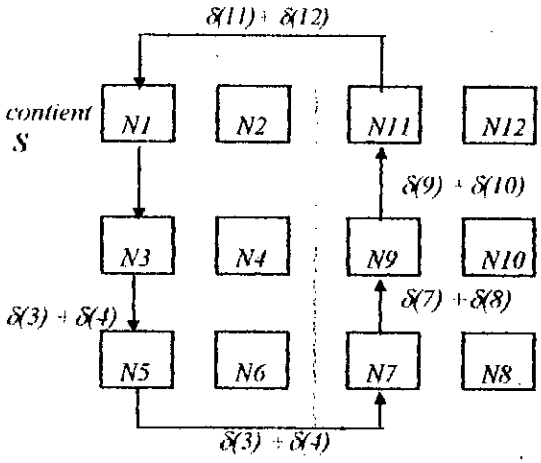


Fig .2.8.b. ***Somme des erreursde C1 et C3***

• Calcul de l'erreur globale pondérée

Le calcul de l'erreur globale pondérée suppose que les erreurs des neurones dans la i^{eme} couche sont connues. La i^{eme} couche calculera l'erreur globale de chaque neurone dans la couche *i-1* avant la mise à jour de ses poids. Chaque neurone dans la i^{eme} couche contribuera par le calcul du terme d'erreur pondérée pour le neurone *Ni* dans la couche *i-1* , et le même principe de circulation de données est adopté (utilisé pour le calcul de l'erreur quadratique moyenne, pour accumuler les erreurs pondérées).

2.4. ARCHITECTURE SYSTOLIQUE DES RNAs

Cette conception commence par un graphe de dépendance d'états pour exprimer la récurrence et le parallélisme associés avec la description des activités espace-temps. C'est une conception facile pour la classe des algorithmes représentés par des matrices ou des algorithmes récursifs. Trois étapes sont à envisager dans cette conception[9] :

1. Exprimer le traitement neural en terme d'opérations de matrices récursives (itératives).
2. Exprimer les opérations des matrices par des graphes de dépendances.
3. Implémenter les graphes de dépendance sur des réseaux systoliques.

Dans le but d'offrir une grande flexibilité, la conception d'une architecture pour un but général pour le traitement neural est souhaitée. En effet, la conception systolique peut être

généralisée pour devenir applicable à une variété de formes de connexions. Les modifications majeures nécessaires sont sur la conception fonctionnelle dans le *PE* pour refléter les fonctions d'activation et les règles d'apprentissage.

2.4.1. REPRESENTATION DES OPÉRATIONS MATRICIELLES PAR DES GRAPHES DE DEPENDANCES

Cette technique est une approche utilisée pour la dérivation d'une structure d'un algorithme manipulant des opérations matricielles. Elle consiste à représenter l'algorithme par un graphe de dépendances de données (*DDG*) dans lequel chaque nœud correspond à une opération et un lien est associé à lui pour communiquer avec les autres nœuds. Cette structure offre les avantages suivants:

- Elle est souhaitée pour une exécution successive d'un algorithme avec des ensembles données différents[12].
- Un minimum de temps de calcul pour chaque opération.
- Un maximum de débit (flux) pour les instances.
- Une utilisation optimale des multiples instances.

Elle souffre de certains inconvénients comme :

- Un grand nombre de *PEs* lorsque la dimension de la matrice est grande.
- Un flux de données important (une large bande d'E/S).
- Des connexions irrégulières ou complexes possibles dans le graphe.

L'algorithme de produit vecteur-matrice (*VxB*), tel que $V=\{v_i,\ i=1...4\}$, ainsi que l'algorithme de produit de deux matrices (4x4) A et B, représentées respectivement par $\{a_{ij}, i=1...4, j=1...4\}$ et $\{b_{ij}, i=1...4, j=1...4\}$, sont représentés par les graphes de dépendances suivants.(Fig.9, Fig.10)

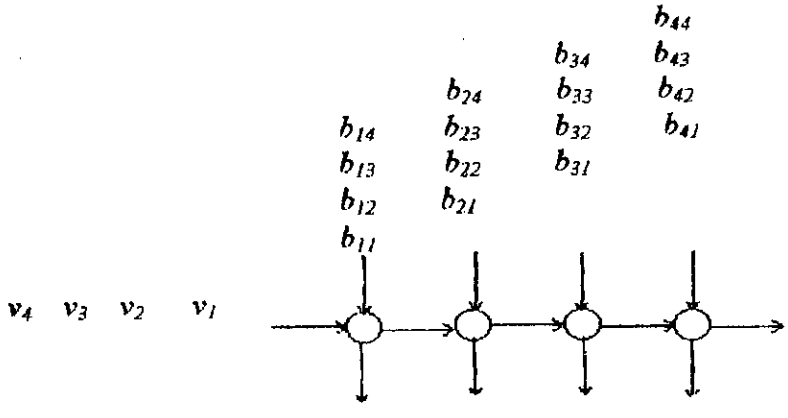


Fig.2.9. Graphe de dépendances de la multiplication Vecteur-Matrice(VxB)

« Les nœuds exécutent des opérations de multiplication et addition (*MAC*) »

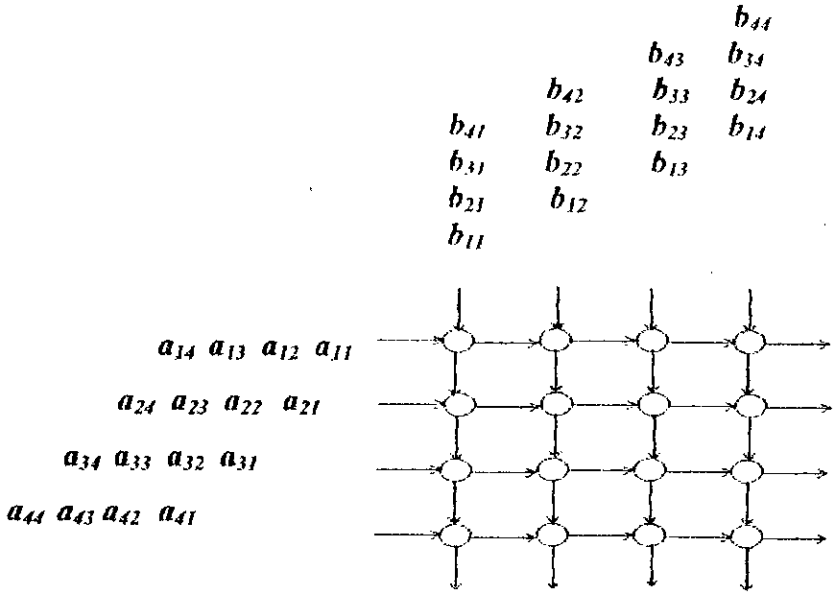


*Fig.2.10. **Graphe de dépendances de la multiplication de deux matrices (AXB).***

Dans le but de dériver une implémentation effective pour un algorithme de matrices, on doit avoir comme objectif la détermination d'une structure régulière avec une bande *d'E/S* et un nombre de *PEs* raisonnables. Cela signifie que le réseau doit avoir un nombre de cellules plus petit que le nombre des nœuds dans le graphe de dépendances. Donc, il est nécessaire d'implémenter les opérations sur les cellules en des étapes de temps adéquates. Cette procédure nécessite la détermination des caractéristiques du réseau, à quel moment et où, chaque opération est exécutée, et comment les données circulent dans le graphe. Par conséquent, l'implémentation d'un algorithme de matrices dans un réseau revêt deux aspects[12] :

1.Architecture :

Dans cet aspect on donne les caractéristiques des modules composant le réseau et leur interconnexion (cellules de traitement, mémoires et ports d'E/S). Dans la dérivation de l'architecture, il est nécessaire de prendre en considération les caractéristiques de l'algorithme (types d'opérations et dépendances) et les contraintes de la technologie.

2. Mapping:

Le mapping de l'algorithme sur l'architecture est de spécifier les opérations exécutées dans chaque cellule et à quel moment (dans quel ordre), et de spécifier aussi le flux de données à travers le réseau. Ce mapping doit vérifier que si deux opérations sont associées à la même cellule, elles ne doivent pas être exécutées en même temps.

2.4.2.RESEAUX SYSTOLIQUES

Les réseaux systoliques sont des processeurs intégrés spécialisés dont les caractéristiques principales sont les suivantes[13] :

- Un parallélisme massif et décentralisé.
- Des communications locales.
- Un mode opératoire synchrone.

Ils sont implémentés pour effectuer un traitement parallèle des données d'un algorithme manipulant des opérations matricielles comme la multiplication de deux matrices, la multiplication matrice-vecteur, résolution des systèmes d'équations linéaires...

La connexion entre processeurs peut être unidimensionnelle formant un réseau linéaire ou multidimensionnelle en des formes géométriques variantes (voir figure .11).

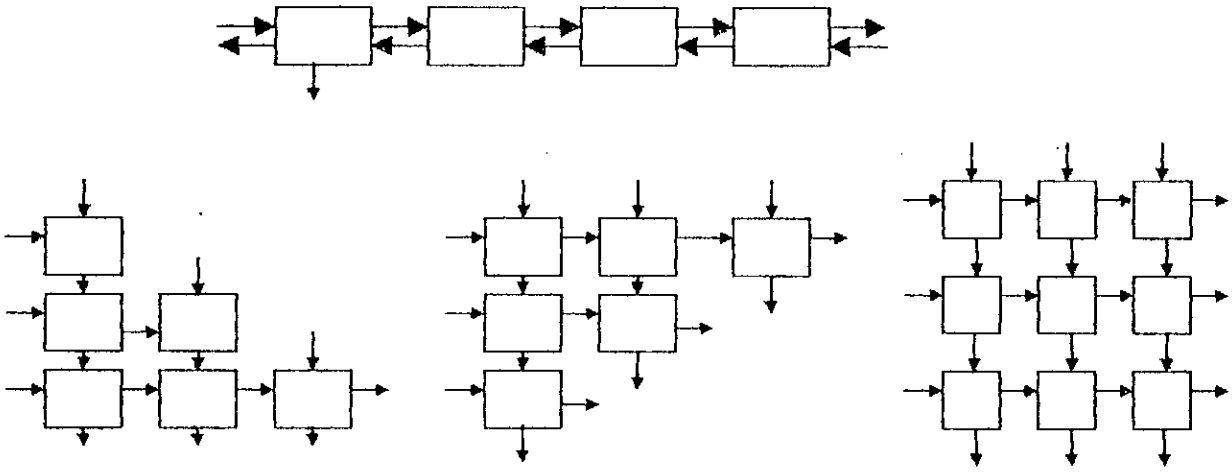


Fig.2.11. ***Exemples de réseaux systoliques types.***

Si l'option *VLSI* n'est pas retenue, la meilleure solution pour le traitement systolique des données consiste à exécuter les algorithmes sur des calculateurs à architectures parallèles. Ces derniers sont classés selon un critère de sélection : qui est le mode de contrôle des opérations élémentaires effectuées par les différents processeurs.

Deux classes de base ont été proposées, selon la multiplicité de flux d'instructions et de données disponibles [13].(voir fig 2.12 et 2.13)

- L'architecture SIMD: ue seule instruction et plusieurs données.
- L'architecture MIMD : plusieurs instructions et plusieurs données.

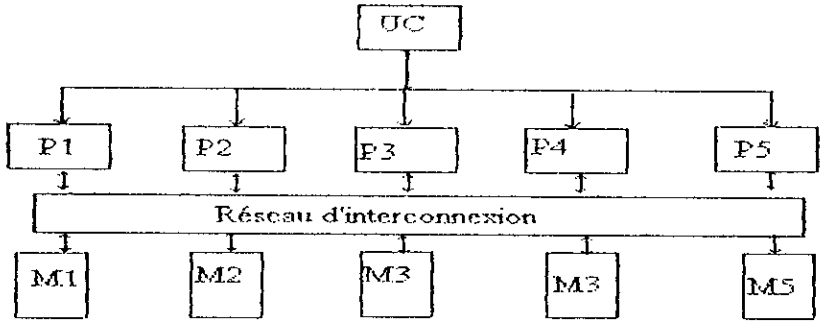


Fig.2.12. Architecture SIMD.

Où UC est l'unité de contrôle, M_i (i = 1 ... 5) sont des mémoires et P_i (i=1 ... 5)sont des processeurs.

Dans cette architecture toutes les unités de traitement sont supervisées par la même unité de contrôle, mais opèrent sur des ensembles de données distincts.

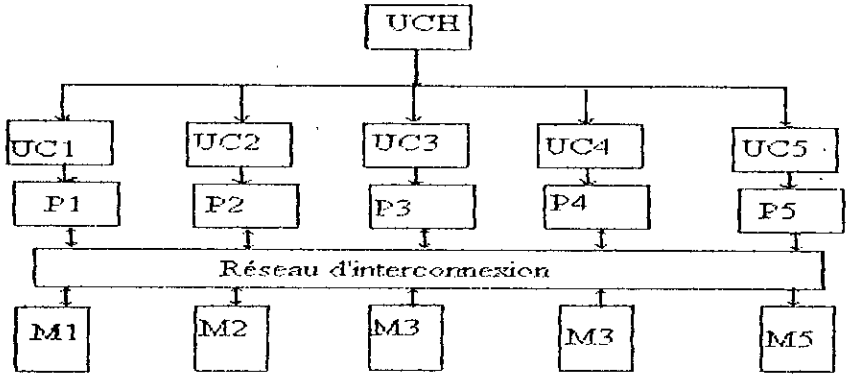


Fig.2.13. Architecture MIMD.

Dans ce cas chaque processeur possède sa propre unité de contrôle, donc ils ont un fonctionnement indépendant et exécutent des instructions différentes avec des flux de données différentes. Mais toujours il y' a une unité de contrôle hôte qui contrôle cette fois-ci les unités de contrôle.

2.4.3. REGULARISATION DES ALGORITHMES.

La régularisation des algorithmes est une transformation souhaitée pour l'implémentation des algorithmes sur des réseaux systoliques. L'application d'une technique de transformation à un algorithme se constitue de deux étapes : la régularisation de l'algorithme et l'implémentation sur réseaux systoliques.

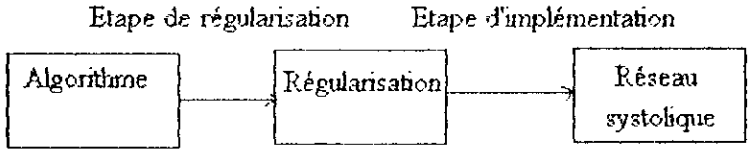


Fig2. 14. Etapes nécessaires pour l'implémentation d'un algorithme[12].

Les méthodes de transformation des algorithmes se différencient par la forme régularisée et la manière souhaitée pour effectuer les transformations afin de dériver une implémentation.

La plus part des représentations des algorithmes sont des expressions algébriques ou des descriptions graphiques.

Le graphe régularisé est projeté directement ou à travers d'autres transformations sur un réseau systolique

La notion d'équivalence des graphes est utilisée pour représenter en plusieurs manières la même instruction ; par exemple l'expression, $x = a + b + c + d$, peut être représentée par :

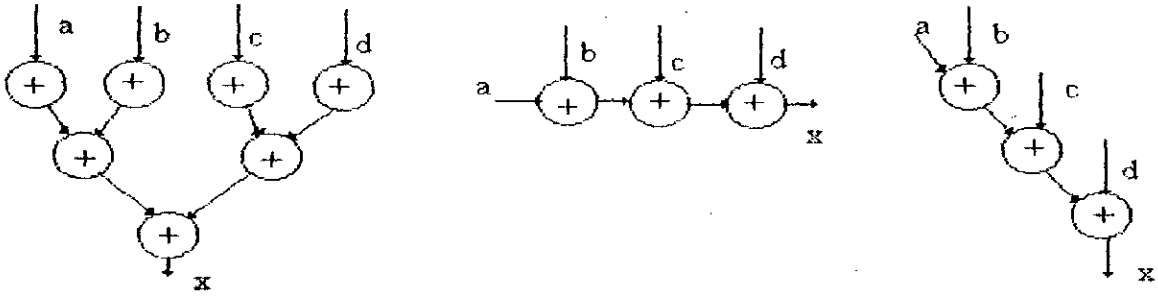


Fig.2.15. Graphes équivalents pour le calcul de x.

2.4.3.1. REGULARISATION DE L'ALGORITHME DE LA PHASE DE RELAXATION.

Le calcul du potentiel suivi par la fonction d'activation dans la phase de relaxation pour la couche $(l+1)$ dans un réseau de neurones multicouches est donné par :

$$\begin{cases} U(l+1) = W.a(l) + \theta \\ a(l+1) = F(U(l+1)) \end{cases} \qquad (2.4)$$

Où W est la matrice des poids reliant les neurones la couche l à la couche l+1, $a_i(l)$ est la sortie du neurone i de la couche l et θ : le biais de chaque neurone dans la couche l+1.

Cette équation est formulée comme un problème de multiplication matrice-vecteur (***MVM***) suivie par la fonction d'activation sigmoïde afin d'être implémentée sur des réseaux systoliques

Pour un réseau multicouches ayant N neurones dans la couche $(l+1)$ et N neurones dans la couche (l), le premier terme dans l'équation (2.4) peut être explicité comme suit :

$$U_1(l+1) = w_{11}.a_1(l) + w_{12}.a_2(l) + \ldots\ldots\ldots + w_{1N}.a_N(l) + \theta_1$$
$$U_2(l+1) = w_{21}.a_1(l) + w_{22}.a_2(l) + \ldots\ldots\ldots + w_{2N}.a_N(l) + \theta_2$$

$$U_{N-1}(l+1) = w_{N-1\,1}.a_1(l) + w_{N-1\,2}.a_2(l) + \ldots\ldots\ldots + w_{N-1\,N}.a_N(l) + \theta_{N-1}$$
$$U_N(l+1) = w_{N\,1}.a_1(l) + w_{N\,2}.a_2(l) + \ldots\ldots\ldots + w_{N\,N}a_N(l) + \theta_N$$

Une première transformation algébrique de cet algorithme est de plier ces équations suivant la diagonale, ce qui permet de les écrire sous la forme suivante :

$$U_1(l+1) = w_{11}.a_1(l) + w_{12}.a_2(l) + \ldots\ldots\ldots + w_{1\,N-1}.a_{N-1}(l) + w_{1N}.a_N(l) + \theta_1$$
$$U_2(l+1) = w_{22}.a_2(l) + w_{23}.a_3(l) + \ldots\ldots\ldots + w_{2N}.a_N(l) + w_{21}.a_1(l) + \theta_2$$

$$U_{N-1}(l+1) = w_{N-1\,N-1}.a_{N-1}(l) + w_{N-1\,N}.a_N(l) + \ldots\ldots\ldots + w_{N-1\,1}.a_1(l) + w_{N-1\,2}.a_2(l) + \theta_{N-1}$$
$$U_N(l+1) = w_{N\,N}.a_N(l+1) + w_{N\,1}.a_1(l) + \ldots\ldots\ldots + w_{N\,N-2}.a_{N-2}(l) + w_{N\,N-1}.a_{N-1}(l) + \theta_N$$

La représentation par un *DDG* de ces équations permet d'obtenir le graphe suivant :

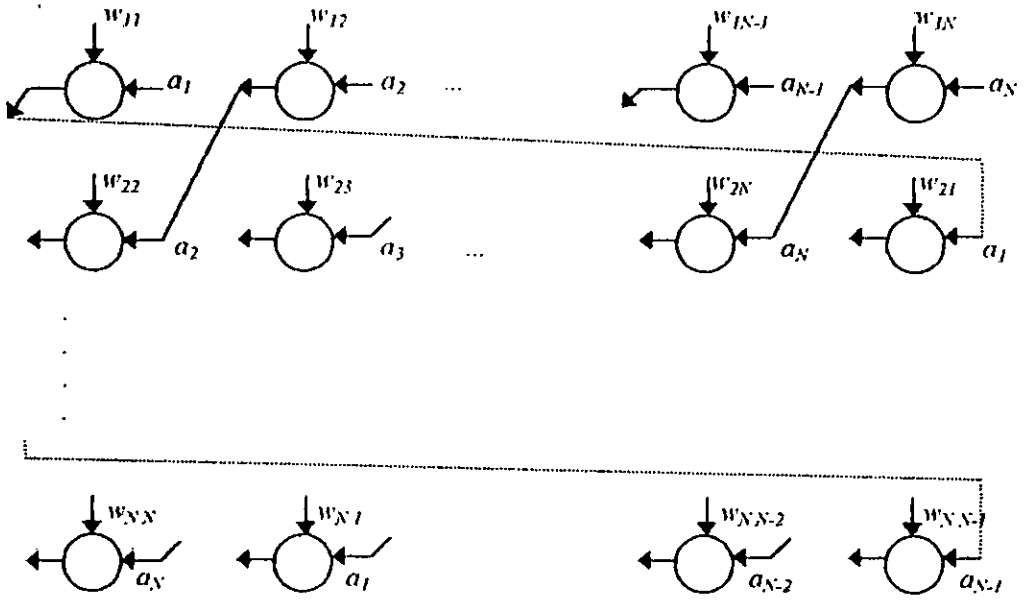


Fig.2.16. DDG pour le calcul du potentiel dans la phase de relaxation

Une projection sur la verticale des poids $\{w_{ij}\}$ et une projection sur l'horizontale des $\{a_i\}$ sont données par le graphe d'équivalence suivant :

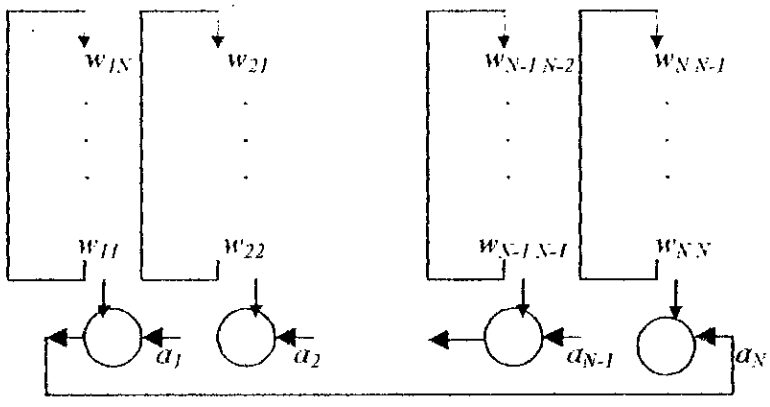


Fig.2.17. ***Projection du DDG de calcul du potentiel.***

Cette structure peut être implémentée directement sur un réseau systolique en anneau, en utilisant des cellules *MAC* avec une circulation verticale des poids et une circulation horizontale des $\{a_i\}$.

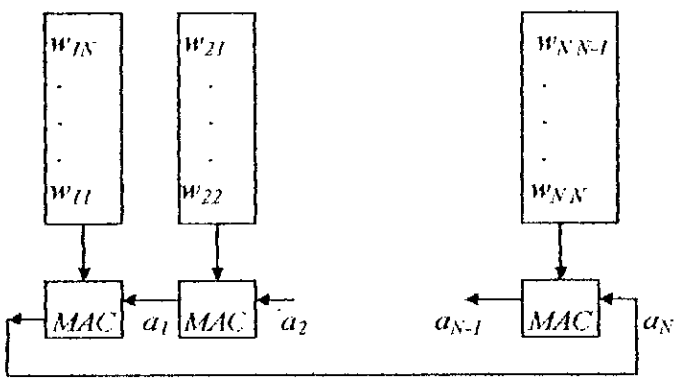


Fig. 2.18. ***Anneau systolique pour le calcul du potentiel.***

Après N itérations de multiplication et accumulation on fait appel à la fonction d'activation f pour calculer $\{a_i(l+1)\}$, qui eux même constituent les entées de la couche $(l+2)$. Donc des graphes de dépendances de données en cascade permettent de décrire les algorithmes et de dériver des réseaux systoliques en anneau pour un réseau multicouches.

2.4.3.2. REGULARISATION DE L'ALGORITHME DE LA PHASE D'APPRENTISSAGE.

La phase d'apprentissage consiste à ajuster les poids synaptiques selon une règle d'apprentissage parmi plusieurs telles que : la règle *Delta*, la règle de *Hebb*, la règle de *Boltzman*, la règle d'apprentissage compétitif etc...

La mise à jour des poids est exprimée par :

$$W^*_{ij}(l+1) = w_{ij}(l+1) + \Delta w_{ij}(l+1). \qquad (2.5)$$

Où $w^*_{ij}(l+1)$ *est la nouvelle valeur de* $w_{ij}(l+1)$ ($w_{ij}(l+1)$ *ajustée).*

Avec une formulation rigoureuse de $\Delta w_{ij}(l+1)$ sous forme de produits de deux termes $g_i(l+1)$ et $h_j(l+1)$, la variation du poids est écrite :

$$\Delta w_{ij}(l+1) = g_i(l+1).h_j(l+1). \qquad (2.6)$$

Pour l'algorithme *RPG*(Retro-Propagation du Gradient), les valeurs de h_j et g_i sont résumées dans le tableau suivant.

$g_i(l+1)$	$h_j(l+1)$	*Numéro de la couche*
$a_i(L) - t_i$	$-\eta.a_j(L-1)$	*La dernière couche (L)*
$\delta_i.f'_i(l+1)$	$-\eta.a_j(l)$	*La couche* ($0<l<L$)

Fig. 2.19 . ***Valeurs de h_j et g_i pour l'algorithme RPG.***

a_i *: représente l'état du neurone i .*

t_i *: représente la sortie désirée pour le neurone i.*

- **Calcul de l'erreur corrective** δ

Pour un réseau de L couches, les valeurs de $\delta_i(l)$ sont données par :

$$\delta_i(l) = \sum g_i(l+1).w_{ij}(l+1) \quad \text{telle que } l<L. \qquad (2.7)$$

Cette équation est représentée sous forme de multiplication vecteur-matrice (*VMM*) pour la dérivation de sa structure systolique.

La valeur de $\delta_i(L)$ pour chaque neurone i de la dernière couche est donnée par la différence entre la valeur de sortie calculée $a_i(L)$ et la valeur de sortie désirée t_i *c.a.d :*

$$\delta_i(L) = a_i(L) - t_i. \qquad (2.8)$$

Pour l'algorithme *RPG* l'apprentissage se fait par retour en arrière selon la règle d'apprentissage Delta généralisée. L'équation. peut être écrite :

$$w^*_{ij}(l+1) = w_{ij}(l+1) + g_i(l+1).\ h_j(l+1). \qquad (2.9).$$

Dans le but de dériver sa structure systolique l'équation (2.9) est formulée comme des opérations de mise à jour de produits externes(*OPU: Outer Product Updating*) .

Ainsi que l'équation (2.7) peut être développée comme suit :

$$\delta_1(l)=g_1.w_{11}+g_2.w_{21}+\ldots\ldots\ldots\ldots\ldots+g_{N-1}.w_{N-1\,1}+g_N.w_{N\,1}$$

$$\delta_2(l)=g_1.w_{12}+g_2.w_{22}+\ldots\ldots\ldots\ldots\ldots+g_{N-1}.W_{N-1\,2}+g_N.w_{N\,2}$$

.....

.....

$$\delta_N(l)=g_1.w_{1N}+g_2.w_{2N}+\ldots\ldots\ldots\ldots\ldots+g_{N-1}.w_{N-1\,N}+g_N.w_{N\,N}$$

De la même manière que la phase de relaxation un pliage suivant la diagonale conduit à la représentation suivante:

$$\delta_1(l)=g_1.w_{11}+g_2.w_{21}+\ldots\ldots\ldots\ldots\ldots+g_{N-1}.w_{N-1\,1}+g_N.w_{N\,1}$$

$$\delta_2(l)=g_2.w_{22}+g_{N-1}.W_{N-1\,2}+\ldots\ldots\ldots\ldots\ldots+g_N.W_{N\,2}+g_1.w_{12}$$

$$\delta_N(l)=g_N.w_{N\,N}+g_{N-1}.w_{N-1\,N}+\ldots\ldots\ldots\ldots\ldots+g_1.w_{1N}+g_2.w_{2N}$$

La représentation graphique de l'algorithme par un *DDG* est donnée par la figure suivante :

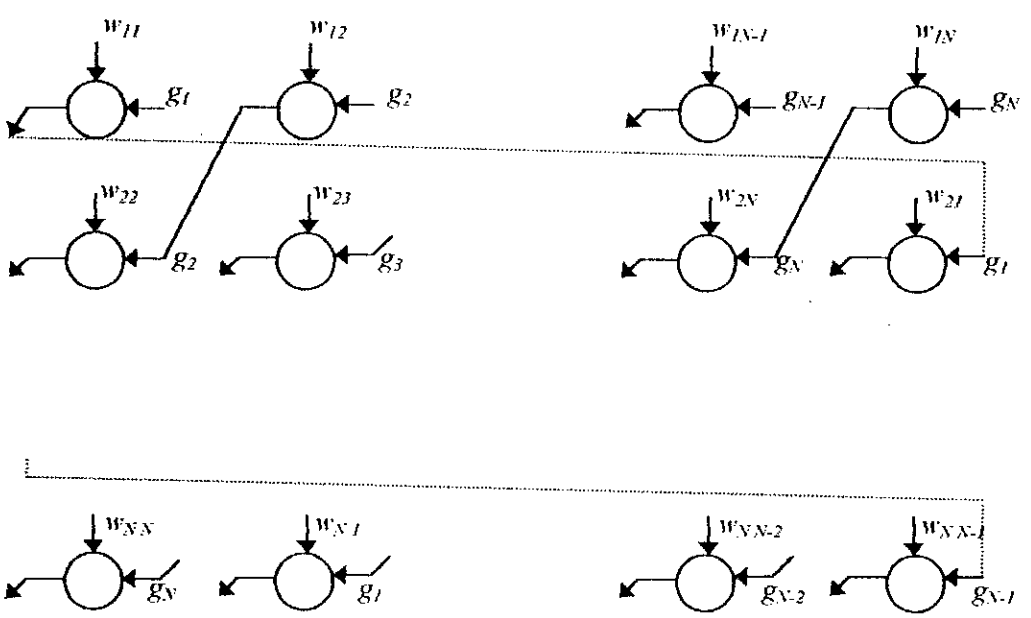


Fig. 2.20. ***DDG pour le calcul de l'erreur δ(l)***

Le graphe donné par la figure suivante représente une projection sur la verticale des poids et une projection sur l'horizontale des valeurs des erreurs δ_i du graphe précédent.

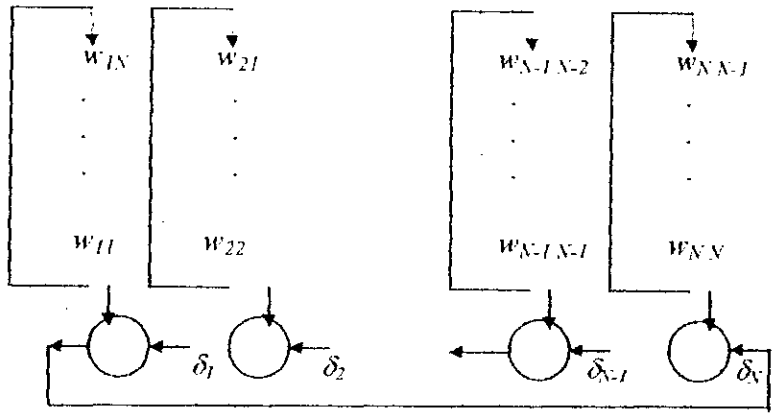


Fig.2.21. ***Projection du DDG de calcul de $\delta_i(l)$.***

Ce qui mène à un réseau systolique en anneau.

«On remarque que l'ordre des poids reste inchangé par rapport à la phase de relaxation, seulement cette fois-ci la valeur calculée de δ_i c'est elle qui circule et accumule les produits correspondants.»

•Mise à jour des poids :

La mise à jour des poids se fait selon l'équation: $w^*_{ij}(l+1) = w_{ij}(l+1) + g_i(l+1).\ h_j(l+1)$ qui peut être décrite par le *DDG* suivant :

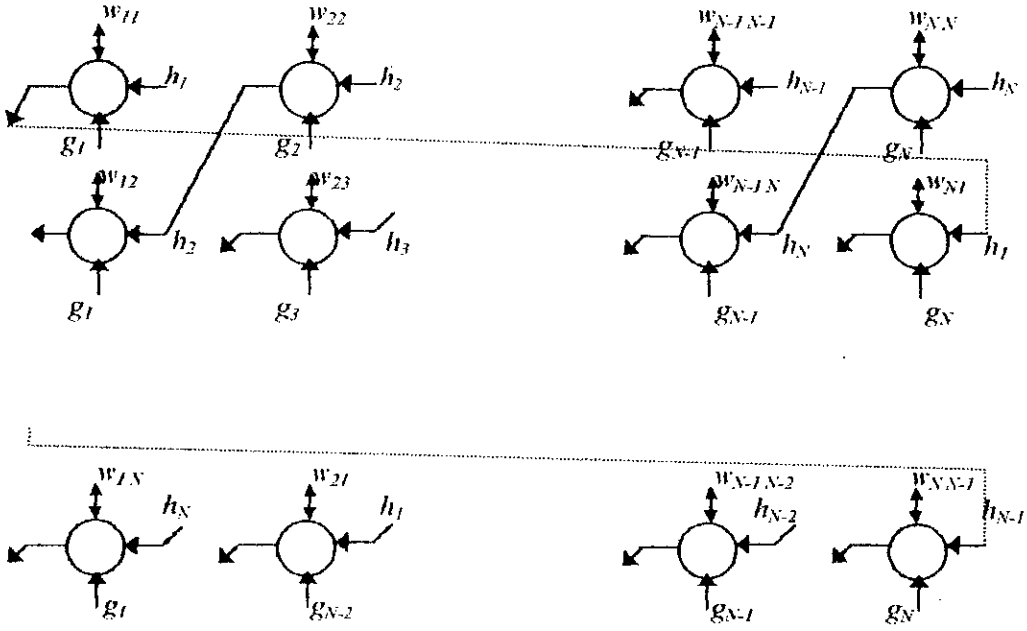


Fig.2.22. ***DDG pour la mise à jour des poids.***

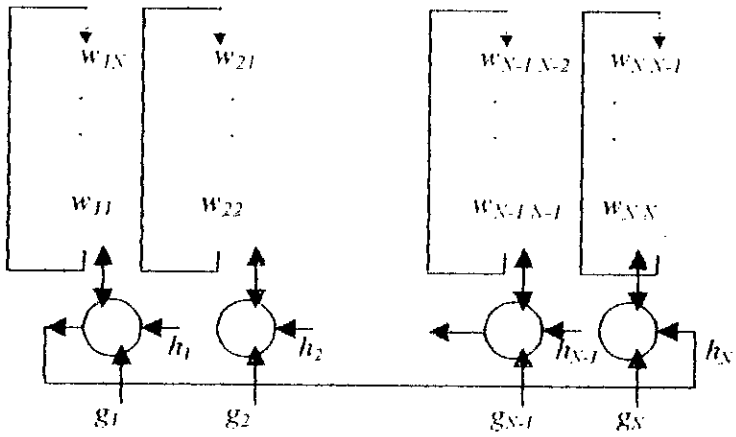


Fig. 2.23. ***Projection du DDG de la mise à jour des poids***

D'après ce qui précède on remarque que la configuration des graphes de dépendances de données est la même pour les trois équations précédentes, seulement le flux de données constituant les entrées des nœuds diffère d'un graphe à un autre ; ce qui conduit à un anneau systolique pouvant exécuter les opérations des deux phases pour une couche donnée d'un réseau de neurones.

L'extension à L couche se fait par la mise de L *DDGs* en cascade, ce qui revient à implémenter des anneaux systoliques en cascade comme il est montré dans la figure 2.24.

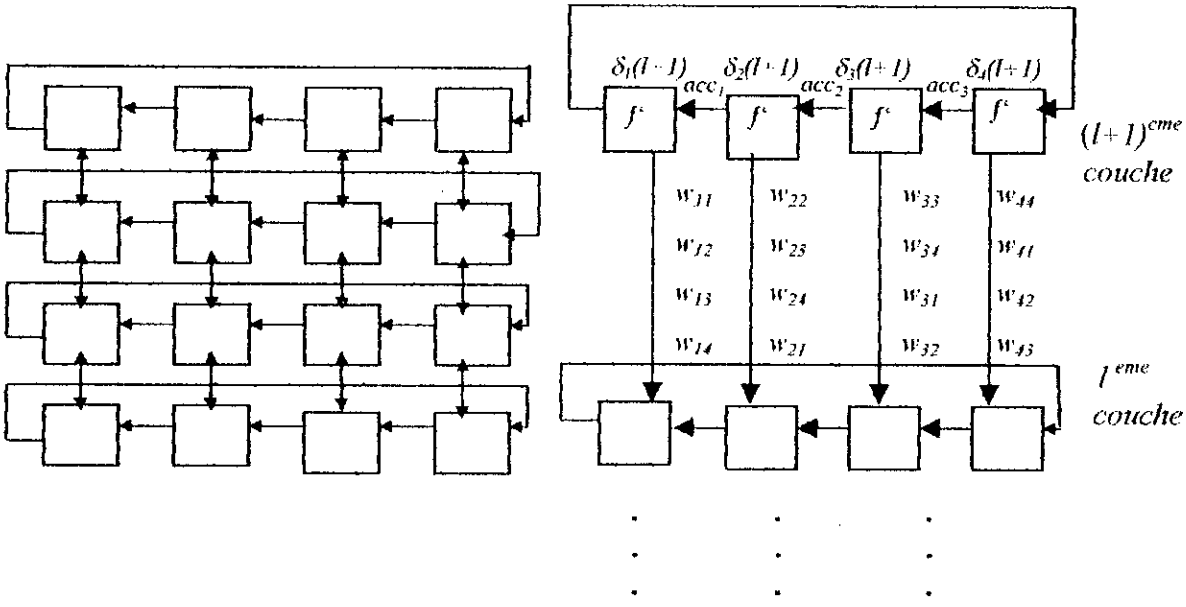


Fig.2.24.a. ***Réseau systolique en cascade pour un réseau multicouches.***

Fig .2 .24.b ***Réseau systolique en cascade pour l'apprentissage de la rétro-propagation***

Comme la sortie de la couche *(l)* constitue l'entrée de la couche *(l+1)*, donc les opérations d'une couche à l'autre se font en séquence et un seul anneau systolique suffit pour effectuer les opérations relatives à toutes les couches du réseau, tout en respectant l'ordonnancement des opérations et le flux de données (donnée correspondante au moment correspondant).

Remarque : Dans le cas d'un traitement en pipeline multicouches, il faudra prendre autant d'anneaux que d'étages en pipeline.

2.4.4.ARCHITECTURE EN ANNEAU SYSTOLIQUE DES RNAs

Nous examinons une architecture *VLSI* systolique en anneau pour l'implémentation d'un réseau de neurones artificiel appliqué à la navigation d'un *RMA(Robot Mobile Autonome)*.

Il est démontré que les réseaux de neurones sont souhaités pour trois types d'applications en robotique :

- la planification d'une tâche,
- la planification d'un chemin,
- Le contrôle d'un chemin.

Pour ces tâches un réseau systolique programmable est développé; Ce réseau peut exploiter les capacités des circuits *VLSI* pour fournir un calcul intensif et pipelinisé sur une couche. Les deux phases de relaxation et d'apprentissage sont intégrées dans la même conception. Ce qui rend l'architecture proposée plus universelle.

Comme nous l'avons vu, les deux phases de relaxation et d'apprentissage de la plus part des modèles des réseaux de neurones peuvent être formulées comme des multiplications matrice-vecteur (*Matrrix-vector Multiplication* : *MVM*) consécutives ou comme des problèmes de mise à jour de produit externe (*Outer Product Updating* : *OPU*). De point de vue structure du réseau, toutes les formulations mènent à la même architecture d'un réseau systolique universel. En terme de fonctionnement elles mènent à des processus de multiplication et accumulation (*Multiplication And Accumulation: MAC*).[10].

Le réseau systolique dérivé pour la phase de relaxation peut être adapté à la phase d'apprentissage pour simplifier l'accès aux poids synaptiques $\{w_{ij}\}$ dans les deux phases.

Le même processeur élémentaire *PE* sera utilisé pour le traitement des deux phases.

2.4.4.1. PROCESSEUR ÉLÉMENTAIRE.

C'est un processeur pouvant exécuter les opérations suivantes:

- Multiplication pour calculer les $\{h_{ij}\}$ où: ($h_j(l-1) \cdot \eta.a_j(l)$),
- multiplication et accumulation: c'est l'opération la plus utilisée dans les deux phases.
- soustraction pour calculer $\delta_i(L) = a_i(L) - t_i$.
- activation de la fonction sigmoïde f_i et sa dérivé f_i' pour chaque neurone i.

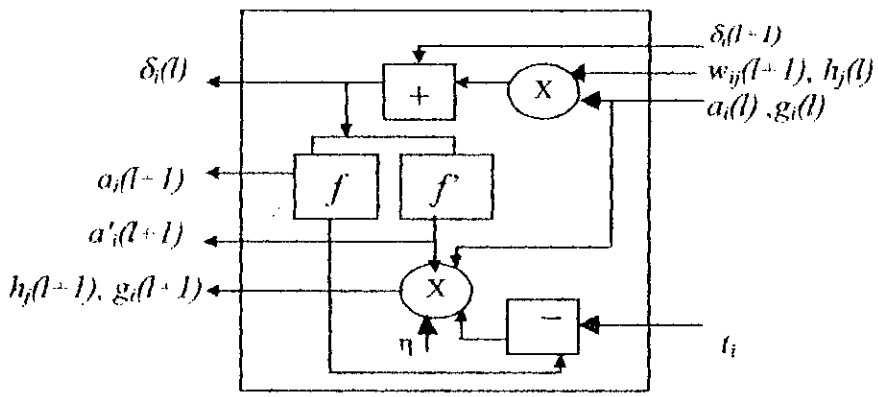


Fig 2.25. ***PE utilisé par l'algorithme RPG.***

Ce processeur est développé pour exécuter les opérations nécessaires dans la phase de relaxation et d'apprentissage pour toute application utilisant l'algorithme *RPG*.

Lors de la phase d'utilisation(fonctionnement), seules les fonctions relatives à la phase de relaxation sont activées pour associer à un vecteur d'entrée un vecteur de sortie correspondant. Les opérations relatives à la phase d'apprentissage(retro-propagation) ne sont pas utilisées dans cette phase.

Remarque : Notre application envisagée (développée dans le chapitre 3) nécessite, en plus de l'algorithme *RPG* à apprentissage supervisé utilisant ce *PE* (fig2.25) comme un processeur de calcul, un autre bloc(bloc d'action) utilisant l'apprentissage renforcé pour la mise à jour de ces poids synaptiques.

Les processeurs utilisés par ce bloc sont développés dans le chapitre suivant après l'explicitation des équations nécessaires pour l'apprentissage renforcé dans l'environnement d'application.

La dérivation d'une architecture systolique pour ce bloc est directe, vu sa simple structure (Ce n'est qu'une phase d'association des sorties des blocs *RPG* aux actions) (voir chapitre 3).

2.4.4.2. ANNEAU SYSTOLIQUE POUR LA PHASE DE RELAXATION

La dynamique du système dans la phase de relaxation peut être formulée comme des *MVMs* suivies par la fonction d'activation non linéaire.

La caractéristique principale des *MVMs* consécutives est que les sorties de chaque couche seront utilisées comme des entrées pour la prochaine couche. Par conséquent, un arrangement attentif d'association des processus et d'ordonnancement "*Schedule*" dans la conception d'architecture conduit à une structure en anneau comme nous l'avons expliqué précédemment.

• **Traitement systolique des *MVMs*** :

Dans la $(l+1)^{eme}$ couche de la phase de relaxation, chaque *PE* peut être traité comme une unité du réseau et les poids $\{w_{i1}, w_{i2}, \ldots\ldots w_{iN}\}$ sont stockés dans la mémoire qui fait un décalage circulaire de (i-1 positions)(voir fig 2.26). Les opérations de la $(l+1)^{eme}$ couche peuvent être décrites comme suit:

1. Chaque valeur d'activation d'une unité, crée au i^{eme} *PE*, est multipliée par w_{ij}. Le produit est ajouté à l'entrée accumulateur $U_i(l+1)$ du réseau, qui a une valeur initiale égale à θ_i pour l'algorithme *RPG*. Après l'opération multiplication-accumulation *MAC*, $a_i(l)$ circule dans le sens des aiguilles d'une montre à travers le réseau en anneau et visite tous les autres *PEs* une fois toutes les N unités d'horloge.
2. Lorsque $a_j(l)$ arrive au i^{eme} *PE*, elle est multipliée par w_{ij} et le produit est ajouté à $U_i(l+1)$.
3. Après N unités d'horloge, l'accumulateur $U_i(l+1)$ collecte tous les produits nécessaires.
4. Plus qu'une unité d'horloge est nécessaire pour que $a_i(l)$ retourne au i^{eme} *PE* et le processeur est prêt à exécuter la fonction non linéaire f_i pour créer $a_i(l+1)$ pour la prochaine couche.

Cette procédure peut être exécutée d'une façon complètement pipelinisée, ainsi elle peut être exécutée récursivement (avec l'accroissement de l jusqu'à ce que L couches soient terminées. La fonction non linéaire utilisée par l'algorithme *RPG* peut être approximée par des segments linéaires de (7 à 15 segments)[14](voir le chapitre 4).

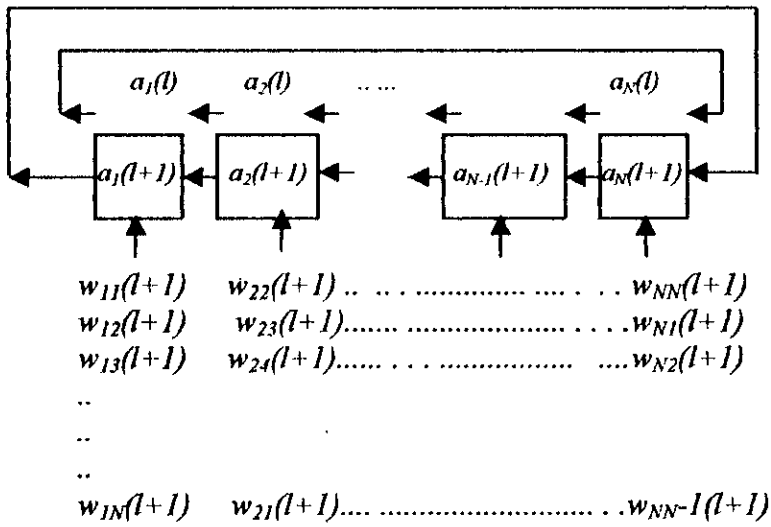


Fig 2.26. ***Architecture en anneau systolique des opérations MVMs pour la phase de relaxation .***

2.4.4.3. ANNEAU SYSTOLIQUE POUR L'APPRENTISSAGE.

Il y' à deux types d'opérations demandées dans la phase d'apprentissage de l'algorithme *RPG* .

1. Les mises à jour des produits externes consécutifs *OPUs* et
2. les multiplications consécutives vecteur-matrice *VMMs* .

Les deux opérations peuvent être implémentées efficacement avec un réseau systolique en anneau basé sur les mêmes configurations (stockage en mémoire et matériel de traitement) que la phase de relaxation:

1. La mise à jour des poids peut être calculée en se basant sur les opérations *OPUs*,
2. Le signal correctif d'erreur de la $l^{ème}$ itération peut être calculé en se basant sur les *VMMs* consécutives.

•Traitement systolique des opérations *OPUs*.

En général, la formule de mise à jour des poids (addition + multiplication) à la $(l+1)^{eme}$ couche dans l'algorithme *RPG* peut être résumée par des opérations *OPUs*.

Pour un réseau de neurones implémenté sur un anneau systolique, ces opérations peuvent être décrites brièvement comme suit:

1. La valeur de $g_i(l+1)$ est calculée au niveau du i^{eme} *PE*. La valeur $h_j(l+1)$ produite au j^{eme} PE sera circulairement "canalisée" à gauche pour tous les *PEs* dans le réseau de neurones en anneau systolique pendant N tops d'horloge.
2. Quand $h_j(l+1)$ arrive au i^{eme} *PE*, elle est multipliée par la valeur mémorisée $g_i(l+1)$, le produit sera ajouté à l'accumulateur du poids w_{ij}, qui est initialement mis à "zéro" à la $l^{ème}$ couche.

"On note que les accumulateur $\{\Delta w_{ij}\}$ sont accédés à des séquences à décalage circulaire, juste comme l'ordonnancement de l'accès à $\{w_{ij}\}$ ".

1. Après N tops d'horloges, tous les NxN poids sont incrémentés. L'anneau systolique est prêt maintenant à accumuler les $OPUs$ de la prochaine couche.

Après toutes les accumulations incrémentales dans L couches des $\{w_{ij}\}$, le réseau est prêt à exécuter la mise à jour locale de chaque PE, ce qui nécessite aussi N unités de temps.

Sachant que la valeur de η est fixée dans l'algorithme de la RPG.

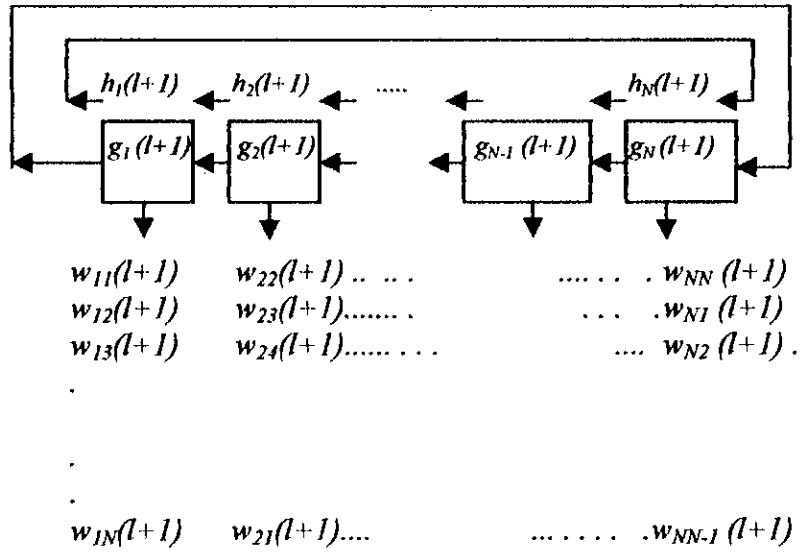


Fig 2.27. ***Architecture en anneau systolique pour l'opération OPU.***

• Traitement systolique des opérations *VMMs*

Le calcul d'erreur retro-propagée pour l'algorithme RPG peut être formulée comme des opérations $VMMs$ consécutives .

Les opérations des $VMMs$ consécutives dans l'anneau systolique sont décrites par les étapes suivantes.

1. Le signal $g_i\ (l+1)$ et la valeur de w_{ij} sont disponibles pour la $(l+1)^{eme}$ itération. La valeur de w_{ij} est, donc, multipliée par $g_i\ (l+1)$ au j^{eme} PE.
2. Le produit est ajouté au paramètre accumulateur acc_i arrivée de nouveau (le paramètre acc_i est initialisé au i^{eme} PE à zéro) et se décale à gauche circulairement à travers le réseau en anneau .
3. Après N opérations d'accumulation, le paramètre acc_i retourne au j^{eme} PE après avoir accumulé tous les produits.

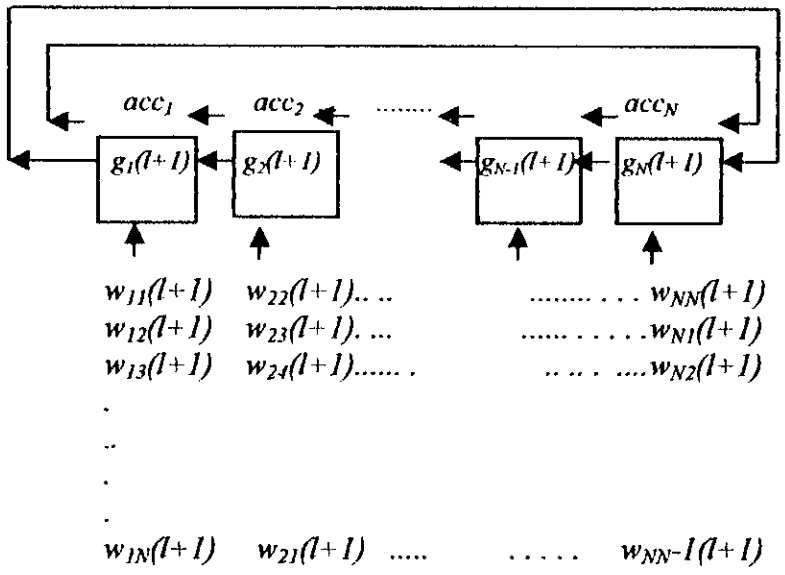


Fig.2.28. ***Architecture en anneau systolique pour les opérations VMMs .***

2.5.CONCLUSION

Dans ce chapitre, nous avons présenté deux approches d'implémentation digitale des réseaux de neurones. Ces deux approches prennent en considération l'implémentaion des deux phases de calcul, phase de relaxation et phase d'apprentissage pour plusieurs modèles de réseaux de neurones.

La première approche est une approche à commutateurs programmables nécessitant une programmation software de ses commutateurs. Par conséquent, elle est jugée non efficace pour notre cas(pour des raisons de simplicité).

La deuxième approche, dite approche systolique ou "réseaux systoliques", est plus universelle et plus efficace pour la navigation. L'application de cette approche à l'algorithme de la retro-propagation du gradient appliqué à la navigation d'un robot mobile autonome est notre objectif dans le chapitre suivant.

Chapitre 3

APPLICATION A LA NAVIGATION D'UN RMA

3.1. INTRODUCTION

Le problème de la navigation d'un robot mobile autonome (*RMA*) dans un environnement avec des obstacles est traité par l'approche des réseaux de neurones.

En utilisant l'algorithme de la retro-propagation du gradient à apprentissage supervisé pour commander un tel robot dans un environnement sur lequel est défini un champ de température crée par la cible.

La tâche du robot consiste à se diriger vers le maximum de température tout en évitant les obstacles. La stratégie pour remonter le champ de température ainsi que le comportement permettant d'éviter les obstacles sont acquis par apprentissage grâce aux propriétés de classification des réseaux de neurones [16].

Notre objectif dans ce chapitre est d'implémenter cet algorithme sous forme de réseaux systoliques pouvant effectuer toutes les opérations de relaxation et d'apprentissage.

3.2. STRUCTURE GENERALE

A chaque étape le *RMA* reçoit les informations sur la distance aux obstacles et les températures de la cible de tous les cotés, et doit prendre une décision concernant la direction du prochain mouvement. La recherche du maximum de température peut être interprétée comme le but du robot, quant à l'évitement d'obstacles il peut être regardé comme un "reflex" du système[17].

Les actions prises par le robot peuvent être résumées en trois actions :

- un pas en avant,
- tourner à droite et un pas en avant,
- tourner à gauche et un pas en avant.

La structure générale du système est représentée par la figure.3.1.

Dans le premier bloc, appelé MAP, constitue une application entre les espaces des entrées *Xs*(distances aux obstacles*)*, *Xt* (champs de température de tous les cotés du robot*)* et les valeurs de sortie *S* et *T* respectivement[17](voir les détailles dans 3.3).

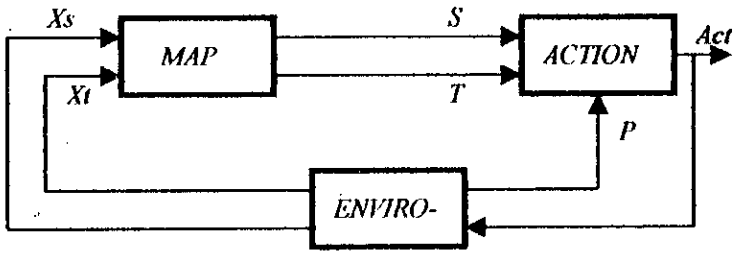


Fig.3.1. ***Structure générale***

Ce bloc (bloc *MAP*) joue le rôle d'un classificateur, ayant comme structure principale le réseau de classification utilisant l'algorithme *RPG* à apprentissage supervisé.

Pour cela deux réseaux de classification sont nécessaires pour construire ce bloc, un pour la classification des champs de température et l'autre pour la classification des situations spatiales.

Le deuxième bloc (bloc d'action) ne fait qu'associer chaque vecteur de sortie produit par le classificateur à un vecteur *Act*, caractérisant l'action à prendre par le *RMA*. Cette association est effectuée par "essai et erreur". Comme il est connu dans le champ des réseaux de neurones, l'apprentissage par essai et erreur est guidé seulement par un processus de retour c'est-à-dire : par le signal *P* fourni par l'environnement. Ce signal renforce l'association entre une situation donnée et une action si cette action mène à une conséquence favorable pour le robot. Autrement le signal *P* cause une dissociation [16][17].

Pendant la phase d'utilisation (fonctionnement) la structure des blocs reste inchangée, seulement les fonctions relatives à l'apprentissage, implémentées dans les blocs, ne sont pas utilisées.

3.3. SYNOPTIQUE DU SYSTEME GLOBAL

Le système global peut être développé en trois phases (voir fig.3.2):

Les deux phases, phase 1 et phase 2, sont deux blocs structurellement identiques recevant en parallèle les vecteurs *Xs* et *Xt*.

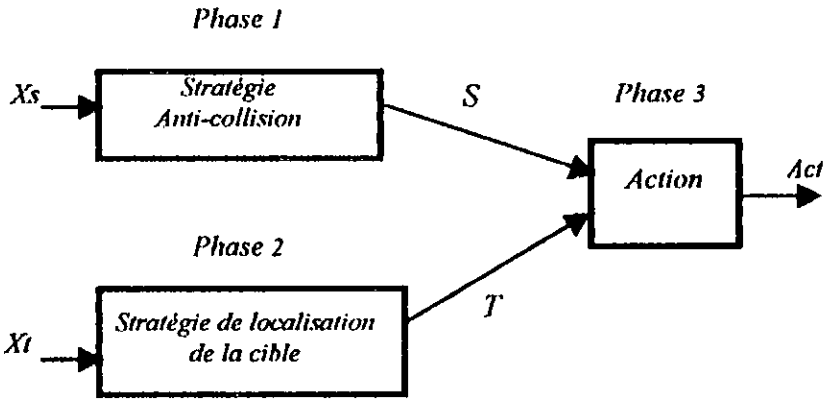


Fig.3.2. ***Synoptique du système global***

Xs fournit les informations sur les distances aux obstacles en avant, à droite et à gauche du *RMA*.

Xt représente le champ de température de la cible.

Les deux vecteurs *S* et *T*, produits par les deux phases constituent les entrées du bloc d'action.

Les deux premières phases doivent constituer deux apprentissages différents.

La première fait apprendre au robot six (06) situations relatives à un chemin qui permet d'éviter les obstacles, alors que la deuxième consiste à lui faire apprendre six (06) autres situations relatives à la localisation de la cible grâce à sa température.

La troisième phase doit décider d'une action grâce à l'association et la coordination des résultats des deux apprentissages précédents[17].

a. *Phase1.*

La tâche du *RMA* consiste à prendre un chemin libre d'obstacles. Pour se faire, le *RMA* doit tout d'abord détecter l'existence des obstacles en avant à sa gauche et à sa droite. Le *RMA* doit connaître à chaque étape dans quelle situation se trouve.

La méthode d'évitement des obstacles utilisée est une stratégie anti-collision devant être acquise par apprentissage, grâce aux propriétés de classification et de mémorisation des réseaux de neurones Elle est appliquée, ici, à un *RMA* évoluant dans un environnement partiellement structuré.

Le vecteur *Xs*: est un vecteur d'entrée ayant les composantes suivantes:

Da : *distance à l'obstacle se trouvant en avant.*

Dd *: distance à l'obstacle se trouvant à droite.*

Dg *: distance à l'obstacle se trouvant à gauche.*

Ces distances sont normalisées entre [0, 1].

3.4. DEFINITION ET MODELISATION DE L'ENVIRONNEMENT

3.4.1. MODELISATION DE L'ENVIRONNEMENT SPATIAL

L'environnement d'application doit être partiellement structuré dans lequel les obstacles éventuels ont une forme rectangulaire ou d'assemblage rectangulaire.

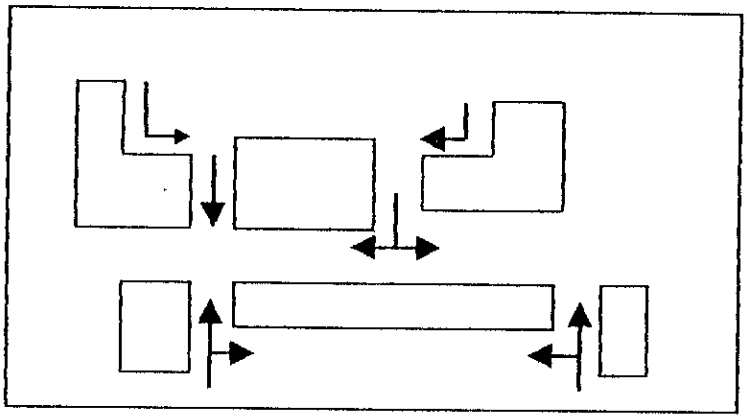


Fig.3.3. ***Modélisation de l'environnement.***

A partir d'un tel environnement et en tenant compte des différents mouvements possibles du robot, les six (6) situations représentatives englobant presque tous les cas possibles sont dégagés comme suit[19]:

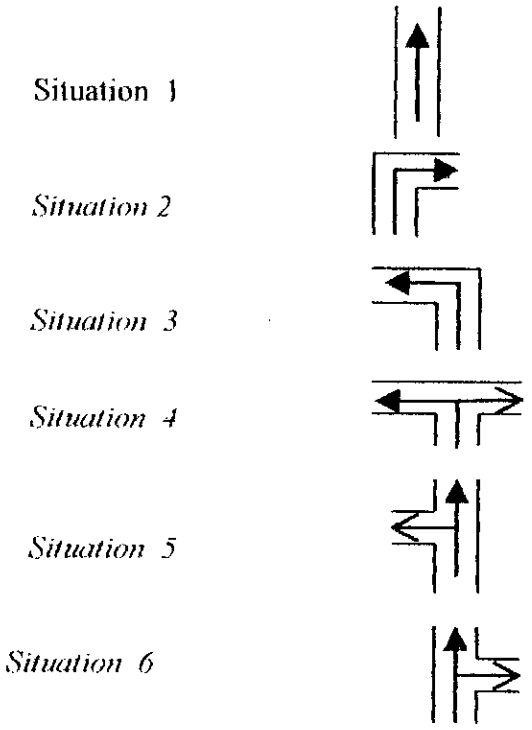


Fig.3.4. Les différentes situations d'anti-collision.

3.4.2. MODELISATION DU ROBOT

Le *RMA* est supposé évoluer à chaque instant (période d'échantillonnage) avec une distance fixe. Il est géométriquement modélisé par un carré.(petit carré, voir le dernier chapitre).

En réalité le *RMA* est doté d'un certain nombre de capteurs à ultrasons qui se disposent sur son devant, sur sa droite et sur sa gauche.

Ces emplacements des capteurs sont imposés par les trois mouvement possibles du robot[17].

3.4.3. DEFINITION DES CHAMPS DE TEMPERATURE

Les différents champs d'orientations de la cible par rapport au *RMA* sont décrits et choisis comme dans la figure.3.5.

Six situations peuvent être détectées par le robot, chacune d'elle correspond à une orientation particulière de la cible[19].

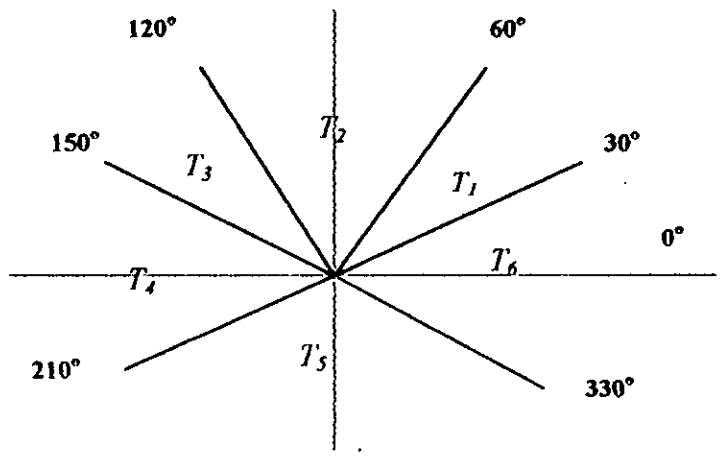


Fig .3.5. Situations relatives au champ de température.

La situation T_1 correspond à l'angle compris entre [30° et 60°].

La situation T_2 correspond à l'angle compris entre [60° et 120°].

La situation T_3 correspond à l'angle compris entre [120° et 150°].

La situation T_4 correspond à l'angle compris entre [150° et 210°].

La situation T_5 correspond à l'angle compris entre [210° et 330°].

La situation T_6 correspond à l'angle compris entre [330° et 30°].

Le vecteur *Xt* est vecteur composé de trois éléments suivants:

Ta *: température en avant.*

Td *: température à droite.*

Tg *: température à gauche.*

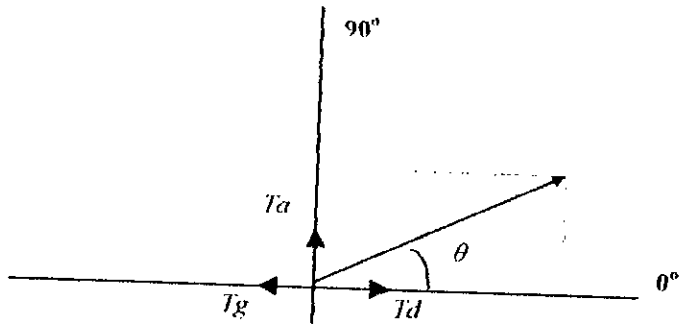


Fig 3.6. ***Composantes du champ de température***

Le vecteur Xi est normalisé entre [0, 1].

Pour le calcul de ce vecteur, quatre cas sont possibles, à savoir :

1^{eme} *cas :* $0° \leq \theta \leq 90°$

2^{eme} *cas :* $90° \leq \theta \leq 180°$

3^{eme} *cas :* $180° \leq \theta \leq 270°$

4^{eme} *cas :* $270° \leq \theta \leq 360°$

θ *est l'angle d'orientation de la cible par rapport au robot .*

Le calcul se fait de la même manière dans les différents cas.

Là aussi, à chaque fois qu'on applique au réseau après apprentissage, un vecteur d'entrée Xi nous obtenons un vecteur de sortie T.

1^{eme}	*cas:*	$Ta = sin(\theta)$,	$Td = cos(\theta)$,	$Tg = 0$.
2^{eme}	*cas:*	$Ta = sin(\theta)$,	$Td = 0$,	$Tg = -cos(\theta)$.
3^{eme}	*cas:*	$Ta = 0$,	$Td = 0$,	$Tg = -cos(\theta)$.
4^{eme}	*cas:*	$Ta = 0$,	$Td = cos(\theta)$,	$Tg = 0$.

3.4.4. STRATEGIE ANTI-COLLISION.

Ici, l'association entre les situations spatiales et les actions appropriées est effectuée par un *RNA* à deux couches 6 et 3 neurones sachant qu'il y' à 6 situations spatiales (voir 3.4.1) et 3 actions (3.4.3). L'information de température n'est pas prise en considération ($T = 0$)[16][17].

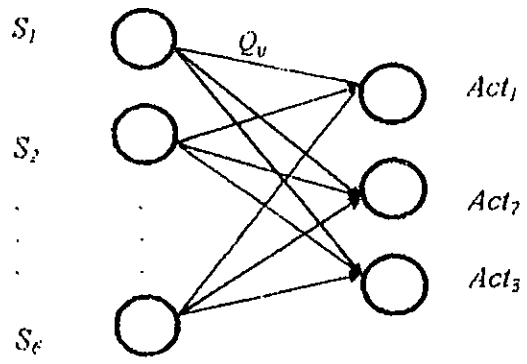


Fig. 3.7. Bloc d'action anti-collision.

Chaque neurone Act_i est connecté à tous les neurones S_j à travers des connections pondérées par des coefficients adaptatifs Q_{ij}.

$$\frac{\tau . dQ_{ij}}{dt} = Act_i . S_j . (-Q_{ij} + \alpha - P) \qquad (3.1)$$

$$P = \begin{cases} P_1 & si\ collision \\ 0 & si\ non \end{cases}$$

et la sortie Act_i est donnée par :

$$Act_i = G(\sum_j Q_{ij} . S_j) + N_i \qquad (3.2)$$

Où N_i est une variable aléatoire avec une distribution uniforme entre [0, β], il permet de favoriser une action par rapport à l'autre [16].

$$G(x) = \begin{cases} x & si\ x > 0. \\ 0 & si\ non. \end{cases} \qquad (3.3)$$

Les constantes α, β et P_1 sont couplées comme suit $0 < \beta < \alpha < P_1$

A chaque étape le maximum des A_i est mis à 1, tant que les autres sont mis à zéro. Le signal P peut être regardé comme un signal de punition.

La solution de l'équation différentielle du premier ordre (equation.*3.1*) est :

$$Q_{ij} = \alpha . (1 - e^{-Act_i . S_j . t / \tau}) - P_1 \qquad (3.4)$$

S_j est active ($S_j = 1$) et l'action A_i ne provoque pas une collision c'est-à-dire ($P = 0$), avec l'évolution du temps Q_{ij} se sature à α, ce qui renforce la liaison (A_i, S_j).

Si l'action Act_i provoque une collision, (c'est-à-dire $P = P_1$) Q_{ij} se sature à ($\alpha - P < 0$) avec la même constante du temps. Donc Q_{ij} devient négative dissociant S_j de Act_i.

Pour chaque situation S_j on fait la même procédure avec les trois actions possibles.

Les connections donnant une action favorable sont renforcées, par contre les connections donnant une action défavorable sont dissociées et sont saturées à des valeurs négatives[17].

3.4.5. STRATEGIE DE LOCALISATION DE LA CIBLE.

L'association entre les différentes situations du champ de température et les actions appropriées afin d'atteindre le maximum est effectuée dans un environnement libre d'obstacles (S=0); en utilisant le bloc d'action avec le vecteur T.

Le mécanisme d'association est le même que celui décrit précédemment, à l'exception du signal P qui est défini comme suit[16][17]:

$$P = \begin{cases} P_2 & si\ Z = 0 \\ 1 & si\ non. \end{cases}$$

Z: est définie comme variation de température(voir fig.3.9).

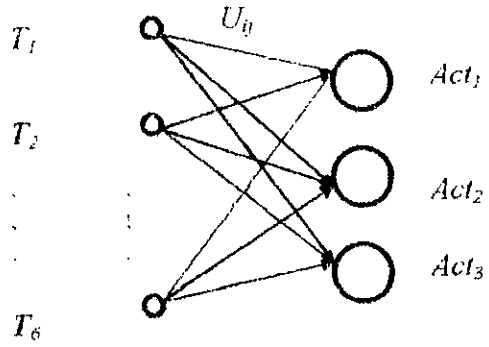


Fig.3.8. Bloc d'action du champ de température

Chaque neurone Act_i est connecté à tous les neurones T_j à travers des connections pondérées par des coefficients adaptatifs U_{ij}.

$$\frac{\tau.dU_{ij}}{dt} = Act_i.T_j.(-U_{ij} + \alpha - P) \qquad (3.5)$$

$$P = \begin{cases} P_2 & si\ Z = 0. \\ 1 & si\ non. \end{cases}$$

et la sortie Act_i est donnée par :

$$Act_i = G(\sum_j U_{ij}.T_j) + N_i \qquad (3.6)$$

La solution de l'équation différentielle du premier ordre (*eq 3.5*) est :

$$U_{ij} = \alpha.(1 - e^{-Act_i.T_j.1/\tau}) - P_2 \qquad (3.7)$$

L'élément Z est défini comme la variation de la température mesurée au niveau du *RMA* entre deux itérations successives. Le signal de punition a lieu lorsque le mouvement du *RMA* n'est pas dans la direction du gradient de température locale.[18]

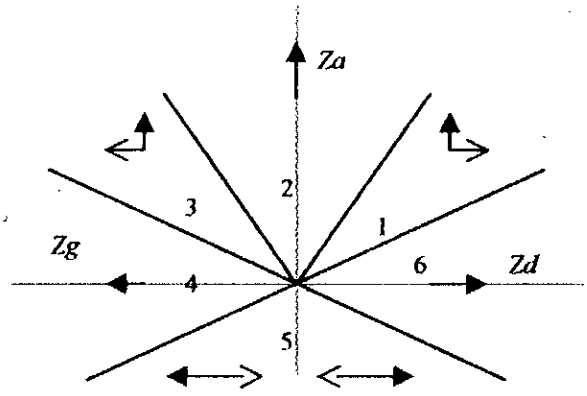


Fig 3.9. ***Les Différents cas de l'élément Z.***

pour le cas de T_1 : $Za = 1,\ Zd = 1,\ Zg = 0.$

pour le cas de T_2 : $Za = 1,\ Zd = 0,\ Zg = 0.$

pour le cas de T_3 : $Za = 1,\ Zd = 0,\ Zg = 1.$

pour le cas de T_4 : $Za = 0,\ Zd = 0,\ Zg = 1,$

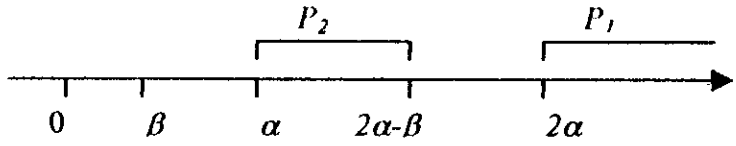
pour le cas de T_5 : $Za = 0,\ Zd = 1,\ Zg = 1.$

pour le cas de T_6 : $Za = 0,\ Zd = 1,\ Zg = 1.$

3.4.6. COORDINATION DES TACHES.

A ce niveau le robot est capable d'accomplir deux taches indépendantes, exécute la navigation libre d'obstacles et cherche le maximum du champ de température dans un environnement libre d'obstacles.

Comme nous avons dit avant, les actions générées par le contexte spatial relatives à la présence d'obstacles peuvent être interprétées comme le reflex du *RMA*, elles doivent avoir prudence plus que les actions générées pour atteindre le maximum du champ de température. Ceci peut être fait en fixant la constante P_1 plus grande que P_2 ; ou en autres mots: la punition causée par une collision est plus sévère que celle occasionnée par un mauvais choix de la direction de température[16][17].



" Les valeurs retenues dans l'application sont explicitées dans le chapitre suivant (voir 4.6.2)"

La sortie de chaque neurone Act_i est donnée par l'expression

$$Act_i = G(\sum_j Q_{ij}.S_j + U_{ij}.T_j) + N_i . \qquad (3.8)$$

Ce qui traduit les quatre possibilités suivantes:

$Act_i = (\alpha.S_j + \alpha.T_j)$: cas de non collision et orientation vers la cible.

$Act_i = (\alpha.S_j + (\alpha-P_2).T_j)$: cas de non collision et non orientation vers la cible.

$Act_i = ((\alpha - P_1).S_j + \alpha.T_j)$: cas de collision et orientation vers la cible.

$Act_i = ((\alpha - P_1).S_j + (\alpha - P_2).T_j)$. : cas de collision et non orientation vers la cible.

3.4.7. ARCHITECTURE NEURALE DU SYSTEME.

Parler, généralement, d'un système d'architecture générale est de chercher à maximiser les performances suivantes[11]:

- Une configuration d'un réseau effective, flexible, programmable.
- Dans le cas des réseaux de grandes tailles avec un petit nombre de *PEs* disponibles, Il est important de fournir un support matériel et / ou un support logiciel pour un arrangement de partitionnement efficace, qui permet de décomposer un grand nombre de problèmes à des petits sous problèmes (qui peuvent être résolus par le réseau).
- Les tâches de plusieurs réseaux de neurones peuvent être associées à partager le même *PE* sans changer la stratégie de calcul / communication.

En se basant sur cette approche, l'architecture peut être adaptée facilement à des réseaux multicouches non uniformes.

Avec une association appropriée des tâches de plusieurs neurones, la charge de calcul pour chaque couche peut être distribuée uniformément sur *N PEs*. Ce qui garantit une efficacité complète de pipelinisme.

Pour la commande du *RMA*, nous avons utilisé la configuration du réseau de neurones suivante(fig 3.10).

Les deux réseaux de classification sont deux réseaux identiques.

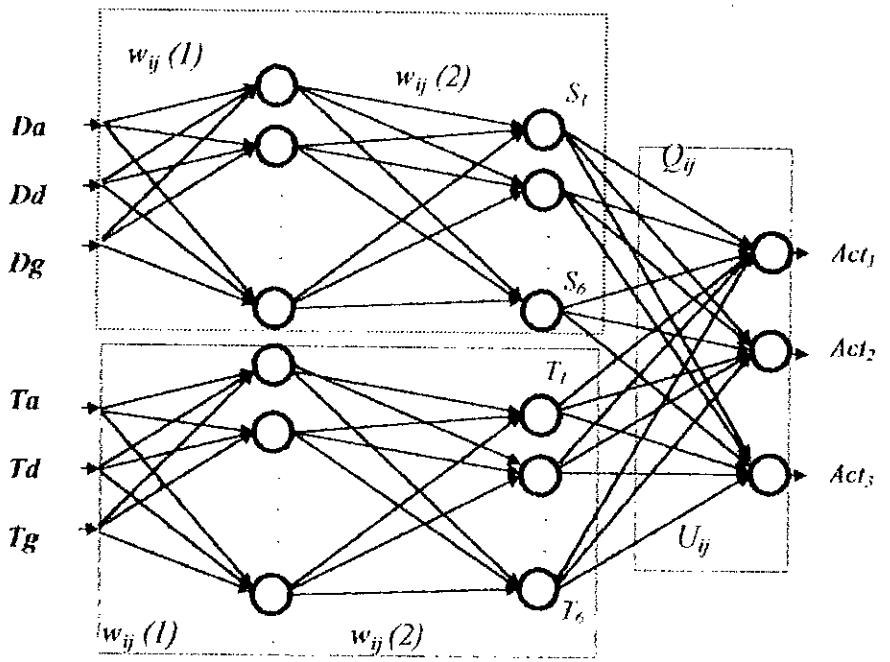


Fig 3.10 . ***Structure du réseau appliqué à la navigation d'un RMA***

Sans perte de généralité, le réseau donné par la figure 3.11. représente une configuration du réseau de classification utilisé. Il illustre le partitionnement d'un réseau non uniforme.

Le choix de cette configuration devait répondre à deux critères :

1. Le nombre de neurones dans la couche de sortie et le nombre de neurones dans la couche d'entrée sont imposés par l'application ; les trois neurones de la couche d'entrée sont dus aux trois actions possibles du robot et les six neurones de la couche de sortie pour la classification des six situations.
2. Le nombre de neurones dans la couche cachée doit, d'une part, permettre d'atteindre une bonne précision de calcul ; d'autre part, il est un multiple de trois pour avoir une structure régulière qui facilite l'implémentation.(traitement par lot de 3).

Avec une configuration (3-12-6), 3 entrées, 12 neurones cachés et 6 neurones de sortie , si le nombre de neurones n'est pas choisi égal à un multiple de 3, il est toujours possible par l'addition d'un nombre d'unités neurales (non-opérations: les poids connectés à ces unités sont mis à zéro) pour égaliser la taille.

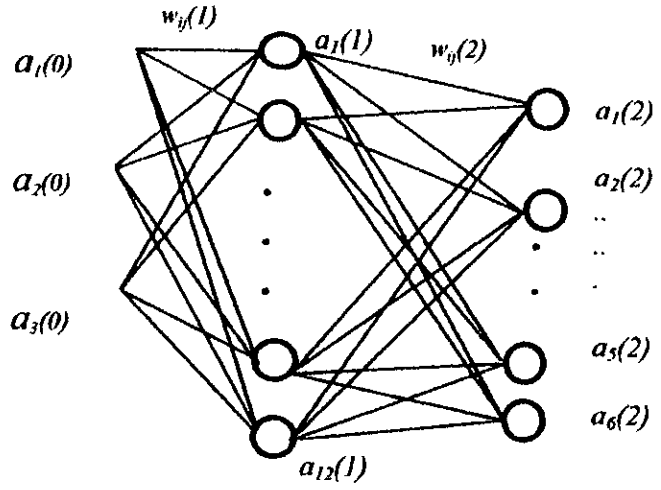


Fig 3.11. ***Réseau classificateur à 2 couches avec une configuration de (3-12-6).***

3.5. IMPLÉMENTATION SYSTOLIQUE DU SYSTÈME NEURAL.

3.5.1. ALLOCATION DES RESSOURCES.

La synthèse architecturale est la génération de description structurelle d'un modèle du niveau architectural et du temps d'exécution des opérations. Elle a comme objectifs[20]:

- Identifier les ressources hardwares qui peuvent implémenter l'opération.
- Ordonnancer le temps d'exécution des opérations (séquensement temporel des opérations).
- Lier les opérations aux ressources électroniques.

Pour notre application trois *PEs* peuvent effectuer un traitement par lot de trois pour les deux phases. Ces processeurs exécutent des opérations de multiplication, multiplication et accumulation, soustraction, la fonction sigmoïde et sa dérivée. Dans chaque étape les trois processeurs exécutent en parallèle la même opération avec des données différentes. L'ordonnancement des opérations et le flux de données (E/S) de chaque opérateur ainsi que le nombre de fois nécessaire pour exécuter l'algorithme *RPG* «propagation et retro-propagation » sont résumés par le graphe de séquensement suivant.

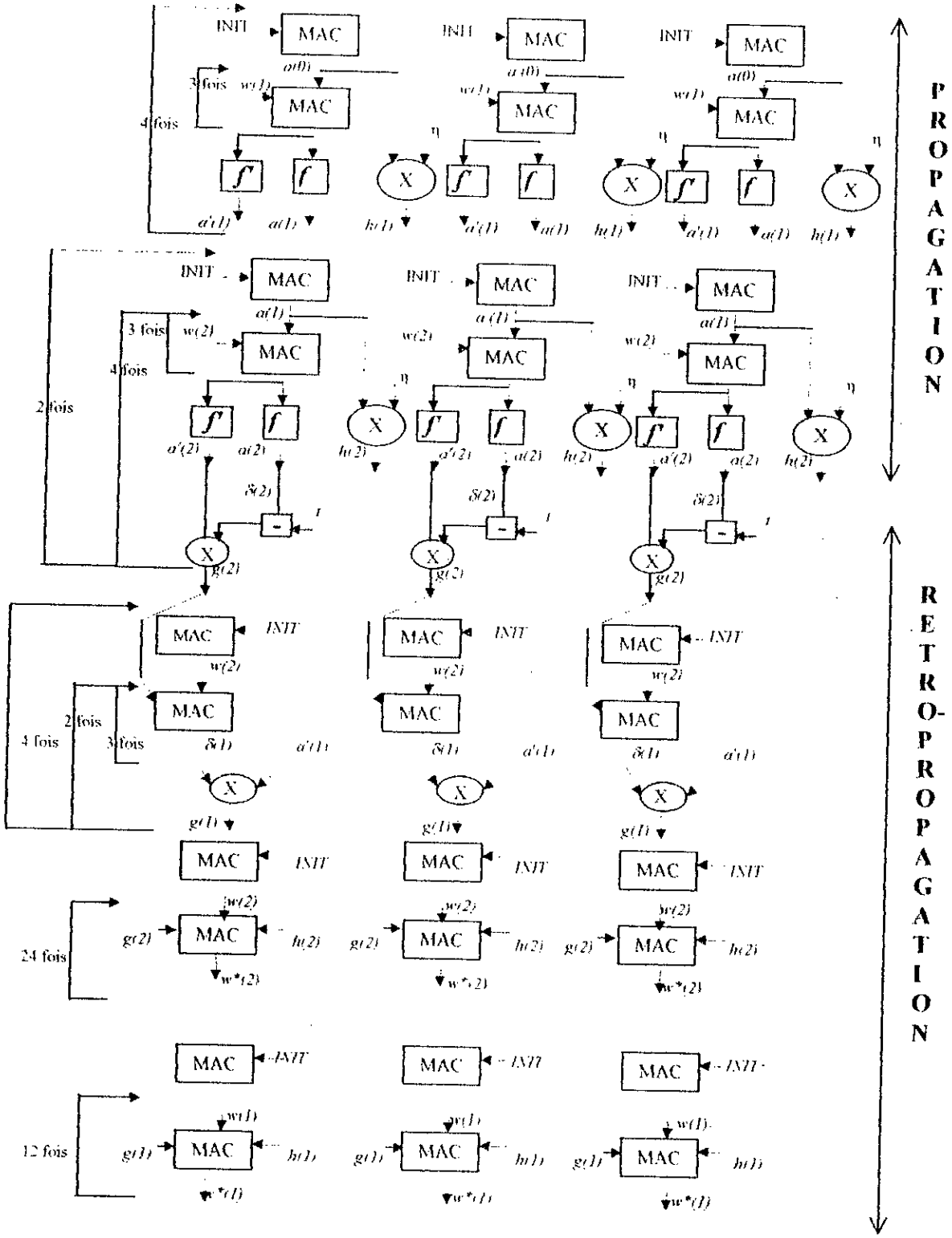


Fig 3.12 ***Graphe de séquensement des opérations.***

3.5.2. MAPPING DE L'ALGORITHME RPG.

3.5.2.1. MAPPING DE LA PHASE RELAXATION.

Les opérations *MVMs* consécutives peuvent être partitionnées en se basant sur l'ordonnancement globalement séquentiel localement parallèle (*LPGS*)[10](voir fig 3.12). Les tâches des 12 neurones de la première couche cachée sont distribuées uniformément sur les 3 *PEs* qui forment un pipeline de 3 étages et les poids associés sont arrangés a priori en avance. Au lieu de tourner une fois pour générer les 12 activations dans le réseau en anneau, les 3 entrées $\{a_i(0)\}$ vont tourner 4 fois pour générer les 12 activations $\{a_i(1)\}$. c'est-à-dire $\{a_i(0),\ i = 1...3\}$ et $\{a_i(1),\ i = 1...12\}$. Les 12 $\{a_i(1)\}$ résultants circuleront dans l'anneau séquentiellement (avec un lot de 3) huit fois (pour générer les 6 valeurs d'activations $\{a_i(2)\}$ de la couche de sortie. Les quatre premiers temps pour générer les 3 Les premières valeurs d'activations$\{a_i(2),\ i = 1...3\}$ et les quatre deuxièmes temps pour générer les trois deuxièmes valeurs de sortie $\{ai(2),\ i = 4....6\}$.

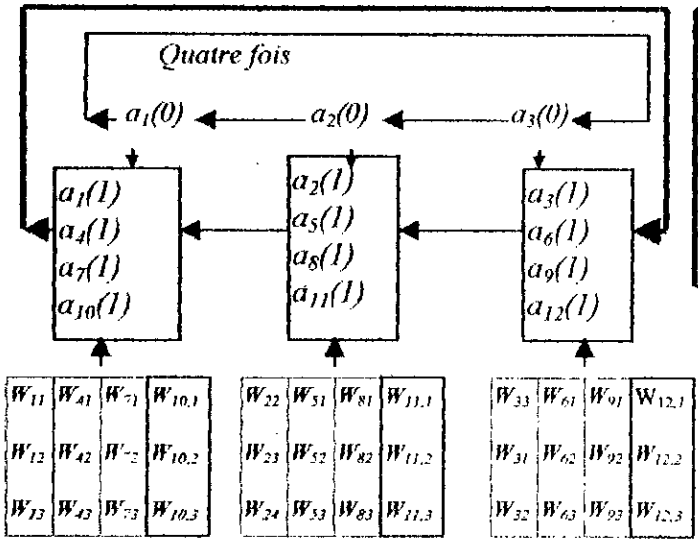


Fig 3.12.a. ***Les opérations MVMs dans la première couche(cachée).***

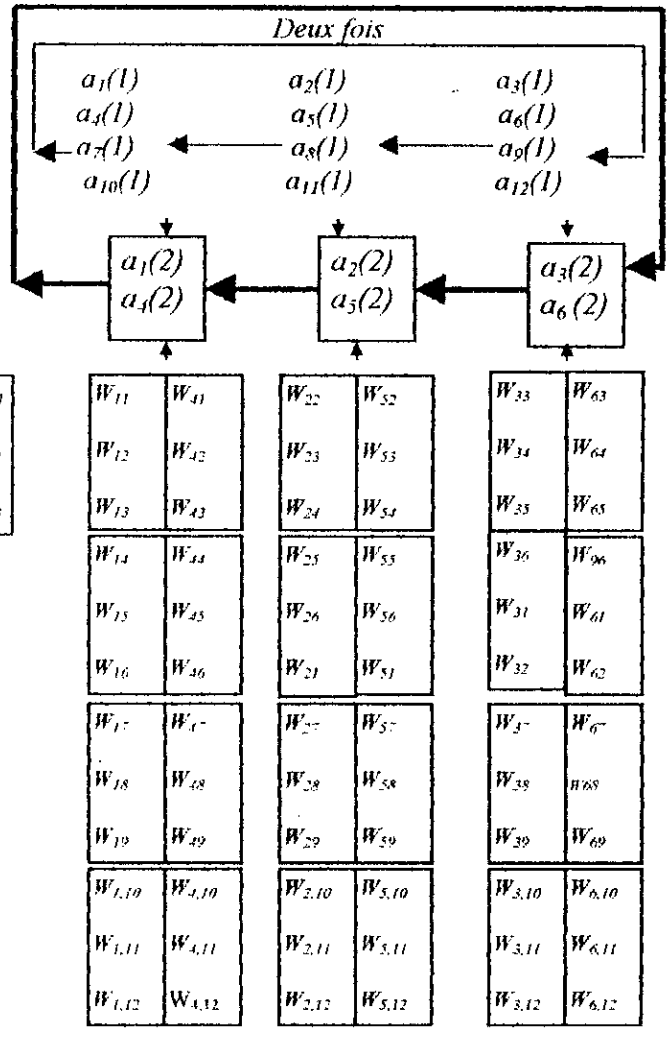
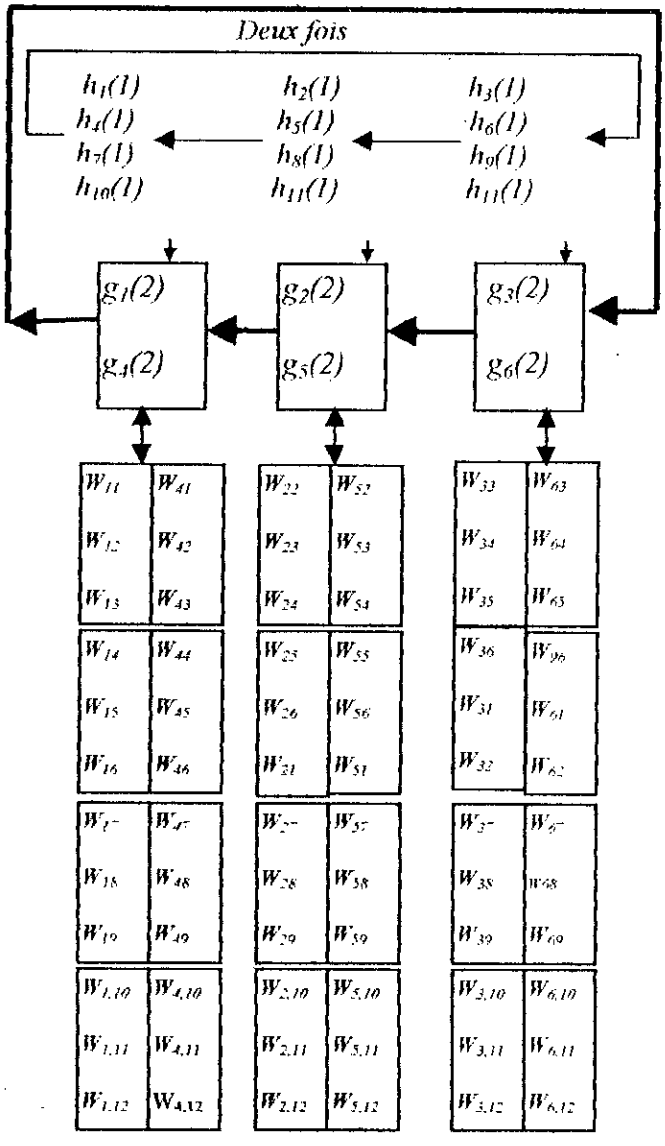


Fig 3.12.b. ***Les opérations MVMs dans la deuxième couche (sortie)***

3.5.2.2. MAPPING DE LA PHASE D'APPRENTISSAGE

3.5.2.2.1. ANNEAU SYSTOLIQUE POUR LES OPUs

Les opérations des *OPUs* d'un réseau multicouches non uniforme peut être implementées dans l'anneau systolique, prenons notre réseau de (3-12-6), en connaissant que les 6 $\{gi(2)=\delta i(2).f'(2)\}$ et les 12 $\{hj(2) = \eta.aj(1)\}$ sont chargés dans les *PEs* correspondants, de même pour les 12 $\{gi(1) = \delta i(1).f'(1)\}$ et les 3 $\{hj(1) = \eta.aj(0)\}$. Les opérations peuvent être décrites brièvement comme suit :



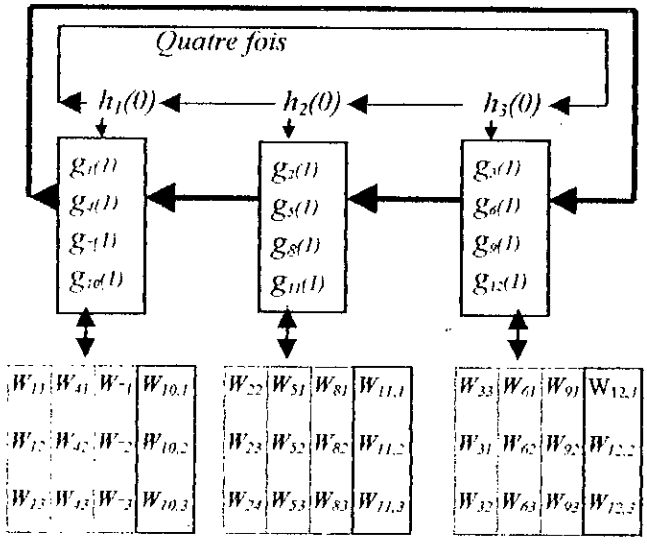


Fig 3.13.b. ***Les opérations OPUs de la première couche (cachée).***

Fig 3.13.a.. ***Les opérations OPUs dans la deuxième couche (sortie).***

1. Les 12 $\{h_j(2)\}$ circulent dans le réseau en anneau dans quatre lots séquentiels $\{h_j(2), j=1.....3\}$ $\{hj(2), j=4.....6\}$, $\{h_j(2), j=7.....9\}$et$\{h_j(2), j=10.....12\}$, pour mettre à jour en premier les poids associés aux 3 neurones dans la couche de sortie (la deuxième couche) $\{w_{ij}(2), i=1.....3, j=1...12\}$. Donc ces quatre lots séquentiels tournent aussi dans l'anneau pour mettre à jour les poids associés aux 3 neurones dans la couche de sortie $\{w_{ij}(2), i=4....6, j=1....12\}$
2. Les mêmes 3 $\{h_j(1)\}$ tourneront dans le réseau en anneau 4 fois pour mettre à jour les poids associés avec les 12 neurones de la couche cachée. Chaque cycle de $\{h_i(1)\}$ mettra à jour les poids associés aux 3 neurones dans la couche cachée. c'est-à-dire:

$\{w_{ij}(1), i=1...3, j=1...3\}$, $\{wij(1), i=4...6, j=1...3\}$,

$\{w_{ij}(1), i=7...9, j=1...3\}$, $w_{ij}(1), i=10...12, j=1...3\}$

3.5.2.2.2. ANNEAU SYSTOLIQUE POUR LES VMMs

Quand les opérations d'*OPUs* sont exécutées, les *VMMs* consécutives s'exécutent pour calculer $\{\delta i(1)\}$. Les opérations peuvent être décrites brièvement comme suit:*Fig 3.14.*

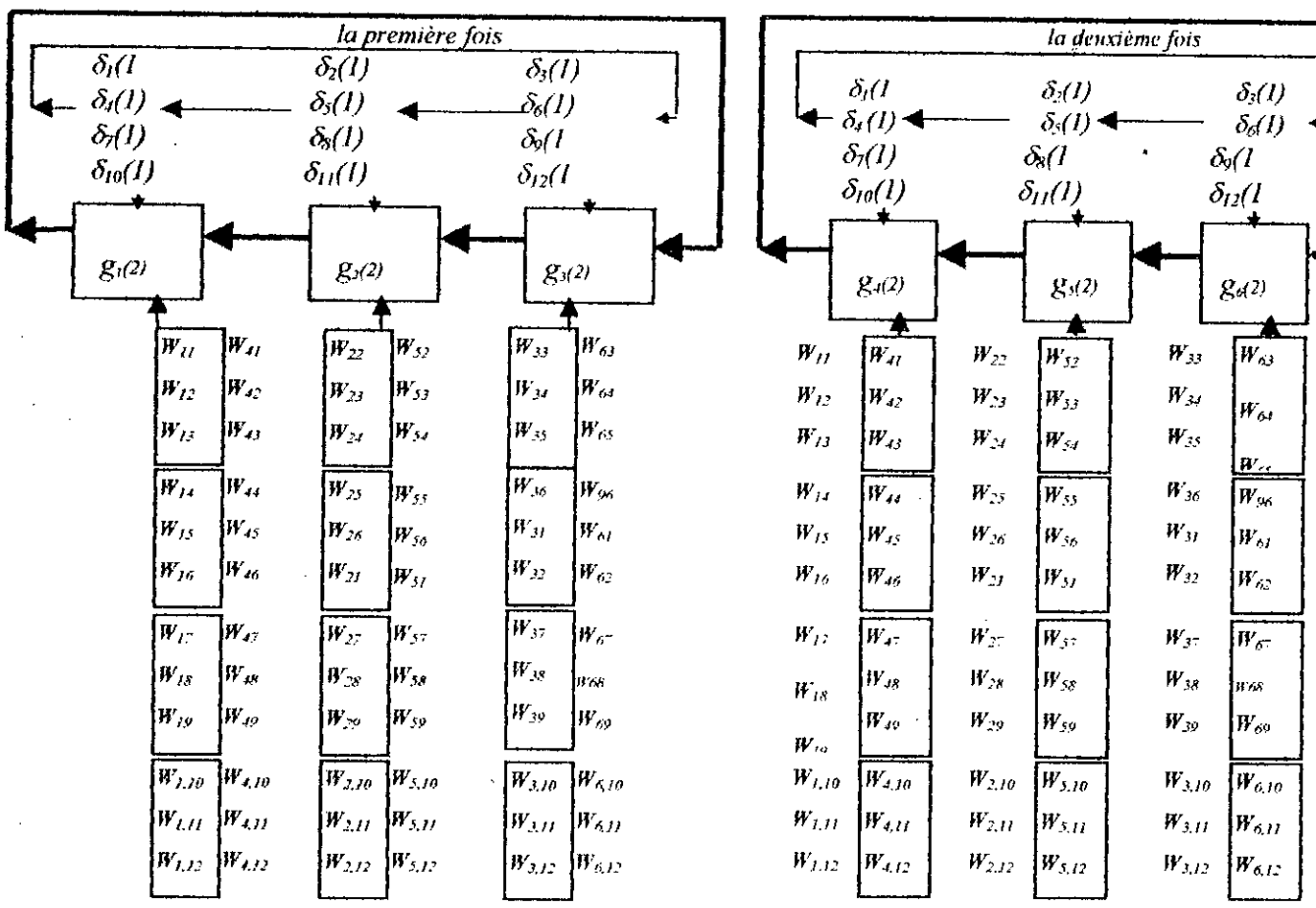


Fig 3.14 L'accumulation du signal correctif d'erreur retro-propagée.

1. Avec la circulation de chaque $h_i(2)$ dans le réseau en anneau pour l'opération *OPU*, il y' a aussi δi(1) tournant dans le réseau en anneau dans quatre séries séquentielles $\{\delta_i(1),\ i=1....3\}$, $\{\delta_i(1),\ i=4....6\}$, $\{\delta_i(1), i=7.....9\}$ et $\{\delta_i(1),\ i=9.....12\}$ pour accumuler les premiers signaux correctifs de la retro-propagation , c'est-à-dire au même moment que la mise à jour de tous les poids associés avec les 6 premiers neurones dans la couche de la sortie(voir la figure .3.14.*a*) et $\delta_i = \sum g_i(2).w_{ij}(2) \quad i = 1,2,3$.
2. Quand la deuxième -3 fois de cycles- sont exécutées pour mettre à jour les poids associés avec les 6 derniers neurones dans la couche de sortie , le $\delta_j(1)$ est aussi accumulé pour former: $\delta_i = \sum g_i(2).w_{ij}(2),\ i = 4,5,6.$ (voir figure .3.14).

3.5.3. SYNTHESE ARCHITECTURALE CONNEXIONNISTE DU BLOC RPG.

Comme nous avons vu précédemment qu'un seul anneau systolique peut exécuter les opérations des deux phases, tout en fournissant la donnée correspondante à l'entrée correspondante. La sélection d'une donnée, arrivée de l'intérieur ou de l'extérieur du bloc de classification, se fait par des multiplexeurs(MUX). Cette dernière sera adressée à l'entrée correspondante d'un processeur grâce à un démultiplexeur(DEM).

On note que les symboles écrits en majuscule utilisés dans le schéma représentent des vecteurs de données à savoir :

$A(1) = \{a_1(0), a_2(0), a_{12}(0)\}$

$A(1) = \{a_1(1), a_2(1), \ldots a_{12}(1)\}$

$A'(1) = \{a_1'(1), a_2'(1), \ldots a_{12}'(1)\}$

$H(1) = \{h_1(1), h_2(1), h_3(1)\}$

$G(1) = \{g_1(1), g_2(1), \ldots g_{12}(1)\}$

$\delta(1) = \{\delta_1(1), \delta_2(1), \ldots \delta_{12}(1)\}$

$A'(2) = \{a_1'(2), a_2'(2), \ldots a_6'(2)\}$

$H(2) = \{h_1(2), h_2(2), \ldots h_6(2)\}$

$G(2) = \{g_1(2), g_2(2), \ldots g_6(2)\}$

$\delta(2) = \{\delta_1(2), \delta_2(2), \ldots \delta_6(2)\}$.

Le schéma suivant représente une architecture du système global du réseau de classification. Pour notre application deux réseaux identiques sont nécessaires.

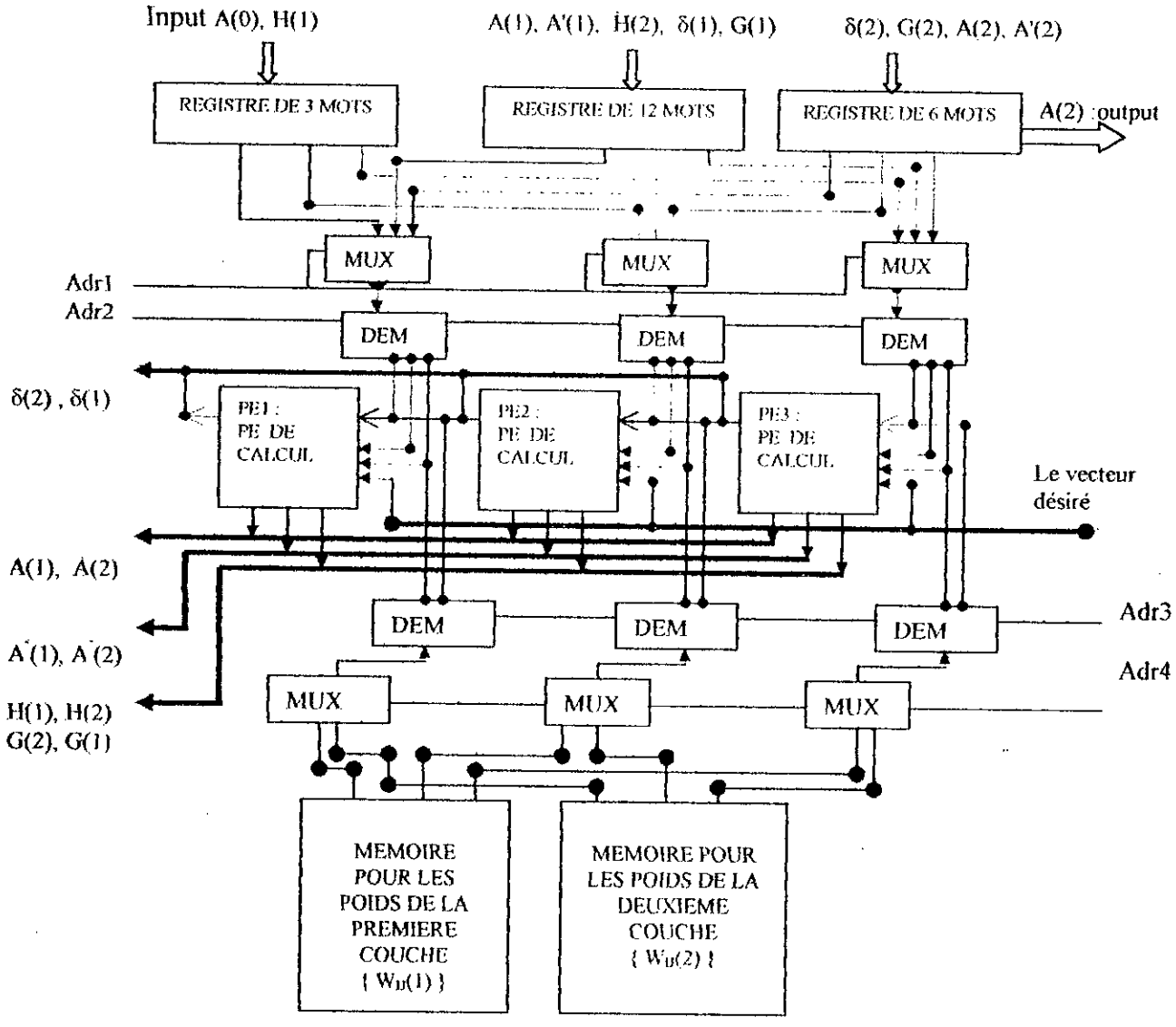


*Fig 3.15. **Schéma du bloc RPG.***

Les trois processeurs sont connectés aux trois premières cases mémoires des registres par le biais des MUXs et des DEMs. Les autres valeurs arrivent par décalage circulaire.

Pendant la phase d'apprentissage les poids synaptiques sont mémorisés dans des mémoires RAMs. A la fin de l'apprentissage les RAMs peuvent être remplacées par des PROMs.

3.5.3 IMPLÉMENTATION DU RÉSEAU D'ACTION

Deux types de processeurs élémentaires sont nécessaires pour implémenter les équations (3.4,3.7, 3.8) sous forme de réseaux systoliques en anneau.

• Un processeur pour la phase d'action: c'est-à-dire calculer les valeurs de Act_i à partir des situations d'anti-collision et des champs de température classés par le classificateur. Ce qui revient à implémenter l'équation (3.8).

• Un autre pour la mise à jour des poids, en utilisant les équations(3.4) et (3.7).

Les situations S_j et T_j arrivent par décalage et les coefficients Q_{ij} et U_{ij} sont disponibles au niveau de la mémoire . A chaque fois la valeur du premier élément de la mémoire est multiplié par la valeur de la situation trouvée au niveau du PE , le produit sera ajouté à l'accumulateur ; six fois pour calculer $\sum Q_{ij}.S_j$ et Six fois pour calculer $\sum U_{ij}.T_j$.

La sortie de l'accumulateur constitue l'entrée de la fonction $G(x)$.

Le résultat trouvé sera ajouté à la variable N_i pour donner la valeur de Act_i.

On utilise la même mémoire des poids et les processeurs spéciaux pour le calcul des poids pour calculer les Q_{ij} et les U_{ij}.

La valeur de Act_i est une entrée fixe pour le processeur alors que l'autre entrée est variable, c'est les différentes situations S_j ou T_j . Chaque fois le processeur effectue un calcul de la valeur du poids, cette dernière remplace l'ancienne valeur c'est-à-dire elle chargée dans la mémoire en écrasant le contenu précédent et un décalage se fait au niveau des cases mémoires et au niveau du registre contenant les situations S et T.

3.5.4.1. PROCESSEURS UTILISES.

Deux types de processeurs élémentaires sont utilisés par le réseau de décision, un pour l'action et l'autre pour la mise à jour des poids

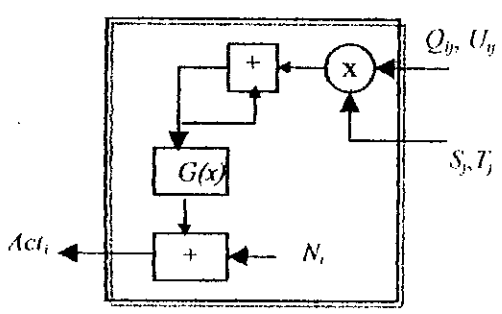


Fig 3.16.a. ***PE pour le calcul des actions.***

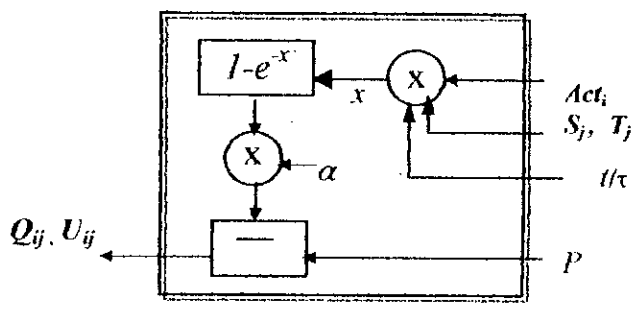


Fig 3.16.b. ***PE pour le calcul des poids***

3.5.4.2. MAPPING DU CALCUL DES ACTIONS Act_is

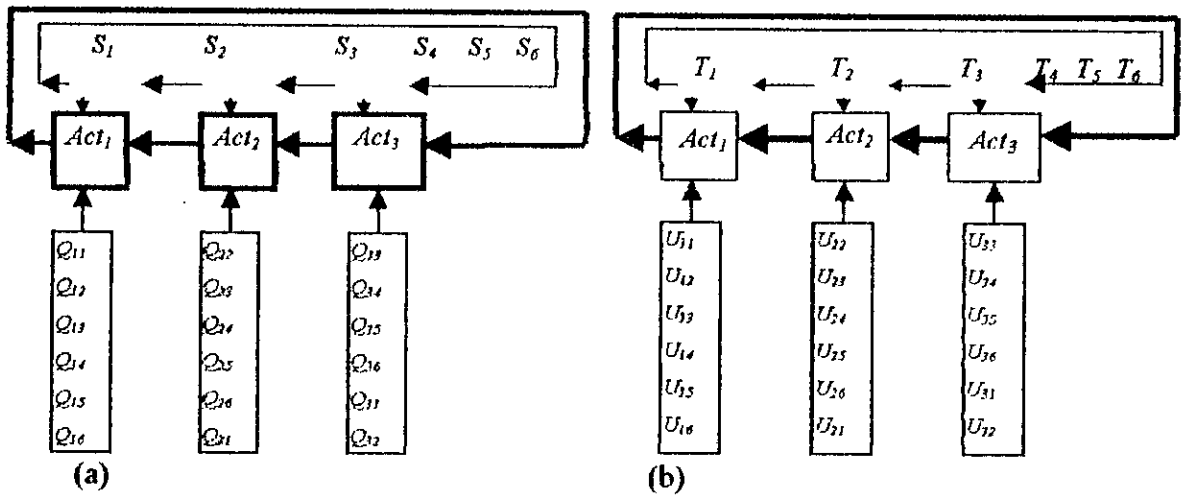


Fig.18. Anneaux systoliques pour le calcul des Act_is.

3.5.4.3. MAPPING DU CALCUL DES POIDS.

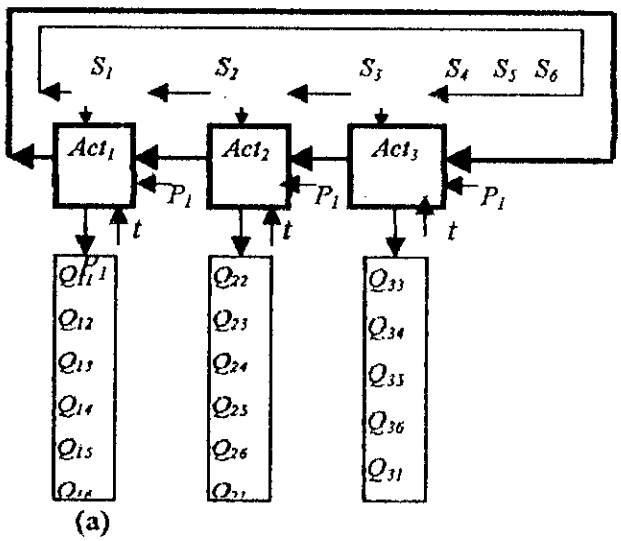


Fig .19.a. anneau systolique pour le calcul des Q_{ij}

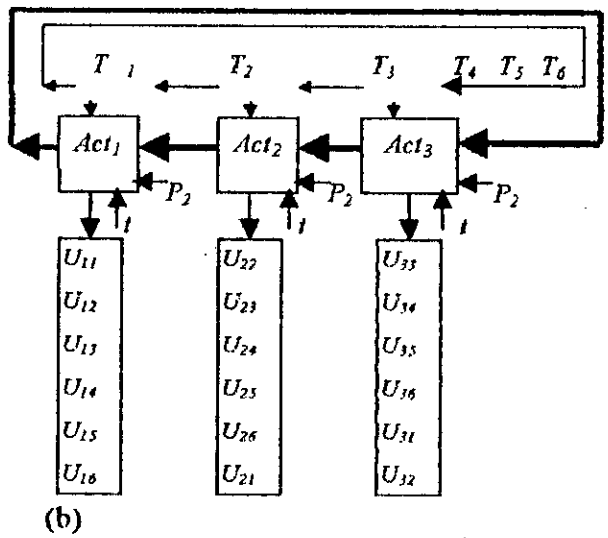


Fig .19.b. anneau systolique. pour le calcul des U_{ij}

3.5.4.4. SYNTHESE ARCHITECTURALE DU BLOC D'ACTION

Dans ce bloc les poids sont associés avec les *PEs* de calcul des poids pour les mettre à jour par un apprentissage renforcé puis avec les *PEs* d'action pour calculer l'action appropriée.

Pendant la phase d'apprentissage dans l'environnement, les deux types de processeurs sont utilisés. Une fois l'apprentissage est fait, seulement les PEs d'actions sont nécessaires. Les autres *PEs* (*PEs* de calcul des poids) ne sont pas utilisés.

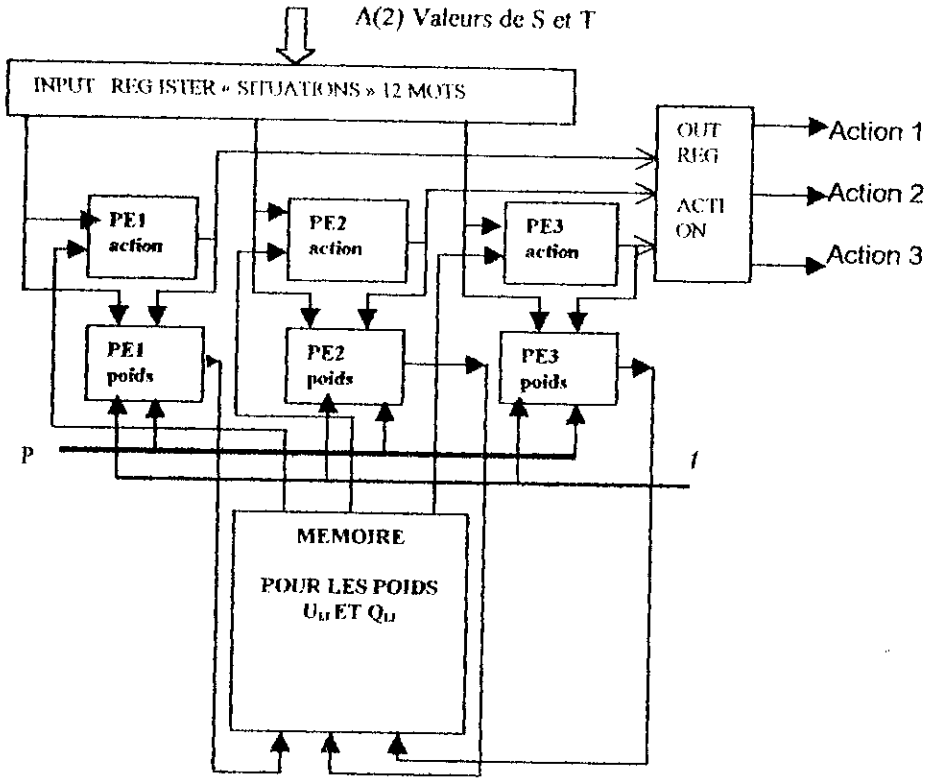


Fig 3.19. ***Schéma du bloc d'action.***

3.5.3. RESEAU CONNEXIONNISTE TOTAL :

Le réseau total est une association entre deux blocs de réseaux de classification et le bloc d'action comme il est montré dans la figure 3.20.

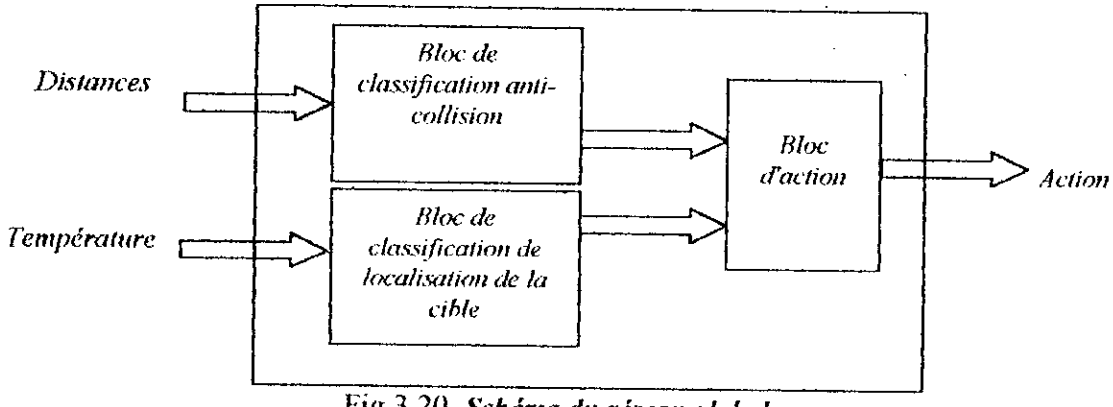


Fig 3.20. ***Schéma du réseau global***

3.6. CONCLUSION

Un système neuronal était développé pour la navigation d'un robot mobile autonome dans un environnement partiellement structuré, dans lequel est défini un champ de température indiquant la position de la cible à atteindre.

Pour des raisons de facilité de dérivation d'une architecture systolique et son implémentation, une structure régulière (3,4x3,2x3) d'un réseau de neurones était choisie. Cette structure a permis de partitionner le traitement de l'algorithme *RPG* du système entier en un traitement par lot de 3 ; ce qui a conduit à utiliser trois processeurs élémentaires, travaillant en parallèle et d'une manière complètement pipelinisée, pour exécuter toutes les opérations nécessaires durant les deux phases de traitement de l'algorithme *RPG* .

Avec un ordonnancement temporel des opérations et une exploitation des ressources de calcul d'une façon optimisée. En utilisant des ressources électroniques supplémentaires de multiplexage et de démultiplexage afin d'assurer le flux de données correspondant à chaque *PE*, nous avons pu implémenter cet algorithme avec le même anneau systolique.

De la même manière un réseau d'action, qui constitue un bloc d'application pour deux réseaux de classification, était implémenté par un autre anneau systolique.

Chapitre 4

MODELISATION ORIENTEE OBJET DES RNAs

4.1. INTRODUCTION

Les techniques orientées-objet (T.O.O.) fournissent une méthode uniforme pour la représentation du hardware et du software sur le même pied d'égalité. Cette représentation unifiée permet à ces techniques d'être appliquées d'un domaine à un autre grâce aux abstractions de données. Par conséquent, les concepts de T.O.O utilisés dans le domaine du software peuvent être utilisés dans le domaine du hardware. Ainsi que, les techniques software existantes, comme celles utilisées pour la vérification de la justesse des implémentations de données abstraites peuvent être utilisées dans la conception hardware, comme elles peuvent aussi assister la gestion de la complexité de la conception[2][3].

4.2. MOTIVATIONS POUR LES TECHNIQUES ORIENTEES OBJET.

L'application des *T.O.Os* à la conception hardware est une fertilisation réciproque de l'ingénierie du logiciel à l'ingénierie du matériel.

Avec l'arrivée des langages de description du hardware tel que le *VHDL* et l'accroissement de l'utilisation de la simulation, la modélisation dans la conception hardware a poussé les chercheurs à réfléchir à des techniques de programmation qui améliorent les procédures de modélisation. Dans certains cas les développeurs de la modélisation hardware ont besoin du rôle de programmeurs du software[2].

Il est naturel de réfléchir à des ressources hardware comme composants ayant des états et des opérations manipulant ces états. En plus, pour la construction des systèmes utilisant des composants de bibliothèque réutilisables.

Un grand avantage de la T.O.O se manifeste dans le domaine du hardware comme une conception utilisant des blocs de construction. L'utilisation de ces blocs permet aux systèmes d'être construits rapidement, un coût de conception faible et une fiabilité augmentée.

La T.O.O offre les avantages suivants, lorsqu'elle est appliquée à la conception hardware[2].

- Augmentation de la fiabilité et de la maintenabilité.
- Instantiation des composants avec plusieurs paramètres.
- Une composition rapide d'un nouveau composant.
- Aptitude à identifier et réutiliser des composants communs.
- Création et destructions d'objets dynamiques.
- Possibilité d'employer des synthèses software existantes et de techniques de vérification.

4.3. TYPES DE DONNEES

Les types de données sont présentés au hardware de la même manière que le software. Les langages de programmation offrent un ensemble de types de données prédéfinis, comme entier, caractère, booléen, réel etc.... Les machines supportent les types de données, bit, entier, flottant etc.... Ces types sont certainement communs.

Quelques langages de programmation permettent la création de types de données abstraites (abstraction de données) et de types de données définis par l'utilisateur, qui représentent un peu l'abstraction d'une entité de la vie réelle. De la même manière que pour des nouvelles fonctions qui peuvent être ajoutées à la machine virtuelle définie par un langage de programmation, des nouveaux types de données, chaînes, listes etc...., peuvent être ajoutées aussi.[3]

4.4. MODELISATION DES COMPOSANTS HARDWARE COMME CLASSES

L'abstraction des données peut être utilisée pour représenter le hardware.

Le C++ supporte l'abstraction de données par le concept de classe. En général, une classe correspond à un ensemble d'éléments ayant des caractéristiques communes. De cette façon, un composant hardware peut être traité comme une classe contenant des états avec une collection d'opérations associées pouvant manipuler ces états.[3]

4.5. DERIVATION DE COMPOSANTS SPECIALISES

Il est possible de dériver plusieurs classes spécialisées avec une collection de classes de base avec la conservation de toutes ses spécifications et en ajoutant d'autres fonctions associées ainsi que d'autres types de données.

Les dérivations des processeurs spécialisés souhaités pour une application particulière sont exécutées en commun dans la conception des systèmes.

Pour supporter une telle spécialisation, plusieurs outils d'automatisation sont développés pour améliorer des processeurs spécifiques, cette procédure est accomplie à travers la technique d'héritage[2].

4.6. PARAMETRES ET FONCTION D'ACTIVATION.

4.6.1. NUMERISATION DE LA FONCTION SIGMOIDE.

La fonction sigmoïde $f(x) = 1/(1+e^{-x})$ peut être approximée par des segments linéaires ou mémorisée comme une table de valeurs dans une mémoire (Look-Up Table: LUT). Sa dérivée est donnée par une valeur fixe pour chaque segment linéaire.

Les valeurs de cette fonction approximée par 7 segments linéaires et sa dérivée, inspirées d'un travail développé dans ce domaine (voir référence 14), sont données par les deux tableaux ci-dessous qui sont :

Input range(x)	Function output range	Derivate function output
-8 ... -4	0 ... 0.0625	0.015625
-4 ... -2	0.0625 ... 0.125	0.03125
-2 ... -1	0.125 ... 0.25	0.125
-1 ... 1	0.25 ... 0.75	0.25
1 ... 2	0.75 ... 0.875	0.125
2 ... 4	0.875 ... 0.9375	0.03125
4 ... 8	0.9375 ... 1	0.015625

*Fig 4.1.a. **Numérisation de la fonction sigmoïde et sa dérivée***[15].

4.6.2. NUMERISATION DE LA FONCTION EXPONENTIELLE DU RESEAU D'ACTION

La fonction exponentielle $1-e^{-x}$ utilisée par le réseau d'action est aussi approximée par des segments linéaires qui sont explicitées dans le tableau suivant(fig 4.1.b).

Input range(x)	*Output*
0...1	*0.633.x*
1...2	*0.232.x - 0.401*
2...3	*0.085.x - 0.69*
3...4	*0.031.x - 0.857*
4...5	*0.012.x - 0.933*
5...6	*0.004.x - 0.973*
6...∞	*1*

Fig 4.1.b ***Numérisation de la fonction exponentielle du réseau d'action***

4.6.3. VALEURS DES PARAMETRES UTILISES

Le taux d'apprentissage $\eta = 0.25$: pour le réseau de classification.

Pour le réseau de décision, les paramètres doivent vérifier:

$0<\beta<\alpha<P_2<2\alpha-\beta$ et $P_1>2\alpha$.

Pour cela nous avons choisi $\alpha=5$, $\beta=0.33$, $P_1=11$, $P_2=9$, $\tau=3s$.

β=0.33 est une variable aléatoire générée par la fonction Random qui génère une séquence aléatoire comprise entre *[0 ... 32627]* que nous avons normalisé par rapport à 10^{+5}.

4.6.4. JUSTIFICATION DES APPROXIMATIONS ET DES PARAMETRES

Le choix du taux d'apprentissage ($\eta = 0.25$) ne doit pas être :

1- très grand pour assurer la convergence vers la solution.

2- très petit pour minimiser le nombre d'itérations(temps de calcul).

Les paramètres du réseau d'action (β, α, P_2, P_1) doivent répondre seulement au critère cité dans 4.6.3. Leur choix dépend de l'utilisateur.

Les approximations des fonctions, sigmoïde et exponentielle, sont approximées par des segments linéaires ; c'est la méthode la plus utilisée dans le domaine de l'implémentation digitale des fonctions non linéaires.

Ces choix des paramètres et approximation des fonctions ont prouvé leur validité lors des résultats.

4.7. MODELISATION ORIENTEE OBJET DES RNAs

Pour clarifier les notions de base de la programmation orientée objet (*P.O.O*), qui seront appliquées à la modélisation des composants hardware par des classes software, nous explicitons les définitions suivantes.

Objet : La P.O.O est fondée justement sur le concept d'objet, à savoir une association des données et des procédures (qu'on appelle méthodes) agissant sur ces données[19].

méthodes + données = Objet.

a. Encapsulation des données : Cette association n'est qu'une simple juxtaposition. En effet dans la P.O.O, on réalise ce qu'on appelle une encapsulation des données. Cela signifie qu'il n'est pas possible d'agir directement sur les données d'un objet; il est nécessaire de passer par l'intermédiaire de ses méthodes qui jouent ainsi le rôle d'interface obligatoire. On traduit parfois cela en disant que l'appel d'une méthode est en fait l'envoi d'un "message" à l'objet[19]. Le grand mérite de l'encapsulation est que vue de l'extérieur, un objet se caractérise par les spécifications de ses méthodes, la manière dont sont implémentées les données étant sans importance. On décrit souvent une telle situation en disant qu'elle réalise une "abstraction des données". (Ceux qui expriment bien les détails concrets d'implémentation sont cachés). A ce propos on peut remarquer qu'en programmation structurée, une procédure pouvait également être caractérisée (de l'extérieur) par ses spécifications, mais que, faute d'encapsulation, l'abstraction des données n'était pas réalisée[20].

L'encapsulation des données présente un intérêt manifeste en matière de qualité de logiciel. Elle facilite considérablement la maintenance[19]: une modification éventuelle de la structure de données d'un objet n'a incidence que sur l'objet lui-même. Les utilisateurs de l'objet ne seront pas concernés par la teneur de cette modification (ce qui n'était bien sûr pas avec la programmation structurée). De la même manière, l'encapsulation des données facilite grandement la réutilisation de l'objet comme les composants hardware.

b. Classe : En P.O.O. apparaît généralement le concept de classe. Ce dernier correspond simplement à la généralisation de la notion de type que l'on rencontre dans les langages classiques. Une classe, en effet, n'est rien d'autre que la description d'un ensemble d'objets ayant une structure commune et disposant des mêmes méthodes. Les objets apparaissent comme des variables d'un type classe; en *P.O.O* on dit aussi qu'un objet est une "instance" de la classe[19][20].

c. Héritage : Un autre concept important en P.O.O est celui d'héritage. Il permet de définir une nouvelle classe à partir d'une classe existante (qu'on réutilise en bloc) ou de plusieurs classes, à laquelle on ajoute de nouvelles données et de nouvelles méthodes. La conception de la nouvelle classe qui ' hérite' des propriétés et des aptitudes de l'ancienne ou des anciennes, peut aussi s'appuyer sur des réalisations antérieures parfaitement au point, et les spécialiser .

d. Polymorphisme : le polymorphisme veut dire multi-formes. Un objet peut avoir plusieurs formes au court du temps. Le polymorphisme est une caractéristique des fonctions membres de la classe plus qu'une caractéristique de l'objet lui même. Il est implémenté à travers l'architecture de la classe. Par conséquent, seulement, les fonctions membres de la classe peuvent être polymorphiques.

4.7.1. MODELISATION DES CLASSES DU RNA

Dans la pratique, en utilisant le concept de la classe comme "composant logiciel", en effet, dans un souci de réutisabilité, la classe sera fournie comme un composant séparé pouvait l'être d'ailleurs une fonction C destinée à être employée par plusieurs programmes.

En utilisant cette notion de classe pour la modélisation d'un réseau de neurones appliqué à la navigation d'un *RMA*. Cette application a nécessité deux classes de réseaux.

- une classe: réseau de classification (classificateur),
- une classe: réseau d'action (actionneur).

Dans le but d'arriver à la modélisation de ces deux classes de réseaux, il était nécessaire de modéliser les classes des composants suivants: ,

- un processeur élémentaire pour le réseau classificateur,
- registre à décalage circulaire pour les entrées (*Da, Dd* et *Dg* ou *Ta, Td* et *Tg*),
- un registre à décalage circulaire pour les états de la couche cachée,
- un registre pour la sortie (vecteur *S* ou *T*),
- une mémoire pour stocker les poids de la couche cachée,
- une mémoire pour stocker les poids de couche de sortie,
- un processeur élémentaire d'action pour réseau d'action,
- un processeur pour le calcul des poids du réseau d'action,
- une mémoire pour stocker les poids du réseau d'action,

4.7.1.1. LES CLASSES REGISTRES.

Ceux sont des registres à décalage circulaire ayant des variables privés qui représentent les valeurs internes des registres.

La mise à jour de ces variables se fait par les fonctions membres (méthodes)suivantes.

- une fonction membre *INIT* (initialise le registre à 0 au début de chaque opération),
- une fonction membre *LOAD* (charge le registre par des valeurs d'entrée ou calculées),
- une fonction membre *SHIFT* (décale circulairement les mots du registre d'une position),
- une fonction membre de lecture (*READ*)(permet de lire les valeurs internes du registre).

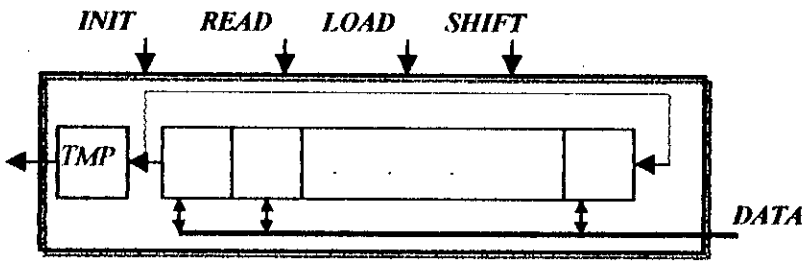


Fig 4.2. ***Classe Registre***

Le code écrit en C++ est décrit par:

```
CLASS REGISTER
{ private: double data[size], tmp ;      /* données encapsulées*/
  public:
  void init ( )    /* fonction d'initialisation */
{ for ( p = &data[0 ]; p<&data[size]; p++)  *p = 0 ;
                }
 void load ( double tab[size]) /* fonction de chargement du registre par le tableau T */
        { for (I=0; I<size; I++) { data[I] =tab[I] }
        }
 double read ( ) /* fonction permettant de lire la valeur tampon du registre */
                { return(tmp);}
 void shift( )  /* fonction permettant le décalage d'une seule position */
        {   tmp=data[0];
            for (p1=& p[0];p1<& p[size-1];p1++)
             {p2=p1+1;* p1 = * p2;}
                  data[size-1]= tmp;
        }
};
```

4.7.1.2. LES CLASSES MEMOIRES

Une mémoire est un ensemble de registres pouvant mémoriser des mots ayant la même longueur.

Les variables déclarées privées sont aussi les valeurs internes des cases mémoires.

Les fonctions membres déclarées publiques, comme celles des registres, permettant de réaliser les fonctions suivantes :

- une fonction d'initialisation (initialise les positions mémoires à des valeurs aléatoires),
- une fonction de décalage (permet de décaler circulairement chaque registre-mémoire d'une seule position),
- une fonction membre de lecture (permet de lire les données de la mémoire),
- une fonction membre de chargement (permet de charger la mémoire avec des valeurs calculées par les processeurs).

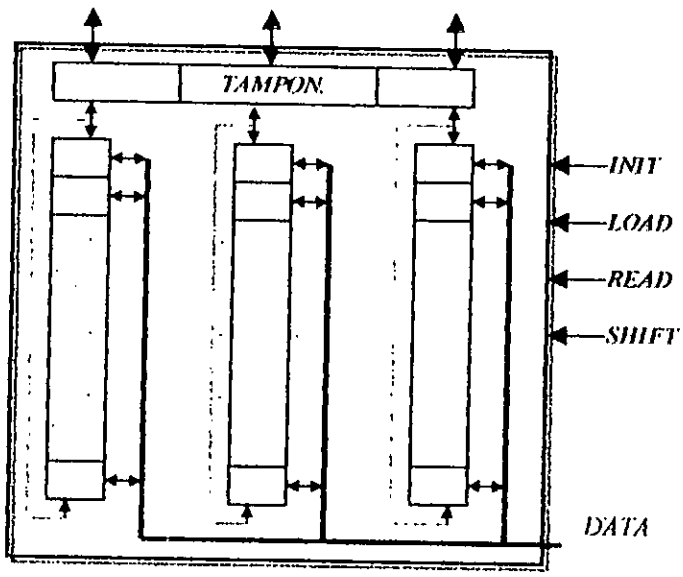


Fig 4.3. Classe mémoire

- ***mémoire 1 (3X12 mots)***
- ***mémoire 2 (3X24 mots).***

Le code écrit en C++ est similaire à ce lui des registres, seulement les données sont représentées sous formes de tableaux à deux dimensions.

4.7.1.3. LA CLASSE PE DE CLASSIFICATION

C'est un processeur élémentaire, il exécute les opérations de multiplication, multiplication et accumulation, soustraction, fonction d'activation et sa dérivée pour le calcul des potentiels dans la phase de relaxation, et le calcul des poids pour la phase d'apprentissage.

Le séquencement des opérations se fait par l'activation des fonctions membres suivantes.

- une fonction membre de multiplication(*PROD*) (fait appel à l'opération de multiplication),
- une fonction membre de soustraction(*SUB*) (fait appel à l'opération de soustraction),
- une fonction membre de produit et accumulation(*MAC*) (est une appel pour exécuter l'opération de multiplication et accumulation),
- une fonction membre d'activation de la fonction (*ACT*) (elle fait appel à la fonction sigmoïde),
- une fonction membre de dérivation (*DER*) (fait appel à la dérivé de la fonction sigmoïde).

" On note que la fonction sigmoïde est approximée par des droites de formes $a.x+b$, ce qui ne nécessite que des opérations de multiplication et d'addition pour sa modélisation".

- une fonction membre d'initialisation (permet d'initialiser l'accumulateur à zéro),

Le code écrit en C++ est décrit par les instructions suivantes.

" Les mots écrits en gras sont les mots clés (Key words) du langages C++".

```
CLASS PE_MUL_ACC  /*processeur élémentaire produit +accum  formant une classe de base*/
 { private : double  p, sum;
   public :
   void init ( ) { p =0; sum=0; }
   double prod_accum(double a, double b)
  { p  = a*b ; sum=sum+prod;return (sum);}
    };
```

La classe du processeur *PE* spécialisé (*PE1* decrit ci-dessous), utilisé par le réseau de classification, hérite le processeur de base (PE de prod + accum) et en ajoutant d'autres fonctions et d'autres données pour dériver ce processeur.

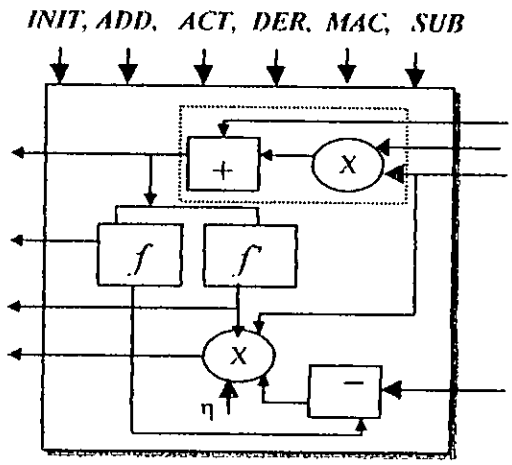


Fig.4.4. ***La classe CPE:PE du réseau classificateur***

```
CLASS CPE : PUBLIC PE_MUL_ACC
    {   private: double ac,acp,ers, ....   /* donnée spéciales à cette classe*/
        puiblic :
   double activation(double var)
          { if ((var <-8) )                  ac =0 ;
            else if((-8<var)&& (var<-4))     ac = 0.015625*var+0.125;
            else if((-4<var)&& (var<-2))     ac = 0.03125*var+0.1875;
            else if((-2<var)&& (var<-1))     ac = 0.125*var+0.3750;
            else if((-1<var)&& (var<1))      ac = 0.25*var+0.5;
            else if((1<var) && (var<2))      ac = 0.125*var+0.625;
            else if((2<var) && (var<4))      ac = 0.031125*var+0.8125;
            else if((4<var) && (var<8))      ac = 0.015625*var+0.875;
            else if(var>8)  ac =1;    return(ac);
          }
         double derivation(double var)
            {       if((-8<var)&& (var<-4))    acp = 0.015625;
               else if((-4<var)&& (var<-2))    acp = 0.03125;
               else if((-2<var)&& (var<-1))    acp = 0.125;
               else if((-1<var)&& (var<1))     acp = 0.25;
               else if((1<var) && (var<2))     acp = 0.125;
               else if((2<var) && (var<4))     acp = 0.031125;
               else if((4<var) && (var<8))     acp = 0.015625;
               else acp =0;    return(acp);
            }
   double prod(double var1,double var2)/* fonction de multiplication */
        { return(var1*var2); }

double  sub(double var1,double var2)/* fonction de soustraction */
        { ers = var1-var2; return(ers); }
};
```

De la même manière que la description de cette classe, la procédure est suivie pour décrire les autres processeurs spécialisés, utilisés par le réseau de décision, tout en héritant le *PE* de base (*PE_PROD_ACC*) et en spécifiant les fonctions membres et les données abstraites à ajouter.

4.7.1.4. LA CLASSE PE D'ACTION

Elle a le même principe que le *PE* de classification; elle hérite le composant de base (multiplieur + accumulateur). Elle diffère seulement par la fonction d'activation et de mise à jour des poids

Ce processeur est utilisé pour le calcul de la fonction $Act_i = G(\sum_j Q_{ij}.S_j + U_{ij}.T_j) + N_i$

Il possède les fonctions membres suivantes.

- une fonction membre d'initialisation (*INIT*),
- une fonction membre de multiplication et accumulation (*MAC*),
- une fonction membre d'activation de la fonction *G(x)(ACT)*,
- une fonction membre d'addition(*ADD*).

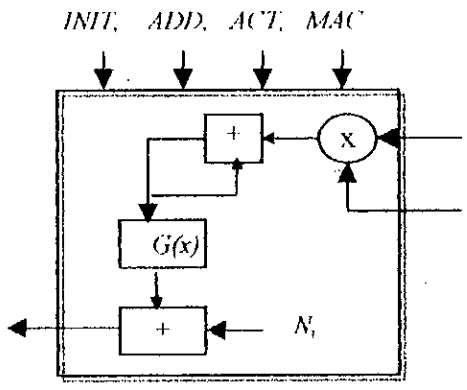


Fig.4.5. La classe APE1: PE d'action du réseau d'action

4.7.1.5. LA CLASSE PE DE CALCUL DES POIDS

Ce processeur est un processeur spécial utilisé pour effectuer la mise à jour des Q_{ij} et des U_{ij}. Les fonctions membres suivantes sont utilisées:

- une fonction membre d'initialisation(*INIT*),
- une fonction membre de soustraction(*SUB*),
- une fonction membre de multiplication(*PROD*),
- une fonction membre d'activation de la fonction exponentielle (*ACT*)

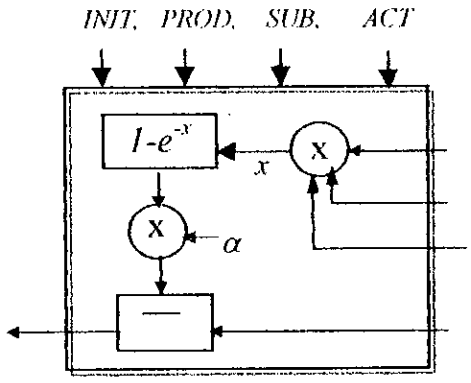


Fig.4.6. la classe APE2: PE pour le calcul des poids

4.7.1.6. LA CLASSE RESEAU CLASSIFICATEUR.

Cette classe n'est rien qu'une classe d'assemblage d'autres composants obtenus (autres classes). Pour notre application envisagée, le réseau classificateur nécessite:

- une mémoire pour les poids de la couche cachée,
- une mémoire pour les poids de la couche de sortie,
- trois processeurs élémentaires de classification,
- un registre à décalage (de trois mots réels) pour les entrées (pour *Da, Dd, Dg* ou *Ta,Td,Tg*),
- un registre à décalage (de douze mots réels) pour la couche cachée,
- un registre à décalage (de six mots réels) pour la couche de sortie,
- une mémoire locale de 75 mots (File-Register) pour mémoriser instantanément les valeurs intermédiaires de H,G,δ...

Ces composants sont déclarés privés dans la classe réseau classificateur, leurs mises à jour sont par les fonctions membres suivantes:

- une fonction membre *MVM1* permettant de calculer les états de la couche cachée,
- une fonction membre *MVM2* permettant de calculer les états de la couche de sortie,
- une fonction membre *VMM* permettant de calculer l'erreur corrective,
- une fonction membre *OPU2* permettant de calculer les poids de la couche de sortie,
- une fonction membre *OPU1* permettant de calculer les poids de la couche cachée.

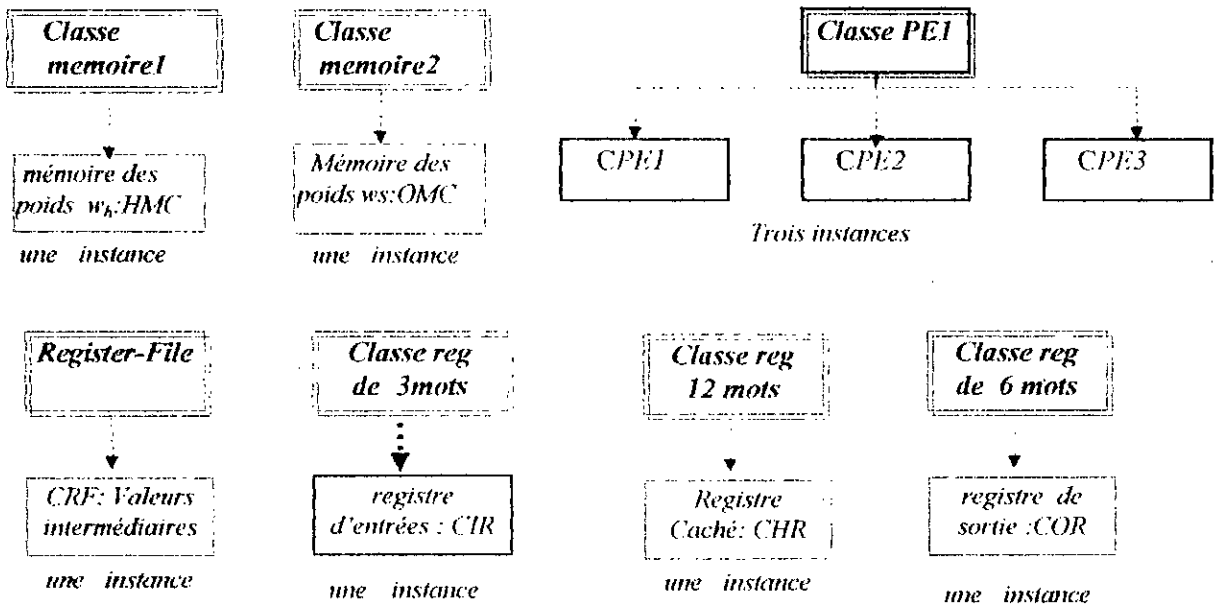


*Fig 4.7.**Composants nécessaires pour l'implémentation du réseau de classification***

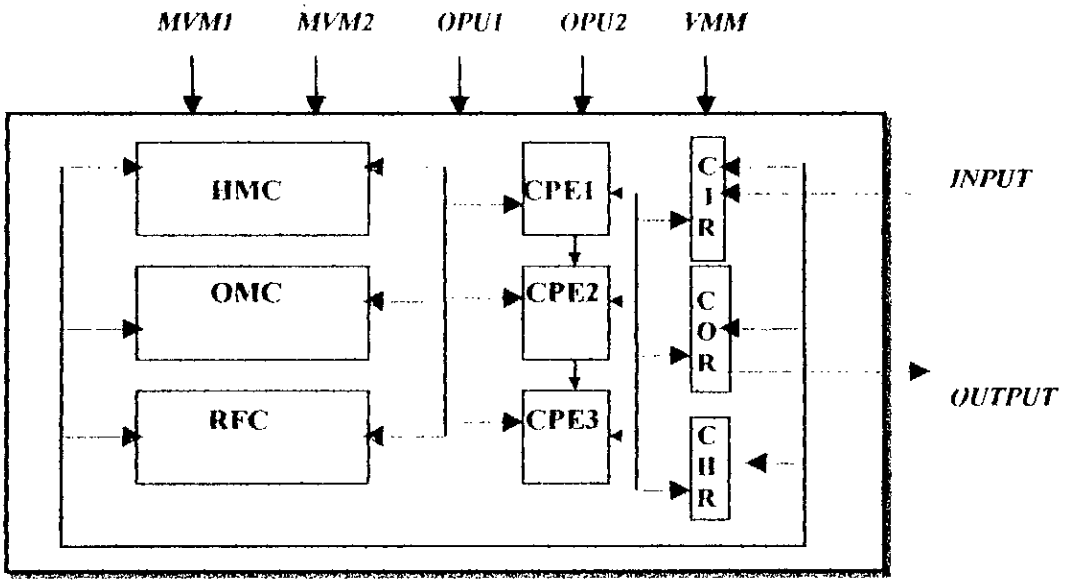


Fig.4.8. ***La classe réseau classificateur: CNC.***

Avant de décrire le réseau en C++ on doit déclarer les classes utilisées ainsi que le nombre des instances. En utilisant la technique de composition d'un composant à partir des composants disponibles pour former la classe réseau. Le code en C++ est décrit par:

```
CLASS classifier
{ private: /*----------Déclaration des classes utilisées par le réseau de classification----------*/
  P_UNIT      PE1[nbr_col];
  class REGIN       input];
  class REGHI       hid_reg1;
  class REGOUT      output;
  class REG FILE    pile;
public:
  class MEM_CACHE   memore1;
  class MEM_SORTIE   memoire2;

  void load_entree(double t[nbr_col]) /*----- chargement des entrées ----------*/
    {.....                }
  void load_sortie(double tb[nbr_sec][nbr_col]) /*-------- chargement des sortie Désirées ----------*/
    { .....               }
  void MVM1() /*-------- Calcul des activations de la couche cachée -------*/
    {.....                }
  void MVM2()/* -------- Calcul des activations de la couche de sortie---------*/
    { .....               }
  void VMM1()/* -------- Calcul des erreurs correctives retro-propagées -----*/
    {.....                }
  void OPU1()/*---------- mise à jour des poids de la couche de sortie----*/
    {.....                }
  void OPU2() /*---------- mise a jour des poids de la couche cachee----*/
    {.....                }
};
```

4.7.1.7. LA CLASSE RESEAU D'ACTION

Dans cette classe on a deux classes de processeurs élémentaires

- une pour le calcul des potentiels Act_i (trois instances),
- une pour le calcul des poids Q_{ij} et U_{ij} (trois instances pour chaque classe),
- une mémoire de 36 mots pour stocker les Q_{ij} et les U_{ij},
- un registre à décalage de 12 mots pour les entrées (S et T).

L'activation de ces composants se fait par les deux fonctions membres suivantes:

- une fonction d'initialisation permettant d'initialiser le réseau.
- une fonction d'activation permettant de calculer les $Act_i s$, elle utilise les processeurs d'actions comme des cellules de calcul et la mémoire pour stocker les poids $\{Q_{ij}\}$ et $\{U_{ij}\}$,
- une fonction permettant de calculer les Q_{ij} et les U_{ij} (Learning), elle utilise les processeurs élémentaires de calcul des poids comme des cellules de calcul et la memoire3 pour le stockage des poids$\{Q_{ij}\}$ et $\{U_{ij}\}$.

Pour l'implémentation de ce réseau les composants montrés dans la figure.4.8 sont utilisés.

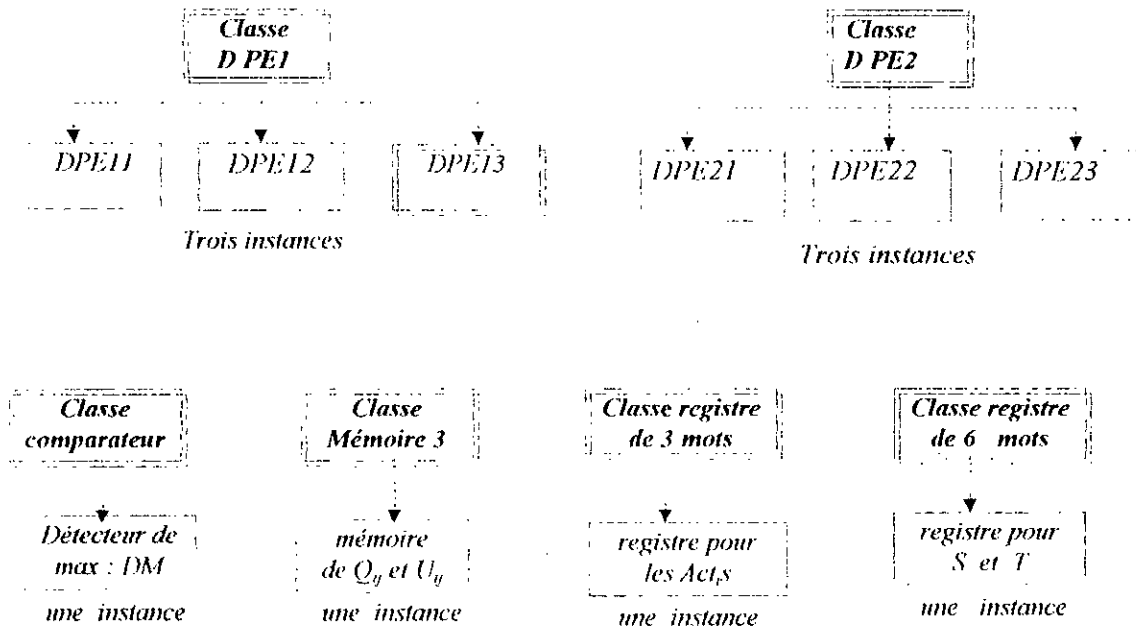


Fig 4.9. Composants nécessaires pour l'implémentation du réseau d'action

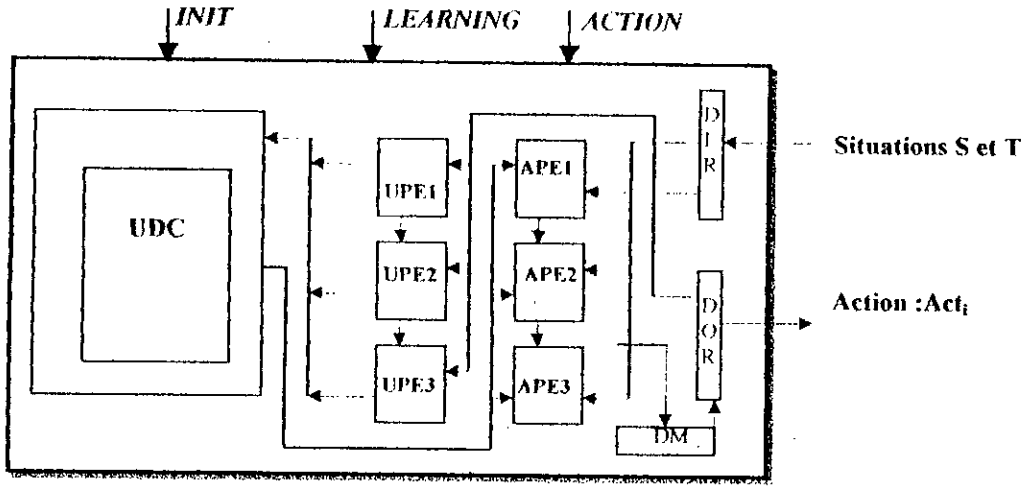


Fig 4.10. La classe réseau d'action: ANC

La description de cette classe en C++ est comme suit:

```
    /*---------- Composition de la nouvelle classe----------*
CLASS actionner
    {
 private:  /*------- Déclaration des classes utilisés par le réseau de décision ----------*/
     class DPE1  actionner[nbr_sec];
     class REGIN2    reg_sit;
     class DPE2    cal_poid[nbr_sec];
     class comparateur detect_max;
     int k,i.....;
 public:
     class memoire36   mem_sit[nbr_sec];
     void init()
      { .....              }
     void learning()
      { .....              }
     void decision()
      { ......             }
};
```

4.7.1.8 LA CLASSE CONTROLEUR

Enfin, après la modélisation de ces deux classes de réseaux, une classe contrôleur utilisant deux instances(*CNC_OBS, CNC_TEMP*) de la classe classificateur (*CNC*) et une instance (*ANC_DEC*) de la classe actionneur(*ANC*). Cette classe constitue un circuit de commande pour le robot ayant comme fonctions membres;une fonction membre(*COM*) pour la commande des réseaux

classificateurs, une fonction membre d'initialisation (*INIT*), une fonction membre (*APP*) pour l'apprentissage et une fonction membre(*DEC*) pour la décision en faveur d'une action.

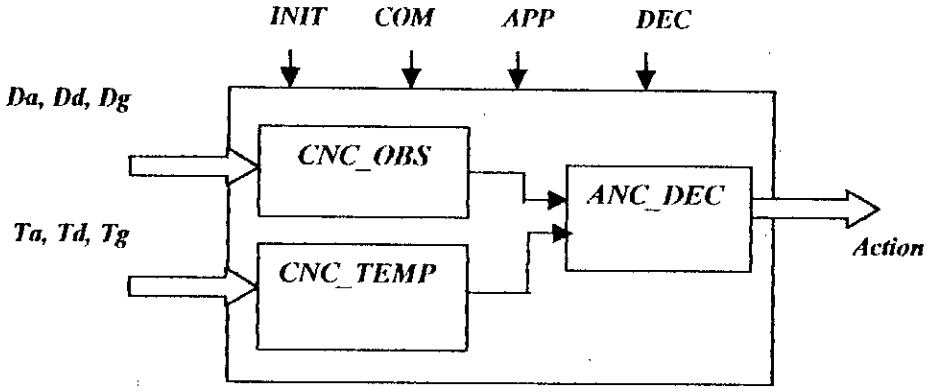


Fig 4.11 ***La classe contrôleur.***

```
CLASS controller
 {   private:
     reseau_classifier  classifieur[2];/* deux instances de classification*/
     reseau_dec actionneur; /* une instance d'action*/
     public:
    void classifer_com(void)
    {
      void Wh_reading();
      void Ws_reading();
      void q_storage();
      void q_reading();
      Wh_reading();
      Ws_reading();
   /*   chargement des entrées, distances et température */
     entree[0][0]=Da; entree[0][1]=Dd; entree[0][2]=Dg;
     entree[1][0]=Ta; entree[1][1]=Td; entree[1][2]=Tg;
     classifieur[0].load_entree(entree[0]);
     classifieur[1].load_entree(entree[1]);
     for(r=0;r<2;r++)
      {   classifieur[r].memoire1.load(whc[r]);
          classifieur[r].memoire2.load(wso[r]);
               /*  opération de classification*/
               classifieur[r].MVM1();
               classifieur[r].MVM2(); }
      }
        q_reading();
         void actionner_init(void)/* initialisation*/
        {  actionneur.init();}
         void actionner_app(void) /*apprentissage dans l'enviro */
        {  actionneur.learning();}
         void actionner_dec(void) /*décision en faveur d'une action */
       {  actionneur.decision();}

   };
```

4.7.2. DIMENSIONS DES DONNEES.

Les variables utilisées par notre application sont déclarées double, qui veut dire qu'ils sont des variables à virgule flottante en double précision, d'où sa représentation est sur huit octets (64 bits).

4.8. CONCLUSION.

En vue d'une modélisation orientée objet de l'architecture systolique des réseaux de neurones, nous avons utilisé la *P.O.O* comme un moyen de simulation. Ses spécifications (de la *P.O.O*) nous ont permis de construire des composants softwares (classes) pour une implémentation de l'algorithme *RPG* et d'autres classes pour le réseau d'action.

L'exploitation de ces classes ne constitue pas un cas particulier pour notre application, elles peuvent être exploitées par une autre application utilisant cet algorithme, tout en choisissant le nombre de composants nécessaires et en modifiant seulement les tailles des registres et des mémoires selon les exigences de l'application.

Les instances composant le réseau de classification sont utilisées durant les deux phases, apprentissage et utilisation.

Lors de l'utilisation, seules les fonctions membres relatives à la phase de relaxation, sont activées. Les fonctions membres relatives à la phase d'apprentissage (retro-propagation) ne sont pas utilisées.

Concernant le réseau d'action, lors de l'utilisation, les processeurs de calcul des poids ne sont pas nécessaires. Le calcul des actions se fait par les *PEs* d'action.

Chapitre 5

RESULTATS ET INTERPRETATIONS

5.1. ENVIRONONEMENT D'APPRENTISSAGE.

L'environnement d'apprentissage peut avoir différentes formes géométriques . Dans ce cas, il doit être de telle sorte que le robot puisse rencontrer toutes les situations spatiales (relatives à l'anti-collision) , ainsi que toutes les situations relatives au champ de température.
L'environnement d'apprentissage suivant a été choisi à ce que toutes les situations (relatives à l'anti-collsion et au champ de température) soient rencontrées.
De plus la largeur des couloirs doit être fixée de façon à avoir une seule action libre en collision permise dans chaque situation, c'est à dire que dans un couloir, le robot ne peut pas tourner à gauche ou à droite.

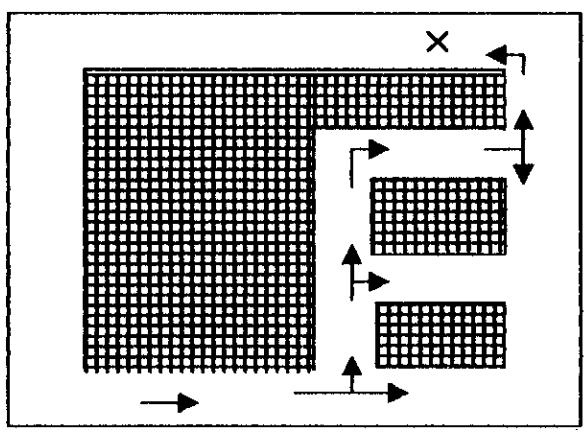


Fig 5.1 ***Environnement d'apprentissage***

5.2 ENVIRONNEMENT VISE (application)

Notre programme offre la possibilité de représenter un ou plusieurs obstacles de formes rectangulaires ou carreés pour simuler l'environnement d'application.

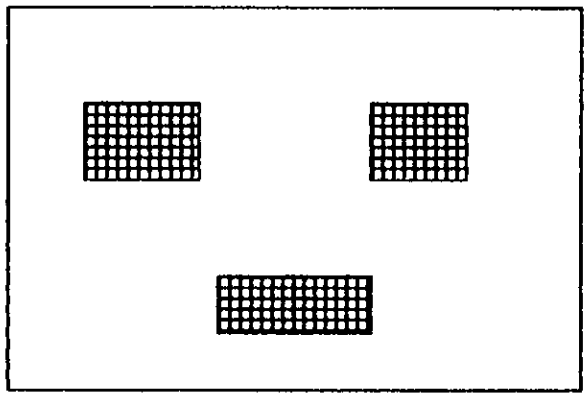


Fig 5.2. ***Environnement visé.***

5.3. SIMULATION ET RESULTATS

Dans ce chapitre, nous présentons une simulation graphique de l'environnement.

Le robot est simulé par un carré dont son avant , sa droite et sa gauche sont définis.

Les obstacles sont simulés sous formes rectangulaires ou similaires.

A chaque instant le programme calcule les distances aux obstacles de tous les cotés du robot et les champs de température par une projection des coordonnées de la cible sur l'axe du robot.

Ces distances et champs constituent les entrées du programme de commande "réseau implémenté sous forme de réseaux systoliques" et génère l'action qui doit être prise par le robot.

Cette procédure continue jusqu'à où le robot atteint sa cible marquée par le symbole (+) dans le graphe de simulation.

5.3.1. TEST DANS UN ENVIRONNEMENT LIBRE D'OBSTACLES

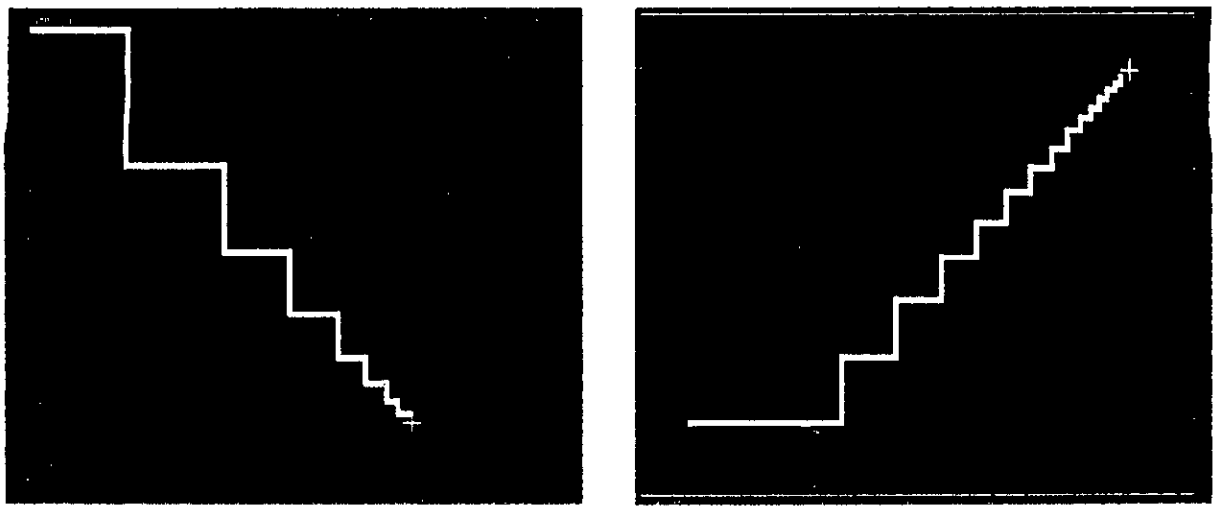


Fig.5.3. ***Test dans un environnement libre d'obstacles***

Dans un environnement libre d'obstacles, c'est le champ de température qui domine l'action.

1) Le maximum du champ de température est en avant du robot , donc il se déplace jusqu'à une nouvelle situation où le maximum du champ de température est à sa droite.
2) Il tourne à droite et va en avant à une autre situation où le maximum du champ de température devient à sa gauche, donc il tourne à gauche et continue à se déplacer, ainsi de suite.

On remarque que plus le robot s'approche de la cible plus le changement de situation pour la cible est fréquent.

5.3.2. TEST DANS L'ENVIRONNEMENT D'APPRENTISSAGE

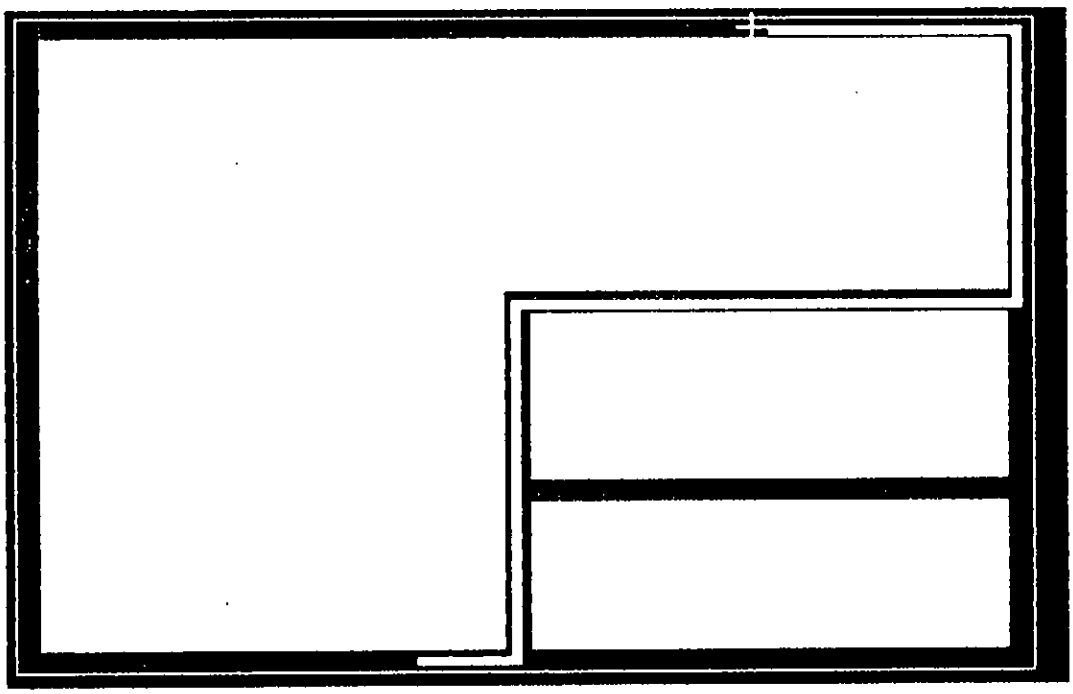


Fig .5.2. *Test dans l'environnement d'apprentissage*

1) Etant donné un couloir, le robot se déplace en avant , c'est la seule possibilité.
2) Arrivé à une situation où il a le choix entre deux actions (aller en avant ou tourner à gauche), c'est la situation T qui domine , donc il tourne à gauche .
3) Il se trouve ensuite dans un couloir il se déplace en avant , jusqu'à atteindre la situation où il a deux possibilités, il continue son chemin en avant car la cible se trouve à son avant.
4) Arrivé à un virage à droite, il doit tourner à droite.
5) Il se trouve à nouveau dans un couloir, il se déplace en avant jusqu'à où il atteint la situation où il a le choix entre deux possibilités (tourner à droite ou à gauche), il tourne à gauche ,c'est T qui domine l'action. Le robot se trouve dans un nouveau couloir, il doit se déplacer en avant.
6) Arrivé à un virage à gauche, il doit tourner à gauche et va en avant jusqu'à atteindre la cible.

5.3.3. TEST DE GENERALISATION

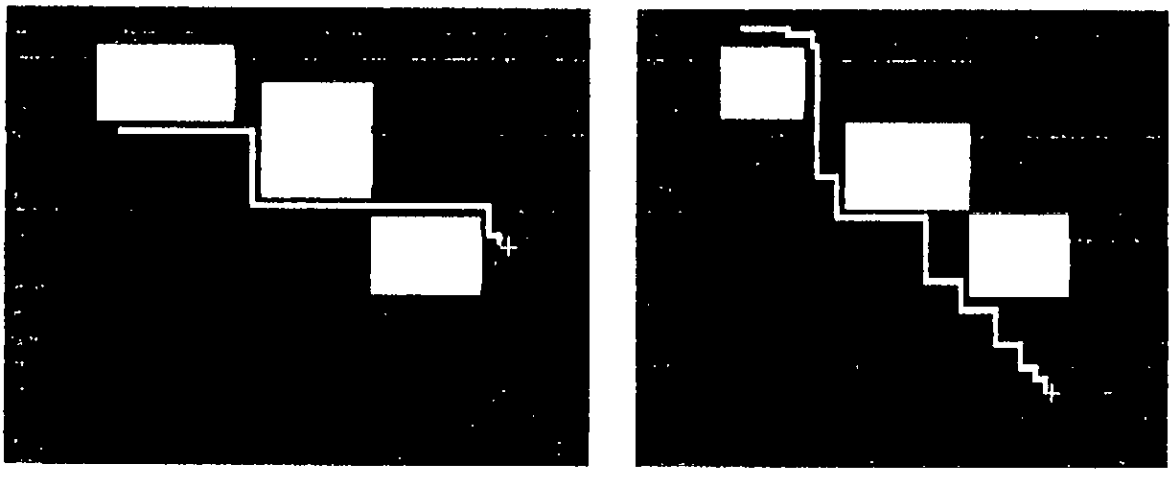


Fig.5.4. Test de généralisation (3 obstacles).

1) Dans le premier schéma le maximum du champ de température est en avant, donc, le robot se déplace en avant.

Arrivé à une situation où il rencontre un obstacle, il tourne à droite évitant l'obstacle et s'avance.

Il se trouve dans une situation où il n'a pas de contraintes vis à vis les obstacles, le robot tourne à gauche, car la cible se trouve à sa gauche. et se déplace contournant le dernier obstacle puis se dirige vers la cible.

2) Dans le deuxième schéma, la cible se trouve à droite du robot, mais dés qu'il tourne à droite il détecte l'obstacle. D'où il tourne à gauche et se déplace en avant.

Le champ de température à droite devient de plus en plus important à chaque fois que le robot fait un pas en avant. Mais, dés qu'il essai de tourner, il s'est trouvé gêné par l'obstacle.

Arrivé à une situation où il trouve une ouverture à droite, il tourne et continue à se déplacer en avant (attiré par la cible), en même temps le champ de température augmente de plus en plus à sa gauche , ce qui fait tourner le robot à gauche et continue à se diriger vers la cible réalisant un compromis entre les actions d'anti-collision et l'orientation vers la cible.

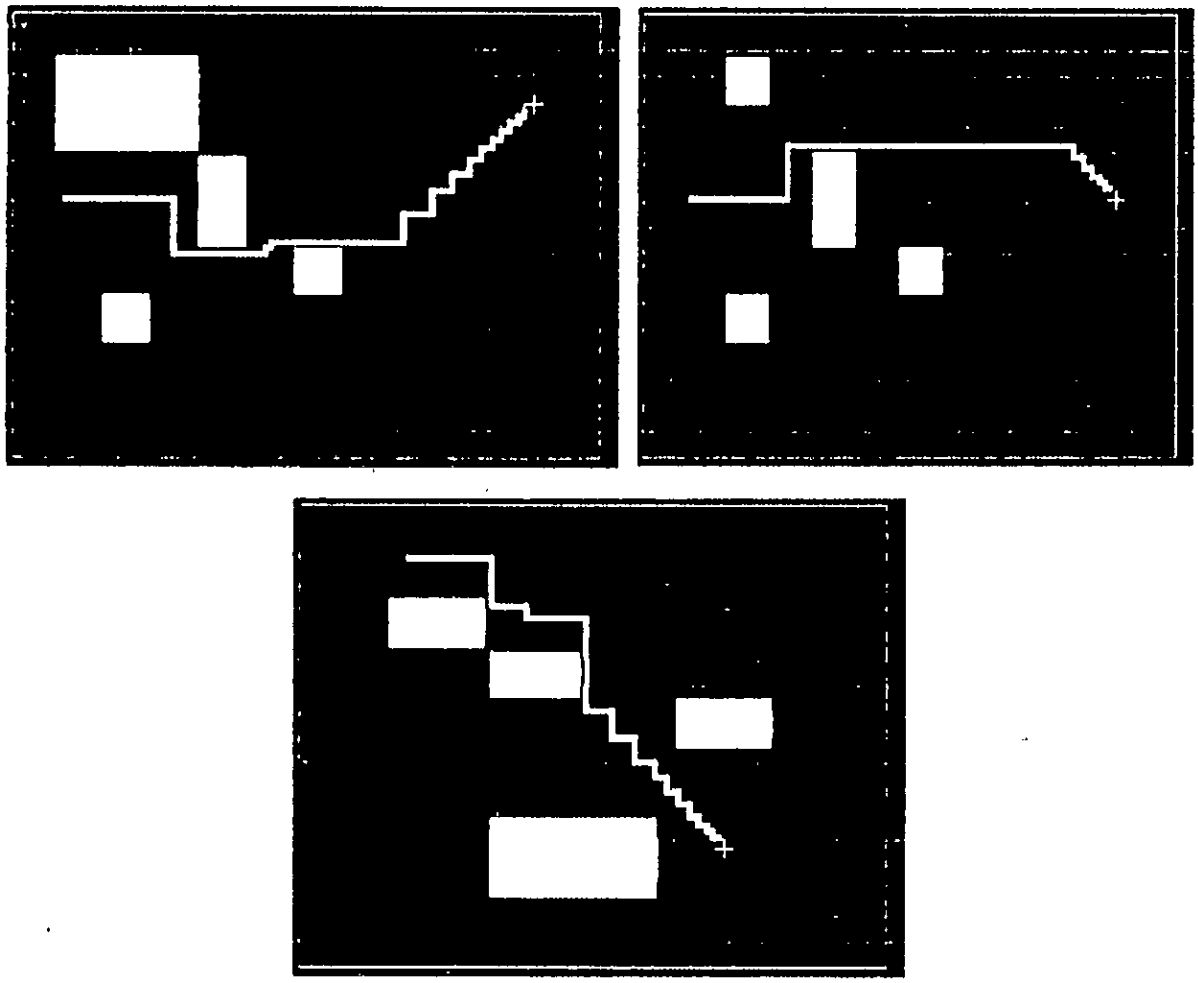


Fig.5.5. ***Test de généralisation (4 obstacles).***

Ces environnements représentent un test au robot dans des milieux partiellement structurés.

Dans ces cas, le robot se dirige vers la cible lors qu'il n'y a aucune contrainte vis à vis les obstacles; il est dévié de son chemin lors de sa rencontre d'un obstacle. Il a réagit d'une manière satisfaisante pour toutes les situations.

Donc le robot a accomplit sa tâche interprétée comme son reflex pour la stratégie anti-collision et son but d'atteindre la cible.

On remarque que le robot a généralisé la navigation pour des nouveaux milieux partiellement structurés, évitant les obstacles comme un reflex et arrivant à la cible comme un but.

5.4. CONCLUSION

Le choix d'un environnement partiellement structuré d'une part et de la navigation par apprentissage d'autre part , nous a imposé en quelque sorte l'environnement d'apprentissage opportun.

Le système global que nous avons développé a permis au robot de coordonner les deux tâches concernant la recherche du maximum du champ de température et de l'évitement d'obstacles en même temps. Cependant les actions générées par le contexte relatif à l'anti-collision peuvent être interprétées comme le reflex du robot mobile et doivent avoir priorités sur les actions générées pour atteindre le maximum du champ de température.

Testé dans l'environnement d'apprentissage, le robot a réussit à éviter convenablement les obstacles de forme rectangulaire ou d'assemblage rectangulaire et atteindre la cible.

Testé dans de nouveaux environnements, le robot a réussit de contourner les obstacles pour atteindre la cible. Ce qui montre que le robot a acquis, par apprentissage, un comportement lui permettant à la fois d'éviter les obstacles et se diriger vers la cible. Ce comportement illustre bien la capacité de généralisation des réseaux de neurones.

CONCLUSION GENERALE

L'objectif de notre travail était l'implémentation orientée hardware des réseaux de neurones sur des réseaux systoliques, avec une application à la navigation d'un robot mobile autonome, et de simuler cette structure par la technique orientée objet ; Ce que nous avons réalisé grâce à un programme dont le code est en C++.

En passant par des transformations de régularisation, nous avons montré que la dérivation d'une architecture systolique est directe pour les algorithmes manipulant des opérations matricielles ou vectorielles.

L'architecture systolique offre un traitement intensif et en pipeline distribué sur plusieurs processeurs et dans laquelle les données circulent par des techniques de décalage.

L'intérêt principal de l'architecture choisie est qu'elle est programmable, c'est à dire que l'architecture physique du réseau permet de traiter les opérations des deux phases par la même configuration et avec les mêmes processeurs, tout en incorporant les opérations nécessaires pour les deux phases, relaxation et apprentissage, dans les *PEs*. En effet au lieu de fixer les poids synaptiques, l'implémentation de la faculté d'apprentissage dans le hardware du système lui donne la possibilité de répondre à d'autres types d'environnement. D'autres part, la taille physique de l'architecture, c'est à dire le nombre de *PEs* utilisés, ne doit pas être obligatoirement égal au nombre de neurones ; Ce qui permet de partitionner le traitement d'un grand problème en traitements de sous problèmes (traitement par lot) implémentés directement sur la structure physique de l'architecture.

L'efficacité du traitement, de point de vue rapidité, augmente avec le nombre de PEs implémentés travaillant en parallèle et/ou en pipeline. On s'est limité à trois processeurs qui travaillent en pipeline vu les exigences de l'application sous considération.

L'organisation des poids dans les mémoires reste inchangée dans les deux phases ; Ce qui offre un grand avantage et facilite largement l'implémentation systolique des *RNAs*.

La structure du réseau de classification, utilisant l'algorithme *RPG* à apprentissage supervisé, est commune pour les deux phases de traitement" propagation et retro-propagation". Dans la phase de fonctionnement (application) on fait appel seulement aux fonctions relatives à la phase de relaxation (phase de propagation dans le cas de l'algorithme *RPG*).

Par contre, dans le bloc d'action, une fois l'apprentissage renforcé est terminé, les processeurs spéciaux au calcul des poids ne sont pas utilisés(sont mis au repos). La prise d'une décision par le robot, dans cette phase (phase de fonctionnement), nécessite seulement les *PEs* d'action.

Les concepts des techniques orientées objet offrent de nouveaux outils pour le développement du matériel au niveau systèmes. Leurs applications offrent de nouvelles possibilités pour la conception hardware/software des systèmes digitaux implantés. Ces modélisations, qui utilisent l'abstraction des données, permettent une fertilisation réciproque entre les solutions hardware et software.

Dans ce cadre nous avons construit deux grandes classes de réseaux, une classe dite réseau classificateur et une autre dite réseau actionneur afin de construire la classe contrôleur qui permet d'assurer la navigation d'un robot mobile autonome pour atteindre sa cible dans un environnement partiellement structuré.

Les résultats des simulations ont été concluants ; Ce qui montre que le système neural développé constitue un contrôleur robuste et intelligent permettant de guider un robot mobile autonome dans un milieu partiellement structuré, dans lequel est défini un champ de température localisant la position de la cible à atteindre.

L'inconvénient majeur de la simulation logicielle, par la technique orientée objet, du système matériel développé est que le parallélisme temporel existant lors de l'exécution des opérations, qui est inhérent aux réseaux de neurones, ne peut pas être observé sur une machine séquentielle. De même, pour la notion de temps simulé, existante dans autres langages de description du hardware comme le *VHDL*, n'est pas encore introduite dans cette technique. La notion de concurrence reste l'une des objectifs de la programmation orientée objet appliquée à la modélisation du hardware.

BIBLIOGRAPHIES

[1] U.RAMACHER," Guide lines to VLSI Design of Neural Nets", in *VLSI design of neural networks*, edited by U.RAMACHER & U.RUCKERT, Kluwer Academic Publishers, 1991.

[2] S.KUMAR, J.H.AYLOR, B.W.JOHNSON, W.A.WULF, "Object-Oriented Techniques in Hardware Design", IEEE Computer, Vol.27 N.6, June 1994, PP 64-70.

[3] S.KUMAR, J.H.AYLOR, B.W.JOHNSON, W.A.WULF, " The Co-Design of Embedded Systems, A Unified Hardware/Software Representation", Kluwer Academic Publishers, USA, 1996.

[4] J.A.FREEMAN, D.M.SKAPURA, *Neural networks applications and programming techniques*, edition Adison- Wesley publishing company, 1992.

[5] E.DAVALO, P.NAIM, *Des réseaux de neurones*, édition Eyrolles, Paris, 1992.

[6] A.NASRI, S.OULAD NAOUI, " Conception et réalisation d'un système de recherche documentaire utilisant l'approche connexionniste", mémoire de Projet de Fin d'Etude d'ingénieur, Institut National d'Informatique, Alger, 1996.

[7] W.S.MC. CULLOCH and W.PITTS "A logical calculus of idea immanent in nervous activity", Bull.Mathematical Bio-Physics, vol 5, PP 115-133, 1943.

[8] A.BENKRID, R.BOUBERTAKH, "Implémentation des réseaux de neurones sur un PC, application à la reconnaissance de l'image", mémoire de Projet de Fin d'Etude d'ingénieur, Département d'Electronique, Ecole Nationale polytechnique, Alger, Sep 1996.

[9] F.Rosemblett, *Principales of neurodynamics*, New York : Spartan 1962.

[10] J.N.HWANG, S.Y.KUNG, "An unified systolic architecture for artificial neural networks" in *VLSI design of neural networks*, edited by U.RAMACHER & U.RUCKERT, Kluwer Academic Publishers, 1991.

[11] S.Y.KUNG, J.N.HWANG, "Neural networks architectures for robot applications", IEEE Transaction on Robotic and Automation, Vol.5, N°.5, October 1989.

[12] J. OUALIER, S.SAUCIER , TRITHE, " Fast design of digital dedicated neurochops" in *VLSI design of neural networks*, edited by U.RAMACHER &U.RUCKERT, Kluwer Academic Publishers 1991.

[13] JAIME H.MORENO TOMASLANG, *Matrix computation on systolic type arrays*, Kluwer Academic Publishers, 1992.

[14] P.QUINTON, YUES ROBERT, *Algorithmes et architectures systoliques*, édition MASSON,1992.

[15] P.MURTAGH AC TSOI, "Implémentation Issues Of Sigmoid Function And Its Derivate For VLSI Digital Neural Networks", IEEE Proceeding-E, Vol N° 5, N° 4, July 1994.

[16] E.SOROUCHYARI, "Mobile Robot Navigation: A Neural Network Approach", Art Coll.Neuro, Ecole Polytechnique de Lausanne, pp159-175, 1989.

[17] A.CHOHRA, " Rapport de recherche sur la navigation d'un robot mobile autonome par l'approche neuronale", Attaché de recherche au CDTA, Algers, 1996.

[18] A.CHOHRA, A.FARAH, R.BENABBAS, "Neuro-Fuzzy Navigation Approach for Autonomous Mobile Robots in Partially Structured Environments", in Proceeding of Second International Conference on Application of Fuzzy Systems and Software Engineering (ICAFS-96), Siegen, Germany, June 25-27, 1996.

[19] A.CHOHRA, A.FARAH, "Hybrid Navigation Approach Combining Neural networks And Fuzzy Logic For Autonomous Mobile Robots.", The third International Conference On Motion And Vibration Control, Chiba, Japan, Sep 1-6, 1997.

[20] G.DE.MICHELI, *Synthesis and Optimisation of Digital Circuits*, M.C.GRAW, INC USA, 1994.

[21] C.DELANNOY, *Programmer en langages C++*, édition Eyrolles, 1995.

[22] C.DELANNOY, *Programmer windows avec le turbo C++*, édition Eyrolles, 1992.

[23] FUJIMOTO, Y.N.FUKUDA, T.AKABANE, " Massively parallel architecture for large scale neural networks simulations", IEEE Transactions on Neural Networks Systems, 3-6, 876-88.1992.

[24] S.Y.KUNG," Parallel architecture for artificial neural networks ", International Conference on Systolic Arrays, Computer Society Press, May 25-27, 1988.

[25] S.LIANGYONG, "VLSI methodologies for artificial neural networks", PhD candidate, university of Waterboo, Waterboo, Ontario, Canada, Sep 1990.

[26] VAN-KEULEN, E.S. COLAK, H. WITHAGEN, H.HEGT," Neural networks hardware performance criteria", Proceeding of the IEEE International Conference on neural networks, June 28- July 2, P1985-P1988, 1994.

ملخص

نتناول فى هذا العمل منهجية لغرس الشبكات العصبونية الاصطناعية على شكل شبكات انقباضية رقمية. بنية على شكل حلقة انقباضية ' التي تعتبر منهجية ممكنة للغرس العتادي للشبكات العصبونية و التي توفر حسا ب مكثف و فعال' استعملت لغرس خوارزمية الانتشار الرجعي للخطا مع تطبيقها لمراقبة آلة متحركة و مستقلة ذاتيا. كلتا المرحلتين، مرحلة الا نسياب، ومرحلة التعلم ادمجتا في التصميم الذي يستعمل فوائد البرمجة الموجهة للشئ في التصميم العتادي. البرنامج مكتوب بلغة الحاسوب المسماة C++

كلمات مفاتيح :االشبكات العصبونية الاصطناعية ' غرس رقمي ' شبكات انقباضية ' آلة متحركة و مستقلة ذاتيا ' البرمجة الموجهة للشئ.

ABSTRACT

In this work we present a digital systolic arrays issue for implementing artificial neural network. A ring systolic architecture, which is a possible way for hardware implementation of ANNs and provides intensive and efficient computing structure, is used to implement the error back-propagation algorithm with application to control an autonomous mobile robot. Both retrieving and learning phases are integrated in the design which use the benefits of Oriented-Object Techniques in hardware design. The code is written in C++.

Key words: Artificial Neural Networks, Digital Implementaion, Systolic Architecrure, Autonomous Mobile Robot, Oriented Object Techniques.

RESUME

Dans ce travail nous présentons une approche digitale sous forme de réseaux systoliques pour l'implémentation des réseaux de neurones artificiels. Une architecture en anneau systolique est développée pour l'implémentation matérielle des RNAs. Elle fournit une structure de calcul intensif et en pipeline, pour implémenter l'algorithme de la rétro-propagation du gradient dans le but de contrôler un robot mobile autonome. Les deux phases, de relaxation et d'apprentissage, sont intégrées dans la conception qui utilise les avantages des techniques orientées objet dans la conception matérielle. Le code est écrit en C++.

Mots clés: Réseaux de Neurones Artificiels, Implémentation Digitale, Architecture systiolique, Robot Mobile Autonome, Techniques Orientées Objet.