الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DR. YAHIA FARES DE MEDEA

FACULTE DE TECHNOLOGIE

Département du Génie Électrique

Mémoire de Fin d'Études

Présenté pour l'obtention du Diplôme
De Master en Génie Electrique

Option : Systèmes de télécommunications

THEME

DEVELOPPEMENT D'UNE ARCHITECTURE SYSTOLIQUE POUR L'ALGORITHME NEURONAL DE RETRO PROPAGATION DU GRADIENT : APPLICATION A LA QUANTIFICATION

Proposé et dirigé par :

Dr Laidi Kamel

Présenté par :

Mme : Souna Sarah

Melle : Zoubiri Lyna

Dédicace

Je dédie ce modeste travail avec un grand plaisir, à tous ceux qui ont crus en moi, spécialement à ceux qui ont été mes guides dans la vie :

A mes chers parents qui m ont entouré de leur amour, leurs sacrifices, leurs protection et de leur générosité durant toute la durée de mes études.

A tous mes enseignants depuis mon premier pas a l'école jusqu'aujourd'hui, qui mon enrichie de leur savoir

A mes très chèrs frères ***Farés*** et ***Yacine*** et ma sœur **imene**, pour leurs chaleureux encouragements tout au long de la période de mes études

Merci d'être existé dans ma vie

A mon binôme ***Sarah*** qui m a toujours accepté tel que je suis, et m' a si souvent remonté le morale…….ensemble pour toujours

A mes copines et copains de ma promotion ***kawther abla selma*** et ***widad***, pour tous les bons moments partagés, pour tant de gentillesse et de disponibilité.

Enfin, a tous ceux sont encrés dans mon cœur ceux que je ne peux pas citer et que le manque de place ne permet pas les nommer ici.

Merci pour les moments grandioses passés et a venir.

Zoubiri Lyna

Dédicaces

A mes très chers parents pour leur tendresse, leurs soins vigilants, leurs sages conseils, leur patience, leurs bons exemples et leur dévouement. Que puisse dieu les darder éternellement heureux.

A ce qui va partager son avenir avec moi

A mes sœurs

A mon frère

A mes beaux frères

A mes tantes et mes oncles

A mes neveux et mes nièces

A ma belle famille

A mes camarades de classe et mes amies exceptionnellement mon binôme Lyna a Widad , Selma, Imen, Marwa et Fayza.

A vous tous …

Je dédie ce mémoire

Souna Sarah

Remerciements

Tout d abord, nous tenons à remercier **Allah**, le tout puissant qui nous avoir donné la force et le courage de mener a bien ce modeste travail

Nos vifs et sincères remerciements s'adressent aussi à notre promoteur et encadreur **Mr Laidi Kamel** pour les
Précieux conseils et ses aides

Nous remercions également,
Les membres de Jury
De nous avoir fait l'honneur de juger ce travail,
Veuillez accepter l'expression de notre vive gratitude
Et notre parfaite déférence.

Nos plus respects s'adressent aussi à tout le cadre professoral et administratif de notre Université **Dr. YAHIA FARES**

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de prés ou de loin à la réalisation de ce travail

ملخص :

نتناول في هذا العمل منهجية لغرس الشبكات العصبونية الاصطناعية على شكل شبكات انقباضية رقمية.بنية على شكل حلقة انقباضية.التي تعتبر منهجية ممكنة للغرس العتادي الشبكات العصبونية و التي توفر حساب مكثف و فعال. و لهدا الغرض لقد قسمنا عملنا إلى جزئين جزء التعلم و دوره حساب الأثقال و جزء لنمذجة مكونات الأجهزة الذي يستعمل فوائد البرمجة الموجهة في التصميم العتادي.البرنامج مكتوب بلغة الحاسوب المسماة C++

الكلمات المفتاحية : الشبكات العصبونية الاصطناعية , شبكات انقباضية, التعلم , البرمجة الموجهة الشيء.

Résumé :

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.
On a divisé notre étude en deux grandes étapes, l'étape d'apprentissage sert à calculer les synapses et l'étape pour la modélisation des composants hardware qui utilise les avantages des techniques orientées objet. Le code est écrit en C++.

Les Mots clés : Réseaux de neurones artificiels, apprentissage, architecture systolique, techniques orientées objet

Abstract :

In This Work we présent a digital systolic array issue for implementing artificial neural network. A ring systolic architecture, winch is a possible way for hardware implementation of ANNs and provides intensive and efficient computing structure. We have divided our study into two main stages, the learning step used to calculate the synapses, and the step for modeling hardware components that uses the advantages of object-oriented techniques. The code is written in C ++.

The Keywords : Artificial neural network, Learning, systolic architecture, oriented-Object techniques

Table de matière

Liste de figure I

Liste des abreviations II

Liste des symboles III

Liste des tableaux IV

I. Introduction general 1

Chapitre1 : Réseaux de neurones artificiels

1.Introduction 3
2. Historique 3
3.Definition 4
4.Neurone biologique 5
4.1 Description de neurone 5
4.1.1 Le corp cellulaire 6
4.1.2 Les dendrites 6
4.1.3 L'axone 6
4.2 La synapse 6
4.3 Principe de fonctionnement 6
5. Neurone formel 7
5.1 Définition 7
5.2 Synoptique 7
5.3 Fonctionnement 8
6. Architecture 9
6.1 Réseaux multicouches 9
6.2 Réseaux à connexions locales 10
6.3 Réseaux de neurones récurrents 10
6.4 Réseaux de neurone à connexion complète 11
7. Apprentissage des réseaux de neurone 11
7.1 Définition 11
7.2Types d'apprentissage 11
7.2.1 Apprentissage supervisé 11
7.2.2 Apprentissage non supervisé 12
7.2.3 Apprentissage par renforcement 12
8. Apprentissage dans quelques réseaux célèbres 12
8.1 Perceptron 12
8.2 Réseaux multicouches : retro propagation du gradient 14
8.3 Adaline 18
8.4 Réseaux de hopfield 19
8.5 Réseaux de kohonen 19
8.6 Réseaux RBF 20
9. Les étapes de la conception d'un réseau 21

9.1 Les choix des échantillons 21
9.2 Élaboration de la structure de réseau 21
9.3 Apprentissage 22
9.4 Validation et tests 22
10. Les avantages et les inconvénients 22
11. Conclusion 23

Chapitre 2: Réseau systolique et architecture digitale des RNAs
1. Introduction 24
2. Définition 24
3. Champs d'application des réseaux systoliques 25
4. Description des unités opératrices 25
5. Fonctionnement générales 27
6. Différents topologies de réseaux systoliques 28
7. Propriétés des architectures systolique 28
7.1 La régularité 28
7.2 La modularité 28
7.3 Le parallélisme 29
7.4 Communications locales 29
8. Architecture systolique des RNAs 30
9. Représentation des opérations matricielles par les graphes de dépendances 30
10. Régularisation des algorithmes 32
10.1 Régularisation de l'algorithme de la phase de relaxation 33
10.2 Régularisation de l'algorithme de la phase d'apprentissage 35
11. Architecture en anneau systolique des RNAs 38
11.1 Processeur élémentaire 38
11.2 Anneau systolique pour la phase de relaxation 39
11.3 Anneau systolique pour l'apprentissage 40
12. Conclusion 42

Chapitre 3 : Implémentation et Simulation
1. Introduction 43
2. Rappel sur le modèle feed-forward 43
3. Architecture d'implémentation du module feed-forward 44
3.1 Unité opératrice 45
3.1.1 La couche 45
3.1.2 Le neurone 45
3.1.3 Le bloc des poids synaptiques 46
3.1.4 Le bloc MAC 46
3.1.5 Le bloc d'activation 47
4. Apprentissage du réseau neurone 47
5. Interprétations 48
6. Choix du langage de modélisation 52
6.1 Motivation pour les techniques orientées objet 52
6.2 Type de donnée 53
6.3 Modélisation des composants hardware comme classes 53
6.4 Dérivation de composants spécialisés 53
6.5 Modélisation orientée objet des RNAs 54

6.6 Modélisation des classes du RNA 55
6.6.1 Les classes registres 55
6.6.2 La classe PE 55
7. Résultats et interprétations 56
7.1Environnement de développement 56
7.2 Les étapes suivent pour construire le programme 56
7.3 Simulation et résultats 56
8. Conclusion 57

Conclusion générale 58
Bibliographie
Annexe

LISTE DES FIGURES

Figure	pages
Fig 1.1 : modèle de neurone biologique	5
Fig 1.2 : neurone formel	8
Fig 1.3 : fonctions d'activations	9
Fig 1.4 : réseau multicouche	10
Fig 1.5 : réseau a connexions locales	10
Fig 1.6: réseau a connexion récurrents	11
Fig 1.7 : réseau a connexions complète	11
Fig 1.8 : le perceptron avec seuil	12
Fig 1.9 : le perceptron avec entrée supplementaire	13
Fig 1.10 : :reseaux RBF	20
Fig 2.1 : Principe d'une architecture systolique	23
Fig 2.2 : differentes cellules constituants le réseau systolique	24
Fig 2.3 : architecture de PE1	25
Fig 2.4: architecture de PE2	26
Fig 2.5 : reseau systolique carée ,triangulaire et hexagonal	27
Fig 2.6 : reseau de processeurs elementaires a)sans communications locales et b)avec communications locales	28
Fig 2.7 : graphe de dependances de la multiplication vecteur_matrice(V*B)	30
Fig 2.8 : graphe de dependances de la multiplication de deux matrices(A*B)	30
Fig. 2.9 : etape necessaire pour l implementation d un algorithme RPG	31
Fig. 2.10 : projection de DDG de calcul du potentiel	33
Fig 2.11 : anneau systolique pour le calcule du potentiel	33
Fig 2.12 : projection du DDG de calcul de $\&_i(1)$	36
Fig 2.13 : projection du DDG de la mise a jour des poids	36
Fig 2.14 : PE utilisé par l algorithme	37
Fig 2.15 : architecture en anneau systolique pour l'operation OPU	40
Fig 3.1 : Propagation des états de la partie Feed-Forward	44
Fig 3.2 **:** Architecture du module Feed-Forward	44
Fig 3.3 : Architecture de la couche	45

Fig 3.4 : Architecture du neurone 46
Fig 3.5 : Le bloc des poids synaptiques 46
Fig 3.6 : Architecture de bloc MAC 47
Fig 3.7 : modèle de test de simulation 48
Fig 3.8 : évolution de l erreur d'apprentissage MSE 50
Fig 3.9 : courbe de convergence y =f(x) 56

Liste des abréviations :

ADALINE: ADAptive LINar Element

CAO: conception assisté par ordinateur

CSA : Carry Save Adder

FA : Full Adder

ITGE: Intégration à Très Grande Échelle

LSI: Large Scaled Integrated

MAC: Multiplication And Accumulation

MIMD: Multi Instruction Multi Data

MLP: Multi Layer Perceptron

MVM: Matrrix-Vector Multiplication

OPU: Outer Product Updating

PE: Processeurs Elémentaires

PGCD: Plus Grand Diviseur Commun

RBF: Radial Basis Fonction

RNA: Réseaux de Neurones Artificiels

RPG : Rétro propagation du gradient

SIMD: Single Instruction Multi Data

TDNN: Time Delay Neural Network

TOO : Techniques orientées objet

VLSI: Very Large Scaled Integrated

Liste des symboles

$\mathbf{X_J}$	signal d'entrée
$\mathbf{W_{IJ}}$	les poids synaptique
$\mathbf{W_0}$	coefficient synaptique
F	fonction d'activation
θ	Seuil de neurone
P	potentiel de neurone
$\mathbf{D_I}$	signal de sortie désiré
$\mathbf{Y_J}$	sortie actuelle du j ^{eme} neurone
E	erreur globale
e	**e**rreur locale
η	Coefficient d apprentissage
$\mathbf{C^K}$	discriminateur
a	activation pondéré
S	somme pondérée des entrées d'un neurone

MSC: Mobile Switching Center

MOS: Mean Opinion Score

MMS: MultiMedia Messaging Service

MBE: Multi Band Excited

MELP: Mixed Excitation Linear Prediction

MPE-LPC: Multi Pulse Excited- Linear Prediction Coefficients

NSS: Network Switching Subsystem

NIST: National Institute of Standards and Technology

OSS: Operating Sub-System

OTAN: Organisation du Traité de l'Atlantique Nord

PLP: Perceptual Linear Prediction

PPCA: Probabilistic Principal Component Analysis

PLDA: Analyse discriminante linéaire probabiliste

PS: Paquet Switched

PCM: Pulse Code Modulation

PSTN: Public Switched Téléphone Network

1G: Première Génération-technologies de communication radio mobile

4G: Quatrième Génération-technologies de communication radio mobile

RAL: Reconnaissance Automatique du Locuteur

RAP: Reconnaissance Automatique de la Parole

RPE: Regular pulse excitation

RPE-LTP: Regular pulse excitation-long term prediction-linear predictive

RNC: Radio Network Controller

$R_{Eb}/_{N0}$**:**Rapport Signal sur Bruit

SVM: Support Vector Machine

SAC: Similarité Angulaire De Cosinus

SBC: Sub-Band Coding

STC: Sinusoidal Transform Coding

SVQ: Split Vector Quantization

S-norm: Symmetric-Normalization

TACS: Total Acces Communication System

TDMA: Time Division Multiple Accès

T-norm: Test-Normalisation

3GPP: Third Generation Partnership Project

3G: Troisième Génération-technologies de communication radio mobile

UBM: Universal Background Model

UMTS: Universal Mobile Telephone System

UIT: Union Internationale des Télécommunications

VAL: Vérification Automatique du Locuteur

VoIP: Voice over Internet Protocol

VSELP: Vector Sum Excited Lineary Prédiction

VAD: Voice Activity Detection

WCCN: Within Class Covariance Normalization

W-CDMA: Wideband-Code Division Multiple Access

WI: Wave Interpolation

Z-norm: Zéro-Normalization

Liste des tableaux

Tab 2.1: valeur de h_i et g_i pour l'algorithme RPG **35**
Tab 3.1 : Numérisation de la fonction logarithmique. **48**
Tab 3.2 : Valeurs pour la couche cachée **50**
Tab 3.3 : Valeurs pour couche sortie **50**
Tab 3.4 : Valeurs de sortie désirée **d** et sortie obtenue **s** **57**

INTRODUCTION GENERALE

Introduction générale :

Le monde de l'informatique a beaucoup gagné des réseaux de neurones. En outre, il n'est pas nécessaire de concevoir un algorithme pour effectuer une tâche spécifique; C'est-à-dire qu'il n'est pas nécessaire de comprendre les mécanismes internes de cette tâche. Ils sont également très bien adaptés pour les systèmes en temps réel en raison de leur réponse rapide et des temps de calcul qui sont dues à leur architecture parallèle.
Actuellement, le domaine d'application des réseaux de neurones artificiels est très répondu : Robotique, traitement d'image, génie industriel, traitement de signal,…etc.

A ce large domaine d'application, plusieurs mises en œuvre matérielles de réseaux de neurones ont été réalisées. Elles diffèrent selon le nombre de neurones, la précision, le nombre de synapses et la vitesse du processus d'apprentissage.

L'architecture systolique est l'une des meilleures architectures parallèles pour la mise en œuvre des réseaux de neurones. Une architecture systolique est agencée en forme de réseau. Ces réseaux se composent d'un grand nombre de cellules élémentaires identiques et localement interconnectées. Chaque cellule reçoit des données en provenance des cellules voisines et effectue un calcul simple puis transmet les résultats aux cellules voisines. Les cellules évoluent en parallèle sous le contrôle d'une horloge globale (synchronisme total).Plusieurs calculs sont effectués simultanément sur le réseau.
L'évolution rapide de la technologie et spécialement les outils de C.A.O ont rapporté de nouvelles techniques de developpement du matérielle 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 consiste 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 type de données définis par l'utilisateur(ou classes).
Dans ce mémoire, nous sommes intéressés par l'implémentation d'un réseau de neurones artificiel en utilisant les réseaux systoliques comme architecture et la technique de programmation orientée objet(T.P.O.O) comme outil de simulation et validation des résultats.

Pour bien cerner le travail qui nous a été proposé, notre mémoire est présenté comme suit:

Le premier chapitre est consacré à la présentation des réseaux de neurones artificiels. Tout d'abord, nous présentons l'élément de base « neurone » et les caractéristiques du réseau de neurones. Par la suite, nous nous intéresserons aux réseaux de neurones multicouches MLP.

Le deuxième chapitre est consacré à la description des architectures systoliques. Tout d'abord nous présenterons les notions générales de l'architecture (définition, champs d'application, principe de fonctionnement et les différentes topologies de l'architecture), en suite nous aborderons leurs propriétés et à la fin nous nous intéresserons à l'architecture systolique des RNAs.

Le troisième chapitre est consacré à la conception d'une architecture d'implémentation et de modélisation du réseau de neurones. Il s'agit de choisir les caractéristiques du réseau de neurones, et par la suite, de concevoir son architecture élémentaire globale.

Une méthodologie pour l'implémentation et la modélisation des réseaux de neurones, basée sur la synthèse et en utilisant les techniques orientées objet a été proposée.

Chapitre I

Réseaux de neurones artificiels

1. Introduction

Les réseaux de neurones sont des modèles théoriques de traitement de l'information inspirés des observations relatives au fonctionnement des neurones biologiques.

Ces dernières années, de nouvelles techniques neuronales d'apprentissage ont été développées. Cet apprentissage avec des réseaux de neurones se fait actuellement en suivant certains algorithmes comme la rétro propagation du Gradient.

Dans ce chapitre nous allons présenter les notions de base pour la compréhension des réseaux de neurones artificiels : (le neurone biologique et artificiel), ainsi que ses caractéristiques, puis nous abordons de manière plus précise l'algorithme d'apprentissage du perceptron multicouches en améliorant son algorithme, la rétro propagation de gradient.

2. Historique

W. James, célèbre psychologue américain introduit le concept de mémoire associative en 1890, et propose ce qui deviendra une loi de fonctionnement pour l'apprentissage sur les réseaux de neurones connue plus tard sous le nom de loi de Hebb. Le champ des réseaux neuronaux va démarrer en 1943 par la découverte par W. McCulloch et W. Pitts du neurone formel qui est une abstraction du neurone physiologique. Le retentissement va être énorme. Par cette présentation, ils veulent démontrer que le cerveau est équivalent à une machine de Turing, la pensée devient alors purement des mécanismes matériels et logiques.

Une machine de Turing se résume à une tête de lecture comportant un nombre fini d'états internes et à un ruban. La puissance de l'analyse de Turing (1912-1954) tient au fait que satête de lecture ne lit qu'un seul symbole à la fois, et que cette lecture, associée à la table d'états adéquate, suffit à effectuer toutes les opérations possibles. La Machine de Turing est toutefois une machine imaginaire, abstraite, et idéale. Elle n'a pas pu être construite. Son fonctionnement implique en effet d'avoir un ruban extensible à volonté donc infini. La combinaison de cette mémoire infinie et d'un nombre d'états fini a cependant apporté la preuve que, si un problème est calculable, alors il existe une machine pour le résoudre.

En 1949, D. Hebb physiologiste américain présente dans son ouvrage «The Organization of Behavior» une règle d'apprentissage. De nombreux modèles de réseaux aujourd'hui s'inspirent encore de la règle de Hebb.

En 1957, F. Rosenblatt développe le modèle du perceptron. C'est un réseau de neurones inspiré du système visuel, et l'applique au domaine de la reconnaissance de formes. Il possède deux couches de neurones : une couche de perception et une couche liée à la prise de décision.

C'est le premier système artificiel capable d'apprendre par expérience.

En 1960, le modèle de L'Adaline (ADAptive LINar Element) a été présenté par B. Widrow et Hoff. Dans sa structure, le modèle ressemble au Perceptron, cependant la loi d'apprentissage est différente. Ce modèle sera par la suite le modèle de base des réseaux multicouches.

En 1969, M. Minsky et S. Papert publient une critique des propriétés du Perceptron.

Limitations alors connues, notamment concernant l'impossibilité de traiter par ce modèle des problèmes non linéaires. Ils étendent implicitement ces limitations à tous modèles de réseaux de neurones artificiels. Leur objectif est atteint, il y a abandon financier des recherches dansle domaine (surtout aux U.S.A.), les chercheurs se tournent principalement vers l'IA et les systèmes à bases de règles.

Jusqu'en 1972, où T. Kohonen présente ses travaux sur les mémoires associatives et propose des applications à la reconnaissance de formes.

C'est en 1982 que J. Hopfield présente son étude d'un réseau complètement rebouclé, dont il analyse la dynamique.

En 1983, la Machine de Boltzmann est le premier modèle connu apte à traiter de manière satisfaisante les limitations recensées dans le cas du perceptron. Mais l'utilisation pratique s'avère difficile, la convergence de l'algorithme étant extrêmement longue (les temps de calcul sont considérables).

1985 : La rétro propagation de gradient apparaît. C'est un algorithme d'apprentissage adapté aux réseaux de neurones multicouches (aussi appelés Perceptrons multicouches). Sa découverte réalisée par trois groupes de chercheurs indépendants indique que "la chose était dans l'air". Dès cette découverte, la réalisation d'une fonction non linéaire d'entrée/sortie sur un réseau en décomposant cette fonction en une suite d'étapes linéairement séparables est devenue possible. De nos jours, les réseaux multicouches et la rétro propagation de gradient restent le modèle le plus étudié et le plus productif au niveau des applications. **[2][3].**

3. Définition

Les réseaux de neurones informatiques sont une partie de l'intelligence artificielle dont l'application s'élargit de jour en jour. Le réseau se compare un peu à une boite noire : on met d'un côté les informations et de l'autre on obtient une décision. Dans un réseau informatique les informations sont en général présentées comme allant de gauche à droite. Chaque neurone collecte les informations venant des neurones précédents et en déduit une information à

transmettre aux neurones suivants. Chaque neurone donne plus ou moins d'importance à certaines des informations qu'il reçoit**. [1]**

4. Neurone biologique

4.1. Description de neurone

Les neurones sont des unités cellulaires de relais et d'interprétation de signaux situés dans le système nerveux et cérébrale de l'humain en un nombre approximatif de 25 milliards à la naissance. Ils sont composés d'un corps cellulaire appelé le soma, où se situent le noyau et toute la machinerie cellulaire, et d'extrémités telles que les dendrites et l'axone [Schmidt, 1985]. C'est le noyau qui est le site de la réaction d'atteinte ou non du seuil lorsque le neurone est stimulé et l'axone est la portion en extrémité qui fait le lien avec d'autres neurones, d'où la notion de relais, par le biais des dendrites des autres neurones.

Fonctionnellement, lorsque des facteurs externes (des hormones, des neurotransmetteurs, une odeur, le touché, etc.) viennent en contact avec les récepteurs situés sur la surface des dendrites du neurone il y a modification physico-chimique dans le soma de la cellule et ou non du seuil d'inactivité de la cellule neuronale ce qui va provoquer l'activité de celle-ci si le seuil est atteint et va activer d'autres neurones par la libération de neurone transmetteurs à l'extrémité de son axone**. [3]**

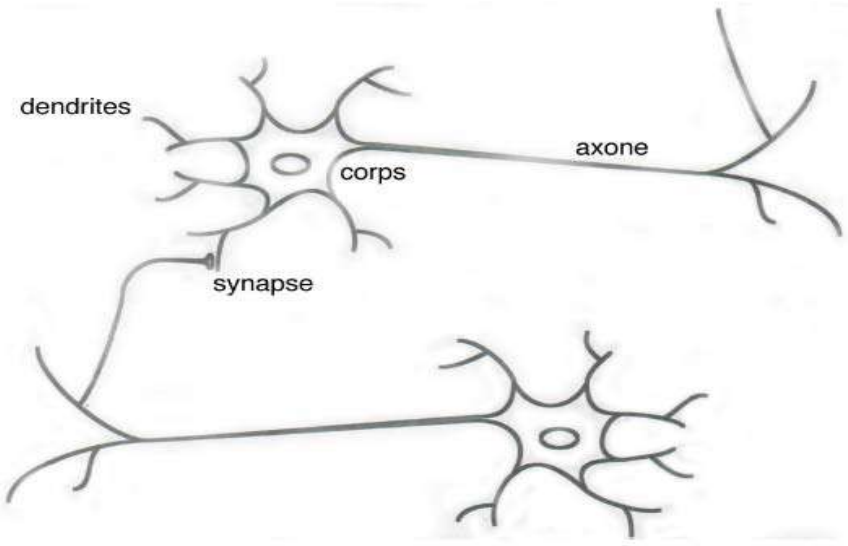


Fig1.1: Modèle de neurone biologique.

Un neurone biologique est constitué de trois parties : le corps, les dendrites et l'axone.

4.1.1 Le corps cellulaire

Il contient le noyau du neurone et effectue les transformations biochimiques nécessaires à la synthèse des enzymes et d'autres molécules qui assurent la vie du neurone. Sa forme est pyramidale ou sphérique dans la plupart des cas, Elle dépend souvent de sa position dans le cerveau, ainsi les neurones du néocortex ont principalement une forme pyramidale. Ce corps cellulaire fait quelques microns de diamètre. **[3]**

4.1.2. Les dendrites

Chaque neurone possède une chevelure de dendrites. Celle-ci est constituée de fines extensions tubulaires, de quelques dixièmes de microns de diamètre et d'une longueur de quelques dizaines de microns. Elles se ramifient, ce qui les amène à former une espèce d'arborescence autour du corps cellulaire. Elles sont les récepteurs principaux du neurone pour capter les signaux qui lui parviennent. **[3]**

4.1.3. L'axone

L'axone est la fibre nerveuse qui sert de transporter les signaux émis par les neurones. Ilse distingue des dendrites par sa forme et par les propriétés de sa membrane externe. En effet, il est généralement plus long que les dendrites et se ramifie à son extrémité, là où il communique avec d'autres neurones, alors que les ramifications des dendrites se produisent plutôt près du corps cellulaire. **[3]**

4.2. La synapse

La synapse est un lieu d'interactions fort complexes où le signal électrique de l'impulsion nerveuse est converti en un signal biochimique. Ce qui nous permet de dire que le rôle des synapses est fondamental pour permettre aux cellules nerveuses de communiquer entre elles. **[3]**

4.3. Principe de fonctionnement

Le fonctionnement classique d'un neurone, consiste en générale à recevoir et émettre des signaux électriques, est cela comme suit :

Au niveau de chaque neurone on a une intégration des signaux reçus (qui parviennent des autres neurones) au cours du temps, c'est en quelque sort une sommation des signaux reçus qui se produit au niveau de chaque neurone, quand cette somme dépasse un certain seuil, ce dernier émet à son tour un signal électrique. Ainsi les neurones communiquent en émettant des trains de potentiels rapides et très courts de l'ordre de quelques millisecondes.

En générale, la membrane externe d'un neurone exécute cinq fonctions :

- Propagation des impulsions électriques le long de l'axone et des dendrites

- Libération des médiateurs à l'extrémité de l'axone

- Réagir à ces médiateurs chimiques au niveau des dendrites

- Réagir au niveau du corps cellulaire aux impulsions électriques issues des dendrites pour générer ou non une nouvelle impulsion

- Permettre au neurone de reconnaître ses voisins pour se situer au cours de la formation du cerveau et trouver à quelles autres cellules il doit ce connecté. **[1]**

5. Neurone formel

5.1. Définition

Un neurone formel est en général une fonction algébrique non linéaire et bornée, dont la valeur dépend de paramètres appelés coefficients ou poids. Les variables de cette fonction sont habituellement appelées « entrées » du neurone, et la valeur de la fonction est appelée sa « sortie ».

5.2. Synoptique

C'est un modèle mathématique très simple dérivé d'une analyse de la réalité des neurones biologiques. Pour le réaliser, on effectue une simple sommation des signaux arrivant au neurone, la sortie est obtenue par comparaison de cette somme au seuil de déclanchement, par exemple, elle peut être égale à 1 si la somme est supérieure au seuil et 0 dans le cas contraire Plus simple, il suffit de soustraire le seuil considéré à la somme des entrées, et de faire passer le résultat par la fonction de transfère, la sortie du neurone est le résultat obtenu après le transfert, ainsi, on obtient une représentation des propriétés physique du neurone biologique. Pour donner plus de puissance à ce modèle, on introduit la connexion synaptique qui est une valeur réelle, la sortie d'un neurone est alors multipliée par la valeur de la connexion synaptique durant le traitement effectué par le neurone récepteur.
En résumer, on peut dire que le neurone formel réalise tout simplement une somme pondérée de ces entrées, ajoute un seuil à cette somme et fait passer le résultat par une fonction de transfert pour obtenir sa sortie. [**4**]

5.3. Fonctionnement

Pour fonctionner, un neurone formel utilise des entrées qui sont des grandeurs réelles. Si on relie un neurone au monde extérieur par des capteurs, il peut réaliser une simple analyse de ce qu'il « perçoit ». Si on représente les valeurs observées par le neurone sous forme d'un vecteur, le neurone réalise alors un découpage de son espace d'entrée (l'espace vectoriel auquel appartient le vecteur d'observation) en deux zones :

- La zone d'activité : dont les vecteurs donnent une sortie égale à 1 ;
- La zone d'inactivité : dont les vecteurs donnent une sortie égale à 0.

Comme le calcul effectué est en fait linéaire, la séparation l'est aussi. Les coefficients synaptiques et le seuil définissent l'équation d'un hyperplan qui est la frontière de la séparation entre les deux zones.

Mathématiquement, le neurone formel est considéré comme une fonction non linéaire, bornée et dont la valeur de sortie dépend de certains paramètres appelés coefficients ou poids. Les entrées de cette fonction représentent les entrées du neurone, tandis que son résultat représente la sortie. **[5]**

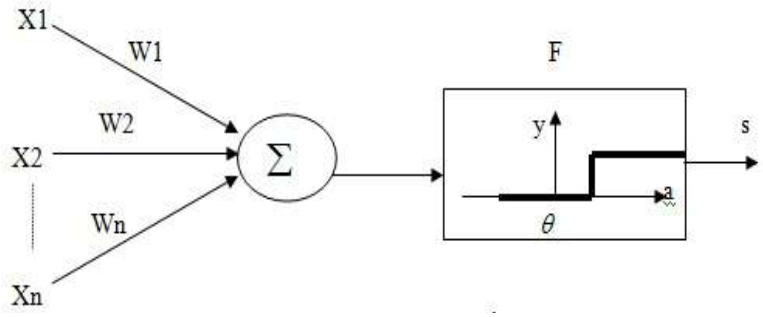


Fig1.2: Neurone formel.

Les Xj sont les n signaux d'entrée, et les W_{ij} représentent les poids qui relient les neurones i, j; L'activation pondérée « a » est donnée par :

$$a=x_1w_{i1}+x_2w_{i2}+\ldots+x_nw_{in}=\sum X_iW_{ij} \qquad Y=F(a) \qquad (1\text{-}1)$$

La sortie finale est donnée par le seuil d'activation comme suit: $Y=\begin{cases}1 & \text{si } a\geq 0\\ 0 & \text{si } a<0\end{cases}$ (1-2)

La fonction d'activation f, détermine l'état interne du neurone en fonction de son entrée totale. Plusieurs modèles de cette fonction existent dans la littérature, on les décrit par la suite :

Fonction Heaviside définie par : $H(x)=\begin{cases}1 \text{ si } x\geq 0\\ 0 \text{ sinon}\end{cases}$ (1-3)

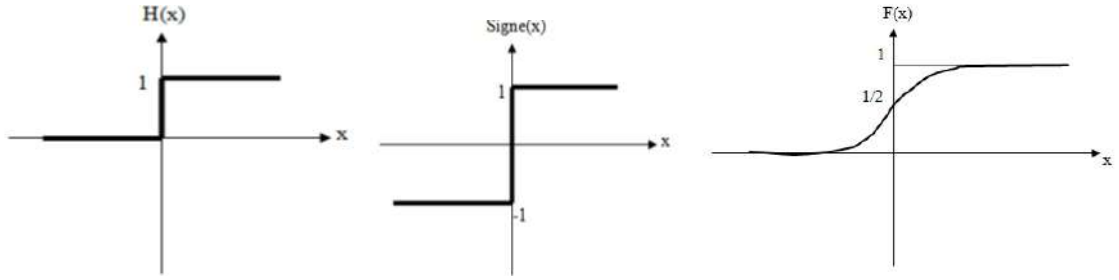
$$\begin{cases} 8 \end{cases}$$

Fonction signe définie par : signe(x)= +1 si x≥0

-1 sinon (1-4)

La fonction linéaire est définie par : F(x)=x (1-5)

La fonction sigmoïde est définit par: $F(x)= \frac{1}{1+\exp(-x)}$ (1-6)



Fonction Heaviside Fonction signe. Fonction *sigmoïde*

Fig1.3:fonctions d'activations

Donc on peut conclure qu'un neurone formel réalise simplement une somme pondérée de ces entrées, ajoute un seuil à cette somme et fait passer le résultat par une fonction de transfert pour obtenir sa sortie.

6. Architecture

Les réseaux neuromimétriques peuvent se répartir en deux grandes catégories : la première est dite « réseaux à couches », la seconde est appelée « réseaux récurrents » qui présentent certaine complexité par rapport à la première.

La connectivité des réseaux peut être totale, c'est-à-dire que tous les neurones sont connectés entre eux ou bien, par couches, et dans ce cas les neurones d'une couche ne sont connectés qu'aux neurones de la couche suivante.**[5]**

6.1. Réseaux multicouches

Réseau multicouche: Il n'y a pas deconnexion entre neurones d'une même couche et les connexions ne se font qu'avec les neurones des couches avales (fig1.4). Habituellement, chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et celle-ci seulement. Ceci nous permet d'introduire la notion de sens de parcours de l'information (de l'activation) au sein d'un réseau et donc définir les concepts de neurone d'entrée, neurone de

sortie. Par extension, on appelle couche d'entrée l'ensemble des neurones d'entrée, couche de sortie l'ensemble des neurones de sortie. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelés couches cachées.

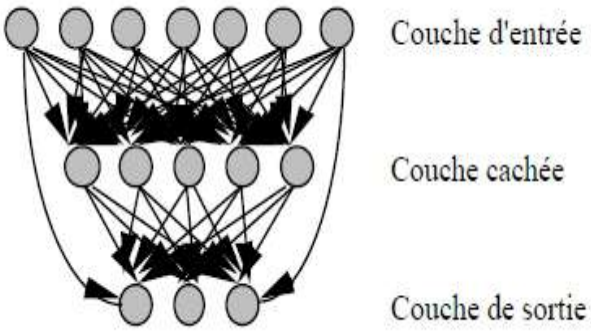


Fig1.4:un réseau multicouche.

6.2. Réseaux a connections locales

Il s'agit d'une structure multicouche, mais qui à l'image de la rétine, conserve une certaine topologie. Chaque neurone entretien des relations avec un nombre réduit et localisé de neurones de la couche avale (fig1.5). Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouche classique.

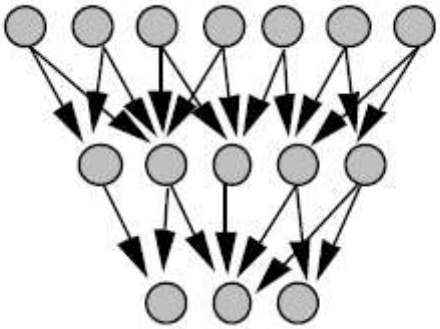


Fig1.5:Réseau à connexions locales

6.3. Réseaux de neurone récurrents

Les connexions récurrentes ramènent l'information en arrièrepar rapport au sens de propagation défini dans un réseau multicouche. Ces connexions sont leplus souvent locales (fig1.6)

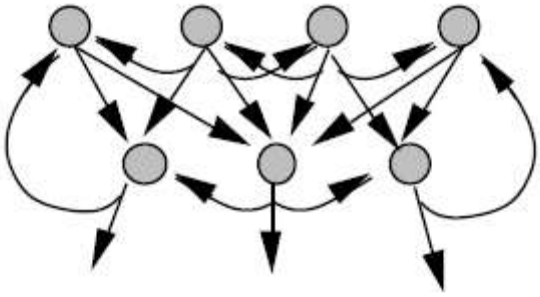


Fig1.6:Réseau à connexions récurrentes.

6.4. Réseaux de neurone à connexion complète

C'est la structure d'interconnexion la plus générale (fig1.7). Chaque neurone est connecté à tous les neurones du réseau (et à lui-même).**[2]**

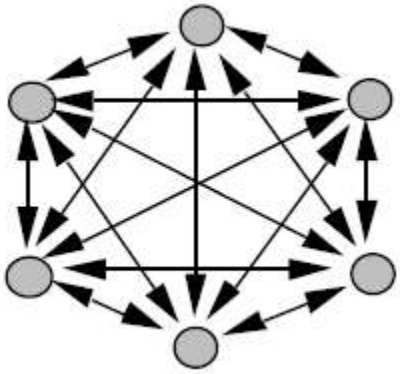


Fig1.7:Réseau à connexion complète.

7. Apprentissage des réseaux de neurone

7.1. Définition

L'apprentissage est une phase de développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré.
On distingue trois grandes classes d'algorithmes d'apprentissage : apprentissage supervisé, apprentissage non supervisé, et apprentissage par renforcement. **[1]**

7.2. Techniques d'apprentissage :

7.2.1. Apprentissage supervisé :

Dans ce cas on fournit au réseau la donnée à traiter mais aussi la sortie attendu. Les résultats corrects (c'est-à-dire les valeurs que l'on désire que le réseau obtienne en sortie) sont fournis au réseau, si bien que celui-ci peut ajuster ses poids de connexions pour les obtenir (la

modification des paramètres internes afin de minimiser l'erreur constatée). Après l'apprentissage, le réseau est testé en lui donnant seulement les valeurs d'entrée mais pas les sorties désirées, et en regardant si le résultat obtenu est proche du résultat désiré.

7.2.2. Apprentissage non supervisé

Pour des réseaux à apprentissage non supervisé, on ne fournit pas au réseau les sorties que l'on désire obtenir. On le laisse évoluer librement jusqu'à ce qu'il se stabilise (s'auto - organise), on utilisant les lois régissant l'évolution des synapses.

7.2.3. Apprentissage par renforcement

L'apprentissage est de type par renforcement lorsque le réseau de neurone interagit avec l'environnement. L'environnement donne récompense pour une réponse satisfaisante du réseau et assigne une pénalité dans le cas contraire. Le réseau doit ainsi, découvrir les réponses qui lui donnent un maximum de récompenses. **[1]**

8. Apprentissage dans quelques réseaux célèbres :

8.1. Perceptron

Un perceptron linéaire à seuil prend en entrée n valeurs $x_1,..., x_n$ et calcule une sortie o. Un perceptron est défini par la donnée de n+1 constantes : les coefficients synaptiques $w_1, ..., w_n$ et le seuil θ. La sortie o est calculée par la formule :

$$0= \begin{cases} 1 \text{ si } w_i x_i > \theta \\ 0 \text{ sinon} \end{cases} \qquad (1\text{-}7)$$

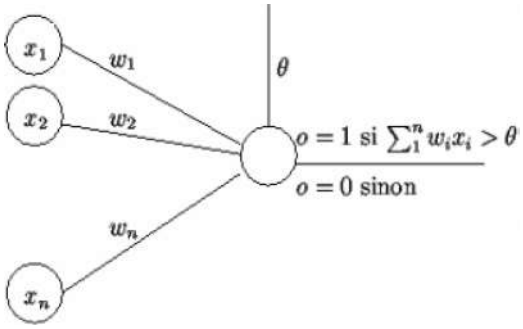


Fig1.8: Le perceptron avec seuil.

Les entrées $x_1,..., x_n$ peuvent être à valeurs dans {0,1} ou réelles, les poids peuvent être entiers ou réels. Une variante très utilisée de ce modèle est de considérer une fonction de sortie

prenant ses valeurs dans {-1,1} plutôt que dans {0,1}. Il existe également des modèles pour lesquels le calcul de la sortie est probabiliste. Dans la suite de cette partie sur le perceptron, nous considérerons toujours le modèle déterministe avec une sortie calculée dans {0,1}.

Pour simplifier les notations et certaines preuves, nous allons remplacer le seuil par une entrée supplémentaire x_0 qui prend toujours comme valeur d'entrée la valeur $x_0=1$. À cette entrée est associé un coefficient synaptique w_0. Le modèle correspondant est décrit dans la figure suivante. On peut décomposer le calcul de la sortie « *o* » en un premier calcul de la quantité $\sum w_i x_i$ appelée *potentiel post-synaptique* ou *l'entrée totale* suivi d'une application d'une *fonction d'activation* sur cette entrée totale. La fonction d'activation est la fonction de Heaviside définie par :

$$o = \begin{cases} 1 \text{ si } x \geq 0 \\ 0 \text{ sinon} \end{cases} \qquad (1\text{-}8)$$

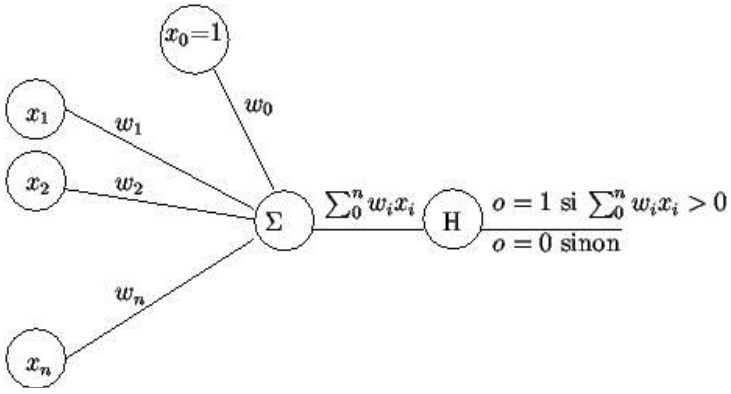


Fig1.9: Le perceptron avec entrée supplémentaire.

8.2. Réseaux multicouches : rétro-propagation du gradient

Les réseaux de neurones multicouches sont parmi les réseaux de neurones les plus utilisés pour des problèmes d'approximation, de classification et de prédiction. Il est habituellement constitué de deux ou trois couches de neurones totalement connectés. S'il est théoriquement possible de construire des réseaux avec un très grand nombre de couches cachées, les réseaux comprenant plus de couches cachées sont très rares, étant donné que chaque nouvelle couche augmente la quantité de calculs d'une manière exponentielle. La plupart des réseaux de neurones multicouches sont, dans la pratique, des perceptrons multicouches (PMC).

Le perceptron multicouche est dans la famille générale des réseaux à «propagation vers l'avant», c'est-à-dire qu'en mode normal d'utilisation, l'information se propage dans un sens unique, des entrées vers les sorties sans aucune rétroaction. Son apprentissage est de type

supervisé, par correction des erreurs. La méthode de modification des poids est très simple avec l'algorithme de Rosenblatt, mais il implique quelques limitations d'apprentissage. Dans le cas de perceptrons multicouches, comme on ne sait pas les sorties désirés des couches cachées, mais seulement de la dernière couche, il faut propager la responsabilité des erreurs de la dernière couche à la première, dans le sens contraire de l'exécution de réseau, d'où le nom rétro-propagation. De plus, les perceptrons multicouches utilisent des neurones munis de la fonction d'activation sigmoïde, celle-ci permettant les nuances nécessaires à une bonne utilisation de la rétro-propagation

Un réseau multicouche n'est rien d'autre qu'un assemblage de couches concaténées les unes aux autres, de la gauche vers la droite, en prenant les sorties d'une couche et en les injectant comme les entrées de la couche suivante. À la section suivante, nous allons voir l'algorithme dit de «rétro propagation des erreurs» qui permet d'entraîner un réseau multicouche. **[6]**

a. Règle de correction d'erreurs

Cette règle s'inscrit dans le paradigme d'apprentissage supervisé, c'est -à-dire dans le cas où l'on fournit au réseau une entrée et la sortie correspondante. Si on considère 'S' comme étant la sortie calculée par le réseau, et 'd' la sortie désirée, le principe de cette règle est d'utiliser l'erreur (d-S), afin de modifier les connexions et de diminuer ainsi l'erreur globale du système. Le réseau va donc s'adapter jusqu'à ce que 'S' soit égal à d. **[6]**

b. Algorithme d'apprentissage

Avant de pouvoir être utilisé, le réseau doit pouvoir déterminer la valeur des poids de chacune de ses connections. Cette valeur est déterminée durant la phase d'apprentissage on utilisant la règle de rétro propagation du gradient. La rétro propagation standard est un algorithme de descente du gradient, comme la loi d'apprentissage de Widrow-Hoff, dans lequel les poids du réseau sont ajustés dans le sens du gradient négatif de la fonction coût. Le terme de rétro propagation veut dire que le gradient est calculé pour des réseaux multicouches non linéaires

L'apprentissage d'un perceptron multicouche avec l'algorithme de rétro-propagation du gradient se fait d'une manière similaire à l'apprentissage d'un perceptron à neurone unique, mais il faut procéder neurone par neurone et poids par poids, en commençant par la dernière couche. Il faut calculer le gradient d'erreur pour chaque neurone, puis calculer la variation de poids pour les liens rattachés à ce neurone. Ensuite on passe à la couche précédente. Cette

méthode devient très rapidement lourde en calculs. Le processus de calcul du gradient de l'erreur et d'adaptation des poids est répété jusqu'au moment où un minimum de l'erreur, ou un point qui en est suffisamment proche, est trouvé. La fin de la phase d'apprentissage peut être décidée lorsque l'erreur moyenne observée à la sortie du réseau devient inférieure à un seuil prédéfini, ou encore lorsque l'amplitude moyenne du gradient de cette erreur devient très faible, puisque, par définition, le gradient est nul au minimum de la fonction de coût

Cet algorithme est basé sur une extension de la règle delta. On va donc utiliser un gradient stochastique pour minimiser l'erreur quadratique sur la sortie. **[1][6]**

On a :

$\mathbf{P^k_{1,i}}=\sum W_{1,ji}*S^k_{1-1,j}- \theta_{1,\iota}$ (1-9)

$\mathbf{S^k_{1,i}}=F(P^k_{1,i})$ (1-10)

$\mathbf{P^k_{1,i}}$: Potentiel du neurone 'i' de la couche 'l' pour l'exemple k.

$\mathbf{W_{1,ji}}$: Poids de la connexion du neurone 'i' vers le neurone 'j' de la couche 'l'.

$\boldsymbol{\theta_{1,\iota}}$: Le seuil du neurone 'i' de la couche 'l'.

$\mathbf{S^k_{1,i}}$: Désigne le signal de sortie du $i^{éme}$ neurone de la couche 'l' lorsque

Le vecteur X^kest présenté à l'entrée du réseau.

F: Fonction de transition non linéaire.

L'erreur sur l'exemple 'k', où 'l' signifier la couche de sortie, est définie par :

$$E^k=\frac{1}{2}\sum^c_{i=1}(S^k_{1,i}-d^k_i)^2 \quad (1\text{-}11)$$

Avec d^k_i le signal désiré pour l'exemple 'k' sur le neurone de sortie 'i', et 'c' le nombre de neurones sur la couche de sortie. L'erreur totale est :

$$E=\sum E^k \quad (1\text{-}12)$$

L'application de la règle delta donne :

$$\Delta^k w_{1,ij}=-\eta*(\frac{\partial E}{\partial W_{1,ij}}) \quad (1\text{-}13)$$

On peut poser :

$$\frac{\partial E^k}{\partial w_{1,ij}}=(\frac{\partial E^K}{\partial P^K_{1,j}})*(\frac{\partial P^K_{1,j}}{\partial w_{1,ij}}) \quad (1\text{-}14)$$

De (1-9) on déduit :

$$\frac{\partial P_{1,ij}^{k}}{\partial w_{1,ij}} = S^{k}_{1-1,i} \qquad (1\text{-}15)$$

On définit $\delta^{\kappa}_{1,j}$ le signal d'erreur :

$$\delta^{\kappa}_{1,j} = -\frac{\partial E^{k}}{\partial P_{1,j}^{k}} \qquad (1\text{-}16)$$

De (1-13), (1-14), (1-15), (1-16) on déduit la fonction d'apprentissage :

$$\Delta^{k} w_{1,ij} = \eta * \delta^{\kappa}_{1,j} * S^{k}_{1-1,i} \qquad (1\text{-}17)$$

On corrige donc les coefficients synaptiques en fonction opposée au signal d'erreur. La correction apportée au coefficient synaptique est également fonction du signal reçu. On considère ainsi qu'une erreur portant sur un signal fort nécessite une correction plus importante qu'une erreur portant sur un signal faible.

Le problème se pose maintenant de la détermination du signal d'erreur. On peut écrire :

$$\delta^{\kappa}_{1,j} = -\frac{\partial E^{K}}{\partial P_{1,j}^{k}} = \left(\frac{\partial E^{K}}{\partial S_{1,j}^{k}}\right) * \left(\frac{\partial S_{1,j}^{k}}{\partial P_{1,j}^{k}}\right) \qquad (1\text{-}18)$$

De (1-10) on déduit :

$$\frac{\partial S_{1,j}^{k}}{\partial P_{1,j}^{k}} = \varphi(P^{k}_{1,j}) \qquad (1\text{-}19)$$

A partir de là, on peut traiter la couche 'l'de sortie. En effet, de (1-11) on déduit :

$$\frac{\partial E^{k}}{\partial S_{1,j}^{k}} = -(d^{k}_{j} - S^{k}_{1,j}) \qquad (1\text{-}20)$$

Soit d'après (1-18), (1-19), (1-20) :

$$\delta^{\kappa}_{1,j} = (d^{k}_{j} - S^{k}_{1,j}) * \varphi(P^{k}_{1,j}) \qquad (1\text{-}21)$$

On a donc d'après (1-14), (1-15), (1-16) et (1-21) :

$$\frac{\partial E^k}{\partial W_{1,ij}} = -\delta^{\kappa}_{1,j} * S^{k}_{1-1,j} = -(d^{k}_{j} - S^{k}_{1,j}) * \varphi(P^{k}_{1,j}) * S^{k}_{1-1,i} \qquad (1\text{-}22)$$

Ce qui, d'après (1-13) nous donne la règle d'apprentissage suivante pour les neurones de sortie:

$$\Delta^{k} w_{1,ij} = \eta * (d^{k}_{j} - S^{k}_{1,j}) * \varphi(P^{k}_{1,j}) * S^{k}_{1-1,i} \qquad (1\text{-}23)$$

Pour chaque neurone de sortie, on corrige l'erreur induite par les neurones de la couche cachée qui lui sont liés. Cette correction est proportionnelle à l'ampleur de l'erreur et à la force du signal reçu.

Prenons le cas où la fonction de transfert est la fonction sigmoïde. On a :

$$\varphi(P^{k}_{1,j}) = \frac{1}{1 + \exp(-P^{k}_{1,j})} \qquad (1\text{-}24)$$

Et

$$\varphi'(P^{k}_{1,j}) = \varphi(P^{k}_{1,j}) * (1 - \varphi(P^{k}_{1,j}))$$

$$\varphi'(P^{k}_{1,j}) = S^{k}_{1,j}(1 - S^{k}_{1,j}) \qquad (1\text{-}25)$$

Soit en substituant dans (1-23) :

$$\Delta^{k} w_{1,ij} = \eta * (d^{k}_{j} - S^{k}_{1,j}) * S^{k}_{1,j}(1 - S^{k}_{1,j}) * S^{k}_{1-1,i} \qquad (1\text{-}26)$$

Pour les neurones cachés, on ne peut pas calculer directement les corrections puisque l'on ne connaît pas leurs contributions respectives directes à l'erreur sur la sortie liée. Toutefois, il est possible d'exprimer l'erreur sur la sortie comme fonction des différents potentiels des neurones cachés. Soit :

$$E^{K} = E^{K}(P^{k}_{1,1}, P^{k}_{1,2}, P^{k}_{1,3}, \ldots\ldots, P^{k}_{1,h1}) \qquad (1\text{-}27)$$

Avec 'hl' et 'hl' sont le nombre de neurones de la couche cachée 'l' et la couche cachée 'l' respectivement.

On a :

$$\frac{\partial E^k}{\partial S^k_{1,i}} = \sum_{j=1}^{hl+1} \left(\frac{\partial E^k}{\partial P^k_{1,1+j}}\right) * \left(\frac{\partial P^k_{1,1+j}}{\partial S^k_{1,i}}\right)$$

$$\frac{\partial E^k}{\partial S^k_{1,i}} = \sum_{j=1}^{hl+1} \left(\frac{\partial E^k}{\partial P^k_{1,1+j}}\right) * \left(\frac{\partial}{\partial S^k_{1,i}}\right) \sum_{a=1}^{hl} w_{1+1,aj} * S^k_{1,a} \qquad (1\text{-}28)$$

$$\frac{\partial E^k}{\partial S^k_{1,i}} = \sum_{j=1}^{hl+1} \left(\frac{\partial E^k}{\partial P^k_{1,1+j}}\right) * w_{1+1,ij}$$

$$= -\sum_{j=1}^{hl+1} (\delta^{\kappa}_{1+1,j} * w_{1+1,ij})$$

Soit d'après (1-18) et (1-19) :

$$\delta^{\kappa}_{1,i} = \varphi'(P^k_{1,i}) * \sum_{j=1}^{hl+1} (\delta^{\kappa}_{1+1,j} * w_{1+1,ij}) \qquad (1\text{-}29)$$

De (1-17) on déduit :

$$\Delta^k w_{1,ti} = \eta * \varphi'(P^k_{1,i}) * \sum_{j=1}^{hl+1} (\delta^{\kappa}_{1+1,j} * w_{1+1,ij} * S^k_{1-1,t}) \qquad (1\text{-}30)$$

Avec la fonction sigmoïde, la règle d'apprentissage devient d'après (1-25) et (1-30) :

$$\Delta^k w_{1,ti} = \eta (S^k_{1,i}(1 - S^k_{1,i})) * \sum_{j=1}^{hl+1} (\delta^{\kappa}_{1+1,j} * w_{1+1,ij} * S^k_{1-1,t}) \qquad (1\text{-}31)$$

Un problème qui apparaît lors d'un apprentissage est le problème de sur apprentissage. Si le réseau de neurone apprend par cœur, il donnera de mauvais résultats quand on lui présentera des données un peu différentes.**[7] [8] [9]**

8.3. ADALINE :

ADAptative LINear Element (Widrow et Hoff, 1960) pour sa part, adopte une fonction d'activation continue :

$$F(x) = x \qquad (1\text{-}32)$$

L'Adaline dispose d'un unique neurone de sortie, il est donc utilisé pour la classification de données en deux catégories correspondant aux états excité et non excité du neurone du sortie. Notons que dans les années soixante, les moyens informatiques existants étaient très limités

Les réseaux monocouche du type Adaline sont limités aux seuls problèmes linéairement séparables. Cette limitation a été principalement mise en avant dans une publication (Minsky et Papert ,1969) qui a considérablement freiné les recherches sur les réseaux de neurone, jusqu'à ce que la limitation soit levée avec la mise au point des réseaux a couches.**[10]**

8.4. Réseaux de Hopfiled

Présenté en 1982, ce modèle est basé sur le principe des mémoires associatives dont la fonction principale est identique et similaire à celle d'un filtre conçu pour restituer une information en tenant compte de sa perturbation ou du bruit. Les réseaux de Hopfield sont à rétroaction et à connexions symétriques. Les sorties dépendent des entrées et du dernier état pris par le réseau. Les neurones d'un tel réseau sont discrets et répondent à une fonction seuil telle que:

$$F(x)=\begin{cases} +1 \text{ si } x>0 \\ -1 \text{ si } x\leq 0 \end{cases} \qquad (1\text{-}33)$$

Le réseau de Hopfield étant l'un des réseaux les plus simples, son principe démontre un apprentissage facile à mettre en œuvre. Les problèmes les plus importants apparaissent au cours de la phase d'évaluation. La règle d'apprentissage repose sur celle de Hebb qui consiste à forcer les poids de connexions entre les neurones actifs au même moment. Par contre, ces poids seront diminués si les neurones sont inactifs. Pour Hopfield, deux neurones dans l'état -1 sont actifs. La modification des poids se fait donc par un calcul analytique très simple. Lors du processus de généralisation, on présente alors un vecteur d'entrée au réseau qui calcule la sortie correspondante et la réinjecte à l'entrée. Le processus sera répété jusqu'à ce que la sortie (entrée de l'étape t+ 1) soit identique à l'entrée à l'étape t. C'est ce qui traduit la récurrence des réseaux de Hopfield.

8.5. Réseaux de Kohonen

Ces réseaux sont de types auto-organisateurs car ils fonctionnent en grande majorité dans un mode non supervisé. Aussi, ce sont des réseaux dynamiques dans la mesure où ils ont la tendance à créer ou à éliminer des neurones. Une autre particularité de ce réseau est la compétition entre les neurones d'une même couche.

Rappelons que pour qu'un réseau soit compétitif, il faut que les neurones de la couche compétitive réagissent différemment aux entrées afin d'élire un vainqueur. On peut aussi

établir un mécanisme de concurrence entre les neurones afin de choisir le plus activé et modifier ses poids de connexion en conséquence. L'apprentissage d'un tel réseau vise donc à favoriser le neurone vainqueur d'une couche en le rapprochant du vecteur d'entrée correspondant. Mais l'inconvénient est que pour une entrée de validation éloignée, ce neurone est pénalisé. **[10]**

8.6. Réseaux RBF (Radial Basis Fonction)

Le réseau RBF est un réseau de neurones supervisé. Il s'agit d'une spécialisation d'un PMC. Un RBF est constitué uniquement de 3 couches :

• La couche d'entrée : elle retransmet les entrées sans distorsion.

• La couche RBF : couche cachée qui contient les neurones RBF.

• La couche de sortie : simple couche qui contient une fonction linéaire.

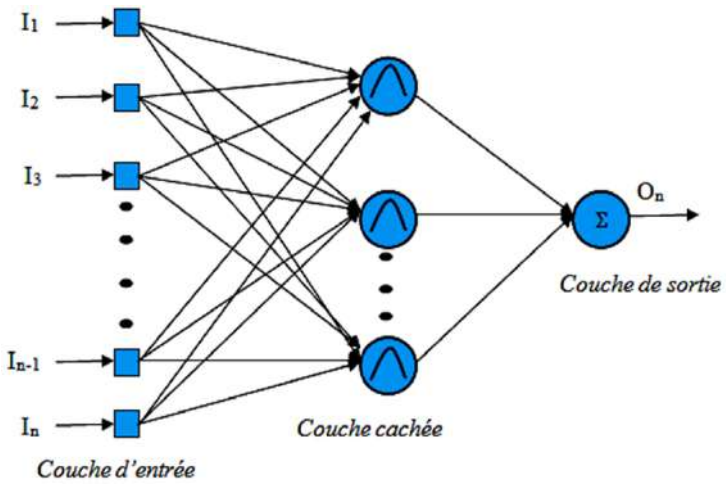


Fig1.10:Réseaux RBF

Chaque neurone RBF contient une gaussienne qui est centrée sur un point de l'espace d'entrée. Pour une entrée donnée, la sortie du neurone RBF est la hauteur de la gaussienne en ce point. La fonction gaussienne permet aux neurones de ne répondre qu'à une petite région de l'espace d'entrée, région sur laquelle la gaussienne est centrée. Donc il y a quatre paramètres principaux à régler dans un réseau RBF.

• Le nombre de neurones RBF (nombre de neurones dans l'unique couche cachée).

• La position des centres des gaussiennes de chacun des neurones.

• La largeur de ces gaussiennes.

• Le poids des connexions entre les neurones RBF et le(s) neurone(s) de sortie. Toute modification d'un de ces paramètres entraîne directement un changement du comportement du réseau.**[10]**

9. Les étapes de la conception d'un réseau

Les débutants sont souvent surpris d'apprendre que pour construire un réseaux de neurones, la première chose à faire n'est pas de choisir le type du réseau mais de bien choisir ses échantillons de données d'apprentissage, de test et de validation. C'est n'est qu'ensuite que le choix du type du réseau interviendra. Afin de clarifier les idées, nous présentons dans ce qui suit les quatre étapes (chronologiquement parlant) qui doivent guider la création d'un réseau de neurone. **[1]**

9.1. Choix des échantillons

Le processus d'élaboration d'un réseau de neurones commence toujours par le choix et la préparation des échantillons de données. Comme dans les cas d'analyse de données, cette étape est cruciale et va aider le concepteur à déterminer le type de réseau le plus approprié pour résoudre son problème. La façon dont se présente l'échantillon conditionne : le type de réseau, le nombre de cellules d'entrée, le nombre de cellules de sortie et la façon dont il faudra mener l'apprentissage, les tests et la validation.

9.2. Elaboration de la structure du réseau :

La structure du réseau dépend étroitement du type des échantillons. Il faut d'abord choisir le type de réseau : un perceptron standard, un réseau de Hopfield, un réseau à décalage temporel (TDNN), un réseau de Kohonen, un ARTMAP etc... Dans le cas du perceptron par exemple, il faudra aussi choisir le nombre de neurones dans la couche cachée. Plusieurs méthodes existent et on peut par exemple prendre une moyenne du nombre de neurones d'entrée et de sortie, mais rien de vaut de tester toutes les possibilités et de choisir celle qui offre les meilleurs résultats. **[1]**

9.3. Apprentissage :

L'apprentissage consiste tout d'abord à calculer les pondérations optimales des différentes liaisons, en utilisant un échantillon. La méthode la plus utilisée est la rétro propagation : on entre des valeurs des cellules d'entrée et en fonction de l'erreur obtenue en sortie (le delta), on

corrige les poids accordés aux pondérations. C'est un cycle qui est répété jusqu'à ce que la courbe d'erreurs du réseau ne soit croissante (il faut bien prendre garde ne pas surentrainer un réseau de neurones qui deviendra alors moins performant).

9.4. Validation et tests

Alors que les tests concernent la vérification des performances d'un réseau de neurones hors échantillon et sa capacité de généralisation, la validation est parfois utilisée lors de l'apprentissage. Une fois le réseau calculé, il faut toujours procéder à des tests afin de vérifier que notre réseau réagit correctement. Il y a plusieurs méthodes pour effectuer une validation : la cross validation, mais pour les tests, dans le cas général, une partie de l'échantillon est simplement écarté de l'échantillon d'apprentissage et conservé pour les tests hors échantillon. On peut par exemple utiliser 60% de l'échantillon pour l'apprentissage, 20% pour la validation et 20% pour les tests. Dans les cas de petits échantillons, on ne peut pas toujours utiliser une telle distinction, simplement parce qu'il n'est pas toujours possible d'avoir suffisamment de données dans chacun des groupes ainsi créent. On a alors parfois recours à des procédures comme la cross-validation pour établir la structure optimale du réseau. **[1]**

10. Les avantages et les inconvénients

Pour les avantages des réseaux de neurones, nous pouvons citer : [**11**]

- Apprentissage automatique des poids (les règles d'apprentissage) ;

- Capacité de généralisation (la phase des tests) ;

- Possibilité de faire le parallélisme (les éléments de chaque couche peuvent fonctionner au parallèle) ;

- Résistance aux pannes (si un neurone ne fonctionne plus, le réseau ne perturbe pas) ;

- Modélisation des fonctions inconnues ;

- Bonne approximation.

Par contre, les réseaux de neurones présentent quelques inconvénients :

- Représentation complexe (généralement l'architecture de réseau de neurones est complexe, et même si on la schématise, elle reste peu lisible) ;

-Paramètre difficiles à interpréter (boite noire) ;

- Inexistence des règles définissant l'architecture du réseau en fonction du problème à résoudre

- Les algorithmes d'apprentissage consomment un temps de calcul considérable.

11. Conclusion

Dans ce chapitre nous avons présenté une étude générale sur les réseaux de neurones et en particulier les réseaux multicouches à rétro propagation de gradient.

Les réseaux de neurones sont une puissante source d'inspiration, L'intérêt porté aujourd'hui aux réseaux de neurones tient sa justification dans les propriétés fascinantes qu'ils possèdent ces outils ont permis aussi le dépassement des limites de l'informatique traditionnelle, en offrant des services variés tels que la classification, la modélisation, l'approximation…, tout ces facteurs ont permis à l'informatique de pénétrer plusieurs domaines: le contrôle et la commande automatique, le traitement d'image et du signal numérique en général.

De plus les réseaux de neurones peuvent être implémentés sur des architectures systoliques.

Chapitre II

Réseau systolique et architecture digitale des RNA

1. Introduction

Pour augmenter la vitesse de calcul, il est connu, d'une part, de diminuer le temps d'exécution des opérations élémentaires, et d'autre part, d'exécuter un grand nombre d'opérations simultanément. L'architecture systolique décrite dans "Why Systolic Architectures" IEEE Computer de janvier 1982, se prête particulièrement bien à la réalisation de circuits exécutant un grand nombre d'opérations simultanées. Cette architecture utilise le traitement en parallèle et le traitement "pipeline".

Le modèle systolique a été introduit en 1978 par Kung et Leiserson pour répondre à la demande sanscesse croissante d'une plus grande puissance de calculà moindre frais. Il s'est révélé être un outil très efficacepour la conception de processeurs intégrés spécialisés.

2. Définition

Les architectures systoliques sont des structures composées de processeurs élémentaires (PE) interconnectés localement. Le travail de ces cellules est synchronisé par une horloge .L'ensemble forme un réseau systolique synchrone réalisant une fonction bien déterminé e.Seules les cellules situées à la frontière du réseau communiquent avec le monde extérieur, (une mémoire externe ou un ordinateur hôte). A chaque cycle d'horloge chaque PE réalise la fonction qui lui a été assigné et propage .Le fonctionnement du réseau systolique ressemble au phénomène de la circulation sanguine dans le corps Humain qui est pulsé par le rythme cardiaque. C'est d'ici que vient le terme systolique. **[12]**

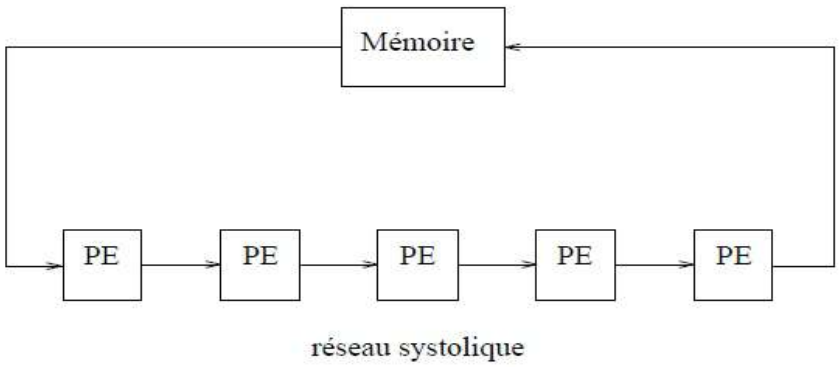


Fig2.1: Principe d'une architecture systolique.

Deux classes de bases on été proposées, selon la multiplicité de flux d'instructions et de données disponibles

-L'architecture SIMD : une seule instruction et plusieurs données.

-L'architecture MIMD : plusieurs instructions et plusieurs données.

3. Champs d'application des réseaux systoliques

-Traitement du signal et de l'image : Filtrage; convolution 1D et 2D; corrélation; lissage par médiane 1D et 2D; transformée de Fourier discrète; projections géométriques; encodage/décodage pour les corrections d'erreurs

-Arithmétique matricielle : Multiplication matrice-vecteur, matrice-matrice; triangularisation (résolution de systèmes linéaires, calcul de l'inverse d'une matrice); décomposition QR (calculaux moindres carrés, inversion de matrice de covariance); problèmes aux valeurs propres.

-Applications non numériques :structures de données : piles; files d'attente; recherche dans un dictionnaire; tri; … graphes et algorithmes géométriques : fermeture transitive; arbre de degré minimum; composantes connexes; enveloppes convexes; … manipulation de chaînes de caractères : occurrences d'un mot; plus longue sous-suite; reconnaissance de langages réguliers; programmation dynamique; opérations sur des bases de données relationnelles; algèbre des polynômes : multiplication; division euclidienne; PGCD; arithmétique entière et dans des corps finis : multiplication; division; PGCD; simulation de type Monte-Carlo**.[13]**

4. Description des unités opératrices

Dans l'architecture systolique les cellules mis en cascade sont optimisés pour divers types de calculs comme par exemple le calcul de convolution, le calcul de corrélation ou le calcul de filtrage autorégressif. L'exécution de ces calculs exigeait donc la réalisation de quatre circuits. Or, ces circuits sont souvent réalisés sous la forme. De circuits intégrés, LSI ou VLSI. Pour de tels circuits, très spécialisés et voués à une faible diffusion, une part importante du prix provient de la conception et de la mise au point du circuit.

L'architecture de réseau systolique est composée de trois types de cellules (fig2.2). Chacune réalise une fonction spécifique. **[14]**

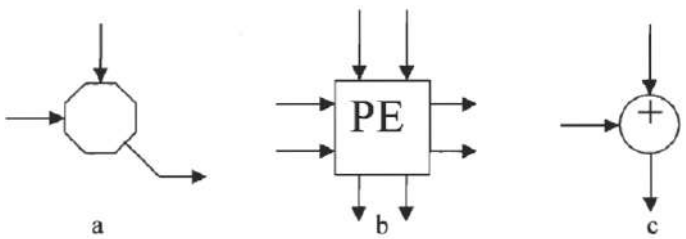


Fig2.2: Différentes Cellules constituants le réseau systolique.

Les cellules circulaires (c) possèdent deux entrées et une sortie ils permettent de réaliser la fonction d'addition simple.

La cellule en octogonale (a) permet de réaliser la fonction de division du numérateur num_kpar le dénominateur C_k

Les cellules (b) représentent les processeurs élémentaires du réseau systoliques. On propose deux architectures internes (PE2 et PE1) pour ce type de cellules. La structure PE 1 est optimisée pour la vitesse alors que PE 2 est optimisée pour la surface.

L'architecture interne des processeurs élémentaires est représentée sur la (fig2.3 et 2.4) PE 1 est composée de deux multiplieurs et un additionneur. PE2 à une structure plutôt complexe, cette dernière permet de réaliser la même tache que PE 1 mais en trois fois le nombre de cycle. Il est composé d'un multiplieur à base d'un réseau de CSA[3]et un FA[4] (fig2.3).

Dans cette architecture on exploite le FA pour réaliser le multiplieur en même temps il sert pour faire l'addition simple.

Au premier cycle d'horloge PE2 réalise la multiplication de. $\mu(\tilde{Y}_k, l_1, t-1)$par. $\mu(\tilde{Y}_k, l_2, t-1)$et met le résultat dans un registre. Au deuxième cycle d'horloge PE2 réalise la multiplication de $\theta^{(l_1,l_2)}$ par le résultat de la première multiplication. Enfin au troisième cycle on réalise l'addition du résultat obtenu par$N^{(l_1-1,l_2)}(t-1)$et nous permet ainsi d'avoir le résultat $N^{(l_1,l_2)}(t)$.en plus PE2 transmet $\mu(\tilde{Y}_k, l_1, t-1)$ et $\mu(\tilde{Y}_k, l_2, t-1)$ aux processeurs voisins.

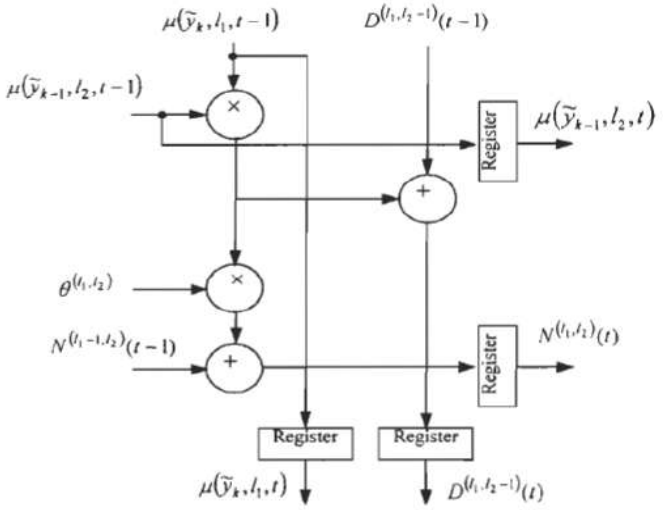


Fig2.3: Architecture interne de PE1.

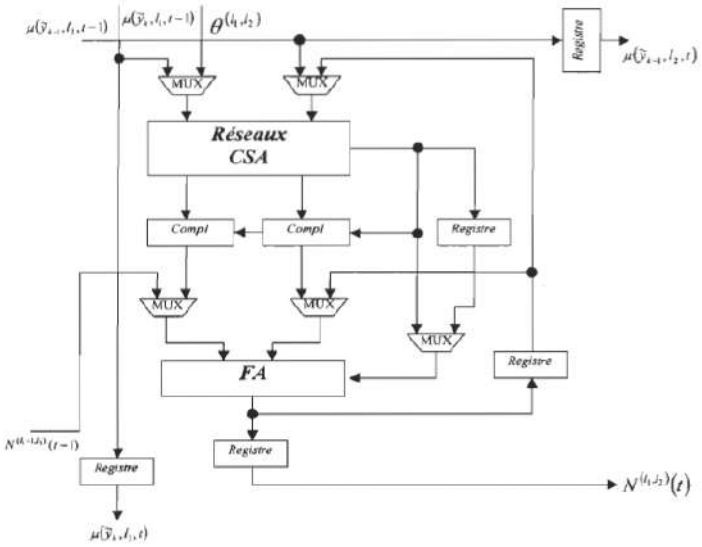


Fig2.4: Architecture de PE2.

Le contenu de tous les registres est transmis aux processeurs voisins au prochain cycle d'horloge. À la fin de chaque colonne ou ligne de processeurs il y a un additionneur qui a pour rôle de faire la somme du dénominateur ou numérateur. Les cellules d'addition qui se situent juste avant le diviseur contiennent la valeur du numérateur et du dénominateur. La dernière étape est de faire la division entre ces deux termes. Ce qui va donner la valeur estimée de l'échantillon d'entré. **[14]**

5. Fonctionnement général

Le fonctionnement d'un réseau systolique peut être vu comme celui d'un ensemble de données qui se déplacent dans le réseau, en étant modifiées par les cellules ou alors simplement transmises. Une même cellule peut présenter des comportements différents selon les temps d'horloge, grâce à des mécanismes de contrôle.

Pour une cellule donnée, chaque cycle d'horloge ne correspond pas obligatoirement a une donnée valide : il peut y avoir un ou plusieurs cycles entre le passage de deux données consécutives. Même si physiquement parlant il y a bien calcul ou transmission de données a chaque cycle, les données non significatives car ne participant pas au calcul du résultat souhaité- ne sont pas prises en compte. Pour un cycle donné, une cellule sera dite active si elle participe au calcul ou à la transmission d'une donnée valide, et passive sinon.

6. Différentes topologies de réseaux systoliques

Il existe plusieurs topologies d'architectures systoliques. Suivant l'algorithme on choisit la topologie qui maximise l'efficacité du réseau. Le choix de la topologie à adopter pour un algorithme donné est une tâche très délicate qui nécessite beaucoup d'imagination et de travail. Les différentes topologies sont: les réseaux linéaires, orthogonales, triangulaires. [**14]**

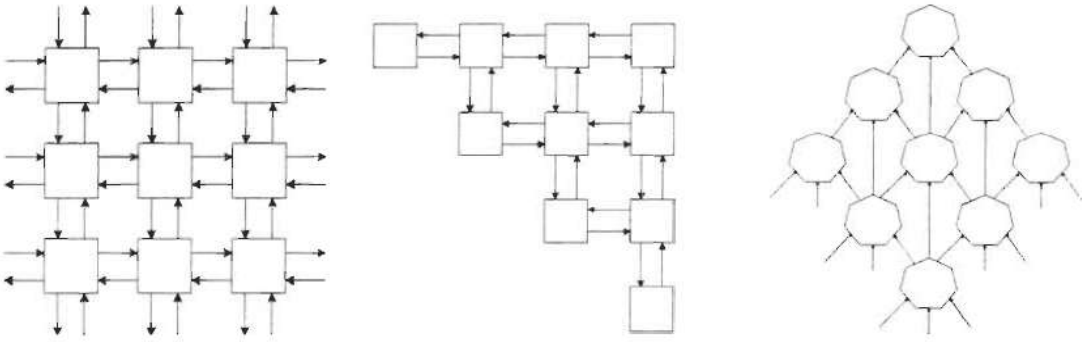


Fig2.5: Réseau systolique carré, triangulaire et hexagonal.

7. Propriétés des architectures systoliques :

Les architectures systoliques présentent plusieurs caractéristiques qui rendent très intéressant leur intégration en ITGE Les principales sont la régularité, la modularité, le parallélisme et les communications locales. **[14]**

7.1La régularité

La régularité vient du fait que les PEsont pour la plupart identiques. Donc, seule la réalisation d'un petit nombre de PEest nécessaire pour ensuite générer l'architecture enentier. Une optimisation locale sur les PE peut être réalisée plutôt qu'une optimisationglobale de l'architecture afin d'obtenir un meilleur compromis entre la surface et le tempsde calcul. Cette régularité simplifie également le placement des divers éléments sur le silicium.

7.2. La modularité

La modularité des architectures systoliques permet d'adapter facilement l'architecture pour différentes dimensions du problème. Par exemple, si la taille des matrices ou des vecteurs augmente, il s'agit seulement de grossir le réseau systolique tout en gardant la même topologie pour adapter l'architecture.

7.3. Le parallélisme

Le principe général du parallélisme est de gagner de la vitesse en effectuant des calculs simples de façon simultanée. Dans les architectures systoliques, le fait que tous les PE fonctionnent en même temps synchronisé sur l'horloge les rend hautement parallèles.
Cependant, la surface d'intégration peut réduire le nombre d'unités arithmétiques qui sont implantées et par le fait même, réduire le degré de parallélisme de l'architecture. Aussi, le fait que l'horloge commande un nombre de commutations, des délais de connections peuvent entraîner un décalage temporel de celle-ci et nuire à la synchronisation du circuit intégré.

7.4. Les communications locales

De manière réaliste, il est impossible physiquement que tous les processeurs soient connectés les uns aux autres. Aussi, il s'agit de ne pas ralentir le circuit à cause de connections trop longues. Les longues connections entraînent des capacités et des résistances parasites dans le circuits qui introduisent des délais temporels qui peuvent devenir non négligeables face au temps de calculs. C'est ici qu'intervient le principe de localité. C'est-à-dire que chaque processeur élémentaire ne communique qu'avec ses voisins immédiats amenant des chemins de longueurs minimales comme le met en évidence (fig2.6).
En suivant le même principe, le rapport entrée-sortie doit suivre la même règle. Il faut donc désigner des processeurs élémentaires hôtes, en périphérie du réseau, qui communiqueront avec l'extérieur.

.

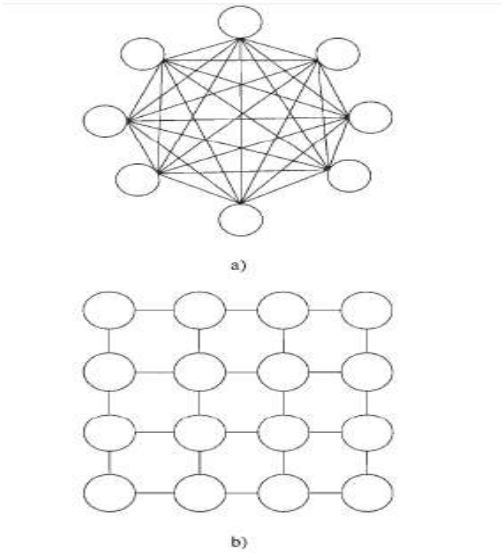


Fig2.6:Réseau de processeurs élémentaires a) sans communications locales et b) avec communications locales.

Si on prend les technologies actuelles qui fabriquent des circuits en deux dimensions, il est impossible de communiquer efficacement qu'avec ses voisins en évitant dans bien des cas de changer de couche d'intégration. Il est donc profitable d'utiliser des communications locales.

8. Architecture systolique des RNAs

Cette conception commence par un graphe de dépendance d'états pour exprimer la récurrence et le parallélisme associé 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 :

-Exprimer le traitement neuronal en termes d'opération de matrices récursives

-Exprimer les opérations des matrices par des graphes de dépendances

-Implémenter les graphes de dépendances 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 généralisé pour devenir applicable a une variété de formes de connexions. Les n majeurs nécessaires sont sur la conception fonctionnelle dans le PE pour refléter l'activation et les règles d'apprentissage.

9. Représentation des opérations matricielles par des graphes de dépendances

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 a une opération et un lien est associé a 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. **[15]**

-Un minimum de temps de calcul pour chaque opération.

-Un maximum de débit pour les instances.

Elle soufre 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 (V*B) ,tel que V={v_i,i=1…4},ainsi que l'algorithme de produit de deux matrices 4*4 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 :

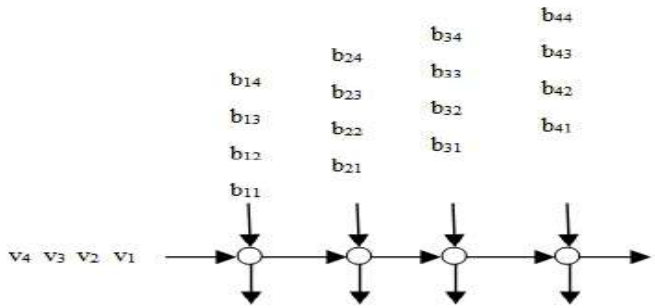


Fig2.7: Graphe de dépendances de la multiplication vecteur-matrice (V*B).

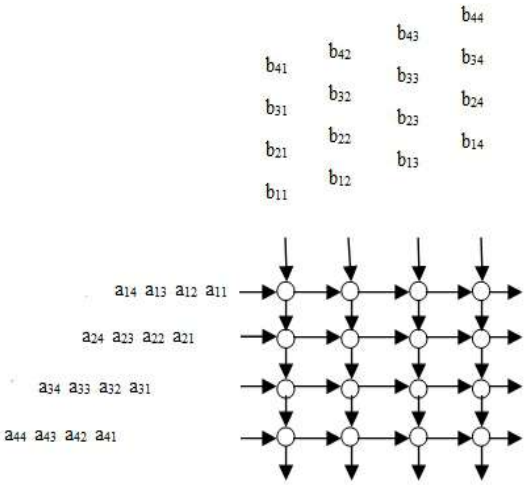


Fig2.8: Graphe de dépendance de la multiplication de deux matrices (A*B)

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 E/S et un nombre de PEs raisonnables. Cela signifié que le réseau doit avoir un nombre de cellules plus petit que le nombre des nœuds dans le graphe de dépendance .donc, il est nécessaire d'implémenter les opérations sur les cellules en des étapes de temps adéquates. Cette procédure nécessite la détermination des caractéristiques de réseau, à quel moment est 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 de revêt deux aspects : **[15]**

a. Architecture

Dans cet aspect on donne les caractéristiques des modules composant le réseau et leur interconnexion (cellules de traitements, 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.

b. 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 a 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.

10. Régularisation 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 réseau systolique. **[15]**

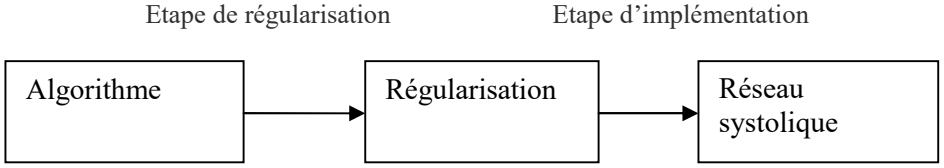


Fig2.9: Etape nécessaire pour l'implémentation d'un algorithme RPG.

Les méthodes de transformations 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'autre transformations sur un réseau systolique.

10.1. Régularisation 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 :

$$\begin{cases} U(l+1)=W\,a(l)+\Theta \\ a(l+1)=F(U(l+1)) \end{cases} \qquad (2\text{-}1)$$

Où West la matrice des poids reliant les neurones la couche à la couche l+1,$a_i(l)$ et la sortie du neurone ide 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)** suivi par la fonction d'activation sigmoïde afin d'être implémentée sur des réseaux systoliques.

Pour un réseau multicouche ayant N neurones dans la couche (l+1) et N neurones dans la couche (l), le premier terme dans l'équation (2-1) peut être explicité comme suit :

$U_1(l+1)=w_{11}a_1(l)+w_{12}a_2(l)+\ldots+w_{N-1}a_N(l)+\Theta_1$

$U_2(l+1)=w_{21}a_1(l)+w_{22}a_2(l)+\ldots+w_{2N}a_N(l)+\Theta_2$

.

.

.

$U_{N-1}(l+1)=w_{N-1\ 1}a_1(l)+w_{N-1\ 2}a_2(l)+\ldots+w_{N-1\ N}a_N(l)+\Theta_{N-1}$

$U_N(l+1)=w_{N\ 1}a_1(l)+w_{N\ 2}a_2(l)+\ldots\ldots\ldots+w_{N\ N}a_N(l)+\Theta_N$

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 :

$U_1(l+1)=w_{11}a_1(l)+w_{12}a_2(l)+\ldots+w_{1\ N-1}a_{N-1}(l)+\Theta_1$

$U_2(l+1)=w_{21}a_1(l)+w_{22}a_2(l)+\ldots+w_{2N}a_N(l)+\Theta_2$

.

.

.

$U_{N-1}(l+1)=w_{N-1\ N-1}a_{N-1}(l)+w_{N-1\ N}a_N(l)+\ldots+w_{N-1\ 1}a_1(l)+w_{N-1\ 2}a_2(l)+\Theta_{N-1}$

$U_N(l+1)=w_{N\ N}a_N(l)+w_{N\ 1}a_1(l)+\ldots+w_{N\ N-2}a_{N-2}(l)+w_{N\ N-1}a_{N-1}(l)+\Theta_N$

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 :

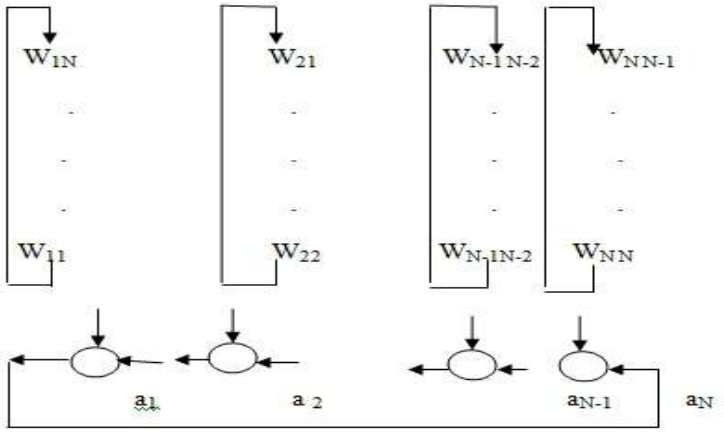


Fig2.10: Projection du DDG de calcul du potentiel.

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 $\{a_i\}$.

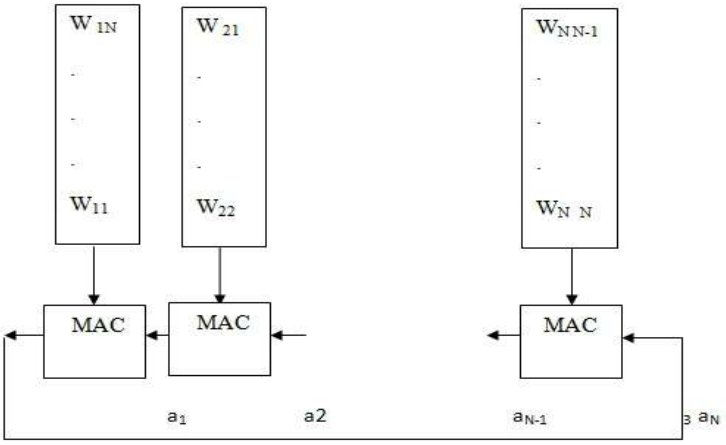


Fig2.11: Anneau systolique pour le calcul du potentiel.

Après N itérations de multiplication et accumulation on fait appel a la fonction d'activation f pour calculer $\{a_i(l+1)\}$, qui eux même constituent les entrées 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.

10.2. Régularisation 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.… **[16]**

La mise à jour des poids est exprimée par :

$$w^*_{ij}(l+1)=w_{ij}(l+1)+\Delta w_{ij}(l+1). \quad (2\text{-}2)$$

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

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 :

$$\Delta w_{ij}(l+1)=g_i(l+1).h_j(l+1) \quad (2\text{-}3)$$

Pour l'algorithme RPG, 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)*
$\delta_r \rho_i(l+1)$	$-\eta a_j(l)$	*La couche(0<l<L)*

Fig2.12: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.*

a.Calcul de l'erreur corrective δ

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

$$\delta_i(l) = \sum g_i(l+1).w_{ij}(l+1) \quad \text{telle que } l<L. \quad (2\text{-}4)$$

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 :

$$\delta_i(L)= a_i(L) - t_i. \quad (2\text{-}5)$$

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

$$w^*_{ij}(l+1)=w_{ij}(l+1)+ g_i(l+1).h_j(l+1). \quad (2\text{-}6)$$

Dans le but de dériver sa structure systolique l'équation (2-6) est formulée comme des opérations de mise a jour de produits externes *(OPU :Outer Product Updating)*.

Ainsi que l'équation (2-4) peut être développée comme suit :

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

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

… …

… …

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

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

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

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

… …

… …

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

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.

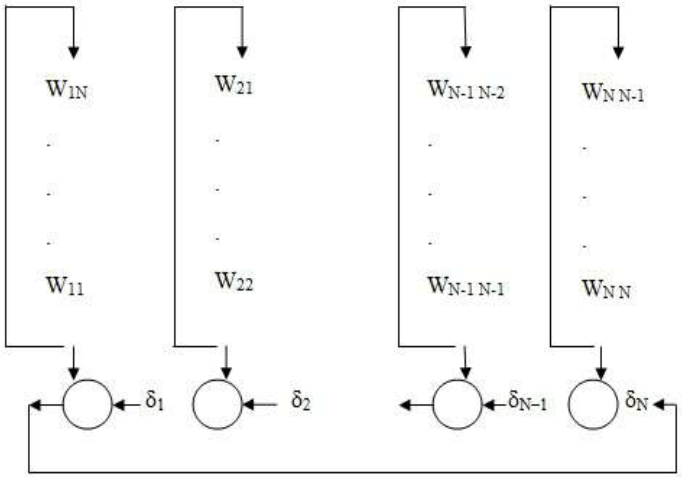


Fig 2.13:projection du DDG de calcul de $\delta_i(l)$.

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. »

b. 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 :

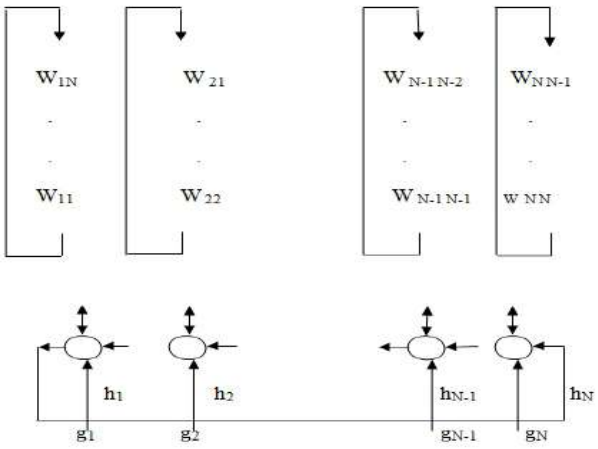


Fig2.14: Projection du DDG de la mise à jour des poids.

Comme la sortie de la couche (l) constitue l'entrée de la couche (l+1), donc les opération 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).**[16]**

11. Architecture en anneau systolique des RNAs

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 Up dating :OPU). De point de vue structure de 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). **[17]**

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.

11.1. Processeur élémentaire

C'est un processeur pouvant exécuter les opérations suivantes :

-multiplication pour calculer les $\{h_j\}$ où : $(h_j(l+1) = \eta.a_j(l))$.

-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.

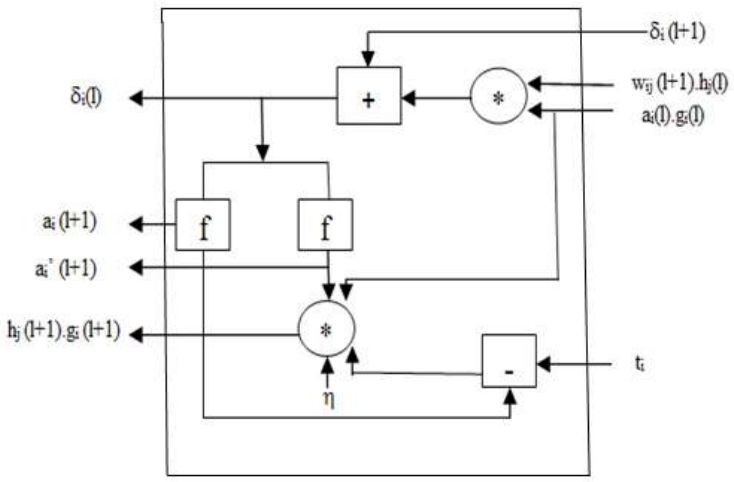


Fig2.15: PE utilisé par l'algorithme RPG.

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.

11.2. 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.

Traitement systolique des MVMs

Dans la$(l+1)^{éme}$ 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 position). Les opération de la $(l+1)^{éme}$ couche peuvent etre décrite comme suit :
-Chaque valeur d'activation d'une unité, crée à l'iémePE, 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.
- Lorsque $a_i(l)$ arrive au $i^{éme}$ PE, elle est multipliée par w_{ij} et le produit est ajouté à $U_i(l+1)$.
- Après N unités d'horloge, l'accumulateur $U_i(l+1)$ collecte tous les produits nécessaires.
- Plus qu'une unité d'horloge est nécessaire pour que $a_i(l)$ retourne au $i^{éme}$ 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 etre exécutée récursivement (avec l'accroissement de l jusqu'à ce L couches soient terminées.
La fonction non linéaire utilisée par l'algorithme RPG peut être approximée par des segments linéaires**. [18]**

11.3. Anneau systolique pour l'apprentissage

Il y'à deux types d'opérations demandées dans la phase d'apprentissage de l'algorithme RPG. Les mises à jour des produits externes consécutifs OPUs et 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 :

-La mise à jour des poids peut être calculée en se basant sur les opérations OPUs.

-Le signal correctif d'erreur de la $l^{éme}$ 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 $(l+1)^{éme}$ 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 :

-La valeur de $g_i(l+1)$ est calculée au niveau du $i^{éme}$ PE. La valeur $h_j(l+1)$ produite au $j^{éme}$ PE sera circulairement 'canalisée' à gauche pour tous les PEs dans le réseau de neurones en anneau systolique pendant N tops d'horloge.

-Quand $h_j(l+1)$ arrive au $i^{éme}$ 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 $l^{éme}$ couche 'on note que les accumulateurs $\{\Delta w_{ij}\}$ sont accédés à des séquences à décalage circulaire, juste comme l'ordonnancement de l'accès à $\{w_{ij}\}$'.

- Après N tops d'horloges, tous les N*N 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.

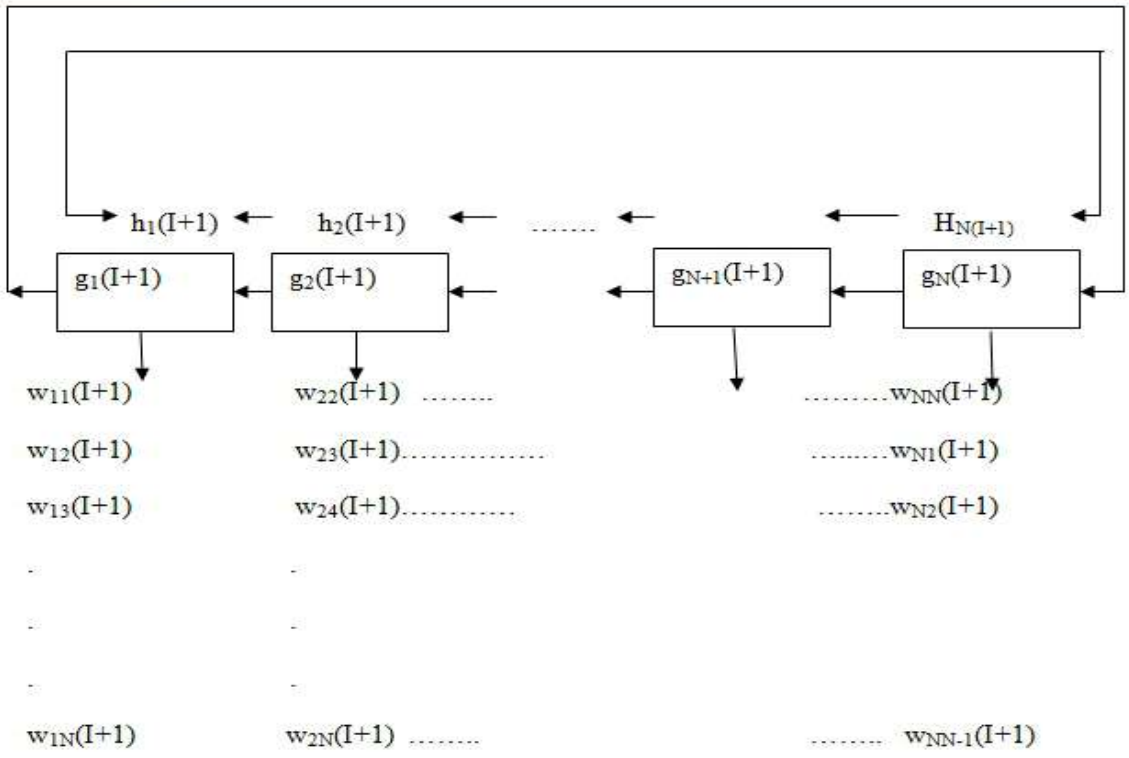


Fig2.16: Architecture en anneau systolique pour l'opération OPU.

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 :

-Le signal $g_i(l+1)$et la valeur de w_{ij} sont disponibles pour la $(l+1)^{éme}$ itération. La valeur de w_{ij} est donc multipliée par $g_i(l+1)$ au $j^{éme}$ PE.

-Le produit est ajouté au paramètre accumulateur acc_i arrivée de nouveau (le paramètre acc_i est initialisé au $i^{éme}$ PE à zéro) et se décale à gauche circulairement à travers le réseau en anneau.

-Après N opérations d'accumulation, le paramètre acc_i retourne au $j^{éme}$ PE après avoir accumulé tous les produits. **[15]**

12. Conclusion

La simulation numérique des réseaux neuromimétiquesde grande taille impose l'utilisation d'outils de calcul adéquats. Les modèles de réseaux neurones étant très divers, le calculateur utilisé doit satisfaire à un compromis, entre une structure universelle et une structure dédiée. L'architecture systolique répond favorablement à cette alternative.

Le réseau systolique est plus universelle et plus efficace dans le traitement de signal l'application de cette architecture à l'algorithme de la rétro propagation (propagation vers l avant)est notre objectif dans le chapitre suivant.

Chapitre III

Implémentation et simulation

1. introduction :

Les réseaux de neurones artificiels ont montré leur efficacité théorique; mais ils ont maintenant à prouver leur intérêt pratique par des architectures efficaces et bon marché pour des utilisations pratiques. Beaucoup de simulations ont été faites sur des machines séquentielles ou des machines parallèles mais les modèles connexionnistes sont très intensifs au niveau communication et ils bénéficieraient d'architectures plus spécialisées, qui pourraient mettre en valeur le caractère parallèle des modèles neuronaux. Dans ce chapitre, nous allons proposé en conséquence, une architecture qui permet l'implémentation d'un réseau de neurones sur une architecture systolique. Pour cela, nous avons opté pour un réseau de neurone à une architectures (1,10,1), utilisant la fonction sigmoïde (tangente sigmoïde) comme règle de transmission.

On va commencer par un rappel sur les différentes équations qui caractérisent la partie Feed-Propagation, on va donner une architecture préalable pour l'implémentation de cette partie, ensuite on va modéliser cette architecture en utilisant la notion d'objet orientée et on va rapporter les résultats. Enfin nous terminerons par une petite conclusion.

2. Rappel sur le module Feed- Forward :

Le module Feed-Forward donne l'état de chaque neurone d'un réseau multicouche. La propagation des états se fait 'en aval' dès la couche d'entrée vers la couche de sortie. L'état d'un neurone i dans la couche q est fonction des états des neurones j des couches précédentes q-1,…etc. **[19]**

Le fonctionnement de la partie Feed-Forward est donné comme suit :

1. L'entrée totale A_i d'un neurone(i) d'une couche(q) est donnée en fonction des sorties x_j des neurones de la couche précédente (q-1) et des poids de connexion entre les deux couches w_{ji}.

$$A_i^q = \sum_j^{N_{q-1}} w_{ji}^q * x_j^q = \sum_j^{N_{q-1}} w_{ji}^q * y_j^{q-1}$$

2. La sortie d'un neurone y_i est obtenue par l'application de la fonction d'activation uniformément dérivable f sur l'entrée A_i. On prend f comme exemple la sigmoïde.

$$y_i^q = x_i^{q+1} = f(A_i^q) = \frac{1}{1+\exp(-aA_i^q)}$$

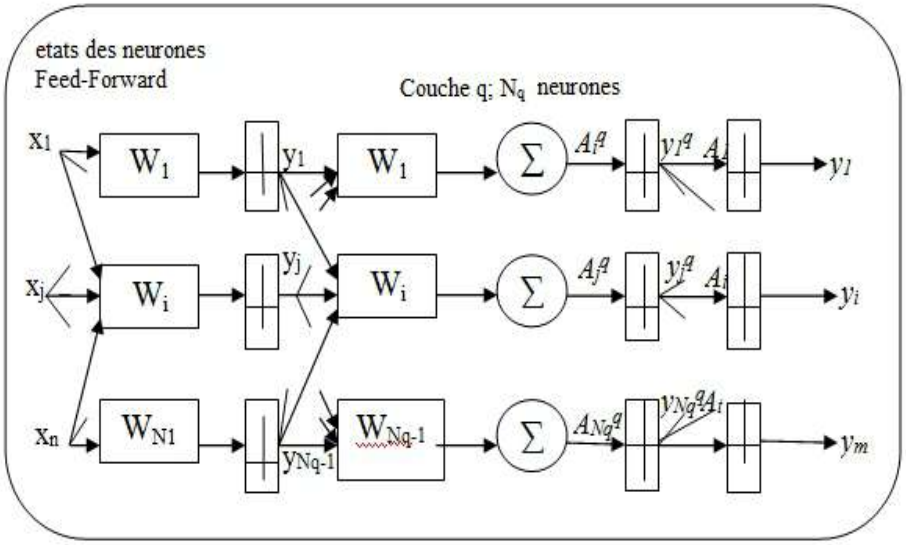


Fig 3.1: Propagation des états de la partie Feed-Forward.

3. Architecture d'implémentation du module Feed-Forward :

A partir de la figure 3.1 et des équations précédentes, nous proposons une architecture pour l'implémentation du module Feed-Forward

Ce module contient une unité de contrôle et une autre opératrice constituée de trois couches, comme le montre la figure 3.2

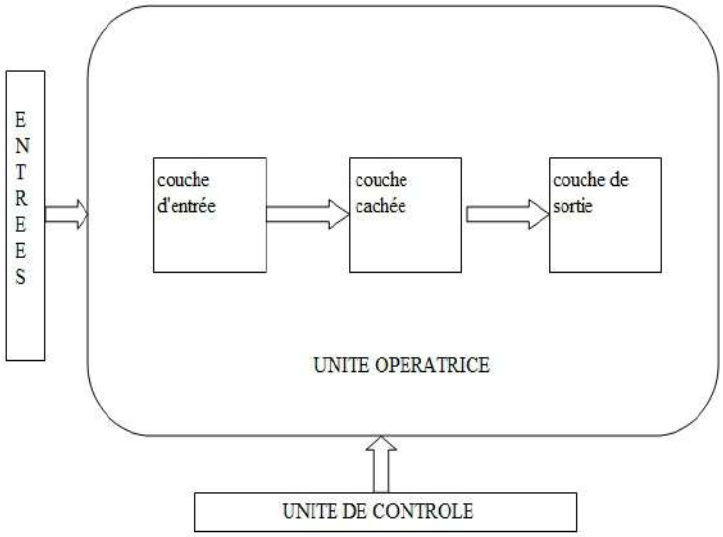


Fig 3.2: Architecture du module Feed-Forward.

3.1. Unité opératrice :

Cette unité du module Feed-Forward est constituée de trois couches de traitement similaires : couche d'entrée, couche interne et couche de sortie. Le rôle de cette unité est de propager les modèles (exemples) qui se présentent à l'entrée du réseau à travers les différentes couches jusqu'à la sortie de cette partie. Cette propagation est faite en calculant les valeurs d'activation des neurones de la couche d'entrée, et qui seront transmises par la suite à la couche interne ou ils seront considérés comme des modèles d'entrée pour cette couche. Ces dernières valeurs seront transmises en sortie à travers la dernière couche pour donner les sorties calculées de réseau.**[19]**

Ceci est la description générale de la partie Feed-Forward. Dans ce qui suit, nous allons détailler chaque élément de cette partie.

3.1.1. La couche :

Le module Feed-Forward est constitué du point de vue structural de couches similaires. Chacune de ces couches est constituée d'un nombre (N) de neurones, qui sont précédés par un bloc de multiplexage, pour que les entrées x_j rentrent en série dans chaque neurone.

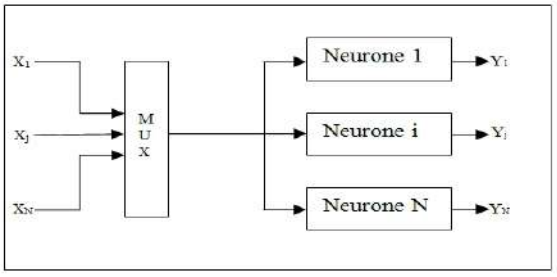


Fig 3.3: Architecture de la couche.

3.1.2. Le neurone :

L'élément de base de la partie Feed-Forward est le neurone. Il est constitué d'un bloc MAC, un bloc des poids synaptiques, et un bloc d'activation, comme on le voit dans la figure 3.4

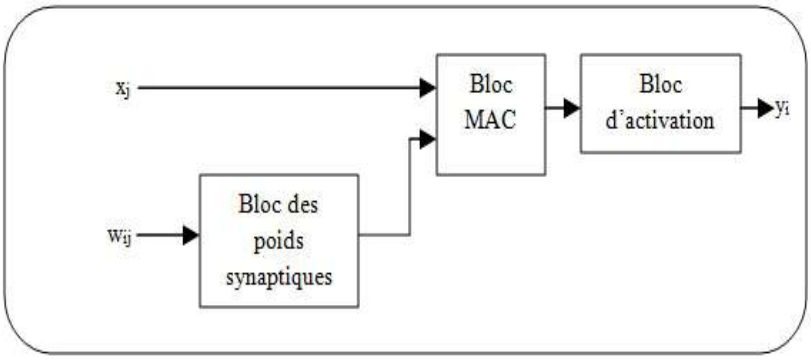


Fig 3.4: Architecture du neurone.

3.1.3. Le bloc des poids synaptiques :

Pour les poids synaptiques il suffit d'utiliser une mémoire pour le stockage de ces valeurs.

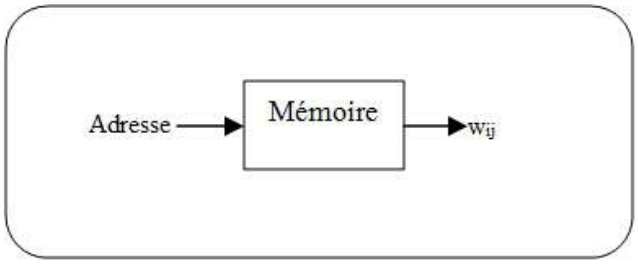


Fig 3.5: Le bloc des poids synaptiques.

3.1.4. Le bloc MAC :

Ce bloc modélise le produit –accumulation. Il est composé d'un multiplieur et d'un accumulateur figure 3.6

Le multiplieur : il permet de faire le produit entre les entrées du neurone et les poids synaptiques, qui est nécessaire pour le calcul de la somme pondérée. Pour la modélisation de la multiplication.

L'accumulateur : il est composé d'un additionneur et d'un registre à décalage. Il commence par le changement de la première valeur reçue du multiplieur et il la charge dans le registre, puis il additionne les prochaines données reçues du multiplieur avec celle qui se trouve dans le registre. **[19]**

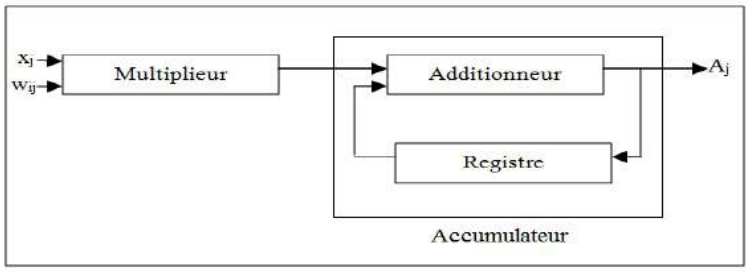


Fig 3.6: Architecture de bloc MAC.

3.1.5. Le bloc d'activation :

Le rôle de bloc de la fonction d'activation est de prendre la valeur de la somme pondérée calculée par le neurone et de lui appliquer la fonction sigmoïde, pour générer la valeur d'activation du neurone. **[19]**

4. Apprentissage du réseau de neurones :

L'idée principale autour de cette application est l'utilisation de l'algorithme de la rétro propagation du gradient (RPG) la partie feed-forward applique comme algorithme d'apprentissage supervisé à un perceptron multi couche (MLP).
Cette conception nécessite le passage par une phase de conception software à fin de pouvoir bien dimensionner le réseau et tester ces performances avons de passer à la phase de la conception et l'implémentation hardware. Pour cela nous avons utilisé l'outil « **Matlab 7.6 ».**
Le choix de Matlab était basé sur l'avantage que ce dernier possède une librairie riche des fonctions prédéfinies qui facilitent la tache de programmation des fonctions complexes comme celles de l'algorithme de la rétro propagation du gradient (**RPG**). Cette librairie dite « Neural network » contient toutes les applications des réseaux de neurones et parmi laquelle on trouve les différentes fonctions prédéfinies qui nous avons utilisé dans cette application à noté : **[20]**

1. *newff :* c'est une nouvelle fonction modélisant l'algorithme Feed_Forward, à chaque appel de la fonction newff, l'outil Matlab crée un nouveau réseau avec une zone de dialogue.
2. *train :* forme les connexions dans le réseau selon la fonction de transferts utilisés et les performances choisies.
3. *sim :* simule le modèle grâce à l'outil « Simulink » en utilisant tous les arrangements de dialogue de paramètre de simulation comprenant des options de la zone de travail I/O.

4. *plot :* dessine le résultat de la fonction train dans un plan.
5. *logsig :* c'est la fonction sigmoïde définie par la forme mathématique suivante :

$$y = \frac{1}{1+\exp(-x)}$$

Où

y : est la sortie du bloc de la fonction d'activation.

x : est la somme pondérée présente à l'entrée du bloc de la fonction d'activation.

6. *purelin:* c'est une fonction de transfert définie par la forme mathématique suivante :

$$y = purelin(x) = x$$

Où

x : une valeur présentée à l'entrée de la fonction linéaire *purelin*

4.1 Numérisation de la fonction logarithmique :

La fonction logarithmique $f(x)=\log_{10}(x)$ peut être approximée par des segments linéaires

Dans notre cas la fonction logarithmique est approximée par sept segments linéaires de la forme Ax+B.

Les valeurs de cette fonction approximée par sept segments linéaires sont données par le tableau suivant :

A	B	gamme d'entrée(x)	gamme de sorties
16	0	$0...\frac{1}{64}$	$0\ldots\frac{1}{4}$
8	$\frac{1}{8}$	$\frac{1}{64}\ldots\frac{1}{32}$	$\frac{1}{4}\ldots\frac{3}{8}$
4	$\frac{1}{4}$	$\frac{1}{32}\ldots\frac{1}{16}$	$\frac{3}{8}\ldots\frac{1}{2}$
2	$\frac{3}{8}$	$\frac{1}{16}\ldots\frac{1}{8}$	$\frac{1}{2}\ldots\frac{5}{8}$
1	$\frac{1}{2}$	$\frac{1}{8}\ldots\frac{1}{4}$	$\frac{5}{8}\ldots\frac{3}{4}$
$\frac{1}{2}$	$\frac{5}{8}$	$\frac{1}{4}\ldots\frac{1}{2}$	$\frac{3}{4}\ldots\frac{7}{8}$
$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{2}\ldots1$	$\frac{7}{8}\ldots1$

Tab 3.1 : Numérisation de la fonction logarithmique.

4.2 Architecture :

Couche d'entrée : elle comporte un seul neurone correspondant à l'entrée x

Couche cachée : le choix de la taille de la couche cachée est souvent réalisé d'une manière empirique, car il n'existe aucune formule d'optimisation. Dans le cadre de notre application, nous avons fait varier le nombre de neurones cachés entre 1 et 20, selon l'erreur quadratique et le temps de réponse le nombre choisi est 10

Couche de sortie : la sortie est représentée par un seul neurone correspondant à la sortie y. La fonction d'activation de la couche de sortie est une fonction linéaire

5. Interprétation :

Pour étudier la convergence de la sortie. On a obtenue les courbes suivantes :

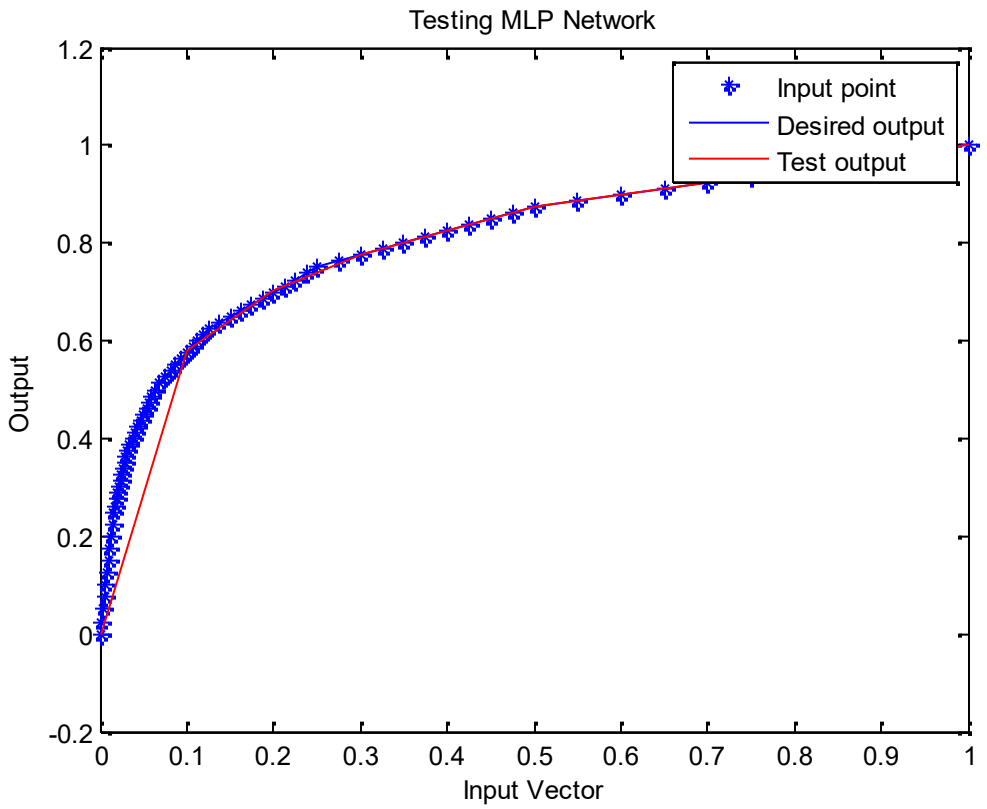


Fig 3.7 : Modèle test de simulation

Dans notre cas on fournit au réseau le vecteur d'entrée à traiter mais aussi la réponse attendue. Le réseau effectue une évaluation de la donnée, puis compare la valeur obtenue avec la valeur désirée, il va ensuite modifier ses paramètres internes afin de minimiser l'erreur constaté.

Les paramètres internes du réseau :

Couche cachée : elle contienne 10 neurones donc 10 valeurs du poids et 10 valeurs du biais

Poids	Biais
56.7678	-55.1585
55.4407	-49.6143
53.6202	-42.8060
71.0215	-49.5922
-157.3374	94.6788
-27.2410	12.4863
-46.6128	16.5270
29.1703	-6.4230
-14.6126	-6.4834
156.8506	-1.0374

Tab 3.2 : Valeurs pour la couche cachée

Couche de sortie : elle contienne un seul neurone donc une valeur du biais (biais=0.5646) et 10 valeurs du poids

Poids
0.0191
0.0218
0.0260
0.0241
-0.0186
-0.0763
-0.0298
0.0920
-332.2944
0.2556

Tab 3.3 : Valeurs pour couche sortie

Pour un bon choix de l'architecture il faut vérifier que le réseau présent une bonne convergence rapide toute en respectons l'erreur minimale de cette convergence figure3.8. Le

cas de cette application ou l'architecture (1 10 1) est adopté après une phase d'apprentissage qui a présenté :

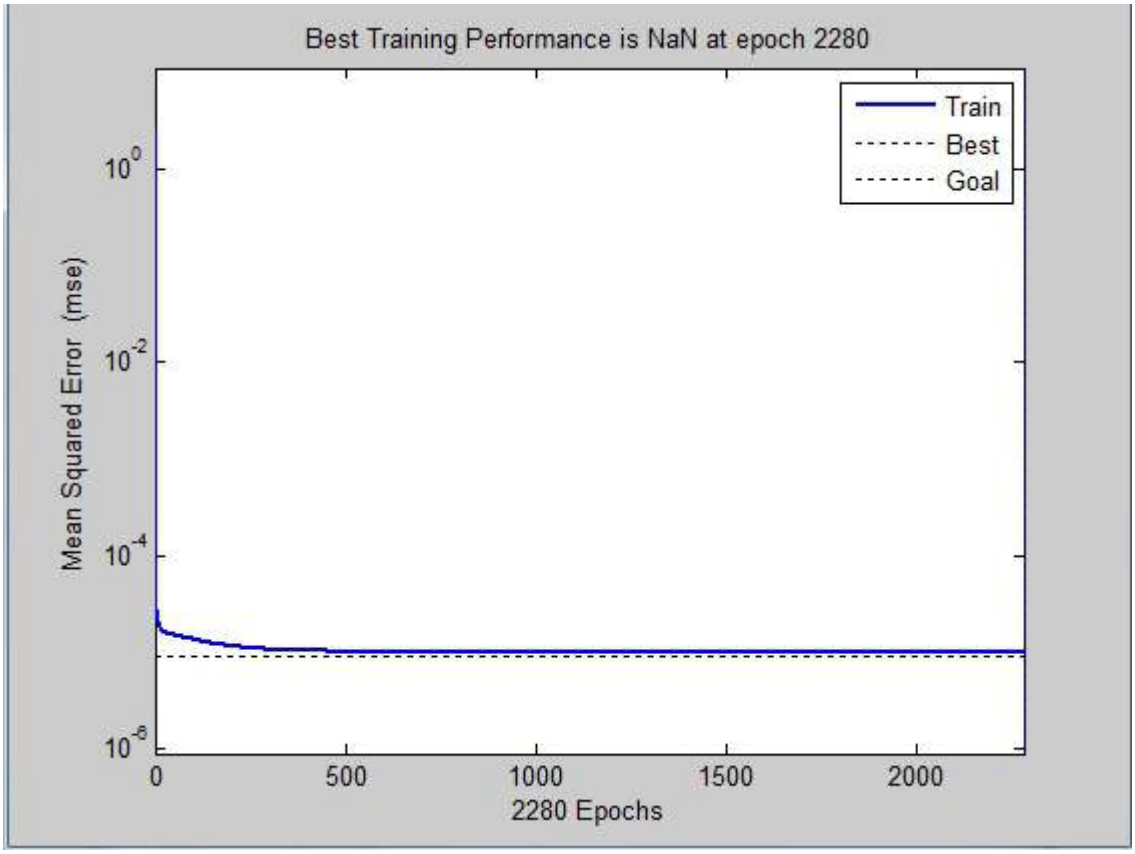


Fig 3.7 : courbe de convergence du réseau.

- Une convergence rapide (2280 itérations).
- une erreur de convergence de l'ordre de (10e-5).

La relation entre la sortie calculée et la sortie désirée montre la performance et la qualité du modèle développé par la méthode de propagation vers l'avant des réseaux de neurones.

Une fois que l'algorithme converge, c'est-à-dire la sortie converge vers une valeur désirée, on prend par la suite les valeurs des synaptiques W pour les utilisés dans l'étape de modélisation

6. Choix de langage pour la modélisation :

Les technique 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 très utilisés dans le domaine du hardware. Ainsi que, les techniques softwares existants, 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. **[21][22]**

6.1 Motivations pour les techniques orientées 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. **[21]**

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 cout de conception faible et une fiabilité augmentée.

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

-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'objet dynamique.

-Possibilité d'employer les synthèses software existantes et de technique de vérification.

6.2 Types de données :

Les types de données sont présentés au hardware de la même manière que le software. Les langages de programmation offrant 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 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. **[22]**

6.3 Modélisation 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 la classe. En générale, 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.

6.4 Dérivation de composants spécialises :

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.

6.5 Modélisation orientée objet des RNAs :

Pour classifier 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 :

a. 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. **[23]**

Méthodes + données = objet.

b. 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 signifier 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.

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 ». A ce propos on peut remarquer qu'en programmation structurée, une procédure pouvait également être caractérisée par ses spécifications, mais que, faut d'encapsulation, l'abstraction des données n'était pas réalisée.

L'encapsulation des données présente un intérêt manifeste en matière de qualité de logiciel.

Elle facilite considérablement la maintenance : 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 sur pas avec la programmation structurée). De la même manière, l'encapsulation des données facilite grandement la réutilisation de l'objet comme les composants hardware.

c. Classe : En P.O apparait 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 types classe ; en P.O.O on dit aussi qu'un objet est une instance de la classe. **[22] [23]**

d. Héritage : Un autre concept important en P.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 à volonté. **[22] [24]**

6.6 Modélisation des classes du RNA :

Dans la pratique, en utilisant le concept de la classe comme « composant logiciel », en effet, dans un souci de réutisabilitié, 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 neurone appliqué à la

Cette application a nécessité :

- Une classe registre
- Une classe PE

6.6.1 Les classes registres :

Ceux sont des registres 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 d'initialisation (*INIT*) (initialise le registre à 0 au début de chaque opération).

Une fonction membre de chargement (*LOAD*) (charge le registre par des valeurs d'entrée ou calculées).

Une fonction membre de lecture (*READ*) (permet de lire les valeurs internes du registre

6.6.2 La classe PE :

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 membre suivantes :

Une fonction membre d'initialisation *(INIT)*.

Une fonction membre de multiplication et accumulation *(MAC)*.

7. Résultats et interprétations

7.1 Environnement de développement :

Notre programme est écrit en Borland C++ version 4.5 sous windows. Le choix de la programmation sous Windows a été pris à cause de l'interface graphique qu'offre l'environnement Windows ; le choix du langage Borland C++ 4.5 était fondamental puisqu'il renferme des outils très puissants.

7.2 Les étapes de construction du programme :

- Nous avons crée une classe registre qui contienne 10 mots
- Nous avons crée une classe PE pour la couche cachée qui possède une fonction d'activation sigmoïde.
- Nous avons crée une classe PE pour la couche de sortie qui possède une fonction d'activation linéaire.
- Après nous avons construis le programme main qui contiens les différentes instructions qui permet d'introduire les paramètres trouvées dans la phase d'apprentissage. On prend 10 processeurs pour la couche cachée, trois registres (registre pour les poids cachée, registre pour les biais et l'autre pour la sortie) on les remplis en utilisant les instructions qui permettent de charger les valeurs et on termine par une boucle de teste qui permet d'afficher les valeurs d'E/S.

7.3 Simulation et résultats :

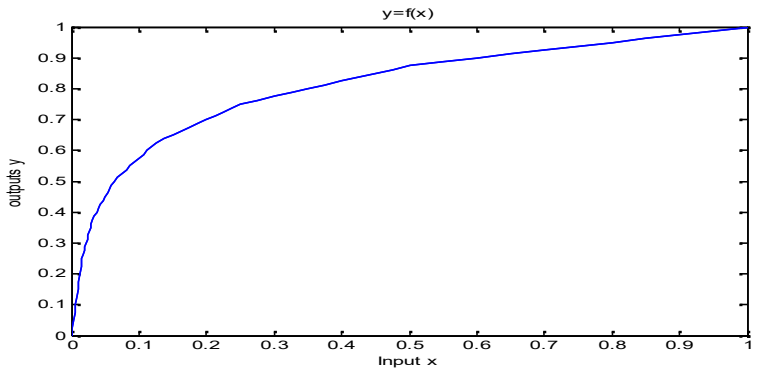


Fig 3.9 : courbe de convergence y=f(x).

La figure montre la caractéristique y=f(x), on voie que cette courbe prend la forme de celle trouver par la simulation (phase d'apprentissage)

Calcul d'erreur :

e	**0**	**0.033**	**0.066**	**0.099**	**0.132**	**0.165**	**0.198**	**0.231**	**0.264**
S	**0.00024**	**0.37856**	**0.50297**	**0.57860**	**0.62824**	**0.66529**	**0.69906**	**0.73157**	**0.7575**
d	**0**	**0.398**	**0.507**	**0.573**	**0.632**	**0.665**	**0.698**	**0.731**	**0.757**

0.297	**0.33**	**0.363**	**0.396**	**0.429**	**0.462**	**0.495**	**0.528**	**0.561**	**0.594**
0.77486	**0.78925**	**0.8066**	**0.8232**	**0.8388**	**0.85646**	**0.8726**	**0.883**	**0.889**	**0.8959**
0.7735	**0.79**	**0.8065**	**0.823**	**0.8395**	**0.856**	**0.8725**	**0.882**	**0.8902**	**0.8985**

0.627	**0.66**	**0.726**	**0.792**	**0.825**	**0.858**	**0.891**	**0.924**	**0.957**	**0.99**
0.9111	**0.9133**	**0.9338**	**0.9471**	**0.9577**	**0.9638**	**0.9720**	**0.8916**	**0.9892**	**0.9918**
0.9067	**0.915**	**0.9315**	**0.948**	**0.9562**	**0.9645**	**0.9727**	**0.989**	**0.9937**	**0.9975**

Tab 3.4 : Valeurs de sortie désirée **d** et sortie obtenue **s**

$$E = \frac{1}{2}\sum_{i=1}^{7}(S-d)^2 \qquad (3\text{-}5)$$

Application numérique :

(E=2,9*10^{e}-5

On constate que l'erreur calculée est plus proche que celle trouvée par le simulateur, donc comme conclusion on dit que notre programme est bien adapté à l'application.

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 (propagation vers l'avant)

Bibliographie

[01] CHIKH Mohammed Tahar, **« Amélioration des images par un modèle de réseau de neurones**», Université Abou-Bakr Belkaid –Tlemcen, mémoire de fin d'étude 2011.

[02] Claude Touzet, « **les réseaux de neurones artificiels, introduction au connexionnisme**», Université Hassani, ENSA Khouribga, 2O14.

[03] Radja Marouf, **« Amélioration de l'apprentissage d'un Modèle Neuronal pour la reconnaissance des anomalies cardiaques»,** université de Tlemcen, mémoire de fin d'étude juillet 2010.

[04] El Hannani F., Koudacke L**, «** ***Élaboration d'un système de détection et*** *de* **reconnaissance des visages & identification des émotions faciales** », Université de Sidi Bel Abess, 2002.

[05] Jadouin J. F**., «** ***Les réseaux neuromimétiques*** », Edition Hermes, 1994

[06] belkadi abdallâh « Approche neuronale pour la classification des images hyper spectrale « université d Oran

[07] Jean-Philippe Rennard : **«** ***Réseaux Neuronaux une Introduction Accompagnée d'un*** **Modèle Java** . ISBN 2-7117-4830-8 Vuibert, Paris, 2006

[08] Judith Dayhoff: **"***Neural Network Architectures An Introduction***".** ISBN 0-442-20744-1 Van Nostrand Reinhold, 1990

[09] M. Laurent Gatet: **"** ***Intégration de Réseaux de Neurones pour la Télémétrie Laser*****".** Thèse de doctorat. Institut National Polytechnique De Toulouse. Septembre 2007

[10] hamza chérif « classifieurs SVM et réseaux de neurones » université de Tlemcen 2011

[11] Hadjila F. & Bouabdellah R**., «** ***Reconnaissance des visages par les réseaux*** **de** ***neurones*** **»,** Mémoire de PFE, Université de Tlemcen, 2003

[12] P. Quinton and Y. Robert**. « Algorithmes et architectures systoliques »** IRISA Campus de Beaulieu 35042 Rennes Cedex 1989

[13] architecture systolique **épreuve commune suggérer pour les candidats**

[14] Mourad zakhama **« implémentation technologie ITGE (VLSI) d un filtre adaptatif sur logique floue pour l égalisation des canaux »**, memoire présenté a l'université du Québec a trois rivières avril 2000

[15] J.OUALIER, S.SAUCIER, TRITHE. **«fast design of digital dedicated neurochops »**in VLSI design of neural network.edited by U.RAMACHER .academic Publisher 1999

[16] laidi Kamel **« implémentation orienté hardware des réseaux de neurones artificiels : application a la navigation d un robot mobile autonome »** thèse de magistère a l'ENP septembre 1998

[17] J.N.HWANG, S.Y.KUNG **« an unifed systolic architecture for artificial neural network »** edited by U.RAMACHER et U.RUCKERT.academic publishers 1991

[18] P.QUINTON .YUES ROBERT **« algorithmes et architecture systoliques »**édition MASSON .1992

[19] Fayçal Gmana et Mohamed Raymon « **simulation et implémentation d un réseau de neurones artificiels MLP sur un circuit FPGA »** dirigé par Mr Fayçal Benrkia

[20] Salhi Dalila **« implémentation d'un classificateur neuronal basé sur l'algorithme RPG des rythmes cérébraux sur FPGA** » ENP mars 2009

[21] S.Kumar,J.H.Aylor **« objected-oriented techniques in hardware design »**IEEE computer.vol27 N.6 ,june 1994

[22] S.Kumar,J.H.Aylor **« the Co-design of Embedded Systems unified Hardware/Software représentation»** kluwer academic publishers.USA 1996

[23] A.chohra,A.Farah, « hybrid navigation approach combining neural networks and fuzzy logic for autonomous mobile robots » chiba japan step 1-6 1997

[24]G.DE.MICHELI. **« synthesis and optimization of digitals circuits »,** INC YSA 1994

10.1. Régularisation 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 :

$$\begin{cases} U(l+1)=W\,a(l)+\Theta \\ a(l+1)=F(U(l+1)) \end{cases} \qquad (2\text{-}1)$$

Où West la matrice des poids reliant les neurones la couche à la couche l+1,$a_i(l)$ et la sortie du neurone ide 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)** suivi par la fonction d'activation sigmoïde afin d'être implémentée sur des réseaux systoliques.

Pour un réseau multicouche ayant N neurones dans la couche (l+1) et N neurones dans la couche (l), le premier terme dans l'équation (2-1) peut être explicité comme suit :

$U_1(l+1)=w_{11}a_1(l)+w_{12}a_2(l)+\ldots+w_{N-1}a_N(l)+\Theta_1$

$U_2(l+1)=w_{21}a_1(l)+w_{22}a_2(l)+\ldots+w_{2N}a_N(l)+\Theta_2$

.

.

.

$U_{N-1}(l+1)=w_{N-1\ 1}a_1(l)+w_{N-1\ 2}a_2(l)+\ldots+w_{N-1\ N}a_N(l)+\Theta_{N-1}$

$U_N(l+1)=w_{N\ 1}a_1(l)+w_{N\ 2}a_2(l)+\ldots\ldots\ldots+w_{N\ N}a_N(l)+\ \Theta_N$

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 :

$U_1(l+1)=w_{11}a_1(l)+w_{12}a_2(l)+\ldots+w_{1\ N-1}a_{N-1}(l)+\Theta_1$

$U_2(l+1)=w_{21}a_1(l)+w_{22}a_2(l)+\ldots+w_{2N}a_N(l)+\Theta_2$

.

.

.

$U_{N-1}(l+1)=w_{N-1\ N-1}a_{N-1}(l)+w_{N-1\ N}a_N(l)+\ldots+w_{N-1\ 1}a_1(l)+w_{N-1\ 2}a_2(l)+\ \Theta_{N-1}$

$U_N(l+1)=w_{N\ N}a_N(l)+w_{N\ 1}a_1(l)+\ldots+w_{N\ N-2}a_{N-2}(l)+w_{N\ N-1}a_{N-1}(l)+\ \Theta_N$

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 :

Annexe

Annexe :

1/ Programmation feed-forward (Matlab) :

```
coef = [ 16 16  16  16  16  16  16  16  16  16 16 ...
         8   8   8   8   8   8   8   8   8   8  8 ...
         4   4   4   4   4   4   4   4   4   4  4 ...
         2   2   2   2   2   2   2   2   2   2  2 ...
         1   1   1   1   1   1   1   1   1   1  1 ...
         1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 ...
         1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4;...

     0   0   0   0   0   0   0   0   0   0  0 ...
     1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 ...
     1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 ...
     3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 3/8 ...
     1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 ...
     5/8 5/8 5/8 5/8 5/8 5/8 5/8 5/8 5/8 5/8 5/8 ...
     3/4 3/4 3/4 3/4 3/4 3/4 3/4 3/4 3/4 3/4 3/4];

x=[ 0:1/640:1/64,  1/64:1/640:1/32,  1/32:1/320:1/16,  1/16:1/160:1/8,
1/8:1/80:1/4, 1/4:1/40:1/2, 1/2:1/20:1]
for i=1:length(x)
     y(i)=coef(1,i)*x(i)+coef(2,i);
 end
 x1 =[ 0:1/640:1/64,  1/64:1/640:1/32,  1/32:1/320:1/16,  1/16:1/160:1/8,
1/8:1/80:1/4, 1/4:1/40:1/2, 1/2:1/20:1]


 for i=1:length(x1)
     y1(i)=coef(1,i)*x1(i)+coef(2,i);
 end
 figure(1); % making a new figure window
plot(x,y,'b*',x1,y1,'b') % drawing the graph (x,y), (x1,y1)
title('Training Vectors'); % a title of figure
xlabel('Input Vector x'); % x label of figure
ylabel('Target Vector y for trainning'); % y label of figure
legend({'input point','desired output y=ax+b'})
% legend of figure
% training the MLP Neural network
%-----------------------------------
% using the newff toolbox function
% input range : 0 ~ 1
% number of neurons of hidden layer : 10, Log-Sigmoid Transfer function
% ouput layer : 1 Linear Transfer function
net = newff([0 1],[10 1],{'logsig'  'purelin'});
% define the MLP neural network architecture
net.trainParam.epochs = 3000;
% Maximum epochs is 10000 ( iteration number of optimization )
net.trainParam.goal = 0.000009; % Error limit is 0.000001
%-------------------------------------------------
% training the MLP by using a training pattern
%-------------------------------------------------
net = train(net,x1,y1);
% training the MLP neural network by using a train function
%-------------------------------------------------
% To draw the result making the test pattern
%-------------------------------------------------
Test=[0:0.01:1]; % making the input of test pattern
```

```
Output=sim(net,Test); % simulate the MLP network by using a sim function
% Output is the result of simulation
%-------------------------------------------------
% drawing the figure of result of simulation
%-------------------------------------------------
figure(2);
plot(x,y,'b*',x1,y1,'b',Test,Output,'r')
title('Testing MLP Network');
xlabel('Input Vector');
ylabel('Output');
legend({'Input point','Desired output','Test output'})
%-------------------------------------------------
```

2/ programmation feed-forward (C++) :

```
#include <iostream.h>
#include <math.h>
//---------------------- classe registre 10 mots----------------
class registre
{
public:
double cel[10];
int i;
void load(double x[10])
{for (i = 0; i < 10; i++) {cel[i]=x[i];}}
                    };
//----------- classe du PE pour la couche cachée-----------------
  class PE_cachee
  {
   private:
    double W, E,Biais, prod_acum, act;
   public:
   void init(double x) { prod_acum=0;Biais=x;}
   void mac (double x, double y) { W = x; E = y; prod_acum=W*E +prod_acum;
  double activation() { act=1/(1+exp(-(prod_acum+Biais))); return(act);}
  };
//----------- classe du PE pour la couche de sortie----------------
 class PE_sortie
                  {
   private:
```

```
        double W, E,Biais, prod_acum, act;
      public:
     void init(double x)  {  prod_acum=0;Biais=x;}
     void  mac (double x, double y)  { W = x;  E = y;  prod_acum=W*E +prod_acum;  }
     double activation() { act=(prod_acum+Biais);  return(act);}
   };
 //----------  programme principale----------------
void main ()
          {
  int k,i;
          double sortie_HL[10];// sorties de la couche cacheee
          double fin;  //  juste pour terminer l'execution
          double poids_cachee[10]={56.7678,55.4407,53.6202,71.0215,-157.3374,-27.2410,-
46.6128,29.1703,-14.6126,156.8506};                  // poids de la couche cachee
          double biais_cachee[10]={-55.1585,-49.6143,-42.8060,-49.5922,94.6788,12.4863,16.5270,-
6.4230,-6.4834,-1.0374};

          double poids_sortie[10]={0.0191,0.0218,0.0260,0.0241,-0.0186,-0.0763,-0.0298,0.0920,-
332.2944,0.2556};                                    // poids de la couche sortie
          double biais_sortie=0.5646;
PE_cachee PE1[10];        // 10 processeurs pour la couche cachee
                  registre registre1, registre2,registre3;
  //registre pour les poids cachee et autre pour la sortie
                  PE_sortie PE2;

          //---------------------------------- chargement et initialisation ----------------------

          registre1.load(poids_cachee);          //  chargement des poids dans le registre
```

```
registre3.load(biais_cachee);
registre2.load(poids_sortie);            //  chargement des poids dans le registre

double  entrenet[35]  ;                                          // entree du reseau
double sortenet[35];                                             //sortie du reseau

entrenet[0]=0;

for (k = 0; k < 35; k++)
{     entrenet[k+1]=entrenet[k] +0.033;
for (i = 0; i < 10; i++)   PE1[i].init(registre3.cel[i]);          // initialisation de tous les PEs
 PE2.init(biais_sortie);                                         // initialisation de tous les PEs
        //--------------------------------------sorties de la couche cachée--------------------------

for (i = 0; i < 10; i++) {  PE1[i].mac(registre1.cel[i],entrenet[k]);     // produit et accumulation
sortie_HL[i]=PE1[i].activation();     }       // fonction d'activation

        //---------------------------------------sorties du reseau --------------------

        for (i = 0; i < 10; i++) {  PE2.mac(registre2.cel[i],sortie_HL[i]); }
                                                // produit et accumulation
sortenet[k]=PE2.activation();                   // fonction d'activation
        //-------------------------------------------------------------------------------------------------------
}
for (k = 0; k < 35; k++) {cout << "entrée  sortie : " <<entrenet[k]<<"     " <<sortenet[k]<<"\n"; }
```