REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI TIZI-OUZOU
FACULTE DE GENIE ELECTRIQUE ET D'INFORMATIQUE
DEPARTEMENT D'ELECTRONIQUE

Mémoire de fin d'études

en vue de l'obtention du diplôme Master II

Spécialité : Electronique

Option : Electronique biomédicale

Thème :

Implémentation d'un réseau de neurones dans un microcontrôleur

Proposé et Encadré par: Pr. M.Laghrouche

Développé par : GANA Massine

Année universitaire : 2015/2016

Remerciements :

Mes remerciements à mes parents qui étaient toujours à mes cotés, à mon frère

et à toute ma famille et à tous mes amis.

Mes remerciements les plus sincères vont à tout ceux qui m'ont soutenu, en

particulier mon promoteur Mr Laghrouche et Mr Idjeri pour leurs conseils et

orientations.

Je remercie aussi les membres du jury.

S o m m a i r e

Introduction Générale

Introduction Générale

Depuis toujours, l'homme n'a cessé d'être curieux, la science en fut toujours la bronche la plus bénéficière, Isaak Newton s'est demandé pourquoi la pomme ne s'est-elle pas envolée vers le haut, Einstein s'est demandé qu'est ce qui se passerait si l'on pouvait chevaucher un rayon de lumière, Et actuellement, la curiosité a amener les scientifiques à imaginer et à innover le domaine de l'intelligence artificielle; cette dernière, bien que critiquée, par de grandes personnalités, tels que Bill Gates et S.W Hawkins [1] sur le faite qu'elle efface petit à petit le charme de l'humanité, néanmoins, elle est soutenu par d'autres tel que J. De Rosnay : << **Il faut avoir moins peur de l'intelligence artificielle que de la stupidité naturelle, d'où l'importance de l'éducation, de la formation, et de l'apprentissage** >> [2].

C'est ainsi que l'homme continu de se poser des questions : Et si on pouvait s'inspirer du comportement des fourmis pour résoudre des problèmes ? Il créa alors l'algorithme de colonie de fourmis [3]; ou encore : Et si on pouvait créer des machines qui seraient capables de penser comme les humains ? Il inventa alors le Réseau de neurones artificiels, connus généralement sous l'acronyme ANNs (Artificial Neural Networks) [4], et ainsi de suite, Notre travail sera basé justement sur la création, la manipulation et l'utilisation des Réseaux de neurones artificiels qu'on a implémenté dans une carte Arduino.

Une des tâches essentielles du cerveau consiste à transformer des informations en connaissances : identifier les lettres qui constituent un texte, les assembler en mots et en phrases, en extraire un sens, sont des activités qui nous paraissent naturelles une fois l'apprentissage nécessaire accompli avec succès. L'objectif des Réseaux de neurones est d'imiter, à l'aide d'algorithmes exécutés par des ordinateurs, la capacité qu'ont les êtres vivants à apprendre par l'exemple, ces réseaux ont affiché des résultats spectaculaires depuis quelques années, dans des domaines très différents : vision par ordinateur, reconnaissance manuscrite et vocale, finance, médecine, production industrielle, météorologie, géologie, physique, etc. Outre les aspects technologiques et commerciaux actuels, le sujet est fascinant car ces réseaux sont inspirés de réseaux de neurones biologiques [5], notamment ceux des cerveaux humains. Ils nous renvoient ainsi à nos propres capacités cognitives et agitent facilement le spectre de l'intelligence artificielle.

Les Réseaux de neurones offrent des solutions plus que satisfaisantes pour un grand nombre de problèmes, dont certains sont difficiles à traiter par les approches classiques

(analytiques, numériques,…) [6,7], ainsi dans le domaine médicale, ils sont largement appliquées pour le diagnostic de maladies, l'analyse du signal électronique issu de différents capteurs et appareils (tels que l'ECG et l'EEG, etc.) [8-11], et de même l'analyse d'images médicales et de radiologie, et là on introduit couramment le terme de "Classification". Quant au domaine de "modélisation", les ANNs permettent entre autres de linéariser des capteurs [12-14] dans le but d'obtenir des résultats adéquats à n'importe quelle étude et à n'importe quel type de montages auxquels on se fit.

L'implémentation des ANNs est aujourd'hui beaucoup plus simple a mettre en œuvre vu l'avancée fulgurante de la technologie et des systèmes informatiques, on abordera trois types d'implémentation : les circuits FPGA, MATLAB et les microcontrôleurs, Certes les résultats obtenues avec MATLAB sont plus fiables, néanmoins l'aspect pratique est en manque d'aisance et de confort, l'idéal serait donc d'embarquer ces modèles dans des cartes électroniques, c'est pour ça qu'on se focalisera en masse sur le troisième cité et plus précisément un microcontrôleur de type ARM CORTEX M3 [15] contenu dans une carte d'interface ARDUINO, Pour cela, on a divisé notre travail en deux grandes parties :

- la première étant purement théorique, et se composera de deux chapitres : le premier fera objet d'une description générale des Réseaux de neurones passant du domaine biologique au domaine artificiel, suivi par quelques domaines d'applications et la procédure quant' à la réalisation et la mise en œuvre de ces Réseaux de neurones, alors que dans le deuxième, toujours dans l'aspect théorique, décrira minutieusement trois méthodes d'implémentation d'un ANN dans différents matériels électroniques ou logiciels qu'on définira au moment opportun.

- la deuxième partie sera entièrement consacrée à la partie pratique, et aux avancées successives de nos résultats, on montrera aussi les difficultés auxquels on a été confronté et les solutions qui nous ont permis d'arriver à nos fins.

Chapitre I :

Les réseaux de Neurones

Introduction :

Depuis la création des ordinateurs dans les années 40s, l'informatique ne cesse d'évoluer, chaque décennie, voire chaque année, une nouveauté s'y affiche, l'ordinateur est aujourd'hui beaucoup plus puissant qu'il était il y'a quelques années [16]; n'empêche que la résolution de certains problèmes d'optimisation restent un souci considérable chez les informaticiens, c'est pour cela que plusieurs algorithmes ont fait surface pour pallier a ce genre de problèmes, et parmi ces algorithmes, on trouve le Réseau De Neurones artificiel (Artificiel Neural Network) [17,18].

Le principe des Réseau de neurones se base sur une approche mathématique du fonctionnement du cerveau humain, dans ce chapitre, on étudiera le réseau de neurones artificiel, son modèle mathématique, ses architectures ainsi que ses domaines d'applications.

I) - Le Réseau de neurone biologique :

I-1) - Le neurone biologique:

I-1-a) - Définition :

Le cerveau humain contient un grand nombre de neurones (environ 100 milliards) fortement interconnectés (environ 10^{15} connexions) [19,20] constituant ainsi des réseaux de neurones. Un neurone est une cellule du système nerveux spécialisée dans la communication et le traitement d'informations, le terme de neurone fut introduit par Waldeyer en 1881.

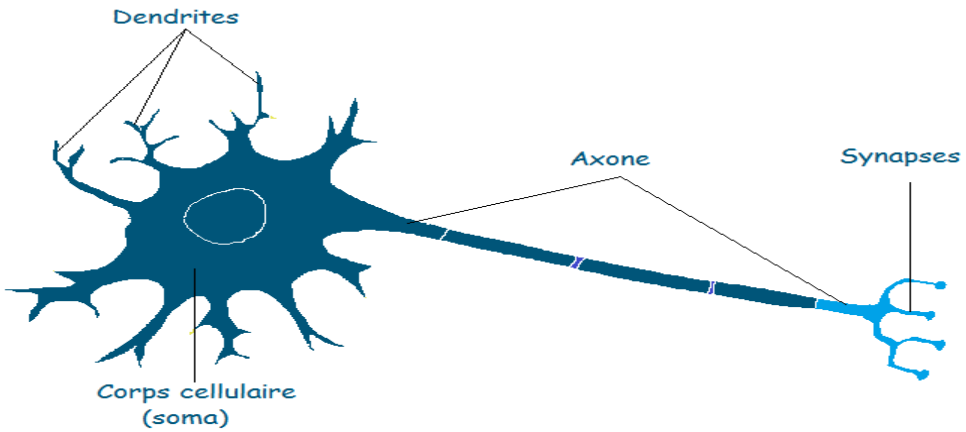


Figure I.1 : Neurone biologique

I-1-b) - Corps Cellulaire : Souvent appelé " Le Soma" effectue la somme des informations qui lui parviennent, il traite ces mêmes informations et renvoie le résultat sous forme de signaux électriques vers les autres neurones à travers l'axone.

I-1-c) - Axone : C'est un long prolongement fibreux du neurone, il fait la liaison entre les neurones et leurs conduit l'influx nerveux.

I-1-d) - Dendrite : Joue le rôle de capteur, car il reçoit les informations venant de l'extérieur vers le corps cellulaire. Certaines dendrites peuvent avoir un effet moteur favorisant la transmission d'une information dans l'axone, d'autres au contraire ont un effet inhibiteur qui bloque la transmission de l'influx dans l'axone.

I-1-e) - Synapses : c'est une zone située entre deux neurones (cellules nerveuses) et assurant la transmission des informations de l'une à l'autre.

I-1-f) - Influx Nerveux : c'est une activité électrique qui parcourt les axones sous la forme d'une séquence de potentiel d'action, provocant la libération des neurotransmetteurs au niveau des synapses. Ces potentiels d'action sont générés par des échanges d'ions entre l'intérieur et l'extérieur des neurones. La vitesse de propagation de l'influx nerveux est de 100m/s.

I-2) - Fonctionnement du neurone biologique :

Les dendrites reçoivent des autres neurones, des informations sous forme de signal électrique (influx nerveux), elles seront envoyés vers le corps cellulaire (Soma) et plus précisément le noyau qui fera la somme des influx arrivant aux dendrites tout en affectant des poids dits "poids synaptiques".

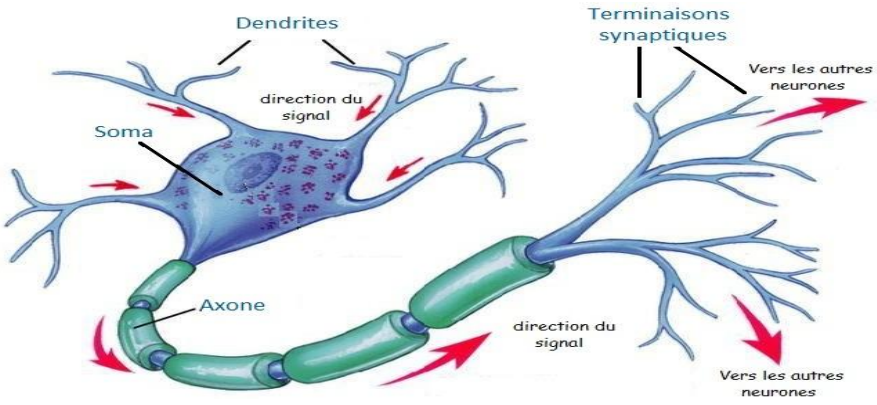


Figure I.2 : Circulation de l'information dans le neurone.

Le résultat transite le long de l'axone jusqu'aux terminaisons synaptiques; et avec l'arrivé du signal, les vésicules synaptiques se fusionnent avec la membrane cellulaire pour libérer les neurotransmetteurs.

II) - Le Réseau de neurone artificiel :

II-1) - Du neurone biologique au neurone formel :

Si nous essayons d'interpréter autrement le fonctionnement précédent du neurone biologique, on pourrait dire que le noyau fait la somme du produit des informations acquis par les dendrites et les poids attribués a ces mêmes informations, ensuite le corps cellulaire analyse le résultat, si ce dernier est supérieur à un seuil, il sera envoyé vers l'axone qui, soit ce subdivisera pour alimenter d'autres neurones, soit à attaquer un élément moteur (muscle par exemple); Mc Culloch et Pitts [21], furent les premiers à mettre en œuvre les réseaux de neurones, ils ont résumé cette approche mathématique par l'équation du neurone formel [22] :

$$S = F\left(\sum_{i=1}^{n} \omega_i E_i\right) \qquad \textbf{I. 1}$$

où :

E_i : Représente l'entrée i du neurone.

ω_i : Représente le poids attribué à l'entrée i.

F : Représente la fonction d'activation (Seuil) du neurone

$\{1$ si F(x) > θ ; 0 sinon $\}$, avec θ : Seuil.

S : Sortie du neurone.

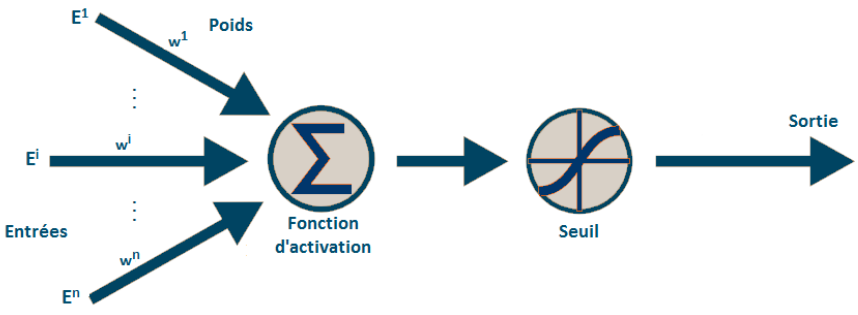


Figure I.3 : Neurone formel.

On en déduit l'équivalence suivante :

Réseaux de Neurones Biologiques	Réseaux de Neurones Artificiels
Soma	Neurone
Dendrite	Entrée
Axone	Sortie
Synapse	Poids

Tableau I.1 : équivalence RNA - RNB

II-2) - Historique :

1890 : La loi de fonctionnement pour l'apprentissage est présenté par W. James .

1943 : Warren Mc Culloch et Walter Pitts proposent le premier modèle du neurone formel.

1949 : Règle de Donald Hebb qui décrit que si les neurones d'une synapses sont activés d'une façon synchrone et répétée, la force de connexion synaptique est croissante [23].

1958 : Création du premier réseau de neurones par Rosenblatt, le "perceptron", ce dernier est inspiré du système visuel. Il permet d'apprendre et d'identifier des formes simples et aussi de calculer certaines fonctions logiques.

1969 : Marvin Minsky et Seymour Papert montrent les limites du perceptron, surtout à l'incapacité de résoudre des problèmes non linéairement séparables (exemple du XOR) [24].

≈≈≈ Période noire des Réseaux de neurones (≈ 15 ans) et beaucoup de déception chez les amateurs de l'intelligence artificielle.

1982 : John Hopfield réactive l'intérêt de l'utilisation de ce domaine grâce a sa découverte sur l'utilisation des réseaux récurrents (feed-back) après la première classe du perceptron [25].

1975 : Werbos propose l'idée d'une possible utilisation d'une rétropropagation du gradient.

1985-1986 : Le Cun et Parker (1985) continuent leurs recherches du pricipe de Werbos (1975), mais les travaux de Rumelhart (1986) furent le vrai départ de l'apprentissage des réseaux de neurones multicouche avec la méthode de rétropropagation du gradient.

II-3) - Domaines d'applications :

Cette dernière décennie a vu l'intégration de plusieurs bronches de différents secteurs dans le monde de l'informatique notamment les réseaux de neurones [26]; et ce grâce à sa capacité a résoudre pas mal de problèmes, on y trouve: le secteur des finances (prévision de bourse), le secteur des transports (détection des défauts dans les métros), le secteur des sciences et d'ingénierie (pilotage des véhicules autonomes), ou encore le secteur de la médecine (diagnostiquer les maladies et prédire la gravité d'une maladie quelconque).

II-4) - Caractéristiques des Réseaux de Neurones Artificiels :

II-4-a) - Présentation : Les réseaux de neurones artificiels sont donc des modèles inspirés du fonctionnement du cerveau humain, dans le but est de concevoir des machines capables de, non d'imiter, mais de se rapprocher au maximum du comportement de l'intelligence dont disposent les êtres humains, d'où la notion de "Intelligence Artificielle".

II-4-b) - Les Entrées : Chaque neurone possède plusieurs entrées, on note E_i $(1 \leq i \leq n)$ les entrées du réseau, avec n : nombre de neurones d'entrées; et à chacune d'entre elle on affecte un poids ω_i, on y ajoute un coefficient dit "biais" ω_0 supposé lié à une entrée $E_0 = 1$, la $i^{ème}$ information qui parviendra au neurone est le produit ω_i x E_i.

Le neurone calculera la somme : $\sum_{i=1}^{n} \omega_i$ x E_i, celle-ci sera le potentiel du neurone, la sortie sera générée par une fonction d'activation, qui sera très importante car elle déterminera par la suite le fonctionnement du réseau.

II-4-c) - Les fonctions d'activations : La fonction d'activation, ou fonction de transfert, est une fonction qui doit renvoyer un réel proche de 1 quand les "bonnes" informations d'entrée sont données et un réel proche de 0 quand elles sont "mauvaises". On utilise généralement des fonctions à valeurs dans l'intervalle réel [0;1]. Quand le réel est proche de 1, on dit que l'unité (le neurone) est **active** alors que quand le réel est proche de 0, on dit que l'unité est **inactive** [27]. Elle peut prendre de nombreuses formes : binaire, linéaire, sigmoïde, et bien d'autres encore. Le tableau ci-dessous dévoile les fonctions d'activations les plus utilisées :

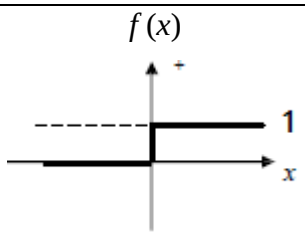
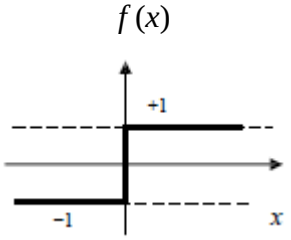
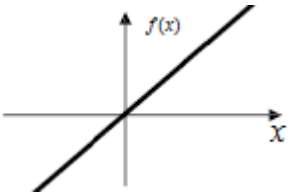
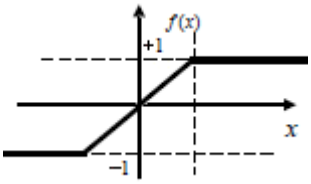
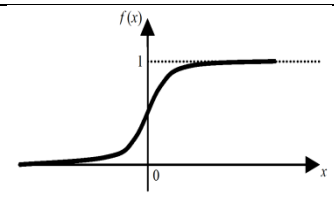
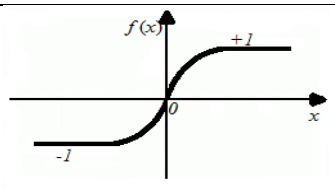
Catégories	Types	Equation	Allure
Seuil	Heaviside	$f(x) = \begin{cases} 0 \; si \;\; x < 0 \\ 1 \; si \;\; x \geq 0 \end{cases}$	
	Signe	$f(x) = \begin{cases} -1 \; si \;\; x \leq 0 \\ 1 \; si \;\; x > 0 \end{cases}$	
Linéaire	Identité	$f(x) = x$	
	Saturé symétrique	$f(x) = \begin{cases} -1 \; si \; x \leq -1 \\ 1 \;\; si \; x \geq 1 \\ x \;\; \text{sinon} \end{cases}$	
Non linéaire	Sigmoïde	$f(x) = \frac{1}{1 + e^{-x}}$	
	Tangente hyperbolique	$f(x) = \frac{2}{1 + e^{-x}} - 1$	

Tableau I.2 : les fonctions d'activations.

II-4-d) - La sortie : peut être utiliser en tant que résultat final, ou pour alimenter d'autres neurones.

II-5) - Apprentissage des Réseaux de neurones :

II-5-a) - Définition : L'apprentissage automatique est utilisé pour doter les ordinateurs et les machines de systèmes de vision; reconnaissance d'objets (visages, langages naturels, écriture...); aide aux diagnostics, médical notamment; classification des séquences d'ADN ...etc.

II-5-b) - Types d'apprentissage : Il présente plusieurs types, les plus utilisés sont :

* L'apprentissage Supervisée : consiste en l'introduction de plusieurs données (attributs et étiquettes) déjà définies et étudiés par un expert; pour la première phase, le réseau apprend ces échantillons en minimisant l'erreur entre la sortie désirée et celle obtenue, vient ensuite la deuxième phase où le réseau prédit la sortie selon ce qu'il a appris auparavant.

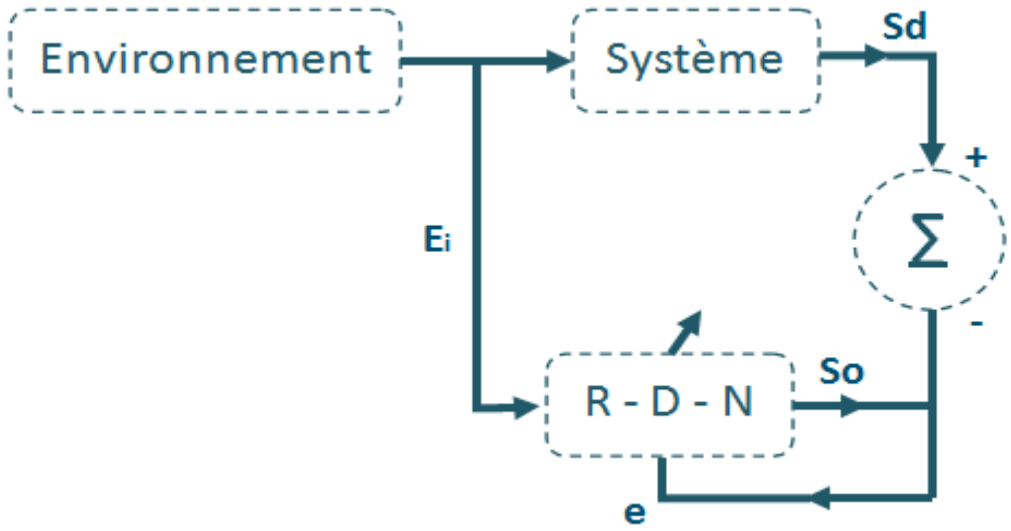


Figure I.4 : Schéma générale de l'apprentissage supervisé

où :

E_i : Représente l'entrée i du neurone.

e : Représente l'erreur.

Sd : Sortie désirée du neurone.

So : Sortie obtenue du neurone.

* L'apprentissage Non - Supervisée : Comme son nom l'indique, il est l'opposé du premier. Dans cet apprentissage, l'algorithme ne possède pas d'étiquettes, il va essayer de les deviner : il est autodidacte. Il va procéder par regroupement en cherchant les données similaires. Il est utilisé lorsque on ne sait ce que qu'on cherche, Il est aussi appelé « Clustering », qui signifie « groupement ».

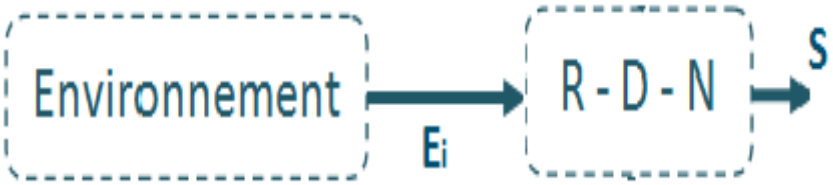


Figure I.5 : Schéma générale de l'apprentissage non-supervisé

$\mathbf{E_i}$: Représente l'entrée i du neurone.

S : Sortie du neurone.

* L'apprentissage Semi - Supervisée : Il est mis en œuvre quand des données (ou "étiquettes") manquent. Le modèle doit utiliser des exemples non étiquetés pour pouvoir se renseigner.

II-5-c) - Problème de Sur-Apprentissage :

Le Sur-apprentissage survient dans l'apprentissage supervisée lorsque l'ajustement des poids et la minimisation de l'erreur totale se font de manière très répétés, ceci est dû généralement soit, au grand nombre d'échantillons à apprendre, ou bien au grand nombre de couches cachés; en d'autres termes, le réseau devient très performant sur les exemples utilisés pour l'apprentissage, mais parvient pas à généraliser pour des informations quelconques.

Donc pour limiter ce genre de problèmes, on doit veiller à utiliser un nombre adéquat de neurones et de couches cachées et même à limiter la base d'apprentissage.

II-6) - Architecture des Réseaux de neurones :

Après avoir vu que l'on pouvait classifier les Réseaux de neurones selon le mode d'apprentissage, on verra maintenant la classification par architecture qui elle aussi se subdivise en deux grandes familles : Les réseaux Feed-forward et les réseaux Feed-back.

II-6-a) - Les réseaux Feed-forward : se sont des réseaux dans lesquels les informations se propagent successivement de couche en couche sans retour en arrière. On y trouve :

* II-6-a-1) - Le perceptron monocouche : c'est le réseau le plus simple en vue de l'architecture et au niveau de sa mise en œuvre, il se compose d'une couche d'entrée et une couche de sortie. Ce réseau est capable de résoudre des problèmes linéairement séparables (ex : fonction logique 'OU' ou 'AND'); à noter que tout les neurones de la couche d'entrée sont liés à tout les neurones de sortie, comme le montre la figure suivante :

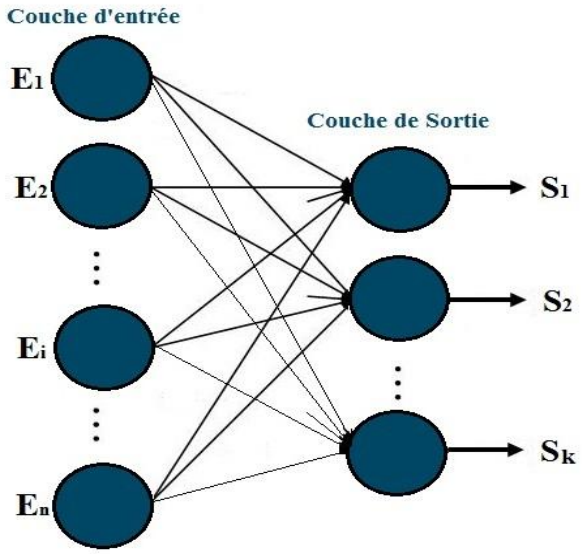


Figure I.6 : Perceptron monocouche

E_1 : Entrée 1 de la couche d'entrée.

E_i : Entrée i de la couche d'entrée.

E_n : Entrée n de la couche d'entrée.

S_1 : Première sortie du réseau.

S_k : Sortie k du réseau (k étant le nombre de sortie).

* II-6-a-1) - Le perceptron multicouche (PMC): contient des couches cachées entre la couche d'entrée et celle de sortie, ce réseau résout des problèmes non-linéairement séparables (ex : fonction logique 'XOR'); et là aussi, chaque neurone des couches cachées est connecté à tout les neurones de la couche qui la précède et celle qui la suit. La figure ci-dessous représente un PMC :

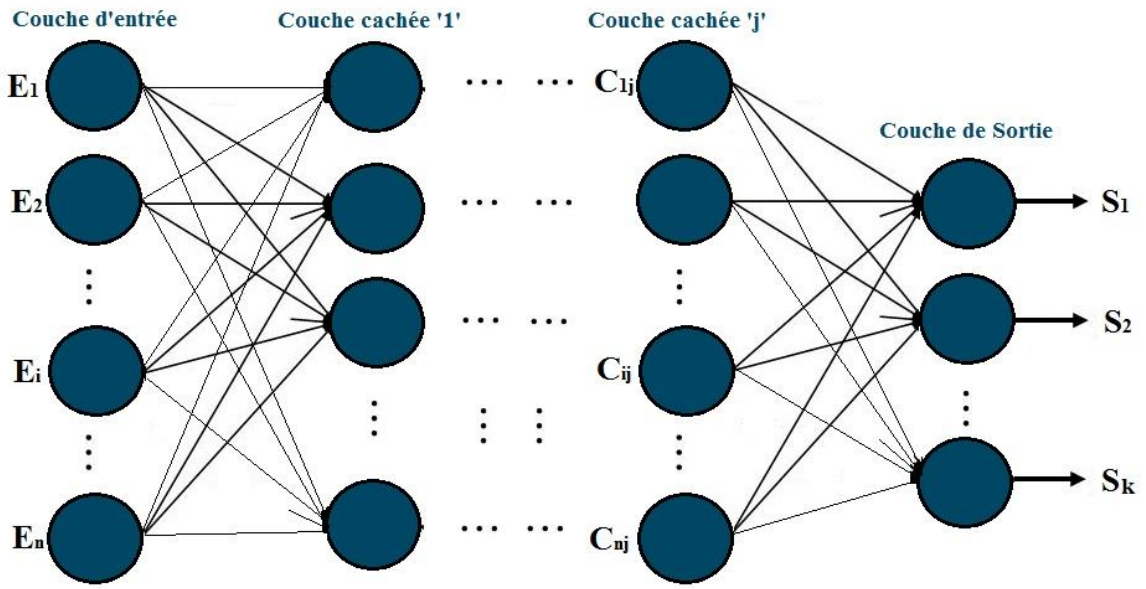


Figure I.7 : Perceptron multicouche

E_1 : Entrée 1 de la couche d'entrée.

E_i : Entrée i de la couche d'entrée.

E_n : Dernière entrée de la couche d'entrée.

C_{1j} : Premier neurone de la couche cachée j.

C_{ij} : Neurone i de la couche cachée j.

C_{nj} : Dernier neurone de la couche cachée j (m: nombre de neurones dans la couche cachée j).

S_1 : Première sortie du réseau.

S_k : Sortie k du réseau (k étant le nombre de sortie).

II-6-a-1) - Le réseau à fonctions radiales (RBF): possède la même architecture que le PMC, la différence réside au niveau de la fonction d'activation des neurones, car les RBF utilisent les fonctions gaussiennes.

II-6-b) - Les réseaux Feed-back : appelés aussi "Réseaux récurrents", sont des réseaux dans lesquels le retour en arrière est possible. On y trouve :

* Cartes auto-organisatrices de Kohonen : C'est un réseau à apprentissage non-supervisée, il est sous forme d'une grille dont chaque nœud représente un neurone associé à un vecteur de poids[10]; la figure ci-dessous explique mieux son architecture :

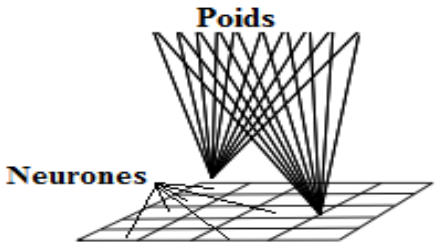


Figure I.8 : Architecture d'un réseau de Kohonen

* Réseaux de Hopfield: Se sont des réseaux entièrement connectés. Dans ce type de réseau, chaque neurone est connecté à chaque autre neurone et il n'y a aucune différenciation entre les neurones d'entrée et de sortie. Ils sont capables de trouver un objet stocké en fonction de représentations partielles ou bruitées. L'application principale des réseaux de Hopfield est l'entrepôt de connaissances mais aussi la résolution de problèmes d'optimisation. Le mode d'apprentissage utilisé est là aussi le mode non-supervisé.

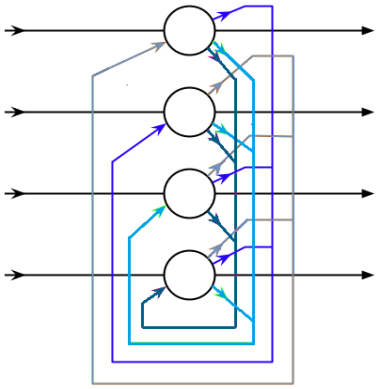


Figure I.9 : Architecture d'un Réseau de Hopfield

* II-6-b-3) - Réseaux ART (Adaptative Resonnance Theory) : Le réseau ART est formé d'une couche d'entrée qui est aussi la couche de sortie et d'une couche cachée. Le terme de couche cachée est emprunté au réseau multicouche. Le problème majeur qui se pose dans ce type de réseaux est le dilemme « stabilité/plasticité ».

En effet, la seule possibilité, pour assurer la stabilité, serait que le coefficient d'apprentissage tende vers zéro, mais le réseau perdrait alors sa plasticité. Les ART ont été conçus spécifiquement pour contourner ce problème. Dans ce genre de réseau, les vecteurs de poids ne seront adaptés que si l'entrée fournie est suffisamment proche, d'un prototype déjà connu par le réseau. On parlera alors de résonnance. Le mode d'apprentissage des ART peut être supervisé ou non.

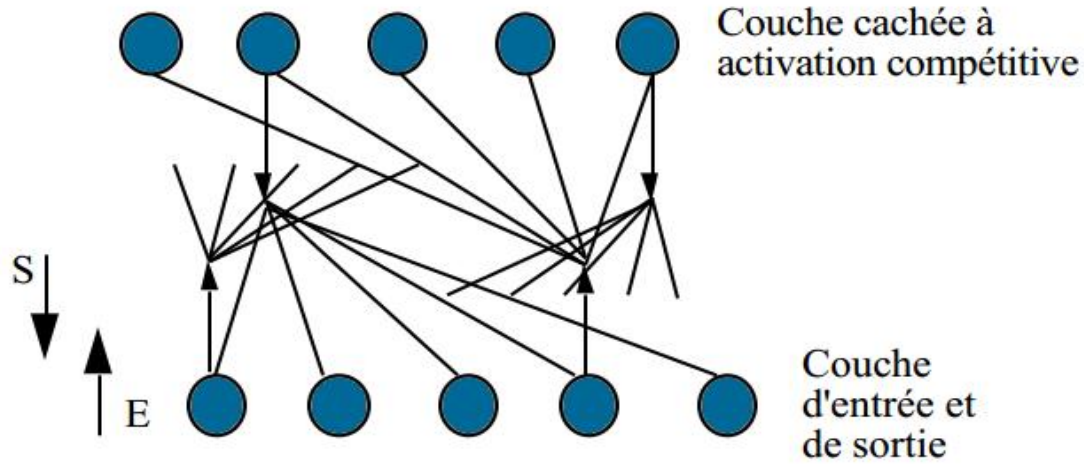


Figure I.10 : Réseau ART.1

II-7) - Avantages et inconvénients des réseaux de neurones artificiels:

On dénombre plusieurs avantages du réseau de neurone artificiel : sa capacité à apprendre, à modéliser les systèmes, de plus ils sont adaptés au traitement des données non linéaires, et prennent compte de variables qualitatives en attribuant des données binaire, mais aussi la possibilité de travailler avec des données bruités ou incomplètes, et bien d'autres encore. Cependant il présente quelques inconvénients telle que l'apparition du problème de sur-apprentissage, l'apprentissage n'est pas toujours évident, la non interprétation des poids ou encore l'indétermination directe du nombre de couches cachées et des neurones.

II-8) - Mise en œuvre des R.N.A:

La mise en œuvre des R.N.A est tributaire de l'objectif fixé, savoir le nombre d'entrée et de sortie est primordial avant toute manœuvre, vient ensuite le problème que redoute tout les utilisateurs, à savoir la détermination du nombre de couches cachées et du nombre de neurones dans chacune d'entre elles, car au jour d'aujourd'hui, il n'y'a pas de règle générale pour en choisir, seuls quelques hypothèses et approches ont été conçues, et parmi elles on trouve la méthode Shepard (1990), de Venugopal et Baets (1994), Mendelson (2001), on developpera ici comme exemple la démarche appliquée par Wierenga et Kluytmans en 1994 qui ont divisé leur méthode en 04 étapes :

* 1ère étape : consiste en la détermination du nombre de couches cachée :

  - ✓ 1 couche cachée permet d'approximer une fonction continu.
  - ✓ 2 couche cachée prend en compte les discontinuité.

* 2ème étape : déterminer la taille de la couche cachée doit être, c.à.d. choisir le nombre de neurones contenus dans la couche cachée :

  - ✓ D'après Wierenga et Kluytmans le nombre neurones doit être égale à celle de la couche d'entrée .
  - ✓ D'après Venugopal et Baets, il faut qu'il soit égale à 75% de celle-ci.
  - ✓ Shepard prétend que ce nombre est égale à la racine carrée du produit du nombre de neurones dans la couche d'entrée et de sortie.

Mais en réalité, les analystes ne sont nullement convaincus, et estiment qu'il est préférable d'essayer le plus de taille possible, c.à.d. changer la taille régulièrement jusqu'à l'obtention de la meilleure performance.

* 3ème étape : choisir la fonction d'activation des neurones contenues dans le réseau.
* 4ème et dernière étape : choisir la fonction d'apprentissage qui s'adapte mieux au système étudié : Levenberg-Marquardt (LM), Resilient Backpropagation (RP), Variable Learning Rate Backpropagation ... etc.

Conclusion :

A travers ce chapitre, on a étudié les Réseaux de neurones d'une manière globale, et on a montré leur origine, leurs différents modes et architectures dont ils disposent, on a aussi vu leur classification selon l'architecture ou le mode d'apprentissage, ainsi que leurs avantages et inconvénients. Dans le chapitre suivant, on étudiera 03 types d'implémentations des réseaux de neurones artificiels, dans les circuits FPGA, dans le logiciel MATLAB et surtout dans un microcontrôleur.

Chapitre II:

L'Implémentation des Réseaux de Neurones

Introduction :

Dans ce deuxième chapitre, on parlera de "l'implémentation" ou d'un autre sens de "l'intégration" des algorithmes dans différents supports matérielles; on a choisi de trois supports distincts, à chacun d'eux son mode de fonctionnement, ses caractéristiques, sa méthode à produire un résultat et bien évidemment à chacun son implémentation.

On commencera par introduire quelques définition utiles à l'implémentation, on s'attaquera ensuite directement à la description, et à la présentation de ces supports en question, et on indiquera la procédure à suivre pour intégrer un algorithme quelconque d'un RDN.

I) - Définitions :

I-1) - L'implémentation :

Plusieurs synonymes ont été présentées pour décrire le terme d'implémentation, les plus proches et surtout, les plus justes sont : l'adaptation, l'exécution, la fabrication, l'intégration, et la réalisation. On peut donc définir l'implémentation comme étant la mise en œuvre d'un outil informatique à partir de documents. Implémenter un ANN peut être interprétée par : écrire, développer et intégrer un algorithme de réseau de neurones dans un outil informatique.

La schéma ci-dessous montre un exemple d'une implémentation d'un réseau de neurones:

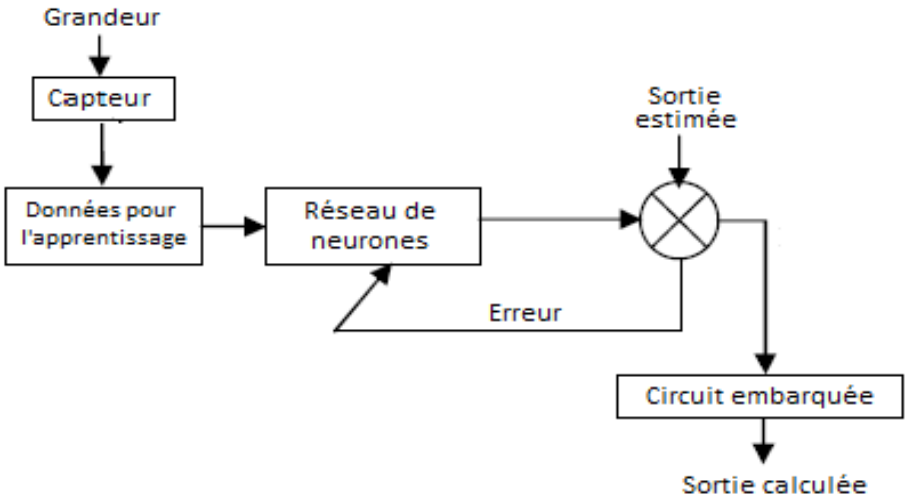


Figure II.1 : Schéma générale d'une implémentation d'un ANN**.**

I-2) - Le calcul séquentiel :

Consiste en l'exécution des tâches l'une après l'autre, d'une façon séquentielle (en série), en plus d'être simple à réaliser, il a le grand avantage qu'il ne nécessite pas beaucoup de ressources, et l'inconvénient majeur est la lenteur du traitement des données. Il est beaucoup plus utilisé dans le domaines des processeurs et ses dérivés : microcontrôleurs, DSP...etc.

En réseau de neurones on utilisera un accumulateur, le principe consiste à multiplier les entrées et leurs poids respectifs ω_i x E_i, ensuite à additionner le résultat obtenu pour être accumulé avec l'ancienne valeur déjà accumulée et ainsi de suite jusqu'à l'obtention de la somme finale, cette dernière sera envoyée à la fonction de transfert déjà choisie. La figure suivante expose les blocs principaux de l'architecture :

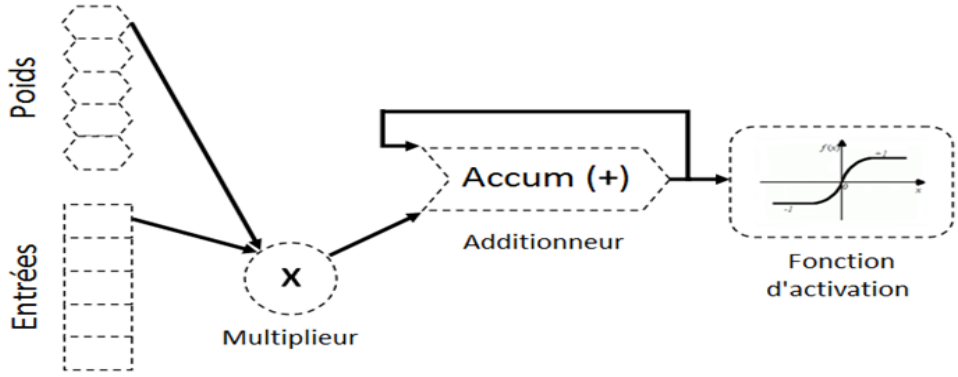


Figure II.2 : Modèle d'un calcul série.

I.3 - Le calcul parallèle :

Consiste en l'exécution simultanée d'une même tâche, partitionnée et adaptée afin de pouvoir être répartie entre plusieurs processeurs en vue de traiter plus rapidement des problèmes plus grands, il est utilisée pour les composants programmables : PAL, GAL, FPGA...etc. En pratique, on peut avoir deux types de calcul parallèle :

I-2-a) - La calcul parallèle partiel : ou Partial Parallel Processing (PPP) en anglais, cette technique utilise autant de multiplieurs que de nombre de paire du produit ω_i x E_i suivis par des additionneurs a deux entrées comme le montre la figure suivante :

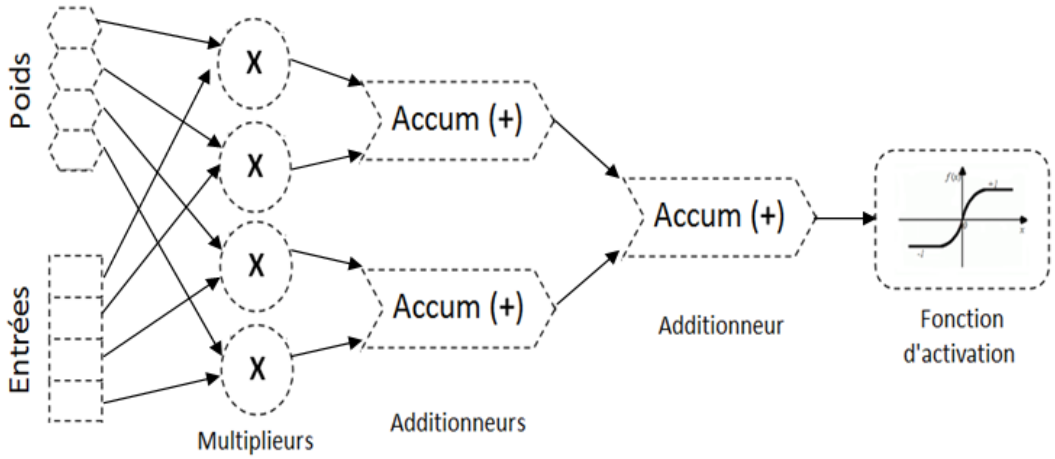


Figure II.3 : Modèle d'un calcul PPP.

I-2-b) - Le calcul parallèle complet : ou Full Parallel Processing (FPP) en anglais, cette technique utilise le même nombre de multiplieurs, comme dans le model PPP, la différence est l'utilisation d'un seul et unique additionneur qui a autant d'entées que le nombre de produits ω_i x E_i . La sortie de cet additionneur sera connectée à l'entée de la fonction de transfert comme le montre la figure suivante :

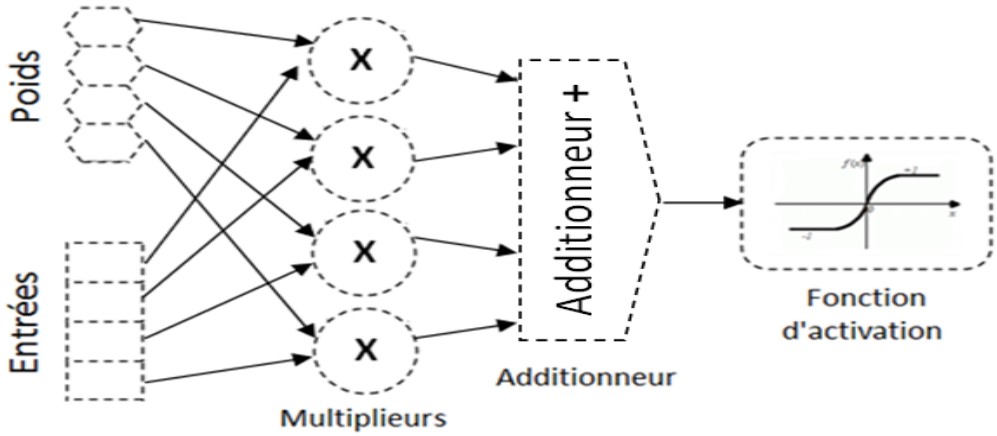


Figure II.4 : Modèle d'un calcul FPP.

Chaque technique présente des avantages et des inconvénients. Ce qui fait que le choix de la technique à utiliser dépend des ressources disponibles, de l'utilisation de notre réseau et le besoin de parallélisme.

II) - Les Réseaux de neurones et les circuits FPGA :

II-1) - Description des circuits FPGA :

Les FPGA (Field Programmable Gate Arrays) ou "Circuits logiques programmables" sont des composants VLSI (Very Large Scale Integration) entièrement reconfigurables, ce qui permet de les reprogrammer à maintes fois afin d'accélérer certaines phases de calculs. Leur architecture diffère d'un fabricant à un autre, chacun d'eux propose des tailles variable, de 100.000 à 10.000.000 portes logiques. Quelque soit la technologie utilisée, réellement aucune porte logique n'est implantée. Il s'agit de blocs logiques programmables, mais versatiles (RAM), et de plusieurs connexions programmables, on évoquera dans cette partie la procédure à suivre pour implémenter un R-D-N dans un composant FPGA [28] de la société XILINX.

Les principaux avantages des circuits FPGA sont : ils disposent d'une très haute densité, une grande vitesse (100 MHz à quelques GHz) et aussi un très grand nombre d'entrées/sorties, par contre les inconvénients sont : prix élevé (mais en baisse), alimentation difficile (plusieurs tensions, courants élevés, connexions multiples), et surtout, ils possèdent des mémoires vives volatiles (cellules RAM).

II-2) - Structure d'un FPGA - Xilinx :

Elle se présente sous forme de deux couches : une couche circuit configurable et un réseau de mémoire SRAM. La structure d'un FPGA est donnée dans la figure suivante :

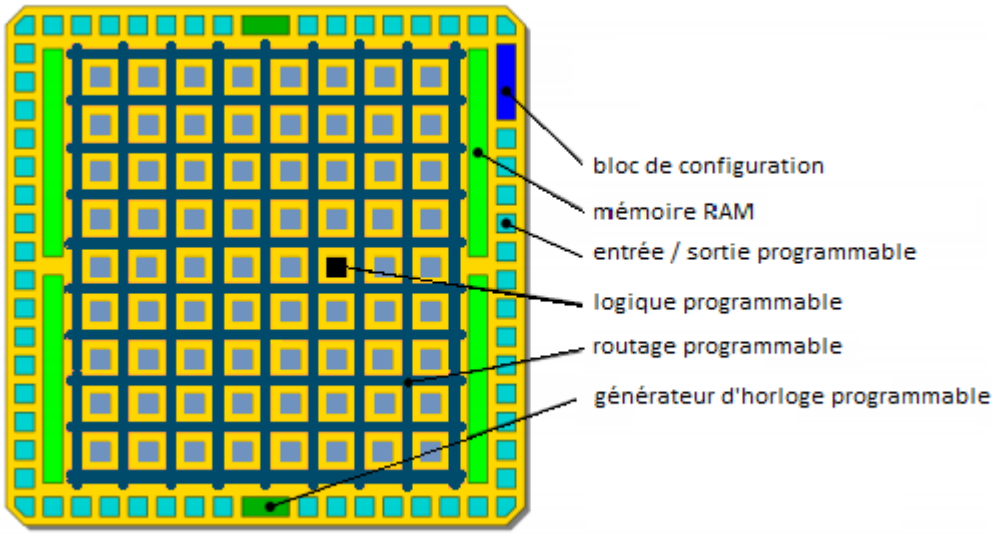


Figure II.5 : Structure d'un FPGA.

Ce circuit ne peut évidemment pas être vu de la sorte, les fonctions logiques n'occupant qu'environ 5% du circuit.

II-3) - Configuration (programmation) des FPGA :

III-3-a) - Définition:

Le langage approprié aux FPGA est le langage VHDL [29] qui signifie VHSIC Hardware Description Langage (VHSIC : Very High Speed Integrated Circuit), il est écrit dans les années 70 pour réaliser la simulation de circuits électroniques, il est indépendant des technologies utilisées.

III-3-b) - Structure d'une description :

Une description VHDL est composée de 2 parties indissociables à savoir l'entité et l'architecture et une partie librairies :

* La partie librairie : réservée pour l'introduction des librairies utilisées.
* L'entité (ENTITY) : elle définit les entrées et sorties.
* L'architecture (ARCHITECTURE) : elle contient les instructions VHDL permettant de réaliser le fonctionnement attendu.

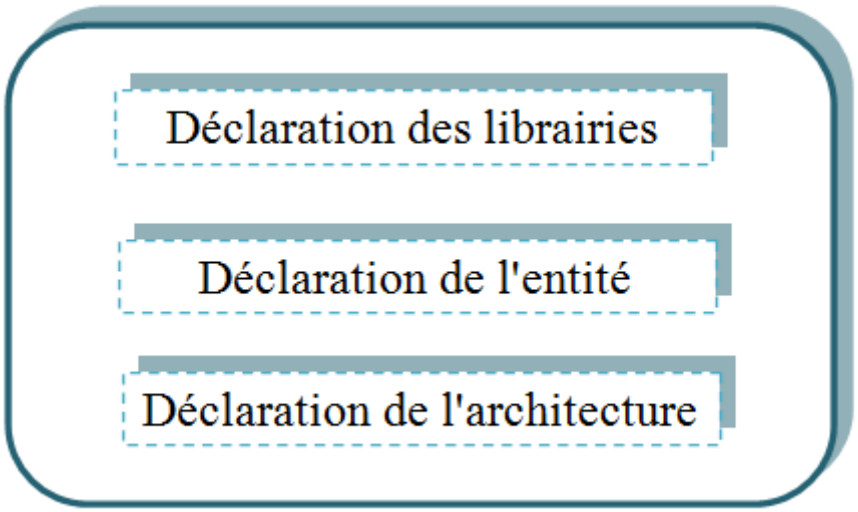


Figure II.6 : Structure d'un description

Exemple d'une description VHDL:

Cette exemple traite le problème de la porte AND à 3 entrées ('E1', 'E2', 'E3'), le résultat sera stocké dans ('S') :

1 / On écrira d'abord la partie entité (déclarer les entrées et les sorties)

2 / On abordera ensuite la partie architecture (partie traitement)

code :

```
entity AND_3 is
  port (
    E1 : in bit;
    E2 : in bit;
    E3 : in bit;
    S  : out bit
      ) ;
end entity;
architecture Sor of AND_3 is
  begin
    S < = E1 and E2 and E3;
  end Sor;
```

II-4) - Implémentation d'un R-D-N dans un FPGA :

II-4-a) - Le logiciel XILINX - ISE :

Ce logiciel est développé par la société Xilinx pour la programmation des composants FPGA de la firme Xilinx. Il est disponible pour les systèmes d'exploitation Windows (XP et Seven 7), et Linux. Il est très simple à utiliser et répond parfaitement aux besoins des programmeurs. Ce logiciel est téléchargeable gratuitement à partir du site officiel de Xilinx

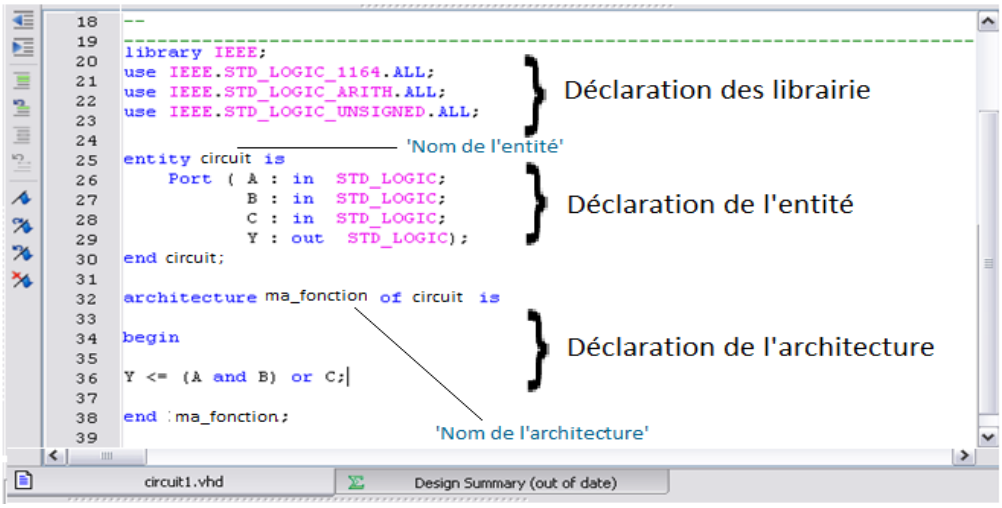


Figure II.7 : Interface du logiciel Xilinx ISE.

II-4-b) - Programmation sur le FPGA :

Les étapes de l'implémentation sur un FPGA sous ISE sont :

* création d'un projet.
* Ecriture du programme, ensuite l'enregistrement.
* Vérification de la syntaxe.
* Correction d'éventuelles erreurs.
* Création des contraintes temporelles du composant à régler selon le composant utilisé et la fréquence.
* Assignement des pins.
* Simulation du module après création d'un module testbench.
* Vérification par ISE des contraintes temporelles.

II-4-c) - L'implémentation :

Elle est très simple à réaliser, tout dépendra de la maîtrise du langage VHDL, on s'inspirera de [30], l'implémentation sera faite, selon la fonction d'activation choisie : à seuil, linéaire, et sigmoïde. Notre travail n'étant pas basé sur la programmation des FPGA, on ne donnera que la procédure à suivre pour réaliser une implémentation d'un R-D-N sur la carte. Tout d'abord et pour faciliter la tâche, on divisera la structure générale en cinq blocs indépendants et arrivant à la programmation on assemblera évidemment le tout par un seul et unique programme qui assurera la fonction du R-D-N, les cinq blocs en question sont :

* L'Accumulateur : Il est forme de deux composants, un additionneur et un registre.
* Le Multiplieur : Il fait le produit entre les valeurs des poids ω_i et les valeurs des entrées E_i pour envoyer le résultat a l'entrée de l'accumulateur.
* La fonction de transfert : (vu au chapitre I) Elle est très importante dans le faite que la sortie dépendra beaucoup d'elle, la plus simple étant la fonction seuil qui délivre soit un '0' ou un '1'.

On peut également utiliser la fonction linéaire, et là on n'en a qu'a essayer d'introduire la fonction rampe : $f(x) = x$, jusqu'ici tout est abordable mais l'ambigüité apparaît lors du désir d'implanter la fonction sigmoïde qui a pour relation : $f(x) = \frac{1}{1+e^{-x}}$; afin de l'implanter, on se doit d'effectuer une approximation linéaire de la fonction sigmoïde. Plusieurs méthodes

existent, parmi elles on trouve celle de Piece Wise Linear Approximation (PWL) [31]. On peut choisir f(x) ainsi :

$$f(x) = \begin{cases} 0 & \text{si} \quad x < -4 \\ 0.0625\,x + 0.25 & \text{si} \quad -4 \leq x < 0 \\ 0.5 & \text{si} \quad x = 0 \\ 0.0625\,x + 0.75 & \text{si} \quad 0 < x \leq 4 \\ 1 & \text{si} \quad x > 4 \end{cases} \qquad \textbf{II.1}$$

L'ensemble de ces trois composants constitue le neurone, sans les valeurs des poids et des entrées.

* Les ROMs : Elles contiennent les valeurs des poids ω_i et des entrées E_i.
* Le Multiplexeur : Il sert pour sélectionner les $\omega_i$et les E_i correspondants.

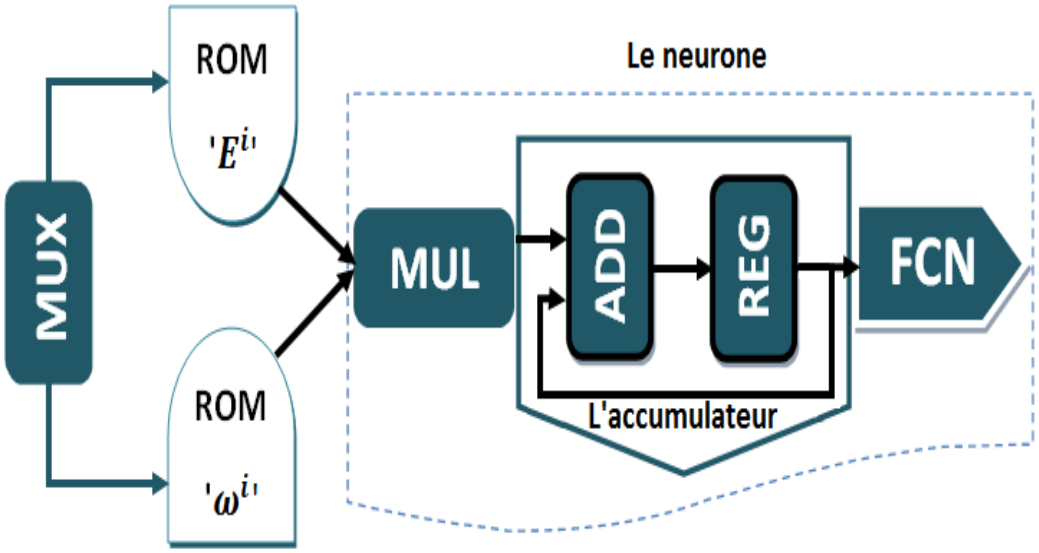


Figure II.8 : Organigramme d'une implémentation d'un ANN sur un FPGA.

où :

MUX : représente le multiplexeur.

MUL : représente le multiplieur.

ADD : représente l'additionneur.

REG : représente le registre.

FCN : représente la fonction d'activation.

III) - Les Réseaux de neurones et les MATLAB :

III-1) - Description de MATLAB :

MATLAB est un langage de programmation utilisé à des fins de calcul numérique, développé par la société The MathWorks. MATLAB permet d'effectuer des opérations sur des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes. C'est le logiciel le plus utilisé dans le monde de l'ingénierie et la science. Il touche plusieurs domaines tels que les systèmes automobiles, les dispositifs de surveillance de la santé, les réseaux électriques intelligents, l'apprentissage automatique, la robotique, le traitement du signal, le traitement d'images, les systèmes de communications, les finances, et bien plus encore.

MATLAB peut s'utiliser tout seul ou avec des boites à outils (généralement connues sous le nom de 'toolbox') et à chaque domaine est associé une ou plusieurs boites à outils spécifiques, par exemple dans le domaine des Mathématiques, Statistiques, et d'Optimisation on trouve la : Partial Differential Equation Toolbox (boite à outils pour la résolution des équations différentiels), ou la : Curve Fitting Toolbox (boite à outils pour l'ajustement des courbe), ou encore la : Neural Network Toolbox (boite à outils pour l'utilisation des réseaux de neurones) qu'on aura l'opportunité d'expliquer ultérieurement.

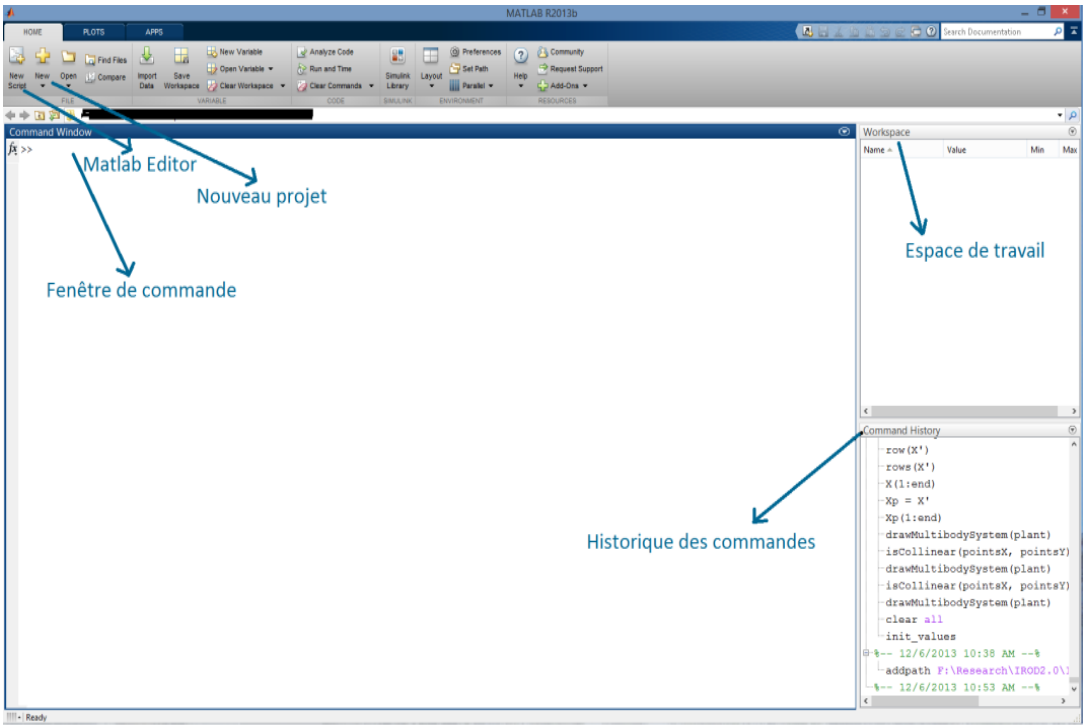


Figure II.9 : Interface MATLAB principale.

Nouveau projet (New) : Comme son l'indique, ce bouton sert a ouvrir une nouvelle fenêtre de Command Window (ouverture d'un nouveau projet).

Fenêtre de commande (Command Window) : C'est là que sont saisîtes toute les instructions de notre programme, dans cette zone, les instructions sont écrites et exécutés une après l'autre.

Editeur (MATLAB Editor) : Contrairement à Command Window, ici les instructions sont écrites à la fois, une fois fini, le programme exécute les instructions sous la forme d'une fonction.

Espace de travail (Workspace) : Dans cette zone, on verra toute les variables et les constantes qu'on utilise actuellement dans le Command Window, leurs nom, leurs format, leurs taille, et leurs types y seront aussi affichés.

Historique des commandes (Command History) : L'historique des commandes affiche un journal d'états qu'on a exécuté dans les sessions de MATLAB actuelles et/ou précédentes.

III-2) - Implémentation d'un R-D-N dans MATLAB:

Avant d'aller plus loin, on se doit de préciser que la simulation des Réseaux de neurones dans MATLAB peut se faire de deux principales méthodes, la première étant l'utilisation de la Neural Network (NN) toolbox et la seconde se fera étape par étape et ce, en programmant nous même l'algorithme dont on désire "Implémenter".

III-2-a) - La Neural Network Toolbox :

Avec cette toolbox défini précédemment, on a plus besoin de parler d'implémentation, tout est déjà prêt, on n'en a qu'a introduire nos paramètres et la toolbox se charge du reste, pour être plus claire concernant cette boite à outils, on détaillera ci-dessous un exemple inspiré de [32].

Classification des données linéairement séparables avec un perceptron :

Description du problème:

Deux groupes de données, appartenant à deux classes distinctes, sont définies dans un espace d'entrée en 2 dimensions. Les classes sont linéairement séparables. La tâche consiste à construire un Perceptron (défini au chapitre I) pour la classification des données.

Pour avoir une solution, on doit définir nos entrées et nos sorties, ensuite de créer la fonction Perceptron et le former (phase d'apprentissage); les figures suivantes sont prises de MATLAB EDITOR, elles représentent les étapes a suivre pour résoudre notre problème, on a

devisé le programme en trois sous-programmes afin d'interpréter au mieux la procédure a suivre quant à l'utilisation de la ANN toolbox.

* Définition des données d'entrée et de sortie

```
close all, clear all, clc, format compact

% nombre d'échantillons pour chaque classe
N = 20;
% définir les entrées et les sorties
offset = 5;  % l'offset est défini pour départager entre les 02 classes
x = [randn(2,N) randn(2,N)+offset];  % Les entrées
y = [zeros(1,N) ones(1,N)];          % Les sorties

% afficher la courbe
figure(1)
plotpv(x,y);
```

Figure II.10 : Programme MATLAB qui initialise aléatoirement les 2 classes et leurs sorties.

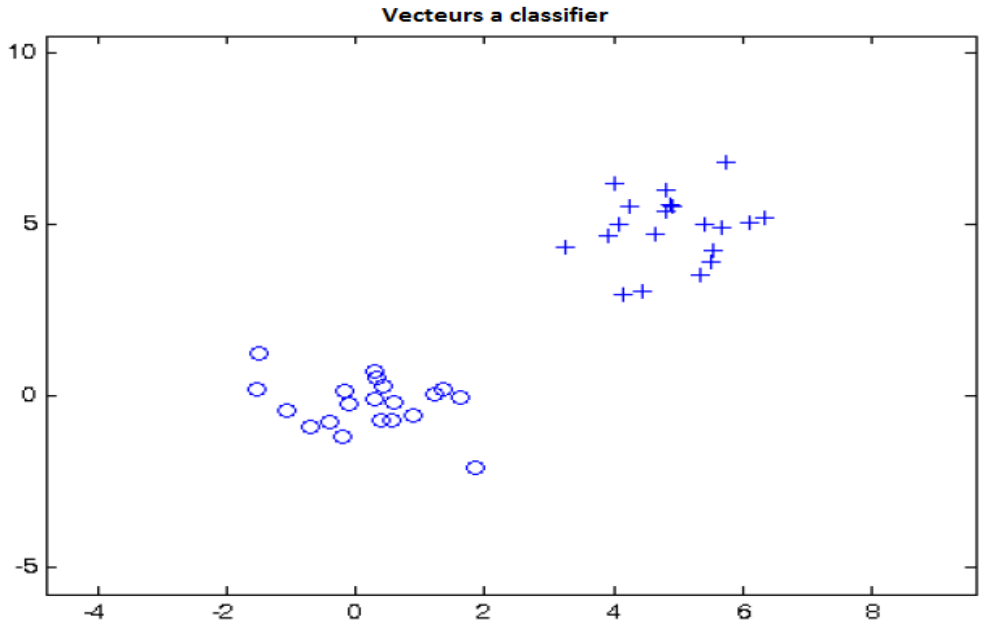


Figure II.11 : Affichage des données définis précédemment.

* Création et apprentissage du perceptron

```
net = perceptron;
net = train(net,x,y);
view(net);
```

Figure II.12 : Code MATLAB pour la création du Perceptron et son apprentissage.

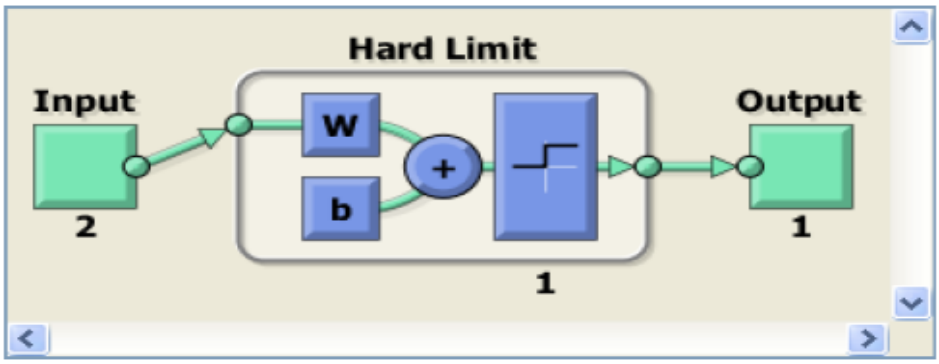


Figure II.13 : Schéma du modèle neuronal dans MATLAB.

* Affichage de la ligne séparatrice formée

```
figure(1)
plotpc(net.IW{1},net.b{1});
```

Figure II.14 : Code MATLAB pour l'affichage de la ligne séparant les 02 classes.

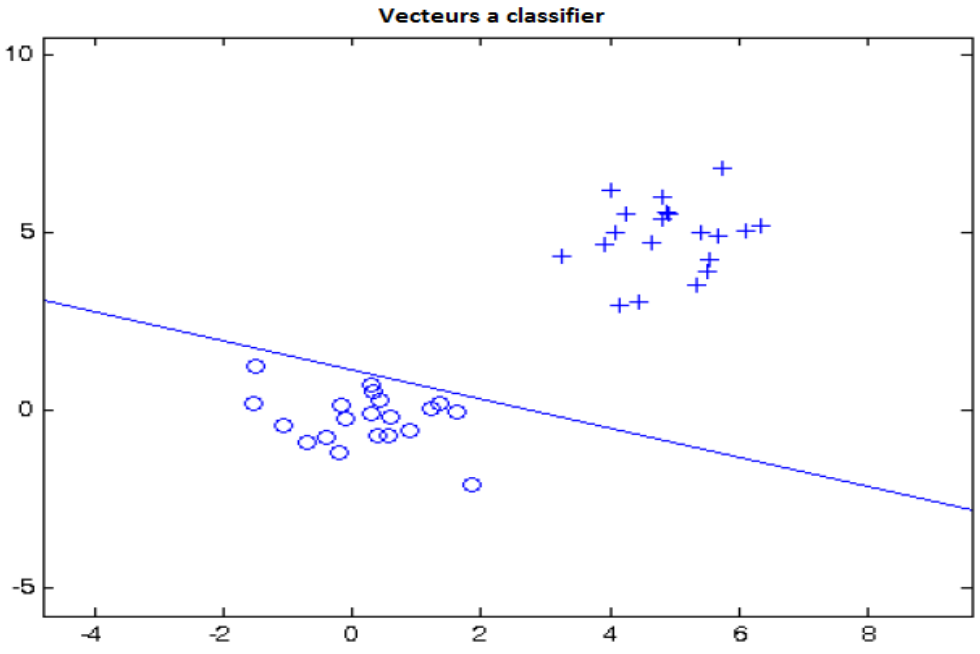


Figure II.15 : Les 02 classes séparées par MATLAB.

III-2-b) - Programmation manuelle du R-D-N :

Le principe ici est le même que le précédent, sauf que dans cette partie, on aura à écrire tout le programme nous même. On commencera par introduire nos paramètres principaux, à savoir : le nombre d'échantillons, le nombre d'entrées du réseau et leurs valeurs, le nombre de couche cachées, leurs poids respectives, et le nombre de sortie, par contre, la suite est bien plus compliquée; alors qu'avec la ANN toolbox, il suffisait seulement de taper et d'exécuter la

fonction train() pour avoir notre solution, là on est obligé de développer cette fonction manuellement en saisissant plusieurs lignes de codes tout en essayant de garder une bonne ossature de la forme du programme. Tout comme les FPGA, même avec MATLAB on peut procéder de deux façons différentes quant à l'implémentation des R-D-N, soit en calcul direct (série) ou en calcul parallèle. le grand avantage de MATLAB, c'est qu'il possède la " Parallel Computing Toolbox Functions " qui permet de résoudre des problèmes de calcul et de données à forte intensité en utilisant des processeurs multi-cœurs, l'exemple ci-dessous montre un simple calcul parallèle avec cette toolbox, mais ô combien importante pour assurer la rapidité de notre code.

Propagation de l'information des entrées au neurones d'une couche cachée :

Description du problème:

Un simple exemple étant le calcul de $\sum(E_i \times \omega_{ji})$ qui revient souvent, imaginons qu'on a deux neurones d'entrées et quatre neurones dans la couche cachée comme le montre la figure suivante :

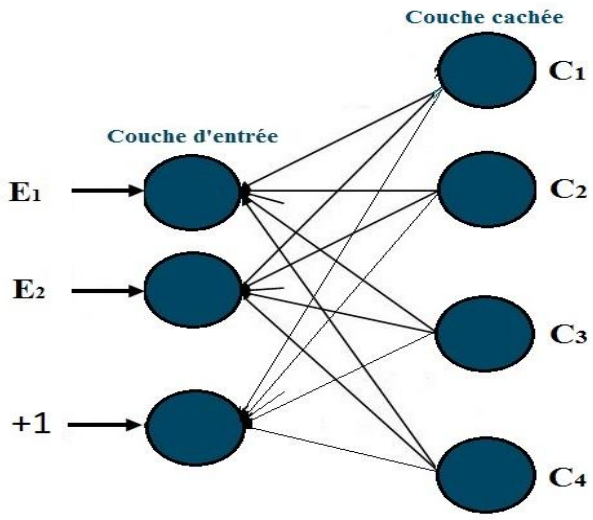


Figure II.16 : Propagation des entrées vers les couches cachées.

On aura donc a calculer quatre fois l'équation précédente ainsi :

$$C_1 = (E_1 \times \omega_{11}) + (E_2 \times \omega_{12}) + (\omega_{1b} \times 1)$$
$$C_2 = (E_1 \times \omega_{21}) + (E_2 \times \omega_{22}) + (\omega_{2b} \times 1)$$
$$C_3 = (E_1 \times \omega_{31}) + (E_2 \times \omega_{32}) + (\omega_{3b} \times 1)$$
$$C_4 = (E_1 \times \omega_{41}) + (E_2 \times \omega_{42}) + (\omega_{4b} \times 1) \qquad \textbf{II.2}$$

sachant que :

$\omega_{j,i}$: représente les poids des connexions entre les entrées et les neurones de la couche cachée.

i : la position des neurones d'entrées.

j : la position des neurones de la couche cachée.

b : Représente le biais.

En programmant, ce qui vient à esprit c'est d'écrire :

```
for  j = 1 :4
    Cj =  0 ;                    % initialiser à zéro pour éviter toute confusion
    for i = 1 : 3
        Cj =  Cj + ( Ei × ωji ) ;
    end
end
```

Cela calculerait en tout douze (12) produits et huit (08) sommes, ce calcul se fera l'un après l'autre, mais grâce a la toolbox du calcul parallèle, on pourra modifier notre programme en intégrant l'instruction de la boucle "parfor" au lieu de "for" pour que le calcul précédent se fera à la fois en raison de l'utilisation du processeur multi-cœurs. Plusieurs autres avantages existent pour la programmation MATLAB, mais on s'en contentera que des brefs descriptions précédentes vu que notre objectif se base sur l'implémentation sur des microcontrôleurs qu'on verra un peu plus tard.

On a montré à travers cet exemple qu'il est possible d'accélérer un code à l'aide des outils de calcul parallèle interactifs, tels que "parfor" et "parfeval" qui a pour instruction d'exécuter une fonction de manière asynchrone.

IV - Les Réseaux de neurones et les microcontrôleurs :

IV-1) - Description des microcontrôleurs :

IV-1-a) - Définition du microcontrôleur :

Un microcontrôleur est un circuit intégré rassemblant dans un même boitier un microprocesseur, plusieurs types de mémoires (RAM, ROM, Flash...etc) et des périphériques de communication (Entrées-Sorties).

IV-1-b) - Structure du microcontrôleur :

Un microprocesseur a besoin de certains éléments pour fonctionner :

* La mémoire morte dite ROM (principalement pour stocker le programme) ;
* La mémoire vive dite RAM (principalement pour stocker les variables) ;
* Des périphériques (principalement pour interagir avec le monde extérieur).
* Une horloge pour le cadencer (principalement à quartz).

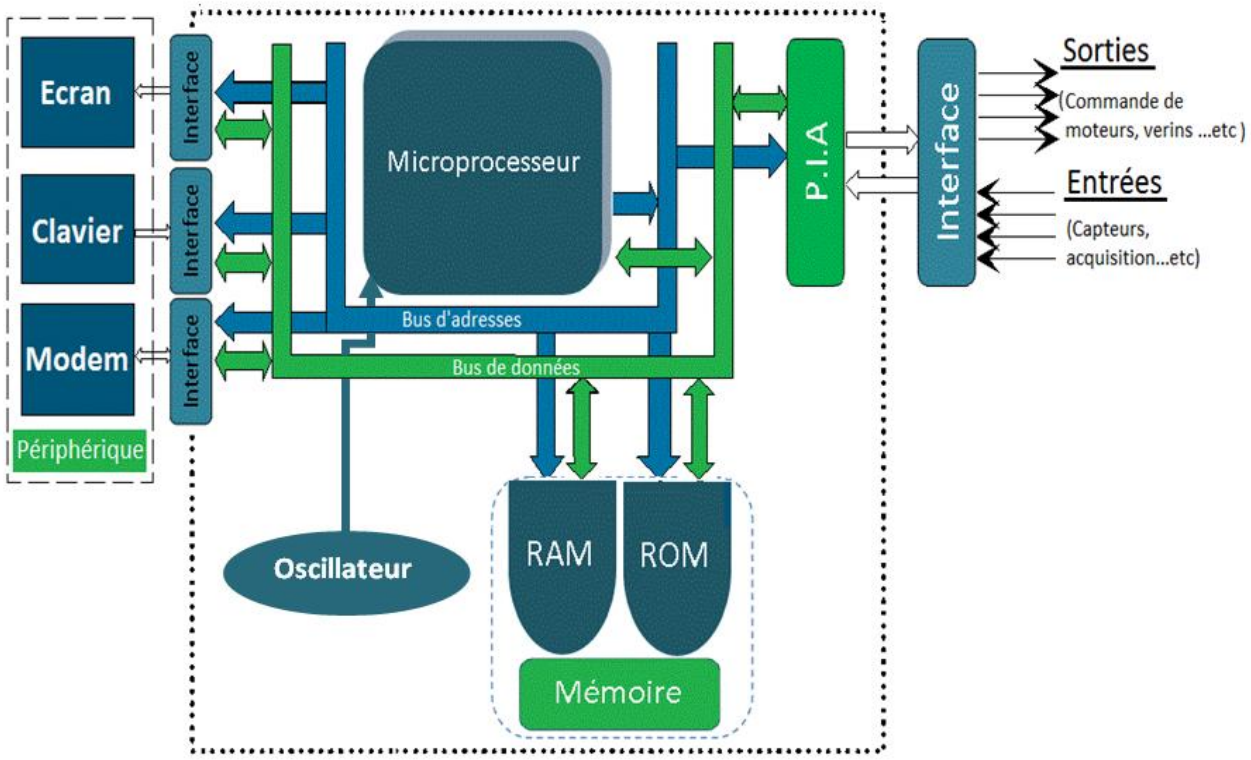


Figure II.17 : Schéma générale d'un système à microprocesseur

Ces différents blocs sont reliés par des bus qui sont des liaisons permettant de relier entre eux les différents organes du système. Concrètement, On distingue trois sortes de bus :

* Le bus de données : Permet de transférer les données, c'est à dire les informations traitées par le microprocesseur. Ces données sont de type binaire.
* Le Bus d'adresse : Permet d'établir la liaison proprement dite entre le microprocesseur et un composant. Chaque composant réagit dans un domaine d'adresse précise (exemple de notre vie quotidienne : lorsque quelqu'un nous téléphone, c'est notre appareil qui sonne et non pas celui de notre voisin).

* Le Bus de contrôle : Son rôle est d'informer le microprocesseur sur l'état du système et de le prévenir en cas de dysfonctionnement, il n'est pas représenté sur le schéma général précédent.

IV-1-c) - Fonctionnement générale du microcontrôleur :

À la mise en route du système, le microprocesseur va chercher dans la mémoire à l'adresse 0 (pour la plupart des processeurs) la première instruction à exécuter, il stocke cette dernière dans un registre interne appelé registre d'instructions, il l'exécute, puis en consultant le registre pointeur d'instruction, il va chercher l'instruction suivante, et ainsi de suite.

IV-2) - Description de Arduino :

II-2-a) - Définition:

La carte Arduino (ou son tout récent synonyme Genuino) est une carte à base de microcontrôleur d'architecture Atmel AVR, elle dispose de plusieurs broches numériques d'entrées/sorties, d'entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques), d'un quartz 16Mhz, des connexions.... etc, elle contient tout ce qui est nécessaire pour le fonctionnement du microcontrôleur; Pour pouvoir l'utiliser et se lancer, il suffit simplement de la connecter à un ordinateur à l'aide d'un câble USB.

II-2-b) - Fonctionnement des cartes Arduino:

Les différentes versions des Arduino fonctionnent sous le même principe général, on se basera sur les broches de la carte Arduino UNO de la figure ci-contre :

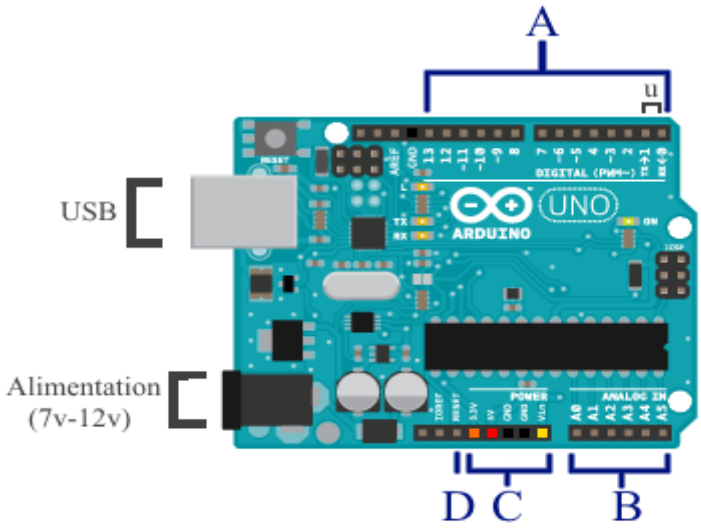


Figure IV.18 : La carte Arduino

* A : ce sont les broches dites numériques (0 ou 1) ou « tout ou rien » ; elles offrent en sortie du 5 V et acceptent en entrée du 5 V sur le même principe.
* B : ce sont les broches dites analogiques, valeur entre 0 V et 5 V.
* C : les différentes broches d'alimentation :
    - Rouge : sortie 5 V (+)
    - Orange : sortie 3,3 V (+)
    - Noire : les masses (−)
    - Jaune : entrée reliée à l'alimentation (7 V-12 V)

IV-2-c) - Types de cartes Arduino:

Il existe plusieurs types de cartes Arduino, les plus utilisées dans le monde des systèmes embarquées sont : la UNO, la LEONARDO, la DUE, la MEGA et sa petite amélioration la MEGA 2560, et la "Yún". Pour notre travail, on implémentera l'ANN sur la carte DUE. Le tableau ci-dessous décrit brièvement quelques importantes caractéristiques des cartes citées auparavant :

Arduino	Microcontrôleur	Flash ko	EEPROM ko	SRAM ko	Broches d'E/S numériques	Broches d'entrée analogique
Uno	ATmega328P	32	1	2	14	6
Leonardo	ATmega32U4	32	1	2,5	20	12
Mega	ATmega1280	128	4	8	54	16
Mega2560	ATmega2560	256	4	8	54	16
Due	Atmel SAM3X8E	512	0	96	54	12
Yun	ATmega32u4	32	1	2,5	20	12

Tableau II.1 : Quelques cartes Arduino et leurs caractéristiques

IV-2-d) - Programmation des cartes Arduino:

Les cartes Arduino peuvent être programmées avec le logiciel Arduino (à télécharger sur le site officiel). Il faut savoir que le langage utilisé par le logiciel Arduino pour programmer le microcontrôleur est basé sur les langages C et C++, il est très simple a écrire, désormais et grâce a l'ascension fulgurante d'Arduino plusieurs initiations et documentations sont accessibles via le net parmi elles.

Maintenant qu'on a défini les microcontrôleurs et surtout les cartes Arduino, il ne reste qu'a décrire la procédure d'implémentation des R-D-N dans les cartes susmentionnées.

IV-3) - Implémentation du R-D-N dans la carte Arduino :

Comme dans les précédentes parties où on a montré la possibilité d'implémenter un ANN que ce soit dans un FPGA ou MATLAB de deux façons différentes, avec les microcontrôleurs on peut aussi programmer soit séquentiellement ou parallèlement, et là aussi, tout dépendra de la maîtrise du langage Arduino. Notre travail est beaucoup plus similaire à la partie **III.2** qu'à la partie **II.4**, on procédera par un calcul série.

La propagation des informations des entrées de notre réseau vers la/les sortie(s) se base toujours sur le calcul de : $\sum_{i=1}^{n} \omega_i E_i$. Quand le signal arrive à la sortie on calculera l'erreur commise par notre réseau, vient ensuite l'étape de mise à jour des poids selon l'algorithme de rétropropagation choisi, on détaillera lors de la prochaine partie deux d'entre eux à savoir : l'algorithme de Rétropropagation par descente du gradient et l'algorithme de Levenberg-Marquardt.

IV-3-a) - Le calcul série avec Arduino:

Le calcul est très simple, on calculera le produit des entrées E_i et des poids ω_i, à chaque incrémentation du vecteur entrées, le produit se répétera et sera additionné à l'ancienne valeur et ainsi de suite, une fois le calcul achevé, la somme sera envoyé a la fonction de transfert afin d'avoir notre sortie, c'est exactement la même procédure que dans **Figure II.2**, tout le calcul se fera dans la SRAM, aucun élément externe n'est requis, ni multiplieur, ni additionneur, tout est programmable sur le logiciel d'Arduino.

IV-3-b) - Le calcul parallèle avec Arduino:

Le calcul ici est plus délicat qu'avec le calcul série, on aura besoin d'utiliser une bibliothèque externe "thread" qu'il faudra télécharger sur le net. C'est le même principe que la "Parallel Computing Toolbox Functions", le terme "thread" veut dire qu'il va y avoir deux instances en train d'interpréter le même programme au sein du même processus, certes la conception du code est difficile, mais question performance c'est l'idéal quant' à la rapidité du programme.

Conclusion :

On a vu a travers ce chapitre trois différentes méthodes d'implémentation d'un réseau neuronal que ce soit dans un circuit FPGA, dans MATLAB, ou dans une carte Arduino, on a remarqué que chacun d'eux contient des avantages et des inconvénients, le choix sera difficile et dépendra du système qu'on voudra réaliser. Pour un choix de performance, on aura a choisir a coup sûr MATLAB vu qu'il utilise la mémoire vive de l'ordinateur qui n'est à comparer nullement avec d'autres. Pour un choix de qualité de service, on pourra bien choisir les circuits FPGA, surtout avec leur baisse de prix, et techniquement, la flexibilité dont ils font part permet aux scientifiques de profiter de leurs avantages. Et enfin pour un choix de souplesse, de qualité et de facilité de conception, on choisira une des cartes Arduino, en plus de l'aisance dans la programmation, ils proposent de nombreuses extensions matérielles pour une bonne configuration et réalisation de maintes projets, mais aussi elle sera mieux adaptée aux systèmes embarqués la comparant à MATLAB et la programmation ne sera nullement effacée comme pour les FPGA dont l'interruption de l'alimentation fait disparaître tout le programme.

Chapitre III :

Partie Pratique

Introduction :

Dans cette deuxième et dernière partie, on présentera deux applications d'implémentation des ANNs dans notre carte Arduino DUE, la première application est une classification de deux maladies du système urinaire, à savoir l'inflammation aiguë et la néphrite aiguë, qu'on définira au un peu plus tard, l'algorithme d'apprentissage qu'on utilisera est celui de la descente du gradient(DG) [33], tandis que la deuxième est une modélisation d'une résistance thermique CTN dont l'algorithme est celui de Levenberg-Marquardt (LM) [34].

I) - Classification par les ANNs pour le diagnostic de deux maladies du systèmes urinaire:

I-1) - Description de l'application:

Elle va traiter une classification des deux maladies citées ci-dessus ; ayant presque les mêmes symptômes, une association des deux pour une étude est donc envisageable. Pour notre ANNs, on prendra comme entrées six paramètres qui peuvent aisément diagnostiquer si un patient souffre soit d'une des deux maladies, soit les deux en même temps, ou bien s'il est tout simplement sain, ces paramètres sont : la température du patient, présence de nausées, douleurs lombaires, besoin continu d'uriner, douleurs lors de la miction et enfin brûleurs de l'uretère (plus des démangeaisons et/ou gonflement de l'urètre).

I-2) - L'inflammation aiguë :

Le diagnostic de l'infection des voies urinaires, également appelée "cystite", fait référence à une inflammation de la vessie. Le plus souvent, les bactéries sont en cause. Parmi les voies urinaires, les uretères sont les conduits urinaires qui transportent l'urine des bassinets rénaux vers la vessie ; l'urètre quant à lui, est le canal de sortie de la vessie, qui amène l'urine à l'extérieur. Lorsque les bactéries se multiplient de façon abondante au sein de ce conduit, il y'a une augmentation de la rétention d'urine, donc le temps de prolifération des bactéries. La cystite s'accompagne toujours d'une "urétrite", l'inflammation de l'urètre.

Différents facteurs favorisent la multiplication d'agents pathogènes au sein des voies urinaires, on y trouve:

- Grossesse et accouchement
- Chez l'homme, hypertrophie de la prostate
- Rétrécissement de l'urètre
- Diabètes de type 1 et de type 2 et goutte
- Mauvaise hygiène
- Faiblesse immunitaire chez les nourrissons, les enfants, les malades chroniques et les patients traités par certains médicaments.

Quant aux symptômes, les malades ressentirons des :

- Douleurs et brûlures à la miction (algurie).
- Difficultés à la miction (dysurie).
- Besoin fréquent d'uriner avec émission de petites quantités d'urine (pollakiurie).
- Urines sombres, éventuellement mêlées à du sang.
- Douleurs et crampes dans le bas ventre.

Si cette inflammation s'aggrave, de nouveaux symptômes pourraient faire surface, les plus fréquents sont :

- Présence de nausées.
- Fièvre élevée (généralement supérieur à 38°C).

Tout symptôme de cystite nécessite une consultation médicale comme l'infection peut s'étendre aux voies urinaires supérieures.

I-3) - La néphrite aiguë :

La néphrite, appelée également "pyélonéphrite" est une est une inflammation des reins, donc elle plus grave que la cystite. Elle désigne l'inflammation du bassinet (la cavité du rein collectant les urines) et du rein lui-même. Celle-ci résulte généralement d'une infection bactérienne de type Escherichia coli. Il peut s'agir d'une complication d'une cystite non traitée ou mal traitée qui conduit à la remontée des bactéries de la vessie vers les reins, et à leur prolifération à ce niveau.

La pyélonéphrite aiguë survient plus souvent chez la femme, et elle est encore plus fréquente chez la femme enceinte. Elle est aussi fréquente chez les enfants dont une malformation des uretères provoque un reflux de l'urine de la vessie vers les reins.

Une néphrite aiguë est diagnostiquée, lorsque ces symptômes apparaissent :

- Fièvre élevée et soudaine (dépasse les 40°C).
- Douleurs lombaires uni ou bilatérale (mal aux reins).
- Frissons et souvent présence de nausée, voire vomissement.
- Affaiblissement et altération importante de l'état général.
- Propagation des douleurs de l'ensemble de l'abdomen.

I-4) - L'algorithme de descente du gradient :

L'algorithme de descente du gradient (algorithme DG) s'applique lorsqu'on cherche le minimum d'une fonction déterministe, qui est dérivable et dont le minimum reste difficile a trouver avec un calcul direct. L'algorithme utilise la dérivée d'ordre 1 de la fonction d'erreur totale, son processus de formation est la convergence asymptotique, on suivra étape par étape l'évolution de l'algorithme de DG selon l'architecture générale des ANNs.

I-4-a) - La propagation :

La propagation du signal d'entrée vers les sorties passant par les couches cachées s'effectue étape par étape c.à.d. d'une couche à une autre, prenons l'exemple d'une architecture à une seule couche cachée, la première étape se fera suivant les relations suivantes :

$$R_j = \sum_{j=1}^{J} \sum_{i=1}^{I} \left(\mathrm{E}_i \times \omega_{ji} \right) \qquad \textbf{III. 1}$$

$$C_j = F(R_j) \qquad \textbf{III. 2}$$

où :

I : Représente le nombre d'entrées.

J : Représente le nombre des neurones dans la couche cachées.

E_i : Représente l'entrée $\boldsymbol{i}$ du réseau.

ω_{ji} : Représente le poids liant le neurone de la couche cachée ***j*** à l'entrée ***i***.

F : Représente la fonction d'activation.

C_j : Représente la sortie ***j*** de la couche cachée, et l'entrée ***j*** de la prochaine couche.

Dans notre étude, on a utilisé la fonction sigmoïde comme fonction d'activation, donc l'équation **III.2** pourra s'écrire ainsi :

$$C_j = \frac{1}{1 + e^{-R_j}} \qquad \textbf{III.3}$$

On répétera la même procédure et on utilisera les mêmes formules pour la propagation de la couche cachée vers les sorties :

$$R_k = \sum_{k=1}^{K} \sum_{j=1}^{J} (C_j \times \omega_{kj}) \qquad \textbf{III.4}$$

$$S_k = \frac{1}{1 + e^{-R_k}} \qquad \textbf{III.5}$$

où :

K : Représente le nombre de sorties.

C_j : Représente l'entrée ***j*** de la couche de sortie.

ω_{kj} : Représente le poids liant la sortie ***k*** au neurone de la couche cachée ***j***.

S_k : Représente la sortie ***k*** du réseau.

I-4-b) - L'erreur quadratique :

D'une manière globale, l'erreur quadratique moyenne M.S.E (Mean Square Error) peut se définir comme étant une grandeur permettant de comparer des estimateurs entre eux, c'est exactement le même cas pour les réseaux de neurones, les estimateurs à comparer seront la sortie obtenue par le réseau après le calcul de la propagation et la sortie attendue :

$$\boldsymbol{e} = \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{1}{2} (D_n - S_{kn})^2 \qquad \textbf{III.6}$$

où :

n : Représente l'indice des échantillons N.

S_{kn} : Représente la sortie obtenue par le réseau.

D_n : Représente la sortie désirée (la sortie attendue).

$\boldsymbol{e}$: Représente l'erreur commise par le réseau.

I-4-c) - La rétropropagation :

L'étape la plus importante consiste à minimiser l'erreur $\boldsymbol{e}$, on peut écrire l'équation **III.6** de cette façon :

$$\boldsymbol{e} = \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{1}{2} \left(D_n - \frac{1}{1 + e^{-(C_j \times \omega_{kj})}} \right)^2 \qquad \textbf{III.7}$$

Cette erreur possède donc un minimum, et dépend de ω, le but est donc de mettre à jour ces pondérations connexions (ω_{ji} et ω_{kj}) afin de trouver l'erreur la plus minimale possible (minimum globale comme le montre la figure ci-dessous), ce calcul se fera inversement, c.à.d. des sorties vers les entrées, d'où l'appellation de "Rétropropagation", et afin d'y parvenir, on calculera le gradient de $\boldsymbol{e}$ par rapport à ω_{ji} et ω_{kj} suivant la relation suivante :

$$\nabla e(\vec{\omega}) = \frac{\partial e}{\partial \omega} \qquad \textbf{III.8}$$

où :

∇ : fait référence au gradient d'une fonction

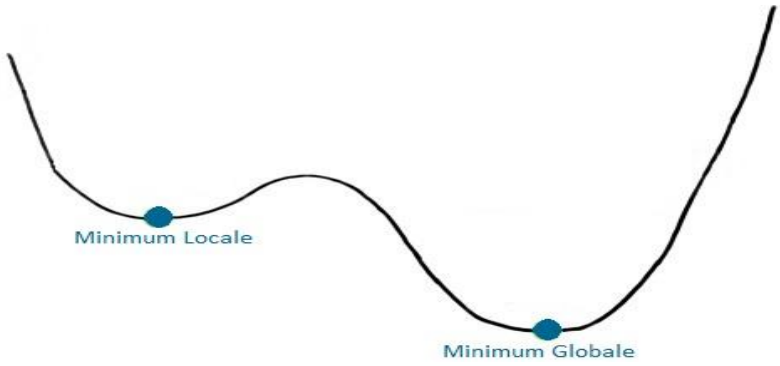


Figure III.1 : Courbe montrant les positions des minimas

Le calcul direct est quasiment impossible, donc afin de simplifier l'équation on utilisera le théorème des dérivées à fonctions composées appelées aussi théorème de chaîne, par exemple pour mettre à jour les poids ω_{kj} il suffira de calculer le produit suivant :

$$\frac{\partial e}{\partial \omega_{kj}} = \frac{\partial e}{\partial S_k} \times \frac{\partial S_k}{\partial R_k} \times \frac{\partial R_k}{\partial \omega_{kj}} \qquad \textbf{III. 9}$$

Ce gradient sera multiplié par un facteur réel $\boldsymbol{\eta}$ ($0 \leq \eta \leq 1$) appelé "taux d'apprentissage", c'est un paramètre qui contrôle la vitesse à laquelle les pondérations sont ajustées, lorsque η est petit, l'algorithme convergera mais le nombre d'itérations va être beaucoup trop grand, par contre dans le cas où η est élevé, la valeur du produit $\eta \times \frac{\partial e}{\partial \omega_{kj}}$ augmentera, l'erreur risque d'osciller autour d'un minimum sans qu'il y ait convergence. L'ajustement des poids ω_{kj} se fera en effectuant la formule suivante :

$$\omega_{kj}{}^{p+1} = \omega_{kj}{}^{p} + \Delta\omega_{kj}{}^{p} \qquad \textbf{III. 10}$$

p : représente l'indice de l'itération actuelle.

p+1 : représente l'indice de la prochaine itération.

$$\Delta\omega_{kj} = -\eta \times \frac{\partial e}{\partial \omega_{kj}} \qquad \textbf{III. 11}$$

A noter que pour éviter les problèmes liés à une stabilisation dans un minimum local, on peut ajouter un terme d'inertie β appelé "momentum", celui-ci permet de sortir des minimums locaux dans la mesure du possible et de poursuivre la descente de la fonction d'erreur. A chaque itération, le changement de poids conserve les informations des changements précédents. Cet effet de mémoire permet d'éviter les oscillations et accélère l'optimisation du réseau. L'équation **III.10** devient ainsi :

$$\Delta\omega_{kj}{}^{p} = -\eta \times \frac{\partial e}{\partial \omega_{kj}} + \beta \times \Delta\omega_{kj}{}^{p-1} \qquad \textbf{III. 12}$$

p-1 : représente l'indice de la précédente itération actuelle.

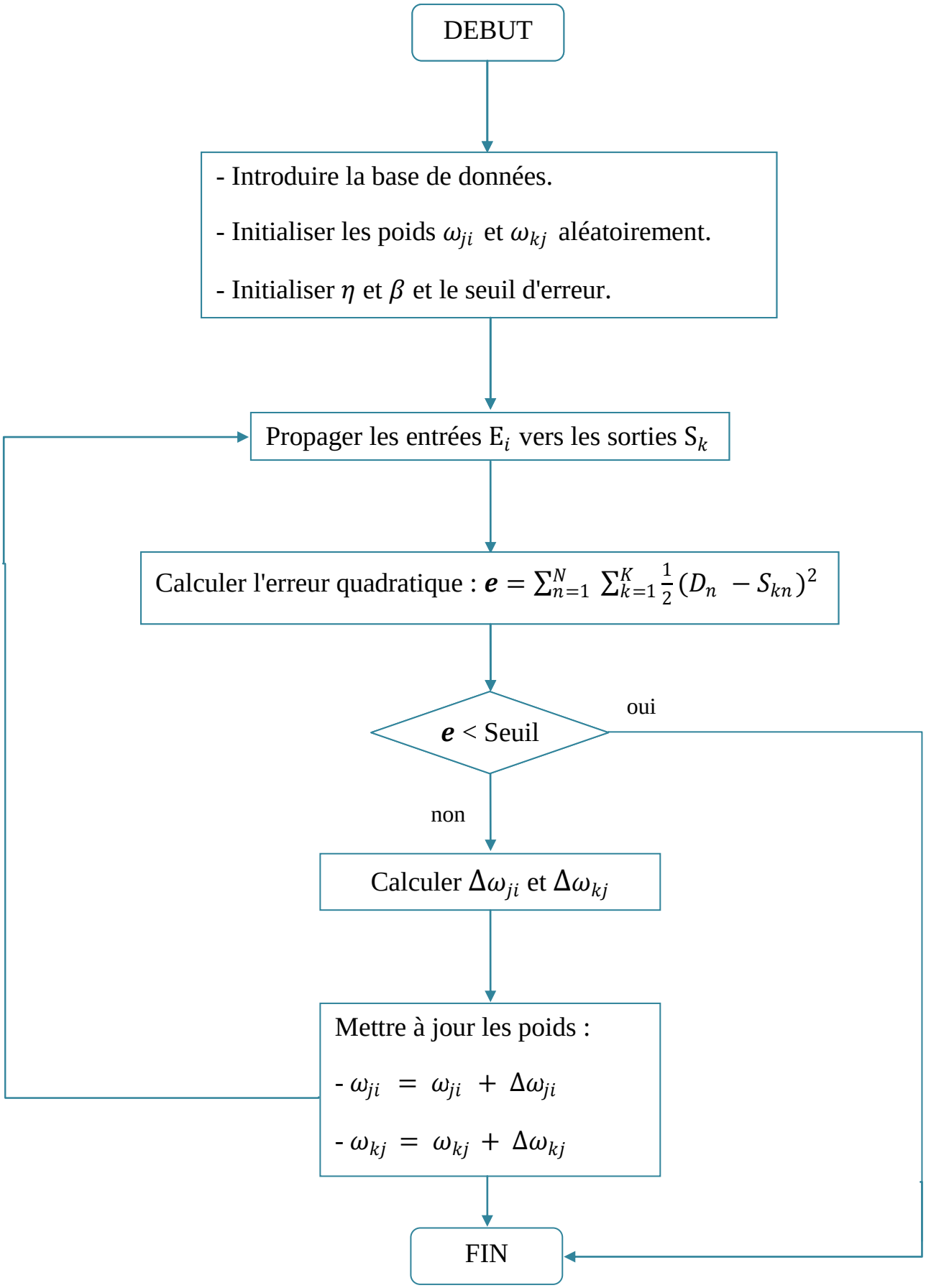


Figure III.2 : Organigramme de l'algorithme de descente du gradient.

I-5) - L'implémentation :

Il existe deux façons d'implémenter l'algorithme de DG, il y'a le type "online", c'est-à-dire que l'on met à jour les poids pour chaque échantillon d'apprentissage présenté dans le réseau de neurones, l'autre méthode est dite en "batch", c'est-à-dire que l'on calcule d'abord les erreurs pour tous les échantillons sans mettre à jour les poids (on additionne les erreurs) et lorsque l'ensemble des données est passé une fois dans le réseau, on applique la rétropropagation en utilisant l'erreur totale.

Comme le montre la figure précédente (**Figure III.2**), on commence par initialiser nos paramètres : taux d'apprentissage, momentum, nombre de neurones dans différentes couche du réseau, et aussi les poids ω_{ji} et ω_{kj}, mais de façon aléatoire (entre -1 et 1), on introduit ensuite nos données à savoir, nos entrées et nos sorties, pour notre cas on introduit 120 échantillons qui représentent des personnes auxquels on a déjà diagnostiquer les deux maladies, chacun doté des 6 paramètres cités auparavant, une fois la paramétrage fini, l'algorithme pourra commencer bien évidemment par la propagation des signaux d'entrée ; il est impératif de mélanger les échantillons pour éliminer toute dépendance entre les exemples successifs, ces derniers ralentissent grandement la convergence de l'algorithme.

Comme il a été mentionné dans la partie **II-8** du chapitre **I**, il est quasiment impossible de déduire l'architecture du réseau le mieux adaptée à notre application, c'est ce qui nous a poussé a intégrer un clavier à 16 touches dans notre système pour la commande de l'ANN, on aura ainsi à saisir le Seuil, le taux d'apprentissage, le momentum, le nombre de couche cachées, et le nombre de neurones contenues dans les couches cachées. Le clavier dispose de huit fils, quatre désignant les lignes et quatre autres désignant les colonnes.

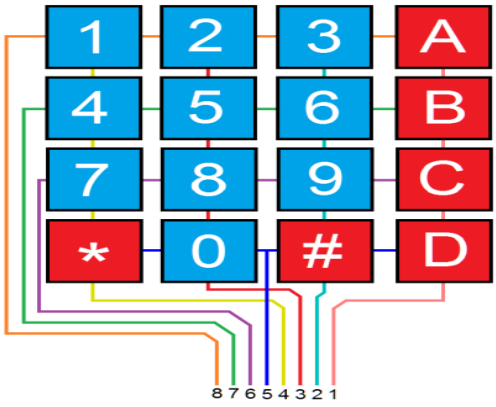


Figure III.3 : Broches de clavier 4x4.

I-6) - Résultats et commentaires :

I-6-a) - Pour un réseau monocouche (1 couche cachée) :

* Pour un Seuil fixe (s = 0.0003) :

On a, pour un premier temps, fixé le Seuil d'erreur à 3×10^{-4} et le taux d'apprentissage à 0.9, et varié le nombre de neurones dans la couche cachée, les 02 tableaux ci-dessous montrent respectivement, les résultats de l'apprentissage avec un momentum de 0.00 et 0.90 :

Architecture du Réseau de neurones	6 - 1 - 2	6 - 2 - 2	6 - 3 - 2	6 - 4 - 2	6 - 5 - 2
Itérations	Pas de convergence	9360	6763	5122	6094
Architecture du Réseau de neurones	6 - 6 - 2	6 - 7 - 2	6 - 8 - 2	6 - 9 - 2	6 - 10 - 2
Itérations	4172	4149	3786	4061	3640

Tableau III.1 : Apprentissage sans momentum.

Architecture du Réseau de neurones	6 - 1 - 2	6 - 2 - 2	6 - 3 - 2	6 - 4 - 2	6 - 5 - 2
Itérations	Pas de convergence	967	851	624	611
Architecture du Réseau de neurones	6 - 6 - 2	6 - 7 - 2	6 - 8 - 2	6 - 9 - 2	6 - 10 - 2
Itérations	470	478	318	335	419

Tableau III.2 : Apprentissage avec momentum.

En comparant les 02 tableaux, on déduit directement que le rôle du momentum est incontournable pour un bon apprentissage, il a réduit le nombre d'itérations de 90% environ.

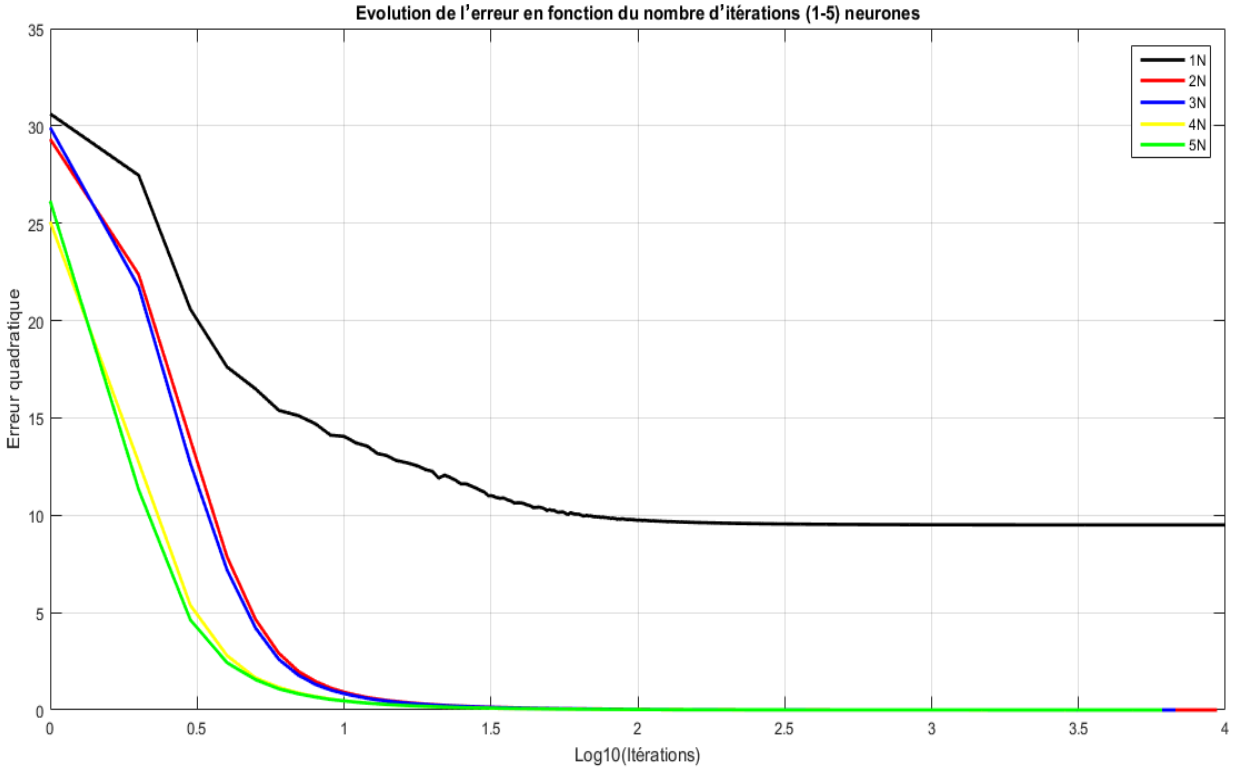


Figure III.4 : Evolution de l'erreur pour des neurones de 1 à 5 dans la couche cachée ($\beta = 0$)

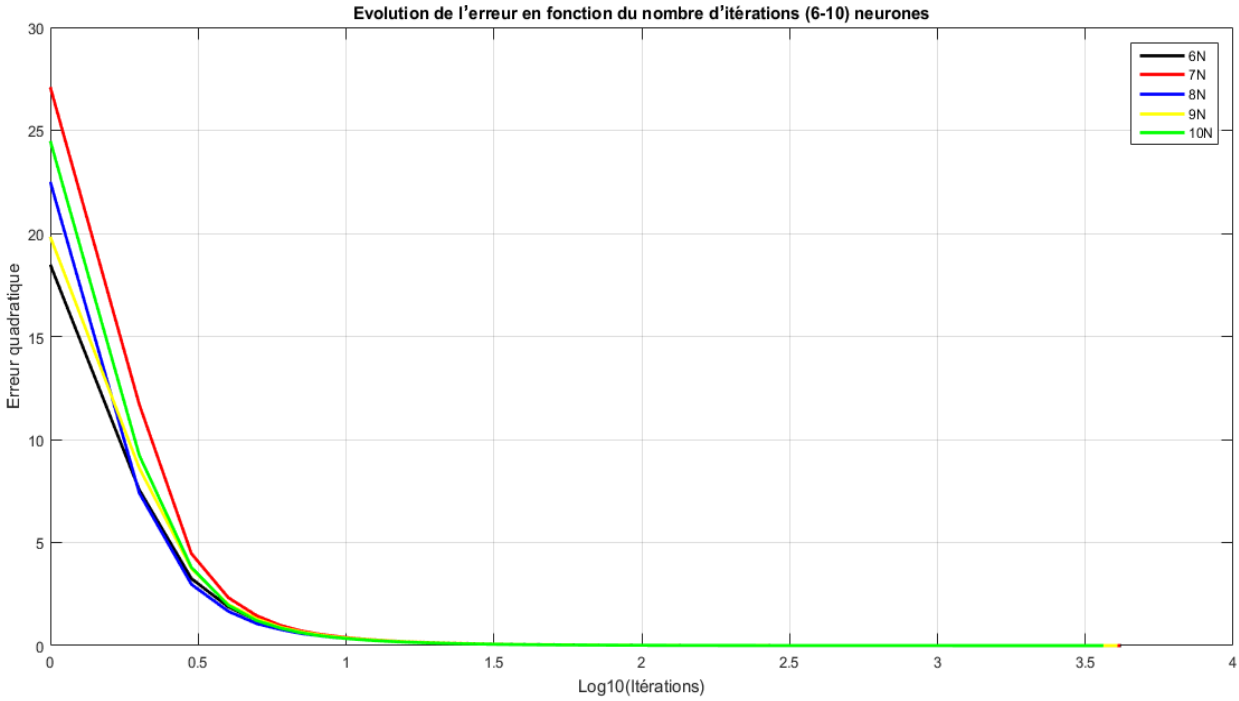


Figure III.5 : Evolution de l'erreur pour des neurones de 6 à 10 dans la couche cachée ($\beta = 0$)

Après 1000 itérations, on remarque que l'erreur quadratique se rapproche plus rapidement du seuil avec 10 neurones dans la couche cachée.

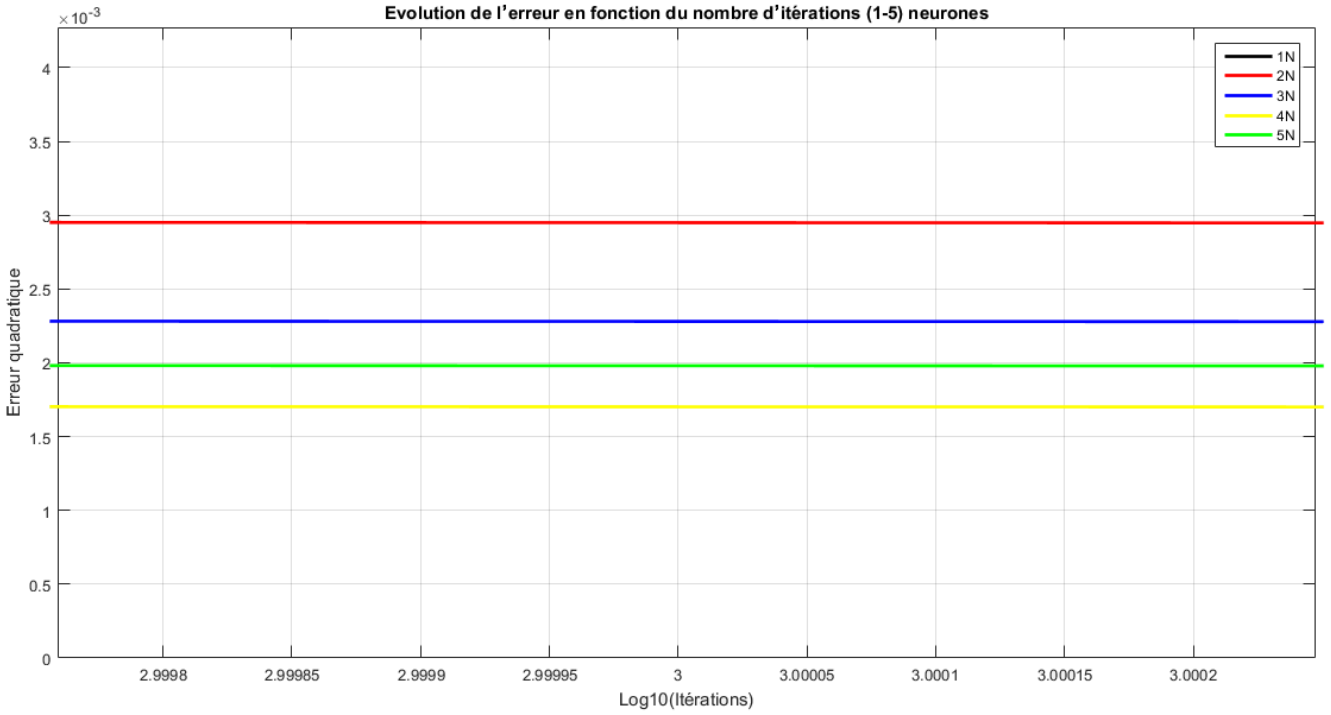


Figure III.6 : Evolution de l'erreur après 1000 itérations (1 à 5 dans la couche cachée).

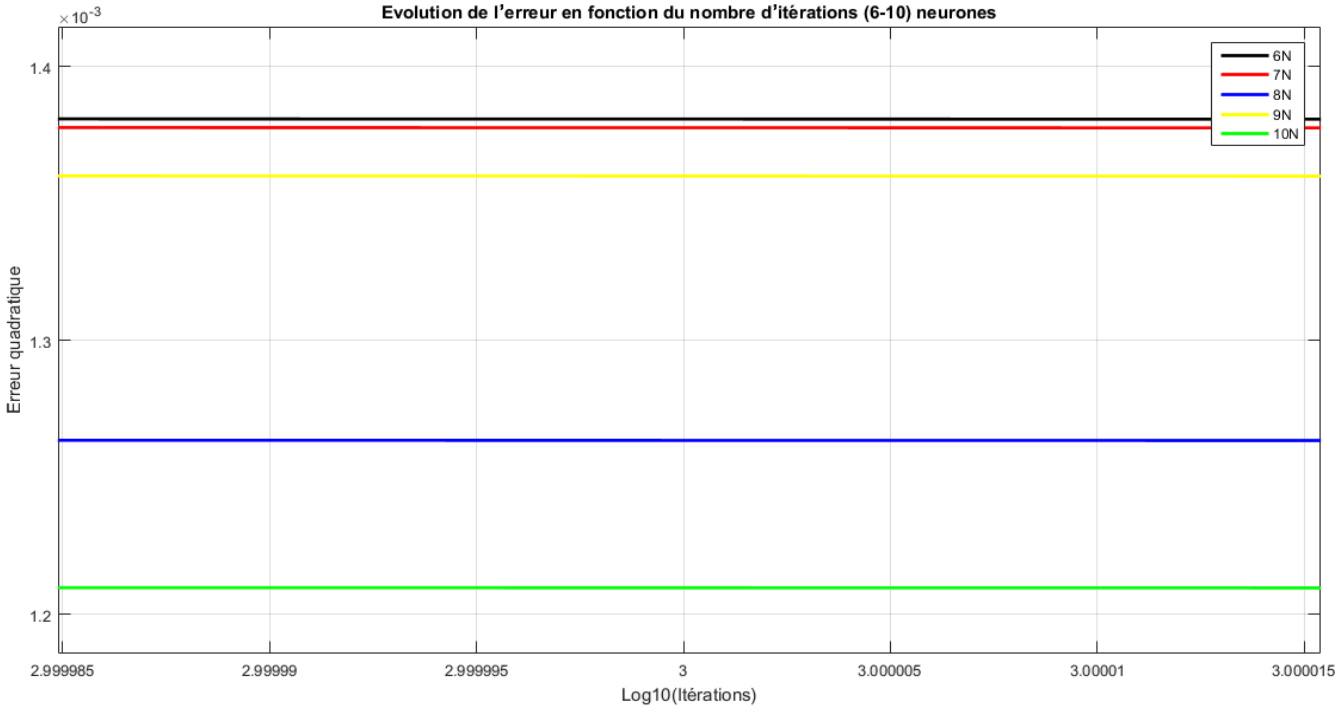


Figure III.7 : Evolution de l'erreur après 1000 itérations (6 à 10 dans la couche cachée).

Donc pour un réseau de neurones sans momentum, l'architecture la mieux adaptée est celle qui possède 10 neurones dans la couche cachée, la figure suivante montre une comparaison des itérations obtenues en fonction du nombre de neurones utilisées.

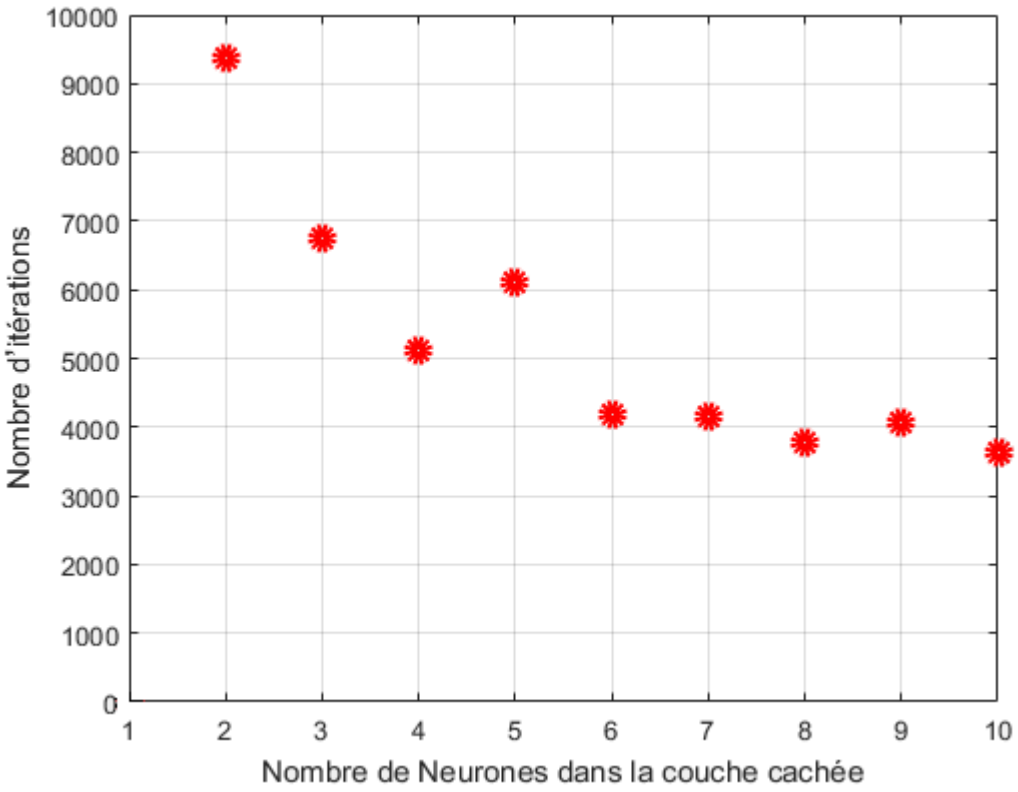


Figure III.8 : Nombre d'itérations en fonction des neurones.

Evidemment les résultats précédents nous importe peu, la seule raison de l'étude réside dans le but de démontrer l'efficacité et l'importance du momentum $\boldsymbol{\beta}$, ainsi et pour ce qui se suivra, l'intégration de ce paramètre sera primordial, on s'intéressera donc au tableau **III.2**.

D'après les résultats acquis, on remarque que plus le nombre de neurones dans la couche cachée est augmenté, le nombre d'itérations est quasiment réduit, mais la meilleure performance est obtenue avec une architecture en 6-8-2, l'erreur quadratique se colle au seuil en 318 itérations seulement. Les courbes ci-dessous représentent des comparaisons entre les différents architectures issues de l'analyse précédente ; la première figure (**Figure III.8)** représente l'évolution de l'erreur en fonction des itérations en utilisant 1 à 5 neurones dans la couche cachée, quant' à la seconde en utilisant 6 à 10; les deux dernières **Figure III.11** et **Figure III.12** sont prises à 300 itérations uniquement pour comparer la rapidité entre les différentes architectures. (Tous les axes des courbes seront à l'échelle logarithmique)

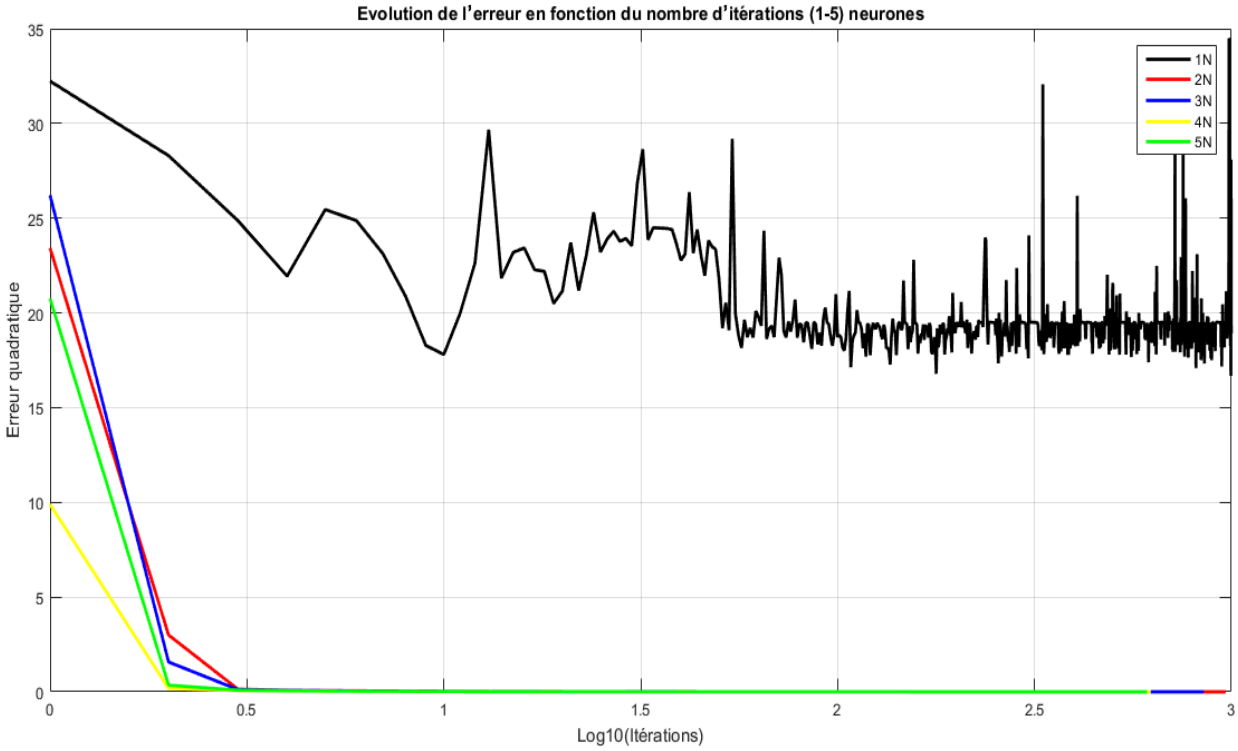


Figure III.9 : Evolution de l'erreur en fonction des itérations (β = 0.9)

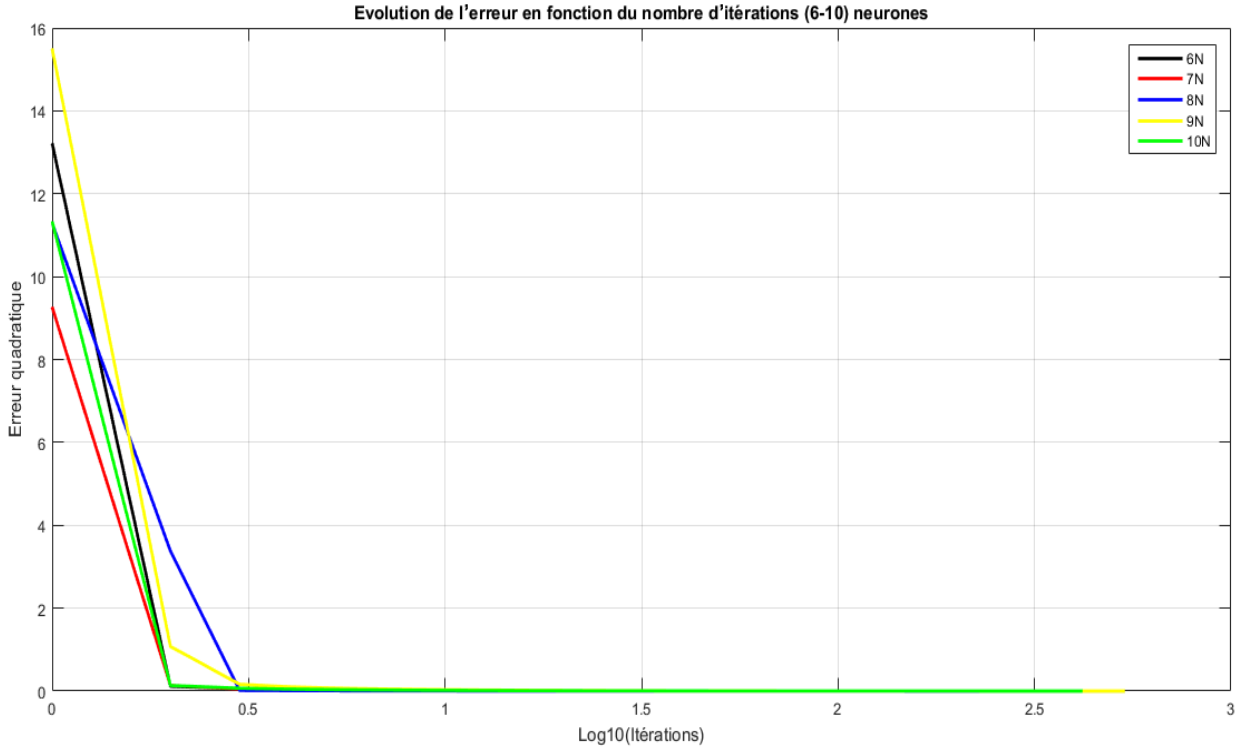


Figure III.10 : Evolution de l'erreur en fonction des itérations (β = 0.9)

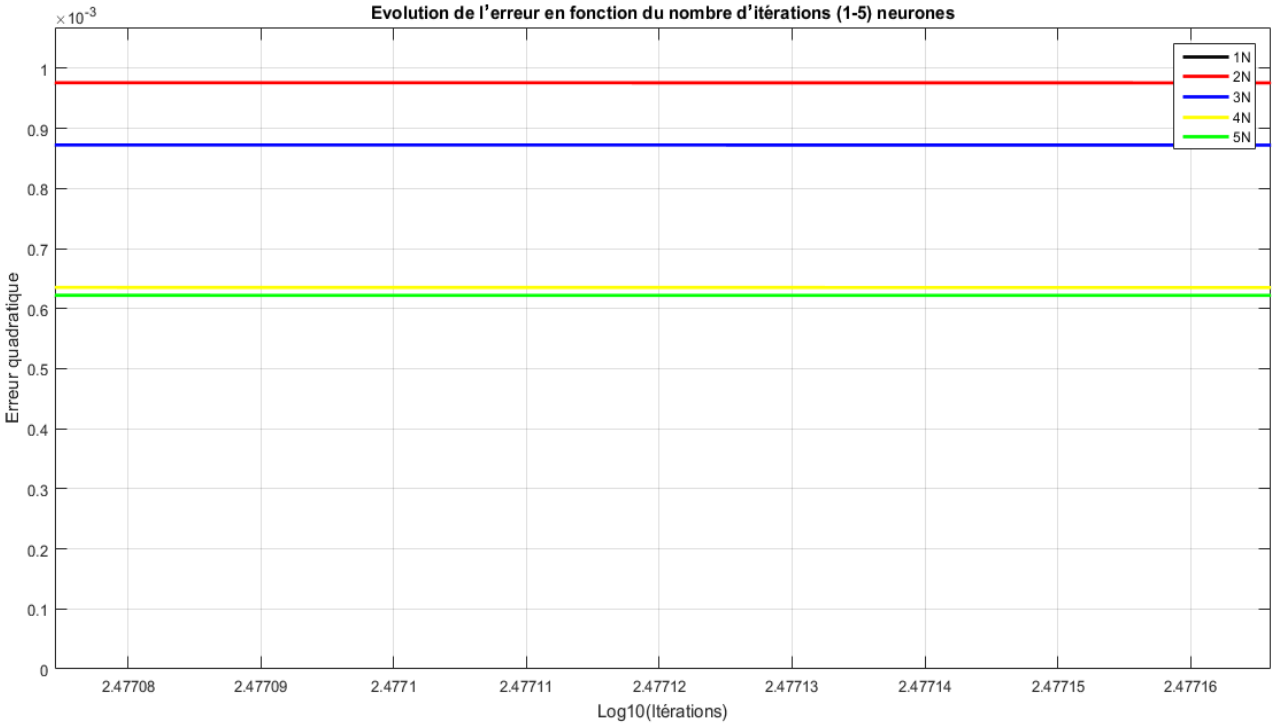


Figure III.11 : Evolution de l'erreur après 300 itérations (1 à 5 dans la couche cachée).

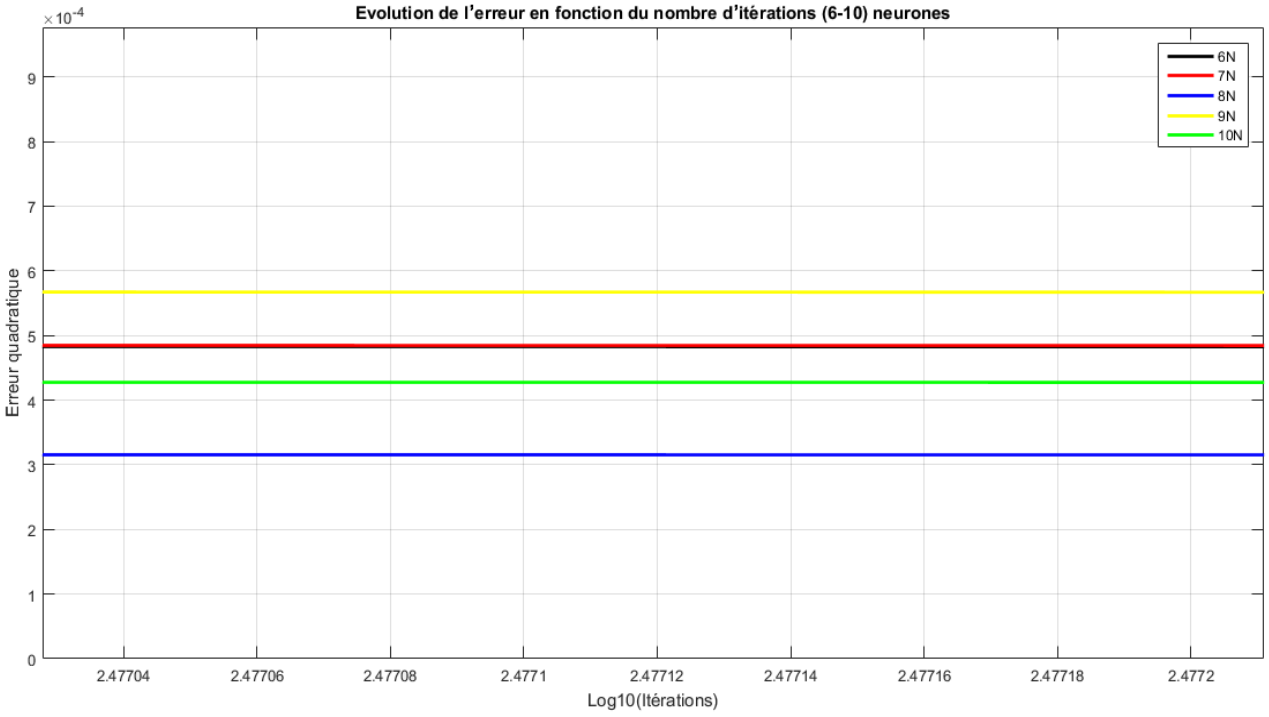


Figure III.12 : Evolution de l'erreur après 300 itérations (6 à 10 dans la couche cachée).

La meilleure performance est obtenue pour une architecture 6-8-2 comme le montre la figure suivante :

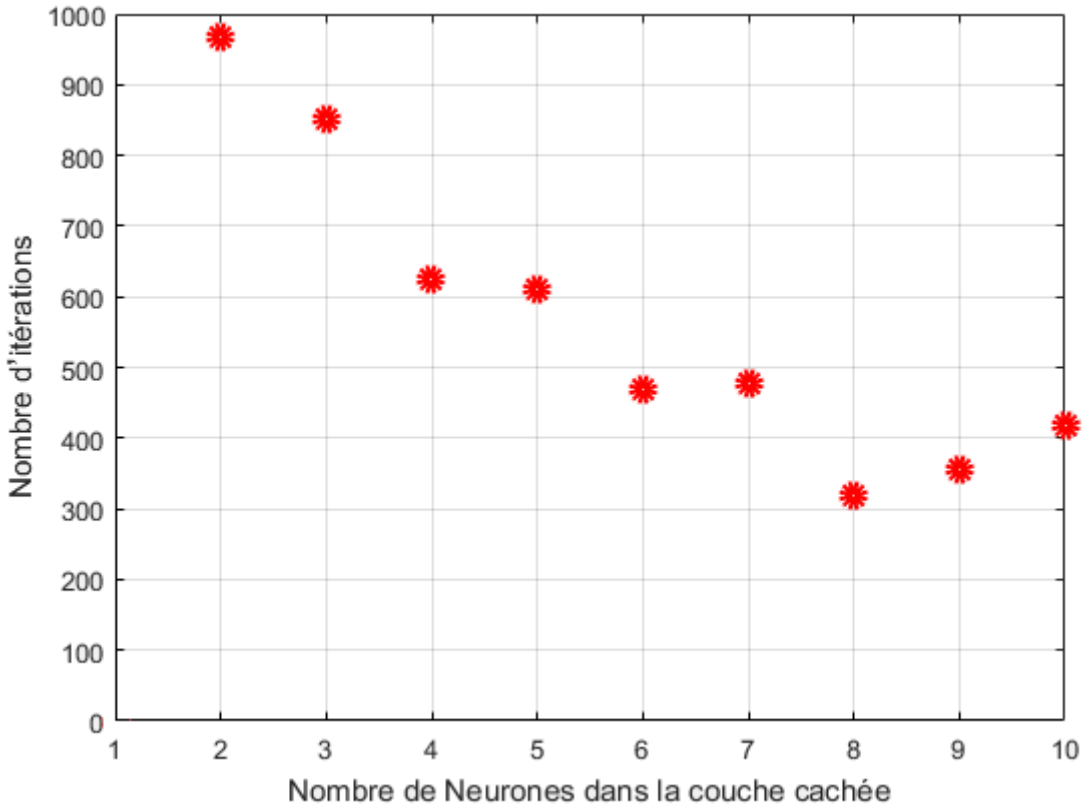


Figure III.13 : Nombre d'itérations en fonction des neurones.

* Pour un nombre d'itérations fixe (i = 1000) :

Le traitement des ANN se fait de plusieurs façons, on traitera le même exemple et on modifiera notre programme pour savoir laquelle des architectures a la plus basse erreur, afin d'y parvenir, chaque architecture effectuera 1000 itérations, on comparera ainsi l'erreur obtenue et la meilleure architecture est celle qui a la plus petite, le tableau ci-dessous décrit les résultats obtenues :

N°	Nombre de neurones Dans la couche cachée	EQM apprentissage (10^{-6})
1	1	pas de convergence
2	2	293.22
3	3	269.43
4	4	243.40
5	5	186.08
6	6	133.41
7	7	127.17
8	**8**	**102.84**
9	9	136.64
10	10	139.44

Tableau III.3 : Erreurs en fonction du nombre de neurones dans la couche cachée.

Les poids associés au réseau de neurones d'architecture 6-8-2 sont les suivants :

* Poids Entrées - Couche cachée :

$$\omega_{ji} = \begin{bmatrix} 0.2365 & -1.0787 & 0.2670 & -2.2685 & -0.6369 & -1.2274 & 0.9681 \\ -0.0732 & -2.0825 & 2.8260 & -2.9487 & -1.1132 & -0.2865 & 1.0669 \\ -6.1518 & 4.2056 & 4.3083 & -0.2234 & -2.2601 & 4.5045 & -2.1985 \\ -1.2337 & -1.0724 & 2.5136 & -2.6440 & -1.2931 & 0.2486 & 0.9442 \\ -2.9673 & 1.2078 & 1.2796 & -1.7637 & -1.7140 & -0.2767 & -0.6326 \\ -2.7896 & 3.3316 & 1.7828 & 0.3202 & -0.4149 & 1.5832 & -1.6618 \\ 0.1861 & 0.6784 & -3.1335 & 2.9137 & 2.4160 & -0.7120 & -1.0478 \\ -0.3451 & -1.5652 & 3.0789 & -3.8249 & -2.1912 & 0.3057 & 1.9146 \end{bmatrix}$$

Les lignes représentent les huit neurones de la couche cachée et les colonnes les 6 entrées + le biais représenté dans la dernière colonne.

* Poids Couche cachée - Sorties :

$$\omega_{kj} = \begin{bmatrix} -1.6859 & -5.0413 & -2.5994 & -3.7824 & -1.5883 & 2.7540 & 4.7539 & -6.4342 & 4.3400 \\ -2.6926 & -2.0569 & 10.2456 & 0.1086 & 1.4415 & 4.2670 & -3.6890 & -1.3506 & -3.2411 \end{bmatrix}$$

Les lignes représentent les sorties et les colonnes, les neurones de la CC + le biais.

I-6-b) - Pour un réseau multicouche (2 couches cachées) :

On verra dans cette partie l'influence de l'ajout d'une deuxième couche cachée sur l'ANN, on essayera quelques architectures et comparera nos résultats aux précédentes (I-6-b), on déduira ainsi le meilleur modèle adapté a notre application. A noter que l'ajout de couches à fonctions linéaires ne sert à rien car la combinaison de plusieurs couches linéaires peut toujours se ramener à une seule couche linéaire équivalente [35].

N°	Nombre de neurones Dans la première couche cachée	Nombre de neurones Dans la deuxième couche cachée	EQM apprentissage (10^{-6})
1	2	4	116.29
2	3	4	186.64
3	3	5	195.96
4	4	5	196.68
5	4	3	271.53
6	5	4	147.57
7	5	3	213.21
8	6	5	115.89
9	6	4	205.44
10	7	4	130.22
11	7	5	103.64
12	7	3	170.85
13	8	3	223.78
14	8	5	111.64
15	8	6	129.53
16	9	6	071.01
17	9	4	224.82
18	9	3	226.70

Tableau III.3 : Erreurs en fonction du nombre de neurones dans les couches cachées.

D'après le tableau ci-dessus, la meilleure architecture pour l'apprentissage de l'ANN est l'architecture N°16 qui dispose de 9 neurones dans la première couche cachée et 6 dans la seconde ; En comparant cette architecture à celle du réseau monocouche (6-8-2), on déduit que la précision est au niveau de la première citée, les courbes ci-dessous représentent l'évolution de l'erreur des deux architectures.

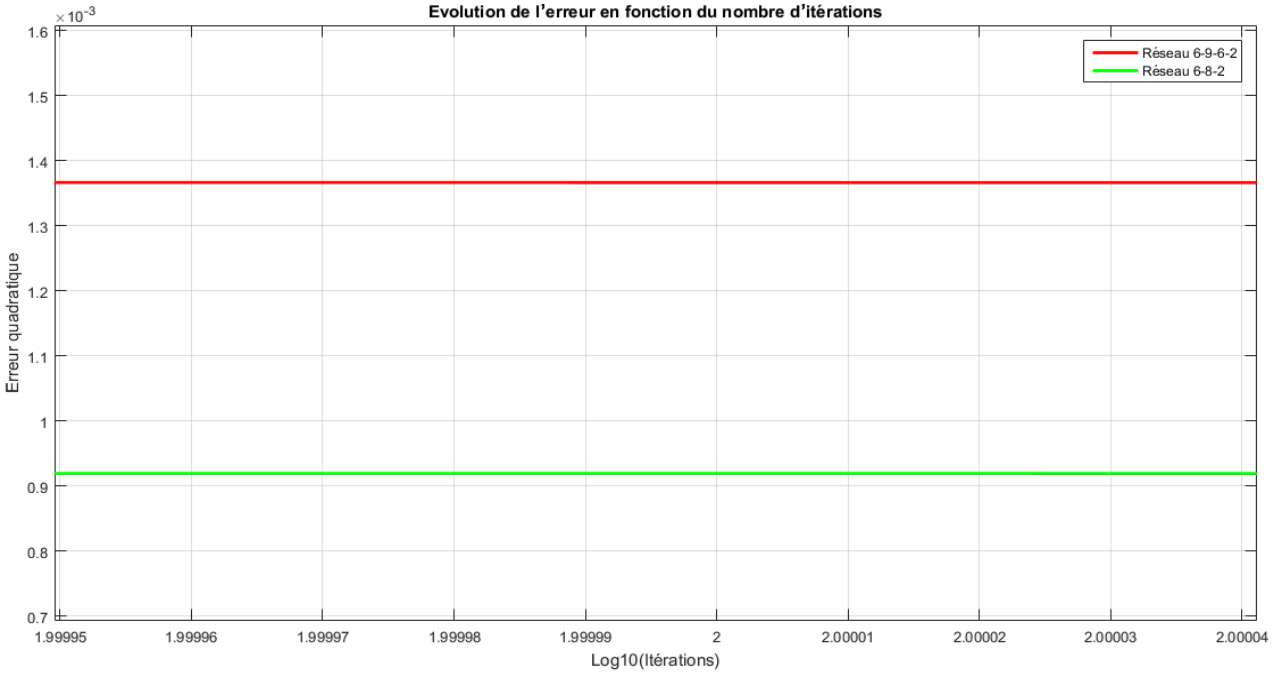


Figure III.14 : Evolution de l'erreur après 100 itérations.

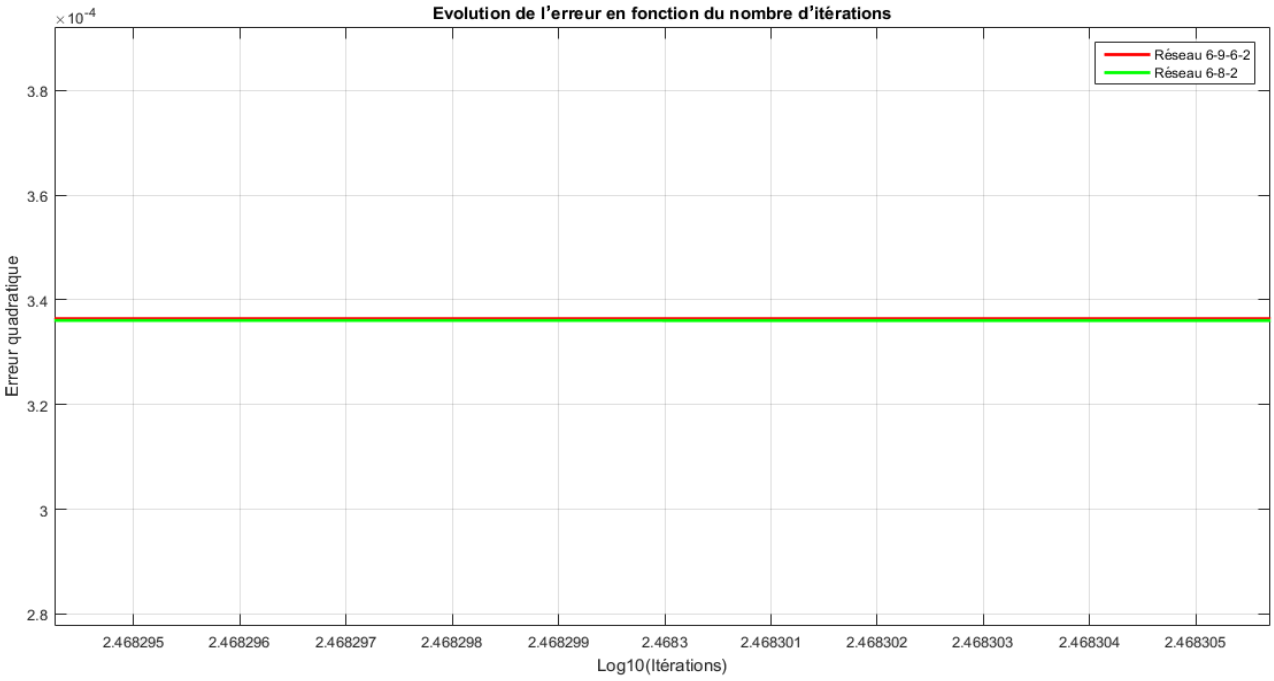


Figure III.15 : Erreur après 294 itérations (chevauchement des 02 erreurs).

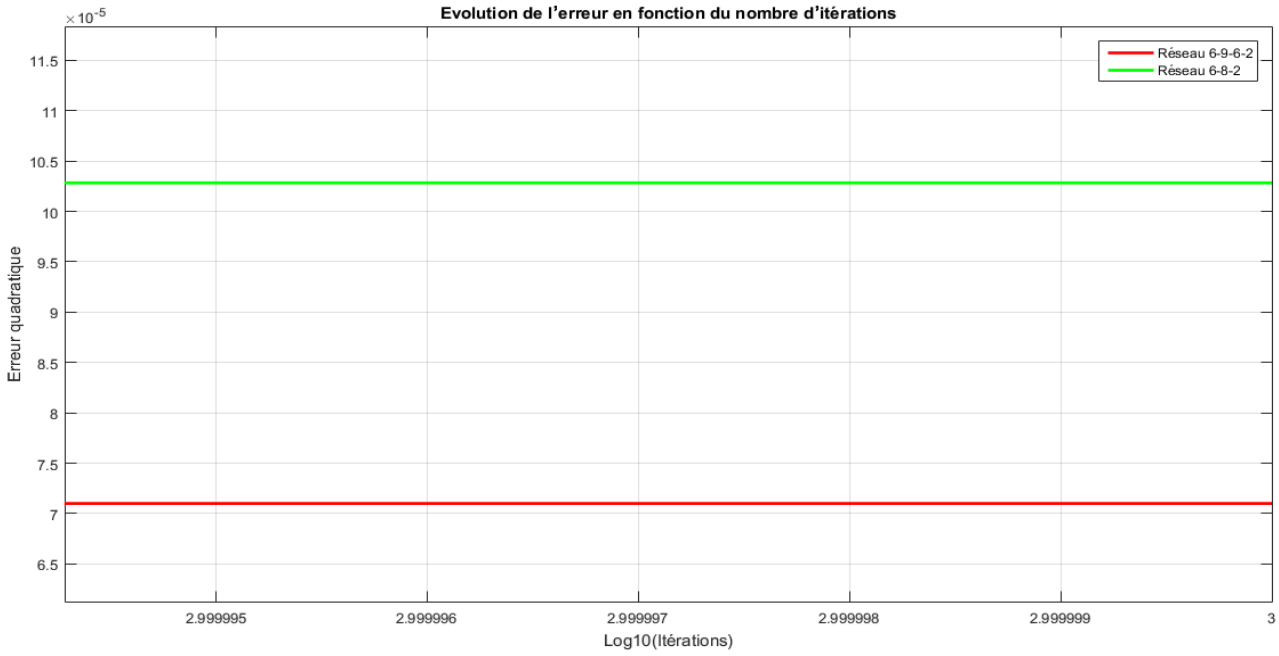


Figure III.16 : Erreur à la fin du cycle (1000ème itération).

Après 100 itérations on remarque que le réseau monocouche (6-8-2) avance plus rapidement que le multicouche (6-9-6-2) jusqu'à l'itération 294, ou les deux modèles se chevauchent à une erreur de 3.36×10^{-4} (Figure III.15) et à partir de là, le multicouche prend le dessus et converge plus rapidement en terminant le cycle d'itérations avec un taux d'erreur plus faible que le monocouche.

I-6-c) - La classification :

L'apprentissage effectué nous a permis d'avoir une structure neuronal complète pour classifier nos deux maladies décrites auparavant, on pourra commander notre modèle en utilisant le clavier 4x4 ; le programme demandera à l'utilisateur d'introduire au fur et à mesure les symptômes et effectuera par la suite la propagation à travers le réseau neuronal et obtiendra les diagnostics voulus.

I-6-c) - Le modèle retenu :

Dorénavant, le modèle qu'on gardera pour la classification est celle du multicouche (6-9-6-2), toutes les sortie des neurones sont obtenues en utilisant la fonction d'activation non-linéaire de type sigmoïde $f\ (x) = \frac{1}{1+ e^{-x}}$, la figure suivante montre l'architecture de notre réseau :

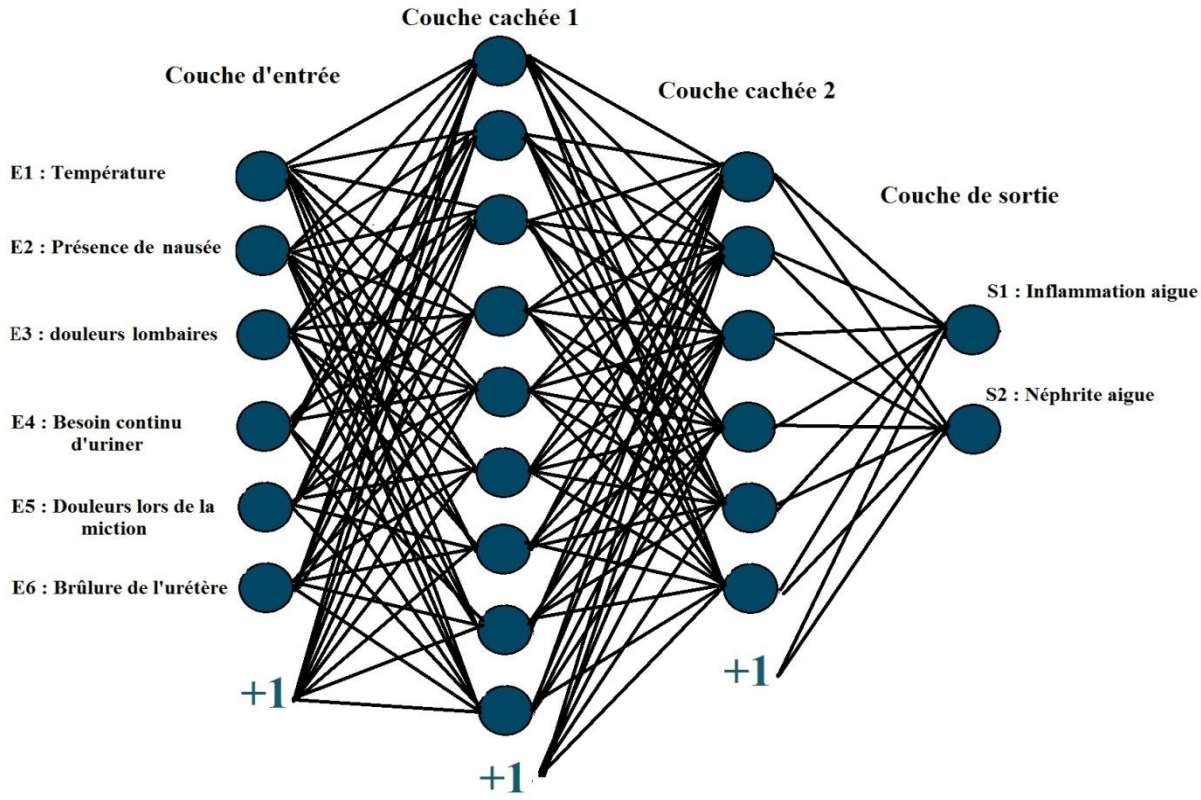


Figure III.17 : Architecture du réseau multicouche 6-9-6-2.

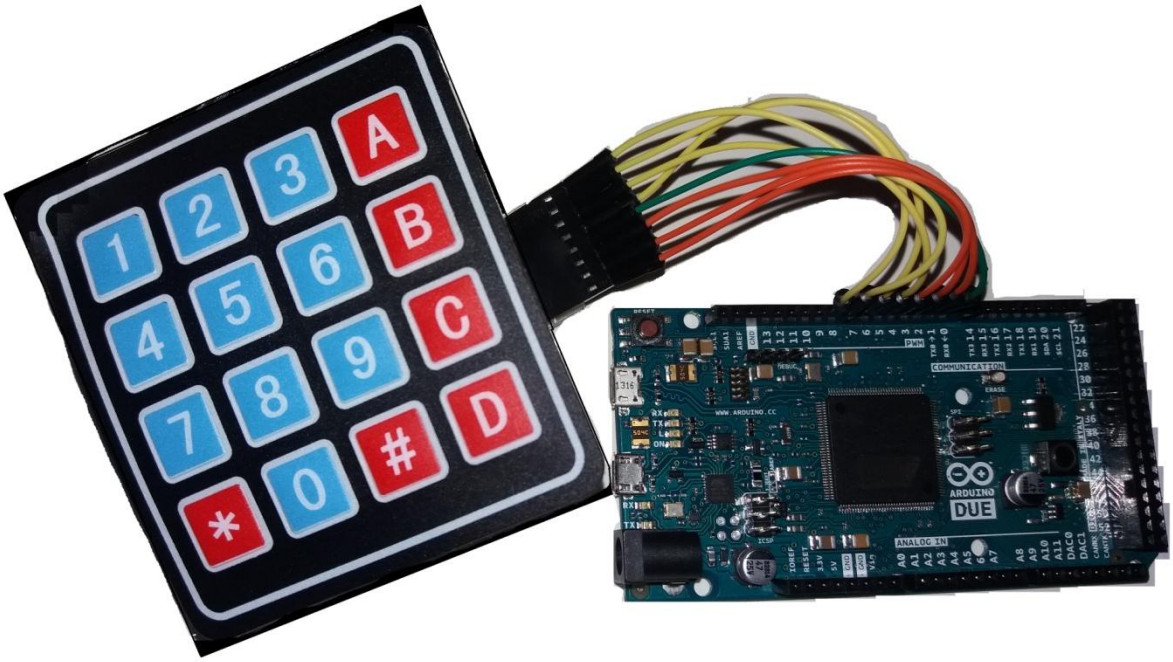


Figure III.18 : Clavier 4x4 relié à l'Arduino DUE.

II) - Modélisation d'une thermistance CTN par les ANNs pour la compensation de la non-linéarité :

II-1) - Description de l'application:

Dans cette deuxième application on modélisera une thermistance CTN, autrement dit, on établira un modèle mathématique qui nous permettra d'estimer la température et ce après avoir effectué l'apprentissage du réseau neuronal, cette modélisation sera accompagnée d'une importante tâche, à savoir la compensation de la non-linéarité du capteur CTN.

Ce dernier, bien qu'il marque une présence plus que remarquable dans le monde d'électronique, précisément l'industrie, néanmoins sa forte non-linéarité pose de sérieux problèmes quant à l'exploitation de sa grandeur physique notamment pour le contrôle de machines industrielles, la mesure du flux d'un liquide ou simplement une acquisition de la température, plusieurs méthodes ont été proposées afin d'avoir une réponse linéaire, soit par l'ajout d'une résistance parallèle à la CTN [36], ou bien par calculer l'équation de la régression linéaire $f(R) = a.T + b$ (où R représente la résistance de la CTN et T la température), ou bien encore la méthode très fructueuse qu'on développera un peu plus tard qui est basée sur les ANN's ; En effet les réseaux de neurones offrent des performances intéressantes d'approximations de fonctions linéaires, ils représentent donc une solution pertinente pour répondre à la problématique citée auparavant. Un autre avantage de l'approche proposée consiste à convertir les sorties des capteurs à un format numérique qui peut être facilement transmis des distances relativement longues sans distorsion [37].

II-2) - La thermistance CTN :

II-2-a) - Définition :

La thermistance CTN (Coefficient de Température Négatif, en anglais NTC, *Negative Temperature Coefficient*) est une résistance dont la résistance varie avec le changement de température, elle diminue de façon uniforme quand la température augmente. elle constitué de matériaux semi-conducteurs (manganèse, cobalt, cuivre et nickel) et, par conséquent, leur résistivité est plus sensible à la température. La figure ci-dessous représente le symbole électrique d'une CTN.

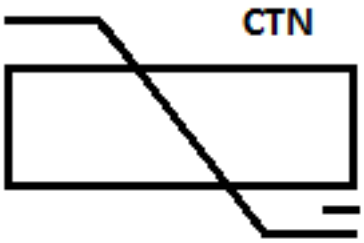


Figure III.19 : Symbole de la CTN.

II-2-b) - Modèles mathématiques :

* L'équation de Steinhart-Hart : L'approximation de la relation de résistance à la température d'une thermistance ne fonctionne bien que dans une petite plage de température. Pour la mesure de température sans erreur, une approximation plus précise sous la forme d'une équation est proposée : l'équation de Steinhart-Hart est un modèle largement utilisé et possède une erreur de 0,02 ° C, sa loi est décrit ci-dessous :

$$\frac{1}{T} = a + b \times ln(R) + c \times (ln(R))^3 \qquad \textbf{III. 13}$$

Où :

a, b et c sont des paramètres Steinhart-Hart, ils sont calculés à partir de mesures expérimentales de résistance

T est la température absolue.

R est la résistance. les constantes a, b et c.

* L'équation du $\boldsymbol{\beta}$: Les thermistances NTC sont caractérisées par un autre type d'équation connue comme équation du paramètre B ou β. L'équation β est similaire à l'équation Steinhart-Hart avec :

$$a = \frac{1}{T_0} - \frac{1}{\beta} \times ln(R_0) \qquad \textbf{III. 14}$$

$$b = \frac{1}{\beta} \qquad \textbf{III. 15}$$

$$c = 0 \qquad \textbf{III. 16}$$

Donc :

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{\beta} \times ln(\frac{R}{R_0}) \qquad \textbf{III.17}$$

Où :

T_0 : Représente la température de référence à laquelle les mesures sont prises

R_0 : est la résistance correspondant à T_0 de température

D'après l'équation, on pourra désormais trouver l'équation de la résistance R, en fonction des paramètres de références T_0 et R_0 et la température T:

$$R = R_0 \times e^{-\beta\,(\frac{1}{T_0} - \frac{1}{T})} \qquad \textbf{III.18}$$

Le paramètre β est très important et dépend des matériaux et composants constituant la CTN, il diffère d'une thermistance à une autre, et varie généralement de 2880 to 4570 K. La figue suivante représente l'évolution de la résistance R de la CTN en fonction de la température T, la figure montre clairement la non linéarité entre eux.

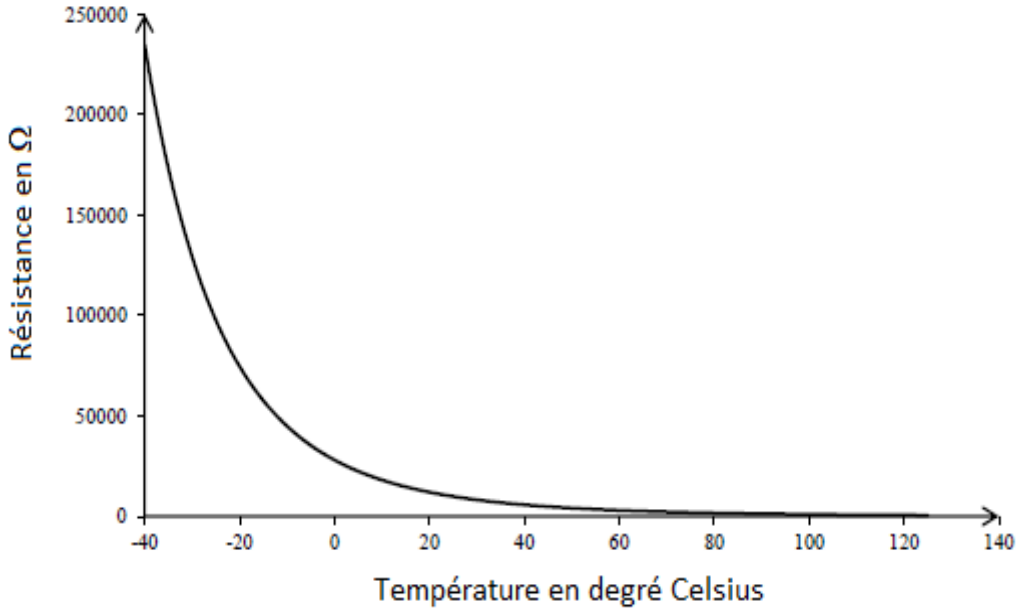


Figure III.20 : Evolution de la résistance en fonction de la température

I-3) - L'algorithme de Levenberg-Marquardt :

L'algorithme de Levenberg-Marquardt (ou simplement algorithme LM) fournit une solution au problème des fonctions non linéaires, sa convergence est rapide et stable, il est très bien adapté aux ANNs de taille moyenne et de petites taille. On répartira cette section en quatre (04) sous section dans le but d'interpréter au mieux l'algorithme LM dans le domaine des ANNs.

I-3-a) - L'algorithme de DG :

On commence par rappeler l'algorithme de DG car c'est à partir de là que celui de LM prend forme, certes le premier nous a permis de résoudre un problème de classification mais sans pour autant le minimiser, il reste globalement lent et pas vraiment fiable pour la résolution des problèmes de non-linéarité, la raison est qu'il est issu de la dérivée première de l'erreur globale, les éléments du vecteur du gradient seraient très faibles ce qui implique un très petit changement de poids.

I-3-b) - La méthode de Newton :

Elle suppose que touts les composants des gradients$\nabla_1 e(\vec{\omega})$,$\nabla_2 e(\vec{\omega})$, ..., $\nabla_N e(\vec{\omega})$ sont des fonctions de poids et tout les poids sont linéairement indépendants :

$$\begin{cases} \nabla_1 = F_1(\omega_1, \omega_2, \dots, \omega_N) \\ \nabla_2 = F_2(\omega_1, \omega_2, \dots, \omega_N) \\ \vdots \\ \nabla_N = F_N(\omega_1, \omega_2, \dots, \omega_N) \end{cases} \qquad \textbf{III. 19}$$

où :

F_1 , F_2, ... , F_N, représentent des relations non-linéaires entre les poids et les composants de gradients.

Pour chaque $\nabla_{i:1\to N}$ de, on effectue la série de Taylor [38] :

$$\begin{cases} \nabla_1 = \nabla_{1,0} + \frac{\partial \nabla_1}{\partial \omega_1}\,\Delta\omega_1 + \frac{\partial \nabla_1}{\partial \omega_2}\,\Delta\omega_2 + \cdots + \frac{\partial \nabla_1}{\partial \omega_N}\,\Delta\omega_N \\ \nabla_2 = \nabla_{2,0} + \frac{\partial \nabla_2}{\partial \omega_1}\,\Delta\omega_1 + \frac{\partial \nabla_2}{\partial \omega_2}\,\Delta\omega_2 + \cdots + \frac{\partial \nabla_2}{\partial \omega_N}\,\Delta\omega_N \\ \vdots \\ \nabla_N = \nabla_{N,0} + \frac{\partial \nabla_N}{\partial \omega_1}\,\Delta\omega_1 + \frac{\partial \nabla_N}{\partial \omega_2}\,\Delta\omega_2 + \cdots + \frac{\partial \nabla_N}{\partial \omega_N}\,\Delta\omega_N \end{cases} \quad \textbf{III.20}$$

on sait que $\nabla_i = \frac{\partial e}{\partial \omega_i}$, donc :

$$\frac{\partial \nabla_i}{\partial \omega_j} = \frac{\partial \frac{\partial e}{\partial \omega_i}}{\partial \omega_j} = \frac{\partial^2 e}{\partial \omega_i \partial \omega_j} \quad \textbf{III.21}$$

En intégrant l'équation **III.21** dans **III.20**, sans oublier que pour avoir les minimums de la fonction d'erreur globale, chaque élément du vecteur gradient doit être nul, par conséquent les éléments du côté gauche de l'équation **III.20** doivent être nuls, sans trop entrer dans les détails de calculs et de simplifications on donnera la solutions des équations, on aura donc N équations pour N paramètres, ainsi tout les $\Delta\omega_i$ pourront être calculés, et les poids pourront être mis à jour selon la forme matricielle suivante :

$$\begin{bmatrix} -\nabla 1 \\ -\nabla 2 \\ \vdots \\ -\nabla_N \end{bmatrix} = \begin{bmatrix} -\frac{\partial e}{\partial \omega_1} \\ -\frac{\partial e}{\partial \omega_2} \\ \vdots \\ -\frac{\partial e}{\partial \omega_N} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 e}{\partial {\omega_1}^2} & \frac{\partial^2 e}{\partial \omega_1 \partial \omega_2} & \cdots & \frac{\partial^2 e}{\partial \omega_1 \partial \omega_N} \\ \frac{\partial^2 e}{\partial \omega_2 \partial \omega_1} & \frac{\partial^2 e}{\partial {\omega_2}^2} & \cdots & \frac{\partial^2 e}{\partial \omega_2 \partial \omega_N} \\ & \vdots & & \\ \frac{\partial^2 e}{\partial \omega_N \partial \omega_1} & \frac{\partial^2 e}{\partial \omega_N \partial \omega_2} & \cdots & \frac{\partial^2 e}{\partial {\omega_N}^2} \end{bmatrix} \times \begin{bmatrix} \Delta\omega_1 \\ \Delta\omega_2 \\ \vdots \\ \Delta\omega_N \end{bmatrix} \quad \textbf{III.22}$$

où $H = \begin{bmatrix} \frac{\partial^2 e}{\partial {\omega_1}^2} & \frac{\partial^2 e}{\partial \omega_1 \partial \omega_2} & \cdots & \frac{\partial^2 e}{\partial \omega_1 \partial \omega_N} \\ \frac{\partial^2 e}{\partial \omega_2 \partial \omega_1} & \frac{\partial^2 e}{\partial {\omega_2}^2} & \cdots & \frac{\partial^2 e}{\partial \omega_2 \partial \omega_N} \\ & \vdots & & \\ \frac{\partial^2 e}{\partial \omega_N \partial \omega_1} & \frac{\partial^2 e}{\partial \omega_N \partial \omega_2} & \cdots & \frac{\partial^2 e}{\partial {\omega_N}^2} \end{bmatrix}$ **III.23** : représente la matrice Hessienne.

En combinant les équations **III.8** et **III.23** avec **III.22** on déduit les relations suivantes :

$$-\nabla = H \times \Delta\omega \quad \textbf{III.24} \qquad \Rightarrow \qquad \Delta\omega = -H^{-1} \times \nabla \quad \textbf{III.25}$$

d'où on peut extraire l'équation générale de la mise à jour des poids ω_{k+1} avec la méthode de Newton telle que :

$$\omega_{k+1} = \omega_k - {H_k}^{-1} \times \nabla_k \qquad \textbf{III.26}$$

II-3-c) - L'algorithme de Gauss-Newton :

Si la méthode de Newton est appliqué pour la mise à jour des poids afin d'obtenir H, il faudra calculer les dérivées de seconde ordre de la fonction d'erreur, ce qui peut être très compliquée, donc pour simplifier le processus de calcul, la matrice Jacobienne est introduite comme suit :

$$J = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial \omega_1} & \frac{\partial e_{1,1}}{\partial \omega_2} & \cdots & \frac{\partial e_{1,1}}{\partial \omega_N} \\ \frac{\partial e_{1,2}}{\partial \omega_1} & \frac{\partial e_{1,2}}{\partial \omega_2} & \cdots & \frac{\partial e_{1,2}}{\partial \omega_N} \\ & \vdots & & \\ \frac{\partial e_{P,M}}{\partial \omega_1} & \frac{\partial e_{P,M}}{\partial \omega_2} & \cdots & \frac{\partial e_{P,M}}{\partial \omega_N} \end{bmatrix} \qquad \textbf{III.27}$$

Où :

P : Représente le nombre d'échantillons pour l'apprentissage.

M : Représente le nombre de sortie du réseau neuronal.

L'équation **III.21** peut se développer et s'écrire en fonction de J :

$$\nabla_i = \frac{\partial e}{\partial \omega_i} = \frac{\partial(\frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} {e_{p,m}}^2)}{\omega_i} = \sum_{p=1}^{P}\sum_{m=1}^{M} e_{p,m}\frac{\partial e_{p,m}}{\partial \omega_i} \qquad \textbf{III.28}$$

$$\nabla = J \times e \qquad \textbf{III.29}$$

la relation entre la matrice Jacobienne et la Hessienne peut être approximée de la sorte :

$$H = J^T \times J \qquad \textbf{III.30}$$

où : J^T représente la matrice Jacobienne transposée.

L'équation **III.26** devient ainsi :

$$\omega_{k+1} = \omega_k - (J_k{}^T \times J_k)^{-1} \times (J_k \times e_k) \qquad \textbf{III.31}$$

l'avantage de l'algorithme de Gauss-Newton (GN) sur la méthode de la méthode de Newton est que dans le premier cité, on aura pas à calculer les dérivées d'ordre 2 de la fonction d'erreur total e, et ce en introduisant la matrice Jacobienne. Cependant l'algorithme GN est confronté au problème de convergence pour l'optimisation de l'errer e. Mathématiquement parlant, le problème se situera dans **III.30**, la matrice H n'étant pas inversible, C'est dans ce contexte que l'algorithme de LM a fait surface.

I-4-d) - L'algorithme de Levenberg-Marquardt :

L'algorithme de LM n'est autre qu'une amélioration de l'algorithme de GN, il combine entre la rapidité de l'algorithme de Gauss-Newton et la stabilité de l'algorithme de descente du gradient, pour cela un terme est ajouté à la matrice $J^T \times J$ afin de veiller à ce qu'elle soit toujours inversible, la matrice Hessienne approximative deviendra ainsi comme suit :

$$H = J^T \times J + \mu \times I \qquad \textbf{III.32}$$

μ : Représente le coefficient de combinaison (ou d'ajustement), $\mu > 0$.

I : Représente la matrice Identité de taille N×N (N : nombre total de poids).

Désormais on utilisera la formule suivante pour l'ajustement des poids :

$$\omega_{k+1} = \omega_k - (J_k{}^T \times J_k + \mu \times I)^{-1} \times J_k \times e_k \qquad \textbf{III.33}$$

A noter que l'algorithme de LM commute entre l'algorithme de GN et celui de DG, ainsi pour une valeur de μ très faible, l'équation **III.33** se rapprochera de l'équation **III.31**, tandis que pour une valeur haute de μ, elle se rapprochera plutôt à l'équation **III.10** avec :

$$\eta = \frac{1}{\mu} \qquad \textbf{III.34}$$

I-4) - L'implémentation :

L'implémentation de l'algorithme de LM est beaucoup plus difficile et beaucoup plus délicat par rapport à celui de DG, il est primordial de résoudre deux problèmes : le premier étant le calcul de la matrice Jacobienne, et le deuxième consiste en l'organisation de l'apprentissage et du calcul des nouveaux poids.

I-4-a) - La calcul de matrice Jacobienne :

Le calcul de J peut être de même que celui de la rétropropagation traditionnelle mais il y'a quelques différence, la première c'est que dans l'algorithme de DG, un seul processus de calcul est requis pour chaque couche, tandis qu'avec l'algorithme de LM, le processus est répété pour chaque sortie afin d'obtenir des rangées consécutives de$J$.

Comme pour l'algorithme précédent, la matrice Jacobienne sera calculée en utilisant le théorème des dérivées composées dont les composants sont directement donnés comme suit :

$$\frac{\partial e_{p,m}}{\partial \omega_{j,i}} = -F'_{m,j} \times S_j \times y_{j,i} \qquad \textbf{III.35}$$

où :

$F'_{m,j}$: Représente la dérivée de la fonction non linéaire entre j et m

S_j : Représente la pente de la sortie du neurone j (dérivée de la fonction d'activation).

$y_{j,i}$: Représente l'entée du neurone i auquel le poids $\omega_{j,i}$ est lié.

I-4-b) - L'apprentissage suivant l'algorithme de LM :

Selon la règle de mise à jour de l'équation **III.33**, si l'erreur globale diminue cela implique une convergence de l'algorithme, on divisera alors le coefficient μ par un entier α (généralement on choisi $\alpha = 10$) pour réduire l'influence de la DG et on garde les nouveaux poids pour la prochaine itération, par contre si l'erreur augmente, il sera nécessaire de suivre la DG ce qui nous poussera à multiplier μ par α, les poids ne seront pas pris en considération et l'itération est refaite, la figure ci-contre représente l'organigramme de l'algorithme de LM.

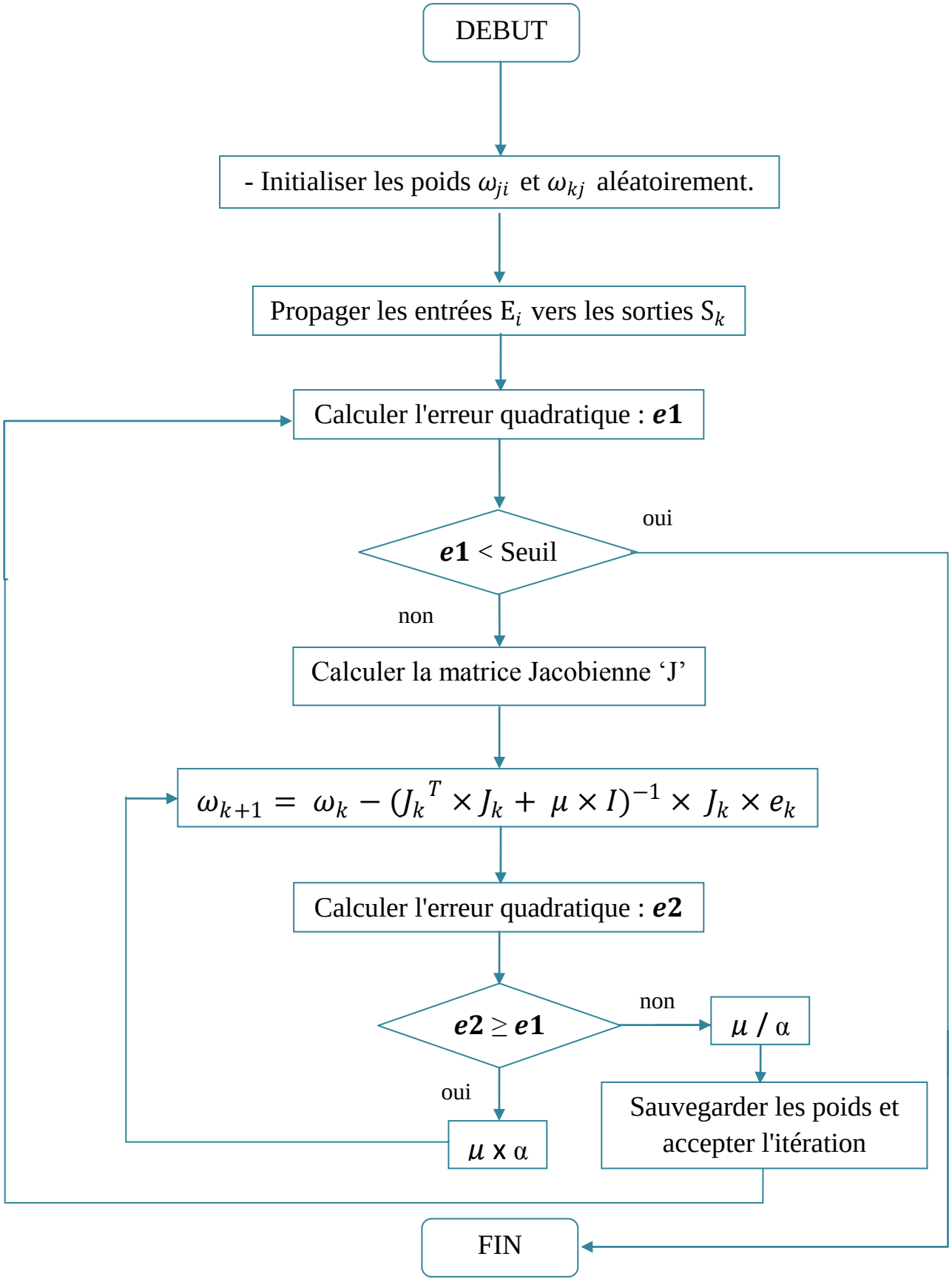


Figure III.21 : Organigramme de l'algorithme de Levenberg-Marquardt.

I-5) - Résultats et commentaires :

Afin de compenser la non-linéarité, on utilisera une modélisation inverse de la thermistance avec, comme entrée de l'ANN, une tension délivrée par un diviseur de tension, quant' à la sortie on aura une l'estimation de la température appliquée. Ce problème est identique à celui du canal d'égalisation dans le récepteur de communication pour annuler les effets négatifs de canal pour les données transmises, en effet la lecture numérique directe de la température appliquée est obtenue en cascade du modèle inverse de la thermistance de manière à compenser sa non-linéarité, La figure ci-contre résume d'une façon générale le contexte de l'application effectuée.

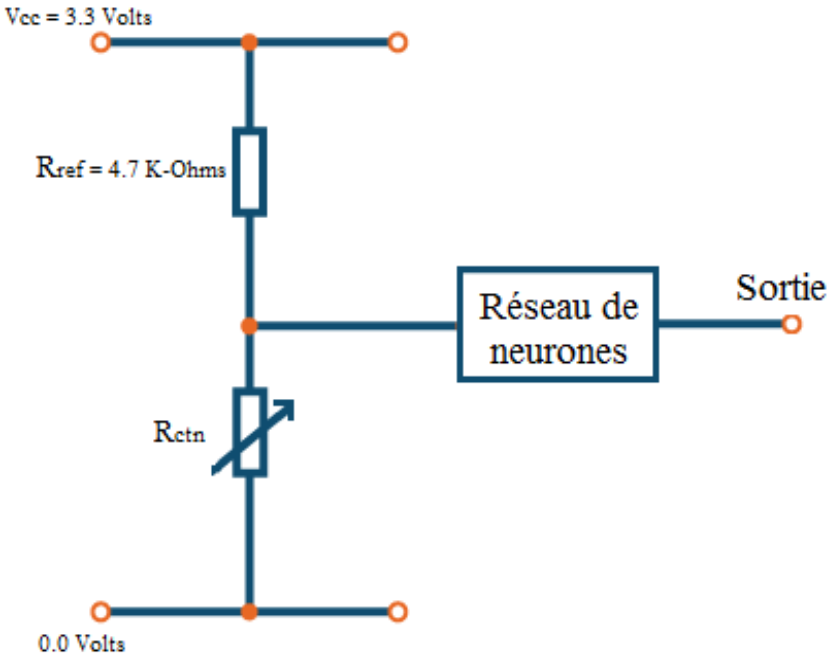


Figure III.22 : Organigramme de modélisation et de linéarisation de la CTN.

I-5-a) – Approximation de la fonction non-linéaire de la CTN:

Approximer une fonction peut être interprétée un peu à la façon des séries de Fourier qui utilisent des sinus et cosinus, seule différence c'est qu'avec les ANN, l'approximation de n'importe quelle fonction se fait par une combinaison de fonctions linéaires et/ou de sigmoïdes. Notre réseau contiendra une seule entrée qui sera la tension issue du diviseur de tension, la sortie sera obtenue après avoir effectué la conversion des valeurs de la résistance en température suivant l'équation **III.17.**

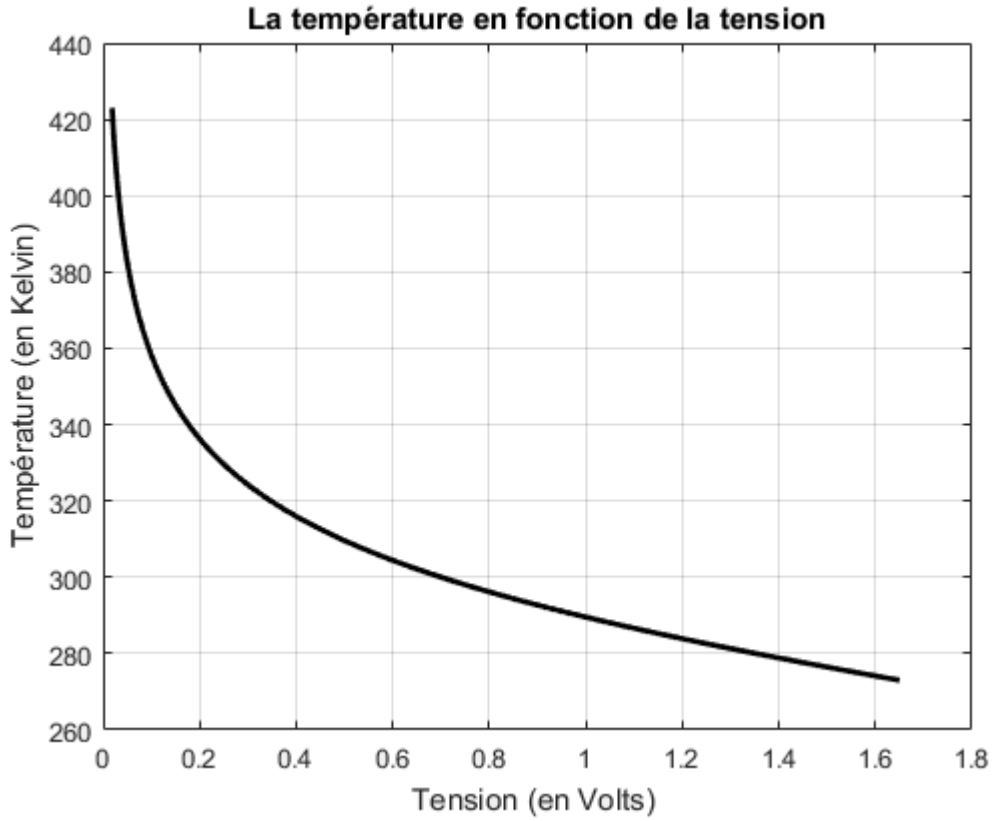


Figure III.23 : L'évolution de la température en fonction de la tension.

I-5-b) – Comparaison des deux algorithmes

Après avoir essayé à maintes fois différentes architectures pour modéliser la CTN, on a obtenu trois réponses de trois types de réseaux de neurones, le premier est celui de l'algorithme de DG à une seule couche, le deuxième contient deux couches, quant' au troisième, il est à base de l'algorithme de Levenberg-Marquardt ; le tableau ci-dessous révèles les performances acquises de l'apprentissage des modèles sélectionnées.

N°	Algorithme d'apprentissage	Nombre de couches	Architecture du réseau	EQM
1	Descente du gradient	1	1 - 4 - 1	24.502×10^{-3}
2	Descente du gradient	2	1 - 3 – 5 - 1	7.515×10^{-3}
3	Levenberg-Marquardt	1	1 - 4 - 1	3.060×10^{-6}

Tableau III.4 : Erreurs en fonction des architectures.

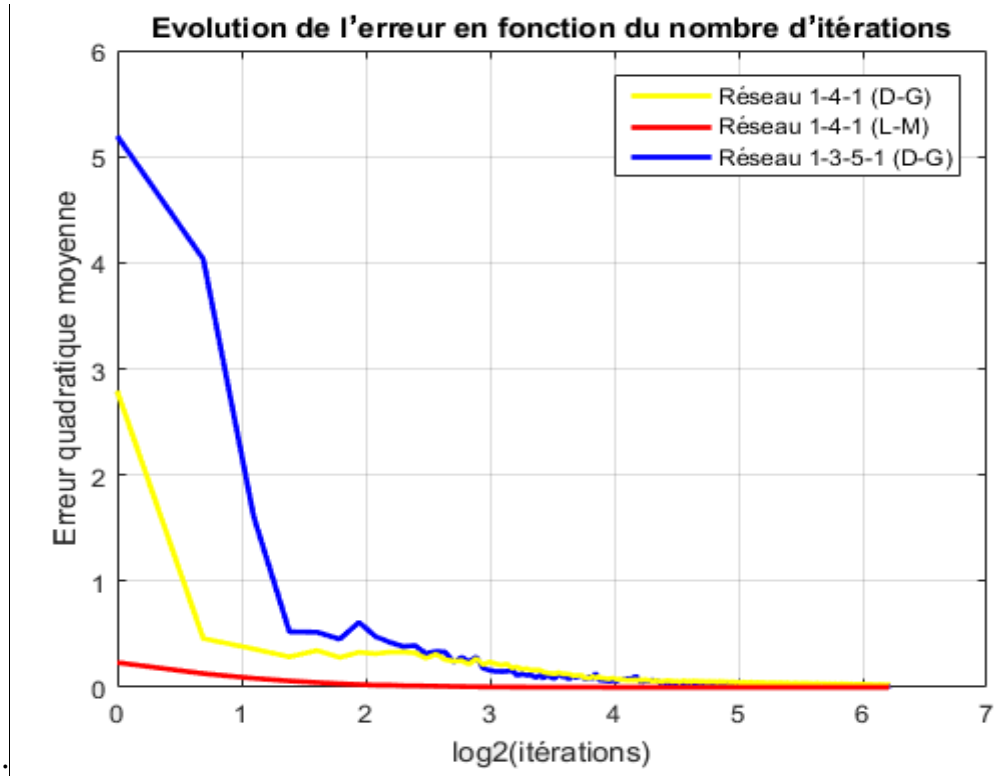


Figure III.24 : L'évolution des erreurs en fonction des itérations.

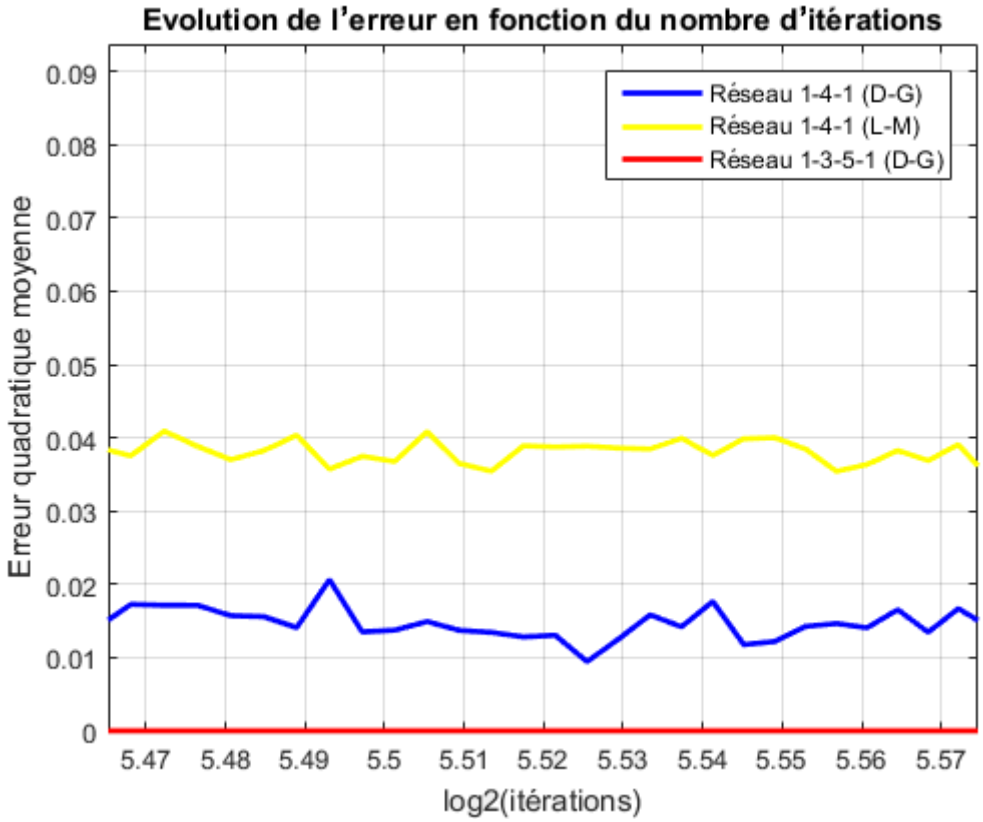


Figure III.24 : L'évolution des erreurs après 250 itérations.

L'efficacité de l'algorithme de Levenberg-Marquardt se fait grandement remarqué, l'erreur est beaucoup plus atténuée par rapport aux deux autres, cette performance fait de lui l'un des meilleurs algorithmes qui existent. La fonction non-linéaire de la CTN a été ainsi approximé avec succès, le modèle choisi est représenté dans la figure ci-dessous suivi par les vecteurs poids (entrée-couche cachée et couche cachée-sortie).

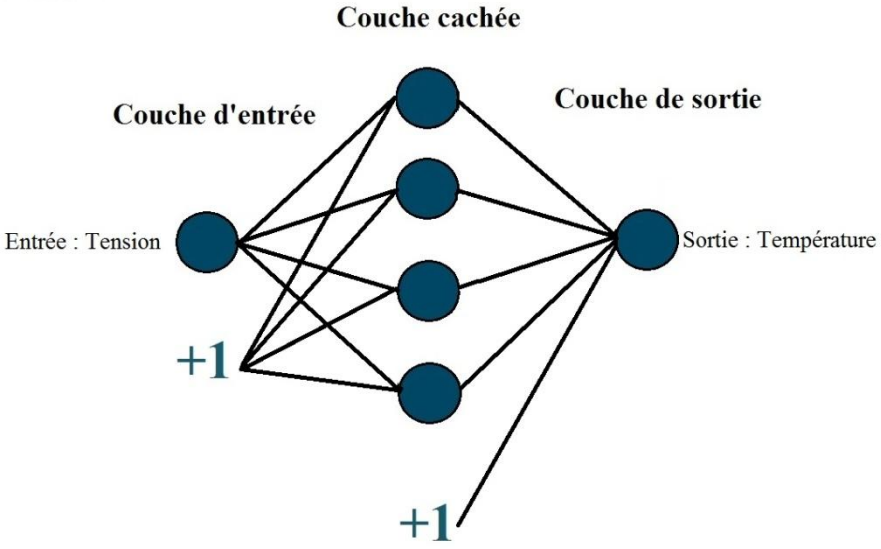


Figure III.25 : L'évolution des erreurs après 250 itérations.

* Poids Entrée - Couche cachée :

$$\omega_{ji} = \begin{bmatrix} -28.2946 & 30.2621 \\ 148.2745 & -3.4095 \\ -2.9322 & 3.5228 \\ 10.5234 & 1.3966 \end{bmatrix}$$

* Poids Couche cachée - Sorties :

$$\omega_{kj} = [-11.3908 \quad 1.7833 \quad -5.9741 \quad 9.2971 \quad 6.2627]$$

Conclusion :

Dans ce chapitre, on a conçu deux modèles neuronales distincts, l'un nous a permis de diagnostiquer deux maladies du système urinaire à savoir l'inflammation aigue et la néphrite aigue ; les poids issus de l'apprentissage ont été utilisés pour classifier nos échantillons en introduisant les symptômes par voie d'un clavier, les résultats obtenues ont été plutôt satisfaisants, l'autre a fait objet d'une modélisation d'une thermistance CTN dont le but

était de compenser la non-linéarité de sa sortie grâce au modèle inverse. Certes, la perfection n'est pas atteinte, l'expérience n'étant pas été faite dans un milieu adéquat et approprié, cependant et après quelques comparaisons avec le capteur de température LM35 dont la sensibilité est corrigé, les résultats s'étaient avérés vraiment acceptable voire, bon.

Conclusion Générale

Conclusion Générale

Ce mémoire a été consacrée au développement d'un des domaines de l'intelligence artificielle à savoir les réseaux de neurones ; après avoir décrit minutieusement leur principe de base, leurs utilités, et leurs domaines d'applications, on s'est mis à décortiquer deux fonctions d'apprentissage de ces ANN, le premier intègre l'algorithme de descente du gradient et est très utilisé pour classifier des données non pas pour sa rapidité d'exécution mais pour sa souplesse et sa mise en œuvre qui reste plutôt abordable, son principe se base sur la minimisation de la fonction d'erreur par voie de dérivée d'ordre 1 néanmoins, on s'y confronté a pas mal de problèmes quant' à la résolution de certains problèmes d'optimisation, on parle surtout des fonctions non linéaires, ce manque de fiabilité est dû à la sensibilité de cet algorithme envers les minimums locaux, et ce malgré l'introduction de quelques paramètres de vitesse et de convergence ; ce qui nous a amené à étudier une deuxième fonction d'apprentissage, l'algorithme de Levenberg-Marquardt qui reste incontournable et indétrônable d'après la plupart des spécialistes, il offre des solutions bien meilleures en vu de son utilisation des dérivées d'ordre 2 de la fonction d'erreur globale, seul bémol c'est qu'il exige trop de capacité et de mémoire au sein du matériel où il fonctionne ; de ce fait il est limité a des petits réseaux avec peu de variables.

Notre travail s'est portée à l'utilisation et à l'implémentation des deux fonctions citées auparavant dans une carte d'interface Arduino DUE dans le but d'un fonctionnement en temps réel ; le premier algorithme nous a permis de procéder à un apprentissage supervisée suivi d'une classification de deux maladies du système urinaire, le réseau se constitue de six entrées à intégrer par un spécialiste permettant largement de diagnostiquer si un patient souffre ou non d'une des deux maladies, ces paramètres sont la température du patient, présence de nausées, douleurs lombaires, besoin continu d'uriner, douleurs lors de la miction et enfin brûleurs de l'uretère (plus des démangeaisons et/ou gonflement de l'urètre). Et sachant qu'il était difficile d'avoir une architecture dès les premiers tests, on a pensé à intégrer un clavier permettant de piloter le réseau, ainsi et au lieu de modifier constamment notre programme, le choix se fera manuellement à chaque cycle d'apprentissage. Revenons maintenant à la raison du choix de la DUE, celui-ci est porté sur le fait que l'algorithme de Levenberg-Marquardt ne peut être implanté que sur des cartes puissantes en mémoire, et la DUE offre des caractéristiques hors normes par rapport aux autres cartes Arduino, se dotant d'un processeur ARM Cortex M3, il nous a permis de modéliser un capteur de température CTN favorisant une compensation automatique d'une non-linéarité plus que gênante, en

insérant la sortie d'un diviseur de tension dans un modèle inverse d'un ANN, on pourra avoir une valeur de température linéarisée et ce en temps réel.

Comme perspectives de ce travail, il serait très judicieux de se proférer à des méthodes plus récentes et plus riches à la procédure de mise en œuvre de notre classification. Ainsi et au lieu de trimbaler le clavier cité auparavant, on pourra le remplacer par une application Android, une intégration de ce système dans un Smartphone ouvrira le porte à une multitude d'applications de ce genre. Profitant d'un point commun entre nos deux applications qui est la température, il serait aussi logique et opportun d'utiliser cette grandeur physique pour en faire un thermomètre électronique mesurant en temps réel la température du patient et envoyant ainsi les données au réseau qui effectuera une classification.

BIBLIOGRAPHIE

[1] MATTHEW SPARKES, "Deputy Head of Technology", TheTelegraph, 13 Jan 2015, http://www.telegraph.co.uk/technology/news/11342200/Top-scientists-call-for-caution-over-artificial-intelligence.html.

[2] FRANÇOIS DOMINIC LARAMEE, "Faut-il avoir peur de l'intelligence artificielle?", Mardi 05 Aout 2014, 15h58, http://branchez-vous.com/2014/08/05/faut-il-avoir-peur-de-lintelligence-artificielle.

[3] MARCO DORIGO, CHRISTIAN BLUM, "Ant colony optimization theory: A survey", Theoretical Computer Science, Volume 344, Issues 2–3, 17 November 2005, Pages 243–278.

[4] PETER BRASPENNING, FRANK THUIJSMAN, ANTONIUS JOZEF MARTHA MARIA WEIJTERS, "Artificial neural networks : an introduction to ANN theory and practice", Berlin Springer, cop. 1995.

[5] ANTHONY MOURAUD, "Approche distribuée pour la simulation événementielle de réseaux de neurones impulsionnels.", Université des Antilles et de la Guyane, le 25 Mai 2009.

[6] CLAUDE GOMEZ, "Le calcul numérique : pourquoi et comment ?", 16 juin 2009.

[7] http://www.ulb.ac.be/di/map/gbonte/calcul/math31_1_dia.pdf.

[8] G. VIJAYA, VINOD KUMAR & H. K. VERMA, "ANN-based QRS-complex analysis of ECG", Journal of Medical Engineering & Technology, Volume 22, pages 160-167, 1998.

[9] K .O. Gupta, P. N. Chatur, "ECG Signal Analysis and Classification using Data Mining and Artificial Neural Networks", International Journal of Emerging Technology and Advanced Engineering, (ISSN 2250-2459, Volume 2, Issue 1, January 2012) 2Head of Department, Computer Science and Engineering, Govt. College of Engineering, Amravati.

[10] DRAŽEN BEGIĆ, VESNA POPOVIĆ-KNAPIĆ, JASMINA GRUBIŠIN, BILJANA KOSANOVIĆ-RAJAČIĆ, IGOR FILIPČIĆ, IRMA TELAROVIĆ & MIRO JAKOVLJEVIĆ, "QUANTITATIVE ELECTROENCEPHALOGRAPHY IN SCHIZOPHRENIA AND DEPRESSION", Department of Psychiatry, University Hospital Center Zagreb, Zagreb, Croatia, Psychiatria Danubina, 2011; Vol. 23, No. 4, pp 355-362.

[11] ABDULHAMIT SUBASI, ERGUN ERÇELEBI, "Classification of EEG signals using neural network and logistic regression", Department of Electrical and Electronics Engineering, Kahramanmaras Sutcu Imam University,Turkey & Department of Electrical and Electronics Engineering, University of Gaziantep,Turkey; accepted 26 October 2004.

[12] Vaegae Naveen Kumar; Komanapalli Venkata Lakshmi Narayana ; Annepu Bhujangarao ; Samickannu Sankar, "Development of an ANN-Based Linearization Technique for the VCO Thermistor Circuit", School of Electronics Engineering, VIT University, Vellore, India, IEEE; Volume:15 , Issue: 2, P 886 - 894, 2015.

[13] Komanapalli Venkata Lakshmi Narayana; Vaegae Naveen Kumar, "Development of an Intelligent Temperature Transducer", School of Electronics Engineering, VIT University, Vellore, India, IEEE; Vol 16 , Issue: 12, P 4696 - 4703, 2016.

[14] Vaegae Naveen Kumar; Komanapalli Venkata Lakshmi Narayana, "Development of an ANN-Based Pressure Transducer", Sch. of Electr. Eng., VIT Univ., Vellore, India, IEEE; Vol 16 , Issue: 1, P 53-60, Jan 2016.

[15] G. Goavec-Mérou, J.-M Friedt, "Le microcontrôleur STM32 : un cœur ARM Cortex-M3", 28 février 2012.

[16] http://josich.over-blog.com/article-19490539.html

[17] Xavier ANTOINE , Pierre DREYFUSS et Yannick PRIVAT, " Introduction `a l'optimisation : aspects théoriques, numériques et algorithmes", Institut National Polytechnique de Lorraine (INPL), Ecole Nationale Supérieure d'Electricité et de Mécanique, Ecole Nationale Supérieure des Mines de Nancy, Département de Génie Industriel, et Institut Elie Cartan Nancy (IECN), Université Henri Poincaré Nancy 1,B.P. 239, F-54506 Vandoeuvre-lès-Nancy, France, (2006-2007).

[18] GEORGES KOEPFLER, " Optimisation et algorithmique", UFR DE MATHEMATIQUES ET INFORMATIQUE, MASTER 1 MATHEMATIQUES APPLIQUEES, http://www.math-info.univ-paris5.fr/~gk/OptiAlgo/OptiAlgo.pdf, 2014-2016

[19] CAMILLO COLGI, The neuron doctrine - theory and facts Nobel Lecture December 11, 1906, Pages 189–217.

[20] MO COSTANDI, The Discovery Of The Neuron, History of Neuroscience, Neuroscience, Tuesday, August 29, 2006

[21] WARREN S. MCCULLOCH, WALTER PITTS, A logical calculus of the ideas immanent in nervous activity, University of illinois, College of medicine, Department of psychiatry at the illinois neuropsychiatric institute, and the university of chicago, Bulletin of mathematical biophysics Volume 5, 1943.

[22] KIYOSHI KAWAGUCHI, "The McCulloch-Pitts Model of Neuron", 17/06/2000, http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node11.html.

[23] DONALD HEBB, The Organization of Behavior. Wiley, New York, 1949.

[24] MARVIN MINSKY, SEYMOUR PAPERT, Perceptrons, An Introduction to Computational GeometryM.I.T. Press, Cambridge, Mass., 1969.

[25] J.J. HOPFIELD, "Neural Networks and physics Systemswith emergent collective Computational Abilities.

[26] P.E. KELLER Commercial applications of artificial neural networks.

[27] Alp Mestan, Introduction aux Réseaux de Neurones Artificiels Feed Forward, http://alp.developpez.com/tutoriels/intelligence-artificielle/reseaux-de-neurones, 2008.

[28] J. M. Granado ; M. A. Vega ; R. Perez ; J. M. Sanchez ; J. A. Gomez "Using FPGAs to Implement Artificial Neural Networks", University of Extremadura. Department of Computer Science. Cáceres, Spain, 2016.

[29] Denis Giacona, "VHDL – Logique programmable", ENSISA, École Nationale Supérieure d'Ingénieur Sud Alsace.

[30] IDIR MELLAL; M. LAGHROUCHE, "Implémentation d'un réseau de neurones d'un micro capteur sur un FPGA", Université Mouloud Mammeri de Tizi-Ouzou, Faculté de Génie Electrique et de l'informatique, 2010.

[31] Mark ZWOLINSKI "Digital System Design with VHDL" ,Pearson,2nd édition 2004 p 206-209.

[32] HOWARD DEMUTH MARK BEALE, "Neural Network Toolbox For Use with MATLAB®", http://www.image.ece.ntua.gr/courses_static/nn/matlab/nnet.pdf.

[33] BRUNO BOUZY, "Descente de gradient", 5 octobre 2005, http://www.math-info.univ-paris5.fr/~bouzy/Doc/AA1/DescenteGradient.pdf.

[34] ANANTH RANGANATHAN , "The Levenberg-Marquardt Algorithm", 8th June 2004

[35] Marc Parizeau, " RESEAUX DE NEURONES" GIF-21140 et GIF-64326, Université Laval, 2006

[36] tp://ressources.univ-lemans.fr/AccesLibre/UM/Pedago/physique/02/electro/thermist.html

[37] Compensation of Sensors Nonlinearity with Neural Networks Nicholas J. Cotton and Bogdan M. Wilamowski Electrical and Computer Engineering Auburn University Auburn, AL 36849 United States

[38] http://thales.math.uqam.ca/~blondin/files/8map110/staylor.pdf.