UNIVERSITE DE TUNIS EL MANAR
FACULTE DES SCIENCES DE TUNIS

ECOLE DOCTORALE EN INFORMATIQUE

MÉMOIRE DE MASTÈRE

présenté en vue de l'obtention du
Diplôme de Mastère en Informatique

Par
Nabil CHOUBA

(Ingénieur en Informatique, FST - Tunis)

Réalisation d'un processeur neuronal dédié, à base de rétro-propagation du gradient

soutenu, devant le jury d'examen

MM. Rafik BRAHAM	***Président***
Rached TOURKI	***Membre***
Mohamed MOALLA	***Directeur du mémoire***
Khaled OUALI	***Co-Directeur du mémoire***
Walid MAROUFI	***STMicroelectronics , Invité***

Remerciements

Je tiens à exprimer mes remerciements à mes encadreurs, Monsieur Mohamed MOALLA, Professeur à la Faculté des Sciences, et Monsieur Khaled OUALI, Maître-Assistant à la Faculté des Sciences de Tunis, pour leurs conseils précieux, leur disponibilité permanente et leur aide pour m'initier à l'écriture scientifique.

Mes remerciements s'adressent également à :

- Monsieur Rafik BRAHAM, Professeur à l'Institut Supérieur d'Informatique et de Technologie de la Communication de Hammam Sousse, pour l'honneur qu'il me fait de présider le jury de soutenance de ce mémoire,

- Monsieur Rached TOURKI, Professeur à la Faculté des Sciences de Monastir, pour l'intérêt qu'il a accordé à ce travail et pour avoir accepté de participer au jury.

Que Monsieur Walid MAAROUFI, Chef de l'Equipe « Design for test » à ST-Microelectronics Tunisie, ainsi que mes amis qui ont été là pour m'aider à chaque fois que j'en avais besoin, puissent trouver ici l'expression de ma reconnaissance.

A mes parents,

A ma fiancé Jihed.

Index

Introduction …….. 1
Chapitre 1 : Techniques Neuronales …….. 3
I. Réseaux de neurones : présentation générale…….. 3
1. Unité de base (le neurone)……3
2. Topologie du réseau…….. 5
a. Architecture…….... 5
b. Connexions…… 6
II. Types d'Applications du Réseau……7
III Apprentissage……. 8
1. Familles d'apprentissages……..8
2. Méthodes d'apprentissage…… 9
3. Algorithme d'apprentissage par rétropropagation du gradient…… 10
a. Principe …….. 10
b. Règle de rétropropagation du gradient …….. 10
4. Extension : Apprentissage par les réseaux évolutifs……. 12
IV. Procédure de développement d'un réseau de neurones……13
1. la collecte des données…….. 13
2. l'analyse des données…… 13
3. le choix final de la famille d'apprentissage……...13
4. la séparation des bases de données……14
5. la mise en forme des données……14
6. le choix d'un réseau de neurones……...14
7. la validation……... 14
V. Apprentissage orienté vers l'intégration ……15

Chapitre 2 : présentation et analyse des implémentations matérielles des réseaux neuronaux ……. 16
I. Introduction…… 16
II. Implémentations de réseaux de neurones sur puces standard……17
III. Implémentations de réseaux de neurones à base d'ASIC……..18
IV. Implémentations de réseaux de neurones à base de FPGA……...19
V. Conclusion…… 20

Chapitre 3 : Implémentation matérielle……..21
I. Introduction……..21
II. Réalisation de la propagation……..22
1. Algorithme……23
2. Câblage de l'algorithme…… 23
3. Interface de l'IP NEURONA……25
a. Mode de lecture / écriture……26
b. Mode de propagation……...26
c. Mode de rétro-propagation…….. 27

4. Etude de cas……...27
5. Conception de l'IP NEURONA……31
a. Explorateur de la topologie du réseau……...31
b. Générateur d'adresse W (gen_adr_w)……33
c. Générateur d'adresse X (gen_adr_x)……...33
d. Unité de calcul……35
e. Fonction d'activation……...37
f. La machine d'états……... 38
III. Réalisation de la rétro-propagation……... 41
1. L'algorithme d'apprentissage……41
2. Traitement des étapes 1 et 2 ……. 42
3. Traitement de la sous-étape 4.3 ……...43
4. Traitement de la sous-étape 4.4 ……...44
5. A propos de l'exécution des calculs proprement dits……45
6. Enrichissements nécessaires au niveau des générateurs
d'adressas et de la machine d'état ……... 47
a. Enrichissements au niveau de l'explorateur de la topologie du réseau …….. 47
b. Enrichissements au niveau du générateur d'adresse W (gen_adr_w)……. 48
c. Enrichissements au niveau du Générateur d'adresse X (gen_adr_x) ……. 48
d. Enrichissements au niveau de la machine d'état …… 48
7. Etude de cas ……49
IV. Résultats de la synthèse ……...54
V. Vers des architecteurs parallèles ……... 57
1. Parallélisme dans les réseaux de neurones à couches …….. 57
a. Parallélisme intra-neurone ……..57
b. Parallélisme des données ……58
c. Parallélisme inter-couches ……..58
d. conclusion …….. 59
Conclusion……..60
Bibliographie ……61

Introduction

Les technologies de l'information doivent beaucoup aux travaux réalisés par Von Neumann et Alan Turing sur les architectures des premières machines en vue de créer un système capable d'imiter le comportement du cerveau humain. Ces deux chercheurs s'étaient pourtant inscrits dans une ancienne tradition qui consiste à imaginer que l'on puisse reproduire le fonctionnement du cerveau humain sans s'inspirer du modèle biologique. Alan Turing est précisément celui qui a ouvert cette piste de recherche, ayant été persuadé que l'intelligence était une question de «logiciel» plus qu'une question de «matériel ». Il n'en demeure pas moins que ce sont les investigations de ces deux chercheurs qui ont donné naissance aux ordinateurs de nos jours.

La vague suivante est celle des chercheurs en sciences cognitives, qui ont développé les modèles formels bio-inspirés : les "Réseaux de Neurones". Ces modèles surpassaient les limites du modèle de Von Neumann sur plusieurs aspects : tolérance aux erreurs, apprentissage, généralisation à partir d'exemples, parallélisme inné. Les modèles bio-inspirés et l'architecture de Von Neumann ont vu le jour à la même époque, mais les premiers ont vite été abandonnés car ils nécessitent de lourdes tâches de calcul, essentiellement dans la phase d'apprentissage.

L'évolution des performances des technologies des ordinateurs a permis, par la suite, de relancer les réseaux de neurones. Ceux-ci sont aujourd'hui utilisés dans plusieurs applications industrielles (reconnaissance de la parole, classification, estimation de fonction etc.). Mais, il reste toujours impossible de créer l'équivalent du cerveau humain. En effet, le cerveau compte 100 milliards de neurones et chaque neurone établit 10.000 connexions avec d'autres neurones. Il est clair que la technologie actuelle ne peut pas réaliser un système pareil avec un million de milliards d'interconnexions. La solution est de trouver des architectures dédiées qui réalisent un compromis entre les limites technologiques et la performance. Le travail présenté ici a visé à concevoir une architecture qui approche cette solution en dépit des limitations des technologies actuelles.

Différents modèles formels de réseaux de neurones existent. Le plus populaire et le plus utilisé par l'industrie est le "Perceptron multicouches" utilisant la rétro-propagation comme processus d'apprentissage. C'est ce modèle que nous avons adopté dans notre travail.

Le premier chapitre rappelle ce modèle formel des réseaux de neurones multicouches ainsi que quelques méthodes d'apprentissage et le flot standard de développement d'une application neuronale. A la fin du chapitre, nous évoquons le problème de l'influence de la limitation de la précision arithmétique sur le comportement des algorithmes implantés.

Le deuxième chapitre présente l'état de l'art en matière d'implémentation du traitement neuronal sur des architectures dédiées (telles que NeuroChip) ou semi dédiées (telles que Neurocomputer).

Le troisième chapitre détaille notre étude d'une nouvelle approche de traitement neuronal basée sur un processeur dédié (spécifique) que nous avons conçu et baptisé NEURONA. Ce processeur cible une architecture générique et dynamiquement reconfigurable, permettant l'implémentation directe d'applications sur un réseau de neurones multicouches capable de supporter toutes les méthodes d'apprentissage à base de rétro-propagation. Il est proposé sous forme d'une propriété intellectuelle (IP) en VHDL synthétisable. La généricité donne la possibilité de fixer, lors de la synthèse, la taille (précision) du traitement arithmétique ainsi que le nombre maximum de couches. Tandis que la reconfiguration est assurée par le programme neuronal et concerne la topologie du réseau caractérisée par le nombre de couches, le nombre de neurones par couche, les poids de connectivité entre neurones et la morphologie de la fonction d'activation du neurone.

La conclusion synthétise les points forts de l'architecture donnée au processeur NEURONA et propose des voies d'approfondissement en perspectives de prolongement du travail réalisé dans le cadre de ce stage de Mastère.

Chapitre 1 : Techniques Neuronales

I. Réseaux de neurones : présentation générale

Le dictionnaire "Petit Larousse" définit un neurone comme suit : « Cellule nerveuse telle que à son corps cellulaire aboutissent des fibres fines et ramifiées….. un seul prolongement myélinisé (l'axone) en repart. L'influx nerveux la parcourt en sens unique, allant des dendrites vers l'axone ». Le réseau de neurones se réalise par la connexion des axones des cellules aux dendrites des cellules voisines.

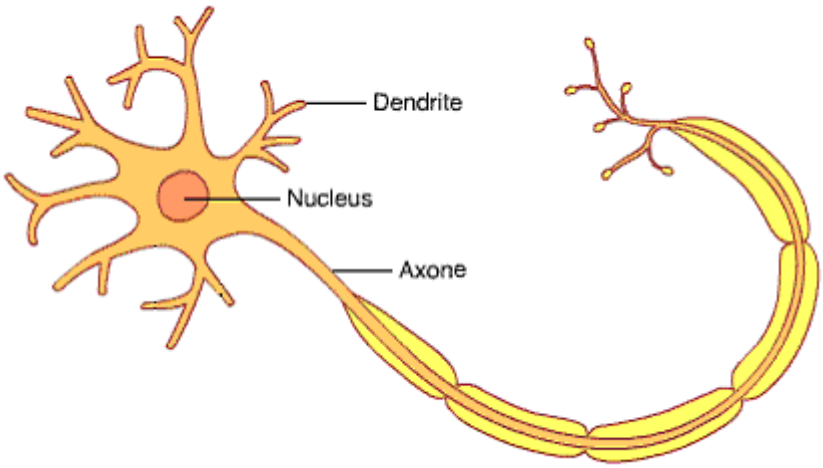


Figure 1.0 Cellule nerveuse

Les réseaux de neurones artificiels sont des réseaux dont l'architecture est inspirée de celle des réseaux de neurones naturels. Leur modélisation revient à décrire le modèle du neurone (unité de base) et le modèle des connexions entre les neurones.

1. Unité de base (le neurone)

Il existe un grand nombre de modèles de neurones. Les plus utilisés sont basés sur le modèle développé par McCulloch & Pitts [MACCULLOCH et al, 43] :

- Le neurone se présente comme une cellule possédant plusieurs liens d'entrée dits "liens synaptiques" et un lien de sortie (figure 1.1).

- Le lien de sortie d'une cellule peut être connecté à un ou plusieurs liens d'entrée

d'autres cellules ; il est porteur d'une valeur qu'il transmet à ces liens d'entrée, laquelle valeur est déterminée à partir des valeurs des entrées propres de sa cellule.

- Chaque lien d'entrée i d'une cellule porte un "poids synaptique" w_i. La cellule est dotée de deux opérateurs :

• un opérateur de sommation qui élabore un "potentiel post-synaptique" **p** égal à la somme pondérée des entrées de la cellule : $p= \sum_i(w_i.x_i)$, avec w_i le poids synaptique et x_i la valeur de l'entrée (état de la sortie du neurone connecté à cette entrée).

• un opérateur de décision qui calcule l'état de la sortie **s** du neurone en fonction de son potentiel **p ;** cet opérateur est appelé "fonction d'activation" ($s= F(p)$).

Ainsi, le calcul de l'état de sortie du neurone est obtenu en calculant le potentiel post-synaptique **p** puis en appliquant à **p** l'opérateur de décision. Le calcul est appelé "mise à jour du neurone".

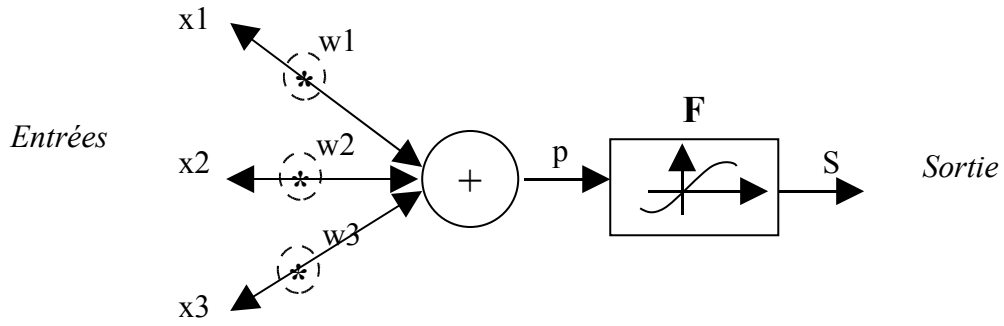
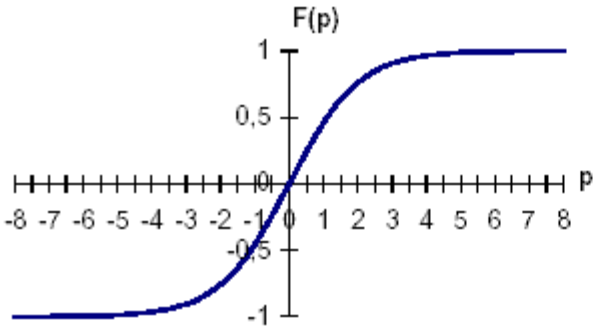


Figure 1-1 : Exemple de neurone avec 3 entrées

Dans le cas du modèle de McCulloch & Pitts, l'opérateur de décision est une fonction à seuil. Le neurone a deux états possibles "0" ou "1" déterminés en fonction du potentiel post-synaptique **p** et du seuil (θ). Ce genre de neurone est appelé "neurone binaire".

Comme l'utilisation d'une fonction non dérivable (telle que la fonction à seuil) présente des inconvénients lors de l'apprentissage, on fait plutôt appel à des fonctions croissantes et dérivables. L'état du neurone est alors multi évalué. Pour effectuer une classification, on utilise souvent une fonction de type « sigmoïde », par exemple une tangente hyperbolique dont la sortie est bornée entre -1 et 1 (figure 1-2).



$$F(p) = \frac{(e^{\alpha p} - 1)}{(e^{\alpha p} + 1)} = th(\frac{\alpha p}{2})$$

p: potentiel post-synaptique

F(p): fonction d'activation

α : pente de la courbe

Figure 1-2 : Fonction d'activation : tangente hyperbolique (th)

Dans un réseau, on distingue trois types de neurones :

- les neurones d'entrée, aussi appelés cellules perceptives du fait de leur propriété à acquérir des données dont la provenance est en dehors du réseau,
- les neurones de sortie, qui définissent la sortie du réseau,
- les neurones cachés, qui n'ont de relation entrée/sortie qu'avec les neurones du réseau.

2. Topologie du réseau

La topologie d'un réseau de neurones est définie par son architecture et la nature de ses connexions.

a. Architecture

La plupart des réseaux de neurones ont une topologie définie sous forme de couches. Il existe quelques exceptions comme, par exemple, lorsque le réseau n'est pas explicitement défini sur plusieurs couches (cas de certaines mémoires associatives neuronales) : l'architecture du réseau est décrite par le nombre de couches et le nombre de neurones dans chaque couche.

De façon similaire aux neurones, on distingue trois types de couches (figure 1-3) :

- la couche d'entrée constituée de l'ensemble des cellules perceptives du réseau,
- la couche de sortie constituée de l'ensemble des neurones de sortie,
- les couches cachées constituées par les couches intermédiaires.

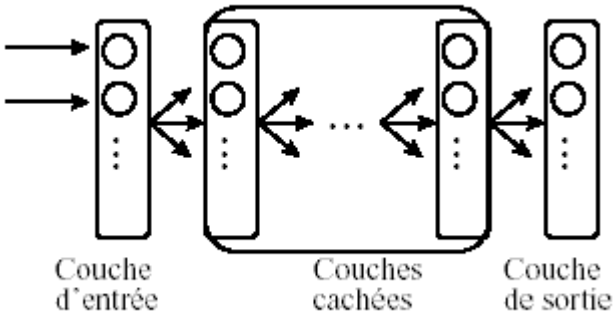


Figure 1-3: Réseau de neurones à couches

Dans un réseau multicouche, les différentes couches sont généralement ordonnées et indexées dans le sens croissant de la couche d'entrée vers la couche de sortie.

b. Connexions

Le modèle de connexion définit la manière selon laquelle sont interconnectés les neurones d'un réseau. Pour les réseaux multicouches, on distingue différents types de connexions : les connexions inter-couches (connexions entre les sorties des neurones d'une couche et les entrées des neurones de la couche qui suit immédiatement), les connexions supra-couche (connexions des neurones de couches non adjacentes), les connexions intra-couches (connexions entre neurones d'une même couche) et l'auto-connexion (la sortie d'un neurone est bouclée sur l'une de ses entrées).

On parle aussi de connexions "directes" et de connexions "récurrentes" (figure 1-4). Les connexions directes sont celles qui sont dirigées d'une couche d'indice inférieur vers une couche d'indice supérieur. Les connexions sont dites récurrentes lorsque des sorties de neurones d'une couche sont connectées aux entrées d'une couche d'indice inférieur.

Figure 1-4 : Connexions directes (a) et récurrentes (b)

Par ailleurs, entre deux couches, les connexions peuvent être partielles ou totales (figure 1-5). Les connexions sont totales si la sortie de chaque neurone d'une couche i est connectée à tous les neurones de la couche i+1. Elle sont partielles sinon.

L'utilisation de connexions partielles permet de regrouper certaines zones du réseau pour effectuer une fonction spécifique.

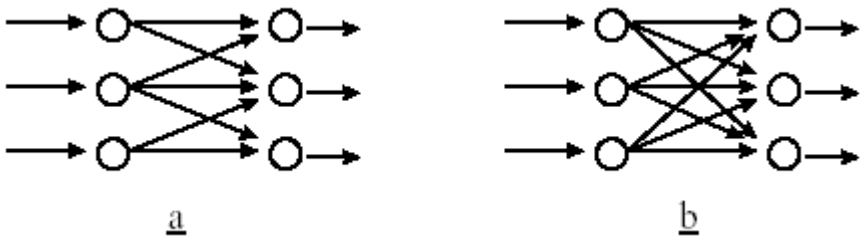


Figure 1-5 : Connexions partielles (a) et totales (b)

Les réseaux que nous ciblons dans le présent travail sont précisément des réseaux multicouches à connexions totales et ne pouvant admettre des connexions récurrentes qu'entre la couche de sortie et la couche d'entrée. Notons qu'il existe d'autres types de réseaux, optimisés pour des tâches particulières. Citons, par exemple, le réseau de Hopfield [HOPFIELD, 82] : un réseau récurrent totalement connecté qui, de par son aptitude à converger vers des états stables (appelés "attracteurs"), est souvent utilisé pour des tâches de mémoire associative.

II. Types d'Applications du Réseau

Les réseaux multicouches peuvent être utilisés pour implanter plusieurs types de tâches telles que la reconnaissance des formes, la classification, la transformation des données (exemple : compression), la prédiction (exemple : prédiction de séries temporelles), la commande de procédés et l'approximation des fonctions. On distingue deux types principaux de réseaux selon le type des sorties fournies et le comportement recherché :

- ***Réseaux pour l'Approximation de Fonctions*** : Il s'agit de réseaux dont la dernière couche est constituée par un seul neurone donnant une sortie continue. Ils sont employés pour l'approximation exacte (*interpolation*) ou pour l'approximation approchée d'une fonction représentée par des données d'apprentissage [Chentouf, 97]. Ce type de réseau est capable d'apprendre une fonction de transformation (ou d'association) des valeurs d'entrée vers la valeur correspondant en sortie ; l'apprentissage s'effectue par détermination des poids synaptiques de chaque neurone du réseau. Une fois l'apprentissage effectué, cette fonction permet alors de prédire la valeur de la sortie étant données les valeurs des entrées. On appelle ce type de problème, un problème de *régression* [Bishop, 97]. On trouve dans [Hecht-Nielsen,

89] de nombreuses démonstrations formelles de la capacité à approximer des fonctions continues de R^n ver R par un réseau de neurones. Cependant, à part la démonstration de l'existence de tels réseaux, il n'est rien dit sur la topologie exacte de ces réseaux ni sur la manière (méthode) dont ils peuvent être construits.

- ***Réseaux pour la Classification*** : Ce type de réseau sert à attribuer à chaque configuration de valeurs d'entrée une parmi un ensemble de classes. La classification est un cas particulier de l'approximation de fonctions où la valeur de sortie est discrète et appartient à un ensemble limité de classes. Dans le cas de l'apprentissage supervisé, cet ensemble de classes est prédéfini. Un réseau adapté à la classification doit avoir des sorties discrètes ou implémenter des méthodes de discrétisation des sorties.

III Apprentissage

1. Familles d'apprentissages

Dans un réseau de neurones, l'information qui régit son fonctionnement est codée par les poids synaptiques.
L'apprentissage est réalisé par des algorithmes de calcul dont le but est d'adapter ces poids de façon à obtenir un comportement désiré face à des stimuli présentés à l'entrée du réseau. Une fois l'apprentissage terminé, les poids ne sont plus modifiés. On distingue trois familles d'apprentissages :

- *Apprentissage Supervisé* : On dispose d'un comportement de référence précis que l'on désire faire apprendre au réseau. L'apprentissage doit mesurer l'écart entre le comportement du réseau et le comportement de référence et ajuster les poids synaptiques du réseau de façon à réduire cet écart. Il utilise pour cela des connaissances empiriques, habituellement représentées par des ensembles.

- *Apprentissage Semi-Supervisé* : On ne dispose que d'une ou de plusieurs indications synthétiques sur le comportement final désiré, sans exemples précis de relations entrées-sorties. C'est le cas, par exemple, si l'on sait tout au plus juger quand un comportement est ("bon" ou "mauvais"). Le réseau doit alors apprendre seul des relations entrées-sorties qui

permettent d'approcher le comportement désiré. L'apprentissage semi-supervisé est aussi appelé apprentissage par renforcement (*reinforcement learning*).

- *Apprentissage non-supervisé* : on ne dispose ici d'aucune information préalable d'appréciation du comportement mais en réagissant à ces stimuli, l'algorithme d'apprentissage va réajuster les paramètres (poids synaptiques) du réseau jusqu'à stabilisation. Ainsi, à la fin de l'apprentissage, le réseau aurait développé une habilité à former des représentations internes des stimuli de l'environnement permettant d'encoder les caractéristiques de ceux-ci et, par conséquent, d'identifier automatiquement des classes de stimuli similaires.

2. Méthodes d'apprentissage

L'apprentissage d'un ensemble de données stimuli peut être réalisé de différentes façons, selon la manière dont le réseau est alimenté par les données :

- *Apprentissage par Paquets (batch-learning)* : l'ensemble des données d'apprentissage est présenté au réseau plusieurs fois de façon à optimiser les poids du réseau et minimiser l'erreur en sortie. Chaque présentation de l'ensemble complet de données d'apprentissage est appelée une époque (*epoch*).

L'algorithme d'apprentissage réduit petit à petit l'erreur de sortie à chaque représentation des données et l'état du réseau doit converger vers la solution du problème. On peut aussi manipuler l'ordre des exemples dans le paquet, ce qui peut avoir des conséquences sur l'évolution de l'apprentissage ; on parle alors d'apprentissage actif. Cette méthode d'apprentissage par paquets constitue l'un des types d'apprentissage les plus utilisés. A la fin de l'apprentissage, les poids synaptique sont fixés définitivement.

- *Apprentissage Continu* : l'algorithme d'apprentissage reste continuellement ouvert à de nouveaux exemples qui lui arrivent (*continuous on-line learning*). Cette méthode est aussi appelée apprentissage incrémental (par rapport aux données). Un des principaux problèmes de l'apprentissage continu est la difficulté de trouver un bon compromis entre adaptation et stabilité : un trop d'adaptation peut amener le réseau à "oublier" des données déjà apprises, il s'écarte des valeurs apprises pour ces données. D'une autre côté, une stabilité avancée peut rendre le réseau incapable d'apprendre (de s'adapter à) de nouvelles entrées.

3. Algorithme d'apprentissage par rétropropagation du gradient

a. Principe :

L'étape d'apprentissage consiste d'abord à présenter l'exemple à apprendre au réseau, puis à calculer l'écart entre la sortie du réseau et la valeur désirée. Vient ensuite le rôle de l'algorithme de rétropropagation pour calculer et transmettre (rétro-propager) l'erreur totale à chaque neurone du réseau afin que l'algorithme utilise ces parts d'erreurs pour corriger les poids synaptiques de chaque neurone.

Il faut itérer l'ensemble des exemples à apprendre jusqu'à aboutir à un taux total d'erreur limité.

b. Règle de rétropropagation du gradient

La règle de rétropropagation du gradient de l'erreur (*delta rule* ou backprobagation) [Widrow, 90] est l'une des règles les plus utilisées pour l'apprentissage de réseaux de neurones. Elle est utilisée, entre autres, pour les réseaux de neurone multicouche.

L'objectif de cet algorithme est de minimiser une fonction de coût "E". L'équation (III-1) exprime cette fonction de coût à partir de l'erreur quadratique, pour un couple entrées-sorties, avec S_k la sortie du neurone d'indice k de la couche de sortie du réseau et d_k la valeur de sortie désirée pour ce neurone.

$$E = \sum_k (d_k - S_k)^2 \qquad \text{(III-1)}$$

L'apprentissage comporte une première phase de calcul dans le sens direct (neurones de couche d'entrée vers neurones de la couche de sortie) où chaque neurone effectue la somme pondérée de ses entrées et applique ensuite la fonction d'activation f pour obtenir la mise à jour de la sortie. L'équation (III-2) correspond à cette mise à jour avec p_i le potentiel post-synaptique du neurone i, x_j l'état du neurone j de la couche précédente et w_{ij} le poids de la connexion entre les deux neurones.

$$p_i = \sum_{j=0}^{n} w_{ij} x_j \quad , \quad S_i = f(p_i) \qquad \text{(III-2)}$$

(La règle de rétropropagation du gradient exige que la fonction f d'activation soit dérivable.)
Cette phase, dite de propagation, permet de calculer les sorties du réseau en fonction des entrées.

L'algorithme de rétropropagation consiste à effectuer une descente de gradient sur le critère "E". Le gradient de E est calculé pour tous les poids de la manière suivante :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial p_i}\frac{\partial p_i}{\partial w_{ij}} = \frac{\partial E}{\partial p_i}x_j \qquad \text{(III-3)}$$

On distingue alors deux cas, selon que le neurone fait partie d'une couche de sortie ou non.

- Dans le cas de la couche de sortie, le gradient attaché aux neurones de sortie est alors obtenu par l'équation (III-4) :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial p_i}x_j = -C_i x_j$$
$$C_i = -\frac{\partial E}{\partial p_i} = -\frac{\partial(\sum_k (d_k - S_k)^2)}{\partial p_i} = 2.(d_i - S_i).f'(p_i) \qquad \text{(III-4)}$$

car seul s_i dépend de p_i , ($s_i = f(p_i)$)

- Pour les neurones des couches cachées et de la couche d'entrée, le calcul des gradients s'effectue par rétropropagation depuis la couche de sortie jusqu'à la couche d'entrée. Ainsi, si i est l'indice du neurone dans la couche considérée et n est le nombre de neurones de la couche suivante alors l'expression du gradient est obtenue comme indiqué dans l'équation (III-5) :

$$C_i = -\frac{\partial E}{\partial p_i} = -\sum_{k=0}^{n}\frac{\partial E}{\partial p_k}\frac{\partial p_k}{\partial p_i} = \sum_{k=0}^{n} C_k\frac{\partial p_k}{\partial p_i} = \sum_{k=0}^{n} C_k\frac{\partial p_k}{\partial S_i}\frac{\partial S_i}{\partial p_i}$$
$$C_i = f'(p_i)\sum_{k=0}^{n} w_{ki}C_k \qquad \text{(III-5)}$$

avec C_k le gradient du neurone k de la couche suivante.

La modification des poids est obtenue suivant l'équation (III-6) où α est une valeur comprise entre 0 et 1 qui représente le pas de déplacement vers la solution désirée.

$$w_{ij}^{t+1} = w_{ij}^{t} + \alpha .C_i S_j \qquad \text{(III-6)}$$

Précisons, enfin, que la phase d'apprentissage est souvent arrêtée lorsque l'erreur calculée sur l'ensemble de la base d'apprentissage est inférieure au seuil déterminé par l'utilisateur.

Dans la méthode du " gradient stochastique", la modification des poids est effectuée à chaque exemple. Par contre, dans la variante de l'"Algorithme du gradient total", les exemples de la

base d'apprentissage sont présentés successivement au réseau. Les gradients sont accumulés au fur et à mesure et la modification des poids n'interviennent qu'après présentation de tous les exemples.

4. Extension : Apprentissage par les réseaux évolutifs

Comme nous l'avons rappelé précédemment, Hecht-Nielsen à démontré dans [Hecht-Nielsen, 89] l'existante, pour toute fonction continu sur l'espace des réels, d'un réseau de neurones approximateur de cette fonction. Il reste à trouver la méthode qui permet de construire l'architecteur de ce réseau et de répondre aux questions suivantes :

- Combien de neurones, doit contenir chaque couche ?
- Quelles sont les connexions qui doivent être établies entre les neurones des différentes couches ?

Il n'existe actuellement aucune méthode complète qui permet de répondre de manière théorique à ces questions. Pourtant, les réponses à ces questions sont fondamentales, le nombre de neurones et de connexions d'un réseau étant des facteurs déterminants de la capacité d'apprentissage puis de généralisation. Il est donc important de trouver les politiques de détermination de la topologie répondant au problème posé.

Des extensions des algorithmes classiques de rétropropagation dans les réseaux multicouches ont donc été proposées avec l'objectif d'apprendre, à la fois, les poids du réseau et l'architecture de celui-ci. Ces algorithmes se divisent en deux classes : les algorithmes de construction dynamique du réseau (dits algorithmes incrémentaux). Exemple : Cascade-Correlation [Hoehfeld et Fahlman, 92] et les algorithmes d'élimination dynamique des liens et des neurones inutiles au problème [karnin, 90] (dits algorithmes d'élagage).

IV. Procédure de développement d'un réseau de neurones

Le cycle classique de développement peut être organisé en sept étapes :

- 1 - la collecte des données,
- 2 - l'analyse des données,
- 3 - le choix final de la famille d'apprentissage,
- 4 - la séparation des bases de données,
- 5 - la mise en forme des données,
- 6 - le choix d'un réseau de neurones,
- 7 - la validation.

1 - Collecte des données :

L'objectif de cette étape est de collecter des données (exemples entrées-sorties) qui vont servir, à la fois, pour développer le réseau de neurones et pour le tester. Ces données doivent être suffisantes (en nombre et en distribution) pour être représentatives des données susceptibles d'intervenir en phase d'exploitation du système neuronal. En effet, le réseau qui sera constitué n'aura de validité que dans le domaine où il a été ajusté. En d'autres termes, la présentation de données très différentes de celles utilisées lors de l'apprentissage peut entraîner une sortie totalement imprévisible.

2 - Analyse des données

Une étude statistique sur les données pourrait permettre d'écarter celles qui sont aberrantes ou redondantes. Les données aberrantes sont des données qui paraissent en contradiction avec d'autres données existantes ; leur présence risque par conséquent de perturber la stabilisation de l'apprentissage du réseau.

Aussi, dans le cas d'un problème de classification, il appartient à l'expérimentateur de déterminer le nombre de classes auxquelles les données de l'apprentissage appartiennent et de décider, pour chaque donnée, la classe à laquelle elle appartient.

Cette optimisation des entrées du réseau a des conséquences, à la fois, sur la taille du réseau (et donc le temps de simulation), sur les performances du système (pouvoir de séparation) et sur le temps de développement (temps d'apprentissage).

3 - Le choix de la famille d'apprentissage,

Comme vu précédemment, l'apprentissage du réseau constitue une étape essentielle d'adoption des paramètres du réseau. Selon les connaissances dont on dispose sur les relations entre entrées et sorties, on choisit parmi l'une des trois familles d'apprentissage : supervisé, semi-supervisé ou non supervisé.

4 - Séparation des bases de données

Le développement d'un réseau pour une application donnée exige que l'on repartisse les données dont on dispose en deux bases de données : une base pour effectuer l'apprentissage et une autre pour tester le réseau obtenu et déterminer ses performances. Ces deux bases sont disjointes. Toutefois, pour mieux conduire la phase d'apprentissage ; l'expérience a révélé l'intérêt d'utiliser, aux fins d'une validation intermédiaire, une base dite "base de validation croisée", constituée d'exemples pris dans la base d'apprentissage et d'exemples pris dans la base de test. Il n'y a pas de règle pour déterminer la répartition des données entre les trois bases. Chaque base doit cependant satisfaire aux contraintes de représentativité de chaque classe de données et doit généralement refléter la probabilité d'occurrence des diverses classes.

5 - Mise en forme des données pour un réseau de neurones

De manière générale, les bases des données doivent subir un prétraitement afin de les adapter aux normes de la présentation des entrées et des sorties d'un réseau de neurones. En effet, les valeurs acceptées sur les entrées et sortie du réseau doivent être dans l'intervalle [-1,1].

6 - Le choix final de la famille d'apprentissage

Il s'agit, à cette étape, de choisir les paramètres qui définissent une architecture, sachant qu'il n'existe pas de méthode pour bien choisir ces paramètres (nombre de couches et nombre de neurones par couche). On procède par le test d'un ensemble de réseaux et on choisit les meilleurs suivant trois critères :

- La capacité du réseau à généraliser correctement son comportement face à de nouveaux couples entrées-sorties,
- la rapidité de convergence qui traduit l'adéquation des paramètres du réseau pour atteindre une réduction rapide des écarts entrés les comportements obtenus et les comportements souhaités.
- L'optimisation de la taille du réseau (nombre de couches et nombre de neurones par couche) sachant que la réduction de la taille favorise généralement la capacité de généralisation.

7 - Validation

Le test après la fin de l'apprentissage permet à la fois d'apprécier les performances du système neuronal et de détecter le type de données qui pose problème. Si les performances ne sont pas satisfaisantes, il faudra soit modifier l'architecture du réseau, soit modifier la base d'apprentissage.

V. Apprentissage orienté vers l'intégration matérielle : problème de limitation de la précision

La rétropropagation expliquée précédemment utilise des fonctions continues et des nombres réels. Pour l'implémentation matérielle, les nombres réels sont approximés par des valeurs en virgule fixe. L'utilisation de la virgule flottante est largement déconseillée pour les applications matérielles à cause de son coût en surface silicium et en cycles d'exécution. De nombreux travaux de recherche ont été menés sur les possibilités et les effets d'utiliser la virgule fixe ou les entiers.

Selon une analyse de l'effet de la virgule fixe faite dans [Leonardo M, 91] [J.L. Holt,93], une précision minimale des poids de 20-22 bits s'avère nécessaire, en général, pour garantir la convergence de l'algorithme de rétropropagation. Cette précision peut être réduite à 14-16 bits moyennant un choix judicieux de pas d'apprentissage η. Les auteurs proposent aussi une méthode pour bien choisir le pas d'apprentissage η. Pour les sorties des neurones, la précision de 8 à 9 bits se révèle largement suffisante.

Une autre étude intéressante [Yun Xie, Marwan Jabri, 91] a montré qu'il n'est pas possible de compenser l'erreur due à la précision par une augmentation du nombre de neurones dans les couches cachées. Cette étude a montré aussi qu'une précision plus importante est toujours nécessaire pour les couches cachées.

Chapitre 2 : Présentation et analyse des implémentations matérielles des réseaux neuronaux

I.Introduction

Avant l'arrivée des cartes accélératrices et des architectures dédiées au calcul neuronal, plusieurs implémentations ont été réalisées sur des machines parallèles. Bien qu'elles n'aient pas été initialement destinées à ce type de calcul, les implémentations réalisées sur ces machines ont pu atteindre de bonnes performances.

Durant les dernières années, plusieurs implémentations sur des architectures plus ou moins dédiées (neurocomputer) ont vu le jour, aussi bien dans le milieu académique que dans le milieu industriel. On a ainsi développé et produit des puces spécifiques (NeuroChip) analogiques, digitales ou hybrides ou adapté des puces standards (cartes accélératrices, systèmes multiprocesseurs (figure 2-1)).Les informations rapportées dans les deux pages qui suivent sont tirées de [Heemskerk, 95]

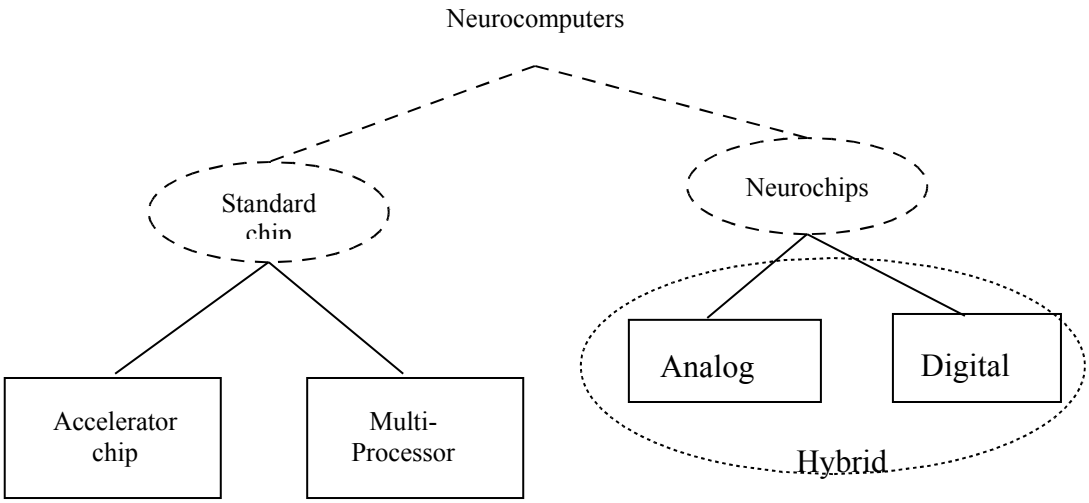


Figure 2-1 classifications *des Neurocomputers*

II. Implémentations de réseaux de neurones sur puces standard

Dans la plupart des modèles, le calcul neuronal consiste à calculer une somme pondérée, puis à l'évolution. C'est surtout le calcul de la somme pondérée qui constitue la partie la plus importante. L'instruction qui permet d'implémenter ce calcul dans un processeur DSP est l'instruction MAC conçue essentiellement pour le calcul des séries de Fourier.

En fait, les implémentations utilisant les DSP ou tout autre type de processeurs ne sont pas parmi les plus efficaces, mais leur coût très réduit et la maîtrise de leur programmation font d'elles la cible de nombreuses applications. Certains projets sont parvenus à interconnecter un nombre très important de processeurs :

Sandy/8 (Fujitsu Laboratories, Japan) jusqu'à 256 DSP,

Topsi (Texas Instruments, USA) jusqu'à 1000 DSP.

III. Implémentations de réseaux de neurones à base d'ASIC

Depuis 1980, plusieurs mises en œuvre matérielles de réseaux de neurones ont été réalisées par de grandes entreprises comme Intel [LeBouquin, 94], AT&T, Hitachi, Philips, Siemens et IBM [Hopfield, 86]. Elles diffèrent selon le nombre de neurones émulés, la précision en bits, le nombre de synapses, la vitesse du processus d'apprentissage, etc. Le tableau 2.1, tiré des travaux de Lindsey [Lindsey, 94], résume plusieurs réalisations de réseaux de neurones. Le processeur neuronique (PN) est l'unité arithmétique du neurone qui, selon l'algorithme, fait habituellement la somme des entrées multipliées par la sensibilité de chaque synapse. Selon que l'implémentation permet ou non l'apprentissage (i.e. mise à jour des poids synaptiques), on utilise la métrique CUPS (nombre de connexions mises à jour par seconde) ou la métrique CPS (nombre de connexions explorées par seconde). Une troisième métrique est le nombre de configurations d'entrée qui peuvent être traitées par seconde (patterns/sec)

Type	Name	Architecture	learn	Precision	Neurones	Synapses	speed
Analog	Intel ETANN	Feed Foward, Multilayer	No	6b x 6b	64	10280	2GCPS
	Synaptics Silicon Retina	Neuromorphic	No	na	48x48	Resistive net	na
Digital	Neuralogix NLX-420	Feed Foward, Multilayer	No	1-16b	16	Off-Chip	300CPS
	HNC 100-NAP	GP,SIMD,FP	program	32b	100 PE	512K off-chip	250MCPS 64MCUPS
	Hitachi WSI	Wafer, SIMD	Hopfield	9b x 8b	576	32k	138MCPS
	Hitachi WSI	Wafer, SIMD	Back Progagation	9b x 8b	144	na	300MCUPS
	Inova N64000	GP, SIMD, Int	program	1-16b	64 PE	128 K	870MCPS 220MCUPS
	IBM ZISC036	RBF	ROI	8b	36	64x36	250k pat/s
	MCE MT19003	Feed Forward, Multilayer	No	13b	8	off-chip	32MCPS
	Micro Devices	Feed Forward, Multilayer	No	1b x 16b	1PE	8	8.9MCPS
	Nestor/Intel NI1000	RBF	RCE, PPN	5b	1PE	256x1024	40k pat/s
	Philips Lneuro-1	Feed Forward, Multilayer	No	1-16b	16PE	64	26MCPS
	Siemens MA-16	matrix ops	No	16b	16PE	16x16	400MCMPS
Hybrid	AT&T ANNA	Feed Forward, ML	No	3b x 6b	16-256	4096	2.1GCPS
	Bellcore CLNN-32	FCR	Boltzmann	6b x 5b	32	992	100MCPS 100MCUPS
	Mesa Research Neuroclassifier	Feed Forward, Multilayer	No	6b x 5b	6	426	21GCPS
	Ricoh RN-200	Feed Forward, Multilayer	Back Propagation	na	16	256	3.0GCPS

Tableau 2-1 implémentations matérielles de réseaux de neurones

IV. Implémentations de réseaux de neurones à base de FPGA

Comme leur non l'indique, les FPGA (Field Programmable Gate Arrays) sont des circuits programmables. Les FPGA d'aujourd'hui sont des composants entièrement reconfigurables, ce qui permet de les reprogrammer à volonté afin de réaliser des accélérateurs de certaines phases de calculs.

Les FPGA ont constitué une véritable avancée technologique au cours des dernières années. Ils fonctionnent désormais à des fréquences de l'ordre de 400Mhz et possèdent des dizaines de millions de portes logiques avec des mémoires intégrées et des macros optimisant les calculs (addition et multiplication). L'évolution de leur architecture interne et le développement en parallèle des outils CAD qui les supportent ont changé les habitudes et le flot de design dans le monde industriel, de même qu'ils ont permis d'ouvrir la porte de la conception hardware au grand public.

Les FPGA peuvent ainsi remplacer les ASIC dans certaines applications dont le produit ne sera pas fabriqué à très grande échelle et dont les exigences de performances en fréquence et en taille ne sont pas très importantes.

D'une façon générale, les codages hdl (hardware description language) peuvent être utilisés aussi bien pour des implémentations ASIC que pour des implémentations FPGA.

Plusieurs développements ont été réalisés à base de FPGA [Blanz, 92] [Marcelo, 94] [Aaron, 94] bénéficiant de l'évolution de ces derniers : jusqu'aux années 95, les FPGA atteignaient à peine les 5000 portes logiques, ce qui permettait tout juste d'implémenter le traitement d'une instruction MAC [Blanz, 92] [Marcelo, 94]. Dans les années qui ont suivi, de nombreuses implémentations [GIRAU, 99] ont été faites en utilisant l'arithmétique série [Beuchat, 01] [Beuchat, 02] [Trivedi, 77]. Le recours à l'arithmétique série (i.e. calcul bit par bit) permet, en effet, de réduire la surface utilisée. En revanche, il oblige à effectuer le calcul par itérations de cycles. Une autre solution pour contourner le problème de la densité des FPGA consiste à diviser le calcul en étapes et à reconfigurer le FPGA à chaque étape. Une telle solution doit présenter un compromis entre temps de reconfiguration et temps de calcul [Elredge, 94] [Beuchat].

V. Conclusion

Les implémentations présentées proposent toutes des solutions pour contourner le problème de la limitation de capacité. Mais, à nos jours, les FPGA et les ASIC offrent de plus en plus de capacité d'intégration et de nouveaux concepts ont vu le jour comme le System On Chip (SOC) et le Network On Chip (NOC).

Dés lors, il apparaît plus intéressant d'évoluer vers des implémentations sur une architecture qui soit le plus possible parallèle, avec une interface utilisateur qui soit simple à manipuler pour la réutilisation. Nous allons présenter, dans le prochain chapitre, une architecture qui va dans ce sens : Une *IP* qui vise à profiter le mieux possible des nouvelles ressources mises à disposition, en offrant le plus possible de paramètres génériques.

Chapitre 3 : Implémentation matérielle

I. Introduction

Nous présentons, dans ce chapitre, notre démarche de conception d'un processeur spécifique du traitement neuronal baptisé NEURONA. Notre approche vise une architecture générique, dynamiquement reconfigurable, permettant l'implémentation directe d'applications sur un réseau de neurones multicouches capable de supporter toutes les méthodes d'apprentissage à base de rétro-propagation [Hoehfeld et Fahlman, 92] [karnin, 90] [Widrow, 90]. Le processeur NEURONA est élaboré sous forme d'une propriété intellectuelle codée en VHDL et synthétisable sur FPGA ou ASIC. La généricité vise à permettre de fixer, lors de la synthèse, la taille de l'arithmétique ainsi que le nombre maximum de couches et de neurones pouvant être supportés par couche. Tandis que la reconfiguration dynamique concerne la topologie du réseau caractérisée par le nombre de couches, le nombre de neurones par couche, les poids de connectivité entre neurones et la morphologie de la fonction d'activation du neurone.

Nous abordons l'étude de l'architecture proposée en commençant, dans une première phase, par rappeler le principe du traitement neuronal à travers la présentation de l'algorithme de propagation. C'est alors que nous proposons une architecture permettant d'implémenter cet algorithme. Pour concrétiser la généricité et la reconfiguration dynamique souhaitées, nous avons défini cette architecture en séparant la mémorisation de l'architecture du réseau de neurones (topologie, poids synaptiques et sorties des neurones), le calcul du potentiel post-synaptique, la sigmoïde fonction d'activation et l'unité de commande qui pilote l'ensemble. La généricité se traduit par la possibilité de fixer, lors de la synthèse, les dimensions maximales des différents modules. La reconfiguration dynamique se traduit par la possibilité de modifier le contenu des mémoires décrivant l'architecture du réseau de neurones et ce, à tout moment de l'exécution. L'unité de commande ainsi que le schéma du chemin des données restent les mêmes. Dans une deuxième phase, nous montrons comment cette même architecture, moyennant certaines modifications, peut être activée dans les deux modes propagation et rétro-propagation. Après avoir modélisé l'unité de commande de NEURONA à

l'aide d'une machine à états finis, nous présentons quelques résultats de synthèse permettant de conclure sur l'intérêt et l'efficacité de NEURONA.

Le faible taux d'occupation de surface obtenu avec les différents essais de synthèse nous a suggéré de complexifier l'architecture proposée en vue d'accélérer les différents calculs par un traitement parallèle. Deux solutions allant dans ce sens sont discutées à la fin de ce chapitre.

II. Réalisation de la propagation

Considérons le réseau de neurones présenté en figure 3.1, constitué de k couches (de 0 à k-1), chaque couche i est composée de P_i neurones. Le réseau accepte N entrées E_i, il élabore q sorties S_j

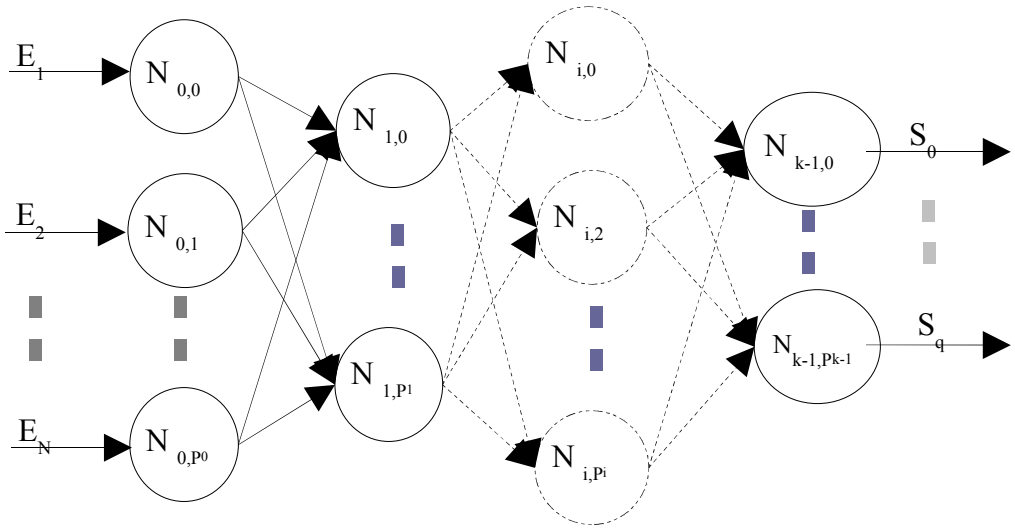


Figure 3-1 Exemple de réseau de neurone

Le traitement neuronal propage les entrées Ei vers les sorties S_j. Les sorties des neurones de la première couche sont initialisées par les entrées E_i. La propagation s'effectue à travers les neurones des différentes couches ; chaque neurone $N_{m,j}$ de la couche m calcule sa sortie $X_{m,j}$ à partir d'une somme pondérée des sorties $X_{m-1,i}$ des neurones de la couche m-1 qui lui sont connectés. Ce traitement est résumé par l'algorithme suivant :

1. Algorithme

Début

Initialisation des sorties des neurones de la première couche : ($X_{0,i} = E_i$)

Pour chaque couche m, en commençant par la couche 1 et en allant ver la couche k-1,

Pour chaque neurone j de m

Calculer la sortie $X_{m,j}$ de ce neurone :

$$X_{m,j} = f\left(\left(\sum_{i=1}^{P_{m-1}} X_{m-1,i} \times W_{i,j}\right) + b_j\right)$$

// $W_{i,j}$ étant le poids synaptique de la connexion $N_{m-1,i}$ - $N_{m,j}$ et b_j est le //facteur "biais" associé au neurone.

//Les sorties finales S_j se confondent avec les $X_{k,j}$.

finPour

finPour

fin

2. Câblage de l'algorithme

Le câblage proposé est schématisé par le chemin des données de la figure 3.2. On distingue :

- la mémoire T qui contient le codage de la topologie du réseau neuronal (nombre de couches et nombre de neurones par couche). Cette mémoire associe à chaque couche i (i allant de 0 à k-1) le mot d'adresse i dont le contenu indique le nombre de neurones constituant la couche. Le mot d'adresse k contient la valeur zéro pour indiquer la fin du réseau.
- la mémoire W qui contient les poids synaptiques $W_{i,j}$ des connexions entre neurones. L'ordre de stockage des W_{ij} dans la mémoire est choisi de façon à simplifier la génération d'adresse lors de l'exécution séquentielle de l'algorithme.
- la mémoire X qui mémorise les valeurs calculées pour les sorties des neurones, que ce soit en mode propagation ou en mode rétro-propagation,
- une UAL qui calcule les sorties des neurones partant des valeurs des X_i sur les entrées et des poids synaptiques W_{ij},
- une mémoire F qui synthétise la fonction d'activation,

une unité de commande CTR qui pilote l'ensemble des composantes précédentes en vue de réaliser le traitement neuronal. (i.e. implémentation de l'algorithme).

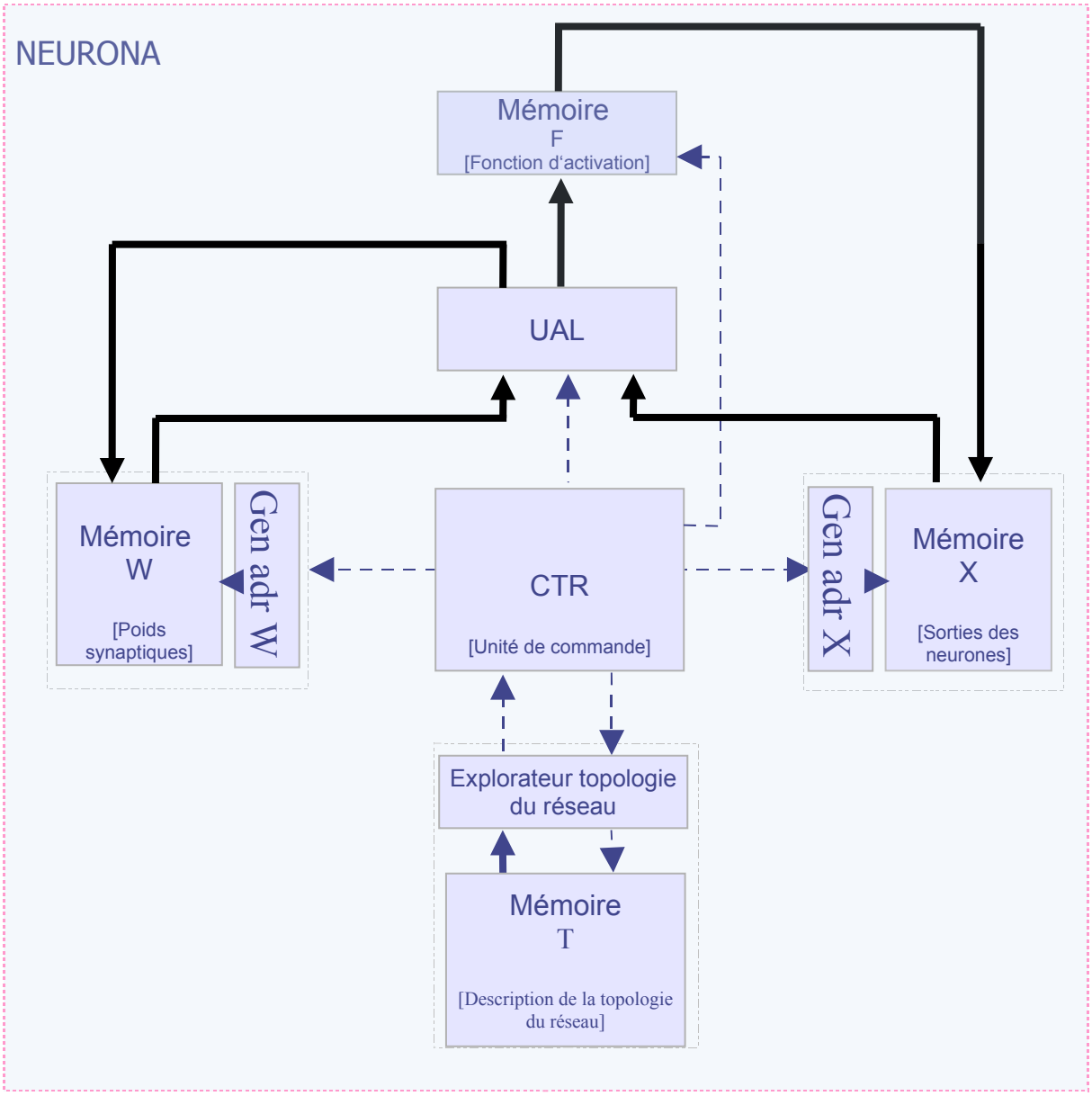


Figure 3-2 Schéma bloc détaillé

Chacune des mémoires T, W et X est dotée d'une logique d'accès :

- Dans le cas de la mémoire T, cette logique permet d'explorer en séquence les différents neurones des différentes couches en commençant par le neurone $N_{0,0}$.

L'exploration est dirigée pas à pas par l'unité de commande CTR. A chaque pas, ladite logique rend au CTR une information précisant si, à ce pas, on va passer au neurone suivant de la même couche ou au premier neurone de la couche suivante.

- L'unité de commande exploite cette information pour diriger l'évolution de l'adressage dans l'accès aux poids synaptiques correspondants dans la mémoire W (à travers la

logique gen_adr_w) et dans l'accès aux valeurs de sortie correspondantes dans la mémoire X (à travers la logique gen_adr_x)

3. Interface de l'*IP* NEURONA

La figure 3-3 donne la schématisation de haut niveau (top level) de l'IP implémentant la machine NEURONA. On distingue trois modes de fonctionnement :

- le mode de lecture/écriture dans l'une des mémoires T, W, X et F. Il s'agit d'un accès direct à ces mémoires en court-circuitant les logiques d'accès pilotées par l'unité de commande.
- le mode d'exécution d'un cycle complet de propagation.
- le mode d'exécution d'un cycle complet de rétro-propagation en apprentissage.

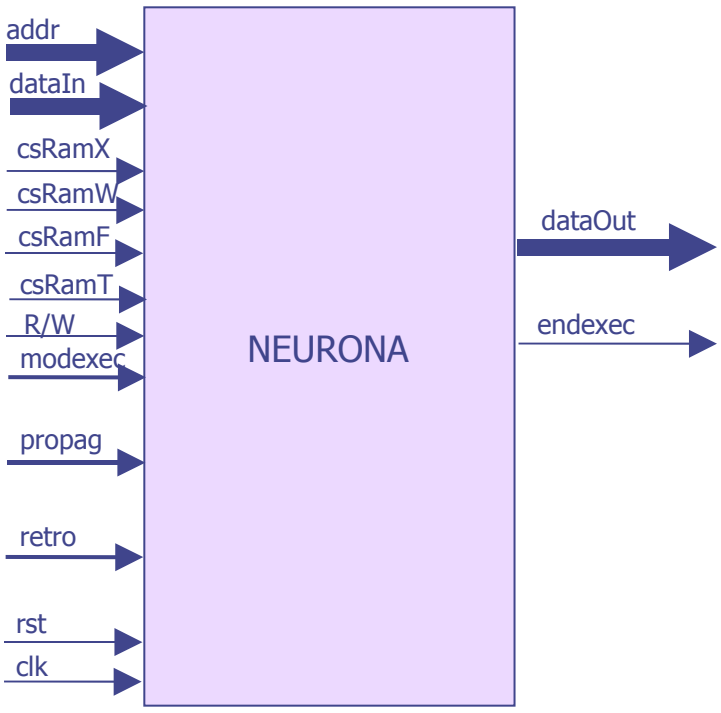


Figure 3-3 Schématisation de haut niveau (top level) de l'IP implémentant la machine NEURONA

a. Mode de lecture / écriture

Ce mode est sélectionné en positionnant à 0 l'entrée modexec. On doit préciser la mémoire ciblée (mise à 1 de l'une des entrées csRamX, csRamW, csRamT ou csRamF), préparer l'adresse sur les entrées addr, indiquer sur l'entrée R/W le type d'accès demandé et, en cas d'écriture, préparer la donnée à écrire sur les entrées dataIn. Un cycle de

lecture/écriture se déclanche au premier top d'horloge qui trouve l'entrée modexec à 0 et l'une des entrées de sélection mémoire à 1. En cas de lecture, l'information est récupérée sur la sortie dataOut.

b. Mode de propagation

Ce mode est sélectionné en positionnant à 1 l'entrée modexec ainsi que l'entrée propag ; l'entrée retro étant maintenue à 0. Un cycle complet de propagation se déclenche alors dès le prochain top d'horloge. La fin du cycle est annoncée par la mise à 1 de la sortie endexec. Cette dernière est remise à 0 à chaque début de cycle et maintenue à cette valeur jusqu'à la fin du cycle.

c. Mode de rétro-propagation

Ce mode est sélectionné en positionnant à 1 l'entrée modexec ainsi que les entrées propag et retro. Un cycle complet de propagation puis de rétro-propagation se déclenche alors dès le prochain top d'horloge. La fin du cycle est annoncée par la mise à 1 de la sortie endexec.

Ainsi, l'utilisation de l'IP doit commencer par le chargement des données relatives à la topologie du réseau, aux poids synaptiques et à la fonction d'activation.

Pour commander une propagation, l'on doit charger les entrées du réseau dans les mots correspondants dans la mémoire X avant de commander le mode propagation (modexec = 1, propag = 1, retro = 0).

Pour commander une rétro-propagation, l'on doit charger dans la mémoire X les entrées du réseau ainsi que les valeurs de sorties désirées puis commander la mode rétro-propagation (modexec = 1, propag = 1, retro = 1). L'IP effectue alors un cycle de propagation suivi d'un cycle rétro-propagation.

4. Etude de cas

On considère le réseau ci-dessous (*figure 3-4)*. Ce réseau est représenté dans la mémoire T par 4 mots contenant la suite des valeurs 3, 2, 3,0. (Réseau à trois couches, avec 3 neurones en couche d'entrée, 2 neurones en couche cachée et 3 neurones en couche de sortie).

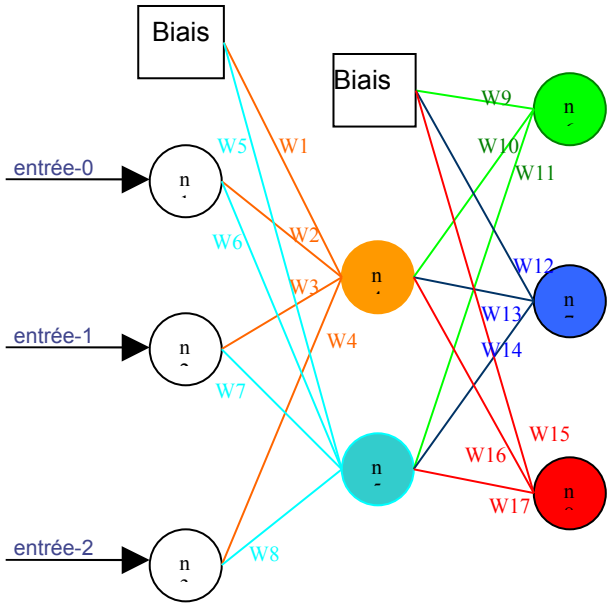


Figure 3-4 Réseau étudié

Le codage complet des données du réseau est donné en *figure 3-5*.

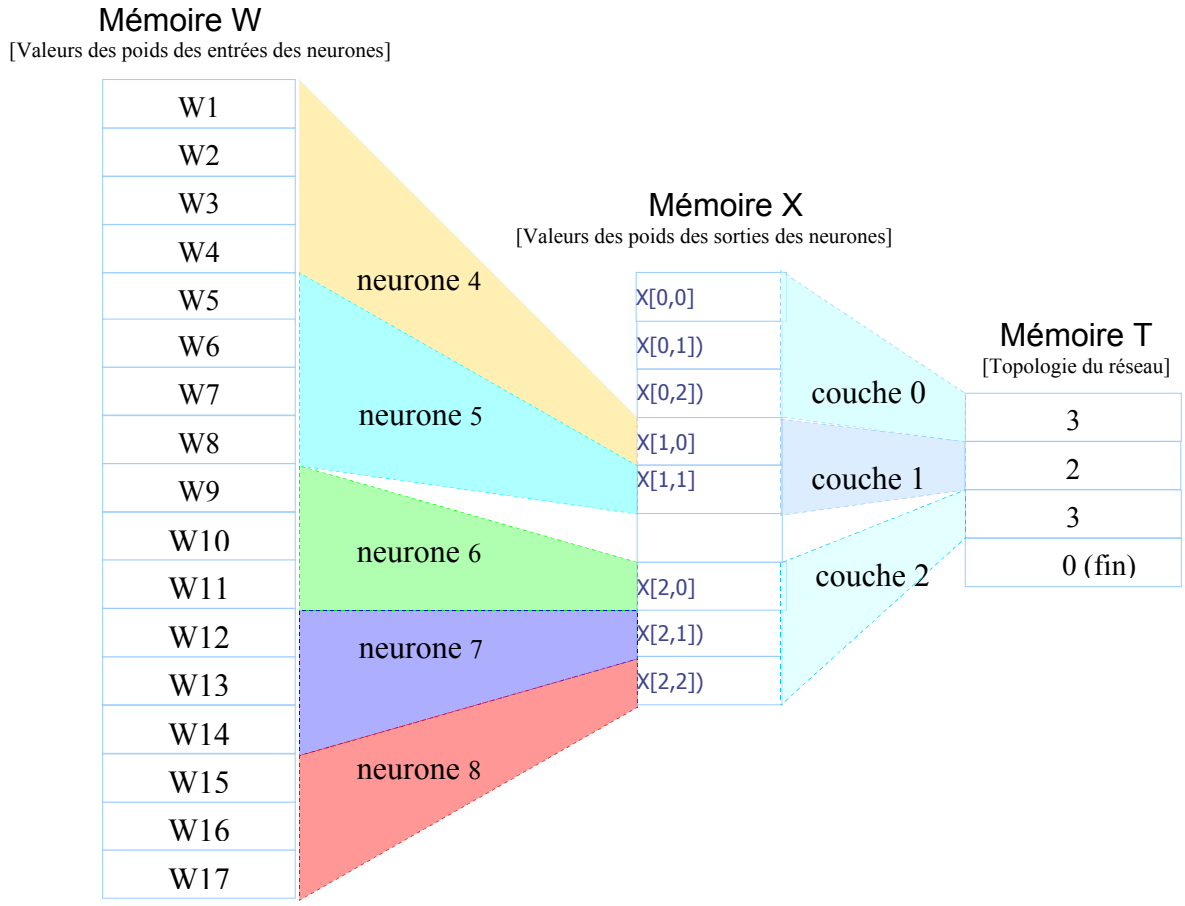


Figure 3-5 Codage mémoire du réseau étudié

Ainsi, le contenu de la mémoire T jusqu'au mot k (fin) détermine l'interprétation des mots de la mémoire X et de la mémoire W. Si la mémoire T est conçue avec n mots de m bits, alors la configuration maximale de la mémoire X doit être de (2^m-1) (n-1) mots et la configuration de la mémoire W doit être de 2^m (2^m-1) (n-1) mots.

Déroulement de l'exécution de l'algorithme de propagation

Les valeurs des entrées du réseau sont chargées directement comme valeurs de sortie X[0,0], X[0,1] et X[0,2] du neurone de la couche 0, Le processeur effectue alors le traitement suivant :

*Couche 1:

Pour le neurone n4 :

L'UAL calcule la somme pondérée :

`acc` ← w1

`acc` ← `acc` + w2 * X[0,0]

`acc` ← `acc` + w3 * X[0,1]

`acc` ← `acc` + w4 * X[0,2]

Passage par la fonction d'activation (qui est implémentée via la mémoire F) et mémorisation de la valeur finale obtenue :

X [1,0] ← sigmoïde (acc)

Pour cette exécution, les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W :	1	2	3	4	
Générateur d'adresse X :	b	[0,0]	[0,1]	[0,2]	**[1,0]**

Pour le neurone n5 :

`acc` ← w5

`acc` ← `acc` + w6 * X[0,0]

`acc` ← `acc` + w7 * X[0,1]

`acc` ← `acc` + w8 * X[0,2]

X [1,1] ← sigmoïde (acc)

Générateur d'adresse W :	5	6	7	8	
Générateur d'adresse X :	b	[0,0]	[0,1]	[0,2]	**[1,1]**

*Couche 2 :

Pour le neurone n6 :

acc ← w9

acc ← acc + w10 * X[1,0]

acc ← acc + w11 * X[1,1]

X [2,0] ← sigmoïde (acc)

Générateur d'adresse W : 9 10 11

Générateur d'adresse X : b [1,0] [1,1] **[2,0]**

Pour le neurone n7 :

acc ← w12

acc ← acc + w13 * X[1,0]

acc ← acc + w14 * X[1,1]

X [2,1] ← sigmoïde (acc)

Générateur d'adresse W : 12 13 14

Générateur d'adresse X : b [1,0] [1,1] **[2,1]**

Pour le neurone n8 :

acc ← w15

acc ← acc + w16 * X[1,0]

acc ← acc + w17 * X[1,1]

X [2,2] ← sigmoïde (acc)

Générateur d'adresse W : 15 16 17

Générateur d'adresse X : b [1,0] [1,1] **[2,2]**

5. Conception de l'*IP* NEURONA

a. Explorateur de la topologie du réseau

La figures 3-6 schématise les entrées / sorties du sous-bloc "Explorateur de la topologie du réseau".

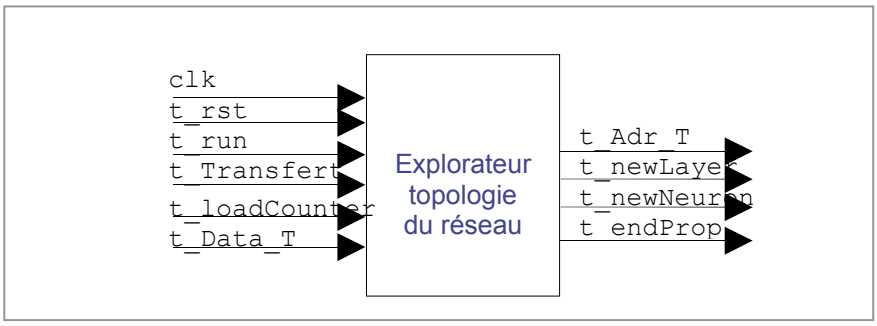


Figure 3-6 entrées / sorties du sous bloc "Explorateur topologie du réseau"

Ce bloc génère trois signaux de sortie (t_newNeuron, t_newLayer, t_endProp) destinés au séquenceur pour lui permettre de suivre sa progression dans l'exploration de la topologie du réseau. Une quatrième sortie t_Adr_T est destinée à l'adressage de la mémoire T. Le chemin des données est donné en *figure 3-7.*

- Rappelons que chaque mot m_i de la mémoire T (en commençant de m_0) indique le nombre de neurones de la couche i. Le premier mot m_j ayant la valeur 0 indique la fin du réseau.
- La propagation des sorties d'une couche i vers les sorties de la couche i+1 exige que l'on dispose en même temps des adresses des neurones de la couche i et des adresses des neurones de la couche i+1.
- Ainsi, on charge m_i dans le registre prev_layer et m_{i+1} dans le registre current_layer ; ce chargement s'effectue par lecture de la mémoire T (t_Data_T) et transfert d'un registre à l'autre. Les contenus des deux registres sont ensuite chargés dans deux compteurs, respectivement counter_N et counter_L.

A chaque top d'horloge et si t_run = 1, le counter_N est décrémenté et si sa valeur devient 0 alors : la sortie "t_newNeuron" est activée pour indiquer au séquenceur que l'on vient de terminer le traitement du neurone courant ; le counter _N est de nouveau chargé.

- En même temps, le counter_L est décrémenté et si sa valeur devient 0 alors : la sortie "newLayer" est activée pour indiquer au séquenceur que l'on vient de terminer le traitement du la couche courante ; le contenu du registre current_layer passe dans le registre prev_layer et le mot m_{i+2} de la mémoire T est chargé dans le registre current_layer.

- Si la nouvelle valeur de current_layer est 0 alors la sortie t_endProp est activée pour indiquer au séquenceur que l'on a atteint la fin du réseau.

- Il faut préciser que la mémoire T est à lecture synchrone : tout changement d'adresse t_adr_T effectué au top d'horloge t provoque l'affichage du résultat de la lecture (t_Data_T) au top d'horloge t+1.

La commande "t_Transfert" est générée par le séquenceur ; elle provoque a) le transfert de current_layer dans prev_layer, b) le chargement de t_Data_T dans current_layer et c) l'incrémentation de counter_T. La commande "t_loadCounter " est également générée par le séquenceur.

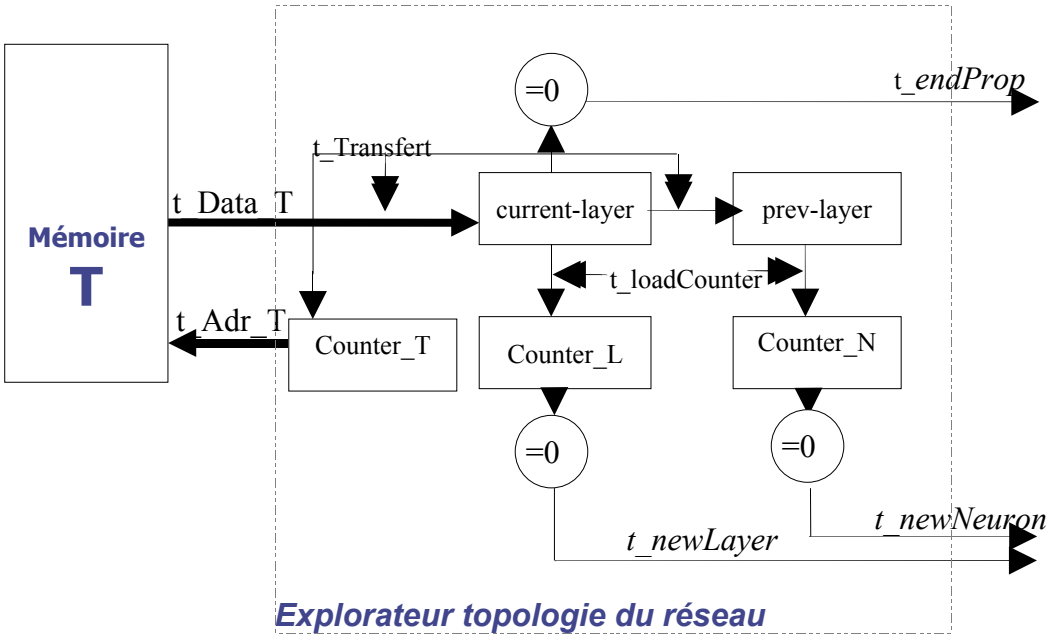


figure 3-7 Le chemin des données du bloc "Explorateur topologie du réseau"

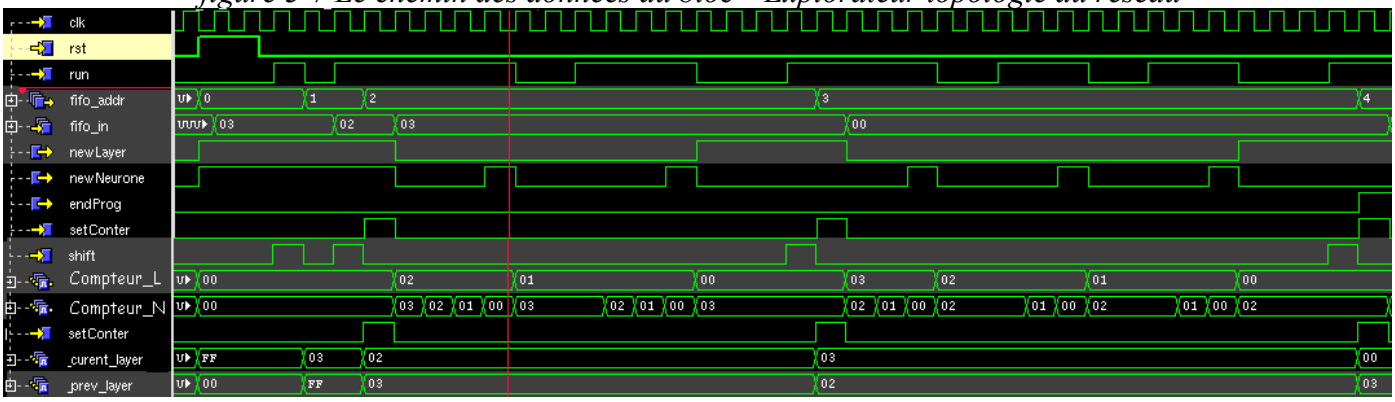


Figure 3-8 Waveform de la séquence d'exécution d'Explorateur topologie du réseau

b. Générateur d'adresse W *(gen_adr_w)*

La figures 3-9 schématise les entrées / sorties du sous bloc " Générateur d'adresse W".

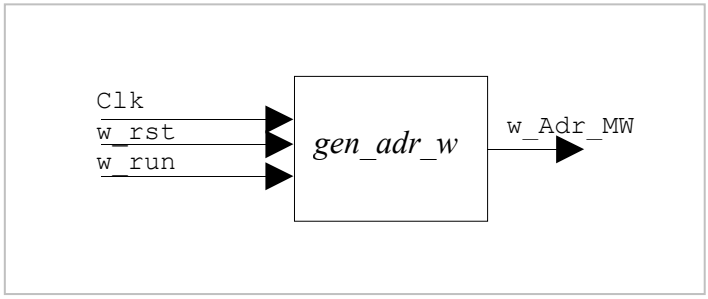


Figure 3-9 entrées / sorties du sous bloc "gen_adr_w"

De par les explications fournies dans le paragraphe 2 et dans l'illustration faite sur l'ensemble étudié, on comprend aisément que le bloc gen_adr_w lit la mémoire W en séquence depuis le mot d'adresse 0. Par conséquent, ce générateur d'adresse peut être réalisé simplement à l'aide d'un compteur. Il est commandé par l'unité de commande soit pour une remise à zéro (à l'initialisation) soit pour une incrémentation (au fur et mesure de l'exécution via le pin t_run)

c. Générateur d'adresse X *(gen_adr_x)*

La figures 3-10 schématise les entrées / sorties du sous bloc "Générateur d'adresse X ".

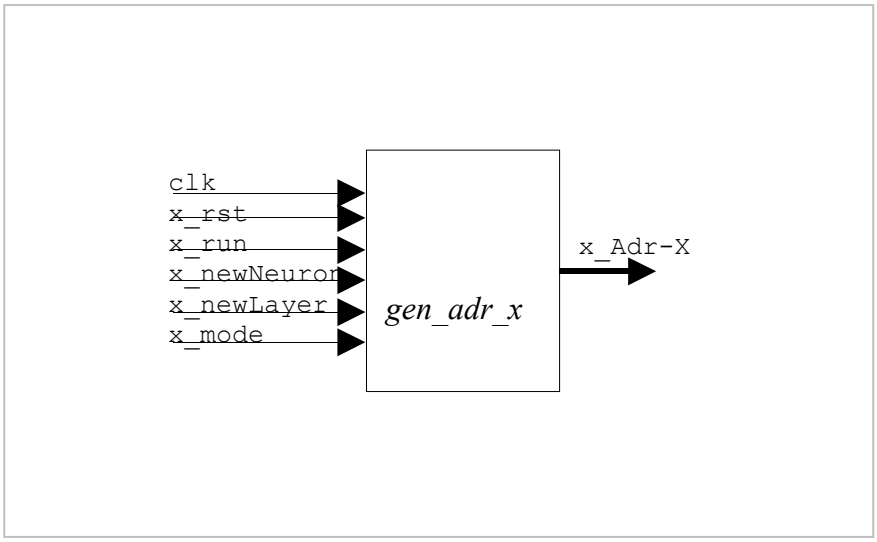


Figure 3-10 entrées / sorties du sous-bloc "gen_adr_x"

Ce bloc génère la sortie Adr_X destinée à l'adressage de la mémoire X ; cette adresse est composée des deux champs "numéro de couche" et "numéro de neurone dans la couche".

Soient [i,0], [i,1], … [i,k_i] les adresses des neurones de la couche i et soient [i+1,0], [i+1,1], … [i+1,k_{i+1}] les adresses des neurones de la couche i+1. Le calcul des sorties des neurones de la couche i+1 va s'effectuer selon la formule :

X[i+1,j] = sigmoid (acc(i+1,j)) avec

acc(i+1,j) = b+ W_{i0}* X[i,0]+ W_{i1}* X[i,1] … W_{ik_i} * X[i,k_i]

Pour ce calcul, le bloc gen_adr_x va donc générer la séquence d'adresses :

[i,0], [i,1], … [i,k_i], **[i+1,0]**

[i,0], [i,1], … [i,k_i], **[i+1,1]**

……...

[i,0], [i,1], … [i,k_i], **[i+1, k_{i+1}]**

Sachant que les adresses en gras [i+1,0], [i+1,1], … [i+1,ki+1] sont des adresses d'écriture tandis que les autres adresses sont des adresses de lecture. Pour ce faire, le calcul est initialisé avec :

- Registre_L_L chargé avec la valeur i,
- Compteur_N_L chargé avec la valeur 0,
- Compteur_L_S chargé avec la valeur i+1,
- Compteur_N_S chargé avec la valeur 0,

La progression est gérée par le unité de contrôle qui exploite les signaux x_neuNeuron et x_newLayer. Ainsi, à chaque impulsion d'horloge (clk) :

- Si l'on a x_neuNeuron =0 et x_newLayer=0, alors Compteur_N_L est incrémenté de 1.
- Si l'on a x_neuNeuron =1 et x_newLayer=0, alors Compteur_L_S est incrémenté de 1 et Compteur_N_S est remis à 0.
- Si l'on a x_newLayer=1, alors la valeur de Compteur_L_S est transférée dans Registre_L_L, Compteur_L_S est incrémenté de 1 et les compteurs Compteur_N_S et Compteur_N_L sont remis à zéro.

La sélection adresse de lecture et adresse d'écriture est gérée par l'entrée mode à 1 (i.e écriture) chaque fois qu'il se prépare à passer à un nouveau neurone ou à une nouvelle couche.

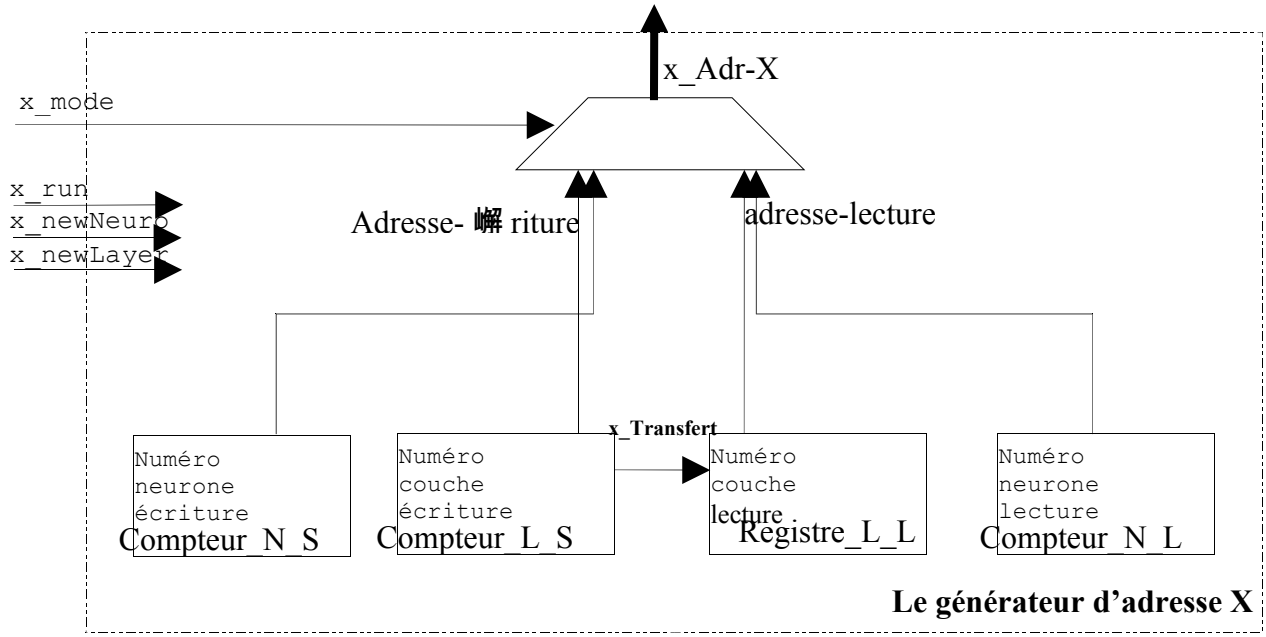


Figure 3-11 Le chemin des données du bloc " générateur d'adresse X "

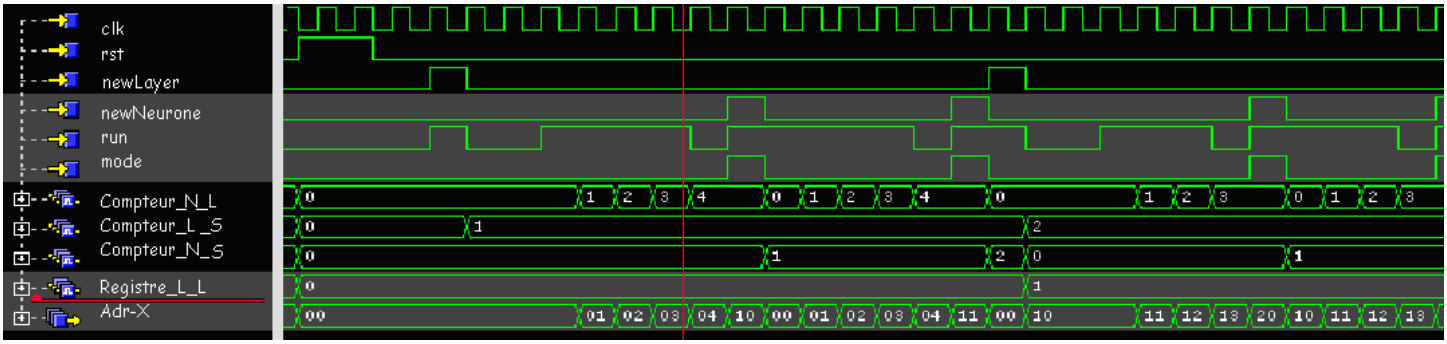


Figure 3-12 Waveform de la séquence d'exécution gen_adr_x

d. Unité de calcul

La figures 3-13 schématise les entrées / sorties du sous-bloc " Unité de calcul ".

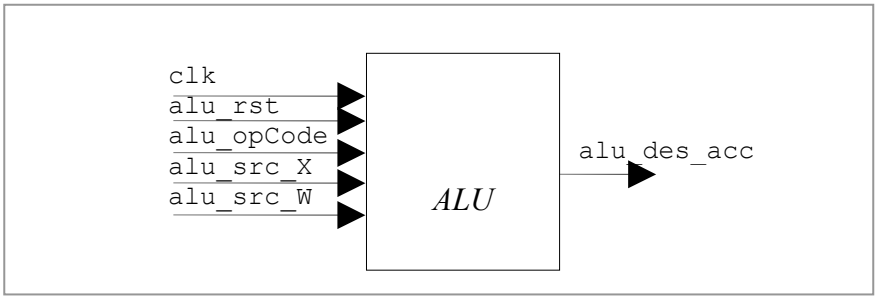


Figure 3-13 L'unité de calcul

Cette unité doit permettre le calcul d'une somme pondérée, connue comme étant l'opération de base des processeurs DSP : il s'agit de l'opération MAC. L'opération MAC est donc une accumulation de résultats de la somme des produits ($\Sigma_j X_j W_{ij}$) matérialisées par le diagramme de flot de donnée de la *figure 3-14*.

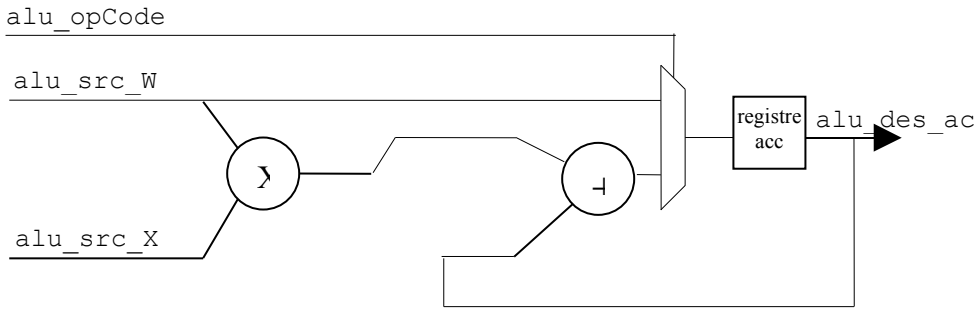


Figure 3-14 Le chemin des données du bloc ALU

L'utilisation de la multiplication en virgule fixe signée s'accompagne parfois d'une perte de valeurs significatives qui mène au blocage de la convergence de l'algorithme. Plusieurs types d'implémentation peuvent être réalisés agissant sur le problème de troncature:
Supposons que la taille du bus W et X sont relativement w et x le résultat de la multiplication (W*X) et w + x.
Pour prévoir la taille du registre acc qui suit l'opération de l'addition, le nombre d'itérations d'accumulation pour notre problème : c'est le nombre maximal de neurones par couche, soit m ce nombre la taille de l'additionneur est dans ce cas égale à w + x + m -1.
Pour réduire la taille de l'additionneur et du registre qui devient non raisonnable dans le cas où m est grand. Deux solutions sont envisageables, la première est de faire une troncature à la sortie de la multiplication, la seconde consiste à choisir une taille du registre raisonnable ce qui implique l'ajout du gestionnaire du débordement.
La sortie de ce module doit passer par la fonction d'activation qui reçoit une entrée de taille voisine à 8 bits. De ce fait, une troncature est obligatoire à la sortie de ce bloc.
L'ALU est générique, ses paramètres sont :
Nombre de bits après et avant la virgule pour W et X.
Nombre de bits après et avant la virgule de la sortie de l'ALU (troncature à la sortie)
Nombre de bits du registre utilisé pour l'opération MAC.

e. Fonction d'activation

Il s'agit d'approximer la composante non linéaire de la fonction de transfert du neurone. Cette composante est modélisée, dans la plus part des cas, par une sigmoïde. L'implémentation de cette composante non linaire repose sur l'approximation par tableau de valeurs implémenté à l'aide d'un bloc mémoire. Les lignes d'adresse sont pilotées par la sortie donnée de l'opérateur MAC permettant ainsi le décodage en sortie la valeur de la fonction.

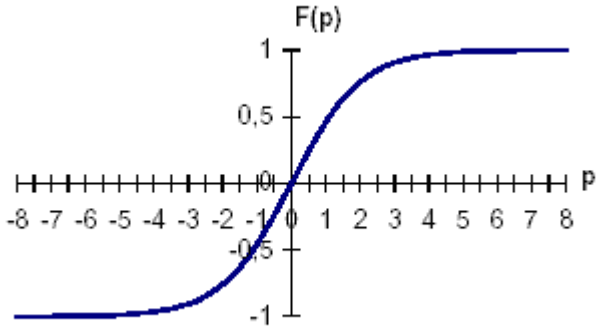


Figure 3-15 - fonctions d'activation

L'implémentation de la fonction d'activation est aussi générique. La généricité concerne :

- le pas d'échantillonnage.
- l'intervalle de variation, ($8<x$, $f(x) \approx 1$ et $-8>x$, $f(x) \approx -1$) ; la différence au delà est inférieure à 10^{-3}.
- la précision de sortie en nombre de bits.

Ces paramètres déterminent la taille de la mémoire et la taille des mots.

f. La machine d'état

La figure 3-16 schématise les entrées / sorties du sous bloc " CTR ".

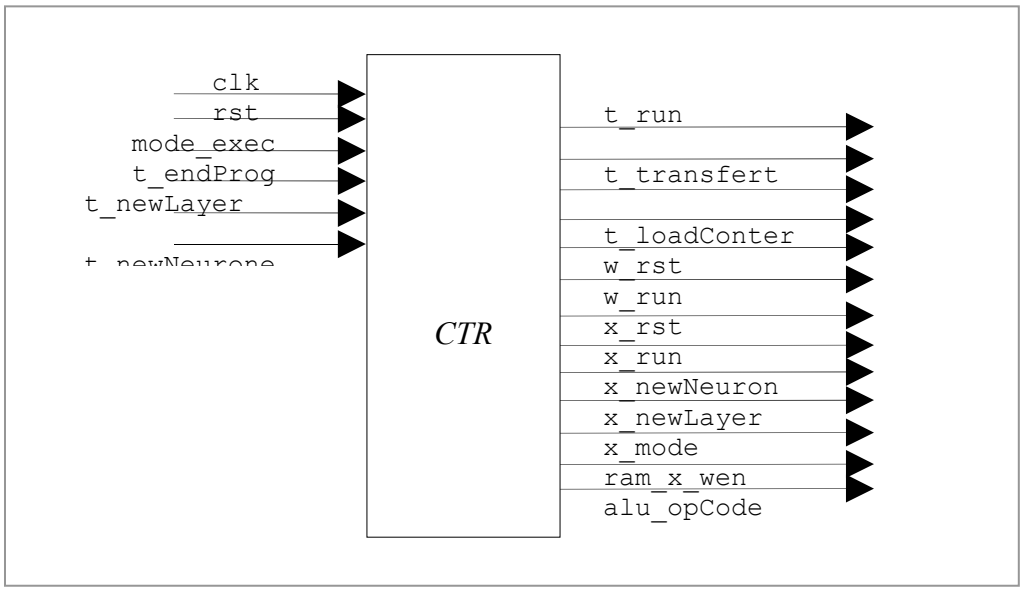


Figure 3-16 le bloc CTR

L'unité contrôle l'ensemble des blocs via les pins de sortie. Au début, le séquenceur initialise les blocs. Ensuite, au lancement de la propagation, via les pins `mode_exec` , `propag`, l'unité contrôle ordonne à l'explorateur de topologie de réseau de lire les deux premiers mots de la mémoire T (voir tableau l'état `init0, shift et setConter`). Une fois initialisé on commence à calculer la somme pondérée $\Sigma_j X_j W_{ij}$ (voire tableau l'état `bais et mac`). `Enfin, la valeur de sortie de la fonction d'activation est enregistré dans la mémoire X` (voire tableau l'état `activFunc et saveOut`).

Le tableau et le graphe d'états résument le fonctionnement de la CTR.

Etat courant	Condition	Etat suivant	Sortie de CTR
idle		Idle	Initialisation de tous les registres des différents blocs.
	Modexec = '0' t_endProg = '0' propag = '1'	Init0	
init0		Init1	lecture du premier mot mémoire T et passage à la lecture de l'adresse suivante. t_run <= '1'; t_transfert <= '1';
Init1		Shift	passage à une nouvelle couche. genaX_run <= '1'; genaX_newLayer <= '1';
Shift		setConter	Lecture du mot mémoire T et

			passage à la lecture de l'adresse suivante. t_run <= '1'; t_transfert <= '1';
SetConter		bais	Charger les compteurs du bloc "Explorateur de la topologie du réseau" t_run <= '1'; t_loadConter <= '1';
Bais	t_endProg = '1'	idle	Initialisation du registre acc par le bais. t_run <= '1'; x_run <= '1'; w_run <= '1'; alu_opCode <= '1';
		Mac	
Mac	t_newNeuron = '1'	activFunc	Lancer la opération MAC,(on ajoute au registre acc le produit W*X). t_run <= '1'; x_run <= '1'; w_run <= '1';
		Mac	
ActivFunc		saveOut	Passage par la fonction d'activation
SaveOut	t_newLayer = '1'	Init1	Sauvegarder la valeur de X et aller au neurone suivant x_mode <= '1'; x_wen <= '0'; x_run <= '1'; x_newNeurone <= '1';
		bais	

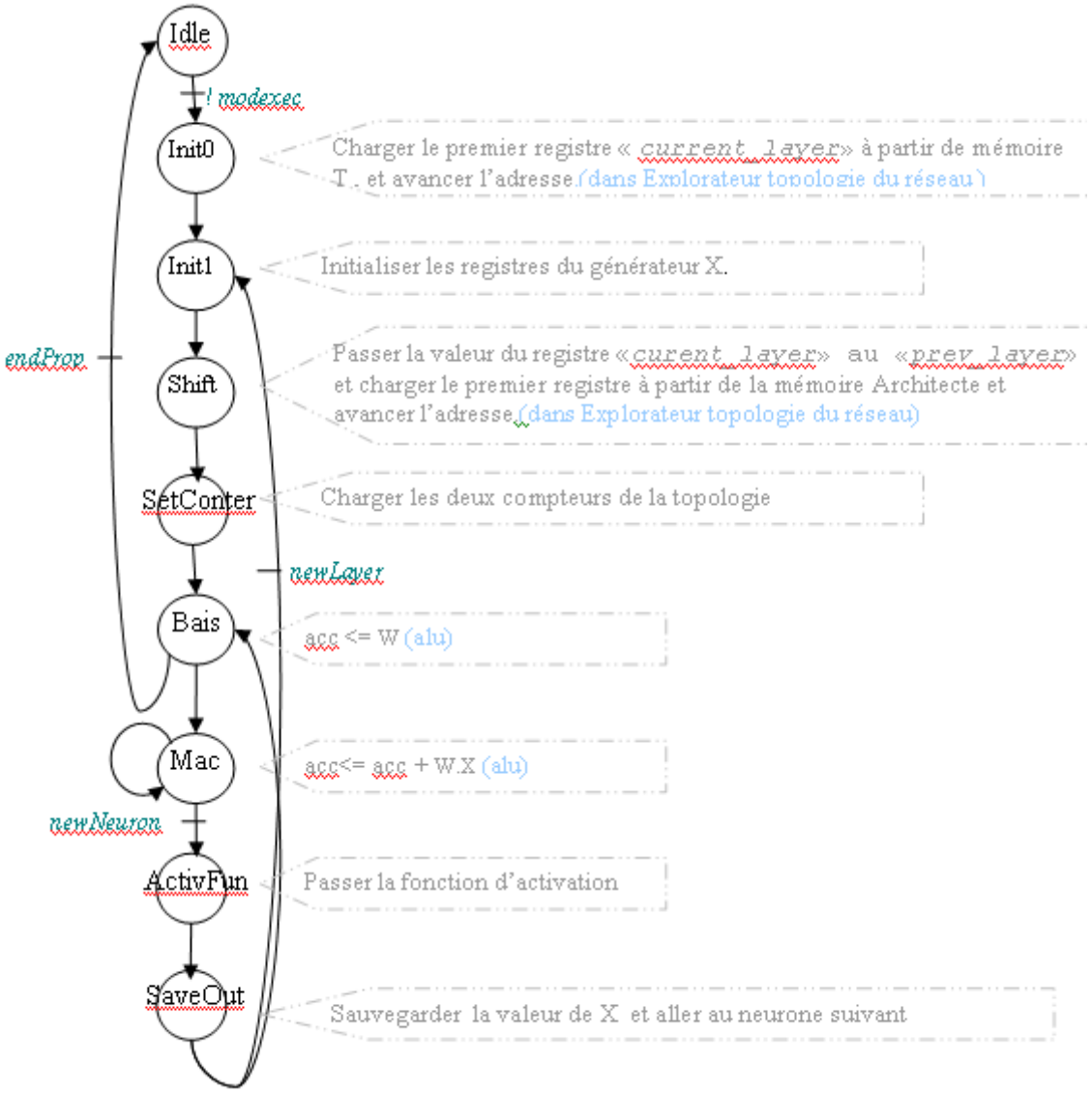
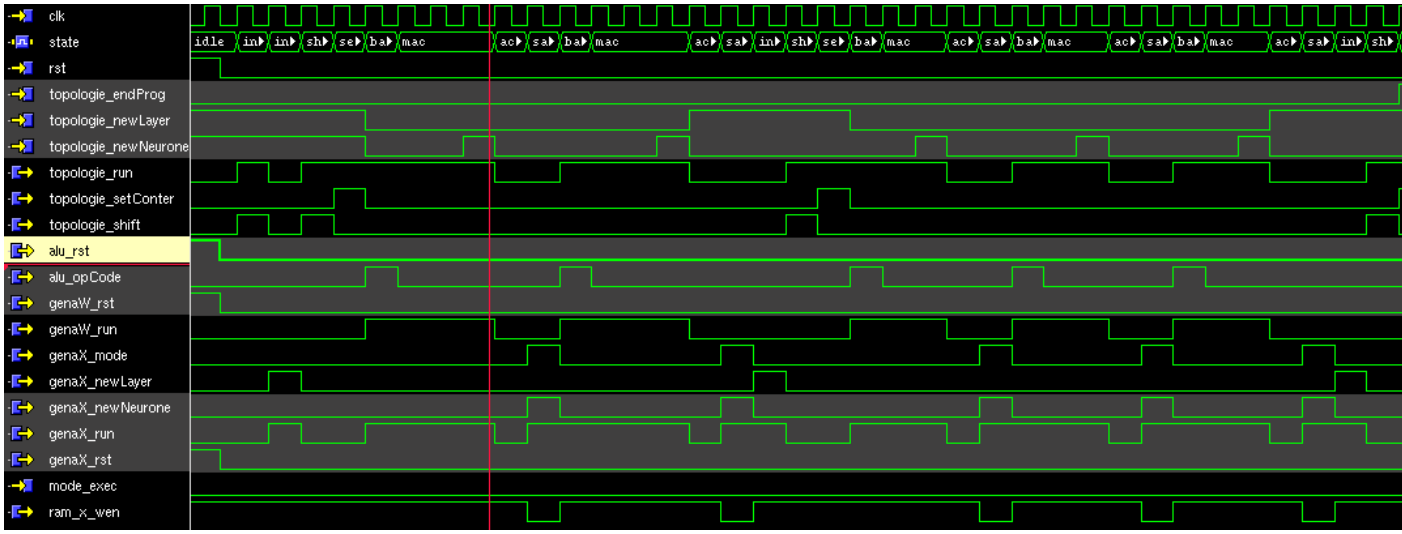


Figure 3-17 La machine d'état



3-18 waveform du la séquence d'exécution de CTR

III. Réalisation de la rétro-propagation

L'étape d'apprentissage consiste, d'abord, à présenter l'exemple (couple entrée E et sortie désirée D) à faire apprendre au réseau, puis à calculer l'écart δ entre la valeur désirée D et la valeur effective de la sortie du réseau. Vient ensuite l'exécution de l'algorithme de rétro-propagation pour répercuter la part d'erreur δi de chaque neurone i de la couche j sur les neurones de la couche j-1 (partant de la couche de sortie et arrivant à la couche d'entrée, en passant par les couches cachées) et ce, afin de corriger en conséquence les poids synaptiques de chaque neurone.

1. L'algorithme d'apprentissage

Cet algorithme se déroule en cinq étapes :

- Étape 1 : initialisation des poids du réseau (généralement par des valeurs aléatoires)
- Étape 2 : initialisation des vecteurs d'entrée du réseau et de la sortie désirée
- Étape 3 : propagation de l'entrée vers la sortie (obtention de la sortie effective)
- Étape 4 : calcul de l'erreur totale et rétro-propagation pour mise à jour des poids synaptiques des neurones.
  - Sous-étape 4.1 : calcul de l'erreur totale:

    Erreur = $\frac{1}{2} \sum (d_i - X_i)^2$ ($i \in [1,...,ns]$ avec ns nombre de neurones de la couche de sortie)
  - Sous-étape 4.2 : calcul de l'erreur de chaque neurone i appartenant à la couche de sortie

    $\delta_i = X_i (1 - X_i) (d_i - X_i)$
  - Sous-étape 4.3 : Pour chaque couche cachée, calcul de l'erreur de chaque neurone i appartenant à la couche

    $\delta_i = X_i (1 - X_i) \sum_j \delta_j W_{ij}$ ($j \in [1,...,np]$ avec np nombre de neurones (de la couche suivante) qui sont connectés à la sortie du neurone i).
  - Sous-étape 4.4 : mise à jour des poids synaptiques
    - $\Delta W_{ki} = \eta \delta_k X_i$ (W_{ki} est le poids qui relie le neurone k au neurone i et η le pas d'apprentissage)
    - $W_{ki}(t+1) = W_{ki}(t) + \Delta W_{ki}$
- Étape 5: analyse de convergence pour décider de la fin de l'apprentissage, ou de l'itération à partir de l'étape 2.

Les étapes 1 et 2 se limitent donc, à des opérations d'initialisation. Les étapes 3 et 4 correspondent à des opérations de calcul et de correction des poids synaptiques des neurones ; ces opérations sont coûteuses en temps et sont câblées. Enfin, l'étape 5 est une étape de test ; la décision est arrêtée par logiciel.

2. Traitement des étapes 1 et 2 : Stockage des valeurs entrée du réseau et sortie désirée ;

Rappelons que la mémoire RAM X (voir architecture neuronale paragraphe II.2) est une mémoire $X_{i,j}$ où j correspond au numéro de la couche et i correspond au rang du neurone dans la couche. La valeur $X_{i,j}$ représente précisément la sortie dudit neurone (i,j). L'initialisation du vecteur d'entrée s'effectue dans la colonne 0. Pour des raisons pratiques qui apparaîtront évidentes par la suite, le vecteur de sortie désirée sera placé dans la colonne qui suit immédiatement la colonne des sorties effectives obtenues par la propagation des entrées (figure 3-19)

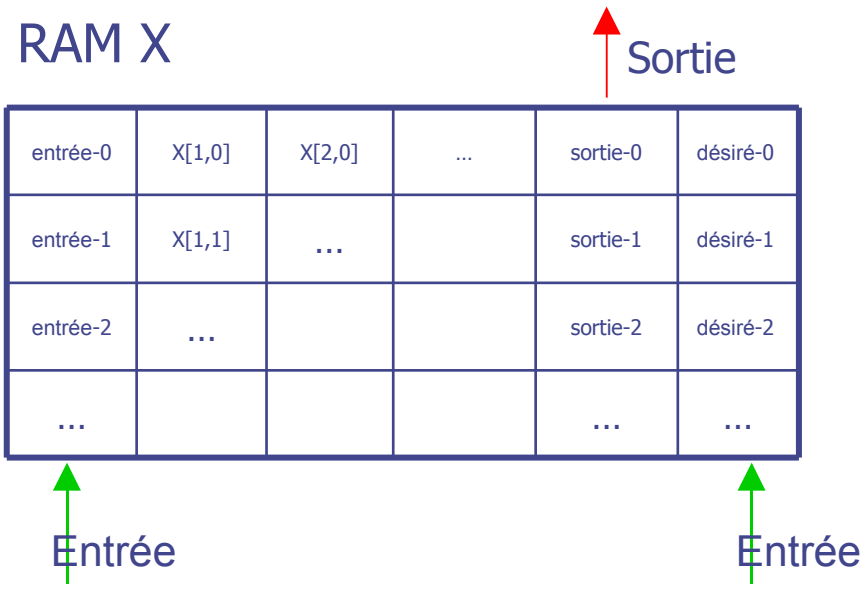
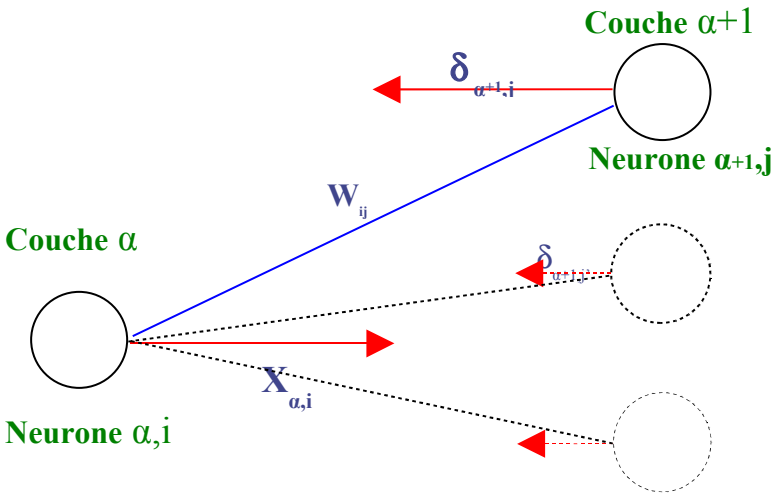


Figure 3-19 Stockage des entrée / sortie dans la mémoire X

3. Traitement de la sous-étape 4.3 : Calcul de l'erreur de chaque neurone i appartenant à la couche cachée;

La figure 3-20 rappelle les diffèrent paramètres intervenant dans le calcul de l'erreur de chaque neurone en phase de rétro-propagation, selon la formule : $\delta_i = X_i\,(1 - X_i)\,\Sigma_j\,\delta_j\,W_{ij}$

dans laquelle il faut remplacer δ_i par $\delta_{\alpha,i}$, X_i par $X_{\alpha,i}$ et δ_j par $\delta_{\alpha+1,j}$



$\delta_{\alpha+1,j}$: erreur du neurone j
$X_{\alpha,i}$: sortie du neurone i
W_{ij} : poids entre les neurones j et i

Figure 3-20 Sous-étape 4.3

Pour optimiser l'espace mémoire utilisé, les valeur $\delta_{\alpha,i}$ seront rangées dans les cases $X_{\alpha+1,i}$ de la mémoire RAM X selon la procédure explicitée par la figure 3-20.

Ce choix aboutit à remplacer progressivement les $X_{\alpha,i}$ par $\delta_{\alpha-1,i}$ en n'écrasant une valeur $X_{\alpha,i}$ qu'une fois tous les calculs qui l'utilisent auront été effectués.

Fin

$X_{0,0}$	$X_{1,0}$	…	…	$X_{f,0}$	d_0
$X_{0,1}$	$X_{1,1}$	…	…	$X_{f,1}$	d_1
…	…	$X_{\alpha,i}$	$X_{\alpha+1,i}$	…	…
$X_{0,n}$	…	…	…	$X_{f,n}$	d_n

de la propagation

1er

$X_{0,0}$	$X_{1,0}$	…	…	$X_{f,0}$	$\delta_{f,0}$
$X_{0,1}$	$X_{1,1}$	…	…	$X_{f,1}$	$\delta_{f,1}$
…	…	$X_{\alpha,i}$	$X_{\alpha+1,i}$	…	…
$X_{0,n}$	…	…	…	$X_{f,n}$	$\delta_{f,n}$

pas de la rétropropagation

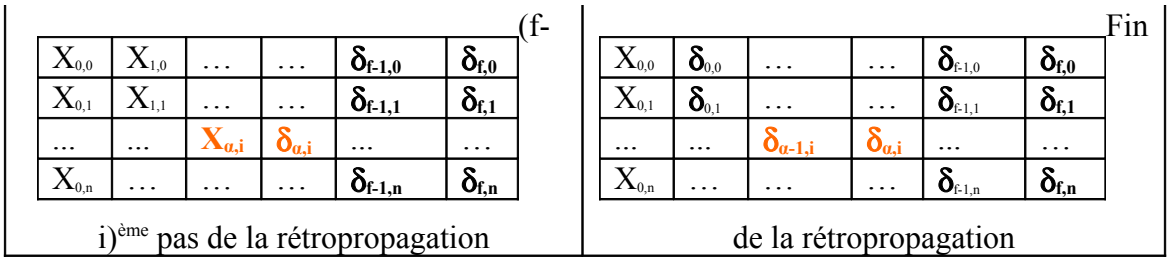


Figure 3-21 Organisation de la mémoire X

4. Traitement de la sous-étape 4.4 : Mise à jour des poids synaptiques.

La figure 3-22 rappelle les diffèrents paramètres intervenant dans le calcul de la mise à jour des poids W_{ij}, selon la formule :

$\mathbf{W_{ij}\ (t+1) = W_{ij}\ (t) + \eta\ \delta_j\ X_i}$ *(W_{ij} est le poids qui relie le neurone j au neurone i)*

dans laquelle il faut remplacer $\mathbf{X_i}$ par $\mathbf{X_{\alpha,i}}$ et $\boldsymbol{\delta_j}$ par $\boldsymbol{\delta_{\alpha+1,j}}$

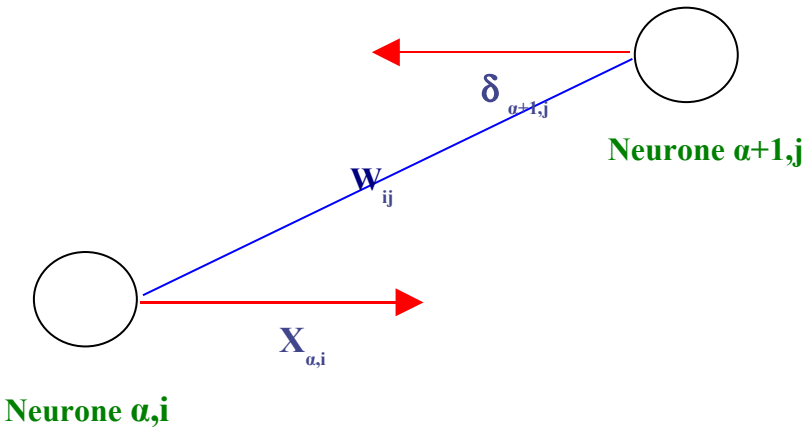


Figure 3-22 Sous-*étape 4.4*

En fait, pour optimiser à la fois le matériel et le temps d'exécution, les traitements des sous-étapes 4.3 et 4.4 sont effectuées simultanément : chaque fois que calcul de l'erreur $\boldsymbol{\delta_{\alpha,i}}$ terminé, cette valeur est stockée dans la case $\mathbf{X_{\alpha+1,i}}$ de RAM X et on poursuit immédiatement par le calcul des poids W_{ij} issus de ce neurone (reliant ce neurone à la couche suivante) et la mise à jour des cases correspondantes dans la mémoire W (voir architecture neuronale paragraphe II.2).

5. A propos de l'exécution des calculs proprement dits

Le chemin des données du bloc ALU donné en figure 3-14 ne prend en compte que les calculs en phase de propagation.

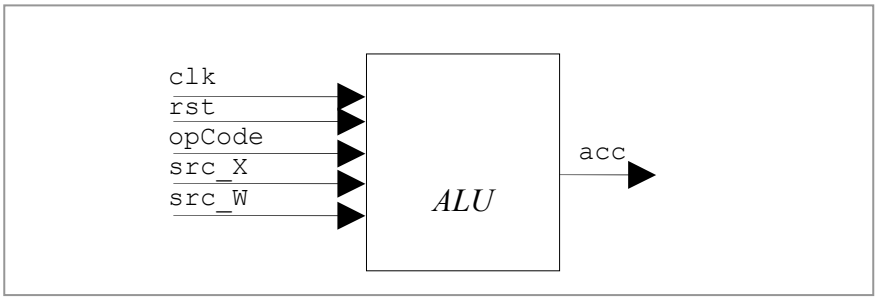


Figure 3-23 L'unité de calcul

Un enrichissement est nécessaire pour faire prendre en charge les calculs en phase de rétro-propagation. Nous proposons le rajout d'un deuxième registre acc2 (en plus du registre acc) et de rendre possible les opérations suivantes : (figure 3-23)

```
acc  ← W (chargement du l'entrée W dans le registre acc)
acc  ← X (chargement du l'entrée X dans le registre acc)
acc  ← acc + X * W  (cette opération est déjà prévue pour la
                     propagation)
acc2 ←  X  - X * X
acc  ←  W  + X * acc2
acc  ← acc * acc2
acc  ← acc - X
acc  ← W + acc2
acc2 ← décalage (X) (décalage de X  de -log2(η) position vers la
                     droite et chargement du registre dans acc)
```

- Application à la programmation de la rétro-propagation :

Étape 4.2 : calcul de l'erreur de chaque neurone de la couche de sortie

$$\delta_i = X_i (1 - X_i) (d_i - X_i)$$

acc ← X (la valeur di est ramenée à entrée de X)

acc ← acc - X (la valeur Xi est aiguillée ver l'entrée de X)

acc2 ← X - X * X (la valeur Xi est aiguillée ver l'entrée de X)

acc ← acc2 * acc (à l'issue de cette opération acc contient δi)

Cette valeur est alors rangée directement dans RAM X sans passer par la fonction d'activation.

Étape 4.3 : calcul de l'erreur de chaque neurone de la couche cachée

$$\delta_i = X_i (1 - X_i) \Sigma_j \delta_j W_{ij}$$

alu_rst (remise à zéro de acc et de acc2)

acc ← acc + X * W (X correspond δ_j)

acc2 ← X - X * X (X correspond Xi)

acc ← acc * acc2 (à l'issue de cette opération acc contient δi)

Cette valeur est alors rangée directement dans RAM X sans passer par la fonction d'activation.

Étape 4.4 : mise à jour des poids

$$W_{ij} (t+1) = W_{ij} (t) + \eta \delta_j X_i$$

-- mise à jour de W

acc2 ← décalage (X) (acc2 correspond à η **Xi**)

acc ← W + X * acc2 (X correspond à δ_j, à l'issue de cette opération acc contient le poids W)

Cette valeur est alors rangée directement dans RAM W sans passer par la fonction d'activation.

-- mise à jour de b (X_{bais}=1.0)

acc2 ← décalage (X) (acc2 correspond à $\eta\delta_j$)

acc ← W + acc2 (à l'issue de cette opération acc contient le poids b)

Cette valeur est alors rangée directement dans RAM W sans passer par la fonction d'activation.

A chaque cycle d'horloge, on peut n'utiliser qu'un additionneur et un multiplicateur. Dans l'implémentation matérielle que nous avons réalisée, nous avons préféré laisser au synthétiseur la liberté d'ajouter de la logique et/ou redondance aux fins d'augmenter la fréquence d'horloge ou pour résoudre des problèmes de routage (problème de planarité dans les FPGA)

Le pas η d'apprentissage est implémenté actuellement sous la forme d'un décalage fixe Il peut être implémenté sous la forme d'un décalage dynamique à travers l'utilisation d'un barrel shift, ce qui offrirait plus de flexibilité.

6. Enrichissements nécessaires au niveau des générateurs d'adresses et de la machine d'état

a. Enrichissements au niveau de l'explorateur de la topologie du réseau

Cet explorateur génère des signaux pour informer le CTR sur la topologie du réseau :

- Pour générer le signal « New-neurone », On utilise un compteur updown à la place du compteur simple.
- Pour générer le signal « New-layer » », On utilise un compteur updown à la place du compteur simple.

La valeur des deux compteurs est utilisée pour :

- gérer la mémoire T
- calculer le saut de pas pour le générateur W
- le chargement des registres du générateur X

b. Enrichissements au niveau du générateur d'adresse W *(gen_adr_w)*

- On ajoute un soustracteur pour faire les sauts de pas dont la valeur est obtenue à partir du bloc topologie.
- On ajoute un registre pour charger et sauvegarder la valeur initiale.
- On ajoute la possibilité d'incrémentation ou de décrémentation des registres.

c. Enrichissements au niveau du Générateur d'adresse X *(gen_adr_x)*

Dans la rétro-propagation, les registres doivent être chargés à partir du bloc topologie. Pour cela :

- On ajoute la possibilité d'incrémentation ou de décrémentation des registres.
- On ajoute 3 compteurs utilisés comme : adresse couche courante, adresse neurone courant et un compteur de neurones pour le calcul.
- On ajoute 2 registres qui seront utilisés comme : adresse couche précédente et adresse couche avant précédente.
- Enfin, on ajoute un multiplexeur pour la sélection de la sortie « adresse x ».

d. Enrichissements au niveau de la machine d'état

Tous les blocs sont contrôlés par le CTR qui est guidé par le bloc topologie. La machine d'état est complétée comme suit :

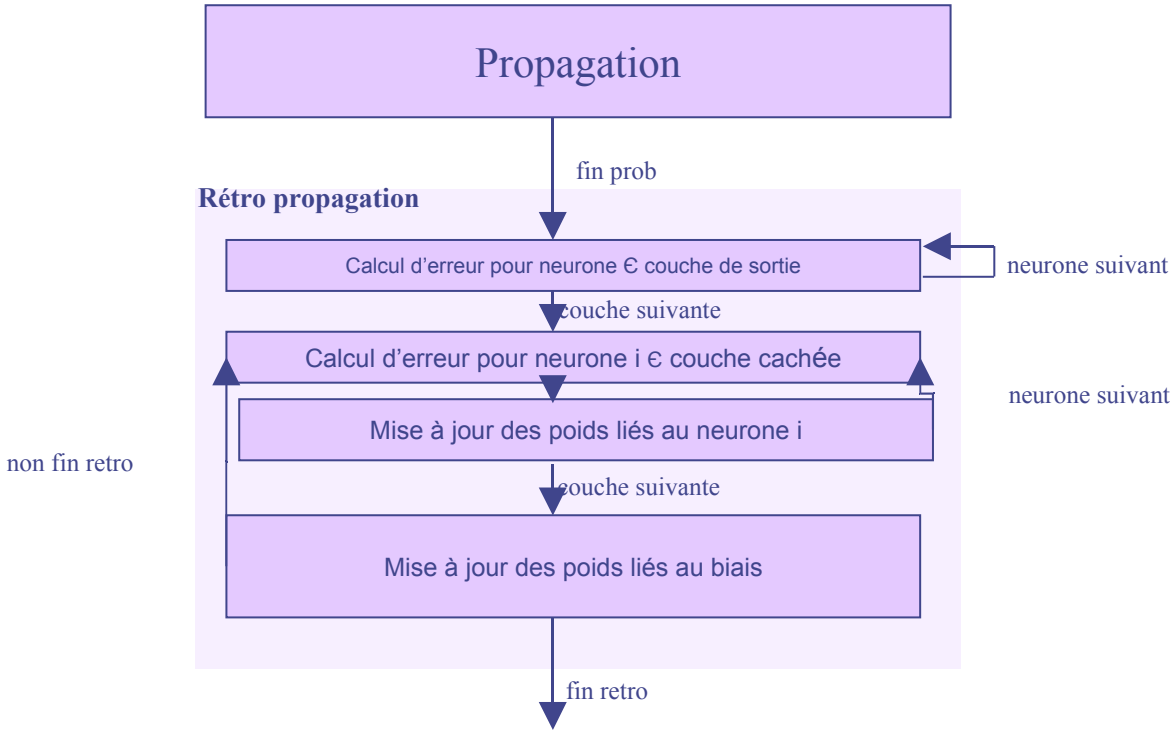


Figure 3-24 Machine d'état du CTR complétée par les besoins de la rétro-propagation

7. Etude de cas

Nous reprenons en figure 3-24 le réseau déjà étudie dans la paragraphe 4 du ce chapitre, pour illustrer dessus le déroulement de l'algorithme de rétro-propagation. Ce réseau est représenté dans la mémoire T par 4 mots contenant la suite des valeurs 3, 2, 3, 0. (réseau à 3 couches, avec 3 neurones en couche d'entrée, 2 neurones en couche cachée et 3 neurones en couche de sortie; le 0 indique la fin du réseau).

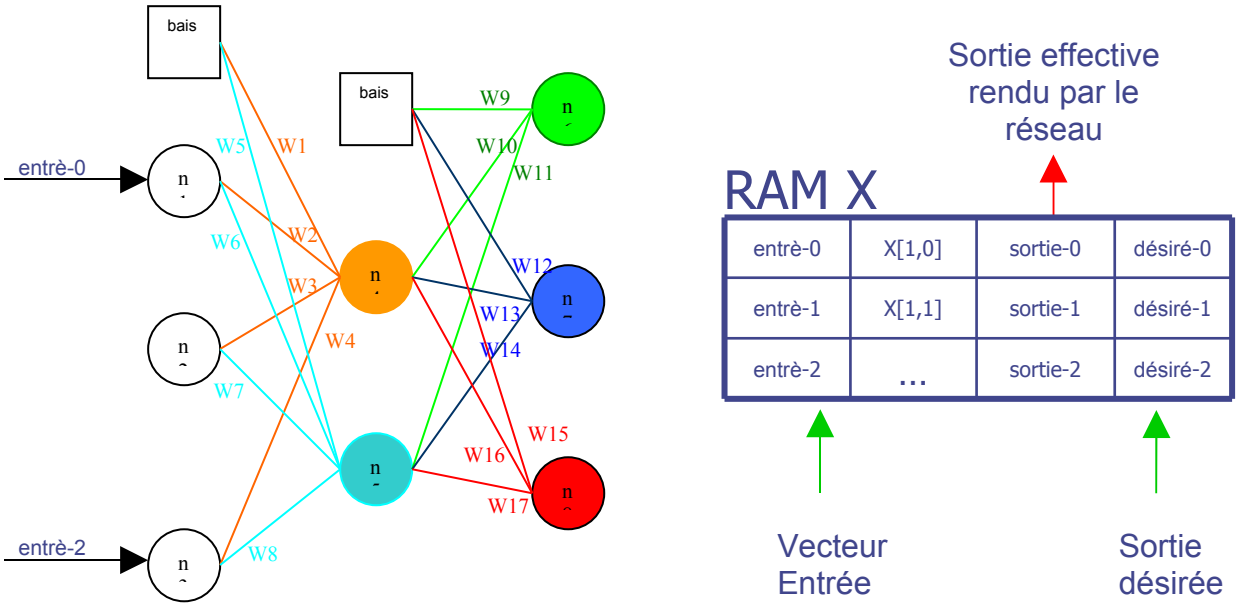


Figure 3-25 Réseau d'étude (rétro-propagation)

Déroulement de l'exécution de l'algorithme de rétro-propagation :

Les valeurs entrée du réseau sont chargées directement comme valeurs de sortie X[0,0], X[0,1] et X[0,2] du neurone de la couche 0. De même les valeurs de sortie désirées sont chargées aux positions X[3,0], X[3,1] et X[3,2], Le processeur effectue alors le traitement suivant :

- Pour la couche de sortie :

La formule à calculer par le processeur est :

$$\delta_i = X_i\,(1 - X_i)\,(d_i - X_i)$$

Pour le neurone n8 :

L'UAL calcule la somme pondérée :

acc ← d_2 (d_2 correspond à X[3,2])

acc ← acc - X_8 (X_8 correspond à X [2,2])

acc2 ← X_8 - X_8 * X_8

acc ← acc2 * acc

on doit bypasser la fonction d'activation et enregistrer acc represemtant écart δ_8 dans RAM X

X [3,2] ← acc

Pour cette exécution, les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : (ici, les poids W_{ij} ne sont pas impliqués dans le calcul)

Générateur d'adresse X : [3,2] [2,2] (save->) **[3,2]**

d_i	X_i	δ_i	**[nomage dans la formule]**
d_2	X_8	δ_8	**[nomage dans notre cas d'étude]**

Pour le neurone n7 :

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : (ici, les poids W_{ij} ne sont pas impliqués dans le calcul)

Générateur d'adresse X : [3,1] [2,1] (save->) **[3,1]**

Pour le neurone n6 :

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : (ici, les poids W_{ij} ne sont pas impliqués dans le calcul)

Générateur d'adresse X : [3,0] [2,0] (save->) **[3,0]**

- Pour la couche 1 :

Les formules à réaliser par le processeur sont :

(1) - *Calcul d'erreur* - $\delta_i = X_i\,(1 - X_i)\,\Sigma_j\,\delta_j\,W_{ij}$

(2) - *mise à jour des poids* - $W_{ij}\,(t+1) = W_{ij}\,(t) + \eta\,\delta_j\,X_i$

Pour le neurone n5 :

- *Calcul d'erreur :* $(\delta_i = X_i\,(1 - X_i)\,\Sigma_j\,\delta_j\,W_{ij})$

alu_rst (acc←acc2←0)

acc2 ← x_5 − x_5 * x_5 (x_5 correspond à X [1,2])

acc ← acc + δ_8 * W_{17} (δ_8 correspond à X [3,2])

acc ← acc + δ_7 * W_{14} (δ_7 correspond à X [3,1])

acc ← acc + δ_6 * W_{11} (δ_6 correspond à X [3,0])

acc ← acc * acc2

on doit bypasser la fonction d'activation et enregistrer acc represemtant écart δ_5 dans RAM X

X [2,2] ← acc

Pour cette exécution, les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W :	-	17	14	11	-
Générateur d'adresse X :	[1,2]	[3,2]	[3,1]	[3,0] (save->)X[2,2]	
	$x_i(1-x_i)$	$\delta_j\,W_{ij}$	$\delta_j\,W_{ij}$	$\delta_j\,W_{ij}$	δ_i
	$x_5(1-x_5)$	$\delta_8\,W_{17}$	$\delta_7\,W_{14}$	$\delta_6\,W_{11}$	δ_5

-mise à jour des poids w17, w14 et w 11 : $(W_{ij}\,(t+1) = W_{ij}\,(t) + \eta\,\delta_j\,X_i)$

acc2 ←décalage(X_5) (acc2 correspond à $\eta\,X_5$ et X_5 correspond à X[1,2])

acc ← W_{17} + δ_8 * acc2 (δ_8 correspond à X [3,2])

W [17] ← acc (mise à jour du poids W17 dans la mémoire W)

acc ← W_{14} + δ_7 * acc2 (δ_8 correspond à X [3,1])

W [14] ← acc (mise à jour du poids W14 dans la mémoire W)

acc ← W_{11} + δ_6 * acc2 (δ_8 correspond à X [3,0])

W [11] ← acc (mise à jour du poids W11 dans la mémoire W)

Pour cette exécution, les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : - 17 (save->) 17 14 (save->)14 11 (save->)11

Générateur d'adresse X : [1,2] [3,2] - [3,1] - [3,0] -

X_i	$W_{ij} + \eta\ \delta_j\ X_i$	$W_{ij} + \eta\ \delta_j\ X_i$	$W_{ij} + \eta\ \delta_j\ X_i$
X_5	$W_{17} + \eta\ \delta_8\ X_5$	$W_{14} + \eta\ \delta_7\ X_5$	$W_{11} + \eta\delta_6\ X_5$

Pour le neurone n4 :

- Calcul d'erreur

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : 16 13 10 -

Générateur d'adresse X : [3,2] [3,1] [3,0] (save->)**[2,2]**

-mise à jour des poids w16, w13 et w10 :

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : - 16 (save->)16 13 (save->)13 10 (save->) 10

Générateur d'adresse X : [1,2] [3,2] - [3,1] - [3,0] -

Pour e biais :

- Calcul d'erreur (le biais ne rétro-propage pas d'erreur)

-mise à jour des poids w15, w12 et w 9 :

acc2 ← décalage (δ_8) (acc2 correspond à $\eta\delta_8$

acc ← W_{15} + acc2

W [15] ← acc (mise à jour du poids W15 dans la mémoire W)

acc2 ← décalage (δ_7) (acc2 correspond à $\eta\delta_7$)

acc ← W_{12} + acc2

W [12] ← acc (mise à jour du poids W12 dans la mémoire W)

acc2 ← décalage (δ_6) (acc2 correspond à $\eta\delta_6$)

acc ← W_9 + acc2

W [9] ← acc (mise à jour du poids W9 dans la mémoire W)

Pour cette exécution, les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : 15 (save->)15 12 (save->)12 9 (save->)9

Générateur d'adresse X : [3,2] - [3,1] - [3,0] -

$W_{ij} + \eta\ \delta_j$ $W_{ij} + \eta\ \delta_j$ $W_{ij} + \eta\ \delta_j$

$W_{15} + \eta\ \delta_8$ $W_{12} + \eta\ \delta_7$ $W_9 + \eta\ \delta_6$

<u>- Pour la couche 0 :</u>

Pour le neurone n3 :

- Calcul d'erreur

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : 8 4 -

Générateur d'adresse X : [2,1] [2,0] (save->)**[1,2]**

-mise à jour des poids w8 et w4 :

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : - 8 (save->)8 4 (save->)4

Générateur d'adresse X : [0,2] [2,1] - [2,0] -

Pour le neurone n2 :

- Calcul d'erreur

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : 7 3 -

Générateur d'adresse X : [2,1] [2,0] (save->)**[1,1]**

-mise à jour des poids w7 et w3 :

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : - 7 (save->)7 3 (save->)3

Générateur d'adresse X : [0,1] [2,1] - [2,0] -

Pour le neurone n1 :

- Calcul d'erreur

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : 6 2 -

Générateur d'adresse X : [2,1] [2,0] (save->)**[1,0]**

-mise à jour des poids w6 et w2 :

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : - 6 (save->)6 2 (save->)2

Générateur d'adresse X : [0,0] [2,1] - [2,0] -

Pour le biais :

- Calcul d'erreur (le biais ne rétro-propage pas d'erreur)

-mise à jour des poids w5 et w1 :

Les blocs gen_adr_w et gen_adr_x génèrent les séquences d'adresses suivantes :

Générateur d'adresse W : 5 (save->)5 1 (save->)1

Générateur d'adresse X : [2,1] - [2,0] -

IV. Résultats de la synthèse

L'architecture neuronale ainsi complétée a été implémentée sur FPGA. Nous donnons, ci-dessous, les caractéristiques et résultats de cette implémentation.

- Outil de synthèse : Symplicity, version 7.3, Build 192R
- FPGA : Apex [ep20k100eqc240-2x]
- Contrainte de synthèse : Frequency 20.0 MHz (la carte d'étude dont nous disposons fonctionne à 20MHz)

Paramètre générique	propagation	Propagation et rétropropagation
--alu paramètre *--la position de la virgule à l'entrée de l'alu* fixedPoint : integer:=10; *--taille de l'accumulateur* bitAcc : integer:=16; *--forme de la sortie de alu (taille de la sortie apreV+ avanV)* apreV : integer:=2; avanV : integer:=2; **--weight parameter** *--taille des poids W* bitWdata: integer:=16; *--taille des adresses des poids W* bitWaddr: integer:=8; *--taille de la mémoire W* memWlenght : integer:=200; **--X parameter** *--taille de la sortie des neurones X* bitXdata : integer:=16; *-- taille des adresses neurones* bitXadrN : integer:=3; *-- taille des adresses couches* bitXadrL : integer:=3; *--taille de la mémoire X* memXlenght : integer:=128; **--fonction d'activation parameter** *--taille de la sortie de la fonction d'activation* bitAVFdata : integer:=16; *--taille de l'entrée de la fonction d'activation* bitAVFadrs : integer:=4; *--taille de la mémoire F* memAVFlenght : integer:=16; **--topology of the network** *--taille mot mémoire T* bitFFdata : integer:=4; *--nombre de mots mémoire T utilisés (adresses)* bitFFaddr : integer:=4; *--taille de la mémoire T* memFFlenght: integer:=16	Frequency : 25.0 MHz Total LUTs: 754 of 4160 (18%) Logic resources: 754 ATOMs of 4160 (18%) ATOM count by mode: normal: 474 arithmetic: 251 counter: 22 qfbk_counter: 7 ESBs: 6 (23% of 26)	Frequency : 23.9 MHz Total LUTs: 2348 of 4160 (56%) Logic resources: 2348 ATOMs of 4160 (56%) ATOM count by mode: normal: 1276 arithmetic: 1048 counter: 21 qfbk_counter: 3 ESBs: 8 (30% of 26)

--alu parameter *--la position de la virgule à l'entrée d'alu* fixedPoint : integer:=3; *--taille de l'accumulateur* bitAcc : integer:=8; *--forme de la sortie de alu (taille de la sortie apreV+ avanV)* apreV : integer:=3; avanV : integer:=3; **--weight parameter** *--taille des poids W* bitWdata: integer:=8; *--taille des adresses des poids W* bitWaddr: integer:=8; *--taille de la mémoire W* memWlenght : integer:=32; **--X parameter** *--taille de la sortie des neurones X* bitXdata : integer:=8; *-- taille des adresses neurones* bitXadrN : integer:=4; *-- taille des adresses couches* bitXadrL : integer:=4; *--taille de la mémoire X* memXlenght : integer:=32; **--fonction d'activation parameter** *--taille de la sortie de la fonction d'activation* bitAVFdata : integer:=8; *--taille de l'entrée de la fonction d'activation* bitAVFadrs : integer:=6; *--taille de la mémoire F* memAVFlenght : integer:=64; **--topology of the network** *--taille mot mémoire T* bitFFdata : integer:=4; *--nombre de mots mémoire T utilisés (adresses)* bitFFaddr : integer:=4; *--taille de la mémoire T* memFFlenght: integer:=16	Frequency : 37.0 MHz Total LUTs: 381 of 4160 (9%) Logic resources: 381 ATOMs of 4160 (9%) ATOM count by mode: normal: 275 arithmetic: 78 counter: 21 qfbk_counter: 7 ESBs: 4 (15% of 26)	Frequency : 36.1 MHz Total LUTs: 984 of 4160 (23%) Logic resources: 984 ATOMs of 4160 (23%) ATOM count by mode: normal: 623 arithmetic: 339 counter: 19 qfbk_counter: 3 ESBs: 4 (15% of 26)

Tableau 3-1 Résultats de synthèse

V. Vers des architecteurs parallèles

Le FPGA utilisé dans l'implémentation actuelle (dont les résultats sont rapportés dans le tableau 3-1 ci-dessus) comprend 100K portes logiques. Or, il est devenu courant maintenant de disposer d'un FPGA ayant une capacité au mois 100 fois plus grande. Dès lors, on peut envisager une nouvelle implémentation qui permet d'introduire du parallélisme et, donc, une plus grande flexibilité.

1. Parallélisme dans les réseaux de neurones à couches

De fait, les réseaux de neurones artificiels sont connus pour être des modèles intrinsèquement parallèles. Nous avons à dessein séparé les trois mémoires W, T et X. On peut entrevoir trois types de parallélisme : parallélisme intra-neurone, parallélisme des données et parallélisme inter-couche.

a. Parallélisme intra-neurone

C'est un parallélisme à l'intérieur du neurone. En effet, on peut paralléliser la somme des produits (c'est-à-dire, faire les multiplications en parallèle puis accélérer les additions).

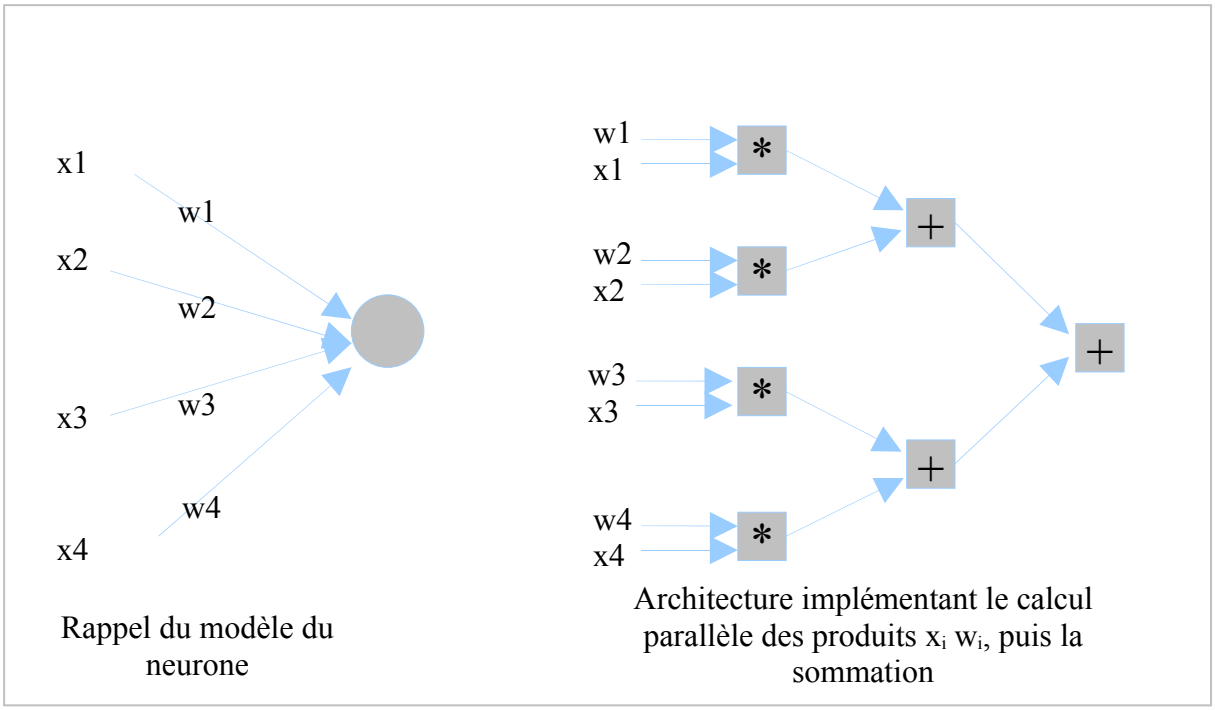


Figure 3-26 Parallélisme intra-neuronal

b. Parallélisme des données

Le calcul de la sortie de chaque neurone d'une couche i (i.e. traitement intra-neurone) utilise, en fait, les sorties des neurones de la couche i-1. On peut, par conséquent, effectuer ce calcul en parallèle pour tous les neurones de la couche.

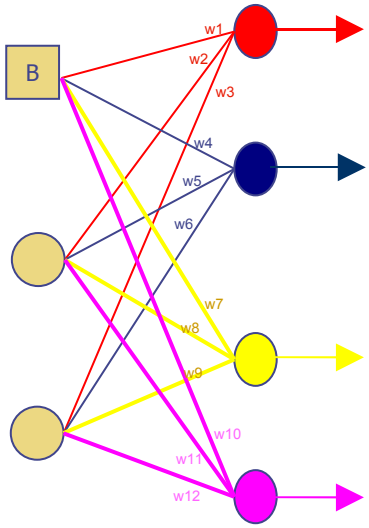


Figure 3-27 Parallélisme des données

c. Parallélisme inter-couches

L'évolution des traitements depuis la couche d'entrée jusqu'à la couche de sortie constitue l'étape de la propagation. La rétro-propagation, quant à elle, s'effectue selon le chemin inverse. On peut envisager d'effectuer aussi bien la propagation que la rétro-propagation selon un traitement en pipe-line. L'unité d'un tel pipe-line peut être la couche. Mais on peut aussi penser à élargir cette unité à un ensemble de couches, ce qui revient à partitionner le réseau en des ensembles équilibrés et à affecter un processeur au traitement de chaque unité (Figure 3-28). Ce partitionnement permettra un gain substantiel en temps d'exécution. Le degré du pipe-line va dépendre, en fait, du nombre de portes logiques dont on dispose sur le FPGA ou de la surface de la puce dans le cas d'un ASIC). Ce degré de pipe-line peut aussi être imposé par les temps globaux d'exécution souhaités ou exigés.

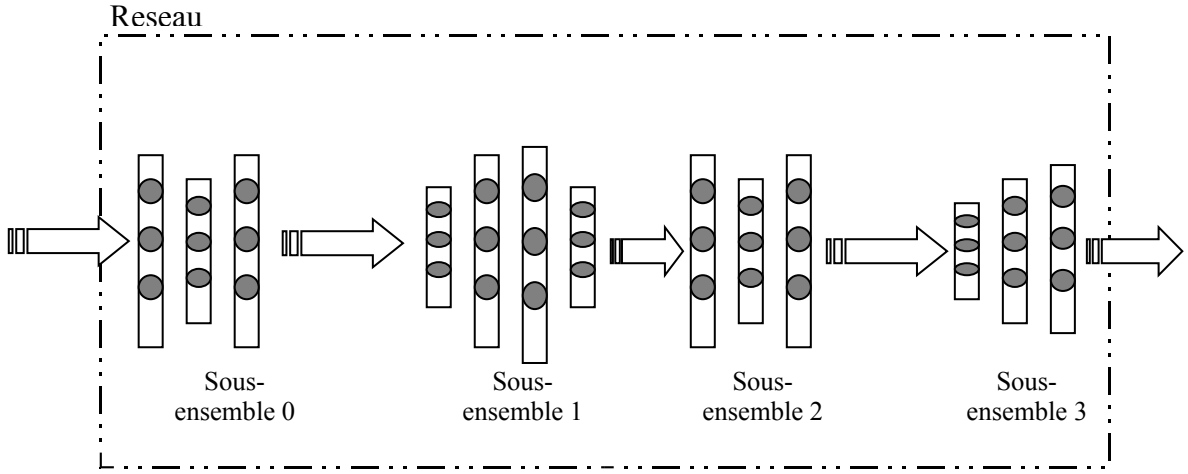


Figure 3-28 Parallélisme inter-couche

d. conclusion

La démarche En fait, la mise en œuvre de ces trois types de parallélisme n'est pas aussi évidente qu'elle ne le parait. Une étude réelle et détaillée va s'affronter à trois difficultés majeures :

- le première est liée au fait que le traitement en propagation diffère en certain point importants, du traitement de la rétro-propagation. De ce fait, une architecture fine et appropriée à l'accélération de l'exécution de la propagation peut ne pas convenir qu'aux besoins de l'accélération de la rétro-propagation.
- la deuxième difficulté majeure est liée à l'organisation des données en mémoire. Un parallélisme effectif exige que soit résolu le problème de la gestion parallèle des accès aux données. Or, les combinaisons de données parallèle utilisées en propagation ne sont pas les mêmes que celles utilisées en rétro-propagation.
- la troisième difficulté est liée à l'inadaptation de l'algorithme de la rétro-propagation au principe de calcul par pipe-line, plusieurs travaux on étudie ce dilemme et proposent des versions modifie du la retro-propagation [Rumelhar al, 86] [Falhman al, 88]. Le problème vient du fait qu'on ne peut pas faire plusieurs mises à jour en même temps.

Conclusion

L'idée du travail réalisé est de développer une architecture de base modélisant les réseaux de neurones multicouches. À partir de cette architecture, on peut construire d'autres architectures plus performantes en augmentant le degré de parallélisme. Les propositions avancées à la fin du 3ème chapitre sont à approfondir dans ce sens.

La propriété intellectuelle développée NEURONA nécessite une couche application pour superviser le réseau de neurones et jouer le rôle d'intermédiaire avec l'environnement extérieur. Il serait fort intéressant de concevoir un environnement permettant de co-simuler l'ensemble du système. En effet, la simulation et co-simulation paraissent demeurer la seule voie possible pour approcher les architectures dédiées en l'absence de méthodes théoriques. De même, un choix direct de précision limitée risque de bloquer la convergence du processus d'apprentissage, la seule solution pour cerner la précision appropriée est la simulation expérimentale.

La couche application qui englobe le processeur NEURONA permet d'implémenter les algorithmes de construction dynamique du réseau (algorithmes incrémentaux) et les algorithmes d'ajustement des liens entre neurones (algorithmes d'élagage). Ces méthodes utilisent la rétro-propagation du gradient pour agir sur l'architecture du réseau.

Une autre voie consiste à implémenter d'autres méthodes d'apprentissage. En effet, en s'appuyant sur le concept de la reconfiguration des FPGA, on peut choisir et charger à la volée (on-line) le composant matériel implémentant la méthode d'apprentissage à appliquer.

Bibliographie

[MACCULLOCH et al., 43] W. S. MACCULLOCH et W. H. PITTS, A logical calculus of the ideas immanent in nervous activity, Bulletin of Math. Biophysics, 5, p. 115-133, 1943.

[Trivedi, 77] K. S. Trivedi and M. D. Ercegovac. On-line Algorithms for Division and Multiplication. IEEE Transactions on Computers, C-26(7):681-687, 1977.

[Hopfield, 86]- J. J. Hopfield and D. W. Tank, ``Computing with neural circuits: A model," science, vol. 233, pp. 625-633, August,1986.

[Rurnelhartet al, 86] D.E. Rurnelhart, J.L. McClelland et the PDP Research Group. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume. MIT Press, Cambridge, 1986.

[Rumelhar al, 86] D.E. Rumelhar, G.E.Hinton, and R.J. Williams : Learning internal representations by error backpropagation, Parallel Distributed processing, Vol 1, MIT Pres. Cambridge, MA,1986, pp 318-362.

[Falhman al, 88] S.E. Falhman: Faster learning variation on backpropagation: An empirical study, Proc. 1988 Connectionist Models Summer School 1988, pp 38-50

[Hecht-Nielsen, 89] R. Hecht-Nielsen. Theory of the backpropagation neural network.
IEEE International Joint Conference on Neural Networks, volume 1,pages 593-605, New York, 1989.(Washington 1989),.

[Widrow, 90] B. Widrow and M. A. Lehr. 30 Years of Adaptativ Neural NetWorks: Perceptron, Madaline and Backpropagation. Proc. IEEE, 78(9):1415-1442, 1990

[karnin, 90] Ehud D.Karnin. A simple procedure for pruning Back-propagation trained neural networks, IEEE Transactions on Neural Networks,1(2):239-242,1990

[Leonardo M, 91] Leonardo M. Reyneri and Enrica Filippi. An Analysis on the Performance of Silicon Implementations of Backpropagation Algorithms for Artificial Neural Networks. 1991 IEEE Transactions on Computers, 40(12):1380--1389, December 1991.

[Yun Xie, Marwan Jabri, 91] Analysis of the Effects of Quantization in Multi-Layer Neural Networks Using Statistical Model (1992)

[Hoehfeld et Fahlman, 92] Markus Hoehfeld et Scott E. Fahlman. Learning with limited numerical precision using the Cascade-correlation algorithm. IEEE Transactions on Neural Networks, 3(4):602-611,1992

[Blanz, 92] COX, C.E.; BLANZ, W.E. GANGLION-A FAST FIELD-PROGRAMMABLE GATE ARRAY IMPLEMENTATION OF A CONNECTIONIST CLASSIFIER . SOLID-STATE CIRCUITS, IEEE JOURNAL OF (MARCH 1992 Volume 27 Number 3)

[J.L. Holt, 93] J.L. Holt and J-N Hwang. Finite precision error analysis of neural network hardware implementations. IEEE Transactions on Computers', 42:1380-1389, 1993.

[Elredge, 94] J.G. Elredge and B.L.Hutchings : RRANN A HARDWARE IMPLEMENTATION OF THE BACKPROPAGATION ALGORITHM USING RECONFIGURABLE FPGAS , IEEE int cof, Neural Network june 1994

[LeBouquin, 94]- J-P. LeBouquin, ``IBM Microelectronics ZISC, Zero Instruction Set Computer'', Proc. of the World Congress on Neural Networks, Supplement,
San Diego, 1994.

[Lindsey, 94] C.S. Lindsey, Th. Lindblad, Review of Hardware Neural Networks – a Users erspective, proceedings of the 3rd Workshop on Neural Networks: From Biology to High Energy Physics, Isola d'Elba, Italy, Sept. 26-30, 1994.

[Marcelo, 94] Marcelo H. Martine ,A Reconfigurable Hardware Accelerator for Back-Propagation Connectionist Classifiers (1994), university of California Santa Cruz

[Aaron, 94] Aaron T. Ferrucci ACME: A Field-Programmable Gate Array Implementation of a Self-Adapting and Scalable Connectionist Network (1994), university of California Santa Cruz

[Heemskerk, 95] Jan N. H. Heemskerk , Overview of neural hardware ,Unit of Experimental and Theoretical Psychology Leiden University, P.O.Box 9555, 2300 RB Leiden The Netherlands

[Bishop, 97] Bishop, Christopher. Classification and Regression. In: Handbook of Neural Computation (section B6.2). E. Fiesler and R. Beale (Eds.) Institute of Physics and Oxford University Press. New York, NY - U.S.A., 1997. Web: http://www.idiap.ch/publications/fiesler-96.1.bib.abs.html
http://www.oupusa.org/acadref/nc_accs.html

[Chentouf, 97] Chentouf, Rachida. Construction de Réseaux de Neurones Multicouches pour l'Approximation. Thèse de Doctorat en Sciences Cognitives, Laboratoire TIRF - INPG,Grenoble - France, Mars 1997.

[GIRAU, 99] Bernard GIRAU Du parallélisme des modèles connexionnistes à leur implantation parallèle, L'école normale supérieur de Lyon (1999)

[Beuchat, 01] Jean-Luc Beuchat. Etude et conception d'opérateurs arithmétiques optimisés pour circuits programmables. Thèse de doctorat, Ecole Polytechnique Fédérale de Lausanne, 2001. Thèse No 2426. BibTeX

[Beuchat, 02] Jean-Luc Beuchat et Arnaud Tisserand. Opérateur en-ligne sur FPGA pour l'implantation de quelques fonctions élémentaires. Actes de la conférence Sympa'8 - Symposium en Architectures Nouvelles de Machines, pages 267-274, 2002. BibTeX

[Beuchat] JL.Beuchat, J-O.Haenni and E. Sanchez, Hardware Reconfigurable Neural Networks