MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DE BLIDA
INSTITUT DE SCIENCES EXACTES
DEPARTEMENT DE MATHEMATIQUES

THESE DE MAGISTER

SPECIALITE: MATHEMATIQUES APPLIQUEES

OPTION: MODELISATION MATHEMATIQUE
ET TECHNIQUES DE DECISION

INTITULE:
LES RESEAUX DE NEURONES SEQUENTIELS : APPLICATION AUX MODELES DE MARKOV

PAR: Mr. AIT AKKACHE
MUSTAPHA

Présentée devant le jury:

Président:	M. djeddi,	Maitre de conférence.	inelec
Rapporteurs:	A. Aissani,	Professeur.	Univ. Blida
	j-p. Asselin de Beauville,	Professeur,	Univ. tours
Examinateurs:	A. Guessoum,	Maitre de conférence.	Unv. Blida
	H. Meliani,	Maitre de conférence.	Univ. Blida
	M. El Bahi,	Charge de Recherche.	Univ. USTHB.

date de soutenance:07/04/ 1996

SOMMAIRE

Pages

INTRODUCTION GENERALE

1- INTRODUCTION ET POSITION DU PROBLEME:

L'idée d'une machine neuronale apparaît pour la première fois vers les années 1940, introduite par Mc culloch et Pitts [34]. Ces derniers cherchent à tirer de l'étude du système nerveux les principes de construction d'une telle machine. Par analogie avec les cellules nerveuses, ils introduisent la notion de neurone formel ou automate qui est l'unité de base du système caractérisée par son état binaire (activé et repos) et par les connexions qui la relient à d'autres unités. Les auteurs définissent une machine neuronale comme un ensemble d'automates, localement connectés, dont les états peuvent évoluer au cours du temps. Mc culloch et Pitts établissent alors qu'il est possible de représenter toutes fonctions calculables à l'aide de réseaux d'automates binaires et construisent une machine capable de reconnaître des formes déduites les unes des autres par des transformations simples. L'intérêt d'une telle approche s'accroît lorsque F.Rosenblatt [40] met en évidence des propriétés d'auto-organisation : en 1957 il propose un réseau d'automates : le PERCEPTRON capable d'apprendre à reconnaître des formes. Dans son livre « Principes de Neurodynamique » [41] il définit les bases théoriques du domaine. Il met en évidence les conditions pour lesquelles l'apprentissage d'une tâche est possible. Toutes ces idées sont liées à celle de la cybernétique dont le but est d'expliquer la régularité de phénomènes par l'organisation de l'information.

La plupart des résultats théoriques concernant ces notions figurent dans le PERCEPTRON [35], l'un des ouvrages les plus marquants des années 60. Ce livre propose une étude détaillée des possibilités liées à certaines architectures neuronales et son mérite a été de montrer que dans le domaine de l'apprentissage aucune machine universelle ne pouvait être construite : " Aucune machine ne peut apprendre à reconnaître un X, à moins qu'elle ne possède, au moins de manière

potentielle, un schéma pour représenter X ". Avant de résoudre un problème, il faut définir si c'est un problème particulier ou une classe de problème que l'on désire traiter, quelle partie du travail doit être confiée à l'utilisateur du réseau ainsi que le degré de liberté qui est lui est laissé. Le PERCEPTRON à su poser de manière claire les grands problèmes à résoudre pour rendre applicable les architectures neuronales à des problèmes réels. Les recherches de M. Minsky et S. Papert [35] ont amené à définir des notions fondamentales comme:

- L'ordre d'un problème qui est une mesure de sa difficulté.
- L'amplitude des coefficients numériques que le réseau manipule.

La généralisation de la démarche utilisée par F. ROSENBLATT a montré que l'on pourrait aborder le champ sous l'angle de problèmes d'optimisation, cette direction de recherches a permis de créer par la suite des domaines d'études comme la reconnaissance des formes la classification statistique et le traitement du signal. Tout en développant une théorie scientifique riche et rigoureuse, ces disciplines ont abandonné le champ ouvert par la notion de coopération entre éléments simples qui constitue l'originalité de la force des réseaux neuronaux.

Le mouvement connexioniste actuel est directement issu des travaux sur les premiers réseaux de neurones . Le renouveau de ces théories s'explique par les quantités gigantesques qui implique la manipulation des données dans des domaines comme la reconnaissance de la parole [10], [26],[45] ou d'image [8], et par la découverte de différents algorithmes d'apprentissage applicables aux réseaux de neurones. Le haut niveau de parallélisme qu'il est permis d'introduire dans la conception des calculateurs neuronaux les rend maintenant adaptables à la reconnaissance des formes complexe et à l'apprentissage.

Les Modèles Markoviens sont des processus stochastiques simples en application, riches en propriétés et fondés sur des bases mathématiques non compliquées. C'est pourquoi, ils permettent de modéliser plusieurs phénomènes physiques avec un grand succes. Ce qui explique leurs dominance dans les systèmes de file d'attente , de fiabilité , de reconnaissance de la parole et de l'image. Dans la littérature [5], [6], [10], [14], [22], on trouve plusieurs types de modèles de Markov. dont :

- Les modèles de Markov observables (ou encore les modelés de Markov)
- Les modèles de Markov cachés

Notre travail consiste a déterminer les liens entre les modèles de Markov et les réseaux de neurones artificiels , ou encore trouver des réseaux connexionnistes pour le traitement de certaines tâches (problèmes) markoviennes qui sont de nature séquentielle, en se basant sur les travaux de M. A. Manzoul et S. Mamoun [32] [33] ; Touzet [49]; et de H. Boulard [12], [13].

2- PRESENTATION DE CE MEMOIRE:

L'ordre de présentation adopté a pour but de mettre en valeur ce qui nous intéresse, et plus précisément, le traitement des modèles de Markov par les réseaux de neurones artificiels. Après le premier chapitre qui comporte les notions générales sur les réseaux de neurones et une synthèse sur les différents modèles à couches , nous présentons dans le deuxième chapitre, une étude théorique sur les réseaux connexionnistes séquentiels. Nous décrivons également dans cette partie, quelques modèles neuronaux issus de tentative effectuées pour résoudre certains problèmes de nature séquentielle. Le troisième chapitre montre le lien entre le réseau MADALINE et le processus de Markov à temps continu. Ce lien nous l'avons utilisé pour faire l'analyse de fiabilité d'une usine de production dans le cadre d'un projet proposé par INMA. Le chapitre suivant est consacré au chaînes de Markov cachées (HMM) et aux problèmes liés à leurs utilisation pratique. Le cinquième chapitre commence par un exemple d'application des HMM pour la reconnaissance automatique de la parole (R.A.P), ensuite nous définissons dans les termes de la RAP les modèles de Markov cachées discriminants (HMMD). La dernière partie de ce chapitre présente les différentes possibilités d'utilisation des réseaux de neurones pour l'évaluation des probabilités de transition d'un HMMD. Dans le dernier chapitre, On s'intéresse a la simulation des HMM par les réseaux de neurones. Pour cette raison nous proposons deux approches différentes appliquées à un HMM à deux états et deux symboles d'observations.

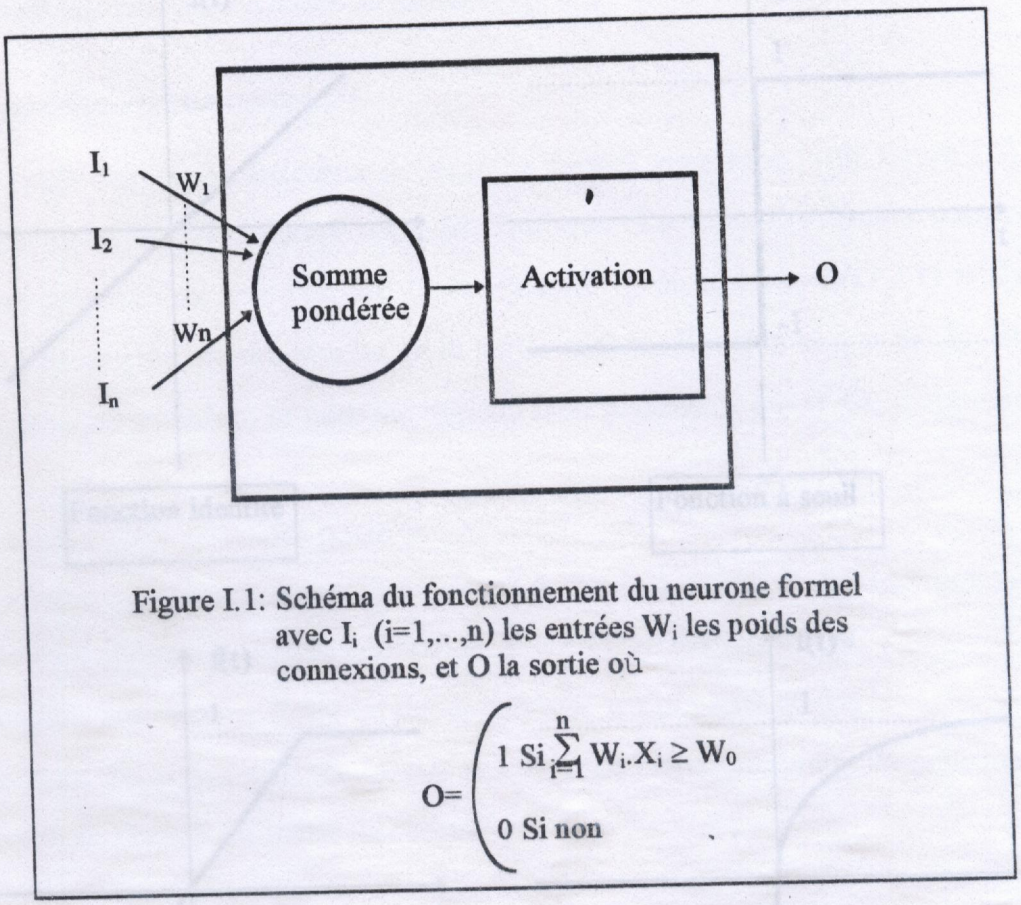


Figure I.1: Schéma du fonctionnement du neurone formel avec I_i (i=1,...,n) les entrées W_i les poids des connexions, et O la sortie où

$$O = \begin{cases} 1 \text{ Si } \sum_{i=1}^{n} W_i.X_i \geq W_0 \\ 0 \text{ Si non} \end{cases}$$

b-. L'automate linéaire généralisé:

Plus généralement, on définit un "automate linéaire généralisé" ou "quasi-linéaire" [23] comme étant l'application d'une fonction f dite de transition ou de transfert au produit scalaire (Métrique euclidienne) entre le vecteur d'entrée $\mathbf{I} = [1, I_1, I_2, ..., I_n]'$ et le" vecteur des poids $\mathbf{W} = [W_0, W_1, W_2, ..., W_n]'$. Les diverses fonctions de transition usuelles sont présentées par la figure ci-dessous.

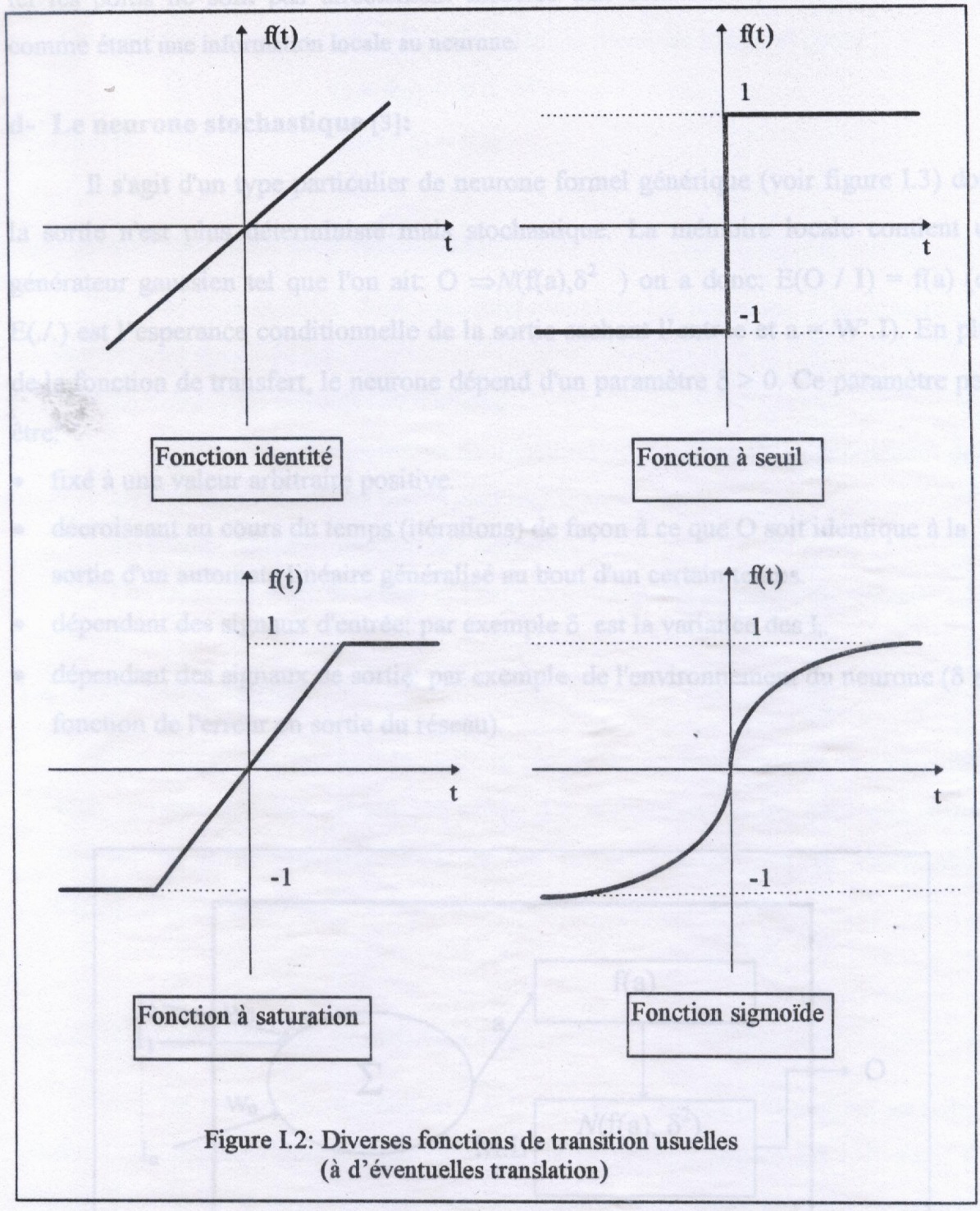


Figure I.2: Diverses fonctions de transition usuelles
(à d'éventuelles translation)

c- Le neurone formel Générique:

Il peut être défini [3] comme étant un élément de calcul dont la sortie O est le résultat d'une fonction de transition f appliquée au vecteur d'entrée **I** et au contenu d'une mémoire locale au neurone ML: O =f(I, ML).

Ici les poids ne sont pas directement associés aux connexions, ils sont considérés comme étant une information locale au neurone.

d- Le neurone stochastique [3]:

Il s'agit d'un type particulier de neurone formel générique (voir figure I.3) dont la sortie n'est plus déterministe mais stochastique. La mémoire locale contient un générateur gaussien tel que l'on ait: $O \Rightarrow N(f(a),\delta^2)$ on a donc: $E(O / I) = f(a)$ (où $E(./.)$ est l'esperance conditionnelle de la sortie sachant l'entrée et $a = W'.I$). En plus de la fonction de transfert, le neurone dépend d'un paramètre $\delta > 0$. Ce paramètre peut être:

- fixé à une valeur arbitraire positive.
- décroissant au cours du temps (itérations) de façon à ce que O soit identique à la sortie d'un automate linéaire généralisé au bout d'un certain temps.
- dépendant des signaux d'entrée; par exemple δ est la variance des I_i.
- dépendant des signaux de sortie: par exemple de l'environnement du neurone (δ est fonction de l'erreur en sortie du réseau).

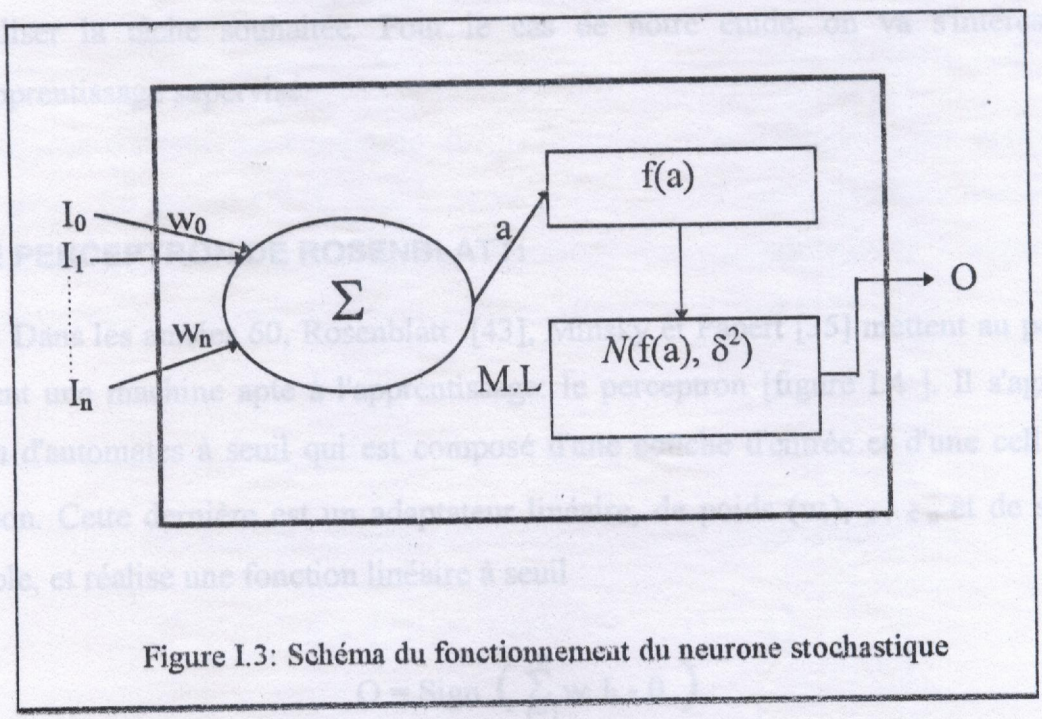


Figure I.3: Schéma du fonctionnement du neurone stochastique

Remarque:

On peut mettre un générateur $N(0, \delta^2)$ dans la mémoire locale et poser :

$$O = f(a) + N(0, \delta^2)$$

On a toujours $E(O / I) = f(a)$.

1.3- ARCHITECTURE DES RESEAUX:

Elle décrit la manière dont sont interconnectés les neurones. L'architecture est entièrement spécifiée par:

- Le nombre de cellules (ou d'unités).
- La nature des unités (fonction de transition ou d'activation) généralement dans un réseau neuronal, les unités sont identiques.
- Le graphe d'interconnexion des cellules.
- Les relations entre le réseau et l'extérieur (les entrées /sorties du réseau).

1.4- L'APPRENTISSAGE:

C'est une phase, pendant laquelle le réseau de neurones adapte sa structure, le plus souvent en déterminant les poids des connexions entre neurones afin de réaliser la tâche souhaitée. Pour le cas de notre étude, on va s'intéresser à l'apprentissage supervisé.

2- LE PERCEPTRON DE ROSENBLATT:

Dans les années 60, Rosenblatt [43], Minsky et Papert [35] mettent au point et étudient une machine apte à l'apprentissage: le perceptron [figure I.4]. Il s'agit d'un réseau d'automates à seuil qui est composé d'une couche d'entrée et d'une cellule de décision. Cette dernière est un adaptateur linéaire, de poids $(w_i)_{1 \le i \le n}$ et de seuil θ variable, et réalise une fonction linéaire à seuil

$$O = \text{Sign}\left(\sum_{i=1}^{n} w_i . I_i - \theta \right)$$

qui permet de séparer les exemples en deux classes (C^+ et C^-), de part et d'autre de l'hyperplan défini par:

$$\sum_{i=1}^{n} w_i.I_i - \theta = 0$$

Dans la pratique, le seuil est généralement identifié à un poids supplémentaire $w_0 = \theta$,appelé biais, et qui est associé à une entrée fixe $I_0 = -1$.

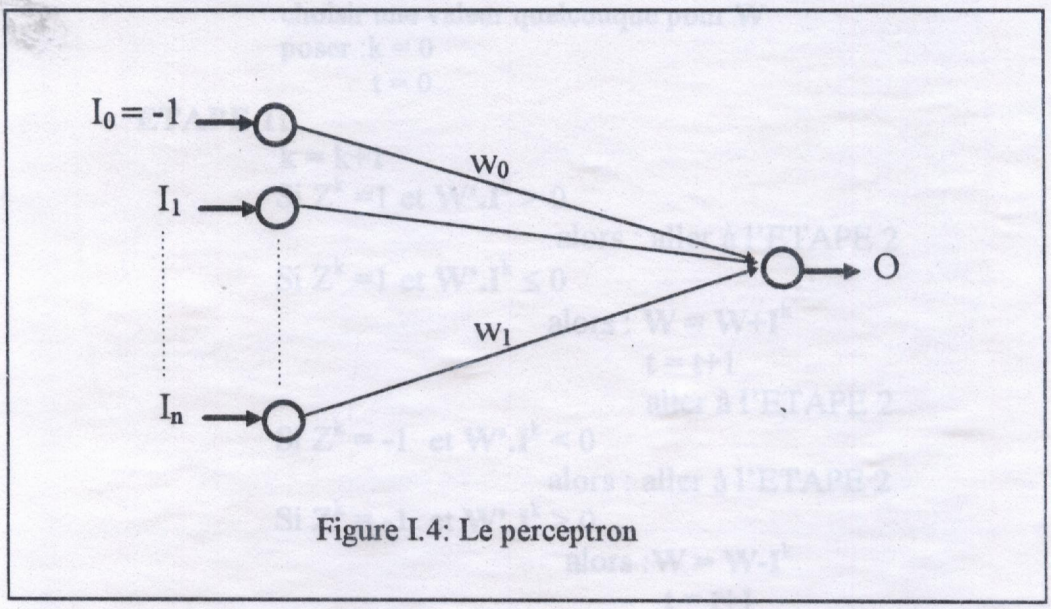


Figure I.4: Le perceptron

La couche d'entrée est composée de (n+1) unités. L'activation de ces dernières est l'identité . En posant $\mathbf{I} = (I_0,...,I_n)'$ et $\mathbf{W} = (w_0,..,w_n)'$, on conviendra que **W'.I** doit être positif pour tout exemple de C^+ et négatif pour tout exemple de C^-. L'apprentissage est supervisé et peut être décrit , en suivant Minsky et Papert [35].

i) Règle d'apprentissage du Perceptron:

Soit E un ensemble d'exemples d'apprentissage de cardinalité L tel que:

$$E = \left\{(I^1,Z^1), (I^2,Z^2),..., (I^L,Z^L)\right\}$$

où $I^k \subset C = C^+ \cup C^-$ tel que $I^k = (1, I^k_1,...,I^k_n)'$ où k =1,...,L

et $Z^k = \begin{cases} 1 \text{ si } I^k \in C^+ \\ -1 \text{ si } I^k \in C^- \end{cases}$ pour k:= 1,2,...,L

Algorithme d'apprentissage

ETAPE 0:

choisir une valeur quelconque pour **W**

poser :k = 0

t = 0

ETAPE 1:

k = k+1

Si $Z^k = 1$ et $W'.I^k > 0$

alors : aller à l'ETAPE 2

Si $Z^k = 1$ et $W'.I^k \leq 0$

alors : $W = W+I^k$

t = t+1

aller à l'ETAPE 2

Si $Z^k = -1$ et $W'.I^k < 0$

alors : aller à l'ETAPE 2

Si $Z^k = -1$ et $W'.I^k \geq 0$

alors : $W = W-I^k$

t = t+1

aller à l'ETAPE 2

ETAPE 2:

Si k< L

alors : aller à l'ETAPE 1

Si k = L et Si condition d'arrêt est vérifiée

alors :STOP

Si non alors : poser k= 0

aller à l'ETAPE 1

La condition d'arrêt est donnée par le théorème de convergence du Perceptron.

Concrètement, les poids ne sont modifiés que lorsqu'un exemple est mal classé.. Cet exemple tire ou repousse les poids selon qu'il s'agit d'un exemple positif ou négatif. Le principe de cet apprentissage est basé sur la correction d'erreur: pour chaque exemple présenté en entrée I^k , on connaît la sortie désirée Z^k , et on compare avec la sortie calculée par le réseau $O^k = W'.I^k$, puis on exploite un signal d'erreur.

On peut étendre la structure du perceptron en disposant en sortie plusieurs cellules de décision [figure I.5]. La fonction linéaire à seuil et la règle d'apprentissage est du type précèdent. Cela permet de traiter des problèmes de classification à un plus grand nombre de classes.

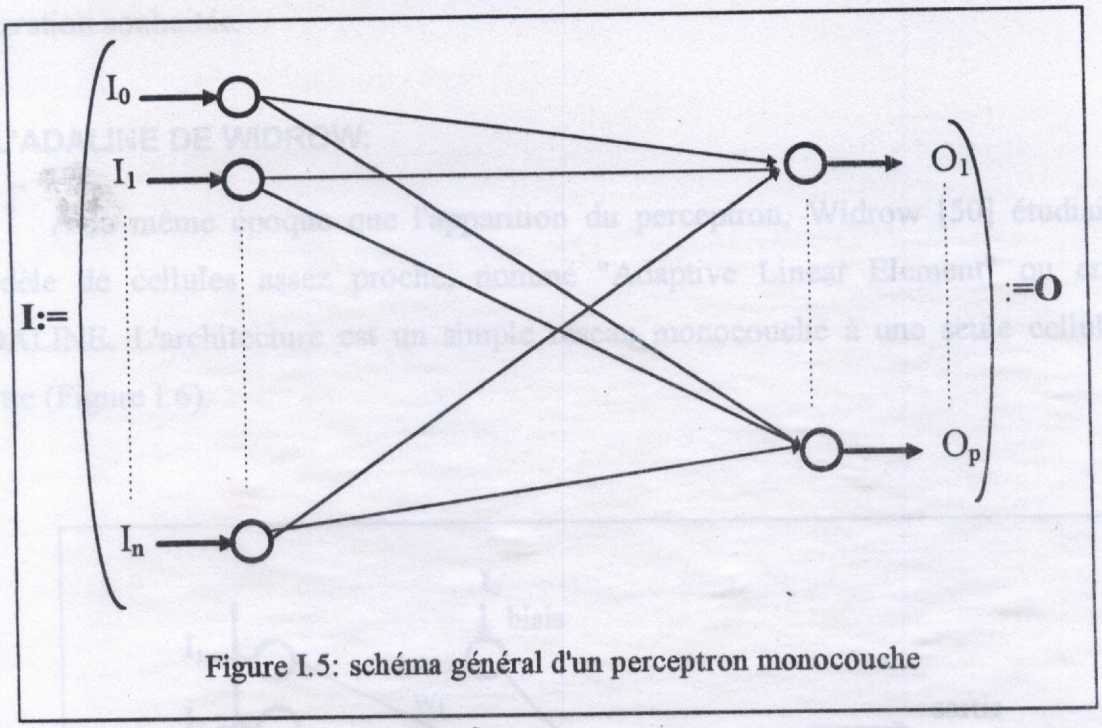


Figure I.5: schéma général d'un perceptron monocouche

La convergence de la règle du perceptron ainsi que ses limites ont été démontrées par Minsky et Papert [35].

ii) Théorème de convergence du perceptron [47]:

Soit E un ensemble quelconque de vecteurs unitaires. S'il existe un vecteur unitaire $\mathbf{W}^*$ et un réel $\zeta > 0$ tels que $\mathbf{W}^{*'}.\mathbf{I}^k > \zeta$ pour tout $\mathbf{I}^k$ dans C^+ , et $\mathbf{W}^{*'}.\mathbf{I}^k < \zeta$ pour tout $\mathbf{I}^k$ dans C^- , alors l'algorithme définissant la règle d'apprentissage du perceptron passera par l'ETAPE 2 seulement un nombre fini de fois.

En d'autres termes ,le théorème indique que, si les deux classes d'exemples sont linéairements séparables, alors le perceptron converge vers une solution correcte, c'est à dire qu'il déterminera au bout d'un nombre fini d'iterations un hyperplan séparateur.

iii) Limites du perceptron:

Lorsque les classes d'exemples ne sont pas linèairement séparables ,l'algorithme d'apprentissage du perceptron ne converge pas en général (l'apparition des mêmes valeurs des poids au cours des itérations) ,et ne garantit même pas que la fonction déterminée après un nombre fini d'étapes soit une bonne approximation de la séparation souhaitée.

3- L'ADALINE DE WIDROW:

A la même époque que l'apparition du perceptron, Widrow [50] étudiait un modèle de cellules assez proche, nommé "Adaptive Linear Element" ou encore ADALINE. L'architecture est un simple réseau monocouche à une seule cellule de sortie (Figure I.6).

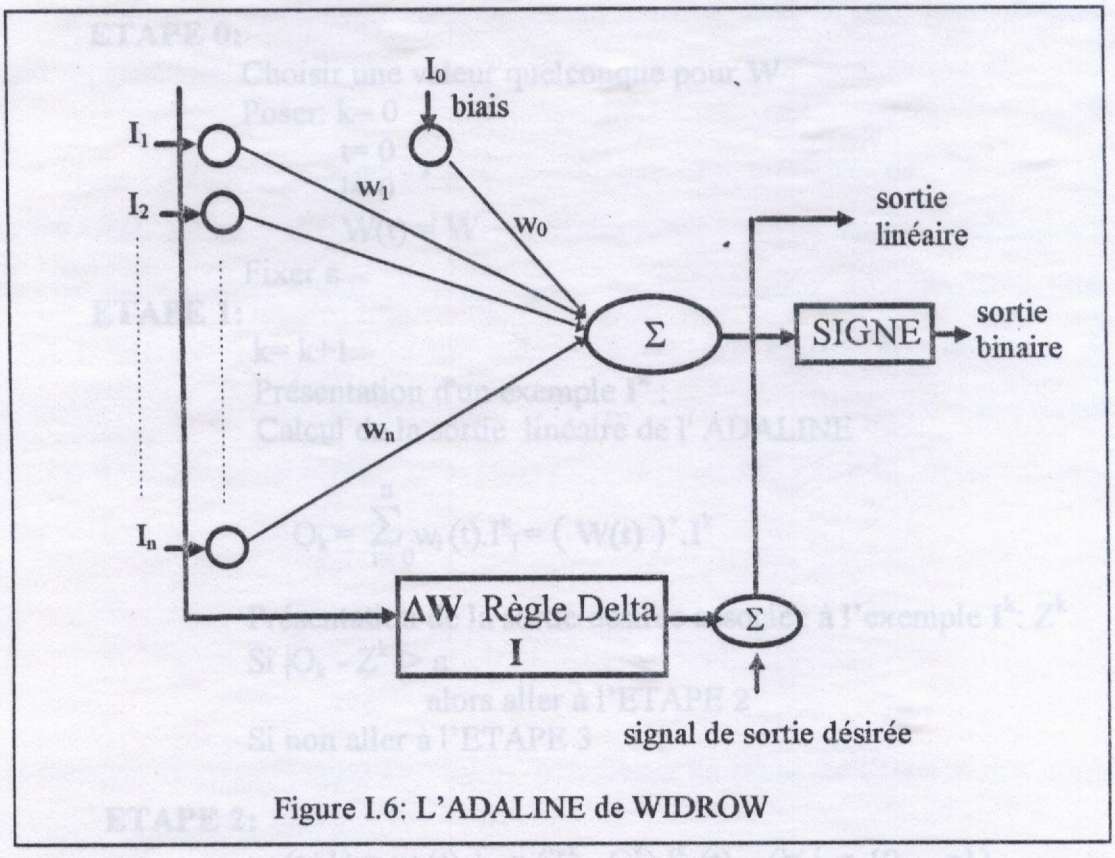


Figure I.6: L'ADALINE de WIDROW

Cette cellule est un adaptateur linéaire comme la cellule de décision du perceptron. Sa sortie peut être réelle ou binaire après seuillage. La différence essentielle avec le perceptron se situe au niveau de sa règle d'apprentissage : l'erreur est évaluée sur la sortie linéaire avant seuillage. Cette règle d'apprentissage à été déduite de la méthode des moindres carrés par Widrow et Hoff , [50].

i) Règle de Widrow-Hoff, ou règle delta:

Soit E un ensemble d'exemples d'apprentissage de cardinalité L tel que:

$$E= \left\{(I^1,Z^1), (I^2,Z^2),..., (I^L,Z^L)\right\}$$

où $I^k \subset C= C^+ \cup C^-$ tel que $I^k= (1, I^k_1,...,I^k_n)'$ et $Z^k \in \Re$

Algorithme d'apprentissage

ETAPE 0:

Choisir une valeur quelconque pour **W**
Poser: k= 0
t= 0
l= 0
W(t) = **W**
Fixer ε

ETAPE 1:

k= k+1
Présentation d'un exemple I^k :
Calcul de la sortie linéaire de l' ADALINE

$$O_k = \sum_{i=0}^{n} w_i(t).I^k_i = (\mathbf{W(t)})'.I^k$$

Présentation de la sortie désirée associée à l'exemple I^k: Z^k
Si $|O_k - Z^k| > \varepsilon$
alors aller à l'ETAPE 2
Si non aller à l'ETAPE 3

ETAPE 2:

$w_i(t+1) = w_i(t) + \alpha.(Z^k - O^k).I^k_i(t) \quad (\forall i \in \{0,...,n\})$
qui peut s'écrire :

$$\Delta W(t) = \alpha.(Z^k - (\mathbf{W(t)})'.I^k).I^k$$

t = t+1

l=l+1

ETAPE 3:

Si k< L

alors :aller à l'ETAPE 1

Si k = L et l ≠ 0

alors : l = 0

k = 0

aller à l'ETAPE 1

Si k = L et l =0

alors STOP

Si l'on note $E(t) = (1/2).(Z_t - O_t)^2$ l'erreur quadratique commise par la cellule sur l'exemple présenté à l'instant t , on observe que:

$$-\partial E(t)/\partial w_i = (Z_t - O_t).I_i(t) \qquad (\forall i \in \{0,...,n\})$$

est proportionnel à $\Delta w_i(t)$. La mise à jour des poids n'a lieu que lorsqu'une erreur est commise, et elle tend à diminuer l'erreur quadratique E(t): C'est une adaptation de la méthode des moindres carrés. Le facteur de proportionnalité α est généralement appelé pas du gradient.

Cet apprentissage est supervisé, et fondé sur une méthode de descente du gradient. En effet, si l'on part de formules semblables, mais si l'on vérifie l'échelle des temps en effectuant la mise à jour des poids après une présentation complète de la base d'exemples E ($|E| = L$), alors la règle de Widrow-Hoff réalise une descente en gradient sur l'erreur globale et minimise cette fonction de coût .Notons E l'erreur globale, c'est à dire la demi somme des carrés des différences entre la sortie calculée et la sortie désirée pour tous les exemples de la base E.

$$E = \sum_{k}^{L} E_k = (1/2).\sum_{k}^{L} (Z_k - O_k)^2$$

Si l'on considère E comme étant l'aptitude associée à un point de l'espace des poids de connexions, et puisque l'on a :

$$\partial E/\partial w_i = \sum_{k}^{L} \partial E_k/\partial w_i \qquad (\forall i \in \{0,...,n\})$$

Alors la règle de Widrow-Hoff réalise une descente suivant la ligne de plus grande pente dans cet espace, et minimise ainsi l'erreur globale E. L'Adaline étant une unité linéaire, E est une fonctionnelle convexe du vecteur de poids W, et donc le minimum est un minimum global. Dans la règle d'apprentissage décrite précédemment, le vecteur des poids est modifié à chaque exemple, et la formule ci dessus n'est plus la même puisque $\partial E_k / \partial w_i$ doit être remplacé par $\partial E(t) / \partial w_i$. C'est le gradient instantané à chaque exemple qui est pris en compte, et cela ne garantit plus la minimisation théorique de l'erreur totale E . Toutefois, en suivant ce chemin la méthode conduit en pratique à un apprentissage satisfaisant car l'introduction du hasard permet de franchir certains minimum locaux de E. L'intérêt de cette approximation tient au caractère local des calculs. Un apprentissage de ce type est nommé méthode du gradient stochastique [10]. Elle est souvent plus efficace qu'une pente du gradient exacte.

Comparaison avec le perceptron :

Dans la mesure où sa règle d'apprentissage est basée sur une réduction de l'erreur quadratique, l'Adaline présente quelques avantages sur le perceptron:

- Lorsque les deux classes d'exemples ne sont pas linéairement séparables, l'Adaline fournit une solution approchée au problème de classification, ce que le perceptron ne sait pas faire.

- Même si les classes sont linèairement séparables, l'hyperplan obtenu par la règle d'apprentissage du perceptron est le premier qui satisfasse la classification , mais pas nécessairement le meilleur, ce qui confère au perceptron de piètres performances en généralisation ; l'Adaline fait mieux: elle fournit un hyperplan séparateur mieux situé par rapport aux classes .

Le réseau MADALINE [3]:

C'est une généralisation de l'Adaline afin de traiter les problèmes de classification à un nombre p quelconque de classes, Il est constitué de p Adalines.

4- LES RESEAUX MULTICOUCHES ET LA RETRO-PROPAGATION:

Depuis la découverte des limites du perceptron , les chercheurs connexionistes mettaient leurs espoirs dans les réseaux multicouches (Figure I.9) , mais ils ne savaient par comment définir une règle d'apprentissage pour les couches cachées. La situation s'est débloquée vers 1985, lorsque plusieurs chercheurs, indépendamment les uns des autres, découvrent comment appliquer des règles de dérivation en chaîne pour contourner la difficulté. Divers travaux ([42],[29]) ont conduit à la définition de l'actuelle règle d'apprentissage par rétro-propagation, souvent nommée Back-propagation.

Un réseau multicouches est une extension du modèle monocouche issu du perceptron, avec une ou plusieurs couches cachées [47] successives entre la couche d'entrée et la couche de sortie. La connectivité la plus courante consiste à définir des connexions uniquement d'une couche vers la couche suivante: on parle alors d'un réseau **Feed-Forward**. On peut aussi autoriser des connexions pontées [3], [30], par exemple de la couche d'entrée vers la seconde couche cachée.

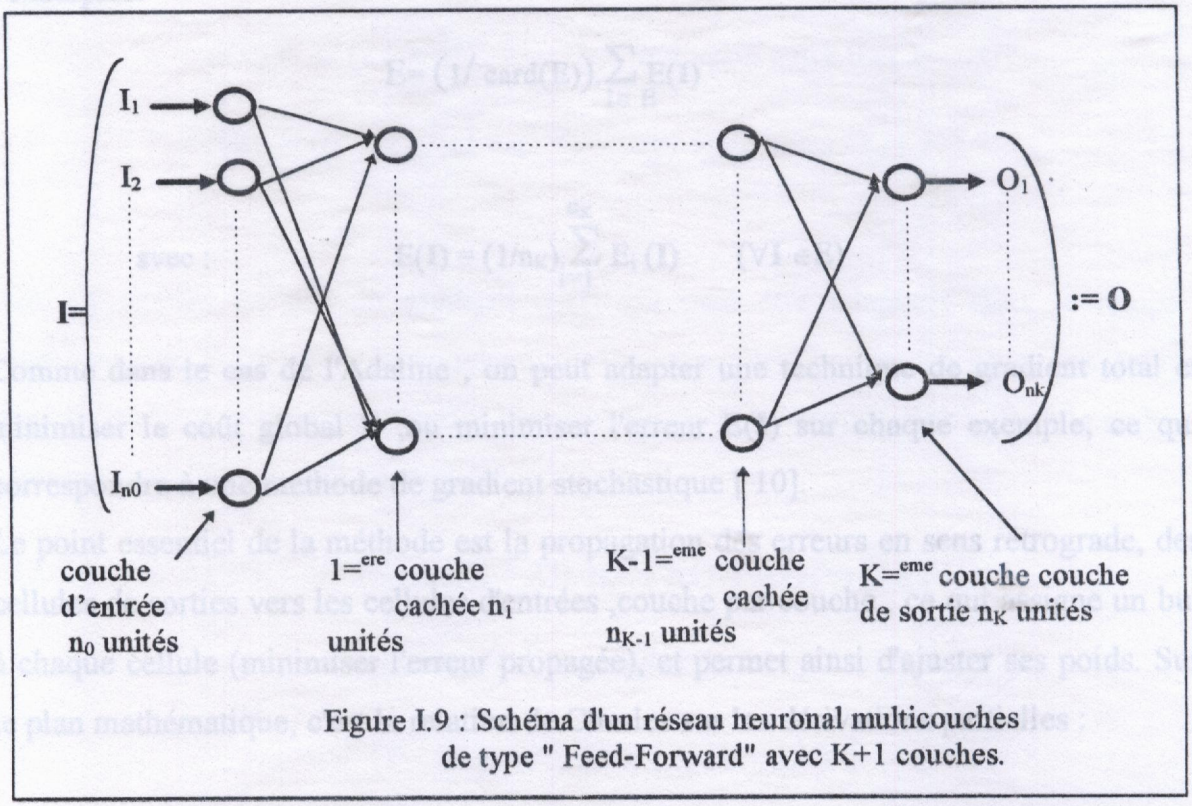


Figure I.9 : Schéma d'un réseau neuronal multicouches de type " Feed-Forward" avec K+1 couches.

Nous exposons ici les règles de propagation, d'activation , et d'apprentissage concernant les réseaux Feed-Forward. La couche d'entrée est parfois omise dans le décompte des couches car ces unités ont pour rôle de transmettre simplement les valeurs d'entrées vers le réseau. Les unités de toutes les autres couches sont des automates quasi-linéaires (§ I.2.b) qui transmettent les activations de couche en couche jusqu'à la couche de sortie . On désigne par W_{ijk} ($1 \leq j \leq n_{k-1}$ $1 \leq i \leq n_k$, $1 \leq k \leq K$) le poids qui relie l'unité j de la couche k-1 à l'unité i de la couche k. Ce type d'apprentissage est supervisé, et est basé sur la méthode du gradient. Plus précisément , il applique la technique de Widrow-Hoff [50] pour minimiser une fonction de coût. Comme pour l'Adaline , on peut définir pour chaque cellule de sortie une erreur quadratique locale: $E_i(\mathbf{I}) = (Z_i(\mathbf{I}) - O_{iK}(\mathbf{I}))^2$ $(\forall\, i \in \{1,...,n_k\})$ où $Z_i(\mathbf{I})$ est la sortie désirée de la cellule i, pour l'exemple **I** et $O_{iK}(\mathbf{I})$ la sortie de l'unité i de la couche K (couche de sortie) qui a été calculée par le réseau multicouches sur la présentation de l'exemple **I**.

On en déduit un critère de coût global en calculant l'erreur totale sur la base d'exemples:

$$E = (1/\text{card}(E)).\sum_{I \in E} E(\mathbf{I})$$

avec :
$$E(\mathbf{I}) = (1/n_K).\sum_{i=1}^{n_K} E_i(\mathbf{I}) \qquad (\forall \mathbf{I} \in E)$$

Comme dans le cas de l'Adaline , on peut adapter une technique de gradient total et minimiser le coût global E ,ou minimiser l'erreur E(**I**) sur chaque exemple, ce qui correspondra à une méthode de gradient stochastique [10].

Le point essentiel de la méthode est la propagation des erreurs en sens rétrograde, des cellules de sorties vers les cellules d'entrées ,couche par couche , ce qui assigne un but à chaque cellule (minimiser l'erreur propagée), et permet ainsi d'ajuster ses poids. Sur le plan mathématique, c'est la relation de Chasles sur les dérivations partielles :

$$\partial f/\partial X = (\partial f/\partial U).(\partial U/\partial X)$$

Nous exposons ici les règles de propagation, d'activation , et d'apprentissage concernant les réseaux Feed-Forward. La couche d'entrée est parfois omise dans le décompte des couches car ces unités ont pour rôle de transmettre simplement les valeurs d'entrées vers le réseau. Les unités de toutes les autres couches sont des automates quasi-linéaires (§ I.2.b) qui transmettent les activations de couche en couche jusqu'à la couche de sortie . On désigne par W_{ijk} ($1 \leq j \leq n_{k-1}$ $1 \leq i \leq n_k$, $1 \leq k \leq K$) le poids qui relie l'unité j de la couche k-1 à l'unité i de la couche k. Ce type d'apprentissage est supervisé, et est basé sur la méthode du gradient. Plus précisément , il applique la technique de Widrow-Hoff [50] pour minimiser une fonction de coût. Comme pour l'Adaline , on peut définir pour chaque cellule de sortie une erreur quadratique locale: $E_i(\mathbf{I}) = (Z_i(\mathbf{I}) - O_{iK}(\mathbf{I}))^2$ $(\forall\, i \in \{1,...,n_k\})$ où $Z_i(\mathbf{I})$ est la sortie désirée de la cellule i, pour l'exemple **I** et $O_{iK}(\mathbf{I})$ la sortie de l'unité i de la couche K (couche de sortie) qui a été calculée par le réseau multicouches sur la présentation de l'exemple **I**.

On en déduit un critère de coût global en calculant l'erreur totale sur la base d'exemples:

$$E = (1/\text{card}(E)).\sum_{I \in E} E(\mathbf{I})$$

avec :
$$E(\mathbf{I}) = (1/n_K).\sum_{i=1}^{n_K} E_i(\mathbf{I}) \qquad (\forall \mathbf{I} \in E)$$

Comme dans le cas de l'Adaline , on peut adapter une technique de gradient total et minimiser le coût global E ,ou minimiser l'erreur E(**I**) sur chaque exemple, ce qui correspondra à une méthode de gradient stochastique [10].

Le point essentiel de la méthode est la propagation des erreurs en sens rétrograde, des cellules de sorties vers les cellules d'entrées ,couche par couche , ce qui assigne un but à chaque cellule (minimiser l'erreur propagée), et permet ainsi d'ajuster ses poids. Sur le plan mathématique, c'est la relation de Chasles sur les dérivations partielles :

$$\partial f/\partial X = (\partial f/\partial U).(\partial U/\partial X)$$

encore appelée **"chain rule"** , qui permet d'établir une fonction de récurrence entre les gradients des unités d'une couche et ceux des unités de la couche de rang supérieur , i.e. celles qui précèdent ... puisqu'on progresse dans le sens rétrograde.

Plus précisément :

$$\partial E(\mathbf{I})/\partial w_{ijk} = (\partial E(\mathbf{I})/\partial \mathbf{a}_{ik}).(\partial \mathbf{a}_{ik}/\partial w_{ijk})$$

$$= (\partial E(\mathbf{I})/\partial \mathbf{a}_{ik}).O_{j,k-1}$$

On introduit une nouvelle variable $D_{ik}(\mathbf{I}) = -(\partial E(\mathbf{I})/\partial \mathbf{a}_{ik})$, et l'on peut démontrer qu'il suffit en fait de calculer et de propager ces gradients-là de couche en couche.

On note $\mathbf{a}_{ik}$: l'activation de l'unité i de la couche k.

O_{ik} :la sortie de l'unité i de la couche k.

L'algorithme peut être défini uniquement si l'on considère des unités semi-lineaires, c'est à dire des unités dont la fonction de transition est différentiable. Les fonctions les plus couramment utilisées sont:

$f(z) = 1/(1+e^{-\lambda z})$ qui varie entre 0 et 1

$f(z) = (e^{\lambda z}-1)/(1+e^{-\lambda z})$ qui varie entre -1 et 1

Enfin, on définit un pas du gradient $\alpha_k(t)$ qui peut être adaptatif dans le temps (dépendance de t) ou selon la couche (indexation par k). Toutes les notations étant définies , on peut maintenant décrire l'algorithme de retro-propagation , dans le cadre du gradient stochastique, de la manière suivante:

i) REGLE D'APPRENTISSAGE PAR RETRO-PROPAGATION:

Soit E l'ensemble des exemples d'apprentissage tel que:

$$E = \{ (\mathbf{I}^1, \mathbf{Z}^1), (\mathbf{I}^2, \mathbf{Z}^2), \ldots, (\mathbf{I}^L, \mathbf{Z}^L) \}$$

où $\mathbf{I}^k = (I^k_1, I^k_2, \ldots, I^k_{n0})'$ et $\mathbf{Z}^k = (Z^k_1, Z^k_2, \ldots, Z^k_{nK})'$

Algorithme de la Retro-propagation

INITIALISATION:

En t=0, initialiser les poids $w_{ijk}(0)$ à de faibles valeurs aléatoires.

REPETER:

Pour chaque exemple $\mathbf{I}^l$ de la base E présenté aléatoirement faire:

- présenter l'exemple $\mathbf{I}^l(t)= (\mathbf{I}^l_{i0}(t))$ aux unités d'entrée;
- calculer les états et les sorties du réseau en appliquant, couche par couche , en sens direct, la règle de propagation des activations:

$$(\forall\, i \in \{1,...,n_0\}) \quad O_{i0}(t) = \mathbf{a}_{i0}(t) = I^l_{i0}(t)$$

$$(\forall\, k \in \{1,...,K\}), (\forall\, i \in \{1,...,n_k\})\ O_{ik}(t) = f(\mathbf{a}_{ik}(t))$$

avec: $$\mathbf{a}_{ik}(t) = \sum_{j=0}^{n_{k-1}} w_{ijk}(t).O_{jk-1}(t)$$

et f une fonction sigmoïde

-présenter les sorties désirées aux unités de sorties, et calculer les gradients associes à ces unités:

$$(\forall\, i \in \{1,...,n_k\}) \quad D_{i,k}(t) = (\partial E(t)/\, \partial \mathbf{a}_{i,k}(t)) . =$$

$$= 2.f'(\mathbf{a}_{i,k}(t)).(Z_i(t) - O_{i,k}(t))$$

-calculer les gradients aux autres unités en appliquant, couche par couche, en sens rétrograde la formule de récurrence:

$$(\forall\, k \in \{1,...,K-1\}), (\forall\, i \in \{1,...,n_k\})$$

$$D_{i,k}(t) = f'(a_{i,k}(t)).\sum_{m=0}^{n_{k+1}} w_{m,i,k+1}(t).D_{m,k+1}(t)$$

-mettre à jour les poids des connexions:

$$(\forall\, k \in \{1,...,K\}), (\forall\, i \in \{1,...,n_k\}), (\forall\, j \in \{1,...,n_{k-1}\})$$

$$w_{i,j,k}(t+1) = w_{i,j,k}(t) + \alpha_k(t).D_{i,k}(t).O_{j,k-1}(t)$$

-t = t +1

tant-que base E non épuisée

jusqu'à critère d'arrêt.

La convergence de l'algorithme stochastique n'étant pas démontrée dans le cas général. On utilise en pratique divers critères d'arrêt:

- un seuil sur la fonction de coût, au dessous duquel on stoppe l'apprentissage
- un seuil sur le taux de succès sur la base d'exemple , au dessus duquel on arrête
- un nombre de passes prédéterminé, i.e. un nombre d'itérations fixées

ii) PHASE DE GENERALISATION:

La phase de reconnaissance , ou de généralisation consiste à présenter de nouveaux motifs (exemples) et à évaluer les sorties calculées par le réseau après avoir figé les poids dans l'état obtenu après apprentissage. Il est usuel de tester les capacités de généralisation d'un réseau en évaluant le taux de succès sur une base de test, entièrement disjointe de la base d'apprentissage, mais pour laquelle les sorties désirées sont connues.

iii) EQUIVALENCE ENTRE LE RESEAU MULTICOUCHES ET LE CLASSIFIEUR BAYESIEN A PROBABILITE D' ERREUR MINIMUM:

Dans ce paragraphe nous allons voir que les sorties d'un réseau multicouches peuvent approximer les probabilités à posteriori du classifieur bayesien [3] . Pour commencer considérons un problème à deux classes C^+ et C^- . La règle de décision bayesienne $\mathbf{D}(\mathbf{x})$ (où $\mathbf{x}$ est un vecteur appartenant à $C = C^+ \cup C^-$) pour une probabilité d'erreur minimale est donnée par:

$$\mathbf{D}(\mathbf{x}) = \begin{cases} C^+ \text{ si } g_0(\mathbf{x}) > 0 \\ C^- \text{ si } g_0(\mathbf{x}) < 0 \end{cases}$$

avec $g_0(\mathbf{x}) = P(C^+ / \mathbf{x}) - P(C^- / \mathbf{x})$ et $P(C^+ / \mathbf{x})$: désigne la probabilité conditionnelle d'avoir la classe C^+ sachant qu'on a le vecteur $\mathbf{x}$. La densité de probabilités du mélange ($\mathbf{x} \in C$) est:

$$f(x) = P(C^+).f(x / C^+) + P(C^-).f(x / C^-) \qquad (I.1)$$

Supposons maintenant un réseau multicouches avec une seule cellule de sortie. On note par $y(x)$ sa sortie pour une forme x sur sa couche d'entrée . Alors $y(x) := F(x, W)$, où F désigne la fonction réalisée par le réseau et W le vecteur des poids des connexions entre les neurones des différentes couches

Pour l'apprentissage du modèle connexioniste, on suppose que les sorties désirées sont données par:

$$Z(x) = \begin{cases} 1 \text{ si } x \in C^+ \\ -1 \text{ si } x \in C^- \end{cases}$$

Calculons l'erreur quadratique moyenne ε^2 sur la sortie du modèle:

$$\varepsilon^2 = \int_C [y(x) - Z(x)]^2.f(x).dx$$

$$= \int_C [F(x, W) - Z(x)]^2.f(x).dx$$

Puisque:

$$[F(x, W) - Z(x)]^2 = [(F(x, W) - g_0(x)) + (g_0(x) - Z(x))]^2$$

On a:

$$\varepsilon^2 = \int_C [F(x, W) - g_0(x)]^2.f(x).dx + 2.\int_C F(x, W).g_0(x).f(x).dx$$

$$-2.\int_C F(x, W).Z(x).f(x).dx - 2.\int_C g_0(x)^2.f(x).dx$$

$$+ 2.\int_C g_0(x).Z(x).f(x).dx + \int_C [g_0(x) - Z(x)]^2.f(x).dx$$

On remarque que les trois derniers facteurs du terme à droite de l'égalité ci-dessus sont indépendants de **W**. Du moment que l'algorithme de la rétro-propagation du gradient consiste à rechercher **W** tel que ε^2 soit minimale, il est donc équivalent de trouver **W** tel que η^2 soit minimale, où:

$$\eta^2 = \int_C [F(\mathbf{x}, \mathbf{W}) - g_0(\mathbf{x})]^2 . f(\mathbf{x}).d\mathbf{x} + 2.\int_C F(\mathbf{x}, \mathbf{W}).g_0(\mathbf{x}).f(\mathbf{x}).d\mathbf{x}$$

$$-2.\int_C F(\mathbf{x}, \mathbf{W}).Z(\mathbf{x}).f(\mathbf{x}).d\mathbf{x} \qquad (I.2)$$

Considérons les deux derniers facteurs du terme à droite de l'égalité (I.2), c'est à dire:

$$\Im = \int_C F(\mathbf{x}, \mathbf{W}).[g_0(\mathbf{x}) - Z(\mathbf{x})].f(\mathbf{x}).d\mathbf{x}$$

Puisque:

$$g_0(\mathbf{x}) = P(C^+ / \mathbf{x}) - P(C^- / \mathbf{x})$$

avec

$$P(C^+ / \mathbf{x}) + P(C^- / \mathbf{x}) = 1 \quad (I.3)$$

Il vient donc:

$$g_0(\mathbf{x}) = 1 - 2.P(C^- / \mathbf{x})$$

D'autre part:

$$\text{Si } \mathbf{x} \in C^+ \Rightarrow Z(\mathbf{x}) = 1 \Rightarrow g_0(\mathbf{x}) = Z(\mathbf{x}) - 2.P(C^+ / \mathbf{x})$$
$$\text{Si } \mathbf{x} \in C^- \Rightarrow Z(\mathbf{x}) = -1 \Rightarrow g_0(\mathbf{x}) = -Z(\mathbf{x}) - 2.P(C^- / \mathbf{x})$$

Donc:

$$g_0(\mathbf{x}) - Z(\mathbf{x}) = \begin{cases} -2.P(C^- / \mathbf{x}) & \text{si } \mathbf{x} \in C^+ \\ 2 - 2.P(C^- / \mathbf{x}) & \text{si } \mathbf{x} \in C^- \end{cases}$$

ou encore en utilisant:

$$P(C^- / \mathbf{x}) = 1 - P(C^+ / \mathbf{x})$$

on obtient

$$g_0(\mathbf{x}) - Z(\mathbf{x}) = \begin{cases} -2.P(C^- / \mathbf{x}) & \text{si } \mathbf{x} \in C^+ \\ 2.P(C^+ / \mathbf{x}) & \text{si } \mathbf{x} \in C^- \end{cases} \qquad (I.4)$$

En remplaçant f(**x**) par (I.1) et (g_0(**x**) - Z(**x**)) par (I.4) , on a:

$$\Im = -2.P(C^+).\int_C F(\mathbf{x}, \mathbf{W}).P(C^- / \mathbf{x}).f(\mathbf{x} / C^+).d\mathbf{x} +$$

$$+2.P(C^-).\int_C F(\mathbf{x}, \mathbf{W}).P(C^+ / \mathbf{x}).f(\mathbf{x} / C^-).d\mathbf{x}$$

Par ailleurs, on a:

$$P(C^+ / \mathbf{x}) = f(\mathbf{x} / C^+).P(C^+) / f(\mathbf{x})$$
$$P(C^- / \mathbf{x}) = f(\mathbf{x} / C^-).P(C^-) / f(\mathbf{x})$$

d'où:

$$\Im = -2.P(C^+).P(C^-)\int_C (F(\mathbf{x}, \mathbf{W}).f(\mathbf{x} / C^-).f(\mathbf{x} / C^+) / f(\mathbf{x})).d\mathbf{x} +$$

$$+2.P(C^-).P(C^+).\int_C (F(\mathbf{x}, \mathbf{W}).f(\mathbf{x} / C^-).f(\mathbf{x} / C^+) / f(\mathbf{x})).d\mathbf{x}$$

On trouve $\Im = 0$.

L'apprentissage du réseau à couche par la rétro-propagation se ramène donc a rechercher **W** tel que:

$$\eta^2 = \int_C [F(\mathbf{x}, \mathbf{W}) - g_0(\mathbf{x})]^2.f(\mathbf{x}).d\mathbf{x}$$

soit minimale. La solution est donc $\mathbf{W}^*$ telle que :

$$F(\mathbf{x}, \mathbf{W}^*) = g_0(\mathbf{x})$$

Ainsi on peut conclure que le réseau multicouches évalue les probabilités à posteriori des classes (sous réserve que l'algorithme de la rétro-propagation du gradient aboutisse bien au minimum global de F(. , .))

LES MODELES CONNEXIONNISTES POUR LE TRAITEMENT DES TACHES SEQUENTIELLES

1- INTRODUCTION :

Les réseaux de neurones artificiels ont prouvé qu'ils sont une alternative prometteuse pour le traitement de certaines tâches de nature séquentielle (exemples : [15]; [16]; [27]; [45]; [48]). Les tâches de nature séquentielle sont délicates car les architectures et les algorithmes d'apprentissage des réseaux de neurones ne sont généralement pas bien adaptés aux modèles variant dans le temps.

L'utilisation des réseaux de neurones s'est plutôt développée dans le traitement des tâches combinatoires. Dans la plupart des travaux, une collection d'attributs caractéristiques est présentée au réseau, et celui-ci doit classer ces entrées dans une ou plusieurs classes. Dans ce cas, on dit que le réseau a un comportement combinatoire, c'est à dire, que la sortie actuelle **O**(t) du réseau dépend dans le temps seulement de l'entrée actuelle **I**(t), ce qui est équivalent à :

$$\mathbf{O}(t) = F_W(\mathbf{I}(t)) \qquad \text{(II.1)}$$

où F_W est la fonction réalisée par le réseau neuronal. Par contraste, les modèles de traitement des tâches de nature séquentielle évoluent dans le temps. La réponse appropriée à une entrée donnée dépend dans le temps non seulement de l'entrée actuelle, mais potentiellement de toutes les entrées antérieures. Dans ce cas, on dit que le réseau a un comportement séquentiel , c'est à dire, qu'il obéit à l'équation suivante :

$$\mathbf{O}(t) = F_W(\mathbf{I}(t), \mathbf{I}(t-1), ..., \mathbf{I}(t-n)) \qquad \text{(II.2)}$$

avec t et n des entiers positifs tels que $0 < n \leq t$. Une tâche de nature séquentielle englobe deux composantes (figure II.1), la première est une mémoire à court-terme qui

retient les aspects de la séquence d'entrée qui sont utiles pour calculer la sortie, la seconde évalue la sortie en se basant sur le contenu de la mémoire à court - terme.

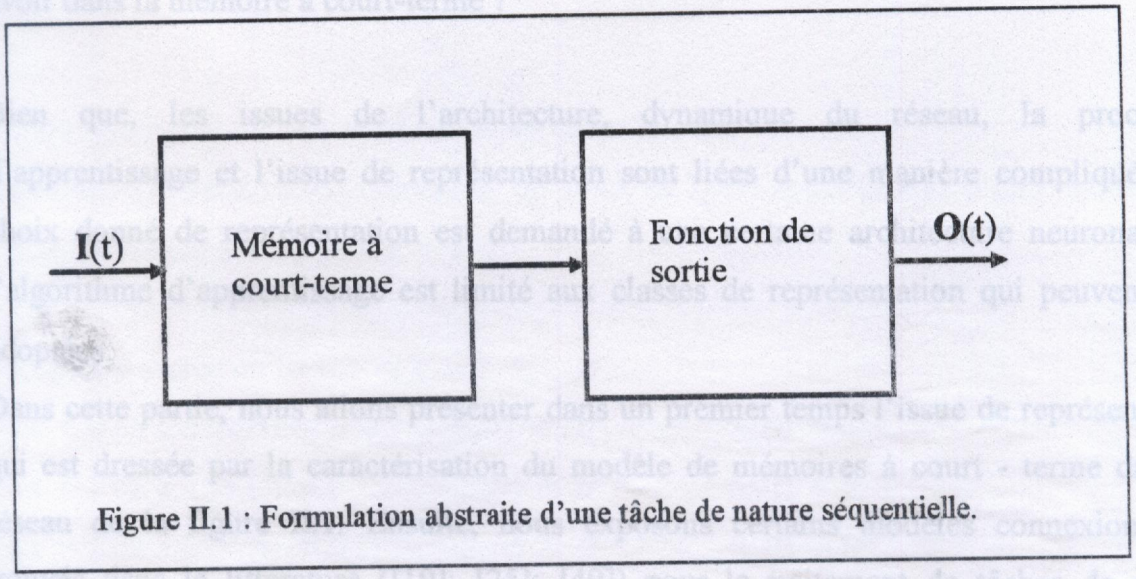


Figure II.1 : Formulation abstraite d'une tâche de nature séquentielle.

Dans l'architecture des réseaux de neurones, la fonction de sortie est en génerale un réseau de neurones à couches (voir chapitre I), tandis que la mémoire à court-terme a souvent des connexions intérieures récurrentes. Dans la construction d'un modèle connexionniste pour le traitement des tâches de nature séquentielle on considère les trois points suivants :

- **L'architecture :** C'est la structure interne du réseau réalisant la mémoire à court - terme et le réseau réalisant la fonction de sortie.

- **L'apprentissage :** Etant donné un ensemble d'exemples d'apprentissage, combien de paramètres internes dans le modèle (les poids des connexions) sont adaptés.

Chaque exemple d'apprentissage k consiste en une séquence d'entrées $(\mathbf{I}^k(1), \mathbf{I}^k(2), ..., \mathbf{I}^k(t_k))$ et une séquence de sortie $(\mathbf{Z}^k(1), \mathbf{Z}^k(2), ..., \mathbf{Z}^k(t_k))$, où t_k : est la longueur de la séquence k et $\mathbf{Z}^k(t)$: est la sortie désirée de l'exemple k au temps t. L'apprentissage du réseau revient à trouver les poids des connexions tels que les sorties réelles $\mathbf{O}^k(t)$ soient les plus proches possibles des sorties désirées $\mathbf{Z}^k(t)$, généralement au sens des moindres carrés.

- **La représentation :** Comment faut-il prendre en compte le temps des séquences d'entrées dans la mémoire à court-terme, la nature et la qualité d'information qu'il faut avoir dans la mémoire à court-terme ?

Bien que, les issues de l'architecture, dynamique du réseau, la procédure d'apprentissage et l'issue de représentation sont liées d'une manière compliquée, un choix donné de représentation est demandé à une certaine architecture neuronale où l'algorithme d'apprentissage est limité aux classes de représentation qui peuvent être adoptées.

Dans cette partie, nous allons présenter dans un premier temps l'issue de représentation qui est dressée par la caractérisation du modèle de mémoires à court - terme dans le réseau de la figure II.1. Ensuite, nous exposons certains modèles connexionnistes trouvés dans la littérature ([19]; [25]; [49]) pour le traitement de tâches de nature séquentielle.

2- MEMOIRE A COURT - TERME :

Le modèle de mémoire à court-terme varie selon les trois dimensions [37] suivantes :

2.1- LA FORME:

Il existe plusieurs formes. Dans cette partie, on étudiera les trois formes suivantes

i) Mémoire à RETARD :

Cette mémoire [17],[37] sélectionne L éléments de la séquence d'entrée $I(t - \omega_1)$, $I(t- \omega_2)$, ..., $I(t - \omega_L)$, et forme un état de représentation $S(t) := (S_1(t), S_2(t), ..., S_L(t))$

avec $$S_k(t) = \sum_{j=1}^{\infty} C_k(t - j).I(j) \qquad (II.3)$$

$$S_k(t) = (1 - \mu_k).\mathbf{I}(t) + \mu_k.S_k(t-1) \qquad \text{(II.6)}$$

avec $S_k(0) = 0 \qquad \forall\ k:=1,2,...,L$

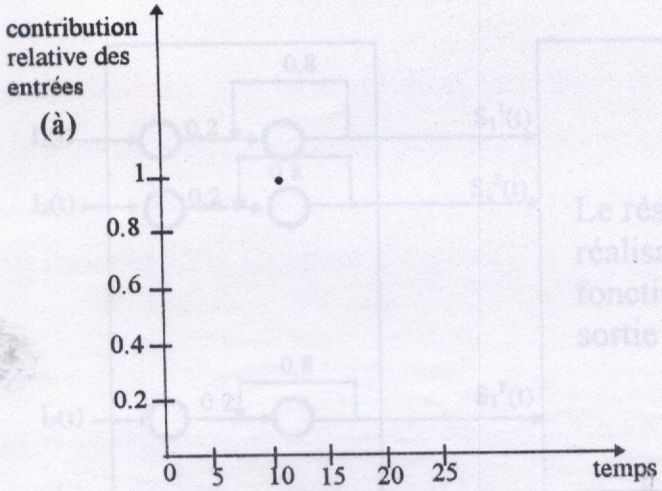
Pour le cas de notre exemple, on a :

$$S_1(t) = (0.2).\mathbf{I}(t) + (0.8).S_1(t - 1)$$

avec $S_1(0) = 0;$

Cette forme de mémoire peut être implantée dans un réseau neuronal (voir figure II.4). Le comportement du réseau dans ce cas de mémoire est donné par l'équation suivante :

$$O(t) = F_W(\mathbf{I}(1), \mathbf{I}(2), ..., \mathbf{I}(t)) \qquad \text{(II.7)}$$



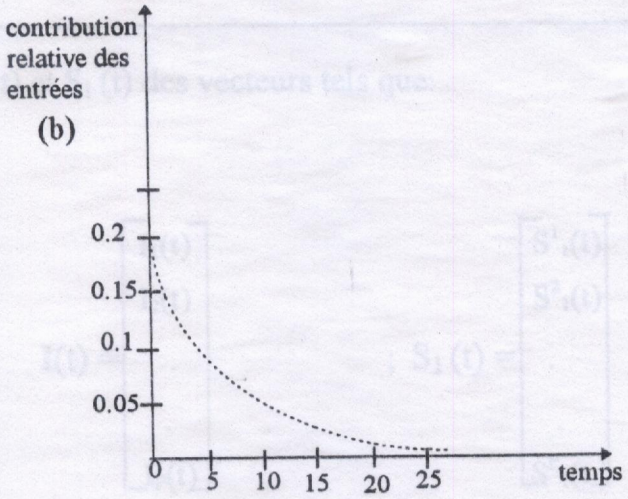


Figure II.3 :

(a) La fonction noyau de la mémoire explorant les lignes à retard avec $\omega := 10$

(b) La fonction noyau de la mémoire à allure exponentielle avec $\mu_1 := 0.8$

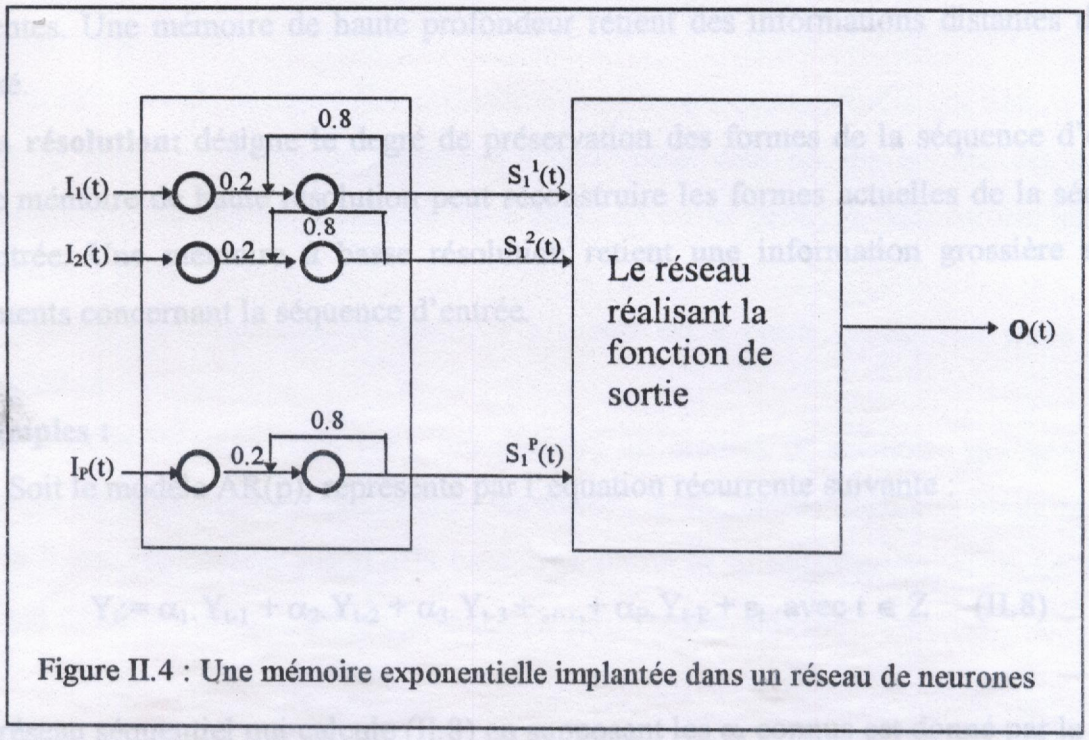


Figure II.4 : Une mémoire exponentielle implantée dans un réseau de neurones

avec $\mathbf{I}(t)$ et $S_1(t)$ des vecteurs tels que:

$$\mathbf{I}(t) = \begin{bmatrix} I_1(t) \\ I_2(t) \\ . \\ . \\ I_P(t) \end{bmatrix} \quad ; \; S_1(t) = \begin{bmatrix} S^1{}_1(t) \\ S^2{}_1(t) \\ . \\ . \\ S^P{}_1(t) \end{bmatrix}$$

Sur la figure II.4, le premier bloc représente le réseau réalisant la mémoire exponentielle avec $\mu_1 := 0.8$, et le deuxième bloc représente le réseau de sortie.

iii) Mémoire GAMMA :

Deux dimensions ([18]; [37]), profondeur et résolution, peuvent être utilisées pour caractériser les deux formes de mémoires précédentes, grossièrement :

reconstruire les formes d'entrée actuelles de la séquence mais pas les formes précédentes.

2) - L'exemple de la figure II.4 présente une mémoire exponentielle qui tient compte de tout le passé en terme de formes d'entrée; pour cela on dit qu'elle est de haute profondeur. Elle est de basse résolution parce qu'on ne peut pas reconstruire les formes d'entrée à partir de S(t).

En termes de dimension, la mémoire à retard est à basse profondeur mais à haute résolution.

De Vries et Principe dans [18], ont proposé un modèle de mémoire qui généralise à travers les deux modèles précédents, les formes de mémoires depuis la haute résolution et la basse profondeur jusqu'à la basse résolution et la haute profondeur. La fonction noyau de cette mémoire (voir figure II.4) s'écrit comme suit :

$$C_k(t) = \begin{cases} C_{\omega k}(t).(1-\mu_k)^{(\omega k+1)}.\mu_k^{(t-\omega k)} & \text{si } t > \omega_k \\ 0 \quad \text{si non} \end{cases} \qquad \text{(II.9)}$$

$\omega_k \in Z$ représente le retard, et $\mu_k \in [0,1]$.

Comme pour la mémoire exponentielle, la convolution de la fonction noyau gamma avec la séquence d'entrée peut être calculée iterativement pour les valeurs μ et ω données. Pour calculer $S_{\mu,\omega}$, il est nécessaire de calculer $S_{\mu,j}$ pour $j := 0, 1, \ldots, \omega - 1$. L'équation récursive est :

$$S_{\mu,j}(t) = (1-\mu).S_{\mu,j-1}(t) + \mu.S_{\mu,j}(t-1) \qquad \text{(II.10)}$$

sous les conditions aux limites

$$S_{\mu,-1}(t) = I(t+1) \quad \text{pour } t \geq 0$$

$$S_{\mu,j}(0) = 0 \quad \text{pour } j \geq 0$$

Dans la construction de la mémoire gamma, le constructeur doit spécifier la grandeur de ω notée par L. Le nombre total de vecteurs d'état est donc L + 1, $S_{\mu,0}$, $S_{\mu,1}$, ..., $S_{\mu,L}$.
Pour un L donné, l'échange entre la résolution et la profondeur est accompli par la variation de μ. Une grande valeur de μ donne une haute profondeur et une basse résolution de la mémoire tandis qu'une petite valeur de μ donne une basse profondeur et une haute résolution de la mémoire.
Ces trois formes de mémoires ne sont pas les seules. En effet, chaque fonction noyau donne une forme de mémoire.

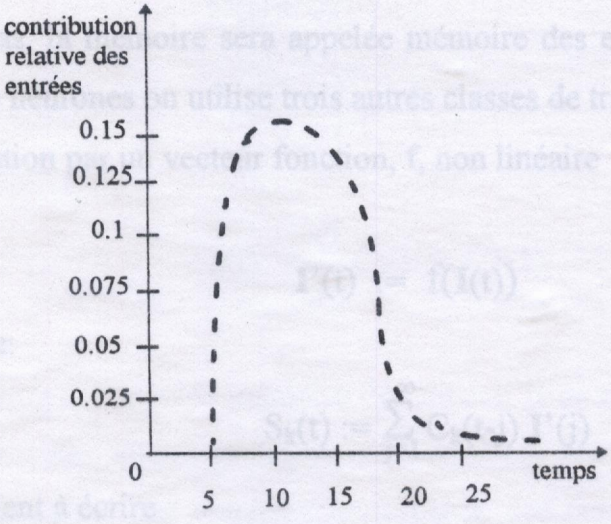


Figure II 6 : Fonction de cerne de la mémoire gamma
avec $\omega = 6$ et $\mu = 0.4$

2.2- CONTENU DE LA MEMOIRE A COURT-TERME:

La mémoire [37] doit contenir les informations appartenant à la séquence d'entrée, Elle ne doit pas être une séquence d'entrée brute. Le processus de codage (codage des entrées avant leurs présentations au réseau) doit posséder une étape de plus dans laquelle la séquence d'entrée, I(1), I(2), ..., I(t), est transformée en une nouvelle représentation I'(1), I'(2), ..., I'(t), et c'est cette représentation transformée qui est

stockée dans la mémoire. De cette façon, les S_k sont définis en terme de **I'** et non pas de **I**.

c'est à dire : $$S_k(t) = \sum_{j=1}^{\infty} C_k(t-j).\mathbf{I}'(j) \qquad (II.11)$$

au lieu de $$S_k(t) = \sum_{j=1}^{\infty} C_k(t-j).\mathbf{I}(j)$$

Dans le cas considéré dans (§.II.2.1.i), la transformation est l'identité.

$$\mathbf{I}'(t) := \mathbf{I}(t) \qquad (II.12)$$

Dans ce cas, la mémoire sera appelée mémoire des entrées. Dans la littérature sur les réseaux de neurones on utilise trois autres classes de transformation. La première est une transformation par un vecteur fonction, f, non linéaire

$$\mathbf{I}'(t) := f(\mathbf{I}(t)) \qquad (II.13)$$

c'est à dire:

$$S_k(t) := \sum_{j=1}^{\infty} C_k(t-j).\mathbf{I}'(j)$$

ce qui revient à écrire

$$S_k(t) := \sum_{j=1}^{\infty} C_k(t-j).f(\mathbf{I}(j)) \qquad (II.14)$$

Cette transformation donne pour résultat la « mémoire des entrées transformées ». En général, f est la fonction d'activation du neurone standard.

$$f(\mathbf{I}(t)) := 1/(1 + \exp(-\lambda.w'.\mathbf{I}(t)))$$

La seconde transformation non linéaire peut être calculée non seulement avec l'entrée présente, mais aussi avec l'état interne présent dans la mémoire.

$$\mathbf{I}'(t) := f(\mathbf{I}(t), S_1(t-1), S_2(t-1), \ldots, S_L(t-1)) \qquad (II.15)$$

Cette mémoire est dite mémoire des entrées et des états transformés. De telles mémoires peuvent être implantées dans un réseau de neurones récurrent dans lequel S_k et **I'** correspondent aux activités dans les deux premières couches cachées, (voir figure:II.7). Notons que cette architecture est tout à fait similaire à l'architecture d'un réseau de neurones récurrent de traitement de séquence ordinaire (figure:II.8 ; exemples: Elman, [19], [20] ; Mozer, [37]).

La troisième transformation comme les tâches séquentielles pour lesquelles la sortie désirée est à un seul pas de l'entrée. (C'est à dire $Z(t) = I(t+1)$). On peut aussi considérer l'alternative suivante :

$$\mathbf{I'}(t) = \mathbf{Z}(t - 1) \qquad \text{(II.16)}$$

$$S_k(t) = \sum_{j=1}^{\infty} C_k(t-j).\mathbf{I'}(j)$$

ce qui est équivalent à :

$$S_k(t) = \sum_{j=1}^{\infty} C_k(t-j).\mathbf{Z}(j-1) \qquad \text{(II.17)}$$

Cette mémoire est appelée mémoire des sorties. Bien entendu, des mémoires des sorties transformées, des sorties et des états transformés peuvent être construites par analogie avec les mémoires des entrées transformées, des entrées et des états transformés. Le contenu de la mémoire est donc caractérisé par la transformation et son application, c'est à dire, si elle est appliquée aux entrées ou aux sorties.

Les mémoires qui combinent les informations d'entrée et de sortie sont incluses dans la catégorie des mémoires des sorties.

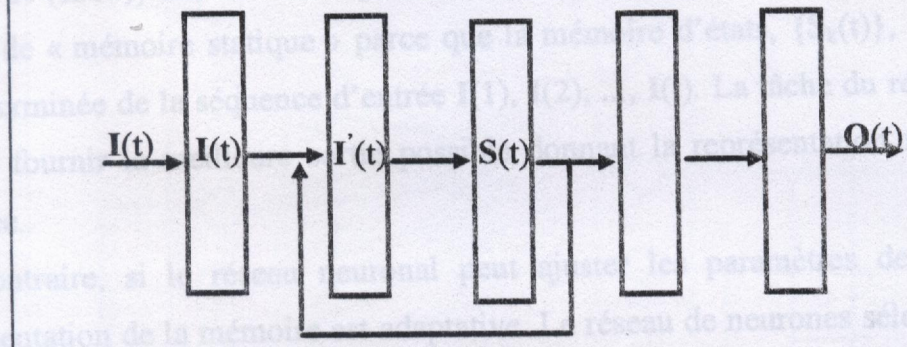


Figure II.7 : Une mémoire des entrées et des états transformés implantée dans une architecture neuronale. Chaque rectangle représente une couche d'unités de traitement. Chaque flèche représente une connectivité complète d'une couche à une autre.

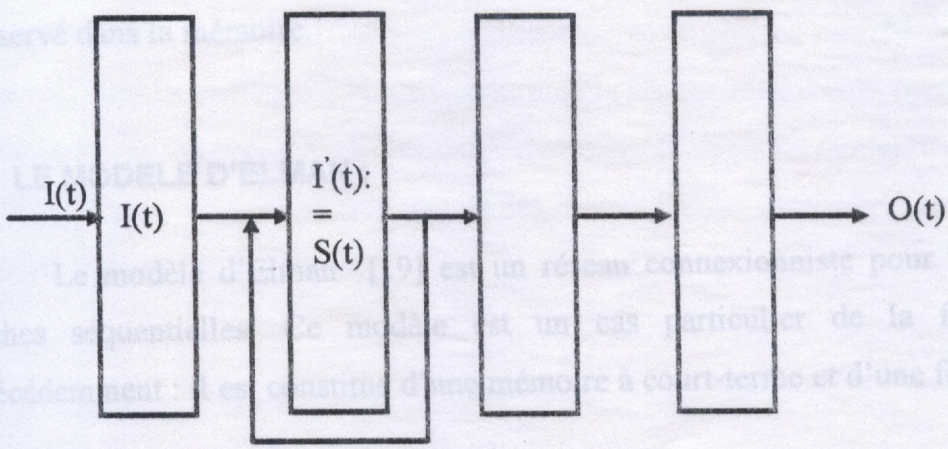


Figure II.8 : Une architecture récurrente standard avec I'(t) = S(t)

2.3- ADAPTABILITE DE LA MEMOIRE :

La dimension finale qui caractérise la mémoire à court-terme est son adaptabilité. Une mémoire possède des paramètres variables ($\{\mu_k\}$, $\{L_i\}$, et les poids W des équations

(II.13) et (II.14)) doivent être spécifiés. Si les paramètres sont fixés à priori, on peut parler de « mémoire statique » parce que la mémoire d'états, $\{S_k(t)\}$, est une fonction prédéterminée de la séquence d'entrée I(1), I(2), ..., I(t). La tâche du réseau de neurone est de fournir la meilleure sortie possible donnant la représentation fixe du passé de l'entrée.

Au contraire, si le réseau neuronal peut ajuster les paramètres de la mémoire, la représentation de la mémoire est adaptative. Le réseau de neurones sélectionne, dans les limites de sa capacité mémoire, les aspects de la séquence d'entrée qui sont utiles pour calculer la sortie désirée. Le réseau neuronal doit aussi apprendre les caractéristiques de la mémoire.

L'apprentissage de chaque architecture neuronale récurrente se fait avec l'algorithme de rétro-propagation à travers le temps [42] ou avec l'algorithme d'apprentissage récurrent à temps réel ([36]; [40]; [44]).

Le modèle de la mémoire statique peut être une approche raisonnable s'il y a un domaine adéquat de connaissance qui contient le type d'information qui doit être préservé dans la mémoire.

3.- LE MODELE D'ELMAN :

Le modèle d'Elman [19] est un réseau connexionniste pour le traitement des tâches séquentielles. Ce modèle est un cas particulier de la formulation citée précédemment : il est constitué d'une mémoire à court-terme et d'une fonction de sortie.

3. 1- MEMOIRE A COURT - TERME :

Le modèle de la mémoire à court - terme dans ce réseau correspond à un cas trivial de la mémoire à allure exponentielle des entrées et des états transformés avec $\mu_l=0$ qui est équivalente à la mémoire à retard des entrées et des états transformés avec L := 0. Le réseau réalisant la mémoire à court - terme est illustré par la figure II.9

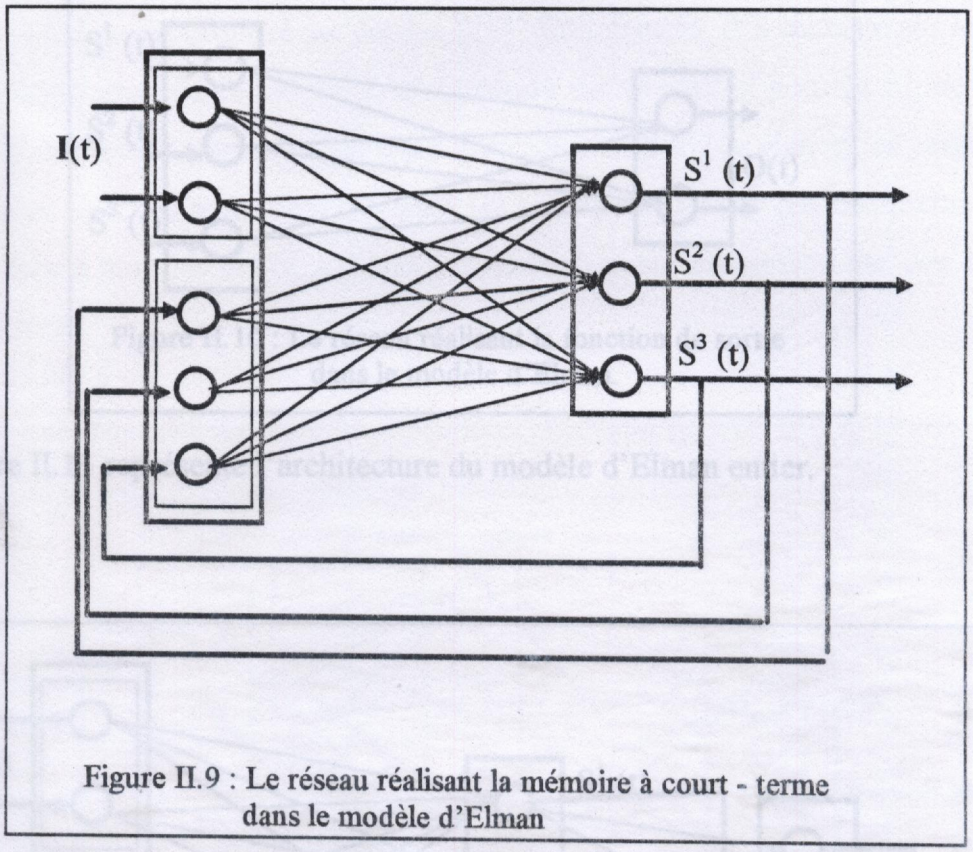


Figure II.9 : Le réseau réalisant la mémoire à court - terme dans le modèle d'Elman

La couche d'entrée consiste en deux ensembles de cellules, les cellules de contexte et les cellules d'entrée, la deuxième couche qui représente la couche cachée du réseau entier (réseau réalisant la mémoire à court-terme, plus le réseau réalisant la fonction de sortie) contient le même nombre de cellules que l'ensemble d'unités de contexte dans la couche d'entrée. Chaque cellule cachée est reliée à une seule unité de contexte par une connexion fixe, de poids égal à 1.

3.2- LA FONCTION DE SORTIE :

Le modèle réalisant la fonction de sortie est un réseau à deux couches, une couche d'entrée qui reçoit les entrées à partir du réseau réalisant la mémoire à court - terme, et une couche de sortie qui calcule la sortie, ceci est illustré par la figure II.10

Ce modèle est proposé par Jordan [25] . Initialement, il a été testé sur le problème de la prononciation de la parole. L'architecture est présentée pour la figure II.12. Les cellules d'entrée se répartissant en deux groupes : les cellules plan et les cellules d'état. Les cellules de sortie sont rebouclées sur les cellules d'état par des connexions de poids fixe, de même pour les cellules d'état qui rebouclent sur elle-même par des connexions de poids fixes.

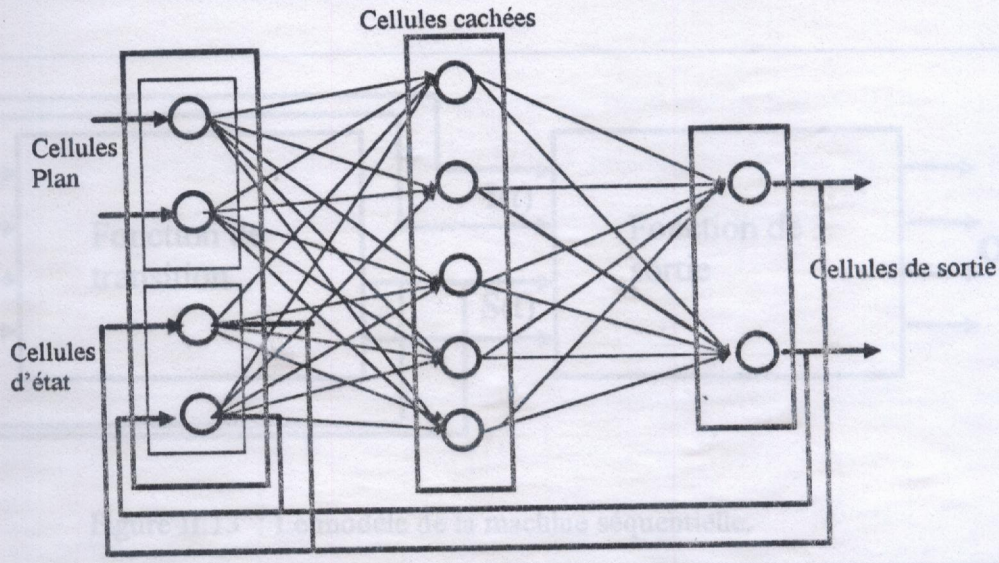


Figure II.12 : L'architecture du modèle de Jordan

Le fonctionnement : une forme d'entrées **I** est appliquée sur les cellules du plan. Ce vecteur, appelé vecteur de plan, ne sera plus modifié par la suite de la production de la séquence. Le vecteur d'état des cellules de sortie varie dans le temps du fait des connexions récurrentes sur les cellules d'état. Ces connexions modifient les valeurs d'activation des cellules d'état et imposent une entrée variable dans le temps au réseau multicouches. En utilisant des vecteurs de plan différents, le même réseau apprend plusieurs séquences différentes.

L'apprentissage : La rétro-propagation du gradient .

Pour ce réseau, il est difficile de déterminer le réseau réalisant la mémoire à court - terme et celui de la fonction de sortie.

5- LE MODELE DE LA MACHINE SEQUENTIELLE :

Le modèle de la machine séquentielle connexionniste est proposé par Touzet [49] (Touzet, 1990) pour le traitement des tâches de nature séquentielle. La formulation de la machine séquentielle est illustrée par la figure II.13.

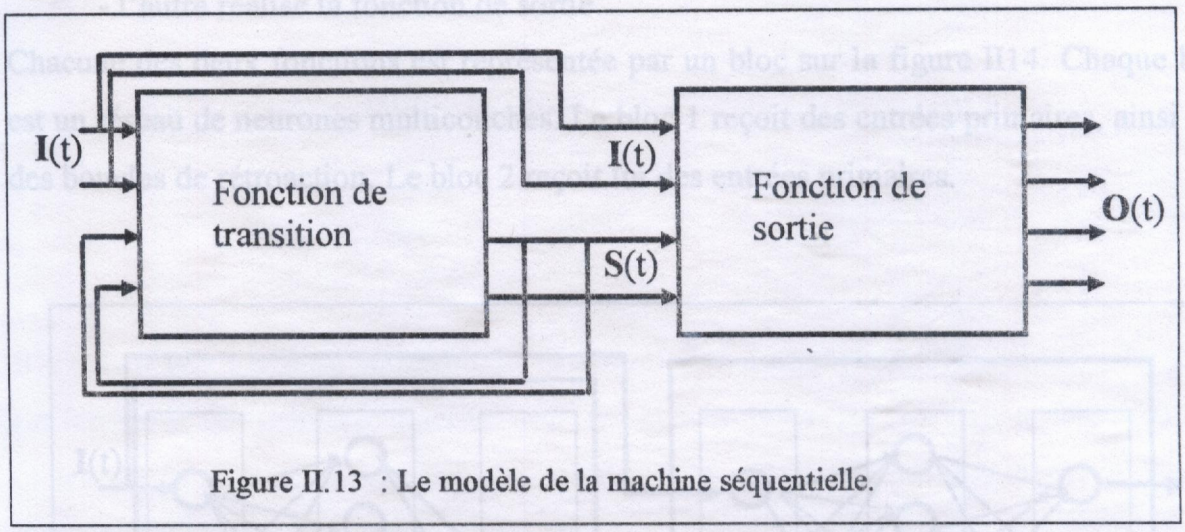


Figure II.13 : Le modèle de la machine séquentielle.

La machine séquentielle englobe deux composantes distinctes :

- La fonction de transition qui calcule l'état interne futur en fonction des formes d'entrée actuelles et de l'état interne présent. Ici, l'état interne représente la sortie de la fonction de transition.
- La fonction de sortie qui évalue la sortie en fonction de l'état interne et des entrées actuelles.

DEFINITION : Une machine séquentielle est un quintuplet

$$M := (\mathbf{I}, \mathbf{0}, \mathbf{S}, \delta, \lambda)$$

où **I, 0, S** sont respectivement les ensembles non vides, finis, des entrées des sorties et des états.

δ: **IxS** $\longrightarrow$ **S** est une fonction de transition. Elle calcule l'état interne futur en fonction de la forme d'entrée actuelle et de l'état interne présent.

λ: **IxS** $\longrightarrow$ **O** est la fonction de sortie. Elle calcule la sortie en fonction de l'état interne présent, et de la forme d'entrée actuelle.

Conformément au modèle classique de la machine séquentielle, Touzet [50] a proposé une architecture connexionniste pour la machine séquentielle. Cette architecture fait apparaître deux réseaux de neurones :

- l'un réalise la fonction de transition
- l'autre réalise la fonction de sortie

Chacune des deux fonctions est représentée par un bloc sur la figure II14. Chaque bloc est un réseau de neurones multicouches. Le bloc 1 reçoit des entrées primaires, ainsi que des boucles de rétroaction. Le bloc 2 reçoit lui des entrées primaires.

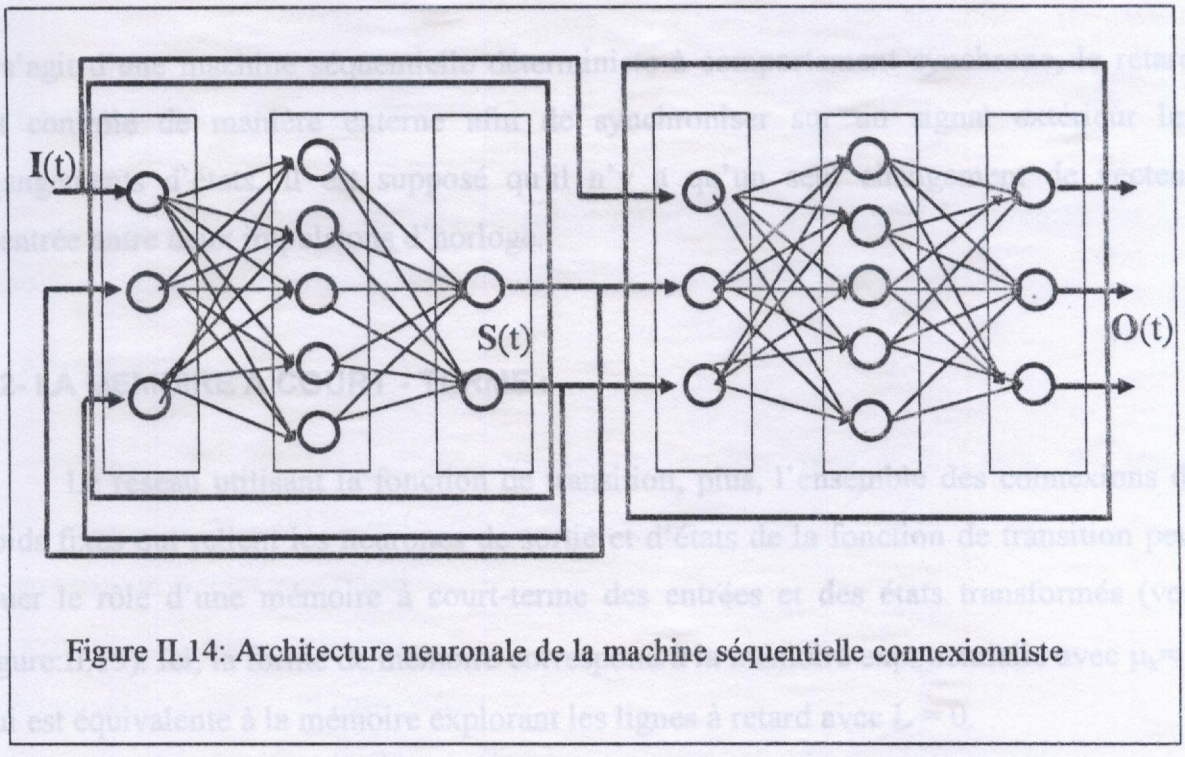


Figure II.14: Architecture neuronale de la machine séquentielle connexionniste

Les connexions en provenance des entrées primaires et celles en provenance de la sortie du réseau réalisant la fonction de transition sont de poids fixes. C'est à dire non modifiés lors de l'apprentissage. Ces connexions sont univoques : chaque neurone de

sortie est connecté à un seul neurone d'entrée. Ce dernier est appelé neurone d'entrée secondaire ou neurone d'états.

5.1- FONCTIONNEMENT :

Le réseau, réalisant la fonction de transition δ, calcule à partir de l'état interne présent **S**(t - 1) et de l'entrée actuelle **I**(t) l'état interne futur **S**(t). Le réseau réalisant la fonction de sortie λ, calcule à partir de l'état interne présent **S**(t - 1) et de la forme d'entrée actuelle **I**(t) la sortie **O**(t), c'est à dire à l'itération t on a :

$$\mathbf{S}(t) = \delta(\mathbf{S}(t-1), \mathbf{I}(t)) \qquad (II.17)$$

et

$$\mathbf{O}(t) = \lambda(\mathbf{S}(t-1), \mathbf{I}(t))$$

Il s'agit d'une machine séquentielle déterministe à comportement synchrone, le retard est contrôlé de manière externe afin de synchroniser sur un signal extérieur les changements d'états. Il est supposé qu'il n'y a qu'un seul changement de vecteur d'entrée antre deux impulsions d'horloge.

5.2- LA MEMOIRE A COURT - TERME :

Le réseau utilisant la fonction de transition, plus, l'ensemble des connexions de poids fixes qui relient les neurones de sortie et d'états de la fonction de transition peut jouer le rôle d'une mémoire à court-terme des entrées et des états transformés (voir figure:II.15). Ici, la forme de mémoire correspond à la mémoire exponentielle avec $\mu_k = 0$ qui est équivalente à la mémoire explorant les lignes à retard avec L = 0.

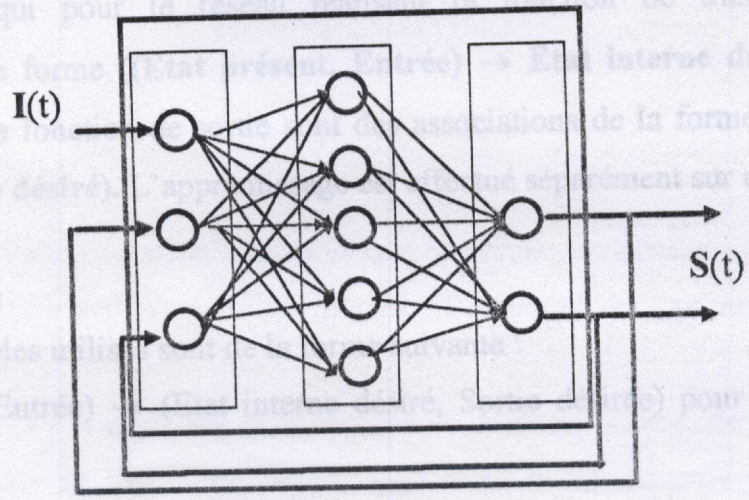


Figure II.15 : Réseau réalisant la mémoire à court terme dans le modèle de la machine séquentielle

C'est une mémoire adaptive parceque sans apprentissage ne peut pas préserver suffisamment les informations rapportées.

5.3- LA FONCTION DE SORTIE :

La fonction de sortie calcule la sortie du réseau en utilisant le contenu de la mémoire à court - terme (fonction de transition) et la forme d'entrée actuelle.

5.4- APPRENTISSAGE :

D'après Touzet [49], l'apprentissage peut être réalisé à l'aide de trois algorithmes, tous utilisant l'algorithme de Rétro-propagation du gradient :

- apprentissage à états supervisés
- apprentissage à états contraints
- apprentissage à états non supervisés

Dans cette partie nous allons nous intéresser à l'apprentissage à états supervisés.

- **APPRENTISSAGE A ETATS SUPERVISES :**

Cet algorithme est utilisé lorsqu'il est possible de définir les séquences d'apprentissage qui pour le réseau réalisant la fonction de transition sont des associations de la forme **((Etat présent, Entrée) → Etat interne désiré)** et pour le réseau réalisant la fonction de sortie sont des associations de la forme **((Etat présent, Entrée) → sortie désiré).** L'apprentissage est effectué séparément sur chaque réseau.

ALGORITHME :

Les exemples utilisés sont de la forme suivante :

{((Etat présent, Entrée) → (Etat interne désiré, Sortie désirée) pour la forme j de la séquence k.

Codage par défaut ou par l'utilisateur des états internes

POUR chacune des séquences de la base d'apprentissage FAIRE

POUR chacune des formes de la séquence FAIRE

DEBUT

// Calcul de l'état du réseau par la propagation

Présenter la forme en entrée des réseaux de transitions et de sortie

Remettre à jour l'état interne

Propagation sur les réseaux de transition et de sortie

Si erreur sur la fonction de transition ALORS

// Apprentissage sur la fonction de transition

Rétropropagation sur le réseau de transition

// Le codage des états fournit les données nécessaires à l'évaluation de

// l'erreur sur la couche de sortie du bloc de transition d'état

Si erreur sur la fonction de sortie ALORS

// Apprentissage sur la fonction de sortie

Rétropropagation sur le réseau de sortie

// L'erreur est la différence entre la valeur désirée de la sortie de la // machine et la valeur calculée pour la propagation

FIN POUR

FIN.

MODELES CONNEXIONNISTES ET PROCESSUS DE MARKOV A TEMPS CONTINU

1- INTRODUCTION :

Dans ce chapitre , nous montrons que le réseau connexionniste **MADALINE** (voir chapitre I.§3) peut être utilisé pour le traitement du processus de Markov homogène à temps continu.

Après un bref rappel sur le processus de Markov homogène à temps continu ,nous déterminons le lien entre le **MADALINE** et les équations de **Chapman-Kolmogorov**. Dans une seconde phase, nous appliquons ce résultat à l'analyse markovienne de fiabilité d'un système de production. Nous traitons ensuite une extension de cette application dans le cas non markovien.

2- PROCESSUS DE MARKOV HOMOGENE A TEMPS CONTINU:

On appelle processus de Markov homogène à temps continu , un processus aléatoire $\{X_t\}_{0\le t\le+\infty}$ homogène à espace d'états E discret tel que :

$$P(X(t_n) = x_{in} / X(t_{n-1}) = x_{in-1} , X(t_{n-2}) = x_{in-2} ,\ldots., X(t_1) = x_{i1}) =$$
$$= P(X(t_n - t_{n-1}) = x_{in} / X(0) = x_{in-1}) \qquad \text{(III.1)}$$

et ce ci quelque soit $0 \le t_1 \le t_2 \le \ldots \le t_{n-1} \le t_n$.

Pour définir un processus homogène de Markov à temps continu, il est nécessaire de définir:

1) N : le nombre d'états du modèle et $E=\{x_1 , x_2 ,\ldots., x_N\}$ l'ensemble des états du modèle

2) la matrice des taux instantanés des transitions

$A= (\rho_{i,j})$ $1\le i,j \le N$ avec

$$\rho_{ij} := \lim_{\Delta t \to 0} (1/\Delta t).P(X(t+\Delta t) = x_j / X(t) = x_i)$$

3) la distribution initiale des états

$P_0 = (P_{01}, P_{02}, ..., P_{0N})$ où $P_{0i} = P(X(0) = x_i)$ pour i:=1,2,....,N

Soit $P_j(t) = P(X(t) = x_j)$ (j =1,2,...,N) la probabilité d'être dans un état j à l'instant t. On montre que ces probabilités sont solution des équations de Chapman-Kolmogorov qui sont de la forme suivante [22]:

$$dP_i(t)/dt = \sum_{\substack{j=1 \\ j \neq i}}^{N} (P_j(t).\rho_{ji}) + P_i(t).\rho_{ii} \qquad \text{(III.2)}$$

pour i:= 1,...,N , avec:

$$\rho_{ii} = -\sum_{\substack{j=1 \\ j \neq i}}^{N} \rho_{ij}$$

La forme discrétisée de ces équations est donnée à $O(\Delta t)$ prés par :

$$P_i(t+\Delta t) = \sum_{\substack{j=1 \\ j \neq i}}^{N} (P_j(t).\rho_{ji}(\Delta t)) + P_i(t).(1+\rho_{ii}.(\Delta t)) \qquad \text{(III.3)}$$

pour i:= 1,2,...,N et Δt une quantité très petite. Cette dernière équation est une approximation du fait qu'on a négligé certains termes dans la partie droite qui sont d'ordre $O(\Delta t)$, où O(.) est le symbole de Landau.

3- LIEN ENTRE LES EQUATIONS DE CHAPMAN-KOLMOGOROV ET LE MADALINE:

Considérons la forme discrète des équations de Chapman-Kolmogorov:

$$P_i(t+\Delta t) = \sum_{\substack{j=1 \\ j \neq i}}^{N} (P_j(t).\rho_{ji}(\Delta t)) + P_i(t).(1+\rho_{ii}.(\Delta t))$$

pour i:=1,2,...,N

Il est clair que chacune des équations ci-dessus peut être implantée dans un réseau neuronal linéaire (ADALINE), (figure III.1), avec N entrées ,pas de couches cachées ,et une seule unité de sortie. Les quantités ($\rho_{1,i}.(\Delta t)$,$\rho_{2,i}.(\Delta t)$,....., $\rho_{i-1,i}.(\Delta t)$, $\rho_{i,i}.(\Delta t)$, $\rho_{i+1,i}.(\Delta t)$,..., $\rho_{N,i}.(\Delta t)$) représentent les poids des connexions

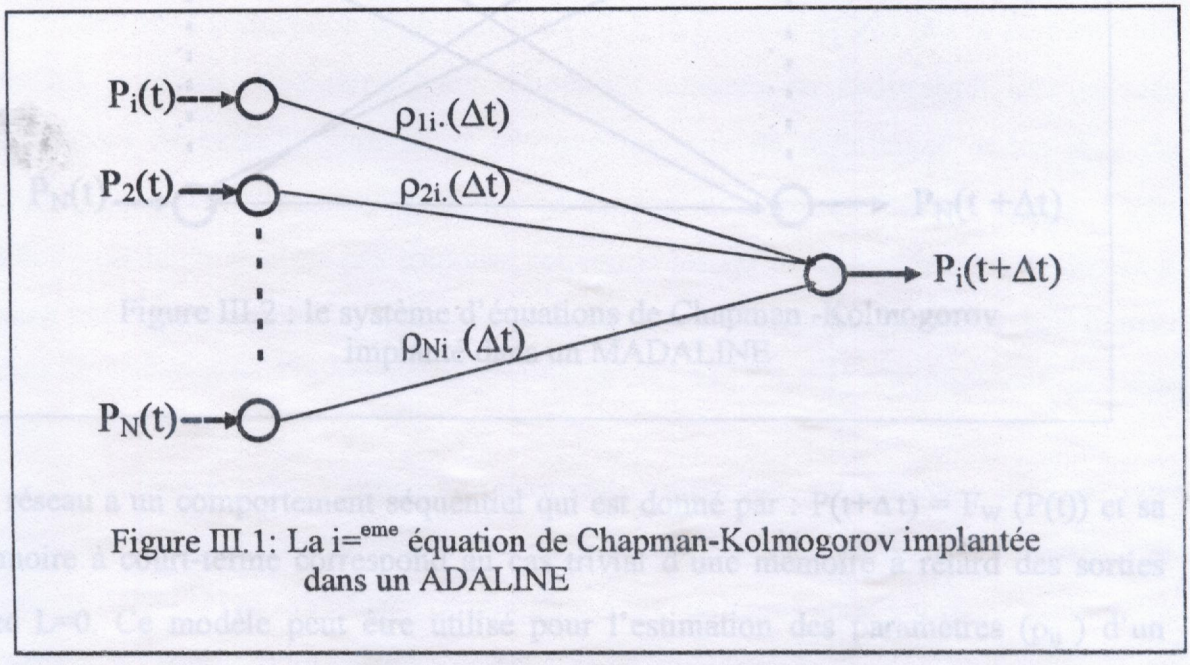


Figure III.1: La i=^{eme} équation de Chapman-Kolmogorov implantée dans un ADALINE

L'ensemble des équations peuvent être implantées dans un MADALINE comme le montre la figure ci dessous

4- APPLICATION A L'ANALYSE DE FIABILITE D'UN SYSTEME:

La fiabilité ([1],[7],[38]) est la probabilité d'un système a accomplir une mission , et la disponibilité est la probabilité d'un système a accomplir une tâche lorsqu'il est sollicité. Ces quantités constituent les caractéristiques principales de tous les systèmes que le concepteur se doit de choisir de manière adéquate.

Lors de la conception d'un système [21], la fiabilité exigée (ou la disponibilité pour les systèmes réparables) est d'habitude atteinte en choisissant des valeurs convenables de certains paramètres tels que les taux de défaillance et les taux de réparation des éléments, en utilisant les techniques usuelles de fiabilité exigée. Cette méthode est lente et n'est pas toujours commode lorsqu'on considère des systèmes à espace des états complexes, tels que les systèmes informatiques tolérants aux fautes. Même les techniques d'optimisation proposées parfois [32],[33] n'aboutissent qu'à des résultats approximatifs.

L'utilisation de l'approche neuronale présentée ci-dessous permet une détermination rapide et plus exacte des paramètres. Elle a été utilisée par Manzoul et Mamoun dans [32] et [33]. Dans ce chapitre, nous utilisons cette approche pour faire l'analyse de fiabilité d'un système de production dans le cadre d'un projet proposé par l'institut national de recherche en maintenance (INMA), et portant sur la conception d'une usine de production de matériel didactique que nous présentons ci- dessous.

4.1-PRESENTATION DE L'USINE:

L'usine [11],[43] a pour mission , la production du matériel didactique dont le manque se fait ressentir dans les laboratoires des universités, des centres de formations , etc... .

La configuration optimale de l'usine trouvée dans [11] est constituée de neuf ateliers assurant chacun une seule opération qui est lui propre.

La description des équipements de production est la suivante:

Atelier N° i	Type de machines et opération	nombre de machines
1	Cisaille (Cisaillement)	2
2	Presse (pressage)	2
3	Perceuse (Perçage)	4
4	Plieuse	2
5	Cintreuse	2
6	Groupe de soudure	2
7	Fer à souder	2
8	Poste de peinture	2
9	Poste de sérigraphie	1

L'opération assemblage est assurée par un atelier qui n'entre pas dans le domaine de notre intérêt.

Comme nous l'avons déjà signalé ,le problème consiste à déterminer les caractéristiques internes du système afin d'atteindre les fiabilités désirées. Ici nous allons considérer deux situations:

1. cas où il n'y a pas de réparateur: On cherche à déterminer les taux de défaillance des éléments de chaque atelier afin de satisfaire la fiabilité désirée . Les éléments d'un même atelier sont identiques , c'est à dire qu'ils ont la même fonction de fiabilité ($R_i(t):= \exp(-\lambda_i.t)$), où λ_i représente le taux de défaillance (i représente l'indice du $i =^{ieme}$ atelier i:=1,...,9).
2. cas où on prévoit un réparateur au niveau de chaque atelier: On cherche à déterminer les taux de défaillance des éléments et les taux de réparation de chaque atelier. La durée de service de chaque réparateur suit une loi exponentielle de paramètre μ_i (i:=1,...,9) inconnu qui représente le taux de réparation. Les mêmes hypothèses que 1 sur les éléments sont considérées ici.

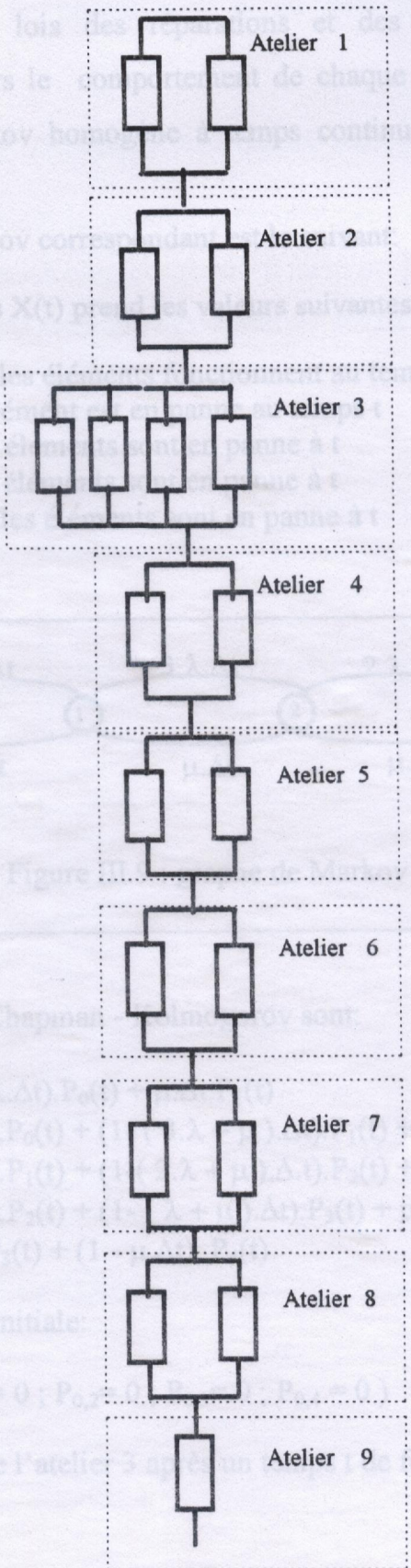


Figure III.3 : Schéma de la configuration de l'usine

4.2 - LES MODELES DE MARKOV CORRESPONDANT A CHAQUE ATELIER:

Puisque les lois des réparations et des pannes de chaque atelier sont exponentielles, alors le comportement de chaque atelier peut être modélisé par un processus de Markov homogène à temps continu. Nous illustrons la methode sur l'atelier 3.

Le modèle de Markov correspondant est le suivant:

Le processus X(t) prend les valeurs suivantes:

$$X(t):=\begin{cases} 0 \text{ si tous les éléments fonctionnent au temps t} \\ 1 \text{ si un élément est en panne au temps t} \\ 2 \text{ si deux éléments sont en panne à t} \\ 3 \text{ si trois éléments sont en panne à t} \\ 4 \text{ si tous les éléments sont en panne à t} \end{cases}$$

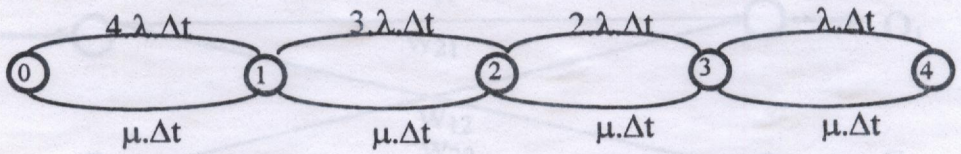


Figure III.9 : graphe de Markov associe

Les équations de Chapman - Kolmogorov sont:

$$\begin{aligned} P_0(t+\Delta t) &= (1 - 4.\lambda.\Delta t).P_0(t) + \mu.\Delta t.P_1(t) \\ P_1(t+\Delta t) &= 4.\lambda.\Delta t.P_0(t) + (1-(3.\lambda+\mu).\Delta t).P_1(t) + \mu.\Delta t.P_2(t) \\ P_2(t+\Delta t) &= 3.\lambda.\Delta t.P_1(t) + (1-(2.\lambda+\mu).\Delta.t).P_2(t) + \mu.\Delta t.P_3(t) \\ P_3(t+\Delta t) &= 2.\lambda.\Delta t.P_2(t) + (1-(\lambda+\mu).\Delta t).P_3(t) + \mu.\Delta t.P_4(t) \\ P_4(t+\Delta t) &= \lambda.\Delta t.P_3(t) + (1 - \mu.\Delta t).P_4(t) \end{aligned} \qquad \text{(III.6)}$$

sous la condition initiale:

$$P_0 = (P_{0,0}= 1 ; P_{0,1}= 0 ; P_{0,2}= 0 ; P_{0,3}= 0 ; P_{0,4} = 0)$$

La disponibilité de l'atelier 3 après un temps t de fonctionnement est:

$D_3(t) = P_0(t) + P_1(t) + P_2(t) + P_3(t)$

Pour évaluer la fiabilité, il faut rendre l'état 4 absorbant. Le cas où il n'y a pas de réparateur s'obtient en posant $\mu = 0$, au quel cas la fiabilité coïncide avec la disponibilité..

4.3 LEs MODELE NEURONAUX ASSOCIES AUX ATELIERS:

En utilisant la relation présentée dans (§ III.3), on fait correspondre aux équations de Chapman-Kolmogorov du modèle un MADALINE que nous présentons ci-dessous .Le MADALINE correspondant à l'atelier 3:

le MADALINE associé aux équations (III.6) de l'atelier 3 est donné par la figure suivante:

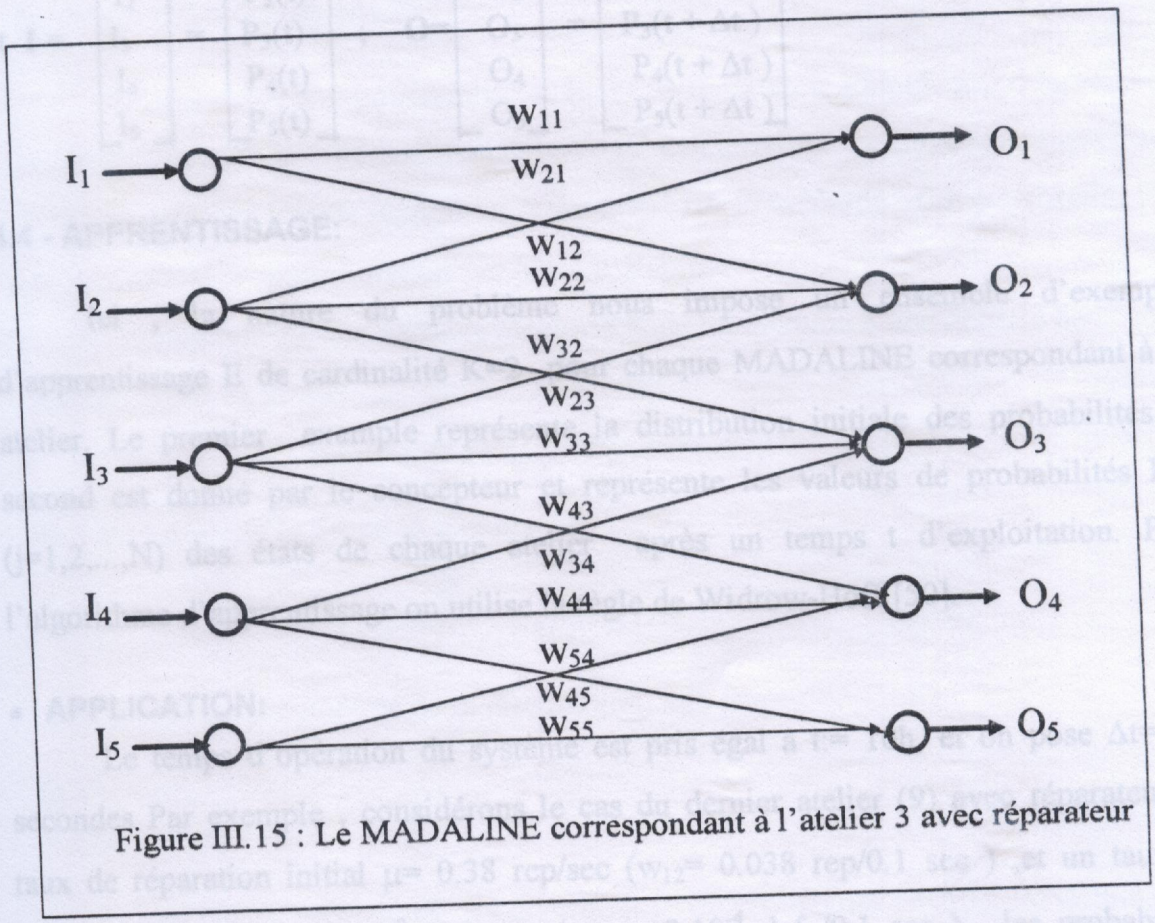


Figure III.15 : Le MADALINE correspondant à l'atelier 3 avec réparateur

Les équations de base du réseau sont:

$O_1 = w_{11}.I_1 + w_{12}.I_2$
$O_2 = w_{21}.I_1 + w_{22}.I_2 + w_{32}.I_3$
$O_3 = w_{32}.I_2 + w_{33}.I_3 + w_{34}.I_4$
$O_4 = w_{43}.I_3 + w_{44}.I_4 + w_{45}.I_5$
$O_5 = w_{54}.I_4 + w_{55}.I_5$

avec les relations suivantes:

$w_{11} = 1-4.w_{54}$; $w_{32} = 3.w_{54}$; $w_{44} = 1-(w_{54} + w_{12})$
$w_{21} = 4.w_{54}$; $w_{33} = 1-(2.w_{54} + w_{12})$; $w_{54} = \lambda.\Delta t$
$w_{22} = 1-(3.w_{54} + w_{12})$; $w_{43} = 2.w_{54}$; $w_{55} = 1 - w_{54}$
$w_{23} = w_{12}$; $w_{12} = \mu.\Delta t$; $w_{34} = w_{12} = w_{45}$

et $\mathbf{I} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \end{bmatrix} = \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \\ P_4(t) \\ P_5(t) \end{bmatrix}$; $\mathbf{O} = \begin{bmatrix} O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \end{bmatrix} = \begin{bmatrix} P_1(t+\Delta t) \\ P_2(t+\Delta t) \\ P_3(t+\Delta t) \\ P_4(t+\Delta t) \\ P_5(t+\Delta t) \end{bmatrix}$

4.4 - APPRENTISSAGE:

Ici , la nature du problème nous impose un ensemble d'exemples d'apprentissage E de cardinalité K=2 pour chaque MADALINE correspondant à un atelier. Le premier exemple représente la distribution initiale des probabilités, le second est donné par le concepteur et représente les valeurs de probabilités $P_j(t)$ (j=1,2,...,N) des états de chaque atelier après un temps t d'exploitation. Pour l'algorithme d'apprentissage on utilise la règle de Widrow-Hoff [50].

- **APPLICATION:**

Le temps d'opération du système est pris égal à t:= 10h, et on pose Δt= 0.1 secondes Par exemple , considérons le cas du dernier atelier (9) avec réparateur de taux de réparation initial μ= 0.38 rep/sec (w_{12}= 0.038 rep/0.1 sec) ,et un taux de défaillance initial $\lambda = 3.10^{-3}$ def /sec ($w_{21} = 3.10^{-4}$ def /0.1 sec) , les probabilités désirées étant $P_0(10h) = 0.9912$ et $P_1(10h) = 8.08.10^{-3}$. Le réseau converge au bout de 12 itérations avec un pas de calcul du gradient α:= 0.01 vers $P_0(10h)$:= 0.992 et

$P_1(10h) = 7.969.10^{-3}$. Les résultats sont obtenus avec une limite de précision $\varepsilon := 2.10^{-4}$ par rapport aux valeurs désirées . Les différents poids à la convergence sont: $w_{12} = 1.97724.10^{-4}$ rep/ 0.1sec et $w_{21} = 3.8.10^{-2}$ def /0.1 sec . A partir de ces valeurs, on calcule les taux de défaillance et de réparation exigés comme suit:

$\lambda = w_{21} / \Delta t = 1.97724.10^{-4}$ déf / sec
$\mu = w_{12} / \Delta t = 3.8.10^{-1}$ rep/ sec

Pour vérifier ces résultats , les valeurs de λ ,μ et du temps (t=10 h) sont substituées dans la solution à temps continu des équations de Chapman-Kolmogorov. On obtient les mêmes valeurs pour $P_0(t)$ et $P_1(t)$. Les résultats expérimentaux pour chaque atelier sont consignés dans les tableaux (annexe 1) .

5 - EXTENSION DE LA METHODE AU CAS D'UNE DISTRIBUTION DE REPARATION GENERALE:

L'hypothèse d'exponentialité de la loi de réparation adoptée précédemment , n'est pas toujours justifiée . On s'intéresse ici à déterminer les caractéristiques du système pour des lois de réparation arbitraires , par la méthode ' des états fictifs ' ,moyennant une approximation par la loi d'**Erlang**.

Rappelons qu'une variable aléatoire X de loi Gamma a une densité de probabilité de la forme:

$$f(x) := (1/ \Gamma(\alpha).). x^{\alpha-1}.\exp(-x /\beta) \qquad \text{(III.7)}$$

où $\Gamma(\alpha)$ désigne la fonction gamma

$$\Gamma(\alpha) = \int_0^{\infty} x^{\alpha-1}. \exp(-x).dx \qquad \text{(III.8)}$$

et α, β sont les paramètres de la distribution, tels que:

$E(X) := \alpha\beta$
$VAR(X) := \alpha.\beta^2$

Une classe spéciale de ces distributions de probabilité Gamma est donnée par le cas où k est un entier positif quelconque. Dans ce cas, on obtient la loi d'**Erlang** de densité:

$$f(x) = (\mu^k / (k-1)!).x^{k-1} .\exp(-\mu.x) \quad (0 < x < \infty) \qquad \text{(III.9)}$$

avec $\alpha = k$ et $\beta = 1/\mu$. Les paramètres de la distribution d'**Erlang** sont donc k et μ . La moyenne et la variance sont données par:

$E(X) := k/\mu := 1/\mu'$
$VAR(X) := k/\mu^2$

5.1 - LA METHODE DES ETATS FICTIFS:

Cette méthode est utilisée de manière plus générale lorsqu'il s'agit d'approcher les caractéristiques de modèles stochastiques complexes, et consiste a approcher les distributions constituantes (lois de réparation arbitraires dans notre cas), par des distributions plus simple. Dans la plupart des problèmes de modélisation stochastique, il est souvent plus commode d'approcher ces distributions constituantes par une distribution construite à partir d'une somme (ou d'un mélange) finie de lois exponentielles. Chaque loi exponentielle peut être assimilée au temps de séjour à un état fictif, d'ou le nom de la méthode. On l'appelle parfois en théorie de fiabilité, méthode de COX, et en théorie des files d'attente, méthode d'Erlang ou des étapes. On peut considérer d'une manière plus générale les distributions P H (Phase-type) définies comme la loi du temps de premier passage dans une chaîne de Markov à temps continu. En pratique, on se limite à des distributions particulières d'Erlang avec les mêmes paramètres. La justification théorique de l'utilisation de mélanges de distributions d'Erlang est donnée par exemple dans Stoyan [46]. Soit G(t) la fonction de répartition du temps de réparation qui est une distribution de probabilité sur $[0 , \infty]$. On définit pour $\Delta > 0$, la distribution de probabilité :

$$F_\Delta(t) = \sum_{k=1}^{\infty} P_{k,\Delta} \left\{1 - \sum_{j=1}^{k-1} e^{-t/\Delta} (t/\Delta)^i / i!\right\}, \; t \geq 0 \qquad \text{(III.10)}$$

Où $P_{k,\Delta} = F(k\Delta) - F((k-1).\Delta)$, $k = 1,2,....$

On montre [46] que $F_\Delta \Rightarrow F$ i.e. $\lim_{t\to\infty} F_\Delta(t) = f(t)$, en tout point t où F est continue.

En effet, (III.10) est équivalent à

consiste a décomposer le passage de l'état i à l'état j en k transitions (voir figure III.16) correspondant à une même loi exponentielle de paramètre μ.

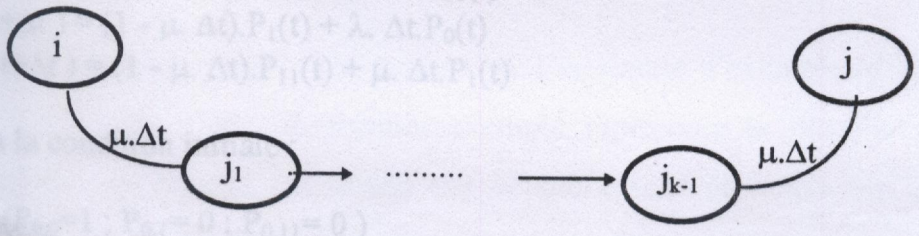


Figure III.16 : Graphe de MARKOV correspondant à la transformation par la méthode des états fictifs d'une transition d'Erlang d'un état i à un état j.

Les k-1 états ($j_1, j_2, \ldots, j_{k-1}$) sont fictifs ,mais permettent de transformer le processus en un processus markovien , en vertu de l'absence de mémoire de la loi exponentielle.

On prend par exemple , le neuvième atelier, qui correspond à une machine de sérigraphie, et on utilise une approximation d'Erlang d'ordre 2 pour la durée de réparation

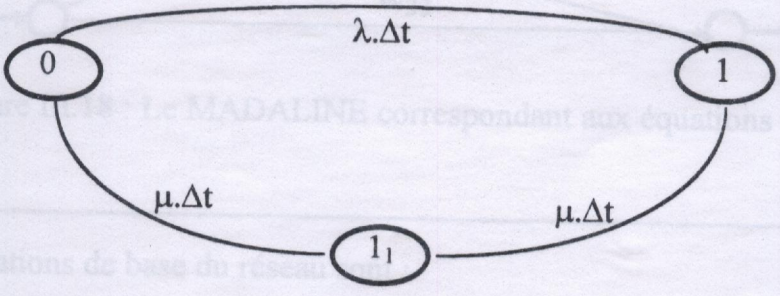


Figure III17: graphe de Markov correspondant (l'état 1_1 est fictif)

Les équations de Chapman- Kolmogorov:

$P_0(t+\Delta t) = (1 - \lambda.\Delta t).P_0(t) + \mu.\Delta t.P_{11}(t)$
$P_1(t+\Delta t) = (1 - \mu.\Delta t).P_1(t) + \lambda.\Delta t.P_0(t)$
$P_{11}(t+\Delta t) = (1 - \mu.\Delta t).P_{11}(t) + \mu.\Delta t.P_1(t)$

sous la condition initiale :

$P_0 = (P_{0,0}=1 ; P_{0,1}= 0 ; P_{0,11}= 0)$

La disponibilité de l'atelier au temps t est :

$D_s(t) = P_0(t)$

Le réseau MADALINE correspondant est représenté par la figure ci- dessous:

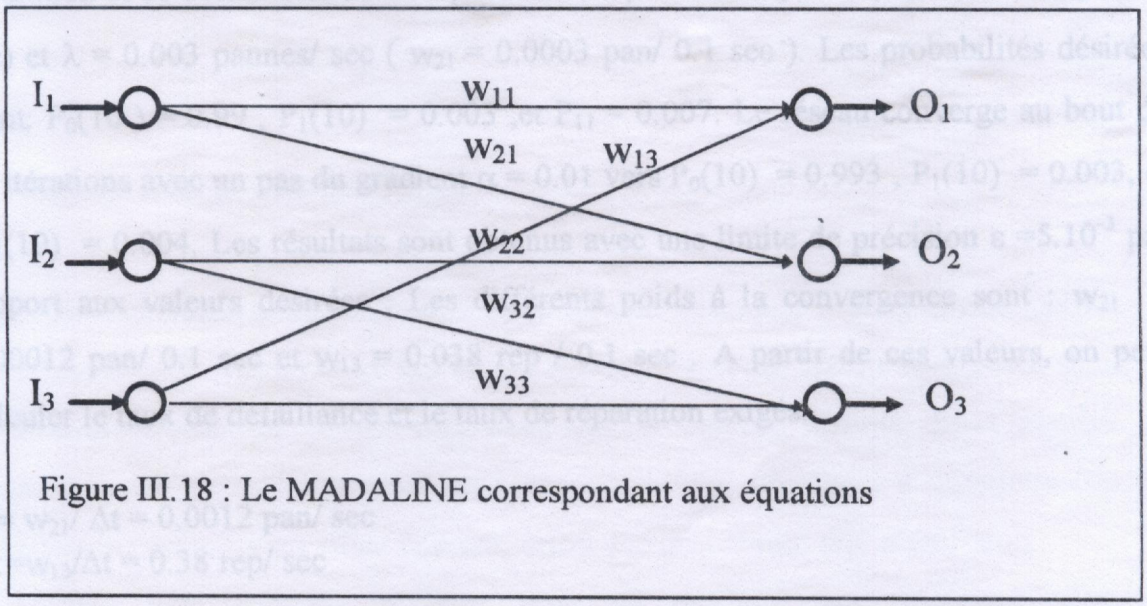


Figure III.18 : Le MADALINE correspondant aux équations

Les équations de base du réseau sont :

$O_1 = w_{11}.I_1 + w_{13}.I_3$
$O_2 = w_{21}.I_1 + w_{22}.I_2$
$O_3 = w_{32}.I_2 + w_{33}.I_3$

avec les relations suivantes:

$w_{11} = 1-w_{21}$; $w_{32} = w_{13}$
$w_{21} = \lambda.\Delta t$; $w_{33} = 1-w_{13}$
$w_{22} = 1-w_{13}$; $w_{13} = \mu.\Delta t$

LES CHAINES DE MARKOV CACHEES

1- INTRODUCTION:

L'avantage des modèles markoviens par rapport à plusieurs autres types de modélisation, réside dans le fait qu'ils sont fondés sur des bases théoriques fortes et rigoureuses. Ce qui explique leur grande utilité dans plusieurs domaines d'application.

Dans ce chapitre, nous allons définir dans un premier temps, les chaines markoviennes cachées standards. Après quoi, nous allons voir comment résoudre les problèmes liés à leurs utilisation dans le monde des applications pratiques.

Les définitions et les notations utilisées sont basées sur celles employées dans [8]; [10] ; [14]; [26]; [28]

2- PRESENTATION DU MODELE:

Soit une chaîne de Markov, c'est à dire un ensemble d'états et une matrice de probabilités de transition entre ces états : A chaque instant, cette « machine » saute de l'état courant vers un nouvel état, avec une probabilité déduite de la matrice de transitions. On suppose en outre qu'à chaque instant, un signal élémentaire (on dit observation) est émis par l'état courant, selon une probabilité dite d'émission.

Une telle machine a pour effet de produire une séquence aléatoire d'observations, et constitue de fait un modèle de production de séquences paramètrées par la matrice des probabilités de transition d'une part, et une matrice de probabilités d'émission d' autre part.

2.1- DEFINITION [26] :

Un modèle markovien caché est une chaîne de markov homogène générant un processus stochastique à deux composantes (X , Y): l'une cachée (X) et l'autre observable (Y).

Un modèle de Markov caché sera noté **HMM**.

i) La loi de X :

Soit $X := (X_t)_{t \geq 1}$ une chaîne de markov homogène qui prend ses valeurs dans l'ensemble des états $E = \{x_1, x_2, \ldots\ldots, x_N\}$

Nous supposons par la suite que :

$$P(X_{t+1} = x_j / X_t = x_i) = a_{ij}$$

pour $1 \leq i, j \leq N$ et $t \geq 1$, et la loi initiale est donnée par :

$$P(X_1 = x_i) = a_{0i}$$

pour i:= 1,2,...,N

Ainsi la loi de probabilité du vecteur aléatoire $X^T = (X_1, X_2, \ldots, X_T)$ est donnée par :

$$P(X^T) := P(X_1, X_2, \ldots, X_T) = \prod_{t=1}^{T} a_{X_{t-1}\, X_t}$$

avec $X_0 = 0$

ii) La loi de (X , Y) :

Soit $Y := (Y_t)_{t \geq 1}$ une famille de variables aléatoires qui prend ses valeurs dans l'ensemble des symboles d'observation $V = \{v_1, v_2, \ldots, v_M\}$

La loi de X étant définie ci-dessus, il reste à définir les lois de Y conditionnelle à X. Nous supposons que les variables $(Y_t)_{t \geq 1}$ sont indépendantes conditionnellement par rapport à X, c'est à dire :

$$P(Y_1, Y_2, \ldots, Y_T / X_1, X_2, \ldots, X_T) = \prod_{t=1}^{T} P(Y_t / X_1, X_2, \ldots, X_T)$$

De plus, la loi de chaque Y_t conditionnelle à X est égale à sa loi conditionnelle à X_t, c'est à dire :

$$P(Y_1, Y_2, ..., Y_T / X_1, X_2, ..., X_T) = \prod_{t=1}^{T} P(Y_t / X_t)$$

Nous noterons par $b_{Xt}(Y_t)$ les probabilités de Y_t conditionnelles à X_t. Ce qui revient à écrire :

$$b_{X_t}(Y_t) = P(Y_t / X_t)$$

donc

$$P(Y_1, Y_2, ..., Y_T, X_1, X_2, ..., X_T) = P(Y_1, Y_2, ..., Y_T / X_1, X_2, ..., X_T).$$

$$.P(X_1, X_2, ..., X_T) = \prod_{t=1}^{T} a_{X_{t-1}, X_t} . b_{X_t}(Y_t)$$

2.2- ELEMENTS D'UN MODELE DE MARKOV CACHE:

Les notations suivantes sont utilisées pour désigner les éléments d'un HMM.

1 - N : Le nombre d'états du modèle
E : est l'ensemble des états de la chaîne de Markov cachée
Nous désignons un état au temps t par X_t ($X_t \in E$).

2 - M : Le nombre de symboles d'observations distincts.
$V = \{v_1, v_2, ..., v_M\}$ est l'ensemble des symboles d'observations.
Nous désignons un symbole au temps t par Y_t ($Y_t \in V$).

3 - La distribution des probabilités de transitions des états

$$A = (a_{ij}) \text{ avec } 1 \le j, i \le N$$

où $a_{ij} = P(X_{t+1} := x_j / X_t := x_i)$

sous la contrainte:

$$\sum_{j=1}^{N} a_{ij} := 1$$

4 - La distribution d'être initialement dans un état: $A_0 = (a_{0i})$ avec $1 \le i \le N$
où $a_{0i} := P(X_1 := x_i)$ sous la contrainte:

$$\sum_{i=1}^{N} a_{0,i} = 1$$

5- La distribution des probabilités des symboles d'observation dans chaque état x_j.

$$B := (b_j(k)) \quad \text{avec } 1 \le j \le N \text{ et } 1 \le k \le M$$

où $b_j(k) = P(Y_t := v_k / X_t := x_j)$

sous la contrainte

$$\sum_{k=1}^{M} b_j(k) := 1$$

En résumé, les éléments d'un HMM sont : N, M et $\lambda := \{a_{0i}, a_{ij}, b_j(k) : 1 \le i, j \le N \text{ et } 1 \le k \le M\}$. λ est appelé ensemble des paramètres du modèle. Dans la suite, on notera P(.) les probabilités réelles et P_λ(.) les modèles paramétriques de ces probabilités.

2.3- GENERATION DES OBSERVATIONS DANS UN HMM:

ETAPE 1

Poser t = 1
Choisir un état initial $X_1 = \omega_i$ avec une distribution a_{0i} où $1 \le i \le N$
Poser l = i

ETAPE 2

Choix de l'observation $Y_t = v_k$ avec une probabilité $b_j(k)$ où $1 \le k \le M$

ETAPE 3

Transition au nouvel état $X_{t+1} = \omega_j$ avec une probabilité a_{ij} avec $1 \le j \le N$

ETAPE 4

t = t + 1
Si $t \le T$; poser l = j ; aller à l'étape 2
Si non fin.

T représente la longueur de la séquence des observations générée.

3- DETERMINATION DES PARAMETRES D'UN MODELE DE MARKOV CACHE:

On dispose maintenant d'un échantillon de séquences d'observations aléatoires $Y^k = (Y_1^k, Y_2^k, ..., Y_N^k)$ k = 1, 2, ... et l'on cherche à estimer le meilleur ensemble de paramètres λ^* du HMM, c'est à dire celui qui conduit à la détermination des meilleures distributions dans V^T des séquences d'observations de longueur T.

3.1- CALCUL DE LA VRAISEMBLANCE D'UNE OBSERVATION:

i) CALCUL DIRECT:

$$P_\lambda(Y_1, Y_2, ..., Y_T) = \sum_{(X_1,...,X_T)\in\Gamma} P_\lambda(Y_1,...,Y_T / X_1,..., X_T)$$

$$= \sum_{(X_1,...,X_T)\in\Gamma} P_\lambda(Y_1,...,Y_T / X_1,..., X_T).P_\lambda(X_1,..., X_T)$$

$$= \sum_{(X_1,...,X_T)\in\Gamma} \prod_{t=1}^{T} a_{X_{t-1},X_t}.b_{X_t}(Y_t)$$

Finalement on a :

$$P_\lambda(Y_1, Y_2, ..., Y_T) := \sum_{(X_1,...,X_T)\in\Gamma} \prod_{t=1} a_{X_{t-1},X_t}.b_{X_t}(Y_t)$$

Notons par Γ :l'ensemble de toutes les séquences d 'états , c'est à dire $\Gamma = E^T$

Il nous faut donc déterminer le vecteur λ^* qui maximise la vraisemblance des séquences d'observations de référence, tout en respectant les contraintes imposées par la nature probabiliste de ces paramètres.

Remarque :

Le calcul direct de la vraisemblance est long. Les variables de **Forward** et **Backward** permettent un calcul récursif de cette quantité.

ii) CALCUL RECURSIF :

Soit $\alpha_t(i)$: la probabilité de la suite d'observation partielle de 1 à t, se terminant à l'état $X_t = x_i$. C'est à dire :

$$\alpha_t(i) = P_\lambda(Y_1, Y_2, \dots, Y_t, X_t = x_i)$$

où $\alpha_t(i)$ est appelé la variable de Forward.

Soit $\beta_t(i)$: la variable de Backward définie comme étant la probabilité de la suite d'observation partielle de t + 1 à T, sachant qu'à l'instant t on était dans l'état $X_t = x_i$. C'est à dire :

$$\beta_t(i) = P_\lambda(Y_{t+1}, Y_{t+2}, \dots, Y_T / X_t = x_i)$$

On peut calculer $P_\lambda(Y)$ en utilisant ces variables.

* En utilisant uniquement la variable de Forward :

$$P_\lambda(Y_1,\dots, Y_T) = \sum_{i=1}^{N} P_\lambda(Y_1,\dots, Y_T, X_T = x_i)$$

$$= \sum_{i=1}^{N} \alpha_T(i)$$

* En combinant les deux variables (Forward et Backward) :

$$P_\lambda(Y_1,\dots, Y_T) = \sum_{i=1}^{N} P_\lambda(Y_1,\dots, Y_T, X_t = x_i)$$

$$= \sum_{i=1}^{N} P_\lambda(Y_{t+1},\dots, Y_T / X_t = x_i, Y_1,\dots, Y_t).P_\lambda(Y_1,\dots, Y_t, X_t = x_i)$$

$$= \sum_{i=1}^{N} P_\lambda(Y_{t+1},\dots, Y_T / X_t = x_i).P_\lambda(Y_1,\dots, Y_t, X_t = x_i)$$

$$:= \sum_{i=1}^{N} \alpha_t(i).\beta_t(i)$$

* Calcul de $\alpha_t(i)$ et $\beta_t(i)$:

Algorithme de Forward

Etape 1 : Initialisation

$$\alpha_1(i) = a_{0i}\, b_i(Y_1) \quad 1 \leq i \leq N$$

Etape 2 : Induction

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i).a_{ij}\right).b_j(Y_{t+1})$$

pour $1 \leq t \leq T-1$ et $1 \leq j \leq N$

Algorithme de Backward

Etape 1 : Initialisation

$$\beta_T(i) = 1 \quad 1 \leq i \leq N$$

Etape 2 : Induction

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}.\ b_j(Y_{t+1}).\ \beta_{t+1}(j)$$

$1 \leq t \leq T-1$, $1 \leq i \leq N$

3.2- RE-ESTIMATION DES PARAMETRES:

L'expression exacte de $P_\lambda(Y_1, Y_2, \ldots, Y_T)$ est malheureusement bien trop compliquée pour pouvoir envisager une résolution analytique du problème de maximisation de vraisemblance ([14]; [10]). Les solutions utilisées ne présentent que des optimisations locales telles que les techniques du gradient et les procédures de ré-estimation.

Connaissant un jeu de paramètres λ, on va chercher un nouveau jeu de paramètres λ', tel que $P_{\lambda'}(Y_1, Y_2, \ldots, Y_T) > P_{\lambda'}(Y_1, Y_2, \ldots, Y_T)$, à l'aide d'une astuce donnée par le théorème de **BAUM**.

i) THEOREME DE BAUM:

Soit la fonction auxiliaire :

$$Q(\lambda, \lambda') := \sum_{(X_1,\ldots,X_T)} P_\lambda(Y_1, Y_2,..,Y_T, X_1,., X_T).\mathrm{Log}(P_{\lambda'}(Y_1, Y_2,..,Y_T, X_1,.., X_T)))$$

Théorème de BAUM [4]

Si $Q(\lambda, \lambda') \geq Q(\lambda, \lambda)$, alors $P_{\lambda'}(Y_1, Y_2, \ldots, Y_T) \geq P_\lambda(Y_1, Y_2, \ldots, Y_T)$

On peut donc commencer par choisir λ^0 arbitrairement, puis maximiser $Q(\lambda^0, \lambda)$ par rapport à λ. Appelons le maximum obtenu λ^1. Evidemment $Q(\lambda^0, \lambda^1) \geq Q(\lambda^0, \lambda^0)$, d'où $P_\lambda{}^1(Y_1, Y_2, \ldots, Y_T) \geq P_\lambda{}^0(Y_1, Y_2, \ldots, Y_T)$. Dans la prochaine étape, on maximise $Q(\lambda^1, \lambda)$ par rapport à λ, et on appelle le maximum λ^2. De la même façon $Q(\lambda^1, \lambda^2) \geq Q(\lambda^1, \lambda^1)$, d'où $P_\lambda{}^2(Y_1, Y_2, \ldots, Y_T) \geq P_\lambda{}^1(Y_1, Y_2, \ldots, Y_T)$. En procédant de cette manière, nous obtenons une séquence $\lambda^0, \lambda^1, \ldots, \lambda^m$, telle que $P_\lambda{}^0(Y) \leq P_\lambda{}^1(Y) \leq \ldots P_\lambda{}^m(Y)$. Il s'agit donc d'un algorithme itératif où à chaque itération la valeur de la vraisemblance est améliorée.

ii) CALCUL DE Q(λ , λ'):

$$Q(\lambda,\lambda') = \sum_{(X_1,\ldots,X_T)} P_\lambda(X_1,\ldots,X_T,Y_1,\ldots,Y_T).\mathrm{Log}(P_{\lambda'}(X_1,\ldots,X_T,Y_1,\ldots,Y_T))$$

$$= \sum_{(X_1,\ldots,X_T)} P_\lambda(X_1,\ldots,X_T,Y_1,\ldots,Y_T).\mathrm{Log}(\prod_{t=1}^{T} a'_{X_{t-1},X_t}.b'_{X_t}(Y_t))$$

$$= \sum_{(X_1,\ldots,X_T)} P_\lambda(X_1,\ldots,X_T,Y_1,\ldots,Y_T)(\sum_{t=1}^{T}(\mathrm{Log}(a'_{X_{t-1}X_t}) + \mathrm{Log}(b'_{X_t}(Y_t))))$$

Nous cherchons à maximiser la fonction $Q(\lambda, \lambda')$ par rapport à λ', où

$$\lambda := \{a_{0i}, a_{ij}, b_j(k) : 1 \leq i, j \leq N \text{ et } 1 \leq k \leq M\}$$

et $$\lambda' := \{a'_{0i}, a'_{ij}, b'_j(k) : 1 \leq i, j \leq N \text{ et } 1 \leq k \leq M\}$$

Les paramètres de λ' sont des variables réelles dans l'intervalle [0 , 1], cependant elles ne sont pas indépendantes. Elles doivent satisfaire :

$$\sum_{i=1}^{N} a'_{0i} = 1; \quad \sum_{j=1}^{N} a'_{ij} = 1 ; \quad \sum_{k=1}^{M} b'_j(k) = 1$$

Le problème s'écrit alors :

$$\underset{\lambda'}{\text{Max}} \left(Q(\lambda, \lambda')\right)$$

$$C_0(\lambda') = \sum_{i=1}^{N} a'_{0i} = 1 \text{ avec } i:= 1,2,...,N$$

$$C_i(\lambda') = \sum_{j=1}^{N} a'_{ij} = 1 \text{ avec } 1 \le i,j \le N$$

$$C_{N+j}(\lambda') = \sum_{k=1}^{M} b'_j(k) = 1 \text{ avec } 1 \le k \le M \text{ et } 1 \le j \le N$$

On peut alors calculer analytiquement l'optimum sous contraintes de $Q(\lambda , \lambda')$ en écrivant la dérivée du lagrangien :

$$\nabla_{\lambda',\mu} \left\{ Q(\lambda',\lambda) - \sum_{i=0}^{2N} (\mu_i . C_i(\lambda') - 1) \right\} = 0$$

dans laquelle les $C_i(\lambda')$ représentent les contraintes sur les paramètres. La résolution de l'équation conduit aux formules de BAUM [5], ou à l'algorithme de BAUM-WELCH [5], [6].

.iii) Formules de Baum :

Les paramètres sont les probabilités de transition a'_{ij} , les probabilités initiales a'_{0i} et les probabilités d'émission $b'_j(k)$. Les formules de réestimations correspondantes sont :

$$a'_{0i} = \alpha_1(i).\ \beta_1(i) \,/\, \left(\sum_{i=1}^{N} \alpha_t(i).\ \beta_t(i) \right)$$

$$a'_{ij} = \left(\sum_{t=1}^{T-1} \alpha_t(i).a_{ij}.b_j(Y_{t+1}).\beta_{t+1}(j)\right) / \left(\sum_{t=1}^{T-1} \alpha_t(i).\beta_t(i)\right)$$

$$b'_j(k) = \left(\sum_{\substack{t=1 \\ Y_t = v_k}}^{T} \alpha_t(j).\beta_t(j)\right) / \left(\sum_{t=1}^{T} \alpha_t(j).\beta_t(j)\right)$$

iv) L'algorithme de BAUM-WELCH:

En résumant ce que nous venons de voir, l'algorithme de BAUM-WELCH [5]; [6] est l'algorithme itératif suivant :

Algorithme

Etape 1: Fixer les valeurs initiales

a^0_{0i} , a^0_{ij} , $b^0_j(k)$ pour $1 \le i,j \le N$ et $1 \le k \le M$

poser l:=0

Etape 2 : Calculer

$\alpha_t(i)$ et $\beta_t(i)$

pour $1 \le i, j \le N$, $1 \le t \le T$.

l:= l+1

Etape 3: Calculer les nouvelles estimations

a^l_{0i} , a^l_{ij} , $b^l_j(k)$ pour $1 \le i,j \le N$ et $1 \le k \le M$

en utilisant les formules de BAUM

Etape 4: Si le test d'arrêt n'est pas vérifié, aller à l'étape 2

Si non STOP

Remarques:

- Le test d'arrêt porte en général sur le nombre des itérations qui est fixé empiriquement

- Cette méthode utilisée pour l'estimation de λ est appelée **E .M** (Expectation Modification).

3.3- CONVERGENCE:

i) Propriétés de la convergence:

La convergence vers un maximum local est montrée dans [51] . Aucune garantie n'est donnée pour trouver un maximum global. La pratique montre qu'en général l'algorithme converge vers une solution acceptable . Cette convergence fait appel à la remarque suivante :

On dispose en pratique de plusieurs séquences d'observations $Y^k = (Y^k_1, ..., Y^k_T)$, k= 1,2,...., pour entraîner notre modèle . Le principe du Maximum de Vraisemblance consiste à chercher le maximum de:

$$\prod_k P_\lambda(Y^k_1, ..., Y^k_T) = \prod_k \sum_\Gamma P_\lambda(Y^k_1, ..., Y^k_T, X_1, ..., X_T)$$

Il faut alors développer ce polynôme, formuler la fonction auxiliaire ($Q(\lambda, \lambda')$), et annuler les dérivées de son lagrangien. Les ré-estimations correspondantes sont comparables aux formules de Baum, mais une sommation supplémentaire sur les exemples pondérés par les inverses de leurs vraisemblances, se glisse aux numérateurs et dénominateurs.

ii) Problèmes numériques (Le problème de L'underflow):

Cet algorithme pose également d'importants problèmes numériques. Les calculs des $\alpha_t(i)$ et $\beta_t(i)$ consistent essentiellement à multiplier entre elles des probabilités inférieures à 1. Toutes ces valeurs tendent donc vers 0 très vite. Les formules de BAUM deviennent alors des quotients de la forme 0 / 0. Une méthode pour s'affranchir de ces problèmes numériques est proposée par Levinson [31]. Soit la suite (N_t) telle que:

$$\alpha'_t(i) = C_t \cdot \alpha_t(i)$$
$$\beta'_t(i) = D_t \cdot \beta_t(i)$$

$$\text{avec } C_t = \prod_{k=1}^{t} N_t \quad \text{et} \quad D_t = \prod_{k=t+1}^{T} N_t$$

En reprenant les récurrences de Forward et Backward, on obtient :

$$\alpha'_{t+1}(j) = (C_{t+1} / C_t).\left(\sum_{j=1}^{N} \alpha_t(i).a_{ij}\right).b_j(Y_{t+1})$$

$$= N_{t+1}.\left(\sum_{j=1}^{N} \alpha_t(i).a_{ij}\right).b_j(Y_{t+1})$$

$$\beta'_t(i) := (D_t / D_{t+1}).\left(\sum_{j=1}^{N} \beta_{t+1}(j).a_{ij}.b_j(Y_{t+1})\right)$$

$$:= N_{t+1}.\left(\sum_{j=1}^{N} \beta_{t+1}(j).a_{ij}.b_j(Y_{t+1})\right)$$

Ces formules de récurrence permettent de calculer les $\alpha'_t(i)$ et les $\beta'_t(i)$. On détermine les N_t au fur et à mesure, de telle sorte que :

$$\sum_{i=1}^{N} \alpha'_t(i) = 1$$

De plus, on peut faire apparaître les coefficients C_t et D_t dans les formules de Baum, en multipliant le numérateur et le dénominateur par C_t . $D_t = C_T = D_1$, on obtient ainsi des formules équivalentes aux formules de Baum, dans lesquelles les $\alpha'_t(i)$ et les $\beta'_t(i)$ remplacent formellement les $\alpha_t(i)$ et les $\beta_t(i)$.

4-. DETERMINATION DU CHEMIN LE PLUS PROBABLE:

L'essentiel dans une application utilisant les HMM est la détermination de façon optimale , de la composante cachée X du processus, étant donnée la composante Y, et les informations probabilistes sur le modèle (N, M et λ). La difficulté de ce problème est la définition du critère d'optimisation.

Le critère le plus naturel et le plus utilisé [26] est celui de trouver la meilleure suite d'état « chemin » qui maximise P(X / Y) i.e. la probabilité de X à posteriori, ce qui est équivalent à maximiser :

$$P_\lambda(X , Y)$$

La solution pour ce dernier critère est souvent appelée la suite d'états de Viterbi [24].

4.1- L'ALGORITHME DE VITERBI [24]:

Pour trouver le meilleur chemin, $X = (X_1, X_2, ..., X_T)$, pour une suite d'observations donnée $Y = (Y_1, Y_2, ..., Y_T)$, nous définissons la quantité $\delta_t(i)$, telle que:

$$\delta_t(i) = \max_{(X_1,...X_t)} P_\lambda(X_1,...,X_t = x_i, Y_1,...,Y_t,)$$

$\delta_t(i)$: la probabilité du meilleur chemin partiel amenant à l'état $X_t = x_t$ guidé par les t premières observations.

Par induction nous avons :

$$\delta_{t+1}(j) = (\max_i \delta_t(i).a_{ij}).bj(Y_{t+1})$$

Algorithme

Etape 1 : Initialisation

$$\delta_1(i) = a_{0i}\, b_i\,(Y_i) \quad i = 1, \ldots, N$$

$$\Phi_1(i) = 0$$

Etape 2 : Induction

$$\delta_{t+1}(j) = (\max_i \delta_t(i).a_{ij}).bj(Y_{t+1}) \quad 1 \le i,j \le N$$

$$\Phi_{t+1}(j) = \arg(\max_i \delta_t(i).a_{ij})$$

Etape 3 : Terminaison

$$P^* = (\max_i \delta_T(i).).$$

$$X^*_T = \arg(\max_I \delta_T(i).).$$

Etape 4 : Chemin d'état retenu (Backtracking)

$$X^*_t = \Phi_{t+1}(X^*_{t+1}) \quad 1 \le t \le T\text{-}1$$

La fonction arg permet de mémoriser l'indice i, entre 1 et N, avec lequel on atteint le maximum des quantités $(\delta_{t-1}(i) \cdot a_{ij})$

4.2 APPRENTISSAGE PAR L'ALGORITHME DE VITERBI :

Le critère de Viterbi peut être considéré comme une version simplifiée du critère du maximum de vraisemblance [10]. Au lieu de prendre en compte toutes les séquences d'états possibles dans le modèle, on considère seulement le chemin le plus probable.

$$P_\lambda(Y_1, Y_2, \ldots, Y_T) := \sum_{(X_1,\ldots, X_T) \in \Gamma} P_\lambda(Y_1,\ldots, Y_T, X_1,\ldots, X_T)$$

Une formulation explicite du critère de Viterbi est obtenu en remplaçant tous les opérateurs somme par un opérateur « max ».

$$P_\lambda(Y_1, Y_2, \ldots, Y_T) := \max_{(X_1,\ldots, X_T) \in \Gamma} P_\lambda(Y_1,\ldots, Y_T, X_1,\ldots, X_T)$$

Les formules de ré-estimation sont identiques à celles développées par l'algorithme de Baum-Welch. sauf, pour les fonctions α et β, les sommes (Σ) sont remplacées par des maximums (max)

LES MODELES DE MARKOV DISCRIMINANTS ET LES MODELES CONNEXIONNISTES

Dans ce chapitre nous allons voir dans un premier temps l'utilisation des HMM standards pour la reconnaissance automatique de la parole (R.A.P). Après quoi, nous allons étudier les HMM discriminants dans les termes de (R.A.P), ensuite nous exposons certains modèles connexionnistes trouvés dans ([10],[12],[13]) pour l'évaluation des probabilités de transition d'un HMM.

1- APPLICATIONS DES MODELES DE MARKOV CACHES POUR LA RECONNAISSANCE AUTOMATIQUE DE LA PAROLE:

1.1- RECONNAISSANCE AUTOMATIQUE DE LA PAROLE:

De façon générale, la reconnaissance automatique de la parole consiste à transcrire l'onde acoustique en symbole [26]. La grande complexité de la R.A.P. provient essentiellement de certaines caractéristiques spécifiques du signal vocal. La parole est caractérisée par une grande variabilité à la fois interlocuteur et interalocuteur. En effet, le conduit vocal présente des différences physiologiques d'un locuteur à un autre. Par ailleurs, les limitations d'ordres mécaniques de l'appareil phonatoire conduisent à une forte dépendance des sons émis.

Une autre caractéristique du signal de la parole est l'absence de marques de segmentation entre les mots d'un même énoncé. De ce fait, la localisation des mots dans le même signal énoncé vocal est une opération difficile nécessitant le plus souvent le recours à des connaissances linguistiques.

Tout ceci explique que l'on ne sache reconnaître la parole que dans des situations biens précises. Les systèmes de reconnaissance de la parole commettent rarement plus de 5 % d'erreurs [10]. Ils ne se distinguent que par les contraintes qu'ils requièrent pour atteindre ce taux de performance.

On peut résumer les contraintes les plus courantes par les oppositions suivantes :

- Est-il mono-locuteur, multi-locuteur, ou indépendant du locuteur ? Cela signifie respectivement qu'il est capable de ne reconnaître la parole que prononcé par un unique locuteur, par un ensemble bien précis de locuteur, ou bien tout locuteur convient.

- Reconnaît des mots isolés ou de la parole continue ? Restreint-il les phrases reconnues à une grammaire contraignante ou non ?
- Quelle est la taille du vocabulaire utilisé ?
- Quelle est la qualité de la parole qu'il utilise ?

i) Chaîne de traitement :

Un système de reconnaissance de la parole est habituellement constitué de trois composantes :

- Un dispositif d'acquisition du signal, usuellement un microphone ou une bande magnétique, associés à un système d'amplification et de filtrage. Le signal est ensuite numérisé afin d'être traité.

- En pratique ensuite des prétraitements, dont le but est d'une part de réduire le flot de données dans le système à un niveau acceptable, d'autre part de présenter les données sous un format Y compatible avec l'étape de reconnaissance.

- La troisième étape consiste à extraire de ces données transformées une séquence X de symboles (des mots ou des phonèmes par exemple). C'est le point le plus délicat des systèmes de reconnaissance.

On peut résumer les contraintes les plus courantes par les oppositions suivantes :

- Est-il mono-locuteur, multi-locuteur, ou indépendant du locuteur ? Cela signifie respectivement qu'il est capable de ne reconnaître la parole que prononcé par un unique locuteur, par un ensemble bien précis de locuteur, ou bien tout locuteur convient.

- Reconnaît des mots isolés ou de la parole continue ? Restreint-il les phrases reconnues à une grammaire contraignante ou non ?
- Quelle est la taille du vocabulaire utilisé ?
- Quelle est la qualité de la parole qu'il utilise ?

i) Chaîne de traitement :

Un système de reconnaissance de la parole est habituellement constitué de trois composantes :

- Un dispositif d'acquisition du signal, usuellement un microphone ou une bande magnétique, associés à un système d'amplification et de filtrage. Le signal est ensuite numérisé afin d'être traité.

- En pratique ensuite des prétraitements, dont le but est d'une part de réduire le flot de données dans le système à un niveau acceptable, d'autre part de présenter les données sous un format Y compatible avec l'étape de reconnaissance.

- La troisième étape consiste à extraire de ces données transformées une séquence X de symboles (des mots ou des phonèmes par exemple). C'est le point le plus délicat des systèmes de reconnaissance.

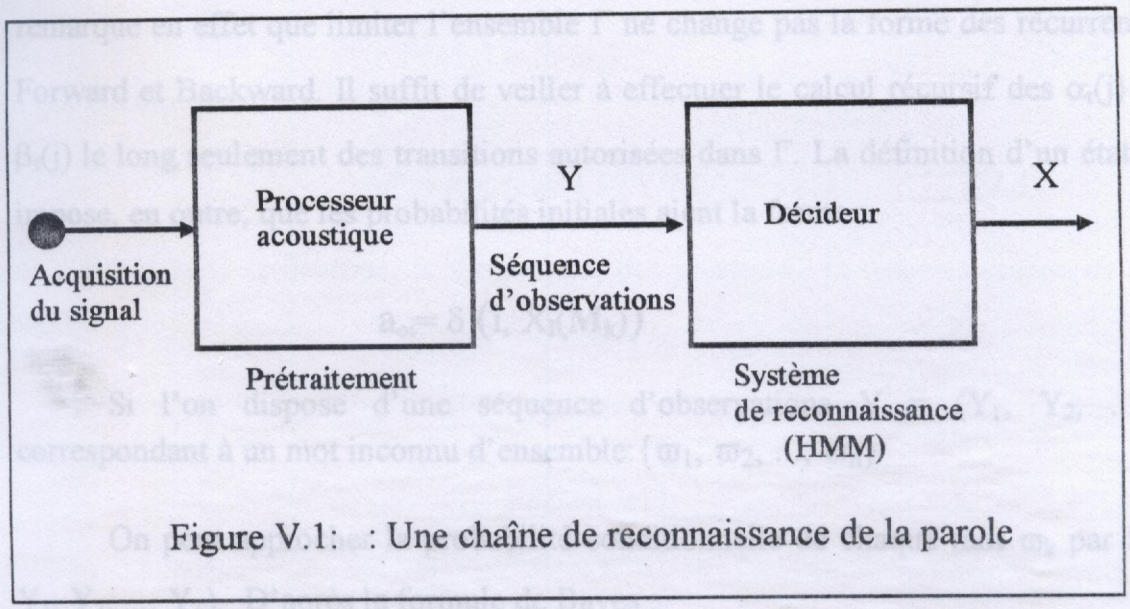


Figure V.1: : Une chaîne de reconnaissance de la parole

Les séquences de symboles (d'états) ainsi générées sont ensuite exploitées par exemple par un programme d'analyse sémantique, dans le cas de commande vocale, ou encore par un programme de transcription orthographique dans le cas d'une machine à écrire vocale. Ces programmes ne seront pas abordés ici. Il faut savoir tout de même qu'ils représentent un problème en général extrêmement difficile.

1.2 LES MODELES DE MARKOV CACHES POUR LA RECONNAISSANCE DE LA PAROLE:

La reconnaissance stochastique de la parole est basée sur la comparaison d'une émission reconnaissable Y avec des HMM à états probabilistes finis. Ceci est illustré par la figure VI.2

Considérons par exemple le problème de reconnaissance de mots isolés :

$$\{\varpi_1, \varpi_2, ..., \varpi_n\}$$

et soient

$$\{M_1, M_2, ..., M_n\}$$

des modèles de Markov cachés censés représenter chacun des mots. Dans chacun de ces modèles. On souhaite définir un état initial $X_I(M_k)$ et un état final $X_F(M_k)$. On remarque en effet que limiter l'ensemble Γ ne change pas la forme des récurrences de Forward et Backward. Il suffit de veiller à effectuer le calcul récursif des $\alpha_t(j)$ et des $\beta_t(j)$ le long seulement des transitions autorisées dans Γ. La définition d'un état initial impose, en outre, que les probabilités initiales aient la forme :

$$a_{oi} = \delta\,(i, X_I(M_k))$$

Si l'on dispose d'une séquence d'observations $Y = (Y_1, Y_2, ..., Y_T)$ correspondant à un mot inconnu d'ensemble: $\{\varpi_1, \varpi_2, ..., \varpi_n\}$

On peut approcher la probabilité conditionnelle de chaque mot ϖ_k par $P(M_k / Y_1, Y_2, ..., Y_T)$. D'après la formule de Bayes :

$$P(\varpi_k / Y_1,...,Y_T) \approx P(M_k / Y_1,...,Y_T) = \\ = P(Y_1,...,Y_T / M_k)\, P(M_k) / P(Y_1,...,Y_T) \qquad (V.1)$$

Pour une séquence d'observations fixée, le dénominateur de cette expression est constant. Le mot reconnu est donc :

$$\varpi^* = \arg\max_k [\, P(Y/M_k).P(M_k)] \qquad (V.2)$$

i) Calcul de $P(Y / M_k)$:

Soit λ^k l'ensemble des paramètres du modèle M_k . En appliquant par exemple l'algorithme de Forward, et en tenant compte des contraintes sur les chemins admissibles, on obtient :

$$P(Y_1, Y_2, ..., Y_T / w_k) = \alpha_T(X_F(M_k)) \qquad (V.3)$$

avec $\alpha_1(X_i(M_k)) = b_{X_i(M_k)}(Y_1)$

et $\alpha_1(j) = 0 \quad \forall\, X_j(M_k) \neq X_I(M_k)$

ii) Estimation de $P(M_k)$:

On évalue cette probabilité en se donnant un modèle langage, c'est à dire en déterminant empiriquement la probabilité d'occurrence de mots.

$$P(\mathbf{M}_k) \approx P(\varpi_k) \qquad (V.4)$$

ce qui donne :

$$\varpi^* = \arg\max_k [\, P(Y/\mathbf{M}_k).P(\varpi_k)] \qquad (V.5)$$

En résumé ci-dessous, le diagramme des phases de la reconnaissance de mots isolés.

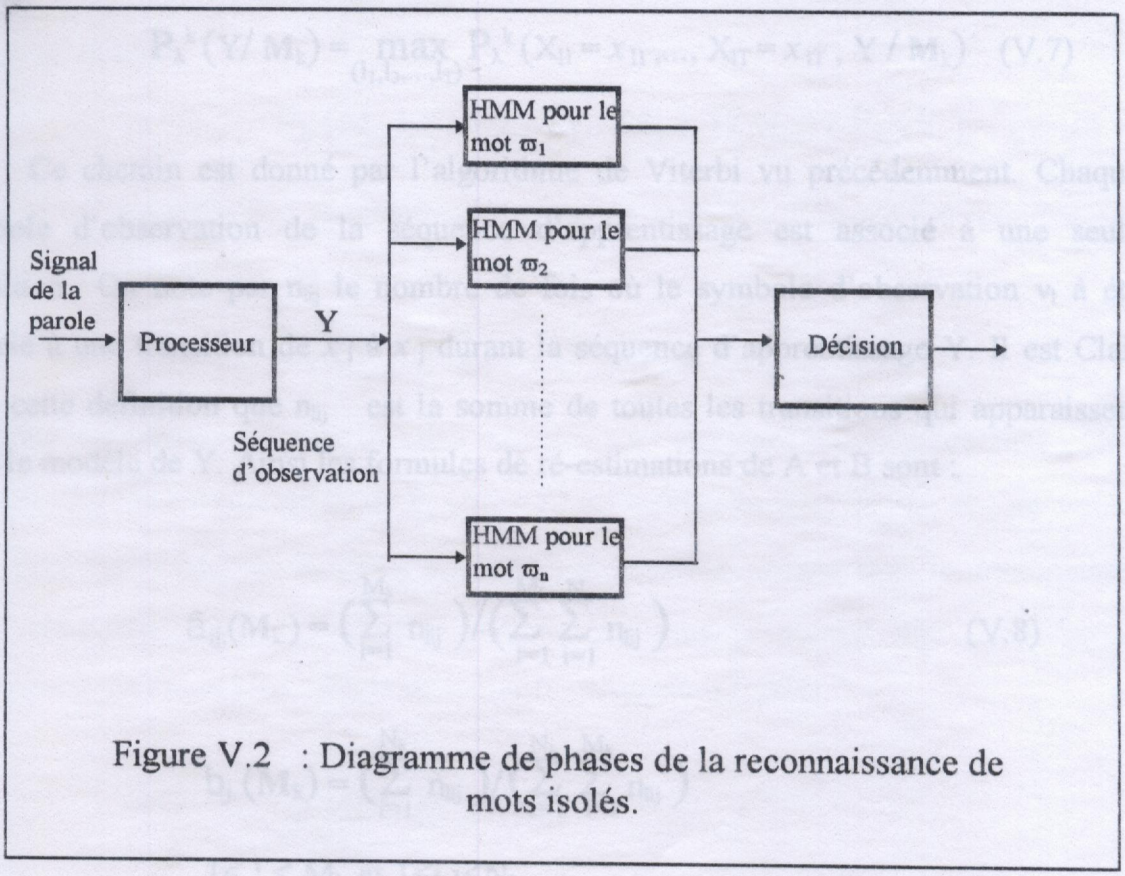


Figure V.2 : Diagramme de phases de la reconnaissance de mots isolés.

iii) Apprentissage « Estimation des paramètres des modèles »

L'apprentissage de chaque modèle $\mathbf{M}_k$ se fait indépendamment des autres. Comme nous allons le voir dans le paragraphe suivant, ceci conduit au critère du

maximum de vraisemblance, c'est à dire à la maximisation de $P_\lambda^k (Y/ M_k)$, où λ^k représente les paramètres du modèle M_k. $P_\lambda^k (Y/ M_k)$ peut être ré-écrite comme suit :

$$P_\lambda^k (Y/ M_k) = \sum_{l_1=1}^{N_k} \ldots \sum_{l_T=1}^{N_k} P_\lambda^k (X_{l1} = x_{l1}, \ldots, X_{lT} = x_{LT}, Y / M_k) \qquad (V.6)$$

où N_k représente le nombre d'états dans le modèle M_k . Comme signalé auparavant, le critère de Viterbi peut être considéré comme une version simplifiée du critère du maximum de vraisemblance. Au lieu de prendre en compte toutes les séquences d'états possibles dans M_k capables de produire Y, on considère le chemin le plus probable :

$$P_\lambda^k (Y/ M_k) = \max_{(l_1, l_2, \ldots, l_T)} P_\lambda^k (X_{l1} = x_{l1}, \ldots, X_{lT} = x_{lT}, Y / M_k) \quad (V.7)$$

Ce chemin est donné par l'algorithme de Viterbi vu précédemment. Chaque symbole d'observation de la séquence d'apprentissage est associé à une seule transition.. On note par n_{lij} le nombre de fois où le symbole d'observation v_l à été associé à une transition de x_i à x_j durant la séquence d'apprentissage Y. Il est Clair avec cette définition que n_{lij} est la somme de toutes les transitions qui apparaissent dans le modèle de Y. Ainsi les formules de ré-estimations de A et B sont :

$$\hat{a}_{ij;}(M_k) = \left(\sum_{l=1}^{M_k} n_{lij}\right) / \left(\sum_{l=1}^{M_k} \sum_{i=1}^{N_k} n_{lij}\right) \qquad (V.8)$$

$$b_j (M_k) = \left(\sum_{i=1}^{N_k} n_{lij}\right) / \left(\sum_{i=1}^{N_k} \sum_{l=1}^{M_k} n_{lij}\right)$$

$1 \le l \le M_k$ et $1 \le i, j \le N_k$

Remarque :

- Les probabilités initiales des états sont supposées à priori connues.

- Les estimations des probabilités $P(X_{t+1} = x_j, Y_{t+1} = v_l / X_t = x_i, M_k)$ effectivement utilisées dans les HMM standards sont données par :

$$P(x_j, v_l / x_i, \mathbf{M_k}) = n_{lij} / \left(\sum_{l=1}^{M_k} \sum_{j=1}^{N_k} n_{lij} \right) \quad (V.9)$$

iv) Capacité discriminante :

Dans la phase d'apprentissage de nos modèles, la probabilité $P_\lambda (\mathbf{M_k} / Y)$ doit être maximisée :

$$P_\lambda (\mathbf{M_k} / Y) = P_\lambda (Y / \mathbf{M_k}). P(\mathbf{M_k}) / P_\lambda (Y). \quad (V.10)$$

où λ est l'ensemble des paramètres de tous les modèles. Puisque les modèles sont mutuellement exclusifs, on a :

$$P_\lambda (Y) = \sum_{i=1}^{n} P_\lambda (Y / \mathbf{M_i}). P(\mathbf{M_i}) \quad (V.11)$$

Puisque nous avons supposé que nos modèles sont entraînés indépendamment l'un de l'autre, alors la maximisation de $P_\lambda (\mathbf{M_k} / Y)$ est limitée au sous espace des paramètres λ^k ($\max P_\lambda(Y/ \mathbf{M_k}) \Leftrightarrow P_\lambda^k(Y)$) car $P(\mathbf{M_k})$ et le terme somme dans le dénominateur sont constants. Cette optimisation modèle par modèle permet une simplification importante en évitant les calculs de toutes les séquences rivales :

$$\sum_{i=1, i\neq k}^{n} P_\lambda(Y/ \mathbf{M_i}).P(\mathbf{M_i})$$

mais au coût d'un pouvoir discriminant faible. Pour cette raison (la discrimination), on introduit les modèles de Markov discriminants permettant d'estimer les probabilités à posteriori de mots, à l'aide du principe du maximum de probabilité à posteriori [10].

2- LES MODELES DE MARKOV CACHES DISCRIMINANTS:

Dans le cas de l'exemple de la reconnaissance de la parole, il est intuitivement plus aisé d'apprendre ce qu'est un mot, en le comparant à d'autres, et en relevant des

caractéristiques discriminantes, c'est à dire en identifiant des traits permettant de distinguer deux mots. C'est à dire maximiser P_λ ($\mathbf{M}_k$ / Y) par rapport à l'espace des paramètres intacts (les paramètres de tous les modèles ($\mathbf{M}_1$, $\mathbf{M}_2$, ..., $\mathbf{M}_n$). Ceci rend la contribution de P_λ ($\mathbf{M}_k$/ Y) . P($\mathbf{M}_k$) dans la formule (V.11) plus importante que celle des modèles rivaux qui doit être réduite.

2.1- EVALUATION DES PROBABILITES A POSTERIORI DANS UN MODELE DE MARKOV CACHE:

De façon analogue au calcul de la vraisemblance d'une observation, on calcule la probabilité à posteriori :

$$P_\lambda(\mathbf{M}_k / Y_1,..., Y_T) = \sum_{j=1}^{N} P_\lambda(\mathbf{M}_k, X_t := x_j / Y_1,... , Y_T) \quad (V.12)$$

avec N_k représente le nombre d'états dans le modèle $\mathbf{M}_k$. On peut décomposer cette expression de la façon suivante, en utilisant les hypothèses sur les modèles de Markov cachés (§.IV.2.ii):

$$P_\lambda(\mathbf{M}_k, X_t := x_j / Y_1,... , Y_T) = \frac{P_\lambda(\mathbf{M}_k, X_t := x_j, Y_{t+1},..., Y_T / Y_1,... , Y_t)}{P_\lambda(Y_{t+1},..., Y_T / Y_1,... , Y_T)}$$

$$= P_\lambda(\mathbf{M}_k, X_t := x_j / Y_1,... , Y_T). \frac{P_\lambda(Y_{t+1},..., Y_T / \mathbf{M}_k, X_t := x_j)}{P_\lambda(Y_{t+1},..., Y_T / Y_1,... , Y_t)} =$$

$$= \alpha_t^d(j).\beta_t^d(j) \quad (V.13)$$

Les fonctions $\alpha_t^d(j)$ et $\beta_t^d(j)$ sont comparables aux variables de Forward et Backward respectivement, à un coefficient de normalisation près.

$$\alpha^d_{t+1}(j) = N_{t+1}.\sum_{i=1}^{N_k} \alpha_t^d(j).b_j(Y_{t+1}).a_{ij}$$

$$\beta^d_t(i) = N_{t+1}.\sum_{j=1}^{N_k} \beta_{t+1}^d(j).b_j(Y_{t+1}).a_{ij} \quad (V.14)$$

On reconnaît les formules $\alpha'_{t+1}(j)$ et $\beta'_t(j)$ proposées pour résoudre le problème de l'underflow (chapitre IV.3.3.ii)

$$\alpha'_t(j) = \alpha^d_t(j)$$
$$\beta'_t(j) = \beta^d_t(j)$$

On a également vu que l'introduction du coefficient de normalisation N_t ne modifie en rien les formules de BAUM lorsque l'on remplace $\alpha_t(j)$ et $\beta_t(j)$ par les grandeurs $\alpha'_t(j)$ et $\beta'_t(j)$.

La maximisation de $P_\lambda(Y_1, Y_2, ..., Y_T / \mathbf{M}_k)$ ou de $P_\lambda(\mathbf{M}_k / Y_1, Y_2, ..., Y_T)$ dans laquelle $(Y_1, Y_2, ..., Y_T)$ est un exemple de mots ϖ_k (k = 1, 2, ..., n), conduit donc aux mêmes formules de ré-estimation des probabilités de transition et d'émission.

En comparant avec les probabilités de transition et d'émission d'un HMM standard, la différence reste dans la normalisation des contributions locales. C'est à dire :

$$\sum_{k=1}^{n} P_\lambda(\mathbf{M}_k / Y_1, Y_2, ..., Y_T) = 1 \qquad (V.15)$$

C'est dans cette contrainte que s'exprime la différence de capacité discriminante entre l'évaluation des vraisemblances $P_\lambda(Y_1, Y_2, ..., Y_T / \mathbf{M}_k)$, et l'évaluation des probabilités à posteriori $P_\lambda(\mathbf{M}_k / Y_1, Y_2, ..., Y_T)$

i- Probabilités conditionnelles de transition:

Pour pouvoir appliquer sainement le principe du maximum de probabilité à posteriori, il faut donc trouver une expression paramétrique $P_\lambda(\mathbf{M}_k / Y_1, Y_2, ..., Y_T)$ pour laquelle la contrainte (V.15) est remplie pour toutes les valeurs du paramètre λ.

Une solution [10] consiste à décomposer les probabilités à posteriori comme suit :

$$P_\lambda(M_k / Y_1, Y_2, \ldots, Y_T) = \sum_{(X_1 .. X_t) \in \Gamma_k} P_\lambda(X_1, X_2, \ldots, X_T / Y_1, Y_2, \ldots, Y_T) \quad (V.16)$$

où Γ_k représente l'ensemble des séquences d'état autorisées pour le modèle $\mathbf{M}_k$. Pour écrire cette égalité, on a supposé que la connaissance d'états permet de déterminer le modèle qui l'a produite. Dans cette optique, un modèle est déterminé par l'ensemble des séquences d'état qu'il autorise. Chaque terme de cette somme peut être factorisé de la façon suivante:

$$P_\lambda(X_1, X_2, \ldots, X_T / Y_1, Y_2, \ldots, Y_T) = P_\lambda(X_1 / Y_1, Y_2, \ldots, Y_T) \cdot$$

$$\cdot P_\lambda(X_1 / Y_1, Y_2, \ldots, Y_T, X_1) \ldots P_\lambda(X_T / Y_1, Y_2, \ldots, Y_T, X_1, X_2, \ldots, X_{T-1})$$

Cette expression peut être simplifiée si l'on suppose que l'état courant ne dépend que de l'état immédiatement ultérieur et de l'observation courante, c'est à dire une hypothèse de type markovienne :

$$P_\lambda(X_t / Y_1, Y_2, \ldots, Y_T, X_1, X_2, \ldots, X_{t-1}) = P_\lambda(X_t / X_{t-1}, Y_t) \quad (V.17)$$

et donc

$$P_\lambda(\mathbf{M}_k / Y_1, Y_2, \ldots, Y_T) = \sum_{\Gamma_k} \left(\prod_{t=1} P_\lambda(X_t / X_{t-1}, Y_t) \right) \quad (V.18)$$

On peut donc utiliser un modèle paramètré $P_\lambda(X_t / X_{t-1}, Y_t)$ des probabilités conditionnelles de transition $P(X_t / X_{t-1}, Y_t)$ pour construire à l'aide de (V.18) un modèle paramètré $P_\lambda(\mathbf{M}_k / Y_1, Y_2, \ldots, Y_T)$ des probabilités à posteriori. Ces probabilités $P_\lambda(X_t / X_{t-1}, Y_t)$ sont équivalentes au probabilités Bayesiennes.

Si la condition de normalisation sur les probabilités conditionnelles de transition:

$$\sum_{j=1}^{N} P_\lambda(X_t := x_j / X_{t-1}, Y_t) = 1 \quad (V.19)$$

est remplie, (V.15) nous permet d'écrire en notant $\Gamma = \cup \Gamma_k$:

$$\sum_{k=1}^{n} P_\lambda(M_k / Y_1,..., Y_T) = \sum_{\Gamma} \left(\prod_{t=1} P_\lambda(X_t / X_{t-1}, Y_t)\right) :=$$

$$= \sum_{k_1=1}^{N} P_\lambda(X_1 := x_{k1} / Y_1) \left(\sum_{k_2=1}^{N} P_\lambda(X_2 := x_{k2} / X_1 := x_{k1}, Y_2) \bullet\bullet\bullet\right.$$

$$\bullet\bullet\bullet \left(\sum_{k_{T-1}=1}^{N} P_\lambda(X_{T-1} := x_{kT-1} / X_{T-2} := x_{kT-2}, Y_{T-1}) \bullet\right.$$

$$\left.\left.\bullet \left(\sum_{k_T=1}^{N} P_\lambda(X_T := x_{kT} / X_{T-1} := x_{kT-1}, Y_T)\right) \bullet\bullet\bullet\right)\right) = 1 \quad \text{(V.20)}$$

la contrainte (V.18) est donc automatiquement satisfaite.

3- RESEAUX DE NEURONES ET HMM DISCRIMINANT:

En modifiant la nature de l'hypothèse (V.17), il est possible d'utiliser des probabilités conditionnelles de formes très différentes :

i) $P(X_t / X_1,..., X_{t-1}, Y_1,..., Y_T) = P(X_t / X_{t-1}, Y_t)$

C'est l'hypothèse markovienne la plus simple, qui a été utilisée ci dessus pour présenter la décomposition des probabilités à posteriori à l'aide de probabilités conditionnelles de transition.

ii) $P(X_t / X_1,..., X_{t-1}, Y_1,..., Y_T) = P(X_t / X_{t-q},..., X_t, Y_t)$

La dépendance avec l'état précédent est étendue ici aux q états les plus récents :C'est un modèle de Markov d'ordre q.

iii) $P(X_t / X_1,..., X_{t-1}, Y_1,..., Y_T) = P(X_t / Y_{t-P},..., Y_t)$

On remplace ici la dépendance avec l'état précédent avec un certain contexte P sur les observations. Ce n'est plus, strictement, un modèle de Markov caché, mais le même formalisme s'applique.

iv) $P(X_t / X_1, \ldots, X_{t-1}, Y_1, \ldots, Y_T) = P^t{}_{Xt} = P(X_t / P^{t-1}{}_{Xt}, Y_t)$

On suppose cette fois ci que les probabilités conditionnelles de transition ne dépendent que de l'observation courante, et de toutes les probabilités conditionnelles à l'instant précédant.

Boulard et Wellekens [12] ont suggéré d'utiliser des modèles connexionnistes pour l'évaluation de ces probabilités. Voici quelques possibilités.

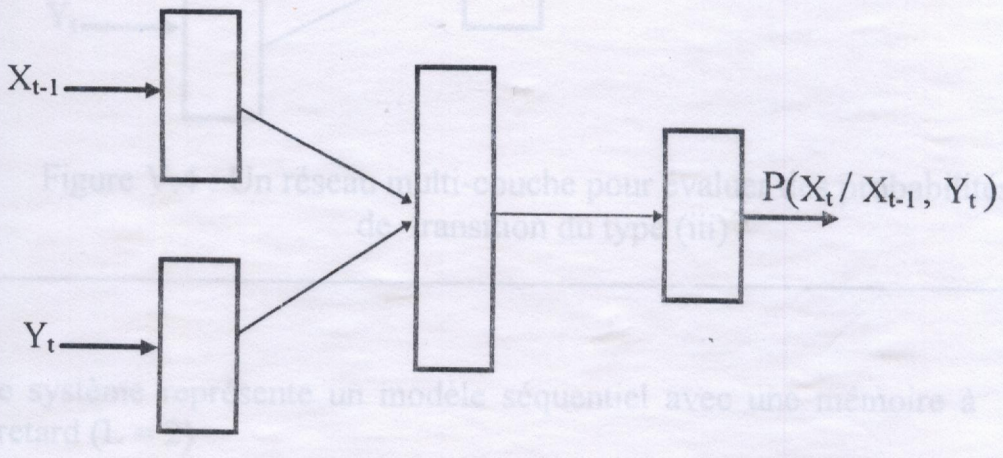


Figure V.3 : un réseau multi-couches pour évaluer des probabilités de types (i)

Avec l'hypothèse i, on obtient le système étudié par Boulard et Wellekens dans l'article cité. Nous savons que l'algorithme de rétro-propagation du gradient maximise un critère des moindres carrés. Il évalue donc les probabilités conditionnelles de chaque classe sachant ses entrées (voir chapitre I), ici l'état antérieur et l'observation courante.

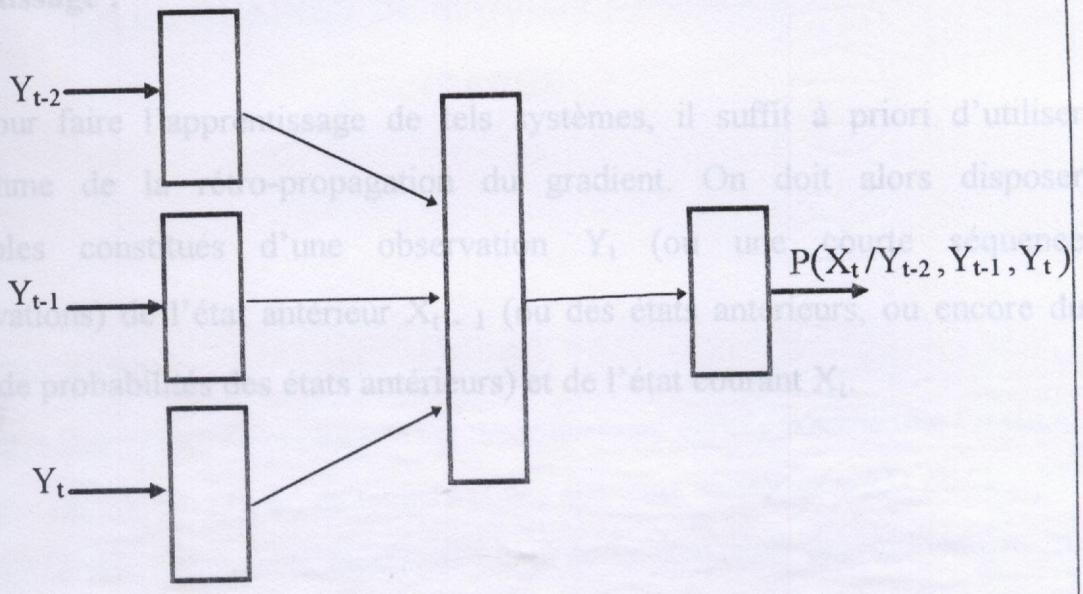


Figure V.4 : Un réseau multi-couche pour évaluer des probabilités de transition du type (iii)

Ce système représente un modèle séquentiel avec une mémoire à court terme à retard (L = 2)

3 - Avec l'hypothèse (iv), on obtient un cas particulier de la machine séquentielle connexionniste de Touzet (voir chapitre II §.5), c'est à dire la fonction de sortie est nulle.

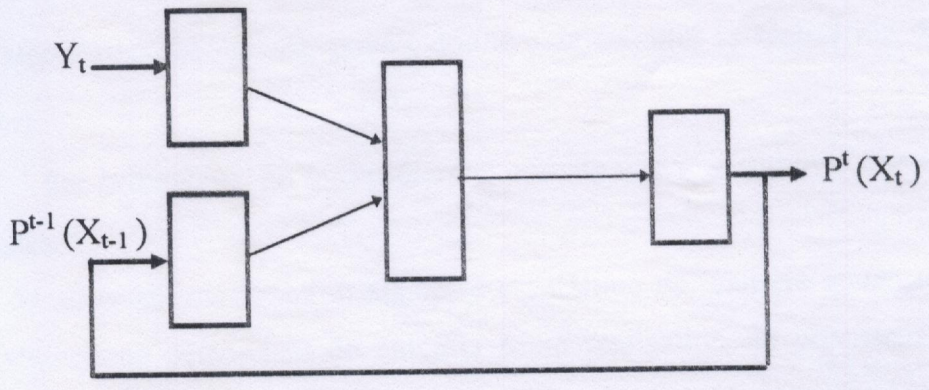


Figure V.5 : Le cas particulier de la machine séquentielle de Touzet pour évaluer les probabilités du type iv.

Apprentissage :

Pour faire l'apprentissage de tels systèmes, il suffit à priori d'utiliser l'algorithme de la rétro-propagation du gradient. On doit alors disposer d'exemples constitués d'une observation Y_t (ou une courte séquence d'observations) de l'état antérieur X_{t-1} (ou des états antérieurs, ou encore du vecteur de probabilités des états antérieurs) et de l'état courant X_t.

SIMULATEURS CONNEXIONNISTES POUR LES CHAINES DE MARKOV CACHEES STANDARDS

1- INTRODUCTION:

Une chaîne Markovienne cachée est théoriquement définie comme une fonction aléatoire $Z_t = (X_t , Y_t) := f(t, \omega)$ du temps t et des éléments aléatoires $\omega \in \Omega$ où $\Omega = \{\omega\}$ est l'ensemble des événements aléatoires muni de la σ-algèbre des événements et d'une mesure de probabilité $P: \sigma \rightarrow [0,1]$.

Soit $E = \{x_1, \ldots, x_N\}$ ensemble des états de la chaîne de Markov X_t, et $V = \{v_1, \ldots, v_M\}$ l'ensemble des observations Y_t. Sur ExE est définit une matrice de probabilités de transition $A := (a_{ij})_{NxN}$ avec $a_{ij} = P(X_{t+1} := x_j \ / \ X_t := x_i)$ et un vecteur de distribution initiale $A_0 := (a_{0i})_{1xN}$, où $a_{0i} = P(X_0 := x_i)$, de même sur VxE est définit la matrice des probabilités des symboles d'observations $B = (b_j(k))_{MxN}$ avec $b_j(k) = P(Y_t := v_k \ / \ X_t := x_j)$.

Cette définition s'interprète bien en terme de simulation statistique [2] en utilisant une définition constructive de la fonction aléatoire $f(t,\omega)$. En effet, cette dernière comporte deux éléments fondamentaux:

- La source des aléas qui donne la réalisation de l'élément aléatoire $\omega \in \Omega$.
- L'algorithme permettant, pour ι donné , de calculer la réalisation du processus $Z_t = f(t, \omega)$.

Sur ordinateur, on interprétera ω comme une réalisation d'une suite de nombres au hasard.

On dit alors qu'une fonction aléatoire est définie de manière constructive, si il existe un algorithme permettant de calculer toute réalisation du processus $Z_t = (X_t, Y_t)$ à partir d'une suite $\{\eta_t\}_t = \{\chi_t, \gamma_t\}_t$ de variable aléatoires indépendantes, de loi uniforme sur [0,1]: $Z_t = f(t, \omega) = f(t, \{\eta_t\})$.

L'algorithme de simulation considère x_j à l'instant t+1 comme une réalisation d'une variable aléatoire χ_{t+1} de loi uniforme sur [0,1] ($\chi_{t+1} \rightarrow U_{[0,1]}$) et de l'état x_i à l'instant t, et v_k à t comme une réalisation d'une variable aléatoire γ_t ($\gamma_t \rightarrow U_{[0,1]}$) et de l'état x_j à l'instant t. Cet algorithme de simulation de la chaîne de Markov cachée $Z_t = (X_t, Y_t)$ comporte deux taches :

- La première concerne la génération de X_t (une chaîne de Markov homogène) qui représente deux aspects: le premier est aléatoire (détermination de la réalisation de χ_t), et le second est temporel (ou séquentiel du à la dépendance directe des états à deux instants successifs). Comme nous allons le voir le modèle neuronal exécutant cette tache doit avoir un comportement donné par:

$$\mathbf{O}(t+1) = F(\mathbf{I}(t+1), \mathbf{O}(t))$$

- La deuxième concerne la génération de Y_t qui en plus de son aspect aléatoire, est combinatoire.

Dans ce chapitre , nous commençons par proposer une première approche connexionniste composée de deux blocs en série pour la simulation d'un HMM. Dans ce réseau neuronal, l'aspect aléatoire n'est pas pris en compte. Ce réseau est une modélisation de l'algorithme de génération des observations dans un HMM (chapitre IV. §2.3). La seconde approche que nous présentons ensuite est faite sur la base de la méthode de simulation statistique en utilisant une définition constructive d'un HMM. Dans cette partie en trouve également les expérimentations réalisées.

2- MODELISATION CONNEXIONNISTE DE L'ALGORITHME DE GENERATION D'OBSERVATIONS D'UN HMM:

Le simulateur connexionniste (Figure VI.1) fait apparaître deux réseaux (blocs):

- L'un réalise la génération de X_t (i.e. la chaîne de Markov)

- L'autre réalise la génération de Y_t (i.e. les observations)

Le bloc 1 reçoit les entrées primaires **I**(t) (qui représentent a_{ij} ou a_{oi}), ainsi que des boucles de rétroactions **O**(t-1) (qui représente l'état présent du HMM) , et fournit en

sortie **O**(t) (l'état futur). Le bloc 2 reçoit les entrées $\mathbf{I}^*(t)$ (qui représentent les $b_j(k)$) et les sorties du bloc 1 (**O**(t)), et fourni en sortie $\mathbf{O}^*(t)$ (qui représentent le symbole d'observation du HMM).

Sur la figure VI.1, nous considérons que tous les neurones sont identiques et que les connexions se répartissent en deux classes, selon la nature fixe ou plastique. les connexions représentées en trait gras sont de poids fixe. Leurs poids ne sont pas modifiés lors de l'apprentissage. Les connexions entre les différentes couches dans chaque bloc sont plastiques, leurs poids sont modifiés durant l'apprentissage.

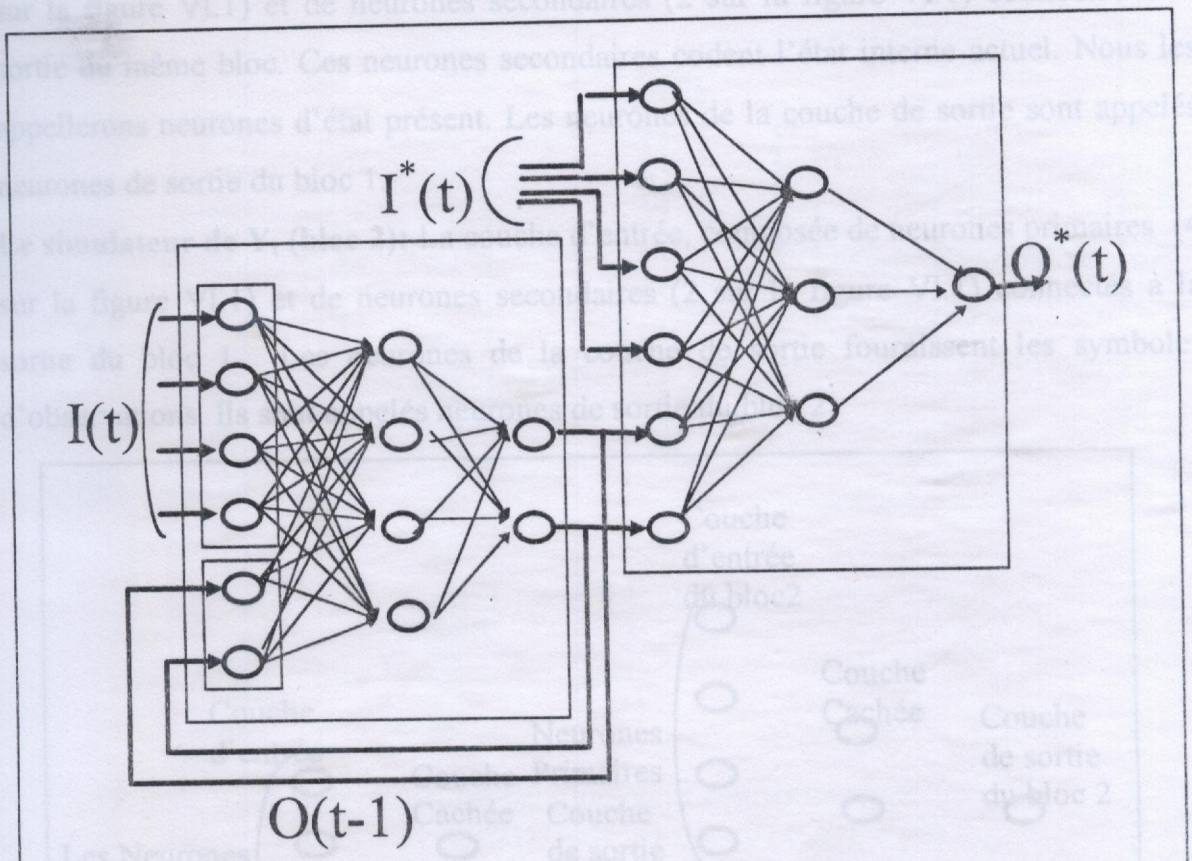


Figure VI.1: Schéma de l'architecture du simulateur connexionniste proposé

2.1- DENOMINATIONS:

Les neurones se répartissent en couches, il y a trois types de couches (Figure VI.2)

- Couche d'entrée

- couche cachée
- couche de sortie

Nous désignons sous le terme de neurone primaire les neurones d'entrée connectés à une entrée et sous le terme de neurones secondaires, les autres neurones de la couche d'entrée.

Le simulateur de X_t (bloc 1): la couche d'entrée se compose de neurones primaires (4 sur la figure VI.1) et de neurones secondaires (2 sur la figure VI.1) connectée à la sortie du même bloc. Ces neurones secondaires codent l'état interne actuel. Nous les appellerons neurones d'état présent. Les neurones de la couche de sortie sont appelés neurones de sortie du bloc 1.

Le simulateur de Y_t (bloc 2): La couche d'entrée, composée de neurones primaires (4 sur la figure VI.1) et de neurones secondaires (2 sur la figure VI.1) connectés à la sortie du bloc 1. Les neurones de la couche de sortie fournissent les symboles d'observations. Ils sont appelés neurones de sortie du bloc 2.

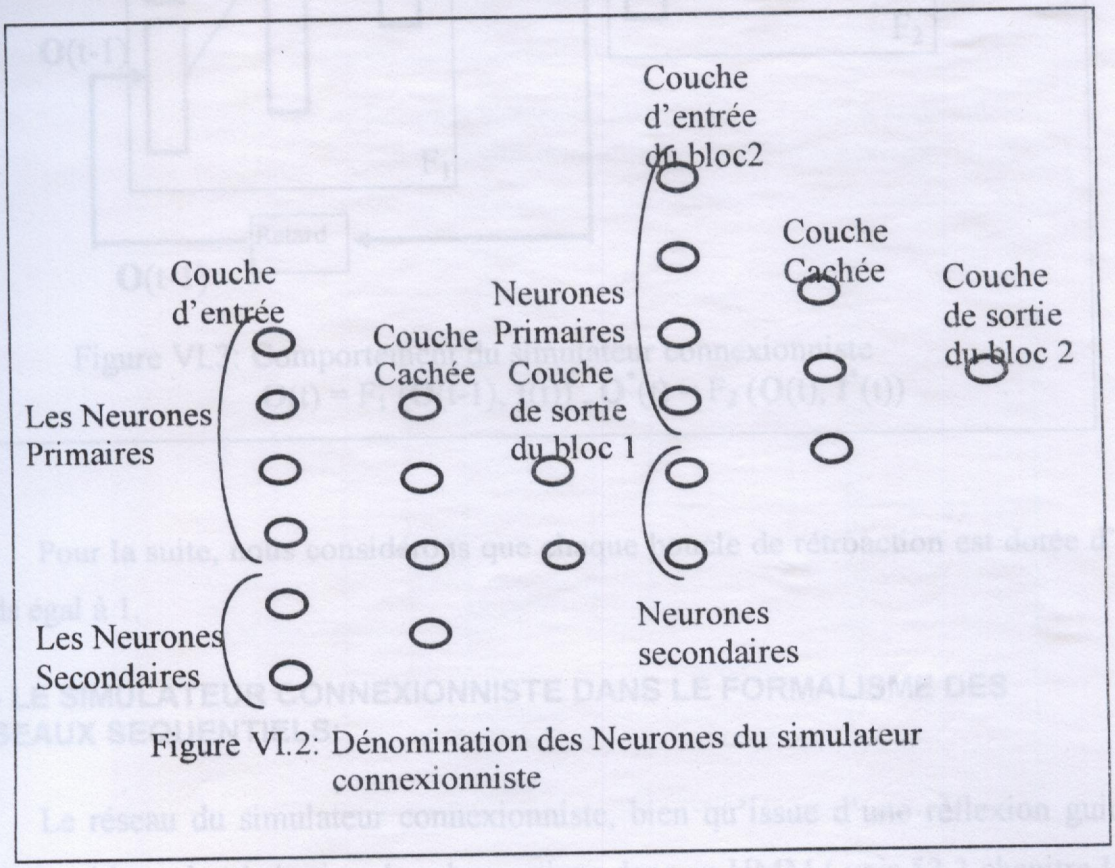


Figure VI.2: Dénomination des Neurones du simulateur connexionniste

2.2- FONCTIONNEMENT:

Le réseau réalisant la génération de X_t calcule à partir de l'état présent (**O**(t-1)) et de l'entrée actuelle (**I**(t)) l'état futur. Le réseau réalisant la génération de Y_t calcule à partir de l'entrée actuelle ($\mathbf{I}^*(t)$) et de l'état (**O**(t)) le symbole d'observation courant. L'ensemble (bloc 1 et bloc 2) réalise la génération d'un HMM. Sur la figure VI.3, nous avons schématisé ce fonctionnement.

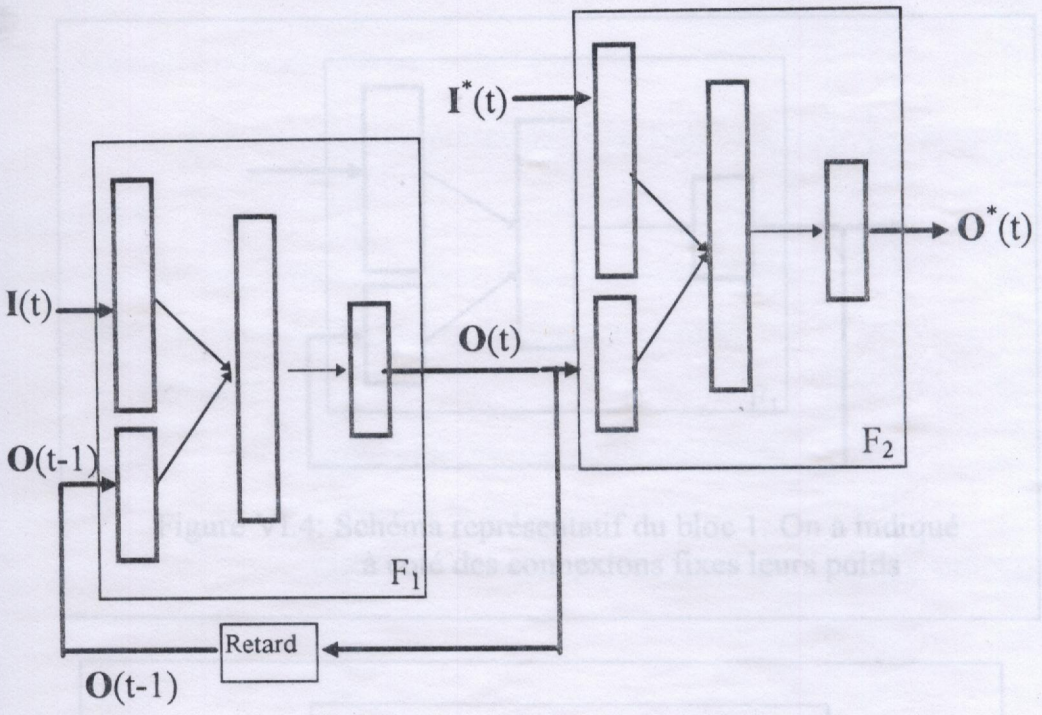


Figure VI.3: Comportement du simulateur connexionniste
$\mathbf{O}(t) = F_1(\mathbf{O}(t-1), \mathbf{I}(t))$, $\mathbf{O}^*(t) = F_2(\mathbf{O}(t), \mathbf{I}^*(t))$

Pour la suite, nous considérons que chaque boucle de rétroaction est dotée d'un poids égal à 1.

2.3- LE SIMULATEUR CONNEXIONNISTE DANS LE FORMALISME DES RESEAUX SEQUENTIELS:

Le réseau du simulateur connexionniste, bien qu'issue d'une réflexion guidée par l'algorithme de génération des observations dans un HMM (voir §2.3 chapitre IV),

rejoint le formalisme des réseaux connexionnistes pour le traitement des tâches de nature séquentielle (chapitre II). On peut voir cette relation à travers le bloc 1 (figure VI.4) qui génère X_t (i.e une chaîne de Markov). Ce dernier possède une mémoire à court-terme qui correspond à un cas trivial de mémoire exponentielle (μ_k= 0) des entrées et des sorties transformées. La différence entre notre simulateur et la machine connexionniste de Touzet (chapitre II § 5) réside dans le bloc 2 (figure VI.5). Dans notre modèle, il a un comportement combinatoire ($\mathbf{O}^*(t) = F(\mathbf{I}^*(t), \mathbf{O}(t))$ qui est tout à fait différent de celui de la machine de Touzet ($\mathbf{O}^*(t) = F(\mathbf{I}(t), \mathbf{O}(t-1))$

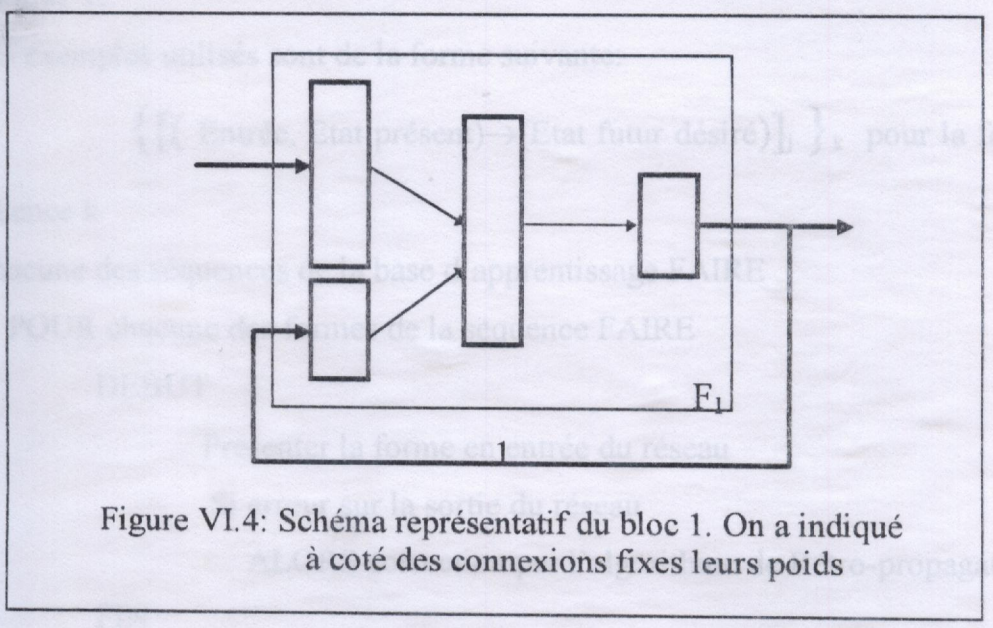


Figure VI.4: Schéma représentatif du bloc 1. On a indiqué à coté des connexions fixes leurs poids

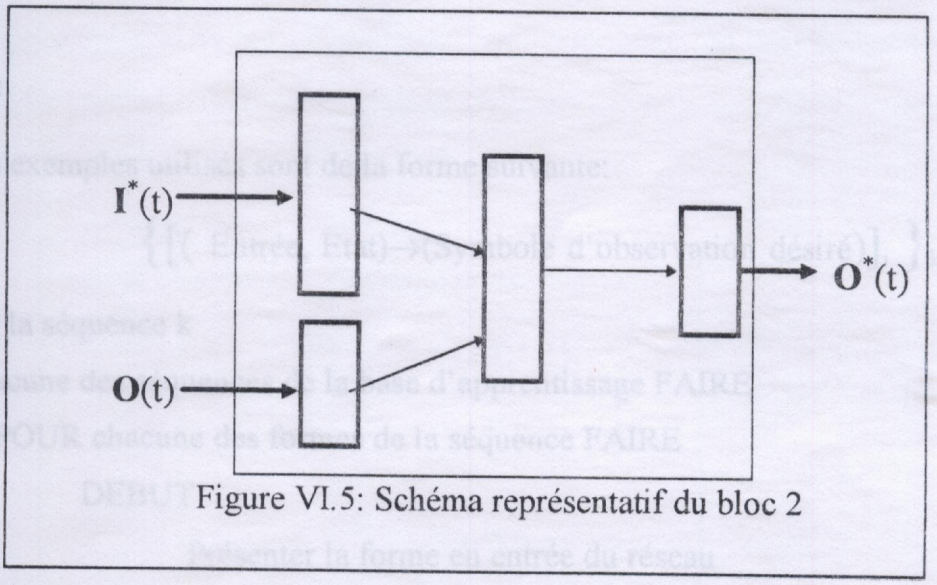


Figure VI.5: Schéma représentatif du bloc 2

2.4- APPRENTISSAGE:

L'apprentissage est réalisé séparément sur chacun des deux réseaux composant le simulateur connexionniste par l'algorithme de rétro-propagation du gradient. On définit des séquences d'apprentissage qui pour le réseau générant X_t sont de la forme ((entrée, état présent)→(état futur désiré)) et pour le réseau générant Y_t sont des associations de la forme ((entrée, état)→(observation désirée)). Bien que sépares durant l'apprentissage, les deux réseaux sont réunis en utilisation

Algorithmes d'apprentissage:

i) pour le bloc 1:

Les exemples utilisés sont de la forme suivante:

$$\{[(\text{ Entrée, Etat présent})\rightarrow(\text{Etat futur désiré})]_j\}_k \text{ pour la forme j}$$

de la séquence k

POUR chacune des séquences de la base d'apprentissage FAIRE

POUR chacune des formes de la séquence FAIRE

DEBUT

Présenter la forme en entrée du réseau

Si erreur sur la sortie du réseau

ALORS correction par l'algorithme de Rétro-propagation

FIN

FIN

ii) bloc 2:

Les exemples utilisés sont de la forme suivante:

$$\{[(\text{ Entrée, Etat})\rightarrow(\text{Symbole d'observation désiré})]_j\}_k \text{ pour la}$$

forme j de la séquence k

POUR chacune des séquences de la base d'apprentissage FAIRE

POUR chacune des formes de la séquence FAIRE

DEBUT

Présenter la forme en entrée du réseau

Si erreur sur la sortie du réseau

ALORS correction par l'algorithme de Rétro-propagation

FIN

FIN

2.5- EXPERIMENTATIONS:

Pour mener nos expérimentations nous avons choisi un exemple d'une chaîne de Markov cachée à deux états et à deux symboles d'observations telle que:

$$A = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = \begin{vmatrix} 0.4 & 0.6 \\ 0.2 & 0.8 \end{vmatrix}$$

$$A_0 = \left[a_{01} , a_{02} \right] = \left[0.3 , 0.7 \right]$$

$$B = \begin{bmatrix} b_1(1) & b_1(2) \\ b_2(1) & b_2(2) \end{bmatrix} = \begin{bmatrix} 0.1 & 0.9 \\ 0.5 & 0.5 \end{bmatrix}$$

Où $E = \{x_1 , x_2\}$ et $V = \{v_1 , v_2\}$

i) Codage des entrées/ sorties du réseau:

Pour l'utilisation de notre simulateur connexionniste, on doit coder nos entrées et sorties. Pour cela on va utiliser le codage suivant:

- codage des probabilités de transitions:

notation	valeur numérique	codage correspondant
a_{01}	0.3	(1,1,1,1)
a_{02}	0.7	(1,1,1,0)
a_{11}	0.4	(1,0,0,0)
a_{12}	0.6	(0,1,0,0)
a_{21}	0.2	(0,0,1,0)
a_{22}	0.8	(0,0,0,1)

- Codage des probabilités des symboles d'observations:

notation	valeur numérique	codage correspondant
$b_1(1)$	0.1	(1,0,0,0)
$b_1(2)$	0.9	(0,1,0,0)
$b_2(1)$	0.5	(0,0,1,0)
$b_2(2)$	0.5	(0,0,0,1)

- Codage des états:

Etat	codage correspondant
ω_1	(1,0)
ω_2	(0,1)

- Codage des symboles d'observations

symbole	codage correspondant
v_1	1
v_2	0

A partir de ce codage, nous avons retenu l'architecture suivante:

- le réseau réalisant la génération de X_t est constitué de 4 neurones primaires et 2 neurones secondaires sur la couche d'entrée, 4 neurones cachés sur la couche cachée et 2 neurones sur la couche de sortie.

- le réseau réalisant la fonction de sortie est constitué de 4 neurones primaires et 2 neurones secondaires sur la couche d'entrée, 3 neurones sur la couche cachée et 1 neurone de sortie.

ii) Comportement en utilisation (Figure VI.6):

Nous supposons que l'apprentissage à été parfaitement réalisé sur notre simulateur. Nous nous attachons ici à décrire son comportement en utilisation.

A la date t =1:

- Le bloc 1:

Initialement les sorties du bloc 1 sont nulles (0,0). Ses dernières sont appliquées sur les entrées secondaires. La forme d'entrée (1,1,1,1) qui désigne a_{01} est appliquée sur les entrées primaires. Par propagation, ce bloc donne (1,0) qui désigne ω_1.

- Le bloc 2:

Le bloc 2 reçoit la sortie du bloc 1 (1,0) sur ses cellules secondaires et l'entrée primaire (1,0,0,0) qui désigne b_1 (1), et donne en sortie (1) qui désigne v_1 .

A la date t:=2:

- Le bloc 1:

Il reçoit l'état présent (1,0) sur ses entrées secondaires. La forme d'entrée (1,0,0,0) qui désigne a_{11} est appliquée sur les entrées primaires. Il donne en sortie (1,0).

- Le bloc 2

Il reçoit la sortie du bloc 1 (1,0) sur ses cellules secondaires et l'entrée primaire (0,1,0,0) qui désigne b_1 (2), et donne (0) qui désigne v_2 .

iii) Apprentissage:

Le codage et la structure du simulateur choisis sont ceux précédemment décrits.

- Fonctions d'activation:

La fonction d'activation des neurones d'entrée composant notre simulateur est l'identité:

$$f(z):= z$$

Pour les autres neurones on a:

$$f(z):= 1/(1+ \exp(-\lambda.z))$$

- Bases d'exemples d'apprentissage:

Pour le bloc 1:

$$E = \{[((0,0),(1,1,1,1))\rightarrow(1,0)];\ [((0,0),(1,1,1,0))\rightarrow(0,1)];$$
$$[((1,0),(1,0,0,0))\rightarrow(1,0)];\ [((1,0),(0,1,0,0))\rightarrow(0,1)];$$
$$[((0,1),(0,0,1,0,))\rightarrow(1,0)];\ [((0,1),(0,0,0,1))\rightarrow(0,1)]\}$$

Pour le bloc 2:

$$E = \{[((1,0),(1,0,0,0))\rightarrow(1)];\ [((1,0),(0,1,0,0))\rightarrow(0)];$$

$$[((0,1),(0,0,1,0))\rightarrow(1)];\ [((0,1),(0,0,0,1))\rightarrow(0)]\}$$

iii) Résultats:

Pour le bloc 1:

Environ 6052 itérations sont nécessaires pour que les exemples d'apprentissage soient parfaitement appris. Le critère utilisé pour arrêter l'apprentissage est une valeur de l'erreur ε inférieure à 0.1. C'est à dire:

$$\varepsilon \geq | O_i(\mathbf{I}) - Z_i(\mathbf{I}) |$$

où $O_i(\mathbf{I})$ désigne la i^{eme} sortie du réseau (i:=1,2) pour l'entrée **I** et $Z_i(\mathbf{I})$ est sa sortie désirée.

Pour le bloc 2:

Le nombre d'itérations nécessaire pour que les exemples d'apprentissage soient parfaitement appris est de l'ordre de 971 itérations pour le même critère d'arrêt que précédemment.

iv) la généralisation:

Dans la phase de généralisation, nous avons testé notre simulateur connexionniste sur des séquences de longueur 7. Les taux de réussite τ_1 et τ_2 (respectivement pour le bloc 1 et le bloc 2) sont de 1 en acceptant une erreur inférieure à ε:= 0.1. Par exemple sur la séquence du tableau suivant la reconnaissance est parfaite.

$$[((0,1),(0,0,1,0))\rightarrow(1)];\ [((0,1),(0,0,0,1))\rightarrow(0)]\}$$

iii) Résultats:

Pour le bloc 1:

Environ 6052 itérations sont nécessaires pour que les exemples d'apprentissage soient parfaitement appris. Le critère utilisé pour arrêter l'apprentissage est une valeur de l'erreur ε inférieure à 0.1. C'est à dire:

$$\varepsilon \geq | O_i(\mathbf{I}) - Z_i(\mathbf{I}) |$$

où $O_i(\mathbf{I})$ désigne la i^{eme} sortie du réseau (i:=1,2) pour l'entrée **I** et $Z_i(\mathbf{I})$ est sa sortie désirée.

Pour le bloc 2:

Le nombre d'itérations nécessaire pour que les exemples d'apprentissage soient parfaitement appris est de l'ordre de 971 itérations pour le même critère d'arrêt que précédemment.

iv) la généralisation:

Dans la phase de généralisation, nous avons testé notre simulateur connexionniste sur des séquences de longueur 7. Les taux de réussite τ_1 et τ_2 (respectivement pour le bloc 1 et le bloc 2) sont de 1 en acceptant une erreur inférieure à ε:= 0.1. Par exemple sur la séquence du tableau suivant la reconnaissance est parfaite.

t	entrées primaires du bloc 1 **I**(t)	réponse désirées du bloc 1 **Z**(**I**(t))	réponse calculées du bloc 1 **O**(t)
1	$(1,1,1,1)\rightarrow a_{01}$	$(1,0)\rightarrow\omega_1$	(0.9361 , 0.0846)
2	$(1,0,0,0)\rightarrow a_{11}$	$(1,0)\rightarrow\omega_1$	(0.9005 , 0.0232)
3	$(1,0,0,0)\rightarrow a_{11}$	$(1,0)\rightarrow\omega_1$	(0.9004 , 0.0234)
4	$(0,1,0,0)\rightarrow a_{12}$	$(0,1)\rightarrow\omega_2$	(0.0026 , 0.9001)
5	$(0,0,0,1)\rightarrow a_{22}$	$(0,1)\rightarrow\omega_2$	(0.0026 , 0.9001)
6	$(0,0,0,1)\rightarrow a_{22}$	$(0,1)\rightarrow\omega_2$	(0.0023 , 0.9014)
7	$(0,0,1,0)\rightarrow a_{21}$	$(1,0)\rightarrow\omega_1$	(0.9001 , 0.0136)

entrées primaires du bloc 2 $\mathbf{I}^*$ (t)	réponse désirées du bloc 2 $\mathbf{Z}^*(\mathbf{I}^*(t))$	réponse calculées du bloc 2 $\mathbf{O}^*$(t)
$(1,0,0,0)\rightarrow b_1(1)$	$(1)\rightarrow v_1$	(0.9004)
$(1,0,0,0)\rightarrow b_1(1)$	$(1)\rightarrow v_1$	(0.9003)
$(0,1,0,0)\rightarrow b_1(2)$	$(0)\rightarrow v_2$	(0.0974)
$(0,0,1,0)\rightarrow b_2(1)$	$(1)\rightarrow v_1$	(0.9001)
$(0,0,1,0)\rightarrow b_2(1)$	$(1)\rightarrow v_1$	(0.9001)
$(0,0,0,1)\rightarrow b_2(2)$	$(0)\rightarrow v_2$	(0.0991)
$(0,1,0,0)\rightarrow b_1(2)$	$(0)\rightarrow v_2$	(0.0992)

Pour les séquences de longueurs supérieures à 7, le taux de réussite décroît avec t. Voici la courbe (Figure VI.8) de τ_1 en fonction de t pour une séquence dont la longueur varie de 1 à 100 avec ε =0.1.

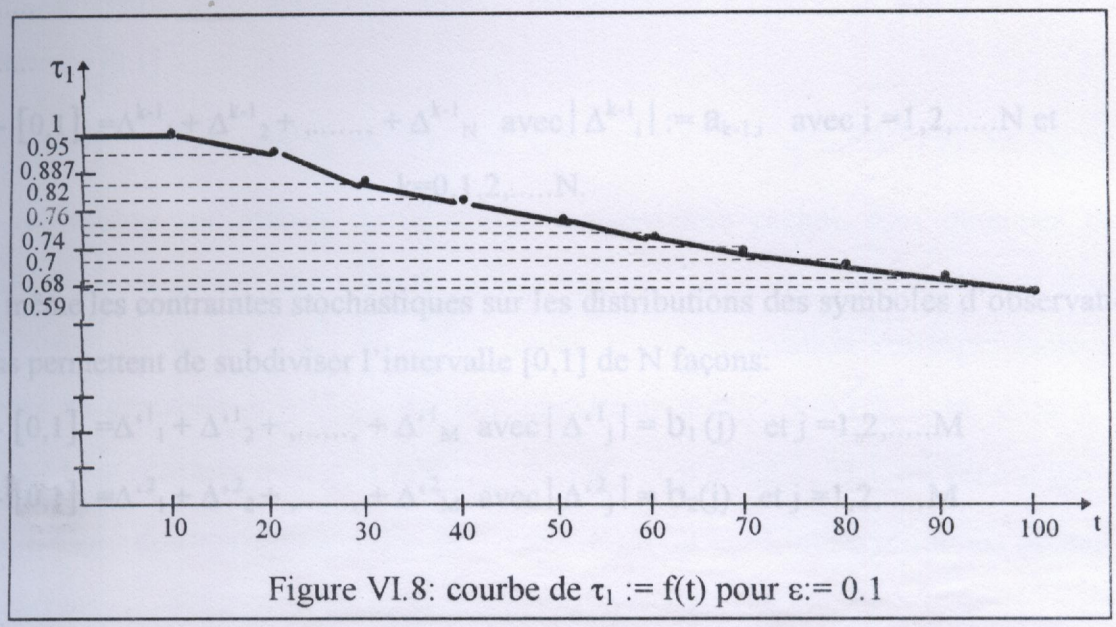


Figure VI.8: courbe de $\tau_1 := f(t)$ pour $\varepsilon := 0.1$

Pour un choix de $\varepsilon := 0.5$, on a traité des séquences de longueur 100 avec des taux de succès (τ_1 et τ_2) de 1.

3- MODELISATION CONNEXIONNISTE DE LA METHODE DE SIMULATION:

Dans cette partie , nous allons proposer un autre simulateur connexionniste différent de celui de la première approche. L'architecture de ce réseau est faite en se basant sur la méthode de simulation statistique [2] d'une chaîne de Markov cachée que nous présentons ci dessous.

3.1- METHODE DE SIMULATION D'UNE CHAINE DE MARKOV CACHEE:

Soit une chaîne de Markov cachée $\{X_t , Y_t\}_{t\geq 1}$ avec E et V des espaces respectivement d'états de symbole d'observations tels que: $|E| := N$ et $|V| := M$. Les contraintes stochastiques sur les probabilités initiales et les probabilités de transition entre états permettent de subdiviser l'intervalle [0,1] de (N+1) façons :

1) - $[0,1] = \Delta^0_1 + \Delta^0_2 + ,\ldots\ldots, + \Delta^0_N$ avec $|\Delta^0_i| = a_{0i}$ et i =1,2,.....N

.

2) - $[0,1] = \Delta^1_1 + \Delta^1_2 + ,\ldots\ldots, + \Delta^1_N$ avec $|\Delta^1_i| = a_{1i}$ et i =1,2,.....N

3) - $[0,1] = \Delta^2_1 + \Delta^2_2 + ,\ldots\ldots, + \Delta^2_N$ avec $|\Delta^2_i| := a_{2i}$ et i =1,2,.....N

.....

.....

.....

k) - $[0,1] = \Delta^{k-1}{}_1 + \Delta^{k-1}{}_2 + ,....., + \Delta^{k-1}{}_N$ avec $|\Delta^{k-1}{}_i| := a_{k-1,i}$ avec i =1,2,.....N et k=0,1,2,.....N.

De même les contraintes stochastiques sur les distributions des symboles d'observation nous permettent de subdiviser l'intervalle [0,1] de N façons:

1) - $[0,1] = \Delta'^{1}{}_1 + \Delta'^{1}{}_2 + ,....., + \Delta'^{1}{}_M$ avec $|\Delta'^{1}{}_j| = b_1(j)$ et j =1,2,.....M

2) - $[0,1] = \Delta'^{2}{}_1 + \Delta'^{2}{}_2 + ,....., + \Delta'^{2}{}_M$ avec $|\Delta'^{2}{}_j| = b_2(j)$ et j =1,2,.....M

.....

.....

i) - $[0,1] = \Delta'^{i}{}_1 + \Delta'^{i}{}_2 + ,....., + \Delta'^{i}{}_M$ avec $|\Delta'^{i}{}_j| = b_i(j)$ avec j =1,2,.....M et i =1,2,...,N.

Donc on a: N.(N+1) subdivisions de type $\Delta^{k-1}{}_i$ et N.M de type $\Delta'^{i}{}_j$. Déterminons les réalisations successives de la chaîne de Markov cachée comme suit: Initialement ,on généré une variable $\chi_0 \rightarrow U_{[0,1]}$, si $\chi_0 \in \Delta^0{}_i$ on a: l'état x_i. Pour déterminer l'observation correspondante à cet état, On génère une variable $\gamma_0 \rightarrow U_{[0,1]}$, si $\gamma_0 \in \Delta'^{1}{}_j$ on a: v_j. Supposons maintenant qu'a l'instant t on se trouve avec x_l (avec l :=1,2,...,N) et v_p (p:=1,2,...,M), alors on détermine l'état et le symbole d'observation à l'instant t+1 en gênerons $\chi_{t+1} \rightarrow U_{[0,1]}$ (si $\chi_{t+1} \in \Delta^1{}_d$ on a: état x_d , d:=1,2,...,N) et $\gamma_{t+1} \rightarrow U_{[0,1]}$ (si $\gamma_{t+1} \in \Delta'^{d}{}_q$ on a: v_q , q:=1,2,...,M).

3.2 ANALYSE ET FORMULATION DU PROBLEME:

Pour atteindre notre but qui est la construction d'un réseau de neurones qui imite la méthode de simulation statistique , nous proposons une petite modification de cette dernière. Au lieu de l'intervalle [0,1] pour la génération des états , on partage l'intervalle [0, N+1] comme suit:

$$[0,N+1] := \Delta^0{}_1 + ... + \Delta^0{}_N + \Delta^1{}_1 + ... + \Delta^1{}_N + ... + \Delta^{k-1}{}_1 + ... + \Delta^{k-1}{}_N + ... + \Delta^N{}_1 + ... + \Delta^N{}_N$$

De même pour les observations , on subdivise l'intervalle [0,N] au lieu de l'intervalle [0,1]

$$[0,N] := \Delta^{'1}_1 + \ldots + \Delta^{'1}_M + \Delta^{'2}_1 + \ldots + \Delta^{'2}_M + \ldots + \Delta^{'i}_1 + \ldots + \Delta^{'i}_M + \ldots + \Delta^{'N}_1 + \ldots + \Delta^{'N}_M$$

Les réalisations successives de la chaîne markovienne cachée sont obtenues par: Initialement ,on généré une variable aléatoire $\chi_0 \rightarrow U_{[0,1]}$, χ_0 doit appartenir à une subdivision Δ^0_i, ou i:= 1,2,...,N ce qui donne x_i . L'observation correspondante est obtenue en gênerons $\gamma_0 \rightarrow U_{[0,1]}$ qui doit appartenir à $\Delta^{'i}_j$, j:= 1,2,....,M, d'ou v_j. A l'instant t on se trouve avec x_l (l := 1,.....,N) et v_p (p:= 1,2,...,M), on génère $\chi_{t+1} \rightarrow U_{[l+0,l+1]}$, χ_{t+1} doit appartenir à Δ^l_d (d:= 1,2,...,N) d'ou x_d. L'observation est obtenu e en générant $\gamma_{t+1} \rightarrow U_{[d-1,d]}$, d'où v_q si $\gamma_{t+1} \in \Delta^{'d}_q$, q:= 1,2,...,M

A partir de ce qu'on vient de voir, le processus se présente comme suit:

- Pour la chaîne de Markov X_t (Figure VI.9):

Etant donné un état ω_l (l:=1,2,...,N) à un instant t , on génère $\chi_{t+1} \Rightarrow U_{[l+0,l+1]}$ qui donne Δ^l_d (d:= 1,2,...,N) auquel correspond x_d.

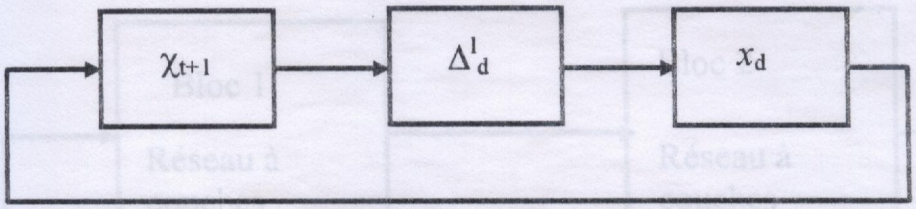


Figure VI.9 : Schéma représentatif du processus de simulation de X_t

- Pour la composante observable Y_t (Figure VI.10):

Etant donné l'état ω_d à l'instant t, on génère $\gamma_{t+1} \downarrow U_{[d-1,d]}$ qui donne $\Delta^{'d}_q$ (q:=1,2,...,M), auquel correspond à v_q.

A son tour le réseau simulateur de Y_t est composé de deux blocs en série: Le premier pour classer les γ_t (pour un γ_t donné ,il correspond un Δ^{i}_{j} pour j:=1,2,...M et i:=1,2,...,N) et le second bloc pour classer les Δ^{i}_{j} (pour un Δ^{i}_{j} donné , il correspond un v_j). Ces deux blocs sont reliés par des connexions de poids fixe.

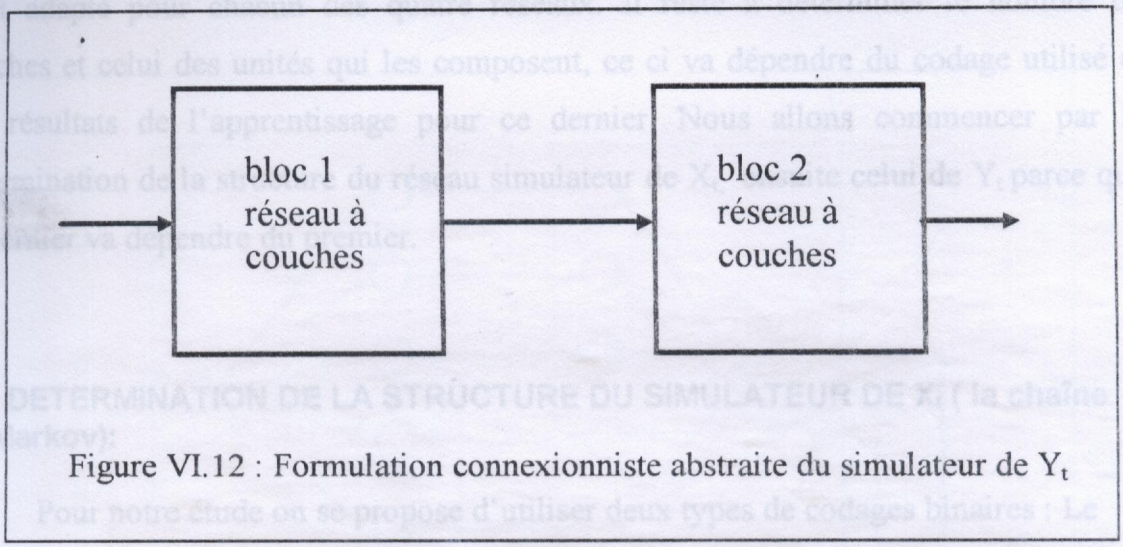


Figure VI.12 : Formulation connexionniste abstraite du simulateur de Y_t

En fin, la formulation du simulateur connexionniste de la chaîne de Markov cachée est représenté par la figure suivante:

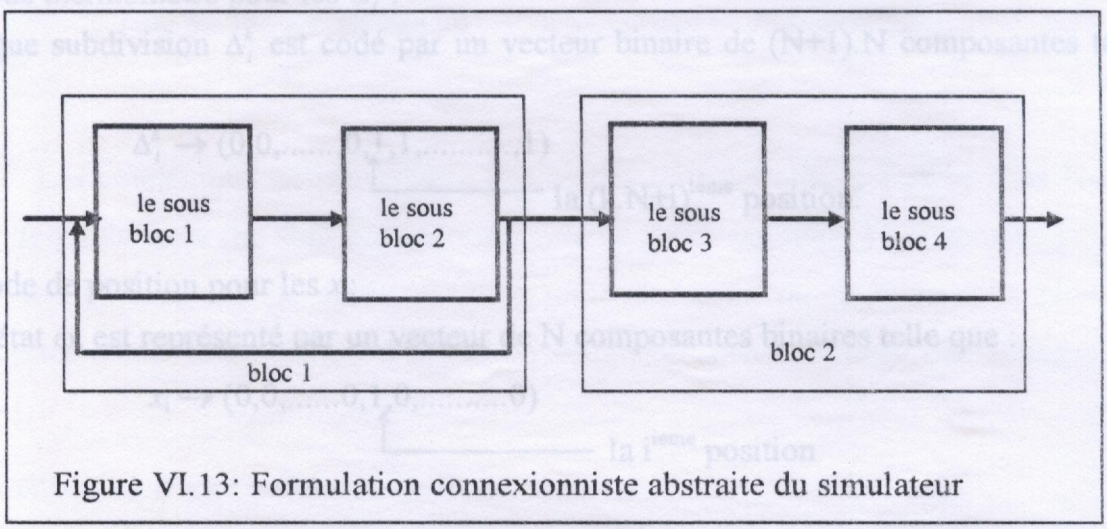


Figure VI.13: Formulation connexionniste abstraite du simulateur

En résumé , le simulateur connexionniste de la chaîne de Markov cachée est composé de deux bloc: Le premier bloc pour X_t (La chaîne de Markov) qui est composé à son tour de deux sous blocs (1 et 2 sur la figure VI.13).Le second comporte aussi deux sous blocs (3 et 4 sur la figure VI.13). Les quatre réseaux connexionnistes composant le simulateur vont servir de classifieurs. Une architecture feed-forward est ainsi adapté pour chacun des quatre réseaux. Il reste à déterminer le nombre de couches et celui des unités qui les composent, ce ci va dépendre du codage utilisé et des résultats de l'apprentissage pour ce dernier. Nous allons commencer par la détermination de la structure du réseau simulateur de X_t, ensuite celui de Y_t parce que ce dernier va dépendre du premier.

3.3- DETERMINATION DE LA STRUCTURE DU SIMULATEUR DE X_t (la chaîne de Markov):

Pour notre étude on se propose d'utiliser deux types de codages binaires : Le code thermomètre pour coder les Δ^k_i (i:=1,2,...,N, k:=0,1,2,...,N) et le code de position pour les ω_i (i:=1,2,...,N).

i) Le codage:

- Code thermomètre pour les Δ_i^k :

chaque subdivision Δ_i^k est codé par un vecteur binaire de (N+1).N composantes telle que :

$$\Delta_i^k \rightarrow (0,0,.....,0,1,1,.....,1)$$

(la $(k.N+i)^{ieme}$ position.)

- Code de position pour les x_i:

Un état ω_i est représenté par un vecteur de N composantes binaires telle que :

$$x_i \rightarrow (0,0,.....0,1,0,.....0)$$

(la i^{ieme} position)

ii) Le nombre d'unités d'entrée et de sortie:

De cette manière ,le premier réseau comporte (N+1).N neurones de sortie et un neurone d'entré, et le second se compose de (N+1).N unités en entré et N unités de sortie .Le nombre de couches cachées et de cellules qui les composent seront déterminés en fonction des résultats de l'apprentissage .Sur la figure VI.14, on montre les neurones et les connexions du modèle sans couches cachées. Les connexions se repartissent en deux classes ,selon la nature fixe ou plastique du poids de la connexion.

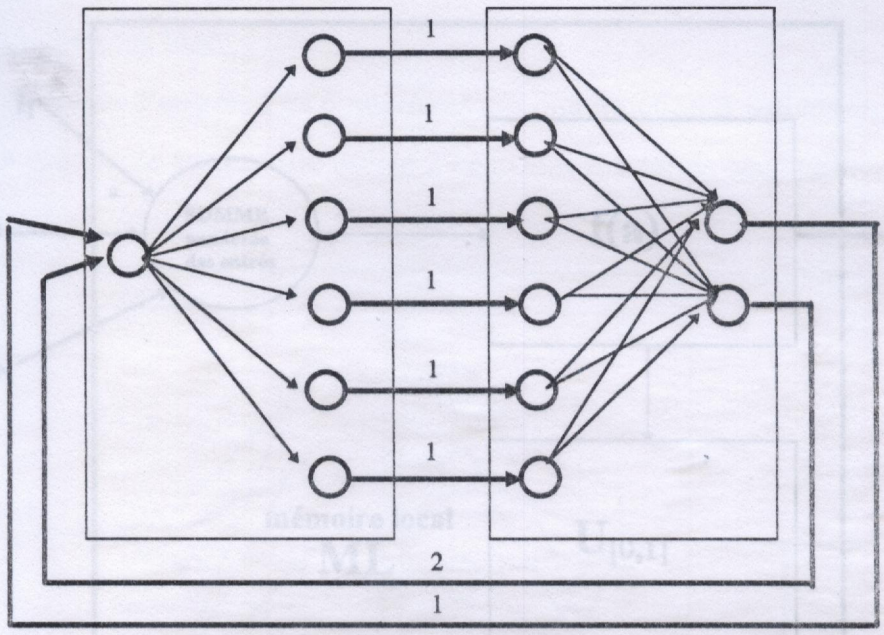


Figure VI.14 : Le schéma du réseau Simulateur de X_t avec N:=2

Les connexions en provenance des sorties du sous bloc 2 vers le sous bloc 1 sont de poids fixes (ne sont pas modifies lors de l'apprentissage) qui sont de poids i pour le i^{ieme} neurone de sortie. Chaque unité de sortie du premier sous bloc est connectée à une seule cellule d'entrée du deuxième sous bloc avec une synapse qui vaut 1.

iii) Utilisation d'une unité stochastique:

Pour introduire l'aspect stochastique de la méthode de simulation dans le modèle connexionniste de X_t , nous proposons que l'unité d'entré du premier réseau

soit stochastique . Pour cela on développe sur la base des travaux de Asselin J. P. [3] un type de neurone formel générique dont la sortie O n'est plus déterministe mais stochastique (figure VI.15). La mémoire locale du neurone contient un générateur uniforme sur l'intervalle [0,1] telle que O =f(a) +u où a est la somme pondérée des entrés et u est la réalisation de la loi uniforme.

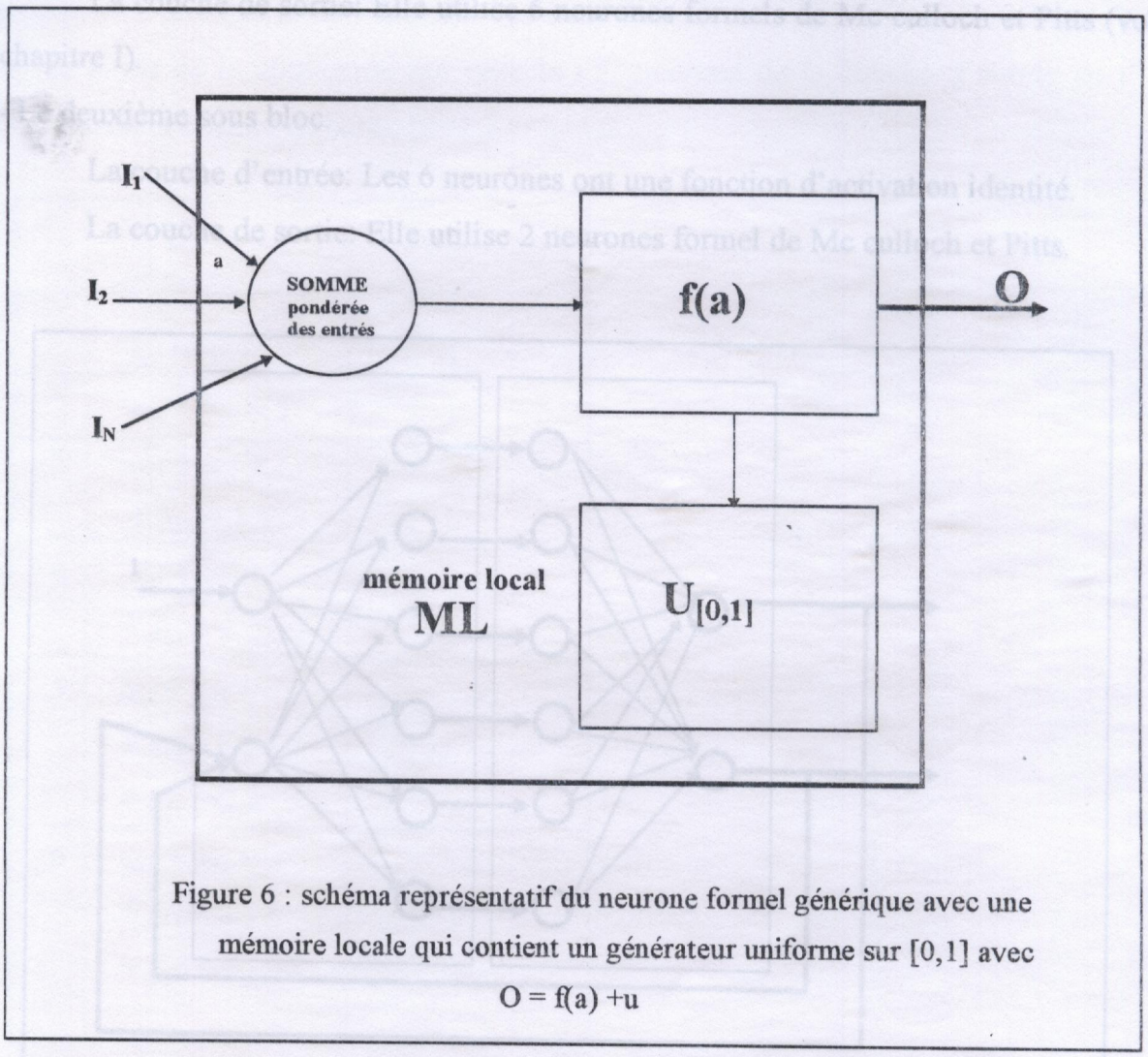


Figure 6 : schéma représentatif du neurone formel générique avec une mémoire locale qui contient un générateur uniforme sur [0,1] avec O = f(a) +u

iv) Les expérimentations:

Cette partie présente les expérimentations que nous avons mené sur le modèle connexionniste de X_t en utilisons un exemple numérique d'une chaîne de Markov à deux états (A et A_0 du § 2.5), nous ont permet de voir que les deux sous blocs n'ont

pas besoin de couche cachées. L'architecture retenue est représenter par la figure VI.16. Les neurones composant ce réseau sont définis comme suit:

- Le premier sous bloc

La couche d'entrée: Le premier neurone a une fonction d'activation identité. Il permet d'introduire des seuils non nuls pour les unités de sortie [3]. Cette unité reçoit comme entrée l'unité. Le second est l'unité stochastique qu'on a déjà défini ci dessus.

La couche de sortie: Elle utilise 6 neurones formels de Mc culloch et Pitts (voir chapitre I).

- Le deuxième sous bloc:

La couche d'entrée: Les 6 neurones ont une fonction d'activation identité.

La couche de sortie: Elle utilise 2 neurones formel de Mc culloch et Pitts.

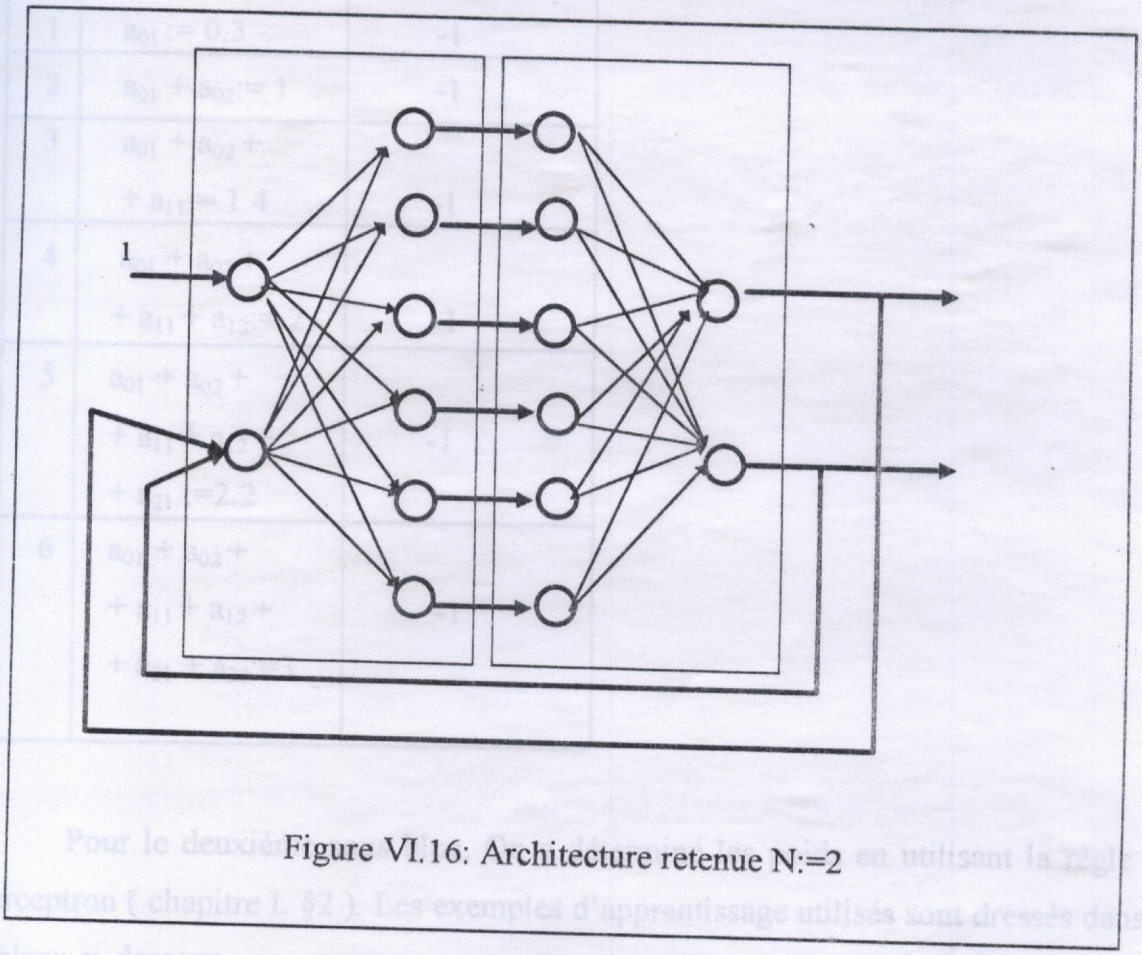


Figure VI.16: Architecture retenue N:=2

- les poids des connexions:

Les valeurs des poids des connexions de poids fixes sont déjà déterminés . Il reste à trouver les valeurs des poids des connexions de poids plastiques des deux sous blocs.

Pour le premier bloc, on a déterminé les poids intuitivement sans recours à l'apprentissage et voici leurs valeurs dans le tableau ci-dessous en fonction des probabilités initiales et de transition des états de la chaîne de Markov. On désigne par w_{ji} le poids de la connexion qui relie le i^{eme} neurone de la couche d'entrée au j^{eme} neurone de la couche de sortie (i:=1,2 et j:=1,2,...,6).

j	w_{j1}	w_{j2}
1	$a_{01} := 0.3$	-1
2	$a_{01} + a_{02} := 1$	-1
3	$a_{01} + a_{02} +$ $+ a_{11} := 1.4$	-1
4	$a_{01} + a_{02} +$ $+ a_{11} + a_{12} := 2$	-1
5	$a_{01} + a_{02} +$ $+ a_{11} + a_{12} +$ $+ a_{21} := 2.2$	-1
6	$a_{01} + a_{02} +$ $+ a_{11} + a_{12} +$ $+ a_{21} + a_{22} := 3$	-1

Pour le deuxième sous bloc, On a déterminé les poids en utilisant la règle du perceptron (chapitre I. §2). Les exemples d'apprentissage utilisés sont dressés dans le tableau ci-dessous.

Les exemples k	Les entrées $I^k := (I^k_1, I^k_2, I^k_3, I^k_4, I^k_5, I^k_6)$	Les sorties désirées $Z^k := (Z^k_1, Z^k_2)$
1	(1,1,1,1,1,1)	(1,0)
2	(0,1,1,1,1,1)	(0,1)
3	(0,0,1,1,1,1)	(1,0)
4	(0,0,0,1,1,1)	(0,1)
5	(0,0,0,0,1,1)	(1,0)
6	(0,0,0,0,0,1)	(0,1)

L'algorithme d'apprentissage converge pour ce réseau vers plusieurs solution selon les poids de départ. On peut retenir la solution présentée ci dessous. On désigne par w_{ji} le poids de la connexion qui relie le i^{eme} neurone de la couche d'entrée au j^{eme} neurone de la couche de sortie (i:=1,..,6 et j:=1,2).

i	w_{1i}	w_{2i}
1	1	-1
2	-1	1
3	1	-1
4	-1	1
5	1	-1
6	-1	0.8

3.4 DETERMINATION DE LA STRUCTURE DU SIMULATEUR DE Y_t :

Pour le deuxième bloc, on utilise comme précédemment (§ 3.3) deux type de codage binaires: Le code thermomètre pour coder les Δ^{i}_{j} (i:=1,2,...,N, j:=1,2,...,M) et le code de position pour les v_j (j:=1,2,...,M).

i) Le codage:

- Code thermomètre pour les Δ^{i}_{j} :

chaque subdivision Δ^{i}_{j} est codée par un vecteur binaire de M.N composantes tel que :

$$\Delta^{i}_{j} \rightarrow (0,0,.....,0,1,1,.....,1)$$

la $((i-1).M+j)^{ieme}$ position.

- Code de position pour les v_j :

Un symbole d'observation v_j est représenté par un vecteur de M composantes binaires tel que :

$$v_j \rightarrow (0,0,.....0,1,0,.....0)$$

la j^{ieme} position

ii) Le nombre d'unités d'entrée et de sortie:

De cette manière ,le premier réseau comporte M.N neurones de sortie et deux neurones d'entrée. Le second se compose de M.N unités en entrée et M unités de sortie .Le nombre de couches cachées est nul. Sur la figure VI.17, on montre les neurones et les connexions du modèle. La figure VI.18 montre les connexions qui relient le simulateur de X_t à celui de Y_t , ses connexions sont de poids fixe .

iv) Les expérimentations:

Nous avons utilisé l'exemple numérique du paragraphe 2.5. Les poids des connexions en provenance des sorties du deuxième sous bloc du simulateur de X_t vers la deuxième unité d'entrée du premier sous bloc du simulateur de Y_t (sur la figure VI.18) sont de poids i pour le i^{eme} neurone de sortie. Les unités composant le réseau de Y_t (Figure VI.17) sont définies comme suit:

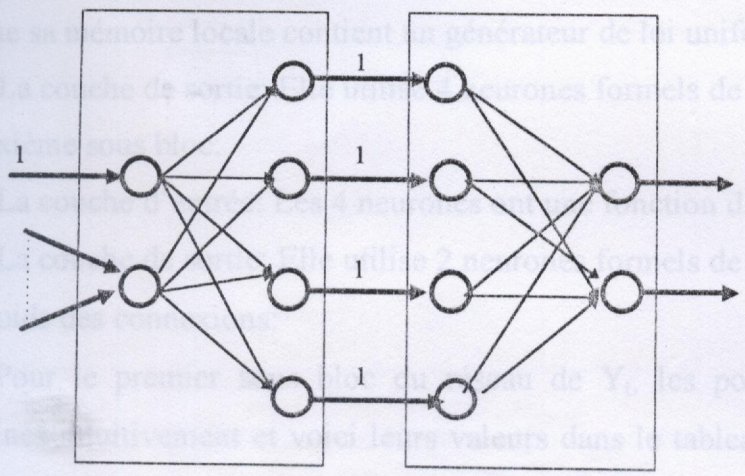


Figure VI.17 : Le schéma du réseau Simulateur de Y_t avec N:=2 et M:=2

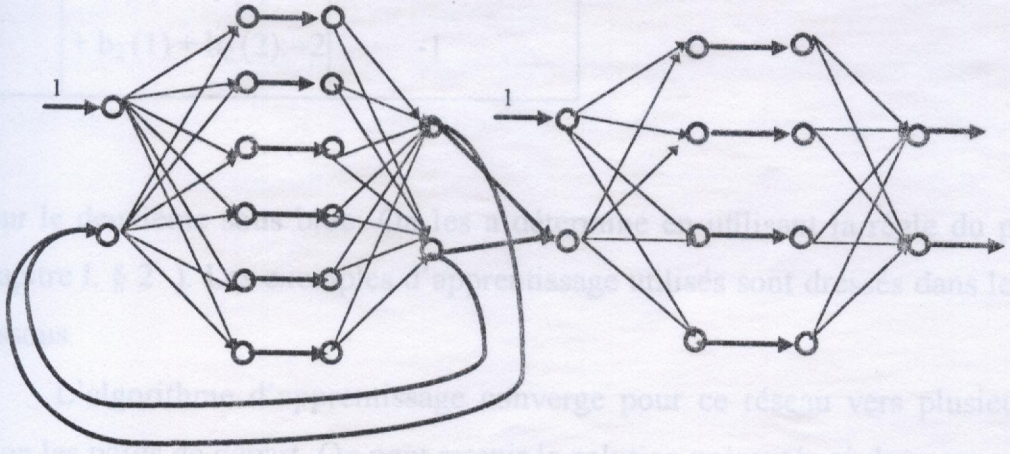


Figure VI.18: schéma du simulateur connexionniste d'HMM avec N:2 et M:=2

- Le premier sous bloc:

Les exemples k	Les entrées $I^k := (I^k_1, I^k_2, I^k_3, I^k_4)$	Les sorties désirées $Z^k := (Z^k_1, Z^k_2)$
1	(1,1,1,1)	(1,0)
2	(0,1,1,1)	(0,1)
3	(0,0,1,1)	(1,0)
4	(0,0,0,1)	(0,1)

L'algorithme d'apprentissage converge pour ce réseau vers plusieurs solutions selon les poids de départs. On peut retenir la solution présentée ci dessous.

i	w_{1i}	w_{2i}
1	1	-1
2	-1	1
3	1	-1
4	-1	0.8

3.5- COMPORTEMENT EN UTILISATION:

Après avoir déterminer les poids des différentes connexions de notre simulateur neuronal de la chaîne de Markov cachée, nous nous attachons ici à décrire son comportement en utilisation.

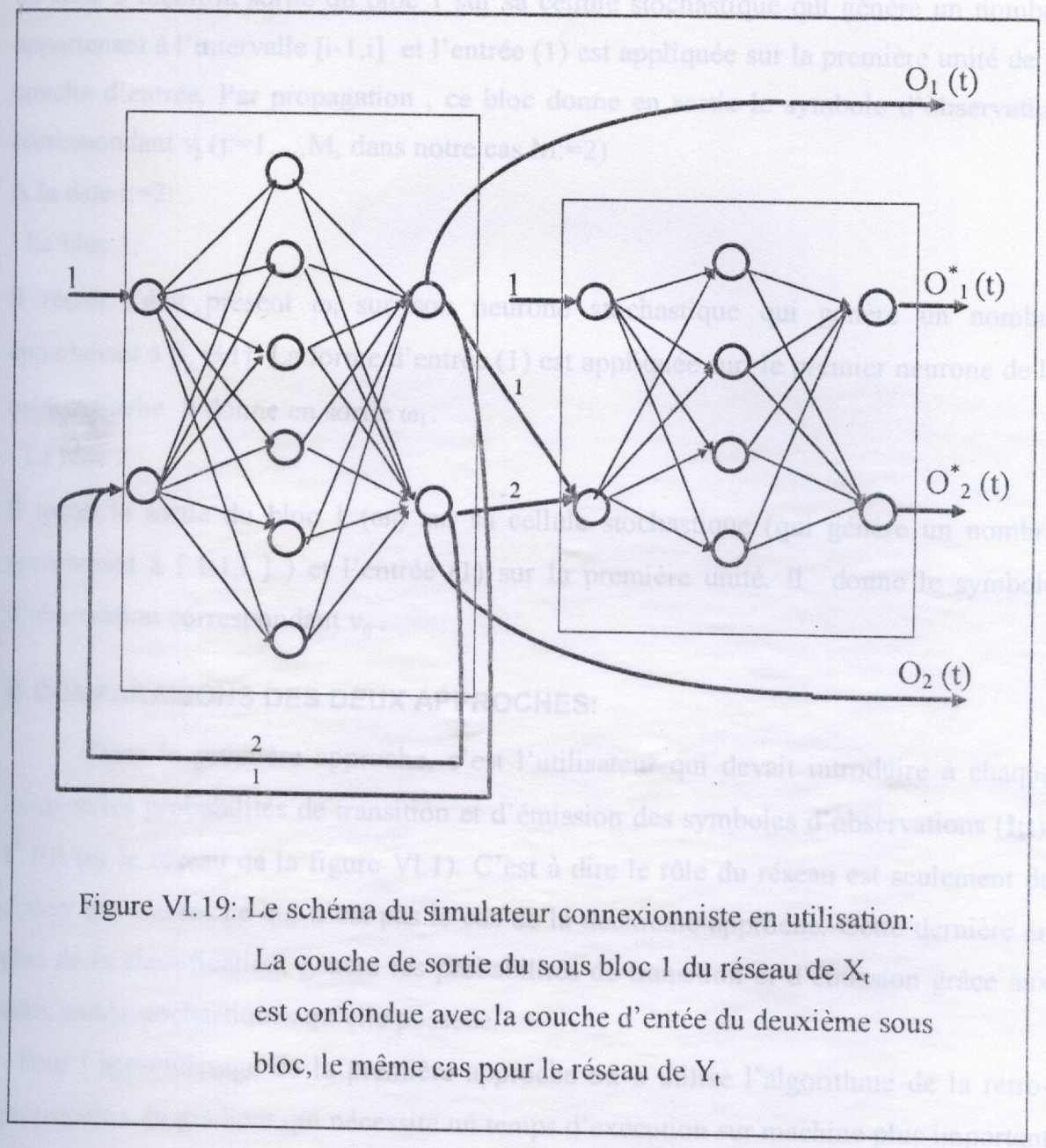


Figure VI.19: Le schéma du simulateur connexionniste en utilisation:
La couche de sortie du sous bloc 1 du réseau de X_t est confondue avec la couche d'entée du deuxième sous bloc, le même cas pour le réseau de Y_t

A la date t:=1

- Le réseau de X_t

Initialement les sorties du bloc 1 sont nulles (0,0). Ses dernières sont appliquées sur le neurone stochastique qui génère un nombre appartenant à l'intervalle [0,1].. La forme d'entrée (1) est appliquée sur le premier neurone de la couche d'entrée. Par propagation, ce bloc donne l'état initial ω_i (i:=1,...,N, dans notre cas N:=2).

- Le bloc 2:

Le bloc 2 reçoit la sortie du bloc 1 sur sa cellule stochastique qui génère un nombre appartenant à l'intervalle [i-1,i] et l'entrée (1) est appliquée sur la première unité de la couche d'entrée. Par propagation , ce bloc donne en sortie le symbole d'observation correspondant v_j (j:=1,...,M, dans notre cas M:=2)

A la date t:=2:

- Le bloc 1:

Il reçoit l'état présent ω_i sur son neurone stochastique qui génère un nombre appartenant à [i, i+1]. La forme d'entrée (1) est appliquée sur le premier neurone de la même couche. Il donne en sortie ω_1.

- Le bloc 2

Il reçoit la sortie du bloc 1 (ω_1) sur sa cellule stochastique (qui génère un nombre appartenant à [l-1,l]) et l'entrée (1) sur la première unité. Il donne le symbole d'observation correspondant v_q .

4- COMPARAISONS DES DEUX APPROCHES:

-Dans la première approche, c'est l'utilisateur qui devait introduire à chaque itération les probabilités de transition et d'émission des symboles d'observations (I(t), I^* (t)) sur le réseau de la figure VI.1). C'est à dire le rôle du réseau est seulement de classer ces entrées ce qui n'est pas le cas de la deuxième approche. Cette dernière en plus de la classification, génère les probabilités de transition et d'émission grâce aux deux unités stochastiques qu'elle possède.

- Pour l'apprentissage de la première approche on a utilisé l'algorithme de la rétro-propagation du gradient qui nécessite un temps d'exécution sur machine plus important que celui de la règle du perceptron utilisée pour la deuxième approche.

- Les sorties de la première approche (O(t) et $O^*(t)$) ne sont pas exactes (elle sont obtenues à ε prés), ceci à cause de la fonction d'activation utilisée (la fonction sigmoïde). Pour cette raison, on ne peut pas générer des séquences (d'états et d'observations) supérieures à 100 avec $\varepsilon \leq 0.5$. Dans la deuxième approche la fonction d'activation est une fonction à seuil, c'est à dire que les réponses du réseau

sont exactes (1 ou 0), ce ci nous permet de générer des séquences infinies sans erreurs à condition que la règle du perceptron converge.

CONCLUSION GENERALE

Notre premier souci à été de clarifier quelques notions du vocabulaire relatif aux réseaux de neurones artificiels et de restreindre notre intérêt aux modèles utilisés pour le traitement des tâches de nature séquentielle. La formulation de ce type de tâches est basée sur le modèle de la mémoire à court-terme qui est souvent difficile à implanter dans un réseau de neurones. Les différents travaux exposés dans ce mémoire montrent que les mémoires à court-terme à retard et exponentielle avec $\mu_k=0$ sont en général plus au moins facile à construire.

La relation exposée dans le chapitre III sur le processus de Markov à temps continu et le réseau MADALINE est un résultat important pour les applications de fiabilité et ceci gârce à la simplicité de l'architecture et de l'apprentissage de ce reseau. Ce résultat peut éventuellement être généralisé pour l'estimation des taux instantanés de transition d'un modèle de Markov à temps continu homogène, à condition d'avoir des exemples d'apprentissage constitués des valeurs des probabilités $P_i(t)$ à des instants donnés.

Le réseau proposé dans le chapitre VI sous le titre modélisation connixioniste de la méthode de simulation est une approche raisonnable, et cela grâce à la simplicité de la méthode de l'apprentissage par la règle du perceptron et aux liens qui existent entre les poids des connexions des sous-bloc 1 et 3 respectivement et les probabilités respectivement de transitions (A_0 ,A) et d'émissions (B). Pour généraliser cette étude, il faut montrer que les classifications (avec les mêmes codages) effectuées par les sous-bloc 2 et 4 sont linéairement séparables pour N et M quelconques et considérer aussi les cas où certains probabilités (a_{ij} ou a_{0i} et $b_j(k)$ pour $1\leq i,j\leq N$; $1\leq k\leq M$) sont nulles.

REFERENCES BIBLIOGRAPHIQUES

[1] Aissani A.(1992), Modéles stochastiques de la thèorie de fiabilité, OPU, Alger.

[2] Aissani A. (à apparaître). Méthode de simulation statistique, OPU, Alger.

[3] Asselin J. P. (1993). Les réseaux de neurones artificiels, cours de D.E.A, Ecole d'ing. en infor. pour l'industrie (E3i), Univ. de Tours , France.

[4] Baum L. E. & Eagon J. A. (1967), An inequality with applications to statistical estimation for probabilistic function of Markov processes and to a model for ecology. Bul. Amer. Math. Soc. 73, pp: 360 - 363.

[5] Baum L. E. , Petrie T., Soules G. & Weiss N. (1970), A maximisation technique occurring in the statistical analysis of probabilistic functions of Markov chaïns, Ann. Math. Stat., 41 (1) , pp: 164 - 171.

[6] Baum L. E. (1972). An inequality and associated maximisation technique in statistical estimation for probabilistic functions of Markov processes Inequalities, 3 ,pp: 1 - 8.

[7] Beicht F. & Franken P. (1983). Reliability and maintenance: Mathematical approch, VERLAG. Technik. Berlin.

[8] Benmiloud B. & Pieczynski W. (1994), Estimation des paramétres dans les chaines de Markov cachées et segmentation d'images, Rap. tech. int, Depart. Signal et Image, Inst. Nat. des télécommunications, France.

[9] Bone R. (1994), Réseaux connexionnistes et phénomènes temporels, Rap. tech. inte., E3i, Univ de Tours, France.

[10] Bottou L. (1991). Une approche théorique de l'apprentissage connexionniste : Application à la reconnaissance de la parole. Thèse Doctorat, Univ. de Paris Sud, Centre d'Orsay.

[11] Boudelal A. & Zahali M. (1993), Influence de la fiabilité sur les décisions relative à la production. Mém. d'ing. en math. appliquées, Univ. de Blida

[12] Boulard H. & Wellekens C. J. (1990). Links between Markov models and multirayer perceptron I.E.E.E. Trans. Pattern., Anal. & Machine Intell, Vol 12. N°12, pp: 1167 - 1178.

[13] Boulard H., Morgan N. & Renals S. (1992) Neural nets and hidden Markov models : Review and generalisations. Speech communication 11, pp. 237 - 246 North - Holland.

[14] Brian G. L. (1992). Maximum - Likelihood estimation for hidden Markov models, stochastic processes and their applications 40, pp: 127 - 143 North - Holland

[15] Corttrell M., Girard B., Girragd Y., Mangeas M., Muller C. (1993). Neural modeling for time series : A statistical stepwise Method for weight élimination, Rap. tech. inte., Centre de recherche SAMOS, Univ. PARIS I.

[16] Curtiss P., Kreider J. & Brandemuchl M. (1992). Adaptive control of HVAC processes using predictive neural networks (Tehnical Report). Boulder, co: Univ of Colorado, Joint centre for energy management.

[17] De Vries B. & Principe J. C. (1991). A theory for neural networks with time delays. In Lippmann R. P., Moody J.& Touretzky D. S. (EDS), Advances in formation processing systems, pp. 162 - 168, San Mateo, CA : Morgan Kaufmann.

[18] De vries B. & Principe J. C. (1992). The Gamma model - A new neural net model for temporal processing. Neural networks 5 pp. 565-576.

[19] Elman J. L. (1988) Finding structure in time. Center for reseach in language, Technical Report 8801, University of California, San Diego.

[20] Elman J. L. (1990) Finding structure in time, Cognitive Science 14, pp: 179 - 212.

[21] Esary J.D. & Proshan F. (1970), A reliability bound for systems of maintenened interdependent components, Amerc journal. stat. Assc. 65, pp:328-338.

[22] Faure R. (1979), Précis de recherche operationnelle, Edition Dounod, pp: 111-126

[23] Folgerman F. & Soulie L. (1987), Le connexionnisme, Support de cours,

[24] Forney G. D. (1973), The Viterbi algorithm. Proc. IEEE, pages 268 - 278, MARI 87, Cognitiva.

[25] Jordan M. I. (1986), Serial order : Parallel distributed processing approach, I C S. Report 8604, Univ. of California, San Diego 92093.

[26] Krioule A. (1990). La reconnaissance automatique de la parole et les modèles markoviens cachés, Thèse Doctorat univ. de nancy , Francel.

[27] Lapedes A. & Farber R. (1987), Non linear signal proceesing using neural Networks Report N°: LA - UR - 87 -2662, Los Alamos, MN : Los Alamos Laboratory.

[28] Lawrencer R. Rabiner L. R.(1988), A tutorial on hidden Markov models and selected applications in Speech Recognition , IEEE Log Number 8825949.

[29] Le Cun Y. (1985), Une procedure d'apprentissage pour les réseaux à seuil asymetrique, Proceeding of Cgnitiva 85, pp: 599-604 Paris.

[30] Le Cun Y. (1987), Modeles connexionnistes de l'apprentissage, Thése doctorat, Univ. de Paris VI.

[31] Levinson S. E., Rabiner L. R. & Sondh M. M. (1983), An introduction to an application of the theory of probabilistic functions of Markov process to automatic Speech Recognition , Bell. Systems, Technical Journal, Vol.: 62, N° 4, pp: 1035 - 1074.

[32] Manzoul M.A. & Mamoun S. (1990), Neural network for reliablity analysis of simplex systems, Microelctron reliab. Vol. 30 N°4 , pp:795-800.

[33] Manzoul M.A. & Mamoun S. (1990), Neural analysis of T-R-M systems on neural networks, Microelctron reliab. Vol. 30 N°4 , pp:801-806.

[34] Mc Culloch W. S. & Pitts W. (1943) a logical calculus of ideas immanent in nervous activity. Bulletin of Mathématical biophysics 5,pp: 115 - 133.

[35] Minsky M. & Papert S.(1969) Perceptrons. Cambridge, MA : Mit Press

[36] Mozer M.C.(1989). Afocused back-propagation algorithm for temporal pattern recognition. Complex System 3, pp: 349 - 381

[37] Mozer M.C. (1993) Neural Net architectures for temporal sequence processing Technical Report. Bouldes, Co 80309 - 0403 Depertment of computer science of institute of cognitive science, University of Colorado.

[38] Pages A. & Goudram M. (1979), Fiabilité des systemes, Edition Eyrolles.

[39] Robinson A. J. & Fallside F. (1987), The utility driven dynamic error propagation network, Tech. Report GUED /EINFENG / TR Cambridge University, Dept. of Engineering.

[40] Rosenblatt F. (1957), The perceptron : a probabilistic model for information storage and organisation in the brain, Psychological Review 65, pp: 386 - 408

[41] Rosenblatt F. (1962), Principes et neurodynamiques New York Spartan.

[42] Rumelhart D. E., Hinton G.E. & Williams, R.J. (1986) Learning representations by back-propagating error, Nature 323, pp:533 - 536.

[43] Saadoune H. & Hammadi A. (1992), Intégration d'une usine de production: aspect maintenance, mém. d'ing. en math. appliquées, Univ. de Blida.

[44] Schmidhuber J. (1992), A fixed size storage $O(n^3)$ time comlexity learning algorithm for fully reccurent continuatly running networks, Neural computation 4, pp:243-248.

[45] Sejnouski T., Rosenberg C. (1986), NETlalk : a parallel network that learns to read aloud , The Johns Hopkins University, Electrical Engineering and computer. Science, Technical Report JHU / EECS - 86 / 01/ 32.

[46] Stoyan D. (1983), Comparaison methods for queuing models and other stochastic models, Wiley

[47] Sylvie T. (1989), L'apprentissage supervisé dans les modèles connexionnistes, Thèse Doctorat, Spécialité Informatique ; Univ Rene Décartes Paris V.

[48] Touzet C. & Giambiasi N. (1988) Reconnaissance de séquences par des réseaux de neurones, Neuro-Naimes 88, Nimes, France.

[49] Touzet C., (1990). Contribution à l'étude et au développement de modèles connexionnistes séquentiels. Thèse Doctorat, Spécialité : Composants, Signaux et Systèmes. Université Montpellier II, France.

[50] Widrow G. & Hoff M. E. (1960) Adaptive switching circuits institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4, pp: 96 - 104.

[51] WU. C.F.J., (1983), On the convergence properties of the EM algorithm. The Annals of Probability, 11 (1), pp : 95 - 103.