Les Réseaux de Neurones Récurrents

Yann MORERE

Septembre 1998

Résumé

Y'a pas résumé t'as qu'a lire le bouquin...

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

État de l'art sur les Réseaux de Neurones Récurrents

1.1 La genèse des Réseaux de Neurones Récurrents

1.1.1 Introduction

Le besoin de développement d'architectures nouvelles de réseau se faisait sentir face aux limites des réseaux à couches d'autres architectures de réseaux. La volonté d'incorporer explicitement de informations connues a priori dans l'architecture. La recherche sur la stabilité des réseaux de neurones ainsi que la vitesse de convergence des algorithmes d'apprentissage concourent aussi à cette volonté de développer de nouvelles architectures.

Parmi ces architectures on trouve une classe très générale de réseaux de neurones : **les réseaux récurrents**.

Les réseaux récurrents ou dynamiques ou feedback sont caractérisés par des connexions bouclées. Ils permettent alors la modélisation de système à mémoire interne ainsi que des systèmes variants dans le temps. Ainsi grâce à sa mémoire adaptable il a la capacité d'implémenter directement des systèmes dynamiques.

Ces réseaux récurrents sont caractérisés par des cycles si on les représente sous forme de graphes. Ceci introduit une dimension temporelle dans son comportement. Ils se caractérisent par deux types de comportement dynamique :

- Une dynamique autonome convergente (pour des entrées fixées)
- Une dynamique non-autonome non-convergente (pour des entrées variantes dans le temps)

Le réseau de Hopfield et la machine de Boltzman rentrent dans la première catégorie. Les réseaux de la deuxième catégorie utilisent le contexte courant et leurs entrées variantes dans le temps pour évoluer suivant leur dynamique.

1.1.2 Le Réseau de Hopfield : premier réseau récurrent

C'est le premier exemple remarquable de réseau récurrent.

En 1982, John Hopfield publie un article qui va relancer les réseaux de neurones. Selon lui le système nerveux recherche les états stables et attracteurs de son espace d'état. Les états voisins tentent de se rapprocher de ces états stables. Il propose donc un modèle nouveau. Ce modèle est basé sur un réseau de neurone type MacCulloch et Pitts avec tous les neurones connectés entre eux (mais il n'y pas d'auto-connexion). Les entrées (à

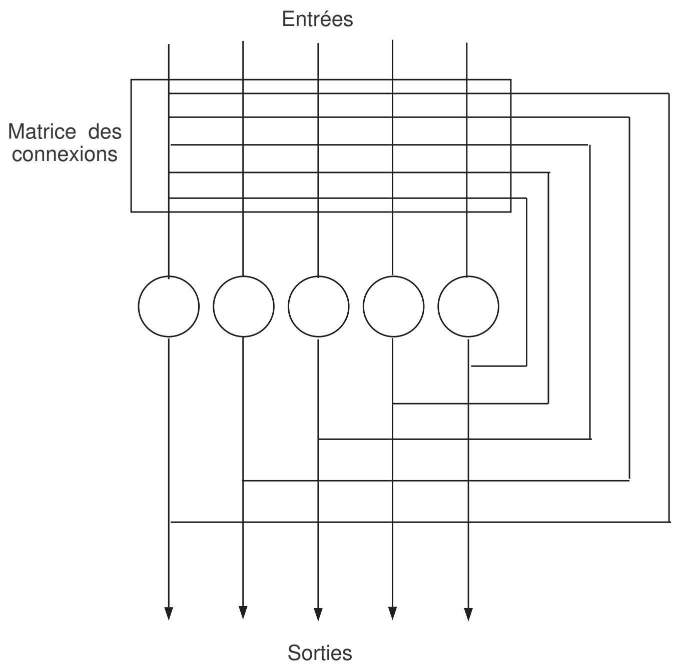

FIG. 1.1 – Représentation d'un réseau de Hopfield

l'origine binaire sont multivaluées). La règle d'apprentissage est la règle de Hebb. L'autre apport est la comparaison avec la physique en ce qui concerne les verres de Spin : on peut constaté une analogie entre la relaxation du réseau de neurones vers un état stable et les verres de Spin d'Ising, dont la fonction d'énergie décroît vers un minimum local.

Le réseau est une fonction dynamique supposée séquencée par une "horloge". Chaque neurone modifie son état à un instant aléatoire de façon asynchrone par rapport aux autres. Une formulation consiste à dire qu'à chaque top d'horloge on tire un neurone au hasard et on évalue son état.

Il y a deux méthodes de mise à jour

- Dynamique synchrone : tous les neurones sont mis à jour simultanément
- Dynamique asynchrone séquentielle : mise à jour l'un après l'autre dans un ordre défini.

Le modèle du neurone est le même que celui du perceptron. On constate donc que le réseau de Hopfield est récurrent et de plus il est biologiquement plausible. On peut représenter ce réseau de Hopfield par le structure suivante (Cf. figure1.1:

Il y a donc rebouclage du réseau sur lui-même. Hopfield a pu exprimer une fonction de Lyapunov pour de telle structure de réseau de neurones. Cette fonction assure la stabilité du système grâce à l'égalité suivante : . La fonction de Lyapunov est une entité mathématique utilisée pour l'étude de l'équilibre des systèmes complexes. Une fonction est dire de Lyapunov si elle admet un extremum pour chaque position d'équilibre du système. L'étude analytique de la fonction de Lyapunov permet de caractériser chaque position de cet équilibre (instable, asymptotiquement stable, stable, oscillatoire...).

Dans le cas de Hopfield, ce type de réseau est intéressant pour le fonctionnement dynamique ce qui lui confère une capacité de généralisation impressionnante. Quand on lui applique de nouvelles entrées, le réseau évolue vers l'état d'équilibre le plus proche au sens de Lyapunov. L'évolution dynamique lui confère une meilleure généralisation.

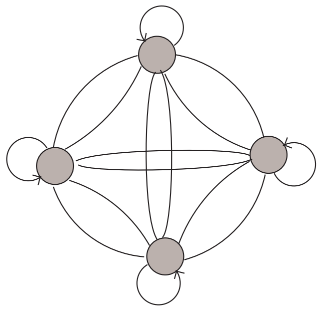

FIG. 1.2 – Réseau complètement interconnecté récurrent

Ce modèle est relativement différent des modèles à couches. L'apprentissage est statique mais la relaxation du réseau est dynamique c-à-d que le réseau peut effectuer un certain nombre d'itérations avant de trouver un état stable.

1.1.3 Utilisation dans les problèmes d'optimisation

1.1.3.1 Recuit simulé et réseau de Hopfield

Les limites Les états poubelles peuvent être de très fort attracteurs : Mais ils ne correspondent pas à un état désiré.
La solution consiste en une modification des poids afin d'éviter ces états poubelles. Le désapprentissage grâce à l'application de la règle de Hebb en sens inverse permet d'éviter ces états poubelles.
L'oubli catastrophique consiste en l'oubli des états attracteurs déjà appris lorsque l'on essaie en apprendre un nouveau. A ce moment on a dépassé la capacité du réseau. (Capacité max. = rapport entre exemples à apprendre et nombre de neurones = 0.14)

Mais cette une architecture très délicate à mettre en oeuvre car l'apprentissage est beaucoup plus difficile que dans le cas du perceptron. Ceci est du au fait du nombre très grand des minima locaux engendrés par le nombre des connexions.

La machine de Boltzman est un réseau de Hopfield dans lequel l'apprentissage se fait par un algorithme de recherche aléatoire de minimum d'énergie : **Le recuit simulé**.

1.2 Les réseaux statiques récurrents ou réseaux bouclés

Ils ne sont plus nécessairement constitués en couches. L'introduction de bouclage dans leur structure rend plus difficile l'obtention d'un point d'équilibre (Cf. figure 1.2).

Ce type de réseau est similaire au réseau de Hopfield et il comporte plusieurs type d'adaptation de ses neurones : Adaptation asynchrone, synchrone, continue ou tous les neurones sont adaptés continuellement et simultanément.

La réponse d'un neurone i à l'équilibre est donnée par l'équation :

$$V_i = f(S_i) = f\left(\sum_{j=0}^{N} \omega_{ij} V_j\right)$$

1.2.1 Algorithme d'apprentissage utilisé

Dans le cas de la machine de Bolzman il s'agit du **recuit simulé**.

1.2.1.1 La rétropropagation récurrente

A la suite de la diffusion de l'algorithme de rétropropagation pour les réseau feedforward, nombreux sont les chercheurs qui ont essayé d'explorer la possibilité d'étendre cet algorithme aux cas des réseaux avec des connexions récurrentes. Une généralisation de la rétropropagation pour les réseaux récurrent avec séquences de relaxation ou chaque neurones suit les équations $\tau_i \dot{y}_{t,i} + y_{t,i} = f\left(a_{t,i} + I_{t,i}\right)$ avec $a_{t,i} = \sum_j \omega_{ij} y_{t,j}$ et $f()$ est une fonction du type tanh.

Pineda a découvert que l'on pouvait appliquer directement l'algorithme de rétropropagation aux réseaux de neurone statiques récurrents

Leurs principales applications concernent l'implémentation de mémoire associative, la résolution de problème contraint.

1.3 Les Réseaux Temporels récurrents

La plupart des applications utilisent les réseaux de neurones non-bouclés (non-récurrents). Alors lorsque l'on a besoin de mémoire interne on contourne et on utilise en entrée un registre à décalage qui se comporte comme une mémoire externe. Ainsi même des réseaux avec des connexions retardées (Time delay neural network Lang 88) le réseau FIR (réponse impulsionnelle finie) ne peuvent capturer que des informations durant un temps limité. Ils ont donc une mémoire à cours terme. Cela ne leur permet pas de prendre en compte des phénomènes temporels de longues échéances.

Une attention particulière a été apportées au réseau dédiés aux traitement des séquences temporelles : il en résulte ce que l'on appelle les réseaux temporels récurrents. Les signaux récurrents sont retardés et permettent ainsi la mémorisation et le rappel de l'état antécédent .Ils peuvent ainsi détecter et stocker des corrélations entre des entrées successives dans le temps.

1.3.1 Les réseaux de Jordan et de Elman

Ils portent aussi le nom de Time Delay Récurrent Neurals Networks du fait de leurs connexions retardées. Les plus connus sont les réseaux de Jordan et de Elman

1.3.1.1 Le réseau de Jordan

C'est une classe simple de réseaux récurrents qui utilise un vecteur d'état qui contient les copies des activations de la couche de sortie du réseau à l'instant précédent (Cf. figure

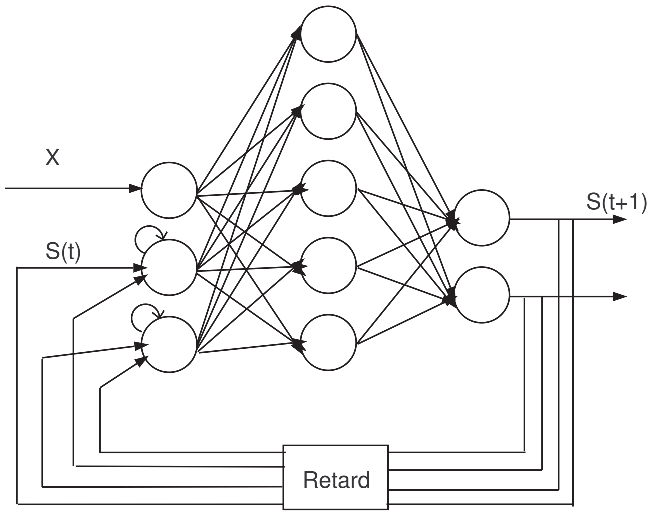

FIG. 1.3 – Exemple de Réseau de Jordan

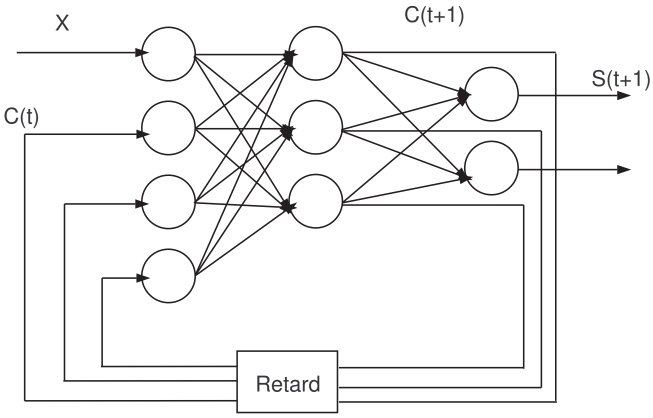

FIG. 1.4 – Exemple de Réseau de Elman

1.3). Les connexions de ce vecteur d'état sont pondérées dans le sens Vecteur d'état → couche de sortie. Les connexions du vecteur d'état vers la couche cachée sont pondérées. On peut comparer ce réseau à un réseau de type feedforward dans lequel on ramène les sorties du réseau avec un retard afin de générer une séquence temporelle. Il s'agit donc d'une classe réduite de réseaux récurrents. Le principal désavantage de ce type de réseau réside dans le fait que l'état qui doit être retenu par le réseau doit se manifester dans le calcul des sorties du réseau. La représentation de nouvelles structures temporelles ne peut pas être crée.

L'apprentissage de tel réseau est assuré par un apprentissage supervisé.

1.3.1.2 Le Réseau de Elman

Ce type de réseau est basé sur la même architecture que le réseau de Jordan, mais Elman lui utilise les copies des activations de la couches cachées à l'instant précédent pour le vecteur d'état.

Dans ces deux types de réseaux le vecteurs d'état permet au réseau de stocker des

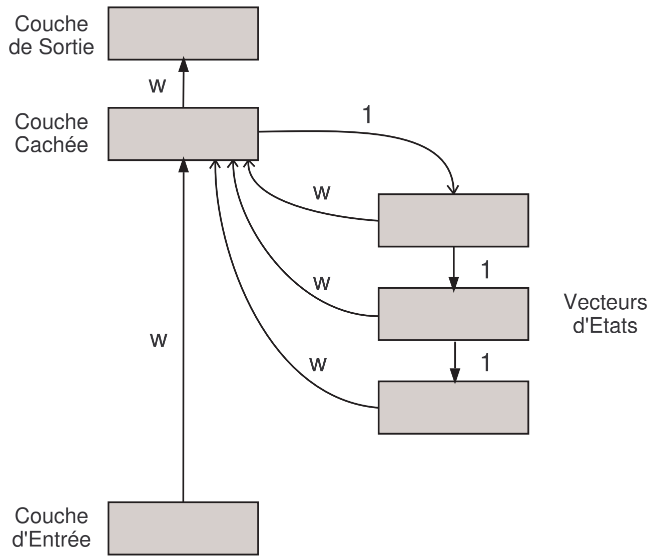

Fig. 1.5 – Exemple de tour de Elman

informations sur les entrées précédentes. C'est pourquoi ils sont utilisés dans la prédiction de séries temporelles, les tâches de classification, l'apprentissage de scénarii. Par rapport au réseau de type feedforward qui travaille avec des fenêtres temporelles, ils ont l'avantage de tenir compte implicitement du contexte temporel.

Pour l'apprentissage de son réseau Elman a développé une version de la rétropropagation dans le temps en ne conservant que une ou deux coups d'échantillonnage. Cette réduction rend l'algorithme direct.

Ces réseaux sont utilisés pour l'apprentissage de structure séquentielle [Wilson 95]. Les réseaux de Elman sont utilisés pour l'apprentissage de structure grammaticale d'un ensemble de phrases générées aléatoirement.

1.3.1.3 Les tours de Jordan et de Elman

Par la suite afin d'augmenter les performances de leurs réseaux Elman et Jordan ont décidé d'intégrer plusieurs vecteurs d'états à l'intérieur de leurs réseaux : on parle alors de Tour de Jordan (Cf. figure 1.6) et de Tour de Elman (Cf. figure 1.5). L'idée étant en multipliant les vecteurs d'états de stocker un maximum d'informations sur les états précédents. En effet il s'avère qu'avec le même nombre de poids (connexions) mais avec un nombre de vecteurs d'états plus grand le réseau de neurones apprend plus vite.

Les limites Les réseaux de Jordan et Elman ont une récurrence restreinte complètement déterminée par la structure. Le gros problème de cette structure est qu'elle ne peut tenir compte des structures temporelles évolutives. En effet les retards sont fixés dans l'architecture du réseau et la part des données réutilisées en entrée est fixée par le concepteur. Ceci peut être tout à fait convenable si on a une connaissance a priori du problème à résoudre.

Problème du réseau de Jordan : son apprentissage est supervisé et d'après sa structure les récurrences apparaissent sur la couche de sortie. Si on supervise ces sorties la récurrence est orientée et on obtient en fait un réseau feedforward avec une entrée en plus qui est

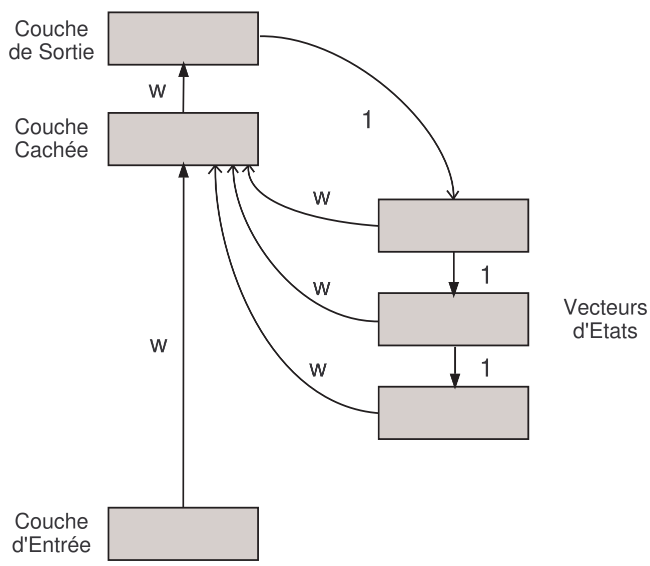

FIG. 1.6 – Exemple de tour de Jordan

la sortie du réseau retardée. C'est un TDNN (Time Delay Neural Network). Ainsi la représentation interne temporelle ne peut être représentée car le réseau ne tient compte que des sorties.

Dans les réseaux récurrents en général on passe outre cette contrainte et ils sont alors capables de capter la séquence temporelle caractéristique de la fonction à apprendre.

1.3.2 Les algorithmes d'apprentissage associés

1.3.2.1 Algorithmes d'apprentissage du point fixe

Dans ces algorithmes on suppose que le réseau converge vers des points fixes stables. Cette convergence vers des points fixes peut être très intéressante pour les problèmes contraints et les tâches de mémoires associatives. Le problème est que les réseaux récurrents ne converge pas tous vers ces points fixes. Mais il existe un certain nombre de cas qui garantissent une convergence :

Des conditions linéaires sur les poids garantissent que la fonction de Lyapunov :

$$L = -\sum_{i,j} \omega_{ij} y_i y_j + \sum_j (y_i \log y_i + (1 - y_i)\log(1 - y_i))$$

décroît jusqu'à un point fixe.

Atiya a montré qu'un point fixe unique est atteint en fonction des conditions initiales si $\sum_{ij} \omega_{ij}^2 \leq \max(\sigma')$. D'autres études empiriques indiquent qu'appliquer des algorithmes du point fixe stabilisent les réseaux en les forçant à un comportement asymptotique. Mais il n'y a pas encore de démonstration de cette remarque.

Problème avec les algorithmes du point fixe : Même si la convergence vers un point fixe est assurée, ces algorithmes d'apprentissage peuvent rencontrer des difficultés. La procédure d'apprentissage calcule la dérivée de la somme des erreurs mesurées par rapport aux paramètres internes du réseau. Ce gradient est utilisé dans une procédure d'optimisation, typiquement une variante de la descente du gradient afin de minimiser une erreur.

De telles procédures d'optimisation suppose que la surface d'erreur générée par les différents calculs est continue, et donc si cette supposition n'est pas vérifiée alors l'algorithme échoue.

1.3.2.2 La Rétropropagation récurrente

Pineda et Almeda ont découvert que l'algorithme de rétropropagation de l'erreur est un cas particulier d'une procédure de calcul du gradient de l'erreur plus général. Les équations sont les suivantes

$$y_i = \sigma(x_i) + I_i \tag{1.1}$$

$$z_i = \sigma(x_i) \sum_j \omega_{ij} z_i + e_i \tag{1.2}$$

$$\frac{\partial E}{\partial \omega_{ij}} = y_i z_i \tag{1.3}$$

où z_i est la dérivée partielle de E par rapport à y_i, E est une erreur métrique sur $y(t_\infty)$ et $e_i = \frac{\partial E}{\partial y_i(t_\infty}$ est la dérivée première de E par rapport à l'état final d'un neurone.

Une solution pour calculer un point fixe est de calculer : $0 = -y_i + \sigma(x_i) + I_i$ et le point fixe sera atteint lorsque $\frac{dy_i}{dt} = 0$ d'où $k\frac{dy_i}{dt} = -y_i + \sigma(x_i) + I_i$. En considérant l'équation 1.2 on obtient un solution pour $\frac{dz_i}{dt} = -z_i + \sigma'(x_i) \sum_j \omega_{ij} z_j + e_i$.

Ces équations peuvent être implémentée directement en supposant que le temps passé au calcul est négligeable par rapport au temps passé au point fixe et que le taux de changement de poids η est lent comparé à la vitesse de présentation des échantillons d'apprentissage. Les poids seront donc mis à jour par une équation du type $\frac{d\omega_{ij}}{dt} = -\eta \frac{\partial E}{\partial \omega_{ij}} = -\eta y_i z_j$ où s'il on a introduit un terme de biais $0 \leq \alpha \leq 1$ on a $\frac{d^2\omega_{ij}}{dt^2} + (1-\alpha)\frac{d\omega_{ij}}{dt} + \eta y_i z_j = 0$.

1.3.2.3 La rétropropagation dans le temps ou dépliage dans le temps

Les algorithmes d'apprentissage du point fixe tels la rétropropagation du gradient de l'erreur sont incapables de capter les attracteurs non-fixes ou de produire un comportement temporel désiré sur un intervalle borné. Il faut donc adapté les procédures pour ces situations de points non-fixes. On doit donc minimiser une erreur de trajectoire de fonction $E(y)$ entre t_0 et t_1. Donc $E = \int_{t_0}^{t_1} (y_0(t) - f(t))^2 \, dt$ mesure la déviation de y_0 par rapport à f. En minimisant E on tend à rapprocher y_0 de f. On va donc utiliser une technique de calcul de $\frac{\partial E(y)}{\partial \omega_{ij}}$ dérivée qui nous permettra d'utiliser la descente de gradient afin de minimiser E.

Afin de mener à bien cette opération il faut conserver en mémoire toutes les activations des neurones au cours du temps dans la passe de propagation avant, afin de pouvoir utiliser l'algorithme de rétropropagation du gradient dans la phase propagation arrière.

Dans cette approche simple on opère un dépliage temporel du réseau continu en un réseau échantillonné avec une valeur d'échantillonnage Δt et on applique la rétropropagation usuelle sur ce réseau discrétisé. En prenant assez petit on conserve un apprentissage quasi-continu dans le temps. Pratiquement on va donc dupliquer les neurones T fois pour une séquences temporelles T. On obtiendra ainsi un réseau déplié dans le temps qui duplique les connexions au cours du temps ce qui lui permet de garder en mémoire l'évolution. Ce réseau résultant possède une structure multicouche c'est pourquoi on peut utiliser la rétropropagation comme dans un réseau non bouclé.

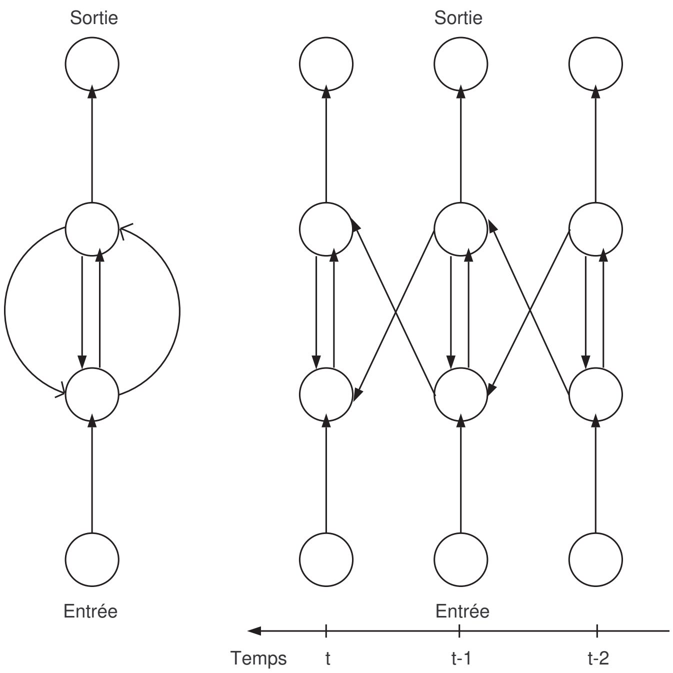

FIG. 1.7 – Dépliage temporel d'un réseau récurrent simple

Cet algorithme est local dans l'espace (les calculs de gradient sont locaux sur un échantillon E/S) mais il n'est pas local dans le temps dès que l'on doit stocker toutes les valeurs passées des activations. Le plus gros problème dans l'implémentation d'un tel algorithme est les ressources qu'il exige. Le complexité spatiale et surtout temporelle qui résulte de la duplication des neurones (Cf. figure 1.7) en fait un algorithme quasi-inutilisable. Dès que les séquences temporelles deviennent longues il devient impossible à utiliser du fait de l'explosion combinatoire.

1.3.2.4 Apprentissage temporel récurrent en temps réel (Real Time Recurrent Learning RTRL)

Développé par Williams et Zipser C'est un autre algorithme pour l'apprentissage des réseaux de neurones récurrents sans restriction sur l'architecture, qui évite la rétropropagation dans le temps. Il opère de façon récursive en gardant en mémoire pendant la passe de propagation avant les dérivées partielles qui indiquent comment évoluent chaque poids du réseau.

La mise à jour des poids se fait en ligne à chaque itération sans attendre la fin de la séquence. Il n'y donc plus de contrainte sur la durée de la séquence et donc la complexité temporelle est indépendante de la durée de la séquence. Afin de pouvoir faire cette adaptation en ligne à chaque couple E/S, on doit calculer le gradient sur la fonction d'erreur mais en local par rapport au paramètre du système $\{w_{ij}\}$ et le stocker.

Par linéarité le gradient se décompose dans le temps comme la somme des gradients à chaque instant soit

$$\Delta\omega_{ij}(t) = -\eta\frac{\partial E(t)}{\partial\omega_{ij}} = \eta\sum_k E_k(t)\frac{\partial y_k(t)}{\partial\omega ij}$$

On obtient ainsi une équation itérative en $\frac{\partial y_k}{\partial \omega_{ij}}$ qui permet d'actualiser les dérivées partielles.

C'est un algorithme local dans le temps mais non local dans l'espace du fait du calcul du gradient $\frac{\partial y_k}{\partial \omega_{ij}}$.

Cet algorithme souffre d'une complexité excessive et le nombre d'opérations requises dans le pire des cas est de l'ordre de $O(N^4)$(N est le nombre d'unité du réseau). D'après Bengio et Tsoi cette architecture souffre d'un manque de stabilité et converge difficilement vers une solution optimale.

1.3.3 Problème théorique avec la descente de gradient

Dans la plupart des problèmes les réseaux de neurones récurrents sont bien meilleurs que les réseaux de types feedforward. Mais ils sont aussi plus difficiles à entraîner. Des expériences montrent qu'ils stagnent souvent dans des solutions sous-optimales qui tiennent compte des dépendances à cours termes et non à long terme. D'après Mozer la rétropropagation n'est pas assez efficace pour apprendre de longues séquences temporelles. Une remarque importante au niveau de l'algorithme de rétropropagation est qu'il perd de sa mémoire les événements les plus anciens et qu'il tient compte en grande partie des événements les plus récents (on parle d'évanouissement du gradient). Ceci est du principalement au calcul du gradient.

On peut alors définir trois choses indispensables pour réaliser un bon apprentissage et stockage de séquences temporelles :

- Que le système soit capable de stocker des séquences temporelles de durée arbitraire
- Le système doit être robuste (au bruit)
- Que les temps d'apprentissage ne soit pas trop longs

1.3.4 Nouveaux algorithmes pour l'apprentissage des réseaux récurrents

Nous venons de voir que l'un des principaux problèmes pour l'apprentissage des RNR est le stockage en mémoire interne de séquences temporelles longues. La rétropropagation semble alors une méthode inadéquate.
L'une des solutions afin de contourner le problème est d'utiliser la connaissance a priori du système.

1.3.4.1 Optimisation tempo-pondérée de pseudo-newton

L'algorithme de pseudo-newton à l'avantage de recaler le pas d'apprentissage de chaque poids dynamiquement. L'intérêt de l'ajustement du pas d'apprentissage serait peut-être de résoudre le problème du gradient qui disparaît (pour les événements anciens).
L'algorithme de Pseudo-Newton calcul une approximation diagonale de la matrice Hessienne (dérivée seconde de la fonction de coût par rapport aux paramètres) et ajuste les paramètres d'après la règle suivante $\Delta\omega_i(p) = -\frac{\eta}{\left|\frac{\partial^2 C(p)}{\partial \omega_{ij}^2}\right| + \mu} \times \frac{\partial C(p)}{\partial \omega_i}$ où $\Delta\omega_i(p)$ est la mise à jour pour le poids ω_i après que l'échantillon aie été présenté au réseau et que le coût $C(p)$ aie été calculé. η et μ sont de petites constantes positives. L'algorithme de pseudo-Newton tempo-pondéré est basé sur l'idée de considérer le dépliage temporel du réseau

récurrent et chaque instance d'un poids (a chaque t) comme une variable. Le mise à jour des poids se fait alors d'après l'équation suivante : $\Delta\omega_i(p) = -\frac{\eta}{\left|\frac{\partial^2 C(p)}{\partial\omega_{ij}^2}\right|+\mu} \times \frac{\partial C(p)}{\partial\omega_{it}}$ ou ω_{it} est une instance pour chaque temps t de ω_i. Il permet de meilleures performances si on adapte en ligne μ avec un "momentum".

1.3.4.2 Propagation de l'erreur discretisée

La base du problème pour l'algorithme de rétropropagation vient du fait du stockage de l'information. Le gradient rétropropagé disparaît quand le système est dans un état stable pendant un moment. Afin de bien opérer il faudrait se souvenir de l'erreur au moment ou le systèmes est entré dans cet état stable.

L'idée est donc de développer un algorithme qui propage l'erreur discrétisée à travers les neurones à fonction non dérivable (seuil) du réseau. Le réseau peut être considéré comme une série d'éléments locaux avec chacun une fonction de propagation avant et une fonction d'erreur.

Il en résulte que lorsqu'un élément avec une fonction d'activation non dérivable est auto-bouclé avec un gain > 1 deux points stables sont induits et le pseudo gradient ne disparaît pas avec le temps.

1.3.5 Le réseau de neurones Aléatoires

Ce type de réseaux de neurones statiques récurrents introduit par Galenbe en 1989 est différent de l'approche de MacCulloch & Pitts. Ce réseau est conçu à partir d'un nombre d'unités stochastiques interconnectées entre lesquelles circulent des signaux soit excitateurs (> 0) soit inhibiteurs (< 0).

Chaque unité est associée à un compteur $k_i(t)$ son potentiel à l'instant t qui est incrémenté si l'unité reçoit un signal > 0 décrémenté sinon. Chaque unité évolue au gré des signaux reçus avant d'être remise à zéro lorsqu'elle passe en phase d'émission. La période aléatoire qui sépare l'émission de deux signaux suit une loi stochastique exponentielle. En plus des signaux internes, des signaux externes sont injectés dans le réseau selon un processus de Poisson.

1.3.6 Conclusion

Il y a eu un engouement pour les réseaux de neurones récurrents afin de palier le problème d'apprentissage de séquences temporelles. Un réseau connexionniste dédié aux séquences temporelles doit avoir une représentation interne du passé. C'est dans cette optique et par soucis de concilier complexité et efficacité que différents chercheurs ont développé des modèles d'architecture nouvelles.

1.4 Des architectures récurrentes locales globalement feedforward (Locally recurrent globally feedforward) ou Réseaux LRGF

Plusieurs choses ont motivé l'introduction de nouvelles architectures.

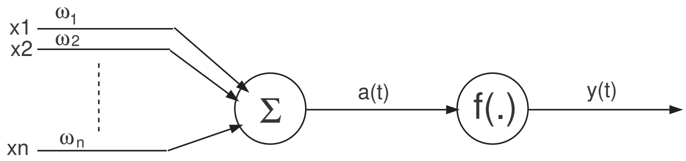

FIG. 1.8 – Architecture générale d'activation

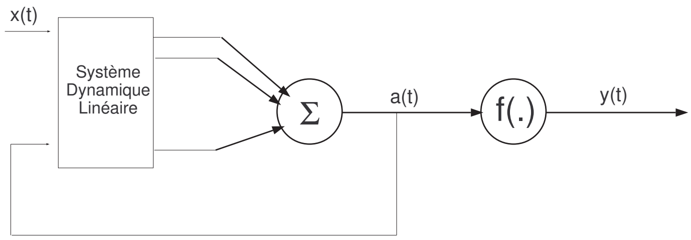

FIG. 1.9 – Architecture avec retour d'information local de l'activation

- Une volonté de trouver d'autres structures que celles de Jordan & Elman, ou encore Wiliams & Zipser
- L'incorporation de connaissances a priori sur le système, comme des informations sur la série temporelle(nombre de retards...)
- Résoudre le problème de stabilité du réseau. En effet l'architecture de type Williams Zipser a des difficultés à se stabiliser autour d'une valeur finie
- Améliorer la vitesse de convergence des algorithmes d'apprentissage

Ces architectures ont une structure globale de réseaux à couches et la récurrence se fait au niveau du neurone.

1.4.1 L'Architecture de Lapedes-Farber

Cette architecture se base sur l'architecture du neurone classique de MacCulloch & Pitts c-à-d $y = f\left(\sum_{i=0}^{n} \omega_i x_i\right)$ ce qui donne $y(t) = f(a(t))$ en considérant le schéma suivant :

l'activation $a(t)$ est une combinaison linéaire des entrées. La sortie du neurone est une fonction non-linéaire $f(\cdot)$ de l'activation $a(t)$ (Cf. figure 1.8). Le retour d'information peut se faire suivant trois méthodes en regard à l'architecture de Lapedes-Farber

Un retour d'information local de l'activation comme le montre le schéma suivant L'entrée peut donc être des versions retardées de l'activation ou une combinaison des entrées et des activations retardées (Cf. figure 1.9).

Un retour d'information local des synapses comme le montre le schéma suivant Ici chaque synapse peut contenir une structure récurrente soit de la forme FIR (Finite Impulse Response) soit IIR (Infinite Impulse Response). On peut noter que le "activation feedback" est un cas particulier du "synapse feedback" car dans ce cas chaque synapse peut être considérée comme une structure individuelle de "local activation feedback" (Cf. figure 1.10).

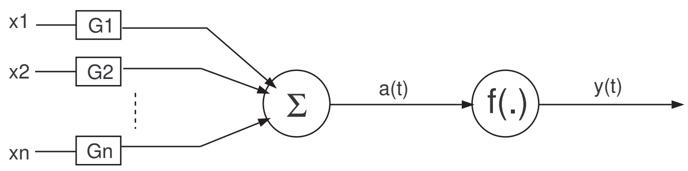

FIG. 1.10 – Architecture avec retour d'information local de s synapses

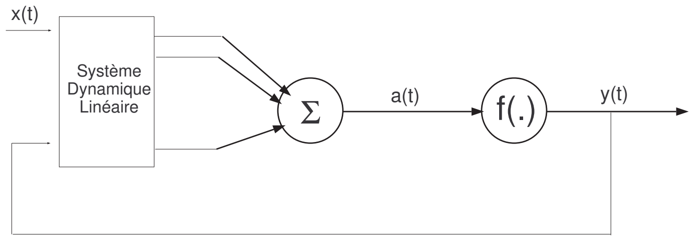

FIG. 1.11 – Architecture avec retour d'information local de la sortie

Un retour d'information local de la sortie comme le montre le schéma suivant Cette structure peut être considérée comme la combinaison des équations $y = f\left(\sum_{i=0}^{n} \omega_i x_i\right)$ et $y(t) = f\left(a(t)\right)$ avec $x_i = y(t-i)$. On observe alors que les entrées sont les sorties retardées de i coups. On peut noter que le retour se fait après la fonction non-linéaire ce qui induit une dynamique différente que dans le premier et le deuxième cas (Cf. figure 1.11).

1.4.2 L'architecture de Back-Tsoi

Ils ont introduit une architecture LRGF munie d'un retour d'information local de la synapse. Mais à la place d'un poids constant, la synapse est caractérisée par une fonction de transfert (avec des pôles et zéros). Ils nomment cette architecture le perceptron multicouche à synapse IIR (Infinite Impulse Response). Le neurone modifié suit alors l'équation suivante

$$y(t) = f\left(\sum_{i=0}^{n} G_i(z-1)x_i(t)\right) \text{ avec } G_i(z^{-1})$$

un fonction de transfert linéaire de la forme

$$G_i(z^{-1}) = \frac{\sum_{j=0}^{m_z} b_j z^{-1}}{\sum_{j=0}^{n_p} a_j z^{-1}}$$

. L'architecture de Lapedes-Farber peut être considérée comme un cas particulier de l'architecture de Back-Tsoi dans laquelle les synapses ont une fonction de transfert linéaire ne contenant que des zéros.

1.4.3 L'Architecture de Frasconi-Gori-Soda

Gori et Al. ont expérimenté une architecture sensiblement différente. En effet dans leurs travaux ils introduisent une architecture avec une retour d'information pris sur

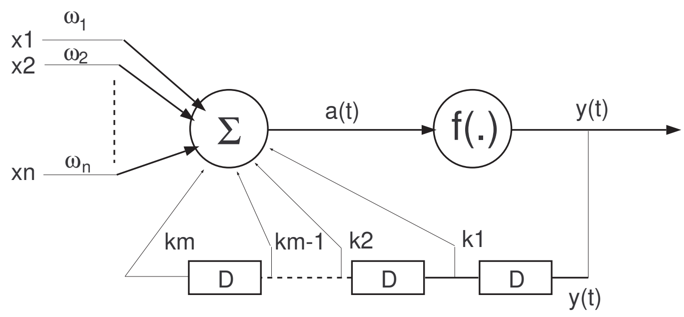

FIG. 1.12 – Architecture de Frasconi-Gori-Soda

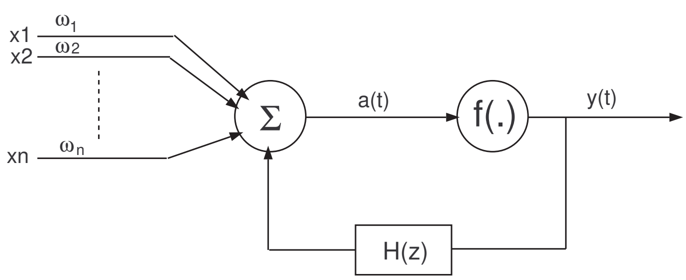

FIG. 1.13 – Architecture de Frasconi-Gori-Soda Généralisée

les sorties de la couche cachée du réseau (Cf. figure 1.12). Ceci est différent de Back-Tsoi en ce sens que les retours sont pris après les non-linéarités. D représente un retard d'une période d'échantillonnage. La sortie suit donc une équation de la forme $y(t) = f\left(k_1 y(t-1)\right) + \sum_{i=0}^{n} \omega_i x_i(t))$ où k_1 est une constante. Ils nomment cette architecture le "local output feedback". On peut aussi voir que c'est une généralisation de l'architecture de Lapedes-Farber dans laquelle ils ont introduit un retour local de la sortie.

En règle plus général la sortie de cette architecture s'écrit :

$$y(t) = f\left(\sum_{j=1}^{m} k_j y(t-j) + \sum_{i=0}^{n} \omega_i x_i(t)\right)$$

où encore plus généralement

$$y(t) = f\left(\sum_{j=1}^{m} k_j y(t-j) + \sum_{i=0}^{n} \sum_{j=0}^{n_d} \omega_{ij} x_i(t-j)\right)$$

On peut reconnaître dans la partie droite de l'équation une structure FIR. On peut alors aisément représenter un modèle plus général de l'architecture de Frasconi-Gori-Soda (Cf. figure 1.13). Une fonction de transfert $H(z)$ qui peut avoir des pôles et des zéros. On peut considérer cette architecture comme une généralisation des architectures de Jordan & Elman dans le fait qu'elle permet de pondérer les retours retardés et de retourner des signaux passées additionnelles (Tours de Elman et Jordan figure 1.5 et 1.6).

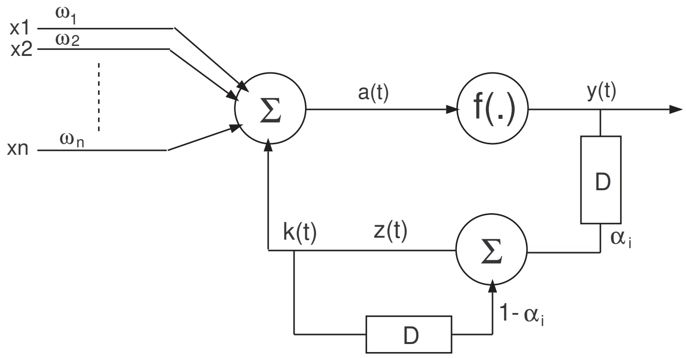

FIG. 1.14 – Architecture de Poddar-Unnikrischnan

1.4.4 Architecture de Vries-Principe

L'architecture de Vries-Principe est basée sur une synapse qui peut être modélisée par un produit de convolution. La structure du neurone se trouve alors modifiée

$$\frac{dx'}{dt} = -ax'(t) + g_0 y(t) + \int_0^1 g(t-s)y(s)\,ds + u(t)$$

où $x'(t)$ est l'état (activation), $u(t)$ l'entrée du neurone, $y(t)$ la sortie du neurone avec $y(t) = f\left(x'(t)\right)$, g_0 est un gain constant et $g(t)$ est le produit de convolution.

Sous certaines conditions le modèle de Vries-Principe et celui de Back-Tsoi sont équivalents. On peut aussi voir qu'il est aussi une généralisation de l'architecture de Frasconi-Gori-Soda.

1.4.5 L'Architecture de Poddar-Unnikrischnan

Cette architecture est basée sur un réseau de type feedforward dans lequel ils ont incorporé des neurones dotés de mémoire. C'est à dire qu'il contiennent des informations sur les activations passées (Cf. figure 1.14). La sortie du neurone se calcul de la façon suivante $y(t) = f\left(\sum_{i=0}^{n} \omega_i x_i(t) + \sum_{i=0}^{n} k_i z_i(t)\right)$ où les z_i sont les sorties des neurones à mémoire de la couche précédente, les k_i sont des constantes. Le neurone à mémoire est défini par $z_i(t) = \alpha y(t-1) + (1-\alpha) z_i(t-1)$ ou α est une constante. D représente un retard

L'architecture de Poddar-Unnikrischnan peut être considérée comme un cas particulier de l'architecture généralisée de Frasconi-Gori-Soda. En effet cette architecture possède un retour sous forme d'une fonction de transfert avec un pôle.

1.4.6 Les Réseaux LRGF sont donc composés de trois Classes

Local Synapse Feedback Network: Dans ce cas l'architecture est identique à un réseau feeforward la seule différence vient de la synapse qui est exprimée en terme de fonction de transfert linéaire.

Cette architecture inclus :
- L'architecture FIR de Lapedes-Farber
- L'architecture IIR de Back-Tsoi
- L'architecture de Vries-Principe

Architecture	Type de Synapse		Position du retour		
	Simple	Dynamique	Synapse	Activation	Sortie
Back-Tsoi	Oui	Oui	Oui	Non	Non
Frasconi-Gori-Soda	Oui	Non	Non	Oui	Oui
De Vries-Principe	Oui	Oui	Oui	Non	Non
Poddar-Unnirischnan	Oui	Oui	Non	Non	Oui

Tab. 1.1 – Résumé des structures LRGF

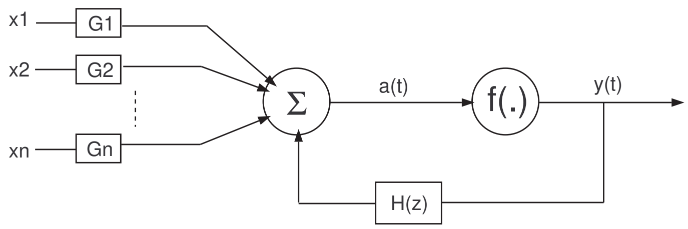

Fig. 1.15 – Une Architecture LRGF Générale

Local Activation feedback Network Dans ce cas le retour est pris avant que le signal entre dans la fonction de sortie non-linéaire. Cette architecture peut être comparée un cas particulier de l'architecture de Frasconi-Gori-Soda.

Local Output feedback Network Dans ce cas l'idée est de prendre le retour sur la sortie du neurone après la non-linéarité. Ce retour peut prendre la forme d'une fonction de transfert simple (que des pôles) voire une fonction de transfert généralisée. Ce réseau inclus l'architecture de Frasconi-Gori-Soda et de Poddar-Unnikrischnan. Dans ce cas les synapses sont seulement des constantes.

1.4.7 Conclusion et résumé de ces structures

Cf. tableau 1.1.

1.4.8 Une Architecture LRGF Générale

On peut remarquer que toutes les architecture précédentes peuvent être englobée dans un architecture plus générale (Cf. figure 1.15) qui inclus aussi bien le retour local de sortie que le retour local en entrée.

Les fonctions de transfert $G_1, G_2, \cdots, G_n$ et H sont générales et peuvent avoir des pôles et des zéros. Ce type de réseau est très général car il englobe même le perceptron.

1.4.9 Propriété de ces architectures

Ils ont la propriété d'approximateurs universels d'après Doya. Ils ont le principe de parcimonie (conserve les données et sont résistant au bruit) Les connexions récurrentes peuvent être placées de plusieurs façons dans la structure. Mais alors la question de l'efficacité par rapport à la structure choisie se pose. Il existe des différences entre les réseaux globalement récurrents et les réseaux localement récurrents. Mais on ne sait pas laquelle est la plus efficace en terme de modélisation de système dynamique. Modèles dépendant de l'état : il serait intéressant de clarifier la façon dont laquelle l'état influence le comportement du réseau LRGF. La robustesse de la structure : Les réseaux LRGF sont ils robustes aux bruits et aux élagages. (suppression d'une connexion) Contrôle adaptatif non-linéaire : Ces réseaux sont-ils probant dans le contrôles et la commande de systèmes non linéaires.

1.5 Les réseaux dynamiques récurrents

Le "Dynamic Recurrent Network" a été introduit dans le but de donner un modèle qui avec un apprentissage raisonnable présente des caractéristiques intéressantes au niveau du traitement des séquences temporelles. Cette classe de réseaux récurrents englobe un grand nombre de réseaux de neurones introduit pour le traitement temporel (Tsoi, Williams...).

Ce modèle de réseau est basé sur des unités type MacCulloch et Pitts à fonction d'activation sigmoïdale. Son architecture est des plus générales puisqu'elle permet des connexions retardées et bouclées. Le principale intérêt de ces réseaux est sa mémoire adaptable qui lui permet d'implémenter directement un système dynamique. Les bouclages et les retards entre les neurones permettent de conserver sous forme codée la mémoire. Ce réseau est décrit par un système d'équations différentielles dont les variables internes sont les activations des neurones cachés dont la forme est récurrente spatialement et temporellement.

Les différents algorithmes utilisés pour l'apprentissage et la mise à jour des ses paramètres sont d'une part la propagation en chaînage avant, la propagation en chaînage arrière, un algorithme itératif en chaînage arrière (TRBP)"Time Recurrent Back Propagation".

1.5.1 Le DRNN comme représentation de la mémoire

Afin de pallier les limitation de réseaux non-bouclés, il faut introduire explicitement une ou des récurrences dans l'architecture du réseau. Les informations intéressantes contenues dans les taches temporelles nécessitent un stockage. Mais quelques fois ces données pertinentes s'étendent sur des intervalles de temps important ce qui exclus la technique des fenêtres temporelles. Le réseau doit donc être muni d'une mémoire interne s'il l'on ne peut lui fournir les informations d'une manière externe.

C'est dans ce but que les poids synaptiques du réseau de type Rumhelhart sont remplacés par des fonctions de transfert d'un filtre linéaire à réponse impulsionnelle finie (FIR). (a une impulsion de durée finie correspond une réponse de durée finie). Alors l'activation $a_i(t)$ d'un neurone i à l'instant t est donnée par

$$x_i(n) = f\left(\sum_j \sum_{d=0}^{D} \omega_{ji}^d x_j(n-d)\right) = f\left(\sum_j \omega_{ji} * x_j(n)\right)$$

où

$$\omega_{ij} = [\omega_{ji}^0, \omega_{ji}^1, \cdots, \omega_{ji}^D,]^T \text{ et } x_j(n) = [x_j(n), x_j(n-1), \cdots, x_j(n-D),]^T$$

Les coefficients du filtre doivent être adaptés afin de correspondre au mieux avec le problème à résoudre. Ceci permet aux interactions neuronales de présenter un vaste choix de mémoires formelles.

$\omega_k^d(i,j)$	poids de la connexion retardée de d entre le neurone i et le neurone j à l'instant k
$w_k^d(j)$	$= [\omega_k^d(1,j), \cdots, \omega_k^d(N,j)]^T$
$W_k^d(j)$	$= [w_k^d(1,j), w_k^d(2,j), \cdots, w_k^d(N,j)]^T$
s_k	$= \sum_{d=0}^{D} W_k^{d^T} v_{k-d}$
v_k	$= g(s_k)$ le vecteur d'activation des neurones
x_k	vecteur d'entrée du réseau à l'instant t

TAB. 1.2 – Résumé les notations d'un DRNN

Cette méthode des connexions FIR a été introduite par Wan aux réseaux à couches. On trouve ici son extension aux réseaux bouclés. La prédiction apparaît comme le domaine de prédilection de ce type de réseau car sa version non bouclée est utilisée dans la prédiction à court terme.

1.5.2 Une dynamique à relaxation

Le DRNN est un réseau totalement bouclés de N neurones. ω_{ij}^d représente le poids de la connexion synaptique retardée de d unités de temps entre le neurone i et le neurones j. D représente le délai maximal c'est à dire l'ordre du filtre. L'activation est la réponse d'un neurone et $v_k = \left(v_k \in \mathcal{R}^{[N+1]}\right)$ sont les vecteurs d'activités des neurones à l'instant k. Le réseau possède des neurones d'entrées dont l'activation est $x_k = \left(x_k \in \mathcal{R}^{[N+1]}\right)$. Les valeurs désirées des neurones de sortie sont notées $d_k = \left(d_k \in \mathcal{R}^{[N+1]}\right)$. Le vecteur poids retardé de d unités de temps est noté $w_k^d(j) = [\omega_k^d(1,j), \omega_k^d(2,j), \cdots, \omega_k^d(N,j)]^T$ et la matrice de dimension $(N \times N)$ $W_k^d(j) = [w_k^d(1), w_k^d(2), \cdots, w_k^d(N)]^T$ contient tous les poids des connexions retardées d'un délai $d < D$.

On détermine les paramètres du système à chaque étape k en relaxant le système

$$T\frac{dv_k(t)}{dt} = -v_k(t) + g\left(s_k(t)\right) + x_k$$

$$s_k(t) = \sum_{d=1}^{D} W_k^{d^T} v_{k-d} + W_k^{0^T} v_k(t)$$

avec le vecteur T,$T = [\tau_1, \tau_2, \cdots, \tau_n]$ convenablement choisi pour amener dans un temps raisonnable le système vers un état d'équilibre.

s_k désigne le vecteur des entrées internes du neurone à l'itération k, et $g()$ est une sigmoïde de la forme $g(s_k) = [g(s_k^1), \cdots, g(s_k^n)]^T$ avec $g(s) = \frac{1}{1+\exp^{-\beta s}}$ ou β est la pente de la sigmoïde.

La convergence d'un tel système vers un point fixe n'est pas garantie (Hertz 91). On suppose donc qu'il en existe au moins un. Si le système admet donc un point d'équilibre stable ou instable il vérifie $v_k = g\left(\sum_{d=0}^{D} W_k^{dT} v_{k-d}\right) + x_k$. On doit remarquer que le système doit garder en mémoire la trace des D dernières actions, ce qui conduit à un volume de stockage de DN valeurs. On peut résumer les notations d'un DRNN dans le tableau 1.2

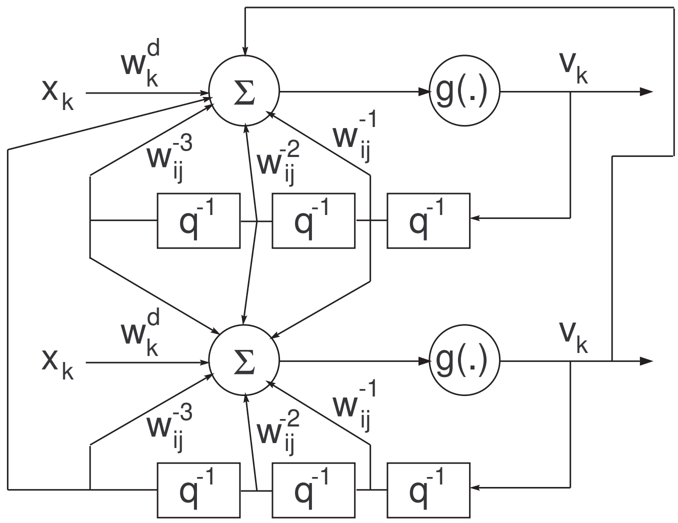

FIG. 1.16 – Exemple de DRNN simple avec deux entrées et deux sorties

1.6 Les Algorithmes d'apprentissage des réseaux dynamiques récurrents

1.6.1 Méthodes d'apprentissage

Afin d'acquérir sa "connaissance", un réseau de neurones passe par une phase d'apprentissage. Celle-ci s'effectue grâce à un algorithme d'apprentissage. Il existe beaucoup d'algorithmes d'apprentissage et le choix d'un d'entre eux reste difficile. Pour un même problème leur efficacité sera différente. Il faut donc choisir l'algorithme d'apprentissage en fonction du problème. Une des méthodes d'apprentissage des réseaux de neurones est la rétropropagation du gradient de l'erreur.

1.6.2 La rétropropagation du gradient de l'erreur

L'algorithme de rétropropagation du gradient de l'erreur à été crée en généralisant les règles d'apprentissages de Widrow-Hoff aux réseaux multicouches à fonction de transfert non-linéaire. C'est un algorithme utilisé avec des réseaux de type feedforward pour l'apprentissage de fonction, la reconnaissance de formes et la classification.

1.6.2.1 Principe

La rétropropagation du gradient de l'erreur est utilisée pour ajuster les poids et les biais du réseau afin de minimiser l'erreur quadratique entre la sortie du réseau et la sortie réelle. A chaque couple entrée/sortie, une erreur est calculée, le gradient, ou pente, de l'erreur est déterminé. Ensuite les poids et les biais sont modifiés en ligne sur le réseau. On réitère ces calculs jusqu'à l'obtention du critère d'arrêt.

1.6.2.2 Algorithme

1. Initialisation des poids W[q] à des petites valeurs aléatoires
2. Présentation d'une entrée x_k et de la sortie désirée d_k.

3. Calcul de la sortie actuelle par propagation à travers les couches par $y_j^{[q]} = F\left(\sum_i W_{ji}^{[q]} \cdot y_i^{[q-1]}\right)$ où F est la fonction de transfert du neurone et $[q]$ la q^e^couche du réseau.
4. Accumulation des erreurs en sortie $\epsilon = \sum_k \left(d_k - y_k^{[s]}\right)$ Où d_k est la sortie désirée associée au vecteur d'entrée x_k. Où $y_k^{[s]}$ est la sortie obtenue sur la dernière couche au temps t. ϵ est l'erreur cumulée pour k présentations de couples (x_k, d_k).
5. Rétropropagation du gradient de l'erreur (δ) depuis la dernière couche vers la première couche
   - F Pour chaque cellule de sortie $\delta_i^{[s]} = -\left(d_i - y_i^{[s]}\right) \cdot F'\left(p_i^{[s]}\right)$
   - F Pour chaque cellule cachée $\delta_i^{[q]} = -\sum_k \delta_k^{[q+1]} \cdot W_{ki} \cdot F'\left(\delta_i^{[p]}\right)$
6. Mise à jour des poids selon la règle : $\Delta W_{ij}^{[q]} = \alpha \cdot \left(\delta_k^{[q]} \cdot x_j^{[q]}\right)$ où α est le coefficient d'apprentissage compris dans l'intervalle $[0, 1]$.
7. Retour à 2 tant qu'il y a des couples à présenter.

Choix du critère à minimiser

Dans le cas de la rétropropagation de l'erreur, le critère à minimiser est une erreur quadratique. L'application de l'algorithme du gradient nécessité la dérivabilité de la fonction de transfert. Le critère de minimisation d'erreur est le suivant $\epsilon = \sum_k \left(d_k - y_k^{[s]}\right)^2$

1.6.2.3 Avantages et inconvénients

Avantages Ce fût un des premiers algorithmes développés pour l'apprentissage des réseaux de neurones multicouches de type feedforward. Il permet de pallier une carence de l'algorithme du perceptron qui est incapable de modifier les poids des couches cachées [DAVALO et NAÏM, 1993]. L'implémentation informatique ne présente pas de difficultés. Il a surtout permis de sortir le connexionnisme de l'impasse que représente l'apprentissage des réseaux de types feedforward multicouches en permettant la modification des poids des couches cachées.

Inconvénients En ce qui concerne l'algorithme

- L'algorithme de rétro-propagation du gradient de l'erreur suit la descente du gradient de l'erreur : un minimum local peut rapidement bloquer la recherche des optima globaux.
- L'algorithme de rétropropagation est gourmand en temps de calcul.
- Importance du choix du coefficient d'apprentissage, si le coefficient est trop grand la dynamique du réseau va osciller autour de l'optimum, si il est trop petit, la convergence est lente et elle va se faire piéger dans un minimum local.
- Les fonctions de transfert doivent être dérivables.

1.6.3 Descente du gradient

La descente du gradient n'est pas applicable directement au réseau récurrent du fait de ses connexions bouclées et retardées. La variation de la valeur d'un paramètre à l'instant présent se répercute sur l'évolution ultérieure du système. La rétropropagation dans le temps (BPTT Backpropagation Through Time) appliquée au réseau récurrent donne au final un grand réseau non-bouclé multicouche dont la taille croit avec le nombre d'itération. Cette méthode devient impraticable lorsque le système requiert pour son apprentissage des fenêtres temporelles illimitées. C'est pourquoi les chercheurs se sont orientés vers des méthodes plus attractives comme le TRBP (Temporal Recurrent BackPropagation)

1.6.4 Propagation récurrente avant et arrière

1.6.4.1 Introduction

L'objet de l'apprentissage est d'ajuster la matrice de poids du réseau $\{\omega_{ij}^d\}$ de façon à le faire évoluer sous l'action des entrées, d'un état initial vers des points fixes désirés correspondant aux sorties désirées du réseau. Il faudra encore minimiser une erreur quadratique. Grâce aux lois d'expansion des dérivées partielles (due à l'architecture du réseau) on obtient deux algorithmes différents : algorithme d'adaptation récursif et algorithme d'adaptation à balayage arrière ou propagation récurrente avant et récurrente arrière (reflet de la direction des dérivées partielles par rapport au données).

Loi d'expansion des dérivées partielles
soit $z_i = f_i(z_1, \cdots, z_{i-1}, z_i)$
et la dérivée partielle
$\frac{\partial^+ z_j}{\partial z_i} = [\frac{\partial z_j}{\partial z_i}]_{\{z_1,\cdots,z_{i-1}\}}$

Première méthode

$$\frac{\partial^+ z_j}{\partial z_i} = \sum_{k=i+1}^{j} \frac{\partial^+ z_j}{\partial z_k} \left(I - \frac{\partial f_k(z)}{\partial z_k}\right)^{-1} \frac{\partial f_k(z)}{\partial z_i}$$

Deuxième méthode

$$\frac{\partial^+ z_j}{\partial z_i} = \sum_{k=i}^{j-1} \left(I - \frac{\partial f_j(z)}{\partial z_j}\right)^{-1} \frac{\partial f_j(z)}{\partial z_k} \frac{\partial^+ z_k}{\partial z_i}$$

La propagation récurrente avant : On utilise la seconde loi d'expansion. On développe $\frac{\partial^+ E}{\partial \omega}$ ou ω est un scalaire de la matrice W. Cet algorithme généralise le RTRL (Real Time Recurrent Learning) développé par Williams & Zipser.

La propagation récurrente arrière Propagation récurrente arrière rétropropagation arrière temporelle (TRBP). Le problème de cette algorithme vient de l'explosion numérique qu'il occasionne. (inversion matricielle, mémoire de stockage requise). D'où l'idée d'appliquer des stratégies afin de réduire la complexité en gardant à l'esprit de garder l'efficacité de l'algorithme

- Ignorer l'influence de l'état courant sur les erreurs futures (problème de résolution des problème temporels).
- Solution qui consiste à approximer les dérivées partielles afin de réduire le coût de stockage.

1.6.4.2 La propagation récurrente arrière

Ici on utilise la première loi et ceci conduit à un algorithme sans inversion matricielle mais qui nécessite la relaxation d'un réseau adjoint (rétropropagation temporelle récurrente). L'algorithme définitif englobe la rétropropagation temporelle dans les réseaux FIR non-bouclés (Wang), pour les réseaux statiques récurrents (almeida), pour les réseaux à délais (Rumelhart). Cet algorithme peut se concevoir comme la rétropropagation récurrente appliquée à un réseau adjoint (Baldi) ou transposé (Hertz).

Un DRNN simple et son transposé sont décrits par :

- inverser les signaux tels que les sorties soient les entrées.
- redéfinir les neurones comme des opérateurs affines en utilisant la dérivée du point fixe.
- inverser le temps.

Cet algorithme nécessite le dépliage d'un adjoint dans le temps. Cette duplication est effectuée jusqu'à ce que tous les délais aient disparus. L'algorithme produit un réseau de propagation temporelle et récurrent dans l'espace. (connexions bouclées mais non retardées uniquement). Mais ceci demande un grande capacité de stockage et de calcul : Ceci donnera l'algorithme TRBP Temporal Recurrent BackPropagation. A un instant donné les gradients sont propagés dans le réseau adjoint en temps inverse ($t = k \rightarrow t = 0$). Le dépliage du réseau adjoint s'opère à la fin de l'itération pour permettre la calcul au fur et à mesure que les nouveaux exemples sont présentés.

1.6.4.3 Stabilité de la procédure d'adaptation des poids

La stabilité asymptotique de l'adaptation de la matrice de poids quand le nombre d'itération tend vers l'infini est indispensable pour assurer le bon fonctionnement du réseau et de son apprentissage.

On peut prouver la convergence locale (sur une itération dans le réseau adjoint), les gradients sont majorés par une suite décroissante à vitesse exponentielle.

1.6.4.4 Critère de dépliage

On cherche à connaître le nombre de rétropropagations requises par l'algorithme afin de minimiser la mémoire requise au stockage de ces gradients.

1.6.4.5 L'algorithme du TRBP Temporal Recurrent BackPropagation

Cet algorithme tire avantage de la décroissance exponentielle des gradients afin de minimiser l'explosion combinatoire. L'utilisation de l'adjoint permet une symétrie utilisée par une implémentation temporelle. Sa principale caractéristique vient de l'ajustement optimal du nombre de passe dans l'adjoint afin de réduire le nombre d'opérations nécessaires.

1.6.4.6 Apprentissage forcé

Il suffit de remplacer les sorties calculées par le réseau par les valeurs désirées en sorties. Ressources requises : L'ordre du réseau est D la complexité spatiale est en $O(DN)$, en contraste avec $O(DN^3)$ pour la propagation récurrente avant. Une inversion matricielle n'est plus nécessaire.

1.6.4.7 L'évanouissement du gradient dans l'adjoint

Problème de l'apprentissage à long terme pour la propagation du gradient dans l'adjoint. Un bon apprentissage peut se résumer à deux choses :

– Le système doit être résistant au bruit.
– Le système converge en temps raisonnable.

Le gradient prend plus en compte les dépendances temporelles à cours termes que les dépendances à longs termes. Si le gradient tend à devenir faible sa propagation ne va pas modifier de façon suffisante les poids et va s'évanouir dans l'adjoint. C'est pourquoi cet algorithme est inadapté pour capturer les séquences temporelles à longs termes.

1.6.5 L'algorithme de SOLLIS et WETS

Devant les limites de l'algorithme de rétropropagation du gradient de l'erreur, l'utilisation d'une méthode d'optimisation s'impose. Le choix s'est tourné vers une méthode d'optimisation stochastique : **l'algorithme de SOLLIS et WETS /GLORENNEC 94 b/**.

Principe

Il est basé sur la recherche aléatoire du meilleur vecteur de paramètres qui minimise la fonction de coût. Cette recherche est guidée par une phase qui "mémorise" les bonnes performances des apprentissages précédents. Soit M un vecteur contenant l'ensemble des paramètres à optimiser. Soit G un vecteur gaussien chargé de faire évoluer les paramètres de M. Un autre vecteur B, contenant la moyenne de G, sera lui utilisé pour mémoriser les réussites des apprentissages précédents.

1.6.5.1 Algorithme de SOLLIS et WETS

Dans cet algorithme nous aurons besoin de trois vecteurs et d'une fonction de coût :

- le vecteur M, qui contient les paramètres à optimiser.
- le vecteur G, de même dimension que M, qui contient un bruit gaussien d'écart-type.
- le vecteur B, de même dimension que M et G, et contient la moyenne de G, c'est le biais. Il mémorise les succès dans la minimisation de la fonction de coût.
- une fonction de coût E.

1. Choisir un vecteur initial $M(0)$. On pose $B(0) = 0$ et $k = 0$
2. Générer un vecteur gaussien, $G(k)$, de moyenne $B(k)$ et d'écart-type σ.
3. si $E\left(M(k) + G(k)\right) < E\left(M(k)\right)$
   alors $M(k+1) = M(k) + G(k)$ et $B(k+1) = 0.4G(k) + 0.2B(k)$
   sinon si $E\left(M(k) - G(k)\right) < E(M(k))$
   alors $M(k+1) = M(k) - G(k)$ et $B(k+1) = B(k) - 0.4G(k)$
   sinon $M(k+1) = M(k) et B(k+1) = 0.5B(k)$
4. si $k > \max$ ou $erreur < seuil$
   alors *stop*
   sinon $k = k + 1$. Aller en 2.

Dans la méthode initiale, l'écart-type σ du gaussien, est le même pour tous les paramètres. On peut aussi ajouter un bruit proportionnel au paramètre à optimiser par la modification suivante $\sigma[i] = C \times N\left(B^{(k)}[i], 1\right) \times M^{(k)}[i]$ où C est une constante et $N\left(B^{(k)}, 1\right)$ un gaussien de moyenne $B^{(k)}[i]$ et d'écart-type 1. Cette modification permet à l'algorithmes de traiter des valeurs non-homogènes.

1.6.5.2 Choix du critère à minimiser

Dans le cas de l'optimisation par SOLLIS et WETS, le critère à minimiser n'est pas obligatoirement une erreur quadratique. Il n'est pas nécessaire pour cet algorithme qu'elle soit dérivable. Le critère à minimiser a donc été défini par $\epsilon = \sum_k |d_k - y_k^{[s]}|$.
Ce critère permet de ne pas minimiser les erreurs quand elles sont inférieures à 1.

1.6.5.3 Avantages et inconvénients

Avantages Cette méthode converge avec une probabilité de 1 vers le minimum global de la fonction de coût /GLORENNEC 94 b/. Elle est adaptable à tout type de structure de réseau de neurones car on optimise un vecteur de paramètres et aucune topologie particulière n'est requise contrairement à la rétropropagation du gradient de l'erreur, dont la convergence est guidée par la seule pente de la fonction de coût, sa recherche est aléatoire et légèrement guidée ce qui lui permet d'agrandir son espace de recherche. Il évite ainsi les minimums locaux

Son implémentation informatique ne présente pas de difficultés majeures. Les fonctions de transfert des neurones importent peu. Elles ne sont pas nécessairement dérivables

Inconvénient Le dernier avantage peut se révéler un inconvénient en rendant la convergence lente par un espace de recherche trop grand. Un autre paramètre important à régler est la puissance du bruit gaussien. C'est à dire la grandeur des valeurs ajoutées ou retranchées aux vecteurs des paramètres. Il est évident que plus le bruit gaussien est important, plus la surface de recherche est importante et inversement.

1.6.6 Le Recuit Simulé

C'est une méthode méta-heuristique destinée à résoudre les problèmes d'optimisation difficiles. Cette méthode permet en principe d'éviter les pièges des minima locaux. Elle est inspirée par la technique expérimentale du "recuit" utilisée par les métallurgistes pour obtenir un état solide bien ordonné d'énergie minimale.

Caractéristiques du recuit simulé Comme d'autres méthodes récentes (Algorithmes génétiques, méthode de recherche Tabou) elle doit permettre de résoudre au mieux des problèmes difficiles composés principalement des problèmes NP difficiles (optimisation combinatoire).
Problèmes dont on n'est pas sur à 100

Le recuit est une méthode méta-heuristique qui possède les caractéristiques suivantes

- Elle est stochastique (seule méthode praticable face à l'explosion combinatoire)
- Elles ne peuvent s'appliquer aux problèmes continus qu'après une adaptation (elles viennent du combinatoire)
- Ce sont des méthodes directes (par opposition au gradient que l'on rétropropage)

1.6.6.1 La méthode du recuit simulé

Elle permet en principe de déterminé les minima globaux d'une fonction de coût. Elle est basée sur des phénomènes physiques que l'on nomme Verre de Spin. Ce phénomène

Problème d'optimisation	Système physique
Fonction objectif	Énergie libre
Paramètres du problème	"Coordonnées" des particules
Trouver une bonne configuration	Trouver des états de basses énergie

Tab. 1.3 – Analogie entre la Physique et l'Optimisation

caractérise les configurations de basse énergie de matériaux magnétiques désordonnés. La recherche des énergies minimales pose un problème d'optimisation redoutable en regard de la surface d'énergie qui présente de grandes irrégularités.

C'est pourquoi S. Kirkpatrick & al. ont proposé de régler ce problème en s'inspirant de la technique expérimentale du "recuit".

L'analogie entre la physique et le problème d'optimisation permet utiliser le recuit comme une méthode d'optimisation (Cf. tableau 1.3).

Pour modifier l'état d'un matériau, le physicien dispose d'un paramètre de commande : **la température**. Le recuit est une stratégie de contrôle de la température. En refroidissant lentement le matériau en marquant des paliers de longueur suffisante on obtient un état solide stable correspondant à un minimum d'énergie. Cette idée a conduit les chercheurs à utiliser cette méthode pour l'optimisation de problème.

1.6.6.2 Algorithme du recuit simulé

Basé sur deux résultats de la physique statistique
Lorsqu'un équilibre thermodynamique est atteint à une température T, la probabilité de posséder une énergie donnée E est proportionnelle au facteur de Boltzman $\exp\left(-fracEk_BT\right)$ où k_b désigne la constante de Boltzman.

Afin de simuler l'évolution d'un système physique vers son équilibre thermodynamique, à une température donnée on utilise l'algorithme de Métropolis. Partant d'un configuration donnée, on faire subir au système une modification élémentaire.

Si cette transformation diminue le fonction d'énergie, alors on accepte cette transformation sinon on accepte cette mauvaise modification d'énergie ΔE avec la probabilité $\exp\left(-frac\Delta ET\right)$.

En appliquant cette règle de façon itérative on engendre une séquence de configurations qui constitue une chaîne de Markov (une configuration dépend de celle directement précédente). En principe la chaîne est de longueur infinie, mais en pratique on arrête d'appliquer la règle quand le système a atteint un équilibre thermodynamique intéressant.

On comprend alors le sens de la température T : à haute température on accepte quasiment la plupart de mouvements dans le sens opposé de la minimisation de la fonction d'énergie.L'algorithme équivaut à une simple recherche aléatoire dans l'espace des configurations.

Par contre à basse température $\exp\left(-frac\Delta ET\right)$ est proche de 0. La plupart des déplacements augmentant : l'algorithme se ramène à une amélioration itérative classique.

Convergence théorique du recuit simulé Sous certaines conditions, le recuit simulé converge en probabilité vers un optimum global. Ceci est intéressant par rapport à d'autres méthodes dont la convergence n'est pas prouvée.

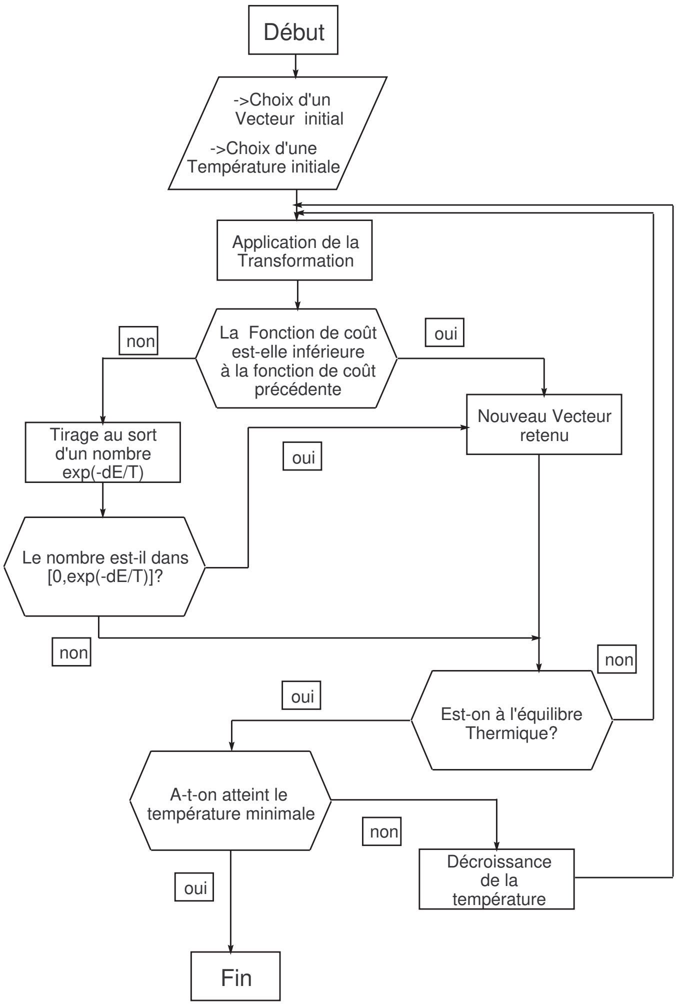

Fig. 1.17 – Algorithme du Recuit Simulé

1.6.6.3 Espace des configurations

Cet espace de recherche joue un rôle fondamental dans l'efficacité de l'algorithme. Cet espace est doté d'une topologie ou paysage d'énergie. Tout le problème réside dans le parcours de ces vallées et plateaux qui sont autant de minima locaux et de zones mortes.

Le choix de la fonction de coût ainsi que les déplacements élémentaires rentre dans une large part aussi dans l'efficacité de l'algorithme.

1.6.6.4 Avantages et inconvénient de la méthode

Les plus Elle procure une solution de bonne qualité. C'est une méthode générale : applicable à un grand nombre de problème et facilement programmable. Elle offre une grande souplesse d'emploi et de nouvelles contraintes peuvent être facilement ajouter au programme.

Les moins Le nombre de paramètre important à régler (température initiale, taux de décroissance, durée des paliers de température, critère d'arrêt...). L'algorithme garde un aspect empirique. Le temps de calcul peut être rédhibitoire.

1.6.7 Apprentissage par les Algorithmes Génétiques

1.6.7.1 Introduction

il existent différentes sortes de méthodes d'apprentissage pour les réseaux de neurones :

Deux classes de rétropropagation pour les RNR.

- Rétropropagation à travers le temps : dépliage temporel du réseau récurrent et application de l'algorithme.
- Rétropropagation récursive ou dynamique (temps réel)

Recuit simulé [Van de Bout Miller 89] occasionne des temps de calcul prohibitif

Nouvelle approche : application des théories de l'évolution sur les RNR.

Les algorithmes génétiques sont appliqués à la recherche d'architectures adaptées au problèmes posés ainsi qu'à la recherche des paramètres.

Pour les réseaux récurrents il y a deux facettes pour l'apprentissage

- Détermination de l'architecture par les Algorithmes génétiques
- Détermination des poids du réseau par différents algorithmes stochastiques (recuit simulé, Sollis et Wets).

Les Algorithmes Génétiques sont utilisés pour le développement des connexions récurrentes. Les algorithmes stochastiques peuvent être utilisés du fait qu'ils ne nécessitent pas de fonctions de transfert particulières. De plus ils facilitent une implémentation en ligne.

Comme dans tout type de réseau, l'apprentissage se résume à la minimisation d'une fonction de coût afin de trouver un minimum global. Ces méthodes garantissent l'obtention d'un optimal mais il reste le problème de la vitesse de convergence.

L'intérêt des GA est leur flexibilité, l'implémentation assez facile et modifiable à un large champs de problème. Mais il faut revoir l'implémentation à chaque problème spécifique. Il reste toujours le problème de réglage des coefficients de mutation, de sélection cross-over...

1.6.8 GNARL Generalized Acquisition of Recurrent Links

1.6.8.1 Introduction

Comme définit précédemment les clefs d'un bon apprentissage avec les RNR est l'apprentissage de l'architecture (sa détermination) et l'adaptation des paramètres.

Les méthodes courantes peuvent être classées en deux types

- Les algorithmes constructifs : on démarre une structure simple et on ajoute des neurones nécessaires.
- Les algorithmes destructifs : on démarre d'un grand réseau et on taille les parties inutiles.

Le problème avec ces méthodes, elle ne permettent pas d'explorer toutes les configurations possibles de structures.

D'où le développement du **GNARL**. Cet algorithme permet d'acquérir en même temps la structure et les paramètres. Utilisation de la programmation évolutionniste.

1.6.8.2 Développement des réseaux connexionnistes

La programmation évolutionniste permet d'avoir une collection d'algorithme pour l'apprentissage structurel et paramétrique des R.N.. Ces algorithmes se distinguent parce qu'ils sont basés sur une population de position de recherche dans l'espace et non pas d'une seule position. Pendant un cycle de recherche tous les membres de la population sont classés par rapport à une fonction de coût. Les meilleurs sont sélectionnés pour être les parents de la prochaine génération. La progéniture est crée à partir de reproductions heuristiques spécialisées des parents.

Les algorithmes génétiques sont la forme la plus connue des théories évolutionnistes. Ils sont basés principalement sur le cross-over.

1.6.8.3 Développement des réseaux de neurones par les GA

Les GA créent de nouveaux individus en recombinant les composants de deux membres de la population. Les GA se basent sur deux espaces de représentation distincts

- Espace de recombination : typiquement chaîne de bits, la structure sur laquelle les opérateurs génétiques sont appliqués. (recherche actuelle)
- Espace d'évaluation : groupe de structures qui sont capables de faire la tâche.

Entre ces deux espaces il y a une fonction de d'interprétation. Cette représentation duale est intéressante car on ne peut pas faire le recherche dans l'espace d'évaluation directement. Si on suppose que la fonction d'interprétation existe, on fait la recherche dans l'espace de recombination. On fait alors la deuxième supposition : le fait d'avoir la meilleure recombination au sens de la fonction d'interprétation entraîne le meilleur individu dans l'espace d'évaluation. Mais la fonction d'évaluation n'existe pas forcément.

1.6.8.4 Problème du Cross-over

Développement de réseaux qui partagent la même topologie et les même poids : le cross-over va créer une progéniture qui auront des composants redondants et perdront les capacités de leurs parents $\rightarrow$ dégradation.

Développement de deux réseaux qui ont une topologie identique mais des poids différents : Il est reconnu que pour une seule topologie plusieurs solutions sont possibles en termes de poids pour une même tâche. Mais chacune est implémentée par une représentation distribuée dans les couches du réseau.

Quand les parents diffèrent trop topologiquement : lors de la recombination des composants, la progéniture crée peut avoir une représentation trop distincte de l'un et de l'autre des parents (entre les deux) et peut donc se révéler incompatible.

1.6.8.5 Réseaux et programmation évolutionniste

A la différence des A.G., la programmation évolutionniste définit des opérateurs de mutations, qui créent la progéniture avec une comparaison avec les parents. L'opérateur de mutation modifie les paramètres du réseau η par l'équation $\omega = \omega + N(0, \alpha\epsilon(\eta))\forall\omega \in \eta$ où ω est un poids; $\epsilon(\eta)$ erreur du réseau sur la tâche; α est une constante; $N(\mu, \tau^2)$ est une variable Gaussienne.

Contrairement aux A.G. cette théorie manipule directement le réseau et n'est pas obligée de passer par la représentation duale (recombination, évaluation). En évitant le cross-over entre les réseaux on conserve l'individualité de la progéniture.

1.6.8.6 L'algorithme GNARL

C'est un algorithme basé sur les théories de l'évolution qui construit des Réseaux de Neurones en tenant compte de l'architecture et des paramètres de son apprentissage. La topologie de ces réseaux est des plus générales sans se limiter aux architectures uniformes et symétriques. Ici l'architecture reflète beaucoup plus la tâche à effectuer.

Sélection, reproduction, et mutation des réseaux
Le GNARL initialise une population une population de R.N. générée aléatoirement (le nombre de neurones cachés est choisi dans une distribution uniforme définie par l'utilisateur; idem pour le nombre de connexions initiales).

Les neurones touchés par une connexion sont choisi en concordance avec les mutations de structure. Quand une topologie est choisie tous les poids sont initialisés de manière aléatoire $[-1, +1]$. A chaque génération de recherche, les réseaux sont évalués par une fonction de coût définie par l'utilisateur. Les réseaux qui sont les meilleurs (1^re^moitié en prenant les erreurs classées) seront les parents de la prochaine génération. Les autres sont détruits.

La méthode de sélection est utilisée dans beaucoup d'algorithme de calcul évolutionniste bien que d'autres méthodes plus compétitives aient vues le jour. La génération de la progéniture de décompose en trois étapes :

- Copiage des parents
- détermination de la mutation à effectuer
- mutation des copies

Les mutation de réseaux sont séparées en deux classes qui correspondent aux types d'apprentissage :

- mutation paramétrique (change les valeurs des poids)
- mutation structurelle (change le nombre de neurones et les connexions; change directement l'espace des paramètres)

1.6.8.7 Sévérité de mutation

Le taux de mutation d'un parent η est dicté par la température du réseau.
$T(\eta) = 1 - \frac{f(\eta)}{f_{\max}}$ où $f_{\max}$ est le coût maximum pour une tâche. La température est

déterminée en fonction de la concordance du réseau à être solution de la tâche. Cette mesure de performance du réseau est utilisée pour recuire les similitudes paramétriques et structurelles entre les parents et la progéniture. Un réseau avec une grande température sera beaucoup muté alors que ceux avec une température plus basse le seront moins. On part d'une recherche grossière qui s'affine petit à petit.

1.6.8.8 Mutation paramétriques des réseaux

Cette mutation est effectuée en perturbant chaque poids ω du réseau η avec un bruit Gaussien [Fogel]. Les poids sont modifiés comme suit : $\omega = \omega + N(0, \alpha T(\eta))\forall\omega \in \eta$où$\alpha$ est une constante proportionnelle et $N(0, \alpha T(\eta))$ une loi normale gaussienne.

On doit occasionnellement avoir de larges mutations paramétriques afin d'éviter les minima locaux pendant la recherche. Ceci affecte la capacité de la progéniture. Afin de compenser cette perte d'efficacité, la mise à jour des poids est faite avec une variante de l'équation : $\hat{T}(\eta) = U(0,1)T(\eta)$ ou $U(0,1)$ est une variable aléatoire uniforme sur $[0,1]$. Cette modification diminue la fréquence des larges mutations paramétriques sans les exclure complètement.

1.6.8.9 Mutation structurelles du réseau

Cette mutation change le nombre d'unités cachées du réseau et les connexions entre les neurones. Afin d'éviter les sauts critiques dans la minimisation de la fonction de coût, les mutations structurelles tendent à préserver le comportement du réseau. Les nouvelles connexions sont initialisées avec des poids nuls, conservant le comportement du réseau. Dans le même ordre d'idée, les neurones ajoutés dans la structure sont sans incidence sur les connexions. Elles seront ajoutées à la suite des mutations structurelles suivantes qui détermineront comment insérer la cellule ajoutée.

Mais cette notion de continuité est assez difficile à mettre en place. Exemple : enlever un neurone revient à modifier complètement la structure du réseau autour de ce neurone (liens + poids). La suppression d'un lien supprime le poids qui lui est attaché. De plus il faut s'intéresser à la classe du neurones et des ses liens (neurones d'entrées, cachés, de sortie). Quand il y a trop de différence entre le nombre de neurones cachés et le nombre de neurones d'entrée et de sortie, le processus de sélection doit être biaisé. Afin d'augmenter l'efficacité de la mutation structurelle on peut encore modifié l'équation d'adaptation comme suit $\Delta_{\min} + [U(0,1)\hat{T}(\eta)(\Delta_{\max} - \Delta_{\min})]$ avec $[\Delta_{\min}, \Delta_{\max}]$ un intervalle donné pour une mutation.

1.6.8.10 Modification du réseau pour adaptation

Afin de déterminer un réseau pour une tâche, l'algorithme GNARL n'a pas besoin d'un vecteur cible explicite, seule l'information de la fonction de coût est nécessaire. Mais si un tel vecteur existe on peut modifier la mesure de l'adéquation du réseau par les calculs suivants (de la moins pénalisante à la plus pénalisante.)

$$\sum_i (y_i - \text{Sortie}(\eta, x_i))^2$$

$$\sum_i |y_i - \text{Sortie}(\eta, x_i)|$$

$$\sum_i e^{y_i - \text{Sortie}(\eta, x_i)}$$

Le choix de la fonction d'adéquation n'altère en rien l'algorithme GNARL.

Utilisation de cet algorithme pour la recherche de nourriture par les fourmis (Nolfi)

1.6.8.11 Conclusion

Cette algorithme contrairement aux algorithmes comme la rétropropagation peut utiliser des fonctions d'activations non dérivables. Il permet aussi une recherche non monotone de structure de réseaux de neurones. Et une grande nouveauté permet la manipulation directe de la structure du réseau, contrairement aux autres algorithmes qui partent d'une structure donnée et qui ne font que l'optimisation des poids sur cette structure.

1.6.9 La méthode des cibles mouvantes

Dans cet algorithme il y ceux phases qui s'alternent. Dans la première phase les cibles de unités cachées sont améliorées si bien que si les cibles sont atteintes une meilleure performance sera obtenue. Dans l'autre phase les poids sont modifiés de telle sorte que chaque unité se rapproche de la cible à atteindre. L'erreur peut être considérées comme ayant 2 termes : une qui pénalise les unités qui sont trop loin des cibles et une autre qui pénalise les cibles qui sont trop loin des valeurs atteintes actuellement par les unités. Cette technique fait appel au découplage temporelle des actions distantes pendant l'apprentissage des poids. Elle possède aussi de désavantage de devoir stocker et mettre à jour les cibles. Lorsque le pas d'apprentissage tend vers zéro la méthode des cibles mouvantes est équivalente à la rétropropagation dans le temps.

Le principal désavantage de cette méthode est pour chaque couple d'E/S à apprendre il faut posséder les cibles des unités cachées. De plus ces cibles doivent être apprises comme les poids du réseau. Cette méthode devient donc inapplicable pour un apprentissage en ligne. (algorithme local avec les nouvelles architectures)

1.7 Forme canonique des réseaux de neurones

Les réseaux de neurones peuvent être considérés comme des filtres non-linéaires généraux.

Les développements des réseaux de neurones ont fait apparaître des analogies entre les approches neuronales et las approches classiques de l'automatique.

Ces analogies peuvent être mises en évidence par des simulations (qui trouvent leurs limitations en terme du système sur lequel elles sont effectuées) ou sur des bases conceptuelles qui tentent de mettre en perspective des méthodes et des algorithmes pour ces différentes approches.

1.7.1 Remarque sur les filtres adaptatifs

Ils sont utilisés pour le traitement du signal, la modélisation, l'estimation et la détection de signal. Ils jouent aussi un rôle important dans la modélisation et la commande de systèmes. Il existe deux types de filtres

- filtre transversal : FIR finite impluse response dont les sorties sont des fonctions linéaires des entrées seules.

- filtre récursif : IIR infinite impulse response dont les sorties sont des fonctions linéaires des entrées et des sorties retardées.

Un filtre transversal peut être définit par $y(n) = \phi[U(n), U(n-1), \cdots, U(n-M+1)]$ M est la longueur de la mémoire du filtre. Le filtre récursif peut être noté par $y(n) = \phi[U(n), U(n-1), \cdots, U(n-M+1), y(n-1), y(n-2), \cdots, y(n-N)]$ où N est l'ordre du filtre. La capacité d'un filtre à effectuer une tâche est exprimée en fonction d'un critère qui est la plus souvent une erreur quadratique. (somme quadratique pondérée des erreurs de sortie)

Un filtre adaptatif est un système qui met à jour ses paramètres continuellement, sans contrôle explicite de l'utilisateur. L'intérêt des filtres adaptatifs découlent de deux faits

- Tayloriser un filtre (concevoir) d'une architecture donnée pour effectuer une tâche spécifique, demande une connaissance a priori du système.
- Filtrer des signaux non-stationnaires nécessite des systèmes qui soient capables de suivre les variations des caractéristiques du signal.

Les principaux algorithmes d'adaptations sont dédiés aux filtres adaptatifs linéaires, on trouve le RLS Recursive Least Square et Fast RLS [Bellanger, Haykin]. Mais ils restent inadaptés pour les systèmes à signaux non-linéaires. On utilise alors les filtres adaptatifs non-linéaires mais il n'existe pas d'algorithme d'adaptation généraliste et on utilise alors les approches heuristiques.

Au contraire pour les réseaux de neurones ils existent des approches générales d'où l'intérêt d'assimiler ces réseaux à des filtres non linéaires adaptatifs.

1.7.2 Les réseaux de neurones

On peut distinguer deux types d'architectures

- Feedforward
- Feedback

Les architectures feedforward sont très connues ainsi que leurs limites. Les architectures feedback le sont un peu moins et étaient jusqu'à présent utilisées dans le cas des mémoires associatives. Dans ce cas les réseaux récurrents laissent évoluer leur dynamique (phase de relaxation) propre vers un point d'équilibre. Ici ils seront utilisés en les sollicitant continuellement par des entrées. Le but ne sera donc pas de stocker des informations mais d'obliger le réseau à capturer une caractéristique non-stationnaire d'un signal ou d'un système. Dans ce cas l'ordre des séquences d'entrées est très importante car il fixe la dynamique du système.

Les réseaux feedforward seront assimilés à des filtres transversaux et les réseaux feedback à des réseaux récursifs.

Dans le cas général des réseaux de neurones ils subissent un apprentissage sur un nombre fini d'échantillon et passe ensuite en phase d'utilisation. Pour les systèmes adaptatifs l'apprentissage est continuelle et le nombre d'échantillons est infini.

1.7.3 Structure d'un réseau de neurone pour les filtres non linéaires

le comportement d'un neurone discret est défini par

$$z_i(n) = f_i\left[v_i(n)\right] = f_i\left[\sum_{j\in P_i}\sum_{\tau=0}^{q_{ij}} c_{ij,\tau} z_i(n-\tau)\right]$$

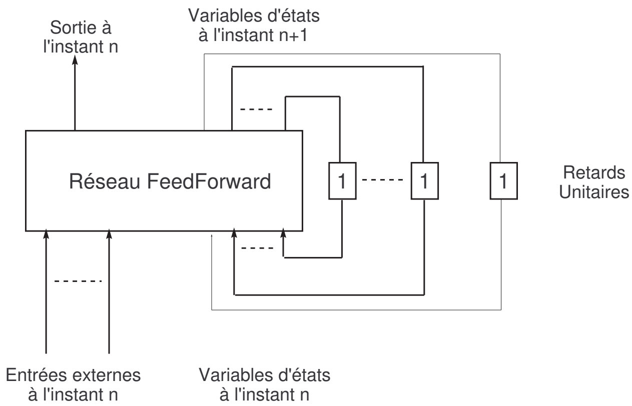

FIG. 1.18 – Forme Canonique d'un réseau récurrent

f_i est la fonction d'activation du neurone i.
v_i est le potentiel du neurone i.
z_j peut être aussi bien la sortie du neurone j qu'une entrée du réseau au neurone i.
P_i est le jeu d'indices des neurones afférents et des entrées du réseau au neurone i.
$c_{ij,\tau}$ est le poids de la synapse qui transfert l'information du neurone j (ou entrée su réseau) au neurone i, et chaque synapse a son propre retard (discret).
q_{ij} est le retard maximal entre le neurone j et le neurone i.
En pratique on aura $c_{ii,0} = 0, \forall i$ afin de préserver la causalité. Et s'il existe un neurone i tel que $i \notin P_i$ et $q_{ij} = 0, \forall j \in P_i$ alors le neurone i est dit statique.

1.7.4 Structure des réseaux de neurones pour le filtrage adaptatif

La topologie des connexions et la distribution des retards peut être partiellement ou complètement imposée par la problème à résoudre. Une connaissance a priori du système peut donner des indications et aider à la conception des structures.

Il peut être alors très intéressant de présenter une représentation canonique d'un réseau de neurones récurrent.

1.7.4.1 Forme canonique d'un réseau récurrent

Les dynamiques d'un réseau récurrent discrètisé peuvent être décrites par une équation différentielle finie d'ordre N qui peut être exprimée par un jeu de N équation du 1$^{\text{er}}$ordre mettant en jeu N variable d'états en plus de M variables d'entrées. De là n'importe quel réseau peut être décrit par une forme canonique (Cf. figure 1.18) qui consiste en un réseau feedforward :

- Dont les sorties sont les sorties désirées plus les variables d'états
- Dont les entrées sont les entrées du réseau et les valeurs des variables d'états retardées d'une unité de temps.

Le choix des variables d'états n'est pas unique. Il en résulte qu'un réseau récurrent de neurones peut avoir plusieurs formes canoniques. L'état du réseau de neurones est le vecteur de variables d'état. Le comportement d'un réseau récurrent mono-entrée/mono-

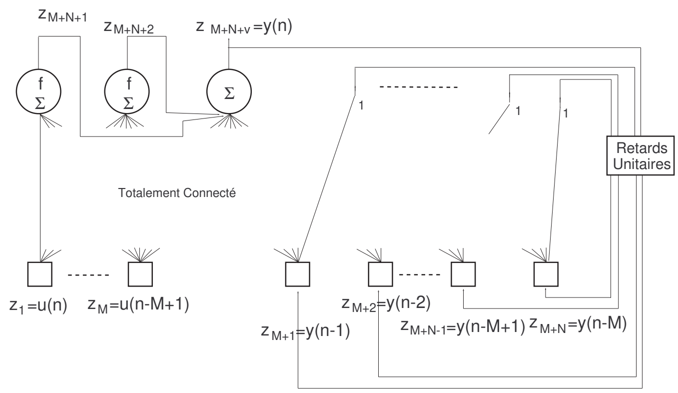

FIG. 1.19 – Forme Canonique générale d'un réseau récurrent général mono-sortie

sortie est décrit par l'équation d'état :

$$\left| \begin{array}{ccc} S(n+1) & = & \Phi\left(S(n), U(n)\right) \\ y(n) & = & \Psi\left(S(n), U(n)\right) \end{array} \right.$$

$U(n)$ est le vecteur des entrées externes $dim(M)$. $S(n)$ vecteur d'état $dim(N)$.

La transformation d'un réseau de neurones récurrent sous sa forme canonique demande la détermination M et N.

Une forme particulière des réseaux récurrent est le modèle **NARMAX** utilisé par Narendra si l'on considère que les variables d'états sont les sorties retardées du réseau de neurones où que l'état du système peut être reconstruit à partir des valeurs de l'entrée et de la sortie.

La forme canonique la plus générale d'un réseau de neurones récurrent mono-sortie est décrite par le schéma 1.19.

Forme canonique d'un réseau feedforward complètement interconnecté dont les variables d'état sont les valeurs retardées de la sortie. Si on considère un réseau récurrent quelconque et son graphe associé, on définit le vecteur de variables d'état comme le vecteur minimal des variables qui doivent être initialisée afin que l'on puisse calculer l'état à chaque instant grâce aux valeurs des entrées externes. L'ordre du réseau est le nombre de ces variables d'état. En clair les seuls neurones dont l'état doit être initialisé sont les neurones dont les sorties sont bouclées. Pour déterminer l'ordre du réseau on doit l'élaguer.

Règles de simplification Si deux connexions sont parallèles, on les combine en gardant le retard le plus grand des deux connexions. Si deux connexion d'une boucle sont séparées par un neurone qui appartient uniquement à cette boucle, On supprime le neurone et on combine les deux connexions en une en ajoutant les retards.

Le réseau est maintenant simplifier Si pour chaque noeud du graphe simplifié on note A_i le retard de la synapse afférente au neurone i qui a le plus grand retard, une borne supérieure très simple pour le degrés du réseau est $N \leq \sum_i A_i$. Attention l'activation d'un neurone i qui possède une synapse afférente retardée ne doit pas être calculée tant que le temps est inférieur au retard, sinon il y a incohérence. On doit donc fixer l'activation de ce neurone i le temps qu'il soit réellement activé.

Le choix des variables d'état n'est pas unique.

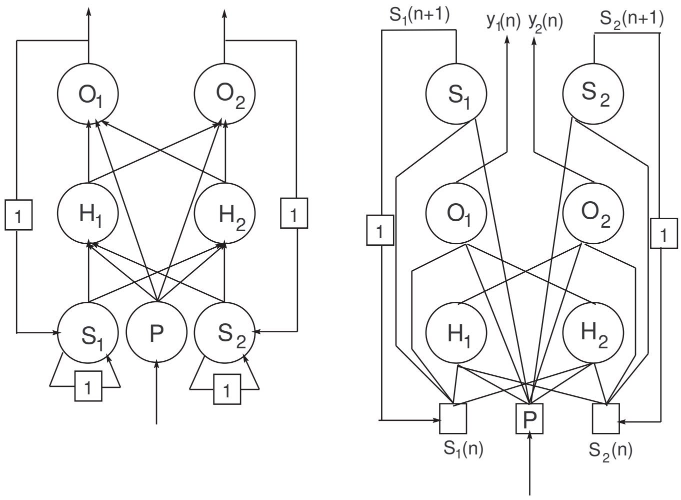

FIG. 1.20 – Architecture de Jordan et sa transcription en forme canonique

1.8 Apprentissage optimal des réseaux de neurones

1.8.1 Apprentissage comme un problème d'optimisation

Ceci met en jeu un réseau de neurones, un environnement d'apprentissage et une fonction de coût.

L'apprentissage des réseaux récurrents doit faire face au problème de l'apprentissage sous-optimal qui dépend de la surface d'erreur. La présence de minima locaux et de grands plateaux est un sérieux problème. Dans les réseaux récurrents le problème de la saturation du neurones est plus importante que dans les réseaux feedforward du fait des longues séquences à apprendre. De même si la surface comporte de trop grande variations une méthode type descente de gradient peut devenir instable.

1.8.2 Phénomène de bifurcation

Bifurcation dans l'apprentissage des réseaux neuronaux par les méthodes de descente de gradient. La différence essentielle entre les feedforward et les récurrents vient du fait que les sorties sont une fonctions continues des poids. A l'inverse la sortie d'un RNR peut être très différentes avec seulement un petit changement de paramètres si l'on passe par un poids de bifurcation. Les méthodes du gradient utilisent pour l'adaptation des poids l'équation $\tau_i \frac{dx_i(t)}{dt} = -x_i(t) + \sum_{j=1}^{n} \omega_{ij} y_i(t) + \sum_{j=1}^{m} b_{ij} z_j(t)$ et $y_i(t) = g(x_i(t))$ où $x_i(t)$ représente l'état du réseau et $y_i(t)$ les sorties du réseau, $g(\cdot)$ représente la fonction sigmoïde, $z_i(t)$ les entrées externes du réseau. L'espace d'état du réseau présente des discontinuités en certains points que l'on nomme points de bifurcation.

Si on considère un seul neurone bouclé, on obtient les équations $\tau \frac{dx(t)}{dt} = -x(t) + \omega y(t) + b$ et $y(t) - \frac{1}{1+e^{-x(t)}}$

On voit que si un biais $b \in [b1, b2]$ est ajouté a l'entrée du neurone alors la sortie du neurone peut osciller entre deux valeurs très distinctes. La zone est très instable ce qui provoque une bifurcation.

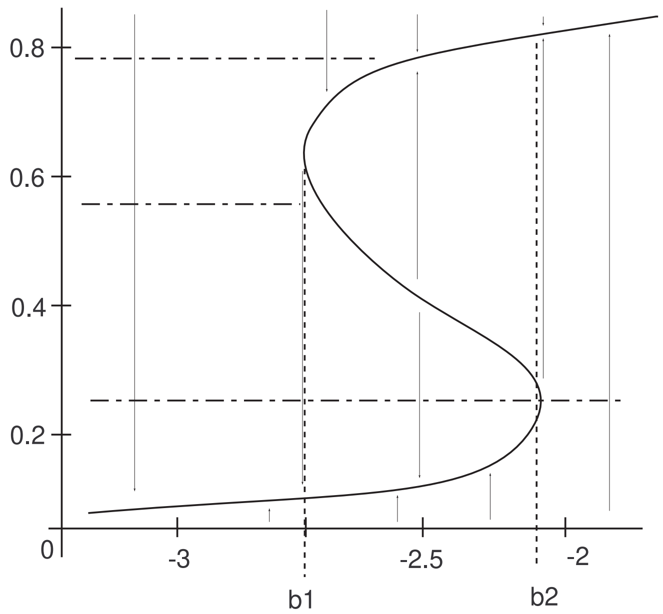

FIG. 1.21 – Diagramme de bifurcation d'un réseau auto-récurrent

1.8.2.1 Instabilité de l'apprentissage des dynamiques

La stabilité d'un système linéaire est assurée lorsque le réseau converge vers un point d'équilibre. Mais lorsque qu'un point d'équilibre bifurque alors le système linéarisé comporte un complexe conjugué à partie réelle nulle. Alors la stabilité asymptotique n'est plus garantie.

1.8.2.2 Problème de non-convergence de l'apprentissage

Parfois à cause de la bifurcation un petit changement de paramètres entraîne une grande discontinuité dans la trajectoire du réseau est donc dans la surface d'erreur. Ceci a pour effet le plus souvent d'augmenter le résultat de la fonction de coût.

1.8.2.3 Confusion entre les entrées et les sorties

Dans les tâches ou le R.N. se comporte comme un automate, son espace d'état doit comporter plusieurs bassins attracteurs afin de stocker les états internes distincts. Si les poids des connexions au début de l'apprentissage sont fixés à de petites valeurs, il n'y a qu'un seul bassin attracteur. Il faut dont passer par des bifurcations pour en avoir plusieurs. Tant que le réseaux n'a pas le bon nombre d'attracteurs i ne peut apprendre correctement la séquence d'entrée et ne donne en sortie qu'une valeur moyenne de la sortie.

1.8.3 Pour surpasser le problème de la bifurcation

La façon la moins bonne est de réduire les dynamiques du réseau. Un réseau avec un seul bassin attracteur peut être utilisé comme mémoire à cours terme.

Mais si on veut profiter pleinement des capacités des RNR il faut faire apprendre le réseau au delà des bornes de bifurcations.

1.8.3.1 Algorithmes d'apprentissage non récurrent

Un moyen d'éviter l'instabilité de l'apprentissage aux points de bifurcation est d'utiliser des règles d'apprentissage non récurrentes. On peut utiliser des approximations en propagation avant des dynamiques récurrentes. De plus ces règles demandent moins de ressources de calculs que pour le RTRL (Real Time Reccurent Learning).

1.8.3.2 Apprentissage supervisé

On remplace ici la sortie calculée par le réseau par la sortie désirée. Le rôle de cette supervision est de synchroniser les dynamiques au signal superviseur, et de réduire les degrés de liberté des dynamiques du réseau.

1.8.3.3 Pré-programmation de l'espace d'état

Si la structure de l'espace d'état est connue a priori, une approche plus pratique sera de prédéfinir l'architecture et les poids du réseau en fonction de ces connaissances.

1.8.4 Techniques avancées pour un apprentissage optimal

Optimisation globale Au lieu d'utiliser des algorithmes locaux (gradient), on utilise des algorithmes qui garantissent une optimisation globale : Utilisation des techniques stochastiques et déterministes. Mais on a alors le problème de temps de convergence qui rentre en jeu. Approche stochastique proposée par Baba (84) donne une convergence garantie avec une probabilité de 1 sur les domaines compacts.

Barhen & Al. (86) ont proposé un algorithme de descente globale. L'optimisation est la solution d'un système d'équations différentielles déterministe ou l'on doit optimiser une fonction E(w) et les poids sont les variables d'état du système. Cet algorithme permet de sortie des minima locaux par une modification de la fonction de coût en cours d'apprentissage.

Chao et Al. ont proposé l'algorithme du "Magic Hair Brushing" qui modifie les dynamiques du système à travers le déformation du champ de gradient afin d'éliminer les minima locaux en préservant la structure global de la fonction.

Algorithme de l'attracteur terminal par De-Wang & Hsu, introduit un gain variant dans le temps dans la règle de modification des poids. Ceci cause un attracteur terminal. Malheureusement la convergence vers un minimum global n'est pas garantie.

Développement et élagage du réseau Les caractéristiques d'un bon réseau de neurones et de son algorithme d'apprentissage est d'avoir une bonne convergence et une capacité de généralisation satisfaisante. Les chercheurs ont développé des algorithmes d'élagage dans lesquels on entraîne un réseau plus grand que nécessaire afin d'avoir une bonne capacité de généralisation et ensuite on élague les connexions qui n'affecte pas de manière significative l'apprentissage. Ces algorithmes peuvent être classés en 2 catégories suivant 2 méthodes: La méthode sensitive, qui modifie un réseau entraîné avec une structure donnée en estimant la sensibilité de l'erreur en fonction d'un élément enlevé (poids ou neurone). Ensuite elle enlève l'élément de moindre effet. La méthode en terme de pénalité (term penalty method) qui modifie la fonction de coût telle que la propagation (basé sur cette fonction) conduit les poids superflus à zéro. Même si les poids ne sont pas enlevés le réseau réagit comme un réseau plus petit.

A l'opposé de l'élagage se trouvent les méthodes qui consistent à faire évoluer le réseau en l'agrandissant. On part d'un petit réseau et on lui rajoute des neurones et des connexions. Ces méthodes garantissent une bonne convergence mais une moins bonne généralisation.

Diviser et régner : une architecture modulaire Un moyen remarquable de passer outre les problèmes d'apprentissage est de faire appel aux architectures modulaires. (basé sur l'aspect biologique du cerveau). Ces nouveaux réseaux ont une grande capacité de généralisation. Pour passer d'une structure globale à un réseau modulaire il faut passer par trois étapes :

- Décomposer les tâches en sous-tâches
- Organiser convenablement le R.N. en considérant les sous-tâches
- Réaliser la communication entre les modules.

A l'opposé des structures globales le R.N. modulaire calcule sa sortie en ne faisant évoluer qu'une partie de sa structure. Il s'agit là d'un calcul local. En effet chaque module est un système indépendant qui interagit avec les autres dans la structure entière. Chaque module peut être considérer comme effectuant une fonction.

Grâce à ces caractéristiques ces réseaux peuvent réduire l'architecture d'un réseau MLP équivalent pour la même tâche. La décomposition en module permet de mieux comprendre le fonctionnement interne d'un réseau. De plus il y a la possibilité d'architecture modulaire hybride : différents fonctions d'activations différentes structures dans les modules. Et même différent algorithme pour l'apprentissage des différents modules.

1.8.4.1 Apprentissage basé sur la connaissance a priori du système

En intégrant soit dans la structure, soit dans la matrice de poids, une connaissance a priori du système, on diminue de façon non négligeable les problèmes calculatoires et d'apprentissage.

1.8.4.2 Apprentissage avec supervision

En se basant sur l'analogie "prof $\Leftrightarrow$ étudiant" la tâche à apprendre par le réseau doit être apprise progressivement avant d'atteindre sa cible. Il faut passer par différentes étapes qui serait dans le cas de l'étudiant des "leçons".

1.9 Universalité des réseaux récurrents complètement interconnectés

1.9.1 Introduction

Il est bien connu que les réseaux MLP avec une seule couche cachée sont des approximateurs universels de fonctions. Dit autrement n'importe quelle fonction peut-être approximée avec suffisament de précision s'il y a assez de neurones sur la couche cachées.

Il est alors possible de supposer que les RNR, d'après leurs capacités dynamiques seront eux aussi des approximateurs universels des systèmes dynamiques. Actuellement on sait qu'un réseau de type feedforward de deux couches avec des récurrences retardées peut approximer n'importe quel système dynamique. Ce type de réseau récurrent est utilisé dans l'apprentissage de séquences.

Une autre approche pour modéliser des systèmes dynamiques est d'utiliser des réseaux complètement connectés récurrents. On va donc voir que n'importe quel système dynamique peut être approximer par un réseau récurrent complètement connecté dans lequel les unités travaillent en parallèle avec le même retard. Dans ce réseau la représentation d'état est distribuée dans l'architecture.

1.9.2 Réseau de neurones comme approximateurs universels

Cas des réseaux non-bouclés La propriété d'approximation universelle énonce que pour toute fonction déterministe suffisament régulière, il existe un réseau de neurone non-bouclé possédant une couche cachée et un neurone de sortie linéaire qui réalise une approximation de cette fonction et de ses dérivées successives avec une précision arbitraire. Ce qui fait le différence des réseaux de neurones avec les autres méthodes d'approximation (série de Fourier, fonctions splines, polynômes...) c'est leur parcimonie : pour l'approximation d'une fonction ils auront beaucoup moins de paramètres que les autres approximateurs. [Hornik 94].

Cas des réseaux bouclés La propriété d'approximateurs universels pour des réseaux de neurones récurrents est plus difficiles. D'après Doya [Doya 93] un réseau récurrent complètement interconnecté permet d'approximer n'importe quelle fonction ou système dynamique continu ou discret. Si on a le système $\left\{\begin{array}{l} x_p(k+1) = f\left(x_p(k), u(k)\right) \\ y_p(k) = g\left(x_p(k)\right) \end{array}\right\}$ où f et g sont des fonctions continues. Il existe bien un réseau récurrent qui mis sous sa forme canonique représentera dans sa partie MLP une représentation de la fonction f par un sous-réseau et une représentation de g par un autre sous-réseau. Mais cela n'implique pas que le réseau approxime le système dynamique en tous ses points.

Sontag [Sontag 93] énonce une propriété d'approximation universelle pour les réseaux récurrents : Pour tout système dynamique, pour toute précision désirée ϵ , pour un intervalle de temps fini $[0, T]$, pour des entrées $u(\cdot) : [0, T] \rightarrow U \subseteq \mathcal{R}^{n_u}$ et un état initial $x(0) \in X \subseteq \mathcal{R}^{n_x}$, il existe un réseau bouclé qui approche le comportement entrée-sortie du système avec la précision ϵ sur l'intervalle $[0, T]$ et sur les ensembles U et X. On restreint donc les propriétés d'approximation sur un intervalle d'entrée, d'état et de temps. Il se peut donc que le réseau ne reflète pas toutes les configurations du système réel.

1.9.3 Représentation d'un système dynamique par un RNR interconnecté

On considère l'approximation d'un système dynamique d'ordre m, $x(t+1) = F\left(x(t), u(t)\right)$ et $x(t) \in \mathcal{R}^m$ où $u(t) \in \mathcal{R}^1$ est la vecteur d'état, est le vecteur d'entrée, $F() : \mathcal{R}^m \times \mathcal{R}^1 \rightarrow \mathcal{R}^m$ est la fonction du système. On doit alors approximer la fonction F apr un réseau de neurone récurrent du type $y(t+1) = S\left(Wy(t) + Vu(t)\right)$ où $y(t) \in \mathcal{R}^n$ représente les sorties des unités, W et V sont les matrices de poids des connexions. $S(\cdot) : \mathcal{R}^n \rightarrow \mathcal{R}^n$ est une fonction de type sigmoïde. A première vue ce type de réseau ne pourrait implémenter qu'un classe limitée de réseau d'ordre n. D'après le théorème d'universalité pour un réseau à deux couches [Hornik 91], il est possible d'approximer n'importe quel système ou fonction F par un réseau à deux couches avec m entrées linéaires et n unités cachées (fonction de sortie sigmoïdale) avec n assez grand. C'est à dire si on prend les matrices suivantes $W_1(n \times m), W_2(m \times n)$ et $V(n \times 1)$ on a

$W_2S(W_1x(t)+Vu(t)) \cong F(x(t),u(t))$. D'où on a $x(t+1) = W_2S(W_1x(t)+Vu(t))$ et si on prend $y(t+1) = S(W_1x(t)+Vu(t))$ comme nouvelle variable d'état alors les dynamiques du système sont exprimées en y par $y(t+1) = S(W_1W_2y(t)+Vu(t))$ on retrouve la forme d'implémentation de notre réseau avec $W = W_1W_2$. Ceci est une conséquences de la linéarité des opérations sur les retards.

Notre réseau comporte alors des bouclages seulement sur les sortie de sa structure. Les dynamiques des états sont contenues dans la structure.

1.10 Capacité de généralisation des réseaux de neurones récurrents

1.10.1 Introduction

Dans les réseaux de neurones récurrents la notion de dynamique temporelle est implicite. Et de petits RNR peuvent remplacés avantageusement les MLP d'une taille plus importante pour la même tâche.

Le problème de l'apprentissage basé sur l'identification d'un séquence d'entrée/sortie comme pour la probabilité de convergence est du à la dimension de Vapnik-Chervonenkis. Si on possède une boite noire qui représente notre système, sa structure est inconnue mais on peut faire des expériences et récupérer des fichier E/S. On peut donc faire l'apprentissage de ces séquences par les différents algorithmes vus précédemment. Mais une question subsiste : Quel est la précision du réseau obtenu à simuler la boîte noire. Quel est la fiabilité de ce réseau en temps que prédicteur de la sortie de la boite noire pour des valeurs de commande prises au hasard. On entre bien dans le problème de la capacité de généralisation des R.N..

1.10.2 Le Risque de Généralisation

Même si on suppose que les échantillons d'apprentissage et les entrées futures ont la même distribution de probabilité, la boite noire ne peut être exactement modélisée par le R.N.. Si les séquences d'apprentissage sont suffisamment riches alors l'erreur de généralisation sera minimale et se rapprochera de l'erreur d'apprentissage.

Il faut donc quantifier ce que sont des données d'apprentissage suffisamment riches pour que l'erreur de généralisation soit aussi proche que possible de l'erreur d'apprentissage. Ces techniques utilisent les lois uniformes des grands nombres. Le nombre de données nécessaires à l'apprentissage pour que le réseau soit fiable en prédiction augmente avec la dimension de l'espace d'état du réseau utilisé comme modèle.

Les réseaux récurrents avec un nombre fini de neurones ont la capacité d'approximateurs universels. Ils peuvent donc implémenter n'importe quelle fonction à laquelle on associe un ensemble d'apprentissage.

Comme défini plus haut le problème de l'apprentissage est de trouver la fonction f dans un ensemble de fonction F avec le risque de se tromper minimal que l'on peut traduire :
$E(f) = \int |f(x) - f_0(x)\, P|$ devienne minimum. Ceci est mis en oeuvre en minimisant le critère : $\hat{E}(f) = \sum_i |f(x_i) - f_o(x_i)|$.
Tant que la fonction f est corrélée aux exemples le risque est quasi nul. Cependant si la

capacité de l'ensemble de fonction F est finie et les exemples d'apprentissage indépendants et uniformément distribués, il existe une borne théorique pour le nombre minimal d'échantillons d'apprentissage nécessaires afin que l'erreur sur l'ensemble d'apprentissage soit représentatif pour l'ensemble complet des entrées.

Cette borne théorique est donnée par la dimension de Vapnik-Chervonenkis.(VC)

1.10.3 La dimension de Vapnik-Chervonenkis.(VC)

Cette dimension permet de mesurer les capacités des familles de fonctions. Dans le cas des réseaux neuronaux on peut caractériser la capacité d'une architecture particulière qui sera la famille de fonction, et les différentes fonctions correspondent aux différents réseaux qui peuvent être obtenu par la modification des poids.

On peut vite remarquer que la convergence sera d'autant plus lente que la dimension de VC sera grande.

1.10.3.1 Conditions nécessaires et suffisantes pour la Convergence uniforme

Cette condition se résume en "la cardinalité de F doit être finie". C'est un condition peut applicable aux réseaux de neurones récurrents et pour lesquels elle n'est jamais vérifiée.

Le cas ou la cardinalité de F est infinie est plus complexe. Il faut attendre les résultats de Vapnik-Chervonenkis pour avoir une condition suffisante pour pouvoir affirmer la convergence uniforme. Depuis Erhenfeucht & Al. ont montré que cette condition est également nécessaire. Le but est d'arriver à trouver une condition pour vérifier $\lim_{n\to\infty} Pr\{\max_F |Pr(F) - f(F)| \geq \epsilon\} = 0$ Vapnik et Chervonenkis ont été les premiers a donner une inégalité qui vérifie cette limite. Elle a évolué pour avoir la forme $Pr\{\max_F |Pr(F) - f(F)| \geq \epsilon\} \leq 4\Pi_F(2n)e^{-n\epsilon^2}$ où Π_F est la fonction de croissance de F et cette fonction vérifie $\Pi_F(k) = \left(\frac{ek}{D_F}\right)^{D_F}$ où $k \leq D_F$. D_F est la dimension de Vapnik-Chervonenkis.

1.10.3.2 Définition

F est un sous-ensemble de l'ensemble des parties de X, ceci en faisant correspondre à chaque fonction F l'ensemble des éléments de X qu'elle recouvre. Pour plus de simplicité l'ensemble recouvert par F sera également noté F. La VC dimension d'une classe de parties permet d'exprimer la capacité de cette classe.

Définitions et notations La trace de F sur un sous-ensemble S de X est définie par $T_f(S) = \{S \cap F / F \in \mathcal{F}\}$. le cas ou la trace de F est égale à l'ensemble des parties de S, on dit que S est pulvérisé.

La fonction de croissance attachée à F est définie par $\Pi_{\mathcal{F}} = \max_{\{S_k\}} |T_{\mathcal{F}}(S_k)|$ où S_k est une partie de X ayant k éléments et ou $\{S_k\}$ est l'ensemble de toutes ces parties. La fonction de croissance k est donc le cardinal de la plus grande trace que l'on peut obtenir à partir d'une partie de X à k éléments. Dans le cas ou il existe une partie S_k pulvérisée on a $\Pi_{\mathcal{F}}(k) = 2^k$

La dimension de Vapnik-Chervonenkis de $\mathcal{F}$ est définie par : $D_{\mathcal{F}} = \max\{k / \Pi_{\mathcal{F}}(k) = 2^k\}$ C'est donc le plus grand entier k tel qu'il existe une partie S_k pulvérisée. C'est donc le plus grand nombre de points de X dont on puisse apprendre sans erreur toutes les dichotomies avec $\mathcal{F}$. L'importance de cette dimension est pressentie pour l'apprentissage par Cover (1965). L'idée est que lorsque le nombre d'échantillons est égal ou du même ordre que la VC dimension de $\mathcal{F}$, alors le fait de trouver une fonction particulière de $\mathcal{F}$ séparant parfaitement les points de l'ensemble d'apprentissage, est sans signification sur la structure réelle des données et donc sur les performances réelles de la fonction.

1.10.4 Estimation de l'erreur de généralisation

Il existe plusieurs méthodes d'estimation de l'erreur de généralisation

Statistique de l'échantillon simple Le critère Bayesien de Schwarz fonctionne bien pour les réseaux de neurones. Le principe de longueur de description minimum de Rissanen se rapproche de la méthode SBC.

Validation par échantillon séparé La méthode la plus utilisée consiste à calculer l'erreur de généralisation sur un ensembles d'échantillons de test qui n'ont pas été utilisé lors de l'apprentissage. Cet ensemble doit être représentatif des cas à généraliser. A la suite de l'apprentissage on utilise l'ensemble d'échantillons de test sur le réseau. Et l'erreur obtenu donne une estimation de l'erreur de généralisation. Cette méthode est réductrice car elle ne prend en compte que les données des ensembles d'apprentissage et de test. Ici on utilise seulement un seul sous-ensemble d'échantillons.

Validation croisée La validation croisée est une amélioration de la méthode précédente qui permet d'utiliser les donnée de l'apprentissage. Le problème de cette méthode est que l'on ré-entraîne le réseau plusieurs fois. Cette méthode est basée sur le rééchantillonnage des données. Elle est utilisée pour choisir les meilleurs modèles parmi un ensemble.

Dans une validation croisée de k-ensemble on divise l'ensemble des échantillons en k sous-ensemble sensiblement de même taille. On ré-entraîne le réseau k fois en laissant de côté à chaque fois un ensemble mais en utilisant cette échantillon pour le calcul de l'erreur de généralisation. Si k est égal au nombre d'échantillon on appel cela le **"leave-one-out cross-validation"**. Une version plus évoluée de la validation croisée est le leave-v-out cross-validation" qui entraîne de plus gros calcul puisque l'on utilise tous les sous-ensembles possibles de l'ensemble d'échantillons dont la cardinalité est v. Choix de la validation croisée
La validation croisée "leave-one-out" fonctionne bien avec les fonctions d'erreurs continues (fonctions carrées) mais est moins bonne avec les fonctions non-continues (valeur absolue). Dans ce cas on préfèrera la méthode avec k ensembles. On peut rencontrer des problèmes avec cette méthode de validation lorsqu'on l'utilise avec des méthodes de séléction : choix du nombre d'unités cachées... . Là encore il s'agit d'un problème de continuité dans les unités (neurones récurrents).

BootStrapping Cette méthode est une évolution de la validation croisée qui donne souvent une meilleure estimation de l'erreur de généralisation. La forme la plus simple de Bootstrapping au lieu d'analyser des sous-ensembles de données on analyse des sous-échantillons de données. Chaque sous-échantillon est un échantillon aléatoire avec remplacement de l'échantillon complet.

1.11 Identification et commande de système par réseau de neurones récurrents

1.11.1 Introduction

Dans le cas linéaire ou pseudo linéaire (linéarisation avec peu de perte de performance) on utilise les bases solides de l'identification et de la commande grâce aux outils de l'algèbre linéaire. Mais la diversité des systèmes non-linéaires empêche une résolution simple pour leurs commandes.

La capacité des réseau de neurones à modéliser des systèmes non-linéaires est largement exploitée pour la synthèse de contrôleur (Cybenko, Lapedes-Farber). Mais des problèmes subsistent :

- Pas de vois systématique pour déterminer la structure pour modéliser un système donné
- La rétropropagation n'assure pas la convergence de l'apprentissage
- Le réseau de neurones est une boite noire

De plus c'est le Perceptron multicouche ou MLP qui est le plus souvent utilisé. Ce modèle de réseau de neurones n'est pas dynamique. On le rend dynamique en lui administrant en entrée, ses entrées et ses sorties aux instants précédents. On peut alors l'interprété comme un modèle **NARMAX** (Non-linear Auto-Regressive Moving Average with eXogenous inputs). Mais on réduit alors de beaucoup les dynamiques contenue dans ce type de réseau par rapport à un réseau récurrent.

1.11.2 Approche de commandes neuronales

La plupart des méthodes de commandes neuronales sont basées sur la commande inverse.

1.11.2.1 Technique de commandes neuronales basées sur le système inverse

Commande neuronale basée sur le copiage : l'imitation ou le copiage est un caractéristique des systèmes biologiques. La première méthode de commande neuronale consiste à dupliquer le contrôleur humain. Cette technique a été appliquée au pendule inversé par Widrow et Smith en 1964. Cette méthode se révèle utile pour le contrôle de processus industriel.

L'apprentissage consiste en un apprentissage de la correspondance entre les informations reçues par le contrôleur humain et la commande d'entrée. La principale difficulté de cette méthodes est de déterminer les informations utilisées par le contrôleur humains (Cf. figure 1.22 et 1.23).

1.11.2.2 Apprentissage de commande neuronale directe inverse

Cette commande inverse a pour but la commande du système en utilisant ses dynamiques inverses. Le réseau reçoit en entrée les sorties du système (Cf. figure 1.24). L'apprentissage est effectué en donnant au contrôleur une séquence d'entrée u. Bien qu'il y ait un retour des sortie du système sur l'entrée du contrôleur le système est globalement en boucle ouverte car on ne tient pas compte de la sortie du système et la sortie désirée. C'est à dire que le réseau de neurone ne fait rien pour rattraper la déviation entre la sorti réelle et la sortie désirée.

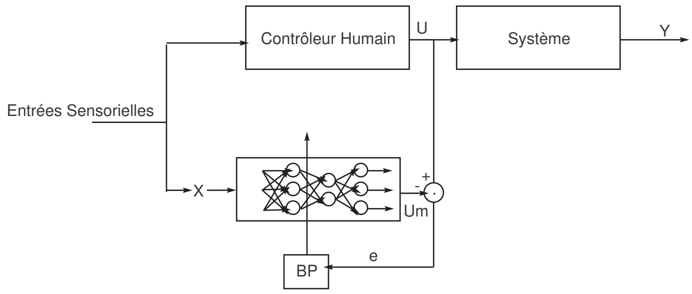

FIG. 1.22 – Copiage de l'expert humain par le neuro-contrôleur sans boucle de retour

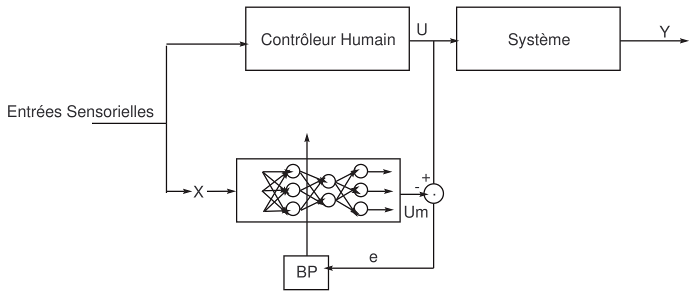

FIG. 1.23 – Copiage de l'expert humain par le neuro-contrôleur avec boucle de retour

Le plus dur reste donc de déterminer la séquence des entrées u du réseau de neurones. Le système doit être porté dans son espace de travail ou le contrôleur devra effectuer la correction. Ceci est très difficile a assurer sans une connaissance a priori très importante du système. (Widrow et Stearn 1985)

1.11.2.3 Apprentissage de la commande neuronale inverse spécialisée (Psatis & Al.)

Ils proposent l'"apprentissage spécialisé inverse". C'est un approche de commande neuronale dirigée par un but. C'est la différence fondamentale avec l'approche décrite précédemment. Le réseau opère un apprentissage en ligne afin de minimiser un critère de commande (idem le critère d'apprentissage) $e_y = r - y$ (Cf. figure **??**)

L'erreur e_y n'est pas directement relation avec la sortie du contrôleur u comme cela

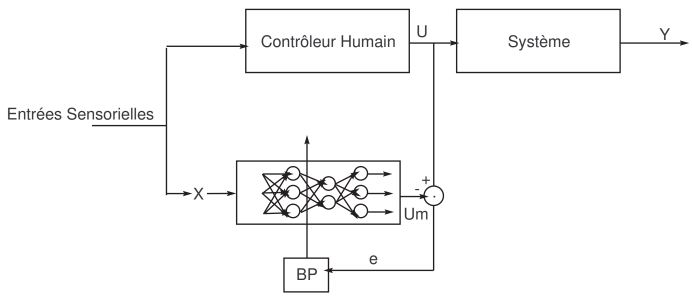

FIG. 1.24 – Architecture de commande neuronale inverse directe

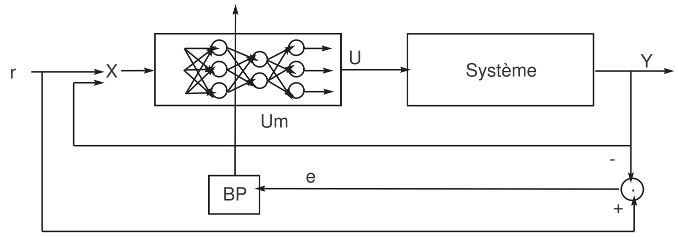

Fig. 1.25 – Architecture de commande inverse spécialisée

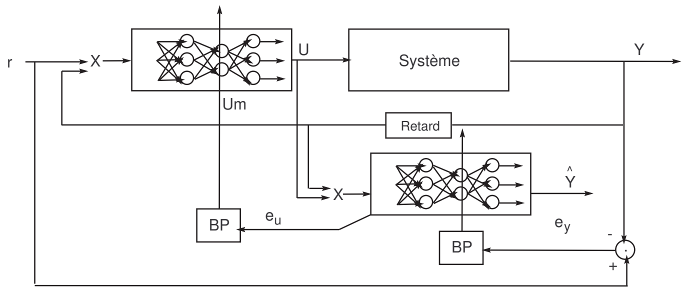

Fig. 1.26 – Architecture de commande neuronale avec rétropropagation à travers le temps

devrait être. De ce fait cette approche ne peut mener vers une solution optimale. D'ici e_y peut être totalement non-corrélée à au critère de performance d'apprentissage de la commande neuronale $e_u = \bar{u} - u$.

1.11.2.4 Apprentissage de contrôleur neuronal par rétropropagation à travers le temps

Pour éviter le problème de la commande spécialisée inverse, des chercheurs (Narendra, Jordan, Rumelhart, Nguyen) ont développé indépendamment la rétropropagation à travers le temps. Le problème de l'apprentissage de la commande spécialisée inverse est que l'erreur de performance $e_y = r - y$ n'est pas fiable. En effet elle n'est pas apparentée directement a la sortie u de neuro-contrôleur. Un erreur fiable d'apprentissage du neuro-contrôleur serait donnée par $e_u = \bar{u} - u$. Les chercheurs ont donc proposé d'émuler eu en utilisant la rétropropagation de e_y a travers le modèle du système. Dans ce sens l'erreur e_y rétropropagée vers la couche d'entrée du modèle feed-forward devrait correspondre à l'erreur $e_u = \bar{u} - u$ (en supposant que l'entrée u a un effet linéaire sur le système). L'apprentissage de commande peut donc être réalisées dans différents cas.

On peut remarquer que le modèle neuronal du système est entraîné avant le contrôleur (Cf. figure 1.26). Bien que l'erreur soit rétropropagée dans le modèle neuronal ce dernier n'est pas adapté à l'apprentissage du contrôleur. Il peut être utile dans certaines utilisations de choisir la sortie du modèle neuronal $\hat{y}$ plutôt que la sortie réelle (lorsque le système ne peut être utiliser lors de la phase d'apprentissage du correcteur).

L'approche étudiée par Narendra est un peut différente car il injecte un modèle de référence dans la structure pour donner une sortie transitoire voulue $\bar{y}$ au système plutôt qu'une sortie désirée r qui ne dépend pas du temps. C'est alors cette sortie voulue que l'on injecte dans le neuro-contrôleur (Cf. figure 1.27).

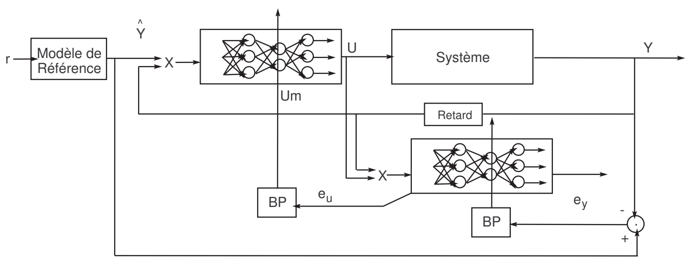

FIG. 1.27 – Architecture de commande neuronale avec rétropropagation à travers le temps et modèle de référence

1.11.2.5 Commande biologique et commande neuronale basées sur le système inverse

Les méthodes de commande neuronale basées sur le système inverse qui sont appliquées le plus souvent en robotique ont de sérieux inconvénients. Le plus important concerne le manque de retour d'informations. Si le neuro-contrôleur n'est pas l'inverse exact du système alors il se peut que que la commande soit sans effet ce qui ne surviendrait pas s'il y avait un retour d'information.

Mais il n'est pas simple d'apprendre l'inverse des dynamiques très retardées d'un système bruité. De plus même si le modèle neuronale est assez précis on se rend compte qu'il est très sensible au bruit.

Le second problème vient du fait que tous les systèmes ne possèdent pas d'inverse. Ces problèmes restreignent donc les domaines d'application de cette méthode. Le retour d'information (feedback ou boucle de retour) possède des propriétés très importantes et doit faire partie d'un processus de commande. Mais alors le problème se complique car a boucle de retour peut entraîner des instabilités.

Dans les systèmes humains on trouve différents contrôleurs. Il semblerait que les systèmes en boucle ouverte et boucle fermée jouent des rôles différents dans la commande humaine. Pour les commandes d'actions rapides les systèmes en boucles ouvertes sont mieux adaptés. Mais pour les actions précises se sont les systèmes bouclés qui sont le mieux adaptés.

Une remarque importante concernant les systèmes humains est qu'avec de l'apprentissage les systèmes de commandes biologiques développent un modèle interne précis du système à commander. Les systèmes humains ont une grande capacité de généralisation par rapport à des situations inconnues.

1.11.2.6 Commande prédictive basée sur le modèle neuronal

Le commande prédictive a été développée principalement pour surmonter le problème des retards dans les systèmes complexes qui ne réagissent pas immédiatement à l'application d'une commande. Le retard du système est le temps que le système met pour réagir.

Le contrôleur ne change pas d'état pendant ce retard. Ceci peut induire des effets indésirables. Une solutions consiste donc à attendre le temps du retard et seulement après de lancer le correcteur. La deuxième solution et la meilleure consiste à utiliser u modèle prédictif du système qui prédira la sortie du système à la suite du retard. A ce moment on peut donc anticiper la commande. Ceci est donc la base de la commande

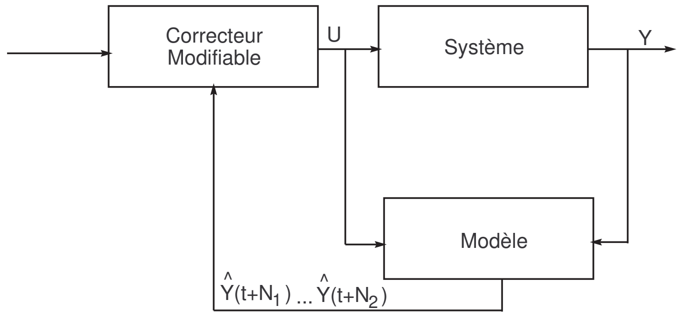

FIG. 1.28 – Structure générale d'un correcteur basé sur un modèle prédictif

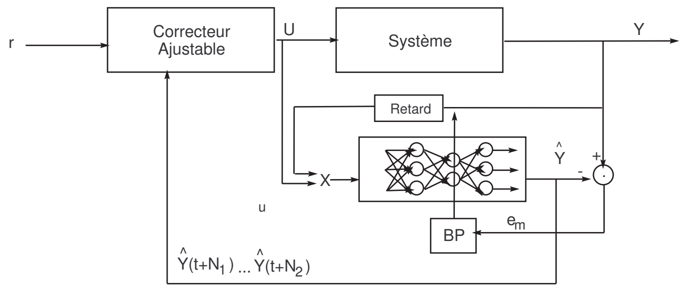

FIG. 1.29 – Structure générale d'un correcteur basé sur un modèle prédictif neuronal

prédictive. On détermine la commande en fonction de la sortie estimée du système par le modèle. Cette idée a été généralisée par Clark & Al. 1987 et appliquée aux systèmes à dynamiques complexes et applelé le "Generalized Predictive Control" GPC. C'est la méthode principale pour la détermination de correcteur basé sur la commande prédictive (Cf. figure 1.28).("Model Based Predictive Control" MBPC). Ce type de correcteur est composé de trois partie :

- Le système
- Le modèle
- Une fonction d'optimisation

Le modèle prédit les sorties du système. Par rapport à ces sorties, la fonction d'optimisation détermine la séquence de commandes la plus adaptées au système. La fonction d'optimisation prend la forme d'une fonction de coût quadratique

$$J(N_1, N_2, N_u, t) = \sum_{i=t+N_1}^{t+N_2} \mu(i) \left[\bar{y}(i) - \hat{y}(i)\right]^2 + \sum_{i=t}^{t+N_u} \lambda(i) \left[\Delta u(i-1)\right]^2$$

où $\bar{y}$ est la sortie désirée, $\hat{y}$ est la sortie du modèle, $\Delta u(i) = u(i) - u(i-1)$ est le pas d'entrée du procédé, N_1 etN_2 définissent l'horizon de prédiction, N_u est l'horizon de commande et λ et μ sont des constantes qui pondèrent le comportement futur.

La caractéristique principale de cette méthode est que la conception d'un correcteur n'est pas nécessaire. Les réseaux de neurones MBPC ont été largement utilisés dans la modélisation de procédés. (Rusnak & Al. 1996, Gomm & Al.) Ceci donne un correcteur basé sur un modèle neuronal prédictif (Cf. figure 1.29). Les réseaux de neurones peuvent aussi être utilisé comme correcteurs (Hortega & EF 1996).

1.11.2.7 Remarque sur es techniques de commandes neuronales

La plupart du temps Les MLP perceptrons multicouches sont utilisés avec tout ce qu'ils comportent comme défaut. On ne peut déterminer avec précision les propriétés de commandes et de modélisation a cause de leur concept de boite noire. Ce qui est très contraignant car on doit souvent faire référence au propriétés de controllabilité et commandabilité pour établir sa stabilité et sa robustesse.

De plus les méthodes d'apprentissage comme le gradient sont lentes et leurs convergences n'est pas prouvée. Ceci est du au minima locaux et aux zones mortes.

1.11.2.8 Commande par réseaux de neurones modulaires sélectionnés

Dans les réseaux modulaires qui sont facilement modifiables on peut réutiliser les modules pour différentes tâches au lieu d'apprendre les parties commune de toutes les tâches. La difficultés principale dans la conception de réseaux de neurones modulaires est la décomposition de la tâche globale en sous-tâches facilement codables sous forme de réseaux de neurones. La modélisation en sous-tâches doit donc être un processus interactif entre les capacités de calcul de chaque module et la complexité des sous-tâches.

Peu d'algorithmes peuvent faire la décomposition en sous-tâches de manière systématique. Ces algorithmes se caractérisent donc par des systèmes de portes dans l'architecture. A chaque instant le système de sélection de modèle opère une sélection des modules les plus aptes pour le calcul. Ces réseaux de neurones modulaires sont appelés les "Gated Modular Neural Networks GMNN". On trouve dans la littérature plusieurs types de réseaux de neurones que l'on peut distingués par le critère utilisé pour la décomposition de l'espace d'entrée en sous-espace.

Les système de sélection de modèle par groupage spatial Les GMNN définit ici opèrent un groupage spacial des entrées de façon à sélectionner et répartir les vecteurs d'entrées en différents modules. L'algorithme le plus aboutit en ce sens est de le "LMN" "Local Model Network" (Poggio & Girosi, 1990, Stockbro & Al. 1991), lequel a été étendue par Johansen & Foss 1992 pour la modélisation et la commande. Le LMN est en fait une généralisation des réseaux neuronaux à fonctions de bases développer au départ pour des tâches de classification. De plus il possède des ressemblances frappante avec le modèle flou de Takagy & Sugeno.

Architecture d'un réseau à modèle local Ce réseau consiste en un réseau monocouche et un système de (porte) sélection de chemin souvent composés par des fonctions radiales de bases (RBF). Les différents modèles locaux sont activés par un vecteur d'entrée X qui varie dans le temps, ce qui correspond souvent au vecteurs des paramètres d'un modèle **NARMAX** du système. Chaque modèle local peut être considérer comme comme un modèle NARMAX local du système (Cf. figure 1.30).

La sortie de chaque modèle sélectionné est pondérée par l'activité w des RBF connectée. De là, la sortie du réseau est la somme des sorties pondérées de tous les modèles.

A chaque instant le système de sélection de modèle opère un groupage hypersphérique de l'espace d'entrée afin de donner le vecteur d'entrée Φ aux meilleurs modèles. Le vecteur Φ est le plus souvent un sous-ensemble du vecteur d'entrée X. Le groupage est effectué par des fonction radiales de bases (RBF) qui ont pour caractéristique un centre et une largeur. L'activation d'une RBF est faite par (Cf. figure **??**)

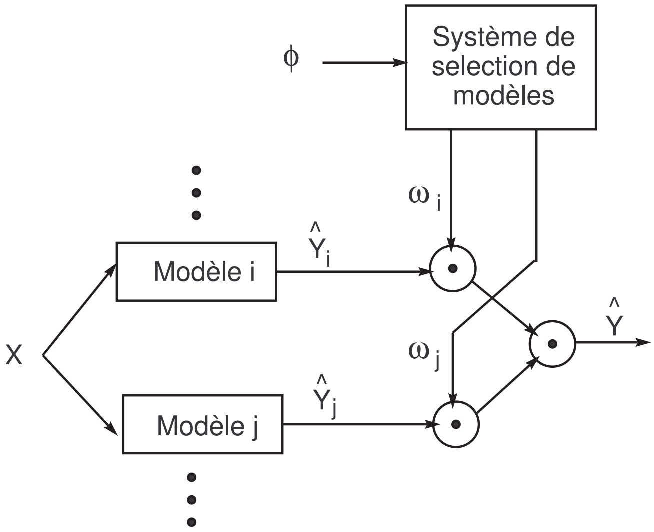

FIG. 1.30 – Le reseau à modèles locaux

- Le calcul d'une distance Euclidienne entre les coordonnées spatiales du vecteur clusterisé $\Phi = [\Phi_1\Phi_2]$ les coordonnées du centre $C_i = [c_{i1}c_{i2}]$ de la fonction radiale de base RBF_i.
- Un potentiel d'activation p_i est déterminé en fonction de la largeur σ_i de la RBF_i $p_i = \frac{(\Phi-C)^2}{\sigma^2}$ où $C_i = [c_{i1}c_{i2}]$ et $[\Phi_1\Phi_2]$ snt respectivement les coordonnées spatiales du centre de la RBF_i et l'entrée clusterisée.

Il est activé par un vecteur d'entrée φ qui est le plus souvent un sous-ensemble du vecteur d'entrée X qui active le réseau de modèles locaux. Le système de sélection de modèle local est composé de RBF utilisée pour le groupage de l'espace d'entrées (Cf. figure 1.32 et 1.33). Les sorties de chaque RBF pondèrent (ω) la sortie du modèle local connecté.

e^{-p_i} est utilisé pour obtenir un groupage non-linéaire qui a des caractéristiques Gaussiennes. L'activation de la fonction radiale de base i est $\omega_i = e^{-p_i}$. ω_i sera proche de 1 si p_i est très petit et proche de 0 si p_i est très grand. L'idée est alors d'utiliser ω_i pour pondérer la sortie la sortie du modèle local
$\hat{y} = f_i(X) \cdot \omega_i$ où $f_i(X)$ fait référence au modèle local i **NARMAX**. La sortie du LMN est la somme des sorties des modèles locaux $y_{LMN} = \sum_{i=1}^{N} y_i \cdot \omega_i$ où N est le nombre de modèles locaux. En résumé on peut dire que le comportement d'un LMN est la décomposition de l'espace du système réel en sous-espace plus petit déterminer par des RBFs. Des modèles locaux simples sont utilisés pour décrire le système dans chacune de ces régions. Un modèle global est formé par l'interpolation des modèles locaux en utilisant une fonction d'interpolation w qui dépend de la condition de fonctionnement Φ (Johansen & Foss 1993).

Le principal avantage est de pouvoir facilement transformer le réseau à modèle local en réseau à régulateur local.

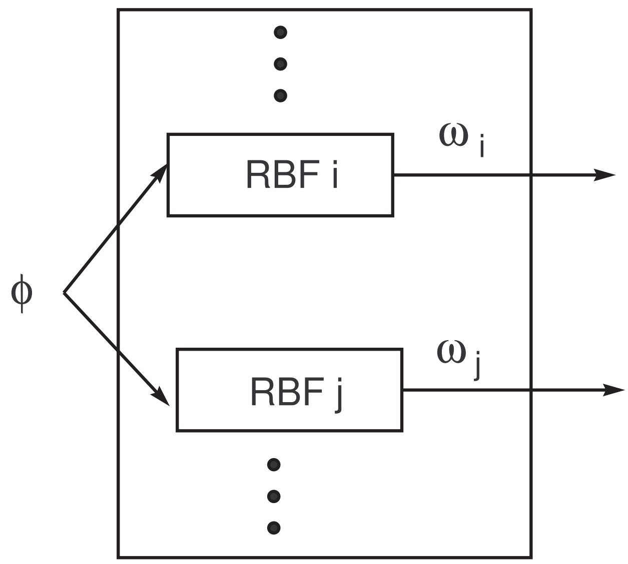

Fig. 1.31 – Le système de sélection de module

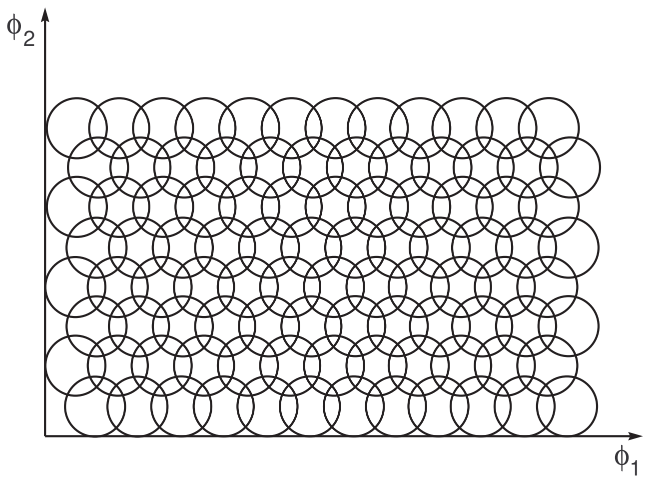

Fig. 1.32 – Exemple d'un groupage hypersphérique d'un espace d'entrée à deux dimensions

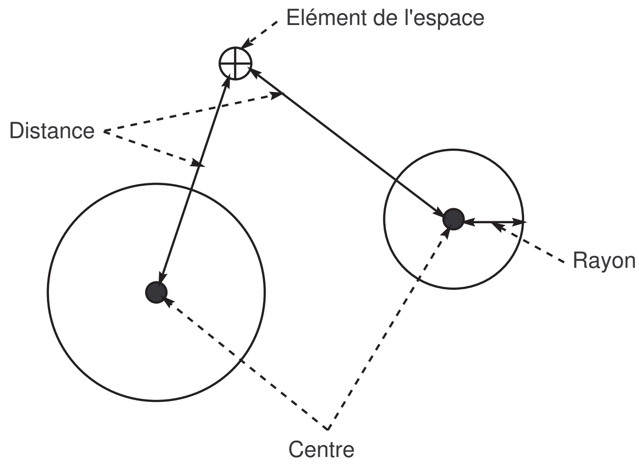

FIG. 1.33 – Description des composants de deux fonction radiales de base

1.11.2.9 Adaptation des paramètres d'un réseau à modèle local

Il y a deux types de paramètres dans un réseau à modèle local. Les premiers sont les paramètres des réseaux locaux, les seconds sont les paramètres des RBFs connectées aux réseaux locaux. L'optimisation des paramètres des réseaux locaux est assez simples et on opte le plus souvent pour une méthode type des moindres carrés.

1.11.2.10 Apprentissage des modèles locaux

Il y a deux façons d'effectuer l'apprentissage des modèles locaux. La première est de considérer le réseau dans son ensemble et de faire l'estimation des paramètres des modèles locaux. La seconde entraîne de faire l'estimation des paramètres de chaque modèle local indépendamment des autres. C'est l'apprentissage local. Il y a plusieurs avantage à utiliser l'apprentissage local plutôt que le global :

- Murray-Smith 1994 ont montré que l'apprentissage local était plus performant en terme de modélisation. L'apprentissage local rend plus précis le modèle local sur sa région ce qui rend le modèle global encore plus performant.
- Un apprentissage local est plus simple à mettre en oeuvre. Le modèle locaux étant linéaires l'apprentissage peut se faire à l'aide d'une simple méthode des moindres carrés. C'est l'interaction entre les différents modules qui provoque les non-linéarités dans la structure globale. Dans le cas d'une optimisation globale et de l'utilisation d'une descente de gradient de l'erreur la convergence vers un minimum global n'est plus assurée et le temps d'apprentissage s'allonge.

Bibliographie

[ANGELINE *et al.*,] P. J. ANGELINE, G. M. SAUNDERS, et J. B. POLLACK. An evolutionnary algorithm thar constructs recurrent neural networks. In *IEEE Transactions On Neurals Networks*.

[AUSSEM, 1995] A. AUSSEM. *Théorie et Applications des Réseaux de neurones Récurrents et Dynamiques à la Prédiction, à la modélisation et au Contrôle Adaptatif des processus Dynamiques.* PhD thesis, Université René Descartes Paris V, 1995.

[BENGIO *et al.*,] Y. BENGIO, P. FRASCONI, M. GORI, et G. SODA. Recurrent neuralnetworks for adaptative temporal processing.

[BIANCHINI et GORI, 1994] M. BIANCHINI et M. GORI. On the problem of local minima in recurent neural networks. In *IEEE Transactions on Neurals Networks*, volume 5, 1994.

[DAVALO et NAÏM, 1993] E. DAVALO et P. NAÏM. *Des Réseaux de Neurones.* EYROLLES, 1993.

[DOYA, 1992] K. DOYA. Bifurcations in the learning of recurrent neural networks. In *IEEE International Symposium on Circuits and Systems*, pages 2777–2780, 1992.

[DOYA, 1993] K. DOYA. Universality of fully-connected recurrent networks. In *IEEE Transactions on Neurals Networks*, 1993.

[GASCUEL,] O. GASCUEL. *Statistiques et méthode neuronales : La Dimension de Vapnik-Chervonenkis Application aux Réseau de Neurones.* Sciences SUP. Ed DUNOD.

[HAMMER,] B. HAMMER. On the generalization capability of simple recurrent networks.

[I.RIVALS *et al.*, 1994] I.RIVALS, L.PERSONNAZ, et G. DREYFUS. Modélisation, classification, et commande par réseaux de neurones : principes fondamentaux, méthodologie de conception et illustation industrielle. 1994.

[I.RIVALS *et al.*, 1995] I.RIVALS, L.PERSONNAZ, et G. DREYFUS. Les réseaux de neurones pour la modélisation et la commandes de procédés. Lavoisier Tec et Doc, 1995.

[JORDAN, 1996] M. I. JORDAN. *Neurals Networks.* CRC Handbook of Computer Science. CRC Press, Boca Raton, FL., 1996.

[KOIRAN et SONTAG, 1997] P. KOIRAN et E. D. SONTAG. Vapnik-chervonenkis dimension of recurrent neural networks. In *3rd European Workshop on Computational Learning Theory, LNCS 1208*, pages 223–237, 1997.

[LEVIN et NARENDRA, 1993] A. U. LEVIN et K. S. NARENDRA. Control of nonlinear dynamical systems using neural networks : Controllability and stabilization. In *IEEE Transactions on Neural Networks*, volume 4, 1993.

[NERRAND *et al.*, 1994a] O. NERRAND, P. ROUSSEL-RAGOT, D. URBANI, L. PERSONNAZ, et G. DREYFUS. Training recurrent neural networks : Why and how? an ilustration in dynamical process modelling. In *IEEE Transactions on Neurals Networks*, volume 5, 1994.

[NERRAND *et al.*, 1994b] O. NERRAND, P. ROUSSEL-RAGOT, D. URBANI, L. PERSONNAZ, et G. DREYFUS. Neural networks and non-linear adaptative filtering : Unifying concepts and news algorithms. In *Neural Computation*, volume 5, pages 165–197, 1994.

[PEARLMUTTER,] B. A. PEARLMUTTER. Dynamic recurrent neural networks. Rapport Technique CMU-CS-90-196.

[RONCO et GAWTHROP, 1995] E. RONCO et P. J. GAWTHROP. Modular neural networks : A state of the art. Rapport Technique csc-95026, 1995.

[RONCO et GAWTHROP, 1997a] E. RONCO et P. J. GAWTHROP. Neural networks for modelling and control. Rapport Technique csc 97008, 1997.

[RONCO et GAWTHROP, 1997b] E. RONCO et P. J. GAWTHROP. Neurals networks for modelling and control. Rapport Technique csc 97008, 1997.

[SONTAG, 1991] E. D. SONTAG. Neural nets as systems models ans controllers. Rapport technique, 1991.

[SONTAG, 1994] E. D. SONTAG. Recurrent neurals networks : Some system-theoric aspect. Rapport technique, 1994.

[SYED, 1994] O. SYED. *Applying Genetic Algorithms to Recurrent Neural Networks for Learning Network Parameters and Architecture*. PhD thesis, 1994.

[TSOI et BACK, 1994] A. C. TSOI et A. D. BACK. Locally recurrent globally feedforward networks : A critical review of architectures. In *IEEE Transaction On Neurals Networks*, volume 5, 1994.

[WILSON, 1993] W. H. WILSON. A comparison of alternatives for recurrent networks. In *Proceedings og the sixth Australian Conference on Neurals Networks (ACNN'93)*, pages 189–192, 1993.

[WILSON, 1995] W. H. WILSON. Stability of learning in classes of recurrent and feedforward networks. In *Proceedings of the sixth Australian Conference on Neurals Networks (ACNN'95)*, pages 142–145, 1995.