Apprentissage dans les réseaux récurrents pour la modélisation mécanique et étude de leurs interactions avec l'environnement

Nicolas Szilas

THESE

présentée pour obtenir le titre de docteur de
l'Institut National Polytechnique de Grenoble
(arrêté ministériel du 30 mars 1992)

Spécialité : SCIENCES COGNITIVES

par

Nicolas SZILAS

Apprentissage dans les réseaux récurrents pour la modélisation mécanique et étude de leurs interactions avec l'environnement

Soutenue le 6 décembre 1995 devant le jury composé de:

Bernard AMY	Examinateur
Claude CADOZ	Directeur de recherche
Giovanni DE POLI	Rapporteur
Christian JUTTEN	Président
Bernard VICTORRI	Rapporteur

Thèse préparée au sein du laboratoire LIFIA/IMAG et de l'association ACROE

THESE

présentée pour obtenir le titre de docteur de
l'Institut National Polytechnique de Grenoble
(arrêté ministériel du 30 mars 1992)

Spécialité : SCIENCES COGNITIVES

par

Nicolas SZILAS

Apprentissage dans les réseaux récurrents pour la modélisation mécanique et étude de leurs interactions avec l'environnement

Soutenue le 6 décembre 1995 devant le jury composé de:

Bernard AMY	Examinateur
Claude CADOZ	Directeur de recherche
Giovanni DE POLI	Rapporteur
Christian JUTTEN	Président
Bernard VICTORRI	Rapporteur

Thèse préparée au sein du laboratoire LIFIA/IMAG et de l'association ACROE

REMERCIEMENTS

Je tiens à remercier ici Messieurs Christian JUTTEN, Bernard VICTORRI et Giovanni DE POLI qui m'ont fait le plaisir et l'honneur de faire partie de mon jury de thèse.

Un grand merci à Bernard AMY et Claude CADOZ qui m'ont accueilli dans leurs équipes et ont encadré mon travail durant ces trois années. Merci aussi à Jean-Loup FLORENS et Annie LUCIANI pour leur travail d'encadrement au sein de l'ACROE.

Je remercie les chercheurs et thésards de l'équipe *Réseaux d'Automates* avec qui les échanges ont été nombreux, fréquents et enrichissants : Arnaud GIACOMETTI, Abderrahim LABBI, Maria MALEK, Bruno ORSIER, Sophie ROUZIER, etc. En particulier, merci à Eric RONCO, qui a amené beaucoup d'éléments à ce travail durant son stage de DEA.

Merci beaucoup à toute l'équipe *Informatique et Création Artistique*, en particulier Pirouz DJOHARIAN pour ses conseils précieux, Arash HABIBI pour son aide constante, Christophe DISCOURS, Jean-Loup FLORENS, Jamel NOUIRI, Olivier RAOULT et Claude UHL pour leur aide technique indispensable, sans oublier Ali BOUZOUITA, Benoît CHANCLOU, Olivier GIRAUD, Eric INCERTI, Stéphane JIMENEZ, etc.

Beaucoup de personnes ont relu des parties de ce mémoire : Bernard AMY, Claude CADOZ, Nathalie COLINEAU, Pirouz DJOHARIAN, Arnaud GIACOMETTI, Olivier GIRAUD, Arash HABIBI, Jean-Philippe MEIYE, Luc RODET, Sophie ROUZIER : je les en remercie chaleureusement.

Enfin, beaucoup d'autres personnes m'ont aidé à différents stades de ce travail, pour la plupart membres de l'association *InCognito* : merci à tous.

TABLE DES MATIÈRES

AVANT-PROPOS

Cette thèse a été préparée dans deux équipes du *Laboratoire d'Intelligence Artificielle et d'Informatique Fondamentale (LIFIA)* :

- l'équipe *Informatique et Création Artistique (ICA)* (directeur Claude CADOZ), liée à l'*Association pour la Création et la Recherche sur les Outils d'Expression (ACROE)*, travaillant sur la modélisation physique appliquée à l'image et au son.
- l'équipe *Réseaux d'Automates* (directeur Bernard AMY), travaillant sur l'apprentissage dans les réseaux neuronaux, et leur intégration avec les systèmes d'intelligence artificielle symbolique.

Les expérimentations concernant les applications musicales ont été menées à l'aide des équipements de l'ACROE, financés par le Ministère de la Culture et de la Communication, Direction de la Musique et de la Danse.

INTRODUCTION

Existe-t-il un rapport entre les comportements dynamiques d'un être humain et les comportements vibratoires d'un violon, et dans quelle mesure peut-on les comparer ?

Dans les deux cas, le chercheur qui étudie ces comportements rencontre de plein fouet le problème de la complexité : le fossé qui sépare les systèmes d'intelligence artificielle les plus évolués (quelle que soit leur nature) des capacités cognitives humaines est tout aussi grand que celui qui sépare les meilleures simulations d'instruments de musique des instruments eux-mêmes (il est erroné de croire qu'un violon soit d'une quelconque simplicité, sous prétexte que ce n'est qu'un objet matériel).

Pour aborder cette complexité, une approche émergente commune est celle des *réseaux* : réseaux neuromimétiques ou plus généralement réseaux connexionnistes d'un côté, réseaux d'automates cellulaires appliqués à la dynamique des fluides ou réseaux de masses ponctuelles interconnectées par des liaisons viscoélastiques de l'autre. Modèles très simplifiés des structures complexes qu'ils représentent, ces réseaux sont en fait souvent plus des métaphores, qu'il s'agisse de la "métaphore du neurone" ou de la "métaphore de l'atome".

Sur le plan de la simulation informatique, nous montrons que ces réseaux partagent un certain nombre de propriétés, ce qui permet d'établir une analogie entre modèles cognitifs et modèles physiques. Cette analogie des modèles, proposée par Claude Cadoz, permet d'aborder le problème technique suivant :

> *Les réseaux de modélisation physique du type de ceux qui sont développés à l'ACROE, (appelés réseaux CORDIS) possèdent de nombreux paramètres à régler (masses, raideurs, viscosités). Afin de déterminer les paramètres nécessaires pour reproduire un comportement physique donné, on cherche une technique permettant le calcul automatique de ces paramètres. Certains réseaux connexionnistes sont justement des réseaux capables d'adapter leurs paramètres pour reproduire un comportement donné. L'objectif est donc d'utiliser pour les réseaux de modélisation physique les techniques connexionnistes d'adaptation des paramètres.*

Ce travail exploite donc l'analogie des modèles pour effectuer un transfert technologique des réseaux connexionnistes vers les réseaux de modélisation mécanique. Nous sommes loin d'affirmer que cette analogie est obligatoire pour l'obtention des

algorithmes d'adaptation de paramètres : les algorithmes connexionnistes qui nous intéressent étant en fait fondés sur les théories de l'identification et de l'optimisation, on aurait pu partir directement de ces disciplines fondatrices pour arriver à nos fins.

Passer par les réseaux connexionnistes est en premier lieu un gain de temps. De plus, au delà des algorithmes d'adaptation de base, les recherches sur les réseaux connexionnistes ont développé beaucoup d'idées telles que les architectures évolutives, les réseaux modulaires, etc., qui sont autant d'idées adaptables aux réseaux de modélisation physique.

L'analogie des modèles fonctionne aussi dans l'autre sens : nous montrons que certaines spécificités des réseaux de modélisation physique peuvent conduire à la conception d'algorithmes originaux dans le domaine du connexionnisme.

Mais il est possible de mener plus loin l'analogie, en rapprochant directement l'apprentissage dans les réseaux de modélisation physique et l'apprentissage humain : dans cette étude, certaines idées issues de la psychologie cognitive sont utilisées pour aborder la question de l'action dans l'apprentissage des modèles physiques, en particulier en ce qui concerne l'apprentissage moteur.

Sur la figure 1 sont représentées les formes d'analogies possibles entre l'humain, l'objet physique, le réseau de neurones formels et le réseau de modélisation physique, en faisant figurer à gauche les entités qui constituent la *référence* pour les *modèles informatiques*, qui sont représentés à droite. Seule l'analogie "objet physique - réseau de neurones formels" (en pointillés) n'a pas été exploitée dans ce travail. Dans ce mémoire, on ne considère pas ces analogies au même titre que des modèles explicatifs, qui nous permettraient de comprendre les comportements physiques ou cognitifs : elles ont avant tout pour fonction de générer de nouvelles idées de modèles, car la confrontation entre deux domaines est un moteur essentiel de la créativité.

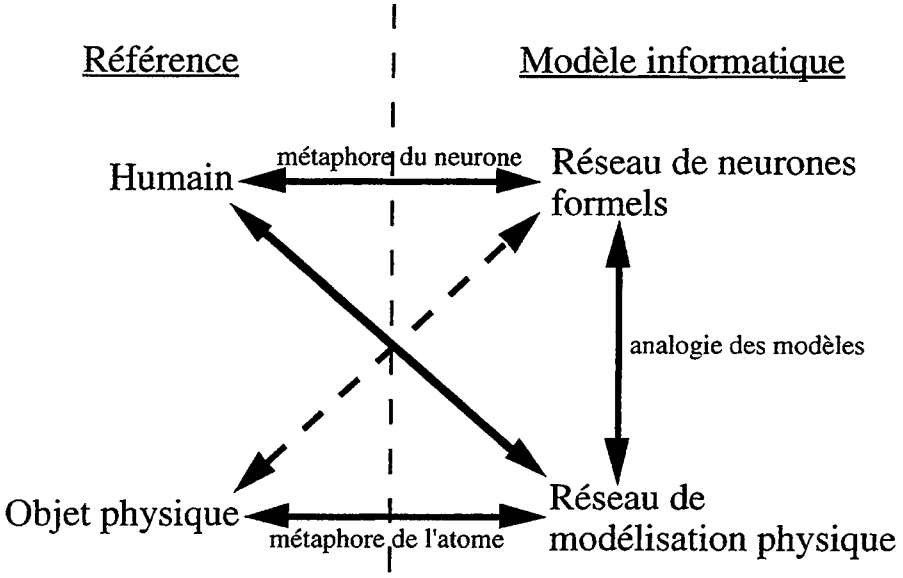

Figure 1. Différentes analogies possibles *entre les entités abordées dans ce mémoire*

Le document est organisé de la manière suivante :

- La première partie constitue une analyse détaillée des différents aspects du sujet. Tout d'abord est proposée une analyse du problème de l'identification de paramètres dans les modèles physiques utilisés en informatique musicale. Ensuite, les approches "réseaux" (modèles physiques, réseaux de neurones) sont comparées et unifiées. Ceci nous permet ensuite de poser le problème et de définir l'approche choisie pour le résoudre.
- La deuxième partie concerne les algorithmes d'apprentissage dans les réseaux de neurones récurrents pour ensuite les transcrire au cas des réseaux CORDIS développés à l'ACROE. A partir de là, des variantes originales sont proposées, fondées sur la mécanique. Des expériences de simulation des algorithmes proposés sont ensuite décrites et commentées.
- La troisième partie est consacrée à l'action dans l'apprentissage. L'idée d'action, issue dans cette étude de la problématique de la modélisation physique, est étudiée dans les champs de recherche des Réseaux de Neurones Formels et de la Psychologie Cognitive, du point de vue bibliographique, conceptuel et expérimental. Dans le cadre de l'apprentissage dans les modèles physiques, un système d'apprentissage coopératif entre l'homme et l'ordinateur est finalement proposé.

Ces trois parties parcourent les quatre objets d'étude de la figure 1 selon le schéma de la figure 2 : dans la première partie, la référence (voir figure 1) se situe au niveau des objets physiques, dans la deuxième, les réseaux sont étudiés en tant que tels, et dans la troisième, la référence est le système cognitif humain.

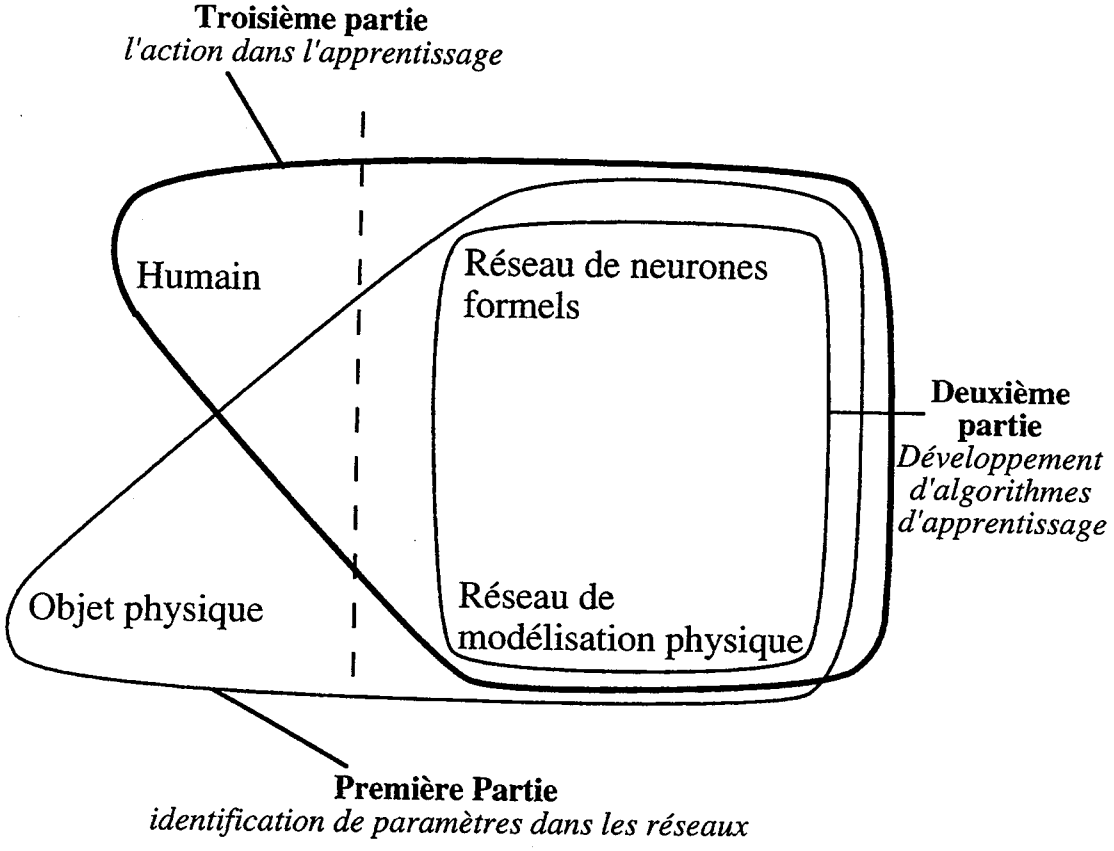

Figure 2. Objets d'étude des trois parties du mémoire

Etant donnée la nature interdisciplinaire de ce mémoire, on comprendra aisément que tous les points n'ont pas pu être traités avec la même profondeur. Le lecteur trouvera ici principalement une réflexion sur les points communs et les différences entre les domaines, ainsi que les idées nouvelles qui ont pu émerger de leur confrontation.

PREMIÈRE PARTIE

MODELISATION PHYSIQUE ET APPRENTISSAGE

Dans cette première partie est proposée une analyse détaillée de notre sujet de recherche. Après l'avoir situé dans le contexte de l'informatique musicale, nous proposons un cadre général pour comprendre l'analyse-synthèse par modèles physiques. Ensuite, nous présentons une étude interdisciplinaire du concept de *réseau*, qui est la base de ce travail. Les enjeux de cette recherche pourront alors être exposés, sous la forme de cinq questions.

I.1 Modéliser la matière

I.1.1 Du phénomène à la cause (du son à l'objet)

Afin d'analyser le sujet qui nous intéresse, il est utile de le replacer dans son contexte historique, celui de la synthèse sonore en informatique musicale. Très loin d'être exhaustif, cet historique a pour objectif de poser quelques jalons qui permettront de cerner ce que l'on peut attendre d'un système de production de sons par ordinateur, et de comprendre ainsi comment a émergé le concept de "modèles physiques" dans ce domaine. Pour une revue détaillée des différentes techniques de synthèse sonore, on pourra se référer à [De Poli 83], à compléter par [Roads 94] pour ce qui concerne les modèles physiques.

Les premières études concernant la production de sons à l'aide de l'ordinateur remontent aux années 1950. La communauté musicale y voyait alors la possibilité de renouveler les instruments de musique et donc la création musicale : alors que l'évolution de la lutherie avait abouti à un nombre finalement restreint d'instruments de musique (l'orchestre symphonique pour la culture occidentale), l'ordinateur pouvait dans son principe servir à produire tous les sons possibles. En effet, un son n'étant rien d'autre qu'une onde vibratoire, on peut, à l'aide de l'ordinateur, synthétiser une séquence temporelle quelconque puis l'envoyer sur un haut-parleur.

Dans cet esprit, une des premières approches a été la synthèse additive [Risset & Mathew 69] : étant donné que tout signal périodique peut se décomposer en une somme de courbes sinusoïdales dont les fréquences sont les multiples successifs d'une fréquence de base, dite fondamentale (décomposition de Fourier), l'idée a été de construire une suite d'oscillateurs à des fréquences f, $2f$, $3f$, etc. et de moduler les différentes amplitudes de chaque oscillateur pour produire une grande variété de sons. Cependant, il s'avère que la synthèse additive non seulement est coûteuse (puisqu'un grand nombre d'oscillateurs est nécessaire) mais surtout nécessite de régler beaucoup de paramètres, chacun de ces paramètres devant avoir sa propre courbe d'évolution temporelle (enveloppe) pour pouvoir modéliser les parties transitoires (les attaques en particulier).

Le cas de la synthèse additive, qui n'est qu'un exemple, nous montre qu'il n'est pas suffisant de posséder un outil capable de réaliser tous les sons possibles. Un outil de synthèse sonore est un modèle générique comportant un certain nombre de *paramètres* (voir figure I.1). La modélisation à l'aide de cet outil est une spécification des paramètres de ce modèle générique, pour en faire un modèle donné. Dans le cas de la synthèse additive, ces paramètres sont les amplitudes en fonction du temps des différents oscillateurs. Il ressort du cas de la synthèse additive qu'il est nécessaire de trouver un modèle générique permettant de *réduire le nombre de paramètres* nécessaires à la synthèse du son.

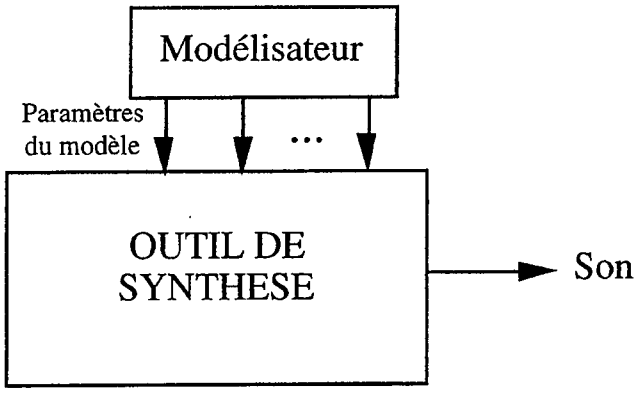

Figure I.1. L'outil de synthèse en tant que modèle générique.

Le cas de la synthèse de sons de cuivres (trompette, trombone, etc.) est à ce titre intéressant. On a pu constater que pour ce type de sons, la richesse spectrale (nombre d'harmoniques) était proportionnelle à l'amplitude : plus on souffle fort, plus le son est strident [Risset & Mathew 69]. En introduisant dans le modèle cette relation de proportionnalité [Pierce 87], il suffit de contrôler l'amplitude pour réaliser des sons de cuivres de très bonne qualité. Toujours dans cette même optique s'est développée l'idée de synthèse par modulation de fréquence : à l'aide de seulement deux ondes sinusoïdales, en utilisant le signal de l'une pour contrôler la fréquence de l'autre, on peut reproduire des sons d'une grande richesse [Chowning 73 *in* De poli 83].

Cependant, la synthèse par modulation de fréquence ainsi que par d'autres techniques apparentées présente deux défauts.

D'une part, les paramètres de contrôle sont difficilement interprétables en termes de caractéristique sonore [Jaffe 95]. La synthèse sonore procède alors par essais-erreurs, et les paramètres obtenus sont difficilement exploitables. Une autre caractéristique importante pour la synthèse sonore par ordinateur est donc la *lisibilité des paramètres*.

D'autre part, les sons obtenus par modulation de fréquence sont très artificiels : il est extrêmement difficile d'obtenir des sons proches de sons réels, en particulier des sons

"chauds". Or Il est important pour tout système de modélisation sonore de pouvoir reproduire les sons réels, quitte ensuite, à des fins musicales, à s'écarter des sons réalistes. L'idée est donc apparue à la fin des années soixante puis dans les années soixante-dix, de s'intéresser aux instruments de musique eux-mêmes, c'est-à-dire à leur propriétés physiques [Hiller & Ruiz 70 *in* Borin *et al. 92*][Cadoz *et al.* 81]. En modélisant l'instrument de musique lui-même, comme on peut modéliser une aile d'avion ou une structure mécanique, on peut espérer reproduire les sons d'un grand nombre d'instruments de musique. Un certain nombre de chercheurs en informatique musicale se sont donc intéressés à ce que l'on appelle la *modélisation physique* d'instruments de musique, en tant que technique de synthèse de sons.

Mais cette notion de modélisation physique est plus qu'une nouvelle technique de synthèse sonore. En effet, fondamentalement, elle consiste en un déplacement de l'objet d'étude : on ne s'intéresse plus au son lui même mais à *l'objet qui le produit*. De la modélisation du *phénomène*, le son, on se penche sur la modélisation de la *cause*, de ce qui a engendré l'émission du son [Cadoz *et al.* 81].

Une première conséquence essentielle de cette modélisation "causale" réside au niveau du rôle de l'instrumentiste. En modélisation physique, le son n'est plus considéré comme un signal (une succession d'informations) mais comme *le produit d'un instrumentiste et d'un instrument*. Il est par conséquent nécessaire d'introduire l'instrumentiste dans le schéma de production du son (figure I.2). Cet instrumentiste peut être soit réel, et se pose alors la problématique de l'interaction instrumentiste-ordinateur, soit lui-même modélisé.

Si l'on souhaite obtenir des sons riches et vivants, on a tout intérêt à choisir la première option, afin de maintenir dans les sons artificiels la richesse de l'interprétation humaine. Comparativement à la synthèse sonore classique, représentée par la figure I.1, la synthèse par modélisation physique revient à ajouter une autre possibilité de contrôle, distinct du contrôle en tant que *modélisateur*, par l'intermédiaire d'informations qui se rapprochent de celles échangées entre un instrumentiste et un instrument de musique réel. Ces "paramètres" supplémentaires sont donc d'une grande pertinence.

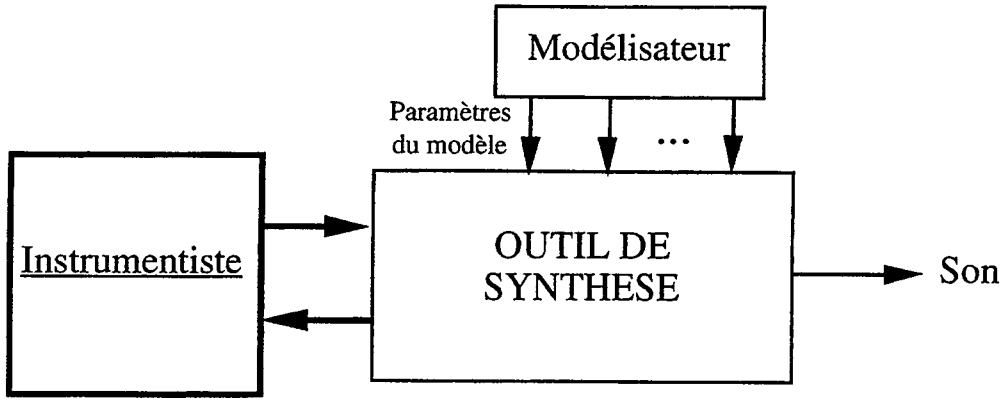

Figure I.2. Intervention de l'instrumentiste dans la synthèse sonore par modélisation physique

En résumé, la modélisation physique est une approche tout à fait différente des autres méthodes de synthèse sonore par ordinateur, et qui présente entre autres les caractéristiques suivantes :

- possibilité de générer des sons très riches à partir de peu de paramètres,
- lisibilité des paramètres,
- sons beaucoup plus vivants, quand ils sont issus d'un geste humain [Cadoz *et al.* 81].

I.1.2 Construire un modèle de l'objet mécanique

Etant donné un mode de représentation, qu'il soit fondé sur la modélisation physique ou non, une préoccupation primordiale est de pouvoir reproduire un instrument de musique quelconque. Ceci permet en effet d'envisager la conception de synthétiseurs de sons capables de reproduire des sons les plus variés, du violon au piano, en passant par l'orgue d'église. Un tel instrument synthétique revient bien moins cher que la somme des coûts de tous les instruments qu'il reproduit. Il est aussi bien moins encombrant. Mais ces motivations pratiques sont mineures pour ce qui concerne la création artistique, qui est à la recherche de nouveaux matériaux sonores. Reproduire la réalité n'est en fait qu'un préliminaire à la création elle-même : une fois obtenu un modèle réaliste d'instrument de musique, le créateur va s'efforcer de transformer le modèle pour évoquer des instruments imaginaires. Par exemple, en synthèse additive, on est capable de réaliser un son qui se situe entre la clarinette et la trompette.

En ce qui concerne les modèles physiques, cette possibilité de reproduire puis de transformer la réalité est très prometteuse. Plus rien ne nous empêcherait en effet de modéliser un piano de dix mètres de haut (transformation géométrique), de pincer une corde de guitare fixée sur des tuyaux d'orgue (assemblage) ou d'entendre le son d'un piano immergé dans de l'air très visqueux (changement de paramètres du milieu). C'est ce

genre de possibilités vers lequel tendent les systèmes de simulation par modèles physiques.

Comment donc réaliser la modélisation fidèle des instruments de musique ? La nature concrète des paramètres du modèle (distances, duretés, viscosités, etc.) permet à l'utilisateur de construire directement un modèle informatique des objets physiques connaissant les propriétés mécaniques des instruments de musique. On peut alors parler de "lutherie artificielle", dans la mesure où il s'agit de "construire" des instruments de musique sur ordinateur.

Cependant, et là réside la justification pratique de notre recherche, une telle lutherie artificielle n'est pas sans poser problème. D'une part, nous sommes finalement loin d'avoir une connaissance satisfaisante des instruments de musique et autres objets mécaniques qui nous entourent. Par exemple, les phénomènes tourbillonnaires d'une clarinette ou les non linéarités d'un frottement d'archet ne sont pas simples à appréhender, donc à modéliser. D'autre part, comme nous le verrons au paragraphe I.3.2, des contraintes informatiques ont amené les chercheurs à se pencher sur des modèles qui peuvent être éloignés du type de connaissance que nous avons des objets. Par exemple, quand une corde est modélisée par un chapelet de masses ponctuelles, même s'il s'agit d'une représentation physique, la correspondance entre les paramètres de ce modèle (nombre de masses, leurs valeurs, valeurs des raideurs et viscosités) et les paramètres habituellement utilisés pour décrire une corde réelle n'est pas si évidente.

Une démarche par essais-erreurs est certes possible, mais ce qui s'avère finalement nécessaire est un *outil d'identification automatique des paramètres des modèles physiques*. Il s'agit, à partir d'un instrument de musique, ou d'enregistrements de celui-ci (par des capteurs divers) de construire automatiquement son modèle informatique.

Cette étude a donc pour application la conception d'un système d'identification de paramètres pour les modèles physiques. Le chapitre I.2 qui suit étudie cette notion d'identification sur le plan *fonctionnel*, pour la relier au concept d'*analyse* traditionnellement utilisé en informatique musicale. Le chapitre I.3 quant à lui propose une analyse *structurelle* et aborde la notion d'*apprentissage*, thème de prédilection du connexionnisme. Dans cette étude, *analyse* et *apprentissage* désignent en fait le même processus, mais vu sous deux angles différents.

I.2 Du réel au virtuel, de l'homme à la machine

L'objectif du présent chapitre est de proposer une analyse *fonctionnelle* générale de la production de sons par modèles physiques, qui inclue les travaux de modélisation déjà effectués ainsi que les outils d'identification vers lesquels nous nous dirigeons. Par analyse fonctionnelle, nous entendons une analyse qui ne prend pas en compte les mécanismes internes des différents composants du système. A l'analyse fonctionnelle s'oppose l'analyse *structurelle*, qui fait l'objet du chapitre I.3.

Cette analyse a pour point de départ la synthèse *sonore*, mais demeure tout aussi valide pour des applications en robotique ou en animation par ordinateur, où les déplacements et les déformations de grande amplitude d'objets en deux ou trois dimensions remplacent les comportements vibratoires. C'est pourquoi, dans ce qui suit, nous utilisons les termes "instrumentiste" et "instrument de musique" quand il s'agit de production sonore, et les termes "opérateur" et "objet" dans le cas général.

I.2.1 L'Homme, la Machine, l'Environnement Physique

Quelles sont les entités mises en jeu lors de la synthèse sonore temps réel par modèles physiques. Un premier niveau d'observation met en lumière deux entités : l'homme (l'instrumentiste) et l'ensemble "ordinateur + interface", comme pour toute autre technique de synthèse sonore. Cette situation est conforme avec le jeu instrumentiste réel : l'interprète est face à son instrument de musique (voir figure I.2).

Mais il est intéressant de mettre aussi en évidence un deuxième niveau d'observation, qui provient du fait que l'approche par modélisation physique ne se ramène pas aux autres techniques de synthèse sonore. Prenons le cas du système CORDIS-ANIMA développé à l'ACROE. Ce système, dans son principe, offre la possibilité de construire un modèle d'un objet donné, puis de voir sur l'écran une image animée de cet objet, de manipuler une interface mécanique bidirectionnelle à retour d'effort qui donne

l'impression de *sentir* cet objet dans ses doigts, et d'entendre un son résultant de cette manipulation. Les signaux envoyés aux différentes interfaces par l'ordinateur sont calculés pour reproduire le plus fidèlement possible l'impression que l'objet est là, présent en face de l'opérateur utilisant le système. Cet objet n'existe en fait que dans l'esprit de l'opérateur, en tant que représentation ; mais dans la mesure où, sans entrer dans le débat philosophique, tout objet matériel n'existe lui aussi qu'en tant que perçu par un être humain, il est intéressant de poser que l'objet *évoqué* par le système "ordinateur + interface" existe lui aussi. Cet objet, on l'appelle alors *objet virtuel*. Superposé au premier niveau d'observation, qui met en évidence un opérateur interagissant avec un instrument de musique électronique, ce deuxième niveau d'observation met en évidence un opérateur interagissant avec un instrument de musique virtuel. Ainsi, le sentiment d'évasion que procure de tels systèmes de *simulation* du monde physique, plus connus sous le terme commercial de "réalité virtuelle", provient précisément du fait que l'utilisateur oublie le monde matériel décrit par premier niveau d'observation pour "plonger" dans le monde virtuel, mis en évidence par le deuxième niveau d'observation.

Une analogie intéressante de cette situation peut se trouver dans l'optique géométrique. Supposons que nous regardions un objet quelconque à travers un miroir à 45 degrés (voir figure I.3). En fait, on peut considérer que ce n'est pas l'objet que nous voyons mais son *image*, qui est de l'autre côté du miroir. Cette image est si présente que l'on est tenté de regarder derrière le miroir pour la saisir. Elle est en fait virtuelle, au même titre que l'objet virtuel de la modélisation physique (voir figure I.4). Le terme consacré en optique est d'ailleurs précisément celui d'image *virtuelle* (cependant, dans cet exemple de l'optique géométrique, l'objet réel correspondant existe en même temps que l'image virtuelle). La notion de virtualité existe donc *explicitement* depuis longtemps.

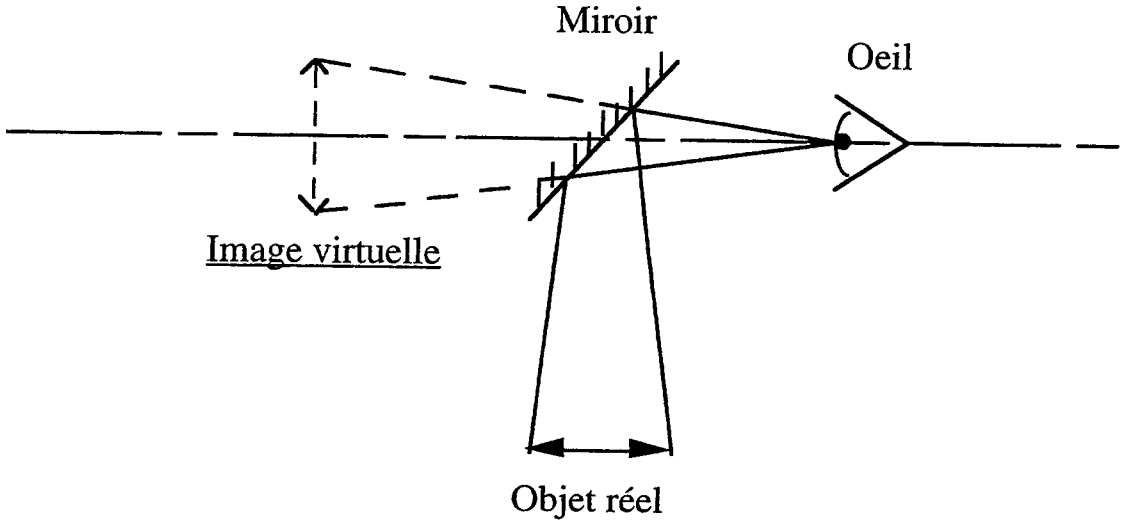

Figure I.3. Virtualité en optique géométrique

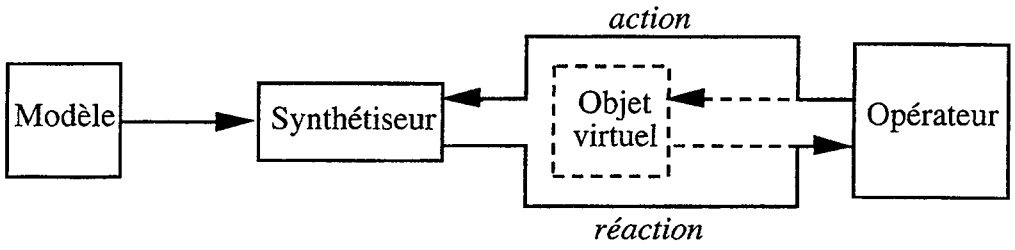

Figure I.4. Virtualité en simulation physique

Si l'on approfondit encore un peu le concept de virtualité, on se rend compte qu'il est intimement lié à celui de *représentation externe*. Par exemple, l'impression de réalisme que peut évoquer un tableau de Monet est une sorte de virtualité, un "paysage virtuel". Ainsi, toute représentation externe, c'est-à-dire toute transcription réalisée par l'homme sur un support quelconque d'un phénomène naturel (un dessin, un enregistrement sonore par exemple) suppose l'existence un objet virtuel, qui a l'aspect d'un objet réel de référence. Ceci permet de distinguer clairement le support de représentation de l'effet qu'il produit : la toile et la gouache, du paysage peint ; la chaîne hi-fi et le signal qui transite jusqu'aux haut-parleurs, du piano que le son émis nous évoque ; le miroir, de l'image représentée ; l'ordinateur et le modèle physique qui y est programmé, de l'objet virtuel manipulé par l'humain.

Le schéma de la figure I.5 résume les différentes entités mises en jeu par le concept de virtualité. L'objet de référence (en gras), qui est *perçu* par l'homme, peut être aussi *modélisé*. Le modèle obtenu est ensuite *simulé* (synthèse) pour permettre l'interaction avec l'homme à nouveau. Cette simulation engendre à la fois une représentation interne chez l'homme et un objet virtuel correspondant. Mais l'homme perçoit aussi la matérialité du système de simulation, c'est-à-dire l'ordinateur et l'interface dans le cas de la modélisation physique. Ce schéma est de portée assez générale ; pour le cas évoqué ci-dessus d'un tableau par exemple, l'objet de référence est le paysage qui a inspiré l'artiste, l'objet virtuel le paysage évoqué, mais dans ce cas, le modèle et le système de simulation se confondent.

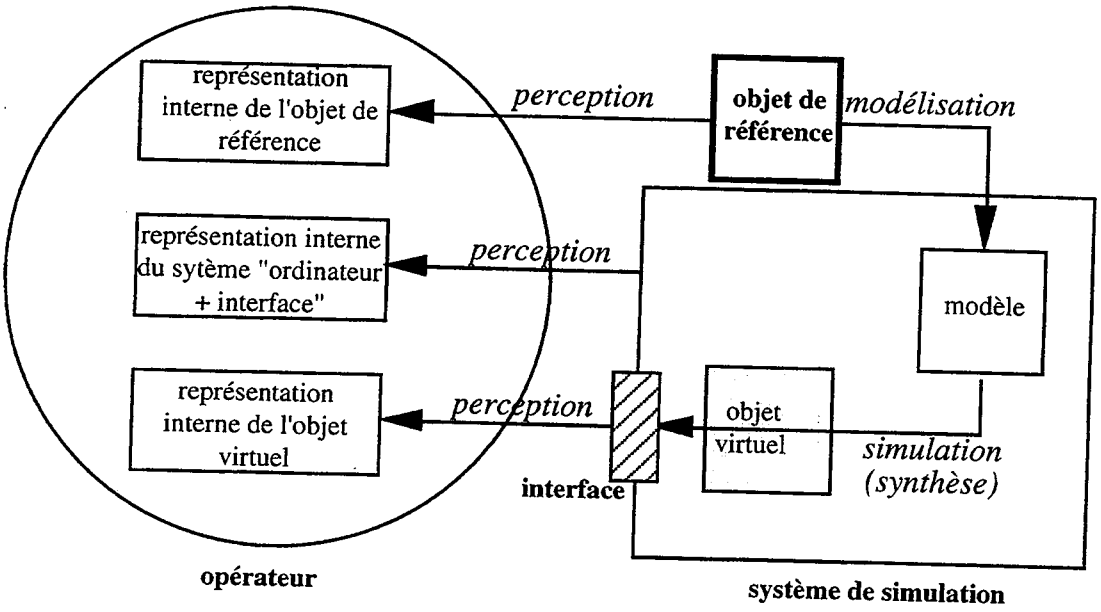

Figure I.5. Différentes représentations impliquées lors de la simulation

Dans la suite, on simplifie ce schéma en faisant finalement abstraction du système de simulation (l'ordinateur et ses interfaces) puisque c'est précisément le but de la simulation d'en faire "oublier" la matérialité. Nous aboutissons finalement à trois entités, chacune sur trois niveaux ou dans trois mondes différents : l'opérateur dans le monde humain, les modèles dans le monde informatique *et* l'objet virtuel dans le monde physique, comme il est représenté dans la figure I.6. On notera que l'opérateur intervient tantôt en tant que manipulateur qui interagit avec l'objet virtuel, tantôt en tant que modélisateur qui construit et observe les modèles informatiques.

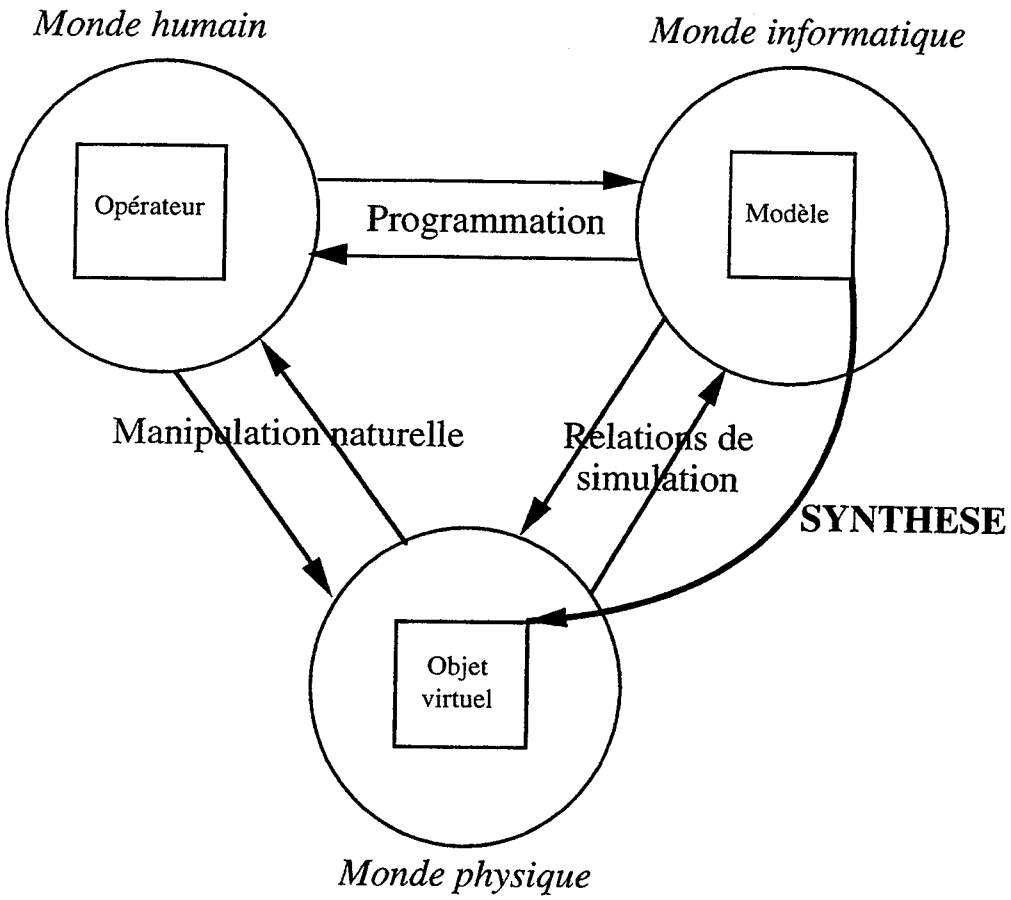

Figure I.6. Ensemble des communications possibles entre les trois mondes.

Dans les figures I.7 à I.11, pour plus de clarté, nous ne feront pas figurer ces communications, mais uniquement les transformations qui en résultent (flèches en gras).

I.2.2 De la synthèse à l'analyse

La production de sons par modèles physiques suppose donc deux entités (en plus de l'instrumentiste) : le *modèle*, qui décrit l'instrument au même titre que le ferait un spectre harmonique en synthèse par série de Fourier, et l'*objet virtuel*, manipulé par l'opérateur. C'est le passage de l'un à l'autre que l'on appelle *synthèse* (voir figure I.6).

Le problème qui nous intéresse, qui est central pour tout système de modélisation-simulation, est celui de la construction d'un modèle en référence à un objet réel. Dans le système CORDIS dans lequel notre projet s'insère, et dans les autres systèmes de modélisation-simulation physique, deux types de construction existent actuellement.

La construction a priori

L'opérateur humain possède une certaine connaissance des objets du monde physique. Cette connaissance peut être acquise par une manipulation présente ou passée

des objets (éventuellement via des instruments de mesure), c'est-à-dire par son *expérience* des objets, ou par une connaissance transmise par d'autres personnes, c'est-à-dire par sa *culture* (voir figure I.7). Grâce à cette connaissance, l'utilisateur peut construire un modèle des objets qu'il connaît. C'est ainsi que l'on modélisera en CORDIS une membrane tendue par un réseau "en toile d'araignée" attaché à sa périphérie, plutôt que par une ligne. Comme nous l'écrivions au paragraphe I.1.2, ce type de "lutherie artificielle" est limité.

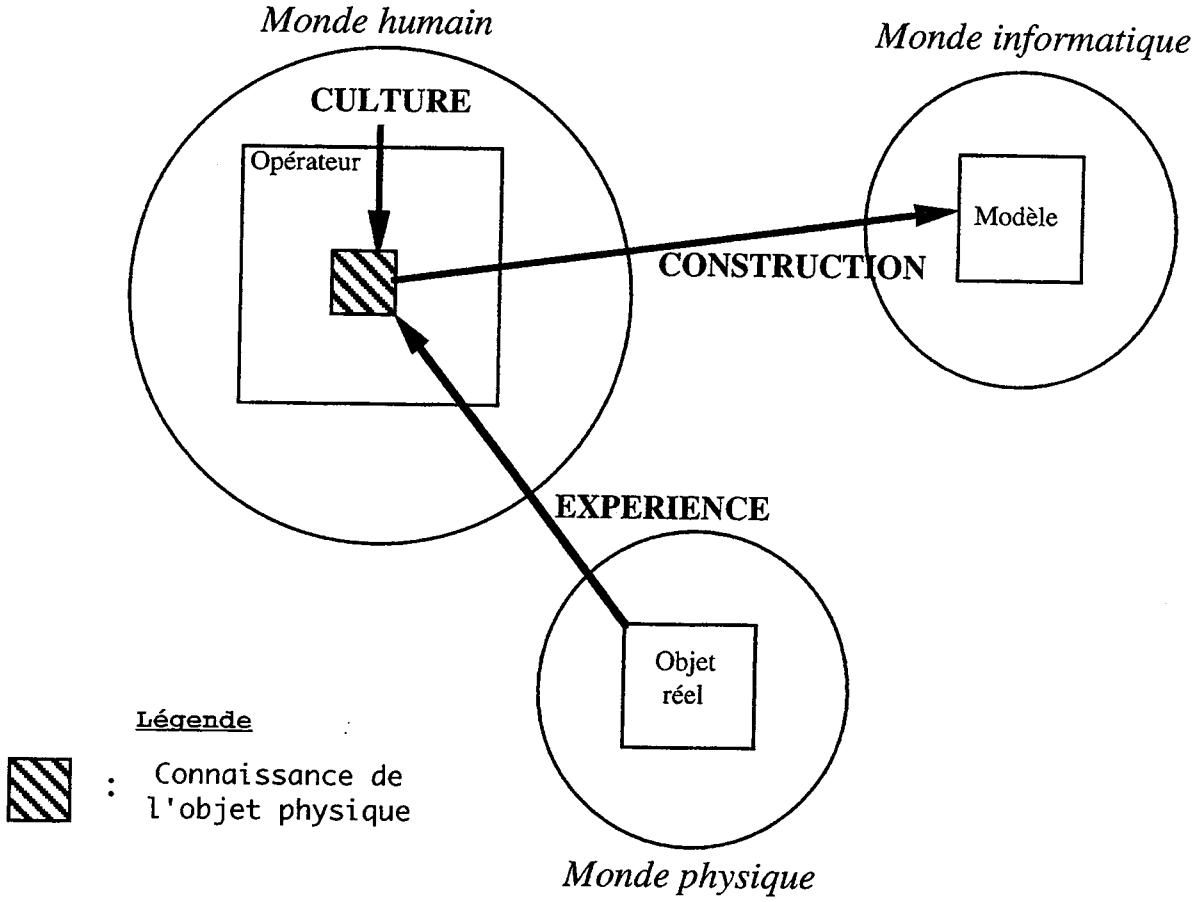

Figure I.7. La construction a priori

La construction a posteriori

Une fois obtenu un modèle par construction a priori, l'opérateur a la possibilité d'expérimenter l'objet virtuel issu de ce modèle. En comparant la perception de cet objet virtuel et celle (actuelle ou passée) de l'objet de référence, l'opérateur peut modifier, ajuster le modèle informatique (voir figure I.8). Ce dernier peut ensuite être synthétisé à nouveau, puis à nouveau comparé, et ainsi de suite jusqu'à obtenir un modèle satisfaisant. Sur le schéma de la figure I.9 est représenté ce cycle, qui peut se répéter de nombreuses fois, selon une stratégie d'essais-erreurs.

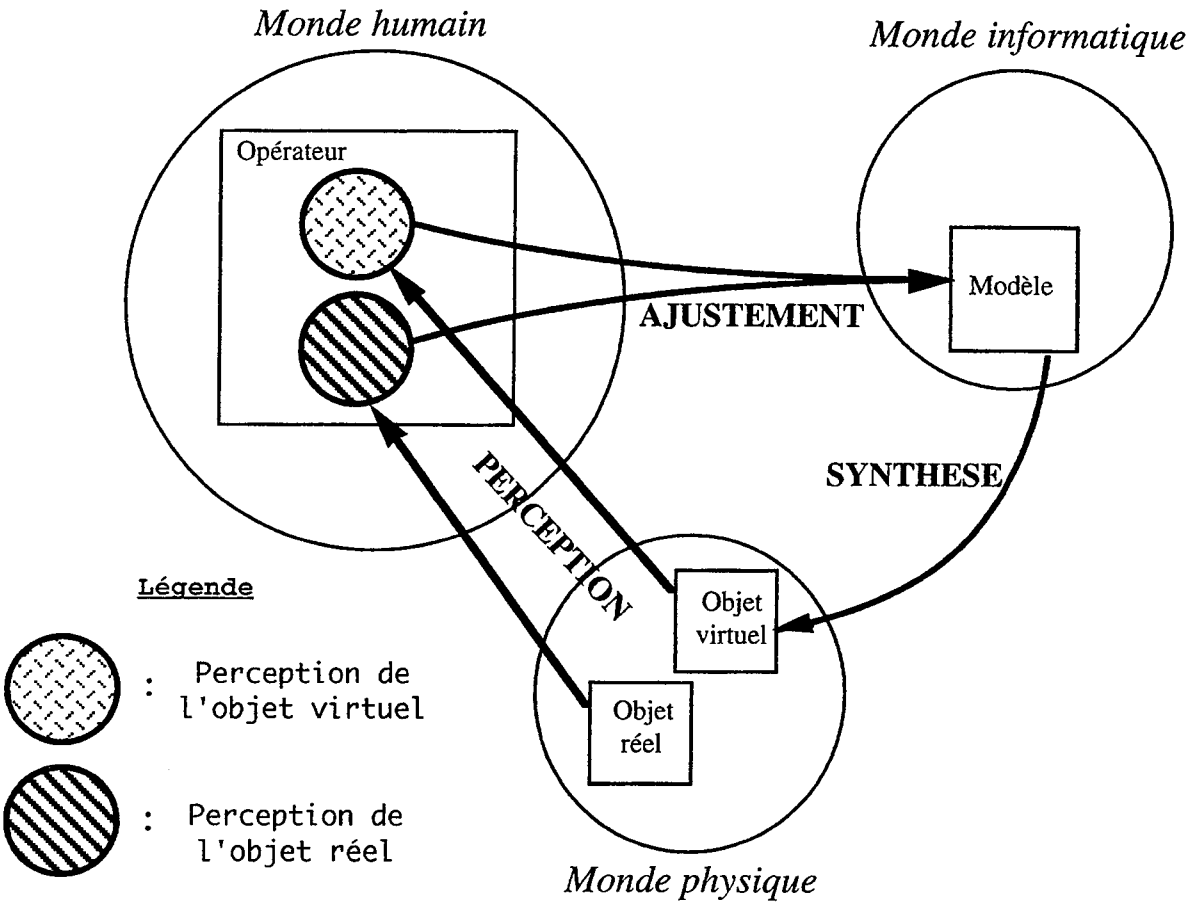

Figure I.8. La construction a posteriori

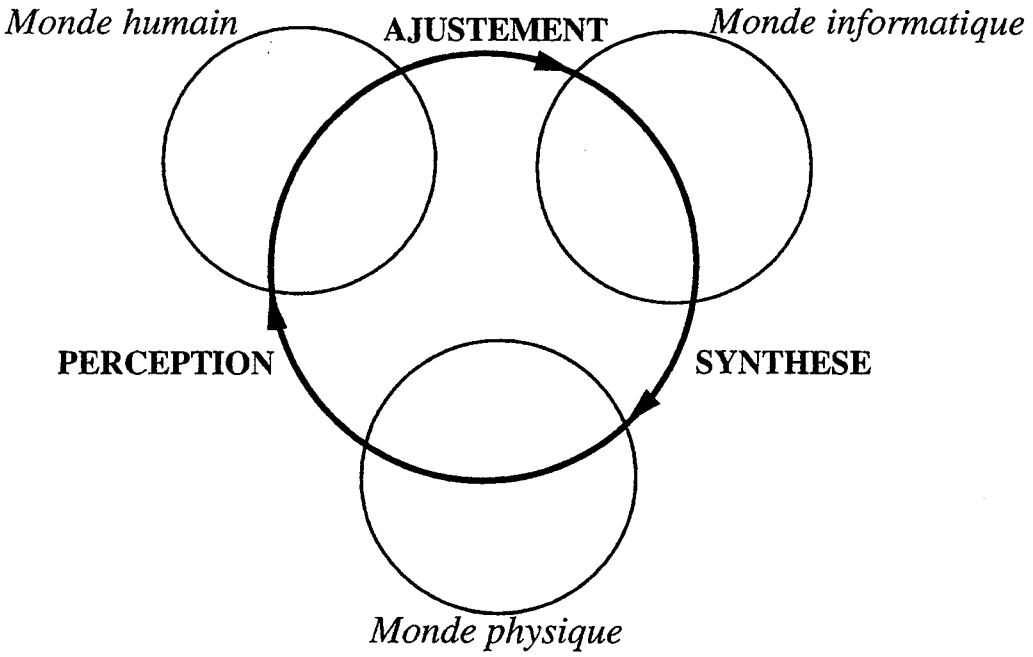

Figure I.9. Cycle de la construction a posteriori

On se rend compte sur les schémas I.7, I.8 et I.9 que la construction du modèle se fait toujours via le système cognitif humain, c'est-à-dire de manière indirecte, et est coûteuse en temps ; une flèche allant directement de l'objet de référence au modèle fait visiblement

défaut. Cette transformation, de l'objet réel à son modèle, symétrique de la synthèse, se dénomme *l'analyse* ; elle est représentée sur la figure I.10.

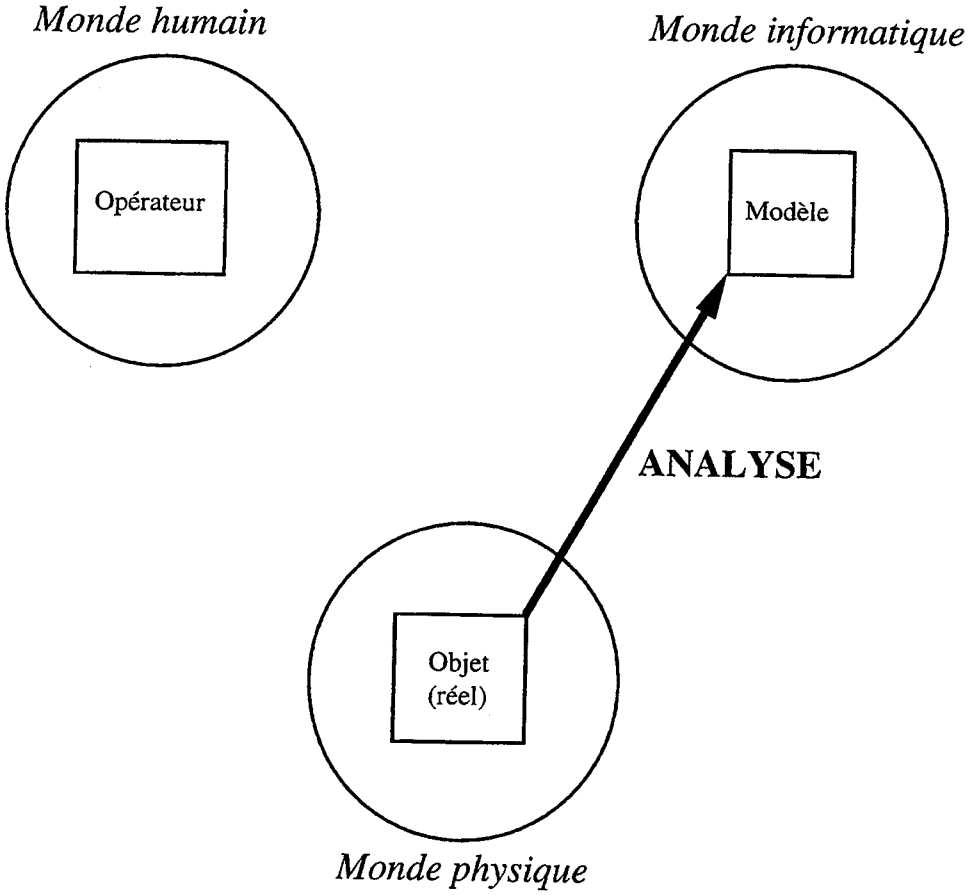

Figure I.10. L'analyse

Cette notion de couple "analyse-synthèse" n'est pas propre à la modélisation par modèles physiques. En informatique musicale, on parle beaucoup par exemple d'analyse-synthèse par série de Fourier : à partir de n'importe quel signal, il est possible de trouver les différents partiels qui le constituent (analyse) puis de reproduire grâce à l'ordinateur tous ces partiels afin de restituer le signal (synthèse). Le fait d'être capable de faire cet aller-retour, du phénomène acoustique au modèle, puis du modèle au phénomène acoustique est en fait une garantie de maîtrise de l'outil de modélisation choisi.

De plus, toujours en référence au cas de la décomposition par série de Fourier, la synthèse et l'analyse semblent pouvoir être symétriques l'un de l'autre *jusque dans les algorithmes de calcul*. Ainsi, notre problématique peut être définie comme suit : **étant donné un algorithme de synthèse, celui développé dans CORDIS, l'objectif est de trouver l'algorithme d'analyse correspondant.**

Il faut remarquer que non seulement la notion de couple analyse-synthèse est très générale, mais aussi que, sur le plan historique, les exemples où la découverte d'une technique de synthèse a précédé celle de la méthode d'analyse correspondante sont

nombreux. Par exemple, alors que dans les années 1830 on était capable, à l'aide de mécanismes rotatifs variés, de *synthétiser* le mouvement, il a fallu attendre les années 1870 pour pouvoir *analyser* le mouvement, c'est-à-dire l'enregistrer automatiquement sur un film. De même, la capacité de représenter de manière réaliste une scène visuelle par la peinture et le dessin précède de plusieurs siècles l'invention du mécanisme d'analyse correspondant, à savoir l'appareil photographique.

Sur le plan fonctionnel, cette symétrie entre l'analyse et la synthèse nous amène à proposer le schéma de la figure I.11, symétrique de celui de la figure I.4. Alors que la synthèse transforme un modèle en un objet virtuel pour l'opérateur, l'analyse agit en tant qu'agent "virtuel" sur l'objet réel pour en déduire le modèle de cet objet. Ainsi apparaît explicitement, par symétrie entre l'analyse et la synthèse le caractère *actif* de l'analyse des objets mécaniques, caractère que nous avons décidé d'approfondir dans ce travail et qui fait l'objet du prochain paragraphe.

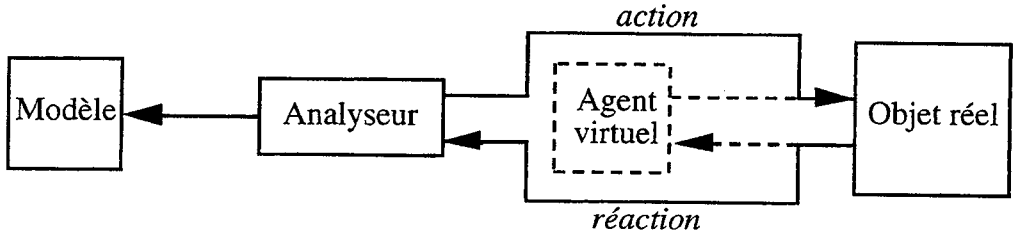

Figure I.11. L'analyse, le symétrique de la synthèse

I.2.3 Analyse interactive

Si l'action apparaît nécessaire à l'analyse des objets physiques, c'est parce que souvent ces derniers ne délivrent pas spontanément d'information. En particulier, les objets que nous étudions sont des objets *passifs* : seuls, ils ne nous donnent aucune information sur leur nature physique, qui se dévoile uniquement si on leur fournit une énergie : le violon n'émet un son que si l'archet est mis en mouvement, un cailloux n'informe de sa dureté que si on le palpe, etc.

Se pose alors immédiatement la question : quelle action fournir à ces objets ? Afin de débroussailler quelque peu cette question, deux points de vue sont à considérer : celui de *l'objectif* de l'analyse, et celui des *moyens* pour atteindre cet objectif.

L'analyse a pour objectif de construire un modèle de l'objet physique. Mais qu'est-ce qu'un objet physique pour l'analyseur qui le manipule ? Rien d'autre finalement que la réponse comportementale à une action de palpation, friction, torsion, etc. Si on élimine toutes les connaissances a priori que l'analyseur peut en avoir, l'objet auquel il fait face se

définit dans le cas général par *l'ensemble des réactions à toutes les actions possibles*. On parle alors d'une connaissance *phénoménologique* [Cadoz 90] de l'objet. Par conséquent, pour être exacte, l'analyse devrait agir de toutes les manières possibles sur l'objet à analyser. On voit là clairement un impossibilité pratique, qui oblige à envisager le problème du *choix* d'une action.

En fait, les objets qui nous intéressent possèdent une particularité qu'il ne faut pas négliger : ils sont destinés à être manipulés par un opérateur qui, lui, est connu. Ainsi, pour reprendre un exemple d'instrument de musique, la guitare qui nous intéresse n'est pas celle dont on a tiré les cordes jusqu'à rupture mais bien celle utilisée par l'instrumentiste à des fins musicales. L'action de l'analyseur devrait donc être celle de l'opérateur lui même. Par conséquent, l'analyseur de la figure I.11, loin d'être entièrement informatique se doit d'inclure l'opérateur humain lui même, et sur le schéma de la figure I.10, le monde humain intervient dans l'analyse.

Nous venons de spécifier l'action de l'analyse d'après l'objet dont elle a pour *objectif* de construire un modèle. Cependant, cette construction n'est pas immédiate, mais au contraire est un long processus qui se déroule dans le temps. La question des *moyens* se pose alors : est-il judicieux, dès le début de l'analyse, de fournir à l'objet l'action correspondant à l'objet à identifier ? Cette action, issue de la manipulation habituelle de l'objet par l'humain, que nous appellerons par la suite *l'action cible*, s'avère relativement riche. Nous proposons donc, afin de faciliter l'analyse, d'introduire la notion d'*action actuelle*, qui est l'action à un instant donné de l'analyse : l'action actuelle peut être différente de l'action cible. Nous postulons en fait que l'interaction au cours de l'analyse est non seulement une obligation, donc une contrainte, mais peut aussi être un atout, si l'action actuelle est choisie de manière à faciliter l'analyse, en termes de réussite, vitesse, qualité du résultat.

A la fin de l'analyse, l'action actuelle est évidemment l'action cible. Au cours de l'analyse, on peut voir l'ensemble des actions actuelles comme une trajectoire dans l'espace des actions qui aboutit à l'action cible, comme l'illustre la figure I.12. La détermination de cette trajectoire est un sujet de recherche à part entière, auquel sera consacrée la troisième partie de ce document.

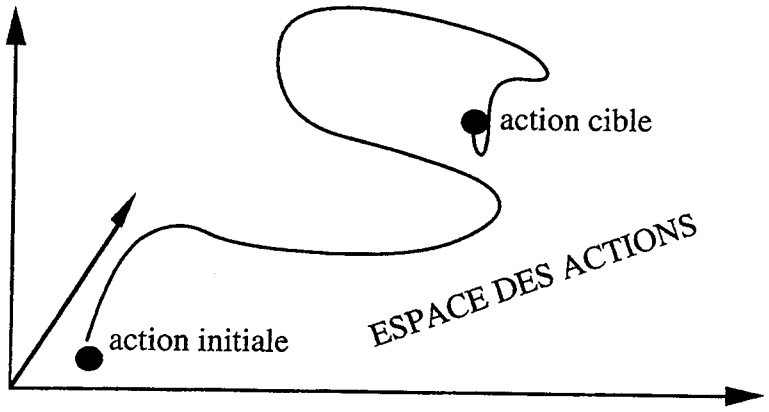

Figure I.12. Trajectoire des actions lors de l'apprentissage

Conjointement à la détermination du contenu de l'action de l'analyse, se pose la question de l'agent responsable de cette action. Alors que dans la synthèse, seules deux entités concrètes interviennent, l'opérateur et l'ordinateur, dans l'analyse, on trouve l'opérateur, l'ordinateur *et* l'objet à analyser. Trois interactions sont donc à envisager : entre l'opérateur et l'ordinateur (et ses interfaces), entre l'opérateur et l'objet à modéliser, entre cet objet et l'ordinateur (et ses interfaces).

D'une part, nous tâcherons de définir les statuts de ces trois interactions. En particulier, l'objet est impliqué dans deux interactions ; il faudra alors déterminer où se situera l'action actuelle évoquée ci-dessus. Notons que même si l'opérateur n'intervient pas directement sur l'objet, il peut le faire via l'ordinateur.

D'autre part, la *nature* de ces trois interactions est à définir. Certaines seront évidemment de nature mécanique, quand il s'agira de manipuler des objets, mais des interactions de type visuel ou textuel sont à envisager entre l'ordinateur et l'opérateur, afin que ce dernier puisse modifier son action en fonction de données concernant le déroulement de l'analyse.

*

Nous avons pu replacer la problématique de l'identification de paramètres définie à la fin du chapitre I.1 dans le contexte de l'interaction homme-machine et en particulier de la synthèse sonore par modèles physiques. Considérer cette problématique en terme d'analyse (c'est-à-dire d'inverse de la synthèse) nous engage ainsi à étudier tout particulièrement *l'action* dans l'identification de paramètres.

Dans le chapitre qui suit, nous allons entrer à l'intérieur des modèles, et étudier une forme de modélisation très générale : les réseaux.

I.3 Des réseaux

I.3.1 Introduction

Le concept de réseau est manipulé dans de nombreux domaines de l'informatique : réseaux d'ordinateurs, réseaux de processeurs, réseaux de communication, réseaux sémantiques, réseaux neuronaux, etc.

Il s'agit donc d'un concept vaste et très utilisé, mais qui néanmoins réfère systématiquement à une même structure, ayant deux composants : des unités et des connexions. Un réseau se définit alors comme un ensemble d'unités, généralement en grande quantité, reliées par des connexions. De manière générale, c'est la théorie des graphes [Kaufmann 70] qui décrit de telles structures, en termes de nœuds et d'arcs.

Pour notre propos, nous nous intéressons aux réseaux en tant que *modèle du traitement de l'information*. Les nœuds ne sont plus des entités quelconques mais des unités prenant en entrée certaines informations et délivrant en sortie une certaine transformation de ces informations, les connexions véhiculant ces informations. On parle alors souvent de *réseaux d'automates*.

L'étude de tels réseaux d'automates peut prendre des formes variées. Tout d'abord, on peut les étudier *en tant que tels*, ou bien par rapport à une *référence* naturelle. Dans ce dernier cas, il s'agit de *modélisation*, et l'on s'intéressera alors à comparer le modèle (le réseau) à la référence.

Ensuite, de nombreuses méthodes sont utilisables :

- l'analyse mathématique, qui permet de comprendre, construire un réseau ainsi que prédire son comportement ;
- la simulation par ordinateur, qui donne une certaine idée du comportement du réseau ;
- la réalisation physique, c'est-à-dire l'implantation sur un substrat physique quelconque d'unités et de connexions.

Ces trois types de méthodes se complètent et peuvent être utilisées conjointement.

Dans ce chapitre, nous aborderons en particulier deux larges classes de réseaux d'automates : les réseaux "physiques" et les réseaux "cognitifs", puis nous tenterons de rapprocher ces deux classes, et enfin nous aborderons le problème spécifique de l'apprentissage.

I.3.2 Les réseaux de modélisation-simulation physique

La modélisation du monde physique qui nous entoure est la préoccupation des sciences physiques et n'est donc pas nouvelle. L'avènement de l'ordinateur au milieu du siècle a permis de décupler les possibilités de simulation des modèles.

Deux premières approches de simulation physique par ordinateur sont directement héritées de l'ensemble des connaissances acquises en sciences physiques.

La première, l'approche microscopique, consiste à modéliser les molécules et leurs interactions. Malheureusement, cette approche devient rapidement inadaptée pour simuler des phénomènes macroscopiques, car les temps de calcul sont gigantesques ("nécessiteraient très vite un temps supérieur à l'âge de l'univers..." [Boon *et al.* 93]).

La deuxième approche, l'approche macroscopique, suppose que l'on dispose d'équations continues de la matière, que l'on peut *analytiquement* résoudre. La simulation est alors très aisée. Malheureusement, les cas où une solution existe sont exceptionnels.

Face à ces deux premières approches, la démarche classiquement adoptée consiste à établir les équations mathématiques de la matière, pour les résoudre *numériquement*. Bien qu'extrêmement puissante et générale, cette dernière approche présente les défauts suivants :

- Il faut pouvoir déterminer l'équation de l'objet.
- Il y a deux représentations successives d'un même objet physique : l'équation mathématique et l'algorithme de simulation. Pour constituer ces deux représentations, deux théories distinctes sont respectivement nécessaires : la physique et l'analyse numérique (voir figure I.13). Ainsi, des difficultés d'interprétation se présentent : les phénomènes obtenus par simulation proviennent-ils de l'équation elle-même ou de sa discrétisation ?
- Les calculs sont en général très lourds.

Pour s'affranchir de ces défauts, on a été amené :

- d'une part à chercher une théorie qui permette de passer *directement* de l'objet physique à sa description algorithmique, c'est-à-dire une "physique algorithmique" [Luciani *et al.* 91][Jimenez 93] ; certains travaux concernant la physique discrète [Greenspan 73] vont dans ce sens ;
- d'autre part à passer d'une description macroscopique de l'objet physique à une description modulaire informatique, c'est-à-dire à décomposer un objet en petits éléments élémentaires de matière en interaction, chaque élément de matière étant directement et aisément modélisable par un algorithme.

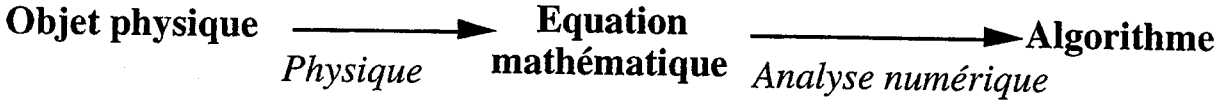

Figure I.13. Simulation physique traditionnelle

Ainsi, différents courants de recherche ont indépendamment abouti à des descriptions sous forme de *réseaux*. Par exemple, on a développé dans les années 1980 l'idée d'appliquer les automates cellulaires [Wolfram 86][Greussay 88] à la modélisation de fluides [Boon *et al.* 93][Amy & Decamp 87]. Un réseau d'automates cellulaires consiste, pour le cas à deux dimensions, en un ensemble d'automates régulièrement répartis sur un plan, une matrice. Chaque automate est connecté à ses voisins et évolue (modifie ses variables internes) de manière *discrète* selon une règle. Ce type d'automate peut être utilisé pour simuler des fluides, par exemple en codant par "1" la présence d'une particule et par "0" son absence [Boon *et al.* 93]. Ce type de modélisation, qui constitue pourtant une forte simplification de la réalité, donne de bons résultats, avec des temps de calculs très courts. De plus, les automates cellulaires sont tout à fait dédiés à une implantation parallèle.

En ce qui concerne la modélisation physique pour la synthèse sonore (voir chapitre I.1.1), on constate de même une "dérive" vers les réseaux, qu'il s'agisse de la technique des guides d'onde [Van Duyne & Smith III 92] ou des réseaux masses-ressorts-frottements [Cadoz *et al.* 81].

Ces derniers, qui sont à la base des modèles CORDIS que nous utiliserons par la suite, sont à cet égard tout à fait intéressants : chaque automate (voir paragraphe II.1.1 pour une présentation détaillée) est conçu pour représenter un élément de matière de complexité minimale. Ainsi, chacun d'eux, même s'il manipule des valeurs décimales (représentant des forces et des positions), n'effectue que quelques additions et multiplications. C'est précisément ce qui permet de réaliser des simulations *temps réel* d'objets mécaniques vibrant à 1 kHz, grâce à des machines spécifiques [Cadoz *et al.* 93].

Mais la notion de réseau en synthèse sonore est aussi intéressante car elle permet de manipuler des modules que l'on peut assembler à loisir. Une modélisation par réseaux

permet, *en gardant toujours le même formalisme,* aussi bien de constituer une microstructure (par exemple une "tranche de corde" de violon) en assemblant quelques automates élémentaires, que d'assembler ces microstructures pour réaliser des macrostructures (une corde) ou d'assembler plusieurs macrostructures pour réaliser un objet plus complexe (un violon).

Il est à noter que pour tous ces réseaux, les contraintes informatiques, qui ont été placées au premier plan, aboutissent à des modèles d'automates non seulement "hybrides", car ils modélisent des phénomènes ni microscopiques ni macroscopiques [Jimenez 93 p. 75], mais aussi simplifiés, éloignés des interactions complexes qui existent dans la matière.

Pour conclure, voici un rappel des traits caractéristiques des réseaux de modélisation-simulation physique :

- le passage de l'objet physique à l'algorithme de simulation est direct,
- les algorithmes sont rapides,
- ils peuvent être implantés sur des machines parallèles ou réalisés en VLSI,
- ils ont la propriété d'assemblage modulaire,
- ils modélisent des comportements ni microscopiques ni macroscopiques,
- les traitements sont très simplifiés par rapport à la réalité physique.

I.3.3 Les réseaux de modélisation-simulation cognitive

Pour caractériser ce qu'on appelle abusivement les "réseaux de neurones", il est nécessaire de comprendre l'historique interdisciplinaire de ce champ très actif de recherche.

I.3.3.a Une "interdiscipline"

Plusieurs disciplines participent à l'étude des réseaux neuronaux.

La neurobiologie observe l'organisation neuronale du cerveau et est amenée à modéliser et simuler partiellement cette organisation, afin d'en comprendre le fonctionnement. Elle est ainsi à l'origine des simulations de réseaux de neurones. Le but de ces simulations est de reproduire les résultats expérimentaux obtenus sur des sujets humains ou des animaux.

La psychologie cognitive s'intéresse aussi aux réseaux pour reproduire des résultats expérimentaux chez l'homme et l'animal, mais il s'agit d'une modélisation *fonctionnelle*, car chaque nœud ou arc du réseau ne tente pas de reproduire précisément le comportement d'une entité biologique tel qu'un neurone ou une synapse mais représente un niveau supérieur, parfois qualifié de subsymbolique [Smolensky 88]. Par contre, l'ensemble du comportement doit mimer un comportement psychologique. Alors qu'en neurobiologie, la notion de réseau provient directement de ce que l'on observe du cerveau (du moins à un certain niveau), la psychologie n'est nullement contrainte d'utiliser une modélisation connexionniste. Il s'agit en fait d'un *choix*, qui s'inscrit historiquement dans le courant "émergence" des sciences cognitives [Varela 89] succédant au courant "cognitiviste" fondé sur la notion de computation symbolique.

Du point de vue de l'informatique, les réseaux sont surtout apparus comme une alternative aux modèles computo-symboliques, en ce qu'ils abordaient en premier lieu la notion d'*apprentissage*. Un système expert, archétype de l'utilisation de l'Intelligence Artificielle symbolique, a besoin d'être informé explicitement des règles à appliquer selon la situation, ce qui n'est pas sans poser de problèmes d'extraction et explicitation des connaissances.

La physique statistique s'intéresse de près aux réseaux en faisant une *analogie* entre un réseau de neurones et certains systèmes de particules : les verres de spin [Weisbuch 89]. La physique statistique s'avère en effet apporter des outils précieux concernant la compréhension de la *complexité* dans les réseaux.

Enfin, il ne faut pas oublier un éventail de branches des mathématiques et des sciences de l'ingénieur qui sont autant d'outils et de formalismes pour comprendre et construire des algorithmes neuronaux : traitement du signal [Hérault & Jutten 94], théorie de l'approximation [Labbi 93], théorie des système dynamiques [Demongeot 92], techniques d'identification de processus [Widrow & Stearns 85][Nerrand *et al.* 93].

Dresser l'historique complet des interactions entre ces disciplines dépasse le cadre de cette étude, mais nous souhaiterions discuter dans ce qui suit du statut du champ de recherche qui nous concerne le plus : les réseaux en tant que discipline de l'informatique.

I.3.3.b Définir les "réseaux de neurones artificiels"

Il faut reconnaître que les réseaux de neurones artificiels s'avèrent finalement une discipline assez décriée, contestée de toute part. En bref, d'un côté on reproche aux modèles neuronaux informatiques d'être trop simplistes pour modéliser quoi que ce soit

de neurobiologique, de l'autre, on clame que les réseaux ne font que reformuler, simplifier ou utiliser sans fondement mathématique des techniques déjà connues, en filtrage adaptatif, théorie de l'approximation, méthodes statistiques de traitement des données, etc. Enfin, on constate que la possibilité de réalisations matérielles, très souvent avancée en faveur des "réseaux de neurones artificiels" demeure encore incertaine, étant donnés les problèmes techniques qui se posent. Nous allons tenter de répondre à ces interrogations.

C'est bien de la neurobiologie dont les modèles neuro-informatiques sont issus. Historiquement, le modèle de McCulloch et Pitts qui date de 1943 est un certain reflet des connaissances de l'époque concernant le cerveau. Mais il est une tendance générale en Intelligence Artificielle à emprunter des modèles aux sciences humaines pour très vite les constituer en champ autonome de recherche et d'application ; par exemple, les recherches du début des années soixante-dix concernant les systèmes à base de règles sont dominées par des motivations psychologiques alors que quelques années plus tard, les systèmes à base de règles font principalement l'objet d'études purement informatiques. Plus récemment, le "raisonnement à base de cas" suit la même évolution. Les réseaux neuronaux n'échappent pas à la règle et se sont ainsi constitués en un champ autonome de l'informatique. Plus exactement, chaque modèle prend sa source dans la neurobiologie puis constitue un objet d'étude informatique à part entière : le modèle de McCulloch & Pitts [Amy & Decamp 87][Davalo & Naïm 89][Hecht-Nielsen 90], les cartes de Kohonen [Kohonen 82], les réseaux à base de prototypes [Giacometti 92]. Depuis 1943, la connaissance du cerveau a fait des progrès fulgurants et le modèle de McCulloch & Pitts, très largement utilisé par les informaticiens, est totalement erroné : les phénomènes temporels de codage, en particulier au niveau dendritique, jouent un rôle important ; le codage n'est certainement pas binaire ou fréquentiel mais certainement plus sophistiqué (voir par exemple les cycles oscillatoires) ; la notion même de *réseau* est peut-être à revoir, vue la complexité des liaisons synaptiques (synapse sur une autre synapse, etc.) ; les interactions chimiques jouent un rôle important, etc. Il faut donc oublier l'inspiration biologique en ce qui concerne la plupart des modèles informatiques étudiés, même si un certain nombre de chercheurs se consacrent judicieusement à l'étude de modèles informatiques plus plausibles du point de vue biologique. Ainsi, par souci de prudence et d'exactitude, il ne faut pas considérer l'*unité* et la *connexion* d'un réseau neuro-informatique comme modèles respectifs d'un neurone ou d'un axone. Nous considérons qu'une connexion d'une unité à une autre dans un réseau neuro-informatique représente plutôt l'influence d'une *zone* du système à une autre, en nous gardant bien de définir combien de neurones englobe cette zone, à quoi correspondent les variables utilisées par le réseau, etc. Il s'agit d'une modélisation ni structurelle ni fonctionnelle, mais entre les deux, qui nous rapproche des modélisations psychomimétiques évoquées ci-dessus [Smolensky 88] (voir figure I.14).

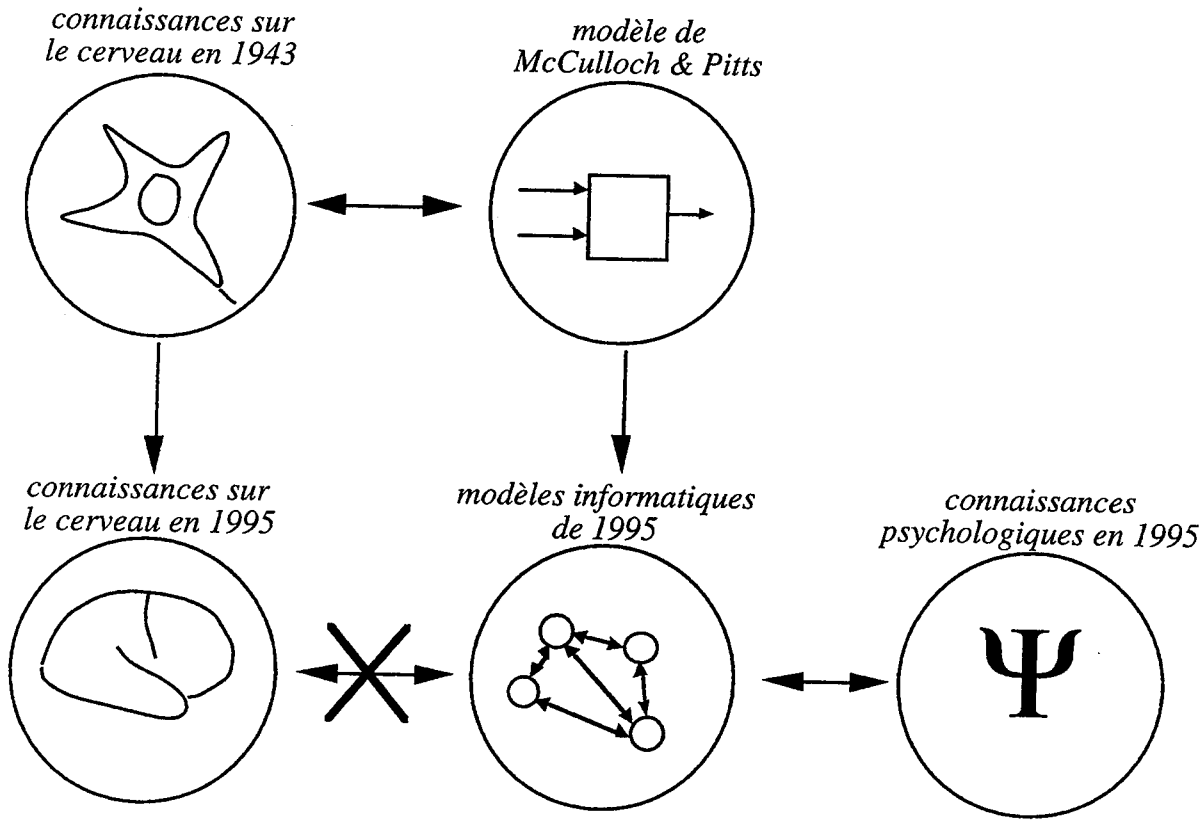

Figure I.14. liens entre les modèles connexionnistes informatiques et les sciences humaines

Par ailleurs, si les modèles neuro-informatiques se détachent des neurosciences, c'est généralement pour étudier théoriquement leurs propriétés. Ainsi, un certain nombre de modèles se sont vus resitués dans des cadres plus théoriques : régression non linéaire, systèmes dynamiques, etc. Pourquoi alors encore parler de réseaux, si ceux-ci ne sont que des cas particuliers de méthodes plus générales ?

D'une part, il existe beaucoup de modèles souvent utilisés, certains très simples, qui n'entrent pas aisément dans une théorie mathématique : les cartes auto-organisatrices de Kohonen [Kohonen 82], les réseaux récurrents simples de Elman [Elman 90], la "cascade correlation" [Fahlman & Lebiere 91], etc. En fait, on peut penser que l'ensemble des modèles connexionnistes que nous découvrons et implantons est et sera toujours bien plus vaste que l'ensemble des modèles mathématiquement démontrés. Nous estimons qu'il est souhaitable de développer des formalismes mathématiques sans pour autant cesser la recherche plus intuitive de nouvelles architectures et algorithmes.

D'autre part, même si le réseau s'écrit sous forme de *formules* et s'inscrit dans une théorie mathématique, *la formule n'est pas le réseau* (voir figure I.15). Ce dernier présente les caractéristiques suivantes :

- il se représente sous une forme graphique unifiée généralement simple ;
- cette représentation graphique explicite les traitements qui sont effectués ;

- cette représentation est à la base de réalisations matérielles de réseaux qui peuvent donner aux réseaux des rapidités de traitement exceptionnelles.

Forme mathématique

$$S = A.e$$

avec : $A = (a_{ij}) \in \mathbb{R}^{n \times m}$

$e \in \mathbb{R}^m$

$s \in \mathbb{R}^n$

Forme graphique

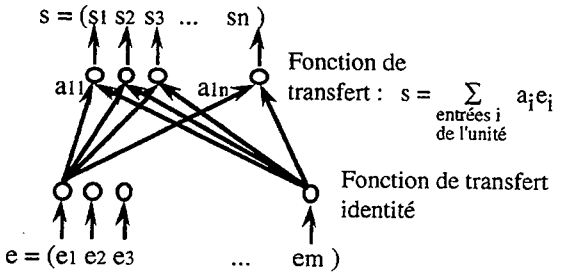

Figure I.15. Deux formes d'un réseau

Bien sûr, la plupart des réseaux actuels sont *simulés* sur des ordinateurs séquentiels classiques [Orsier 95], car la réalisation matérielle de réseaux présente des difficultés ; travailler sur les réseaux constitue un *pari* sur l'avenir. Notons cependant que dès à présent, les réseaux se prêtent tout à fait bien à des implantations sur des machines parallèles, du fait de la simplicité des calculs de base.

Pour finalement cerner ce champ de recherche de l'informatique dont l'appellation "Réseaux Neuronaux" est nous l'avons vu inadaptée, nous proposons :

Une définition

il s'agit de l'étude théorique et expérimentale des comportements d'assemblées d'unités simples interconnectées entre elles sous forme d'un réseau.

Trois objectifs

- la modélisation cognitive à un niveau se situant entre le niveau structurel et le niveau fonctionnel,
- la reformulation et le développement d'algorithmes d'Intelligence Artificielle,
- la réalisation matérielle future de systèmes de reconnaissance de forme, identification de processus, contrôle, classification, etc. extrêmement rapides.

Deux points forts

- des mécanismes d'adaptation des paramètres qui peuvent être vus comme une forme d'apprentissage ;
- une explicitation visuelle des traitements effectués sous une forme simple et générale qui favorise la communication et le développement des idées sur le sujet.

Nous n'avons pas donné ici une définition dans laquelle tous les chercheurs en réseaux de neurones se reconnaîtront, loin de là. Mais il semble qu'elle fédère un certain nombre de recherches sur ce thème.

I.3.3.c Quelques caractéristiques concernant les réseaux connexionnistes

Il est particulièrement difficile de présenter un formalisme général pour les réseaux, tant la diversité des modèles est grande. Un grand nombre de modèles cependant utilise pour nœud une unité de traitement qui applique une fonction de transfert donnée à la somme des entrées, et pour arc une connexion pondérée qui transmet un signal des sorties aux entrées en le multipliant par un poids (figure I.16).

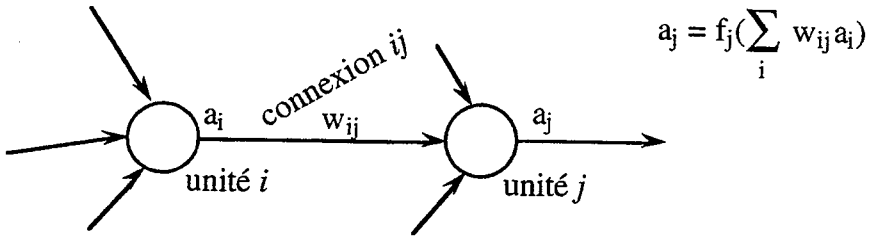

Figure I.16. Forme classique d'un réseau connexionniste ; formulation discrète non temporelle

En plus de cette loi de propagation, il y a souvent une loi d'adaptation ou d'apprentissage, qui modifie les poids. Pour ce faire, d'autres informations peuvent d'ailleurs circuler dans le réseau.

Chaque unité possède différents paramètres, que ce soit pour la propagation ou pour l'apprentissage.

Etant données ces caractéristiques très générales des réseaux connexionnistes, nous proposons de classer les différents modèles selon leur utilisation, leur structure et leur principe d'apprentissage. D'autres critères existent bien sûr pour différencier les modèles (type des unités, nature des signaux propagés, etc.) mais nous présentons ici les traits qui nous paraissent les plus importants.

Utilisation

Nous entendons ici par "utilisation" la manière dont sont utilisées les différentes entrées et sorties du réseau, à la fois en mode *propagation* (dynamique des activités) et en mode *apprentissage* (dynamique des poids).

On distingue généralement les modes de fonctionnement supervisé et non supervisé. Dans le mode supervisé, le réseau reçoit lors de l'apprentissage deux patterns différents : un stimulus et une réponse désirée. Après de multiples présentations de paires "stimulus-réponse désirée", le réseau doit être capable, quand il reçoit uniquement le stimulus, de fournir en sortie une réponse identique à la réponse désirée (ou proche). On trouve très

souvent des définitions de la notion de supervision en terme de "professeur" fournissant le comportement désiré. Or parler de professeur, même avec les guillemets, est déplacé, et ce à deux titres. D'une part un professeur est une entité bien plus complexe qu'une "collection de bonnes réponses" (c'est un système cognitif lui-même, il interagit avec l'élève, etc.) et d'autre part, cette définition restreint l'apprentissage supervisé au cas où la réponse désirée est fournie par un système cognitif accompagnant le réseau, alors que la réponse désirée peut être fournie soit par l'environnement lui-même soit par une autre partie du système dans lequel le réseau s'insère.

Dans une utilisation que l'on qualifie parfois de semi-supervisée, le réseau ne reçoit pas la réponse désirée mais une information indiquant si la réponse est correcte ou non ; il s'agit alors d'apprentissage par renforcement.

En mode non supervisé, le réseau ne reçoit à chaque instant qu'un pattern (il n'y a pas de réponse désirée). On distingue alors les *mémoires associatives*, qui lors de l'apprentissage enregistrent des patterns pour ensuite les restituer lors de la présentation de nouveaux stimuli, et les *analyseurs de données* (comme les réseaux de Kohonen utilisés pour représenter des données multidimensionnelles sur une carte 2D), qui codent un grand nombre de patterns sous une forme mieux exploitable.

Nous n'avons pas dans ce qui précède associé aux notions de "supervisé" et "non supervisé" des types de réseaux, car en fait un même modèle peut souvent être utilisé de plusieurs manières, ce qui entraîne une confusion. Par exemple, les réseaux à rétro-propagation du gradient [Le Cun 87], généralement classés dans les "supervisés" peuvent très bien être utilisés en non supervisés (réseau "diabolo" où l'on met les entrées en sorties désirées), et vice-versa, on peut utiliser un réseau de Hopfield [Hopfield 82] en supervisé, en séparant les unités en deux groupes.

Structure

Selon la manière dont sont agencées les unités et connexions d'un réseau, il existe une multitude de structures possibles. Il faut distinguer les structures élémentaires et les structures composites, que l'on obtient par assemblage de structures élémentaires.

On distingue les structures *récurrentes* et les structures *unidirectionnelles* : une structure récurrente possède au moins un *circuit* dans son graphe d'interconnexion. En toute rigueur, tout réseau récurrent a nécessairement un comportement temporel qui se prolonge à l'infini, du fait du circuit.

La structure (récurrente) la plus simple est bien sûr la structure entièrement connectée (voir figure I.17), qui peut être utilisée en mémoire associative [Hopfield 82] ou en apprentissage de séquences [Pearlmutter 89][Williams & Zipser 89].

Une autre structure courante est la structure à couches (voir figure I.17), qu'il s'agisse de réseaux utilisant des unités à fonction de transfert sigmoïdale [Le Cun 87] ou de réseaux à base de prototypes [Giacometti 92].

Certaines structures, unidirectionnelles ou récurrentes, peuvent aussi être qualifiées de *modulaires*, dans la mesure où les unités se divisent en sous-ensembles : la connectivité est élevée *à l'intérieur* des sous-ensembles, mais faible *entre* ces sous-ensembles.

Enfin, notons que bon nombre de modèles combinent plusieurs structures ou plusieurs réseaux : les réseaux récurrents à couches [Elman 90][Jodouin 93], le réseau à contre-propagation [Hecht-Nielsen 88], etc.

Ces structures, ainsi que quelques autres sont résumées dans la figure I.17.

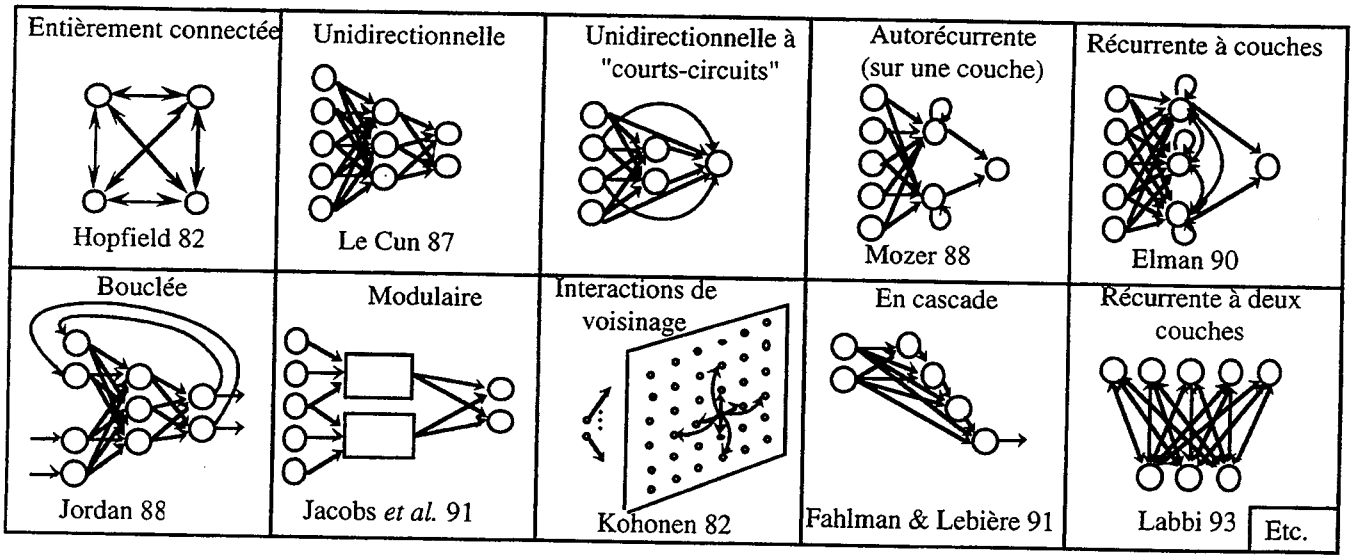

Figure I.17. Exemples de structures de réseaux

Ces structures, que nous représentons graphiquement de manière statique, peuvent cependant être dynamiques, quand unités et connexions se créent pendant l'apprentissage ; on parle alors de réseaux incrémentaux, ou constructifs, ou "ontogéniques" (voir [Fiesler 94] pour un rapide état de l'art).

Principe d'apprentissage

Les réseaux supervisés sont souvent entraînés selon le principe de descente de gradient, issu des techniques d'optimisation et d'identification. Cela consiste à minimiser

une erreur entre le comportement du réseau et le comportement souhaité, en utilisant l'information du gradient qui indique dans quel sens déplacer les paramètres pour diminuer l'erreur.

La loi de Hebb est la première règle qui a été proposée pour modifier les poids d'un réseau. D'inspiration biologique, elle consiste à renforcer le poids de la connexion entre deux unités simultanément actives. D'autres lois sont dérivées de la loi de Hebb, comme par exemple la loi "anti-hebbienne", qui consiste à atténuer la connexion d'unités actives en même temps.

Il existe encore bien d'autres règles d'apprentissage (auto-organisation, loi d'"assimilation-accomodation" de Grossberg, etc.).

Mentionnons, pour finir, le cas où ... il n'y a pas d'apprentissage : les poids des connexions sont *câblés* à l'avance, comme c'est le cas pour les réseaux de Hopfield utilisés en optimisation ou pour les réseaux compétitifs [Hérault & Jutten 94].

I.3.4 Les caractéristiques communes des approches "réseaux" en traitement de l'information

Les comportements matériels et cognitifs sont-ils si différents quand on les simule informatiquement à l'aide de réseaux ? Nous allons montrer que la réponse est négative, et nous proposons pour cela d'énumérer les caractéristiques que l'on retrouve dans ces deux champs de recherche.

La généricité

Que ce soit en modélisation physique ou en modélisation cognitive, l'idée a été de trouver une petite unité simple, mais capable, par assemblage, de réaliser les structures les plus complexes. On manipule un "jeu de construction" contenant quelques éléments de base, mais aux larges possibilités.

L'émergence

C'est l'équivalent de la généricité, mais d'un point de vue *fonctionnel*, c'est-à-dire en s'intéressant aux *fonctions* réalisées par les structures. En effet, on parle de comportement émergent pour désigner sa richesse, relativement aux éléments de base du réseau. Le comportement global n'est pas réductible au comportement local, mais en émerge. Par

exemple, dans les réseaux de Hopfield [Hopfield 82], alors que les unités ne réalisent que le seuillage de la somme des entrées, le réseau est capable de se comporter en mémoire associative. De même, les réseaux de masses-ressorts-frottements de modélisation physique peuvent produire une séquences de valeurs correspondant à un son d'une grande richesse spectrale, alors que chaque unité ne réalise que quelques opérations arithmétiques élémentaires (deux multiplications et quatre additions pour les masses d'une corde).

Le parallélisme

Contrairement à d'autres techniques, l'approche "réseaux" s'adapte mieux aux nouvelles architectures des ordinateurs. En effet, même si les processeurs sont de plus en plus rapides, on arrive vite à une limite, et l'évolution de l'informatique tend vers des machines parallèles, qui permettent d'augmenter encore la rapidité de traitement. Cette contrainte amène la communauté scientifique informatique à étudier de plus près les réseaux (physiques ou cognitifs), car un réseau est intrinsèquement parallèle.

Simplicité d'utilisation

Bien sûr, les modèles neuronaux actuels sont difficiles à mettre au point lors de la phase d'apprentissage. Mais il faut noter que les algorithmes sont généralement très simples à programmer, du fait que les unités de base sont simples. Par exemple, la programmation d'un algorithme de rétropropagation du gradient consiste en quelques boucles et nécessite une centaine de lignes. Dans le domaine de la simulation physique, il est plus simple de programmer un réseau linéaire masses-ressorts-frottements [Cadoz *et al.* 93] que de résoudre numériquement une équation différentielle (même si dans certaines conditions, les deux choses sont équivalentes).

La représentation graphique

Dans tous les cas de réseaux, on remplace une formulation mathématique globale par un schéma de nœuds et d'arcs. C'est souvent une aide à la compréhension, mais pas dans tous les cas, en particulier pour les réseaux connexionnistes, quand il s'agit de faire figurer l'ensemble des connexions (voir la figure II.11 du paragraphe II.2.3.b).

Modèles éloignés de leur référence naturelle

Finalement, même si les réseaux existent dans la nature (réseaux cristallins, moléculaires, de neurones biologiques, etc.), les réseaux utilisés dans les simulations informatiques s'en sont éloignés, car l'ordinateur, limité en capacité, généralement séquentiel et discret, a apporté ses propres contraintes artificielles.

Modèles intermédiaires

Il est intéressant de rapprocher la nature ni microscopique ni macroscopique des réseaux de modélisation physique (voir paragraphe I.3.2) de la nature ni structurelle ni fonctionnelle des réseaux de modélisation cognitive (voir paragraphe I.3.3.b). Les réseaux se placent donc à un niveau intermédiaire et fluctuant. C'est peut-être de là que proviennent leur richesse.

*

Nous concluons de cette liste que les réseaux de modélisation physique et les réseaux de modélisation cognitive sont très proches, car ils sont maintenant avant tout contraints par les caractéristiques des ordinateurs passés et actuels.

Où se trouve la différence ? Bien sûr les automates "physiques" et les automates "cognitifs" n'ont pas les mêmes fonctions de transfert, mais il s'agit là d'une différence superficielle.

Le différence est dans l'apprentissage et comme c'est justement l'objet de notre recherche de doter les réseaux physiques de capacités d'apprentissage, nous en discutons plus longuement dans le paragraphe qui suit.

I.3.5 Et l'apprentissage ?

Nous parlons d'apprentissage, car c'est le terme consacré en connexionnisme. Le terme d'adaptation est lui aussi employé, à propos des systèmes capables de se modifier en restant en perpétuelle interaction avec l'environnement [Widrow & Stearns 85][Nerrand *et al.* 93][Haykin 94]. Pour notre étude, nous employons indifféremment les termes "apprentissage" et "adaptation", auxquels nous donnons la définition très générale suivante : *modification durable du "système apprenant", sous l'influence de l'environnement et dans un but donné.*

Si "apprentissage" et "modèles physiques" semblent deux notions peu compatibles, l'*adaptation* dans les modèles physiques prend plus de sens. En effet, en ce qui concerne le monde physique, les exemples d'adaptation sont fréquents : nous savons tous qu'un stylo à plume s'adapte à son utilisateur principal si bien qu'il devient difficilement utilisable par un autre que son propriétaire ; il en est de même pour des instruments de musique tels que le violon ou la guitare.

L'adaptation met alors en jeu des phénomènes d'usure et de déformation durable que l'on peut essayer de modéliser.

D'autres systèmes matériels ont des propriétés d'adaptation très intéressantes, comme les verres de spin [Weisbuch 89] qui servent justement de base à des modèles de mémoire associative [Hopfield 82], ou certains milieux optiques.

Cependant, nous ne trouvons pas dans la nature de système physique capable de s'adapter de manière à reproduire un comportement dynamique donné comme c'est le cas pour les systèmes cognitifs. La pratique du moulage ou de la gravure sont certes bien des procédés visant à imposer à un objet physique une certaine configuration, mais il s'agit de configurations *statiques* ; le "moulage dynamique", c'est-à-dire l'apprentissage temporel supervisé dans les systèmes physiques, à notre connaissance n'existe pas dans la nature.

C'est pour cette raison que les réseaux de modélisation-simulation physique n'ont pas pour l'instant de propriétés d'apprentissage temporel supervisé.

En ce qui concerne les réseaux connexionnistes, l'apprentissage est au cœur du débat. Cependant, le statut de l'apprentissage dans les réseaux n'est pas clair, et nous allons tenter ici une petite mise au point.

Nous avions précisé qu'un des points forts des réseaux de neurones artificiels est la possibilité d'une représentation graphique des traitements, liée à la possibilité de futures réalisations matérielles. Or, dans la majorité des cas *les traitements réalisés par l'apprentissage lui-même ne sont pas explicités sous forme de réseaux.*

Ce choix d'omettre l'explicitation de l'apprentissage sous forme de réseaux se justifie si et seulement si on considère que l'apprentissage n'est que le moyen de trouver les paramètres. Après apprentissage on peut alors réaliser matériellement le réseau au niveau de la dynamique de ses activités, ce qui est plus simple que de réaliser à la fois les dynamiques des activités et des poids. Cette approche existe aussi en modélisation psychologique, quand on considère que le réseau nous intéresse pour modéliser une capacité cognitive figée, et que l'apprentissage n'est que le moyen technique pour programmer le réseau.

Précisons alors que, selon cette approche :

- Ce n'est pas le réseau qui apprend : parlons alors d'apprentissage *pour* les réseaux.
- Toute technique d'apprentissage est envisageable : optimisation combinatoire, algorithmes génétiques, etc.

- l'apprentissage est non incrémental [Orsier 95 p. 136] dans la mesure où l'utilisation et l'apprentissage sont dissociés.
- Sur le plan de la modélisation cognitive, on n'aborde pas le problème de l'apprentissage (ou bien par d'autres techniques, et le modèle est alors hybride).

Si, au contraire, c'est l'apprentissage pour et *par* les réseaux qui nous intéresse, alors il faut expliciter *sous forme de réseaux* la dynamique des poids. Sur la figure I.18, nous avons représenté d'un côté un réseau à couches tel qu'il est habituellement représenté [Le Cun 87][Rumelhart & McClelland 86][Amy & Decamp 87][Davalo & Naïm 90][Hérault & Jutten 94], de l'autre le réseau tel qu'il doit être représenté si l'on s'intéresse à l'apprentissage par les réseaux, où figurent à la fois les dynamiques d'activation et d'apprentissage. Cette représentation des deux dynamiques est rare : dans [Narendra & Parthasarathy 90][Haykin 94], on trouve la représentation des deux dynamiques sous forme de graphes architecturaux ; dans [Stork 89], chaque traitement élémentaire est représenté par un automate, dans le but d'examiner une éventuelle plausibilité biologique de la rétropropagation du gradient dans les réseaux à couches. Dans tous les cas, dès que l'on parle de réseaux qui apprennent, il faudrait expliciter un tel schéma.

Ainsi, les réseaux connexionnistes ont des capacités d'apprentissage, mais il faut tout de même leur ajouter des connexions, ou les compliquer, et souvent ajouter des unités.

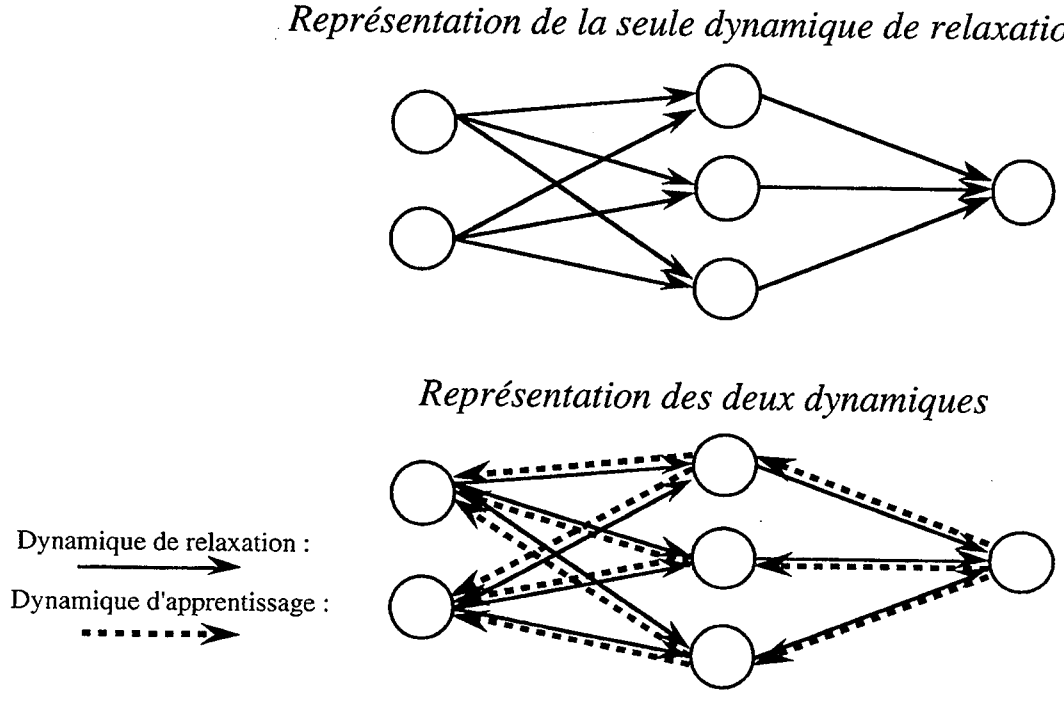

Figure I.18. Les deux flots d'information dans les réseaux à rétropropagation

I.4 Position de notre approche

I.4.1 Les différentes méthodes pour l'identification de paramètres d'un modèle physique

Le problème qui nous préoccupe, celui de l'identification de paramètres dans un modèle physique (voir paragraphe I.1.2) peut être abordé de diverses manières, que nous proposons de passer en revue. Cette présentation est très générale, et non exhaustive. En particulier, nous n'abordons pas en détail le problème de l'estimation de paramètres dans les guides d'ondes [Smith 92], technique la plus souvent utilisée en synthèse sonore par modèle physique, car cette technique manipule des modèles qui relèvent plus du traitement du signal comparativement aux modèles physiques qui nous intéressent (réseaux "masses-ressorts-frottements").

De manière générale, le problème se formule comme suit : "étant donné d'un côté un objet physique dont on peut connaître la sortie en fonction de l'entrée, et de l'autre un modèle physique *générique* de cet objet, c'est-à-dire un modèle qui selon les paramètres qu'on lui fournit s'instancie en divers modèles *spécifiques*, comment trouver de manière systématique et automatique les paramètres du modèle afin qu'il reproduise le comportement de l'objet physique ?

I.4.1.a Approches indépendantes du modèle

Une première classe de méthodes consiste à résoudre le problème d'optimisation ci-dessus mentionné indépendamment des algorithmes de calcul du modèle physique. Ce dernier est alors une boîte noire ayant en plus des entrées/sorties de signaux physiques une entrée de paramètres $(p_1,p_2, \ldots ,p_n)$, comme illustré sur la figure I.19.

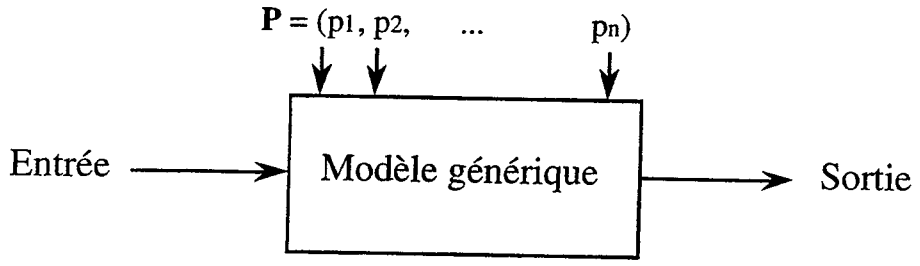

Figure I.19. Le modèle physique en tant que boîte noire

Deux options sont alors envisageables :

Optimisation dans l'espace des paramètres

L'objectif de l'apprentissage est de trouver un jeu de paramètres $P = (p_1,p_2, \ldots ,p_n)$ qui minimise la différence entre le comportement de l'objet et celui de son modèle, tous les deux étant excités par une entrée quelconque e_i. Pour cela, on définit une mesure de différence ou d'erreur E fonction du vecteur P et d'un ensemble d'apprentissage donné : $E(P,\{(e_1,d_1),(e_2,d_2), \ldots ,(e_n,d_n)\})$ avec (e_i,d_i), le $i^{ème}$ couple de l'ensemble d'apprentissage.

A tout P est associée une valeur de l'erreur E. On peut alors utiliser un algorithme d'optimisation quelconque (recherche itérative, recuit simulé [Bonomi & Lutton 88], algorithmes génétiques, etc.) pour trouver le vecteur P optimal (voir figure I.20).

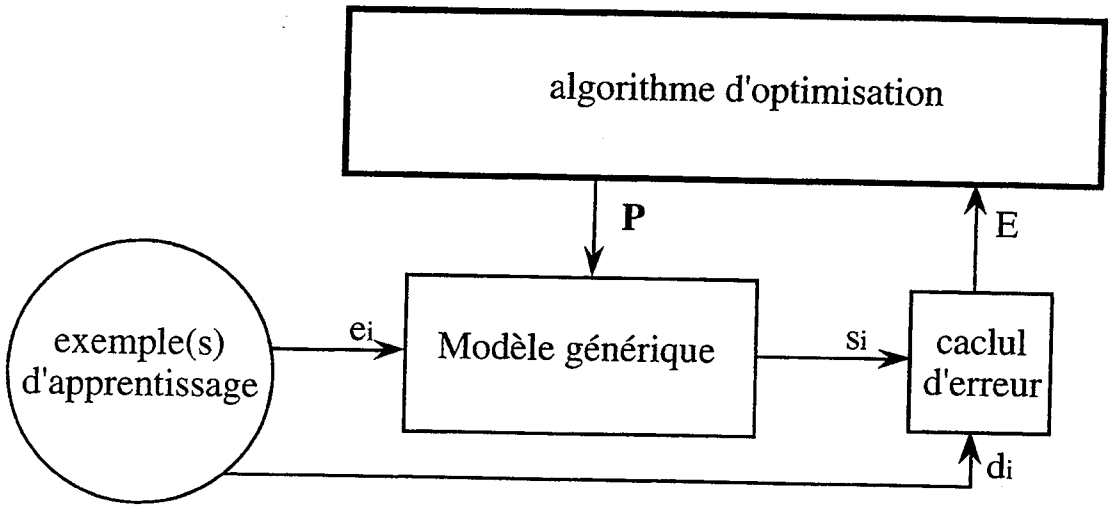

Figure I.20. Identification par optimisation dans l'espace des paramètres

Par exemple, dans [Vuori & Vätimäki 93], pour déterminer les paramètres d'un modèle de flûte, on utilise un algorithme génétique pour minimiser une erreur spectrale (la différence de position des huit premières raies).

Dans [Louchet 94], pour identifier les paramètres d'un modèle à base de masses-liaisons utilisé en animation, on utilise encore des algorithmes génétiques appliqués à un ensemble de fonctions d'erreur locales, c'est-à-dire relatives à une zone limitée du modèle.

Ces techniques présentent deux points forts : d'une part on peut choisir un critère d'erreur le plus pertinent par rapport au problème, c'est-à-dire y introduire une connaissance a priori sur la nature du problème. D'autre part, on peut utiliser un algorithme d'optimisation puissant capable de trouver une solution optimale, évitant en particulier le problème des minima locaux. Par contre, de tels algorithmes sont coûteux en temps, comparativement à de techniques utilisant les dérivées (voir par exemple [Minoux 83] pour un exposé de ces techniques, et [Bengio 94] pour une comparaison expérimentale).

Inversion de la fonction "paramètres -> comportement"

Etant donné le modèle générique de la figure I.19, on peut, pour une entrée e et un jeu de paramètres $\boldsymbol{P}$, connaître la sortie s. Il est alors possible de construire un ensemble de triplets $(e_i,\boldsymbol{P}_i,s_i)$. Cet ensemble, construit d'après la fonction "(e,**P**) -> s" peut être ensuite utilisé pour trouver la fonction "(e,s) -> **P**" à l'aide d'une technique d'approximation. Si l'on dispose d'une bonne approximation de cette dernière fonction, alors le problème est résolu, puisqu'à partir d'un comportement désiré (e,d) on peut déduire les paramètres associés. Sur la figure I.21, qui illustre cette technique, on a donc (étape 1) un module explorateur qui génère des couples $(e,\boldsymbol{P})$, transmis au modèle générique qui fournit la sortie s correspondante. Le triplet $(e,\boldsymbol{P},s)$ alimente ensuite un module approximateur de la fonction "(e,s) -> **P**". Quand cette fonction est correctement approximée (étape 2), un exemple de l'ensemble d'apprentissage est fourni au module approximateur qui donne les paramètres au modèle générique. Dans ce cas, un seul exemple d'apprentissage est considéré ; dans le cas d'un ensemble d'apprentissage contenant n exemples (e,d), il faut envisager n couples (e,s) à l'entrée de l'approximateur.

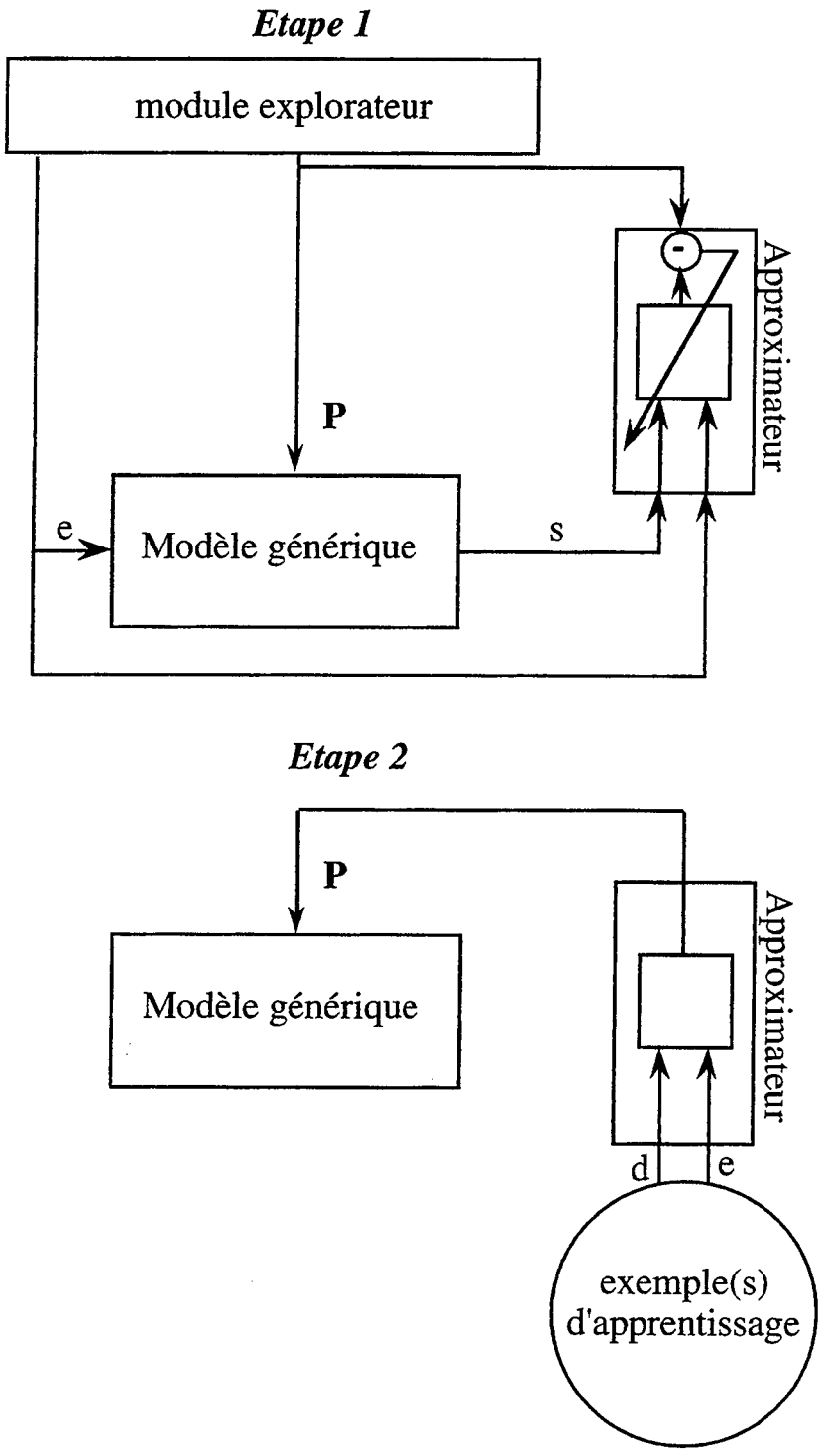

Figure I.21. Identification par inversion de la fonction "paramètres -> comportement"

Contrairement à la technique précédemment décrite, nous n'en connaissons pas d'exemples de mise en œuvre et ne savons donc pas si elle est réalisable ou pas. Son avantage est le suivant : la détermination des paramètres se fait en deux étapes, l'une permettant d'"inverser" le modèle générique, l'autre d'obtenir les paramètres d'un modèle spécifique. Seule la première étape est coûteuse en temps, la deuxième n'étant que l'application d'une fonction de transfert. Donc, si l'on choisit de garder toujours le même

modèle générique, une fois l'apprentissage du modèle inverse réalisé, il est possible de déterminer, avec un coût insignifiant, autant de modèles que l'on souhaite.

Par contre, tout le succès de la méthode repose sur l'approximateur, qui n'est pas sans soulever des difficultés. Tout d'abord, on peut se demander si la fonction qui doit être approximée existe effectivement, dans la mesure où plusieurs jeux de paramètres peuvent donner la même relation d'entrée-sortie. Si ce problème est résolu, il faut de plus espérer que l'algorithme choisi réussira à trouver une fonction approximatrice satisfaisante. Enfin et surtout, il est essentiel que la fonction obtenue donne des résultats corrects sur des exemples non appris (généralisation), si l'on veut identifier les paramètres d'un comportement non envisagé par le module explorateur. Il est donc prudent de ne pas s'engager sur la faisabilité d'une telle technique.

I.4.1.b Analyse modale

L'analyse modale concerne les systèmes mécaniques linéaires. Pour modéliser de tels systèmes, il existe des techniques qui permettent de les représenter par un système fini d'équations, ce qui se note matriciellement par :

$$MX'' + ZX' + KX = F$$

où M est la matrice des masses, Z et K sont les matrices des viscosités et raideurs respectivement, X les coordonnées spatiales (positions) des degrés de liberté du système et F les forces extérieures. On peut supposer (quitte à faire un changement de coordonnées) que M est diagonale, et que K et Z sont symétriques. Dans ce cas, les équations matricielles représentent un réseau de masses interconnectées par des liaisons viscoélastiques.

Sous certaines conditions, en effectuant un changement de repère spécifique, on peut se ramener à l'équation suivante [Djoharian 93] :

$$Y'' + Z_m Y' + K_m Y = G$$

où K_m et Z_m sont deux matrices *diagonales*, et Y et G sont les positions et forces dans le nouveau repère. Cette deuxième équation représente une série d'oscillateurs indépendants de masse unitaire (voir figure I.22).

Mathématiquement, le changement de repère est tout simplement une multiplication par une matrice ${}^tQ$ à l'entrée, et en sortie une multiplication par la matrice Q (${}^tQ$ signifie transposée de Q) : $G = {}^tQF$ et $X = QY$. Les matrices K_m, Z_m s'obtiennent selon les formules ${}^tQKQ = K_m$ et ${}^tQZQ = Z_m$. Il s'agit donc d'un problème de diagonalisation de matrices. Pour plus de détails, voir [Djoharian 93].

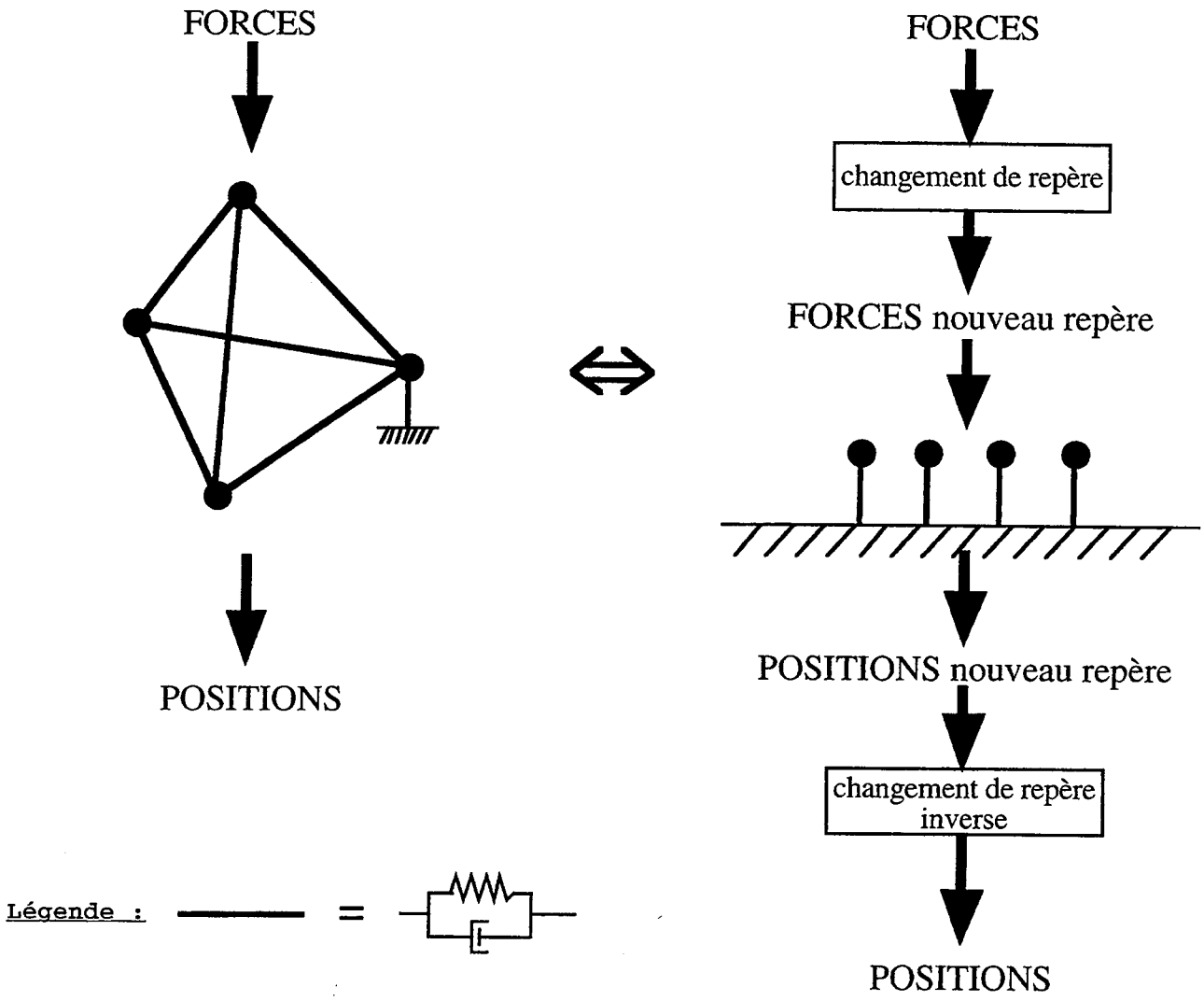

Figure I.22. Représentation modale d'un réseau

L'importance de ce changement de repère est le suivant : les oscillateurs indépendants génèrent chacun un signal sinusoïdal amorti, dont la somme donne le comportement libre du réseau ; Ainsi, à chaque oscillateur, que l'on appelle aussi *mode*, correspond une fréquence. L'ensemble de ces fréquences est le même que l'ensemble des fréquences obtenu par décomposition de Fourier de la réponse impulsionnelle du système. Ainsi, l'opération d'identification des paramètres s'en trouve grandement simplifiée : par exemple, une simple analyse de Fourier sur la réponse du système à une impulsion permet de déterminer les fréquences des modes du réseau. En fait, de nombreuses techniques existent pour déterminer les paramètres du modèle modal (*Km* et *Zm*), techniques que l'on regroupe sous le terme d'"analyse modale expérimentale", et sont largement utilisées en ingénierie pour caractériser les comportements vibratoires de structures variées [Ewins 84].

L'analyse modale nécessite trois entités : un générateur de tests sur la structure matérielle à identifier, un analyseur et un module de changement de base (voir figure I.23). Typiquement, le générateur de tests produit une succession d'excitations

sinusoïdales de fréquence croissante, afin de déterminer la réponse fréquentielle du système. L'analyseur peut traiter les données de différentes manières, que ce soit à l'aide d'une méthode de détection de pics, de moindres carrés ou de gradient. Quant au module de changement de base, celui-ci consiste simplement en quelques multiplications matricielles, si la matrice Q est entièrement déterminée par l'analyseur. Quand ce n'est pas le cas, plusieurs matrices M, K et Z sont alors possibles, et il faudra contraindre la solution.

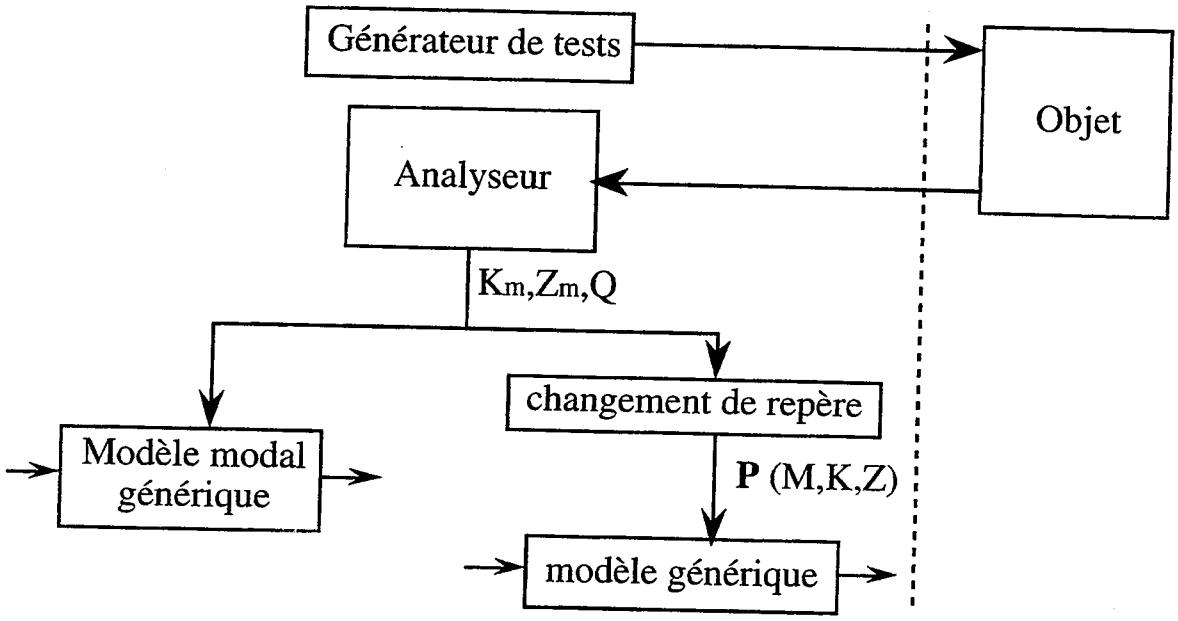

Figure I.23. Analyse modale expérimentale

I.4.2 Les modèles physiques auto-adaptatifs

Après avoir exposé différentes approches possibles vis à vis du problème initialement posé, nous allons définir l'approche choisie dans cette étude, compte tenu des analyses qui ont fait l'objet des chapitres I.1, I.2 et I.3. Nous sommes amenés à soulever un certain nombre de questions auxquelles ce document va tenter de répondre.

Au paragraphe I.2.2, nous pointions la symétrie fonctionnelle entre la synthèse et l'analyse. C'est notre objectif de trouver un mécanisme d'analyse structurellement symétrique par rapport à la synthèse, comme c'est le cas pour l'analyse/synthèse par série de Fourier. La première question que nous posons est donc :

Questions 1 : «Est-il possible de réaliser un mécanisme d'analyse d'objets mécaniques, symétrique du mécanisme de synthèse ?»

Par ailleurs, nous avons précisé au paragraphe I.3.5 les deux positions possibles en ce qui concerne l'apprentissage dans les réseaux connexionnistes. Pour notre étude, nous nous intéressons à l'apprentissage *pour* et *par* les réseaux. En effet, notre motivation est

d'utiliser le plus possible la représentation physique. Une question concernant l'ensemble de ce travail est donc :

Question 2 : «A quel niveau se situe le seuil à partir duquel des représentations non physiques doivent intervenir dans l'apprentissage ?»

De manière plus concrète, nous disposons de réseaux de modélisation physique (réseaux CORDIS), non adaptatifs, et des réseaux connexionnistes, adaptatifs. Ayant constaté les similitudes entre ces deux types de réseaux, il semble possible d'ajouter la propriété d'adaptation aux réseaux CORDIS. La question est alors :

Question 3 : «Quelles propriétés et structures faut-il ajouter aux réseaux physiques pour en faire des réseaux physiques adaptatifs ?»

Dans ce travail, nous nous efforçons de rechercher les propriétés et structures *minimales* non seulement pour des questions de rapidité de simulation, mais aussi par souci de simplicité.

Enfin, pour que l'analyse puisse, au même titre que la synthèse, être efficace, nous tenterons de répondre à la question suivante :

Question 4 : «Les algorithmes obtenus sont-ils aisément parallélisables ? »

Ayant défini notre approche, nous souhaitons souligner en quoi elle se distingue des autres.

Tout d'abord, l'originalité essentielle est d'essayer de ne pas situer la tâche d'apprentissage ou d'adaptation à un autre niveau que celui du réseau physique lui-même.

En effet, un certain nombre de méthodes (I.4.1.a) adaptent les paramètres du modèle sans tenir compte (ou peu) du type de calculs impliqués dans le modèle physique : un algorithme *observe* le système et optimise ses paramètres (voir figure I.20). En ce qui concerne l'analyse modale, la spécificité du modèle est exploitée, mais les modules d'adaptation, à savoir le générateur de test et l'analyseur (voir figure I.23), se situent à un autre niveau que celui du modèle lui même.

Dans l'ensemble de ces méthodes, les représentations "physiques", c'est-à-dire analogiques, sont dissociées des fonctions cognitives d'adaptation. Selon notre approche,

il est au contraire intéressant d'étudier dans quelle mesure il est possible, en ce qui concerne l'adaptation, de superposer les deux niveaux (cognitifs et analogiques), pour réserver les niveaux supérieurs à d'autres tâches (planification, choix de stratégies, etc.). Il a été montré qu'on peut effectivement confier au niveau "physique" bien plus que la simple re-présentation du réel, en planification par exemple [Delnondedieu *et al.* 93].

La résolution de problèmes peut donc dans certains cas se passer de traitements symboliques de haut niveau : une *résolution physique* peut suffire. Ceci renvoie à la problématique plus générale du rôle des représentations *analogiques* dans la cognition [Denis 92]. Notre "objectif" - il s'agit en fait d'un objectif idéal, un point de mire, car nous ne sommes pas sûrs encore qu'il soit réalisable - est d'utiliser la seule représentation physique pour réaliser l'adaptation dans les modèles physiques. Si cet objectif est atteint, il sera alors possible d'envisager - même si cela n'est pas notre but - la réalisation concrète d'une "matière adaptative", capable d'effectuer le moulage dynamique que nous évoquions au paragraphe I.3.5, bref une "pâte à modeler dynamique" !

Par ailleurs, une étude bibliographique a mis en évidence que les réseaux adaptatifs ne sont pas utilisés pour modéliser des comportements physiques de manière *structurelle*.

Dans [Wu *et al.* 90] par exemple, un réseau à couches à rétropropagation du gradient est utilisé pour modéliser les contraintes mécaniques d'un objet soumis à certaines charges: on modélise certes un comportement mécanique mais on n'obtient aucune information sur la *structure* qui engendre ce comportement, car le réseau est utilisé en tant que boîte noire.

Un petit pas supplémentaire est réalisé dans [Cox & Darlington 94] : toujours avec un réseau à couches, on modélise un phénomène très fortement non linéaire de propagation acoustique après une explosion. Comme précédemment, le résultat n'est pas interprétable en terme de structure. Dans une autre expérience cependant, la fonction de transfert "sigmoïde" est légèrement modifiée pour tenir compte des propriétés acoustiques du milieu de propagation : le résultat s'en trouve amélioré. Malheureusement, il n'est pas clair que la cause de l'amélioration soit véritablement l'introduction d'une composante de modélisation physique dans le réseau.

Ces études mettent en évidence le fait que l'on peut considérer le sujet qui nous intéresse en se plaçant du point de vue des Réseaux de Neurones Formels: il s'agit d'ajouter au niveau de chaque automate et chaque connexion d'un réseau de neurones classique une connaissance a priori liée à la nature mécanique des phénomènes modélisés.

Il est important de signaler que l'approche choisie répond plus à un souhait de compréhension de la relation entre apprentissage et modélisation physique qu'une recherche de l'algorithme le plus efficace à court terme. Si en effet ce souci d'efficacité avait été notre préoccupation centrale, certaines approches exposées au paragraphe I.4.1 auraient certainement été préférables, en particulier l'analyse modale expérimentale.

I.4.3 Apprentissage et action

Nous avons défini l'apprentissage par une modification interne durable d'un système, sous l'influence de l'environnement et dans un but donné. Dans notre contexte d'apprentissage supervisé, le but en question est de *reproduire un comportement extérieur*.

Globalement, l'apprentissage revient donc à un transfert d'information unidirectionnel de l'environnement vers le système. Mais faut-il pour autant considérer que *localement*, c'est-à-dire à chaque instant de l'apprentissage, l'échange d'information est aussi unidirectionnel ? Nous avons au contraire montré au paragraphe I.2.3 qu'en ce qui concerne les comportements mécaniques, tout échange est nécessairement bidirectionnel.

L'apprentissage unidirectionnel serait alors un cas dégénéré (au sens mathématique) de l'apprentissage, généralement bidirectionnel ; essayer de développer des algorithmes d'apprentissage unidirectionnels revient peut-être à essayer de faire marcher quelqu'un avec une seule jambe...

Nous allons donc étudier l'apprentissage bidirectionnel, ce qui revient à introduire *l'action* dans l'apprentissage : le système qui apprend n'est plus passif mais agit sur l'environnement qui l'entoure. Ainsi, nous avons décidé de ne pas uniquement nous focaliser sur l'interaction mécanique, mais sur l'interaction en général, durant le déroulement de l'apprentissage. Nous complétons nos quatre questions précédentes par une cinquième question :

Question 5 : «Quels sont les rôles que peut avoir l'action lors de l'apprentissage, et quels types de traitements sont alors mis en jeu ?»

Les deux grandes parties qui suivent reprennent les deux composantes de l'apprentissage que nous venons de mentionner : la modification durable de l'état interne ("Apprentissage dans les réseaux récurrents") et l'interaction avec l'environnement ("Apprentissage actif").

DEUXIÈME PARTIE

APPRENTISSAGE DANS LES RESEAUX RECURRENTS

Cette deuxième partie traite de l'apprentissage dans les réseaux récurrents, et en particulier dans les réseaux CORDIS. Elle montre comment on peut passer des réseaux "neuronaux" aux réseaux CORDIS, et comment les analyses, les problèmes et les solutions apportées dans chacun des deux champs d'étude se répondent.

II.1 Des réseaux CORDIS aux réseaux récurrents supervisés

II.1.1 Les réseaux CORDIS

Les réseaux CORDIS [Cadoz *et al.* 93] sont les réseaux de modélisation physique développés à l'ACROE. Ils sont essentiellement utilisés pour simuler sur ordinateur des objets mécaniques déformables, qui peuvent servir soit à produire des sons, soit à produire des images de synthèse animées, soit les deux associés. De plus, les signaux produits par les modèles CORDIS sont envoyés à un dispositif à retour d'effort, pour combiner aux simulations sonores et visuelles un retour d'effort tactile. Les domaines d'application sont donc l'informatique musicale, l'image de synthèse, la robotique et la communication homme-machine. Après avoir explicité la démarche, ou plutôt les démarches qui ont abouti à ce type de réseaux, nous allons spécifier les algorithmes réalisés par les unités de traitement.

II.1.1.a Trois approches pour les réseaux CORDIS

La discrétisation de la matière et du temps

Comme nous l'abordions au chapitre I.3.2, l'architecture de l'ordinateur impose des contraintes sur la simulation d'objets physiques. En effet, les ordinateurs actuels manipulent toujours des entités *discrètes* alors que les théories de la physique manipulent généralement un formalisme *continu*. Il convient donc, pour réaliser des simulations efficaces, de *discrétiser* la physique classique.

La première discrétisation est celle de la matière. Soit un corps physique donné, dont on cherche à étudier les comportements statiques et dynamiques. Au lieu d'écrire les équations continues de la mécanique sur ce corps, on le considère, par approximation du continu, comme un ensemble fini de points obéissant aux lois de la

mécanique des points. On définit ainsi un système d'équations reliant positions, vitesses, accélérations et forces extérieures des différents points mécaniques en question. Selon les comportements mécaniques que l'on veut modéliser, le degré et le type d'approximation que l'on veut réaliser et enfin les contraintes de simulation informatique (telle la stabilité), une grande palette est alors offerte pour choisir les algorithmes de simulation.

Dans le cadre des réseaux CORDIS, en particulier en ce qui concerne la synthèse de sons, on modélise des comportements dynamiques de corps élastiques et visqueux. Cela aboutit, à l'équation matricielle suivante :

$$M X'' + Z X' + K X = Fext \qquad (1)$$

avec X le vecteur des positions des points matériels, M la matrice (diagonale) des masses, Z la matrice (symétrique positive) des viscosités, K la matrice (symétrique positive) des raideurs, et $Fext$ le vecteur des forces extérieures.

Etant donnés les caractéristiques des matrices M,Z,K, l'équation précédente peut s'interpréter comme la description de la dynamique d'un *réseau*, dont les nœuds sont des points matériels où sont concentrées les inerties (matrice M) et les arcs sont des éléments de liaison, sièges des viscoélasticités, mais sans inertie (matrices Z et K). Ainsi, une telle discrétisation permet d'aboutir à une interprétation physique des calculs effectués : la matière est décomposée en "petites masses" (points matériels) reliées ensemble par des ressorts et des freins (voir figure II.1).

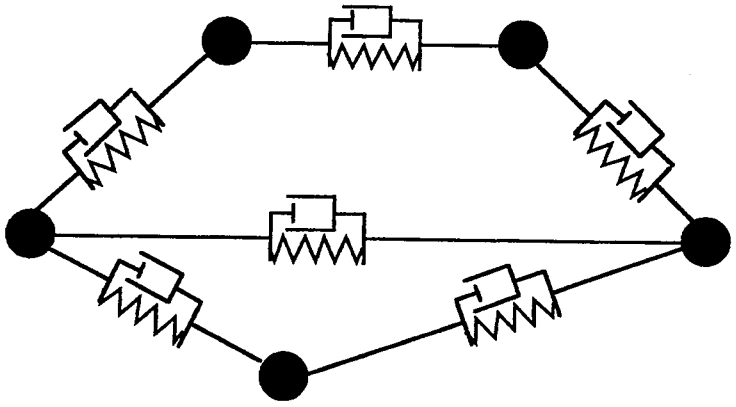

Figure II.1. Réseaux mécaniques

La deuxième discrétisation est celle du temps. Afin de pouvoir effectivement simuler sur ordinateur l'équation (1), il faut considérer le temps comme une suite d'instants, à savoir t, $t+1$, $t+2$, etc. Ensuite, se pose le problème de l'estimation des dérivées premières et secondes du vecteur position X. Plusieurs solutions sont possibles. Par exemple, pour calculer X', on a trois possibilités. Soit t le temps discret et Te la période d'échantillonnage, on pourra en effet prendre [Raoult 91] :

- $X' = (X(t) - X(t-1))/T_e$ (vitesse retardée)
- $X' = (X(t+1) - X(t-1))/T_e$ (vitesse centrée)
- $X' = (X(t+1) - X(t))/T_e$ (vitesse avancée).

Il en est de même pour le calcul discret de X''. On choisira l'un ou l'autre des schémas de discrétisation temporelle selon les critères que l'on souhaite privilégier : simplicité de calcul, stabilité des simulations, cohérence avec la physique continue.

Modélisation par assemblage

La discrétisation de la matière conduit à un système d'équations pouvant s'interpréter comme la description du comportement d'un réseau de points matériels reliés entre eux par des liaisons de type viscoélastique. Cette notion d'éléments de matière reliés par des éléments de liaisons peut à son tour devenir le point de départ pour une modélisation structurelle des objets physiques. A l'échelle atomique, on peut représenter une molécule par un réseau d'atomes interconnectés ; à l'échelle moléculaire, on peut représenter un solide, un liquide ou un gaz par des molécules en interaction (comme les réseaux cristallins) ; à une échelle macroscopique, on peut représenter un tas de sable par des petits grains (unités de matière) qui s'échangent des forces d'action-réaction de contact [Luciani *et al.* 95] ; au niveau visible, on représentera une balle qui rebondit par un point matériel situé au centre de gravité de la balle, qui est en interaction élastique "potentielle" avec tous les éléments de l'environnement : dès que la distance qui sépare la balle d'un de ces éléments est en dessous d'un certain seuil, l'interaction a lieu. Ce dernier type de modélisation est très utilisé pour la robotique [Chanclou & Luciani 95].

C'est certainement dans cette double interprétation des réseaux CORDIS que réside tout leur intérêt : à la fois ils sont un outil de discrétisation de la matière et ils représentent, au niveau de chaque unité, un élément physique clairement interprétable. Ce n'est pas le cas d'autres techniques de modélisation, comme par exemple les éléments finis. La référence d'un modèle CORDIS est donc double : l'objet global discrétisé et le comportement local modélisé.

On remarquera que les exemples de modélisation cités ci-dessus introduisent de nouveaux types d'éléments de liaisons, telle la liaison conditionnelle (deux points matériels entrent en interaction en dessous d'une distance donnée). Ainsi, l'élément de liaison, initialement destiné à contenir un algorithme de liaison viscoélastique, peut abriter une grande quantité d'algorithmes, modélisant des phénomènes physiques variés. Afin donc de mieux comprendre ce que représente les nœuds et arcs d'un réseau CORDIS, et donc quels algorithmes ils peuvent abriter, une approche plus générale a été proposée , qui fait l'objet du paragraphe qui suit.

L'axiomatique CORDIS

Posons nous de manière fondamentale comme objectif de simuler les objets mécaniques sur les ordinateurs actuels. Les contraintes de ces derniers mettent en évidence que [Cadoz *et al.* 93] :

- les communications physiques entre l'opérateur et l'ordinateur sont ponctuelles, puisque typiquement l'information passe par un fil ;
- par ce fil, les informations ne circulent que dans un sens : les communications sont donc unidirectionnelles (cela semble une lapalissade, mais il faut se rappeler qu'à l'inverse, les communications avec les objets physiques sont intrinsèquement bidirectionnelles).

En fait d'interactions, on est alors obligé de se contenter d'entrées-sorties, et se pose alors le problème de la détermination du type de signaux qui entrent et qui sortent. Il est alors intéressant de noter que dans les systèmes physiques existent deux sortes de variables : les variables intensives (forces, pressions, etc.) et les variables extensives (positions, vitesses, etc.). On associera alors ces deux types de variables aux entrées et sorties évoquées ci-dessus, ce qui nous amène à deux types de points de communication : les uns ont pour entrée une variable intensive et pour sortie une variable extensive (dénotés *points M*) et les autres l'inverse (*points L*), comme illustré sur la figure II.2. On choisit comme couple de variables extensive/intensive les *forces* et les *positions*.

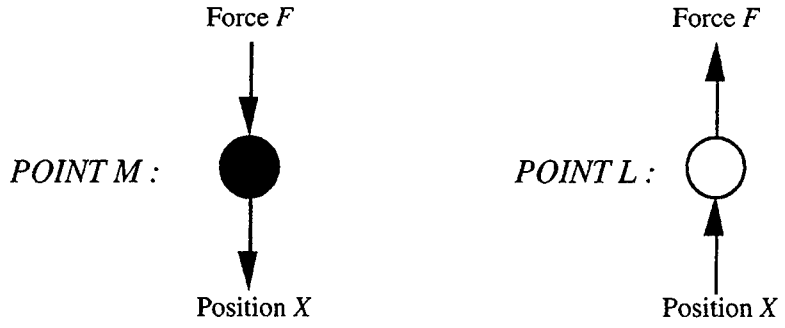

Figure II.2. Deux types de points de communication d'un réseau CORDIS

Il apparaît alors que :

- on ne peut connecter entre eux deux points L ou deux points M, mais seules les connections entre un point M et un point L sont possibles ;
- comme sur le plan physique seules les variables intensives sont additives, on peut connecter plusieurs points L à un point M mais non l'inverse.

Ensuite, étant données ces contraintes, on se propose de déterminer les plus petits éléments de base qui permettront, par assemblage, de réaliser des structures de

topologie quelconque. Par exemple, il est clair que si l'on choisit les points M et les points L comme éléments de base, alors les structures possibles sont limitées (étoiles centrées sur un point M). Ainsi, en effectuant quelques essais (voir [Cadoz *et al.* 93]), on trouve que la solution minimale, au sens de celle où chaque élément de base possède le moins de points M et L, consiste à prendre deux types d'automates : l'un ne contenant qu'un seul point M (dénoté *Point Matériel*), l'autre deux points L (dénoté *Elément de Liaison*), comme illustré sur la figure II.3. A l'aide de ces deux éléments, on peut construire une infinité de structures, qui prennent la forme d'un réseau dont les nœuds sont les Points Matériels et les arcs des Eléments de Liaison (figure II.3). On retrouve ainsi, par un parcours différent, la notion de réseau. Le contenu de chaque automate est ensuite déduit des propriétés mécaniques de la matière.

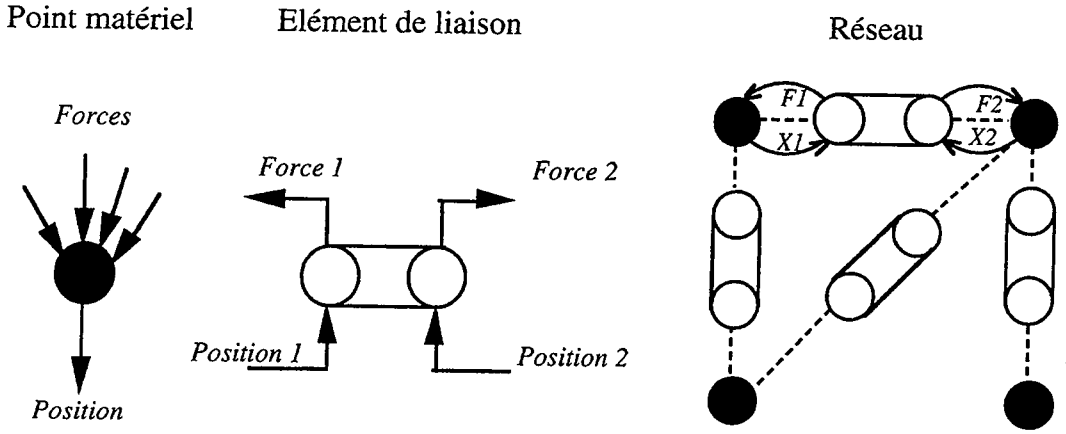

Figure II.3. Eléments constitutifs d'un réseau CORDIS

Notons que le langage *CORDIS* contient d'autres modules, dont en particulier des modules fonctionnels permettant de modifier les paramètres des autres modules. On pourra se référer à [Cadoz *et al.* 93] pour un exposé plus détaillé.

II.1.1.b Les automates

Une fois défini l'aspect général des automates des réseaux *CORDIS*, il faut spécifier quels algorithmes sont réalisés. En fait, il en existe une grande variété, essentiellement en ce qui concerne les éléments de liaison. Nous allons ici exposer le chemin qui, par approximations successives, conduit aux automates que nous tenterons de doter de capacités d'apprentissage.

En ce qui concerne les modèles d'instruments de musique, on distingue souvent l'excitateur du résonateur [Borin *et al.* 92][Boutillon & Guerard 95] : l'excitateur est l'intermédiaire entre le geste, qui évolue à une basse fréquence, et le résonateur, qui évolue à haute fréquence (voir figure II.4). Ainsi, on distingue l'archet du violon lui-même, le marteau de la corde du piano, la mailloche de la peau tendue, etc. Au

niveau de la modélisation, il faut noter que l'excitateur comporte nécessairement des éléments *non linéaires*, afin de transformer des signaux basse fréquence en signaux haute fréquence, alors que le résonateur peut être de nature linéaire. Pour ce qui concerne cette recherche, nous nous limiterons aux modèles de résonateur, en le modélisant par des éléments linéaires.

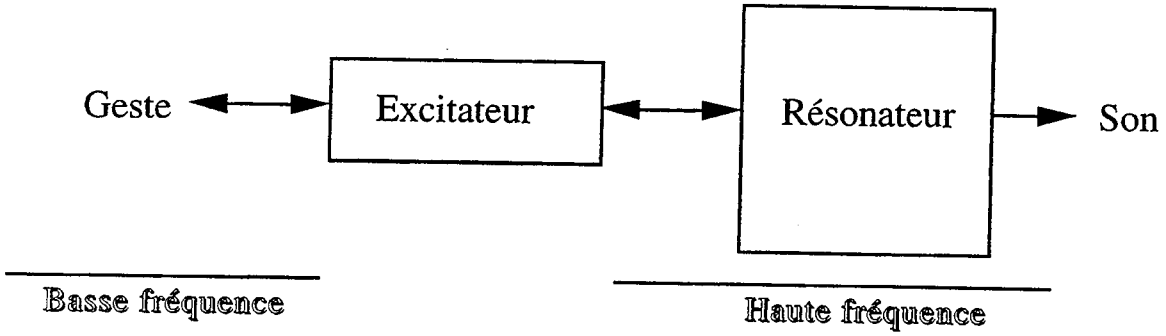

Figure II.4. Les deux parties d'un instrument de musique

Pour n'avoir effectivement que des éléments linéaires, une série d'approximation est cependant nécessaire :

- D'une part, il ne faut considérer que des liaisons à élasticité linéaire (force proportionnelle à l'élongation, comme un ressort idéal), et à viscosité proportionnelle à la vitesse. Les autres types d'interactions doivent être mis de côté, comme les frottements secs, les raideurs "cubiques" [Ewins 84], les chocs, les phénomènes de plasticité [Jimenez 93], etc.
- D'autre part, dans un modèle à deux dimensions ou plus, une telle liaison viscoélastique idéale doit déterminer la *distance* entre deux points matériels, à partir des variables de position circulant dans le réseau. Cette détermination implique le calcul d'une racine carrée, donc une non linéarité. Néanmoins, dans la mesure où, pour la synthèse sonore, ce sont les vibrations qui nous intéressent, on peut considérer que les déplacements sont de faibles élongations autour de la position d'équilibre, ce qui permet de linéariser la racine carrée dans le calcul de la distance (développement limité d'ordre 1). Ainsi, on se ramène à des modèles unidimensionnels linéaires. Ceci n'est bien sûr possible que dans le cas de la synthèse sonore, car au bout du compte, le signal de sortie est unidimensionnel, puisqu'il s'agit de variations de pression de l'air.

Une dernière transformation est effectuée pour simplifier les calculs qui consiste à faire un changement de repère dans lequel toutes les longueurs à vide sont nulles, ce qui aboutit à des modèles relativement surprenant : à l'équilibre, tous les points matériels sont à la position zéro ! Il n'y a donc plus de géométrie dans ces modèles ; on parle alors de modèles *topologiques*.

Enfin, comme nous l'écrivions au paragraphe II.1.1.a, il faut choisir un schéma de discrétisation temporelle. Parce qu'il s'agit du schéma de discrétisation aboutissant aux formules les plus simples, nous choisissons la vitesse retardée et l'accélération centrée, et prenons une période d'échantillonnage de *1*.

Nous sommes maintenant en mesure de donner les formules des deux types d'automates qui seront manipulés tout au long de ce document.

La *masse*, indicée *i*, est décrite par la loi fondamentale de la dynamique :

$$F = m_i \Gamma$$

avec F la force résultante appliquée au point matériel i, Γ l'accélération en ce point et m_i la valeur de la masse.

En écrivant cette relation en temps discret, on a :

$$\sum_{j \in V(i)} F_{ji}(n-1) + Fext_i(n-1) = m_i \frac{\left(X_i(n) - X_i(n-1)\right) - \left(X_i(n-1) - X_i(n-2)\right)}{1}$$

avec n le temps discrétisé, $V(i)$ l'ensemble des indices des masses connectées à la masse i par un élément de liaison (voisinage de i), $F_{ji}(n)$ la force délivrée à la masse i au temps n par la liaison située entre les masses i et j, et $Fext_i\ (n)$ la force extérieure appliquée à la masse i au temps n. On notera que la relation fondamentale de la dynamique a été écrite au temps *n-1* et que le *1* au dénominateur du terme de droite est le carré de la période d'échantillonnage. On obtient ensuite, par réarrangement des termes, l'équation du module *masse* :

$$X_i(n) = 2X_i(n-1) - X_i(n-2) + \frac{1}{m_i}\left(\sum_{j \in V(i)} F_{ji}(n-1) + Fext_i(n-1)\right)$$

Le *ressort amorti* (ou *liaison viscoélastique*), situé entre les masses i et j, prend en entrée deux positions et délivre deux forces, selon l'algorithme :

$$F_{ij}(n) = k_{ij}\left(X_i(n) - X_j(n)\right) + z_{ij}\left(\left(X_i(n) - X_i(n-1)\right) - \left(X_j(n) - X_j(n-1)\right)\right)$$
$$F_{ji}(n) = -F_{ij}(n)$$

avec k_{ij} et z_{ij} respectivement la raideur et la viscosité du ressort amorti.

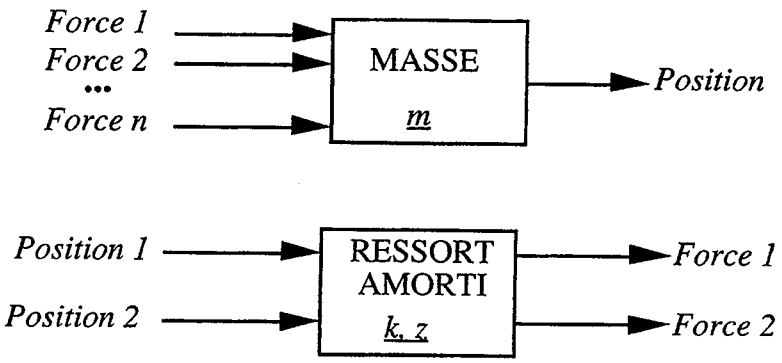

Figure II.5. Bilan des automates utilisés

Les entrées et sorties des deux automates sont illustrées sur la figure II.5. Ces automates sont connectées de telle manière que les ressorts sont toujours entre deux points matériels.

Dans toute la suite, sauf indication contraire, le terme "réseaux CORDIS" désignera les réseaux décrits par les algorithmes ci-dessus.

II.1.2 D'un formalisme à l'autre

Les réseaux de modélisation physiques que nous utilisons étant définis, l'objectif est maintenant de les mettre en relation avec les réseaux connexionnistes. La double question qui se pose est la suivante : un réseau CORDIS est-il un réseau connexionniste, et réciproquement, un réseau connexionniste est-il un réseau CORDIS ?

Si on entend par réseau connexionniste réseau d'automates à seuils, alors les réseaux CORDIS ne sont pas des réseaux connexionnistes. Mais si l'on adopte pour les réseaux connexionniste la définition très générale du paragraphe I.3.3.b (assemblée d'unités simples interconnectées), alors il est clair qu'un réseau CORDIS est un réseau connexionniste. Même si l'on restreint la définition des réseaux connexionnistes à des automates qui appliquent une fonction quelconque à la somme pondérée des entrées, alors on remarque qu'un réseau CORDIS est encore un réseau connexionniste, puisque le point matériel effectue la somme des forces, et le ressort la différence des positions.

Cependant, les réseaux CORDIS ont tout de même un certain nombre de particularités que nous nous proposons de passer en revue.

Tout d'abord, si, partant du principe que tout réseau CORDIS est un réseau connexionniste, on cherche à trouver l'application qui permet de passer d'un réseau CORDIS à un réseau connexionniste dans sa formulation habituelle, alors on se rend compte que les nœuds du réseau CORDIS ne sont pas les mêmes que ceux du réseau connexionniste correspondant. En effet, dans le formalisme des réseaux CORDIS, les arcs sont des automates, alors que dans le formalisme connexionniste, les arcs ne sont que des communications pondérées (voir figure II.6). Ainsi, un réseau CORDIS à p masses entièrement connectées possède en fait $(p+p^2)/2$ automates.

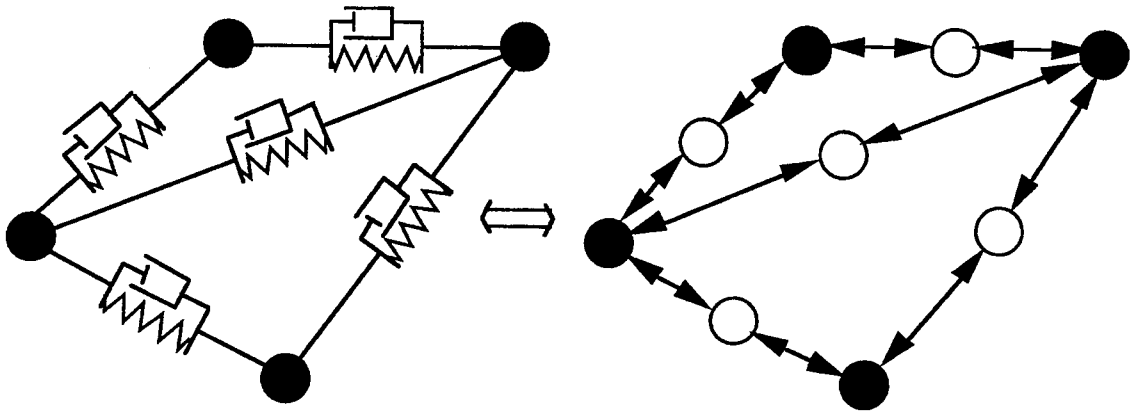

Figure II.6. Correspondance entre un réseau CORDIS et un réseau connexionniste

En conséquence, les réseaux CORDIS sont des réseaux connexionnistes qui non seulement possèdent deux types d'automates, mais en plus possèdent des règles strictes d'assemblage entre ces automates (les automates "ressorts" sont entre deux automates "masses").

Conjointement à cette dualité des automates, on a affaire à une dualité des signaux, forces et positions. Une autre originalité réside dans les automates ressorts : ceux-ci n'ont pas une mais deux sorties.

Enfin, les paramètres du réseau ne résident pas explicitement dans les poids des connexions mais plutôt dans des paramètres internes aux automates, à savoir les masses au niveau des points matériels, les raideurs et les viscosités au niveau des ressorts.

Il apparaît donc qu'un réseau CORDIS est un réseau connexionniste bien particulier, et qu'on ne peut donc pas dire qu'un réseau connexionniste est un réseau CORDIS (il n'y a pas bijection entre les deux formalismes). Notons que toutes les particularités reportées ci-dessus ne sont pas propres aux réseaux CORDIS ; Ainsi, on trouve des réseaux connexionnistes à plusieurs types d'automates, dont les unités ont plusieurs sorties, qui possèdent des paramètres internes adaptables, etc. Par

conséquent, les algorithmes développés pour les réseaux connexionnistes devraient pouvoir être adaptés au cas particulier des réseaux CORDIS. C'est ce que nous étudierons au chapitre II.3, mais auparavant, il convient de dresser un état de l'art sur le type de réseaux qui vont nous intéresser plus particulièrement, à savoir les réseaux récurrents.

II.2 Les réseaux récurrents supervisés

Ce chapitre propose un état de l'art concernant les réseaux récurrents, focalisé sur les types de réseaux qui s'accordent le mieux avec le formalisme des réseaux de modélisation physique CORDIS. En effet, un état de l'art exhaustif et uniforme nécessiterait un livre entier.

D'ailleurs, les réseaux connexionnistes sont dans la plupart des cas récurrents, même si énormément de recherches ont été consacrées aux réseaux unidirectionnels ces dernières années.

Il existe donc de nombreux types de réseaux récurrents. Dans ce chapitre seuls les réseaux supervisés sont abordés : pendant l'apprentissage, un comportement désiré est fourni par l'environnement, comportement que le réseau a pour tâche de reproduire. En particulier, les réseaux fonctionnant en mémoires associatives [Hopfield 82] [Labbi 93] ne seront pas abordés.

Dans ce qui suit est proposée une taxonomie des réseaux récurrents supervisés, suivie de l'exposé détaillé des deux algorithmes les plus utilisés, leurs limites et leurs variantes.

II.2.1 Taxonomie des réseaux récurrents supervisés

Plutôt que de présenter les différents modèles de réseaux récurrents un à un, nous avons préféré faire ressortir dans cet état de l'art les *traits* qui distinguent les algorithmes les uns des autres. Il en ressort qu'un même réseau peut être cité plusieurs fois. Pour un exposé plus didactique des réseaux récurrents, on pourra se référer à [Pearlmutter 90,95] (pour ce qui concerne les réseaux en temps continu),

[Rohwer 94], [Piché 94], les thèses [Raysz 91], [Jodouin 93], et [Crucianu 94], les chapitres des ouvrages généraux [Hecht-Nielsen 91] et [Haykin 94].

Nous allons donc énumérer et commenter les différentes caractéristiques qui permettent de distinguer les différents modèles existants.

Types de comportements appris

De même qu'on apprend aux réseaux unidirectionnels à couches des paires d'entrées-sorties statiques, on apprend généralement aux réseaux récurrents supervisés à associer des paires de patterns *dynamiques*. Selon le type de patterns traités, plusieurs cas ont été étudiés. Tout d'abord, on peut s'intéresser uniquement au comportement asymptotique du réseau, c'est-à-dire au bout d'un temps très grand. On peut ainsi faire apprendre à un réseau des attracteurs de différents types : ponctuels, cycliques voire chaotiques. Par exemple, pour l'algorithme de "rétropropagation récurrente" [Pineda 87], on fait apprendre un ensemble d'apprentissage analogue à celui qu'on utiliserait pour entraîner un réseau unidirectionnel, mais le deuxième élément de chaque paire de l'ensemble (la "sortie désirée") est l'attracteur ponctuel désiré. Une extension de ce type d'apprentissage consiste à faire apprendre une succession d'attracteurs ponctuels [Crucianu 94].

Par ailleurs, de nombreux réseaux récurrents peuvent être utilisés pour *générer* des séquences. A un pattern fixe en entrée doit être associée une succession de patterns donnée. C'est le cas de l'algorithme présenté dans [Jordan 88] utilisé en robotique pour commander un bras de robot (simulé), où un réseau à couches bouclé est entraîné à atteindre une cible donnée. Dans [Guyon *et al.* 88], une variante du réseau de Hopfield est utilisée pour stocker puis restituer une séquence donnée. Dans [Williams & Zipser 89a,b], un réseau est entraîné à produire un signal sinusoïdal (mais l'algorithme peut aussi réaliser d'autres types d'associations).

A l'inverse, on peut utiliser certains réseaux récurrents pour classifier des séquences temporelles, c'est-à-dire qu'en entrée sont présentées des séquences de patterns, et en sortie une classe statique.

Enfin, le cas général est celui où et les entrées et les sorties sont des séquences ([Robinson & Fallside 89][Williams & Zipser 89a][Rumelhart *et al.* 86][Le Cun 87][Rohwer 90], etc.). Bien sûr, tous les algorithmes qui peuvent effectuer des associations de séquences peuvent être utilisés en génération ou reconnaissance de séquences.

Structure

La structure du réseau, c'est-à-dire la topologie du graphe d'interconnexion, est variable selon les études.

La structure entièrement connectée est souvent utilisée [Rumelhart *et al.* 86 p. 354][Le Cun 87 p. 121][Pearlmutter 89], [Gherrity 89][Williams & Zipser 89][Robinson 89][Rohwer 90][Sun *et al.* 91][Toomarian & Barhen 92]. Elle est très générale, puisqu'elle permet de réaliser n'importe quelle séquence, mais donne beaucoup de calculs. De plus, l'apprentissage dans un réseau entièrement connecté est difficile.

La structure "récurrente à couches" [Jodouin 93], qui consiste en un réseau entièrement connecté à l'intérieur d'un réseau à couches (voir figure II.7) est très utilisée, en particulier par le réseau appelé "Simple Recurrent Network" [Elman 90][Jodouin 93][Crucianu 94].

Une autre structure est celle où la couche cachée possède seulement des connections auto-récurrentes [Mozer 88][Watrous *et al.* 90]. Cette structure entre dans la classe générale des algorithmes "localement récurrents globalement unidirectionnels" [Tsoi & Back 94].

Le réseau décrit dans [Jordan 88] est de type unidirectionnel bouclé, c'est-à-dire que le cœur du réseau est unidirectionnel, et la récurrence se fait des sorties vers les entrées de ce bloc unidirectionnel.

Enfin, nous mentionnons aussi la structure décrite dans [Narendra & Parthasarathy 90], qui est une structure à deux couches, la deuxième rebouclée sur la première par l'intermédiaire d'unités à retard, grâce auxquelles chaque unité de la première couche reçoit un historique temporel des unités de sortie.

Ces structures sont illustrées sur la figure II.7.

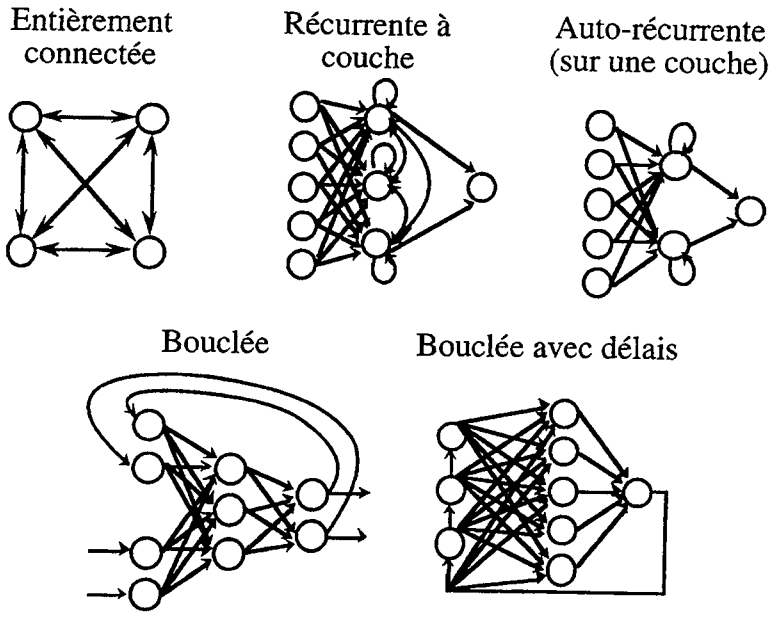

Figure II.7. Différentes structures récurrentes

Formalisme

Deux types de formalisme sont utilisés pour décrire les algorithmes, et en particulier les automates utilisés :

- Le formalisme en temps discret :

$$y_i(t+1) = f\left(\sum_j w_{ij} y_j(t)\right)$$

- Le formalisme en temps continu :

$$\tau \frac{dy_i}{dt}(t) = -y_i(t) + f\left(\sum_j w_{ij} y_j(t)\right)$$

Quels sont les avantages et inconvénients de l'un et l'autre des deux formalismes ?

D'une part, l'implantation du réseau va guider le choix entre les deux formalismes. Si l'objectif est de programmer le réseau sur un ordinateur fonctionnant en discret, alors même si l'on choisit un formalisme continu, il faudra bien discrétiser finalement les équations en continu, ce qui introduira un bruit de quantification dont le modèle ne tient pas compte, alors qu'avec les équation discrètes, il y a identité entre le modèle formel et le modèle effectivement simulé. Cette alternative se retrouve strictement en modélisation physique (voir chapitre I.3.2). Si par contre une implantation *analogique* est envisagée, dans laquelle en particulier aucune horloge ne régule les échanges d'information, alors le formalisme continu doit être utilisé [Gherrity 89][Crucianu 94].

D'autre part, les deux formalismes amènent à utiliser des théories mathématiques différentes. De notre point de vue, l'approche en temps discret conduit à des calculs plus simples ; Cependant, certains algorithmes ont été d'abord démontrés en formalisme continu [Sun *et al.* 91][Toomarian & Barhen 92].

Enfin, comme, dans les deux équations ci-dessus, la première n'est pas la discrétisation de la seconde, on n'obtient pas le même comportement. En effet, l'automate décrit en temps continu a la propriété de mieux maintenir son activité au cours du temps, même si aucun signal ne lui est transmis [Pearlmutter 90] alors que dans le cas discret, l'activité devient instantanément nulle quand aucun signal n'est transmis. En choisissant une formulation discrète qui est effectivement une discrétisation de la formulation continue, on obtient [Tsung 90]:

$$y_i(t+\Delta t) = \left(1-\frac{\Delta t}{\tau}\right) y_i(t) + \frac{\Delta t}{\tau} f\left(\sum_j w_{ij} y_j(t)\right)$$

Δt étant le pas de discrétisation.

Une telle équation reproduit le comportement en temps continu, et peut donner de meilleurs résultats que la formulation discrète classique [Lambert & Hecht-Nielsen 91]. Plus qu'une différence entre discret et continu, comme cela est soutenu dans [Pearlmutter 95], il s'agit donc d'une différence de choix de la fonction de transfert de l'automate.

De même, une autre propriété parfois avancée en faveur du formalisme continu, à savoir la possibilité de varier la discrétisation au cours du temps [Pearlmutter 95], s'avère ne pas être incompatible avec le formalisme discret, puisque dans la formule proposée ci-dessus, on peut faire varier Δt au cours du temps.

Algorithme d'apprentissage

Il existe un nombre relativement limité d'algorithmes permettant d'adapter les poids des réseaux récurrents supervisés. Ils sont fondés sur la minimisation d'une fonction d'erreur entre le comportement du réseau et le comportement désiré, et utilisent l'information de gradient. Les deux principaux sont les algorithmes de "Back-Propagation Through Time" (*BPTT*) [Rumelhart *et al.* 86 p. 354][Le Cun 87 p. 121][Pearlmutter 90] et de "Real-Time Recurrent Learning" (*RTRL*) [Williams & Zipser 89][Robinson 89][Gherrity 89] qui font l'objet du paragraphe qui suit.

L'algorithme nommé "Moving Target" [Rohwer 90] permet aussi l'adaptation des poids. Il présente cependant quelques sérieuses difficultés : d'une part il n'est pas "on-

line" (voir le paragraphe II.2.4.c pour la définition du terme), et d'autre part, selon l'auteur lui-même de l'algorithme, il est extrêmement lent [Rohwer 94].

Les algorithmes développés dans [Sun *et al.* 91] et [Toomarian & Barhen 92] sont des améliorations algorithmiques de l'algorithme *RTRL* qui réduisent la complexité des calculs. Il s'appliquent au formalisme continu.

Chacun de ces algorithmes existe en deux versions, selon que le formalisme discret ou continu est utilisé.

Un certain nombre d'algorithmes sont des cas particuliers de ces algorithmes très généraux. Par exemple, l'algorithme développé dans [Mozer 88] est en fait une application de l'algorithme *RTRL* au cas des réseaux auto-récurrents. De même, l'algorithme du "Simple Recurrent Network" est une variante de l'algorithme de *BPTT* (où le paramètre *h*, décrit plus loin au paragraphe II.2.2.a vaut *1*).

D'autres caractéristiques sont encore à mentionner : la nature des activités (binaires, discrètes ou continues), l'existence ou non d'automates cachés, le type d'automates (fonction de transfert, réseaux du premier ordre ou du second ordre [Giles *et al.* 92], etc.), la nature fixe ou évolutive des structures, etc.

Nous sommes maintenant en mesure de sélectionner parmi les caractéristiques ci-dessus celles qui correspondent aux réseaux CORDIS décrits au chapitre II.1. Les réseaux CORDIS adaptatifs que nous souhaitons concevoir sont donc :

- supervisés : on dispose de l'objet à identifier pour fournir un comportement désiré ;
- destinés à réaliser un appariement de deux séquences temporelles, à savoir le signal d'action produit par le geste et le signal de réponse produisant le son ;
- à structure quelconque : contrairement à beaucoup d'applications, nous n'avons pas à choisir la structure la plus adaptée au problème, puisque celle-ci doit être celle des réseaux CORDIS déjà existants, très variable selon les modèles ;
- avec automates cachés (du comportement d'un seul automate, le son, on cherche l'ensemble de la structure) ;
- décrits en formalisme discret ;
- à activités codées en variables décimales (non binaires).

Cela nous conduit à un certain nombre d'algorithmes, qu'il faut étudier de plus près. En particulier, nous allons nous pencher sur les algorithmes *BPTT* et *RTRL*.

II.2.2 Deux principaux algorithmes d'apprentissage dans les réseaux dynamiques

BPTT et *RTRL* sont les deux principaux algorithmes utilisés pour entraîner les réseaux récurrents supervisés. Ils sont à la fois très proches et très éloignés : à partir d'un même type de réseau (réseaux récurrents), un même critère d'erreur à minimiser (généralement une erreur quadratique), une même technique d'optimisation (descente de gradient) on dérive deux algorithmes aux caractéristiques antithétiques. Plusieurs études ont replacé les algorithmes *BPTT* et *RTRL* au sein d'un cadre théorique unitaire : [Piché 94] dans le cadre des dérivées partielles ordonnées, [Beaufays & Wan 94] dans le cadre de la théorie des graphes de flux, [Nerrand *et al.* 94] dans le cadre du filtrage adaptatif.

Il existe de nombreuses manières d'exposer ces algorithmes ; ici ont été choisies des formulations avant tout claires et didactiques, largement inspirées de [Williams & Peng 90] et [Williams & Zipser 89] respectivement. Dans le présent chapitre, seuls les algorithmes de base et leurs variantes immédiates sont exposés, pour laisser au chapitre suivant (II.2.3) l'analyse plus critique des algorithmes en question, chapitre qui mettra en évidence un certain nombre de leurs difficultés. Le chapitre II.2.4 examinera des solutions qui ont été proposées pour résoudre quelques unes de ces difficultés.

II.2.2.a Rétropropagation dans le temps

On peut voir cet algorithme comme une application de l'algorithme de rétropropagation du gradient dans les réseaux multicouches au cas des réseaux récurrents [Rumelhart *et al.* 86, p. 354][Le Cun 87, p. 121]. L'astuce consiste à représenter les états successifs d'un réseau récurrent comme autant de couches d'un réseau unidirectionnel, chaque couche correspondant à un intervalle de temps. On peut ainsi "déplier" un réseau récurrent en un réseau à couches, comme illustré sur la figure II.8. L'erreur du réseau déplié de n couches est l'erreur du réseau récurrent au temps n. On peut alors modifier les poids du réseau à couches, selon l'algorithme classique de rétropropagation du gradient. Cette modification doit se faire en respectant la contrainte selon laquelle les poids sont les mêmes d'une couche à l'autre, ce qui revient à *partager* ces poids.

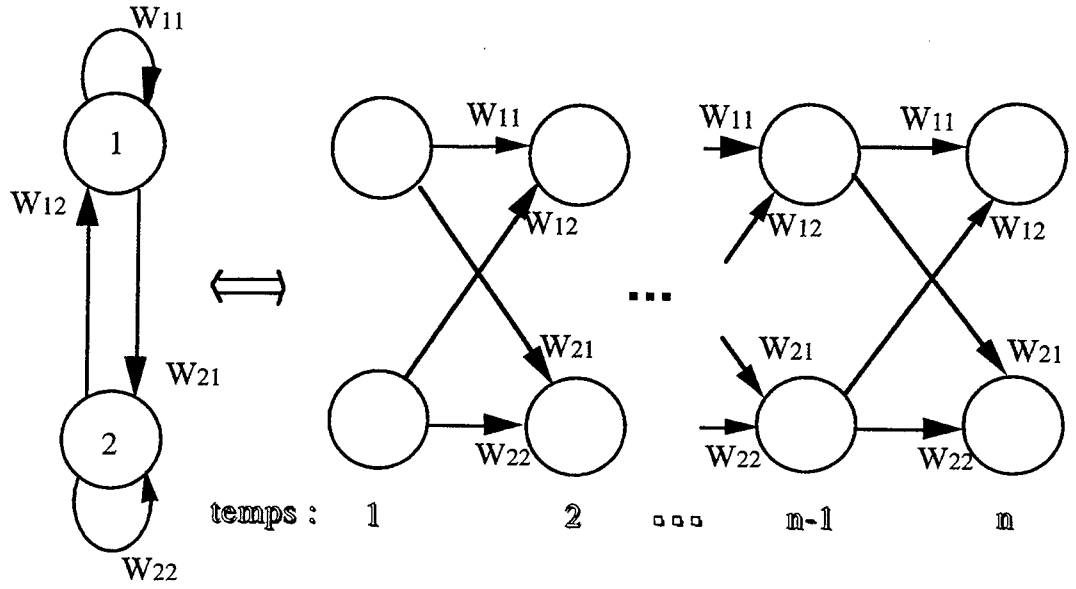

Figure II.8. réseau récurrent déplié dans le temps

Cette équivalence entre réseaux récurrents et réseaux à couches, qui autorise une explication simple et visuelle de l'algorithme *BPTT*, doit cependant laisser la place à une formulation un peu plus mathématique, qui notamment explicite clairement comment tenir compte d'une erreur à chaque instant et évite le passage par la multiplication du nombre de poids puis le partage de ces poids.

Soient :

- $x^e_i(t)$ la valeur de l'automate d'entrée i à l'instant t
- $y_i(t)$ la valeur de la sortie de l'automate i à l'instant t (son activité)
- U l'ensemble des indices portant sur les automates du réseau
- I l'ensemble des indices portant sur les automates d'entrée
- w_{ij} la valeur du poids de la connexion allant de l'automate j à l'automate i

Posons ensuite :

$$x_i(t) = \begin{cases} x^e_i(t) \text{ si } i \in I \\ y_i(t) \text{ si } i \in U \end{cases} \qquad (1)$$

Le calcul effectué par chaque unité peut s'écrire selon les deux équations :

$$\begin{cases} s_i(t+1) = \sum_{j \in U \cup I} w_{ij} x_j(t) \\ y_i(t+1) = f\left(s_i(t+1)\right) \end{cases} \qquad (2)$$

avec f une fonction différentiable. f est généralement non linéaire. Dans la pratique, on choisit souvent la fonction sigmoïde :

$$X \to f(X) = \frac{1}{1+e^{-X}}$$

qui a la propriété simplificatrice suivante :

$$f'(X) = f(X) \times (1 - f(X))$$

On remarquera que le réseau a une structure entièrement connectée. Mais cette propriété n'intervient pas dans la démonstration de l'algorithme, qui s'applique à toutes les structures.

La dynamique des activités étant définie, c'est bien sûr celle des poids qui va nous intéresser, puisqu'elle représente *l'apprentissage* du réseau.

Nous définissons alors :

- t_0, t_1 les instants de début et de fin de la séquence
- $T(t)$ l'ensemble des indices portant sur les automates pour lesquels l'environnement fournit une valeur désirée à l'instant t
- $d_i(t)$ la valeur désirée de l'activité de l'automate i à l'instant t
- $e_i(t)$ l'erreur locale de l'automate i à l'instant t :

$$e_i(t) = \begin{cases} d_i(t) - y_i(t) & \text{si } i \in T(t) \\ 0 & \text{sinon} \end{cases} \qquad (3)$$

- $E(t)$ l'erreur à l'instant t :

$$E(t) = \frac{1}{2} \sum_{i \in U} e_i(t)^2 \qquad (4)$$

- $E^{total}(t_0, t_1)$ l'erreur entre le temps t_0 et le temps t_1 :

$$E^{total}(t_0, t_1) = \sum_{\tau = t_0 + 1}^{t_1} E(\tau) \qquad (5)$$

L'objectif de l'apprentissage est de réduire l'erreur $E^{total}(t_0,t_1)$. Pour cela, on utilise la dynamique du gradient, qui consiste à définir $E^{total}(t_0,t_1)$ comme une fonction des poids du réseau et d'écrire :

$$\begin{cases} \Delta w_{ij} = -\alpha \dfrac{\partial E^{total}(t_0, t_1)}{\partial w_{ij}} & \text{(a)} \\ w_{ij}^{\ nouveau} \leftarrow w_{ij}^{\ ancien} + \Delta w_{ij} & \text{(b)} \end{cases} \qquad (6)$$

où Δwij est la modification à apporter au poids wij et α est une constante positive nommée pas d'apprentissage. Rappelons l'interprétation visuelle de cette loi de modification des poids : $E^{total}(t_0,t_1)$ en tant que fonction des poids définit une *surface* multidimensionnelle (voir figure II.9).

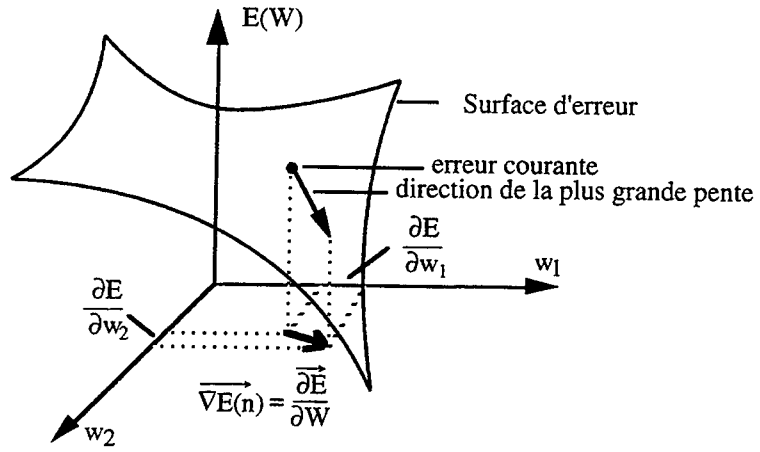

Figure II.9. Représentation graphique du principe de descente de gradient

L'état courant du réseau (dans sa dynamique des poids) définit un *point* sur cette surface ; l'objectif étant de minimiser l'erreur, on va chercher à déplacer ce point *vers le minimum* de cette surface, ce qui peut se faire comme si on "laissait rouler" le point courant *selon la ligne de plus grande pente*, autrement dit le gradient. On se rend compte que le minimum atteint peut ne pas être un minimum global et peut donc correspondre à une solution non optimale.

Dérivons maintenant les équations du réseau.

On peut écrire, d'après (6) :

$$\Delta w_{ij} = -\alpha \sum_{\tau=t_0+1}^{t_1} \frac{\partial E^{total}(t_0,t_1)}{\partial s_i(\tau)} \cdot \frac{\partial s_i(\tau)}{\partial w_{ij}}$$

car $E^{total}(t_0,t_1)$ dépend de w_{ij} via les états de l'automate i aux temps successifs entre t_0 et t_1.

Si on pose :

$$\delta_i(\tau) = -\frac{\partial E^{total}(t_0,t_1)}{\partial s_i(\tau)}$$

Alors on obtient :

$$\Delta w_{ij} = \alpha \sum_{\tau=t_0+1}^{t_1} \delta_i(\tau) x_j(\tau-1) \qquad (7)$$

Calculons maintenant $\delta_i(\tau)$:

$$\delta_i(\tau) = \frac{\partial E^{total}(t_0,t_1)}{\partial y_i(\tau)} \cdot \frac{\partial y_i(\tau)}{\partial s_i(\tau)}$$

Or $E^{total}(t_0,t_1)$ dépend de $y_i(\tau)$ d'une part via la contribution à l'erreur $e_i(\tau)$ et d'autre part via les automates qui reçoivent le signal $y_i(\tau)$. Donc :

$$\begin{cases} \dfrac{\partial E^{total}(t_0,t_1)}{\partial y_i(\tau)} = \dfrac{\partial E^{total}(t_0,t_1)}{\partial e_i(\tau)} \cdot \dfrac{\partial e_i(\tau)}{\partial y_i(\tau)} & \text{si } \tau = t_1 \\ \dfrac{\partial E^{total}(t_0,t_1)}{\partial y_i(\tau)} = \dfrac{\partial E^{total}(t_0,t_1)}{\partial e_i(\tau)} \cdot \dfrac{\partial e_i(\tau)}{\partial y_i(\tau)} + \sum_{j\in U} \dfrac{\partial E^{total}(t_0,t_1)}{\partial s_j(\tau+1)} \cdot \dfrac{\partial s_j(\tau+1)}{\partial y_i(\tau)} & \text{si } \tau \in]t_0,t_1[\end{cases}$$

Donc, on obtient :

$$\begin{cases} \delta_i(\tau) = f'\left(s_i(\tau)\right) \cdot e_i(\tau) & \text{si } \tau = t_1 \\ \delta_i(\tau) = f'\left(s_i(\tau)\right) \cdot \left[e_i(\tau) + \sum_{j\in U} w_{ji}\delta_j(\tau+1) \right] & \text{si } \tau \in]t_0,t_1[\end{cases}$$

Si l'on pose $\delta_j(t_1+1) = 0,\ \forall j \in U$, on obtient finalement :

$$\forall \tau \in]t_0,t_1],\ \delta_i(\tau) = f'\left(s_i(\tau)\right) \cdot \left[e_i(\tau) + \sum_{j\in U} w_{ji}\delta_j(\tau+1) \right] \qquad (8)$$

Comme $\{\delta_i(\tau)\}_{i\in U}$ dépend de $\{\delta_i(\tau+1)\}_{i\in U}$, il faut calculer ces variables en "remontant le temps", donc une fois toute la séquence terminée (d'où le terme de "rétropropagation dans le temps"). Les équations des signaux δ_i décrivent l'évolution dynamique d'un autre réseau, dont le graphe orienté est l'inverse du graphe orienté des signaux y_i, et que l'on nomme parfois "système adjoint" [Srinivasan, *et al.* 94][Toomarian & Barhen 94].

Cet algorithme présente l'avantage d'être peu coûteux en calculs : si le calcul de la propagation avant prend t secondes, alors le calcul des δi prendra t secondes également, auxquelles il faudra seulement ajouter le coût de la modification des poids selon (8).

Par contre, cet algorithme nécessite de stocker les signaux x_i sur toute la séquence (voir équation (7)). Imaginons un réseau de *10* automates, utilisé sur un signal de *10* secondes échantillonné à *40* kHz (cas d'un signal sonore), alors *24* Moctets sont déjà nécessaires. Nous reviendrons sur les caractéristiques computationnelles de l'algorithme *BPTT* au paragraphe II.2.3.a.

Plusieurs variantes de l'algorithme *BPTT* ont été proposées, visant à l'utiliser "en ligne" ("on-line"), c'est-à-dire à adapter les paramètres avant la fin de la séquence. Souvent, il s'agit *d'approximations* de l'algorithme original. Le principe de ces variantes est d'effectuer plusieurs rétropropagations dans le temps dans une même

séquence (puisqu'il faut faire plusieurs adaptations) et pour chacune de ces rétropropagations, jouer, selon les variantes, d'une part sur l'intervalle sur lequel est calculé l'erreur rétropropagée, et d'autre part sur le nombre de pas temporels sur lesquels est rétropropagée l'erreur.

Rétropropagation dans le temps en ligne : à chaque instant, on rétropropage l'erreur $E(t)$ définie dans (4). Pour une séquence de longueur T, il faut rétropropager T fois sur un intervalle de plus en plus long. Cette technique n'est donc pas efficace comparativement à la version de *BPTT* où l'on rétropropage une seule fois.

Rétropropagation dans le temps tronquée : c'est une variante de ce qui précède où l'on ignore les longues dépendances dans le temps. Au lieu de rétropropager jusqu'au début de la séquence, on rétropropage uniquement sur h intervalles de temps. il s'agit donc clairement d'une *approximation*, qui permet non seulement de diminuer la quantité de calculs mais aussi de la répartir uniformément dans le temps. Plus h est petit, plus forte est l'approximation et plus faible est le coût computationnel.

Rétropropagation dans le temps par blocs : il s'agit d'une extension de la rétropropagation dans le temps tronquée où l'erreur rétropropagée n'est plus calculée sur un seul instant mais sur h' pas temporels, avec $h'<h$ (pour que toutes les erreurs $e_i(t)$ soient prises en compte). Cette variante permet de rétropropager moins souvent, à savoir un nombre de fois égal à la partie entière de T/h'.

On peut représenter les quatre algorithmes (*BPTT* et ses trois variantes) dans un cadre unifié :

- soit T la longueur de la séquence
- soit t l'instant courant sur la séquence : $t \in [1,T]$
- soit b le nombre d'échantillons sur lesquels on calcule l'erreur à rétropropager
- soit p la profondeur temporelle sur laquelle on rétropropage cette erreur
- soit u l'indice de la rétropropagation courante.

alors on a, pour les quatre algorithmes, les valeurs de b et p suivantes :

	BPTT	BPTT en ligne	BPTT tronquée	BPTT par blocs
b	T	1	1	h'
p	T	t	h (pour t>h-1)	h (pour t>h'-1)

Pour les deux dernières versions, on peut en fait négliger les débuts de séquence en supposant $T>>h$, donc $t>h-1$ dans la plupart des cas (idem pour h'). Le nombre de

rétropropagations effectuées vaut T/b, soit 1, T, T, T/h' respectivement (on suppose que h' divise T).

Le coût computationnel lié aux rétropropagations est proportionnel à :

$$\sum_{u=1}^{T/b} p(u)$$

c'est-à-dire T, $T(T+1)/2$, hT, hT/h' respectivement. On remarque que la version de base reste la plus efficace.

Sur la figure II.10, nous proposons une représentation des quatre algorithmes sur un graphique temporel. L'échelle horizontale représente le temps t de la séquence et l'échelle verticale les opérations successivement réalisées par le réseau. Les flèches fines de gauche à droite représentent la propagation avant des activités x_i et leur stockage, l'échantillonnage des sorties désirées d_i, le calcul et le stockage des erreurs e_i. Les flèches en gras de droite à gauche représentent d'une part la rétropropagation relative à l'erreur courante à minimiser, c'est-à-dire le calcul des δi, et d'autre part le calcul final des Δw_{ij}. L'indice u est aussi représenté. Ce schéma résume les comparaisons que l'on peut faire entre les quatre algorithmes.

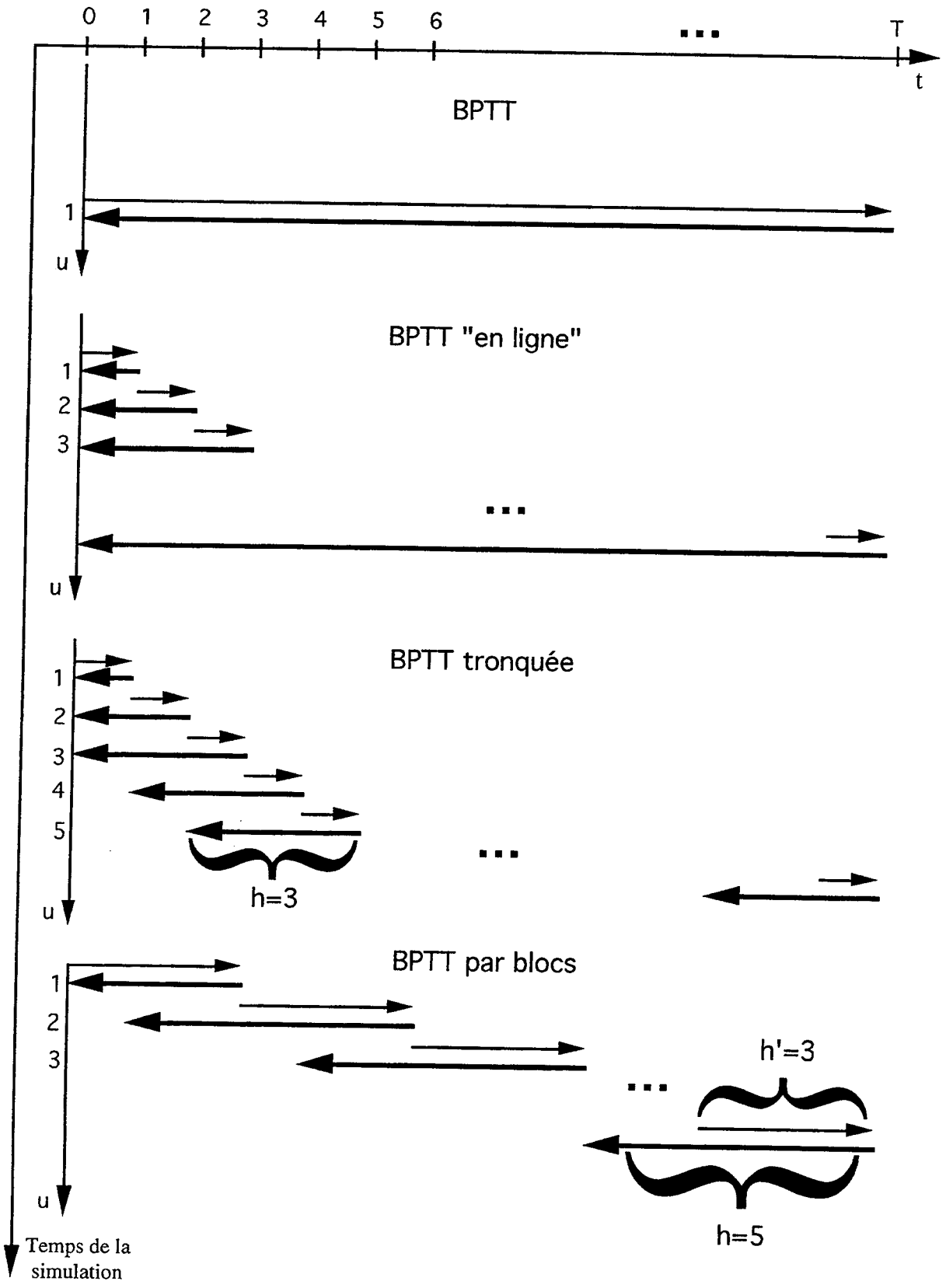

Figure II.10. Comparaison des quatre variantes de l'algorithme BPTT

Pour finir, il convient de se demander si le cadre unifié qui vient d'être présenté peut conduire à d'autres versions de l'algorithme *BPTT*. Dans le cas général, les paramètres b et p peuvent être choisis variables dans le temps. Une multitude d'algorithmes pourraient alors être développés. Nous proposons deux pistes pour de

nouveaux algorithmes. La première consiste, à partir de la rétropropagation par blocs ci-dessus décrite, à faire varier aléatoirement les paramètres h et h' autour de valeurs moyennes. Une telle introduction de l'aléatoire introduit une sorte de bruit qui peut améliorer la convergence. La deuxième consiste à modifier la profondeur de rétropropagation dans le temps en fonction des valeurs désirées provenant de l'environnement, afin d'utiliser au mieux la puissance de calcul disponible. En effet, dans un certain nombre d'applications, les valeurs désirées ne sont pas disponibles à chaque instant. Si, entre t_1 et t_2, aucune valeur désirée n'est donnée, alors il n'y a aucune rétropropagation à effectuer : on peut alors prolonger la rétropropagation précédente. Ainsi, on profite mieux du temps de calcul disponible. Cette idée peut s'étendre au cas où l'algorithme *choisit* de ne pas prendre en compte toutes les données de l'environnement (échantillonnage sélectif).

II.2.2.b Propagation avant du gradient

Nous allons décrire l'algorithme "Real-Time Recurrent Learning" (*RTRL*). Ce terme a été introduit dans [Williams & Zipser 89a] pour désigner une méthode radicalement différente de la rétropropagation dans le temps pour faire apprendre un réseau récurrent. En fait, la technique utilisée dans *RTRL* n'était pas nouvelle, d'une part parce qu'elle avait été décrite deux ans plus tôt dans [Robinson & Fallside 87 *in* Robinson 89] dans un formalisme un peu différent et d'autre part parce qu'il s'agit en fait d'une technique classique en optimisation et en filtrage adaptatif. De l'avis même de l'un des auteurs [Williams 92], l'algorithme en question aurait été redécouvert huit fois dans le cadre des réseaux ! Dans ce qui suit sera néanmoins présentée la version de [Williams & Zipser 89a], essentiellement en vertu de sa clarté.

On reprend les notations adoptées pour l'algorithme *BPTT* dans le paragraphe II.2.2.a. Comme pour ce dernier algorithme, l'objectif est de minimiser l'erreur $E^{total}(t_0,t_1)$ selon une descente de gradient :

$$\begin{cases} \Delta w_{ij} = -\alpha \dfrac{\partial E^{total}(t_0,t_1)}{\partial w_{ij}} \\ w_{ij}^{\ nouveau} \leftarrow w_{1ij}^{\ ancien} + \Delta w_{ij} \end{cases}$$

Mais la manière de dériver cette équation va être très différente.

Tout d'abord, en remplaçant $E^{total}(t_0,t_1)$ par son expression donnée dans (5), on écrit :

$$\Delta w_{ij} = \sum_{t=t_0+1}^{t_1} -\alpha \frac{\partial E(t)}{\partial w_{ij}}$$

$$\Delta w_{ij} = \sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t)$$

avec $$\Delta w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}}$$

Il suffit donc de calculer $\Delta w_{ij}(t)$. Pour ce faire, remplaçons $E(t)$ par son expression donnée dans (4) :

$$\Delta w_{ij}(t) = -\alpha \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} = -\alpha \sum_{k \in U} e_k(t).p_{ij}^k(t) \qquad (9)$$

avec $p_{ij}^k(t) = \dfrac{\partial y_k(t)}{\partial w_{ij}}$

Il faut maintenant calculer $p_{ij}^k(t)$:

$$p_{ij}^k(t+1) = \frac{\partial y_k(t+1)}{\partial w_{ij}} = f'\left(s_k(t+1)\right) \frac{\partial s_k(t+1)}{\partial w_{ij}}$$

$$p_{ij}^k(t+1) = f'\left(s_k(t+1)\right)\left[\sum_{l \in U \cup I} w_{kl} \frac{\partial x_l(t)}{\partial w_{ij}} + \delta_{ik} x_j(t)\right]$$

où δ_{ik} dénote le delta de Kronecker :

$$\begin{cases} \delta_{ik} = 1 \text{ si } i = k \\ \delta_{ik} = 0 \text{ si } i \neq k \end{cases}$$

Comme :

$$l \in U \Rightarrow \frac{\partial x_l(t)}{\partial w_{ij}} = \frac{\partial y_l(t)}{\partial w_{ij}} = p_{ij}^l(t)$$

$$l \in I \Rightarrow \frac{\partial x_l(t)}{\partial w_{ij}} = \frac{\partial x^e{}_l(t)}{\partial w_{ij}} = 0$$

On obtient finalement :

$$p_{ij}^k(t+1) = f'\left(s_k(t+1)\right)\left[\sum_{l \in U} w_{kl} p_{ij}^l(t) + \delta_{ik} x_j(t)\right] \qquad (10)$$

Les variables $\left\{p_{ij}^k\right\}$ forment donc un système dynamique dont on peut calculer les valeurs successivement dans le temps, en prenant $p_{ij}^k(0) = 0$ pour condition initiale. Ces variables s'appellent variables de sensibilité.

Contrairement à l'équation (8) de la rétropropagation dans le temps, on n'a pas de variable dont la valeur au temps t dépende de sa valeur au temps $t+1$. Les calculs

d'apprentissage s'effectuent donc en même temps que les activités du réseau sont calculées. A la fin de la séquence, les poids peuvent être adaptés immédiatement selon la formule (6b).

Cependant, le calcul des $\{p_{ij}^k\}$ est coûteux car il s'agit d'une matrice à trois dimensions : pour un réseau à n automates entièrement connectés et m entrées, il y a n^2+mn poids w_{ij} et n^3+mn^2 variables p_{ij}^k, qui seront autant de variables supplémentaires à stocker. Comme de plus l'équation (10) contient une somme de n termes, la quantité d'opérations effectuées variera avec n^4.

L'algorithme *BPTT* est donc bien plus économique. Mais tout l'intérêt de l'algorithme *RTRL* réside dans sa variante immédiate dans laquelle les paramètres du réseau sont adaptés "en ligne".

En effet, puisqu'à chaque instant t on calcule une contribution $\Delta w_{ij}(t)$ au terme Δw_{ij}, on peut directement effectuer la modification des poids selon la formule :

$$w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij}(t)$$

L'algorithme *RTRL* permet ainsi de modifier les poids au cours de la simulation sans avoir besoin de stocker explicitement les activités du réseau sur une certaine fenêtre temporelle, comme c'est le cas pour la rétropropagation dans le temps.

Une variante intéressante a été proposée dans [Zipser 89]. Afin de diminuer les coûts computationnels, on peut regrouper les unités par blocs, et considérer que chaque bloc est indépendant, c'est-à-dire que l'erreur calculée au niveau d'une unité d'un bloc ne se propage pas dans les autres blocs. Ainsi, la matrice des variables p_{ij}^k contient beaucoup de zéros, ce qui simplifie les calculs. Une contrainte dans cette stratégie de regroupement est bien sûr d'avoir par bloc au moins une unité où un signal désiré est donné, afin que chaque bloc reçoive un signal d'erreur. Il s'agit d'une approximation de la descente de gradient rigoureuse, qui est efficace dans un certain nombre de cas [Zipser 89]. Le choix des blocs est laissé à l'utilisateur du réseau, qui devra faire le compromis entre un algorithme approximatif mais rapide et un algorithme plus exact mais plus lent.

II.2.3 Analyse et difficultés de ces réseaux

II.2.3.a Coûts computationnels

Deux caractéristiques sont généralement étudiées : le stockage et la quantité de calculs, que l'on mesure par le nombre de "réels" (nombres décimaux) stockés et le nombre d'opérations arithmétiques respectivement. On évalue les *ordres de grandeur* de ces quantités, selon la notation "O", qui se définit par :

$$f(x) \text{ est } O(g(x)) \text{ si } \exists y0,\ c>0 \text{ tels que } \forall y>y0,\ f(y)<cg(y)$$

Dans un premier temps, on peut prendre un réseau entièrement connecté à M entrées et N automates, simulé pendant T instants successifs. On obtient le tableau comparatif suivant :

	BPTT	BPTT en ligne	BPTT tronquée	BPTT par blocs	RTRL
Nombre d'opérations arithmétiques	N^2T	N^2T^2	N^2Th	N^2Th/h'	$(N^4+MN^3)T$
Nombre de réels stockés	NT	NT	Nh	Nh	N^3+MN^2

Ce tableau signifie que le coût computationnel est en $O(N^2T)$ pour la *BPTT*, $O((N^4+MN^3)T)$ pour *RTRL*, etc. Les performances opposées des algorithmes *BPTT* et *RTRL* apparaissent clairement : *BPTT* est deux ordres de grandeur moins coûteux en calculs, mais s'il doit être utilisé en ligne sans tronquer la rétropropagation, le temps de calcul devient proportionnel au carré de la taille de la séquence, ce qui le rend complètement inadapté à ce type d'utilisation.

Sur le plan du stockage, *BPTT* a besoin d'un stockage temporel (ce qui le rend inadapté aux longues séquences) alors que *RTRL* a besoin d'un stockage spatial (ce qui le rend inadapté aux grands réseaux).

Dans un second temps, si l'on considère une structure plus simple, où il y a seulement $O(N)$ poids (structures en lignes, en anneau), alors on obtient le tableau suivant :

	BPTT	BPTT en ligne	BPTT tronquée	BPTT par blocs	RTRL
Nombre d'opérations arithmétiques	NT	NT^2	NTh	NTh/h'	N^2T
Nombre de réels stockés	NT	NT	Nh	Nh	N^2

Le handicap de *RTRL* est donc moins grand dans ce cas.

Il faut cependant noter que ce type de comparaison manque de précision [Logar 93]. En effet, les comparaisons ci-dessus n'ont de sens que quand N est grand. Or la plupart des applications ne vérifient pas cette condition.

Dans [Logar 93], les coûts computationnels sont estimés plus finement en tenant compte du coût relatif des différentes opérations (multiplications, division, etc.). Pour l'algorithme *BPTT*, on trouve comme coût de calcul :

$$7\,N^2T + 71\,NT + 2\,N^2$$

et pour l'algorithme *RTRL* :

$$(N^4 + 5\,N^3 + 6\,N^2 + 30\,N + 2N^3M + 5\,N^2M + 6\,NM)\,T$$

Enfin, remarquons que l'algorithme *RTRL* se parallélise bien, dans la mesure où l'expression (10) constitue un système de N^2 équations pouvant se calculer en parallèle. Le temps de calcul peut ainsi être réduit de $O(N^2)$ dans le cas d'une structure entièrement connectée.

II.2.3.b Non-localités spatiales

Un autre axe de comparaison entre *RTRL* et *BPTT* concerne la "non-localité".

L'algorithme *RTRL* est souvent qualifié de non local, dans la mesure où chaque poids doit avoir accès non seulement à l'ensemble de la matrice des poids (équation (10)) mais aussi au vecteur d'erreur $e_k(t)$ (équation (9))[Williams & Zipser 89a]. Il semble donc y avoir deux non-localités. Ce n'est pas le cas pour l'algorithme *BPTT*.

Dans [Schmidhuber 90], on dénote "localité spatiale", par opposition à "localité temporelle", la propriété selon laquelle la complexité des calculs par connexion dans une unité de temps est bornée (est $O(1)$), quelle que soit la taille du réseau. Cette formulation de la localité se ramène totalement à une comparaison computationnelle

(et non structurelle) du type de celle proposée dans les deux tableaux du chapitre précédent (dans lesquels on diviserait les valeurs par N^2T).

Au delà de la disparité des définitions, le concept de non-localité apparaît en contradiction avec le fondement même des réseaux de neurones; en effet, selon [Hecht-Nielsen 90] (traduit par nos soins), dans un réseau de neurones, tout traitement effectué par une unité (en relaxation ou en apprentissage) "doit dépendre seulement des valeurs des signaux entrant dans l'unité via les connexions entrantes et des valeurs qui y sont stockées". L'algorithme *RTRL* ne serait-il pas un réseau de neurones ? ("réseau de neurones" étant pris dans son acceptation informatique.)

Le paradoxe provient tout simplement de la "tradition" connexionniste qui consiste à ne pas expliciter le réseau d'apprentissage, mais uniquement le réseau de propagation (voir chapitre I.3.5). Si un poids, à un endroit du réseau, doit avoir accès à un paramètre ailleurs dans le réseau, alors cela signifie, si l'on veut rester dans un cadre connexionniste, qu'il y a une connexion entre les deux points du réseau. Nous introduisons alors deux définitions :

- un apprentissage est local si le graphe du réseau fonctionnant en phase d'apprentissage est le même que le graphe du réseau en relaxation ;
- un apprentissage est strictement local si le graphe orienté du réseau fonctionnant en phase d'apprentissage est le même que le graphe orienté du réseau en relaxation.

Notons que selon cette définition, l'algorithme de rétropropagation du gradient dans un réseau à couches est local, mais pas strictement local ; il en va de même pour l'algorithme *BPTT*.

La question est donc d'exprimer clairement l'algorithme *RTRL* sous forme d'un réseau afin de pouvoir observer sa connectivité, et d'identifier quels traitements rendent l'apprentissage non local.

On remarque que la formule (10) du paragraphe II.2.2.b décrit le comportement d'un réseau de N unités ayant, pour (i,j) fixé, les activités $\left\{p_{ij}^1, p_{ij}^2, \ldots, p_{ij}^N\right\}$ et une entrée $x_j(t)$. Il y a autant d'équations (10) que de poids adaptables, donc autant de réseaux supplémentaires que de poids adaptables, soit $(M+N)M$ dans le cas entièrement connecté.

La formule (9) montre que tout poids adaptable reçoit une information des unités où une erreur $e_k(t)$ peut être calculée, ainsi que certains signaux p. On peut aussi considérer, et c'est la solution que nous adoptons, que des unités supplémentaires

calculent le produit $v_{ij}^k(t) = e_k(t)p_{ij}^k(t)$, dont les sorties sont envoyées au poids correspondant.

La figure II.11 illustre le réseau qui apprend, pour une structure particulière donnée comportant cinq automates partiellement connectés dont une entrée et deux sorties (le poids sur *x0* est supposé fixé à *1*). La complexité du réseau qui apprend explique certainement en partie pourquoi il n'est jamais représenté. Le réseau initial est répliqué autant de fois que de paramètres adaptables, chaque réseau recevant une activité du réseau initial (flèches rectilignes en diagonale) et les signaux $f'(s_i(t))$ (flèches en pointillés fins). Autant d'unités que de sorties désirées (on suppose $T(t)$ introduit dans la formule (3) constant) sont ajoutées pour calculer $e_k(t)$. Des unités sont aussi ajoutées pour calculer les produits $v_{ij}^k(t) = e_k(t)p_{ij}^k(t)$; les sorties de ces unités sont connectées vers les automates où sont stockés les poids (flèches en gras), qui doivent ensuite être transmis à tous les automates correspondants des réseaux supplémentaires (flèches en pointillés fins à nouveau).

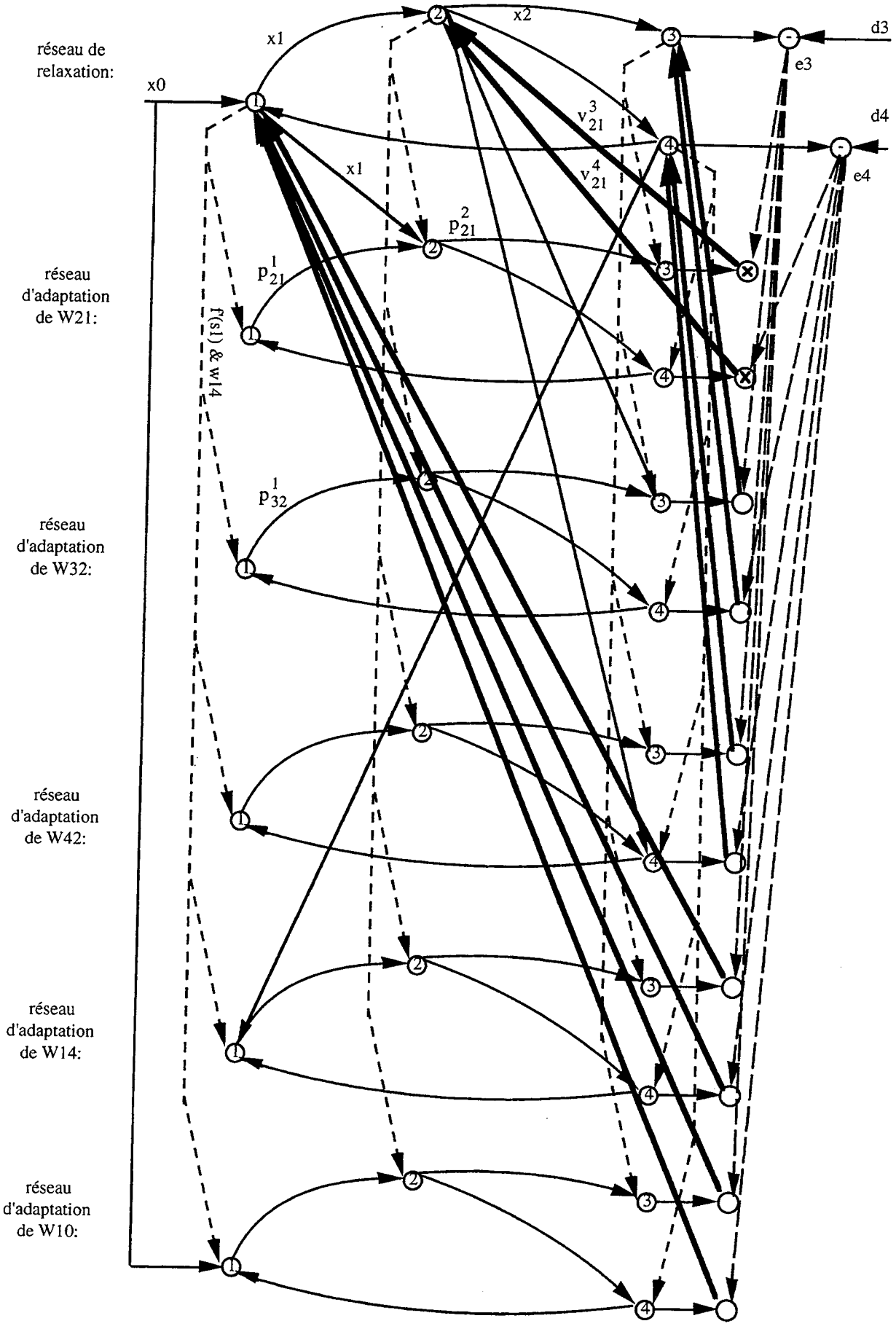

Figure II.11. Représentation complète de l'algorithme RTRL

Une telle représentation n'est pas unique et dépend essentiellement du grain de description adopté ; par exemple, l'ajout des automates qui calculent les valeurs $v_{ij}^k(t)$ n'était pas obligatoire.

En particulier, une description en macro-unités de l'algorithme *RTRL* est intéressante à développer. Il s'agit, à partir du schéma II.11, de former des colonnes, chaque colonne regroupant les automates ayant la même position dans le réseau initial et les réseaux ajoutés du schéma II.11.

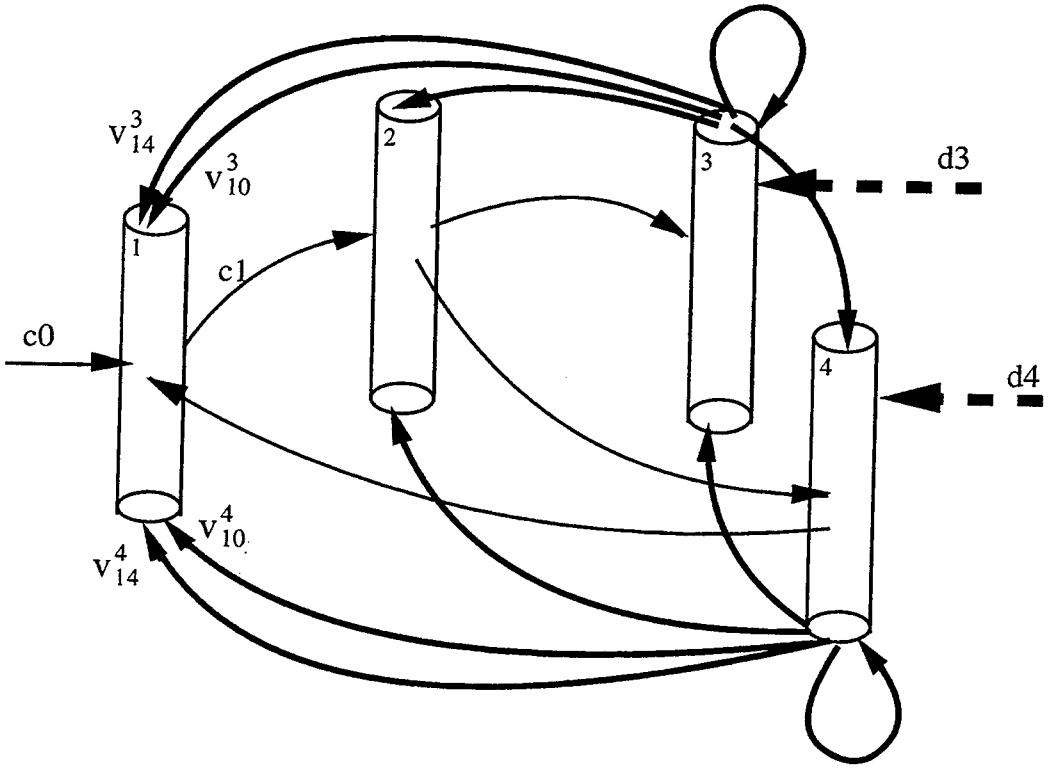

Figure II.12. Réseau de colonnes pour l'algorithme RTRL

La figure II.12 explicite ce réseau à colonnes. Les automates de ce nouveau réseau échangent d'une part les signaux $v_{ij}^k(t)$ (flèches en gras) et d'autre part des signaux multiples, c'est-à-dire des vecteurs, que l'on dénote c_i, i désignant l'automate d'où part le signal ; dans l'exemple ci-dessus, les signaux c_i sont :

$$c_i = (x_i, p_{21}^i, p_{32}^i, p_{42}^i, p_{14}^i, p_{10}^i)$$

On constate alors que :

- les calculs des $p_{ij}^k(t)$ utilisent le même graphe orienté que les calculs des $x_i(t)$;
- l'apprentissage est non local à cause des variables $v_{ij}^k(t)$ c'est-à-dire du fait des calculs de $w_{ij}.(t)$ selon la formule (9).

Selon le réseau en colonnes, il n'y a donc qu'une seule source de non-localité, contrairement au réseau représenté sur la figure II.11. Parmi les deux non-localités classiquement évoquées concernant l'algorithme *RTRL* [Williams & Zipser 89a], l'une d'elle est finalement *réductible* si l'on regroupe les traitements en colonnes. Il ne semble par contre pas possible de réaliser un tel groupement qui supprimerait la non-localité engendrée par l'équation (9).

Remarquons tout de même que dans chaque macro-unité de la figure II.12, il y autant de variables p que de paramètres adaptables dans *tout* le réseau ; en conséquence, la *construction* du réseau doit tenir compte, en chaque unité, du réseau dans sa globalité.

Le cas particulier de l'algorithme neuronal *RTRL* que nous venons d'étudier met en évidence une *méthodologie générale* d'analyse architecturale des réseaux. Etant données les équations décrivant l'évolution des différentes variables d'un système dynamique, il apparaît qu'il n'y a pas un unique réseau pouvant réaliser les traitement décrits par les équations, selon le groupement que l'on effectue. Se posent alors les questions suivantes :

- comment trouver toutes les structures de réseau correspondant à un ensemble d'équations ?
- quels peuvent être les critères guidant le choix de l'une ou l'autre des solutions ?

Traiter ces deux questions en profondeur nous éloignerait du sujet qui nous préoccupe, et nous nous limiterons donc à ébaucher quelques éléments de réponse.

Détermination de toutes les structures possibles : on remarque qu'on peut dériver une structure de réseau d'une autre par *regroupement* des unités dans une unité plus grande (voir figure II.13). Si l'on arrive à obtenir une structure *unique* la plus développée possible (la moins regroupée), alors toutes les structures pourront en être dérivées, par combinaison systématique de tous les regroupements possibles. Cette structure unique s'obtient tout naturellement d'après les équations, en considérant que toute fonction est un automate ayant autant d'entrées que de dimensions de l'espace d'entrée et autant de sorties que de dimensions de l'espace de sortie, tout opérateur arithmétique est un automate à deux entrées et une sortie, toute somme a autant d'entrées que d'éléments à sommer et une sortie, etc. En effet, développer davantage le réseau est possible mais ne relève plus des équations elles-mêmes. La figure II.13 représente différents réseaux qui peuvent implanter une équation donnée, en commençant par le réseau le plus développé.

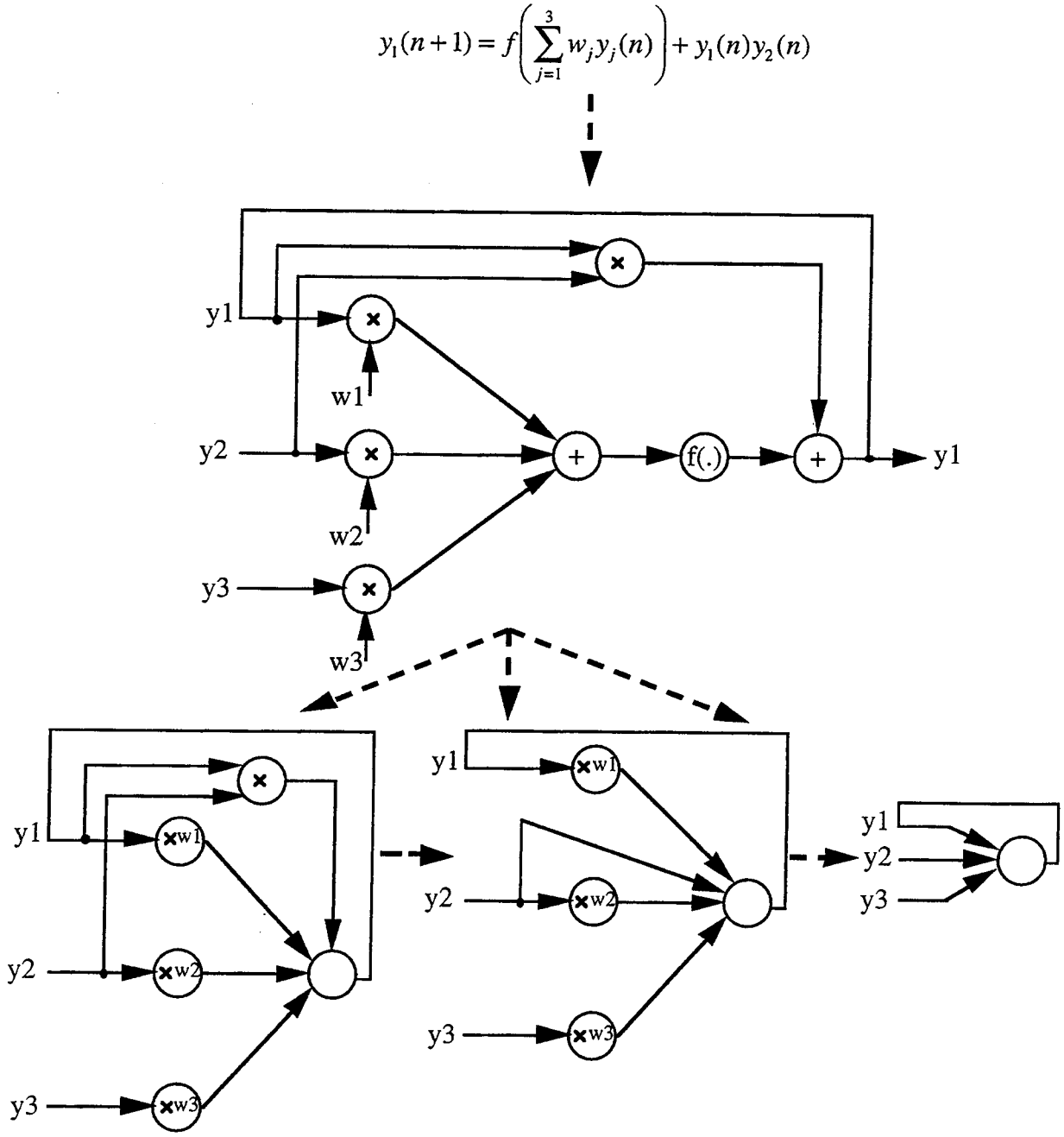

Figure II.13. Dérivations de quelques réseaux correspondants à une équation

Critère de regroupement : dans le cas d'une réalisation matérielle d'un réseau, ce sont les contraintes liées au support qui doivent guider le choix d'un regroupement en unités de traitement. Par contre, dans le cas précédent (*RTRL*) on a cherché à obtenir un graphe donné, celui du sous-réseau de relaxation. Dans la plupart des cas, on cherchera un réseau tel que toutes les unités soient de même type, ou d'un nombre limité de types différents.

II.2.3.c algorithmes "temps réels" ?

Les réseaux récurrents ci-dessus décrits ont des comportements temporels et peuvent se modifier en fonction des données issues de l'environnement.

La manière dont s'effectuent ces modifications fait l'objet de nombreuses classifications dichotomiques des algorithmes en question : temps réel/temps différé [Williams & Zipser 89ab], adaptatifs/non adaptatifs [Nerrand *et al.* 93], "on-line"/"off-line" [Williams & Peng 90][Piché 94], local dans le temps ou non [Schmidhuber 90].

Toutes ces notions se recoupent mais ne sont pas équivalentes. Nous proposons donc de les définir avec précision.

Algorithmes "en ligne"

Il convient tout d'abord de distinguer les algorithmes qui autorisent une modification des paramètres *pendant* l'exécution du réseau, par opposition aux algorithmes où la modification des poids ne peut s'effectuer qu'à la *fin* de la séquence générée par le réseau. Dans le premier cas, il s'agit d'algorithmes "en ligne" ("on-line") [Williams & Peng 90][Piché 94], dans le second d'algorithmes "off-line", par epochs ("epochwise" [Piché 94]).

Applications temps réel

Par ailleurs, l'adjectif "temps réel" est souvent utilisé en même temps que "on-line" à propos de tel ou tel algorithme. Cependant, l'aspect temps réel n'est en fait pas relatif à un algorithme en tant que tel mais à une *application*, c'est-à-dire l'utilisation d'un algorithme pour un système physique donné, que ce soit pour le visualiser, l'identifier, le contrôler, etc. Un algorithme n'est donc pas temps réel, mais fonctionne en temps réel sur un certain système physique. Une application est dite temps réel si les calculs s'effectuent au moins aussi vite que le système physique concerné.

Cela amène à échantillonner le système physique, c'est-à-dire à saisir les données de l'environnement tous les τ unités de temps. Une application sera temps réel si elle effectue ses calculs relatifs à l'échantillon courant en un temps $\theta(t)$ inférieur à τ ("temps réel" ne veut donc pas dire "rapide", puisque τ peut valoir 15 minutes). Il convient de souligner que tout algorithme d'apprentissage "en ligne" peut fonctionner en temps réel : pour une séquence finie (durée de vie du système) la durée de calcul maximale $\theta = \max\{\theta(t)\}$ d'une modification des paramètres est nécessairement bornée, et l'algorithme peut donc fonctionner en temps réel pour tout

système tel que $\tau > \theta$. Ainsi, même l'algorithme de "rétropropagation dans le temps en ligne" (voir paragraphe II.2.2.a) peut fonctionner en temps réel.

Notons qu'un algorithme "en ligne" peut ne pas fonctionner en temps réel, si il modifie ses paramètres en cours d'exécution mais avec un *retard* par rapport au système réel ($\theta > \tau$).

Optimisation des calculs

Il apparaît que si l'on utilise un algorithme tel la "rétropropagation dans le temps en ligne", pour laquelle $\theta(t)$ est de l'ordre de t, alors les calculs sont concentrés à la fin de la séquence, et la puissance de calcul sera en moyenne sous utilisée. Autrement dit, pour une puissance de calcul donnée, la constante de temps τ à partir de laquelle on pourra utiliser cet algorithme en temps réel va augmenter avec la durée de la séquence. Une contrainte pratique essentielle est donc de travailler avec des algorithmes tels que l'ordre de grandeur de θ soit borné et ne dépende pas de t, soit θ=O(1). Ceci est exactement la définition donnée dans [Schmidhuber 91] de la "localité dans le temps". Cette définition est proche de celle d'un système adaptatif donnée dans [Nerrand *et al.* 93] : un système adaptatif est un système entraîné en continu (équivalent de "en ligne") et utilisé sur un nombre infini d'échantillons. En effet, si la séquence est infinie, l'ordre de grandeur de θ doit être borné.

De manière plus précise, on aura tendance à rechercher des algorithmes dont la répartition des calculs est uniforme ($\theta(t) = constante$) afin d'utiliser toute la puissance de calcul à chaque instant, dans le cas où l'échantillonnage est lui aussi uniforme.

Enfin, pour une application temps réel rapide, on s'intéressera à réduire le plus possible θ, non seulement en ordre de grandeur mais aussi en valeur. Ainsi, pour la rétropropagation dans le temps par blocs (paragraphe II.2.2.a), on cherchera à minimiser h et h' tout en gardant une bonne approximation du gradient. Dans le cas d'une implémentation parallèle, l'algorithme *RTRL* est le plus rapide, puisque $\theta(t)$ est sensiblement égal au temps de calcul des activités du réseau.

II.2.3.d problèmes de convergence

Les "problèmes de convergence" désignent les situations où la dynamique des poids du réseau ne conduit pas à une solution satisfaisante. Ces situations sont nombreuses dans les réseaux récurrents. Nous passons d'abord en revue quatre des problèmes qui existent aussi dans les réseaux à couches, pour ensuite aborder des problèmes spécifiques des réseaux récurrents.

Un grand nombre de résultats théoriques sur les réseaux concernent l'*existence* d'un réseau pouvant réaliser telle fonction. Ainsi, pour les réseaux récurrents, on montre que toute trajectoire peut être approximée avec une précision arbitraire par un réseau récurrent entièrement connecté [Li 92]. Ce type de résultats ne prédit pas si l'on peut trouver les paramètres du réseau en question par apprentissage. Le mode d'apprentissage souvent choisi, qui consiste à utiliser le gradient d'un critère d'erreur, garantit uniquement que sous certaines conditions, l'on peut aboutir à un minimum (local ou global) de la surface d'erreur.

La première source de problème réside donc dans le fait que le minimum atteint peut donc correspondre à une erreur élevée (minimum local). Dans de nombreuses utilisations des réseaux à couches, on se satisfait de cette situation, estimant que les minima locaux sont rares ou correspondent à une erreur assez faible. Cependant, d'après [Piché 94], les problèmes de minima locaux à valeur élevée seraient plus fréquents dans les réseaux récurrents que dans les réseaux à couches ; les expériences relatées au chapitre II.5 confirment cette observation. Certains résultats théoriques, établissent des conditions dans lesquelles un réseau récurrent n'a pas de minimum local [Bianchini *et al.* 94].

Deuxièmement, la convergence vers un minimum n'est assurée que pour un pas infiniment petit, ce qui, dans la pratique, n'est pas réalisable. En fait, au contraire, le concepteur d'algorithmes neuronaux maximise le pas, afin de rendre l'algorithme plus rapide. On n'est alors plus du tout dans les conditions de validité de l'algorithme.

Troisièmement, la surface d'erreur comporte un certain nombre de *pièges* [Le Cun 87] en particulier des plateaux : la surface d'erreur est presque horizontale et les paramètres n'évoluent donc pratiquement pas. Empiriquement, l'effet est le même que dans le cas d'un minimum local : l'apprentissage devient de plus en plus lent.

Quatrièmement, la convergence vers un minimum se fonde sur un formalisme *continu* alors qu'au niveau de l'ordinateur, seules des valeurs *discrètes* sont manipulées. La convergence vers un minimum n'est donc pas garantie, sur un ordinateur actuel. Par exemple, au niveau d'un plateau, le gradient est très faible et peut s'annuler, alors qu'il ne s'agit pas d'un minimum local. Une erreur d'arrondi peut aussi faire remonter l'erreur.

Nous abordons maintenant les problèmes plus spécifiques des réseaux récurrents ; tous se manifestent lorsque le réseau manipule de longues séquences.

Tout d'abord, un algorithme qui adapte les paramètres en ligne obtenu à partir d'un algorithme qui adapte les paramètres en fin de séquence ne suit pas la direction exacte du gradient [Williams & Zipser 89a]. En effet, ce type d'algorithme minimise une erreur qui est fonction des poids du réseau : ces poids sont implicitement supposés fixes. Or dans les versions en ligne, ces poids sont modifiés au cours du déroulement de la séquence. Ce n'est donc plus le gradient de l'erreur calculée avec les poids initiaux fixes qui est évalué.

D'après [Williams & Zipser 89a] à propos de l'algorithme *RTRL*, on évite ce problème si la variation des poids est bien plus lente que celle des activités, et il suffit donc de choisir un pas d'apprentissage suffisamment petit. Or, comme on le fait remarquer dans [Catfolis 93], même avec un pas d'apprentissage très faible, les poids finissent toujours par varier de manière importante, pour une séquence suffisamment longue, et l'algorithme devient faux. Autrement dit, l'algorithme RTRL n'est valide que si les poids évoluent peu, ce qui restreint fortement son utilisation. Bien sûr, on peut présenter plusieurs fois la même séquence avec un pas suffisamment petit pour que sur chaque présentation les poids évoluent peu et faire ainsi fortement varier les poids sans trop s'éloigner de la direction du gradient, mais l'algorithme *RTRL* n'est alors plus utilisé en continu, et l'algorithme *BPTT* est dans ce cas préférable.

Un dernier problème relatif aux longues séquences est le problème de "l'explosion du gradient". Dans un certain nombre de situations, la dérivée de l'erreur instantanée $E(t)$ selon un poids du réseau peut devenir très importante au fur et à mesure que la simulation se prolonge. Cette explosion du gradient pose en particulier des problèmes quand on dépasse les capacités de codage de l'ordinateur. Nous avons pu relever deux situations où les gradients explosent.

La première situation se réfère au problème des bifurcations. Certains systèmes dynamiques sont tels qu'en certains points de l'espace des paramètres, un variation infinitésimale d'un paramètre peut aboutir à un comportement totalement différent à l'infini [Pearlmutter 90][Doya 94][Crucianu 94]. Pour certains algorithmes, comme celui décrit dans [Pineda 87], qui sont fondés sur la dérivée de l'état à l'infini par rapport aux poids, un problème théorique évident se pose : cette dérivée s'avère ne pas être définie en certains points [Pearlmutter 90][Doya 94]. Pour les apprentissages de séquences *finies*, l'erreur est dérivable partout (c'est une somme finie de termes dérivables), mais le gradient peut croître rapidement [Doya 94][Crucianu 94].

La deuxième situation concerne les systèmes oscillatoires, pour lesquels le problème ne se rencontre pas uniquement pour des points de bifurcation particuliers.

Si on entraîne un réseau à osciller, une très faible modification d'un poids va entraîner un décalage de phase et donc fortement modifier l'erreur à un instant t grand, dans le cas habituel de l'erreur quadratique.

Supposons ainsi qu'un réseau soit entraîné à produire un signal $y(t) = sin(\omega t)$. Supposons que ω soit explicitement codé dans l'un des paramètres w du réseau, ce qui n'est pas le cas, mais nous fournit une première idée. Alors on aurait :

$$\frac{\partial y(t)}{\partial w} = t\cos(wt)$$

Donc le gradient oscillerait avec une amplitude de plus en plus élevée. Nous avons vérifié ce comportement en entraînant avec *RTRL* un réseau de 12 automates entièrement connectés sur une trajectoire circulaire centrée en *0,5* et de rayon *0,4*, tâche qui a été étudiée dans [Pearlmutter 89] (les unités sont à fonction de transfert sigmoïde entre *0* et *1*, les poids initialisés entre *-0,5* et *0,5*, le réseau entraîné sur *19* points - un cercle entier comportant *16* points- pendant *10000* présentations). La dérivée de l'erreur instantanée par rapport à un poids arbitraire du réseau est représentée sur la figure II.14. On trouve effectivement une croissance sans fin du gradient : les maxima de cette courbe croissent linéairement.

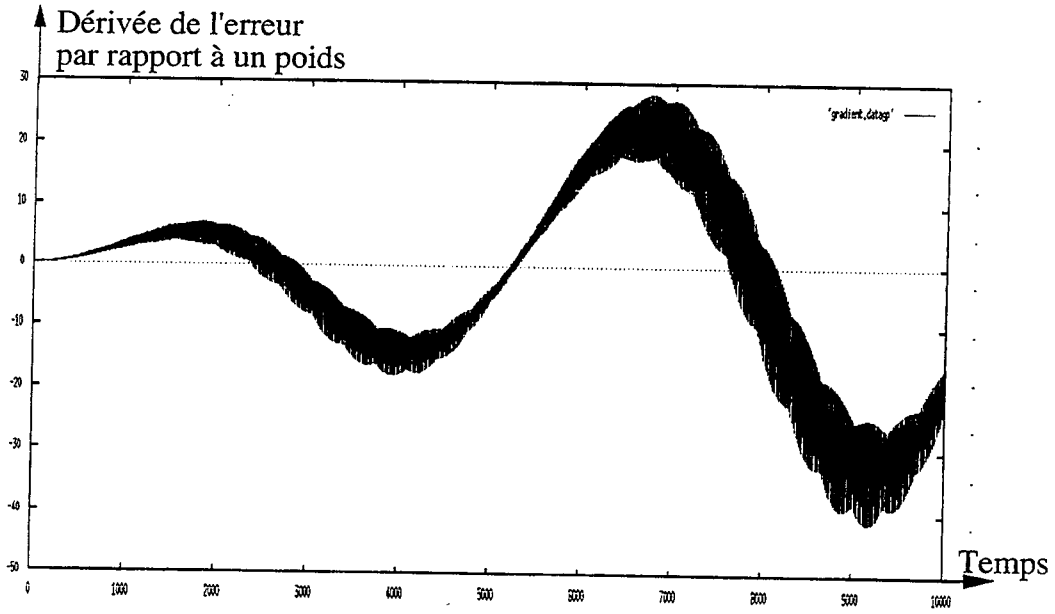

Figure II.14. Explosion du gradient

II.2.4 Variantes et améliorations existantes

II.2.4.a Forçage du réseau

Une variante est très souvent associée à l'algorithme *RTRL*, que l'on nomme "teacher forcing", *forçage* en français. Nous en décrivons d'abord les aspects algorithmiques, pour ensuite aborder les aspects cognitifs que sous-tendent certaines formules exposées.

Le forçage consiste à obliger le réseau à suivre la trajectoire désirée, même si naturellement il s'en écarte. Le comportement désiré $d_i(t)$ qui provient de l'environnement sert non seulement à calculer une erreur, selon la formule (3) du paragraphe II.2.2.a, mais est aussi utilisé pour modifier les activités des unités où un comportement désiré est fourni, selon la formule :

$$x_i(t) = \begin{cases} x^e{}_i(t) \text{ si } i \in I \\ d_i(t) \text{ si } i \in T(t) \\ y_i(t) \text{ si } i \in U - T(t) \end{cases}$$

Cette expression remplace l'expression (1) du paragraphe II.2.2.a. De plus, les termes $p_{ij}^l(t)$ sont mis à zéro après la modification des poids selon (9) (voir paragraphe II.2.2.b), ce qui revient à remplacer l'équation (10) par :

$$p_{ij}^k(t+1) = f'\left(s_k(t+1)\right)\left[\sum_{l \in U-T(t)} w_{kl} p_{ij}^l(t) + \delta_{ik} x_j(t)\right]$$

Cette variante évite que le comportement du réseau ne s'éloigne trop du comportement désiré, et est surtout intéressante pour les longues séquences [Crucianu 94]. Dans certains cas, l'algorithme ne converge vers une bonne solution que si cette variante est utilisée [Williams & Zipser 89ab]. En particulier, le forçage permet d'éviter l'explosion du gradient [Doya 94]. Nous l'avons vérifié en répliquant l'expérience décrite à la fin du paragraphe II.2.3.d (voir aussi la figure II.14) : avec le forçage, le gradient reste borné.

En fait, si le *terme* "teacher forcing" est introduit dans [Williams & Zipser 89a], cette technique est empruntée au domaine du contrôle, et se dénote alors "series parallel model" [Narendra & Parthasarathy 90] ainsi qu'au domaine du filtrage adaptatif et se dénote alors "equation error" [Nerrand *et al.* 93]. Dans [Nerrand *et al.* 93], les versions forcées et non forcées de *RTRL* sont replacées dans le cadre général du filtrage adaptatif, ce qui permet d'envisager plusieurs autres variantes.

Il faut préciser que le forçage du réseau peut être utilisé quel que soit l'algorithme d'apprentissage du réseau récurrent (algorithmes *RTRL* et *BPTT*, apprentissage du point fixe [Pineda 88 *in* Crucianu 94]), à condition bien sûr que la sortie soit rebouclée sur elle-même ou sur les automates de la couche cachée.

Une autre alternative pour forcer le réseau a été proposée dans [Toomarian & Barhen 91,92], que l'on peut nommer "forçage par l'erreur". Au lieu d'affecter les valeurs désirées aux activités du réseau, on ajoute une entrée supplémentaire au réseau dont la valeur est liée à l'erreur commise par le réseau. La formulation originale est en formalisme continu ; en formalisme discret, on obtient, à la place du premier terme de l'équation (2) :

$$s_i(t+1) = \sum_{j \in U \cup I} w_{ij} x_j(t) + \lambda \left(\left[d_i(t) \right]^{1-\beta} \left[d_i(t) - y_i(t) \right]^{\beta} \right)$$

avec λ et β deux nombre positifs.

De plus, les auteurs proposent de diminuer λ au cours du temps (λ est modifié uniquement entre deux présentations de la séquence entière), selon la loi :

$$\lambda(\tau) = 1 - e^{-E^{total}(\tau)}$$

avec τ le numéro de la présentation de la séquence entière.

Expérimentalement, les auteurs obtiennent de bons résultats, mais difficilement exploitables car d'une part le forçage est combiné avec une modification de l'algorithme même d'adaptation des poids et d'autre part le nouveau type de forçage n'est pas comparé avec l'algorithme de forçage classique.

De manière générale, l'efficacité du forçage n'est pas garantie du point de vue théorique [Jodouin 93]. Au niveau expérimental, les travaux décrits dans [Williams & Zipser 89a] reviennent à constater que sur certains problèmes (apprentissage d'oscillations), le forçage est indispensable (ce qui est confirmé dans [Tsung 90]) sans systématiquement comparer les versions avec et sans forçage pour *tous* les problèmes, comme on le fait remarquer dans [Hecht-Nielsen 90]. De même, aucune étude expérimentale n'a été réalisée à notre connaissance pour comparer les deux algorithmes de forçage ci-dessus décrits.

Le principal problème du forçage est le suivant : *l'erreur mesurée pendant l'exécution forcée n'est pas identique à l'erreur mesurée quand le réseau s'exécute librement* : elle est en l'occurrence plus faible. Ainsi, une erreur très faible en cours d'apprentissage peut finalement donner une mauvaise solution (voir [Tsung 90]).

Enfin, nous souhaitons maintenant aborder les aspects cognitifs de l'algorithme de forçage. En effet, dans [Toomarian & Barhen 91, 92], sans qu'il soit question de modélisation, on insiste particulièrement sur l'inspiration provenant de données humaines dans la conception de l'algorithme de forçage. Selon [Toomarian & Barhen 92] (traduit en français pour le présent document) :

> *"supposez qu'un père veuille apprendre à son enfant à faire du vélo. Bien sûr, le père ne va pas rester à la maison, laisser son enfant faire du vélo, et, de temps en temps, lui dire dans quelle mesure les performances sont bonnes ou mauvaises (comme cela se passe en apprentissage supervisé classique). La meilleure méthode serait pour le père d'accompagner son enfant pendant l'apprentissage"*

Comparer l'algorithme de forçage et l'apprentissage moteur nous apparaît très intéressant. Cependant, les termes précis dans lesquels l'analogie est effectuée sont à discuter.

D'une part, l'apprentissage supervisé classique ne correspond pas à un jugement donné "de temps en temps" mais au contraire le signal désiré $d_i(t)$ peut être présent à tout instant.

D'autre part, la position imposée par le père n'est pas nécessairement la position désirée, car il y a une certaine souplesse dans le forçage. L'analogie ne vaut donc que pour le forçage par l'erreur, et non pour la version originale du forçage [Williams & Zipser 89].

Ensuite, l'attitude du parent n'est pas uniforme avec le temps.

Ce dernier point a amené les auteurs à proposer l'analogie suivante :

> *"Quand un père apprend à son enfant à faire du vélo, dans les premières étapes il garde les mains sur le vélo pour accompagner son enfant. Cependant, dès que l'enfant fait preuve d'une certaine habileté pour contrôler ses mouvements, le père retire ses mains de plus en plus souvent, afin de laisser l'enfant avancer tout seul."*

Cette analogie a amené les auteurs à proposer la formule $\lambda(\tau) = 1 - e^{-E^{total}(\tau)}$ ci-dessus. Cependant, réaliser algorithmiquement ce qui est cité ci-dessus à propos de l'apprentissage humain conduirait à faire varier la *fréquence* du forçage, comme il est suggéré dans [Tsung 90], et non *l'intensité* du forçage (paramètre λ).

temporelles, les réseaux récurrents ont justement été introduits afin d'éviter de fixer une fenêtre temporelle ; en effet, il est possible d'utiliser des réseaux à couches pour traiter des données temporelles, en "spatialisant le temps", c'est-à-dire en donnant en entrée une fenêtre temporelle du signal [Waibel *et al.* 89].

La réponse classique est de dire qu'il faut choisir la taille temporelle en fonction de ce qu'on connaît du système à identifier ; cette connaissance a priori facilite l'apprentissage. Dans le cas des réseaux récurrents, cette fenêtre temporelle peut être variable, contrairement à la fenêtre d'un réseau à couches. Dans cet esprit, des idées sont certainement à explorer, par exemple tronquer l'historique quand le gradient devient trop grand, tronquer plus ou moins selon les paramètres, tronquer aléatoirement, etc.

II.2.4.c Architectures évolutives

Par opposition aux algorithmes neuronaux classiques, où l'apprentissage réside dans la *modification* des poids des connexions entre les unités, certains algorithmes peuvent apprendre par ajout et/ou suppression d'unités et/ou de connexions, et possèdent donc une architecture évolutive.

Deux grandes classes d'algorithmes sont utilisées [Alpaydin 91] : les algorithmes constructifs (ou incrémentaux), dont la structure est minimale au départ pour ensuite grossir selon les besoins, et les algorithmes destructifs (ou d'élagage) qui, après apprentissage, suppriment les unités et connexions inutiles.

Ces architectures sont beaucoup étudiées pour les réseaux à couches (voir [Alpaydin 91], [Fiesler 94] pour des revues bibliographiques). Elle offrent les avantages suivants :

- optimisation du temps de calcul, puisque une structure minimale, ou presque minimale est obtenue ;
- meilleure généralisation, pour la même raison ;
- facilitation de l'apprentissage dans les réseaux constructifs : au départ, le réseau est de petite taille, donc apprend plus facilement, et est réutilisé dans le réseau plus grand.

Très peu d'études ont été consacrées aux architectures récurrentes évolutives.

Le premier algorithme proposé, de type constructif est la "Recurrent Cascade Correlation" [Fahlman 91], transformation pour les réseaux récurrents de l'algorithme

Pour finir, précisons que la réelle interaction parent-enfant dans l'exemple de l'apprentissage du vélo est certainement bien plus sophistiquée qu'un simple forçage élémentaire. A ce titre, le terme de *"teacher* forcing" est évidemment abusif, ce qui explique que nous l'ayons seulement traduit par "forçage". Nous reviendrons sur le forçage plus loin dans ce document, aux paragraphes II.4.2 et III.2.2.b.

II.2.4.b Elagage de l'historique

Même si, pour une certaine clarté mathématique, les algorithmes d'apprentissage sont exposés sous la forme de la minimisation d'un critère d'erreur calculé sur une séquence entière, voire infinie, un certain nombre de raisons conduisent au contraire à "découper" cette séquence.

En plus des aspects techniques que posent les longues séquences (voir paragraphe II.2.3.d), un raisonnement intuitif invite aussi à "découper" la séquence : si le gradient est calculé sur la séquence entière, il risque alors de contenir des termes calculés alors que les poids étaient totalement différents, termes qui peuvent en fait faire augmenter l'erreur estimée avec les poids actuels.

Il semble donc que dans un apprentissage en ligne, il y ait une fenêtre temporelle à considérer, partant du présent vers le passé. Les versions tronquées et par blocs de l'algorithme *BPTT* (voir paragraphe II.2.2.a) introduisent explicitement cette notion de fenêtre, et peuvent d'ailleurs répondre au problème de l'explosion des gradients [Doya 94][Crucianu 94]. De même, en ce qui concerne l'algorithme *RTRL*, une variante a été proposée dans laquelle les signaux p_{ij}^k sont initialisés à zéro de temps en temps [Catfolis 93] : cette modification rend l'algorithme *RTRL* plus efficace.

On pourrait penser que la version de [Gherrity 89] introduit aussi une fenêtre temporelle pouvant aider l'apprentissage, comme il est écrit dans [Williams & Peng 90], puisque elle utilise comme fonction d'erreur instantanée (adaptée ici au formalisme discret) :

$$E(t) = \frac{1}{2}\sum_{\tau=0}^{t} \lambda^{(t-\tau)} \sum_{i \in U} e_i(\tau)^2$$

mais avec une telle expression de $E(t)$, le critère global qui est minimisé, à savoir $E^{total}(0,T)$, prend en fait plus en compte le début de la séquence que la fin.

L'introduction d'une fenêtre temporelle, qui revient à ignorer les longues dépendances, pose un nouveau problème : comment choisir la taille de cette fenêtre ? Ce problème est d'autant plus aiguë que dans le domaine du traitement de séquences

de "Cascade Correlation" [Fahlman & Lebiere 91] destiné aux réseaux à couches. Afin de pouvoir réaliser cette transformation, on est obligé de se restreindre à une structure limitée, auto-récurrente en cascade (voir figure II.15), proche de celle étudiée dans [Mozer 88].

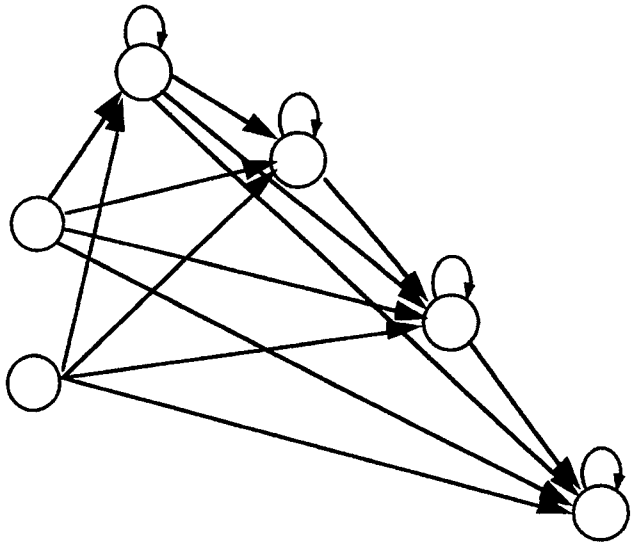

Figure II.15. "Recurrent Cascade Correlation"

Ce type de structure est moins général que la structure entièrement connectée, ce qui restreint le champ d'application [Chen *et al.* 93].

Dans [Chen *et al.* 93] est abordé le problème des réseaux constructifs entièrement connectés, mais seul un schéma très rudimentaire de croissance est testé : on rajoute une unité toutes les cinquante présentations. Les réseaux convergent très rapidement sur une application simple.

En ce qui concerne les techniques d'élagage, un algorithme est proposé dans [Omlin & Giles 93] qui s'avère efficace. Dans cet algorithme, les unités inutiles (selon un critère donné) sont successivement retirées du réseau après apprentissage.

La technique classique de décroissance des poids ("weight decay"), qui peut être utilisée à des fins d'élagage, a aussi été testée sur les réseaux récurrents [Kamimura 92][Chen *et al.* 93]. Celle-ci consiste à ajouter dans l'erreur un terme dont la minimisation entraînera une diminution des poids : ainsi, certains poids nuls ou quasi nuls peuvent être supprimés. Les résultats expérimentaux sont positifs [Kamimura] mais moins bons que la technique d'élagage [Chen *et al.* 93].

En conclusion, il apparaît que beaucoup reste à faire en ce qui concerne les architectures évolutives récurrentes : expérimentations, nouveaux algorithmes et théorie.

II.3 Application aux réseaux CORDIS

II.3.1 Introduction et notations

Dans ce chapitre, nous allons appliquer les algorithmes d'apprentissage dans les réseaux récurrents (voir chapitre II.2) aux réseaux de modélisation physique *CORDIS* (voir chapitre II.1). Les équations obtenues sont équivalentes à celles dérivées au chapitre précédent même si elles peuvent apparaître plus complexes. Cette complexité d'écriture provient essentiellement du fait que l'on manipule des réseaux qui ont deux types d'automates, et trois types de paramètres.

Les automates sont décrits au chapitre II.1.1.b. Nous en rappelons les équations :

Pour les masses :

$$X_i(n) = 2X_i(n-1) - X_i(n-2) + \frac{1}{m_i}\left(\sum_{j \in V(i)} F_{ji}(n-1) + Fext_i(n-1) \right) \qquad (11)$$

Pour les liaisons :

$$\begin{aligned} F_{ij}(n) &= k_{ij}\left(X_i(n) - X_j(n)\right) + z_{ij}\left(\left(X_i(n) - X_i(n-1)\right) - \left(X_j(n) - X_j(n-1)\right)\right) \\ F_{ji}(n) &= -F_{ij}(n) \end{aligned} \qquad (12)$$

Les signaux $X_i(n)$ sont initialisés à zéro pour n inférieur ou égal à zéro. Nous introduisons les notations suivantes :

$Xdes_i(n)$	position désirée de la masse i
A	ensemble des indices pour lesquels la position désirée est connue (on suppose A constant)
N	indice du dernier élément de la séquence
δ_{ij}	symbole de Kronecker : vaut *1* si et seulement si $i=j$
$\delta_{ij,kl}$	symbole de Kronecker étendu : vaut *1* si et seulement si $i=k$ et $j=l$

Toutes les variables décrites avec deux indices (hormis les symboles de Kronecker) ne sont définies et calculées que pour des valeurs différentes de ces indices.

On définit les erreurs suivantes :

Erreur instantanée locale : $e_i(n) = \begin{cases} X_i(n) - Xdes_i(n) & \text{si i} \in \text{A} \\ 0 & \text{si i} \notin \text{A} \end{cases}$

Erreur instantanée : $E(n) = \sum_{i \in A} \left(X_i(n) - Xdes_i(n)\right)^2 = \sum_{i \in A} e_i(n)^2$

Erreur totale : $Etot = \sum_{n=1}^{N} E(n)$

Les paramètres sont adaptés selon le principe de descente de gradient (voir figure II.9) :

$$\Delta \, 1/m_i = -\alpha \frac{\partial Etot}{\partial (1/m_i)} = \sum_{n=1}^{N} \Delta \, 1/m_i(n) \text{ avec } \Delta \, 1/m_i(n) = -\alpha \frac{\partial E(n)}{\partial (1/m_i)} \qquad (13)$$

$$\Delta k_{ij} = -\beta \frac{\partial Etot}{\partial k_{ij}} = \sum_{n=1}^{N} \Delta k_{ij}(n) \text{ avec } \Delta k_{ij}(n) = -\beta \frac{\partial E(n)}{\partial k_{ij}} \qquad (14)$$

$$\Delta z_{ij} = -\gamma \frac{\partial Etot}{\partial z_{ij}} = \sum_{n=1}^{N} \Delta z_{ij}(n) \text{ avec } \Delta z_{ij}(n) = -\gamma \frac{\partial E(n)}{\partial z_{ij}} \qquad (15)$$

où les constantes positives α, β et γ, sont les pas d'apprentissage.

II.3.2 Rétropropagation dans le temps (BPTT) dans un réseau CORDIS

Afin de ne pas oublier de terme dans les dérivations, on a intérêt à représenter le réseau déplié dans le temps, ce qui permet de faire figurer toutes les dépendances (voir figure II.16). On a été amené à dissocier en deux unités les traitements effectués par les ressorts amortis, même si les deux unités délivrent deux signaux égaux au signe près. On remarquera un certain nombre de connexions en "court-circuit", qui sautent d'une couche à l'autre ; elles sont dues aux mémoires des automates.

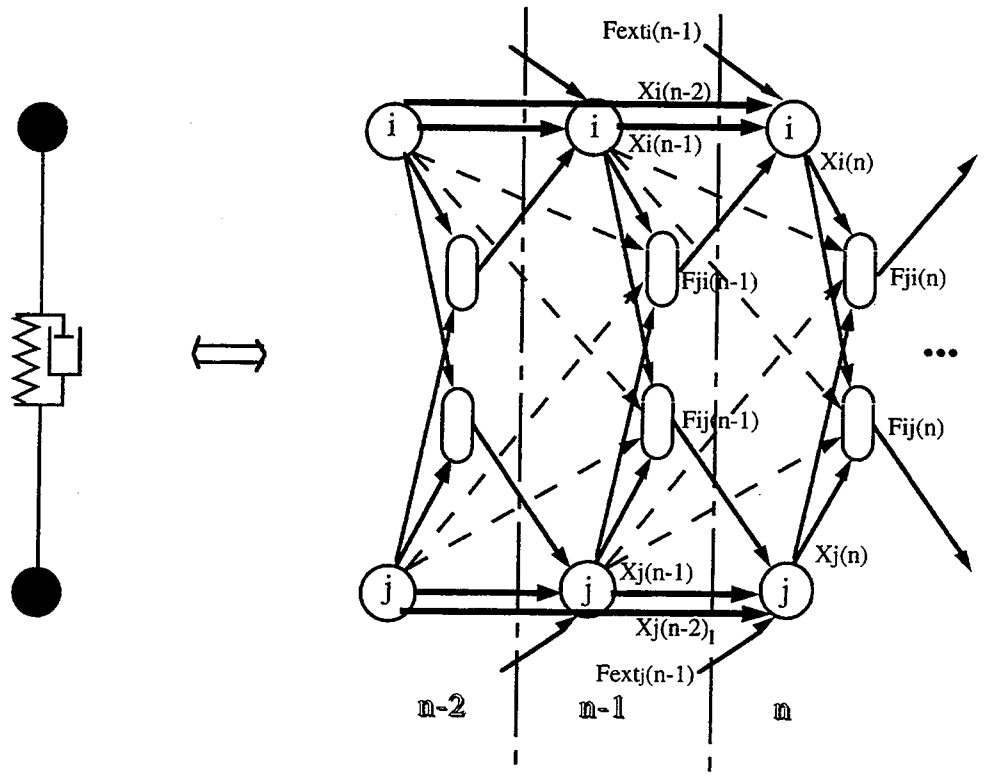

Figure II.16. Réseau CORDIS déplié

En appliquant la règle de la dérivation d'une fonction composée à (13), on a :

$$\Delta {}^{1}\!/_{m_i} = -\alpha \sum_{\eta=1}^{N} \frac{\partial Etot}{\partial X_i(\eta)} \cdot \frac{\partial X_i(\eta)}{\partial {}^{1}\!/_{m_i}} = -\alpha \sum_{\eta=1}^{N} \frac{\partial Etot}{\partial X_i(\eta)} \cdot \left(\sum_{j \in V(i)} F_{ji}(\eta-1) + Fext_i(\eta-1) \right)$$

On pose alors :

$$D_i(\eta) = \frac{\partial Etot}{\partial X_i(\eta)}$$

ce qui donne :

$$\Delta {}^{1}\!/_{m_i} = -\alpha \sum_{\eta=1}^{N} D_i(\eta) \cdot \left(\sum_{j \in V(i)} F_{ji}(\eta-1) + Fext_i(\eta-1) \right) \qquad (16)$$

Pour les raideurs, on utilise de la même manière l'expression (14) :

$$\Delta k_{ij} = -\beta \sum_{\eta=1}^{N} \frac{\partial Etot}{\partial F_{ij}(\eta)} \cdot \frac{\partial F_{ij}(\eta)}{\partial k_{ij}} + \frac{\partial Etot}{\partial F_{ji}(\eta)} \cdot \frac{\partial F_{ji}(\eta)}{\partial k_{ij}}$$

Les deux termes de la somme proviennent du fait que chaque raideur est utilisée pour produire *deux* signaux ; Le schéma de la figure II.16 montre clairement cette double dépendance. On a ensuite, d'après (12) :

$$\frac{\partial F_{ij}(\eta)}{\partial k_{ij}} = X_i(\eta) - X_j(\eta) = -\frac{\partial F_{ji}(\eta)}{\partial k_{ij}}$$

et l'on peut poser :

$$\delta_{ij}(\eta) = \frac{\partial Etot}{\partial F_{ji}(\eta)} - \frac{\partial Etot}{\partial F_{ij}(\eta)}$$

ce qui donne finalement :

$$\Delta k_{ij} = -\beta \sum_{\eta=1}^{N} \delta_{ij}(\eta)\big(X_j(\eta) - X_i(\eta)\big) \tag{17}$$

De même, pour les viscosités, on obtient :

$$\Delta z_{ij} = -\gamma \sum_{\eta=1}^{N} \delta_{ij}(\eta)\Big[\big(X_j(\eta) - X_j(\eta-1)\big) - \big(X_i(\eta) - X_i(\eta-1)\big)\Big] \tag{18}$$

Il reste maintenant à calculer les $D_i(\eta)$ et les $\delta_{ij}(\eta)$.

L'erreur *Etot* dépend de $X_i(\eta)$ d'une part directement via $e_i(\eta)$, et d'autre part par la propagation de $X_i(\eta)$ dans le réseau, c'est-à-dire via $X_i(\eta+1)$, $X_i(\eta+2)$, $\{F_{li}(\eta), F_{il}(\eta), F_{li}(\eta+1), F_{il}(\eta+1)\}_{l\in V(i)}$, comme l'illustre le réseau déplié de la figure II.16. On a donc, pour $\eta \le N-2$:

$$\begin{aligned}
D_i(\eta) &= \frac{\partial Etot}{\partial X_i(\eta)} \\
&= \frac{\partial Etot}{\partial e_i(\eta)} \cdot \frac{\partial e_i(\eta)}{\partial X_i(\eta)} \\
&\quad + \frac{\partial Etot}{\partial X_i(\eta+1)} \cdot \frac{\partial X_i(\eta+1)}{\partial X_i(\eta)} + \frac{\partial Etot}{\partial X_i(\eta+2)} \cdot \frac{\partial X_i(\eta+2)}{\partial X_i(\eta)} \\
&\quad + \sum_{l\in V(i)} \left(\frac{\partial Etot}{\partial F_{li}(\eta)} \cdot \frac{\partial F_{li}(\eta)}{\partial X_i(\eta)} + \frac{\partial Etot}{\partial F_{il}(\eta)} \cdot \frac{\partial F_{il}(\eta)}{\partial X_i(\eta)} \right. \\
&\qquad \left. + \frac{\partial Etot}{\partial F_{li}(\eta+1)} \cdot \frac{\partial F_{li}(\eta+1)}{\partial X_i(\eta)} + \frac{\partial Etot}{\partial F_{il}(\eta+1)} \cdot \frac{\partial F_{il}(\eta+1)}{\partial X_i(\eta)} \right)
\end{aligned}$$

Or :

$$\frac{\partial Etot}{\partial e_i(\eta)} \cdot \frac{\partial e_i(\eta)}{\partial X_i(\eta)} = \begin{cases} 2e_i(\eta)\times 1 & \text{si } i \in A \\ 0 & \text{si } i \notin A \end{cases}$$

donc, comme dans le cas où $i \notin A$, on a par définition $e_i(\eta) = 0$, on obtient :

$$\frac{\partial Etot}{\partial e_i(\eta)} \cdot \frac{\partial e_i(\eta)}{\partial X_i(\eta)} = 2e_i(\eta)$$

D'autre part :

$$\frac{\partial Etot}{\partial X_i(\eta+1)} \cdot \frac{\partial X_i(\eta+1)}{\partial X_i(\eta)} = D_i(\eta+1)\times 2 = 2D_i(\eta+1)$$

$$\frac{\partial Etot}{\partial X_i(\eta+2)} \cdot \frac{\partial X_i(\eta+2)}{\partial X_i(\eta)} = D_i(\eta+2) \times (-1) = -D_i(\eta+2)$$

$$\frac{\partial F_{li}(\eta)}{\partial X_i(\eta)} = -k_{li} - z_{li} = -\frac{\partial F_{il}(\eta)}{\partial X_i(\eta)}$$

$$\frac{\partial F_{li}(\eta+1)}{\partial X_i(\eta)} = z_{li} = -\frac{\partial F_{il}(\eta+1)}{\partial X_i(\eta)}$$

Donc finalement :

$$D_i(\eta) = 2e_i(\eta) + 2D_i(\eta+1) - D_i(\eta+2) + \sum_{l \in V(i)} (k_{li} + z_{li})\delta_{li}(\eta) - z_{li}\delta_{li}(\eta+1) \qquad (19)$$

En ce qui concerne le calcul des $\delta_{ij}(\eta)$, on peut de même écrire, pour $\eta \leq N-1$:

$$\delta_{ij}(\eta) = \frac{\partial Etot}{\partial X_i(\eta+1)} \cdot \frac{\partial X_i(\eta+1)}{\partial F_{ji}(\eta)} - \frac{\partial Etot}{\partial X_j(\eta+1)} \cdot \frac{\partial X_j(\eta+1)}{\partial F_{ij}(\eta)}$$

ce qui donne :

$$\delta_{ij}(\eta) = \frac{D_i(\eta+1)}{m_i} - \frac{D_j(\eta+1)}{m_j} \qquad (20)$$

Les expressions (19) et (20) restent valides pour η=*N-1* et η=*N*, en posant comme conditions initiales pour la rétropropagation :

$$D_i(N+1) = D_i(N+2) = \delta_{ij}(N+1) = 0 \qquad (21)$$

L'algorithme d'apprentissage se résume donc d'une part par les formules (16) (17) (18), qui correspondent à la formule (7) en réseau de neurones classique, et d'autre part aux formules (19) (20) qui correspondent à la formule (8) ; les conditions initiales sont fixées par (21).

Notre objectif étant rappelons-le de résoudre le problème d'identification en utilisant le plus possible les modèles physiques (voir chapitre I.4.2), il faut se demander en quoi l'algorithme d'adaptation *BPTT* appliqué aux réseaux mécaniques s'interpréterait lui-même de manière mécanique.

On peut observer une analogie entre les expressions (19) et (20) et les expressions (11) et (12), qui décrivent l'évolution dynamique d'un réseau mécanique. Effectivement, si l'on pose :

$$\begin{cases} \overline{D}_i(\eta) = \dfrac{1}{m_i} D_i(\eta) \\ \overline{\delta}_{ij}(\eta) = (k_{ij} + z_{ij})\delta_{ij}(\eta - 1) - z_{ij}\delta_{ij}(\eta) \end{cases}$$

on obtient la paire d'équations suivante :

$$\begin{cases} \overline{D}_i(\eta) = 2\overline{D}_i(\eta+1) - \overline{D}_i(\eta+2) + \dfrac{1}{m_i}\left(e_i(\eta) + \sum_l \overline{\delta}_{il}(\eta+1) \right) \\ \overline{\delta}_{ij}(\eta) = (k_{ij} + z_{ij})\left(\overline{D}_i(\eta) - \overline{D}_j(\eta)\right) + z_{ij}\left(\left(\overline{D}_i(\eta) - \overline{D}_i(\eta+1)\right) - \left(\overline{D}_j(\eta) - \overline{D}_j(\eta+1)\right)\right) \end{cases}$$

Cette paire d'équations, analogues aux équations (11) et (12), décrit donc l'évolution dynamique d'un réseau "masses-ressorts-frottements" auquel on applique les forces $e_i(\eta)$, dont le comportement se déroule en remontant le temps. A partir de là, pour adapter les paramètres selon les expressions (16) à (18), le passage de $D_i(\eta)$ à $\overline{D}_i(\eta)$ est aisé, mais le passage de $\delta_{ij}(\eta)$ à $\overline{\delta}_{ij}(\eta)$ est plus délicat, sauf si les amortissements sont nuls. Quoiqu'il en soit, l'analogie physique ne peut être poussée plus loin, car les équations (16) à (18) impliquent un *stockage* sur toute la séquence, difficilement interprétable en termes mécaniques.

II.3.3 RTRL appliqué aux réseaux CORDIS

Pour l'algorithme qui va être décrit, les signaux suivants sont introduits :

$$S_i^j(n) = \frac{\partial X_j(n)}{\partial(1/m_i)}$$

$$R_i^{lj} = \frac{\partial F_{lj}(n)}{\partial(1/m_i)}$$

$$\sigma_{ij}^l(n) = \frac{\partial X_l(n)}{\partial k_{ij}}$$

$$\rho_{ij}^{hl}(n) = \frac{\partial F_{hl}(n)}{\partial k_{ij}}$$

$$\tau_{ij}^l(n) = \frac{\partial X_l(n)}{\partial z_{ij}}$$

$$\chi_{ij}^{hl}(n) = \frac{\partial F_{hl}(n)}{\partial z_{ij}}$$

Partant de l'équation (13), on peut écrire :

$$\Delta\, 1/m_i(n) = -\alpha \frac{\partial E(n)}{\partial(1/m_i)} = -2\alpha \sum_{l\in A}\left(X_l(n) - Xdes_l(n)\right)\frac{\partial X_l(n)}{\partial(1/m_i)}$$

ce qui donne :

$$\Delta\, 1/m_i(n) = -2\alpha \sum_{l\in A}\left(X_l(n) - Xdes_l(n)\right)S_i^l(n) \qquad (22)$$

De même, on a :

$$\Delta k_{ij}(n) = -2\beta \sum_{l \in A} \left(X_l(n) - Xdes_l(n)\right)\sigma_{ij}^l(n) \tag{23}$$

$$\Delta z_{ij}(n) = -2\gamma \sum_{l \in A} \left(X_l(n) - Xdes_l(n)\right)\tau_{ij}^l(n) \tag{24}$$

Calculons les signaux S, R, σ, ρ, τ et χ, en dérivant les équations (11) et (12) par rapport aux paramètres du réseau. En dérivant (11) par rapport aux inverses des masses :

$$S_i^l(n) = 2S_i^l(n-1) - S_i^l(n-2) + \sum_{j \in V(l)} \frac{\partial F_{jl}(n-1)}{\partial(1/m_i)} + \delta_{il} \sum_{j \in V(l)} F_{jl}(n-1)$$

donc :

$$S_i^l(n) = 2S_i^l(n-1) - S_i^l(n-2) + \frac{1}{m_l} \sum_{j \in V(l)} R_i^{jl}(n-1) + \delta_{il} \sum_{j \in V(l)} F_{jl}(n-1) \tag{25}$$

De même, à partir de (12), on peut écrire :

$$R_i^{jl}(n) = \left(k_{jl} + z_{jl}\right)\left(S_i^j(n) - S_i^l(n)\right) - z_{jl}\left(S_i^j(n-1) - S_i^l(n-1)\right) \tag{26}$$

Les quatre autres signaux s'obtiennent de manière analogue :

$$\sigma_{ij}^l(n) = 2\sigma_{ij}^l(n-1) - \sigma_{ij}^l(n-2) + \frac{1}{m_l} \sum_{h \in V(l)} \rho_{ij}^{hl}(n-1) \tag{27}$$

$$\rho_{ij}^{hl}(n) = \left(k_{hl} + z_{hl}\right)\left(\sigma_{ij}^h(n) - \sigma_{ij}^l(n)\right) - z_{hl}\left(\sigma_{ij}^h(n-1) - \sigma_{ij}^l(n-1)\right) + \delta_{hl,ij}\left(X_i(n) - X_j(n)\right) \tag{28}$$

$$\tau_{ij}^l(n) = 2\tau_{ij}^l(n-1) - \tau_{ij}^l(n-2) + \frac{1}{m_l} \sum_{h \in V(l)} \chi_{ij}^{hl}(n-1) \tag{29}$$

$$\chi_{ij}^{hl}(n) = \left(k_{hl} + z_{hl}\right)\left(\tau_{ij}^h(n) - \tau_{ij}^l(n)\right) - z_{hl}\left(\tau_{ij}^h(n-1) - \tau_{ij}^l(n-1)\right) + \delta_{hl,ij}\left(\left(X_i(n) - X_i(n-1)\right) - \left(X_j(n) - X_j(n-1)\right)\right) \tag{30}$$

Ce signaux sont initialisés à zéro au temps zéro.

L'algorithme se résume donc par les formules (22), (23) et (24) d'une part, qui correspondent à la formule (9) pour l'algorithme *RTRL* classique, et aux formules (25) à (30), qui correspondent à la formule (10).

Quelle interprétation physique peut être donnée à ces formules ? Les équations (25) à (30) forment trois systèmes de deux équations, décrivant chacun l'évolution d'un réseau "masses-ressorts-frottements". Par exemple, dans les équations (25) et (26), S correspond exactement à une position, et R à une force, l'indice i étant fixé. Il y a autant de ces réseaux, que l'on nomme *réseaux d'adaptation*, que de paramètres adaptables, indicés par i et ij dans les expressions (25) à (30). Le schéma de la figure II.17 représente tous ces réseaux ; il correspond à une sous-partie de celui de la figure II.11. Par contre, l'adaptation proprement dite, selon les formules (23), (24) et (25), ne trouve pas immédiatement d'interprétation mécanique.

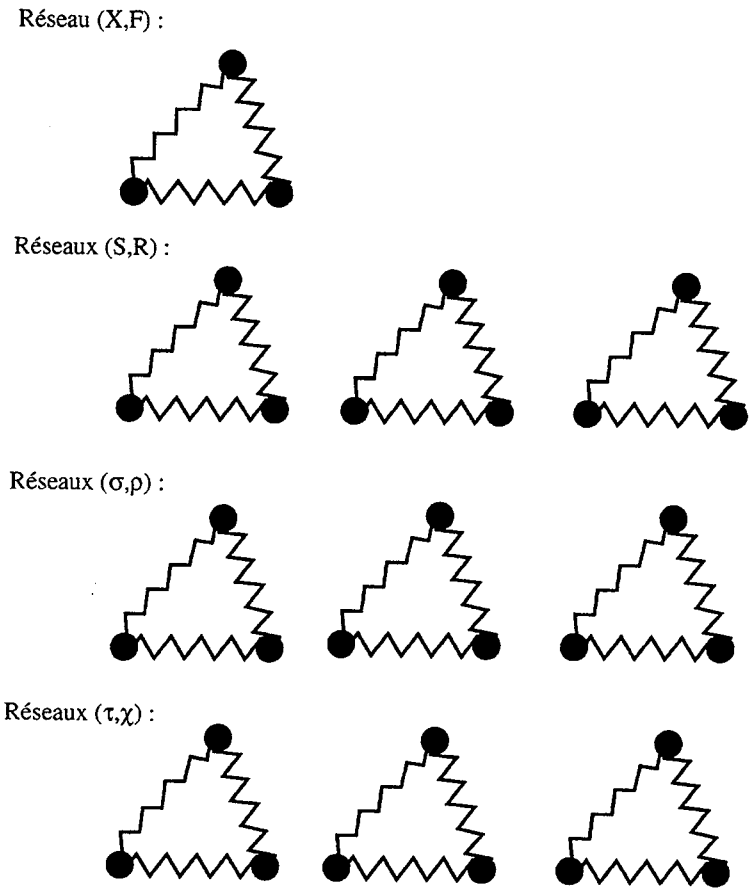

Figure II.17. Réseau initial et réseaux d'adaptation CORDIS

II.3.4 Apprentissage d'un modèle modal

Le modèle modal, qui a été évoqué au chapitre I.4.1.b, est une représentation que l'on peut dériver de tout réseau linéaire décrit par les formules (11) et (12), et qui est strictement équivalente à celui-ci. Nous étudions ici comment, à l'aide des techniques adaptatives des réseaux de neurones, on peut rendre un modèle modal adaptatif.

Un modèle modal comporte trois éléments : une matrice d'entrée, une série d'oscillateurs élémentaires (une masse de valeur *1* attachée au sol par un ressort

amorti de raideur k_i et de viscosité z_i) et une matrice de sortie (voir figure II.18). La matrice d'entrée transforme toutes les forces extérieures en une série de forces, notées $Fm_i(n)$, qui sont appliquées aux oscillateurs élémentaires ; celles-ci délivrent des positions au temps n, dénotées $Xm_i(n)$, qui sont fournies à la matrice de sortie qui délivre les positions des masses du réseau (voir figure II.18). On note la matrice de passage Q et ${}^tQ$ sa transposée.

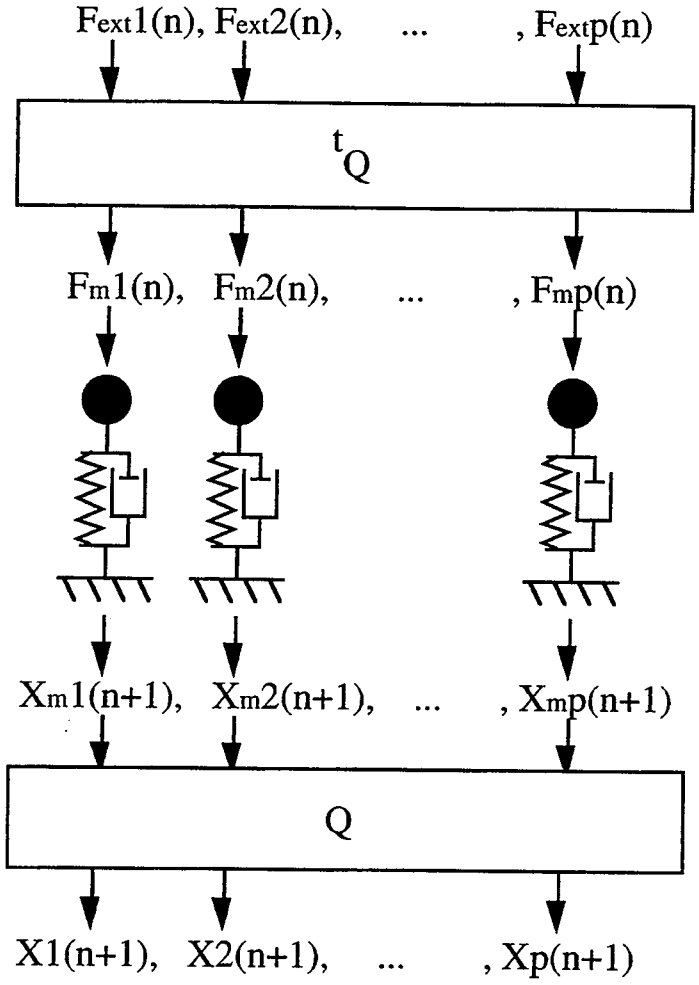

Figure II.18. Synthèse modale

Les équations de ce modèle sont donc :

$$Fm_i(n) = \sum_{j=1}^{p} ({}^tQ)_{ij} Fext_j(n) \tag{31}$$

$$Xm_i(n) = (2 - k_i - z_i)Xm_i(n-1) - (1 - z_i)Xm_i(n-2) + Fm_i(n) \tag{32}$$

$$X_i(n) = \sum_{j=1}^{p} q_{ij} Xm_j(n) \tag{33}$$

La figure II.19 représente ce modèle sous forme de réseau connexionniste. Il s'agit donc d'un réseau à trois couches :

- la première couche est entièrement connectée à la seconde ;

- la seconde couche est "auto-récurrente", c'est-à-dire que les automates ont une mémoire de leurs états passés (ici de leurs *deux* états passés) ;
- la troisième couche est entièrement connectée à la seconde.

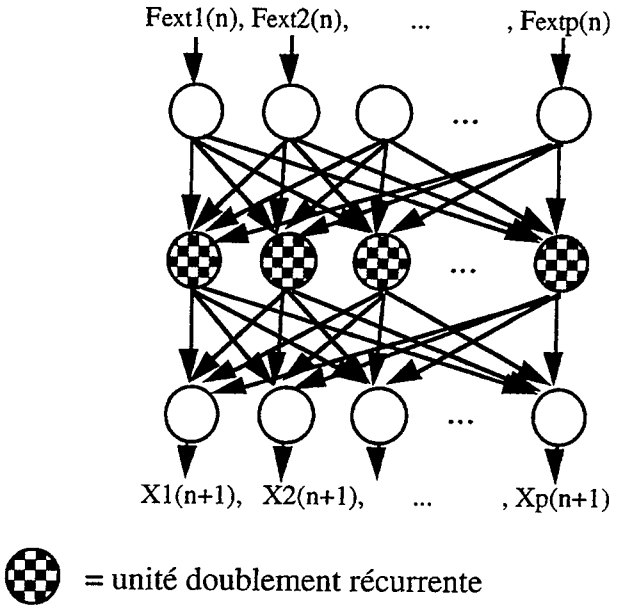

Figure II.19. Synthèse modale sous la forme d'un réseau à couches

Ce type de réseau a la même structure que les réseaux étudiés dans [Mozer 88], sauf que les automates de la couche cachée sont ici doublement récurrents : chaque automate reçoit en entrée non seulement son état précédent mais aussi son état encore précédent. Il rentre dans la catégorie des modèles localement récurrents globalement unidirectionnels [Tsoi & Back 94]. Il faut aussi rapprocher le réseau de la figure II.19 du réseau étudié dans [Codina *et al.* 94] où les unités cachées d'un réseau à couches entraîné avec la rétropropagation du gradient ont pour fonction de transfert des fonctions sinus et cosinus, afin que le réseau reproduise une analyse de Fourier : de même, dans la représentation modale, chaque cellule produit un signal sinusoïdal.

On peut adapter un tel réseau en utilisant l'algorithme *RTRL*. Le calcul est alors beaucoup plus simple que dans le cas entièrement interconnecté, car les récurrences sont concentrées à l'intérieur de chaque automate de la couche cachée.

Il y a trois types de paramètres à identifier : les raideurs, les viscosités *et* les éléments de la matrice Q. Les pas d'apprentissage respectifs de ces trois types de paramètres sont α,β et γ. On définit trois signaux :

$$A_i(n) = \frac{\partial Xm_i(n)}{\partial k_i}$$

$$B_i(n) = \frac{\partial Xm_i(n)}{\partial z_i}$$

$$C_{ij}(n) = \frac{\partial Xm_j(n)}{\partial q_{ij}}$$

On peut alors utiliser le même critère d'erreur que précédemment et écrire l'adaptation des paramètres selon le gradient de l'erreur :

$$\Delta k_i(n) = -\alpha \frac{\partial E(n)}{\partial k_i} = -2\alpha \sum_{j \in A} \left(X_j(n) - Xdes_j(n) \right) \frac{\partial X_j(n)}{\partial k_i}$$

Or :

$$\frac{\partial X_j(n)}{\partial k_i} = \sum_{k=1}^{p} q_{jk} \frac{\partial Xm_k(n)}{\partial k_i} = q_{ji} \frac{\partial Xm_i(n)}{\partial k_i}$$

car seul Xm_i dépend de k_i (c'est à ce niveau que l'algorithme est simplifié).

Donc :

$$\Delta k_i(n) = -2\alpha \left(\sum_{j \in A} \left(X_j(n) - Xdes_j(n) \right) q_{ji} \right) A_i(n) \qquad (34)$$

On a de même :

$$\Delta z_i(n) = -2\beta \left(\sum_{j \in A} \left(X_j(n) - Xdes_j(n) \right) q_{ji} \right) B_i(n) \qquad (35)$$

Pour l'adaptation des éléments de la matrice de passage, nous détaillons un petit peu les calculs :

$$\Delta q_{ij}(n) = -2\gamma \sum_{k \in A} \left(X_k(n) - Xdes_k(n) \right) \frac{\partial X_k(n)}{\partial q_{ij}}$$

$$\frac{\partial X_k(n)}{\partial q_{ij}} = \frac{\partial}{\partial q_{ij}} \sum_{l=1}^{p} q_{kl} Xm_l(n) = \delta_{ik} Xm_j(n) + \sum_{l=1}^{p} q_{kl} \frac{\partial Xm_l(n)}{\partial q_{ij}}$$

$$= \delta_{ik} Xm_j(n) + q_{kj} \frac{\partial Xmj(n)}{\partial q_{ij}}$$

Donc :

$$\Delta q_{ij}(n) = -2\gamma \left(\sum_{k \in A} \left(X_k(n) - Xdes_k(n) \right) q_{kj} \right) C_{ij}(n)$$
$$-2\gamma 1_A \left(X_i(n) - Xdes_i(n) \right) Xm_i(n) \qquad (36)$$

avec :

$$1_A = \begin{cases} 1 \text{ si } i \in A \\ 0 \text{ sinon} \end{cases}$$

Il faut maintenant calculer les signaux *A,B* et *C*. Ceux-ci se calculent comme les signaux *S, R,* σ, ρ, τ et χ : on dérive l'équation (32) par rapport aux paramètres pour obtenir :

$$A_i(n) = (2 - k_i - z_i)A_i(n-1) - (1 - z_i)A_i(n-2) - Xm_i(n-1) \quad (37)$$

$$B_i(n) = (2 - k_i - z_i)B_i(n-1) - (1 - z_i)B_i(n-2) - (X_i(n-1) - X_i(n-2)) \quad (38)$$

$$C_{ij}(n) = (2 - k_j - z_j)C_{ij}(n-1) - (1 - z_j)C_{ij}(n-2) + Fext_i(n-1) \quad (39)$$

Les valeurs initiales de *A, B* et *C* sont zéro.

L'algorithme d'apprentissage se résume par les équations (34) à (39).

Il est encore une fois possible de donner une interprétation physique à certaines de ces formules. On remarquera en effet que les équations (37) à (39) décrivent des oscillateurs élémentaires, identiques à ceux décrits par l'équation (32), excités par un signal spécifique.

Par rapport aux algorithmes d'adaptations pour les réseaux mécaniques qui ont été proposés plus haut, l'adaptation du modèle modal présente certains avantages.

D'une part, le coût computationnel est très faible. Alors qu'un réseau à p masses entièrement connectées entraîné avec *RTRL* nécessite $O(p^2)$ et $O(p^4)$ calculs par unité de temps pour la relaxation et l'apprentissage respectivement, seulement O(p) et O(p) sont nécessaires pour l'algorithme qui vient d'être décrit.

D'autre part, les paramètres k_i et z_i ont une signification précise en terme de signal. Par exemple, la valeur $\sqrt{k_i} / 2\pi$ est la *fréquence* du mode i de la structure. Il est donc possible d'interpréter du point de vue acoustique les paramètres obtenus par apprentissage. De plus et surtout, on peut *injecter dans le réseau une connaissance a priori*, en initialisant les paramètres non plus aléatoirement mais en fonction d'une estimation des fréquences propres du système à identifier.

Ces fréquences propres peuvent être obtenues par une simple analyse de Fourier de la réponse impulsionnelle. Une méthode plus physique peut néanmoins être

utilisée, comme la technique du*banc de cellules* [Incerti & Cadoz 95] : en excitant par un signal donné un très grand nombre de cellules masses-ressorts-frottements dont le rapport raideur/masse est croissant, on obtient les fréquences approchées du signal en sélectionnant les cellules dont les amplitudes sont les plus élevées (phénomène de résonance). On pourrait envisager un tel banc de cellules comme un prétraitement qui permettrait de sélectionner les cellules et donc les paramètres initiaux pour un modèle modal adaptatif.

Notons tout de même qu'un tel prétraitement n'est pas non plus incompatible avec les autres algorithmes décrits dans ce mémoire.

II.4 Variantes inspirées par la mécanique

II.4.1 Couplage mécanique

Les algorithmes qui viennent d'être décrits sont des transcriptions d'algorithmes neuronaux classiques au cas des réseaux mécaniques. Nous avons pu, dans une certaine mesure, interpréter les algorithmes obtenus en termes mécaniques, grâce à une analogie formelle entre les expressions décrivant certaines variables de l'apprentissage (variables de sensibilité pour l'algorithme *RTRL* et variables de rétropropagation pour l'algorithme *BPTT*) et les équations d'un réseau CORDIS. Mais on peut aussi tenter d'introduire des éléments mécaniques directement dans les mécanismes d'apprentissage.

Ainsi, les échanges d'information entre l'objet à identifier et l'objet simulé se limitent jusqu'à présent à un simple transfert d'une position désiré de l'un vers l'autre. On peut proposer de réaliser un couplage *mécanique* entre l'objet à identifier et l'objet simulé. Du point de vue de la modélisation, cela revient à insérer un *ressort amorti* simulé entre les deux objets au niveau de chaque point d'accès (voir figure II.20).

Le critère qui est minimisé, à savoir l'erreur quadratique entre le signal produit par le réseau et le signal désiré, correspond alors au carré de la force produite par le ressort de couplage, c'est-à-dire à l'*énergie* qui transite dans ce ressort. Quand l'énergie qui transite est nulle, alors le ressort ne "travaille" pas et le réseau a le même comportement que l'objet à identifier. Le critère à minimiser a donc une signification physique.

Le ressort amorti de couplage introduit, décrit comme toute liaison viscoélastique par la formule (12), n'est pas anodin dans le déroulement de l'apprentissage, puisqu'il implique que :

- l'environnement modifie directement le *comportement* du réseau, en plus de la modification des paramètres ;
- l'échange d'information n'est plus unidirectionnel mais *bidirectionnel*, c'est-à-dire que non seulement l'objet à identifier fournit sa position mais de plus la position de l'objet simulé modifie le comportement de l'objet de l'environnement ;
- l'apprentissage ne peut être que *temps réel*, puisque l'évolution du signal désiré est lui même fonction de l'évolution du réseau simulé.

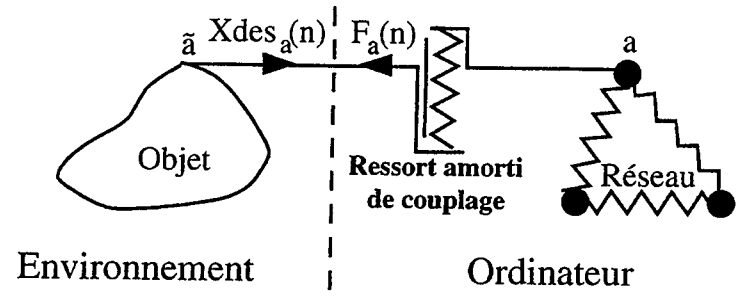

Figure II.20. Couplage entre l'objet à identifier et le réseau (cas où le point d'accès est unique)

Nous allons étudier l'effet sur l'algorithme *RTRL* de l'introduction d'un ressort de couplage entre l'objet à identifier et le réseau. Les résultats sont transposables à l'algorithme *BPTT*.

En tout point d'accès est ajouté un ressort de couplage avec l'objet de l'environnement. Pour tout $a \in A$, on définit $\tilde{a}$, indice du point sur l'objet de l'environnement qui fournit le comportement désiré $Xdes_a(n)$. Les points a et $\tilde{a}$ sont donc reliés par une liaison de couplage. On a alors :

$$\begin{aligned} &\tilde{a} \in V(a) \\ &Xdes_a(n) = X_{\tilde{a}}(n) \end{aligned} \qquad (40)$$

Selon cette notation supplémentaire, le réseau couplé peut être décrit par les mêmes équations (11) et (12) que précédemment : au niveau des points d'accès, les forces en provenance des liaisons de couplages sont prises en compte.

Afin d'obtenir les équations d'adaptation (22), (23) et (24), on a dérivé l'erreur instantanée $E(t)$ en fonction des paramètres. Reprenons ce calcul dans le cas de ressorts de couplage :

$$\begin{aligned} \Delta\, {}^{1}\!/_{m_i}(n) &= -\alpha \frac{\partial E(n)}{\partial(1/m_i)} \\ &= -2\alpha \sum_{l \in A} \left(X_l(n) - Xdes_l(n)\right) \left(\frac{\partial X_l(n)}{\partial(1/m_i)} - \frac{\partial Xdes_l(n)}{\partial(1/m_i)} \right) \end{aligned}$$

Or il n'est plus possible de considérer que la position désirée ne dépend pas des paramètres du réseau, puisque cette position dépend des positions du réseau via le ressort de couplage, donc des paramètres de masse, raideur et viscosité du réseau. Donc :

$$\frac{\partial Xdes_l(n)}{\partial(1/m_i)} \neq 0$$

De plus, cette dérivée est *incalculable*, car les paramètres du réseau ont une influence sur les positions désirées d'une part directement mais d'autre part indirectement, via l'objet mécanique entier lui-même. Or comme justement on ne connaît pas cet objet, il est impossible de savoir comment il est influencé par les paramètres du réseau. On se trouve donc dans une impasse, un cercle vicieux même : pour reproduire le comportement d'un objet de l'environnement, on a besoin de connaître cet objet, or c'est ce que l'on cherche...

Ce problème est évoqué dans [Hecht-Nielsen 90], mais aucune solution n'est proposée. Nous n'avons pas connaissance de solution à ce problème dans le champ de l'identification de systèmes. Il apparaît donc que le fait de coupler bidirectionnellement un système à identifier au système simulé qui doit reproduire le système à identifier conduit à une impossibilité de résolution.

Nous faisons finalement le choix d'affecter zéro aux termes non calculables, pour finalement conserver les équation (22), (23) et (24). Ceci constitue bien sûr une approximation : on ne suit plus la direction du gradient.

Les formules (25) à (30), qui se rapportent aux réseaux supplémentaires d'adaptation (figure II.17) sont toujours valides, si l'on tient compte du fait que la fonction de voisinage $V(i)$ est étendue au niveau de certains points de l'objet à identifier, selon (40). Ceci amène à définir et calculer, pour $a \in A$, les signaux suivants :

$$R_i^{\tilde{a}a}(n) = \frac{\partial F_{\tilde{a}a(n)}}{\partial \, 1/m_i} = -(k_{\tilde{a}a} + z_{\tilde{a}a}) S_i^a(n) + z_{\tilde{a}a} S_i^a(n-1)$$

$$\rho_{ij}^{\tilde{a}a}(n) = \frac{\partial F_{\tilde{a}a(n)}}{\partial k_{ij}} = -(k_{\tilde{a}a} + z_{\tilde{a}a}) \sigma_{ij}^a(n) + z_{\tilde{a}a} \sigma_{ij}^a(n-1)$$

$$\chi_{ij}^{\tilde{a}a}(n) = \frac{\partial F_{\tilde{a}a(n)}}{\partial z_{ij}} = -(k_{\tilde{a}a} + z_{\tilde{a}a}) \tau_{ij}^a(n) + z_{\tilde{a}a} \tau_{ij}^a(n-1)$$

On a supposé pour obtenir ces expressions que les positions désirées ne dépendaient pas des paramètres.

L'interprétation de ces formules est très simple : chaque réseau d'adaptation est attaché à un point fixe (masse infinie, qui renvoie donc une nulle), par un ressort amorti de raideur $k_{\tilde{a}a}$ et de viscosité $z_{\tilde{a}a}$ (voir figure II.21).

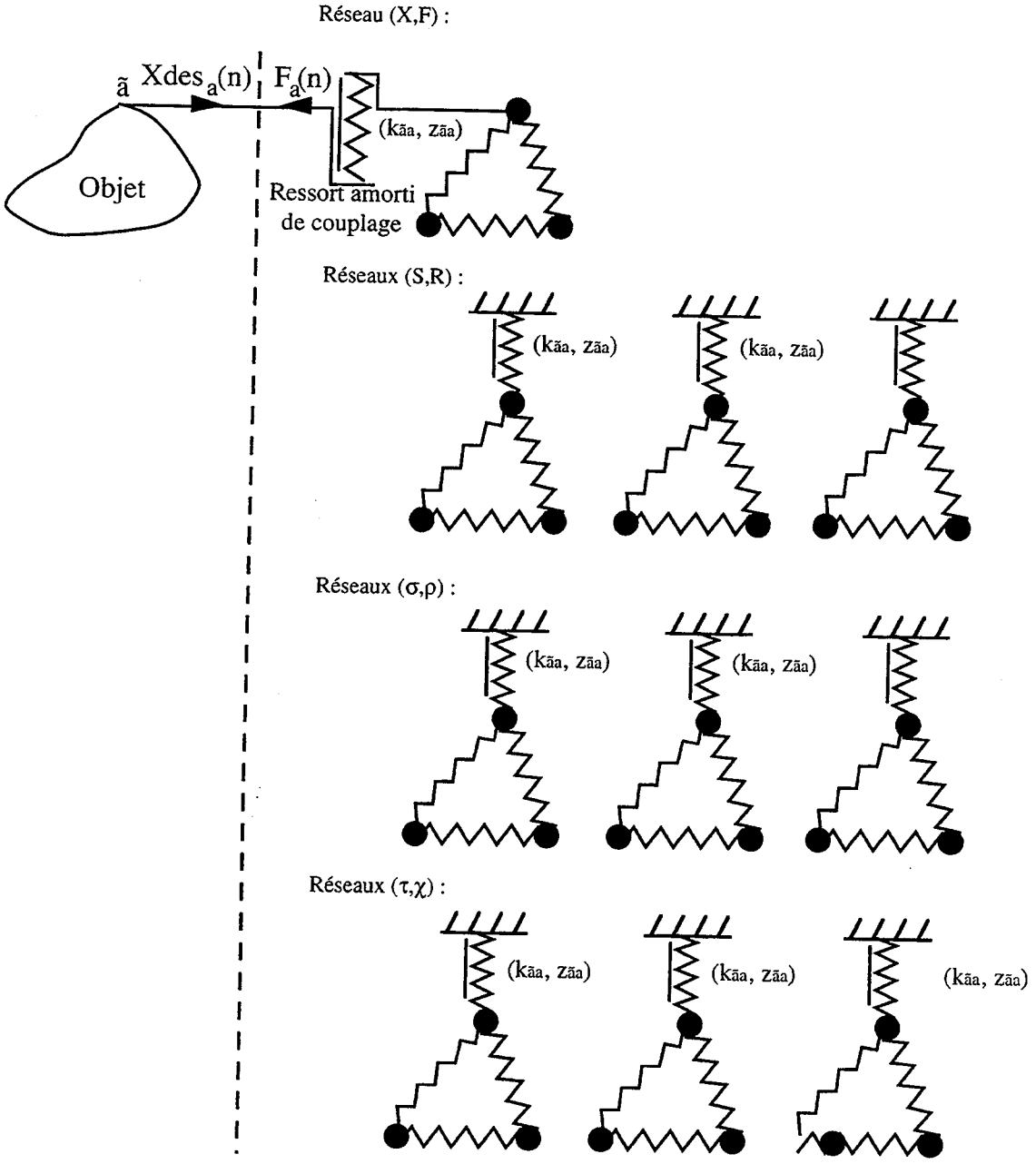

Figure II.21. Réseau initial et réseaux d'adaptation dans le cas du couplage

Il est tout à fait intéressant de noter que si les couplages ont une viscosité non nulle, alors les réseaux d'adaptation, qui calculent les termes de sensibilité, sont amortis. Or un problème des réseaux récurrents entraînés avec l'algorithme *RTRL* est justement que les termes de sensibilité ont tendance à exploser (voir paragraphe II.2.3.d). Le couplage est donc une solution potentielle à ce problème. Amortir les termes de sensibilité apparaît alors comme un moyen de faciliter l'apprentissage en *raccourcissant* la fenêtre temporelle utilisée pour l'adaptation (voir paragraphe

II.2.4.b), et ce de manière graduelle, contrairement à toutes les autres méthodes existantes (voir par exemple [Catfolis 93]).

Mentionnons tout de même que si les dérivées des positions désirées par rapport aux paramètres sont incalculables, il existe néanmoins une solution pour trouver ces valeurs. Comme nous allons le voir, la solution proposée ne présente qu'un intérêt théorique, car elle est totalement irréalisable en pratique.

Afin de simplifier les calculs qui vont suivre, on considère que le réseau possède un seul point d'accès a, mais le résultat peut être directement étendu au cas général. Le système {réseau + ressort de couplage} envoie à l'objet à identifier une force $F_a(n)$ et en reçoit une position $Xdes_a(n)$, comme illustré sur la figure II.20.

Etant donnés les calculs réalisés par un réseau CORDIS, exprimé par les formules (11), (12), on peut considérer que $F_a(n)$ se calcule selon le système :

$$\begin{cases} F_a(n) = G\big(Xdes_a(n), I(n)\big) \\ I(n) = H\big(I(n-1), Xdes_a(n-1)\big) \end{cases} \tag{41}$$

avec $I(n)$ l'ensemble des variables internes du réseau (y compris les états mémorisés), G et H deux fonction linéaires. Le fait que ces fonctions soient linéaires permet d'écrire, pour tout paramètre p du réseau (masse, raideur ou viscosité) :

$$\begin{cases} \dfrac{\partial F_a(n)}{\partial p} = G\left(\dfrac{\partial Xdes_a(n)}{\partial p}, \dfrac{\partial I(n)}{\partial p}\right) \\ \dfrac{\partial I(n)}{\partial p} = H\left(\dfrac{\partial I(n-1)}{\partial p}, \dfrac{\partial Xdes_a(n-1)}{\partial p}\right) \end{cases} \tag{42}$$

On constate que les formules (41) et (42) sont analogues : on retrouve le fait que les réseaux d'adaptation sont eux aussi des réseaux mécaniques, *identiques au réseau initial.*

De même, on peut écrire l'objet à identifier sous la forme :

$$\begin{cases} Xdes_a(n) = \tilde{G}\big(F_a(n-1), \tilde{I}(n-1)\big) \\ \tilde{I}(n) = \tilde{H}\big(\tilde{I}(n-1), F_a(n-1)\big) \end{cases} \tag{43}$$

où $\tilde{I}(n)$ désigne l'ensemble des variables internes de l'objet à identifier au temps n, c'est-à-dire ses degrés de liberté.

Posons maintenant l'hypothèse simplificatrice suivante : l'objet à identifier se comporte de manière linéaire, C'est-à-dire $\tilde{G}$ et $\tilde{H}$ sont linéaires. Alors on obtient de même :

$$\begin{cases} \dfrac{\partial Xdes_a(n)}{\partial p} = \tilde{G}\left(\dfrac{\partial F_a(n-1)}{\partial p}, \dfrac{\partial \tilde{I}(n)}{\partial p} \right) \\ \dfrac{\partial \tilde{I}(n)}{\partial p} = \tilde{H}\left(\dfrac{\partial \tilde{I}(n-1)}{\partial p}, \dfrac{\partial F_a(n-1)}{\partial p} \right) \end{cases} \tag{44}$$

De même que (41) et (42) sont analogues, (43) et (44) sont analogues, ce qui signifie que l'on peut calculer les dérivées par rapport aux paramètres en utilisant une copie de l'objet à identifier (44), couplé avec une copie du réseau initial (42), comme il est représenté sur la figure II.22.

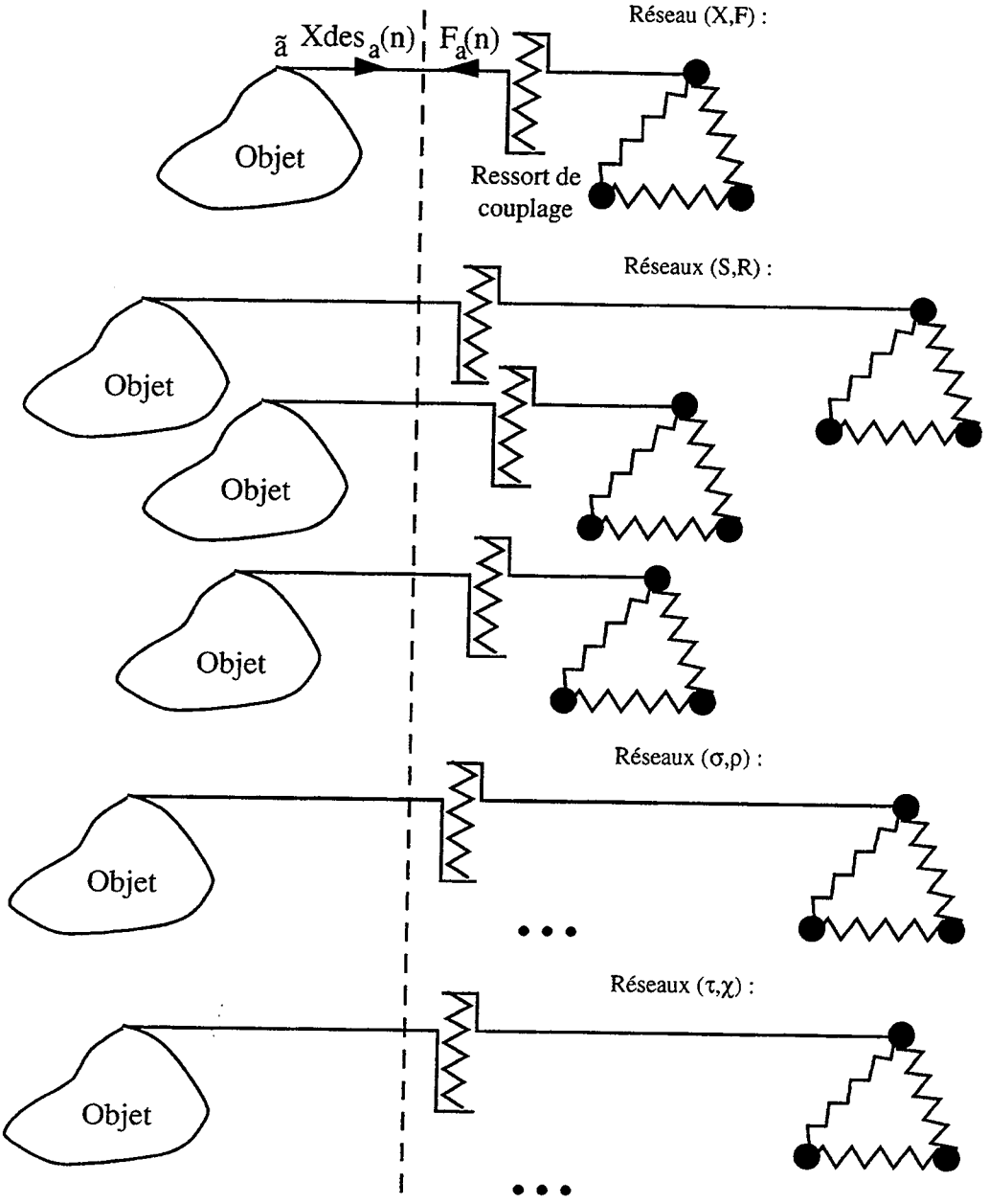

Figure II.22. Solution au problème du calcul du gradient exact dans le cas du couplage

Cette possibilité est irréalisable en pratique, puisqu'elle nécessite pour identifier un objet mécanique, d'en disposer autant de copies que de paramètres adaptables ! Il s'agit néanmoins de la seule possibilité de calculer le gradient exact en ligne dans le cas du couplage

II.4.2 Semi-couplage et forçage

Trois raisons nous ont amenés à étudier une variante du couplage, à savoir le *semi-couplage* :

- le couplage présente des problèmes pour dériver les formules exactes ;

- dans la pratique, il est rarement possible de réaliser un tel couplage, puisqu'il nécessite une interface *bidirectionnelle* entre le réseau et l'objet à identifier ; même si dans le cadre de la présente étude nous disposons d'un Transducteur Gestuel Rétroactif (voir le chapitre III.4), il est souhaitable de développer des algorithmes moins contraints par des aspects technologiques ;
- comme nous l'avons écrit plus haut, le couplage implique d'une part une modification directe des *activités* du réseau par l'objet à identifier, et d'autre part une *action* du réseau sur cet objet. Il serait intéressant de dissocier ces deux effets, afin en particulier de saisir précisément le rôle de l'action dans l'apprentissage.

Le semi-couplage consiste ainsi à insérer entre le réseau et l'objet à identifier, au niveau des points d'accès, des "pseudo ressorts", à la place des ressorts utilisés dans le cas du couplage. Ces "pseudo ressorts" ont la particularité de ne pas émettre de force vers l'objet. On a donc :

$$F_{\tilde{a}a}(n) = k_{\tilde{a}a}\left(Xdes_a(n) - X_a(n)\right) + z_{\tilde{a}a}\left(\left(Xdes_a(n) - Xdes_a(n-1)\right) - \left(X_a(n) - X_a(n-1)\right)\right)$$
$$F_{a\tilde{a}}(n) = 0$$

Ce n'est donc plus un réel lien mécanique entre le réseau et l'objet à identifier, puisque la liaison est unidirectionnelle.

Les formules d'apprentissage sont exactement les mêmes que pour le couplage, mais sans qu'il soit nécessaire de tronquer les formules arbitrairement, car dans ce cas, les dérivés des valeurs désirées par rapport aux paramètres sont effectivement nulles.

Cet algorithme a cette particularité commune avec l'algorithme de forçage ("teacher forcing") que le comportement du réseau est ramené sur la trajectoire correcte : les raideurs de couplage ont pour effet de solidariser le réseau et l'objet mécanique. Il s'avère donc important d'étudier pour comparaison l'algorithme de forçage appliqué aux réseaux mécaniques.

Si l'on transpose les résultats de [Williams & Zipser 89a], l'une des manières de présenter le forçage est la suivante : après avoir effectué les adaptations selon les formules (22), (23) et (24), pour tout $a \in A$, on affecte les valeur de $Xdes_a(n)$ aux variables $X_a(n)$, et zéro aux variables $S_i^a(n)$, $\sigma_{ij}^a(n)$ et $\tau_{ij}^a(n)$. Cette formulation, qui ne pose certes pas de problème à programmer, est néanmoins gênante, puisqu'elle revient à avoir deux valeurs différentes pour une même variable au même indice temporel.

Pour l'affectation forcée de $X_a(n)$, si l'on veut éviter ces deux valeurs successives, il est nécessaire de dissocier deux variables : la position utilisée dans les formules (22), (23) et (24) d'une part, et la position utilisée dans la formule (12) d'autre part. Pour les remise à zéro des $S_i^a(n)$, $\sigma_{ij}^a(n)$ et $\tau_{ij}^a(n)$, il faut de même, pour éviter les doubles affectations, modifier les calculs des $R_i^{al}(n)$ (de même pour les signaux $\sigma_{ij}^{al}(n)$ et $\tau_{ij}^{al}(n)$) en utilisant, la place de (26) la formule :

$$R_i^{al}(n) = -(k_{al} + z_{al})S_i^l(n) + z_{al}S_i^l(n-1) \qquad (45)$$

tout en gardant l'équation (25).

Dans ce cas, les réseaux d'adaptation ne sont plus mécaniques, puisque d'un côté ils sont attachés au sol, selon la formule (45) ci-dessus, et de l'autre, les positions de ces "sols" sont variables selon (25).

Il devient clair donc que l'algorithme de forçage tel qu'il est présenté dans [Williams & Zipser 89a] se prête mal à une interprétation physique. Nous trouvons un résultat bien différent en ce qui concerne le forçage par l'erreur de [Toomarian & Barhen 91, 92]. En effet, si on adapte la formule proposée (voir chapitre II.2.4.a) aux réseaux mécaniques, on obtient, pour $a \in A$:

$$X_a(n) = 2X_a(n-1) - X_a(n-2) + \frac{1}{m_a}\left(\sum_{j \in V(a)} F_{ja}(n-1) + Fext_a(n-1)\right)$$
$$+\lambda\left(\left[Xdes_a(n-1)\right]^{1-\beta}\left[Xdes_a(n-1) - X_a(n-1)\right]^{\beta}\right)$$

Dans le cas où β vaut 1, cette formule devient :

$$X_a(n) = 2X_a(n-1) - X_a(n-2) + \frac{1}{m_a}\left(\sum_{j \in V(a)} F_{ja}(n-1) + Fext_a(n-1)\right)$$
$$+\lambda\left(Xdes_a(n-1) - X_a(n-1)\right)$$

On retrouve alors exactement l'algorithme de semi-couplage avec un viscosité de couplage nulle, λ étant la raideur de couplage ! En effet, le dernier terme de l'équation ci-dessus n'est rien d'autre que l'équation d'un ressort.

Les conséquences de cette correspondance sont multiples.

D'une part, si un même algorithme peut s'obtenir par deux voies totalement différentes, l'on se pose immédiatement la question de savoir quelle formulation est plus générale que l'autre, laquelle englobe l'autre. Aucune en fait, puisque d'un côté le forçage par l'erreur est plus général si l'on considère les cas où $\beta \neq 1$, et de l'autre côté, l'algorithme de semi-couplage est plus général si l'on considère le cas où $z \neq 0$

(l'algorithme de semi-couplage peut être traduit pour des neurones formels classiques).

D'autre part, on pourrait être déçu de passer par toute une analogie mécanique pour aboutir finalement à un algorithme qui existe quasiment déjà. Au contraire, il faut se réjouir de pouvoir constater que l'analogie mécanique peut aboutir à des algorithmes tout à fait pertinents, et cela encourage à approfondir dans cette voie.

Ensuite, il est remarquable que c'est précisément une analogie mécanique qui est utilisée dans [Toomarian & Barhen 91, 92], pour illustrer la notion de forçage (apprentissage du vélo à un enfant : voir chapitre II.2.4.a), mais les auteurs n'ont pas été jusqu'à considérer que l'erreur fournie en entrée au réseau peut être interprétée comme une force.

Plus remarquable encore, ces mêmes auteurs proposent, par analogie au guidage mécanique du professeur, de faire varier dans le temps le paramètre λ, c'est-à-dire, si l'on interprète mécaniquement les équations, la raideur de couplage. Or l'on montre par des études biomécaniques que l'un des paramètres principal de contrôle moteur est l'équivalent de la raideur du muscle en question. Faire varier la raideur de couplage au cours du temps est donc très plausible du point de vue biomécanique.

Toutes ces "coïncidences" nous conduisent donc à penser que l'analogie entre l'identification paramétrique d'un objet mécanique et l'apprentissage moteur guidé par un professeur, est à approfondir, que ce soit pour construire des algorithmes plus performants que pour modéliser et comprendre l'apprentissage moteur.

II.4.3 Couplage/découplage

Quel que soit l'algorithme de forçage ou de couplage, on constate que quand le réseau reproduit parfaitement le comportement de l'objet, alors le réseau forcé ou couplé se comporte comme si il était libre. Ceci nous a suggéré une nouvelle idée pour adapter les paramètres dans un réseau mécanique.

Considérons *deux* réseaux identiques (mêmes structure et paramètres) : l'un couplé avec l'objet à identifier, l'autre libre, les deux réseaux recevant les mêmes forces extérieures. Dans le cas où le réseau couplé reproduit exactement le comportement de l'objet, alors le réseau couplé et le réseau libre se comportent de manière identique. Ainsi, un critère à minimiser peut être l*a différence entre le réseau libre et le réseau couplé* (voir figure II.23).

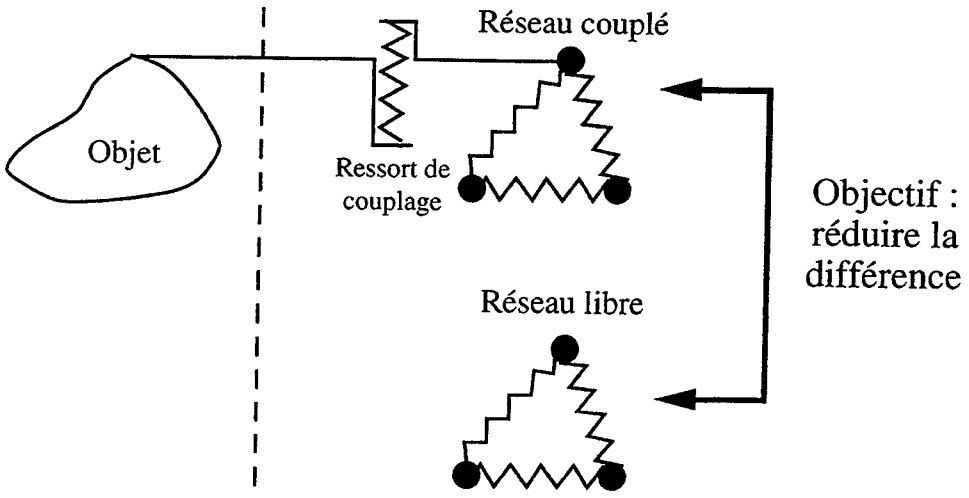

Figure II.23. Principe de l'algorithme de couplage/découplage

Dans un premier temps, on peut mesurer cette différence uniquement aux points d'accès, ce qui conduit à l'algorithme qui va suivre.

Les variables des réseaux libre et couplé sont dénotées respectivement par les indices "L" et "C".

Les équations décrivant les réseaux libres et couplés, et les réseaux d'adaptation correspondant sont les mêmes que précédemment et n'ont pas besoin d'être rappelées. Le critère instantané à minimiser est :

$$E(n) = \sum_{i \in A} \left(X_{L_i}(n) - X_{C_i}(n) \right)^2 \tag{46}$$

En dérivant cette erreur selon les paramètres, on trouve :

$$\Delta \, 1/m_i(n) = -2\alpha \sum_{l \in A} \left(X_{L_l}(n) - X_{C_l}(n) \right) \left(S_{L_i}^l(n) - S_{C_i}^l(n) \right) \tag{47}$$

$$\Delta k_{ij}(n) = -2\beta \sum_{l \in A} \left(X_{L_l}(n) - X_{C_l}(n) \right) \left(\sigma_{L_{ij}}^l(n) - \sigma_{C_{ij}}^l(n) \right) \tag{48}$$

$$\Delta z_{ij}(n) = -2\gamma \sum_{l \in A} \left(X_{L_l}(n) - X_{C_l}(n) \right) \left(\tau_{L_{ij}}^l(n) - \tau_{C_{ij}}^l(n) \right) \tag{49}$$

Les termes S, σ et τ se calculent pour les deux réseaux comme précédemment, selon les équations (25), (26) et (27) respectivement. Pour le réseau couplé, on n'oubliera pas les ressorts supplémentaires qui attachent les réseaux d'adaptation au sol (voir figure II.21).

L'algorithme qui vient d'être décrit est intéressant à tester mais ne présente a priori pas d'avantage sur la version originale ; au contraire, il y a deux fois plus de variables à calculer, puisque deux réseaux sont nécessaires.

Nous en proposons donc une autre version, approximée, qui va avoir l'avantage de supprimer la non-localité "irréductible" de *RTRL* (voir chapitre II.2.3.b et la figure II.12).

Au lieu de considérer l'erreur uniquement au niveau des points d'accès, selon la formule (46), rien ne nous empêche de la calculer *en tout point du réseau*. Ceci est possible car le critère à minimiser, la différence entre les deux réseaux, n'utilise pas la position désirée. L'erreur à minimiser devient donc :

$$E(n) = \sum_i \left(X_{L_i}(n) - X_{C_i}(n)\right)^2 = \sum_i E_i(n) \qquad (50)$$

avec :

$$E_i(n) = \left(X_{L_i}(n) - X_{C_i}(n)\right)^2$$

On fait alors l'approximation suivante :

$$\Delta 1/m_i(n) = -2\alpha \frac{\partial E(n)}{\partial(1/m_i)} \approx -2\alpha \frac{\partial E_i(n)}{\partial(1/m_i)}$$

Cette approximmation revient à considérer que l'erreur locale à un endroit donné du réseau est principalement influencée par les paramètres à cet endroit.

On obtient alors la formule d'adaptation suivante :

$$\Delta 1/m_i(n) = -2\alpha\left(X_{L_i}(n) - X_{C_i}(n)\right)\left(S_{L_i}^i(n) - S_{C_i}^i(n)\right) \qquad (51)$$

Pour les raideurs, on va considérer de même que :

$$\Delta k_{ij}(n) = -2\beta \frac{\partial E(n)}{\partial k_{ij}} \approx -2\beta \frac{\partial}{\partial k_{ij}}\left(E_i(n) + E_j(n)\right)$$

ce qui aboutit à la formule d'adaptation pour les raideurs :

$$\Delta k_{ij}(n) = -2\beta \begin{pmatrix} \left[X_{L_i}(n) - X_{C_i}(n)\right]\left[\sigma_{L_{ij}}^i(n) - \sigma_{C_{ij}}^i(n)\right] \\ +\left[X_{L_j}(n) - X_{C_j}(n)\right]\left[\sigma_{L_{ij}}^j(n) - \sigma_{C_{ij}}^j(n)\right] \end{pmatrix} \qquad (52)$$

Pour les viscosités, on a de même :

$$\Delta z_{ij}(n) = -2\gamma \begin{pmatrix} \left[X_{L_i}(n) - X_{C_i}(n)\right]\left[\tau_{L_{ij}}^i(n) - \tau_{C_{ij}}^i(n)\right] \\ +\left[X_{L_j}(n) - X_{C_j}(n)\right]\left[\tau_{L_{ij}}^j(n) - \tau_{C_{ij}}^j(n)\right] \end{pmatrix} \qquad (53)$$

Cette algorithme de couplage-découplage local, présente les avantages suivants :

- l'implémentation matérielle est facilitée ;

- si l'on veut utiliser une technique de regroupement (voir [Zipser 89] et le chapitre II.2.2.b), on n'a plus besoin d'avoir une sortie désirée par groupe.

Par contre, étant donnée l'approximation effectuée, la convergence vers un minimum n'est plus garantie. Le coût computationnel est du même ordre de grandeur que l'algorithme *RTRL*, mais toutefois deux fois plus élevé, du fait des deux réseaux. Les deux réseaux étant indépendants, l'algorithme devrait bien se paralléliser.

Une variante de cet algorithme de couplage-découplage, dont le développement est immédiat, consiste non plus à prendre un réseau couplé (ou semi-couplé) et un réseau libre mais deux réseaux (semi-)couplés, avec des coefficients de raideurs et viscosités de couplage différents. Cette variante, que nous nommons *couplage différentiel*, permettrait de profiter des avantages du couplage.

L'algorithme de couplage-découplage est intéressant sur le plan conceptuel, puisqu'il rend l'algorithme *RTRL* local, selon un schéma de regroupement analogue à celui représenté sur la figure II.12. Nous estimons qu'il constitue un premier pas vers une simplification de l'algorithme *RTRL*, et que d'autres approximations sont possibles pour trouver de nouvelles améliorations.

La notion de couplage-découplage peut être tout à fait adaptée aux réseaux de neurones traditionnels, en ayant deux réseaux, l'un libre et l'autre forcé. Nous avons pu d'ailleurs constater une certaine similitude entre l'algorithme de couplage-découplage et la Machine de Boltzmann ; dans ce dernier réseau, récurrent mais qui réalise des associations de patterns statiques, la loi d'adaptation tend en effet à modifier les poids de telle sorte que le comportement libre du réseau se rapproche du comportement forcé, dans le cadre d'une dynamique stochastique des activités [Ackey *et al.* 85].

*

La notion mécanique de couplage a finalement été très fructueuse, puisqu'elle a permis de proposer plusieurs algorithmes d'apprentissage intéressants, exposés ici dans le cadre des réseaux mécaniques, mais qui sont aisément traduisibles dans le cadre des réseaux de neurones formels habituels.

Des réflexions sur d'autres phénomènes mécaniques pouvant donner lieu à de nouveaux algorithmes sont encore à mener. En particulier, l'adaptation des paramètres dans les réseaux devraient être étudiés en référence aux modèles physiques de déformation plastique, usure, fatigue, etc. qui sont autant de domaines analogiques pour l'étude des algorithmes d'apprentissage.

II.5 Expérimentations

II.5.1 Méthodologie expérimentale

Toutes les expériences qui seront exposées dans ce chapitre ont été entièrement *simulées* sur ordinateur : le réseau adaptatif bien sûr, mais aussi l'interface, l'objet à identifier et l'opérateur. Il s'agit là d'une première étape qui a pour but de tester la validité des algorithmes présentés au chapitre II.4. Dans une seconde étape, l'interface, l'objet à identifier et l'opérateur seront bien réels, ce qui ajoutera une composante de bruit non négligeable.

Pour simuler l'objet réel, nous avons utilisé un réseau CORDIS, c'est-à-dire le même type de représentation que le réseau adaptatif lui-même. C'est évidemment un choix réducteur, puisqu'il revient à considérer que les modèles CORDIS que nous utilisons sont de bons modèles des objets réels, alors que l'on sait qu'ils sont le résultat de nombreuses approximations, en particulier au niveau de la linéarité (voir chapitre II.1.1.b). Cependant, ce choix, rarement fait dans le domaine des réseaux de neurones, présente l'intérêt de garantir qu'une solution parfaite existe. En effet, si l'on décide que le réseau simulant l'objet à identifier et le réseau adaptatif ont la même topologie, alors le jeu de paramètre du premier réseau est une solution pour le second réseau. Dans la suite, nous dénommerons respectivement *réseau fixe* et *réseau adaptatif* ces deux réseaux. Bien sûr, les deux réseaux ne s'échangent que leurs forces et positions au niveau de leurs points d'accès. Toutes les expériences ont été réalisées dans le cas où les deux réseaux ont la même topologie : 5 masses, dont une infinie représentant le sol, entièrement connectées par des ressorts amortis.

L'interface est réduite à sa plus simple expression, puisqu'elle est négligée. Ainsi, on se retrouve dans les conditions théoriques du chapitre précédent. Le nombre de points d'accès du réseau fixe, c'est-à-dire le nombre de points où un comportement désiré est donné, est fixé à *1*. Cela autorise une visualisation aisée des comportements des réseaux et simplifie les calculs, sans pour autant simplifier le problème, au contraire : quand il y a peu de points d'accès, certaines unités cachées du réseau

adaptatif doivent recevoir des informations pour être modifiées, tout en étant éloignées de tout point d'accès.

L'opérateur quant à lui est simulé par une excitation impulsionnelle au temps *0*, appliquée à chacun des points d'accès des deux réseaux. Il y donc à ce niveau une rupture avec le principe mécanique de bidirectionnalité ; un modèle interactif d'opérateur serait beaucoup plus délicat à mettre en place. Le choix d'une excitation impulsionnelle est guidé par le fait que son spectre harmonique est le plus complet : il comporte toutes les fréquences. Ainsi, l'excitation impulsionnelle, malgré son apparente simplicité est celle qui va révéler le plus de richesse spectrale d'une structure donnée. Notons que le point d'excitation de l'opérateur, choisi identique au point d'accès du réseau fixe, aurait pu se situer en un tout autre point de la structure : chaque réseau aurait eu un point d'accès et un point d'excitation. Le schéma II.24 illustre les simulations réalisées.

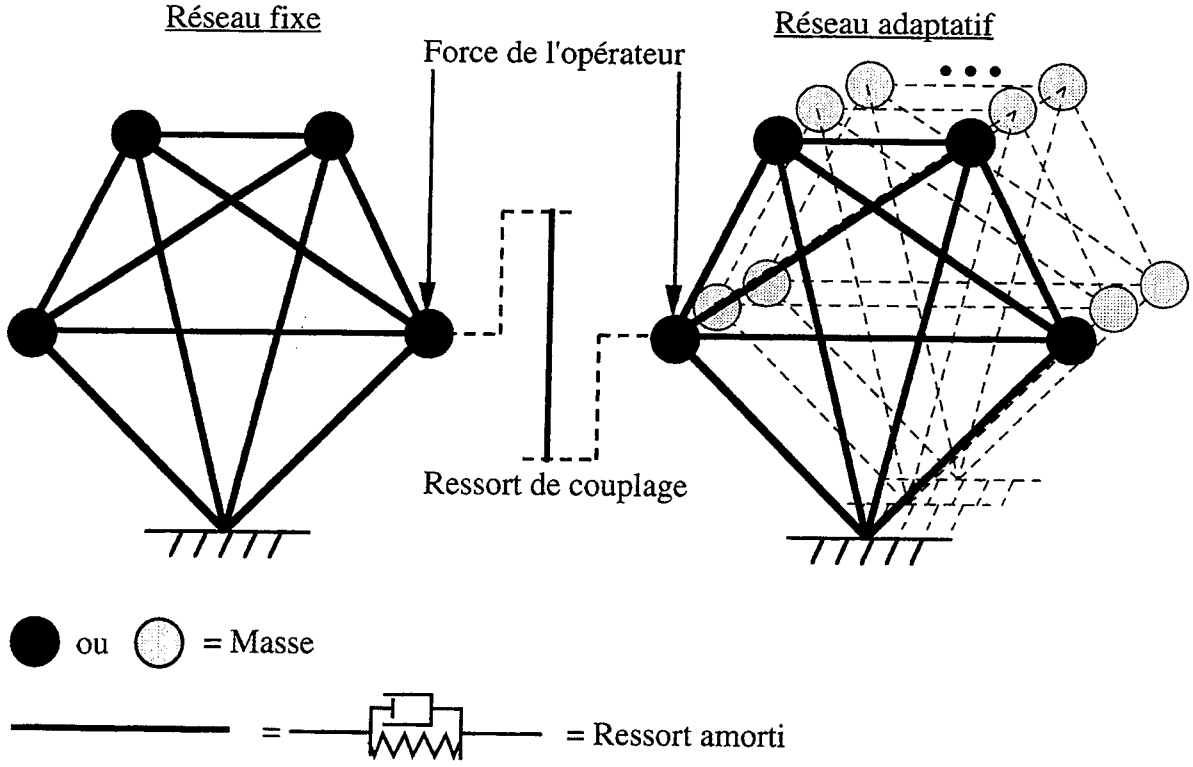

Figure II.24. Réseau fixe et réseau adaptatif (partiellement représenté)

Comme pour tout algorithme d'apprentissage connexionniste, le déroulement de l'apprentissage, et en particulier son succès, sont conditionnés par l'initialisation, aléatoire, des paramètres. Par conséquent, on ne peut se limiter à une seule expérience pour montrer la capacité d'un réseau, car l'on risquerait de tomber sur un cas particulier, favorable ou défavorable. Il faut donc réaliser une série de tests, et effectuer quelques statistiques sur ces tests. Dans tout ce chapitre, chaque algorithme a été testé vingt fois ; les vingt réseaux sont les mêmes quand on change d'algorithme (pour un

même type de test). Mais la méthodologie utilisée a été différente de celle habituellement utilisée en Réseaux de Neurones Formels. Au lieu de répliquer vingt fois l'apprentissage pour un réseau fixe donné en changeant l'initialisation du réseau adaptatif, nous avons changé *et* l'initialisation du réseau adaptatif *et* le choix des paramètres du réseau fixe. L'évaluation des capacités d'apprentissage s'effectue donc en initialisant les réseaux fixe et adaptatif de vingt manière différentes dans un intervalle de paramètres donné. Cela évite de définir arbitrairement les paramètres du réseau fixe. On comprend alors que la difficulté du problème d'apprentissage est très liée aux bornes de paramètres choisies : si les intervalles sont de petite taille, alors les deux réseaux auront des paramètres proches, donc des comportements voisins : l'apprentissage sera aisé. Par contre, dans le cas où les bornes des paramètres sont distantes, alors les paramètres pourront être très éloignés, entre les deux réseaux.

Cela nous a permis de réaliser deux types de tests. Le premier type de test, dit de *convergence locale*, consiste à prendre des intervalles très petits, afin que les deux réseaux soient dès le début proches. Dans ce cas, sur la surface d'erreur du réseau (voir figure II.8), le point représentant le jeu de paramètres du réseau adaptatif a toutes les chances d'être dans le même bassin d'attraction que celui représentant le réseau fixe, situé au creux du bassin. Donc, un algorithme qui consiste à suivre la direction du gradient doit théoriquement converger, sous certaines conditions (voir paragraphe II.2.3.d pour une discussion sur ces conditions). Par conséquent, nous souhaitons obtenir vingt succès sur vingt. Ce test n'est pas de pure formalité, si l'on considère le fait qu'un certain nombre d'algorithmes présentés ci-dessus ne suivent pas la direction exacte du gradient.

Le deuxième type de test, dit *test difficile*, s'approche d'une expérience réelle, dans la mesure où les intervalles de paramètres sont choisis suffisamment grands pour donner deux comportements nettement différents. C'est ce type de test qui permet de comparer les performances des différents algorithmes d'apprentissage. Les résultats *absolus* que l'on peut obtenir sont néanmoins difficilement exploitables : dire que tel algorithme converge dans par exemple *80* % des cas ne signifie rien, puisque ce taux est dépendant de la taille des intervalles de paramètres choisi au préalable pour effectuer le test. On s'intéressera donc essentiellement à *comparer* les résultats entre eux. Un contrôle visuel permet quand même de vérifier que les deux comportements étaient éloignés au départ et se sont effectivement rapprochés au cours de l'apprentissage.

Afin de décider quand un apprentissage est réussi ou non, il faut pouvoir évaluer une différence entre les deux comportements, pour ensuite définir un seuil relatif à

cette mesure. Certes, l'algorithme d'apprentissage minimise une erreur qu'on pourrait utiliser pour définir un critère d'arrêt, à savoir l'erreur :

$$Etot = \sum_{n=1}^{N} \left(X_a(n) - Xdes_a(n) \right)^2$$

avec a le point d'accès.

Mais cette erreur présente les défauts de croître avec le temps, de tenir compte autant du début du signal que de la fin et de ne pas être normalisée par rapport aux amplitudes des positions X. Nous avons opté pour le critère de différence suivant, que nous appelons *Différence Relative Moyenne (DRM)*, qui s'écrit :

$$DRM_M(n) = 100 \times \frac{\sum_{\eta=n-M+1}^{n} \left| X_a(\eta) - Xdes_a(\eta) \right|}{M \left(\max_{\eta=n-M+1,\ \ldots\ ,n} \left\{ Xdes_a(\eta) \right\} - \min_{n=n-M+1,\ \ldots\ ,n} \left\{ Xdes_a(\eta) \right\} \right)} \qquad (54)$$

Ce critère, estimé en continu à chaque échantillon, calcule la différence moyenne sur les M derniers échantillons entre les comportements des deux réseaux, normalisée par l'amplitude du signal désiré. Dans le cas des séquences courtes répétées qui font l'objet du chapitre III.5.2, on calculera simplement $DRM_N(N)$, N étant la longueur de la séquence. Précisons que plusieurs autres calculs de la différence auraient pu être choisis.

Le seuil pour décider que l'apprentissage est réussi a été fixé à *1* pour les tests de convergence locale, et à *5* pour les tests difficiles. Dans le premier cas, les courbes sont presque confondues, dans le deuxième, elles sont très proches. (voir figure II.25).

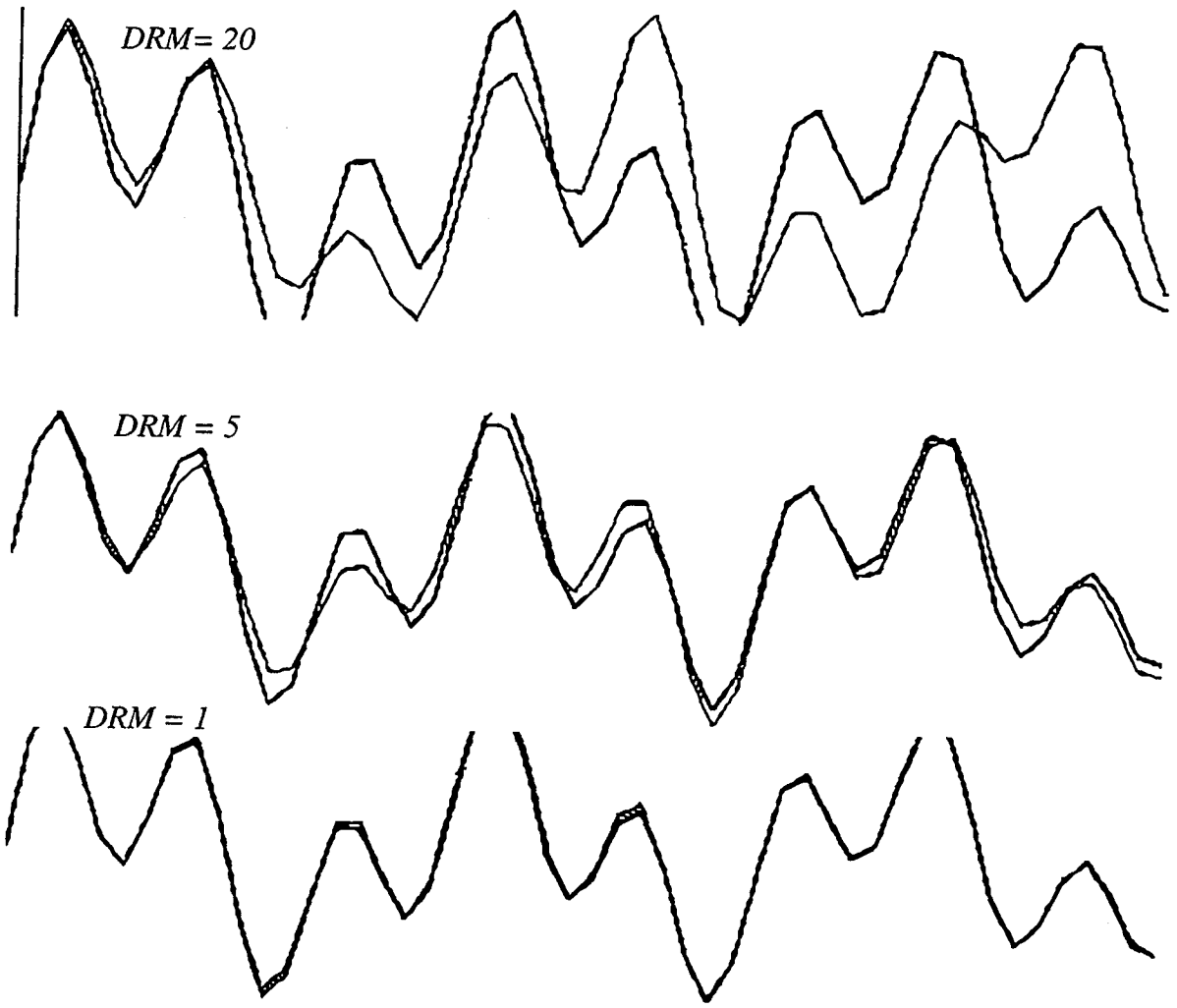

Figure II.25. Aspect des signaux selon la DRM, pour des séquences courtes (50 échantillons)

Un problème se pose quant à l'évaluation du succès des algorithmes "non libres", à savoir les algorithmes de couplage, semi-couplage et forçage. En effet, pour ces algorithmes, le comportement du réseau adaptatif en cours d'apprentissage n'est pas son comportement libre. La *DRM* mesurée en cours d'apprentissage ne correspondra donc pas à la différence réelle entre le réseau adaptatif et le réseau fixe, mesurée sans couplage ni forçage. On peut certes dupliquer le réseau adaptatif pour le tester en parallèle avec le même réseau couplé ou forcé, mais c'est une solution bien coûteuse. Nous avons plutôt opté pour une diminution des seuils de réussite. Il faut donc à l'aide d'expériences préliminaires, déterminer le *vrai seuil sur la DRM*, plus faible que le *seuil désiré*, tel que, en moyenne, si la *DRM* est en dessous de ce seuil *pendant* l'apprentissage, alors elle est en dessous de la *DRM* désirée après apprentissage, en évolution libre. Cela n'est pas la solution idéale, car cela implique un réglage manuel de ce seuil.

Quoiqu'il en soit, les algorithmes ont d'autres paramètres qui doivent être réglés manuellement par tâtonnement : les pas d'apprentissage, les raideur(s) et viscosité(s)

de couplage pour l'algorithme de couplage et ses variantes, les paramètres h et h' pour l'algorithme de rétropropagation par blocs. Cela pose le problème suivant, que l'on retrouve dans un grand nombre d'expériences utilisant des Réseaux de Neurones Formels : si tel algorithme a de meilleurs résultats que tel autre, est-ce dû à l'algorithme lui même ou à la performance du concepteur humain dans sa mise au point des paramètres ? Bien sûr, celui-ci essaiera dans tous les cas de trouver les paramètres optimaux, mais on n'a aucune garantie, surtout quand les paramètres sont nombreux. On ne peut donc analyser avec précision les résultats obtenus : c'est pourquoi aucun test statistique du type "t de Student", qui permet de mettre en évidence des effets fins, n'a été réalisé, quoique les données pour faire de tels tests soient disponibles. Nous nous sommes donc limités aux moyennes et écarts types.

Ont été mesurées pour ces simulations d'une part le nombre de présentations de la séquence (ou le nombre d'adaptations selon les cas) jusqu'à réussite de l'apprentissage, et d'autre part le nombre de succès, c'est-à-dire le nombre d'essais où l'apprentissage a réussi. Ce nombre de succès, ou taux de réussite si on le divise par le nombre d'essais total (c'est-à-dire *20*), est le critère principal qui nous permettra d'évaluer les performances de chacun des algorithmes.

L'apprentissage est considéré comme ayant échoué quand le nombre d'adaptations dépasse une valeur donnée, généralement *50000*. Un autre critère d'arrêt a été programmé pour gagner du temps : quand les paramètres dépassent certaines valeurs (10^6 pour les masses inversées, 100 pour les raideurs et les viscosités), alors on arrête l'apprentissage et on considère qu'il a échoué. En effet, on sait que quand les paramètres sont très grands, il y une "explosion" des forces et positions du réseau, qui ne décrivent plus des phénomènes périodiques mais exponentiels [Raoult 91]. Par contre, nous n'avons pas empêché les paramètres d'être négatifs, pour deux raisons : d'une part un paramètre négatif peut dans certains cas correspondre à un comportement intéressant ; d'autre part, il se peut qu'un paramètre au cours de l'apprentissage devienne négatif, puis à nouveau positif : la trajectoire des paramètres peut *passer* par des zones négatives.

II.5.2 Tests sur séquences courtes répétées

Dans une première série d'expériences, on a voulu tester et comparer les algorithmes dans le cas simplifié où les séquences sont courtes. En effet, comme on l'a développé au paragraphe II.2.3.d, un certain nombre de problèmes se posent pour les réseaux récurrents dès que l'on aborde les longues séquences, problèmes que dans un premier temps on a choisi d'éviter, pour les étudier au chapitre II.5.3.

Ces séquences sont courtes par rapport à l'application visée, l'identification de paramètres dans les structures vibrantes, puisqu'elles ne contiennent que *50* échantillons. Précisons qu'étant produits par des réseaux linéaires "masses-ressorts-frottements", les valeurs des séquences varient continûment, et ont l'aspect de sommes de fonctions sinusoïdales, si la fréquence d'échantillonnage est élevée. L'apprentissage est répété de nombreuses fois, jusqu'à succès de l'apprentissage ou dépassement d'un nombre maximum de séquences, à savoir *1000*, ou explosion des paramètres.

Le test de convergence locale est le suivant :

- Intervalle pour les masses : [1,5 ; 1,7]
- Intervalle pour les raideurs : [0,09 ; 0,11]
- Intervalle pour les viscosités : [0 ; 0,05]

Pour ces intervalles, la valeur moyenne de la *DRM* sur les vingt réseaux testés est de *5,035509*, ce qui est assez faible (voir graphique II.25). Tous les paramètres sont adaptables : *4* masses, *10* raideurs et *10* viscosités, soit *24* paramètres au total.

Le test difficile est le suivant :

- Intervalle pour les masses : [1 ; 1]
- Intervalle pour les raideurs : [0,05 ; 0,3]
- Intervalle pour les viscosités : [0 ; 0]

Dans ce test, les paramètres de masses et viscosités sont égaux au départ. Ils sont contraints à rester fixe durant tout l'apprentissage (les pas d'apprentissage correspondants sont nuls). Seuls les dix raideurs sont adaptables. Cela ne signifie en rien que l'apprentissage est facilité. En particulier, le fait de prendre des coefficients de viscosité nuls complique la tâche, car les signaux ne s'amortissent pas. La valeur moyenne de la *DRM* sur les vingt réseaux testés est dans ce cas de *25,172169*. Précisons que l'intervalle sur les raideurs a été en fait choisi tel que l'algorithme de référence *RTRL* donne un taux de réussite autour de *10/20*.

II.5.2.a "Real-Time Recurrent Learning" (RTRL)

Il s'agit de l'algorithme *RTRL* appliqué aux réseaux CORDIS (voir chapitre II.3.3), dans sa version "en ligne".

Convergence locale

Les pas d'apprentissage ont été fixés à $2\ 10^{-5}$.

Pour les vingt réseaux, les nombres de présentations successives de la séquence jusqu'à l'atteinte du critère d'arrêt, sont les suivants :

Nombre de présentations de la séquence									
54	96	134	56	203	49	52	50	96	148
108	175	42	106	14	98	35	105	171	49

Les résultats se résument par le tableau suivant :

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
20/20	92	14	203	51

L'algorithme *RTRL* parvient donc à adapter ses paramètres dans *100* % des cas quand les paramètres sont proches de la solution.

Test difficile

Les pas ont été fixés à 10^{-5}.

On obtient les tableaux de résultats suivants :

Nombre de présentations de la séquence									
402	355	*	*	308	Explo	*	516	146	153
151	*	*	*	*	*	326	43	*	*

avec "*" signifiant que la limite des 1000 présentations a été dépassée, et "Explo" signifiant que les paramètres ont dépassé les valeurs maximales prédéfinies.

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
9/20	267	43	516	143

Les statistiques sur les nombres de présentations sont faites uniquement sur les essais réussis.

Le taux de *9/20* correspond à un score moyen auquel on va comparer les différents autres algorithmes testés.

II.5.2.b Rétropropagation dans le temps (BPTT)

Il s'agit de l'algorithme *BPTT* appliqué aux réseaux CORDIS (voir chapitre II.3.2), en effectuant l'adaptation à la fin de chaque séquence.

Convergence locale

Les pas d'apprentissage ont été fixés à 10^{-4}.

Les résultats sont les suivants :

Nombre de présentations de la séquence									
76	22	31	16	30	14	14	14	33	35
28	37	9	23	4	24	7	23	31	13

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
20/20	24	4	76	15

Encore une fois, le test de convergence locale est réussi.

Test difficile

Les pas ont été fixés à $2\ 10^{-5}$.

On obtient les tableaux de résultats suivants :

					Nombre de présentations de la séquence				
*	99	*	*	61	*	159	36	17	34
99	214	*	83	63	162	47	31	131	189

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
15/20	95	17	214	61

Les résultats sont donc bien meilleurs qu'avec l'algorithme *RTRL*, aussi bien pour le taux de réussite (*15/20* contre *9/20*) que pour le nombre de présentations (*95* contre *267*).

II.5.2.c Algorithme de couplage

Il s'agit de l'algorithme où un ressort amorti simulé est placé entre le réseau adaptatif et le réseau fixe (voir chapitre II.4.1).

Convergence locale

Les pas d'apprentissage sont fixés à $2\ 10^{-5}$. La raideur de couplage est de *0,1*, tandis que la viscosité de couplage est nulle. Le seuil sur la *DRM* mesurée en cours d'apprentissage pour déclarer l'apprentissage réussi est de *0,65*.

Les résultats sont les suivants :

Nombre de présentations de la séquence									
730	152	452	215	356	77	177	176	449	760
369	473	95	604	22	183	196	418	277	80

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
20/20	313	22	760	210

La *DRM* après apprentissage, calculée sans couplage, moyennée sur les vingt essais est de *0,990*, qui est effectivement inférieur à *1*.

Test difficile

Les pas sont fixés à 10^{-3} la raideur de couplage à *0,3* et la viscosité de couplage à *0,6*. Le seuil sur la *DRM* mesurée en cours d'apprentissage pour déclarer l'apprentissage réussi est de *0,58*.

On obtient les tableaux de résultats suivants :

Nombre de présentations de la séquence									
138	316	11	154	64	91	296	356	10	307
18	83	*	Explo	52	86	99	*	*	*

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
15/20	139	10	356	116

La *DRM* après apprentissage, calculée sans couplage, moyennée sur les quinze essais réussis est de *4,17* qui est effectivement inférieur à *5*.

Le couplage améliore nettement les performances de l'algorithme *RTRL*, puisque le taux de réussite est de *15/20* contre *9/20* pour *RTRL*.

II.5.2.d Algorithme de semi-couplage

Il s'agit d'une variante de l'algorithme de couplage où le ressort simulé de couplage n'envoie pas de force vers le réseau fixe (voir chapitre II.4.2).

Convergence locale

Les pas d'apprentissage sont fixés à $2\ 10^{-5}$. La raideur de couplage est de *0,1*, tandis que la viscosité de couplage est nulle. Le seuil sur la *DRM* mesurée en cours d'apprentissage pour déclarer l'apprentissage réussi est de *0,7*.

Les résultats sont les suivants :

Nombre de présentations de la séquence									
213	2	106	77	33	29	24	20	172	10
75	176	39	11	4	61	81	29	8	5

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
20/20	59	2	213	61

La *DRM* après apprentissage, moyennée sur les vingt essais est de *0,988*, qui est effectivement inférieur à *1*.

Test difficile

Les pas sont fixés à $2\ 10^{-4}$, la raideur de couplage à *0,05* et la viscosité de couplage à *0,5*. Le seuil sur la *DRM* mesurée en cours d'apprentissage pour déclarer l'apprentissage réussi est de *1,5*.

On obtient les tableaux de résultats suivants :

Nombre de présentations de la séquence									
201	571	14	*	38	846	*	370	16	434
31	169	103	*	99	86	208	*	*	*

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
14/20	228	14	846	237

La *DRM* après apprentissage, moyennée sur les *14* essais réussis est de *4,46* qui est effectivement inférieur à *5*.

Le semi-couplage donne des résultats comparables au couplage.

II.5.2.e Algorithme de couplage-découplage

Il s'agit de l'algorithme consistant à dupliquer le réseau, l'un couplé, et l'autre libre (voir chapitre II.4.3). Seule la version *locale* a été testée, car c'est la seule qui est intéressante du point de vue computationnel. Le réseau couplé est en fait "semi-couplé".

Convergence locale

Les pas d'apprentissage sont fixés à $1,4\ 10^{-4}$. La raideur de couplage est nulle, tandis que la viscosité de couplage vaut *1,5*.

Les résultats sont les suivants :

Nombre de présentations de la séquence									
53	21	39	17	63	10	12	16	15	76
25	178	10	29	3	21	7	32	61	9

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
20/20	35	3	178	39

Bien qu'il s'agisse d'un algorithme qui ne suive pas la direction exacte du gradient en négligeant certains termes (voir paragraphe II.4.3), l'algorithme converge localement.

Test difficile

Les pas sont fixés à $2\ 10^{-5}$. La raideur de couplage a été fixée à *0,5* et la viscosité de couplage à *0,3*.

On obtient les tableaux de résultats suivants :

Nombre de présentations de la séquence									
*	436	*	38	154	*	*	*	114	61
*	252	*	*	*	*	93	44	*	*

	Nombre de présentations de la séquence			
Tx de réussite	Moyenne	Minimum	Maximum	Ecart type
8/20	149	38	436	237

Comparativement aux autres algorithmes, l'algorithme de couplage-découplage est peu performant au niveau du taux de réussite. Celui-ci est du même ordre que pour l'algorithme *RTRL*.

II.5.2.f Forçage

Pour comparer avec les algorithmes de couplage, nous avons aussi testé l'algorithme de forçage (algorithme de "teacher forcing" décrit dans [Williams & Zipser 89ab], voir paragraphe II.4.2).

Les résultats sont décevants : même pour la convergence locale, l'apprentissage échoue. Avec un pas de 10^{-3}, on obtient les résultats suivants :

- en fixant le seuil sur la *DRM* à *0,2*, seuls *4* réseaux sur *20* convergent, mais la *DRM* testée sur le réseau libre est de *2,2*, en moyenne, ce qui dépasse le chiffre *1* requis pour le test de convergence locale ;
- en diminuant le seuil sur la *DRM* de *0,2* à *0,1*, alors plus aucun réseau ne converge.

En diminuant la taille des intervalles de paramètres, à savoir [1,5 ; 1,55] pour les masses, [0,1 ; 0,12] pour les raideurs et [0,02 ; 0,05] pour les viscosités, ce qui correspond à une *DRM* qui vaut en moyenne *2,81* sur tous les réseaux avant apprentissage, on ne parvient pas non plus à obtenir *20* succès sur *20*.

Le test difficile se solde par un échec également.

II.5.2.h Bilan des résultats

Tous les algorithmes sauf l'algorithme de forçage ont passé avec succès le test de convergence locale.

Tous les algorithmes sauf l'algorithme de forçage ont réussi le test difficile, puisque le taux de réussite est au moins supérieur à 8/20.

Pour ces cinq algorithmes, les résultats sont les suivants :

	RTRL	BPTT	Couplage	semi-couplage	couplage - découplage
Taux de réussite	9/20	15/20	15/20	14/20	8/20
Nb moyen de présentations	267	95	139	228	149

L'algorithme le plus efficace est l'algorithme de Rétropropagation dans le temps. Les algorithmes de couplage et semi-couplage donnent aussi de bons taux de réussite,

mais nécessitent plus de présentations. L'algorithme de couplage nécessite sensiblement moins de présentations que l'algorithme de semi-couplage.

Les résultats de l'algorithme de couplage/découplage sont moyens, mais nous estimons que le fait que son taux de réussite soit du même ordre que le taux de réussite de *RTRL* et que le nombre de présentations soit faible est très encourageant, dans la mesure où il s'agit d'un algorithme dans lequel certains termes ont été négligés.

II.5.3 Tests sur une longue séquence

Sachant, grâce aux expériences précédentes, qu'un certain nombre d'algorithmes d'apprentissage parviennent à adapter leurs paramètres afin de reproduire un comportement temporel donné, il convient maintenant de se placer dans une configuration plus réaliste par rapport à la situation réelle, c'est-à-dire dans le cas où le réseau adaptatif et le réseau fixe (représentant l'objet à identifier) interagissent de manière continue, sur *une seule longue séquence*. La notion de "nombre de présentations" n'est plus valide dans ce cadre, et se trouve remplacée par le nombre d'adaptation effectué, c'est-à-dire la longueur de la séquence entière.

Afin de conserver une certaine similitude avec les expériences précédentes, on choisit les paramètres suivants :

- la longueur maximale de la séquence, à partir de laquelle on considère que l'apprentissage a échoué est fixé à *50000* : il y a donc autant d'adaptations possibles que dans les expériences précédentes, où une séquence de longueur *50* était répétée jusqu'à *1000* fois ;
- la valeur de M utilisée pour calculer la *DRM* selon (54) a été fixée à *50*. Pour évaluer la *DRM* après apprentissage, on simule uniquement les M premiers échantillons.
- l'opérateur est simulé en envoyant une impulsion unitaire tous les *50* échantillons.

Cependant, comme il va être détaillé ci-dessous, certains paramètres des tests ont été changé au vu des premiers résultats.

Seuls ont été expérimentés les algorithmes qui se sont avérés les plus efficaces pour le test d'apprentissage de séquences courtes : *RTRL*, *BPTT* (version par blocs), couplage et semi-couplage.

Le test difficile (voir paragraphe II.5.2), a été légèrement modifié. Pour des séquences courtes, on étudie le cas où les amortissements sont nuls ; or pour les

séquences longues, cela pose un problème car les réseaux étant excités plusieurs fois (tous les *50* échantillons), on peut avoir des amplitudes qui deviennent de plus en plus grandes. Ce phénomène se manifeste en particulier du fait de la périodicité de l'excitation : un phénomène de résonance peut avoir lieu avec l'un des deux réseaux, rendant l'identification des paramètres délicate. En conséquence, nous avons ajouté un léger amortissement de *0,01*, qui, sur *50* échantillons amortit peu le signal, comme l'illustre la figure II.26. Avec ce nouveau test, la *DRM* avant apprentissage vaut *20,543549* en moyenne sur les vingt essais.

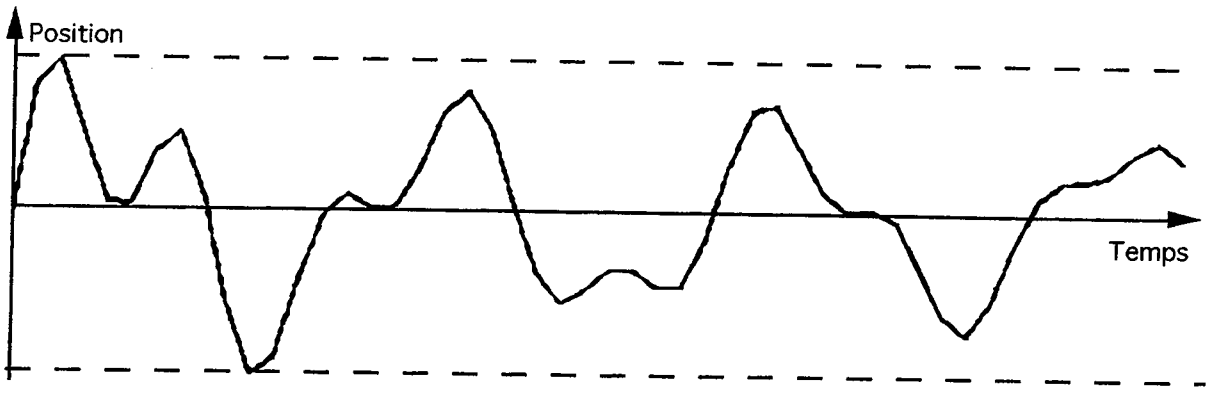

Figure II.26. Effet d'un amortissement de 0,01.

II.5.3.a Algorithme RTRL

Convergence locale

Nous ne sommes pas parvenus à faire converger le réseau sur le test de convergence locale.

Par exemple, en faisant exécuter les vingt réseaux pendant *50000* itérations (longueur maximale), avec un pas d'apprentissage de $5\ 10^{-4}$, seulement sept réseaux aboutissent à une *DRM* inférieure à *1*, tandis que six autres "explosent".

Afin de s'assurer que cette explosion des paramètres n'étaient pas due à un pas d'apprentissage trop élevé, l'expérience suivante a été réalisée : pour un réseau donné, c'est-à-dire une initialisation donnée, on a mesuré la *DRM* au bout de *50000* adaptations en augmentant progressivement le pas d'apprentissage à chaque mesure. La courbe de la figure II.27 qui reporte ces résultats montre bien, par extrapolation, que le réseau n'atteint pas une *DRM* inférieure à *1* avant de diverger. Les causes de cet échec sont analysées plus loin.

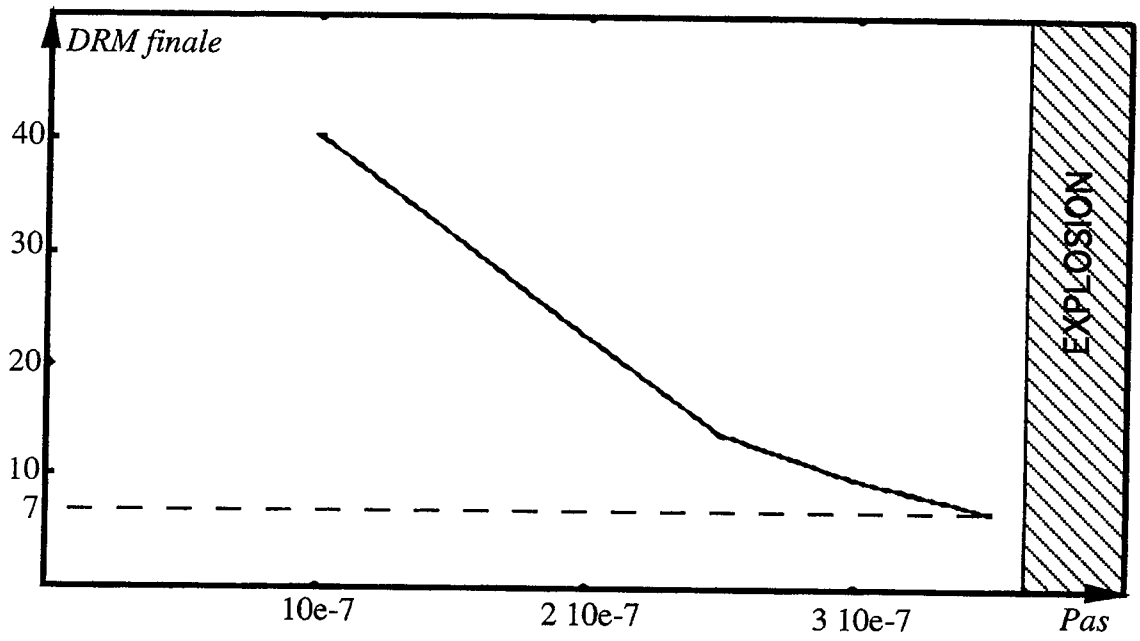

Figure II.27. Problème de convergence locale de RTRL

Au vu de ces résultats très négatifs, nous n'avons pas effectué le "test difficile".

II.5.3.b Algorithme BPTT

Il s'agit de l'algorithme de rétropropagation dans le temps par blocs (voir paragraphe II.2.2.a) appliqué aux réseaux CORDIS (voir chapitre II.3.2).

Convergence locale

Des difficultés ont été rencontrées pour mettre en évidence la convergence locale de l'algorithme.

D'une part, du fait que l'on ne mesure la *DRM* que sur les cinquante derniers échantillons courant, il arrive, au cours de la longue séquence, que ces cinquante échantillons donnent une *DRM* faible, alors que les cinquante suivant ou précédent donnent une *DRM* plus élevée. L'apprentissage peut alors s'arrêter alors que la *DRM* mesurée sur les cinquante premiers échantillons est élevée. Cela amène à utiliser la technique déjà décrite (voir paragraphe II.5.1) du vrai seuil sur la *DRM* inférieur au seuil désiré. Cependant, même avec cette technique, les différences entre les réseaux empêche de fixer ce vrai seuil. Ainsi, sur une même série d'essais, où le vrai seuil sur la *DRM* a été fixé à *0,45*, la valeur finale de la *DRM* (calculée sur les *50* premiers échantillons) va être de *0,57* pour un des réseaux, alors qu'elle sera de *2,15* pour l'autre.

D'autre part, on a pu constater des cas où la *DRM* augmentait au cours de l'apprentissage.

En conséquence, les expériences sont réalisées jusqu'à une longueur temporelle fixe déterminée. Etant données les difficultés de l'algorithme, cette longueur a été portée à *70000*.

Les pas d'apprentissage ont été fixés à $2\ 10^{-3}$. Le paramètre h est de *10* et le paramètre h' est de *3*.

Les résultats sont les suivants :

DRM après apprentissage									
1,89	0,362	1,28	2,33	0,471	0,325	0,688	0,272	1,828	0,673
1,191	1,835	0,834	0,999	0,177	0,515	0,790	0,827	1,454	0,544

DRM finale moyenne	Nombre de réseaux sous le seuil (qui vaut 1)
0,964	13

Donc, si l'on se contente de regarder la moyenne des *DRM*, le test de convergence locale semble positif. Mais si l'on regarde les essais un par un, on constate que *13* d'entre eux seulement sont en dessous du seuil. La moyenne des *DRM* pour les sept autres est de *1,7*. Les performances de l'algorithme sont donc limitées, au niveau de la convergence locale.

Test difficile

Le paramètre h est de *30* et le paramètre h' est de *5*. Les pas d'apprentissage sont fixés à $1,9\ 10^{-4}$. Le seuil sur la *DRM* pour déclarer l'apprentissage réussi est de *2,2*. Le nombre maximum d'itérations est de *50000*.

On obtient les tableaux de résultats suivants :

Nombre d'adaptations						
*	3680	*	*	4093	2892	3228
*	308	1247	1842	5606	*	1326
1536	3813	3839	199	2132	*	

Nombre d'adaptations			
Moyenne	Minimum	Maximum	Ecart type
2596	308	5606	1459

14 réseaux sur *20* ont convergé, selon le critère : $DRM < 2,2$ en cours d'apprentissage. Comme mentionné ci-dessus, ce dernier critère d'arrêt est moins fiable en ce qui concerne les séquences longues. Ainsi, à la fin de l'apprentissage, *15*

essais ont en fait une *DRM* inférieure à *5*, car l'un des réseaux a été simulé jusqu'à *50000* itérations, alors que sa *DRM* était *en dessous* du seuil requis. Après apprentissage, en moyenne, la *DRM* vaut *3,68*. Pour la suite, nous reporterons donc le nombre d'essais dont la *DRM* est inférieure à *5* après apprentissage.

II.5.3.c Algorithme de couplage

Convergence locale

Comme pour l'algorithme de rétropropagation dans le temps, les expériences ont été réalisées sur *70000* itérations.

Les pas d'apprentissage ont été fixés à $5\ 10^{-4}$. La raideur de couplage vaut *0,7* et la viscosité de couplage *0,2*.

Les résultats sont les suivants :

DRM après apprentissage									
0,740	0,237	0,683	1,046	0,462	0,374	0,261	0,244	0,639	0,281
0,664	0,615	0,344	0,810	0,308	0,797	0,628	0,969	0,745	0,348

DRM finale moyenne	Nb essais sous le seuil (qui vaut 1)
0,560	19

Le test de convergence locale n'a pas été non plus parfaitement passé par l'algorithme de couplage sur les longues séquences, puisque pour un des essais, la *DRM* est supérieure à *1*. Nous sommes parvenus à trouver un jeu de paramètres (raideur et viscosité du couplage, pas d'apprentissage) tel que la *DRM* pour cet essai descende sous *1*, mais dans ce cas, c'était un autre essai qui échouait.

On notera cependant que l'essai échoue de peu, puisque la *DRM* n'est que de *1,046*.

Test difficile

Les expériences ont été réalisées avec un seuil sur la *DRM* de *0,4* et jusqu'à un maximum de *50000* itérations.

La raideur de couplage vaut *0,1 et* la viscosité de couplage *0,5*. Les pas d'apprentissage sont fixés à $5\ 10^{-4}$.

On obtient les tableaux de résultats suivants :

Nombre d'adaptations						
13323	38352	1537	11905	3007	7437	40675
25304	1669	21322	*	*	*	*
8063	17359	25773	*	*	*	

Nombre d'adaptations			
Moyenne	Minimum	Maximum	Ecart type
16594	1537	40675	12587

Nb succès	Nb essais sous le seuil (qui vaut 5)	*DRM* finale moyenne
13	14	3,2

L'algorithme de couplage fonctionne donc correctement dans le cas des longues séquences.

II.5.3.d Algorithme de semi-couplage

Convergence locale

Les expériences ont été réalisées sur *70000* itérations.

Les pas d'apprentissage sont fixés à 10^{-4}. La raideur de couplage vaut *0,6* et la viscosité de couplage *0,2*.

Les résultats sont les suivants :

DRM après apprentissage									
0,890	0,430	0,972	0,794	0,669	0,386	0,442	0,217	0,662	0,452
0,852	0,978	0,53	0,881	0,403	0,959	0,810	1,19	0,827	0,522

DRM finale moyenne	Nb essais sous le seuil (qui vaut 1)
0,694	19

Encore une fois, pour un des essais, la *DRM* n'est pas descendue sous *1*, puisque qu'après *70000* adaptations, elle est à *1,19*.

Test difficile

Les expériences ont été réalisées avec un seuil sur la *DRM* de *0,7* et jusqu'à un maximum de *50000* itérations.

La raideur de couplage est nulle et la viscosité de couplage vaut *0,5*. Les pas d'apprentissage sont fixés à $2\ 10^{-4}$.

On obtient les tableaux de résultats suivants :

Nombre d'adaptations						
14375	*	3003	*	18661	*	*
*	4471	31654	*	*	Explo	*
10246	17855	16431	44152	*	*	

Nombre d'adaptations			
Moyenne	Minimum	Maximum	Ecart type
17872	3003	44152	12267

Nb succès	Nb réseaux sous le seuil (qui vaut 5)	*DRM* finale moyenne
9	15	2,86

Pour cet algorithme apparaît clairement le fait que le critère de mesure de la *DRM* en cours d'apprentissage n'est pas valide, puisqu'il donne *9* succès alors que *15* réseaux on finalement un bon résultat.

II.5.4 Discussion

Généralités

Cette série d'expériences a permis d'évaluer les capacités d'apprentissage des réseaux de modélisation physique adaptatifs. Dans l'ensemble, on a pu montrer que l'apprentissage était possible dans ce type de réseau. En effet, mis à part l'algorithme de forçage, la convergence locale sur des petites séquence est assurée, et tous les algorithmes ont un taux de réussite non négligeable en ce qui concerne le test difficile sur les courtes séquences (entre 8/20 et 15/20 pour le test en question).

Pour ce qui concerne les longues séquences, un certain nombre d'algorithmes parviennent à trouver de bons paramètres, ce qui est un résultat primordial pour une implémentation future.

On a pu constater certaines difficultés concernant le critère d'arrêt de l'apprentissage. Pour les algorithmes de couplage, le critère d'arrêt ne peut se définir sans essais préalables, ce qui est assez et coûteux et imprécis. On perçoit une relation entre les paramètres de couplage et ce seuil : plus le couplage est fort, plus les deux réseaux ont un comportement proche pendant l'apprentissage même si les comportements libres diffèrent, donc plus le seuil doit être faible. Pour les longues séquences, le critère d'arrêt est lié à un facteur de hasard, car les deux réseaux peuvent au milieu de la séquence avoir *50* échantillons proches, ce qui arrête l'apprentissage.

Une solution pour éviter ce problème est d'augmenter la valeur de M dans le calcul de la *DRM*.

Difficultés de l'algorithme *RTRL*

L'examen des résultats met clairement en évidence les limites de l'algorithme *RTRL*. Sur les séquences courtes, on observe *9* succès sur *20*, contre *14* ou *15* pour les algorithmes de rétropropagation dans le temps, de couplage et de semi-couplage. Au niveau du nombre de présentations, *267* présentations sont en moyenne nécessaires, contre *95* pour la rétropropagation dans le temps. Enfin et surtout, pour les longues séquences, l'algorithme *RTRL* échoue.

En particulier, les cas d'échecs sont des explosions des paramètres et des variables du réseau (positions et forces). On retrouve à ce niveau une manifestation du problème d'"explosion du gradient" que nous évoquions au paragraphe II.2.3.d. Les réseaux mécaniques sont d'autant plus sensibles à ce problèmes que leurs variables ne sont pas bornées, contrairement à un grand nombre de réseaux neuronaux classiques.

De bons résultats pour la rétropropagation dans le temps

Sur des séquences fixes, l'algorithme de rétropropagation dans le temps donne de très bons résultats, aussi bien pour le taux de réussite que pour la vitesse de convergence.

Pour les longues séquences, cet algorithme ne souffre pas des problèmes de l'algorithme *RTRL* : il n'y a pas de problèmes d'explosion. Ceci est dû au fait que la rétropropagation est fortement tronquée : l'erreur est rétropropagée sur *30* échantillons, alors que les séquences comprennent parfois plusieurs dizaines de milliers d'échantillons.

L'algorithme a néanmoins des problèmes en ce qui concerne la convergence locale : en exécutant la séquence sur *70000* itérations, seuls *13* réseaux sont parvenus à une *DRM* inférieur à *1*. Les sept autres ont des valeurs ne dépassant pas *2,34*, ce qui demeure acceptable (il n'y a pas de divergence). Ce problème de convergence locale est certainement dû à l'approximation que constitue la troncature de la rétropropagation.

Forçage, couplage et semi-couplage

Les algorithmes de couplage et de semi-couplage améliorent nettement les résultats de l'algorithme *RTRL* dont ils sont dérivés. D'une part, pour les test sur les courtes séquences, ils donnent un taux de succès de *14* ou *15* sur *20*, contre *9* pour

l'algorithme *RTRL*. Ces résultats sont aussi bons que l'algorithme de rétropropagation dans le temps. D'autre part, ils ne souffrent pas du problème des longues séquences : après apprentissage, on trouve à nouveau *14* ou *15* essais dont la *DRM* est en dessous de *5*.

Deux explications peuvent être apportées pour expliquer les bonnes performances sur les longues séquences.

D'une part, le fait de contraindre le réseau dans la bonne trajectoire évite le problème de l'explosion du gradient, comme il a été discuté au paragraphe II.2.4.a, à propos de l'algorithme de forçage ("teacher forcing"). L'algorithme de forçage quant à lui donne de très médiocre résultats, même en ce qui concerne les courtes séquences (chapitre II.5.2.h). On peut supposer que l'algorithme de forçage échoue car il est mal adapté aux réseaux mécaniques utilisés ici. Les algorithmes de couplage et semi-couplage le remplacent avantageusement pour éviter l'explosion du gradient.

D'autre part, on peut constater que les résultats sont souvent obtenus dans le cas d'une viscosité de couplage élevée : pour le test difficile, sur *5* expériences, la viscosité de couplage était de *0.48* en moyenne, alors que la raideur de couplage valait *0.19* en moyenne. Ceci confirme l'idée émise au paragraphe II.4.1 selon laquelle l'amortissement empêche l'explosion du gradient, et améliore les performances. Ce résultat est à rapprocher de l'étude de [Catfolis 93], où les termes de sensibilité (dérivées des signaux par rapport aux paramètres) sont périodiquement mis à zéro, variante qui améliore les performances de *RTRL*.

Enfin, les algorithmes de couplage et semi-couplage ne souffrent pas du problème de convergence locale de l'algorithme de rétropropagation dans le temps. En effet, dans les deux cas, sur *20* essais, *19* réseaux ont une erreur inférieure à *1* après *70000* itérations (contre *13* pour la rétropropagation dans les temps).

Comparaison entre le couplage et le semi-couplage

Il convient de rappeler que la seule différence algorithmique entre les algorithmes de couplage et de semi-couplage réside dans la présence ou l'absence respective de forces envoyées par le réseau adaptatif vers l'objet à identifier (le réseau fixe dans les expériences ci-dessus) au niveau des points d'accès. La comparaison des deux algorithmes s'avère donc primordiale pour évaluer le rôle de cette *action* du réseau adaptatif vers l'environnement.

Au niveau des taux de réussite, les deux algorithmes ont des résultats similaires. L'action du ressort de couplage ne permet donc pas d'augmenter les chances de succès de l'algorithme.

L'algorithme de couplage converge plus rapidement que l'algorithme de semi-couplage dans le cas de courtes séquences, alors que l'inverse est observé pour les longues séquences ; il n'est donc pas possible de tirer de conclusion en ce qui concerne les rapidité relatives des deux algorithmes.

Il s'avère donc que l'utilité de l'action dans l'apprentissage n'apparaît *pas* à ce niveau. L'efficacité de l'algorithme de couplage est plutôt dû au fait que le réseau adaptatif est *guidé* par le comportement désiré.

L'algorithme de couplage-découplage

L'algorithme de couplage-découplage a passé avec succès le test de convergence locale. Le taux de réussite pour le test difficile (*8/20*) est du même ordre que celui de l'algorithme *RTRL* (*9/20*). L'approximation qui est faite dans l'algorithme semble donc pertinente. Cependant, le taux de réussite demeure relativement faible, comparativement aux performances obtenues par les autres algorithmes.

La rapidité de convergence est quant à elle tout à fait satisfaisante, puisque l'apprentissage nécessite une moyenne de *149* présentations de la séquence.

L'idée d'un critère d'erreur local est donc à approfondir.

Quels algorithmes choisir ?

Il ressort de cette série d'expérimentations que les algorithmes efficaces sont la rétropropagation dans le temps par blocs et les algorithmes de couplage et semi-couplage. Au niveau du coût computationnel, la rétropropagation dans le temps est de loin l'algorithme le meilleur, mais il présente des difficultés au niveau de la convergence locale. Les algorithmes de couplages et de semi-couplage n'ont pas ce problème.

Dans le cas d'une implémentation parallèle, l'algorithme de couplage est intéressant, car tous les réseaux d'adaptation peuvent être simulés en parallèle. Dans le cas contraire, on aura intérêt à l'utiliser pour des structures moins complexes, ayant de l'ordre de N paramètres adaptables pour N masses, et non N^2 comme c'est le cas pour les réseaux expérimentés jusqu'à présent.

Pour le réglage des paramètres d'apprentissage, tous les algorithmes dépendent des pas d'apprentissage, et possèdent en plus deux paramètres : h et h' pour la rétropropagation dans le temps, et les raideurs et viscosités de couplage dans les deux autres algorithmes. Contrairement aux paramètres h et h', les raideurs et viscosités de couplage ne changent pas le coût computationnel de l'algorithme.

Il est intéressant de noter que les essais qui échouent ne sont pas les mêmes entre les algorithmes. Si l'on observe les tableaux de résultats, on se rend compte que pour les séquences courtes, aucun essai n'a échoué à la fois pour les trois algorithmes, et un seul à la fois pour l'algorithme de rétropropagation dans le temps et l'algorithme de couplage. Pour les séquences longues, seulement deux essais ont échoué à la fois pour l'algorithme de rétropropagation dans le temps et l'algorithme de couplage. Ceci s'explique certainement par la nature très différente de ces deux algorithmes. Il est donc éventuellement envisageable, dans un futur système, d'intégrer la rétropropagation dans le temps et l'algorithme de couplage en tant que deux outils pouvant être utilisés alternativement pour déterminer les paramètres.

Limites des résultats

Bien que les taux de réussite soient bons (15/20 par exemple), il faut rappeler que dans nos expériences, ce taux a peu de valeur en absolu, dans la mesure où il dépend de la taille des intervalles dans lesquels ont été choisis les paramètres. Il convient donc, afin que nos résultats correspondent mieux à la situation réelle où les comportements initiaux peuvent être très différents, d'augmenter encore le taux de réussite, ou bien d'obtenir le même taux avec des intervalles plus grands. Pour aller dans ce sens, plusieurs possibilités sont classiquement proposées comme l'utilisation d'algorithmes d'optimisation plus puissants que le simple gradient à pas fixe ou l'application d'un algorithme permettant d'éviter le problème des minima locaux (recuit simulé par exemple). Ce type de techniques est généralement lourd en calculs, et convient mal à l'apprentissage en temps réel et rapide que nous envisageons. C'est l'une des raisons qui nous a amené à étudier une autre voie pour améliorer les taux de réussite, à savoir l'apprentissage actif (partie III).

Un autre aspect mérite d'être mentionné concernant la méthode expérimentale utilisée. Comme il a été expliqué au paragraphe II.5.1, à chacun des vingt essais, on modifie non seulement l'initialisation des poids du réseau adaptatif, mais aussi les paramètres du réseau fixe. En conséquence, si un de ces essais échoue, on ne sait pas si les paramètres du réseau fixe peuvent être appris avec une autre initialisation des poids du réseau adaptatif, ou si l'apprentissage échoue pour la plupart des initialisations, ce qui serait problématique. Pour avoir cette information, il serait

nécessaire de choisir aléatoirement vingt réseaux fixes, et pour chacun de ces vingt réseaux, de tester vingt initialisations aléatoires des poids. Le grand nombre de simulation que cette méthode entraîne justifie qu'elle n'ait pas été employée. Dans le futur, pour un algorithme en particulier dont on connaît déjà les performances, ce test devra néanmoins être réalisé.

*

Dans cette partie ont été mis en correspondance les réseaux de modélisation mécanique et les réseaux de neurones récurrents, ce qui a permis à la fois d'appliquer les techniques des Réseaux de Neurones Formels aux réseaux de modélisation physique (algorithmes *BPTT* et *RTRL*), et d'utiliser certaines contraintes mécanique pour la conception de nouveaux algorithmes (algorithmes de couplage, de semi-couplage et de couplage-découplage). Les expériences de simulation ont délivré des résultats positifs qui montrent l'intérêt des deux démarches, et invitent à approfondir les liens entre ces deux champs de recherche.

TROISIÈME PARTIE

APPRENTISSAGE ACTIF

L'utilisation des réseaux connexionnistes dans le cadre de la simulation de systèmes mécaniques nous a amené naturellement vers le concept d'interaction entre le système qui apprend et son environnement.

Les expériences relatées au chapitre II.5 montrent que la modélisation de cette interaction au niveau physique le plus simple, c'est-à-dire par un "ressort", ne permet pas de mettre en évidence un effet très tranché de l'action dans l'apprentissage.

C'est pourquoi on se propose d'aborder ce problème d'une part à un plus haut niveau que le niveau mécanique et d'autre part selon un point de vue interdisciplinaire.

Précisons que l'objectif n'est pas d'étudier l'action en général : la diversité de définitions du terme dans les sciences cognitives rendrait cette approche tout à fait périlleuse. Nous abordons donc exclusivement *l'action pour apprendre,* que nous définissons comme suit : *étant donné un système plongé dans son environnement, l'action pour apprendre consiste en l'envoi d'une action spécifique vers l'environnement pour contrôler et modifier les données entrant dans le système, et ce dans le but d'améliorer le déroulement de l'apprentissage.*

Si cette problématique de l'action pour apprendre est relativement peu étudiée en connexionnisme, elle semble en revanche largement présente en psychologie de l'apprentissage, didactique, apprentissage moteur, résolution de problèmes, biologie, etc.

Par conséquent, un vaste champ de disciplines est concerné. Nous avons choisi de restreindre notre étude aux Réseaux de Neurones Formels (chapitre III.1) et à quelques champs de la Psychologie Cognitive (chapitre III.2).

Cela nous amènera à exposer quelques expériences relatives à l'apprentissage actif (chapitre III.3) et enfin à tracer les grandes lignes du système d'apprentissage envisagé dans le cadre de l'apprentissage des modèles physiques (chapitre III.4).

III.1 Agir pour apprendre dans les réseaux connexionnistes

III.1.1 Généralités

Un réseau supervisé reçoit pendant l'apprentissage des couples (entrée , sortie). Ceux-ci sont peuvent provenir de l'environnement de différentes manières. Selon les cas, l'apprentissage actif prend diverses formes.

Dans un certain nombre de cas, le réseau est directement relié à l'environnement : un générateur d'entrées envoie des entrées aléatoires, ou balayant systématiquement l'environnement, au réseau et à l'environnement : l'environnement renvoie alors une sortie qui est la sortie désirée pour le réseau. L'action pour apprendre consiste à faire générer les actions par le système connexionniste, de manière à faciliter l'apprentissage (voir figure III.1). Les système connexionniste agit alors *directement* vers l'environnement. Les réseaux CORDIS adaptatifs étudiés au chapitre II relèvent de ce cas de figure.

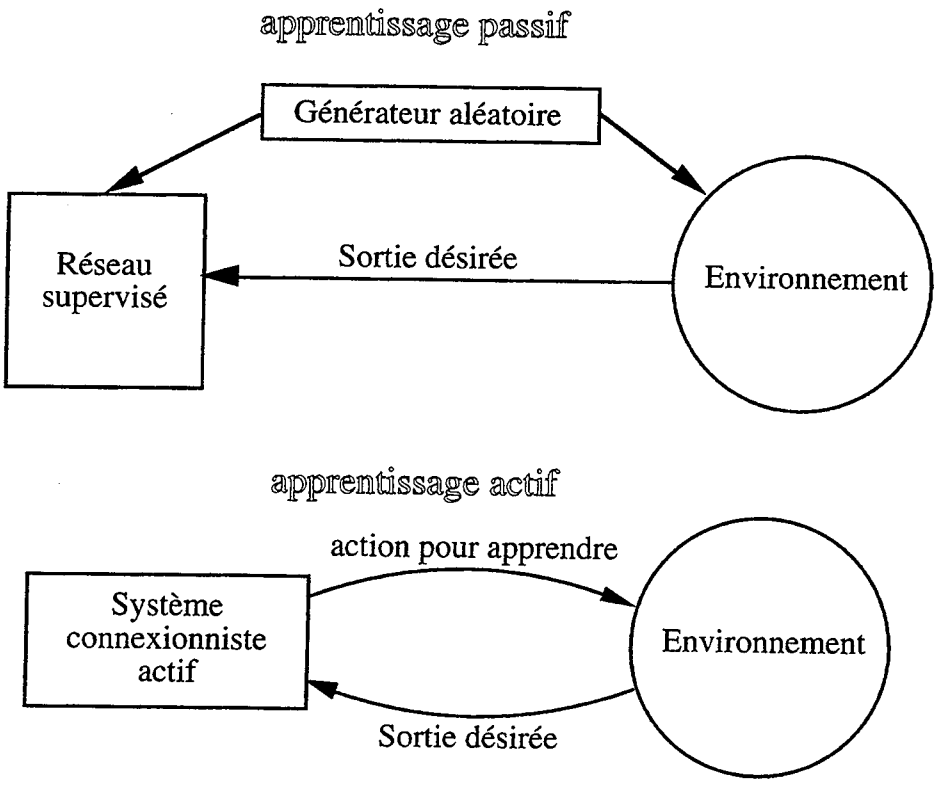

Figure III.1. action pour apprendre directe

Dans de nombreux autres cas, le réseau apprend d'après une *base d'exemples* (ou ensemble d'apprentissage) : on a au préalable collecté des couples (entrée , sortie) de l'environnement pour construire cette base, qui est ensuite donnée au réseau. Dans ce cas, l'action pour apprendre consiste à *sélectionner* dans la base quels exemples apprendre à chaque instant de l'apprentissage. Il s'agit alors d'une action indirecte (voir figure III.2). On peut considérer, par extension, que la base d'exemples constitue l'environnement du réseau, que l'on nomme *environnement proche* pour faire la distinction avec l'environnement lui-même.

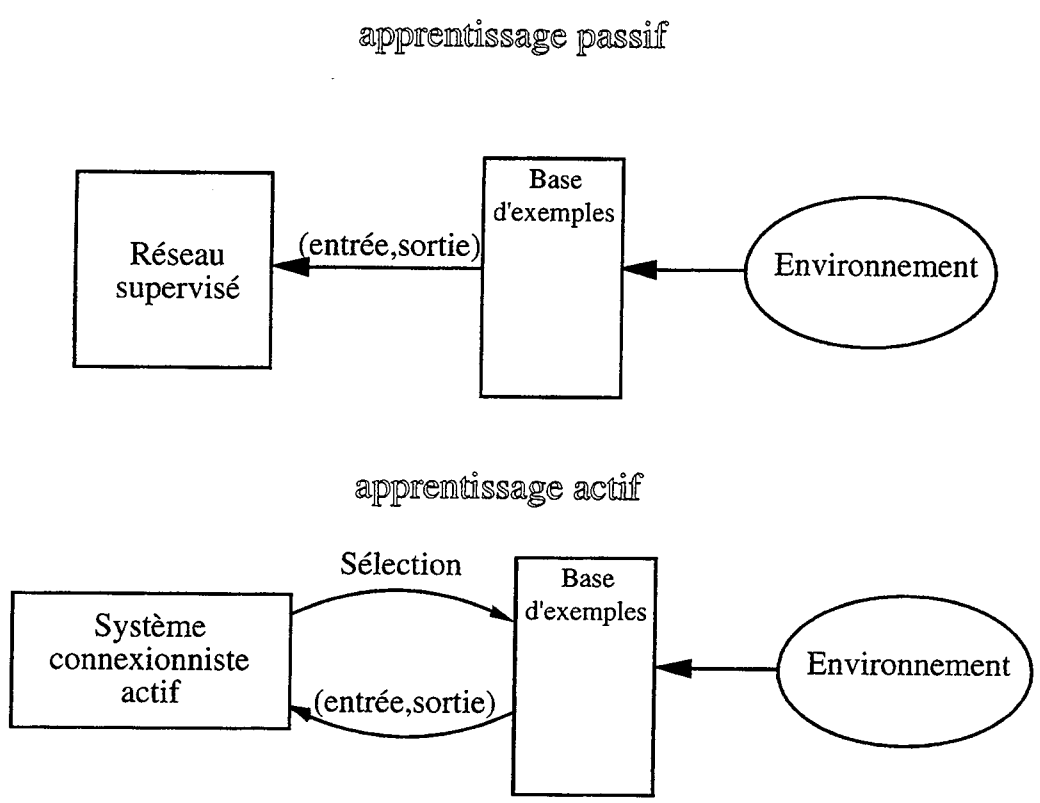

Figure III.2. action pour apprendre indirecte

De plus, nous nous intéresserons aussi au cas où ce n'est pas le système informatique lui même qui agit mais le concepteur humain, qui choisit quelles données doivent être successivement présentées au réseau, comme illustré sur la figure III.3.

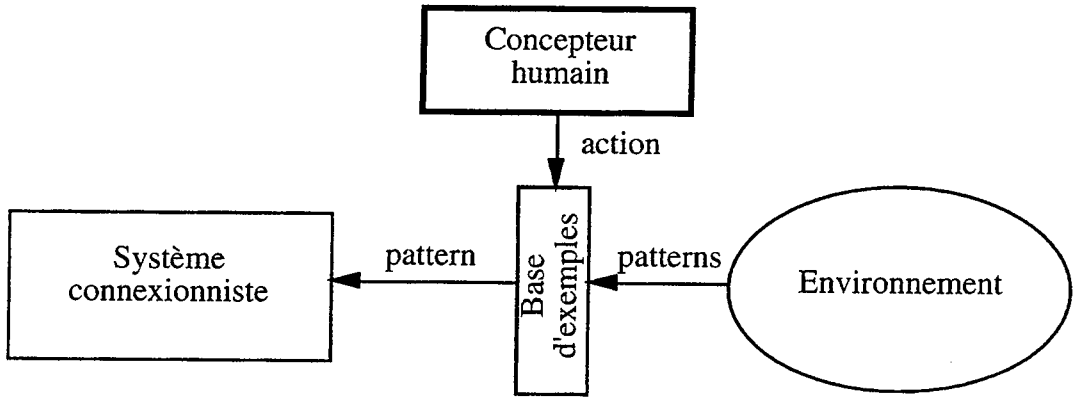

Figure III.3. Intervention du concepteur humain

Nous nous intéressons donc, de manière générale, à tout ce qui concerne la manipulation de l'environnement dans les réseaux connexionnistes. La question qui se pose est alors : étant donné un réseau, quel est l'effet du choix des exemples présentés au réseau à chaque instant, et comment contrôler cet effet.

III.1.2 Etat de l'art

Ce chapitre se restreint à l'étude des réseaux supervisés.

Parmi la grande quantité de publications concernant les réseaux de neurones, très peu se penchent sur l'environnement du réseau, à savoir les exemples à apprendre.

Ainsi, l'utilisation classique d'un réseau à couches à rétropropagation du gradient consiste à prendre un ensemble d'apprentissage donné fixe, et à le présenter au réseau, soit d'un bloc, soit par sélection aléatoire ("stochastic backpropagation"). Dans le premier cas, on adapte le réseau après présentation de l'ensemble d'apprentissage en entier et l'ordre n'a aucune influence ; dans le deuxième cas, l'adaptation s'effectue à chaque présentation des exemples, mais tout effet d'ordre tend a être gommé par le fait que les exemples sont tirés aléatoirement.

Dans ce contexte, la seule variable de l'environnement qui peut être manipulée est le *nombre* d'exemples à apprendre [Baum & Haussler 89], ce qui est tout à fait réducteur.

Même si l'influence de l'ensemble d'apprentissage n'est jamais niée, ce facteur n'est généralement pas étudié [Latimer 94][Szilas 94]. Parmi les raisons de cette "lacune", citons la difficulté à réaliser des expériences systématiques sur le sujet [Hrycej 94 p. 179] (une base de dix exemples peut être présentée dans 10!=3628800 ordres différents !) et le peu de données théoriques sur le sujet. Néanmoins, un certain nombre de travaux traitent cet aspect des réseaux de neurones. Ils se divisent pour la plupart en deux catégories, selon qu'il s'agit de maximiser l'information en provenance de l'environnement ou de graduer la difficulté de la tâche. Seule l'étude de [Murre 92], qui étudie comment la présentation des exemples peut éviter le problème de l'oubli dans les réseaux à couches, n'entre pas dans l'une de ces deux catégories.

III.1.2.a Action informative

Nous regroupons dans une même catégorie un grand nombre d'études pour lesquelles les exemples sont sélectionnés dans le but de maximiser l'information apportée au réseau. Dans ce qui suit sont présentés les motivations pour sélectionner les exemples informatifs, comment se déroule la session d'apprentissage, puis les critères utilisés pour sélectionner les exemples et enfin le critère de passage d'un exemple (ou d'un sous-ensemble d'apprentissage) au suivant.

Pourquoi sélectionner les exemples les plus informatifs ?

Il y a plusieurs réponses, qui sont autant de motivations pour étudier ce type d'apprentissage actif.

Premièrement, dans un ensemble d'apprentissage donné (fini ou infini), il y une certaine *redondance* : deux éléments peuvent contenir la même information, ou une information proche [Strand & Jones 92]. Par exemple, dans le cas extrême, faire apprendre à un réseau cinquante fois le même exemple est une perte de temps : on réapprend toujours la même chose ; au contraire, choisir cinquante exemples *significatifs* de la base d'exemples est plus efficace. L'apprentissage actif informatif permet donc d'augmenter la rapidité de convergence [Allred & Kelly 89][Atlas *et al.* 90][Strand & Jones 92][Plutowski & White 93][Cachin 94]. En comparant l'apprentissage actif avec l'apprentissage passif, on obtient des courbes d'apprentissage du type de celles représentées sur la figure III.4.

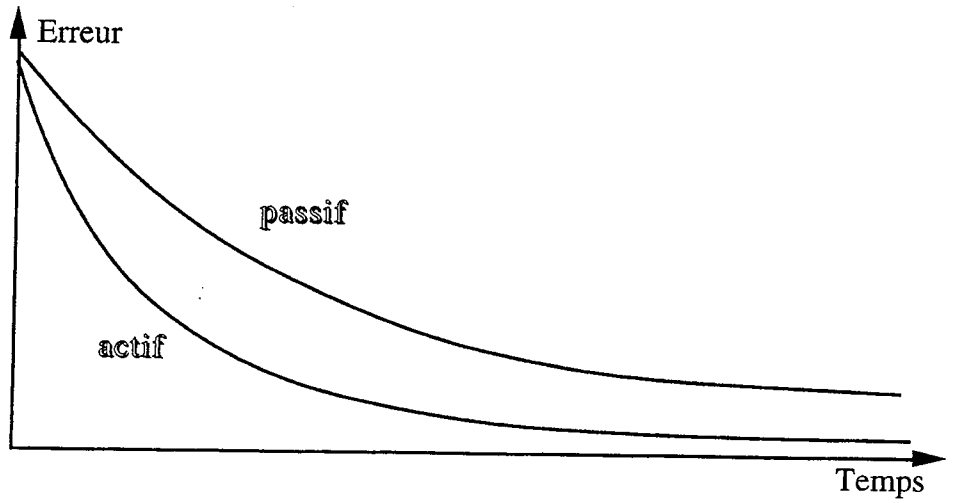

Figure III.4. Courbes d'apprentissages dans le cas passif et dans le cas actif

En corollaire, sélectionner les exemples les plus informatifs permet de diminuer l'erreur finale de la solution obtenue [Sanzogni & Vaccaro 93][Hwang *et al.* 91]. En effet, sans prolonger l'apprentissage excessivement, l'apprentissage actif converge plus vite vers la solution, et donne donc une erreur plus faible (voir figure III.4).

Deuxièmement, sélectionner les exemples les plus informatifs peut aider à trouver une meilleure solution du point de vue de la généralisation [Biehl & Schwarze 92][Strand & Jones 92][Röbel 94][Zhang 94][RayChaudhuri & Hamey 95][Krogh & Vedelsby 95]. En effet, toujours en se plaçant dans une situation exagérée, apprendre toujours le même exemple va conduire à une solution spécifique de cet exemple, solution qui généralisera très mal sur d'autres régions de l'ensemble

d'apprentissage. Ainsi, la sélection active permet par exemple de corriger une base d'apprentissage dont les exemples ne sont pas répartis uniformément.

Troisièmement, dans certaines applications, obtenir un exemple d'apprentissage peut être assez coûteux, quand un procédé de mesure complexe est mis en jeu [Atlas *et al.* 90][Hwang *et al.* 91][RayChaudhuri & Hamey 95]. Dans ce cas, on aura intérêt à échantillonner l'environnement le moins possible et donc de choisir à chaque fois l'exemple le plus utile.

Quatrièmement, on peut envisager l'apprentissage actif informatif comme le moyen de sortir d'un minimum local : une fois le réseau stabilisé, en apprenant davantage les exemples mal appris, on déforme la surface d'erreur et on peut ainsi sortir du minimum local [Le Cun 87].

Comment se déroule une session d'apprentissage actif, selon les différents algorithmes

Dans le cas d'une action indirecte (voir paragraphe III.1.1 et la figure III.2), le réseau dispose déjà d'un ensemble d'apprentissage.

Un premier type d'algorithme va alors consister à construire à l'aide de cet ensemble un sous-ensemble contenant moins d'exemples [Röbel 94][Plutowski & White 93][Zhang 94][Krogh & Vedelsby 95]. Le sous-ensemble est construit de manière incrémentale : les exemples sont ajoutés un à un.

Dans un deuxième type d'algorithme, on balaie *tous* les exemples de l'ensemble mais on *pondère* leur influence : plus un exemple apporte de l'information, plus on l'apprend. Cette pondération peut se faire :

- en décidant, pour chaque exemple si on l'apprend ou pas, ce qui revient à "sauter par dessus" les exemples peu informatifs [Haffner 88][Allred & Kelly 89][Strand & Jones 92] ;
- en tenant plus compte, dans le calcul de l'erreur, de la part de certains exemples [Sanzogni & Vaccaro 93] ;
- en manipulant la *fréquence* de présentation de chaque exemple [Munro 92][Cachin 94], ce qui revient implicitement à pondérer les exemples dans la fonction d'erreur.

De nombreuses autres stratégies de présentations sont exposées dans [Cachin 94].

Dans le cadre de l'action directe sur l'environnement, on parle généralement d'apprentissage par "query" (demande) : on donne à l'environnement (parfois dénommé "oracle" dans ce contexte) un pattern d'entrée (un stimulus) et celui-ci doit

fournir la réponse. Dans [Atlas *et al.* 90], [RayChaudhuri & Hamey 95] et [Hwang *et al.* 91], une telle stratégie est utilisée pour construire un ensemble d'apprentissage de taille réduite. Dans [Biehl & Schwarze 92], à chaque adaptation des poids, le nouvel exemple à apprendre est déterminé par "query".

Différents critères utilisés pour déterminer quel exemple ou quel ensemble d'exemples est le plus informatif

Le critère le plus utilisé est l'*erreur* du réseau : on privilégie l'apprentissage des exemples pour lesquels le réseau donne les moins bon résultats, c'est-à-dire la plus forte erreur [Allred & Kelly 89][Munro 92][Cachin 94][Le Cun 87][Haffner 88][Strand & Jones 92][Röbel 94][Zhang 94]. Ce critère est choisi parce qu'il est simple à calculer et correspond au premier critère qui vient intuitivement à l'esprit. Dans [Zhang 94] cependant, le fait de choisir les exemples dont l'erreur est la plus élevée est obtenu dans le cadre de la théorie de l'information.

Un critère beaucoup plus sophistiqué, et complexe à calculer, a été proposé dans [Plutowski & White 93] ; il consiste à choisir dans l'ensemble d'apprentissage l'exemple qui maximise la décroissance de l'erreur calculée sur tout l'ensemble d'apprentissage. On montre que les exemples sélectionnés de cette manière ne correspondent pas aux exemples ayant une erreur maximale. Dans [Plutowski *et al.* 95], ce critère est comparé avec le critère "erreur" (ainsi qu'avec un sélection aléatoire et une sélection uniforme) : dans une première application, les deux algorithmes se valent, alors que dans une deuxième, le critère "décroissance de l'erreur" est meilleur.

Un autre critère de sélection, spécifique des problèmes de classification, consiste à choisir d'abord les exemples que le réseau ne sait pas classer, c'est-à-dire les entrées qui correspondent à une sortie sur la *frontière* entre deux classes, pour le réseau. Une étude expérimentale [Wann *et al.* 89] montre qu'étant donné un ensemble d'apprentissage, le meilleur sous-ensemble que l'on peut en extraire est celui qui comprend le plus d'exemples frontières. Dans le cas d'un perceptron qui donne *-1* ou *+1* en sortie, on détermine les exemples frontières en choisissant un vecteur d'entrée *orthogonal* au vecteur poids, ce qui donnera une sortie à zéro, donc sur la frontière [Biehl & Schwarze 92]. Dans le cas d'un réseau à couches, on peut *inverser* le réseau [Hwang *et al.* 91], c'est-à-dire trouver par rétropropagation les entrées qui donnent une sortie donnée, à savoir la sortie "frontière" (par exemple *0.5* dans le cas d'une classification entre *0* et *1*). Ce critère est lié à l'erreur, puisque lorsque le réseau donne une sortie à la frontière, celle-ci est nécessairement élevée (mais la réciproque est fausse). Cependant, il est important de noter que, contrairement au critère "erreur", on

détermine les exemples frontières sans connaître la sortie désirée : ce type d'algorithme est donc adapté au cas d'une action directe par "queries".

On peut reprocher au critère "frontière" de ne considérer qu'un sous-ensemble des exemples qu'il faudrait sélectionner [Atlas *et al.* 90]. En effet, l'algorithme de rétropropagation est tel que dans des régions de l'espace d'entrée peu explorées, le réseau va donner une réponse tranchée, sans posséder un quelconque degré de certitude sur cette réponse. Dans [Atlas *et al.* 90], on propose donc de sélectionner les exemples dans une *zone d'incertitude* ; celle-ci est déterminée comme suit : deux réseaux sont entraînés sur les exemples déjà saisis, et des entrées "de fond", aléatoirement choisies dans l'espace d'entrée, pouvant correspondre à des zones non explorées. Le premier réseau associe à ces entrées de fond la première classe, et le deuxième réseau la deuxième classe (a priori, on ne sait pas où les classer). Les entrées incertaines, qui sont donc les "queries" à envoyer vers l'environnement (ou oracle), sont celles pour lesquelles les deux réseaux donnent deux réponses différentes. La rapidité et la généralisation sont grandement améliorées par cette technique, comparativement à une sélection aléatoire.

Une technique très intéressante pour sélectionner les exemples consiste à utiliser plusieurs réseaux entraînés sur plusieurs sous-ensembles [RayChaudhuri & Hamey 95][Krogh & Vedelsby 95]. En mesurant sur chaque exemple le *désaccord* entre ces réseaux, on sélectionne les exemples les plus ambiguës, donc ceux qui apportent le plus d'information. Dans [Krogh & Vedelsby 95], on montre que l'ambiguïté est très liée à l'erreur elle-même : l'erreur globale de tous les réseaux sur une entrée donnée est supérieure ou égale à l'ambiguïté de cette entrée, définie comme la variance entre les valeurs de sorties des différents réseaux. L'intérêt de cette mesure d'ambiguïté est qu'elle n'utilise pas les sorties désirées. On remarquera une légère similitude entre cette approche et celle de [Atlas *et al.* 90], ou l'on sélectionne aussi les exemples selon un désaccord entre des réseaux (*deux* réseaux dans ce cas).

Critère de passage d'un exemple (ou un ensemble d'exemples) au suivant

Dans un certain nombre d'algorithmes, la question ne se pose pas explicitement, puisqu'à chaque adaptation des poids on sélectionne un nouvel exemple [Biehl & Schwarze 92].

Dans [Munro 92] est rapportée une expérience très simple néanmoins instructive : à chaque adaptation des poids, on apprend l'exemple dont l'erreur est la plus forte. Cet algorithme échoue. Cette expérience montre qu'il est nécessaire de répéter plusieurs fois l'exemple sélectionné, jusqu'à faire descendre l'erreur en dessous

d'un certain seuil, avant de sélectionner un autre exemple, et met ainsi en évidence l'importance du critère de passage d'un exemple au suivant.

Dans le même ordre d'idée, des expériences réalisées dans [Cachin 94] montrent que changer trop souvent l'exemple présenté au réseau (en utilisant en particulier un critère de sélection probabiliste) gêne l'apprentissage.

Dans le cas où l'on construit petit à petit un ensemble d'apprentissage, on sélectionne l'exemple suivant quand l'erreur descend en dessous d'un certain seuil [Plutowski & White 93][Zhang 94][RayChaudhuri & Hamey 95][Krogh & Vedelsby 95], sauf dans [Röbel 94] où le critère de passage s'appuie sur le facteur de généralisation de l'ensemble courant.

*

L'apprentissage actif informatif est donc un champ de recherche très "actif", à la fois sur le plan expérimental et théorique. Un certain nombre d'algorithmes ont été développés mais il existe encore peu d'études expérimentales qui comparent les algorithmes (citons tout de même [Cachin 94] et [Plutowski *et al.* 95]). Une étude comparative générale est difficilement réalisable car d'une part les algorithmes s'appliquent dans des contextes différents et d'autre part, dans certains algorithmes, l'augmentation de la taille de l'ensemble d'apprentissage est combinée avec l'augmentation de la taille du réseau [Baum 91][Plutowski & White 93].

Du point de vue cognitif, l'idée de sélectionner les parties les plus informatives de l'environnement est cohérent avec les données psychologiques : sur le plan perceptif, le regard se focalise sur les zones de contraste et de mouvement, c'est-à-dire les zones riches en information ; de même, l'exploration tactile se focalise sur les régions informatives [Hatwell 86]. A un niveau plus élevé, l'apprenant, quand il pose une question, la choisit quand il est incertain de la réponse, et non au hasard.

Ainsi, l'apprentissage humain et l'apprentissage automatique actif partagent la même propriété, en suivant un principe d'économie de traitement : l'exploration exhaustive ou aléatoire est évitée.

III.1.2.b Apprentissage progressif

Parallèlement à tous les travaux sur l'action informative, ont été réalisées quelques études concernant un autre aspect de l'influence sur l'apprentissage de la manière dont sont présentées les données à un réseau. L'idée est de considérer

l'apprentissage comme un processus temporel non seulement au niveau des dynamiques d'activités et de poids mais aussi au niveau de la tâche même que le réseau doit résoudre ; au lieu d'apprendre un ensemble d'apprentissage dans sa globalité et le plus rapidement possible, un réseau peut être amené à apprendre une succession d'ensembles d'apprentissage qui le conduiront, à la fin, à s'attaquer à l'ensemble d'apprentissage cible. Vu de cette manière, il apparaît qu'un véritable apprentissage *progressif* est envisageable, qui permettra soit d'apprendre plus rapidement [Cloete & Ludik 93][Jacobs 88] soit de faire réussir un apprentissage [Elman 93].

Etant donné le petit nombre d'études sur le sujet, nous allons les passer toutes en revue, dans un ordre à la fois chronologique et thématique. Cet état de l'art concerne exclusivement les réseaux supervisés, unidirectionnels ou récurrents, utilisant le gradient de l'erreur pour adapter leurs poids.

Il faut d'abord noter que dès 1987, l'idée d'apprentissage progressif dans les réseaux à rétropropagation du gradient est présente : dans [Le Cun 87 p. 175], on suggère d'utiliser une "pédagogie" consistant à choisir les exemples afin d'avoir au départ une fonction de coût assez lisse, puis d'ajouter ensuite des exemples plus "atypiques" qui complexifient la surface d'erreur. Mais aucune expérience n'est réalisée sur cette idée.

En 1988, un étude expérimentale intéressante a été réalisée sur l'apprentissage progressif dans les réseaux à couches [Wieland & Leighton 88]. Les auteurs ont eu l'idée de s'inspirer de la notion de façonnement ("shaping") utilisée en psychologie expérimentale pour faire apprendre des associations complexes à un animal : l'expérimentateur apprend d'abord à l'animal des associations simples, puis des associations progressivement plus difficiles. Cette idée a été utilisée pour entraîner un réseau à couches à distinguer des images de "A", "B", "C", "D" indépendamment de la rotation, en ajoutant petit à petit des patterns à l'ensemble d'apprentissage : l'apprentissage est quatre fois plus rapide. Malheureusement, la manière précise dont sont choisis les patterns n'est pas explicitée dans [Wieland & Leighton 88], qui n'est qu'un résumé d'une page. Les auteurs n'ont pas poursuivi ce thème de recherche depuis.

Dans [Jacobs 88], des expériences sur deux problèmes simples artificiels (problèmes "jouets") sont réalisées sur un réseau à couches où la tâche est décomposée en plusieurs tâches successives.

Le premier problème consiste en l'apprentissage d'associations entre deux scalaires telles que deux entrées proches donnent deux sorties différentes. On compare les résultats entre l'apprentissage de cette association seule et l'apprentissage de cette association précédée de trois autres associations, de moindre difficulté (entrées moins proches). Les résultats (voir tableau III.2) mesurés sur 25 essais montrent clairement l'avantage de graduer la difficulté.

	Apprentissage normal	Apprentissage progressif
	0,49 -> 0,1 0,51 -> 0,9	0,4 -> 0,1 0,6 -> 0,9
		0,45 -> 0,1 0,55 -> 0,9
		0,48 -> 0,1 0,52 -> 0,9
		0,49 -> 0,1 0,51 -> 0,9
Nombre d'epochs :	18371,7	4447,1

Tableau III.2. Première expérience de [Jacobs 88]

Le deuxième problème consiste à apprendre une fonction $x \rightarrow sin\ 2x$, x faisant partie de $[0\ ,\ 2\pi]$. L'apprentissage progressif consiste à faire apprendre des fonctions intermédiaires (voir tableau III.3) dont la difficulté est moindre car les fonctions sont plus proches de fonctions linéaires.

	Apprentissage normal	Apprentissage progressif
	x->sin(2 x)	x->sin(0,5 x)
		x->sin(x)
		x->sin(1,5 x)
		x->sin(2 x)
Nombre d'epochs :	31185	23977

Tableau III.3. Deuxième expérience de [Jacobs 88]

L'apprentissage progressif donne encore une fois de meilleurs résultats, mais moins tranchés. L'étude de [Jacobs 88] est inspirée de celle de [Selfridge *et al.* 85] qui, sans utiliser la rétropropagation du gradient, porte sur un apprentissage non symbolique. La tâche consiste à contrôler un pendule inversé. En augmentant progressivement la masse du pendule, en diminuant progressivement sa longueur et en diminuant progressivement la longueur du rail de guidage, l'apprentissage est plus rapide.

Notons que R.A. Jacobs n'a pas publié depuis 1988 de nouveaux résultats concernant la stratégie d'apprentissage, s'étant focalisé sur les réseaux modulaires [Jacobs *et al.* 91].

Dans [Elman 91, 93] sont rapportées des études concernant des tâches temporelles utilisant un "Simple Recurrent Network" (voir chapitre II.2.1). Il s'agit d'une tâche de prédiction grammaticale : des phrases générées par une grammaire artificielle sont présentées au réseau, qui doit prédire à chaque instant le mot suivant (plus exactement la probabilité que le mot suivant soit tel ou tel mot). Dans une première expérience, on présente au réseau le corpus en entier : l'apprentissage échoue. Dans une seconde expérience, le corpus est divisé en deux parties : les phrases "simples", et les phrases "complexes" ; la complexité des phrases est directement liée à leur longueur. La proportion relative des phrases complexes par rapport aux phrases simples est augmentée progressivement, selon la stratégie exposée dans le tableau III.4. Avec cette stratégie, le réseau parvient à apprendre cette tâche de prédiction.

N° de la phase	Nb. de phrases simples	Nb. de phrases complexes	Nombre d'epochs
1	10000	0	5
2	7500	2500	5
3	5000	5000	5
4	2500	7500	5
5	0	10000	5

Tableau III.4. Stratégie d'apprentissage de [Elman 91, 93]

Le domaine cognitif dans lequel J.L. Elman se place est celui de l'acquisition du langage. D'après [Elman 93], même si l'adulte adapte son langage quand il parle à l'enfant, l'environnement de l'enfant ne change pas si radicalement que dans le cas de l'expérience décrite ci-dessus. Il montre alors que l'on obtient le même type de résultats en faisant non plus changer l'environnement mais l'algorithme lui-même : au départ l'algorithme est limité en mémoire, puis cette mémoire augmente progressivement.

Une série d'études a été réalisée plus récemment concernant des tâches utilisant aussi le "Simple Recurrent Network" :

- tâches de comptage [Cloete & Ludik 93][Ludik & Cloete 93, 95] : un flux de bits (des "1" ou des "0") arrive au réseau et celui-ci doit donner en sortie le nombre, codé en binaire, de "1" consécutifs.

• tâches d'addition [Cotrell & Tsung 91][Ludik & Cloete 94][Cloete & Ludik 94a] : deux nombres arrivent successivement au réseau qui doit délivrer en sortie le chiffre à inscrire et les actions permettant de *poser* l'opération d'addition (retenue, décalage, etc.).

Une première stratégie appelée *Combined Subset Training (CST)*, qui est la stratégie la plus élémentaire, consiste à doubler la taille de l'ensemble d'apprentissage à chaque étape. Les exemples sont sélectionnés aléatoirement. On passe d'une étape à l'autre quand l'erreur sur l'ensemble d'apprentissage courant est inférieure à un certain seuil. Afin d'éviter que le réseau ne se spécialise dans un sous-ensemble d'apprentissage, ce seuil est diminué à chaque étape, selon une loi de décroissance linéaire. Cette stratégie améliore la vitesse d'apprentissage pour la tâche d'addition. Une variante de cette stratégie, appelée *Incremental Subset Training (IST)* [Cloete & Ludik 94a][Ludik & Cloete 95] consiste non plus à doubler la taille de l'ensemble d'apprentissage mais à augmenter sa taille d'un incrément fixe. Les performances de cette variante sont très bonnes, que ce soit pour la tâche d'addition ou pour une tâche artificielle de classification avec un réseau à couches sur laquelle l'algorithme *IST* a aussi été testé. Dans [Plutowski *et al.* 95] on reporte aussi un effet positif de l'ajout à chaque étape d'un exemple tiré au hasard, dans le cadre d'un réseau incrémental (des unités sont ajoutées pendant l'apprentissage). Enfin, l'algorithme de "Training Set Partition" décrit dans [Cachin 94] est très proche de l'algorithme *IST* : au lieu d'avoir un seuil pour passer d'une partition à l'autre, on effectue ce passage toutes les 500 epochs. L'algorithme de "training Set Partition" s'avère cependant inefficace, dans le cadre des deux applications proposées dans [Cachin 94] : problème de contiguïté et prévision de séries temporelles.

L'algorithme *Incremental Complexity Training (ICT)* [Cloete & Ludik 93] consiste à construire les sous-ensembles successifs selon un ordre de complexité : pour les tâches d'addition, on commence par des nombres à un chiffre, puis on passe à des nombres à deux chiffres, puis enfin à des nombres à trois chiffres. Pour les tâches de comptage, dont nous allons présenter les résultats, on présente d'abord des séquences ne présentant qu'un nombre limité de bits consécutifs ; autrement dit, on apprend d'abord au réseau à compter des petits nombres, puis à compter des nombres de plus en plus grands. L'ensemble est ainsi construit en quatre phases, de complexité croissante. On passe d'une phase à l'autre quand le taux de bonne réponse atteint 100%. Les résultats sont exposés sur le tableau III.5, et comparés avec le cas où l'ensemble d'apprentissage est fixe et avec l'algorithme *CST*.

Nb de patterns	nombre de bits consécutifs	*ensemble fixe*	*ICT*	*CST*
10	0-1		12	95
20	0-3		8	115
40	0-7		14	3
80	0-15		19	69
Total		71	53	282

Tableau III.5. Evaluation de l'algorithme ICT *(voir texte)*

Quelle que soit la stratégie, le problème est résolu, mais on remarque que :

- la stratégie *CST* a ici un effet négatif par rapport à l'apprentissage sans aucune stratégie,
- la stratégie *ICT* améliore le temps d'apprentissage de 25 % par rapport à l'apprentissage sans aucune stratégie.

Une variante est aussi proposée dans [Cloete & Ludik 93] qui consiste à fixer a priori un critère de passage *variable* d'une phase à l'autre, et plus précisément décroissant : l'idée est que l'apprentissage des premiers sous-ensembles ne doit pas être poussé trop loin pour éviter d'aboutir dans une solution trop particulière à ce sous-ensemble. Ainsi, les taux de bonne réponse pour passer d'un sous-ensemble à l'autre sont, dans l'ordre, de 70%, 85% et 85%. L'apprentissage est arrêté quand le taux atteint 100% pour le dernier ensemble. Les résultats sont exposés sur le tableau III.6, et comparés avec l'algorithme *CST* utilisé avec ces mêmes seuils.

Nb de patterns	nombre de bits consécutifs	*ensemble fixe*	*ICT*	*CST*
10	0-1		2	81
20	0-3		6	35
40	0-7		12	24
80	0-15		15	68
Total		71	35	208

Tableau III.6. Evaluation de l'algorithme ICT *avec critères de passage variables (voir texte)*

Les résultats sont encore améliorés, puisque le nombre d'epochs est diminué de 50% par rapport à l'apprentissage classique.

Dans [Ludik & Cloete 94], les mêmes auteurs proposent une nouvelle variante nommée *Incremental Increased Complexity Training (IICT)*. On construit d'abord une suite d'exemples ordonnés selon la difficulté, puis l'ensemble d'apprentissage est

construit de manière incrémentale en prenant les exemples dans l'ordre de cette suite. Les paramètres de l'algorithme sont les suivants :

- distribution des exemples dans la suite (on peut mettre plus ou moins d'exemples faciles) ;
- taille de l'incrément : on peut construire l'ensemble en ajoutant des groupes d'exemples plus ou moins grands ;
- critère de passage : seuils sur l'erreur.

Comme il y a beaucoup plus de phases que dans l'algorithme *ICT*, les seuils sont déterminés automatiquement, selon une loi de décroissance linéaire, comme cela est aussi fait pour l'algorithme *CST* dans [Cotrell & Tsung 91]. Quatre différentes distributions des exemples sont testées, ainsi que plusieurs tailles d'incrément.

On obtient les résultats suivants : l'incrément optimal est *4*, la distribution optimale est celle où il y a le plus d'exemples simples ; dans ce cas, *IICT* est 11% meilleur que *ICT* (avec des seuils fixés à la main). Ce résultat doit être relativisé par le fait qu'en pratique, cet algorithme possède de nombreux paramètres à mettre au point ; par exemple, il n'est pas du tout sûr que sur une autre tâche, l'incrément de *4* soit optimal. En particulier, il semble y avoir une certaine redondance entre la taille de l'incrément et la construction de la suite d'exemples initiale.

Enfin, une technique originale est proposée dans [Cloete & Ludik 94b][Ludik & Cloete 95], qui ne nécessite pas de définir au préalable un ordre de complexité. Il s'agit d'un prétraitement sur les données qui est le suivant : une matrice de similitude, affectant une distance entre tous les exemples est construite. Cette distance se calcule sur les valeurs d'entrée. A l'aide de cette matrice, on peut déterminer les exemples les plus proches de tous les autres, donc plus difficiles à discerner, et les plus éloignés des autres. On obtient ainsi un *classement* des exemples. Ce classement est ensuite utilisé de diverses manières pour dériver trois algorithmes : apprentissage dans l'ordre, dans l'ordre inverse, et en alternant entre les deux extrémités de la liste. Les résultats de cette stratégie sont très positifs, à la fois sur un réseau récurrent (génération de code Morse) que sur un réseau à rétropropagation du gradient (classification de caractères digitaux). Il est cependant difficile, au vu des résultats, de déterminer lequel des trois algorithmes doit être utilisé.

Nous avons fait dans ce paragraphe III.1.2.b le tour des travaux explicitement consacrés à ce que nous appelons l'apprentissage progressif dans les réseaux neuronaux. Néanmoins, il faut compléter ce panorama par un certain nombre d'études qui, sans se consacrer spécifiquement à ce sujet, utilisent une stratégie d'apprentissage qui revient à un apprentissage progressif.

Tout d'abord, pour l'apprentissage de séquences temporelles, une stratégie utilisée consiste à apprendre d'abord une séquence courte, puis à augmenter progressivement la longueur de la séquence. Ainsi [Pearlmutter 89] rapporte que pour l'apprentissage de trajectoires complexes, il est nécessaire d'augmenter progressivement la longueur temporelle de la trajectoire pour faire réussir l'apprentissage. Une telle stratégie est aussi utilisée dans [Toomarian & Barhen 91, 92].

Dans [Fahlman 91], dans une tâche de reconnaissance temporelle du code Morse, un réseau récurrent incrémental apprend plus rapidement si les lettres dont le code Morse est court sont présentées d'abord, puis les lettres plus longues, et ainsi de suite jusqu'à l'apprentissage de tout l'alphabet.

Dans un tout autre domaine, plusieurs études ont testé les capacités des réseaux à reproduire des caractéristiques développementales de l'enfant confronté à la tâche de la balance. Il s'agit de déterminer de quel côté va pencher une balance selon les poids qui sont disposés de chaque côté du balancier, et leurs distances au balancier. Dans [McClelland 89], qui utilise un réseau à couches, on utilise un environnement "biaisé", dans lequel la proportion de problèmes où les deux distances sont les mêmes est plus importante. Dans [Shultz 94], cette expérience est répliquée, mais en utilisant un algorithme incrémental, à savoir l'algorithme de "Cascade Correlation" [Fahlman & Lebiere 91], et en augmentant continûment la taille de l'ensemble d'apprentissage au cours de l'apprentissage. [Shultz 94] rapporte que si on présente l'ensemble dans sa totalité, l'apprentissage devient très difficile. Même si une justification cognitive est donnée en faveur du biais de l'environnement (les enfant ont plus l'occasion de manipuler des objets pesants que de comparer des distances), une interprétation computationnelle est la suivante : comme l'ensemble est au départ de petite dimension *et* comme il y a plus d'exemples "à distances égales", le réseau apprend d'abord les problèmes à distances égales ; puis quand l'environnement s'enrichit, les exemples à distances inégales sont pris en compte. D'une certaine manière donc, cette expérience montre l'effet d'une stratégie d'apprentissage progressif.

On pourrait certainement allonger la liste de telles expériences, où de manière parfois implicite est utilisée une stratégie d'apprentissage progressif.

Enfin, il convient de signaler un domaine de recherche lié à l'apprentissage progressif : l'étude du *transfert* dans les Réseaux de Neurones, concept issu d'études sur l'apprentissage humain. En psychologie en effet, de nombreuses études ont été réalisées pour montrer l'influence de l'apprentissage d'une tâche sur l'apprentissage d'une autre tâche ; quand l'influence est positive, on parle de transfert positif, dans le cas contraire de transfert négatif. Un certain nombre de chercheurs ont remis en cause

le fait qu'un réseau de neurones était dédié à une seule tâche précise, et ont proposé d'étudier les transfert dans les réseaux.

Par exemple, dans [Sharkey & Sharkey 92], on fait apprendre une première tâche à un réseau à rétropropagation du gradient possédant trois couches, puis on utilise les poids entre les deux premières couches pour initialiser un réseau qui apprend une seconde tâche. Si les deux tâches ont certains points communs, on constate un transfert positif.

On trouvera d'autres études sur le transfert dans les réseaux connexionnistes dans [Pratt *et al.* 91] et [Thrun & Mitchell 94]. Tous ces travaux ont un lien évident avec l'apprentissage progressif, car dans ce dernier, on compte sur un transfert positif d'un sous-ensemble à l'autre.

III.1.2.c Discussion

Il ressort clairement de cet état de l'art un déséquilibre en faveur de l'apprentissage actif informatif : alors que de nombreuse études, aussi bien sur le plan expérimental que sur le plan théorique, ont été réalisées sur l'apprentissage actif informatif, le nombre de chercheurs s'intéressant à ce que nous appelons l'apprentissage progressif se compte sur les doigts d'une main. En fait, l'apprentissage progressif est ou a été étudié *indépendamment* par quelques chercheurs ; les travaux ne se réfèrent pas l'un à l'autre ; il n'y a pas vraiment de communauté scientifique sur ce thème. Nous pensons cependant que c'est un axe à approfondir, et nous proposons pour cela d'examiner ce qu'il reste à faire en apprentissage progressif, compte tenu des études citées ci-dessus.

Il apparaît que les études portant sur l'apprentissage progressif faisant intervenir la notion de difficulté ou de complexité ne relèvent pas vraiment de l'apprentissage actif, puisque c'est le programmeur humain qui intervient pour choisir la progression de l'apprentissage, comme sur le schéma III.3, et non le système lui même.

L'objectif à long terme est donc de rendre ces algorithmes autonomes : l'idée est de concevoir un système capable de sélectionner par lui-même les exemples à apprendre de manière progressive, sans information a priori concernant la tâche. L'enjeu est grand, puisqu'un tel système permettrait de surmonter certaines difficultés rencontrées par les réseaux pour apprendre des environnements complexes (voir [Smith & Zipser 89] ou [Fahlman & Lebiere 91] pour des exemples de tâche complexe). Ce mémoire ne contient pas la solution à ce problème, mais propose des pistes de

recherches, fondées d'un côté sur une analyse informatique des réseaux connexionnistes (chapitre III.1.2) et de l'autre sur une étude psychologique du système cognitif humain (paragraphe III.2).

A toute méthode d'apprentissage actif est associé, explicitement ou implicitement, un *critère de difficulté*. Par exemple, dans [Jacobs 88], il s'agit du rapprochement des entrées donnant deux sorties différentes (voir Tableau III.2) ; pour [Elman 93], il s'agit du degré d'enchâssement des phrases, c'est-à-dire à leur longueur ; pour [Cloete & Ludik 93], il s'agit du nombre jusqu'auquel il faut apprendre à compter. La limitation principale de ce type d'algorithmes, est que, dans le cas général, on ne dispose pas d'un critère de difficulté relatif à la tâche proposée. L'objectif à long terme exposé ci-dessus est donc subordonné à la définition d'un critère de difficulté général, qui ne nécessite pas une connaissance a priori sur la nature de la tâche.

Mais en amont de ce problème se pose la question suivante : existe-t-il, pour n'importe quelle tâche, un critère de difficulté permettant d'ordonner les exemples de telle manière que l'apprentissage soit facilité ? Nous postulons que pour une tâche atteignant une certaine complexité, la réponse est positive. Autrement dit, nous affirmons, sans le démontrer, qu'à partir d'une certaine complexité, l'apprentissage passif "d'un bloc" n'est jamais la meilleure méthode pour faire apprendre un réseau. Cependant, il est difficile de s'avancer en ce qui concerne la *possibilité* de déterminer ce critère : il n'est peut-être pas possible de trouver ce critère *a priori*, c'est-à-dire sans avoir déjà une forte connaissance de la tâche, et en particulier l'avoir déjà résolue.

Le critère de difficulté ne suffit pas à spécifier entièrement un algorithme d'apprentissage progressif. Comme on peut le remarquer, dans un apprentissage progressif, l'apprentissage est découpé en *phases*. Un aspect primordial d'une session d'apprentissage progressif est le *critère de passage* d'une phase à l'autre, c'est-à-dire le critère qui permet de décider que l'ensemble d'apprentissage courant est suffisamment appris, et que l'on peut aborder le suivant. Différents critères ont été utilisés :

- nombre d'epochs fixe [Elman 93][Cachin 93] : chaque ensemble d'apprentissage est présenté un nombre fixe de fois au réseau, déterminé avant l'expérience ;
- Seuil sur l'erreur fixe [Selfridge *et al.* 85][Jacobs 88][Cloete & Ludik 93] : chaque ensemble est appris jusqu'à succès, c'est-à-dire jusqu'à ce que l'erreur descende au dessous d'un certain seuil ; ce seuil est identique à celui qui permet de décider en fin d'apprentissage que le problème est résolu ;
- Seuils sur l'erreur multiples, déterminés avant apprentissage [Cloete & Ludik 93] : à chaque changement de phase est associé un seuil sur l'erreur pour passer d'une phase à l'autre ; la suite des seuils est décroissante ;

- Suite de seuils déterminés par une loi linéaire [Cotrell & Tsung 91][Ludik & Cloete 94] : le seuil suit une loi linéaire décroissante, partant d'une valeur élevée (par exemple l'erreur initiale) pour atteindre un faible valeur, correspondant à la tâche résolue.

Pour tous ces critères, une intervention a priori est nécessaire, que ce soit pour fixer le nombre d'epochs, les seuils, ou la loi de décroissance des seuils. Comme le choix de ces seuils s'avère déterminant pour les performances, l'objectif est de trouver un moyen de déterminer ces seuils automatiquement et dynamiquement, selon le déroulement de l'apprentissage.

Pour finir, on remarquera que la plupart des cas d'apprentissage progressif concernent les réseaux récurrents [Pearlmutter 89][Toomarian & Barhen 91][Elman 91][Fahlman 91][Cloete & Ludik 93] : dans toutes ces études, le critère de difficulté se ramène au bout du compte à la *longueur* des séquences traitées. Alors que les réseaux à couches font l'objet de très nombreuses investigations ces dernières années, seul [Jacobs 88] les étudie vraiment sur le plan de l'apprentissage progressif, mais sur une tâche très simple. Une lacune est donc à combler ; il faut montrer plus clairement que d'autres réseaux que les réseaux récurrents peuvent faire l'objet d'un apprentissage progressif, en utilisant d'autres critères de difficulté que la longueur

III.1.3 Vers une formalisation de l'apprentissage progressif

Etant donné la jeunesse du champ de recherche que constitue l'apprentissage progressif, nous ne sommes pas en mesure d'en proposer une théorie complète et définitive. Ce sont les premières pierres que nous posons ici : dans ce qui suit, nous nous attachons d'une part à construire un cadre adapté à l'étude de l'apprentissage progressif, qui pourra ensuite être utilisé de manière plus formelle, et d'autre part à donner quelques pistes de recherche qui iraient dans ce sens.

III.1.3.a Les deux dimensions de l'apprentissage actif

Au chapitre précédent ont été distinguées l'action informative et l'action progressive. Cette distinction doit être nuancée et précisée.

En effet, il est intéressant de considérer les aspects "informatif" et "progressif" non plus comme deux *classes* d'algorithmes, mais comme deux *dimensions*, deux traits, caractérisant les algorithmes. La dimension informative caractérise les

algorithmes fondés sur la maximisation de l'information reçue de l'environnement ; la dimension progressive caractérise les algorithmes fondés sur la progressivité de la difficulté des informations reçues de l'environnement.

On se rend compte que de nombreux algorithmes que nous avons classés au chapitre précédent dans la catégorie "action informative" ont en fait une composante progressive : quand les exemples sont ajoutés un à un dans l'ensemble d'apprentissage (apprentissage incrémental), la difficulté du problème croît avec le temps [Röbel 94]. L'augmentation de performance est donc due à la fois au fait que seules les informations pertinentes sont apprises et au fait que l'apprentissage est progressif. Une variante de ces algorithmes utilisant uniquement une action informative consisterait à sélectionner un sous-ensemble représentatif de l'ensemble d'apprentissage, *puis* de le présenter au réseau, comme il est réalisé dans [Wann *et al.* 89]. Les algorithmes qui sautent des exemples sont quant à eux purement informatifs.

Si les algorithmes informatifs de type *incrémental* sont progressifs, pourquoi alors s'attacher à chercher d'autres algorithmes d'apprentissage progressif ? Parce que ces algorithmes ne sont que "faiblement progressifs" : le critère de difficulté est *le nombre d'exemples non redondants* (ceci est très proche de l'algorithme "combined subset training"), critère qui donne très rapidement une tâche difficile à apprendre. Même si la notion de difficulté est manipulée de manière plutôt intuitive, nous nous permettons d'imaginer les courbes de la difficulté en fonction du temps selon les différents types d'algorithmes (figure III.5) : dans l'apprentissage classique ou purement informatif, la difficulté est tout de suite maximale ; dans l'apprentissage informatif incrémental, la difficulté croît rapidement. Dans l'apprentissage progressif, la difficulté croît plus graduellement, par exemple de manière continue ou en escalier.

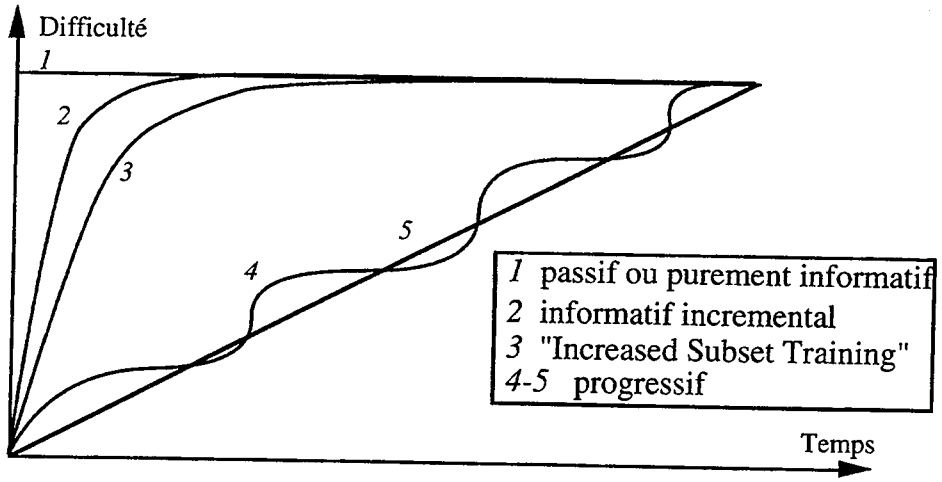

Figure III.5. Evolution supposée de la difficulté pour différents algorithmes

Pour finir, nous proposons un autre graphique (figure III.6), illustrant l'espace à deux dimensions dans lequel se placent les différents algorithmes d'apprentissage actif. L'abscisse représente l'action "progressive" et l'ordonnée l'action "informative". En *A*, à l'origine, se placent tous les algorithmes passifs ; en *B* se placent les algorithmes purement fondés sur l'action informative [Wann *et al.* 89][Strand & Jones 92] ; en *C* se placent les algorithmes actifs informatifs incrémentaux ; en *D* se situent les algorithmes progressifs qui ont fait l'objet du paragraphe précédent. Nous avons aussi distingué les algorithmes autonomes (représentés par un rond) des algorithmes où le concepteur humain intervient (représentés par un carré). Les points *E* et *F* correspondent à des algorithmes encore à concevoir. Les deux axes de ce graphique ne correspondent pas seulement à deux moyens d'accélérer la convergence des algorithmes. En effet, l'action "progressive" permet en plus à un réseau d'atteindre une solution qu'il ne pourrait pas atteindre sans elle, comme le montre l'expérience de [Elman 91]. L'objectif, en étudiant l'action progressive, est effectivement d'étendre la classe de problèmes que les réseaux connexionnistes actuels sont capables de résoudre.

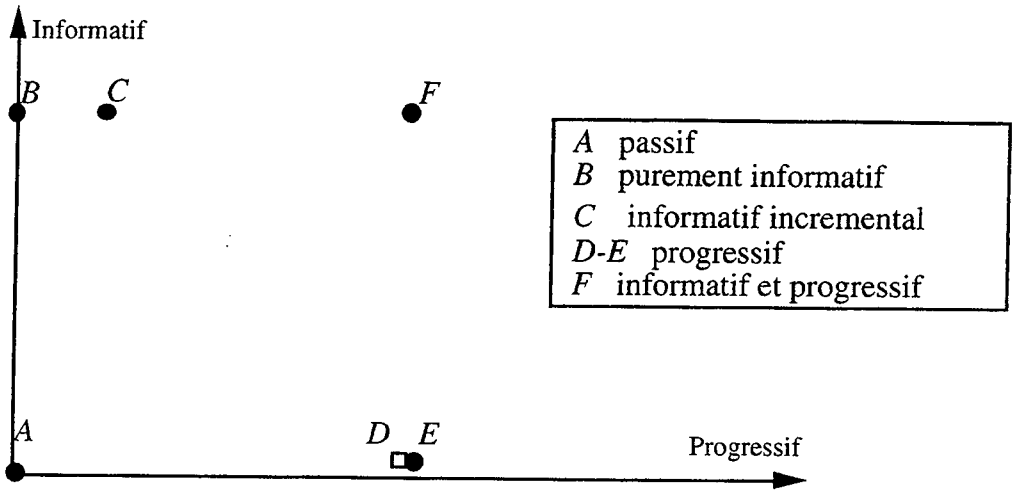

Figure III.6. Taxonomie des différents types d'algorithmes actifs

Autant la dimension "informative" est relativement bien formalisée, puisqu'elle peut faire appel à la théorie de l'information, et se rapproche de techniques statistiques déjà connues, la dimension "progressive" est peu formalisée : la démarche a été jusqu'ici plutôt empirique.

III.1.3.b Tâche cible et sous-tâches

Effectuer un apprentissage progressif revient à considérer que le problème à résoudre se modifie au cours de l'apprentissage. Intuitivement, si, alors que les poids sont en train d'évoluer, le problème lui-même se modifie, l'apprentissage sera perturbé : les poids changeront de direction perpétuellement. C'est un peu ce problème

qui est mentionné dans [Munro 92] (voir paragraphe III.1.2.a) : quand l'exemple à apprendre change à chaque itération, l'apprentissage est en difficulté. Il semble donc important d'introduire une discrétisation temporelle macroscopique du déroulement de l'apprentissage, c'est-à-dire de considérer l'apprentissage comme une succession d'un nombre limité de phases, au cours desquelles seuls les poids évoluent. A chaque phase est associée une *tâche*, notion qui va être définie avec précision.

Une *tâche* est constituée de deux éléments :

- une *évaluation de performance* du réseau, sur une zone de l'environnement donnée. Cette évaluation prend la forme mathématique suivante :

$$E = F(f_r(W,x), f_e(x)) \ , \ x \in D \tag{54}$$

avec :

W le vecteur des poids du réseau ;
x le vecteur présenté en entrée du réseau ;
fr, la fonction réalisée par le réseau ;
fe, la fonction réalisée par l'environnement ;
F la fonction de calcul de l'évaluation ;
D l'ensemble des vecteurs d'entrée x.

- un critère d'arrêt qui permet de décider quand la tâche est suffisamment apprise.

Pour illustrer cette formulation très générale, nous donnons le cas classique de l'apprentissage passif d'une base d'exemple $\{(x1,y1), \ldots (x_n,y_n)\}$, dans lequel il n'y a qu'une seule tâche :

$$E = (f_r(W,x_i) - y_i)^2, \ D = \{(x_1,y_1), \ \ldots \ ,(x_n,y_n)\}$$

Dans ce cas, on a posé $y_i=f_e(x_i)$; F est le carré de la différence.

Etant donné une tâche d'apprentissage proposée à un réseau, que l'on nomme *tâche cible*, constituée d'une *fonction d'évaluation cible* et d'un *critère d'arrêt cible*, l'apprentissage progressif consiste à définir une succession de sous-tâches qui vont aboutir à cette tâche cible. On peut ainsi définir trois grandes catégories d'actions, à partir de la formule (54) :

- <u>action sur "x" :</u> cette action consiste à modifier les données qui arrivent au réseau. La plupart des algorithmes progressifs sont de ce type, puisqu'ils consistent à sélectionner les données pour former un certain sous-ensemble d'apprentissage. L'action sur les données est bien sûr adaptée au cas de l'action par "query". Notons que "x" peut être choisi dans D mais aussi à l'extérieur de D : les sous-tâches peuvent amener à présenter au réseau des exemples qui ne sont pas dans la tâche cible.

- action sur "f_e" : il s'agit de modifier la fonction réalisée par l'environnement, c'est-à-dire le problème posé lui-même. On peut en effet envisager le cas où une sous-tâche va consister en un autre problème que le problème cible, comme par exemple un problème analogue. C'est ce qui est réalisé dans [Jacobs 88], dans le cas où des fonctions sinusoïdales de pulsation croissante sont présentées au réseau (voir tableau III.3). Dans la perspective d'une action du système vers l'environnement, cela suppose que le problème est *paramétré*, c'est-à-dire modifiable par l'intermédiaire d'entrées supplémentaires. Cette condition n'est pas toujours réalisable en pratique.
- action sur "F" : il est possible de modifier la fonction de calcul de l'évaluation, c'est-à-dire la fonction d'erreur dans la plupart des cas. On peut ainsi modifier le calcul de l'erreur pour prendre en compte plus un pattern que l'autre, se focaliser sur une entrée, ajouter des termes supplémentaires, etc. L'action sur "F" peut rejoindre l'action sur "x", quand elle revient à focaliser sur des patterns particuliers.

Pour la suite, quel que soit le type d'action envisagé, nous codons une action par un *vecteur A*. Ce vecteur peut représenter des variables de l'apprentissage très diverses : une région de l'espace d'entrée, des poids pondérant les contributions des exemples dans la surface d'erreur, une séquence temporelle d'entrées pour un réseau récurrent, etc.

L'action A peut être directe ou indirecte. Dans le cas général, on considère que l'action A s'applique sur *l'environnement proche* du réseau, qui est soit l'environnement lui même dans le cas d'une action directe, soit la base d'exemples dans le cas d'une action indirecte (voir figures III.1 et III.2).

III.1.3.c Organisation fonctionnelle

Nous souhaiterions étudier maintenant, indépendamment des principes et algorithmes utilisés pour déterminer l'action qui est envoyée vers l'environnement, l'organisation générale d'un système actif et préciser les schémas III.1 et III.2.

Un réseau supervisé classique, c'est-à-dire passif, reçoit de l'environnement des couples $(x , f_e(x))$, qu'il utilise pour modifier son comportement. Pour rendre un tel système actif, il faut ajouter une unité de traitement, que nous appelons *module actif*, qui d'un côté agit sur l'environnement proche en produisant un vecteur A à chaque instant, et de l'autre fournit au réseau un couple $(x , f_e(x))$, comme illustré sur la figure III.7.

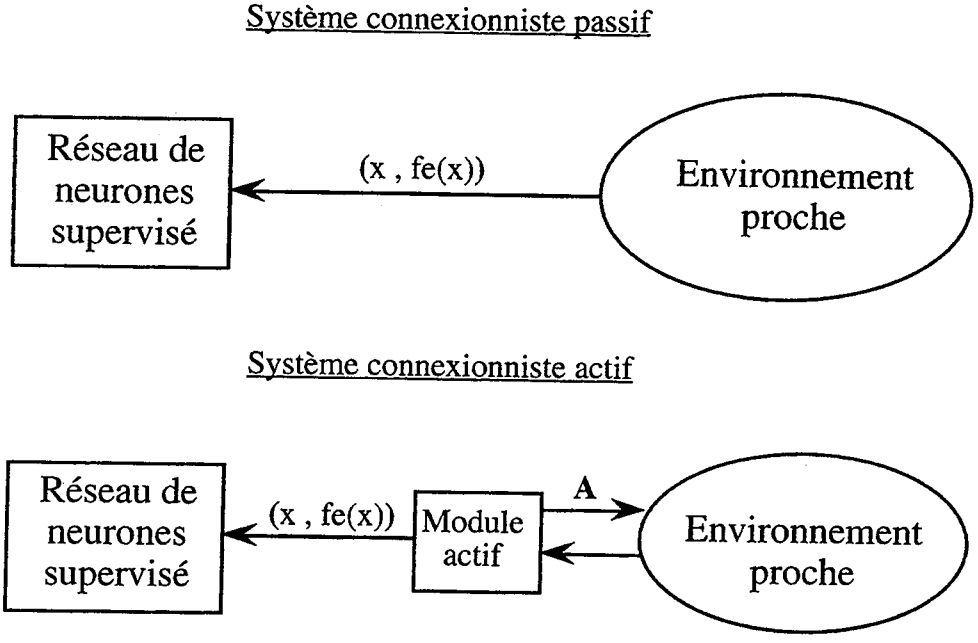

Figure III.7. Schémas des apprentissages passif et actif

Dans ce schéma, le réseau lui-même demeure passif, et le module actif réalise une sorte de prétraitement des données en provenance de l'environnement proche. Ce cas de figure est cependant limité, car l'action est indépendante du réseau : les différents exemples du réseau sont présentés d'une manière qui ne dépend que de l'environnement proche.

Or, la dynamique du réseau doit influencer l'action vers l'environnement. On peut en effet supposer qu'il n'existe pas de présentation idéale des patterns : selon l'initialisation du réseau, certains patterns doivent être appris avant d'autres. Par exemple, supposons que le réseau soit par hasard initialisé proche d'une solution parfaite, alors il ne sert à rien de réaliser un apprentissage progressif : les premières tâches ne vont pas faciliter l'apprentissage, puisque le réseau était déjà bien initialisé. Autre exemple : supposons que pour apprendre *C*, il faille apprendre *A*, puis *B*, puis *C*, ou *B*, puis *A*, puis *C* ; si le réseau est initialisé de manière à faciliter l'apprentissage de *B*, alors c'est l'ordre "*B*, puis *A*, puis *C*" qu'il faudra choisir. Ainsi, l'action pour apprendre a intérêt à être dépendante de l'état du réseau, selon le schéma III.8. On dénote ce type d'apprentissage actif apprentissage *réflexif*. Les informations provenant du réseau utilisées par le module actif peuvent être diverses : erreur du réseau sur un exemple, dynamique des poids, etc.

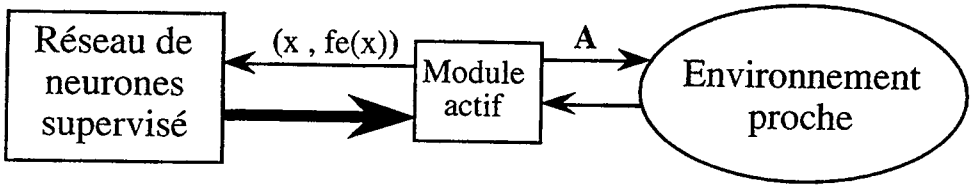

Figure III.8. Apprentissage actif réflexif

Selon le schéma de la figure III.8, le réseau n'est plus totalement passif, puisqu'il y a une transmission d'informations en provenance du réseau de neurones en direction de l'environnement, via le module actif. Ce dernier peut a priori être constitué de tout type de mécanismes, connexionnistes, mais aussi symboliques.

III.1.3.d Surfaces d'erreur

La plupart des réseaux supervisés sont des algorithmes de minimisation qui utilisent l'information du gradient pour modifier leurs paramètres. Ces algorithmes se visualisent bien en deux dimensions sous la forme d'une *surface d'erreur,* dont on cherche le minimum, en "laissant rouler une bille dessus" (voir figure II.9 au paragraphe II.2.2.a).

Une manière intuitive de formaliser la notion de difficulté est de la relier aux différentes caractéristiques de cette surface d'erreur. En particulier, si l'apprentissage échoue (c'est le cas de figure qui nous intéresse), c'est parce que la surface d'erreur contient des *pièges*. Le piège le plus connu est le minimum local à valeur élevée, duquel l'algorithme ne peut s'échapper. Un autre cas de figure est le plateau : la surface d'erreur est très plate, mais l'erreur élevée. Nous définissons un piège comme *une caractéristique de la surface d'erreur qui empêche le réseau de s'améliorer de manière significative.*

Quand l'apprentissage échoue, on peut donc dire qu'il y a un piège dans la surface d'erreur. Or la surface d'erreur dépend uniquement :

- de la topologie du réseau (liée à la fonction f_r),
- de la manière de calculer l'erreur (fonction F),
- de l'ensemble des entrées envoyées à l'environnement (ensemble D) et de la fonction réalisée par l'environnement (fonction f_e) dans le cas d'une action directe, ou de l'ensemble d'apprentissage dans le cas d'une action indirecte.

Modifier l'ensemble d'apprentissage ou les entrées envoyées à l'environnement permet donc de modifier la surface d'erreur, et éventuellement d'en supprimer certains pièges. Lors d'une session d'apprentissage actif, *la surface d'erreur se modifie durant l'apprentissage* ; la bille roule sur un "terrain mouvant".

Dans ce cadre, l'idée de l'apprentissage actif est la suivante : dans une première phase où la tâche est "simple", les poids (représentés par une bille sur la surface d'erreur) vont évoluer vers une certaine valeur qui servira de point de départ pour la deuxième tâche d'apprentissage, et ainsi de suite. Ainsi, la première tâche initialise le

réseau dans une configuration qui est censée l'aider pour la seconde tâche. Lorsque, comme c'est le cas en apprentissage progressif, le réseau doit finalement s'attaquer à la tâche cible, il démarre dans un état bien meilleur que l'initialisation aléatoire traditionnelle, aidé par les tâches précédentes.

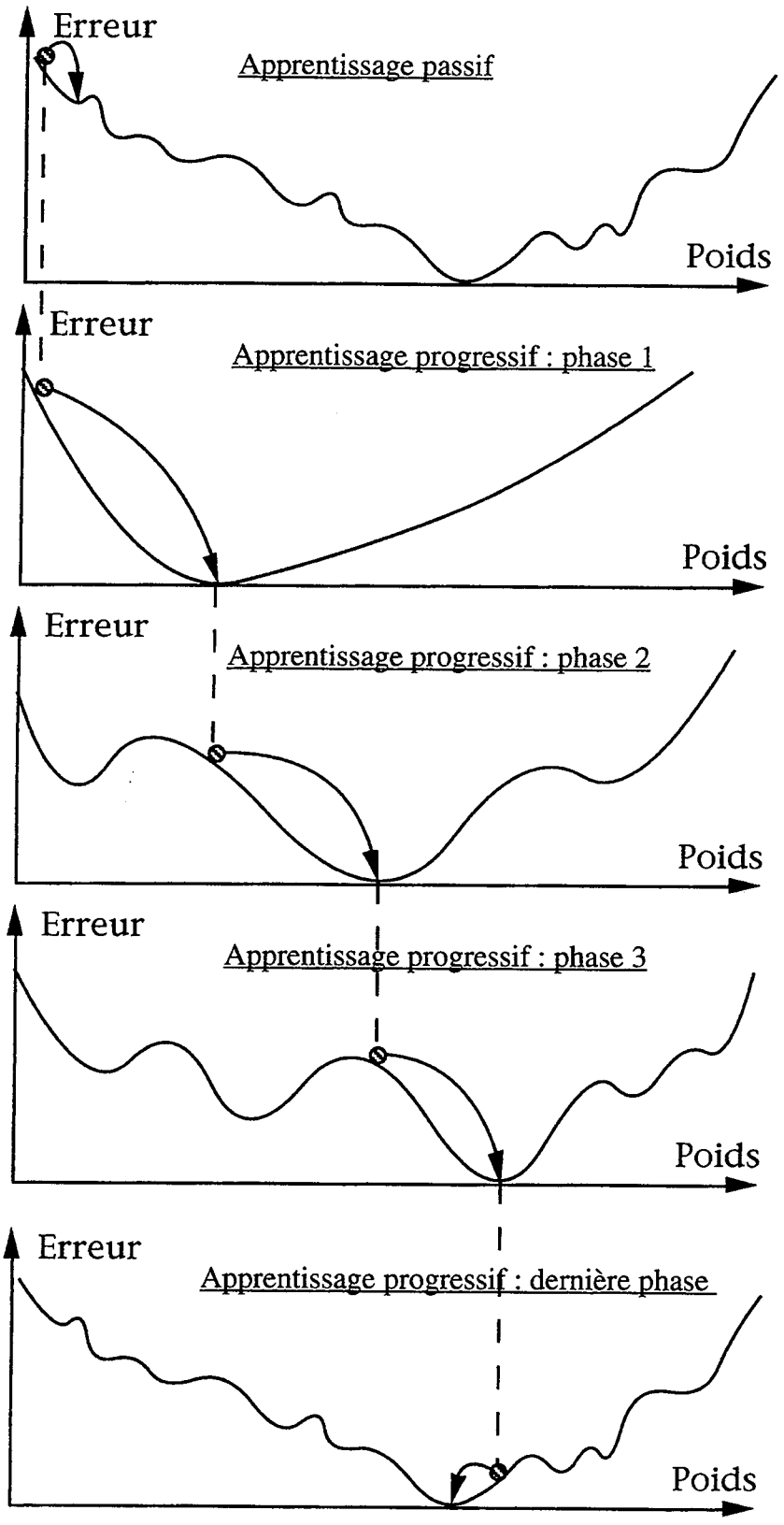

Figure III.9. Augmentation progressive de la difficulté dans l'apprentissage actif

La figure III.9 illustre ce phénomène, dans un cas imaginaire où le réseau comporte un seul poids. On voit que dans le cas d'une surface accidentée, l'initialisation aléatoire a beaucoup de chance de tomber dans un piège, un minimum local dans ce cas. Par contre, les surfaces intermédiaires comportent moins de pièges (car plus simples), et permettent de s'approcher de la solution.

En conséquence, une direction de recherche est de chercher l'influence des diverses formes d'action sur la forme et la complexité de la surface d'erreur, afin de pouvoir sélectionner des premiers ensembles "faciles".

Une telle vision de l'apprentissage progressif est à mettre en relation avec les travaux consacrés à *l'initialisation* des poids dans les réseaux à couches : toutes les tâches intermédiaires avant la tâche finale peuvent être vues comme un moyen d'initialiser correctement le réseau. On montre que l'initialisation des poids a une grande influence sur les résultats de l'apprentissage, ne serait-ce que l'amplitude d'initialisation aléatoire autour de zéro [Kolen & Pollack 91]. Des études ont montré l'intérêt d'initialiser les poids en prenant en compte des exemples prototypiques, représentatifs de la tâche [Denoeux & Lengellé 93]. Mais dans ces expériences, l'initialisation est réalisée "par l'intérieur", c'est-à-dire par calcul des poids, alors que dans l'apprentissage progressif, cette initialisation se fait via la tâche elle même, sans qu'aucune modification de l'algorithme d'adaptation des poids ne soit utilisée.

D'autres algorithmes font aussi mention d'une modification de la surface d'erreur au cours de l'apprentissage : il s'agit des études sur l'ajout d'un bruit stochastique sur l'évolution des poids, contrôlé par une température (recuit simulé [Bonomi & Lutton 88]) ; on montre que ce bruit revient à lisser la surface d'erreur. De même, l'ajout d'un bruit sur les entrées est utilisé pour lisser la surface d'erreur. De telles techniques sont peut-être une forme d'apprentissage progressif, puisqu'en diminuant progressivement la température on complexifie la surface d'erreur, mais il s'agit d'un cas très particulier : les surfaces d'erreurs successives sont telles que les solutions de la surface d'erreur finale sont aussi celles des surfaces d'erreur intermédiaires. De plus, les techniques de ce type sont souvent très lentes, un ordre de grandeur plus lentes que la rétropropagation [Bengio *et al.* 94], et l'on perd alors l'un des attraits de l'apprentissage actif.

Une vision alternative à celle d'une surface d'erreur variable avec le temps consiste à considérer l'erreur comme fonction non seulement des poids mais aussi de l'action courante A. Cette erreur définit une surface que nous dénotons surface "poids-action", pour la différencier de la *surface d'erreur* dont il est question plus haut. Il apparaît que la surface d'erreur est une projection de la surface "poids-action" sur le

sous-espace correspondant à A = *action cible*. Graphiquement, on peut représenter la surface "poids-action" en se limitant au cas où il y a un seul poids et une seule composante d'action (figure III.10).

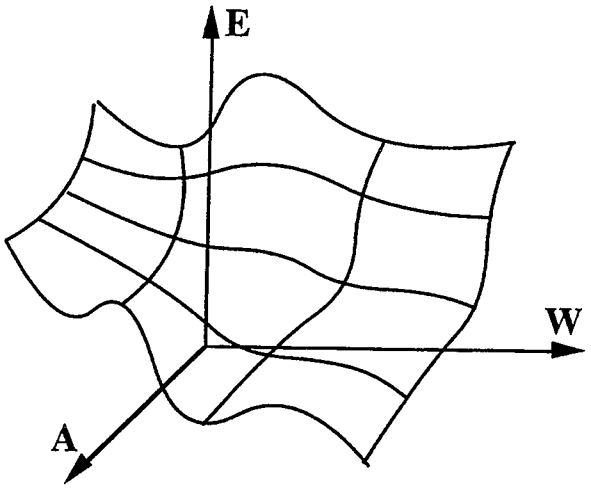

Figure III.10. Surface "poids-action"

Le fait que la surface d'erreur varie avec le temps correspond tout simplement au fait que la tâche évolue selon les dimensions de l'action sur la surface "poids-action".

L'ajout de ces dimensions supplémentaires présente l'avantage suivant : si l'apprentissage est bloqué dans un minimum local, on peut s'en échapper en se déplaçant selon les coordonnées de l'action, comme illustré idéalement sur la figure III.11.

Plus généralement, le but de l'apprentissage progressif dans ce contexte est de modifier les coordonnées d'action afin de se placer dans une zone comportant moins de pièges.

Si le vecteur A peut évoluer en continu et si l'erreur est dérivable selon toutes les composantes de A, on peut utiliser des informations sur le gradient pour se déplacer sur la surface "poids-action". Ainsi, cette surface ouvre peut-être la voie vers une approche plus formelle de l'apprentissage progressif.

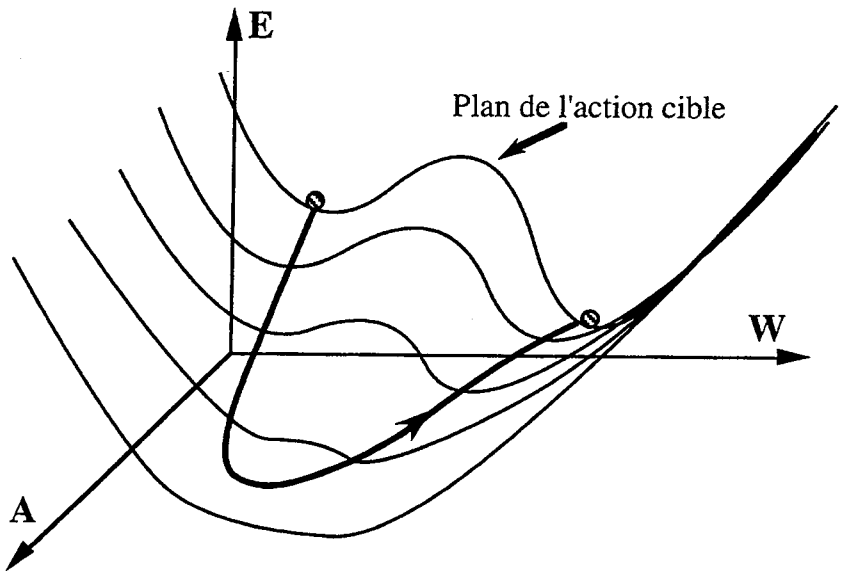

Figure III.11. Visualisation de l'effet positif de l'apprentissage actif progressif

III.1.3.e Le critère de difficulté de l'apprentissage

Le point central de la notion d'apprentissage progressif est le critère de difficulté qui va permettre de déterminer quelle action doit être choisie à un moment donné de l'apprentissage. Ce paragraphe a pour but de dégager quelques caractéristiques générales relatives à cette notion de difficulté.

L'informatique s'est déjà penchée sur le problème de la *complexité* de certains problèmes, en termes de coût computationnel nécessaire pour résoudre la tâche. Dans le domaine du connexionnisme, ce coût demeure très difficile à estimer car il dépend surtout de la trajectoire sur la surface d'erreur : si la trajectoire sur la surface est régulière, l'apprentissage réussira rapidement, tandis que dans le cas contraire, l'apprentissage échouera ou sera très long (voir figure III.9).

Ces considérations sur la surface d'erreur amènent à mettre en évidence le caractère *relatif* de la difficulté : suivant l'état du réseau, c'est-à-dire selon son vecteur poids, une même tâche peut être difficile ou facile : si par exemple la position actuelle du réseau sur la surface d'erreur est telle que le chemin de ce point à la solution soit dépourvu de pièges, alors l'apprentissage est facile ; si au contraire il y a un piège entre la position actuelle et la solution, l'apprentissage est difficile. *La difficulté de l'apprentissage est donc relative à l'état du réseau qui apprend la tâche* (voir figure III.12).

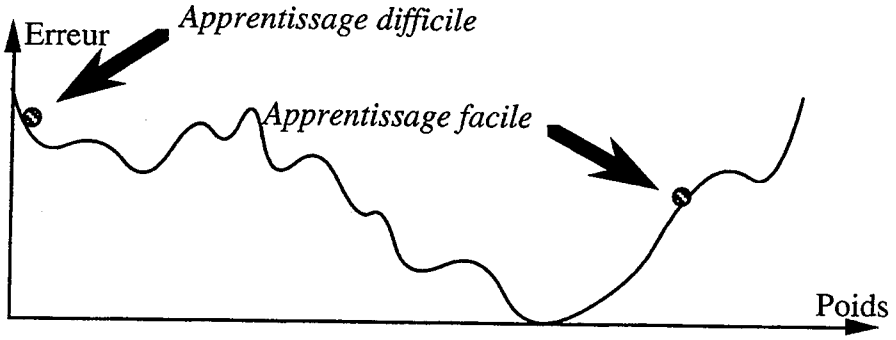

Figure III.12. Difficulté relative d'une tâche

Par conséquent, l'apprentissage progressif a pour objectif de maintenir la difficulté relative en dessous d'un certain seuil. Ce seuil correspond d'une certaine manière à la limite computationnelle du réseau : confronté à une tâche plus difficile, il échoue ou atteint difficilement la solution. Si par contre il est confronté à une succession de tâches faciles, relativement à son état, il pourra, en fin de session d'apprentissage, apprendre une tâche qu'il n'aurait pas pu apprendre d'un coup. *C'est donc paradoxalement en posant que le réseau est limité, que l'on peut envisager augmenter ses performances par apprentissage progressif.*

Notons que la difficulté absolue d'une tâche peut tout de même être définie, comme la moyenne de la difficulté relative sur toutes les configurations possibles des poids du réseau. C'est cette notion de difficulté absolue qui est utilisée quand il s'agit "d'augmenter progressivement la difficulté (ou la complexité) de la tâche".

Un autre aspect fondamental concernant la notion de difficulté doit être maintenant discuté : les tâches successives ne doivent pas être choisies uniquement pour rendre la tâche actuelle facile. Pour expliquer ce point, nous prenons un exemple de classification en deux classes d'éléments situés dans un plan : l'abscisse et l'ordonnée correspondent aux deux entrées du réseau. Sur la figure III.13a est représentée la tâche de classification : les croix représentent la première classe, les cercles la deuxième. Supposons que l'on augmente progressivement le nombre d'exemples de l'ensemble d'apprentissage (voir figure III.13b). Si on choisit d'abord les exemples *1* et *2*, le réseau apprendra la séparation linéaire *L1*, que l'on retrouve dans la tâche cible ; si ensuite on ajoute les exemples *3* et *4*, alors une autre séparation *L2* sera créée ; or cette dernière ne correspond à aucune séparation pour la tâche cible : elle n'aidera pas l'apprentissage, et pourra même le bloquer. L'ajout des exemples *3* et *4* revient donc à présenter une tâche facile au réseau, mais celle-ci "ne va pas dans le sens" de la tâche cible. La sélection de la sous-tâche suivante doit donc être non seulement locale, c'est-à-dire fondée sur la difficulté relative de tâche, mais aussi globale, tenant compte de la tâche cible.

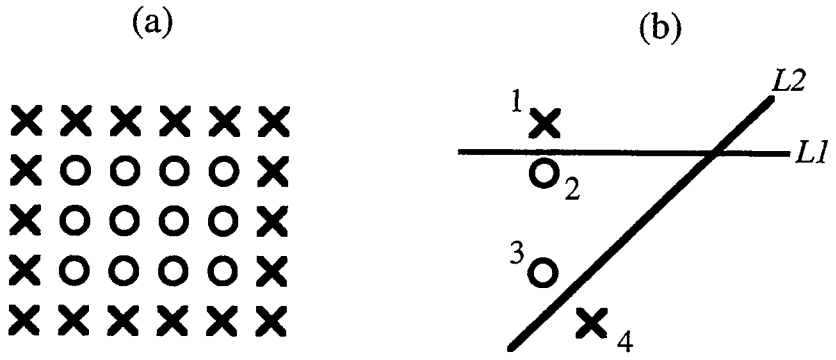

Figure III.13. Exemple de sous-tâche facile qui gêne le réseau

Examinons maintenant quelques critères que l'on peut trouver dans la littérature, à la lumière des remarques précédentes.

Le critère le plus simple est la *nouveauté*. Il consiste à sélectionner les exemples non vus par le réseau. Augmenter la taille de l'ensemble d'apprentissage, comme c'est le cas des algorithmes présentés dans [Cotrell & Tsung 89][Cloete & Ludik 94a][Cachin 94], relève de ce critère, dans le cas où aucun exemple n'est en double. La notion d'exploration étudiée dans [Thrun & Möller 92] relève aussi du critère de nouveauté. Le critère de nouveauté est très approximatif, car l'ajout d'un exemple nouveau peut aussi bien donner une tâche facile que difficile.

Un critère très utilisé en apprentissage informatif est l'erreur du réseau sur les exemples (voir paragraphe III.1.2.a). La difficulté d'un exemple varie-t-elle avec l'erreur commise par le réseau sur cet exemple ? Il est évident que si l'erreur est très faible, alors l'exemple est facile à apprendre, puisqu'il est déjà presque appris. Mais la réciproque est fausse : une tâche à erreur élevée peut être plus facile à apprendre, qu'une tâche à faible erreur, comme on peut le comprendre en raisonnant sur la surface d'erreur (figure III.14).

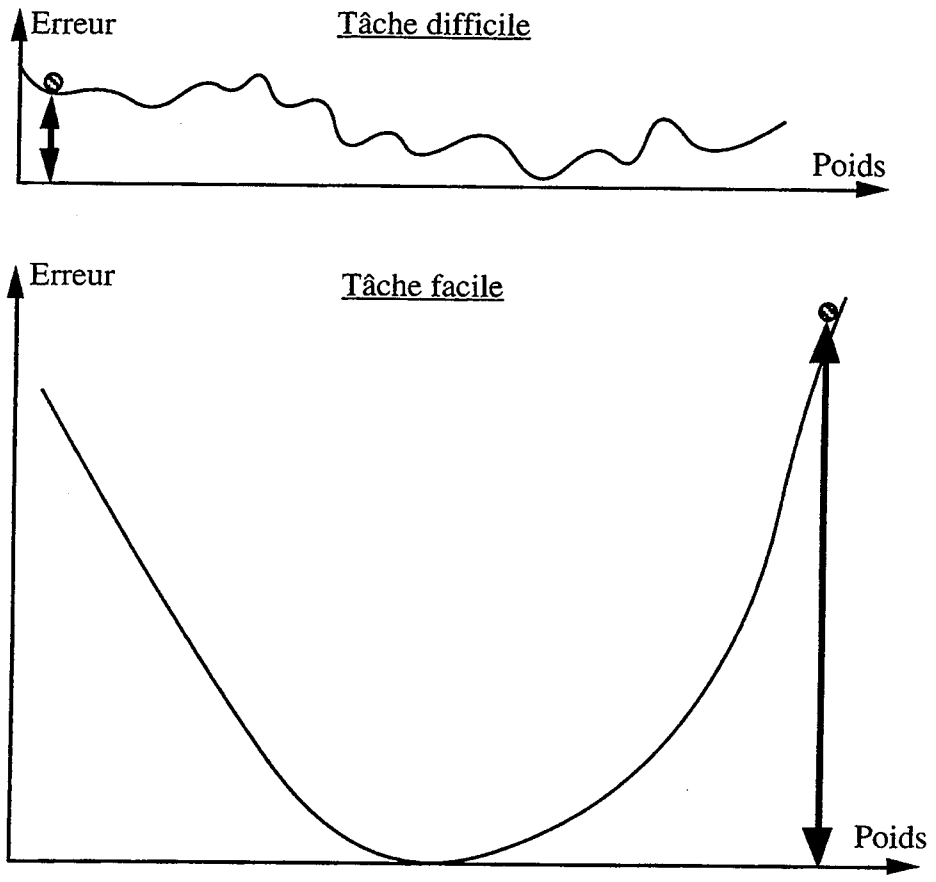

Figure III.14. La difficulté n'est pas proportionnelle à l'erreur

Dans [Ludik & Cloete 95], un critère de proximité sur l'espace d'entrée est utilisé. Même si les auteurs obtiennent des résultats très intéressants avec ce critère, il apparaît que l'aspect *relatif* de la difficulté n'est pas prise en compte, puisque la session d'apprentissage est entièrement prédéfinie avant l'apprentissage. De plus, la difficulté est estimée uniquement sur les valeurs d'entrées, sans prendre en compte les sorties ; or la difficulté d'une tâche de classification est liée aux sorties, et en particulier à la variation des sorties par rapport aux entrées (voir l'étude de [Jacobs 88]).

Enfin, la *stagnation* à erreur élevée est utilisée dans [Fahlman & Lebiere 91] pour décider qu'il faut ajouter des unités à un réseau incrémental constructif ("Cascade Correlation"). Ce critère mesure bien la difficulté, puisqu'il détecte l'existence d'un piège dans la surface d'erreur.

On dispose donc de moyens *a posteriori* de décider si un exemple ou un ensemble d'exemples est facile (erreur faible) ou difficile (stagnation). Ainsi on peut envisager des algorithmes effectuant une recherche par essai-erreur des exemples à apprendre, afin de trouver une sous-tâche facile. Cependant, une telle démarche est *locale* et risque d'aboutir au cas de la figure III.13.

III.1.3.f Les critères de passage

La définition des tâches successives nécessite non seulement le choix d'une action selon un certain critère de difficulté (le "quoi"), mais aussi le choix d'un critère d'arrêt de chaque tâche, ou de passage d'une action à la suivante (le "quand").

Comme le montre l'étude bibliographique du chapitre III.1.2, le critère de passage, plus ou moins élaboré, est déterminant pour la réussite de l'apprentissage.

Dans la plupart des algorithmes d'apprentissage progressif, le critère est un seuil sur l'erreur calculée sur l'ensemble d'apprentissage courant. Le seuil n'est généralement pas constant, mais est spécifique de chaque sous-tâche. Il est choisi décroissant en fonction du temps, pour la raison suivante :

> *"Quand le réseau ne voit qu'un sous-ensemble de l'ensemble d'apprentissage entier, de nombreuses solutions partielles* pour ce sous-ensemble *sont possibles. Apprendre avec une trop grande précision peut forcer le réseau à choisir l'une de ces solutions locales, solution qui peut ne pas se généraliser à la solution globale désirée. Arrêter à un critère d'erreur élevé laisse le choix ouvert, et empêche le réseau de plonger trop profondément vers un minimum local." [Cotrell & Tsung 91] (traduit de l'anglais)*

Ces seuils sont soit réglés à la main, soit fixés par une loi de décroissance linéaire. A la lecture de cette explication de G. W. Cotrell et F. S. Tsung, on peut proposer une procédure de calcul automatique des seuils sur l'erreur qui évite d'imposer une loi linéaire. Dans le cas d'un ensemble d'apprentissage fixe donné, en même temps que le réseau apprend le sous-ensemble courant, on peut mesurer l'erreur sur l'ensemble entier et détecter les *remontées* de cette erreur, qui indiquent que le réseau est en train de "plonger trop profondément" vers une solution particulière.

Soient *A1, A2, ..., An,* les actions successives, que l'on suppose déjà ordonnées de manière optimale pour réaliser un apprentissage progressif. L'algorithme de détermination des critères de passage est le suivant :

DEBUT
Pour i variant de *0* à *n*
Adapter le réseau avec l'action *Ai* tant que
• *E(An)* diminue
et • *E(An)* > Seuil d'arrêt
FIN

avec *E(Ai)* l'erreur du réseau pour l'action *Ai*.

Cet algorithme est relativement proche de la technique employée dans [Röbel 94] dans le cadre de l'apprentissage informatif, où chaque exemple est ajouté à l'ensemble d'apprentissage courant quand la généralisation sur l'ensemble de validation remonte.

Il s'agit d'un algorithme "prudent", dans la mesure où on est garanti qu'à chaque étape l'erreur pour la tâche cible ne peut augmenter (ou très peu, le temps d'une itération). Cet algorithme présente uniquement le risque que les différentes actions *Ai* défilent très vite et que le réseau soit très rapidement confronté à la tâche cible.

Le défaut majeur de cet algorithme est que l'on ne s'autorise finalement pas à sortir d'un minimum local : l'erreur sur l'ensemble d'apprentissage entier ne peut pas remonter. Autrement dit, si, sur la surface d'erreur, un obstacle empêche d'aboutir à la solution (une remontée de la surface), alors les poids ne pourront franchir cet obstacle. Une trajectoire comme celle de la figure III.11 n'est pas envisageable.

Il est donc souhaitable de voir l'erreur sur la tâche cible remonter alors que le réseau apprend la tâche courante. Dans [Shultz *et al.* 94], on montre que l'apprentissage incrémental de la "tâche de la balance" par un réseau de neurones de type "Cascade Correlation", se fait avec une ou plusieurs remontées de l'erreur sur la tâche cible. Les auteurs rapprochent ce comportement du phénomène connu en psychologie de "développement en U des performances" : un enfant peut apprendre une connaissance locale qui le rendra moins performant sur une tâche plus difficile, mais lui permettra par la suite de résoudre cette tâche. Ce type de développement en U est certainement souhaitable pour l'apprentissage progressif connexionniste.

Aussi proposons-nous un autre algorithme pour le passage d'une tâche à l'autre, qui au lieu d'observer la remontée d'erreur sur la tâche cible, l'observe seulement sur la tâche suivante :

DEBUT
Pour *i* variant de *0* à *n-1*
Début
    Adapter le réseau avec l'action *Ai* tant que :
        E(Ai+1) diminue
Fin
Adapter le réseau avec l'action *An* jusqu'à *An* < Seuil d'arrêt
FIN

Ce dernier algorithme devrait permettre de remplacer la mise au point manuelle des seuils sur l'erreur, sans nécessiter d'imposer une décroissance linéaire de ces seuils. Il serait d'ailleurs intéressant de tracer après apprentissage les valeurs de l'erreur à chaque passage d'une sous-tâche à l'autre, afin de mettre en évidence la loi de décroissance obtenue.

Précisons que des algorithmes intermédiaires entre les deux algorithmes décrits ci-dessus pourraient être de même développés.

*

La conception d'algorithmes d'apprentissage progressif dans les réseaux demeure donc un problème ouvert. Nous avons tenté de dégager quelques idées importantes pour s'attaquer à ce problème, telles que la définition de la notion de tâche, la nature relative de la difficulté, l'interprétation en termes de surfaces d'erreur, la détermination des critères de passage. Le principal travail qui reste à réaliser concerne certainement une formulation plus rigoureuse du problème qui puisse ouvrir la voie vers des solutions générales et performantes.

III.2 Les apports de la psychologie

L'idée de réaliser un apprentissage progressif dans les réseaux de neurones provient d'une part d'une analyse "systémique" de l'apprentissage dans les réseaux de neurones (apprentissage unidirectionnel versus bidirectionnel) et d'autre part de l'idée intuitive selon laquelle l'homme réalise effectivement un apprentissage progressif. A partir de là a pu être menée une réflexion concernant la conception de stratégies d'apprentissage pour les réseaux connexionnistes (chapitre III.1.3).

L'objectif maintenant est de dépasser le stade "intuitif" concernant les apports de la psychologie dans la conception d'algorithmes, autrement dit aller au delà de la psychologie "de tous les jours" et introspective. Pour cela, on se propose d'examiner les données scientifiques de la psychologie cognitive.

Il s'agit de s'inspirer de ces données pour concevoir de nouveaux algorithmes. Cette démarche se distingue de la *modélisation* psychologique, qui consiste à reproduire des comportements psychologiques à l'aide d'un ordinateur. Elle n'en est pas moins tout aussi importante dans le cadre des sciences cognitives.

III.2.1 Progression de l'apprentissage dans les théories de l'automatisation

Il a été mis en évidence au paragraphe III.1.3.e que la notion d'apprentissage progressif était lié à une *limitation* du système de traitement de l'information qui ne pourrait apprendre qu'un tâche facile, relativement à ses performances. Cette relation entre *progression* et *limitation* s'avère pertinente pour certains types d'apprentissage humains, que nous avons décidé d'expliciter dans ce qui suit.

III.2.1.a Théories psychologiques de l'automatisation

En psychologie, plusieurs théories de l'apprentissage, et plus généralement de traitement de l'information chez l'homme, envisagent l'existence de deux types de traitements de l'information distincts : les processus contrôlés et les processus automatiques (selon la terminologie de [Schneider & Shiffrin 77]). Intuitivement, les processus automatiques sont comme des réflexes, car ils s'activent sans qu'on y prête nécessairement attention, et sans effort cognitif, comme la marche, le geste d'écriture, la récitation des tables de multiplications ; les processus contrôlés correspondent à une activité consciente et réfléchie, comme la réalisation d'un problème de mathématique difficile ou le jeu d'échec en milieu de partie.

La distinction entre ces deux types de processus a été particulièrement approfondie dans [Schneider & Shiffrin 77] et depuis largement utilisée en psychologie cognitive [Perruchet 88a]. Toutefois, il n'y a pas de consensus en la matière et d'autres théories, plus ou moins proches, impliquant ces deux processus ont été élaborées ([Logan 88] et[Anderson 90] par exemple).

Dans [Schneider & Shiffrin 77], la distinction entre traitements automatiques et contrôlés est mise en évidence dans des tâches de "recherche " : on présente dans un premier temps au sujet un ensemble de lettres cibles, puis on lui présente un ensemble de lettres en lui demandant de déterminer le plus rapidement possible la présence ou l'absence d'une lettre cible parmi ces lettres. Dans certains cas, le temps de traitement va être proportionnel au nombre de lettres cibles et de lettres présentées, ce qui laisse supposer que chaque lettre est examinée l'une après l'autre (traitement contrôlé), tandis que dans d'autres cas, le temps de réponse est constant, ce qui laisse supposer que le cerveau traite toutes les informations en parallèle (traitement automatique). On pourra consulter [Camus 88] pour un exposé détaillé de ces expériences de "recherche".

D'autres méthodes ont été utilisées pour mettre en évidence ces deux types de processus (voir [Perruchet 88b] pour une revue de ces méthodes). Tous les travaux mettent en évidence les caractéristiques opposées des deux types de traitements, que nous avons résumées dans le tableau III.7. Ce tableau totalement "noir et blanc" est en fait caricatural, car la frontière entre processus contrôlés et automatiques est bien plus subtile.

- Les processus automatiques sont des processus qui se *déclenchent* sans contrôle intentionnel, c'est-à-dire involontairement, alors que les processus contrôlés sont activés de manière consciente et contrôlée.

- Les processus automatiques se déroulent sans qu'on y prête attention, contrairement aux processus contrôlés ; cependant, au moins en ce qui concerne certains processus automatiques, le sujet humain peut être conscient du déroulement du traitement, et peut, certes avec effort, le modifier ou l'interrompre de manière contrôlée [Camus 88][Perruchet 88b][Barsalou 92 p. 62]. Selon [Logan 88], les processus automatiques pourraient être contrôlés, mais plus difficilement du fait de leur rapidité.
- Les traitements automatiques peuvent se déclencher en parallèle [Shiffrin et Schneider 77][Richard *et al.* 90] : plusieurs processus peuvent fonctionner en même temps ; ainsi, je peux à la fois marcher et chanter une chanson que je connais bien, tout en recherchant du regard une personne donnée. A l'inverse, les processus contrôlés sont intrinsèquement sériels : chaque traitement s'applique successivement ; ainsi, je ne puis résoudre deux problèmes mathématiques en même temps.
- Les processus contrôlés s'appuient sur une *mémoire de travail* ou *mémoire à court terme* dont la capacité est *limitée* [Schneider & Shiffrin 77], en nombre de patterns et en temps (par opposition à la *mémoire à long terme* qui contient l'ensemble de nos connaissances). Par conséquent, ils ne peuvent traiter qu'un nombre limité de données à la fois, contrairement aux processus automatiques [Schneider & Shiffrin 77]. La distinction automatique/contrôlé est donc étroitement liée à l'étude de la mémoire, un pan extrêmement vaste de la psychologie cognitive qu'il n'est pas possible d'aborder ici. [Logan 88] étudie d'ailleurs les processus automatiques sur le plan de la mémoire. Précisons que même si nous employons le terme "mémoire à court terme", introduit dans les années soixante, il est établi par des études plus récentes que plutôt qu'une structure séparée, la mémoire à court terme serait plus un *focus* sur la mémoire à long terme.
- De par la nature attentionnelle et sérielle des processus contrôlés, ceux-ci s'exécutent lentement, alors que les processus automatiques s'exécutent rapidement.
- Par contre, la *mise en place* d'un processus contrôlé est rapide [Shiffrin et Schneider 77], alors que les processus automatiques se construisent très lentement, selon un processus que nous allons détailler plus loin.
- De manière corollaire, les processus automatiques sont spécifiques d'un stimulus donné et rigides, alors que les processus contrôlés sont souples [Allen & Brooks 91]. Cependant, il convient de nuancer car les processus automatiques peuvent être par exemple supprimés [Camus 88].
- Les processus automatiques sont indivisibles [Shiffrin et Dumais 81] : on ne peut utiliser la moitié d'un automatisme.
- Enfin, un traitement automatique donne une réponse sans que le sujet puisse *expliquer* pourquoi et selon quel cheminement cette réponse est donnée [Barsalou 92], contrairement aux processus contrôlés.

Examinons maintenant comment coopèrent les deux types de traitement. On constate que les processus automatiques et contrôlés sont très complémentaires : sans les premiers, aucun traitement rapide et complexe n'est possible ; sans les seconds, aucune réaction à une situation inattendue n'est envisageable. Cependant, dans la plupart des cas, confronté à une situation, le sujet n'utilise pas soit des processus automatiques, soit des processus contrôlés, mais une *combinaison* de ces deux types de traitements [Shiffrin & Dumais 81]. Un certain nombre de prétraitements sont effectués automatiquement, dont les résultats sont traités de manière contrôlée, ce qui génère des sous-problèmes et sous-buts, eux-mêmes traités automatiquement, etc.

PROCESSUS CONTROLÉS	PROCESSUS AUTOMATIQUES
déclenchés volontairement	déclenchés involontairement
attentionnels	non attentionnels
sériels	parallèles
liés à la mémoire à court terme	indépendants de la mémoire à court terme
exécution lente	exécution rapide
mise en place rapide	mise en place lente
souples	rigides et spécifiques
divisibles	indivisibles
mécanismes explicables	mécanismes non explicables

Tableau III.7. Caractéristiques caricaturales des deux types de processus (voir texte)

Nous allons maintenant étudier de plus près la coopération des deux types de processus sur le plan de l'apprentissage, car c'est à ce niveau qu'intervient la notion d'apprentissage progressif.

Dans la présente étude, on met de côté l'ensemble des automatismes innés (prétraitements perceptifs) pour se consacrer à ce que l'on appelle les processus automatisés. Ces processus automatisés se constituent par un mécanisme dénommé *automatisation*.

L'automatisation consiste en un passage d'un traitement contrôlé à un traitement automatique pour un même stimulus ou une même famille de stimuli. Avant automatisation, ce sont les processus contrôlés qui sont activés pour fournir une réponse appropriée à un stimulus donné. Ces processus utilisant des ressources limitées du fait de la limitation de la mémoire à court terme, le système cognitif humain va devoir ne prendre en compte qu'un nombre réduit de patterns, c'est-à-dire effectuer un découpage, une partition de l'environnement [Ronco 94].

Une fois cette partition effectuée, au fur et à mesure qu'un même stimulus est présenté et que la réponse est donnée par les processus contrôlés, une procédure d'automatisation s'enclenche qui construit petit à petit l'automatisme qui va finalement remplacer les processus contrôlés pour le traitement du stimulus en question [Schneider 85 *in* Camus 88]. Les processus contrôlés sont ainsi libérés pour d'autres tâches [Schneider & Fisk 83 *in* Camus 88] ; ils peuvent alors être appliqués sur une nouvelle partition tout en réutilisant les traitements qui viennent d'être automatisés. Cette procédure d'automatisation se répète de nombreuses fois, permettant au système cognitif de traiter des patterns de l'environnement de plus en plus complexes [Shiffrin et Schneider 77]. L'apprentissage consiste donc en une succession d'automatisations.

Deux facteurs sont essentiels pour que l'automatisation ait lieu [Schneider & Shiffrin 77] :

- la fréquence de cooccurrence : les paires stimulus-réponse doivent se présenter souvent afin que l'automatisme se crée ;
- la consistance de l'association : à un stimulus donné doit toujours être associé la même réponse.

Notons que la notion d'automatisation apparaît comme une donnée assez solide en psychologie cognitive, car elle est abordée, en des termes qui peuvent certes varier, par de nombreuses théories :

- dans [Schneider & Shiffrin 77] est exposée la théorie classique de l'automatisation que nous avons relatée ci-dessus ;
- dans [Logan 88], l'automatisation est abordée sur le plan de la mémoire ; mais l'aspect limité des traitements algorithmiques (contrôlés) est remis en cause ;
- dans [Anderson 90], l'automatisation est étudiée en tant que *compilation* de connaissances, c'est-à-dire en tant que passage de connaissances déclaratives à des connaissances procédurales.
- Dans [Fitts 64 *in* Rosenbaum 91] est mis en évidence trois phases dans l'apprentissage moteur : la phases cognitive ou verbale, la phases associative et la phases autonome, ou automatisée.
- Dans [Hayes & Broadbent 88], l'apprentissage sélectif est proche de l'apprentissage par automatisation, puisqu'il s'appuie sur une "mémoire de travail abstraite", sous-partie de la mémoire à court terme.
- Dans [Newell & Rosenbloom 81], on introduit la théorie du "chunking" où des regroupements d'éléments de l'environnement ("chunks") sont créés afin d'augmenter les capacités de traitement, car un "chunk", même s'il regroupe plusieurs éléments, occupe une seule "case" dans la mémoire à court terme.

D'autres études concernant l'acquisition de l'expertise confirment l'idée d'une automatisation. Dans [Murphy & Wright 84], les auteurs comparent les performances de novices et d'experts. Les résultats montrent que le novice prend en compte moins de traits caractéristiques que l'expert, et que ces traits sont les traits les plus distinctifs. Ceci est tout à fait cohérent avec l'idée que le novice utilise davantage les processus contrôlés, limités en capacités, que l'expert. De plus, le novice est plus analytique alors que l'expert raisonne de manière holistique [Giacometti 92].

Pour bien comprendre comment s'effectue l'apprentissage progressif selon les théories de l'automatisation, il est essentiel d'examiner de plus près comment l'environnement est découpé par les processus contrôlés :

- Le découpage est contraint par les limitations de la mémoire à court terme. Il s'agit d'une part d'une limitation en nombre : de nombreuses expériences montrent que si l'on donne une série de stimuli à un sujet humain (par exemple on lui dit des chiffres) puis on lui demande de rappeler ces chiffres, alors celui-ci ne pourra rappeler en moyenne que 7 éléments (le chiffre 7 est appelé *empan mnésique*). D'autre part, la mémoire à court terme est aussi limitée en temps : les patterns ne peuvent être rappelés que dans les secondes qui suivent le stockage (d'où le nom "mémoire à court terme").
- Les processus contrôlés se caractérisent essentiellement en ce qu'il sont capables de s'assigner un but [Barsalou 92] avant de produire la séquence d'actions permettant d'atteindre ce but. Les limitations de la mémoire à court terme vont donc amener les processus contrôlés à décomposer le problème en buts, sous-buts, sous-sous-buts, etc., afin de se ramener à des problèmes pouvant être résolus étant données les limitations de la mémoire à court terme.
- Les processus contrôlés, attentionnels, sélectionnent dans l'environnement les traits considérés comme les plus *pertinents* (la sélection de traits non pertinents est possible et gêne l'automatisation [Schneider & Shiffrin 77]). Ceci est confirmé par l'étude de [Murphy & Wright 84] (voir ci-dessus) où il est montré que ce sont les traits les plus pertinents qui sont d'abord pris en compte.

*

L'apprentissage par automatisations successives est donc un exemple d'apprentissage progressif chez l'homme. Il repose essentiellement sur la notion de limitation de la mémoire à court terme, limitation qui paradoxalement ne freine pas l'apprentissage mais permet au contraire un apprentissage progressif.

Précisons cependant que certains apprentissages ne relèvent pas de l'automatisation. C'est le cas de l'apprentissage implicite [Reber 76][Cleermans 93] ou

l'apprentissage non sélectif [Hayes & Broadbent 88] : dans certaines conditions, quand l'environnement possède beaucoup de caractéristiques sans qu'il soit possible d'en dégager les plus pertinentes, un apprentissage peut quand même s'effectuer, sans passer par les processus contrôlés, en construisant directement un processus automatique adapté.

III.2.1.b Analogie automatisme - réseau de neurones formels

Ayant exposé en quoi, selon la théorie de l'automatisation, le système cognitif humain effectue un apprentissage progressif, nous souhaitons utiliser ces données pour le champ des réseaux connexionnistes. Cela amène à énoncer deux hypothèses de travail concernant les processus automatiques et contrôlés :

- On suppose tout d'abord que les processus contrôlés et automatiques constituent deux traitements séparés, c'est-à-dire qu'ils peuvent s'exécuter indépendamment l'un de l'autre et qu'ils ne communiquent que par les résultats de leurs traitements. Chaque type de traitement est donc réalisé par un *module*. Notons qu'en ce qui concerne les processus automatiques, le module en question peut se subdiviser en plusieurs modules correspondant chacun à un traitement automatique.
- Dans la mesure où il y a deux modules distincts, l'automatisation, c'est-à-dire le passage du traitement contrôlé au traitement automatique, s'opère uniquement par les patterns entrant et sortant du module contrôlé. Or le module contrôlé effectue ces traitements à l'aide d'une mémoire à court terme, ou mémoire de travail. La communication entre les deux modules s'effectue via cette mémoire de travail. Autrement dit, le processus automatique ne se construit pas à partir des mécanismes de traitement utilisé par les processus contrôlés, mais seulement grâce aux *résultats* des traitements, traitements intermédiaires compris (voir figure III.15).

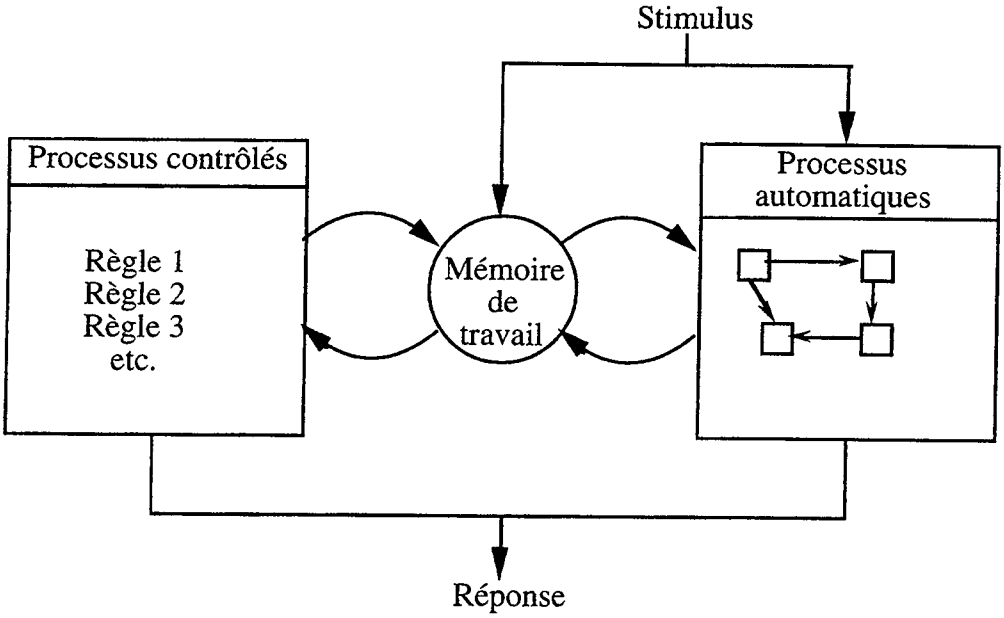

Figure III.15. Les deux types de processus en tant que deux modules

Ensuite, on propose l'analogie suivante : en tant que technique d'apprentissage par l'exemple, un réseau de neurone formel de type perceptron (monocouche ou multicouche) est un bon candidat pour modéliser un traitement automatique.

Cependant, il existe beaucoup de techniques d'apprentissage par l'exemple qui elles aussi sont "candidates". Nous allons donner quelques arguments en faveur des perceptrons multicouches (*PMC*). On se propose de comparer les *PMC* sur le plan de la similitude avec les traitements automatiques d'une part avec les arbres d'induction, qui consistent à classer des patterns à l'aide de tests concernant chacun des traits, tests ordonnés de manière optimale, et d'autre part avec les techniques mémorisant des exemples ou des prototypes, éventuellement implantées de manière neuronale : "Instance Based Learning", "k Nearest Neighbour", "Radial Basis Functions", Réseaux à Prototypes [Giacometti 94].

Premièrement, le temps d'exécution d'un arbre de décision, *quelle que soit l'implantation*, croît avec le nombre de traits. Par contre les PMC et les techniques à base d'exemples mémorisés, si ils sont implantés de manière parallèle, ne sont pas plus lents quand il y a plus de traits. C'est typiquement le cas des traitements automatiques, capables de traiter un grand nombre de caractéristiques avec une grande rapidité. Les arbres d'induction apparaissent donc comme de moins bons candidats pour modéliser des processus automatiques.

Deuxièmement l'apprentissage avec les arbres d'induction nécessite un seul passage de la base d'exemples ; pour les réseaux à base d'exemples mémorisés quelques passages suffisent [Alpaydin 91] (sauf quand on les combine avec des techniques à base

du gradient). A l'inverse les PMC nécessitent de nombreuses présentations des exemples pour apprendre. Ceci est généralement considéré comme un point faible des PMC, mais comme précisément la création d'un automatisme est un phénomène lent et sensible à la fréquence de présentation des mêmes patterns (voir ci-dessus), la lenteur des PMC est intéressante de ce point de vue. Cependant, on peut tout de même considérer que les arbres d'induction et les algorithmes à base d'exemples mémorisés sont aussi à retenir pour l'automatisme, dans la mesure où un autre mécanisme freinerait leur apprentissage.

Troisièmement, il est tentant de rapprocher la non explicabilité des processus automatiques et celle des PMC. Contrairement aux PMC, les arbres d'induction peuvent expliquer leurs traitements en termes de tests successifs sur chacun des traits, et les algorithmes à base d'exemples mémorisés en termes d'exemple(s) ou prototype(s) le(s) plus proche(s). Encore une fois, on pourra toujours contourner cet argument en supposant qu'un mécanisme supplémentaire empêche de fournir l'explication dans ces deux derniers algorithmes.

Les trois arguments discutés ci-dessus, sans être radicaux, encouragent à faire l'analogie entre un automatisme et un PMC, plus qu'avec d'autres techniques d'apprentissage par l'exemple. Ainsi, c'est la manière dont les processus contrôlés traitent les données de l'environnement, avec les contraintes qui ont été discutées au paragraphe précédent (III.2.1.a), qui détermine quels exemples sont fournis au PMC. L'analogie PMC - automatisme va donc permettre d'utiliser ce que l'on sait de la résolution de problèmes par les processus contrôlés pour envisager des stratégies d'apprentissage progressif.

Pour se rapprocher davantage des théories de l'automatisation, une voie importante de recherche [Ronco 94][Szilas & Ronco 95] est de considérer qu'un automatisme est analogue à un *module* connexionniste (au sens des réseaux modulaires de [Jacobs *et al.* 91]) et que l'apprentissage revient à construire incrémentalement un réseau de modules . De tels réseaux voient le jour actuellement [Chen 93]. Nous écartons cette étude du présent travail, car cela nous éloignerait de l'objectif initial, à savoir la sélection active et progressive des données de l'environnement.

Notons que l'analogie PMC - automatisme a aussi été abordée dans [Handelman *et al.* 92], dans le cadre de la modélisation de l'apprentissage moteur. On y retrouve deux processus distincts, l'un utilisant une base de connaissance et l'autre un réseau de neurones formels, le premier faisant apprendre le deuxième. Mais l'aspect progressif n'est pas abordé. Dans [Schmidhuber 92a], on évoque en quelques lignes une analogie

entre le principe de compression historique introduit dans l'article et le principe d'automatisation chez l'homme.

III.2.1.c Apports pour les réseaux connexionnistes

Après avoir défini une analogie entre l'apprentissage humain et l'apprentissage artificiel quelles idées d'algorithmes peut-on en déduire ?

limiter le nombre de patterns

La limitation de la mémoire à court terme en nombre de patterns suggère un algorithme où le nombre de patterns nouveaux à apprendre serait limité. On aboutit alors à un algorithme proche de l'algorithme *IST* décrit au paragraphe III.1.2.b, qui consiste à ajouter petit à petit des exemples dans la base d'exemples, algorithme dont les limites ont déjà été discutées ci-dessus.

Pour les réseaux temporels, limiter le nombre de patterns entrant revient à limiter la longueur des séquences temporelles apprises. Comme il a été observé au paragraphe III.1.2.b, c'est une stratégie classique pour entraîner les réseaux : augmenter progressivement la longueur des séquences.

En ce qui concerne les réseaux mécaniques CORDIS, cette stratégie est envisageable, mais demeure limitée : on sait par exemple que pour un signal périodique, toute l'information est contenue dans une période ; il n'est donc pas pertinent d'augmenter progressivement la longueur au delà d'une période.

Limiter le nombre de patterns ne se limite cependant pas à prendre les *premiers* patterns. On peut imaginer un algorithme capable de se focaliser sur une zone du signal contenue dans un intervalle temporel limité, et apprendre cette zone en particulier. Pour éviter que le réseau ne dévie trop du comportement désiré *avant* que cette zone soit atteinte, il faut envisager de forcer ou coupler le réseau jusqu'à la zone de focus. D'autres algorithmes sont aussi envisageables, dans lesquels les patterns appris, en nombre limité, ne sont pas contigus.

Quoi qu'il en soit, la limitation en nombre n'est pas un critère suffisant pour sélectionner les exemples.

Limiter le nombre de traits

Les résultats concernant la différence entre les novices et les experts (voir paragraphe III.2.1.a) semblent applicables à un réseau connexionniste. L'idée est de limiter à chaque étape le nombre de traits nouveaux, et de prendre en compte d'abord les traits les plus significatifs. Classer les traits des plus significatifs aux moins significatifs peut se faire par différentes méthodes ; l'une d'elle consiste à reprendre les critères utilisés par les arbres d'induction, comme par exemple l'entropie de l'algorithme *ID3*. On peut donc proposer l'algorithme suivant :

```
DEBUT
    Initialiser p
    q <- 0
    Classer les traits du plus significatif au moins significatif
    Répéter
        q <- q+p
        q <- Min(q,Nombre de traits)
        Sélectionner un sous-ensemble où seuls les q premiers
            traits varient
        Apprendre ce sous-ensemble
    Jusqu'à q = Nombre de traits
FIN
```

La valeur de p sera choisie petite, par exemple *3* ou *4*. Cet algorithme est particulièrement intéressant pour une approche par "query" (voir paragraphe III.1.2.a), car il est alors possible de "demander" à l'environnement la réponse pour une entrée ayant certains traits constants.

Pour l'apprentissage dans les réseaux CORDIS, cet algorithme n'est bien sûr pas envisageable dans l'immédiat, puisque toutes les expériences réalisées jusqu'à présent n'ont qu'une seule entrée, donc qu'un seul "trait".

Décomposition en sous-buts

La décomposition en buts et sous-buts est le propre des traitements contrôlés. Si le problème est hiérarchisé, cette stratégie peut être intéressante : on présenterait d'abord des problèmes élémentaires, puis des problèmes plus complexes, réutilisant les résultats des problèmes élémentaires.

Dans une structure unidirectionnelle, cette stratégie ne peut s'appliquer telle quelle, car un réseau ne peut réutiliser ses sorties. Il est alors nécessaire de rentrer dans

la structure du réseau pour le rendre modulaire et hiérarchique : plusieurs sous-réseaux sont entraînés séparément pour finalement former une structure entière, comme cela est souvent fait en traitement de la parole (on s'intéresse aux syllabes, puis aux mots, puis aux phrases). Pour pouvoir réellement appliquer une stratégie de décomposition en sous-buts, il faudrait envisager un système de classification où les sorties rebouclent sur l'entrée dans le but d'enchaîner les traitements des sous-buts successifs.

Dans le cas des réseaux CORDIS, ce type de stratégie ne s'applique pas, étant donnée la nature non hiérarchique du problème.

III.2.2 Apprentissage "élève-professeur"

Nous intéressant à l'interaction entre un système cognitif et son environnement, l'interaction entre un sujet humain apprenant et un sujet humain enseignant mérite une attention toute particulière. Dans ce vaste sujet, trois domaines vont être abordés : la didactique, le développement de l'enfant et l'apprentissage moteur. Ensuite, nous tenterons d'en extraire des idées pour la conception d'algorithmes interactifs.

III.2.2.a Quelques données sur l'interaction apprenant-enseignant

La didactique

La didactique est la discipline qui se penche sur la manière d'organiser l'enseignement de connaissances données (exemple : connaissances mathématiques) afin d'optimiser l'apprentissage du sujet apprenant.

La relation élève-professeur est bien sûr au cœur de la didactique. Il y a en particulier deux types de relations [Laborde 92] : le *diagnostic*, qui permet au professeur de connaître l'élève, sa "représentation", afin de poser le problème adéquat, et le *problème*, qui a pour but de faire évoluer la représentation de l'élève.

Le choix du problème est donc la problématique centrale en didactique. Il est lié aux connaissances que possède l'élève à un stade donné, d'où l'importance du diagnostic. En particulier, le problème doit viser à l'assimilation d'une connaissance ni trop facile, auquel cas l'élève ne va pas évoluer, ni trop inconnue et difficile, car il y a alors un risque de *rejet cognitif* par l'élève. Par exemple, l'enseignant pourra poser

un problème que les connaissances de l'élève permettent de résoudre mais de manière très coûteuse ; ainsi, l'élève cherchera une méthode de résolution plus efficace.

Au delà de ces données très générales, la didactique nous apporte finalement relativement peu d'idées pour notre propos, dans la mesure où les connaissances enseignées sont souvent conceptuelles, et donc éloignées des réseaux de neurones formels.

De plus, les expériences et théories didactiques sont généralement relatives à une discipline particulière : il s'agit de la didactique des mathématiques, la didactique de la physique, etc. Il y a peu de résultats *généraux* susceptibles d'être applicables aux réseaux de neurones formels.

Le développement de l'enfant

Des études ont été réalisées concernant l'apprentissage chez l'enfant en interaction avec l'adulte.

On montre [Bruner 83 *in* Winnykamen 90] que si les activité ou les discussions proposées par l'adulte sont trop éloignées des capacités de l'enfant, alors la progression ne peut se réaliser. Ainsi, la mère agit à un niveau de réalisation légèrement supérieur à la compétence de l'enfant.

De plus, diverses activités d'étayage lors de la réalisation de la tâche par l'enfant vont être utilisées en vue :

- d'éveiller l'intérêt,
- de diminuer la difficulté,
- de maintenir l'orientation vers le but ou sous-but,
- de signaliser les caractéristiques déterminantes,
- de démontrer.

L'imitation joue un rôle important dans le développement de l'enfant. L'imitation ne se limite pas au fait de reproduire un comportement donné, car le modèle, l'adulte, modifie son propre comportement en fonction de la production du sujet imitant (l'enfant) [Winnykamen 90]. Ainsi, l'apprentissage par imitation procède par *interaction* entre le novice et l'expert. Un exemple d'une telle interaction est celui de l'apprentissage du langage au cours duquel la mère s'exprime parfois à son enfant selon un langage spécifique adapté au langage de l'enfant [Elman 93].

Les activités de l'expert, dont le but est de maintenir l'interaction en vue de faire progresser le novice, sont complexes : elles se situent à la fois sur le plan cognitif et sur le plan motivationnel et affectif ; elles font intervenir un *modèle* du novice [Winnykamen 90].

L'apprentissage moteur

L'apprentissage moteur est un domaine de recherche très actif, présentant l'intérêt d'autoriser de nombreuse expériences dont les résultats sont directement observables, applicables à l'entraînement des sportifs. La nature continue des activité motrices les rend particulièrement intéressantes dans l'optique d'une modélisation connexionniste.

De nombreuses théories ont été élaborées concernant l'apprentissage moteur (voir [Rosenbaum 91]), dont nous exposons ici uniquement les aspects les plus intéressants relativement à notre problématique.

Tout d'abord, la notion d'apprentissage hiérarchique est omniprésente en apprentissage moteur. On montre en effet comment un mouvement peut être décomposé en sous-mouvements élémentaires pouvant être appris séparément.

Par ailleurs, la théorie de Fitts, qui décrit l'apprentissage moteur (voir paragraphe III.3.1.a), est très proche de la théorie de l'automatisation, puisqu'elle considère l'apprentissage comme le passage d'une phase verbale, attentionnelle, à une phase automatisée. Il est intéressant de noter qu'au début de l'automatisation, un guidage pas trop fort peut aider l'apprentissage [Famose *et al.* 91].

En ce qui concerne l'apprentissage moteur, il faut aussi rappeler que certains changements physiques ont lieu au cours de l'apprentissage [Rosenbaum 91]. D'une part les muscles évoluent, d'autre part le nombre de degrés de liberté évolue avec le temps.

Examinons maintenant de plus près comment un professeur peut faire apprendre à un élève certaines activités motrices. On constate que le principe de base en apprentissage moteur est de doser la difficulté [Famose 90]. Ainsi, si l'on demande à des sujets de répéter une tâche difficile à réaliser, on ne constate pas d'amélioration ; au contraire, si on gradue la difficulté, on peut constater une augmentation très sensible de la performance.

Il s'agit, selon [Famose 90], d'adapter le niveau de difficulté de la tâche au niveau d'habileté du sujet. Ainsi, la tâche doit être à la portée du sujet. Notons que des facteurs émotionnels interviennent aussi dans la définition de la tâche, tels l'ennui, l'inquiétude.

Un premier ensemble de méthodes pour graduer la difficulté est de décomposer la tâche en sous-tâches. Cette stratégie a été étudiée depuis fort longtemps, puisque dès 1897, des expériences ont été réalisées en faisant apprendre le télégraphe de manière hiérarchique : on apprend d'abord les points et les traits, puis les lettres, et enfin les mots [Bryan & Harter 1897 *in* Rosenbaum 91].

Un second ensemble de méthodes consiste à changer les conditions environnementales, comme le poids d'un objet, la position du corps et divers autres paramètres de la tâche.

III.2.2.b Apports de ces données pour la conception d'algorithmes d'apprentissage progressif

L'analogie entre l'apprentissage humain et les réseaux de neurones est simple dans ce cas : sur la figure III.8, le "réseau de neurone supervisé" correspond à l'apprenant et le "module actif" à l'enseignant.

Les données qui ont été collectées concernant l'apprentissage "élève-professeur" convergent toutes vers l'idée *d'interaction* [Famose 90][Winnykamen 90][Laborde 92]: non seulement la tâche proposée à l'élève modifie celui-ci, mais aussi l'état de l'élève modifie cette tâche via le professeur. Cette interaction, représentée sur la figure III.16, confirme l'importance que nous avions accordé dans le paragraphe III.1.3.d à l'action *réflexive*, fonction du comportement même du réseau.

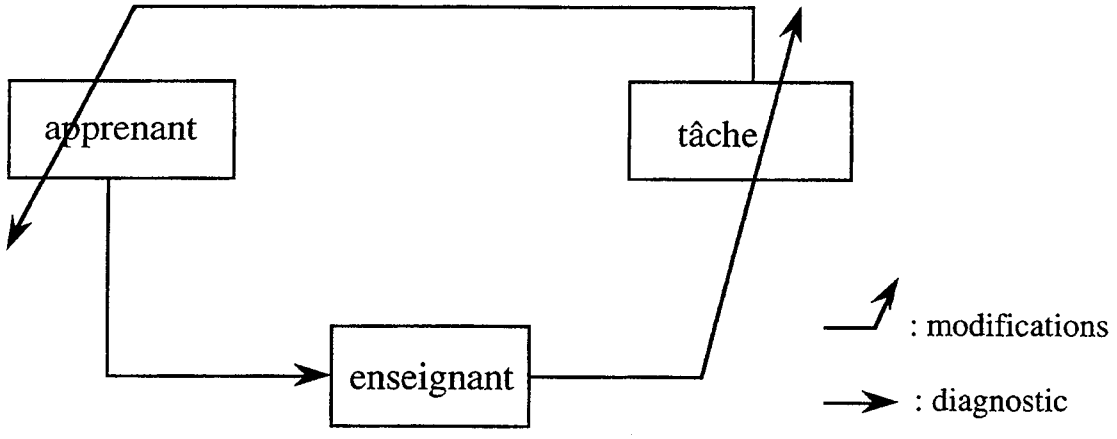

Figure III.16. Interaction dans l'apprentissage humain

Un autre point commun à la didactique, le développement de l'enfant et l'apprentissage moteur concerne la définition de la tâche lors de l'interaction. La tâche doit être d'une difficulté légèrement supérieure aux capacités de l'apprenant, ce qui confirme l'idée de difficulté relative introduite au paragraphe III.1.3.e. Cela suppose donc qu'un algorithme d'apprentissage actif doit d'une part être capable de réaliser un *diagnostic* du réseau adaptatif, puis de déterminer une action d'une difficulté liée au résultat du diagnostic.

Ces considérations nous amènent à proposer un "proto-algorithme", c'est-à-dire un algorithme encore général qui ne peut s'appliquer immédiatement, pour l'action dans les réseaux connexionnistes :

```
DEBUT
    Répéter
        Diagnostic du réseau
        Détermination d'une action A qui donne une tâche
            un peu plus difficile que les capacités du réseau
        Répéter
            Adapter les poids avec A
        Jusqu'à réussite
    Jusqu'à A = action cible
FIN
```

On remarquera que cet algorithme suppose que toutes les sous-tâches ont un critère de passage fixe, puisqu'on apprend jusqu'à réussite. L'idée d'un critère de passage décroissant qui est discutée au paragraphe III.1.3.f ne semble pas se retrouver en psychologie cognitive.

Le fait que l'apprentissage moteur peut être favorisé par un guidage pas trop fort en début d'apprentissage est tout à fait intéressant pour notre étude. En effet, dans le cas réseaux récurrents, un guidage peut être vu comme un *forçage* (voir paragraphe II.2.4.a). Dans le cas des réseaux mécaniques, il s'agit bien entendu d'un *couplage*. Il est alors intéressant de voir le couplage en tant que moyen de faciliter l'apprentissage : effectivement, quand on couple le réseau adaptatif avec l'objet à identifier, la différence entre les deux objets, que nous mesurons par la "différence relative moyenne" (*DRM*) est réduite.

Dans les expériences du chapitre II.5, nous avions diminué en conséquence le seuil pour considérer qu'un apprentissage était réussi, c'est-à-dire compensé la facilitation due au couplage par une complexification due à un seuil plus bas. Si on considère que le couplage est un élément facilitateur, l'idée serait donc de commencer avec un couplage assez fort, jusqu'à atteindre une *DRM* inférieur au seuil souhaité, puis diminuer l'intensité du couplage (déterminée par la raideur et la viscosité du couplage), et ainsi de suite jusqu'à atteindre un couplage nul. Cela donne par exemple l'algorithme suivant :

```
DEBUT
    Kc <- Kc0
    Répéter
        Diminuer Kc
        Répéter
            Apprendre avec couplage Kc
        Jusqu'à DRM < DRM souhaitée
    Jusqu'à Kc = 0
FIN
```

La diminution progressive du couplage possède les avantages potentiels suivants :

- le taux de réussite est augmenté (effet d'apprentissage progressif) ;
- les performances sont moins sensibles aux paramètres : la raideur initiale *Kc0* et le taux de diminution de *Kc* remplacent les valeurs de raideur et de viscosité, auxquelles les algorithmes de couplage ou de semi-couplage sont très sensibles ;
- à la fin de l'apprentissage, le couplage est nul, donc l'erreur mesurée pendant l'apprentissage correspond bien à l'erreur mesurée en utilisation, contrairement aux algorithmes de forçage et de couplage. Le problème du critère d'arrêt évoqué au chapitre II.5 est partiellement évité (partiellement car le critère d'arrêt pose d'autres problèmes pour les longues séquences, comme il a été décrit au chapitre II.5).

Des variantes sont bien sûr envisageables, en particulier en faisant intervenir la viscosité de couplage. On relèvera une certaine similitude entre cet algorithme et la variante de "teacher forcing" (forçage) proposée dans [Toomarian & Barhen 91,92], dans laquelle le coefficient de forçage se modifie selon une loi inversement proportionnelle à l'erreur commise par le réseau (voir paragraphe II.2.4.a).

III.3 Expérimentations sur l'apprentissage progressif

III.3.1 Dans les réseaux à couches

III.3.1.a Objectifs

Les expériences qui vont être décrites dans ce chapitre, qui ont été partiellement exposées dans [Ronco 94] et [Szilas & Ronco 95], se situent au même niveau que celles rapportées dans l'analyse bibliographique du paragraphe III.1.2.b : il s'agit d'exhiber des cas où une certaine présentation des données au réseau, allant dans le sens d'une complexification de la tâche à résoudre, améliore sensiblement les performances d'apprentissage. Par rapport aux études existantes, nous apportons les éléments suivants :

D'une part, nous abordons la question de l'apprentissage progressif dans les réseaux à couches, ce qui a été fort peu étudié. En effet, dans [Jacobs 88], la tâche est tout à fait élémentaire (voir paragraphe III.1.2.b) ; dans [Wieland & Leighton 88], la méthodologie de présentation des exemples n'est pas clairement explicitée ; dans [Cloete & Ludik 94] et [Ludik & Cloete 95], le critère de difficulté semble très spécifique, lié au problème considéré.

D'autre part, notre objectif est surtout de montrer que l'apprentissage progressif augmente les chances de réussite d'un algorithme d'apprentissage, autrement dit que certains problèmes peuvent être résolus par un réseau donné, doté d'une règle d'apprentissage donnée, uniquement si une stratégie d'apprentissage progressif est employée (voir paragraphe III.1.3.a). Ceci rejoint les travaux de [Elman 93]. L'augmentation de la vitesse de convergence est importante mais non primordiale pour cette étude.

Enfin, une étude *systématique* de l'influence de l'ordre sur les performances d'un apprentissage connexionniste a pu être réalisée, rendue possible par le choix d'un problème comportant très peu d'exemples, comme expliqué dans le paragraphe qui suit.

III.3.1.b Expériences systématiques sur le problème du "ou exclusif"

Il s'agit ici d'une expérience tout à fait élémentaire, qui ne constitue qu'un préliminaire aux expériences décrites au paragraphe suivant. Elle permet néanmoins de tirer quelques conclusions intéressantes.

Le problème considéré n'est pas très difficile mais a été choisi parce que sa base d'exemples ne contient que quatre éléments, ce qui permet de réaliser de nombreux essais. Il s'agit du fameux problème du "ou exclusif", problème d'association de deux variables binaires à une troisième, selon la table suivante :

Entrée 1	Entrée 2	Sortie
0	0	0
0	1	1
1	0	1
1	1	0

Tableau III.7

Ce problème a la particularité d'être non linéaire : la sortie n'est pas une combinaison linéaire des entrées. Un réseau possédant une couche cachée est donc nécessaire pour le résoudre. On représente ce type de problème sur un plan, où les deux coordonnées sont les entrées d'un réseau, comme illustré sur la figure III.17 (cette représentation est aussi utilisée sur la figure III.13). Sur la figure III.17, on voit qu'une droite ne peut séparer les deux classes : deux droites sont nécessaires, et donc une couche supplémentaire. Le réseau qui a été utilisé dans cette étude est représenté sur la figure III.18.

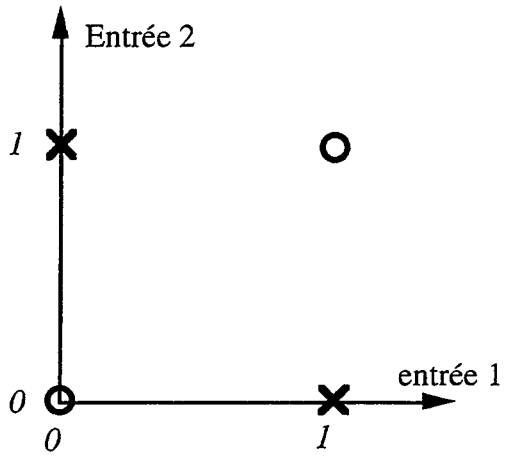

Figure III.17. Problème du "ou exclusif"

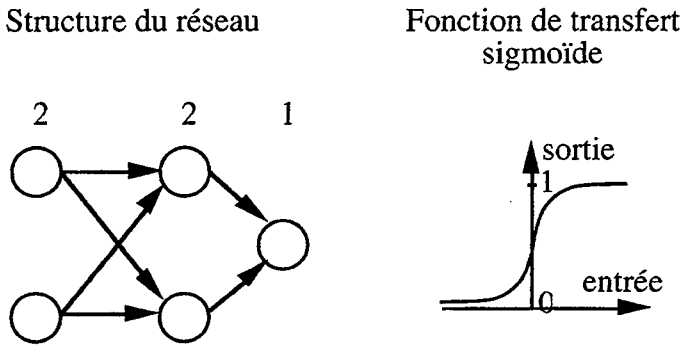

Figure III.18. Réseau utilisé pour le "ou exclusif"

L'influence de l'ordre est estimée en divisant l'ensemble d'apprentissage en deux sous-ensembles, et en faisant apprendre au réseau d'abord le premier sous-ensemble, puis les deux sous-ensembles réunis, c'est-à-dire les quatre exemples. Une telle *partition* de l'ensemble d'apprentissage n'est pas unique ; l'objectif de l'expérience est d'expérimenter la totalité des partitions possibles. L'algorithme utilisé est l'algorithme de rétropropagation de gradient, utilisé dans sa version "en ligne" : on adapte le réseau après la présentation de chaque exemple.

L'ensemble d'apprentissage comportant *4* exemples, il y a *4!=24* manières possibles d'ordonner les exemples. Pour un ordre donné, on peut soit construire le premier sous-ensemble avec le premier exemple (partition "1+3"), soit mettre les deux premiers exemples dans le premier sous-ensemble (partition "2+2"), soit mettre les trois premiers exemples dans le premier sous-ensemble (partition "3+1"), soit enfin apprendre les quatre exemples d'un coup (expérience de référence), ce qui fait au total *4* configurations. Il y a donc *24 X 4 = 96* situations différentes. Pour chacune de ces situations, l'apprentissage est reproduit pour *25* initialisations différentes des poids du réseau.

L'apprentissage de chaque ensemble ou sous-ensemble est réalisé *dans l'ordre des exemples*, et non de manière aléatoire comme il est souvent fait. Pour cette raison, les partitions {(Exemple 1 , Exemple 2),(Exemple 3 , Exemple 4)} et {(Exemple 2 , Exemple 1),(Exemple 3 , Exemple 4)} ne sont pas équivalentes. Elles sont néanmoins proches. Dans le cas d'un tirage aléatoire à l'intérieur des sous-parties, il n'y aurait plus *96* mais *15* partitions différentes.

Les autre paramètres de l'apprentissage sont le type de fonction de transfert, qui est une sigmoïde entre *0* et *1* (voir figure III.18), la pente à l'origine de cette fonction sigmoïde, *1* dans cette expérience, l'intervalle dans lequel les poids initiaux sont choisis, à savoir [-0,5 ; 0,5], les pas d'apprentissage (*0,2*) et enfin le critère de passage du sous-ensemble à l'ensemble entier.

Ce critère de passage est défini comme suit : on arrête l'apprentissage du sous-ensemble quand l'erreur absolue en sortie (valeur absolue de la différence entre la sortie et la sortie désirée) est inférieure à un certain seuil, *0,1* dans cette expérience, *pour chacun des exemples*. Ce même critère a été choisi pour déclarer l'apprentissage terminé. Ce critère est plus strict que le traditionnel seuil sur l'erreur quadratique moyenne, pour lequel un exemple mal appris peut être compensé par un exemple bien appris.

On considère que l'apprentissage a échoué quand le nombre de présentations de l'ensemble d'apprentissage (ou le sous-ensemble) dépasse *20000*. Précisons enfin que la valeur *0* du tableau III.7 est codé par *0.1* et que la valeur *1* est codée par *0.9*.

Certains de ces paramètres seront modifiés pour des expériences complémentaires décrites ci-dessous.

L'ensemble des expériences réalisées a fourni beaucoup de données, qu'il serait ennuyeux de reproduire ici. Dans ce qui suit sont exposées plusieurs analyses qui ont pu être effectuées.

Gains du découpage de l'ensemble d'apprentissage

Pour certaines partitions, on a pu constater des résultats très nets qui montrent l'intérêt de décomposer l'apprentissage du "ou exclusif".

On étudie de plus près la partition représentée sur la figure III.19. On compare la *situation globale*, où les exemples sont présentés d'un coup, avec la *situation progressive* où les exemples sont présentés deux par deux.

① (0,9 ; 0,1) ⟶ 0,9
② (0,9 ; 0,9) ⟶ 0,1
③ (0,1 ; 0,9) ⟶ 0,9
④ (0,1 ; 0,1) ⟶ 0,1

Figure III.19. Partition plus particulièrement étudiée

Dans la situation globale, *3* réseaux ont échoué, alors que tous ont convergé dans la situation progressive. Pour affiner ce résultat, *100* initialisation différentes ont été testées pour cette partition. Dans la situation globale, *22* réseaux ont échoué, contre *11* dans la situation progressive. Le taux de réussite passe donc de *78 %* à *89 %* grâce à l'apprentissage progressif.

En ce qui concerne la vitesse de convergence on obtient les résultats suivants : sur les *22* essais qui ont convergé dans la situation globale, *9637* présentations ont été nécessaires en moyenne, contre *4116* dans la situation progressive, sur les *25* essais. Cette différence est très nettement significative. On remarquera en plus qu'étant donné que dans la situation progressive, une partie de l'apprentissage se fait sur un sous-ensemble de *deux* exemples au lieu de quatre, les *présentations* décomptées ci-dessus sont en moyenne plus rapide dans le cas de l'apprentissage progressif. On aurait pu faire le décompte du nombre d'exemples présentés, ce qui aurait montré un avantage encore plus net en faveur de la situation progressive.

Le découpage de l'ensemble d'apprentissage peut donc être bénéfique, en termes de taux de convergence et de rapidité de convergence, pour un problème aussi élémentaire que le "ou exclusif", qui ne comporte que quatre exemples. Dans [Beslon & Biennier 94] est évoqué un résultat similaire pour un autre type de réseau : l'apprentissage du "ou exclusif" par parties favoriserait l'apprentissage.

Variations des paramètres

Les résultats qui viennent d'être obtenus sont ils valables uniquement pour le réseau spécifique qui a été étudié ? Pour le savoir, nous avons répliqué l'expérience sur la même partition en faisant varier certains paramètres du réseau, à savoir le pas d'apprentissage, le moment, et la pente de la sigmoïde. Les résultats sont les suivants :

Paramètres:	Pas	0.2	1	0.1	1	1	1
	Moment	0	0	0.5	0.5	0.5	0
	Pente	1	1	1	1	2	2
Global :	Moyenne	9637	1732	5225	862	388	1005
	Echecs	3	4	5	7	8	11
Progressif :	Moyenne	4116	832	2089	406	150	440
	Echecs	0	0	0	0	2	3

Tableau III.7. Résultats pour différents jeux de paramètres

Même si on est loin d'un balayage systématique de l'ensemble des paramètres, on peut constater une certaine régularité dans le ratio du nombre de présentations entre les deux situations, qui varie entre *2,08* et *2,59*.

On constate encore une fois que le taux d'échecs est bien moins fort dans le cas d'un apprentissage progressif, en particulier dans le cas où les paramètres sont élevés : le découpage de la base d'exemple permettrait au réseau de mieux supporter des pas d'apprentissage exagérément forts.

Une seule série d'essais a été réalisée avec un réseau de topologie "2-4-1" (pas *1*, moment nul, pente de la sigmoïde *2*). L'apprentissage progressif permet de passer de *7217* présentations à *3515* présentations. Le taux de réussite est par contre identique dans les deux cas et égal à *100 %*.

Ces expériences laissent donc supposer que l'amélioration des performances persiste, quelque soit le réseau utilisé.

Variations des partitions

Une partition a donc donné des résultats intéressants en faveur d'un découpage de la base d'exemples en deux parties égales. Qu'en est-il des *71* autres partitions expérimentées ?

Tout d'abord, examinons les partitions "2+2", au nombre de *24*. L'examen a priori d'un certain nombre d'entre elles permet de supposer qu'elles n'amélioreront pas sensiblement l'apprentissage. En effet, si les deux exemples appris en premier appartiennent à une même classe, alors le réseau va simplement apprendre à donner la sortie correspondant à cette classe quelle que soit l'entrée, ce qu'il va apprendre très rapidement. Ensuite, l'ajout des deux autres exemples va brutalement augmenter la complexité du problème.

On constate en effet que pour les huit partitions qui sont dans ce cas, les performances en taux de réussite et en vitesse de convergence sont quasiment identiques entre les deux situations (apprentissage progressif et global).

Pour les seize partitions restantes, les résultats ne vont cependant pas tous dans le sens de la partition de la figure III.19, bien au contraire : seulement sept partitions donnent des résultats positifs. Les neufs autres partitions donnent des résultats très négatifs : la convergence est plus lente dans la situation progressive que dans la situation globale, et surtout le taux d'échec est nettement plus élevé.

Du fait de ces neufs échecs, la moyenne des taux d'erreurs sur l'ensemble des *24* partitions "2+2" est de *5,6* dans la situation progressive, contre *3,7* dans la situation globale. En conséquence, une stratégie de sélection aléatoire des partitions, comme on le fait dans les algorithmes *CST* ou *IST* (voir chapitre III.1.2.b) serait inefficace (on ne peut prédire exactement les performances des algorithmes *CST* et *IST* dans la mesure où ceux-ci utilisent des critères de passage spécifiques pour chaque partition).

Qu'est-ce qui différencie les sept partitions positives des neuf partitions négatives ? Nous avons pu constater que les neuf partitions négatives correspondent exactement aux neufs partitions (sur les seize partitions dont il est question) pour lesquelles les entrées du réseaux ont trois fois sur quatre la valeur *0,1* pour le sous-ensemble de deux exemples, comme illustré sur la figure III.20. Nous émettons donc l'hypothèse qu'il y a une relation de cause à effet : le nombre *0,1* pour coder le zéro logique gênerait l'apprentissage progressif, selon un mécanisme qu'il ne nous est pas possible de décrire dans l'état actuel de nos connaissances. Le fait que le zéro binaire soit codé par *0,1* et le un binaire par *0,9* ne serait donc pas neutre dans les résultats. Un moyen de neutraliser cet effet serait d'utiliser un codage équivalent pour les zéros ou les uns, par exemple en utilisant deux unités pour chaque chiffre : la première entrée est active et la deuxième inactive pour coder le zéro, et l'inverse pour coder le un ; cela ferait quatre entrées en tout.

Pour les sept partitions qui ont donné des résultats positifs, la situation progressive améliore la vitesse de convergence et la taux de réussite, mais de manière moins nette pour certaines partitions que pour d'autres.

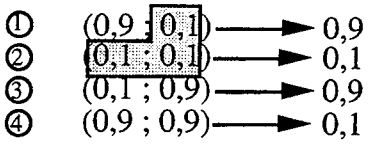

Figure III.20. Exemple de partition n'améliorant pas les performances

Ces expériences sur le "ou exclusif" ont permis de montrer non seulement l'intérêt de partitionner une base d'apprentissage mais aussi le fait que la partition devait être choisie avec soin pour ne pas risquer de gêner l'apprentissage au lieu de le favoriser.

III.3.1.c Expériences sur le problème de la double spirale

La tâche de la double spirale est une tâche de classification d'un vecteur en deux dimensions en deux classes distinctes, au même titre que le "ou exclusif" (voir figure

III.21). Chacune des classes est représentée par des points de l'espace d'entrée distribués sur une spirale ; les deux classes forment ainsi deux spirales fortement imbriquées l'une dans l'autre, représentées par *194* points.

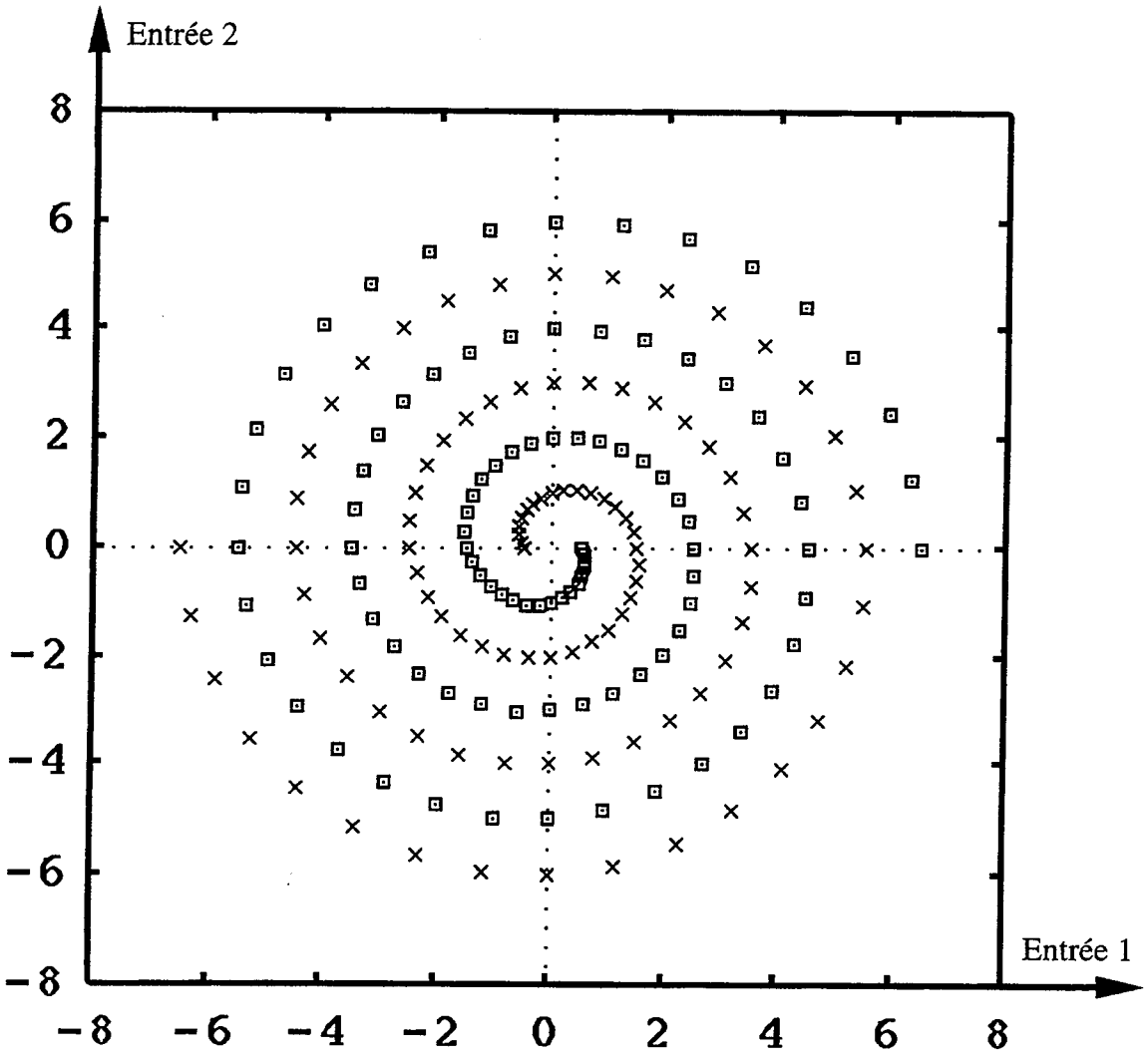

Figure III.21. Tâche de la double spirale

Pourquoi choisir cette tâche pour tester l'apprentissage progressif ? Essentiellement parce qu'il s'agit d'une tâche très difficile à résoudre pour un réseau à couches non incrémental : un réseau à rétropropagation du gradient ne parvient pas à apprendre les exemples [Baum & Lang 91][Fahlman & Lebiere 91] ; des expériences utilisant des variantes de la rétropropagation parviennent à un résultat, en un nombre de présentations très variable, selon les variantes (minimum *7600*). Dans ces expériences, le taux de réussite n'est pas mentionné ; autrement dit, on s'est contenté de montrer un ou plusieurs réseaux qui avaient réussi à atteindre une solution, sans se demander s'il s'agissait d'une réussite fortuite.

La difficulté de la tâche réside dans la forte imbrication des deux classes, ce qui impose donc de nombreuses séparations linéaires pour différentier les exemples.

On pourra reprocher à cette tâche d'être de nature "artificielle" : il n'y a que 194 exemples, les données ne sont pas bruitées, les exemples sont uniformément répartis sur le plan d'entrée. Surtout, on peut se demander si les difficultés de la tâche sont celles que l'on pourra rencontrer sur une tâche réelle. Mais l'intérêt de la tâche de la double spirale est justement qu'il s'agit d'une tâche "de laboratoire", car elle présente les deux avantages suivants :

- elle comporte peu d'exemples (*194*) relativement à sa complexité ;
- comme les stimuli sont des vecteurs à deux dimensions, une visualisation sur un plan est possible, ce qui va permettre d'ordonner manuellement les exemples dans le sens d'un apprentissage progressif.

Plus encore que pour le problème du "ou exclusif", c'est sur le taux de réussite que va porter toute notre attention : pour un certain nombre d'initialisations aléatoires différentes, on compte le nombre de cas où les *194* exemples sont appris. Il n'y a pas de test de généralisation. On se démarque ainsi de nombreuses expériences réalisées avec des réseaux de neurones artificiels, où le problème est d'une difficulté raisonnable, et où l'on montre soit une amélioration de vitesse de convergence entre l'algorithme de base et les variantes proposées, soit une amélioration des capacités de généralisation. Car nous estimons qu'il est probable que les réseaux de neurones artificiels ne savent apprendre qu'une classe très limitée de problèmes. Cela apparaît peu dans la littérature, dans la mesure où les articles publiés ne le seraient généralement pas si les résultats étaient négatifs ! Ainsi, on manque de résultats concernant les limites d'apprentissage des réseaux connexionnistes, bien que chacun sache qu'elles sont vite atteintes. Par exemple, les réseaux souffrent d'un problème de "taille" ("scaling effect" [Haykins 94]) : ils s'appliquent mal à des problèmes de grande taille (en nombre d'entrées, de sorties et d'unités) et complexes. Le taux de réussite est donc un critère d'évaluation très important.

La stratégie d'apprentissage a été la suivante : on présente les exemples *32* par *32*, dans l'ordre d'enroulement de la spirale, c'est-à-dire en commençant par l'extérieur, puis en prenant de plus en plus d'exemples à l'intérieur de la spirale (voir figure III.22). L'ensemble d'apprentissage comprend d'abord *32* exemples, puis *64*, puis *96*, puis *128*, puis *160*, puis *192* et enfin *194*. Sur la figure III.22, on peut comprendre en quoi la complexité de la tâche augmente : la première partition ne nécessite qu'une simple séparation linéaire, et serait même réalisable par un perceptron. La deuxième partition comprend d'avantages de séparations linaires pour différencier les deux arcs extérieurs des spirales. En ajoutant des exemples dans cet ordre, on augmente le nombre de séparations.

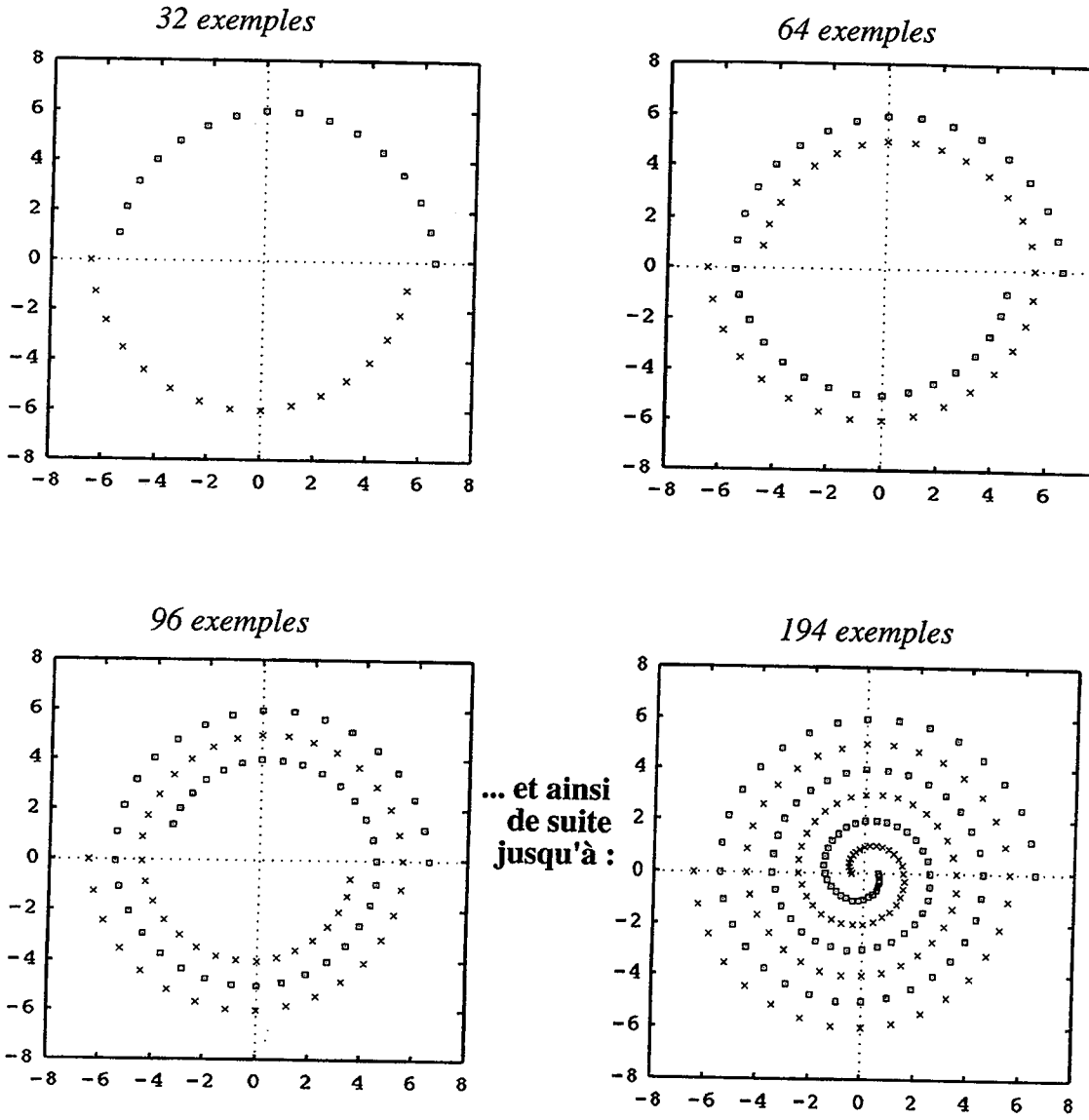

Figure III.22. Stratégie d'apprentissage pour la double spirale

Comme pour l'expérience concernant le "ou exclusif", l'ordre *à l'intérieur* de la partition est conservé (la présentation n'est pas aléatoire), on passe d'un sous-ensemble à l'autre quand l'erreur absolue sur chaque exemple est inférieure à un seuil de *0,1* et les zéro et un binaires sont codés respectivement par *0,1* et *0,9*.

Le réseau est un réseau unidirectionnel à deux couches cachées, l'une comportant *20* unités, l'autre *10*. Les pas d'apprentissage ont été fixés à *0,1* et les moments à *0,1*. Les poids ont été initialisés entre -2,5 et 2,5. On considère que l'apprentissage a échoué lorsque le nombre de présentations dépasse *100000* ; certains essais réussiraient peut-être avec plus de présentations, mais un seul essai comportant *100000* présentations nécessite déjà plus de cinq milliards d'adaptations de poids !

Les résultats sont très nets : quand on fait apprendre l'ensemble entier (apprentissage global), *4* essais convergent vers une solution, *21* échouent, ce qui

confirme les essais infructueux de [Baum & Lang 91] ; dans le cas où l'apprentissage est progressif, *20* essais convergent, *5* échouent. L'apprentissage progressif permet donc à un réseau donné, doté d'un algorithme d'apprentissage donné, de résoudre une tâche qui est insoluble quand l'ensemble d'apprentissage est présenté en entier (pour une initialisation des poids donnée).

Sur le plan du nombre de présentations, l'apprentissage progressif est aussi intéressant, puisque en moyenne, il est presque deux fois plus efficace. Ces résultats sont résumés sur le tableau III.8.

	Taux de réussite	Nombre de présentations		
		moyenne	minimum	maximum
Global	4/25	39877	17661	91591
Progressif	20/25	21948	10340	81192

Tableau III.8. Résultats pour la double spirale

Des expériences complémentaires ont été réalisées :

- on a montré que si les exemples sont ajoutés aléatoirement, comme pour l'algorithme *IST* de [Cloete & Ludik 94a], l'apprentissage échoue (on se retrouve dans la situation décrite par la figure III.13) ;
- les essais en parcourant la double spirale dans le sens inverse, en partant du centre, se sont révélés infructueux ;
- des expériences en changeant les seuils de passage d'une partition à l'autre ont donné des résultats similaires en moyenne aux résultats ci-dessus, en particulier avec la séquence de seuil : *(0.1, 0.1, 0.5, 0.5, 0.3, 0.1)*. Même si les moyennes sont proches, sur *chaque* réseau, c'est-à-dire pour chaque initialisation de poids, les performances sont très différentes pour les deux séquences de seuils.

III.3.1.d Conclusions

Les expériences décrites mettent clairement en évidence l'apport d'une stratégie d'apprentissage de type "progressif" dans l'apprentissage des réseaux à couches.

D'une part on montre que l'apprentissage progressif augmente le taux de réussite d'un apprentissage. Dans un apprentissage vraiment difficile, cette augmentation est très sensible, si bien que l'on peut affirmer que l'apprentissage progressif fait réussir l'apprentissage.

D'autre part, la rapidité de convergence est accrue, puisque le nombre de présentations de l'ensemble d'apprentissage diminue d'un facteur 2 environ.

De plus, il a été aussi mis en évidence que ce résultat n'était valable que pour certaines partitions de l'ensemble d'apprentissage, pas pour d'autres. Si la partition est mal choisie, on obtient soit aucun changement important dans les résultats soit une détérioration catastrophique des performances. Découper un ensemble d'apprentissage en sous-ensembles successivement appris peut donc à la fois être bénéfique ou néfaste. Il s'agit donc d'une stratégie à appliquer avec la plus grande précaution.

Ces expériences ont aussi montré l'importance des seuils de passage d'un sous-ensemble à un autre. On comprend bien que pour une sous-tâche donnée, si ce seuil est trop faible, l'apprentissage est inutilement lent (on apprend trop bien une connaissance partielle) et le réseau risque de se spécialiser dans la sous-tâche, et que si ce seuil est trop élevé, on perd tout l'avantage d'un apprentissage progressif. Il y donc un seuil optimal pour chaque sous-tâche. Mais une des expériences sur le problème de la double spirale montre que pour une certaine initialisation, telle séquence est meilleure que l'autre, alors que pour une autre initialisation, c'est l'inverse. On en conclut qu'il n'y a pas de suite de seuils optimale quelle que soit l'initialisation, mais que pour *chaque* initialisation, une suite optimale différente doit être trouvée. La détermination de l'instant de passage à l'ensemble suivant devrait donc se faire au cours de l'apprentissage, de manière réflexive. Les algorithmes décrits dans le paragraphe III.1.3.f sont des solutions potentielles.

III.3.2 Dans les réseaux mécaniques

III.3.2.a Gérer la difficulté dans les réseaux mécaniques

Dans le cas de l'apprentissage de comportements mécaniques, l'action peut prendre plusieurs formes. Contrairement aux expériences précédentes, on ne manipule pas d'ensemble d'apprentissage, car le réseau apprend en interagissant avec l'objet mécanique, qui fournit le comportement désiré en temps réel.

L'action est de type "direct" (voir figure III.1) : elle consiste en l'envoi d'une séquence de forces par l'opérateur à la fois au réseau adaptatif et à l'objet à identifier. Dans toutes les expériences décrites ci-dessus, cette séquence était limitée à un impulsion unitaire (de valeur *1*) à l'instant *1*.

Une autre possibilité d'action existe aussi : dans les algorithmes faisant intervenir un (semi-)couplage on peut en cours d'apprentissage modifier les paramètres de couplage (raideur et viscosité du ressort amorti de couplage), ce qui revient à modifier

les forces envoyées vers l'objet à identifier, dans le cas d'un vrai couplage, et surtout à modifier les signaux entrant dans le réseau.

Comme il a été discuté au paragraphe III.2.2.b, une première composante de l'apprentissage actif est le diagnostic de difficulté. Nous avons choisi les deux critères a posteriori d'estimation de la difficulté à savoir :

- l'action courante est jugée facile si la *DRM* (différence relative moyenne, voir la formule (54) au paragraphe II.5.1) est faible ;
- l'action courante est jugée difficile si la *DRM* évolue peu.

Nous avons pu remarquer, conformément à ce qui est discuté au paragraphe III.1.3.e que la difficulté n'était pas du tout proportionnelle à l'erreur : pour une même *DRM* élevée, un réseau va aisément converger, alors qu'un autre va échouer.

Le problème central qui se pose ensuite est celui du choix d'une action : quelle action est facile, quelle action est difficile ?

Pour aborder cette difficile question en ce qui concerne le geste de l'opérateur, on peut considérer qu'une action est difficile si elle révèle beaucoup d'information sur l'objet à identifier. Or pour les systèmes linéaires, on peut relier cette idée intuitive d'information aux modes fréquentiels de l'objet que révèle une excitation donnée : une structure mécanique possède un certain nombre de modes ; selon l'excitation, certains de ces modes vont apparaître dans le signal, d'autres non, ou plus faiblement (typiquement, si on excite une corde en son milieu, seuls les modes pairs vont apparaître).

L'excitation par un impulsion unitaire au temps *1* (impulsion de Dirac) est celle pour laquelle les modes s'expriment le plus, puisque la structure est libre. Cette excitation est donc l'action la plus difficile. C'est celle qui a été choisie dans les expériences du chapitre II.5. N'importe quelle autre excitation sera donc plus simple. On peut donc envisager les actions suivante, pour simplifier l'apprentissage :

- Double impulsion : pour un signal de longueur fixe, en plus de la force unitaire au temps *1*, on ajoute une deuxième force impulsionnelle à un instant temporel quelconque (voir figure III.23). Cette deuxième impulsion a pour effet de remettre en phase le réseau et l'objet, avant que tous les modes se soient entièrement exprimés.
- Impulsions multiples : on n'ajoute plus une mais plusieurs forces impulsionnelles. Si on en ajoute un grand nombre (de l'ordre de la longueur de la séquence), on aboutit alors à une force constante.

- Excitation sinusoïdale : si on excite une structure par un signal sinusoïdal d'une fréquence donnée *f*, alors les modes de l'objet de fréquence proche de *f* vont être plus présents dans le signal. En particulier, si *f* correspond exactement à un mode de l'objet, un phénomène de *résonance* se produira, et ce mode dominera tous les autres. Ce type de résonance est utilisé pour l'identification des différents modes en analyse modale expérimentale [Ewins 84]. On peut donc envisager d'exciter d'abord par un signal sinusoïdal de basse fréquence, puis augmenter progressivement la fréquence d'oscillation.
- Amortissement : des études ont montré que l'apprentissage de séquences dans les réseaux récurrents est facilité si on commence avec des séquences courtes, puis on rallonge la longueur de la séquence (voir paragraphe III1.2.b). En amortissant l'objet peu après l'excitation, on peut découper une longue séquence en séquences plus petites donc plus faciles. Cet amortissement s'effectue à travers le geste de l'opérateur.

Ces différentes excitations sont résumées sur la figure III.23.

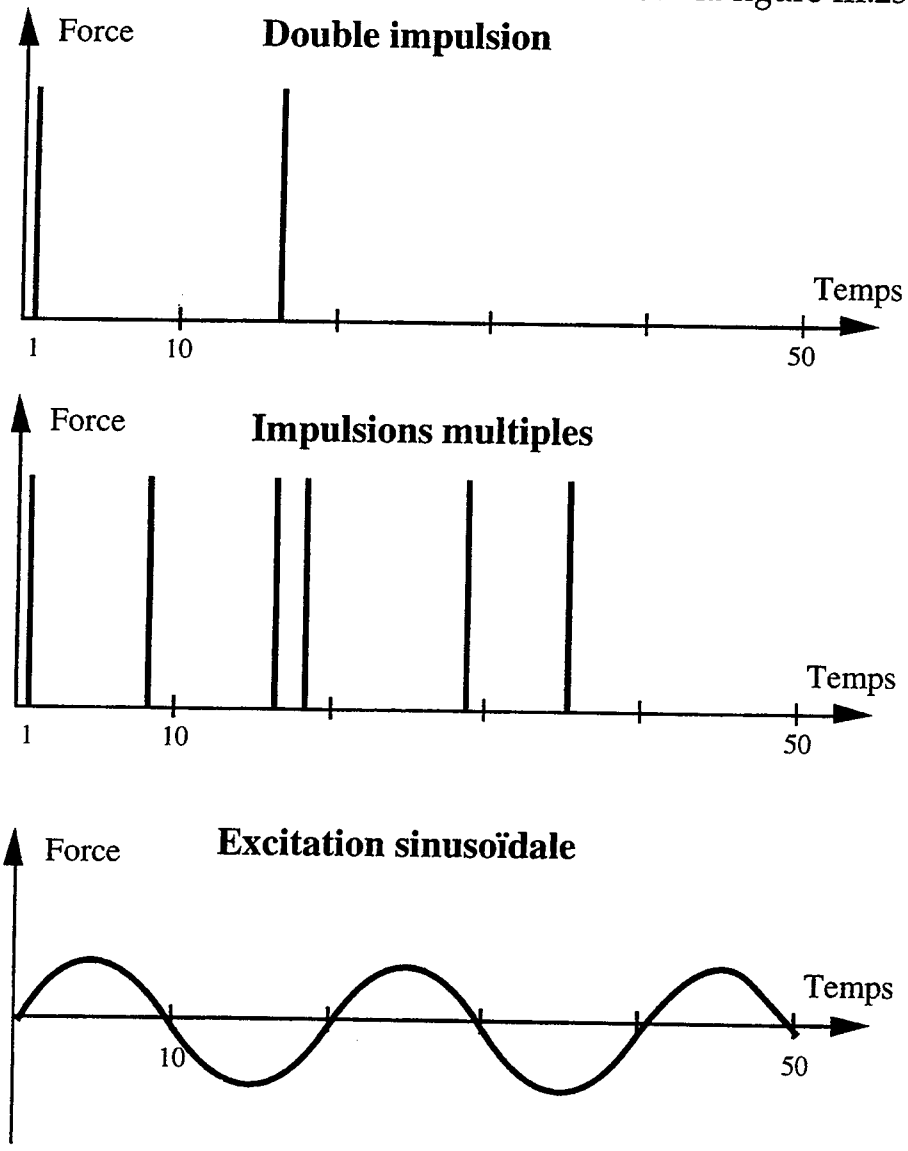

Figure III.23. Différentes excitations pour simplifier l'apprentissage

Parallèlement à l'excitation, on peut agir sur le couplage entre l'objet et le réseau, pour les algorithmes de couplage, semi-couplage et couplage différentiel (voir chapitre II.4).

D'une part, dans la mesure où le couplage a pour effet de rapprocher les deux trajectoires, on peut supposer que si le couplage est fort, il est ensuite plus facile de rapprocher encore ces deux trajectoires par adaptation. Pour un couplage très fort, la *DRM* est sous le seuil souhaité avant même adaptation. Bien sûr, comme il a été noté au paragraphe II.5.1, le fait que le réseau ait appris à diminuer l'erreur sous un certain seuil avec un couplage fort ne garantit en rien que les deux comportements sont proches sans couplage. Une possibilité d'action consiste donc à commencer avec une raideur de couplage élevée puis diminuer petit à petit ce couplage, comme il a été suggéré au chapitre III.2.2.b.

D'autre part, la viscosité de couplage a pour effet d'amortir les réseaux d'adaptation (voir le paragraphe II.4.1 et la figure II.21), c'est-à-dire diminuer la fenêtre temporelle prise en compte pour l'adaptation. Plus cette viscosité est élevée, plus courtes sont les dépendances prises en compte, donc plus l'apprentissage est facile. On peut donc envisager de commencer l'apprentissage avec une viscosité élevée, puis diminuer progressivement cette viscosité.

On remarquera que l'action via le couplage et l'action directe par le geste sur l'objet à identifier peuvent être combinées.

III.3.2.b Résultats expérimentaux

Si toutes les possibilités d'action n'ont pu être testées dans cette étude, des résultats ont néanmoins été dégagés.

Tout d'abord, les résultats concernant l'action sur le couplage ont été décevants. En reprenant les expériences du paragraphe II.5.2.d, l'adjonction d'une stratégie d'apprentissage n'a pas permis d'augmenter les taux de réussite : les meilleurs résultats donnent un taux de réussite de *13/20*, c'est-à-dire voisin de ce que l'on obtient avec un couplage fixe.

Les meilleurs résultats ont été obtenus avec des stratégies d'apprentissage utilisant des doubles impulsions ou des impulsions multiples. Pour toutes les expériences en question, le même "proto-algorithme" a été utilisé :

```
DEBUT
    Initialiser A
    Répéter
        Adapter les poids
        Si A difficile, simplifier A
        Si A facile, compliquer A
    jusqu'à A = Action_cible et apprentissage réussi
FIN
```

avec :

- *A* difficile = $100\ (DRM(n) - DRM(n-1))/DRM(n) < Seuil\_stagnation$
- *A* facile = $DRM < Seuil\_DRM$
- *Action_cible* = impulsion unitaire
- Apprentissage réussi = $DRM < Seuil\_DRM$

Première stratégie : double impulsion

Les expériences ont été faites pour des séquences courtes de *50* échantillons, sur le test "difficile" décrit au paragraphe II.5.1. L'algorithme utilisé est le semi-couplage. Rappelons qu'avec une impulsion unitaire, on obtient un taux de convergence de *14/20*, en une moyenne de *228* présentations de la séquence.

La simplification de l'action *A* consiste à choisir comme excitation une double impulsion unitaire, la première en *1* et la deuxième aléatoirement choisie entre *2* et *49*. La complexification consiste en un retour à l'impulsion unique.

On notera qu'avec cette stratégie, si une double impulsion n'a pas permis de réduire la *DRM* avant que le seuil de stagnation soit atteint, alors une nouvelle double impulsion est appliquée. Ainsi se met en place un processus automatique d'essais successifs pour trouver une action permettant de réduire la *DRM*.

Les paramètres de l'algorithme ont été les suivants :

Raideur de couplage	0,05
Viscosité de couplage	0,5
Pas d'apprentissage	0,0002
Seuil_DRM	1,5
Seuil_stagnation	0,5

Les résultats sont les suivants :

Taux de réussite	Nombre de présentations			
	moyenne	minimum	maximum	Ecart type
17/20	142	16	631	147

La *DRM* après apprentissage, calculée sans couplage, moyennée sur les essais réussis est de *4,2* qui est effectivement inférieur à *5*.

On obtient donc une augmentation du nombre de succès de *14/20* à *17/20*, grâce à la stratégie d'apprentissage, ce qui constitue une résultat positif.

Il est intéressant de remarquer que les paramètres de couplage sont les mêmes que pour l'algorithme avec action unitaire (voir paragraphe II.5.2.d). Le fait de modifier l'action peut donc être indépendant de l'algorithme d'adaptation des poids.

Le nombre de présentations est de *142* en moyenne contre *228* avec une simple impulsion.

Deuxième stratégie : impulsions multiples

Il s'agit de la même stratégie que précédemment, sauf que l'action de simplification consiste à *ajouter* une impulsion unitaire à un instant quelconque de la séquence de force délivrée au réseau et à l'objet. Par conséquent, si, plusieurs fois de suite, la *DRM* stagne (critère de difficulté), alors l'excitation contient de plus en plus d'impulsions, jusqu'à l'obtention d'une force constante.

Les paramètres de l'algorithme ont été les suivants :

Raideur de couplage	0,05
Viscosité de couplage	0,5
Pas d'apprentissage	0,0004
Seuil_DRM	1,5
Seuil_stagnation	1

Les résultats sont les suivants :

Taux de réussite	Nombre de présentations			
	moyenne	minimum	maximum	Ecart type
19/20	71	9	260	61

La *DRM* après apprentissage, calculée sans couplage, moyennée sur les essais réussis est de *4,0* qui est effectivement inférieur à *5*.

On obtient d'excellents résultats, puisque *19* réseaux sur *20* ont convergé. De plus, seulement *71* présentations ont été nécessaires, en moyenne. Une simulation supplémentaire sur 100 essais a confirmé ce bon résultat, puisque l'apprentissage a réussi dans *91* cas, avec une moyenne de *104* présentations.

Enfin, une série d'expériences complémentaires a mis en évidence le peu de sensibilité à la valeur du seuil de stagnation : en faisant varier ce paramètre d'un facteur *10* dans les deux sens, le taux de réussite reste supérieur ou égal à *17/20* (voir figure III.24). Le nombre de présentations augmente plus nettement quand le seuil est élevé (les impulsions sont ajoutées plus souvent). Ce paramètre est donc facile à régler, ce qui n'est pas le cas des paramètres de couplage.

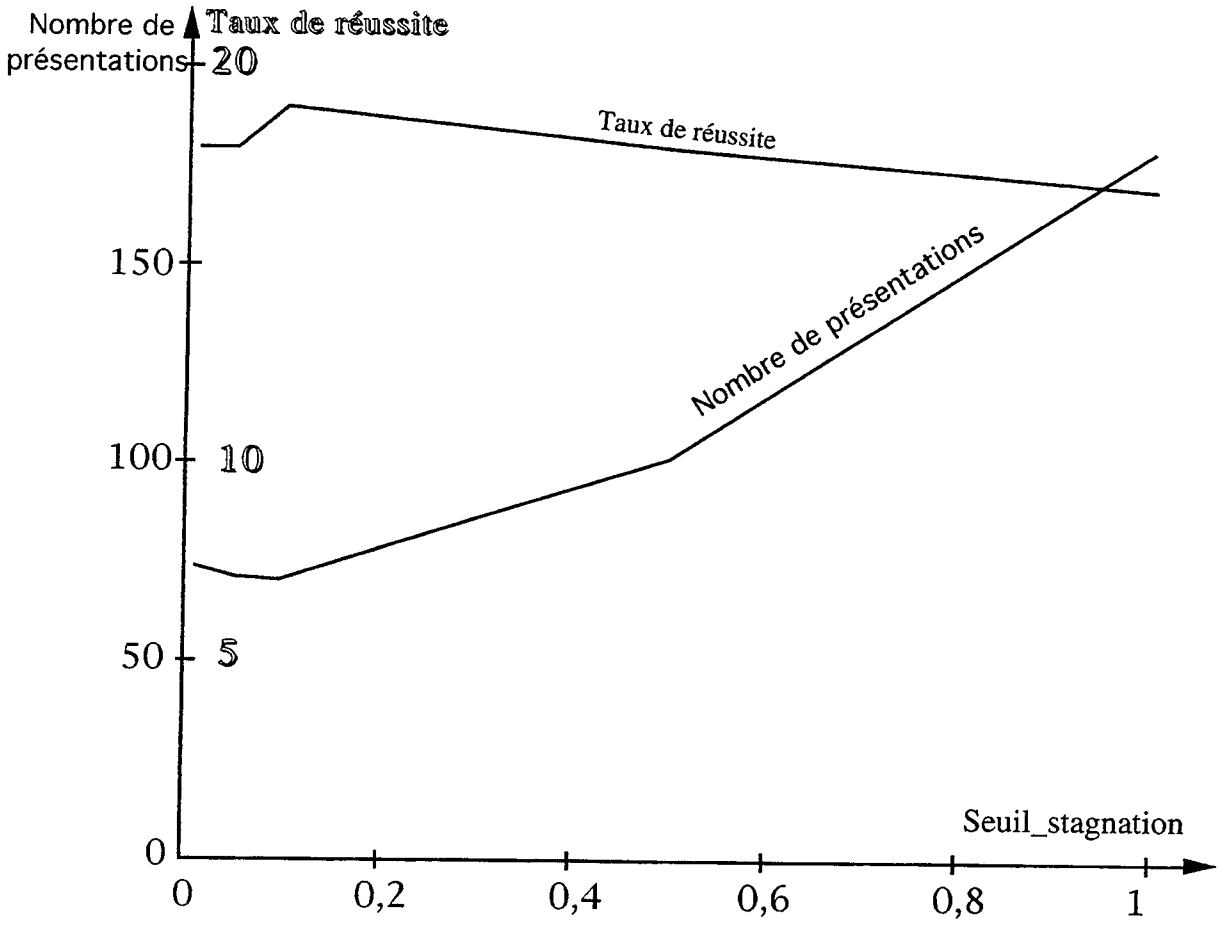

Figure III.24. Sensibilité au paramètre de stagnation

*

Ces expériences montrent donc que le choix d'une action appropriée peut significativement aider l'apprentissage en accélérant la convergence et surtout en augmentant le taux de réussite.

Cependant, la conception d'algorithmes actifs efficaces s'avère délicate, comme dans le cas des réseaux de neurones classiques (chapitre III.3.1). Une étude mécanique plus poussée est certainement nécessaire pour tenter de quantifier la notion de difficulté d'une action mécanique.

En effet, le choix de l'action demeure ici très simpliste, fondé sur l'aléatoire. Si déjà cette action donne de bons résultats, il faut rechercher d'autre types d'excitations dont on contrôle mieux l'effet.

III.4 De l'apprentissage automatique à l'apprentissage coopératif

III.4.1 Rôles respectifs de l'homme et l'ordinateur dans l'apprentissage artificiel

III.4.1.a L'apprentissage artificiel est rarement automatique

De manière générale, l'apprentissage automatique a pour objectif de remplacer l'homme dans des tâches de reconnaissance, de classification, de modélisation, de contrôle, etc. Les algorithmes neuronaux constituent un des outils utilisés à cette fin.

Cependant, il s'avère qu'en ce qui concerne les réseaux de neurones, on ne peut pas vraiment affirmer que l'ordinateur remplace totalement l'homme :

- Premièrement, le choix d'un modèle adapté n'est pas automatique : il nécessite à la fois un expert en réseaux de neurones et un expert dans le domaine d'application.
- Deuxièmement, le codage du problème sous une forme favorisant l'apprentissage n'est pas aisé, et nécessite une grande connaissance des réseaux de neurones.
- Troisièmement, la mise au point des paramètres (pas d'apprentissage, amplitudes d'initialisation des poids, etc.) se fait généralement de manière empirique, par essais et erreurs successifs.

Ainsi, en fait d'apprentissage automatique, on a affaire à un apprentissage informatique qui fait aussi intervenir l'homme : on pourrait parler d'apprentissage "semi-automatique". Mais par le fait que les réseaux de neurones sont perçus en tant que technique d'apprentissage automatique, le rôle de l'homme dans le processus d'apprentissage est considéré comme marginal et secondaire. Or son rôle est primordial et non trivial : il est impossible de trouver les paramètres du premier coup, et la démarche du concepteur de réseaux neuronaux est réellement celle d'un expert, par exemple pour régler les paramètres.

Deux attitudes sont possibles face à ce constat : soit on tente d'améliorer les algorithmes pour les rendre plus autonomes, soit au contraire on considère qu'ils ne sont pas autonomes et il faut alors redéfinir les rôles respectifs de l'homme et de l'ordinateur.

De nombreux travaux relèvent de la première attitude. Par exemple, la variante de la rétropropagation appelée *RPROP* [Riedmiller 94] élimine les pas d'apprentissage de l'algorithmes en ne prenant en compte que le signe de chaque composante du gradient. Ainsi, un paramètre est supprimé. En ce qui concerne le codage, l'algorithme de *contre-propagation* proposé dans [Hecht-Nielsen 88] a précisément pour but de prétraiter automatiquement les données afin d'optimiser l'apprentissage. Enfin, il faut citer l'approche très originale développée dans [Nassersharif & Rajan 93] : le réglage des paramètres est confié à un système expert qui peut appliquer en même temps de nombreuses heuristiques (une de ces heuristiques concerne même l'ensemble d'apprentissage). Les résultats avec le système expert est meilleur que les meilleurs résultats obtenus par un réglage manuel ; ceci s'explique par le fait que le système expert modifie les paramètres en cours d'apprentissage.

Dans ce qui suit, c'est la deuxième attitude qui va être développée, non qu'elle soit meilleure que la première, mais parce que d'une part l'opérateur intervient déjà dans le problème d'identification de paramètres d'objets physiques pour produire le geste (voir paragraphe I.2.3) et d'autre part on a vu au chapitre III.1.3 toutes les difficultés qu'il y a à automatiser l'apprentissage progressif.

Nous introduisons donc la notion d'apprentissage *coopératif*, qui consiste à considérer que l'ordinateur et l'homme coopèrent pour faire réussir l'apprentissage. Par rapport à l'approche classique, semi-automatique, ou coopérative déguisée, il s'agira tout d'abord de définir avec précision le rôle de l'homme dans l'apprentissage.

L'idée sous-jacente de l'apprentissage coopératif serait que tout ne pourrait être automatisé dans l'apprentissage, et qu'en conséquence, certains aspects de

l'apprentissage devraient être confiés à l'homme. Cette idée semble particulièrement pertinente dans le cas de l'apprentissage progressif.

III.4.1.b Principes pour un apprentissage coopératif

Il s'agit maintenant de reposer le problème de l'apprentissage automatique en termes d'*apprentissage coopératif.*

Dans cette coopération, deux agents interviennent, l'homme et l'ordinateur, qui sont engagés dans la *même* tâche d'apprentissage. Les caractéristiques très différentes de ces deux agents vont permettre de définir leurs rôles respectifs dans l'apprentissage. L'homme est capable de réaliser des traitements de hauts niveaux, mais ne peut délivrer un grand nombre de résultats en même temps ; de plus, tout au moins lorsqu'il est encore débutant, l'homme ne peut prendre en compte qu'un petit nombre de variables. A l'inverse, l'ordinateur peut effectuer des calculs longs et complexes, sur de nombreuses variables en entrée et en sortie, mais ses capacités "intelligentes" sont limitées, en ce qui concerne les raisonnements globaux et stratégiques, l'adaptation des stratégies par apprentissage, etc.

Quand le problème qui est à résoudre consiste à trouver les paramètres d'un modèle pour reproduire un comportement complexe, l'ordinateur remplace avantageusement l'homme quand il s'agit de modifier une centaine de paramètres à la fois. Par contre, les heuristiques qui peuvent exister pour *contrôler* le déroulement de l'apprentissage peuvent être confiées à l'homme. Percevant l'environnement et l'ordinateur, l'homme peut à la fois agir sur l'environnement et modifier des paramètres sur l'ordinateur pour modifier le cours de l'apprentissage. Une caractéristique importante de cette coopération est son aspect "temps réel" : l'homme peut modifier l'environnement ou l'algorithme au fur et à mesure du déroulement de l'apprentissage.

Dans le cadre de l'apprentissage actif, il semble raisonnable de laisser agir l'homme pendant que l'ordinateur procède au mécanisme d'identification de paramètres. Ceci va nous amener à définir plus précisément un système fondé sur ce principe appliqué à l'apprentissage dans les modèles physiques.

III.4.2 Apprentissage coopératif dans les modèles physiques

III.4.2.a spécifications générales

Nous présentons ici un système général qui s'applique à tout apprentissage dans les modèles physiques, que ce soit pour la modélisation d'objets vibrants, l'identification de modèles physiques pour l'animation, l'identification de caractéristiques mécaniques de terrains en robotique, etc.

Dans le cadre de l'apprentissage des modèles physiques, rappelons que le réglage entièrement manuel des paramètres est vite impossible, quand il y plus de trois ou quatre paramètres. L'ordinateur, à l'aide des algorithmes décrits dans les chapitres II.3 et II.4, peut modifier un grand nombre de paramètres, selon la dynamique du gradient. Mais cette dynamique est limitée par le problème des "pièges" sur la surface d'erreur (voir paragraphe III.1.3.d). L'action peut permettre d'éviter ces pièges ou de s'en échapper, et peut être confiée à l'homme qui a la possibilité d'agir mécaniquement sur les objets de manière à faciliter l'apprentissage.

La question qui se pose alors est la suivante : l'homme saura-t-il choisir l'action appropriée ? Nous supposons que oui, au vu des résultats déjà obtenus par l'algorithme d'impulsions multiples au paragraphe III.3.2.b. Grâce à ces expériences, l'opérateur humain ne part pas de rien, puisqu'il a déjà une idée de quelle action est plus difficile que l'autre. De plus, contrairement aux algorithmes automatiques, celui-ci a la possibilité de modifier sa stratégie, autrement dit d'apprendre à agir. Dans ce qui suit, on définit les conditions dans lesquelles l'interaction entre l'opérateur, l'ordinateur et l'objet à identifier autorise un apprentissage progressif coopératif dans les modèles physiques.

Le schéma de la figure III.25 décrit l'apprentissage coopératif dans les modèles physiques. Afin que l'objet réel et le modèle reçoivent la même force de l'opérateur, condition essentielle pour réaliser l'identification de paramètres, l'opérateur agit sur l'objet à identifier via l'ordinateur, et plus particulièrement via un élément mécanique simulé appelé "coupleur", qui a pour fonction de distribuer équitablement les forces sur le modèle et l'objet. L'ordinateur communique mécaniquement avec l'opérateur et l'objet à identifier via deux *Interfaces Rétroactives* (*IR* sur le schéma), qui sont chacune composées d'un ou plusieurs capteur(s)-effecteur(s).

Etant donné que le choix de l'action met surtout en jeu un raisonnement de haut niveau, il est à supposer que le seul retour d'effort tactile ne suffira pas à l'opérateur pour choisir quelle action appliquer. C'est pourquoi un système d'aide au diagnostic doit être intégré, permettant à l'opérateur d'obtenir les données nécessaires à son choix d'action. La manière dont ce module présente les informations à l'opérateur est de toute première importance ; afin d'éviter les interférences, le mode de communication est nécessairement visuel dans le cas où l'objet produit un son, et éventuellement sonore quand la déformation de l'objet doit être vue par l'opérateur.

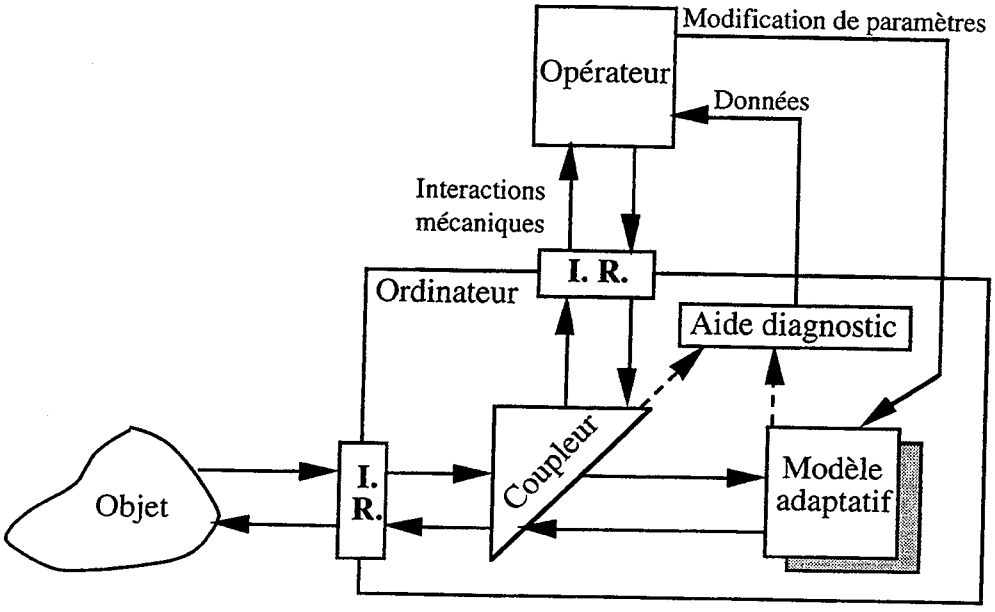

Figure III.25. Apprentissage coopératif dans les modèles physiques

Précisons qu'en plus de l'action sur l'objet, l'opérateur a la possibilité d'intervenir sur différents paramètres d'apprentissage, en particulier les pas d'adaptation. Par souci de clarté, nous n'avons pas fait figurer sur le schéma la possibilité d'action sur le coupleur ou le calcul de l'erreur.

Dans ce qui suit sont présentés une expérience où l'opérateur agit en temps réel sur l'objet et le réseau, puis le futur système d'analyse des objets mécaniques vibrants que nous envisageons.

III.4.2.b Expérimentations

Description de l'expérience

L'expérience qui va être décrite quoique novatrice n'en est pas moins tout à fait rudimentaire. Des contraintes avant tout technologiques limitent en effet les possibilités d'expérimentations actuelles.

Les interfaces rétroactives destinées à l'opérateur et à l'objet à identifier (voir figure III.25) ont été modélisées par des Transducteurs Gestuels Rétroactifs (*TGR*) développés à l'*ACROE* depuis *1988* [Cadoz *et al.* 90]. Un *TGR* est une interface capable d'une part de saisir un geste humain au moyen d'un capteur de position et d'autre part de renvoyer une force vers l'opérateur. La fréquence d'échantillonnage est de l'ordre de *1000 Hz*, ce qui est tout à fait performant pour la manipulation gestuelle. Mais il s'agit d'une fréquence bien insuffisante en ce qui concerne la synthèse sonore, qui nécessite des fréquences d'échantillonnage de l'ordre de *20000 Hz*. Ainsi, le dispositif que nous avons réalisé est limité à l'identification d'objet déformables ayant des comportements lents.

Une deuxième contrainte provient des défauts mécaniques du *TGR* : un important frottement sec amortit rapidement les oscillations éventuelles de l'interface. En conséquence, il est impossible d'utiliser un *TGR* pour décrire des mouvements oscillatoires comportant de nombreux modes, auxquels les réseaux CORDIS sont dédiés. En conséquence, nous avons choisi de limiter le modèle à une cellule de base (une masse attachée au sol par un ressort amorti).

L'objet à identifier devant avoir un comportement linéaire, nous avons tout simplement attaché un *TGR* à un support fixe avec deux élastiques tendus. Le dispositif expérimental est illustré sur la figure III.26.

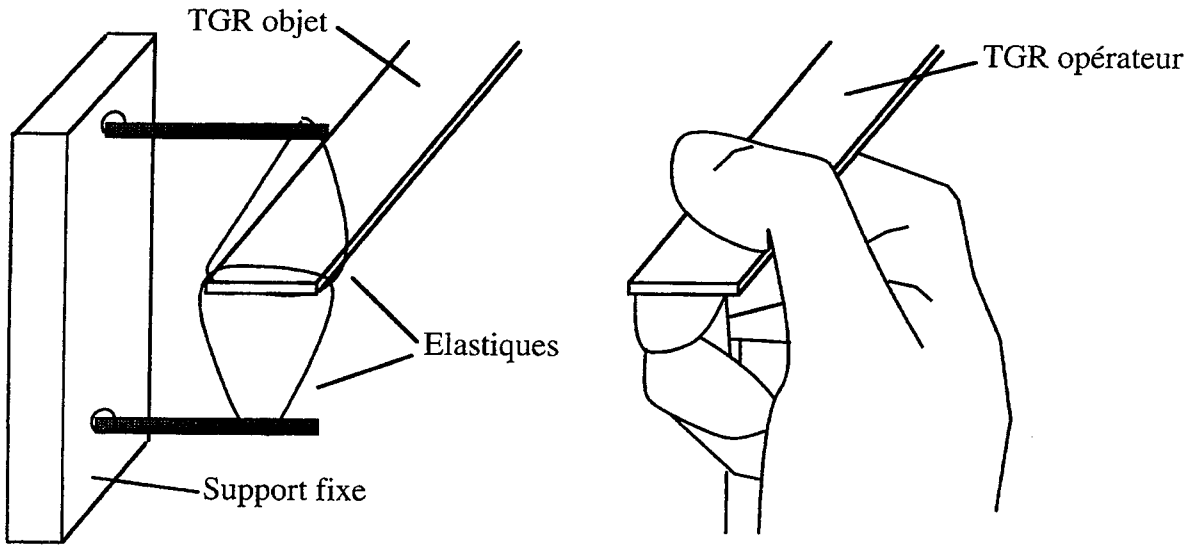

Figure III.26. Dispositif expérimental

Le système informatique est représenté sur la figure III.27. Le coupleur (voir figure III.25) est simplifié au maximum, ne comprenant que deux ressorts (éventuellement amortis) : l'un entre le *TGR* de l'opérateur et le *TGR* de l'objet, l'autre entre le *TGR* de l'opérateur et le modèle. Ces deux ressorts sont identiques pour répartir les forces de l'opérateur sur le réseau et l'objet. On notera cependant que cette

répartition n'est pas parfaite, quand le réseau et le modèle se comportent différemment, puisque la force émise par chaque ressort dépend de la position des deux extrémités. Cette imperfection est une conséquence inévitable de la bidirectionnalité des communications mécaniques. Par rapport aux expériences utilisant un couplage décrites au chapitre II.5, on ne retrouve plus de ressort directement entre l'objet et le réseau (comparer avec la figure II.20 du paragraphe II.4.1) : on aurait pu l'introduire, mais les deux ressorts déjà présents suffisent à assurer un couplage entre l'objet et le réseau.

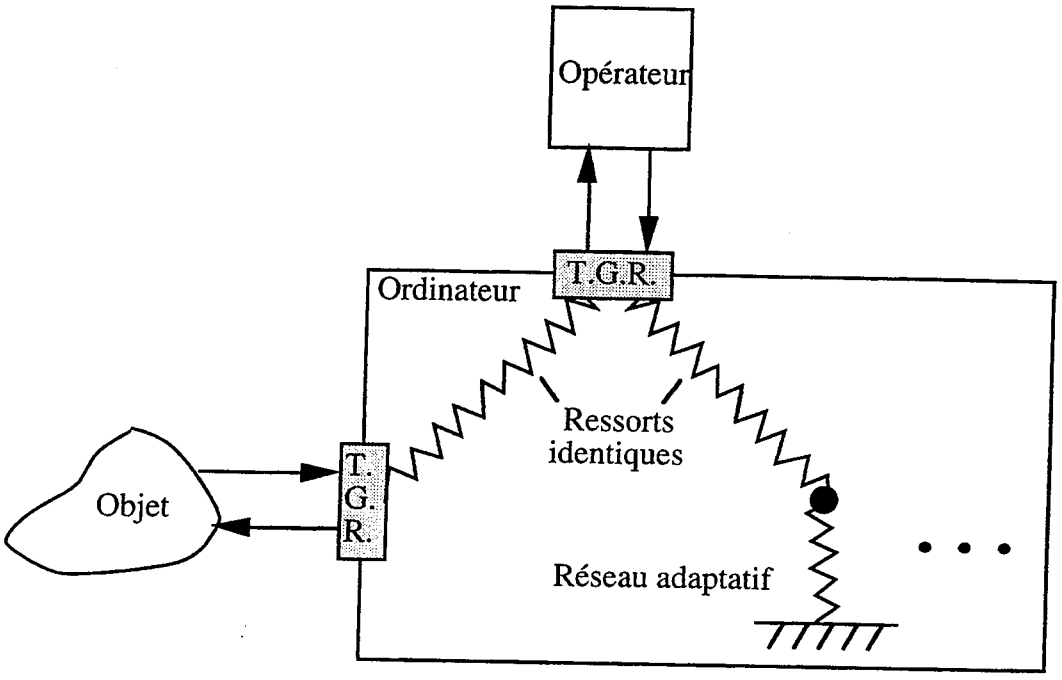

Figure III.27. Système d'apprentissage coopératif expérimenté

L'algorithme d'apprentissage est l'algorithme de *RTRL* avec couplage, exposé au paragraphe II.4.1. Les paramètres *Kc* et *Zc* sont les raideur et viscosité du ressort entre le réseau et le *TGR* de l'opérateur. Sur le schéma de la figure III.27, on se rend compte clairement que ce qu'apprend le réseau n'est pas l'objet "élastique", mais l'ensemble "élastique + *TGR*" ; en effet, on ne saurait négliger les caractéristiques mécaniques de l'interface [Nouiri 91].

Au niveau de l'implantation informatique, les programmes ont été écrits en langage *C* compilé sur une machine utilisant massivement des techniques de vectorisation (Modèle "Power Challenge" de *SiliconGraphics*). Il ne s'agit pas véritablement d'une machine parallèle adaptée au calcul des réseaux d'automates, mais le fait que ces calculs comprennent essentiellement des boucles imbriquées peut être exploité par le compilateur. Des tests de vitesse ont établi que l'on pouvait simuler l'apprentissage en temps réel à *1000 Hz* d'un réseau de quatre masses entièrement connectées et connectées au sol (voir la figure II.24 du chapitre II.5.1). Avec le modèle

expérimenté, la cellule de base, on n'exploite donc pas les capacités de la machine au maximum.

Méthodologie de test

La caractéristique majeure de cette expérience par rapport à toutes celles décrites ci-dessus est qu'elle fait intervenir l'humain dans le courant de l'apprentissage, qui fournit le geste à l'objet et au réseau, via le *TGR*. Cette action de l'opérateur humain n'est pas reproductible, car on ne peut faire deux fois exactement le même geste : il est donc nécessaire de réaliser de nombreux essais, en faisant varier non seulement l'initialisation des réseaux, comme on fait généralement dans les réseaux connexionnistes, mais aussi le geste de l'opérateur et l'opérateur lui-même.

Par ailleurs, le fait que le geste ne soit pas reproductible pose des problèmes d'*évaluation* de la solution obtenue : on ne peut se contenter de mesurer l'erreur entre les comportements du modèle et du réseau, puisque celle-ci dépend du geste. Nous estimons que la meilleure évaluation du système serait perceptive, fondée sur la méthodologie utilisée en psychologie expérimentale : on fait comparer à de nombreux sujets humains l'objet et le réseau, avant et après apprentissage.

Une telle évaluation perceptive n'a pas été réalisée car l'expérience est encore à un stade très préliminaire qui ne justifie pas le développement d'une telle méthodologie.

Deux évaluations ont donc été utilisées :

- sur un plan quantitatif, on a mesuré un critère de différence entre le réseau et l'objet à identifier :

$$d_T(n) = \frac{1}{T} \sum_{\eta=n-T+1}^{n} \left| X_a(\eta) - X_{\tilde{a}}(\eta) \right|$$

où T mesure la taille de la fenêtre temporelle sur laquelle on mesure l'erreur. On détermine ce critère pour une manipulation humaine de type oscillatoire (de l'ordre de deux Hz), et aussi sans aucune manipulation. En effet, du fait du bruit et surtout du frottement sec qui stabilise le *TGR* dans une position non nulle, $d_T(n)$ n'est pas nulle quand l'opérateur n'agit pas. Il faut donc soustraire cette erreur de "bruit de fond" à l'erreur mesurée. On définit finalement l'erreur comme suit :

$$\Delta d_T = d_T(N) - d_T^{\text{sans excitation.}}(N)$$

avec N la longueur de la séquence. Cette mesure est très approximative puisqu'elle dépend du geste.

- sur un plan qualitatif, on a fait simplement comparer à quelques sujets humains l'objet et le réseau (une simple touche permet de passer en cours de simulation de la manipulation de l'objet à la manipulation du réseau) avant et après apprentissage.

Résultats

Un seul objet à identifier a été testé ; deux initialisation différentes ont été testées et pour chacune de ces deux initialisations trois essais positifs sont décrits dans le tableau III.9.

m_{init}	k_{init}	z_{init}	α	β	μ	Δd_{10000} initial	n° de l'essai	Δd_{10000} final	nombre d'adaptations
2	10^{-6}	$3\ 10^{-5}$	10^{-3}	10^{-8}	10^{-5}	0.14	1	0.03	36495
							2	0.02	26244
							3	0.02	27756
1	10^{-6}	0	$5\ 10^{-4}$	10^{-8}	10^{-5}	0.2	1	0.02	40102
							2	0.02	40032
							3	0.04	27302

Tableau III.9. Résultats pour l'expérience d'apprentissage temps réel

On constate donc que l'erreur Δd_T a sensiblement baissé après l'apprentissage, parfois d'un facteur *10*. Contrairement aux expériences du chapitre II.5, les nombres d'adaptations reportés dans le tableau III.9 ne correspondent pas à un critère d'arrêt précis. Ils donnent cependant une idée de la longueur des séquences.

Sur le plan qualitatif, la manipulation de l'objet et du réseau avant apprentissage fait apparaître une forte différence subjective : l'objet à identifier (les élastiques) est assez difficile à déplacer et léger, revenant très rapidement à sa position d'équilibre si on l'en écarte, alors que l'objet virtuel est beaucoup plus facile à déplacer, et l'on sent l'inertie de son déplacement dans ses doigts, ainsi qu'une très légère oscillation. Après apprentissage, la différence entre les deux objets, réel et virtuel, n'est pas perceptible.

Les deux évaluations montrent donc que le réseau a adapté ses paramètres pour reproduire les caractéristiques mécaniques de l'objet à identifier.

Pendant l'apprentissage, l'opérateur est en interaction avec les deux objets ; il lui est cependant difficile de sentir dans ses doigts ce qui se passe, et de pouvoir dire que l'un des objets s'est rapproché de l'autre. L'action qu'il émet vers le *TGR* est donc arbitraire. Cette expérience montre donc la nécessiter de donner plus d'informations à l'opérateur que le simple retour d'effort, si l'on veut lui offrir la possibilité d'appliquer une stratégie d'action progressive.

III.4.2.c Vers un système coopératif pour l'apprentissage de structures mécaniques en temps réel

L'expérience qui vient d'être décrite ne saurait être un aboutissement de cette étude : elle est au contraire l'ouverture vers la réalisation d'un système opérationnel d'analyse en temps réel d'objets mécaniques. Nous allons en effet décrire ici les grandes lignes d'un système d'analyse de structures vibrantes pour la modélisation d'instruments de musique, système dont la réalisation est encore hors d'atteinte pour des problèmes technologiques qui devraient être résolus sous peu.

L'objectif du système futur envisagé est d'implanter les algorithmes d'apprentissage décrits ci-dessus dans le cadre de l'identification d'objets vibrant échantillonnés à *20 kHz*, fréquence minimale requise pour un échantillonnage correct du son. Comme cela a été expliqué au paragraphe II.1.1.b, cela implique un non-linéarité au niveau de l'excitateur situé entre l'opérateur et la structure vibrante (ou résonateur). Or dans le contexte de cette étude, nous n'étudions pas l'adaptation de l'excitation non linéaire. Deux solutions sont alors envisageables (voir figure III.28).

Dans la première solution, on situe l'excitateur hors du système informatique : l'opérateur agit sur l'objet virtuel via un *Transducteur Haute Fréquence (THF)*, à l'aide d'un excitateur, à savoir une mailloche, un archet de violon, son ongle, etc. Entre l'ordinateur et l'objet est placé un deuxième *THF*. Pour ce dernier, on peut utiliser un microphone dit de contact, classiquement utilisé pour la prise de son d'instruments de musique acoustiques, associé à un "haut-parleur de contact", proche des divers types d'excitateurs électromagnétiques utilisés dans l'industrie pour analyser des structures mécaniques quelconques [Ewins 84]. On pourrait aussi envisager un unique transducteur qui à la fois excite la structure vibrante et capte la position résultante. Dans [Caussé & Weinreich 89 *in* Boutillon & Guérard 95], un mécanisme de couplage haute fréquence a été utilisé, puisqu'il s'agissait d'exciter une corde vibrante a l'aide d'un excitateur électromagnétique commandé par ordinateur ("the digital bow"). Au niveau du *THF* situé entre l'opérateur et l'ordinateur, la conception est plus délicate, puisque, contrairement à l'autre *THF*, on ne peut l'attacher à l'excitateur : la non-linéarité provient justement du fait que l'excitateur et la structure vibrante ne sont pas attachés. Un tel transducteur a été utilisé dans le cas du frottement d'archet dans [Boutillon & Guérard 95] : l'opérateur joue avec un archet sur une portion de corde de *6* mm attachée à un capteur-effecteur qui transmet les forces à l'ordinateur et se déplace selon les positions calculées.

Dans la deuxième solution, on inclue l'excitateur dans l'ordinateur en le simulant : il se situe au niveau du coupleur représenté sur la figure III.25. Ainsi, l'interface rétroactive entre l'opérateur et l'ordinateur est un *TGR* (voir figure III.28). Nous préférons cette solution pour les raisons suivantes :

- elle évite le problème du *THF* destiné à l'excitation de l'opérateur ;
- elle permet de changer de type d'excitation bien plus aisément : les *THF* actuels sont à notre connaissance différents si on les frotte avec un archet, si on les percute, etc. ;
- de manière plus pragmatique, elle nous permet de réutiliser une partie du dispositif déjà existant pour la capture du geste.

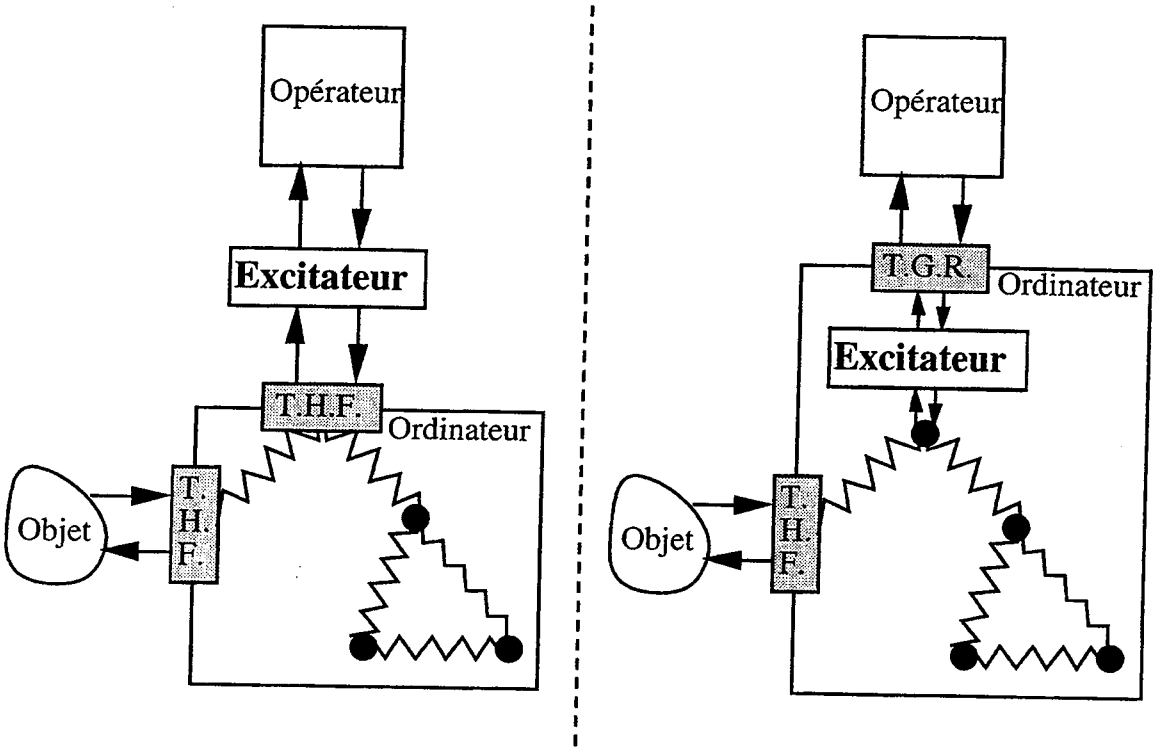

Figure III.28. Deux dispositifs possibles pour l'analyse d'objets vibrants

En plus de ces problèmes d'interface, le passage de *1 kHz* à *20 kHz* pose évidemment des problèmes de rapidité de calcul. Plusieurs raisons font penser que cet obstacle devrait être surmonté sans trop de difficulté.

Premièrement, étant donnée la complexité de l'algorithme *RTRL*, les 4 masses attachées au sol que nous pouvons actuellement simuler en temps réel (en mode apprentissage) représentent déjà un calcul assez complexe. Si on prend comme unité de complexité le nombre de liaisons simulées, on trouve en effet *250* liaisons simulées (*10* liaisons pour le réseau initial, et *24×10* liaisons pour les réseaux d'adaptation supplémentaires car il y a *24* paramètres adaptables). Avec cette structure, pour *4* modes, *250* liaisons sont nécessaires. L'idée est de simplifier la structure pour simuler avec le même nombre de liaisons d'avantage de modes. La structure la plus simple est

la *ligne*, ayant toutes ses masses égales (voir figure III.29). Avec cette structure, le réseau initial comporte n liaisons, il y a $2n+1$ paramètres donc $n(2n+1)$ liaisons dans les réseaux d'adaptation supplémentaires, ce qui fait $2n^2+2n$ liaisons. Donc, avec la capacité de calcul actuelle, on peut en fait déjà simuler une ligne de *10* masses ($n=10 \Rightarrow 2n^2+2n = 220 < 250$). Pour simuler une corde harmonique, *20* masses sont environ nécessaires, soit *840* liaisons. Il faut donc augmenter la performance approximativement d'un facteur *4*.

Deuxièmement, des optimisations informatiques devraient significativement accélérer les calculs. Les programmes, qui ont déjà été optimisés pour tourner efficacement sur le processeur "R8000", peuvent encore être améliorés si on les rend très spécifiques : à la place d'un programme simulant un réseau entièrement connecté de taille variable, avec des paramètres spécifiés en cours d'utilisation, on peut concevoir un programme qui simule uniquement une ligne de *20* masses.

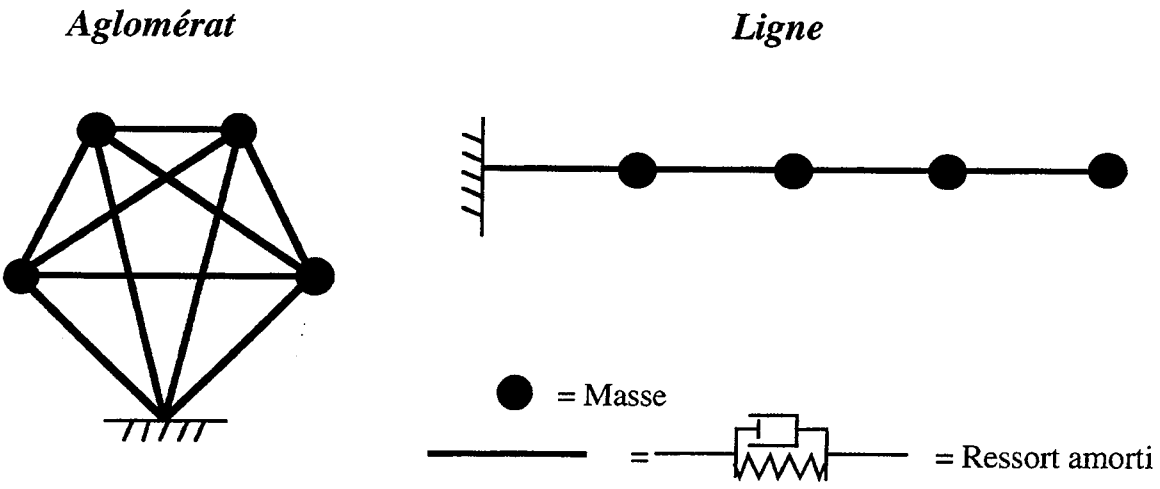

Figure III.29. Les structures extrêmes en terme de complexité de calcul

Dans ce futur système, contrairement aux expériences précédentes, l'opérateur reçoit un retour sonore de l'objet et du modèle, ce qui lui permet d'évaluer la qualité du modèle (ce retour n'est pas représenté sur la figure III.28). Le module de visualisation de l'état de l'apprentissage peut permettre de quantifier cette différence, mais aussi de visualiser la pente de l'erreur, son évolution dans le temps, etc. Il s'agit donc d'un véritable système d'apprentissage multimodal et temps réel.

CONCLUSION

Ce mémoire a été consacré aux différents aspects de l'adaptation des paramètres dans les réseaux de modélisation physique et plus généralement dans les réseaux connexionnistes supervisés. L'étude interdisciplinaire qui a été menée (Modèles Physiques, Réseaux de Neurones Formels, Informatique Musicale et Psychologie Cognitive) a conduit aux résultats suivants :

- Après avoir mis en correspondance les réseaux de neurones formels et les réseaux de masses-ressorts-frottements, on a traduit les algorithmes "neuronaux" d'apprentissage *BPTT* et *RTRL* et leurs variantes pour les réseaux de modélisation physique. Ensuite, de nouveaux algorithmes, tous fondés sur la notion de couplage mécanique entre l'objet à identifier et le réseau, ont été proposés et analysés.
- Parallèlement, une étude plus générale sur l'identification de paramètres dans un système de modélisation et simulation d'objets physiques a mis en évidence l'importance de *l'action* dans l'apprentissage. Ainsi, on a pu proposer le concept d'apprentissage actif progressif dans les réseaux et étudier les apports possibles de la psychologie cognitive sur ce sujet. Les premières expériences informatiques encouragent à mener une réflexion plus théorique, pour laquelle quelques jalons ont été posés.
- Enfin, cette réflexion a abouti à la description d'un système coopératif d'apprentissage dans les modèles physiques, fondé sur un couplage entre l'opérateur humain, l'objet à identifier et le réseau. Une expérience a été réalisée pour illustrer un tel système.

Nous sommes maintenant en mesure de répondre aux cinq questions posées en fin de première partie :

- *Est-il possible de réaliser un mécanisme d'analyse d'objets mécaniques symétrique du mécanisme de synthèse ?* Dans le cas de l'application des algorithmes de descente de gradient aux réseaux CORDIS, la réponse s'est avérée négative : l'analyse est beaucoup plus coûteuse que la synthèse. Cela s'explique en partie par la présence d'automates "cachés", pour lesquels aucun comportement n'est imposé, et dont il faut quand même trouver les paramètres. Il s'agit du classique "credit assignment problem".
- *A quel niveau se situe le seuil à partir duquel des représentations non physiques doivent intervenir dans l'apprentissage ?* La réponse dépend de l'algorithme utilisé. Dans le cas de l'algorithme *BPTT*, si la rétropropagation dans le réseau adjoint

s'effectue certes à l'aide d'un réseau mécanique, le *stockage* de la séquence temporelle n'a aucune pertinence physique. Pour *RTRL*, il a été montré que les calculs de sensibilité peuvent être réalisés par des réseaux mécaniques, mais l'adaptation elle-même s'interprète plus difficilement en termes physiques, en partie à cause de la non-localité de l'algorithme. L'algorithme de couplage-découplage qui a été proposé est un pas vers une solution en termes mécaniques, dans la mesure où la non-localité de *RTRL* est supprimée. Dans tous les cas, même si tous les traitements ne sont pas d'ordre physiques, ils restent tous dans un cadre connexionniste, étant locaux et distribués.

- *Quelles propriétés et structures faut-il ajouter aux réseaux physiques pour en faire des réseaux physiques adaptatifs ?* Si l'algorithme *BPTT* est choisi, alors il faut introduire dans chaque automate une *mémoire* stockant les états sur une période donnée du signal, ainsi qu'un réseau d'adaptation. Dans le cas de *RTRL*, il faut ajouter de nombreuses unités et connexions pour réaliser l'adaptation. Si on considère le regroupement en colonnes (voir figure II.12), alors l'adaptation se fait en compliquant les unités et les signaux transitant d'une unité à l'autre, sans oublier les connexions des points d'accès vers tous les automates. Ces dernières sont supprimés dans l'algorithme de couplage-découplage.
- *Les algorithmes obtenus sont-ils aisément parallélisables ?* La réponse est positive, car tous les calculs sont effectués par des réseaux. Dans *RTRL* par exemple, les réseaux de sensibilité peuvent s'exécuter en parallèle.
- *Quels sont les rôles que peut avoir l'action lors de l'apprentissage, et quels types de traitements sont alors mis en jeu ?* La modélisation de l'action par un simple ressort s'est avérée sans effet significatif sur les performances de l'apprentissage. Nous avons montré que l'action pouvait cependant avoir pour fonction non seulement d'accélérer la convergence mais aussi d'augmenter les chances de réussite de l'apprentissage. L'action fait alors intervenir des traitements de plus haut niveau que ceux de la représentation physique.

En répondant aux cinq questions ci-dessus, ce mémoire en a finalement posées beaucoup d'autres, concernant par exemple l'apprentissage progressif, la modélisation connexionniste des théories de l'automatisation, l'apprentissage coopératif, etc. Après avoir enclenché ainsi un certain nombre de thèmes de recherche, nous choisissons de développer ici deux perspectives qui émergent de nos travaux.

<u>L'apprentissage coopératif dans les modèles physiques</u>

Le plus gros effort de recherche sur les Réseaux de Neurones Formels porte actuellement sur l'amélioration des algorithmes d'apprentissage. Dans cette optique, on applique judicieusement les nombreuses connaissances de l'automatique, de

l'optimisation, du traitement du signal, etc. aux algorithmes neuronaux. Le système d'apprentissage coopératif pour les modèles physiques qui a été proposé au chapitre III.4 se dirige dans une toute autre direction.

Tout d'abord, les contraintes de simulation temps réel qui sont imposées, avec des vitesses devant atteindre dans le futur *20000* échantillons par seconde, imposent des algorithmes avant tout simples : on cherche à minimiser le temps de calcul pour chaque adaptation, même si beaucoup plus d'adaptation sont finalement nécessaires pour faire converger le réseau. En ce sens, la technique du gradient à pas fixe a été adoptée, bien qu'elle soit une technique dont la lenteur de *convergence* est reconnue, car c'est la technique la plus simple. Elle peut être encore simplifiée en effectuant une relaxation cyclique sur les paramètres (masses, raideurs, viscosités) ou groupes de paramètres : par exemple, au lieu de modifier cent paramètres, on modifie d'abord les dix premiers, puis les dix suivants, et ainsi de suite jusqu'aux dix derniers, pour ensuite revenir aux dix premiers, etc. Dans cet exemple, le temps de calcul pour chaque adaptation est divisé par dix. On dispose de peu de résultats théoriques concernant ce type de méthodes ; on montre théoriquement sur des cas particuliers, et expérimentalement que la technique de relaxation cyclique converge plus lentement que la méthode de la plus forte pente (amélioration de la technique du gradient dans laquelle à chaque itération, on cherche le *minimum* de la surface d'erreur dans la direction du gradient) [Minoux 83]. C'est certainement le prix à payer pour atteindre une vitesse de simulation élevée.

Par ailleurs, l'accent doit être porté sur les conditions permettant à l'utilisateur humain du système d'apprentissage d'appliquer un geste approprié vers l'objet physique et de modifier efficacement les paramètres de contrôle de l'algorithme. Plusieurs problèmes se posent alors :

- En plus du geste, quels sont les paramètres de contrôle qui permettent à l'utilisateur de simplifier l'apprentissage en cours de simulation ? En particulier, il est possible d'affecter des pas d'apprentissage différents pour chacun des paramètres (masses, etc.) du réseau, mais on a alors énormément de réglages ; il faut trouver une solution intermédiaire entre un seul réglage pour tous les pas et autant de réglages que de pas (par exemple une focalisation sur une zone du réseau).
- Quelles informations présenter à l'utilisateur afin qu'il puisse trouver le bon geste et la bonne modification de paramètres de contrôle. Certaines de ces informations sont de type abstrait, concernant l'évolution de l'erreur et plus généralement le parcours sur la surface d'erreur. D'autres informations concernent le réseau lui même : voir le réseau se modifier en temps réel (à une fréquence de *30* Hz par exemple) peut être riche en information car cela peut permettre de constater par exemple qu'une partie du réseau ne s'adapte pas, ou au contraire s'est trop modifiée. Dans cette optique, il

est important de manipuler des réseaux dont la structure soit *lisible* : plutôt que de grands réseaux entièrement connectés, des modèles à connexions de voisinage (cordes ou membranes par exemple) doivent être développés. Cette lisibilité des structures est nécessaire au delà de la visualisation en temps réel : après apprentissage, il est important de pouvoir *observer* et interpréter la structure obtenue.

- Comment présenter les informations à l'utilisateur et lui permettre d'appliquer ses commandes ? Il s'agit d'un problème typique de conception d'une interface homme-machine, qu'il ne faudrait surtout pas négliger.

L'apprentissage actif progressif

Quelques expériences relatées au paragraphe III.3 ont mis en évidence les effets positifs de l'ordre de présentation des données à un réseau adaptatif. Ainsi, si un réseau pouvait *agir* sur l'environnement pour choisir quelles données apprendre, dans le sens d'un apprentissage progressif de ces données, alors un nouveau pas serait franchi pour l'apprentissage connexionniste.

Bien que, comme nous l'avons mis en évidence, peu de chercheurs abordent la question, cette démarche s'inscrit bien dans la tendance actuelle des réseaux supervisés. En effet, historiquement, les réseaux à rétropropagation du gradient tels qu'ils ont été proposés dans le milieu des années *1980* [Rumelhart *et al.* 86][Le Cun 87], sont essentiellement "monoblocs" et homogènes. Les défauts de cette approche ont été bien identifiés : difficulté pour choisir la taille du réseau [Fiesler 94][Jutten & Chentouf 95], structure arbitraire pas nécessairement adaptée à la tâche, mauvaises capacités de généralisation [Alpaydin 91][Le Cun 89], difficultés d'apprentissage [Fahlman & Lebiere 91][Haykins 94], défauts qui ont amené les chercheurs à proposer d'autres approches :

- l'approche modulaire "manuelle", qui consiste à diviser le réseau en petits modules et à entraîner séparément chaque module [Hrycej 92][Kämmerer & Küpper 89][Haffner *et al.* 88] ;
- l'approche modulaire automatique, qui consiste à avoir plusieurs modules, qui se spécialisent au fur et à mesure de l'apprentissage dans une sous-partie de l'espace d'entrée [Jacobs *et al.* 91] ;
- l'approche incrémentale, qui consiste à ajouter les unités une par une dans le réseau [Fahlman & Lebiere 91][Fahlman 91][Alpaydin 91][Jutten & Chentouf 95] ;
- l'approche modulaire automatique et incrémentale, qui combine les deux aspects précédents [Chen 93].

Toutes ces approches ont en commun qu'elles cherchent à "casser la complexité" du réseau, en le construisant partie par partie. L'apprentissage actif progressif va

exactement dans le même sens, mais du point de vue des données présentées au réseau : il s'agit de "casser la complexité" de l'environnement.

Pour avancer dans cette voie, nous estimons qu'il est grandement profitable de se pencher sur un cas où l'apprentissage progressif est omniprésent : l'être humain. Comme il a été écrit dans ce mémoire, la progressivité de l'apprentissage humain peut s'observer à de nombreux niveaux : automatisation des traitements [Perruchet 88a], développement de capacités de résolution de problèmes chez l'enfant [Shultz *et al.* 94], développement de la mémoire à court terme [Elman 93], entraînement à des conduites motrices [Famose 90], etc.

Quel que soit le niveau d'étude choisi, le système d'apprentissage qui en sera inspiré sera nécessairement complexe, car à la place du réseau de neurones traditionnel, utilisé pour résoudre une tâche donnée bien cernée, on sera amené à construire un système cognitif plus général, qui résout successivement plusieurs tâches, et acquiert une connaissance *avant* et *après* la tâche spécifique qu'il doit résoudre.

Cette idée est abordée par d'autres chercheurs, et prend le nom d'apprentissage *continu* [Ring 94] ou *apprentissage sur toute une vie* ("longlife learning") [Thrun & Mitchell 94] (mais les systèmes réalisés par ces auteurs sont encore subordonnés au concepteur humain qui choisit les tâches successives). Si on applique cette idée à la résolution de problèmes industriels, il faut imaginer, avec certes un peu d'utopie, un système qui, confronté aux données d'un problème spécifique, est capable d'exploiter toute les expériences qu'il a pu accumuler auparavant en résolvant diverses autres problèmes ; en d'autre termes, c'est un système qui a sa propre histoire.

*

Finalement, les deux perspectives proposées, à savoir l'apprentissage coopératif et l'apprentissage actif progressif, quoique très différentes, nous suggèrent que l'avenir des Réseaux de Neurones Formels n'est peut-être pas tant dans les algorithmes d'apprentissage utilisés pour rendre un réseau adaptatif que dans l'architecture cognitive dans laquelle ce réseau adaptatif s'insère.

REFERENCES BIBLIOGRAPHIQUES

[Ackey *et al.* 85] David H. Ackley, Geoffrey E. Hinton & Terrence J. Sejnowski. A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, **9**, p. 147-169, 1985.

[Allen & Brooks 91] Scott W. Allen & Lee R. Brooks. Specializing the Operation of an Explicit Rule. *Journal of Experimental Psychology: General*, **120**(1), p. 3-19, 1991.

[Allred & Kelly 89] Lloyd Allred & Gary E. Kelly. Supervised Learning Techniques for Backpropagation Networks. *Proc. IJCNN*, vol. I, p. 721-728, Washington DC, 1989.

[Alpaydin 91] Ethem Alpaydin. GAL: Networks that grow when they learn and shrink when they forget. Rapport technique, TR 91-032, ICSI, Berkeley, mai 1991.

[Amy & Decamp 88] Bernard Amy & Erik Decamp. *Neurocalcul et réseaux d'automates - Le point sur les recherches et les applications*, Nanterre, France: EC2, 1988.

[Anderson 90] John R. Anderson. *Cognitive psychology and its implications - Third edition*, New York: W. H. Freeman and Company, 1990.

[Atlas *et al.* 90] Lee Atlas, David Cohn, Richard Ladner, M. A. El-Sharkawi, R. J. Marks II, M. E. Aggoune & D. C. Park. Training Connectionist Networks with Queries and Sampling. In D. S. Touretzky (Ed.): *Advances in Neural Information Processing Systems 2*, p. 566-573, San Mateo: Morgan-Kaufmann, 1990.

[Barsalou 92] Lawrence W. Barsalou. *Cognitive psychology : an overview for cognitive scientists*, Hillsdale, New Jersey : L. Erlbaum Associates, 1992.

[Baum 91] Eric B. Baum. Neural Algorithms That Learn in Polynomial Time from Examples and Queries. *IEEE Trans. on Neural Networks*, **2**(1), p. 5-19, janvier 1991.

[Baum & Lang 91] Eric B. Baum & Kevin J. Lang. Constructing Hidden Units using Examples and Queries. In R.P. Lippmann *et al.* (Eds) *Advanced in Neural Information Processing 3*, San Mateo, California: Morgan Kaufman, p. 904-910, 1991.

[Baum & Haussler 89] Eric B. Baum & David Haussler. What Net Gives Valid Generalization? *Neural Computation*, **1**, p. 151-160, 1989.

[Beaufays & Wan 94] Françoise Beaufays & Eric A. Wan. Relating Real-Time Backpropagation and Backpropagation-Through-Time: An Application of Flow Graph Interreciprocity. *Neural Computation*, **6**, p. 296-306, 1994.

[Beslon & Biennier 94] Guillaume Beslon & Frédérique Biennier. Architectures Connexionnistes pour le "Behavior-Based-Control". *Actes. NSI'94*, Chamonix, 2-5 mai 1994.

[Bianchini *et al.* 92] M. Bianchni, M. Gori & M. Maggini. On the Problem of Local Minima in Recurrent Neural Networks. *IEEE Trans. on Neural Networks*, **5**(2), p. 167-177, mars 1994.

[Biehl & Schwarze 92] M. Biehl & H. Schwarze. On-Line Learning of a Time-Dependant Rule. *Europhysics Letters*, **20**(8), p. 733-738, 1992.

[Bonomi & Lutton 88] Ernesto Bonomi & Jean-Luc Lutton. Le recuit simulé. *Pour la Science*, **129**, juillet 1988.

[Boon *et al.* 93] Jean-Pierre Boon, Uriel Frish, Dominique d'Humières. L'hydrodynamique modélisée sur réseau. *La Recherche*, **253**(24), p. 391-399, avril 1993.

[Borin *et al.* 92] Gianpaolo Borin, Giovanni De Poli & aoûto Sarti. Algorithms and Structures for Synthesis Using Physical Models. *Computer Music Journal*, **16**(4), p. 30-42, Winter 1992.

[Boutillon & Guérard 95] Xavier Boutillon & Jean Guérard. Hybrid Synthesis. *Proc. of the International Symposium on Musical Acoustics*, p. 440-445, Dourdan, France, 2-6 juillet 1995.

[Cachin 94] Christian Cachin. Pedagogical Pattern Selection Strategies. *Neural Networks*, 7(1), p. 175-181, 1994.

[Cadoz 90] Claude Cadoz. Simuler pour connaître / Connaître pour simuler - Réflexions sur la représentation, la Modélisation, la Simulation et la Création avec l'Ordinateur. *Actes du colloque Modèles physiques création musicale et ordinateur*, vol. III, p. 661-708, Grenoble, France, 1990 (Paris: Ed. de la maison des sciences de l'homme, 1994).

[Cadoz *et al.* 81] Claude Cadoz, Annie Luciani & Jean-Loup Florens. Synthèse musicale par simulation des mécanismes instrumentaux. Transducteurs Gestuels Rétroactifs pour l'étude du jeu instrumental. *Revue d'acoustique*, **59**, p. 279-292, 1981.

[Cadoz *et al.* 90] Claude Cadoz, Leszek Lisowski & Jean-Loup Florens. A modular Feedback Keyboard Design. *Computer Music Journal*, **14**(2), p. 47-51, 1990.

[Cadoz *et al.* 93] Claude Cadoz, Annie Luciani & Jean-Loup Florens. CORDIS-ANIMA: a modelling and simulation system for sound and image synthesis - the general formalism. *Computer Music Journal*, **17**(1): p. 19-29, 1993.

[Catfolis 93] Thierry Catfolis. A method for Improving the Real-Time Recurrent Learning Algorithm. *Neural Networks*, **6**(6), p. 807-821, 1993.

[Camus 88] Jean-François Camus. La distinction entre les processus contrôlés et les processus automatiques chez Schneider et Shiffrin. In Pierre Perruchet (Ed.) *Les automatismes cognitifs*, p. 55-80, Liège: Pierre Mardaga, 1988.

[Chanclou & Luciani 95] Benoît Chanclou & Annie. Luciani. Physical models and dynamic simulation of planetary motor vehicles with a great number of degrees of freedom. *Proc. Intelligent Autonomous Sytems - 4*, Karlsruhe, Germany, mars 1995.

[Chen 93] James Ron Chen. Cascaded Partitioning Network. *Proc. of the World Congress On Neural Networks*, vol. III, p. 411-414, Portland, Oregon, 11-15 juillet 1993.

[Chen *et al.* 93] D. Chen, C. L. Giles, G. Z. Sun, H. H. Chen, Y. C. Lee, M. W. Goudreau. Constructive Learning of Recurrent Neural Networks. *Proc. IEEE ICNN*, vol. III, p. 1196-1201, San Francisco, 1993.

[Cleeremans 93] Axel Cleeremans. *Mechanisms of Implicit Learning - Connectionist Models of Sequence Processing*, in the Neural Networks Modeling and Connectionism series. Cambridge, Massachusetts : MIT Press, 1993.

[Cleeremans *et al.* 89] Axel Cleeremans, David Servan Schreiber & James L. McClelland. Finite State Automata and Simple Recurrent Networks. *Neural Computation*, **1**, p. 372-381, 1989.

[Cloete & Ludik 93] Ian Cloete & Jacques Ludik. Increased Complexity Training. *Proc. IWANN'93*, Sitges, Spain, juin 1993.

[Cloete & Ludik 94a] Ian Cloete & Jacques Ludik. Incremental Training Strategies. *Proc. ICANN*, p. 743-746, Sorrento, Italie, 26-29 mai 1994.

[Cloete & Ludik 94b] Ian Cloete & Jacques Ludik. Delta trainning strategies. *Proc. IEEE Conf. on Neural Networks*, p. 295-298, Orlando, Florida, 28 juin - 2 juillet 1994.

[Codina *et al.* 94] Joan Codina, J. Carlos Aguado & Josep M. Fuertes. Capabilities of a Structured Neural Network - Learning and Comparison with Classical Techniques. *Proc. ESANN*, p. 13-18, Brussels, Belgium, 20-22 avril , 1994.

[Cotrell & Tsung 91] Garrison W. Cotrell & Fu Sheng Tsung. Learning Simple Arithmetic Procedures. In J. A. Barnden, J. B. Pollack (Eds): *High-Level Connectionist Models*, dans la série Advances in Connectionist and Neural Computation Theory, **1**, p. 305-321, 1991.

[Cox & Darlington 94] Trevor J. Cox & Paul Darlington. Using neural networks to model non-linear acoustic propagation. *Acta Acustica*, **2**, p. 95-99, avril 1994.

[Crucianu 94] Mihail Crucianu. Représentation structurée dans les réseaux connexionnistes. Thèse de doctorat, Univ. Paris XI Orsay, 1994.

[Davalo & Naïm 90] Eric Davalo & Patrick Naïm. *Des Réseaux de Neurones.* Paris: Eyrolles, 1990.

[Delnondedieu *et al.* 93] Yves Delnondedieu, Annie Luciani & Claude Cadoz. Physical elementary component for medeling the sensory - motricity : the primary muscle. A. Luciani & D. Thalmann (Eds.) *Proc. Fourth Eurographics Animation and Simulation Workshop*, p. 193-207, Barcelona, Spain, 4-5 septembre 1993.

[Demongeot 92] Jacques Demongeot. *Systèmes dynamiques.* Notes de cours du DEA de Sciences Cognitives, Grenoble, 1992.

[Denis 92] Michel Denis. *La représentation mentale.* Notes de cours du DEA de Sciences Cognitives, Grenoble, 1992.

[Denoeux & Lengellé 93] Thierry Denoeux & Régis Lengellé. Initializing Back Propagation Networks With Prototypes.*Neural Networks*, **6**(3), p. 351-363, 1993.

[De Poli 83] Giovanni De Poli. A Tutorial on Digital Sound Synthesis Techniques. *Computer Music Journal*, **7**(4), p. 8-26, Winter 1983.

[Djoharian 93] Pirouz Djoharian. Generating Models for Modal Synthesis. *Computer Music Journal*, **17**(1), p. 57-65, 1993.

[Doya 94] Kenji Doya. Bifurcations of Recurrent neural networks in Gradient Decent Learning. Submitted to *IEEE Transactions on neural networks*, 1994.

[Elman 90] Jeffrey L. Elman. Finding Structure in Time. *Cognitive Science*, **14**, p. 179-211, 1990.

[Elman 93] Jeffrey L. Elman. Learning and development: the importance of starting small. *Cognition*, **48**, p. 71-99, 1993.

[Erwins 84] D. J. Erwins. *Modal Testing: Theory and Practice.* Taunton, Somerest, Angleterre: Research Study Press, 1984.

[Fahlman & Lebiere 91] Scott E. Fahlman & Christian Lebiere. The Cascade-Correlation Learning Architecture. Rapport Technique, CMU-CS-90-100, août 1991.

[Fahlman 91] Scott E. Fahlman. The Recurrent Cascade-Correlation Architecture. Rapport Technique, CMU-CS-91-100, mai 1991.

[Famose 90] Jean-Pierre Famose. *Apprentissage moteur et difficulté de la tâche.* INSEP, 1990.

[Famose, *et al.* 91] J. P. Famose, P. Fleurance & Y. Touchard. *L'apprentissage moteur.* Paris: Revue E.P.S., 1991.

[Fiesler 94] Emile Fiesler. Comparative Bibliography of Ontogenic Neural Networks. *Proc. ICANN*, p. 793-796, Sorrento, Italie, 26-29 mai 1994.

[Gherrity 89] Michael Gherrity. A learning algorithm for analog fully recurrent neural networks. *Proc. IJCNN*, vol. I, p. 643-644, Washington DC, 1989.

[Giacometti 92] Arnaud Giacometti. Modèles hybrides de l'expertise. Thèse de doctorat, ENST, Paris, 1992.

[Giles *et al* 92] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, Y. C. Lee. Learning and Extracting Finite State Automata with Second Order Recurrent Neural Networks. *Neural Computation*, **4**, p. 393-405, 1992.

[Greenspan 73] Donald Greenspan. *Discrete Models.* Addison-Wesley, 1973.

[Greussay 88] Patrick Greussay. L'ordinateur cellulaire. *La Recherche*, novembre 1988.

[Guyon *et al.* 88] I. Guyon, L. Personnaz, G. Dreyfus. Of points and loops. In *Neural Computers*, Springer Verlag, 1988.

[Haffner *et al.* 88] P. Haffner, A. Waibel, H. Sawai & K. Shikano. Fast Back-Propagation Learning Methods for Neural Networks in Speech. ATR Technical Report TR-I-0058, 1988.

[Handelman *et al.* 92] David A. Handelman, Stephen H. Lane & Jack J. Gelfand. Robotic skill acquisition based on biological principles. In Abraham Kandel & Gideon Langholz (Eds.): *Hybrid Architectures for Intelligent Systems*, p. 301-327, CRC Press, 1992.

[Hatwell 86] Yvette Hatwell. *Toucher l'Espace.* Presse Universitaire de Lille, 1986.

[Hayes & Broadbent 88] Neil A. Hayes, Donald E. Broadbent. Two modes of learning for interactive tasks. *Cognition*, **28**, p. 249-276, 1988.

[Haykin 94] Simon Haykin. *Neural networks - A Comprehensive Foundation.* New-York: Macmillan College Publishing Company, 1994.

[Hecht-Nielsen 90] Robert Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, 1990.

[Hérault & Jutten 94] Jeanny Hérault & Christian Jutten. *Réseaux neuronaux et traitement du signal.* Paris: Hermès, 1994.

[Hopfield 82] J. J. Hopfield. Neural Networks and physical systems with emergent collective computationnal abilitie. *Proc. Natl. Acad. Science USA*, **79**, p. 2554-2558, 1982.

[Hrycej 92] Tomas Hrycej. *Modular Learning in Neural Networks - a Modularized Approach to Neural Networks Classification* in Branko Soucek (Ed.): Sixth-Generation Computer Technology Series. Wiley-Interscience, 1992.

[Hwang *et al.* 91] Jenq-Neng Hwang, Jai J. Choi, Seho Oh & Robert J. Marks II. Query-Based learning applied to Partially Trained Multilayer Perceptrons. *IEEE Trans. on Neural Networks*, **2**(1), p. 131-136, janvier 1991.

[Incerti & Cadoz 95] Eric Incerti & Claude Cadoz. Time-frequency decomposition by means of physical model. *Proc. of the International Symposium on Musical Acoustics*, p. 458-464, Dourdan, France, 2-6 juillet 1995.

[Jacobs 88] Robert A. Jacobs. Initial Experiments On Constructing Domains of Expertise and Hierarchies In Connectionist Systems. *Proc. Connectionist Models Summer School*, p. 144-153, 1988.

[Jacobs *et al.* 91] Robert A. Jacobs, Michael I. Jordan & Andrew G. Barto. Task Decomposition Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks. *Cognitive Science*, **15**, p. 219-250, avril 1991.

[Jaffe 95] David A. Jaffe. Ten Criteria for Evaluating Synthesis Techniques. *Computer Music Journal*, **19**(1), p. 76-87, 1995.

[Jimenez 93] Stéphane Jimenez. Modélisation et simulation physique d'objets volumiques déformables complexes. Thèse de doctorat, INPG, Grenoble, Novembe 1993.

[Jodouin 93] Jean-François Jodouin. Réseaux de neurones et Traitement du Langage Naturel - Etude des réseaux récurrents et de leurs représentations. Thèse de doctorat, Univ. Paris XI Orsay, mars 1993.

[Jordan 88] Michael I. Jordan. Supervised learning and systems with excess degrees of freedom. COINS Technical Report 88-27, MIT, 1988.

[Jutten & Chentouf 95] Christian Jutten & Rachida Chentouf. A new scheme for incremental learning. *Neural Processing Letters*, **2**(1), p. 1-4, 1995.

[Kamimura 92] Ryotaro Kamimura. Activated Hidden Connections to Accelerate the Learning in Recurrent Networks. *Proc. IJCNN*, vol. I, p. 693-700, Baltimore,1992.

[Kämmerer & Küpper 89] Bernhard, R. Kämmerer, Wolfang A. Küpper. Design of hierarchical structures and their application to the task of isolated-word recognition. *Proc. IJCNN*, vol. I, p. 243-249, Washington DC, 1989.

[Kaufmann 70] A. Kaufmann. *Des points et des flèches ... la théorie des graphes*. Paris: Dunod, 1970.

[Kohonen 82] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, **43**, p. 59-69, 1982.

[Kolen & Pollack 91] John F. Kolen & Jordan B. Pollack. Back Propagation is Sensitive to Initial Conditions. In R.P. Lippmann *et al.* (Eds) *Advanced in Neural Information Processing 3*, San Mateo, California: Morgan Kaufman, p. 860-867, 1991.

[Labbi 93] Abderrahim Labbi. Sur l'approximation et les systèmes dynamiques dans les réseaux neuronaux. Thèse de doctorat, Université Joseph Fourier, Grenoble, 1993.

[Lambert & Hecht-Nielsen 91] Jean Michel Lambert & Robert Hecht-Nielsen. Application of feedforward and Recurrent neural Networks to Chemical Plant Modelling. *Proc. IJCNN*, vol. I, p. 373-378, Seattle, 1991.

[Latimer 94] Dany Latimer. Computer Modelling of Cognitive Processes. In J. Wiles, C. Latimer & C. Stevens (Eds.) *Collected papers from a Symposium on Connectionist Models and Psychology*, février 1994.

[Le Cun 87] Yann Le Cun. Modèles connexionistes de l'apprentissage. Thèse de doctorat, Université Paris VI, 1987.

[Le Cun 89] Yann Le Cun. Generalization and Network Design Strategies. In R. Pfeifer, Z. Schreter, F. Fogelman-Soulié & L. Steels (Eds) *Connectionism in perspective*, North Holland: Elsevier Science Publishers B.V, 1989.

[Li 92] Leong Kwan Li. Approximation theory and recurrent networks. *Proc. IJCNN*, vol. II, p. 286-271, Baltimore, 1992.

[Li 90] Shidong Li. An optimized Backpropagation with Minimum Norm Weights, *Proc. IJCNN*, vol. I, p. 697-702, Seattle, 1991.

[Logan 88] Gordon D. Logan. Toward an Instance Theory of Automatization. *Psychological Review*, **95**(4), p. 492-527, 1988.

[Logar *et al.* 93] Antonette M. Logar, Edward M. Corwin & William J.B. Oldham. A comparison of Recurrent Neural Networks Learning Algorithms. *Proc. IEEE Conf. on Neural Networks*, p. 1129-1134, San Francisco, 1993.

[Louchet 94] Jean Louchet. Identification évolutive de modèles physiques d'animation à base de masses-liaisons. *Actes du séminaire du groupe de travail de travail "Animation et Simulation"*, p. 79-83, Lille, France, 20-21 octobre, 1994.

[Luciani *et al.* 91] Annie Luciani, Stéphane Jimenez, Jean-Loup Florens & Olivier Raoult. Computational physics: a modeler simulator for animated physical objects. *Proc. of the European Computer Graphics Conference and Exhibition*, p. 425-436, Vienne, Autriche, septembre 1991.

[Luciani *et al.* 95] Annie Luciani, Arash Habibi & Emmanuel Manzotti. A Multi-Scale Physical Model Of Granular Materials. *Proc. Graphics Interface'95*, p. 136-146, Québec, 17-19 mai, 1995.

[Ludik & Cloete 93] Jacques Ludik & Ian Cloete. Training schedules for improved convergence. *Proc. IJCNN*, p. 561-164, Nagoya, Japon, 25-29 octobre, 1993.

[Ludik & Cloete 95] Jacques Ludik & Ian Cloete. Training Strategies for Architecture-specific Recurrent Neural Networks. *S. A. Computer Journal*, 1995.

[Ludik & Cloete 94] Jacques Ludik & Ian Cloete. Incremental Increased Complexity Training. *Proc. ESANN*, p. 161-165, Brussels, Belgium, 20-22 avril, 1994.

[McClelland 89] Parallel distributed processing: Implications for cognition and development. In R.G.M. Morris (Ed.) *Parallel distributed processing: Implications for psychology and neurobiology* , p. 8-45, Oxford University Press, 1989.

[Minoux 83] Michel Minoux. *Programmation mathèmatique - Théorie et algorithmes (Tome 1).* Coll. Technique et Scientifique des Télécommunications, Dunod, 1983.

[Mozer 88] Michael C. Mozer. A focused back-propagation algorithm for temporal pattern recognition, Rapport Technique, Univ. of Toronto, Dep^{ts} Psychology and Computer Science, CRG-TR-88-3, 1988.

[Munro 92] Paul W. Munro. Repeat Until Bored: A pattern Selection Strategy. In J. F. Moody *et al.* (Eds.): *Advances in Neural Information Processing Systems 4* , p. 1001-1008, San Mateo,CA: Morgan-Kaufmann, 1992.

[Murphy & Wright 84] Gregory L. Murphy & Jack C. Wright. Changes in Conceptual Structure With Expertise: Differences Between Real-World Experts and Novices. *Journal of Experimental Psychology*, **10**(1), p. 145-155, 1984.

[Murre 92] Jacob M. J. Murre. The Effect of Pattern Presentation on Interference in Backpropagation Networks. *Proc. of the fourthteenth Ann. Conf. of the Cognitive Science Soc.*, p. 54-59, Chicago, Illinois, 7-10 août, 1992.

[Narendra & Parthasarathy 90] Kumpati S. Narendra & Kannan Parthasarathy. Identifiaction and Control of Dynamical Systems Using Neural Networks. *IEEE Trans. on Neural Networks*, **1**(1), p. 4-27, mars 1990.

[Nassersharif & Rajan 93] B. Nassersharif & V. Rajan. Automated Training of Backpropagation Neural Networks Using a Real-Time Expert System. *Proc. of the World Congress On Neural Networks*, vol. I, p. 536-541, Portland, Oregon, 11-15 juillet 1993.

[Nerrand *et al.* 93] O. Nerrand, P. Roussel-Ragot, L. Personnaz & G. Dreyfus. Neural Networks and Nonlinear Adaptive Filtering: Unifying Concepts and New Algorithms. *Neural Computation*, **5**, p. 165-199, 1993.

[Newell & Rosenbloom 81] A. Newell & P.S. Rosenbloom. Mechanisms of skill acquisition and the law of practice. In J.R. Anderson (Ed.), *Cognitive skill and their acquisition* , p. 1-55, Hillsdale, NJ: Erlbaum, 1981.

[Nouiri 91] Jamel Nouiri. Modélisation et évaluation d'un Transducteur Gestuel à Retour d'effort. Rapport de stage de DEA, INPG, 1991.

[Omlin & Giles 93] Christian W. Omlin & C. Lee Giles. Pruning Recurrent Neural Networks for Improved Generalization Performance. Rapport Technique, Computer Science Departement, Rensselear Polytechnic Institute, Troy, N.Y, n° 93-6, avril 1993.

[Orsier 95] Bruno Orsier. Etude et application de systèmes hybrides neurosymboliques. Thèse de doctorat, Université Joseph Fourier, 1995.

[Pearlmutter 89] Barak A. Pearlmutter. Learning State Space Trajectories in Recurrent Neural Networks, *Neural Computation*, **1**, p. 263-269, 1989.

[Pearlmutter 90] Barak A. Pearlmutter. Dynamic Recurrent Neural Networks. Rapport Technique, CMU-CS-90-196, décembre 1990.

[Pearlmutter 95] Barak A. Pearlmutter. Gradient Calculations for Dynamic Recurrent Neural Networks: A Survey. *IEEE Trans. on Neural Networks*, **6**(5), p. 1212-1228, septembre 1995.

[Perruchet 88a] Pierre Perruchet (Ed.). *Les automatismes cognitifs*. Liège: Pierre Mardaga, 1988.

[Perruchet 88b] Pierre Perruchet. Une évaluation critique du concept d'automaticité. In Pierre Perruchet (Ed.). *Les automatismes cognitifs*, p. 27-54, Liège: Pierre Mardaga, 1988.

[Pierce 87] John R. Pierce. *Le son musical - musique, acoustique et informatique.* Paris: Pour la Science, 1987.

[Piché 94] Stephen W. Piché. Steepest Descent Algorithms for Neural Network Controllers and Filters. *IEEE Trans. on Neural Networks*, **5**(2), p. 198-212, 1994.

[Pineda 87] Fernando J. Pineda. Generalization of back propagation to recurrent neural networks, *Physical review letters*, **59**(19), p. 2229-2232, 1987.

[Plutowski & White 93] Mark Plutowski & Halbert White. Selecting Concise Training Sets from Clean Data. IEEE Trans. on Neural Networks, **4**(2), mars 1993.

[Plutowski *et al.* 95 Mark Plutowski, Garrison Cottrell & Halbert White. Experience with selecting exemplars from clean data. A paraître dans *Neural Networks*, 1995.

[Pratt *et al.* 91] Lorien Y. Pratt, Jack Mostow & Candace A. Kamm. Direct transfer of learned information among neural networks. *Proc. of the Ninth National Conference on Artificial Intelligence*, vol. II, p. 584-589, Menlo Park: MIT Press, 1991.

[Raoult 91] Olivier Raoult. Etude des oscillateurs numériques. Rapport de recherche ACROE 91-19, 1991.

[RayChaudhuri & Hamey 1995] T. RayChaudhuri & L.G.C. Hamey. An Algorithm for Active Data Collection for Learning - Feasibility Study with Neural Networks. Rapport technique, 95-173C, Dept of Computing, School of MPCE, Macquarie Univ., Australie, 1995.

[Raysz 91] Jean-Pierre Raysz. Algorithmes d'apprentissage pour réseaux connexionnistes récurrents, application à la modélisation de processus d'interprétation. Thèse de doctorat, Université de Caen, 1991.

[Reber 76] Arthur S. Reber. Implicit learning of synthetic languages: the role of instructional set. *Journal of Experimental Psycholgy: Human Learning and Memory*, **2**(1), p. 88-94, 1976.

[Richard *et al.* 90] J. F. Richard, C. Bonnet & R. Ghiglione. *Traité de psychologie cognitive tome II*, Paris: Dunod, 1990.

[Riedmiller 94] Martin Riedmiller. Rprop - Description and Implementation Details. Rapport Technique, Institut für Logik, Komplexität und Deduktionssysteme, Université de Karlsruhe, janvier 1994.

[Risset et Mahews 69] Jean-Claude Risset & Max V. Mathew. Analysis of musical-insrument tones. *P'hysics today*, **22**(2), février 1969.

[Roads 94] Curtis Roads. Physical modeling. *Keyboard*, p. 89-94 & 100-103, septembre 1994.

[Röbel 94] Axel Röbel. Dynamic Pattern Selection for Faster learning and Controlled Generalization of Neural Networks. *Proc. ESANN*, p. 187-192, Brussels, Belgium, 20-22 avril, 1994.

[Robinson 89] Anthony John Robinson. Dynamic Error Propagation Networks. PhD Thesis, Cambridge Univ., Engineering Departement, 1989.

[Rohwer 90] The "moving target" training algorithm. In L. B. Almeida & C. J. Wellekens (Eds.): *Proc. EURASIP Workshop on Neural Networks*, p. 101-109, Sesimbra, Portugal, février 15-17 1990. Dans la série Lecture notes in computer sciences, vol. 412, Springer, 1990.

[Rohwer & Forrest 87] Richard Rohwer, Bruce Forrest. Training Time Dependance in Neural Networks, *Proc. IEEE first annual International Conference on Neural Networks*, vol. II, p. 701-708, San Diego, 1987.

[Ronco 94] Eric Ronco. Apprentissage à complexité progressive dans les systèmes connexionnistes, Rapport de stage de DEA, LIFIA-IMAG, INPG, Grenoble, 1994.

[Rosenbaum 91] David A. Rosenbaum. *Human motor control*, San Diego, CA: Academic Press, 1991.

[Rumelhart & McClelland 86] David E. Rumelhart & James L. McClelland (Eds). *Parallel Distributed Processing*, volume 1, Cambridge, Mass.: MIT Press, 1986.

[Sanzogni & Vaccaro 93] Louis Sanzogni & John a. Vaccaro. Use of weighting functions for focusing of learning in artificial neural networks. *Neurocomputing*, **5**, p. 175-184, 1993.

[Schmindhuber 90] Jürgen Schmindhuber. A local learning algorithm for dynamic feedforward and recurrent networks. Rapport Technique, TUM, Munich, Allemagne, FKI-124-90, 1990.

[Schmindhuber 92] Jürgen Schmindhuber. Learning Complex, Extended Sequences Using the Principle of History Compression. *Neural Computation*, **4**, p. 232-242, 1992.

[Schneider & Shiffrin 77] W. Schneider & Richard. M. Shiffrin. Controlled and automatic human information processing. Detection, search and attention. *Psychological review* **84**, p. 1-66, 1977.

[Selfridge *et al.* 85] Olivier G. Selfridge, Richard S. Sutton & Andrew G. Barto. Training and tracking in robotics. *Proc. IJCAI*, p. 670-672, 1985.

[Sharkey & Sharkey 93] Noel E. Sharkey & Amanda J. C. Sharkey. Adaptive Generalization. *Artificial Intelligence Review*, **7**, p. 313-328, 1993.

[Shiffrin & Schneider 77] Richard. M. Shiffrin & W. Schneider. Controlled and automatic human information processing II. Perceptual Learning, Automatic Attending, and a General Theory. *Psychological review* **84**(2), p. 127-190, 1977.

[Shiffrin & Dumais, 81] Richard. M. Shiffrin & Susan T. Dumais. The development of automatism. In J. R. Anderson (Ed.): *Cognitive Skills and their acquisition*, p. 111-139, Hillsdale, New Jersey: LEA, 1981.

[Shultz *et al.* 94] Thomas R. Shultz, Denis Mareshal & William C. Shmidt. Modeling Cognitive Development on Balance Scale Phenomena. *Machine Learning*, **16**, p. 57-92, 1994.

[Smith 92] Julius O. Smith III. Physical Modeling Using Digital Waveguides. *Computer Music Journal*, **16**(4), Winter 1992.

[Smith & Zipser 89] Anthony W. Smith & David Zipser. Encoding sequential structure: experience with the real-time recurrent learning algorithm. *Proc. IJCNN*, vol. I, p. 645-648, Washington DC,1989.

[Srinivasan, *et al.* 94] B. Srinivasan, U.R. Prasad & N.J. Rao. Back Propagation Through Adjoints for the Identification of Nonlinear Dynamic Systems Using Recurent Neural models. *IEEE Transaction on Neural Networks*, **5**(2), p. 213-228, mars 1994.

[Stork 89] D.G. Stork. Is Backpropagation biologically plausible?. *Proc. IJCNN*, vol. II, p. 241-246, Washington, DC, juin 1989.

[Strand & Jones 92] Eugene M. Strand & Warren T. Jones. An active Pattern Set Strategy dor Enhancing Generalization While Improving Backpropagation Training Efficiency. *Proc. IJCNN*, vol. I, p. 830-834, Baltimore, 1992.

[Sun *et al.* 91] Guo-Zheng Sun, Hsing-Hen Che & Yee-Chun Lee. A Fast On-Line Learning Algorithm For Recurrent Neural Networks. *Proc. IJCNN*, vol. II, p. 13-18, Seattle,1991.

[Szilas 94] Nicolas Szilas. Utilisation de la "stratégie d'apprentissage" dans les systèmes connexionnistes. *Proc. Colloque Jeunes Chercheurs en Sciences Cognitives*, La Motte d'Aveillans, Isère, France, 23-25 mars 1994.

[Szilas & Cadoz 93] Nicolas Szilas & Claude Cadoz. Physical Models that learn. *Proc. International Computer Music Conference*, Tokyo, Japon, 10-15 septembre 1993.

[Szilas & Ronco 95] Nicolas Szilas & Eric Ronco. Action for learning in non-symbolic systems. *Proc. of the European Conference on Cognitive Science* , p. 55-63, Saint-Malo, France, 4-7 avril 1995.

[Thrun & Mitchell 94] Sebastian Thrun & Tom M. Mitchell. Learning one more thing. Rapport Technique, CMU-CS-94-184, Carnegie Mellon University, septembre 1994.

[Thrun & Møller 92] Sebastian B. Thrun & Knut Møller. Active Exploration in Dynamic Environments. *Proc. of NIPS-4*, Denver, Colorado, 1992.

[Toomarian & Bahren 91] N. Toomarian & J. Bahren. Fast Temporal Neural Learning Using Teacher Forcing. *Proc. IJCNN*, vol I, p. 817-822, Seattle, 1991.

[Toomarian & Bahren 92] Nikzad Benny Toomarian & Jacob Bahren. Learning a Trajectory Using Adjoint Functions and Teacher Forcing. *Neural Networks*, **5**, p. 473-484, 1992.

[Tsoi & Back 94] A.C. Tsoi & A. Back. Locally recurrent Globally Feedforward Networks. *IEEE Trans. on Neural Networks*, **5**(2), p. 229-239, 1994.

[Tsung 90] Fu-Sheng Tsung. Learning in recurrent Finite Difference Networks. In D. E. Touretzky, J. L. Elman, T. J. Sejnowski & G. E. Hinton (Eds.): *Proc. of the 1990 Connectionist Models Summer School*, p. 124-130, 1990 (San Mateo, CA: Morgan Kaufmann, 1991).

[Varela 89] Francisco J. Varela. *Connaître - les sciences cognitives - tendances et perspectives.* Paris: Seuil, 1989.

[Vuori & Vätimäki 93] Jarkko Vuori & Vesa Vätimäki. Parameter Estimation of Non-Linear Physical Models by Simulated Evolution - Application to the Flute Model. *Proc. ICMC'93*, p. 402-404, Tokyo, Japon, 10-15 septembre 1993.

[Waibel *et al.* 89] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano & Kevin J. Lang. Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Trans. on Accoustics, speech, and signal processing*, **37**(3), p. 328-339, mars 1989.

[Watrous *et al.* 90] Raymond L. Watrous, Bruce Ladendorf, Gary Kuhn (1990). Complete gradient optimization of a recurrent network applied to /b/, /d/, /g/ discrimination, *Journal of Acoutical Soc. Am.*, **87**(3), p. 1301-1309, mars 1990.

[Weisbuch 89] Gérard Weisbuch. *Dynamique des systèmes complexes - Une introduction aux réseaux d'automates.* InterEditions/Editions du CNRS, 1989.

[Widrow & Stearns 85] Bernard Widrow & Samuel D. Stearns. *Adaptive signal processing.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

[Wieland & Leighton 88] Alexis Wieland & Russel Leighton. Shaping schedules as a method for accelerated learning. *First Ann. INNS Meeting*, p. 231 (résumé), Boston, 1988

[Williams 92] Ronald J. Williams. Training Recurrent Networks Using the Extended Kalman Filter. *Proc. IJCNN*, vol. IV, p. 241-246, Baltimore, 1992.

[Williams & Peng 90] Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, **2**, p. 490-501, 1990.

[Williams & Zipser 89a] Ronald J. Williams & David Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, **1**, p. 270-280, 1989.

[Williams & Zipser 89b] Ronald J. Williams & David Zipser. Experimental Analysis of the Real-time Recurrent Learning Algorithm. *Connection Science*, **1**(1), p. 87-111, 1989.

[Winnykamen 90] Fayda Winnykamen. *Apprendre en imitant*. P.U.F., Psychologie d'aujourdhui, 1990.

[Wolfram 86] S. Wolfram. *Theory and applications of cellular automata*. World Scientific, 1986.

[Wu *et al.* 90] X. Wu, J. H. Garett, J. Gaboussi. Representation of Material Behavior : Neural Network-based Models. *Proc. IJCNN*, vol. I, p. 229-234, San Diego, 1990.

[Zhang 94] Byoung-Tak Zhang. Accelerated learning by active example selection. *International Journal of Neural Systems*, **5**(1), p. 67-75, mars 1994.

[Zipser 89] David Zipser. A Subgrouping Strategy that Reduces Complexity and Speeds Up Learning in Reurrent Networks. *Neural Computation*, **1**, p. 552-558, 1989.

Mots-clefs : connexionnisme, réseaux de neurones, réseaux récurrents, informatique musicale, modèles physiques, apprentissage, apprentissage actif, apprentissage progressif.

Résumé : Issus d'une analogie avec les réseaux de neurones biologiques du cerveau, les *réseaux récurrents* sont utilisés pour modéliser des comportements dynamiques complexes et pour reproduire - apprendre - ces comportements. Les propriétés adaptatives de ces réseaux peuvent être exploitées par les réseaux de modélisation physique de phénomènes vibratoires dédiés à la simulation informatique d'instruments de musique. Ces réseaux de modélisation mécanique possèdent des paramètres d'inertie, d'élasticité et de viscosité que l'on souhaite déterminer automatiquement dans le but de reproduire un comportement physique donné ; cette détermination est possible grâce aux réseaux récurrents. Nous développons ainsi un certain nombre d'algorithmes de réseaux de modélisation physique adaptatifs et proposons des algorithmes originaux, inspirés de modèles mécaniques.
En particulier, ce travail aborde la notion d'interaction avec l'environnement dans ce type de réseaux, et plus généralement dans les réseaux connexionnistes supervisés. A travers plusieurs expériences, nous montrons que, sous certaines conditions, l'interaction avec l'environnement permet la réussite de l'apprentissage, en particulier si cette interaction autorise un apprentissage à complexité progressive. De plus, nous établissons des rapprochements entre ce type d'apprentissage et certains apprentissages humains.
Cela nous amène à poser les bases d'un système d'identification de paramètres pour la modélisation d'instruments de musique. Ce système fait interagir en temps réel un instrumentiste, un instrument de musique et un ordinateur simulant le modèle adaptatif.

Keywords: Connectionism, neural networks, recurrent networks, computer music, physical models, learning, active learning, progressive learning.

Abstract: Recurrent neural networks, which are inspired from biological principles, are used to model complex dynamic behaviours and to reproduce - to learn - such behaviours. The adaptive properties of these nets can be applied to physical modelling networks dedicated to the simulation of musical instruments.
These physical modelling nets contain inertia, stiffness and damping parameters that one wishes to set automatically in order to reproduce a given mechanical behaviour ; this is possible thanks to recurrent neural algorithms. Algorithms for the parameter adaptation of physical models are developed and simulated. Original algorithms are also developed, based on mechanical principles.
In particular, this work is concerned with the notion of interaction with the environment for this type of networks and in general for supervised connectionist networks. Thanks to several experiments, it is shown that, under certain conditions, the interaction makes the learning succeed, especially if the interaction with the environment allows learning of progressive complexity. Furthermore, some analogies between this kind of learning and human learning are established.
At last, we present the basis of a system for parameter identification in physically based models of musical instruments. This system involves the interaction of an instrumentalist, an instrument and a computer that simulates the adaptive network.