UNIVERSITÉ DU QUÉBEC À MONTRÉAL

CAPACITÉ D'UNE MÉMOIRE ASSOCIATIVE À FONCTION DE SORTIE CHAOTIQUE

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

MOUNIA CHERIF

DÉCEMBRE 2010

REMERCIEMENTS

Mes remerciements vont aux Professeurs Mounir Boukadoum de l'UQÀM, et Sylvain Chartier de l'université d'Ottawa. Qu'ils trouvent dans cet ouvrage un témoignage de ma profonde reconnaissance.

Je présente ma gratitude à tous ceux qui ont pu contribuer de près ou de loin à l'aboutissement de ce travail.

TABLE DES MATIÈRES

LISTE DES FIGURES

RÉSUMÉ

Un des thèmes de recherche privilégié pour les sciences cognitives et l'intelligence artificielle est l'étude des capacités d'association du cerveau humain. L'objectif est de développer des modèles de mémoires dotés de caractéristiques similaires, que ce soit en termes d'adaptabilité, d'efficacité, ou de robustesse. Plusieurs modèles de mémoires associatives ont été développés et présentés dans la littérature, parmi eux le modèle de mémoire associative bidirectionnelle BAM de Kosko (Kosko, 1988). Ce modèle utilise une règle d'apprentissage hebbienne qui le rend plausible biologiquement, mais il possède plusieurs limitations cependant. En effet, sa règle d'apprentissage impose des contraintes d'orthogonalité entre les différents motifs appris qui entraine une faible capacité de mémorisation et une faible résilience face au bruit. De plus, le modèle peut apprendre uniquement des patrons encodés en binaire et linéairement séparables.

De nombreux efforts ont été, et continuent aujourd'hui à être déployés pour tenter d'améliorer le modèle de Kosko. La plupart visent l'augmentation de la capacité de stockage et l'amélioration de la performance de rappel. Quelques-uns des modèles proposés réussissent à classifier des problèmes non séparables linéairement, mais s'éloignent de l'architecture originale de Kosko ou parfois, utilisent des méthodes d'apprentissage qui s'écartent du principe de Hebb, ce qui les rend moins plausibles biologiquement.

Dans le présent mémoire, nous approfondissons l'étude d'un modèle récent de BAM, proposé par Chartier et Boukadoum (2006a) et caractérisé par une fonction de sortie chaotique, une architecture asymétrique, et une règle d'apprentissage hebbienne modifiée. Plus spécifiquement, nous étudions l'impact de modifier la fonction de sortie, en lui ajoutant un paramètre d'asymétrie, sur la capacité du réseau à traiter des tâches de classification non linéairement séparables. Nous nous inspirons de la théorie des catastrophes pour le cadre théorique de notre étude.
Nous expérimentons sur le modèle en vue d'améliorer sa performance de classification sans complexifier son architecture ou nous écarter de la plausibilité biologique de la règle d'apprentissage. Pour ce faire, nous utilisons et comparons plusieurs algorithmes de recherche heuristiques, dont certains inspirés de l'évolution naturelle, afin de concevoir des modèles de classification puissants, potentiellement capables de reproduire l'efficacité des processus cognitifs naturels.

Les principes exposés dans ce mémoire, se sont montrés efficaces pour le modèle BAM et peuvent faire l'objet de recherches intéressantes, notamment pour l'amélioration du potentiel des modèles connexionnistes récurrents.

Mots-clés: mémoire associative bidirectionnelle, réseaux de neurones artificiels, classification, dynamique chaotique, catastrophe fronce.

INTRODUCTION

Le modèle BAM proposé par Bart Kosko (Kosko, 1988) est une mémoire hétéro-associative qui généralise le modèle de Hopfield (Hopfield, 1982). Une de ses principales limitations est sa faible capacité de stockage. Wasserman (Wasserman, 1989) démontre que la capacité de stockage du modèle, lorsque n est la taille de la plus petite couche, est de $n/2\log_2 n$. La BAM de Kosko utilise aussi des interconnections symétriques qu'on ne retrouve pas dans le cerveau ; par conséquent, elle est peu plausible d'un point de vue biologique. Un autre problème est l'interférence catastrophique, une conséquence de la mémorisation de stimuli corrélés. Enfin, le modèle de Kosko peut mémoriser uniquement des patrons encodés en binaire et linéairement séparables.
Une approche pour surmonter les faiblesses de la BAM-Kosko fut utilisée par Personnaz, Guyon et Dreyfus (1986) ; elle emploie une matrice de projection basée sur la minimisation de l'erreur quadratique moyenne. Cette solution augmente la capacité de stockage et la performance de rappel, mais sa règle d'apprentissage, basée sur un principe de matrice inverse (pseudo-inverse), n'est pas un processus local. En effet, trouver l'inverse d'une matrice nécessite le calcul d'un déterminant qui lui a besoin de l'information de toutes les unités du réseau; par conséquent, le processus est global et de ce fait, l'apprentissage n'est pas de type hebbien.

Récemment, un modèle BAM a été introduit (Chartier, Renaud et Boukadoum, 2008; Chartier, Boukadoum et Amiri, 2009) possédant une dynamique non linéaire avec une fonction de sortie chaotique. Ce modèle effectue un apprentissage en ligne en utilisant une boucle de rétroaction permettant de faire contribuer la fonction de sortie à la convergence des poids des connexions. Ce même modèle, utilisé avec des limites de saturation dans la fonction de sortie (Chartier et Boukadoum, 2006a) est parmi les rares modèles capables de créer des attracteurs à valeurs réelles dans l'espace d'états sans aucun traitement préalable. C'est également un modèle capable de réduire le nombre de faux attracteurs tout en maintenant une bonne performance vis-à-vis du bruit et une bonne capacité de stockage.

Dans une mémoire associative bidirectionnelle, chaque état d'équilibre (ou point fixe) divise l'espace d'états. Cependant, plus le nombre de prototypes appris augmente, plus les rayons d'attraction qui leur sont associés diminuent (Kanter et Sompolinsky, 1987). Un modèle BAM basé sur la technique de pseudo-inverse comprenant n unités peut réaliser jusqu'à n-1 associations (Kanter et Sompolinsky, 1987). À la limite (n-1), les rayons d'attraction sont nuls et le réseau ne peut plus traiter d'exemplaires,

perdant ainsi sa capacité à la tolérance au bruit. Dans la plupart des conceptions de modèles BAM, le problème est évité en fixant le rapport des patrons par rapport au nombre d'unités à environ 50% (Personnaz, Guyon et Dreyfus, 1986). À ce niveau, le modèle peut néanmoins rappeler les patrons stockés à partir de leurs versions bruitées avec des niveaux de bruit allant de faibles à modérés. Le modèle BAM offre de ce fait une capacité de stockage de n-1 si on utilise la pseudo-inverse (ou autre technique similaire) comparativement à $n/2\log_2 n$ pour un apprentissage strictement hebbien. Si on veut tenir compte du bruit, alors pour un bon niveau de bruit la capacité de stockage effective pour la pseudo-inverse est d'environ $n/2$, alors que la capacité de stockage pour apprentissage strictement hebbien sera inférieure à $n/2\log_2 n$.

Les principales limites des modèles BAM sont leur faible capacité de stockage et leur incapacité à mémoriser des patrons non linéairement séparables. Une solution qui permet de contrer ces limitations, particulièrement la seconde, est de permettre au réseau d'ignorer certains points fixes pendant la phase de rappel, sans pour autant les effacer de la mémoire. De cette manière, on réduit le nombre de bassins d'attraction tout en conservant la même surface d'attraction. Les attracteurs restants bénéficieraient ainsi d'une augmentation de leurs rayons d'attraction. Ce qui représente une bonne façon de contourner leur limitation. Pour ce faire, on introduit un paramètre d'asymétrie à la fonction de sortie durant la phase de rappel. Ce paramètre va forcer le système à converger vers un attracteur particulier en biaisant l'espace de recherche en faveur des régions à plus forte probabilité. Un des mécanismes permettant de contrôler les bassins d'attraction est l'apprentissage par renforcement (RL). On peut en effet, retrouver le paramètre asymétrique approprié via un procédé essai/erreur. Avec un tel procédé, la BAM essaie ce qu'elle connaît déjà afin d'obtenir une récompense en plus d'explorer de nouveaux comportements afin d'effectuer des meilleurs choix dans le futur. Par conséquent, la BAM découvre d'elle-même l'action qui rapporte la meilleure récompense en les essayant. Le procédé RL va permettre de retrouver, pour chaque patron d'entrée, le paramètre d'asymétrie approprié pour la fonction de sortie. Ce paramètre va en quelque sorte forcer la BAM à converger vers l'attracteur désiré.

Trouver le paramètre d'asymétrie adéquat pour la fonction de sortie constitue l'objet de ce travail. Pour ce faire, on utilise un procédé qui exploite les propriétés d'une catastrophe fronce (*cusp*) (Chartier, Boukadoum et Amari, 2009) dans la dynamique de la BAM. La fronce représente une variation dans le comportement d'un système dynamique qui résulte de petits changements de paramètres. On montre que la fonction de transmission du modèle BAM, décrite dans (Chartier et Boukadoum, 2006), peut être modifiée pour inclure un paramètre d'asymétrie "$\boldsymbol{h}$". Celui-ci va permettre la création d'une catastrophe de type fronce qui va modifier le comportement qualitatif de la BAM. Il sera montré qu'en

modifiant ce paramètre supplémentaire, on parvient à améliorer la performance de classification de la BAM en incluant une information préalable. L'information préalable à fournir à la BAM sera déterminée au moyen d'algorithmes d'optimisation stochastiques de type évolutionnaires, parmi eux les algorithmes génétiques, les essaims particulaires et les colonies de fourmis.

La suite de ce mémoire est divisée comme suit : le premier chapitre présente une revue des principaux modèles neuronaux utilisés en classification et en reconnaissance ; le second chapitre présente différentes approches évolutionnaires pour la résolution de problèmes d'optimisation par recherche heuristique, comme les algorithmes génétiques et l'optimisation par colonies de fourmis et essaims particulaires; ces algorithmes serviront à trouver les paramètres asymétriques optimaux pour la fonction de sortie. Le Chapitre 3 présente, dans un premier temps, des notions sur les systèmes dynamiques et chaotiques et, par la suite, l'étude théorique sur laquelle nous nous sommes basés pour modifier la fonction de sortie de la BAM. Enfin, dans le chapitre 4, nous présentons les résultats expérimentaux obtenus à partir des simulations pour la classification de problèmes non linéairement séparables et pour l'amélioration de la performance de rappel. Nous concluons notre étude en mettant de l'avant les avantages de l'utilisation des modèles connexionnistes hybrides et d'une géométrie de catastrophe fronce (*cusp*) pour la réalisation de modèles de classification performants, robustes et possédant une meilleure capacité de stockage.

Chapitre 1

NOTIONS FONDAMENTALES SUR LES MODÈLES DE CLASSIFICATION

1.1 Introduction

Les réseaux de neurones artificiels sont des modèles formels pouvant permettre de rendre compte de comportements sous-jacents à des résultats empiriques observés. Ils sont essentiellement utilisés pour faire de la classification et de la catégorisation, mais peuvent aussi servir à modeler des processus cognitifs.

1.2 Classification et séparabilité linéaire

La classification a longtemps été le domaine privilégié d'utilisation des réseaux de neurones. Dans ce cadre d'application, on peut considérer qu'un réseau de neurones réalise une estimation de la probabilité d'appartenance d'un objet à une classe parmi d'autres. Par exemple, un classificateur à deux classes définit une surface de séparation dans l'espace des entrées. Un objet inconnu est affecté à l'une ou l'autre des deux classes relativement à sa position par rapport à la surface de séparation, la plus simple étant un hyperplan. Dans le cas où l'on peut séparer l'espace avec un hyperplan[1], on dit que la fonction à approcher est linéairement séparable (ex. fonction OR, fig. 1.1 (a)). Dans le cas contraire, on dit que la fonction est non linéairement séparable (ex. fonction XOR, fig. 1.1(b)).

La droite séparant les deux classes est déterminée par les poids synaptiques, a, b et la valeur seuil t de l'équation suivante :

$$ax_1 + bx_2 = t$$

Dans le cas du problème XOR, la séparabilité linéaire exige qu'une ligne droite soit tracée séparant les points (-1, -1) et (1, 1) des points (1, -1) et (-1, 1), ce qui est clairement impossibles (fig. 1.1 (b)).

---

[1] Dans le cas d'un espace bidimensionnel (fig. 1.1), la fonction XOR consiste à séparer les points (-1, -1) et (1, 1) des points (-1, 1) et (1, -1). C'est en fait un cas particulier d'un problème plus général qui consiste à classifier des points sur un hypercube-unité tels que chaque point sur l'hypercube soit dans la classe -1 ou dans la classe 1.

Cependant, on peut résoudre le problème en se servant d'un réseau qui réalise une fonction de séparation non linéaire ou bien un réseau capable de tracer deux surfaces de décision parallèles et en définissant les classes par rapport à ces deux surfaces (figure 1.1 (b)) ; c'est ce que fait le réseau très connu MLP (« Multi Layer Perceptron »).

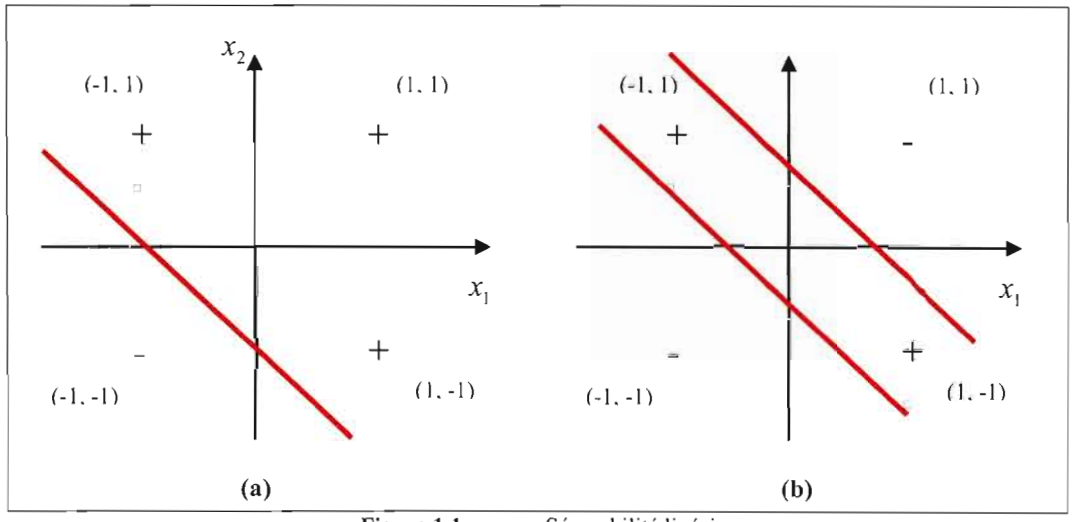


Figure 1.1 Séparabilité linéaire

(a) Fonction OR : Séparatrice linéaire $ax_1 + bx_2 = t$, le problème est "linéairement séparable";
(b) Fonction XOR : Aucune séparatrice linéaire, le problème est "non linéairement séparable".

1.3 Apprentissage des réseaux de neurones

L'apprentissage d'un réseau de neurones artificiel est réalisé par une procédure itérative d'ajustement de ses paramètres internes (les poids synaptiques). Cette procédure d'ajustement est décrite par un algorithme d'apprentissage. Un apprentissage peut se faire de différentes manières et selon différentes règles. Il existe trois types fondamentaux d'apprentissage :

1.3.1 Apprentissage supervisé

C'est actuellement le mode d'apprentissage le plus couramment utilisé. Son principe est élémentaire : on soumet au réseau un grand nombre d'exemples pour lesquels l'entrée et la sortie associée sont connues et les poids sont modifiés de façon à corriger l'erreur entre la sortie désirée et la réponse du réseau à l'entrée correspondante.

Le plus répandu des algorithmes d'apprentissage supervisé est l'algorithme de rétro-propagation d'erreur par descente de gradient qui utilise la dérivée de la fonction de transfert des neurones pour calculer l'erreur (Rumelhart, Hinton et Williams, 1986). L'avantage de cet algorithme c'est qu'il peut être utilisé à des réseaux multicouches. Ainsi, l'erreur est obtenue à la sortie puis propagé en amont d'une couche à l'autre jusqu'à la couche d'entrée.

Les paramètres d'un classificateur, dans le cas d'un apprentissage supervisé, sont estimés en minimisant une fonction de coût appropriée (erreur quadratique, coût de l'entropie croisée, etc.) sur l'ensemble d'apprentissage composé de vecteurs d'entrée/sortie ou (entrée; classe d'appartenance).

La performance d'un classificateur se mesure par la probabilité d'erreur de classement qu'il accomplit. Cette probabilité est estimée par la proportion d'objets mal classés sur un ensemble indépendant de celui d'apprentissage ou par validation croisée (Fig. 1.3).

1.3.2 Apprentissage non-supervisé et auto-organisation

Dans ce type d'apprentissage, uniquement l'information par rapport aux objets est connue (entrée); la sortie désirée est inconnue. L'apprentissage non-supervisé est utilisé pour regrouper ces objets dans des classes non définies à l'avance (catégorisation ou *clustering*). Le réseau doit donc déterminer lui-même ses sorties en fonction des similarités détectées entre les différentes entrées, par exemple en fonction d'une règle d'auto-organisation. La méthode la plus connue d'apprentissage non-supervisé est celle des cartes de Kohonen (Kohonen, 1989).

1.3.3 Apprentissage par renforcement

Ce type d'apprentissage est inspiré de travaux en psychologie empirique de Thorndike (1911) et qui se résume comme suit :

"Lorsqu'une action (décision) prise par le réseau engendre un indice de satisfaction positif, alors la tendance du réseau à prendre cette action doit être renforcée. Autrement, la tendance à prendre cette action doit être diminuée".

L'apprentissage par renforcement constitue un modèle intermédiaire entre l'apprentissage non supervisé et l'apprentissage supervisé. Dans de nombreux problèmes on ne dispose pas de l'information nécessaire à la construction d'une base d'apprentissage complète. On ne dispose souvent que d'une information qualitative, constituée de succès-échecs, permettant l'évaluation de la validité

de la réponse calculée, sans pour autant connaître la réponse la plus adaptée. De ce fait, on n'indique pas à l'algorithme d'apprentissage l'action à prendre comme dans le cas de l'apprentissage supervisé, il doit plutôt découvrir lesquelles des actions rapportent la meilleure récompense en les essayant.
Cette forme d'apprentissage (aussi qualifiée d'apprentissage par pénalité/récompense ou par essai/erreur, est de plus en plus utilisée, en particulier pour les applications qui posent généralement des problèmes de « Credit-Assignment » ou « Temporal-Credit-Assignment », et en dépit de sa simplicité, l'information obtenue ainsi par l'interaction avec l'environnement permet au réseau de générer ses propres comportements, ce qui peut mener à des tâches de classification relativement complexes.

1.4 Règles d'apprentissage

1.4.1 Règles de Hebb

Parmi l'ensemble des règles proposées de nos jours, l'apprentissage de type Hebbien (Hebb, 1949) est encore considéré comme étant celui qui est le plus plausible biologiquement. Cette règle stipule que si deux neurones sont actifs en même temps, alors le poids de la connexion qui les relie est augmenté. On évalue cette augmentation ω_{ij} à :

$$\omega_{ij} = \eta a_i a_j$$

Où η est une constante d'apprentissage, a_i la valeur d'activation du neurone i et a_j la valeur d'activation du neurone j. Il y a renforcement de la connexion entre i et j lorsque i et j sont activés, c.à.d. $a_i > 0$ et $a_j > 0$.

1.4.2 Règle du delta (Widrow-Hoff)

Cette règle d'apprentissage s'inscrit dans le paradigme d'apprentissage supervisé. Elle repose sur la minimisation d'une erreur quadratique par une procédure d'approximation.

1.4.3 Règle de descente du gradient

Cette règle, utilisée par les réseaux multicouches, consiste en une descente de gradient. Elle cherche à minimiser une fonction d'énergie, reliée à l'erreur entre la sortie désirée et la sortie obtenue, en suivant les lignes de plus grande pente. La rétro-propagation de base utilise le gradient de l'erreur globale (obtenue avec tous les exemplaires d'apprentissage), mais présente l'inconvénient de se faire piéger par le premier minima local rencontré.

1.4.4 Règle d'apprentissage par compétition

Cet apprentissage permet la spécialisation sélective des neurones pour apprendre les vecteurs stimuli. Dans sa forme extrême, un seul neurone par essai d'apprentissage, celui avec la plus grande activation, voit ses poids synaptiques mis à jour. Deux architectures majeures utilisent l'apprentissage compétitif, les réseaux de Kohonen et les réseaux ART (*Asaptive Resonnance Theory*). Les premiers mettent en œuvre une approche d'apprentissage qui préserve la topologie de l'espace des vecteurs stimuli; les seconds, permettent de reconnaître et d'apprendre des formes qui ne font pas partie de l'ensemble d'apprentissage initial.

1.5 Réseaux récurrents et réseaux non bouclés

Il existe essentiellement deux types de réseaux de neurones, les réseaux non bouclés, dits aussi à action directe (*feedforward neural networks*) et les réseaux récurrents ou dynamiques (*dynamic neural networks*).

1.5.1 Réseaux de neurones non bouclés

Les réseaux de neurones non bouclés sont les plus utilisés, aussi bien en modélisation qu'en classification. Dans ce type de réseau, la connectivité est restreinte car un neurone d'une couche inférieure est seulement relié à des neurones des couches suivantes. L'architecture la plus utilisée, spécialement en classification, est le Perceptron Multi-Couches (*multilayer perceptron ou MLP*), particulièrement la version à une seule couche cachée (voir figure 1.2).

a. Perceptron multicouches (MLP)

Un réseau de neurones MLP consiste en une couche d'entrée composée de neurones (unités), un nombre arbitraire de couches de traitement (couches cachées), et d'une couche de sortie (Fig. 1.2). Les réseaux MLP pour la classification comportent autant de neurones d'entrée que les patrons d'entrée ont de mesures. La couche de sortie comprend généralement autant de neurones que l'ensemble de données a des classes. La fonction de transfert d'un neurone (généralement une sigmoïde) transforme la somme de toutes les entrées pondérées en signal de sortie, qui servent comme entrées aux neurones de la couche suivante.

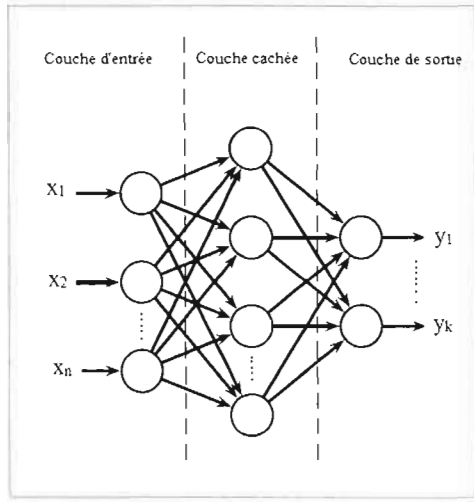


Figure 1.2 Architecture de base d'un réseau MLP

Fonction de sortie

Les réseaux multicouches apprennent généralement en utilisant la règle de rétro-propagation de l'erreur vers les couches internes, Cette règle nécessite l'utilisation d'une fonction de sortie dérivable. La fonction souvent utilisée est la sigmoïde qui est à la fois suffisamment discriminante pour pouvoir afficher les classes de sortie, et aussi différentiable en tout point afin de pouvoir appliquer la méthode de rétro-propagation d'erreur. Elle s'exprime comme suit .

$$y = f(a) = \frac{1}{1+e^{-\sigma a}} \quad \cdots(1.1)$$

Des manipulations simples permettent d'exprimer sa dérivée comme une fonction de la sortie seulement :

$$y' = f'(a) = \sigma y(1-y)$$

Dans le cas de patrons bipolaires, la fonction de sortie utilisée est souvent la tangente hyperbolique :

$$f(x) = \tanh\left(\frac{\sigma a}{2}\right) \quad \cdots(1.2)$$

Sa dérivée est alors :

$$y' = f'(a) = \frac{\sigma}{2}(1+y)(1-y)$$

Apprentissage des réseaux MLP

L'algorithme utilisé pour l'apprentissage du MLP est l'algorithme par rétro-propagation d'erreur, qui vise à minimiser la distance entre la sortie du réseau et la sortie désirée, et qui est typiquement mis en œuvre à l'aide d'une technique d'approximation par descente de gradient. L'algorithme modifie systématiquement les poids jusqu'à ce que la courbe d'erreur ne soit plus croissante.

Lors de l'apprentissage d'un MLP, il faut s'assurer de ne pas surentraîner le réseau qui devient alors performant seulement pour les patrons ayant servi à l'entrainer. En outre, pour vérifier la capacité de généralisation du réseau, on divise les données à apprendre en ensemble d'apprentissage et ensemble de test, et on estime aussi l'erreur de classification pour ce dernier (c.-à-d. pour des patrons ne faisant pas partie de l'ensemble d'apprentissage). Les méthodes couramment utilisées sont la validation croisée et la technique de "Leave-one-out".

Validation croisée

Il s'agit de scinder la base d'apprentissage en k groupes de taille approximativement égale, qui vont successivement servir d'ensemble de test. On réalise alors k apprentissages du modèle en laissant à chaque fois un des k groupes de côté pour le valider. On peut alors calculer une erreur de test pour chacun des groupes et en faire la moyenne, ce qui constitue l'estimateur de l'erreur de test par validation croisée (Fig. 1.3).

Leave-one-out

Cette méthode est un cas particulier de validation croisée dans lequel, chacun des ensembles d'apprentissage est constitué de l'ensemble complet des données dont on a retiré un seul individu.

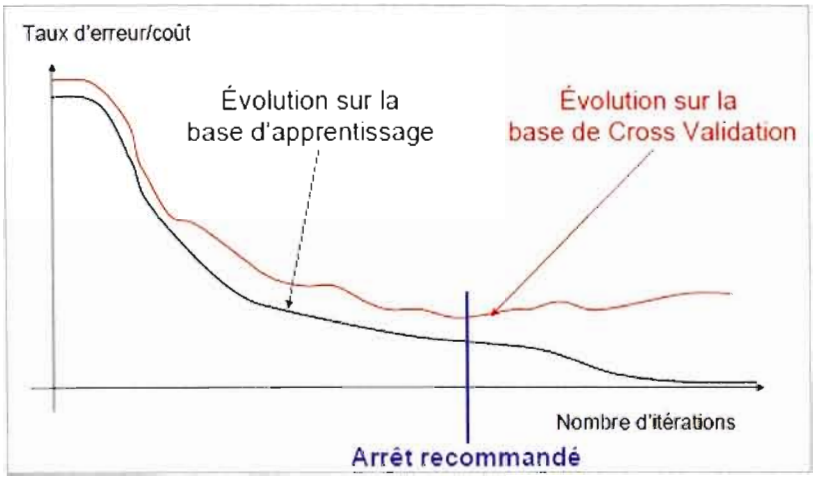


Figure 1.3 Technique de validation croisée

Algorithme de rétro-propagation

L'algorithme de rétro-propagation est exécuté selon les étapes suivantes:

1. Initialiser tous les poids du réseau à des petites valeurs aléatoires;
2. Normaliser les patrons d'apprentissage. Ex. Normalisation par rééchelonnement des données, de manière à ce que les valeurs soient comprises dans le même intervalle [-1, 1] : x[a, b] → [-1, 1] par la formule (2(x-a)/(b-a)) - 1;
3. À chaque itération (ou *epoch*), présenter les patrons de la base d'apprentissage à l'entrée du réseau, selon un tirage aléatoire et sans remise;
4. Pour chaque patron d'apprentissage x_j présenté :
   (a) Calculer la sortie correspondante;
   (b) Ajuster les poids du réseau en rétro-propageant l'erreur e_j observée entre le patron désiré x_j et la sortie correspondante, en appliquant la règle suivante:

$$\omega_{ji}(n) = \omega_{ji}(n-1) + \eta\delta_j(n)\, y_i(n) + \alpha\,\Delta\omega_{ji}(n-1) \qquad \cdots(1.3)$$

où :

$0 \leq \eta \leq 1$, représente le taux d'apprentissage;

$0 \leq \alpha \leq 1$, représente le *momentum* qui est comparable à une inertie dans le changement de poids;

δ_i, représente le gradient local défini par :

$$\delta_j(n) = \begin{cases} e_j(n)\, y_j(n)[1 - y_j(n)] & Si\ j \in couche\ de\ sortie \\ y_j(n)[1 - y_j(n)] \sum_k \delta_k(n)\, \omega_{kj}(n) & Si\ j \in couche\ cachée \end{cases}$$

5. Répéter les étapes 3 et 4 jusqu'à atteindre une certaine condition d'arrêt (seuil inférieur de l'erreur, nombre d'itérations maximal ou validation croisée).

Le MLP a été utilisé avec succès dans des domaines variés, mais il est peu populaire dans sa forme classique en science cognitive. Cela est dû partiellement à sa convergence trop lente et au fait qu'il est souvent piégé dans des minima locaux. Mais surtout, le MLP entrainé par algorithme de rétro-propagation d'erreur constitue un modèle non biologiquement plausible, car aucune évidence neurologique n'existe sur la façon dont l'erreur est propagée aux couches précédentes de traitement.

Par ailleurs, plusieurs améliorations ont été proposées afin de rendre le MLP plus robuste, dont des solutions heuristiques comme l'addition d'un paramètre *momentum*, ou l'utilisation d'une vitesse d'apprentissage adaptative. Néanmoins, son exploitation est toujours alourdie par l'incapacité présente de déterminer automatiquement certains de ses paramètres fondamentaux, comme la valeur initiale de l'ensemble des poids de connexion. Certaines techniques d'optimisation (algorithmes génétiques, recuit simulé) ont été envisagées, mais le prix à payer est l'augmentation de la complexité des calculs.

b. Réseau à fonctions radiales de base (RBF)

Les réseaux RBF (Poggio et Girosi, 1989) reposent sur le fait que toute fonction continue peut être approchée par une somme ou combinaison linéaire de fonctions radiales. Ce sont des réseaux possédant une couche cachée dans laquelle les neurones utilisent des fonctions à base radiale et une couche de sortie qui effectue une somme pondérée des réponses des neurones cachés (sortie linéaire) (Fig. 1.4). Divers types de fonctions peuvent être utilisés comme noyaux ou fonctions de base (ex. gaussienne, thin-plate-spline, multi-quadratique, etc.), la fonction gaussienne reste cependant la plus utilisée.

À l'instar des MLP, ces réseaux sont aussi des approximateurs universels (Park et Sandberg, 1991) et sont très efficaces pour des tâches de classification et de régression. Un de leurs avantages est leur comportement local très intuitif. Leur utilisation requiert un choix de paramètres. Parmi eux, le nombre de neurones cachés ainsi que la moyenne et l'écart-type.

Une fonction radiale de base (RBF) est une fonction symétrique autour d'un centre μ_i :

$$\phi_i(x) = \phi\left(\|x - \mu_i\|, \sigma_i\right)$$

$\|\cdot\|$, représente la norme euclidienne;

σ_i, représente la largeur de la fonction.

La sortie d'un réseau RBF calcule une combinaison linéaire des fonctions radiales de la couche cachée. Dans le cas de fonctions gaussiennes, la sortie est définie par:

$$y(x) = \sum_{i=1}^{N} w_i \exp\left(-\frac{\|x - \mu_i\|^2}{2\sigma_i^2}\right) \quad \cdots(1.4)$$

L'utilisation d'un modèle RBF nécessite la détermination de son architecture, c.à.d. le nombre N de fonctions radiales ϕ_i (Fig. 1.4), la fixation des paramètres de ces fonctions, μ_i et σ_i, et la détermination des poids des connexions w_i vers la couche de sortie.

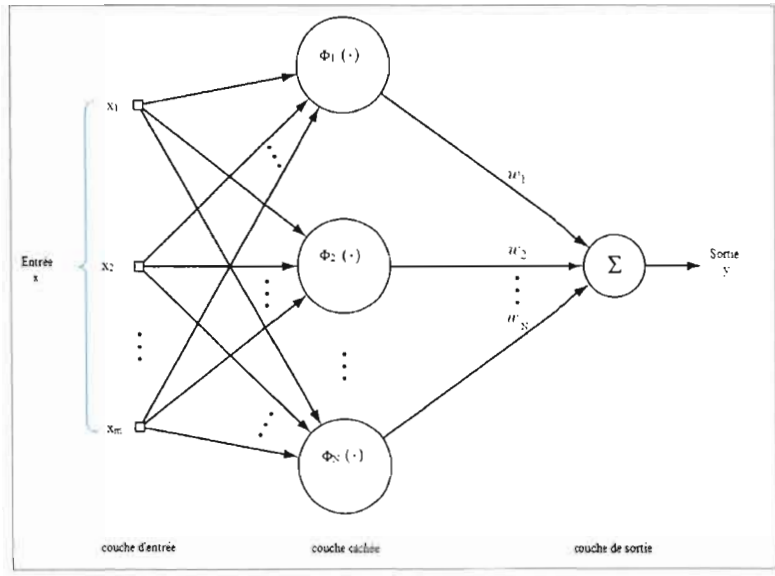


Figure 1.4 Architecture d'un réseau RBF

Estimation des centres

L'approche la plus simple consiste à effectuer un choix aléatoire à partir des patrons d'entrainement :

$$c_i = x_{\alpha_i}, \qquad i = 1, \cdots, N$$

Tels que :

$\alpha_i \in \{1, \cdots, P\}$;

N, représente le nombre de centres et est généralement déterminé par procédé essais/erreurs.

Cette approche est moins utilisée lorsqu'on dispose d'un nombre significatif de patrons. Par ailleurs, elle est justifiée seulement lorsque les patrons d'apprentissage sont distribués de façon homogène pour représenter le problème. En général, une approche plus efficace consiste à employer un algorithme de catégorisation, comme l'algorithme des k-moyennes, afin de garantir une meilleure répartition des centres.

Détermination des largeurs

Pour déterminer les valeurs des écarts-type (paramètres σ_i), on utilise en général une heuristique basée soit sur la distance de chaque centre par rapport à son voisin, soit sur le calcul de la variance de

l'ensemble d'exemples rattachés à un centre (ceux pour lesquels ce centre est le plus proche) (Lacerda et al., 2000) :

$$\sigma = \left\langle \left\| c_i - c_j \right\| \right\rangle \quad \cdots(1.5)$$

c_j : représente le centre le plus proche de c_i

$\langle\ \rangle$: indique la moyenne calculée sur chaque prototype

Une autre approche efficace consiste à utiliser un algorithme de descentes de gradient pour la détermination des centres et de la largeur de chacune des gaussiennes.

Apprentissage des réseaux RBF

Les poids optimaux d'un réseau RBF se calculent en trouvant la solution de l'équation :

$$y(x) = \sum_{i=1}^{N} w_i \phi\left(\left\| x - \mu_i \right\|, \sigma_i\right) = \sum_{i=1}^{N} w_i z_i(x)$$

On recherche la solution w qui minimise l'erreur entre la sortie du réseau RBF y et la sortie désirée t. On a donc le système d'équations linéaires suivant :

$$t = Z\,w + e$$

La matrice Z, de taille $P \times N$, donne les réponses des N centres RBF sur les P prototypes d'apprentissage, e représente le vecteur d'erreur.

Le critère à optimiser est :

$$E = e^T e$$

La solution s'obtient par un calcul de pseudo-inverse, et s'exprime comme suit:

$$w = \left(Z^T Z\right)^{-1} Z^T y$$

Le problème majeur que rencontrent les modèles RBF est lié à leur comportement lorsque la dimension de l'espace d'entrée augmente. Ce qui peut non seulement affecter le temps de calcul mais augmente aussi le nombre de patrons requis pour l'estimation correcte des paramètres.

1.5.2 **Réseaux de neurones récurrents** (dynamiques)

Un réseau de neurones récurrent, ou dynamique (*feedback neural network*) est décrit par des équations aux différences portant sur l'évolution du système à des instants successifs. L'un des premiers réseaux récurrents proposés est le réseau de Hopfield (Hopfield, 1982), qui réalise une fonction de mémoire auto-associative.

Réseau de Hopfield

Hopfield (1982) a montré que la dynamique d'un système est dominée par un nombre d'états stables correspondant à des états d'énergie minimales et vers lesquels il est attiré, appelés points fixes attracteurs. De ce fait, on peut considérer le système comme une mémoire pouvant être adressée par son contenu (celui des états de convergence stables). Si, nous pouvions choisir un ensemble quelconque d'états et en faire des points fixes attracteurs, alors le système devient un dispositif de mémoire.

Du point de vue d'un système dynamique, on peut percevoir, dans une information incomplète ou altérée par du bruit, un point instable dans l'espace d'états. On retrouve l'information entière une fois que le système ait convergé vers un point fixe attracteur et ce, à partir de n'importe où à l'intérieur des régions qui l'entourent. Qualitativement parlant, des états de haute énergie peuvent être instables et avoir tendance à se diriger vers des états adjacents de basse énergie, jusqu'à ce que le système atteigne un état stable d'énergie minimale.

Dans le modèle de Hopfield, il n'y a pas de distinction entre neurone d'entrée et neurone de sortie (Fig. 1.5). Les neurones sont en effet reliés deux à deux par des connexions symétriques.

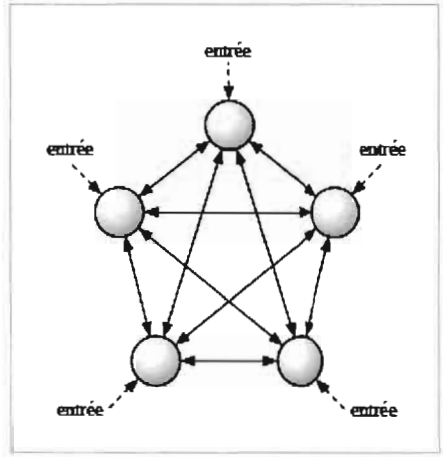


Figure 1.5 Réseau de Hopfield

Apprentissage

L'apprentissage dans le modèle de Hopfield se fait hors-ligne (apprentissage statique) et utilise la règle de Hebb. La matrice des poids de connexions représente la corrélation entre les entrées et elles-mêmes:

$$W = \frac{1}{N}\left(XX^T - I\right)$$

X, représente la matrice des patrons d'entrées;

N, est le nombre de patrons et I la matrice identité;

Rappel

Le rappel dans un réseau Hopfield utilise une fonction de type *signum*, définie, dans le cas de patrons bipolaires, comme suit :

$$signum(x) = \begin{cases} 1 \; si \; x > 0 \\ 0 \; si \; x = 0 \\ -1 \; si \; x < 0 \end{cases}$$

On a :

$$x(t+1) = sgn\left(W\,x(t)\right)$$

L'utilisation du modèle de Hopfield est limitée par de nombreuses imperfections. En effet, la loi de Hebb entraîne des contraintes d'orthogonalité entre les différents motifs appris, sous peine de créer des minima locaux ne correspondant pas aux états appris. D'autre part, ses capacités de stockage sont limitées à $0.14n$ patrons (Amit et al., 1987), où n représente le nombre de neurones du réseau. De plus, ce réseau permet de traiter les stimuli corrélés à la condition que l'apprentissage soit unique, c'est-à-dire que chaque patron n'est vu qu'une seule fois. En effet, la règle de hebb étant strictement additive, la présentation répétée des mêmes stimuli d'apprentissage risque de provoquer une explosion des poids de connexion. Par conséquent, l'apprentissage est effectué hors-ligne, une solution peu plausible biologiquement. Le modèle de Hopfield reste donc limité à des problèmes restreints en nombre de contraintes et en nombre de motifs à mémoriser. Néanmoins, Le réseau de Hopfield demeure le premier modèle neuronal proposé pour résoudre des problèmes d'optimisation, en plus de ceux de classification.

1.6 Mémoire associative bidirectionnelle (BAM)

La mémoire associative bidirectionnelle BAM (Kosko, 1988) est une généralisation du modèle de Hopfield. C'est une mémoire hétéro-associative, qui permet d'associer deux informations de natures différentes. Le réseau est composé de deux couches de neurones pouvant être de taille distincte, totalement interconnectées, de façon bidirectionnelle (Fig. 1.6 (a)). L'information est réverbérée entre les deux couches de neurones, jusqu'à atteindre un état d'équilibre. Cet état d'équilibre constitue la mémoire restaurée du réseau, laquelle en règle générale, correspond au patron qui ressemble le plus au patron appris (attracteur).

Cependant, la nature non déterministe du réseau peut parfois l'amener à converger vers un faux attracteur qui ne figure pas dans l'ensemble des patrons appris ou bien à montrer un comportement chaotique et continuer d'errer dans une petite région dans l'espace d'états.

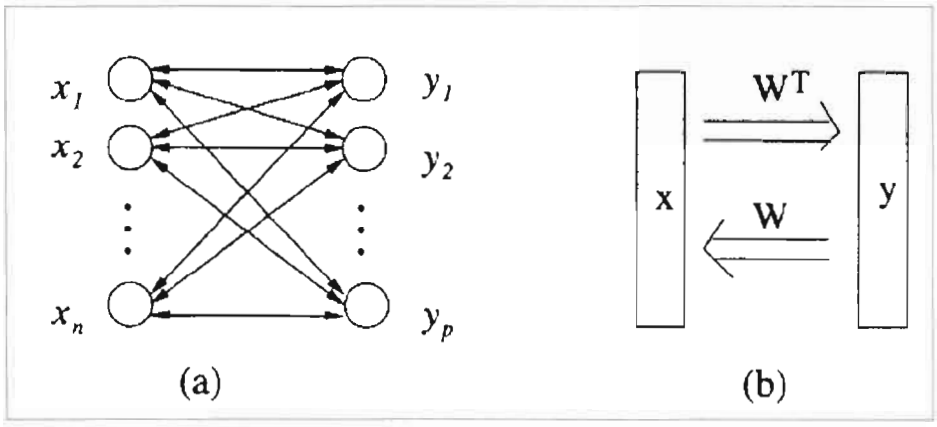


Figure 1.6 (a) Modèle BAM;
(b) Flux de signal entre la couche X et la couche Y.

Apprentissage

De même que dans le modèle de Hopfield, l'apprentissage dans le modèle BAM-Kosko utilise une règle hors-ligne strictement hebbienne. La matrice de poids est calculée comme suit :

$$W = YX^T \quad \cdots(1.6)$$

Où :

- X représente la matrice des vecteurs d'entée et Y celle des sorties désirées correspondantes. Le rapport d'entrée-sortie est illustré sur la figure 1.6 (b).
- W représente la matrice de poids qui se veut une mesure de corrélation entre l'entrée et la sortie du réseau. Si l'on procède dans le sens inverse de W, on obtient la matrice transposée W^T.

Rappel

À l'instar du modèle de Hopfield, la BAM de Kosko utilise une fonction non-linéaire de type *signum* pour effectuer le rappel à partir de patrons bruités. Le rappel dans les réseaux de type BAM suit les équations suivantes :

$$\begin{cases} y(t+1) = \mathrm{sgn}(W\,x(t)) \\ x(t+1) = \mathrm{sgn}(W^T\,y(t)) \end{cases}$$

Dans ces équations, y(t) et x(t) représentent la paire de vecteurs choisis lors de l'itération *t* et *sgn* est la fonction *signum*.

La BAM de Kosko, malgré son élégance, présente plusieurs défauts. Elle est incapable de traiter des patrons en tons de gris et peut seulement traiter des stimuli binaires ou bipolaires. De plus, sa règle d'apprentissage étant strictement hebbienne, l'apprentissage de stimuli corrélés produit beaucoup d'états (attracteurs) indésirables, ce qui réduit sa performance de rappel. Aussi, sa capacité de stockage est limitée. Enfin, la BAM-Kosko est incapable de mémoriser des patrons non linéairement séparables.
Au cours des dernières années, plusieurs améliorations de ce modèle ont été proposées (Shi, Zhao et Zhuang, 1998; Xu, Leung et He, 1994). La plupart visaient à augmenter la capacité de stockage, réduire la quantité de faux attracteurs, et améliorer la performance de rappel. Toutefois, ces méthodes sont associées à une complexification de l'architecture ou de la procédure d'apprentissage.

Chartier et Boukadoum (2006) ont proposé un modèle de mémoire hétéro-associative BAM à fonction de sortie chaotique, architecture asymétrique contenant deux matrices de poids de connexions distinctes dans les deux directions, et règle d'apprentissage hebbienne modifiée avec apprentissage en ligne. Ce modèle offre une performance de rappel supérieure aux autres modèles BAM et réduit le nombre de faux attracteurs. De plus, il peut traiter autant des patrons binaires qu'en tons de gris. Plus récemment, les mêmes auteurs ont proposé une modification au niveau de la fonction de sortie de la BAM, en y introduisant un paramètre d'asymétrie durant la phase de rappel dans le but de lui permettre d'apprendre des patrons non linéairement séparables (Chartier, Boukadoum et Amari, 2009).

1.7 Objectifs et structure du mémoire

Dans ce qui précède, nous avons brièvement décrit les modèles classiques de réseaux de neurones pour la classification et la reconnaissance, ainsi que leurs caractéristiques fondamentales, notamment leurs architectures, leurs règles d'apprentissage et leurs fonctions de sortie. Nous avons aussi abordé leurs limitations, particulièrement en ce qui a trait aux mémoires BAM.

Nous reprenons dans la suite du mémoire le modèle de BAM proposé par Chartier, Boukadoum et Amiri et approfondissons son étude à travers des modifications architecturales et des simulations, en vue d'en améliorer la performance de rappel et augmenter la capacité de stockage. Nous étudions en particulier l'impact de modifier la fonction de sortie, en lui ajoutant un paramètre d'asymétrie, sur la capacité du réseau à traiter des tâches de classification non linéairement séparables. Nous nous inspirons de la théorie des catastrophes pour le cadre théorique de notre étude.

Nous expérimentons sur le modèle en vue d'améliorer sa performance de classification sans complexifier son architecture ou nous écarter de la plausibilité biologique de la règle d'apprentissage. Pour ce faire, nous faisons appel à des algorithmes d'optimisation stochastiques, inspirés de la théorie naturelle de l'évolution, afin de concevoir des modèles de classification puissants, capables de reproduire les processus cognitifs et le plus fidèlement possible. Ces algorithmes sont utilisés pour fin d'optimisation du paramètre d'asymétrie de la fonction de sortie de la BAM ainsi que pour la détermination de l'ensemble des poids de connexion des architectures multicouches, qui seront utilisées dans les modèles hybrides. Parmi ces algorithmes, les colonies de fourmis, l'optimisation par essaims particulaires et les algorithmes génétiques. Ces algorithmes sont connus pour être robustes, car moins susceptibles d'être piégés par des optima locaux. De plus, un mauvais choix des paramètres de départ n'empêche pas leur convergence vers des solutions optimales, contrairement aux approches d'optimisation classiques (ex. rétro-propagation d'erreur).

Dans ce qui suit, nous décrivons dans le prochain chapitre les principes généraux des algorithmes évolutionnaires pour l'optimisation de problèmes combinatoires et continus. On se sert de ces algorithmes au cours de nos simulations d'une part pour l'apprentissage des modèles classiques, RBF et MLP dans les architectures hybrides et d'autre part, pour la recherche du paramètre asymétrique optimal pour la fonction de sortie du modèle BAM. Dans le chapitre 3, on présente dans un premier temps quelques notions sur les systèmes dynamiques et par la suite, on présente la théorie des catastrophes en montrant différentes façons de provoquer des bifurcations dans une dynamique au moyen des types élémentaires de cette théorie. On présente à la fin du chapitre 3, le modèle BAM utilisé en décrivant ses caractéristiques, entre autres son architecture, sa fonction de sortie ainsi que les règle d'apprentissage et de transmission. On a vu dans le précédent chapitre que les réseaux RBF utilisent des fonctions radiales de base permettant d'effectuer des approximations locales de projections non-linéaires contrairement aux modèles MLP qui eux, construisent des approximations globales de ces mêmes projections. On expérimente dans le chapitre 4 chacun de ces deux modèles placés dans des architectures hybrides parallèlement à une BAM. Le but serait d'observer leurs contributions dans l'amélioration de la performance de classification en présence d'une dynamique de catastrophe fronce.

Chapitre 2

ALGORITHMES ÉVOLUTIONNAIRES

2.1 Introduction

Les algorithmes évolutionnaires (AE) représentent une classe de méthodes d'optimisation par recherche probabiliste de solution basée sur le modèle de l'évolution naturelle. Ils s'inspirent de la théorie naturelle de l'évolution des espèces animales, en vue de s'adapter à leurs environnements. Leur avantage principal est leur capacité à explorer très largement l'ensemble des solutions possibles dans l'espace de recherche. En effet, ils sont capables de localiser rapidement des solutions sous-optimales, lorsque l'espace de recherche possède une taille et une complexité suffisamment importantes pour détourner toute garantie d'optimalité (De Jong, 1988). De plus, ces algorithmes se font moins piéger par des optima locaux que les algorithmes d'optimisation classiques comme la descente de gradient, ou même le recuit simulé, ont du mal à contourner. Finalement, ces méthodes offrent de bonnes performances pour des problèmes NP-complets qui sont généralement hors d'atteinte de méthodes déterministes plus classiques.

Plusieurs travaux ont montré l'intérêt d'appliquer les algorithmes évolutionnaires sur des réseaux de neurones (Belew et al., 1990; Whitley, 1995), et particulièrement lorsqu'ils sont associés avec un apprentissage de type descente de gradient. Avec les réseaux connexionnistes classiques, les problèmes sont souvent résolus par une méthode de type recherche locale. C'est le cas, par exemple, de l'apprentissage d'un réseau de neurones par rétro-propagation d'erreur ; partant d'une configuration de poids initiale, la méthode va chercher la meilleure solution dans le voisinage de cette configuration. Cependant, cette solution optimale à l'échelle locale peut ne pas l'être globalement, car il est possible qu'une meilleure solution existe qui ne soit pas dans le voisinage de la configuration initiale considérée. Les AE peuvent remédier à ce problème en procurant au réseau un ensemble adéquat de poids initiaux, ce qui permet non seulement une convergence rapide du réseau, mais aussi lui évite les pièges des optima locaux (fig. 2.1).

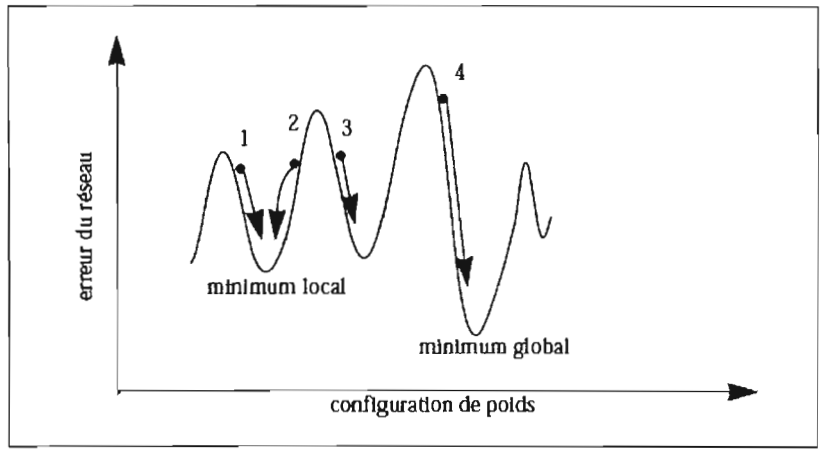


Figure 2.1 Convergence d'un réseau; optima local et optima global; Convergence locale d'un réseau de neurones (config. 1, 2 et 3); Convergence globale d'un réseau de neurones (config. 4).

2.2 Principe généraux des algorithmes évolutionnaires (AE)

Les algorithmes évolutionnaires modélisent une population d'individus par des points dans l'espace de recherche. Chaque individu possède une performance relative au problème, appelée *fitness*, qui représente son degré d'adaptabilité au sein de la population. L'objectif des algorithmes évolutionnaires est de faire évoluer cette population d'individus par générations successives afin de trouver la solution optimale.

Les AE sont capables d'éviter la convergence des systèmes vers des optima locaux, aussi bien lorsqu'ils sont combinés avec des méthodes de recherche locales comme la rétro-propagation du gradient (Belew et al., 1990) que lorsqu'ils sont seuls (Goldberg, 1989).

2.3 Algorithmes génétiques (AG)

Développés dans les années 70 par Holland (Holland, 1975) puis approfondis par Goldberg (Goldberg, 1989), les algorithmes génétiques (AG) s'inspirent de l'évolution darwinienne des populations biologiques par sélection naturelle.

Les AG manipulent des populations (ou chromosomes) qui représentent des points de l'espace de recherche et les font évoluer au moyen d'opérations stochastiques comme la sélection, le croisement et la mutation.

Représentation des individus (chromosomes)

Dans un algorithme génétique (AG), un individu est codé dans un génotype composé de plusieurs gènes, correspondant aux valeurs des paramètres du problème à traiter. Le génotype d'un individu correspond à une solution potentielle au problème posé. La forme codée d'une solution est donc une chaîne qu'on appellera chromosome. Il existe principalement deux types de codage utilisés dans les algorithmes génétiques.

- Codage binaire : Chaque chromosome est représenté par une série de bits {0, 1}
- Codage par valeur : Ce type de codage peut être employé dans les problèmes où l'utilisation du codage binaire serait compliquée. Dans le codage par valeur, chaque chromosome est une séquence de valeurs (gènes). Ces valeurs sont liées au type de problème. Ils peuvent être des nombres (entiers ou réels), des caractères ou tout autre objet. Le codage réel est utilisé notamment dans le cas où l'on recherche un optimum d'une fonction réelle. Chaque chromosome est dans ce cas représenté par une série de valeurs réelles.

2.3.1 Principe de l'algorithme génétique (AG)

Les AG utilisent des opérateurs de recherche tels que : l'évaluation, la sélection, le croisement et la mutation. Ces opérateurs permettent de polariser l'espace de recherche dans les régions de solutions prometteuses. L'algorithme génétique débute avec une population initiale, composée de plusieurs individus générés aléatoirement. À chaque itération, un opérateur d'évaluation est appliqué sur ces individus afin d'évaluer leur niveau d'adaptation ou leur *fitness*. L'étape de sélection vient par la suite

choisir les individus ayant les meilleurs niveaux d'adaptation de sorte qu'ils puissent être assignés à une probabilité plus élevée pour la survie.

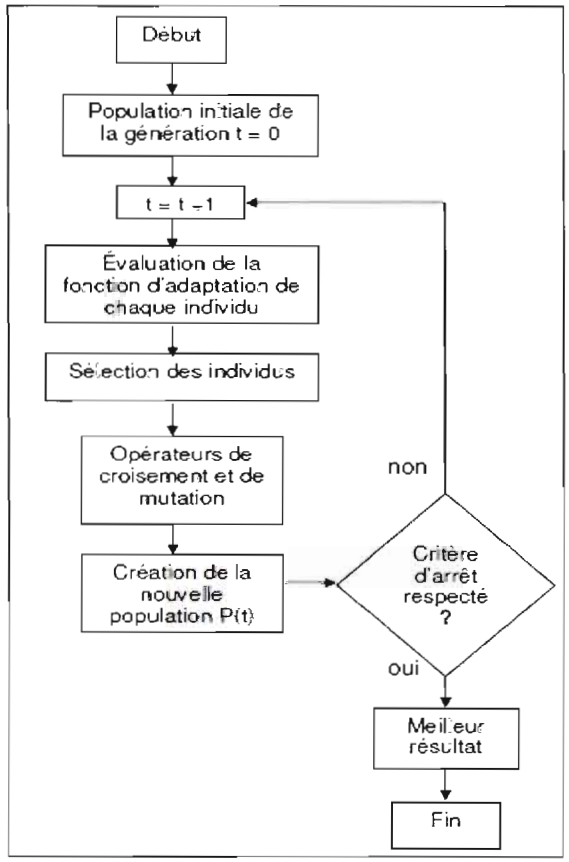


Figure 2.2 Organigramme d'un AG standard

Évolution

L'évolution de la population se fait à l'aide d'opérateurs de sélection, de recombinaison et de mutation. (Goldberg, 1989)

Sélection

La sélection a pour objectif d'identifier les individus qui doivent se reproduire. Dans sa version classique, elle consiste à sélectionner les individus proportionnellement à leur niveau d'adaptation. La probabilité de survie d'un individu est directement liée à son efficacité relative au sein de la population. Un individu ayant une forte valeur d'adaptation a plus de chances d'être sélectionné et de contribuer à la génération suivante qu'un individu mal adapté à l'environnement (Goldberg, 1989).

L'étape de sélection, ne créé pas de nouveaux individus dans la population. Il existe plusieurs types de sélection (Thierens et Goldberg, 1994), les plus communément utilisés sont la sélection par roulette biaisée (*roulette wheel*), la sélection par classement (*ranking*), la sélection élitiste et la sélection par tournoi.

• La méthode "roulette biaisée" (*roulette wheel*) ou "loterie biaisée" : Chaque individu a une chance d'être sélectionné proportionnelle à sa performance (*fitness*). Si on imagine une roulette où tous les chromosomes de la population sont placés, alors la taille des sections dans la roulette sera proportionnelle à la valeur de la fonction fitness de chaque chromosome. Plus la valeur est grande, plus les chances de sélection sont grandes. Dans une population contenant P individus, la probabilité de sélection d'un individu i s'écrit :

$$\text{Prob}(i) = \frac{Fitness(i)}{\sum_{j=1}^{P} Fitness(j)} \quad \cdots(2.1)$$

• La sélection par classement (rang) : La sélection par roulette biaisée peut causer des problèmes lorsqu'il y a de grandes différences entre les valeurs de fitness. Par exemple, si la meilleure valeur de fitness d'un chromosome est de 90% de la somme de toutes les valeurs de fitness de la population, les autres chromosomes ont très peu de chances d'être choisis. La sélection par rang classe d'abord la population et ensuite assigne à chaque chromosome la valeur de fitness déterminée par son rang.
Le plus mauvais aura la performance 1, le deuxièmes plus mauvais 2 etc. et le meilleur chromosome aura la performance N; N étant le nombre de chromosomes dans la population.
Avec cette méthode de sélection, les chromosomes de faible rang ont de meilleures chances d'être sélectionnés qu'avec la méthode de la roulette biaisée, toutefois elle peut mener à une convergence plus lente étant donné que les meilleurs chromosomes ne différent plus tellement des autres.

• La méthode "élitiste": En créant une nouvelle population par la méthode de croisement et de mutation, nous avons de grandes chances de perdre les meilleurs chromosomes. L'élitisme permet de copier le chromosome le mieux adapté (ou quelques meilleurs chromosomes) à la nouvelle population. L'élitisme peut rapidement augmenter la performance de l'AG car il empêche la perte des meilleures solutions.

• La sélection par tournoi : Elle consiste à choisir itérativement des individus parmi un sous ensemble de la population et les faire participer à un tournoi. Le chromosome qui possède la plus grande valeur de fitness l'emporte. Le type le plus commun de sélection par tournoi est celui où seulement deux individus sont choisis.

Goldberg et Deb ont démontré que les sélections par tournoi et par classement maintiennent une croissance potentielle de la fonction fitness de la population. La sélection par tournois requière cependant des calculs moindres que la sélection par classement.

Opérateur de croisement

Après l'étape de sélection, des opérateurs de croisement et de mutation sont appliqués avec certaines probabilités de croisement P_c et de mutation P_m.

Le croisement a pour but d'enrichir la diversité de la population en manipulant les structures de chromosomes. Il s'effectue en deux étapes. D'abord les éléments produits par la sélection (parents) sont appariés, ensuite chaque paire de chaînes subit un croisement et génèrent deux enfants.

Plusieurs types de croisement peuvent être utilisés, dont les principaux sont :

- Croisement à un point (*One-point crossover*) : On sélectionne aléatoirement une position et on permute les parties. (Fig. 2.3)

- Croisement à n points : Cela consiste à sélectionner aléatoirement plusieurs positions et à permuter les différentes parties. Ex. croisement en deux-points (fig. 2.4)

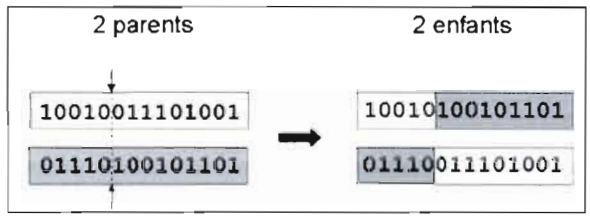


Figure 2.3 Croisement à un point

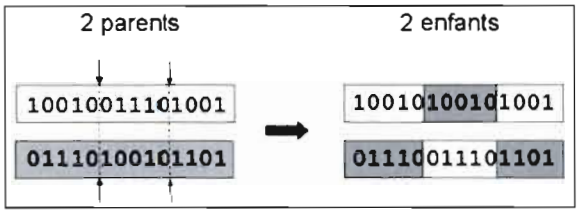


Figure 2.4 Croisement à deux points

- Croisement uniforme: Dans ce type de croisement, on utilise un masque qui consiste en un vecteur généré aléatoirement, de longueur identique aux chromosomes parents, et composé de 0 et 1. Lorsque le bit du masque vaut 0, l'enfant hérite le bit du premier parent, sinon il hérite de celui du second parent. Le second enfant est le complémentaire du premier.

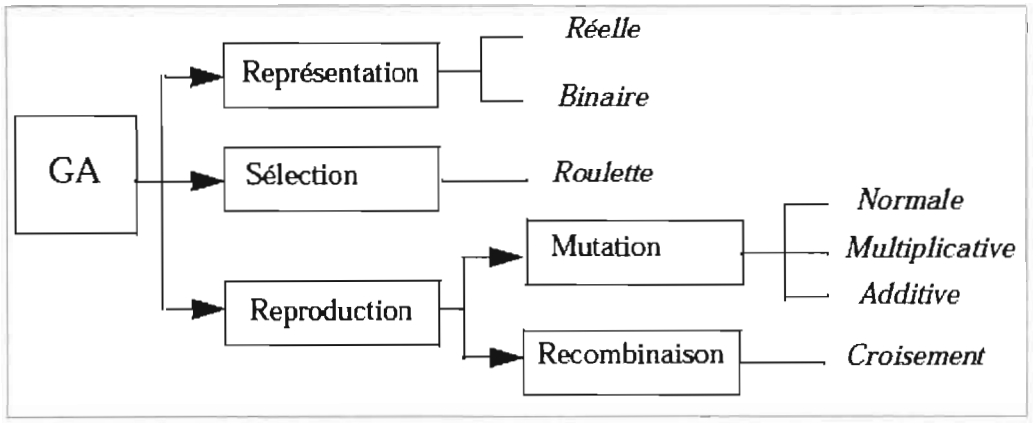


Figure 2.5 Représentation schématique d'un algorithme génétique classique

Opérateur de mutation

L'opérateur de mutation consiste à prendre des gènes aléatoires et à les altérer selon leur type. Dans le cas du codage binaire, cela revient à changer un 1 en 0 et vice-versa. L'opérateur de mutation introduit de la diversité dans le processus de recherche des solutions et permet d'éviter une convergence prématurée vers des optima locaux.

Généralement, le nombre d'individus survivants dans une population est choisi suivant la formule : (**taille de population** * (**1** - P_c)). Les individus écartés de la population sont alors remplacés par les nouveaux individus issus des processus de croisement et de mutation.

2.4 Algorithmes de colonies de fourmis (ACO)

Les algorithmes ACO font partie de l'intelligence d'essaim (*Swarm intelligence*) qui étudie l'utilisation de certaines propriétés de groupes sociaux pour des tâches telles que l'optimisation. La source d'inspiration de l'ACO est le comportement de recherche de vraies colonies de fourmis, qui permet de trouver le chemin le plus court entre leur nid et une source de nourriture. La capacité des vraies colonies de fourmis à trouver le plus court chemin est exploitée dans ces algorithmes afin de résoudre des problèmes d'optimisation lorsqu'ils sont codés sous forme de graphes. L'idée principale de l'ACO est la coopération indirecte entre un certain nombre de fourmis artificielles par l'intermédiaire de traînées de phéromones étendues sur le chemin. Chaque fourmi contribue avec un peu d'effort à la solution. Le résultat final émerge des interactions entre les fourmis de la colonie.

L'optimisation par colonies de fourmis (ACO) est une forme d'optimisation stochastique qui, à l'origine, avait été présentée pour résoudre des problèmes d'optimisation combinatoires (Dorigo et al, 1990). Elle a été appliquée avec succès à un large ensemble de problèmes d'optimisation combinatoires, tels que le problème de voyageur de commerce, le problème de l'assignation quadratique, le routage de véhicules et le routage de messages dans les réseaux de télécommunications. Des recherches ont par la suite menées au développement d'algorithmes ACO pour des problèmes d'optimisation continus (Blum et Socha, 2005).

Optimisation ACO combinatoire

2.4.1 Binary Ant System (BAS)

BAS est un algorithme d'optimisation ACO appliqué dans un espace binaire, dans lequel les fourmis artificielles construisent des solutions en se déplaçant sur un chemin dans un espace à n+1 dimensions (fig. 2.6). À chaque itération, un certain nombre de fourmis n coopèrent pour effectuer une recherche dans l'espace binaire de solutions. Chaque fourmi construit sa solution an avançant séquentiellement du nœud 1 au nœud n+1 sur le graphique de progression.

À chaque nœud i, la fourmi choisit le chemin supérieur i_0 ou le chemin inférieur i_1 pour avancer vers le nœud i+1. La sélection de i_0, implique $x_i = 0$; et la sélection de i_1, implique $x_i = 1$. La probabilité de sélection dépend des phéromones distribuées sur le chemin :

$$p_{is}(t) = \tau_{is}(t); \qquad i = 1, \cdots, n; \quad s \in \{0,1\}$$

Où t représente le nombre d'itérations.

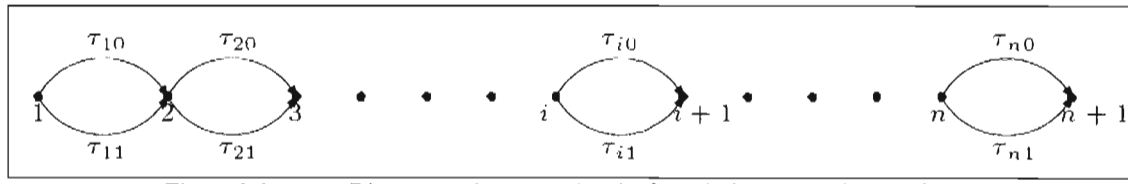


Figure 2.6 Diagramme de progression des fourmis dans un système BAS (Kong et Tian, 2006)

Probabilité de mouvement d'une fourmi

La probabilité de mouvement d'une fourmi k est l'opération principale d'un algorithme ACO. Dans un système BAS, cette probabilité s'exprimée comme suit:

$$P_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha}}{\sum_{k}\left[\tau_{ik}(t)\right]^{\alpha}} \quad \cdots(2.2)$$

Où τ_{ij}, représente l'intensité des phéromones entre le nœud i et le nœud j;

α, est un paramètre représentant l'importance de la traînée de phéromones.

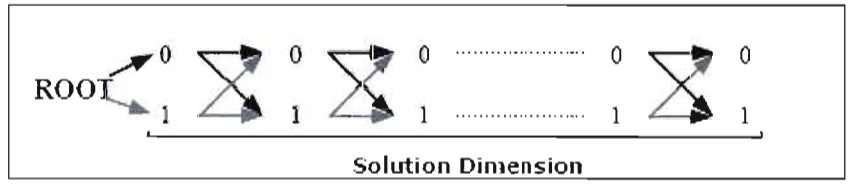


Figure 2.7 Recherche du chemin dans un espace binaire

Création de la population initiale

Initialement, la colonie de fourmis est aléatoirement distribuée dans l'espace binaire de solutions.

Mise à jour des phéromones (Kong et Tian, 2006)

Initialement, BAS fixe l'intensité de toutes les phéromones à une valeur initiale, $\tau_{is}(0) = \tau_0$. La procédure de mise à jour des phéromones est la suivante :

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \sum_{k} \Delta\tau_{ij}^{k} \quad \cdots(2.4)$$

Tels que :

ρ, représente le paramètre d'évaporation des phéromones, qui peut être fixé initialement $\rho = \rho_0$ (ex. $\rho_0 = 0.5$).

$\sum_{k} \Delta\tau_{ij}^{k}$, est la somme des quantités de phéromones résiduelles déposées par chacune des fourmis à la fin d'une itération. Elle est exprimée comme suit:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q & \text{si le chemin } ij \text{ est utilisé par la fourmi } k \text{ à l'itération } t \\ 0 & \text{sinon} \end{cases} \quad \cdots(2.5)$$

où Q est un paramètre libre de l'algorithme (généralement égal à 1).

2.4.2 MAX-MIN Ant System (MMAS)

Contrairement à l'algorithme BAS, l'algorithme MMAS (*MAX-MIN Ant System*) réalise une forte exploitation de l'historique de recherche en permettant seulement aux meilleures solutions d'ajouter des phéromones pendant la mise à jour des traînées de phéromones (Stützle et Hoos, 2000). L'utilisation d'un simple mécanisme pour limiter l'intensité des traînées de phéromones peut en effet éviter la convergence prématurée de la recherche.

Mise à jour de la traînée de phéromones

Dans MMAS, seule la fourmi offrant la meilleure solution est employée pour mettre à jour la traînée de phéromones (Stützle et Hoos, 2000). Par conséquent, la règle de mise à jour de la traînée de phéromones est modifiée de la manière suivante :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{best} \quad \cdots(2.6)$$

Où

$$\Delta\tau_{ij}^{best} = Q.f(s^{best}) \quad \cdots(2.7)$$

Telle que $f(s^{best})$, représente la fonction fitness reliée à la meilleure solution trouvée. La fourmi choisie représente soit la meilleure solution de l'itération courante S^{ib} (*iteration best*) ou bien la meilleure solution globale S^{gb} *(global best)* depuis la première itération de l'algorithme.

Le choix entre la solution meilleure-itération S^{ib} et meilleure-global S^{gb} pour la mise à jour des traînées de phéromones commandes la façon dont l'historique de recherche est exploité. En effet, en utilisant seulement S^{gb}, la recherche peut rapidement se concentrer autour de cette solution, et l'exploration des meilleures solutions devient rapidement limitée avec le risque de converger vers des optima locaux. Ce risque est réduit lorsque la meilleure solution S^{ib} est choisie pour la mise à jour des traînées de phéromones, puisque celles-ci peuvent différer considérablement d'une itération à une autre, ce qui permet de renforcer occasionnellement un plus grand nombre d'éléments.

L'algorithme MMAS étant déjà plus performant que le BAS, on peut encore l'améliorer en y introduisant une méthode de recherche locale.

2.4.3 Recherche locale (*Local Search*)

Afin d'optimiser la convergence des algorithmes ACO pour les problèmes combinatoires, on peut utiliser une méthode de recherche locale pour aider l'algorithme à rebondir des optima locaux (Kong et Tian, 2006). Son principe consiste à modifier les meilleures solutions trouvées (locale et globale) à la fin de chaque itération, avec des opérations de renversement de bits et de voir si les solutions obtenues sont meilleures que les originales. Si elles sont meilleures, elles sont mises à jour, autrement, elles sont conservées. Cette opération peut être appliquée plusieurs fois durant une itération. L'utilisation de l'algorithme MMAS parallèlement avec la recherche locale pour l'optimisation combinatoire, et le choix d'une stratégie de mélange dynamique entre la meilleure solution locale S^{ib} et la meilleure solution globale S^{ib} pour la mise à jour des phéromones, offre de bonnes performances pour la recherche de solutions (Kong et Tian, 2006).

En résumé, l'algorithme d'optimisation ACO combinatoire comporte les étapes suivantes:

1- Initialisation aléatoire des traînées de phéromones;
2- Tant que la condition d'arrêt n'est pas atteinte, faire :

2.1. Construction des solutions en utilisant les traînées de phéromones;

2.2. Application du principe de la recherche locale (facultatif);

2.3. Mise à jour des phéromones (renforcement et évaporation).

2.4.4 Optimisation ACO continue

Les algorithmes $ACO_{\Re}$ sont des algorithmes itératifs qui tentent de résoudre des problèmes d'optimisation dans un espace continu. À chaque itération, des solutions candidates sont construites d'une manière probabiliste en échantillonnant une distribution de probabilité dans l'espace de recherche. Cette distribution de probabilité est au fur et à mesure des itérations modifiée en utilisant la meilleure solution parmi les solutions construites, le but étant de biaiser (polariser) l'échantillonnage vers les régions de l'espace de recherche contenant les meilleures solutions (Socha et Dorigo, 2006).

Dans les algorithmes d'optimisation ACO combinatoire, la distribution de probabilité est discrète et est dérivée de l'information provenant des phéromones. D'une certaine manière, l'information reliée aux phéromones représente l'expérience de recherche mémorisée de l'algorithme. En revanche, l'algorithme $ACO_{\Re}$ pour l'optimisation continue, utilise une fonction de densité de probabilité continue PDF (*Probability Density Function*). Cette fonction est produite à partir d'une population P de solutions que l'algorithme garde à tout moment.

2.4.5 Fonction de densité de probabilité (PDF) et $ACO_{\Re}$

L'idée fondamentale à la base de l'algorithme $ACO_{\Re}$ est le passage d'une distribution de probabilité discrète à une distribution de probabilité continue par le biais d'une fonction de densité de probabilité ou *"Probability Density Function"* (PDF) (fig. 2.8). Dans $ACO_{\Re}$, au lieu de choisir un composant à partir d'un ensemble limité de valeurs, une fourmi échantillonne une PDF. Dans la partie qui suit, nous présentons brièvement le concept d'échantillonnage d'une PDF. Par la suite, nous présentons une description détaillée de l'algorithme $ACO_{\Re}$.

En principe, la fonction de densité de probabilité d'une variable aléatoire x peut être n'importe quelle fonction $p(x) \geq 0 \quad \forall x$, telle que :

$$\int_{-\infty}^{\infty} p(x)\, dx = 1$$

Une des fonctions les plus populaires employée comme PDF est la fonction gaussienne. Elle a des avantages clairs, tels qu'un procédé facile d'échantillonnage, ex. méthode de Box-Muller (Box et Muller, 58). Pour cette raison, nous employons une fonction PDF basée sur des fonctions gaussiennes. Nous définissons un mélange de fonctions gaussiennes $G^i(x)$ comme étant la somme pondérée de plusieurs fonctions gaussiennes unidimensionnelles $g_l^i(x)$:

$$G^i(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} e^{-\frac{(x-\mu_l^i)^2}{2\sigma_l^{i2}}} \quad \cdots(2.8)$$

La fonction PDF résultante est paramétrée par trois vecteurs :

ω : Vecteur des pondérations des différentes fonctions gaussiennes;

μ^i : Vecteur des moyennes;

σ^i : Vecteur des déviations ou des écarts-type.

La cardinalité de chacun de ces vecteurs est égale au nombre de fonctions gaussiennes constituant la PDF. Pour simplifier, nous prenons le paramètre de cardinalité k : $|\omega| = |\mu^i| = |\sigma^i| = k$.

Une telle fonction (fig. 2.8) permet un procédé d'échantillonnage relativement simple.

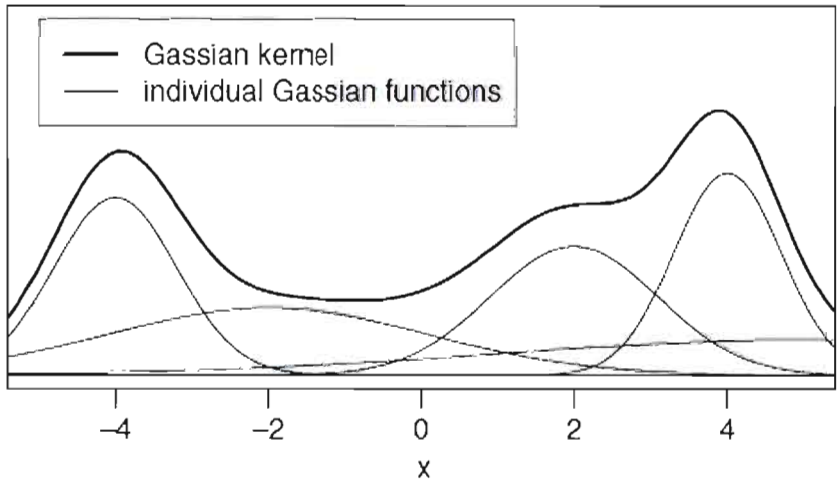


Figure 2.8 Exemple de cinq fonctions gaussiennes et la PDF résultant de leur superposition (Socha et Dorigo, 2006)

2.4.6 Représentation des phéromones dans $ACO_{\Re}$

Dans le cas de l'algorithme ACO pour optimisation combinatoire, l'information relative aux phéromones est stockée dans un tableau. Dans le cas de l'optimisation continue, le choix d'une fourmi n'est pas limité à un ensemble fini. Par conséquent, il est impossible de représenter les phéromones sous forme de tableau. Une approche différente doit être adoptée.

Dans $ACO_{\Re}$ nous maintenons un certain nombre de solutions dans un tableau désigné « archive de solutions » et noté T. Dans un problème n-dimensionnel, chaque solution s_l conserve les valeurs de ses n valeurs ainsi que sa fonction objective $f(s_l)$ dans l'archive T. La structure de l'archive de solutions T est présenté à la fig. 2.9.

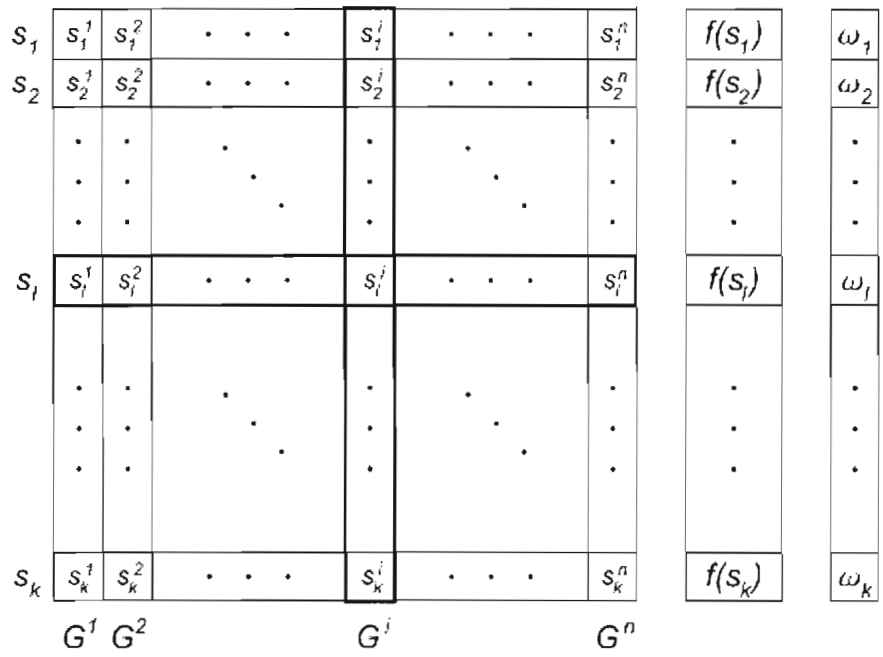


Figure 2.9 Archive T des solutions

Rem. : La $i^{\text{ième}}$ variable de la solution s_l est dénotée par s_l^i.

2.4.7 Principe de l' $ACO_{\mathbb{R}}$

On considère une population de taille k initialisée avec des solutions aléatoires. À chaque itération, un ensemble de solutions m est produit et ajouté à la population P. Le même nombre de mauvaises solutions est enlevé de P. Ce processus vise à polariser la recherche vers les régions de meilleures solutions dans l'espace de recherche.

Les trois vecteurs, ω, μ^i et σ^i étant les paramètres de G^i, On se sert des solutions dans l'archive T pour les déterminer. Une fois la fonction G^i définie, elle sera utilisée pour guider les fourmis dans leur processus de recherche.

On remarque que pour chaque dimension $i = 1, \cdots, n$ du problème, il existe une fonction de densité de probabilité G^i différente (voir fig. 2.9).

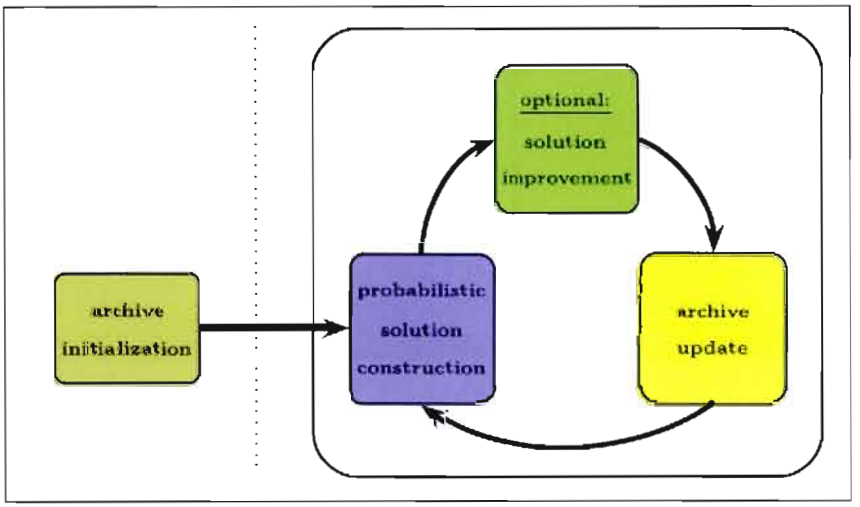


Figure 2.10 Principe de l'optimisation $ACO_{\mathbb{R}}$
(Socha et Dorigo, 2006)

Détermination des moyennes

Le vecteur paramètre μ^i est constitué de l'ensemble des $i^{ième}$ composantes de chacune des k solutions de l'archive T :

$$\mu^i = \{\mu_1^i, \mu_2^i, \cdots, \mu_k^i\} = \{s_1^i, s_2^i, \cdots, s_k^i\} \qquad \cdots(2.9)$$

Détermination des poids des gaussiennes

Chaque solution ajoutée dans l'archives T est évaluée et classée selon son rang; par, exemple, une solution s_l possède un rang l.

Le poids ω_l de la solution s_l est calculé selon la formule suivante :

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} . e^{-\frac{(l-1)^2}{2q^2k^2}} \quad \cdots(2.10)$$

k : taille de la population;

l : rang de la solution;

q : constante.

Le poids est donc défini comme la valeur d'une fonction gaussienne de valeur moyenne 1.0, d'écart type qk et d'argument l. Lorsque la valeur du paramètre q est petite, les solutions les mieux classées sont fortement préférées et lorsque sa valeur est grande, la probabilité de choix des solutions devient plus uniforme. À l'étape de construction i, seules les informations de la $i^{\text{ième}}$ dimension sont employées, de telle façon que la fonction de densité de probabilité G_i résultante soit différente d'étape en étape.

Détermination des vecteurs écarts-type σ_i

Le procédé d'échantillonnage est effectué en deux phases. La première consiste à choisir une parmi les fonctions gaussiennes composant la fonction de densité de probabilité PDF (Socha et Dorigo, 2006).

La probabilité de choisir la $l^{\text{ième}}$ fonction est donnée par :

$$P_l = \frac{w_l}{\sum_{r=1}^{k} w_r}, \quad \forall \quad l = 1, \cdots, k \quad \cdots(2.11)$$

Dans la deuxième phase, la fonction choisie (c.-à-d., fonction g_l^i à l'étape i) est échantillonnée. Le procédé d'échantillonnage peut se faire en utilisant un générateur de nombres aléatoires à distribution

normale, ou bien un générateur aléatoire uniforme en conjonction avec, par exemple, la méthode de Box-Muller (Box et Muller, 1958).

La valeur du paramètre écart type σ^i doit être calculée uniquement pour la fonction gaussienne choisie dans la première phase g_l^i. Par conséquent, nous ne calculons pas le vecteur σ^i en entier, mais uniquement la composante liée à la solution s_l, c.à.d. la composant σ_l^i.

On peut remarquer que la fonction de distribution gaussienne G_l^i diffère à chaque étape de construction i étant donné que les paramètres $\mu_l^i = s_l^i$ et σ_l^i sont calculés dynamiquement à chaque étape.

Afin d'établir la valeur de la norme de déviation σ_l^i à l'étape i de construction, nous calculons la distance moyenne de la solution choisie s_l par rapport aux autres solutions de l'archive T, que nous les multiplierons par le paramètre ξ :

$$\sigma_l^i = \xi \sum_{e=1}^{k} \sqrt{\frac{\left| s_e^i - s_l^i \right|}{k-l}} \quad \cdots(2.12)$$

Le paramètre $\xi > 0$, est le même pour toutes les dimensions. Ce paramètre a un effet semblable à celui du taux d'évaporation des phéromones dans l'optimisation combinatoire ACO. Plus sa valeur est grande, plus la vitesse de convergence de l'algorithme est petite.

Ce processus d'échantillonnage est répété pour chaque dimension $i = 1, 2, \cdots, n$ et à chaque étape de l'itération, la moyenne des distances σ_l^i est calculée uniquement sur une dimension simple i.

2.4.8 Mise à jour des phéromones $ACO_{\Re}$

L'information reliée aux phéromones est stockée en tant qu'archive de solutions, ce qui implique que la procédure de mise à jour doit s'effectuer au niveau de cette même archive. Dans une première étape, l'archive des solutions T est initialisée en générant k solutions par échantillonnage aléatoire uniforme. La mise à jour des phéromones est effectuée par la suite et ce, en ajoutant l'ensemble des meilleures solutions nouvellement crées dans l'archive T, puis en éliminant le même nombre de mauvaises solutions, de sorte que la taille de l'archive reste la même. Ce processus assure le maintien des

meilleures solutions, de façon à guider convenablement le processus de recherche de solutions. Noter que puisque la taille k de l'archive T étant un paramètre de l'algorithme, sa valeur ne doit pas être plus petite que le nombre de dimensions du problème à résoudre.

2.5 Optimisation par essaims particulaires (PSO)

2.5.1 Introduction

Des scientifiques ont constaté que la synchronisation dans les groupes sociaux est effectuée en maintenant des distances optimales entre les différents membres et leurs voisins. Ils en ont déduit que la notion de vitesse dans un essaim est indispensable car elle permet aux membres de s'ajuster à cette distance optimale. Ainsi, la simulation d'un groupe d'oiseaux à la recherche de nourriture afin de déterminer leur comportement social a révélé qu'afin d'arriver au but, les individus du groupe déterminent leurs vitesses par deux facteurs, leur propre meilleure expérience et la meilleure expérience parmi les autres membres du groupe. Selon ce concept, Kennedy et Eberhart ont développé en 1995 l'algorithme PSO pour l'optimisation des fonctions non linéaires continues. L'algorithme d'optimisation PSO a été appliqué avec succès aux réseaux de neurones ainsi qu'à de multiples problèmes d'optimisation non linéaires.

2.5.2 Méthode PSO

L'algorithme d'optimisation par essaims particulaires (PSO) est une autre méthode d'optimisation stochastique. Elle est souvent utilisée pour trouver l'optimum de fonctions non-linaires. À l'instar des autres algorithmes évolutionnaires, l'algorithme PSO fonctionne aussi avec une population, désignée sous le nom d'essaim. Chaque individu de cet essaim s'appelle une particule.

Dans un essaim, chaque particule vole au-dessus de l'hyperespace des solutions afin d'explorer les régions prometteuses selon ses propres expériences et celles du groupe. Par conséquent, les individus profitent des découvertes et expériences des autres particules pendant leur processus de recherche. Ce mécanisme flexible permet de combiner la recherche globale et la recherche locale, garantissant ainsi la convergence vers des solutions optimales. L'algorithme PSO possède une bonne capacité de recherche au-dessus d'une large perspective dans l'espace des solutions.

Dans la formulation originale de PSO, chaque particule est définie comme solution potentielle au problème dans l'espace *n*-dimensionnel et chaque particule maintient en mémoire sa meilleure position précédente ainsi que celle de l'essaim. De toutes les positions des particules, celle possédant la valeur fitness la plus élevée durant la course complète s'appelle la meilleure globale (désigné par le vecteur *s* dans la suite du document).
Une particule d'un essaim est donc caractérisée, à l'instant t par :

- $\vec{x}_i(t)$: sa position dans l'espace de recherche;
- $\vec{v}_i(t)$: sa vitesse;
- $\vec{x}_{pbest_i}(t)$: la position de la meilleure solution par laquelle elle est passée;
- $\vec{x}_{sbest_i}(t)$: la position de la meilleure solution connue de l'essaim;
- $pbest_i$: la valeur de fitness de sa meilleure solution;
- $sbest_i$: la valeur de fitness de la meilleure solution connue de l'essaim;

2.5.3 Principe de l'algorithme PSO

Optimisation continue

Dans l'algorithme d'optimisation PSO original, un nombre fixe de solutions (particules) sont aléatoirement initialisés dans un espace de solutions de dimension *n*. Une particule *i* au temps *t* possède une vitesse v_i^t et une position x_i^t représentant une solution. Une fonction objective *f* détermine la qualité de la position de chaque particule.

Une particule *i* dispose d'un vecteur p_i représentant sa meilleure position antérieure, associée à une valeur fitness, $pbest_i = f(p_i)$.
De même, la meilleure position que l'essaim ait pu visitée durant tout le parcours est stockée dans un vecteur s, dont la valeur fitness est désignée par $gbest_i = f(s)$. (Montes de Oca, Stutzle, Birattari et Dorigo, 2006).

Le processus de mise à jour dans PSO s'effectue selon les formules suivantes :

$$v_i^{t+1} = \omega\, v_i^t + \varphi_1 * (p_i - x_i^t) + \varphi_2 * (s - x_i^t) \quad \cdots(2.13)$$

$$x_i^{t+1} = x_i^{t+1} + v_i^{t+1} \qquad \cdots(2.14)$$

Tels que φ_1, φ_2 représentent les coefficients pondérant le comportement conservateur (tendance de la particule à vouloir suivre son instinct) et le panurgisme (tendance à suivre le groupe). Ces coefficients sont définis par :

$$\begin{cases} \varphi_1 = r_1.c_1 \\ \varphi_2 = r_2.c_2 \end{cases}$$

Où:

r_1, r_2, représentent deux vecteurs aléatoires, uniformément distribués n-dimensionnels dans l'intervalle [0, 1].

c_1, c_2, sont deux constantes positives appelées coefficients d'apprentissage cognitif et social respectivement.

Le facteur d'inertie ω permet de définir la capacité d'exploration de chaque particule en vue d'améliorer la convergence de l'algorithme. Une grande valeur (ω>1) est synonyme d'une grande amplitude de mouvement et donc une capacité d'exploration globale, et à contrario, une faible valeur (ω< 1) est synonyme de faible amplitude de mouvement, et donc une capacité d'exploration locale. Fixer ce facteur revient donc à trouver un compromis entre l'exploration locale et l'exploration globale.

Pour éviter que les particules ne se déplacent trop rapidement dans l'espace de recherche, passant éventuellement à côté d'une solution optimale, il est essentiel de saturer la vitesse de déplacement des particules v_{Max}.

$$v_i^t = \begin{cases} v_i^t = v_{Max} & si\ v_i^t \geq v_{Max} \\ v_i^t = v_{Min} & si\ v_i^t \leq v_{Min} \\ v_i^t & \text{sinon} \end{cases} \qquad \cdots(2.15)$$

Une fois la mise à jour terminée, l'algorithme réitère en mettant à jour à chaque étape d'une itération la vitesse et la position des particules et ce, jusqu'à atteindre un critère d'arrêt.

Optimisation combinatoire

L'algorithme PSO original était limité à des espaces continus. Cependant, beaucoup de problèmes d'optimisation concernent des problèmes combinatoires. Pour contrer les limites, Kennedy et Eberhart (Kennedy et Eberhart, 1997) ont développé une version discrète de l'algorithme PSO qui diffère essentiellement de l'original dans deux aspects (Ching-Jong, Chao-Tang et Pin, 2007) :

- Les particules, dans le cas combinatoire sont composées de valeurs binaires $\{0, 1\}$;
- La vitesse est transformée pour le calcul des probabilités.

De la même manière que dans le cas continu, chaque particule ajuste sa vitesse en fonction de sa meilleure solution trouvée et la meilleure solution de l'essaim, Eq. (2.13). Selon Kennedy et Eberhart (Kennedy et Eberhart, 1997), lorsque la valeur de la vitesse reliée à un bit dans une particule est élevée, elle favorise le choix de position 1, alors que lorsqu'elle est faible, elle favorise le choix de position 0. De plus, ils contraignent la valeur de la vitesse à un intervalle [0, 1] (Kennedy et Eberhart, 2001), en utilisant une fonction sigmoïde.

$$S(v_i^t) = \frac{1}{1+\exp(-v_i^t)} \qquad \cdots(2.16)$$

Tel que $S(v_i^t)$ représente la probabilité du bit x_i^t à prendre la valeur 1.

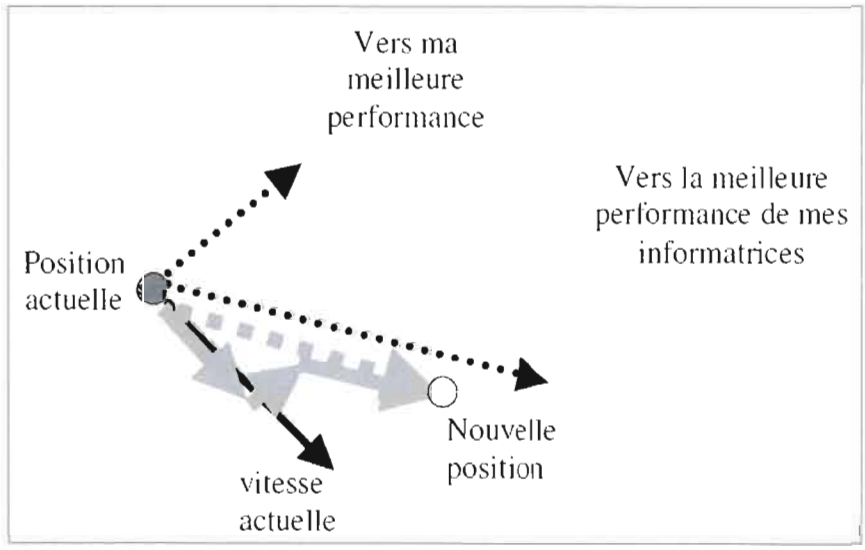


Figure 2.11 Schéma de principe du déplacement d'une particule
(Clerc et Siarry, 2003)

Les avantages de l'algorithme PSO est qu'il utilise très peu de paramètres. D'autre part, il converge très rapidement vers une solution optimale aussi bien dans le domaine continu que dans le domaine discret.

2.6 Conclusion

Nous avons présenté dans ce chapitre les principes des algorithmes d'optimisation évolutionnaires (AG, ACO et PSO). Ces derniers seront introduits dans la partie simulation pour la recherche de solutions optimales. On les utilise dans un premier temps avec un procédé d'apprentissage par renforcement pour la classification de problèmes non linéairement séparables. Par la suite, en s'en sert pour effectuer l'apprentissage des réseaux classiques, RBF et MLP où les individus seront définis par des génotypes représentant les poids de connexion des neurones.

Dans ce travail, on utilise un modèle de réseau récurrent (dynamique) qui consiste en une mémoire associative bidirectionnelle (Chartier et Boukadoum 2006a). Dans le prochain chapitre, nous présentons dans un premier temps des notions de base sur les systèmes dynamiques, les phénomènes chaotiques ainsi que les différents types d'attracteurs présents dans l'espace d'états de ces systèmes. Par la suite nous considérerons plus particulièrement le modèle BAM à fonction de sortie chaotique et étudierons sa dynamique interne afin de mieux connaître ses capacités et contrôler son comportement vis-à-vis d'événements externes, cela en vue d'optimiser sa performance face à des problèmes non linéairement séparables.

Chapitre 3

SYSTÈMES DYNAMIQUES ET CHAOS

3.1 Introduction

Un système dynamique est un modèle permettant de décrire l'évolution d'un ensemble d'objets en interaction au cours du temps (Bergé, Pomeau et Vidal, 1988). L'ensemble d'objets est généralement défini par le modélisateur. Une pratique courante pour étudier ces systèmes est de les décrire à l'aide de variables d'état dont les valeurs évoluent au cours du temps et d'interactions entre ces variables. En particulier ces variables peuvent être celles d'un algorithme récursif qui change leurs valeurs à chaque itération.

Un système dynamique est spécifié par :

- Une *représentation d'état* : Décrit ce qui relève du domaine structurel. Il s'agit d'une liste de variables, que l'on appelle vecteur d'état, permettant de décrire à tout instant notre ensemble d'objets. Le nombre de ces variables correspond au nombre de *degrés de liberté* du système.

- une *fonction de transition* : Décrit ce qui relève du domaine temporel. Elle définit toutes les forces, contraintes élastiques, collisions, et plus généralement les échanges d'énergie entre les objets. Elle décrit l'évolution d'un vecteur d'état entre deux instants t_1 et t_2.

Un système dynamique est donc la donnée d'un vecteur d'état et d'une fonction de transition.

3.2 Espace des phases

On appelle espace des phases, un espace abstrait dont les axes sont les variables dynamiques du système, et trajectoire de phases toute courbe dans cet espace représentative d'une évolution du système. Un ensemble de trajectoires de phases constitue un portrait de phases. L'espace des phases est donc un outil visuel d'analyse de la dynamique des systèmes complexes. Une courbe fermée et régulière représente un comportement stable ou cyclique; une courbe déformée ou une succession de boucles signifie que les mouvements subissent une perturbation. Cette représentation graphique rend

plus aisé d'évaluer la stabilité et la complexité d'un système dynamique (voir Fig. 3.1 pour des exemples).

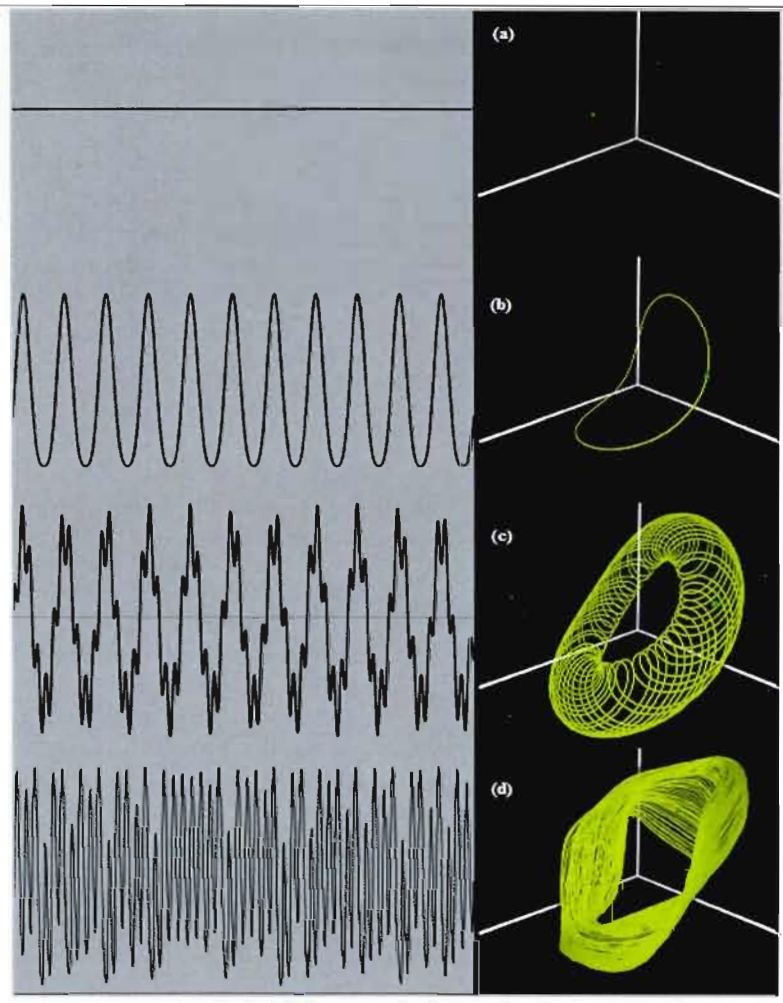


Figure 3.1 Exemples de représentation de la dynamique dans l'espace des phases

3.3 Systèmes chaotiques

Les systèmes chaotiques sont des systèmes dynamiques décrits par des fonctions de transition déterministes, mais qui peuvent exhiber des comportements imprévisibles, pouvant sembler aléatoires. En particulier, ils se caractérisent par une très grande sensibilité aux conditions initiales. Cependant, chaque condition initiale détermine entièrement l'évolution future et il n'y a pas de hasard qui intervient dans le processus. Néanmoins, deux conditions initiales très proches peuvent avoir des évolutions complètement différentes. L'évolution du système peut alors être considérée imprévisible car une erreur insignifiante au départ conduit à des résultats complètement différents au bout d'un certain temps. C'est le chaos déterministe.

Les études théoriques actuelles concernent plus particulièrement les dynamiques chaotiques des systèmes, décrits par des modèles continus et/ou discrets. L'approche adoptée est celle des méthodes qualitatives de la dynamique, c'est-à-dire l'identification de la structure de l'espace d'états (dit aussi de phases) et l'espace paramétrique des systèmes modélisés en vue de leur analyse et de leur synthèse. L'évolution des systèmes considérés dépend généralement de leurs paramètres, rendant nécessaire de s'intéresser aux phénomènes de bifurcation, c'est-à-dire aux changements de comportements qualitatifs d'un système sous l'effet de petites variations quantitatives de ses paramètres.

3.4 Diagramme du Cobweb

Le diagramme du Cobweb est une procédure spécialement adaptée pour l'analyse qualitative du comportement d'une fonction itérative f à une dimension. Ce diagramme est très utile pour déterminer l'évolution des itérations de la fonction f pour une condition initiale donnée et pour une valeur de paramètre donnée (Fig. 3.5).

3.5 Diagramme de bifurcation

Une évolution d'un point fixe vers le chaos n'est pas progressive, mais marquée par des changements discontinus de comportement appelés bifurcations. Une bifurcation marque le passage soudain d'un régime dynamique à un autre, qualitativement différent. Le diagramme de bifurcation est une portion de l'espace des paramètres sur laquelle sont représentés tous les points de bifurcation. Le diagramme de bifurcation permet de mieux visualiser l'évolution d'un système vers le chaos en fonction ses paramètres.

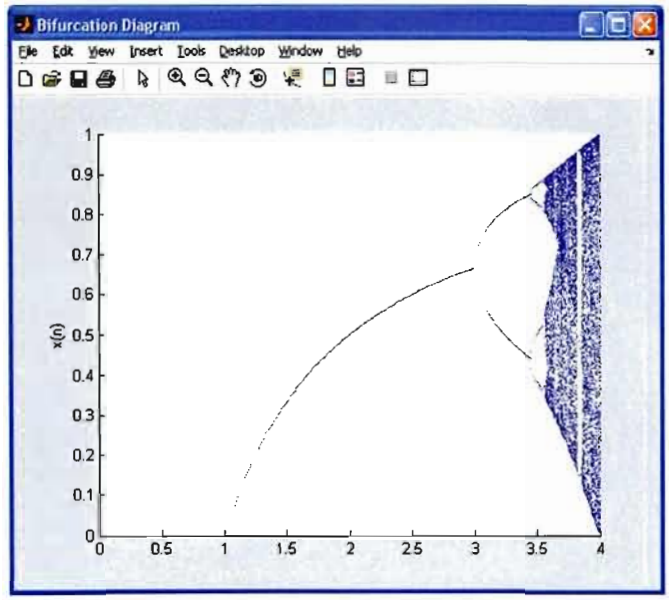


Figure 3.2 Diagramme de bifurcation d'une carte logistique $g_a(x) = \mu x(1-x)$

Dans le cas d'une carte logistique (Fig. 3.2), lorsque μ =3, on observe une instabilité du point fixe et un doublement de période, c'est une *bifurcation*. Avant de basculer dans le chaos, il y une cascade de dédoublements de période. Après un dédoublement de périodes, on se retrouve dans un régime chaotique dans lequel les orbites périodiques précédentes sont toujours présentes mais deviennent instables, ce qui explique qu'elles ne soient pas visibles sur le diagramme de bifurcation. Le régime chaotique possède donc *une infinité d'orbites périodiques*.

3.6 Attracteur

Dans un système dynamique, tout ensemble de conditions initiales appartenant à un volume donné peut converger vers un ensemble, appelé attracteur. Ce dernier est défini comme un ensemble de volume nul, invariant par le flot. Tout point de l'espace d'état qui appartient à un attracteur demeure à l'intérieur de cet attracteur en tout temps. Un attracteur peut être un point, un cycle limite, un tore, ou avoir une structure plus complexe encore, de type fractale; on parle alors d'*attracteur étrange*.

Lorsque l'attracteur est un point fixe, le système dynamique tend à se comporter de manière statique. Lorsque l'attracteur est un cycle limite, le système présente un comportement oscillatoire qui se maintient sur le long terme, et lorsque l'attracteur est étrange, la trajectoire sur l'attracteur est complexe et manifeste la propriété de sensibilité aux conditions initiales

Un bassin d'attraction représente l'ensemble de conditions initiales qui conduisent la trajectoire vers l'attracteur (Fig. 3.4).

3.7 Attracteur chaotique

Afin de bien rendre compte de ce qui peut être observé dans un espace des phases à trois dimensions, on présente l'attracteur de Rössler (fig. 3.3). Il s'agit d'un *attracteur chaotique* (ou étrange).

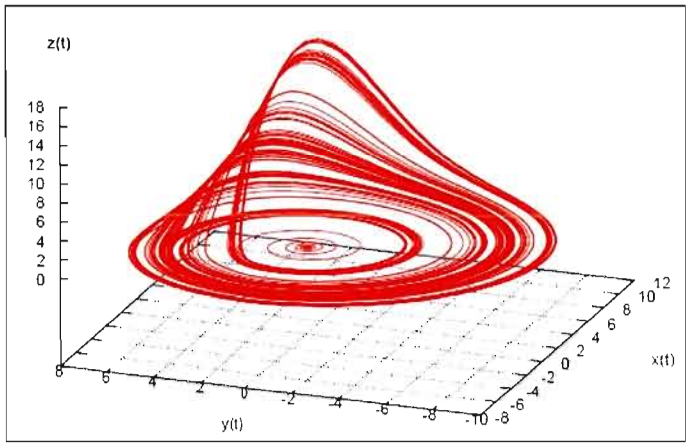


Figure 3.3 Attracteur chaotique de Rössler.

Les équations du système de Rössler sont données par le système différentiel suivant, pour lequel $a = b = 0.2$, et $c = 5$:

$$\begin{cases} \dfrac{dx}{dt} = -(y + z) \\ \dfrac{dy}{dt} = x + ay \\ \dfrac{dz}{dt} = b - cz + xz \end{cases} \quad \cdots(3.1)$$

Un attracteur chaotique possède notamment la propriété remarquable suivante : *la trajectoire ne repasse jamais par un même état.* Cela signifie, entre autres, que cette trajectoire passe par une infinité d'états.

Attracteur de Hénon

Les équations du système de Hénon sont données par:

$$\begin{cases} x_{n+1} = y_n + 1 - a\,x_n^2 \\ y_{n+1} = b\,x_n \end{cases} \quad \cdots(3.2)$$

On prend les valeurs suivantes : a = 1.28, b= -0.3.

En très peu d'itérations, le système converge vers l'attracteur chaotique illustré sur la fig. 3.4 et désigné attracteur de Hénon.

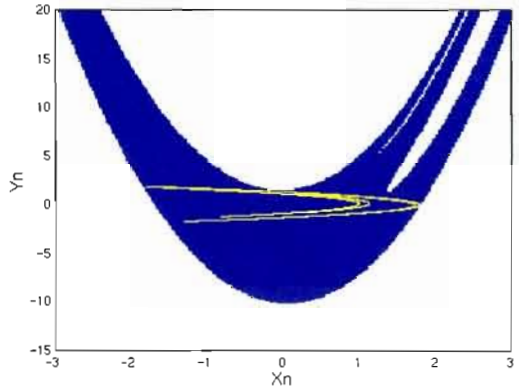


Figure 3.4 Attracteur chaotique de Hénon (en jaune)
Bassin d'attraction (en bleu)

Attracteur de Bernoulli

Les équations du système de Bernoulli sont données par :

$$\begin{cases} x_{n+1} = y_n \\ y_{n+1} = \mathrm{mod}(\delta\, x_n, 1) \end{cases} \quad \cdots(3.3)$$

3.8 Carte logistique

Pour de nombreux processus de croissance, le nombre d'individus de la génération suivante x_{n+1} est une fonction linéaire de la génération présente x_n :

$$x_{n+1} = \mu\, x_n$$

Où μ est un paramètre de croissance. Si la croissance n'est pas limitée, elle suit une loi géométrique:

$$x_n = \mu^n x_0$$

et tend vers l'infini pour $\mu > 1$. Mais la croissance est souvent limitée par les ressources disponibles. En d'autres termes, plus une population x_n est grande, plus le taux de croissance μ est petit. La façon la plus simple pour modéliser le déclin du taux de croissance est de remplacer μ par $\mu(x_{max} - x_n)$. Ainsi, lorsque x_n approche la limite x_{max}, le taux de croissance μ tend vers 0.

En normalisant la population à 1, on obtient la loi de croissance :

$$x_{n+1} = f(x_n) = \mu x_n (1 - x_n) \quad \cdots(3.4)$$

appelée *carte quadratique*, ou encore, en raison de son utilisation historique en logistique militaire et de sa forme parabolique, *carte logistique* ou *parabole logistique*.

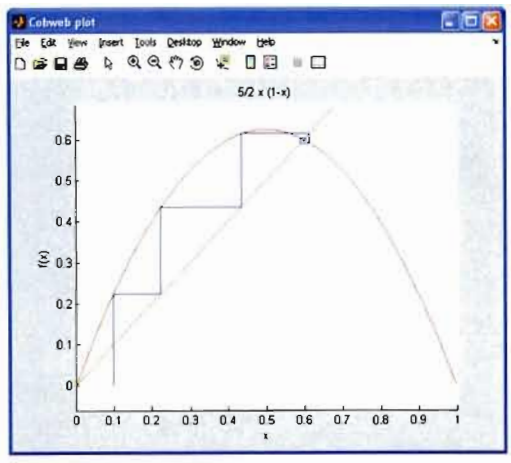
L'équation logistique a été introduite en 1845 par le sociologue et mathématicien belge Pierre-François Verhulst (1804-1848) pour modéliser la croissance de populations dans un milieu aux ressources limitées. L'emploi du terme "logistique" ne s'est toutefois généralisé qu'à partir de 1875.

La dynamique de ce système présente un comportement très différent selon la valeur du paramètre μ :

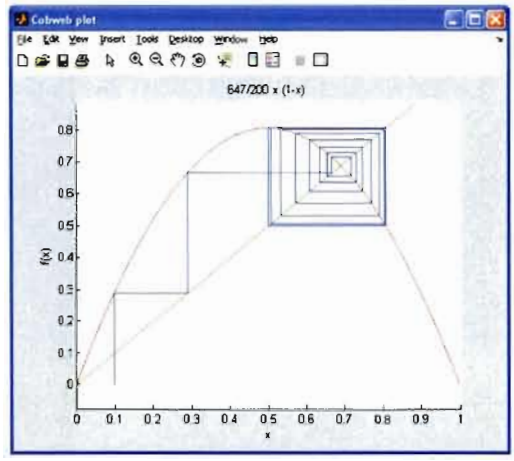
- Pour $0 \leq \mu < 1$, le système converge vers un point fixe attracteur, x=0.
- Pour $1 < \mu \leq 3$, le système converge vers un point fixe attracteur, x= $\frac{\mu - 1}{\mu}$. Celui-ci devient instable lorsque μ=3.
- Pour $3 < \mu < 3{,}57$, le système possède un attracteur qui est une orbite périodique, de période 2^n où n est un entier qui tend vers l'infini lorsque μ tend vers 3,57...
- Lorsque $\mu = 3.57...$, le système possède un attracteur de Feigenbaum (attracteur non chaotique) découvert par May en 1976.
- $3.57 < k \leq 4$: La longueur du cycle s'allonge et devient tellement complexe que l'on peut difficilement suivre son évolution, le système passe par une phase que l'on qualifie de chaotique, et son comportement semble être aléatoire.

On obtient donc une succession de bifurcations de la régularité vers le chaos lorsque le paramètre augmente.

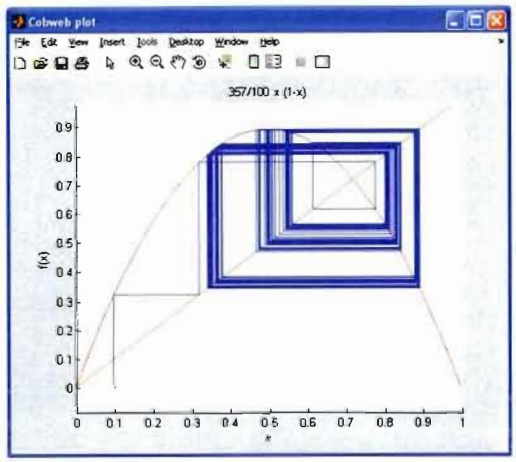
La carte logistique revêt une importance particulière en dynamique des systèmes car elle rend compte du caractère universel de la transition vers le chaos par doublement de période.



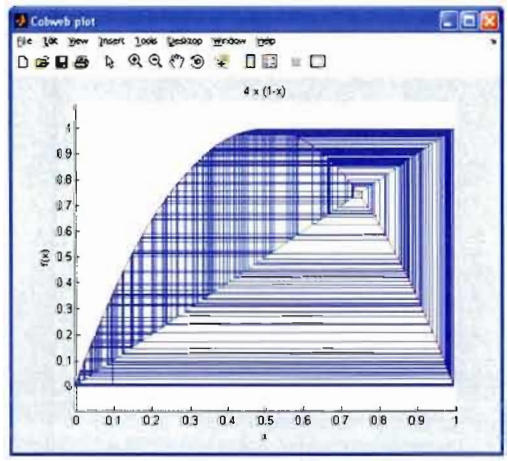
(a) μ =2.5: point fixe



(b) μ =3.235: Orbite périodique



(c) μ =3.5: Orbite périodique



(d) μ =4: Comportement chaotique

Figure 3.5 Diagramme du Cobweb d'une carte logistique pour différentes valeurs du paramètre μ

3.9 Théorie des catastrophes

La théorie des catastrophes, fondée par René Thom dans les années 1960 (Stewart, 1983), étudie et classe les phénomènes caractérisés par des changements soudains dans le comportement, résultant de petits changements de circonstances. Elle vise à décrire ces phénomènes discontinus à l'aide de modèles mathématiques continus. Les catastrophes sont des bifurcations entre les différents états d'équilibres d'un système dynamique. Elles peuvent être classées selon le nombre de paramètres de contrôle variant simultanément. Par exemple, si l'on a deux paramètres de contrôle, on retrouve le type de catastrophe le plus commun, appelé catastrophe fronce ou *cusp catastroph*. Dans l'espace des états, il existe des points correspondant à des valeurs catastrophiques des paramètres. Ces points forment des frontières dont le franchissement correspond au passage d'un point fixe stable à un autre point fixe stable. La codimension d'une catastrophe désigne le nombre de paramètres de contrôle nécessaires pour rendre les singularités (variations soudaines) stables. Lorsque la codimension est inférieure ou égale à 5, les catastrophes élémentaires sont en nombre fini.

Le théorème fondamental de la théorie des catastrophes stipule l'existence des sept catastrophes élémentaires suivantes :

1. Pour un seul paramètre de contrôle a (codimension 1) et une variable d'état x :
   - Le pli (Fig. 3.6 (a)) : $V = x^3 + ax$
2. Pour deux paramètres a et b (codimension 2) et une variable d'état x :
   - La fronce (Fig. 3.6 (b)) : $V = x^4 + ax^2 + bx$
3. Pour trois paramètres a, b et c (codimension 3) et,
   a. une variable d'état x :
      - La queue d'aronde (Fig. 3.6 (c)) : $V = x^5 + ax^3 + bx^2 + cx$
   b. deux variables d'état x et y :
      - L'ombilic hyperbolique (vague) (Fig. 3.6 (d)): $V = x^3 + y^3 + axy + bx + cy;$
      - L'ombilic elliptique (poil) (Fig. 3.6 (e)): $V = x^3 / 3 - xy^2 + a(x^2 + y^2) + bx + cy$
4. Pour quatre paramètres de contrôle a, b, c et d (codimension 4) et,
   a. une variable d'état x :
      - Le papillon (Fig. 3.6 (f)) $V = x^6 + ax^4 + bx^3 + cx^2 + dx$
   b. deux variables d'état x et y :
      - l'ombilic parabolique (champignon) (Fig. 3.6 (g)):
        $V = x^2y + y^4 + ax^2 + by^2 + cx + dy$

Avec cinq paramètres, il existe encore quatre formes de catastrophes; ainsi, avec au plus cinq paramètres, il n'existe que onze formes de catastrophes distinctes. Enfin, avec six paramètres de contrôle ou plus, la classification des catastrophes devient infinie.

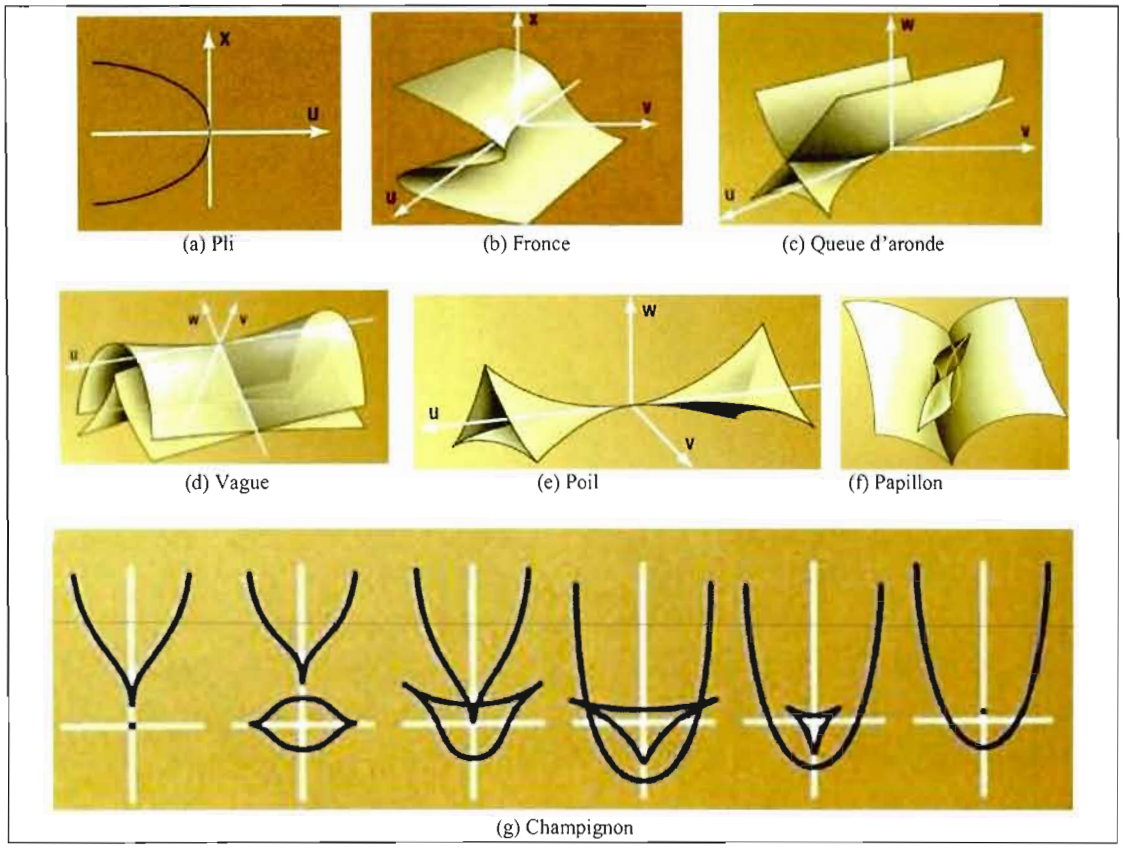


Figure 3.6 Les sept catastrophes élémentaires

3.10 Exposants de Lyapunov

L'exposant de Lyapunov est une mesure du degré de sensibilité d'un système dynamique aux conditions initiales. Les points fixes stables possèdent des exposants strictement négatifs. Les cycles limites possèdent un exposant nul, et les autres négatifs. Les tores T^2 ont deux exposants nuls, et les autres négatifs. Si la dynamique possède au moins un exposant positif, alors il existe une direction pour laquelle deux conditions initialement très proches, produisent des trajectoires qui tendent à s'éloigner à une vitesse exponentielle. Un exposant positif marque ainsi la sensibilité aux conditions initiales, donc le caractère chaotique d'un système dynamique.

Ex. : Un système décrit par une suite x vérifiant la relation de récurrence, $x_{i+1} = f(x_i)$ possède un exposant de Lyapunov défini par :

$$\lambda = \lim_{n \to +\infty} \frac{1}{n} \sum_{t=t_0+1}^{t_0+n} \ln \left| \frac{dx(t+1)}{dx(t)} \right| \quad \cdots(3.5)$$

Une carte logistique possède donc un exposant de Lyapunov dont l'expression est :

$$\lambda = \lim_{n \to +\infty} \sum_{i=0}^{n} \ln \left| \mu(1-2x_i) \right|$$

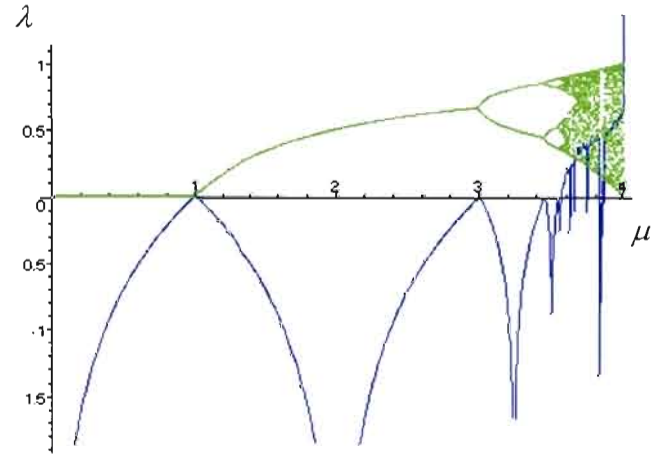


Figure 3.7 Diagramme de bifurcation et exposant de Lyapunov de la carte logistique

Lorsque l'exposant de Lypunov λ <0, l'erreur croit de manière non exponentielle et le système n'est pas dans un régime chaotique. En revanche, si λ>0, l'erreur croit de manière exponentielle et le système est chaotique (Fig. 3.7).

3.11 BAM à fonction de sortie chaotique

3.11.1 Architecture

Le modèle BAM à fonction de sortie chaotique (Chartier et Boukadoum, 2006a) est illustré à la fig. 3.8. Il est constitué de deux réseaux de neurones de type Hopfield interconnectés, qui permettent un flux récurrent bidirectionnel de l'information. La couche *y* retourne l'information à la couche *x* et vice-versa. Ce modèle BAM peut constituer une mémoire auto-associative ou une mémoire hétéro-associative et les deux couches peuvent être de dimensions différentes. Et contrairement à la plupart des modèles BAM, la matrice de poids d'un côté n'est pas nécessairement la transposée de l'autre.

x(0) et y(0) sont les valeurs d'entrée initiales (stimuli); *t* est le nombre d'itérations; et **W** et **V** sont les matrices de poids.

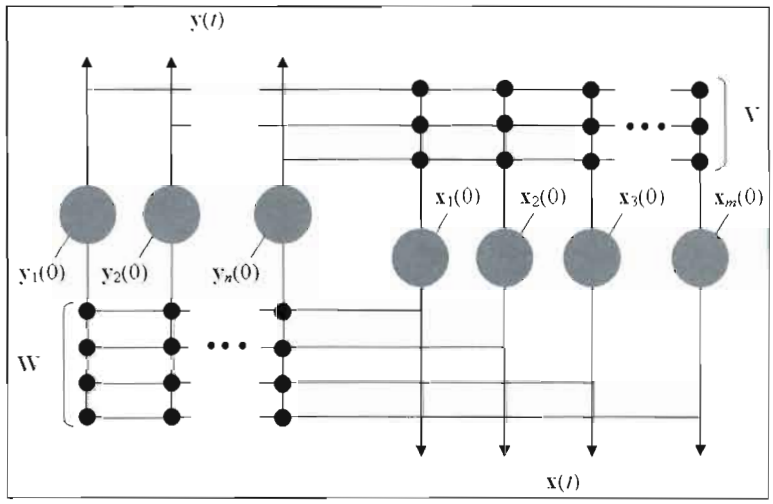


Figure 3.8 Architecture de la mémoire hétéro-associative bidirectionnelle (Chartier, Renaud et Boukadoum, 2008)

Dans ce modèle, le réseau tente de résoudre les contraintes non linéaires suivantes:

$$\begin{cases} X = f(VY) \\ \qquad\qquad \cdots(3.6) \\ Y = f(WX) \end{cases}$$

La fonction de sortie f est définie par les équations suivantes (Chartier, Renaud et Boukadoum, 2008):

$$\forall i = 1,\ldots,N; \qquad y_i(t+1) = (\delta+1)Wx_i(t) - \delta(Wx_i(t))^3 \qquad \cdots(3.7)$$

et

$$\forall i = 1,\ldots,M; \qquad x_i(t+1) = (\delta+1)Vy_i(t) - \delta(Vy_i(t))^3 \qquad \cdots(3.8)$$

δ est un paramètre général de transmission, qui doit être préalablement fixé. Si on veut assurer un comportement stable de la BAM, on doit fixer sa valeur entre 0 et 0.5 (voir fig. 3.10). Lorsque sa valeur augmente, la BAM peut converger aussi bien vers des attracteurs stables, cycliques ou chaotiques (Fig. 3.9 et 3.10).

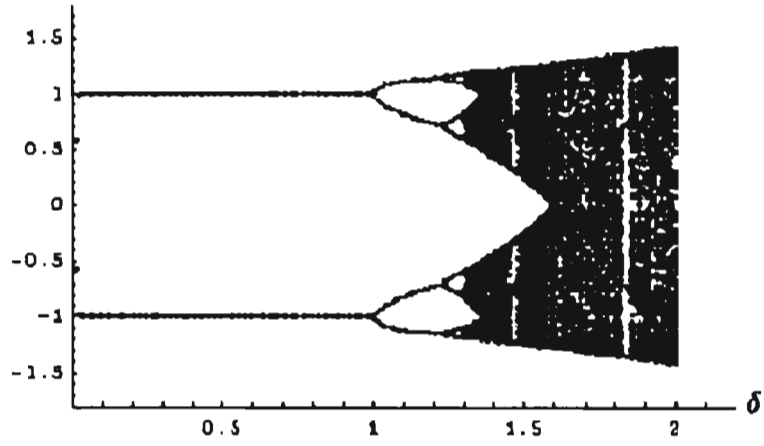


Figure 3.9 Diagramme de bifurcation de la BAM, en fonction de la valeur du paramètre de transmission δ

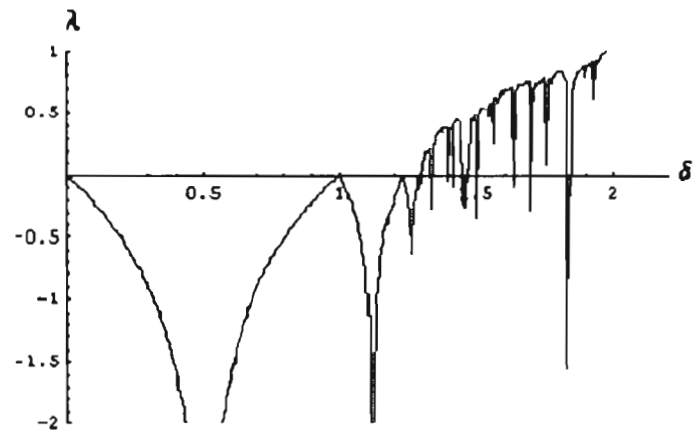


Figure 3.10 Exposant de Lyapunov de la BAM, en fonction de la valeur du paramètre de transmission δ

3.11.2 Règle d'apprentissage BAM

Le modèle BAM utilise une règle d'apprentissage d'inspiration hebbienne, dérivée de l'approche hebb/anti-hebb qui suit un principe d'association hebbienne basée sur les différences temporelles, et formellement exprimée par les équations suivantes (Chartier et Boukadoum, 2006a) :

$$W(k+1) = W(k) + \eta(y(0) - y(t))(x(0) + x(t))^T \qquad \cdots(3.9)$$
$$V(k+1) = V(k) + \eta(x(0) - x(t))(y(0) + y(t))^T$$

où

k , représente le nombre d'essais d'apprentissage;

η , est un paramètre général de l'apprentissage;

W et V représentent les matrices des poids de connexion.

Dans un premier temps, des patrons d'entrée x(0) and y(0) alimentent le réseau. Par la suite, ils sont itérés t fois à travers le réseau (Fig. 3.8). Les valeurs de poids s'auto stabilisent lorsqu'on obtient une valeur de retour (*feedback*) semblable à celle de l'entrée initiale (t=0), c.à.d. lorsque y(t)=y(0) et x(t)=x(0) ou à son complément, c.à.d. y(t) = - y(0) et x(t) = - x(0). En d'autres termes, lorsque le réseau développe un point fixe. C'est un procédé en ligne qui diffère de celui utilisé par la plupart des BAM proposées dans la littérature et pour lesquelles l'apprentissage est effectué hors-ligne.

Afin de permettre la convergence des poids, la valeur du paramètre d'apprentissage η doit être fixée en accord avec la condition suivante (Chartier, Renaud et Boukadoum, 2008) :

$$\eta < \frac{1}{2(1-2\delta)Max[N,M]}, \quad \delta \neq 1/2 \qquad \cdots(3.10)$$

N et M sont les nombres d'unités de chacune des deux couches, δ est le paramètre de transmission.

Dans la partie qui suit, on présente une étude théorique dévoilant le rôle et l'avantage que peut avoir un paramètre d'asymétrie dans la fonction de sortie pour l'amélioration de la performance du modèle BAM.

3.11.3 Paramètre symétrique une-dimension

La fonction de sortie utilisée dans le modèle BAM est inspirée de l'équation classique de Verhulst. Étant donné que la carte logistique qui en découle possède un seul point fixe stable, l'équation à été modifiée pour donner lieu à une carte cubique possédant deux points fixes (1 et -1 pour des patrons bipolaires) (voir fig. 3.9). Cette fonction est décrite par l'équation dynamique suivante (Chartier, Boukadoum et Amari, 2009) :

$$\frac{dx}{dt} = f(x) = h + r(w * x) - (w * x)^3 \qquad \cdots(3.11)$$

w : représente les poids de connexions;

x : est une valeur de l'entrée;

h : représente un biais ou paramètre d'asymétrie;

r : est un paramètre qui contrôle l'état d'équilibre du système.

Lorsque $h = 0$, la fonction de sortie est identique à celle donnée dans (Chartier, Renaud et Boukadoum, 2008). La fig. 3.11 illustre sa forme pour $r = w = 1$. Les racines de l'Eq. 3.11 représentent les points fixes du système.

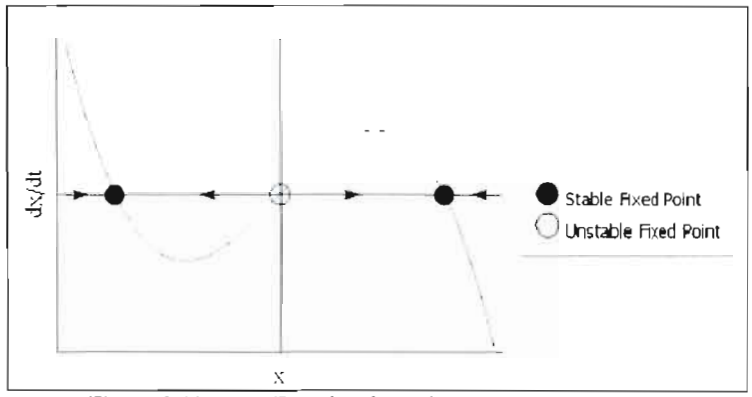


Figure 3.11 Fonction de sortie pour $w = r = 1$ et $h = 0$.
Les deux cercles pleins représentent les points fixes stables;
Le cercle vide représente un point fixe instable.
(Chartier, Boukadoum et Amari, 2009)

Pour cet exemple, les points fixes correspondants seraient x = -1, 0 et 1. La stabilité de ces points fixes est déterminée à partir de la fonction dérivée de l'Eq. 3.11, son expression est :

$$f'(x) = r * w - 3w^3 * x^2$$

Lorsque la dérivée sur un point fixe est supérieure à zéro, toute petite perturbation provoquerait une amplification (point fixe instable); si par contre la dérivée est négative, une légère perturbation provoquerait plutôt un affaiblissement et une dégradation (point fixe stable). Dans l'exemple donné précédemment, le point fixe $x = 0$ est instable, et les deux points $x = 1$ *et* $x = -1$ sont stables (voir fig. 3.11).

Une autre façon de visualiser la dynamique du réseau est basée sur la notion d'énergie du système (Hopfield, 1982).

L'énergie $E(x)$ est définie par:

$$-\frac{dE}{dx} = \frac{dx}{dt}$$

Le vecteur d'état converge globalement vers les états décroissants d'énergie. L'équilibre est atteint lorsque le vecteur d'état converge vers un minima local correspondant à un point fixe stable, ou un maxima local correspondant à un point fixe instable. Trouver les points fixes revient alors à trouver la solution $E(x)$, telle que :

$$-dE/dx = h + r(w*x) - (w*x)^3$$

La solution générale est donnée par :

$$E(x) = -h*x - \frac{r*w*x^2}{2} + \frac{w^3*x^4}{4} + C \qquad \cdots(3.12)$$

Telle que C est une constante arbitraire ($C = 0$ pour simplifier).

La figure 3.12 illustre la fonction d'énergie lorsque $r = w = 1$ et $h = 0$. Le système exhibe un double puits de potentiel possédant deux points d'équilibre stables $x = -1$ *et* $x = 1$.

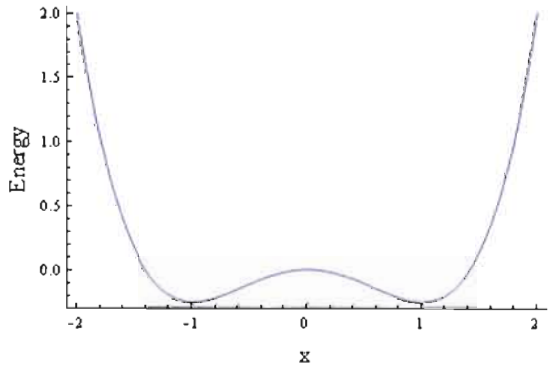


Figure 3.12 Paysage d'énergie d'un système unidimensionnel (Chartier, Boukadoum et Amari, 2009)

Le paramètre r joue un rôle important dans la détermination du nombre et de la position des points fixes. La fig. 3.13 illustre la situation lorsque $w = 1$ et $h = 0$.

$r < 0$, le système possède un seul point fixe stable, $x = 0$;

$r > 0$, le système possède deux points fixes stables $x = \pm\sqrt{r}$;

$r = 0$, on obtient une bifurcation de type pitchfork.

De ce fait, afin que le modèle BAM puisse être capable de stocker des stimuli binaires, il est important d'avoir $r > 0$.

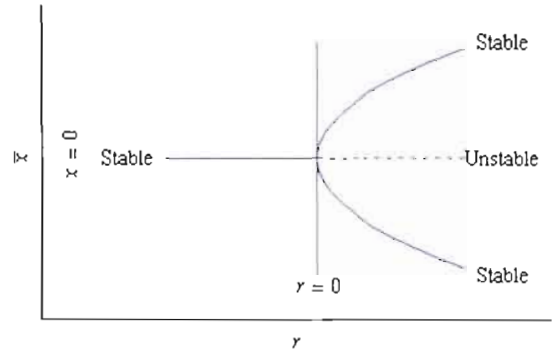


Figure 3.13 Diagramme de phase indiquant la présence d'une bifurcation pour $r = 0$ (Chartier, Boukadoum et Amari, 2009)

Les résultats précédents ont été obtenus avec une fonction de sortie symétrique ($h = 0$). Lorsque $h \neq 0$, la symétrie est rompue et l'Eq. 3.11 mène à un système catastrophique de type fronce (*Cusp*).

Afin de déterminer la valeur de r et h qui mènent vers une dynamique à un versus deux points fixes stables, l'Eq. 3.11 est mise à zéro et divisée en deux parties. Si l'on fixe $w = 1$ pour simplifier, on obtient les deux équations (Chartier, Boukadoum et Amari, 2009) :

$$y = rx - x^3 \qquad \cdots(3.13a)$$

$$y = -h \qquad \cdots(3.13b)$$

Les racines de la dérivée sont alors :

$$x_{Min/Max} = \pm\sqrt{\frac{r}{3}} \qquad \cdots(3.14)$$

En substituant le résultat dans l'Eq. 3.13b, on obtient :

$$rx_{Min/Max} - x^3_{Min/Max} = \pm\frac{2r}{3}\sqrt{\frac{r}{3}} \qquad \cdots(3.15)$$

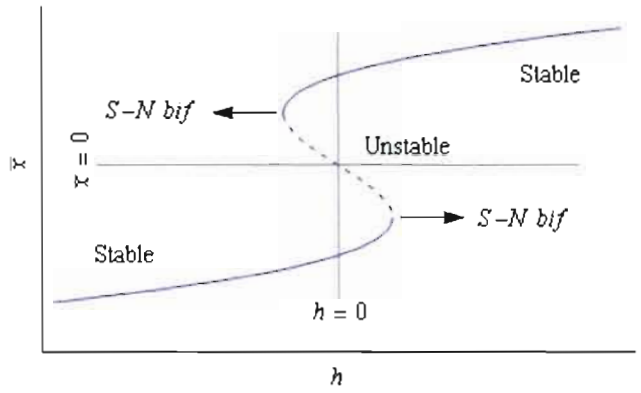


Figure 3.14 Stabilité des points fixes en fonction du paramètre d'asymétrie *h*, pour $r > 0$. Au centre, la branche du milieu est instable, les branches supérieure et inférieure sont stables; S-N bif représentent des bifurcations des points selles
(Chartier, Boukadoum et Amari, 2009)

Pour $h \neq 0$ et $r > 0$, il existe au moins un point fixe stable (fig. 3.14).

Si l'on prend $r = w = 1$:

- Pour $h < \frac{-2}{3\sqrt{3}}$: on obtient un point fixe stable négatif;
- Pour $h > \frac{2}{3\sqrt{3}}$: on obtient un point fixe stable positif;
- Pour $\frac{-2}{3\sqrt{3}} < h < \frac{2}{3\sqrt{3}}$: on obtient deux points fixes stables (un positif et un négatif);
- Pour $h = \pm\frac{2}{3\sqrt{3}}$: une bifurcation point selle apparaît et les deux points fixes entrent en conflit. Il s'ensuit une apparition d'un point fixe instable ou point selle (S-N bif pour *saddle node bifurcation*, voir fig. 3.14).

On peut ainsi conclure qu'une bifurcation point selle apparaît lorsque $h = \pm h_c(r)$, tel que (Chartier, Boukadoum et Amari, 2009) :

$$h_c(r) = \pm\frac{2r}{3}\sqrt{\frac{r}{3}} \qquad \cdots(3.16)$$

Ou d'une manière équivoque, lorsque $r = \pm r_c(h)$, tel que :

$$r_c(h) = \sqrt[3]{\frac{27h^2}{4}} \qquad \cdots(3.17)$$

D'après le diagramme de stabilité de la fig. 3.15, le nombre de points fixes stables varie entre un et deux, dépendamment des valeurs de h et r. Les bifurcations points selles apparaissent le long de la frontière de la région; Au niveau du point fronce $(h,r) = (0, 0)$, on peut observer une bifurcation de codimension 2.

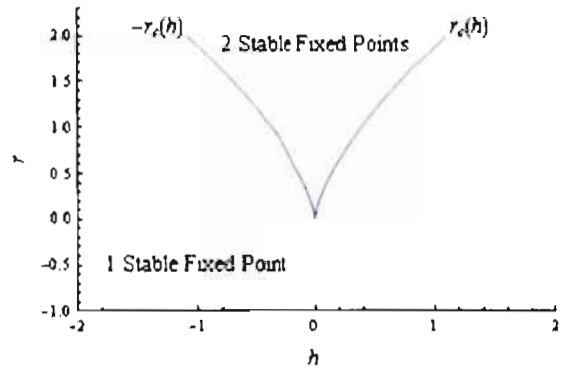


Figure 3.15 Diagramme de stabilité : dépendamment des valeurs de h et r. Le système représente un seul ou deux points fixes stables (Chartier, Boukadoum et Amari, 2009)

La fig. 3.15 peut résumer cette situation. En effet pour des valeurs positives de r, et dépendamment de la valeur du paramètre h, on obtient soit un point fixe stable positif, ou un point fixe stable négatif, ou bien deux points fixes stables, un positif et l'autre négatif.

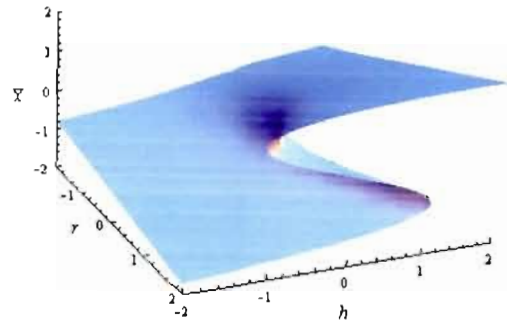


Figure 3.16 Paysage d'énergie avec une bifurcation fronce (Chartier, Boukadoum et Amari, 2009)

Un autre moyen pour illustrer l'impact qu'a le paramètre d'asymétrie h sur les points fixes est d'utiliser la fonction d'énergie (Eq.3.12) comme illustré sur les figures 3.16 et 3.17.

En effet, on remarque que lorsque $h = 0$, on obtient un double puits. Si de plus on suppose $\omega = r = 1$ et qu'on initialise x(0) avec une valeur négative. Après un temps t, $x(t)$ converge vers un point fixe $x = -1$, comme illustré par le cercle plein à gauche sur la Fig. 3.17a.
Si par contre, on prend une valeur de $h \neq 0$ (ex. $h = 0.3$), le rayon d'attraction du point attracteur droit devient plus grand que celui du point attracteur gauche sur la Fig. 3.17b. Le stimulus reste sur le point fixe attracteur de gauche tant que la valeur de h reste inférieure ou égale à $\frac{2}{3\sqrt{3}}$.

Si maintenant, on augmente encore la valeur du paramètre h (ex. $h = 0.6$), on obtient un point fixe attracteur prédominent (voir fig. 3.17c). Le stimulus est dans ce cas attiré par l'attracteur du côté droit et sera enfin stabilisé sur une valeur positive.

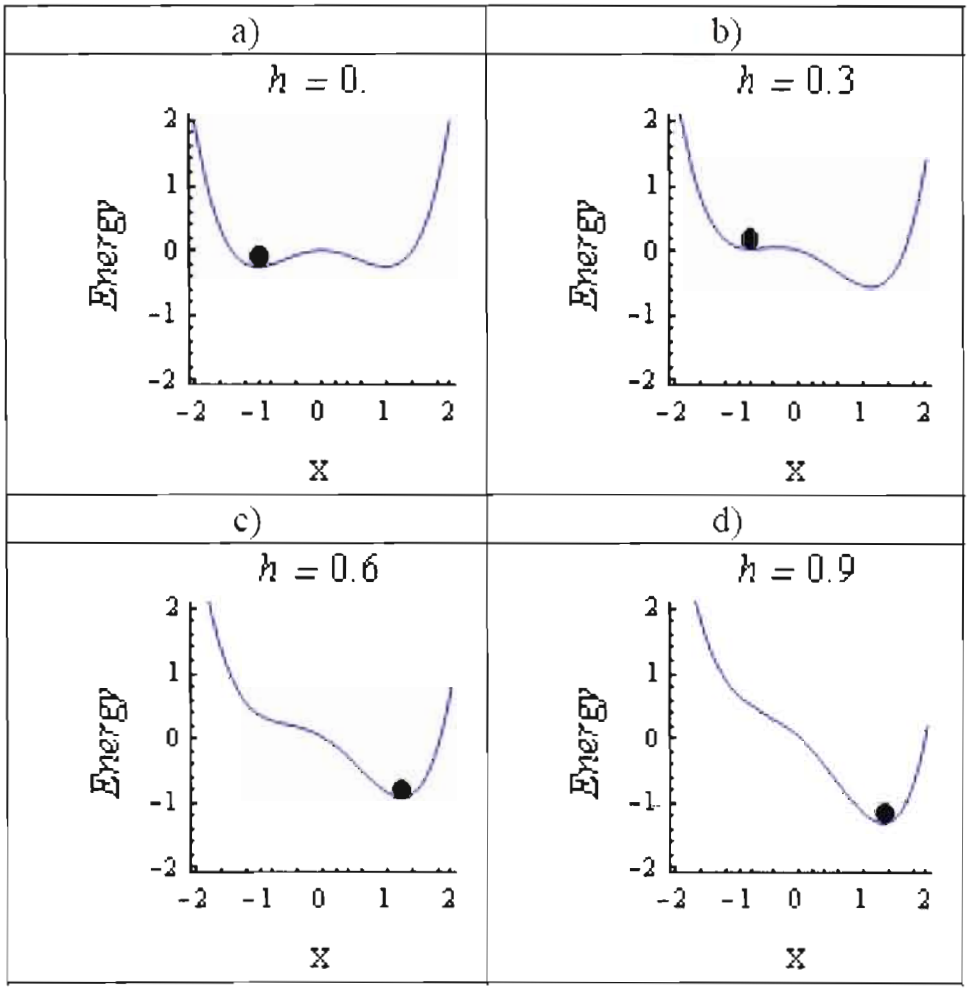


Figure 3.17 Modification du rayon d'attraction en fonction de différentes valeurs de h. (Chartier, Boukadoum et Amari, 2009)

L'effet du paramètre d'asymétrie h est donc de forcer un stimulus de converger d'un attracteur à un autre. Ce qui est comparable à désactiver momentanément un point fixe attracteur sans pour autant l'effacer de la mémoire du système.

La figure 3.17 montre que lorsque la valeur du paramètre d'asymétrie h augmente, le point fixe $x = 1$ se déplace vers des valeurs plus grandes. Si par exemple, $h = 0.9$ (fig. 3.17d), le point fixe devient $x = 1.3$. La fonction *signum* peut être, dans ce cas, utilisée afin de ramener la valeur à 1.

3.11.4 Cas d'un espace à *m* dimensions

On a illustré l'impact du paramètre d'asymétrie h sur les points fixes attracteurs. Dans le cas d'un espace *m*-dimensionnel, le même raisonnement peut être appliqué. On ne présentera dans ce mémoire que sommairement le cas 2-dimensions en s'appuyant sur un exemple particulier. Une étude théorique peut être consultée dans l'article (Chartier, Boukadoum et Amiri, 2009) cité en référence.

Dans le cas d'un espace à *m*-dimensions, l'Eq. 3.11 prend la forme vectorielle suivante (Chartier, Boukadoum et Amiri, 2009) :

$$\frac{dx}{dt} = f(x) = h + r * Wx - (Wx)^3 \qquad \cdots(3.18)$$

Tels que :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, W = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mm} \end{bmatrix}, h = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{bmatrix}, r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix}$$

Comme dans le cas d'un système à une dimension, les points fixes sont déterminés à partir des racines de l'Eq.3.18. La stabilité des points fixes est déterminée à partir de leur matrice Jacobienne.

Ex. : Si on considère dans un espace 2-dimensions :

$$W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; r = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \text{ et } h = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

On obtient neuf points fixes, comme illustré sur la fig. 3.18. Parmi ces neuf points fixes, seulement quatre d'entre eux sont stable: $(-1,-1),(-1,1),(1,-1)$ et $(1,1)$ illustrés par des cercles pleins. Les cinq points fixes restants sont instables et sont illustrés par des cercles vides.

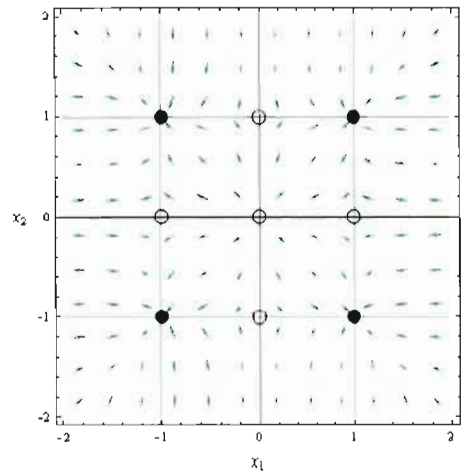


Figure 3.18 Portrait de phase d'un réseau à deux dimensions.
Les points fixes stables sont représentés par des cercles pleins
Et les points fixes instables par des cercles vides
(Chartier, Boukadoum et Amari, 2009)

La stabilité des points fixes pouvant être déterminée à partir de la fonction d'énergie du système. Dans le cas *m*-dimensions, sa forme est la suivante (Chartier, Boukadoum et Amari, 2009) :

$$E(x) = -h^T x - \frac{x^T (r * Wx)}{2} + \frac{x^T (Wx)^3}{4} \qquad \cdots(3.19)$$

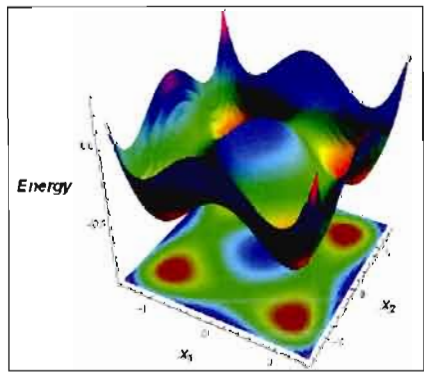


Figure 3.19 Surface d'énergie d'un system à deux dimensions
(Chartier, Boukadoum et Amari, 2009)

On peut remarquer sur la fig. 3.19, que les points fixes stables correspondent à des minima locaux sur le graphe représentant la surface d'énergie. L'espacc de rappel est ainsi partitionné en quatre puits

égaux. Par exemple, si la sortie désirée est $\mathbf{x} = [1, 1]^T$, la probabilité pour que ce point fixe attire un patron x à distribution uniforme est de 25%.

Lorsque la valeur de h augmente, elle fait en sorte de faire pivoter la surface d'énergie (Fig. 3.20). La rotation angulaire est fonction de la valeur du paramètre h.

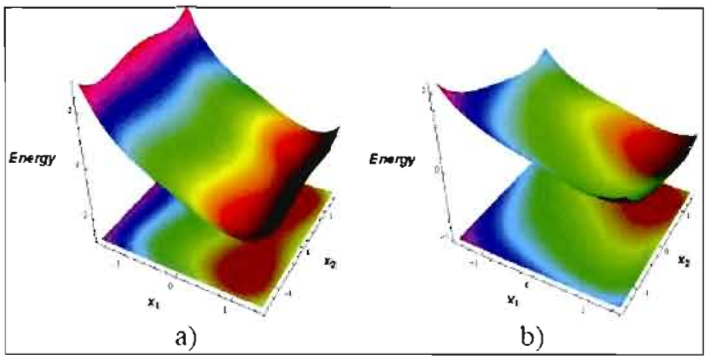


Figure 3.20 Surface d'énergie en fonction de la valeur du paramètre h
a) $\mathbf{h} = [1, 0]^T$; b) $\mathbf{h} = [1, 1]^T$ (Chartier, Boukadoum et Amari, 2009)

- Lorsque $h = \left[h_1 > \frac{2}{3\sqrt{3}}, 0 \right]^T$: x_1 sera positif (fig. 3.20a). Le système possède deux points fixes stables. La probabilité que le point fixe $\mathbf{x} = [1, 1]^T$ attire un stimulus aléatoire augmente de 50%.

  En pratique, dans le cas du modèle BAM, cela revient à dire que si une connaissance antérieure à la sortie désirée est disponible, celle-ci peut être transformée en un paramètre h qui fera augmenter la performance d'association du réseau.

- Lorsque $h = [0.8, 0.7]^T$: il y aura un unique attracteur dans le système et la probabilité que le point fixe $\mathbf{x} = [1, 1]^T$ attire une valeur aléatoire donnée sera de 100% (fig. 3.20b).

- Enfin, si l'on a une entrée $\mathbf{x}(0) = [0.2, 0.7]^T$, celle-ci est supposée converger vers le point fixe: $\mathbf{x} = [1, 1]^T$. Mais supposons que le premier élément de la sortie désirée ne doive pas être positif; il est possible d'introduire cette information dans le réseau en fixant le premier

élément du paramètre h à une valeur quelconque qui soit inférieure à $\frac{-2}{3\sqrt{3}}$ (ex. -0.7). Le résultat est que $\mathbf{x}(0)$ converge vers un autre attracteur, dans ce cas $\mathbf{x} = [-1, 1]^T$.

Par conséquent, le comportement du réseau vis-à-vis d'une entrée donnée est modifié dépendamment du contexte.

En résumé, le paramètre "***h***" peut être utilisé dans le but de générer différentes sorties à partir d'une entrée donnée. Son rôle est de biaiser l'espace de recherche vers les régions d'attraction désirées.

3.12 Conclusion

Nous avons exposé dans ce chapitre une étude détaillée, accompagnée d'exemples, de la dynamique interne du modèle BAM à fonction de sortie chaotique. Nous avons décelé que cette dynamique peut parfois présenter des états instables et des catastrophes de type fronce (*cusp*), à partir desquels la stabilité du modèle bifurque. On a aussi montré que le paramètre d'asymétrie ***h*** joue un rôle dans la stabilité et que, dépendamment de sa valeur, il peut créer une rupture et faire passer la dynamique du système d'un point stable à un autre point stable. L'utilisation d'un paramètre asymétrique au niveau de la fonction de sortie peut de ce fait, être un bon moyen pour parvenir à des comportements spécifiques de la BAM et guider le processus de rappel vers les états stables désirés. En particulier, elle peut pallier l'incapacité notoire des mémoires BAM à résoudre les problèmes de classification à séparabilité non linéaire.

Dans le chapitre qui suit, nous présentons les résultats obtenus lors de l'implémentation des différents mécanismes décrits plus haut. La première partie de la simulation est consacrée à la classification d'un problème non linéairement séparable, on prend le cas du OU-Exclusif (XOR). La seconde partie vise principalement à comparer des architectures hybrides basées sur le modèle BAM, par rapport à leurs performances de rappel et leurs robustesses en présence de prototypes bruités.

Chapitre 4

EXPÉRIMENTATIONS ET RÉSULTATS

Dans le présent chapitre, nous utilisons la version à apprentissage supervisé du modèle BAM présenté au chapitre 3, utilisant la même fonction de sortie (Eq. 3.7, 3.8). Afin de préserver la plausibilité biologique du modèle, nous conservons le même principe d'apprentissage hebb/anti-hebb utilisé par le modèle original. De plus, nous utilisons la même architecture (fig. 4.1) ainsi que la même règle d'apprentissage. La seule distinction concerne la fonction de sortie, dans laquelle on introduit un paramètre d'asymétrie ***h*** durant la phase de rappel (transmission).

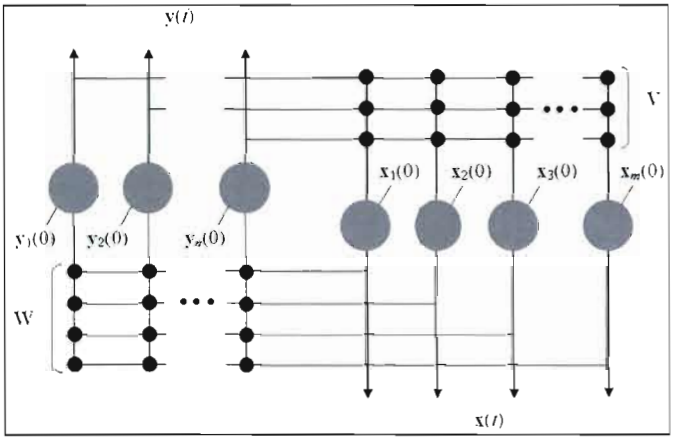


Figure 4.1 Modèle mémoire hétéro-associative bidirectionnelle

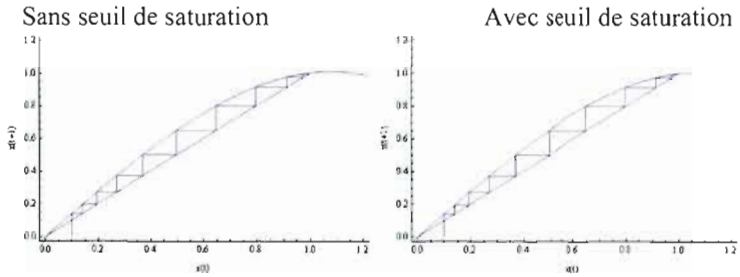
Afin d'assurer que les valeurs de sortie ne s'étendent pas en dehors de l'intervalle [-1, 1], on applique à la fonction de sortie deux seuils de saturation comme suit (Chartier et Boukadoum, 2006a) :

$$\forall i = 1,\ldots,N;\ y_i(t+1) = \begin{cases} 1, & si \quad Wx_i(t) > 1 \\ -1, & si \quad Wx_i(t) < -1 \\ (\delta+1)Wx_i(t) - \delta(Wx_i(t))^3, & \text{sinon} \end{cases} \quad \cdots(4.1)$$

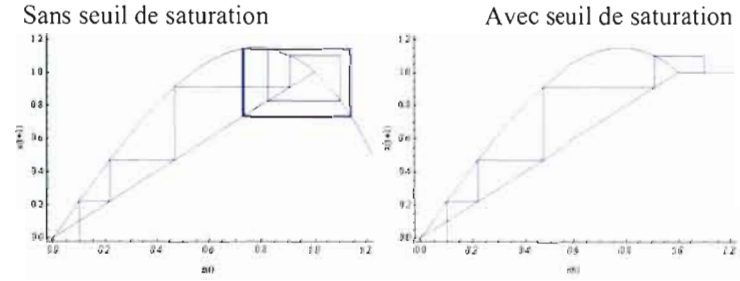
et

$$\forall i = 1,\ldots,M;\ x_i(t+1) = \begin{cases} 1, & si \quad Vy_i(t) > 1 \\ -1, & si \quad Vy_i(t) < -1 \\ (\delta+1)Vy_i(t) - \delta(Vy_i(t))^3, & \text{sinon} \end{cases} \quad \cdots(4.2)$$

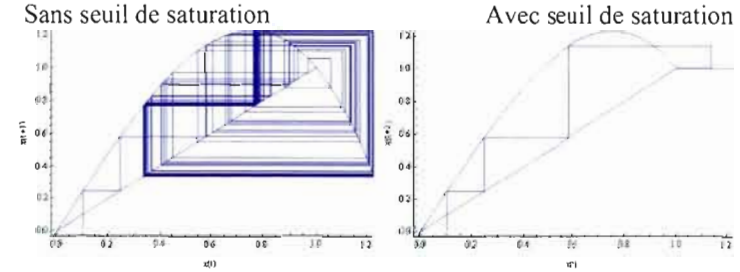
On a vu dans le chapitre 3, que la valeur du paramètre de transmission δ de la fonction de sortie est cruciale pour la performance du réseau : si δ est trop grand, le réseau peut converger aussi bien vers des attracteurs stables, cycliques ou apériodiques (chaotiques) (fig. 3.9, 3.10). Dès lors qu'on applique des seuils de saturation à la fonction de sortie, le modèle BAM ne présente plus de comportements chaotiques et devient un système à points fixes tel que montré sur les diagrammes de Cobweb ci-dessous (fig. 4.2).



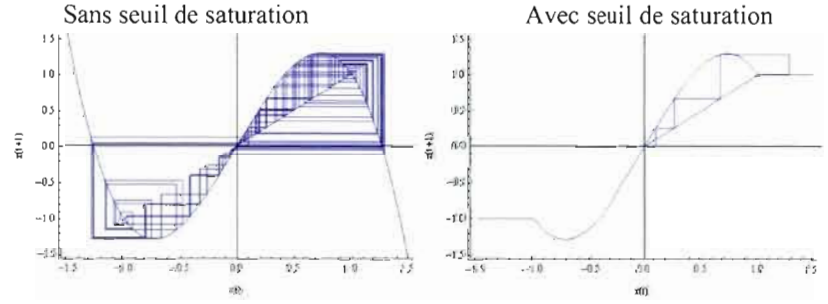
(a) Point fixe ($\delta = 0.4$)



(b) Point fixe ($\delta = 1.2$)



(c) Apériodique ($\delta = 1.45$)



(d) Apériodique ($\delta = 1.65$)

Figure 4.2 Diagrammes de Cobweb de la fonction de sortie
Sans limites de saturation vs. en présence de limites de saturation

Pour nos simulations, on utilise la fonction de sortie 4.1 et 4.2 durant la phase d'apprentissage de la BAM. Tandis que durant sa phase de rappel, on y introduit un nouveau paramètre "***h***" qui brise la symétrie. On obtient la fonction de sortie suivante durant la phase de rappel :

$$\forall i = 1,\ldots,N;\ y_i(t+1) = \begin{cases} 1+h, & si \quad Wx_i(t) > 1 \\ -1+h, & si \quad Wx_i(t) < -1 \\ (\delta+1)Wx_i(t) - \delta(Wx_i(t))^3 + h, & sinon \end{cases} \quad \cdots(4.3)$$

et

$$\forall i = 1,\ldots,M;\ x_i(t+1) = \begin{cases} 1+h, & si \quad Vy_i(t) > 1 \\ -1+h, & si \quad Vy_i(t) < -1 \\ (\delta+1)Vy_i(t) - \delta(Vy_i(t))^3 + h, & sinon \end{cases} \quad \cdots(4.4)$$

On montrera, à travers des expériences, le rôle du paramètre d'asymétrie "***h***" et son utilité dans l'amélioration de la performance de classification et de la robustesse de la BAM.

L'utilisation d'une fonction de sortie symétrique dans la phase de transmission, contraint la BAM à classifier uniquement des problèmes linéairement séparables. De ce fait, on tentera dans les tests qui vont suivre, d'introduire un biais dans la fonction de sortie, qui changera en fonction du patron d'entrée, afin de permettre à la BAM de converger vers un attracteur désiré. Un moyen simple pour y arriver est d'utiliser un apprentissage par renforcement basé sur un procédé essai/erreur, dans le but de corriger la BAM lorsqu'elle commet une erreur de classification, ou bien de renforcer son choix dans le cas contraire. Plus précisément, pour chaque stimulus d'entrée, on évalue la sortie et on biaise l'espace de recherche de la BAM dépendamment de la qualité de cette sortie. Biaiser l'espace de recherche se fera en introduisant un paramètre d'asymétrie ***"h"*** dans la fonction de sortie. Ce paramètre aura comme rôle de guider la BAM à converger vers les attracteurs désirés.

Dans ce chapitre, nous observons la contribution du paramètre ***"h"*** durant la phase de transmission pour la résolution des problèmes non linéairement séparables et nous proposons des méthodes pour déterminer sa valeur optimale. Dans la première partie, nous expérimentons nos procédés pour la classification du XOR qui est l'exemple typique de problème non linéairement séparable. Par la suite, dans la seconde partie, nous utilisons des architectures hybrides pour tenter d'améliorer la capacité de stockage de la BAM ainsi que sa performance de rappel. Finalement, on comparera les résultats

obtenus des différentes architectures hybrides avec ceux obtenus par le modèle BAM original qui lui, utilise une fonction de sortie symétrique durant la phase de rappel.

Nous donnons, dans ce qui suit, les expressions de certaines grandeurs relatives aux algorithmes évolutionnaires et dans le contexte des réseaux connexionnistes qui seront couramment mentionnées dans la suite de ce chapitre.

Erreur quadratique moyenne

L'erreur quadratique moyenne est calculée entre la sortie du réseau BAM observée et la sortie désirée. On a :

$$MSE = \sum_{p=1}^{P} \sum_{i=1}^{N} (t_i^p - o_i^p)^2 \qquad \cdots(4.5)$$

où :

P : taille de l'ensemble d'apprentissage;

N : nombre d'unités dans la couche de sortie.

t_i^p, o_i^p : la sortie désirée et la sortie observée sur le neurone i pour le parton p.

Fonction fitness

$$Fitness = \frac{1}{1 + MSE} \qquad \cdots(4.6)$$

Apprentissage BAM:

L'apprentissage du modèle BAM se résume aux étapes suivantes :

1. Initialiser les matrices de poids V et W (fig. 4.1) à 0 ;
2. Présenter chacune des 4 paires (entréc, sortie) (fig. 4.3) aléatoirement en suivant une distribution uniforme ;

3. Pour chaque paire, calculer la sortie de la BAM $x(1)$ et $y(1)$ selon les équations 4.1 et 4.2 (afin de limiter le temps de calcul, on exécute une seule itération (t = 1)) ;
4. Mettre à jour les poids de connexion selon l'équation 3.9 ;
5. Refaire les étapes 1 à 4 jusqu'à atteindre une condition d'arrêt (nombre d'itérations maximale ou erreur MSE minimale entre les sorties $y(0)$ et $y(1)$).

On fixe les différents paramètres come suit :

- Paramètre d'asymétrie h : 0 ;
- Paramètre de transmission apprentissage : 0.1;
- Paramètre de transmission rappel : 1,5;
- Paramètre d'apprentissage η : 0.001, en accord avec la condition 3.10 (valeur maximale: 0.017).

Une fois l'apprentissage de la BAM complété, on se sert dans la première partie de ce chapitre des algorithmes d'optimisation évolutionnaires (algorithmes génétiques, algorithmes de colonies de fourmis et l'optimisation par essaims particulaires) pour trouver le paramètre d'asymétrie adéquat ***"h"*** et ce, en utilisant un procédé essais/erreurs pour la classification du problème non séparable linéairement XOR. Par la suite, on se sert dans la seconde partie, de ces mêmes algorithmes pour la détermination de l'ensemble des poids de connexion des réseaux multicouches (RBF et MLP) utilisés dans les architectures hybrides. Le recours à ces techniques d'optimisation permet de garantir l'optimalité des solutions et de conserver la plausibilité biologique du modèle. La suite du présent chapitre présente donc plusieurs implémentations reliées aux différents mécanismes décrits ci-haut.

Toutes les simulations ont été réalisées en Matlab en utilisant des patrons bipolaires; les valeurs -1 et +1 ont été assignées aux pixels blancs et noirs respectivement. Le Tableau ci-dessous (Tableau 1) résume les réalisations effectuées.

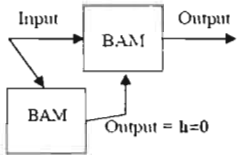
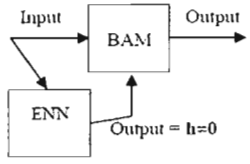
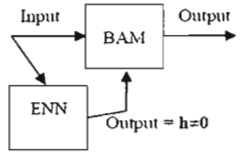
	Simulation n°1	Simulation n°2	Simulation n°3
Objectif	**Classification de problème non linéairement séparable : XOR**	**Classification de problème non linéairement séparable : XOR**	**Classification et amélioration de la performance vis-à-vis du bruit.**
Modèles utilisés	**BAM-BAM**	**BAM-ENN**	**BAM-ENN (ENN : RBF ou MLP)**
Architecture appliquées			
Patrons d'apprentissage utilisés	Bipolaires : 4 patrons d'entrée; 4 patrons de sortie. (Fig. 4.3)	Bipolaires : 4 patrons d'entrée; 4 patrons de sortie. (Fig. 4.3)	Bipolaires : 12 patrons d'entrée; 12 patrons de sortie. (Fig. 4.15)
BAM principale Apprentissage	BAM symétrique : h=0; Étape 1 : Hebb\anti-hebb (fonction OR); Étape 2 : Apprentissage par renforcement XOR (récompense "*h*" fournie par la BAM secondaire)	BAM symétrique : h=0; Hebb\anti-hebb.	BAM symétrique : h=0; Hebb\anti-hebb.
BAM secondaire Apprentissage	Hebb\anti-hebb : Stocker les valeurs du paramètre "*h*".	—	—
Réseau multicouche ENN Apprentissage	—	Optimisation $ACO_{\mathbb{R}}$, PSO ou AG	RBF : $ACO_{\mathbb{R}}$, PSO ou AG. MLP : PSO – BP, AG – BP ou $ACO_{\mathbb{R}}$ - BP.
BAM principale Transmission	Fonction de sortie asymétrique : $h \neq 0$; "*h*" fourni par la BAM secondaire.	Fonction de sortie asymétrique : $h \neq 0$; "*h*" fourni par ENN.	Fonction de sortie asymétrique : $h \neq 0$; "*h*" fourni par ENN (RBF ou MLP).
Fonction à optimiser	Minimisation de l'erreur MSE entre la sortie BAM et la sortie désirée	Minimisation de l'erreur MSE entre la sortie BAM et la sortie désirée	Minimisation de l'erreur de classification

Tableau 1. Récapitulatif des trois simulations réalisées avec Matlab

Partie 1

Problèmes non linéairement séparables

Apprentissage du OU-Exclusif

La fonction OU-Exclusif (XOR) est l'objet d'un test élémentaire largement utilisé dans les travaux portant sur le connexionnisme ; elle permet de vérifier si un réseau peut distinguer deux classes non linéairement séparables. La disponibilité de données de comparaison dans la littérature nous permet d'évaluer les performances obtenues dans nos simulations par rapport à l'état de l'art.

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	1/0
		OR/XOR

Figure 4.3 Fonction OR/XOR

Simulation 1

4.1 Réseau hybride BAM-BAM et apprentissage par renforcement

Dans cette simulation, nous utilisons une architecture comprenant deux BAMs dans laquelle une BAM primaire réalise une tâche de classification non linéaire guidée par une BAM secondaire qui biaise son comportement (fig. 4.4). La BAM secondaire est entraînée de façon à produire le bon paramètre d'asymétrie pour la fonction de sortie de la BAM principale. Tel que mentionné au Chapitre 3, ce procédé vise à forcer la BAM principale à converger vers le bon attracteur en exploitant les propriétés d'une catastrophe fronce de sa dynamique interne.

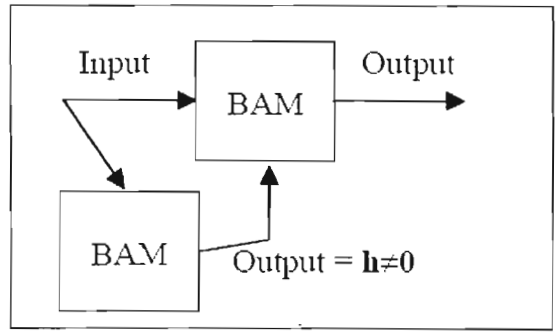


Figure 4.4 Une BAM secondaire fournit le paramètre asymétrique *h* à la BAM principale

Étape 1 : Apprentissage supervisé d'un problème linéairement séparable (fonction OR)

La BAM utilise un apprentissage supervisé pour stocker des partons linéairement séparables. Pour ce faire, on choisit la fonction OR (fig. 4.3).

Étape 2 : Apprentissage par renforcement XOR

Une fois l'apprentissage du problème linéairement séparable (OR) terminé, on utilise un apprentissage par renforcement, basé sur un procédé essai/erreur, afin de forcer la BAM à converger vers le bon attracteur, lorsque celle-ci fait des erreurs. Pour le procédé essai/erreur, on propose un apprentissage par renforcement basé sur des algorithmes évolutionnaires. Les algorithmes évolutionnaires serviront à déterminer le paramètre "***h***" optimal qui va réduire l'erreur entre la sortie de la BAM et la sortie désirée. Pour chaque patron d'entrée et à chaque itération, l'algorithme recherche le meilleur individu

parmi une population d'individus "***h***". L'individu désigné représente une des meilleures solutions relativement à la minimisation de l'erreur quadratique moyenne (MSE) calculée entre la sortie de la BAM principale et la sortie désirée. La fonction fitness dans notre cas étant choisie inversement proportionnelle à l'erreur, minimiser l'erreur revient donc à maximiser cette fonction fitness (Eq. 4.6).

Recherche du paramètre "***h***" optimal par optimisation évolutionnaire :

On a expérimenté le procédé de recherche du paramètre optimal avec trois différents algorithmes d'optimisation de type évolutionnaires, l'algorithme génétique (AG), l'optimisation par essaims de particules (PSO) et l'algorithme de colonies de fourmis (ACO).

Pour chacun des quatre patrons de la fig. 4.3, faire :

1. Initialiser une population P(t) d'individus "***h***" (chromosomes (AG), particules (PSO) ou fourmis (ACO));
2. Répéter les étapes suivantes jusqu'à atteindre une condition d'arrêt (nombre d'itérations maximal ou erreur MSE minimale entre la sortie BAM principale et la sortie désirée).
   a. Évaluer la fonction fitness de chaque individu (inversement proportionnelle à l'erreur MSE) entre la sortie de la BAM et la sortie désirée;
   b. Appliquer les variations/mises à jour relatives à chaque algorithme :
      - AG : Opérations de sélection, croisement et mutation sur les individus "***h***" afin de générer une nouvelle population P'(t);
      - PSO : Déplacement de chaque particule "***h***" de l'essaim dans l'espace d'états en fonction de la meilleure solution trouvée et la meilleure solution de l'essaim;
      - ACO : Mise à jours des traînées de phéromones et déplacement des fourmis "***h***" vers les chemins à forte intensité;

Tant que la condition d'arrêt n'est pas atteinte (erreur minimale entre la sortie-BAM et la sortie désirée ou nombre maximum de générations), on réitère l'algorithme pour améliorer la qualité des solutions (paramètres "***h***"). Une fois le paramètre "***h***" trouvé pour un patron d'entrée donné, il est stocké dans la BAM secondaire afin d'être réutilisé dans la phase de rappel de la BAM principale.

Étape 3 : Phase de rappel

Durant la phase de rappel, la BAM secondaire, ayant préalablement appris les paires (Entrée, "***h***"), va pouvoir fournir pour chaque entrée le paramètre stocké correspondant. On présente dans la section qui suit, les résultats obtenus à la sortie de la BAM principale ainsi que la solution "***h***" optimale trouvée par chacun des trois algorithmes d'optimisation utilisés AG, PSO et ACO et ce, pour chacun des stimuli d'entrée de la figure 4.2.

Simulation 1.1 - BAM-BAM$_{GA}$:

D'après l'Eq. 4.6, l'expression de la fonction fitness est :

$$Fitness = \frac{1}{1 + MSE}$$

Au cours des générations, on approche de plus en plus de la solution optimale (paramètre "***h***" optimal). En effet, on remarque sur la fig. 4.5 que la valeur fitness de la meilleure solution est une fonction croissante au cours des générations. Au moment où l'algorithme converge (dans notre cas ~ 50 générations), la valeur de la fonction fitness est très proche de 1. À ce moment-là, on considère la meilleure solution de la dernière génération comme étant la valeur optimale du paramètre ***h*** recherché. La solution trouvée pour le paramètre ***h*** sera par la suite utilisée dans la fonction de sortie chaotique de la BAM.
La recherche du paramètre ***h*** est effectuée pour chacun des stimuli de l'ensemble d'apprentissage (fig. 4.5).

1. Étapes de l'AG et choix des opérateurs :

- Générer aléatoirement une population de paramètres "***h***";
- Calculer la valeur fitness inversement proportionnelle à la MSE entre la sortie de la BAM et la sortie désirée, et pour chaque individu (ou chromosome) "***h***" de la population;
- Opérateur de sélection : Roulette biaisée;
- Croisement : En deux points;
- Mutation : Effectuer occasionnellement un inversement de bits, choisi aléatoirement.

2. Paramètres de l'algorithme AG choisis :

Taille de la population : 400 individus;
Probabilité de croisement (pCrossover) : 0.9;
Probabilité de mutation (pMut) : 0.1.

3. Résultats obtenus BAM-BAM$_{AG}$:

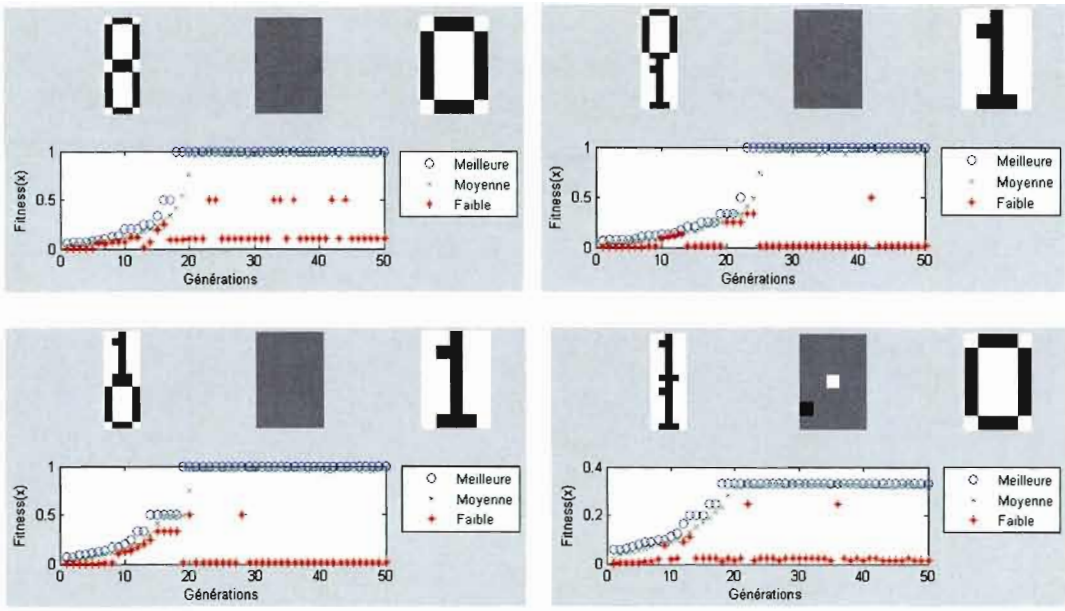


Figure 4.5 Prototype d'entrée, paramètre h et sortie de la BAM principale
Évolution génétique de la détermination du *h* avec optimisation GA.

Rem. : le vecteur "***h***", lorsqu'il n'intervient pas dans la fonction de sortie est un vecteur nul (cas des prototypes d'entrée (0, 0), (0, 1) et (1, 0)). Une composante nulle du vecteur "***h***" est représentée par un pixel gris (fig. 4.5).

Résultats :

- Le nombre de générations atteint avant convergence de l'algorithme : 50
- Les résultats obtenus sur la fig. 4.5 illustrent le principe du procédé essais/erreurs appliqué lors de l'apprentissage par renforcement. En effet, la BAM ayant préalablement appris la fonction OR (linéairement séparable), elle n'a pas besoin de se faire corriger pour les prototypes (0, 0), (0, 1) et (1, 0), puisqu'elle converge vers les bons attracteurs. Par contre pour le prototype (1, 1), des opérations essais/erreurs sont appliquées afin de forcer la BAM à converger vers l'attracteur 0 plutôt que l'attracteur 1. Le rôle de l'AG est de trouver le paramètre ***h*** optimal qui permette de créer une catastrophe fronce (cusp) et faire basculer la stabilité de la BAM du point fixe 1 vers le point fixe 0.

Simulation 1.2 - BAM-BAM$_{PSO}$:

De la même manière que précédemment, on applique une architecture BAM-BAM. Mais pour la détermination du paramètre optimal de la fonction de sortie, on opte cette fois-ci pour un algorithme d'optimisation combinatoire PSO, appliqué dans un espace binaire.

1. On rappelle les équations régissant la position et le mouvement des particules de l'algorithme PSO (Eq. 2.13, 2.14, 2.15 et 2.16) dans un espace de recherche discret et qui sont définies par :

- $v_i^{t+1} = \omega\, v_i^t + r_1 c_b * (p_i - x_i^t) + r_2 c_g * (s - x_i^t)$
- $x_i^{t+1} = x_i^{t+1} + v_i^{t+1}$
- $v_i^t = \dfrac{1}{1+\exp(-v_i^t)}$: Représente la probabilité de déplacement vers la position 1
- $v_i^t = \begin{cases} v_i^t = v_{Max} & si\ v_i^t \geq v_{Max} \\ v_i^t = v_{Min} & si\ v_i^t \leq v_{Min} \\ v_i^t & \text{sinon} \end{cases}$

2. Les paramètres de l'algorithme PSO utilisés pour la simulation :

Taille de la population : 100 particules;

c_b : Coefficient d'apprentissage cognitif (local) : varie de 2.5 à 0.5;

c_g : Coefficient d'apprentissage social (global) : varie de 0.5 à 2.5;

ω, facteur d'inertie : varie de 0.9 à 0.4;

V_{Max}, Vitesse Maximale des particules : 10.

3. Résultats obtenus BAM-BAM$_{PSO}$:

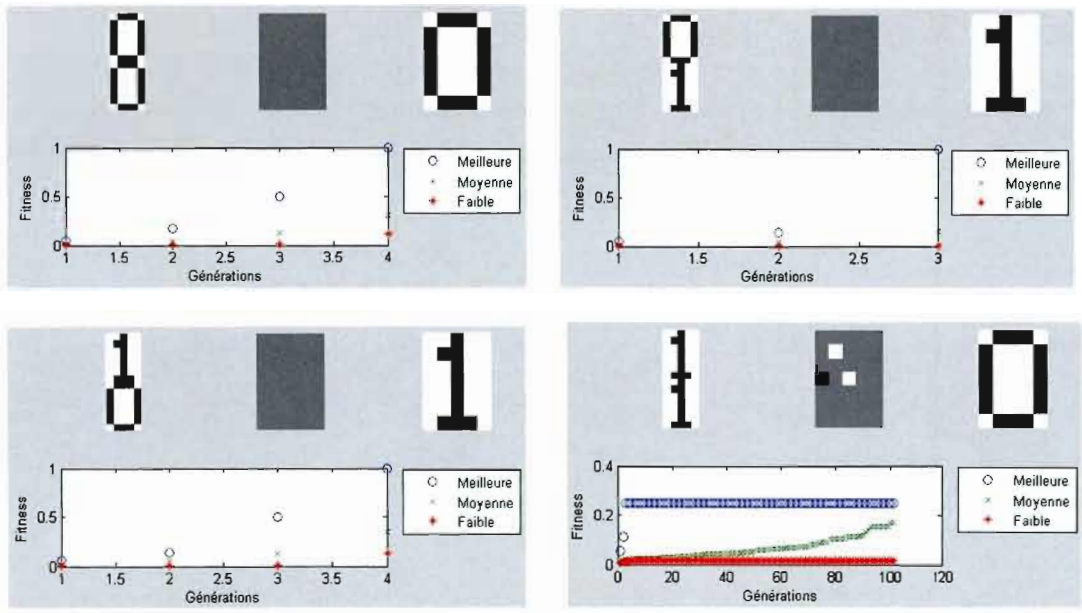


Figure 4.6 Prototype d'entée, paramètre *h* et sortie de la BAM principale
Évolution par essaims particulaire de la détermination du *h* avec optimisation PSO

Résultats :

- L'algorithme de recherche du paramètre "***h***" optimal converge très rapidement pour les prototypes (0, 0), (0, 1) et (1, 0). Par contre, il nécessite 100 itérations pour trouver l'optimal qui minimise l'erreur MSE entre la sortie de la BAM principale et la sortie désirée.
- Les attracteurs reliés aux trois premiers prototypes correspondent bien aux réponses apprises lors de l'apprentissage de la fonction OR, on n'a donc pas besoin de biaiser la recherche de solution pour ces prototypes, ce qui explique une solution "***h***" nulle. Par contre, pour le prototype (1, 1), plutôt que de converger vers la solution apprise 1, l'algorithme PSO va trouver le paramètre "***h***" qui va générer une dynamique de catastrophe fronce afin de faire basculer l'état d'équilibre de la BAM du point fixe 1 au point fixe désiré, 0.

Simulation 1.3 - BAM-BAM$_{ACO}$:

Dans cette simulation, on utilise l'algorithme MMAS (MAX-MIN Ant System) conjointement avec une méthode de recherche locale (Stutzle et Hoos, 2000) pour la détermination du paramètre asymétrique de la fonction de sortie de la BAM.

Recherche locale utilisée :

1. Avant la fin de chaque itération (génération), on prend la meilleure solution "***h***" et on lui applique un inversement sur 2 bits choisis aléatoirement (2-flips) c.-à.-d. : $0 \rightarrow \pm 1, +1 \rightarrow (0 \text{ ou } -1), -1 \rightarrow (0 \text{ ou } +1)$;
2. On calcule la valeur fitness de la solution modifiée;
3. Si la valeur fitness de la solution résultante est meilleure, on garde la nouvelle solution en remplaçant l'originale, sinon on conserve celle-ci.
4. On exécute le procédé N fois (dans notre cas N=100) à la fin de chaque itération.

Dans cet algorithme, on a opté pour un choix hybride entre la meilleure solution globale et la meilleure solution locale (itération en cours) pour mettre à jour les traînées de phéromones. Seule la meilleure des deux solutions effectue cette mise à jour.

1. Pour appliquer l'algorithme ACO combinatoire, on utilise les équations Eq. (2.6) et Eq. (2.7) pour la mise à jour de la traînée de phéromones. On utilise également l'Eq. (2.2) pour le calcule de la probabilité de mouvement des fourmis :

- $\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{best}$
- $\Delta\tau_{ij}^{best} = Q.f(s^{best})$
- $P_{ij}^{k}(t) = \dfrac{[\tau_{ij}(t)]^{\alpha}}{\sum_{k}[\tau_{ik}(t)]^{\alpha}}$ (Probabilité de mouvement vers 0 ou 1)

2. Paramètres de l'algorithme ACO combinatoire choisis :

Nombre de fourmis : 100;

ρ , degré d'évaporation des phéromones : 0.5;

C, intensité initiale des phéromones = 0.001;

Q, constante de l'algorithme : 1.

3. Résultats obtenus "BAM-BAM$_{ACO}$" :

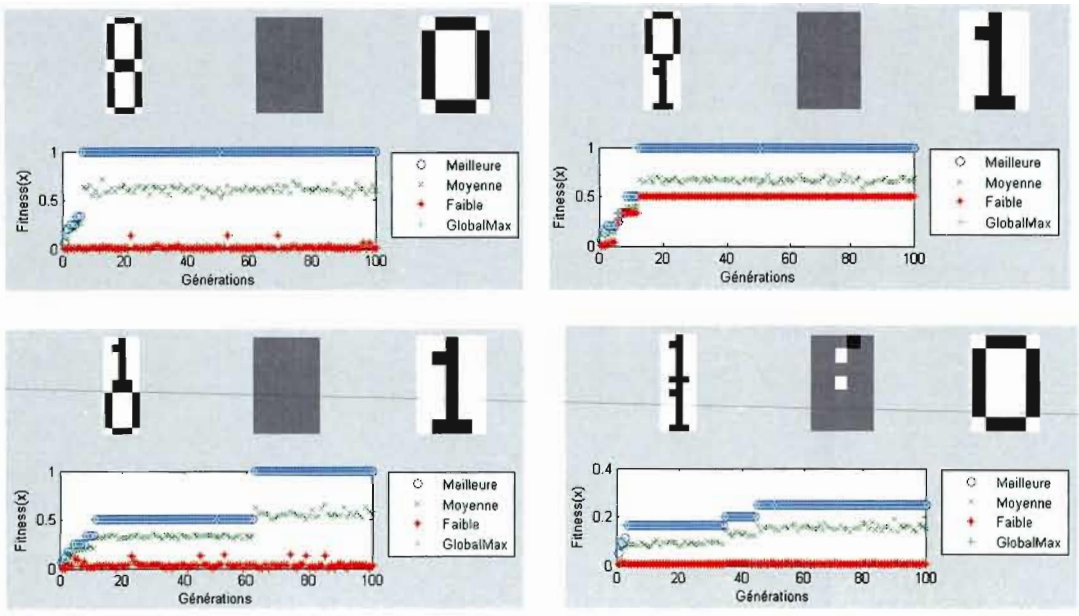


Figure 4.7 Prototype d'entée, paramètre *h* et sortie de la BAM principale
Évolution par colonie de fourmis de la détermination du *h* avec optimisation ACO

Résultat:

- L'utilisation de l'algorithme ACO combinatoire donne des résultats similaires aux résultats précédemment obtenus avec les algorithmes AG et PSO.
- L'algorithme converge vers la solution "***h***" optimal après 100 générations et pour les 4 stimuli.

Conclusion 1

Le recours à un procédé d'apprentissage par renforcement et l'utilisation d'une BAM secondaire a permis à la BAM principale de classifier correctement un problème non linéairement séparable (fonction XOR). Les algorithmes évolutionnaires AG, PSO et ACO jouent un rôle important, celui de trouver le paramètre "h" optimal permettant de faire basculer l'état de la BAM d'un point fixe à un autre, dès que celle-ci converge vers un mauvais attracteur.

Dans les trois simulations précédentes, le procédé de recherche du paramètre "h" optimal a été effectué sur plusieurs itérations et ce, en minimisant l'erreur MSE entre la sortie BAM principale et la sortie désirée. On peut conclure en disant que les trois méthodes ont donné des résultats très concluants puisque la BAM a réussit dans les trois cas à effectuer la tâche de classification du problème XOR, ce qu'elle ne pouvait pas réaliser avec une fonction de sortie symétrique. Aussi, on remarque que des trois algorithmes d'optimisation, l'algorithme PSO converge plus rapidement vers une solution sous optimale que l'AG et le ACO. L'algorithme ACO étant le plus long (100 générations).

Simulation 2

4. 2 Réseau hybride BAM-ENN et apprentissage par renforcement

Nous cherchons aussi dans cette simulation à déterminer le paramètre "***h***" adéquat pour la fonction de sortie, afin d'effectuer correctement la classification de la fonction XOR, mais au lieu d'utiliser une BAM secondaire qui stocke une valeur du paramètre d'asymétrie pour chaque patron d'entrée, on place un réseau de neurones non bouclé (*feedforward*) entrainé par un algorithme d'optimisation évolutionnaire et placé parallèlement à la BAM, qu'on notera ENN (pour *evolutionary neural network*). La sortie de cc réseau ENN servira de paramètre "***h***" pour configurer la fonction de sortie de la mémoire BAM. Donc, le réseau ENN devra fournir pour chaque stimulus d'entrée (fig. 4.5), le paramètre "***h***" adéquat afin classifier correctement le problème XOR.

On considère un réseau de neurones non bouclé (fig. 4.8) à une seule couche cachée. L'apprentissage de ce réseau ENN se fait non pas par des méthodes classiques comme celle de la rétro-propagation d'erreur, mais plutôt en utilisant un algorithme d'optimisation évolutionnaire (AG, PSO ou ACO). Ces derniers auront comme tâche de repérer la combinaison de poids optimale qui minimise l'erreur MSE entre la sortie de la BAM et la sortie désirée. De la même manière que précédemment (modèle BAM-BAM), ce processus nécessite l'apprentissage préalable d'un problème linéairement séparable (ex. fonction OR).

Choix du codage

Afin de limiter le nombre d'opérations (déjà coûteux) effectuées lors de l'apprentissage, nous avons choisi de représenter les poids des connexions du réseau non bouclé par des valeurs réelles. On va donc réutiliser les mêmes algorithmes évolutionnaires utilisés dans la première simulation (GA, PSO et ACO) sauf que, dans ce cas-ci, ils opéreront dans des espaces de recherche continus.

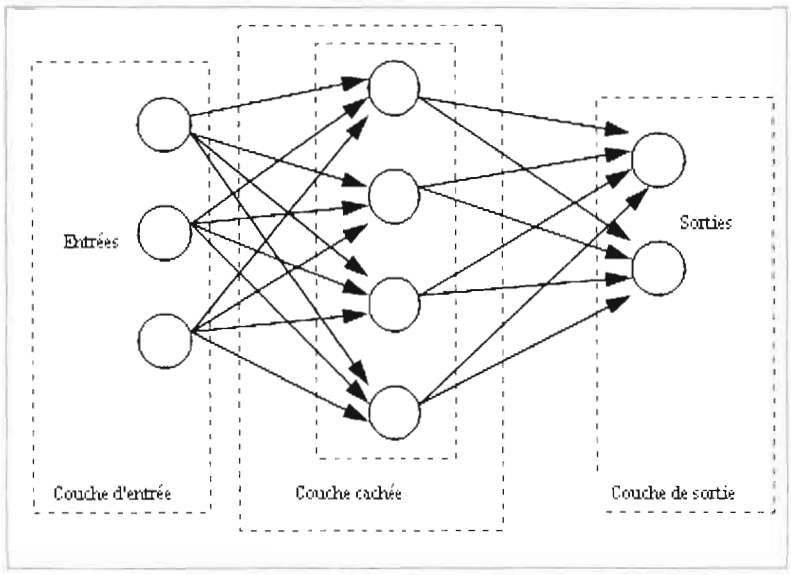


Figure 4.8 Réseau non-bouclé évolutionnaire à une couche cachée (ENN)

Dans cette simulation, le réseau multicouche ENN résout un problème d'optimisation consistant à minimiser l'erreur MSE entre la sortie de la BAM et la sortie désirée. Pour chaque prototype d'entrée, la sortie de la BAM est comparée avec le prototype de sortie, et l'algorithme évolutionnaire fait en sorte de trouver la meilleure combinaison de poids en minimisant cette erreur. Étant donné que la fonction fitness est inversement proportionnelle à la valeur de l'erreur MSE (Eq. 4.6), minimiser l'erreur revient à maximiser la fonction fitness. Les trois algorithmes d'optimisation AG, PSO et $ACO_{\mathbb{R}}$ ont été testés, et les résultats sont présentés ci-dessous.

Simulation 2.1 - Réseau BAM-ENN$_{AG}$:

Dans cette partie, un algorithme génétique est utilisé afin d'optimiser les poids des connexions du réseau ENN. Le problème à optimiser consiste à maximiser la fonction fitness qui prend des valeurs inversement proportionnelles à l'erreur calculée à la sortie de la BAM.

1. Paramètres de l'algorithme AG choisis :

- Nombre de neurones d'entrée (dimension de l'espace d'entrée) pour 2 patrons superposés : 70;
- Nombre de neurones de sortie (dimension de l'espace de recherche) : 35;
- Nombre de neurones cachés : 10;
- Nombre maximum de générations : $2x10^4$;
- Taille de la population : 1200 chromosomes;
- Sélection :
  - Élitisme;
  - Roulette biaisée pour les 1000 premières itérations;
  - Sélection par rang pour le reste des itérations.
- Pourcentage de croisement : 0.7;
- Pourcentage de mutation : 0.3.
- La fonction de sortie du réseau ENN est une tangente hyperbolique (Eq 1.2).

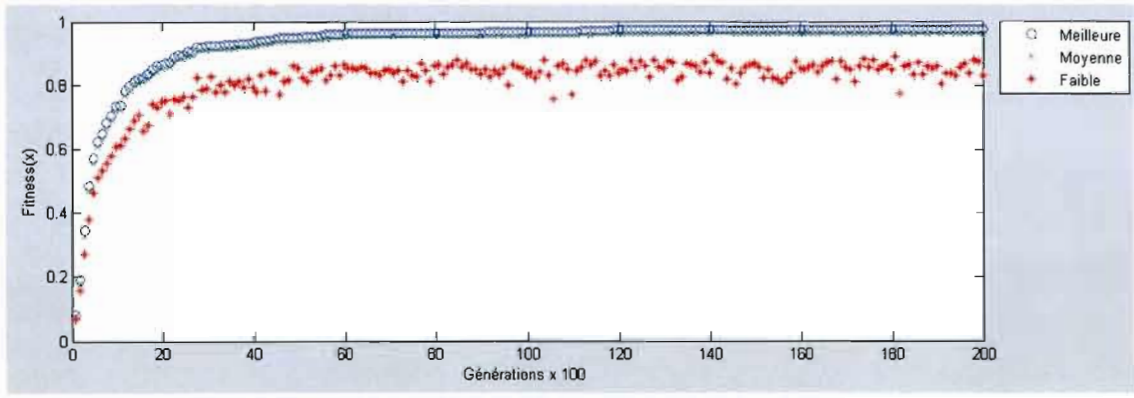


Figure 4.9 Fonction fitness lors de l'apprentissage de la fonction XOR du réseau ENN$_{AG}$

D'après la figure 4.8, l'apprentissage des poids de connexion du réseau ENN avec l'algorithme AG, converge après $2x10^4$ générations.

Une fois l'apprentissage du réseau ENN terminé, on présente chacun des stimuli de la figure 4.4 à l'entrée du réseau ENN. La sortie de ce dernier est par la suite fournie à la BAM comme paramètre d'asymétrie pour sa fonction de sortie. Les résultats illustrés sur la figure 4.9 montrent l'avantage d'utiliser un paramètre d'asymétrie dans la fonction de sortie du modèle BAM pour la classification de problèmes non linéairement séparables. On obtient de ce fait, un modèle de BAM plausible d'un point de vu biologique et offrant des capacités de classification supérieures, particulièrement pour les tâches non séparables linéairement.

2. Résultats obtenus BAM-ENN$_{AG}$:

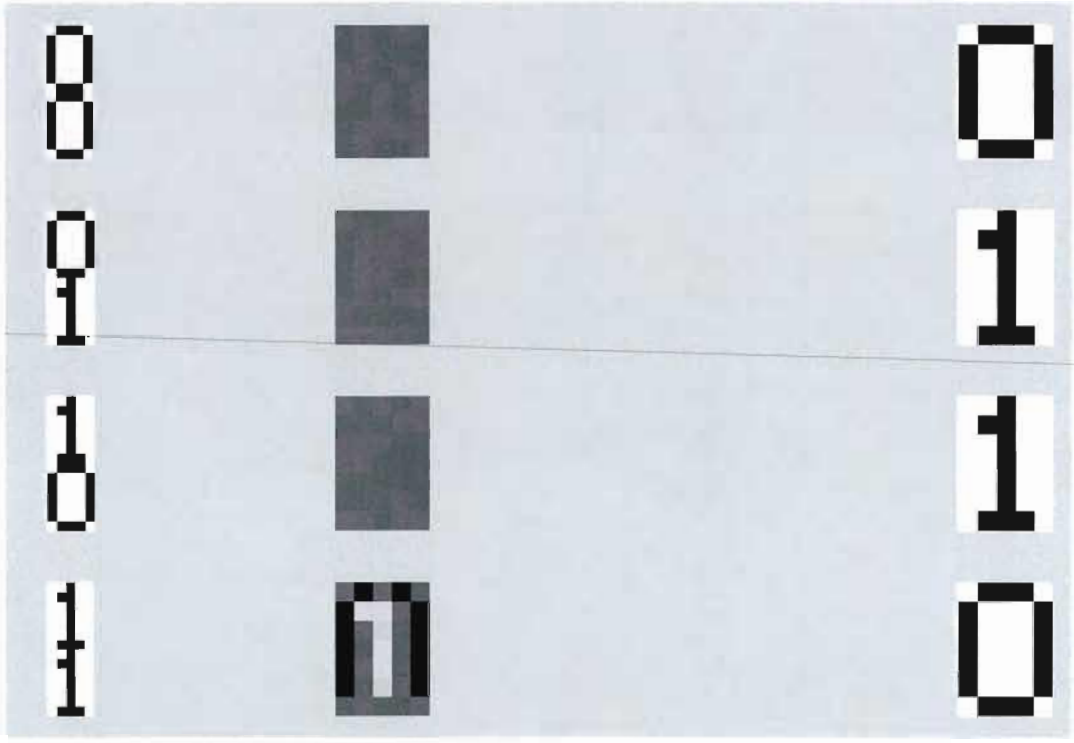


Figure 4.10 Prototypes d'entrée, paramètre d'asymétrie *h* fourni ENN entraîné par GA, Sortie de la BAM

Les patrons de gauche sur la figure 4.9, illustrent les stimuli d'entrée de la BAM (fig. 4.5), et ceux du milieu le paramètre "***h***" trouvé par l'algorithme AG pour chacun de ces stimuli. Les patrons à droite représentent la sortie de la BAM relativement à chaque prototype.

Résultats :

- L'architecture hybride proposée BAM-ENN$_{AG}$, a permis à la BAM d'effectuer correctement la tâche de classification du problème non linéairement séparable XOR.
- Un inconvénient pour ce modèle est la lenteur d'apprentissage du réseau ENN. En effet, lors de notre simulation, la recherche du paramètre "$\boldsymbol{h}$" a nécessité $2x10^4$ générations (fig. 4.9).

Simulation 2.2 - Réseau BAM-ENN$_{PSO}$:

Dans cette partie, on utilise un processus identique à celui de la précédente simulation (BAM-ENN$_{AG}$), sauf que dans celle-ci, l'apprentissage du réseau ENN est effectué en utilisant un algorithme d'optimisation par essaims particulaires PSO.

1. Paramètres de l'algorithme PSO choisis :

- Nombre de neurones d'entrée (dimension de l'espace d'entrée) pour 2 patrons superposés : 70;
- Nombre de neurones de sortie (dimension de l'espace de recherche) : 35;
- Nombre de neurones cachés : 10;
- Nombre maximum de générations : 10^5;
- Nombre de particules dans la population : 60;
- Paramètre d'inertie : 0.729;
- Coefficient d'apprentissage cognitif : 1.496;
- Coefficient d'apprentissage social : 1.496;
- $v_{Max} = 1$;
- La fonction de sortie du réseau ENN est une tangente hyperbolique (Eq. 1.2).

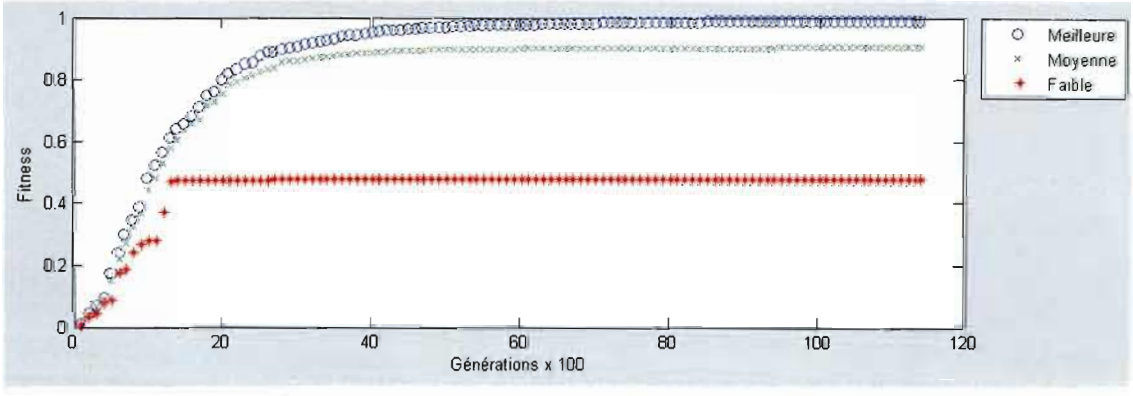


Figure 4.11 Fonction fitness lors de l'apprentissage de la fonction XOR du réseau ENN$_{PSO}$

L'apprentissage du réseau ENN avec l'algorithme PSO converge après 1.2×10^4 générations.

2. Résultats obtenus BAM-ENN_{PSO}

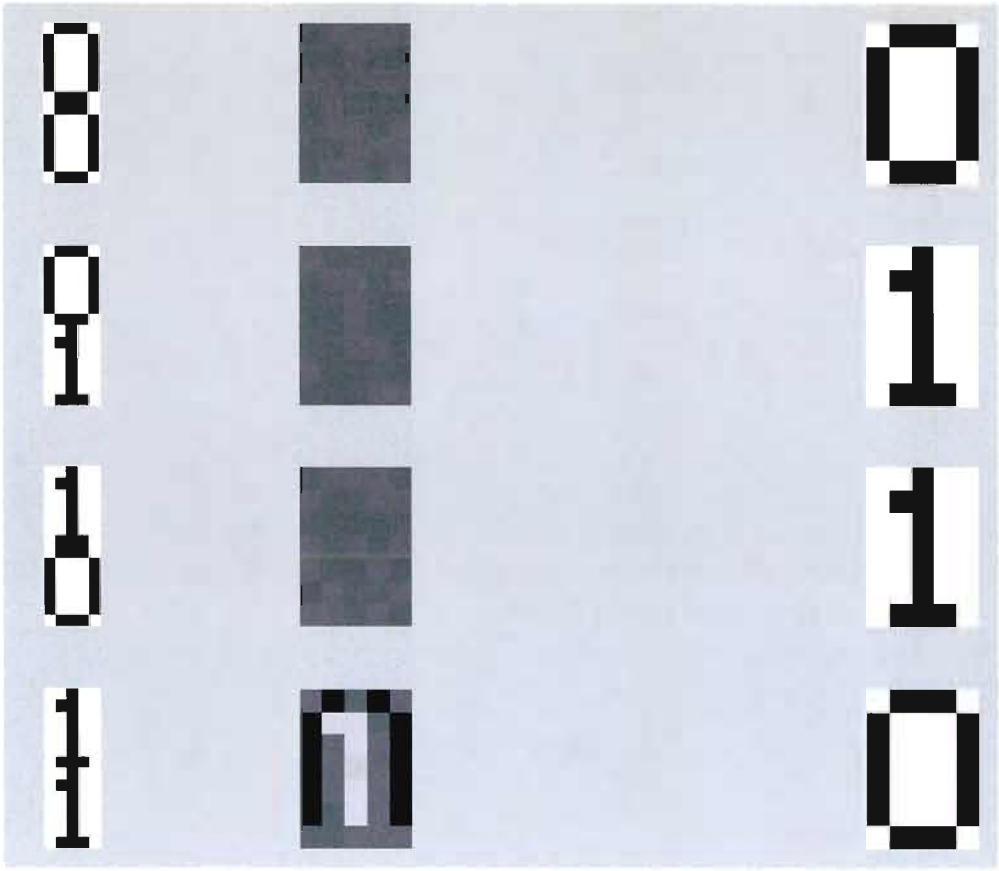


Figure 4.12 Prototypes d'entrée, paramètre d'asymétrie *h* fourni ENN entraîné par PSO Sortie de la BAM

Résultats :

- L'architecture hybride BAM-ENN_{PSO} constitue une autre approche permettant à la BAM d'effectuer correctement la classification du problème non linéairement séparable, XOR.
- L'apprentissage du réseau ENN a convergé après 1.2 $x10^4$ générations (fig. 4.11). Cette approche est donc moins coûteuse en temps de calcul que BAM-ENN_{AG}, tout en offrant des performances similaires.

Simulation 2.3 - Réseau BAM-ENN $ACO_{\Re}$

On expérimente dans cette partie le même modèle BAM-ENN en adoptant un apprentissage de l'ENN basé sur l'optimisation par colonie de fourmis.

Étant donné que le problème à optimiser comprend des valeurs continues (poids de connexion), on utilise un algorithme ACO adapté pour les espaces continus, $ACO_{\Re}$.

1. Paramètres de l'algorithme $ACO_{\Re}$ choisis :

- Nombre de neurones d'entrée (dimension de l'espace d'entrée) pour 2 patrons superposés : 70;
- Nombre de neurones de sortie (dimension de l'espace de recherche) : 35;
- Nombre de neurones cachés : 10;
- Nombre maximum de générations : 10^4;
- Nombre de fourmis : 200;
- Paramètre ξ, pour le calcul de l'écart type (Eq. 2.12) : 0.65;
- Constante q, pour le calcul des poids des fonctions gaussiennes (Eq. 2.10) : 0.01;
- La fonction de sortie du réseau ENN est une tangente hyperbolique (Eq. 1.2).

« ENN: 10 neurones cachés »

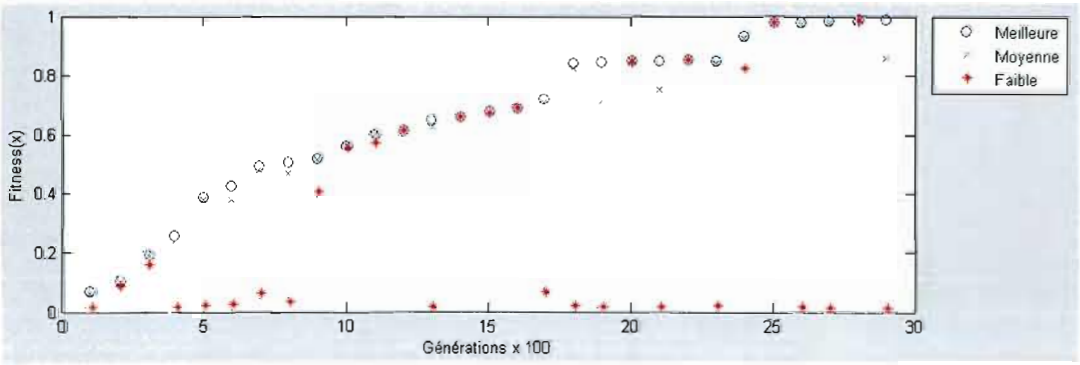


Figure 4.13 Fonction fitness lors de l'apprentissage de la fonction XOR du réseau ENN $ACO_{\Re}$

L'apprentissage du réseau ENN avec l'algorithme $ACO_{\Re}$ converge après 3 x10^3 générations.

2. Résultats obtenus BAM-ENN $_{ACO_{\Re}}$

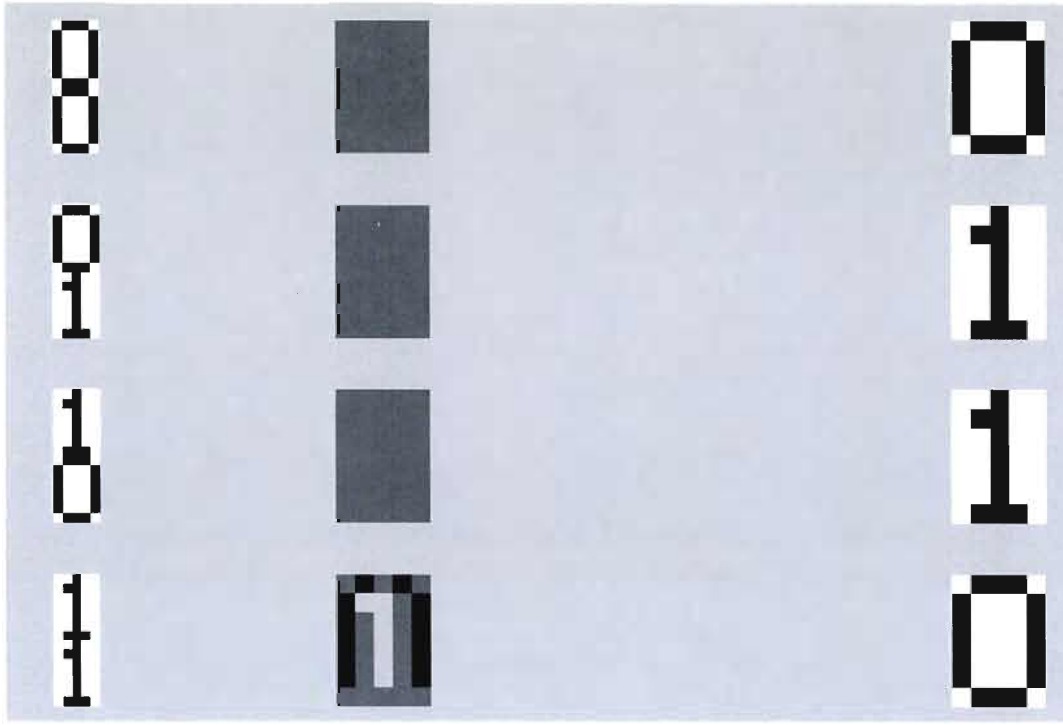


Figure 4.14 Prototypes d'entrée, paramètre d'asymétrie *h* fourni ENN entraîné par ACO $_{\Re}$, Sortie de la BAM

Résultats :

- Le modèle BAM-ENN$_{ACO_{\Re}}$ offre des résultats similaires pour la classification du problème XOR.
- L'apprentissage dans le modèle BAM-ENN$_{ACO_{\Re}}$ semble plus rapide. En effet, l'apprentissage du réseau ENN s'effectue au bout de 3 x10^3 (Fig. 4.13) générations, contrairement à 1.2x10^4 (Fig. 4.11) pour le PSO et 2 x10^4 pour l'AG (Fig. 4.9).

Conclusion 2

Les résultats obtenus lors des précédentes simulations, nous font constater l'avantage de combiner une dynamique de catastrophe fronce pour la BAM avec un procédé d'apprentissage par essais/erreurs basé sur un réseau évolutionnaire. En effet, avec une telle architecture hybride, la BAM réussit à réaliser parfaitement la tâche de classification du problème non séparable linéairement XOR, ce qui est impossible à réaliser avec une BAM à fonction de sortie symétrique. On constate également l'avantage d'utiliser des algorithmes évolutionnaires pour la recherche du paramètre "h" optimal pour la fonction de sortie.

On conclut dans cette première partie que le fait d'introduire un paramètre "h" adéquat dans la fonction de sortie lors du rappel, constitue une façon relativement simple pour faire ignorer à la BAM certains attracteurs de l'espace d'états. Les attracteurs désirés bénéficient ainsi d'une augmentation de leurs rayons d'attraction, permettant à la BAM une plus grande probabilité de convergence.

On remarque également, pour un même stimulus d'entrée, que les sorties du réseau ENN par rapport aux 3 algorithmes GA, PSO et $ACO_{\Re}$, sont très similaires, ce qui montre leurs propriétés de recherche globale de la solution dans l'espace d'états. En effet, ils convergent tous, après quelques itérations, dans le même voisinage de la solution optimale. Enfin, des trois algorithmes évolutionnaires utilisés pour l'apprentissage du modèle ENN, l'$ACO_{\Re}$ est celui qui converge plus rapidement vers la solution sous optimale.

Le réseau multicouche de type ENN placé seul n'est pas considéré comme un bon classificateur. En effet, il est capable d'approcher une solution sans jamais l'atteindre, raison pour laquelle il est souvent utilisé avec un autre classificateur à recherche locale. Expérimentalement, on a pu constater que le réseau ENN est en mesure d'exécuter par lui-même la tâche non séparable linéairement XOR (sans la BAM), sauf que souvent, les résultats à sa sortie n'approchent pas d'aussi près les patrons désirés que lorsqu'il est utilisé conjointement avec la BAM. C'est ce qui justifie souvent le recours à des architectures hybrides (BAM-ENN) lorsqu'il s'agit de ce type de réseau.

Partie 2

Amélioration de la performance face au bruit

Simulation 3

Dans cette partie, nous expérimentons l'apport des architectures hybrides et des algorithmes évolutionnaires dans l'amélioration des performances de rappel de la BAM et de sa tolérance vis-à-vis du bruit. L'architecture hybride utilisée est semblable à celle utilisée dans la simulation 2, elle consiste à placer un réseau multicouche ENN parallèlement à la BAM et entrainé par un algorithme d'optimisation évolutionnaire (AG, $ACO_{\Re}$ ou PSO). Sauf que dans cette partie, le problème à résoudre par les algorithmes évolutionnaires n'est plus la minimisation de l'erreur à la sortie de la BAM, mais plutôt l'optimisation de la tâche de classification.

Le modèle hybride BAM-ENN utilisé pour cette simulation nécessite une première étape d'apprentissage consistant à entrainer les deux modèles séparément, d'un côté, la BAM avec la règle d'apprentissage hebb/anti-hebb (Eq. 3.9), et de l'autre côté le réseau ENN avec un algorithme d'optimisation évolutionnaire (AG, $ACO_{\Re}$ ou PSO). Une fois l'apprentissage terminé, les deux réseaux sont combinés pour effectuer simultanément une tâche de classification donnée. Par la suite, dans la phase de rappel, un stimulus quelconque est présenté à l'entrée du modèle hybride. Le réseau ENN le traite en premier et fournit la sortie correspondante à la BAM, qui de son côté l'incorpore comme paramètre d'asymétrie "$\boldsymbol{h}$" dans sa fonction de sortie. Le prototype d'entrée est par la suite traité par la BAM pour être classifié.

Dans ce modèle hybride, le rôle du réseau multicouche ENN est de participer au processus de rappel de la BAM en lui fournissant le paramètre "$\boldsymbol{h}$" adéquat et ce, dans le but de biaiser la recherche vers les régions de meilleures solutions. L'ENN permet ainsi aux points fixes de la BAM de disposer d'un rayon d'attraction plus important, ce qui permet d'augmenter la robustesse vis-à-vis du bruit et ainsi augmenter la capacité de stockage. Nous verrons, à travers les résultats des simulations, l'avantage que peut apporter un tel procédé pour l'amélioration de la performance de rappel.

Plusieurs modèles hybrides ont été testés, parmi eux, le modèle BAM-RBF combinant une BAM avec un réseau RBF à fonctions gaussiennes et un autre modèle, BAM-MLP combinant la BAM avec un perceptron multicouche.

La première partie de cette simulation expérimente le réseau hybride constitué d'un modèle RBF placé en parallèle à la BAM et entrainé en utilisant un algorithme évolutionnaire ($ACO_{\Re}$ ou PSO). Dans la seconde partie, on utilise un réseau MLP pour lequel l'apprentissage s'effectue en deux phases. Une première phase consiste à trouver une solution globale à l'ensemble des poids de connexion en utilisant un apprentissage évolutionnaire (AG, $ACO_{\Re}$ et PSO) et une seconde phase qui consiste à utiliser cette solution globale comme ensemble de poids initiaux pour effectuer un apprentissage local par rétro-propagation d'erreur. Ce dernier modèle (BAM-MLP) n'est pas plausible biologiquement, mais peut être utilisé dans une perspective d'ingénierie. De plus, il est utilisé comme référence dans ce travail afin de comparer les modèles BAM original (à fonction de sortie symétrique) et BAM-RBF.

Patrons utilisés pour l'apprentissage :

La fig. 4.15 présente les 12 prototypes sélectionnés pour l'apprentissage. Chaque prototype est constitué d'une matrice de 5 x 7 Pixels représentant une lettre de l'alphabet. Ces prototypes constituent des points dans un espace 35-dimensionnel et leur niveau de corrélation varie entre 0.03 à 0.83.

Notre modèle BAM (fig. 4.1) est constitué de 35 unités d'entrée, 35 unités de sortie et stocke 12 prototypes. Sa charge mnésique est donc de 34% (12 prototypes pour 35 neurones).

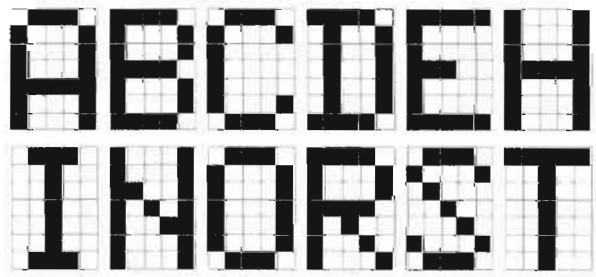


Figure 4.15 Patrons utilisés pour l'apprentissage

Les valeurs -1 et +1 sont assignées aux pixels blancs et noirs respectivement (fig. 4.15).

Apprentissage BAM:

Les étapes d'apprentissage du modèle BAM se résument comme suit :

1. Initialiser les matrices V et W à 0 (fig. 4.1);
2. Présenter chacune des 12 paires [entrée, sortie] aléatoirement en suivant une distribution uniforme;
3. Pour chaque paire, calculer les sorties de la BAM $x(1)$ et $y(1)$ selon les équations Eq. 4.1 et 4.2. Afin de limiter le temps de calcul, itérer une seule fois (t = 1);
4. Mettre à jour les poids de connexion selon les équations (Eq. 3.9);
5. Répéter les étapes 1 à 4 jusqu'à atteindre une condition d'arrêt (nombre d'itérations maximale ou erreur MSE minimale entre les sorties $y(0)$ et $y(1)$.

Dans la phase d'apprentissage de la BAM, on fixe le paramètre d'apprentissage η à 0.001 et le paramètre "h" à 0 (fonction de sortie symétrique).

Déroulement des tests

Durant nos prochaines simulations, on mesure le niveau de bruit par le pourcentage des pixels qui ont été inversés (flippés) de -1 à 1 et vice versa. Par exemple, un niveau de bruit de 20% signifie que 7 sur les 35 valeurs de pixels ont été aléatoirement inversés. De plus, pour chaque pourcentage de bruit, on génère aléatoirement 500 prototypes bruités et on détermine la performance de rappel dans chaque cas.

4.3 Modèle hybride BAM-RBF

- Le modèle RBF utilise des fonctions à base radiale de type Gaussienne (Eq. 1.4). On prend les 12 prototypes d'apprentissage (fig. 4.15) comme centres de ces fonctions :

$$g_i(x) = \exp\left(-\frac{\|x - c_i\|^2}{2\sigma^2}\right)$$

  Tel que c^i représente le $i^{ième}$ prototype.

- Les écarts-type sont pris tous égaux (Eq. 1.5) et leur valeur est fixée à la moyenne des distances minimales de chaque unité cachée par rapport aux autres unités :

$$\sigma = \left\langle \|c_i - c_j\| \right\rangle;$$

  Tel que c_j représente le centre le plus proche de c_i et $\langle\ \rangle$ constitue la moyenne de cette distance à travers toutes les paires.

Une fois les paramètres du réseau RBF fixés, on exécute la phase d'apprentissage de ce dernier en utilisant un algorithme évolutionnaire AG, PSO ou $ACO_{\mathbb{R}}$. Le problème à optimiser par ces derniers étant la minimisation de l'erreur quadratique moyenne MSE entre le prototype à apprendre et la sortie du réseau ENN.

Les trois algorithmes d'optimisation AG, PSO et $ACO_{\mathbb{R}}$ ont été testés pour l'apprentissage du RBF et les résultats de rappel obtenus du modèle hybride BAM-RBF sont présentés ci-après.

A. Réseau hybride BAM-RBF $_{ACO_{\Re}}$:

Dans cette expérience, la règle d'apprentissage du réseau RBF est l'algorithme d'optimisation par colonie de fourmis $ACO_{\Re}$.

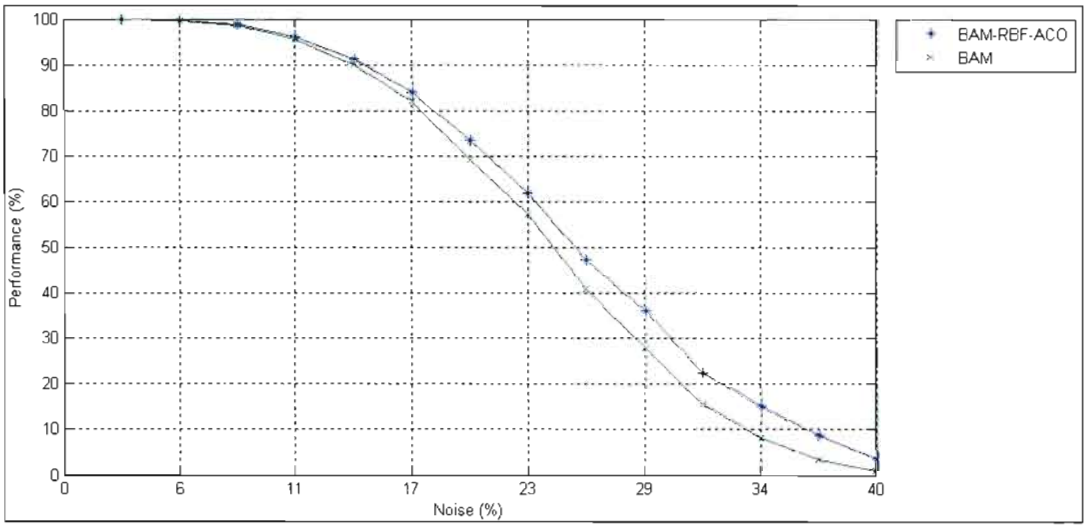


Figure 4.16 Performance de classification en fonction du nombre de pixels flippés. Modèle BAM standard vs. BAM-RBF $_{ACO_{\Re}}$

Résultat BAM-RBF $_{ACO_{\Re}}$

D'après la fig. 4.16, on remarque que la différence de performance entre le modèle hybride BAM-RBF entrainé par $ACO_{\Re}$ et le modèle BAM atteint jusqu'à 10% pour des prototypes bruités à un niveau de bruit moyen à élevé. On a bien un regain de robustesse comparativement au modèle BAM utilisant une fonction de sortie symétrique dans la phase de transmission.

B. Réseau hybride BAM-RBF $_{PSO}$:

Une deuxième expérience est effectuée en utilisant un réseau RBF entrainé par un algorithme d'apprentissage par optimisation PSO.

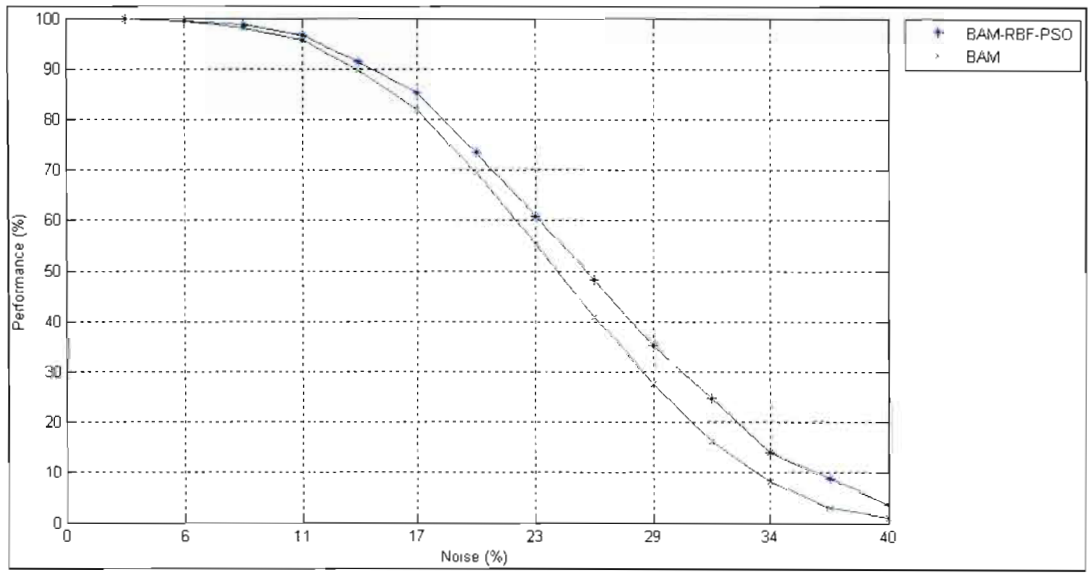


Figure 4.17 Performance de classification en fonction du nombre de pixels flippés. Modèle BAM standard vs. BAM-RBF $_{PSO}$

Résultat BAM-RBF $_{PSO}$

Des performances similaires à BAM-RBF $_{AG}$ sont observées lorsque le réseau RBF est entrainé avec un algorithme d'optimisation PSO.

4.4 Modèle hybride BAM-MLP

Il est connu que les méthodes de descente de gradient sont assujetties à des variations de performances dues à la configuration initiale des poids de connexion, menant parfois à une convergence vers des minima locaux. Les méthodes évolutionnaires assurent à l'opposé une recherche dans le domaine global. En effet, au fur et à mesure des générations, l'espace de recherche est affiné vers des sous-espaces de solutions potentiellement optimales. Cependant, il est courant pour ces algorithmes AE de trouver une solution qui soit proche de la solution optimale sans jamais l'atteindre. D'où l'intérêt d'hybrider des méthodes évolutionnaires avec des méthodes de recherche locale; les deux sont complémentaires (Belew et al. 90).

Nous étudions dans cette partie les capacités du système à s'adapter aux changements de conditions, à l'aide d'une approche globale (par algorithmes évolutionnaires) et d'une approche locale (par rétro-propagation du gradient), en vue de trouver un système de classification et de reconnaissance optimal. Dans cette simulation, on choisit un réseau de neurones multicouche à une seule couche cachée, contenant 70 unités à fonction de sortie tangentielle hyperbolique. On effectue l'apprentissage de ce réseau MLP en deux phases. La première phase consiste à trouver la configuration de poids optimale globale en utilisant un des algorithmes évolutionnaires (AG, $ACO_{\mathbb{R}}$ ou PSO). La deuxième consiste à utiliser cette solution globale pour un apprentissage par rétro-propagation de l'erreur BP (*backpropagation*). Voir fig. 4.24 dans le cas de l'ACO.

1. Paramètres du réseau MLP et paramètres d'apprentissage :

- Nombre de neurones d'entée : 35;
- Nombre de neurones de sortie : 35;
- Nombre de neurones cachés : 70;
- Taux de momentum : 0.1
- Taux d'apprentissage, $\mu : 0.025$;
- Erreur minimale MSE : 10^{-5};
- La fonction de sortie du réseau MLP est une tangente hyperbolique (Eq. 1.2);
- Apprentissage en ligne (poids ajustés pour chaque patron).

Chacune des figures 4.18, 4.20 et 4.22 illustrent les performances des 3 modèles de classification suivants :

- Une BAM seule à fonction de sortie symétrique, **BAM**,
- Un modèle hybride : une BAM et un réseau MLP_{BP}, entrainé par méthode de rétro-propagation de l'erreur, lui fournissant le paramètre "***h***", **BP-BAM**
- Un modèle hybride : une BAM et un réseau $MLP_{ENN\text{-}BP}$, entrainé par un algorithme évolutionnaire suivi d'un algorithme de rétro-propagation de l'erreur, fournissant le paramètre "***h***", **ENN-BP-BAM**.

A. Réseau hybride BAM-MLP $_{AG-BP}$:

MLP $_{AG}$ « 70 neurones cachés »

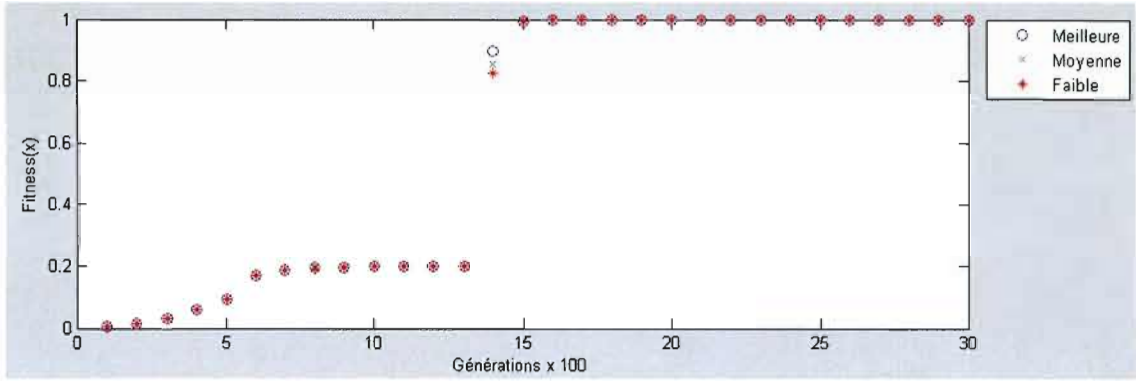


Figure 4.18 Fonction fitness dans la phase d'apprentissage du réseau MLP par optimisation **AG**

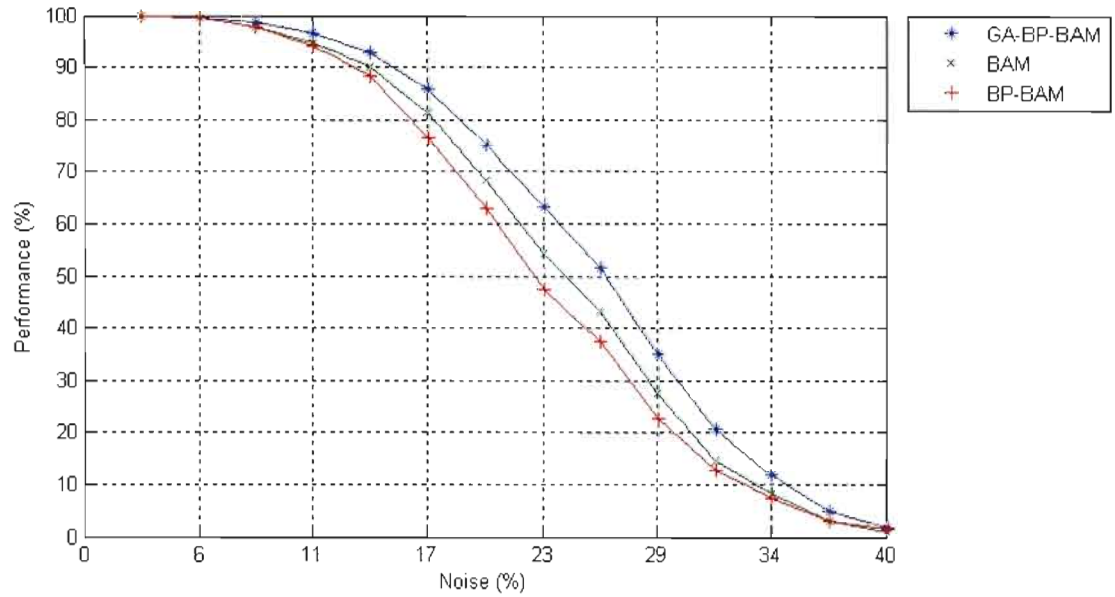


Figure 4.19 Performance de classification en fonction du nombre de pixels flippés.
Modèle BAM standard (vert) vs. BAM-MLP $_{BP}$ (rouge) vs. BAM-MLP $_{AG-BP}$ (bleu)

B. Réseau hybride BAM-MLP $_{PSO-BP}$:

MLP $_{PSO}$ « 70 neurones cachés »

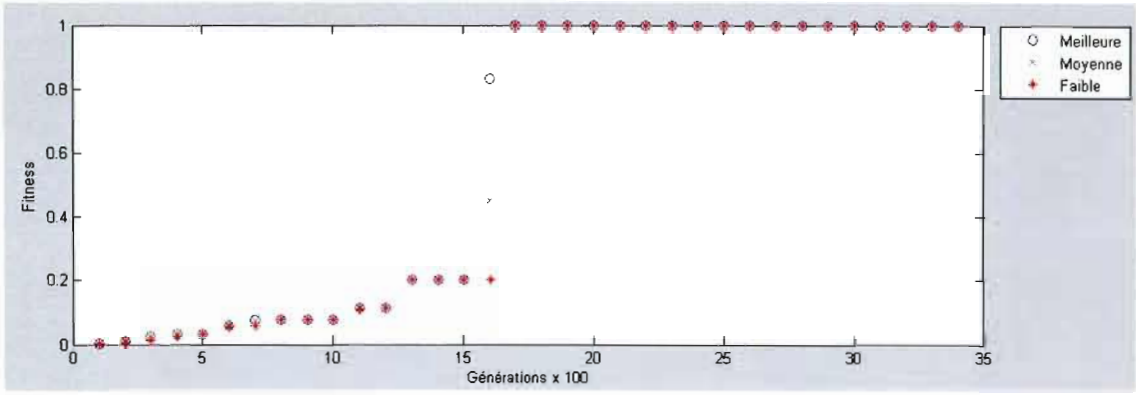


Figure 4.20 Fonction fitness lors de l'apprentissage du réseau MLP par optimisation PSO

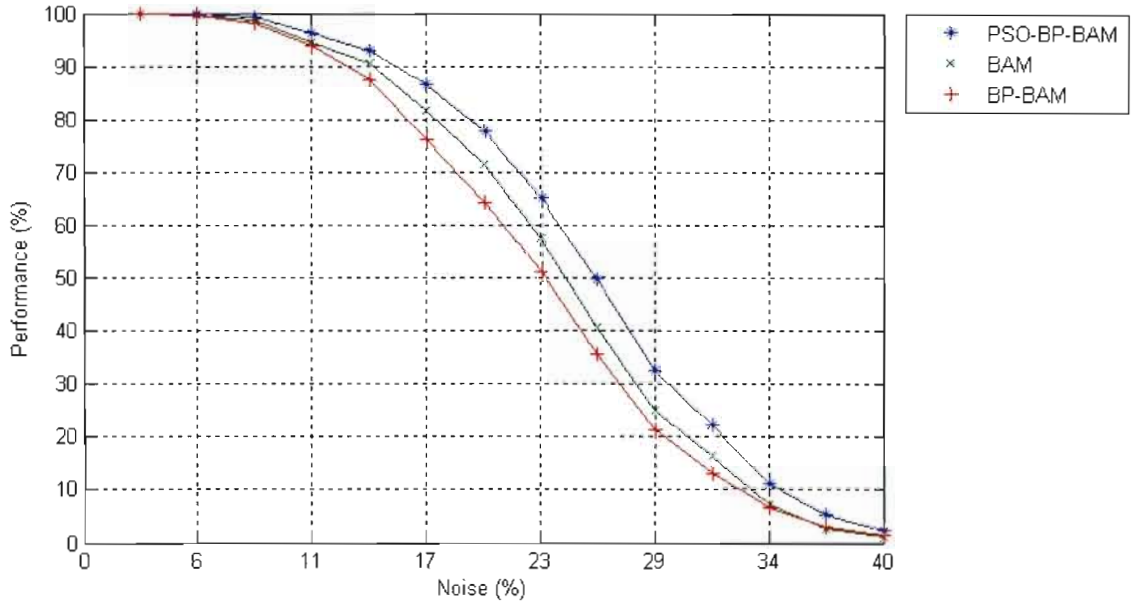


Figure 4.21 Performance de classification en fonction du nombre de pixels flippés.
Modèle BAM (vert) standard vs. BAM-MLP $_{BP}$ (rouge) vs. BAM-MLP $_{PSO-BP}$ (bleu)

C. <u>Réseau hybride BAM-MLP</u> $_{ACO_{\mathfrak{R}}-BP}$ **:**

MLP $_{ACO_{\mathfrak{R}}}$ « 70 neurones cachés »

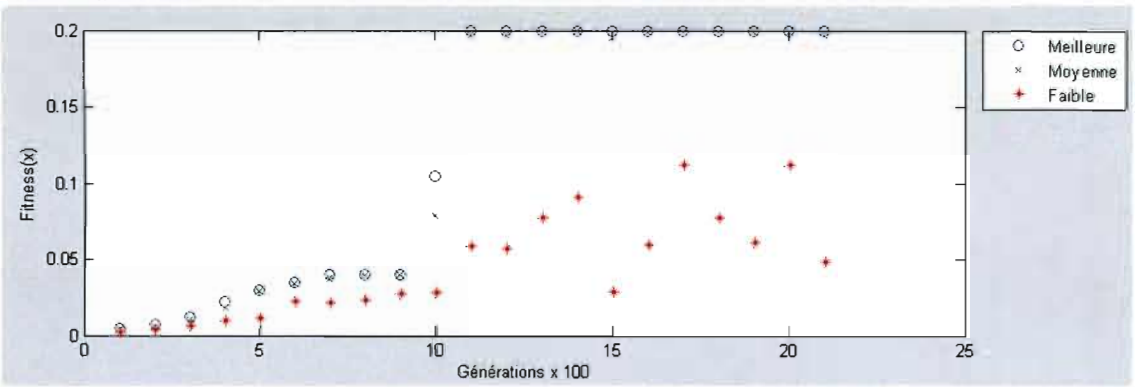


Figure 4.22 Fonction fitness dans la phase d'apprentissage du MLP par optimisation **ACO** $_{\mathfrak{R}}$

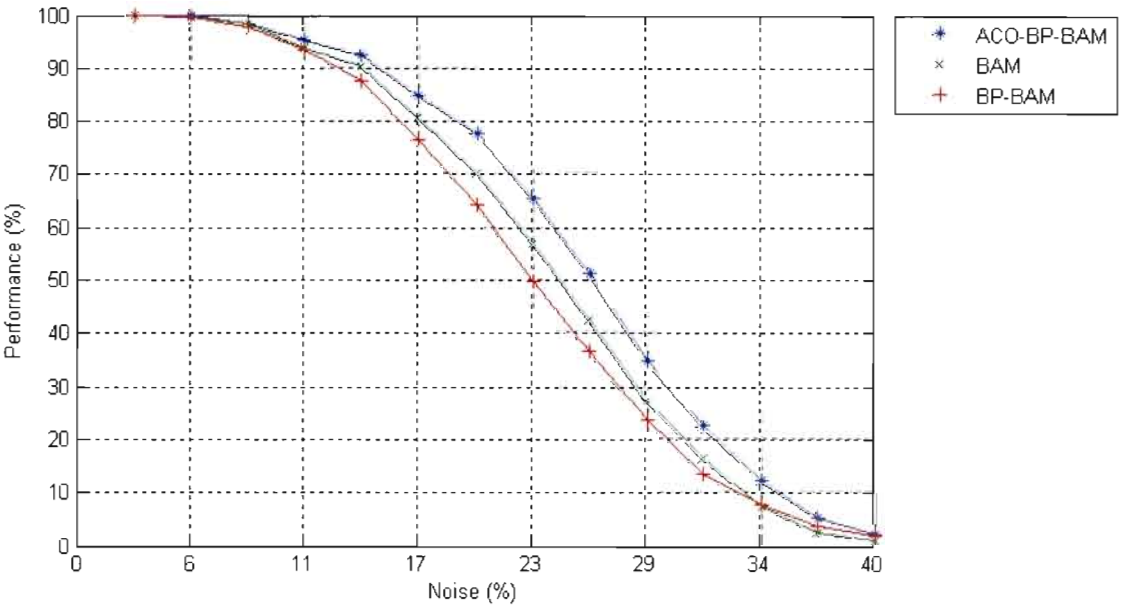


Figure 4.23 Performance de classification en fonction du nombre de pixels flippés.
Modèle BAM standard (vert) vs. BAM--MLP $_{BP}$ (rouge) vs. BAM- MLP $_{ACO_{\mathfrak{R}}-BP}$ (bleu)

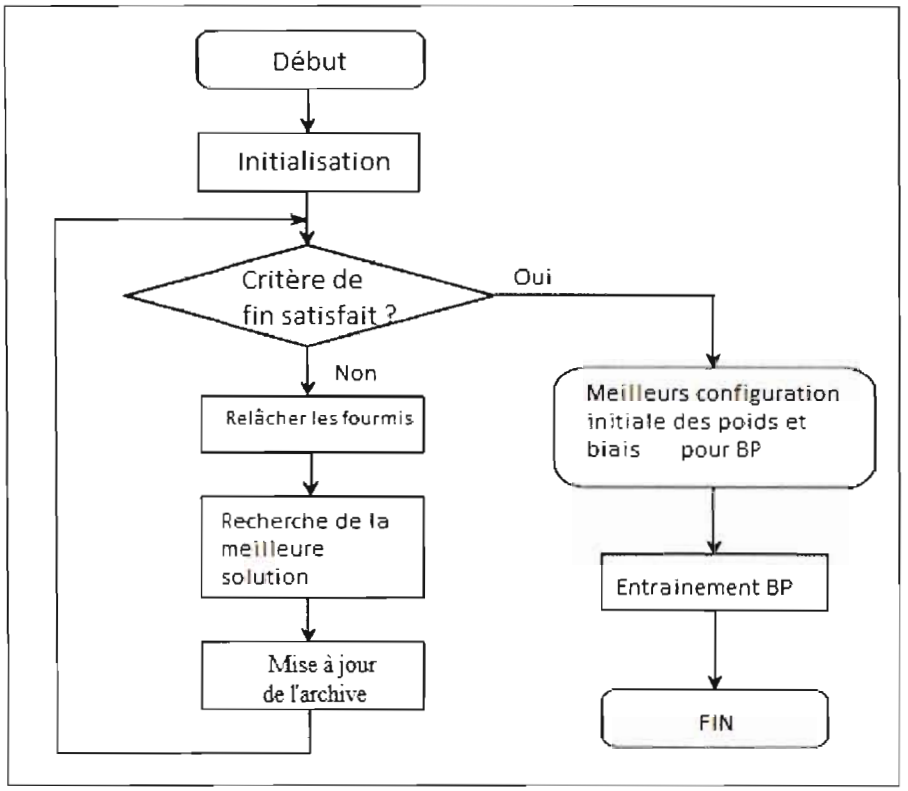


Figure 4.24 Système hybride pour la recherche de solution optimale par $ACO_{\Re}$-BP

Conclusion 3

Les résultats des simulations réalisées sur les modèles hybrides, BAM-RBF et BAM-MLP basés sur des réseaux ENN, montrent une amélioration au niveau de la performance de rappel. En effet, les modèles utilisant des connaissances antérieures (paramètres asymétriques "$\boldsymbol{h}$") sont plus robustes que les modèles sans connaissances antérieures (BAM à fonction de sortie symétrique,"$\boldsymbol{h}$"=0). Ces modèles peuvent apporter jusqu'à 10% d'amélioration pour des stimuli avec un niveau de bruit modéré à élevé.

De plus, on remarque que le modèle BAM-MLP utilisant un apprentissage par rétro-propagation du gradient est moins performant que le même modèle entrainé avec un algorithme d'apprentissage combiné (global évolutionnaire et local) BAM-MLP_{AE-BP}. Ceci peut être dû au fait que l'algorithme

par rétro-propagation d'erreur est plus sensible aux pièges des minima locaux et de ce fait, peut guider la BAM vers le mauvais attracteur, diminuant ainsi sa performance.

Enfin, le modèle BAM-MLP $_{AE-BP}$ est plus performant que le modèle BAM sans connaissances préalables ($\boldsymbol{h} = 0$).

Le tableau ci-dessous donne un aperçu sur les performances obtenues avec les différents modèles de BAM utilisés ainsi que les gains en performance des modèles hybrides.

	BAM	**BAM-RBF**			**BAM-ENN-BP**		
Bruit %		**GA**	**PSO**	**ACO $_{\Re}$**	**AG**	**PSO**	**ACO $_{\Re}$**
11	94	96 (2%)	96 (2%)	96 (2%)	**97 (3%)**	**97 (3%)**	95 (1%)
17	80	84 (4%)	85 (5%)	85 (5%)	86 (6%)	**87 (7%)**	85 (5%)
23	57	61 (4%)	60 (3%)	61 (4%)	63 (6%)	**66 (9%)**	65 (8%)
29	28	35 (7%)	36 (8%)	**38 (10%)**	35 (7%)	32 (4%)	35 (7%)
34	7	**15 (8%)**	**15 (8%)**	**15 (8%)**	11 (4%)	11 (4%)	12 (5%)
40	0	**7 (7%)**	5 (5%)	5 (5%)	2 (2%)	2 (2%)	2 (2%)

Tableau 2. Récapitulatif des performances BAM, BAM-RBF et BAM-ENN-BP

BAM-RBF:

- Améliore jusqu'à 10% la performance de rappel pour des niveaux de bruit moyens à élevés en raison des caractéristiques locales du modèle.
- L'algorithme d'apprentissage ACO est légèrement plus performant que les autres.

BAM-ENN-BP:

- Améliore jusqu'à 9% la performance de rappel pour des niveaux de bruit moyens.
- L'algorithme PSO pour l'apprentissage de l'ENN est légèrement plus performant que l'AG et l'ACO $_{\Re}$

On peut conclure que l'utilisation d'un modèle hybride BAM et ENN améliore la performance de rappel en augmentant les rayons d'attraction des différents points fixes de la BAM, ce qui permet une amélioration de sa robustesse et une augmentation de sa capacité de stockage.

Si l'on compare le modèle BAM-ENN au modèle hybride BAM-GRNN dans (Chartier, Boukadoum et Amiri, 2009), on constate que ce dernier contribue à améliorer jusqu'à 50% la performance de rappel de la BAM comparativement à 10% avec le modèle BAM-ENN. On peut dire que malgré la faible contribution de ce dernier, il demeure par contre plus plausible d'un point de vue biologique. En effet, le GRNN pour fonctionner, doit comparer chaque stimulus d'entrée à l'ensemble des stimuli mémorisés; ce qui va totalement à l'encontre des approches utilisées en neurosciences.

La valeur du paramètre de transmission δ de la fonction de sortie est déterminante pour la performance de la BAM. En effet, dans sa phase d'apprentissage, il est crucial de fixer sa valeur entre 0 et 0.5 afin de garantir une approche monotone vers des états d'équilibre. Par ailleurs, dans sa phase de transmission, on a pu constater des meilleures performances pour des valeurs de δ plus grandes (dans notre cas, δ=1.5). La figure 3.9 montre que pour des valeurs élevées du paramètre δ (ex. δ=1.5), le réseau BAM à fonction de sortie non saturée opère dans sa région chaotique, toutefois notre modèle ne présente pas un tel comportement en raison de la présence des limites de saturation (Eq. 4.1 et 4.2). En effet, l'ajout des seuils de saturation à la fonction de sortie chaotique fait en sortie de prévenir la croissance des valeurs de sortie jusqu'à l'infini et permet à la BAM à converger vers une dynamique stable à points fixes.

RÉSUMÉ ET CONCLUSION

L'objectif du présent mémoire était d'étudier un modèle de mémoire associative bidirectionnelle BAM et d'expérimenter de nouveaux procédés sur ce modèle dans le but d'améliorer ses performances de classification. On a procédé dans un premier temps à une analyse des travaux effectués dans le domaine, ce qui nous a permis de mettre en évidence les performances ainsi que les limitations des modèles BAM proposés. De ces analyses, on a pu constater que la plupart des modèles font preuve de limitations et particulièrement au niveau de leur capacité de stockage. En effet, si l'on considère le modèle BAM-Kosko, celui-ci possède une capacité de stockage qui se limite à $n/2\log_2 n$ (n étant le nombre d'unités de la plus petite couche du réseau). De même, un modèle BAM basé sur une technique de pseudo-inverse, peut stocker un maximum de $n/2$ patrons. On a également pu constater que la plupart des modèles plausibles biologiquement sont incapables de classifier des patrons non linéairement séparables.

Dans ce mémoire, nous avons utilisé un nouveau modèle de mémoire auto et hétéro-associative BAM à fonction de sortie chaotique (Chartier et Boukadoum, 2006a). Nous avons étudié l'effet d'introduire dynamiquement une asymétrie dans la fonction de sortie dans le but d'obtenir une meilleure performance de classification et une meilleure capacité de stockage. Nous avons utilisé à cet effet, une géométrie de catastrophe fronce (*cusp*) et avons montré que celle-ci dépend uniquement de la valeur du paramètre d'asymétrie introduit "$\boldsymbol{h}$" dans la fonction de sortie. Ce paramètre permet d'écarter temporairement des attracteurs de l'espace d'état, augmentant ainsi les rayons d'attraction des points fixes désirés. Les résultats obtenus, appliqués à des tâches de classification, montrent le rôle important du paramètre d'asymétrie "$\boldsymbol{h}$" dans la phase de transmission. En effet, celui-ci joue le rôle de guide dans le processus de recherche de la BAM, lui assurant ainsi une meilleure convergence vers l'attracteur désiré.

La première série de simulations montrent l'influence du paramètre d'asymétrie dans la tâche de classification de problèmes non linéairement séparables. En effet, en utilisant un procédé d'apprentissage par renforcement, la BAM a réussi à classifier correctement le problème XOR, tâche que la plupart des BAM proposées ainsi que la BAM proposée par Chartier et Boukadoum (Chartier et Boukadoum, 2006a) à fonction de sortie symétrique, sont incapables d'effectuer. Dans cette première série de simulations, on a pu déterminer la valeur du paramètre d'asymétrie "$\boldsymbol{h}$" grâce à des

algorithmes d'optimisation. Ces algorithmes ont été choisis pour garantir l'optimalité de la solution "$\boldsymbol{h}$".

Dans la deuxième série de simulations, on a pu constater une amélioration de la performance de rappel en utilisant un modèle de réseau hybride. On a pu constater un gain de performance allant jusqu'à 10% pour le modèle BAM-RBF et jusqu'à 9% pour le modèle BAM-MLP. Malgré le faible gain de performance du modèle hybride BAM-ENN par rapport au modèle BAM-GRNN proposé dans (Chartier, Boukadoum et Amari, 2009), il constitue cependant un bon compromis performance-plausibilité biologique. En effet, on a tenté de maintenir la plausibilité du modèle BAM en utilisant des réseaux hybrides constitués de réseaux multicouche et entrainés par des algorithmes d'optimisation évolutionnaires. De plus, on a constaté que l'amélioration de la performance de la BAM-asymétrique nécessite, pour chaque stimulus d'entrée, une détermination appropriée du paramètre "$\boldsymbol{h}$" correspondant. En effet, un mauvais choix du paramètre "$\boldsymbol{h}$" risque de guider la BAM vers le mauvais attracteur, d'où le besoin de recourir à des algorithmes robustes de recherche globale tels que les algorithmes d'optimisation évolutionnaires.

En plus de ces constatations, on souligne le fait que la fonction de sortie du présent modèle BAM est une extension de celle du modèle original (Chartier et Boukadoum, 2006a), ce qui lui permet d'hériter de toutes les fonctionnalités de ce dernier telles que, la reconnaissance de structures multiples, les associations un-à-plusieurs (Chartier et Boukadoum, 2006b), l'extraction des caractéristiques et la catégorisation (Chartier et al., 2007). Par conséquent, l'adjonction de nouvelles fonctionnalités telles que la classification de problèmes non linéairement séparables et une meilleure qualité de rappel, place cette mémoire parmi les rares modèles, plausibles biologiquement et capable d'une telle diversité de comportements.

BIBLIOGRAPHIE

1- CONNEXIONNISME

Adachi M., Aihara K., 1997. « Associative dynamics in a chaotic neural network ». *Neural Networks,* 10, pp. 83-98.

Aihara K., Takabe T., Toyoda M., 1990. « Chaotic neural networks ». *Physical Letters* A, 144, pp. 333-340.

Alligood K.T., Sauer T., Yorke J.A., 1997. « Chaos: An Introduction to Dynamical Systems », *Springer.*

Bégin J., Proulx R., 1996. « Categorization in Unsupervised Neural Networks: The Eidos Model ». *IEEE Transactions on Neural Networks,* 7, 147-154.

Bergé P., Pomeau Y., Vidal C., 1988. « Le chaos: Théorie et expériences ». Eyrolles.

Chartier S., Boukadoum M., Amari M., 2009. « BAM learning of nonlinearly separable tasks by using an asymmetrical output function and reinforcement learning ». *IEEE Transactions on Neural Networks,* 20, pp. 1281-1292.

Chartier S., Renaud P., Boukadoum M., 2008. « A nonlinear dynamic artificial neural network model of memory ». *New Ideas in Psychology,* vol. 26, pp. 252-277.

Chartier S., Giguère *G.,* Renaud P., Proulx R., Lina J.-M., 2007. « FEBAM: A feature-extracting bidirectional associative memory ». *Proceeding of the International Joint Conference on Neural Networks (IJCNN'07),* pp. 1679-1684.

Chartier S., Boukadoum M., 2006a. « A bidirectional heteroassociative memory for binary and grey-level Patterns ». *IEEE Transactions on Neural Networks,* vol. 17, pp.385-396.

Chartier S., Boukadoum M., 2006b. « A sequential dynamic heteroassociative memory for multistep pattern recognition and one-to-many association ». *IEEE Transactions on Neural Networks,* vol. 17-1, pp. 59-68.

Chartier, S., Boukadoum, M., 2006c. « A Chaotic Bidirectional Associative Memory ». *Actes de Maghrebian Conference on Software Engineering and Artificial Intelligence (MCSEAI 2006), Agadir (Maroc),* pp. 498-501.

Chartier S., Proulx R., 2005. « NDRAM: Nonlinear dynarnic recurrent associative memory for learning bipolar and nonbipolar correlated patterns ». *IEEE Transactions on Neural Networks,* vol.16, pp. 1393-1400.

Chazal G., 2004. « Les mediations théoriques », *Edition Champ Vallon.*

Eom T., Choi C., Lee J., 2002. « Generalized asymmetrical bidirectional associative memory for multiple association ». *Applied Mathematics and Computation,* vol. 127, pp. 221-233.

Fausett L., 1994. « Fundamentals of Neural Networks: Architectures, Algorithms, And Applications ». *Prentice-Hall, Inc.*

Hassoun M. H., 1989. « Dynamic Heteroassociative Neural Memories ». *Neural Networks,* vol. 2, pp. 275-287.

Haykin S., 1997. « Neural Networks: A Comprehensive Foundation ». *PrenticeHall,Inc.*

Holland J.H., 1975. « Adaptation in Natural and Artificial Systems ». *University of Michigan Press, Ann Arbor, MI, USA.*

Kanter I., Sompolinsky H,, 1987. « Associative recall of memory without errors ». *Physical Review A,* vol. 35, pp. 380-392.

Kosko B., 1990. « Unsupervised learning in noise ». *IEEE Transactions on Neural Networks,* 1, 44-57.

Kosko B., 1988. « Bidirectional associative memories ». *IEEE Transactions on Systems, Man and Cybernetics* 18, 49–60.

Personnaz L., Guyon L., Dreyfus G., 1986. « Collective computational properties of neural networks: new learning mechanisms ». *Physical Review A,* vol. 34, pp. 4217-4228.

Personnaz L., Guyon L., Dreyfus G., 1985. « Information storage and retrieval in spin-glass like neural networks ». *J. Physique Letter*, vol. 46, pp. 359-365.

Wang T., Zhuang X., Xing X., 1992. « Weighted learning of bidirectional associative memories by global minimization ». *IEEE Transactions on Neural Networks,* vol. 3, pp. 1010-1018.

Stewart I., 1983. « Elementary catastrophe theory ». *IEEE Transactions on Circuits and Systems,* vol. 30, No. 8, pp. 578-586.

Tokuda, Nagashima T., Aihara, K., 1997. « Global bifurcation structure of chaotic neural networks and its application to traveling salesman problem ». *Neural Networks,* 10, pp. 1673-1690.

Wang T., Zhuang X., Xing X., 1992. « Weighted learning of bidirectional associative memories by global minimization ». *IEEE Transactions on Neural Networks,* vol. 3, pp. 1010-1018.

Wang Y. F., Cruz J. B., Mulligan Jr., 1990. « Two coding strategies for bidirectional associative Memory ». *IEEE Transactions on Neural Networks,* vol. 1, pp. 81-92.

Wang Z., 1996. « A bidirectional associative memory based on optimal linear associative memory ». *IEEE Transactions on Computers,* vol. 45, 1171-1179.

Wasserman, P.D., 1989. « Neural computing theory and practice ». *Van Nostrand Reinhold.*

Zhuang X., Huang Y., Chen S., 1993. « Better learning for bidirectional associative memory ». *Neural Networks*, vol. 6, pp. 1131-1146.

2- ALGORITHMES ÉVOLUTIONNAIRES

Belew R.K., 1990. « Evolution, learning and culture: computational metaphores for adaptive algorithms ». *Complex Systems 4,* 11-49.

Blum C., Dorigo M., 2004. « The hyper-cube framework for ant colony optimization ». *IEEE Tran. on Man, Systems and Cybernetics(B) Vol 34(2)*, 1161–1172.

Bonabeau E., Dorigo M., Theraulaz G., 1999. « Swarm intelligence: from natural to artificial systems ». *Oxford University Press.*

Ching-Jong L., Chao-Tang T., Pin L., 2007. « A discrete version of particle swarm optimization for flowshop scheduling problems ».

Clerc M., Siarry P., 2003. « Une nouvelle méta-heuristique pour l'optimisation difficile : la méthode des essaims particulaires ». *Université de Paris 12 Val-de-Marne, LERISS, France Télécom R&D.*

De Jong K.A., 1988. « Learning with genetic algorithms : an overview ». *Machine learning 3,* 121-138

Dorigo M., Socha K., 2004. « Ant colony optimization ». *The MIT Press,Cambridge, Massachusetts, London, England.*

Dorigo M., Stutzle T., 2004. « Ant Colony Optimization ». *MIT Press, Cambridge, MA.*

Dorigo M., Di Caro G., 1999. « The ant colony optimization meta-heuristic ». *in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, London,* pp. 11–32.

Dorigo M., Di Caro G., Gambardella L.M., 1999. « Gambardella, Ant algorithms for distributed discrete optimization ». *Artificial Life 5,* 137–172.

Dorigo M., Maniezzo V., Colorni A., 1996. « Ant System: Optimization by a colony of cooperating agents ». *IEEE Transactions on Systems, Man, and Cybernetics – Part B 26 (1),* 29–41.

Dorigo M., Maniezzo V., Colorni A., 1991. « Positive feedback as a search strategy ». *Technical Report 91-016, DIPARTIMENTO DI ELETTRONICAQ – POLITECNICO DI MILANO*

Eberhart R., Kennedy J., 1995. « New optimizer using particle swarm theory ». *In Proc. 6th Int. Symp. Micro Machine Human Science*, pp. 39–43.

Goldberg, D.E., 1989 « Sizing Populations for Serial and Parallel Genetic Algorithms ». *Proceedings of the Third International Conference on Genetic Algorithms*, pp.70-79.

Grefenstette J., 1986. « Optimization of control parameters for genetic algorithms », *IEEE Transactions on Systems, Man and Cybernetics 16,* 122–128.

Hoos H.H., Stutzle T., 2004. « Stochastic Local Search: Foundations and Applications ». *Morgan Kaufmann Publishers, San Francisco, CA, USA.*

Hopfield J.J., 1982. « Neural Networks and Physical Systems with Emergent Collective Computational Abilities ». *Proceedings of the National Academy of Sciences,* vol. 79, pp. 2554-2558.
Janson S., Middendorf M., 2005. « A hierarchical particle swarm optimizer and its adaptive variant ». *IEEE Transactions on Systems, Man and Cybernetics - Part B 35(6)* 1272-1282

Kanter I., Sompolinsky H., 1987. « Associative recall of memory without errors ». *Physical Review A,* vol. 35, pp. 380-392.

Kennedy J, Eberhart RC, Shi Y, 2001. « Swarm intelligence ». *San Francisco, CA: Morgan Kaufmann.*

Kennedy J., Eberhart R. C., 1997. « A Discrete Binary Version of the Particle Swarm Optimization ». *Proc. Of the conference on Systems, Man, and Cybernetics SMC97*, pp. 4104-4109.

Kennedy J., Eberhart R., 1995. « Particle swarm optimization ». *In: Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, IEEE Press,* 1942-1948

Kong M., Tian P., 2006. « Introducing a Binary Ant Colony Optimization ». *In: Dorigo M. et al. (Eds.): ANTS 2006, LNCS 4150*, pp. 444–451.

Kong M., Tian P., 2005. « A Binary Ant Colony Optimization for the Unconstrained Function Optimization Problem ». *Y. Hao et al. (Eds.): CIS 2005, Part I, LNAI 3801*, pp. 682–687.

Socha K., 2004. « ACO for continuous and mixed-variable optimization ». *In: Dorigo M. et al. (Eds.): ANTS 2004, LNCS 3172*, pp. 25–36.

Montes de Oca M.A., Stutzle T., Birattari M., Dorigo M., 2006. « A Comparison of Particle Swarm Optimization algorithms Based on Run-Length Distributions ». *Proceedings of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence, Brussels, Belgium*

Socha, K., Dorigo, M., 2006. « Ant colony optimization for continuous domains ». *European Journal of Operational Research, in press.*

Stutzle T., Hoos H.H., 2000. « MAX–MIN Ant System. Future Generation Computer Systems ». 16(8), 889–914.

Thierens, D., Goldberg, D.E. 1994. « Convergence models of genetic algorithm selection schemes ». *In Parallel Problem Solving fron Nature, PPSN III*, pp. 119-129.

Whitley D., 1995. « Genetic Algorithms and Neural Networks ». *Genetic Algorithms in Engineering and Cornouter Science, edited by G. Winter et al., Wiley Publishers,* pp.203-216.

Zheng, Y.L., Ma L.H., Zhang L.Y., Qian J.X., 2003. « On the convergence analysis and parameter selection in particle swarm optimization ». *In: Proceedings of the IEEE International Conference on Machine Learning and Cybernetics, Piscataway, NJ, IEEE Press,* 1802-1807.