Modélisation neuronale avec une structure minimale

Ouahib GUENOUNOU[1], Ali BELMEHDI[1], Boutaib DAHHOU[2,3]

[1]Laboratoire de technologie ; industrielle et de l'information
Université A.Mira de Béjaia, Route Targa-Ouzemour, Béjaia, Algérie.
wguenounou@yahoo.fr, albelme@yahoo.fr

[2] CNRS ; LAAS ; 7 avenue du colonel Roche ; F-31077, Toulouse, France.

[3] Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077, Toulouse, France.
dahhou@laas.fr

Résumé— **Le présent travail porte sur l'optimisation de la structure d'un réseau de neurones MLP(Multi-Layer Perceptron) à deux couches cachées par un algorithme hybride qui se déroule en deux étapes. Au cours de la première étape l'algorithme de rétropropagation est utilisé pour un premier ajustement des poids de connexions et des biais du réseau neuronal. Dans la deuxième étape on utilise le NSGA-II, algorithme génétique multi-objectifs (Nondominated Sorting Genetic Algorithm-II) pour l'optimisation simultanée des paramètres et du nombre de neurones dans chaque couche cachée du réseau. L'efficacité de cet algorithme hybride proposé est évaluée à travers un problème classique de modélisation d'un système non linéaire.**

Mots-clés— **Réseau de neurones, Optimisation, Algorithme génétique, NSGA-II, rétropropagation, Algorithme hybride.**

I. Introduction

Les réseaux de neurones artificiels présentent une alternative prometteuse pour de nombreux domaines ; à savoir la reconnaissance de formes, le traitement d'images, le contrôle industriel et l'identification [1]. S'inspirant des règles de la génétique, les techniques neuronales interviennent dans un contexte où les outils courants ont atteint leurs limites.

Grâce à leur propriété d'approximation universelle, les réseaux de neurones ont vu leur champ d'application s'étaler à de nouvelles classes de problèmes, réputés de complexes, avec succès. Les études antérieures ont montré qu'avec un réseau neuronal à une seule couche cachée et un nombre suffisant de neurones cachés, on peut identifier n'importe quel système avec n'importe qu'elle précision[2]. Toutefois, la convergence de l'algorithme d'apprentissage pour certains systèmes est lente et leur stabilisation est ardue. L'addition d'une couche cachée supplémentaire peut réduire le temps de convergence d'une part, et rendre le système plus stable d'autre part.

Si le nombre d'entrées et de sorties est imposé par la fonction à approximer, celui de couches cachées et de neurones par couche restent à déterminer. Généralement, on procède par tâtonnement mais ce n'est pas toujours évident d'y faire.

Dans ce travail on propose un algorithme hybride pour une recherche automatique du nombre optimal de neurones à mettre dans les couches cachées d'un réseau MLP. L'article est organisé comme suit :

La section deux présente la structure et les équations du réseau MLP utilisé dans le papier. La section trois rappelle le principe de fonctionnement du NSGA-II (Nondominated Sorting Genetic Algorithm-II), algorithme d'optimisation multi-objectifs. La section quatre est consacrée à la description de l'algorithme hybride proposé. Les résultats de simulation sont donnés dans la section cinq. Une conclusion termine le papier.

II. Réseaux de neurones MLP à deux couches cachées

Parmi les architectures des réseaux de neurones les plus utilisées, on peut citer l'architecture multicouches (MLP : Multi Layer Perceptron). Les neurones composant ce réseau s'organisent en N couches successives ($N \geq 3$). Dans l'exemple suivant (figure 1), nous présentons un perceptron à quatre couches. Les neurones de la première couche, nommée couche d'entrée, voient leur activation forcée à la valeur d'entrée. La dernière couche est appelée couche de sortie. Elle regroupe les neurones dont les fonctions d'activation sont généralement de type linéaire. Les couches intermédiaires sont appelées couches cachées. Elles constituent le cœur du réseau. Les fonctions d'activation utilisées sont de type sigmoïde.

Sur la figure 1, les termes b_i^l et ω_{ij}^l désignent respectivement le biais du neurone i de la couche l et le poids de connexion entre le neurone j de la couche $l-1$ et le neurone i de la couche l.

Tenant compte de ces notations, la sortie du neurone i dans la couche l est peut être donnée par :

$$U_i^l = \sum_j^{N_{l-1}} \omega_{ij}^l \times O_j^{l-1} + b_i^l \quad (1)$$

$$O_i^l = g^l(U_i^l) \quad (2)$$

$$l = 1, 2, 3$$

où $g^l(.)$ est la fonction d'activation des neurones de la couche l.

On peut réécrire les équations ci-dessus sous forme matricielle comme suit :

$$\underline{U^l} = W^l \times \underline{O^{l-1}} + \underline{b^l} \quad (3)$$

$$\underline{O^l} = \underline{g^l}(\underline{U^l}) \quad (4)$$

avec : $\underline{U}^l = (U_1^l, U_2^l, ..., U_{N_l}^l)^T$, $\underline{O}^l = (O_1^l, O_2^l, ..., O_{N_l}^l)^T$, $\underline{b}^l = (b_1^l, b_2^l, ..., b_{N_l}^l)^T$ et

$$W^l = \begin{pmatrix} \omega_{11}^l & \omega_{12}^l & \cdots & \omega_{1N_{l-1}}^l \\ \omega_{21}^l & \omega_{22}^l & \cdots & \omega_{2N_{l-1}}^l \\ \vdots & \vdots & \ddots & \cdots \\ \omega_{N_l 1}^l & \omega_{N_l 2}^l & \cdots & \omega_{N_l N_{l-1}}^l \end{pmatrix}$$

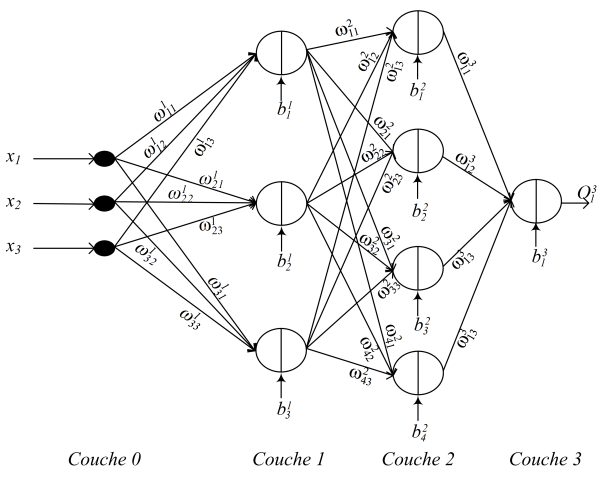

Fig. 1. Réseau MLP à deux couches cachées

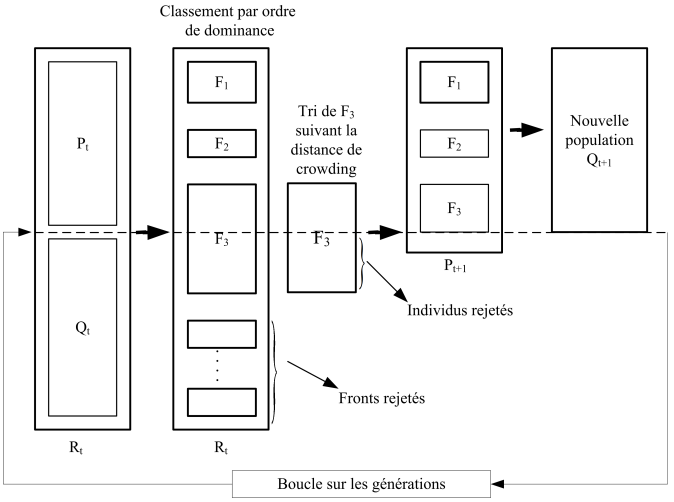

Fig. 2. Schéma de l'évolution de l'algorithme NSGA-II

III. Optimisation multio-bjectifs par NSGA-II

Le principe d'une optimisation multi-objectifs est différent de celui d'une optimisation mon-objectif. Le but principal d'une optimisation mono-objectif est de trouver la solution optimale qui résulte en la meilleure valeur (plus petite ou plus grande) de la fonction objectif. Dans un problème multi-objectifs il y a plus d'une fonction objectif, chaque fonction pouvant avoir une solution optimale différente. Le but est de chercher un compromis, un ensemble de solutions (pareto-optimales) plutôt qu'une seule solution. Dans cette section, nous présenterons le principe général de NSGA-II qui est un algorithme génétique multi-objectif introduit par Deb [3].

A. Principe du NSGA-II

En proposant le NSGA-II, Deb a tenté de résoudre toutes les critiques faites sur le NSGA [4] : non élitiste, complexité de calcul et utilisation de sharing qui implique le réglage d'un ou plusieurs paramètres. Dans cet algorithme, à chaque génération t une population de parents (P_t) de taille N et une population d'enfants (Q_t) de même taille sont assemblées pour former une population (R_t) de taille $2N$, comme indiqué sur la figure 2. Cet assemblage permet d'assurer l'élitisme. La population (R_t) est ensuite répartie en plusieurs fronts (F_1, F_2, ...) par une procédure de tri, plus rapide que celle proposée dans la première version de NSGA. Une nouvelle population parent (P_{t+1}) est formée en ajoutant les fronts au complet (premier front F_1, second front F_2, etc...) tant que ceux-ci ne dépassent pas N. Si le nombre d'individus présents dans (P_{t+1}) est inférieur à N, une procédure de crowding est appliquée sur le premier front suivant F_i non inclus dans (P_{t+1}). Le but de cet opérateur est d'insérer les $(N - P_{t+1})$ meilleurs individus de F_i qui manquent dans la population (P_{t+1}). Une fois que les individus de la population (P_{t+1}) sont identifiés, une nouvelle population enfant (Q_{t+1}) est créée par sélection, croisement et mutation. La sélection par tournoi est utilisée mais le critère de sélection est maintenant basé sur l'opérateur de comparaison $(\prec_n)$ défini ci-dessous. Le processus se répète d'une génération à une autre jusqu'à satisfaction d'un critère d'arrêt.

B. Procédure de tri rapide (Fast sorting non dominated)

La répartition de la population en plusieurs fronts s'effectue de la manière suivante :

1. Pour chaque solution p de (R_t) on calcule deux paramètres :
   - n_p (compteur de dominance), il représente le nombre de solutions qui dominent la solution p
   - S_i ensemble de solutions dominées par p.
2. $i = 1$ initialisation du compteur de front
3. Identification des solutions non dominées $n_p = 0$, ces solutions forment le front F_i.
4. Pour chaque solution p de F_i on parcourt l'ensemble S_p et on retranche 1 au n_p de chaque solution.
5. $i = i + 1$ incrémente le compteur de front
6. On recommence les étapes à partir de 3 jusqu'à ce que tous les points soient traités.

Cet algorithme est d'une complexité de $O(k.N^2)$, alors que celui utilisé dans la première version est de $O(k.N^3)$. k étant la taille du vecteur objectifs

C. Distance de crowding

La dernière critique faite sur le NSGA est l'utilisation du sharing. Une méthode qui exige le réglage d'un ou plusieurs paramètre(s). Dans NSGA-II, Deb et al remplacent la procédure de sharing par une procédure de crowding, basée sur un calcul de distance (distance de crowding) qui ne nécessite aucun paramétrage et qui est également d'une complexité algorithmique moindre que celle de sharing. La

distance de crowding d'une solution particulière i se calcule en fonction du périmètre de l'hypercube ayant comme sommets les points les plus proches de i sur chaque objectif. Sur la figure 3, est représenté l'hypercube en deux dimensions associé au point i. Le calcul de la distance de crowding nécessite, avant tout, le tri des solutions selon chaque objectif, dans un ordre ascendant. Ensuite, pour chaque objectif, les individus possédant des valeurs limites se voient associés une distance infinie. Pour les autres solutions intermédiaires, on calcule une distance de crowding égale à la différence normalisée des valeurs des fonctions objectifs de deux solutions adjacentes. Ce calcul est réalisé pour chaque objectif. La distance de crowding d'une solution est obtenue en sommant les distances correspondantes à chaque objectif.

L'algorithme 1 reprend toutes les étapes décrites ci-dessus.

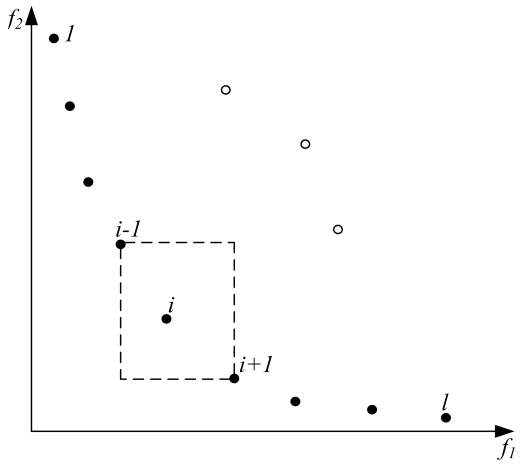

Fig. 3. Distance de crowding (les points noirs sont des solutions appartenant au même front)

Dans cet algorithme, f_m^{i+1} et f_m^{i-1} représentent respectivement les valeurs de la $m$ième fonction objectif des solutions i+1 et i−1 alors que les paramètres f_m^{Max} et f_m^{Min} désignent les valeurs maximale et minimale de la $m$ième fonction objectif.

Algorithm 1 Calcul de la distance de crowding pour chaque solution d'un front

1: $l = |I|$, nombre de solutions dans le Front I
2: Pour chaque solution i poser $I[i]_{distance} = 0$, Initialisation des distances.
3: Pour chaque objectif m :
-$I = trier(I, m)$, trier I par ordre croissant selon le critère m
-$I[1]_{distance} = I[l]_{distance} = +\infty$
-Pour i=2 jusqu'à $l-1$ faire
$I[i]_{distance} = I[i]_{distance} + (\frac{f_m^{i+1} - f_m^{i-1}}{f_m^{Max} - f_m^{Min}})$

D. Opérateur de crowding de comparaison

Cet opérateur est utilisé pour guider le processus de sélection comme suit : chaque solution i de la population est identifiée par son rang i_{rank} et sa distance de crowding $i_{distance}$. L'opérateur $\prec_n$, défini ci-dessous, permet d'établir un ordre de préférence entre deux solutions :

$$\begin{aligned} i \prec_n j \quad & si \quad (i_{rank} < j_{rank}) \\ & ou \quad (i_{rank} = j_{rank}) \; et \; (i_{distance} > j_{distance}) \end{aligned}$$

Entre deux solutions de fronts différents, on préfère la solution avec le plus petit front. Pour deux solutions qui appartiennent au même front, on préfère la solution située dans une région dépeuplée, c'est-à-dire la solution possédant la plus grande valeur de distance de crowding.

IV. Algorithme d'optimisation

Dans cette section, nous allons présenter l'algorithme hybride proposé pour l'optimisation de la structure du réseau MLP à deux couches cachées. Le principe de cet algorithme est très simple : il consiste à utiliser dans un premier temps l'algorithme de rétropropagation pour un ajustement de paramètres(poids et biais) et dans un second temps un algorithme génétique (NSGA-II) pour un deuxième ajustement de paramètres tout en minimisant le nombre de de neurones dans chaque couche cachée.

A. Première étape :Optimisation par l'algorithme de rétrptopagation du gradient

L'algorithme de rétropropagation (backpropagation) est l'un des algorithmes supervisé les plus utilisés pour l'apprentissage des réseaux de neurones. C'est d'ailleurs à sa découverte au début des années 80 [5] que l'on doit le renouveau d'intérêt pour les réseaux de neurones. L'objectif de cet algorithme est de modifier les poids du réseau dans le sens contraire du gradient du critère de performance.
Dans ce qui suit, nous allons présenter les équations constituant l'algorithme en utilisant un réseau multicouches. Une mise sous forme matricielle sera aussi faite afin de faciliter l'implantation de l'algorithme sous un logiciel bien adapté aux calculs matriciels.
Considérons le réseau multicouche décrit précédemment. Pour alléger l'exposé, on suppose que l'apprentissage se fait à chaque présentation d'un couple entrée/sortie de l'ensemble d'apprentissage. Le critère de performance à minimiser peut être alors exprimé par :

$$J(t) = 0.5 \times \sum_{i=1}^{N_L} (O_i^L(t) - d_i(t))^2 \tag{5}$$

avec :
$J(t)$est la valeur du critère à l'instant t.
$d_i(t)$ est la ième sortie désirée à l'instant t.
Les paramètres du réseau sont modifiés suivant la règle du gradient comme suit :

$$\omega_{ij}^l(t+1) = \omega_{ij}^l(t) - \eta \frac{\partial J(t)}{\partial \omega_{ij}^l(t)} \tag{6}$$

$$b_i^l(t+1) = b_i^l(t) - \eta \frac{\partial J(t)}{\partial b_i^l(t)} \tag{7}$$

avec η est une constante positive appelée taux d'apprentissage.
Le calcul des quantités $\frac{\partial J}{\partial \omega}$ et $\frac{\partial J}{\partial b}$ fait intervenir les

décompositions ci-dessous :

$$\frac{\partial J(t)}{\partial \omega_{ij}^l(t)} = \frac{\partial J(t)}{\partial U_i^l(t)} \times \frac{\partial U_i^l(t)}{\partial \omega_{ij}^l(t)} \quad (8)$$

$$\frac{\partial J(t)}{\partial b_i^l(t)} = \frac{\partial J(t)}{\partial U_i^l(t)} \times \frac{\partial U_i^l(t)}{\partial b_i^l(t)} \quad (9)$$

De l'équation (1) on déduit que :

$$\frac{\partial U_i^l(t)}{\partial \omega_{ij}^l(t)} = O_j^{l-1} \quad (10)$$

$$\frac{\partial U_i^l(t)}{\partial b_i^l(t)} = 1 \quad (11)$$

En posant, $\delta_i^l(t) = \frac{\partial J(t)}{\partial U_i^l}$ on obitent :

$$\frac{\partial J(t)}{\partial \omega_{ij}^l(t)} = \delta_i^l(t) \times O_j^{l-1} \quad (12)$$

$$\frac{\partial J(t)}{\partial b_i^l(t)} = \delta_i^l(t) \quad (13)$$

La quantité δ_i^l exprime la sensibilité du critère de performance aux changements du potentiel U_i^l du neurone i de la couche l. Dans le cas où i est l'indice d'un neurone de sortie ($l = L$), on obtient :

$$\delta_i^L(t) = \frac{\partial J(t)}{\partial U_i^L} = \frac{\partial J(t)}{\partial O_i^L} \times \frac{\partial O_i^L}{\partial U_i^L} = (O_i^L(t) - d_i(t)) \times \dot{g}^l(U_i^L(t)) \quad (14)$$

avec
$\dot{g}^L(U_i^L(t)) = \frac{dg^L(U_i^L(t))}{dU_i^L(t)}$
Dans le cas où i est l'indice d'un neurone caché ($1 < l < L-1$), on peut vérifier aisément que les fonctions de sensibilité satisfont la relation récurrente ci-dessous [6] :

$$\underline{\delta^l} = \dot{G}^l(\underline{U^l}) \times (W^{l+1})^T \times \underline{\delta}^{l+1} \quad (15)$$

où

$$\dot{G}^l(\underline{U^l}) = \begin{pmatrix} \dot{g}^l(U_1^l(t)) & 0 & \cdots & 0 \\ 0 & \dot{g}^l(U_2^l(t)) & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \dot{g}^l(U_1^l(t)) \end{pmatrix} \quad (16)$$

Pour résumer, l'algorithme de mise à jour des paramètres du réseau se déroule comme suit : premièrement le vecteur d'entrée $\underline{U}^0 = (x_1, x_2, \cdots, x_{N_0})^T$ est propagé vers la sortie en utilisant les équations (1) et (2). Ensuite, on calcule les fonctions de sensibilités par rétropropagation de l'erreur de sortie à l'aide des équations (14) et (15). Finalement on modifie les poids et les biais en utilisant les équations (6), (7), (8) et (9).

B. Deuxième étape : Optimisation par algorithme génétique

Une fois, les paramètres du réseau sont obtenus, un algorithme génétique (NSGA-II) peut être appliquer pour un deuxième réglage des paramètres toute en minimisant le nombre de neurones dans chaque couche cachée.
La figure 4 montre la structure du chromosome adoptée pour notre problème. Elle contient deux types de gènes (gènes de contrôle et gènes de paramètres) : les gènes de contrôle sont utilisés pour activer les neurones des deux couches cachées alors que les gènes de paramètres permettent de coder les poids et les biais du réseau. Quant " 1 " est assigné au gène de contrôle ($z_i^l = 1$), le neurone correspondant devient actif. Dans le cas contraire (un" 0 " est assigné au gène de contrôle), ce neurone devient inactif (i.e éliminé de la couche cachée) et les gènes de paramètres associés à ce neurone deviennent aussi inactifs.
Ainsi, si $z_i^l = 0, \forall i = 1, 2, \cdots, N_l$ et $\forall l = 1, 2$, aucun neurone n'est actif dans chaque couche cachée, dans ce cas, la réseau ne peut fournir une sortie. Pour contourner cette singularité, nous introduisons la contrainte suivante :

$$\sum_{i=1}^{N_l} z_i^l \geq M, \qquad l = 1, 2 \quad (17)$$

Avec M le nombre minimal de neurones actifs dans chaque couche cachée.

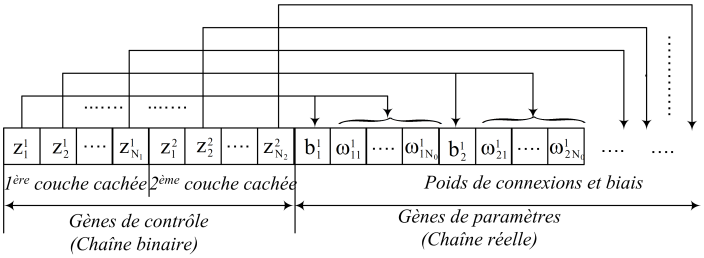

Fig. 4. Structure du chromosome optimisant le réseau de neurones

B.1 Opérateur de croisement

Le croisement consiste à inter-changer entre deux parents une partie de leur chaîne pour créer de nouveaux individus (enfants). Plusieurs opérateurs de croisement ont été proposés dans la littérature [7] à savoir le croisement en un point, le croisement uniforme et le croisement en deux points. C'est ce dernier qui est utilisé dans cette étude, il est appliqué au niveau des gènes de paramètres avec une probabilité de croisement p_{cp} et au niveau des gènes de contrôle avec une probabilité p_{cc} inférieure à p_{cp}.

B.2 Opérateur de mutation

Généralement la mutation est appliquée avec une probabilité inférieure à celle du croisement, elle est utilisée pour éviter de confondre les optimums locaux avec l'optimum global. Elle existe sous plusieurs formes et dépend souvent du type de codage (binaire, décimal, réel $\cdots$ etc.) adopté. Comme il existe deux types de chaînes dans le chromosome (gènes de contrôle : chaîne binaire et gènes de paramètres : chaîne réelle), il convient alors de spécifier pour chacune d'elle le type de mutation permettant une meilleure exploration de l'espace de recherche.

1. **Mutation des gènes de contrôle** : Cette mutation est appliquée avec une probabilité p_{mc} sur chaque gène de contrôle. Il s'agit d'inverser, les gènes qui satisfont le test de probabilité (pour chaque gène, un nombre r est généré aléatoirement, si ce dernier est inférieur à p_{mc} le gène est muté).

2. **Mutation des gènes de paramètres** : Une mutation gaussienne est appliquée au niveau des gènes de paramètres. Le principe de base de ce type de mutation est d'ajouter avec une probabilité p_{mp} un bruit gaussien centrée $N(0,\sigma)$ au gène que l'on désire faire muter :

$$x_k^{'} = x_k + N(0,\sigma) \tag{18}$$

C. Fonctions objectifs

En générale la structure d'un réseau de neurone (MLP) est définie par le nombre de couches cachées et le nombre de neurones dans chaque couche. Dans notre cas le nombre de couches cachées est fixé à deux et on cherche le nombre optimal de neurones dans chaque couche tout en garantissant une bonne qualité d'identification. Deux objectifs ont été alors définis :

$$Ob_1 = \frac{1}{N}\sum_{k=1}^{N}(y^d(k) - y(k))^2 \tag{19}$$

$$Ob_2 = \sum_{l=1}^{2}\sum_{i=1}^{N_l} z_i^l \tag{20}$$

Le premier objectif Ob_1 représente l'erreur quadratique moyenne, il est le critère le plus utilisé pour une mesure de performances dans le cas d'approximation de fonctions. Le deuxième objectif Ob_2 est relatif aux nombre total de neurones dans les deux couches cachées, il est obtenu par une simple sommation des gènes de contrôle.

D. Critère d'arrêt

Il n'existe pas de conditions d'arrêt universellement acceptées pour les algorithmes génétiques multi-objectifs. Nous arrêtons notre algorithme après avoir exécuté un certains nombre de générations.

V. Résultats de simulations

On ce propose, ici, de tester les performances de l'algorithme d'apprentissage hybride à travers un problème classique de modélisation d'un système non linéaire proposé par Narendra et Pathasarathy [8].

A. Modèle de simulation

Le système non linéaire à modéliser est décrit par l'équation aux différences suivante :

$$y(k) = [\frac{y(k-1)y(k-2)(y(k-1)+2.5)}{1+y^2(k-1)+y^2(k-2)} + u(k-1)] \tag{21}$$

Où $u(k)$ et $y(k)$ sont respectivement l'entrée et la sotie du système.
Le système est identifié par un réseau MLP à deux couches cachées décrit dans la section 2. Il possède trois entrée $u(k-1)$, $y(k-1)$ et $y(k-2)$ et une seule sortie $y(k)$ ce qui correspond à trois neurone dans la couche d'entrée et un seul neurone de la couche de sortie. Le nombre de neurones dans chaque couche cachée a été fixé à 15.

B. Première phase

Dans cette première phase, l'algorithme de retropropagation est utilisé pour l'adaptation des paramètres du réseau (poids et biais). Une base de données de 500 points, générés à partir de l'équation (21) et une entrée $u(k)$ uniformément répartie dans l'intervalle $[-2,2]$, a été considérée. Les poids du réseau sont initialisés par de valeurs aléatoires prisent dans l'intervalle [-1,1] et le pas d'apprentissage η a été fixé à 0,1.
La figure 5 donne les réponses du système et du réseau après la première étape d'apprentissage pour une entrée de test :

$$u(k) = \sin(2\pi k/25) \tag{22}$$

La sortie du réseau suit celle du système avec des erreurs

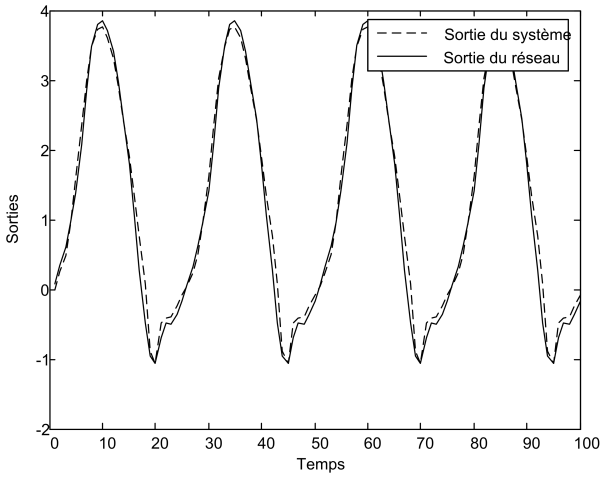

Fig. 5. Test du réseau aprés la première étape d'apprentissage

relativement importantes. Ce résultat insuffisant est dû d'une part aux inconvénients de l'algorithme de retropropagation (blocage au niveau des optimums locaux) et d'autre part au choix de la structure de réseau qui a été fixée intuitivement à 15 neurones dans chaque couche cachée.
Deuxième phase
Une fois la première phase terminée, le NSGA-II est utilisé pour un deuxième ajustement des paramètres tout en optimisant le nombre de neurone dans chaque couche cachée. Les paramètres du NSGA-II sont résumés ainsi :

- probabilité de croisement des gènes de paramètres : 0,85
- Probabilité de croisement des gènes de contrôle : 0,20
- Probabilité de mutation des gènes de paramètres : 0,01
- Probabilité de mutation des gènes de contrôle : 0,005.
- Taille de la population : 40
- Maximum de génération : 1000
- Le nombre minimal de neurones actifs : 5

La figure 6 montre la distribution des objectifs correspondants aux chromosomes de la première population. On constate que l'objectif 1 (EQM) prend ses valeurs dans l'intervalle $[0,053, 0,068]$ proche de la solution obtenue par rétropropagation ($Ob_1 = 0,056$, voir figure 6) tandis que l'objectif 2 prend ses valeurs autour de 30. Cette répartition ne reflète que la méthode adoptée ici pour l'initialisation des chromosomes. Contrairement aux méthodes traditionnelles qui se basent sur un processus aléatoire, dans notre cas nous utilisons le résultat obtenu par retropropagation

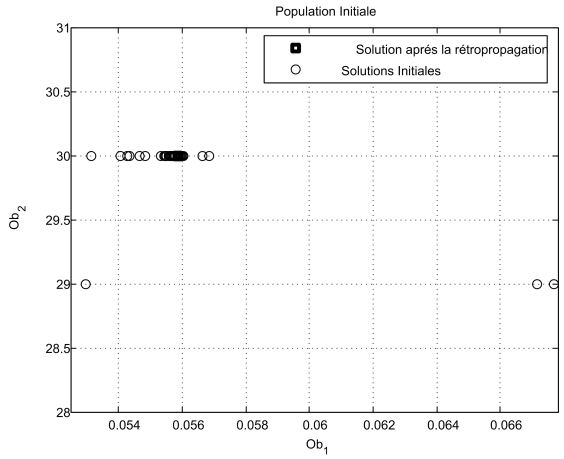

Fig. 6. Répartition des solutions de la première génération

pour générer la population initiale. Au début un premier chromosome mère est créé par codage des paramètres du réseau conformément à la structure illustrée par la figure 4. Pour activer l'ensemble des neurones des deux couches cachées, les gènes de contrôle sont forcés à 1. Ensuite les autres chromosomes sont générés par mutation avec des taux faibles des gènes de paramètres et de contrôle de ce premier chromosome. Cela permet comme l'indique la figure 6 de créer des chromosomes puissants au sens de Ob_1. Nous rappelons que la première phase ne permet d'améliorer que l'objectif 1, l'objectif 2 étant fixé à 30 (tous les neurones sont actifs). Comme le NSGA-II, est un algorithme d'optimisation multi-objectifs élitiste, le résultat obtenu à la fin de la première phase ne risque pas de se perdre à travers les générations, s'il n'est pas amélioré à la dernière génération, il sera au moins préservé. La figure 7 montre la répartition des objectifs relatifs aux chromosomes du front de Pareto de la dernière génération. On constate, une mélioration significative des deux objectifs.

Une fois l'apprentissage accompli, les performances du

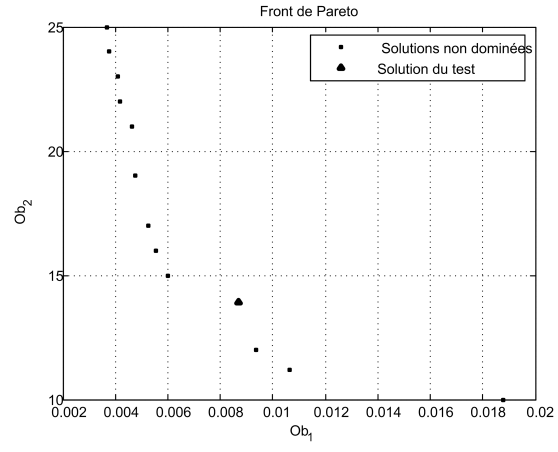

Fig. 7. Répartition des solutions du front de Pareto

réseau sont alors évaluées en utilisant le même signal d'entrée de l'équation (22).

Le résultat de simulation obtenu est donné par la figure 8. On constate que le réseau réalise une bonne approximation du système système à identifier. Pour ce résultat, nous avons choisi les paramètres relatifs au chromosome appartenant au front de Paréto de la population finale et indiqué par un triangle. Dans ce chromosome, uniquement 14 neurones (6 dans la première couche et 8 dans la deuxième couche) sont considérés, le reste (16 neurones) est désactivé par les valeurs prises par les gènes de contrôle (les gènes de contrôle correspondant sont nuls).

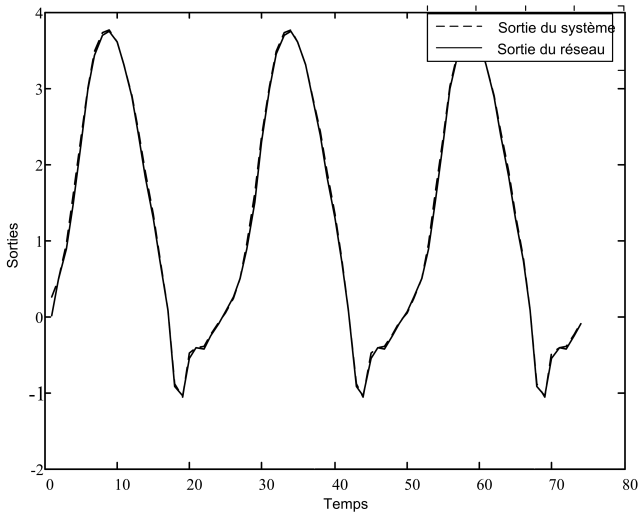

Fig. 8. Test du modèle

VI. Conclusion

Une nouvelle approche a été proposée pour obtenir un modèle neuronal avec une structure optimale. Cette approche est basée sur un algorithme hybride qui combine les avantages de l'algorithme de rétropropagation (algorithme d'ajustement rapide) et les algorithmes génétiques (méthode de recherche globale). Durant la première étape, la rétropropagation est utilisée pour ajuster les paramètres du réseau de manière à minimiser l'erreur entre le système et le modèle neuronal. Le NSGA-II est appliqué ensuite pour ajuster encore une fois les paramètres du réseau tout en minimisant le nombre de neurones dans les couches cachées.

Références

[1] Renders J.M. *Algorithmes génétiques et réseaux de neurones*. Hermes Science Publications, 1995.

[2] Hornik K., Stinchcombe M. et White H. Multilayer feedforward networks are universal approximators. *Neural Networks*, vol. 2, n° 5, pp. 359-366, 1989.

[3] Deb K.,Pratap A., Agarwal S. et Meyarivan T. A Fast and Elitist Multiobjective Genetic Algorithm : NSGA-II. *IEEE Transactions on Evolutionary Computation*, vol. 6, n° 2, pp. 182-197, 2002.

[4] Srinivaset V. et Deb K. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, vol. 2, n° 3, pp. 221-248, 1993.

[5] Rumelhart D.E. et McClelland J. L. Parallel Distributed Processing :Explorations in the Microstructure of Cognition. *M.I.T. Press*, vol. 1, 1986.

[6] Hagan M.T. et Menhaj M.B. Training Feedforward Networks with Marquardt Algorithm. *Evolutionary Computation*, vol. 5, n° 6, pp. 989-993, 1993.

[7] Man K.F., Tang K.S et Kwong S. Genetic Algorithms : Concepts and Applications. *Evolutionary Computation*, vol. 43, pp. 519-534, 1996.

[8] Narendra K.S et Parthasarathy K. Identification and Control of Dynamical Systems Using Neural Networks. *IEEE Transaction on Neural Networks*, vol. 1, n° 1, pp. 4-27, 1990.