Initialisation des Réseaux de Neurones Non Récurrents à coefficients réels par Algorithmes Evolutionnistes[1]

Cédric GEGOUT[•◇⋆] & Fabrice ROSSI[2◇⋆]

• Ecole Normale Supérieure de Lyon,
Laboratoire de l'Informatique du Parallélisme,
46 avenue d'Italie, 69007 LYON FRANCE

◇ Ecole Normale Supérieure de Paris,
45 rue d'Ulm, 75005 PARIS FRANCE

⋆ THOMSON-CSF/SDC/DPR/R4,
7 rue des Mathurins, 92223 BAGNEUX FRANCE
Téléphone: (33 1) 40 84 29 05 Télécopie: (33 1) 40 84 29 50
e-mail: gegout@clipper.ens.fr; cgegout@lip.ens-lyon.fr

Résumé: L'utilisation des algorithmes évolutionnistes est efficace pour l'apprentissage des réseaux de neurones booléens mais semble complexe pour l'apprentissage des réseaux de neurones non-récurrents à coefficients réels. Pour l'apprentissage de ce type de réseaux de neurones, quelques méthodes de descente de gradient fournissent une solution optimale globale en un temps raisonnable, malheureusement elles peuvent converger vers des optima locaux. Dans ce cadre d'étude nous proposons une approche évolutionniste pour initialiser un algorithme de descente de gradient utilisant une rétro-propagation de l'erreur. L'algorithme fournit une bonne solution sur laquelle on applique une méthode de descente de gradient pour obtenir une solution plus précise. Les simulations montrent que notre méthode d'initialisation réduit le temps nécessaire pour obtenir un réseau optimal et améliore la robustesse des algorithmes de descente de gradient.

Mots clés: Algorithmes Génétiques, Réseaux de neurones non-récurrents, recherche par descente de gradient, apprentissage de perceptrons multicouches, optimisation, paramétrisation géométrique des réseaux de neurones.

Abstract: The use of evolutionary algorithms to design and train boolean neural networks have shown to be efficient, but seems to be complex for the learning process of any real feedforward networks. For feedforward net learning, some gradient-descent methods are more appropriate than the evolutionary ones in converging on an exact optimal solution in a reasonable time, unfortunately they are inclined to fall into local optima. In this framework, this paper proposes an evolutionary approach to initialize the weight and bias values of neural networks before back-propagation trainings. The evolutionary algorithm provides a good solution, then we apply a gradient-descent method to obtain a more accurate optimal solution. Simulation results show that our initialization reduces the neural network training time and improves the robustness of gradient-descent algorithms.

Keywords: Genetic Algorithms, Feedforward Neural Networks, gradient-descent search, multilayer perceptron training, optimization, geometrical parametrization of neural networks.

Catégorie: Outils & Techniques (Systèmes hybrides)

[1]Publié dans les actes des Journées internationales sur Les Réseaux Neuromimétiques et leurs Applications. Disponible à l'URL http://apiacoa.org/publications/1994/neuronimes94b.pdf

[2]Les coordonnées actuelles de Fabrice Rossi sont disponibles à l'URL http://apiacoa.org/.

1 Introduction

Les résolutions de problèmes d'apprentissage de réseaux de neurones par descente de gradient sont caractérisées par leur incapacité remarquée de s'échapper d'optima locaux et dans une moindre mesure par leur lenteur. Les algorithmes évolutionnistes apportent dans certains domaines un grand nombre de solutions: entraînement de réseaux à architecture variable ([12]), génération automatique de réseaux de neurones booléens pour la résolution d'une classe de problèmes d'optimisation ([8]). Mais l'effort de recherche s'est surtout porté sur la génération et l'apprentissage de réseaux discrets.

Dans cet article, nous proposons une méthode d'initialisation des algorithmes de descente de gradient par algorithmes évolutionnistes adaptés à l'optimisation sur un espace de recherche non discret. Les *algorithmes évolutionnistes* sont basés sur l'étude du processus d'évolution naturelle. Les principes importants de ce type d'algorithmes sont les suivants:

- L'algorithme travaille sur une population d'individus. Chaque individu correspond à un point de recherche sur un espace de solutions (espace sur lequel on veut obtenir une solution optimale).
- La population est initialisée aléatoirement. Elle évolue par le biais d'opérations telles que la mutation d'un individu ou la recombinaison d'individus.
- L'adaptation d'un individu à son environnement est mesuré grâce à une fonction appelée *force* (*fitness* en anglais) qui associe à chaque individu un réel positif.

Les *algorithmes génétiques* (AG) forment un sous-ensemble des *algorithmes évolutionnistes*. Un individu est entièrement déterminé par un génotype (chromosome), un phénotype et un réel (*force*) [6]. Par analogie avec le processus d'évolution naturelle, un chromosome est souvent une chaîne de bits ou d'entiers. Les opérations de mutation et de recombinaison sont effectuées par des fonctions unaires ou binaires appelées opérateurs. Des études empiriques ont montré que les AG convergeaient vers un optimum global pour une grande classe de problèmes d'optimisation définis sur un ensemble discret [3]. Des études théoriques ont démontré que sous certaines hypothèses de régularité pour la fonction *force*, ces algorithmes convergent asymptotiquement vers des solutions optimales globales (cf. [4]).

L'optimisation par AG de fonctions de $\mathbb{R}^n$ dans $\mathbb{R}$ nécessite un codage des paramètres réels en chaîne d'entiers et entraîne une forte complexité. D'autres algorithmes évolutionnistes n'utilisent pas de génotype. Les opérateurs travaillent alors directement sur le phénotype qui peut avoir une structure de donnée à coordonnées réelles. Une partie de ces algorithmes nommée les *stratégies évolutives* est considérée dans [1].

Dans cette communication, un algorithme évolutionniste travaillant sur des données non-discrètes est proposé. En utilisant la terminologie du domaine de l'évolution artificielle (*evolutionary computation*), on peut considérer que cet algorithme s'apparente à une *stratégie évolutionniste* utilisant un opérateur de recombinaison. Il est utilisé pour initialiser des méthodes de descente de gradient utilisant une rétro-propagation de l'erreur.

2 L'Algorithme Génétique travaillant sur des données réelles (AGDR)

Dans cette partie nous décrivons un algorithme évolutionniste (AGDR) permettant d'obtenir les solutions d'un problème d'optimisation défini sur un espace vectoriel de dimension finie. Pour spécifier notre algorithme, nous devons définir un mécanisme de sélection, un opérateur de recombinaison et un opérateur unaire dit de mutation.

Le mécanisme de sélection

Pour rendre efficace l'implémentation de l'algorithme sur une machine parallèle, nous avons préféré utiliser une méthode de sélection par tournoi. Ainsi comme dans [2], la sélection d'un

individu ne nécessitera que des communications locales entre les processeurs d'une machine parallèle. Les autres modes de sélection couramment utilisés introduisent une forte synchronisation dans l'algorithme car il faut évaluer tous les individus de la population au même moment. Notre mécanisme de sélection tire aléatoirement 3 individus dans la génération précédente et choisit le meilleur individu suivant un critère d'évaluation à définir.

L'opérateur de recombinaison

L'algorithme effectue une recherche sur un espace vectoriel. Il nous a paru essentiel d'utiliser des opérateurs indépendants du choix de la base de cet espace vectoriel. C'est pourquoi nous utilisons l'opérateur appelé communément *l'appariement arithmétique*. Si deux individus I_1 et I_2 sont caractérisés par les vecteurs X et Y, l'opérateur arithmétique construit un descendant I_3 à partir de I_1 et I_2 tel que le vecteur caractérisant I_3 appartiennent au segment $[X;Y]$. D'autres opérateurs de recombinaison sont communément utilisés, une description exhaustive de ces opérateurs se trouve dans [9].

L'opérateur de mutation

Soit un individu I_1 caractérisé par un vecteur $\vec{X}$. Notre opérateur de mutation construit à partir de I_1, un individu I_2 caractérisé par un vecteur $\vec{Y}$ en effectuant une translation sur $\vec{X}$ d'un vecteur $\lambda\vec{U}$ où $\lambda \in [0;D]$ et $\vec{U} \in I\!R^n$ est choisi aléatoirement sur la sphère unité. Le paramètre D est fixé au début de l'algorithme. On a donc $\vec{Y} = \vec{X} + \lambda\vec{U}$

Etape 1 Définir aléatoirement une population initiale $\Pi_{t=0}$ contenant N individus.

Etape 2 Choisir un ensemble de parents et définir une population temporaire $\tilde{\Pi}$.

Etape 3 Répéter N fois les 3 opérations:

- Sélectionner deux individus.
- Produire un couple de descendants avec les parents.
- Mettre un de ces descendants dans $\tilde{\Pi}$.

Etape 4 Comparer le meilleur de $\tilde{\Pi}$ avec le meilleur de Π_t et garder le plus "fort" des deux dans $\tilde{\Pi}$.

Etape 5 $\Pi_{t+1} = \tilde{\Pi}$, $t = t + 1$ et retourner à l'étape 2.

Table 1: Description de l'algorithme

L'algorithme est décrit par la table 1. Cet algorithme est destiné à trouver un bon individu (correspondant à un réseau de neurones) pour initialiser une méthode de descente de gradient. Il est essentiel que cette étape d'initialisation nécessite peu de temps de calcul, et puisse s'implémenter sur une architecture parallèle. La version présentée est séquentielle mais il est facile d'implémenter une version distribuée. La population est divisée en sous-populations et distribuée sur les processeurs. Chaque sous-population peut explorer une région différente de l'espace de recherche. Cela permet d'éviter une convergence prématurée de la population entière vers une solution non optimale. Les processeurs peuvent envoyer leurs meilleurs individus à leurs processeurs voisins et ainsi les empêcher de tomber dans un optimum local. On maintient donc la diversité de la population tout en assurant une convergence globale. Pour une approche théorique des algorithmes génétiques distribués, se reporter à [7].

Pour les algorithmes génétiques, le génotype est généralement une chaîne binaire (S) et les paramètres réels (x) sont codés de la façon suivante:

$$S = a_0 a_1 \dots a_{n-1} \in \{0;1\}^n \longrightarrow x = \frac{1}{2^n} \sum_{i=0}^{n-1} a_i 2^i$$

Le principal problème d'optimisation de fonction définie sur un espace vectoriel par algorithmes

génétiques est la discrétisation de l'espace de recherche. Des études ont été faites pour résoudre ce problème, notamment en élaborant des méthodes de codage dynamique des paramètres réels. Cependant la complexité engendrée rend les algorithmes génétiques peu efficaces. Une alternative a été adoptée par H.-P. Schwefel. Elle est décrite dans les modèles des *stratégies évolutionnistes*. Le modèle le plus simple fait évoluer une population de vecteurs réels. De nouvelles solutions sont créées en utilisant des opérateurs travaillant sur des vecteurs réels et un mécanisme de sélection. Ces méthodes nécessitent en général peu de temps calcul puisque le codage des structures de données réelles en chaîne est rendu inutile.

3 Initialisation par l'algorithme évolutionniste AGDR

3.1 Enoncé du problème

L'apprentissage des réseaux de neurones à coefficients réels peut être considéré comme la minimisation d'une fonction d'erreur mesurée entre la sortie du réseau et un ensemble d'apprentissage. Dans ce cadre, les algorithmes de descente de gradient en version *off-line* constituent des outils très puissants pour de nombreuses applications, ils ne requièrent pas la connaissance a priori de paramètres nécessaires avant la phase d'apprentissage proprement dite. Cependant, un des désavantages principaux de ce type de méthode est le manque de contrôle sur l'optimalité de la solution obtenue. On prend pour exemple les algorithmes utilisant un gradient conjugué qui ont tendance à converger vers l'optimum local le plus proche du point d'initialisation. Pour éviter cette convergence vers une solution sous-optimale, on peut penser à utiliser des algorithmes évolutionnistes qui effectuent une recherche globale. Cependant si l'on veut qu'une recherche globale basée sur l'évolution d'une population d'individus ne soit pas trop coûteuse en temps calcul, on obtient souvent une recherche peu précise. C'est pourquoi l'utilisation des algorithmes évolutionnistes pour l'initialisation de la méthode de descente de gradient du type gradient conjugué semble être un bon compromis pour l'utilisation d'une recherche globale et celle de l'information du gradient. L'ajout d'une recherche globale peut empêcher la convergence vers une solution non optimale.

3.2 Description des simulations

Nous avons testé notre approche sur deux exemples différents.

Pour évaluer la robustesse de cette approche, nous avons considéré le problème de l'approximation d'une fonction $f(x)$ calculée par un perceptron multicouches ayant un neurone d'entrée, un neurone de sortie et 4 neurones dans la couche cachée. Les poids et les seuils de ce réseau ont été fixés pour que la dérivé $\frac{df}{dx}$ s'annule trois fois sur l'intérieur du segment de validation. L'ensemble d'apprentissage contient 30 exemples distribués uniformément sur ce segment. (La figure 1 montre la fonction $f(x)$ sur l'ensemble d'apprentissage). Dans ce cas simple on a observé que le plus petit perceptron multicouches (i.e., avec le plus petit nombre d'unités et le plus petit nombre de couches) qui pouvait approcher précisément $f(x)$ possédait au moins 3 neurones dans une unique couche cachée. Les simulations montrent que l'approximation de $f(x)$ avec un réseau ayant 4 unités cachées est un problème difficile pour un algorithme de descente de gradient en version *off-line*.

Pour comparer notre approche avec d'autres, nous avons choisi comme fonction à apprendre la fonction utilisée dans [13]. Une description de cette fonction $g(x)$ et de l'ensemble d'apprentissage est fournie par la figure 2. $g(x)$ est définie par:

$$\begin{array}{rcll} g(x) & = & -2.186x - 12.864 & si \ -10 \leq x \leq -2 \\ & = & 4.246x & si \ -2 \leq x \leq 0 \\ & = & 10e^{-0.05x-0.5} sin((0.03x + 0.7)x) & si \ 0 \leq x \leq 10 \end{array}$$

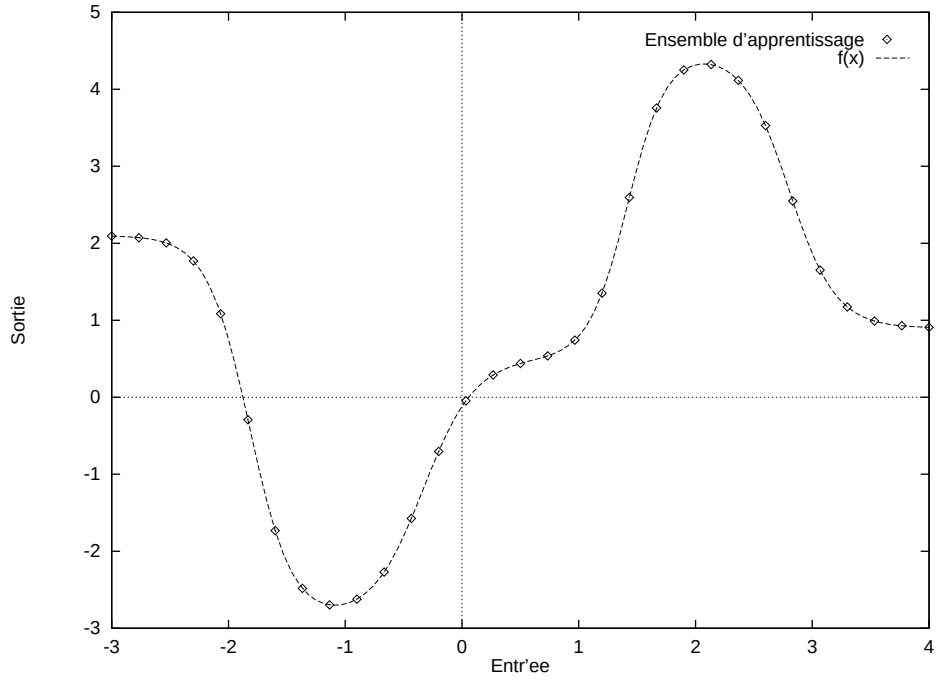

Figure 1: Sortie d'un perceptron avec 4 unités cachées.

La sortie d'un perceptron ayant deux couches avec un neurone linéaire dans la dernière couche est donnée par:

$$f(\vec{x}) = \sum_{i=1}^{n} \gamma_i \sigma(\vec{\alpha}_i . \vec{x} + \beta_i) + \gamma_0, \tag{1}$$

où $\vec{x}$ est l'entrée du réseau et σ est la fonction de transfert des neurones cachés. Comme dans [13] et [10], nous proposons d'utiliser une méthode de paramétrisation géométrique. Nous réécrivons l'équation 1 en:

$$f(\vec{x}) = \sum_{i=1}^{n} \gamma_i \sigma \left(\vec{d}_i . (\vec{x} - \vec{t}_i) \right) + \gamma_0 \tag{2}$$

La méthode de descente de gradient par gradient conjugué de Polak-Ribière (PRCG) initialisée aléatoirement ou avec l'AGDR a été testée sur les deux types de paramétrisation. Les résultats des simulations sont exposés dans la table 3.

3.3 Résultats des simulations

L'algorithme de descente de gradient de Polak-Ribière est reconnu comme étant rapide et efficace pour de nombreux problèmes d'apprentissage ([5]). Nous l'avons testé sur 4 types de perceptrons différents: avec 3, 4 et 5 unités cachées. Nous avons limité le nombre maximum d'itérations de l'algorithme à 1000. Une exécution est considérée comme *convergente* si l'erreur quadratique moyenne tombe en dessous de 0.1 avant les 1000 itérations. Aucune convergence vers un réseau dont la sortie approchait $f(x)$ n'a été obtenue avec 3 unités cachées.

Pour l'algorithme évolutionniste, la probabilité d'utilisation de l'opérateur de mutation est de 0.6 et celui de l'opérateur de recombinaison est 0.4. La population contient 100 individus et l'algorithme effectue 50 générations.

Grace à l'utilisation de l'opérateur de recombinaison chaque nouvelle génération d'individus reste proche de l'enveloppe convexe de la génération précédente (on insiste sur le fait que l'ensemble des descendants est inclus dans l'ensemble des segments que l'on peut former à l'aide des individus de la génération précédente). La population reste donc cantonnée dans un espace de recherche

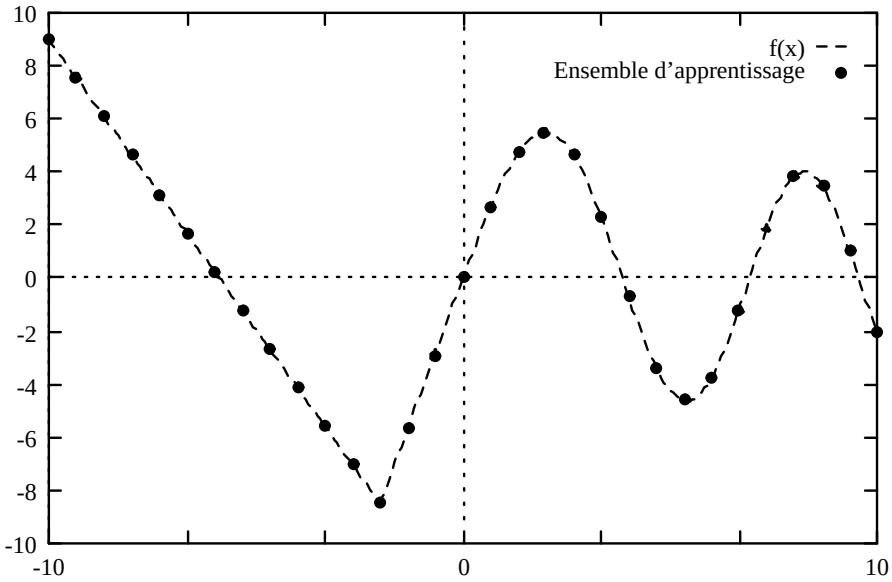

Figure 2: Fonction à apprendre dans la deuxième simulation.

fixé au départ. Ceci permet de limiter le risque d'une divergence des valeurs des connexions des réseaux. Le critère d'évaluation choisi pour le mécanisme de sélection est le suivant: le "meilleur" individu parmi les 3 individus tirés aléatoirement est celui ayant la plus petite erreur quadratique moyenne sur l'ensemble d'apprentissage. Ainsi l'algorithme AGDR fournit un réseau avec une faible erreur quadratique, l'algorithme de descente de gradient de Polak-Ribière est appliqué sur ce réseau.

Initialisation	Aléatoire	avec AGDR
4 unités cachés	8042 (40% de convergence)	5628 (72% de convergence)
5 unités cachés	6006 (86% de convergence)	4463 (95% de convergence)

Table 2: *Evaluations* moyennes sur 1000 exécutions

Les résultats sont exposés dans la table 2. On a mesuré pour ces expériences que le temps de calcul nécessaire pour calculer le gradient d'un réseau par la méthode de rétro-propagation est égale à environ 4 fois le temps de calcul nécessaire pour calculer la sortie de ce même réseau. On désigne donc par *évaluation* une unité de complexité qui est égale au nombre d'opérations nécessaires pour calculer la sortie des perceptrons utilisés. Cette évaluation permet de comparer les complexités de chaque algorithme. Sachant que l'algorithme AGDR avec le critère de sélection énoncé plus haut n'effectue aucun calcul de gradient et que l'on a négligé le coût apporté par les opérateurs génétiques (l'erreur engendrée sur les résultats de complexité énoncés est alors inférieure à 5%), on mesure qu'une exécution de AGDR avec ce critère de sélection est équivalente à 4000 *évaluations*. Ainsi les résultats expérimentaux montrent que l'initialisation par AGDR diminue le nombre d'exécutions divergentes mais augmente la complexité du processus d'apprentissage.

Dans la deuxième simulation la probabilité d'utilisation de l'opérateur de mutation est de 0.9 et celui de l'opérateur de recombinaison est de 0.8. La population contient toujours 100 individus et l'algorithme effectue 50 générations. Les critères de convergence retenus dans cette ensemble de simulations sont les mêmes que ceux du premier ensemble de simulations.

On a utilisé un autre critère de sélection. En effet pour diminuer le nombre d'itérations nécessaires à la convergence de l'algorithme de descente de gradient, il faut éliminer les individus qui correspondent à des minima locaux et dont le gradient a une norme presque nulle. Le mécanisme de

sélection utilisé dans la deuxième simulation est le suivant: sur les 3 individus choisis aléatoirement, on ne garde dans un premier temps que les deux meilleurs, puis on calcule le gradient des deux individus sur un exemple tiré aléatoirement dans l'ensemble d'apprentissage, on choisit enfin l'individu dont le gradient a la plus petite norme. La population contient 100 individus. Puisque l'AGDR effectue 50 générations il est impossible de calculer le gradient sur tout l'ensemble d'apprentissage pour chaque sélection sans augmenter drastiquement la complexité de l'algorithme. C'est pourquoi on se restreint à ne calculer le gradient que sur un exemple. Une exécution de l'algorithme AGDR correspond alors à 333 calculs de gradient total. Le nombre de calculs de la fonction de sortie des perceptrons correspondant aux individus de la population est égal à 5000. Une exécution de l'AGDR est donc équivalente à 6200 *évaluations*. Pour comparer ce résultat avec la complexité de l'algorithme de gradient de Polak-Ribière, ajoutons que 100 itérations de cet algorithme correspond à plus de 14000 *évaluations* en moyenne (l'algorithme effectue environ 10 minimisations unidimensionnelles et calcule une fois le gradient à chaque itération).

Les neurones utilisés possèdent une fonction de transfert de type sigmoïde avec un point d'inflexion coïncidant avec le point d'annulation de la fonction de sortie du neurone. Pour les deux simulations, on a remarqué que les points d'inflexion des neurones des réseaux obtenus par l'algorithme AGDR sont bien répartis sur l'ensemble d'apprentissage (ils sont assez proches de ceux de la fonction à apprendre) mais les régions saturantes de ces neurones ne sont pas positionnées avec suffisamment de précision.

D'autres méthodes d'initialisations ont été étudiées dans la littérature ([11], [13]). Ces méthodes positionnent les points d'inflexion des fonctions de sortie des neurones de couches cachées. Le réseau ainsi obtenu n'est pas un minimum local de la fonction d'erreur quadratique. Nous avons testé ces méthodes d'initialisation. Celle qui a donné les meilleures résultats pour le deuxième ensemble de simulations n'a fait converger l'algorithme de Polak-Ribière qu'après 916 itérations. Ces méthodes ont été utilisées pour la paramétrisation géométrique ([13] fournit une bonne description de ces méthodes pour les réseaux d'ondelettes).

Initialisation Paramétrisation	Aléatoire Standard	avec AGDR Standard	Aléatoire Géométrique	avec AGDR Géométrique
Convergence avant 200 itérations	0	1	8	26
Nombre moyen d'itérations (exécutions convergentes)	567 (14)	496 (13)	228 (13)	81.40 (30)

Table 3: Résultats de la deuxième simulations pour 30 exécutions

4 Conclusion

Nous avons présenté un algorithme évolutionniste (AGDR) adapté à l'optimisation de fonctions sur $\mathbb{R}^n$. Cet algorithme a été utilisé pour initialiser des méthodes de descente de gradient. Certaines de ces méthodes, notamment les gradients conjugués, convergent rapidement et précisément vers une solution optimale si la solution initiale fournie ne se trouve pas à proximité d'optima locaux. L'emploi de notre algorithme pour l'obtention d'un "bonne" solution initiale se montre très efficace pour l'apprentissage des réseaux de neurones du type perceptron multicouches. Néanmoins l'intérêt de la méthode d'initialisation proposée est sa portabilité pour l'apprentissage des réseaux de neurones non-récurrents en général. En effet l'algorithme AGDR n'utilise aucune propriété sur l'architecture du réseaux.

On obtient donc un algorithme hautement parallélisable fournissant de bonnes solutions pour une grande classe de problèmes d'optimisation. Généralement les solutions obtenues sont proches des solutions optimales mais l'algorithme n'a pas pu converger vers une solution optimale avec précision. C'est pourquoi l'ajout de méthodes d'optimisation effectuant une descente de gradient

permet d'obtenir une méthode robuste et précise de génération automatique de réseaux de neurones à architecture fixée. Des travaux futurs sont envisagés pour appliquer ce type d'approche à l'apprentissage de réseaux à architecture variable et à l'entraînement de réseaux récurrents.

References

[1] T. Bäck and H.P. Schwefel. An Overview of Evolutionary Algorithms for Parameters Optimization. *Evolutionary Computation*, 1(1), 1993.

[2] J. Collins and D.R. Jefferson. Selection in Massively Parallel GA. In *Fourth Proceedings on GA*, 1991.

[3] L. Davis. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[4] A. E. Eiben, E. H. L. Aaarts, and K. M. Van hee. Global Convergence of Genetic Algorithms: A Markov chain analysis. In H. P. Schwefel and Eds R. Männer, editors, *Parallel Problem Solving from Nature*, 1991.

[5] M. Fombellida and J. Destiné. Méthodes heuristiques et méthodes d'optimisation non contraintes pour l'apprentissage des perceptrons multicouches. In *Neuro-Nîmes*, 1992.

[6] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[7] F. Gruau. The mixed parallel genetic algorithm. In *PARCO93*, 1993.

[8] F. Gruau and D.Whitley. Adding learning to the cellular developmental process: a comparative study. *Evolutionary Computation*, 1(3), 1993.

[9] Z. Michalewicz. A Hiearchy of Evolution Programs: An Experimental Study. *Evolutionnary Computation*, 1, 1993.

[10] Fabrice Rossi and Cédric Gégout. Geometrical Initilization, Parametrization and Control of Multilayer Perceptrons: Application to Function Approximation. In *Proceedings of WCCI ICNN*, volume I, pages 546–550, Orlando (Florida), June 1994. IEEE.

[11] L. Wessels and E. Barnard. Avoiding False Local Minima by Proper Initialisation of Connections. *IEEE Trans. on Neural Networks*, 3(6), Novembre 1992.

[12] D. Withley, T. Starkweather, and C. Bogart. Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity. *Parallel Computing*, 14, 1990.

[13] Q. Zhang and A. Benveniste. Wavelet networks. *IEEE Trans. On Neural Networks*, 3(6), November 1992.