Optimisation incrémentale de réseaux de neurones RBF pour la régression via un algorithme évolutionnaire : RBF-Gene

Virginie LEFORT *, Guillaume BESLON **

*Laboratoire ERIC, Université Lumière Lyon 2, 5 avenue Pierre Mendès-France, 69676 Bron Cedex, France, virginie.lefort@univ-lyon2.fr
**Laboratoire LIRIS, UMR 5205 CNRS, Bâtiment Blaise Pascal, INSA de Lyon, 69621 Villeurbanne Cedex FRANCE, guillaume.beslon@liris.cnrs.fr

Résumé. Les réseaux de neurones RBF sont d'excellents régresseurs. Ils sont cependant difficiles à utiliser en raison du nombre de paramètres libres : nombre de neurones, poids des connexions, ... Des algorithmes évolutionnaires permettent de les optimiser mais ils sont peu nombreux et complexes. Nous proposons ici un nouvel algorithme, RBF-Gene, qui permet d'optimiser la structure *et* les poids du réseau, grâce à une inspiration biologique. Il est compétitif avec les autres techniques de régression mais surtout l'évolution peut choisir dynamiquement le nombre de neurones et la précision des différents paramètres.

1 Introduction

Les réseaux de neurones supervisés sont d'excellents régresseurs permettant à la fois de classer des données ou de trouver des relations entre des entrées et des sorties (régression). Cependant ils sont bien souvent durs à mettre en œuvre de part le nombre important de paramètres. L'utilisation d'algorithmes évolutionnaires (en particulier les algorithmes génétiques) permet de faciliter la mise en œuvre de ces réseaux, en définissant leur structure ou les poids des connexions. Grâce à une inspiration fortement biologique, nous proposons un nouvel algorithme, RBF-Gene, qui permet une optimisation incrémentale d'un réseau RBF (structure et connexions) de manière efficace.

2 Algorithmes évolutionnaires et réseaux de neurones

Dans un réseau de neurones chaque neurone réalise un traitement simple et ce sont le nombre de neurones et leur connectivité qui vont faire toute la puissance du réseau. Dans les réseaux de neurones dits "en couche", le réseau est constitué de trois sous-ensembles de neurones : les neurones d'entrée, les neurones cachés (complètement connectés aux neurones d'entrée ou, si le réseau possède plus d'une couche cachée, à la couche précédente) et les neurones de sortie connectés à la dernière couche de neurones cachés.

Les réseaux RBF (Poggio et Girosi, 1989), pour "Radial Basis Function", sont des réseaux à une couche cachée dont les neurones cachés utilisent une fonction de transfert gaussienne tandis que les neurones de sortie réalisent une "simple" somme pondérée des réponses des

neurones cachés. Ces réseaux sont des approximateurs universels (Park et Sandberg, 1991) et sont très efficaces pour des tâches de classification ou de régression. Leur utilisation passe par le choix d'un grand nombre de paramètres libres (le nombre de neurones cachés, leurs paramètres – moyenne et écart-type – et les poids de leur combinaison linéaire). Or ces paramètres sont tous interdépendants ce qui rend leur détermination difficile.

Les algorithmes évolutionnaires (AE) ont souvent été employés pour paramétrer des réseaux de neurones. Ils recouvrent différentes techniques inspirées de l'évolution biologique : algorithmes génétiques, programmation génétique, stratégies d'évolution, évolution grammaticale, etc... Cependant, le principe général est toujours le même : une population de solutions est générée aléatoirement puis va successivement subir une phase de sélection puis de reproduction jusqu'à un critère de terminaison.

On peut classer les AEs permettant l'optimisation de réseaux de neurones en trois catégories. Les optimiseurs de poids (Blanco et al., 2001) permettent d'optimiser les poids d'un réseau dont la structure a été fixée préalablement. Les optimiseurs de structure (MacLeod et Maxwell, 2001; Barrios et al., 2001) permettent de tester différentes structures pour le réseau de neurones (un second algorithme étant utilisé pour déterminer les poids). Enfin, les optimiseurs de structure *et* de poids (Arotaritei et Negoita, 2002; Kuşçu et Thornton, 1994) permettent d'optimiser simultanément la structure du réseau et les poids des connexions.

Dans ces derniers algorithmes, la représentation choisie est souvent de taille variable, une partie correspondant à la structure du réseau (nombre de neurones cachés et leur placement) et une autre aux poids des connexions entre les neurones cachés et les sorties. Or la structure d'un réseau a une forte influence sur le nombre de connexions et donc de poids. Toute mutation dans la partie "structure" doit entraîner une modification de la taille de la partie "connexions". L'évolution va alors se faire en deux temps : tout d'abord l'évolution de la première partie et son décodage, puis l'évolution de la seconde en fonction des informations de la première. Des processus spécifiques doivent être mis en œuvre pour maintenir la cohérence des génomes, en particulier pour les opérations de croisement (crossover).

3 Le modèle RBF-Gene

Nous proposons un nouvel algorithme, RBF-Gene permettant d'optimiser à la fois la structure d'un réseau RBF et ses connexions. RBF-Gene est inspiré des mécanismes biologiques de traduction, en particulier l'existence d'un niveau d'organisation intermédiaire entre le génotype et le phénotype (le protéome) et le code génétique permettant le "calcul" de ce niveau intermédiaire. L'analogie entre la biologie et notre algorithme est simple : dans les deux cas, nous voulons obtenir un phénotype (une solution) constitué de l'assemblage d'unités homogènes (protéines ou neurones cachés), qui doivent être en nombre variable, et pour lesquelles l'ordre ou la place du gène correspondant sur le génome n'a pas d'importance.

La structure du réseau (le nombre de neurones) est déduite du protéome. Nous pouvons donc augmenter ou diminuer facilement le nombre de neurones au cours de l'évolution, et toujours obtenir des réseaux valides. Un autre avantage de la présence du protéome est qu'il permet de diminuer les contraintes sur la structure du génome : les gènes peuvent se déplacer, se dupliquer, disparaître... L'ordre des gènes peut donc se modifier au cours du temps pour rapprocher des gènes liés, et la taille des séquences non codantes (entre les gènes) peut s'adapter dynamiquement aux opérateurs.

Nos individus possèdent donc trois niveaux d'organisation : un génome composé d'un ensemble de gènes détectés localement, un protéome composé de l'ensemble de nos protéines virtuelles (des neurones cachés entièrement définis), et un phénotype qui est le réseau RBF construit à partir du protéome. Le passage du protéome au phénotype est trivial : le réseau construit est simplement l'assemblage des différents neurones cachés. Nous nous intéresserons donc surtout au passage du génome au protéome, et aux différents opérateurs génétiques. Une présentation détaillée de l'algorithme peut être trouvée dans (Lefort, 2007).

3.1 Code génétique et mapping génotype-protéome

Chacun de nos gènes code pour un neurone caché et la séquence de ce gène doit définir totalement ce neurone de la même façon qu'un gène détermine totalement la séquence d'une protéine. Ceci est rendu possible par l'utilisation d'un code génétique qui va transformer le contenu du gène en une suite d'acides aminés, celle-ci étant ensuite décodée pour calculer les paramètres du neurone : la moyenne (qui est un vecteur dont la dimension est celle des entrées), l'écart-type et le vecteur de poids de sortie.

Nous avons choisi de coder chacun de ces paramètres avec un code binaire dont la longueur pourra évoluer, de manière à pouvoir avoir des valeurs grossières en début d'évolution (gènes courts) et, si nécessaire, précises en fin d'évolution (gènes longs).

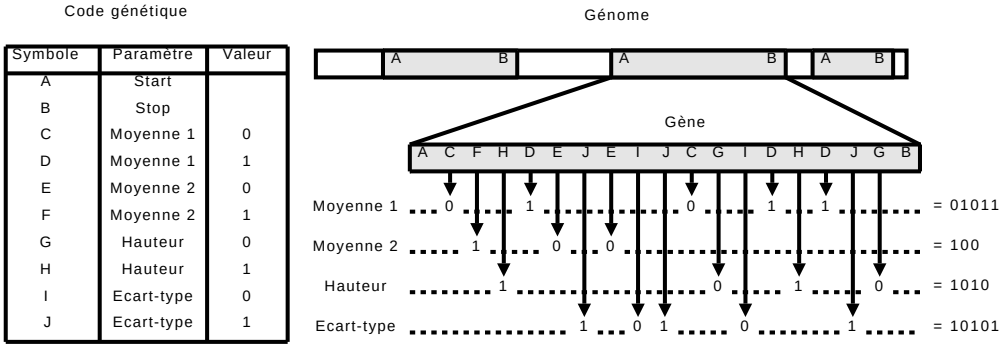

FIG. 1 – *Exemple de mapping génome - protéome. Pour 2 entrées et une sortie, nous avons besoin de 10 bases différentes. Les gènes sont délimités par "start" (A) et "stop" (B). Ensuite, à partir du code génétique, on reconstitue le code binaire de chaque caractéristique du neurone caché. Par exemple, les bases C et D, nous permettent de déduire que la première moyenne aura pour code binaire 01011.*

Notre génome est constitué d'une suite de lettres (ou bases), deux lettres particulières (A pour "start" et B pour "stop") permettant de délimiter les gènes, les autres étant utilisées par paires pour coder les différents paramètres. Le nombre de lettres nécessaires dépend du problème : outre le "start" et "stop", il faut deux lettres pour chaque caractéristique du neurone (une pour représenter un 0 binaire, et une pour le 1). Ainsi, sur un problème avec une entrée et une sortie, nous avons besoin de 8 lettres.

Pour reconstituer la valeur d'un paramètre, il suffira de rechercher les deux bases correspondantes dans le gène, de les extraire (les différents paramètre pouvant être mixés) et de les transformer en suite de 0 et de 1. Il suffit ensuite de normaliser la valeur binaire pour obtenir une valeur réelle (figure 1).

3.2 Opérateurs

Notre algorithme conserve la boucle générationnelle des algorithmes évolutionnaires. Cependant la structure génétique nous autorise à élargir le répertoire des opérateurs de mutation puisque ceux-ci peuvent modifier les gènes et/ou la structure du génome.

Nous avons sept opérateurs de mutations. Trois opérateurs locaux n'agissent que sur une seule base à la fois et vont principalement modifier les valeurs des paramètres : le switch (qui remplace une base par une autre), la délétion ponctuelle (qui supprime une base) et l'insertion (qui en rajoute une). Trois opérateurs globaux agissant sur des segment génétiques (qu'ils contienne des gènes ou non) permettent des remaniements de la structure génétique. Il s'agit de la translocation (qui déplace une zone à un autre endroit du génome), de la duplication (qui insère une copie d'une zone dans le génome) et de la délétion large (qui supprime une zone du génome). Enfin, nous avons un crossover à un point, qui sera effectué après un alignement à gauche des génomes pour permettre un échange de gènes.

4 Étude en régression

Nous avons étudié notre algorithme sur des benchmarks de régression, afin de vérifier tout d'abord la facilité de paramétrage, puis les résultats en convergence et enfin le déroulement de l'évolution. Le benchmark utilisé ici est Boston Housing (Automatic Knowledge Miner (AKM) Server, 2003), qui possède 13 entrées pour une sortie. Il est constitué de 506 points (405 pour l'apprentissage et 101 pour la validation). L'algorithme a été testé sur d'autres benchmarks (1D et 2D Sine Wave, Abalone), les résultats en convergence pouvant être trouvés dans (Lefort, 2007).

Le paramétrage n'est pas détaillé ici, mais nous utilisons les réglages que l'on peut trouver dans (Lefort, 2007). On notera que notre algorithme ne possède que deux paramètres qui ne soient pas classiques aux algorithmes évolutionnaires (taux des mutations larges et taille initiale des génomes).

Même si la comparaison de notre algorithme avec d'autres algorithmes de régression n'est pas le cœur de ce travail, notre algorithme a montré qu'il était compétitif avec les principaux algorithmes de régression utilisés dans (Madigan et Ridgeway, 2004). Les détails des comparaisons se trouvent dans (Lefort, 2007).

Nos solutions au terme de l'évolution sont compétitives avec d'autres algorithmes de régression. Notre algorithme possède cependant de nombreux degrés de liberté (nombre de neurones, taille des gènes, taille des génomes...) qu'il est intéressant d'étudier. Nous allons nous intéresser à deux indicateurs : la taille du génome et le taille moyenne des gènes. Les évolutions de ces indicateurs sont représentées figure 2.

On remarque que dans une toute première phase, la taille du génome augmente de manière exponentielle. Cependant, et malgré l'absence de processus limitant la taille, celle-ci se stabilise à 15000 bases (la valeur de stabilisation dépendant du problème et du taux de mutation).

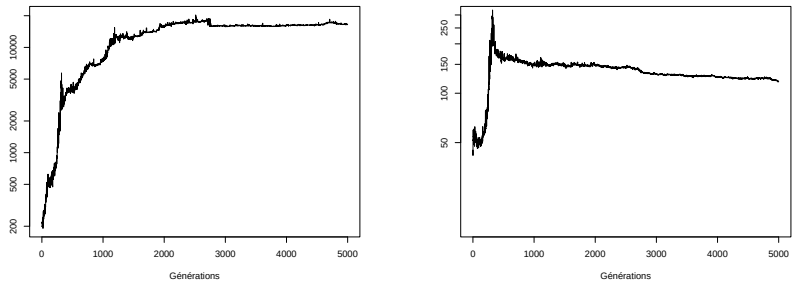

FIG. 2 – *Évolution de la taille des génomes (à gauche) et de la taille des gènes (à droite). Il s'agit de moyennes effectuées sur les meilleures individus de 10 simulations.*

Contrairement à d'autres algorithmes (de type programmation génétique) nous n'avons donc pas une explosion de la taille.

Cette augmentation de la taille est due à deux facteurs. Le premier est l'augmentation du nombre de gènes (de 2 à 80 par génome) ce qui permet de résoudre efficacement le problème Boston Housing. Le deuxième correspond à la taille des gènes qui reste stable dans les toutes premières générations puis augmente fortement pour se stabiliser vers 150 bases. Cette augmentation correspond à une amélioration de la précision du codage : à la convergence, chaque paramètre est codé en moyenne sur dix bits.

Notre algorithme gère donc de façon dynamique le nombre de gènes comme leur taille (et donc le nombre de neurones cachés et la précision des paramètres), notre précision tout comme notre nombre de neurones restant cohérents avec le problème posé.

5 Conclusion

En nous inspirant de la biologie, nous avons proposé un algorithme évolutionnaire qui permet d'optimiser des réseaux RBF pour la régression. Notre génome est homogène et composé de gènes indépendants les uns des autres et pouvant se déplacer, se dupliquer ou changer de taille. Chacun de ces gènes représente un neurone caché complètement défini, dont les paramètres seront issus uniquement du contenu du gène, transformé via un code génétique artificiel en un code binaire de taille variable.

L'évolution artificielle va donc pouvoir optimiser les valeurs des différents paramètres, mais aussi la précision de chacun d'entre eux, le nombre de gènes et leur disposition sur le génome. Nos résultats sont compétitifs avec d'autres algorithmes dédiés à la régression, mais surtout notre algorithme permet un choix dynamique de la taille du génome ou de celle des gènes, qui se stabilisent toutes les deux. Il serait maintenant intéressant de l'appliquer à des problèmes réels plus complexes pour tester ses limites.

Références

Arotaritei, D. et M. G. Negoita (2002). Optimization of recurrent nn by ga with variable length genotype. *LNAI (Springer-Verlag) AI 2002*(2557), 681–692.

Automatic Knowledge Miner (AKM) Server (2003). Data mining analysis (request abalone). Technical report, AKM (WEKA), University of Waikato, Hamilton, New Zealand.

Barrios, D., D. Manrique, R. M. Plaza, et R. Juan (2001). An algebraic model for generating and adapting neural networks by means of optimization methods. *Annals of Mathematics and Artificial Intelligence 33*(1), 93–111.

Blanco, A., M. Delgado, et M. Pegalajar (2001). A real-coded genetic algorithm for training recurrent neural networks. *Neural Networks 14*(1), 93–105.

Kuşçu, I. et C. Thornton (1994). Design of artificial neural networks using genetic algorithms: review and prospect. Technical Report 319, Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton, Sussex, UK.

Lefort, V. (2007). *Evolution de second ordre et algorithmes évolutionaires : l'algorithme RBF-Gene*. Ph. D. thesis, Institut National des Sciences Appliquées (INSA) de Lyon, Villeurbanne, France.

MacLeod, C. et G. M. Maxwell (2001). Incremental evolution in anns: Neural nets which grow. *Artificial Intelligence Review 16*(3), 201–224.

Madigan, D. et G. Ridgeway (2004). Discussion of "least angle regression" by efron et al. *The Annals of Statistics 32*, 465–469.

Park, J. et I. Sandberg (1991). Universal approximation using radial-basis-function networks. *Neural Computation 3*(2), 246–257.

Poggio, T. et F. Girosi (1989). A theory of networks for approximation and learning. Technical Report AIM-1140, Massachusetts Institute of Technology (MIT), Artificial Intelligence Laboratory, Cambridge, MA, USA.

Summary

RBF neural networks are a powerfull regression tool. But they are often difficult to use because they need a lot of choices : neurons number, their center and standard deviation, and the weight of the connections. Evolutionary algorithms can optimise these networks, but they are few of them and complex to use. We propose here a new algorithm, RBF-Gene, what can optimise both the structure and the weights of such networks, thanks to a biological analogy: our genome is made of genes (encoding for a complete hidden neuron) in variable order and place. The whole will constitute our network. We will see that our algorithm is competitive with other regression techniques and that evolution can choose dynamically the neurons number and the precision of the parameters.