Memetic Algorithms to Product-Unit Neural Networks for Regression[1]

Francisco Martínez-Estudillo[1], César Hervás-Martínez[2], Alfonso Martínez-Estudillo[1], and Domingo Ortíz-Boyer[2]

[1] Department of Management and Quantitative Methods, ETEA, Spain
{acme, fjmestud}@etea.com
[2] Department of Computing and Numerical Analysis of the University of Córdoba, Spain
{chervas, ma1orbod}@uco.es

Abstract. In this paper we present a new method for hybrid evolutionary algorithms where only a few best individuals are subject to local optimization. Moreover, the optimization algorithm is only applied at specific stages of the evolutionary process. The key aspect of our work is the use of a clustering algorithm to select the individuals to be optimized. The underlying idea is that we can achieve a very good performance if, instead of optimizing many very similar individuals, we optimize just a few different individuals. This approach is less computationally expensive. Our results show a very interesting performance when this model is compared to other standard algorithms. The proposed model is evaluated in the optimization of the structure and weights of product-unit based neural networks.

1 Introduction

Evolutionary algorithms (EAs) are efficient at exploring the entire search space; however, they are relatively poor at finding the precise optimum solution in the region where the algorithm converges to [1]. During the last few years new methods have been developed in order to improve the lack of precision of the EAs using local optimization algorithms [2]. These new methodologies are based on the combination of local optimization procedures, which are good at finding local optima (local exploiters), and evolutionary algorithms (global explorers). These are commonly known as hybrid algorithms.

In this paper, we propose an alternative approach to hybrid algorithms using local optimization procedures. The methodology is based on the combination of an evolutionary algorithm, a clustering process and a local improvement procedure. If we want to efficiently use the hybrid algorithm, we have to reduce the computation time spent by the local search. So, our approach is to select a subset of the best individuals, perform a cluster analysis to group them, and optimize only the best individual of every group.

[1] This work has been financed in part by the TIC2002-04036-C05-02 project of the Spanish Inter-Ministerial Commission of Science and Technology (CICYT) and FEDER funds.

The main advantage of this method is that the computational cost of applying the optimization algorithm to just a few individuals hardly affects the total time spent by the algorithm. On the other hand, the use of a clustering algorithm allows the selection of individuals representing different regions in the search space. In this way, the optimized individuals are more likely to converge towards different local optima. The model developed is applied to the evolution of the structure and weights of evolutionary product-unit based neural networks. These kinds of networks are a very interesting alternative to sigmoid-based neural networks, but their major drawback is that the optimization algorithms usually used for training a network are quite inefficient for training product-unit networks. So, an effective algorithm for training these networks is of great practical interest. In order to test the performance of the proposed algorithms, the networks are applied to benchmark regression problems.

This paper is organized as follows: Section 2 describes our model in depth; Section 3 is dedicated to a short description of product-unit based networks; Section 4 states the most relevant aspects of the evolution of product-unit neural networks using the proposed approach; Section 5 explains the experiments carried out; and finally Section 6 summarizes the conclusions of our work.

2 Hybrid Evolutionary Programming Algorithms

We propose two methods of hybrid evolutionary algorithms based on the use of a clustering algorithm for deciding which individuals are subject to local optimization. We have two different versions for the hybrid evolutionary algorithm depending on the stage when we carry out the local search and the cluster partitioning. The hybrid evolutionary programming with the clustering (HEPC) algorithm applies the clustering process on a large enough subset of the best individuals of the final population. In this method it is very important to determine the number of best individuals to consider as well as the number of clusters. After that, we apply the L-M algorithm to the best individual of each cluster. The algorithm named dynamic hybrid evolutionary programming with clustering (Dynamic HEPC) carries out both the clustering process and the L-M local search dynamically every G_0 generation. The final solution is the best individual among the local optima found during the evolutionary process. The basic aim of our methodology is the optimization of the number of times a local optimization algorithm is applied without reducing the efficacy of this algorithm. This is especially important when the algorithm is of a high computational cost. On the other hand, removing the local optimization procedure usually yields a worse performance. So, our method is a good compromise, as we applied the optimization algorithm to a reduced number of individuals. Moreover, the clustering process avails us with the possibility of selecting a subset of individuals with different features. In this way, the optimization algorithm is more efficient.

The local optimization algorithm used in our work is the Levenberg- Marquardt (L-M) optimization method. This algorithm is designed specifically for minimizing a sum-of-squares error [3]. In any case, any other local optimization algorithm can be used in a particular problem. Another feature of our approach is that the optimized individuals are not included in the new population. Once the optimization algorithm is applied, we think that any further modification of the individual would be counter-productive. So, these individuals are stored in a separate population till the end of the evolutionary algorithm.

3 Product-Unit Neural Networks

In order to test the validity of our model we have chosen a difficult problem, hard enough to justify the use of complex approaches. The problem is the automatic determination of the structure and weights of product-unit neural networks [4]. Product units enable a neural network to form higher-order combinations of inputs, having the advantages of increased information capacity and smaller network architectures when we have interaction between the input variables. Neurons with multiplicative responses can act as powerful computational elements in real neural networks. Product-unit based neural networks have a major drawback, since their training is more difficult than the training of standard sigmoid based networks. Unfortunately, the error surface for product units can be extremely convoluted, with numerous minima that trap backpropagation. This is because small changes in the exponents can cause large changes in the total error. Several efforts have been made to develop learning methods for product units [5],[6],[7]. Let us consider the family of real functions F defined by:

$$F = \left\{ f : A \subset \mathbb{R}^k \to \mathbb{R} : f(\mathbf{x},\boldsymbol{\theta}) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right) \right\}$$

where $\mathbf{x} = (x_1, x_2, ..., x_k)$, $\boldsymbol{\theta} = (\beta_j, w_{ij})_{i,j}$, $\beta_j \in [-M, M] \subset \mathbb{R}$, $w_{ij} \in [-L, L] \subset \mathbb{R}$, $i = 1,2,...,k$, $j = 1,2...,m$ y $m \in \mathbb{N}$. The domain of the definition of f is the subset A of $\mathbb{R}^k$ given by $A = \{(x_1, x_2, ..., x_n) \in \mathbb{R}^k : 0 < x_i \le k_0\}$. Using the Stone-Weierstrass Theorem it is straightforward to prove that this family of functions is a dense subset of the continuous functions space defined in a compact. On the other hand, every function of the family can be represented as a neural network. The network must have the following structure: an input layer with a node for every input variable, a hidden layer with several nodes, and an output layer with just one node. There are no connections between the nodes of a layer and none between the input and output layers either. The network has k inputs that represent the independent variables of the model, m nodes in the hidden layer, and one node in the output layer. The activation of j-th node of the hidden layer is given by $B_j(\mathbf{x}, \mathbf{w_j}) = \prod_{i=1}^{k} x_i^{w_{ji}}$ where w_{ji} is the weight of the connection between input node i and hidden node j. The activation of the output node is given by $\sum_{j=1}^{m} \beta_j B_j(\mathbf{x}, \mathbf{w_j})$ where β_j is the weight of the connection between the hidden node j and the output node. The transfer function of all nodes is the identity function.

4 Hybrid Evolutionary Model

The evolution of product-unit networks uses the operations of replication and two types of mutation: structural and parametric. There is no crossover, as this operation is usually regarded as harmful [8] for network evolution. In the following paragraphs we are going to describe each one of the different aspects of the algorithm in detail.

Initial population. We generate randomly $10N_P$ networks and we select the best N_P among them. So, we construct the initial population B whose size is N_P.

Selection plan. The $r\%$ best individuals of the population are selected from the set $B^* = B - \{\text{best individual of } B\}$ of cardinality $N^* = N_P - 1$. With these individuals we make up the population B' of size $\lfloor rN_P/100 \rfloor$.

Structural and parametric mutations. Every individual of the population B' is subject to structural mutation, obtaining B'_{struc}. The parametric mutation is applied only to the best $(100-r)N_p^*/100$ individuals of B' obtaining B'_{param}. We construct the population $B'' = B'_{struc} \cup B'_{param}$, where the cardinality of B'' is $N^* = N_P - 1$.

The structural mutation implies a modification in the structure of the function performed by the network and allows an exploration of different regions in the search space. There are five different structural mutations: node addition, node deletion, connection addition, connection deletion and node fusion. Parametric mutation is accomplished for each coefficient of the model with Gaussian noise, using a self-adaptive annealing algorithm. For more details see [9].

Updated plan. The new population will be $B = B'' \cup \{\text{Best of } B\}$.

4.1 Clustering Partitioning Techniques

Let $D = \{(\mathbf{x}_l, y_l) : l = 1,2,..,n_T\}$ be the training data set, where the number of samples is n_T. We define the following application from the family of functions F to the Euclidean space $\mathbb{R}^{n_T}$:

$$H : F \to \mathbb{R}^{n_T}$$
$$f(\mathbf{x},\boldsymbol{\theta}) \to H\left(f(\mathbf{x},\boldsymbol{\theta})\right) = \hat{y}_f = \left(f\left(\mathbf{x}_l,\boldsymbol{\theta}\right)\right)_{l=1,2,...,n_T}$$

where $\boldsymbol{\theta}$ is the set of parameters of f. The application assigns to each function of the family the vector obtained with the values of the function over the training data set. Thus we can define the distance between two functions of the family as the Euclidean distance between the associated vectors:

$$d(f,g) = \left\|\hat{y}_f - \hat{y}_g\right\| = \left[\sum_{l=1}^{n_T} \left|f(\mathbf{x}_l,\boldsymbol{\theta}) - g(\mathbf{x}_l,\boldsymbol{\theta})\right|^2\right]^{1/2}$$

With this distance measure, the proximity between two functions is related to their performance. So, similar functions using this distance will have a similar performance over the same regression problem. Now, considering a set of functions $\{f_1, f_2, ..., f_M\}$ of the family F, we can build the set of associated vectors $\{\hat{y}_{f_1}, \hat{y}_{f_2}, ..., \hat{y}_{f_M}\}$ of $\mathbb{R}^n$. The minimum sum-of-squares clustering problem is to find a partition $P = \{C_1, C_2, ..., C_K\}$ of $\{\hat{y}_{f_1}, \hat{y}_{f_2}, ..., \hat{y}_{f_M}\}$ in K disjoint subsets (clusters) such that the sum of squared distances from each point to the centroid of its cluster is minimum. We use k-means clustering [10]. In this algorithm, the cluster centroid is defined as the mean data vector averaged over all items in the cluster. The number of clusters must be pre-assigned.

4.2 Hybrid Evolutionary Algorithm

As stated, we have two different algorithms depending on the stage when the clustering and local search algorithms are carried out:

1. **Algorithm HEPC.** We apply the clustering process to the best $\tilde{s}N_p$ individuals of the final population which is divided into K clusters $C_1, C_2, ..., C_K$. After that, we apply the L-M algorithm to the best individual of each cluster. The individuals obtained with the local-search algorithm are stored in a local optimum set C.
2. **Algorithm Dynamic HEPC.** We apply the clustering process and the L-M algorithm to the best individual of each cluster every G_0 generation and in the final population. The clustering process is applied on each selected generation to the best $\tilde{s}N_p$ individuals of the current population .The individuals obtained with the local-search algorithm are stored in C.

In cases 1 and 2, the final solution is the best individual among the local optimum of set C.

On the other hand, the parameters used are: the exponents w_{ji} are initialized in the interval $[-5,5]$, the coefficients β_j are initialized in $[-10,10]$. The size of the population is $N_p = 1000$. The maximum number of generations is 4000. The only parameter of the L-M algorithm is the tolerance of the error to stop the algorithm, in our experiment this parameter has the value 0.01. The k-means algorithm is applied to 25% of the best individuals of the population. The number of clusters K is 4 in the static version of HEPC. In the dynamic version the number of clusters varies from 6, at the beginning of the evolution, to a final value of 4 in the last generation.

5 Experiments

In order to test the performance of the proposed algorithms, they are applied to two benchmark regression problems: Friedman #1 and Sugeno functions.

5.1 Friedman#1 Function

This is a synthetic benchmark dataset proposed in [11]. The function is given by:

$$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon$$

where ε is a Gaussian random noise N(0;1) and the input variables are uniformly distributed in the interval (0;1]. 1000 samples are created randomly, 750 are randomly selected for the training set, and the remaining 250 are used as a test set. Table 1 shows the results for Friedman#1 function. The final solution has 6 nodes in the hidden layer. Table 1 shows a comparison of the results of the different algorithms proposed, where MSE_G is

$$\text{MSE}_\text{G} = \frac{1}{n_G}\sum_{i=1}^{n_G}[y_i - \hat{y}_i]^2$$

where the $\hat{y}_i$ are the predicted values. In order to test the soundness of the clustering approach, we carried out two additional experiments. First, the same methodology of HEPC was applied, removing the cluster analysis. Instead of performing the clustering, we applied the L-M algorithm to $K = 4$ randomly selected individuals from the best 25% of the population. The generalization results after 10 runs of the algorithm using both methods are shown in Table 2.

Table 1. Statistics of MSE_G for different models in the Friedman#1 problem. Results of NeuralBAG (NBAG), Simple and General Regression Neural Networks (GRNNFA) are adapted from [12] and [13]

Algorithms	Mean	SD	Runs
NBAG	4.502	0.268	20
Simple	4.948	0.589	20
GRNNFA	4.563	0.195	20
HEPC	1.105	0.102	30
Dynamic HEPC	1.081	0.040	30

Table 2. Statistics of MSE_G for Friedman#1 function for 10 runs of HEPC and randomly selecting the individuals to optimize

Algorithm	Mean	SD	Best	Worst	Mann-Whitney's U test
HEPC	1.095	0.114	1.066	1.607	
Random	1.408	0.665	1.068	2.981	p value=0.003

Table 2 shows that, on average, the results using the clustering are better and have less variance. We performed a Mann-Whitney's U test, as the results of HEPC did not follow a normal distribution, with a p-value of 0.003. We can conclude that the application of the clustering algorithm improves the results for the Friedman function.

The second experiment was carried out to confirm the performance of our approach. The HEPC method was compared to an evolutionary process, denoted by L-M all, where, instead of applying the optimization algorithm to the best individual of each cluster, we apply the optimization algorithm to every individual in the best 25% of the population. The results in performance and time are shown in Table 3.

Table 3. Results for Friedman#1 function for 10 runs of HEPC and selecting all the individuals to be optimized. The time is measured in average seconds per generation

	MSE_G				
Algorithm	Mean	SD	Best	Worst	t-test
HEPC	1.081	0.009	1.064	1.093	
L-M all	1.081	0.010	1.071	1.098	0.993
	Time				
Algorithm	Mean	SD	Best	Worst	t-test
HEPC	5.3	0.67	4	6	
L-M all	76.9	8.65	64	91	0.000

5.2 Sugeno Function

The second benchmark problem concerns an approximation of the Sugeno function defined as $f(x_1, x_2, x_3) = (1 + x_1^{0.5} + x_2^{-1} + x_3^{-1.5})^2$. For training, 216 points for [1,6] interval and 125 points for testing from [1.5,5.5] interval were randomly created. The Average Percentage Error (APE) was used as a measure of approximation error

$$APE = \frac{1}{N}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right| * 100\%$$

Final networks had at most 11 nodes in the hidden layer. Results using Dynamic HEPC are compared to other results obtained by Kosinski and Hirikawa [14] and Jankowski [15] (see Table 4).

Table 4. Results for the Sugeno function for 30 runs of dynamic HEPC. Results of Fuzzy Neural Networks (FNN3), MDelt, FuzzyVINET, Incremental Neural Network (IncNet) and Incremental Neural Network with Rotation (Incnet Rot) are adapted from [15]

	FNN3	MDelt	FuzzyIne	FuzzyVine	IncN	IncNetRo	Dyn.HEPC
APE_T	0.63	0.72	0.18	0.076	0.119	0.053	0.065
APE_G	1.25	0.74	0.24	0.18	0.122	0.061	0.060

6 Conclusions

In this work we have proposed a new method for using local optimization procedures in hybrid evolutionary algorithms. This approach is based on the application of a clustering algorithm for the selection of a small number of individuals subject to local optimization. The algorithm is applied to the optimization of the structure and weights of product-unit based neural networks. The results obtained in two benchmark problems of regression show that the hybrid algorithm provides a very good compromise between performance and computatioal cost.

References

[1] Houck, C. R., Joines, J. A., and Kay, M. G.: Empirical investigation of the benefits of partial lamarckianism, Evolutionary Computation, Vol. 5, no. 1, (1997), pp. 31-60.

[2] Moscató, P. and Cotta, C., "A gentle introduction to memetic algorithms": In: Glover, F. and Kochenberger, G., (eds.): Handbook of Metaheuristics, pp. 1-56. Kluwer, 1999.

[3] Bishop, C. M.: Neural Networks for Pattern Recognition. Oxford University Press, (1995).

[4] Durbin, R., Rumelhart D.: Product units: A computationally powerful and biologically plausible extension to backpropagation networks. Neural Computation, Vol. 1, (1989), pp. 133-142.

[5] Ismail, A., Engelbrecht, A. P.: Training product units in feedforward neural networks using particle swarm optimisation. In: Bajic, V. B., Sha, D. (eds.): Development and Practice of Artificial Intelligence Techniques, Proceeding of the International Conference on Artificial Intelligence, Durban, South Africa, (1999), pp. 36-40.

[6] Leerink, L. R., Giles, C. L., Horne, B. G, Jabri, M. A.: Learning with product units. Advances in Neural Information Processing Systems 7, MIT Press, (1995), pp. 537-544.

[7] Saito, K., Nakano, R.: Extracting regression rules from neural networks. Neural Networks, Vol. 15, no. 10, (2002). pp. 1279-1288

[8] Angeline P. J., Saunders, G. M., Pollack, J. B.: An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks, Vol. 5, no. 1, January (1994), pp. 54-65.

[9] Martínez, A., Martínez, F., Hervás, C., García, N.: Model Fitting by evolutionary computation using product units. Neural Networks, (2004) (Submitted).

[10] Fukunaga, K.: Introduction to Statistical Pattern Recognition. Academic Press, (1990).

[11] Friedman, J., Multivariate adaptive regression splines (with discussion). Ann. Stat., Vol. 19, (1991), pp. 1-41.

[12] Carney, J., Cunningham, P.: Tuning diversity in bagged ensembles. Int. J. Neural Systems, Vol. 10, no. 4, (2000), pp. 267-279.

[13] Lee, W. M., Lim, Ch. P., Yuen, K. K., Lo, S. M.: A hybrid neural network model for noisy data regression. IEEE Transactions on Systems, Man and Cybernetics, Part B, Vol. 34, no. 2, (2004), pp. 951-960.

[14] Kosinski, W., Weigl, M.: Mapping neural networks and fuzzy inference systems for approximation of multivariate function. In Kacki E., (eds.): System Modelling Control, Artificial Neural networks and Their Applications, Vol.3, Lodz, Poland, May (1995), pp. 60-65.

[15] Jankowski, N.: Approximation with RBF-type neural networks using flexible local and semi-local transfer functions. In 4th Conference on Neural Networks and Their Applications, Zakopane, Poland, May (1999), pp. 77-82.