Hybrid Evolutionary Algorithm with Product-Unit Neural Networks for Classification

Francisco J. Martínez- Estudillo[1], César Hervás-Martínez[2], Alfonso C. Martínez-Estudillo[1], and Pedro A. Gutiérrez-Peña[2]

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

Abstract. In this paper we propose a classification method based on a special class of feed-forward neural network, namely product-unit neural networks, and on a dynamic version of a hybrid evolutionary neural network algorithm. The method combines an evolutionary algorithm, a clustering process, and a local search procedure, where the clustering process and the local search are only applied at specific stages of the evolutionary process. Our results with the product-unit models and the evolutionary approach show a very interesting performance in terms of classification accuracy, yielding a state-of-the-art performance.

Keywords: Classification, Product-Unit Neural Networks, Evolutionary algorithms.

1 Introduction

We propose a classification method that combines a nonlinear model and a hybrid evolutionary neural network algorithm that finds the optimal structure of the model and estimates the corresponding parameters. The hybrid algorithm combines a clustering process and a local search procedure, where the clustering process and the local search are only applied at specific stages of the evolutionary process. The underlying idea is that we can achieve a very good performance if, instead of optimizing many very similar individuals in the final generation, we explore different regions of the search space visited by the algorithm throughout its evolution. The proposed non-linear model corresponds to a special class of feed-forward neural network, namely product-unit neural networks, PUNN, introduced by Durbin and Rumelhart [1]. They are an alternative to sigmoidal neural networks and are based on multiplicative nodes instead of additive ones.

The algorithm proposed evolves both the weights and the structure of the network using evolutionary programming. It is usually very difficult to know beforehand the most suitable structure of the network for a given problem; however, the evolution of the structure partially alleviates this problem. It is well known that evolutionary algorithms (EA) are efficient at exploring an entire search space; however, they are relatively poor at finding the precise optimum solution in the region in which the

F. Sandoval et al. (Eds.): IWANN 2007, LNCS 4507, pp. 351–358, 2007.

algorithm converges. The hybrid algorithm combines the EA (global explorer) and the local optimization procedure (local exploiter). The cluster process creates a group of mutually close points that could correspond to relevant regions of attractions, and finally, the local search procedure enables us to improve the performance of the selected individuals in the cluster process. The purpose of the dynamic version is to gather into one set the best solutions the evolutionary algorithm finds in the exploration of the search space at different stages. 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. Moreover, we do not use the crossover operator because this operation is usually regarded as being less effective for network evolution. We evaluate the performance of our methodology in four data sets taken from the UCI repository. This paper is organized as follows: Section 2 is dedicated to a description of the product-unit model: Section 3 describes the hybrid evolutionary algorithm; Section 4 includes the experimental results and, finally, Section 5 summarizes the conclusions of our work.

2 Product-Unit Neural Networks Classifiers

In this section we present the family of product-unit basis functions used in the classification process and its representation by means of a neural network structure. PUNN are built with basis functions (1) that express the possible strong interactions between the variables, where the exponents may even take on real values and are suitable for automatic adjustment:

$$y_j = \prod_{i=1}^{k} x_i^{w_{ji}} \tag{1}$$

k being the number of inputs. Some advantages of product-unit based neural networks are its increased information capacity and the ability to form higher-order input combinations. Besides that, it is possible to obtain the upper bounds of the VC dimension in product-unit neural networks similar to those obtained in sigmoidal neural networks [2]. Finally, it is a straightforward consequence of the Stone-Weierstrass Theorem to prove that product-unit neural networks are universal approximators [3]. Despite these advantages, product-unit based networks have a major drawback. Networks based on product units have more local minima and more probability of getting trapped in them [4]. The main reason for this difficulty is that small changes in the exponents can cause large changes in the total error surface. Because of this, their training is more difficult than the training of standard sigmoidal based networks For example, it is well known [5] that back-propagation is not efficient in training product units. So far, the studies carried out on PUNNs have not tackled the problem of the simultaneous design of the structure and weights in this kind of neural network, using either classic or evolutionary based methods. Moreover, product units have been applied mainly to solve regression problems [3],[6],[7].

We consider a product-unit neural network with the following structure: an input layer with k nodes, a node for every input variable, a hidden layer with m nodes,

and an output layer with J nodes, one for each class level. There are no connections between the nodes of a layer, and none between the input and output layers either. The activation function of the j-th node in 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 and $\mathbf{w}_j = (w_{j1}, ..., w_{jk})$ the weights vector. The activation function of output node l is given by $\beta_0^l + \sum_{j=1}^{m} \beta_j^l B(\mathbf{x}, \mathbf{w_j})$, where β_j^l is the weight of the connection between hidden node j and output node l and β_0^l the corresponding bias. The transfer function of all hidden and output nodes is the identity function. In this way, the estimated function $f_l(\mathbf{x}; \boldsymbol{\theta}_l)$ from each output is given by:

$$f_l(\mathbf{x}; \boldsymbol{\theta}_l) = \beta_0^l + \sum_{j=1}^{m} \beta_j^l B_j(\mathbf{x}, \mathbf{w}_j), \quad l = 1, 2, ..., J \tag{2}$$

where $\boldsymbol{\theta}_l = (\boldsymbol{\beta}^l, \mathbf{w}_1, ..., \mathbf{w}_m)$ and $\boldsymbol{\beta}^l = (\beta_0^l, \beta_1^l, ..., \beta_m^l)$.

We consider the softmax activation function given by:

$$g_l(\mathbf{x}, \boldsymbol{\theta}_l) = \frac{\exp f_l(\mathbf{x}, \boldsymbol{\theta}_l)}{\sum_{l=1}^{J} \exp f_l(\mathbf{x}, \boldsymbol{\theta}_l)}, \; l = 1, 2, ..., J \tag{3}$$

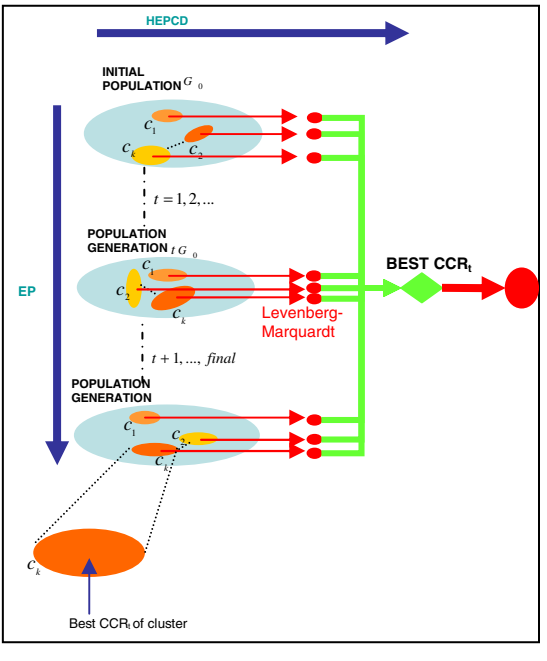

Fig. 1. HEPCD algorithm framework

Let $D = \{(\mathbf{x}_n, \mathbf{y}_n) : n = 1, 2, ..., n_T\}$ be the training data set, where $x_{in} > 0$, $\forall i, n$ and $\mathbf{y}_n$ is the class level of the n-th individual. We adopt the common technique of representing class levels using a "1-of-J" encoding vector $\mathbf{y} = \left(y^{(1)}, y^{(2)} ..., y^{(J)}\right)$, such as $y^{(l)} = 1$ if $\mathbf{x}$ corresponds to an example belonging to class l and, otherwise $y^{(l)} = 0$. The cross-entropy error function for those observations is:

$$l(\boldsymbol{\theta}) = -\frac{1}{n_T} \sum_{n=1}^{n_T} \sum_{l=1}^{J} y_n^{(l)} \log g_l\left(\mathbf{x}_n, \boldsymbol{\theta}_l\right) \tag{4}$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_J)$. The optimum rule $C(\mathbf{x})$ is the following:

$$C(\mathbf{x}) = \hat{l}, \text{ where } \hat{l} = \arg\max_l g_l(\mathbf{x}, \hat{\boldsymbol{\theta}}), \text{ for } l = 1, 2, ..., J \tag{5}$$

Finally, we define the corrected classified rate by $CCR = (1/n_T) \sum_{n=1}^{n_T} I(C(\mathbf{x}_n) = \mathbf{y}_n)$, where $I(\bullet)$ is the zero-one loss function.

3 The Hybrid Evolutionary Neural Network Algorithm

The algorithm called dynamic hybrid evolutionary programming with clustering HEPCD carries out a clustering process and a local search procedure throughout the evolutionary process. Concretely, we apply the clustering process and the local search to the best individual of each cluster in different stages of the evolution and in the final population. The clustering process is applied only to a percentage of the best individuals of the current population. The local search is applied to the best individual of each cluster and the fitted individuals are stored in a separate population B. The final solution is the best individual among the local optima found during the evolutionary process. The local optimization algorithm used in our work is the Levenberg-Marquardt (L-M) optimization method. In any case, any other local optimization algorithm can be used in a particular problem.

The general framework of the *Dynamic Hybrid Evolutionary Programming with Clustering* (HEPCD) is the following (see Figure 1):

1. Generate a random population of size N_P.
2. Repeat until the stopping criterion is fulfilled
   - 2.a) Apply parametric mutation to the best 10% of individuals. Apply structural mutation to the remaining 90% of individuals.
   - 2.b) Calculate the fitness of every individual in the population.
   - 2.c) Add best *fitness* individual of the last generation (*elitist algorithm*).
   - 2.d) Rank the individuals with respect to their *fitness*.
   - 2.e) Best 10% of population individuals are replicated and substitute the worst 10% of individuals.

Apply the following process every G_0 generations:

- 2.f) Apply k-means process to best $s\%$ individuals of the population in the current generation, assigning a cluster to each individual.

2.g) Select the best CCR solution in each cluster and apply the L-M algorithm to each selected individual.
2.h) Select the best CCR individual among optimized ones and add it to the B set.

3. Select the best *CCR* individual in set B and return it as the final solution, using *CCR* as the selection criterion.

Next, we describe parametric and structural mutations and the clustering process in detail.

3.1 Structural and Parametric Mutations

The fitness measure is a strictly decreasing transformation of the entropy error $l(\boldsymbol{\theta})$ given by $A(g) = \dfrac{1}{1+l(\boldsymbol{\theta})}$, where g is a product-unit neural network given by the the multivaluated function $g(\mathbf{x},\boldsymbol{\theta}) = (g_1(\mathbf{x},\boldsymbol{\theta}_1),...,g_l(\mathbf{x},\boldsymbol{\theta}_l))$. Parametric mutation is accomplished for each coefficient w_{ji}, β_j^l of the model with Gaussian noise:

$$w_{ji}(t+1) = w_{ji}(t) + \xi_1(t), \qquad \beta_j^l(t+1) = \beta_j^l(t) + \xi_2(t) \tag{6}$$

where $\xi_k(t) \in N(0, \alpha_k(t))$, for each $k = 1,2$, represents a one-dimensional normally-distributed random variable with mean 0 and variance $\alpha_k(t)$. Once the mutation is performed, the fitness of the individual is recalculated and the usual simulated annealing process is applied. Thus, if ΔA is the difference in the fitness function after and preceding the random step, the criterion is: if $\Delta A \geq 0$, the step is accepted, and if $\Delta A < 0$, the step is accepted with a probability $\exp(\Delta A / T(g))$, where the temperature $T(g)$ of an individual g is given by $T(g) = 1 - A(g)$, $0 \leq T(g) < 1$. The variances $\alpha_k(t)$ are updated throughout the evolution of the algorithm. There are different methods to update the variance. We use the 1/5 success rule of Rechenberg [8], one of the simplest methods.

Structural mutation implies a modification in the neural network structure and allows explorations of different regions in the search space while helping to keep up the diversity of the population. There are five different structural mutations: node deletion, connection deletion, node addition, connection addition and node fusion. The first four are similar to the mutation in the GNRL model [9]. In the node fusion, two randomly selected hidden nodes, a and b, are replaced by a new node c, which is a combination of the two. The connections that are common to both nodes are kept, with a weight given by:

$$\beta_c^l = \beta_a^l + \beta_b^l, \qquad w_{jc} = (1/2)\left(w_{ja} + w_{jb}\right) \tag{7}$$

The connections that are not shared by the nodes are inherited by c with a probability of 0.5 and their weight is unchanged. The stop criterion is reached if one of the following conditions is fulfilled: a number of generations is reached or the variance of the fitness of the best ten percent of the population is less than 10^{-4}.

3.2 Clustering Partitioning Technique

Let $D = \{(\mathbf{x}_n, \mathbf{y}_n)\}$ be the training data set. We assign to each classifier g the binary vector $\hat{\mathbf{y}}_g$ of n_T dimension, where the i coordinate is 1 if the $\mathbf{x}_i$ pattern is correctly classified and otherwise 0. Thus we can define the distance between two neural networks classifiers g and h as the Euclidean distance between the associated vectors $d(g,h) = \|\hat{y}_g - \hat{y}_h\|$. With this distance measurement, the proximity between two classifiers is related to their performance and the diversity of the classification task. So, similar functions using this distance will have a similar performance for the same classification problem. We use K-means clustering. The centroid of each is defined as the mean data vector averaged over all items in the cluster and does not correspond to any concrete model of the population. We use the centroid only as a tool of the algorithm. The choice of the K-means has been made mainly because it is simple, fast and easy to implement. The number of clusters must be pre-assigned.

4 Experimental Results

We evaluate the performance of our methodology on four data sets with different features taken from the UCI repository [10]: Breast-w, Breast-Cancer (Cancer), Balance-scale and Australian card. The experimental design was conducted using a 10-fold stratified cross-validation procedure and 10 runs per each fold. The parameters used in all experiments were: the exponents w_{ji} were initialized in the interval $[-5,5]$, the coefficients β_j were initialized in $[-10,10]$, the size of the population was $N_p = 1000$ and $\alpha_1(0) = 0.01$, $\alpha_2(0) = 0.1$. The maximum number of generations was 200. The only parameter of the L-M algorithm is the tolerance of the error to stop the algorithm, in our experiment this parameter had the value 0.01. The K-means algorithm was applied to $s = 25\%$ of the best individuals of the population. The number of K clusters was 4 and the maximum number of hidden nodes was 6. The clustering process and the local search were carried out in the 100, 150 and 200th generation ($G_0 = 50$). Table 1 shows the statistical results of the HEPCD algorithm. Moreover, we compare our approach to recent results [11] obtained using eleven classification techniques: Logistic model tree algorithm, LMT, two logistic regression (with attribute selection, SLogistic, and for a full logistic model, MLogistic); induction trees (C4.5 and CART [12]); a naïve Bayes tree learning algorithm NBTree [13]; two functional tree learning algorithms LTreeLin and LTreeLog [14] and finally, multiple-tree models M5´ for classification [15], and boosted C4.5 trees using AdaBoost.M1 with 10 and 100 boosting interactions. Under the hypothesis of the normality of the results, we carried out a t-student test (5% level significance) comparing our HEPCD approach to the best algorithm (in bold face) for each dataset. The asterisk in Table 2 shows that there are significant differences, in the mean of the CCR_G, between HEPCD and LTreeLin for the Balance dataset. There are not significant differences between HEPCD and the best algorithm for the rest of the datasets.

Table 1. Statistical results of training and testing for 100 executions of the HEPCD algorithm

	CCR Training				*CCR* Generalization				# conn	
Datasets	Mean	SD	Best	Worst	Mean	SD	Best	Worst	Mean	SD
Breast-w	76.44	1.16	79.07	74.03	73.50	6.83	85.71	57.14	10.81	2.14
Cancer	97.67	0.26	98.25	0.26	96.71	1.94	100.00	1.94	10.67	1.41
Balance	97.39	0.95	100.00	96.09	96.10	2.69	100.00	88.70	19.21	5.45
Australian	87.77	0.81	90.50	85.99	85.46	3.99	95.65	72.46	33.96	12.9

Table 2. Mean classification accuracy and standard deviation of CCR_G for LMT, SLogistic, MLogistic, C4.5, CART, NBTree, two tree functional learning algorithms (LTReeLin and LTreeLog), M5' for classification and ABoost(10) and ABoost(100). The results were taken from [11].

Datasets	LMT	SLogistic	MLogistic	C4.5	CART	NBTree
Breast-w	96.18±2.20	96.21±2.19	96.50±2.18	95.01±2.73	94.42±2.70	96.60±2.04
Cancer	74.91±6.29	**74.94±6.25**	67.77±6.92	74.28±6.05	69.40±5.25	70.99±7.94
Balance	89.71±2.68	88.74±2.91	89.44±3.29	77.82±3.42	78.09±3.97	75.83±5.32
Australian	85.04±3.84	85.04±3.97	85.33±3.85	85.57±3.96	84.55±4.20	85.07±4.03
Datasets	LTreeLin	LTreeLog	M5'	ABoost(10)	ABoost(100)	HEPCD
Breast-w	96.68±1.99	**96.75±2.04**	95.85±2.15	96.08±2.16	96.70±2.18	**96.71±1.94**
Cancer	70.58±6.90	70.45±6.78	70.40±6.84	66.75±7.61	66.36±8.18	**73.50±6.83**
Balance	**92.86±3.22**	92.78±3.49	87.76±2.23	78.35±3.78	76.11±4.09	**96.10±2.69***
Australian	84.99±3.91	84.64±4.09	85.39±3.87	84.01±4.36	**86.43±3.98**	**85.46±3.99**

5 Conclusions

We have proposed a new approach to solve classification problems based on the combination of an evolutionary neural network algorithm; a clustering process and a local-search procedure, where the clustering partitioning and the local searches are carried out in different stages of the evolutionary process. The algorithm evolves the non-linear model given by product-unit neural networks. The experiments carried out suggest that a product-unit neural network is an efficient nonlinear model to solve classification problems. Finally, the reader can observe that the basic framework of the algorithm can be applied to different neural network models and could be tuned by using other clustering and local search methods.

Acknowledgments. This work has been partially supported by TIN2005-08386-C05-02 projects of the Spanish Inter-Ministerial Commission of Science and Technology (MICYT) and FEDER funds.

References

1. Durbin, R., Rumelhart, D.: Products Units: A computationally powerful and biologically plausible extension to backpropagation networks. Neural Computation 1, 133–142 (1989)
2. Schmitt, M.: On the Complexity of Computing and Learning with Multiplicative Neural Networks. Neural Computation 14, 241–301 (2001)

3. Martinez-Estudillo, A., et al.: Evolutionary product unit based neural networks for regression. Neural Networks 19(4), 477–486 (2006)
4. Ismail, A., Engelbrecht, A. P.: Global optimization algorithms for training product units neural networks. In: International Joint Conference on Neural Networks IJCNN‘2000, Como, Italy (2000)
5. Janson, D.J., Frenzel, J.F.: Training product unit neural networks with genetic algorithms. IEEE Expert 8(5), 26–33 (1993)
6. Engelbrecht, A.P., Ismail, A.: Training product unit neural networks. Stability and Control: Theory and Applications 2(1-2), 59–74 (1999)
7. Saito, K., Nakano, R.: Extracting Regression Rules From Neural Networks. Neural Networks 15, 1279–1288 (2002)
8. Rechenberg, I.: Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Evolution, Stuttgart Framman-Holzboog Verlag (1975)
9. Angeline, P.J., Saunders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks 5(1), 54–65 (1994)
10. Blake, C., Merz, C. J.: UCI repository of machine learning data bases (1998) www.ics.uci.edu/ mlearn/MLRepository.thml
11. Landwehr, N., Hall, M., Eibe, F.: Logistic Model Trees. Machine Learning 59, 161–205 (2005)
12. Breiman, L., et al.: Classification and Regression Trees, Belmont, CA Wadsworth (1984)
13. Kohavi, R.: Scaling up the accuracy of naive bayes classifiers: A decision-tree hybrid. In: Proc. 2nd International Conference on Knowledge Discovery and Data Mining Menlo Park, AAAI Press, CA (1996)
14. Gama, J.: Functional trees. Machine Learning 55(3), 219–250 (2004)
15. Wang, Y., Witten, I.: Inducing model trees for continuous classes. In: Proceedings of Poster Papers, European Conference on Machine Learning. Prague, Czech Republic. (1997)