2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Classification by means of Evolutionary Product-Unit Neural Networks

César Hervás, Francisco J. Martínez, and Pedro A. Gutiérrez

Abstract— **We propose a classification method based on a special class of feed-forward neural network, namely product-unit neural networks. They are based on multiplicative nodes instead of additive ones, where the nonlinear basis functions express the possible strong interactions among the variables. We apply an evolutionary algorithm to determine the basic structure of the product-unit model and to estimate the coefficients of the model. The empirical results show that the proposed model is very promising in terms of classification accuracy, yielding a state-of-the-art performance.**

I. Introduction

The simplest method for classification provides the class level given its observation via linear functions in the predictor variables. This process of model fitting is quite stable, resulting in low variance but a potentially high bias. Frequently, in a real-problem of classification, we cannot make the stringent assumption of additive and purely linear effects of the variables. A traditional technique to overcome these difficulties is to augment/replace the input vector with new variables, basis functions, which are transformations of the input variables, and then to use linear models in this new space of derived input features. One approach would be to augment the inputs with polynomial terms to achieve higher-order Taylor expansions, for example, with quadratic terms and multiplicative interactions. Once the number and the structure of the basis functions have been determined, the models are linear in these new variables and the fitting is a standard procedure. Methods like sigmoidal feed-forward neural networks [1], projection pursuit learning [2], generalized additive models [3] and PolyMARS [4], a hybrid of multivariate adaptive splines (MARS) [5] specifically designed to handle classification problems, can be seen as different basis functions models. The major drawback of these approaches is to state the number and the typology of the corresponding basis functions.

We tackle this problem by proposing a nonlinear model and an evolutionary algorithm that finds the optimal structure of the model and estimates the corresponding parameters. Concretely, our approach tries to overcome the nonlinear effects of the variables proposing a model based on nonlinear basis functions constructed with the product of the inputs raised to arbitrary powers. These basis functions express the possible strong interactions between the variables, where the exponents are not fixed and may even take real values. Moreover, we avoid the huge number of coefficients involved in the polynomial model. The proposed model corresponds to a special class of feed-forward neural network, namely product-unit neural networks, PUNN, introduced by Durbin and Rumelhart [6]. They are an alternative to standard sigmoidal neural networks (when a sufficient number of highly correlated input variables exist) and are based on multiplicative nodes instead of additive ones. Unfortunately, the error surface associated with product-unit neural networks is extremely convoluted with numerous local optima and plateaus.

On the other hand, classical training algorithms assume a fixed architecture; nevertheless it is very difficult to know a priori the most suitable structure of the network for a given problem. There have been many attempts to design the architecture automatically, such as constructive and pruning algorithms [7], [8]. However, these methods are susceptible to becoming trapped at structural local optima. Evolutionary artificial neural networks (EANNs) have been a key research area in the past decade providing a better platform for optimizing both network performance and architecture simultaneously. Miller *et al*. [9] proposed that evolutionary computation was a very good candidate to be used to search the space of architectures because the fitness function associated with that space is complex, noisy, non-differentiable, multi-modal and deceptive. Since then, many evolutionary programming methods have been developed for evolving artificial neural networks, see for example [10-15]. Stanley and Miikkulainen, [16], demonstrate that evolving

This work has been financed in part by TIN 2005-08386-C05-02 projects of the Spanish Inter-Ministerial Commission of Science and Technology (MICYT) and FEDER funds.

F. J. Martínez is with Department of Management and Quantitative Methods, ETEA, Escritor Castilla Aguayo 4, 14005, Córdoba, Spain, (corresponding author, phone +34957222120; fax +34957222107; email:fjmestud@etea.com).
C. Hervás is with Department of Computing and Numerical Analysis of the University of Córdoba, Campus de Rabanales, 14071, Córdoba, Spain (email: chervas@uco.es).
P. A. Gutiérrez is with Department of Computing and Numerical Analysis of the University of Córdoba, Campus de Rabanales, 14071, Córdoba, Spain (email:zamarck@yahoo.es).

structure along with connection weights can significantly enhance the performance of the neural network. Therefore, evolutionary algorithms are better candidates to design a near optimal architecture than the constructive and pruning algorithms mentioned before. This fact, together with the complexity of the error surface associated with a product-unit neural network, justifies the use of an evolutionary algorithm (EA) to design the structure and training of the weights. The evolutionary process determines the number of basis functions of the model, the associated coefficients and the corresponding exponents.

In our approach, we encourage parsimony of evolved networks by attempting different mutations sequentially, where deletion and fusion mutations are made with higher probability than addition ones. Similar to the EPNet model [11], our experimental results show that evolving parsimonious networks by sequentially applying different mutations is an alternative to the use of a regularization term in the fitness function to penalize large networks.

We evaluate the performance of our methodology on four data sets taken from the UCI repository [17]. The empirical results show that the proposed method performs well compared to several learning classification techniques. We obtain a classifier with very promising results in terms of classification accuracy and the complexity of the classifier.

This paper is organized as follows: Section II is dedicated to a description of product-unit based neural networks; Section III, describes the evolution of product-unit neural networks; Section IV explains the experiments carried out; and finally, Section V summarizes the conclusions of our work.

II. Product- Unit Neural Networks

In this section we present the family of functions used in the classification process and its representation by means of a neural network structure. An alternative to the standard sigmoidal neural networks are the networks based on multiplicative nodes instead of additive ones. This class of multiplicative neural networks comprise such types as sigma-pi networks and product unit networks. A multiplicative node is given by:

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

where k is the number of the inputs. If the exponents are $\{0,1\}$ we obtain a higher-order unit, also known by the name of sigma-pi unit. In contrast to the sigma-pi unit, in the product-unit the exponents are not fixed and may even take real values. Advantages of product-unit based neural networks (PUNNs) are increased information capacity and the ability to form higher-order combinations of the inputs. Durbin and Rumelhart [6] determined empirically that the information capacity of product units (measured by their capacity for learning random Boolean patterns) is approximately $3N$, compared to $2N$ of a network with additive units for a single threshold logic function, where N denotes the number of inputs to the network. On the other hand, it is possible to obtain upper bounds of the VC dimension of product-unit neural networks similar to those obtained for sigmoidal neural networks [18]. Finally, it is a straightforward consequence of the Stone-Weierstrass Theorem to prove that product-unit neural networks are universal approximators, (observe that the set of polynomial functions in several variables is a subset of the product-unit models).

Despite these advantages, product-unit based neural networks have a major drawback. Networks based on product units have more local minima and more probability of becoming trapped in them [19], [20]. 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. It is a well known problem [21] that back-propagation is not efficient in training product units.

Several efforts have been made to carry out learning methods for product units. Janson and Frenzel [21] developed a genetic algorithm for evolving the weights of a network based on product units with a predefined architecture. The major problem of this kind of algorithm is how to obtain the optimal architecture beforehand. Ismail and Engelbrecht [19], [20] applied four different optimization methods to train product unit neural networks: random search, particle swarm optimization, genetic algorithms, and leapfrog optimization. They concluded that random search is not efficient in training this type of network, and that the other three methods show an acceptable performance in three problems of function approximation with low dimensionality. In a posterior paper [22] they used a pruning algorithm to develop the structure as well as the training of the weights of a product-unit based neural network. Leerink *et al.* [23] tested different local and global optimization methods for product-unit networks. Their results show that local methods, such as backpropagation, are prone to be trapped in local minima, and that global optimization methods, such as simulated annealing and random search, are impractical for larger networks. They suggested some heuristics to improve backpropagation, and the combination of local and global search methods. In short, the works carried out on PUNNs have not tackled the problem of the design of both the structure and weights, either using classic or evolutionary based methods. Moreover, in the above mentioned papers, product-unit based neural networks have been applied mainly to solve regression problems.

On the other hand, it is interesting to note that a problem arises with networks containing product units that receive negative inputs and have weights that are not integers. A negative number raised to some non-integer power yields a complex number. Since neural networks with complex outputs are rarely used in applications, Durbin and Rumelhart [6] suggest discarding the imaginary part and using only the real component for further processing. This manipulation would have disastrous consequences for the VC dimension when we consider real–valued inputs. No finite dimension bounds can in general be derived for networks containing such units [18]. To avoid this problem, the input domain is restricted, and we consider the set given by $\left\{(x_1, x_2, \ldots, x_k) \in \mathbb{R}^k : x_i > 0, i = 1, 2, \ldots, k\right\}$.

We consider a product-unit neural network with the following structure (Fig. 1): an input layer with a node for every input variable, a hidden layer with several nodes, and an output layer with c nodes, one for each category. 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. The activation function of each output node 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 the hidden node j and the output node l. The transfer function of all output nodes is the sigmoidal function $\sigma(t) = \frac{1}{1+e^{-t}}$. In this way, the signal from each output is a function $g_l(\mathbf{x})$ given by:

$$g_l(\mathbf{x}) = \sigma\left(\beta_0^l + \sum_{j=1}^{m} \beta_j^l \left(\prod_{i=1}^{k} x_i^{w_{ji}}\right)\right)$$

Our predictor $G(\mathbf{x})$ takes values in a discrete set Λ with c classes. We have c such indicators, Y_l, $l = 1, 2, \ldots, c$, with $Y_l = 1$ if $G = l$, or else $Y_l = 0$. The rule of classification considered is: Given a new observation with an input $\mathbf{x}$, we compute the fitted output vector $\hat{g}_l(\mathbf{x})$, identify the largest component and classify accordingly $\hat{G}(\mathbf{x}) = \arg\max_{l \in \wedge} \hat{g}_l(\mathbf{x})$.

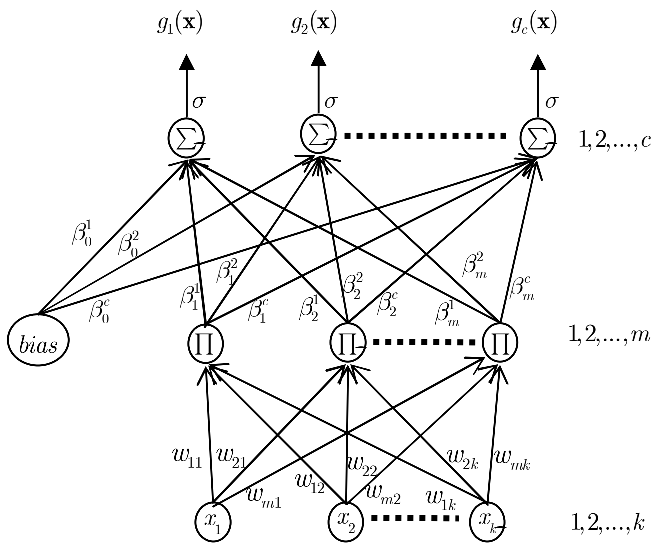

Fig. 1. Model of a product-unit based neural network.

III. EVOLUTIONARY ALGORITHM

We use an evolutionary algorithm to design the structure and learn the weights of product-unit neural networks. The search begins with an initial population, and, in each iteration, the population is updated using a population-update algorithm. The population is subjected to the operations of replication and mutation. Crossover is not used due to its potential disadvantages in evolving artificial networks [10], [11]. With these features the algorithm falls into the class of evolutionary programming [24], [25]. The general structure of the EA is the following:

(1) Generate a random population of size N.
(2) Repeat until the stopping criterion is fulfilled
  (a) Calculate the fitness of every individual in the population.
  (b) Rank the individuals with respect to their fitness.
  (c) The best individual is copied into the new population.
  (d) The best 10% of population individuals are replicated and substitute the worst 10% of individuals.
  Over that intermediate population we:
  (e) Apply parametric mutation to the best 10% of individuals.
  (f) Apply structural mutation to the remaining 90% of individuals.

We consider the Percentage of Correctly Classified Examples (PCCE) in the training data set as fitness measure $A(g)$ of an individual g of the population.

Parametric mutation consists of a simulated annealing algorithm [26], [27]. The severity of a mutation to an individual g is dictated by the temperature $T(g)$, given by

$$T(g) = 1 - A(g), \quad 0 \leq T(g) < 1$$

Thus, the temperature is determined by closeness of the function to any solution of the problem. Parametric mutation is accomplished for each coefficient w_{ji}, β_j^l of the model with Gaussian noise and where the variance depends on the temperature:

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

where $\xi_k(t) \in N(0, \alpha_k(t)T(g))$, $k = 1,2$, represents a one-dimensional normally distributed random variable with mean 0 and variance $\alpha_k(t)T(g)$. Once the mutation is performed, the fitness of the individual is recalculated and the usual simulated annealing is applied. Thus, if ΔA is the difference in the fitness function before and after the random step, the criterion is: if $\Delta A \geq 0$ the step is accepted, if $\Delta A < 0$, the step is accepted with a probability $\exp(\Delta A / T(g))$.

The parameters $\alpha_k(t)$ allow the adaptation of the learning process throughout the evolution:

$$\alpha_k(t+1) = \begin{cases} (1+\lambda)\alpha_k(t) & if \;\; A(g_s) > A(g_{s-1}), \; \forall s \in \{t, t-1, ..., t-\rho\} \\ (1-\lambda)\alpha_k(t), & if \;\; A(g_s) = A(g_{s-1}), \; \forall s \in \{t, t-1, ..., t-\rho\} \\ \alpha_k(t) & otherwise \end{cases}$$

where $k = 1,2$, $A(g_s)$ is the fitness of the best individual, g_s, in the generation s, λ and ρ must be set by the user.

It should be pointed out that the modification of the exponents w_{ji} is different from the modification of the coefficients β_j^l, therefore $\alpha_1(t) \ll \alpha_2(t)$. The adaptation tries to avoid being trapped in local minima and to speed up the evolutionary process when the conditions of the searching are suitable. A generation is defined as successful if the best individual of the population is better than the best individual of the previous generation, that is: $A(g_s) > A(g_{s-1})$. If many successes are observed, this indicates that the best solutions are residing in a better region of the search space. In this case, we increase the strength hoping to find even better solutions closer to the optimum solution. If the fitness of the best individual is constant during several generations, $A(g_s) = A(g_{s-1})$, we decrease the mutation rate. Otherwise, the mutation strength is constant.

Structural mutation implies a modification of the neural network structure and allows the explorations of different regions in the search space while helping to keep the diversity of the population. There are five different structural mutations: node deletion, connection deletion, node addition, connection addition and node fusion. These five mutations are applied sequentially to each network. The first four are similar to the mutations in the GNARL model [10]. In the node fusion, two randomly selected nodes, a and b, are replaced by a new node c, which is a combination of both. The connections that are common to both nodes are kept, with a weight given by:

$$\beta_c^l = \beta_a^l + \beta_b^l$$
$$w_{ic} = \frac{w_{ia} + w_{ib}}{2}$$

The connections that are not shared by the nodes are inherited by c with probability 0.5 and its weight is unchanged. For each mutation (excepting node fusion) there is a minimum value, Δ_{Min}, and a maximum value, Δ_{Max}, and the number of elements (nodes or connections) involved in the mutation is calculated as

$$\Delta_{Min} + \lfloor u \, T(g) \, (\Delta_{Max} - \Delta_{Min}) \rfloor$$

where u is a random uniform variable in the interval $[0,1]$.

In our algorithm we encourage parsimony in evolved networks by attempting different mutations sequentially, where node or connection deletion and node fusion is always attempted before addition. Moreover, the deletion and fusion operations are made with higher probability ($T(g)$ for deletion and fusion mutations and $T^2(g)$ for addition ones). If a deletion or fusion mutation is successful, no other mutation will be made. If the probability does not select any mutation, one of the mutations is chosen at random and applied to the network.

IV. Experiments

We evaluate the performance of our methodology on four data sets taken from the UCI repository [17]. The experimental design for the four classification benchmark problems was conducted using a holdout cross-validation procedure. In the following experiments each data set was partitioned as follows:

- For the balance data set, the first 469 examples were used for the training set and the following 156 for the testing set.
- For the cancer data set, the first 525 examples were used for the training set and the following 174 for the testing set.
- For the pima data set, the first 576 examples were used for the training set and the following 192 for the testing set.
- For the glass data set, the first 161 examples were used for the training set and the following 53 for the testing set.

The parameters used in the evolutionary algorithm are common for the four problems. We have considered $\alpha_1(0)=0.5$, $\alpha_2(0)=1$, $\lambda=0.1$ and $\rho=5$. The exponents w_{ji} are initialized in the $[-5,5]$ interval, the coefficients β_j^l are initialized in $[-5,5]$. The maximum number of hidden nodes is $m=6$. The size of the population is $N=2000$. The number of nodes that can be added or removed in a structural mutation is within the $[1,2]$ interval. The number of connections that can be added or removed in a structural mutation is within the $[1,6]$ interval.

The stop criterion is reached whenever one of the following two conditions is fulfilled: i) for 20 generations there is no improvement either in the average performance of the best 20% of the population or in the fitness of the best individual, ii) The algorithm achieves a determined number of generations.

We have done a simple linear rescaling of the input variables in the interval $[1,2]$, being X_i^* the transformed variables. The lower bound is chosen to avoid input values near 0 that can produce very large values of the outputs for negative exponents. The upper bound is chosen to avoid dramatic changes in the outputs of the network when there are weights with large values (especially in the exponents).

In order to determine the meta-parameter of our algorithm given by the maximum number of generations and to establish the most suitable values for such meta-parameter, (in the sense of their influence on the percentage of correctly classified examples in the testing data set $PCCE_T$), the ANalysis Of the VAriance (ANOVA) statistical method was used. This statistical tool is based on the analysis of the mean variance. The theory of ANOVA was mainly developed by Fisher [28] during the 1920s. ANOVA examines the effects of some quantitative or qualitative variables (called factors) on one quantitative response. The best objective for that analysis is to try to determine if the influence of a change in a meta-parameter value is significant in mean on the $PCCE_T$ obtained in our algorithm. In our case the linear model has the form:

$$PCCE_{Tijk} = \mu + D_i + G_j + DG_{ij} + e_{ijk}$$

for $i=1,2,3,4$; $j=1,2,3$ and $k=1,2,...,30$. The first factor D_i analyzed the effect over the $PCCE_T$ of the i-th level of that factor, where D_i represents the data set used in our experimentation, with levels: $(i=1)$ for balance, $(i=2)$ for Cancer, $(i=3)$ for Pima and $(i=4)$ for Glass. The second factor G_j is the effect associated with the j-th level of this factor, where G_j is the maximum number of generations, with levels: 50 $(j=1)$, 100 $(j=2)$ and 200 $(j=3)$. The term μ is the fixed effect that is common to all the populations; the term DG_{ij}, named the interaction term, denotes the joint effect of the presence of the level i of the first factor and the level j of the second one. The term e_{ijk} is the influence on the result of everything what could not be assigned, or of random factors.

Thus, 360 simulations were carried out, corresponding to all the possible combinations of application of the four levels for the first factor and the three levels of the second factor. The results of the ANOVA analysis show that:

1) The data set factor effect is statistically significant at the level of confidence of 95%.

2) The number of generation's effect is not statistically significant at the level of confidence of 95%.

3) There is no interaction between the number of generations and the data sets.

Table I shows the statistical results over 30 runs of the evolutionary algorithm for the four benchmark problems with different maximum number of generations: 50, 100 and 200.

With the objective of presenting an empirical evaluation of the performance of the evolutionary PUNN model, we compare our approach to the most recent results [29] obtained using eleven different methodologies (see Table II): logistic model tree algorithm, LMT, two logistic regression (with attribute selection, SLogistic, and for a full logistic model, MLogistic); induction trees (C4.5 [30] and CART [31]); two logistic tree algorithms: LTreeLog [32] and Lotus [33] with two methodologies: one using simple logistic regression (LotusS) and another using multiple logistic regression (LotusM), both for two class data set; and finally, multiple-tree models M5´ [34] for classification, and boosted C4.5 trees using AdaBoost.M1 with 10 and 100 boosting interactions. The results in Table II have been taken from [29]. We can see that the results obtained by PUNN are competitive with the learning schemes mentioned previously.

Finally, Table III shows the best models for each data set. The models can be easily implemented and the reader can reproduce and compare the results.

V. CONCLUSIONS

We propose a classification method based on a special class of feed-forward neural network, namely product-unit neural networks, where the corresponding nonlinear basis functions express the possible strong interactions between the variables. The model proposed evolves both the weights and the structure of the network by means of an evolutionary algorithm. Usually it is very difficult to know the most suitable structure of the network for a given problem beforehand. The evolution of the structure partially alleviates this problem. On the other hand, the algorithm encourages parsimony in evolved networks by means of the priority of the structural mutations given by deletion and fusion nodes, rather than using a regularization term in the fitness function. The empirical results show that the product-unit model performs well compared to other learning classification techniques. We obtain very promising results in terms of classification accuracy and the complexity of classifier. Moreover, we show the best model for each problem. As future work, it would be of interest to increase the number and typology of the data set considered and to try to state relationship between the level of interaction of the input variables for each data set and the level of performance obtained in the product-unit models.

REFERENCES

[1] M. Bishop, Neural Networks for Pattern Recognition: Oxford University Press, 1995.

[2] J. Friedman and W. Stuetzle, "Projection pursuit regression," Journal of the American Statistical Association, vol. 76, pp. 817-823, 1981.

[3] T. J. Hastie and R. J. Tibshirani, Generalized Additive Models. London: Chapman & Hall, 1990.

[4] C. Kooperberg, S. Bose, and C. J. Stone, "Polychotomous Regression," Journal of the American Statistical Association, vol. 92, pp. 117-127, 1997.

[5] J. Friedman, "Multivariate adaptive regression splines (with discussion)," Ann. Stat., vol. 19, pp. 1-141, 1991.

[6] R. Durbin and D. Rumelhart, "Products Units: A computationally powerful and biologically plausible extension to backpropagation networks," Neural Computation, vol. 1, pp. 133-142, 1989.

[7] R. Setiono and L. C. K. Hui, "Use of quasinewton method in a feedforward neural-network construction algorithm," IEEE Trans. Neural Networks, vol. 6, pp. 273-277, 1995.

[8] R. Reed, "Pruning algorithms-A survey," IEEE Trans. Neural Networks, vol. 4, pp. 740-747, 1993.

[9] G. F. Miller, P. M. Todd, and S. U. Hedge, "Designing neural networks using genetic algorithms," presented at Proc. 3er Int. Conf. Genetic Algorithms and Their Applications, San Mateo, CA, 1989.

[10] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," IEEE Transactions on Neural Networks, vol. 5 (1), pp. 54-65, 1994.

[11] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," IEEE Transactions on Neural Networks, vol. 8 (3), pp. 694-713, 1997.

[12] D. B. Fogel, "Using evolutionary programming to greater neural networks that are capable of playing Tic-Tac-Toe," presented at International Conference on Neural Networks, San Francisco, CA, 1993.

[13] W. Yan, Z. Zhu , and R. Hu, "Hybrid genetic /BP algorithm and its application for radar target classification," presented at Proceedings of the IEEE National Aerospace Electronics Conference, Piscataway, NJ, USA, 1997.

[14] X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," IEEE Transactions and System Man and Cybernetics-Part B: Cybernetics, vol. 28, pp. 417-425, 1998.

[15] N. García-Pedrajas, C. Hervás-Martínez, and J. Muñoz-Pérez, "Multiobjetive cooperative coevolution of artificial neural networks.," Neural Networks, vol. 15, pp. 1255-1274, 2002.

[16] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," Evolutionary Computation, vol. 10, pp. 99-127, 2002.

[17] C. Blake and C. J. Merz, " UCI repository of machine learning data bases," www.ics.uci.edu/ mlearn/MLRepository.thml, 1998.

[18] M. Schmitt, "On the Complexity of Computing and Learning with Multiplicative Neural Networks," Neural Computation, vol. 14, pp. 241-301, 2001.

[19] A. Ismail and A. P. Engelbrecht, "Training products units in feedforward neural networks using particle swarm optimisation.," presented at Development and practice of Artificial Intelligence Techniques, Proceeding of the International Conference on Artificial Intelligence, Durban, South Africa, 1999.

[20] E. A. P. Ismail A., "Global optimization algorithms for training product units neural networks," presented at International Joint Conference on Neural Networks IJCNN`2000, Como, Italy, 2000.

[21] D. J. Janson and J. F. Frenzel, "Training product unit neural networks with genetic algorithms," IEEE Expert, vol. 8, pp. 26-33, 1993.

[22] E. A. P. Ismail A., "Pruning product unit neural networks," presented at Proceedings of the International Conference on Neural Networks, Honolulu, Hawai, 2002.

[23] L. R. Leerink, C. L. Giles, B. G. Horne, et al., "Learning with products units," Advances in Neural Networks Processing Systems, vol. 7, pp. 537-544, 1995.

[24] D. B. Fogel, A. J. Owens, and M. J. Wals, Artificial Intelligence Throught Simulated Evolution. New York: Wiley, 1966.

[25] D. B. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. New York: IEEE Press, 1995.

[26] S. Kirkpatric, C. D. J. Gellat, and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, pp. 671-680, 1983.

[27] R. H. J. M. Otten and L. P. P. P. van Ginneken, The annealing algorithm. Boston, MA.: Ed. Kluwer, 1989.

[28] R. A. Fisher, "Theory of statistical estimation," Proc. of Cambridge Philosophical Soc., vol. 22, pp. 700-725, 1925.

[29] N. Landwehr, M. Hall, and F. Eibe, "Logistic Model Trees," Machine Learning, vol. 59, pp. 161-205, 2005.

[30] R. Quinlan, C4.5: Programs for Machine Learning: Morgan Kauffman, 1993.

[31] L. Breiman, H. Friedman, J. A. Olshen, et al., Classification and Regression Trees. Belmont, CA: Wadsworth, 1984.

[32] J. Gama, "Functional trees," Machine Learning, vol. 55, pp. 219-250, 2004.

[33] K. Y. Chan and W. Y. Loh, "LOTUS: An algorithm for building accurate and comprehensible logistic regression trees," Journal of Computational and Graphical Statistics, vol. 13, pp. 826-852, 2004.

[34] Y. Wang and I. Witten, "Inducing model trees for continuous classes," presented at Proceedings of Poster Papers, European Conference on Machine Learning., Prague, Czech Republic, 1997.

TABLE I
STATISTICAL RESULTS OF TRAINING AND TESTING *PCCE* FOR 30 EXECUTIONS OF PUNN MODEL (50,100 AND 200 GENERATIONS)

PCCE	Training				Test				# conn	
Balance										
# gen.	Mean	SD	Best	Worst	Mean	SD	Best	Worst	Mean	SD
50	93.90	2.12	96.59	91.04	93.21	2.57	96.79	89.74	19.00	3.23
100	94.05	2.22	97.23	91.26	92.88	2.04	96.15	91.03	18.40	4.03
200	**94.82**	2.86	**97.87**	91.26	**94.42**	2.51	**97.44**	91.03	16.00	5.56
Pima										
# gen.	Mean	SD	Best	Worst	Mean	SD	Best	Worst	Mean	SD
50	77.93	0.65	78.99	76.91	77.92	2.13	80.21	72.92	17.30	6.50
100	77.88	0.63	79.34	77.26	78.18	2.14	80.73	73.44	16.20	3.99
200	**78.65**	0.43	**79.34**	78.13	**78.70**	2.13	**82.29**	76.04	17.30	4.85
Cancer										
# gen.	Mean	SD	Best	Worst	Mean	SD	Best	Worst	Mean	SD
50	97.39	0.09	97.52	97.33	97.87	0.67	98.85	97.13	15.60	6.40
100	97.39	0.13	97.52	97.14	**97.99**	0.68	98.85	97.13	11.90	2.69
200	**97.54**	0.29	**98.10**	97.14	97.64	1.13	**99.43**	95.40	14.00	4.46
Glass										
# gen.	Mean	SD	Best	Worst	Mean	SD	Best	Worst	Mean	SD
50	65.40	1.85	68.94	62.11	**65.28**	5.22	**71.69**	56.60	28.30	8.73
100	68,13	2.2	70.18	63.97	64.91	3.41	69.81	58.49	36.30	9.79
200	**69.00**	1.8	**71.42**	65.83	64.70	2.60	67.92	60.37	30.50	5.77

TABLE II
MEAN CLASSIFICATION ACCURACY AND STANDARD DEVIATION FOR LMT, SLOGISTIC, MLOGISTIC, C4.5, CART, LOTUS USING SIMPLE LOGISTIC REGRESSION (LOTUSS) AND LOTUS USING MULTIPLE LOGISTIC REGRESSION (LOTUSM), M5' FOR CLASSIFICATION, ABOOST, LTREELOG (SEE [29]) AND PUNN MODEL

Dataset	LMT	SLogistic	MLogistic	C4.5	CART	LotusS
Balance	89.71±2.68	88.74±2.91	89.44±3.29	77.82±3.42	78.09±3.97	-
Pima	77.08 ± 4.65	77.10 ±4.65	77.47 ±4.39	74.49 ±5.27	74.50 ±4.70	75.08 ±5.14
Cancer	96.18 ±2.20	96.21 ±2.19	96.50 ±2.18	95.01 ±2.73	94.42 ±2.70	94.61 ±2.66
Glass	69.15± 8.99	65.29±8.03	63.12±4.39	67.63±9.31	68.09±10.49	-
Dataset	LotusM	M5'	ABoost(10)	ABoost(100)	LTreeLog	PUNN
Balance	-	87.76±2.23	78.35±3.78	76.11±4.09	92.78±3.49	**94.42**±2.51
Pima	77.47 ±4.39	76.56 ±4.71	71.81 ±4.85	73.89 ±4.75	76.64±4.69	**78.70**±2.13
Cancer	96.44 ±2.13	95.85 ±2.15	96.08 ±2.16	96.70 ±2.18	96.75±2.04	**97.99**±0.68
Glass	-	71.30±9.08	75.15±7.59	**78.78**±7.80	64.78±9.90	65.09±4.81

TABLE III

BEST MODELS FROM PUNN TO BALANCE, PIMA, CANCER AND GLASS DATA SETS, WHERE X_i^* REPRESENT THE TRANSFORMED VARIABLES

Balance
$\hat{g}_1(\mathbf{x}) = \frac{1}{1+\exp\{-4.312 - 1.582B_1 + 5.011B_3\}}$; $\hat{g}_2(\mathbf{x}) = \frac{1}{1+\exp\{0.982 - 2.253B_2\}}$
$\hat{g}_3(\mathbf{x}) = \frac{1}{1+\exp\{2.000 - 2.661B_3\}}$; $B_1 = (X_3^*)^{-4.005}$; $B_2 = (X_3^*)^{-0.713}$; $B_3 = (X_1^*)^{-4.636}(X_2^*)^{-4.795}(X_3^*)^{4.446}(X_4^*)^{4.805}$,
$m = 3$, # coefficients= 12
Pima
$\hat{g}_1(\mathbf{x}) = \frac{1}{1+\exp\{-4.718 + 0.754B_1 - 2.320B_2\}}$; $\hat{g}_2(\mathbf{x}) = \frac{1}{1+\exp\{2.860 - 0.166B_1 - 4.710B_2\}}$
$B_1 = (X_1^*)^{0.601}(X_2^*)^{2.700}(X_3^*)^{-0.268}(X_5^*)^{-0.178}(X_6^*)^{1.219}(X_7^*)^{1.018}$; $B_2 = (X_2^*)^{-4.046}(X_3^*)^{-3.019}(X_4^*)^{-1.248}(X_5^*)^{0.915}(X_6^*)^{3.038}(X_7^*)^{2.390}(X_8^*)^{-1.730}$,
$m = 2$, # coefficients= 19
Cancer
$\hat{g}_1(\mathbf{x}) = \frac{1}{1+\exp\{-0.686 + 1.168B_1 + 4.771B_2\}}$; $\hat{g}_2(\mathbf{x}) = \frac{1}{1+\exp\{-1.071 + 1.498B_1\}}$,
$B_1 = (X_2^*)^{0.668}(X_3^*)^{0.763}(X_6^*)^{0.245}(X_9^*)^{2.457}$, $B_2 = (X_1^*)^{-4.786}(X_2^*)^{1.438}(X_4^*)^{-3.849}(X_6^*)^{-2.839}(X_7^*)^{-2.731}(X_8^*)^{-2.790}$
$m = 2$, # coefficients= 15
Glass
$\hat{g}_1(\mathbf{x}) = \frac{1}{1+\exp\{-4.841 + 4.758B_1 + 2.666B_2\}}$; $\hat{g}_2(\mathbf{x}) = \frac{1}{1+\exp\{2.042 - 3.147B_1\}}$; $\hat{g}_3(\mathbf{x}) = \frac{1}{1+\exp\{0.653 + 0.820B_1 + 3.021B_3\}}$
$\hat{g}_4(\mathbf{x}) = \frac{1}{1+\exp\{-0.408 + 5.200B_1 + 4.334B_2 + 2.672B_3\}}$; $\hat{g}_5(\mathbf{x}) = \frac{1}{1+\exp\{3.719 + 4.363B_1 + 4.596B_2 - 0.430B_3\}}$;
$\hat{g}_6(\mathbf{x}) = \frac{1}{1+\exp\{4.292 - 3.874B_1 - 1.231B_3\}}$
$B_1 = (X_1^*)^{-1.955}(X_3^*)^{-1.199}(X_4^*)^{2.088}(X_5^*)^{1.779}(X_7^*)^{-1.879}(X_8^*)^{2.197}(X_9^*)^{1.964}$; $B_2 = (X_1^*)^{2.300}(X_3^*)^{-4.560}(X_7^*)^{-0.116}$; $B_3 = (X_9^*)^{-2.984}$
$m = 3$, # coefficients= 30