Using Genetic Programming for Artificial Neural Network Development and Simplification

DANIEL RIVERO, JULIAN DORADO, JUAN RABUÑAL, ALEJANDRO PAZOS
Department of Information & Communications Technologies
University of A Coruña
Campus Elviña, 15071, A Coruña, Spain
SPAIN
{drivero, julian, juanra, apazos}@udc.es http://sabia.tic.udc.es

Abstract: - The creation process of Artificial Neural Networks (ANNs) used to be quite slow and the human expert had to test several architectures until finding the one that achieves the best results for the solution of a certain problem. This work presents a new technique that uses Genetic Programming (GP) for automatically creating ANNs. This technique also allows the obtaining of simplified networks with few neurons for solving the problem. In order to measure the performance of the system and to compare the results with other ANN generation and training methods with Evolutionary Computation (EC) techniques, several tests were performed with problems based on some of the most used test databases. The results of those comparisons showed that the system achieved good results comparable with already existing techniques and, in most of the cases, they worked better than those techniques.

Key-Words: - Artificial Neural Networks, Evolutionary Computation, Genetic Programming, Data Mining

1 Introduction

ANNs are learning systems that have solved a large amount of complex problems related to different disciplines (classification, clustering, regression, etc.) [1]. The interesting characteristics of this powerful technique have induced its use by researchers in different environments [2].

Nevertheless, the use of ANNs has some problems, mainly related to their development process. This process can be divided into two parts: architecture development and training and validation. As the network architecture is problem-dependant, the design process of this architecture used to be manually performed, meaning that the expert had to test different architectures and train them until finding the one that achieves best results after the training process. The manual nature of the described process determines its slow performance although the recent use of ANNs creation techniques have contributed to achieve a more automatic procedure.

2 State of the Art

2.1 Genetic Programming

Genetic Programming (GP) [3] is based on the evolution of a given population. In this population, every individual represents a solution for a problem that is intended to be solved. The evolution is achieved by means of selection of the best individuals – although the worst ones also have a little chance of being selected – and their mutual combination for creating new solutions. This process is developed using selection, crossover and mutation operators. After several generations, it is expected that the population might contain some good solutions to the problem.

In GP, the codification of the solutions is in shape of trees. Therefore, the user must specify which nodes of the tree are the terminals (leaves of the tree) and the functions (nodes that have descendants) for being used by the evolutionary algorithm in order to build complex expressions.

The wide application of GP to different environments and its consequent success are due to its capability for being adapted to numerous different problems. Although the main application is the generation of mathematical expressions [4], GP has also been used in many others fields such as rule generation [5], filter design [6], etc.

2.2 ANN development with EC tools

The development of ANNs is a topic that has been extensively dealt with very diverse techniques. The world of evolutionary algorithms is no exception, and proof of that is the great amount of works that have been published about the different techniques in this area, even with GAs or GP [3] [12] [14] [18] [20]

[21]. These techniques follow the general strategy of an evolutionary algorithm: an initial population consisting of different types of genotypes, each one of them codifying different parameters (typically, the weight of the connections and/or the architecture of the network and/or the rules of learning) is randomly created. This population is evaluated in order to determine the fitness of each individual. Afterwards, this group is made to evolve repeatedly by means of different genetic operators (replication, crossover, mutation, etc.) until a determined termination criteria is satisfied (for example, a sufficiently good individual is obtained, or that a predetermined maximum number of generations is reached).

As a general rule, the field of ANN generation using evolutionary algorithms is divided into three main fields: evolution of weights, architectures and learning rules.

First, the weight evolution starts from an ANN with an already determined topology. In this case, the problem to be solved is the training of the connection weights, attempting to minimize the network error. Most of training algorithms, such as back-propagation (BP) algorithm, are based on gradient descent, which has several drawbacks [7]. The most important is the possibility of getting stuck into a local minimum of the fitness function. With the use of an evolutionary algorithm, the weights can be represented either as a concatenation of binary values [8] or of real numbers [9]. The main disadvantage of this type of encoding is the permutation problem. This problem means that the order in which weights are taken at the string might cause that equivalent networks correspond to completely different chromosomes, making the crossover operator inefficient.

Second, the evolution of architectures includes the generation of the topological structure. This means establishing the connectivity and the transfer function of each neuron. The network architecture is highly important for the successful application of the ANN, since the architecture has a very significant impact on the processing ability of the network. Therefore, the network design, traditionally performed by a human expert using trial and error techniques on different architectures, is crucial. The automatic architecture design has been possible thanks to the use of evolutionary algorithms. In order to use them to develop ANN architectures, it is needed to choose how to encode the genotype of a given network for it used by the genetic operators.

At the first option, direct encoding, there is a one-to-one correspondence between each of the genes and their subsequent phenotypes. The most typical encoding method consists of a matrix that represents an architecture where every element reveals the presence or absence of connection between two nodes [10]. These types of encoding are generally quite simple and easy to implement. However, they also have a large amount of inconveniences such as scalability [11], the incapability of encoding repeated structures, or permutation [12].

Apart from direct encoding, there are some indirect encoding methods. In these methods, only some characteristics of the architecture are encoded in the chromosome. These methods have several types of representation. First, the parametric representations represent the network as a group of parameters such as number of hidden layers, number of nodes for each layer, number of connections between two layers, etc [13]. Although the parametric representation can reduce the length of the chromosome, the evolutionary algorithm performs the search within a restricted area in the search space containing all the possible architectures. Another non direct representation type is based on grammatical rules [11]. In this system, the network is represented by a group of rules, with the shape of production rules, that make a matrix that represents the network, which has several restrictions.

The growing methods represent another type of encoding. In this case, the genotype does not encode a network directly. Instead of it, it contains a group of instructions for building up the phenotype. The genotype decoding will consist on the execution of those instructions [14].

With regards to the evolution of the learning rule, there are several approaches [15], although most of them are only based on how learning can modify or guide the evolution and also on the relationship among the architecture and the connection weights.

3 Model

The GP-development of ANNs is performed by means of the GP typing property [16]. This property provides the ability of developing structures that follow a specific grammar. In this case, the nodes to be used will be the following:

- ***ANN***. Node that defines the network. It appears only at the root of the tree. It has the same number of descendants as the network expected outputs, each of them a neuron.
- ***n-Neuron***. Node that identifies a neuron with n inputs. This node will have 2*n descendants. The first n descendants will be other neurons, either input or hidden ones. The second n descendants will be arithmetical sub-trees. These sub-trees

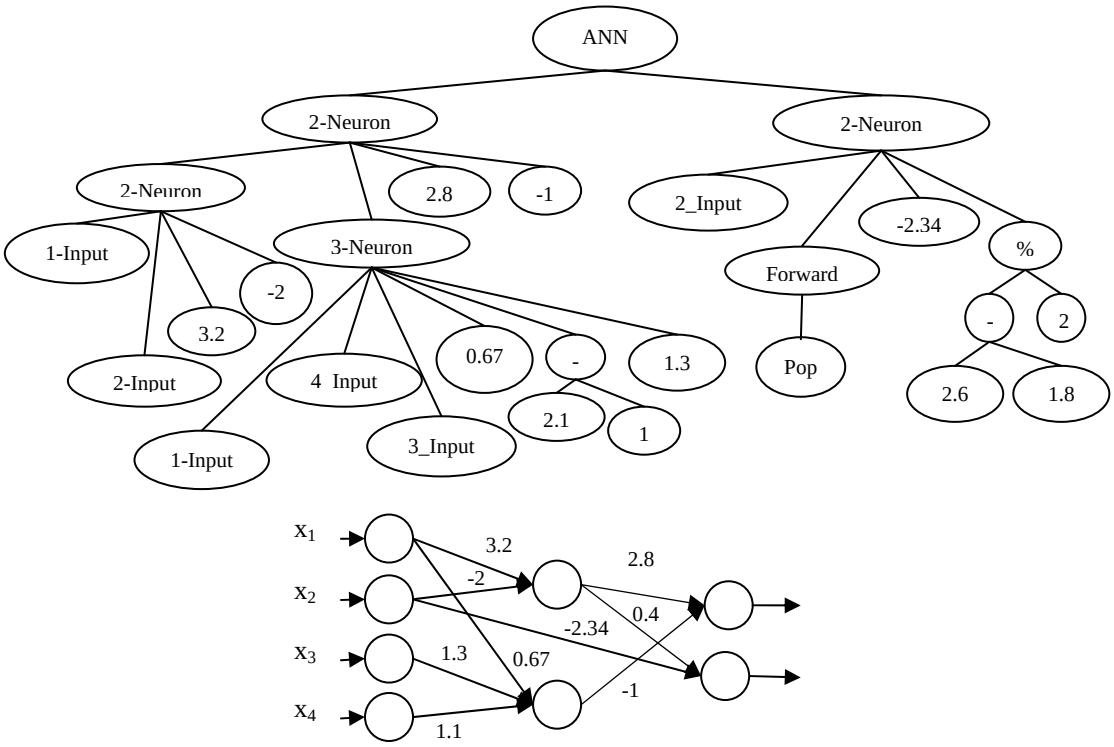

Fig. 1. GP tree and its resulting network

represent real values. These values correspond to values of the respective connection weights of the input neurons – the first descendants – of this neuron.

- ***n-Input*** neuron. Nodes that define an input neuron which receives its activation value from the input variable n. These nodes do not have any descendants.
- Finally, the arithmetic operators set {+,-,*,%}, where % designs the operation of protected division (returns 1 as result if the divisor is 0). They will generate the values of connection weights (sub-trees of the n-Neuron nodes). These nodes perform operations among constants in order to obtain new values. As real values are also needed for such operations, they have to be introduced by means of the addition of random constants to the terminal set in the range [-4, 4].

ANNs can be generated with these operator sets. However, these networks would not allow, for a given neuron, the existence of output connections to more than one different neuron. For such reason, the system has been endowed with a list where neurons are being added as the tree is being evaluated, and an index that points to a specific element of the list. In order to extract neurons from the list, and therefore to operate with it, the operator sets were added with the following operators:

- ***"Forward"***. This node advances the index list one unit. This node has one descendant.
- ***"Pop"***. This node extracts from the list the neuron at the position pointed by the index. This node substitutes the evaluation of a neuron, as it returns an already existing one, so it has no descendants.

Every time a neuron is created, it is added to the list once its descendants have been evaluated. In such way, they are not allowed to reference that neuron, so recurrent links will be avoided.

Note that, during the creation of a neuron, a given neuron - either an input or a hidden - can be repeated several times as input of that neuron. In such case, there is no new input connection from that processing element, but the weight of the already existing connection will be added with the value of the new connection.

Once the tree has been evaluated, the genotype turns into phenotype. In other words, it is converted into an ANN with its weights already set (thus it does not need to be trained) and therefore can be evaluated. The evolutionary process demands the assignation of a fitness value to every genotype. Such

value is the result after the evaluation of the network with the pattern set that represents the problem. This result is the mean square error (MSE) of this evaluation.

Nevertheless, this error value considered as fitness value has been modified in order to induce the system to generate simple networks. The modification has been made by adding a penalization value multiplied by the number of neurons of the network. In such way, and given that the evolutionary system has been designed in order to minimize an error value, when adding a fitness value, a larger network would have a worse fitness value. Therefore, the existence of simple networks would be preferred as the penalization value that is added is proportional to the number of neurons of the ANN. The calculus of the final fitness is as follows:

$$fitness = MSE + N * P$$

Where MSE is the mean square error of the ANN in the training patterns set, N is the number of neurons of the network and P is the penalization value for such number.

4 Problems to be solved

This technique has been used for solving problems of different complexity taken from the UCI database [17]. All these problems are knowledge-extraction problems from databases where, taking certain features as a basis, it is intended to perform a prediction about another attribute of the database. The value that is intended to be predicted might be a diagnosis value (when using medical databases), a classification value or a prediction one. A small summary of the problems to be solved can be seen on Table 1.

	Number of inputs	Number of data points	Number of outputs
Breast Cancer	9	699	1
Iris Flower	4	150	3
Heart disease	13	303	1
Ionosphere	34	351	1

Table 1. Summary of the problems to be solved

All these databases values have been normalized between 0 and 1 and the patterns divided into two parts for each problem, taking the 70% of the database for training and using the remaining 30% for performing tests.

5 Results

Several experiments have been performed in order to evaluate the system performance. The values taken for the parameters at these experiments were the following:

- Crossover rate: 95%.
- Mutation probability: 4%.
- Selection algorithm: 2-individual tournament.
- Creation algorithm: Ramped Half&Half.
- Population size: 500.
- Tree maximum height: 6.

		0.1	0.01	0.001	0.0001	0.00001	0
Breast Cancer	Neurons	1.75	2	2.65	9.3	22.65	27.833
	Connections	8.8	9.2	13.1	47.6	100.7	126.944
	Training	0.03371	0.01968	0.01801	0.01426	0.01366	0.01257
	Test	0.03392	**0.02063**	0.02096	0.02381	0.02551	0.02514
Iris Flower	Neurons	3	4	8.45	24.2	38.9	42.85
	Connections	8.8	11.95	27.85	86.55	140.25	157.05
	Training	0.06079	0.03021	0.01746	0.01658	0.01681	0.01572
	Test	0.07724	0.05222	**0.03799**	0.04084	0.04071	0.04075
Heart Cleveland	Neurons	1	1.3	4.95	16.4	24.8	30.4
	Connections	7.25	8.7	28.45	88.1	118.95	167.5
	Training	0.10589	0.10731	0.07517	0.06867	0.06600	0.06218
	Test	**0.15728**	0.16335	0.17654	0.17816	0.17472	0.17292
Ionosphere	Neurons	1	2.35	8.15	21.6	32.4	42.45
	Connections	6.1	11.95	48.3	128.15	197.4	261.8
	Training	0.09836	0.06253	0.03097	0.02114	0.02264	0.01781
	Test	0.10941	0.09127	**0.06854**	0.07269	0.07393	0.07123

Table 2. Comparison of the results with different penalization values

	Proposed here			[18]			
	Inputs number	Hidden neurons number	Output neurons number	Inputs number	Hidden neurons number	Output neurons number	BP Epochs
Breast cancer	9	1.65	1	9	5	1	20
Iris Flower	4	5.45	3	4	5	3	80
Heart Cleveland	13	3.95	1	26	5	1	40
Ionosphere	34	7.15	1	34	10	1	40

Table 3. Comparison of the architectures used

- Maximum inputs for each neuron: 12.

Table 2 shows a comparison of the results obtained for different penalization values. The values range from very high (0.1) to very small (0.00001 or 0). High values only enables the creation of very small networks with a subsequent high error, and low values lead to overfitting problem. This overfitting can be noticed at the table in the training error decrease together with a test error increase.

The number of neurons as well as of connections that were obtained at the resulting networks is also shown in table 2. Logically, such number is higher as the penalization decreases. The results correspond to the MSE obtained after both, the training and the test of every problem. As it can be observed, the results clearly prove that the problems have been satisfactorily solved and, as far as penalization parameter is concerned, intermediate values are preferred for the creation of networks. These intermediate values in the penalization parameter allow the creation of networks large enough for solving the problem, avoiding overfitting, although it should be changed for every problem.

6 Comparison with other methods

In order to evaluate its performance, the system presented here has been compared with other methods for ANN generation and training.

The method 5x2cv [19] is used in [18] for the comparison of different evolutionary methods-based techniques used in ANN generation and training. This work presents as results the average precisions obtained in the 10 test results generated by this method. Such values are the basis for the comparison of the described technique with other well known ones.

Regarding the parameters used for ANN generation, the comparison was done using the same ones than in section 5, with 0.00001 penalization value.

The algorithms compared with this technique are widely explained with detail in [18]. Such work shows the average times needed to achieve the results. Not having the same processor that was used, the computational effort needed for achieving the results can be estimated. This effort represents the number of times that the pattern file was evaluated. The computational effort for every technique can be measured using the population size, the number of generations the number of times that the BP algorithm was applied, etc. This calculation varies for every algorithm used. All the techniques that are compared with the work are related to the use of evolutionary algorithms for the ANN design. Five iterations of a 5-fold cross-validation test [20] were performed in all these techniques in order to evaluate the accuracy of this technique. The mentioned techniques are the following:

- Connectivity matrix.
- Pruning.
- Parameter search.
- Graph rewriting grammar.

Table 3 shows a summary of the number of neurons used in [18] in order to solve the problems that were used with connectivity matrix and pruning techniques. Such number is compared with the average neurons number of the ANNs (table 2) that solved the problem with the method proposed here. The epoch number of the BP algorithm, when used, is also indicated here.

	Connectivity matrix	Pruning	Parameters	Grammar
Chromosome length (L)	(hidden+output)*input + output*hidden	(hidden+output)*input + output*hidden	36	256
Population size	$\lfloor 3\sqrt{L} \rfloor$	$\lfloor 3\sqrt{L} \rfloor$	25	64
Crossover points	l/10	l/10	2	l/10
Mutation rate	1/L	1/L	0.04	0.004

Table 4. Parameters of the techniques used during the comparison

	Matrix	Pruning	Parameters	Grammar
Breast cancer	96.77 96.19 92000	96.31 96.10 4620	96.69 96.19 100000	96.71 96.15 300000
Iris	92.40 94.98 320000	92.40 82.99 4080	91.73 94.95 400000	92.93 95.27 1200000
Heart Cleveland	76.78 80.53 304000	89.50 79.45 7640	65.89 80.66 200000	72.8 80.46 600000
Ionosphere	87.06 88.65 464000	83.66 83.21 11640	85.58 88.13 200000	88.03 88.71 600000
Average	88.25 90.08	90.46 85.43	84.97 89.98	87.61 90.14

Table 5. Comparison of the results obtained with other methods and with the present one

Table 4 shows the parameter configuration used by these techniques. The execution was stopped after 5 generations with no improvement or after 50 total generations.

The results obtained with these 4 methods are shown in table 5. Every box of the table indicates 3 different values: precision value obtained in [18] (left), computational effort needed for obtaining such value with that technique (below) and precision value obtained with the technique described here and related to the previously mentioned computational effort value (right).

Watching this table, it is obvious that the results obtained with the method proposed here are, not only similar to the ones presented in [18], but better in many cases. The reason of this lies in the fact that these methods need a high computational load since training is necessary for every case of network (individual) evaluation, which therefore turns to be time-consuming. During the work described here, the design and training procedures are performed simultaneously, and therefore, the times needed for designing as well as for evaluating the network are combined.

Most of the techniques used for the development of ANNs are quite costly, due in some cases to the combination of training with architecture evolution. The technique described here can achieve good results with a low computational cost and besides, the added advantage is that, not only the architecture and the connectivity of the network are evolved, but also the network itself undergoes an optimization process.

Table 5 also shows a small overfitting problem. This is due to the fact that the system has been left to training up to a certain number of fitness function evalutions. This usually leads to overfitting the training set when it keeps training for a long time.

7 Conclusions

This paper presents a technique for ANN generation with GP. This system has been compared to different techniques that use evolutionary algorithms for generating and training networks. The conclusion of such comparison is that the results of 5x2cv tests using this method are not only comparable to those obtained with other methods, but also better than them in most of the cases. It should be borne in mind that if the parameters of the system were adapted to every problem to be solved the results would have been better. However, the parameters used were the same for all the problems because it is intended that the whole of them might serve for any problem, trying to remove the participation of the expert during algorithm application. In such way, it can be stated that even without human participation, this method can improve the results of other algorithms.

This system has another advantage over other methods of ANN generation since -after a short analysis by the system- it is possible to differentiate the variables that are not relevant for problem solving, as they will not be present in the ANN.

8 Future Works

Once the system has been proved, the work continues towards several directions. One interesting research line would be the possible integration of a GA into the system in order to train the networks that were generated. In such way, the GP system would only create different architectures that would be trained by the GA.

As was explained earlier, this system has an overfitting problem. Another research line could be to use any technique to avoid overfitting, such as early stop.

9 Acknowledgements

This work was supported in part by the Spanish Ministry of Education and Culture (Ref. TIC2003-07593, TIN2006-13274), the IMBIOMED network (Ref P10/52048) financed by the Carlos III Health Institute, grants from the General Directorate of Research of the Xunta de Galicia (Ref. PGIDIT03-PXIC10504PN PGIDIT04-PXIC10503PN, PGIDIT04-PXIC10504PN), and the European project Interreg (Ref. IIIA-PROLIT-SP1E194/03).

The development of the experiments described in this work, has been performed with equipments belonging to the Super Computation Center of Galicia (CESGA).

The Cleveland heart disease database was available thanks to Robert Detrano, M.D., Ph.D., V.A. Medical Center, Long Beach and Cleveland Clinic Foundation.

References:

[1] Haykin, S., *Neural Networks (2nd ed.)*, Englewood Cliffs, NJ: Prentice Hall, 1999.

[2] Rabuñal, J.R., Dorado J., (eds.) *Artificial Neural Networks in Real-Life Applications*, Idea Group Inc, 2005.

[3] Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA, MIT Press, 1992.

[4] Rivero D., Rabuñal J.R., Dorado J., Pazos A., Time Series Forecast with Anticipation using Genetic Programming, *IWANN 2005*, 2005, pp. 968-975.

[5] Bot, M., *Application of Genetic Programming to Induction of Linear Classification Trees*, Final Term Project Report, Vrije Universiteit, Amsterdam, 1999.

[6] Rabuñal J.R., Dorado J., Puertas J., Pazos A., Santos A., Rivero D., Prediction and Modelling of the Rainfall-Runoff Transformation of a Typical Urban Basin using ANN and GP, *Applied Artificial Intelligence*, 2003.

[7] Sutton, R.S., Two problems with backpropagation and other steepest-descent learning procedure for networks, *Proc. 8th Annual Conf. Cognitive Science Society*, Hillsdale, NJ: Erlbaum, 1986, pp. 823-831.

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

[9] Greenwood G.W. Training partially recurrent neural networks using evolutionary strategies, *IEEE Trans. Speech Audio Processing*, vol. 5, 1997, pp. 192-194.

[10] Alba E., Aldana J.F., Troya J.M., Fully automatic ANN design: A genetic approach, *Proc. Int. Workshop Artificial Neural Networks (IWANN'93), Lecture Notes in Computer Science*, vol. 686. Berlin, Germany: Springer-Verlag, 1993, pp. 399-404.

[11] Kitano H., Designing neural networks using genetic algorithms with graph generation system, *Complex Systems*, vol. 4, 1990, pp. 461-476.

[12] Yao X., Liu Y., Towards designing artificial neural networks by evolution, *Appl. Math. Computation*, vol. 91, no. 1, 1998, pp. 83-90.

[13] Harp S.A., Samad T., Guha A., Towards the genetic synthesis of neural networks, *Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications*, J.D. Schafer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 360-369.

[14] Nolfi S. & Parisi D., Evolution of Artificial Neural Networks, *Handbook of brain theory and neural networks, Second Edition*, Cambridge, MA: MIT Press, 2002, pp. 418-421.

[15] Turney P. Whitley D., Anderson R., Special issue on the baldwinian effect, *Evolutionary Computation*, vol. 4, no. 3, 1996, pp. 213-329.

[16] Montana D.J., Strongly typed genetic programming, *Evolutionary Computation*, Vol. 3, No. 2, 1995, pp. 199-200.

[17] Mertz C.J., Murphy P.M., UCI repository of machine learning databases. http://www-old.ics.uci.edu/pub/machine-learning-databases, 2002

[18] Cantú-Paz E., Kamath C., An Empirical Comparison of Combinations of Evolutionary Algorithms and Neural Networks for Classification Problems, *IEEE Transactions on systems, Man and Cybernetics – Part B: Cybernetics*, 2005, pp. 915-927.

[19] Dietterich T.G., Approximate statistical tests for comparing supervised classification learning algorithms, *Neural Computation*, Vol. 10, No. 7, 1998, pp. 1895-1924.

[20] Herrera F., Hervás C., Otero J., Sánchez L., Un estudio empírico preliminar sobre los tests estadísticos más habituales en el aprendizaje automático, R. Giraldez, J.C. Riquelme, J.S. Aguilar (Eds.) *Tendencias de la Minería de Datos en España, Red Española de Minería de Datos y Aprendizaje*, 2004, pp. 403-412.

[21] Gruau F., Genetic Micro Programming of Neural Networks, *Advances in Genetic Programming*, K. Kinnear, Ed., Cambridge, MA:MIT Press, 1994, pp. 495-518.