Red-Black Tree Based NeuroEvolution of Augmenting Topologies

William R. Arellano[1], Paul A. Silva[1], Maria F. Molina[1], Saulo Ronquillo[1], and Francisco Ortega-Zamorano[2](✉)

[1] School of Mathematical and Computational Sciences, University of Yachay Tech, San Miguel de Urcuquí, Ecuador

[2] Department of Computer Science, University of Málaga, Málaga, Spain
fortega@lcc.uma.es

Abstract. In Evolutionary Artificial Neural Networks (EANN), evolutionary algorithms are used to give an additional alternative to adapt besides learning, specially for connection weights training and architecture design, among others. A type of EANNs known as *Topology and Weight Evolving Artificial Neural Networks* (TWEANN) are used to evolve topology and weights. In this work, we introduce a new encoding on an implementation of NeuroEvolution of Augmenting Topologies (NEAT), a type of TWEANN, by adopting the Red-Black Tree (RBT) as the main data structure to store the connection genes instead of using a list. This new version of NEAT efficacy was tested using as case of study some data sets from the UCI database. The accuracy of networks obtained through the new version of NEAT were compared with the accuracy obtained from feed-forward artificial neural networks trained using back-propagation. These comparisons yielded that the accuracy were similar, and in some cases the accuracy obtained by the new version were better. Also, as the number of patterns increases, the average number of generations increases exponentially. Finally, there is no relationship between the number of attributes and the number of generations.

Keywords: NEAT · Red-black tree · Back-propagation · Classification

1 Introduction

An Artificial Neural Network (ANN) is defined as a computational model that simulates a biological representation of the interconnections of the units of the nervous system, the neurons [2]. Nowadays, ANNs are used widely in many different fields, such as natural resources management [7], financial applications [10], and health [3]. The architecture of an artificial neural network defines the number of layers, the number of neurons per layer and how the interconnections of the neurons are performed [2].

ANNs have been broadly studied in recent years, mainly considering their positive results in learning process for different engineering fields. This learning is

© Springer Nature Switzerland AG 2019
I. Rojas et al. (Eds.): IWANN 2019, LNCS 11507, pp. 678–686, 2019.
https://doi.org/10.1007/978-3-030-20518-8_56

achieved through an algorithm experience process that customizes the connection weights to perform tasks. As a consequence, the functionality and feasibility in which the ANN can be implemented for certain fields depends in a great percentage on the complexity patterns that the system implies, and the kind of ANN that is chosen to work with [11].

Deep artificial neural networks help to solve some optimization problems better than shallow networks. Defining which is the most suitable topology for a network can be a very time consuming endeavor as these are complex in nature [5]. Parameters as the number of layers or, the number of neurons per layer are not easy to choose. Other shortcoming is that one topology might work well for a problem but under-perform on others as the topologies are chosen taking in consideration the boundaries of those problems.

Genetic Algorithms (GA) were described by Holland in [4]. For Holland, a GA is a method of Darwinian selection that in conjunction with genetics-inspired operators (crossover, mutation, inversion and selection), moved a population of chromosomes to a new one. A chromosome consists of genes that express a particular trait in the solution. The selection operator is used to choose the chromosomes that will be allowed to reproduce. The crossover operator creates two new offspring by exchanging parts of two chromosomes. The mutation operator randomly changes the value of genes. Finally, the inversion operator reorders the genes structure, by selecting a fragment of the genome and then inverting it.

In EANNs, evolutionary algorithms are used to add another way of adaptation besides learning, where the algorithm is used to perform, and connection weights training and architecture design [11]. EANNs where topology and weights evolve are known as TWEANNs. Many methods for incrementally evolving TWEANNs have been implemented, for example: NEAT [9], *Evolutionary Acquisition of Neural Topologies* (EANT) [6], and *Hypercube-based NeuroEvolution of Augmenting Topologies* (HyperNEAT) [8]. All these methods start from a minimal structure and add any necessary complexity to the topology along the evolutionary process.

In this work we introduce a new encoding format for the structure of NEAT. We define this new encoding as Red-Black Tree based NeuroEvolution of Augmenting Topology (RBTNEAT) because it characterizes the network topology and the connection weights using a typical implementation of NEAT but adopting the RBT as the main data structure to store the connection genes instead of a list. This new approach is inspired by the fact that basic dynamic-set operations such as access and search take $O(m \log(\frac{n}{m}+1))$ time in the worst case for RBT, whereas those operations take $O(n \cdot m)$ time in the worst case for lists [1], where the values of m and n represent the size of the sets (number of elements in the trees) to be compared, in this case it would be the size of the two genomes to be compared. We compare its performance against a network with back-propagation, using several data sets from the UCI Data Set Problems. Then, its behavior is observed when the number of attributes and patterns vary.

2 Methodology

2.1 NEAT

The process of the algorithm NEAT is described in Algorithm 1. The first population is created by the function *initialPopulation*, and it is composed of networks with connections only between inputs and outputs. To preserve the innovation in NEAT, the population is put through a process of speciation, where the population is divided into species based on a distance function. Then, the population is evaluated and the best fitness is obtained. To measure the genome's fitness Eq. 1 is used, where N is the number of patterns, z is the expected output and y the output of the network. The main part of the algorithm is the while loop, where in each iteration (generation) a new population is obtained by first selecting the parents genomes, then applying the crossover operation over these parents and finally mutating this new population.

$$f = 1 - \frac{1}{N}\sum_{i=1}^{N}(z_i - y_i)^2 \tag{1}$$

Algorithm 1. NEAT

```
1: procedure NEAT(iterMax, fitnessThreshold)
2:     $population \leftarrow initialPopulation()$
3:     $species \leftarrow speciation(population)$
4:     $iter \leftarrow 0$
5:     $bestFitness \leftarrow evaluate(population)$
6:     while $iter < iterMax$ & $bestFitness < fitnessThreshold$ do
7:         $parents \leftarrow selection(population, species)$
8:         $population \leftarrow crossover(parents)$
9:         $population \leftarrow mutate(population)$
10:        $bestFitness \leftarrow evaluate(population)$
11:        $species \leftarrow speciation(population)$
12:        $iter \leftarrow iter + 1$
13:    $bestGenome \leftarrow best(population)$
14:    return $bestGenome$
```

In the original NEAT, genomes were encoded as a list of nodes and connection genes. In this work, we introduce a new genome encoding that uses RBT as the data structure to store the connection genes. More details of this new approach are explained in the following section.

2.2 RBTNEAT

In this new approach, a RBT is chosen to store the connection genes as it allows fast crossover, whereas a list is still used to store the nodes. A connection gene indicates the nodes to be connected, the weight, the *innovation number* which is used to track genes that express the same structure, and whether or not the connection is expressed. The connections genes tree is constructed based on each gene's innovation number. For example, Fig. 1 shows how the genome looks like. And, it can be observed that the node genes are stored as a list, whereas the connection genes are stored as a tree. Figure 2 shows the phenotype, the expression of the example genome.

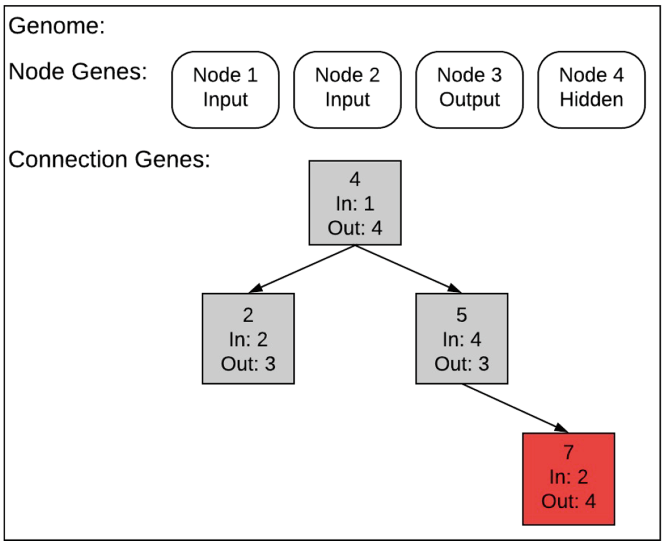

Fig. 1. An example of genome using RBTNEAT encoding.

Crossover is the process whereby two genomes are lined up and exchange their genes to produce offspring. This alignment of the genomes is based on each gene's innovation number. The matching genes are those with matching innovation numbers, whereas the genes without matching innovation numbers are known as disjoint or excess genes. The offspring is obtained by choosing the matching genes from the most fit parent, and choosing randomly the non-matching genes (disjoint or excess).

In NEAT and RBTNEAT[1], the sets of matching and non-matching are calculated by using set operations on lists and red-black trees, respectively. The set of matching genes is obtained by their interception, and the non-matching genes is obtained by their symmetric difference. For lists these set operations have worst time complexity $O(n \cdot m)$, whereas for red-black trees these operations have worst time complexity $O(m \log(\frac{n}{m} + 1))$. Thus, red-black trees are faster for calculating the matching and non-matching genes.

[1] Source code available at https://github.com/cptrodolfox/rbtneat.

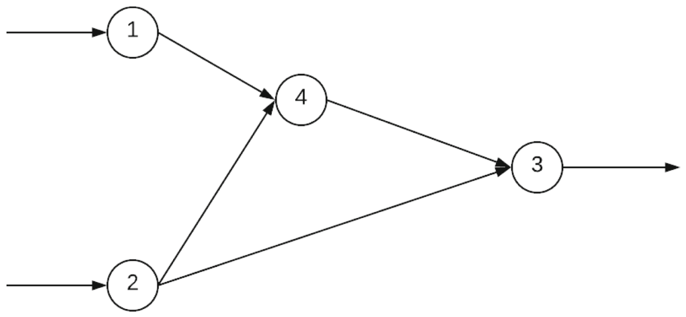

Fig. 2. The example's phenotype (i.e. the expressed neural network).

The sets of matching and non-matching genes are required to calculate the distance between genomes which is used in the process of speciation of the population, the distance is calculated by Eq. 2 where K is the number of non-matching genes, N the number of connection genes in the larger genome, and $\overline{W}$ the average weight differences of matching genes.

$$\delta = c_1 \frac{K}{N} + c_2 \overline{W} \tag{2}$$

3 Results

In order to evaluate RBTNEAT, its accuracy was first compared to the accuracy of a feed-forward artificial neural network with back-propagation on several classification sets of real world medical problems from the UCI database. The classification ability was obtained using a sigmoid activation function.

The network architecture consisted of 10 neurons to obtain the best mean of accuracy for the data sets, and had a learning rate of 0.2 where the synaptic weights were randomly initialize between 1 and -1. These tests were carried out using a 10-fold cross-validation for the training and generalization sets respectively, including a validation set to decide the termination criterion (20% of the training set) with a Hamming window to reduce the oscillations and a maximum of 1000 epochs.

RBTNEAT had a population of 150 networks (genomes) and each started as a full connected network. It ran either 1000 generations or up to it reached a fitness threshold. The probability of adding a new connection was of 0.5 and the probability of adding a new node was of 0.2. A 10-fold cross-validation was also performed on RBTNEAT. The classification ability of both methods can be seen in Table 1.

For evaluating the performance of the approach the time per generation and the number of generations was compared to the number of patterns. Figure 3 shows the evolution of the time execution and the number of generations as a function of the numbers of patterns of the dataset. The card dataset (see

Table 1. Back-propagation and RBTNEAT classification ability in terms of accuracy using several data sets from the UCI Data Sets

Method	# Attributes	# Inputs	BP	RBTNEAT
Blood transfusion	5	748	78.92	79.28
Cancer	9	683	95.56	96.48
Card	51	690	84.10	85.80
Climate	18	540	91.65	94.44
Diabetes	8	768	75.25	76.96
heartc	35	303	79.7	81.15
Ionosphere	34	351	89.23	87.76
Sonar	60	208	75.55	74.48
Statlog	13	270	80.04	83.70
Vertebral column	6	310	85.16	84.52

Table 1) was used to perform this study in which the number of patterns for the learning process was selected from 70 to all patterns (690) in steps of 70. Also, another test was carried out to analyze the impact on the time used per generation and the number of generations if the numbers of attributes (inputs) increase. The card dataset (see Table 1) was also used to perform this study in which the number of attributes for the learning process was selected from 5 to all attributes (50) in steps of 5 (Fig. 4).

4 Discussion

In Table 1 it can be seen that using red-black tree as a main data structure to store the connection genes gives an improvement in 7 of 10 data sets. This means that changing the data structure did not worsen the ability of getting a good classification.

A linear relationship between the number of patterns and the average time per generation is obtained. This may be caused by the increment of operations employed to select the parents genomes done in each species. Additionally, we realize that the average number of generations has exponential growth with respect to the number of patterns. This could be related to speciation. In which more patterns involve more individuals that are protected by competing only within their species. In fact, the speciation step seems to be a critical for the performance of the algorithm but it is also crucial to have a good solution.

In addition, we can appreciate that there is not apparent relationship between the number of attributes and the number of generations. This could be because NEAT make a sort of selection of the attributes that are more relevant to the output in order to get the simplest network possible. The RBTNEAT network on Fig. 5 is an example. Four attributes of the network are unconnected to the rest of the network implying that those attributes are not particularly important

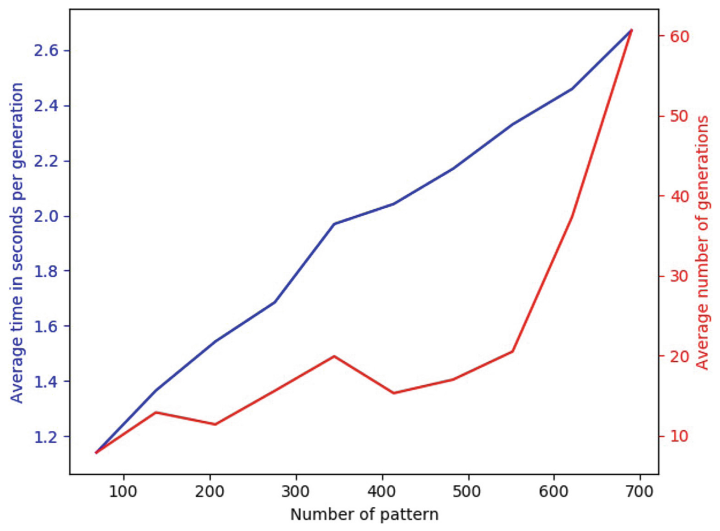

Fig. 3. Evolution of the time execution and the number of generations as a function of the numbers of patterns of the data set

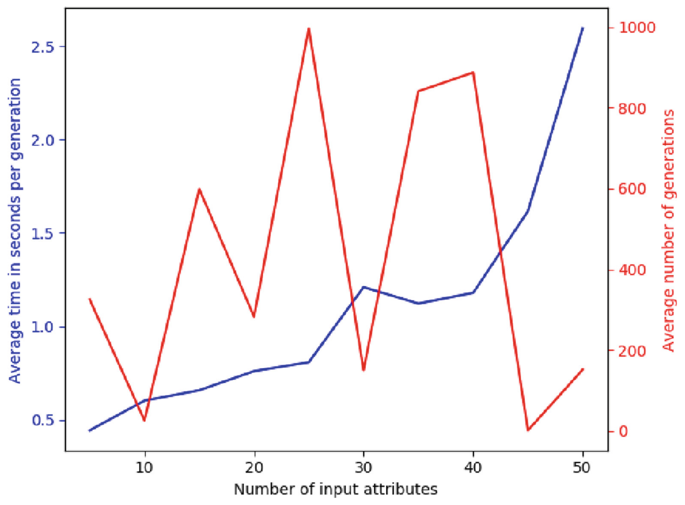

Fig. 4. Time used per generation and the number of generations if the numbers of attributes (inputs) increase.

when predicting output. However, the average time per generation seems to have an exponential relationship with the number of input attributes. The grown of the amount with the number of initial nodes might be the cause.

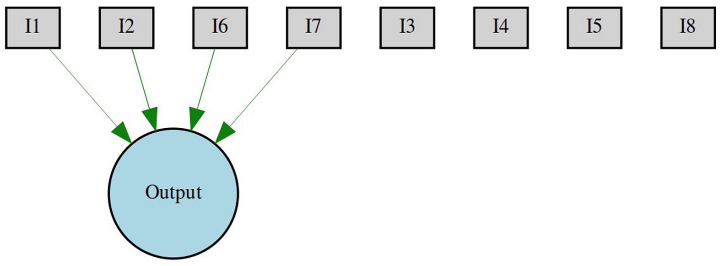

Fig. 5. RBTNEAT example of a winning network for the diabetes data set.

5 Conclusion

RBTNEAT proves to be an effective method to find a well performing topology for an artificial neural network. Results shows that RBTNEAT have similar performance than a feed forward network with back-propagation. Additionally, the number of generations seems to be independent to the number of attributes, but not to the number of patterns. Moreover, the increment in the amount of patterns and attributes cause an increment time per generation. This is due to the growth of the number of operations done to find the parents' genome and evaluating the population.

In future work, RBTNEAT should be analyzed with an initial population of partially connected networks, meaning not all nodes are connected. As, NEAT in general creates a minimal structure, if we start the population with complex structures the resulting search tree increases in size.

Acknowledgements. The authors acknowledge support through grant TIN2017-88728-C2-1-R from MICINN (Spain) that includes FEDER funds and from Plan Propio from Universidad de Málaga (Spain) and the Yachay Tech University (Ecuador).

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, Cambridge (2009)
2. Floreano, D., Du, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evol. Intell. **1**(1), 47–62 (2008)
3. Hansapani Rodrigo, C.P.T.: Artificial neural network model for predicting lung cancer survival. J. Data Anal. Inf. Process. **5**, 33–47 (2017)
4. Holland, J.H.: Adaptation in Natural and Artificial Systems, 1st edn. University of Michigan Press, Ann Arbor (1975)
5. Hunter, D., Yu, H., Pukish III, M.S., Kolbusz, J., Wilamowski, B.M.: Selection of proper neural network sizes and architectures-a comparative study. IEEE Trans. Industr. Inf. **8**(2), 228–240 (2012). https://doi.org/10.1109/TII.2012.2187914
6. Kassahun, Y., Sommer, G.: Efficient reinforcement learning through evolutionary adquisition of neural topologies. In: Proceedings of the 13th Annual European Symposium on Artificial Neural Networks, ESANN. d-side Publishing, Bruges, April 2005

7. Saman Mohammadi, M.S.: Application of artificial neural networks in order to predict mahabad river discharge. Open J. Ecol. **6**, 427–434 (2016)
8. Stanley, K.O., D'Ambrosio, D., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life J. **15**(2), 185–212 (2009)
9. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**, 99–127 (2002)
10. Vincenzo Pacelli, M.A.: An artificial neural network approach for credit risk management. J. Intell. Learn. Syst. Appl. **3**, 103–112 (2011)
11. Yao, X.: Evolving artificial neural networks. Proc. IEEE **87**(9), 1423–1447 (1999)