J. KSIAM Vol.9, No.2, 83-89, 2005

A NEW ALGORITHM OF EVOLVING ARTIFICIAL NEURAL NETWORKS VIA GENE EXPRESSION PROGRAMMING

KANGSHUN LI, YUANXIANG LI, HAIFANG MO, ZHANGXIN CHEN

ABSTRACT. In this paper a new algorithm of learning and evolving artificial neural networks using gene expression programming (GEP) is presented. Compared with other traditional algorithms, this new algorithm has more advantages in self-learning and self-organizing, and can find optimal solutions of artificial neural networks more efficiently and elegantly. Simulation experiments show that the algorithm of evolving weights or thresholds can easily find the perfect architecture of artificial neural networks, and obviously improves previous traditional evolving methods of artificial neural networks because the GEP algorithm imitates the evolution of the natural neural system of biology according to genotype schemes of biology to crossover and mutate the genes or chromosomes to generate the next generation, and the optimal architecture of artificial neural networks with evolved weights or thresholds is finally achieved.

1. INTRODUCTION

The artificial neural network (ANN) [3, 4] imitates natural neural networks of biology of animal brains to solve complex social problems, such as a classifier problem of diseases. The technology of ANNs is an interdisciplinary area that involves neuroscience, mathematics, statistics, physics, computer science, and engineering. ANNs are divided into two classes: supervised and unsupervised. They have been applied to almost all the fields, such as modeling, time series analysis, pattern identification, and signal process and control. A biological neuron may have million different inputs, and may send its outputs to many other neurons; neurons act in a three-dimensional space pattern. Therefore, the neural networks of biological brains are much more complex than any artificial neural network. Despite this fact, the artificial neural network can handle many difficult, nonlinear, and complex processes.

To simulate a biological neural network more closely, we need to design an artificial neural network by using algorithms that are random and close to the nature. There

Key words and phrases: gene expression programming, evolutionary, artificial neural network, genotype, phenotype.

This work is partly supported by the National Natural Science Key Foundation of China with Grant No. 60133010, the Research Project of Science and Technology of Education Department of Jiangxi Province with Grant No. Gan-Jiao-Ji-Zi [2005] 150, and the Key Laboratory of High Performance Computing Technology of Jiangxi Province with Grand No. JXHC-2005-003.

are many traditional designing algorithms of artificial neural networks, such as mathematical, numerical, and evolutionary algorithms and other hybrid algorithms. In this paper a new algorithm is introduced to design weights or thresholds of artificial neural networks to generate an optimal architecture of ANNs. This algorithm is closer to biological neural networks that use gene schemes, chromosomes, multi-genes, and multi-chromosomes to crossover and mutate each other than any other traditional algorithm. Experiments indicate that this algorithm is very simple and can easily and accurately construct an optimal architecture of ANNs.

The paper is organized as follows. In section 2, a basic GEP theory will be introduced. The algorithm of evolving weights and architectures of ANNs via GEP is discussed in section 3. Then, in section 4 experiments are given to demonstrate the algorithm of solving optimal artificial neural networks via GEP. Finally, in section 5 conclusions are given.

2. Basic GEP Theory

GEP [1, 2] was introduced by Ferreira in 2001. It belongs to the research field of evolutionary computations [6, 9] like a genetic algorithm (GA) and genetic programming (GP) [8], and it is a natural development of GAs and GP. It also simulates the natural biological evolution to generate new individuals of population through crossover and mutation gene codes, i.e., chromosomes of the individuals at random. GEP has been applied to many practical areas, such as solving system regression problems, fitting economic prediction curves, building mathematical models to approximate complex functions, solving parameter identification problems, and solving data mining problems. The goal of this paper is to focus on how to evolve weights or thresholds to build an optimal architecture of artificial neural networks and to perform the corresponding algorithm efficiently.

2.1. GEP structure. In GEP, a genome or chromosome consists of a linear and symbolic string of fixed length composed of one or more genes. Despite their fixed length it will be shown that GEP chromosomes can code expression trees with different sizes and shapes. The structural organization of GEP genes is better understood in terms of open reading frames (ORFs). In biology, an ORF or coding sequence of a gene begins with the "start" codon, continues with the amino acid codons, and ends at a termination codon. However, a gene possesses more than a respective ORF, with sequences upstream from the start codon and sequences downstream from the stop codon.

In addition, genes are structurally organized in a head that consists of arguments including function symbols, variables, and constants, and in a tail that includes only variables or constants. It is this structural and functional organization of GEP genes that always guarantees production of valid programs or expression trees, no matter how much and how profoundly we modify the chromosomes.

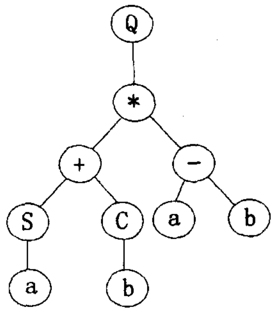

FIGURE 1. Expression tree of function (1).

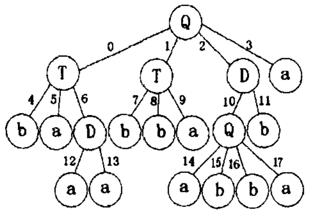

FIGURE 2. Expression tree of ANN.

2.2. GEP encoding and decoding. GEP uses the same kind of diagram representation as GP [5, 7], but the entities (expression trees or parse trees) decoded via GEP are the expression of a genome, which is encoded using GEP rules and is different from GP. These kinds of expression trees decoded via GEP are phenotype representations of genome's genotype of GEP individuals. Through crossover and mutation of genes, chromosomes, multi-genes, and multi-chromosomes, optimal weights of artificial neural networks can be produced, and the optimal architecture of artificial neural networks can be obtained.

2.3. GEP representation. We consider an example of an algebraic expression:

$$\sqrt{(\sin a + \cos b) * (a - b)}. \tag{1}$$

We call function (1) the original problem of GEP, and then convert it into an expression tree as in Fig. 1.

In Fig. 1, "Q", "S", and "C" represent the square root, sin, and cos functions, respectively. We define this kind of diagram as a representation of the phenotype of GEP individuals of function (1). Function (1) is the chromosome function of GEP, and the following gene sequence is the genotype of GEP individuals of function (1) according to the GEP rule:

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \\ Q & * & + & - & S & C & a & b & a & b \end{array} \tag{2}$$

where the first line denotes the order of the arguments of a GEP gene; i.e., there are ten arguments in this gene. The corresponding organization method between the phenotype in Fig. 1 and the genotype of expression (2) will be given below.

We define a decoding method from the genotype like expression (2) into the phenotype as in Fig. 1 with the reading rules of a GEP gene sequence: The genotype of a GEP gene sequence is transferred from the straightforward reading of the expression tree from left to right and from top to bottom.

2.4. Genotype and phenotype of ANNs. In order to illustrate a GEP representation of ANNs, we introduce a logical expression tree as shown in Fig. 2 as a phenotype of a complex ANN with the transferring threshold function from a genotype of ANN, which is a single gene consisting of 12 places of head and 12 places of tail in the same way as follows:

$$\tag{3}\begin{array}{cccccccccccccccccccccccc} 1&2&3&4&5&6&7&8&9&0&1&2&3&4&5&6&7&8&9&0&1&2&3&4\\ Q&T&T&D&a&b&a&D&b&b&a&Q&b&a&a&a&b&b&a&b&b&a&b&b\\ &&&&&\text{h}&\text{e}&\text{a}&\text{d}&&&&&&&&&\text{t}&\text{a}&\text{i}&\text{l}&&& \end{array}$$

where "Q" represents a node with four argument inputs, "T" a node with three argument inputs, "D" a node with two argument inputs, and the a's and b's are external inputs (terminal inputs) of an ANN neuron. In Fig. 2, each number from 0 to 17 represents the subscript of a weight, which has been obtained at random in the interval range and represents the time order of generating random numbers of weights.

3. A New Algorithm

We consider a complex feedforward multi-layer artificial neural network consisting of a set of processing elements [10], also known as neurons or nodes, which are interconnected. It can be described as a directed graph in which each node i performs a transfer function f_i of the form

$$\tag{4} y_i^{m+1} = f_i^{m+1}\left(\sum_{j=1}^{n} w_{ij}^m x_j^m + b_j^m\right), \quad m = 0, 1, \ldots, M-1,\ i = 1, 2, \ldots, n,$$

where M is the number of layers in the network, y_i^{m+1} is the output of node i in the $m+1$th layer, x_j^m is the jth input to the node in the mth layer, w_{ij}^m is the connection weight between nodes i and j in the mth layer, and b_j^m is the threshold (or bias) of the node in the mth layer. Usually, f_i^{m+1} is nonlinear, such as a heaviside, sigmoid, or Gaussian function.

Assume the training set $\{(\mathbf{X}_i, y_i) : i = 1, 2, \ldots, k\}$, where $\mathbf{X}_i$ are the input vectors and y_i are the target outputs (the gene size is 30 with head size 8 and tail size 22).

In order to see the advantages of evolving ANNs in a better way via GEP, we set the fitness function

$$\tag{5} \text{fitness} = \sum_{i=1}^{k}(M - |\hat{y}_i - y_i|),$$

where $M = 100$, y_i are the target outputs, and $\hat{y}_i$ are the computing outputs. If $|\hat{y}_i - y_i| \le \varepsilon$ for some $i = 1, 2, \ldots, k$, change $|\hat{y}_i - y_i|$ to 0. The sizes of M and ε depend on an ANN problem. Thus we can easily see that the larger the fitness value, the better the evolved ANN architecture. Therefore, the algorithm can be described according to the genotype and phenotype representation of ANNs in section 2.4 as follows:

1. Create an initial population of an ANN genotype using the gene expression (3) and the corresponding initial population of weights and thresholds according to the number of weights of each GEP individual at random.
2. Transfer all the genotypes into phenotypes of ANN genes, and calculate all the corresponding fitness values.
3. Select a single point gene (or chromosome) or multi-point gene (or chromosome, multi-gene or multi-chromosome) of two ANN gene parents to cross, and then select two weights of every individual to mutate randomly to generate new individuals.
4. Select a single point gene (or chromosome) or multi-point gene (or chromosome, multi-gene or multi-chromosome) of an ANN gene parent to mutate, and then select a weight of this individual to mutate to produce new individuals.
5. Recalculate the fitness value of the evolved individual if the value is larger than the smallest fitness value of individuals in the population, then replace the bad individual with the new individual, and generate the next population.
6. If the stop criterion is reached, i.e., if the fitness value of the best individual in the population or the iteration number is more than a prescribed number, save the best individual as the best architecture of an ANN. Otherwise, go to step 3.

4. Simulation Experiments

In order to demonstrate the validity of constructing the optimal ANN architecture using GEP, only the exclusive-or function (XOR) is used to perform our experiments in this paper.

The XOR is a simple Boolean function of two activities with values 0 or 1, and when the values of two activities are equal, the output of the final layer is 0; otherwise, the output of the final layer is 1. It is known that this kind of function can be solved easily using linear encoded neural networks. However, if we evolve this neural network using GEP, all the structural and non-structural ANN architectures satisfying the XOR, the inputs, and the final layer outputs can be obtained. This is the main difference between a GEP algorithm and other mathematical algorithms.

In the GEP evolving, we set $T = 100$, the function set is $\{Q, T, D\}$, the population size is 20, the terminal set is $\{a, b\}$, the weights range in the interval $[-2, 2]$, the head length is 8, the tail length is 22, $\varepsilon = 0.01$, and we use a heaviside transfer function to calculate all the outputs. We run the GEP program 30 times consecutively, and then find the perfect solution as the best individual as follows:

$$\begin{array}{cccccccccccccccccccccccccccccc} 1&2&3&4&5&6&7&8&9&0&1&2&3&4&5&6&7&8&9&0&1&2&3&4&5&6&7&8&9&0 \\ Q&T&Q&Q&D&D&b&a&a&a&a&b&b&b&b&a&a&b&a&b&b&a&b&a&a&a&a&b&b&a \end{array} \tag{6}$$

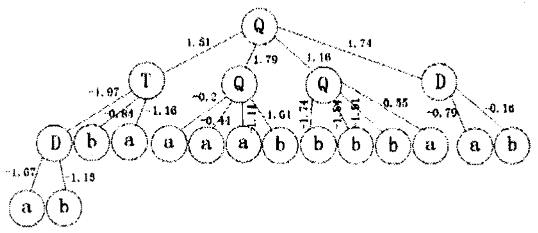
FIGURE 3. Expression tree of the exclusive-or function in an ANN

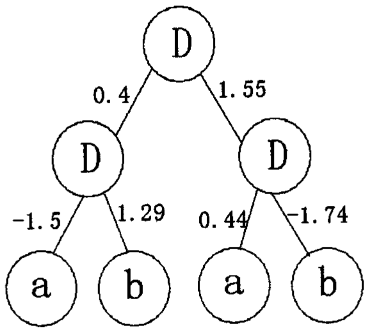

FIGURE 4. {DDDabab}

where

$$\text{Subscripts of Weights} = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17\}$$
$$\text{Random weights} = \{1.51,1.79,1.16,1.74,-1.97,-0.84,-1.16,-0.2,-0.44,$$
$$-1.11,1.61,-1.74,-1.84,-1.91,0.55,-0.79,-0.16,-1.67,-1.15\}.$$

According to the rule of transferring the genotype to the phenotype, expression (6) can be written in the expression tree as in Fig. 3.

In the above experiments, if the function set is changed to $\{D\}$ and other parameters remain the same, then a two-layer artificial neural network can be obtained, a shorter perfect ANN gene solution can be generated, and its expression tree is shown as in Fig. 4. Hence we can easily see from the experiments that the ANN architecture obtained using GEP is not unique, its architecture and shape depend on the function set, the number of layers, the head length the tail length, etc.

5. Conclusions

From the above theoretical and experimental analysis of GEP, we see that the algorithm of solving ANNs optimal architecture via GEP is easier than other algorithms. In addition to the above experiments, some evolving experiments of complex ANNs architectures have been also performed and in general we can fins an optimal solution of ANNs in less than 100 iterations. Compared with GP, the convergence speed and shape of evolving ANNs using GEP are much better; the main reason is that GP's genotype and phenotype are nonlinearly represented and the evolved genes sometimes have syntax mistakes, but GEP has overcome these shortcomings and linearly represents the genotype and phenotype. Furthermore, the evolved genes are legal at any time. Therefore, the GEP evolving algorithm has a wider application range in solving the architectures and weights and thresholds of ANNs. We have used GEP to train neural network ensembles to broaden its applications.

References

[1] C. Ferreira, Gene expression programming: A new adaptive algorithm for solving problems, *Complex Systems* **13** (2001), 87–129.

[2] C. Ferreira, Function finding and the creation of numerical constants in gene expression programming, 7th Online World Conference on Soft Computing in Industrial Applications, September 23-October 4, 2002.

[3] M. T. Hagan, Neural Network Design, China Machine Press, 2002.

[4] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd Edition, China Machine Press, 1998.

[5] K. Li, Z. Chen, Y. Li, and A. Zhou, An application of genetic programming on economic forecasting, Proceedings of the International Conference on High Performance Computing and Applications, Z. Chen *et al.*, eds., Springer-Verlag, Heidelberg, 2005, 71–80.

[6] K. Li, Y. Li, Z. Chen, and Z. Wu, A new dynamic evolutionary algorithm based on particle transportation theory, Proceedings of the International Conference on High Performance Computing and Applications, Z. Chen *et al.*, eds., Springer-Verlag, Heidelberg, 2005, 81–92.

[7] K. Li, Y. Li, C. Teng, and L. Wang, Application of genetic programming on data mining, *Computer Engineering and Applications* **3** (2005).

[8] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed., Springer-Verlag, Berlin, Heidelberg and New York, 1996.

[9] Z. Pan, L. Kang, and Y. Chen, Evolutionary Computation, Tsinghua University Press, 1998.

[10] X. Yao, Evolving artificial neural networks, Proceedings of the IEEE, **87**, 1423-1447, September 1999.

Kangshun Li:
SCHOOL OF INFORMATION ENGINEERING, JIANGXI UNIVERSITY OF SCIENCE & TECHNOLOGY, JIANGXI 341000, CHINA.
E-mail: lks@public1.gzptt.jx.cn

Yuanxiang Li and Haifang Mo:
STATE KEY LABORATORY OF SOFTWARE ENGINEERING, WUHAN UNIVERSITY, WUHAN 430072, CHINA.

Zhangxin Chen:
CENTER FOR SCIENTIFIC COMPUTATION AND DEPARTMENT OF MATHEMATICS, SOUTHERN METHODIST UNIVERSITY, DALLAS, TX 75275-0156 USA.
E-mail: zchen@mail.smu.edu