A Deep Learning Assisted Gene Expression Programming Framework for Symbolic Regression Problems

Jinghui Zhong(✉), Yusen Lin, Chengyu Lu, and Zhixing Huang

Guangdong Provincial Key Lab of Computational Intelligence and Cyberspace Information, School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
jinghuizhong@gmail.com

Abstract. Genetic programming is a powerful evolutionary algorithm that solves user-defined tasks through the evolution of computer programs. Selecting a proper set of function primitives is a fundamental and challenging operation in applying GP to real applications. Traditional manual design methods require a lot of domain knowledge and are not effective and convenient enough. To address this issue, this paper proposed an automatic function primitive identification mechanism. The key idea is to train a deep convolutional neural network to predict the probability of the existence of a function primitive in the target solution. During the evolution of GP, function primitives with higher probabilities are more likely to be selected to construct solutions. The proposed method is tested on nine benchmark problems and the experimental results have demonstrated the efficacy of the proposed method.

Keywords: Genetic Programming (GP) · Deep Learning (DL)
Convolutional Neural Network (CNN)

1 Introduction

Genetic Programming (GP) is a population-based meta-heuristic search algorithm that solves user-defined tasks through the evolution of computer programs [4,6,9]. Due to its good effectiveness in real applications, GP has attracted increasing attention recently. Various enhanced GP variants have been proposed, such as Cartesian Genetic Programming (CGP) [13], Semantic Genetic Programming (SGP) [3,7,14], Grammatical evolution (GE) [15], Gene Expression Programming (GEP) [5], Linear Genetic Programming (LGP) [2]. So far, GP and its variants have been applied to a range of real applications, including time series prediction, classification, combinatorial optimization, knowledge discovery and data mining [1,19–22].

One fundamental operation in applying GP to real applications is to select a proper set of function primitives to efficiently construct solutions (e.g., a mathematic formula). The function primitives (e.g., numerical functions such as power

L. Cheng et al. (Eds.): ICONIP 2018, LNCS 11307, pp. 530–541, 2018.
https://doi.org/10.1007/978-3-030-04239-4_48

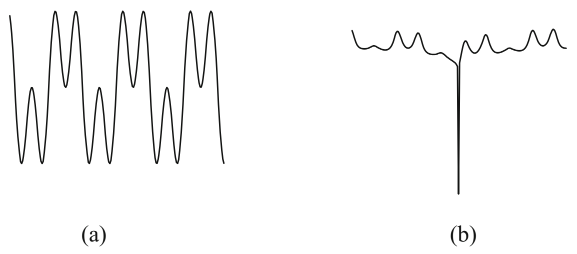

Fig. 1. Examples of time series generated by different functions.

function and logarithmic function) are used to link terminals (e.g., variables x and y) to generate new outputs. Traditionally, function primitives are selected manually by experts in an ad-hoc manner, relying on experts' domain knowledge and experience.

For example, the time series in Fig. 1(a) is very likely to involve trigonometric functions (e.g., sin) because of the periodicity reflected in the curve. However, for a more complicated case as shown in Fig. 1(b), it will become more difficult for experts to guess the functions in the target formula. To ensure high quality solutions can be found in cases similar to Fig. 1(b), the function set should be set large enough. However, this will significantly increase the search space, slow down the search efficiency and make GP get trapped into local optima easily. How to select proper function set to balance the effectiveness and efficiency of GP in real applications is still a challenging problem remained to be explored in the GP community. In this scenario, this paper proposes an enhanced GP (named DL-GEP), which utilizes a deep learning based method to select important functions for efficient solution construction. Deep Learning is a hot machine learning method which shows impressive performance in many applications such as image processing and pattern recognition [8,11]. However, little work has been reported in the literature on applying deep learning to improve GP. This paper makes an early attempt to fill in this gap.

Specifically, in the proposed DL-GEP, a function predictor is built based on a deep convolutional neural network. The goal of the function predictor is to predict the probability of the existence of a function primitive in the target formula. A multi-label classification problem is modelled and a large set of artificial functions are generated to train the deep convolutional neural networks, so that it can automatically identify important function primitives in the original function set. By assigning higher selecting probability to those primitives which are more important, the search efficiency of GP can be improved. Furthermore, a new mechanism is proposed to adaptively adjust the selection probabilities of function primitives based on the promising individuals in the current population. The proposed DL-GEP is tested on nine benchmark problems. The experiment results have demonstrated that the DL-GEP can offer very promising performance.

2 Preliminaries

This section briefly introduces some background knowledge and reviews the related works to help readers comprehend our proposed method.

2.1 Deep Learning

Deep Learning refers to the machine learning methodologies based on deep neural networks (DNNs) and recurrent neural networks, among others [11]. Generally, a neural network has the following components: input layer, hidden layer and output layer. The input layer receives raw input data and the output layer outputs the probability of different labels. A DNN is a neural network containing more than one hidden layer. In recent years, the DNN has been developing rapidly and applied to a wide range of applications such as classification and image processing [8,11].

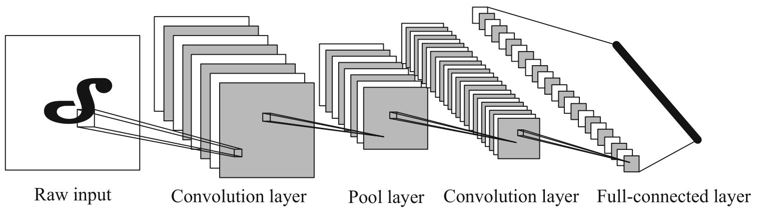

Fig. 2. The schematic of CNN

The Convolutional Neural Network (CNN) is one of the most successful DNNs which is composed of convolution layer, pool layer and fully-connected layer, as shown in Fig. 2. The key idea of CNN is substituting the full connected topology between layers into a convolution layer to reduce the number of weights. The convolution layer maps a group of inputs from last layer into a single output and delivers this output to the activation function. Though the convolution layer filters the input into an output with smaller dimensions, the output still has relatively large dimensions which may make the trained model over-fitting easily. Thus, the pooling layer is introduced to cut down the dimension. The most common pooling layers are maximum pooling and average pooling. The maximum pooling takes the maximum output in a small group of neuron as the single output while the average pooling takes the average of outputs from the group of neurons. The introduction of pool layer guarantees the next layer to be insensitive to the small fluctuation from the previous layers. The full-connected layer is adopted to generate the final outputs. One promising characteristic of CNN is the sharing of weights, which can reduce the number of parameters, the complexity of the network, and can accelerate the training efficiency.

2.2 Deep Learning Meets GP

In the literature, various Evolutionary Algorithms (EAs) such as GP have been proposed to optimize the parameters and structures of Artificial Neural Networks (ANNs) [18]. For example, Maryam proposed to use Cartesian Genetic Programming (CGP) to evolve the structure of ANN [12]. Andrew developed a recurrent CGP (RCGP) to describe the topology of recurrent networks, which has been shown to be very effective in optimizing the structure of the recurrent ANN [17]. Recently, Lamos-Sweeney et al. [10] have proposed a GA-based method to optimize the structure of a DNN, using a similar idea of the Restricted Boltzmann Machine (RBM). The model uses GA to optimize the weights of the layer so that the outputs of each layer are as many as possible to be similar with the inputs. Although various efforts have been made in applying EAs to improve deep learning, little work has been reported on applying deep learning to improve EAs. This paper makes an early attempt to utilize deep learning to automatically identify important function primitives for GP, which can improve the search efficacy of GP.

3 Proposed Method

3.1 The General Framework

Figure 3 illustrates the general structure of the proposed framework, which consists of two modules: the CNN training module and the GP training module. The CNN training module is used to train a CNN to predict the weights of functions in the original function set. Functions with larger weight are more likely to be a component in the target solution. To train the CNN, a set of artificial functions with random function primitives is generated as training data. The training functions should be many enough to achieve high prediction accuracy. Once the well-tuned CNN model is obtained, it then can be used to predict the weights of function primitives for new regression problems.

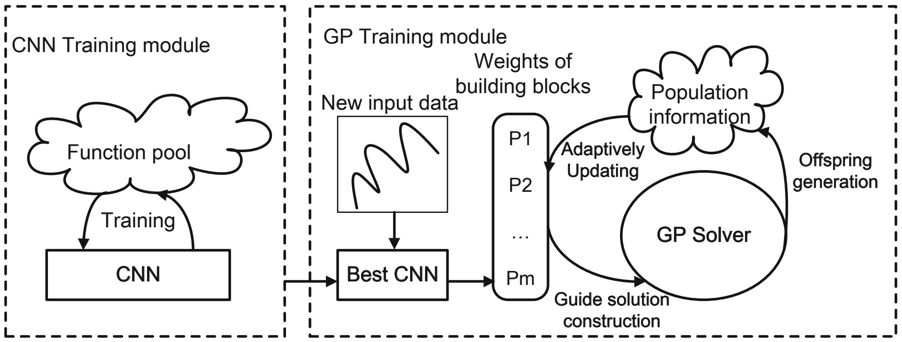

Fig. 3. The general structure of the proposed framework.

Meanwhile, the GP training model is used to learn an optimal formula to fit the given data. In the GP training module, a GP solver selects function primitives in the given function set to construct solutions based on the weights predicted by the well-tuned CNN. Functions with larger weights are more likely to be selected to construct solutions. Further, to improve the search efficiency, the weight vectors are updated adaptively by considering the frequencies of the function primitives in the current population. Suppose $\mathbf{P}_g$ is the current weight vector, and $\mathbf{Q}_g$ is the frequency of function primitives in the survival individuals, the weight vector is updated by

$$\mathbf{P}_{g+1} = \lambda\mathbf{P}_g + (1-\lambda)\mathbf{Q}_g \tag{1}$$

where λ is the update rate. Usually, $\lambda \in [0,1]$ is set to a relatively large value (e.g., $\lambda = 0.9$).

3.2 The CNN Training Module

The CNN Training Module is used to generate a weight vector to describe the importance of function primitives. Specifically, the inputs of the CNN training module are images of different categories. The images are the curves of function generated by randomly linking elementary functions such as sin, cos, exp and log. In this study, a GP variant is adopted to generate a large number of random solutions. Each solution represents a random function, and we can generate an image for the function by plotting the curve of the function. The labels of each random solution can be set automatically based on the valid elements in the solution. Noting that each image can have multiple labels if it contains multiple elementary function primitives. For example, a function $exp(sin(x))$ has two labels: one for exp and one for sin. Based on the above steps, we can convert the weight prediction problem to a multi-label classification problem.

Inception-v3 [16] is an excellent CNN-based tool to solve muti-label classification problem. Inception-v3 is applied to solve a larger variety of computer vision tasks and achieves great success by utilizing deeper and wider networks. It can recognize multi-label images and output each probability of the given categories. Besides, it has already been trained and published in the Internet[1]. In this study, we choose Inception-v3 to solve the multi-label classification problem. To apply Inception-v3 to the multi-label classification problem, the only task is to set up the training images and the responding labels as input for training a new top layer that can recognize other classes of images.

3.3 The GP Training Module

In this study, the SL-GP proposed in [22] is adopted as the GP solver in the GP training module. In SL-GEP, each chromosome is fixed-length and contains a

[1] The Inception-v3 can be downloaded from http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz.

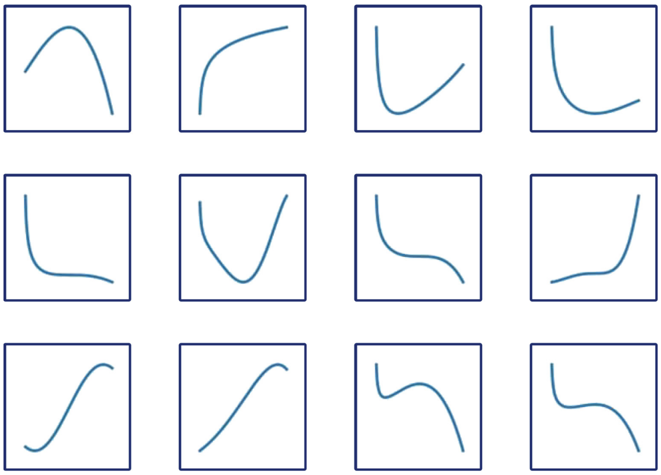

Fig. 4. Example images for training the CNN.

main program part and an automatically defined function (ADF) part. The main program part generates the final output, while the ADF part contains a number of subfunctions which are used as building blocks in the main program part. The main program and ADF parts can be respectively decoded to expression trees by using the breadth-first traversal scheme [6]. Suppose a typical chromosome is given as:

$$[G, exp, G, sin, y, x, x, y, y, +, sin, -, b, b, a, a] \tag{2}$$

where G is an ADF (i.e., sub-function), x and y are terminals, and a and b are the input arguments of ADFs. Suppose that the length of the main program and the ADF parts are 9 and 7 respectively. As illustrated in Fig. 5, the given chromosome can be decoded as:

$$\begin{aligned} \Gamma &= G(exp(sin(x)), G(y, x)) \\ &= sin(sin(b) + (b - a) + sin(b) + (b - a) - exp(sin(x)) \end{aligned} \tag{3}$$

Based on the above chromosome representation, the proposed DL-GEP performs the following steps to find the optimal solutions.

Step 1 – Initialization. In this step, a random population of chromosomes are generated. Each chromosome is a vector of symbols, i.e.,

$$X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}] \tag{4}$$

where n is the length of chromosomes. To set the value of $x_{i,j}$, a primitive type is randomly selected among {function, sub-function, terminal, input argument}. Then a random value of the selected feasible type is assigned to $x_{i,j}$. If the

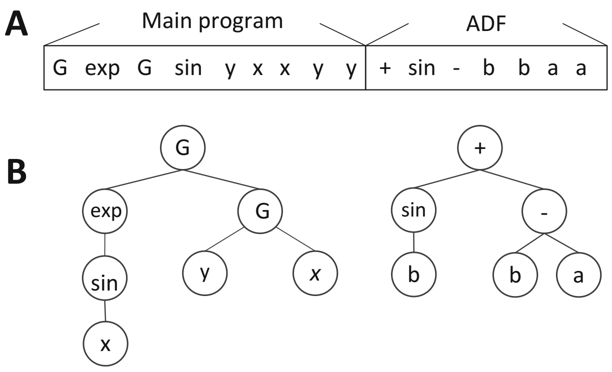

Fig. 5. The decoded expression trees of the example chromosome in (3).

primitive type is function, then the value is selected using a route-wheel selection strategy based on the weights **P** generated by the best CNN. Functions with larger weights have larger selection probabilities.

Step 2 – Reproduction. In this step, the selection probabilities of functions are updated at first using Eq. (1). Then, the genetic operators proposed in SL-GEP [22] (i.e., the mutation, crossover and selection) are performed iteratively to update the population. Specifically, In the mutation, the extended "DE\current-to-best\1" mutation operation is performed on each target vector to generate a mutant vector:

$$Y_i = X_i + F \cdot (X_{best} - X_i) + \beta \cdot (X_{r1} - X_{r2}). \tag{5}$$

where X_{best} is the best-so-far solution and X_{r1} and X_{r2} are two random individuals. In [22], the numerical operators in Eq. (5) (i.e., $\cdot$, $+$ and $-$) are redefined to evolve solutions in a discrete search space. As shown in Fig. 5, each solution is represented by tree structures that comprise of two types of nodes: function and terminal. A function node has one or multiple children (e.g., $+$ and sin), while a terminal node is a leaf node without any child (e.g., variables and constants). Based on the redefined operators, the result of Eq. (5) is equivalent to mutating element of Y_i with a probability vector. The reader is referred to [22] for the implementation details of the mutation operation of SL-GEP.

After the mutation operation, the crossover generates a trial vector for each target vector:

$$u_{i,j} = \begin{cases} y_{i,j}, \text{if} \quad rand(0,1) < CR \quad \text{or} \quad j = k \\ x_{i,j}, \text{otherwise} \end{cases} \tag{6}$$

where U_i is the trial vector, CR is the crossover rate, k is a random integer within $[1, n]$, and $x_{i,j}$, $y_{i,j}$ and $u_{i,j}$ are the jth variables of X_i, Y_i and U_i respectively.

The selection performed following the crossover, the selection operator is performed to select the better one among each pair of target and trial vectors:

$$X_i = \begin{cases} U_i, \text{if} \quad f(U_i) \leq f(X_i) \\ X_i, \text{otherwise} \end{cases} \tag{7}$$

where $f(X)$ is the fitness evaluation function.

The reproduction step is performed repetitively until finding the optimal solution or the termination condition is met (e.g., the maximum number of generations is reached).

4 Experiment Studies

4.1 Results of CNN Training Module

In this subsection, we evaluate the performance of the CNN training module. First of all, we create 100,000 training images by generating 100,000 random functions (i.e., chromosomes), which are composed of seven elementary functions: $\{+, -, *, sin, cos, exp, log\}$. We also select 20,000 testing images from the training images. We focused on predicting the weights of the last four functions (i.e., *sin*, *cos*, *exp*, *log*). That is, each image is assigned a weight vector with four dimension, i.e., $P = [p_1, p_2, p_3, p_4]$. If log is in a function, p_4 of the corresponding image will set to 1. Otherwise, p_4 is set to 0. Since the Inception-v3 has been trained except the top layer, the only task is to place the function images and the corresponding labels. This experiment iterates for 10,000 times. The final training accuracy is 87.0% and the final test accuracy is 88.1%. The testing results indicate that the trained CNN is able to offer promising prediction accuracy.

4.2 Results of GP Training Module

In this subsection, the GP training module is applied to solve nine benchmark problems, whose target formula are listed in Table 1. For all problems, the function set of all compared algorithms is set to be $\{+, -, \times, \div, sin, cos, exp, ln(|x|)\}$.

First, we investigate the effectiveness of the CNN training module. Table 2 lists the weights of the functions for each problem. It can be observed that the prediction results are correct to some extend. Taking F_5 for example, the CNN classifies F_5 to two classes, *sin* and *cos* and successfully removes two useless functions, *exp* and *log*. Similar results can be found on other problems such as F_2, F_3, F_7, and F_8. However, the CNN classifies F_9 into wrong classes. This may be because the given regression data is not sufficient enough to capture the landscape feature of F_9.

Next, we compare the proposed DL-GEP with SL-GEP to demonstrate its effectiveness. The common settings of DL-GEP and SL-GEP are set the same as [22]. The DL-GEP contains one additional parameter λ, which is set to 0.95. As in [22], the 10-fold cross validation approach is adopted to test the effectiveness of the compared algorithms. Each algorithm will terminate when the number

Table 1. Nine symbolic regression benchmarks for testing.

P	Objective function	Data set
F_1	$exp(sin(x))$	$U[-1, 1, 200]$
F_2	$cos(x)ln(x)$	$U[-1, 1, 200]$
F_3	$sin(x) + ln(x) - x$	$U[-1, 1, 200]$
F_4	$exp(x)sin(x^2)$	$U[-1, 1, 200]$
F_5	$cos(x^2)sin(x)$	$U[-1, 1, 200]$
F_6	$cos(x) + cos(x^2)$	$U[-1, 1, 200]$
F_7	$sin(x^2)cos(x) - 1$	$U[-1, 1, 200]$
F_8	$sin(x) + sin(x + x^2)$	$U[-1, 1, 200]$
F_9	$ln(x + 1) + ln(x^2 + 1)$	$U[0, 2, 200]$

$U[a, b, c]$ represents c uniform random samples from a to b.

Table 2. Predicted weights of the elementary functions on the nine problems.

P	*sin*	cos	exp	log
F_1	0.84	0.72	0.67	0.14
F_2	0.71	0.77	0.46	0.74
F_3	0.72	0.67	0.41	0.79
F_4	0.86	0.82	0.53	0.31
F_5	0.90	0.94	0.48	0.12
F_6	0.50	0.84	0.42	0.24
F_7	0.81	0.72	0.40	0.23
F_8	0.94	0.87	0.58	0.22
F_9	0.86	0.87	0.36	0.14

of evaluations reaches a maximum of 1,000,000. When an algorithm converges to a solution with fitting error smaller than 10^{-4}, a successful search convergence or perfect hit is achieved. In the empirical studies, three performance metrics adopted in [22] are used for comparison analysis. The first is the RMSE which is the average testing accuracy of the 100 independent runs based on the 10-fold cross validation. The second is the success rate of achieving perfect hits (denoted as Suc), which is calculated by:

$$Suc = \frac{C_s}{C} \cdot 100\% \tag{8}$$

where C is the number of independent runs and C_s is the number of successful runs achieving a perfect hit. The third metric is the number of fitness evaluations required to achieve a perfect hit (denoted as Run Time, *RT*). This metric measures the convergence speed of an algorithm. For each problem, a Wilcoxon signed-rank test detects significant differences on the RMSE and the RT.

Table 3. Comparsion results of SL-GEP and DL-GEP.

Problem	SL-GEP			DL-GEP		
	Suc	RMSE	RT	Suc	RMSE	RT
F_1	100	0 $\approx$	549 $-$	100	0	383
F_2	1	0.1240 $+$	992,197 $-$	5	0.1257	977,152
F_3	40	0.0263 $-$	784,942 $-$	53	0.0177	704,517
F_4	100	0 $\approx$	12,479 $\approx$	100	0	10,526
F_5	100	0 $\approx$	33,444 $-$	100	0	15,065
F_6	100	0 $\approx$	10,816 $-$	100	0	4,530
F_7	87	0.0004 $-$	443,884 $-$	95	1.76E-04	338,833
F_8	100	0 $\approx$	14,582 $+$	100	0	17,066
F_9	22	0.0074 $+$	855,736 $+$	21	0.0076	944,694

Symbol $-$, $\approx$, $+$ represents SL-GEP is relatively significantly worse than, similar to and better than DL-GEP according to Wilcoxon signed-rank test at $\alpha = 0.05$.

Table 3 shows the comparison results of DL-GEP and SL-GEP on the nine problems. It can be observed that DL-GEP reported higher *Suc* on F_2, F_3 and F_7 than SL-GEP. As for *RT*, DL-GEP performs better than SL-GEP on six out of the nine problems,i.e., F_1, F_2, F_3, F_5, F_6 and F_7. The above results indicate that by using the selection probabilities predicted by the CNN training model, the DL-GEP can converge faster than SL-GEP. To further validate this, we take F_7 for example and plot the evolution curve of the best fitness offered by DL-GEP and SL-GEP on this problem. The evolution curve in Fig. 6 shows that the DL-GEP can find better solution at the beginning, and DL-GEP can converges faster

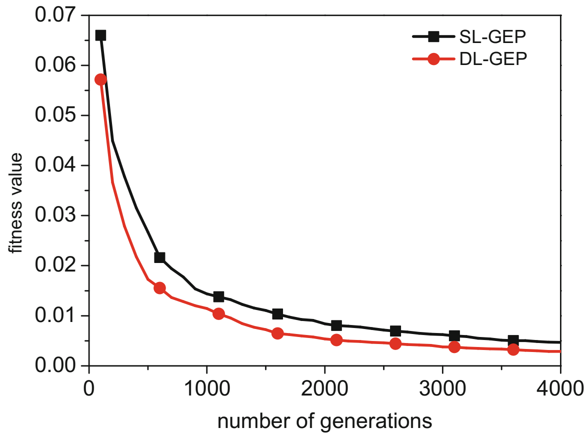

Fig. 6. Evolution of the best fitness values derived from SL-GEP and DL-GEP on F_7

during the evolution process. The above results demonstrate that the proposed deep learning based mechanism is effective to improve the search efficiency and accuracy.

5 Conclusions

In this paper, we proposed a deep learning assisted GP framework for symbolic regression. The key idea is to train a deep convolutional neural network to predict the weights of function primitives for solution construction. The predicted weights are then used to guide the solution construction of GP during the evolution process. The experiments on nine benchmark problems show that the proposed framework is effective to improve the search efficiency.

In this study, the CNN training module are trained using images of 2D function curves. Thus, the proposed framework can only work on one-dimensional symbolic regression problem (e.g., time series prediction problem). As for future work, we plan to extend the proposed method for complicated symbolic regression problems with more input variables. The second future research work is to apply the proposed framework to solve real problems. In addition, developing new machine learning techniques to assigned weights to both terminals and functions is another promising research direction.

Acknowledgments. This work was supported by the National Natural Science Foundation of China (Grant No. 61602181), the Fundamental Research Funds for the Central Universities (Grant No. 2017ZD053), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X183), and the Guangzhou Science and Technology Plan Project (Grant No. 201804010245).

References

1. Berry, M.J., Linoff, G.S.: Data Mining Techniques. Wiley, Hoboken (2009)
2. Brameier, M.F., Banzhaf, W.: Linear Genetic Programming. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-31030-5
3. Castelli, M., Vanneschi, L., Silva, S.: Semantic search-based genetic programming and the effect of intron deletion. IEEE Trans. Cybern. **404**(1), 103–113 (2014). https://doi.org/10.1109/TSMCC.2013.2247754
4. Cramer, N.L.: A representation for the adaptive generation of simple sequential programs. In: Proceedings of the 1st International Conference on Genetic Algorithms, pp. 183–187. L. Erlbaum Associates Inc., Hillsdale (1985)
5. Ferreira, C.: Gene expression programming: a new adaptive algorithm for solving problems. Complex Syst. **13**(2), 8–129 (2001)
6. Ferreira, C.: Gene Expression Programming. Springer, Berlin (2006). https://doi.org/10.1007/3-540-32849-1
7. Ffrancon, R., Schoenauer, M.: Memetic semantic genetic programming. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp. 1023–1030. ACM (2015)

8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
9. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. MIT press, Cambridge (1992)
10. Lamos-Sweeney, J.D.: Deep learning using genetic algorithms. Dissertations and Theses - Gradworks (2012)
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436 (2015)
12. Mahsal Khan, M., Khan, G.M., Miller, J.: Evolution of optimal ANNs for non-linear control problems using cartesian genetic programming, vol. 1, pp. 339–346 (2010)
13. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46239-2_9
14. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012. LNCS, vol. 7491, pp. 21–31. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32937-1_3
15. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Trans. Evol. Comput. **5**(4), 349–358 (2001)
16. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)
17. Turner, A.J., Miller, J.F.: Recurrent cartesian genetic programming of artificial neural networks. Genet. Program. Evolvable Mach. **18**(2), 185–212 (2017)
18. Yao, X.: Evolving artificial neural networks. Proc. IEEE **87**(9), 1423–1447 (1999)
19. Zhong, J., Feng, L., Cai, W., Ong, Y.: Multifactorial genetic programming for symbolic regression problems. IEEE Trans. Syst. Man Cybern. Syst. (2018, in press). https://doi.org/10.1109/TSMC.2018.2853719
20. Zhong, J., Cai, W., Lees, M., Luo, L.: Automatic model construction for the behavior of human crowds. Appl. Soft Comput. **56**, 368–378 (2017)
21. Zhong, J., Feng, L., Ong, Y.S.: Gene expression programming: a survey. IEEE Comput. Intell. Mag. **12**(3), 54–72 (2017)
22. Zhong, J., Ong, Y.S., Cai, W.: Self-learning gene expression programming. IEEE Trans. Evol. Comput. **20**(1), 65–80 (2016)