TWO-LAYER GENETIC PROGRAMMING

J. Merta∗, T. Brandejský

Abstract: This paper focuses on a two-layer approach to genetic programming algorithm and the improvement of the training process using ensemble learning. Inspired by the performance leap of deep neural networks, the idea of a multi-layered approach to genetic programming is proposed to start with two-layered genetic programming. The goal of the paper was to design and implement a two-layer genetic programming algorithm, test its behaviour in the context of symbolic regression on several basic test cases, to reveal the potential to improve the learning process of genetic programming and increase the accuracy of the resulting models. The algorithm works in two layers. In the first layer, it searches for appropriate sub-models describing each segment of the data. In the second layer, it searches for the final model as a non-linear combination of these sub-models. Two-layer genetic programming coupled with ensemble learning techniques on the experiments performed showed the potential for improving the performance of genetic programming.

Key words: two-layer genetic programming, ensemble learning, deep learning, bootstrapping, symbolic regression

Received: May 10, 2022
DOI: 10.14311/NNW.2022.32.013
Revised and accepted: August 30, 2022

1. Introduction

This paper focuses on a two-layer approach to genetic programming and improving the training process using approaches known from ensemble learning. Inspired by the performance leap of deep neural networks, the idea of a multi-layered approach to genetic programming is proposed to start with two-layered genetic programming. Another inspiration was the success of ensemble learning methods in optimizing machine learning algorithms. Combining multiple sub-models of data into a more accurate final model also corresponds to the principles of deep learning, which attempts to learn more complex functions by combining simpler functions. In the first layer, genetic programming is used to generate sub-models from a base set of terminals and functions. In the second layer, the final model is composed from the sub-models created in the first phase.

The goal of this paper was to design and implement a two-layer genetic programming algorithm, test its behaviour in the context of symbolic regression on
several basic test cases, and find appropriate settings that have the potential to
make the genetic programming learning process more efficient and increase the ac-
curacy of the resulting models. The paper does not aim to describe the dynamics of
two-layer (or possibly multi-layer) genetic programming in general detail, and does
not seek general relationships between the settings of the different parameters.

2. Background

2.1 Genetic programming

Genetic programming [1] uses evolution to evolve computer programs. It is an
extension of genetic algorithms [2]. The genetic algorithm (GA) was introduced
by John Holland in 1975 [3]. It is a stochastic optimization method based on a
population strategy. Genetic programming allows high-level automatic solution of
given problems without explicit writing of the program by the programmer [2],
when it is sufficient to define the expected behaviour and basic program elements
from which the resulting solution should be composed. The expected behaviour of
the desired program is defined using a fitness function. The GP is provided with
basic program elements from which it gradually builds more complex programs.
The search space of programs is enormously large and complex. A single prob-
lem is generally matched by many different programs [4]. Individuals in genetic
programming represent individual programs. They are represented by non-linear
hierarchical chromosomes of variable length in the form of syntactic trees. Syntac-
tic trees consist of two types of genes: terminals and non-terminals (collectively,
primitives) [5].

The mean length of programmes increases during evolution. According to [5],
this is not a problem caused by crossover but by selection. While crossover makes
some programs longer, it makes other programs shorter. Longer programs have
better rankings compared to short programs in the initial steps of evolution, which
gives longer programs a selection advantage. During evolution, the amount of code
that has no positive benefit in a program often grows. This phenomenon is called
bloat. This can be countered by controlling the maximum size of the trees. Another
effective strategy appears to be tree pruning [6], where large trees are randomly
pruned.

2.2 Symbolic regression

Symbolic regression is a supervised machine learning method for finding symbolic
descriptions of mathematical models that approximate a finite set of data points [5].
The goal is to find the relationship between input and output variables. The result
of symbolic regression is a mathematical equation composed of a set of allowed
arithmetic expressions, functions, variables and constants. In addition to the pa-
rameters of the mathematical model, symbolic regression also searches for its op-
timal structure (the best combination of operations, functions and parameters).
Symbolic regression can be solved using GP but also using other methods. Fig. 1
shows the tree representation of the equation $y = x + 2\sin x$.  

216
2.3 Deep learning and ensemble learning

Deep learning algorithms process the raw input data and try to learn its correct representation [7]. The main idea of deep learning is a compositional approach, where more complex higher-level representations of the problem being solved are formed from its simpler lower-level representations. More complex concepts are built from simpler lower-level concepts. Mathematically, this principle can be described using successive functions. The first level functions have as input variables. Higher level functions are built by composing simpler functions so that the output of the lower level functions is the input of the higher level functions. Each function always expresses a new representation of the original raw input. Models built using ensemble learning methods follow a similar idea.

The main idea of ensemble learning is inspired by human decision making based on the opinions of multiple experts or voting. Ensemble learning is a group of statistical and machine learning methods that combine the final model using multiple sub-models to refine model prediction, improve generalization ability, and reduce overfitting [8]. The combination of sub-models results in a so-called ensemble. By combining multiple sub-models trained over different subsets of the original training data, it is possible to get close enough to the optimal model, avoid overfitting, and improve generalization [8,9]. Proper combination of multiple sub-models also improves the expressive power of the algorithms [8]. The predictions of the sub-models must not be correlated because they would not add new information to the decision process. The overall accuracy of the final models depends on the accuracy of the sub-models as well as the diversity of their outputs [8].

Non-generative ensemble methods only select or combine sub-models but do not create them. Generative ensemble methods create sub-models using a machine learning algorithm and diversify the training data set to create diversified sub-models and increase the accuracy and generalization of the ensemble model. Resampling based methods use bootstrapping technique to diversify the training data [8]. The bootstrapping technique generates several different subsets of training data from the training data set using random selection. Then, by applying a learning algorithm on each generated subsets, sub-models are created and finally
Neural Network World 4/2022, 215–231

2.3.1 Ensemble architecture

The ensemble architecture deals with the creation and selection of sub-models, the topology of their interconnection, and the design of a fusion module that combines the outputs of the sub-models [9]. The design of the fusion module, called fuser, deals with finding the combination of individual sub-models into the final output of the ensemble model. There are two basic variants of the fuser. In the first variant, the fuser combines only the outputs of the sub-models. In the second variant, the merger combines the outputs of the sub-models with the values of the input variables [9]. Fusion modules can also be trainable (e.g., in the stacking method) [9]. A trainable fusion model is represented by a model over sub-models. This metamodel is not trained on exactly the same data as the sub-models, because the ensemble model may be overtrained.

2.4 Optimization of genetic programming and symbolic regression

Genetic programming is a relatively young method that, with increasing computational power, is gradually finding practical applications. Genetic programming as a machine learning method still has gaps in the accuracy of the generated models and various improvements are currently being proposed to help with this problem. Each improvement in the efficiency of genetic programming brings the method closer to real-world use. The main ways to optimize GP include: reducing the fitness evaluation time and improving the convergence of GP. The fitness function evaluation can be reduced by reducing the size of the trees [5], reducing the training set or introducing caching. Using a smaller training set can speed up the fitness function evaluation. This approach can avoid overfitting the model and improve its generalization. The risk is that only part of the desired function is learned [5].

Another approach is to hybridize the GP with an optimization method. The goal of the hybrid approach is to improve the performance of genetic programming using some local learning methods and heuristics. Most of the methods are based on the idea that genetic programming can generate a tree with the correct terminal structure during evolution whose performance will be negatively affected only by the wrong combination of numerical constants in the terminals. One approach is constant optimization, in which the optimal combination of numerical constants is sought for a tree with a given structure of non-terminals using other optimization methods. Evett and Fernandez [10] proposed a simple mutation to tune the constants. Raidl and Guenther in [11] introduced HGP (hybrid genetic programming), added weights to the top-level tree members and optimized them using a robust least squares method. For example, gradient descent [12, 13], simulated annealing combined with the simplex method [14], particle swarm optimization (PSO) [15], multiple regression in the STROGANOFF method [16, 17], evolutionary strategies [13, 18, 19], genetic algorithms [13], self-organizing migrating algorithm (SOMA) [13, 20], the Bison Seeker algorithm [21], and non-linear optimization using
the Levenberg-Marquardt algorithm \cite{22,23} can be used to optimize the constants. There are many modern approaches for GP optimization. Keijzer in \cite{24} proposed an improvement of symbolic regression using interval arithmetic and linear scaling. Linear scaling deals with minimizing the error by shifting the function and stretching it, resulting in a function with the correct shape during selection and is found faster. Multistage genetic programming (MSGP) \cite{25} sequentially builds models for each feature separately using GP and then builds a model of how each feature interacts with each other. Multiple basis function genetic programming (MBF-GP) \cite{26} is very similar to classical GP. The only difference is that each individual consists of multiple trees. Genetic operators are used by MBF-GP at two levels \cite{22}. At the microscopic level, the original crossover between the trees of two randomly selected macroscopic individuals takes place. Macroscopic crossover, on the other hand, exchanges entire trees between two macroscopic individuals. Multi-Gene GP (MGGP) \cite{27} extends conventional GP and increases its accuracy \cite{27}. Similar to MBF-GP, the chromosome consists of multiple sub-models in the form of short-length classical GP trees. Each such tree represents one gene. In contrast to MBF-GP, the trees are not formed sequentially but in parallel. The resulting model is a linear combination of the individual sub-models. Other methods include sequential symbolic regression (SSR) \cite{28}, multiple regression genetic programming (MRGP) \cite{29}, evolutionary feature synthesis (EFS) \cite{30}, geometric semantic genetic programming (GSGP) \cite{31}, FFX \cite{32,33}, elite basis regression (EBR) \cite{34}, etc.

3. Two-layer genetic programming

Inspired by success ensemble learning methods in the area of machine learning optimization authors proposed two-layer genetic programming is divided into two layers. Combination with ensemble learning methods should help GP to exploit search space of programmes and increase its expressive abilities. It can be thought of as two genetic programming algorithms, where the second genetic programming algorithm follows the first one. In the first layer, genetic programming is used to generate sub-models from a base set of terminals and functions. In the second layer, the final model (meta-model) is composed from the sub-models created in the first phase. The whole two-layer GP algorithm can be imagined as the creation of basic building blocks and the subsequent search for their appropriate combination.

The whole two-layer GP (GP-2L) algorithm can be described in several steps:

1. Creating sub-models using separate runs of the plain GP.
2. Adding the sub-models to the set of terminals for the second layer of the two-layer GP.
3. One run of the second layer using the classical GP with sub-models.
4. Return the result from the second layer.

The number of runs of the first layer depends on the number of sub-models generated. The second layer has separate parameter settings. The resulting model is the product of the second layer run.
3.1 Motivation

The design of the two-layer GP algorithm is based on several different motivations:

1. The desire to avoid bloat and the creation of increasing trees without improving the fitness function (reducing model error) by restarting GP, in which the currently created trees will be used as building blocks for a new run of GP, where the “old” code will be used in a new constructive way.

2. The ability to change GP settings during evolution, as different tree (model) complexities may require different operations and settings.

3. Create more robust final models using ensemble learning techniques.

The ensemble nature of the proposed two-layer GP algorithm is based on a parallel topology of sub-models from the first layer with a trainable fuser on the second layer. The individual sub-models are generated by separate runs of the first GP layer.

3.2 Characteristics of layers

The two layers can have different settings of GP parameters and hyperparameters, including the number of generations, population size, set of terminals, and set of functions. The two layers can also be independently optimized using BSA on integer constants.

3.2.1 First layer

For the creation of the sub-models at the first layer, bootstrapping method was chosen to diversify the training dataset, which splits the whole dataset into different subsets for each sub-model. The algorithm can also be run without creating diversified subsets of the training data.

In the bootstrapping method, the size of the training set for the sub-models (bootstrap size) was chosen and then random points were added from the whole training data set. The actual implementation of this method takes as a parameter the size of the training set as a percentage of the size of the original full training dataset, for example the set size would be 30% of the full training dataset. In the implementation of the bootstrapping method used, it was possible for a single data point to occur multiple times in the training set for a sub-model.

3.2.2 Second layer

Second layer uses all sub-models from first layer without no exception. Depending on the choice of functions in the second stage, it can be: simple linear folding or non-linear folding of sub-models. If the sub-models are added (or subtracted), multiplied by a constant, or folded by arithmetic or weighted averages, this is linear sub-model folding. Sub-models may also be folded non-linearly in the second stage if functions such as multiplication, inversion or division are used. Scheme of two-layer genetic programming is visualized in the Fig. 2.
3.2.3 Implementation

The two-layer genetic programming algorithm was implemented in Java on top of the Watchmaker framework. It was necessary to implement the genetic programming module and then its two-layer GP variant in the framework. The standard Random class from the java.util package was used to generate random numbers.

4. Experiments

All experiments used only integers as constants. The following parameters (Table I and Table II) were the same for all experiments on all data sets.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elitism</td>
<td>1</td>
</tr>
<tr>
<td>Probability of mutation</td>
<td>0.01</td>
</tr>
<tr>
<td>Initiation method</td>
<td>growth method</td>
</tr>
<tr>
<td>Initialization tree depth</td>
<td>3</td>
</tr>
<tr>
<td>Target fitness of plain GP</td>
<td>0.0005</td>
</tr>
<tr>
<td>Tree length for pruning</td>
<td>65</td>
</tr>
<tr>
<td>Selection method</td>
<td>deterministic tournament selection</td>
</tr>
<tr>
<td>Tournament size</td>
<td>2</td>
</tr>
</tbody>
</table>

**Tab. I** Tree representation of the equation \( y = x + 2 \sin x \).

The remaining parameters are described for specific experiments. All experiments conducted in this chapter were performed in 100 repetitions. The root mean square error of the model set was used as fitness to evaluate the quality of the final models.

4.1 Data sets

In the experiments, symbolic regression problems were performed on several datasets (the values of the tested functions were generated equidistantly):
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elitism</td>
<td>1</td>
</tr>
<tr>
<td>Probability of a layer 1 mutation</td>
<td>0.01</td>
</tr>
<tr>
<td>Probability of 2nd layer mutation</td>
<td>0.1</td>
</tr>
<tr>
<td>Initialization method</td>
<td>growth method</td>
</tr>
<tr>
<td>Initialization tree depth</td>
<td>3</td>
</tr>
<tr>
<td>Target fitness of layer 1</td>
<td>0.005</td>
</tr>
<tr>
<td>Target fitness of layer 2</td>
<td>0.0</td>
</tr>
<tr>
<td>Tree length for pruning</td>
<td>65</td>
</tr>
<tr>
<td>Selection method</td>
<td>deterministic tournament selection</td>
</tr>
<tr>
<td>Tournament size</td>
<td>2</td>
</tr>
</tbody>
</table>

**Tab. II** *Scheme of two-layer genetic programming.*

- hyperbola according to Eq. (1), 60 points with \( x \in (0.1, 14.85) \),
- a function composed of three different sine functions (2), 120 points with \( x \in (-4, 2) \),
- a dataset (first 399 points) of measured temperatures from the Indian city of Delhi\(^1\).

\[
y_1 = \frac{1}{x}, \quad (1)
\]

\[
y_3 = \sin (x) + \sin (2x) + \sin (3x + 5). \quad (2)
\]

### 4.2 Hyperbola

The experimental setup for the hyperbola function is described in Tab. III and Tab. IV.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>50</td>
</tr>
<tr>
<td>Generation limit</td>
<td>230</td>
</tr>
<tr>
<td>Set of terminals</td>
<td>(x; -1.0; 1.0; 2.0; 3.0)</td>
</tr>
<tr>
<td>Set of non-terminals</td>
<td>*; +; -; sqrt; sin; pow2; pow3</td>
</tr>
</tbody>
</table>

**Tab. III** *Invariant two-layer GP setup.*

Three different bootstrap sizes were tested for the bootstrapping method: 10\%, 20\% and 30\%.

4.3 Compound sinus

The parameter settings for the composite sine of Eq. (2) were the same as for the simple sine, except that the number of individuals was set to 150 (on both layers). After that experiments with more generations and individuals were performed.

4.4 Temperatures of Delhi

In a recent series of experiments, a real dataset capturing temperatures in the Indian city of Delhi was tested. The feature set contained the functions *; +; −; %; inv; sqrt; pow2 on the first layer and *; +; −; avg on the second layer. The generation limit was set to 530 for plain GP and 500+30 for 2L-GP. Basic experiments compared GP and 2L-GP with 50 individuals on both layers with 250 individuals on both layers.

5. Results

5.1 Hyperbola

Three different bootstrap sizes were tested for the bootstrapping method:

1. plain GP,
2. plain GP, 1500 individuals,
3. GP-2L without diversification,
4. GP-2L with bootstrapping (10%),
5. GP-2L with bootstrapping (30%),
6. GP-2L with bootstrapping (60%).
Population size (2nd layer) 50
Generation limit (2nd layer) 30
Base set of terminals (2nd layer) x; –1.0; 1.0; 2.0; 3.0
Set of non-terminals (2nd layer) *; +; –; sqrt, sin, pow2, pow3, avg

Three different bootstrap sizes were tested for the bootstrapping method: 10%, 20% and 30%.

4.3 Compound sinus

The parameter settings for the composite sine of equation (2) were the same as for the simple sine, except that the number of individuals was set to 150 (on both layers). After that experiments with more generations and individuals were performed.

4.4 Temperatures of Delhi

In a recent series of experiments, a real dataset capturing temperatures in the Indian city of Delhi was tested. The feature set contained the functions *; +; –; %; inv; sqrt; pow2 on the first layer and *; +; –; avg on the second layer. The generation limit was set to 530 for plain GP and 500+30 for 2L-GP. Basic experiments compared GP and 2L-GP with 50 individuals on both layers with 250 individuals on both layers.

5 Results

5.1 Hyperbola

Three different bootstrap sizes were tested for the bootstrapping method: 1. plain GP with additional, 2. plain GP with additional, 1500 individuals, 3. GP-2L without diversification with additional, 4. GP-2L with bootstrapping (10%), 5. GP-2L with bootstrapping (30%), 6. GP-2L with bootstrapping (60%).

Fig. 3 Comparison of plain GP with two-layer GP with bootstrapping.

More generations and individuals on the second layer

Due to the lack of accuracy of the models, experiments were conducted with this settings:

1. GP-2L+bootstrapping (60 %), 30 sub-models, 50 individuals, 30 generations (layer 2),
2. GP-2L+bootstrapping (60 %), 30 sub-models, 150 individuals, 30 generations (layer 2),
3. GP-2L+bootstrapping (60 %), 30 sub-models, 50 individuals, 120 generations (layer 2),
4. GP-2L+bootstrapping (60 %), 30 sub-models, 250 individuals, 120 generations (layer 2),
5. GP-2L+bootstrapping (60 %), 30 sub-models, 350 individuals 120 generations (layer 2),
6. GP-2L+bootstrapping (60 %), 30 sub-models, 500 individuals, 120 generations (layer 2).

Fig. 4 shows that, in this case, increasing the number of individuals and generations on the second layer had a significant positive effect on the error of the generated models, and most of the models generated using the last three settings had an error in the order of $10^{-3}$ and the lower quartile of error was below $10^{-4}$.

Increasing the number of individuals and generations on both layers and adding features improved the quality of the models and reduced the upper quartile error below 0.01. Increasing the number of individuals and generations on the first layer also had a positive effect.
More generations and individuals on the second layer

Due to the lack of accuracy of the models, experiments were conducted with this settings:

1. GP-2L+bootstrapping (60%), 30 sub-models, 50 individuals, 30 generations (layer 2),
2. GP-2L+bootstrapping (60%), 30 sub-models, 150 individuals, 30 generations (layer 2),
3. GP-2L+bootstrapping (60%), 30 sub-models, 50 individuals, 120 generations (layer 2),
4. GP-2L+bootstrapping (60%), 30 sub-models, 250 individuals, 120 generations (layer 2),
5. GP-2L+bootstrapping (60%), 30 sub-models, 350 individuals 120 generations (layer 2).

Figure 4 – Two-layer GP with bootstrapping (60%) with different numbers of individuals and generations on the second layer.

Figure 4 shows that, in this case, increasing the number of individuals and generations on the second layer had a significant positive effect on the error of the generated models, and most of the models generated using the last three settings had an error in the order of $10^{-3}$ and the lower quartile of error was below $10^{-4}$.

Increasing the number of individuals and generations on both layers and adding features improved the quality of the models and reduced the upper quartile error below 0.01. Increasing the number of individuals and generations on the first layer also had a positive effect.

5.2 Compound sine

The bootstrapping method was compared with plain GP and two-layer GP:

1. plain GP of 150 individuals,
2. two-layer GP without training data diversification of 150 individuals,
3. two-layer GP with bootstrapping 30% of 150 individuals,
4. two-layer GP with bootstrapping 60% of 150 individuals.

Results can be seen in the Fig. 5.

Figure 5 – Comparison of a plain GP and a two-layer GP with a two-layer GP with bootstrapping on a compound sine function.

The bootstrapping method yielded a noticeable improvement on the compound sin, with 60% bootstrap producing better models on average than the 30% bootstrap and producing results comparable to the simple two-layer GP.

An attempt to increase the accuracy of the models

Experiments with increased populations and generations were designed to increase precision:

1. GP-2L with bootstrapping 60%, 150 individuals,
2. As 1., 200 generations (layer 1), 350 individuals, 240 generations (layer 2),
3. As 1., 150 individuals, 230 generations (layer 1), 500 individuals, 350 generations (layer 2) + sqrt, pow2 function (layer 1), 90 sub-models.

Figure 6 – Comparison of different configurations of the bootstrapping (60%) on the compound sine function

Figure 6 shows that when increasing the parameters of the two-layer GP with the bootstrapping method, the error of the resulting models decreased visibly and the last configuration had an error around 0.1.

5.3 Temperatures of Delhi

The first set of experiments with this dataset looked like this:

1. plain GP, 50 individuals,
The bootstrapping method yielded a noticeable improvement on the compound sin, with 60% bootstrap producing better models on average than the 30% bootstrap and producing results comparable to the simple two-layer GP.

An attempt to increase the accuracy of the models

Experiments with increased populations and generations were designed to increase precision:

1. GP-2L with bootstrapping 60%, 150 individuals,
2. As 1., 200 generations (layer 1), 350 individuals, 240 generations (layer 2),
3. As 1., 150 individuals, 230 generations (layer 1), 500 individuals, 350 generations (layer 2) + sqrt, pow2 function (layer 1), 90 sub-models.

Fig. 6 shows that when increasing the parameters of the two-layer GP with the bootstrapping method, the error of the resulting models decreased visibly and the last configuration had an error around 0.1.

5.3 Temperatures of Delhi

The first set of experiments with this dataset looked like this:

1. plain GP, 50 individuals,
2. two-layer GP, bootstrapping 60%, 50 individuals on both layers,
3. plain GP, 250 individuals,
4. two-layer GP, bootstrapping 60%, 250 individuals on both layers.
The results of these experiments are shown in the Fig. 7.

The two-layer GP with bootstrapping method managed to improve the accuracy of the resulting models compared to the plain variant. The greater improvement occurred in experiments with 250 individuals.

5.3.1 Increasing performance

To improve the performance of the two-layer GP on this dataset, the following experiments were designed:

1. GP-2L, bootstrapping 60%, 50 individuals,

2. As 1., 50 individuals on layer 1, 250 individuals and 120 generations on layer 2,

3. As 1., 150 individuals on layer 1, 250 individuals and 120 generations on layer 2, set of functions on layer 1 = \{*, +, –, %, inv, pow2, pow3, sqrt, sin, abs\},

4. As 1., 250 individuals on both layers, 500 and 30 generations, set of functions on layer 1 = \{*, +, –, %, inv, pow2, pow3, sqrt, sin, abs\}, 60 sub-models.

Fig. 8 shows that by gradually increasing the parameter values, the error was reduced to a relatively low value (given the nature of the data, the functions used and the GP implementation with integer constants). The error corresponds more to learning the trend without overfitting. The addition of more functions on the first layer also had a positive effect on reducing the error of the generated models. The last setting generated the most consistent models of all settings tested.
6. Discussion

The goal of this paper was not to find exact models, but to explore the dynamics and benefits of a two-layer genetic programming algorithm on hard problems. For this reason, the experiments were chosen so that finding mathematical models and searching the state space would be difficult for genetic programming. In the case of the mathematical model search for the hyperbola, no operator or division function (or % or inverse) was present in the feature set. The search for a simple sine function used relatively simple functions. The search for the composite sine function also omitted the sine function from the set of functions. And the entire genetic programming algorithm used only integer constants. Thus, the real constants in genetic programming had to evolve by evolution using a suitable combination of integer constants and the functions of division, inversion, or the square root function.

Two-layer genetic programming coupled with ensemble learning techniques on the experiments performed showed the potential for improving the performance of genetic programming. Simple two-layer genetic programming in the selected settings produced, on average, more accurate and consistent mathematical models of the tested datasets. Some of the accuracies of the generated trees did not achieve usable results. Also, two-layer genetic programming coupled with bootstrapping produced better results on average than plain GP. The second layer responded positively to the greater variety of models generated by the first layer. The selection of the feature set also had a great influence on the quality of the resulting models, and this was done on both layers.

7. Conclusion

In future research, it would be useful to investigate the dynamics of the parameter settings in more detail, especially the number of sub-models, the number of
generations and individuals on both layers. Another area of investigation could be the appropriateness of individual features on the second layer. Furthermore, it would be interesting to try two-layer programming on data containing noise, on datasets with a larger number of variables, and on problems in the big data category. Another interesting topic for research would be to use different methods to build sub-models on the first layer simultaneously, thus providing more diversity in the sub-models.

Another subject of investigation could be the intentional diversification of feature subsets for sub-model creation. Methods for diversifying input variables would also need to be added when looking for models of multivariate data. Creating sub-models may be more time consuming than a plain GP approach due to the larger number of evaluations. Another promising idea would be to introduce microscopic genetic operators into the second layer of a two-layer GP. The possibility of adding additional layers would deserve further investigation. Adding additional layers would bring genetic programming closer to the principles of deep learning, giving rise to the term multilayer or deep genetic programming.

References


229


