Exploring Hidden Semantics in Neural Networks with Symbolic Regression

Yuanzhen Luo
Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum-Beijing
Beijing, China
strugglingluo@gmail.com

Qiang Lu*
Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum-Beijing
Beijing, China
luqiang@cup.edu.cn

Xilei Hu
College of Artificial Intelligence , China University of Petroleum - Beijing
Beijing, China
yeshendd@163.com

Jake Luo
Department of Health Sciences and Administration, University of Wisconsin Milwaukee, Milwaukee
WI, United State
jakeluo@uwm.edu

Zhiguang Wang
Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum-Beijing
Beijing, China
cwangzg@cup.edu.cn

ABSTRACT

Many recent studies focus on developing mechanisms to explain the black-box behaviors of neural networks (NNs). However, little work has been done to extract the potential hidden semantics (mathematical representation) of a neural network. A succinct and explicit mathematical representation of a NN model could improve the understanding and interpretation of its behaviors. To address this need, we propose a novel symbolic regression method for neural works (called SRNet) to discover the mathematical expressions of a NN. SRNet creates a Cartesian genetic programming (NNCGP) to represent the hidden semantics of a single layer in a NN. It then leverages a multi-chromosome NNCGP to represent hidden semantics of all layers of the NN. The method uses a (1+λ) evolutionary strategy (called MNNCGP-ES) to extract the final mathematical expressions of all layers in the NN. Experiments on 12 symbolic regression benchmarks and 5 classification benchmarks show that SRNet not only can reveal the complex relationships between each layer of a NN but also can extract the mathematical representation of the whole NN. Compared with LIME and MAPLE, SRNet has higher interpolation accuracy and trends to approximate the real model on the practical dataset[1].

CCS CONCEPTS

• **Computing methodologies → Genetic programming**.

KEYWORDS

symbolic regression, neural network, Cartesian genetic programming

ACM Reference Format:
Yuanzhen Luo, Qiang Lu, Xilei Hu, Jake Luo, and Zhiguang Wang. 2022. Exploring Hidden Semantics in Neural Networks with Symbolic Regression. In *Proceedings of The Genetic and Evolutionary Computation Conference 2022 (GECCO '22).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Neural networks (NNs) have been successfully applied in many problems, such as CNN for object recognition [27], RNN for time series analysis [10], and Bert for natural language processing (NLP) [4]. However, neural networks are often seen as black-boxes because their input-output (IO) relationships are difficult for a human to understand [11]. Sometimes, it is almost impossible to interpret the NN behaviours when models make unexpected predictions on some datasets, such as adversarial examples [32] and white noise images [23]. Some of the recent work has been centred on researching and explaining the black box behaviours, as summarized in several survey papers [1, 30].

In this paper, we aim to develop a new symbolic regression-based method to explore hidden semantics in NNs. Here, a typical hidden semantics interpretation method refers to explaining a NN with an explicit math function. If a function $f(x^{(i)})$ explains a single hidden layer $h(x^{(i)})$ on a certain input-output $\left(x^{(i)}, y^{(i)}\right)$ or a small range of inputs-outputs, it is called **local explanation** [37], such as LIME [28] and MAPLE [25]. If a function $f(x)$ is able to explain a hidden layer $h(x)$ on the whole dataset, it is called **global explanation**, such as Visualization method [3, 18] and Net2vec [9]. Although these methods show some degree of success in extracting hidden semantics from NN, they have the following three limitations. (1) Local explanation methods can give a mathematical expression, such as a linear model and a decision tree, for each $(x^{(i)}, y^{(i)})$. However, they cannot obtain a general expression for the whole dataset. Although most global explanation methods can visualize

*Corresponding author.
[1]Code and appendix at https://kgae-cup.gihub.io

GECCO '22, July 9–13, 2022, Boston, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

NNs on the whole dataset, they cannot give a mathematical expression for explaining the dataset. (2) These local or global methods often leverage pre-defined interpretable models to explain the hidden semantics of NNs. For example, LIME can use linear models, decision trees, or falling rule lists as interpretable models. However, these pre-defined models may not capture hidden semantics in some situations because the real characteristics of a NN model are often unknown, and applying a predefined model to explain the networks may be inappropriate. (3) These methods cannot generate a mathematical expression that can represent the hidden semantics of all layers in a NN.

To overcome these limitations, this paper leverages the **symbolic regression** (SR) method to explain a NN. In SR, for a given dataset $\{x, y\}$, the algorithm can find a symbolic function $f(x) = y'$ that minimizes the distance between y and y' in the mathematical expression space. SR has great flexibility in generating mathematical expressions; hence, it does not need a predefined model to capture the relationships in the dataset. However, classical SR methods, such as GP [13, 31], GEP [7], ABGEP [36], SPGEP [17], SPJGEP [16] and linear GP [2], usually handle the symbolic function $f(x)$ with a single output y', i.e., y' is a number and not a vector. They cannot represent the relationship $g(W_i h_{i-1} + b)$ of each layer i in a NN because each layer's output is a vector, matrix, or tensor. Therefore, when these GPs explain NN [5, 8], they can only give a mathematical expression to show the semantics of the whole NN with a single output value, not each layer in the NN. Although Cartesian Genetic Programming (CGP) [22] supports multiple outputs, CGP cannot represent semantics in a NN. Because each CGP output corresponds to a hidden node, and it cannot provide a general model f that represents hidden semantics in the layer. To obtain a general model, we assume that the relationship between input and output in a layer (or a NN) has the mathematical expression format $w_i^s f_i(h_{i-1}^s) + b_i$, where w_i and b_i may be a number, vector, matrix, or tensor, and f_i is a mathematical function that represents the hidden semantics of the layer.

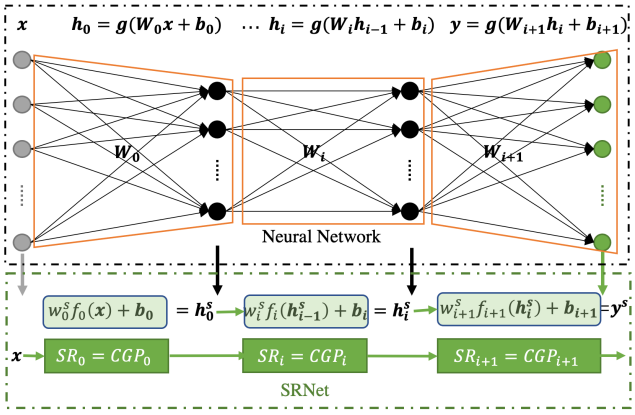


Figure 1: SRNet for exploring hidden semantics in NN.

Based on the above assumption, this paper proposes a novel SR method (called **SRNet**) to mine hidden semantics of all layers in a NN simultaneously, as shown in Figure 1. SRNet is an evolutionary computing algorithm. In each evolution, SRNet first leverages the Cartesian Genetic Programming (CGP) [20, 21] to find each layer's mathematical function $f_i(h_{i-1}^s)$. It then uses the Newton-Raphson method [29] (or L-BFGS method [15]) for few (or many) variables to obtain w_i^s and b_i so that $h_i^s = w_i^s f_i(h_{i-1}^s) + b_i$ approximates the output h_i of the layer i in a NN. At the end of the evolution, SRNet will capture hidden semantics of all layers in a NN when $h_i^s \approx h_i$ (including $y^s \approx y$). The main contributions in the paper are summarized as follows:

- The paper proposes a new method called SRNet to explain hidden semantics of all layers in a NN. SRNet generates a mathematical expression in the format of $w_i^s f_i(h_{i-1}^s) + b_i$ that can be used to explain a NN.
- To speed up SRNet, we create a multi-chromosome CGP [34] evolutionary strategy embedded in the Newton-Raphson method.
- Experiments show that the proposed SRNet can capture hidden semantics in NN in 12 SR benchmarks and 5 classification benchmarks. Compared with LIME and MAPLE, SRNet has higher interpolation accuracy and trends to approximate the real model on the practical dataset.

The remainder of this paper is organized as follows. In Section 2, we introduce the background knowledge about Cartesian Genetic Programming. Then, we propose SRNet to explore hidden semantics in NNs in Section 3. Section 4 and 5 report the experimental results. We conclude the paper in Section 6.

2 CARTESIAN GENETIC PROGRAMMING

CGP is a directed acyclic graph-based genetic programming algorithm for addressing the SR problem [22]. In CGP, the graph consists of a two-dimensional grid of computational nodes, as shown in Figure 2. These nodes are classified into three categories: input, output, and function. The input (or output) nodes represent the input (or output) values x (or o), which is encoded into an integer, such as x_0 encoded by "1" and O_A encoded by "4". The function nodes are computational expressions. Each function node has three parts, input, computational expression, and output, encoded by a series of integers. As each node has only one output, the output code is used to index the node. For example, a function node "+" is regarded as the code "$\langle \underline{0}012 \rangle$". In this code, the first integer $\underline{0}$ is the code of the function "+". The two middle integers "0" and "1" represent two inputs of the function "+", and they are also the outputs of two previous nodes. The last integer "2" is the index of the node.

In the example CGP diagram, there is no edge between any two nodes in the same column. Two nodes at different columns can be linked if one's input code equals the other's output code. A genotype is used to represent a CGP, as shown in Figure 2. The genotype contains two categories of nodes: the functional nodes and the output nodes. As the genotype has multiple output nodes, the genotype can describe multiple computational expressions. For example, the genotype in Figure 2 generates three mathematical expressions, $O_A = -x_1$, $O_B = 2x_0x_1 + x_1^2$, and $2x_0 + x_1$.

CGP usually leverages the $(1 + \lambda)$ evolutionary strategy [12, 26] to find the best fitted mathematical expression. In each evolution, $(1 + \lambda)$ EA utilizes mutation to generate λ offsprings. For CGP, the mutation randomly chooses a gene location and changes the allele at the location to another valid random value. A valid value is from

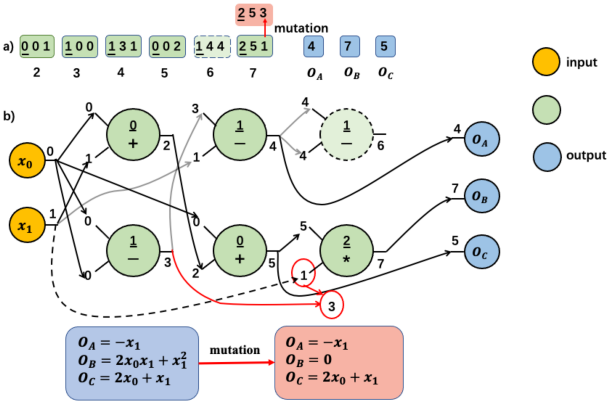


Figure 2: An example of CGP. a) genotype. b) phenotype.

the function look-up table if a computational expression gene is chosen for mutation. If an input gene is chosen, a valid value is from the output set of its previous nodes. The mutation does not change the output gene. For example, in Figure 2, a mutation changes the input gene "1" of the node "7" to "3". Then, the output O_B becomes "0".

There are two of important features of CGP compared with other traditional tree-based GP: 1) Few primary parameters are required in the evolution process (i.e. number of columns and rows of function nodes, point mutation rate), makes CGP easy to tune. 2) the length of expressions produced by CGP can flexible changes. This is because the input of node could be mutated to the another previous node, which possibly make the final expression less complexity (e.g. see how the O_B changed in Figure 2). Thus, instead of leveraging traditional tree-based GP algorithms, we use CGP in this paper.

The multi-chromosome Cartesian genetic programming (MCGP) [34] encodes multiple chromosomes into a single genotype. Each chromosome code is similar to the genotype of CGP. So, MCGP can provide a solution to a large problem by dividing it into many smaller sub-problems. For simultaneously exploring hidden semantics of all layers in a NN, MCGP encodes each layer semantics as a chromosome into a genotype. Thus, a chromosome represents the semantics of one layer, and the genotype represents all NN layers' semantics. After MCGP uses the $(1+\lambda)$ multi-chromosome evolutionary strategy [35] to acquire a best-fitted individual, it also obtains these semantics.

3 SRNET

This section proposes the SRNet, a method based on MCGP [34] to simultaneously explain the hidden semantics of layers in a NN. As shown in Figure 1, SRNet can find a group of $h_i^s = w_i^s f_i(h_{i-1}^s) + b_i$ that approximates each NN layer output h_i, i.e.,

$$\{h_0^s, ..., h_n^s\} = \underset{h_i^s \in \mathcal{F}}{argmin} \sum_{i=0}^{n} \mathcal{L}(h_i, h_i^s). \tag{1}$$

To find the group $\{h_i^s\}$ quickly, SRNet needs to address the following problems: 1) how to encode these h_i^ss, and 2) how to find functions to explain hidden semantics in a NN. Section 3.1 shows our solution to the first problem, and section 3.2 provides a multi-NNCGP evolutionary strategy to address the second problem.

3.1 SRNet Encoding

The hidden semantics h_i^s of each layer in a NN are represented as

$$w_i^s f_i(h_{i-1}^s) + b_i, \tag{2}$$

where $f_i(h_{i-1}^s)$ is a mathematical expression that represents the general semantics in the layer i, and w_i^s and b_i are a weight vector, matrix, or a tensor, as shown in Figure 3.

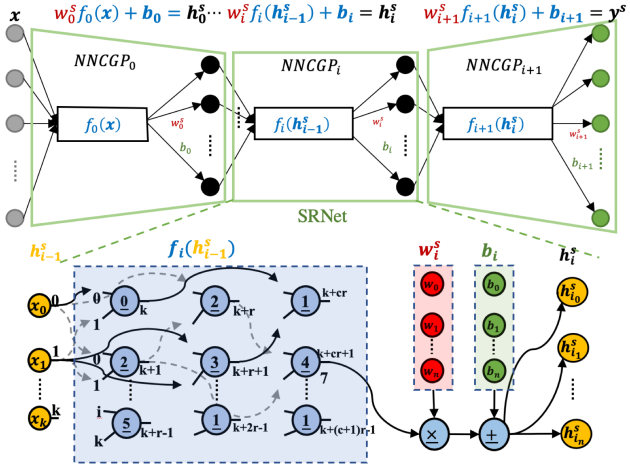


Figure 3: SRNet encoded by Multiple NNCGPs

To capture a NN layer's semantics h_i^s, we define that the neural network CGP (NNCGP) consists of three components, including general semantic model, constant, and operator. The general semantic model is used to generate $f_i(h_{i-1}^s)$. It has three parts, k inputs, $c \times r$ functions and an output. 2) The constant component includes the weight vector w_i^s and the bias vector b_i, where $|w_i^s| = |b_i| = |h_i^s|$. 3) The operator component consists of the two functions, "×" and "+".

SRNet uses multiple NNCGPs (called MNNCGP) to encode genotype. Since a NNCGP can represent one layer's semantic, multiple NNCGPs can be used to capture hidden semantics of all layers in a NN. These NNCGPs are regarded as chromosomes that constitute a genotype. In the genotype, each NNCGP$_i$'s output h_i^s is the input of its next NNCGP$_{i+1}$.

3.2 Evolution Strategy

SRNet leverages a multi-NNCGP evolutionary strategy embedded by the Newton-Raphson method (called **MNNCGP-ES**) to find the best-fitted genotype that represents hidden semantics of all layers in a NN. MNNCGP-ES is similar to the $(1+\lambda)$ multi-chromosome evolutionary strategy [35]. MNNCGP-ES includes the following operations: mutation, fitness evaluation, and selection. Mutation, with a certain probability, change each allele in MMCGP to another valid random value [22].

3.2.1 Fitness Evaluation. To evaluate a genotype encoded by MNNCGP, the fitness function is defined as the following equation,

$$fitness = \frac{1}{N} \sum_{i=0}^{N-1} \mathcal{L}(h_i, h_i^s) + \mathcal{L}_o(y, y^s) \tag{3}$$

where $\mathcal{L}$ is the mean squared error (MSE) of each middle layer. $\mathcal{L}_o$ is an error function of the output layer. It is a cross-entropy loss in the classification task, while it is MSE in the regression task. h_i (h_i^s) is the output of the ith layer in a NN (NNCGP$_i$). y and y^s are the outputs of the NN and SRNet, respectively.

Equation 2 indicates that obtaining h_i^s needs two computations, as shown in Figure 3. One is $f(h_{i-1}^s)$ that generates an output by the CGP code. The other is the parameter computation that obtains the constant vectors, w_i^s and b_i, by the Newton-Raphson method that performs an update operation according to the following equation.

$$p = p - H^{-1}(p)\nabla l(p), \tag{4}$$

where p is w_i^s or b_i, $H(p)$ is Hessian, $\nabla l(p)$ is the gradient of the loss function $h_i - (w_i^s f(h_{i-1}^s) + b_i)$. The Hessian is difficult to obtain if it is a high-dimensional matrix (i.e., many neurons). Therefore, to solve this problem, the Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS) [15] is used for limited memory and time-saving.

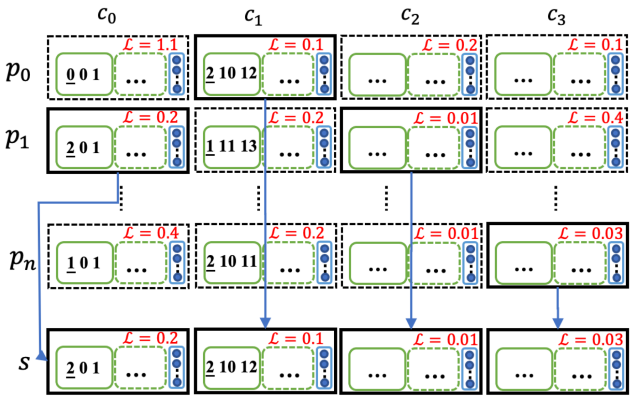


Figure 4: Selecting chromosomes at each points from all individuals as a "super" individual.

3.2.2 Selection. Selection aims at generating a 'super' individual from the population. The 'super' individual consists of a set of super chromosomes that have the best fitness at each position from all individuals, i.e., $min\ \mathcal{L}(h_i, h_i^s)$, as shown in Figure 4. Since these chromosomes constitute an input-output sequence where each chromosome output is the input of its next chromosome, they need to be selected in the order of their positions in the population. After evaluating the finesses of the chromosomes at a certain position (c_0) of the population, the selection picks up the chromosome that has the best fitness score as the super chromosome (s_0). Then, it evaluates chromosome at the next position(c_1) with s_0 as input. Moreover, it obtains a chromosome with the best fitness as the next super chromosome (s_1). Repeating the above evaluation policies results in a group of selected chromosomes $\{s_0, s_1, ...\}$. These selected chromosomes form a super individual.

However, for high-dimensional problems, evaluating the finesses of the chromosomes is very time-consuming when using the Newton-Raphson method. So, L-BFGS is used to replace the Newton-Raphson method to compute the two weight vectors (w_i^s and b_i) of all individuals. L-BFGS can speed up the fitness evaluation.

3.2.3 MNNCGP-ES. The MNNCGP-ES pseudocode is listed in Algorithm 1. MNNCGP-ES combines the fitness evaluation and the selection method mentioned before, using the $(1 + \lambda)$ evolution strategy to evolve generation-by-generation to obtain the optimal individual.

Algorithm 1 MNNCGP-ES

Input: $\mathbb{D}s(\boldsymbol{h_0}, \boldsymbol{h_1}, \ldots, \boldsymbol{h_n})$, λ
Output: a best-fitted individual

```
randomly initializes λ individuals with MNNCGP
while fitness > 1e − 4 and max generation not reached do
    //obtain an new parent s by the selection operation
    s ← NULL
    calculate each chromosome's output h_i^s at the position i according to Equations 2 and 4;
    obtain the chromosome c_i^k by Equation 3
    s ← s ∪ c_i^k
    execute λ mutations on s to generate λ offsprings.
end while
Return the best-fitted parent s according to the fitness computed by Equation 3.
```

4 EXPERIMENTS

To validate the SRNet's ability to explain hidden semantics in the neural network (NN), we tested the SRNet on the built NNs of 12 symbolic regression benchmarks as well as 5 classification benchmarks, listed in Table 1. Moreover, the sample sizes and feature sizes of the 17 benchmarks are illustrated in Figure 5.

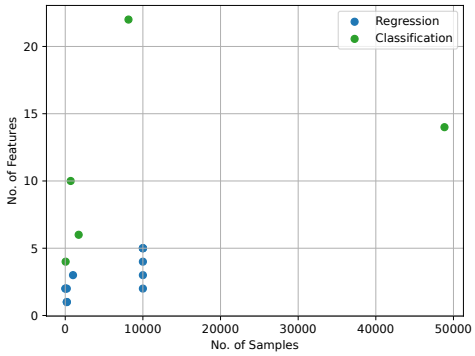


Figure 5: The sizes of samples and features in 17 benchmarks

Table 2 lists the parameters of the three algorithms, LIME, MAPLE, and SRNet. All parameters of the three algorithms are fixed on all benchmarks.

4.1 Regression Task

To validate the SRNet's ability to explore hidden semantics of NN in the regression task, we chose 12 symbolic regression benchmarks $K0 - K5$ and $F0 - F5$. The benchmarks $K0 - K5$ were chosen from the commonly used SR Benchmarks [19], while $F0 - F5$ are from physical laws [33]. We generated 12 datasets (called **true datasets**)

Alias	Function/Name	Training Dataset	MLP
$K0$	$sin(x) + sin(x + x^2)$	$[-1, 1, 200]$	[3, 3] s0.01
$K1$	$2sin(x)cos(y)$	$[-1, 1, 200]$	[3, 3] s0.1
$K2$	$3 + 2.13ln\|x\|$	$[-50, 50, 200]$	[5, 5] s0.03
$K3$	$\frac{1}{1+x^{-4}} + \frac{1}{1+y^4}$	$[-5, 5, 10^4]$	[4, 4, 4] a0.03
$K4$	$\frac{30xy}{(x-10)z^2}$	$x, y : [-1, 1, 10^3]$ $z : [1, 2, 10^3]$	[4, 4] a0.003
$K5$	$xy + sin((x-1)(y-1))$	$[-3, 3, 20]$	[5, 5] a0.003
$F0$	$\frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}$	$m_0 : [1, 5, 10^4]$ $v : [1, 2, 10^4]$ $c : [3, 10, 10^4]$	[3, 3] a0.01
$F1$	$q_1 q_2 \frac{r}{4\pi\epsilon r^3}$	$[1, 5, 10^4]$	[3, 3] a0.01
$F2$	$Gm_1m_2(\frac{1}{r_2} - \frac{1}{r_1})$	$[1, 5, 10^4]$	[3, 3] a0.01
$F3$	$\frac{1}{2}kx^2$	$[1, 5, 10^4]$	[3, 3] a0.01
$F4$	$-6.4\frac{G^4}{c^5}\frac{1}{r^5}(m_1m_2)^2(m_1+m_2)$	$m_1, m_2 : [1, 5, 10^4]$ $G, c, r : [1, 2, 10^4]$	[5, 5] a0.03
$F5$	$\frac{q}{4\pi\epsilon y^2}[4\pi\epsilon V_e d - \frac{qdy^3}{(y^2-d^2)^2}]$	$q, V_e, \epsilon : [1, 5, 10^4]$ $d : [4, 6, 10^4]$ $y : [1, 3, 10^4]$	[3, 3] a0.03
$P0$	adult	48842	[100, 100] s0.01
$P1$	analcatdata_aids	50	[200, 100, 100] s0.01
$P2$	agaricus_lepiota	8145	[100, 100] s0.01
$P3$	breast	699	[100, 100] s0.03
$P4$	car	1728	[100, 100, 100] s0.01

Table 1: The dataset of training 17 MLPs. In each cell of the column 'MLP', the integer list is the number of neurons in each hidden layer, 'a' or 's' is the Adam or SGD optimization method, respectively, float number being the learning rate.

Name	Parameter	Value
LIME	Number of Features	10
	Number of Samples	5000
	Distance Metric	Euclidean
	Regressor Model	Ridge
MAPLE	Number of Estimators	200
	Max Features	0.5
	Min Samples Leafs	10
	Regularization	0.001
	Ensemble Model	Random Forest
	Regressor Model	Ridge
	Classifier Model	Logistic Regressor
SRNet	Number of Rows	10
	Number of Cols	10
	Function Set	$+, -, \times, \div, sqrt, square, \sin, \cos, ln, \tan, exp$
	Number of Constants	1
	Population Size	200
	Max Generations	5000
	Mutation Probability	0.4

Table 2: Algorithm parameters

according to these benchmarks. Each of these datasets has different sample sizes (see 'Training Dataset' in Table 1). For example, for the $K1$ problem in Table 1, we randomly sampled 200 x and y values in the range of $[-1, 1]$, respectively. For each of the 12 datasets, we randomly took 80% samples from it as the **training dataset**, and the other as the **test dataset**. Then, we built 12 Multi-Layer Perception neural networks (**MLP**) with a sigmoid activation function using the training datasets. The training parameters are listed in Table 1. For example, for $K1$, we created an MLP with two hidden layers where each layer had three hidden nodes. The MLP was trained by the SGD optimization method with a learning rate of 0.01.

After each MLP was trained, we collected each NN layer's input and output data of the 12 MLPs as the NN explanation datasets. We then ran the MNNCGP-ES, LIME, and MAPLE 30 times on each NN explanation dataset.

4.2 Classification Task

To validate the SRNet's ability to explore hidden semantics of NN in the classification task, we chose 5 classification benchmarks named $P0 - P4$ from the PMLB [24] with the different number of samples and features (see Figure 5). The column 'Training Dataset' on the rows "P0-P4" indicates the number of samples, as shown in Table 1. Training 5 MLPs is similar to the regression task except for the function "softmax" that replaces their output functions.

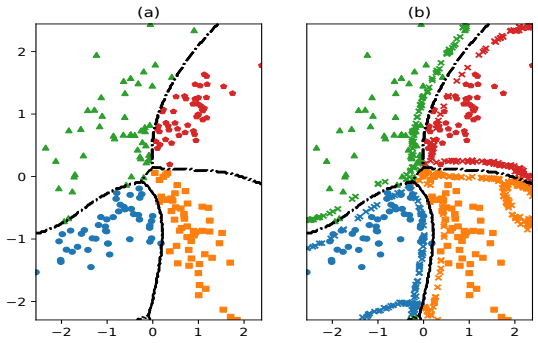


Figure 6: (a) The decision boundary of a classification model with 4 classes. (b) sampling around decision boundaries. The marker 'X' is represented as the new sample around the decision boundaries.

After training these classification MLPs, we need to compare SRNet with their decision boundaries, not their outputs. Because the MLP's outputs on the training datasets are sparse and do not fully represent the NN's classification ability, as shown in Figure 6(a). Using these outputs to train SRNet may result in wrong results. To obtain the decision boundaries of the trained MLP, we leverage a uniform sample method around its decision boundary (called **USDB**). USDB first evaluates the range of the training dataset. It then randomly samples n points in the range. It finally selects s points with the shortest distance to the decision boundary according to Equation 5 [6, 14].

$$d(x_i, B) = \sum_{k}^{C} |p_k(x_i) - \frac{1}{C}| \tag{5}$$

, where x_i is a sample, B is the decision boundary of a NN, and C is the number of sample classifications. p_k is the probability that the x_i belongs to the kth classification, which is the NN output owing to its activation function "softmax".

After each classification MLP was trained, we utilized USDB to generate samples around the decision boundary of the MLP. We then fed these samples into the MLP and collected each NN layer's input and output as the MLP explanation dataset. We finally ran

the MNNCGP-ES, LIME, and MAPLE 30 times on the explanation dataset.

5 RESULT AND ANALYSIS

5.1 Regression Task

In the following regression tasks we only show results on the six benchmarks, $K0$, $K1$, $F0$, and $F1$. The regression results on all benchmarks are in the appendix.

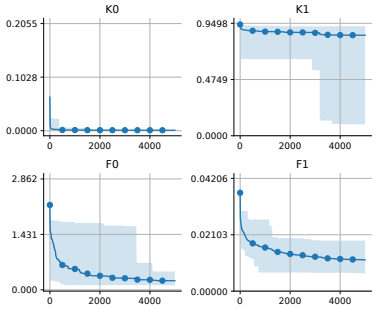


Figure 7: MNNCGP-ES convergence curve.

5.1.1 Fitness Convergence. Figure 7 illustrates the convergence curve of the fitness scores of MNNCGP-ES on each MLP for the regression tasks. The blue line is the average fitness score in 30 experiments. It gradually decreases at the beginning and trends to a flat curve. It means that, statistically, MNNCGP-ES could find the mathematical expressions that approximate the hidden semantics of each layer in NNs according to Equation 3, as shown in Table 3. It also indicates the feasibility to use the combination of these mathematical expressions to represent the whole semantics of each MLP, as shown in Table 4.

Dataset	$h_0^s(x)$	$h_1^s(h_0^s)$	$h_2^s(h_1^s)$	y^s
$K0$	$sin(0.26x+sin(sin(x))-0.068)$ **(6.74e-04)**	$0.88-\cos(h_0^s)_0$ **(7.54e-05)**	–	$-sin(sin((h_0^s)_0-0.36))$ **(2.37e-04)**
$K1$	$5.15e\text{-}05x_1-0.0072sin(x_0)-5.15e\text{-}05$ **(5.34e-02)**	$-(h_0^s)_0+(h_0^s)_2$ **(1.39e-03)**	–	$-(h_1^s)_0+(h_1^s)_2+0.0011$ **(6.88e-02)**
$F0$	$cos((\frac{0.67x_1}{x_2}+\log(x_0)))$ **(6.35e-03)**	$-(h_0^s)_1+\cos\left((h_0^s)_2\right)$ **(1.04e-02)**	–	$tan((tan(((h_1^s)_0-0.33))+0.11))$ **(9.63e-02)**
$F1$	$\log\left(\frac{x_0x_1}{x_2x_3^{\frac{3}{2}}}\right)$ **(9.59e-03)**	$(h_0^s)_1^4$ **(2.46e-04)**	–	$(h_1^s)_0+\sin\left((h_1^s)_0\right)$ **(1.45e-03)**

Table 3: The mathematical expressions of each layer in NNs.

Not all ranges of fitness scores (light blue areas) become smaller as the MNNCGP-ES runs, such as $k1$. However, the low bounds of these lines always become smaller and trend to be zero at the later stage. It means that the more episodes the MNNCGP-ES runs, the more likely MNNCGP-ES is able to find the mathematical expressions that can be used to explain the hidden semantics of a NN. The slow decrease of fitness curves also indicates the need to run MNNCGP-ES with sufficient times to obtain the best-fitted mathematical expressions.

5.1.2 Semantics Evaluation. The results show that the proposed SRNet method can acquire a fitted mathematical expression to explain hidden semantics of each layer, as shown in Table 3. Each of these mathematical expressions represents the general semantics f_i in the expression $w_i^s f_i(h_{i-1}^s)+b_i$ (Equation 2). The number below each f_i is its fitness. For example, for K_1, MNNCGP-ES finds the general semantics $-(h_0^s)_0+(h_0^s)_2$ at the second hidden layer h_1. The fitness of $w_i^s\times(-(h_0^s)_0+(h_0^s)_2)+b_i$ is $1.39e-03$. All fitness values in the hidden layers are less than 0.1. It means that f_i captured by MNNCGP-ES can represent (approximate) the semantics of each hidden layer in a NN.

Dataset	$O^s(x)$
$K0$	$0.29-4.01sin((sin((2.36cos((0.41sin((0.26x+\sin(\sin(x))-0.068))+0.49))-2.03))))$ **(6.11e-04)**
$K1$	$-0.01x_1+1.69\sin(x_0)+0.0021$ **(9.62e-02)**
$F0$	$3.05-7.00tan((tan((0.58\cos\left(\log(m_0)+\frac{0.67v}{c}\right)-0.76\cos\left(0.22\cos\left(\log(m_0)+\frac{0.67v}{c}\right)-1.01\right)+0.35))-0.11))$ **(1.05e-01)**
$F1$	$0.022\left(0.46\log\left(\frac{q_1q_2}{er^{\frac{3}{2}}}\right)+1\right)^4+3.22\sin\left(0.0068\left(0.46\log\left(\frac{q_1q_2}{er^{\frac{3}{2}}}\right)+1\right)^4-0.00072\right)+0.0059$ **(6.37e-03)**

Table 4: The mathematical expression of each whole NN.

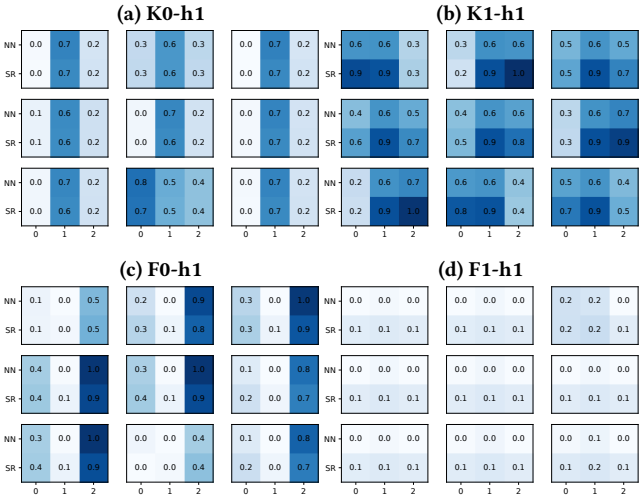


Figure 8: The outputs of the SRNet layer vs the NN layer with 9 random input values.

Figure 8 shows the details of mathematical expressions that MNNCGP-ES finds. The expressions approximate the output of each layer in a NN. To evaluate each layer, we input 9 random values into the mathematical expression and the hidden nodes in the NN, respectively. We then obtained 9 heat maps to show the difference between the output of the mathematical expression and that of the NN hidden nodes. For example, the sub-figure "$K1-h1$" represents the $K1$ outputs of SRNet vs NN in the layer $h1$ with 9 random input values. In the first heat map in "$K1-h1$", $[0.6, 0.6, 0.3]$ are the outputs of three hidden nodes in the NN layer $h1$ with the first input value, while $[0.9, 0.9, 0.3]$ are the output of the mathematical expression in the SRNet layer $h1$. So, Figure 8 explains why a mathematical expression has low fitness, and another has high fitness. Comparing $K1-h1$ with $K0-h1$ in Figure 8, the outputs between NN and SRNet in $K0-h1$ are closer than the output between them in $K1-h1$. Thus,

the fitness "$7.54e-05$" of "$0.88-cos(h_0^s)_0$" is less than "$1.39e-03$" of "$-(h_0^s)_0+(h_0^s)_2$" in Table 3.

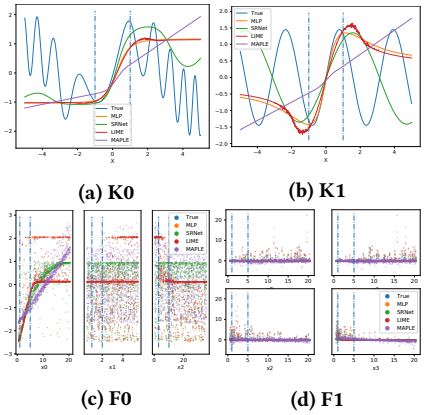
For the output layers in the NNs, the last column y^s in Table 3 lists the mathematical expressions to present the outputs. Their fitness scores of the output layer are better than the scores of the previous layers. There are two formulas of $K5$ and $F5$ whose fitness scores are greater than 1 (See appendix). The reason causing the higher scores at the output layers is that, in a NN, the network structure of the outer layer is different from the hidden layers. The computation function on the output layer is $y = W_{i+1}h_i + b_{i+1}$, while that on the hidden layer is $h_i = W_i h_{i-1} + b_i$. Since y is a number and h_i is a vector, $y = W_{i+1}h_i + b_{i+1}$ is a multivariate linear equation. However, $y^s = w^s_{i+1} f_{i+1}(h^s_i) + b_{i+1}$ on the output layer in SRNet is one-variable linear equation because w^s_{i+1} and b_{i+1} are two numbers, not vectors. Although MNNCGP-ES can find a fitted mathematical function f_{i+1} to represent the NN layers, the one-variable linear equation is still hard to approximate the multi-variable linear equation in the layers.

Table 4 lists the mathematical expressions that represent the whole NN semantics for the regression tasks. Each of final expressions is obtained by combining the mathematical expressions in different layers shown in Table 3. The complexity and length of the mathematical expressions could increase substantially, such as the mathematical expression in $F1$, due to the combination of expressions on every layer. If there are more layers in a NN, the length of the mathematical expression could be longer. Although the math representation generated by SRNet could be lengthy, it provides a straightforward expression to show all layers' hidden semantics of the whole NN.

5.1.3 Performance Comparison. To evaluate the SRNet performance on regression tasks, we ran and compared SRNet, LIME [28], and MAPLE [25] on an interpolation dataset and an extrapolation dataset. The interpolation dataset consists of the NN input-output values. In contrast, the extrapolation dataset consists of the data sampled directly from the original symbolic expression in the column "Function" in Table 1. The interpolation domain is the same as the range of the training dataset shown in Table 1. The size of the extrapolation domain is five times that of the interpolation domain.

Figure 9 illustrates the curves (or distribution points) of the true dataset, as well as the results of NN(MLP)s, SRNet, LIME, and MAPLE, on different symbolic regression benchmarks. For the high dimension datasets, it is not easy to visualize the curves. So, the curves are projected into samples on multiple planes. The curves between two vertical blue lines represent interpolated results, while those outside the two lines are the extrapolated results.

The interpolated results show that SRNet can find the mathematical expressions close to MLP on most of these benchmarks. SRNet can find smoother results than LIME, while it can find results closer to MLP compared with MAPLE. The extrapolated results show that LIME can find the model closest to MLP because LIME is the local explanation method that generates a model for local (several) samples. For a extrapolate dataset, LIME divides the datasets into many groups of local samples and generates a model for each group of samples. Therefore, it needs many local models to explain the extrapolated dataset and cannot provide a general model to



(a) K0 (b) K1

(c) F0 (d) F1

Figure 9: SRNet vs LIME vs MAPLE on the interpolation and extrapolation domain. The area between two blue vertical lines is the interpolation domain. The other area is the extrapolation domain

describe the whole dataset. Unlike LIME, SRNet finds the mathematical expression that represents the whole dataset. In addition, it can trend towards the true dataset generated by the symbolic regression benchmark. The SRNet could help us judge why a NN fails to predict some test data. Under the assumption that the mathematical expression found by SRNet can represent the real model on the practical dataset. For example, on the benchmark $K1$, if a NN is unable to predict certain output, the prediction result could be compared with the output of $-0.01x_1 + 1.69\sin(x_0) + 0.0021$ found by SRNet. The difference between the NN output and SRNet can be used to analyze the gap between the NN and the real model.

5.2 Classification Task

In the following classification tasks we only show the results on the two benchmarks, $P0$, and $P1$. The classification results on all classification benchmarks are included in the appendix.

5.2.1 Fitness Convergence. Figure 10 shows the fitness convergences curves of the SRNet for the classification tasks on the two benchmarks, $P0$ and $P2$. SRNet converges rapidly, especially within about 100th generations. As MNNCGP-ES runs L-BFGS to obtain weight vectors (w^s_i and b_i) of all individuals ($w^s_i f_i(h^s_{i-1}) + b_i$) every 50 generations, MNNCGP-ES only runs L-BFGS two times, and it can converge to an accurate result. The reason is that SRNet does not need a whole dataset to be trained, but a thousand samples around the decision boundary of a NN. For example, on the benchmark P_0, although it has 48842 points, these points are only used to train a classification NN. After training the NN, USDB (Section 4.2) randomly samples 1000 points around the classification NN. Then, the 1000 points are used to train SRNet. So, the process of training SRNet in the classification task is fast.

5.2.2 Semantics Evaluation. Table 5 lists a mathematical expression of each NN layer obtained by SRNet for the two classification tasks, $P0$ (adult) and $P2$ (agaricus_lepiota). Moreover, Table 6 shows the final mathematical expressions obtained by SRNet. For the adult dataset, the two mathematical expressions, $Pr0$ and $Pr1$, represent the prediction probability of SRNet for class 0 (adult makes over

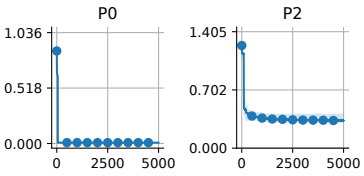


Figure 10: MNNCGP-ES convergence curve.

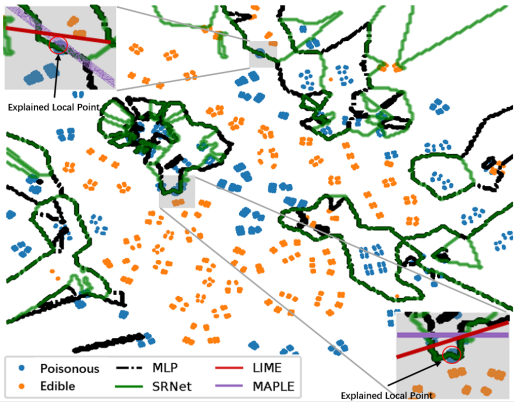


Figure 11: LIME vs MAPLE vs SRNet on $P2$ decision boundary. The samples in the original dataset are reduced to 2D by TSNE, as shown as blue and orange point.

$50K a year) and class 1 (adult makes below $50k a year), respectively. Moreover, $Pr0$ and $Pr1$ indicate that the classification results only depends on the three features, x_6 (relationship), x_7 (race), and x_9 (capital-gain). In addition, they provide a mathematical explanation of how the trained NN classifies data. For the agaricus_lepiota dataset, SRNet gives a simple linear model as the explanation of the classification NN, as shown in $P2$ in Table 6. According to $Pr0$ and $Pr1$, the classification NN mainly focused on the four features, x_{10} (stalk-root), x_{20} (population), x_8 (gill-color) and x_9 (stalk-shape). In this case, SRNet degenerates into a simple linear model. So, when SRNet explains a NN on the classification task, it not only shows how the NN computes on the dataset, but also represents which features (variables) the NN focuses on.

Dataset	$h_0^s(x)$	$h_1^s(h_0^s)$	$h_2^s(h_1^s)$	y^s
$P0$	$-x_6 + x_7 + x_9$ **(1.58e-02)**	$(h_0^s)_{85}$ **(3.09e-04)**	-	$\frac{(h_1^s)_{41}}{(h_1^s)_{31}}$ **(0.0)**
$P2$	$-x_{10} - x_{20} + x_8 + x_9$ **(2.08e-02)**	$(h_0^s)_{63}$ **(5.58e-03)**	-	$(h_1^s)_{33}$ **(2.91e-01)**

Table 5: The mathematical expression of each layer in NNs.

5.2.3 performance comparison. Table 7 lists their average (+ std) prediction accuracy on the train and test dataset. Interestingly, SRNet is better than LIME and MAPLE on most classification tasks and only fails on the 'agaricus_lepiota' ($P2$) task. The fail reason is that the dataset 'agaricus_lepiota' has 22 input features, making 1000 points sampled by USDB very sparse in the high-dimensional space. However, SRNet still shows better or competitive accuracy in the other four classification tasks than LIME and MAPLE.

Although LIME and MAPLE have good performance in explaining local classification samples, their decision boundaries cannot approximate the MLP's decision boundary. The reason is that they only leverage a linear model to explain several local samples, as shown in red lines and purple lines in Figure 11. So, once the MLP's decision boundary is a complex curve at some local samples, the line generated by LIME or MAPLE cannot approximate it. In contrast, SRNet can approximate the complex decision boundary of MLP on all samples. Therefore, SRNet is more suitable for explaining NN on the classification task than LIME and MAPLE.

Dataset	$O^s(x)$
$P0$	$Pr0 = -14.16 - \frac{0.00043x_6 - 0.00043x_7 - 0.00043x_9 + 0.42}{-0.0025x_6 + 0.0025x_7 + 0.0025x_9 + 0.489}$ $Pr1 = 14.57 + \frac{0.00047x_6 - 0.00047x_7 - 0.00047x_9 + 0.46}{-0.0025x_6 + 0.0025x_7 + 0.0025x_9 + 0.49}$ **(8.05e-03)**
$P2$	$Pr0 = 0.66x_{10} + 0.66x_{20} - 0.66x_8 - 0.66x_9 - 1.57$ $Pr1 = -0.53x_{10} - 0.53x_{20} + 0.53x_8 + 0.53x_9 + 1.17$ **(3.04e-01)**

Table 6: The mathematical expressions of the whole NN.

Dataset	Method	Train	Test
$P0$(adult)	LIME	84.47 ± 2.3%	71.33 ± 1.78%
	MAPLE	98.47 ± 1.51%	92.11 ± 0.80%
	SRNet	**100 ± 0%**	**100 ± 0%**
$P1$(analcatdata_aids)	LIME	85.18 ± 0.11%	90.89 ± 0.62%
	MAPLE	**93.78 ± 0.23%**	**100 ± 0%**
	SRNet	91.89 ± 8.11%	100 ± 0%
$P2$(agaricus_lepiota)	LIME	88.28 ± 0.16%	93.81 ± 0.30%
	MAPLE	**99.22 ± 0.04%**	**98.16 ± 0.12%**
	SRNet	75.82 ± 0%	75.60 ± 0%
$P3$(breast)	LIME	100 ± 0%	100 ± 0%
	MAPLE	100 ± 0%	100 ± 0%
	SRNet	100 ± 0%	100 ± 0%
$P4$(car)	LIME	100 ± 0%	100 ± 0%
	MAPLE	100 ± 0%	100 ± 0%
	SRNet	100 ± 0%	100 ± 0%

Table 7: LIME vs MAPLE vs SRNet.

6 CONCLUSION

This paper proposes a new evolutionary algorithm called SRNet to address the NN's black box problem. SRNet leverages MNNCGP-ES to find the mathematical expressions that can represent each NN layer's hidden semantics. The combination of every layer's expression represents the whole NN. Compared with the models found by LIME and MAPLE, the mathematical expression provided by SRNet is closer to the NNs in the interpolated domain. The experiment also shows the SRNet models trend to approximate the real data model that used trains NN. The close alignment of SRNet with the real model and its explicit mathematical expression can be used to facilitate the explanation of NN prediction behaviours, such as regression and classification.

Although SRNet demonstrated great potential to explain NN, it needs future work to overcome the two challenges: First, expression complexity with more layers in NNs. As the number of layers in a NN increases, the length of final overall expression explodes, making its poor readability. In future work, we would like to add an appropriate "readability" item to the objective loss function penalizing the complexity of the final expressions. Second, invalidation to high dimensions. If the number of neurons in each network layers becomes larger, the MNNCGP-ES algorithm would be time-consuming and hard to converge. In future work, we focus on improving the SRNet encoding method to speed the evolution process.

REFERENCES

[1] Francesco Bodria, Fosca Giannotti, Riccardo Guidotti, Francesca Naretto, Dino Pedreschi, and Salvatore Rinzivillo. 2021. Benchmarking and survey of explanation methods for black box models. *arXiv preprint arXiv:2102.13076* (2021).

[2] Markus F Brameier and Wolfgang Banzhaf. 2007. *Linear genetic programming*. Springer Science & Business Media.

[3] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. 2019. Exploring neural networks with activation atlases. *Distill*. (2019).

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[5] Benjamin P Evans, Bing Xue, and Mengjie Zhang. 2019. What's inside the black-box? a genetic programming method for interpreting complex machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1012–1020.

[6] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard, and Stefano Soatto. 2018. Empirical study of the topology and geometry of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3762–3770.

[7] Candida Ferreira. 2001. Gene expression programming: a new adaptive algorithm for solving problems. *arXiv preprint cs/0102027* (2001).

[8] Leonardo Augusto Ferreira, Frederico Gadelha Guimarães, and Rodrigo Silva. 2020. Applying genetic programming to improve interpretability in machine learning models. In *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[9] Ruth Fong and Andrea Vedaldi. 2018. Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8730–8738.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[11] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–42.

[12] Thomas Jansen. 2013. *Analyzing evolutionary algorithms: The computer science perspective*. Springer Science & Business Media.

[13] John R Koza. 1992. *Genetic Programming II, Automatic Discovery of Reusable Subprograms*. MIT Press, Cambridge, MA.

[14] Yu Li, Lizhong Ding, and Xin Gao. 2018. On the decision boundary of deep neural networks. *arXiv preprint arXiv:1808.05385* (2018).

[15] Dong C Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical programming* 45, 1 (1989), 503–528.

[16] Qiang Lu, Shuo Zhou, Fan Tao, Jake Luo, and Zhiguang Wang. 2021. Enhancing gene expression programming based on space partition and jump for symbolic regression. *Information Sciences* 547 (2021), 553–567.

[17] Qiang Lu, Shuo Zhou, Fan Tao, and Zhiguang Wang. 2019. Space partition based gene expression programming for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 348–349.

[18] Aravindh Mahendran and Andrea Vedaldi. 2015. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5188–5196.

[19] James McDermott, David R White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, et al. 2012. Genetic programming needs better benchmarks. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 791–798.

[20] Julian Francis Miller. 2019. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines* (Aug. 2019), 1–40.

[21] Julian Francis Miller and Simon L. Harding. 2008. Cartesian Genetic Programming. In *Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '08)*. ACM, New York, NY, USA, 2701–2726.

[22] Julian F. Miller and Peter Thomson. 2000. Cartesian Genetic Programming. In *Genetic Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 121–132.

[23] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 427–436.

[24] Patryk Orzechowski, William La Cava, and Jason H Moore. 2018. Where are we now? A large benchmark study of recent symbolic regression methods. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1183–1190.

[25] Gregory Plumb, Denali Molitor, and Ameet Talwalkar. 2018. Model agnostic supervised local explanations. *arXiv preprint arXiv:1807.02910* (2018).

[26] Ingo Rechenberg. 1978. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*. Springer, 83–114.

[27] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28 (2015), 91–99.

[28] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[29] Victor S Ryaben'kii and Semyon V Tsynkov. 2006. *A theoretical introduction to numerical analysis*. Chapman and Hall/CRC.

[30] Wojciech Samek and Klaus-Robert Müller. 2019. Towards explainable artificial intelligence. In *Explainable AI: interpreting, explaining and visualizing deep learning*. Springer, 5–22.

[31] Michael Schmidt and Hod Lipson. 2009. Distilling Free-Form Natural Laws from Experimental Data. *Science* 324, 5923 (2009), 81–85.

[32] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[33] Silviu-Marian Udrescu and Max Tegmark. 2020. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances* 6, 16 (2020), eaay2631.

[34] James Alfred Walker, Julian Francis Miller, and Rachel Cavill. 2006. A multi-chromosome approach to standard and embedded cartesian genetic programming. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 903–910.

[35] James Alfred Walker, Julian F. Miller, Paul Kaufmann, and Marco Platzner. 2011. *Problem Decomposition in Cartesian Genetic Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 35–99. https://doi.org/10.1007/978-3-642-17310-3_3

[36] Congwen Xu, Qiang Lu, Jake Luo, and Zhiguang Wang. 2021. Adversarial bandit gene expression programming for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 269–270.

[37] Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. 2020. A survey on neural network interpretability. *arXiv preprint arXiv:2012.14261* (2020).