Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis

Hengzhe Zhang¹ · Aimin Zhou¹ · Xin Lin¹

Received: 13 April 2020 / Accepted: 4 July 2020 / Published online: 25 July 2020
© The Author(s) 2020

Abstract

Reinforcement learning based on the deep neural network has attracted much attention and has been widely used in real-world applications. However, the black-box property limits its usage from applying in high-stake areas, such as manufacture and healthcare. To deal with this problem, some researchers resort to the interpretable control policy generation algorithm. The basic idea is to use an interpretable model, such as tree-based genetic programming, to extract policy from other black box modes, such as neural networks. Following this idea, in this paper, we try yet another form of the genetic programming technique, evolutionary feature synthesis, to extract control policy from the neural network. We also propose an evolutionary method to optimize the operator set of the control policy for each specific problem automatically. Moreover, a policy simplification strategy is also introduced. We conduct experiments on four reinforcement learning environments. The experiment results reveal that evolutionary feature synthesis can achieve better performance than tree-based genetic programming to extract policy from the neural network with comparable interpretability.

Keywords Reinforcement learning · Genetic programming · Policy derivation · Explainable machine learning

Introduction

Reinforcement learning [31] has shown its extraordinary performance in computer games [22] and other real-world applications [29]. The neural network is widely used as a dominant model to solve reinforcement learning problems. Generally, we call these methods deep reinforcement learning algorithms, since these algorithms use a deep neural network as the value function approximator or the policy function approximator. Deep q-learning (DQN) [22], double DQN [9], dueling DQN (DDQN) [36] are prestigious algorithms that train a deep neural network for reinforcement learning problems. However, the black-box property of the deep neural network prevents DNN to be directly used in the high-stake scenarios [33]. Therefore, building an interpretable model is essential, and even more priority than interpreting the black-box model in the current machine learning field [27].

There are a variety of ways to build interpretable models [15,34]. Among them, the genetic programming (GP), which builds a symbolic expression as an explainable model through the genetic algorithm, is a promising one. Recently, GP has been applied to reinforcement learning. The idea is to evolve an explainable model to extract the policy from the deep neural network. In [11], an explainable reinforcement learning policy model is built by using the tree-based genetic programming (GP) [24] algorithm. However, it is argued in [11] that it is hard for GP to mimic the behavior of the deep neural network. Therefore, in that paper, DNN is used as a surrogate model, and then GP is used to evolving strategy on that model. This method is far from meeting the requirements of what we desired, deriving a policy from a pre-trained neural network model built by the cutting-edge algorithms such as DQN [22] or Actor-Critic [21] algorithms.

In this paper, we propose a method to extract policy from the pre-trained deep neural network based on the evolution feature synthesis (EFS) algorithm [3]. Our method first generates a behavior sequence by running the neural network in the real environment and then evolve a set of regressors to mimic that behavior sequence as similar as possible. We improve the performance using an evolutionary method to automatically select operators from a large

✉ Aimin Zhou
amzhou@cs.ecnu.edu.cn

¹ Shanghai Key Laboratory of Multidimensional information Processing, School of Computer Science and Technology, East China Normal University, Shanghai, China

set of predefined operators without domain experts involved. Moreover, we introduce a policy simplification approach to balance the tradeoff between interpretability and performance. We conduct comparative experiments on the CartPole [31], the Acrobot [30], the MountainCar [23], and the Industrial Benchmark [10] environments. All these experimental results prove that the policy extracted by EFS outperforms the policy extracted by GP with similar interpretability. Furthermore, the operator optimization strategy has significantly improved final rewards and eventually makes EFS surpass the neural network and common interpretable machine learning algorithms in all environments.

The main contributions of our work are summarized as follows:

- We derive the latent policy from the pre-trained deep neural network by EFS. Our experimental results demonstrate that EFS can achieve comparable performance to the deep neural network.
- We design an operator optimization strategy based on the evolutionary algorithm. This strategy can automatically select appropriate operators from a large set of predefined operators without additional domain knowledge.
- We propose a progressive simplification method to construct a series of control policies based on the best EFS policy model to balance the tradeoff between the model complexity and performance.
- We evaluate our algorithm on the CartPole, the Acrobot, the MountainCar, and the Industrial Benchmark problems. The experimental results are compared with the results of GP, DNN, Linear regression, Decision tree, K-Nearest neighbors (KNN). The experimental results show that our algorithm achieves better performance than GP, DNN, and common interpretable machine learning algorithms with comparable interpretability.

The rest of this paper is organized as follows: the next section introduces the related works of reinforcement learning based on genetic programming algorithms. The following section gives a brief introduction to EFS. The next section presents the details of the EFS-based policy derivation algorithm. The following section presents experimental settings and experimental results of the proposed algorithm. Finally, we conclude this paper with some future works in the last section.

Related work

Reinforcement learning by genetic programming can retrospect to [17], which uses GP to solve the broom balancing problem. Followed by [28], an equation is generated by GP to solve the movable inverted pendulum problem. Sub-

sequently, genetic network programming (GNP) [14] is proposed to surpass GP on the food-collecting problem [12]. Soon after, the performance of GNP is enhanced by Q-Learning [20] and by utilizing the information obtained during tasks [19].

Recently, the value function approximation by GP becomes a hot spot in the evolutionary reinforcement learning domain. Reference [25] introduced a method to get a near-optimal value function on an illustrative Markov decision process (MDP) environment. Lately, [18] proposed an approach to calculate the symbolic value function in a policy iteration way. GP based value function approximation approach has been proven that it can obtain an accurate value function even for a small batch of data [6,7].

The incorporation of the model-based reinforcement learning and the symbolic regression algorithm has been studied in [11]. Reference [11] introduced a method of evolving interpretable policies based on GP. A neural network that builds based on trajectory data is used as a world model to evaluate the fitness value of generated symbolic policies.

In addition to leverage genetic programming to generate interpretable policies, the influence of the smooth property of symbolic approximation function in reinforcement learning has also been studied. References [1] and [2] proposed a symbolic regression method to obtain a smooth v -function from the original value function, and experiment results reveal that policy derived from the smooth value function performs much better than the policy derived from the original value function.

However, to the best of our knowledge, up to now, GP is hard to derive policy from the deep neural network agent [11]. EFS is the first model that exhibits that the genetic programming algorithm can derive policy directly from DNN to achieve comparable performance with stronger interpretability.

Evolutionary feature synthesis

EFS [3] is a novel genetic programming algorithm, which evolves non-linear features to compose a linear model iteratively. The search space of EFS is all possible non-linear features which are composed of variables and operators in a predefined set. The goal of the evolution process is to find a set of non-linear features, which constitute a linear model that can perform well on a specific problem. After a feature initialization stage, the evolution process of EFS comprises three stages, i.e., the feature composition stage, the feature importance evaluation stage, and the feature selection stage.

First, in the initialization stage, each original variable x in the training data X is added to the feature set U . From the perspective the evolutionary computation, the feature set U is also called as a population of features. The original variables

Fig. 1 Evolutionary feature synthesis

are marked by a distinctive mark so that they will not be removed from the population in the following process.

In the feature composition stage, an operator o is randomly selected from the operator set O . If o is a unary operator, such as log or square, then one feature u is selected from the population U and then the operator o is used to compose a new feature $o(u)$. If o is a binary operator, such as $+$ or $*$, then two features u_1, u_2 are selected from the population U , and then the operator o is used to compose a new feature $o(u_1, u_2)$. Suppose the population size is $|P|$, then $|P| - |X|$ features should be generated in this stage because original variables remain unchanged.

In the stage of the feature importance evaluation, the importance of each feature is calculated by a score function g . A typical score function is the f-regression function

$$\text{correlation}(x) = \frac{(x - \bar{x}) * (y - \bar{y})}{\sigma_x * \sigma_y} \quad (1)$$

$$g(x) = \frac{\text{correlation}(x)^2 * (|Y| - 2)}{1 - \text{correlation}(x)^2} \quad (2)$$

where $\bar{x}$ represents the mean value of feature values x , $\bar{y}$ represents the mean value of target values y . σ_x represents the variance of x , and σ_y represents the variance of y . $|Y|$ represents the count of data items.

In the stage of feature selection, features are ranked by the importance value. Then the top $|P|$ features are selected as the basis of the linear model, and other features are discarded.

Finally, in the stage of model construction, a linear model is composed of multiple non-linear features P . The parameters of the linear model are determined by the least square method. The overall process is presented in Fig. 1 and Algorithm 1. More details about Algorithm 1 are referred to [3].

EFS-based policy derivation

In this section, we propose a new policy derivation method based on EFS, extracting the well-performed policy from the deep neural network by evolutionarily constructing a linear model with non-linear features. Formally, there are two essential components of our EFS based reinforcement learning algorithm.

Algorithm 1 Evolutionary features synthesis

```

1:  $P \leftarrow X$  ▷ Feature initialization
2:  $i \leftarrow 0$ 
3: while  $i < N$  do ▷ Counter
4:    $j \leftarrow \text{population size} - |X|$  ▷ Old features inheritance
5:    $P' \leftarrow P$  ▷ Feature composition
6:   while  $j > 0$  do
7:      $o \leftarrow \text{random choice from } O$ 
8:     if  $o$  is unary operator then
9:        $a \leftarrow \text{random choice from } P$ 
10:       $\text{new\_feature} \leftarrow o(a)$ 
11:     else if  $o$  is binary operator then
12:        $a, b \leftarrow \text{random choice from } P$ 
13:        $\text{new\_feature} \leftarrow o(a, b)$ 
14:     end if
15:      $P' \leftarrow P' \cup \{\text{new\_feature}\}$ 
16:      $j \leftarrow j - 1$ 
17:   end while
18:    $P \leftarrow X$  ▷ Population resetting
19:   while  $|P| < \text{population size}$  do ▷ Feature selection
20:      $x \leftarrow \arg \max_{x \in (P' - P)} g(x)$ 
21:      $P \leftarrow P \cup \{x\}$ 
22:   end while
23:    $\text{model} \leftarrow \text{model construction}(P)$ 
24:    $i \leftarrow i + 1$ 
25: end while

```

- Deep neural network: DNN is used to approximate the optimal value function [22] or the policy function [32] of an environment. Through DNN, we can generate behavior sequences S , which comprise states s and actions a by simulating in the real environment. Then, EFS can utilize these behavior sequences $(s, a) \in S$ to generate an interpretable policy model.
- EFS model set: EFS model set is composed of several EFS models $m \in M$. Each EFS model is to mimic a part of the behavior sequences S to decide which action a should be chosen at a specific state s . The size of the model set $|M|$ is equal to the size of the action space $|A|$. The details of the training and the prediction process are described in the following section.

The policy derivation process is composed of five sub-processes.

- Data preparation: In the data preparation process, we use a pre-trained neural network to generate optimal decision schemes in the reinforcement learning environment under different circumstances.
- Operator optimization: Operator optimization is an optional process, which pre-select an optimal subset of operators by the genetic algorithm based on small-scale experiments.
- EFS evolution: In the EFS evolution process, we evolve a set of EFS regressors based on the training data obtained in the preparation stage.

Fig. 2 The workflow of the EFS based reinforcement learning

- EFS ensembling: After getting a set of regressors, we aggregate these regressors as a single model based on a simple decision function.
- Policy simplification: Policy simplification is also an optional operation, which can balance the trade-off between complexity and accuracy.

In the following sections, we first present the essential process of our algorithm, and then we introduce two optional strategies to achieve a better result. The workflow of our algorithm is presented in Fig. 2 and Algorithm 2.

Data preparation

In the behavior sequences preparation stage, the most convenient way is to leverage the pre-trained neural network model to generate behavior sequences in the real environment. First, it is necessary to emphasize that all of environments following the Markov property, i.e. $s, r \leftarrow \text{environment}(s, a)$. Then, we can use the neural network to generate an action a at each state s

$$a \leftarrow \arg \max_a \text{DNN}(s, a) \quad (3)$$

where DNN represents the neural network. In Q-Learning, $\text{DNN}(s, a)$ generates a vector that represents the estimated reward when performing different actions a at the current state s . The action corresponding to the best estimate reward is selected as the real action. Each state-action pair is stored in the behavior sequences S , and the behavior sequences generation should be continuously executed until the predefined e episode number is met. Figure 3 presents a schematic of the data preparation process based on the neural network. In Algorithm 2, lines 1–12 presents the details of the data preparation process.

Fig. 3 An illustration of data preparation

Algorithm 2 EFS based policy derivation

```

1:  $D \leftarrow \{\}$  ▷ Dataset initialization
2:  $x \leftarrow 0$ 
3: while  $x < e$  do ▷ Data generation
4:    $s \leftarrow \text{environment initialization}$ 
5:   while  $s \neq \text{Done}$  do
6:      $a \leftarrow \arg \max \text{DNN}(s, a)$  ▷ Action prediction
7:      $D_a \leftarrow D_a \cup \{(s, 1)\}$ 
8:      $D_{A-\{a\}} \leftarrow D_{A-\{a\}} \cup \{(s, 0)\}$ 
9:      $s, r \leftarrow \text{environment}(s, a)$ 
10:  end while
11:   $x \leftarrow x + 1$ 
12: end while
13:  $f \leftarrow \{\}$ 
14: for  $D_i$  in  $D$  do
15:    $f_i \leftarrow \text{EFS evolution}(D_i)$  ▷ EFS evolution
16:    $f \leftarrow f \cup \{f_i\}$  ▷ Controller integration
17: end for
18:  $s \leftarrow \text{environment initialization}$  ▷ Policy validation
19: while  $s \neq \text{Done}$  do ▷ EFS ensembling
20:    $v \leftarrow \{\}$ 
21:   for  $f_i$  in  $f$  do
22:      $v \leftarrow v \cup \{f_i(s)\}$ 
23:   end for
24:    $a \leftarrow \arg \max v$ 
25:    $s, r \leftarrow \text{environment}(a)$ 
26: end while
  
```

EFS evolution

In the EFS evolution process, the target is to evolve a set of regressors M to mimic the behavior sequences as well as possible. In other words, the evolution target is to minimize the loss value of each EFS regressor $f_a \in M$ on the behavior sequences

$$\text{Loss}(m) = \sum_{i=1}^{|S|} (I_a(s_i) - f_a(s_i))^2, \quad (4)$$

where $f_a(s_i)$ is the predicted value of the EFS regressor f_a . $I_a(s_i)$ is an indicator function which indicates whether the action a has been chosen by the neural network under state s_i , similar to the one-hot encoding scheme.

$$I_a(s_i) = \begin{cases} 0 & a \neq a_{s_i} \\ 1 & a = a_{s_i} \end{cases} \quad (5)$$

Fig. 4 An illustration of feature selection

The evolution process of each EFS regressor in the regressor set is the same as previously mentioned in “Evolutionary feature synthesis” section, consisting of three major steps: feature generation, feature evaluation, feature selection.

- Feature generation: New features are generated by randomly composing old features and new operators in this stage.
- Feature evaluation: The importance of each feature is calculated by the f-regression or the mutual information algorithm in this stage.
- Feature selection: Features are ranked according to the feature importance after the feature evaluation process. Top-ranked features are selected to construct a new linear model. Model parameters are fitted by the least square method.

Figure 4 depicts the feature selection process based on the p value.

EFS ensembling

In a real environment, for a specific state s , action a should be generated by the policy model. For EFS, the action chosen by EFS is corresponding to the action label of the regressor, which has the most significant predicted value. Formally, action a is defined by the following formula.

$$a = \arg \max_a f_a(s) \quad (6)$$

The action generation process is presented in the Fig. 5. Due to the action A has the largest predicted value, the action A is chosen as the real action at the current state.

Operator optimization

Domain knowledge is useful in the reinforcement learning domain. Some specific control operators may be more expressive than other operators in a specific problem. Therefore, incorporating domain knowledge to select operators can

Fig. 5 An illustration of action generation

dramatically reduce the search space and consequently obtain much better results.

However, operator selection based on the domain knowledge is heavily dependent on the domain experts. In many cases, domain experts are lacked. Sometimes, even with domain experts exist, the well-performed operator may be counter-intuitive. Therefore, automatically selecting the best operators are preferred in the operator optimization process.

In this paper, we use a binary encoding based genetic algorithm to select the best operators for a specific problem. First, in the initialization stage, we predefine an operator set O contains all potential useful operators, such as $O = \{+, -, *, /\}$. Then, binary chromosomes P with the length of $|O|$ each chromosome are created, and $|O|$ is the number of operators in the operator set O . Each chromosome $p \in P$ represents an operator selection scheme $O' \subseteq O$.

In the reproduction process, new chromosomes are generated by uniform random two-point crossover and one-point mutation [13]. In the evaluation process, for each operator selection scheme O' , an EFS with less training iterations and evaluating episodes is used here to assessing the performance of the operator subset. In our experiments, we use two-fifths of the normal training iterations and one-fifth of the normal evaluating episodes as parameters. After that, the fitness $T_{O'}$ is determined by the score of the final regressor on the specific environment. In the selection process, chromosomes are selected by the tournament selection algorithm [13]. The overall process is repeated until N iterations have reached. Algorithm 3 presents the whole process of the operator optimization algorithm.

Policy simplification

In the interpretable machine learning domain, multiobjective models are preferred by practitioners to achieve the balance between the tradeoff of complexity and performance. Due to the unique nature of the EFS algorithm, with the policy simplification method, a series of models with different complexity and performance can be obtained from the best policy model trivially.

Algorithm 3 Operator optimization

```

1:  $P \leftarrow$  population initialization
2:  $i \leftarrow 0$ 
3: while  $i < N$  do ▷ Evolution
4:    $P \leftarrow$  mutation-crossover( $P$ )
5:   for  $p$  in  $P$  do
6:      $O' \leftarrow$  chromosome decode( $p$ )
7:      $T_p \leftarrow$  evaluation( $O'$ ) ▷ Operators evaluation
8:   end for
9:    $P \leftarrow$  tournament-selection( $P, T$ )
10:   $i \leftarrow i + 1$ 
11: end while

```

Suppose that the number of features $|F|$ is the complexity indicator of the model. The ultimate goal of the policy simplification is to derive a series of policy models with a different number of features and reasonable rewards. Therefore, for the best model evolved by the aforementioned algorithm, we can rank the importance of those features in that model by the p-value. Then progressively integrating those features to compose a new feature set $F' \subset F$ by the rank value. Finally, a series of feature sets compose a series of linear models R . It should be noted that for each model f , the model parameters are determined by the training data D through the least square method. Formally, the whole process of policy simplification is presented in Algorithm 4.

Algorithm 4 Policy simplification

```

1:  $F \leftarrow$  sort by p-value ( $F$ )
2:  $R \leftarrow \{\}$  ▷ Regressor set
3:  $i \leftarrow 0$ 
4: while  $i < |F|$  do
5:    $F' \leftarrow F[:i]$  ▷ Get the top  $i$  features
6:    $f \leftarrow$  build model( $F'$ )
7:    $f \leftarrow$  model fitting( $f, D$ )
8:    $R \leftarrow R \cup \{f\}$ 
9:    $i \leftarrow i + 1$ 
10: end while

```

Experiment

Experiment settings

- Baseline method: GP, Neural network, and common interpretable machine learning algorithms are used as baseline methods. The GP method derives the policy form the neural network by using the tree-based genetic programming algorithm [17]. The neural network is used as the baseline method because the derived policy can even surpass the original neural network in the following experiments. For the interpretable machine learning algorithms, we use linear regression, decision tree, and K-nearest neighbor algorithms in the follow-

ing experiments. In the machine learning community, these algorithms are widely considered as the high interpretable models [5].

- Benchmark datasets: We compare various methods on the CartPole environment, the Acrobot environment, the MountainCar environment, and the Industrial Benchmark environment.
- Evaluation metrics: Mean square error (MSE) and real environment reward are used to assess the performance of different algorithms. The mean square error is defined as

$$\text{MSE}(f) = \frac{\sum_{a \in A} (f_a(x) - y)^2}{|S||A|}, \quad (7)$$

where $|S|$ represents the size of training data, x represents the state data $s \in S$, y represents $I_a(s)$. The real environment reward is defined by the specific problem.

- Complexity metrics: Tree height and weighted number of leaf nodes are used as complexity metrics to appraise the interpretability of GP and EFS. Similar to the previous study [35], we prefer a flatter tree with fewer layers than a deeper tree. Therefore, the weighted number of leaf nodes is calculated as $\sum_{l \in L} \text{depth}(l)$, where $\text{depth}(l)$ represents the depth of the leaf node l . Due to each model is composed of several regressors, we sum up the tree height and weighted number of leaf nodes as the complexity value of a specific model.

Experiment environment

CartPole

In the cart-pole problem [31], an un-actuated pole is connected to a car among a frictionless track. The goal is to prevent the pole fall over from the cart. In each step, two actions {left, right} can be chosen to apply a force from the left of the cart or from the right of the cart. The reward $r(s)$ is 1 at each step until the game over. In our experiment, 1001 trials are selected as the upper limit trials because some interesting phenomenons will emerge when longer trials are used.

Acrobot

In the acrobot problem [30], two links are connected by a joint. The target of this problem is to swing the end of the lower link to reach a specific height. In each step, three actions {left, neutral, right} can be chosen to apply a force to the joint. The reward $r(s)$ is -1 at each step until the joint reaches target height.

Table 1 Experiment parameters of evolution algorithms

	GP	EFS
Number of experiments	15	15
Iteration of evolution	50	50
Operator optimization	True	True
Game episodes	100	100
Feature number	–	10
Feature size	–	8
Max height	25	–

MountainCar

In the mountain car problem [23], an underpowered vehicle is situated in a valley at the position $p_0 \in (-0.6, -0.4)$ with speed $v_0 = 0$. The destination is located at $p_d = 0.5$, and the target of the problem is to reach the target location with the lowest cost. In each step, three actions {left, neutral, right} can be chosen to execute, and the reward $r(s)$ at state s is -1 until it reaches the target location.

$$r(s) = \begin{cases} -1 & s < 0.5 \\ 0 & s \geq 0.5 \end{cases} \quad (8)$$

Industrial benchmark

In the Industrial Benchmark problem [10], we control a simulation environment through 27 kinds of operations. The target of this problem is to minimize the cost, which is defined as three times the fatigue f , plus the consumption c , i.e., $\text{cost} = 3 \times f + c$. Each episode is composed of 1000 steps, the final reward is calculated as the sum of costs in the episode. The details of the simulation environment are presented in [10].

Parameter settings

Due to the randomness of the evolutionary algorithm, experiments of GP and EFS are independently repeated 30 times, and at each time, the final model plays 100 episodes of games consecutively. For GP and EFS, 50 iterations of evolution are performed. The max height of GP is limited as 25. The number of constructed features of EFS is restricted as 10, and the max size of each feature is set as 8. The f-regression is used as the feature importance evaluation approach. The details of the experiment parameters are presented in Table 1.

We try to cover a broad range of arithmetic operators to describe the control policy in the experiment. operator $\in \{+, -, *, /, \text{negative}, \%, \text{max}, \text{min}, \text{logistic}(x), \text{log}_2(x), \ln(x), \text{log}_{10}(x), \sqrt{|x|}, x^3, x^2, x^{1/3}, \sin(x), \cos(x), |x|\}$ operation optimization has been performed. For the cart-pole

problem, the selected operators are $\{+, -, *, /, \%, \text{logistic}(x), \ln(x), \text{log}_2(x), \text{log}_{10}(x), x^2\}$. For the mountain car problem, the selected operators are $\{+, -, *, /, \ln(x), \text{log}_2(x), \sqrt{|x|}, \sqrt[3]{x}, \sin(x)\}$. For the acrobot problem, the selected operators are $\{+, -, *, /, \text{max}, \ln(x), x^2, \sqrt{|x|}, \sin(x), \cos(x)\}$. For the industrial benchmark problem, the selected operators are $\{+, -, *, /, \%, \text{min}, \text{log}_2(x), x^2, \sin(x), |x|\}$. In the experiment, the standard logistic function is used as the logistic function, which is defined below.

$$\text{logistic}(x) = \frac{1}{1 + e^{-x}} \quad (9)$$

Due to some operators have restrictions on the input values, we take some protective measures to avoid accident results. For logarithmic operators, we wrap these operators by the following function.

$$\log(x) \leftarrow \begin{cases} 0 & x < 10^{-6} \\ \log(|x|) & x \geq 10^{-6} \end{cases} \quad (10)$$

For the division operator, we avoid division by zero exception by using the following function. The modulo operator is also wrapping by a similar function.

$$\text{division}(x, y) \leftarrow \begin{cases} \text{division}(x, 1) & x < 10^{-6} \\ \text{division}(x, y) & x \geq 10^{-6} \end{cases} \quad (11)$$

The neural networks are pre-trained by Q-learning [22] and policy gradient [32]. For the mountain car and the cart-pole problem, a network with two fully connected hidden layers with 256 neurons is used. For the acrobot problem, a network with one hidden layer with 32 neurons is used. For the industrial benchmark problem, we use 5 hidden layers with 64 neurons, followed by a layer with 32 neurons. For all problems, the rectified linear unit (ReLU) [8] is used as the activation function, and Adam optimizer is used as the learning rate optimizer [16]. For the mountain car problem and the cartpole problem, the neural network trained until the target reward has reached with 100 consecutive trials. For the acrobot problem and the industrial benchmark problem, the neural network trained until the maximum iteration number is reached. The detailed parameters are presented in Table 2.

For common interpretable machine learning algorithms, we use the default parameters implemented in the Scikit-learn package [26]. The default parameters do not restrict the height of the decision tree, and the nearest neighbors of the K-nearest neighbor algorithm is set as five.

Experimental result

Figure 6 presents the training error of four problems. The experimental results show that the training error of EFS is

Table 2 Experiment parameters of neural networks

	Cartpole	MountainCar	Acrobot	IndustrialBenchmark
Algorithm	Q-learning	Q-learning	Policy-gradient	Q-learning
Hidden layer	(256, 256)	(256, 256)	(32)	(64, 64, 64, 64, 64, 32)
Learning rate	0.0001	0.0001	0.001	0.0003
Decay factor	0.99	0.99	1	0.99
Batch size	64	64	64	–
Target reward	≥ 195	≥ -110	–	–
Maximum iteration	2000	2000	5000	50000

Table 3 Rewards of different algorithms of four problems

	Cartpole	Acrobot	MountainCar	IndustrialBenchmark
EFS	982.642	– 81.572	– 105.054	– 1994.72
GP	376.405 (–)	– 525.27 (–)	– 143.467 (–)	– 2175.653 (–)
LinearRegression	844.786 (–)	– 784.058 (–)	– 188.665 (–)	– 2097.91 (–)
DecisionTree	238.14 (–)	– 87.955 (–)	– 106.909 (–)	– 2107.087 (–)
KNN	286.465 (–)	– 89.584 (–)	– 107.191 (–)	– 2107.951 (–)
NeuralNetwork	303.455 (–)	– 88.81 (–)	– 106.606 (–)	– 2103.837 (–)

The bold values indicate the best ones

“–” in parenthesis indicates the reward is smaller than that obtained by EFS at 95% significance level by a wilcoxon rank sum test

Table 4 Rewards of different operator optimization strategies of four problems

Optimization	Cartpole	Acrobot	MountainCar	IndustrialBenchmark
With optimization	982.642	– 81.572	– 105.054	– 1994.718
Without optimization	268.772 (–)	– 861.282 (–)	– 117.352 (–)	– 2030.8 (–)

The bold values indicate the best ones

“–” in parenthesis indicates the reward is smaller than that obtained by optimized EFS at 95% significance level by a wilcoxon rank sum test

Fig. 6 Training error of each algorithm on four problems **Fig. 7** Real environment reward of each algorithm on four problems

Fig. 8 Training error of each operator optimization strategy on four problems

Fig. 9 Real environment reward of each operator optimization strategy on four problems

Fig. 10 Real environment rewards of simplified EFS models on four problems

Table 5 Rewards of simplified efs models of four problems

Complexity	Cartpole	Acrobot	MountainCar	IndustrialBenchmark
0	838.6883	− 770.961	− 195.407	− 2094.63
1	832.0437	− 775.208	− 119.027	− 2115.07
2	610.118	− 1000.14	− 119.007	− 2231.56
3	609.8793	− 992.318	− 118.175	− 2208.93
4	614.0813	− 84.2313	− 107.378	− 2141.03
5	507.805	− 87.3803	− 106.857	− 2106.96
6	500.565	− 80.7847	− 108.048	− 2213.42
7	508.3603	− 81.0543	− 116.686	− 2227.26
8	939.5413	− 81.0843	− 108.468	− 2178.19
9	931.3643	− 80.3657	− 108.012	− 2032.13
10	981.6653	− 81.0917	− 104.921	− 1992.19

The bold values indicate the best ones

Table 6 Depth of different algorithms of four problems

Algorithm	Cartpole	Acrobot	MountainCar	IndustrialBenchmark
EFS	8	14.5	23	25
GP	18.767 (+)	16.167 (~)	34.9 (+)	48.533 (+)

The bold values indicate the best ones

“+” or “~” in parenthesis indicates the result is greater than or similar with that obtained by EFS at 95% significance level by a wilcoxon rank sum test

Table 7 Weighted number of leaf nodes of different algorithms of four problems

Algorithm	Cartpole	Acrobot	MountainCar	IndustrialBenchmark
EFS	120	214	295	421
GP	415.1 (+)	248.233 (~)	597.267 (~)	703.767 (~)

The bold values indicate the best ones

“+” or “~” in parenthesis indicates the result is greater than or similar with that obtained by EFS at 95% significance level by a wilcoxon rank sum test

Table 8 Training time (seconds) of different algorithms of four problems

Algorithm	Cartpole	Acrobot	MountainCar	IndustrialBenchmark
EFS	179.529	50.481	20.877	1014.458
GP	49.947	3.181	7.795	285.678

The bold values indicate the best ones

similar to GP. However, the real environment rewards of four problems presented in Fig. 7 show that EFS outperforms GP significantly. It should be noted that the performance of EFS exceeds our expectations, which outperforms the neural network on all problems, this may be caused by that operator selection has selected proper operators to describe the control policy of these problems. The numerical results of different algorithms are presented in Table 3.

From the numerical results, we find that EFS is not only superior to genetic programming but also surpass traditional interpretable machine learning models. For the simplest model, linear regression, we find that it is hard to mimic the behavior of the deep neural network and performs poorly on the Acrobot and MountainCar problem. However, the linear regression model performs well on the Cartpole problem. The reason behind this phenomenon is that the low complexity of the linear model may lead it more robust in the real environment. For the decision tree and the KNN, we find that the performances of the two algorithms are very similar to each other. These two models are accurately memorized the behavior of the neural network. However, they are confined by the performance of the neural network. In contrast to these algorithms, EFS significantly outperforms these algorithms by using expressive features.

Influence of operator optimization

Figure 8 presents the training error of four problems with different operator optimization strategies. Figure 9 presents the real environment rewards of four problems with different operator optimization strategies. From these results, we find that even though the operator optimization reduce the fitting accuracy, especially on the CartPole problem. The final rewards of the policy composed by optimized operators outperform rewards of the policy without optimized operators. The discrepancy between the fitting accuracy and final rewards shows the importance of suitable operators for different problems. In some cases, even if some inappropriate

operators achieve good accuracy in the training process, it may not generalize well. The reason might be that it does not capture the true pattern of the control rules. The numerical results of two different strategies are presented in Table 4.

Result of policy simplification

The results of policy simplification are depicted in Fig. 10, and the numerical results are presented in Table 5. The complexity represents the number of evolved features in the linear model, and complexity with 0 means that only original features are contained in the model. Four problems exhibit different characteristics. For the cartpole and the industrial benchmark problem, lower complexity models attain much reward than midsize models. For the mountain car problem, the original linear model (complexity 0) needs a mean cost of 195 to solve this task. However, adding only one evolved feature, then the model can achieve a reasonable result with an average reward of −119. The acrobot problem behaves similar to the mountain car problem, a linear model with at least four evolved features can achieve a reasonable result.

Structural complexity analysis

Tables 6 and 7 presents the depth and weighted number of leaf nodes of different algorithms. It is necessary to emphasize that the complexity of a specific model is the sum of the complexity of regressors in the model. Table 6 suggests that policy derived by EFS is significantly shallower than GP in the Cartpole, Acrobot, and IndustrialBenchmark environments. Table 7 reveals that the weighted number of leaf nodes of EFS is significantly lesser than GP in the Cartpole environment and similar to GP in other environments. From these results, we can conclude that the explainability of EFS is comparable to GP.

Fig. 11 An EFS controller of the industrial benchmark problem

Time complexity analysis

Table 8 presents the training time of different algorithms of four problems. Experimental results show that EFS requires more time than GP on all problems. The time used by EFS is 2.68–15.87 times of the running time used by GP. In the future, the more efficient feature importance evaluation method is worth to be explored to achieve better performance.

Explainable result

We obtain a controller from the experiment mentioned above. For brevity, the value is rounded to three decimal places. The control function is described below.

$$\begin{aligned}
 f(x) = & -1.613 \times x_3 + 0.039 \times (x_1 \% x_3) - 0.044 \times x_1 \\
 & + 0.086 \times (x_1 \times \log_{10}(x_0)) - 0.099 \times \log_{10}(x_1) \\
 & - 0.457 \times x_2 - 0.02 \times \log_2(x_3) - 1.407 \\
 & \times (x_1 \% x_2) + 1.037 \times ((x_1 \% x_2) \% x_0) \\
 & + 0.564 \times (\log_{10}(x_1) \% (x_1 \% x_2)) \\
 & - 0.039 \times \log_{10}(x_0) - 1.013 \times x_0 + 0.322
 \end{aligned}$$

$X = [x_0, x_1, x_2, x_3]$ represents the state value, and the corresponding action a is determined by the result of $f(X)$

$$a = \begin{cases} 1 & f(X) \leq 0.5 \\ 0 & f(X) > 0.5 \end{cases} \quad (12)$$

We convert this function into a Python code fragment and evaluate it on the cartpole environment implemented by OpenAI-Gym [4]. We conduct 100 consecutive experiments, and it achieves an average reward of 993. This experimental result is similar to the previous result, and the decimal round operation has not damaged the function performance.

Besides the controller example of the Cartpole problem, we also present an EFS controller for the industrial benchmark problem in Fig. 11, which obtains an average reward of −1981 in 100 consecutive trials. We only show five actions due to only those five actions are used by the trained neural network, and the training data generated by the neural network only contain those five actions. Therefore, the EFS controller will not try to use other actions.

Conclusion

In this paper, we propose a novel method to derive interpretable policy from the neural network by evolutionary feature synthesis (EFS). We first propose a framework derives interpretable policy from the behavior data of the neural network. Then an EA based operator optimization is designed to automatically select the best operators for a specific problem to reduce the search space and enhance the performance. A policy simplification algorithm is also proposed to balance the tradeoff between the performance and the interpretability. We study the performance of EFS based on four control problems. The experimental results show that the policy derived by EFS outperforms the policy derived by tree-

based genetic programming (GP) and common interpretable machine learning algorithms, with comparable interpretability to the genetic programming algorithm.

For future work, it is deserved to investigate the performance of policy derived by EFS on more complicated problems, such as computer games. It is also worth to explore that combining more sophisticated domain operators to achieve better performance.

Acknowledgements This work is supported by the Science and Technology Commission of Shanghai Municipality (No. 19511120600) and the National Nature Science Foundation of China (Nos. 61773296, 61673180).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Alibekov E, Kubalik J, Babuska R (2016) Symbolic method for deriving policy in reinforcement learning. In: 55th IEEE conference on decision and control, CDC 2016. IEEE, pp 2789–2795
- Alibekov E, Kubalik J, Babuska R (2018) Policy derivation methods for critic-only reinforcement learning in continuous spaces. *Eng Appl Artif Intell* 69:178–187
- Arnaldo I, O'Reilly U, Veeramachaneni K (2015) Building predictive models via feature synthesis. In: Proceedings of the genetic and evolutionary computation conference, GECCO 2015. ACM, pp 983–990
- Brockman G, Cheung V, Pettersson L et al (2016) Openai gym. *CoRR*. [arXiv:1606.01540](https://arxiv.org/abs/1606.01540)
- Cayamcela MEM, Lee H, Lim W (2019) Machine learning for 5g/b5g mobile and wireless communications: potential, limitations, and future directions. *IEEE Access* 7:137184–137206
- Derner E, Kubalik J, Babuska R (2018) Data-driven construction of symbolic process models for reinforcement learning. In: 2018 IEEE international conference on robotics and automation, ICRA 2018. IEEE, pp 1–8
- Derner E, Kubalik J, Babuska R (2018) Reinforcement learning with symbolic input-output models. In: 2018 IEEE/RSJ international conference on intelligent robots and systems, IROS 2018. IEEE, pp 3004–3009
- Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics, AISTATS 2011, JMLR proceedings, vol 15, pp 315–323. JMLR.org
- van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the thirtieth AAAI conference on artificial intelligence, AAAI 2016. AAAI Press, pp 2094–2100
- Hein D, Depeweg S, Tokic M et al (2017) A benchmark environment motivated by industrial control problems. In: 2017 IEEE symposium series on computational intelligence, SSCI 2017. IEEE, pp 1–8
- Hein D, Udluft S, Runkler TA (2018) Interpretable policies for reinforcement learning by genetic programming. *Eng Appl Artif Intell* 76:158–169
- Hirasawa K, Okubo M, Katagiri H, Hu J, Murata J (2001) Comparison between genetic network programming (gnp) and genetic programming (gp). In: Proceedings of the 2001 congress on evolutionary computation, CEC 2001, vol 2. IEEE, pp 1276–1282
- Hussain A, Muhammad YS (2019) Trade-off between exploration and exploitation with genetic algorithm using a novel selection operator. In: *Complex & intelligent systems*, pp 1–14
- Katagiri H, Hirasawa K, Hu J (2000) Genetic network programming—application to intelligent agents. In: Proceedings of the IEEE international conference on systems, man & cybernetics: “cybernetics evolving to systems, humans, organizations, and their complex interactions”, SMC 2000. IEEE, pp 3829–3834
- Kim S, Ha J, Kim H, Zhang B (2019) Bayesian evolutionary hypernetworks for interpretable learning from high-dimensional data. *Appl Soft Comput* 81:104577. <https://doi.org/10.1016/j.asoc.2019.05.004>
- Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: 3rd international conference on learning representations, ICLR 2015
- Koza JR, Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection, vol 1. MIT Press, Cambridge
- Kubalik J, Zeglitz J, Derner E, Babuska R (2019) Symbolic regression methods for reinforcement learning. *CoRR*. [arXiv:1903.09688](https://arxiv.org/abs/1903.09688)
- Mabu S, Hirasawa K, Hu J (2004) Genetic network programming with reinforcement learning and its performance evaluation. In: Genetic and evolutionary computation—GECCO 2004, Lecture notes in computer science, vol 3103. Springer, pp 710–711
- Mabu S, Hirasawa K, Hu J, Murata J (2002) Online learning of genetic network programming (gnp). In: Proceedings of the 2002 congress on evolutionary computation, CEC 2002, vol 1. IEEE, pp 321–326
- Mnih V, Badia AP, Mirza M et al (2016) Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd international conference on machine learning, ICML 2016, JMLR workshop and conference proceedings, vol 48, pp 1928–1937. JMLR.org
- Mnih V, Kavukcuoglu K, Silver D et al (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529
- Moore AW (1990) Efficient memory-based learning for robot control. Tech. rep., University of Cambridge, Computer Laboratory
- Nguyen S, Mei Y, Zhang M (2017) Genetic programming for production scheduling: a survey with a unified framework. *Complex Intell Syst* 3(1):41–66
- Onderwater M, Bhulai S, van der Mei R (2016) Value function discovery in markov decision processes with evolutionary algorithms. *IEEE Trans Syst Man Cybern Syst* 46(9):1190–1201
- Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: machine learning in python. *J Mach Learn Res* 12(Oct):2825–2830
- Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat Mach Intell* 1(5):206–215
- Shimooka H, Fujimoto Y (1998) Generating equations with genetic programming for control of a movable inverted pendulum. In: Simulated evolution and learning, second Asia-Pacific conference on simulated evolution and learning, SEAL 98, Lecture notes in computer science, vol 1585. Springer, pp 179–186
- Silver D, Schrittwieser J, Simonyan K et al (2017) Mastering the game of go without human knowledge. *Nature* 550(7676):354

30. Sutton RS (1995) Generalization in reinforcement learning: successful examples using sparse coarse coding. In: *Advances in neural information processing systems*, vol 8, NIPS 1995. MIT Press, pp 1038–1044
31. Sutton RS, Barto AG (2018) *Reinforcement learning: an introduction*. MIT Press, Cambridge
32. Sutton RS, McAllester DA, Singh SP, Mansour Y (1999) Policy gradient methods for reinforcement learning with function approximation. In: *Advances in neural information processing systems*, vol 12, NIPS 1999. The MIT Press, pp 1057–1063
33. Vellido A (2019) The importance of interpretability and visualization in machine learning for applications in medicine and health care. *Neural Comput Appl* 1–15. <https://doi.org/10.1007/s00521-019-04051-w>
34. Verma L, Srivastava S, Negi P (2018) An intelligent noninvasive model for coronary artery disease detection. *Complex Intell Syst* 4(1):11–18
35. Vladislavleva EJ, Smits GF, Den Hertog D (2008) Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Trans Evolut Comput* 13(2):333–349
36. Wang Z, Schaul T, Hessel M et al (2016) Dueling network architectures for deep reinforcement learning. In: *Proceedings of the 33rd international conference on machine learning, ICML 2016, JMLR workshop and conference proceedings*, vol 48, pp 1995–2003. JMLR.org

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.