Effective Policy Gradient Search for Reinforcement Learning Through NEAT Based Feature Extraction

Yiming Peng(✉), Gang Chen, Mengjie Zhang, and Yi Mei

School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand
{yiming.peng,gang.chen,mengjie.zhang,yi.mei}@ecs.vuw.ac.nz

Abstract. To improve the effectiveness of commonly used Policy Gradient Search (PGS) algorithms for Reinforcement Learning (RL), many existing works considered the importance of extracting useful state features from raw environment inputs. However, these works only studied the feature extraction process, but the learned features have not been demonstrated to improve reinforcement learning performance. In this paper, we consider NeuroEvolution of Augmenting Topology (NEAT) for automated feature extraction, as it can evolve Neural Networks with suitable topologies that can help extract useful features. Following this idea, we develop a new algorithm called NEAT with Regular Actor Critic for Policy Gradient Search, which integrates a popular Actor-Critic PGS algorithm (i.e., Regular Actor-Critic) with NEAT based feature extraction. The algorithm manages to learn useful state features as well as good policies to tackle complex RL problems. The results on benchmark problems confirm that our proposed algorithm is significantly more effective than NEAT in terms of learning performance, and that the learned features by our proposed algorithm on one learning problem can maintain the effectiveness while it is used with RAC on another related learning problem.

Keywords: NeuroEvolution · NEAT · Policy Gradient Search · Actor-Critic · Reinforcement learning · Feature extraction

1 Introduction

In many real-world applications such as robotics control, game playing, and system optimization, Reinforcement Learning (RL) has attracted increasingly attentions from both academic researchers and industrial practitioners [20,22]. In a typical RL setting, an intelligent agent is designed to consecutively interact with an unknown environment. In one interaction, the agent observes at any state and takes an action, then it enters another state meanwhile perceives an instant reward to assess the action. Through such continuous interactions, the agent aims to learn suitable behaviors (a.k.a., policies) that can maximize the expected total rewards (a.k.a., value function) [20].

Y. Shi et al. (Eds.): SEAL 2017, LNCS 10593, pp. 473–485, 2017.
https://doi.org/10.1007/978-3-319-68759-9_39

The paper is focused on a widely-used Actor-Critic Policy Gradient Search (AC-PGS) framework, where the critic defines the value function learner whereas the actor defines the policy learner [5,21]. Both models share the same collection of state features represented as numerical vectors. Such features are extracted from the raw environmental inputs. However, most of existing AC-PGS algorithms mainly focus on improving the critic via various gradient-descent algorithms without significantly enhancing the feature extraction process. Traditionally, it is often assumed that good state features can be designed by experienced domain experts, which is usually time-consuming and error-prone [2,14,15,22].

In view of this, many researchers have considered to automate the feature extraction process [6,15]. This is normally achieved through two steps. First, a parametric function is chosen carefully as a feature base, including Radial Basis Function Network [15], Fourier Basis Function [11] and other types of bases [20]. Next, the feature function parameters are learned by optimizing carefully-designed score functions such as the Bellman Error [14,15], and Mean Squared Error [6]. However, the aim of these methods is to accurately approximate the value function based on given solutions to the RL problems. Therefore, they do not aim to solve the RL problems by finding effective policies. Motivated by this, we intend to develop a new algorithm that can seamlessly integrate automatic feature extraction with effective policy search.

For the purpose of extracting useful features, we particularly consider an Evolutionary Computation (EC) method, i.e., NeuroEvolution of Augmenting Topology (NEAT) because of two reasons. First, NEAT produces Neural Networks (NNs) which are widely applicable with proven effectiveness as feature extraction models [2,20]. Second, compared to other EC methods, NEAT has the capability of maintaining the structural simplicity of NNs. The structural simplicity of an NN can help to reduce the variances of its outputs [1]. In doing so, we expect to stabilize the learning of value function and policy [9].

For the purpose of learning effective policies, we use a popular algorithm called Regular Actor-Critic (RAC-PGS) [3]. We choose RAC-PGS due to its simplicity and proven effectiveness [3]. More importantly, it can be easily adapted to accommodate NEAT as a feature extractor during the entire learning process.

Goals: The overall goal of the paper is to develop an effective RAC-PGS algorithm through NEAT based feature extraction called NEAT with Regular Actor Critic for Policy Gradient Search (NEAT-RAC-PGS). Through developing this algorithm, we intend to achieve four specific objectives:

- To design a new method to automatically extract suitable state features through NNs evolved by NEAT.
- To utilize state features extracted by the evolved NNs in RAC-PGS for effective policy search.
- To evaluate the effectiveness of NEAT-RAC-GPS on two benchmark problems.
- To examine the usefulness of the feature learned by NEAT-RAC-PGS on one learning problem in improving the effectiveness of RAC on a related learning problem.

2 Related Work

In literature, the applicability of EC techniques to the RL domain has already been widely studied. NEAT and its variations (e.g., Hyper-NEAT) have reported outstanding performances on solving classic control problems and intelligent game-play problems [18,19]. Genetic Programming has been integrated as an RL agent to solve real-world robotic problems with notable success [10]. Many other evolutionary algorithms, such as Learning Classifier Systems, have also been successfully applied to addressing sophisticated RL problems [4,12]. Unlike NEAT-RAC-PGS, these methods only focus on improving learning performance, but their outputs cannot be used to solve other different but similar problems. NEAT-RAC-PGS can not only solve the problem effectively, but also produce useful feature extractors that can be applied to similar problems.

We are not the first to consider using NEAT for feature learning during RL. For example, FS-NEAT [23] and its variations [13] are limited to perform only feature selection rather than feature extraction, where no high level features are explicitly evolved. NEAT+Q [22] takes feature extraction into consideration, but different from NEAT-RAC-PGS, it is a value function based indirect search which does not explicitly learn a policy. In addition, as the extracted features of NEAT+Q are embedded in value functions, they cannot be reused for other learning algorithms. This paper will address these gaps by developing the NEAT-RAC-PGS algorithm.

3 A New NEAT Based Policy Gradient Search Algorithm

In this section, we propose the NEAT-RAC-PGS algorithm to tackle RL problems. Firstly, we present the overall design of NEAT-RAC-PGS. Next, we describe details of NEAT-RAC-PGS with support of algorithmic descriptions. Lastly, we discuss the new characteristics of NEAT-RAC-PGS with comparison to existing RL algorithms.

3.1 Overall Design

Figure 1 shows an overall design of our NEAT-RAC-PGS algorithm. As shown in the figure, NEAT-RAC-PGS evolves a population of RL learning agents, $P = \{A_1, \ldots, A_p\}$. Each agent A_i consists of three components: an NN $\phi(\boldsymbol{s}) \in \mathbb{R}^z$, a parameter vector for value function $\boldsymbol{\omega}_i$, and a parameter vector for policy $\boldsymbol{\theta}_i$. Based on the extracted features $\phi(\boldsymbol{s})$, the value function in each agent is approximated as,

$$V^{\pi}(\boldsymbol{s}) \approx \boldsymbol{\omega}^{\pi T} \cdot \phi(\boldsymbol{s}), \tag{1}$$

and is learned following the reducing direction of Temporal Difference (TD) error at every t time when the agent reaches a new state at $t+1$, i.e.,

$$\delta_t^{\pi} = r_{t+1} + \gamma V^{\pi}(\boldsymbol{s}_{t+1}) - V^{\pi}(\boldsymbol{s}_t). \tag{2}$$

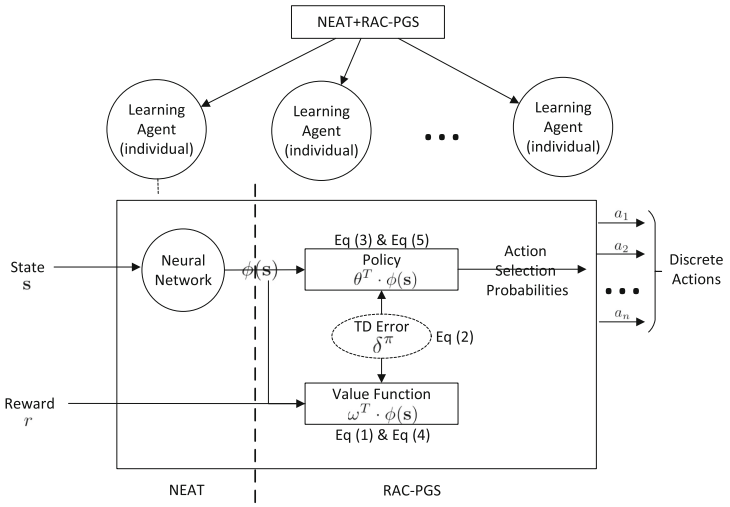


Fig. 1. The overall design of NEAT-RAC-PGS.

Additionally, the policy for action selection is formulated as,

$$\pi_{\boldsymbol{\theta}}(a|\boldsymbol{s}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(a-\mu)^2}{2\sigma^2}}, \tag{3}$$

where $\mu = \boldsymbol{\theta}^T \cdot \phi(\boldsymbol{s})$. $\sigma = 1.0$ is used to control the level to explore new actions. Note that, π at the RHS of (3) is the circumference ratio.

In association with (1), (2) and (3), the updating rule for value function parameters is,

$$\boldsymbol{\omega}_{t+1}^{\pi} \leftarrow \boldsymbol{\omega}_t^{\pi} + \alpha_t \delta_t^{\pi} \phi(\boldsymbol{s}_t), \tag{4}$$

and the rule for policy parameters is

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \beta_t \delta_t^{\pi} \Phi(\boldsymbol{s}, a), \tag{5}$$

respectively. Note, α_t and β_t are the learning rates, and $\Phi(\boldsymbol{s}, a) = \nabla_{\boldsymbol{\theta}} \ln \pi(\boldsymbol{s}, a) = \frac{a_t - \boldsymbol{\theta} \cdot \phi(\boldsymbol{s}_t)}{\sigma^2} \cdot \phi(\boldsymbol{s}_t)$ [21].

3.2 Policy Gradient Search Through NEAT Based Feature Extraction

NEAT-RAC-PGS is designed with four learning stages, namely *initialization*, *evolution*, *evaluation*, and *termination*.

Initialization. The initialization stage of NEAT-RAC-PGS is to initialize a population with p learning agents (i.e., individuals). For the NN in each agent, its inputs and outputs are defined as states $\boldsymbol{s}$ and extracted state features $\phi(\boldsymbol{s})$

respectively. Additionally, its input nodes are directly connected to its output nodes without any hidden nodes, and weights of each NN are randomly initialized. For the critic and actor model in each agent, the value function parameters and policy parameters are initialized to $\boldsymbol{\omega}_0$ and $\boldsymbol{\theta}_0$ respectively. Note, $\boldsymbol{\omega}_0$ and $\boldsymbol{\theta}_0$ are vectors with arbitrary values.

Evolution. NEAT-RAC-PGS evolves a new population of agents by performing two sequential tasks according to Sect. 3.1. The first task is to evolve p NNs ($\{\phi_1, \ldots, \phi_p\}$) based on the standard evolutionary operators of NEAT defined in [19]. The second task is to initialize $\boldsymbol{\omega}_i$ and $\boldsymbol{\theta}_i$ to $\boldsymbol{0}$ for every agent A_i in the new population. This ensures a fair evaluation across all agents.

Evaluation. The evaluation stage of the NEAT-RAC-PGS has two objectives, one is to compute the fitness value for a single individual, and the other is to find good policies. The fitness value of each individual is directly computed by averaging total rewards obtained by the learning agent over all episodes. Using an average value can help reduce the variance as rewards are collected in a stochastic environment. To search for the good policies, we use the RAC-PGS to learn the critic and actor models for an individual from its continuous interactions with the environment. This stage is presented in Algorithm 1.

Termination. The termination stage manages stop criteria for the learning process of NEAT-RAC-PGS. To stop the feature extraction process, we define two stop criteria: the first is to stop when the predefined maximum number of generations is reached, and the second is when the fitness values of all individuals do not improve over 10 generations. To cease the policy search process, we setup the maximumly allowed learning episodes for an RL agent to interact with an environment. Each episode is composed of multi interaction steps, and it terminates at the situations when either the predefined maximum number of steps is reached, or the target RL problem is solved (see Sect. 4.1). A full algorithmic description for NEAT-RAC-PGS is given in Algorithm 2.

3.3 Key Characteristics of NEAT-RAC-PGS

In this subsection, we discuss three key characteristics of NEAT-RAC-PGS in contrast to traditional PGS approaches and EC based RL methods, namely *multiple diverse learning agents*, *automated feature extraction*, and *reusable features extractors*. These three characteristics enable each of NEAT-RAC-PGS learning agent to produce useful and reusable state features meanwhile to find good policies that solve sophisticated RL problems.

Multiple Diverse Learning Agents. As explained in Sect. 3.1, NEAT-RAC-PGS is a population-based method composed of multiple diverse learning agents whereas traditional PGS methods are often constructed as a single learning

Algorithm 1. NEAT-RAC-PGS Evaluation

Require: P: a population of learning agents, p: population size, e_g: maximum training episodes per generation, T: maximum training steps per episode, α: value function learning rate, β: policy learning rate, $\tilde{R}$: total rewards
Ensure: P: a population of learning agents

function EVALUATION(P, p, e_g, T, α, β)
  for $k = 1, 2, ..., p$ **do**
    $\tilde{R} \leftarrow 0$
    for $j = 1, 2, ..., e_g$ **do**
      $\boldsymbol{s}_t \leftarrow \boldsymbol{s}_0$
      for $t = 0, 1, \ldots, T-1$ **do**
        $a_t \sim \pi_\theta(a|\boldsymbol{s}_t)$ ▷ See (3)
        Take action a_t, observe reward r_{t+1} and new state $\boldsymbol{s}_{t+1}$
        $\delta_t \leftarrow r_{t+1} + \gamma\omega_t^T \cdot \phi(\boldsymbol{s}_{t+1}) - \omega_t^T \cdot \phi(\boldsymbol{s}_t)$ ▷ See (2)
        ▷ Note: $\phi = P[k].N$, $\omega_t = P[k].\omega$, $\theta_t = P[k].\theta$
        $\omega_{t+1} \leftarrow \omega_t + \alpha\delta_t \cdot \phi(\boldsymbol{s}_t)$ ▷ See (4)
        $\theta_{t+1} \leftarrow \theta_t + \beta\delta_t \cdot \Phi(\boldsymbol{s}_t, a_t)$ ▷ See (5)
        $\tilde{R} \leftarrow \tilde{R} + r_{t+1}$
        if TERMINAL-STATE($\boldsymbol{s}_{t+1}$) **then**
          break
        end if
      end for
    end for
    $P[k].N.fitness \leftarrow \frac{\tilde{R}}{e_g}$
  end for
  return P
end function

agent. In dosing so, NEAT-RAC-PGS has better potential to find good policies, even when several learning agents are stuck in local optima.

Automated Feature Extraction. Unlike those PGS methods [5] where feature extractors are predetermined and are fixed through the entire reinforcement learning process, NEAT-RAC-PGS incorporates NEAT as a new component to automatically learn suitable feature extractors along the policy search process as shown in Fig. 1. This helps the learning agent to find useful features which can further enhance its ability to find good policies.

Reusable Feature Extractors. Unlike many existing EC based RL methods where feature extraction and policy search are mingled together, NEAT-RAC-PGS treats them as two subsequent stages. To do so, the RAC algorithm is wrapped as a fitness evaluator for each individual. Following this design principle, we can explicitly represent the feature extractors as NNs. Thus, after learning these feature extractors on one problem, it becomes possible to reuse them for various RL algorithms on a related problem.

Algorithm 2. NEAT-RAC-PGS Algorithm

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, p: population size, g: number of generations, e_g: maximum training episodes per generation, T: maximum training steps per episode, d: state dimension, z: feature dimension, α: value function learning rate, β: policy learning rate, ω_0: initial value function parameters, β_0: initial policy parameters

Ensure: P: the population of individuals, N^*: the optimal Neural Network ($\phi(\boldsymbol{s})$), ω^*: the best value function parameter, θ^*: the best policy parameter

Initialization:
$P \leftarrow$ `INITIALIZATION`(p, d, z, ω_0, β_0) ▷ See Sect. 3.2
Learning Process:
for $i = 1, 2, ..., g$ **do**
  $P \leftarrow$ `EVALUATION`(P, p, e_g, T, α, β) ▷ See Algorithm 1
  if $i < g$ **then** ▷ Stop evolution at the final generation
    $P \leftarrow$ `EVOLUTION`(P) ▷ See [19]
  end if
end for
$k^* \leftarrow \text{argmax}_k(P[k].N.fitness)$
$N^* \leftarrow P[k^*].N$
$\omega^* \leftarrow P[k^*].\omega$
$\theta^* \leftarrow P[k^*].\theta$
return P, N^*, ω^*, θ^*

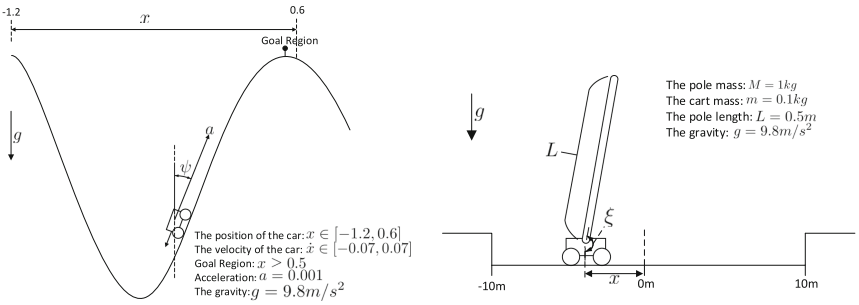
4 Experiment Design

In this section, we present our experiment design. We first describe the two continuous benchmark problems, i.e., the Cart-Pole problem and the Mountain Car problem. We then describe detailed setups for the experiments. Lastly, we present two separate experiment designs to examine the usefulness of extracted features and to evaluate the effectiveness of learned policies respectively.

4.1 Benchmark Problems

We choose two continuous benchmark problems, i.e., the Mountain Car problem [20] and the Cart Pole problem [20] for experiments. Because they are widely exploited to examine effectiveness of new RL algorithms. In addition, neural networks have already been successfully used as feature extractors on these problems [20,22].

Mountain Car Problem [20]**.** The Mountain Car problem, as shown in Fig. 2(a), models a two-dimensional environment where the two dimensions of the state represent the position of the car x and the velocity of the car $\dot{x}$. The goal of the problem is to thrust a car from the bottom of the valley to a steep mountain at the right side. As the power of the car's engine is weaker than the gravity, the car needs the descending acceleration generated by sliding from the opposite slope at the left side. The top of the mountain is set as a goal region, and the car aims to use the minimum steps to reach the region. In the problem, the car



(a) The Mountain Car problem (b) The Cart Pole problem

Fig. 2. The two benchmark problems drawn based on description in [20]

moves a step meanwhile receives a penalty "−1", a reward "+100" is given until the car reaches the goal region. The car updates its location and speed following the equation below,

$$\ddot{x} = \dot{a}\mathcal{F} - 0.0025\ \cos(3x),$$

where $\dot{a} = 0.001$ denotes the sliding acceleration obtained by the car. $\mathcal{F}$ represents a force (i.e., continuous action $a \in [-1.0, 1.0]$) performed by the system.

Note that, the initial state for the Mountain Car problem is fixed as $\boldsymbol{s} = [-5.0, 0.0]$, and the terminative condition of a learning episode is defined as either the maximum learning steps or the goal regions is reached.

Cart Pole Problem [20]**.** In the Cart Pole problem shown in Fig. 2(b), a learning agent learns to generate an action $\mathcal{F}$ (i.e., $a \in [-10, 10]$) in the form of a horizontal force that drives a cart to move along a fixed length track. Meanwhile, it aims to balance the hinged rigid pole on the cart to the up-right position. The state contains four dimensions, including x_t (the relative position of the center of the cart to the center of the track), $\dot{x}_t$ (the velocity of the cart), ξ_t (the relative angle of the pole to the up-right position), and $\dot{\xi}_t$ (the angular velocity of the pole). Accompanied with the cart's current movement, the environment provides an instant reward determined by

$$r_{t+1} = \begin{cases} 0.0, & \text{if } |\xi| > 0.628 \text{ or } |x| > 10.0 \\ 0.2\pi - \xi, & \text{otherwise} \end{cases}. \tag{6}$$

In addition, the dynamics of cart and pole can be defined as,

$$\begin{aligned} \ddot{\xi} &= \frac{g \cdot sin(\xi) - cos(\xi)(F + M \cdot L \cdot \dot{\xi}^2 sin(\xi))}{\frac{4}{3}L - \frac{m \cdot cos(\xi^2)}{m+M}} \\ \ddot{x} &= \frac{\mathcal{F} + M \cdot L \cdot \dot{\xi}^2 \cdot sin(\xi) - M \cdot L \cdot \ddot{\xi} \cdot cos(\xi)}{m+M} \end{aligned}.$$

Moreover, a learning episode for the Cart Pole problem is alway starts from the initial state $\boldsymbol{s} = [0.0, 0.0, 0.15, 0.0]$, and it terminates whenever a maximum

number of learning steps have been conducted in the episode, or the terminating condition (i.e., $|\xi| > 0.628$ or $|x| > 10.0$) is satisfied.

4.2 Experiment Setup

To identify any significant performance difference, for each benchmark problem, we conduct 30 independent runs for our algorithms. Among all runs, the population size and the number of generations are set to 100 and 100 respectively. For the Mountain Car problem, we perform 1000 learning episodes with 200 steps in each episode and 25 independent testing episodes where each has 200 testing steps. For the Cart Pole problem, we perform 200 learning steps in each of the 5000 learning episodes. Also, we conduct 1000 testing steps in each of 25 independent testing episodes. All independent tests are performed by the best learning agent of each generation.

Some important meta parameter settings are summarized as follows. Firstly, for meta parameters of NEAT and NEAT component of NEAT-RAC-PGS, we adopt identical settings reported in [22] where the effectiveness of NEAT with these settings has been verified on several benchmark problems. Secondly, we choose the commonly used settings ($\alpha = 0.1$, $\beta = 0.01$, $\gamma = 0.99$) reported in [3,16] for RAC-PGS algorithm and RAC-PGS component of NEAT-RAC-PGS. These meta parameter settings are used for experiments on both benchmark problems described in Sect. 4.1.

4.3 Experiment Design

In this paper, driven by the last two research objectives, we will conduct two different types of experiments. In the first type of experiment, as discussed in Sect. 4.2, we evaluate the performance of NEAT-RAC-PGS on each benchmark problem to understand the effectiveness of the algorithm. The performance is measured by the number of steps for balancing the pole, or the number of steps to drive the car to mountain top.

In the meantime, driven by the last objective, we intend to evaluate the usefulness of the state features on a relevant problem. We expect that the learned useful state features can be reused on a related problem and still can maintain or improve the learning performance. For this purpose, in our feature evaluation, we firstly maintain the feature extractor of the current best learning agent by NEAT-RAC-PGS (i.e., the NN evolved by NEAT) on Cart Pole. Next, we modify the Cart Pole problem described in Sect. 4.1 to obtain a relevant problem. To do so, we uniformly set an arbitrary value from $[-0.05, 0.05]$ for every dimension of initial states at the beginning of every learning episode. Lastly, we adopt the RAC-PGS learning algorithm with the maintained feature extractor on the modified Car Pole problem to observe the learning performance for understanding the usefulness of state features.

5 Results and Discussion

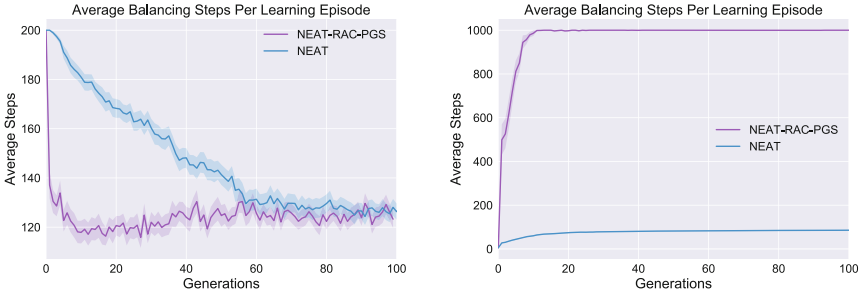
The experimental results are presented and analyzed in this section. We firstly analyze the learning effectiveness of NEAT-RAC-PGS contrary to NEAT on the two benchmark problems. Next, we discuss the usefulness of learned features by NEAT-RAC-PGS.

5.1 Learning Effectiveness Evaluation

To evaluate the learning effectiveness, we present the learning performances obtained by NEAT-RAC-PGS and NEAT on the Mountain Car problem in Fig. 3(a) and on the Cart Pole problem in Fig. 3(b) respectively.

As shown in Fig. 3(a), after a certain period of time, both NEAT-RAC-PGS and NEAT achieve reasonably good performance (around 120 steps), which is consistent with the results of NEAT reported in [22]. The two high peaks of NEAT-RAC-PGS do not imply that it performs sometimes worse than NEAT, as no significance can be found after the 53-th generation according to a statistical test. Meanwhile, owing to the capability of NEAT-RAC-NEAT to make more effective use of learned features, eventually it achieved better performance than NEAT.

Figure 3(b) evidently shows that NEAT-RAC-PGS outperforms NEAT with averaging 1000 steps after the 10-th generation. As a matter of fact, NEAT achieves the desirable performance (approximately 100 steps) according to the OpenAI GYM benchmark [7]. Without comparing it with NEAT-RAC-PGS, NEAT should be treated as an effective algorithm. However, our NEAT-RAC-PGS algorithm can achieve significantly higher performance than many recently developed algorithms that cannot compete on the same benchmark.

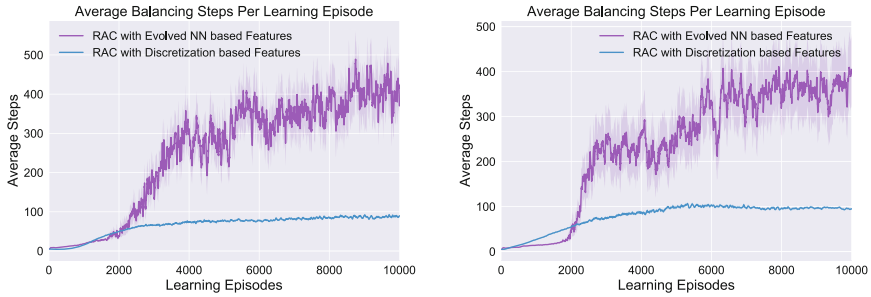


(a) Mountain Car (b) Cart Pole

Fig. 3. The comparison of learning performance of NEAT-RAC-PGS and NEAT on two benchmark problems: (a) displays the averaging steps to reach the goal region on Mountain Car (the smaller the better), (b) displays the averaging steps to balance the pole to the upright position on Cart Pole (the larger the better).

5.2 Feature Usefulness Evaluation

To evaluate the usefulness of features, in Fig. 4, we compare the learning performances of RAC-PGS with two different feature extractors on the standard Cart Pole problem and on the modified Cart Pole problem respectively. The first feature extractor is a learned NN by the NEAC component of an NEAT-RAC-PGS learning agent on the standard Cart Pole with the highest fitness value. The second feature extractor is the widely-used discretization feature extractor where each dimension of environment state is discretized into 20 bins as described in [20].



(a) Cart Pole (b) Modified Cart Pole

Fig. 4. The comparison of learning performance of RAC-PGS with two different feature extractors (an evolved NN feature extractor and a predefined discretized feature extractor) on the two related problems (see Sect. 4.2): (a) displays learning performances obtained on the standard Cart Pole problem, (b) displays learning performances obtained on the modified Cart Pole problem.

Figure 4(a) reveals that, on the standard Cart Pole problem, RAC with evolved NN features performs significantly and consistently better than RAC with discretized features after the 2370-th generation. On the other hand, when the learning environment is changed to the modified Cart Pole problem, we are still able to find a similar observation in Fig. 4(b) that RAC with evolved NN features outperforms RAC with discretized features after the 2214-th generations. Statistical tests reject the null hypothesis after the time points. The RAC with discretized features achieves averaging 100 steps on both problems, which can be regarded effective as reported in [17]. These results suggest that the features learned by NEAT-RAC-PGS on one problem can not only be used to solve the original problem, but also be adopted to solve similar but different problems.

Nevertheless, on both problems, the RAC-PGS with evolved NN features appears to fluctuate. Actually, the step-based learning process of NNs has already been reported unstable in literature [8], because step-based learning always picks up the recent state transitions for estimating gradients, which may bring bias into gradient estimation resulting in the unstable updating. A possible solution to address the issue is experience replay, but this is not the focus of this paper.

The reason that evolved NN features are superior to discretized features is given below. Firstly, the real-word environment (e.g., Cart Pole) often exhibits high non-linearity, NN is well known as suitable non-linear models in comparison to other models. Secondly, the raw environment input is continuous, some intrinsic information, that is useful for finding good policies, may be lost during the discretization. On the other hand, NN is capable of smoothly producing continuous values as high level features. More importantly, NEAT can evolve both weights and structures for NNs, which further provides higher chances to find suitable state features.

6 Conclusions

This paper has successfully achieved the goal of developing a new algorithm to tackle reinforcement learning problems by integrating a modern AC-PGS algorithm, i.e., RAC, with NEAT for automated feature extraction. This integration brings two contributions. First, different from most existing approaches only concentrating on automated feature extraction, the proposed NEAT-RAC-PGS algorithm can also find better policies to solve the reinforcement learning problems in addition to extracting good features automatically. Second, compared to traditional policy search methods, NEAT-RAC-PGS can identify useful state features that can be further reused in related learning problems. Experiment results confirmed the potential advantages of the proposed algorithm. This work opens possibilities of exploiting other cutting-edge AC-PGS algorithms based on the same design principle of NEAT-RAC-PGS. More comprehensive experiments involving a wide range of benchmarks can also help truly understand the real efficacy of NEAT-RAC-PGS.

References

1. Balduzzi, D., Frean, M., Leary, L., Lewis, J.P.: The shattered gradients problem: if resnets are the answer, then what is the question? arXiv.org (2017)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1798–1828 (2013)
3. Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Natural actor-critic algorithms. Automatica **45**(11), 2471–2482 (2009)
4. Chen, G., Douch, C.I.J., Zhang, M.: Accuracy-based learning classifier systems for multistep reinforcement learning: a fuzzy logic approach to handling continuous inputs and learning continuous actions. IEEE Trans. Evol. Comput. **20**(6), 953–971 (2016)
5. Deisenroth, M.P., Neumann, G., Peters, J.: A survey on policy search for robotics. Found. Trends Robot. **2**(1–2), 1–142 (2013)
6. Castro, D., Mannor, S.: Adaptive bases for reinforcement learning. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010. LNCS (LNAI), vol. 6321, pp. 312–327. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15880-3_26

7. Grondman, I., Busoniu, L., Lopes, G.A.D., Babuška, R.: A survey of actor-critic reinforcement learning: standard and natural policy gradients. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **42**(6), 1291–1307 (2012)
8. Gu, S., Lillicrap, T.P., Sutskever, I., Levine, S.: Continuous deep q-learning with model-based acceleration. In: ICML, pp. 2829–2838 (2016)
9. Hermundstad, A.M., Brown, K.S., Bassett, D.S., Carlson, J.M.: Learning, memory, and the role of neural network architecture. PLoS Comput. Biol. **7**(6), e1002063 (2011)
10. Kamio, S., Iba, H.: Adaptation technique for integrating genetic programming and reinforcement learning for real robots. IEEE Trans. Evol. Comput. **9**(3), 318–333 (2005)
11. Konidaris, G., Osentoski, S., Thomas, P.: Value function approximation in reinforcement learning using the fourier basis. In: 2011 AAAI, pp. 380–385 (2011)
12. Lanzi, P.L.: Learning classifier systems: then and now. Evol. Intell. **1**(1), 63–82 (2008)
13. Loscalzo, S., Wright, R., Yu, L.: Predictive feature selection for genetic policy search. AAMAS **2014**, 1–33 (2014)
14. Menache, I., Mannor, S., Shimkin, N.: Basis function adaptation in temporal difference reinforcement learning. Ann. Oper. Res. **134**(1), 215–238 (2005)
15. Parr, R., Painter-Wakefield, C., Li, L.: Analyzing feature generation for value-function approximation. In: ICML, pp. 737–744 (2007)
16. Peng, Y., Chen, G., Zhang, M., Pang, S.: A sandpile model for reliable actor-critic reinforcement learning. In: IJCNN, pp. 4014–4021. IEEE (2017)
17. Peng, Y., Chen, G., Zhang, M., Pang, S.: Generalized compatible function approximation for policy gradient search. In: Hirose, A., Ozawa, S., Doya, K., Ikeda, K., Lee, M., Liu, D. (eds.) ICONIP 2016. LNCS, vol. 9947, pp. 615–622. Springer, Cham (2016). doi:10.1007/978-3-319-46687-3_68
18. Schrum, J., Miikkulainen, R.: Discovering multimodal behavior in ms. pac-man through evolution of modular neural networks. IEEE Trans. Comput. Intell. AI Games **8**(1), 67–81 (2016)
19. Stanley, K.O., Miikkulainen, R.: Evolving neural network through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
20. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. MIT press, Cambridge (1998)
21. Sutton, R.S., Mcallester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: NIPS, pp. 1057–1063 (1999)
22. Whiteson, S., Stone, P.: Evolutionary function approximation for reinforcement learning. J. Mach. Learn. Res. **7**(5), 877–917 (2006)
23. Whiteson, S., Stone, P., Stanley, K.O., Miikkulainen, R., Kohl, N.: Automatic feature selection in neuroevolution. In: 2005 GECCO, pp. 1225–1232 (2005)