Automated State Feature Learning for Actor-Critic Reinforcement Learning through NEAT

ABSTRACT

Actor-Critic (AC) algorithms are important approaches to solving sophisticated reinforcement learning problems. However, these algorithms rely typically on good state features that are often designed manually. This requires a lot of efforts from domain experts and if important state features were overlooked then the learning performance could be seriously affected. To address this issue, we propose to adopt an evolutionary approach based on NeuronEvolution of Augmenting Topology (NEAT) to automatically identify useful state features with the help of an evolved neural network that can transform raw environmental inputs directly into state features. Following this idea, we have developed a new algorithm called NEAT+AC which combines Regular-gradient Actor-Critic (RAC) with NEAT. It can simultaneously learn suitable state features as well as good policies that can solve any given reinforcement learning problems. Experiments on benchmark problems confirm that our new algorithms are significantly more effective in comparison to the baseline algorithm, i.e., NEAT.

CCS CONCEPTS

•Computing methodologies → Neural networks; •Computer systems organization → Embedded systems; *Redundancy;* Robotics;

KEYWORDS

NeuroEvolution, NEAT, Actor-Critic, Reinforcement Learning, Feature Extraction, Feature Learning

ACM Reference format:
. 2017. Automated State Feature Learning for Actor-Critic Reinforcement Learning through NEAT. In *Proceedings of The Genetic and Evolutionary Computation Conference, Berlin, Germany, July 2017 (GECCO'17),* 8 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Reinforcement Learning (RL) is a typical machine learning paradigm in which an agent learns a policy for sequential action selection while observing states in an unknown environment [8, 11, 13, 20, 25]. The goal of RL is to find an optimal policy that maximizes the expected multi-step cumulative rewards. Many real-world applications can be categorized into the paradigm, such as robotics control, game playing, and system optimization [8, 20, 23, 25]. Thus, it is substantial to develop effective RL algorithms to meet the real-world application demands.

Among all existing RL algorithms, an important family is the so-called Actor-Critic Reinforcement Learning (ACRL) algorithms [6, 7, 20]. These algorithms are designed to directly search the policy (a.k.a., actor) with the aid of value functions (a.k.a, critic) [3, 7, 20]. Conceptually, the actor is a mapping used to recommend an action in any encountered state, whereas the critic is an criticizer to assess the quality of the actor by estimating the cumulative rewards obtainable via the actor. Accordingly, the overall process of ACRL can be divided into two separated learning subprocesses, namely critic learning and actor learning.

Many existing ACRL algorithms assume that the policy or actor can be mathematically described through a linear function of multiple numerical state features. Accordingly, the critic is often modeled through another linear function of compatible state features. In [22], Sutton discovered an important relationship in between the critic and the actor such that the compatible state features can be uniquely derived from the state features for the actor. Driven by this mathematical framework, many effective ACRL algorithms have been proposed recently, including Regular-gradient Actor-Critic (RAC), Natural-gradient Actor-Critic (NAC) and so forth [3, 6, 7, 17, 18].

For all these algorithms, it is taken for granted that suitable state features are immediately accessible during reinforcement learning. However, the validity of this assumption is often under challenge in practice. As a matter of fact, researchers found that, for effective RL, these state features must be carefully designed, usually with the support of domain experts. Obviously, engineering useful state features is a time-consuming and error-prone procedure [2, 12, 15, 23]. Even for experienced domain experts, it is difficult to accurately predetermine suitable state features [2, 25]. This difficulty may bring about inappropriate state features, where important state feature information may be overlooked, resulting in serious deterioration to learning performance.

To address the issue, researchers have increasingly focused on developing interesting algorithms for state feature extraction. This can be achieved by firstly choosing a parametric function for a feature base, such as Radial Basis Function Network [15], Fourier Basis Function [10] or other types of bases [20]. Next, state feature learning is conducted by optimizing some predefined score functions, such as the bellman-error-based approach [15], or the weighted 2-norm of the value function approximation approach [12]. However, these methods need careful selection of the score functions for different problem domains. Additionally, when neural networks are chosen as the feature base, its topology also requires to be well designed prior to the activation of any learning algorithms. These new issues motivate us to consider exploiting an evolutionary approach towards fully automated state feature learning which can be performed simultaneously with any ACRL algorithm.

Evolutionary Computation (EC) techniques have been proven to be very effective for solving complex and difficult RL problems [19, 23, 24]. Generally, EC techniques enjoy many advantages [14, 23, 25]. First of all, they can be directly used for tackling discontinuous problems, for example, whose reward functions do not have to be contiguous. Traditional RL methods may not be suitable in such cases because the continuity is often a critical assumption for their employment [21]. Secondly, evolutionary techniques are scalable to large and continuous action space which pose challenges to traditional reinforcement learning methods. Thirdly, evolutionary methods can be easily applied to sophisticated practical problems without any domain knowledge. For example, the traditional way of using neural networks often requires experienced practitioners to well design the structures and back-propagation training methods in advance. Fourthly, EC methods also exhibit desirable robustness. They do not need to fine tune lots of meta parameters (e.g., learning rate). Lastly, EC algorithms are easy to understand in comparison to many traditional learning algorithms.

In this research, we are specifically interested in one major EC method for NeuroEvolution, i.e, NeuroEvolution of Augmenting Topology (NEAT). This is because of several reasons. 1) Neural networks are well recognized as good feature bases for various learning paradigm including RL [2]. 2) NEAT has a strong capability of evolving both structure and weights simultaneously, this is important for us to identify suitable feature extractor for the RL problems. 3) NEAT introduces a unique innovation number to each individual preserve useful structural innovations for future learning. 4) It adopts a strategy to evolve increasingly complicated neural networks starting from the simplest structures. This is also very important for our feature learning, since we expect the structures of our feature extraction neural networks to be as simple as possible but very useful for future tuning. 5) NEAT is widely demonstrated as a very effective algorithm for RL [19, 23, 24]. This is why we aim to explore the strength of NEAT and build up a more effective RL algorithm.

In the literature, many existing research use NEAT as a basis to propose a number of variations, such as HyperNEAT [5], FS-NEAT [24], NEAT+Q [23] and so forth [23, 25]. The neural network that evolved by all these works will be directly used as action selectors, on the other hand, we want to use evolved neural networks only as feature basis. To the best of our awareness, no existing works have ever considered using NEAT to explicitly learn state features and integrating NEAT seamlessly with an ACRL algorithm.

1.1 Goals

Motivated by this understanding, the overall goal of this research is to develop a new algorithm (NEAT+AC) based on NEAT and RAC. Through the seamless integration of NEAT and AC, we can learn good features, in the mean time use the learned features to identify desirable policies. With the help of this algorithm, we intend to achieve five specific objectives:

1 To develop a new ACRL algorithm for effective and efficient reinforcement learning on the basis of NEAT for useful feature learning.
2 To evaluate the learning performance of new algorithm in comparison with NEAT.
3 To demonstrate the relationships in-between useful state features for RL.

1.2 Organization

The rest of the paper is structured as follows. Section 2 reviews the preliminary concepts for RL, several key techniques including ACRL framework and NEAT for developing the new algorithm. Section 3 proposes the NEAT+AC algorithm. Section 4 provides detailed experimental setups. Section 5 presents and analyzes the experimental results. The paper draws conclusions in Section 6.

2 PRELIMINARIES

This section begins with the background of RL as well as foundational techniques for building our NEAT+AC algorithm.

2.1 Reinforcement Learning

Reinforcement Learning (RL) describes an interactive process in-between an agent and an unknown environment [20]. In the process, the agent takes an action a_t at a state $\vec{s}_t$ and subsequently transfers to another state $\vec{s}_{t+1}$, meanwhile it receives a numerical reward r_t as an environmental feedback. Such a process is often formed as a Markov Decision Process (MDP) containing a state space $\mathcal{S}$, a discrete action space $\mathcal{A}$, a transition probability function $\mathcal{P}$, a reward function $\mathcal{R}$ and a discount factor γ.

RL aims to find an optimal policy that maximizes the expected cumulative rewards. In this work, we consider the stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ in an infinite horizon. The expected cumulative reward achieved by an agent upon following any policy π can be defined as

$$J^{\pi} = V^{\pi}(\vec{s}_0) = \mathrm{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \vec{s}_t = \vec{s}_0], \tag{1}$$

where V^{π} denotes a state-value function for the expected cumulative rewards starting from any state $\vec{s}_0$. Accordingly we can also defined the action-value function Q^{π}, i.e.,

$$Q^{\pi}(\vec{s}, a) = \mathrm{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \vec{s}_t = \vec{s}, a_t = a]. \tag{2}$$

The relationship between (1) and (2) is formulated as below:

$$V^{\pi}(\vec{s}) = \int_{a \in \mathcal{A}} \pi(\vec{s}, a) Q^{\pi}(\vec{s}, a) da. \tag{3}$$

Consequently, the optimal policy that an RL algorithm seeks for is formulated as,

$$\pi^* = \operatorname*{argmax}_{\pi} V^{\pi}(\vec{s}_0). \tag{4}$$

2.2 Actor-Critic Reinforcement Learning

At the beginning of the section, we briefly introduce the general ACRL formulations, and explains the working principles of ACRL. Afterwards, we specifically discuss one representative ACRL algorithm, i.e., RAC, which serves as the baseFal for developing our new algorithm.

2.2.1 General ACRL Formulations. ACRL algorithms commonly treat actor and critic as two separate parametric functions that can be learned simultaneously [7, 9, 22]. Hence, the goal of RL to search an optimal policy is actually to seek optimal policy parameters. For simplicity of learning, both actor and critic parametric functions are normally represented as linear functions [1].

Accordingly, the stochastic policy is formulated as [6],

$$\begin{aligned} a &\sim \pi_{\vec{\theta}}(a|\vec{s}) \\ \mathrm{E}[a|\pi] &= \vec{\theta} \cdot \phi(\vec{s}) \end{aligned} \tag{5}$$

where $\vec{\theta} \in \vec{\Theta}$ denotes the policy parameters, and the policy π is only linearly dependent on $\vec{\theta}$.

Similarly, we have the linear value function defined as [6],

$$V^{\pi}(\vec{s}) \approx \tilde{V}^{\pi}(\vec{s}) = \vec{\omega}^{\pi T} \cdot \phi(\vec{s}), \tag{6}$$

where $\vec{\omega}^{\pi T} \in \vec{\Omega}$ consists of the value function parameters that are linearly associated with the state features $\phi(\vec{s}) = [\phi_1(\vec{s}), \dots, \phi_m(\vec{s})] \in \mathbb{R}^m$.

In (5) and (6), $\phi(\vec{s})$ represents the state features [1] (a.k.a, basis functions) defined as,

$$\phi(\vec{s}) = [\phi_1(\vec{s}), \dots, \phi_m(\vec{s})], \tag{7}$$

where $\phi_i \in \mathbb{R}$ for $i = 1, \dots, m$, and m is the feature dimension.

Based on (5) and (4), the goal of RL is actually to search optimal policy parameters, i.e.,

$$\vec{\theta}^* = \underset{\theta}{\operatorname{argmax}}\, V^{\pi_{\vec{\theta}}}(\vec{s}_0). \tag{8}$$

In order to obtain a solution for (8), a majority of ACRL algorithms adopt Gradient Ascent technique [6], where they attempt to update policy parameters through

$$\Delta\vec{\theta} \propto \nabla_{\vec{\theta}} J(\vec{\theta}), \tag{9}$$

in which

$$\nabla_{\vec{\theta}} J(\vec{\theta}) = \int_{\vec{s} \in \mathcal{S}} d^{\pi}(\vec{s}) \int_{a \in \mathcal{A}} \nabla_{\vec{\theta}} \pi_{\vec{\theta}}(a|\vec{s}) Q^{\pi}(\vec{s}, a) da d\vec{s}, \tag{10}$$

where $d^{\pi}(\vec{s}) = \lim_{t \to T} Pr\{\vec{s}_t = \vec{s}|\vec{s}_0, \pi\}$ is the stationary probability distribution of the states under π [22].

However, the policy gradient $\nabla_{\vec{\theta}} J(\vec{\theta})$ in (9) is not analytically obtainable in practice. Hence, many algorithms have been proposed to build unbiased estimations of $\nabla_{\vec{\theta}} J(\vec{\theta})$ [3, 6, 17]. One of the most effective and representative algorithms is RAC [3]. Owing to its simplicity and ease of adaptation, we believe that it provides a suitable basis for building our new algorithm.

2.2.2 Regular-gradient Actor-Critic. In RAC, the critic learning process follows the reducing direction of the renowned Temporal Difference error (TD error). The error is commonly defined as,

$$\delta_t^{\pi} = r_{t+1} + \gamma V^{\pi}(\vec{s}_{t+1}) - V^{\pi}(\vec{s}_t). \tag{11}$$

In fact, the critic learning is actually a process to closely approximate the true value function $V^{\pi}(\vec{s})$ in (11) by adjusting the value function parameters in the direction of reducing the TD error, i.e.,

$$\vec{\omega}_{t+1}^{\pi} \leftarrow \vec{\omega}_t^{\pi} + \alpha_t \delta_t^{\pi} \phi(\vec{s}_t), \tag{12}$$

where α is the critic learning rate at time t.

Actor learning updates along with the ascending direction of the policy gradient defined in (10). In (10), Q^{π} is approximated as,

$$Q^{\pi}(\vec{s}) \approx \tilde{Q}^{\pi}(\vec{s}) = \vec{v}^{\pi T} \cdot \Phi(\vec{s}, a), \tag{13}$$

where $\vec{v}^{\pi}$ are parameters to estimate estimate the true action value function $Q^{\pi}(\vec{s})$.

[1]Note, the output of ϕ should be multi-dimensional, i.e, $\vec{\phi}$. For simplicity, we will just use ϕ to represent the state feature functions.

$\Phi(\vec{s}, a) = \nabla_{\vec{\theta}} \ln \pi(\vec{s}, a)$ is the compatible feature developed by Sutton in [22] that bridges the critic and the actor. It can be used to form an unbiased estimation of $\nabla_{\vec{\theta}} J(\vec{\theta})$ in (10). By employing the policy gradient's unbiased estimation $\delta_t^{\pi} \Phi(\vec{s}, a)$, the actor learning rule can be determined as,

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \beta_t \delta_t^{\pi} \Phi(\vec{s}, a), \tag{14}$$

where β_t is the actor learning rate.

By iteratively implementing the application of application of (12) and (14), RAC is widely shown to successfully solve many benchmark RL problems [3, 16].

3 A NEAT BASED ACTOR-CRITIC REINFORCEMENT LEARNING ALGORITHM

In this section, we propose a new hybrid algorithm of NEAT and RAC. The algorithm aims to automatically learn state features as well as critic by NEAT and search for suitable policy parameters by RAC simultaneously. We expect to discover good features that can maintain/improve the learning effectiveness.

3.1 Overall Design

In this subsection, we will introduce the four phases of the new NEAT+AC algorithm, including initialization, evolution, evaluation, and termination. Figure 1 shows an overall design of our new NEAT+AC algorithm.

3.1.1 Initialization. Similar to the standard NEAT, NEAT+AC also starts with a population with a fixed number of randomly generated individuals. Each individual is designed differently from that of the standard NEAT. It is composed of three main parts, a neural network, an actor and a critic.

The neural network in the individual is used to represent the state features defined as ϕ in (7). The input of the neural network is the raw state inputs $\vec{s}$ observed from the environment. Its outputs are the high-level state features $\phi(\vec{s})$ with a predefined dimension. The actor here is the parametric function determined by (5) which can be simply represented as a policy parameter $\vec{\theta}$. Similarly, the individual also includes a critic parametric representation (i.e., value function parameters $\vec{\omega}$) as defined in (6).

The ultimate goal of NEAT+AC is to find an optimal solution that contains good features ϕ^* as well as good policy parameters ϕ^*.

3.1.2 Evolution. Aiming at searching good features, the evolution phase of NEAT+AC is to evolve solely the state features $\phi(\vec{s})$ of every individual designed above. In other words, the actor and the critic components in the individual will not get involved in the evolution process, they will be taken care of by the evaluation phase.

We use the standard evolutionary operators defined in the work [19], including both crossover and mutation. We refer readers to [19, 23] for the detailed explanations of the two operators.

3.1.3 Evaluation. The core of NEAT+AC is the evolution phase. In the phase, we aim to accomplish to primary tasks. One is to compute the fitness value for each individual guiding the evolution

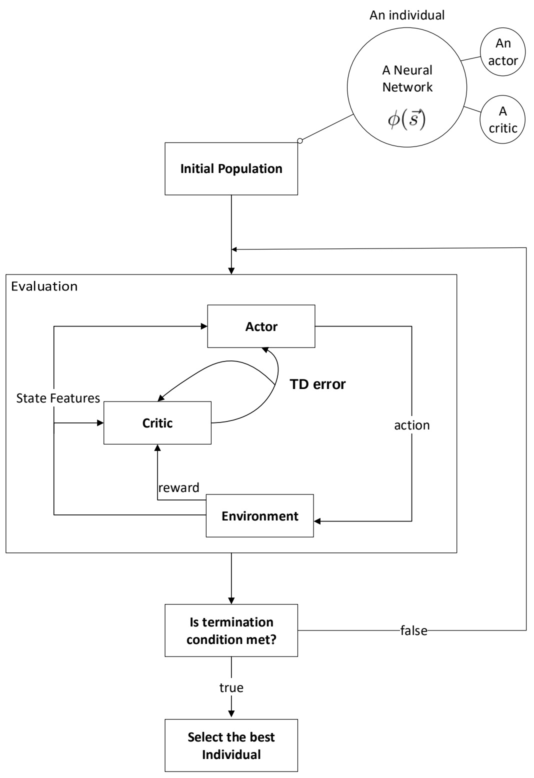

Figure 1: The overall design of NEAT+AC

process search towards the direction of good features, the other is to search optimal policy parameters which is the key for solving the RL problems.

On one hand, the fitness of each individual is straightforwardly defined as an average of the cumulative rewards obtained through simulation over all trained episodes of one generation. This is can be defined as,

$$N.fitness \leftarrow \frac{\tilde{R}}{e_g}. \quad (15)$$

The reason of using the average value is to maintain a fair evaluation of fitness across all individuals over all generations. As during the evolution process, there might be some individuals survive to next generation. Such individuals will be retrained e_g episodes, in total they are trained for $n \times e_g$ episodes where n is the number of generations that passed. However, for those who are newly reproduced from the current generation, they maximally can be trained e_g at the end of current generation. Hence, it is better to use the average value of all cumulative rewards.

On the other hand, the evaluation can also be viewed as a policy direct search process where the policy parameters are incrementally learned and updated towards a local optima. This is done by following the RAC learning process as stated in Section 2.2.2. Note, to better observe the behavior of NEAT+AC, the actor and the critic are always initiated to all zeros at the start of every generation. In doing so, for one particular generation, the evolved features are fixed. At this point, we are able to observe how well AC learning with evolved features can perform.

3.1.4 Termination. Since NEAT+AC consists of two components (NEAT and RAC), each of them has to be set a terminative condition. For NEAT feature learning process, the termination occurs either when the predefined maximum number of generations is reached, or when the fitness value does not improve over 50 generations. The RAC learning process terminates at the condition when the maximum number of interactive episodes is reached. Each episode also has its own termination condition as defined in Section 4.1 for the two benchmark problems respectively.

3.2 Algorithmic Description

In NEAT+AC, NEAT is incorporated with RAC algorithm in several steps. Firstly, we design a different individual from traditional one by coordinating with an actor and a critic. Secondly, we adopt the AC learning process to evaluate fitness of each individual meanwhile seek for optimal policy in each generation. In the following, we present an algorithmic description of NEAT+AC in Algorithm 1.

4 EXPERIMENT DESIGN

To evaluate the effectiveness of our algorithm, the learning performance of NEAT+AC is tested on two benchmark problems and compared against NEAT and NEAT+Q. The section starts with descriptions of the two problems, and then discusses the experimental setups.

4.1 Benchmark Problems

Two continuous benchmark problems, i.e., the Mountain Car problem and the Cart Pole problem, have been chosen in this research. Since they are widely exploited to study any new RL algorithms. Secondly, they have already been demonstrated as beneficial exemplars of using neural network to extracting features from raw state inputs [20, 23].

4.1.1 Mountain Car Problem [20]. The Mountain Car problem, as seen in Figure 2, models a two-dimensional environment where the two dimensions of the state represent the position of the car (i.e., $x \in [-0.5, 0.5]$) and the velocity of the car (i.e., $\dot{x} \in [-0.08, 0.08]$). 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 (i.e., $x \geq 0.5$), and the car aims to use the minimum steps to reach the region. In the problem, the car moves a step meanwhile receives a penalty "-1", a reward "+10" 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., action) performed by the system. Discrete action setting is used for NEAT and NEAT+Q, which is determined as $\{-1.0, 0.0, 1.0\}$. In contrast, NEAT+AC generates

Algorithm 1 NEAT + AC

```
Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
 1: $p$: population size
 2: $m_n$: node mutation rate
 3: $m_l$: link mutation rate
 4: $g$: number of generations
 5: $e_g$: episode per generation
 6: $T$: maximum training steps per episode
 7: $d$: feature dimensionality (the number of NN outputs)
 8: $\alpha$: value function learning rate
 9: $\beta$: policy learning rate
10: $\sigma$: variance for the stochastic Policy
Ensure: $P$ the population of individuals
11: $N^*$ is the optimal Neural Network ($\phi(\vec{s})$),
12: $\vec{\omega}^*$ is the best value function parameter
13: $\vec{\theta}^*$ is the best policy parameter
14: $\tilde{R} \leftarrow 0$
15: Initialization:
16: $P \leftarrow$ INIT-POPULATION($\mathcal{S}, p$)
17: Learning Process:
18: for $i = 1, 2, ..., g$ do
19:     $\vec{\Omega} \leftarrow$ INIT-PARAMETER ($\vec{\omega}_0, p$), $\vec{\omega} \in \mathbb{R}^d$ and $\vec{\Omega} \in \mathbb{R}^{p*d}$
20:     $\vec{\Theta} \leftarrow$ INIT-PARAMETER ($\vec{\theta}_0, p$), $\vec{\theta} \in \mathbb{R}^d$ and $\vec{\Theta} \in \mathbb{R}^{p*d}$
21:     for $k = 1, 2, ..., p$ do
22: ----------------------------------------
23:         $N \leftarrow P[k]$
24:         $\vec{\omega} \leftarrow \vec{\Omega}[k]$
25:         $\vec{\theta} \leftarrow \vec{\Theta}[k]$
26:         $\tilde{R} \leftarrow 0$
27:         for $j = 1, 2, ..., e_g$ do
28:             $\vec{s}_t \leftarrow \vec{s}_0$
29:             for $t = 0, 1, \ldots, T-1$ do
30:                 $a_t \sim \pi_{\vec{\theta}}(a|\vec{s}_t)$ represented by a Gaussian distribution $\mathcal{N}(\vec{\theta} \cdot \phi(\vec{s}_t), \sigma)$.
31:                 Take action $a_t$, observe reward $r_{t+1}$ and new state $\vec{s}_{t+1}$
32:                 $\delta_t \leftarrow r_{t+1} + \gamma \vec{\omega}_t^T \cdot \phi(\vec{s}_{t+1}) - \vec{\omega}_t^T \cdot \phi(\vec{s}_t)$
33:                 $\vec{\omega}_{t+1} \leftarrow \vec{\omega}_t + \alpha \delta_t \cdot \phi(\vec{s}_t)$
34:                 $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta \delta_t \cdot \frac{a_t - \vec{\theta} \cdot \phi(\vec{s}_t)}{\sigma^2} \cdot \phi(\vec{s}_t)$
35:                 if TERMINAL-STATE($\vec{s}_{t+1}$) then
36:                     break
37:                 end if
38:                 $\tilde{R} \leftarrow \tilde{R} + r_{t+1}$
39:             end for
40:         end for
41:         $N.fitness \leftarrow \frac{\tilde{R}}{e_g}$
42: ----------------------------------------
43:     end for
44:     $P \leftarrow$ NEATUpdate($P, p, m_n, m_l$)        ▷ See Algorithm 2
45: end for
46: $N^* \leftarrow P[k^*]$, $P[k^*]$ is the individual with the highest fitness value.
47: $\vec{\omega}^* \leftarrow \vec{\Omega}[k^*]$
48: $\vec{\theta}^* \leftarrow \vec{\Theta}[k^*]$
49: return $N^*, \vec{\omega}^*, \vec{\theta}^*, P$
```

Algorithm 2 NEAT Updating [19, 23]

```
Require: $P, p, m_n, m_l$
Ensure: $P$
 1: function NEATUpdate($P, p, m_n, m_l$)
 2:     $P'[] \leftarrow$ new array of size $p$
 3:     for $k = 1, 2, ..., p$ do
 4:         $P'[k] \leftarrow$ BREED-NET($P[]$)
 5:         if $probability < m_n$ then
 6:             ADD-NOTE-MUTATION($P'$[k])
 7:         end if
 8:         if $probability < m_l$ then
 9:             ADD-LINK-MUTATION($P'$[k])
10:         end if
11:     end for
12:     $P[] \leftarrow P'[]$
13:     return $P$
14: end function
```

continuous action outputs ranging from $[-2.0, 2.0]$ for a consistent comparison to other two algorithms.

Note that, the terminative condition of a learning episode for the Mountain Car problem is defined as either the maximum learning steps or the goal regions is reached.

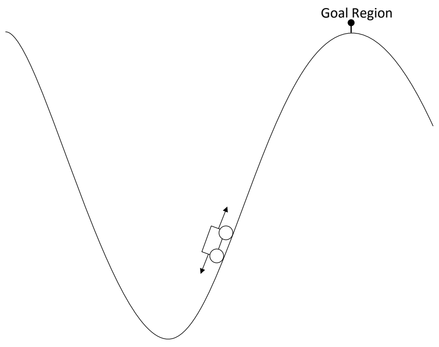

Figure 2: The Mountain Car problem, drawn based on description in [20].

4.1.2 Cart Pole Problem [20]. The Cart Pole problem (a.k.a., the inverted pendulum problem) stands for a classic control problem. In the problem, a learning agent learns to generate an action $\mathcal{F}$ in the form of a horizontal force that drives a cart to move along a fixed length track, i.e, $[-10, 10]$. 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

$$r_{t+1} = \begin{cases} 0.0, & \text{if } |\xi| > 0.628 \text{ or } |x| > 10.0 \\ 0.2\pi - \xi, & \text{otherwise} \end{cases} \quad . \tag{16}$$

In addition, the dynamics of cart and pole can be defined as [4],

$$\ddot{\xi} = \frac{g \cdot sin(\xi) - cos(\xi)(F + M \cdot L \cdot \dot{\xi}^2 sin(\xi))}{\frac{4}{3}L - \frac{m \cdot cos(\xi^2)}{m+M}}$$

$$\ddot{x} = \frac{\mathcal{F} + M \cdot L \cdot \dot{\xi}^2 \cdot sin(\xi) - M \cdot L \cdot \ddot{\xi} \cdot cos(\xi)}{m+M}$$

where $\ddot{\xi}$ and $\ddot{x}$ are the cart moving and the angle changing acceleration respectively. $M = 0.1kg$ represents the pole mass and $m = 1kg$ is the cart mass. Also, $L = 0.5m$ is used to denote the pole length. The system adopts the standard gravity $g = 9.8m/s^2$. Similar to the Mountain Car problem, the $\mathcal{F}$ is the action output from the learning algorithms. It is either discretized as $\{-9.0, -1.0, 0.0, 1.0, 9.0\}$ for NEAT and NEAT+Q, or constrained as continuous values in $[-9.0, 9.0]$ for NEAT+AC.

Moreover, a learning episode for the Cart Pole problem 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.

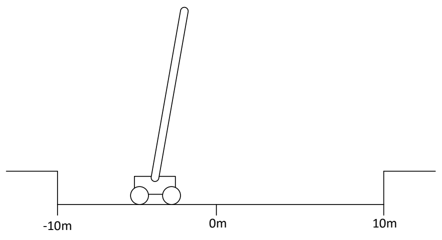

Figure 3: The Cart Pole problem, drawn based on description in [20].

4.2 Experiment Setup

This subsection discusses the experiment setup for this research. Firstly, the common settings for experiments with all learning algorithms are depicted including particularly the implementation of stochastic policies and the settings for the evolutionary process of NEAT. At end of the subsection, all meta parameters used for the work are presented.

4.2.1 Common Settings. To identify any significant performance difference, for each benchmark problem, we conduct 30 independent runs with respect to each learning algorithm. In these runs, the population size and the number of generations are set to 100 for testing all algorithms. Also, for any individual in one single generation, it will perform 200 learning episodes, each of which contains 200 steps while testing on the Mountain Car problem. Similarly, each individual of one generation will learn 100 steps over 5000 learning episodes. At the end of every generation, 50 independent tests are conducted to verify the learning effectiveness based on the evolved NN with the highest fitness. In line with these settings, more detailed settings for each learning algorithm are given below.

4.2.2 Stochastic Policy Implementation. We choose a Gaussian distribution to explicitly represent the stochastic policy used for NEAT+AC, as the distribution has already been well-studied for coping with continuous problems [17]. Based on this policy, the action a to be taken at any state $\vec{s}$ is defined by the following probability density,

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

where $\mu = \vec{\theta}^T \cdot \phi(\vec{s})$. σ is a meta parameter used to control the level to explore new actions, which is fixed to 1.0 for all experiments. Note that, π at the RHS of (17) is the circumference ratio.

4.2.3 Meta Parameter Settings. In this subsection, we summarize important meta parameter settings here. As the baseline algorithms to be compared, the meta parameters for NEAT and NEAT+Q have been set same as those used in [23]. Moreover, the evolution process for NEAT+AC has also adopted the exactly same meta parameter settings. Due to the space limitation, we will not present in this venue but refer readers to the work [23] for more details. In the Table 1, we only present those meta parameters used for the ACRL component in NEAT+AC.

Algorithm	**Problems**	**Meta Parameters**			
		α	β	γ	d
NEAT+AC	**Mountain Car**	0.1	0.01	0.99	10
	Cart Pole	0.1	0.01	0.70	10

Table 1: The meta-parameter settings for experiments of NEAT+AC on the Mountain Car problem and the Cart Pole problem.

5 RESULTS AND DISCUSSION

The experimental results are presented and analyzed in this section. The discussion will be separated based on results obtained from the two benchmark problems. For the discussion, we will primarily focus on comparing the learning performance of the two algorithms. Based on the analysis of performance differences, we will further discuss the behavior of NEAT+AC in comparison with NEAT.

5.1 Experimental Results on Mountain Car

To evaluate the learning performance of NEAT+AC and NEAT, we present the average steps to reach the goal region over 100 generations in Figure 4. These results are obtained from 50 independent tests on the best individual selected at each generation. As seen in the figure, both algorithms converge eventually. NEAT+AC converges at the 10-th generation which is slower than NEAT does at the 2nd generation. In addition to this, to identify the performance difference, we perform a statistical test which gives p-value 0.039. But the performances of two algorithms are not observably different, and only 0.4 average step difference is obtained at the 100-th generation test shown in Figure 5.

Based on these results, we can definitely determine that both algorithms are effective on solving the Mountain Car problem. Also, we believe that the performances are very close. In fact, the Mountain Car problem is not a hard problem with a small value range of state, and it only has two dimensions. This is why both algorithms can effortlessly find a good solution.

Another fact is that NEAT+AC converges slower than NEAT. The reason is because of the difference between discrete actions and continuous actions. Theoretically, it is easier to solve the Mountain Car problem when using a stronger force (i.e., action). For this research, NEAT works as a discrete action selector which has only three choices $-2.0, 0.0, 2.0$, so it will enjoy a high chance (33%) to choose the two stronger actions. However, for NEAT+AC, as we use continuous actions in the range $[-2.0, 2.0]$, it clearly has much less possibilities to reach those action boundaries. It is reasonable that more time (more generations) are needed to learn how to act in such an environment.

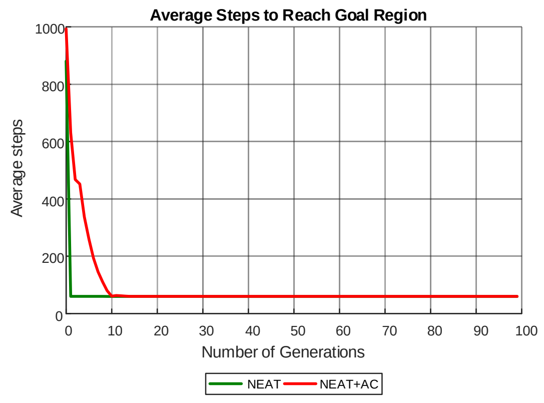

Figure 4: Average Steps Per Generation

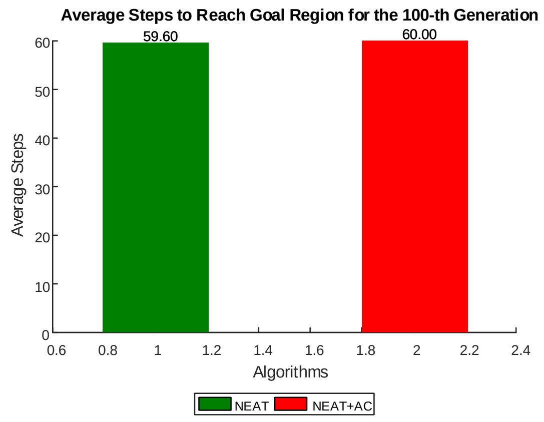

Figure 5: Average Steps for the 100-th Generation

5.2 Experimental Results on Cart Pole

In comparison to those obtained from the Mountain Car problem, very different results are collected on the Cart Pole problem.

Figure 6 presents averaging balancing steps of 50 tests on NEAT and NEAT+AC over 100 generations. As seen in the figure, both algorithms are showing a converging trend, where NEAT+AC clearly is the leading competitor. NEAT+AC has already reached 100 steps at the 4th generation, whereas NEAT is even far away to reach 80 steps at the last generation. We can see from Figure 7 that, at the 100-th generation, NEAT+AC yields 130.4 steps on average, yet NEAT has only 72.37 steps. The statistical test of p-value 1.18366×10^{-11} suggests that NEAT+AC performs significantly better than NEAT on the Cart Pole problem.

Such a performance superiority is also because of that the continuous actions on this hard problem is more suitable than the discrete actions. In fact, the process of solving the Cart Pole problem can be viewed as a careful fine tune process. The problem is very sensitive to the chosen actions. Although the 5 actions we used in this research is widely recognized as a suitable discrete action group to solve the problem, it is still difficult for NEAT learn a good actions combination. In contrast, NEAT+AC naturally uses continuous actions, it owns a wide range of actions to conduct a fine-tune process. Furthermore, NEAT+AC is more guided in comparison to NEAT itself when being applied to RL problems.

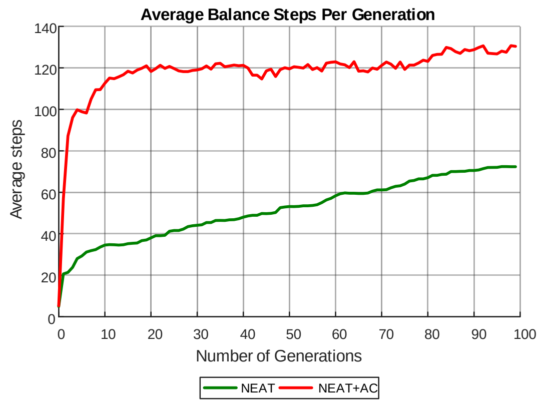

Figure 6: Average Balancing Steps Per Generation

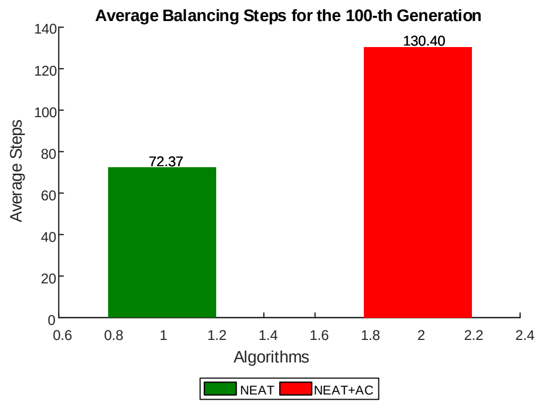

Figure 7: Average Balancing Steps for the 100-th Generation

5.3 Summary

With results of the above experiments, we have achieved our objectives presented in Section 1. With regard to the first objective, the newly proposed algorithm NEAT+AC has shown to be effective on solving both benchmark RL problems, and the gradual improvements on performance show usefulness of learned features. Regarding the second objective, we found that the learning performance of NEAT and NEAT+Q are problem specific. However, neither of the algorithm can achieve the theoretical optimal performance [4]. More experiments on more testbeds are necessary to further evaluate the algorithms. For the last objective, we can clearly observe from experiment results that a relationship exists in-between the learned features and learning performance. As we re-initialize the AC component at every generation, during evaluation process in each generation, only the changes in newly learned features will affect current AC learning performance.

6 CONCLUSIONS

In this paper, guided by the aim of exploiting NEAT for learning state features, we have successfully developed a new algorithm termed NEAT+AC that seamlessly integrated a modern Actor-Critic based reinforcement learning algorithm with the neuroevolution process for tackling challenging reinforcement learning problems. Our algorithm achieved the research goal of evolving useful neural networks as feature extrators which accept raw state information as their input and subsequently produce a vector of numerical state features as the basis of ACRL. Experiments showed that NEAT+AC performed significantly better than NEAT on the Cart Pole problem. On the Mountain Car problem, NEAT+AC also performed highly competitively as NEAT which enjoys the advantage of only considering three optimal actions in each state. On the other hand, NEAT+AC must learn to select the suitable actions from a continuous range, which makes the Mountain Car problem much harder. Based on the experiment results, it is clear that NEAT+AC is an effective algorithm for reinforcement learning. Meanwhile, NEAT+AC is purposefully designed to ensure that every newly evolved neural network will always be trained for the same number of episodes, starting from identical initial settings. In view of this fact, the steady improvement of learning performance during the evolutionary process, as witnessed in our experiments, serves as a solid evidence that NEAT+AC is capable of learning useful state features embodied in the form of neural networks.

There is a big room for future research. Specifically, the effectiveness of NEAT+AC is only verified on two benchmark problems in this paper. We plan to conduct more comprehensive experiments involving a wide range of benchmarks to truly understand the real efficacy of NEAT+AC. We will also study the possibility of exploiting other cutting-edge ACRL algorithms based on the same design principle of NEAT+AC. Moreover, the learning process can be made much more efficient with the help of important sampling techniques. This could further improve the practical usefulness of NEAT+AC.

REFERENCES

[1] Ethem Alpaydin. 2014. *Introduction to machine learning.* MIT press.

[2] Yoshua Bengio, Aaron Courville, and Pierre Vincent. 2013. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35, 8 (2013), 1798–1828.

[3] Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. 2009. Natural actor-critic algorithms. *Automatica* 45, 11 (2009), 2471–2482. DOI: http://dx.doi.org/10.1016/j.automatica.2009.07.008

[4] G Chen, C I J Douch, and M Zhang. 2016. Reinforcement Learning in Continuous Spaces by using Learning Fuzzy Classifier Systems. *IEEE Transactions on Evolutionary Computation* PP, 99 (2016), 1. DOI: http://dx.doi.org/10.1109/TEVC.2016.2560139

[5] Jeff Clune, Kenneth O. Stanley, Robert T. Pennock, and Charles Ofria. 2011. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation* 15, 3 (2011), 346–367. DOI: http://dx.doi.org/10.1109/TEVC.2010.2104157

[6] M P Deisenroth, G Neumann, and J Peters. 2013. *A Survey on Policy Search for Robotics.* https://books.google.co.nz/books?id=hPLGoQEACAAJ

[7] Ivo Grondman, Lucian Busoniu, Gabriel A D Lopes, and Robert Babuška. 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. (2012). DOI: http://dx.doi.org/10.1109/TSMCC.2012.2218595

[8] M. I. Jordan and T. M. Mitchell. 2015. Machine learning: Trends, perspectives, and prospects. *Science* 349, 6245 (2015), 255–260. DOI: http://dx.doi.org/10.1126/science.aaa8415

[9] Vijay R Konda and John N Tsitsiklis. 2003. Actor-Critic Algorithms. *Control Optim* 42, 4 (2003), 1143–1166. DOI: http://dx.doi.org/10.1137/S0363012901385691

[10] George Konidaris, Sarah Osentoski, and Philip Thomas. 2011. Value Function Approximation in Reinforcement Learning using the Fourier Basis. *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence* (2011), 380–385.

[11] Michael L. Littman. 2015. Reinforcement learning improves behaviour from evaluative feedback. *Nature* 521, 7553 (2015), 445–51. DOI: http://dx.doi.org/10.1038/nature14540

[12] Ishai Menache, Shie Mannor, and Nahum Shimkin. 2005. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research* 134, 1 (2005), 215–238. DOI: http://dx.doi.org/10.1007/s10479-005-5732-z

[13] Thomas M Mitchell. 1997. *Machine Learning* (1 ed.). McGraw-Hill, Inc., New York, NY, USA.

[14] Gregory Morse and Kenneth O. Stanley. 2016. Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks. *Gecco* Gecco (2016). DOI: http://dx.doi.org/10.1145/2908812.2908916

[15] Ronald Parr, Christopher Painter-Wakefield, and Lihong Li. 2007. Analyzing feature generation for value-function approximation. *Proceedings of the 24th International Conference on Machine Learning (ICML)* (2007), 737–744. DOI: http://dx.doi.org/10.1145/1273496.1273589

[16] Yiming Peng, Gang Chen, Mengjie Zhang, and Shaoning Pang. 2016. Generalized Compatible Function Approximation for Policy Gradient Search. Springer International Publishing, Cham, 615–622. DOI: http://dx.doi.org/10.1007/978-3-319-46687-3_68

[17] Jan Peters and Stefan Schaal. 2006. Policy gradient methods for robotics. In *IEEE International Conference on Intelligent Robots and Systems.* 2219–2225. DOI: http://dx.doi.org/10.1109/IROS.2006.282564 arXiv:1512.04105

[18] Jan Peters and Stefan Schaal. 2008. Natural Actor-Critic. *Neurocomputing* 71, 7-9 (2008), 1180–1190. DOI: http://dx.doi.org/10.1016/j.neucom.2007.11.026

[19] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural network through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127. DOI: http://dx.doi.org/10.1162/106365602320169811

[20] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement Learning : An Introduction.* DOI: http://dx.doi.org/10.1016/j.brainres.2010.09.091

[21] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, Eric Wiewiora, and Csaba Szepesvari. 2009. Fast Gradient-descent Methods for Temporal-difference Learning with Linear Function Approximation. *Proceedings of the 26th Annual International Conference on Machine Learning* 1, 7 (2009), 993–1000. DOI: http://dx.doi.org/10.1145/1553374.1553501

[22] Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems 12* (1999), 1057–1063. DOI: http://dx.doi.org/10.1.1.37.9714

[23] Shimon Whiteson and Peter Stone. 2006. *Evolutionary Function Approximation for Reinforcement Learning.* Vol. 7. 877–917 pages. http://portal.acm.org/citation.cfm?id=1248578

[24] Shimon Whiteson, Peter Stone, Kenneth O. Stanley, Risto Miikkulainen, and Nate Kohl. 2005. Automatic feature selection in neuroevolution. *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05* (2005), 1225–1232. DOI: http://dx.doi.org/10.1145/1068009.1068210

[25] M Wiering and M van Otterlo. 2012. *Reinforcement Learning: State-of-the-Art.* Springer Berlin Heidelberg. https://books.google.co.nz/books?id=YPjNuvrJR0MC