An Overview of the Action Space for Deep Reinforcement Learning

Jie Zhu
zhujie2020@iscas.ac.cn
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences
Haidian District, Beijing, China

Fengge Wu
fengge@iscas.ac.cn
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences
Haidian District, Beijing, China

Junsuo Zhao
junsuo@iscas.ac.cn
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences
Haidian District, Beijing, China

ABSTRACT

In recent years, deep reinforcement learning has been applied to tasks in the real world gradually. Especially in the field of control, reinforcement learning has shown unprecedented popularity, such as robot control, autonomous driving, and so on. Different algorithms may be suitable for different problems, so we investigate and analyze the existing advanced deep reinforcement learning algorithms from the perspective of action space. At the same time, we analyze the differences and connections between discrete action space, continuous action space and discrete-continuous hybrid action space, and elaborate various reinforcement learning algorithms suitable for different action spaces. Applying reinforcement learning to the control problem in the real world still presents huge challenges. Finally, we summarize these challenges and discuss how reinforcement learning can be appropriately applied to satellite attitude control tasks.

CCS CONCEPTS

• **Computing methodologies → Reinforcement learning**;

KEYWORDS

deep reinforcement learning, action space, satellite attitude control

ACM Reference Format:
Jie Zhu, Fengge Wu, and Junsuo Zhao. 2021. An Overview of the Action Space for Deep Reinforcement Learning. In *2021 4th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI'21), December 22–24, 2021, Sanya, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3508546.3508598

1 INTRODUCTION

In recent years, reinforcement learning has been receiving extensive attention from researchers, and a lot of innovative research has been conducted. Deep reinforcement learning is based on the traditional reinforcement learning algorithm, through the neural network to model the states and actions, so it can efficiently process high-dimensional state space and complex action space. In all problems that require making decision, deep reinforcement learning can maximize the target benefits by interacting with the environment. At present, reinforcement learning has been applied to many problems in the real world, such as robot control[19], intelligent transportation[24], autonomous driving[32], recommendation system[44], etc. Various fields have different requirements for algorithms, which means different tasks emphasize different algorithm characteristics. The field of robot control, autonomous driving tasks and intelligent transportation require algorithms with high safety performance, while recommendation systems require algorithms with diversity and accuracy. Without exception, all tasks hope that the algorithms adopted are better and more stable.

Among these fields, the field of control has always been the main battlefield for reinforcement learning. In order to better apply the reinforcement learning algorithm to the control field, we must have a deeper understanding of the action space of the task. Common action spaces include discrete action space, continuous action space and discrete-continuous hybrid action space. In the real world, there are few tasks that only contain discrete action spaces, and more are continuous action spaces and hybrid action spaces. Discrete action spaces are common in some simple games, such as Atari games[4]. The continuous action space algorithm is more used in the control field, and the algorithm program is required to output the degree of implementation of a certain action, such as the control task of a robotic arm, which requires the algorithm to output the strength or angle of the implementation of the robotic arm. In a more complex environment where the tasks contain discrete and continuous actions, it is necessary to deal with two action dimensions. We call this action space as discrete-continuous hybrid action space, it is also called parameterized action space[25]. This article gives a detailed introduction to model-free algorithms in different action spaces, and analyzes their respective algorithm characteristics. For advanced reinforcement learning algorithms, such as meta reinforcement learning[39], inverse reinforcement learning[31] and transfer reinforcement learning[20], this article will not cover them.

Although reinforcement learning algorithms can play a good role in a large number of simulation environments[7, 33], the development speed is not as fast as expected. The reason we think is that the problems in the real world are often complex, The simulation environment cannot perfectly simulate real tasks. At the same time, the simulation environment has high research costs and long iteration times, which has been an influencing factor hindering the implementation of reinforcement learning for a long time. So we have summarized various challenges faced by reinforcement

ACAI'21, December 22–24, 2021, Sanya, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8505-3/21/12.
https://doi.org/10.1145/3508546.3508598

learning in the real world, and have discussed the issue of satellite attitude control with reinforcement learning.

All in all, The main work of this article is as follows:

(1) We give a detailed introduction to the action space, and analyze their specific application scenarios and common algorithms of different types of action spaces.
(2) We analyze various algorithms in discrete action space, continuous action space and hybrid action space.
(3) We summarize the challenges faced by deep reinforcement learning in the real world, and discuss how to apply them to satellite attitude control tasks.

2 BASICS OF ACTION SPACES

2.1 Differences of action spaces

In a specific reinforcement learning environment, the set of all effective actions of the agent is called action space. The action space must have two properties: completeness and validity. Completeness guarantees the possibility that the agent can achieve the expected goal. If a certain action is missing, the agent has a high probability of failing to complete the task. An example of this is that a car must have the function of accelerating, decelerating, turning and braking to ensure safety. The validity requires that the actions in the action space must be legal, and you can never add the function of flying to the action space of driving car.

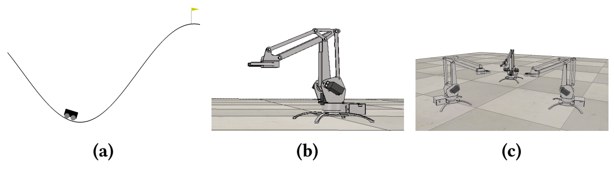

Figure 1: (a) discrete action in MountainCar-V0 game. (b) continuous action of robotic arm. (c) hybrid action composed of multiple robotic arms.

Reinforcement learning may be divided into three main categories according to the action space, which are discrete action space, continuous action space and discrete-continuous hybrid action space. Discreteness and continuity are a relative concept, the former is countable and the latter is uncountable. For discrete actions, we can clearly count the number of actions, so we can use one-hot vectors to indicate whether an action is executed. As shown in Figure 1, in the MountainCar-V0 environment of Atari, there are three actions: push the car to the left, push the car to the right, and stand still. In the continuous action space problem, we cannot count the number of actions, but we can describe their range. This can be illustrated briefly by the task of the robotic arm, the output action of a robot arm includes angles from 0 to 360 degrees and force from 0 to a certain strength. In terms of realization, we can describe the action within a given range by outputting a scalar between 0 and 1. For discrete-continuous hybrid action spaces, it requires the algorithm to output discrete actions and the continuous parameters performed by the discrete actions. In a similar case of multiple robotic arm control tasks, choosing which robotic arm to perform at a certain time can be considered as a discrete action, and then output the continuous angle and force performed by the specific robotic arm.

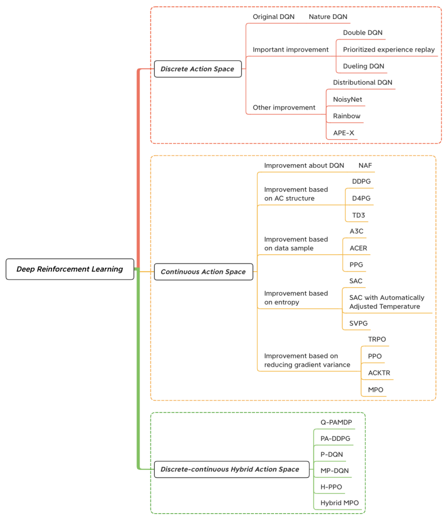
Deep Reinforcement Learning
Discrete Action Space
Original DQN
Nature DQN
Important improvement
Double DQN
Prioritized experience replay
Dueling DQN
Other improvement
Distributional DQN
NoisyNet
Rainbow
APE-X
Continuous Action Space
Improvement about DQN
NAF
Improvement based on AC structure
DDPG
D4PG
TD3
Improvement based on data sample
A3C
ACER
PPG
Improvement based on entropy
SAC
SAC with Automatically Adjusted Temperature
SVPG
Improvement based on reducing gradient variance
TRPO
PPO
ACKTR
MPO
Discrete-continuous Hybrid Action Space
Q-PAMDP
PA-DDPG
P-DQN
MP-DQN
H-PPO
Hybrid MPO

Figure 2: Structure diagram of reinforcement learning algorithm in different action spaces

In reinforcement learning, the algorithms that deal with discrete action spaces are mostly deep Q-learning algorithms (DQN[27]), such as nature DQN[28], double DQN[38], dueling DQN[41], Rainbow DQN[17], NoisyNet[10], etc. Simultaneously, most of the algorithms for processing continuous action spaces are based on policy gradient, which developed into actor-critic structures subsequently, such as TRPO[35], PPO[37], PPG[6], A3C[26], ACER[40], DDPG[21], TD3[11], D4PG[2], SAC[14], MPO[1], etc. Of course, some Algorithms can also be applied to discrete action spaces, which we will talk about in section 5,. For hybrid action space, researchers generally modify the previous algorithms in order to output appropriate actions. At present, there are not much researches in this area, and the main algorithms include Q-PAMDP[25], PA-DDPG[16], P-DQN[43], H-PPO[9], MP-DQN[5], Hybrid MPO[30], etc. Refer to Figure 2 for the specific algorithm structure diagram based on action space. More specifically, we will discuss discrete action algorithms in section 4, continuous action algorithms in section 5, and hybrid action algorithms in section 6.

2.2 Differences in algorithm implementation

In general, action decisions are only related to the state s at the current time t. In terms of specific algorithm implementation, actions are divided into deterministic actions and random actions. For DQN algorithm, the optimal policy will select whichever action maximizes the expected return from starting in s. If we have Q^* function, we can directly obtain the optimal action

$$a^*(s) = \arg\max_a Q^*(s, a). \tag{1}$$

Then we can use ϵ greedy policy to select a random action with probability ϵ otherwise follows the deterministic action, which are conducive to jumping out of the local optimal solution.

In policy gradient algorithm, the deterministic action is usually directly selected according to the policy network. It can be expressed as $a_t = \mu_\theta(s_t)$, where θ is the parameter of the policy network. On the other hand, the random action is to give the execution probability of each action through the policy network, and the action is randomly sampled through this probability distribution. The mathematical expression of action is $a_t \sim \pi_\theta(\cdot|s_t)$. For random strategy in the discrete action space, softmax is usually used to convert the output of the policy network into a probability, so that actions can be sampled under this probability. The random strategy in the continuous action space assumes that the action space obeys a certain distribution, which could be gaussian distribution, multivariate normal distribution and so on. For example, assuming that it obeys the gaussian distribution, the mean and standard deviation of the distribution are output through the policy network. Given the mean action $\mu_\theta(s)$ and standard deviation $\sigma_\theta(s)$ of continuous actions, and a vector z from gaussian noise($z \sim \mathcal{N}(0, I)$), then we can get the continuous action through the following formula:

$$a = \mu_\theta(s) + \sigma_\theta(s) \odot z. \tag{2}$$

3 MARKOV DECISION PROCESS

3.1 Common MDPs

RL problems can be defined as a Markov decision process, and its specific mathematical form is a five-tuple $\langle S, A, R, P, \rho_0 \rangle$, where S is the set of all valid states, A is the action space, and P is a transition probability $p(s_{t+1}|s_t, a_t)$, which represents the probability of going from state s_t to state s_{t+1} under the action of a_t. R indicates that the reward function $r(s, a) \in R$, ρ_0 indicates the initial state distribution. We consider the reinforcement learning problem as finding the optimal strategy π in a reinforcement learning environment with limited discounted returns. Then, we can use $\gamma \in (0, 1)$ represent the limited discount rate, $\pi(a|s, \theta)$ represent the probability of choosing action a in state s under the strategy with the parameter θ, similarly for deterministic actions $\mu_\theta(s)$.

The goal of reinforcement learning is to select the optimal strategy π to maximize the expected return of the agent. For a trajectory $\tau_\pi = \{(s_0, a_0) \ldots (s_T, a_T)\}$ sampled by the strategy π, we can calculate its probability distribution

$$P_\pi(\tau) = p(s_0) \prod_{t>=0} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t). \tag{3}$$

Then the expectation of return is expressed as

$$J(\pi) = \int_\tau P(\tau|\pi)R(\tau). \tag{4}$$

At the same time, the optimal strategy can be expressed as

$$\pi^* = \arg\max_\pi J(\pi). \tag{5}$$

3.2 Parameterized action MDPs

In a discrete or continuous action space, since the types of actions are either discrete or continuous, all actions can be represented by one single variable a. In hybrid action space, we define the action space as a set of discrete actions $A_d = \{a_1, a_2, \ldots, a_k\}, k \in R$. In the formula, each $a \in A_d$ has a continuous action parameter $X_a \subseteq R^{m_a}$. Then each action is a primitive form (a, x), where a represents a discrete action, and x represents the continuous parameter of its action. Then the mathematical form of the hybrid action space is

$$A = \bigcup_{a \in A_d} \{(a, x)|x \in X_a\}. \tag{6}$$

We call the MDPs in the hybrid action space as parameterized action MDPs (PAMDPs).

4 RL OF DISCRETE ACTION SPACE

In the discrete action space, there are two commonly used model-free methods, one is value-based and the other is policy-based. Algorithms based on policy gradient are often not only suitable for discrete action spaces, but also it is used to solve the problem of continuous action space in more situations. The DQN series of algorithms often output discrete actions because the selection of actions is based on the action-value function, so in most cases they are only suitable for discrete action spaces.

We believe that its mainstream problems include the choice of network structure, overestimation of action-value function, sample efficiency, and space exploration. This section will focus on the DQN algorithms in discrete action space.

4.1 Nature DQN

Compared with Q-Learning, Nature DQN[28] has three main improvements. The first change is using a neural network to approximate the action-value function. This way can enable the algorithm to be applied to the situations where the dimension of the state space is larger or the state space is continuous. The second change is using a special target Q network to calculate the target Q value instead of directly using the pre-updated Q network. The loss function is as follows:

$$L = (r + \gamma max_{a'} Q'(s', a', w^-) - Q(s, a, w))^2, \tag{7}$$

where the loss function is mean square error, the Q' represents the target Q function with parameter w^-. The purpose of this is to reduce the correlation between the target calculation and the current value. Finally, the last improvement is using the experience replay mechanism. The experience replay mechanism has always been a major method of the off-policy algorithm, which effectively improves the utilization of data. Experiment results show that DQN surpasses the previous algorithm in most Atari 2600 games, which makes more people devote themselves to the research of deep Q-learning.

4.2 Important improvement

Double DQN. Traditional DQN will generally overestimate the Q value of the action, and the estimation error will increase with the increase of the number of Actions, which could makes the agent unable to find the optimal strategy. Double DQN[38] selects actions through the current Q network, and then finds the Q value of the action as the target value through the target Q network. The loss function is as follows:

$$L = (r + \gamma Q'(s', argmax_{a'} Q(s', a', w), w^-) - Q(s, a, w))^2. \tag{8}$$

In the formula, we can find the difference between Nature DQN and Double DQN is that the target Q network in Double DQN is only used to calculate the target value, and all the places where the optimal action needs to be selected are calculated by current Q network. Therefore, we can say the Double DQN decouples action selection and value estimation.

Prioritized Experience Replay. An important reason for DQN to use experience replay is to disrupt the correlation between data, In the experience replay mechanism, each sample is selected randomly, which means the probability is the same. However, not all samples selected randomly are good samples. The greater the TD error, the greater the effect on our backpropagation. Therefore, Prioritised Replay DQN[34] assigns each sample a priority proportional to the absolute value of its TD error, and stores it through the binary tree structure of sumtree. The probability of sampling transition i is:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{9}$$

where π is the priority of sample i. The hyperparameter α determines the degree of priority, with $\alpha = 0$ equal to the uniform case. Prioritized experience replay greatly improves the convergence speed, and avoids some worthless iterations.

Dueling DQN. The difference between Dueling DQN[41] and DQN lies in the network structure. Dueling DQN considers dividing the Q network into two parts. The first part with parameter α outputs the value of the current state s, which are denoted as $V(s; w, \alpha)$, among them w is the network parameter of the public part. The second part with parameter β is related to the state state s and the action a. This part is called the advantage function part, which are denoted as $A(s, a; w, \beta)$, then our value function can be expressed as follows:

$$Q(s, a; w, \alpha, \beta) = V(s; w, \beta) + \left(A(s, a; w, \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s, a'; w, \alpha)\right). \tag{10}$$

At the same time, in order to solve the unidentifiable problem, it normalized the advantage function. We believe that the advantage of this architecture is the introduction of an advantage function, which greatly improves the stability of the model.

4.3 Other improvement

Distributional DQN. Distributional DQN[3] changes the output of the action-value function from the expectation of the action to the value distribution of the action. In this way, the KL divergence is used to measure the distance between the two distributions, so as to further update the gradient of the model. From the distributional perspective to model the deep Q-learning algorithm, more useful information can be obtained. This reduces overestimation to a certain extent, so as to make the model more stable and robust.

NoisyNet. For efficient exploration, the ϵ greedy strategy is often used in DQN algorithms. The disadvantage of the method is that it is limited to small states and small action spaces. NoisyNet[10] adds gaussian noise to the last layer of the network, and the uncertainty caused by noise is greater than ϵ greedy strategy. The advantage of adding noise is that it allows the agent to decide when and in what proportion to introduce the weight into uncertainty. which greatly improves the efficiency of exploration.

Rainbow. Rainbow DQN[17] integrates and analyzes six effective Q-learning techniques. The six improvements include Double DQN, Dueling DQN, prioritized experience replay, multi-step learning, Distributional DQN, NoisyNet. Experimental results show that the combination provides the most advanced performance of the Atari 2600 benchmark in terms of data efficiency and final performance. Among these improvements, prioritized experience replay and multi-step learning are the two most critical technologies.

APE-X DQN. The traditional DQN has only one learner and one experience replay buffer. The prioritized replay buffer has the greatest impact on performance, while it only uses a single actor to collect data, which is inefficient. APE-X[18] uses the distributed architecture of deep RL on a large scale. It uses hundreds of participants to collect data and apply a large number of replay buffers with different priorities through each participant. Compared with Rainbow, APE-X has doubled its performance in a shorter training time. It not only speeds up the convergence speed, but also effectively reduces the interference of noisy data, making the model more robust. Moreover, it reveals that distributed computing is becoming one of the most important parts of deep RL.

4.4 Summary of DQN algorithms

The DQN algorithms described above can be applied to discrete action spaces, and they have improved DQN in different directions. Referring to Table 1, we summarize the main improvement directions of these algorithms.

Table 1: Different improvement directions of DQN algorithms

Improvement direction	Algorithm
Sample efficiency	Nature DQN Prioritized Experience Replay APE-X
Overestimation	Double DQN Distributional DQN
Network structure	Dueling DQN
Exploration	NoisyNet

Note: Rainbow is an algorithm that integrates multiple tricks.

5 RL OF CONTINUOUS ACTION SPACE

In the previous section, we have introduced many DQN algorithms in discrete action space. However, DQN algorithms have the following inherent limitations:

(1) DQN algorithms are difficult to process continuous action space. The proposal of NAF[13] algorithm extends DQN to continuous action space, but it has not been well promoted.
(2) Its output is deterministic and it cannot effectively solve the problem of random strategy. Although there are ϵ greedy strategy and NoisyNet to improve the exploration space, which should not be a completely random strategy.

(3) The degradation of the strategy can easily cause the algorithm to fail to obtain the optimal solution.

The policy-based algorithm can effectively solve the problems faced by DQN, but also has its own limitations. It is easier to converge to a local optimal value and it is difficult to accurately evaluate the quality of a strategy. Later, the policy-based algorithm evolved into an actor-critic structure. It is worth noting that most algorithms based on policy gradients can also be applied to discrete action spaces, such as A3C[26], TRPO[35], PPO[37], and so on. In this section, we will first introduce the NAF algorithm, and then present the policy-based algorithm in the continuous action space according to different directions of improvement. Specifically, we summarize these improvements into the following four points: improvements based on actor-critic structure, improvements based on data sample, improvements based on entropy, and improvements based on reducing gradient variance.

5.1 DQN in continuous action space

NAF. Since DQN's action-value function outputs the value of each action in the current state, if the continuous action is discretized with a certain accuracy, a large number of discrete actions would be generated. In this case, DQN cannot enumerate all actions. The proposal of NAF[13] provides the possibility for DQN to solve the continuous action problem. The basic idea is to use a network to output both the best action and its Q value. It has a unique structure design for the Q network. The input is still the current state s, and the output is the current state value V, continuous action μ, and a vector L. In order to obtain the Q value, NAF introduces the advantage function. The formula for calculating the Q value is as follows:

$$Q(x,u|\theta^Q) = A(x,u|\theta^A) + V(x|\theta^V), \tag{11}$$

$$A(x,u|\theta^A)) = -\frac{1}{2}(u-\mu(x|\theta^\mu))^T P(x|\theta^P)(u-\mu(x|\theta^\mu)). \tag{12}$$

Where x is the state, u is the action, and μ is the output action of the neural network, $P(x|\theta^P)$ is a state-dependent, positive-definite square matrix, which is parametrized by $P(x|\theta^P = L(x|\theta^P)L(x|\theta^P)^T$. More specifically, L is transformed into a lower triangular matrix, and then the positive-definite matrix P can be obtained by the cholesky decomposition principle from L. Although the NAF algorithm is beautifully designed, it is actually quite complicated to implement.

5.2 Actor-critic structure

The actor-critic model consists of two models: actor and critic. Actor refers to the policy function $\pi_\theta(a|s)$, aimed to get the highest possible return. On the other side, critic refers to the value function $V^\pi(s)$, which estimates the value function based on the current policy, that is, evaluates the quality of the actor. Of course we can choose whether to share the parameters of the shallow network. Except for a few algorithms like DDPG[21] and TD3[11], the actor-critic algorithm can handle both discrete and continuous action spaces. Of course, we can get the advantage function $A^\pi(s,a)$ only through the value function $V^\pi(s)$. This is the idea of Advantage Actor-Critic which we also called A2C[26]. The advantage function can be used as an indicator to measure the difference between the value of the selected action and the average value of all actions.

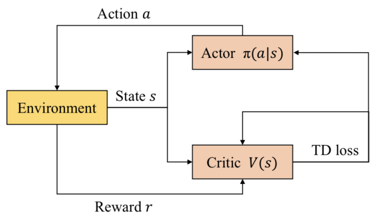

Figure 3: The structure of actor-critic algorithm

However, the advantage function has the drawback of large variance. The proposal of GAE[36] improves the estimation of the advantage function and controls the variance to a certain range. The method is to perform multi-step estimation of the advantage function, and combines these multi-step estimations with attenuation factors. The formula is as follows:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=1}^{\infty}(\gamma\lambda)^l \delta_{t+l}^V = \sum_{l=1}^{\infty}(\gamma\lambda)^l(r_t + \gamma V(s_{t+l+1}) - V(s_{t+l})). \tag{13}$$

Among them, γ controls the return discount rate, while λ controls the step size. We think in this way, GAE achieves a balance between bias and variance, and the estimated advantage that adopts the GAE method can effectively reduce the variance of the gradient estimation, thereby reducing the samples required for training.

5.3 Improvement based on actor-critic structure

DDPG. The algorithm DDPG[21] learns a deterministic policy while extending it to continuous action space through the actor-critic structure. The DDPG algorithm outputs continuous action $a_t = \mu(s_t|\theta^\mu)$ through the actor network, and uses the action-value function to assist the update of the policy. Also, it defines the behavior strategy β firstly, and explores potential better policy by adding noise to the output of the actor network, so as to get the data set required for training. Then update the deterministic policy μ through the Q network, and only use the actor network μ to make decisions during the testing and verification process. In summary, DDPG is an extension of DQN in the continuous action space and can only be used for deterministic continuous actions.

D4PG. Distributed Distributional DDPG (D4PG)[2] has made a series of improvements on the DDPG algorithm. The first improvement is that it uses distributed critics, which means it no longer only estimates the expected value of action-value function, but estimates the distribution of expected Q values. The idea is the same as that of Distributed DQN[3]. The second improvement is that multiple distributed parallel actors are used in this structure. It uses k independent actors to collect training samples in parallel and stores them in the same prioritized experience replay buffer[34]. It is obvious that D4PG is an extended version of DDPG and has greatly improved performance. The distributional critic can prevent the overestimation of the value function while the distributed parallel actor greatly improves the sampling efficiency of samples. Same as DDPG, it is only suitable for continuous action space.

TD3. It is well known that Q-learning has always had the problem of overestimating the value function Q. Overestimation will continue to spread with the training process and eventually will have a negative impact on policy learning. Twin Delayed Deep Deterministic(TD3)[11] applies many new improvements on the basis of the DDPG algorithm to prevent the overestimation of the value function. Firstly, TD3 uses the minimum value of the two Q networks for estimation, which tends to use underestimated deviations that are difficult to propagate through training. Secondly, it updates the target and policy network at a very low frequency. aimed to reduce the variance and avoid the inaccuracy of the value function or the poor strategy. Finally, it uses target policy smoothing regularization. We believe that TD3 makes the algorithm update more stable through unique structure design, while ensuring that it has a certain exploration ability.

5.4 Improvement based on data sample

A3C. The purpose of experience replay in DQN is to disrupt the correlation between data, so that the data satisfies independent and identical distribution. However, the interaction between the agent and the environment requires a lot of memory and computing power; and also the experience replay mechanism requires the agent to adopt an off-policy method for learning. A3C[26] uses a multi-process method to collect data asynchronously, and updates the policy by merging the global gradient, which greatly improves the data sampling rate of the on-policy algorithm, and reduces the impact of poor samples.

ACER. Actor-Critic with Experience Replay(ACER)[40] is the off-policy version of A3C algorithm, The main obstacle is how to ensure the stability of the off-policy estimator. First of all, ACER use a more efficient TRPO algorithm which uses the first derivative of the KL divergence as a constraint. Secondly, the estimation of Q value uses Retrace[29] technology. Last but not least, ACER use bias correction parameter c to cut off the importance weight:

$$\bar{\rho_t} = \min(c, \rho_t), \rho_t = \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}, \tag{14}$$

where $x_t \sim \beta$ and $a_t \sim \mu$. All in all, ACER greatly improves the effectiveness of sampling and reduces the correlation between training data.

PPG. There are pros and cons to share parameters between policy and value network. It allows policy and value function to share information with each other, but it may cause conflicts between competing goals and require the same data to train two networks at the same time. The phasic policy gradient(PPG)[6] modifies the traditional PPO[37] algorithm to conduct separate training for policies and value functions. It means that PPG algorithm decouples the training of actor and critic. Since the policy and the value function can be trained separately, the update frequency of the two can be adjusted in each cycle, so that the sample utilization rate can be improved.

5.5 Improvement based on entropy

Entropy is a rough measure of the uncertainty of random variables. It means that the entropy of a certain event is relatively low. Suppose that the probability of occurrence of an event X is P. Then, the entropy H of X is calculated from its distribution P according to

$$H(X) = \sum_{x\sim P}(-\log P(x)). \tag{15}$$

This is closely related to the trade-off between exploration and exploitation. Increasing entropy will lead to more exploration, which can prevent the policy from prematurely converging to a bad local optimum.

SAC. SAC[14] makes the goal of RL become both to maximize the return and maximize the entropy of the selected action. Then, the optimal policy is calculated by the following formula:

$$\pi^* = \arg\max_{\pi} E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t))\right)\right]. \tag{16}$$

It trains random policy through entropy regularization and explores them in a policy-based way. The temperature parameter α determines the relative importance of the entropy term against the reward. When α equals to 0, it becomes to the conventional policy-based algorithm. The higher α means more exploration, and the lower α means more exploitation. We believe that the introduction of entropy regularization provides greater possibilities for the exploration space of RL.

SAC with Automatically Adjusted Temperature. The adjustment of hyperparameter α has always been an urgent problem to be solved. SAC with automatically adjusted temperature[15] uses a neural network to learn the setting of α while maximizing the target reward automatically. This can solve the complexity and inaccuracy caused by manual adjustment.

SVPG. Stein variational policy gradient algorithm(SVPG)[23] applies the Stein variational gradient descent method(SVGD)[22] to update the policy parameter θ. SVPG considers the policy parameter θ as a random variable and seeks a distribution $q(\theta)$ to maximize the expected return:

$$\hat{J}(\theta) = E_{\theta\sim q}[J(\theta)] - \alpha D_{KL}(q\|q_0). \tag{17}$$

If we consider q_0 as a uniform distribution and set $q_0(\theta)$ to a constant, then the above objective function becomes SAC. SVPG assumes that the optimal distribution of θ is $q(\theta) \propto \exp\left(\frac{1}{\alpha}J(\theta)\right)q_0(\theta)$, where $q(\theta)$ can be seen as the posterior distribution; $\exp\left(\frac{1}{\alpha}J(\theta)\right)$ is the likelihood function; $q_0(\theta)$ is the prior distribution. The key of SVPG is to use the SVGD method to estimate the target posterior distribution $q(\theta)$, thereby improving the efficiency of exploration.

5.6 Improvement based on reducing gradient variance

TRPO. A small difference in parameter can make a big difference in performance. This makes it dangerous to use large step sizes with standard policy gradient. TRPO[35] uses KL divergence to constrain the distance between the new and old policies while optimizing the optimal target:

$$\begin{aligned}\theta_{k+1} &= \arg\max_{\theta}\mathcal{L}(\theta_k, \theta)\\ &\text{s.t.}\bar{D}_{KL}(\theta||\theta_k) \leq \delta.\end{aligned} \tag{18}$$

Intuitive explanation is that TRPO keeps new and old policies close in parameter space. A deep understanding is that it avoids policy collapse caused by poor sampling.

PPO. PPO[37] is motivated by the same questions as TRPO. The most popular idea of PPO is that it relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy. The optimization goal is as follows:

$$L(s, a, \theta_k, \theta) = \min(r(\theta)A^{\pi_{\theta_k}}(s, a), clip\,(r(\theta), 1-\epsilon, 1+\epsilon)\,A^{\pi_{\theta_k}}(s, a)), \tag{19}$$

where $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$. Through the clip operation, the weight $r(\theta)$ is constrained between $1-\epsilon$ and $1+\epsilon$. PPO-Clip has no KL divergence term on the target, and no constraints at all. while TRPO attempts to use complex second-order methods to solve this problem, PPO is a family of first-order methods that uses simple techniques to make the new policy close to the old policy. The PPO method is significantly easier to implement, and empirically, its performance is comparable to TRPO.

ACKTR. Actor-Critic algorithm using Kronecker-Factored Trust Region (ACKTR)[42] uses Kronecker-Factored Approximated Curvature(KFAC) [12] to the fisher matrix to perform efficient approximate natural gradient updates. It adds a KL loss to the objective function as a soft constraint, so KFAC algorithm can take advantage of its speed in calculating gradients. ACKTR is a highly efficient algorithm that can greatly increase the amount of information that the model obtains when it is updated. Compared with TRPO and PPO, ACKTR achieves faster training speed.

MPO. MPO[1] is a RL algorithm based on the Expectation Maximization (EM) algorithm. It aims to maximize a posterior policy optimisation, and it assumes the following distribution:

$$p(O=1|\tau) \propto \exp(\sum_t r_t/\alpha). \tag{20}$$

where α is a hyperparameter, $O = 1$ refers to the action chosen to maximize the return of the corresponding trajectory. The goal is to maximize the probability of the event $O = 1$:

$$\begin{aligned}\log p_\pi(O=1) &= \log \int p_\pi(\tau)p(O=1|\tau)\,d\tau \\ &\geq \int q(\tau)[\log p(O=1|\tau) + \log \frac{p_\pi(\tau)}{q(\tau)}]\,d\tau \\ &= E_q[\sum_t r_t/\alpha] - KL(q(\tau)\|p_\pi(\tau)) \\ &= J(q,\pi).\end{aligned} \tag{21}$$

Where MPO uses an auxiliary distribution q to replace the real distribution p, and uses evidence lower bound (ELBO) to get the above lower bound. So we can use the EM algorithm to solve this problem, the E-step optimizes J by updating q, and M-step optimizes J by updating parameterized π, thereby iteratively solving the optimal target value. Compared with PPO and TRPO, MPO is more novel and performs better. Also it is worth noting that it is an off-policy algorithm.

5.7 Summary of algorithms in continuous action space

In addition to NAF, the other algorithms in the continuous action space are all based on the actor-critic structure. At the same time, most of the algorithms can also be applied to discrete action spaces, such as A3C, PPO, etc. and some algorithms like SAC can also handle discrete actions after certain processing of the action output. In short, we summarize the applicable action space of different algorithms in the following table:

Table 2: Differences in Continuous Action Space Algorithms

Algorithms	Algorithm type	Applicable type of action space
NAF	off-policy	continuous
DDPG	off-policy	continuous
D4PG	off-policy	continuous
TD3	off-policy	continuous
A3C	on-policy	continuous, discrete
ACER	off-policy	continuous, discrete
PPG	on-policy	continuous, discrete
SAC	off-policy	continuous
SAC with Automatically Adjusted Temperature	off-policy	continuous
SVPG	on-policy	continuous, discrete
TRPO	on-policy	continuous, discrete
PPO	on-policy	continuous, discrete
ACKTR	on-policy	continuous, discrete
MPO	off-policy	continuous, discrete

6 RL OF HYBRID ACTION SPACE

6.1 Universal method

In order to apply the commonly used DRL algorithm to discrete-continuous hybrid action space, there are usually two most straightforward approaches. The action space can refer to Formula 6:

(1) Approximate A through a finite discrete set. For each X_a, we can discretize it within the range of X_a by setting a certain precision. But in the other side, we may lose the natural structure of X_a.
(2) Relax A into a continuous set. It means the discrete action in A_d is processed by continuous parameters. Approximate form is as follows:

$$\tilde{A} = \bigcup_{a\in A_d} \{(f_{1:|A_d|}, x_{1:|A_d|})|f_a \in F_a, x_a \in X_a\}, \tag{22}$$

where $F_a \in R$, $f_1, f_2, \ldots, f_{|A_d|}$ is used to select discrete actions either deterministically ($argmax_a(f_a)$) or randomly ($softmax(f)$). Compared with the original action space A, it makes the algorithm more complicated.

6.2 Important algorithms

Q-PAMDP. Given a predefined parameterized action space, Q-PAMDP [25] focuses on learning how to choose an action with continuous parameter. The algorithm alternately trains the action selection process and the continuous parameter selection process. It first selects the discrete action through the Q function, and then selects the continuous parameter for the discrete action through the policy network. The main algorithms include Q-PAMDP(1) and Q-PAMDP(∞). The parameters in parentheses indicate the update frequency of the policy network in each iteration. Q-PAMDP theoretically proves that it can converge to the local or global optimum by alternate training.

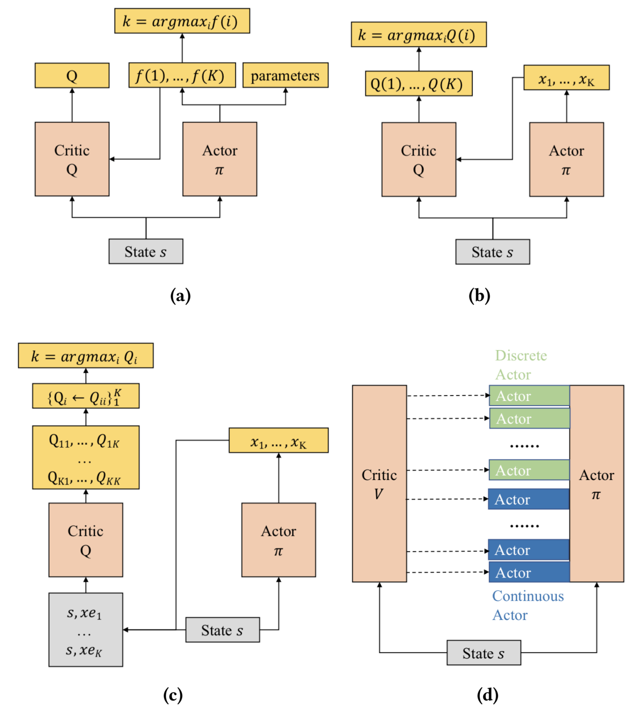

Figure 4: (a) Architecture of PA-DDPG. (b) Architecture of P-DQN. (c) Architecture of MP-DQN. (d) Architecture of H-PPO.

PA-DDPG. PA-DDPG[16] extends DDPG[21] to the discrete-continuous hybrid action space, and restricts the gradient bounds of the action space. First of all, it relaxs action space A into a continuous set so that it can handle hybrid action. Its network structure is shown in the Figure 4(a). In addition, since continuous actions have a certain range, PA-DDPG must limit the gradient of the action space. It provides three methods to solve the gradient boundary problem, which are zeroing gradients, squashing gradients, inverting gradients. Although the methods are different, their purpose is to update the gradient within the allowable range of the continuous parameters. It should be noted that it also proposes three gradient boundary processing schemes, which can be used in other algorithm. The proposal of PA-DDPG provides an alternative method for the algorithm of hybrid action space.

P-DQN. Parametrized deep Q network(P-DQN)[43] proposes a framework that can directly work on the discrete-continuous hybrid action space without approximation or relaxation. It can be seen as an extension of the well-known DQN algorithm to the hybrid action space. The algorithm structure is shown in Figure 4(b). Similar to the deterministic policy gradient algorithm, the algorithm first defines a deterministic function to map the state to the continuous parameters corresponding to each discrete action. Then the state and continuous action parameters are mapped to a Q value through the Q network, and finally the specific discrete action is selected by selecting the optimal Q value. Although P-DQN can converge and the effect is very good, it still needs more researches in the logic of discrete actions selection and continuous actions selection.

MP-DQN. Multi-pass deep Q network (MP-DQN)[5] solves the problem of excessive parameterization of P-DQN by using several parallel batch processing to assign action parameters to the Q network. Specifically, updating the continuous action parameter of any action will disrupt the Q value of all actions, not just the Q value associated with the action parameter. Referring to Figure 4(c), MP-DQN separates the continuous parameters and inputting them into the Q network separately without changing the P-DQN structure. Then we can get a matrix of Q values and use $\arg\max_i Q_i$ to select the optimal action. In my opinion, MP-DQN reduces the influence of a single discrete action on other continuous action parameters, which is more helpful for selecting the optimal hybrid action.

H-PPO. H-PPO[9] proposes an algorithm based on actor-critic structure in hybrid action space, which is an implementation based on PPO[37] architecture. Referring to Figure 4(d), it contains multiple parallel sub-actor networks to process a single discrete action and its continuous actions, and it also has a global critic network to update policies. The biggest contribution of this algorithm is to propose a novel actor-critic network structure to handle hybrid action. Its actors can output discrete actions and continuous actions at the same time, and all parameterized actions are judged through the critic network, which has a certain design rationality.

Hybrid MPO. Hybrid MPO[30] defines a hybrid policy $\pi(a|s)$ as a state dependent distribution that jointly models discrete and continuous random variables. Moreover, it assumes independence between action dimensions a^i, then,

$$\pi_\theta(a|s) = \prod_{a^i \in a^C} \pi_\theta^c(a^i|s) \prod_{a^i \in a^D} \pi_\theta^d(a^i|s), \tag{23}$$

where a^C is the sub-set of discrete actions and a^D is the sub-set of continuous actions. Then, Hybrid MPO assumes the continuous action policy $\pi_\theta^c(a^i|s)$ as a normal distribution with mean $\mu_{i,\theta}(s)$ and standard deviation $\sigma_{i,\theta}(s)$:

$$\pi_\theta^c(a^i|s) = \mathcal{N}(\mu_{i,\theta}(s), \sigma_{i,\theta}^2(s)), \tag{24}$$

and the discrete action $\pi_\theta^d(a^i|s)$ as a categorical distribution parameterized by state-dependent probabilities $\alpha_\theta(s)$:

$$\pi_\theta^d(a^i|s) = Cat^i(\alpha_\theta^i(s)), \forall i, s \sum_{k=1}^{K} \alpha_{k,\theta}^i(s) = 1, \tag{25}$$

where K is the number of discrete actions. When updating the model gradient, Hybrid MPO adopts the same training method based on the EM algorithm as MPO.

7 CHALLENGES AND THE APPLICATION OF RL IN REAL WORLD

7.1 Challenges

Reinforcement learning has been proven to be effective in a large number of simulation environments, but the problems in the real world have been handled much more slowly. We believe that this is due to the big difference between the current experimental RL settings and the poorly defined reality in the real world. For many real systems, not only is there no existing simulator, but it can be very difficult to build one. The lack of available simulators means that data from the real system must be used to complete the learning, and all operations and explorations must be completed on the real system. Although some researchers[8] have analyzed the challenges in the real world, we think that some challenges can be summarized and sorted out. So we summarize the challenges into the following five points:

1. The problem of data. Firstly, it is hard to learn a RL algorithm in the real world with limited sample. Moreover, it is difficult to learn the decision-making information needed by RL from other external behavioral strategies.
2. Algorithm limitations. A large number of offline samples cannot be used, and most algorithms require online learning.
3. The problem of algorithm design. How to design the state, action and reward in the algorithm still needs to be studied in depth, especially when the task has high-dimensional state, high-dimensional action space or sparse reward space.
4. Security issues. Security issues have always been one of the important issues to be solved in the control field. Once safety is not guaranteed, it will cause very serious consequences.
5. Hardware limitations. There will always be a time delay in the hardware, or some instrument error.

7.2 Application

This article discusses the model-free algorithm of reinforcement learning in different action spaces, with the purpose of applying reinforcement learning to the attitude control of satellites. Satellite attitude refers to the state of the space pointing in the orbit of the satellite orbit. The origin of the cartesian coordinate system is placed on the star, the Z-axis pointing to the ground reflects the yaw direction, the Y-axis reflects the pitch direction, and the X-axis reflects the rolling direction. Under normal circumstances, we will use momentum wheel and magnetic torque device for three-axis satellite attitude control.

The traditional satellite attitude is controlled by the basic proportional-integral-derivative(PID) algorithm, and the PID algorithm requires a lot of human tuning and human experience, so we consider applying RL to satellite attitude control tasks. Satellites usually need to perform specific tasks through their own rotation. Assuming that the satellite is composed of three momentum wheels to complete three-dimensional spin, then we can use algorithms in continuous action space to process, such as PPG, MPO, ACER, and so on. This is just an ideal situation. Normally, due to the limited life of the momentum wheel, We usually use a certain number of momentum wheel on different axes to perform tasks, which requires us to choose different momentum wheels for different coordinate axes. So it is necessary to define it as a hybrid action space. The use of each momentum wheel is regarded as a discrete action, and the input of each momentum wheel is a continuous parameter. Advanced algorithms such as H-PPO and H-MPO can be selected for satellite attitude control tasks. we leave this investigation as future work.

8 CONCLUSION

In this survey, we deeply explore and analyze the action space of reinforcement learning, and divide it into three categories: discrete action space, continuous action space and discrete-continuous hybrid action space. Then, we carefully introduce and summarize the model-free algorithm in different action spaces. Finally, we briefly analyze how to apply RL algorithms in satellite attitude control tasks. The challenges of applying reinforcement learning in the real world are still huge. For control problems in the real world, we hope that the analysis of the action space in this article can provide help for other researchers to carry out scientific research tasks.

REFERENCES

[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. 2018. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920* (2018).

[2] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617* (2018).

[3] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*. PMLR, 449–458.

[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[5] Craig J Bester, Steven D James, and George D Konidaris. 2019. Multi-pass q-networks for deep reinforcement learning with parameterised action spaces. *arXiv preprint arXiv:1905.04388* (2019).

[6] Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. 2020. Phasic policy gradient. *arXiv preprint arXiv:2009.04416* (2020).

[7] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).

[8] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. 2019. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901* (2019).

[9] Zhou Fan, Ruilong Su, W. Zhang, and Y. Yu. 2019. Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space. In *IJCAI*.

[10] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. 2017. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295* (2017).

[11] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*. PMLR, 1587–1596.

[12] Roger Grosse and James Martens. 2016. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*. PMLR, 573–582.

[13] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. 2016. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*. PMLR, 2829–2838.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*. PMLR, 1861–1870.

[15] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[16] M. Hausknecht and P. Stone. 2016. Deep Reinforcement Learning in Parameterized Action Space. *CoRR* abs/1511.04143 (2016).

[17] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[18] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. 2018. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933* (2018).

[19] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

[20] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17, 1 (2016), 1334–1373.

[21] T. Lillicrap, Jonathan J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2016).

[22] Qiang Liu and Dilin Wang. 2016. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. In *NIPS*.

[23] Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. 2017. Stein variational policy gradient. *arXiv preprint arXiv:1704.02399* (2017).

[24] Patrick Mannion, Jim Duggan, and Enda Howley. 2016. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. *Autonomic road transport support systems* (2016), 47–66.

[25] Warwick Masson, Pravesh Ranchod, and George Konidaris. 2016. Reinforcement learning with parameterized actions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[26] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[29] R. Munos, Tom Stepleton, A. Harutyunyan, and Marc G. Bellemare. 2016. Safe and Efficient Off-Policy Reinforcement Learning. In *NIPS*.

[30] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. 2020. Continuous-discrete reinforcement learning for hybrid control in robotics. In *Conference on Robot Learning*. PMLR, 735–751.

[31] Andrew Y Ng, Stuart J Russell, et al. 2000. Algorithms for inverse reinforcement learning.. In *Icml*, Vol. 1. 2.

[32] Matthew O'Kelly, Aman Sinha, Hongseok Namkoong, John Duchi, and Russ Tedrake. 2018. Scalable end-to-end autonomous vehicle testing via rare-event simulation. *arXiv preprint arXiv:1811.00145* (2018).

[33] OpenAI. 2018. OpenAI Five. https://blog.openai.com/openai-five/.

[34] Tom Schaul, John Quan, Ioannis Antonoglou, and D. Silver. 2016. Prioritized Experience Replay. *CoRR* abs/1511.05952 (2016).

[35] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. PMLR, 1889–1897.

[36] John Schulman, P. Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR* abs/1506.02438 (2016).

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[38] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[39] Joaquin Vanschoren. 2018. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* (2018).

[40] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. 2016. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224* (2016).

[41] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1995–2003.

[42] Yuhuai Wu, Elman Mansimov, Roger B. Grosse, Shu Liao, and Jimmy Ba. 2017. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *NIPS*.

[43] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. 2018. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394* (2018).

[44] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.