Deep Reinforcement Learning: A Survey

Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, *Senior Member, IEEE*, Bin Dai, and Qiguang Miao, *Senior Member, IEEE*

***Abstract*— Deep reinforcement learning (DRL) integrates the feature representation ability of deep learning with the decision-making ability of reinforcement learning so that it can achieve powerful end-to-end learning control capabilities. In the past decade, DRL has made substantial advances in many tasks that require perceiving high-dimensional input and making optimal or near-optimal decisions. However, there are still many challenging problems in the theory and applications of DRL, especially in learning control tasks with limited samples, sparse rewards, and multiple agents. Researchers have proposed various solutions and new theories to solve these problems and promote the development of DRL. In addition, deep learning has stimulated the further development of many subfields of reinforcement learning, such as hierarchical reinforcement learning (HRL), multiagent reinforcement learning, and imitation learning. This article gives a comprehensive overview of the fundamental theories, key algorithms, and primary research domains of DRL. In addition to value-based and policy-based DRL algorithms, the advances in maximum entropy-based DRL are summarized. The future research topics of DRL are also analyzed and discussed.**

***Index Terms*— Deep learning, deep reinforcement learning (DRL), imitation learning, maximum entropy deep reinforcement learning (RL), policy gradient, value function.**

I. Introduction

REINFORCEMENT learning (RL) is one of the subdomains of machine learning. The goal is to let the agent learn how to act based on the environment state to maximize the expected long-term rewards, where the learning problem can usually be modeled as Markov decision problems (MDPs) [1]. Fig. 1 shows the interactive feedback loop between the agent and the environment. Early RL research mainly focused on tabular and approximation-based algorithms [2], [3], [4], [5], [6]. Due to the lack of representation ability, traditional RL algorithms can only solve tasks with low-dimensional state and action spaces. However, tasks that are more complex and closer to the real-world situations usually have a higher dimensional state space and continuous action space, thus limiting the application of RL [7], [8].

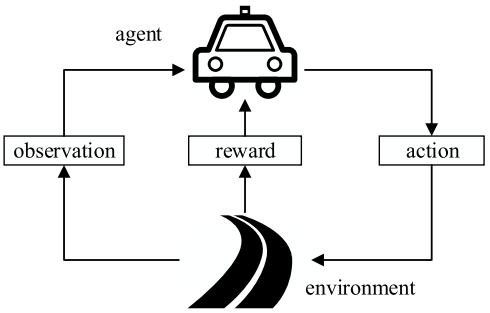

Fig. 1. Interaction between the agent and the environment: at each time step, after the agent observes the environment, it chooses an action according to its policy. After the action is executed, the environment gives a reward signal to the agent and transit to a new state.

The achievements in image classification [9], [10], [11], natural language processing (NLP), and other fields have shown that deep learning has powerful representation capabilities, by extracting multilevel features from high-dimensional abstract inputs (such as images) [12]. Moreover, deep neural networks have been proven to be general function approximators [13], [14], [15], which can be used to approximate value functions and policies in complex tasks with high-dimensional input. Therefore, deep reinforcement learning (DRL)has received much attention in recent years.

Since the successful application of deep Q-network (DQN) in game playing [8], more and more deep learning techniques and algorithms have been combined with RL, which are used to solve not only difficult conventional RL tasks but also inspires new research fields (meta DRL, transfer DRL *et al.*). Not only in theory, DRL has also made some important advances in applications, such as robot control [16], [17], [18], [19], games [20], [21], [22], [23], [24], NLP [25], [26], [27], autonomous driving [28], [29], [30], recommendation system [31], [32], and computer vision [33], [34], [35].

What's more, many research teams have built different open-source DRL algorithm libraries including baselines [36],

Manuscript received 10 June 2020; revised 14 December 2020, 15 June 2021, and 17 October 2021; accepted 29 August 2022. Date of publication 28 September 2022; date of current version 5 April 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFC0807500; in part by the National Natural Science Foundations of China under Grant 61772396, Grant 61772392, Grant 61902296, and Grant 61825305; in part by the Xi'an Key Laboratory of Big Data and Intelligent Vision under Grant 201805053ZD4CG37; in part by the National Natural Science Foundation of Shaanxi Province under Grant 2020JQ-330 and Grant 2020JM-195; in part by the China Postdoctoral Science Foundation under Grant 2019M663640; and in part by the Guangxi Key Laboratory of Trusted Software under Grant KX202061. *(Corresponding author: Qiguang Miao.)*

Xu Wang, Sen Wang, and Qiguang Miao are with the Xi'an Key Laboratory of Big Data and Intelligent Vision, Xidian University. Xi'an 710000, China (e-mail: xuwangxw@foxmail.com; xidian_wangsen@126.com; qgmiao@xidian.edu.cn).

Xingxing Liang and Jincai Huang are with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410000, China (e-mail: doublestar_l@163.com; huangjincai@nudt.edu.cn).

Dawei Zhao and Bin Dai are with the National Innovation Institute of Defense Technology, Beijing 100071, China (e-mail: zhaodawei12@nudt.edu.cn; bindai.cs@gmail.com).

Xin Xu is with the College of Intelligence Science, National University of Defense Technology (NUDT), Changsha 410000, China (e-mail: xinxu@nudt.edu.cn).

Digital Object Identifier 10.1109/TNNLS.2022.3207346

coach [37], tf_agents [38], Tianshou [39]. These libraries provide performance analysis tools and can be easily integrated into more complex DRL agents, making contributions to the development of the DRL community.

Previous survey papers summarized the development of DRL in various aspects [40], [41], [42], [43]. Compared with them, the contributions of this article includes the following three aspects. First, based on the different learning objectives, we classify DRL algorithms into three categories and clearly show the relationships among these three classes of DRL methods: value-based, policy-based, and maximum entropy-based algorithms. Second, the commonly used DRL algorithms that have been implemented in different code bases are analyzed in detail. Finally, further research topics that focus on solving different challenges of conventional RL and DRL are analyzed and discussed. The following sections begin with the background of RL and deep learning. Sections III and IV, respectively, introduce the value-based and policy-based algorithms. Maximum entropy DRL algorithms are presented in Section V. Several directions that have gained recent attention and their primary progress are discussed in Section VI.

II. Background

The basic knowledge of RL is covered in this section, including the basic framework for solving RL tasks and algorithms based on MDP, dynamic programming, Monte Carlo (MC), and temporal difference (TD) methods. This section also includes some features of deep learning used in conjunction with RL.

A. Reinforcement Learning

1) Markov Decision Process: MDP [1], [44] is a classic framework for solving sequential decision-making problems. MDP is based on the following assumptions. 1. The environment is Markovian, which means the state at the next time step is determined only by the current state, independent of the previous states. 2. The environment is fully observable. That is, the agent can observe all the environmental information at any moment. This assumption is inappropriate in some context, so many variants of MDP have been proposed, such as partially observable MDP [45].

MDP can be expressed as a quintuple (S, A, ρ, f, γ), where it is mentioned as follows.

1) S is a set of all observable states.

2) A is a set of actions. The actions can be discrete or continuous. If the state changes, the agent will sample the next action according to a certain probability distribution, which is called the policy: $\pi{:}S \rightarrow A$. The state and action trajectory obtained by the interaction is expressed as $\tau{:}(s_0, a_0, s_1, a_1, \ldots, a_{t-1}, s_t)$.

3) $\rho{:}S \times A \rightarrow R$ is the immediate scalar reward.

4) $f{:}S \times A \times A \rightarrow [0, 1]$ is called the state transition function. $f(s, a, s') = P(s'|s, a)$ is the probability that state s transits to s' after action a is performed.

5) $\gamma \in [0, 1]$ is a discount factor whose intention is to reduce the impact of future rewards on the present.

MDP is a popular mathematical form of RL tasks [3]. The goal of an RL task under MDP is to find the trajectories that can maximize the cumulative rewards, which is expressed as the sum of all discounted rewards starting from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

where R_i is the immediate reward at time step i and G_t is called the return.

2) Bellman Equations: Due to the existence of state transition probability and policy, calculating the return of a state is not easy. We need to calculate the return of all trajectories starting from this state and calculate their expectations. The expected return of a state s is defined as the state value function V

$$\begin{aligned} V(s) &= E_{s,a\sim\tau}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] \\ &= \sum_{(s_t,a_t,\cdots)\sim\tau} \pi(a_t|s_t)\, p(s_{t+1}|s_t, a_t) \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \\ &= E_\pi\left[R_{t+1} + \gamma V(s_{t+1})|s_t = s\right] \end{aligned} \quad (2)$$

where τ means an interaction trajectory, $\pi(a_t|s_t)$ is the policy, and $p(s_{t+1}|s_t, a_t)$ is the state transition probability. As shown in (2), the state value function can be expressed in a recursive form. Thus, the goal of finding trajectories that can maximize the cumulative rewards is turned to finding policies that can maximize $V(s_t)$. Similarly, the expected return of performing a under state s is defined as the state-action value function Q

$$Q(s, a) = E_\pi\left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1})|s_t = s, a_t = a\right]. \quad (3)$$

Equations (2) and (3) are called the Bellman equations [4], [46]. The Bellman equations calculate the weighted average of all trajectories' cumulative rewards based on their occurrence probabilities and are the basic equations to solve RL tasks.

The optimal value function refers to the one with the largest value in all states: $V^* = \max v_\pi(s)$. Similarly, the optimal action value function is: $Q^*(s, a) = \max Q(s, a)$. For all MDPs, there are always one or more optimal policies, and the value functions of all optimal policies are the same. Thus, the optimal policy can be found by maximizing the optimal Q function

$$\pi_*(a \mid s) = \begin{cases} 1, & \text{if } a = \underset{a\in\mathcal{A}}{\operatorname{argmax}} Q_*(s, a) \\ 0, & \text{otherwise .} \end{cases} \quad (4)$$

3) On-Policy and Off-Policy Methods: On-policy and off-policy refer to two different ways of training, whose main difference [3] is whether the behavior policy and the target policy are the same. The behavior policy is the policy used to interact with the environment to generate training data. The target policy is the policy that we want the agent to learn.

The target policy in one on-policy methods is directly used to generate data for the next round of policy optimization. The data are discarded once the next round of policy optimization finishes. Accordingly, the behavior policy is the same as the target policy. An off-policy method stores samples generated

by behavior policies into a buffer when interacting with the environment. During training, samples from the buffer are used to update the target policy. The training data may be from old policies, so the behavior policy is not the same as the target policy. The advantage of the on-policy methods is that they can directly optimize the policy. While the off-policy methods have higher data efficiency.

4) Dynamic Programming Methods: When the environment model is fully known, the Bellman equations can be solved by dynamic programming methods [46]. Dynamic programming is a method to solve complex problems by decomposing the original problem into relatively simple subproblems. The algorithms based on dynamic planning are mainly policy iteration and value iteration [3].

The idea of policy iteration is to iteratively execute policy evaluation and policy improvement steps until the policy converges to the optimal. In the policy evaluation step, the policy is used to calculate the current value. In the policy improvement step, the previous policy evaluation step's value is used to generate a better policy. The process of value iteration is: in each state, perform each action in turn and calculate the Q values. The optimal Q value is served as the value of the current state. The iteration ends when the optimal value of each state no longer changes.

The policy evaluation step in policy iteration requires the value function to converge. While in value iteration, the optimal value and the optimal policy converge simultaneously. Therefore, once the value function reaches the optimal value, the policy also converges to its optimal, thus simplifying the iteration steps.

5) Monte Carlo Methods: Mostly, some environment properties are difficult to obtain, meaning fully modeling the environment is usually impossible. In this case, the MC method can be used to evaluate the value. The steps of MC method to evaluate the value of a certain state are as follows. First, multiple simulations are carried out from this state to obtain multiple trajectories. Then calculate the cumulative reward of each trajectory. Finally, calculate the value of this state through $V(s_t) = V(s_t) + \alpha(G_t - V(s_t))$. Since the rewards for updating the estimated value are from real interaction trajectories, the estimated value is unbiased.

6) Temporal Difference Methods: TD methods [47], similar to MC, learn directly from experience and do not need to know the dynamics model of the environment. The difference is that TD methods simulate only one step under current state instead of reaching the terminal state. The simplest TD method, TD(0) [48] is

$$V(s_t) = V(s_t) + \alpha(R_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \tag{5}$$

where $R_{t+1} + \gamma V(s_{t+1})$ is estimated value at $t+1$ and is called TD target. $R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is called TD error. The above formula can be inferred that the TD method can learn from incomplete sequences, thereby avoiding episode updates and improving the convergence speed. The update of TD(0) is based on existing estimates to some extent, similar to DP, so it is also a bootstrap method [3].

Agents that use TD error to update the value function can be trained using on-policy methods, like SARSA [3], [49], or off-policy methods, like Q-learning [50]. Q-learning was a breakthrough of early RL algorithms [3]. The update equation of Q-learning is

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \tag{6}$$

where $\alpha \in (0, 1]$ means step size. When computing the current Q values, Q-learning does not follow the interaction sequence but selects the action with the biggest Q value at the next time step. This approach leads to an overestimation of Q values, and this problem will be discussed later in this article.

7) Policy Gradient Theorem: As discussed above, the policy: $\pi : S \rightarrow A$ is a mapping from states to actions, representing the probability of choosing action a in state s, and is usually defined as a probability distribution or probability density function of all actions

$$\pi_\theta(a|s) = P(a|s, \theta) \tag{7}$$

where θ is the parameter of the policy.

The above dynamic programming, MC and TD methods all need to calculate the optimal Q values before using (4) to obtain the optimal policy, which are called value-based methods. Another class of methods that can directly optimize the policy is called the policy gradient methods.

Defining the expected return as the performance measure of policy π_θ

$$J(\theta) = V_{\pi_\theta}(s) = E_{\pi_\theta(s)}\left[\sum_a Q(s, a)\pi_\theta(a|s)\right] \tag{8}$$

then the equations for policy optimization can be obtained by the policy gradient theorem by differentiating θ [51]

$$\nabla_\theta J(\theta) = E_{\pi_\theta(s)}\left[\sum_a Q(s, a)\nabla_\theta \pi_\theta(a|s)\right] \tag{9}$$

$$\theta = \theta + \alpha \nabla_\theta J \tag{10}$$

where α is the step size.

The Reinforce algorithm [52] is a conventional policy gradient RL algorithm. It uses estimated cumulative returns obtained from sampled trajectories by the MC method to update the policy since the expected value of the sample's gradients is an unbiased estimate of the actual gradient. Its most commonly used variant is the form with a baseline, whose purpose is to reduce the variance generated when estimating the gradient

$$\nabla_\theta J(\theta) = E_{\pi_\theta(s)}[\nabla_\theta log \pi_\theta(a|s)(Q(s, a) - b)] \tag{11}$$

where b is usually a learned state-value function independent of a.

When using value-based methods to solve tasks with continuous action spaces, the action spaces must be discretized first. However, discretization always suffers from the curse of dimensionality [53]. What's worse, if the step size of discretization is too large this can lead to an unacceptable lack of smoothness in control outcomes [54]. The optimal policy obtained by policy gradient methods is a probability distribution or probability density function on actions, which can be continuous or discrete, avoiding the above shortcomings.

8) Actor-Critic Methods: The actor-critic [3] method generally refers to the simultaneous learning of a policy and a value function, where the value function is used to evaluate the policy. The actor is responsible for generating policies, selecting actions, and interacting with the environment. The actor is updated by gradients calculated from (9) and (10). The critic evaluates the value function of the actor's policy at each time step. Different measures can be used to evaluate the actor's policy, such as action-value function $Q(s, a)$, state-value function $V(s)$, or advantage function $A(s, a)$.

B. Deep Learning

With the continuous development of deep neural network technologies [55], [56] such as convolutional neural network (CNN) [9] and recurrent neural network (RNN) [57] and the improvement of computing power, deep learning has become more and more influential. The achievements of deep learning in image recognition, text processing, and other domains have shown that it has a strong fitting capability as well as powerful representation capability when processing high-dimensional data. The high-precision image recognition network AlexNet proposed by Krizhevsky *et al.* [10] has set off an upsurge in deep learning research based on deep neural networks. In 2013, word2vector [58], [59] was proposed, which has a huge impact on the subsequent NLP technology based on deep learning, and is widely used in machine translation [60], word representation [61], and other domains [62]. (Generative Adversarial Network (GAN) [63] allows neural networks to generate the data we want and is also widely used as a component of complex deep neural networks. He *et al.* [11] proposed the first deep neural network that surpassed the human level in visual recognition problems: Resnet [11], which became one of the basic feature extraction network in the general computer vision field. In 2014, Deepmind used the Attention mechanism on the RNN model for image classification [64]. Bahdanau *et al.* [65] used a mechanism similar to Attention to perform translation and alignment on machine translation tasks at the same time. This is the first application of Attention on NLP. In 2017, the Google machine translation team used a lot of self-attention (Self-Attention) mechanisms to learn text representation [66].

The representation ability of deep learning mainly relies on multilayer neural networks with neurons [67] as the basic units. The perceptron [68] is the earliest neural network prototype known as a single-layer neural network (no hidden layers). It can only complete the simplest linear classification tasks and cannot solve the XOR problem [69]. Due to the increased number of neurons and layers, the multilayer perceptron has outstanding nonlinear approximating capabilities. Hornik *et al.* [13] proved that the multilayer perceptron can approximate any nonlinear function.

As a combination of deep learning and RL, DRL uses the powerful representation ability of neural networks to process high-dimensional input and approximate values or policies to solve RL problems with excessive state space and continuous action space. Taking Go as an example, each position on the board has three states, which creates a large state space, making it impossible for conventional RL to calculate each state's value. With the help of deep learning, a deep neural network representing the state of the chessboard can be trained. Then, based on the state representation, RL can be used to learn how to choose the placement position and obtain the largest cumulative reward. Generally, the above two processes can be regarded as: mapping original states to features and mapping features to actions, which are respectively completed by deep learning and RL. Because the deep neural network can be used as a black box, DRL can consider these two processes as a whole. DRL focuses more on RL [40] and still solves decision-making problems, so the following section mainly introduces how deep learning is combined with RL.

III. Value-Based DRL Methods

Value-based methods are an essential class of RL methods that focus on representing the value function and finding the optimal value function. The Q-learning algorithm mentioned above is the most classic value-based algorithm. This section introduces Deep Q Network, the seminal value-based DRL method, then discusses various variants and improvements.

A. Deep Q-Learning

Conventional Q-learning uses the TD method to update Q values and stores them in the Q table, which is not feasible for problems with large state spaces and action spaces. Mnih *et al.* proposed the Deep Q Network [70], which combines deep neural networks and Q-learning and exceeds the level of human players on most Atari 2600 games. Algorithm 1 shows the training details of the Deep Q Network. Compared with conventional Q-learning, improvement has the following three aspects:

1) Deep Q Network: Under the premise of using Q-learning as the RL algorithm, a simple deep neural network called Deep Q Network is used to extract low-level features from raw images of Atari games and approximate the action-value function without any other domain knowledge. The hidden layer of DQN includes three convolutional layers and a fully connected layer, as shown in Fig. 2. The result of the output layer is the Q value of each action. The approximated value of DQN at time step t is:

$$y = R + \gamma \max_a Q(s_{t+1}, a; \theta) \tag{12}$$

where θ stands for parameters of deep Q network and are updated through minimizing the MSE between the approximated and the real Q value.

To enhancing exploration ability, DQN use the $\epsilon-$ greedy method which execute stochastic actions with a certain probability ϵ, and execute optimal actions with the remaining probability $1-\epsilon$, which is equivalent to adding a certain amount of noise in the policy.

2) Experience Replay: The trajectories obtained from interaction have some correlation in the time domain. Directly using these trajectories for training may cause a growing gap between the estimated value and the expected value. Experience replay is proposed to eliminate the correlation between training samples by storing historical transitions into

Algorithm 1 Deep Q-learning Algorithm [70]

Initialize experience replay buffer D;
Initialize Q network with random weights θ and target network Q^- with weights $\theta^- = \theta$;
for *episode=1, 2...N* **do**
  Reset environment and initial state $\phi(s_0)$ using preprocessed video game images;
  for *t=0, 1, 2...T* **do**
    Use $\phi(s_t)$ as input of Q network and get Q value of each action;
    Choose action $a_t = \epsilon - greedy(s_t)$;
    Take action a_t under s_t, observe reward R and new state $\phi(s_{t+1})$;
    Push $(\phi(s_t), a_t, R, \phi(s_{t+1}))$ into D;
    Randomly take m samples $(\phi(s_j), a_j, R_j, \phi(s_{j+1}))$ from D;
    Compute approximate value:

$$y_j = \begin{cases} R_j & \phi(s_{j+1}) \text{ is terminal} \\ y_{nt} & \phi(s_{j+1}) \text{ is nonterminal} \end{cases}$$

    where

$$y_{nt} = R_j + \gamma \max_a Q^-\left(\phi(s_{j+1}), a; \theta^-\right)$$

    Update Q network by executing gradient decent on loss function:

$$L(\theta) = 1/m \sum_{j=1}^{m} (y_j - Q(\phi(s_j), a_j; \theta))^2$$

    Set $\theta^- = \theta$ every C steps;
  end
end

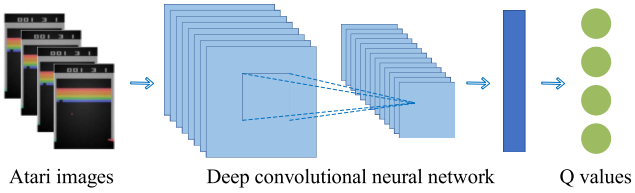

Fig. 2. Architecture of Deep Q Network in [70]. Deep Q-learning uses the deep CNN to approximate the value function and takes the game screenshot of atari 2600 as the direct input. The final output is the Q value for each executable action.

a buffer and choosing samples from it for training, randomly and uniformly. Moreover, experience replay allows DQN to use an off-policy manner for training, significantly improving data efficiency.

3) Target Network: The values of adjacent time steps used to update DQN are obtained by the same network with different parameters because of parameters updating, which may cause problems like unstable output. Mnih *et al.* fixed this problem by introducing a target network with the same structure as the main network.

At the beginning of training, both networks use the same parameters. During the whole training process, the main network is responsible for interacting with the environment and getting training samples. At each training step, the target value from the target network and the estimated Q value from the main network are used to update the main network. Every time the training completes a certain number of steps, the main network's parameters are synchronized to the target network. The target network keeps the target value unchanged for some time, thereby enhancing the stability of DQN. With the help of the target network, the approximated value is turned to

$$y = R + \gamma \max_a Q\left(s_{t+1}, a; \theta^-\right) \tag{13}$$

where Q^- stands for the target network with parameters θ^-.

B. Double DQN

As early as 1993, Thrun and Schwartz [71] discovered the problem of overestimated Q value in Q-learning algorithm, and regarded it as the impact of inadequate function approximation ability. Hasselt [72] attributed this to environmental noise. Deepmind [73] proved that any kind of error would lead to the overestimation of Q value, whether it was environmental noise, function approximation, nonstationarity, or any other reason. The solution to the problem is double Q-learning, and the algorithm is called the double deep Q network (DDQN) after combining with DQN.

The target network introduced in DQN represents a single value function with parameter θ^- that is responsible for selecting actions and calculating target values as shown in (13). DDQN aims to decompose these two parts into two different value functions, that is, two networks. The main network is used to select the optimal action, and the target network is used to estimate the value function. The target network in the DQN framework provides a natural candidate for the second value function, which eliminates the need to introduce an additional network. Let the parameter of the main network be θ, the formula for solving the target value is

$$y_j = R_{j+1} + \gamma\, Q\left(s_{j+1}, \text{argmax}_a Q\left(s_{j+1}, a; \theta\right), \theta^-\right). \tag{14}$$

The parameter update method of the two networks is similar to that in DQN.

C. Prioritized Experience Replay

When sampling transitions from experience replay for training, each transition is selected with the same probability, that is, each transition is learned with the same frequency. But in fact, different transitions have different effects on the backpropagation of DQN due to different TD errors. The greater the TD error, the greater the effect on backpropagation. Moreover, the size of the buffer is limited, and data helpful to learning may be discarded before being sampled. Prioritized experience replay is proposed by Schaul *et al.* [74] to solve these problems.

Prioritized experience replay uses the TD error of each transition as the criterion for evaluating priority. The form of

TD error used by Schaul *et al.* [74] is

$$\delta_t = R_t + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1}). \tag{15}$$

The larger the absolute value of δ_t, and the greater the probability that the corresponding transition will be selected, the greater the contribution to policy improvement. In the sampling process, Schaul *et al.* [74] adopted stochastic prioritization and importance-sampling methods. The stochastic prioritization operation can not only make full use of the transitions but also ensures diversity. Importance-sampling slows down the parameter update speed and ensures the stability of learning.

D. Dueling Architecture

Unlike most other works that focus on improving control and RL algorithms, Wang *et al.* [75] focused on innovating better neural network architectures and proposed the dueling architecture. Adopting the idea of advantage updating algorithm [76] where the advantage function is defined as $A(s, a) = Q(s, a) - V(s, a)$ meaning the relative advantage of taking each action in a certain state, the dueling architecture estimates both the value $V(s)$ and advantage $A(s, a)$, and $Q(s, a)$ is a combination of the above two streams. The introduction of the advantage function can achieve better policy evaluation in the case of multiple similar-valued actions.

The deep neural networks model used in DQN proposed by Minh *et al.* [70] in 2015 has three convolutional layers followed by two fully connected layers, and the output is the Q values of each action. Different from the DQN architecture, the dueling architecture divides the extracted features of the convolutional layers into two streams. One stream estimates the state value function $V(s)$, and the other estimates the advantage function $A(s)$. $Q(s, a)$ can be obtained by simply adding outputs of the two steams

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \tag{16}$$

where α and β are parameters of two fully connected layers from the A stream and V stream. Given $Q(s, a)$, an infinite number of possible combinations of V and A exit, of which a small number are reasonable so the A function must be restricted. Finally, Wang *et al.* adopted the following method to calculate $Q(s, a)$:

$$\begin{aligned} Q(s, a; \theta, \alpha, \beta) = & V(s; \theta, \beta) + A(s, a; \theta, \alpha) \\ & - 1/|A| \sum_{a'} A\big(s, a'; \theta, \alpha\big). \end{aligned} \tag{17}$$

E. Noisy Network

Enhancing exploration ability is a common problem both in RL and DRL. DQN uses the $\epsilon-$greedy method. This section introduces another way: noisy network [77]. The idea of noisy network is to add noise to the neural networks to affect the final value output, thus enhancing the exploration ability of the policy. The larger the noise, the greater the difference between the policy and the original policy, the stronger the exploration ability. Taking DQN as an example, add random noise ϵ and ϵ^- with parameters δ and δ^- to the target network and main network. The new objective function turns to

$$\begin{aligned} & L(\theta) \\ & = E_{(s_j, a_j, r_j, s_{j+1}) \sim D} \\ & \quad \times \big[R_{j+1} + \gamma\, Q(s_{j+1}, \operatorname{argmax}_a Q(s_{j+1}, a, \epsilon; \theta, \delta), \epsilon^-; \theta^-, \delta^-) \\ & \quad - Q(s_j, a_j, \epsilon; \theta, \delta)\big]^2. \end{aligned} \tag{18}$$

F. Multistep Learning

Q-learning uses the current immediate reward and the value estimate of the next time step as the target value. If the previous policy is poor and the network parameter deviation is large, the target value deviation obtained by this method is also large, resulting in slow learning speed. To speed up the convergence, (3) can be further expanded, not only using the reward of the next time step but also adding the rewards of more following time steps to the target value. The new equation is as follows:

$$\begin{aligned} Q(s_t, a_t) = & R_{t+1} + \gamma\, R_{t+2} + \cdots \\ & + \gamma^{n-1} R_{t+n} + \max_a \gamma^n Q(s_{t+n+1}, a). \end{aligned} \tag{19}$$

The target value can be more accurately estimated at the early stage of training through this way, thus accelerating the training speed. The above method is called multistep learning [78].

G. Distributional Approach

In conventional RL, the output of the value function is the expected return of each action. With the assumption that the distribution of value is more reliable than its expectation, the idea of distributed RL is to regard the value as a random variable, and its goal is to estimate the distribution of the value [79]. In [79], Z is defined as the value distribution whose expectation is Q. According to the conventional definition of the Bellman equation, the distributional Bellman equation that describes the relationship between the current value distribution and the future value distribution is defined as

$$Z(s_t, a_t) = R + \gamma\, Z(s_{t+1}, a_{t+1}). \tag{20}$$

Determining the distribution category is an important step to figure out the concrete form of Z. Parameter distribution, which indicates that a set of learnable parameters controls the distribution, is used to model Z. Finally, Z is defined as follows:

$$Z_\theta(s_t, a_t) = z_i \text{ w.p. } p_i(s_t, a_t) = \frac{e^{\theta_i(s_t, a_t)}}{\sum_j e^{\theta_j(s_t, a_t)}}. \tag{21}$$

Using the above discrete distribution, Bellemare *et al.* proposed categorical algorithm to ensure that the distribution of target value and the distribution of estimated value are the same. The Wasserstein distance [80] is a better way to measure the distance between two probability distributions. However, when using the Wasserstein distance as the loss function, the stochastic gradient descent technique cannot be used for optimization, so Bellemare *et al.* [79] used Kullback-Leibler (KL) divergence as a measure of distance. Dabney *et al.* [81] solved this problem using quantile regression [82] to minimize Wasserstein loss.

H. Other Improvements and Variants

The DRL community has also made other modifications to improve the performance of DQN on different tasks and expand its scope of application. Hausknecht and Stone [83] proposed Deep Recurrent Q-Network (DRQN), which uses Long short-term memory (LSTM) combined with DQN to solve the partially observable Markov decision process (POMDP) [45] problems. To make the agent have stronger exploration ability, inspired by Thompson sampling [84], Osband *et al.* [85] proposed the bootstrap DQN, which can significantly reduce the learning time and improve the performance on most Atari games. Du *et al.* [86] found a way to effectively explore the state space using difference maximization Q-learning (DMQ) and linear function approximation and realized an algorithm that can learn a near-optimal policy. Kapturowski *et al.* [87] studied the effects of parameter lag resulting in representational drift and recurrent state staleness and empirically derived an improved training policy.

Each of these algorithms can improve a certain aspect of performance on different tasks, but it is unclear which of these extensions or variants are complementary and can be combined effectively. Hessel *et al.* [88] studied double DQN, prioritized experience replay, dueling network, noisy network, multistep learning, and distributional DQN, proved that these six components are complementary through experiments on 57 Atari games, and proposed a combined algorithm called rainbow. This combination technique had been adopted by Wang *et al.* [75] earlier, where they proved through experiments that the dueling architecture can be combined with double DQN with prioritized experience replay and improve its performance. Since each component improves performance to a certain extent, Rainbow achieved the optimal performance when proposed.

IV. Policy-Based DRL Methods

Methods introduced in this section are extensions of the policy gradient methods and actor-critic method introduced earlier, which are more efficient in high-dimensional state space and continuous action space.

A. Advantage Actor-Critic Methods

Conventional policy gradient algorithms are usually updated in an on-policy manner, resulting in slow policy convergence and low data efficiency. To speed up the training of the actor-critic Methods, Mnih *et al.* [89] proposed asynchronous advantage actor-critic (A3C) to collect data asynchronously, where the agent interacts with the environment simultaneously in N threads. The interaction trajectories obtained from each thread will not be the same as long as their environment settings are different, thus speeding up the sample collection speed. After the samples are collected, each thread completes the training independently and updates the global model parameters asynchronously. The structure of A3C is shown in Fig. 3. Moreover, Mnih *et al.* used the advantage function to serve as the measure of the policy to better balance the bias and variance. N-step return technique [90], [91] is also adopted to update the value model more efficiently. To enhance the exploration ability of the model, the entropy of the policy

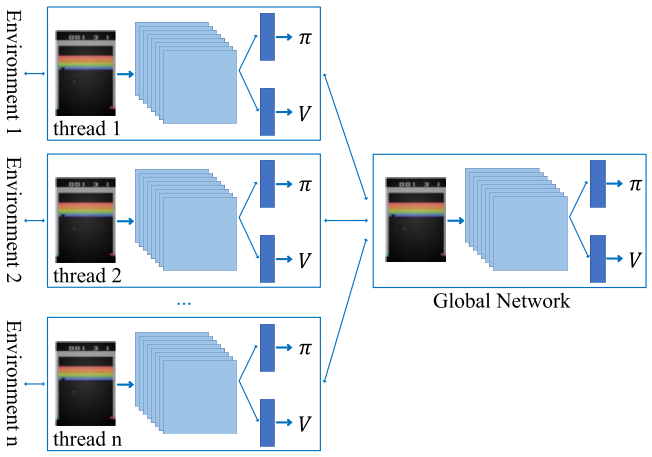

Fig. 3. A3C architecture [89]: the global network uses the actor-critic framework and has n worker threads with the same network structure as the global network. Each thread independently interacts with its own environment and is trained to update the global network parameters. Every few time steps, the parameters of these threads are synchronized to that of the global network.

is added to the objective function. As a result, the complete policy gradient equation becomes

$$\nabla_\theta J(\theta)=\frac{1}{T}\sum_{t}^{T}\nabla_\theta \log \pi(\theta)\left(\sum_{i=1}^{n}\gamma^{i-1}\boldsymbol{r}_{t+1}+v(\boldsymbol{s}_{t+n})-v(\boldsymbol{s}_t)\right)+\beta\nabla_\theta \mathrm{H}(\pi(\boldsymbol{s}_t;\theta)) \tag{22}$$

where $\sum_{i=1}^{n}\gamma^{i-1}\boldsymbol{r}_{t+1}+v(\boldsymbol{s}_{t+n})-v(\boldsymbol{s}_t)$ is the advantage item with n-step return, and β is the weighting factor.

A2C [36] is a synchronous version of A3C proposed by OpenAI, which can achieve the same or better performance as A3C while making more efficient GPU use. Actor-critic with experience replay (ACER) [92] is an off-policy actor-critic algorithm that can be applied to both continuous and discrete action spaces and can be regarded as A3C's off-policy counterpart. When the off-policy method is used for training policy gradient methods, the difference between the target policy and the behavior policy may lead to instability. ACER proposes three methods to solve this problem. The first is to use the Q value estimation method in Retrace [93], which is an off-policy return-based RL algorithm that has three good properties: 1) it has low variance; 2) data collected from behavioral policies can be used safely; and 3) it has high data efficiency. Second, ACER adopts importance sampling to adjust the weight of the policy gradient. As a result, the policy gradient is

$$\hat{g}^{\mathrm{imp}}=\left(\prod_{t=0}^{k}\rho_t\right)\sum_{t=0}^{k}\left(\sum_{i=0}^{k-t}\gamma^{i}r_{t+i}\right)\nabla_\theta \log \pi_\theta(a_t \mid x_t) \tag{23}$$

where ρ_t denotes the importance weight. However, the multiplication operation causes the weight to be too large or too small when calculating the importance weight. In response to this problem, Wang *et al.* proposed a bias correction trick to truncate the importance weight. Third, hoping that the change of update step should not be too large in the policy space, ACER adopts the idea of trust region policy optimization (TRPO) [94] to limit the range of KL divergence.

B. Trust Region-Based Algorithms

In the equations of policy optimization (9), (10), α controls the step size of the update. Setting a good step size can make the policy converge faster, but an inappropriate one will cause the policy to be unstable and even deteriorate. Empirically, an appropriate step size should make the new policy no worse than the old policy, or in other words, the policy should increase monotonically. Schulman *et al.* [94] proposed TRPO algorithm and proved that the expected advantage of one policy over another is

$$\eta(\tilde{\pi}) - \eta(\pi) = E_{s_0,a_0,s_1,a_1,\ldots}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right] \quad (24)$$

where η stands for the estimated value of a policy. $\tilde{\pi}$ is the new policy. π is the old policy. a_t is sampled from $\tilde{\pi}(a_t|s_t)$, and s_0 is the initial state. So, as long as $E_{s_0,a_0,s_1,a_1,\ldots,}[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)]$ remains positive, the monotonic increase of the policy can be guaranteed. Schulman *et al.* introduced a trust-region constraint to make sure the old and new policy would not diverge too much after each round of updates, thus improving the training stability and guaranteeing a monotonic increase. The constraint is defined by the KL divergence [95] between the predicted distributions of the old and new policy on the same batch of data.

Schulman *et al.* proposed an improved TRPO algorithm, proximal policy optimization (PPO) [96], which simplifies TRPO using a truncated objective function. The objective function in PPO is

$$L_t = \frac{1}{N}\sum_{t=1}^{N}\Big[\min(r(\theta)A_{\pi_{\text{old}}}(s_t, a_t), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)A_{\pi_{\text{old}}}(s_t, a_t))\Big] \quad (25)$$

where r is the policy ratio of the new and old policy, and A is the advantage function. To avoid the unstable policy update caused by either the update step size or the policy ratio being too large, PPO adds two insurances to the objective function: the first is a restriction on $r_t(\theta)$. This ratio is limited to $[1-\epsilon, 1+\epsilon]$, ensuring that every update does not have too much fluctuation. The second insurance is the min function, which chooses the lower value of the two results.

Ye *et al.* [97] observed that in large-scale off-policy environments, the standard PPO using an on-policy training method would fail, because when $\pi_\theta(a_t|s_t) \gg \pi_{\theta_{\text{old}}}(a_t|s_t)$ and $A_t < 0$, $r_t(\theta)$ will escape the boundary, making the policy difficult to converge. Dual-clipped PPO is proposed to alleviate this issue by further clipping $r_t(\theta)$ with a lower bound. The objective turns to

$$\hat{E}_t\big[\max(\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t), cA_t)\big] \quad (26)$$

where $c > 1$ is a constant.

There are also other ways to express the trust-region constraints. Actor Critic using Kronecker-Factored Trust Region (ACKTR) [98] adopts the trust-region formulation of Kronecker-factored Approximate Curvature (K-FAC) [99], [100] to limit the update step size of the policy. K-FAC can approximate the Fisher information matrix in a short time, and the time required for each update is similar to stochastic gradient descent method used in TRPO and PPO, so it can update the model using natural gradients [101]. As a result, ACKTR achieves a faster training speed than TRPO and PPO. Nachum *et al.* [102] proposed trust-PCL, which introduced discounted relative-entropy trust-region to ensure the stability of optimization and used an off-policy method to improve sample efficiency.

C. Deterministic Policy Gradient

Previously discussed policies modeled as probability distributions or probability density functions are all stochastic policies since all actions have probabilities of being sampled, given any state. However, in many tasks, the policy is deterministic. In the deterministic policy context, when the agent encounters the same state, the selected action is the same and is expressed as $a = \mu(s)$. Silver *et al.* [103] derived the deterministic policy gradient (DPG) theorem as follows:

$$\nabla_\theta J(\theta) = E\big[\nabla_a Q_\mu(s, a)\nabla_\theta \mu(s)|_{a=\mu_\theta(s)}\big] \quad (27)$$

and proposed on-policy and off-policy deterministic actor-critic algorithms based on the fact that the theorem can be integrated into the general policy gradient framework.

The deep deterministic policy gradient (DDPG) algorithm [53] learns deterministic policies and expands them into continuous action space through the actor-critic architecture. What's more, the authors applied two techniques of DQN, experience replay, and the target network to DDPG.

1) *Experience Replay:* In the deterministic policy problem, the value function is only related to the state, so the off-policy method can be used. DDPG adopts a replay buffer to collect samples and randomly selects a part of them to optimize the value function, thereby improving data utilization.
2) *Target Network:* Like DQN, to avoid overestimating Q values, DDPG also uses the target network technique. However, since the value function in DDPG is more complicated than that in DQN, a different method from DQN is required to synchronize the target network. The method is soft replace, which makes the training more stable by letting the target network "slowly approach" in the direction of the main network in each training iteration. The synchronize method turns to

$$\begin{aligned} w' &\leftarrow \tau w + (1-\tau)w' \\ \theta' &\leftarrow \tau\theta + (1-\tau)\theta' \end{aligned} \quad (28)$$

where θ and θ^- are parameters of the main actor network and the target actor network, ω and ω^- are parameters of the main critic network and the target critic network, and $\tau \ll 1$ is the update coefficient.

In addition to these two skills, the authors also adopted Ornstein–Uhlenbeck [104] noise to enhance the exploration ability of DDPG. Fujimoto *et al.* [105] applied many techniques to DDPG to prevent overestimating the value function and proposed twin delayed deep deterministic (TD3). These techniques are as follows.

1) *CLIPPED Double Q-Learning:* TD3 uses two separate critics and selects the smaller estimated value to update the target value.
2) *Delayed Update of Target and Policy Networks:* Td3 updates the policy after every d times update of the critic instead of a simultaneous update to reduce the error accumulated in multiple updates.
3) *Target Policy Smoothing:* TD3 smooths and regularizes the estimated value of a small area around the target action, reducing the overfit introduced by value function estimation.

V. Maximum Entropy DRL

A. Maximum Entropy Reinforcement Learning Framework

The objective of conventional RL can be expressed as maximizing the sum of expected rewards $\sum_{t=0}^{T} E_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t)]$, where ρ_π is the distribution induced by the policy [106]. The idea of maximum entropy RL is augmenting the reward with an entropy term, such that the optimal policy aims to maximize its entropy and reward in each state simultaneously

$$J(\pi)=\sum_{t=0}^{T} E_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t)+\alpha H(\pi(.|s_t))] \tag{29}$$

where α is a temperature parameter that can control the optimization objective to make it pay more attention to the reward or entropy.

Maximum entropy RL has many advantages: 1) the agent can learn the optimal stochastic policy because of the entropy term; 2) stronger exploration capabilities. The entropy term encourages policies to explore more extensively while abandoning hopeless approaches; and 3) the policy can seek out the best mode in a multimodal reward landscape [106].

The idea of combining RL and entropy has been discussed in many early works. Boltzmann exploration [107] and policy gradient and Q-learning (PGQ) [108] learn to maximize entropy at each time step greedily. Ziebart *et al.* [109] and Boularias *et al.* [110] combined maximum entropy and inverse reinforcement learning [111] to calculate probability distributions of expert trajectories. Nachum *et al.* [112] studied the connection between value and policy-based RL in the context of entropy regularization and proposed path consistency learning (PCL) that outperforms A3C and DQN baselines. The optimization objective in (29) is in the same form as A3C's loss function, while the entropy term in A3C is only used as a regularizer.

B. Soft Q-Learning and SAC

The policy given by the conventional RL is a distribution centered on the optimal Q value. The action sampled from this policy is always near the optimal Q value, thus neglecting the suboptimal Q values, making the agent unable to learn multiple modes of accomplishing the task. Haarnoja *et al.* [106] proposed an energy-based model that uses exponentiated Q value to define the policy

$$\pi(\mathbf{a}|\mathbf{s})\propto \exp Q(\mathbf{s},\mathbf{a}). \tag{30}$$

In this way, the policy can assign a specific probability to each action and becomes a stochastic policy. In this context, the key to finding a good policy lies in finding a good Q function.

Let the soft Q function and V function be defined by

$$Q^*_{\text{soft}}=r_t+E_{(s_t+1,\ldots)\sim\rho_\pi}\left[\sum_{l=1}^{\infty}\gamma^l(r_{t+l}+\alpha H(\pi^*_{\text{Maxent}}(\cdot\mid s_{t+l})))\right] \tag{31}$$

$$V^*_{\text{soft}}=\alpha*\log\int_A \exp\left(\frac{1}{\alpha}Q^*_{\text{soft}}(s_t,a')\right)da'. \tag{32}$$

These two definitions are proved to satisfy Bellman equations' form by Haarnoja *et al.* and are called soft Bellman backups. Consequently, the form of the optimal policy can be derived as

$$\pi^*_{\text{MaxEnt}}(a_t\mid s_t)=\exp\left(\frac{1}{\alpha}Q^*_{\text{soft}}(s_t,a_t)-V^*_{\text{soft}}(s_t)\right). \tag{33}$$

For a better approximation, the soft Q function is modeled by a neural network parameterized with θ. Importance sampling is used to transform soft Q iteration into a random optimization problem, and stochastic gradient descent is used to update the Q network. A state-conditioned stochastic neural network is used to approximate the energy-based policy to get the unbiased action sample from the energy-based model.

Schulman *et al.* [113] proved that the soft Q-learning and policy gradient methods are equivalent because the loss gradient of soft Q-learning can be expressed as a policy gradient item with a baseline-error-gradient item. Wei *et al.* [114] adopted soft Q-learning in multiagent RL and achieved better performance in cooperative tasks compared with the state-of-the-art method, Multi-agent Deep Deterministic Policy Gradient (MADDPG) [115](Section VI-C).

Haarnoja *et al.* [116] proved that the soft policy iteration could converge to the optimal policy in the tabular case. To enable the application in large continuous domains, they used neural networks with parameters θ and ϕ to represent soft Q function and policy: $Q_\theta(s_t,a_t)$ and $\pi_\phi(a_t|s_t)$, and proposed soft actor-critic (SAC), the first RL algorithm that combines off-policy, actor-critic, and maximum entropy methods.

VI. Further Research Topics

Most of the DRL methods introduced in the previous sections are only suitable for games and simulation environments where agents can obtain rich reward signals and perform unlimited rollouts. However, more complex environments or real-world environments often face problems of limited samples, sparse rewards, and multiple agents. To handle the above problems, RL researchers have pioneered further research directions such as model-based RL, hierarchical RL, and meta-RL. With the help of deep learning, these directions have been further developed. Some of them are briefly introduced below.

A. Model-Based Methods

All algorithms introduced above are model-free, which directly learn the optimal value functions or policies from data obtained by the agent interacting with the environment. Although model-free methods have achieved excellent

performance on many tasks, applying to practical problems is still tricky because of the need for a large amount of training samples. Moreover, the error introduced by transitions in experience replay is a burden hard to eliminate for off-policy model-free methods [117], [118], [119]. Another type of RL algorithm is the model-based method whose purpose is to learn a dynamic model that can describe how the environment changes [120]. Once the learned model approximates the environment, the optimal policy can be found directly through some planning methods. What's more, the learned model can be directly used for on-policy training [121].

The high sampling complexity limits the range of use of model-free algorithms, especially when facing high-dimensional function approximators. Gu *et al.* [122] studied the sampling complexity in continuous control tasks and proposed an algorithm that mainly contains two complementary technologies. First, by simplifying the actor-critic architecture, the normalized advantage functions (NAFs) are proposed, in which Q-learning can adapt to the continuous action space. Then continuous Q-learning is combined with the model learned from off-policy experience to accelerate the training process of the algorithm in the continuous action space.

Based on the analysis that compounding errors make long-horizon rollouts unreliable, Janner *et al.* [123] proposed a model-based policy optimization (MBPO) algorithm that learns a model to generate short rollouts from previously encountered real data. Hafner *et al.* [124] proposed the Dreamer algorithm, which first learns a world model from experience, and then uses an actor-critic algorithm to interact with the learned world model and learn the optimal policy. The training is carried out in the learned model, so the multistep cumulative reward can be obtained and long-horizon planning can be carried out.

B. Hierarchical Deep Reinforcement Learning

HRL is proposed to solve tasks whose rewards are sparse and require targeted exploration. The conventional HRL ideas are to decompose complex goals into easy-to-solve subgoals and finally complete the original task by solving these subgoals [125] or abstract different levels of control layers [126]. However, due to the lack of representation ability, the selection of subgoals relies on expert knowledge. With the help of deep learning, HRL can solve problems with large state spaces and action spaces, and can directly process high-dimensional input. Moreover, the representation ability of deep learning provides more choices for the form of subgoals [125]. Hierarchical DRL (HDRL) has become an important research direction of DRL.

As Fig. 4 shows, Kulkarni *et al.* [127] introduced a hierarchical model called HIRO, composed of a higher-level controller to provide coarse-grained subgoals and a lower level controller to complete these subgoals. The option framework, first put forward by Sutton *et al.* [126], is a class of conventional HRL. Bacon *et al.* [128] derived the option policy weight theorem and proposed an option-critic architecture that can learn the option's internal policy and termination conditions. Andrychowicz *et al.* [129] added goals to the transition tuples in experience replay. Using the similarity between goals, transition tuples with different goals can be used to train other

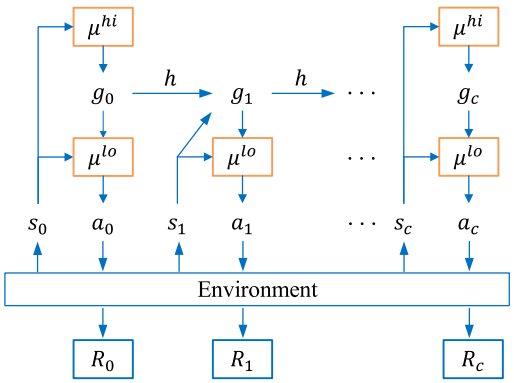

Fig. 4. HIRO's interactive process: the environment first produces an initial state s_0, which is then passed to μ^{hi} and μ^{lo}. The high-level controller generates a goal g_0 based on μ^{hi}. Sequentially, the lower level controller takes s_0 and g_0 as input and selects an action a_0 based on μ^{lo}. A reward R_0 is given by the environment and the state transits to s_1. The current goal g_1 is generated from the goal at the previous time step and the current state, via function h. The above operations are performed c times before μ^{hi} is used to generate a new goal.

goals. Eysenbach *et al.* [130] proposed the Diversity Is All You Need (DIAYN) algorithm, which allows the agent to learn a variety of task-solving skills even when there is no way to get a real reward, thereby helping the agent to explore the environment. What's more, the combination of these skills can be used to solve more complex problems with sparse rewards.

C. Multiagent Deep Reinforcement Learning

A multiagent system can generally be described as a Markov game [131] where all agents simultaneously select and perform their actions based on the current state. The behavior of each agent would affect the transition of the environment. In some complex tasks, each agent may only observe part of the environment [45]. Before the advent of deep learning, researchers have used RL to solve fully cooperative, fully competitive, and mixed tasks [132].

Multiagent DRL (MADRL) applies the idea of DRL to the learning and control of multiagent systems, thus overcoming some of the difficulties encountered by traditional methods, such as high-dimensional input, continuous action space, etc. Lowe *et al.* [115] proposed a multiagent version of DDPG, MADDPG, which allows each agent to obtain the action information of all other agents in the critic section, thereby achieving centralized training and decentralized execution.

However, as MADRL is increasingly used in complex state space and action space tasks, many problems have also emerged. To encourage those agents that are more helpful to the entire multiagent task, Foerster *et al.* [133] put forward the idea of "Counterfactual Baseline," which is used to solve the credit assignment tasks. Value decomposition networks (VDNs) [134] integrates each agent's value function to obtain a joint Q function, solving the spurious rewards problem and the lazy agent problem caused by the partial observability of the environment. QMIX [135] is an extension of VDN. It uses a mixed network to merge a single agent's local value functions and adds global state information during the training process to improve algorithm performance.

D. Learning From Demonstrations

In some tasks with sparse rewards or multiobjectives, learning algorithms based on cumulative rewards often fail to make agents act intelligently because of the massive exploration space and the difficulty of setting rewards. To ensure the agent learns an optimal or suboptimal policy faster, learning the reward function or policy from the expert's demonstration are usually good ideas, which are adopted by inverse RL [111] and imitation learning [136] respectively.

Early methods based on inverse RL mainly include apprenticeship learning [137], structured classification [138], maximum margin planning [139], and maximum entropy methods [109], [110]. In recent years, some inverse RL methods using deep learning techniques have been proposed. Finn *et al.* [140] studied the connection between GANs and inverse RL. Fu *et al.* [141] proposed adversarial inverse RL (AIRL) and demonstrates that AIRL can recover reward functions that are robust to the changes of environment dynamics.

The main idea of imitation learning is to use supervised learning methods to learn policies from expert demonstrations [142]. Conventional imitation learning methods focus on eliminating compound errors between behavioral policies and learned policies [143] and data augmentation methods [144]. Imitation learning can easily be combined with deep learning due to its supervised training manner [145]. Ho and Ermon [146] proposed a new general framework called Generative Adversarial Imitation Learning (GAIL) for directly extracting a policy from data by combining GANs and imitation learning and outperformed other imitation learning algorithms in large, high-dimensional environments. Song *et al.* [147] proposed the AdaBoost Maximum Eentropy Deep Inverse Reinforcement Learning (AME-DIRL) algorithm, which can learn nonlinear rewards when expert demonstration data is limited and imbalanced.

E. Meta-Reinforcement Learning

The purpose of meta learning [148] is to generalize knowledge to new tasks by learning experiences from multiple other tasks. The main idea of meta-reinforcement learning [149] is to apply the prior knowledge learned from a large number of RL tasks to new RL tasks, and to improve the learning speed and generalization ability of the agent. Meta-reinforcement learning mainly solves and optimizes the shortcomings below of DRL: low sample utilization [150], difficulty in designing reward functions [151], exploration strategies in unknown tasks [152], and lacking of generalization ability [153].

F. Offline Reinforcement Learning

The training process of RL requires the agent to interact with the environment. After a lot of trial-and-errors, the optimal policy can finally be found. When there is no simulation and the cost of interacting with the environment is high, how to use the previously collected data for training is a key issue. Offline RL [154] is a method that has been continuously developed in recent years. It refers to the use of offline data collected in the past to train the agent without interaction during the training process. A key issue of offline RL is the distributional shift [154] between behavioral policy π_β of offline datasets and agent policy π in offline datasets.

At present, offline RL methods are mainly divided into three categories [154]. The first class of methods uses importance sampling to directly evaluate the policy return $J(\pi)$ or estimate the corresponding policy gradient through the offline data sampled from π_β [155]. The second method is to use RL algorithms based on dynamic programming, such as those based on Q-learning. Such algorithms cannot be directly applied in the absence of online data, so policy constrained methods [117] and uncertainty-based methods [156] have been proposed. In the policy constraint method, the learned policy is restricted to the vicinity of the behavior policy to eliminate the distribution shift. Uncertainty-based methods try to use the epistemic uncertainty of Q-values to detect distribution shift. The last class is the model-based method introduced earlier. Model-based methods learn the state transition probability of one environment and can use supervised methods [157], [158], so they can make better use of offline datasets.

G. Transfer Learning in Reinforcement Learning

In most RL algorithms, small state changes will cause previously trained strategies to fail, and training from scratch is expensive. Transfer learning (TL) [159] can improve the training efficiency of the target task by applying the experience learned from the source task. Therefore, to improve the learning speed in new tasks, TL for RL [160] has become an attractive research direction. According to what knowledge is transferred, TL can help the DRL algorithm in the following aspects [161]: reward shaping, learning from demonstrations, policy transfer, intertask mapping, and representation transfer. Lan *et al.* [162] proposed a novel transfer reinforcement learning approach via meta-knowledge extraction using autopruned decision trees. Zhang *et al.* [163] proposed the DQDR method, which extracts domain knowledge from humans and expresses it as a set of rules and couples it with a DQN to improve the transferability of RL algorithms.

VII. Conclusion

DRL can directly make decisions based on the input data by incorporating the advantages of deep learning and reinforcement learning. This article introduces the algorithms implemented in the current popular DRL libraries, including value-based, strategy-based, and maximum entropy-based algorithms, and briefly lists the research subfields of DRL in recent years. The research on DRL is a crucial step toward building an autonomous system with a higher understanding of the world.

References

[1] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.
[2] M. Minsky, "Steps toward artificial intelligence," *Proc. IRE*, vol. 49, no. 1, pp. 8–30, Jan. 1961.
[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[4] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, Jul. 1966.
[5] D. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1. Belmont, CA, USA: Athena Sci., 2000.
[6] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Develop.*, vol. 3, no. 3, pp. 210–229, 1959.

[7] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Mach. Learn.*, vol. 49, no. 2, pp. 291–323, 2002.

[8] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, 2012, pp. 1097–1105.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[12] L. Deng and D. Yu, "Deep learning: Methods and applications," *Found. Trends Signal Process.*, vol. 7 nos. 3–4, pp. 197–387, 2013.

[13] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Dec. 1989.

[14] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Syst.*, vol. 2, no. 4, pp. 303–314, Dec. 1989.

[15] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Netw.*, vol. 3, no. 5, pp. 551–560, 1990.

[16] L. Tai, J. Zhang, M. Liu, and W. Burgard, "Socially compliant navigation through raw depth inputs with generative adversarial imitation learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 1111–1117.

[17] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 31–36.

[18] O. Zhelo, J. Zhang, L. Tai, M. Liu, and W. Burgard, "Curiosity-driven exploration for mapless navigation with deep reinforcement learning," 2018, *arXiv:1804.00456*.

[19] J. Hwangbo *et al.*, "Learning agile and dynamic motor skills for legged robots," *Sci. Robot.*, vol. 4, no. 26, Jan. 2019, Art. no. eaau5872.

[20] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[21] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[22] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018.

[23] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[24] C. Berner *et al.*, "Dota 2 with large scale deep reinforcement learning," 2019, *arXiv:1912.06680*.

[25] D. A. Hudson and C. D. Manning, "Compositional attention networks for machine reasoning," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–20.

[26] X. Wang, W. Chen, J. Wu, Y.-F. Wang, and W. Y. Wang, "Video captioning via hierarchical reinforcement learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4213–4222.

[27] L. Wu, F. Tian, T. Qin, J. Lai, and T.-Y. Liu, "A study of reinforcement learning for neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 3612–3621.

[28] V. Talpaert *et al.*, "Exploring applications of deep reinforcement learning for real-world autonomous driving systems," in *Proc. 14th Int. Joint Conf. Comput. Vis., Imag. Comput. Graph. Theory Appl.* Setúbal, Portugal: SCITEPRESS, 2019, pp. 564–572.

[29] S. Milz, G. Arbeiter, C. Witt, B. Abdallah, and S. Yogamani, "Visual SLAM for automated driving: Exploring the applications of deep learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2018, pp. 247–257.

[30] J. Li, L. Yao, X. Xu, B. Cheng, and J. Ren, "Deep reinforcement learning for pedestrian collision avoidance and human-machine cooperative driving," *Inf. Sci.*, vol. 532, pp. 110–124, Sep. 2020.

[31] G. Zheng *et al.*, "DRN: A deep reinforcement learning framework for news recommendation," in *Proc. World Wide Web Conf.*, 2018, pp. 167–176.

[32] M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, and E. H. Chi, "Top-K off-policy correction for a REINFORCE recommender system," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, Jan. 2019, pp. 456–464.

[33] S. Yun, J. Choi, Y. Yoo, K. Yun, and J. Y. Choi, "Action-decision networks for visual tracking with deep reinforcement learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2711–2720.

[34] S. A. Eslami *et al.*, "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.

[35] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum, "Learning to see physics via visual de-animation," in *Proc. NIPS*, 2017, pp. 153–164.

[36] P. Dhariwal *et al.* (2017). *OpenAI Baselines*. [Online]. Available: https://github.com/openai/baselines

[37] I. Caspi, G. Leibovich, S. Endrawis, and G. Novik, "Reinforcement learning coach," Version 0.10.0, Zenodo, Dec. 2017, doi: 10.5281/zenodo.1134899.

[38] S. Guadarrama *et al.* (2018). *TF-Agents: A Library for Reinforcement Learning in Tensorflow*. Accessed: Jun. 25, 2019. [Online]. Available: https://github.com/tensorflow/agents

[39] J. Weng *et al.* (2020). *Tianshou*. [Online]. Available: https://github.com/thu-ml/tianshou

[40] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[41] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, nos. 3–4, pp. 219–354, Dec. 2018.

[42] Y. Li, "Deep reinforcement learning: An overview," 2017, *arXiv:1701.07274*.

[43] H.-N. Wang *et al.*, "Deep reinforcement learning: A survey," *Frontiers Inf. Technol. Electron. Eng.*, vol. 21, pp. 1726–1744, Oct. 2020.

[44] R. Bellman, "A Markovian decision process," *Indiana Univ. Math. J.*, vol. 6, no. 4, pp. 679–684, Apr. 1957.

[45] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, nos. 1–2, pp. 99–134, 1998.

[46] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA, USA: MIT Press, 1960.

[47] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, 1988.

[48] C. Szepesvári, "Algorithms for reinforcement learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 4, no. 1, pp. 1–103, 2010.

[49] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Citeseer, 1994, vol. 37.

[50] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[51] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. NIPs*, vol. 99, 1999, pp. 1057–1063.

[52] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[53] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. ICLR*, 2016, pp. 1–14.

[54] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "End-to-end deep reinforcement learning for lane keeping assist," 2016, *arXiv:1612.04340*.

[55] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[56] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, Dec. 2015.

[57] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[58] T. Mikolov, L. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Adv. Neural Inf. Process. Syst.*, vol. 2, Dec. 2013, pp. 3111–3119.

[59] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. 1st Int. Conf. Learn. Represent.*, in Workshop Track Proceedings, Scottsdale, AZ, USA, May 2013, pp. 1–12.

[60] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1–5.

[61] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.

[62] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," 2015, *arXiv:1510.03820*.

[63] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 2, 2014, pp. 2672–2680.

[64] V. Mnih *et al.*, "Recurrent models of visual attention," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2204–2212.

[65] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.

[66] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[67] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 2, Dec. 2014, pp. 3320–3328.

[68] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.

[69] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY, USA: Springer, 2009.

[70] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[71] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proc. 4th Connectionist Models Summer School*, Hillsdale, NJ, USA, 1993, pp. 255–263.

[72] H. V. Hasselt, "Double Q-learning," in *Proc. Neural Inf. Process. Syst. (NIPS)*, vol. 23, Dec. 2010, pp. 2613–2621.

[73] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, vol. 30, no. 1, pp. 1–7.

[74] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. ICLR*, 2016, pp. 1–21.

[75] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.

[76] L. C. Baird, "Advantage updating," Wright Lab, Tech. Rep. WL-TR-93-1146, 1993.

[77] M. Fortunato *et al.*, "Noisy networks for exploration," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–21.

[78] T. Hester *et al.*, "Deep Q-learning from demonstrations," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–8.

[79] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 449–458.

[80] P. J. Bickel and D. A. Freedman, "Some asymptotic theory for the bootstrap," *Ann. Statist.*, vol. 9, no. 6, pp. 1196–1217, Nov. 1981.

[81] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–10.

[82] R. Koenker and K. Hallock, "Quantile regression," *J. Econ. Perspect.*, vol. 15, no. 4, pp. 143–156, 2001.

[83] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Ser.*, 2015, pp. 1–9.

[84] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, pp. 285–294, Dec. 1933.

[85] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, "Deep exploration via bootstrapped DQN," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4033–4041.

[86] S. S. Du, Y. Luo, R. Wang, and H. Zhang, "Provably efficient *Q*-learning with function approximation via distribution shift error checking Oracle," in *Proc. Adv. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2019, pp. 1–11.

[87] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–19.

[88] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–14.

[89] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[90] C. J. C. H. Watkins, "Learning from delayed rewards," King's College, Cambridge, U.K., 1989.

[91] J. Peng and R. J. Williams, "Incremental multi-step Q-learning," in *Machine Learning Proceedings 1994*. Amsterdam, The Netherlands: Elsevier, 1994, pp. 226–232.

[92] Z. Wang *et al.*, "Sample efficient actor-critic with experience replay," 2016, *arXiv:1611.01224*.

[93] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1054–1062.

[94] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[95] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[96] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[97] D. Ye *et al.*, "Mastering complex control in MOBA games with deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 4, pp. 6672–6679.

[98] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5285–5294.

[99] R. Grosse and J. Martens, "A Kronecker-factored approximate Fisher matrix for convolution layers," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 573–582.

[100] J. Martens and R. Grosse, "Optimizing neural networks with Kronecker-factored approximate curvature," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2408–2417.

[101] S. M. Kakade, "A natural policy gradient," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 14, 2001, pp. 1–8.

[102] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Trust-PCL: An off-policy trust region method for continuous control," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–14.

[103] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 387–395.

[104] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Phys. Rev.*, vol. 36, p. 823, Sep. 1930.

[105] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.

[106] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1352–1361.

[107] B. Sallans and G. E. Hinton, "Reinforcement learning with factored states and actions," *J. Mach. Learn. Res.*, vol. 5, no. 8, pp. 1063–1088, 2004.

[108] B. O'Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih, "Combining policy gradient and Q-learning," 2016, *arXiv:1611.01626*.

[109] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[110] A. Boularias, J. Kober, and J. Peters, "Relative entropy inverse reinforcement learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, in JMLR Workshop and Conference Proceedings, 2011, pp. 182–189.

[111] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proc. ICML*, vol. 1, 2000, pp. 1–2.

[112] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2772–2782.

[113] J. Schulman, X. Chen, and P. Abbeel, "Equivalence between policy gradients and soft Q-learning," 2017, *arXiv:1704.06440*.

[114] E. Wei, D. Wicke, D. Freelan, and S. Luke, "Multiagent soft Q-learning," in *Proc. AAAI Spring Symp. Ser.*, 2018, pp. 1–7.

[115] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6382–6393.

[116] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[117] A. Kumar, J. Fu, G. Tucker, and S. Levine, "Stabilizing off-policy Q-learning via bootstrapping error reduction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–11.

[118] D. Silver, R. S. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, 2008, pp. 968–975.

[119] R. I. Brafman and M. Tennenholtz, "R-MAX—A general polynomial time algorithm for near-optimal reinforcement learning," *J. Mach. Learn. Res.*, vol. 3, pp. 213–231, Oct. 2002.

[120] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *J. Intell. Robot. Syst. Theory Appl.*, vol. 86, no. 2, pp. 153–173, May 2017.
[121] L. Kaiser *et al.*, "Model-based reinforcement learning for Atari," 2019, *arXiv:1903.00374*.
[122] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2829–2838.
[123] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32. Red Hook, NY, USA: Curran Associates, 2019, pp. 1–12.
[124] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–20.
[125] A. S. Vezhnevets *et al.*, "Feudal networks for hierarchical reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3540–3549.
[126] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, nos. 1–2, pp. 181–211, 1999.
[127] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. NIPS*, 2016, pp. 1–9.
[128] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proc. AAAI Conf. Artif. Intell.*, 2017, vol. 31, no. 1, pp. 1–9.
[129] M. Andrychowicz *et al.*, "Hindsight experience replay," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5055–5065.
[130] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–22.
[131] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings 1994*. Amsterdam, The Netherlands: Elsevier, 1994, pp. 157–163.
[132] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in Multi-Agent Systems and Applications—1*. Berlin, Germany: Springer, 2010, pp. 183–221.
[133] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–9.
[134] P. Sunehag *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proc. 17th Int. Conf. Auton. Agents MultiAgent Syst.*, 2018, pp. 2085–2087.
[135] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4295–4304.
[136] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends Cognit. Sci.*, vol. 3, no. 6, pp. 233–242, 1999.
[137] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. 21st Int. Conf. Mach. Learn. (ICML)*, 2004, pp. 1–8.
[138] E. Klein, M. Geist, B. Piot, and O. Pietquin, "Inverse reinforcement learning through structured classification," in *Proc. NIPS*, 2012, pp. 1–9.
[139] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*, 2006, pp. 729–736.
[140] C. Finn, P. Christiano, P. Abbeel, and S. Levine, "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models," 2016, *arXiv:1611.03852*.
[141] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adverserial inverse reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–15.
[142] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Found. Trends Robot.*, vol. 7, nos. 1–2, pp. 1–179, 2018.
[143] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, in JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.
[144] B. Kim, A.-M. Farahmand, J. Pineau, and D. Precup, "Learning from limited demonstrations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2859–2867.
[145] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–35, 2017.
[146] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4572–4580.
[147] L. Song, D. Li, X. Wang, and X. Xu, "AdaBoost maximum entropy deep inverse reinforcement learning with truncated gradient," *Inf. Sci.*, vol. 602, pp. 328–350, Jul. 2022.
[148] V. Ricardo and D. Youssef, "A perspective view and survey of meta-learning," *Artif. Intell. Rev.*, vol. 18, pp. 77–95, Sep. 2001.
[149] N. Schweighofer and D. Doya, "Meta-learning in reinforcement learning," *Neural Netw.*, vol. 16, no. 1, pp. 5–9, Jan. 2003.
[150] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5331–5340.
[151] J. X. Wang *et al.*, "Prefrontal cortex as a meta-reinforcement learning system," *Nature Neurosci.*, vol. 21, no. 6, pp. 860–868, 2018.
[152] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5307–5316.
[153] A. Nagabandi *et al.*, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," 2018, *arXiv:1803.11347*.
[154] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020, *arXiv:2005.01643*.
[155] C.-A. Cheng, X. Yan, and B. Boots, "Trajectory-wise control variates for variance reduction in policy gradient methods," in *Proc. Conf. Robot Learn.*, 2020, pp. 1379–1394.
[156] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," 2020, *arXiv:2006.04779*.
[157] G. Kahn, P. Abbeel, and S. Levine, "BADGR: An autonomous self-supervised learning-based navigation system," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 1312–1319, Apr. 2021.
[158] N. Rhinehart, R. McAllister, and S. Levine, "Deep imitative models for flexible inference, planning, and control," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–19.
[159] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Jan. 2009.
[160] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, no. 7, pp. 1633–1685, 2009.
[161] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," 2020, *arXiv:2009.07888*.
[162] Y. Lan, X. Xu, Q. Fang, Y. Zeng, X. Liu, and X. Zhang, "Transfer reinforcement learning via meta-knowledge extraction using auto-pruned decision trees," *Knowl.-Based Syst.*, vol. 242, Apr. 2022, Art. no. 108221.
[163] Y. Zhang, J. Ren, J. Li, Q. Fang, and X. Xu, "Deep Q-learning with explainable and transferable domain rules," in *Proc. Int. Conf. Intell. Comput.* Cham, Switzerland: Springer, 2021, pp. 259–273.

Xu Wang received the bachelor's degree in software engineering from Xidian University, Xi'an, China, in 2015, where he is currently pursuing the Ph.D. degree in computer science and technology.

His research interests include autonomous driving and deep reinforcement learning.

Sen Wang received the B.Sc. degree in electrical engineering and automation from Southeast University, Nanjing, China, in 2016. He is currently pursuing the master's degree with the School of Computer Science and Technology, Xidian University, Xi'an, China.

His current research interests include deep reinforcement learning.

Xingxing Liang received the B.A. degree from the College of Systems Engineering, National University of Defense Technology (NUDT), Changsha, China, in 2014, and the M.Sc. degree from the Science and Technology on Information Systems Engineering Laboratory, NUDT, in 2016, where he is currently pursuing the Ph.D. degree with the College of Systems Engineering.

His research interests include deep reinforcement and multiagent system for wargame.

Dawei Zhao received the Ph.D. degree in control science and engineering from the National University of Defense Technology, Changsha, Hunan, China, in 2018.

He is currently an Assistant Professor with the National Innovation Institute of Defense Technology, Beijing, China. His research interests include computer vision, machine learning, and autonomous vehicle.

Jincai Huang is a Professor with the National University of Defense Technology, Changsha, Changsha, Hunan, China, and a Researcher with the Science and Technology on Information Systems Engineering Laboratory. His main research interests include artificial general intelligence, deep reinforcement learning, and multiagent systems.

Xin Xu (Senior Member, IEEE) received the B.S. degree in electrical engineering from the Department of Automatic Control, National University of Defense Technology (NUDT), Changsha, China, in 1996, and the Ph.D. degree in control science and engineering from the College of Mechatronics and Automation, NUDT, in 2002.

He is currently a Full Professor with the College of Intelligence Science and Technology, NUDT.

Bin Dai received the Ph.D. degree in control science and engineering from the National University of Defense Technology, Changsha, Hunan, China, in 1998.

He is currently a Professor with the National Innovation Institute of Defense Technology, Beijing, China. His research interests include pattern recognition, data mining, and autonomous vehicle.

Qiguang Miao (Senior Member, IEEE) received the Ph.D. degree in computer application technology from Xidian University, Xi'an, China, in December 2005.

He is a Professor and a Ph.D. Student Supervisor with the School of Computer Science and Technology, Xidian University. He has published over 100 papers in the significant domestic and international journals or conferences. His research interests include machine learning, intelligent image processing, and malware behavior analysis and understanding.