Reinforcement Learning: A Literature Review

Chuanjin Fan, Wenyao Liu, Jialu Xu and Yiyang Zhao

February 2023

Abstract

Reinforcement learning is a research hotpot in the machine learning area, which is considering a process of agent-environment interaction, sequential decision making, and total reward maximization. Reinforcement learning is worthy of in-depth research and a wide range of applications in the real world, and represents a vital step toward the artificial general intelligence. In this survey, we review the research progress and development in the algorithms and applications for reinforcement learning. We start with a brief review of the principle of reinforcement learning, including Markov decision process, value function, and exploration v.s. exploitation. Next, we discuss traditional RL algorithms, including value-based, policy-based, and actor-critic algorithms. Additionally, an in-depth review of the deep reinforcement learning research is given, mainly introducing the value function-based and policy gradient-based deep reinforcement learning. The three tasks of multi-intelligent reinforcement learning, fully cooperative, fully competitive, and hybrid, are also briefly described. Finally, we summarize briefly and prospect the development trends of reinforcement learning.

1 Introduction

In recent years, Reinforcement Learning (RL) has become one of the top three machine learning techniques, along with supervised learning and unsupervised learning, because of its powerful exploration and autonomous learning capabilities [47]. Along with the booming development of deep learning, powerful deep reinforcement learning algorithms have emerged. These algorithms have been successfully applied in a variety of fields, including game competition [75, 59, 65], robot control [40, 63], urban transportation [88, 12, 46], and business activities [38, 102, 15], which have achieved remarkable results. David Silver, the father of AlphaGo [75], pointed out that deep learning + reinforcement learning = artificial general intelligence [74]. Subsequent research has shown that reinforcement learning is a crucial step to achieving general AI.

1.1 Markov Decison Processes

The core of reinforcement learning is the study of the interaction between an agent and its environment to make sequential decisions and maximize the reward by continuously learning the optimal strategy [80]. The reinforcement learning process can be described as a Markov decision process (MDP) as shown in Figure 1, where the parameter space can be represented as a five-tuple $\langle A, S, P, R, \gamma \rangle$, including action space A, state space S, state transfer $P : S \times S \times A \to [0, 1]$, reward $R : S \times A \to R$ and discount factor $\gamma \in [0, 1]$. In some cases, the agent cannot observe the full state space, and such problems are called partially observed Markov decision processes (POMDP), which are particularly common in multi-agent RL settings [61].

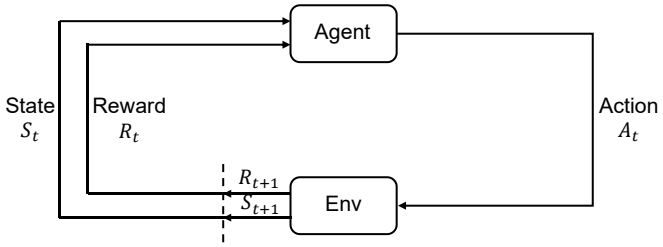

Figure 1: Agent-environment interaction in MDP. [80]

In the implementation, an agent observes its environment and current state $s_t \in S$ at moment t, takes an action $a_t \in A(S)$ according to policy π. At the next moment $t+1$, the environment gives a reward $r_{t+1} \in R \subset R$ according to the action taken by the agent and enters a new state s_{t+1}. The agent adjusts the strategy according to the obtained reward and moves to the following decision process. The sequence obtained in the MDP process is as follows:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \cdots, s_{n-1}, a_{n-1}, r_n \tag{1}$$

Through continuous learning, the agent will find the optimal strategy π^* that brings the maximum long-term cumulative return. After time t, the long-term cumulative return with discount factor $\gamma \in [0, 1]$ is as follows:

$$R_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2}$$

Considering the stochastic nature of the agent's environment and the delay in reward acquisition, the MDP uses a discount factor to reflect the fact that the deeper the reward at the end, the smaller the contribution to the cumulative reward at the current time t [80].

1.2 Value Function

After the agent learns the optimal policy π^*, the MDP degenerates into a Markov Reward Process (MRP) for a given policy. As a result, the state value function $V^{\pi}(s)$ and the action value function $Q^{\pi}(s, a)$ are denoted as:

$$V^{\pi^*}(s) = \mathrm{E}\left[R_t \mid s_t = s\right] \tag{3}$$

$$Q^{\pi^*}(s, a) = \mathrm{E}\left[R_t \mid s_t = s, a_t = a\right] \tag{4}$$

Converting the above equation into the form of Bellman optimality equations is:

$$V^{\pi^*}(s) = \max_a \sum_{s', r} P\left(s', r \mid s, a\right) \left[r + \gamma V^{\pi^*}\left(s'\right)\right] \tag{5}$$

$$Q^{\pi^*}(s, a) = \sum_{s', r} P\left(s', r \mid s, a\right) \left[r + \gamma \max_{a'} Q^{\pi^*}\left(s', a'\right)\right] \tag{6}$$

After obtaining the state value function and action value function, theoretically, the optimal strategy can be obtained by strategy iteration to solve the value function. However, strategy iteration is inefficient and computationally expensive in practice, so a manually designed linear function or a nonlinear function (e.g., neural network) is usually used to approximate the value function [53].

1.3 Exploration and Exploitation

In reinforcement learning problems, an agent needs to balance exploration and exploitation to obtain the optimal strategy and thus maximize the cumulative reward [85]. Taking random actions to fully explore all uncertain strategies may experience a large number of poor strategies, resulting in low rewards. However, the continuous use of the existing optimal strategy to select the highest-value actions and the lack of exploration of the state space may lead to missing the global optimal strategies and unstable rewards.

For the exploration and exploitation problem in reinforcement learning, simple greedy exploration, i.e., the ε-greedy method, is mainly used for improvement, where $\varepsilon \in [0, 1]$ is a small value close to 0. In the $\varepsilon-$ greedy method, the agent has a higher probability of picking the action with the highest value under the existing optimal policy $a = \arg\max_{a \in A} Q(s, a)$ with $1-\varepsilon$. Still, it keeps a small probability of picking the action randomly with ε to achieve the continuous exploration of the state space. During the implementation, the ε of greedy exploration decays until it decreases to a fixed, low exploration rate. In addition to the most commonly used greedy exploration methods such as ε-greedy, methods such as Upper Confidence Bound (UCB) [5] also consider the size of the value function itself and the number of searches, which can automatically achieve an automatic balance between exploration and exploitation, and can effectively reduce the number of explorations.

2 Classical Algorithms

From Bellman's dynamic programming approach [11] to AlphaGo's defeat of the human Go champion [75], reinforcement learning has evolved over the past 60 years and become the most popular research and application direction in machine learning. In 2006, the introduction of deep learning [37] led to the second wave of machine learning, which has continued to gain momentum in both academic and business circles and successfully contributed to the boom of deep reinforcement learning after 2010.

There are many ways to classify reinforcement learning algorithms, such as model-free and model-based algorithms, based on whether they construct a model or not. Depending on whether the execution policy and evaluation policy are compatible, the algorithms may also be divided into on-policy and off-policy. According to the method's update mechanism, they may also be separated into the Monte-Carlo (MC) algorithm for round updates and the Temporal-Difference (TD) algorithm for single-step updates. Among them, the model-free, on-policy, and TD algorithms are the mainstream directions under their respective classifications, and there is some crossover in the algorithms under different categories. In addition, based on the action selection method of the agent, the reinforcement learning algorithms can be classified into value-based, policy-based, and actor-critic, which is the most mainstream classification at present [16]. A comparison of the three main classes of reinforcement learning algorithms is given in Table 1, and each type is described in the following section.

2.1 Value-based

Value-based reinforcement learning algorithms implicitly construct optimal policies by obtaining the optimal value function and selecting the action corresponding to the maximum value function. Representative algorithms include Q-learning [18], SARSA [67], and Deep Q-Network (DQN) [57, 58], which are combined with deep learning. Most of these methods obtain the optimal value function by dynamic programming or value function approximation, and single-step or multi-step updates are performed by time-difference (TD) methods instead of Monte Carlo (MC) round robin updates to ensure efficiency. For example, the off-policy Q-learning algorithm uses the non-exploration strategy to compute the TD error, while the on-policy SARSA algorithm uses the exploration strategy to calculate the TD error. The value-based algorithm has high sample utilization, low variance of the value function valuation, and does not easily fall into local optimum. However, these algorithms can only solve

Table 1: Comparison of three mainstream reinforcement learning algorithms.

Category	Representative Algorithm	Mechanism	Advantage	Disadvantage	Applicable scenario
Value-based	Q-learning[18] SARSA[67] DQN Series[57, **?**, 84, 68, 86, 22, 9, 36]	Calculate the value function, select the action corresponding to the maximum value function, and implicitly obtain a deterministic strategy.	High sample utilization, low variance of value function valuation, and less likely to fall into local optimum	Prone to overfitting, limited problem complexity, and poor convergence properties.	Discrete motion space
Policy-based	REINFORCE[90] TRPO[70] PPO[73]	Does not depend on the value function, maximizes the cumulative return to select actions, updates the strategy, and usually obtains the optimal stochastic strategy.	Its strategy function is easy to compute, comes with stochastic exploration, good stability and convergence properties.	Lower sample utilization, easily falls into local optimum, and the evaluation strategy is usually less efficient with large variance.	Discrete or continuous action space
Actor-critic	AC[41] A3C[56] DPG[77] DDPG[48] TD3[25] SAC[29]	Actor (policy-based) updates the policy according to the value function and selects the action. Critic (value-based) calculates the value function according to the action and updates it in a single step.	High sample utilization, low variance of value function estimation, fast overall training speed.	Insufficient stability of the algorithm, sensitive to hyperparameters.	Discrete or continuous action space

discrete action space problems, which are prone to overfitting and limited in the complexity of the issues they can handle. At the same time, the value-based algorithm has poor convergence properties because the action selection is susceptible to the change in the value function.

The DQN algorithm [57], which uses a Convolutional Neural Network (CNN) to estimate the value function, is the first deep reinforcement learning algorithm that extends the application of value-based methods to high-dimensional problems and continuous space problems. The DQN end-to-end reinforcement learning algorithm uses experience replay and target network to stabilize the value function estimation, significantly reducing the requirement for domain-specific knowledge and improving the generalization capability of the algorithm. Since then, DQN algorithms have evolved into a variety of variations, including the Double DQN algorithm, which employs different network evaluation strategies and estimated value functions [84]. Different experience replay frequencies are used in the prioritized experience replay algorithm [68]. Using competing network structures, the dueling DQN algorithm estimates the state value function and the associated dominance function individually before combining them to assess the action value function as a whole [86]. The Distributional DQN (C51) algorithm [9] expands to distributed value functions, the NoisyNet technique [22] adds network parameter noise to improve exploration, and the Rainbow DQN algorithm [9] combines all three algorithms. These DQN algorithms can effectively solve the overfitting problem with higher learning efficiency, better value function evaluation, more adequate spatial search capability, and wider applicability. The performance comparison of DQN algorithms and various variants of algorithms is shown in Figure 2.

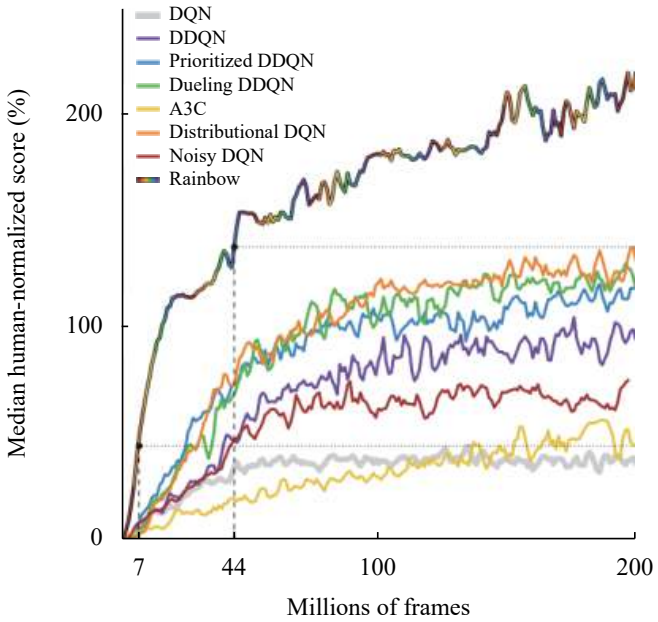

Figure 2: Performance of various DQN algorithms in Atari games. [36]

It should be noted that although DQN and its variants have achieved excellent performance in discrete action space problems represented by video games, and even overwhelmingly beat human players in some games [58], DQN algorithms for discrete action space cannot cope with a large number of continuous action space problems such as robot arm control and vehicle driving in real production and life. At the same time, although the asynchronous DQN algorithm has a high sample efficiency compared to the synchronous algorithms such as SARSA, even the most advanced Rainbow DQN algorithm in the DQN family still needs to learn about 15 million frames and train for one day to reach the level of human players in a simple Atari game, as shown in Figure 2 [36]. In contrast, a human can master the same game in just a few hours. Therefore, the sampling efficiency of the DQN algorithm still needs to be negligible.

2.2 Policy-based

Policy-based reinforcement learning algorithms span the value function and directly search for the best policy. It updates the policy parameters by maximizing the cumulative returns, and is divided into gradient-based and gradient-free algorithms [4]. The gradient-free algorithm [26, 43] can better handle low-dimensional problems, and the policy-gradient-based algorithm is still the most used class of reinforcement learning algorithms, especially when dealing with complex issues, such as the fantastic performance of AlphaGo [75] in Go games. Compared with the value-based algorithm, the policy-based algorithm can handle discrete or continuous space problems and have better convergence. At the same time, the policy-based approach has a large trajectory variance and low sample utilization, which makes it easy to fall into the local optimum dilemma.

The classical policy gradient algorithm REINFORCE [90] uses the MC method to estimate the gradient strategy, which has good stability but low sample efficiency. Meanwhile, the MC method contains information over the whole trajectory, which introduces a significant variance in estimating the strategy gradient. By introducing unbiased estimation with a small amount of noise, for example, by subtracting the baseline from the returns, the estimation variance can be effectively reduced. In 2002, Kakade proposed the natural policy gradient [39] to improve the stability and convergence speed of the algorithm, which led to subsequent trust region methods, such as the well-known Trust Region Policy Optimization (TRPO) [70]) and Proximal Policy Optimization (PPO) [73]. Both TRPO and PPO are on-policy algorithms,

which are based on the classical policy gradient algorithm by artificially or adaptively selecting hyperparameters and constraining the update step size within a specific range to ensure monotonically undiminished returns at each step and continuously obtaining better policies to prevent policy collapse. In addition, Nachum et al. in 2017 proposed Trust Path Consistency Learning (Trust PCL), an off-policy confidence path consistency learning algorithm with higher sample efficiency [61], and in the same year, Heess et al. extended the PPO algorithm to the distributed policy gradient Distributed PPO algorithm [35].

TRPO and PPO algorithms have been chosen as the base algorithms for many research works due to their excellent experimental results [19, 20, 28, 54], and PPO has become the default algorithm of OpenAI [47]. However, although TRPO and PPO algorithms have gained much attention in academic research because of their excellent hyperparametric performance, as typical synchronous policy algorithms, each policy update requires a large number of samples to be sampled under the current policy for training and ensuring the convergence of the algorithm. Therefore, the limitations of TRPO and PPO algorithms are apparent, such as the low sampling efficiency and the need for a large amount of computing power, which significantly limits the application of the algorithms.

2.3 Actor-Critic

The actor-critic algorithm combines the value-based (i.e., critic) approach with the policy-based (i.e., actor) approach, learning both the policy and the value function [41]. The actor trains the policy based on the value function fed by the critic, while the critic trains the value function using the time difference (TD) method for single-step updates. The framework of the Actor-critic algorithm is shown in Figure 3. In general, actor-critic is considered a class of policy-based methods, with the unique feature of using value as a benchmark for policy gradient, which is an improvement of policy-based methods for estimating variance. Actor-critic combines the advantages of policy-based and value-based methods, such as slight estimation variance of the value function, high sample utilization, and fast training speed of the algorithm. At the same time, the actor-critic method also inherits the corresponding disadvantages, such as the actor's insufficient sample exploration and the critic's propensity for overfitting. Moreover, the critic, which is challenging to converge, has worse convergence properties when combined with the actor. These problems have been alleviated to some extent by introducing deep learning in the later developed algorithms.

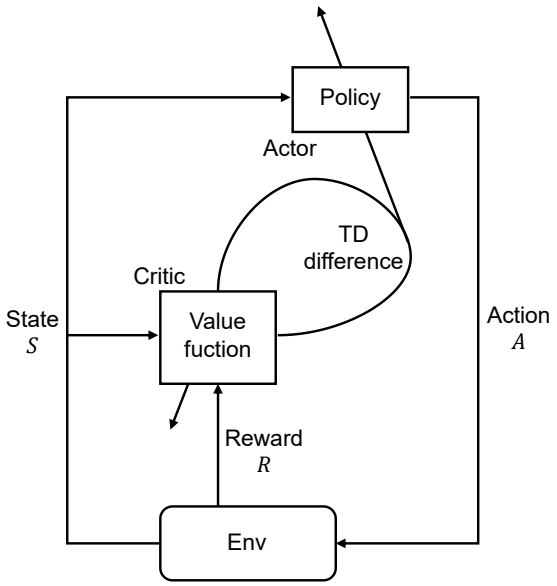

Figure 3: Actor-critic algorithm framework.

In recent years, many improved actor-critic algorithms have been developed. The most representative ones include the deterministic policy gradient algorithm (DPG) [77] and its deeply improved version deep deterministic policy gradient (DDPG) [48], the asynchronous advantage actor-critic algorithm (A3C) [56], the twin-delayed deep deterministic policy gradient (TD3) [25], and the soft actor-critic algorithm (SAC) [29]. The DPG algorithm [77] only integrates deterministic policy gradients in the state space, which dramatically reduces the sampling requirement and can handle problems with large action spaces. The DDPG algorithm [48] inherits the target network of DQN and uses Critic estimation of policy gradients with asynchronous policies to make the training more stable and straightforward. The well-known A3C algorithm [56] uses online Critic to integrate the policy gradient and reduce the correlation of training samples, which improves the sampling efficiency and training speed while ensuring stability and unbiased estimation. TD3 algorithm [25] introduces a double DQN with better performance based on DDPG, and takes the minimum value between two Critic to limit overfitting. In the SAC algorithm [29], which is the same period as TD3, the Actor has the maximum entropy in addition to the ultimate reward, which significantly improves the exploration ability of the algorithm. Figure 4 compares the performance of several state-of-the-art policy-gradient algorithms on the same reinforcement learning benchmark problem, with the overall comparison being approximately SAC = TD3 > DDPG = TRPO = DPG > VPG [3], where VPG refers to the classical policy-gradient algorithm, such as REINFORCE [90].

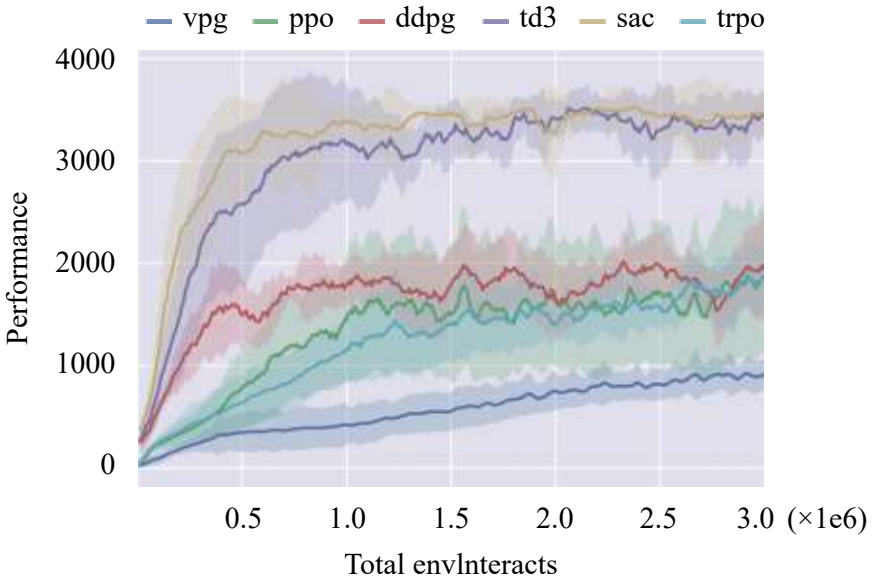

Figure 4: Comparison of the effectiveness of policy-based reinforcement learning algorithms in Hopper problem [3].

Actor-critic algorithms such as DPG, DDPG, TD3, and SAC are typical of off-policy algorithms, among which critic adopts Q-learning and DQN algorithms with off-policy. The A3C can be trained synchronously or asynchronously according to the algorithm used by critic, and can be applied to both on-policy and off-policy. Therefore, actor-critic algorithms are mostly off-policy algorithms, which can solve the problem of sampling efficiency by experience replay. However, the coupling of strategy update and value evaluation leads to the lack of stability of the algorithm, especially sensitive to hyperparameters. Actor-critic algorithms are difficult to reconcile and reproduce, and their robustness is one of the central concerns when extended to applications.

3 Deep Reinforcement Learning

3.1 Deep Reinforcement Learning based on Value Function

3.1.1 Deep Q-Network

Mnih et al. [57, 58] proposed the Deep Q-Network (DQN) model by combining the convolutional nerve network with the Q-learning [87] algorithm in traditional RL. This model is used to handle control tasks based on visual perception, which is a pioneering work in the field of DRL.

The input to the DQN model is the four pre-processed images closest to the current moment. This input is nonlinearly transformed by three convolutional layers and two fully connected layers to produce the Q-value of each action in the output layer. Figure 5 shows the model architecture of the DQN.

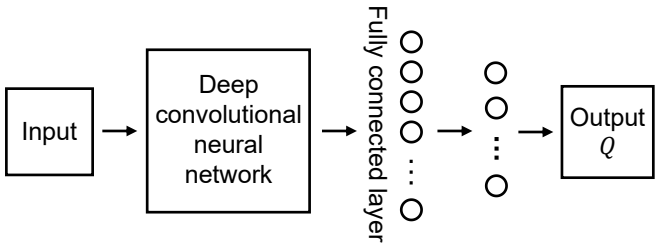

Figure 5: Model structure of DQN.

Figure 6 depicts the training flow of DQN. To alleviate the problems such as instability in the representation of value functions by nonlinear networks, DQN mainly makes three improvements to the traditional Q-learning algorithm.

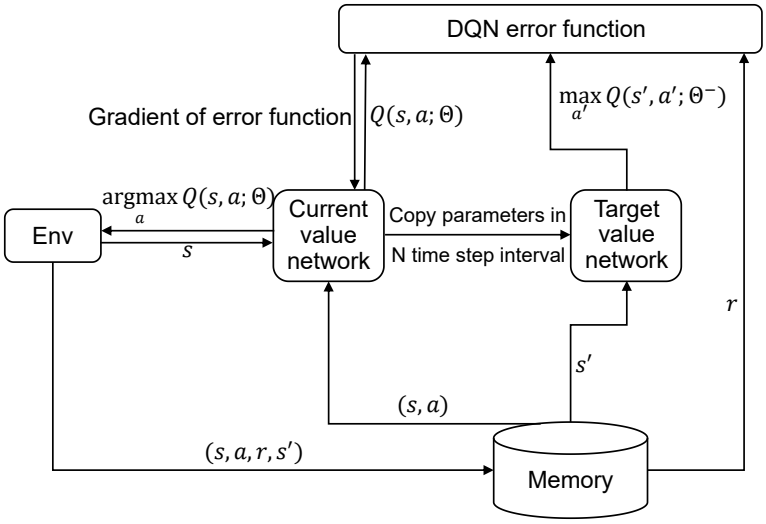

Figure 6: Training process of DQN.

1) DQN uses the experience replay mechanism [49] during the training process, processing the transfer samples $e_t = (s_t, a_t, r_t, s_{t+1})$ online. At each time step t, the transfer samples obtained from the interaction between the agent and the environment are stored in the playback memory unit $D = \{e_1, \cdots, e_t\}$. During training, a small number of transfer samples are randomly selected from D each time, and the network parameters θ are updated using Stochastic Gradient Descent (SGD). When training a deep network, the samples are usually required to

be independent. This random sampling method dramatically reduces the correlation between the samples and thus improves the algorithm's stability.

2) In addition to using a deep convolutional network to approximate the current value function, the DQN also operates a separate network to generate the target Q values. Specifically, $Q(s, a \mid \theta_i)$ represents the output of the current value network, which is used to evaluate the value function of the current state action pair. $Q\left(s, a \mid \theta_i^-\right)$ denotes the output of the target value network, generally using $Y_i = r + \gamma \max_{a'} Q\left(s',\ a' \mid \theta_i^- theta_i^-\right)$ approximates the optimization objective of the value function, i.e., the objective Q value. The parameters of the current value network θ are updated in real time, and the parameters of the existing value network are copied to the target value network after every N iterations. The network parameters are updated by minimizing the mean square error between the current Q value and the target Q value. The error function is:

$$L(\theta_i) = \mathrm{E}_{s,a,r,s'}\left[\left(Y_i - Q(s, a \mid \theta_i)\right)^2\right] \tag{7}$$

By taking the partial derivative of the parameter θ, the following gradient is obtained:

$$\nabla_{\theta_i} L(\theta_i) = \mathrm{E}_{s,a,r,s}\left[\left(Y_i - Q(s, a \mid \theta_i)\right) \nabla_{\theta_i} Q(s, a \mid \theta_i)\right] \tag{8}$$

After introducing the target value network, the target Q value is kept constant for a period of time, which reduces the correlation between the current Q value and the target Q value to a certain extent and improves the stability of the algorithm.

3) DQN narrows the reward and error term to a limited interval, ensuring that both Q and gradient values are within a reasonable range and improving the algorithm's stability. Experiments show that DQN exhibits a competitive level comparable to human players when solving complex problems in realistic-like environments such as Atari 2600 games [58]. Even in some less complicated non-strategic games, DQNs outperformed experienced human players. In solving various visual perception-based DRL tasks, DQNs use the same set of network models, parameter settings, and training algorithms, demonstrating the adaptability and versatility of the DQN approach.

3.1.2 Improvement of DQN Training Algorithm

When approximating the optimization objective of the value function in DQN using $Y_i = r + \gamma \max_{a'} Q\left(s', a' \mid \theta_i^-\right)$, the maximum Q in the next state is selected each time value in the next state. The selection and evaluation of the actions are based on the parameters θ^- of the target value network, which causes the problem of overestimation of Q values during the learning process.

Van Hasselt et al. [84] proposed a Deep Double Q-Network (DDQN) algorithm based on the double Q learning algorithm [31] (double Q-learning). There are two different sets of parameters in double Q learning: θ and θ^-. Where θ is used to select the action corresponding to the maximum Q value, and θ^- is used to evaluate the Q value of the optimal action. The two sets of parameters separate action selection from policy evaluation, reducing the risk of overestimating the Q value. Thus DDQN uses the parameter θ of the current value network to select the optimal action and the parameter θ^- of the target value network to evaluate that optimal action. The target Q value takes the following form:

$$Y_i^{\mathrm{DDQN}} = r + \gamma Q\left(s', \arg\max Q(s', a \mid \theta_i), \theta_i^-\right) \tag{9}$$

DDQN is otherwise consistent with DQN. Experimentally, DDQN is able to estimate more accurate Q values and obtain more stable and effective strategies in some Atari 2600 games.

In addition, Lakshminarayanan et al. [79] proposed the Dynamic Frame Skip Deep Q-Network (DFDQN) algorithm using dynamic frame skipping instead of repeating k times per moment in DQN. Experiments show that DFDQN achieves better performance in some Atari 2600 games. Van Hasselt et al. [83] replaced the interval trimming method in the traditional

DQN with a dynamic normalization operation called Pop-Art. To accelerate the convergence of deep networks without losing important state information, Francois-Lavet et al. [23] used adaptive discount factors and learning rates in DQNs.

3.1.3 Improvement of DQN Model Structure

In the traditional RL approach, partial observability of state information has been a pressing problem. DQN, by stacking the four nearest historical images to the current moment to form the input states, effectively alleviates the problem of partial observability of state information but increases the computational and storage burden of the network. To address this problem, Hausknecht et al. [32] proposed the DRQN model by using a recurrent neural network structure to memorize the continuous historical state information on the time axis.

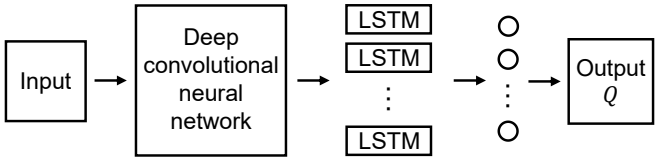

Figure 7: DRQN model structure

As shown in Figure 7, the DRQN replaces the first fully connected layer of the DQN with 256 LongShort-Term Memory (LSTM) components. The model's input is only an image at the current moment, which reduces the computational resources used by the deep network to perceive the image features. Experiments show that DRQN performs better than DQN in the case of partial state observable. Therefore, the DRQN model is suitable for complex tasks where the problem of partial state observability prevails.

With the proposal of various novel network modules in the field of DL, the future DRL models will develop in the direction of structural diversification and modular complexity. For example, the powerful perceptual capability of deep residual networks can be used to improve the agent's representation of complex state spaces. In addition, the visual attention mechanism (VAM) [93] can be added to the model to accelerate the learning process by focusing the agent's attention on the region that is conducive to making decisions in different states.

3.2 Deep Reinforcement Learning based on Policy Gradients

Strategy gradient is a commonly used strategy optimization method, which updates the strategy parameters by continuously calculating the gradient of the desired total reward of the strategy for the strategy parameters, and finally converges to the optimal strategy [81]. Therefore, in solving the DRL problem, a deep neural network with parameter θ can be used to parametrically represent the strategy, and the strategy gradient method can be used to optimize the strategy. It is worth noting that the first choice in solving the DRL problem is often to adopt the strategy gradient-based algorithm. The reason is that it can directly optimize the desired total reward of the strategy and search for the optimal strategy directly in the strategy space in an end-to-end manner, eliminating the tedious intermediate steps. Therefore, compared with DQN and its improved models, the policy gradient-based DRL approach is more applicable and effective in policy optimization.

3.2.1 Origin and Development of Deep Strategy Gradient

The strategy gradient method is a method that directly uses an approximator to approximate the representation and optimize the strategy and finally obtain the optimal strategy. This method optimizes the desired total reward of the strategy:

$$\max_{\theta} \mathrm{E}\left[R \mid \pi_{\theta}\right] \tag{10}$$

where $R = \sum_{t=0}^{T-1} r_t$ denotes the sum of the rewards obtained within an episode. The most common idea of strategic gradient is to increase the probability of the occurrence of episodes with higher total rewards. The specific procedure of the strategic gradient method is as follows:

Suppose the trajectory of states, actions, and rewards of a complete episode is $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$. Then the policy gradient is expressed in the following form:

$$g = R\nabla_\theta \sum_{t=0}^{T-1} \log \pi \left(a_t \mid s_t; \theta\right) \tag{11}$$

Using this gradient to adjust the strategy parameters:

$$\theta \leftarrow \theta + \alpha g \tag{12}$$

where α is the learning rate, which controls the rate at which the policy parameters are updated. The $\nabla_\theta \sum_{t=0}^{T-1} \log \pi \left(a_t \mid s_t; \theta\right)$ gradient term in Eq. (11) indicates the direction that increases the probability of occurrence of the trajectory τ. After multiplying the score function R, it is possible to make the higher the total reward τ within a single episode the more "hard to pull together" the probability density. That is, if many trajectories with different total rewards are collected, the above training process will shift the probability density toward the trajectories with higher total rewards and maximize the probability of occurrence of the trajectories with higher rewards τ.

However, in some cases, the total reward R for each episode is not negative, and all the gradients g have values greater than or equal to 0. In this case, each trajectory τ encountered in the training process will "pull" the probability density in the positive direction, which dramatically slows down the learning speed. This makes the variance of the gradient g very large, so some normalization operation can be used on R to reduce the variance of the gradient g. This technique allows the algorithm to increase the probability of the occurrence of trajectories τ with a larger total reward R and decrease the probability of the occurrence of trajectories τ with a smaller total reward R. Based on the above idea, Williams et al. [91] proposed the REINFORCE algorithm, which unifies the form of the strategy gradient as

$$g = \nabla_\theta \sum_{t=0}^{T-1} \log \pi \left(a_t \mid s_t; \theta\right) (R - b) \tag{13}$$

where b is a baseline associated with the current trajectory τ, which is usually set as an expectation estimate of R, with the purpose of reducing the variance of R. It can be seen that the more R exceeds the baseline b, the higher the probability of the corresponding trajectory τ being selected. Therefore, in the large-scale state DRL task, the optimal policy can be solved by parametrically representing the policy by a deep neural network and using the traditional policy gradient method.

In addition, another way of thinking about optimization strategies is to increase the probability of "good" actions. in RL. In RL, the dominance function is generally used to evaluate the In RL, good and bad actions are typically assessed by the dominance function, so the dominance function term can be used to construct the strategy Gradient.

$$g = \nabla_\theta \sum_{t=0}^{T-1} \hat{A}_t \log \pi \left(a_t \mid s_t; \theta\right) \tag{14}$$

where $\dot{A}_t$ denotes an estimate of the (s_t, a_t) dominance function of the state action, which is usually constructed as follows:

$$\hat{A}_t^\gamma = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - V\left(s_t\right) \tag{15}$$

where $\gamma \in [0,1]$ denotes the discount factor. The sum of the discounted rewards $r_t + \gamma r_{t+1} + \gamma r_{t+2} + \cdots$ is equivalent to R in Eq. (13), and the discounted state value function $V(s_t)$ is equivalent to the base b in Eq. (13). When $\hat{A}_t^{\gamma} > 0$, the probability of the corresponding action being selected increases, while $\hat{A}_t^{\gamma} < 0$, the probability of the corresponding action being selected decreases.

In addition, Hafner et al. [30] used the value function to estimate the reward sum with a discount, which further reduces the variance of the gradient term. The one-step truncated $\hat{A}_t^{\gamma}$ is then expressed as:

$$\hat{A}_t^{\gamma} = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{16}$$

Similarly, the two-step truncated $\hat{A}_t^{\gamma}$ is denoted as

$$\hat{A}_t^{\gamma} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \tag{17}$$

However, using the value function to estimate the discounted reward sum, also produces some estimation bias. In order to reduce the variance while keeping the bias small, Schulman et al. [72] proposed a generalized advantage function:

$$\hat{A}_t^{\gamma} = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots + (\gamma\lambda)^{T-t-1}\delta_{T-1} \tag{18}$$

where, $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$.λ is a modulation factor, ranging from $0 < \lambda < 1$. When λ is close to 0, $\hat{A}_t^{\gamma}$ is low variance and high error, and when λ is close to 1, $\hat{A}_t^{r}$ is high variance and low error. The shortcoming of the generalized dominance function-based policy gradient approach is that it is difficult to determine a proper step size parameter α to ensure the stability of learning in the process of global optimization of the policy using Eq. (12). To address this problem, Schulman et al. [71] proposed a method called Trust Region Policy Optimization (TRPO). The core idea of TRPO is to forcibly restrict the KL difference between the old and new policy prediction distributions on the same batch of data, thus avoiding parameter update steps that cause too much change in the policy. To extend the application to large scale state space DRL tasks, the TRPO algorithm uses a deep neural network to parameterize the strategies and achieve end-to-end control while receiving only the original input images. Experiments show that TRPO performs well in robot control in a series of 2D scenes and the Atari 2600 game task. Since then, Schulman et al. [72] have tried to combine the generalized dominance function with the TRPO method and achieved a breakthrough in a series of robotic control tasks in 3D scenarios.

In addition, another research direction of deep strategy gradient methods is to facilitate strategy search by adding additional human supervision. The famous AlphaGo Go robot, for example, first uses supervised learning to predict human movement behavior from human experts' games, and then uses a strategy gradient approach to fine-tune the strategy parameters for the real goal of winning a match [75]. However, in some tasks where supervised data are lacking, such as robot control in realistic scenarios, the policy search process can be handled by the guided policy search method [45]. In the real scenario where only the original input signal is accepted, the guided policy search achieves the manipulation of the robot.

3.2.2 Deep Strategy Gradient Approach based on Actor-Critic

The basic idea of deep strategy gradient methods is to directly optimize the strategies parametrically represented by deep neural networks through various strategy gradient methods. Such methods need to sample trajectories $\{\tau_i\}_{i=1}^{N}$ of batch size N at each iteration step to update the policy gradient. However, it isn't easy to obtain large amounts of training data in many complex real-world scenarios online. For example, in real-world robot manipulation tasks, it is costly to collect and utilize a large amount of training data online, and the continuous nature of the motion makes it impossible to achieve good coverage by extracting batch trajectories online. These problems can lead to the emergence of locally optimal solutions. The traditional Actor-Critic (AC) framework in RL can be extended to the deep policy gradient

approach to address this problem. Figure 8 illustrates the learning structure of the deep policy gradient method based on the AC framework.

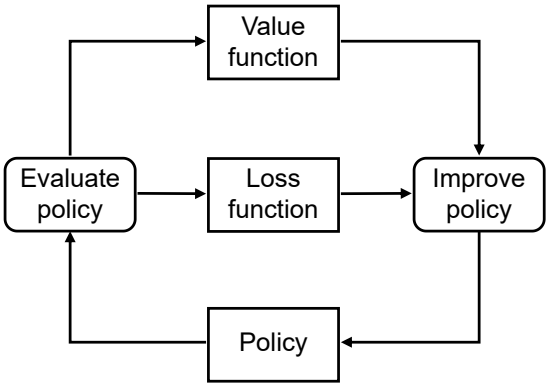

Figure 8: Structure of deep policy gradient method based on actor-critic

Lillicrap et al. [48] adapted the Deterministic Policy Gradient (DPG) method using DQN to extend the Q learning algorithm and proposed a Deep Deterministic Policy Gradient (DDPG) algorithm based on the AC framework, which can be used to solve the DRL problem on continuous action space. The DDPG uses deep neural networks with parameters θ^{μ} and θ^{α} to represent the deterministic policy $a = \pi\left(s \mid \theta^{\mu}\right)$ and the value function $Q\left(s, a \mid \theta^{Q}\right)$, respectively. Among them, the policy network is used to update the policy, corresponding to the actors in the AC framework; the value network is used to approximate the value function of the state action pair and provide the gradient information, corresponding to the critics in the AC framework. In DDPG, the objective function is defined as a reward sum with discount:

$$J\left(\theta^{\mu}\right) = \mathrm{E}_{\theta^{\mu}}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots\right] \tag{19}$$

Then, the stochastic gradient descent method is used for end-to-end optimization of the objective function. Silver et al. [70] proved that the gradient of the objective function for θ^{μ} is equivalent to the expected gradient of the Q-valued function for θ^{μ}:

$$\frac{\partial J\left(\theta^{\mu}\right)}{\partial \theta^{\mu}} = \mathrm{E}_s\left[\frac{\partial Q\left(s, a \mid \theta^{Q}\right)}{\partial \theta^{\mu}}\right] \tag{20}$$

According to the deterministic strategy $a = \pi\left(s \mid \theta^{\mu}\right)$ we get

$$\frac{\partial J\left(\theta^{\mu}\right)}{\partial \theta^{\mu}} = \mathrm{E}_s\left[\frac{\partial Q\left(s, a \mid \theta^{Q}\right)}{\partial a}\frac{\partial \pi\left(s \mid \theta^{\mu}\right)}{\partial \theta^{\mu}}\right] \tag{21}$$

The reviewer network is updated by updating the value network in the DQN, at which point the gradient information is

$$\frac{\partial L\left(\theta^{Q}\right)}{\partial \theta^{Q}} = \mathrm{E}_{s,a,r,s' \sim D}\left[\left(y - Q\left(s, a \mid \theta^{Q}\right)\right)\frac{\partial Q\left(s, a \mid \theta^{Q}\right)}{\partial \theta^{Q}}\right] \tag{22}$$

where $y = r + \gamma Q\left(s', \pi\left(s' \mid \hat{\theta}^{\mu}\right) \mid \hat{\theta}^{Q}\right), \hat{\theta}^{\mu}$ and $\hat{theta}^{Q}$ denote the parameters of the target policy network and the target value network, respectively. DDPG uses an empirical replay mechanism to obtain training samples from D, and passes the gradient information about the actions from the critic network to the actor network by the Q-value function. The parameters of the strategy network are updated in the direction of the Q-values according to equation (25).

Experiments show that DDPG not only performs stably in a series of continuous action space tasks, but also requires far fewer time steps to find the optimal solution than DQN. Compared with the value function-based DRL approach, the AC framework-based deep policy gradient approach optimizes the strategy more efficiently and solves it faster.

However, the policies are generally random in complex environments with noise disturbances.DDPG uses a deterministic policy gradient method. For scenarios with random environments, the method is not applicable. To address this problem, Heess et al. [34] proposed a general framework for continuous action space tasks called Stochastic Value Gradient (SVG) method. The value gradient inverse update (VGR) approach was put forth by Balduzzi et al. [6] and is based on the notion of compatible function approximation. By fusing multiple policy networks and the corresponding value networks, Peng et al. [66] proposed a deep strategy gradient method based on the mixture of actor-critic Experts (MACE). This method has made substantial progress in adaptive robot control tasks.MACE has a faster learning speed than the deep policy gradient method guided by a single AC framework. Subsequently, Heess et al. [33] further extended the applicability of the DPG and SVG algorithms using a cyclic nerve network and proposed the Recurrent Deterministic Policy Gradient (RDPG) and the Recurrent Stochastic Value Gradient (RSVG). RDPG and RSVG can handle the control task of continuous actions in a series of partially observable scenarios. The depth policy gradient approach was further extended by Hausknecht et al. [32] to the issue of parametric continuous action space. Later, Schulman et al. [69] proposed a formal stochastic computation graphs model to investigate complex depth strategy gradients that include both stochastic and deterministic operations.

3.2.3 Asynchronous Advantage Actor-Critic

Different types of deep neural networks provide efficient operational representations for policy optimization tasks in DRL. To alleviate the instability that occurs when traditional policy gradient methods are combined with neural networks, various types of deep policy gradient methods (e.g., DDPG, SVG, etc.) use an empirical replay mechanism to eliminate the correlation between the training data. However, the empirical replay mechanism has two shortcomings: 1) each real-time interaction between the agent and the environment requires a lot of memory and computational power. 2) the empirical replay mechanism requires the agent to adopt the off-policy method for learning, and the off-policy method can only be updated based on the data generated by the old policy. To address these problems, Mnih et al. [56] proposed a lightweight DRL framework based on Asynchronous Reinforcement Learning (ARL), which can use asynchronous gradient descent to optimize the parameters of the network controller and can combine multiple RL algorithms. Among them, the Asynchronous Advantage Actor-Critic (A3C) algorithm performs best on various continuous action space control tasks.

Specifically, the A3C algorithm executes multiple agents asynchronously using the CPU multi-thread function in parallel. Thus, the parallel agents will experience many different states at any given time, eliminating the correlation between the state transfer samples generated during the training process. Therefore, this low-consumption asynchronous execution can be an excellent alternative to the experience playback mechanism.

The A3C algorithm reduces the hardware requirements during training. The depth policy gradient algorithm relies heavily on the computationally powerful Graphics Processing Unit (GPU), while the A3C algorithm requires only a standard multicore CPU in practice. From Table 2, the A3C algorithm reduces the hardware requirements of the model by applying multithreading techniques, and the average performance of the A3C algorithm on the Atari 2600 game task is significantly improved with less training time. Moreover, the A3C algorithm can learn an effective strategy for walking a 3D maze based only on the original visual input. In addition, the A3C algorithm can be widely applied to various continuous action space problems. In summary, the A3C algorithm can be widely used for multiple 2D, 3D discrete and continuous action space tasks, achieving the best results. This shows that A3C is one of

the most popular and successful DRL algorithms. Combining A3C with recent deep policy gradient algorithms may further improve its performance.

Table 2: Average training time and game performance improvement of different DRL models on 57 Atari games.

Model	Training condition	Training time(day)	Average performance improvement (percentage)
DQN	GPU	8	121.9
Gorila	100 mainunit	4	215.2
DDQN	GPU	8	332.9
Dueling DDQN	GPU	8	343.8
Prioritized DDQN	GPU	8	463.6
A3C, FF	GPU	1	344.1
A3C, FF	GPU	4	496.8
A3C LSTM	GPU	4	623.0

4 Multi-Agent Reinforcement Learning

Multi-intelligent systems evolved from distributed artificial intelligence, which has the characteristics of autonomy, distributed, and coordinated, with learning, reasoning, and self-organizing capabilities. Although the concept of agent emerged in the 1940s, there was little research on multiple agents as a whole system until the 1970s. It was not until the late 1980s that distributed AI began to develop significantly. Building on the concept of game theory, distributed AI evolved and eventually formed multi-agent systems. Later, among the distributed problem solving models for multi-agent systems, distributed constraint reasoning (DCR) models, such as distributed constraint satisfaction problems (DCSP) and distributed constraint optimization problems (DCOP), were more widely studied and used. DCR has been applied to a variety of distributed problems, such as distributed sensor task assignment and distributed constraint optimization. DCR has been applied to a variety of distributed problems, such as distributed sensor task assignment and distributed meeting scheduling strategies.

Recently, a large amount of research has focused on finding methods to solve uncertainty problems for multi-agent systems. Among various models and solution methods, the distributed Markov decision process (Dec-MDP) and the distributed partially observable Markov decision process (Dec-POMDP) are the two most commonly used models in the uncertainty case. Unfortunately, solving Dec-POMDP is usually difficult. The development of reinforcement learning has provided a new train of thought for multi-agent systems to solve problems such as uncertainty, and multi-agent reinforcement learning is becoming the most popular area of interest among the many subfields of MAS. The following is a brief introduction of the basic concepts and classical algorithms of multi-agent reinforcement learning.

First, the environment of MARL is a stochastic game framework based on Markov decision processes, which is such a tuple$< S, A_1, \cdots, A_n, R_1, \cdots, R_n, P >$ where n refers to the number of multiple agents; A is the joint action space set of all agents, $A = A_1 \times \cdots \times A_n$;$R_i$is the reward function of each agent, $R_i : S \times A \times S \to R$; P is the state transfer function, $P : S \times A \times S \to [0, 1]$. We assume that the reward function is bounded.

In the multi-agent case, the state transition is the result of all the agents acting together, so the reward of the agents also depends on the joint strategy. Define the strategy H to be the joint strategy of the agents $H_i : S \times A \to H$ Accordingly the reward for each agent is rewarded as:

$$R_i^H = E[R_{t+1} \big| S_t = s, A_{t,i} = a, H] \tag{23}$$

its Bellman equation is:

$$v_i^H(s) = E_i^H[R_{t+1} + \gamma V_i^H(S_{t+1})\big|S_t = s] \tag{24}$$

$$Q_i^H(s,a) = E_i^H[R_{t+1} + \gamma Q_i^H(S_{t+1}, A_{t+1}\big|S_t = s, A_t = a)] \tag{25}$$

Based on the task type, multi-agent reinforcement learning can be classified as fully cooperative, fully competitive, and hybrid. In the fully cooperation stochastic game, the reward function is the same for all the agents, i.e., $R_1 = R_2 = \cdots = R_n$, so the rewards are also the same. The goal of multi-agent is maximize the common reward. If $n = 2, R_1 = -R_2$, then the two agents have opposite goals and the stochastic game is perfectly competitive. In addition, there exist strategies that are neither perfectly competitive nor perfectly cooperative, which are called hybrid strategies.

In a fully cooperative stochastic game, rewards can be maximized jointly. In other cases, the rewards of the agents are usually different and correlated, and they cannot be maximized independently. Therefore, specifying good and general MARL objectives is a challenge. Reviewing the definition of learning goals in the existing literature, they can be summarized as two main aspects: stability and adaptability.

Stability refers to the stability of the learning motivation of the agent and the fact that the strategy converges to fixed. Adaptability ensures that the performance of an agent does not decline as other agents change their strategies. Convergence to an equilibrium state is the basic requirement for stability, i.e., the strategies of all the agents converge to a coordinated equilibrium state, most often a Nash equilibrium. Adaptability is reflected in two criteria: rationality or no-regret. Rationality means that the agent converges to the optimal feedback when the other agents are stable; no-regret means that the reward of the strategy that eventually converges cannot be worse than the rewards of any other strategy. After determining the learning goals, we classified and reviewed the classical reinforcement learning algorithms according to different task types.

4.1 Full Cooperation

In a stochastic game of full cooperation, the agents have the same reward function, at which time the learning objective can be formulated as follows:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[r_{t+1} + \gamma max Q_{t+1}(s_{t+1}, a') - Q_t(s_t, a_t)] \tag{26}$$

As with single agent, agents employ greedy strategies to maximize the rewards:

$$h_i(x) = arg_{a_i} max_{a_i \cdot a_n} max Q^*(s,a) \tag{27}$$

However, the agents are non-independent in making decisions, even though they parallelly learn a common goal. So considering the collaboration between agents becomes necessary. The Team-Q algorithm [50] avoids the collaboration problem by assuming that the optimal joint action is unique. The Distributed-Q algorithm [44] solves collaborative tasks with limited computation without assuming coordination, and its computational complexity is similar to that of single-agent Q-learning. However, this algorithm is only applicable to deterministic problems with non-negative reward functions. The above algorithms have some limitations, namely, they all rely on accurate measurements of states, some of them also require accurate measurements of the effects of other agents, and they are also subject to the effects of other agents. and also suffer from dimensional catastrophe.

4.2 Full Competition

In a perfectly competitive stochastic game (for two agents, i.e., R1 = R2, the principle of minimize and maximize can be applied): the worst-case assumption of maximizing the reward

of one agent is the assumption that the adversary will always try to minimize its reward. minimax-Q algorithm uses the principle of minimal greatness to compute the strategy of the stage game and values, as well as a temporal difference rule similar to Q-learning. The following algorithm is given for agent 1:

$$h_{1,t}(s_t, \cdot) = argm_1(Q_t, x_t) \tag{28}$$

$$Q_{t+1}(s_t, a_{1,t}, a_{2,t}) = Q_t(s_t, a_{1,t}, a_{2,t}) + \alpha[r_{k+1} + \gamma m_1(Q_t, a_{t+1}) - Q_t(s_t, a_{1,t}, a_{2,t})] \tag{29}$$

where $m1$ is the minimal maximization reward of agent 1.

$$m1(Q, s) = \max_{h_1(s,\cdot)} \min_{a_2} \sum_{a_1} h_1(x, a_1) Q(s, a_1, a_2) \tag{30}$$

where the random strategy of agent 1 at state s is denoted by $h_{1(s,\cdot)}$ and the dots represent the action parameters. The optimization problem can be solved by linear programming.

4.3 Hybrid Tasks

In hybrid stochastic games, the reward functions of the agents are not constrained and this model is best suited for selfish agents. The concept of game-theoretic equilibrium is most often used in hybrid stochastic games, and a large number of algorithms in this category are only for static tasks. Single-agent algorithms like Q-learning can be directly applied to hybrid tasks. The updating of parameters requires the use of Q-tables of all the agents, so each agent has to replicate the Q-tables of the other agents, which requires that all the agents use the same algorithm and can measure all actions and rewards. Even with these assumptions, the equilibrium selection problem arises when the strategies derived by different agents are not unique. One common approach is NashQ-learning, in addition to Correlated Equilibrium Q-learning (CE-Q) [27] or Asymmetric Q-learning[42], which can solve the equilibrium problem by using correlation or Stackelberg (leading-following) equilibrium, respectively. For asymmetric Q-learning, the follower does not need to model the leader's Q table, but the leader must know how followers choose their actions. Table 3 briefly compares and summarizes the various algorithms.

Table 3: Comparison of classic MARL algorithm based on game theory.

Algorithm	Value function update	Action select
Q-learning	$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r + \gamma \max Q\left(s', a'\right)\right]$	ε - greedy strategy, or action selection based on Boltzmann machine distribution.
Minimax Q	$Q(s,a,o) \leftarrow (1-\alpha)Q(s,a,o) + \alpha\left[r + \gamma V\left(s'\right)\right]$ * $V\left(s'\right) = \max_{\pi} \min_{o} \sum aQ\left(s', a, o\right)\pi\left(s', a'\right)$	Action selection based on the currently learned strategy π.
Nash Q	$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r_i + \gamma NashQ_i\left(s'\right)\right]$	Nash equilibrium action selection based on the current game
CE-Q	$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r_i + \gamma CEQ_i\left(s'\right)\right]$	Relevant equilibrium action selection based on the current game

* o represents the action taken by the opponents of the agents playing against each other.

5 Application of Reinforcement Learning

5.1 Application in Game Confrontation

Games are the perfect test bed for artificial intelligence algorithms, from which many representative algorithms have emerged. In many previous video games, reinforcement learning

algorithms have achieved good results and even surpassed human players in some games, such as DQN and its various variants in Atari 2600 [57, 36]. Of course, the most famous of these is Silver et al.'s proposal for the zero-sum and information-complete turn-based chess game programs AlphaGo, AlphaGo Zero and Alpha Zero [75, 78, 76].The "Alpha series" uses the Monte-Carlo Tree Search (MCTS) [13] infrastructure, combining the value network, policy network, and fast rollout modules to form a complete system. Reinforcement learning extends the depth and width of tree search, balances exploration and exploitation, and achieves significant results through self-play of the agents. The "Alpha series" of programs defeated the then human world Go champion, and extended this advantage to Chinese chess and Japanese Shogi.

Also, reinforcement learning algorithms have been applied to multiplayer games, such as in Texas Hold'em, a multiplayer game with non-complete information and involving psychology, the recursive reasoning of Counter Factual Regret minimization (CFR) [59, 62] was used to deal with information asymmetry and achieve generalized Nash equilibrium, and defeated five top human players for the first time in a six-player Texas Hold'em game. In addition, in multiplayer video games where the maps are not fully disclosed, OpenAI Five defeated human masters in the highly complex, locally observed, and highly cooperative 5v5 Dota2 game [1], and the program designed by Pang et al. performed well in the StarCraft II game [65].

5.2 Application in Robotics

Robotics is the most classical and promising application of reinforcement learning [94]. The MDP sequential decision-making feature at the core of reinforcement learning offers the possibility of complex engineering designs for robots, such as robotic arm movements [19, 21, 98, 60], helicopter and drone manipulation [63], and automatic robot navigation [55, 7]. In the application of robot playing table tennis [60], the robot observes the position and speed change of the table tennis ball, as well as the position and speed of the arm joints and other state information, and adjusts the arm swinging strategy and action until it learns to knock back the table tennis balls flying in different directions. In a series of work [19, 21, 98] based on the MAML algorithm [14] at BAIR, the robot was trained on a large number of meta-tasks by watching a human action demonstration and a video demonstration, respectively, and gradually learned the meta-learning strategy based on the demonstration. Subsequently, the robot was then able to quickly perform actions such as grasping and categorizing objects when faced with new tasks that it had not seen before. In addition, some studies have begun to explore the problem of human-robot collaboration in real production lines [51, 52].

In the practical application, due to the difficulty of sample acquisition, the high dimensionality of agent's state space, and the difficulty of capturing the characteristics of dynamic systems in the model, no real industrial application has been realized yet [40].

5.3 Application in Urban Transportation

In modern urban traffic, the number of motor vehicles is increasing. Some roads are heavily congested, and pedestrians and non-motorized vehicles have a high degree of randomness. The road conditions are very complex and pose a great challenge to the smooth traffic and the safety of participants. As a result, urban transportation networks and motorists are turning their attention to the field of artificial intelligence technologies for the development of urban smart transportation and automated or assisted driving technologies [8]. Among them, reinforcement learning algorithms are gaining more and more attention and applications as their core MDP process is highly compatible with the needs of urban transportation network deployment. Some recent works have studied the unified regulation of traffic signals [88, 96, 17] in actual urban traffic , and design problems of urban roads [10] to study how to improve real urban traffic. Meanwhile, autonomous or assisted driving technologies for motor vehicles are of great interest to major automobile manufacturers and technology companies [2]. Among them, the assisted or autonomous driving control system, as an agent in the MDP process, fully senses the surrounding road conditions by observing the movement and distribution of motor vehicles,

traffic signals, and surrounding vehicles, pedestrians and non-motorized vehicles. According to the observed environment state, the system is controlled by a reinforcement learning method based on value functions or strategies. The control system issues a series of instructions to assist human drivers to achieve intelligent navigation, route planning, avoiding pedestrians, non-motorized vehicles and emergency avoidance operations to ensure the safety of the participants and the smooth flow of roads [12]. In the follow-up work, researchers had further work for dense vehicles in urban traffic [24] and few extreme road conditions [64] and the self-driving car simulations were carried out.

With the development of online taxi economy, more and more people choose to travel by online taxi. In order to improve the service effectiveness, reinforcement learning has been heavily applied to the online taxi dispatching business. A lot of research work and application practices have been carried out by the corporate research institutes represented by the AI Lab of DDT [46, 82, 95]. Among them In this study, the distance between passengers and potential drivers, road congestion and driver's service rating are used as environmental factors. The dispatching system continuously optimizes the strategy for dispatching orders to match passengers with the most suitable drivers, minimize passenger waiting time, and reduce the waiting time of empty drivers, so as to obtain the maximum maximize revenue.

5.4 Application in Business

In recent years, search engines, digital media, and e-commerce have gradually entered people's daily life and changed people's way of life. Reinforcement learning is an effective method for modeling and maximizing the cumulative benefits of user-system interaction. modeling and maximizing the cumulative benefit, it has widely application potential and many successful cases [100] in information retrieval, product recommendation, advertising push, and other scenarios.

Relevance ranking is the key to information retrieval applications, and Learning-to-Rank (LTR) is one of the core techniques [97]. In an information retrieval system, the search engine (agent) makes an action at each request from the user (environment), and the user gives feedback to the search engine based on the results given by the search engine in terms of clicks, page turns, etc. Accordingly, the search engine will make a new ranking decision when it receives a new request. This decision process continues until the user purchases the product or exits the search [38, 89]. The core of a recommendation system is to recommend as accurately as possible the product or information that best matches the user's preferences based on the user's historical behavior [99]. In the MDP setting, the user's preference is the state of the environment, and the transfer function describes the dynamic change property of the user's preference over time. Each time the system recommends a product or information to the user, the user gives a corresponding feedback, such as skip, click to browse or buy, which reflects the user's satisfaction with the recommended product. Based on the user's historical behavior, the system adjusts the determination of the user's preference, i.e., the environmental state changes, and makes the next recommendation [101]. The goal of the recommendation system is to recommend the most appropriate product or information to the user to maximize the user's click-through rate and time spent [102]. The goal of online advertising is to push the right ads to the right users, where reinforcement learning provides the publisher with cooperative strategies [92] and bidding strategies [15] to maximize the revenue, Click Through Rate (CTR), or Rate Of Investment (ROI) of the ad campaign.

6 Conclusion and Outlook

Reinforcement learning as an end-to-end learning process, based on MDP to make sequential decisions and train optimal policies, has attracted a lot of attention from academia and enterprises and is considered as a key step to achieve general-purpose AI. This paper reviews the progress and development of reinforcement learning algorithms and applications, focus-

ing on representative reinforcement learning methods based on value functions, policy-based search, combined value and search, and cutting-edge research on multi-agent reinforcement learning and meta-reinforcement learning, all of which have contributed to the development of reinforcement learning in a more generalized and convenient direction. Finally, this paper provides an overview of successful applications of reinforcement learning in games, robotics, urban transportation, and business, demonstrating the benefits and potential of reinforcement learning for intelligent decision features.

Although reinforcement learning has achieved some success in research and application areas, it is still essentially limited to ideal, highly structured experimental data in a simulated environment, and does not yet have human-like autonomous learning, reasoning, and decision-making capabilities. In order to move further towards the goal of general artificial intelligence, there are several directions for reinforcement learning research and applications:

(1) Improve robustness of reinforcement learning by means of supervised learning. Policy gradient-based reinforcement learning algorithms are the mainstream of existing research. However, they inevitably have the disadvantage of large variance, which affects the stability of the algorithm. In this regard, more efficient and stable supervised learning methods, such as imitation learning and behavioral cloning, can be combined to make full use of expert experience to quickly train better strategies.

(2) Build smarter representations and problem formulation of reinforcement learning. By focusing on the mathematical nature of the algorithm, we design interpretable and simple reinforcement learning strategies and abandon the simple "tuning" approach to expand the applicability of the algorithm from the root, reduce the complexity of the algorithm, and break through the core problems of exploration and application, sparse returns and sample efficiency in reinforcement learning.

(3) Add memory modules. Use contextual information to enhance the autonomous learning capability of reinforcement learning. By integrating different types of memory modules, such as LSTM and GRU, into the reinforcement learning model, additional reward and previous action and state information are introduced to enable the agent to learn more task-level information and thus to acquire more autonomous learning, reasoning and decision-making functions.

(4) Extending meta-learning and transfer learning to multi-agent strong chemistry learning research and application areas. For multi-agent systems that are prevalent in real task scenarios, such as production line robots, urban road vehicles, etc., avoid the high cost and uncertainty of training a large number of agents from scratch. The idea of meta-learning and transfer learning is adopted to train models that can be quickly adapted to new tasks using a prior knowledge. This will alleviate the need for strong computational support for MARL, and take it a step further towards the application of complex scenarios.

(5) Develop reinforcement learning algorithms for physical inputs to cope with real industrial production applications. In real production and life, agent decent to high-dimensional environment such as real objects, video images and other physical information, rather than the original pixel-level information. In this process, using unsupervised learning or other machine learning techniques to understand and extract features from physical objects and inter-physical relationships will greatly improve the efficiency of reinforcement learning algorithms and facilitate their application in real-life scenarios.

References

[1] Openai five 2016–2019. `https://openai.com/projects/five/`.

[2] Tesla vehicle deliveries and autopilot mileage statistics. `https://lexfridman.com/tesla-autopilot-miles-and-vehicles/`.

[3] Benchmarks for spinning up implementations, 2018. `https://spinningup.openai.com/en/latest/spinningup/bench.html#hopper-pytorch-versions`.

[4] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[5] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[6] David Balduzzi and Muhammad Ghifary. Compatible value gradients for reinforcement learning of continuous deep policies. *arXiv preprint arXiv:1509.03005*, 2015.

[7] Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J Chadwick, Thomas Degris, Joseph Modayil, et al. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433, 2018.

[8] Ana LC Bazzan and Franziska Klügl. Introduction to intelligent systems in traffic and transportation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3):1–137, 2013.

[9] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

[10] Francois Belletti, Daniel Haziza, Gabriel Gomes, and Alexandre M Bayen. Expert level control of ramp metering based on multi-task deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 19(4):1198–1207, 2017.

[11] R Bellman. Dynamic programming, princeton, nj: Princeton univ. *versity Press. BellmanDynamic Programming1957*, 1957.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[13] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[14] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[15] Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, and Defeng Guo. Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 661–670, 2017.

[16] M. A. Cheng-Qian, W. Xie, and W. J. Sun. Research on reinforcement learning technology: A review. *Command Control Simulation*, 2018.

[17] Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1086–1095, 2019.

[18] Peter Dayan and CJCH Watkins. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[19] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and P. Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *ArXiv*, abs/1611.02779, 2016.

[20] Chelsea Finn, P. Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

[21] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pages 357–368. PMLR, 2017.

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

[23] Vincent Francois, Raphael Fonteneau, and Damien Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies. 12 2015.

[24] Lex Fridman, Jack Terwilliger, and Benedikt Jenik. Deeptraffic: Crowdsourced hyperparameter tuning of deep reinforcement learning systems for multi-agent dense traffic navigation. *arXiv preprint arXiv:1801.02805*, 2018.

[25] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[26] Faustino Gomez and Jürgen Schmidhuber. Evolving modular fast-weight networks for control. In *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005: 15th International Conference, Warsaw, Poland, September 11-15, 2005. Proceedings, Part II 15*, pages 383–389. Springer, 2005.

[27] Amy Greenwald and Keith B. Hall. Correlated q-learning. In *International Conference on Machine Learning*, 2003.

[28] Abhishek Gupta, Russell Mendonca, Yuxuan Liu, P. Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Neural Information Processing Systems*, 2018.

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

[30] Roland Hafner and Martin A. Riedmiller. Reinforcement learning in feedback control. *Machine Learning*, 84:137–169, 2011.

[31] H. V. Hasselt. Double q-learning. In *NIPS*, 2010.

[32] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[33] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[34] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

[35] Nicolas Manfred Otto Heess, TB Dhruva, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyun Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *ArXiv*, abs/1707.02286, 2017.

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

[37] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[38] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 368–377, 2018.

[39] Sham M. Kakade. A natural policy gradient. In *NIPS*, 2001.

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

[41] Vijay R Konda and John N Tsitsiklis. Onactor-critic algorithms. *SIAM journal on Control and Optimization*, 42(4):1143–1166, 2003.

[42] V. Kononen. Asymmetric multiagent reinforcement learning. In *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003.*, pages 336–342, 2003.

[43] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068, 2013.

[44] Martin Lauer. An algorithm for distributed reinforcement learning in cooperative multiagent systems. In *Proc. 17th International Conf. on Machine Learning*, 2000.

[45] Sergey Levine, Chelsea Finn, Trevor Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17:39:1–39:40, 2015.

[46] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The world wide web conference*, pages 983–994, 2019.

[47] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[48] TP Lillicrap, JJ Hunt, A Pritzel, N Heess, T Erez, Y Tassa, D Silver, and D Wierstra. Continuous control with deep reinforcement learning, 4th int. In *Conf. Learn. Represent. ICLR*, 2016.

[49] Longxin Lin. Reinforcement learning for robots using neural networks. 1992.

[50] Michael L Littman. Value-function reinforcement learning in markov games. *Cognitive systems research*, 2(1):55–66, 2001.

[51] Changliu Liu and Masayoshi Tomizuka. Algorithmic safety measures for intelligent industrial co-robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3095–3102. IEEE, 2016.

[52] Changliu Liu and Masayoshi Tomizuka. Designing the robot behavior for safe human–robot interactions. *Trends in Control and Decision-Making for Human–Robot Collaboration Systems*, pages 241–270, 2017.

[53] Quan Liu, J Wei Zhai, Zong-Zhang Zhang, Shan Zhong, Qian Zhou, Peng Zhang, and Jin Xu. A survey on deep reinforcement learning. *Chinese Journal of Computers*, 41(1):1–27, 2018.

[54] Russell Mendonca, Abhishek Gupta, Rosen Kralev, P. Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. In *Neural Information Processing Systems*, 2019.

[55] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

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

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

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

[59] Matej Moravčík, Martin Schmid, Neil Burch, and Viliam Lisy. , dustin morrill, nolan bard, trevor davis, kevin waugh, michael johanson, and michael bowling. deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[60] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

[61] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[62] Todd W Neller and Steven Hnath. Approximating optimal dudo play with fixed-strategy iteration counterfactual regret minimization. In *Advances in Computer Games: 13th International Conference, ACG 2011, Tilburg, The Netherlands, November 20-22, 2011, Revised Selected Papers 13*, pages 170–183. Springer, 2012.

[63] AY Ng, HJ Kim, MI Jordan, and S Sastry. Advances in neural information processing systems, chapter autonomous helicopter flight via reinforcement learning, 16, 2004.

[64] Matthew O'Kelly, Aman Sinha, Hongseok Namkoong, Russ Tedrake, and John C Duchi. Scalable end-to-end autonomous vehicle testing via rare-event simulation. *Advances in neural information processing systems*, 31, 2018.

[65] Zhen-Jia Pang, Ruo-Ze Liu, Zhou-Yu Meng, Yi Zhang, Yang Yu, and Tong Lu. On reinforcement learning for full-length game of starcraft. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4691–4698, 2019.

[66] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[67] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[68] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[69] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in neural information processing systems*, 28, 2015.

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

[71] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, page 1889–1897. JMLR.org, 2015.

[72] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

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

[74] D Silver. Deep reinforcement learning, a tutorial at icml 2016, 2016.

[75] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[76] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[77] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[78] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[79] A. Srinivas, Sahil Sharma, and Balaraman Ravindran. Dynamic frame skip deep q network. *ArXiv*, abs/1605.05365, 2016.

[80] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[81] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. page 1057–1063. MIT Press, 1999.

[82] Xiaocheng Tang, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. A deep value-network based approach for multi-driver order dispatching. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1780–1790, 2019.

[83] Hado Van Hasselt, Arthur Guez, Matteo Hessel, and David Silver. Learning functions across many orders of magnitudes. 02 2016.

[84] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[85] Haoran Wang, Thaleia Zariphopoulou, and Xunyu Zhou. Exploration versus exploitation in reinforcement learning: a stochastic control approach. *arXiv preprint arXiv:1812.01552*, 2018.

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

[87] Chris Watkins. Learning from delayed rewards. 1989.

[88] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2496–2505, 2018.

[89] Zeng Wei, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. Reinforcement learning to rank with markov decision process. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 945–948, 2017.

[90] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992.

[91] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.

[92] Di Wu, Cheng Chen, Xun Yang, Xiujun Chen, Qing Tan, Jian Xu, and Kun Gai. A multi-agent reinforcement learning method for impression allocation in online display advertising. *ArXiv*, abs/1809.03152, 2018.

[93] Ke Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, 2015.

[94] Kelvin Xu, Ellis Ratner, Anca D. Dragan, Sergey Levine, and Chelsea Finn. Learning a prior over intent via meta-inverse reinforcement learning. In *International Conference on Machine Learning*, 2018.

[95] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 905–913, 2018.

[96] Zhang L Yang WC and Zhu F. Multi-agent reinforcement learning based traffic signal control for integrated urban network: survey of state of art. *Application Research of Computers*, 35(6):1613–1618, 2018.

[97] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, et al. Ranking relevance in yahoo search. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 323–332, 2016.

[98] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.

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

[100] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. " deep reinforcement learning for search, recommendation, and online advertising: a survey" by xiangyu zhao, long xia, jiliang tang, and dawei yin with martin vesely as coordinator. *ACM Sigweb Newsletter*, (Spring):1–15, 2019.

[101] Xiangyu Zhao, L. Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. Recommendations with negative feedback via pairwise deep reinforcement learning. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[102] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Jessie Li. Drn: A deep reinforcement learning framework for news recommendation. *Proceedings of the 2018 World Wide Web Conference*, 2018.