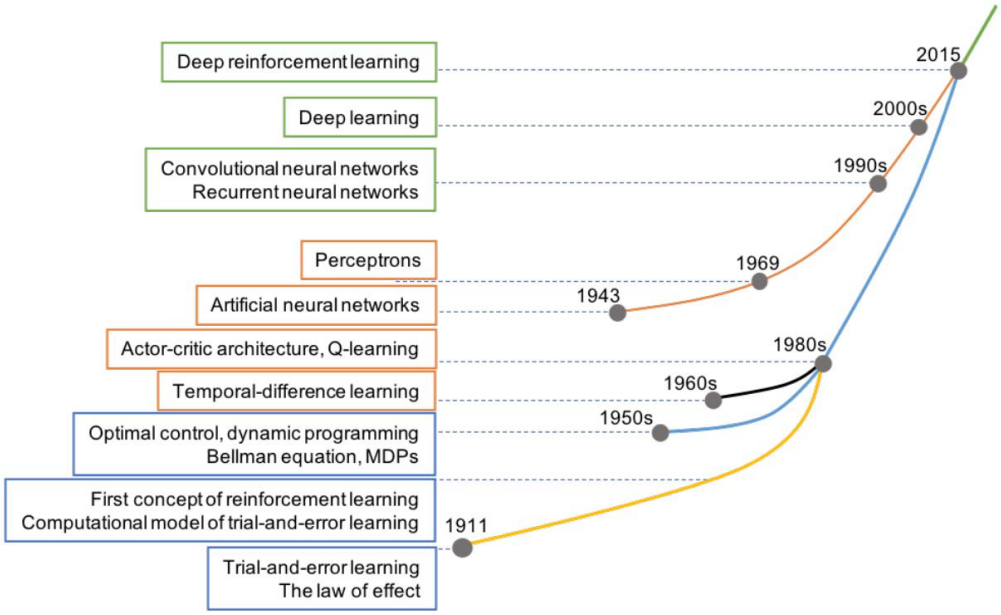
Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications

Introduction：

Multi-agent problems require multiple agents to communicate and cooperate to solve complex tasks.

Including multi-agent training schemes 多智能体训练方式、multi-agent transfer learning 多智能体迁移学习、trial and error (TE) procedure & the mechanism of temporal-difference (TD) learning 时序差分学习.

The picture below brought the theory of optimal control including Bellman equation and Markov decision process together with temporal-difference learning to form a well-known Q-learning.



SL(supervised learning) is learning from data that define input and corresponding output (often called “labeled” data) by an external supervisor, whereas RL is learning by interacting with the unknown environment. **What are the differences between SL & RL.**

RL is not an unsupervised learning (UL) method. UL is learning to explore the hidden structure of data where output information is unknown (“unlabelled” data). In contrast, RL is a goal-directed learning, i.e., it constructs a learning model that clearly specifies output to maximize the long-term profit. **What are the differences between UL & RL.**

使用深度学习的近似器处理高维数据.

Examine the stability and adaptation aspects of agents.

Review methods for knowledge reuse autonomy in multi-agent RL (MARL).

RL：

RL is a TE learning 1) by interacting directly with the environment 2) to self-teach over time

and 3) eventually achieve designating goal.

The interactions between agent and environment are described via three essential elements: state s, action a, and reward r

In this case, a series of states, actions, and rewards from initial state to terminal state is called an episode.

The agent's decision by defining a concept of policy. A policy is deterministic if the probability of choosing an action a from s: p(a|s) = 1 for all state s. In contrast, the policy is stochastic, if there exists a state s so that p(a|s) < 1.

Any RL problem satisfies this "memoryless" condition is known as Markov decision process (MDP). Therefore, the dynamics (model) of an RL problem is completely specified by giving all transition probabilities p(ai|s)

$$\boldsymbol{R_1}: \pi \mapsto \pi_d = \Psi_d(s) = \begin{cases} 1, & a_i = a_j \wedge j = \arg\max\limits_k \pi(s, a_k) \\ 0, & \forall a_i \in \Delta_\pi \wedge a_i \neq a_j \end{cases},$$

In this respect, we call that policy πt+1 is better than policy πt and denoted as πt+1 > πt. Therefore, we have a series of policies improved over time as follows:

π0 < π1 < ... < πt < πt+1 < ... < π* .

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-t-1} r_T = \sum_{i=0}^{T-t-1} \gamma^i r_{t+i+1},$$

where γ is a discounted factor so that 0 ≤ γ < 1. The agent becomes far-sighted when γ approaches to 1 and vice versa the agent becomes short-sighted when γ is close to 0

The next step is to define a value function that is used to evaluate how "good" of a certain state s or a certain state-action pair (s, a).

We can use value functions to compare how "good" between two policies π and π0 using the following rule：

$$\pi \leq \pi' \Longleftrightarrow \Big[V_\pi(s) \leq V_{\pi'}(s)\ \forall s\Big] \vee \Big[Q_\pi(s,a) \leq Q_{\pi'}(s,a)\ \forall (s,a)\Big].$$

$$V_\pi(s) = \sum_a \pi(s,a) \sum_{s'} p(s'|s,a)\Big(\mathbb{W}_{s\to s'|a} + \gamma V_\pi(s')\Big), \tag{5}$$

$$Q_\pi(s,a) = \sum_{s'} p(s'|s,a)\Big(\mathbb{W}_{s\to s'|a} + \gamma V_\pi(s')\Big), \tag{6}$$

where Ws→s0|a = E[rt+1|st = s, at = a, st+1 = s0] ≈ rt+1, Equations (5) and (6) are called Bellman equations and widely used in policy improvement.

Instead of repeating policy improvement process, we can estimate directly the value function of optimal policy π* using the following optimality Bellman equation：

$$V_{\pi^*}(s) = \max_a \sum_{s'} p(s'|s,a)\Big(\mathbb{W}_{s\to s'|a} + \gamma V_{\pi^*}(s') \Big), \tag{8}$$

$$Q_{\pi^*}(s,a) = \sum_{s'} p(s'|s,a)\Big(\mathbb{W}_{s\to s'|a} + \gamma \max_{a'} Q_{\pi^*}(s',a') \Big). \tag{9}$$

we can derive an optimal deterministic policy π* using：

$$\boldsymbol{R_2} : \pi \mapsto \pi' = \Psi'(s) = \begin{cases} 1, & a_i = a_j \wedge j = \arg\max\limits_k Q_\pi(s,a_k) \\ 0, & \forall a_i \in \Delta_\pi \wedge a_i \neq a_j \end{cases}.$$

Although we can use dynamic programming to approximate the solutions of Bellman equations, DP 要用完整的信息. We will review two model-free RL methods (require no knowledge of transition probabilities p(ai|s)) to approximate the value functions.

RL Methods：

In practice, MC and TD learning often use table memory structure (tabular method) to save value function of each state or each state-action pair.

1. Monte-Carlo

Monte-Carlo (MC) method estimates value function by repeatedly generating episodes and recording average return at each state or each state-action pair.

$$V_\pi^{MC}(s) = \lim_{i\to+\infty} \mathbb{E}\Big[r^i(s_t) \;\Big|\; s_t = s, \pi \Big],$$

where $r^i(s_t)$ denotes observed return at state s_t in episode ith. Similarly, we have value function of state-action pair:

$$Q_\pi^{MC}(s,a) = \lim_{i\to+\infty} \mathbb{E}\Big[r^i(s_t,a_t) \;\Big|\; s_t = s, a_t = a, \pi \Big].$$

this approach has made two essential assumptions to ensure the convergence happens:

1) the number of episodes is large
2) every state and every action must be visited with a significant number of times.

$$\boldsymbol{R_3} : \pi \mapsto \pi' = \Psi'(s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\Delta_\pi(s)|}, & a_i = a_j \wedge j = \arg\max\limits_k Q_\pi(s,a_k) \\ \frac{\epsilon}{|\Delta_\pi(s)|}, & \forall a_i \in \Delta_\pi \wedge a_i \neq a_j \end{cases}, \tag{10}$$

use ε -greedy algorithm for **Convergence**，

Generally, MC algorithms are divided into two groups: on-policy and off-policy. In on-policy methods, we use policy π for both evaluation and exploration purpose. Therefore, the policy π must be stochastic or soft. In contrast, off-policy uses different policy π0 = π to generate the episodes and hence π can be deterministic. off-policy 随机性不太够，不太稳定但是很简单。on-policy method is more stable when working with continuous state-space problems and when using together with a function approximator (such as neural networks)

2. Temporal-Difference learning

It makes an update on every step within the episode by leveraging 1-step Bellman equation.

TD learning is also divided into two categories: on-policy TD control (Sarsa) and off-policy TD control (Q-learning).

Sarsa:

$$\boldsymbol{U_2}: Q^i(s_t, a_t) \longleftarrow \alpha Q^{i-1}(s_t, a_t) + (1-\alpha)\left(r_{t+1} + \gamma Q^{i-1}(s_{t+1}, a_{t+1})\right).$$

Q-leaning:

$$\boldsymbol{U_3}: Q^i(s_t, a_t) \longleftarrow \alpha Q^{i-1}(s_t, a_t) + (1-\alpha)\left(r_{t+1} + \gamma \overbrace{\max_{a_{t+1}^j} Q^{i-1}(s_{t+1}, a_{t+1}^j)}^{\text{deterministic subpolicy}}\right).$$

TD learning 使用以前的价值估计函数 Vi-1 以更新 Vi, 被称为 bootstrapping method.

3. AC: Actor-Critic

AC can be on-policy or off-policy depending on the implementation details. Specifically, AC includes two separate memory structures for an agent: actor and critic. Actor structure 产生动作扔到 critic 去估计判断. Critic structure uses the following TD error to decide future tendency of the selected action:

$$\delta(a_t) = \beta\left(r_{t+1} + \gamma V(s_{t+1})\right) - (1-\beta)V(s_t),$$

where $0 < \beta < 1$; and if $\delta(a_t) > 0$, the tendency to select the action a_t in the future is high and vice versa.

Table 1: Characteristics of RL methods

Category	Pros	Cons
Model-free	• Dynamics of environment is unknown • Deal with larger state-space environments	○ Requires "exploration" condition
On-policy	• Stable when using with function approximator • Suitable with continuous state-space problems	○ Policy must be stochastic
Off-policy	• Simplify algorithm design • Can tackle with different kinds of problems • Policy can be deterministic	○ Unstable when using with function approximator
Bootstrapping	• Learn faster in most cases	○ Not as good as nonbootstrapping methods on mean square error

Table 2: Comparisons between RL methods

Category	Dynamic programming	On-policy MC	Off-policy MC	Sarsa	Q-learning	AC
Model-free		✓	✓	✓	✓	✓
On-policy		✓		✓		✓
Off-policy			✓		✓	✓
Bootstrapping	✓			✓	✓	✓

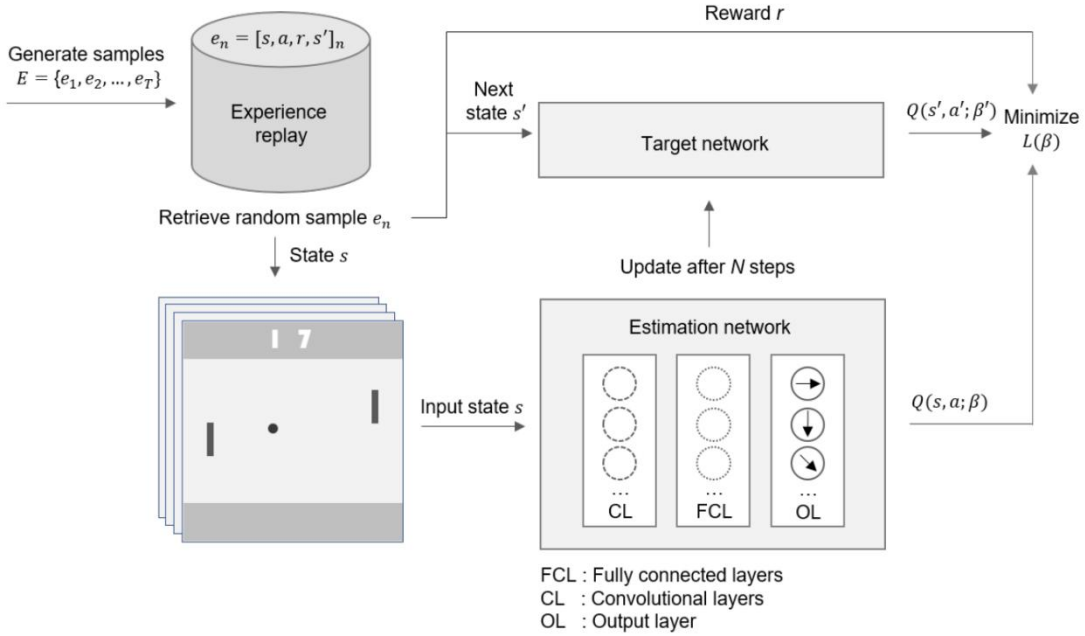
Deep RL：For single

1. DQN

The output of DQN produces Q-values of all possible actions a ∈ Δτ taken at state s,

$$\mathcal{L}(\beta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'|\beta) - Q(s, a|\beta)\right)^2\right].$$

为使样本不相关，创建一个 N 步更新的目标网络，同时样本从 experience replay memory 中间取来训练。



$$\begin{cases} \mathcal{L}_{\mathrm{DQN}}(\beta) = \mathbb{E}\left[(r + \gamma \max_{a'} Q(s', a'|\beta') - Q(s, a|\beta))^2\right] \\ \beta' \longleftarrow \beta \text{ for every N steps} \end{cases}$$

DQN 的缺点:

1. 有些状态在训练中会发现是冗余的（即对于奖赏没什么作用，而有些 policy 会采用某些冗余状态，但是这些状态是很少被采用的，因此 policy evaluation 并不能更新到收敛，因此对于评估工作是困难的。

2. 输入不够，因此状态依赖于过去信息和部分观测情况下，DQN 无法解决。

为什么 DQN 不使用原来的参数进行未来状态动作对估计会发散？

当 DQN 使用当前参数进行动作选择时，这些参数的意义在于动作选择倾向性，而使用当前的价值函数则会使得 DQN 打破了随机性，加大了倾向性而容易陷入局部最优，而发散。

2. DQN 的变种

DDQN:

DDQN 的想法在于将贪心的选择动作和动作评估分开，以消除对于动作价值函数过度估计的问题。

$$\mathcal{L}_{\mathrm{DDQN}}(\beta) = \mathbb{E}\left[\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a'|\beta)|\beta') - Q(s, a|\beta)\right)^2\right].$$

DDQN with PER(Prioritized experience replay)优先级经验回放: **为解决 DQN 的缺点 1**

特别的，我们更希望经常使用目标相关的样本进行探索策略。

prioritized experience replay that gives priority to a sample i based on its absolute value of TD error:

$$p_i = |\delta_i| = |r_i + \gamma \max_a Q(s_i, a|\beta') - Q(s_{i-1}, a_{i-1}|\beta)|$$

DRQN: **为解决 DQN 的缺点 2**

直观解决方法是使用 LSTM 结构代替连接最后一个卷积层的全连接层。这种 DQN 变种叫做 deep recurrent Q-network (DRQN)

DRQN 升级版：Deep attention recurrent Q-network (DARQN)

Deep RL：For Multi-Agent: MADRL

在多智能体学习领域下，每个智能体的价值函数也受共有动作、共有策略影响。

1. Challenges & Solutions

Non-stability 不稳定性

智能体互动和一起学习以重构环境导致了不稳定性。每一个智能体的单独策略影响整体最优策略，因此，无法单纯学习单个智能体的最优策略。Q-learning 在多智能体环境下无法保证收敛性，因为在不稳定环境中，马尔可夫性质无法保证。

在多智能体设定下，有针对探索和利用困境的 DQN 算法变种。学习何时独立决策何时选择合作，同时可以在不稳定环境中收敛。proposed two variants of DQN, namely deep repeated update Q-network (DRUQN) and deep loosely coupled Q-network (DLCQN), to deal with the non-stability problem in MAS(multi-agent system). **DRUQN??DLCQN??**

处理协作的多智能体物体运输的问题 LDQN: is compared with the hysteretic-DQN (HDQN)

WDDQN:Weighted double deep Q-network (WDDQN) in [120] to deal with non-stability in MAS.

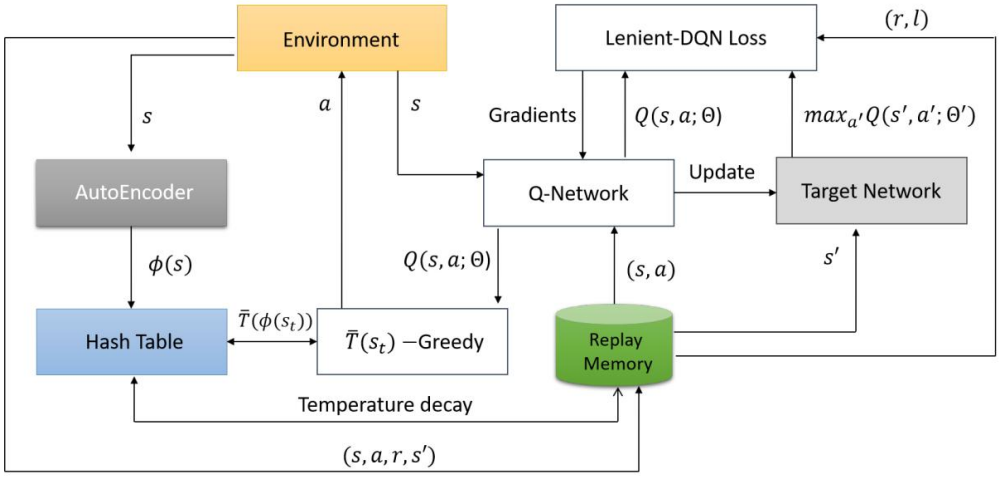


Fig. 7: Architecture of LDQN.

Partially-observed 局部可观测性

存在有智能体只观测到部分环境信息的情况下，在每一步做最好的决策问题，问题可以被抽象成 POMDP 问题。the partially observable Markov decision process (POMDP)

推荐的 DRQN 是基于 LSTM 的循环网络。在循环网络结构下，智能体可以更健全地在部分观察环境下学习改善后的策略。

由 DRQN 扩展的 DDRQN 是为了处理 POMDP 问题。有三个特点构成，上个动作输入，（LSTM 循环网络特点）；内部智能体权重分享，所有智能体共用一个网络结构，残缺经验回放？？简单排除了 DQN 经验回放的特点。

CL 方法集成了 3 中 DRL 方法，包括策略梯度，时序差分，和 AC，原则是学习简单的任务进行知识经验积累，以处理复杂问题。

深度策略推理 QN，是深度循环策略推理 QN 的升级版以处理部分可观测环境。a deep policy inference Q-network (DPIQN) to model multi-agent systems and its enhanced version deep recurrent policy inference Q-network (DRPIQN) to cope with partial observation.

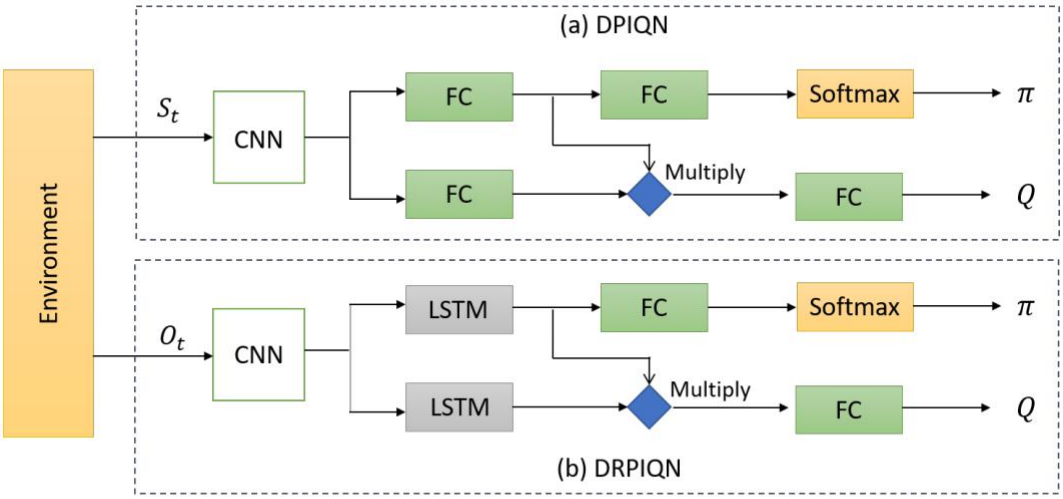


Fig. 8: Architecture of DPIQN and DRPIQN.

还需要处理与真实状态无关的噪声观测。

MAS-training 多智能体系统训练问题

另一种流行的方法是通过信道集中学习和分散执行策略，即独立基于局部观测进行动作选择，适用于部分可观测和通信受限。

集中学习每个智能体的策略已成为多智能体设定的标准范例，因为通信有时可能会受限，同时获得额外的状态信息。

集中策略使用共同的奖赏信号进行学习以便从共有的观测中获得共有的动作。而分散执行策略独立基于私人观测，但同时使用参数分享方式允许智能体之间充分使用其他智能体的经验。

参数分享的 TRPO 对于高维部分可观测数据和连续动作空间有很好的效果，以下是算法：

Algorithm 1 PS-TRPO

Initialize parameters of policy network Θ_0, and trust region size Δ
for $i \longleftarrow 0, 1, \ldots$ **do**
  Generate trajectories for all agents as $\tau \sim \pi_{\theta_i}$ using the policy with shared parameters.
  For each agent m, compute the advantage values $A_{\pi_{\theta_i}}(o^m, m, a^m)$ with m is the agent index.
  Search $\pi_{\theta_{i+1}}$ that maximizes $L(\theta) = E_{o\sim p_{\theta_k}, a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|o,m)}{\pi_{\theta_k}(a|o,m)} A_{\theta_k}(o,m,a)\right]$
  subject to $\bar{D}_{KL}(\pi_{\theta_i} \,\|\, \pi_{\theta_{i+1}}) \leq \Delta$ where D_{KL} is the KL divergence between distributions of two policies, and p_θ are the discounted frequencies of state visitation caused by π_θ.
end for

DIAL 方法通过信道将梯度从一个智能体传输给其他智能体，允许在智能体之间端到端后向传播。Developed communication neural net (CommNet) 允许智能体在完全协作任务中与他们的策略持续互相通信。

将分散执行和集中学习应用于分级主从结构中形成 MS-MARL 以解决多智能体系统的通信问题。

MADDPG 使用 DDPG 和集中学习、分散执行的技巧，critic 使用额外的信息来简化培训过程同时 actor 根据局部观测进行动作选择。Fig. 9 illustrates the multi-agent decentralized actor and centralized critic components of MADDPG where only actors are used during the execution phase.

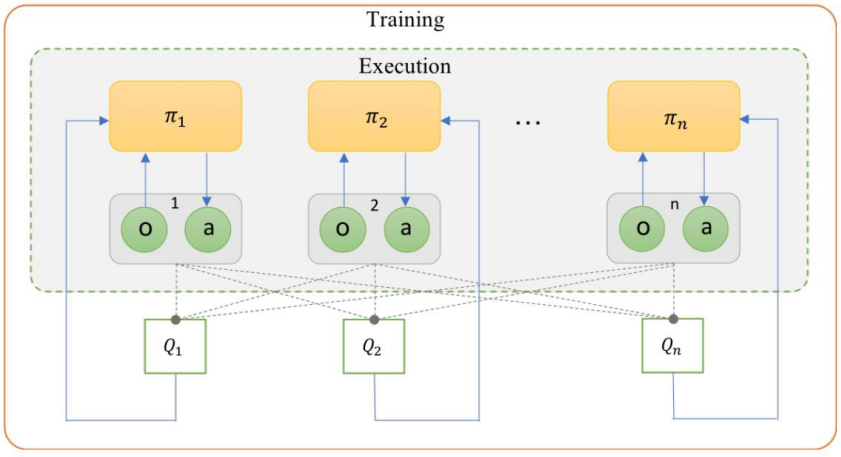


Fig. 9: Centralized learning and decentralized execution based MADDPG where policies of agents are learned by the centralized critic with augmented information from other agents' observations and actions.

Continuous Action Space 连续动作空间问题

trust region policy optimization (TRPO) 可以扩展到连续动作空间，以优化 robotic locomotion and image-based game playing.

Off-policy: DDPG 使用 AC 结构处理连续动作空间问题。

Extended DDPG to recurrent DPG (RDPG)以解决部分可观测且为连续动作空间的问题。

Transfer-learning for MADRL 多智能体深度强化学习的迁移学习问题

训练成本很高，而深度强化学习的知识迁移又是很重要的。

the actor-mimic method for multi-task and transfer learning 改善了 deep-policy-network 的训练速度。

Table 3: Multi-agent learning challenges and their solving methods

Challenges	Value-based	Actor-critic	Policy-based
Partial observability	DRQN [36]; DDRQN [24]; RIAL and DIAL [23]; Action-specific DRQN [121]; MT-MARL [85]; PS-DQN [30]; RL as a Rehearsal (RLaR) [55]	PS-DDPG and PS-A3C [30]; MADDPG-M [48]	DPIQN and DRPIQN [42]; PS-TRPO [30]; Bayesian action decoder (BAD) [26]
Non-stationarity	DRUQN and DLCQN [12]; Multi-agent concurrent DQN [18]; Recurrent DQN-based multi-agent importance sampling and fingerprints [25]; Hysteretic-DQN [85]; Lenient-DQN [86]; WDDQN [120]	MADDPG [68]; PS-A3C [30]	PS-TRPO [30]
Continuous action spaces		Recurrent DPG [39]; DDPG [66]	TRPO [101]; PS-TRPO [30]
Multi-agent training schemes	Multi-agent extension of DQN [111]; RIAL and DIAL [23]; CommNet [109]; DRON [38]; MS-MARL [53 54]; Linearly fuzzified joint Q-function for MAS [69]	MADDPG [68]; COMA [27]	
Transfer learning in MAS	Policy distillation [96]; Multi-task policy distillation [117]; Multi-agent DQN [20]	Progressive networks [97]	Actor-Mimic [87]

2. Applications

Table 4: A summary of typical MADRL applications in different fields

Applications	Basic DLR	Features	Limitations
Federated control [58]	Hierarchical-DQN (h-DQN) [57]	•Divide the control problem into disjoint subtasks and leverage *temporal abstractions*. •Use *meta-controller* to guide decentralized controllers. •Able to solve distributed scheduling problems such as multi-task dialogue, urban traffic control.	•Does not address the non-stationarity problem. •Number of agents is currently limited at six. •Meta-controller's optimal policy becomes complicated and inefficient when the number of agents increases.
Large-scale fleet management [67]	Actor-critic and DQN	•Reallocate vehicles ahead of time to balance the transport demands and supplies. •Geographic context and collaborative context are integrated to coordinate agents. •Two proposed algorithms, *contextual multi-agent actor-critic* and *contextual deep Q-learning*, can achieve explicit coordination among thousands of agents.	•Can only deal with discrete actions and each agent has a small (simplified) action space. •Assume that agents in the same region at the same time interval (i.e. same spatial-temporal state) are homogeneous.
Swarm systems [45]	DQN and DDPG	•Agents can only observe local environment (partial observability) but not the global state. •Use guided approach for multi-agent learning where actors make decisions based on locally sensed information whilst critic has central access to global state.	•Can only work with homogeneous agents. •Unable to converge to meaningful policies in huge dimensionality and partial observed problem.
Traffic lights control [11]	DDDQN and IDQN	•Learning multiple agents is performed using IDQN where each agent is modelled by DDDQN. •First approach to address heterogeneous multi-agent learning in urban traffic control. •Fingerprint technique is used to stabilize the experience replay memory to handle non-stationarity.	•The proposed deep RL approach learns ineffectively in high traffic conditions. •The fingerprint does not improve the performance of experience replay although the latter is required for efficient learning.
Task and resources allocation [83]	CommNet [109]	•Propose distributed task allocation where agents can request help from cooperating neighbors. •Three types of agents are defined: manager, participant and mediator. •Communication protocol is learned simultaneously with agents' policies through CommNet.	•May not be able to deal with heterogeneous agents. •Computational deficiencies regarding the decentralization and reallocation characteristics. •Experiments only on small state action spaces.
Energy sharing optimization [92]	DQN	•Each building is characterized by a DRL agent to learn appropriate actions independently. •Agents' actions include: consume and store excess energy, request neighbor or supply grid for additional energy, grant or deny requests. •Agents collaborate via shared or global rewards to achieve a common goal, i.e. zero-energy status.	•Agents' behaviors cannot be observed in an online fashion. •Limited number of houses, currently ten houses at maximum were experimented. •Energy price is not considered.
Keepaway soccer [59]	DQN	•Low-dimensional state space, described by only 13 variables. •Heterogeneous MAS, each agent has different experience replay memory and different network policy.	•Number of agents is limited, currently setting with 3 keepers vs. 2 takers. •Heterogeneous learning speed is significantly lower than homogeneous case.
Action markets [100]	DQN	•Agents can trade their atomic actions in exchange for environmental reward. •Reduce greedy behavior and thus negative effects of individual reward maximization. •The proposed approach significantly increases the overall reward compared to methods without action trading.	•Agents cannot find prices for actions by themselves because they are given at design time. •Strongly assume that agents cannot cheat on each other by making offers which they do not hold afterwards.
Sequential social dilemma (SSD) [62]	DQN	•Introduce an SSD model to extend MGSD to capture sequential structure of real-world social dilemmas. •Describe SSDs as general-sum Markov games with partial observations. •Multi-agent DQN is used to find equilibria of SSD problems.	•Assume agent's learning is independent and regard the others as part of the environment. •Agents do not recursively reason about one another's learning.
Common-pool resource (CPR) appropriation [90]	DQN	•Introduce a new CPR appropriation model using an MAS containing spatially and temporally environment dynamics. •Use the *descriptive agenda* [104] to describe the behaviors emerging when agents learn in the presence of other learning agents. •Simulate multiple independent agents with each learned by DQN.	•Single agent DQN is extended to multi-agent environment where the Markov assumption is no longer hold. •Agents do not do anything of *rational negotiation*, e.g. bargaining, building consensus, or making appeals.

Conclusion：

文章讲述多智能体问题的挑战和目前在深度强化学习领域的解决办法。

一方面，模仿学习尝试使用监督学习方法将状态映射到动作空间中。

逆强化学习需要从专家经验中推断出环境的一个奖赏函数。

These methods however have not yet been explored fully in multi-agent environments.

Model-free deep RL 已经在多智能体和单智能体领域解决了许许多多复杂的问题。

Many applications of MARL can now be solved effectively by MADRL based on its high-dimension handling capability. Therefore, there is a need of further empirical research to apply MADRL methods to effectively solve complex real-world problems such as the aforementioned applications.