Deep Reinforcement Learning for Multi-Agent Systems:

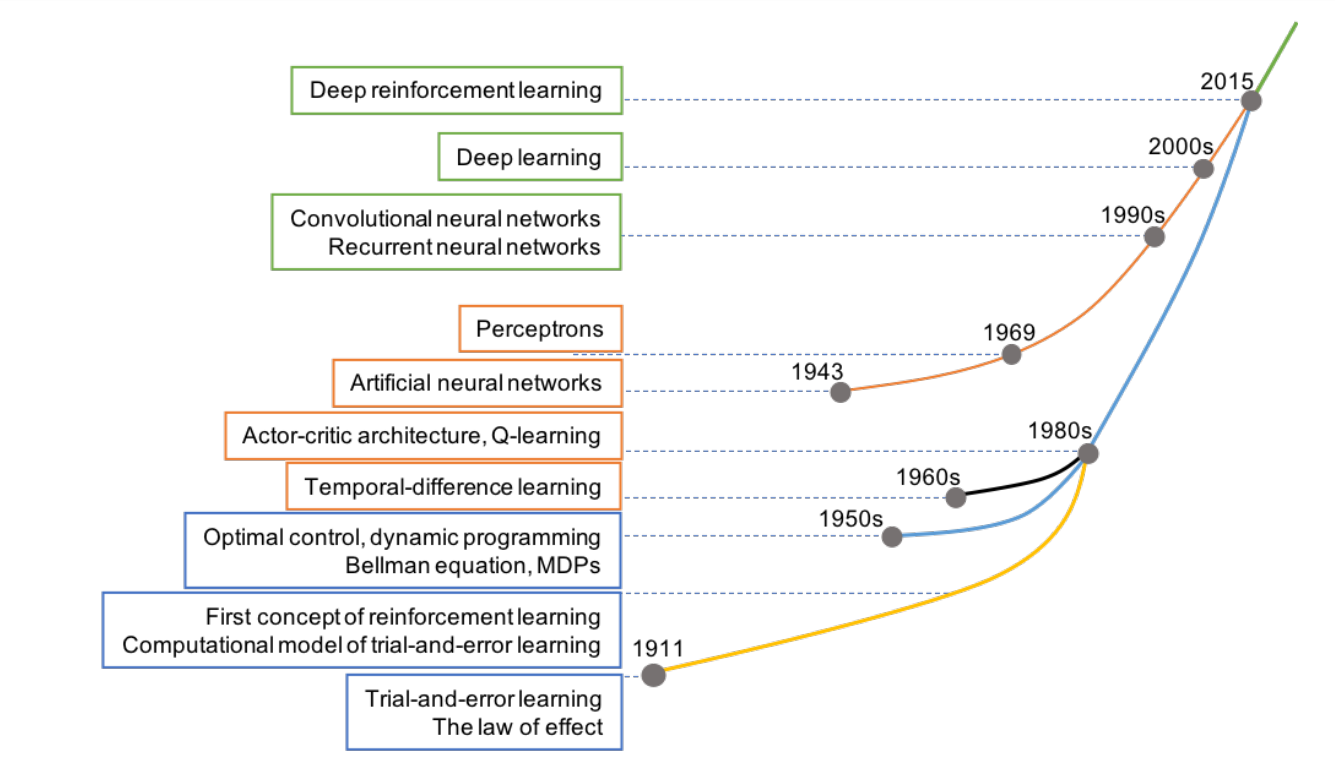
A Review of Challenges, Solutions and Applications

**Introduction：**

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

Including multi-agent training schemes、multi-agent transfer learning、trial and error (TE) procedure & the mechanism of temporal-difference (TD) learning.

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.**

Deep RL uses deep learning as an approximator to deal with high-dimensional data.

Examine the stability and adaptation aspects of agents.

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

**RL：**

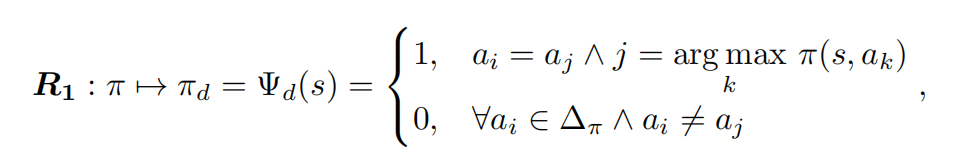
RL is a TE learning 1) by interacting directly with the environment 2) to self-teach over time and 3) eventually achieve designating goal.

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.

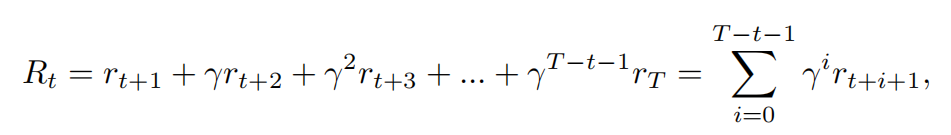
The agent’s decision by defining a concept of policy. A policy is deterministic if the probability of choosing an action a from s: p(a|s) = 1 for all state s. In contrast, the policy is stochastic, if there exists a state s so that p(a|s) < 1.

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)



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:

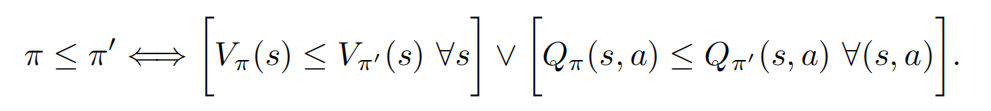
π0 < π1 < ... < πt < πt+1 < ... < π∗.

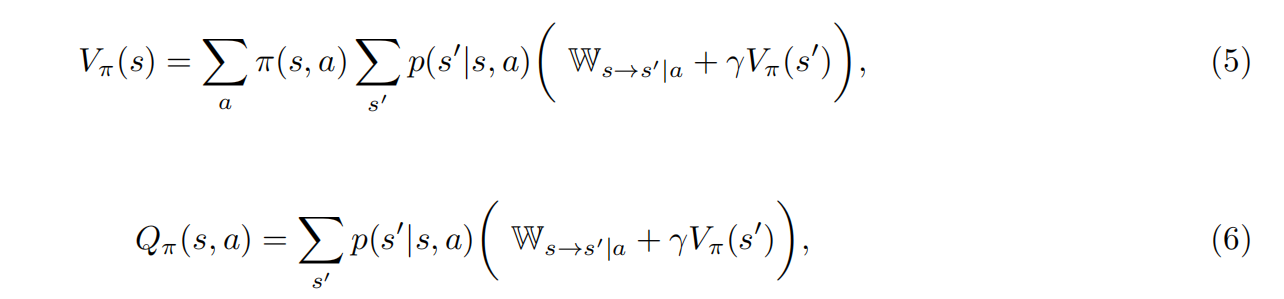


where γ is a discounted factor so that 0 ≤ γ < 1. The agent becomes far-sighted when γ approaches to 1 and vice versa the agent becomes short-sighted when γ is close to 0

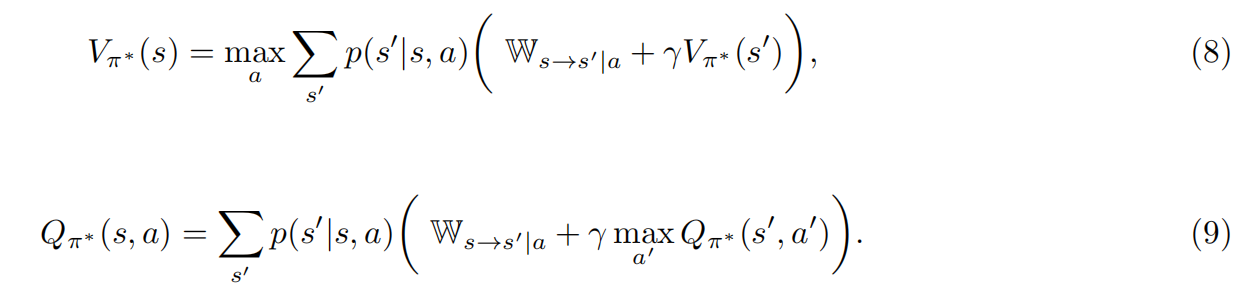
The next step is to define a value function that is used to evaluate how “good” of a certain state s or a certain state-action pair (s, a).

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

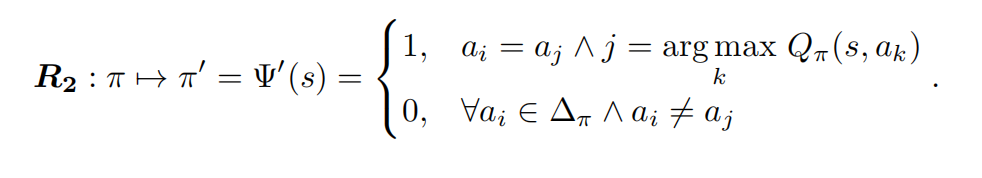


 where Ws→s0|a = E[rt+1|st = s, at = a, st+1 = s0] ≈ rt+1, Equations (5) and (6) are called Bellman equations and widely used in policy improvement.

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



we can derive an optimal deterministic policy π∗using：



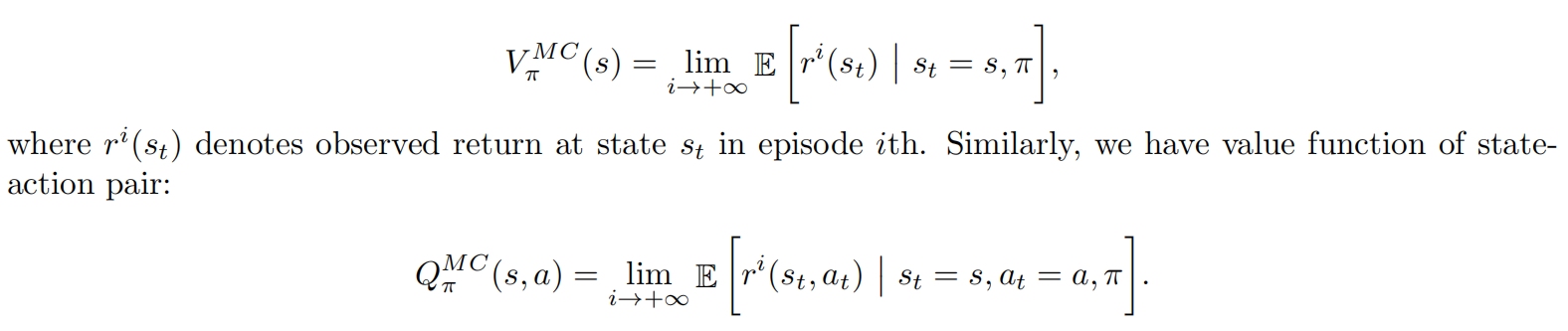
Although we can use dynamic programming to approximate the solutions of Bellman equations, it requires the complete dynamics information of the problem. We will review two model-free RL methods (require no knowledge of transition probabilities p(ai|s)) to approximate the value functions.

**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.

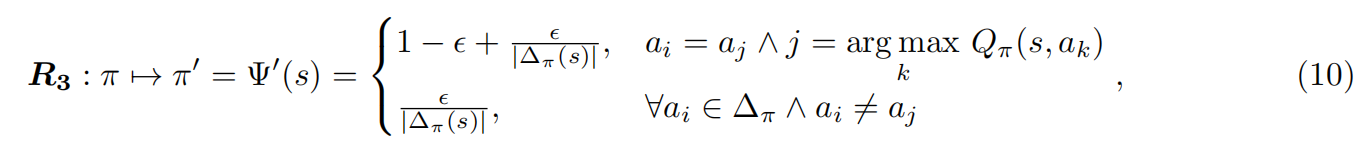
1. Monte-Carlo

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



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.



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

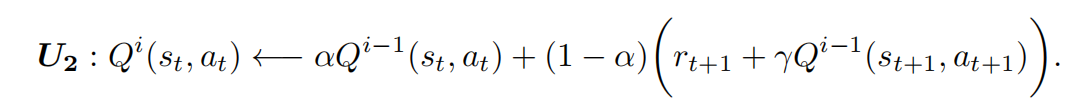
Generally, MC algorithms are divided into two groups: on-policy and off-policy. In on-policy methods, we use policy π for both evaluation and exploration purpose. Therefore, the policy π must be stochastic or soft. In contrast, off-policy uses different policy π0 = π to generate the episodes and hence π can be deterministic. Although off-policy is desirable due to its simplicity, 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)

1. Temporal-Difference learning

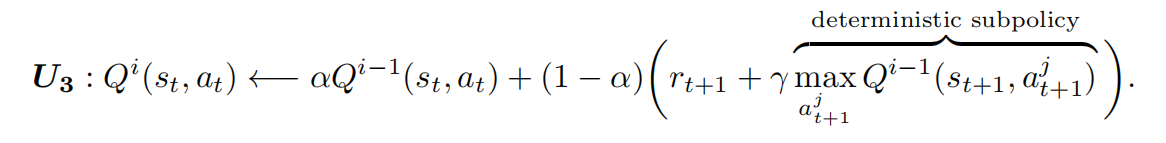
It makes an update on every step within the episode by leveraging 1-step Bellman equation.

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

Sarsa:



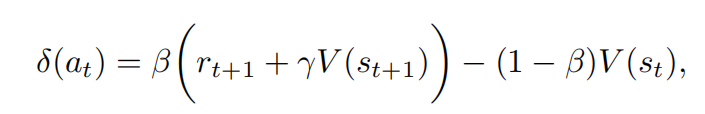
1. leaning:



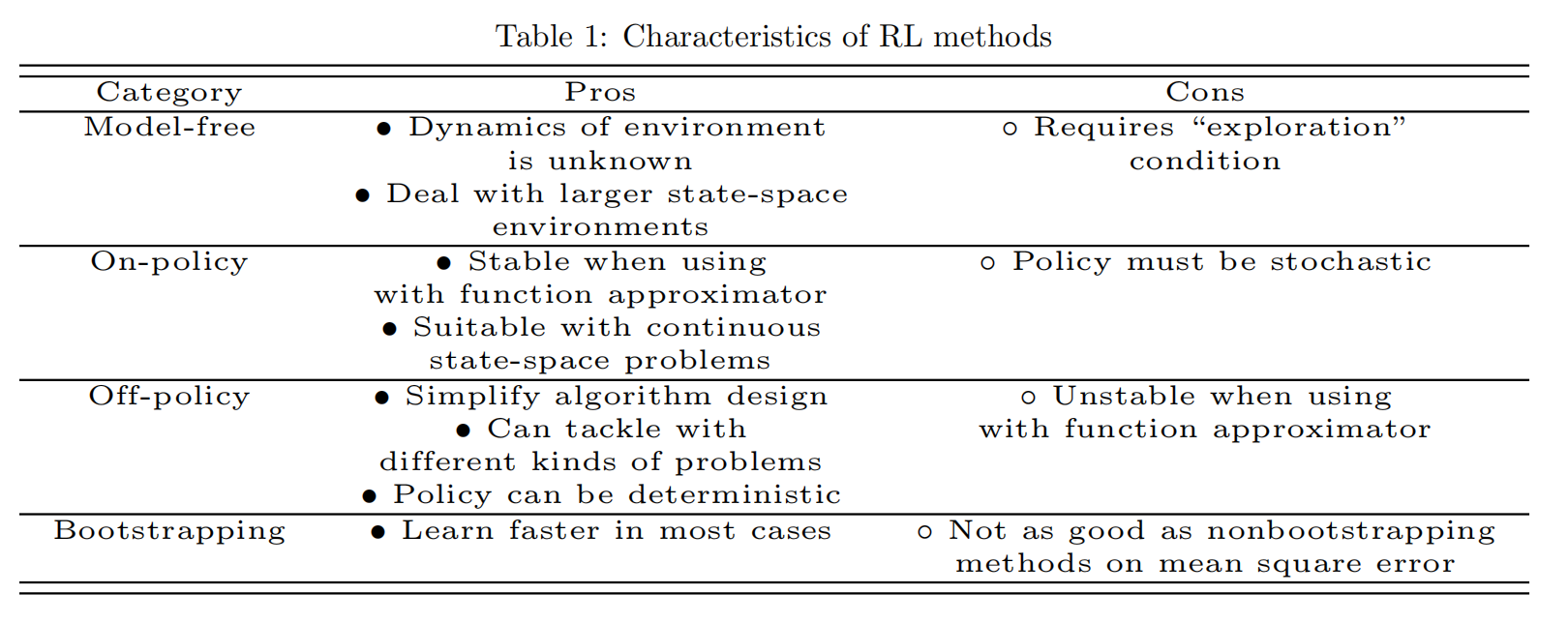
TD learning uses previous estimated values Vi-1to update the current ones Vi, which is known as bootstrapping method.

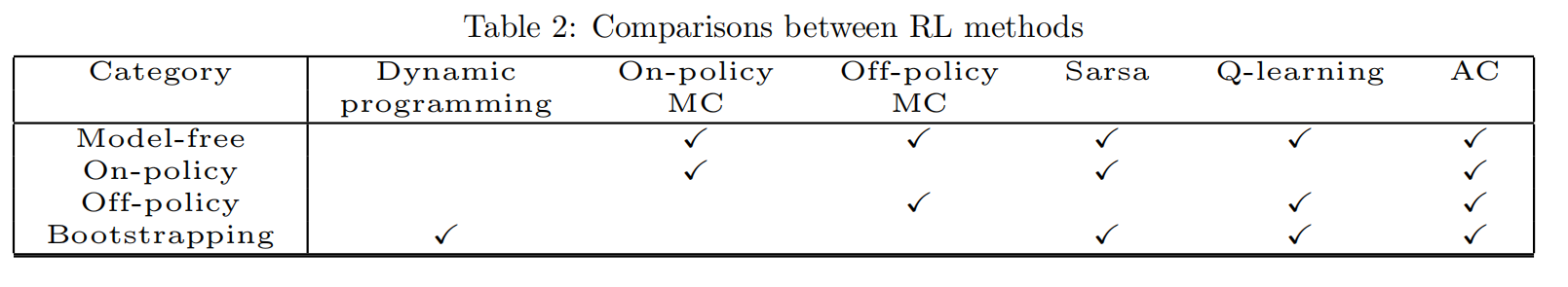
1. AC: Actor-Critic

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 is used to select a suitable action according to the observed state and transfer to critic structure for evaluation. Critic structure uses the following TD error to decide future tendency of the selected action:





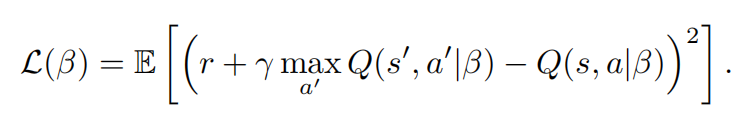




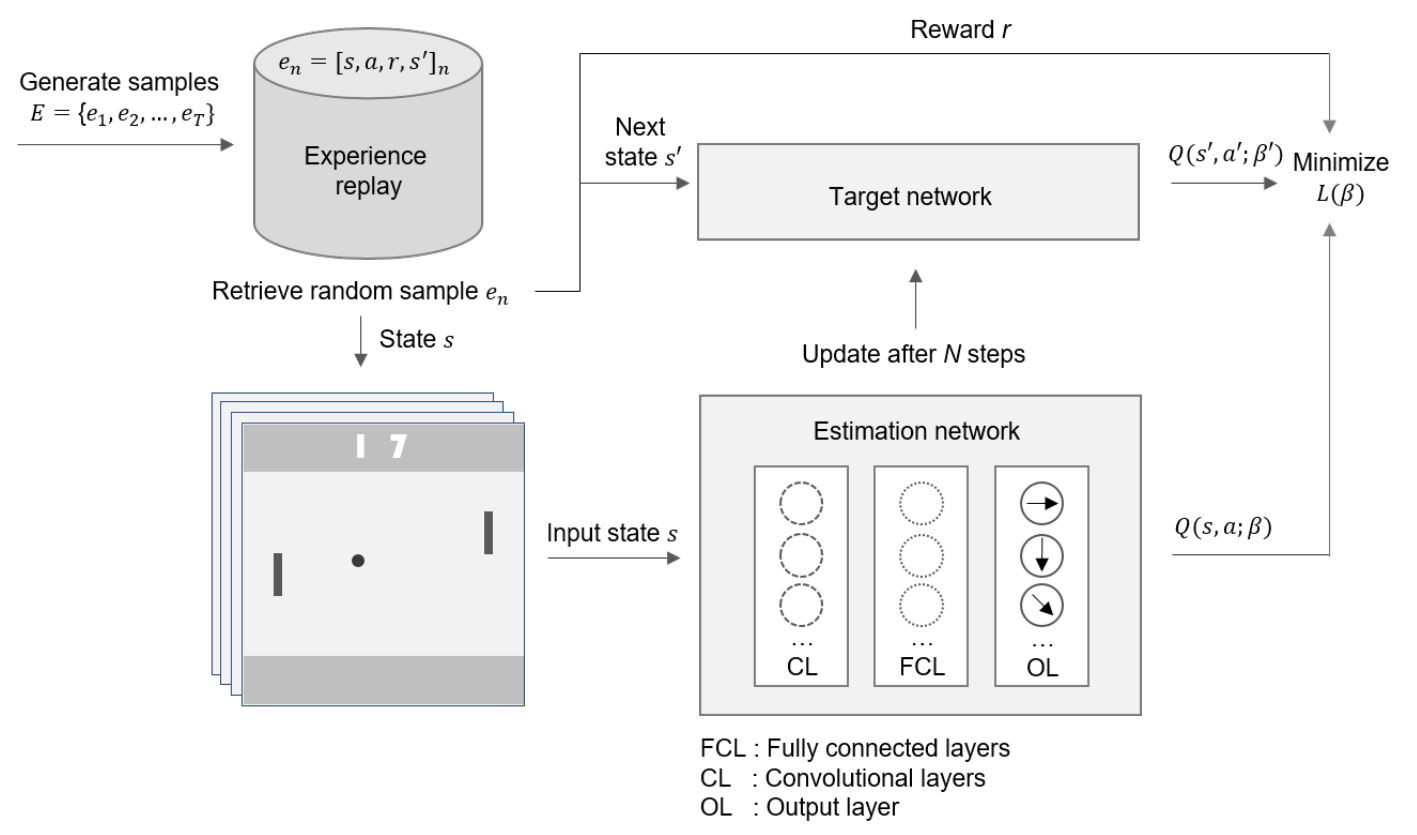
**Deep RL：For single**

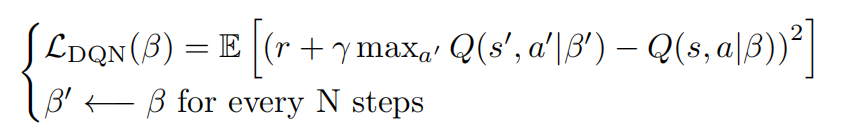
1. DQN

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



To make the samples uncorrelated, created a target network τ0, parameterized by β0, which is updated in every N steps from estimation network τ . Moreover, generated samples are stored in an experience replay memory. Samples are then retrieved randomly from the experience replay and fed into the training process.





DQN’s drawbacks:

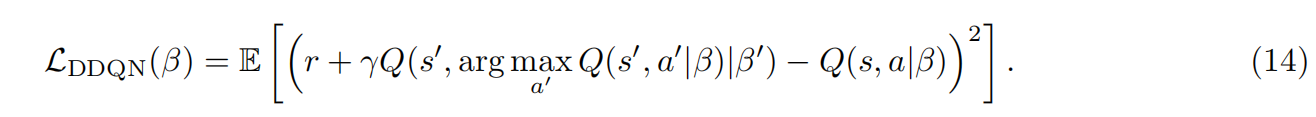
1. DQN’s policy evaluation process is struggle to work in “redundant” situations.
2. It uses a history of four frames as an input to policy network.DQN is therefore inefficient to solve problems where the current state depends on a significant amount of history information, partially observable MDP problems.

**WHY WILL IT BE DIVERGENCE WHEN USING DQN WITHOUT THE PREVIOUS EQUALTION?**

1. DQN - variants

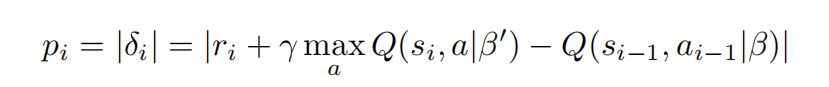
DDQN:

The idea of DDQN is to separate the selection of “greedy” action from action evaluation. In this way, DDQN expects to **reduce the overestimation of Q-values in the training process**.



DDQN with PER(Prioritized experience replay): **For drawback 1**

Specifically, we prefer rare and goal-related samples to appear more frequent than redundancy ones.prioritized experience replay that gives priority to a sample i based on its absolute value of TD error:



DRQN: **For drawback 2**

The straightforward solution is to replace the fully-connected layer right after the last convolutional layer of the policy network with a recurrent long short term memory, This DQN’s variant named deep recurrent Q-network (DRQN)

Deep attention recurrent Q-network (DARQN)

**Deep RL：For Multi-Agent: MADRL**

In a multi-agent learning domain, the MDP is generalized to a stochastic game. The value function of each agent is dependent on the joint action and joint policy, which is characterized.

1. **Challenges & Solutions**

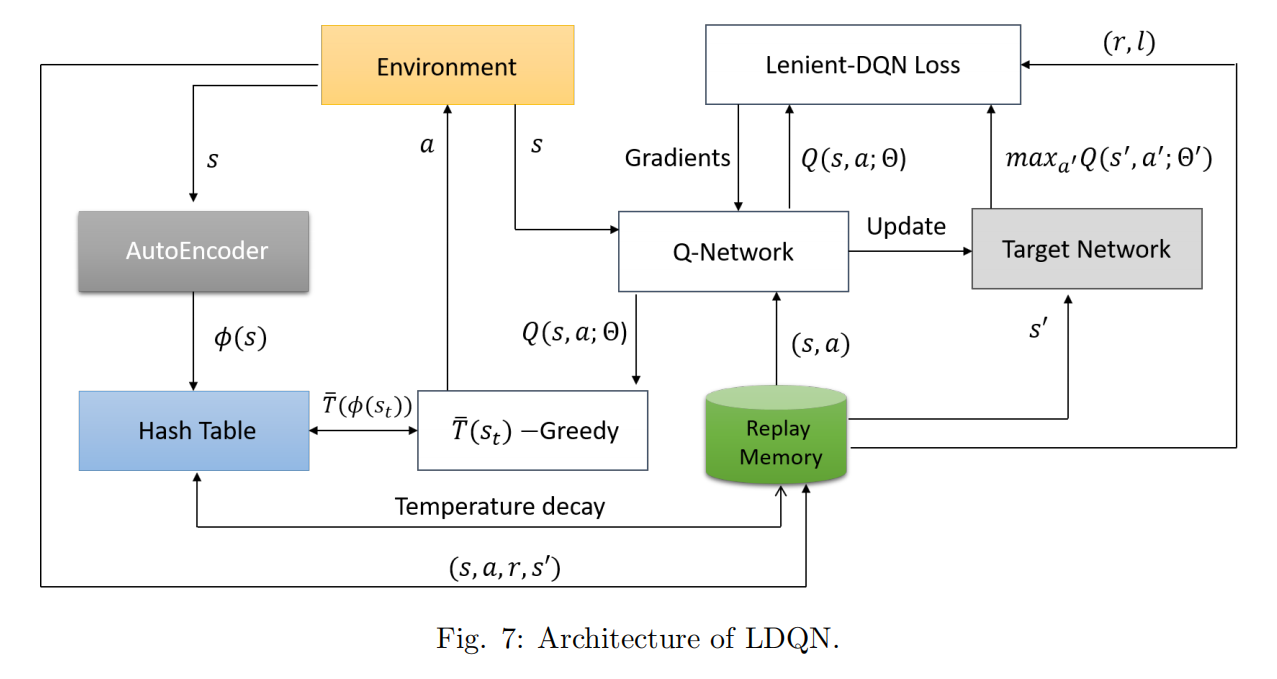
**Non-stability**

agents potentially interact with each other and learn concurrently, reshape the environment and lead to non-stability. In this case, learning among the agents sometimes causes changes in the policy of an agent, and can affect the optimal policy of other agents. The convergence theory of Q-learning applied in single agent setting is not guaranteed to most multi-agent problems as the Markov property does not hold anymore in the non-stationary environment.

The exploration-exploitation dilemma could be more involved under multi-agent settings, 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. **DRUQN??DLCQN??** the agent learns to decide whether it needs to act independently or cooperate with other agents in different circumstances. A multi-agent concurrent DQN and demonstrated that this method can converge in a non-stationary environment.

LDQN: That method is applied to the coordinated multi-agent object transportation problems and its performance is compared with the hysteretic-DQN (HDQN)

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



**Partially-observed**

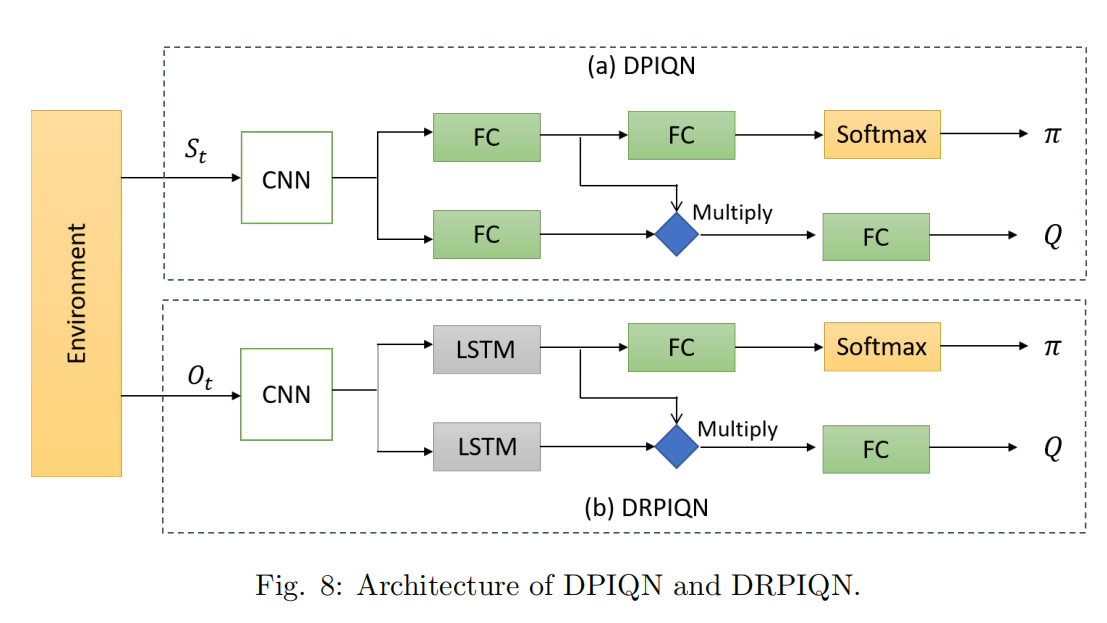
In such situations, the agents observe partial information about the environment, and need to make the best decision during each time step. This type of problem can be modeled using the partially observable Markov decision process (POMDP)

proposed deep recurrent Q-network (DRQN) based on a long short term memory network. With the recurrent structure, the DRQN-based agents are able to learn the improved policy in a robust sense in the partially observable environment.

DRQN is extended to deep distribute recurrent Q-network (DDRQN) to handle multi-agent POMDP problems. last-action inputs, inter-agent weight sharing, and disabling experience replay. last-action inputs, requires the provision of previous action of each agent as input to its next step. The inter-agent weight sharing means that all agents use weights of only one network, which is learned during the training process. The disabling experience replay simply excludes the experience replay feature of DQN.

CL(curriculum learning) integrates with three classes of deep RL, including policy gradient temporal-difference error, and actor-critic methods. The curriculum principle is to start learning to complete simple tasks first to accumulate knowledge before proceeding to perform complicated tasks.

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. Both DPIQN and DRPIQN are learned by adapting network’s attention to policy features and their own Q-values at various stages of the training process.



Apart from partial observation, there are circumstances that agents must deal with extremely noisy observations, which are weakly correlated with the true state of the environment.

**MAS-training**

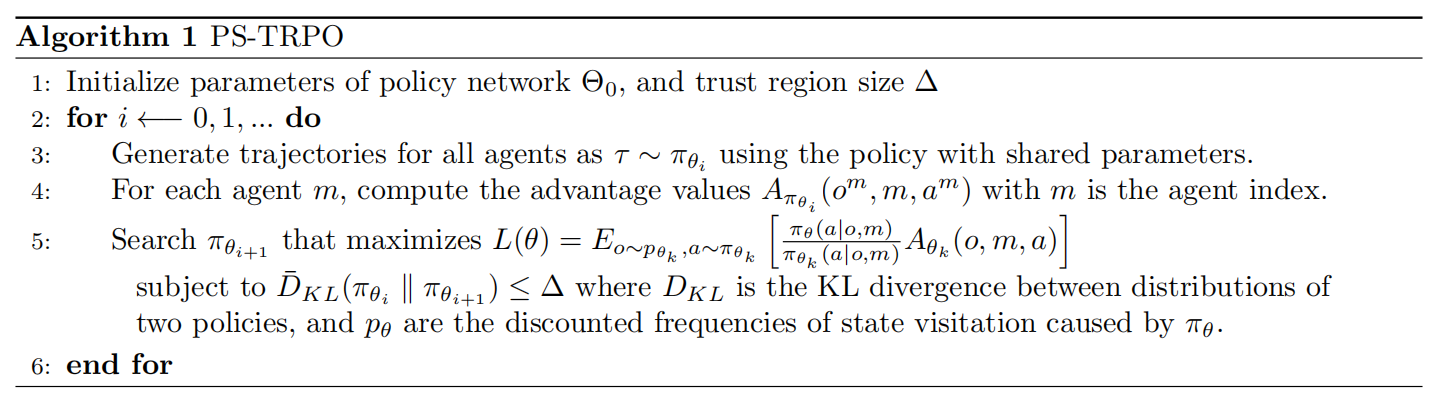
An alternative and popular approach is the **centralized learning an decentralized execution** where a group of agents can be trained simultaneously by applying a centralized method via an open communication channel.

Decentralized policies where each agent can take actions based on its local observations have an advantage under partial observation and in limited communications during execution.

Centralized learning of decentralized policies has become a standard paradigm in multi-agent settings because the learning process may happen in a simulator and a laboratory where there are no communication constraints, and extra state information is available.

Centralized policy attempts to obtain a joint action from joint observations of all agents whilst the concurrent learning trains agents simultaneously using the joint reward signal. In the latter, each agent learns its own policy independently based on private observation. Alternatively, the parameter sharing scheme allows agents to be trained simultaneously using the experiences of all agents although each agent can obtain unique observations.

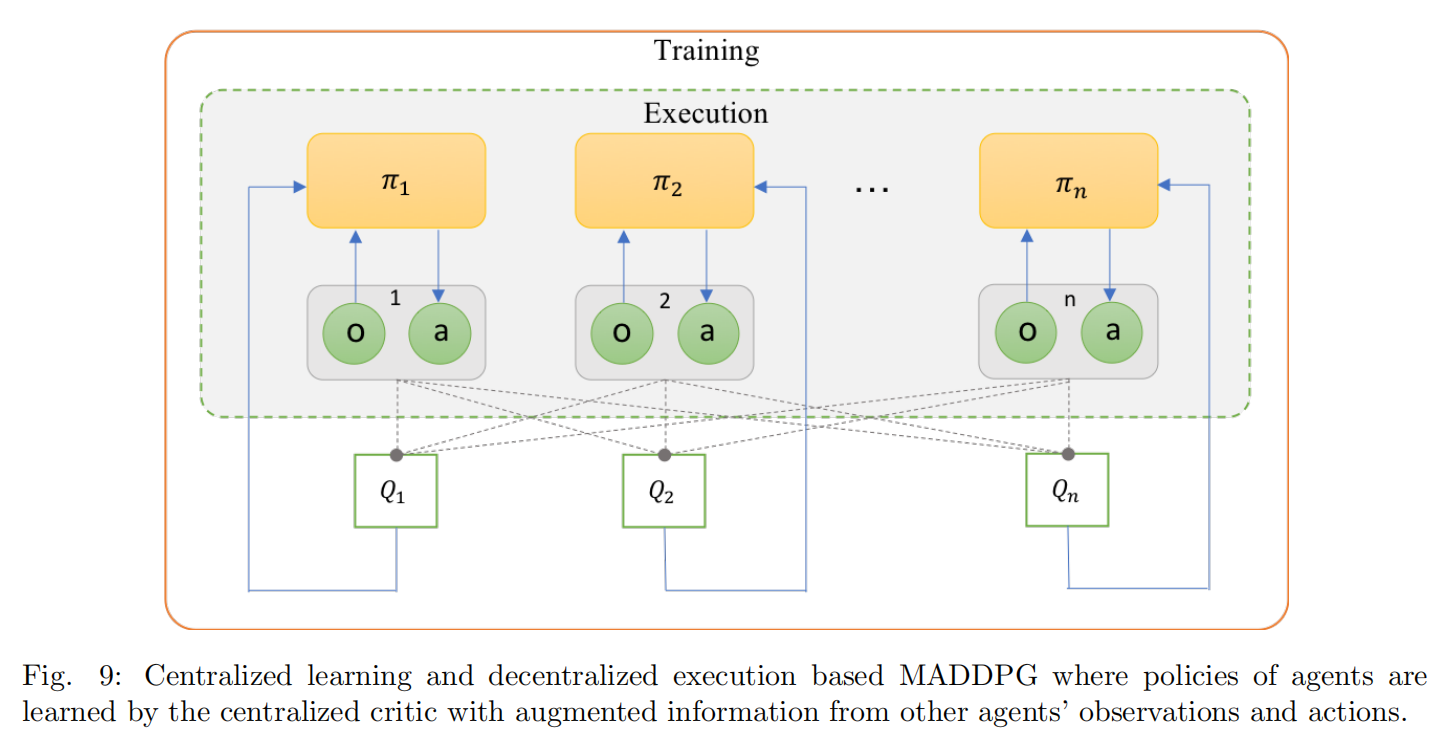
The PS(parameter sharing)-TRPO has demonstrated great performance when dealing with high-dimensional observations and continuous action spaces under partial observation.



DIAL pushes gradients from one agent to another through a channel, allowing end-to-end back-propagation across agents. developed communication neural net (CommNet) allowing dynamic agents to learn continuous communication alongside their policy for fully cooperative tasks.

both decentralized and centralized perspectives into the hierarchical master-slave architecture to form a model named master-slave multi-agent RL (MS-MARL) to solve the communication problem in MAS.

MADDPG features the centralized learning and decentralized execution paradigm in which the critic uses extra information to ease training process whilst actors take actions based on their own local observations. Fig. 9 illustrates the multi-agent decentralized actor and centralized critic components of MADDPG where only actors are used during the execution phase.



**Continuous Action Space**

trust region policy optimization (TRPO) can be extended to continuous states and actions, for optimizing stochastic control policies in the domain of robotic locomotion and image-based game playing.

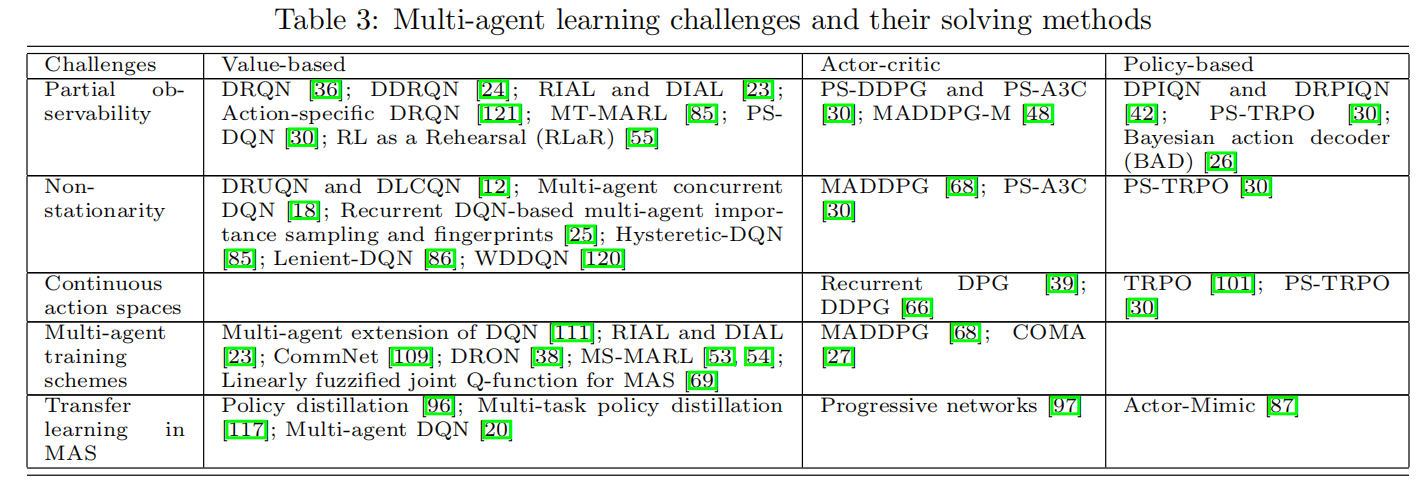
an off-policy algorithm, namely deep deterministic policy gradient (DDPG), which utilizes the actor-critic architecture to handle the continuous action spaces.

Extended DDPG to recurrent DPG (RDPG) to handle problems with continuous action spaces under partial observation, where the true state is not available to the agents when making decisions

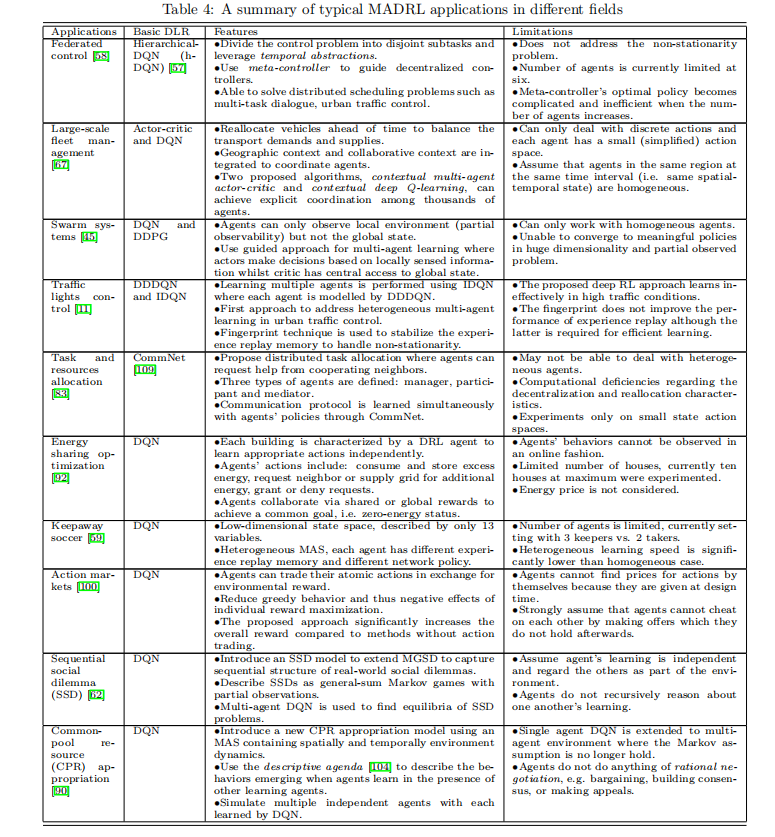
**Transfer-learning for MADRL**

Training is computationally expensive. This is significantly severe. knowledge transfer for deep RL.

the actor-mimic method for multi-task and transfer learning that improves learning speed of a deep policy network.



1. Applications



**Conclusion：**

This paper presents an overview of different challenges in multi-agent learning and solutions to these challenges using deep RL methods.

On one hand, imitation learning tries to map between states to actions as a supervised approach.

inverse RL agent needs to infer a reward function from the expert demonstrations.

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

Model-free deep RL has been able to solve many complicated problems both in single agent and multi-agent domains.

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.