An Overview for Reinforcement Learning

Haoxin Liu

August 26, 2020
(first version)

1 Basic definition of RL and its history

1.1 Basic definition

We use a 4-tuple $< S, A, R, T >$ to define the Markov dicision process in reinforcement learning as below:

1. S: the state space of the environment.
2. A: the action space of an agent.
3. $R(s, a)$: reward function, the return value of this function represents the reward of performing action a in state s.
4. $T(s^{'}|s, a)$: transition probability function, representing the probability of jumping to state $s^{'}$ by performing action a in state s.

The main goal of RL is to find a policy π which can maximize the expected future reward $E(\sum_{t=0}^{n} \gamma^t R_t)$, where R_t is the reward obtained at time step t and γ is the discount factor.
If we know everything in MDP, we can directly solve it without making any action in the environment. We usually call the decision made before the action is executed as *'planning'*. In this case, some classical planning algorithms can directly solve MDP problems, including *value iteration* and *policy iteration*.
But in reinforcement learning, the agent is not so easy to know all the elements in the MDP, especially *transition function* T and *reward function* R. What an agent can do is: based on the current policy π, choose the local best action a in current state s and perform it to the environment, then observe the feedback r given by the environment and the next state s', and update their policy π according to the feedback r. Repeated this iteration, until we find an optimal strategy $\pi^{'}$ can obtain maximum positive feedback.

1.2 History of RL

Here, I will briefly show the history of reinforcenment learning in the table below. I think understanding the development trend of reinforcement learning will be helpful for future research. Therefore, reinforcement learning has been developed for decades and is not a new technology. However, Google Deepmind team combined deep learning with reinforcement learning and proposed DQN in 2013, which is a critical time point. After that DRL(deep reinforcement learning) became one of the most popular technologies in these years, especially after AlphaGo defeated Lee Sedol in 2016. Meanwhile, with the technology of single-agent perception and reinforcement learning becoming more and more mature, multi-agent dicision making problem gradually draw more researcher's attention and multi-agent reinforcement learning(MARL) become the key method of solving this problem.

Year	Author	Algorithm name
1956	Bellman	dynamic programming
1988	Sutton	Temporal-difference learning
1992	Watkins	Q-learning
1994	Rummery	Sarsa
2014	Silver	Policy Gradients
2015	Google Deepmind	Deep-Q-Network(DQN)

2 Classification of RL methods

Reinforcement learning methods can be categorized according to different criteria. Understanding these categories helps us to feel the property and differences of different RL approaches, so I will briefly talk about four categories of RL and there is a table to show all classification in Fig. 1.

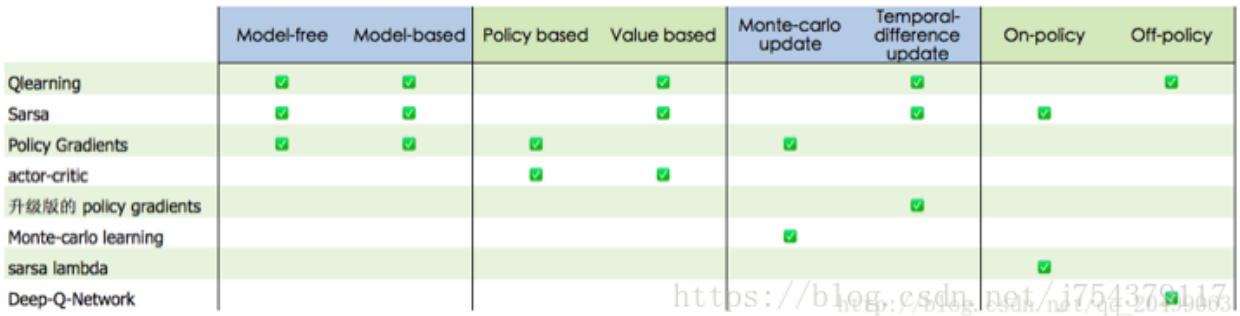

Fig. 1. RL-classification

2.1 Model-free and Model-based

Model-based reinforcement learning methods try to understand the environment through an agent and build a model to represent the environment. The *'model'* here refers to the description of the probability distribution of transitions between states in an environment. This model's goal is to learn these two functions:

1. Transition function from states ($T : S \rightarrow S$).

2. Reward function R.

Specifically, when an agent in state s_1 performed action a_1, then observed the environment from s_1 to s_2 and received rewards r, then this information can be used to improve the estimate accuracy on $T(s_2|s_1, a_1)$ and $R(s_1, a_1)$. If the learning model is very close to the real environment, then the agent can directly use some planning algorithm to find the optimal strategy.
While model-free reinforcement learning methods try to learn *policy* directly. The most significant example of model-free method is *Q-learning*, which directly estimates the future return Q(s, a).
The distinction between model-free and model-based generally refers to whether the state transition matrix is fully known. Here is a way to identify the two methods: before the agent executes its action, can it make a prediction on the next state and return? If yes, then it is model-based method; if not, it is model-free method.
The common method in the two categories is shown in Fig. 2.

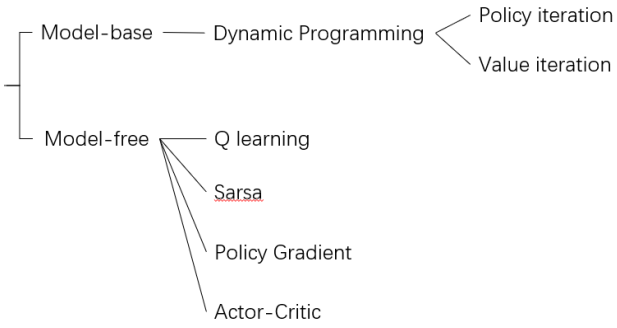

Fig. 2. model-base and model-free

2.2 Value-based and Policy-based

Policy-based learning is the most direct type of reinforcement learning. It can directly output the probability of various actions to be taken in the next step, and then take actions according to the probability. Therefore, each action is likely to be selected with different possibilities. And the value-based methods output the value of all actions, we will choose the action with the highest value. The Fig. 3 clearly shows their difference.

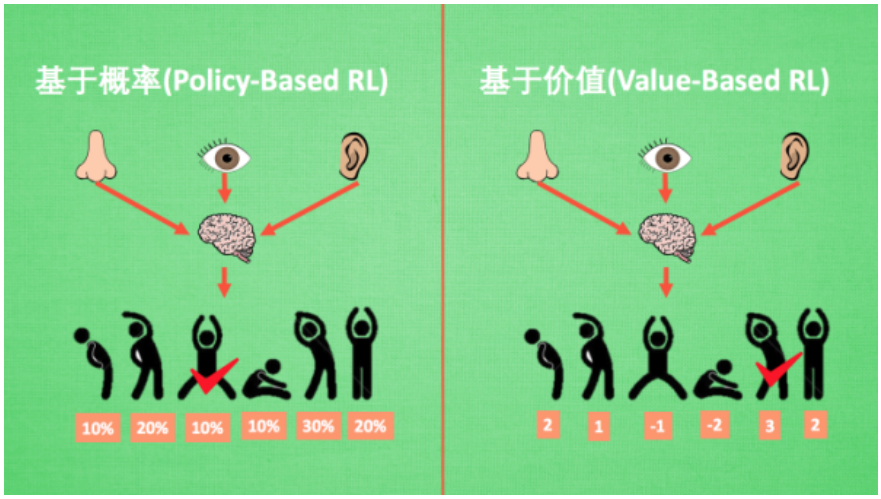

Fig. 3. policy-base and value-base

For value-based learning, we define such value functions:

– G: Cumulative rewards, G_t represents the cumulative reward from time step t to the end.

$$G_t = \sum_{T=t}^{\infty} r_t$$

– $Q_\pi(s, a)$: state-ation value function, which represents the expected cumulative reward of performing action a in state s at time t under policy π.

$$Q_\pi(s, a) = E[G_t | S_t = s, A_t = a]$$

– $V_\pi(s)$: state value function, which represents the expected cumulative reward in state s under policy π.

$$V_\pi(s) = E[G_t|S_t = s]$$

The value-based method is based on these two functions ($V_\pi(s)$, $Q_\pi(s,a)$) to figure out the strategy for maximizing reward:

$$\pi_* = arg\max_\pi V_\pi(s) \quad or \quad \pi_* = arg\max_\pi Q_\pi(s,a)$$

There are many powerful value-based RL methods, why do we still need policy-based method? This because value-based learning has some drawbacks:

1. Insufficient handling of continuous action.
2. Insufficient ability to deal with problems under restricted state.
3. The need for a random strategy cannot be met. In some games, we can't have a definite strategy, such as rock- paper-scissors. Once our strategy is discovered by the other side, our strategy will fail, so our strategy needs to be a little bit random.

All value-based methods are ultimately learned through Value functions, which may be related to actions or states. The policy-based method, on the other hand, uses the gradient method to learn the strategy directly. In the V-B (value-based) method, the strategy is $\pi(s)$, only dependent on the state, and we modify the strategy by feedback of the Value function, and then calculate the Value function according to the strategy. In the P-B (policy-based) method, the strategy is $\pi(s|a,\theta)$, with a new parameter θ added. Instead of indirectly optimizing $\pi(s)$ based on the V-Q function, we aim to optimize $\pi(s)$ directly for the strategy by optimizing the parameter θ.
The Fig. 4 shows common methods of the two categorizes.

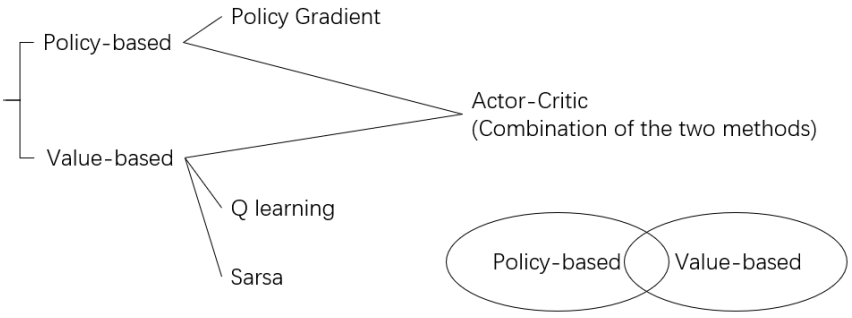

Fig. 4. Common methods

2.3 Monte-carlo update and Temporal-difference update

Monte-carlo update means that after the game starts, we wait for the game to finish, then summarize all the turning points in the episode and update our policy. Temporal-difference update means we update at every step of the game, without waiting for the end of the game, so that we can learn as we play. The Fig. 5 shows their difference.

The Fig. 6 shows common methods of the two categorizes.

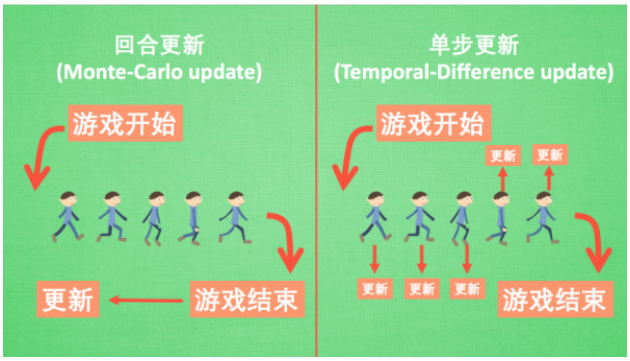

Fig. 5. Monte-carlo update and Temporal-difference update

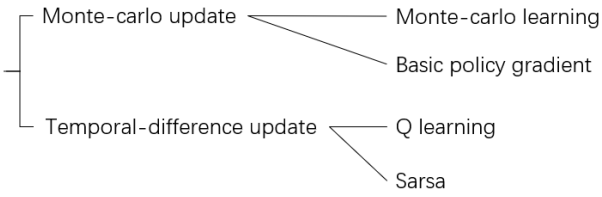

Fig. 6. common methods of the two categorizes

2.4 On-policy and Off-policy

Reinforcement learning can be divided into *off-policy* (offline) and *on-policy* (online) learning methods. According to my understanding, the basis for judging whether an reinforcement learning is off-policy or on-policy is whether the policy (value-funciton) in the generated samples and the policy (value-funciton) in the network parameter update are the same. Let's take Q-learning and SARSA as examples.

Q-learning uses *max* operation to directly select the optimal action when calculating the expected reward of the next state. However, the current policy does not necessarily select the optimal action. Therefore, the policy generated here is different from the policy during learning, which is the off-policy algorithm. The disadvantages of off-policy methods are tortuous and slow convergence, but the advantages are more powerful and universal. It is powerful because it ensures that the data is comprehensive and that all behaviors can be covered.

SARSA, on the other hand, performs an action selection directly based on the current policy, and then updates the current policy with this sample, so the policy generated is the same as the policy during leaning, and the algorithm is on-policy algorithm. On-policy methods will encounter the contradiction of exploration and exploition. By simply using the currently known optimal choice, it may not be able to learn the optimal solution and converge to the local optimal solution, while adding exploration will reduce the learning efficiency. The epsilon-greedy algorithm is a compromise among these contradictions. The advantage is straightforward, fast, the disadvantage is not necessarily to find the optimal strategy.

The essential difference between on-policy and off-policy is whether the method used to update the Q value follows the established policy or uses the new policy.

The Fig. 7 shows common methods of the two categorizes.

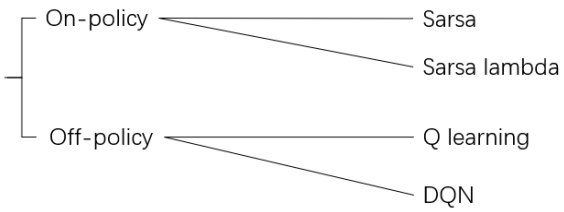

Fig. 7. common methods of the two categorizes

3 Conclusion of common reinforcement learning methods

3.1 Q-learning

Q-learning is the most significant method in reinforcement learning. I will brief conclude Q-learing according to the graph 8 obtained online.

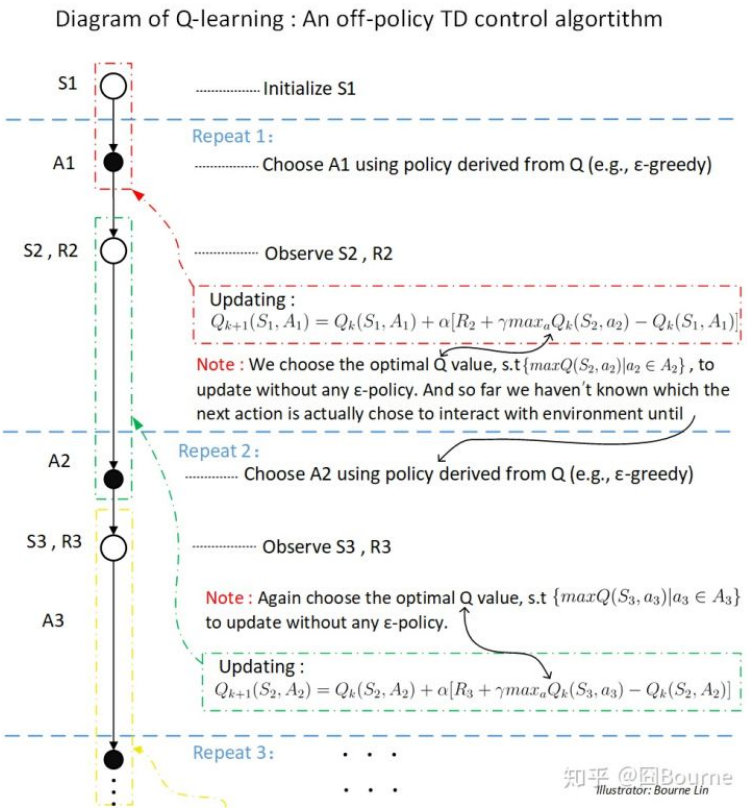

Fig. 8. Q-learning

1. An episode begins, randomly choose and initialize the first state S_1.
2. Enter Repeat 1: Firstly, choose an action A_1 from state S_1 according to $\epsilon - greedy$ strategy. After conducting action A_1, observe next state S_2 and get the immediate reward R_2. Now, update the Q function.

3. After the update, the Repeat 2is performed: the action is selected to interact with the environment in the state S_2 based on the $\epsilon - greedy$ strategy (the action was not taken previously during updating in the state S_2). It is worth noting that the action a_2 selected by us in the Repeat 1is the only one with the highest Q-value ($\max_a Q_k(S_2, a_2)$). In Repeat 2, the second action A_2 that will interact with the environment is based on the $\epsilon - greedy$ strategy.

3.2 SARSA: state–action–reward–state–action

Sarsa is similar but different to q-learning, I will briefly conclude SARSA according to the graph 9 obtained online.

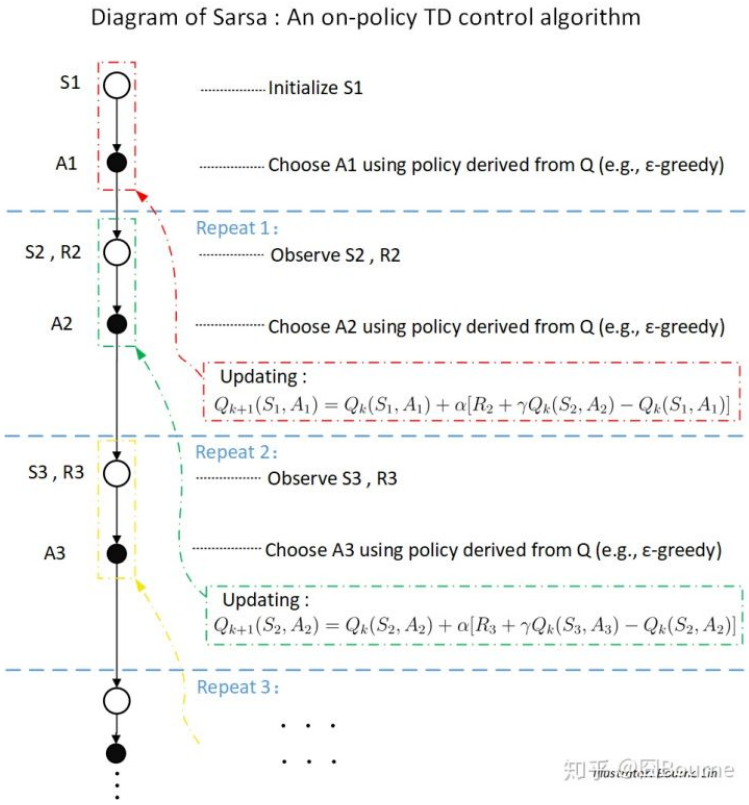

Fig. 9. SARSA

1. An episode begins, randomly choose and initialize the first state S_1. Then, choose action A_1 in state S_1 based on the $\epsilon - greedy$ strategy.
2. Enter the Repeat 1: conduct action A_1, observe next state S_2 and get the immediate reward R_2. Now, choose action A_2 in state S_2 based on the $\epsilon - greedy$ strategy again. After we get A_2, we can update Q-function using the action obtained from the $\epsilon - greedy$ strategy, which is the difference to Q-learning and why sarsa is on-policy learning.
3. Repeat the second step until the end state.

3.3 Policy Gradients

(To be continued...)

4 Brief summay of deep reinforcement learning

4.1 DQN and its variants

Deepmind and its DQN is the pioneer in the field of deep reinforcement learning. The concept of DQN proposed in 2013 and 2015. The interesting thing is that Deepmind was bought by Google because of the DQN paper. Fig. 10 shows the history of DQN:

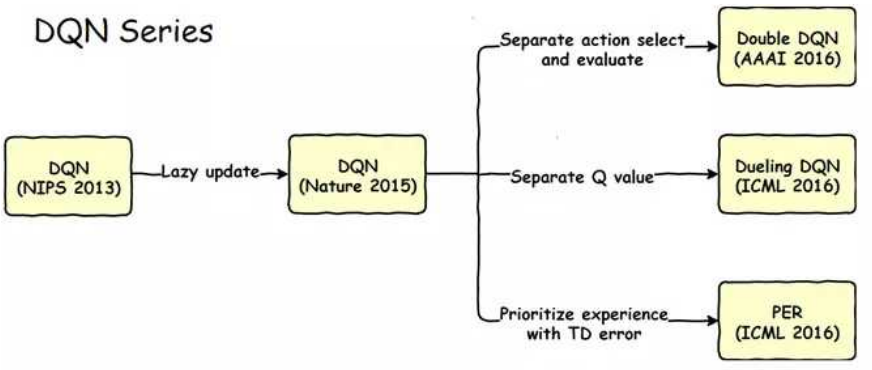

Fig. 10. DQN and its variants

Previously, there were some challenges in combining RL and DL: 1. Deep learning algorithm requires large amounts of labeled data, most *'rewards'* learned by RL are sparse with noise, and delayed (*'delay'* means the gap between action and resultant rewards). 2.DL assumes that the samples are independent while the adjacent states are correlated; 3.DL assumes a fixed distribution, while RL changes the data distribution as it learns new behaviors. DQN solves these problems by using the *reward* to construct labels through Q-learning and using experience buffer.

DQN proposed two new features:

1. Experience replay: Each time the DQN is ready to update, some previous experiences are randomly selected for learning. Random sampling disrupts correlations between experiences, making neural network updates more efficient.
2. Fixed Q-targets: DQN uses two neural networks with the same structure but different parameters, the neural network that predicts Q-estimation has the latest parameters, while the neural network that predicts Q reality uses parameters that are kept for some period.

The flow diagram in Fig. 11 shows how DQN works.
The detailed description for DQN will be implemented here later...

4.2 DDPG

(To be continued...)

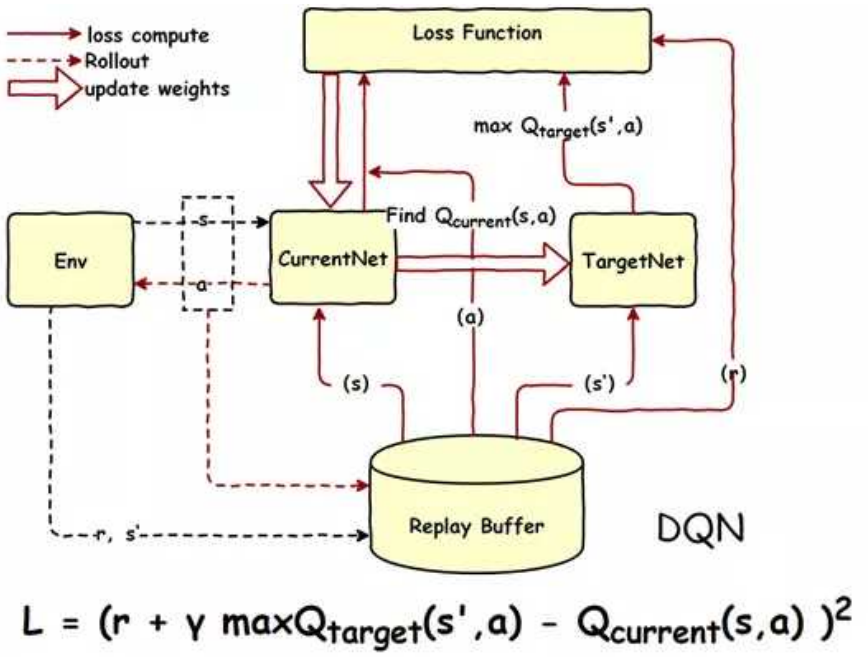

Fig. 11. Deep-Q-Network

4.3 Actor-Critic

(To be continued...)

5 Brief summary of multi-agent reinforcement learning

Four environment: fully-cooperative, fully-competitive, semi-cooperative, mixed environment. I briefly introduce one model of each environment.

5.1 Fully-cooperative

(To be continued...)

5.2 Fully-competitive

(To be continued...)

5.3 Mixed environment: MADDPG

(To be continued...)

5.4 Semi-cooperative: PED-DQN

(To be continued...)

6 Application of reinforcement learning

(To be continued...)

References