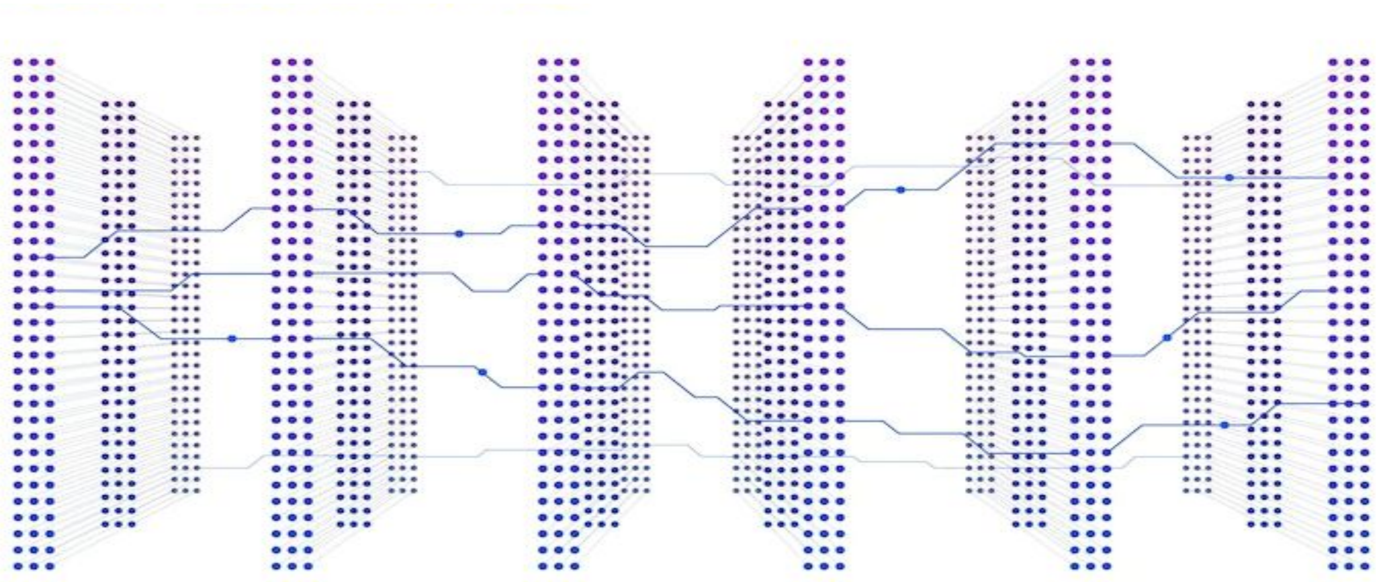
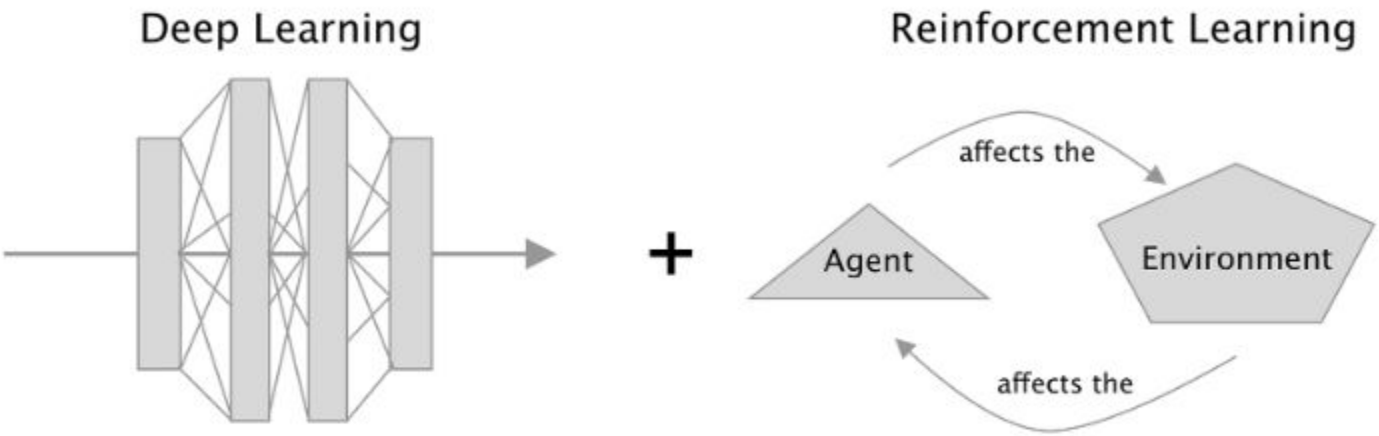


Deep Reinforcement learning

Double Deep Q Networks

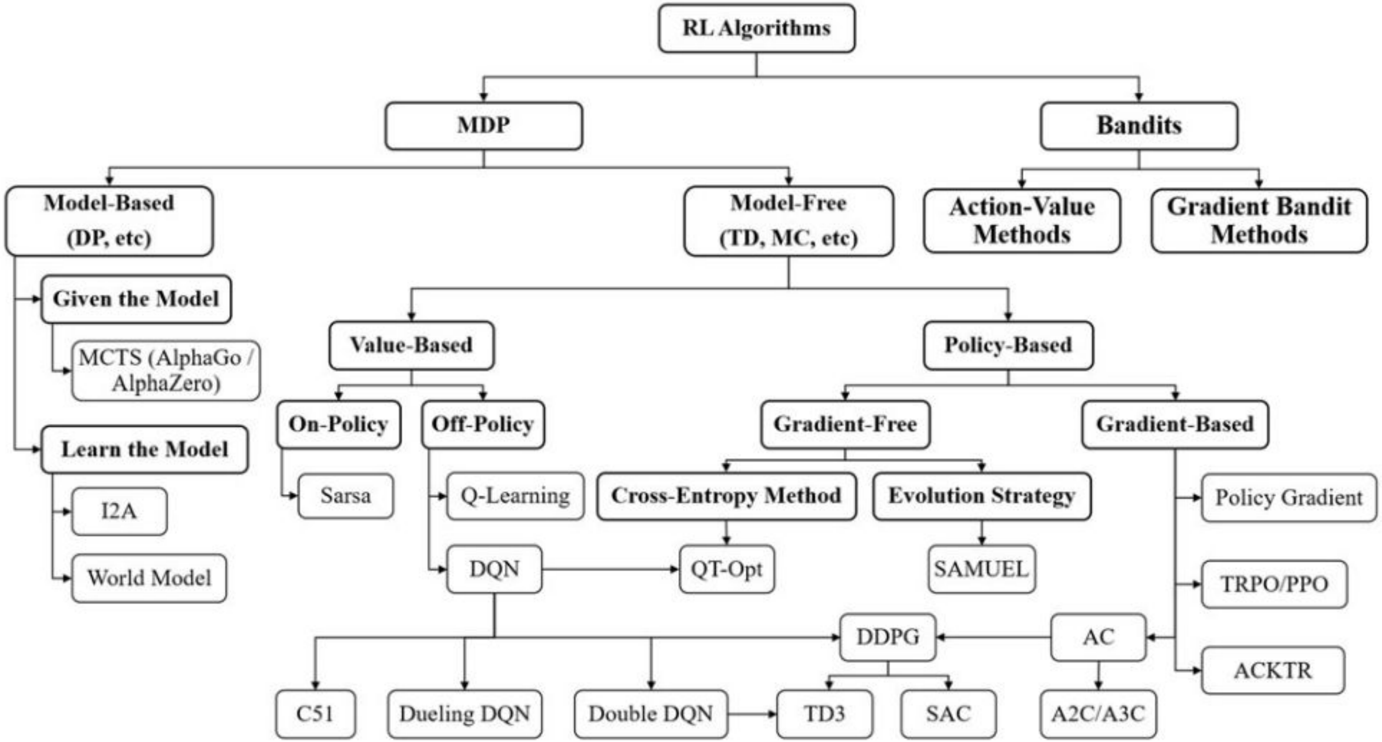
Mohammad.H.Nili
Mersad.Esalati

DL + RL

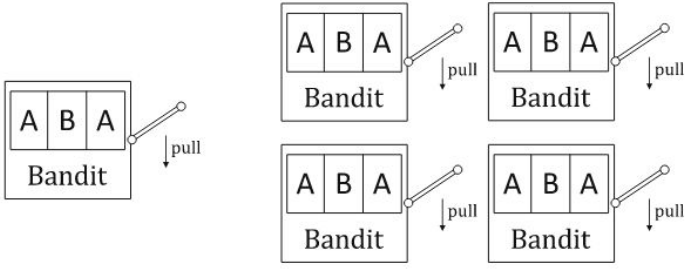

Deep Learning
+
Reinforcement Learning
affects the
Agent
Environment
affects the

=

Taxonomy of Reinforcement Learning Algorithms

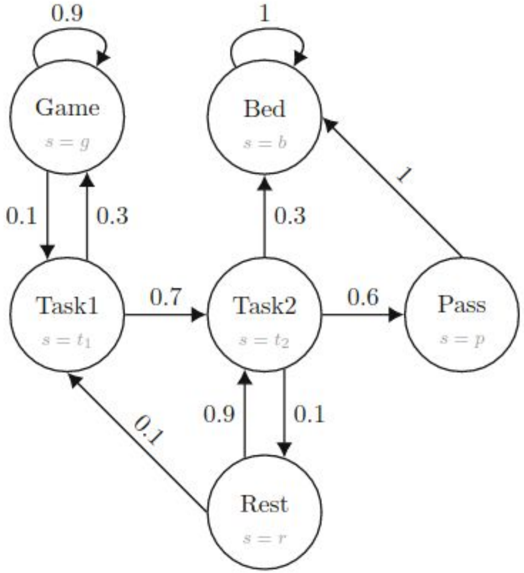


Bandit and MDP



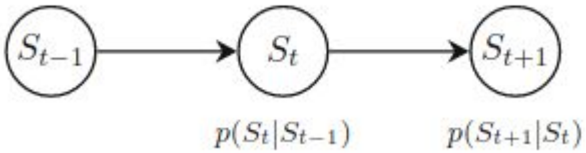
Single-armed Bandit

Multi-armed Bandits



Markov decision process

Markov chain:



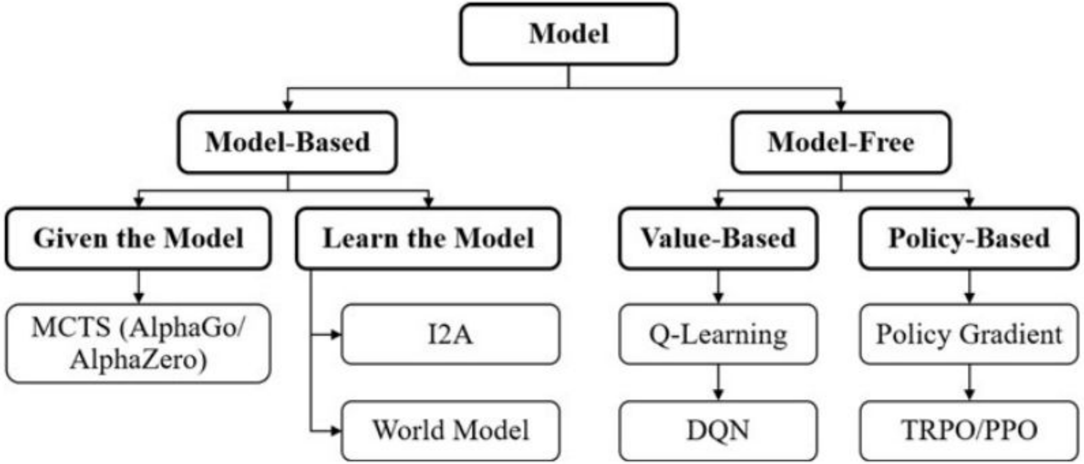
$$p(S_{t+1}|S_t) = p(S_{t+1}|S_0, S_1, S_2, \ldots, S_t)$$

$$p(s'|s) = p(S_{t+1} = s'|S_t = s)$$

$$\begin{array}{c} \quad\; g \quad\; t_1 \quad t_2 \quad\; r \quad\; p \quad\; b \\ \boldsymbol{P} = \begin{bmatrix} 0.9 & 0.1 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0.6 & 0.3 \\ 0 & 0.1 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{array}{c} g \\ t_1 \\ t_2 \\ r \\ p \\ b \end{array} \end{array}$$

Model-based and model-free

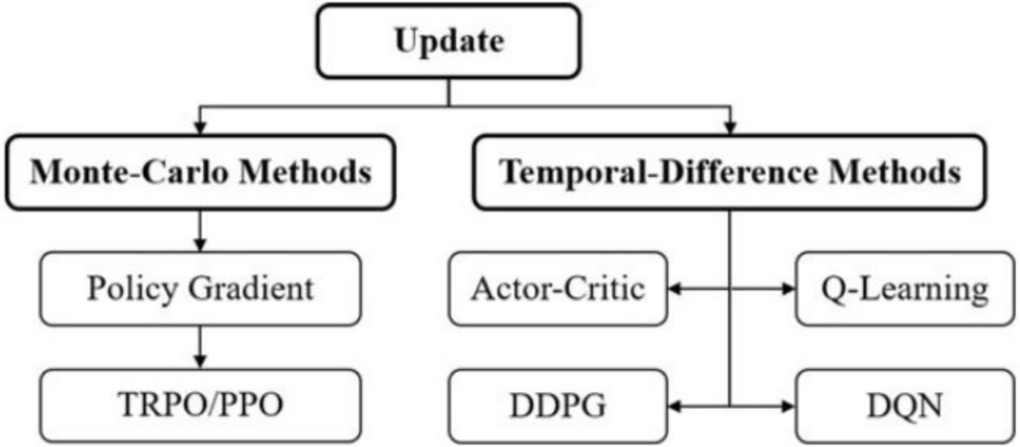
- State space
- Action space
- Reward function
- Transition function
- Discount factor



The difference between model-based and model-free is whether the agent will get or learn the model (or dynamics) of the environment, such as the transition function and the reward function.

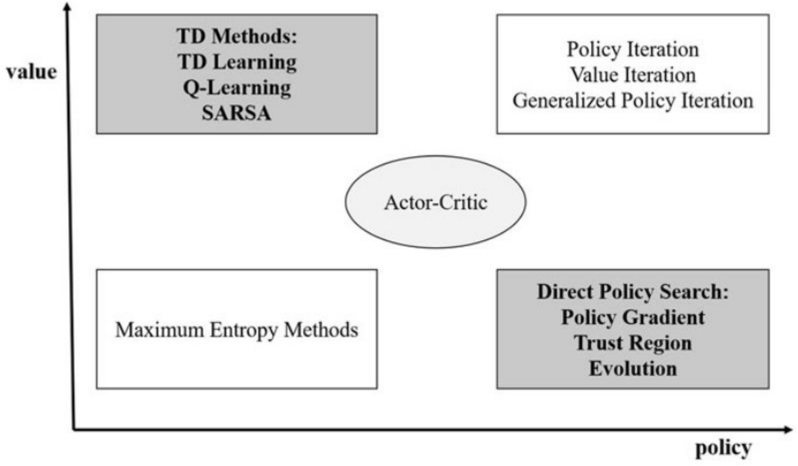
The key advantage of model-based methods is that the future states and rewards can be anticipated in advance via the environment model, which helps the agent to make better planning.

Monte Carlo methods and Temporal-difference



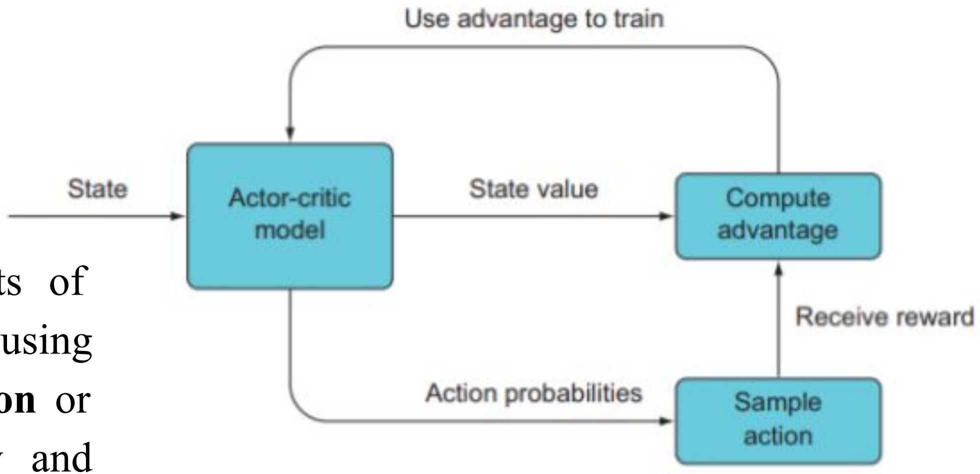
TD is an intermediate form between dynamic programming (**DP**) and **MC** methods. Both TD and DP use **bootstrapping** for estimation and both TD and MC do not require the full knowledge of the environment. What makes MC differ from TD the most is how the learning update is done. **MC** has to **wait** until an **episode is finished** to update, whereas **DP** can do an **update** at each time step. This difference will let **TD methods have larger biases**, whereas **MC methods have larger variance**.

Value-based and policy-based



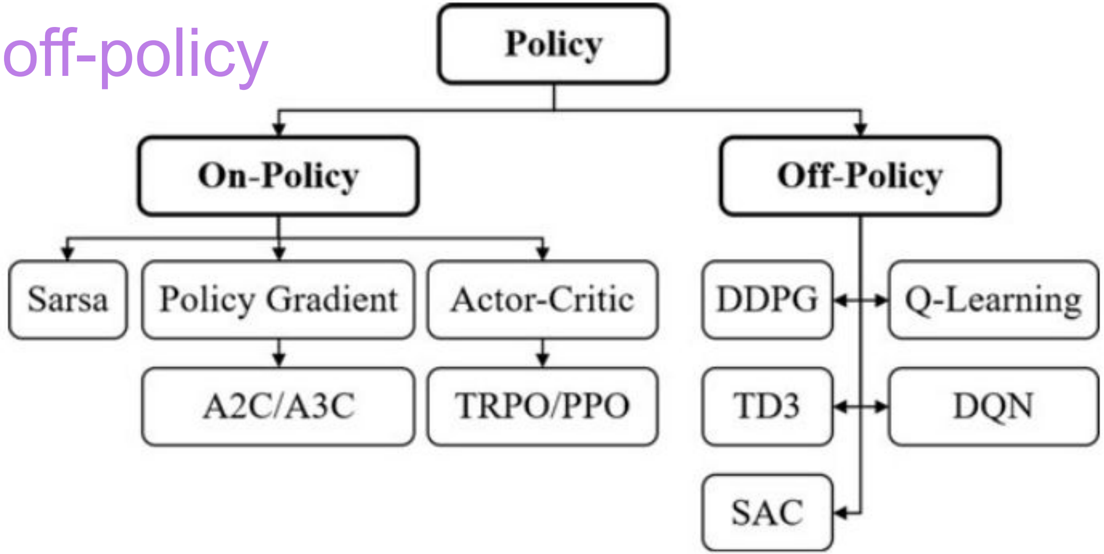
The **advantages** of value-based method lie in the **sample efficiency** is **high**, the variance of value function **estimation** is **small**, and it is not easy to fall into local optimum. The **disadvantages** are that it usually **cannot** handle the **continuous action space** problem, and the **ε-greedy strategy** and the max operator such as in **DQN** can easily result in overestimation. **policy-based** method has the advantages of **simpler policy** parameterization, **better convergence**, and is suitable for **continuous** or **high dimensional action space**.

The actor-critic



The **actor-critic** method **combines** the merits of **value-based** method and **policy-based** method, using the **value-based** methods **to learn a Q function** or value function to improve sample efficiency and using the **policy-based** methods **to learn the policy function**, which is suitable for discrete or **continuous** action space. This kind of method can be regarded as an **extension** of the **value-based** methods in continuous action space, or as an improvement of the policy-based method for **reducing sampling variance**. Although this method absorbs the advantages of the two methods, it also inherits the corresponding disadvantages. For example, the critic also has the problem of **overestimation**, and the actor has the problem of insufficient explore.

On-policy and off-policy



On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data. The **on-policy** method requires the agent itself to **interact** with the environment; that is to say, the interact with the environment and the **policy to be improved must be the same one**. The off-policy method does not need to conform to it, the experience of other agents interacting with the environment can also be used to improve the policy. In the on-policy method the policy that **interacts with the environment** and the **updated policy** is the **same one**.

Q-learning: Off-policy TD Control

Q-learning is a typical **off-policy** method. It adopts the max operation and an **ε-greedy** policy when **choosing actions**, which makes the policy that interacts with the environment and the **updated policy not the same policy**. It updates Q function as follow:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big].$$

- State space
- Action space
- Reward function
- Transition function
- Discount factor

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

Double Q-learning

The idea of **double learning** extends naturally to algorithms for full MDPs. For example, the double learning algorithm analogous to Q-learning, called Double Q-learning, **divides** the **time steps** in **two**, perhaps by flipping a coin on each step. If the coin comes up heads, the update is:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big]$$

If the coin comes up tails, then the same update is done with Q1 and Q2 switched, so that Q2 is updated.

Two Q-tables, in essence two value estimates, to **reduce bias**.

Double Q-learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

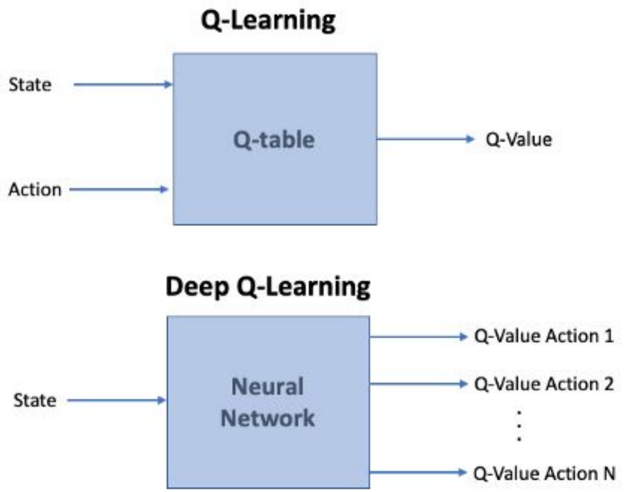
$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Why Deep Learning: Value Function Approximation



In **tabular** settings, the **action-value functions** can be represented by a **big two-dimensional table**, i.e., **one entry** for **each discrete state and action**. However, it is inefficient to deal with large-scale space such as raw pixels input, and let alone **continuous control tasks**. Fortunately, generalization from different inputs by function approximation has been widely studied, and we can utilize this technique in **value-based reinforcement learning**.

Deep Q-networks (DQN)

One of the most significant breakthroughs in reinforcement learning was the development of an **off-policy temporal difference (TD)** control algorithm, known as Q-learning.

Q-Learning has been proven to **converge** towards the **optimal solution** in a **tabular** case or using **linear function** approximation. However, it is known that **Q-learning** is **unstable** or even to **diverge** when using a **nonlinear** function approximator such as a **neural network** to represent the Q-value functions. With the advances in training deep neural networks, deep Q-networks (DQN) addressed this issue and ignited the research of deep reinforcement learning.

Original DQN has the **problem** of **overestimating** the **Q-value**, which **decreases** the **learning performances** in practice, and the **double/dueling DQN** techniques are proposed to alleviate the problem.

Generally, **policy-based** methods with **policy gradients** have stronger **convergence guarantee** compared with **value-based** method.

DQN: Replay buffer

To achieve the **end-to-end** decision-making in complex problems with **raw pixel input**, DQN combines Q-learning with deep learning with **two key ideas** to address the instability issue and achieves significant progress on Atari games:

- **Replay buffer (**In practice, only the last N experience tuples are stored in the replay buffer**)** At each time step t, DQN stores the experience of the agent (S_t , A_t , R_t , S_{t+1}) into replay buffer, and then draws a mini-batch of samples from this buffer uniformly to apply the Q-learning update. Replay buffer has several advantages over the fitted Q iteration:
  - The **experience** in each step can be **reused** to learn the Q-function, which allows for greater data efficiency.
  - If there is **no replay buffer**, as in the fitted Q iteration, **mini-batch** samples are collected consecutively, i.e., they are highly correlated, which increases the variance of the updates.
  - Experience replay avoids the situation that the samples used to train are determined by the previous parameters, which **smooths out learning** and **reduces oscillations** or divergence in the parameters.

DQN:Target network

- **Target network** (improve the stability with NN)
  Instead of the desired Q-network, a separate network is used to generate the Q-learning targets. Furthermore, at every C-steps, the target network will be synchronized with the primary Q-network by copying directly (hard update) or exponentially decaying average (soft update). The target network makes the generation of the Q-learning target delay with old parameters, which reduces the divergence and oscillations much more. For example, the update making Q-value increase on action (St , At) may increase Q(St+1, a) for all action a because of the similarity between St and St+1 , where the training target constructed by Q-network will be overestimated.

- **Gradient (loss) clipping**
  Iterations the TD error in Q-learning can be large. This large TD-error will generate large gradients of parameters with respect to loss which can destabilize the whole learning →**(RNN)**

Bounding the loss Vs clipping the gradient

There is a subtle **difference** between bounding the loss using the **huber-loss** technique and clipping the gradient as explained below:

1. While **clipping** the gradients, we first **compute** the **gradients** of **each trainable variables** using **back propagation** then we clip the gradients and apply it to change the variables using the Stochastic Gradient Descent (**SGD**) method.

2. In **bounding** the loss, we **change** the **loss function** such that the gradient of the loss with respect to error will be bounded. Then this bounded gradient will be **back-propagated** in **computing** the gradients for **all** other **trainable variables** in the neural network architecture.

Overestimation problem in classic DQN.

The Q-learning target **Rt + γ maxa Q(St +1, a)** contains a max operator. Q is **noisy**, which may be caused by environment, **non-stationarity**, function approx, or any other reasons. Note that the expectation of maximum noise is **<u>not less</u>** than the maximum expectation of noises, i.e., **E[max(1, . . . , n)] ≥ (max(E[1], . . . , E[n]))**. So the next Q-values are always overestimated.

Algorithm 3 DQN

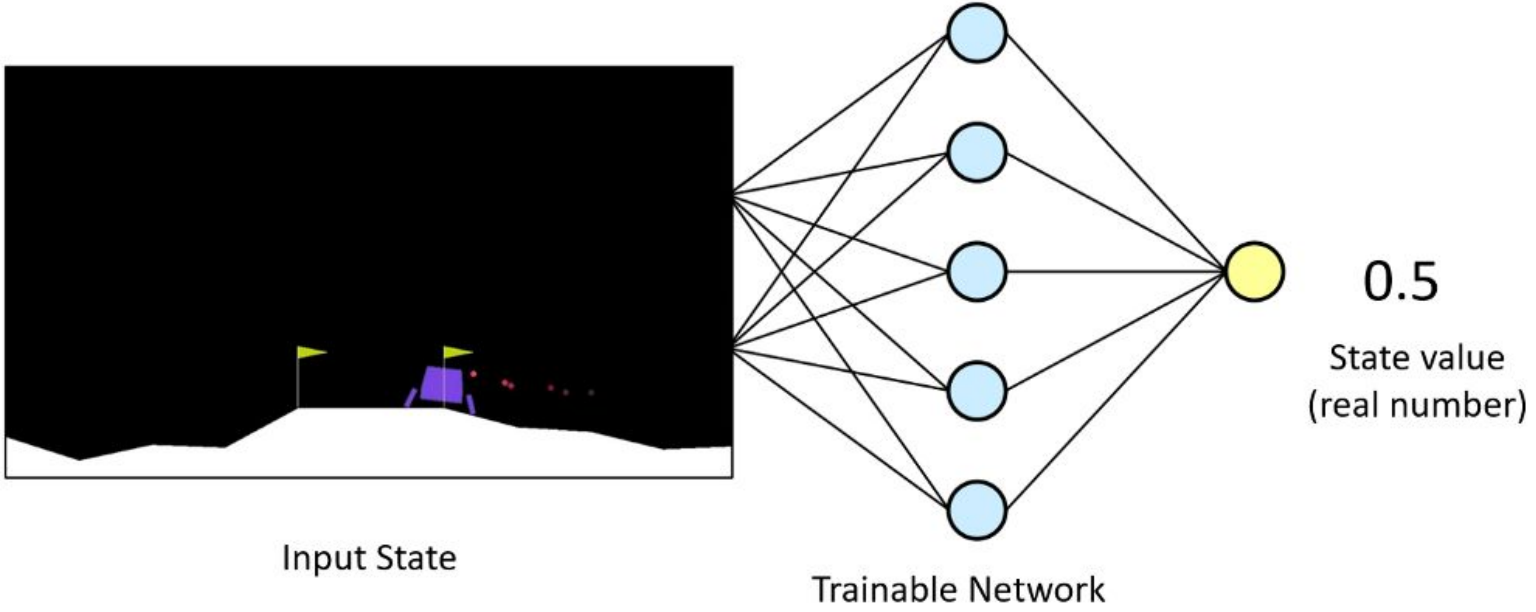
```
Hyperparameters: replay buffer capacity N, reward discount factor γ, delayed steps C for target action-value function update, ε-greedy factor ε
Input: empty replay buffer D, initial parameters θ of action-value function Q
Initialize target action-value function Q̂ with parameter θ̂ ← θ
for episode = 0, 1, 2, . . . do
    Initialize environment and get observation O_0
    Initialize sequence S_0 = {O_0} and preprocess sequence φ_0 = φ(S_0)
    for t = 0, 1, 2, . . . do
        With probability ε select a random action A_t, otherwise select A_t = arg max_a Q(φ(S_t), a; θ)
        Execute action A_t and observe O_{t+1} and reward R_t
        If the episode has ended, set D_t = 1. Otherwise, set D_t = 0
        Set S_{t+1} = {S_t, A_t, O_{t+1}} and preprocess φ_{t+1} = φ(S_{t+1})
        Store transition (φ_t, A_t, R_t, D_t, φ_{t+1}) in D
        Sample random minibatch of transitions (φ_i, A_i, R_i, D_i, φ'_i) from D
        If D_i = 0, set Y_i = R_i + γ max_{a'} Q̂(φ'_i, a'; θ̂). Otherwise, set Y_i = R_i
        Perform a gradient descent step on (Y_i − Q(φ_i, A_i; θ))^2 with respect to θ
        Synchronize the target Q̂ every C steps
        If the episode has ended, break the loop
    end for
end for
```

Improvement: Rods and cones

- **Double DQN** is an enhancement of DQN for reducing
- **Dueling DQN** For some states, different actions are not relevant to the expected value, and we do not need to learn the effect of each action for such states.
- **Prioritized Experience Replay (PER)** is a better sampling strategy for experience replay
- **Distributional Reinforcement Learning**
- **Multi-Step Learning**
- **Noisy Nets**
- **Rainbow**

	Double DQN	Prioritized replay	Dueling architecture	Multi-step learning	Distributional RL	Noisy nets
Biased targets	✓			✓	✓	
Overestimation bias	✓				✓	
Sample inefficiency		✓	✓			✓
Unstable training	✓		✓		✓	
Many actions	✓		✓			
Suboptimal architecture			✓		✓	✓
Suboptimal exploration					✓	✓
Many hyperparameters						✓

LunarLander-v2



References

Books:

[1] R. S. Sutton, A. G. Barto (2018) Reinforcement Learning (2nd ed.)

[2] Dimitri B. (2019) Reinforcement Learning and Optimal Control (1st ed.)

[3] Laura G., Wah L.K. (2020) Foundations of Deep Reinforcement Learning (1st ed.)

[4] Boris Belousov (2021) Reinforcement Learning Algorithms: Analysis and Applications (1st ed.)

[5] Alexander Zai Brandon Brown (2020) Deep Reinforcement Learning in Action (1st ed.)

[6] H.Dong, Z.Ding, S.Zhang (2020) - Deep Reinforcement Learning_ Fundamentals (1st ed.)

Articles :

[7] Ning, Brian, et al. "Double Deep Q-Learning for Optimal Execution." *ArXiv:1812.06600 [Cs, q-Fin, Stat]*, June 2020. *arXiv.org*, http://arxiv.org/abs/1812.06600.

[8] van Hasselt, Hado, et al. "Deep Reinforcement Learning with Double Q-Learning." *ArXiv:1509.06461 [Cs]*, Dec. 2015. *arXiv.org*, http://arxiv.org/abs/1509.06461.

[9] Moreno-Vera, Felipe. "Performing Deep Recurrent Double Q-Learning for Atari Games." *ArXiv:1908.06040 [Cs, Stat]*, Oct. 2019. *arXiv.org*, http://arxiv.org/abs/1908.06040.