Deep Reinforcement Learning (I)

Markov Decision Process and Deep Q-Learning

SUN Yinghan
2021. 12. 03

Outline

Part I: Markov Decision Process (MDP)

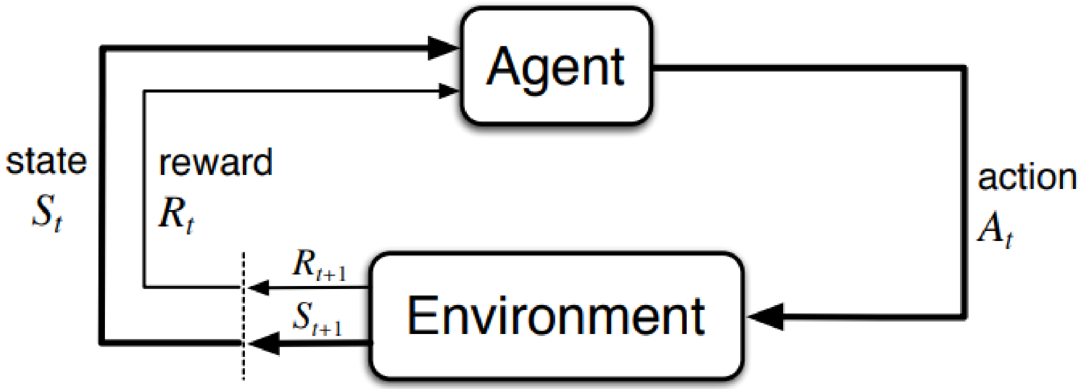
Part II: Q-Learning: Model-Free RL

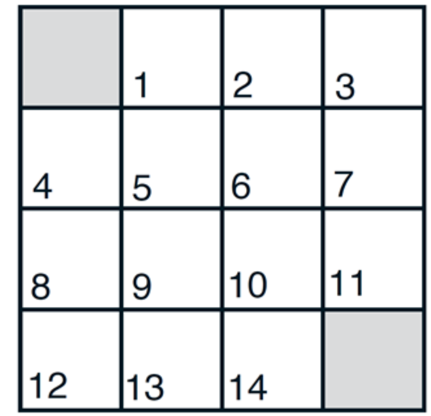
Part III: The 1st DRL Algorithm: Deep-Q Learning

Part IV: Further Discussion

Part I: Markov Decision Process (MDP)

Definition





Markov property: Current state completely characterizes the state of the world.

A Markov decision process is defined by a tuple: $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma >$.

- $\mathcal{S}$: set of possible states — State = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
- $\mathcal{A}$: set of possible actions — Action = {up, down, left, right}
- $\mathcal{R}$: distribution of reward — Reward = -1 for all transitions
- $\mathbb{P}$: transition probability — $p(s', r|s, a) = \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$
- γ: discount factor

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Policies and Value Functions

Policy: ways of acting. It is a mapping from states to probabilities of selecting each possible action.

Reinforcement learning methods specify how the agent's policy is changed as a result of its experience.

Example: equiprobable random policy

Value Function: the expected return when starting in *s* and following π thereafter.

State-value function for policy π: $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$
Action-value function for policy π: $q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$

Relationship:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma v_\pi(s')]$$

Objective: find the optimal policy, which maximizes cumulative discounted reward (value function).

Exploitation and Exploration

Greedy actions: the action whose estimated value is greatest in one step.

Exploitation: select one of the greedy actions. It is the right thing to do to maximize the expected reward on the one step.

Exploration: select one of the non-greedy actions. It may produce the greater total reward in the long run.

The need to balance exploration and exploitation is a distinctive challenge that arises in reinforcement learning.

ε-greedy policy: choose greedy actions most of the time, but every once, with small probability ε, randomly select an action from action space.

Bellman Optimality Equation

Bellman equation expresses a relationship between the value of a state and the values of its successor states.

$$
\begin{aligned}
q_\pi(s,a) &= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \\
&= \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')]
\end{aligned}
$$

The optimal action-value (Q-value) function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$
\begin{aligned}
Q^*(s,a) &= \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} Q^*(s',a')\right] \\
&= \mathbb{E}_{s'\sim\epsilon} \left[r + \gamma \max_{a'} Q^*(s',a') | S_t = s, A_t = a\right]
\end{aligned}
$$

Intuition: if the optimal state-action values for the next time-step Q*(s', a') are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s', a')$. **Value Iteration.**

The optimal policy: $\pi^* = \arg\max_a Q^*(s,a)$

Part II: Q-Learning: Model-Free RL

Monte Carlo Methods

Question: What if we do not know the environment?

Monte Carlo methods are ways of solving the reinforcement learning problem based on *averaging sample returns*. The only requirement is the *experience* — sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.

As more returns are observed, the average should converge to the expected value.

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Incremental Implementation

Recall: about the *expected return*
The state-value function for policy π is the *expected return* when starting in S=s and following π thereafter.
The action-value function for policy π is the *expected return* when starting from S=s, taking the action A=a, and thereafter following policy π.

Question: How these *expected returns* can be computed in a computationally efficient manner?

Solution: Incremental implementation

$$\begin{aligned}
V_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} G_i \\
&= \frac{1}{k+1}\left(G_{k+1} + \sum_{i=1}^{k} G_i\right) \\
&= \frac{1}{k+1}G_{k+1} + \frac{k}{k+1}V_k \\
&= V_k + \frac{1}{k+1}\left(G_{k+1} - V_k\right)
\end{aligned}$$

Q-Learning: A Temporal-Difference Learning Method

Question: What if some applications have very long episodes so that delaying all learning until the end of the episode is too slow?

Solution: Temporal-Difference Learning. Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to value functions, TD methods need to wait only until the next time step.

$$\text{MC Methods: } V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$
$$\text{TD Methods: } V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

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

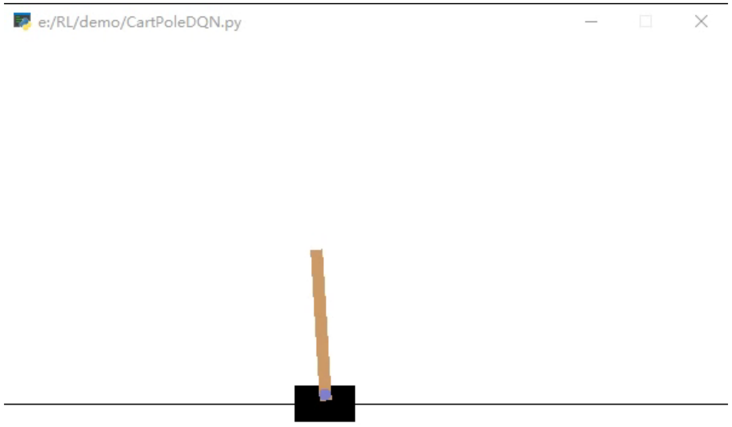
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

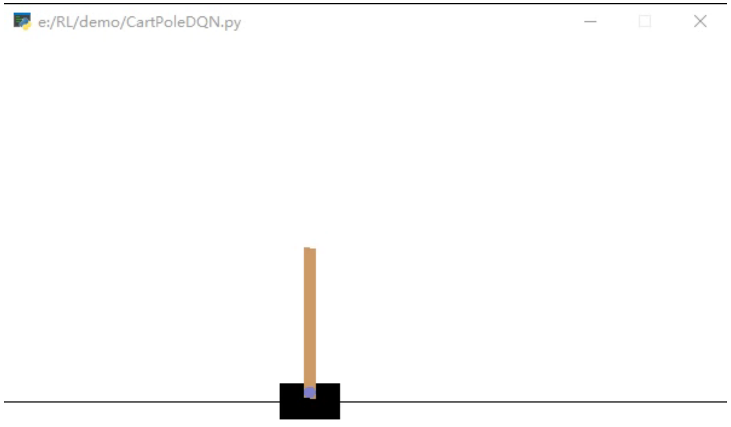
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
 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

Result: Q-Learning

```
Episode:24414    time:3.0499    step_num:183
Episode:24415    time:3.3165    step_num:199
Episode:24416    time:2.2831    step_num:137
Episode:24417    time:3.6677    step_num:220
Episode:24418    time:2.9493    step_num:177
Episode:24419    time:2.4158    step_num:145
Episode:24420    time:2.3654    step_num:142
Episode:24421    time:2.6506    step_num:159
Episode:24422    time:3.0648    step_num:184
Episode:24423    time:3.5821    step_num:215
Episode:24424    time:2.3838    step_num:143
```

```
Episode:24434    time:2.2160    step_num:133
Episode:24435    time:2.7504    step_num:165
Episode:24436    time:3.0823    step_num:185
Episode:24437    time:2.0835    step_num:125
Episode:24438    time:376.1777    step_num:22569
Episode:24439    time:1182.3005    step_num:70935
Episode:24440    time:3.5333    step_num:212
Episode:24441    time:2.4996    step_num:150
Episode:24442    time:2.4006    step_num:144
Episode:24443    time:2.6329    step_num:158
Episode:24444    time:2.7166    step_num:163
```





Part III: The 1st DRL Algorithm: Deep-Q Learning

From Q-Learning to Deep Q-Learning

Question: What if there are large number of states, or even infinitely many states?

Solution: Use a function approximator to estimate the action-value function.

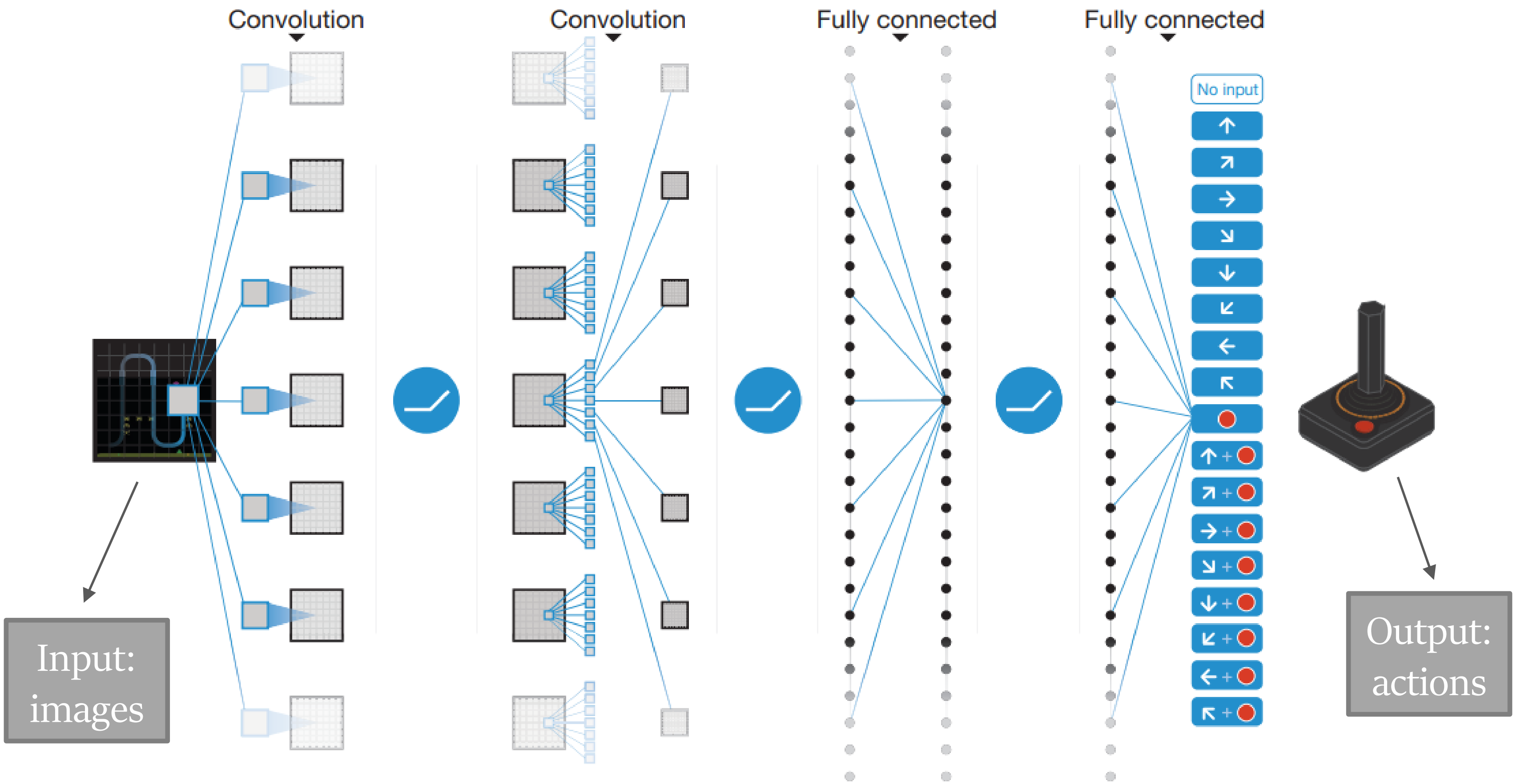
Example: Neural Network.

Universal Approximation Theorem: A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.

If the function approximator is a deep neural network, we call the algorithm as the **deep Q-learning**.

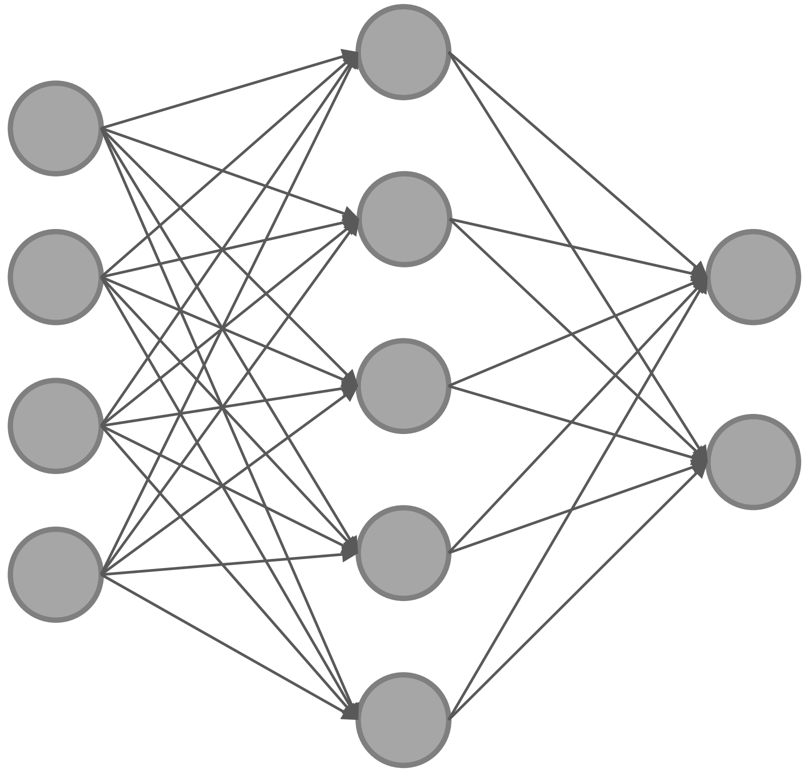
$$Q(s, a; \theta) \approx Q^*(s, a)$$

Value Function Approximator: Neural Network



Convolutional Neural Network (CNN)

Value Function Approximator: Neural Network



Multilayer Perceptron (MLP)

SGD: Solving for Optimal Policy

Loss Function:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a\right]$$

Backward Progress:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Question: The data is not independent and identically distributed.

Solution: Experience replay!

Experience Replay:

- Continually update a replay memory table of transitions (s, a, r, s') as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Algorithm: Deep Q-Learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

SGD: Solving for Optimal Policy

Loss Function:

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[\left(y_i - Q(s,a;\theta_i) \right)^2 \right]$$

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Backward Progress:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i) \right) \nabla_{\theta_i} Q(s,a;\theta_i) \right]$$

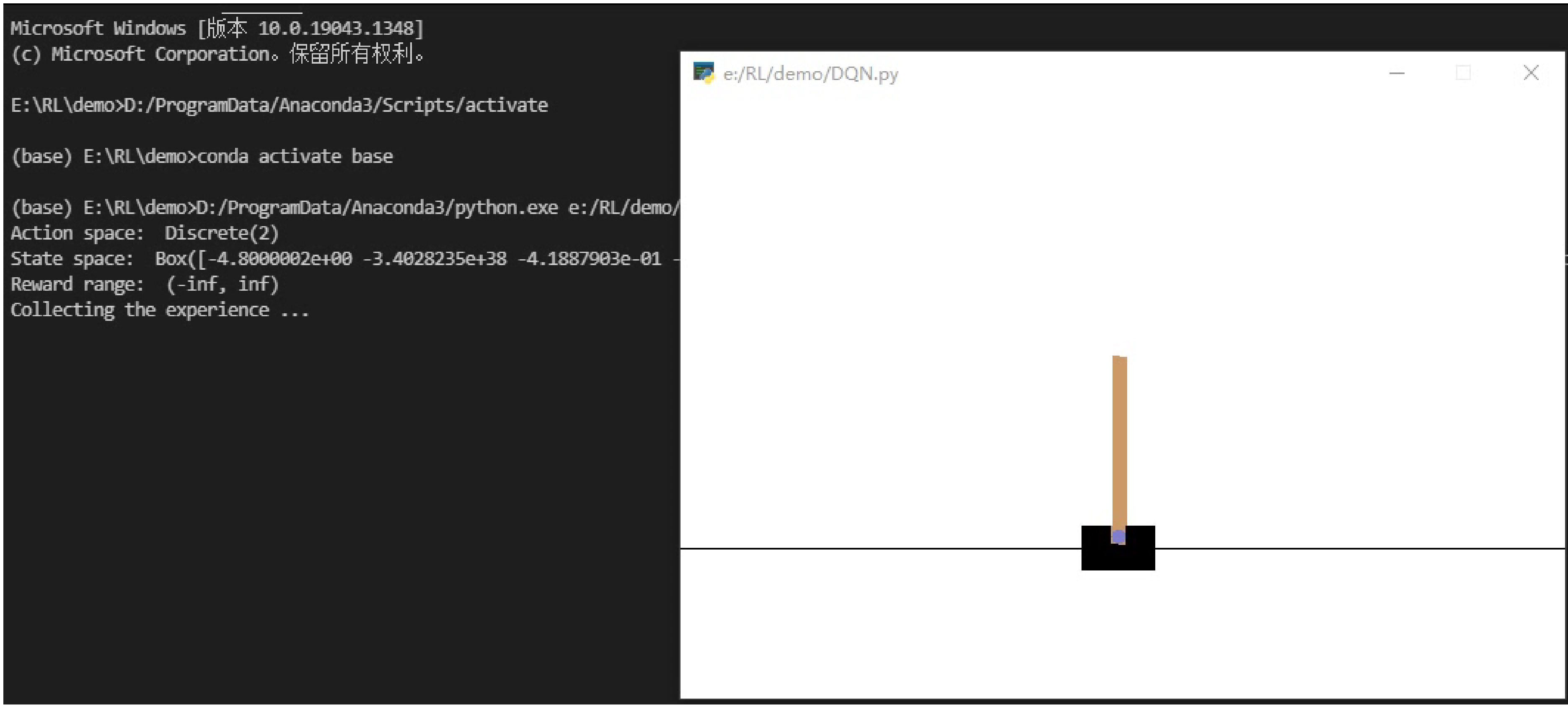
Question: The training progress is nonstationary! The target for Q(s, a) depends on the current weight θ.

Solution: Use a slow-moving “target” Q-network that is delayed in parameter updates to generate target value labels for the Q-network.

Algorithm: Deep Q-Learning with Experience Replay

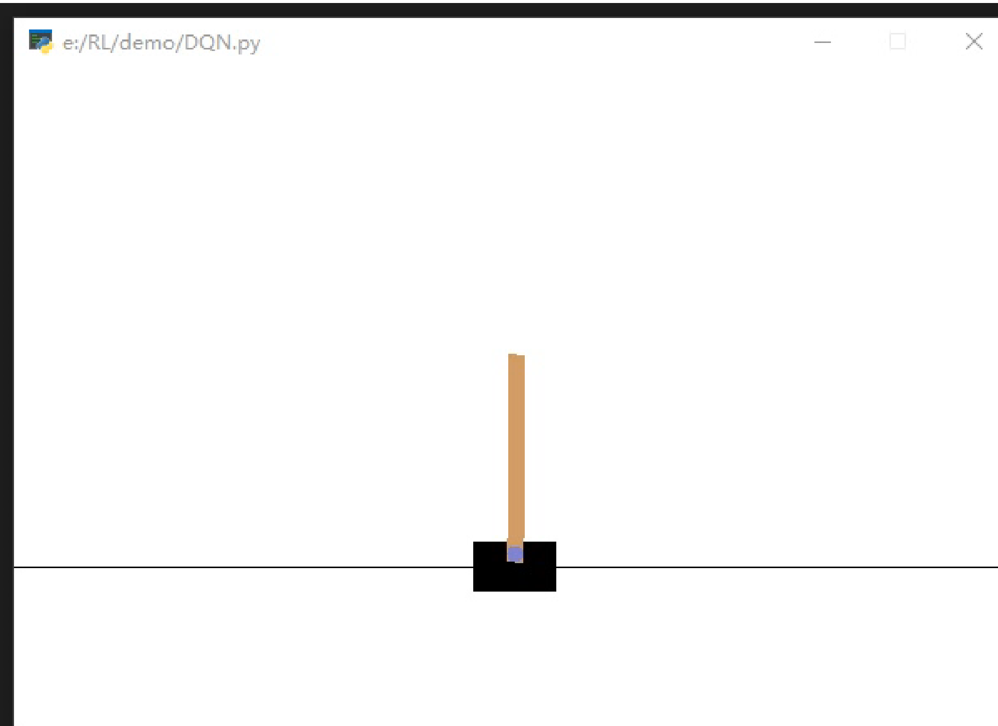
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Results: Cart-Pole Experient



Results: Cart-Pole Experient

```
Episode:  6195 | reward:  128.03 | time:  3.3328449726104736
Episode:  6196 | reward:  127.03 | time:  3.332977771759033
Episode:  6197 | reward:  121.66 | time:  3.3318655490875244
Episode:  6198 | reward:  132.07 | time:  3.3320820331573486
Episode:  6199 | reward:  126.39 | time:  3.332265853881836
Episode:  6200 | reward:  127.34 | time:  3.33249568939209
Episode:  6201 | reward:  130.19 | time:  3.332406997680664
Episode:  6202 | reward:  122.83 | time:  3.33284854888916
Episode:  6203 | reward:  134.29 | time:  3.3328135013580322
Episode:  6204 | reward:  131.08 | time:  3.3327932357788086
Episode:  6205 | reward:  127.49 | time:  3.333214044570923
Episode:  6206 | reward:  125.2 | time:  3.334056854248047
Episode:  6207 | reward:  132.25 | time:  3.331310749053955
Episode:  6208 | reward:  120.82 | time:  3.3326401710510254
Episode:  6209 | reward:  126.04 | time:  3.3326404094696045
Episode:  6210 | reward:  125.91 | time:  3.3324623107910156
Episode:  6211 | reward:  132.05 | time:  3.332202911376953
Episode:  6212 | reward:  130.37 | time:  3.332364082336426
Episode:  6213 | reward:  132.62 | time:  3.3322670459747314
Episode:  6214 | reward:  131.0 | time:  3.333185911178589
Episode:  6215 | reward:  131.54 | time:  3.332672595977783
Episode:  6216 | reward:  128.03 | time:  3.332085371017456
Episode:  6217 | reward:  124.42 | time:  3.3493564128875732
Episode:  6218 | reward:  131.34 | time:  3.3332676887512207
Episode:  6219 | reward:  128.4 | time:  3.332059383392334
Episode:  6220 | reward:  131.94 | time:  3.3332455158233643
Episode:  6221 | reward:  126.97 | time:  3.3317248821258545
Episode:  6222 | reward:  128.08 | time:  3.3321571350097656
Episode:  6223 | reward:  129.53 | time:  3.3322041034698486
Episode:  6224 | reward:  126.18 | time:  3.3494322299957275
Episode:  6225 | reward:  118.87 | time:  3.3316643238067627
Episode:  6226 | reward:  128.79 | time:  3.3328592777252197
Episode:  6227 | reward:  123.68 | time:  3.3324906826019287
Episode:  6228 | reward:  131.05 | time:  3.399315118789673
```

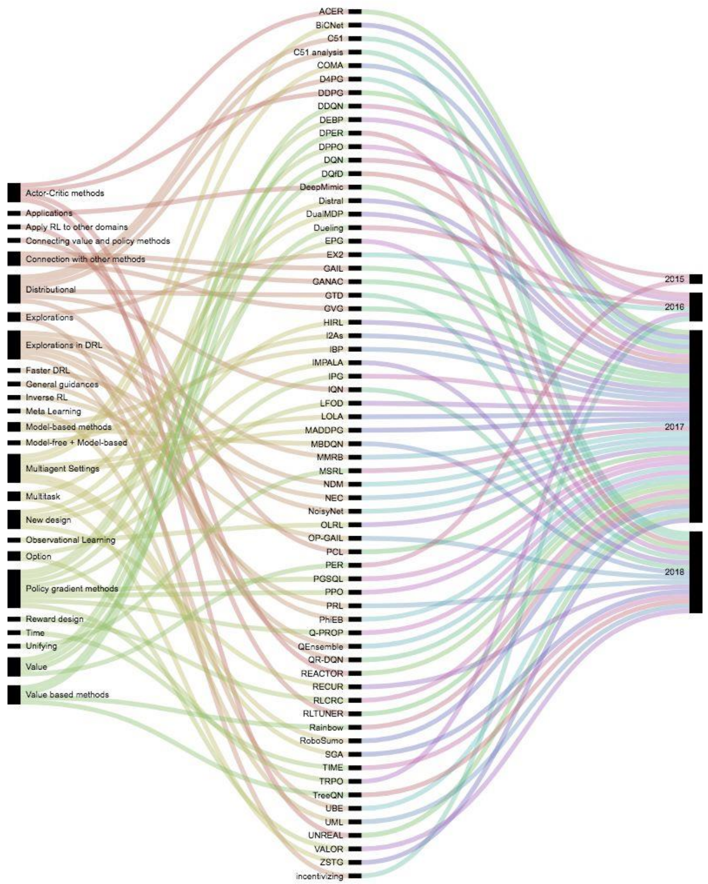


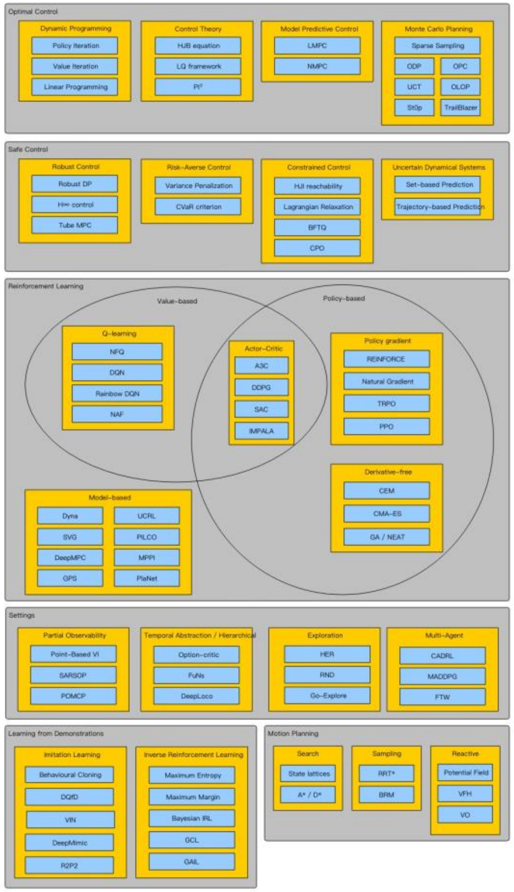
Part IV: Further Discussion

More RL Algorithms

Question: What if there are a large number of actions, or even infinitely many actions?

Solution: There are some other DRL algorithms can handle this situation.





References

[1] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.

[2] Bertsekas D. Reinforcement learning and optimal control[M]. Athena Scientific, 2019.

[3] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[4] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[5] Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay[J]. arXiv preprint arXiv:1511.05952, 2015.

Thanks for Listening