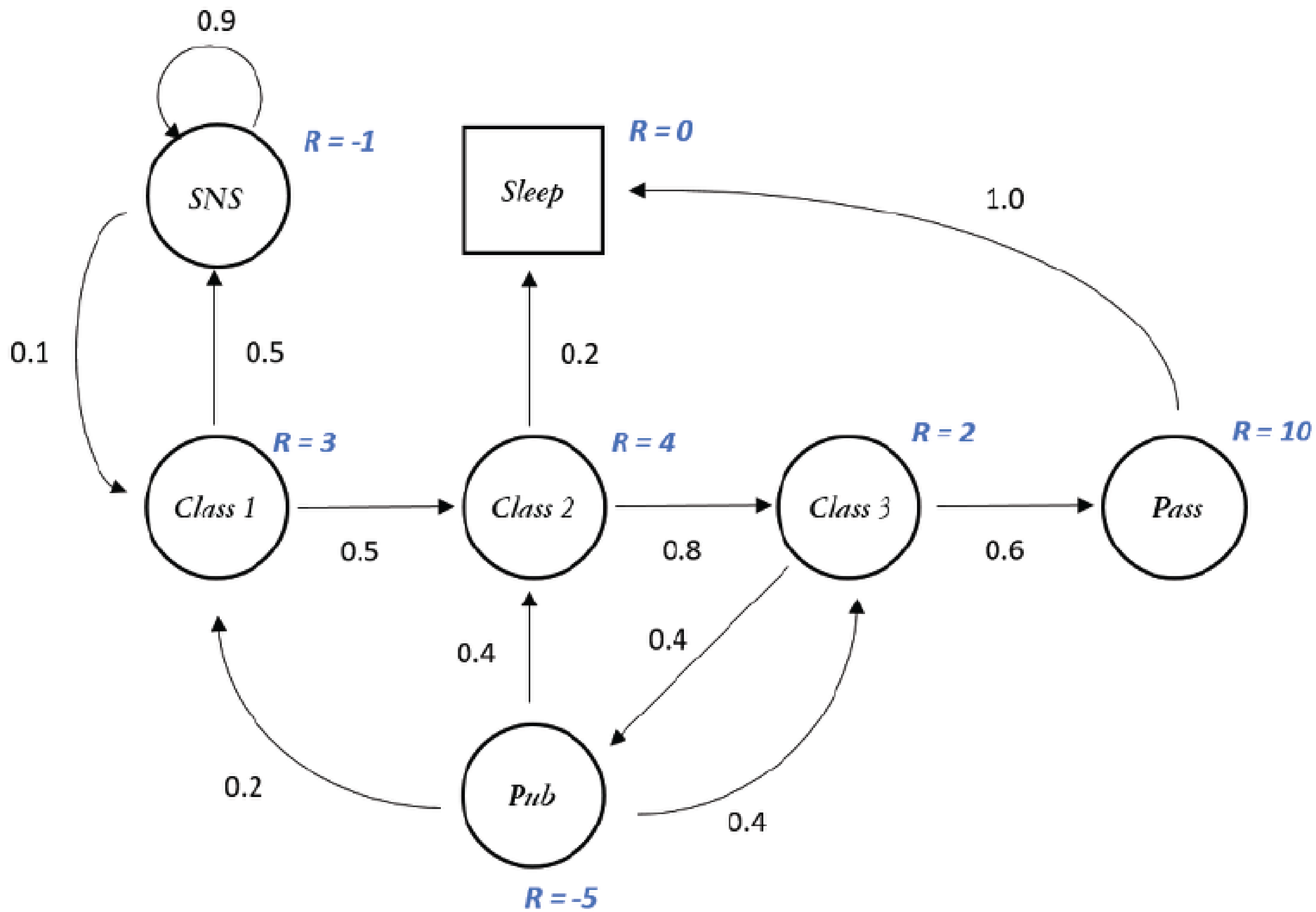
Introduction to RL – Part 2

TA. Bogyeong Suh

연세대학교 YONSEI UNIVERSITY

Markov Decision Process

- The process of RL is usually given as **Markov Decision Processes (MDPs)**, which is a mathematical framework used for modeling decision making in situations where the outcomes are partly random and partly under the control of a decision maker.

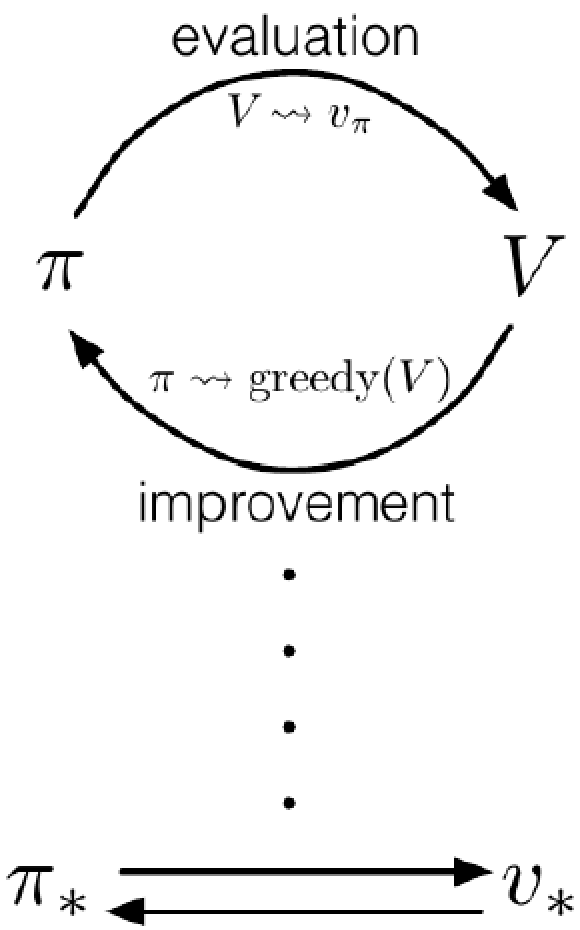


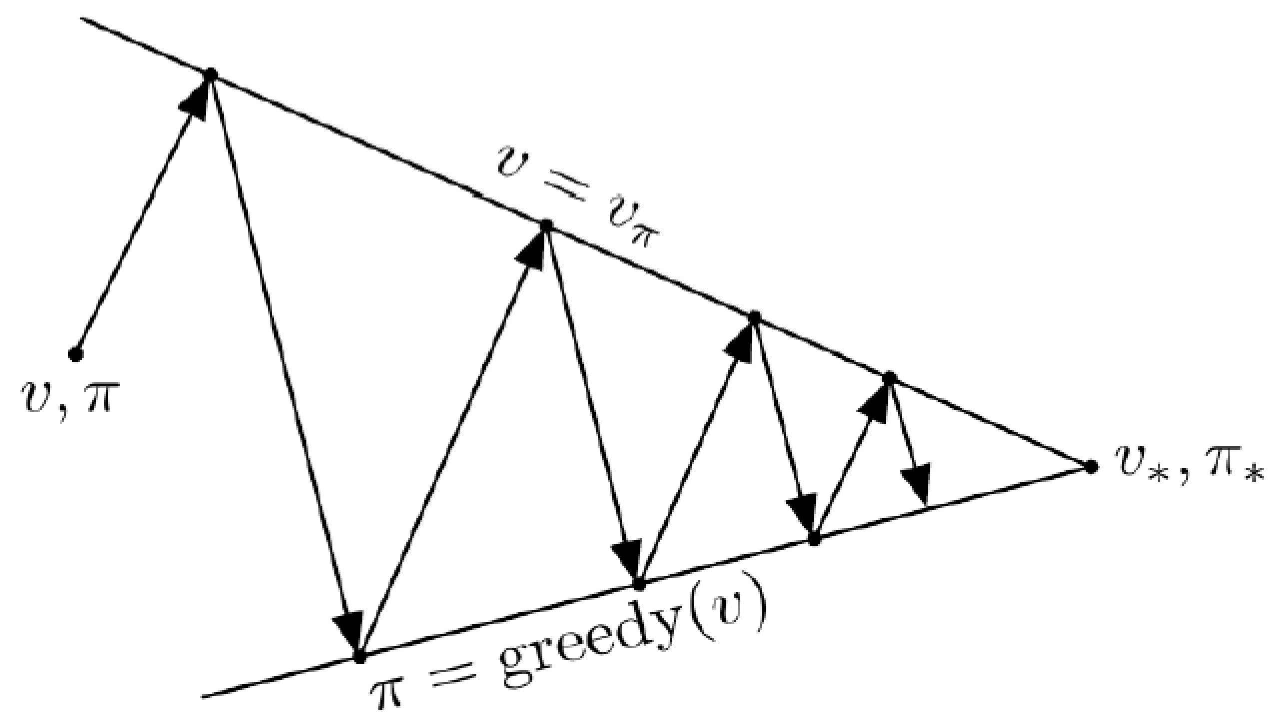
Markov Reward Process (MRP)

Policy Iteration

1) Policy Evaluation

-Given an arbitrary policy, we calculate value function V for **all states** under this policy. We iterate through each state and update V until it converges. ➔ We find true value function with iteration





Value Iteration

k = 0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

at (1,2) state : state 1

Up $V_1(s) = -1 + 0$
Down $V_1(s) = -1 + 0$
Left $V_1(s) = -1 + 0$
Right $V_1(s) = -1 + 0$

$\therefore V_1(1) = \max V_1(s) = \mathbf{-1}$

k = 1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,2) state : state 1

Up $V_2(s) = -1 +(-1)$
Down $V_2(s) = -1 +(-1)$
Left $V_2(s) = -1 +(0)$
Right $V_2(s) = -1 +(-1)$

$\therefore V_2(s) = \max V_2(s) = \mathbf{-1}$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,3) state : state 2

Up $V_2(s) = -1 +(-1)$
Down $V_2(s) = -1 +(-1)$
Left $V_2(s) = -1 +(-1)$
Right $V_2(s) = -1 +(-1)$

$\therefore V_2(s) = \max V_2(s) = \mathbf{-2}$

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-1
-2	-2	-1	0

k = 2

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

k = 3

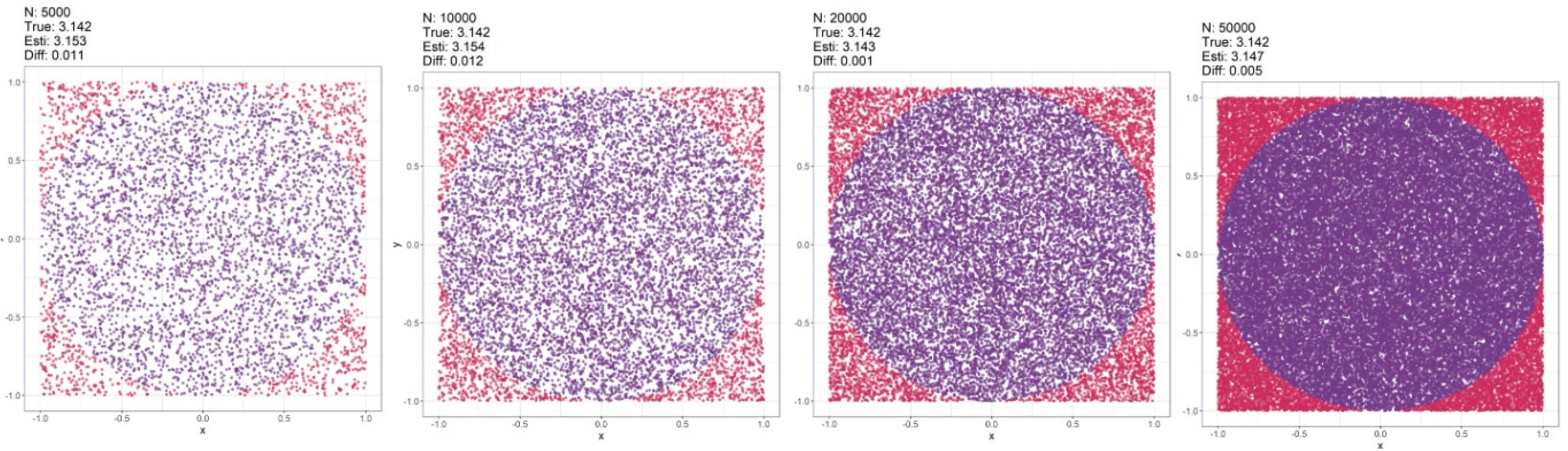
…

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

k = ∞

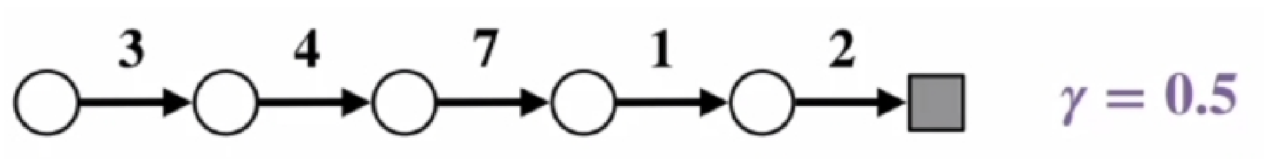
Monte Carlo method

- Generally, MC is a method that relies on **repeated random sampling** to obtain numerical results
- MC methods learn directly from episodes of **experience**
- In MC, we learns from *complete* episodes: no bootstrapping
- MC is *model-free*: no knowledge of MDP transitions / rewards



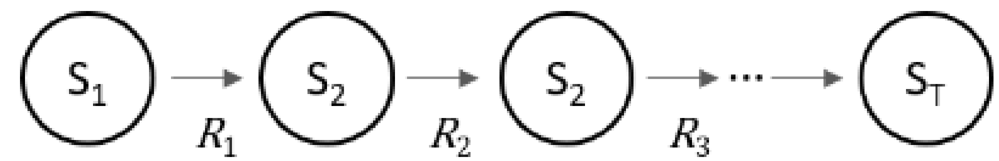
e.g. Finding the area of a circle with radius=1 using random samples of (x,y)

Monte Carlo method



Example of MC prediction

$$
\begin{aligned}
G_0 &= R_1 + \gamma G_1 \\
G_1 &= R_2 + \gamma G_2 \\
G_2 &= R_3 + \gamma G_3 \\
G_3 &= R_4 + \gamma G_4 \\
G_4 &= R_5 + \gamma G_5 \\
G_5 &= 0
\end{aligned}
\quad \rightarrow \quad
\begin{aligned}
G_5 &= 0 \\
G_4 &= R_5 + \gamma G_5 = 2 + 0.5 * 0 = 2 \\
G_3 &= R_4 + \gamma G_4 = 1 + 0.5 * 2 = 2 \\
G_2 &= R_3 + \gamma G_3 = 7 + 0.5 * 2 = 8 \\
G_1 &= R_2 + \gamma G_2 = 4 + 0.5 * 8 = 8 \\
G_0 &= R_1 + \gamma G_1 = 3 + 0.5 * 8 = 7
\end{aligned}
$$



$$
\begin{aligned}
G(s_1) &= R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots \\
G(s_2) &= R_2 + \gamma^2 R_3 + \cdots \\
G(s_3) &= R_3 + \cdots
\end{aligned}
$$

<Fig 2. The returns of each states in 1 episode>

- Goal: learn a policy that maxmimizes the total cumulative reward it receives over time
- Transition probability: $p(s', r|s, \pi(s))$
- State Value function: $\boldsymbol{V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]}$ **→Average values of returns from episodes**
- Action value function: $\boldsymbol{q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]}$

Monte Carlo method

```
1. Initialization
Initialize:
        π(s) ∈ A(s), ∀s ∈ 𝕊
        Q(s,a) ∈ ℝ, ∀s ∈ 𝕊, ∀a ∈ A(s)
        Returns(s,a) ← empty list, ∀s ∈ 𝕊, ∀a ∈ A(s)
2. Monte Carlo
Loop forever(for each episode):
        Choose S_0 ∈ 𝕊, A_0 ∈ A(S_0), randomly such that all pairs have probability> 0
        Generate an episode from S_0, A_0 following π: S_0, A_0, R_1, ⋯, S_{T−1}, A_{T−1}, R_T
        G ← 0
        Loop for each step of episode, t = T − 1, T − 2, ⋯, 0
                G ← γG + R_{t+1}
                Unless the S_t, A_t appears in S_0, A_0, S_1, A_1, ⋯, S_{t−1}, A_{t−1}:
                        Append G to Returns(S_t, A_t)
                        Q(S_t, A_t) ← Average(Returns(S_t, A_t))
                        π(S_t) ← argmax_a Q(S_t, a)
```

- Using randomly generated episodes, calculate action value function from cumulative reward G

$$q_\pi(s,a) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i(s)$$

- $N(s)$: number of times we visited state s we from start to end in total episodes
- $G_i(s)$: return of state s in episode i

Comparison of DP and MC

- **Dynamic Programming**

  - Pros: **bootstrapping** (update value estimates based on other learned estimates, without waiting for a final outcome)

  - Cons: **Model-Based** (requires complete and accurate model of the environment)

- **Monte Carlo method**

  - Pros: **Model-free** (learn directly from experience)

  - Cons: applicable to **episodic tasks** (require episodes to end to calculate returns, making them unsuitable for continuous tasks)

➔ **Temporal Difference Learning (TD) takes advantages from DP and MC**; It is **model-free** (uses random sampling), which learns directly from raw experience without a model of the environment, and **bootstraps** by updating estimates based in part on other learned estimates, without waiting for a final outcome.

Temporal Difference Learning

Value function

$$V_\pi(s) = \mathbb{E}_\pi\left[\, G_t \mid S_t = s \,\right]$$

Using Monte Carlo method

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

The episode should be finished to get the return G_t.

Total expected return G_t

$$G_t = \sum_{i=t+1}^{T} \gamma^{i-t-1} R_i$$
$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$$
$$= R_{t+1} + \gamma G_{t+1}$$

State value function

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= R_{t+1} + \gamma v_\pi(S_{t+1}) \end{aligned}$$

Temporal Difference (TD) Learning

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD Target}} - V(S_t)]$$

TD Target

TD Error

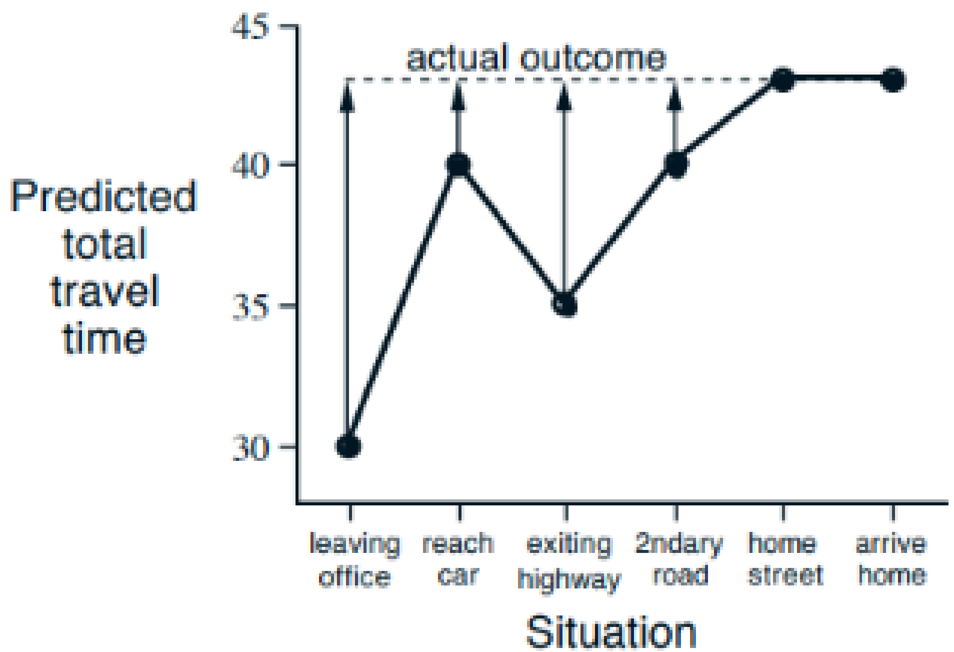
Estimated value of the current state

- It updates the value of the current state towards the estimated return

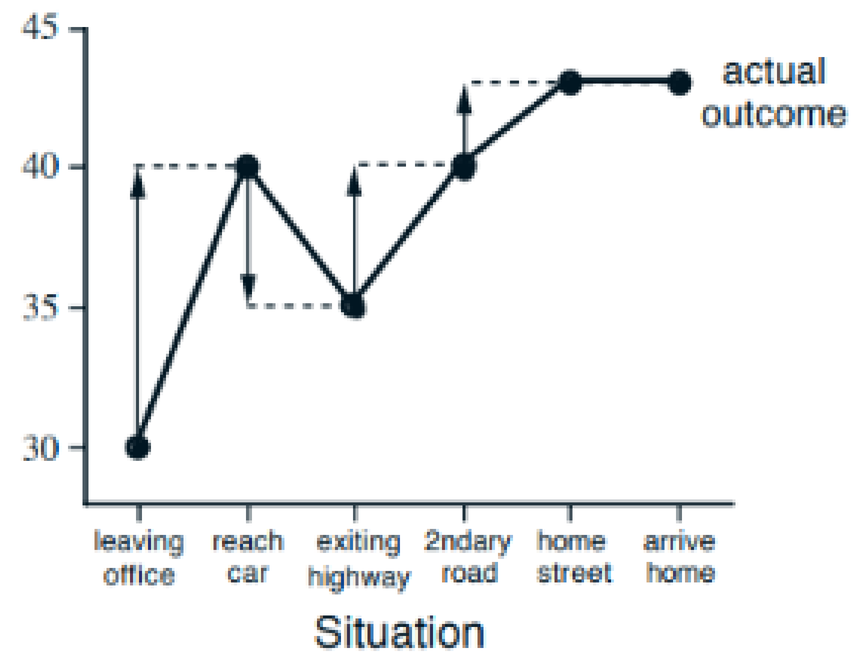
- TD error: measures the difference between the predicted value of the current state and the observed reward plus the estimated value of the next state

Temporal Difference Learning

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43



Monte-Carlo method



Temporal Difference learning

SARSA

-On-policy TD control

$$S_t, A_t, R_{t+1}, S_{t+1}, \color{red}{A_{t+1}}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\underbrace{\left(R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q(S_t, A_t)\right)}_{\text{TD Error}}$$

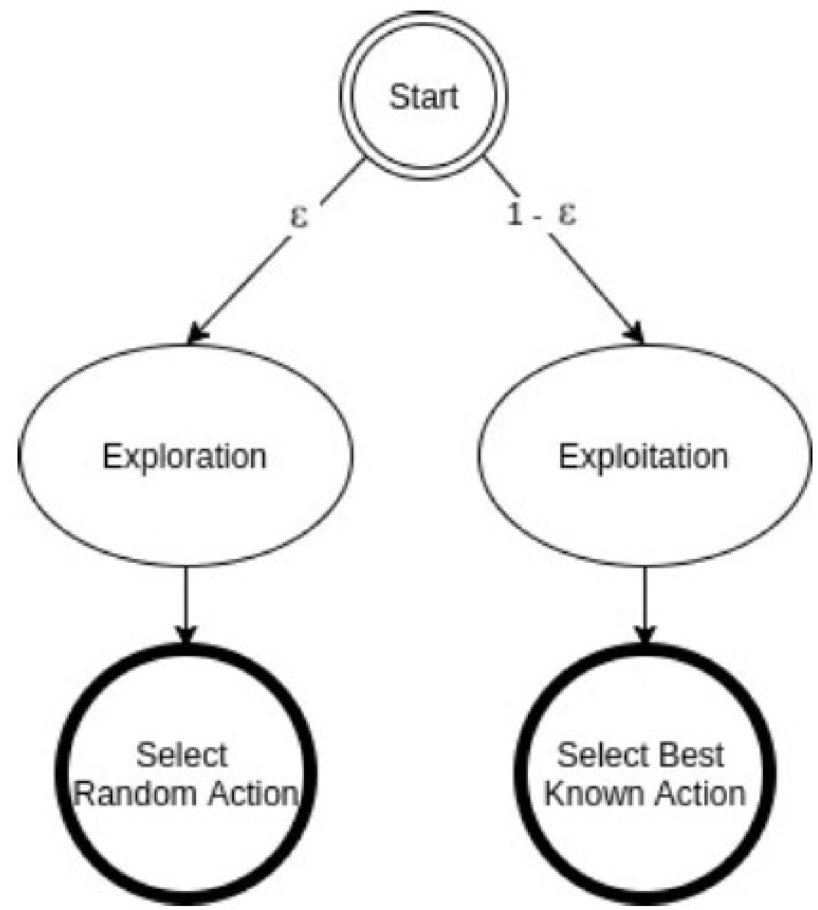
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

$\epsilon-$greedy

$$A \leftarrow \begin{cases} \arg\max_a Q(S, A) & \text{with probability } 1 - \epsilon \\ \text{a random action} & \text{with probability } \epsilon \end{cases}$$



$\epsilon-$greedy

Q-learning

-Off-policy TD control

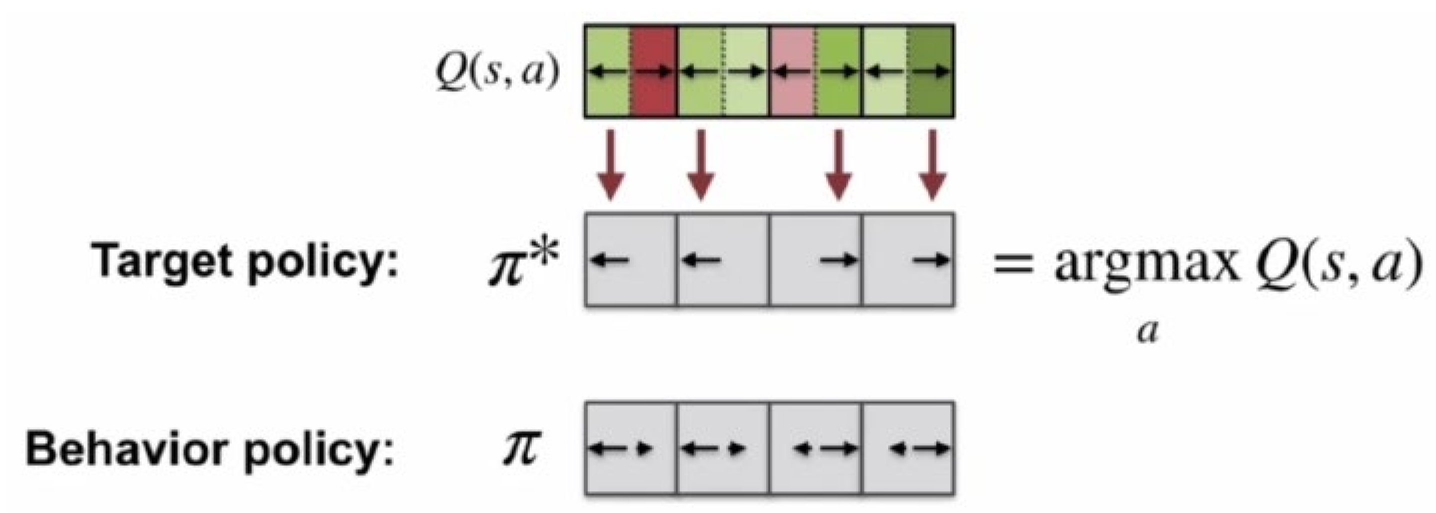
Update rule in SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q(S_t, A_t)\right)$$

Update rule in Q-learning

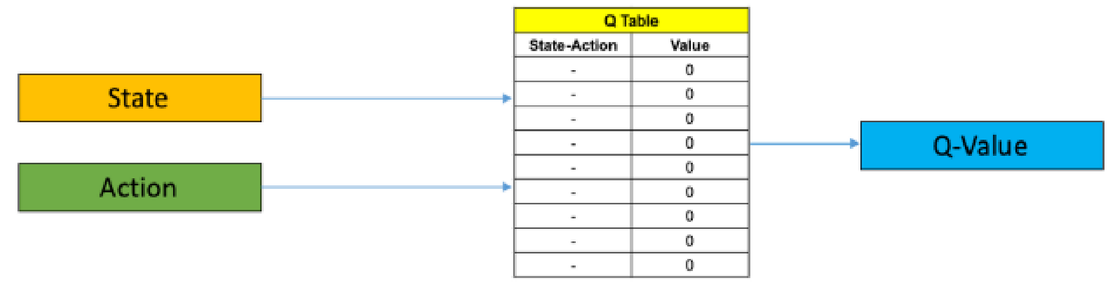
From π_*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \underbrace{\max_{a'} Q(S_{t+1}, a')}_{\text{Target policy } \pi_*} - Q(S_t, A_t)\right)$$

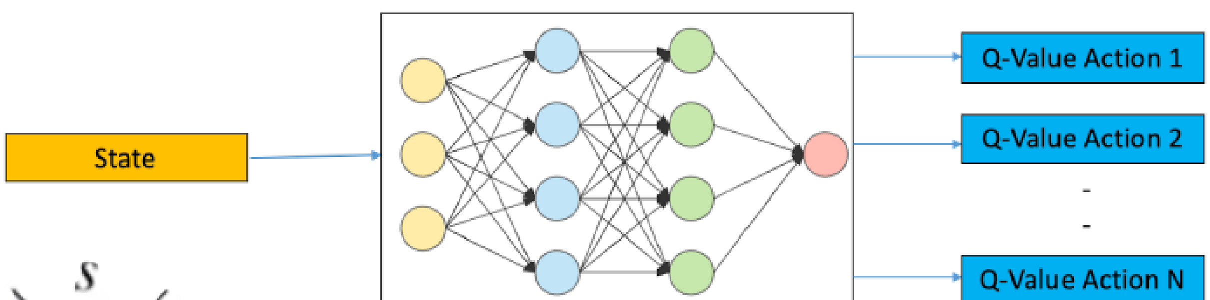


Target and behavior policies

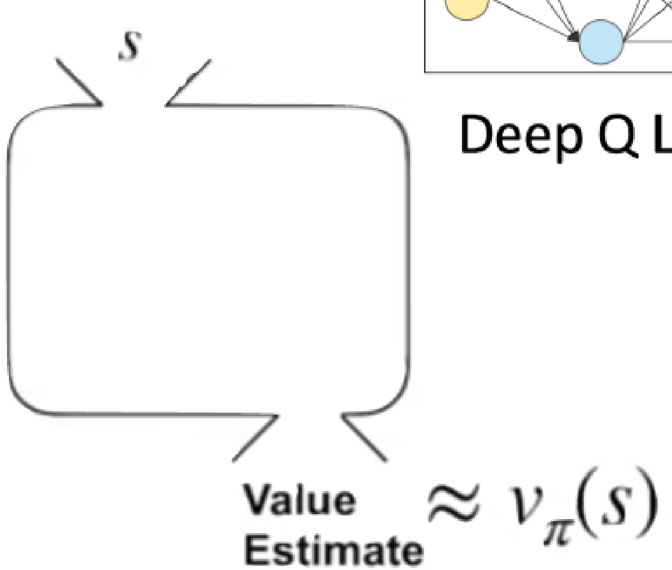
Q-learning



Q Learning



Deep Q Learning



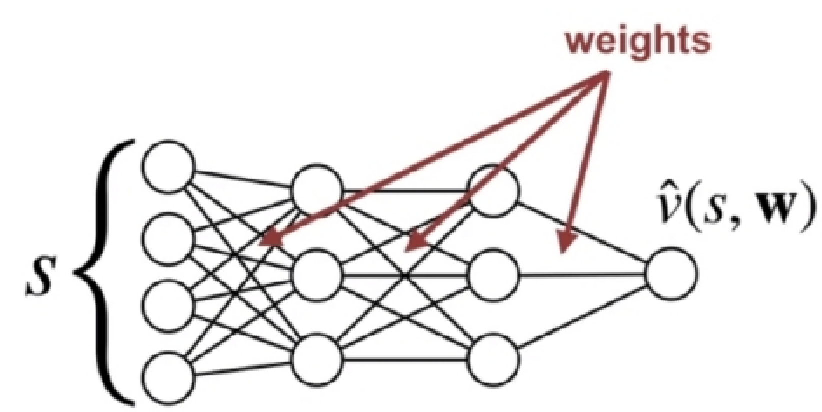
Parameterized value function

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

Linear value function approx.

$$\hat{v}(s, \mathbf{w}) \doteq \sum w_i x_i(s)$$
$$= < \mathbf{w}, \mathbf{x}(s) >$$

Nonlinear value function approx. w/ NN



Neural Network

Deep Q-Networks (DQN)

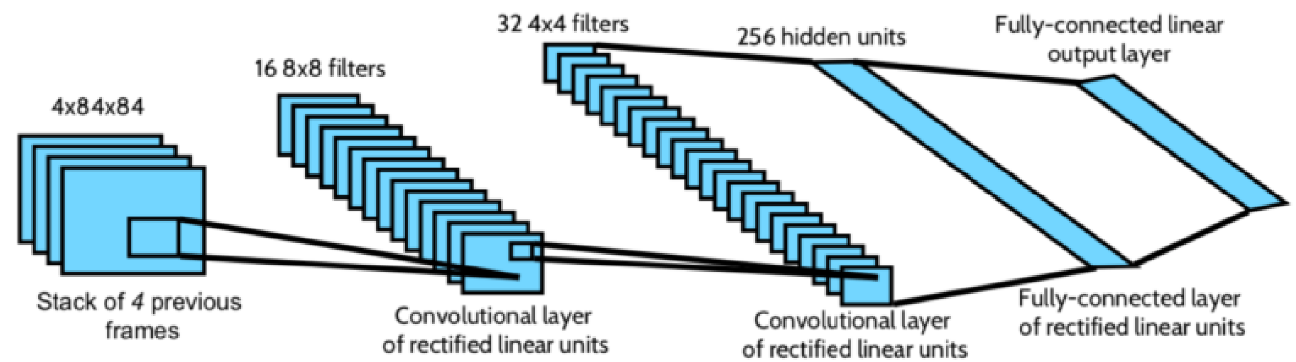
Replay buffer

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow$ s, a, r, s'

$s_t, a_t, r_{t+1}, s_{t+1}$ $\rightarrow$

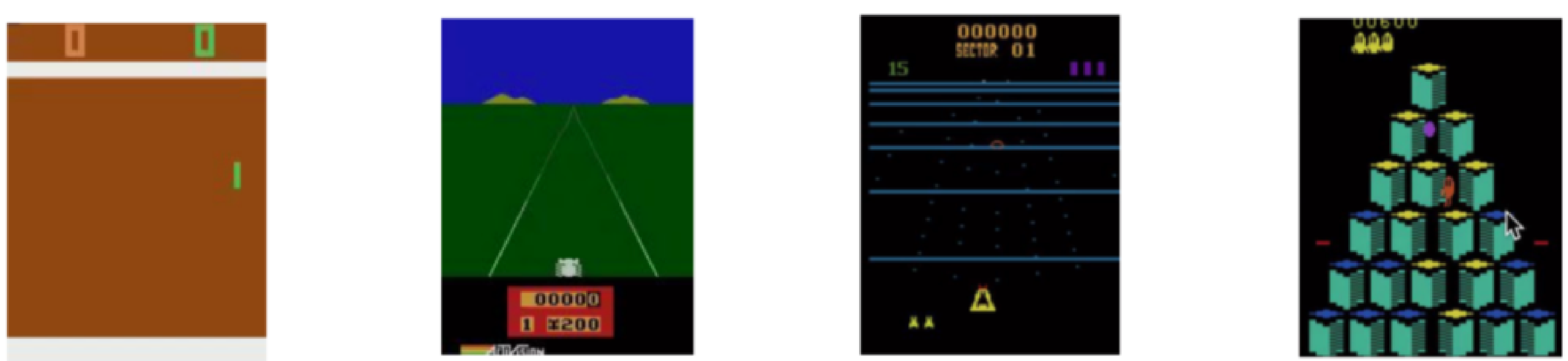
To remove correlations btw samples.



DQN on Atari (Network Architecture)

Update with fixed target network θ^-

$$l = \left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2$$

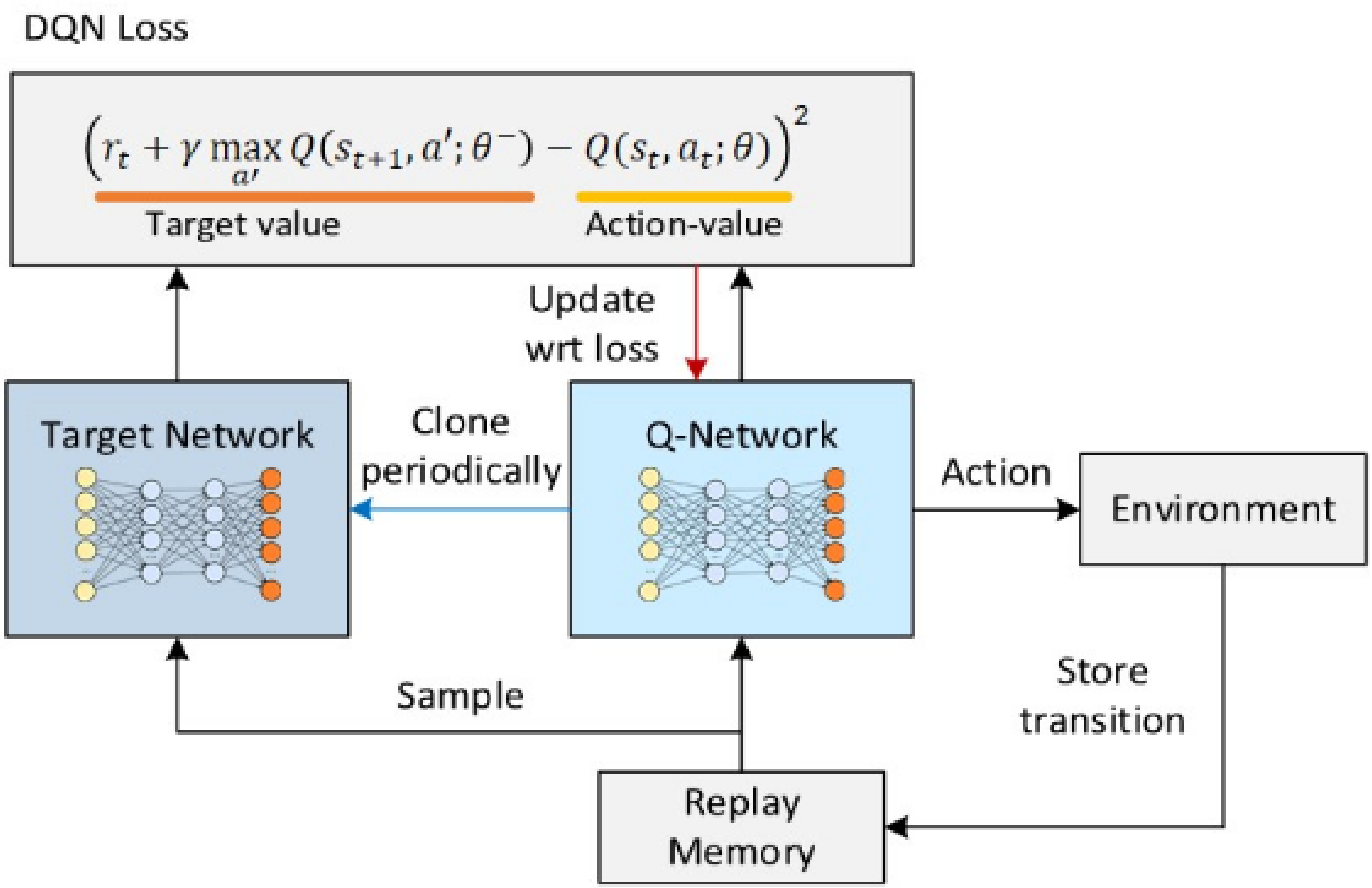


Pong Enduro Beamrider Q*bert

Atari games

Deep Q-Networks (DQN)

- Target network



Policy Gradient Theorem

Policy

$$\pi(a \mid s) = Pr(A_t = a \mid S_t = s)$$

$$\downarrow$$

$$\pi_\theta$$

Reward function (policy objective function)

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a \mid s) Q^\pi(s, a)$$

Stationary distribution (d^π)

$$d^\pi(s) = \lim_{t \to \infty} P(s_t = s \mid s_0, \pi_\theta)$$

Starting from s_0, a probability that the state becomes s.
(under the policy π_θ)