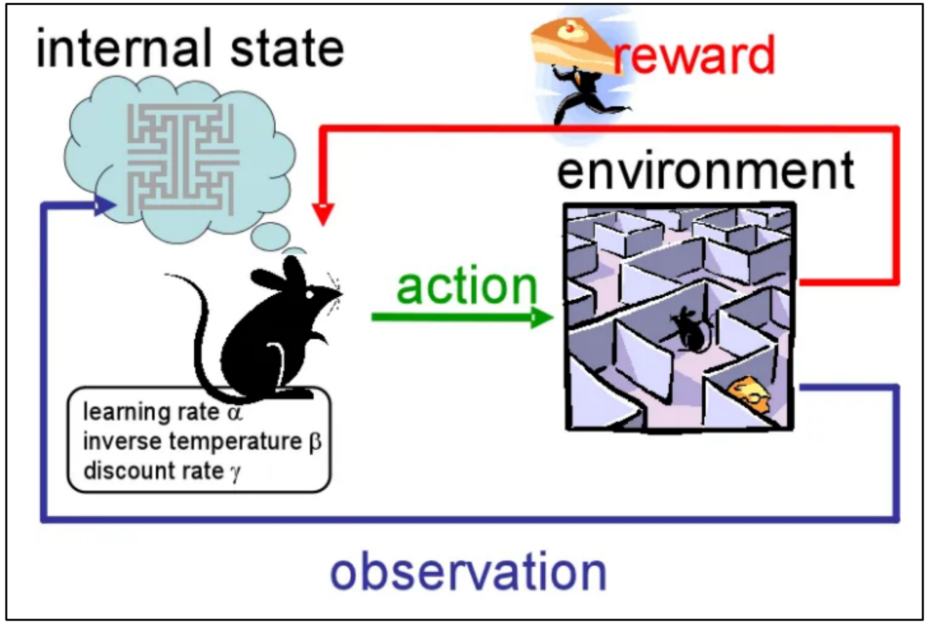
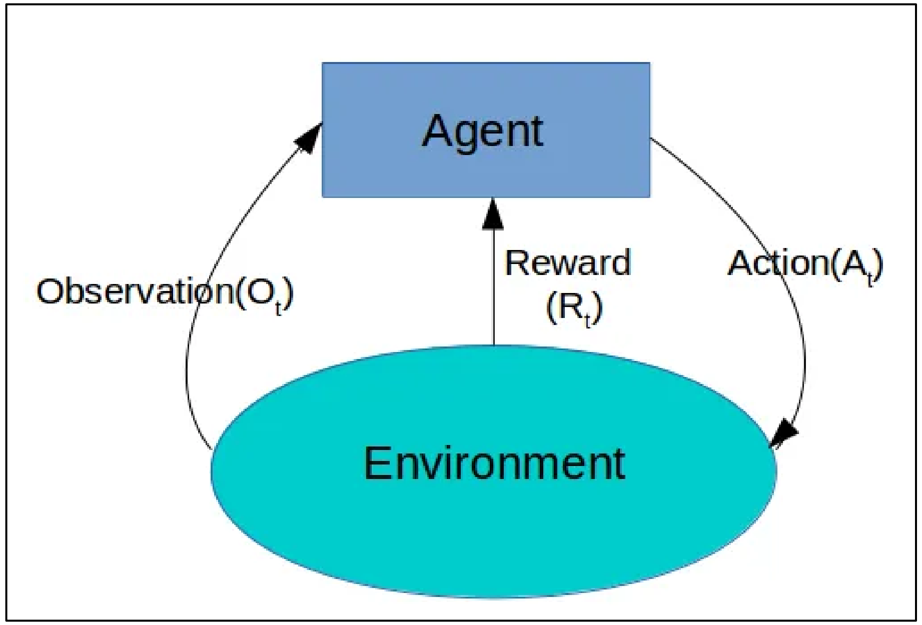
Introduction to Reinforcement Learning

Jaewoong Han

Machine Learning &
Control Systems Lab.

연세대학교
YONSEI UNIVERSITY

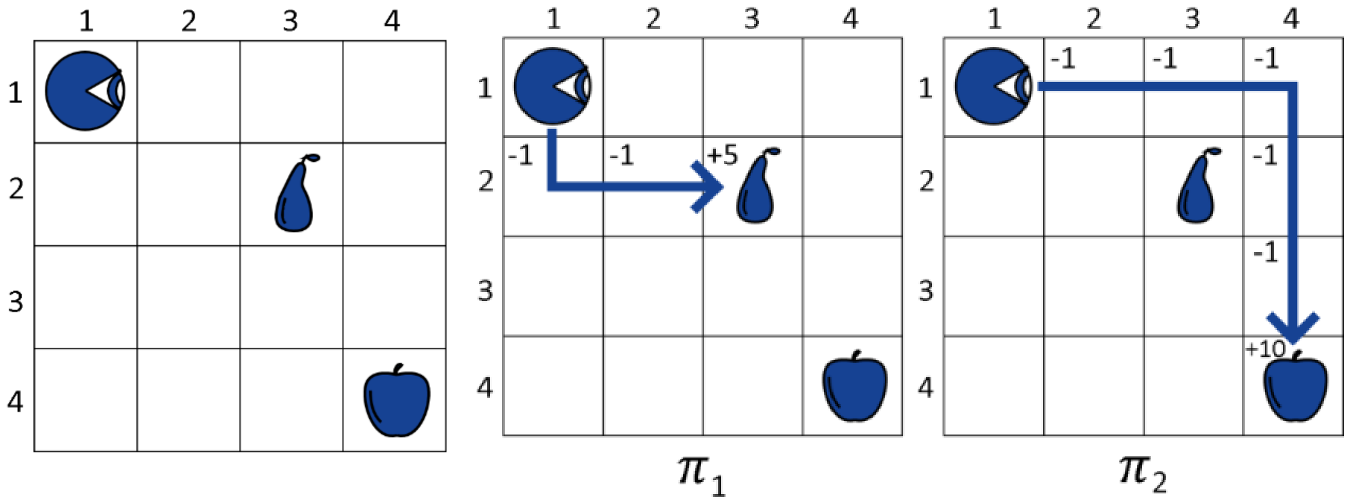




- Goal-oriented Learning, where the primary objective is to **train models to make sequences of decisions by discovering strategies that maximize a reward signal**
- Two primary entities in RL are **agent** and **environment**. Agent learns and make decisions, and environment is where the agent operates.
- Agent interacts with the environment, takes an action based on a policy, receives a reward and observes the next state.

1. Introduction to Reinforcement Learning

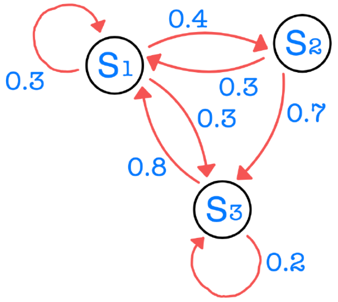
- State s: state space
- Action a: action space
- Reward r: reward function
- Policy π: A strategy or mapping from states to actions. Defines the agent's behavior.



- States $s_0 = (1,1)$
- Action a: Up, Down, Left, Right
- Reward r: No fruit (-1), Pear $(+5)$, Apple $(+10)$
- Policy π_1=down, right, right, π_2=right, right, right, down, down, down

2. Markov property

- Stochastic or random process is a collection of random variables indexed by a time set.
  - discrete time random process $S_0, S_1, \dots, S_{t-1}, S_t, S_{t+1}, \dots$
  - continuous time random process $\{S_t | t \geq 0\}$



$$
\begin{array}{c|ccc}
 & S_1 & S_2 & S_3 \\
\hline
S_1 & & & \\
S_2 & & \begin{bmatrix} 0.3 & 0.4 & 0.3 \\ 0.3 & 0.0 & 0.7 \\ 0.8 & 0.0 & 0.2 \end{bmatrix} & \\
S_3 & & &
\end{array}
$$

- Stochastic process $\{S_t\}$ is a Markov process (or Markov chain) if holds Markov property

$$P(S_{t+1} = s' | S_t = s) = P(S_{t+1} = s' | S_0 = s_0, S_1 = s_1, \dots, S_t = s_t)$$

"Given the present state $S_t = s$, the future state $S_{t+1} = s'$ does not depend on the past states."

* Brownian motion is a famous Markov process

- $P(S_{t+1} = s' | S_t = s)$ is called the state transition probability from state s to state s'.
- Markov process is a tuple (S, P)

S: a (finite) set of states

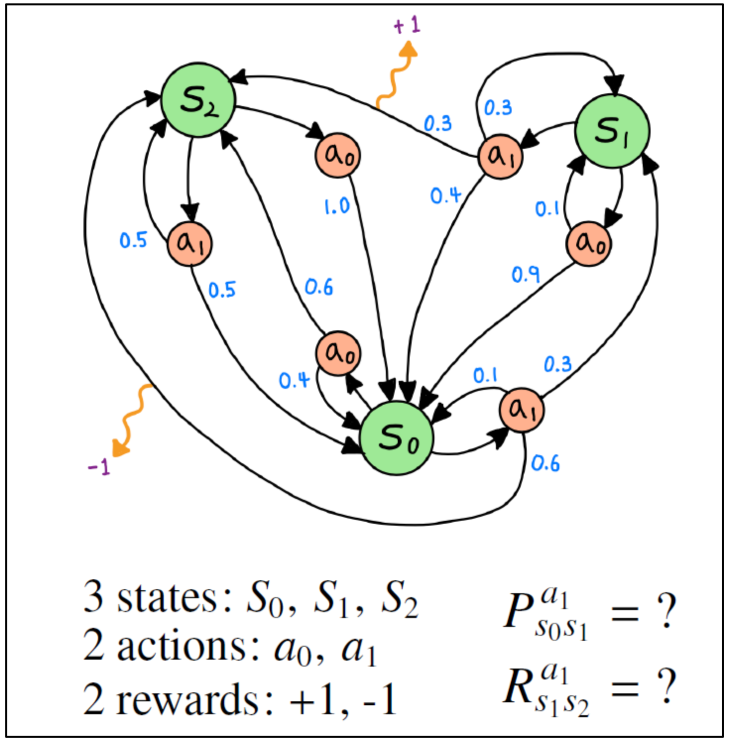
P: state transition probability matrix $[P_{ij}]$

$P_{ij} = P_{s_i s_j} = p(s_j | s_i) = P(S_{t+1} = s_j | S_t = s_i)$

where the sum of entries in each rows is 1

2. Markov Decision Process

- MDP is a tuple (S, A, P, R, γ) where state has Markov property.
  - S: state space
  - A: action space
  - P: (state) transition probability from s to s′ given a
    $P^a_{ss'} = p(s'|s, a) = P(S_{t+1} = s' \mid S_t = s, A_t = a)$
  - R: reward function
    - $R^a_{ss'}$ is the immediate reward received after transitioning from s to s' given a
  - $\gamma \in [0, 1]$: Discount factor



- Model-based RL vs Model-free RL
  - Model-based RL: Known MDP (Known P, R)
  - Model-free RL: Unknown MDP

3. Concepts – Reward

- Reward R_t is a scalar feedback indicating how well the agent is doing at step t.
- The agent's job is to maximize the cumulative sum of rewards.
- Reinforcement Learning is based on the Reward Hypothesis

[Reward Hypothesis]

All goals can be described by the maximization of the expected value of the cumulative sum of rewards.

- Under known dynamics $p(s', r|s, a)$ of all transitions (s, a, s', r), one can compute the followings.
  - State transition probability
    $P_{ss'}^{a} = p(s'|s,a) = P(S_{t+1} = s'|S_t = s, A_t = a) = \sum_{r\in R} p(s', r|s, a)$
  - Expected reward for state-action pair
    $R_s^a = r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in R} r \sum_{s'\in S} p(s', r|s, a)$
  - Expected reward for state-action-next_state triple
    $R_{ss'}^{a} = r(s,a,s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r\in R} r\, p(s',r|s,a)}{p(s'|s,a)}$

3. Concepts – Return

- Return G_t is the total discounted reward from time step t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Discount $\gamma \in [0, 1]$ is used to compensate for the effect of immediate and future rewards.
- Most MDPs are discounted. Why?
  - Mathematically convenient (avoiding infinite return)
  - Uncertainty of the future (values of rewards decay exponentially)
  - In practice, immediate rewards may earn more interest than delayed rewards.
  - Sometimes, if all sequences are terminated, use undiscounted.

3. Concepts – Policy

- (Stochastic) Policy π is a probability distribution over actions for given states.

  $\pi(a|s) = P(A_t = a|S_t = s)$

  * Deterministic policy is $\pi(s) = a$

- Policy π provides the guideline on what is the optimal action a to take at each state s with the goal to maximize the return

  * MDP policies depend on the current state only but not the past states.

- Under known MDP, deterministic optimal policy $\pi_*(s)$ exists.
- Under unknown MDP, ϵ−greedy policy (stochastic policy) is needed.

Bellman equation

1. Value functions

- Value functions measure the goodness of each state s (or state-action pair (s, a)) when following a policy π in terms of the expectation of returns G_t (total discounted reward).

For a trajectory (episode): $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, \ldots$

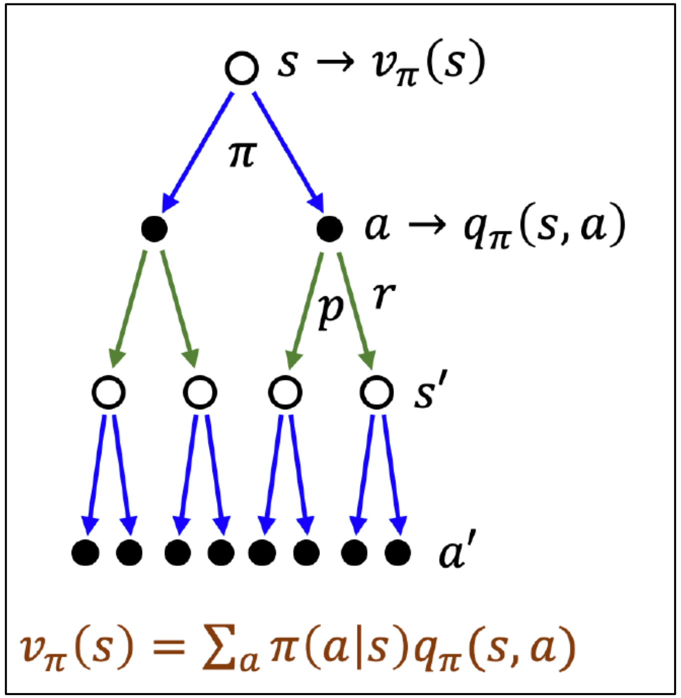
- State-value function $v_\pi(s)$ for policy π is the expected return starting from state s when following policy π.

  $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$

- Action-value function $q_\pi(s, a)$ for policy π is the expected return starting from state s, taking action a, and following policy π.

  $q_\pi(s, a) = \mathbb{E}_\pi[G_{t+1} | S_t = s, A_t = a]$

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



확률변수 X의 확률분포함수가 $f(x)$ 일 때, X의 기대값 E(X)는

1. 이산확률변수

$$E(X) = \sum_x x \cdot f(x)$$

2. 연속확률변수

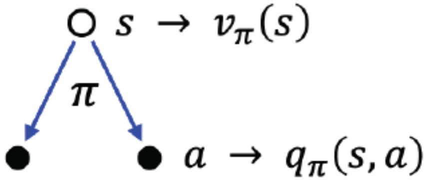
$$E(X) = \int_{-\infty}^{\infty} x \cdot f(x)\, dx$$

2. Bellman Equation

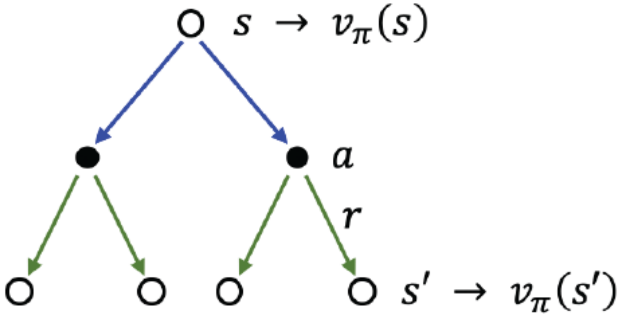
- Bellman expectation equation is a recursive equation decomposing state-value function $v_\pi(s)$ into immediate reward R_{t+1} and discounted next state-value $\gamma v_\pi(S_{t+1})$.

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

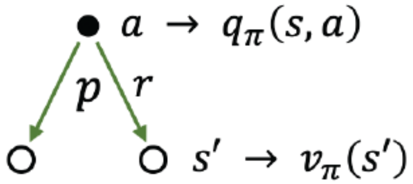
$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \end{aligned}$$



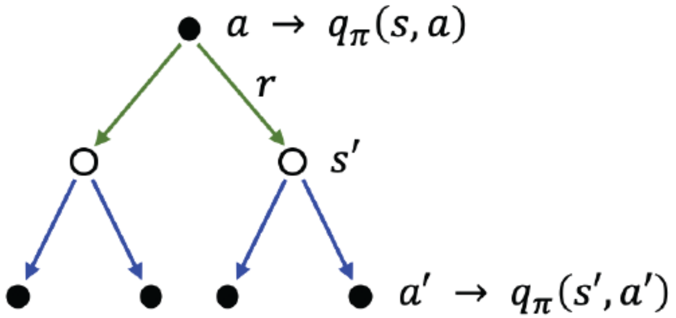
$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$$



$$v_\pi(s) = \sum_a \pi(a|s)[R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')]$$



$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')$$



$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_{a'} \pi(a'|s') q_\pi(s',a')$$

3. Optimal value functions and Policy

The optimal value function yields maximum value compared to all other value functions.

MDP is 'solved' when we find the optimal value functions.

- Optimal state-value function $v_*(s) = \max_\pi v_\pi(s)$
- Optimal action-value function $q_*(s,a) = \max_\pi q_\pi(s,a)$

Define a partial ordering policies $\pi' \geq \pi \;\; if \;\; v_{\pi'}(s) \geq v_\pi(s) \; for \; all \; s$

[Theorem] Any MDP satisfies the followings.

- There exists an optimal policy $\pi_* \geq \pi$ all π
- All optimal policies achieve the optimal state-value function $v_{\pi_*}(s) = v_*(s)$
- All optimal policies achieve the optimal action-value funct $q_{\pi_*}(s,a) = q_*(s,a)$

4. Finding an optimal policy

An optimal policy can be found by maximizing over

$$\pi_*(a|s) = \begin{cases} 1 & if\ a = \arg\max_a q_*(s,a) \\ 0 & otherwise \end{cases}$$

- There is always a deterministic optimal policy for any MDP.
- If we find $q_*(s,a)$, we immediately have the optimal policy $\pi_*(s) = \arg\max_a q_*(s,a)$
- Furthermore,

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

$q_*(s,a) = r(s,a) + \gamma \sum_{s'} p(s' \mid s,a)\, v_*(s')$

  - $v_*(s)$ can be obtained directly from $q_*(s,a)$
  - $q_*(s,a)$ can't be obtained directly from $v_*(s)$

    Instead, we need to know the probabilities $p(s'|s,a)$ under model-based.

- Under model-free, to learn optimal policy $\pi_*(s)$, we directly compute Q-values $Q(s,a)$ using random samples to approximate $q_*(s,a)$ in Reinforcement Learning

5. Bellman optimality equation

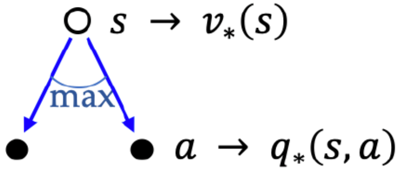
- $v_*(s) = \max_{a \in A(s)} q_*(s,a)$

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

- $q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') |\, S_t = s, A_t = a]$

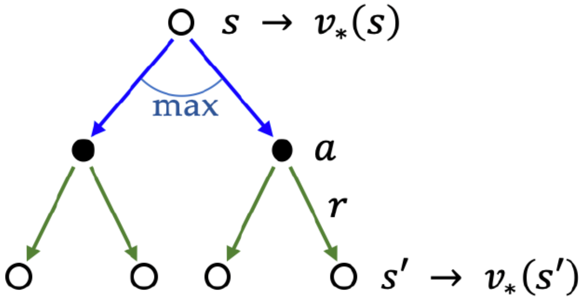
  $= \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a} q_*(s',a')]$

- Under model-based (known MDP, i.e., $p(s',r|s,a)$ and $r(s,a)$), these functions can be iteratively evaluated by Dynamic Programming.



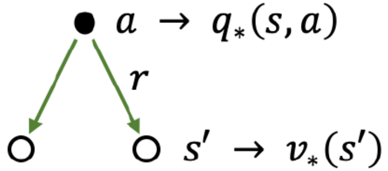
$$v_*(s) = \max_a q_*(s,a)$$

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



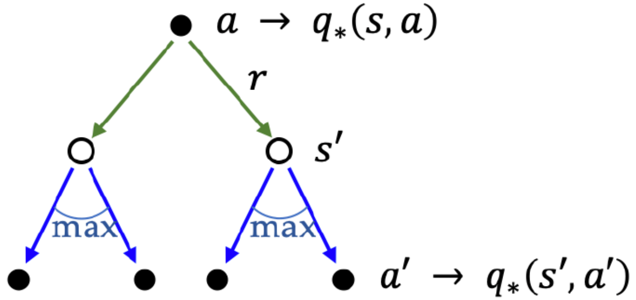
$$v_*(s) = \max_a [R_s^a + \gamma \sum_{s'} P_{ss'}^a v_*(s')]$$

(cf. $v_\pi(s) = \sum_a \pi(a|s)[R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')]$)



$${}_*(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a v_*(s')$$

f. $q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')$)



$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \max_{a'} q_*(s',a')$$

(cf. $q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_{a'} \pi(a'|s') q_\pi(s',a')$)

1. Dynamic Programming

- Dynamic Programming (DP), Bellman 1950s, is a method for solving complex problems by
  - Substructure: decompose the original problem into smaller sub-problems.
  - Table Structure: after solving each sub-problems, store the computed solutions in a table to be re-used many times.
  - Bottom-up Computation: using table, combine the solution of smaller sub-problems to solve larger sub-problems and eventually arrive at a solution of the original problem.
- DP works when a problem has the following properties.
  - Optimal substructure: optimal solution of the problem can be obtained by using optimal solutions of its sub-problems. (e.g. the shortest path problem)
  - Overlapping sub-problems: solutions of same sub-problems are needed repeatedly, so we can store computed solutions in a table to avoid re-computing them.

Markov Decision Process satisfies both properties.

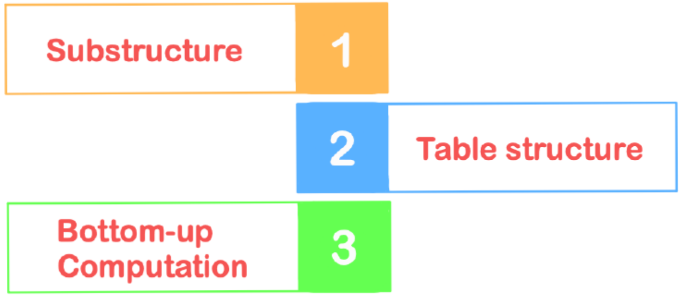
- Bellman equation gives recursive decomposition.
- Value function stores and re-uses solutions.

* Under known MDP, planning with full knowledge of MDP, we solve MDP by using DP.

2. Two solution methods for Sequential Decision Problem

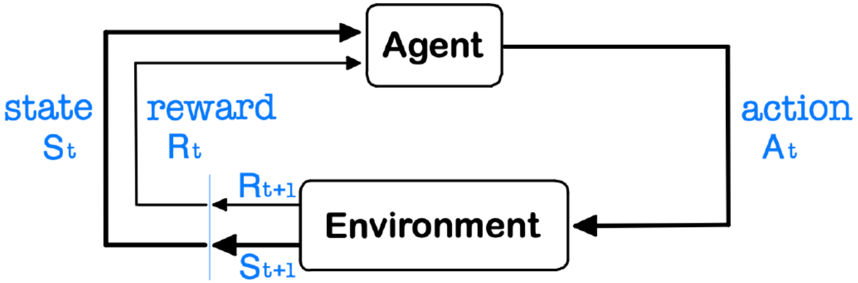
Under known MDP (model-based),
planning with full knowledge,
we use Dynamic Programming.

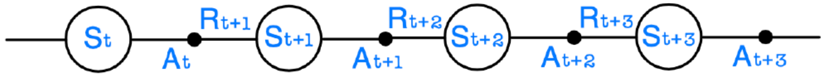
Elements of Dynamic Programming



Planning is computing value functions
by updates of backup operations applied to
simulated experience generated by the model.

Under unknown MDP (model-free),
learning with incomplete information,
we use Reinforcement Learning



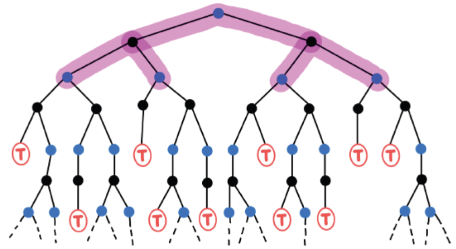


Learning uses real experience
generated by the environment.

3. DP, MC, TD

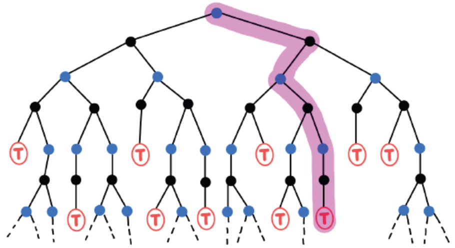
- Dynamic Programming (DP): full backup

  $V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$



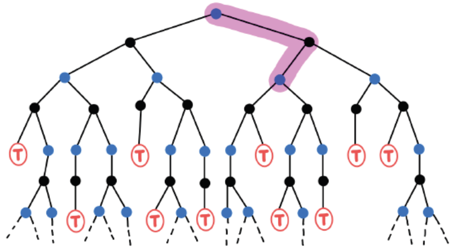
- Monte Carlo (MC): sample multi-step backup

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



- Temporal Difference (TD): sample backup

  $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$



- Bellman expectation equation: $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})|S_t = s]$
- Backup: updating the value of a state using values of future states.

4. Value Iteration

연세대학교 YONSEI UNIVERSITY

- Bellman optimality equation : $v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)][r + \gamma v_*(s')]$
- Value Iteration iteratively computes the following until convergence.

  $V_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)][r + \gamma V_k(s')]$ (compute the maximum over all actions)

  (1) Initialize $V_0(s) = 0$ for all state s (or randomly)

  (2) Update $V_{k+1}(s)$ iteratively from all $V_k(s)$ (full back) until convergence to $V^*(s)$ using

  - synchronous backups: compute $V_{k+1}(s)$ for all s and update simultaneously
  - asynchronous backups: compute $V_{k+1}(s)$ for one s and update it immediately

  (3) Compute the optimal policy π_* (one-step lookahead)

  $\pi_*(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)][r + \gamma V^*(s')]$

- Disadvantages: (1) The action argument inducing the max at each state rarely changes,

  so the policy often converges long before the values converge.

  (2) It is slow as $O(S^2A)$ per iteration and needs many iterations to converge
- The convergence means that for sufficiently small ϵ, $|V_{k+1}(s) - V_k(s)| < \epsilon$ for all s.

5. Policy Iteration

Policy Iteration repeats policy evaluation and policy improvement until convergence.

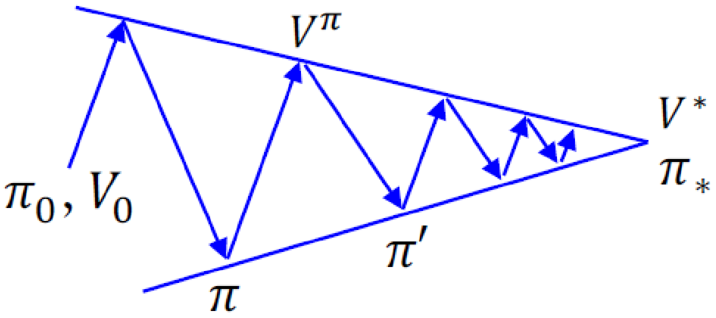
- Policy Evaluation: computing V^{π} from the deterministic policy π.

  $V_{k+1}(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r+\gamma V_k(s')]$ (compute the maximum over all actions)

(1) Initialize $V_0(s) = 0$ for all state s.

(2) Update every $V_{k+1}(s)$ from all $V_k(s')$ (full backup) until convergence to $V^{\pi}(s)$.

* It is faster than Value Iteration because we consider only one action.



- Policy Improvement: improving π to π' by greedy policy based on V^{π}.

  $\pi_*(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)][r+\gamma V^{\pi}(s')] = \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s,a)$

  * In value Iteration, π_* is the optimal policy based on V^* at the end of process.

- Compare to Value Iteration, we need much fewer iterations to reach optimal policy.
- Since $Q^{\pi}(s,\pi'(s)) \geq V^{\pi}(s) = \sum_a \pi(a|s)\, Q^{\pi}(s,a)$, always either (1) π' is strictly better than π, or (2) π' is optimal when $\pi = \pi'$. (Policy Improvement Theorem)

1. DP vs RL

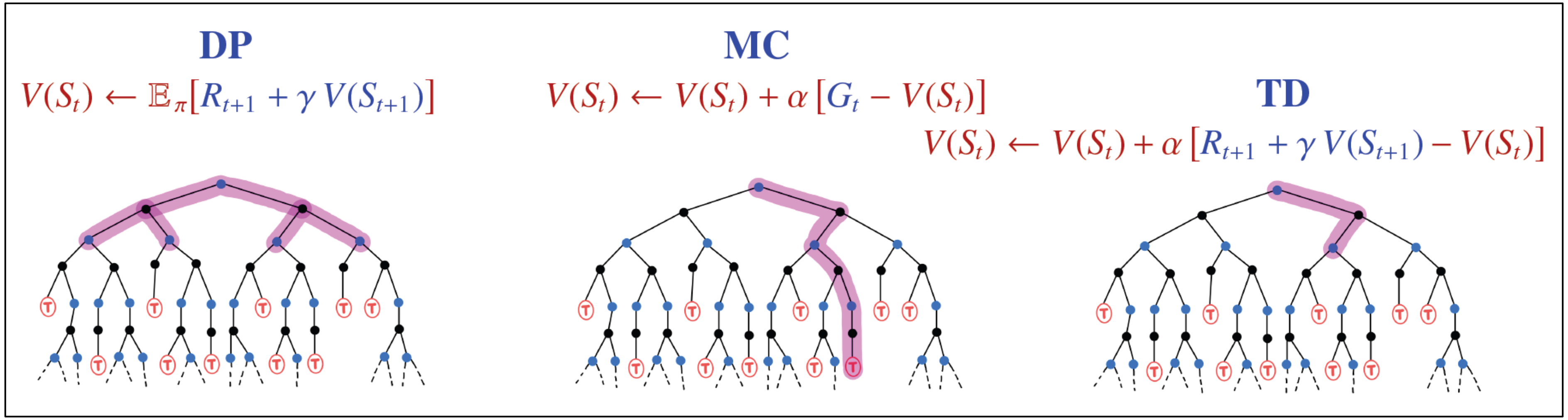
- Dynamic Programming (DP): Planning under model-based using full backup

$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)][r + \gamma V^{\pi}(s')]$$

- Reinforcement Learning (RL): Learning under model-free using sample backup and approximately solving Bellman Optimality Equation

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\ Q^{\pi}(s,a)$$

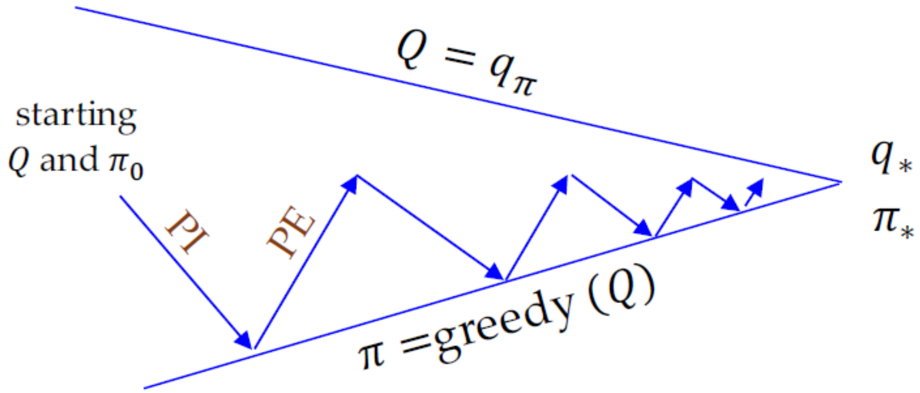
Because V(s) is not sufficient to determine optimal policy



2. Generalized Policy Iteration

- Policy Iteration
  - Policy Evaluation: Makes the value function 'consistent with the current policy'.
  - Policy Improvement: Makes the policy 'greedy w.r.t. the current value function'.
- GPI uses the repeatedly approximated value function to the true value of the current policy
- If both PE and PI stabilize, then the value function and policy must be optimal since Bellman optimality equation holds.

$$\pi_0 \rightarrow q_{\pi_0} \rightarrow \pi_1 \rightarrow q_{\pi_1} \rightarrow \cdots \rightarrow \pi_* \rightarrow q_*$$



연세대학교 YONSEI UNIVERSITY

- MC Policy Iteration adapts GPI based on episode-by-episode of PE estimating $Q(s,a) = q_\pi(s,a)$ and ϵ-greedy PI.
- MC doesn't update value estimates based on value estimates of successor states => No bootstrapping
  - Less harmed by violations of Markov property

- Goal: learn q_π from entire episodes of real experience under policy π
- Action-value function: $q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$
- To estimate $q_\pi(s,a)$, at first/every time-step t that the state *s* was visited and the action *a* was selected in an episode.
- Increment counter $n(s,a) \leftarrow n(s,a) + 1$
- Increment total return $S(s,a) \leftarrow S(s,a) + G_t$
- Value is estimated by mean return $Q(s,a) = \frac{S(s,a)}{n(s,a)}$
- $Q(s,a) \rightarrow q_\pi(s,a)\ as\ n(s,a) \rightarrow\ \infty$ by the law of large numbers.

* Incremental Mean

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{1}{k}\left(\sum_{i=1}^{k-1} x_i + x_k\right) = \frac{1}{k}\big((k-1)\mu_{k-1} + x_k\big) = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

[Incremental Monte Carlo updates]

- Update $Q(s,a)$ incrementally after one-episode $s_0, a_0, r_1, s_1, \dots, r_T, s_T$. For each state-action pair (S_t, A_t) with return G_t,

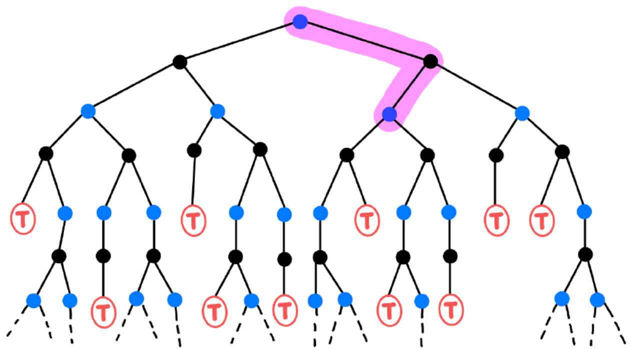
$$n(S_t, A_t) \leftarrow n(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{n(S_t, A_t)}[G_t - Q(S_t, A_t)]$$

[Constant- α MC Policy Evaluation]

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)]$$

- TD Policy Iteration adapts GPI based on one-step transitions of sampled episodes.
- Advantages of TD
  - Bootstrapping of DP: update estimates without waiting for final outcomes (online).
  - Sampling of MC: do not require knowing next-state transition probabilities.



[Three RL Policy Evaluations]

- Monte Carlo: On-Policy MC Prediction

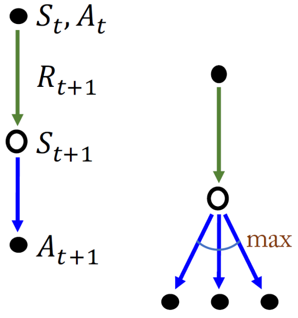
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)]$$

- Sarsa: On-Policy TD Prediction

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

- Q-Learning: Off-Policy TD Prediction

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



- Target Policy: that an agent trying to learn (agent learns value function for this policy)
- Behavior Policy: that is being used by an agent for choosing actions and generating data
- On-Policy: target policy = behavior policy
- Off-Policy: target policy π $\neq$ behavior policy μ

5. TD(λ)

연세대학교 YONSEI UNIVERSITY

- N−step return

$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$ ← TD target

$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$

$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$ ← MC target

- TD(n): n−step TD learning

$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t^{(n)} - V(S_t) \right]$ ← n−step backup

- λ−return: it combines all n−step return using weight $(1-\lambda)\lambda^{n-1}$

$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$

- TD(λ)

$V(S_t) \leftarrow V(S_t) + \alpha [G_t^{\lambda} - V(S_t)]$

1. Deep Reinforcement Learning (DRL)

Reinforcement Learning (RL)

- Tabular updating method: making state–action value $Q(s, a)$ table and then finding optimal policy by updating it repeatedly, using Bellman equation

Deep Reinforcement Learning (DRL)

- Function approximation method: approximating the state–action value function or policy by deep neural networks.
  - Value function $Q(s, a)$ ⇒ Depp Q–Network (DQN)
  - Policy $\pi(a|s)$ ⇒ Policy Gradient (REINFORCE)
  - Value function + policy ⇒ Actor–Critic (A3C)

2. DQN

- It stabilizes training Q-value function approximation with CNN using
  - Experience replay (Replay buffer) ← to overcome the temporal correlation problem
  - Target network ← to overcome the non-stationary target problem
  - Clipping rewards

Experience Replay (Replay Buffer)

- Online RL issues:
  - Strongly temporally-correlated updates that break independent identically distribution (i.i.d.) assumption,
  - Rapid forgetting of rare experiences that would be useful later on.

- Experience replay stores experiences in Replay buffer of large scale including 4-tuples of state transitions, actions and rewards $\{(s, a, r, s')\}$ to perform Q-learning.

2. DQN

- It stabilizes training Q-value function approximation with CNN using
  - Experience replay (Replay buffer) ← to overcome the temporal correlation problem
  - Target network ← to overcome the non-stationary target problem
  - Clipping rewards

Target Network

- If the target function is changed frequently, then this moving target function makes training difficult (non-stationary target problem).
- Target network technique fixes previous parameters $\hat{\theta}$ to Target $\hat{Q}$-network and updates parameter θ only on Behavior Q-network.

- Forward pass

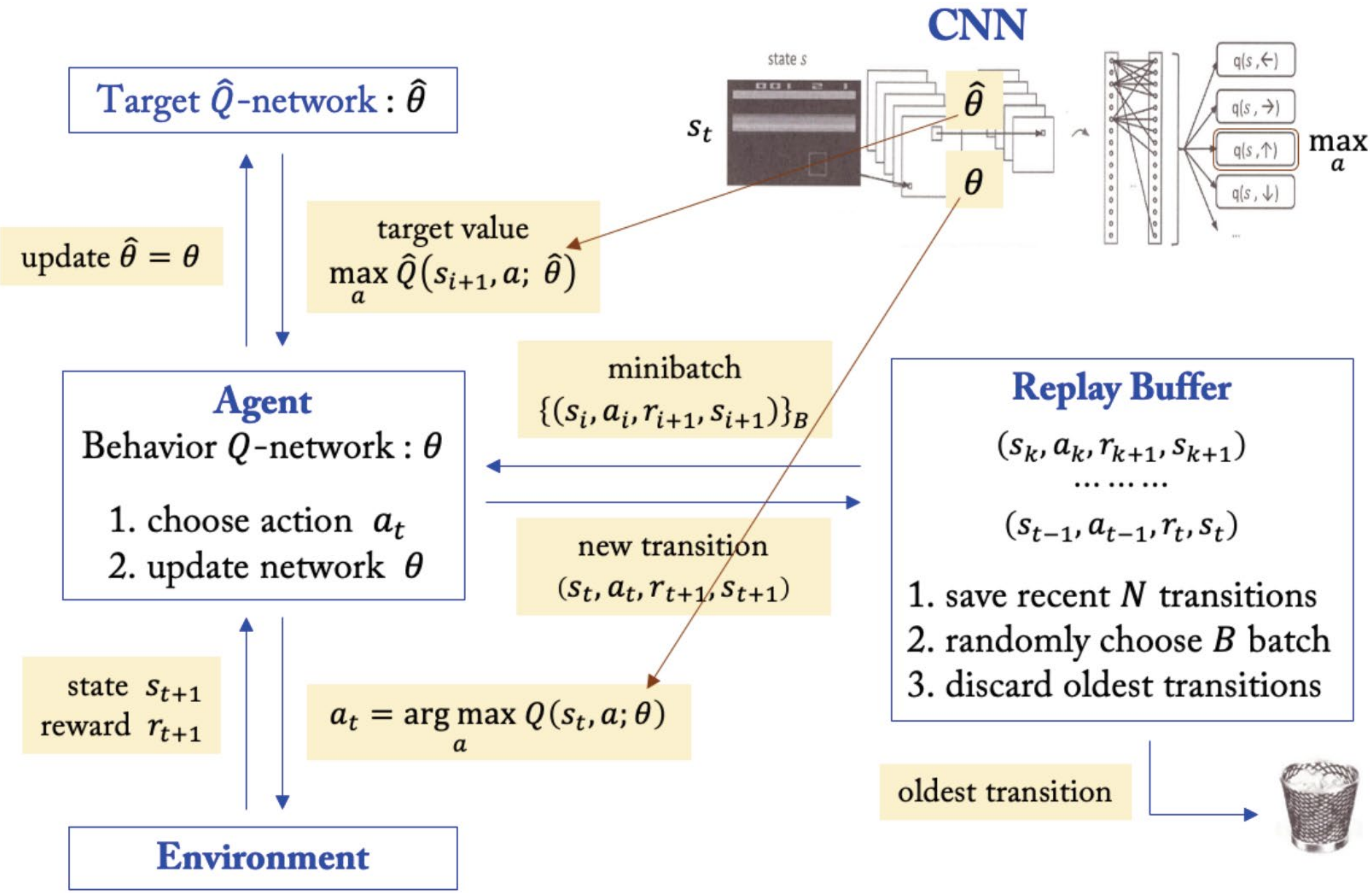
$$L(\theta) = \frac{1}{B}\sum_{|\{i\}|=B}\left[r_{i+1} + \gamma \max_a \hat{Q}(s_{i+1}, a; \hat{\theta}) - Q(s_i, a_i; \theta)\right]^2$$

- Backward pass

$$-\nabla_\theta L(\theta) = \frac{1}{B}\sum_{|\{i\}|=B}\left[r_{i+1} + \gamma \max_a \hat{Q}(s_{i+1}, a; \hat{\theta}) - Q(s_i, a_i; \theta)\right]\nabla_\theta Q(s_i, a_i; \theta)$$

- DQN weight Update: $\theta \coloneqq \theta - \alpha \nabla_\theta L(\theta)$

2. DQN


CNN
state s
s_t
$\hat{\theta}$
θ
q(s,←)
q(s,→)
q(s,↑)
q(s,↓)
$\max_a$
Target $\hat{Q}$-network : $\hat{\theta}$
update $\hat{\theta} = \theta$
target value
$\max_a \hat{Q}(s_{i+1}, a; \hat{\theta})$
minibatch
$\{(s_i, a_i, r_{i+1}, s_{i+1})\}_B$
Agent
Behavior Q-network : θ
1. choose action a_t
2. update network θ
new transition
$(s_t, a_t, r_{t+1}, s_{t+1})$
Replay Buffer
$(s_k, a_k, r_{k+1}, s_{k+1})$
… … …
$(s_{t-1}, a_{t-1}, r_t, s_t)$
1. save recent N transitions
2. randomly choose B batch
3. discard oldest transitions
state s_{t+1}
reward r_{t+1}
$a_t = \arg\max_a Q(s_t, a; \theta)$
oldest transition
Environment

3. Policy Gradient algorithm

- It directly learns the optimal policy by a parametric probability distribution $\pi_\theta(a|s)$, that stochastically selects action a (as a network output) in state s according to parameter θ.

  (without knowing Q-value function as in DQN, which needs a tremendous amount of time)

- It typically proceeds by sampling this stochastic policy and adjusting θ in the direction of greater total reward.

  (no need for Experience replay as in DQN)

DQN $\quad q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \approx Q(s,a;\theta)$

$$L(\theta) = \frac{1}{B}\sum_{|\{i\}|=B}[r_{i+1} + \gamma \max_a \hat{Q}(s_{i+1}, a; \hat{\theta}) - Q(s_i, a_i, \theta)]^2$$

$$\theta \coloneqq \theta + \alpha\frac{1}{B}\sum[r_{i+1} + \gamma \max_a \hat{Q}(s_{i+1}, a; \hat{\theta}) - Q(s_i, a_i, \theta)]\nabla_\theta Q(s_i, a_i; \theta)$$

Define θ to parameterize the policy π.

- Trajectory: $\tau = s_0, a_0, r_1, s_1, a_1, \cdots, s_T$
- Total reward: $r(\tau)$
- Objective function: $J(\theta) = \mathbb{E}_{\pi_\theta}[r(\tau)] = \int p(\tau;\theta)r(\tau)d\tau$ where $p(\tau;\theta) = \pi_\theta(\tau)$ is the probability density function of τ.

To find the optimal θ^* which maximizes $J(\theta)$, we use gradient ascent.

- Policy gradient update: $\theta \coloneqq \theta + \alpha\nabla_\theta \mathrm{J}(\theta)$

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{\pi_\theta}[r(\tau)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)]$$

(1) Do not need to know $p(\tau;\theta)$, which are practically hard to model.
(2) The expectation can be approximated by sampling (using minibatch).
-> MCMC (Markov Chain Markov Carlo)

3. Policy Gradient algorithm – REINFORCE

- REINFORCE (Monte Carlo Policy Gradient) algorithm is a popular policy gradient algorithm.

Repeat (1) ~ (3)

(1) Execute M trajectories

(each starting in state s and executing (stochastic) policy π_θ)

(2) Approximate the gradient of the objective function $J(\theta)$

$$g_\theta \coloneqq \frac{1}{M}\sum_{i=1}^{M}\left(\sum_{t=0}^{T-1} G_t^{(i)} \nabla_\theta \log \pi_\theta \left(a_t^{(i)} \middle| s_t^{(i)}\right)\right) \approx \nabla_\theta J(\theta)$$

(3) Update policy (network parameters) to maximize $J(\theta)$

$$\theta \coloneqq \theta + \alpha g_\theta \approx \theta + \alpha \nabla_\theta J(\theta)$$

- This is directly updating policy itself.
- Here, the total reward $r(\tau^{(i)})$ is replaced by the discounted return $G_t^{(i)}$.

4. Actor-critic Method

- Critic network: updates parameter ϕ for value function $V(s;\phi)$ or $Q(s,a;\phi)$.
- Actor network: updates parameter θ for policy $\pi_\theta(a|s)$ using Policy Gradient $\nabla_\theta J(\theta)$.

- Learning value function in addition to policy is valuable since knowing value function can assist policy update, as in REINFORCE with baseline to reduce variance.

- REINFORCE with baseline is unbiased but learns slowly because of still high variance of G_t like MC methods, and it is inconvenient to implement for online learning.

To eliminate these inconveniences, we use TD Actor-Critic method with bootstrapped Critic,

- Critic and Actor practically share the network, but use their own weight parameters ϕ and θ.

4. Actor–critic Method

Critic estimates the value function by minimizing the loss $L(\phi)$.

- MC target G_t: $\Delta\phi = \beta\left(G_t - V_\phi(s_t)\right)\nabla_\phi V_\phi(s_t)$

  $\Rightarrow$ code: $\delta \leftarrow G_t - V(s_t;\phi)$
  $\phi \leftarrow \phi + \beta\delta\nabla_\phi V(s_t;\phi)$

- TD target $r + \gamma V_\phi(s')$: $\Delta\phi = \beta\left(r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)\right)\nabla_\phi V_\phi(s_t)$

  $\Rightarrow$ code: $\delta \leftarrow r + \gamma V(s_{t+1};\phi) - V(s_t;\phi)$
  $\phi \leftarrow \phi + \beta\delta\nabla_\phi V(s_t;\phi)$

Actor estimates the policy by using the policy gradient $\nabla_\theta J(\theta)$

- MC policy gradient update: $\Delta\theta = \alpha\left(G_t - V_\phi(s_t)\right)\nabla_\theta log\pi_\theta(a_t|s_t)$

  $\Rightarrow$ code: $\theta \leftarrow \theta + \alpha\gamma^t\delta\,\nabla_\theta \log\pi(a_t|s_t;\theta)$

- TD policy gradient update: $\Delta\theta = \alpha(r_{t+1} + \gamma V\_\phi(s_{t+1})$

  $\Rightarrow$ code: $\theta \leftarrow \theta + \alpha\gamma^t\delta\,\nabla_\theta \log\pi(a_t|s_t;\theta)$

End