



Robotic Navigation and Exploration

Unit 04: Reinforcement Learning

Min-Chun Hu anitahu@cs.nthu.edu.tw

CS, NTHU

直播連結 <https://www.youtube.com/@NTHURNE-I9v>

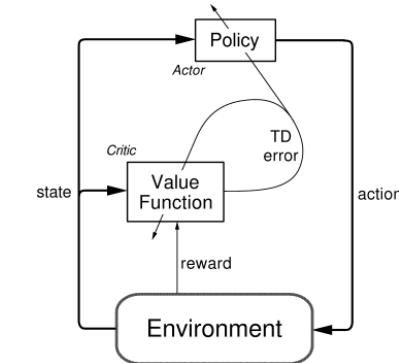
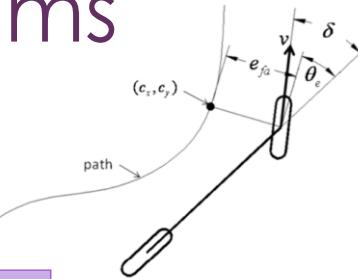
Review of Control Algorithms

$$\delta = K_p e(t) + K_i \sum_0^t e_t + K_d (e(t) - e(t-1))$$

PID Control

$$\delta = \arctan\left(-\frac{ke}{v_f}\right) + \theta_e$$

Stanley Control



**Model-free
Reinforcement Learning**

Apply the kinematic property.

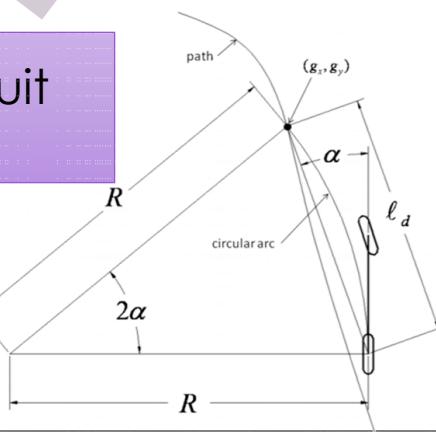
Consider the progressive stability.

More complex motion model.

Don't need model.
Non-linear case.

Pure-Pursuit
Control

$$\delta = \arctan\left(\frac{2L \sin(\alpha)}{L_d}\right)$$



LQR Control

DARE:

$$P_t = Q + A^T P_{t+1} A - A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$$

Outline

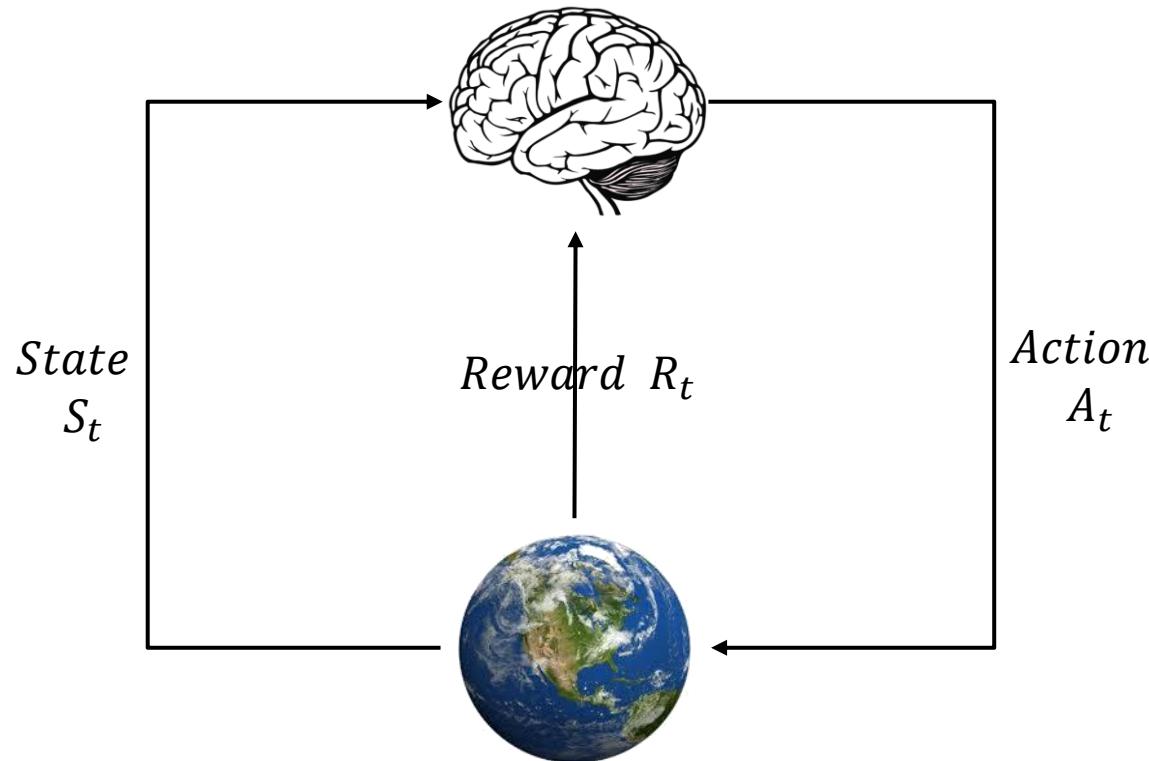
- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- Value-based Reinforcement Learning
 - From MDP to RL
 - Temporal Difference Learning (SARSA / Q-Learning)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- Policy-based Reinforcement Learning
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- Value-based Reinforcement Learning
 - From MDP to RL
 - Temporal Difference Learning (SARSA / Q-Learning)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- Policy-based Reinforcement Learning
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

Reinforcement Learning

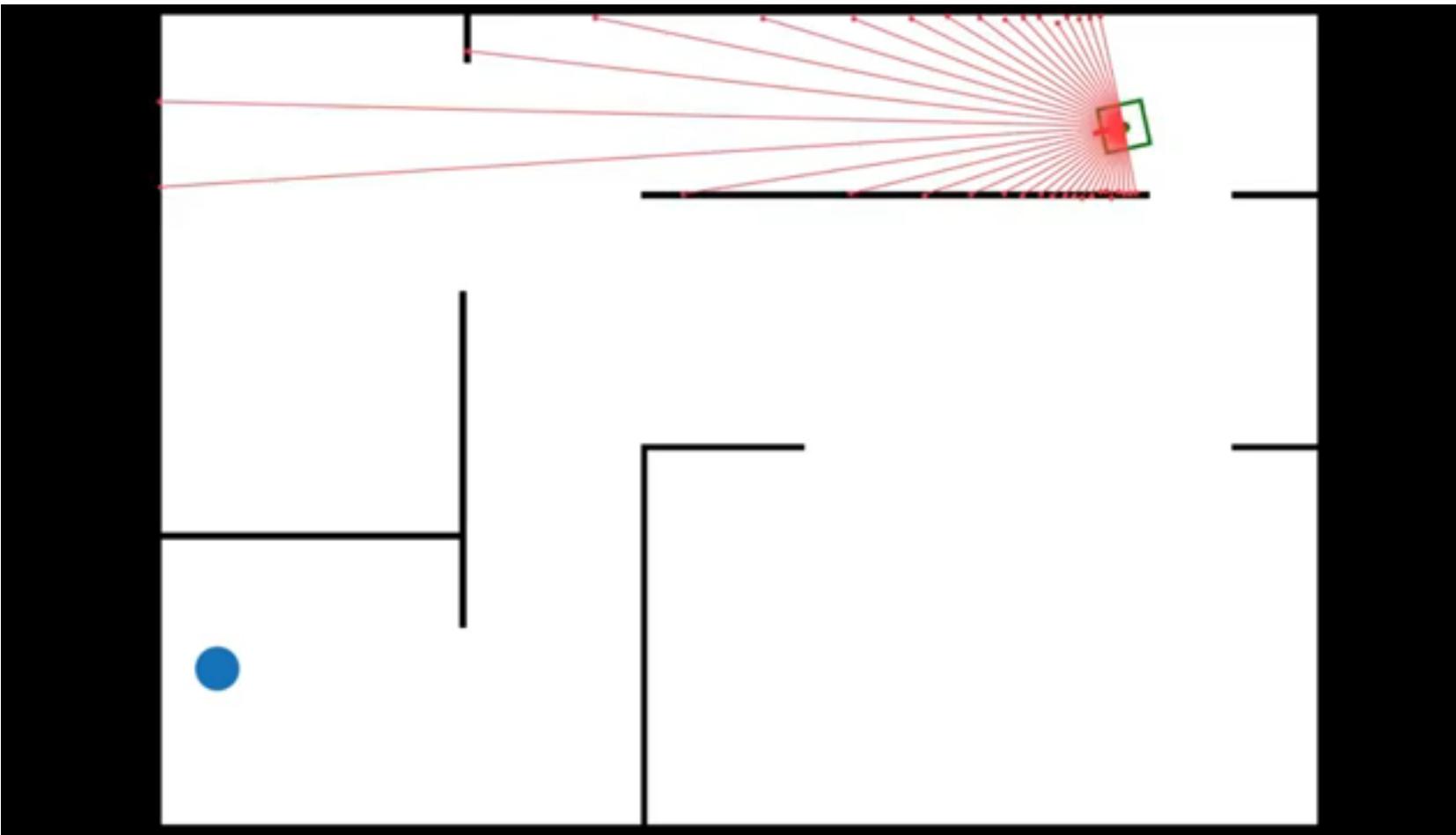
- Reinforcement Learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.



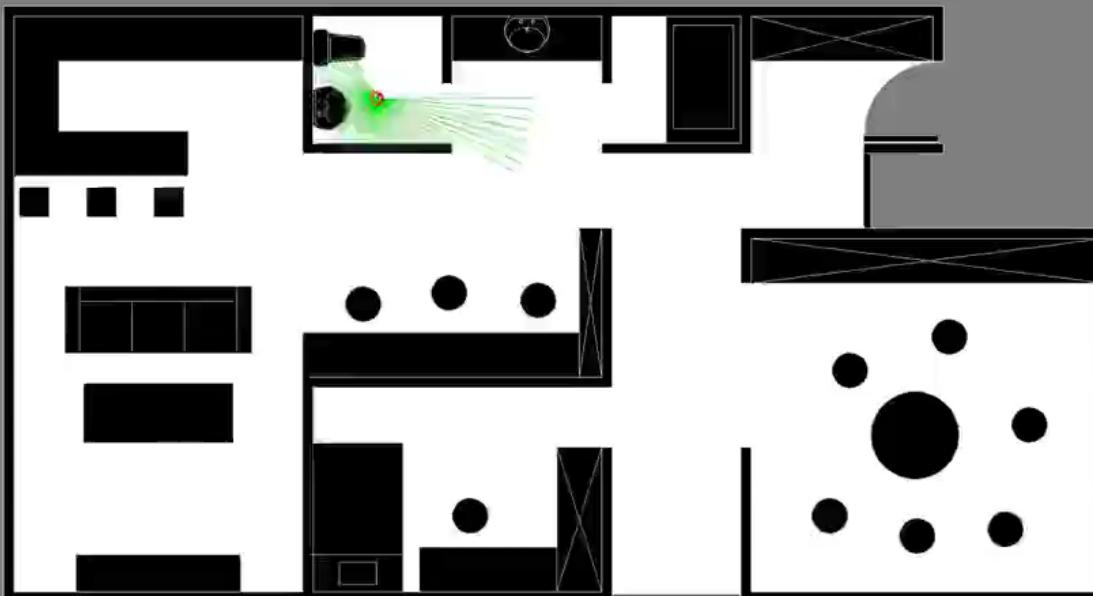
At each step t

- Agent
 - Receive state S_t
 - Receive reward R_t
 - Execute action A_t
- Environment
 - Receive action A_t
 - Emit state S_{t+1}
 - Emit reward R_{t+1}

RL Example – Mapless Navigation



RL Example – Environment Exploration



Reinforcement Learning

- Conceptually, we can treat the “reward” as the “label” in supervised learning, both provide a goal for the model to optimize.
 - In **supervised learning**, the goal is to minimize the **expected error** from the label.
 - In **reinforcement learning**, the goal is to maximize the **expected sum of reward**.
- As for “**unsupervised learning**” and “**meta learning**”, there are corresponding tasks in reinforcement learning field, which are “**No explicit reward RL**” and “**meta RL**” respectively.

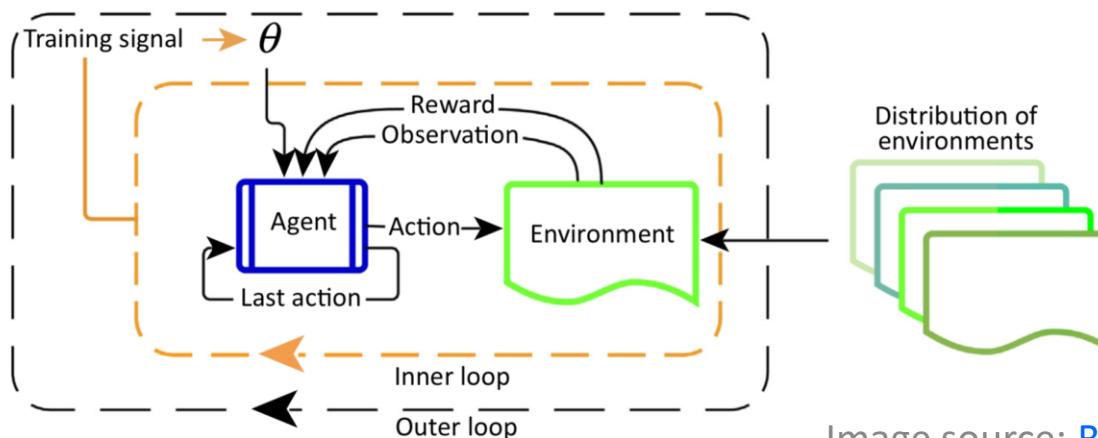


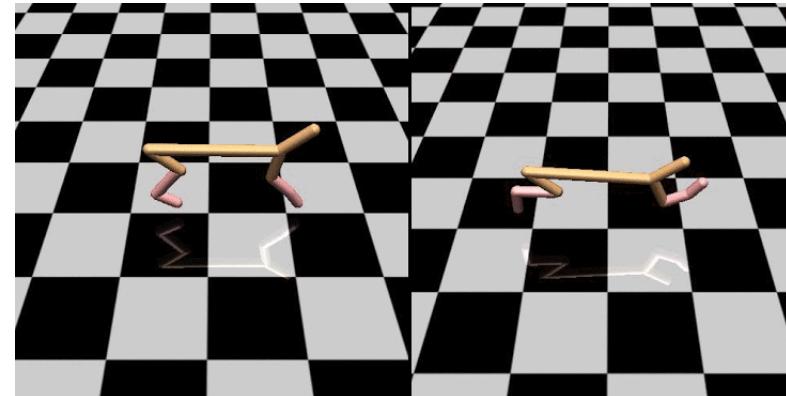
Image source: [Botvinick, et al. 2019](#)

Reinforcement Learning without Reward

- Unsupervised learning aims to find the **potential structure** of the **data** without **label**.
- No reward RL aims to find the **potential strategy** of the **environment** without **explicit reward**.
- For example, given the control of legs, we can first learn the skills that can **critically change the state**, such as raising, moving forward, turning, etc., and then use these potential skills to quickly adapt to new tasks.

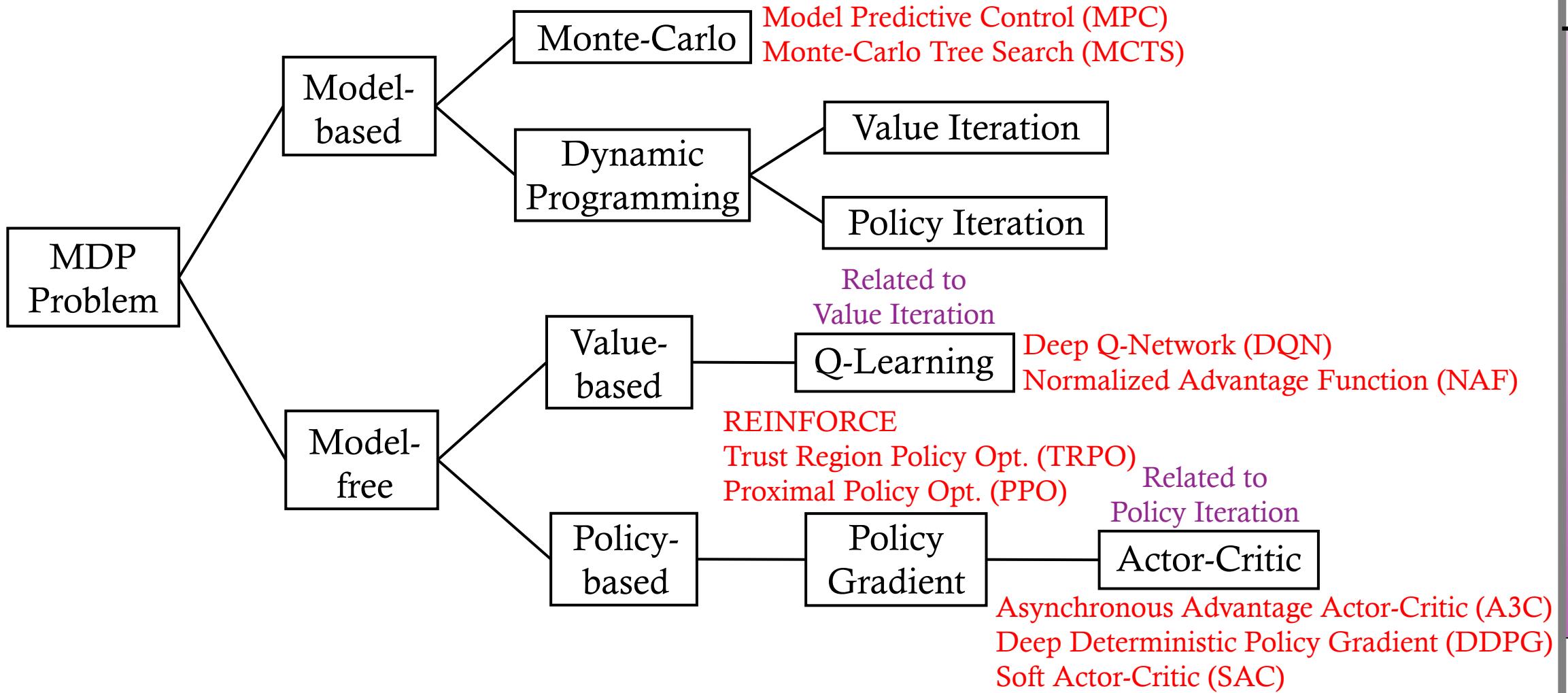


Curiosity-Driven



Skill-Driven

Reinforcement Learning Algorithms



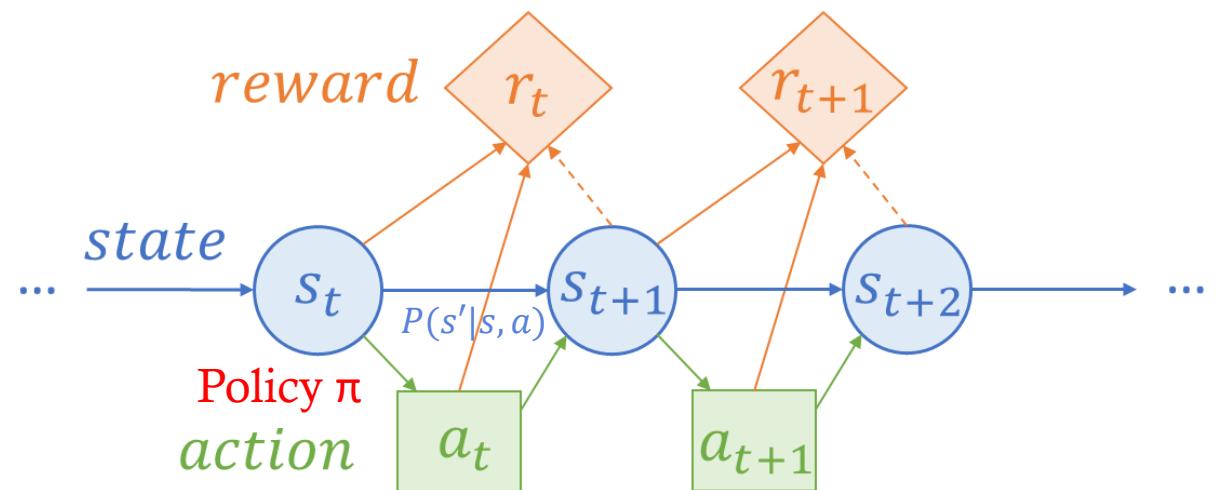
Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- Value-based Reinforcement Learning
 - From MDP to RL
 - Temporal Difference Learning (SARSA / Q-Learning)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- Policy-based Reinforcement Learning
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

Markov Decision Process (MDP)

- A mathematical framework for modeling the **decision-making process**, where the transition probability is assumed to have **Markov property**.
- The goal of MDP is to find the optimal policy π^* to maximize the **expected total reward**.

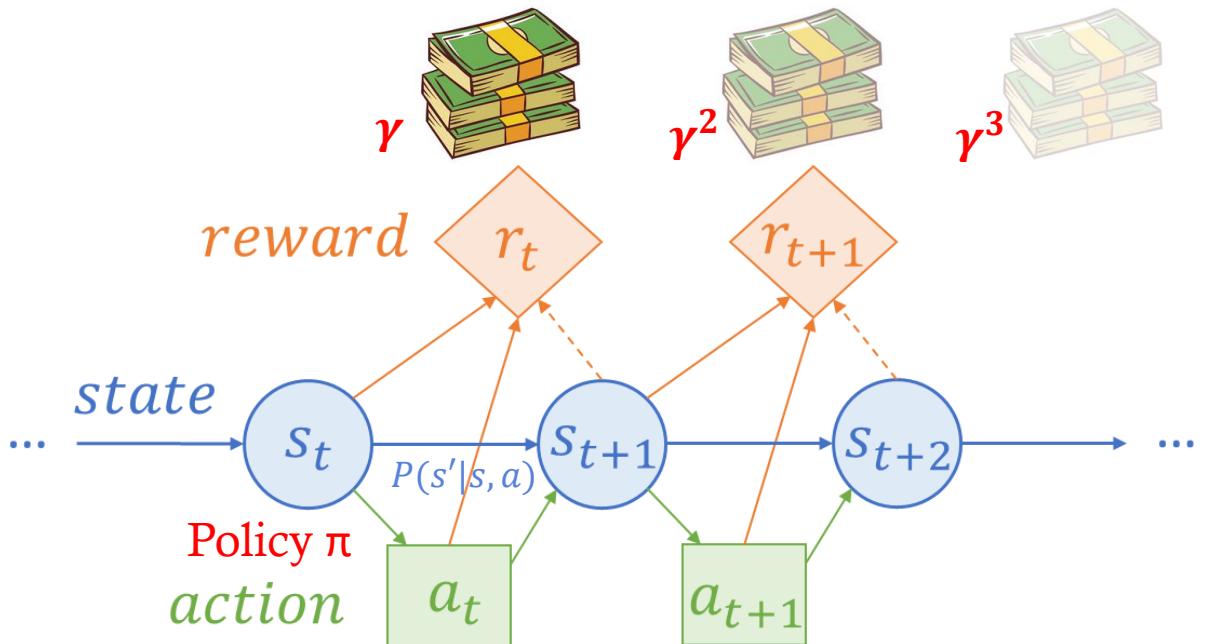
$$\begin{aligned}\pi^* &= \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{(s_t, a_t) \sim \pi, P} [r_0 + r_1 + \dots + r_T] \\ &= \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^T \mathbb{E}_{(s_t, a_t) \sim \pi, P} [r(s_t, a_t)]\end{aligned}$$



Discount Factor

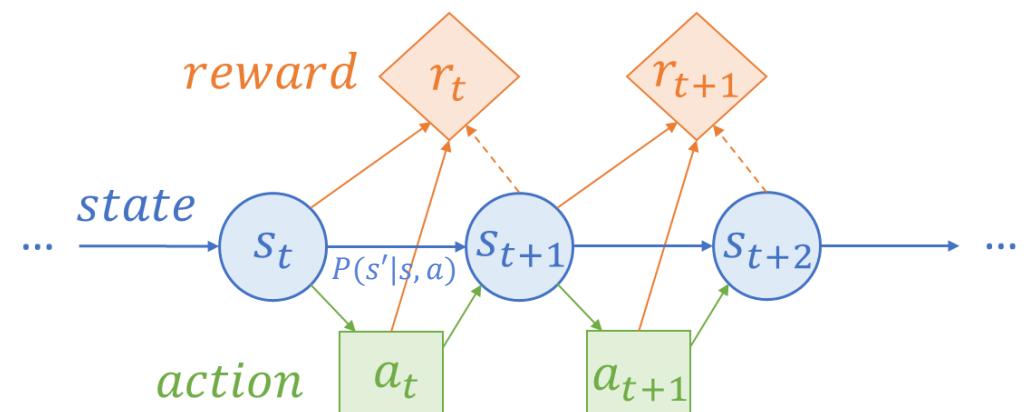
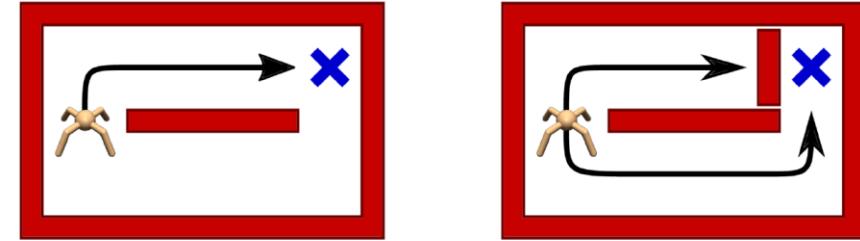
- The expected total reward may not converge to finite number.
- The current reward are more important than the future reward.

$$\begin{aligned}\pi^* &= \operatorname{argmax}_{\pi} \mathbb{E}_{(s_t, a_t) \sim \pi, P}[r_0 + \gamma^1 r_1 + \dots + \gamma^T r_T] \\ &= \operatorname{argmax}_{\pi} \sum_{t=0}^T \mathbb{E}_{(s_t, a_t) \sim \pi, P}[\gamma^t r(s_t, a_t)]\end{aligned}$$



MDP Formulation

- 5-Tuple $\langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$
 - State: $s \in \mathcal{S}$
 - Action: $a \in \mathcal{A}$
 - Transition Probability: $P(s'|s, a)$, $P: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$
 - Reward Function: $r(s, a)$, $r: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
 - Discount Factor: $\gamma \in [0,1]$
- Policy Form
 - Deterministic Policy: $\mu(s) \rightarrow a$
 - Stochastic Policy: $\pi(s) \rightarrow p(a|s)$



State

- A state describes the current status.
- Sometimes it is partially observed.
- For example, position, velocity, angle, ...



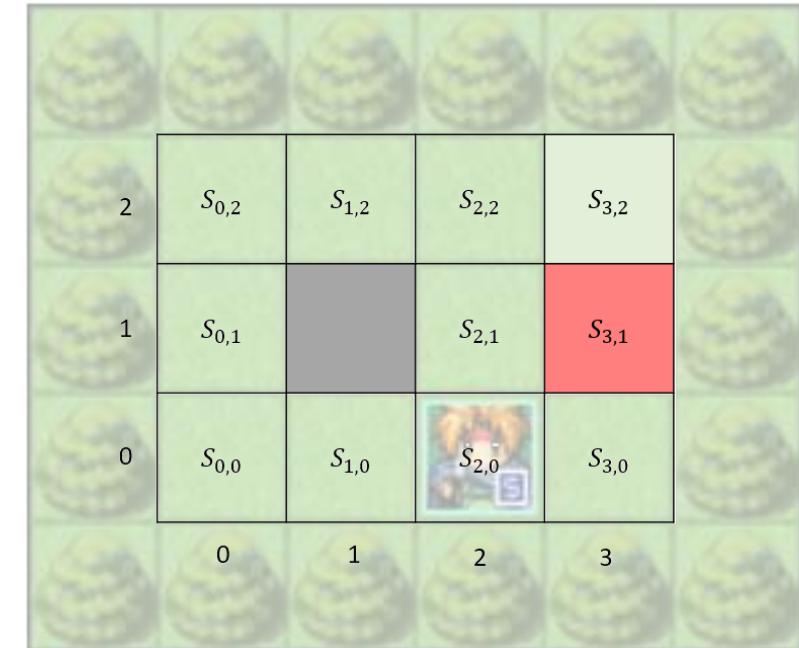
$$s_{1,2} = (1,2)$$

$$s_{0,0} = (0,0)$$

$$s_t = s_{2,1} = (2, 1)$$

$$s_{t+1} = s_{3,1} = (3, 1)$$

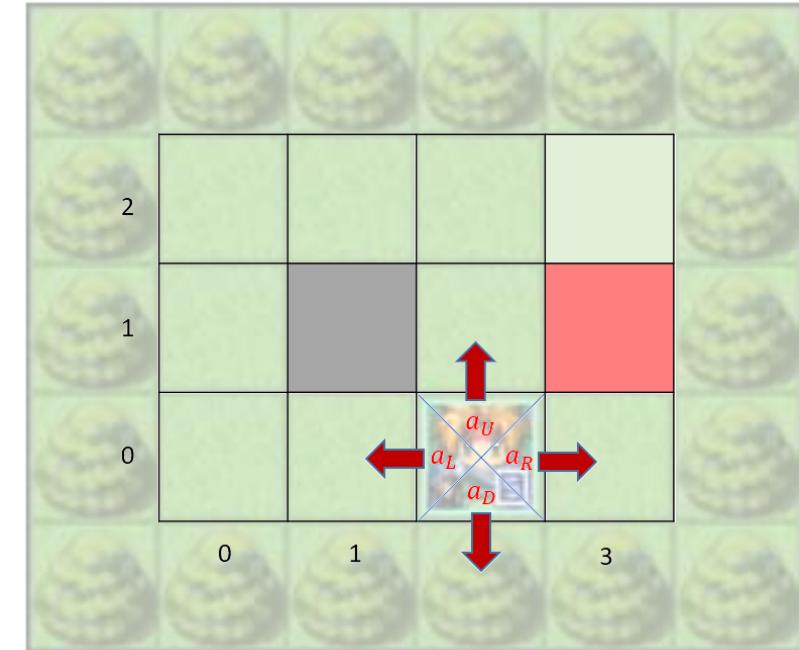
⋮



Action

- The action space defines all possible actions that can be taken for each timestep.
- For example, go forward, turn right, add force, ...

$$\begin{aligned} a_t &= a_U = \uparrow \\ a_{t+1} &= a_R = \rightarrow \\ &\vdots \end{aligned}$$



Transition Probability

- It defines the **(stochastic) model** of the MDP.
- This probability is used to encode all the factors of the environment dynamics.

$$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$$

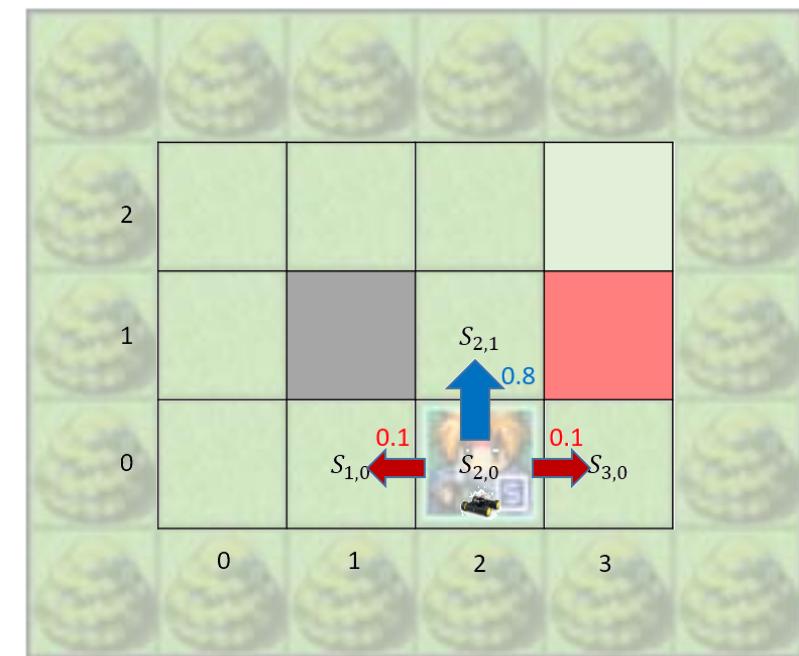
$$s' \sim P(s'|s, a)$$

$$P(s_{2,1}|s_{2,0}, a_U) = 0.8$$

$$P(s_{1,0}|s_{2,0}, a_U) = 0.1$$

$$P(s_{3,0}|s_{2,0}, a_U) = 0.1$$

$$P(s_{0,0}|s_{2,0}, a_U) = 0$$



Reward Function

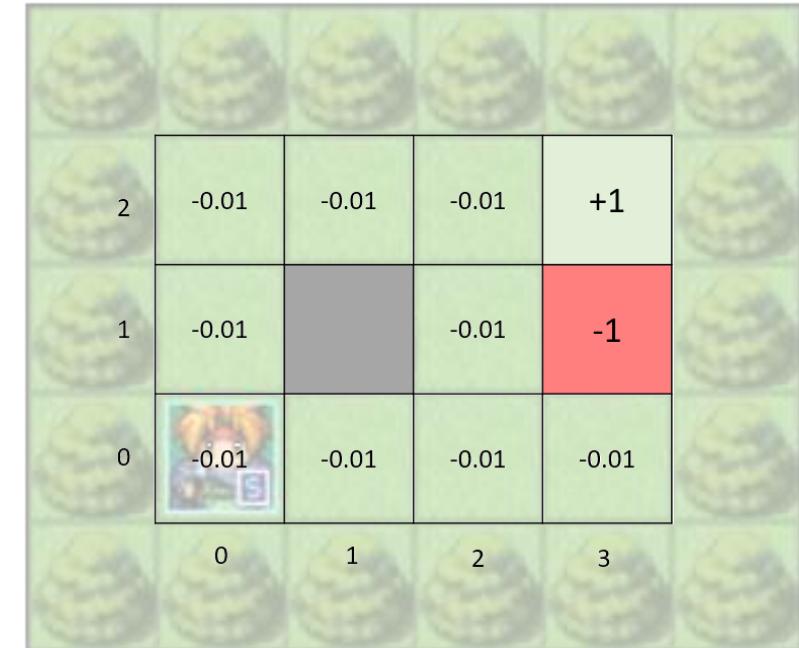
- For each timestep, it gives a reward according to the state-action pair (s_t, a_t) .
- It can be thought of as a signal to guide the agent to take the right actions.

$$r_{t+1} = r(s_t, a_t)$$

$$r_t = r(s_{2,1}, a_U) = r((2, 1), \uparrow) = -0.01$$

$$r_{t+1} = r(s_{2,2}, a_R) = r((2, 2), \rightarrow) = +1$$

⋮



Policy

- We need a policy to decide the action to be taken at each timestep.
 - The policy can be deterministic or stochastic.

$$a \sim \pi(a|s)$$

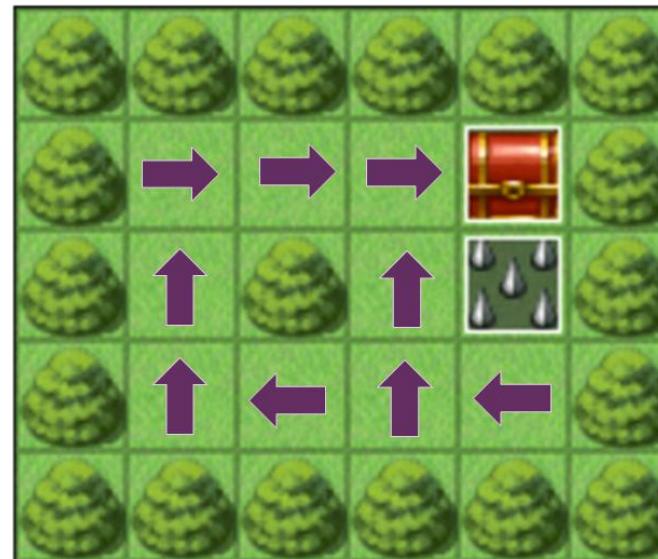
$$\pi(a_U | s_{2,1}) = 1$$

$$\pi(a_D | s_{2,1}) = 0$$

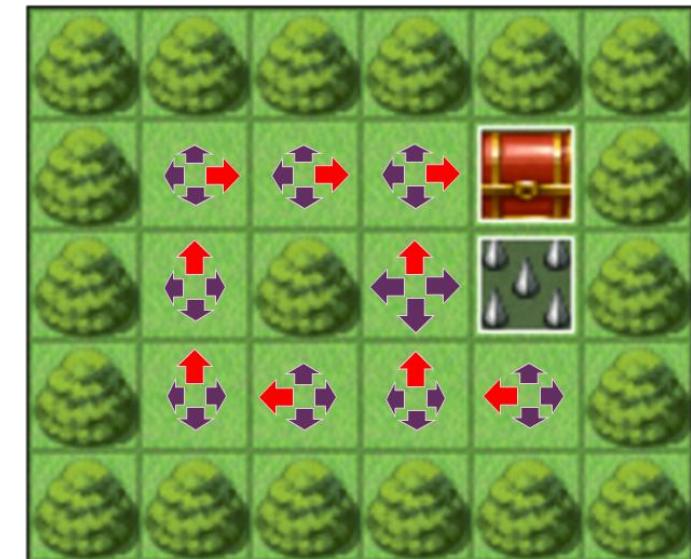
$$\pi(a_L | s_{2,1}) = 0$$

$$\implies \pi(a|s_{2,1}) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Deterministic



Stochastic



Outline

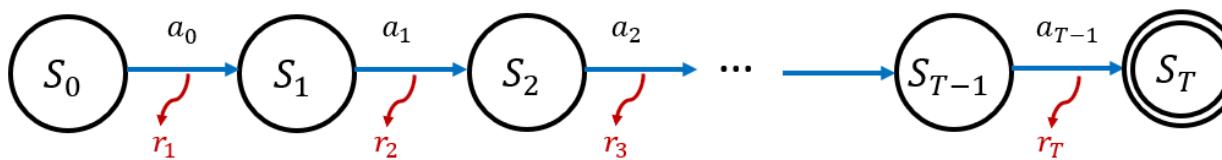
- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- Value-based Reinforcement Learning
 - From MDP to RL
 - Temporal Difference Learning (SARSA / Q-Learning)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- Policy-based Reinforcement Learning
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

Return

- The goal of the MDP is to take a sequence of actions to maximize the **expected total reward**.
- Given a sequence of actions, we can define the “**Return**” as the sum of reward from current state to the terminal state.

$$G_0 = r_1 + r_2 + r_3 + \dots + r_T = \sum_{t=1}^T r_t \quad \text{Undiscounted}$$

$$G_0 = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots + \gamma^{T-1} r_T = \sum_{t=1}^T \gamma^{t-1} r_t \quad \text{Discounted}$$



Return

- Consider the return from timestep t
- Define the return recursively:

$$G_t = r_{t+1} + \gamma G_{t+1}$$

$$G_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \dots + \gamma^{T-2} r_T)$$

1-step

2-step

⋮

N-step

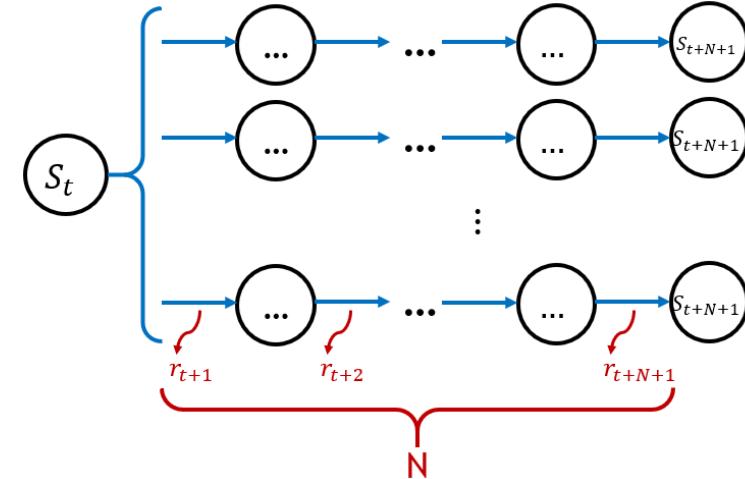
$$= r_{t+1} + \gamma G_{t+1}$$

$$= r_{t+1} + \gamma r_{t+2} + \gamma^2 G_{t+2}$$

= ...

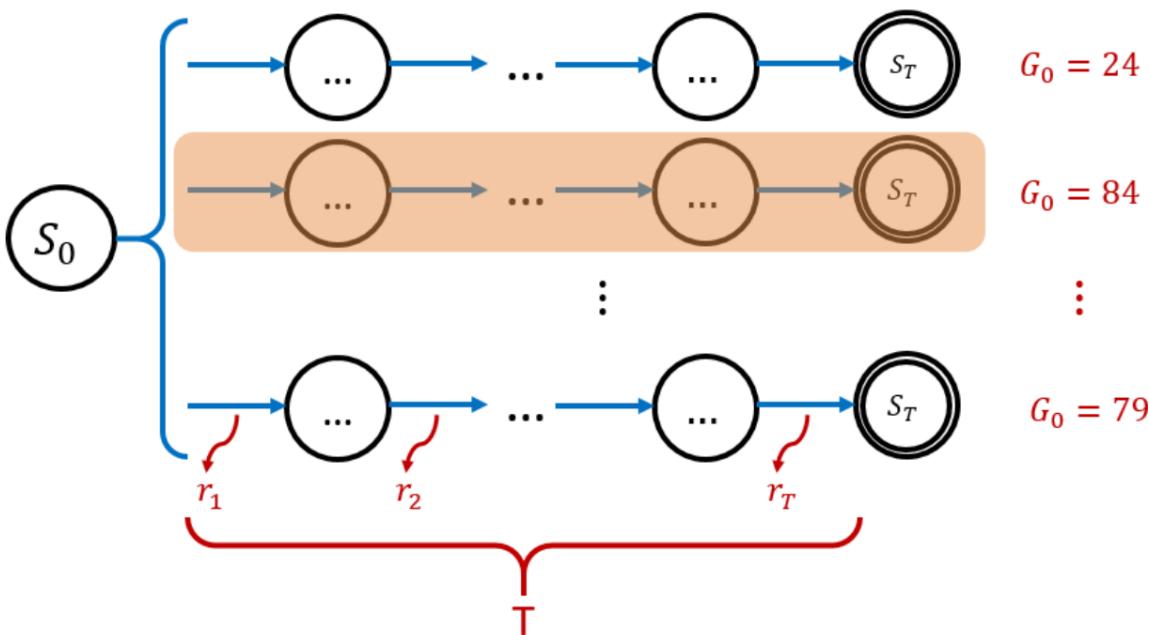
$$= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots + \gamma^N r_{t+N+1} + \gamma^{N+1} G_{t+N+1}$$

$$= \sum_{k=0}^N \gamma^k r_{t+k+1} + \gamma^{N+1} G_{t+N+1}$$



Return

- However, we **CANNOT** directly find the path that maximizes the return because we have to consider the **stochasticity of the environment**.
- Even though the same action sequence is taken for each episode, we probably won't have the same state sequence (won't have the same return).



State-Value Function

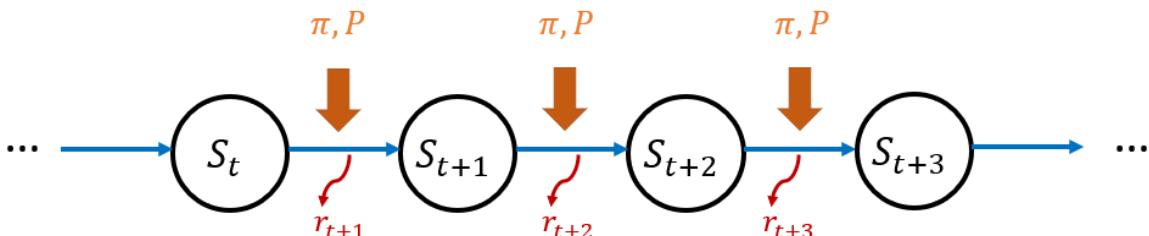
- Consider the expected return from current state

$$\begin{aligned} V^\pi(s_t) &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^T r_T | s_t = s] \\ &= E_\pi[G_t | s_t = s] \end{aligned}$$

- Recursive Definition

$$\begin{aligned} V(s) &= E_\pi[G_t | s_t = s] \\ &= E_\pi[r_{t+1} + \gamma G_{t+1} | s_t = s] \text{ Apply recursive form of return} \\ &= E_{a \sim \pi(a|s)} E_{s' \sim P(s'|s,a)} [r_{t+1} + \gamma E_\pi[G_{t+1} | s_{t+1} = s'] | s_t = s] \\ &= E_{a \sim \pi(a|s)} E_{s' \sim P(s'|s,a)} [r_{t+1} + \gamma V(s') | s_t = s] \end{aligned}$$

$V(S_{0,2})$	$V(S_{1,2})$	$V(S_{2,2})$	+1
$V(S_{0,1})$		$V(S_{2,1})$	-1
$V(S_{0,0})$	$V(S_{1,0})$	$V(S_{2,0})$	$V(S_{3,0})$

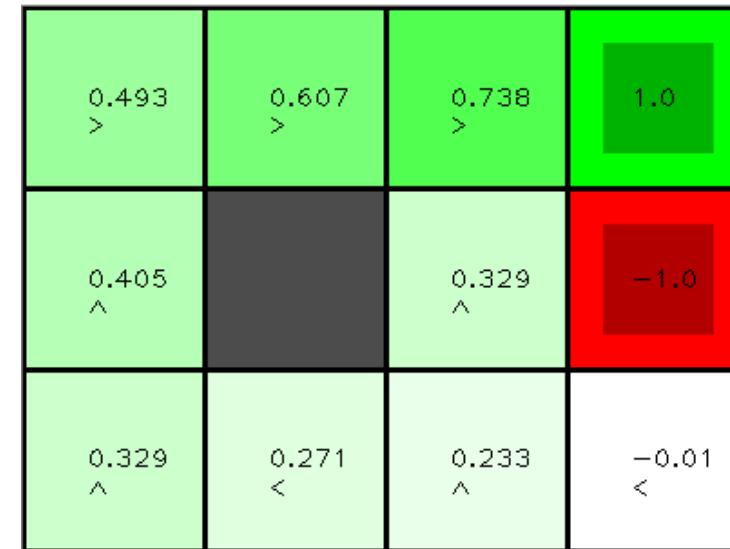
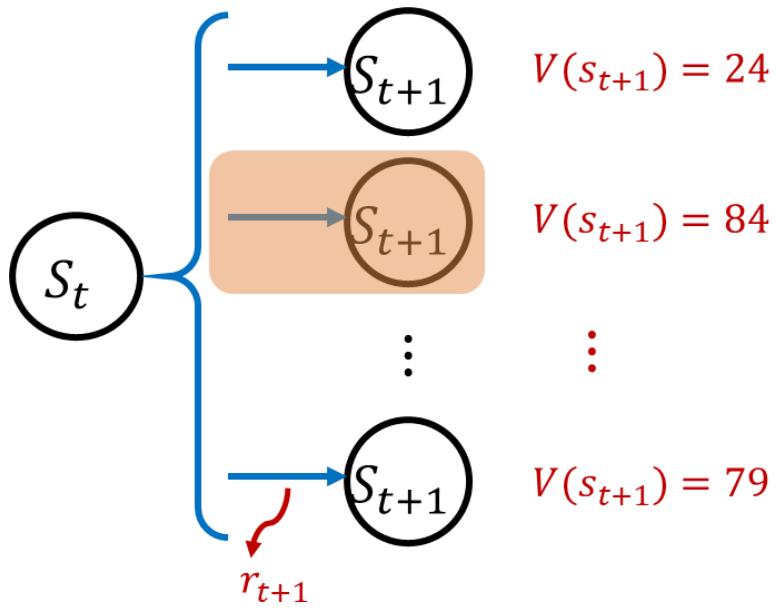


$$\begin{aligned}
 V(s) &= E_{\pi}[G_t | s_t = s] \\
 &= E_{\pi}[r_{t+1} + \gamma G_{t+1} | s_t = s] \\
 &= E_{a \sim \pi(a|s)} E_{s' \sim P(s'|s,a)} [r_{t+1} + \gamma E_{\pi}[G_{t+1} | s_{t+1} = s'] | s_t = s] \\
 &= E_{a \sim \pi(a|s)} E_{s' \sim P(s'|s,a)} [r_{t+1} + \gamma V(s') | s_t = s]
 \end{aligned}$$

State-Value Function

- If we have the state-value function, we can design a policy to choose an action according to the maximum state-value of the next state s_{t+1} .

$$\pi(s) = \max_a \{ \mathbb{E}_{s' \sim P} [r(s, a) + \gamma V(s')] \}$$



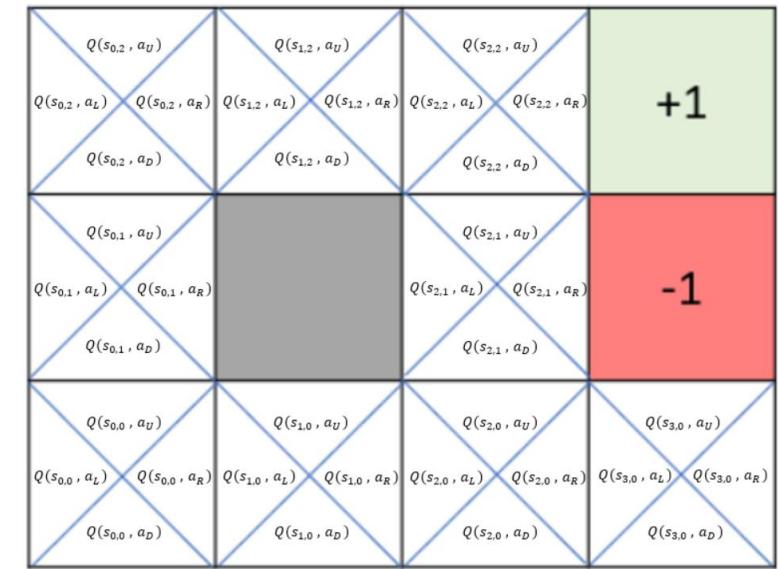
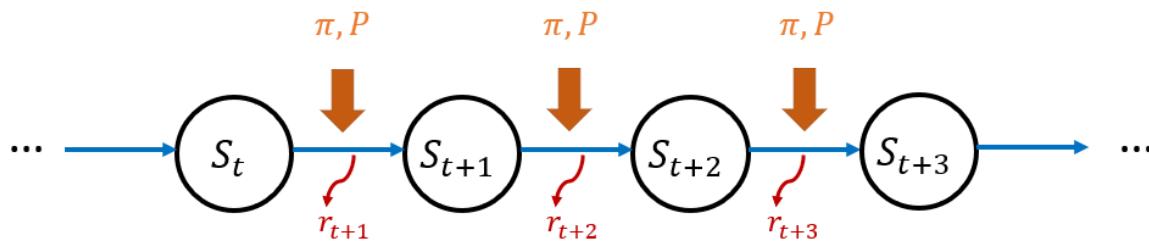
$$\pi(s) = \max_a \mathbb{E}_{S' \sim P}[r(s, a) + \gamma V(s')]$$

Action-Value Function (Q-function)

- Consider the expected return from current state s given the action a .

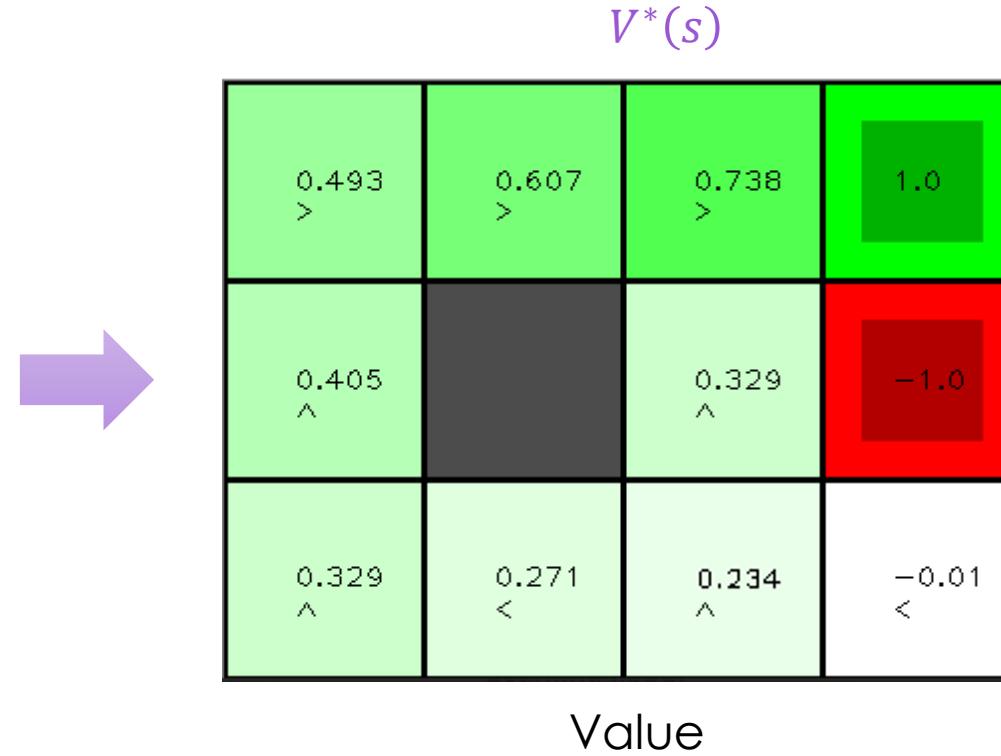
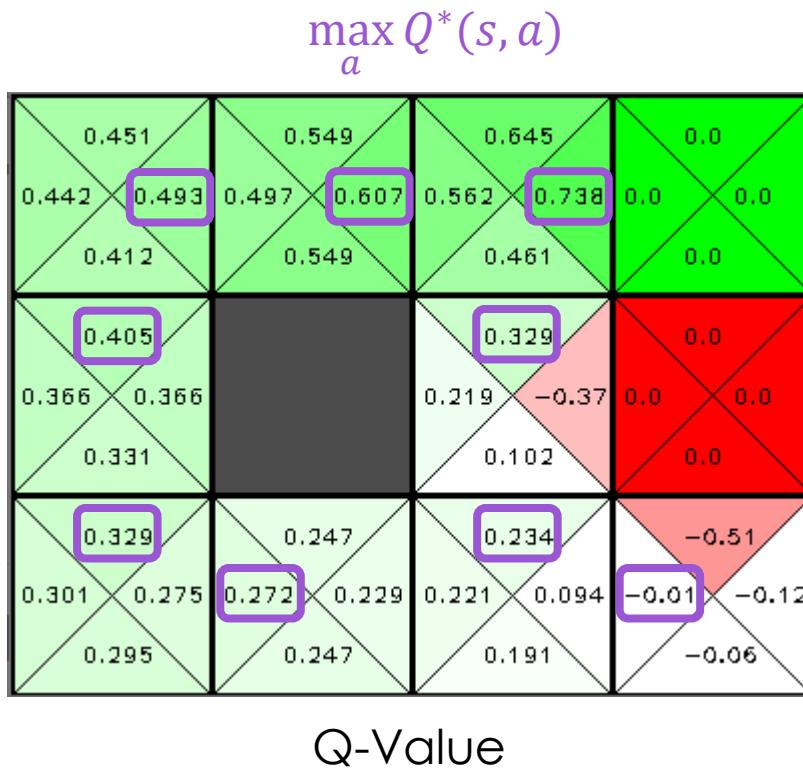
$$\begin{aligned} Q^\pi(s_t) &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^T r_T | s_t = s, a_t = a] \\ &= E_\pi[G_t | s_t = s, a_t = a] \end{aligned}$$

$$\begin{aligned} Q(s, a) &= E_\pi[G_t | s_t = s, a_t = a] \\ &= E_\pi[r_{t+1} + \gamma G_{t+1} | s_t = s, a_t = a] \\ &= E_{s' \sim P(s'|s,a)}[r_{t+1} + \gamma E_\pi[G_{t+1} | s_{t+1} = s'] | s_t = s, a_t = a] \\ &= E_{s' \sim P(s'|s,a)}[r_{t+1} + \gamma V(s') | s_t = s, a_t = a] \\ &= \sum_{s'} P(s'|s, a)[r(s, a) + \gamma V(s')] \end{aligned}$$



Action-Value Function (Q-function)

- If we have the action-value function, we can design a policy to choose an action with maximum action-value.



Bellman Expectation Equation

- Bellman Expectation Equation is the recursive form of value functions.
- State-Value

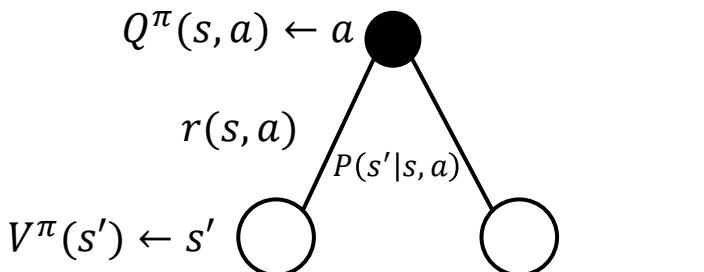
$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^T r_T | s_t = s] \\ &= \sum_a \pi(a|s) Q^\pi(s, a) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a) [r(s, a) + \gamma V^\pi(s')] \end{aligned}$$

- Action-Value

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^T r_T | s_t = s, a_t = a] \\ &= \sum_{s'} P(s'|s, a) [r(s, a) + \gamma V^\pi(s')] = \sum_{s'} P(s'|s, a) \left[r(s, a) + \gamma \sum_{a'} \pi(a'|s') Q(s', a') \right] \end{aligned}$$

Recursive Relation of Value Functions

$$V^\pi(s) \leftarrow s$$
$$Q^\pi(s, a) \leftarrow a$$
$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s, a)$$

$$Q^\pi(s, a) \leftarrow a$$

$$r(s, a)$$
$$V^\pi(s') \leftarrow s'$$
$$P(s'|s, a)$$
$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s')$$

$$V^\pi(s) \leftarrow s$$
$$\pi(a|s)$$
$$r(s, a)$$
$$P(s'|s, a)$$
$$V^\pi(s') \leftarrow s'$$
$$V^\pi(s) = \sum_a \pi(a|s) [r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s')]$$

$$Q^\pi(s, a) \leftarrow a$$
$$r(s, a)$$
$$P(s'|s, a)$$
$$\pi(a'|s')$$
$$Q^\pi(s', a') \leftarrow a'$$
$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') Q^\pi(s', a')$$

Bellman Optimality Equation

- Bellman Optimality Equation is the recursive form of optimal value functions.
- State-Value

$$\begin{aligned} V^*(s) &= \max_a Q^*(s, a) \\ &= \max_a \left\{ r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right\} \end{aligned}$$

$$\begin{aligned} V^\pi(s) &= \sum_a \pi(a|s) Q^\pi(s, a) \\ &= \sum_a \pi(a|s) \sum_{s'} P(s'|s, a) [r(s, a) + \gamma V^\pi(s')] \end{aligned}$$

- Action-Value

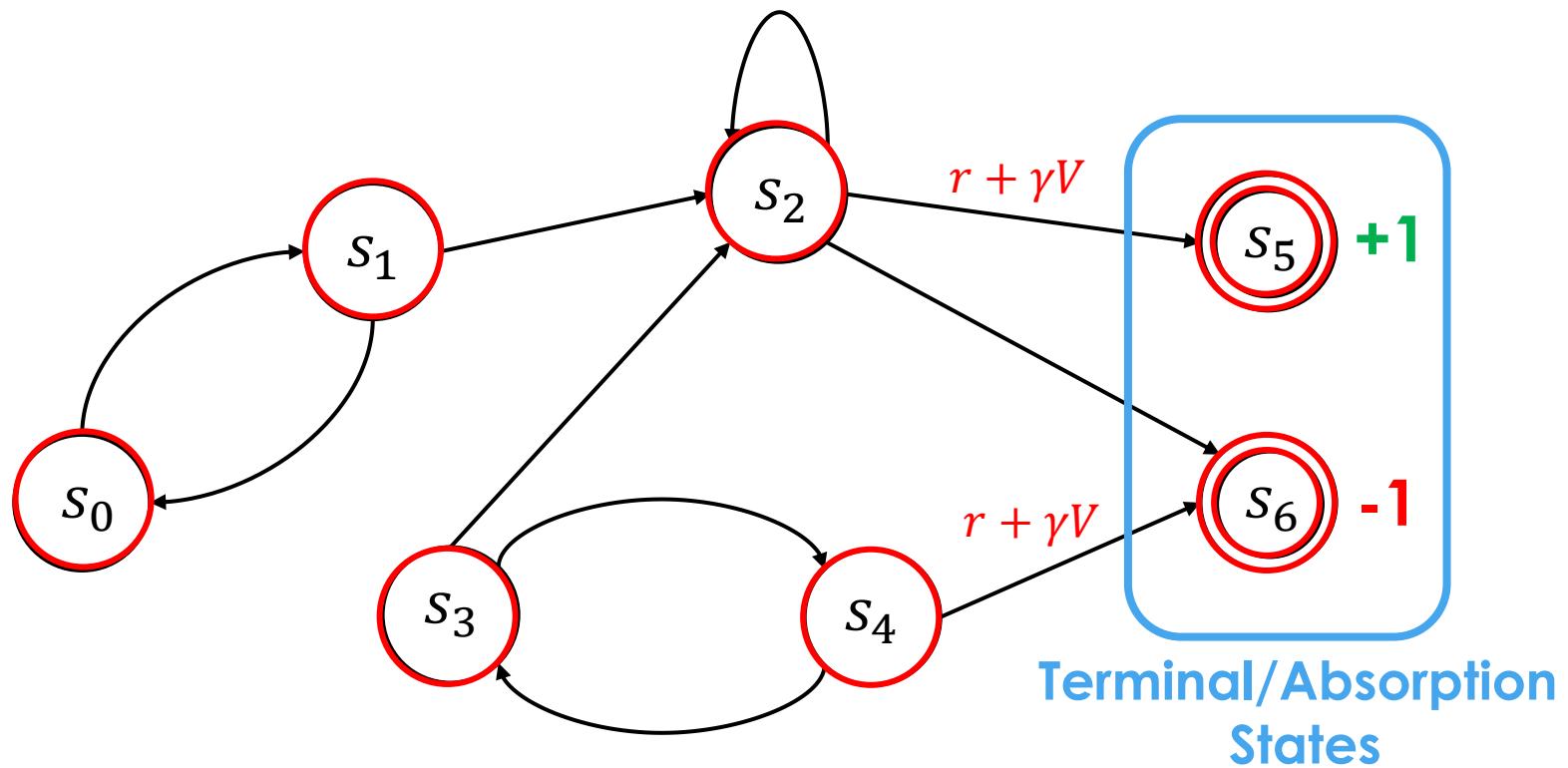
$$\begin{aligned} Q^*(s) &= r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \\ &= r(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_a Q^*(s, a) \end{aligned}$$

$$\begin{aligned} Q^\pi(s, a) &= \sum_{s'} P(s'|s, a) [r(s, a) + \gamma V^\pi(s')] \\ &= \sum_{s'} P(s'|s, a) \left[r(s, a) + \gamma \sum_{a'} \pi(a'|s') Q(s', a') \right] \end{aligned}$$

Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- Value-based Reinforcement Learning
 - From MDP to RL
 - Temporal Difference Learning (SARSA / Q-Learning)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- Policy-based Reinforcement Learning
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

Dynamic Programming



Bellman Expectation Backup

- Define Bellman **Expectation** Backup Operator: \mathcal{T}^π

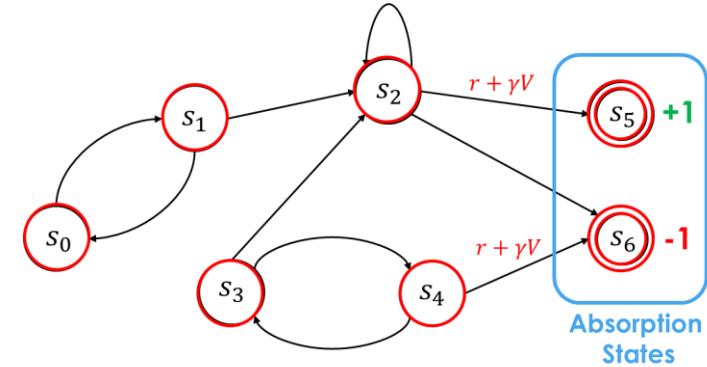
$$\mathcal{T}^\pi V(s_t) \triangleq \mathbb{E}_{s_{t+1} \sim P}[r(s_t, a_t) + \gamma V(s_{t+1})]$$

$$\mathcal{T}^\pi Q(s_t, a_t) \triangleq \mathbb{E}_{s_{t+1} \sim P}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q(s_{t+1}, a_{t+1})]]$$

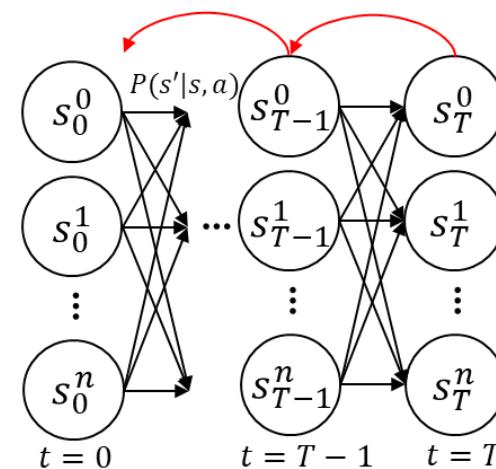
- Convergence of Bellman Expectation Backup

$$V \rightarrow \mathcal{T}^\pi V \rightarrow (\mathcal{T}^\pi)^2 V \rightarrow (\mathcal{T}^\pi)^3 V \rightarrow \dots \rightarrow V^\pi$$

$$Q \rightarrow \mathcal{T}^\pi Q \rightarrow (\mathcal{T}^\pi)^2 Q \rightarrow (\mathcal{T}^\pi)^3 Q \rightarrow \dots \rightarrow Q^\pi$$



$$V(s_0) = r_0 + \gamma(r_1 + \gamma(\dots(r_{T-1} + \gamma V(s_T))))$$



Bellman Optimality Backup

- If we operate the Bellman **Optimality** Backups to V^* and Q^*

$$\mathcal{T}^*V(s_t) \triangleq \max_a \mathbb{E}_{s_{t+1} \sim P}[r(s_t, a_t) + \gamma V(s_{t+1})]$$

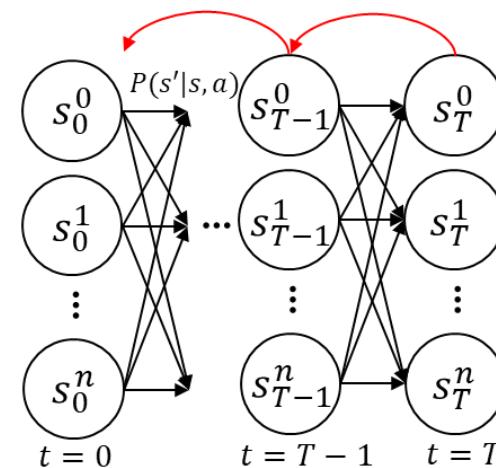
$$\mathcal{T}^*Q(s_t, a_t) \triangleq \mathbb{E}_{s_{t+1} \sim P} \left[r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a_{t+1}) \right]$$

- Convergence of Bellman Optimality Backup

$$V \rightarrow \mathcal{T}^*V \rightarrow (\mathcal{T}^*)^2V \rightarrow (\mathcal{T}^*)^3V \rightarrow \dots \rightarrow V^*$$

$$Q \rightarrow \mathcal{T}^*Q \rightarrow (\mathcal{T}^*)^2Q \rightarrow (\mathcal{T}^*)^3Q \rightarrow \dots \rightarrow Q^*$$

$$V(s_0) = r_0 + \gamma(r_1 + \gamma(\dots(r_{T-1} + \gamma V(s_T))))$$



Convergence Proof

- Convergence of Bellman **Expectation** Backup

$$\mathcal{T}^\pi V(s_t) \triangleq \mathbb{E}_{s_{t+1} \sim P, a_t \sim \pi}[r(s_t, a_t) + \gamma V(s_{t+1})] = R^\pi + \gamma P^\pi V$$

$$\begin{aligned}\|\mathcal{T}^\pi V - V^\pi\|_\infty &= \|\mathcal{T}^\pi V - \mathcal{T}^\pi V^\pi\|_\infty = \|(R^\pi + \gamma P^\pi V) - (R^\pi + \gamma P^\pi V^\pi)\|_\infty \\ &= \|\gamma P^\pi(V - V^\pi)\|_\infty \leq \gamma \|V - V^\pi\|_\infty\end{aligned}$$

maximum element
transition probability, ≤ 1

- Convergence of Bellman **Optimality** Backup

$$\begin{aligned}\mathcal{T}^* V(s_t) &\triangleq \max_a \mathbb{E}_{s_{t+1} \sim P}[r(s_t, a_t) + \gamma V(s_{t+1})] = \max_a \{R + \gamma P^a V\} \\ \|\mathcal{T}^* V - V^*\|_\infty &= \|\mathcal{T}^* V - \mathcal{T}^* V^*\|_\infty = \left\| \max_a \{R + \gamma P^a V\} - \max_a \{R + \gamma P^a V^*\} \right\|_\infty \\ &\leq \max_a \| \{R + \gamma P^a V\} - \{R + \gamma P^a V^*\} \|_\infty = \gamma \left\| \max_a \{P^a\} (V - V^*) \right\|_\infty \leq \gamma \|V - V^*\|_\infty\end{aligned}$$

$\left\| \max_x f(x) - \max_x g(x) \right\| \leq \max_x \|f(x) - g(x)\|$

Value Iteration

Bellman Optimality Equation:

$$V^*(s) = \max_a Q^*(s, a) = \max_a \left\{ r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right\}$$

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π , such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

Value Iteration Demo - Initialization



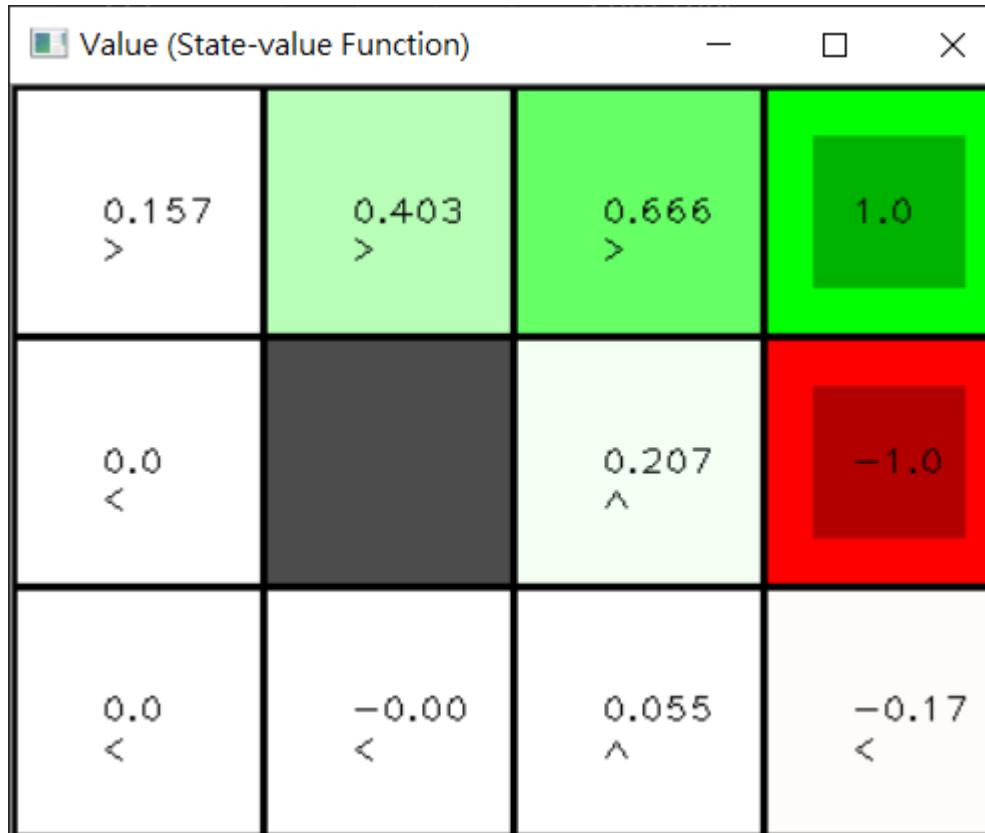
Value Iteration Demo - Iteration 1



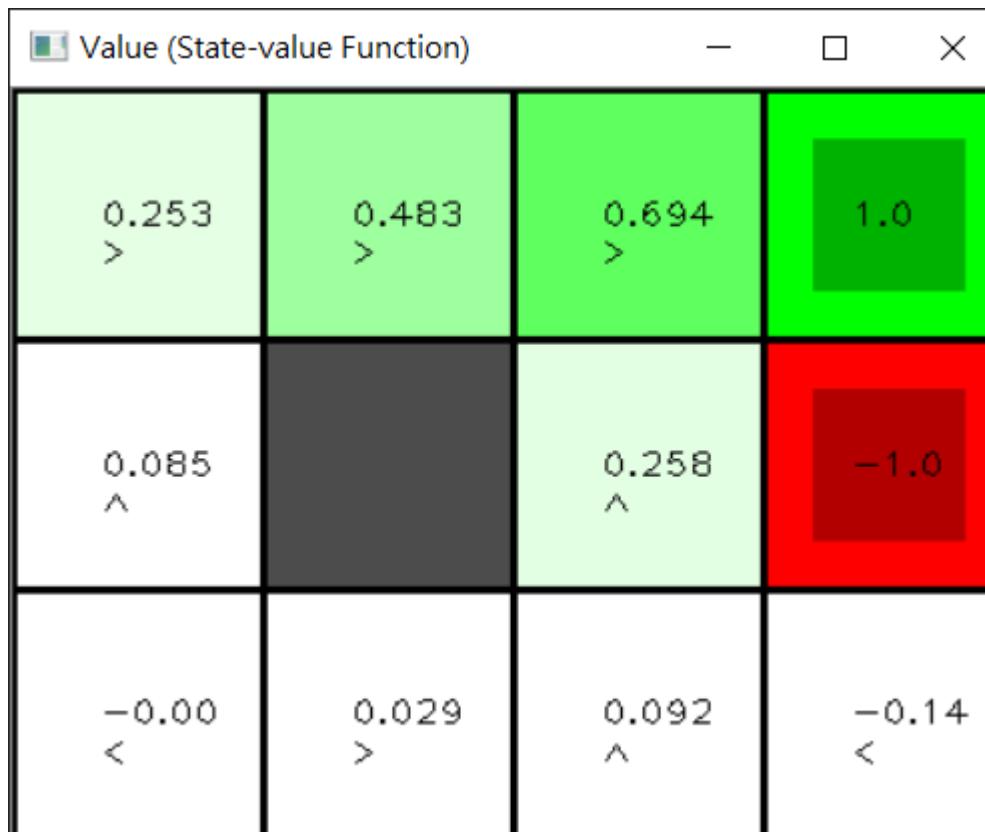
Value Iteration Demo - Iteration 2



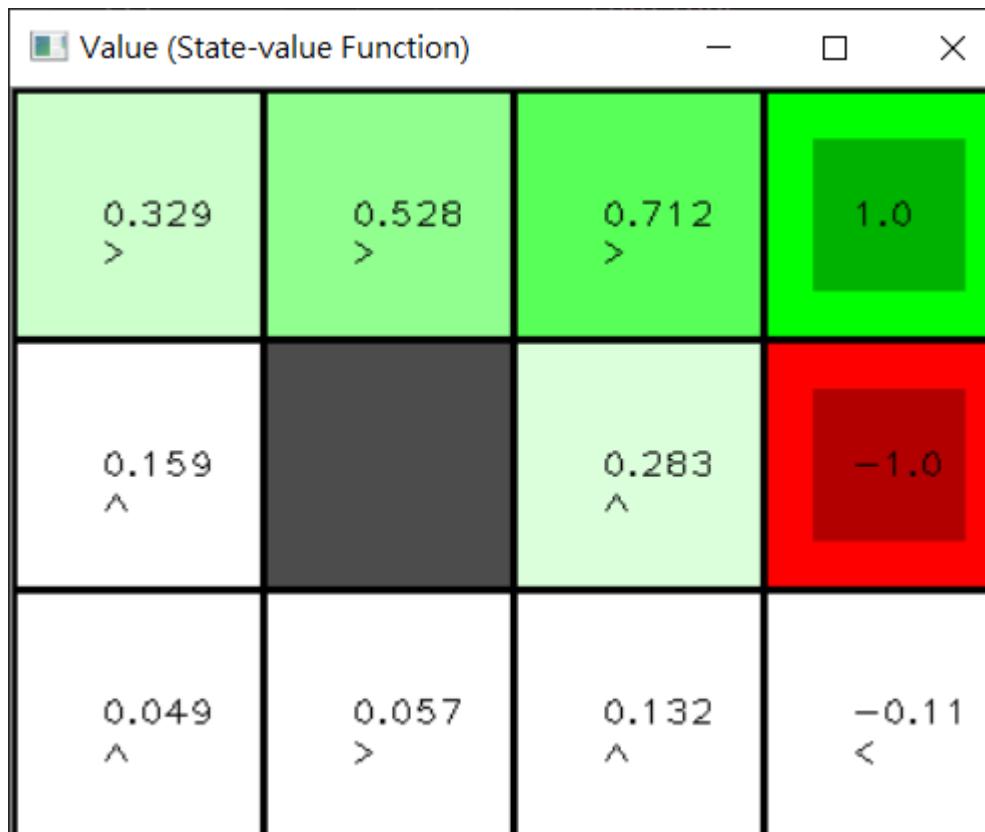
Value Iteration Demo - Iteration 3



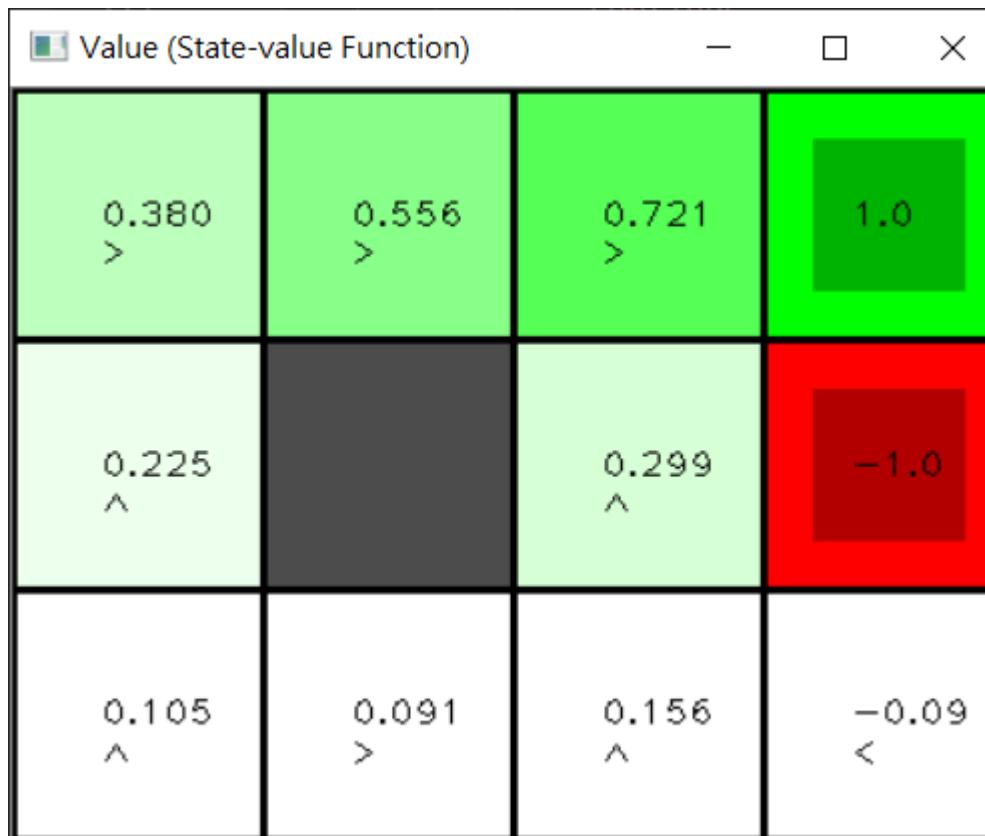
Value Iteration Demo - Iteration 4



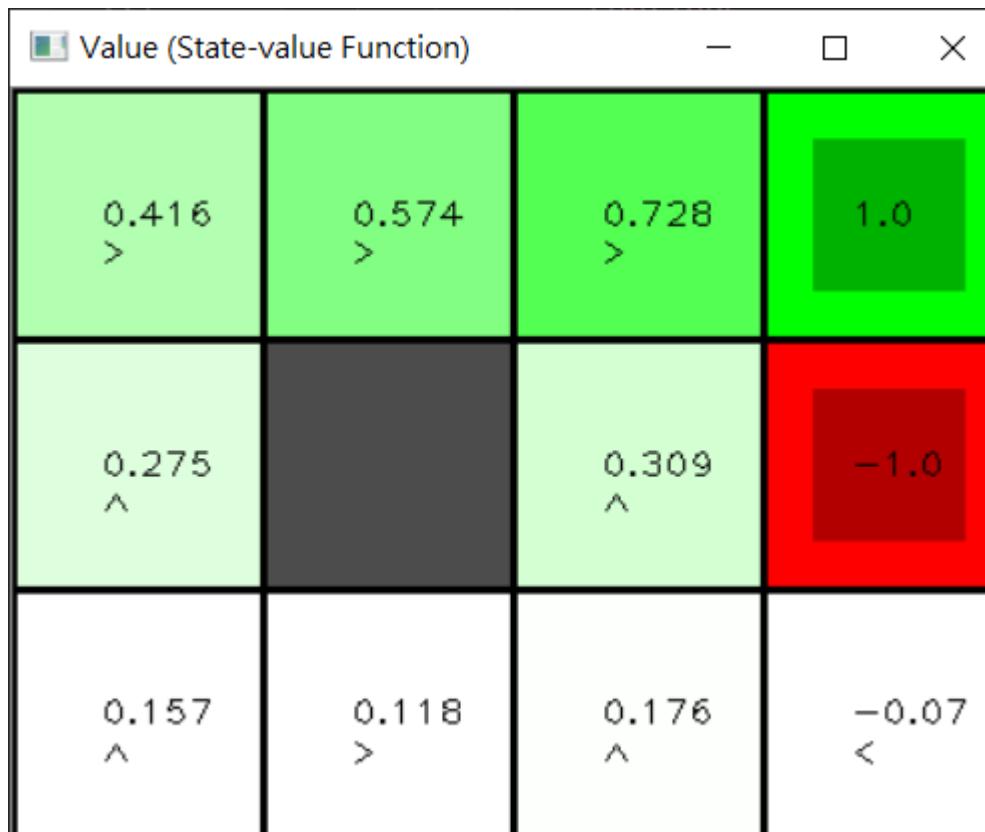
Value Iteration Demo - Iteration 5



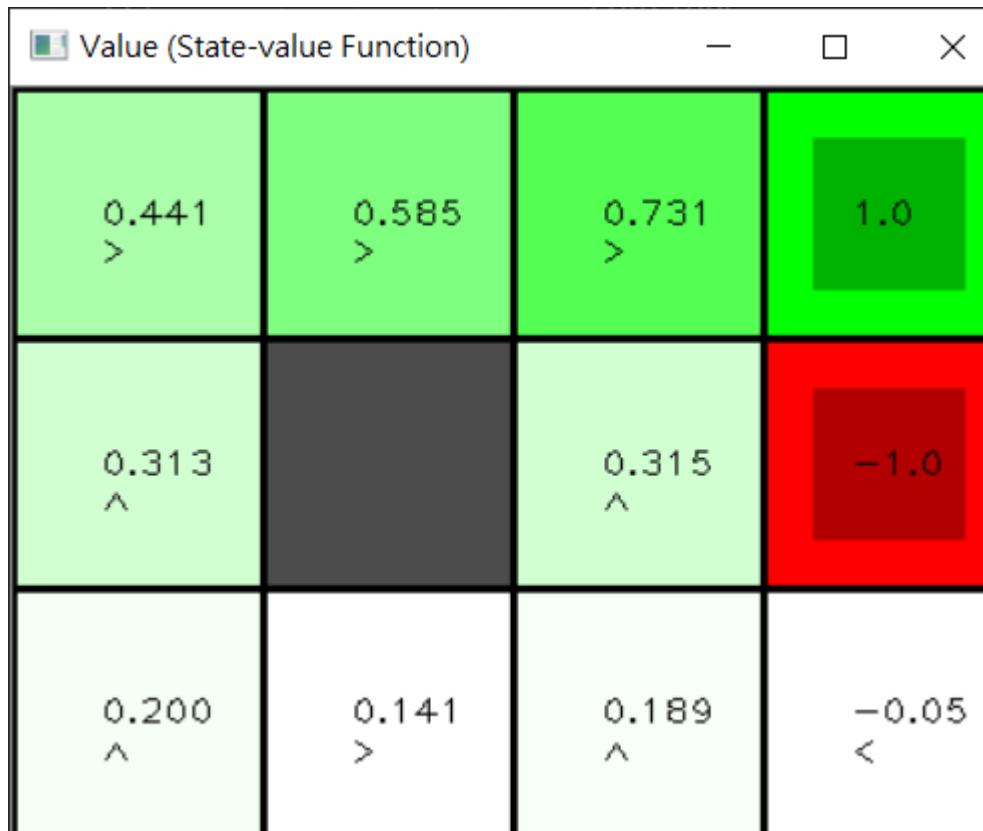
Value Iteration Demo - Iteration 6



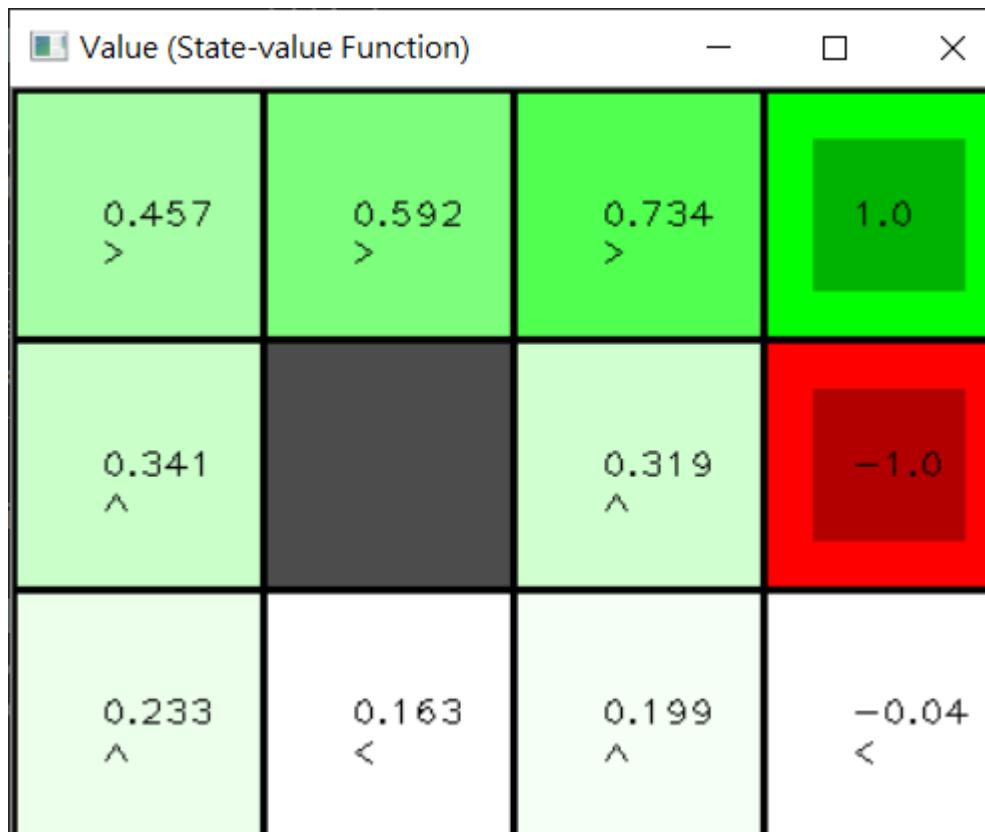
Value Iteration Demo - Iteration 7



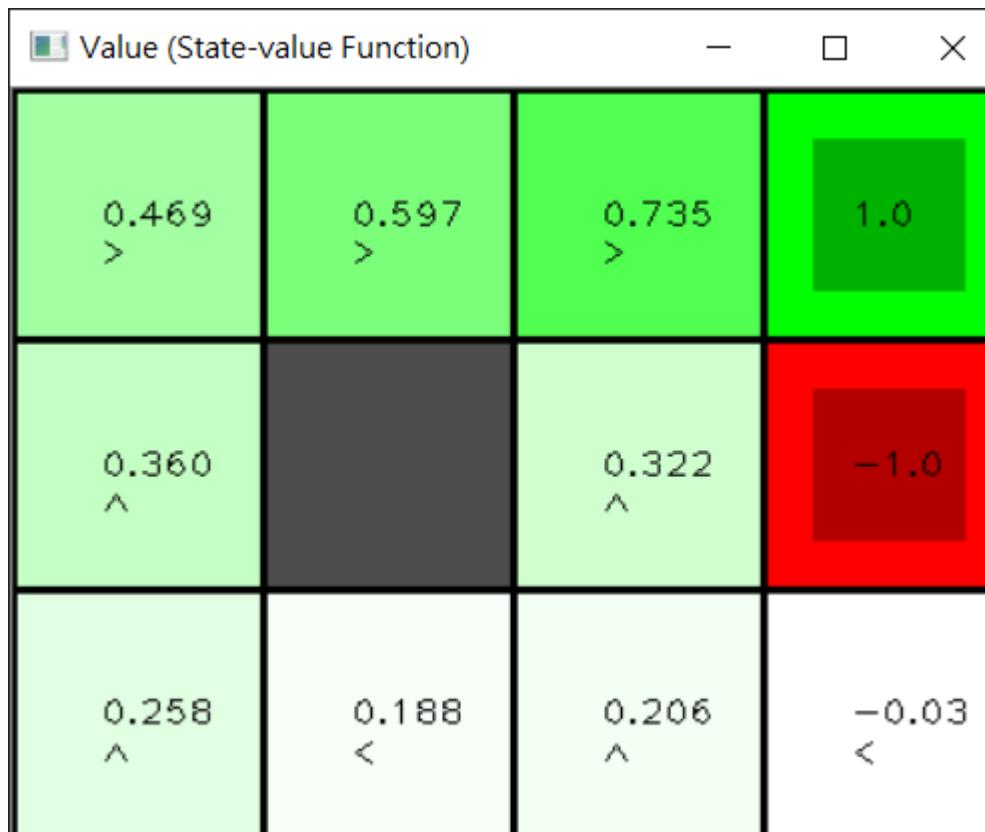
Value Iteration Demo - Iteration 8



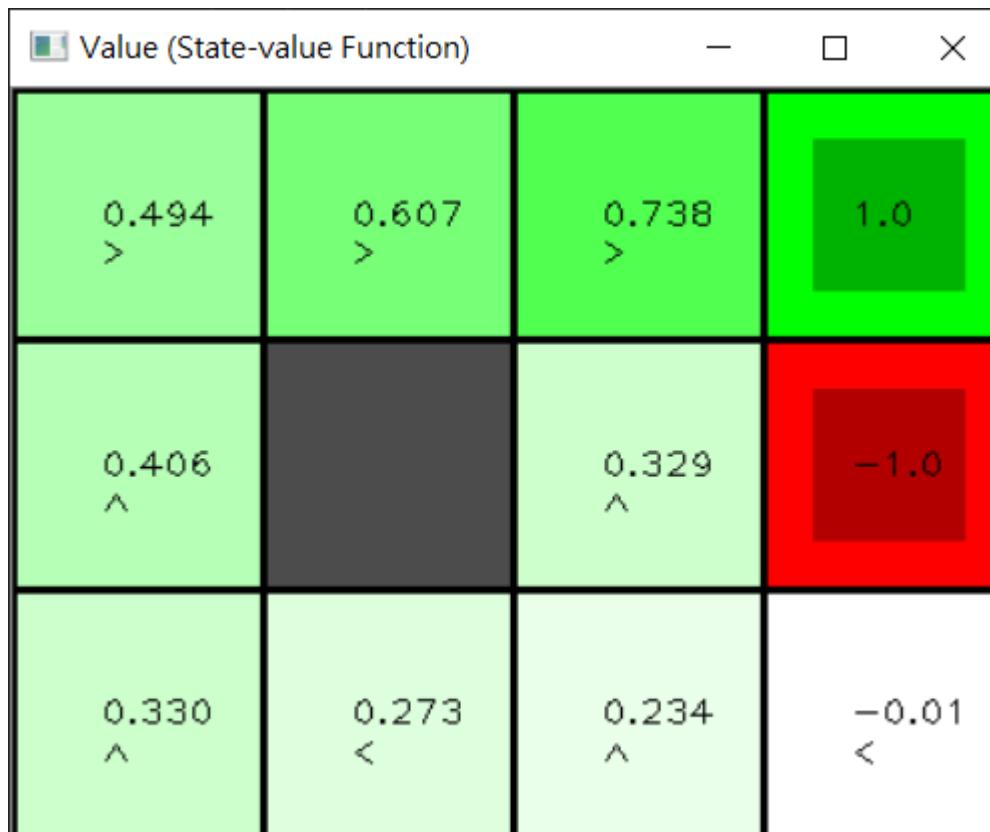
Value Iteration Demo - Iteration 9



Value Iteration Demo - Iteration 10



Value Iteration Demo - Iteration 30



Bellman Expectation Equation:

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s, a) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a) [r(s, a) + \gamma V^\pi(s')]$$

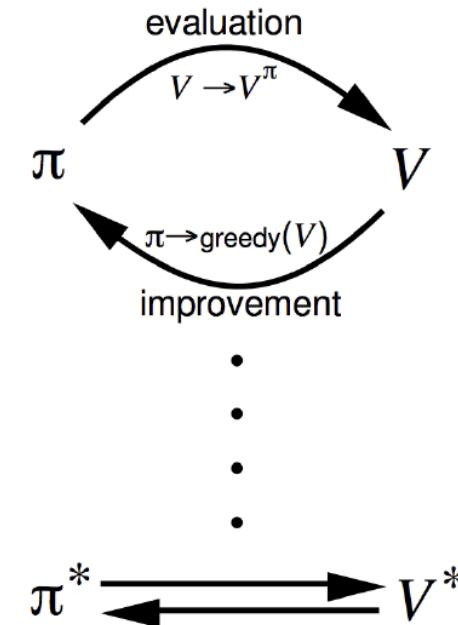
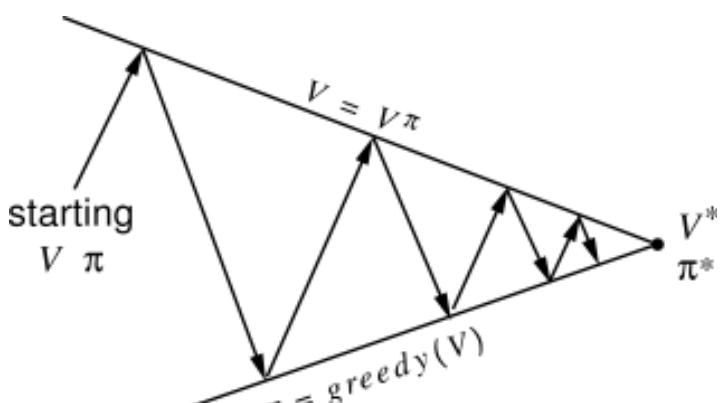
Policy Iteration

- Given a policy π
 - Evaluate the policy π (Policy Evaluation)

$$V^\pi(s_t) = E_\pi[G_t | s_t = s]$$

- Improve the policy by acting greedily with respect to V (Policy Improvement)

$$\pi' = \text{greedy}(V^\pi)$$



Improve the Policy

- Consider a deterministic policy $a = \pi(s)$
- Improve the policy greedily

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

- Improves the value from any state s over one step

$$Q^\pi(s, \pi'(s)) = \max_a Q^\pi(s, a) \geq Q^\pi(s, \pi(s)) = V^\pi(s)$$

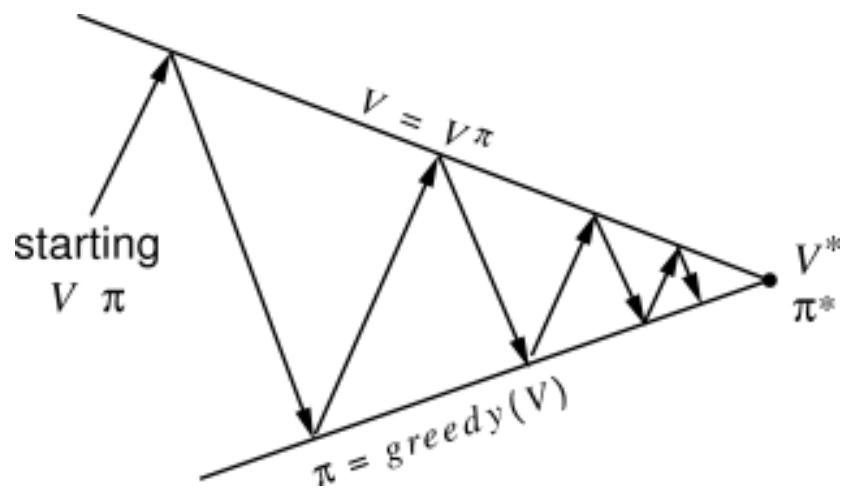
- Proof of improvement

$$\begin{aligned} V^\pi(s) &\leq Q^\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}[r_t + \gamma V^\pi(s_{t+1}) | s_t = s] \\ &\leq \mathbb{E}_{\pi'}[r_t + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) | s_t = s] \\ &\leq \mathbb{E}_{\pi'}[r_t + \gamma r_{t+1} + \gamma^2 Q^\pi(s_{t+2}, \pi'(s_{t+2})) | s_t = s] \\ &\leq \mathbb{E}_{\pi'}[r_t + \gamma r_{t+1} + \dots | s_t = s] \\ &= V^{\pi'}(s) \end{aligned}$$

$Q^\pi(s, \pi'(s)) \geq V^\pi(s)$

$V^\pi(s) \leq V^{\pi'}(s)$

Policy Iteration



1. Initialization

$v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$temp \leftarrow v(s)$$

$$v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) [r(s, \pi(s), s') + \gamma v(s')]$$

$$\Delta \leftarrow \max(\Delta, |temp - v(s)|)$$

until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

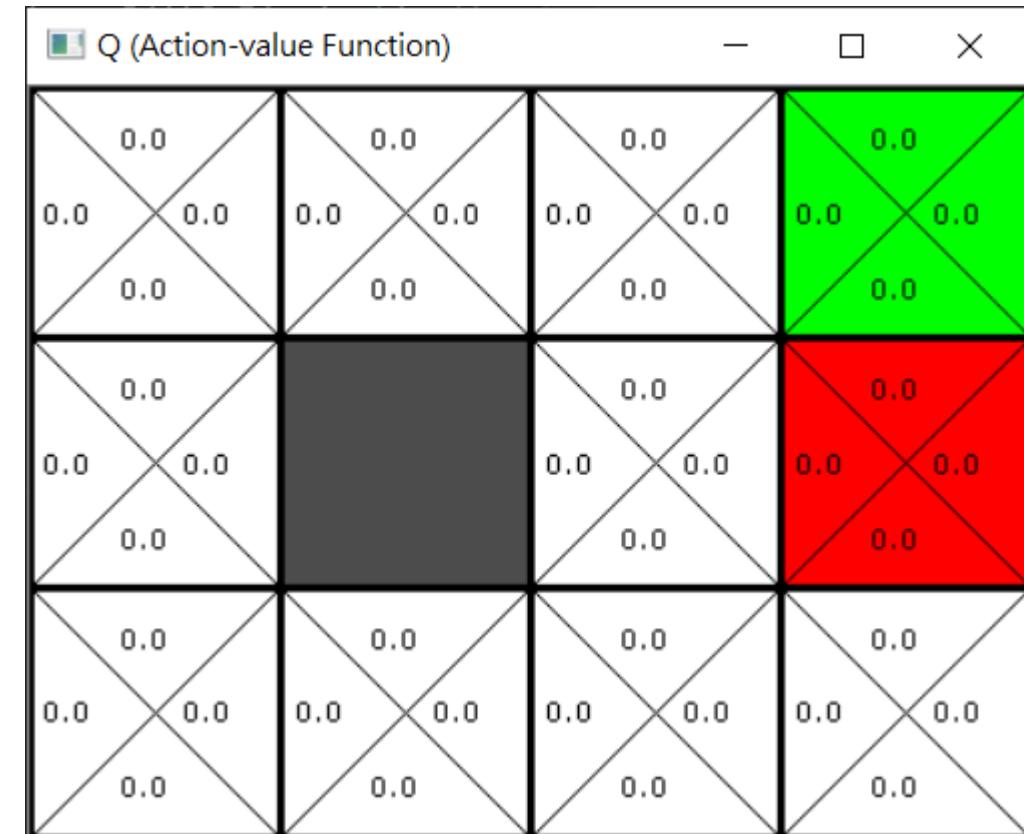
$$temp \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg \max_a \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v(s')]$$

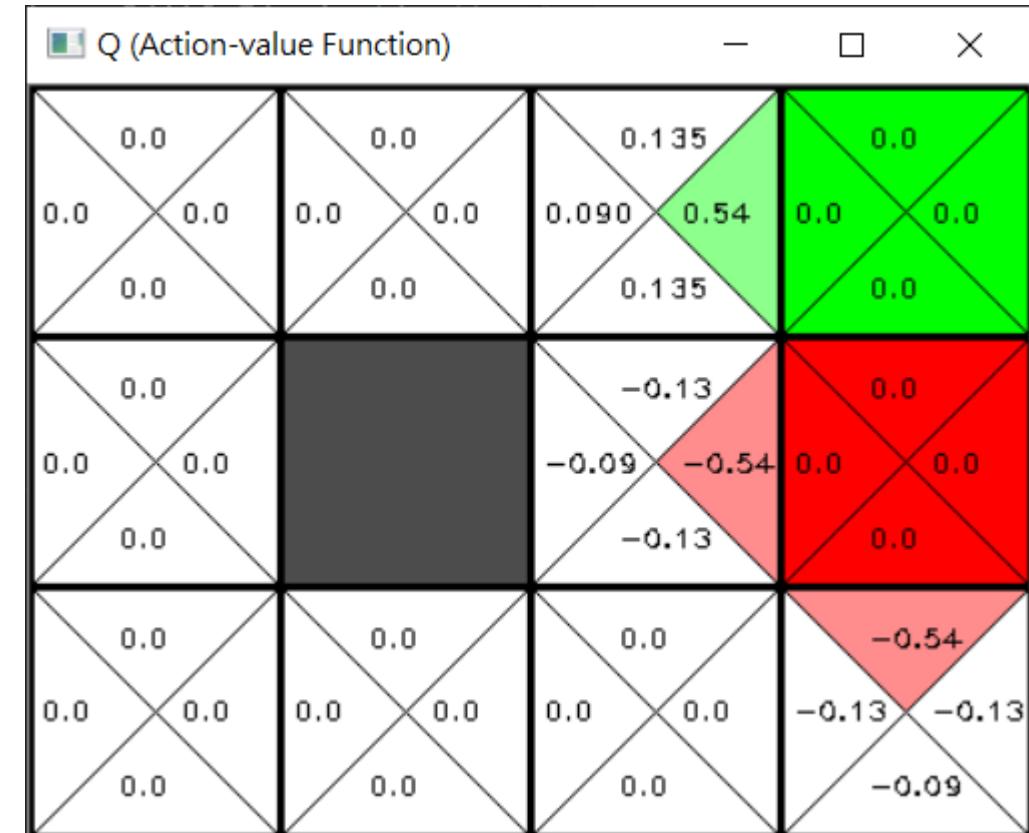
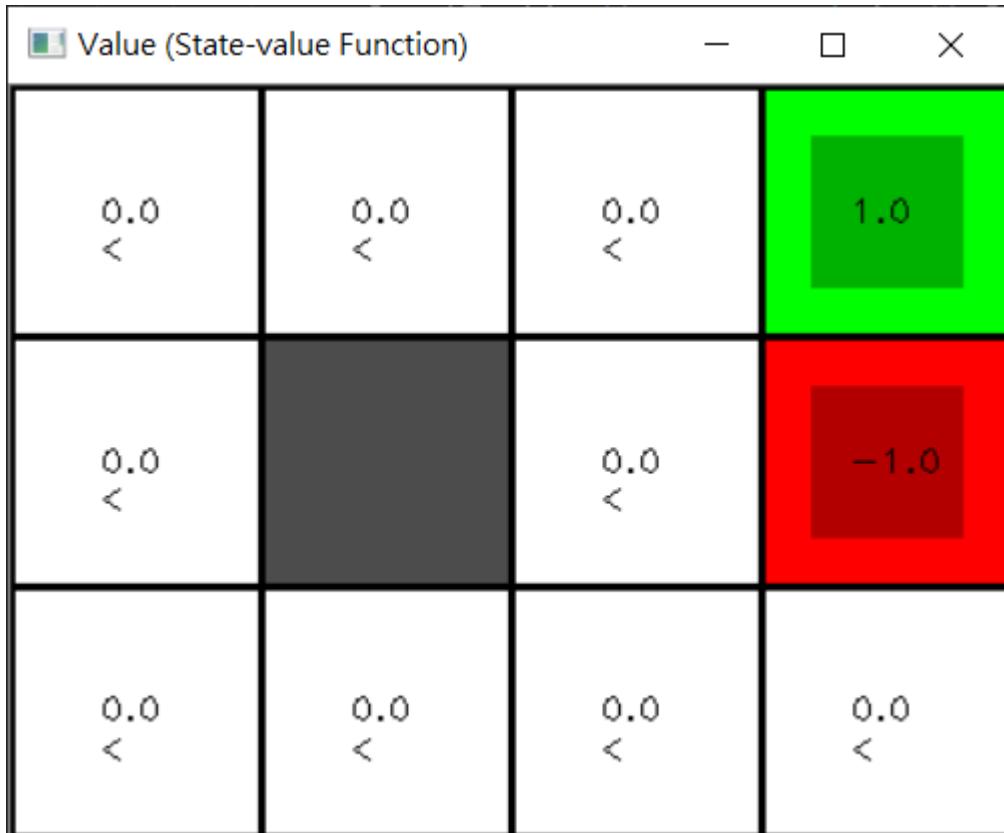
If $temp \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return v and π ; else go to 2

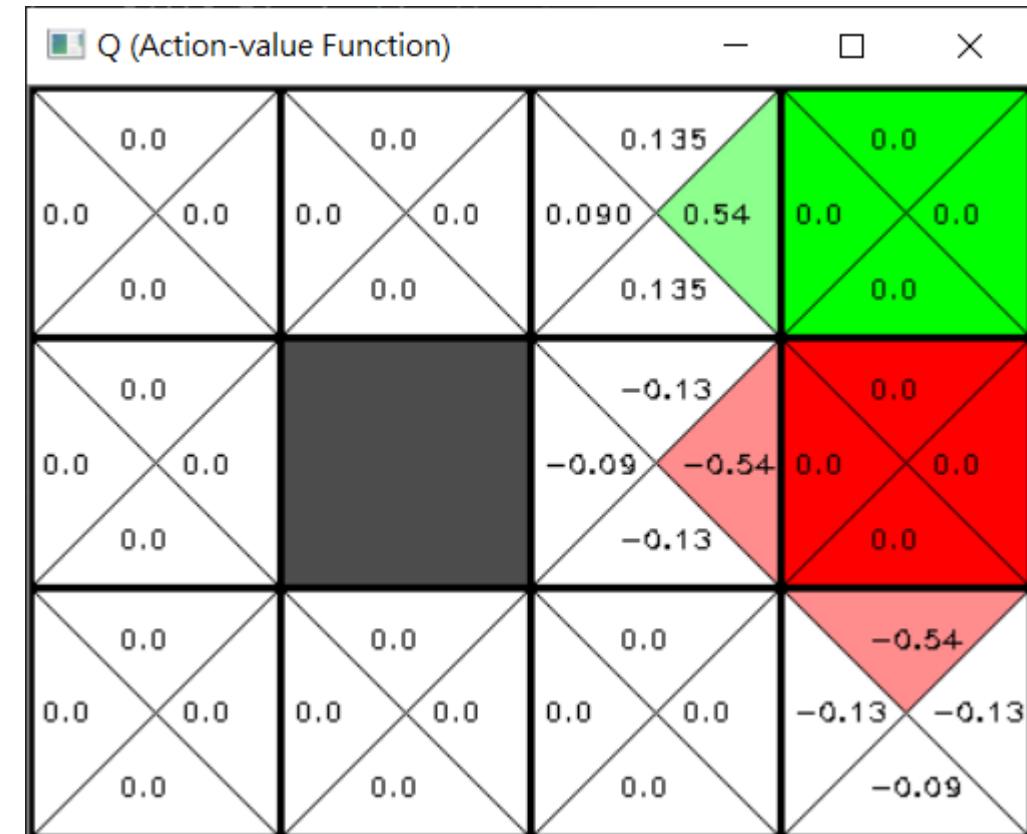
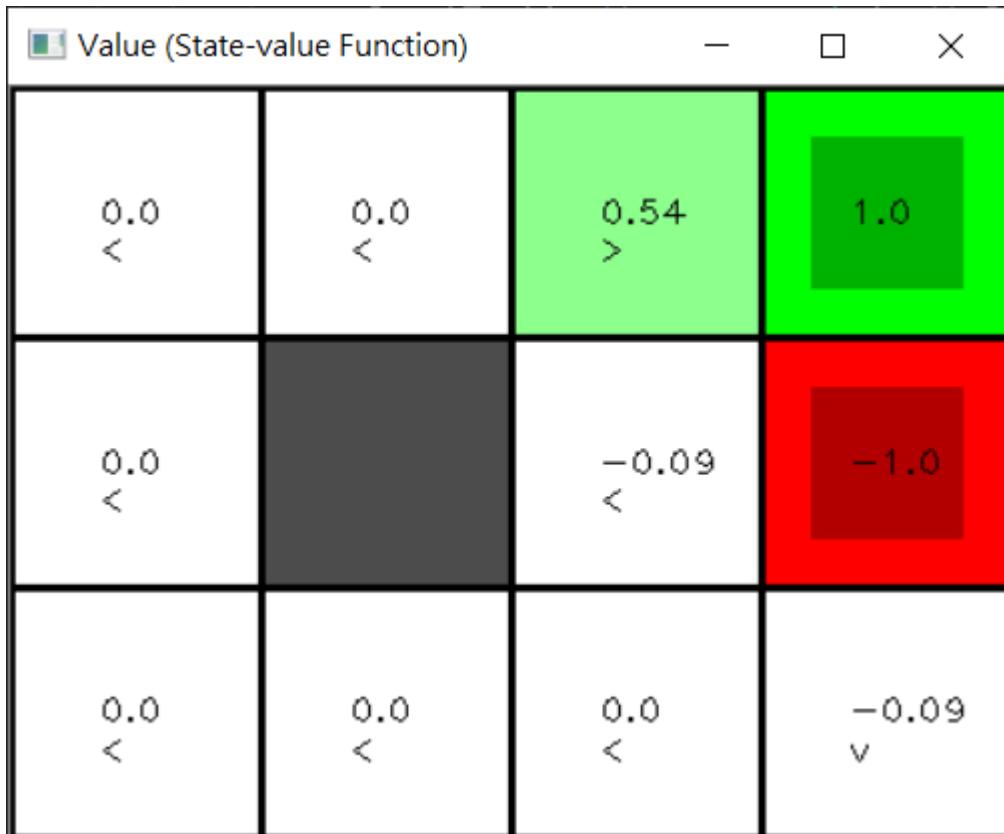
Policy Iteration Demo - Initialization



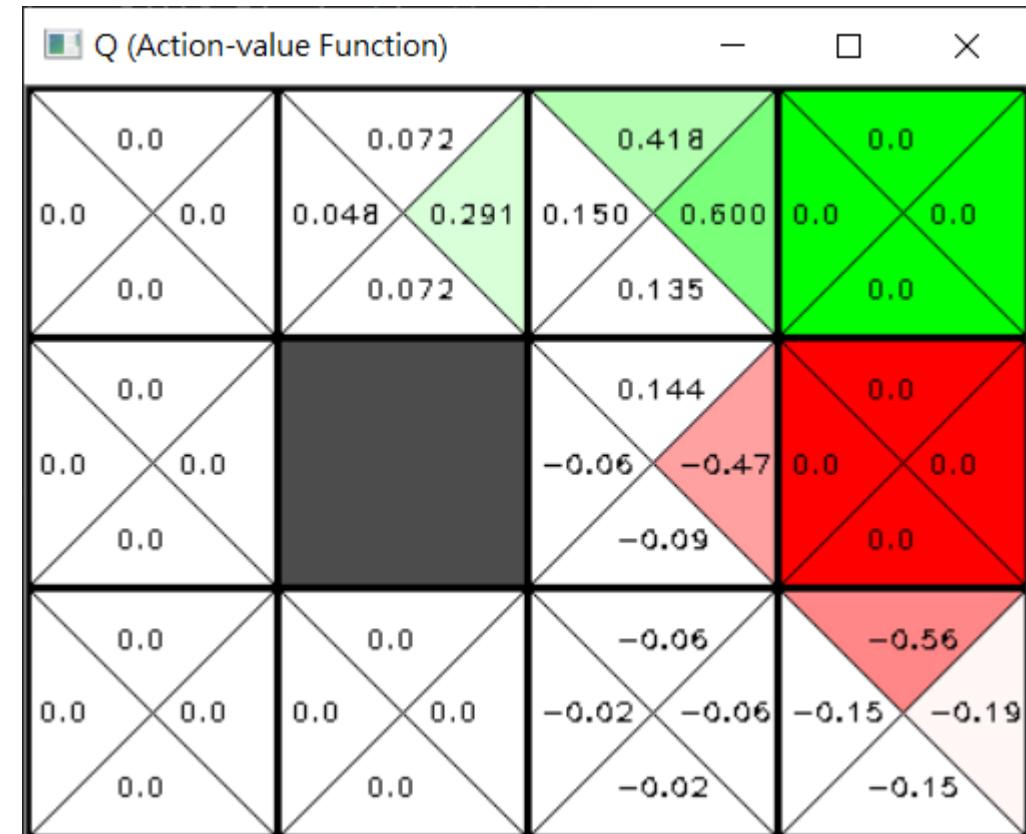
Policy Iteration Demo - Evaluation 1



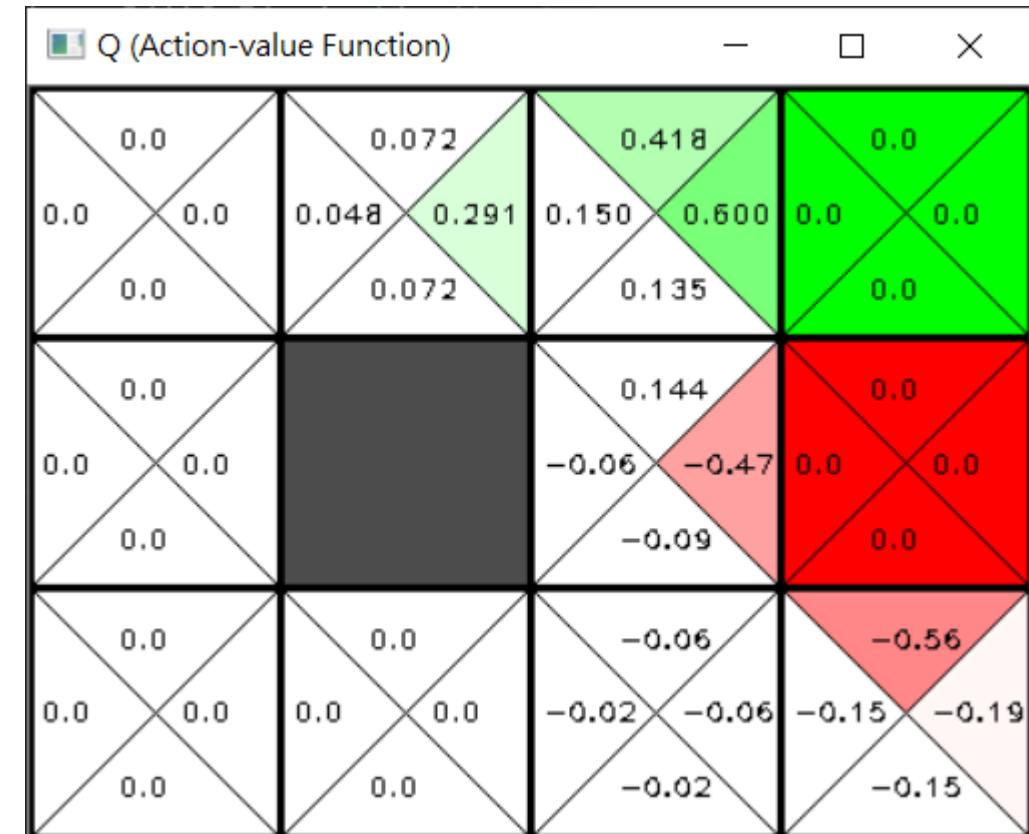
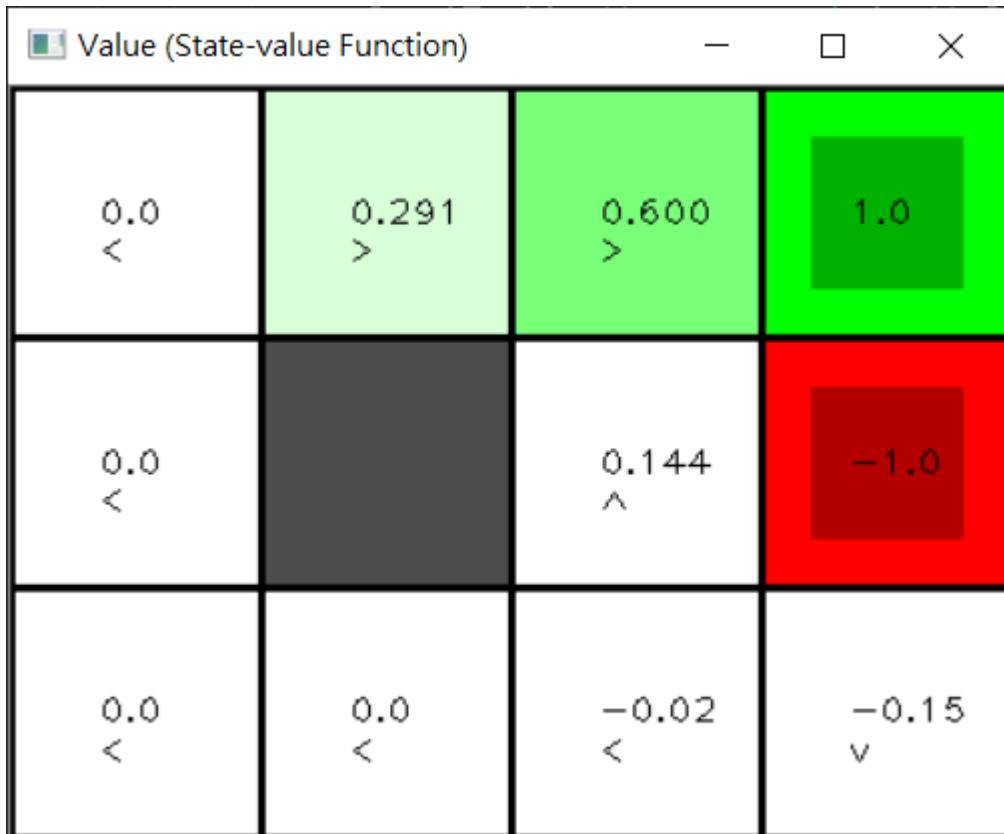
Policy Iteration Demo - Improvement 1



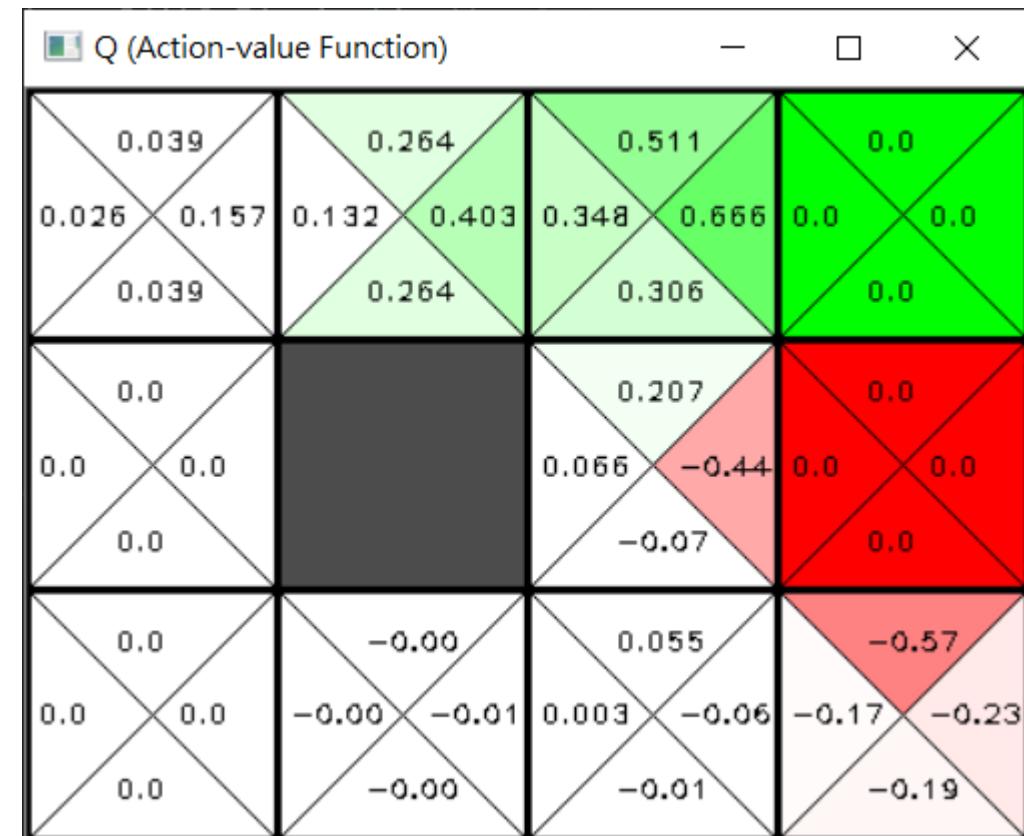
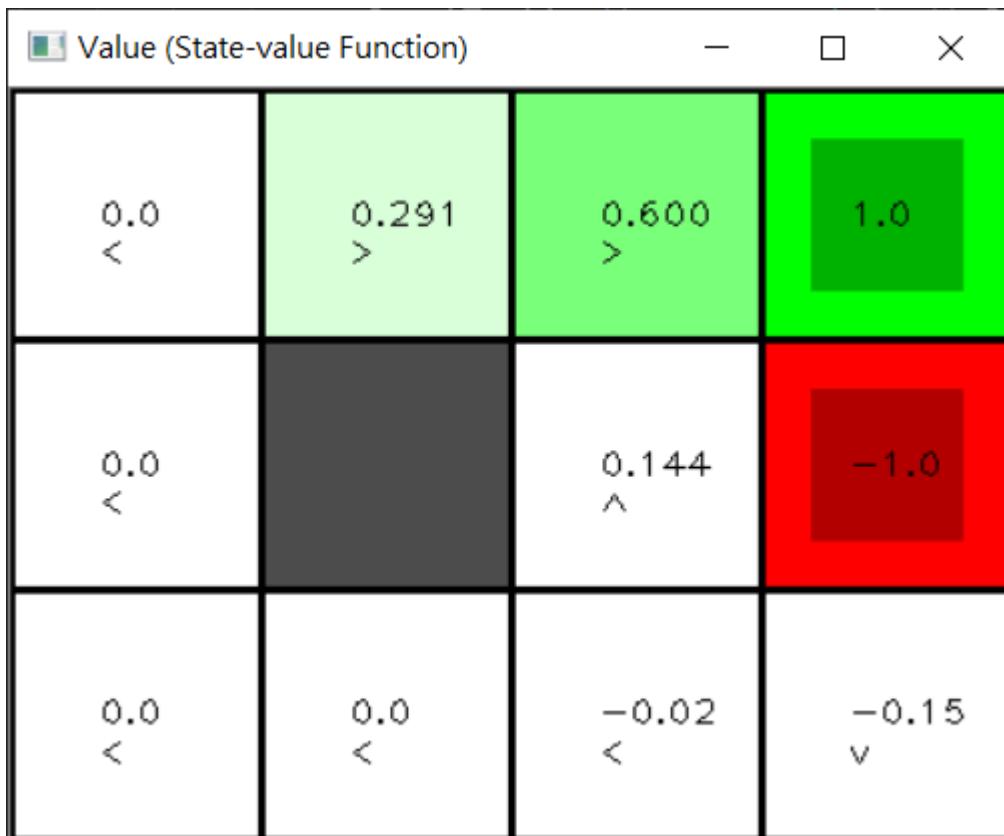
Policy Iteration Demo - Evaluation 2



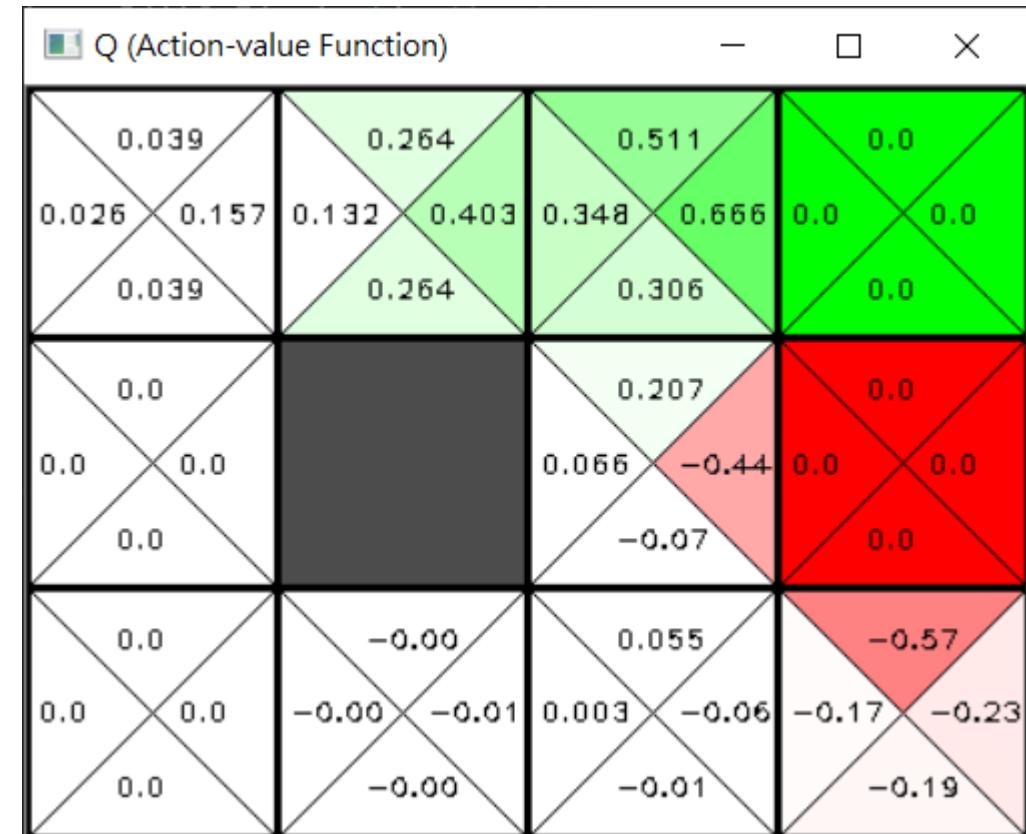
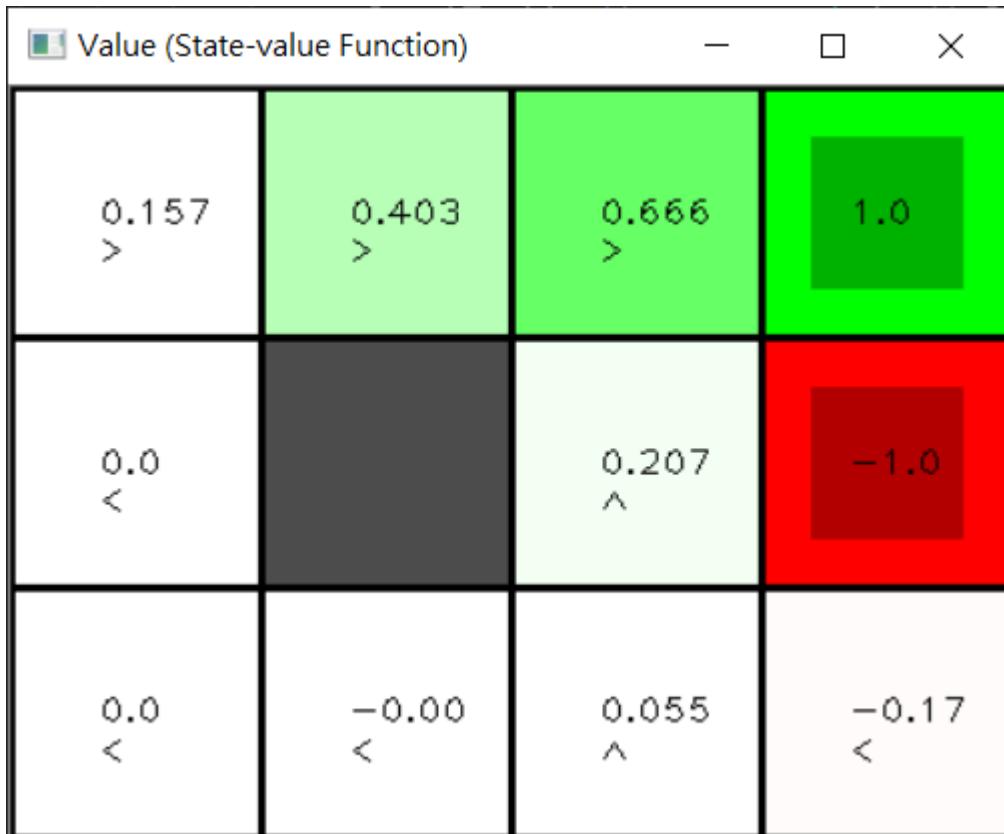
Policy Iteration Demo - Improvement 2



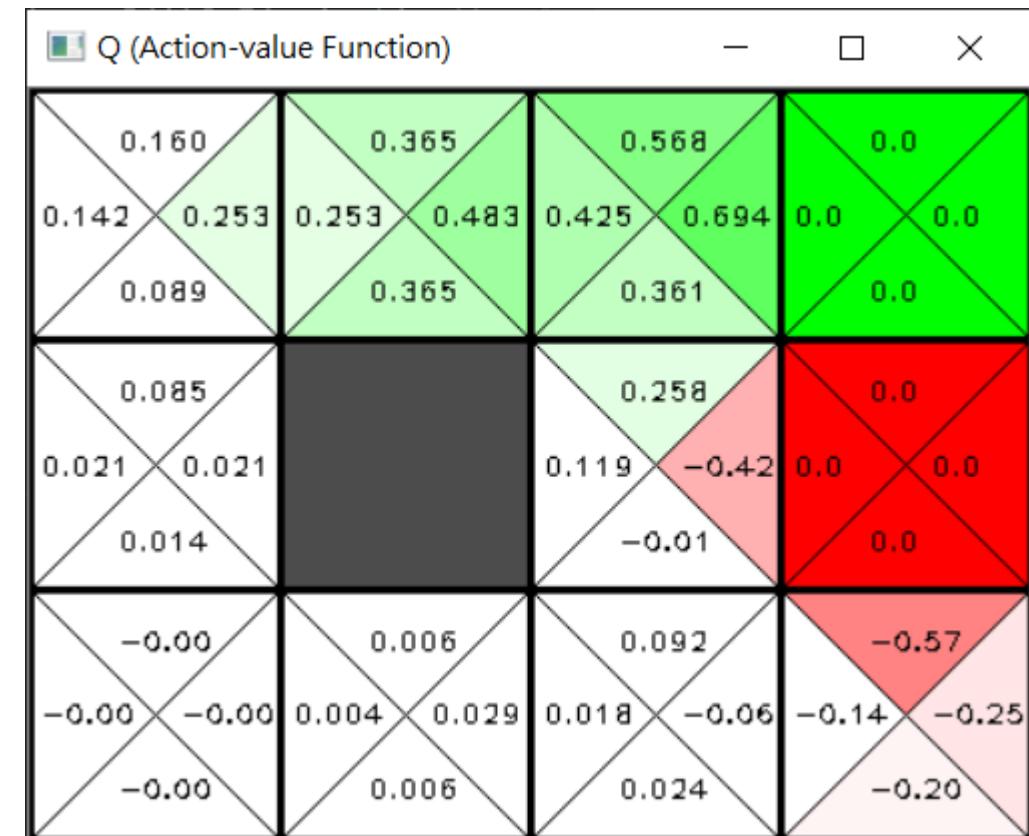
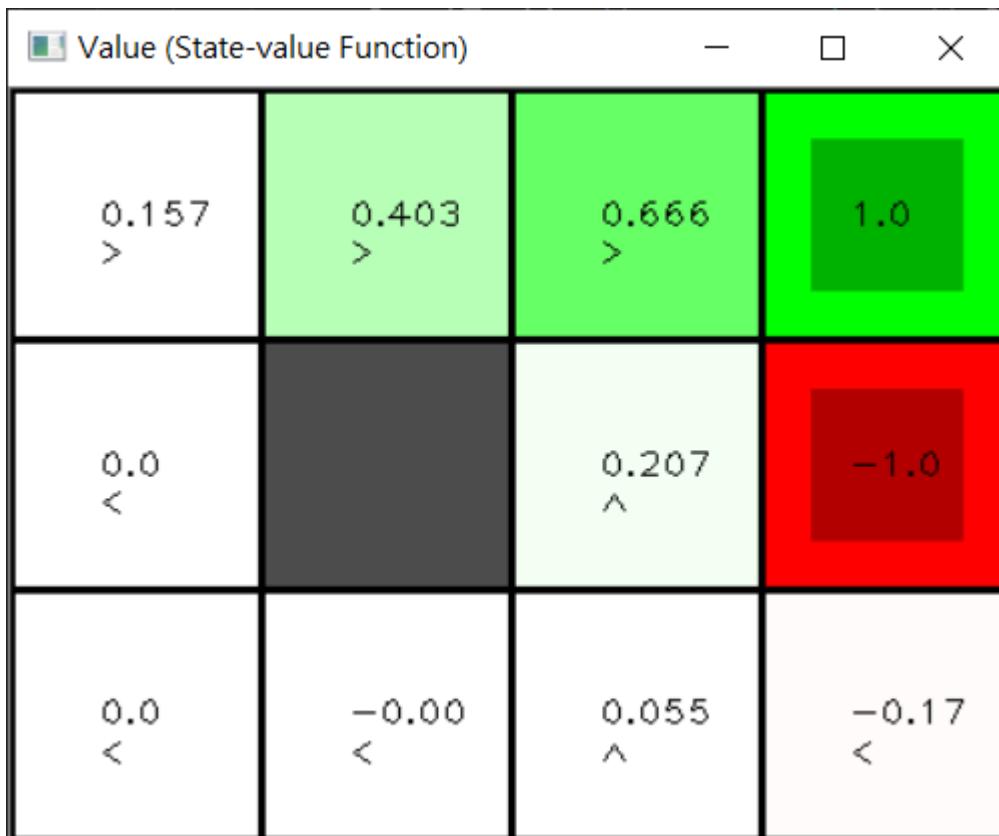
Policy Iteration Demo - Evaluation 3



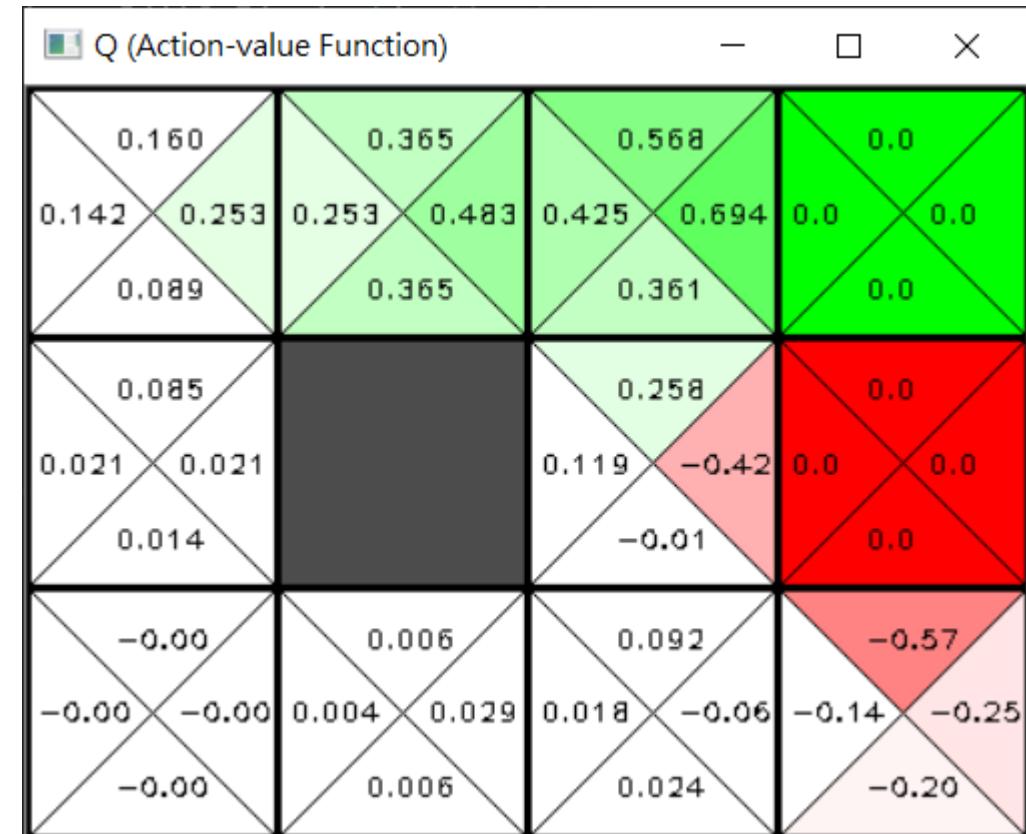
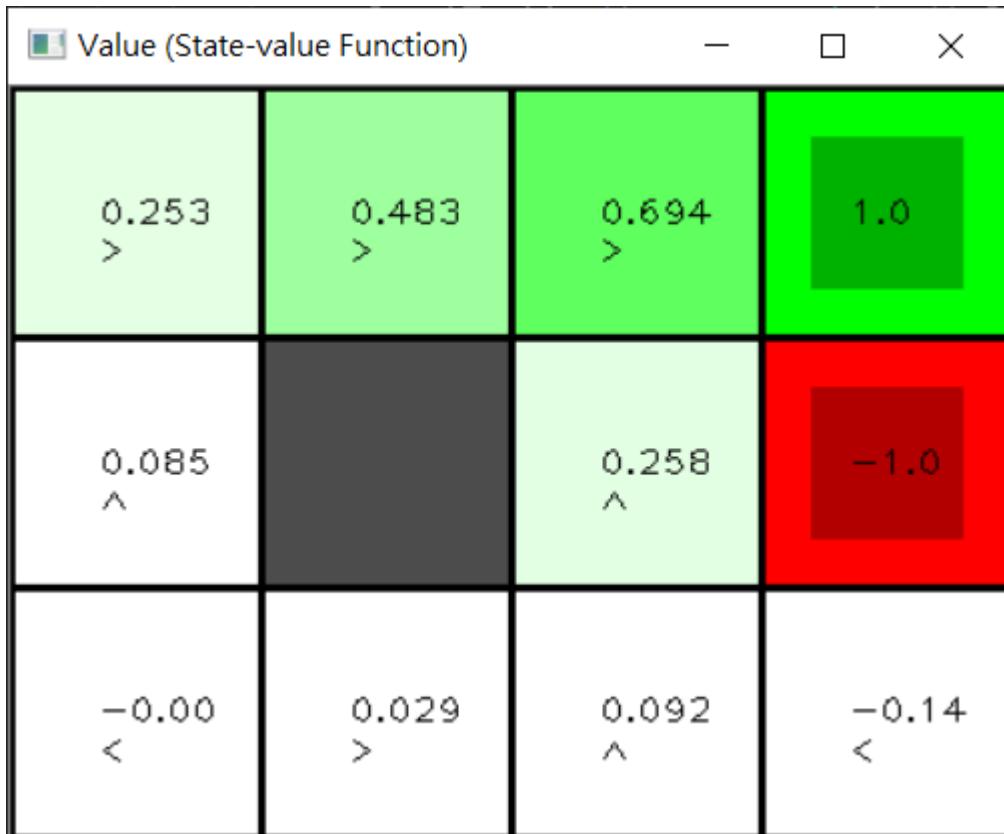
Policy Iteration Demo - Improvement 3



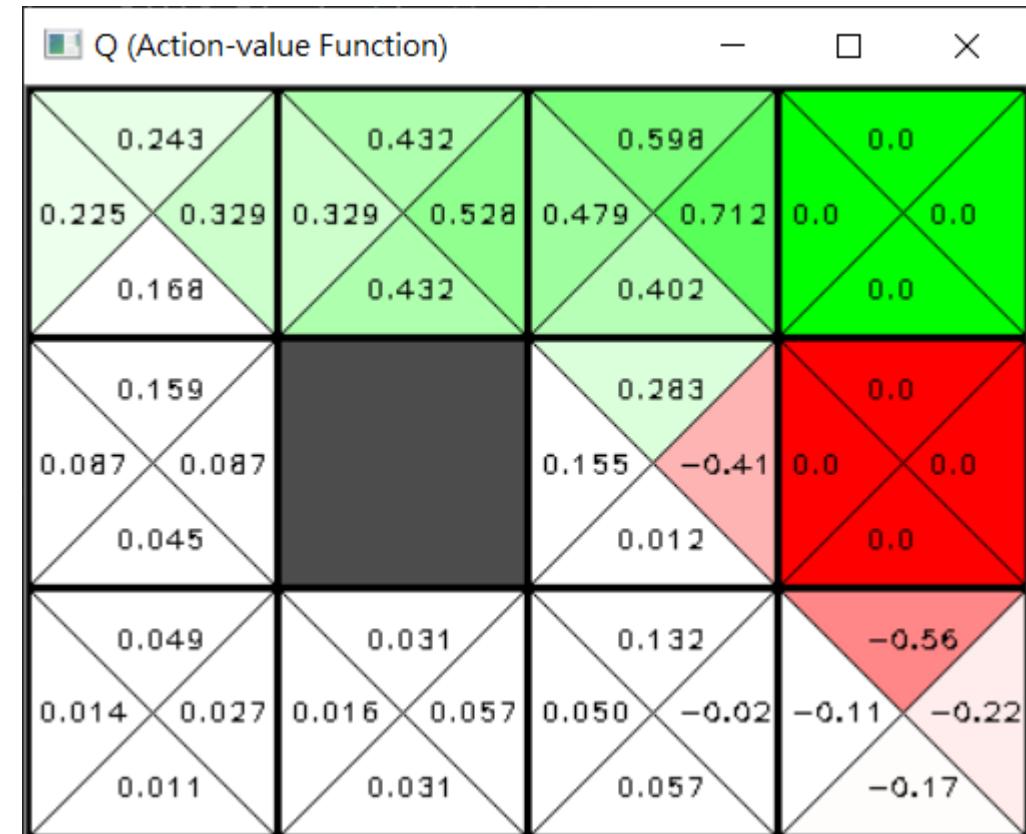
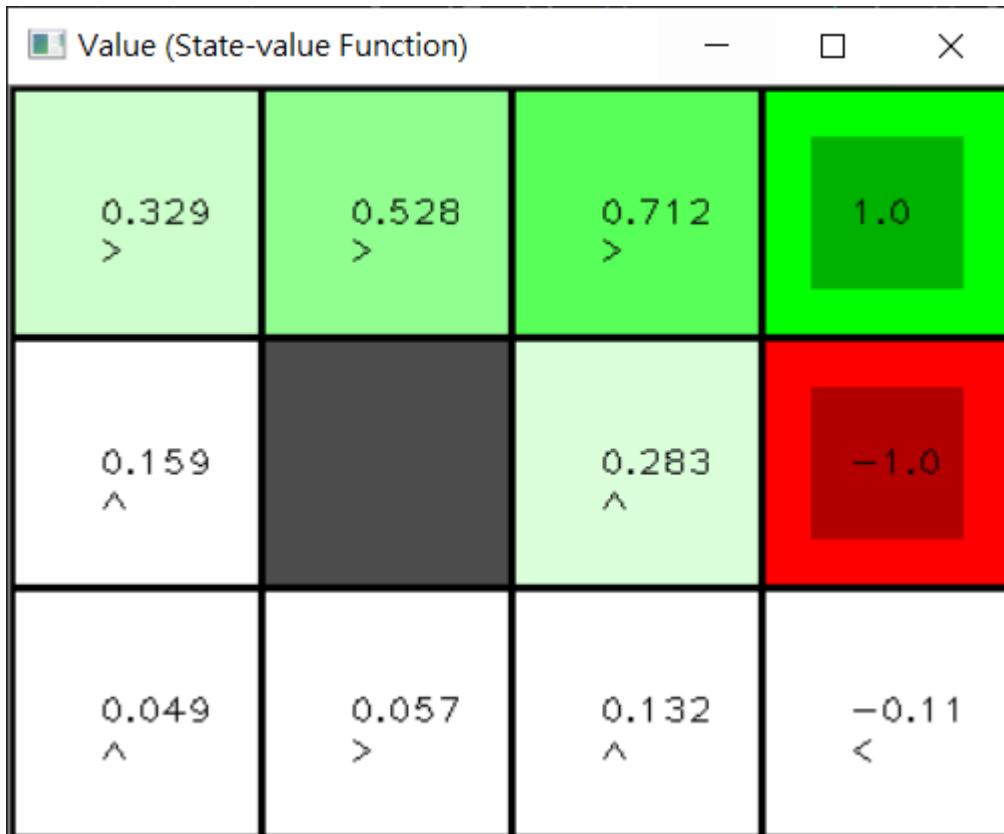
Policy Iteration Demo - Evaluation 4



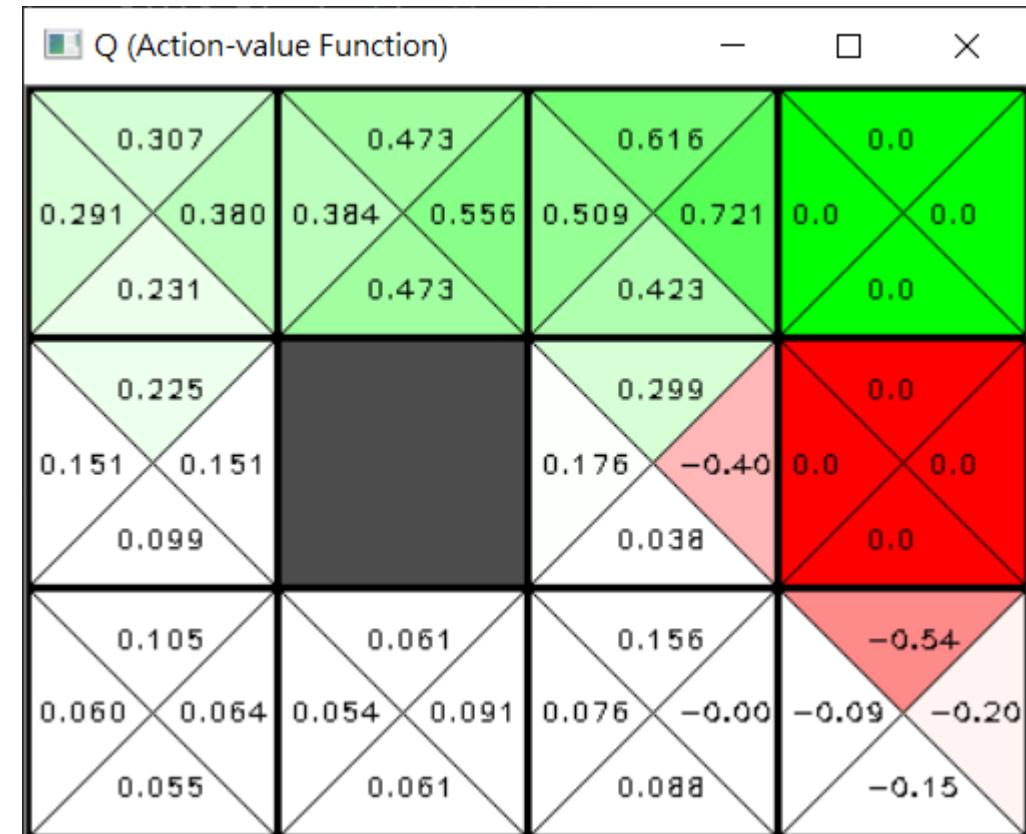
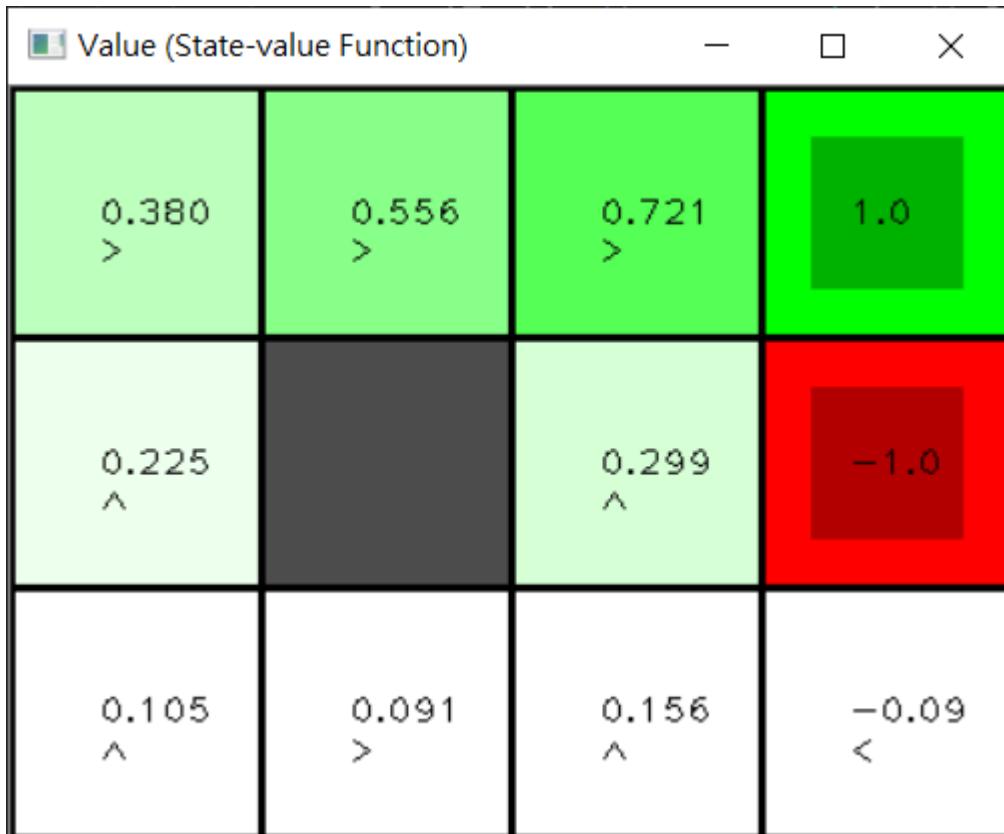
Policy Iteration Demo - Improvement 4



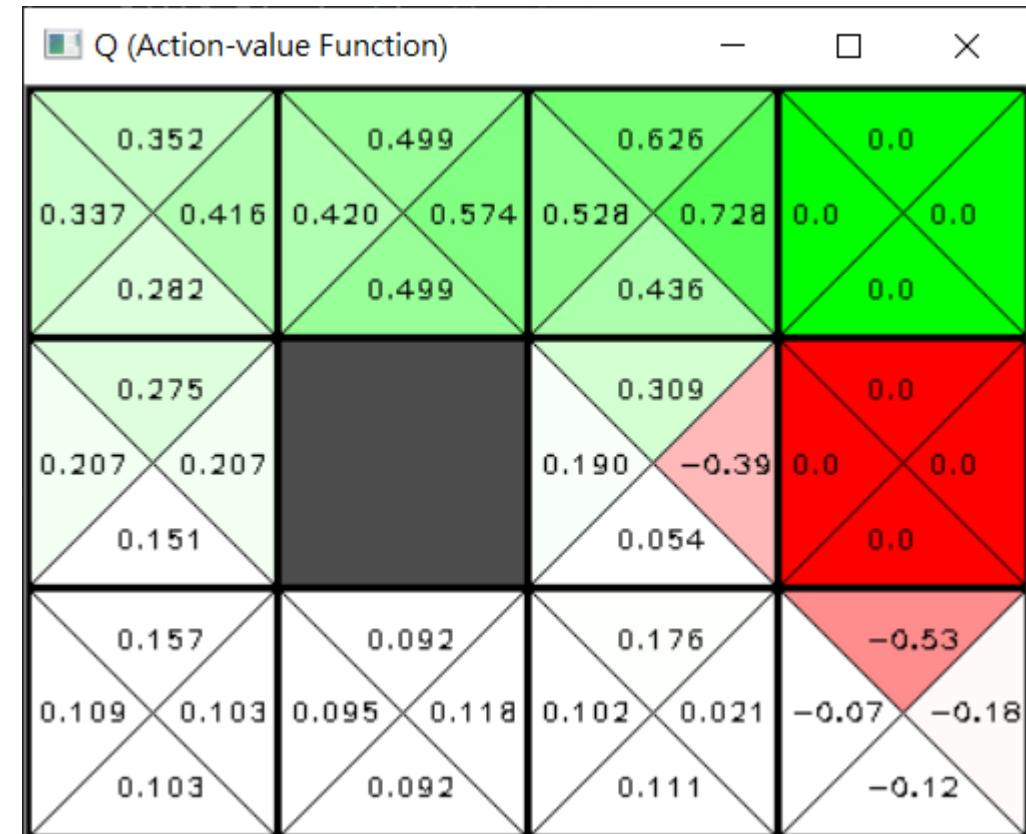
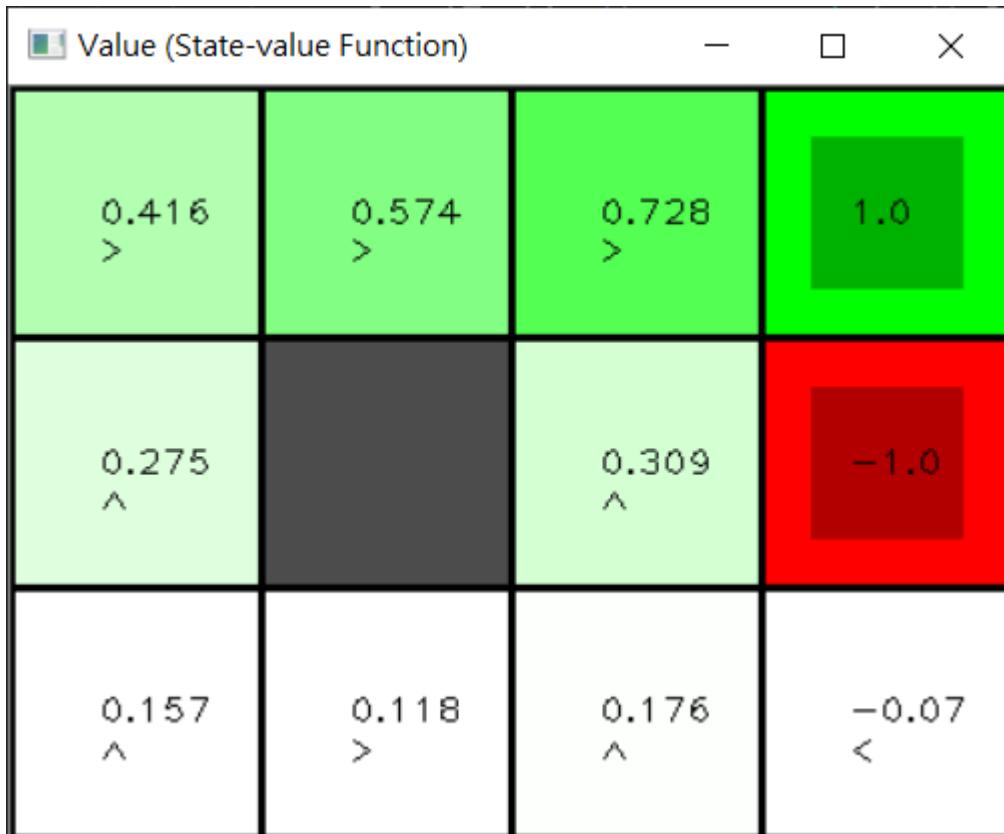
Policy Iteration Demo - Iteration 5



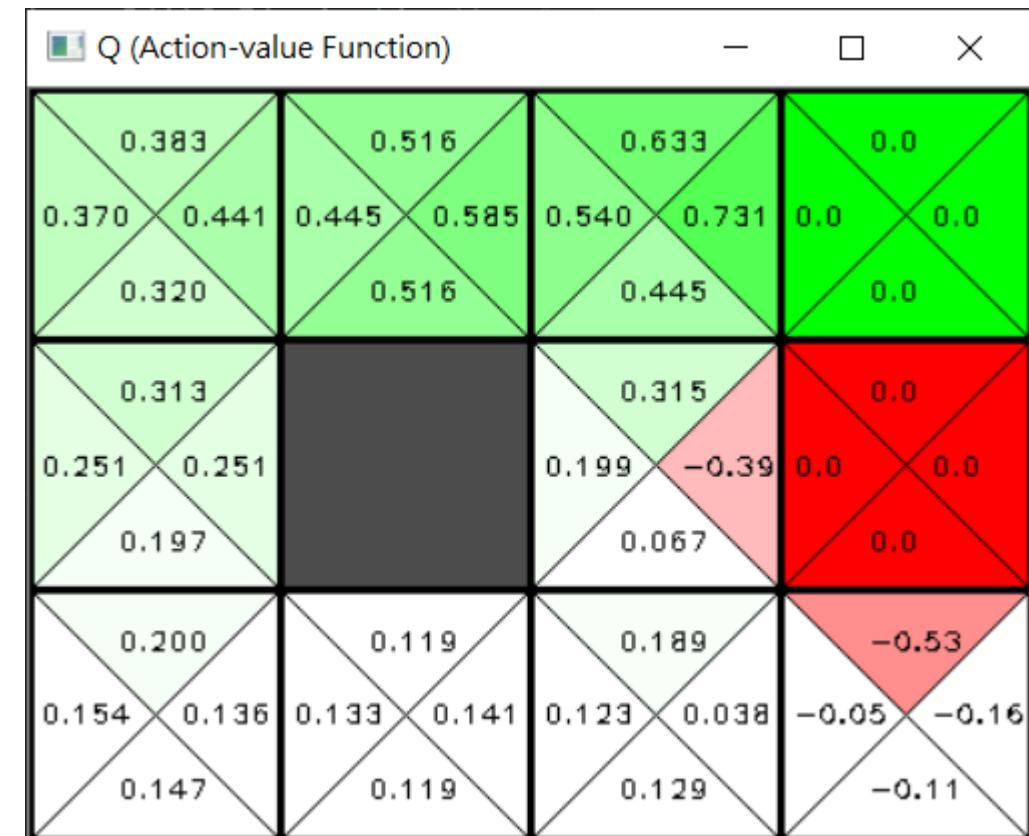
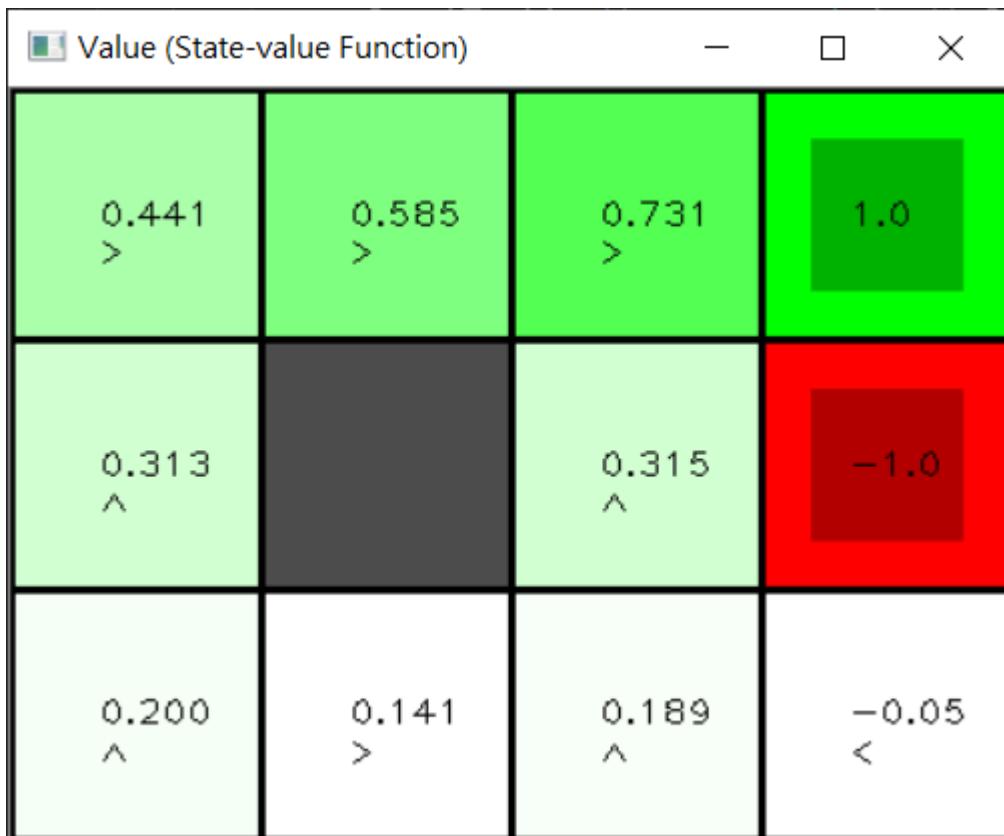
Policy Iteration Demo - Iteration 6



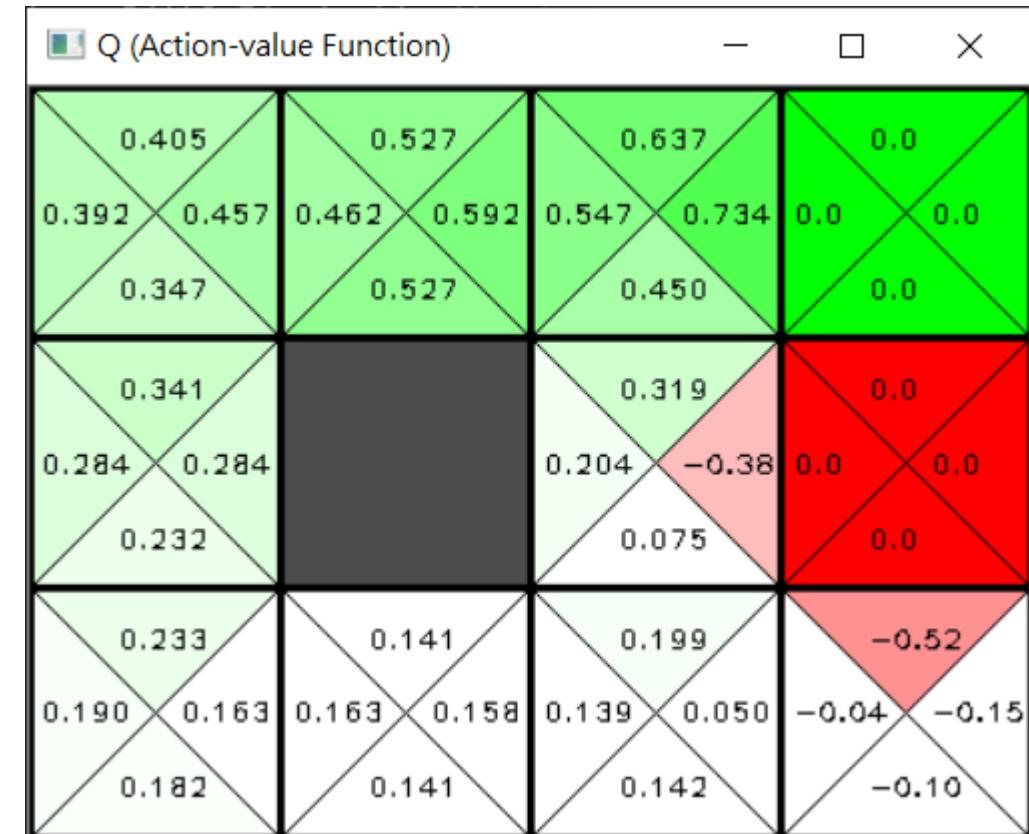
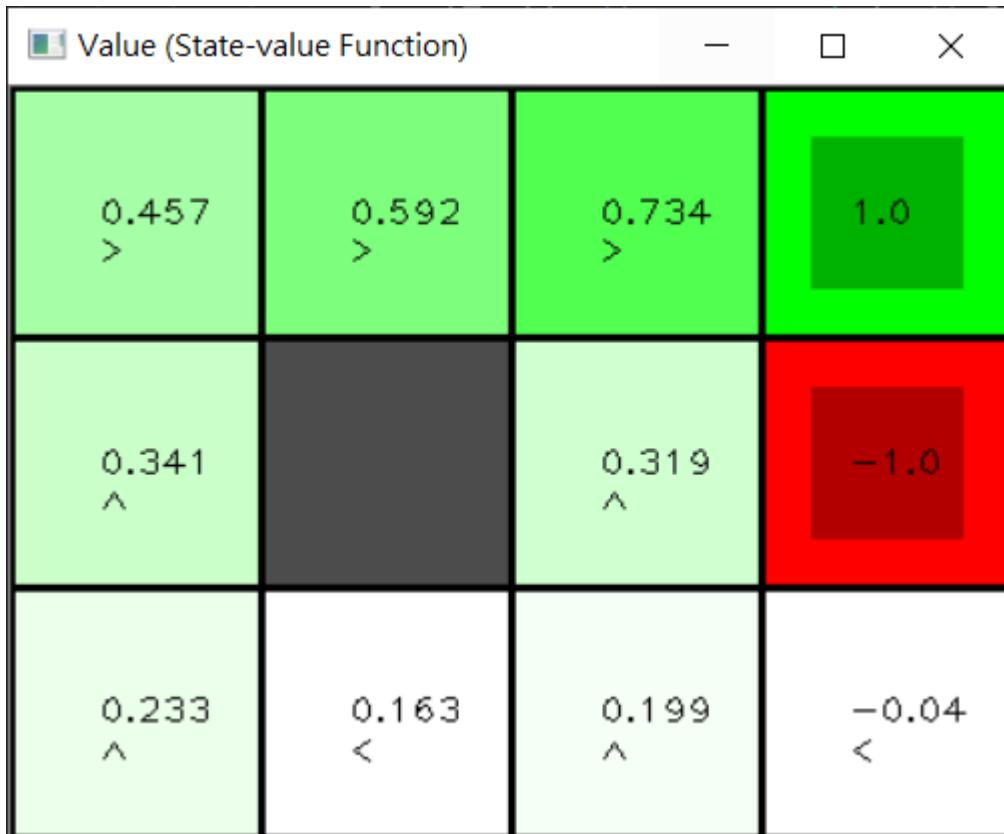
Policy Iteration Demo - Iteration 7



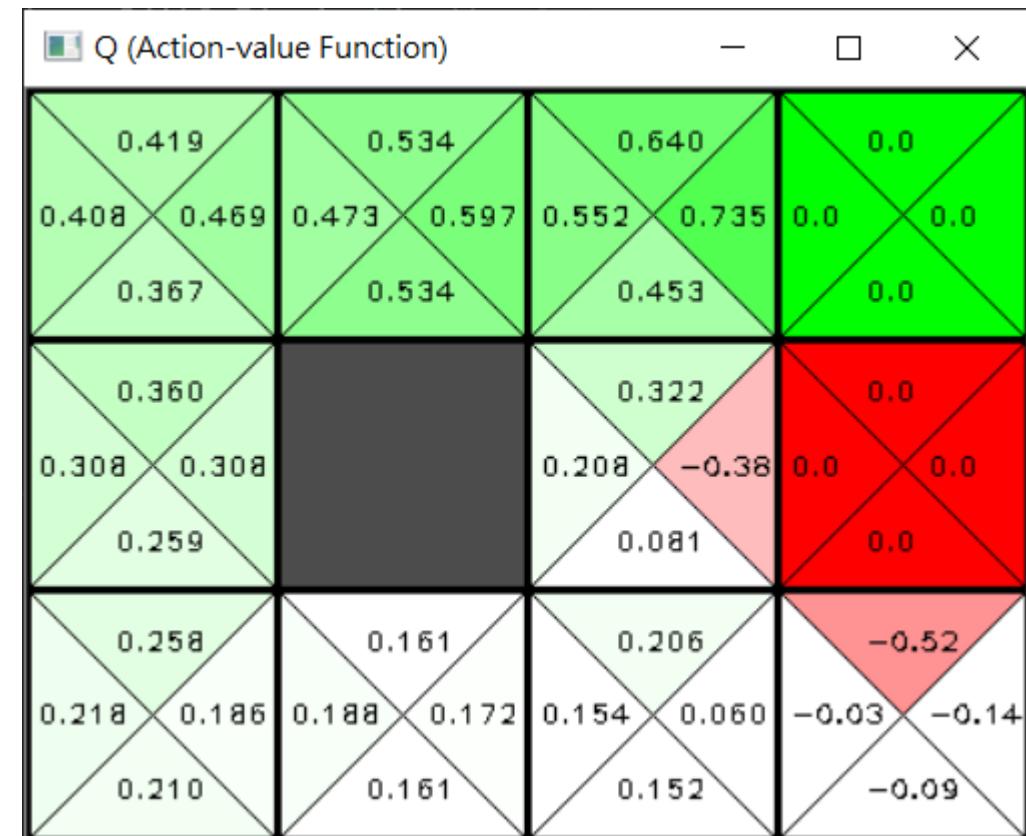
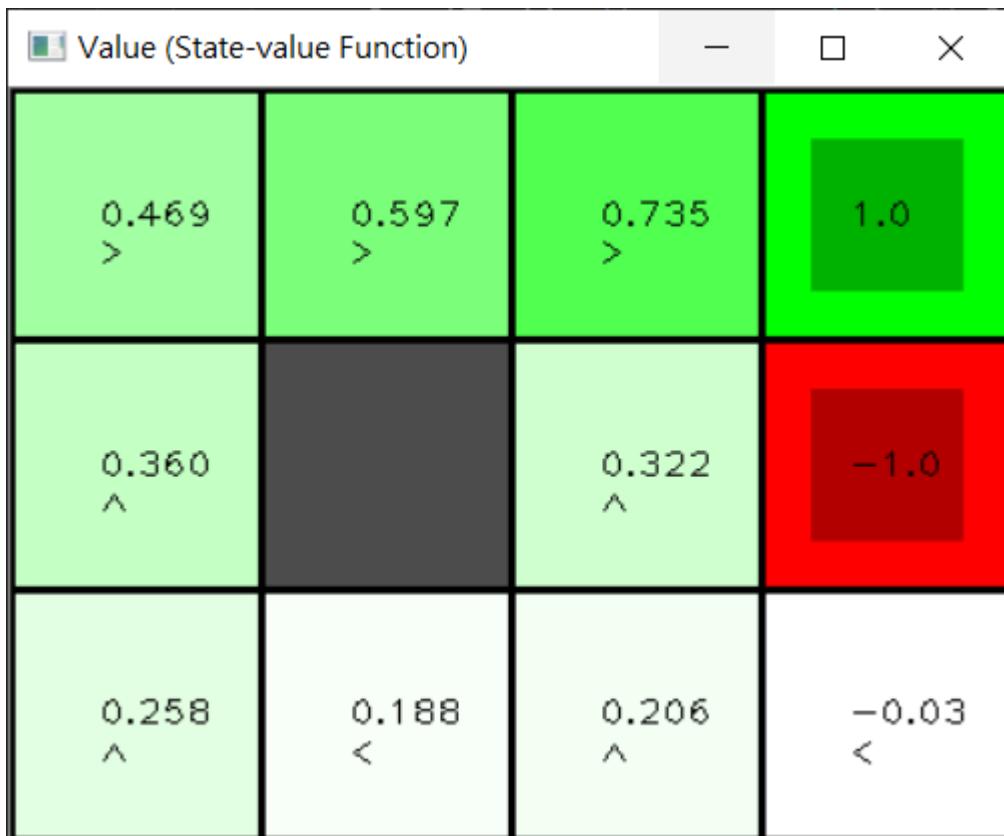
Policy Iteration Demo - Iteration 8



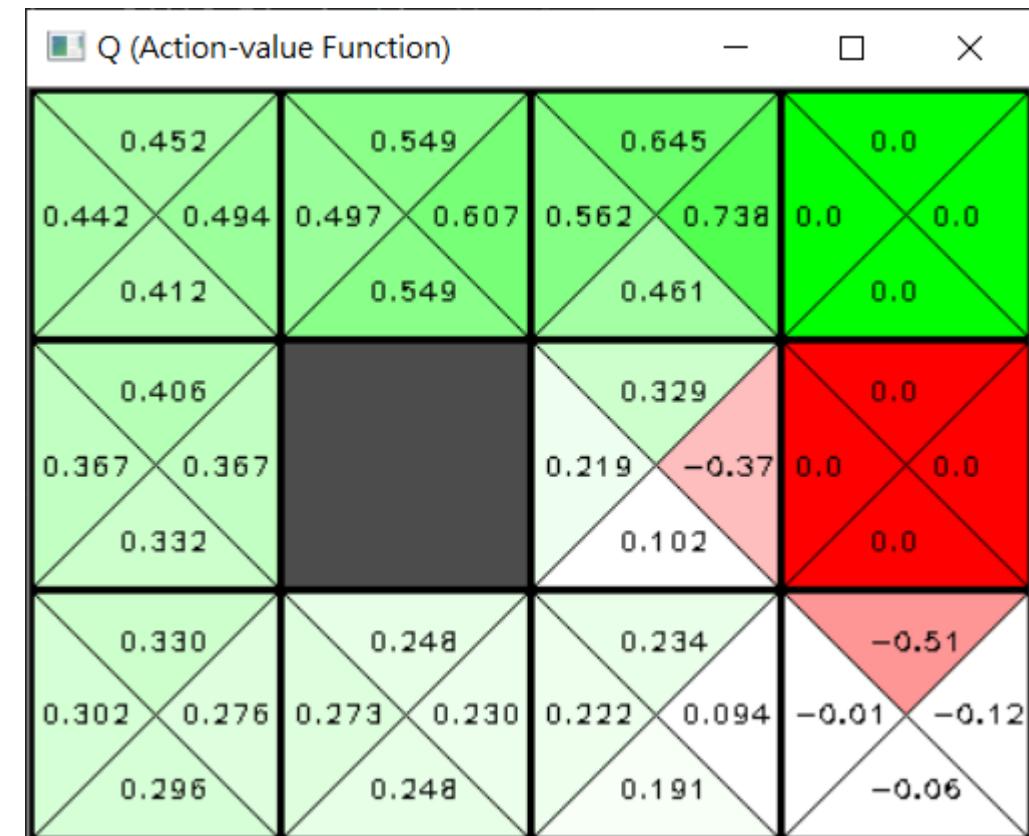
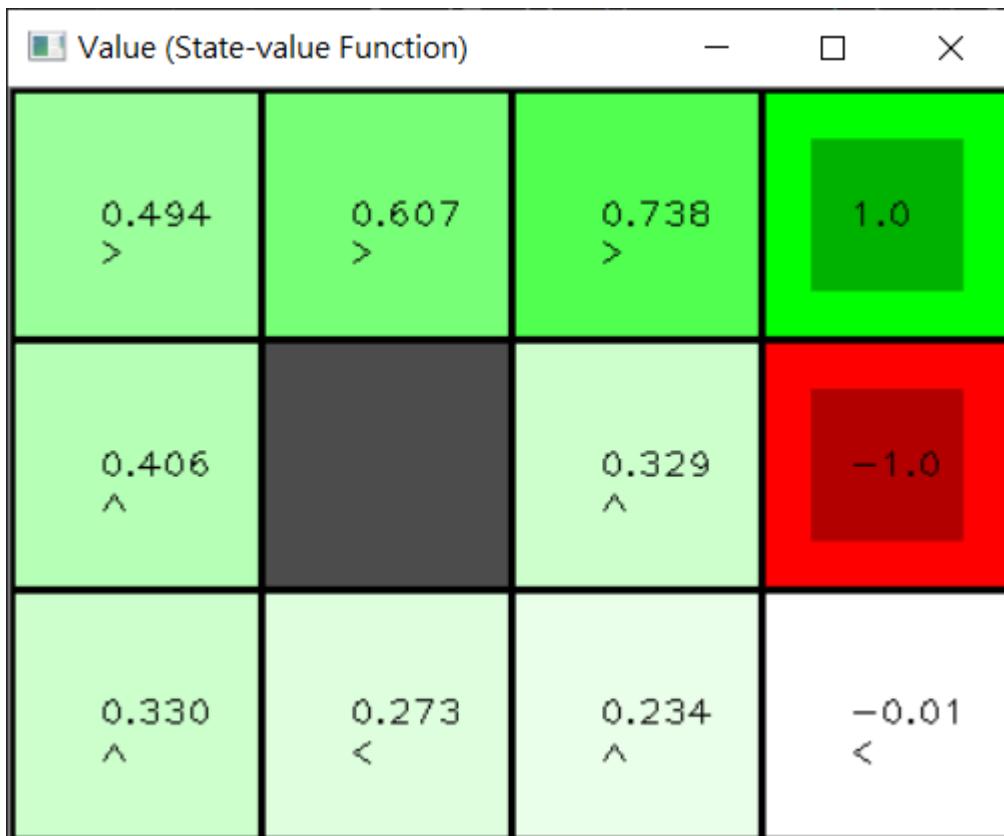
Policy Iteration Demo - Iteration 9



Policy Iteration Demo - Iteration 10



Policy Iteration Demo - Iteration 30



Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- **Value-based Reinforcement Learning**
 - From MDP to RL
 - Temporal Difference Learning (SARSA / Q-Learning)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- **Policy-based Reinforcement Learning**
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

From MDP to RL

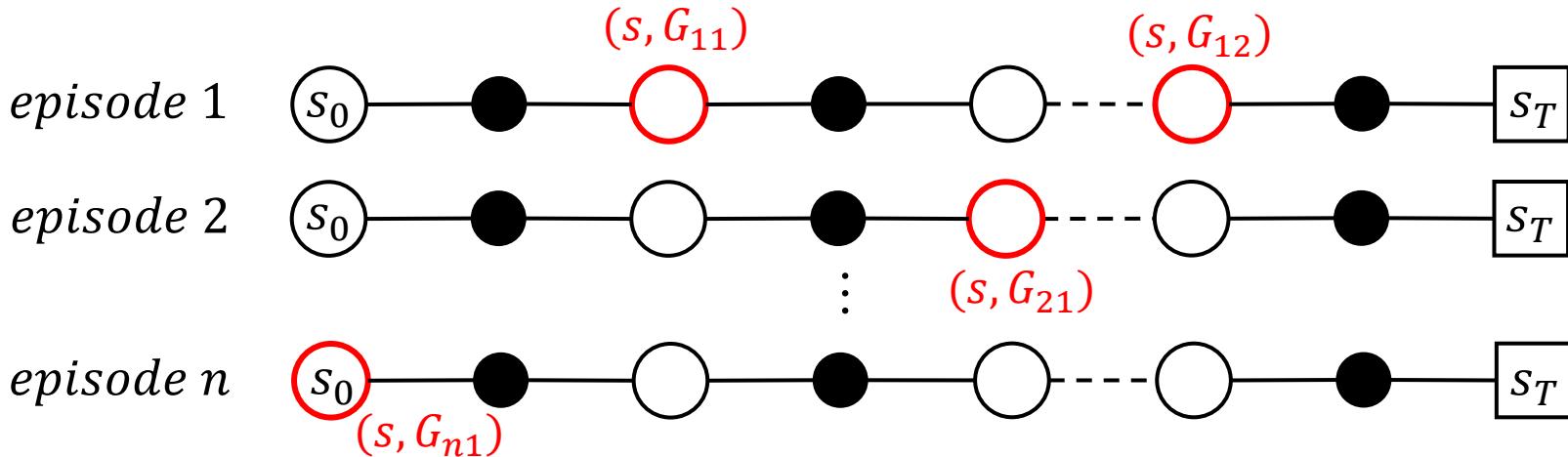
- In MDP problem, we can utilize dynamic programming and bellman backup operator to get the converge value function.
- However, the **environment dynamic $P(s'|s, a)$** (transition probability) is unknown in practical, thus we cannot calculate the expectation term.

$$V^\pi(s) = \sum_a \pi(a|s)[r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s')]$$

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_a \pi(a'|s') Q^\pi(s', a')$$

- In reinforcement learning, we utilize **sampling method** to estimate the value function $V(s), Q(s, a)$.
 - Monte-Carlo Method
 - Temporal-Difference Method

Monte-Carlo Method (MC)



$$G_t = r_t + \gamma r_{t+1} + \cdots + \gamma^T r_T \quad (\text{Return})$$

$$V^\pi(s_t) = \mathbb{E}_\pi[G_t]$$

$$N(s_t) \leftarrow N(s_t) + 1 \quad (\text{visit times})$$

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)} (G_t - V(s_t))$$

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1 \quad (\text{visit times})$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)} (G_t - Q(s_t, a_t))$$

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

Temporal-Difference Method (TD)

- Learns from incomplete episodes, by bootstrapping.

Monte-Carlo Estimation

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(G_t - Q(s_t, a_t))$$

Temporal-Difference

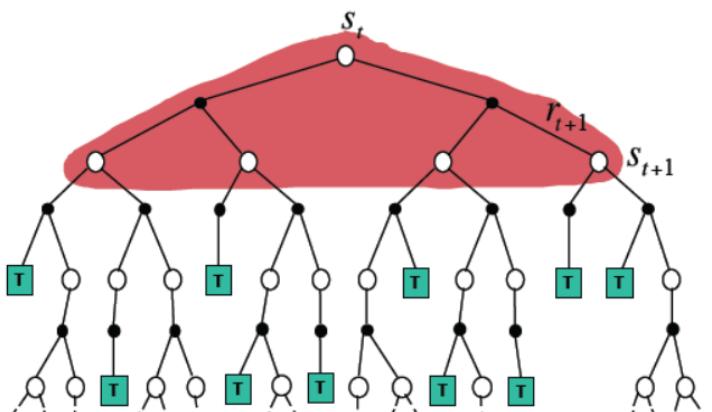
$$V(s_t) \leftarrow V(s_t) + \alpha \frac{r_t + \gamma V(s_{t+1}) - V(s_t)}{\text{TD-Error}}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \frac{r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)}{\text{TD-Error}}$$

Comparison of DP/MC/TD

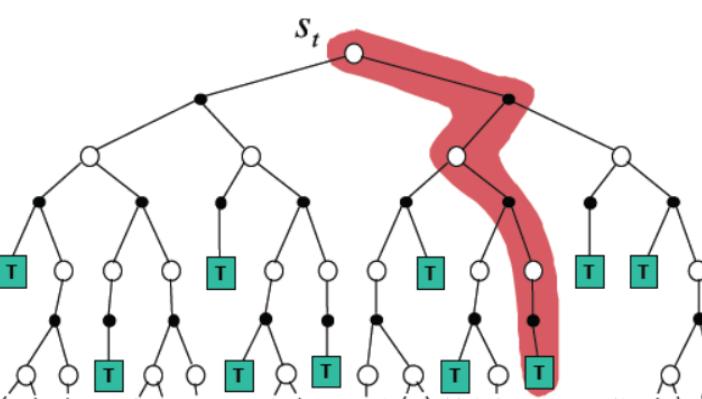
Dynamic Programming

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



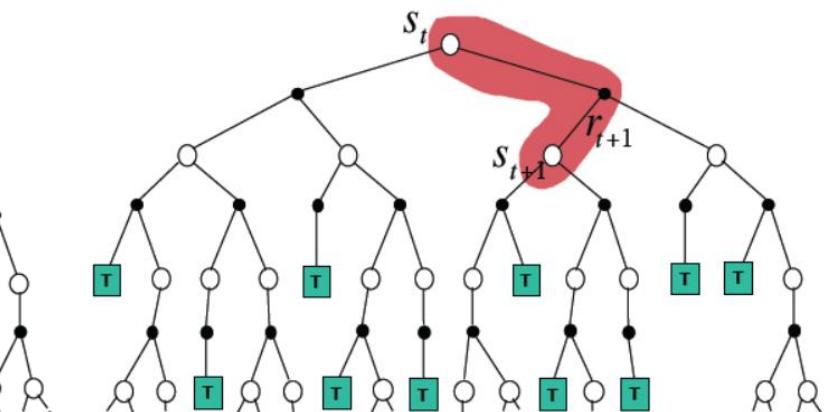
Monte-Carlo

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



Temporal-Difference

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



Exploration and Exploitation

- By using Monte-Carlo or temporal-difference method, we can estimate the value functions. However, different from the common estimation problem, the policy will affect the data we sample.
- The estimation of the value function is **inaccurate** at first. If we always use the estimated value function to select the action, we might trap in the local maxima and lose the chance to find the better policy.
- We need to not only select different actions to **explore** the environment for finding potential policy but also **exploit** the policy from learned experience.

How do we decide when to explore and when to exploit ?

Multi-armed Bandit Problem

- The multi-armed bandit problem is a classic problem that well demonstrates the **exploration-exploitation dilemma**.
- Here we have a one-armed bandit machine. Once we pull the arms will receive a reward. We can estimate the expected reward after several tries.



One-armed Bandit

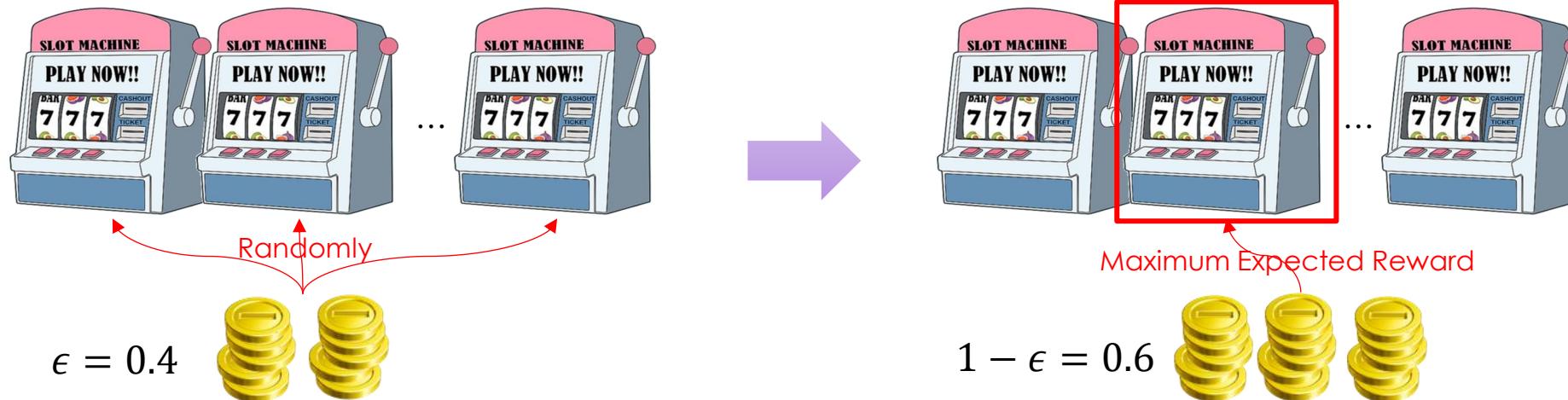
Multi-armed Bandit Problem

- The multi-armed bandit problem is a classic problem that well demonstrates the **exploration-exploitation dilemma**.
- Here we have a one-armed bandit machine. Once we pull the arms will receive a reward. We can estimate the expected reward after several tries.
- Then, suppose we have multiple one-armed bandit machines, each time we can choose one to pull the arms. **Now the problem is, how can we get the most total reward after limit attempts ?**



ϵ – first

- Exploration:
 - We do not know the expectation of each bandit at first, so we randomly select the actions at the first ϵ percent attempts.
- Exploitation:
 - After several attempts, we can calculate the expected reward. In the remaining $(1-\epsilon)$ percent attempts, we can always select the machine with maximum expected reward.



ϵ – greedy

- Are the first ϵ percent trials enough for estimating the expectation correctly? Can we use each trial more efficiently to earn rewards ?
- ϵ – greedy allow us to **simultaneously estimate and earn reward**.
- In each round of selection, there will be a probability of ϵ to randomly select a slot machine (**exploration**), and a probability of $(1-\epsilon)$ will select the machine with the largest average return in the past (**exploitation**).



Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- Value-based Reinforcement Learning
 - From MDP to RL
 - Temporal Difference Learning (Q-Learning / SARSA)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- Policy-based Reinforcement Learning
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

On-Policy vs. Off-Policy

- On-Policy:
 - The agent learned and the agent interacting with environment is the same.
- Off-Policy:
 - The agent learned and the agent interacting with the environment is different.



On-Policy: 阿光下棋



By Hung-yi Lee

Off-Policy: 佐為下棋，阿光在旁邊看

Q-Learning

Bellman Optimality Equation:

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

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

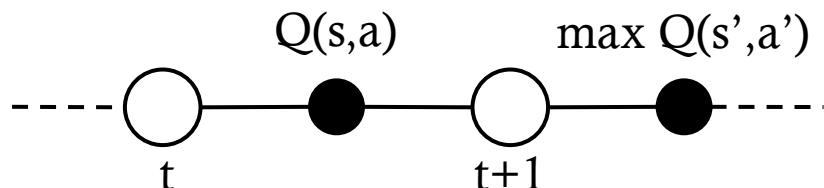
 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal

Off-Policy Method



Q-Learning Demo



SARSA

- State-Action-Reward-State-Action (SARSA)

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

Policy Improvement

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

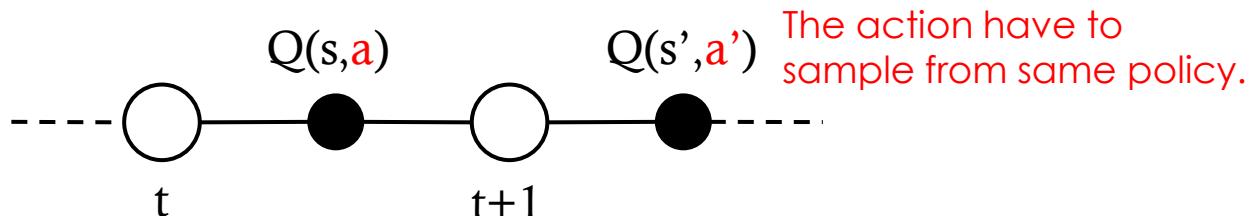
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$;

Policy Evaluation

 until S is terminal

On-Policy Method



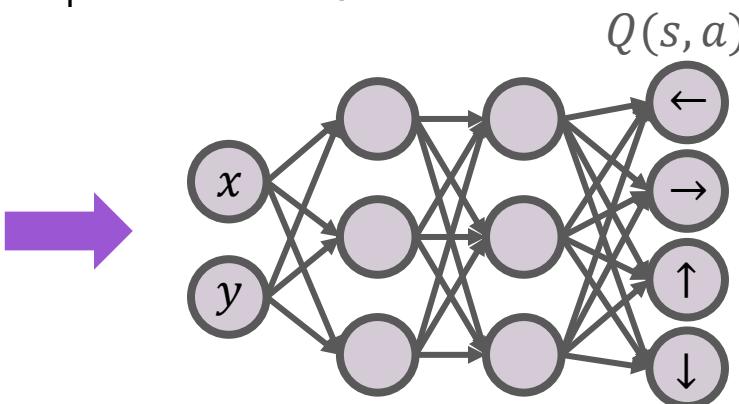
Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- Value-based Reinforcement Learning
 - From MDP to RL
 - Temporal Difference Learning (SARSA / Q-Learning)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- Policy-based Reinforcement Learning
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

Deep Q-Network

- Large State Dimension Problem
 - Utilize function approximator to replace the Q table

$Q(s, a)$	\leftarrow	\rightarrow	\uparrow	\downarrow
(0,0)				
(0,1)				
(1,0)				
(1,1)				



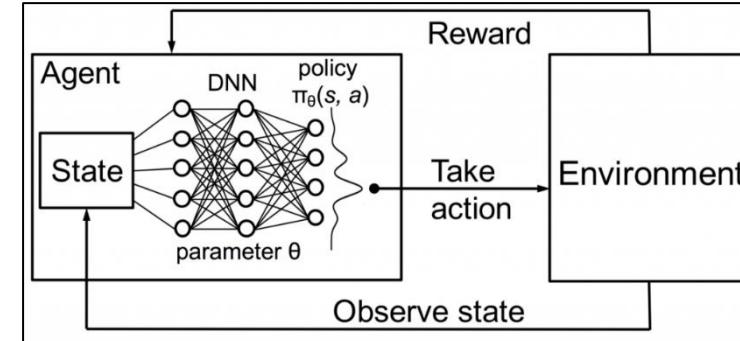
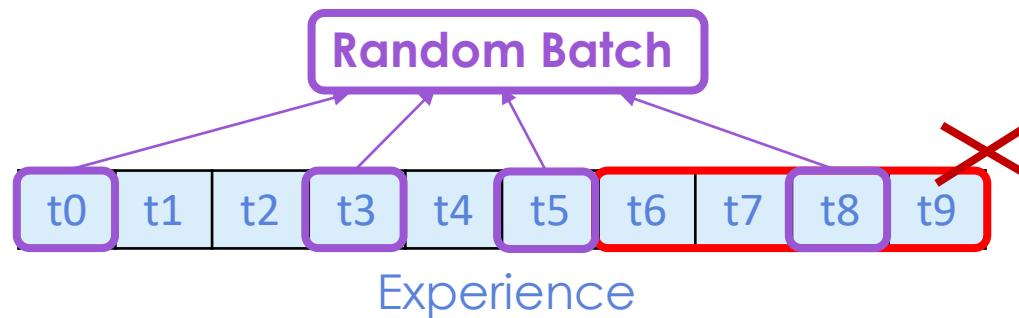
- Minimize the objective of td-error

Mean-Square Error TD-Learning :

$$L(\theta) = \mathbb{E}_{\{s, a, r, s'\}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta) \right)^2 \right]$$

Deep Q-Network

- Some design details:
 - **Two Q Networks (Evaluation/Target)** :
Stable the optimization process.
 - **Clip Reward** :
Stable the optimization process.
 - **Experience Replay**:
Break the time relation of sample.



Algorithm 1: deep Q-learning with experience replay.

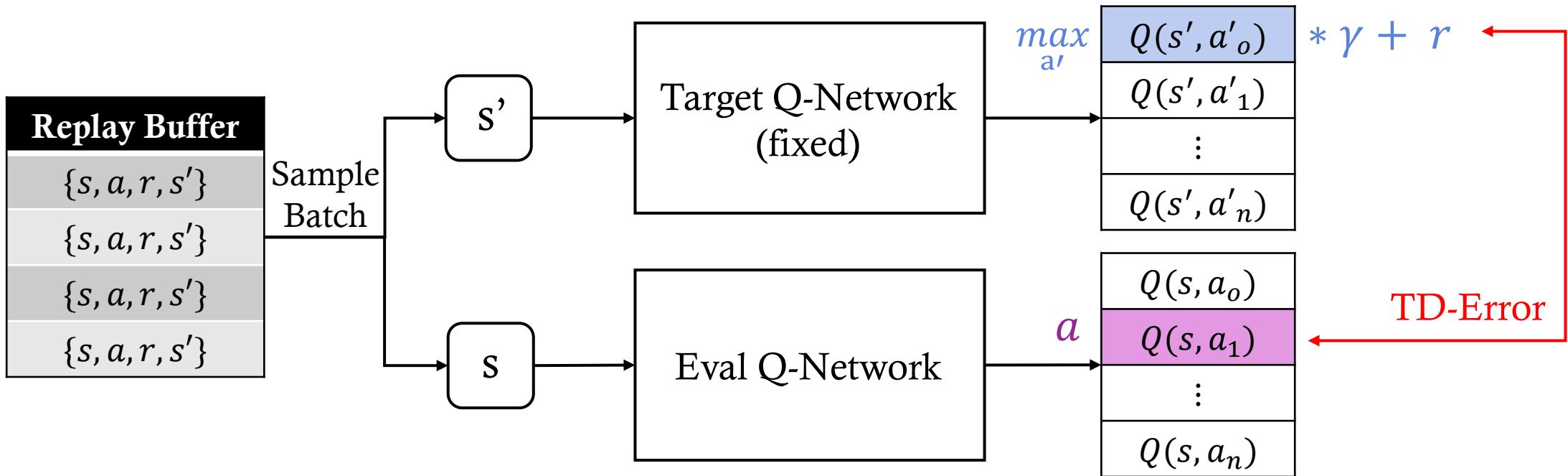
```

Initialize replay memory  $D$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights  $\theta$ 
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  $\epsilon$  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  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$ 
        Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$ 
        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the
        network parameters  $\theta$ 
        Every  $C$  steps reset  $\hat{Q} = Q$ 
    End For
End For

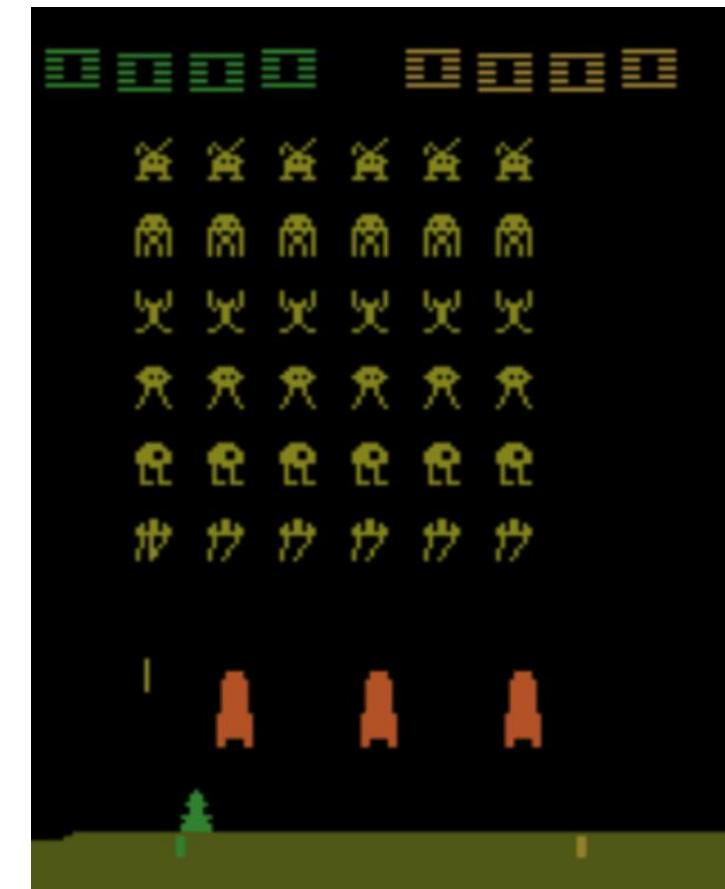
```

Deep Q-Network

$$L(\theta) = \mathbb{E}_{\{s,a,r,s'\}} \left[\left(r + \gamma \max_{a'} Q_{target}(s', a'; \theta^{old}) - Q_{eval}(s, a; \theta) \right)^2 \right]$$



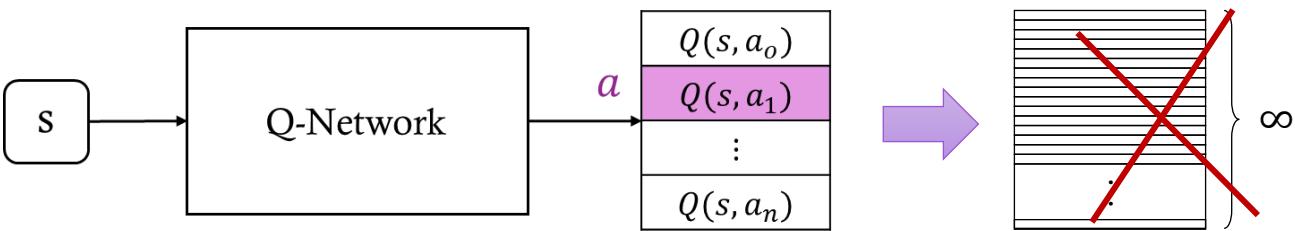
Deep Q-Network



$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

Continuous Deep Q-Learning

- What if the action space is continuous ?



- The network output $V(s)$, $\mu(s)$, $L(s)$, define **Normalized Advantage Function** $A(s, a)$.

$$Q(s, a) = A(s, a|\theta^A) + V(s|\theta^V)$$

$$A(s, a|\theta^A) = -\frac{1}{2}(a - \mu(s|\theta^\mu))^T M(s|\theta^M)(a - \mu(s|\theta^\mu))$$

Make sure the maximum Q given the action $a = \mu(s|\theta^\mu)$

Single Variable Situation:

$$Q(s, a) = -\frac{1}{2}m(a - \mu)^2 + V(s)$$

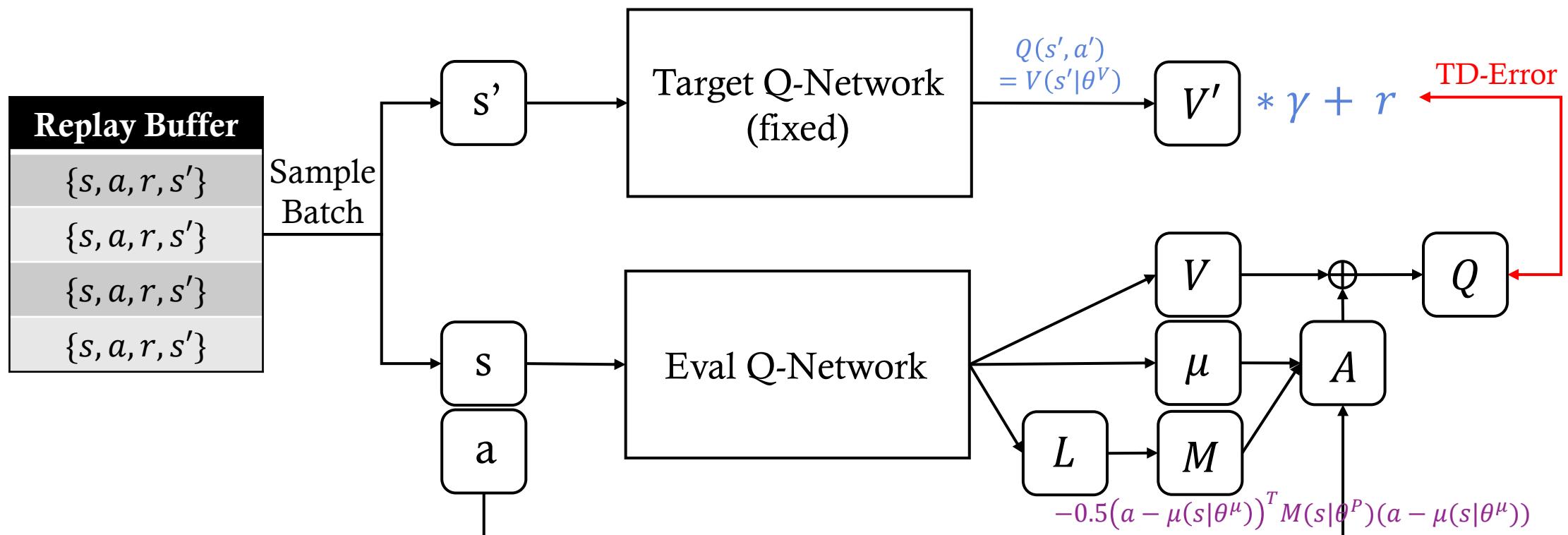
$$M(s|\theta^M) = L(s|\theta^M)L(s|\theta^M)^T$$

Symmetric Matrix Upper-Triangle Matrix

- We can select the action $a = \mu(s|\theta^u)$ as the maximal Q control.

Normalized Advantage Function (NAF)

$$L(\theta) = \mathbb{E}_{\{s,a,r,s'\}} \left[\left(r + \gamma \max_{a'} Q_{target}(s', a'; \theta^{old}) - Q_{eval}(s, a; \theta) \right)^2 \right]$$



Normalized Advantage Function (NAF)



Algorithm 1 Continuous Q-Learning with NAF

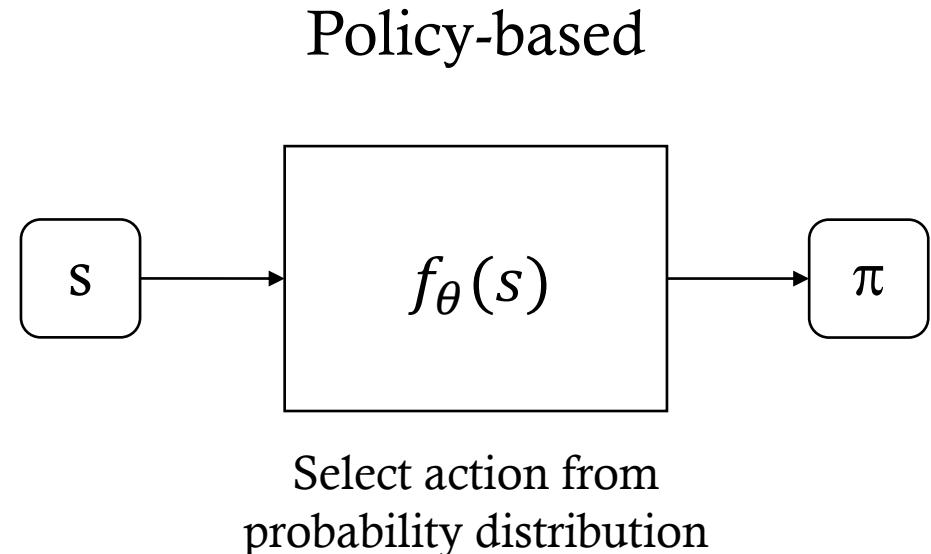
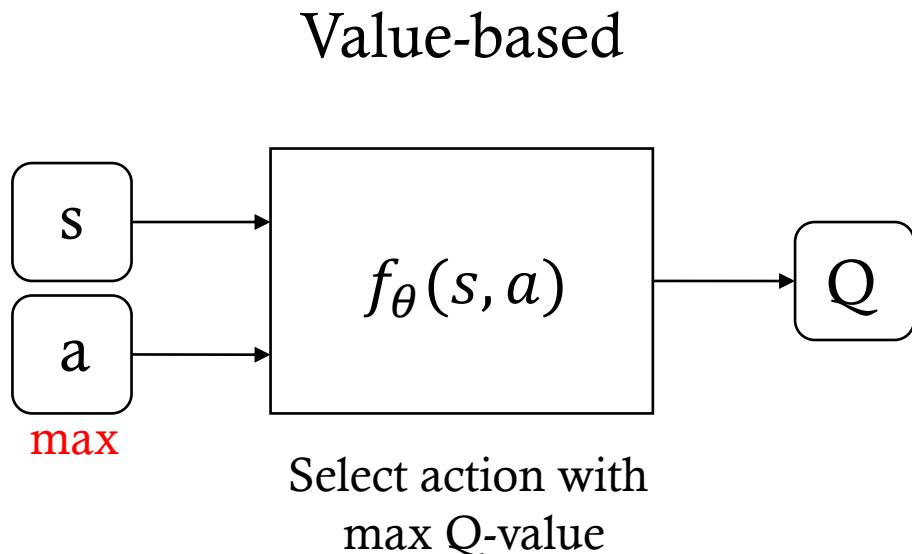
```
Randomly initialize normalized Q network  $Q(\mathbf{x}, \mathbf{u}|\theta^Q)$ .  
Initialize target network  $Q'$  with weight  $\theta^{Q'} \leftarrow \theta^Q$ .  
Initialize replay buffer  $R \leftarrow \emptyset$ .  
for episode=1,  $M$  do  
    Initialize a random process  $\mathcal{N}$  for action exploration  
    Receive initial observation state  $\mathbf{x}_1 \sim p(\mathbf{x}_1)$   
    for t=1,  $T$  do  
        Select action  $\mathbf{u}_t = \mu(\mathbf{x}_t|\theta^\mu) + \mathcal{N}_t$   
        Execute  $\mathbf{u}_t$  and observe  $r_t$  and  $\mathbf{x}_{t+1}$   
        Store transition  $(\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1})$  in  $R$   
        for iteration=1,  $I$  do  
            Sample a random minibatch of  $m$  transitions from  $R$   
            Set  $y_i = r_i + \gamma V'(\mathbf{x}_{i+1}|\theta^{Q'})$   
            Update  $\theta^Q$  by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(\mathbf{x}_i, \mathbf{u}_i|\theta^Q))^2$   
            Update the target network:  $\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$   
        end for  
    end for  
end for
```

Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
 - MDP Formulation
 - Value Function and Bellman Equation
 - Dynamic Programming Methods
- Value-based Reinforcement Learning
 - From MDP to RL
 - Temporal Difference Learning (SARSA / Q-Learning)
 - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)
- Policy-based Reinforcement Learning
 - Policy Gradient and Actor Critic
 - Sampling Efficiency Problem (TRPO & PPO)
 - Deep Deterministic Policy Gradient (DDPG) & Soft Actor-Critic (SAC)

Policy-based vs. Value-based Method

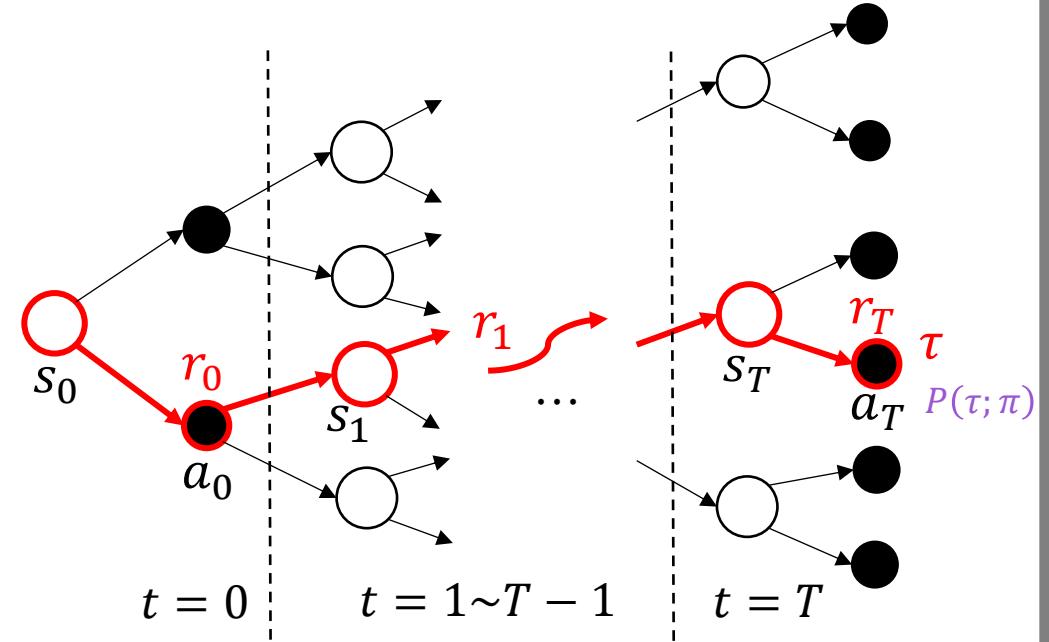
- Value-based method aims to learn the action-value function and policy-based method aims to learn the policy function.



Policy Gradient

- For policy-based method, we construct a function approximator to learn the mapping from state to the action.
- If the computation of gradient for the parameters is achievable, we can utilize iterative optimization method to optimize the policy.
- Definition
 - Trajectory $\tau = \{s_0, a_0, s_1, a_1, \dots, s_T, a_T\}$
 - Return $G(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots + \gamma^T r_t$
 - Probability of a trajectory

$$P(\tau; \pi) = P(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$



Policy Gradient

- Gradient of the policy

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} E \left[\sum_{t=0}^T \gamma^t r_t \right] \\&= \nabla_{\theta} \sum_{\tau} P(\tau; \pi_{\theta}(a|s)) * G(\tau) \\&= \sum_{\tau} \nabla_{\theta} P(\tau; \pi_{\theta}) * G(\tau) \\&= \sum_{\tau} P(\tau; \pi_{\theta}) \frac{\nabla_{\theta} P(\tau; \pi_{\theta})}{P(\tau; \pi_{\theta})} G(\tau) \\&= \sum_{\tau} P(\tau; \pi_{\theta}) \nabla_{\theta} \log P(\tau; \pi_{\theta}) * G(\tau) \\&= \mathbb{E}_{\tau} [\nabla_{\theta} \log P(\tau; \pi_{\theta}) * G(\tau)]\end{aligned}$$

- Probability of a trajectory

$$\begin{aligned}\nabla_{\theta} \log P(\tau; \pi_{\theta}) \\&= \nabla_{\theta} \log P(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \pi_{\theta}(a_t|s_t) \\&= \nabla_{\theta} \sum_{t=0}^{T-1} \log P(s_{t+1}|s_t, a_t) + \nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t) \\&= \nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t)\end{aligned}$$

Policy Gradient:

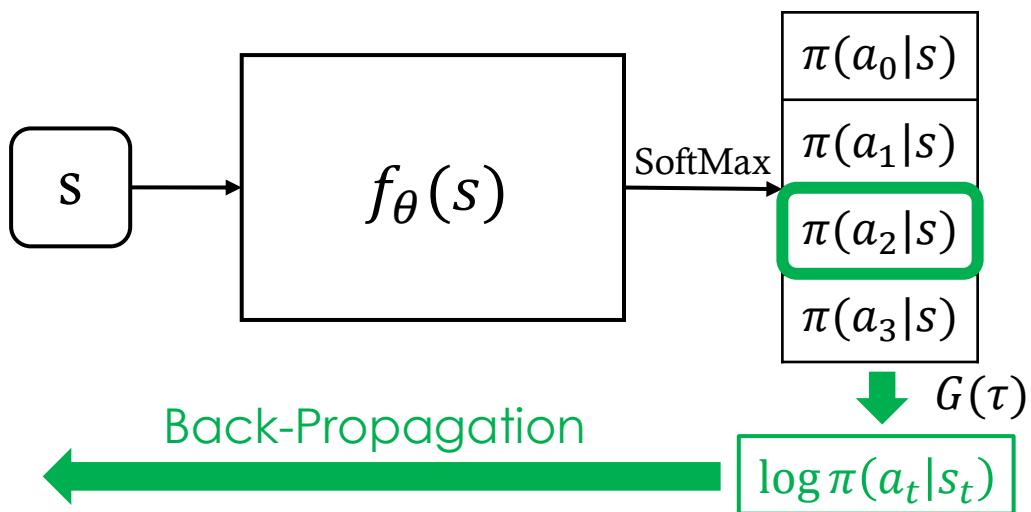
$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau} \left[\nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t) * G(\tau) \right]$$

Policy Gradient

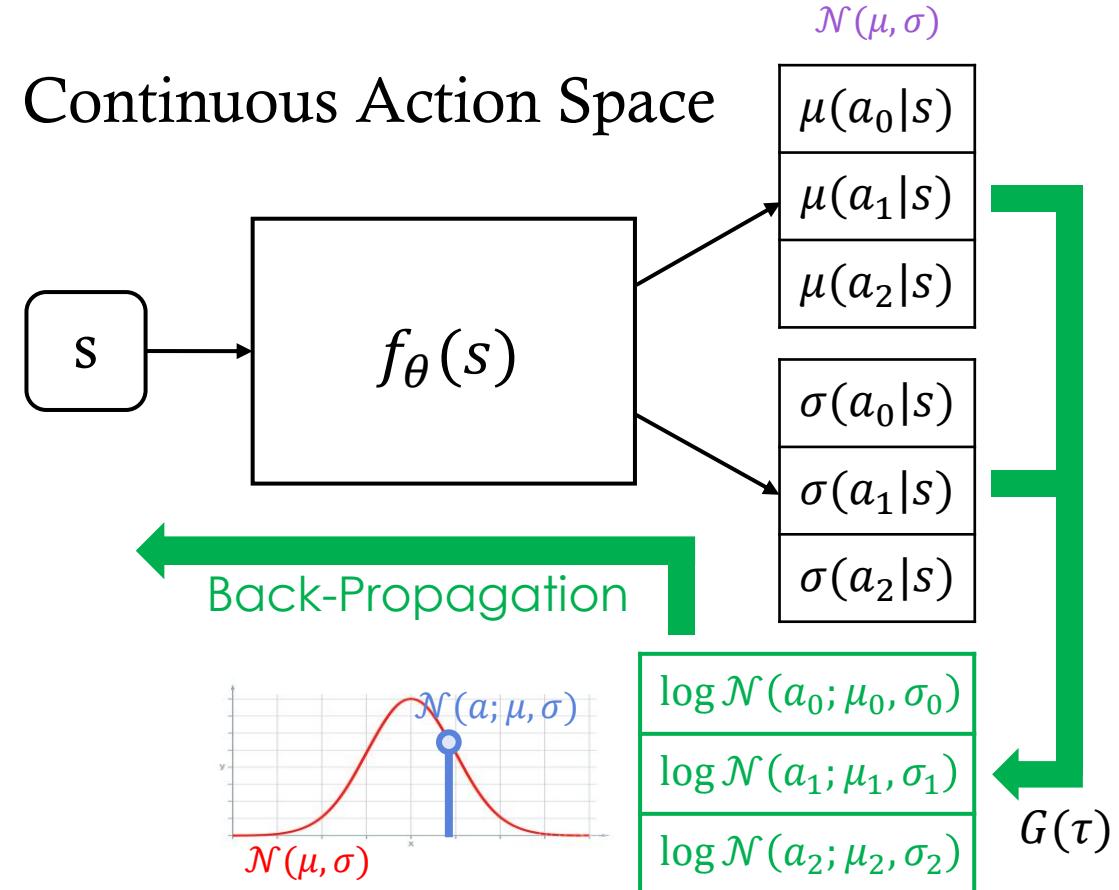
Policy Gradient:

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

Discrete Action Space

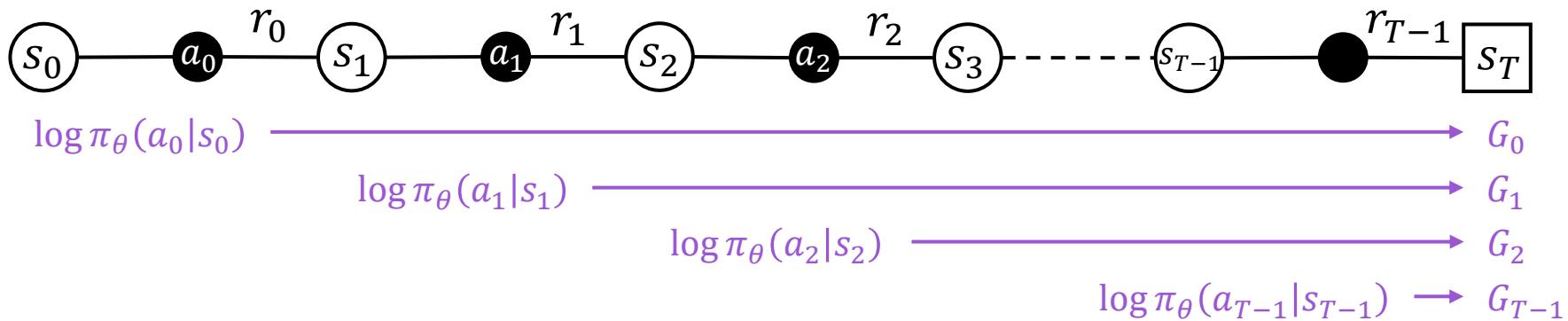
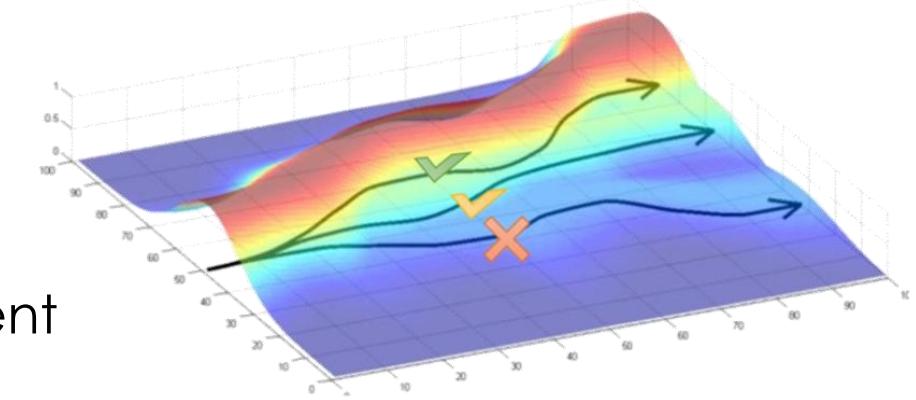


Continuous Action Space



REINFORCE Algorithm

- Monte-Carlo estimation of the policy gradient



$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau} \left[\nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t | s_t) * G_t(\tau) \right] \approx \frac{1}{m} \sum_{i=1}^m \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) * G_t(\tau^i)$$

(Episode Update)

Baseline for Policy Gradient

- Adding an appropriate baseline function can reduce the variance of the estimation for policy gradient.

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\tau}[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) * G_t(\tau)] = \mathbb{E}_{\tau}[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) * (G_t(\tau) - b(s_t))] \\ &\Rightarrow \mathbb{E}_{\tau}[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) * b(s_t)] = 0\end{aligned}$$

Proof:

$$\begin{aligned}&\mathbb{E}_{\tau}[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) * b(s_t)] \\ &= \mathbb{E}_{s_{0:t}, a_{0:t}} [\mathbb{E}_{s_{t+1:T}, a_{t+1:T}} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) * b(s_t)]] \\ &= \mathbb{E}_{s_{0:t}, a_{0:t}} [b(s_t) * \mathbb{E}_{s_{t+1:T}, a_{t+1:T}} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)]] \\ &= \mathbb{E}_{s_{0:t}, a_{0:t}} \left[b(s_t) * \sum_{a_t \in A} \sum_{s_{t+1} \in S} \dots \sum_{s_T \in S} \pi_{\theta}(a_t, s_t) P(s_{t+1} | s_t, a_t) \dots P(s_T | s_{T-1}, a_{T-1}) (\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)) \right] \\ &= \mathbb{E}_{s_{0:t}, a_{0:t}} \left[b(s_t) * \sum_{a_t \in A} \pi_{\theta}(a_t | s_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \sum_{s_{t+1} \in S} P(s_{t+1} | s_t, a_t) \sum_{a_{t+1} \in A} \dots \sum_{s_T \in S} P(s_T | s_{T-1}, a_{T-1}) \right] \\ &= \mathbb{E}_{s_{0:t}, a_{0:t}} [b(s_t) * \mathbb{E}_{a_t} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)]] = \mathbb{E}_{s_{0:t}, a_{0:t}} [b(s_t) * 0] = 0\end{aligned}$$

$$\begin{aligned}
& \mathbb{E}_{a_t} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] \\
&= \sum_{a_t \in A} \pi_{\theta}(a_t | s_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \\
&= \sum_{a_t \in A} \pi_{\theta}(a_t | s_t) \frac{\nabla_{\theta} \pi_{\theta}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} \\
&= \sum_{a_t \in A} \nabla_{\theta} \pi_{\theta}(a_t | s_t) \\
&= \nabla_{\theta} \sum_{a_t \in A} \pi_{\theta}(a_t | s_t) \\
&= \nabla_{\theta} 1 \\
&= 0
\end{aligned}$$

$$\text{Var}(X) = \mathbb{E}[X^2 - 2X\mathbb{E}[X] + (\mathbb{E}[X])^2] = \mathbb{E}[X^2] - 2\mathbb{E}[X]\mathbb{E}[X] + (\mathbb{E}[X])^2 = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

Baseline for Policy Gradient

- Consider the variance of policy gradient

$$\begin{aligned} & \text{Var}\left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) * (G_t(\tau) - b(s_t))\right) \\ & \approx \sum_{t=0}^{T-1} \mathbb{E}_{\tau}\left[\left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) * (G_t(\tau) - b(s_t))\right)^2\right] \\ & \approx \sum_{t=0}^{T-1} \mathbb{E}_{\tau}[(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t))^2] * \mathbb{E}_{\tau}[\left(G_t(\tau) - b(s_t)\right)^2] \end{aligned}$$

Setting $b(s_t) \approx \mathbb{E}_{\tau}[G_t(\tau)]$ to approximate the expected return will have low variance.

Off-Policy with Importance Sampling

- We compute the policy gradient of π_θ by collecting the data under the probability of policy π_θ . Once we update the policy, we need to collect the data again.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [G(\tau) \nabla_\theta \log p_\theta(\tau)]$$

- If we want to reuse the data sampled before updating the policy, we can apply the **importance sampling** technique to estimate the policy gradient.

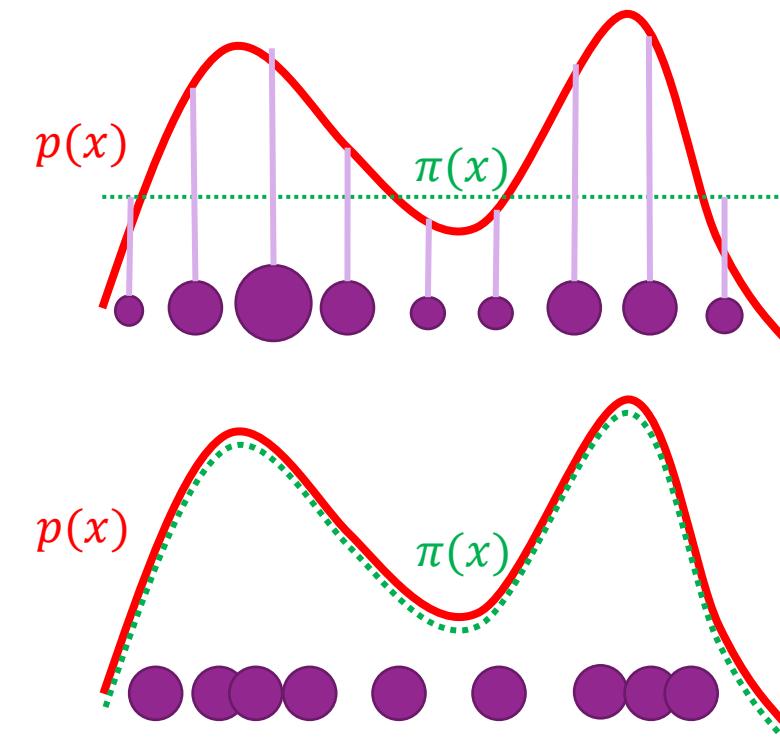
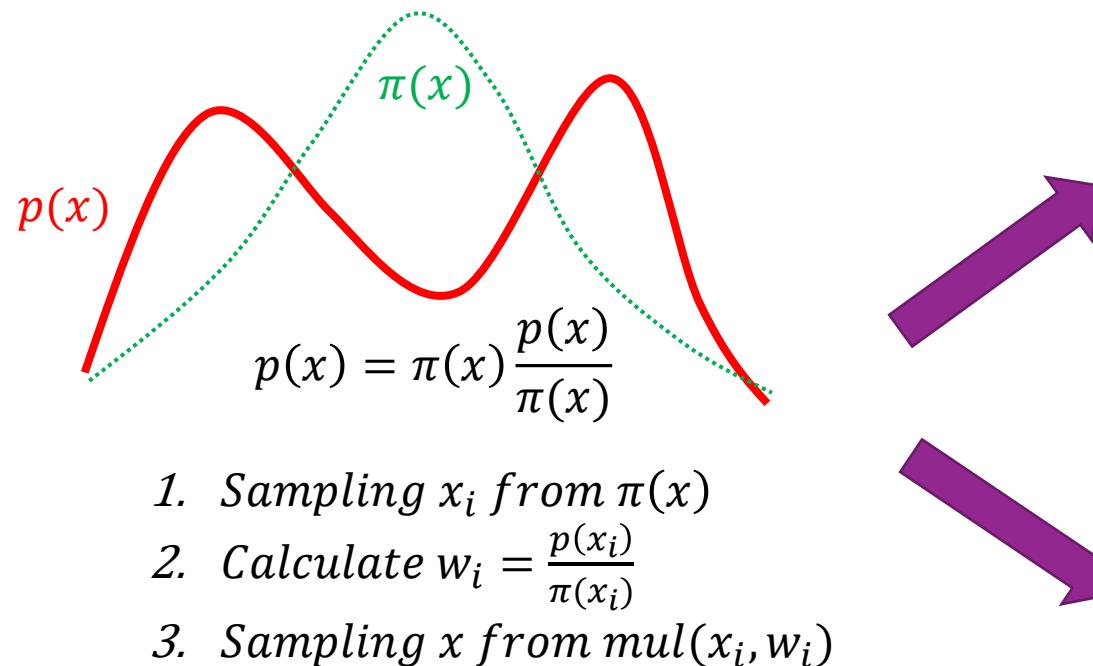
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)} G(\tau) \nabla_\theta \log p_\theta(\tau) \right]$$

Sample from $p_{\theta'}(\tau)$ Importance Weight

$$\mathbb{E}_{x \sim p}[f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Importance Sampling Review

- Important sampling adopts discrete multinomial to approximate arbitrary distribution. More sampling particles will have more accurate approximation.



Off-Policy with Importance Sampling

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\tau \sim p_{\theta}(\tau)}[G(\tau) \nabla_{\theta} \log p_{\theta}(\tau)] \\&= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_{\theta}(\tau)}{p_{\theta'}(\tau)} G^{\theta'}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) \right] \\&= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_{\theta}(s_t, a_t)}{p_{\theta'}(s_t, a_t)} G^{\theta'}(\tau) \nabla_{\theta} \log p_{\theta}(s_t, a_t) \right] \\&= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_{\theta}(a_t|s_t)}{p_{\theta'}(a_t|s_t)} \frac{p_{\theta}(s_t)}{p_{\theta'}(s_t)} G^{\theta'}(\tau) \nabla_{\theta} \log p_{\theta}(s_t, a_t) \right]\end{aligned}$$

$$J^{\theta'}(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_{\theta}(a_t|s_t)}{p_{\theta'}(a_t|s_t)} G^{\theta'}(\tau) \right]$$

Trust Region Policy Optimization (TRPO)

$$J_{TRPO}^{\theta'}(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_{\theta}(a_t|s_t)}{p_{\theta'}(a_t|s_t)} G^{\theta'}(\tau) \right]$$

Subject to $KL(\theta, \theta') < \delta$

Proximal Policy Optimization (PPO)

$$\nabla_{\theta'}^{PPO} J(\theta) = J^{\theta'}(\theta) - \beta KL(\theta, \theta')$$

$$J^{\theta'}(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_{\theta}(a_t|s_t)}{p_{\theta'}(a_t|s_t)} G^{\theta'}(\tau) \right]$$

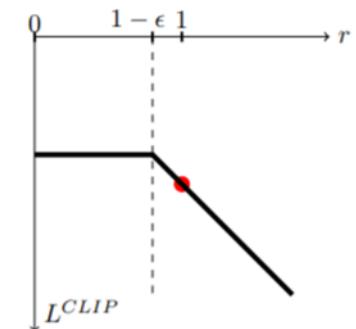
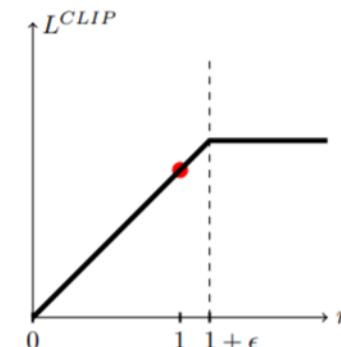
Proximal Policy Optimization

Proximal Policy Optimization (PPO)

$$\nabla_{\theta'}^{PPO} J(\theta) = J^{\theta'}(\theta) - \beta KL(\theta, \theta')$$

$$J^{\theta'}(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_{\theta}(a_t | s_t)}{p_{\theta'}(a_t | s_t)} G^{\theta'}(\tau) \right]$$

Proximal Policy Optimization (Clip Version)



$$\nabla_{\theta'}^{PPO2} J(\theta) \approx \sum_{s_t, a_t} \min \left(\frac{p_{\theta}(a_t | s_t)}{p_{\theta'}(a_t | s_t)} G^{\theta'}(\tau), \text{clip} \left(\frac{p_{\theta}(a_t | s_t)}{p_{\theta'}(a_t | s_t)}, 1 - \varepsilon, 1 + \varepsilon \right) G^{\theta'}(\tau) \right)$$

Actor-Critic

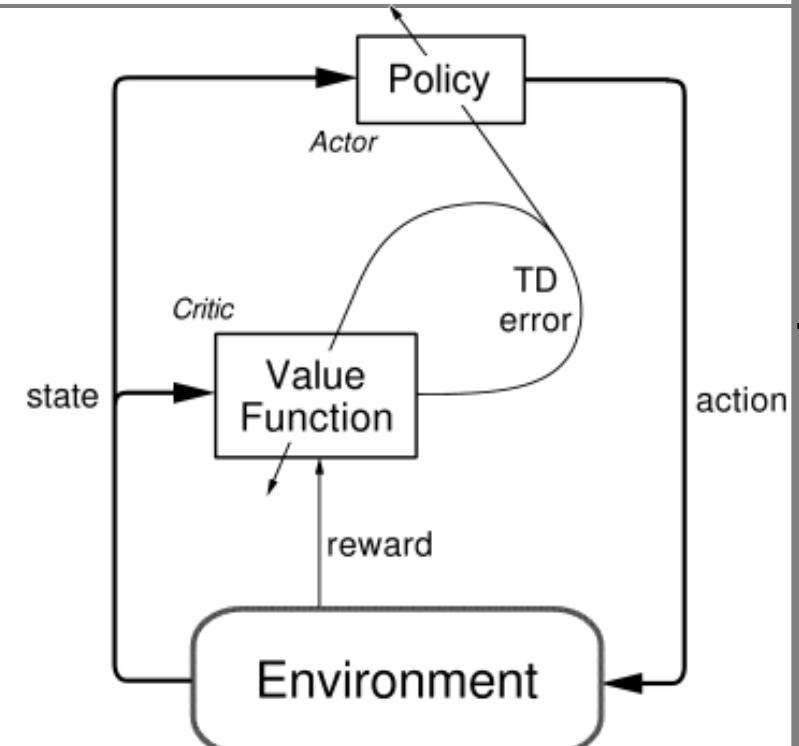
$$\nabla_{\theta} J(\theta) = E_{\pi} \left[\nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t | s_t) G_t(\tau) \right] \quad \text{Policy Gradient}$$

$$= E_{\pi} \left[\nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t | s_t) Q(s_t, a_t) \right] \quad \text{Actor-Critic}$$

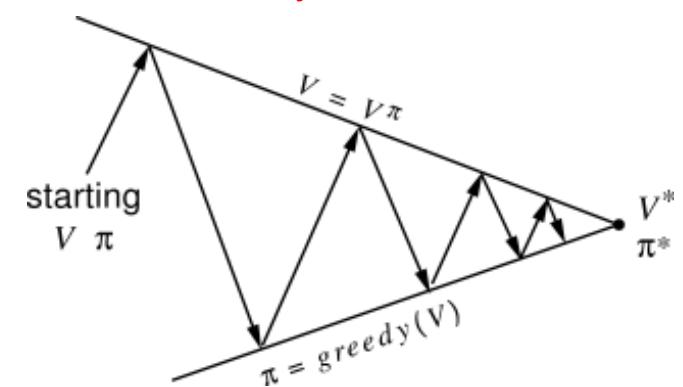
$$= E_{\pi} \left[\nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t | s_t) [Q(s_t, a_t) - B(s_t)] \right] \quad \text{Actor-Critic with Baseline}$$

$$= E_{\pi} \left[\nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t | s_t) [Q(s_t, a_t) - V(s_t)] \right] \quad \text{Advantage Actor-Critic}$$

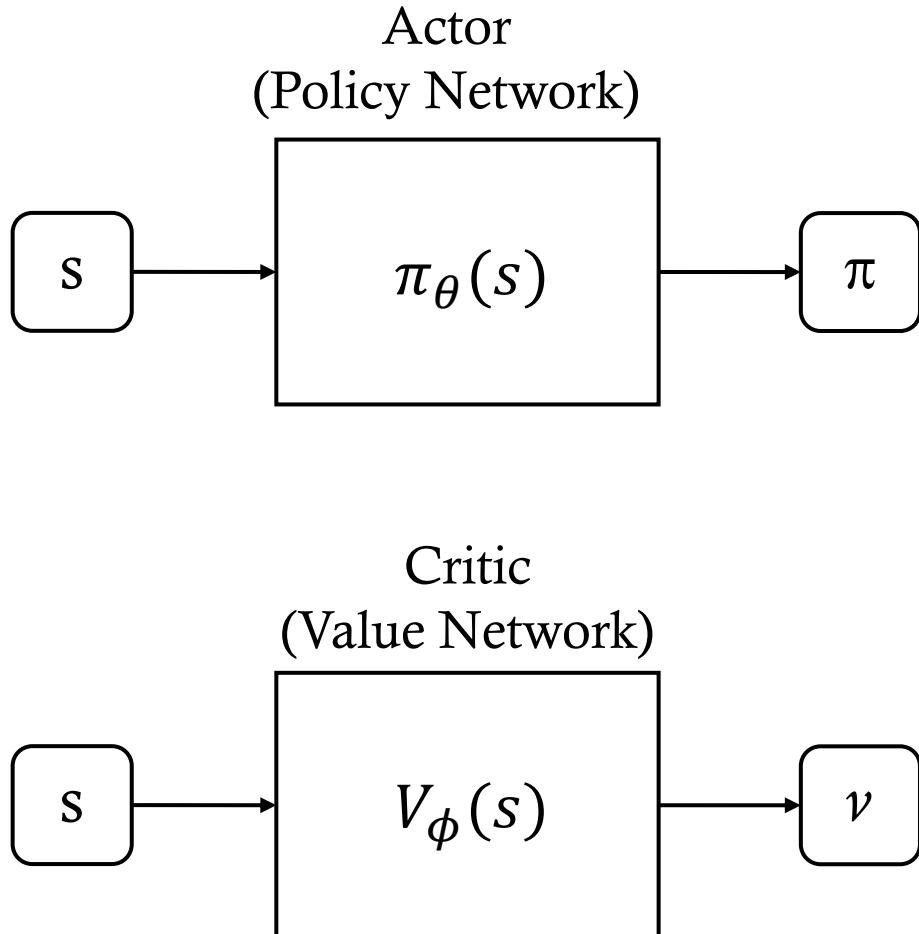
$$= E_{\pi} \left[\nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t | s_t) [r_t + \gamma V(s_{t+1}) - V(s_t)] \right] \quad \text{TD Actor-Critic}$$



Policy Iteration



TD Actor-Critic



Actor (Policy) Loss:

$$L(\theta) =$$

$$\mathbb{E}[-\log \pi_\theta(a|s) (r + \gamma V(s') - V(s)) - \beta H(\pi_\theta(a|s))]$$

Advantage

Entropy Term
(Encourage
Exploration)

On-Policy
Learning

Critic (Value) Loss:

$$L(\phi) = \mathbb{E}\left[\frac{1}{2}(r + \gamma V(s') - V(s))^2\right]$$

TD-Error

Deterministic Policy Gradient (DPG)

- The DPG algorithm maintains an actor function $\mu(s | \theta)$ which specifies the current policy by deterministically mapping states to a specific action.

- **Stochastic Policy**

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim P}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]]$$

- **Deterministic Policy**

$$Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim P}[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))]$$

- Without the stochastic property of the policy distribution, the policy gradient can be estimated simpler by chain rules.

$$\begin{aligned}\nabla_\theta J(\mu_\theta) &\approx \int_s P^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a) \Big|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s \sim P^\mu} [\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a) \Big|_{a=\mu_\theta(s)}]\end{aligned}$$

On-policy and Off-policy DPG

- On-policy DPG

$$\begin{aligned}\nabla_{\theta} J(\mu_{\theta}) &= \int_s P^{\mu}(s) \nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a) ds \\ &= \mathbb{E}_{s \sim P^{\mu}} [\nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a) |_{a=\mu_{\theta}(s)}]\end{aligned}$$

Update Parameters

$$\begin{aligned}Q^w(s, a) &\approx Q^{\mu}(s, a) \\ \delta_t &= r_t + \gamma Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t) \\ w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha_{\theta} \nabla_{\theta} \mu_{\theta}(s_t) \nabla_a Q^w(s_t, a_t) |_{a=\mu_{\theta}(s)}\end{aligned}$$

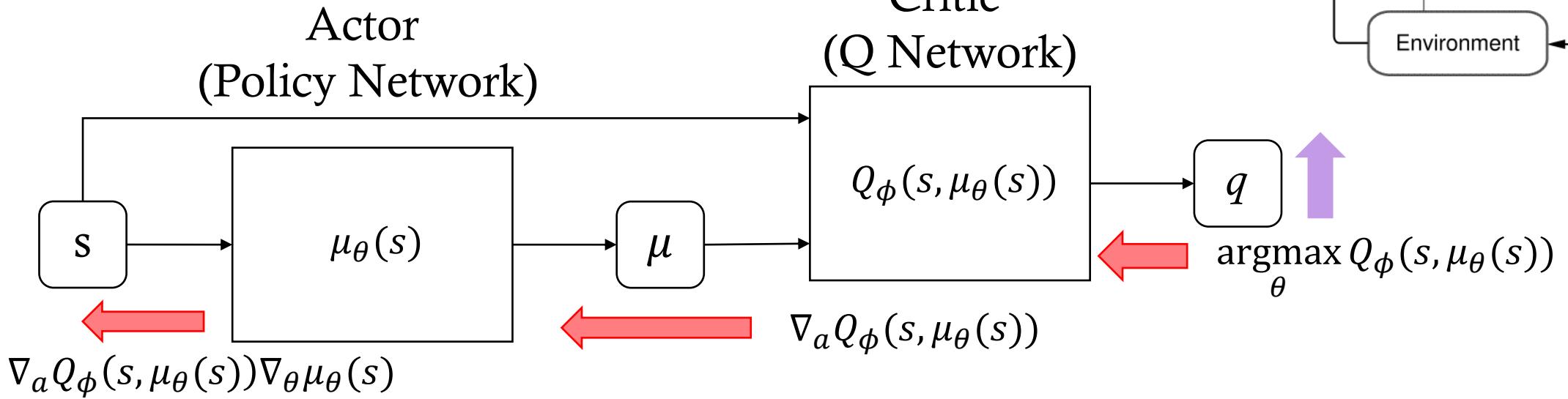
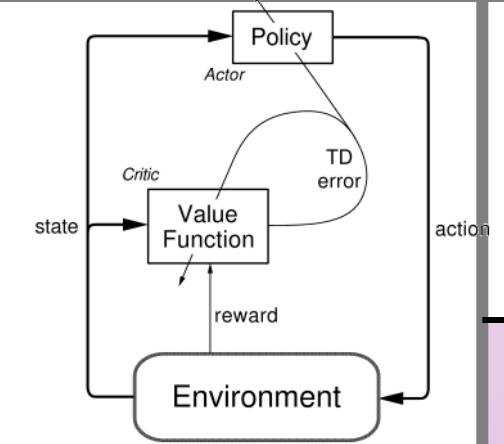
- Off-policy DPG

$$\begin{aligned}\nabla_{\theta} J_{\beta}(\mu_{\theta}) &\approx \int_s P^{\beta}(s) \nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a) ds \\ &= \mathbb{E}_{s \sim P^{\beta}} [\nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a) |_{a=\mu_{\theta}(s)}]\end{aligned}$$

Assume $\underset{a}{\operatorname{argmax}} Q(s, a) = \mu_{\theta}(s_{t+1})$

$$\begin{aligned}\delta_t &= r_t + \gamma Q^w(s_{t+1}, \mu_{\theta}(s_{t+1})) \xrightarrow{\text{↑}} - Q^w(s_t, a_t) \\ w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha_{\theta} \nabla_{\theta} \mu_{\theta}(s_t) \nabla_a Q^w(s_t, a_t) |_{a=\mu_{\theta}(s)}\end{aligned}$$

Deterministic Policy Gradient (DPG)



Actor (Policy) Loss:

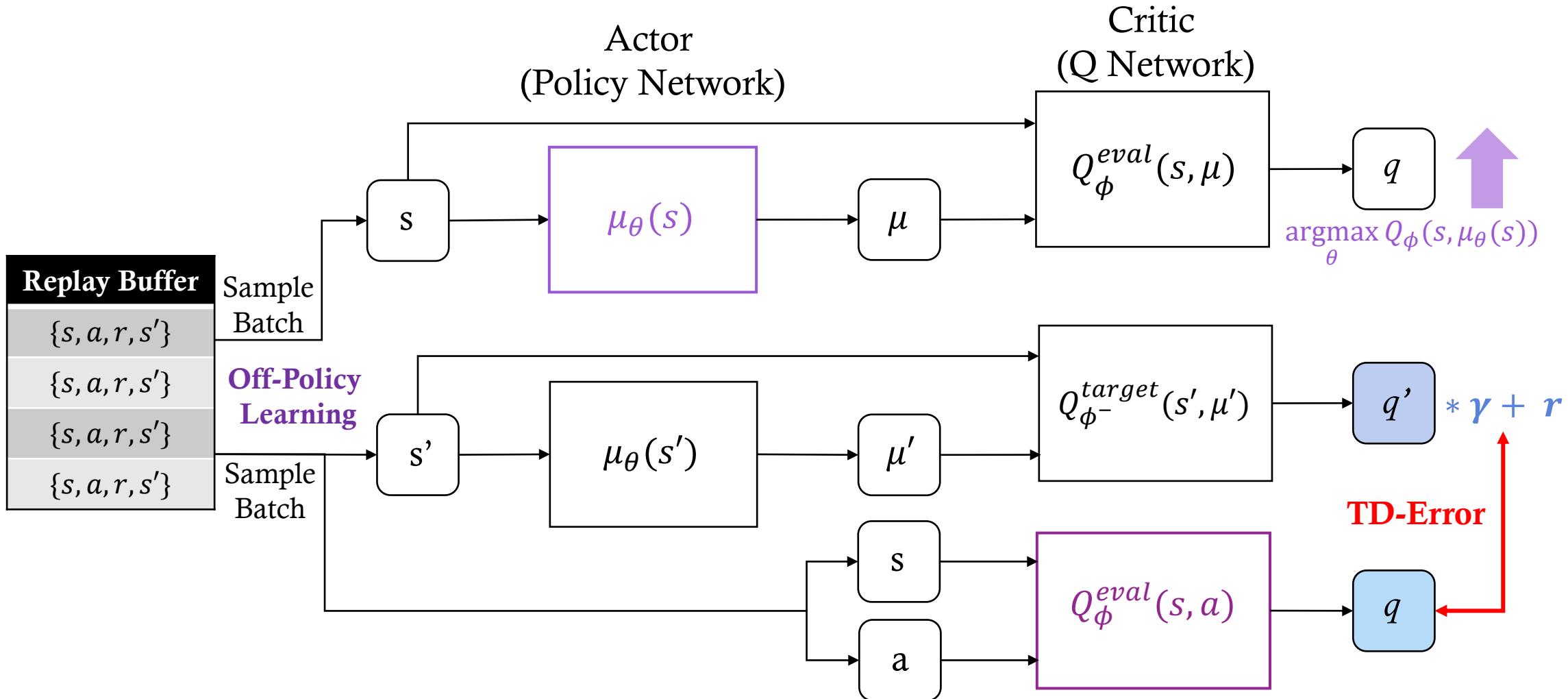
$$L(\theta) = \mathbb{E}[-Q_\phi(s, \mu_\theta(s))]$$

Critic (Q) Loss:

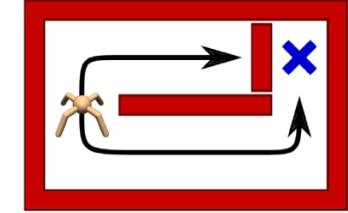
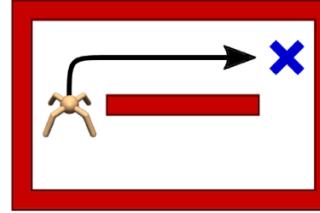
$$L(\phi) = \mathbb{E}\left[\frac{1}{2}\left(r + \gamma Q_\phi(s', \mu(s')) - Q(s, a)\right)^2\right]$$

TD-Error

Deep Deterministic Policy Gradient (DDPG)



Exploration Issue



- The formulation of classic RL framework implies an optimal deterministic policy, while it conflict to the stochastic policy that encourages the exploration of the environment.
- We usually add an regularized entropy term to maintain the stochastic property in stochastic policy algorithm such as policy gradient or actor-critic.

$$L_{actor}(\theta) = \mathbb{E}[-\log \pi_\theta(a|s) A(s) - \beta H(\pi_\theta(a|s))]$$

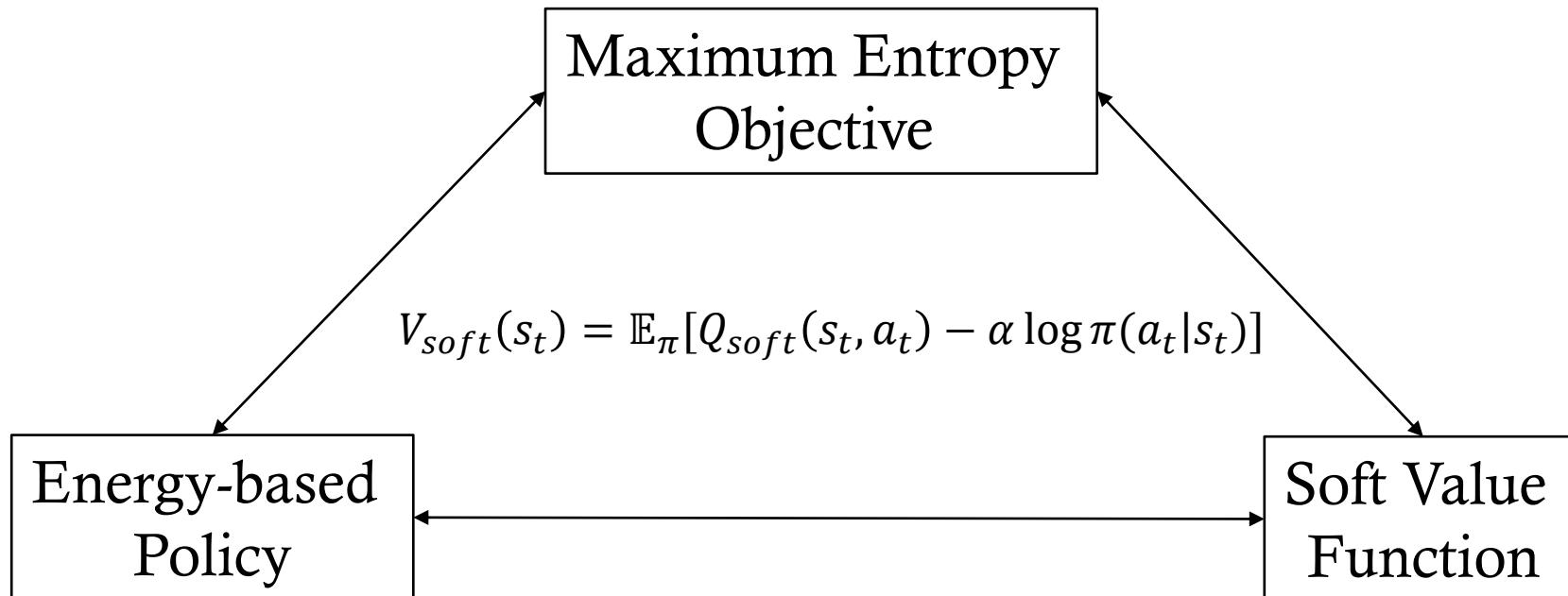
- An extension of classic RL framework is **Maximum Entropy Reinforcement Learning Framework**, which take the long-term entropy of the policy distribution into consideration to ensure the exploration of the environment.

$$\pi_{MaxEnt}^* = \operatorname{argmax}_\pi \sum_t \mathbb{E}_\pi [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))]$$

Consider the future entropy

Maximum Entropy RL Framework

$$\pi_{MaxEnt}^* = \operatorname{argmax}_{\pi} \sum_t \mathbb{E}_{\pi} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))]$$



$$\varepsilon(s_t, a_t) = -\frac{1}{\alpha} Q_{soft}(s_t, a_t)$$

$$\pi(a_t | s_t) \propto \exp(Q_{soft}(s_t, a_t))$$

$$\begin{aligned} V_{soft}(s_t) &= \log \int \exp\left(\frac{1}{\alpha} Q_{soft}(s_t, a)\right) da \\ &= \operatorname{softmax}_a Q_{soft}(s_t, a) \end{aligned}$$

Soft Actor-Critic

- Soft Actor Critic (SAC) algorithm is an off-policy algorithm based on the soft policy iteration of maximum entropy RL framework.
- SAC can be seen as the stochastic extension of the DDPG, which adopt the **reparameterization trick** to estimate the gradient of gaussian policy.

Actor (Policy) Loss:

$$L(\theta) = \mathbb{E}[\alpha \log \pi_\theta(f_\theta(\epsilon_t; s_t) | s_t) - Q_\phi(s_t, f_\theta(\epsilon_t; s_t))]$$

$$a_t = f_\theta(\epsilon_t; s_t)$$
 (Reparameterize)

Actor Loss (DDPG):

$$L(\theta) = \mathbb{E}[-Q_\phi(s, \mu_\theta(s))]$$

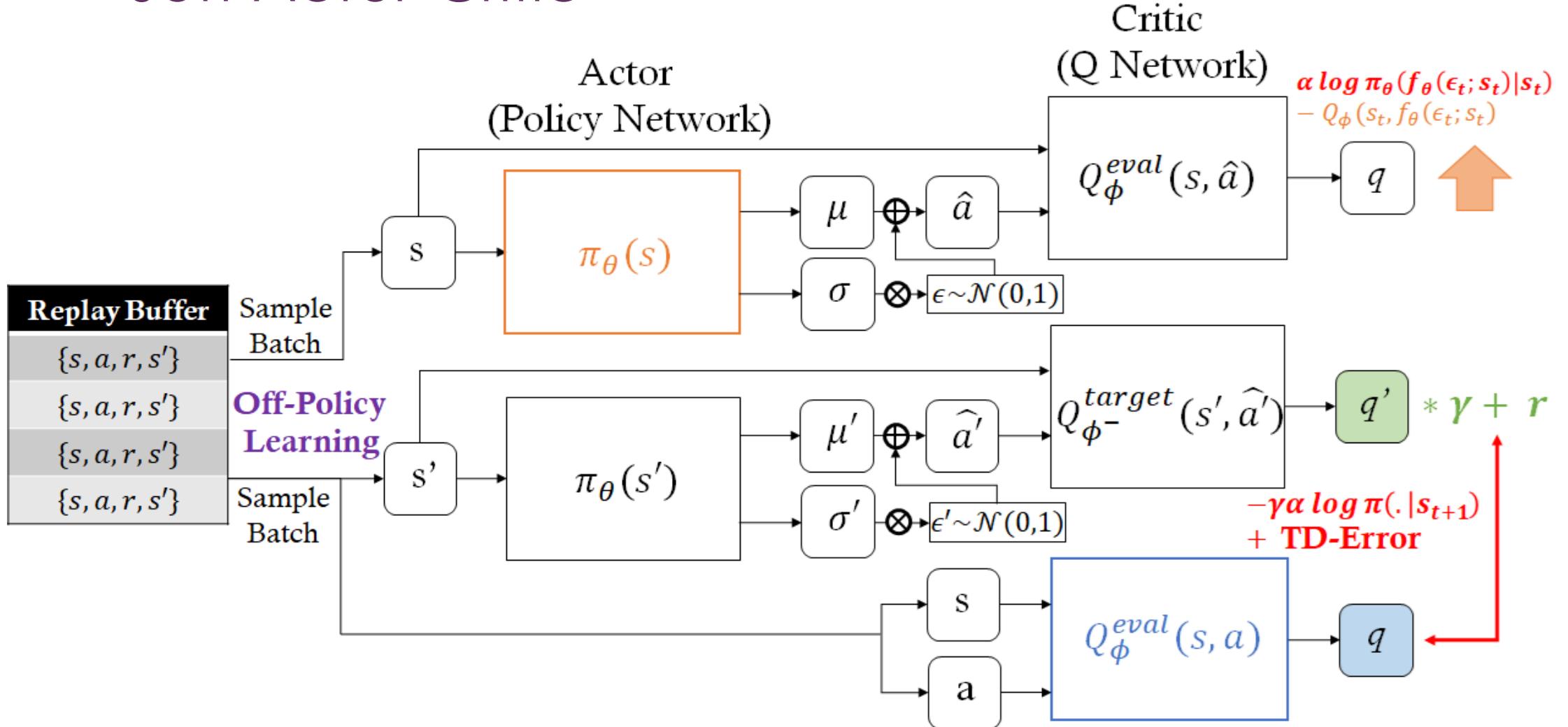
Critic Loss (DDPG):

$$L(\phi) = \mathbb{E}\left[\frac{1}{2}(r + \gamma Q_\phi(s', \mu(s')) - Q(s, a))^2\right]$$

Critic (Q) Loss:

$$L(\phi) = \mathbb{E}\left[\frac{1}{2}(-\gamma \alpha \log \pi(\cdot | s_{t+1}) + r(s_t, a_t) + \gamma Q_\phi(s_{t+1}, f_\theta(\epsilon_{t+1}; s_{t+1})) - Q_\phi(s_t, a_t))^2\right]$$

Soft Actor-Critic



Reparameterization

- Unbias gradient estimator for Gaussian sampling

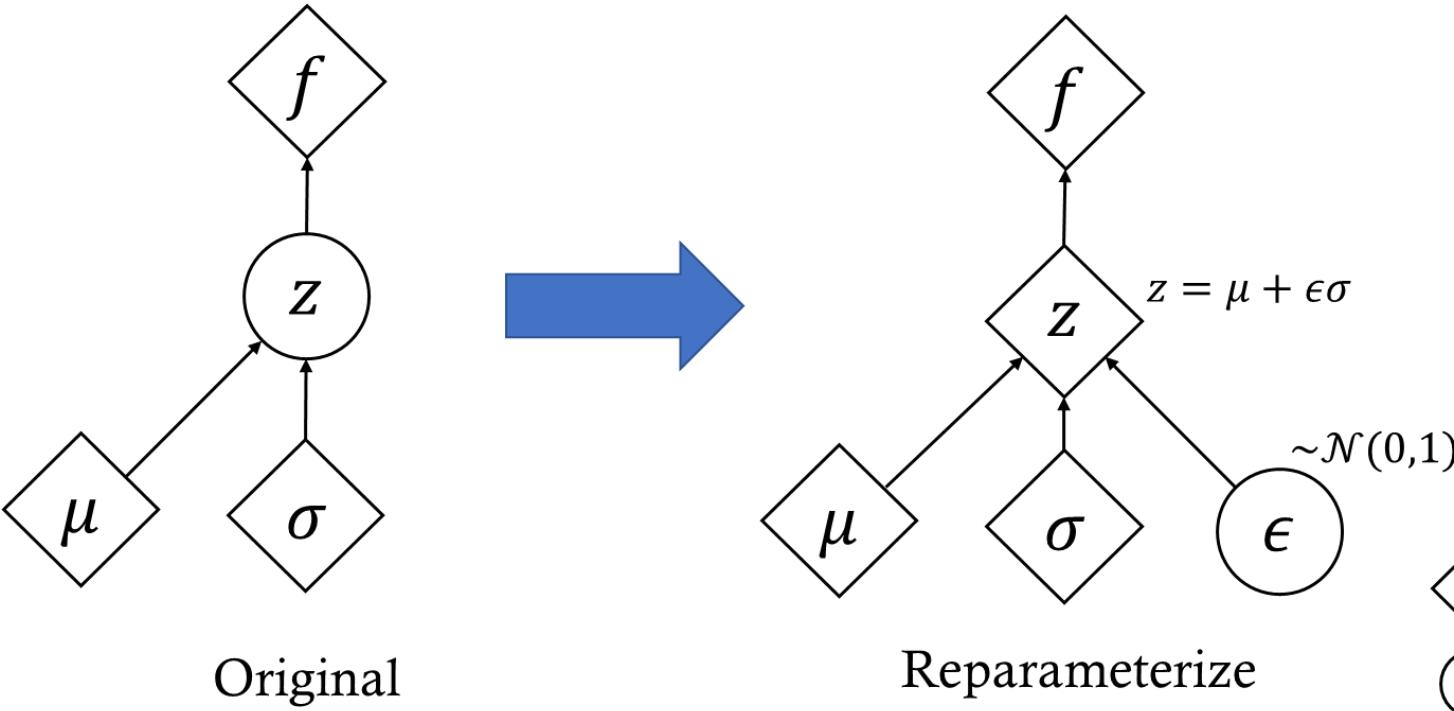
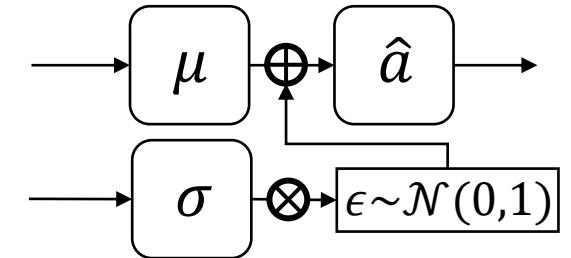


Diagram keys:
◊ : Deterministic Node
○ : Random Node

Reinforcement Learning Algorithms

