ROBOT INTELLIGENCE LAB

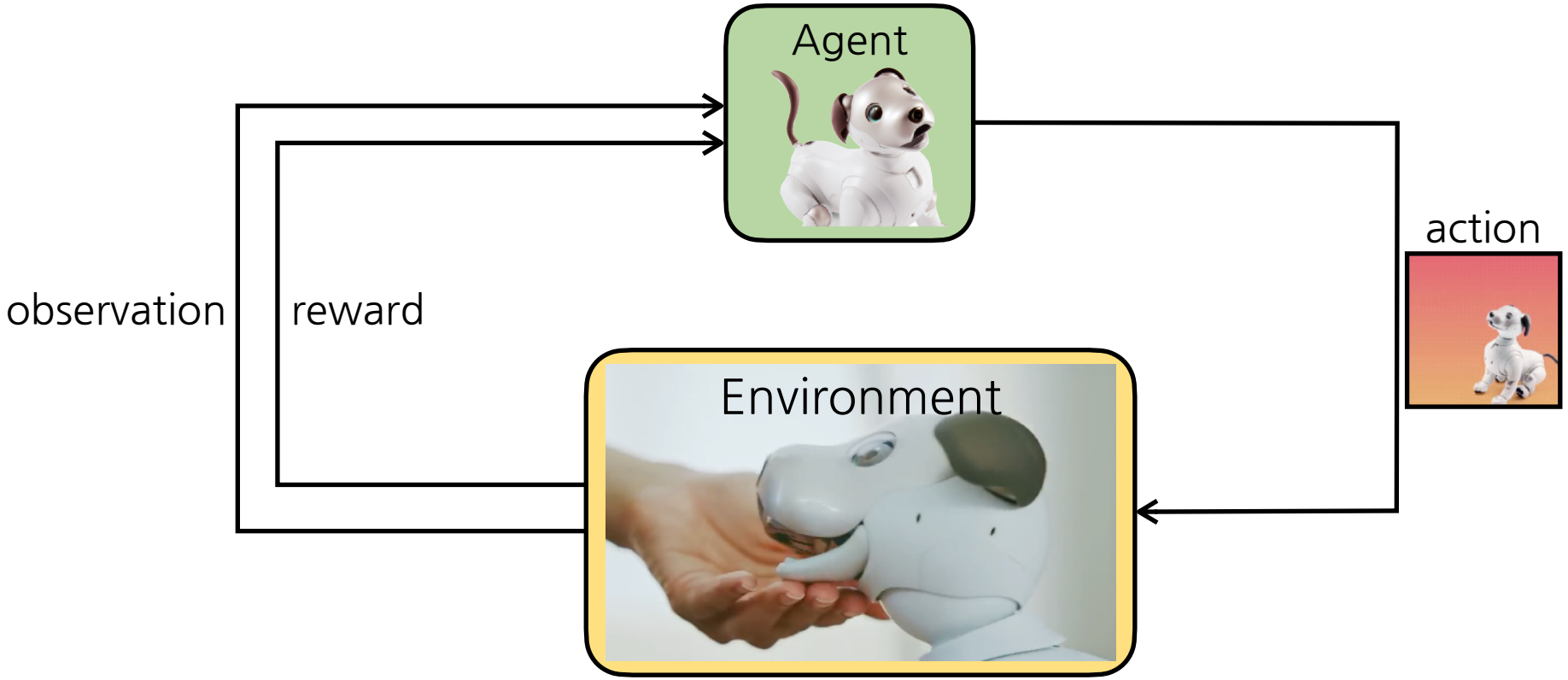
Introduction to Deep Learning

10 Reinforcement Learning

Sungjoon Choi, Korea University

kakaobrain

Reinforcement Learning



Key Question

ROBOT INTELLIGENCE LAB

What makes RL different from other ML methods (e.g., supervised or unsupervised learning)?

ROBOT INTELLIGENCE LAB

Markov Process

Introduction

ROBOT INTELLIGENCE LAB

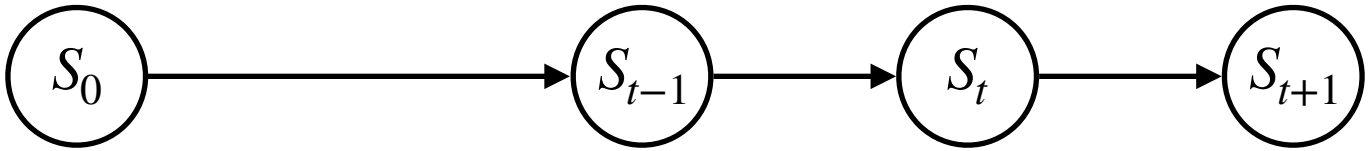
When can we formulate our problem as reinforcement learning?

Markov Process

ROBOT INTELLIGENCE LAB

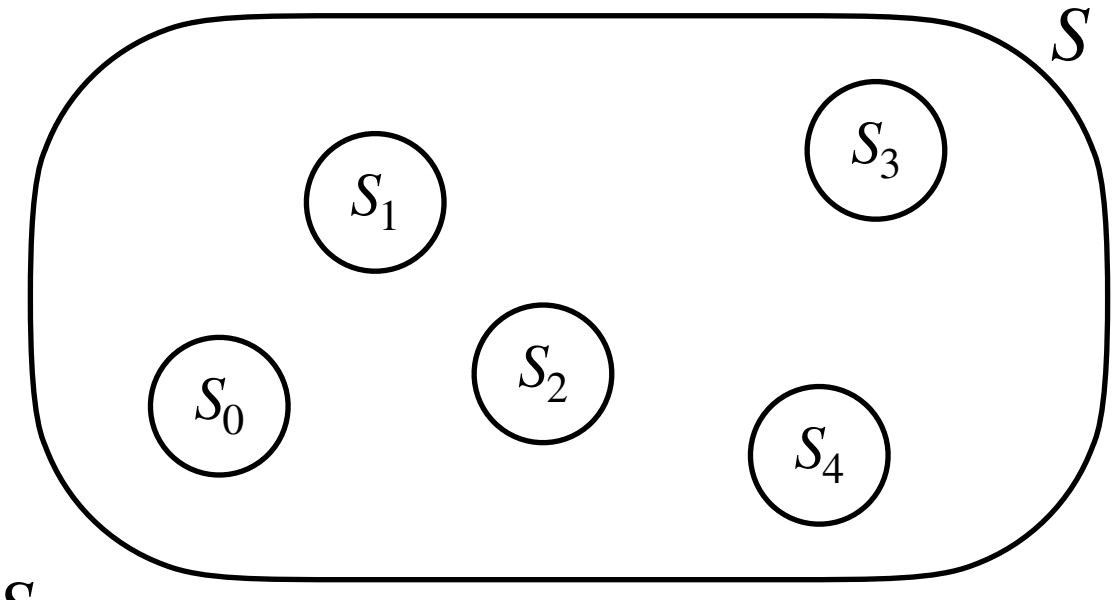
- A **Markov chain** is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event (aka **Markov** property).

- Random variable: S_t (state)



Markov Process

- **State space**
  - All possible state values



- Random variable: S_t
- Outcome: $S_t = s_t$

Markov Property

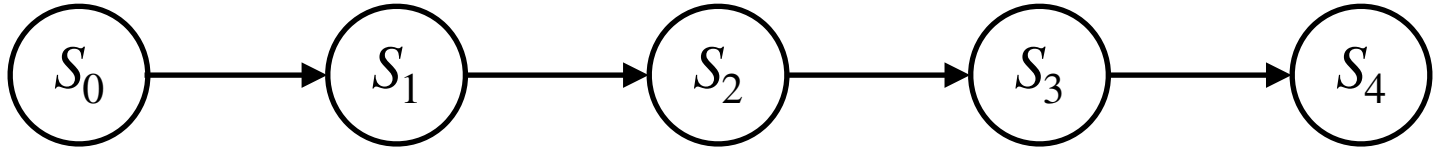
Andrey Markov (1856-1922)

- "Markov" (or Markov property) means that given the present state, the future and the past are independent.

$$P(\underbrace{S_{t+1} = s'}_{\text{future}} | \underbrace{S_t = s_t}_{\text{present}}, \underbrace{S_{t-1} = s_{t-1}, \ldots, S_0 = s_0}_{\text{past}}) = P(\underbrace{S_{t+1} = s'}_{\text{future}} | \underbrace{S_t = s_t}_{\text{present}})$$

Markov Process

- Given a state sequence $\{s_0, s_1, s_2, s_3, s_4\}$, how **plausible** is this?



$$P(S_0 = s_0, S_1 = s_1, S_2 = s_2, S_3 = s_3, S_4 = s_4)$$

$$= p(s_0, s_1, s_2, s_3, s_4)$$

Chain Rule

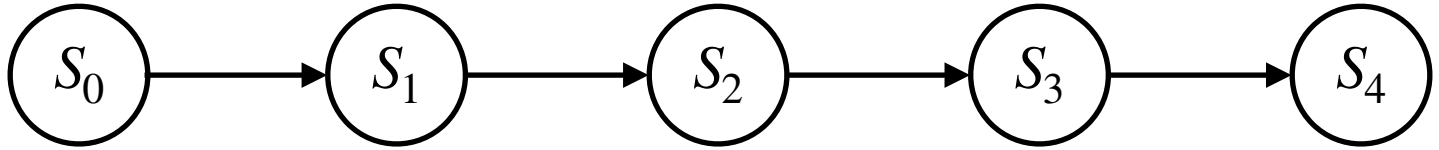
$$= p(s_4 \mid s_3, s_2, s_1, s_0)p(s_3 \mid s_2, s_1, s_0)p(s_2 \mid s_1, s_0)p(s_1 \mid s_0)p(s_0)$$

Markov Property

$$= p(s_4 \mid s_3)p(s_3 \mid s_2)p(s_2 \mid s_1)p(s_1 \mid s_0)p(s_0)$$

Markov Process

- Given a state sequence $\{s_0, s_1, s_2, s_3, s_4\}$, how **plausible** is this?

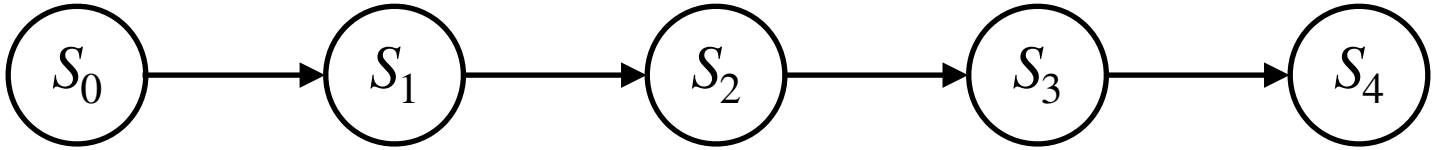


$$
\begin{aligned}
&P(S_0 = s_0, S_1 = s_1, S_2 = s_2, S_3 = s_3, S_4 = s_4) \\
&\quad = p(s_0, s_1, s_2, s_3, s_4) \\
&\quad = p(s_4 \mid s_3, s_2, s_1, s_0)p(s_3 \mid s_2, s_1, s_0)p(s_2 \mid s_1, s_0)p(s_1 \mid s_0)p(s_0) \\
&\quad = p(s_4 \mid s_3)p(s_3 \mid s_2)p(s_2 \mid s_1)p(s_1 \mid s_0)p(s_0)
\end{aligned}
$$

Transition Probability

Initial Probability

Markov Process

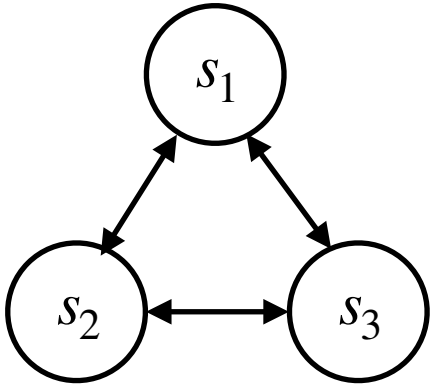


- A **Markov process** is specified by the **initial probability** and the **transition probabilities**.
  - Initial probability
    - $P(S_0 = s)$
  - Transition probability
    - $P(S_{t+1} = s' | S_t = s)$

Markov Process

- Initial distribution vector
  - $d_i = P(S_0 = s_i)$

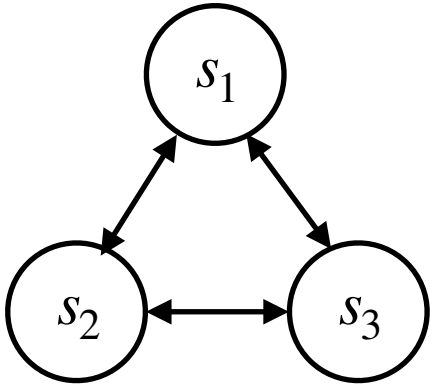
$$\mathbf{d} = \begin{matrix} S_1 \\ S_2 \\ S_3 \end{matrix} \begin{bmatrix} P(S_0 = s_1) \\ P(S_0 = s_2) \\ p(S_0 = s_3) \end{bmatrix}$$



Markov Process

- Transition probability matrix
  - $P_{(i,j)} = P(S_{t+1} = s_j \mid S_t = s_i)$

$$P = \begin{array}{c|ccc} & S_1 & S_2 & S_3 \\ \hline S_1 & p(s_1 \mid s_1) & p(s_2 \mid s_1) & p(s_3 \mid s_1) \\ S_2 & p(s_1 \mid s_2) & p(s_2 \mid s_2) & p(s_3 \mid s_2) \\ S_3 & p(s_1 \mid s_3) & p(s_2 \mid s_3) & p(s_3 \mid s_3) \end{array}$$



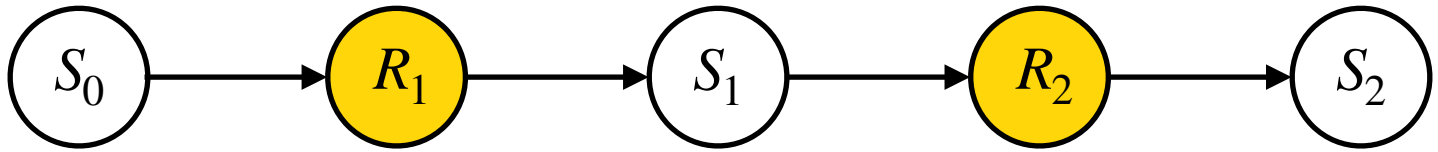
ROBOT INTELLIGENCE LAB

Markov Reward Process

Markov Reward Process

ROBOT INTELLIGENCE LAB

- Now, we obtain rewards as we move to states.



- We have two random variables.
  - State: S_t
  - Reward: R_t
- Since the reward is a random variable, we take expectation to compute the reward function.
  - Reward function: $r(s) = \mathbb{E}[R_{t+1} \mid S_t = s]$

Return

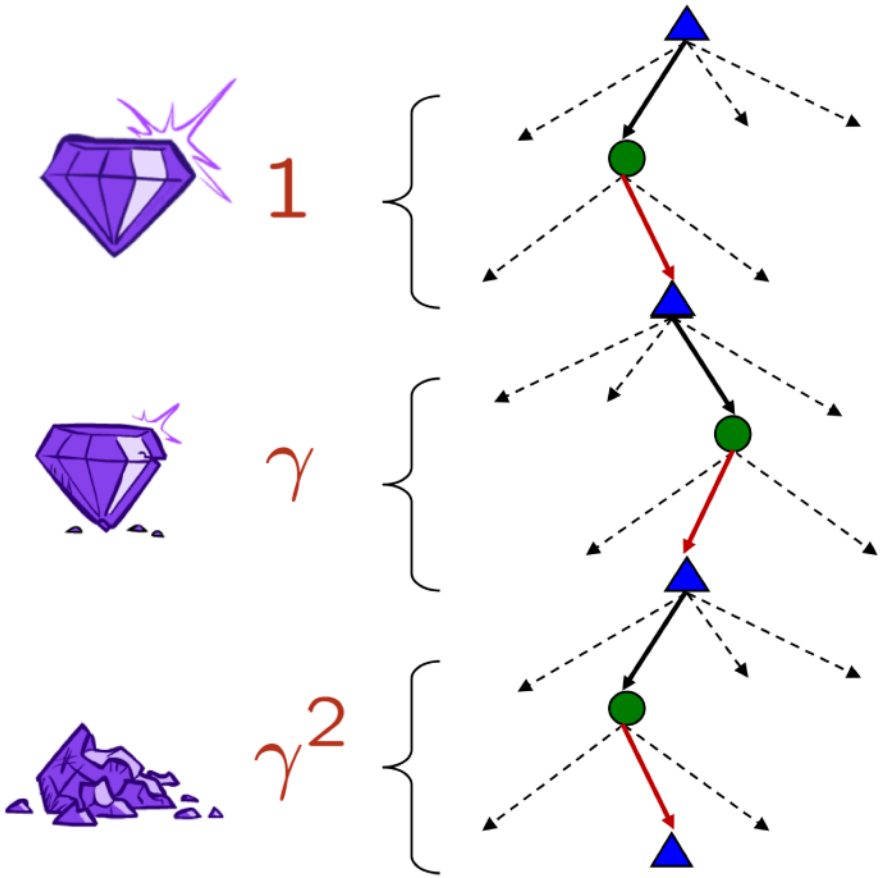
- **Return** G_t is the (discounted) sum of future rewards, and hence, also a random variable.

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

- Discount factor γ:

Discount Factor

- How to **discount**?
  - Each time we descend a level, we multiply the d
- Why **discount**?
  - Sooner rewards may have higher utility that late
  - It also helps the algorithms to converge.



(State) Value Function

ROBOT INTELLIGENCE LAB

- The **state-value function** $V(s)$ is a function of a state and is the expected return starting from the state s.

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+1} \cdots \mid S_t = s]
\end{aligned}
$$

(State) Value Function

ROBOT INTELLIGENCE LAB

- The (state) **value function** is the expected return (discounted sum of rewards) starting from state s.

$$\mathbb{E}[G_t \mid S_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots \mid S_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

Recursive equation

(State) Value Function

ROBOT INTELLIGENCE LAB

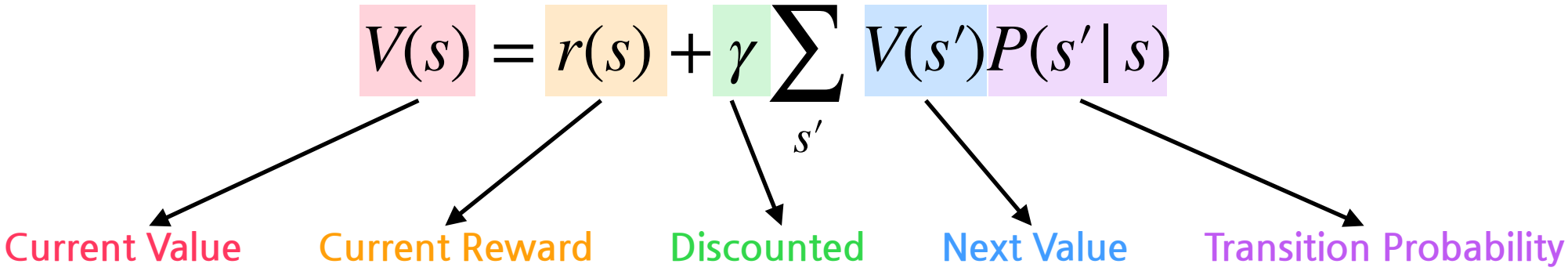
$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t \,|\, S_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \,|\, S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}[G_{t+1} \,|\, S_{t+1}] \,|\, S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \,|\, S_t = s] \\
&= \mathbb{E}[R_{t+1} \,|\, S_t = s] + \gamma \mathbb{E}[V(S_{t+1}) \,|\, S_t = s] \\
&= r(s) + \gamma \sum_{s'} V(s') P(s' \,|\, s)
\end{aligned}
$$

Bellman Equation for Markov Reward Processes

ROBOT INTELLIGENCE LAB

$$V(s) = r(s) + \gamma \sum_{s'} V(s')P(s' \mid s)$$

Bellman Equation for Markov Reward Processes



Bellman Equation for Markov Reward Processes

$$V(s) = r(s) + \gamma \sum_{s'} V(s')P(s' \mid s)$$

For multiple states s_1, s_2, s_3

$$V(s_1) = r(s_1) + \gamma \left(V(s_1)P(s_1 \mid s_1) + V(s_2)P(s_2 \mid s_1) + V(s_3)P(s_3 \mid s_1) \right)$$

$$V(s_2) = r(s_2) + \gamma \left(V(s_1)P(s_1 \mid s_2) + V(s_2)P(s_2 \mid s_2) + V(s_3)P(s_3 \mid s_2) \right)$$

$$V(s_3) = r(s_3) + \gamma \left(V(s_1)P(s_1 \mid s_3) + V(s_2)P(s_2 \mid s_3) + V(s_3)P(s_3 \mid s_3) \right)$$

Bellman Equation for Markov Reward Processes

$$V(s) = r(s) + \gamma \sum_{s'} V(s') P(s' \mid s)$$

(matrix form)

$$V = R + \gamma P V$$

$$V = R + \gamma P(s' \mid s) V$$

Bellman Equation for Markov Reward Processes

ROBOT INTELLIGENCE LAB

- The **Bellman equation** is a **linear equation**!

$$V = R + \gamma PV$$

Solve the equation w.r.t V

$$V = (I - \gamma P)^{-1} R$$

Markov Reward Process

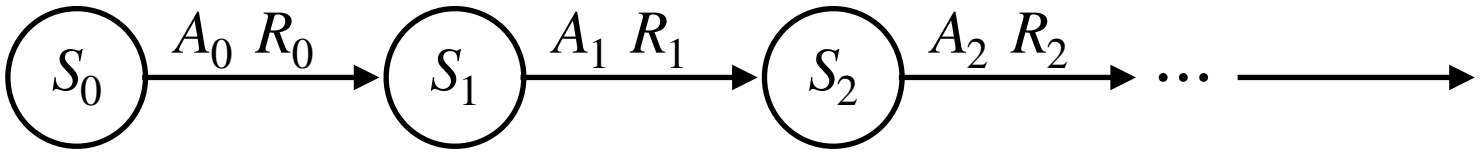
ROBOT INTELLIGENCE LAB

- Summary
  - S is a finite set of states.
  - d is an initial state distribution.
  - P is a state transition probability matrix.
    - $P_{ss'} = P(S_{t+1} = s' | S_t = s)$
  - r is a reward function.
    - $r(s) = \mathbb{E}\left[R_{t+1} | S_t = s\right]$
  - Note that there is no notion of **action** here.

ROBOT INTELLIGENCE LAB

Markov Decision Process

Markov Decision Process



- Now, we have three random variables:
  - State: S_t
  - Reward: R_t
  - **Action: A_t**

Markov Decision Process

ROBOT INTELLIGENCE LAB

- Formally, an MDP is a tuple (S, A, P, R, d):
  - A set of states $s \in S$.
  - A set of actions $a \in A$
  - A state transition function (or matrix)
    - $P(s' \mid s, a) = P(S_{t=1} = s \mid S_t = s, A_t = a)$
    - $P_{sas'} = P(s' \mid s, a)$
  - A reward function
    - $r(s) = \mathbb{E}[R_{t+1} \mid S_t = s]$
    - It depends on both state and action.
  - An initial state distribution d

Policy

ROBOT INTELLIGENCE LAB

- A **policy** is a **distribution over actions** given state.

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

  - In an MDP, a policy is a function of the current state s.
  - A policy is stationary (time-independent).
  - A deterministic policy can also be represented by a distribution.

Policy

ROBOT INTELLIGENCE LAB

- Given a fixed policy, a **Markov decision process** becomes and a **Markov reward process**.
- We denote $P^{\pi}_{ss'}$ a policy-conditioned state transition probability matrix:

$$P^{\pi}_{ss'} = \sum_{a} P(S_{t+1} = s' \,|\, S_t = s, A_t = a)\pi(A_t = a \,|\, S_t = s)$$
$$= \sum_{a} \pi(a \,|\, s) P_{sas'}$$

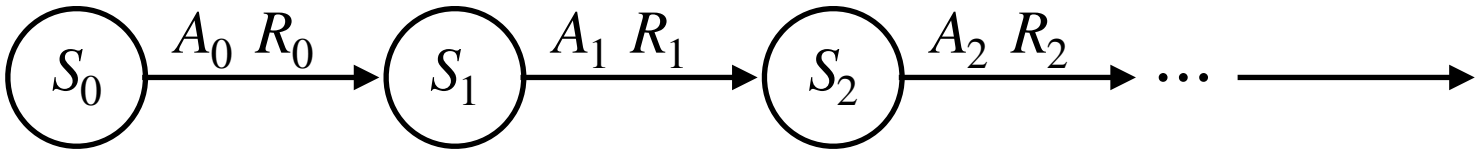
- Similarly, we can compute a policy-conditioned reward function

$$r^{\pi}(s) = \mathbb{E}[R_{t+1} \,|\, S_t = s] = \sum_{a} r(s, a)\pi(a \,|\, s)$$

Return

- Given a policy π , return G_t is determined.

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



State Value and State Action Value of Policy

ROBOT INTELLIGENCE LAB

- Given a fixed policy π
  - State value function, $V_\pi(s)$, is the expected return starting from state s, and then following policy

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

  - State-action value function, $Q_\pi(s, a)$, is the expected return starting from state s and action a, and then following policy

$$Q_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$$

Bellman Equation (1)

ROBOT INTELLIGENCE LAB

- Given a fixed policy π

$$V_\pi(s) = \sum_a \pi(a \mid s) Q_\pi(s, a)$$

Relation between a **state value function**, $V_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$, and a **state-action value function**. $Q_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$.

Bellman Equation (2)

ROBOT INTELLIGENCE LAB

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s]$$

$$V(s) = \mathbb{E}[G_t \mid S_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s]$$
$$= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$
$$= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1}] \mid S_t = s]$$
$$= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s]$$
$$= \mathbb{E}[R_{t+1} \mid S_t = s] + \gamma \mathbb{E}[V(S_{t+1}) \mid S_t = s]$$
$$= r(s) + \gamma \sum_{s'} V(s') P(s' \mid s)$$

Bellman equation of a Markov reward process

$$= r(s) + \sum_{s'} V_\pi(s') P_\pi(s' \mid s)$$

Definition of reward and transition probability

$$= \sum_a r(s,a)\pi(a \mid s) + \sum_{s'} V_\pi(s') \sum_a \pi(a \mid s) P(s' \mid s, a)$$

Simple rearrange

$$= \sum_a \left[r(s,a) + \sum_{s'} V_\pi(s') P(s' \mid s, a) \right] \pi(a \mid s)$$

Bellman Equation (3)

ROBOT INTELLIGENCE LAB

- Similarly, for $Q_\pi(s,a)$,

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s, A_t = a] \\
&= \mathbb{E}[R_{t+1} \,|\, S_t = s, A_t = a] + \gamma \mathbb{E}[G_{t+1} \,|\, S_t = s, A_t = a] \\
&= r(s,a) + \gamma \mathbb{E}[V(S_{t+1}) \,|\, S_t = s, A_t = a] \\
&= r(s,a) + \gamma \sum_{s'} V(s') P(s' \,|\, s, a)
\end{aligned}
$$

Bellman Equations of an MDP

ROBOT INTELLIGENCE LAB

- Summary

$$V_\pi(s) = \sum_a \pi(a \mid s) Q_\pi(s, a)$$

$$V_\pi(s) = \sum_a \left[r(s, a) + \sum_{s'} V_\pi(s') P(s' \mid s, a) \right] \pi(a \mid s)$$

$$Q_\pi(s, a) = r(s, a) + \gamma \sum_{s'} V(s') P(s' \mid s, a)$$

Bellman (Expectation) Equation of Q_π

- State-action value function $Q_\pi(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a, \pi]$
  - The expected return starting from state s, taking action a, and then follows the policy π

$$Q_\pi(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma \sum_{a'} Q_\pi(s',a')\pi(a' \mid s') \right] P(s' \mid s,a)$$

ROBOT INTELLIGENCE LAB

Optimality

Optimality

ROBOT INTELLIGENCE LAB

What does it mean by **solving** an MDP?

Optimal Value and Policy

ROBOT INTELLIGENCE LAB

- Optimal state value

$$V^*(s) = \max_{\pi} V_{\pi}(s)$$

- Optimal state-action value

$$Q^*(s,a) = \max_{\pi} Q_{\pi}(s,a)$$

- Optimal policy

$$\pi^*(a \mid s) = 1 \text{ for } a = \arg\max_{a'} Q^*(s,a')$$

Bellman Optimality Equation

ROBOT INTELLIGENCE LAB

$$V^*(s) = \max_a Q^*(s,a)$$

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

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

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

Summary

- Bellman **Optimality** Equation

- Bellman Equation

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

$$Q_\pi(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma V_\pi(s') \right] P(s'|s,a)$$

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

$$Q_\pi(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma \sum_{a'} Q_\pi(s',a') \pi(a'|s') \right] P(s'|s,a)$$

$$V^*(s) = \max_a Q^*(s,a)$$

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

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

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

$$\pi^*(a|s) = \arg\max_{a'} Q^*(s,a')$$

Value Iteration

Value Iteration

ROBOT INTELLIGENCE LAB

Given an MDP (S, A, P, R, d), how can we **solve** an MDP?

Value Iteration

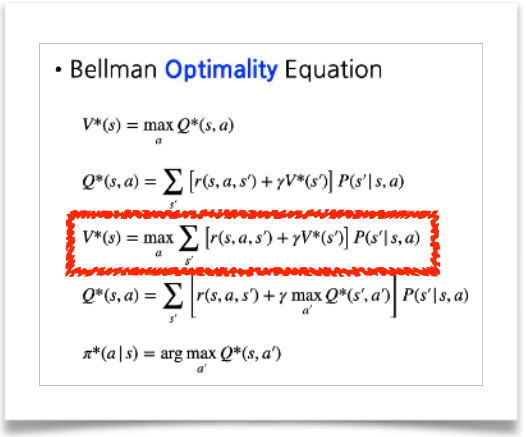
ROBOT INTELLIGENCE LAB

- Value iteration utilizes the **principle of optimality**.

$$V^*(s) = \max_{\pi} V_{\pi}(S)$$

- Then, the solution $V^*(s)$ can be found by one-step lookahead

$$V^*(s) = \max_{a} \sum_{s'} \left[r(s,a,s') + \gamma V^*(s') \right] P(s' \mid s,a)$$



- The main idea is to apply these updates iteratively, repeat until convergence.
  - It is guaranteed to converge to the unique optimal value.
- However, there is no explicit policy.

Value Iteration

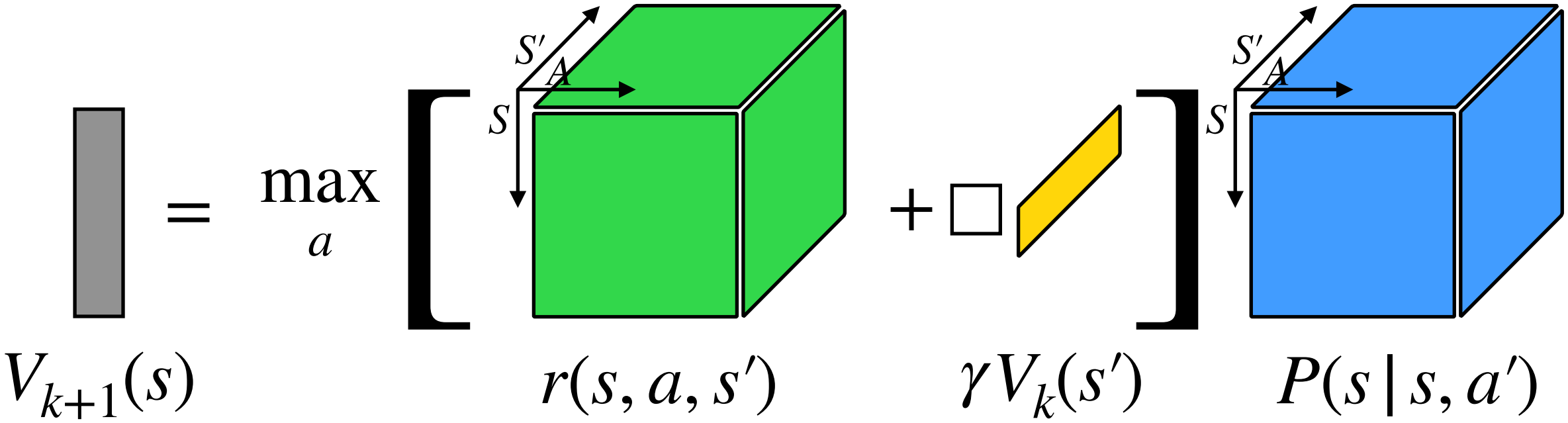
- Start from a random initial V_0
- Initialize V_{k+1} with $-\infty$
- Loop
  - For all states $s \in S$:
    - For all actions $a \in A$:
      - For all next states $s' \in S$:
        - If $V_{k+1}(s) \leq [r(s,a,s') + \gamma V_k(s')]P(s'|s,a)$
          - $V_{k+1}(s) = [r(s,a,s') + \gamma V_k(s')]P(s'|s,a)$
  - If $|V_{k+1} - V_k|_\infty = \max_s |V_{k+1}(s) - V_k(s)| \leq \epsilon$
    - stop

Bellman Optimality Equation

$$V_{k+1}(s) = \max_a \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$

Value Iteration

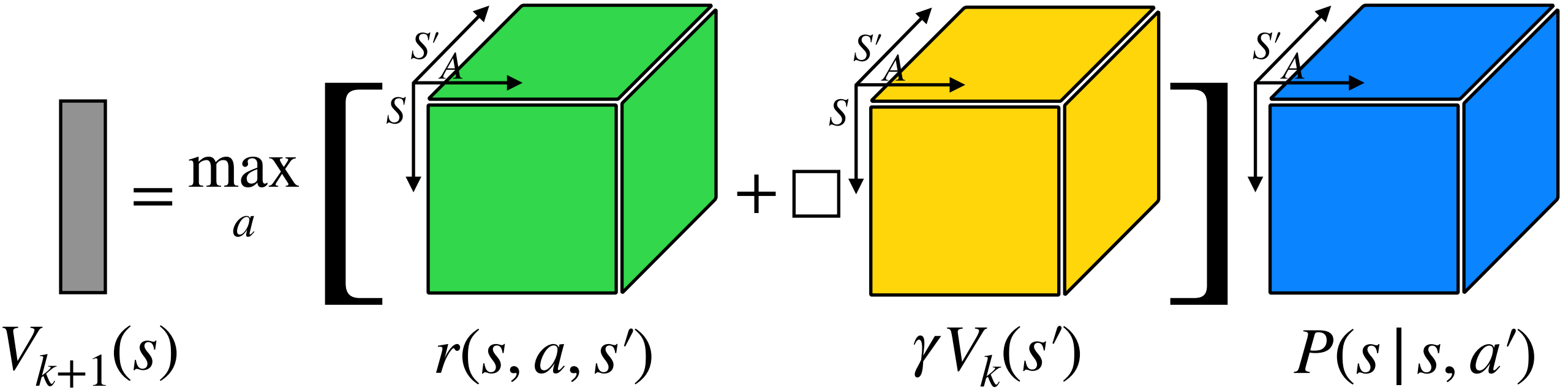
$$V_{k+1}(s) = \max_a \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$



Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s' \mid s,a)$$

(1) Tile (duplicate) to match dimensions

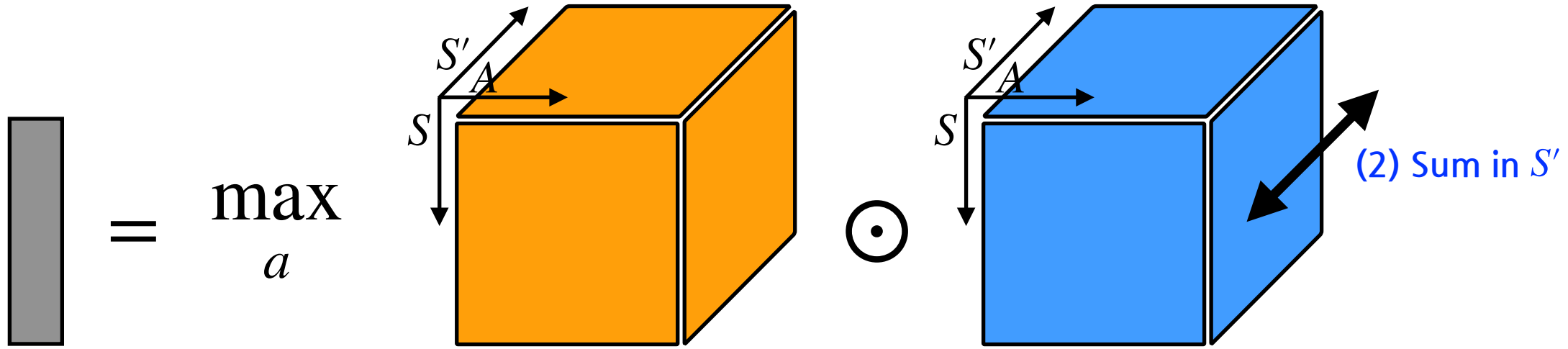


(2) Element-wise sum of $r(s,a,s')$ and $\gamma V_k(s,a,s')$

Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$

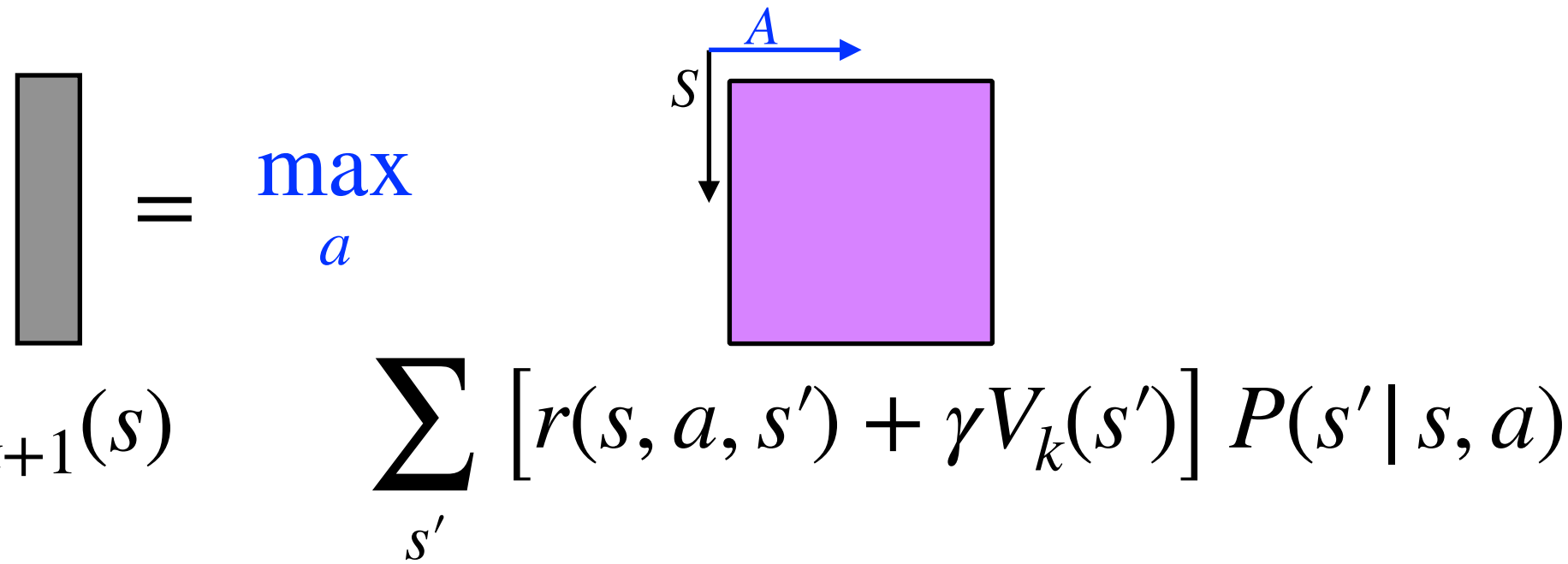
(1) Element-wise multiplication



$$V_{k+1}(s) = \max_a \; r(s,a,s') + \gamma V_k(s') \odot P(s|s,a')$$

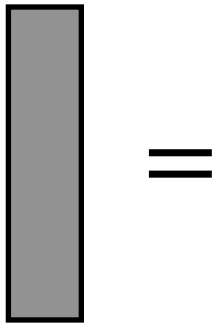
Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s' \mid s, a)$$

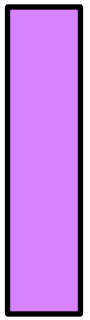


Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s' \mid s,a)$$



$=$



$$V_{k+1}(s) \qquad \max_a \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s' \mid s,a)$$

Value Iteration

ROBOT INTELLIGENCE LAB

- Does **value iteration** converge?
  - Yes.
- **Why** does it converge?
  - The Bellman (Optimality) Backup operation is a **contraction** mapping.

Bellman Optimality Equation

$$V_{k+1}(s) = \max_{a} \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$

Value Iteration

ROBOT INTELLIGENCE LAB

Bellman Optimality Backup Operator

$$TX(s) = \max_a \sum_{s'} \left[r(s,a,s') + \gamma X(s') \right] P(s' \mid s,a)$$

$$X \in \mathbb{R}^{|S|}$$

- Value iteration can be represented as $V_{k+1} = TV_k$.
- Bellman optimality equation becomes $V^* = TV^*$.
- Then, we can show that T is a γ-contraction mapping:
  - $|TU - TV|_\infty \leq \gamma |U - V|_\infty$

Value Iteration

ROBOT INTELLIGENCE LAB

- Value iteration can be represented as $V_{k+1} = TV_k$.
- Bellman optimality equation becomes $V^* = TV^*$.
- Then, we can show that T is a γ-contraction mapping:
  - $|TU - TV|_\infty \leq \gamma |U - V|_\infty$
  - $|TU - TV|_\infty = \max_s |TU(s) - TV(s)|$

$$\leq \max_s \left| \max_a \sum_{s'} \gamma |U(s') - V(s')| P(s'|s,a) \right|$$

$$\leq \max_s \left| \max_a \max_{s'} \gamma |U(s') - V(s')| \right|$$

$$= \gamma \max_{s'} |U(s') - V(s)| = \gamma |U - V|_\infty$$

Q-Value Iteration

ROBOT INTELLIGENCE LAB

- However, it is not straightforward to come up with an **optimal policy** solely from $V^*(s)$.
- Hence, we use following relations between $V^*(s)$ and $Q^*(s,a)$ (aka the Bellman optimality equation).

$$V^*(s) = \max_a Q^*(s,a)$$

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

- Bellman Equation

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

$$Q_\pi(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma V_\pi(s') \right] P(s'|s,a)$$

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

$$Q_\pi(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma \sum_{a'} Q_\pi(s',a') \pi(a'|s') \right] P(s'|s,a)$$

- Bellman Optimality Equation

$$V^*(s) = \max_a Q^*(s,a)$$

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

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

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

$$\pi^*(a|s) = \arg\max_{a'} Q^*(s,a')$$

Q-Value Iteration

ROBOT INTELLIGENCE LAB

- Start from the random initial V_0
- For all states $s \in S$:

$$Q_k(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s' \mid s,a)$$

$$V_{k+1}(s) = \max_{a'} Q_k(s,a')$$

- We now have an explicit form of the policy:

$$\pi_{k=1}(a \mid s) = 1 \text{ for } a = \arg\max_{a'} Q_k(s,a')$$

  - Note that this policy is **deterministic**.

ROBOT INTELLIGENCE LAB

Policy Iteration

Policy Iteration

ROBOT INTELLIGENCE LAB

- Basic concept
  - Step1) Predict the value of the policy
    - Evaluate the expected return of the current policy
  - Step2) Improve the policy
    - Update the current policy to get a better expected return
  - Iterate

Policy Iteration

ROBOT INTELLIGENCE LAB

- **Step 1: Policy evaluation**: calculate the value function for a fixed policy until convergence.

- **Step 2: Policy improvement**: update the policy using one-step lookahead using the converged value function.

- Repeat steps until the **policy** converges.

- It is guaranteed to converge to the optimal policy.
- It often converges much faster than value iteration.

Policy Iteration

ROBOT INTELLIGENCE LAB

- **Step 1: Policy evaluation**: calculate the value function for a fixed policy until convergence.
  - Start from a random initial V_0.
  - Iterate until value converges:

$$V_{k+1}(s) = \sum_a \pi_i(a \mid s) \sum_{s'} [r(s, a, s') + \gamma V_k(s')] P(s' \mid s, a)$$

- **Step 2: Policy improvement**: update the policy using one-step lookahead using the converged value function.
  - One-step lookahead:

$$Q_{\pi_k}(s, a) = \sum_{s'} [r(s, a, s') + \gamma V_{\pi_i}(s')] P(s' \mid s, a)$$

$$\pi_{i+1}(a \mid s) = 1 \text{ for } a = \arg\max_{a'} Q_{\pi_i}(s, a')$$

Policy Evaluation

ROBOT INTELLIGENCE LAB

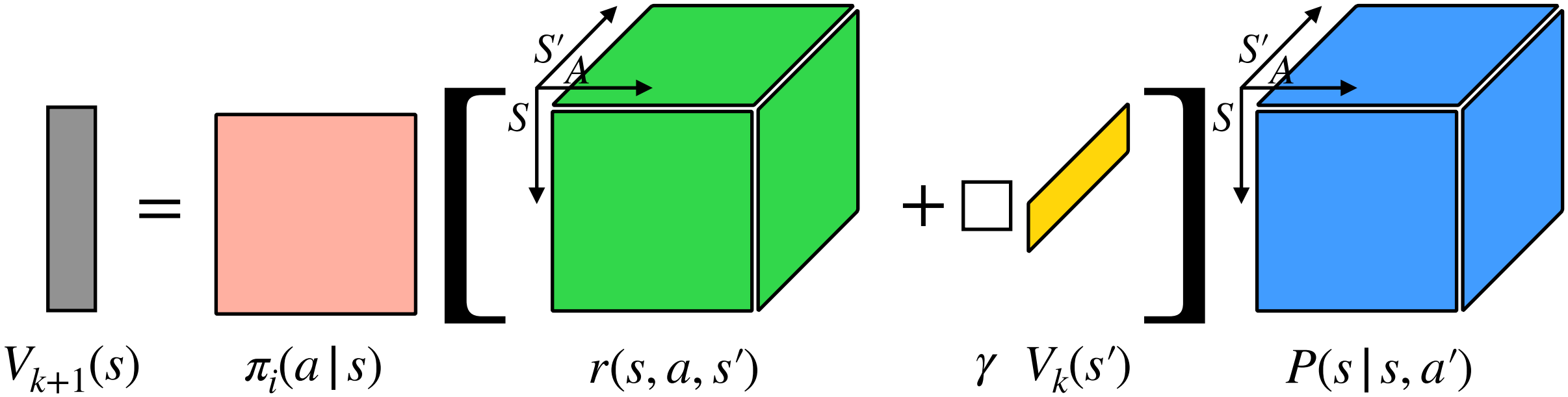
- Start from a random initial V_0
- Initialize V_{k+1} with a zero vector.
- Loop
  - For all states $s \in S$:
    - For all actions $a \in A$:
      - For all next states $s' \in S$:
        - $V_{k+1}(s) = V_k(s) + [r(s,a,s') + \gamma V_k(s')]P(s'|s,a)\pi_i(a|s)$
  - If $|V_{k+1} - V_k|_\infty = \max_s |V_{k+1}(s) - V_k(s)| \leq \epsilon$
    - stop

Bellman Equation

$$V_{k+1}(s) = \sum_a \pi_i(a|s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s')\right] P(s'|s,a)$$

Policy Evaluation

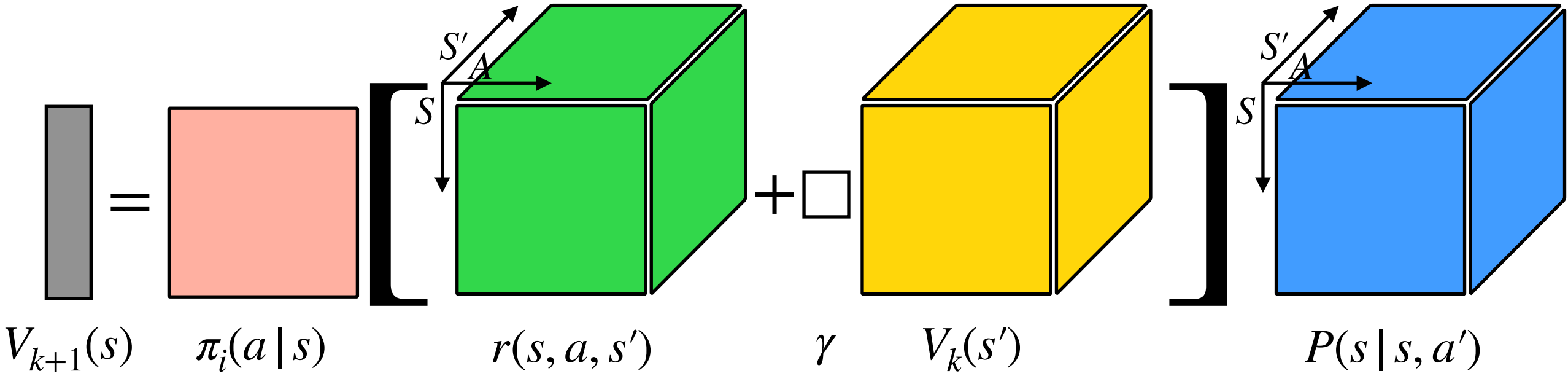
$$V_{k+1}(s) = \sum_{a} \pi_i(a|s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$



Policy Evaluation

$$V_{k+1}(s) = \sum_a \pi_i(a|s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$

(1) Tile (duplicate) to match dimensions



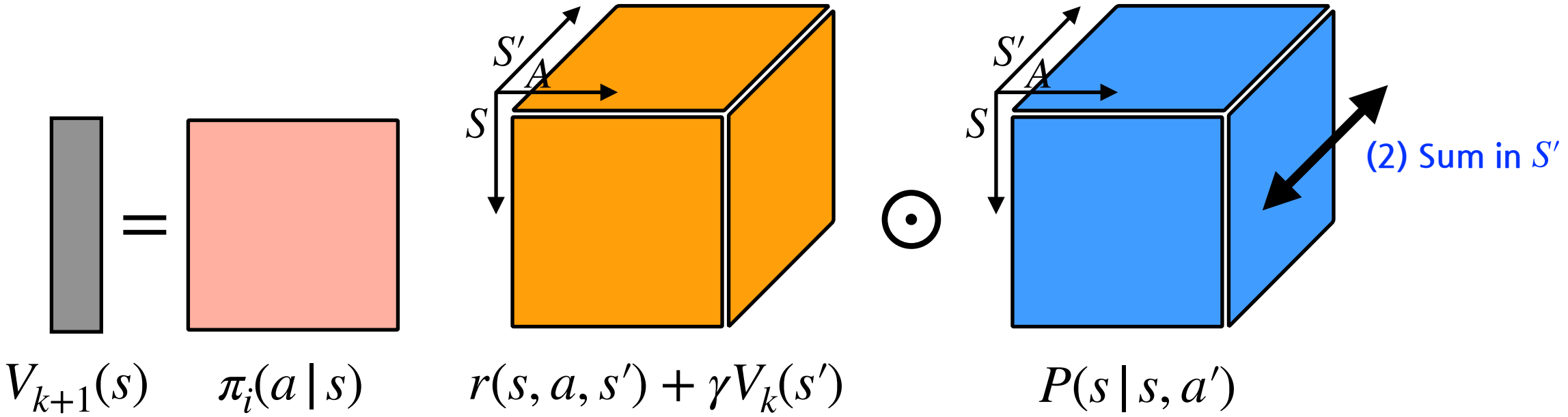
$V_{k+1}(s)$ $\pi_i(a|s)$ $r(s,a,s')$ γ $V_k(s')$ $P(s|s,a')$

(2) Element-wise sum of $r(s,a,s')$ and $\gamma V_k(s,a,s')$

Policy Evaluation

$$V_{k+1}(s) = \sum_a \pi_i(a|s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$

(1) Element-wise multiplication



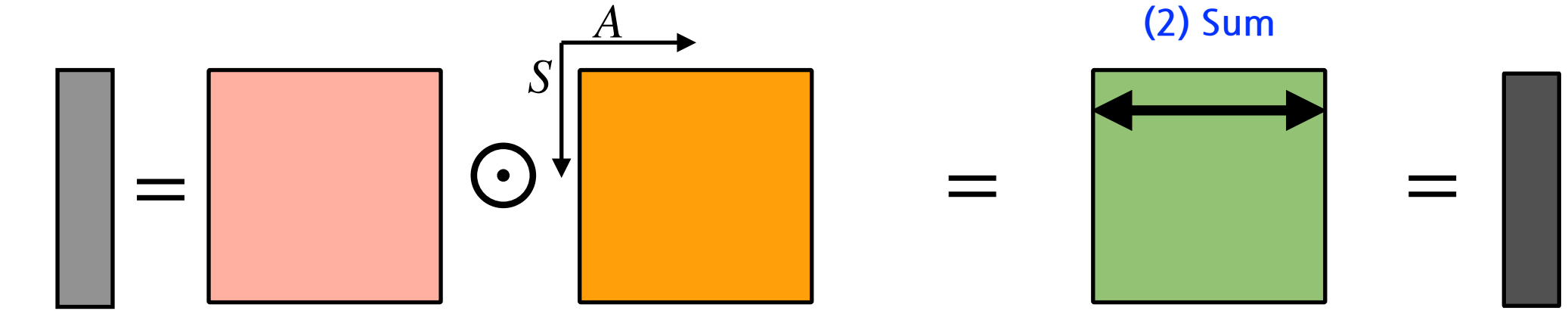
$V_{k+1}(s)$ $\pi_i(a|s)$ $r(s,a,s') + \gamma V_k(s')$ $P(s|s,a')$

Policy Evaluation

$$V_{k+1}(s) = \sum_a \pi_i(a|s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$

(1) Element-wise multiplication

(2) Sum



$V_{k+1}(s)$ $\pi_i(a|s)$ $\sum_{s'} [r(s,a,s') + \gamma V_k(s')] P(s'|s,a)$

Policy Evaluation

ROBOT INTELLIGENCE LAB

- Does policy evaluation converge?
  - Yes.
- Why does it converge?
  - The Bellman Backup operation is a **contraction** mapping.

Policy Evaluation

Bellman Backup Operator

$$T_\pi X(s) = \sum_a \pi(a \mid s) \sum_{s'} \left[r(s, a, s') + \gamma X(s') \right] P(s' \mid s, a)$$

$$X \in \mathbb{R}^{|S|}$$

$$\begin{aligned}
|T_\pi U - T_\pi V|_\infty &= \max_s |T_\pi U(s) - T_\pi V(s)| \\
&= \max_s \left| \sum_a \pi(a \mid s) \sum_{s'} [r(s, a, s') + \gamma U(s')] P(s' \mid s, a) - \sum_a \pi(a \mid s) \sum_{s'} [r(s, a, s') + \gamma V(s')] P(s' \mid s, a) \right| \\
&= \max_s \left| \gamma \sum_a \sum_{s'} (U(s') - V(s')) P(s' \mid s, a) \pi(a \mid s) \right| \\
&\leq \max_s \left| \gamma \sum_a \max_{s'} |U(s') - V(s')| \right| \\
&= \gamma \max_{s'} |U(s') - V(s')| = \gamma |U - V|_\infty
\end{aligned}$$

Policy Improvement

ROBOT INTELLIGENCE LAB

- Let V_{π_i} be the value evaluated from the policy π_i.
- Our goal is to find a policy π_{i+1}.

- Compute the state-action value $Q_{\pi_i}(s,a)$ from state value $V_{\pi_i}(s)$:

$$Q_{\pi_i}(s,a) = \sum_{s'} [r(s,a,s') + \gamma V_{\pi_i}(s')] P(s' \mid s,a)$$

- The policy is updated via:

$$\pi_{i+1}(a \mid s) = 1 \text{ for } a = \arg\max_{a'} Q_{\pi_i}(s,a)$$

Policy Improvement

ROBOT INTELLIGENCE LAB

- It is guaranteed that the **policy improvement** improves the value.
- The value of the updated policy is

$$V_{\pi_{i+1}}(s) = \sum_a Q_{\pi_i}(s,a)\pi_{i+1}(a\,|\,s)$$

- We can show that $V_{\pi_{i+1}}(s) \geq V_{pi_i}(s)$, hence is improves the value function.

$$\begin{aligned} V_{\pi_{i+1}}(s) &= \sum_a Q_{\pi_i}(s,a)\pi_{i+1}(a\,|\,s) \\ &= \max_{a'} Q_{\pi_i}(s,a') \\ &\geq \sum_s Q_{\pi_i}(s,a)\pi_i(a\,|\,s) \\ &= V_{\pi_i}(s) \end{aligned}$$

Policy Iteration

ROBOT INTELLIGENCE LAB

- **Step 1: Policy evaluation**: Evaluate V_π.
- **Step 2: Policy improvement**: Generate π' where $V_{\pi'} \geq V_\pi$.

- **Policy iteration** is often more effective that **value iteration**, why?
  - It is often the case that a policy function reaches the optimal policy (policy iteration) much sooner than a value function reaches the optimal value function (value iteration).
  - In many cases, we are more interested in finding the optimal policy function rather than the optimal value function.

ROBOT INTELLIGENCE LAB

Monte-Carlo Learning

Monte-Carlo Learning

ROBOT INTELLIGENCE LAB

- **Monte-Carlo Method**
  - A Monte-Carlo method is a broad class of computational algorithms that rely on **repeated random sampling** to obtain numerical results.

Monte Carlo Casino in Monaco

Monte-Carlo Learning

ROBOT INTELLIGENCE LAB

- **Monte-Carlo (MC)** Method

Expectation of $f(x)$ where $x \sim P(X = x)$: $\mathbb{E}[f(x)] = \int f(x)\, dP(x)$

MC approximation of $\mathbb{E}[f(x)]$: $\frac{1}{N}\sum_{i=1}^{N} f(x_i)$ where $x_i \sim P(x)$

- Recall the following definitions:
  - The **return** is

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$$

  - The **value function** of π is

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

  - The **MC policy evaluation** is

$$V_\pi(s) \approx \frac{\sum G_0^k}{K}$$

Monte-Carlo Learning

- Here, an **incremental Monte-Carlo** update rule is used:
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

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

Step size

Error correction

- Example (**incremental** averaging):

$$\mu_{k+1} = \frac{1}{k+1}\sum_{j=1}^{k+1} x_j = \frac{1}{k+1}\left(x_{k+1} + \sum_{j=1}^{k} x_j\right)$$

$$= \frac{1}{k+1}(x_{k+1} + k\mu_k) = \mu_k + \frac{1}{k+1}(x_{k+1} - \mu_k)$$

Step size

Error correction

Monte-Carlo Learning

ROBOT INTELLIGENCE LAB

- Sample a trajectory with length T (i.e., $\tau = \{s_t, R_t\}_{t=0}^{T}$)
- Initialize $G_{T+1} = 0$
- From $t = T$ to $t = 0$ (backward)

$$G_t \leftarrow R_{t+1} + \gamma G_{t+1}$$

$$N(s_t) \leftarrow N(s_t) + 1$$

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

Monte-Carlo Policy Iteration

ROBOT INTELLIGENCE LAB

- Sample a trajectory with length T (i.e., $\tau = \{s_t, a_t, R_t\}_{t=0}^{T}$)
- Initialize $G_{T+1} = 0$
- From $t = T$ to $t = 0$ (backward)

$$G_t \leftarrow R_{t+1} + \gamma G_{t+1}$$

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

Monte-Carlo Policy Iteration

ROBOT INTELLIGENCE LAB

```
def update_Q(self):
    """
    Update Q
    """
    Q_old = self.Q # [S x A]
    g = 0
    G = Q_old # [S x A]
    for t in reversed(range(len(self.samples))): # for all samples in a reversed way
        state,action,reward,_ = self.samples[t]
        g = reward + self.gamma*g # g = r + gamma * g
        G[state][action] = g # update G

    # Update Q
    self.Q = Q_old + self.alpha*(G - Q_old) # [S x A]
    # Empty memory
    self.samples = []
```

Exponential Moving Average

ROBOT INTELLIGENCE LAB

- **Incremental Monte-Carlo update** resembles **exponential moving average:**

$$\widehat{V}(s_t) \leftarrow \widehat{V}(s_t) + \frac{1}{N(s_t)}(G_t - \widehat{V}(s_t)) \text{ (incremental MC update)}$$

$$\widehat{V}(s_t) \leftarrow \left(1 - \frac{1}{N(s_t)}\right)\widehat{V}(s_t) + \frac{1}{N(s_t)}G_t \text{ (rearrange of MC update)}$$

$$\widehat{V}(s_t) \leftarrow (1-\alpha)\,\widehat{V}(s_t) + \alpha G_t \text{ (exponential moving average)}$$

- Note that $V(s) = \mathbb{E}[G_t \mid S_t = s]$.

Monte-Carlo Learning Summary

- MC methods learn directly from episodes of experience.
- MC learning is **model-free**.
  - No knowledge of MDP transitions is required.
- MC learning requires **complete** episodes.
  - It may not be suitable for infinite horizon problems.
- MC learning leverages the simplest possible idea:
  - Value is the expected return.

ROBOT INTELLIGENCE LAB

Temporal Difference Learning

Temporal Difference Learning

ROBOT INTELLIGENCE LAB

- In **MC learning**, we use G_t as an estimator of $V_\pi(s_t)$.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

- How about using the current value estimate $\widehat{V}(s_t)$?

- In other words, we approximate the value using the current value estimate:

$$V(s_t) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots] \approx R_{t+1} + \gamma \widehat{V}(s_{t+1})$$

Temporal Difference Learning

ROBOT INTELLIGENCE LAB

- The goal of **temporal difference (TD) learning** is to estimate V_π online from the experiences under the policy π.
- $TD(\gamma)$ updates $\widehat{V}(s_t)$ towards the estimated return $R_{t+1} + \gamma \widehat{V}(S_{t+1})$:

$$\widehat{V}(s_t) \leftarrow \widehat{V}(s_t) + \alpha \left(R_{t+1} + \gamma \widehat{V}(s_{t+1}) - \widehat{V}(s_t) \right)$$

  - $R_{t+1} + \gamma \widehat{V}(s_{t+1})$ is called the **TD target**.
  - $\delta_t = R_{t+1} + \gamma \widehat{V}(s_{t+1}) - \widehat{V}(s_t)$ is called the **TD error**.

- Note that the simplest $TD(0)$ update rule becomes:

$$\widehat{V}(s_t) \leftarrow \widehat{V}(s_t) + \alpha \left(R_{t+1} - \widehat{V}(s_t) \right)$$

Monte-Carlo vs. Temporal-Difference

ROBOT INTELLIGENCE LAB

- Advantages of **TD learning**
  - TD learning can learn before knowing the final outcome.
  - TD learning can learn in an **online** manner (after every step (s, a, s')).
  - TD learning works well in infinite-horizon tasks.

- Disadvantages of **MC learning**
  - MC learning must wait until the end of the episode.
  - MC learning requires the complete sequences.
  - MC learning only works for finite-horizon tasks.

Bias Variance Trade-Off

ROBOT INTELLIGENCE LAB

- The return G_t used in **MC learning** is an **unbiased** estimate of $\widehat{V}(s_t)$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

- TD target $R_{t+1} + \gamma \hat{V}(s_{t+1})$ is a **biased** estimate of $\widehat{V}(s_t)$:

$$\text{TD target} = R_{t+1} + \gamma \hat{V}(s_{t+1})$$

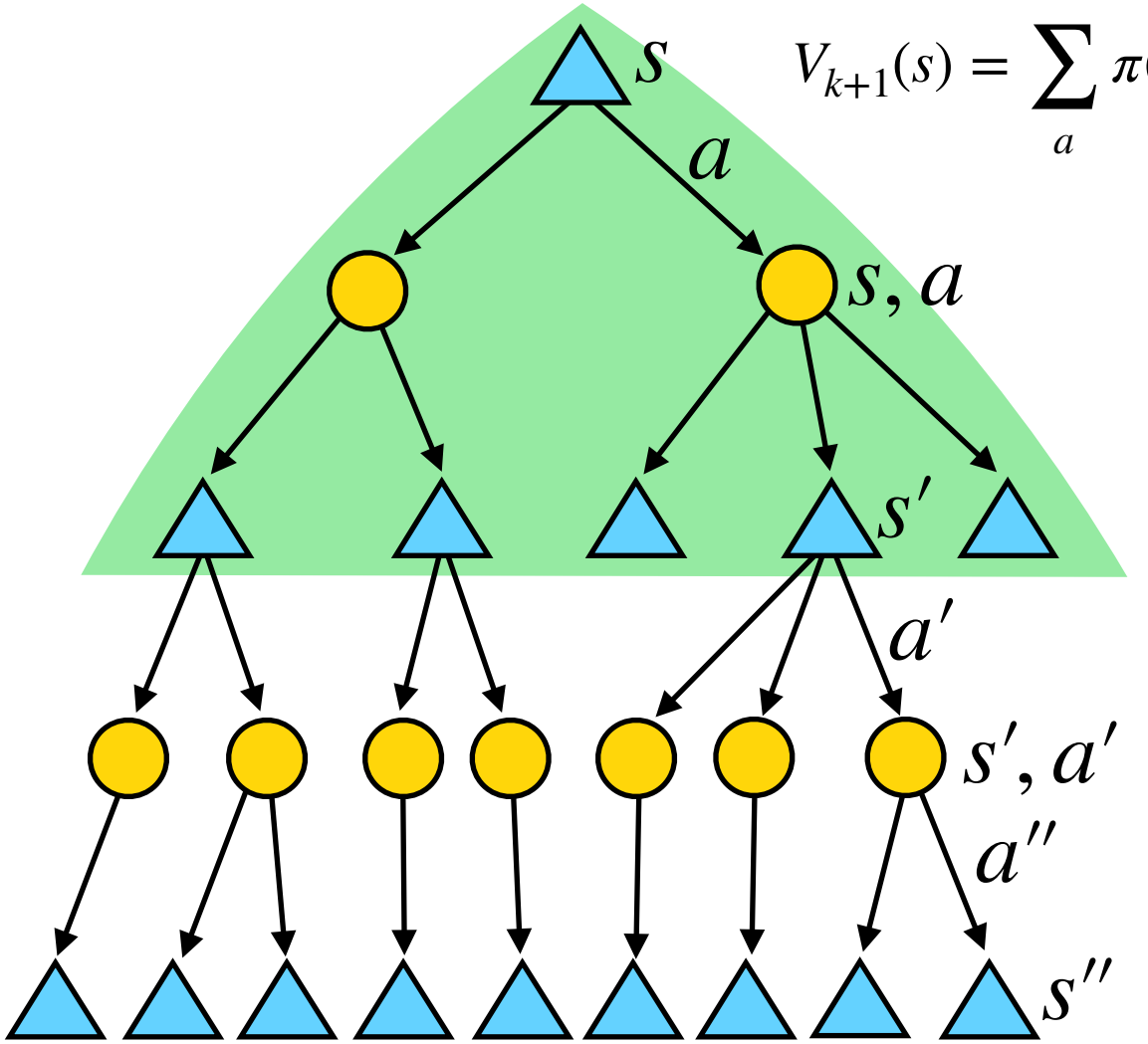
- However, TD target estimate has much **lower variance** than the return.
  - Return is affected by the **randomness of future** actions, transitions, and rewards.
  - TD target depends on a **single random** action, transition, and reward.

ROBOT INTELLIGENCE LAB

Dynamic Programming

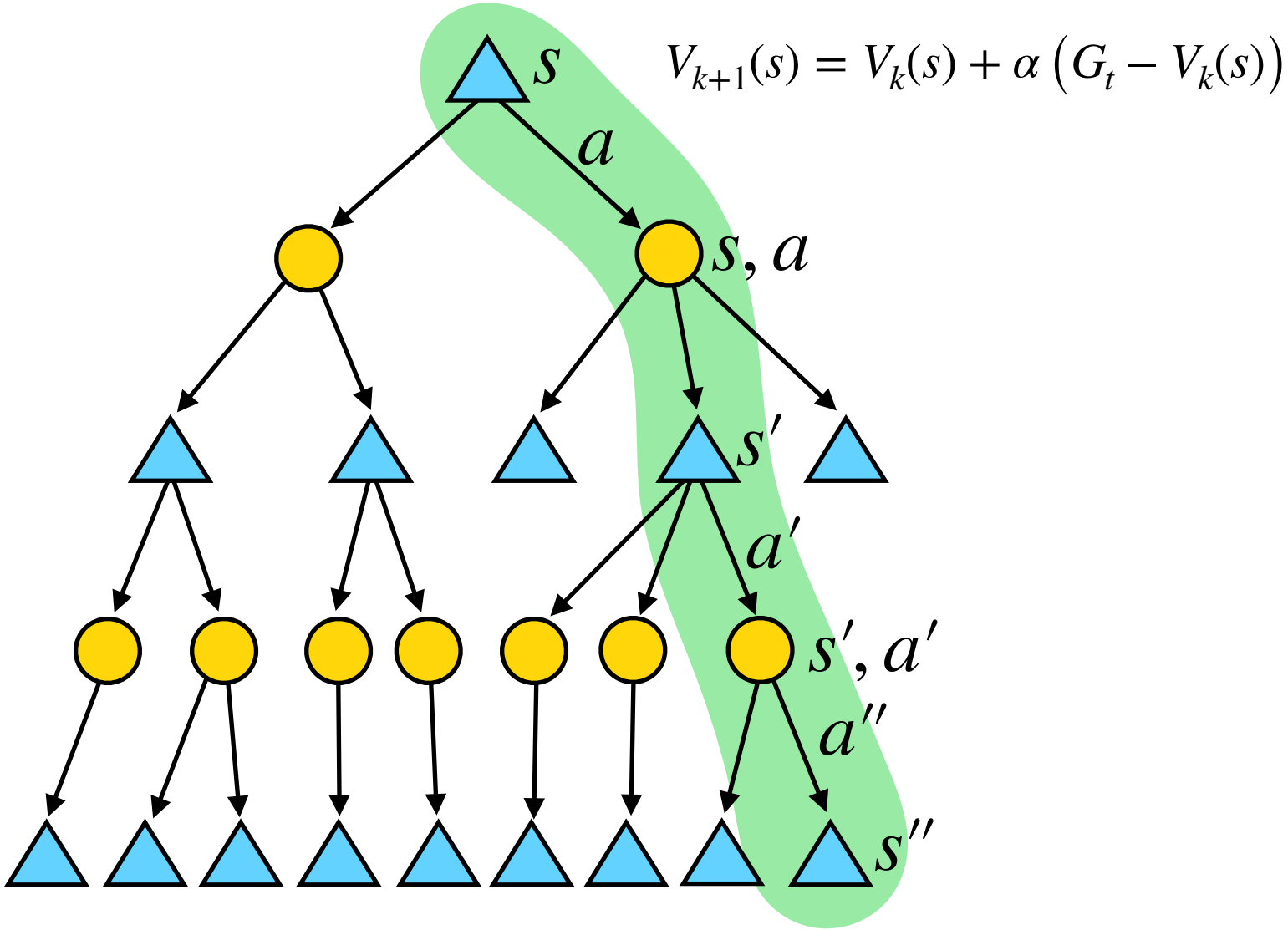
Value Iteration or Policy Iteration

Dynamic Programming



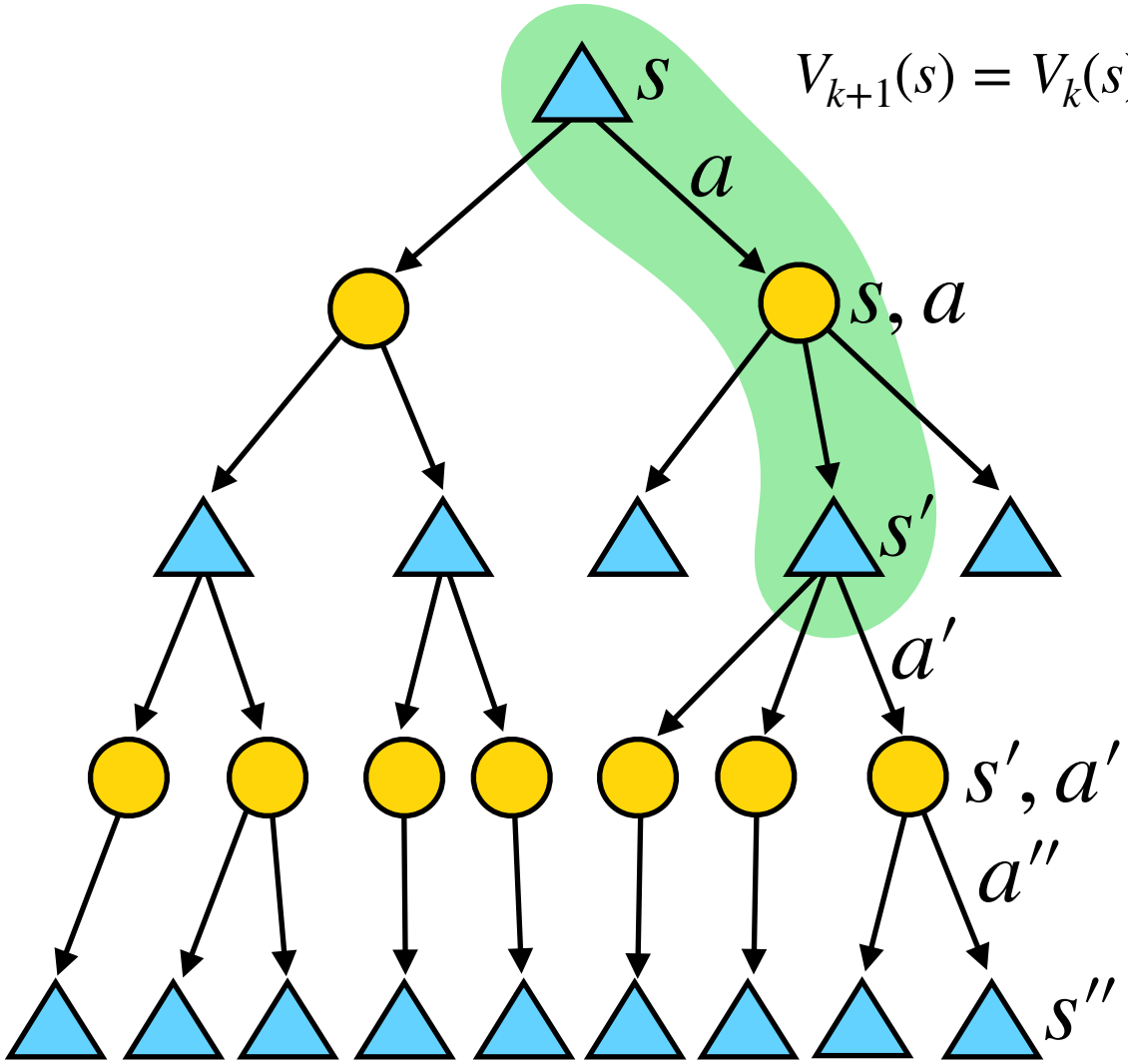
$$V_{k+1}(s) = \sum_{a} \pi(a \mid s) \sum_{s'} \left[r(s, a, s') + \gamma V_k(s') \right] P(s' \mid s, a)$$

Monte-Carlo Learning

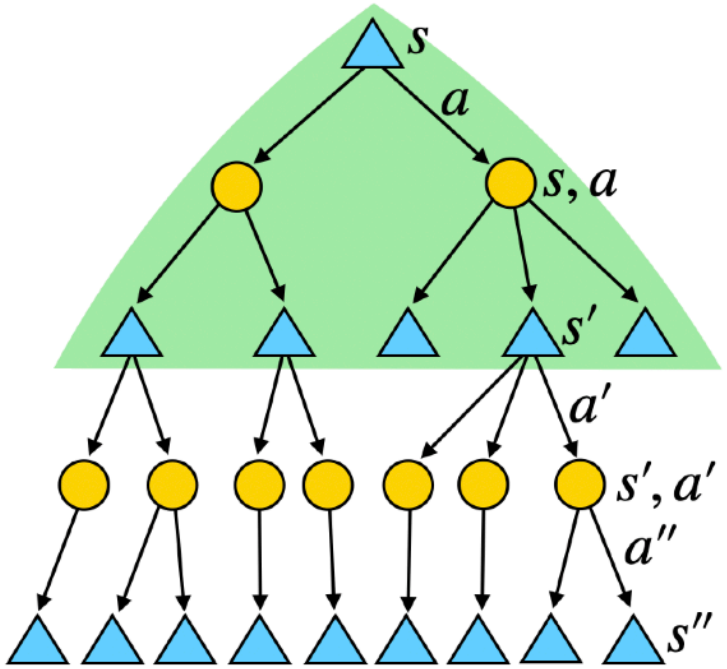


Temporal Difference Learning

$$V_{k+1}(s) = V_k(s) + \alpha \left(r(s, a, s') + \gamma V_k(s') - V_k(s) \right)$$

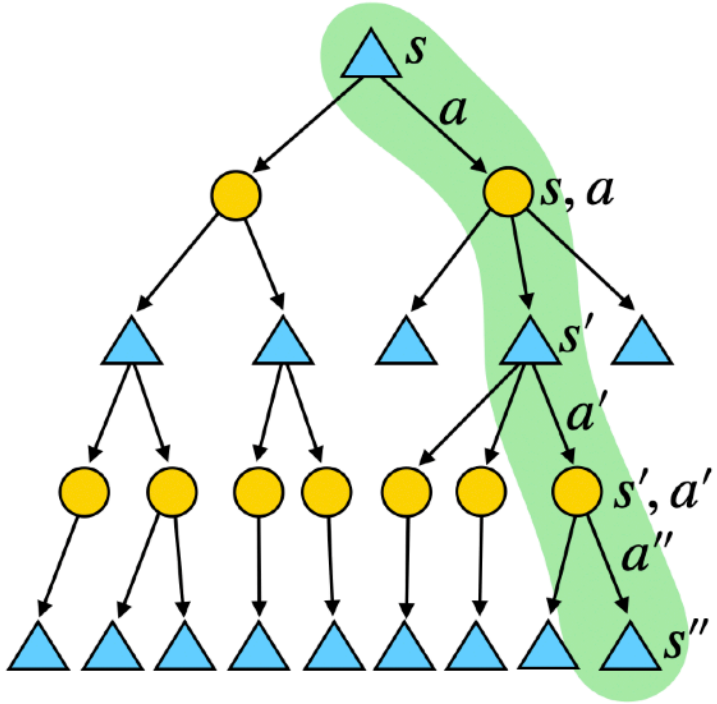


DP vs. MC vs. TD



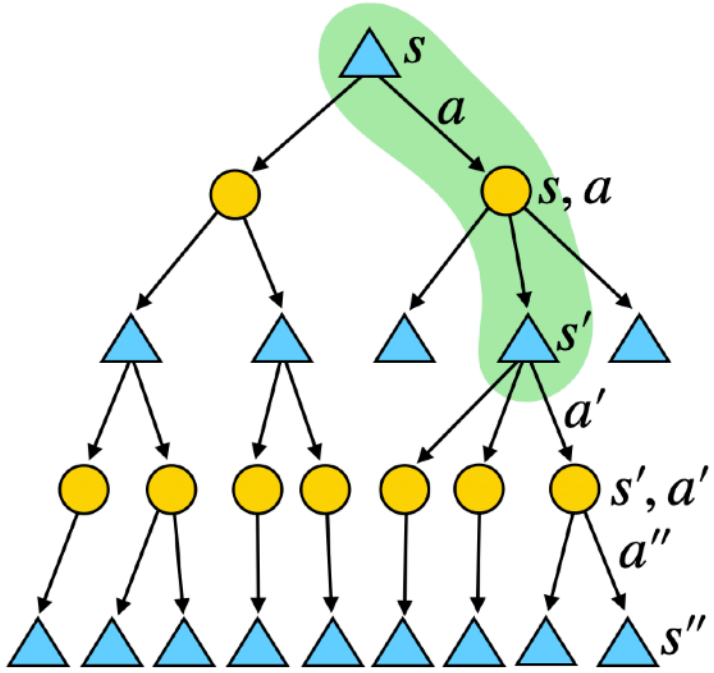
Dynamic Programming

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$



Monte-Carlo Learning

$$V_{k+1}(s) = V_k(s) + \alpha \left(G_t - V_k(s) \right)$$



Temporal Difference Learning

$$V_{k+1}(s) = V_k(s) + \alpha \left(r(s,a,s') + \gamma V_k(s') - V_k(s) \right)$$

ROBOT INTELLIGENCE LAB

N-step TD Learning

N-step Temporal Difference Learning

ROBOT INTELLIGENCE LAB

- Suppose we have N-step return

1-step TD: $G_t^{(1)} = R_{t+1} + \gamma \widehat{V}(s_{t+1})$

2-step TD: $G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 \widehat{V}(s_{t+2})$

…

N-step TD: $G_t^{(N)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{N-1} R_{t+N} + \gamma^N \widehat{V}(s_{t+N})$

Monte-Carlo Estimate | TD Estimate

…

∞-step TD: $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \gamma^4 R_{t+5} + \cdots$ (=MC Learning)

- **N-step temporal-difference** learning

$$V(s) \leftarrow V(s) + \alpha \left(G_t^{(N)} - V(s) \right)$$

N-step TD Target

N-step Temporal Difference Learning

ROBOT INTELLIGENCE LAB

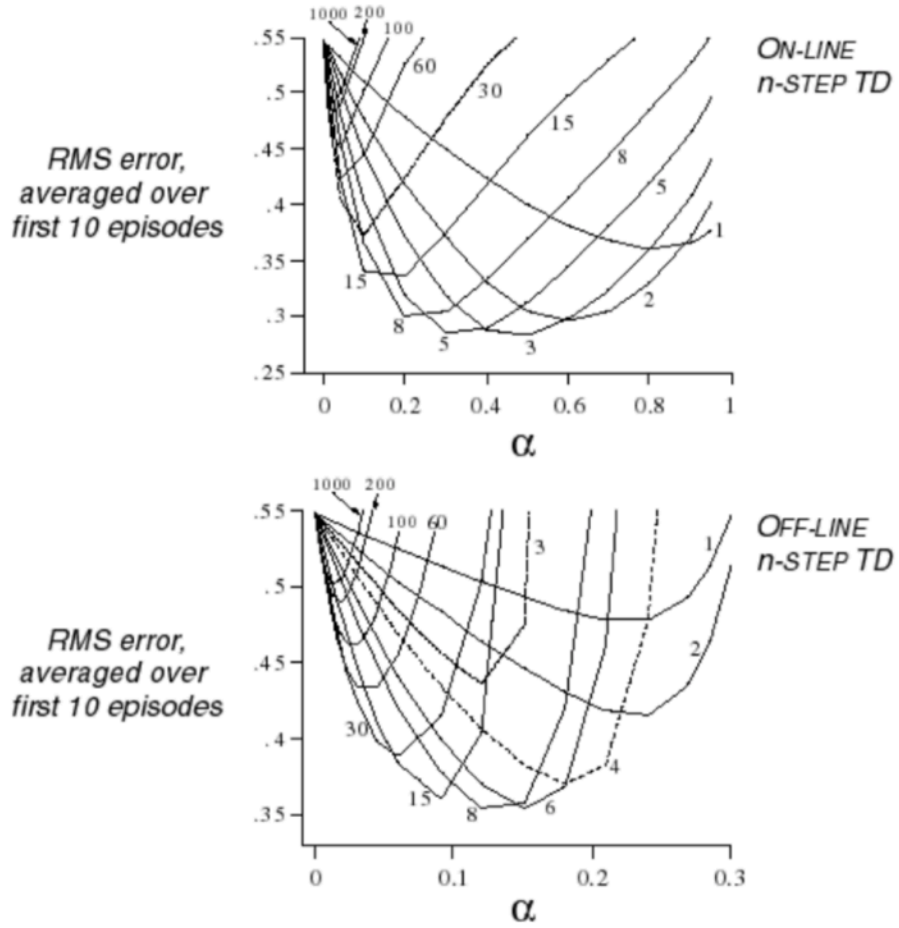
What happens if we increase/decrease N in N-step temporal difference learning?

N-step Temporal Difference Learning

ROBOT INTELLIGENCE LAB

What is the optimal step length N for N-step temporal-difference learning?

N-step Temporal Difference Learning



Recall the **bias-variance trade-off** of Monte-Carlo learning and temporal difference learning.

ROBOT INTELLIGENCE LAB

TD(λ) Learning

TD(λ) Learning

ROBOT INTELLIGENCE LAB

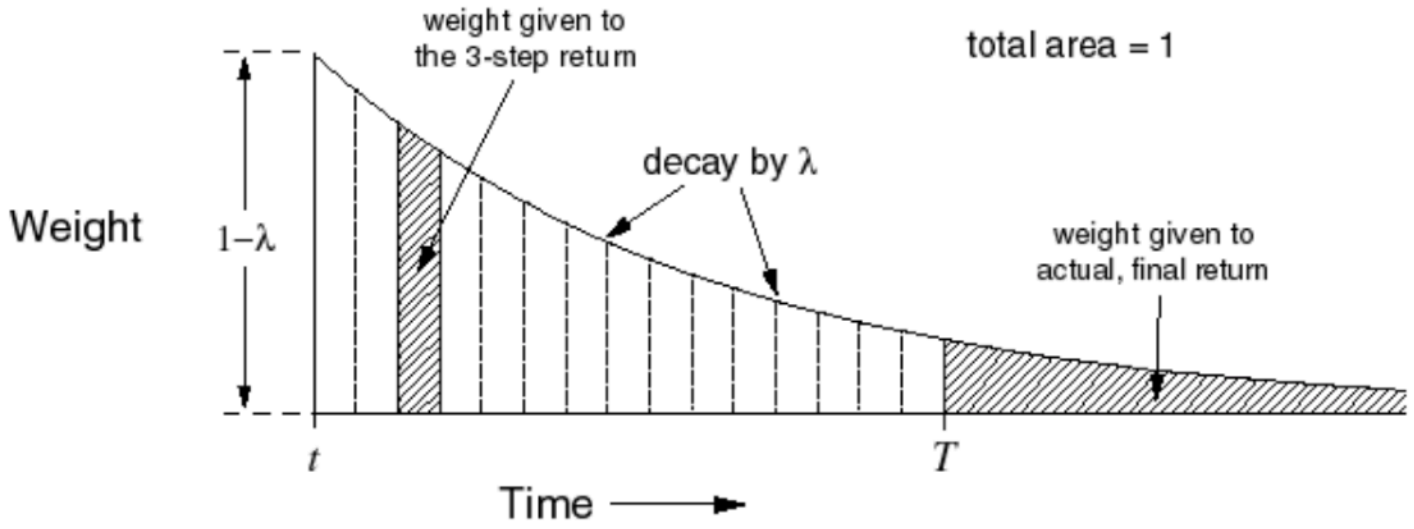
- $TD(\lambda)$ combines information at every time step using λ-return.
  - Like MC learning, it can only be computed from the **complete episodes**.
- λ-Return G_t^λ combines all n-step returns $G_t^{(n)}$.
  - $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{N-1} R_{t+N} + \gamma^N \widehat{V}(s_{t+N})$
  - How?
  - Using the weight $(1-\lambda)\lambda^{n-1}$ for each $G_t^{(n)}$

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

- Then, the update of the value function becomes

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

$TD(\lambda)$ Learning



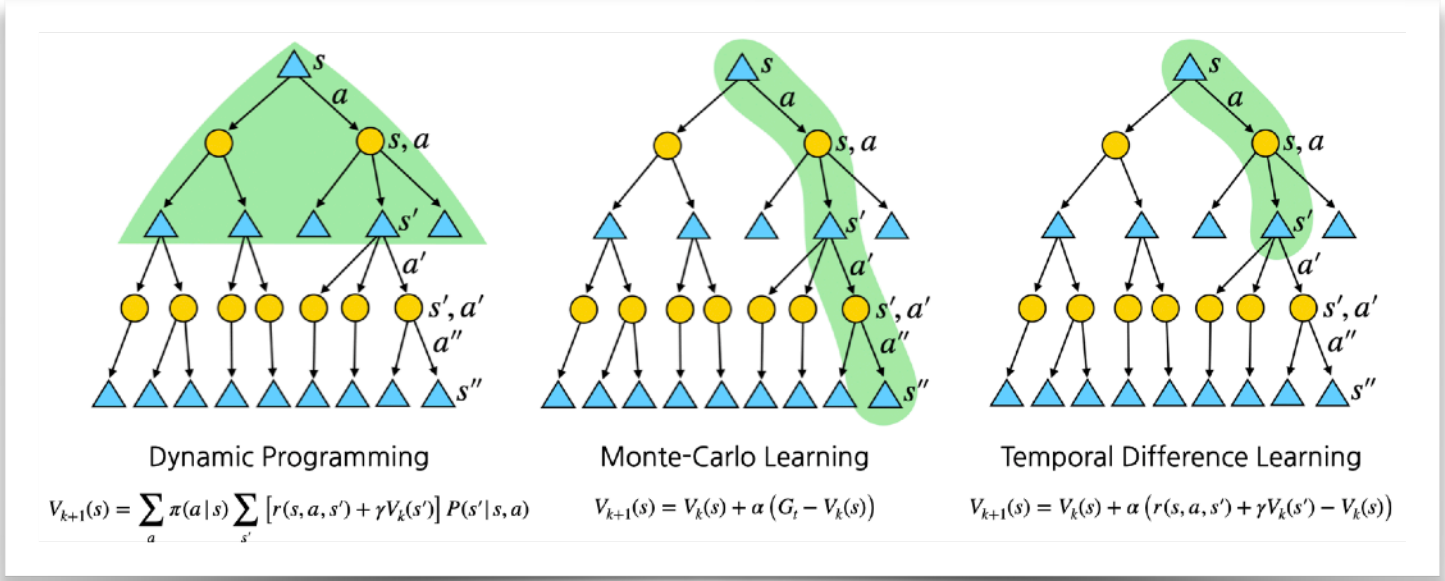
$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{N-1}R_{t+N} + \gamma^N \widehat{V}(s_{t+N})$$

ROBOT INTELLIGENCE LAB

SARSA

SARSA



- So far, we have seen **MC learning**, **TD learning**, and **TD(λ) learning**.
- For such methods, unlike dynamic programming, we need samples (episodes) to estimate the value function.
  - Note that any sequences collected from a random policy will work.
  - However, it may be too **ineffective** to use in practice.
  - How about using an ϵ-greedy policy?

Policy Iteration (revisited)

ROBOT INTELLIGENCE LAB

- **Step 1: Policy evaluation**: calculate the value function for a fixed policy until convergence.
  - Start from a random initial V_0.
  - Iterate until value converges:

$$V_{k+1}(s) = \sum_a \pi_i(a \mid s) \sum_{s'} [r(s, a, s') + \gamma V_k(s')] P(s' \mid s, a)$$

- **Step 2: Policy improvement**: update the policy using one-step lookahead using the converged value function.
  - One-step lookahead:

$$Q_{\pi_k}(s, a) = \sum_{s'} [r(s, a, s') + \gamma V_{\pi_i}(s')] P(s' \mid s, a)$$

$$\pi_{i+1}(a \mid s) = 1 \text{ for } a = \arg\max_{a'} Q_{\pi_i}(s, a')$$

Policy Iteration (revisited)

ROBOT INTELLIGENCE LAB

- **Step 1: Policy evaluation**: calculate the value function for a fixed policy until convergence.
  - Start from a random initial V_0.
  - Iterate until value converges:

State Transition Probability

$$V_{k+1}(s) = \sum_a \pi_i(a \mid s) \sum_{s'} [r(s,a,s') + \gamma V_k(s')] P(s' \mid s, a)$$

- **Step 2: Policy improvement**: update the policy using one-step lookahead using the converged value function.
  - One-step lookahead:

$$Q_{\pi_k}(s,a) = \sum_{s'} [r(s,a,s') + \gamma V_{\pi_i}(s')] P(s' \mid s, a)$$

$$\pi_{i+1}(a \mid s) = 1 \text{ for } a = \arg\max_{a'} Q_{\pi_i}(s,a')$$

Policy Iteration (revisited)

ROBOT INTELLIGENCE LAB

- **Step 1: Policy evaluation**
  - Instead of finding $V(s)$, find $Q(s,a)$.
  - Start from a random initial Q_0.
  - Iterate until value converges:

$$Q_{k+1}(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma \sum_{a'} Q_k(s',a')\pi(a'|s') \right] \boxed{P(s'|s,a)}$$

We still need to handle this.

- **Step 2: Policy improvement**
  - Now, we do not need the policy improvement step as we get the policy for free.

$$\pi_{i+1}(a|s) = 1 \text{ for } a = \arg\max_{a'} Q_{\pi_i}(s,a')$$

Model-Free Policy Iteration

ROBOT INTELLIGENCE LAB

- **Step 1: Policy evaluation**
  - Estimate $\widehat{Q}(s,a)$ from samples using a model-free method (e.g., **MC**, **TD**, or **TD(λ)**).

$$\widehat{Q}(s_t,a_t) \leftarrow \widehat{Q}(s_t,a_t) + \alpha\left(G_t - \widehat{Q}(s_t,a_t)\right) \text{ (MC)}$$

$$\widehat{Q}(s_t,a_t) \leftarrow \widehat{Q}(s_t,a_t) + \alpha\left(R_{t+1} + \gamma\widehat{Q}(s_{t+1},a_{t+1}) - \widehat{Q}(s_t,a_t)\right) \text{ (TD)}$$

$$\widehat{Q}(s_t,a_t) \leftarrow \widehat{Q}(s_t,a_t) + \alpha\left(G_t^{\lambda} - \widehat{Q}(s_t,a_t)\right) \text{ (TD}(\lambda)\text{)}$$

- **Step 2: Policy improvement**
  - For the sake of better exploration, we use an ϵ-greedy policy.
    - With probability $1-\epsilon$, choose the greedy action $a = \arg\max_{a'} Q_{\pi_i}(s,a')$.
    - With probability ϵ, choose a random action.

Model-Free Policy Iteration

ROBOT INTELLIGENCE LAB

- **Step 1: Policy evaluation**
  - Estimate $\widehat{Q}(s,a)$ from samples using (S, A, R, S', A')

$$\widehat{Q}(s_t,a_t) \leftarrow \widehat{Q}(s_t,a_t) + \alpha\left(R_{t+1} + \gamma\,\widehat{Q}(s_{t+1},a_{t+1}) - \widehat{Q}(s_t,a_t)\right) \text{ (TD)}$$

- **Step 2: Policy improvement**
  - For the sake of better exploration, we use an ϵ-greedy policy.
    - With probability $1-\epsilon$, choose the greedy action $a = \arg\max_{a'} Q_{\pi_i}(s,a')$.
    - With probability ϵ, choose a random action.

SARSA

ROBOT INTELLIGENCE LAB

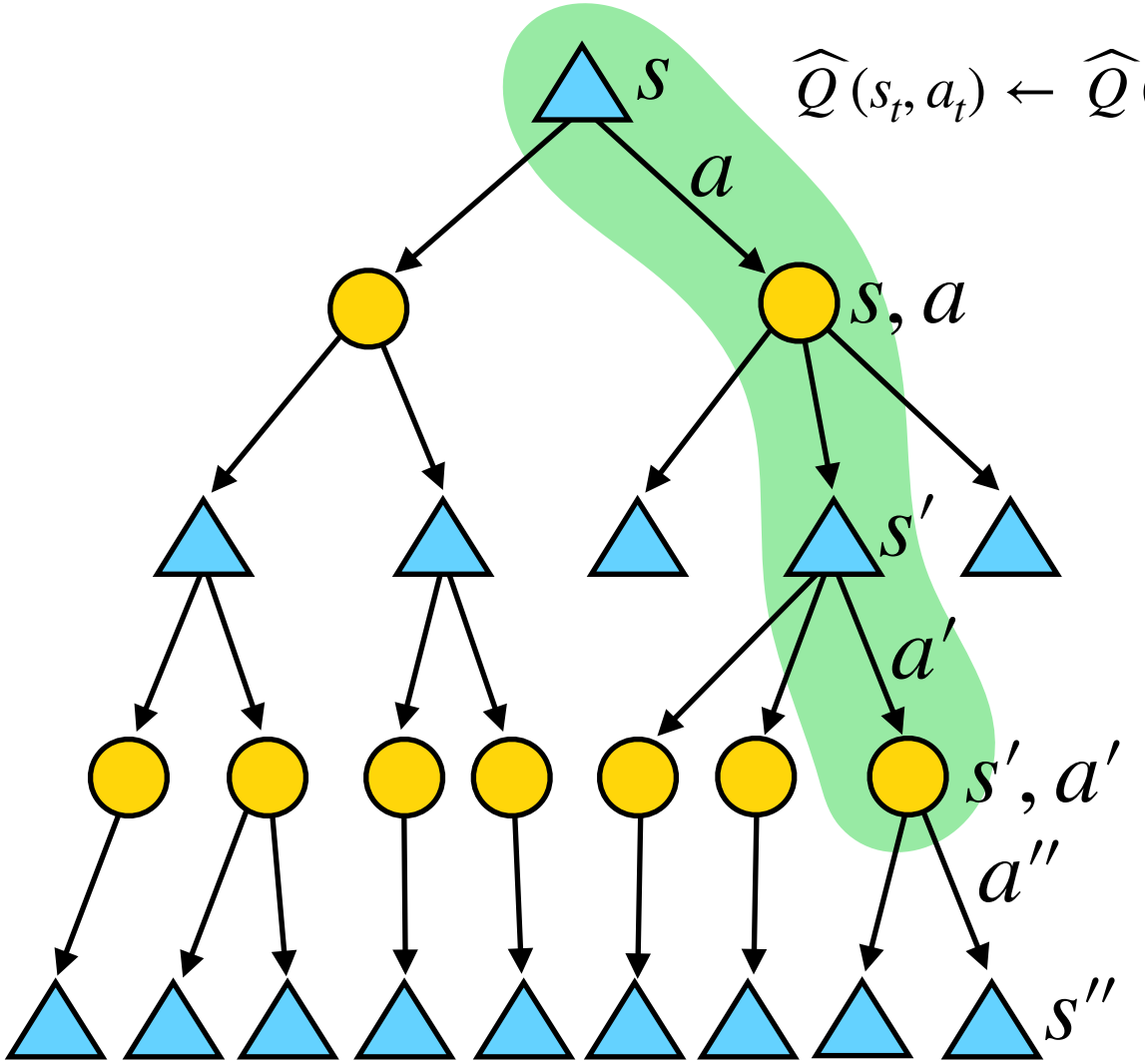
- Initialize $Q(s, a)$
- Repeat (for each episodes)
  - Sample an initial state s_0.
  - Sample a_0 from an ϵ-greedy policy π.
  - Repeat (for each time step t)
    - Get reward r_{t+1} and next state s_{t+1}.
    - Sample a_{t+1} from the ϵ-greedy policy π.
    - Update $\widehat{Q}(s, a)$ using $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$
      - $\widehat{Q}(s_t, a_t) \leftarrow \widehat{Q}(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \widehat{Q}(s_{t+1}, a_{t+1}) - \widehat{Q}(s_t, a_t) \right)$

SARSA

ROBOT INTELLIGENCE LAB

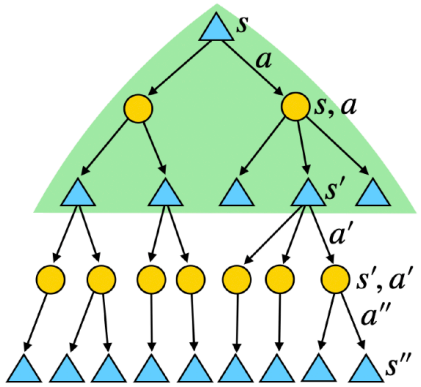
```
def update_Q(self,state,action,reward,state_prime,action_prime,done):
    """
    Update Q value using TD learning
    """
    Q_old = self.Q[state][action]
    if done:
        td_target = reward # for the last step, Q = reward
    else:
        td_target = reward + self.gamma*self.Q[state_prime][action_prime]
    td_error = td_target - Q_old
    # Update Q value
    self.Q[state,action] = Q_old + self.alpha*td_error
```

SARSA



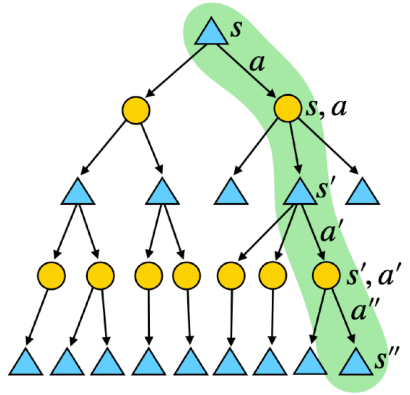
$$\widehat{Q}(s_t, a_t) \leftarrow \widehat{Q}(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \widehat{Q}(s_{t+1}, a_{t+1}) - \widehat{Q}(s_t, a_t) \right)$$

DP vs. MC vs. TD vs. SARSA



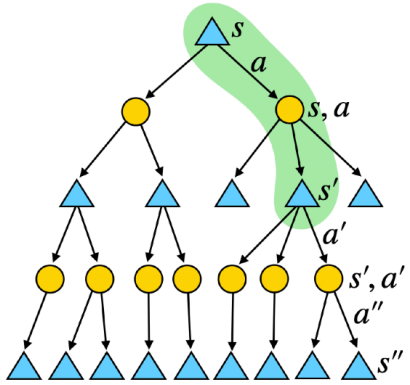
Dynamic Programming

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$



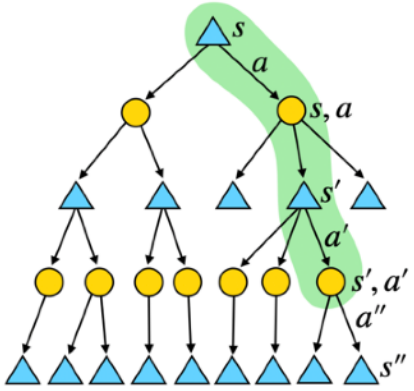
Monte-Carlo Learning

$$V_{k+1}(s) = V_k(s) + \alpha \left(G_t - V_k(s) \right)$$



Temporal Difference Learning

$$V_{k+1}(s) = V_k(s) + \alpha \left(r(s,a,s') + \gamma V_k(s') - V_k(s) \right)$$



SARSA

$$\widehat{Q}(s_t, a_t) \leftarrow \widehat{Q}(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \widehat{Q}(s_{t+1}, a_{t+1}) - \widehat{Q}(s_t, a_t) \right)$$

ROBOT INTELLIGENCE LAB

On-Policy vs. Off-Policy

On-Policy vs. Off-Policy Learning

ROBOT INTELLIGENCE LAB

- On-Policy Learning (MC, SARSA)
  - Learn the value of the policy π using the episodes sampled from π.
- Off-Policy Learning
  - Learn the value of the policy π using the episodes sampled from **any arbitrary** μ.

- Why is off-policy learning important?
  - Off-policy methods enable learning from observing humans or other agents.
  - Re-use experiences generated from old polices (**experience replay**).
  - Learn the optimal policy while following other exploratory policy.

Importance Sampling for Off-Policy Learning

ROBOT INTELLIGENCE LAB

- Importance Sampling
  - $\mathbb{E}_P[X] \approx \frac{\sum x_i}{N}, x_i \sim P(X)$
  - $\mathbb{E}_P[X] \approx \frac{\sum \frac{P(x_i)}{Q(x_i)} x_i}{N}, x_i \sim Q(X)$
- Off-Policy Monte-Carlo
  - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots + \gamma^T R_{t+1+T}$
  - When actions are being sampled from μ instead of π:
    - $G_t^{\pi/\mu} = \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)} \frac{\pi(a_{t+1} \mid s_{t+1})}{\mu(a_{t+1} \mid s_{t+1})} \cdots \frac{\pi(a_{t+1+T} \mid s_{t+1+T})}{\mu(a_{t+1+T} \mid s_{t+1+T})} G_t$ where $\mu()$ is the policy for sampling and $\pi()$ is the true policy to update.
  - Importance sample can often dramatically increase the variance.

Importance Sampling for Off-Policy Learning

ROBOT INTELLIGENCE LAB

- Off-Policy Temporal-Difference
  - TD target = $R_{t+1} + \gamma V(S_{t+1})$
  - Weighted TD target: multiply importance sampling corrections along the whole episodes

$$V(s_t) \leftarrow V(s_t) + \alpha \left(\frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)} \left[R_{t+1} + \gamma V(s_{t+1}) \right] - V(s_t) \right)$$

  - It has much lower variance than Monte-Carlo importance sampling.

ROBOT INTELLIGENCE LAB

Q-Learning

Q-Learning

ROBOT INTELLIGENCE LAB

- Basic concepts of **Q-Learning**
  - It is a **model-free value iteration**.

$$V_{k+1}(s) = \max_a \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s'|s,a)$$

  - Q-values are more useful in terms of getting π.

$$Q_{k+1}(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right] P(s'|s,a)$$

- But we still need the (transition) model $P(s'|s,a)$.
- Suppose that we are using **any behavior policy μ** to get a at any state s, and proceed to the next state s' following $P(s'|s,a)$,

$$Q_{k+1}(s,a) \approx r(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

Q-Learning

ROBOT INTELLIGENCE LAB

- It can be regarded as a **model-free value iteration**.
- **Q-Learning**
  - For each time step

$$\widehat{Q}(s,a) \leftarrow \widehat{Q}(s,a) + \alpha\left(r(s,a,s') + \gamma \max_{a'} \widehat{Q}(s',a') - \widehat{Q}(s,a)\right)$$

SARSA: $\widehat{Q}(s_t,a_t) \leftarrow \widehat{Q}(s_t,a_t) + \alpha\left(R_{t+1} + \gamma \widehat{Q}(s_{t+1},a_{t+1}) - \widehat{Q}(s_t,a_t)\right)$ (TD)

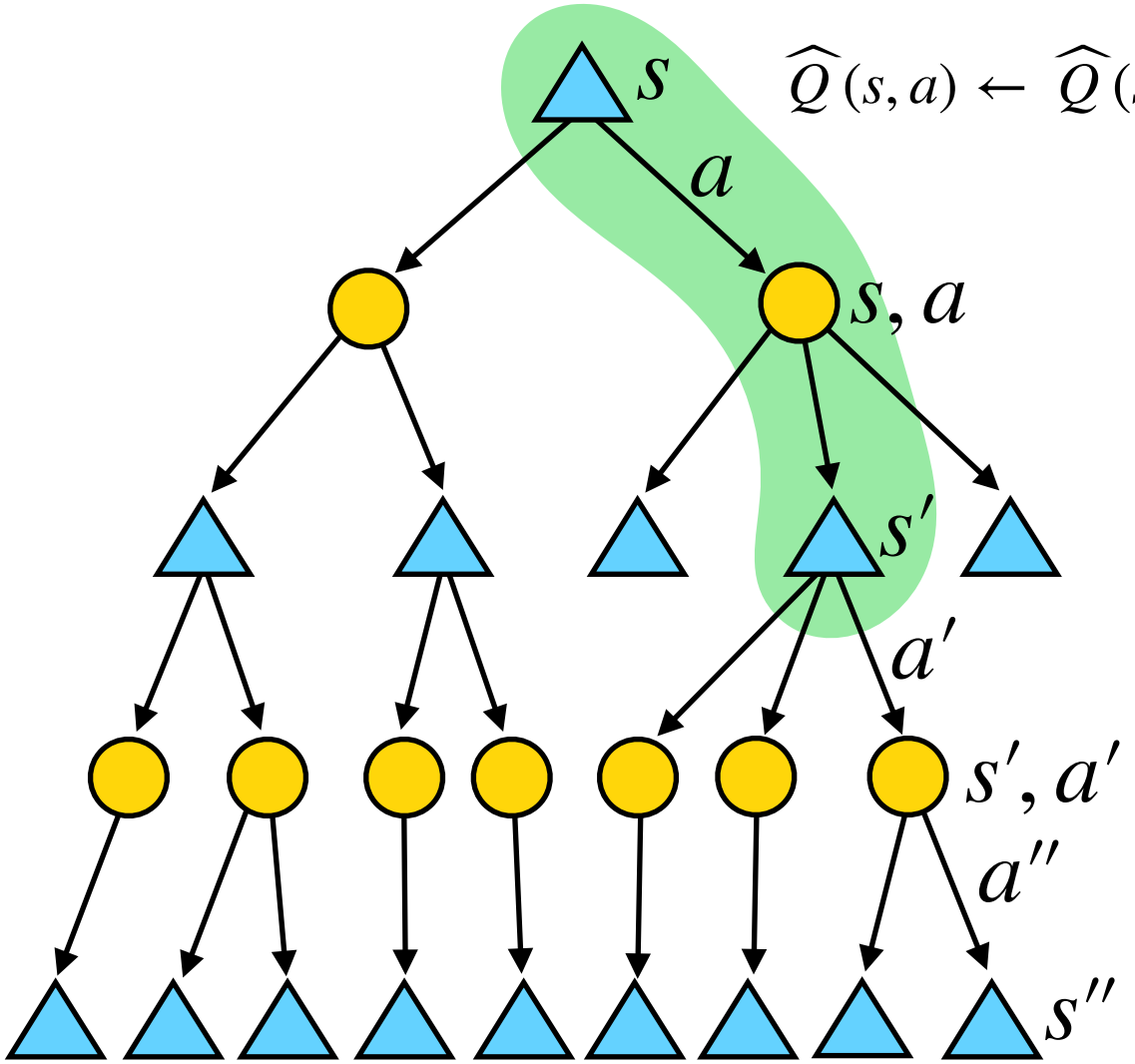
- Note that $\max_{a'} \widehat{Q}(s',a')$ does not require the next action (compared to **SARSA**), hence we only need (s,a,s') for **Q-Learning**.
- We can use any **arbitrary behavior policy** to sample episodes as long as $s' \sim P(s'|s,a)$ which enables to utilize experience replays.

Q-Learning

ROBOT INTELLIGENCE LAB

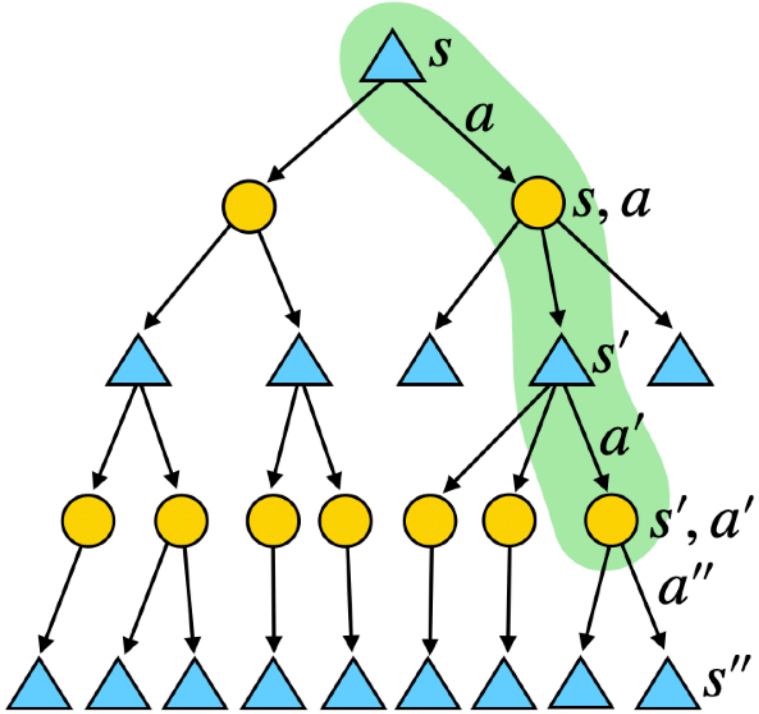
```
def update_value(self,state,action,reward,state_prime,done):
    """
    Update value
    """
    Q_old = self.Q[state][action]
    # TD target
    if done:
        td_target = reward
    else:
        td_target = reward + self.gamma*np.max(self.Q[state_prime])
    td_error = td_target - Q_old # TD error
    self.Q[state,action] = Q_old + self.alpha*td_error # update Q
```

Q-Learning



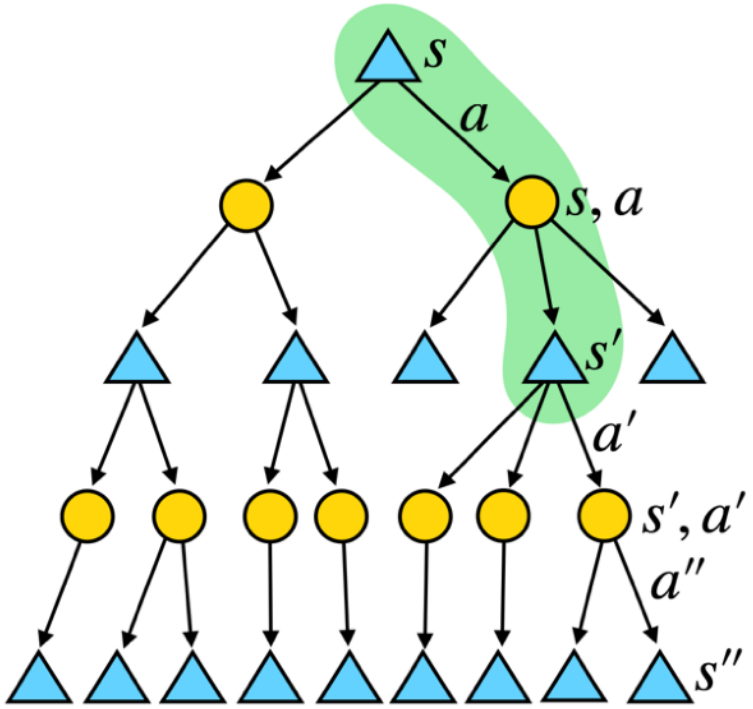
$$\widehat{Q}(s,a) \leftarrow \widehat{Q}(s,a) + \alpha\left(r(s,a,s') + \gamma \max_{a'} \widehat{Q}(s',a') - \widehat{Q}(s,a)\right)$$

SARSA vs. Q-Learning



SARSA

$$\widehat{Q}(s_t, a_t) \leftarrow \widehat{Q}(s_t, a_t) + \alpha\left(r_{t+1} + \gamma\,\widehat{Q}(s_{t+1}, a_{t+1}) - \widehat{Q}(s_t, a_t)\right)$$



Q-Learning

$$\widehat{Q}(s, a) \leftarrow \widehat{Q}(s, a) + \alpha\left(r(s, a, s') + \gamma \max_{a'} \widehat{Q}(s', a') - \widehat{Q}(s, a)\right)$$

ROBOT INTELLIGENCE LAB

DQN

"Human-level control through deep reinforcement learning," Nature, 2015

Deep Q Network (DQN)

ROBOT INTELLIGENCE LAB

Finally, we have arrived to the **DQN**.

Deep Q Network (DQN)

ROBOT INTELLIGENCE LAB

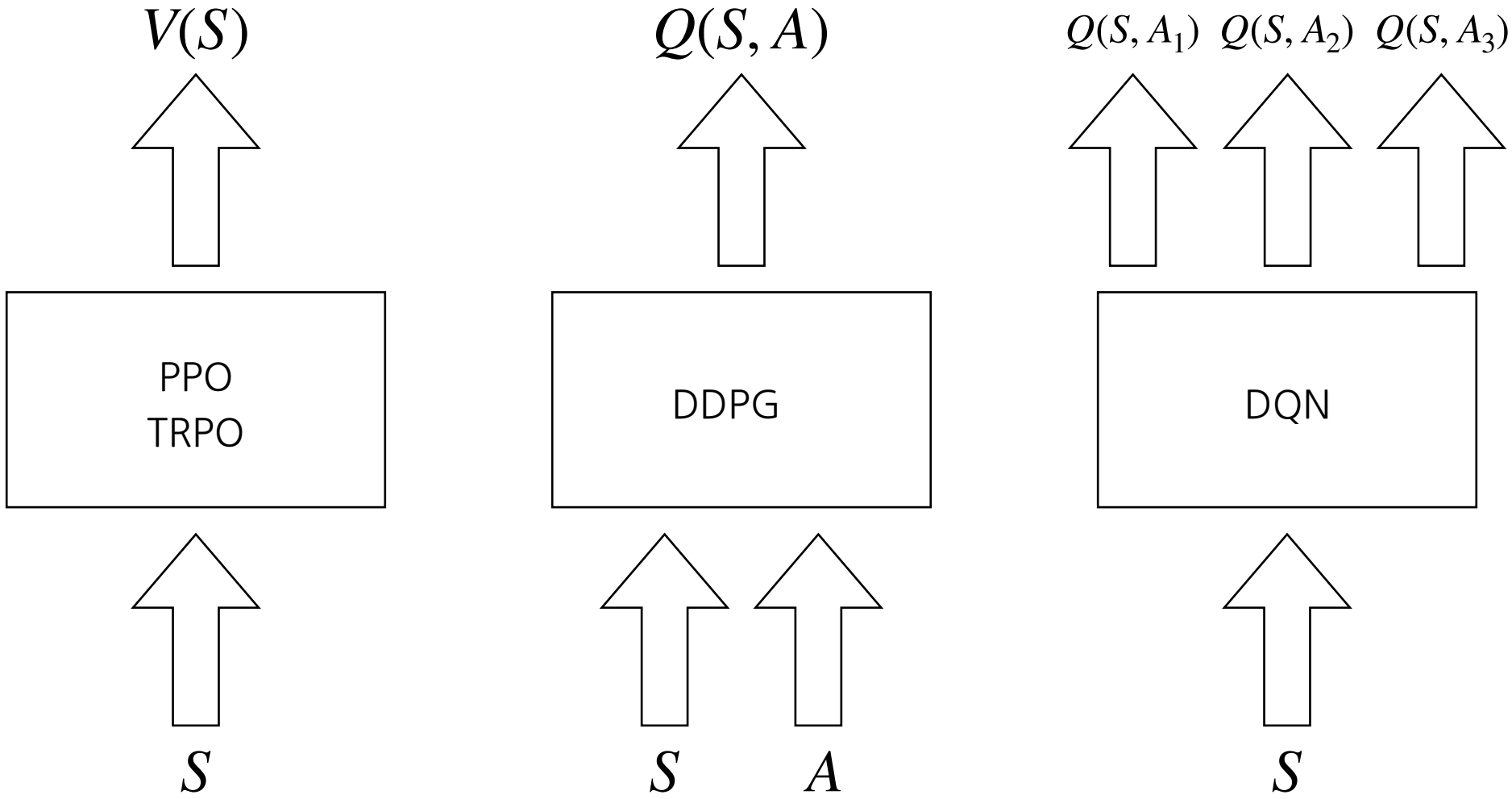
- Recall the original Q-Learning

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \left(\underbrace{r(s,a,s') + \gamma \max_{a'} Q_k(s',a')}_{\text{Target}} - \underbrace{Q_k(s,a)}_{\text{Prediction}} \right)$$

- From the Q-learning objective, we can derive the following loss function:

$$L(\theta) = \sum_i \left(\underbrace{r_i + \gamma \max_{a'} Q(s_i', a'; \theta)}_{\text{Target}} - \underbrace{Q(s_i, a_i; \theta)}_{\text{Prediction}} \right)^2$$

Deep Q Network (DQN)



Deep Q Network (DQN)

ROBOT INTELLIGENCE LAB

- (Stable) Update Rule

$$L(\theta) = \sum_i \left(r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

- Delayed update
  - For numerical stability, slowly update the target network
  - θ^-: previous parameter (for the Q estimation)
  - θ: current parameter to update
- Other tricks
  - Gradient clipping
  - Input normalization

Deep Q Network (DQN)

ROBOT INTELLIGENCE LAB

- (Stable) Update Rule

$$L(\theta) = \sum_i \left(r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

- Delayed update
  - For numerical stability,
  - θ^-: previous parameter
  - θ: current parameter to
- Other tricks
  - Gradient clipping
  - Input normalization

```
def update_main_network(self, o_batch, a_batch, r_batch, o1_batch, d_batch):
    o1_q = self.target_network(o1_batch)
    max_o1_q = o1_q.max(1)[0].detach().numpy()
    d_batch = d_batch.astype(int)
    expected_q = r_batch + self.gamma*max_o1_q*(1.0-d_batch)
    expected_q = expected_q.astype(np.float64) # R + gamma*max(Q)
    expected_q = torch.from_numpy(expected_q)
    main_q = self.main_network(o_batch).max(1)[0]
    loss = F.smooth_l1_loss(main_q.float(), expected_q.float())

    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()

    return loss
```

Experience Replay

ROBOT INTELLIGENCE LAB

- Experience replay stores transitions (s, a, r, s') to memory.
- To resolve the correlated data problem.
  - Most of machine learning methods assumes that the training data are collected from iid distributions.
- Why is it possible?
  - Q-learning is off-policy learning.
  - Hence, it does not care about the sampling policy.

Double Deep Q Network (DDQN)

ROBOT INTELLIGENCE LAB

- Recall the original DQN loss function

$$L(\theta) = \sum_i \left(r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

- Note that the function for selecting the current optimal action and evaluating the optimal action is the same ,$Q(s, a; \theta^-)$.
- It often leads to **over-optimism** of the Q function.
- To resolve this issue,

$$L(\theta) = \sum_i \left(r_i + \gamma Q(s_i', \arg\max_{a'} Q(s_i', a'; \theta); \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

Target Prediction

Prioritized Experience Replay (PER)

ROBOT INTELLIGENCE LAB

- The intuition is to give **more emphasis** on unfitted data.
- Implementation is simple:
  - Sample k transitions from the experience replay from $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$

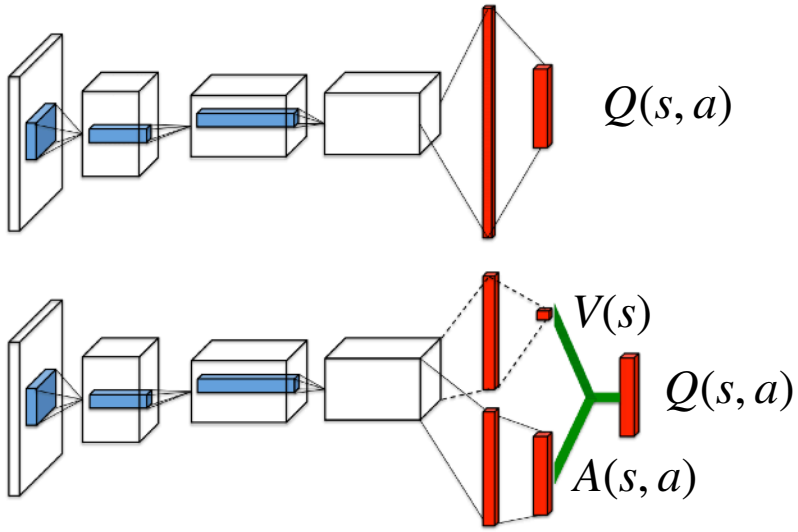
$$\text{where } p_i = \left| r_i + \gamma Q(s_i', \arg\max_{a'} Q(s_i', a'; \theta); \theta^-) - Q(s_i, a_i; \theta) \right|.$$

  - The (weighted) update rule is:

$$L(\theta) = \sum_i w_i \left(r_i + \gamma Q(s_i', \arg\max_{a'} Q(s_i', a'; \theta); \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

$$\text{where } w_i = \left(\frac{1}{n \cdot p_i} \right)^\beta / \max(w).$$

Dueling Architecture



- The main idea is to separate $Q(s,a)$ into $V(s)$ and $A(s,a)$ which is the advantage.
- Advantage function
  - $Q_\pi(s,a) = V_\pi(s) + A_\pi(s,a)$
  - $\arg\max_{a'} A(s,a') = \arg\max_{a'} Q(s,a')$

ROBOT INTELLIGENCE LAB

Policy Gradient Theorem

Limitations of Previous Methods

ROBOT INTELLIGENCE LAB

What happens if we want to model **continuous** action space?

Policy Optimization

ROBOT INTELLIGENCE LAB

- Policy gradient methods cast reinforcement learning into an optimization problem.

Policy Optimization

ROBOT INTELLIGENCE LAB

- Policy gradient methods cast reinforcement learning into an optimization problem.

- Find θ that maximizes the return:

$$\eta(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \,|\, \pi_\theta\right]$$

Policy Optimization

ROBOT INTELLIGENCE LAB

- Policy gradient methods cast reinforcement learning into an optimization problem.

- Find θ that maximizes the return:

$$\eta(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \,|\, \pi_\theta\right]$$

- We update the parameters of the **policy** function by computing the **gradient** of the parameters of the objective function:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \eta(\pi_\theta)$$

How to compute the gradients

ROBOT INTELLIGENCE LAB

$$\nabla_\theta \eta(\pi_\theta) = \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \,|\, \pi_\theta\right]$$

- Policy Gradient Theorem:

$$\nabla_\theta \eta(\pi_\theta) = \frac{1}{(1-\gamma)} \sum_s \rho_{\pi_\theta} \sum_a \nabla_\theta \pi_\theta(a \,|\, s) Q^{\pi_\theta}(s,a)$$

$$\nabla_\theta \eta(\pi_\theta) \approx \nabla_\theta \log \pi_\theta(a_t \,|\, s_t) Q_{\pi_\theta}(s_t, a_t)$$

- Note that we only require the gradient of $\pi_\theta(\cdot)$ not $Q^{\pi_\theta}(\cdot)$!

R. Sutton et al, "Policy Gradient Methods for Reinforcement Learning with Function Approximation", NeurIPS, 2000

State Visitation

ROBOT INTELLIGENCE LAB

- Stationary distribution of the state given $\pi_\theta(\cdot)$

$$\rho_{\pi_\theta}(s) = (1-\gamma)\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \mathbb{I}_{(St=s)}\right] = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_{\pi_\theta}(S_t = s)$$

State Visitation

ROBOT INTELLIGENCE LAB

- Stationary distribution of the state given $\pi_\theta(\cdot)$

$$\rho_{\pi_\theta}(s) = (1-\gamma)\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \mathbb{I}_{(St=s)}\right] = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_{\pi_\theta}(S_t = s)$$

- $\rho_{\pi\theta}(s)$ is a probability mass function

$$\sum_s \rho_{\pi_\theta}(s) = (1-\gamma)\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \sum_s \mathbb{I}_{(S_t=s)}\right] = (1-\gamma)\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\right] = (1-\gamma)\frac{1}{1-\gamma} = 1$$

Proof of Policy Gradient (1/10)

ROBOT INTELLIGENCE LAB

- Return $\eta(\pi_\theta)$ of a policy $\pi_\theta(\cdot)$ and its gradient:

$$\eta(\pi_\theta) = \sum_s d(s) V_{\pi_\theta}(s)$$

$$\nabla_\theta \eta(\pi_\theta) = \sum_s d(s) \nabla_\theta V_{\pi_\theta}(s)$$

Proof of Policy Gradient (1/10)

ROBOT INTELLIGENCE LAB

- Return $\eta(\pi_\theta)$ of a policy $\pi_\theta(\cdot)$ and its gradient:

$$\eta(\pi_\theta) = \sum_s d(s) V_{\pi_\theta}(s)$$

$$\nabla_\theta \eta(\pi_\theta) = \sum_s d(s) \nabla_\theta V_{\pi_\theta}(s)$$

- Let's select an arbitrary state, say s_1, and compute $\nabla_\theta V_{\pi_\theta}(s_1)$:

$$\nabla_\theta V_{\pi_\theta}(s_1) = \nabla_\theta \sum_a \pi_\theta(a \mid s_1) Q_{\pi_\theta}(s_1, a)$$

$$\nabla_\theta V_{\pi_\theta}(s_1) = \sum_a \nabla_\theta \pi_\theta(a \mid s_1) Q_{\pi_\theta}(s_1, a) + \pi_\theta(a \mid s_1) \nabla_\theta Q_{\pi_\theta}(s_1, a)$$

Proof of Policy Gradient (10/10)

ROBOT INTELLIGENCE LAB

$$\nabla_\theta \eta(\pi_\theta) = \frac{1}{1-\gamma} \sum_{s'} \sum_{a} \nabla_\theta \pi_\theta(a \mid s') Q_{\pi_\theta}(s', a) \rho_{\pi_\theta}(s')$$

$$\propto \mathbb{E}_{s \sim \rho_{\pi_\theta}} \left[\sum_{a} \nabla_\theta \pi_\theta(a \mid s) Q_{\pi_\theta}(s, a) \right]$$

- Note that the states should be sampled from the distribution induced from the current policy (i.e., $s \sim \rho_{\pi_\theta}(s)$)
- This makes policy gradient methods **on-policy**.

Log Ratio Trick

$$\nabla_\theta \eta(\pi_\theta) = \frac{1}{1-\gamma} \sum_{s'} \sum_{a} \nabla_\theta \pi_\theta(a \mid s') Q_{\pi_\theta}(s', a) \rho_{\pi_\theta}(s')$$

- However, we should summate over all possible states and actions.

Log Ratio Trick

$$\nabla_\theta \eta(\pi_\theta) = \frac{1}{1-\gamma} \sum_{s'} \sum_{a} \nabla_\theta \pi_\theta(a \mid s') Q_{\pi_\theta}(s', a) \rho_{\pi_\theta}(s')$$

- However, we should summate over all possible states and actions.

- We can use the log ratio trick to overcome this issue.

$$\nabla_\theta \mathbb{E}\left[f(x)\right] = \sum_x f(x) \nabla_\theta p_\theta(x) = \sum_x f(x) p_\theta(x) \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} = \sum_x f(x) p_\theta(x) \nabla_\theta \log p_\theta(x) = \mathbb{E}\left[f(x) \nabla_\theta \log p_\theta(x)\right]$$

- To summarize

$$\nabla_\theta \mathbb{E}\left[f(x)\right] = \mathbb{E}\left[f(x) \nabla_\theta \log p_\theta(x)\right]$$

Log Ratio Trick

$$\nabla_\theta \eta(\pi_\theta) = \frac{1}{1-\gamma} \sum_{s'} \sum_{a} \nabla_\theta \pi_\theta(a \mid s') Q_{\pi_\theta}(s', a) \rho_{\pi_\theta}(s')$$

$$= \frac{1}{1-\gamma} \sum_{s'} \sum_{a} \pi_\theta(a \mid s') \frac{\nabla_\theta \pi_\theta(a \mid s')}{\pi_\theta(a \mid s')} Q_{\pi_\theta}(s', a) \rho_{\pi_\theta}(s')$$

$$= \frac{1}{1-\gamma} \sum_{s'} \sum_{a} \pi_\theta(a \mid s') \nabla_\theta \log \pi_\theta(a \mid s') Q_{\pi_\theta}(s', a) \rho_{\pi_\theta}(s')$$

$$= \frac{1}{1-\gamma} \mathbb{E}_{a \sim \pi_\theta, s \sim \rho_\pi(S)} \left[\nabla_\theta \log \pi_\theta(a \mid s') Q_{\pi_\theta}(s', a) \right]$$

$$\approx \nabla_\theta \log \pi_\theta(a_t \mid s_t) Q_{\pi_\theta}(s_t, a_t)$$

- To summarize

$$\nabla_\theta \eta(\pi_\theta) \approx \nabla_\theta \log \pi_\theta(a_t \mid s_t) Q_{\pi_\theta}(s_t, a_t)$$

Off-Policy vs. On-Policy Methods

ROBOT INTELLIGENCE LAB

- Off-Policy Learning
  - In **Q-Learning**, do we have any assumptions on the trajectories?

$$\widehat{Q}(s,a) \leftarrow \widehat{Q}(s,a) + \alpha \left(r(s,a,s') + \gamma \max_{a'} \widehat{Q}(s',a') - \widehat{Q}(s,a) \right)$$

- On-Policy Learning
  - How about **policy gradient**?

$$\nabla_\theta \eta(\pi_\theta) \approx \nabla_\theta \log \pi_\theta(a_t \mid s_t) Q_{\pi_\theta}(s_t, a_t)$$

Trust Region Policy Optimization (TRPO)

"Trust Region Policy Optimization", 2015

PG as an optimization problem

ROBOT INTELLIGENCE LAB

- Policy-based reinforcement learning is an optimization problem.

- The goal is to find θ that maximizes

$$\eta(\pi_\theta) = \mathbb{E}_{s,a}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right]$$

where $s_0 \sim \rho_0(s)$, $a_t \sim \pi(a_t \mid s_t)$, $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$.

PG as an optimization problem

ROBOT INTELLIGENCE LAB

- Policy-based reinforcement learning is an optimization problem.

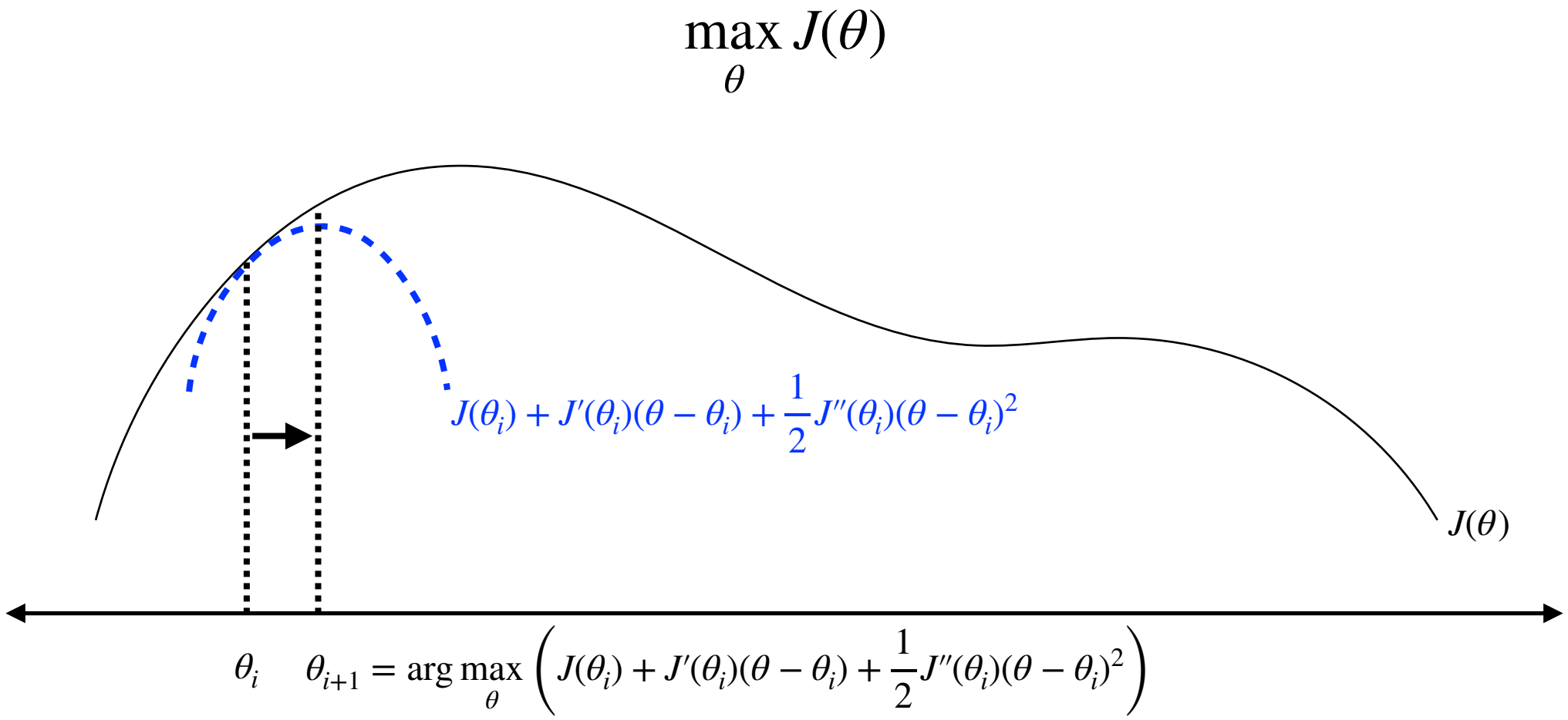
- The goal is to find θ that maximizes

$$\eta(\pi_\theta) = \mathbb{E}_{s,a}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right]$$

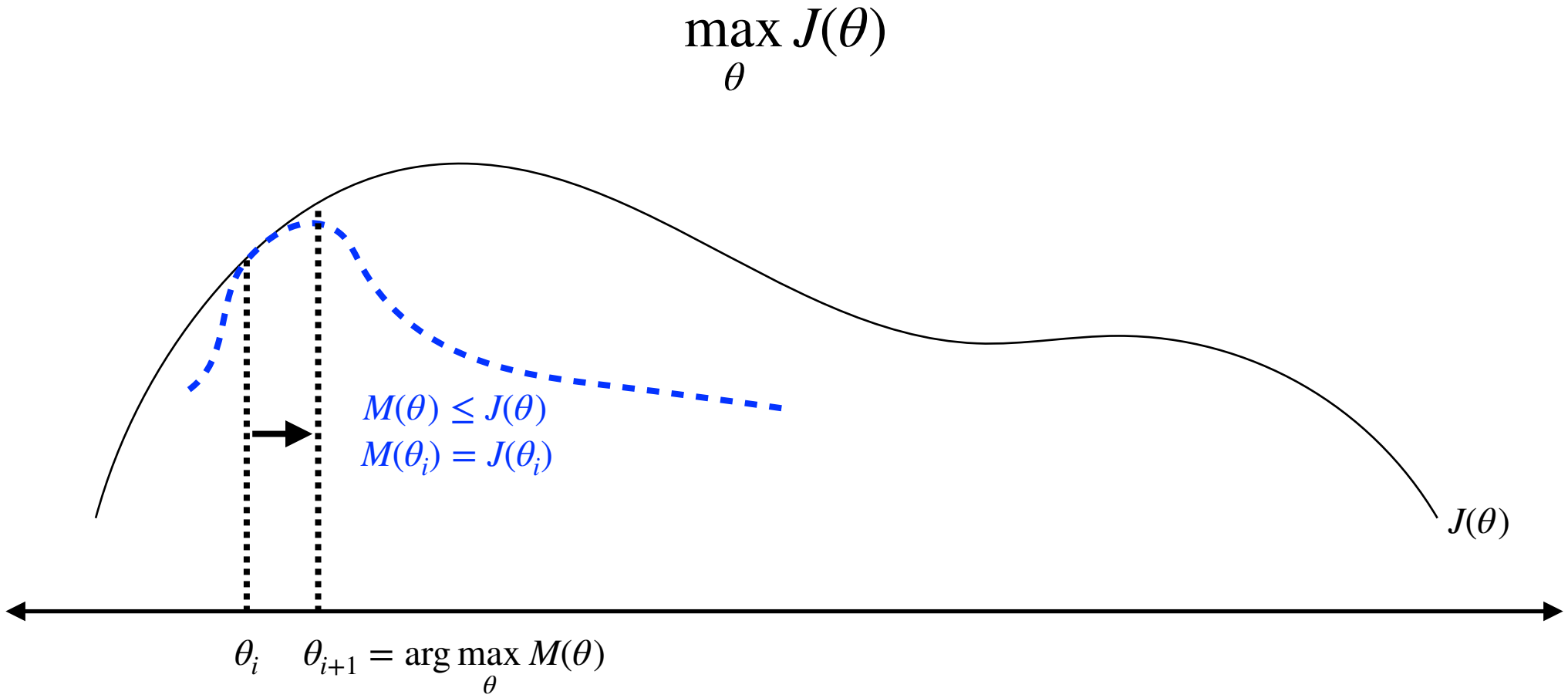
where $s_0 \sim \rho_0(s)$, $a_t \sim \pi_\theta(a_t \mid s_t)$, $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$.

- We can use optimization techniques:
  - Minorization maximization
  - Conjugate gradient descent

Newton Method



Minorization Maximization



Preliminaries

ROBOT INTELLIGENCE LAB

- The goal of reinforcement learning is to find π_θ that maximizes the expected return:

$$\eta(\pi_\theta) = \mathbb{E}_{s,a}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right]$$

where $s_0 \sim \rho_0(s)$, $a_t \sim \pi(a_t \mid s_t)$, $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$.

- Basic definitions of Markov decision processes

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s,a}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right]$$

$$V_\pi(s_t) = \mathbb{E}_{s,a}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right]$$

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

Useful Identity

- The improvement of the expected return:

$$\eta(\pi') = \eta(\pi) + \mathbb{E}_{s,a\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t, a_t)\right]$$

Improvement of π' over π

- Let $\rho_\pi(s)$ be the (unnormalized) discounted visitation frequencies

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \cdots$$

- Then the return improvement can be written as

$$\eta(\pi') = \eta(\pi) + \sum_s \rho_{\pi'}(s) \sum_a \pi'(a \mid s) A_\pi(s, a).$$

Performance Improvement

ROBOT INTELLIGENCE LAB

- The return improvement

$$\eta(\pi') = \eta(\pi) + \sum_s \rho_{\pi'}(s) \sum_a \pi'(a \mid s) A_\pi(s, a)$$

- Then, the **policy improvement** step of policy iteration will increase the policy performance if the following is guaranteed:

$$\sum_a \pi'(a \mid s) A_\pi(s, a) \geq 0$$

- Hence, $\pi'(s) = \arg\max_a A_\pi(s, a)$ will improve the policy there is at least one state-action pair with a positive value per each state s.
- However, due to the approximation of $A_\pi(s, a)$, it is not always guaranteed.

Performance Improvement

ROBOT INTELLIGENCE LAB

- The following return improvement is not practical due to $\rho_{\pi'}(s)$:

$$\eta(\pi') = \eta(\pi) + \sum_s \rho_{\pi'}(s) \sum_a \pi'(a \mid s) A_\pi(s, a)$$

- Why?

Performance Improvement

ROBOT INTELLIGENCE LAB

- The following return improvement is not practical due to $\rho_{\pi'}(s)$:

$$\eta(\pi') = \eta(\pi) + \sum_s \rho_{\pi'}(s) \sum_a \pi'(a \mid s) A_\pi(s, a)$$

- Why?

- The following local approximation, $\rho_{\pi'}(s) \Rightarrow \rho_\pi(s)$, is made:

$$L_\pi(\pi') \approx \eta(\pi')$$

$$L_\pi(\pi') = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \pi'(a \mid s) A_\pi(s, a)$$

Performance Improvement

ROBOT INTELLIGENCE LAB

- Local approximation:

$$L_{\pi}(\pi') = \eta(\pi) + \sum_{s} \rho_{\pi}(s) \sum_{a} \pi'(a \mid s) A_{\pi}(s, a)$$

- Hence the following can be used as a learning objective:

$$\mathbb{E}_{s_t \sim P, a_t \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A_{\pi}(s_t, a_t) \right]$$

- If we want to use the state-action pairs collected from the current policy π, we can use importance-sampling:

$$\mathbb{E}_{s_t \sim P, a_t \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t \mid s_t)}{\pi(a_t \mid s_t)} A_{\pi}(s_t, a_t) \right]$$

$$= \mathbb{E}_{s_t \sim \rho_{\pi}, a_t \sim \pi} \left[\frac{\pi'(a_t \mid s_t)}{\pi(a_t \mid s_t)} A_{\pi}(s_t, a_t) \right]$$

Minorization for RL

$$L_{\pi}(\pi') = \mathbb{E}_{s\sim\rho_{\pi},a\sim\pi}\left[\frac{\pi'(a_t \mid s_t)}{\pi(a_t \mid s_t)} A_{\pi}(s_t, a_t)\right]$$

- We can define the following **minorization** of $\eta(\pi)$ using KLD

$$M_{\pi}(\pi') = \eta(\pi) + L_{\pi}(\pi') - cD_{KL}^{max}(\pi, \pi')$$

where $D_{KL}^{max}(\pi, \pi') = \max_{s} D_{KL}\left(\pi(\cdot \mid s), \pi'(\cdot \mid s)\right)$

- Then the following properties hold:

$$M_{\pi}(\pi) = \eta(\pi)$$

$$M_{\pi}(\pi') \leq \eta(\pi')$$

Minorization for RL

- Now, we optimize

$$L_\pi(\pi') = \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}\left[\frac{\pi'(a\,|\,s)}{\pi(a\,|\,s)}A_\pi(s,a)\right]$$

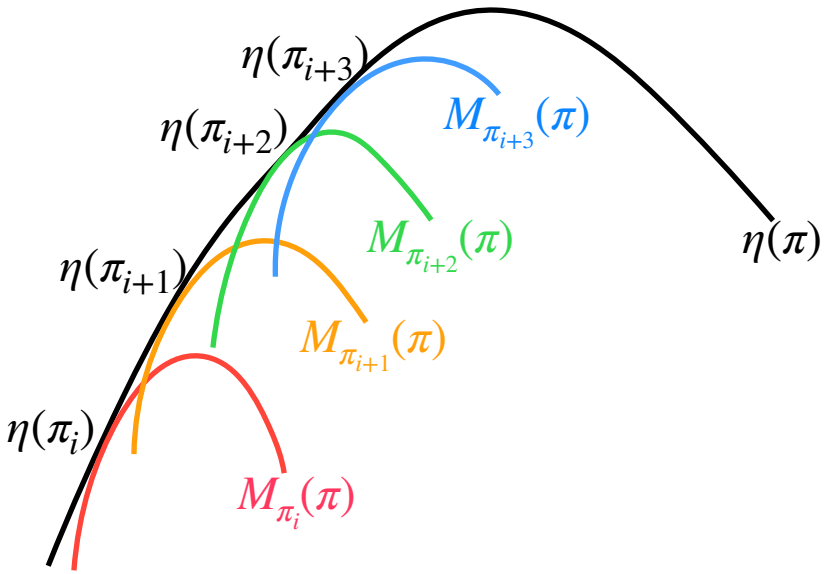
$$M_\pi(\pi') = \eta(\pi) + L_\pi(\pi') - cD_{KL}^{max}(\pi, \pi')$$

$$\max_{\theta_{i+1}} M_{\pi_{\theta_i}}(\pi_{\theta_{i+1}}) = \eta(\pi_{\theta_i}) + L_{\pi_{\theta_i}}(\pi_{\theta_{i+1}}) - cD_{KL}^{max}(\pi_{\theta_i}, \pi_{\theta_{i+1}})$$

- Lagrangian relaxation becomes

$$\max_{\theta_{i+1}} L_{\pi_{\theta_i}}(\pi_{\theta_{i+1}})$$

$$\text{subject to } D_{KL}^{max}(\pi_\theta, \pi_{\theta_{i+1}}) \le \delta$$



Trust Region Policy Optimization

$$\max_{\theta_{i+1}} L_{\pi_{\theta_i}}(\pi_{\theta_{i+1}}) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_i}}, a\sim\pi_{\theta_i}}\left[\frac{\pi_{\theta_{i+1}}(a\,|\,s)}{\pi_{\theta_i}(a\,|\,s)} A_{\pi_{\theta_i}}(s,a)\right]$$

$$\text{subject to } D_{KL}^{max}(\pi_\theta, \pi_{\theta_{i+1}}) \le \delta$$

- We approximate the KL divergence:

$$D_{KL}^{max}(\pi_\theta, \pi_{\theta_{i+1}}) = \max_s D_{KL}\left(\pi_{\theta_i}(\cdot\,|\,s), \pi_{\theta_{i+1}}(\cdot\,|\,s)\right)$$

$$\downarrow$$

$$D_{KL}^{\rho}(\pi_\theta, \pi_{\theta_{i+1}}) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_i}}}\left[D_{KL}\left(\pi_{\theta_i}(\cdot\,|\,s), \pi_{\theta_{i+1}}(\cdot\,|\,s)\right)\right]$$

Trust Region Policy Optimization

ROBOT INTELLIGENCE LAB

$$\max_{\theta_{i+1}} L_{\pi_{\theta_i}}(\pi_{\theta_{i+1}}) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_i}}, a\sim\pi_{\theta_i}}\left[\frac{\pi_{\theta_{i+1}}(a\,|\,s)}{\pi_{\theta_i}(a\,|\,s)} A_{\pi_{\theta_i}}(s,a)\right]$$

$$\text{subject to } D_{KL}^{\rho}(\pi_{\theta}, \pi_{\theta_{i+1}}) \le \delta$$

- In summary,
  - TRPO is a minorization maximization framework for RL.
  - Interpretation of the trust region method:
    1. Update policy distribution slowly
    2. Consider the geometry of the distribution space
  - There are two approximations: 1) $\mathbb{E}_{s\sim\rho_{\pi'}} \Rightarrow \mathbb{E}_{s\sim\rho_{\pi}}$ and 2) $D_{KL}^{\max} \Rightarrow D_{KL}^{\rho}$

Trust Region Policy Optimization

ROBOT INTELLIGENCE LAB

- How do we estimate $A_{\pi_{\theta_i}}$?
- We estimate $Q_{\pi_{\theta_i}}(s,a)$ instead of $A_{\pi_{\theta_i}}(s,a)$:

$$A_{\pi_{\theta_i}}(s,a) = Q_{\pi_{\theta_i}}(s,a) - V_{\pi_{\theta_i}}(s)$$

- We use the Monte Carlo Estimate of Q:

$$Q_{\pi_{\theta_i}}(s_t,a_t) \approx G_t = \sum_{k=1} \gamma^k R_{t+1+k}$$

Trust Region Policy Optimization

ROBOT INTELLIGENCE LAB

$$\max_{\theta_{i+1}} L_{\pi_{\theta_i}}(\pi_{\theta_{i+1}}) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_i}}, a\sim\pi_{\theta_i}}\left[\frac{\pi_{\theta_{i+1}}(a\,|\,s)}{\pi_{\theta_i}(a\,|\,s)} A_{\pi_{\theta_i}}(s,a)\right]$$

$$\text{subject to } D_{KL}^{\rho}(\pi_{\theta}, \pi_{\theta_{i+1}}) \le \delta$$

⇩ Linear approximation to the loss and quadratic approximation to the constraint

$$\max_{\theta} \nabla_{\theta} L_{\theta_{old}}(\theta)\,|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old})$$

$$\text{subject to } \frac{1}{2}(\theta_{old} - \theta)^T H(\theta_{old})(\theta_{old} - \theta) \le \delta$$

$$\text{where } H(\theta_{old})_{(i,j)} = \frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}\mathbb{E}_{s\sim\rho_{\pi}}\left[D_{KL}(\pi(\,\cdot\,|\,s,\theta_{old})\|\pi(\,\cdot\,|\,s,\theta))\right]|_{\theta=\theta_{old}}$$

Trust Region Policy Optimization

ROBOT INTELLIGENCE LAB

- The final TRPO objective becomes:

$$\max_{\theta} g(\theta_{old})^T(\theta - \theta_{old})$$

$$\text{subject to } \frac{1}{2}(\theta_{old} - \theta)^T H(\theta_{old})(\theta_{old} - \theta) \leq \delta$$

$$\text{where } g(\theta_{old}) = \nabla_\theta L_{\theta_{old}}(\theta)\,|_{\theta=\theta_{old}} \text{ and}$$

$$H(\theta_{old})_{(i,j)} = \frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}\mathbb{E}_{s\sim\rho_\pi}\left[D_{KL}(\pi(\cdot\,|\,s,\theta_{old})\|\pi(\cdot\,|\,s,\theta))\right]|_{\theta=\theta_{old}}$$

- The update rule of the above problem is

$$\theta_{new} = \theta_{old} + \frac{1}{\lambda}H(\theta_{old})^{-1}g(\theta_{old})$$

Trust Region Policy Optimization

ROBOT INTELLIGENCE LAB

- The update rule of the above problem is

$$\theta_{new} = \theta_{old} + \frac{1}{\lambda} H(\theta_{old})^{-1} g(\theta_{old})$$

- However, the hessian matrix $H(\theta_{old}) \in \mathbb{R}^{n \times n}$ where n is the number of parameters and the computational complexity of the inverse becomes $O(n^3)$.

- Instead of computing H^{-1}, we solve the linear equation $Hx = g$ using a conjugate gradient method.

ROBOT INTELLIGENCE LAB

Proximal Policy Optimization (PPO)

"Proximal Policy Optimization Algorithms", 2017

Preliminaries

- Policy gradient method
  - The gradient estimate of the policy w.r.t. the return is

$$\hat{g} = \mathbb{E}_{s_t, a_t}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\right]$$

where $\hat{A}_t$ is an estimator of the advantage function.

- Trust region method
  - The TRPO objective is

$$\max_\theta \mathbb{E}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t\right]$$

$$\text{s.t. } D_{KL}^{\rho}\left[\pi_{old}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)\right] \leq \delta$$

Clipped Surrogate Objective

ROBOT INTELLIGENCE LAB

- The objective of the TRPO is:

$$L(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t\right] = \mathbb{E}\left[r_t(\theta)\hat{A}_t\right]$$

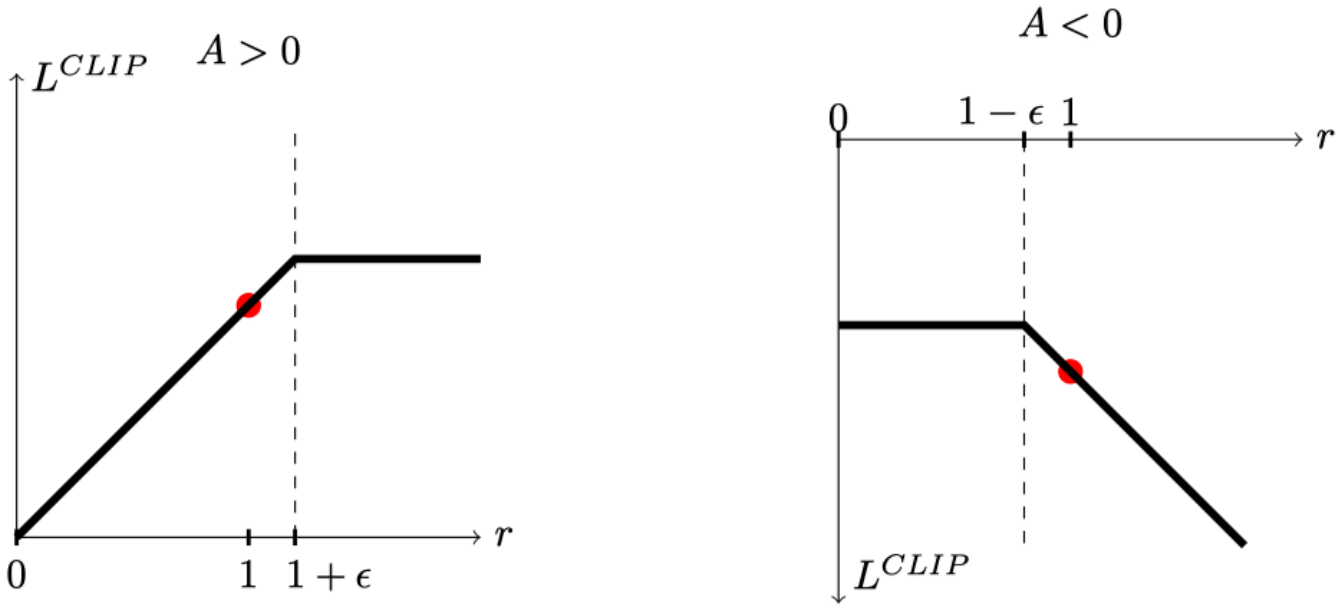
- The main objective of clipped surrogate is:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

  - The first term $r_t(\theta)\hat{A}_t$ is identical to the TRPO objective.
  - The second term clips the probability ratio $r_t(\theta)$, which removes the incentive for moving $r_t(\theta)$ outside of the interval $[1-\epsilon, 1+\epsilon]$.

Clipped Surrogate Objective

$$L(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t\right] = \mathbb{E}\left[r_t(\theta)\hat{A}_t\right] \text{ and } L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$



- When $A_t > 0$, we have to worry about increasing $L(\theta)$ by increasing $r_t(\theta)$, and vice versa. Hence, we clip the objective when $r_t(\theta)$ exceeds $1+\epsilon$ when $A_t > 0$.

Proximal Policy Optimization (Adaptive KL Penalty)

ROBOT INTELLIGENCE LAB

- The TRPO objective is:

$$\max_{\theta} \mathbb{E}\left[\frac{\pi_{\theta}(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t\right] \text{ s.t. } D_{KL}^{\rho}\left[\pi_{old}(\cdot \mid s_t), \pi_{\theta}(\cdot \mid s_t)\right] \leq \delta$$

- The unconstrained objective of TRPO is:

$$L(\theta) = \max_{\theta} \mathbb{E}\left[\frac{\pi_{\theta}(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t - \beta D_{KL}^{\rho}\left[\pi_{\theta_{old}}(\cdot \mid s_t), \pi_{\theta}(\cdot \mid s_t)\right]\right]$$

- The adaptive KL penalty method for PPO is to adaptively change β by checking $d = \mathbb{E}_t\left[D_{KL}[\pi_{\theta_{old}}, \pi_{\theta}]\right]$:
  - If $d < d_{targ}/1.5$, $\beta \leftarrow \beta/2$
  - If $d > d_{targ} \times 1.5$, $\beta \leftarrow \beta \times 2$

PPO Implementation

ROBOT INTELLIGENCE LAB

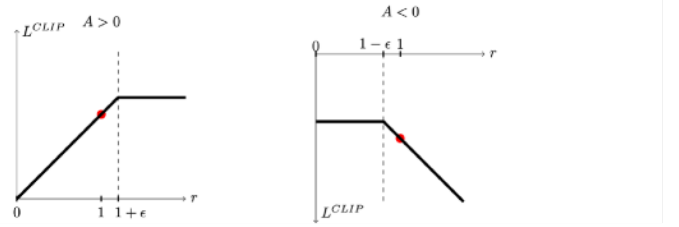
```python
# Update policy
for _ in range(self.train_pi_iters):

    # Cipped Surrogate Obejctive
    _, logp_a, _, _ = self.actor_critic.policy(obs, act)
    ratio = torch.exp(logp_a - logp_a_old)  # pi(a|s) / pi_old(a|s)
    min_adv = torch.where(adv > 0, (1 + self.clip_ratio) * adv, (1 - self.clip_ratio) * adv)
    pi_loss = -torch.mean(torch.minimum(ratio * adv, min_adv))

    # Gradient clipping
    self.train_pi.zero_grad()
    pi_loss.backward()
    self.train_pi.step()

    # KL divergence upper-bound (trust region)
    kl = torch.mean(logp_a_old - logp_a)
    if kl > 1.5 * self.target_kl:
        break
```

$$L(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] = \mathbb{E}\left[r_t(\theta)\hat{A}_t\right] \text{ and } L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$



- The adaptive KL penalty method for PPO is to adaptively change β by checking $d = \mathbb{E}_t\left[D_{KL}[\pi_{\theta_{old}}, \pi_\theta]\right]$:
  - If $d < d_{targ}/1.5$, $\beta \leftarrow \beta/2$
  - If $d > d_{targ} \times 1.5$, $\beta \leftarrow \beta \times 2$

ROBOT INTELLIGENCE LAB

Generalized Advantage Estimation (GAE)

"HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION," 2018

Advantage Function Estimation

ROBOT INTELLIGENCE LAB

- Let V be an approximate value function. Then define

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

i.e., the TD residual of V with discount γ.

- Note that δ_t^V can be considered as an estimate of the advantage of the action a_t, i.e., $\hat{A}_t$. Now, let's define the following series:
  - $\hat{A}_t^{(1)} = \delta_t^V$
  - $\hat{A}_t^{(2)} = \delta_t^V + \gamma\delta_{t+1}^V$
  - $\hat{A}_t^{(3)} = \delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V$
- Finally, we define the λ-exponentially-weighted average of $\hat{A}_t$:

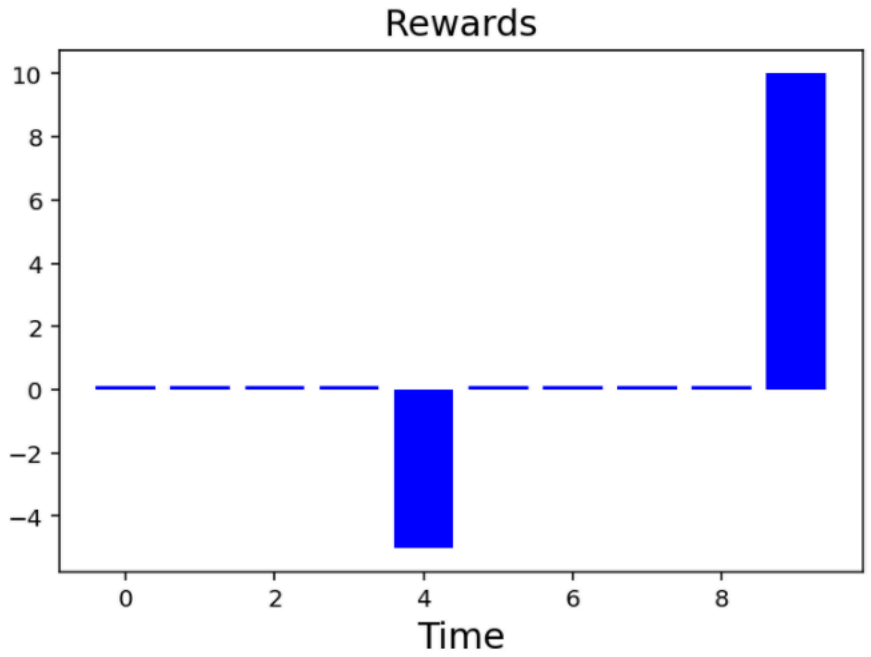
$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots\right) = \sum_{t=0}^{\infty}(\gamma\lambda)^l\delta_{t+1}^V$$

Advantage Function Estimation

ROBOT INTELLIGENCE LAB

Rewards

```
1  # Plot rewards
2  plt.bar(times,rewards,color='b')
3  plt.title("Rewards",fontsize=15)
4  plt.xlabel("Time",fontsize=15)
5  plt.show()
```



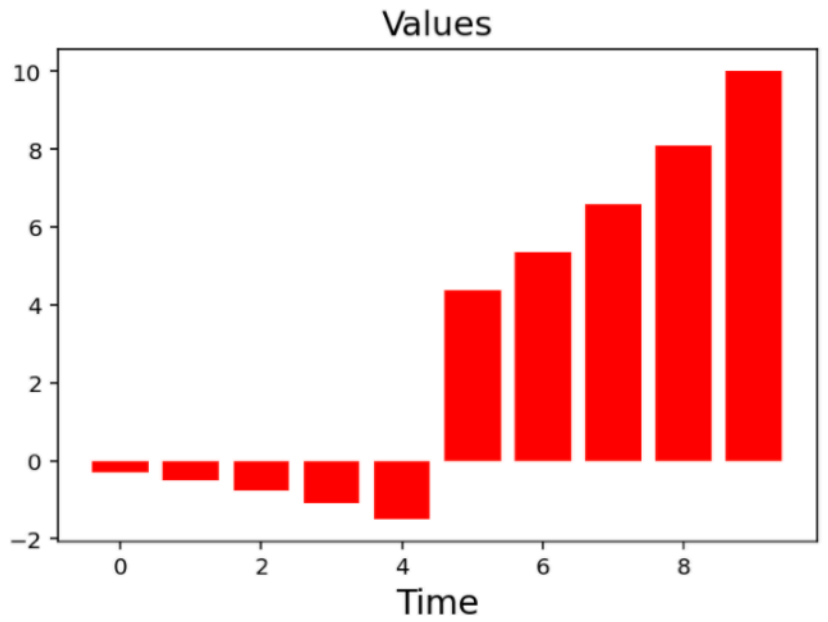
Advantage Function Estimation

ROBOT INTELLIGENCE LAB

Values

$V(s_t) = \sum_{l=0}^{\infty} \gamma^l r(s_{t+l})$ and $V(s_t) = r(s_t) + \gamma V(s_{t+1})$

```
values = np.zeros(L); values[L-1] = rewards[L-1]
for t in reversed(range(L-1)):
    values[t] = rewards[t] + gamma*values[t+1]
# Plot values
plt.bar(times,values,color='r')
plt.title("Values",fontsize=15)
plt.xlabel("Time",fontsize=15)
plt.show()
```



Advantage Function Estimation

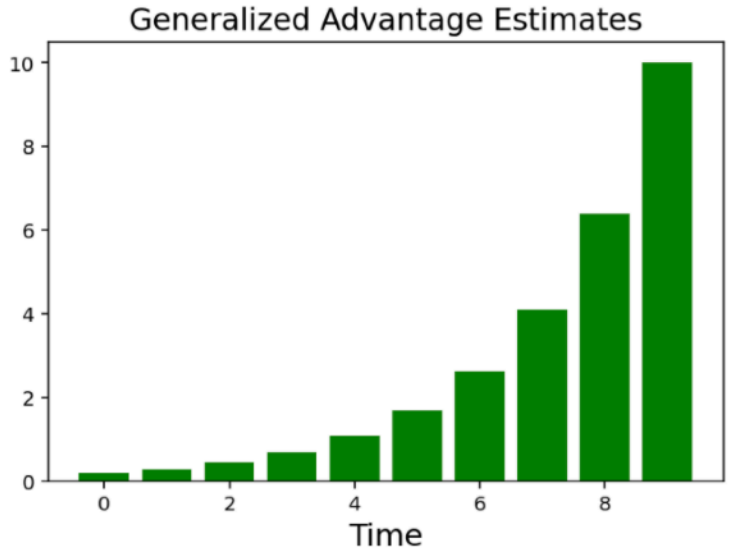
ROBOT INTELLIGENCE LAB

Generalized Advantage Estimates

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) \text{ - (9)}$$

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \text{ - (16)}$$

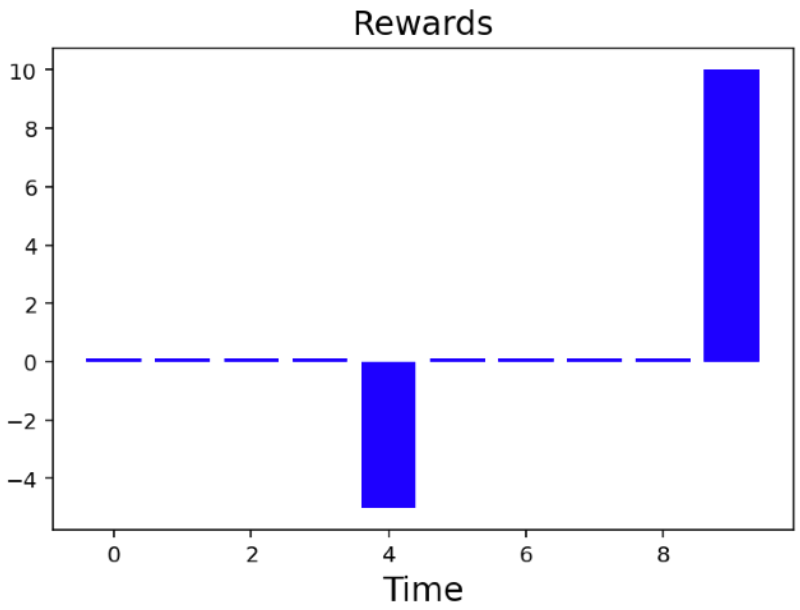
```
1  gaes = np.zeros(L); gaes[L-1] = rewards[L-1]
2  for t in reversed(range(L-1)):
3      delta = rewards[t] + (gamma*values[t+1]) - values[t]
4      gaes[t] = delta + (gamma*lamda*gaes[t+1])
5  # Plot GAEs
6  plt.bar(times,gaes,color='g')
7  plt.title("Generalized Advantage Estimates",fontsize=15)
8  plt.xlabel("Time",fontsize=15)
9  plt.show()
```

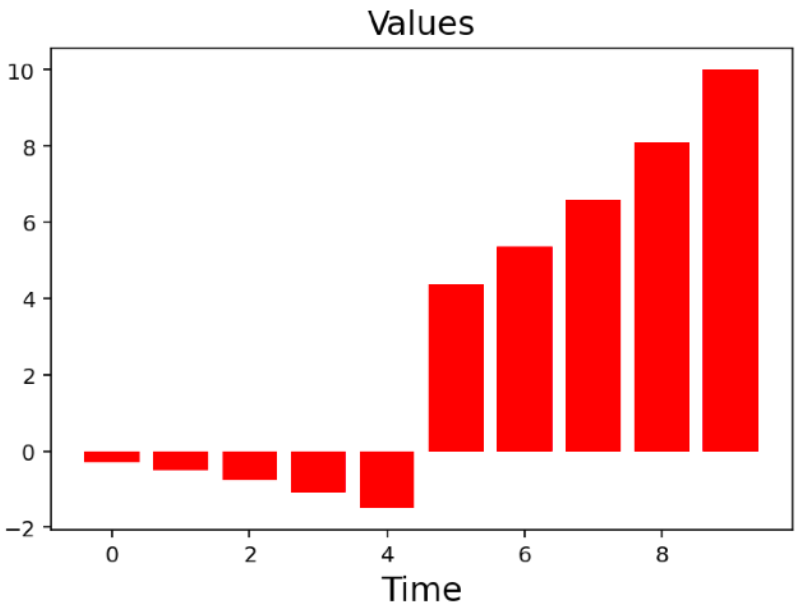


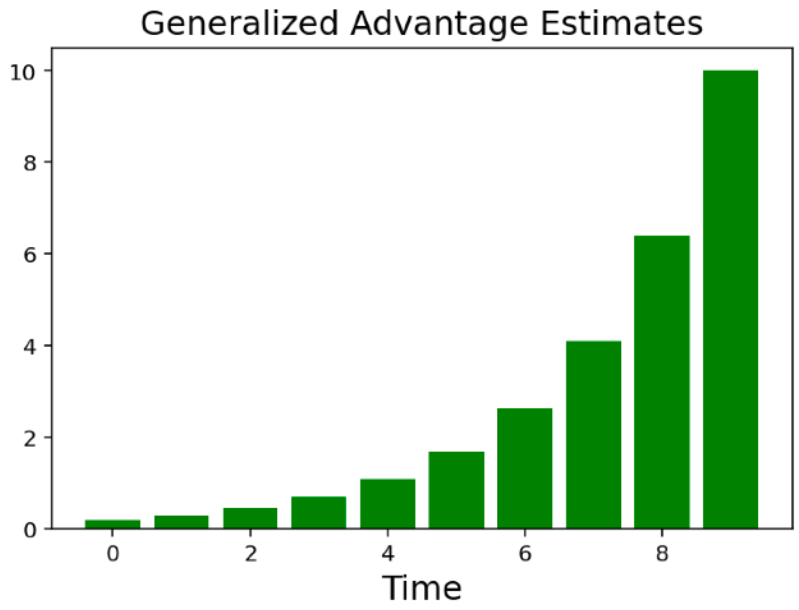
Advantage Function Estimation

ROBOT INTELLIGENCE LAB

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots\right) = \sum_{t=0}^{\infty}(\gamma\lambda)^l\delta_{t+1}^V$$

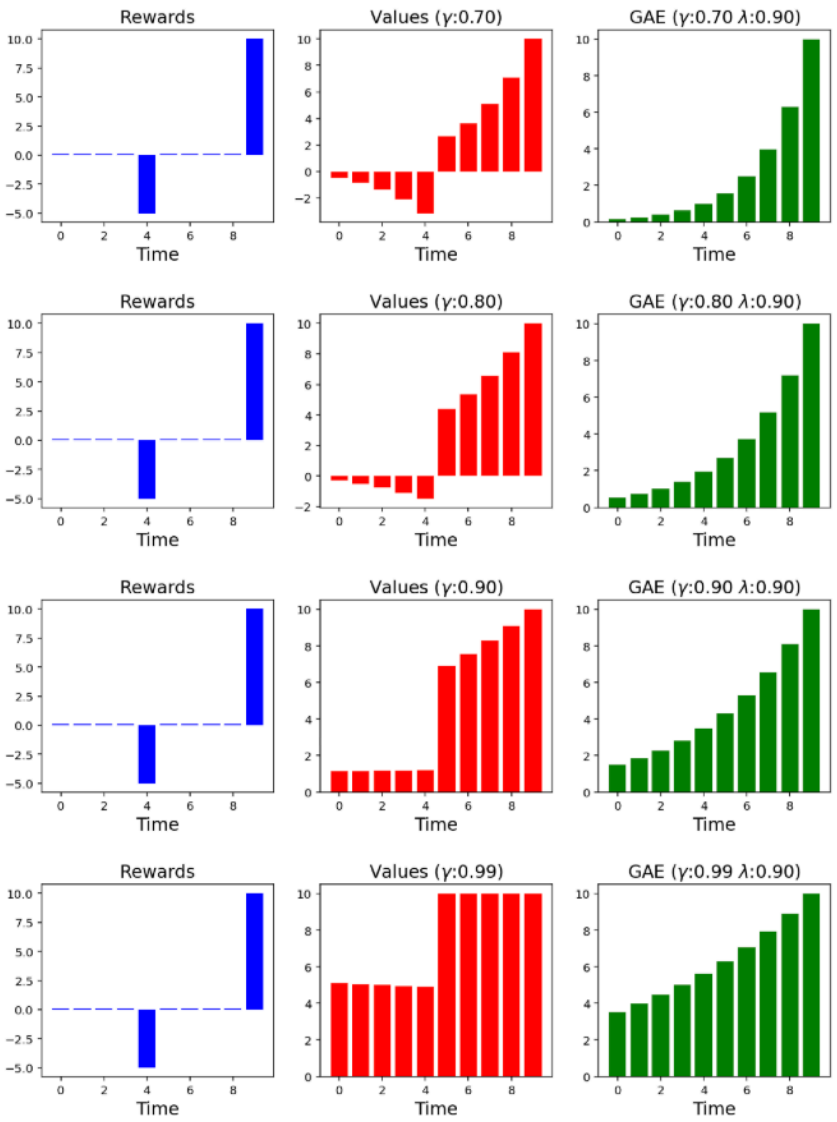


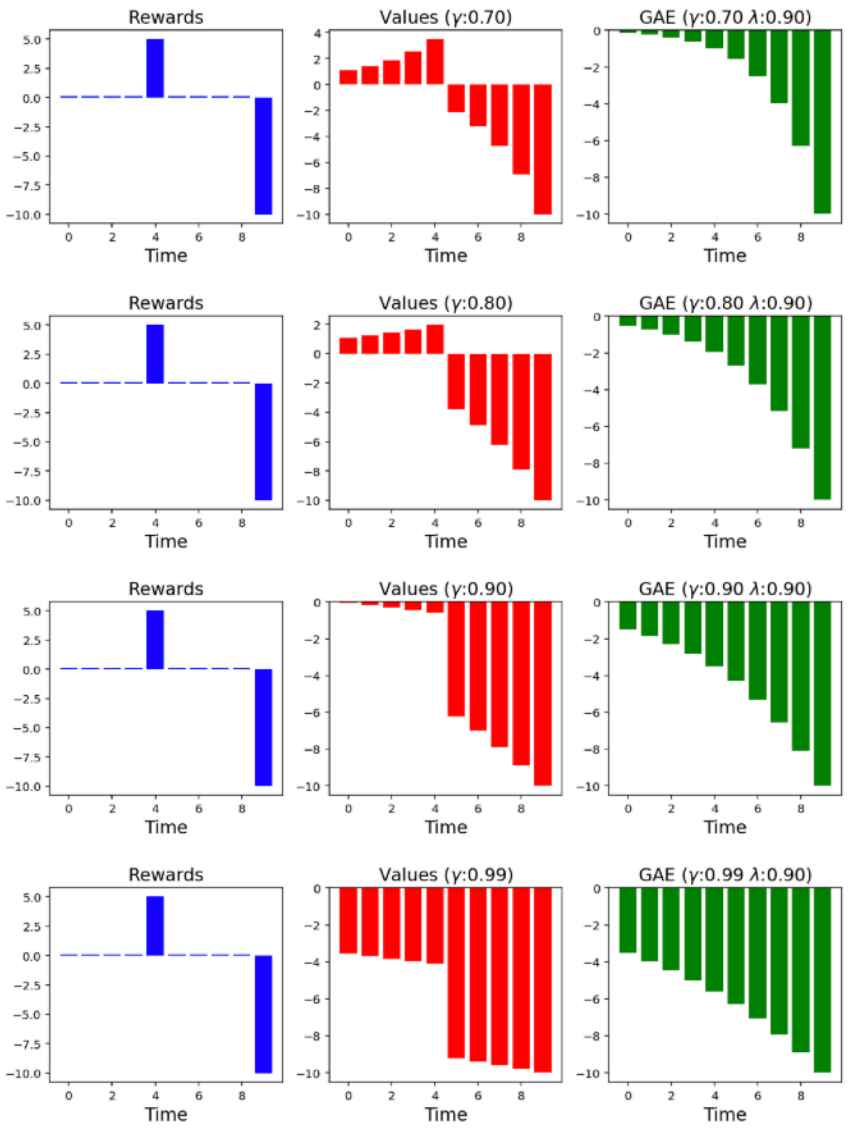




https://gist.github.com/sjchoi86/38c7a378cfa482a1cde5630e5dde937e

Advantage Function Estimation



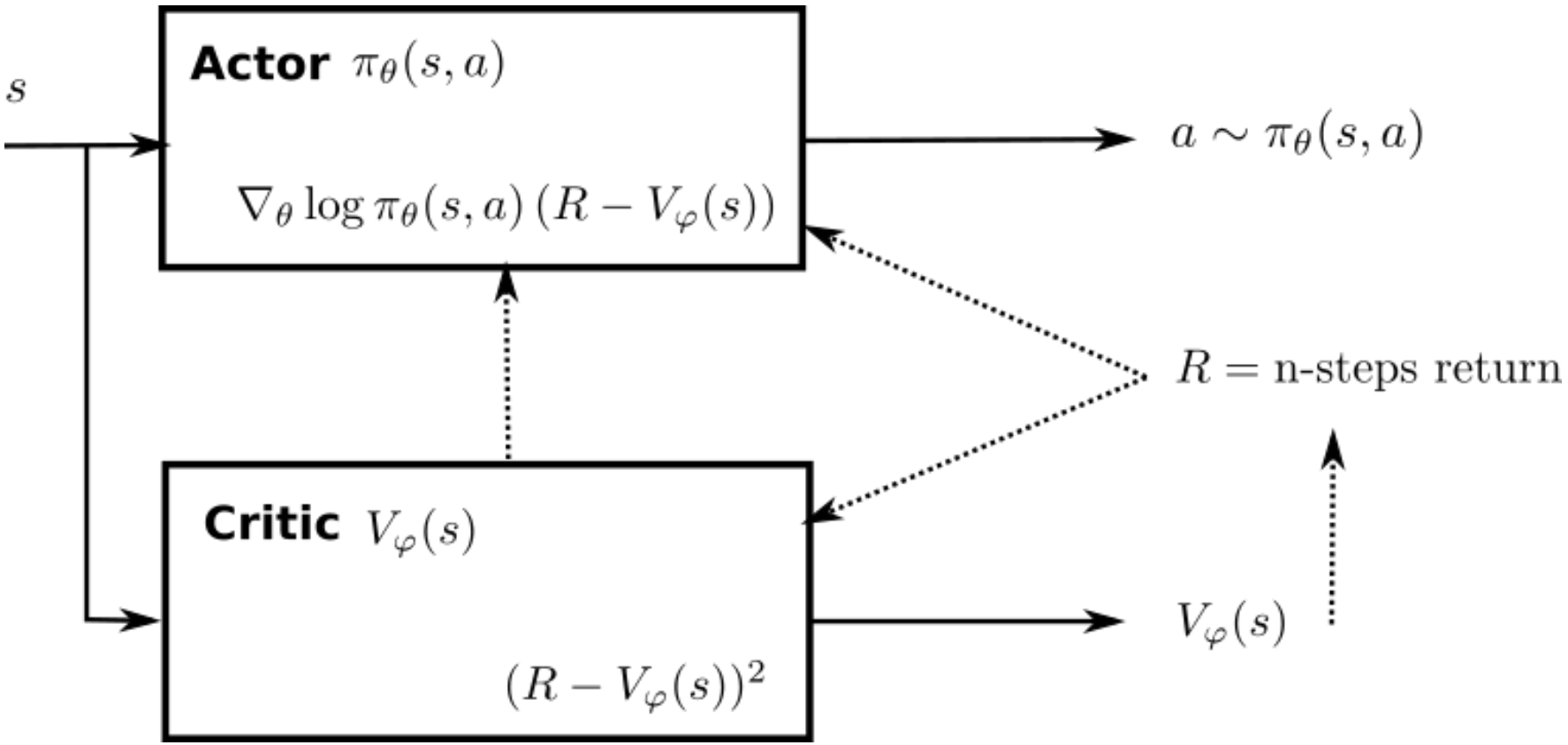


ROBOT INTELLIGENCE LAB

Soft Actor-Critic (SAC)

"Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," 2018

Actor-Critic Method



Actor-Critic Method

ROBOT INTELLIGENCE LAB

1. Acquire a batch of transitions (s, a, r, s') using the current policy π_θ.
2. For each state encountered, compute the discounted sum of the next n rewards and use the critic to estimate the value of the state:

$$R_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} V_\phi(s_{t+n+1})$$

3. Update the actor using:

$$\nabla_\theta J(\theta) = \sum_i \nabla_\theta \log \pi_\theta(s_t, a_t)(R_t - V_\phi(s_t))$$

4. Update the critic using:

$$L(\phi) = \sum_t (R_t - V_\phi(s_t))^2$$

5. Repeat

Maximum Entropy RL

ROBOT INTELLIGENCE LAB

- Standard RL objective

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) \right]$$

- Maximum Entropy RL objective

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha \mathscr{H} \left(\pi(\cdot \,|\, s_t) \right) \right]$$

Maximum Entropy RL

ROBOT INTELLIGENCE LAB

- Maximum Entropy RL objective

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha \mathscr{H} \left(\pi(\cdot \mid s_t) \right) \right]$$

- **Policy evaluation step**
  - The Bellman backup operator for Max-Ent RL is:

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

$$\text{where } V(s_{t+1}) = \mathbb{E}_{a_t\sim\pi} \left[Q(s_t, a_t) - \log \pi(a_t \mid s_t) \right].$$

- **Policy improvement step**

$$\pi_{new} = \arg\min_{\pi'} D_{KL} \left(\pi'(\cdot \mid s_t) \| \frac{\exp(Q^{\pi_{old}}(s_t, \cdot))}{Z^{\pi_{old}}(s_t)} \right)$$

Soft Actor-Critic

ROBOT INTELLIGENCE LAB

- SAC learns three functions: $V_\psi(s)$, $Q_\theta(s,a)$, and $\pi_\phi(a\,|\,s)$.
- For learning $V_\psi(s)$:

$$J_V(\psi) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\frac{1}{2}\left(V_\psi(s_t) - \mathbb{E}_{a_t\sim\pi_\phi}\left[Q_\theta(s_t,a_t) - \log\pi_\phi(a_t\,|\,s_t)\right]\right)^2\right]$$

where actions are being sampled from the current policy $\pi_\phi(a\,|\,s)$ not from the replay.

- For learning $Q_\theta(s,a)$:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t)\right)^2\right] \text{ where } \hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}}\left[V_\psi(s_{t+1})\right]$$

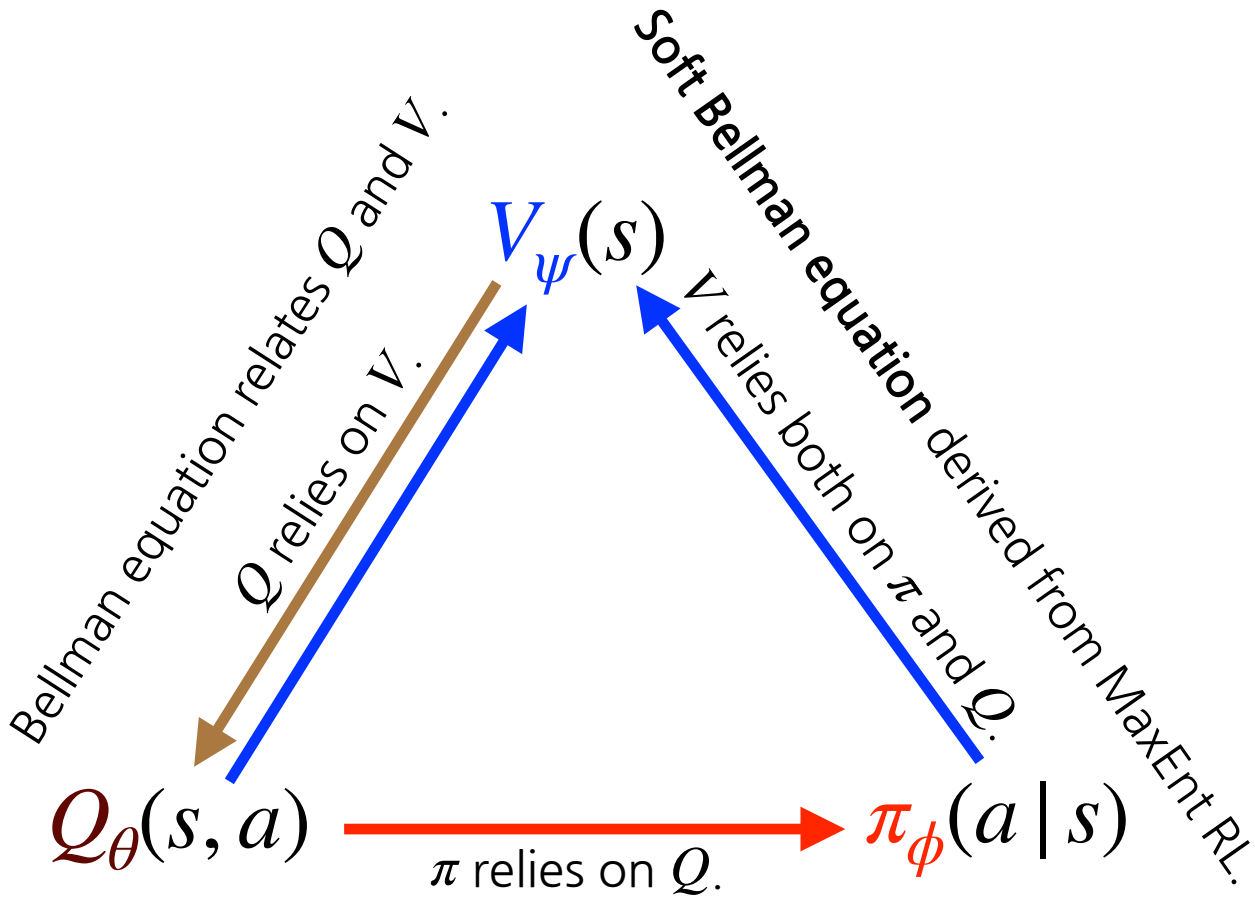
- For learning $\pi_\phi(a\,|\,s)$:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[D_{KL}\left(\pi_\phi(\,\cdot\,|\,s_t)\|\frac{\exp\left(Q_\theta(s_t,\,\cdot\,)\right)}{Z_\theta(s_t)}\right)\right]$$

If we re-parameterize the stochastic policy $a_t = f_\phi(\epsilon_t;s_t)$ where ϵ_t is sampled from some distribution,

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim D,\epsilon_t\sim\mathcal{N}}\left[\log\pi_\phi\left(f_\phi(\epsilon_t;s_t)\,|\,s_t\right) - Q_\theta\left(s_t, f_\phi(\epsilon_t;s_t)\right)\right]$$

Soft Actor-Critic



Policy improvement with KL control

SAC Implementation

ROBOT INTELLIGENCE LAB

```
def update_policy(self, data):
    o = data['obs1']
    _, pi, logp_pi = self.policy(o)
    q1_pi = self.q1(o, pi)
    q2_pi = self.q2(o, pi)
    min_q_pi = torch.minimum(q1_pi, q2_pi)
    pi_loss = torch.mean(self.alpha_pi*logp_pi - min_q_pi)

    self.train_pi.zero_grad()
    pi_loss.backward()
    self.train_pi.step()

    return pi_loss, logp_pi, min_q_pi

def update_Q(self, target, data):
    o,a,r,o2,d = data['obs1'],data['acts'],data['rews'],data['obs2'],data['done']
    _, pi_next, logp_pi_next = self.policy(o2)
    q1_targ = target.q1(o2, pi_next)
    q2_targ = target.q2(o2, pi_next)
    min_q_targ = torch.minimum(q1_targ, q2_targ)
    q_backup =  r + self.gamma*(1-d)*(min_q_targ - self.alpha_q*logp_pi_next
    q1 = self.q1(o, a)
    q2 = self.q2(o, a)
    q1_loss = 0.5*F.mse_loss(q1,q_backup.detach())
    q2_loss = 0.5*F.mse_loss(q2,q_backup.detach())
    value_loss = q1_loss + q2_loss

    self.train_q1.zero_grad()
    q1_loss.backward()
    self.train_q1.step()

    self.train_q2.zero_grad()
    q2_loss.backward()
    self.train_q2.step()

    return value_loss, q1, q2, logp_pi_next, q_backup, q1_targ, q2_targ
```

- SAC learns three functions: $V_\psi(s)$, $Q_\theta(s,a)$, and $\pi_\phi(a\,|\,s)$.
- For learning $V_\psi(s)$:

$$J_V(\psi) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\frac{1}{2}\left(V_\psi(s_t) - \mathbb{E}_{a_t\sim\pi_\phi}\left[Q_\theta(s_t,a_t) - \log\pi_\phi(a_t\,|\,s_t)\right]\right)^2\right]$$

where actions are being sampled from the current policy $\pi_\phi(a\,|\,s)$ not from the replay.

- For learning $Q_\theta(s,a)$:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t)\right)^2\right] \text{ where } \hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}}\left[V_\psi(s_{t+1})\right]$$

- For learning $\pi_\phi(a\,|\,s)$:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[D_{KL}\left(\pi_\phi(\cdot\,|\,s_t)\|\frac{\exp\left(Q_\theta(s_t,\cdot)\right)}{Z_\theta(s_t)}\right)\right]$$

If we re-parameterize the stochastic policy $a_t = f_\phi(\epsilon_t;s_t)$ where ϵ_t is sampled from some distribution,

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim D,\epsilon_t\sim\mathcal{N}}\left[\log\pi_\phi\left(f_\phi(\epsilon_t;s_t)\,|\,s_t\right) - Q_\theta\left(s_t, f_\phi(\epsilon_t;s_t)\right)\right]$$

Summary

ROBOT INTELLIGENCE LAB

- Policy Gradient Theorem
  - Optimize the policy directly via $\nabla_\theta \eta(\pi_\theta) \approx \nabla_\theta \log \pi_\theta(a_t \mid s_t) Q_{\pi_\theta}(s_t, a_t)$
- Trust Region Policy Optimization (**TRPO**)
  - From policy improvements using minorization maximization to a trust-region method.
- Proximal Policy Optimization (**PPO**)
  - Approximate TRPO with policy ratio clipping and adaptive KL weights.
- Generalized Advantage Estimation (**GAE**)
  - More robust than the value estimate, similar to TD(λ).
- Soft Actor-Critic (**SAC**)
  - Entropy-regularized RL with an actor-critic method.

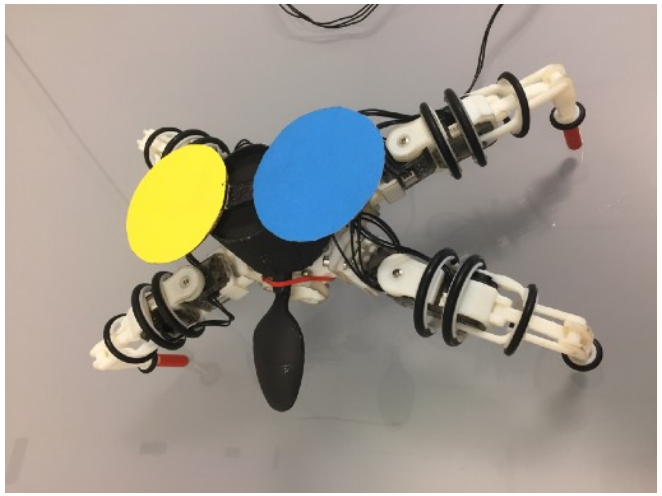
ROBOT INTELLIGENCE LAB

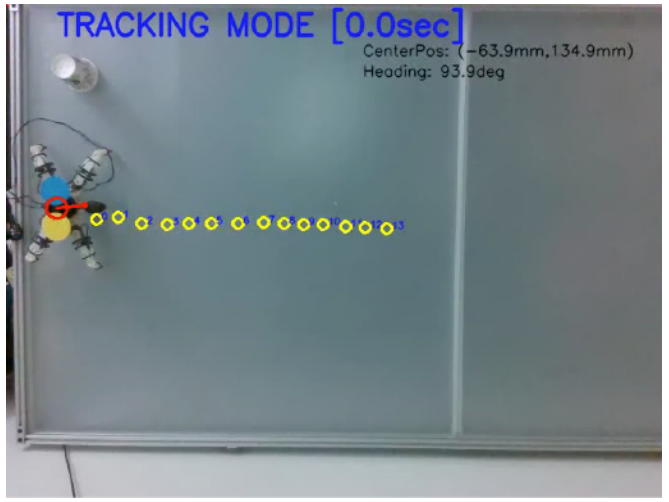
Trajectory-based RL

"Deep Latent Policy Gradient for Quadruped Locomotion with Snapbot," 2019

Motivation

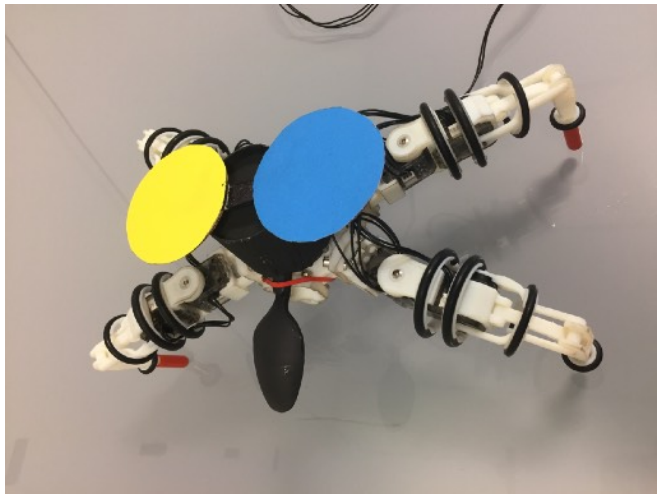
ROBOT INTELLIGENCE LAB

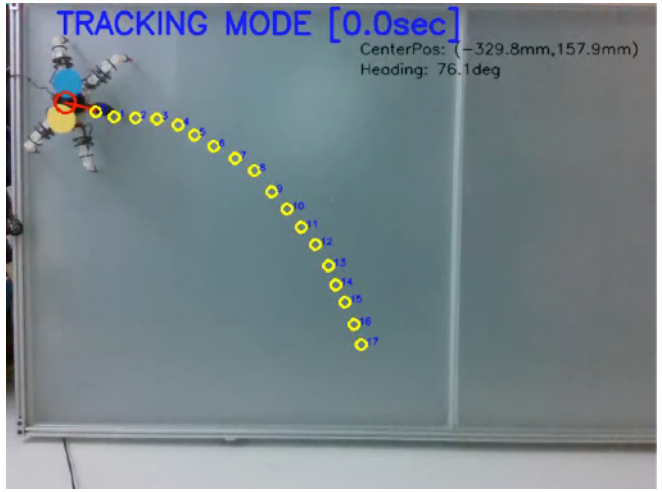




Use reinforcement learning to make **Snapbot** walk.

Motivation





We focus on two known drawbacks of **reinforcement learning** when applying RL to real physical systems.

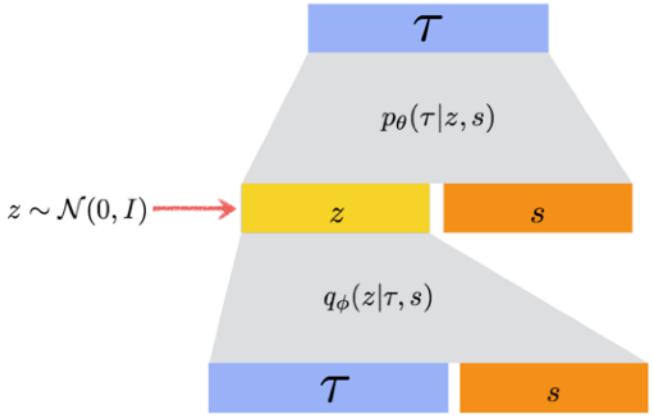
1. Inefficient **exploration** phase

2. Hard to design a proper **reward** function

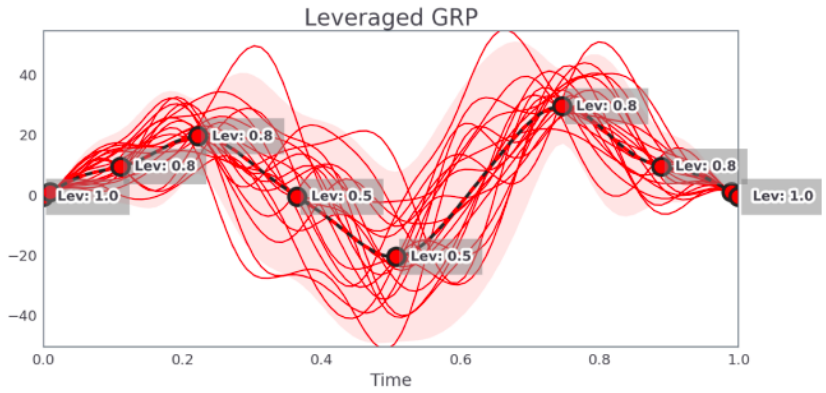
Proposed method

ROBOT INTELLIGENCE LAB

Deep Latent Policy Gradient



Leveraged Gaussian Random Path



- We propose a deep latent policy gradient method for optimizing a stochastic policy function.
- We present a leveraged Gaussian random path which defines a probability distribution over smooth trajectories.

Experiments

ROBOT INTELLIGENCE LAB

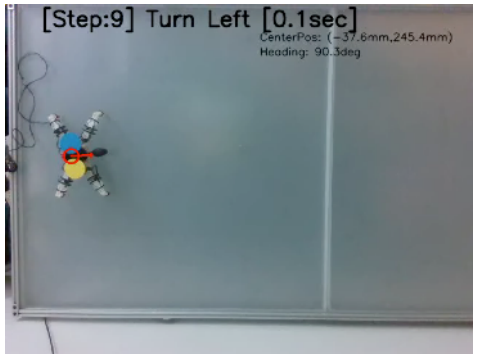
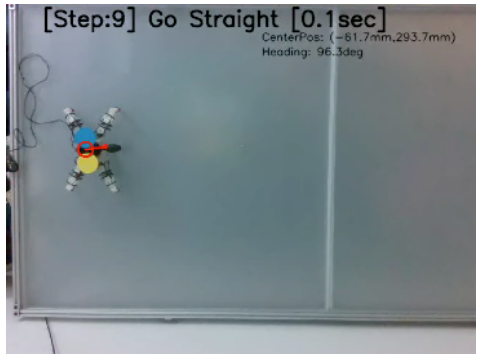
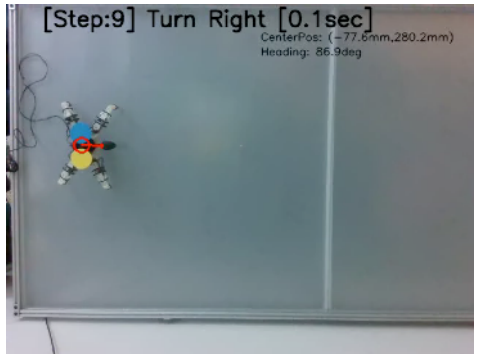
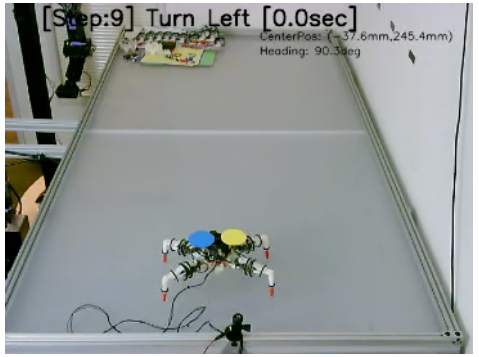
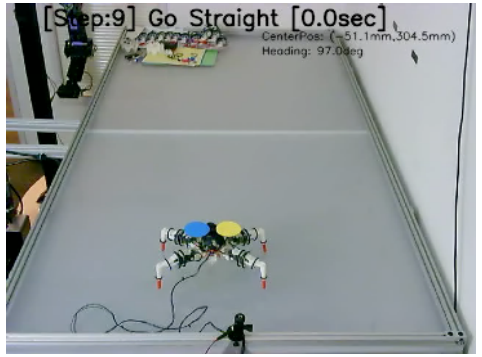
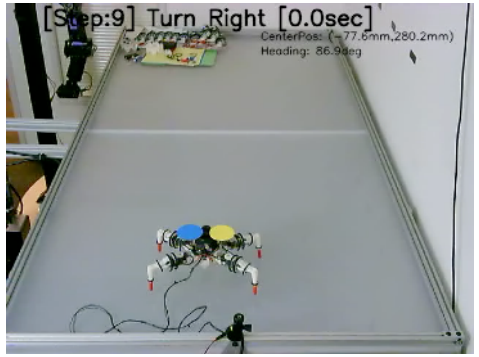
- **Learning from Scratch**: The initial policy function is randomly initialized and optimized with a single-stage policy learning.

- **Learning with Prior**: The policy function is initialized with four manually designed trot-gain motions to go forward and optimized with a single-stage policy learning.

- **Curriculum Learning**: The initial policy function is randomly initialized and optimized with a two-stage policy learning. In the first stage, Snapbot is incentivized by simply going forward. Then, in the second stage, high reward is given when Snapbot moves accordingly to the given control signal.

Experiments - Learning from Scratch

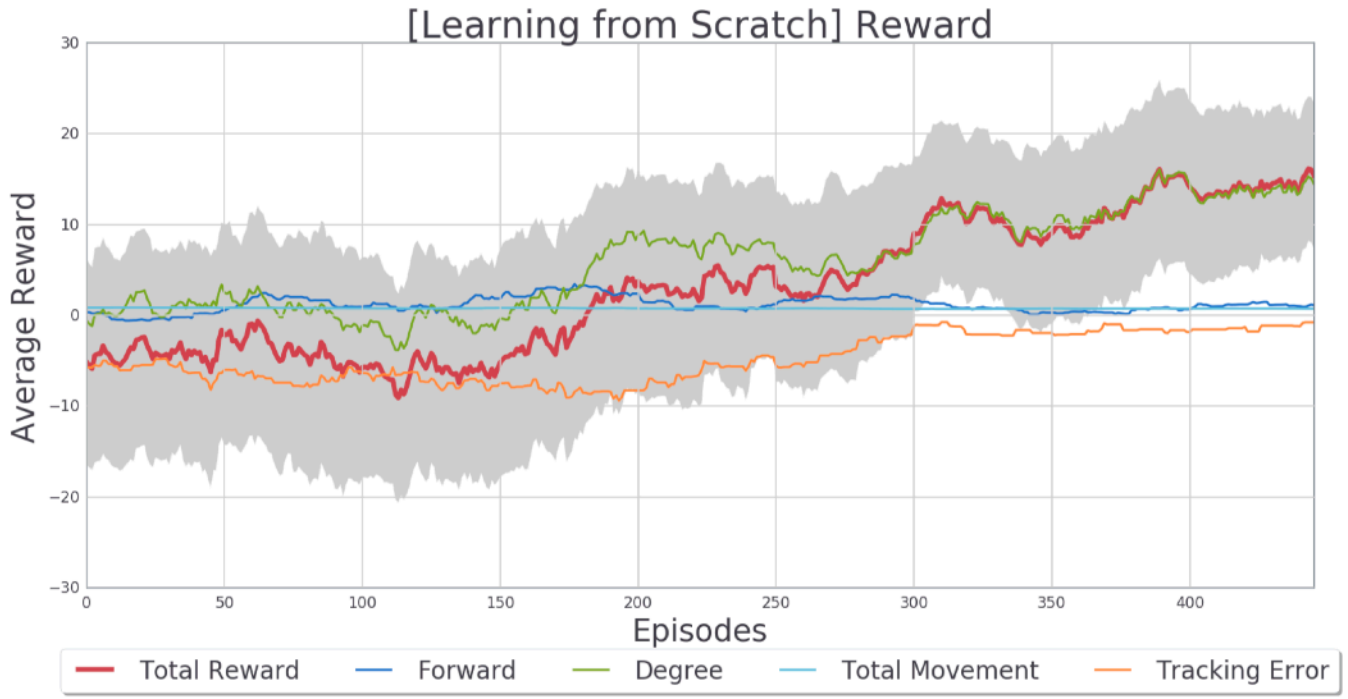
ROBOT INTELLIGENCE LAB

Turn Left	Go Straight	Turn Right
		
		

- Final motions of **Learning from Scratch** show that while Snapbot can turn left or right, it cannot go straight.

Experiments - Learning from Scratch

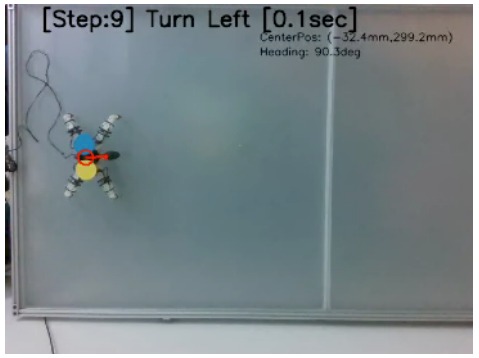
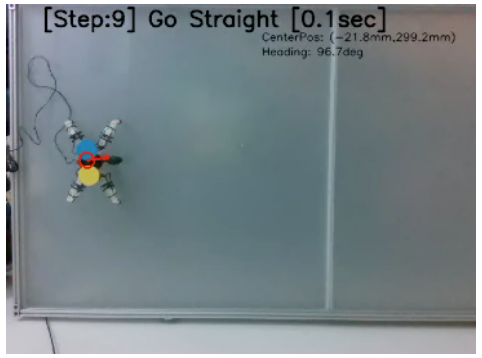
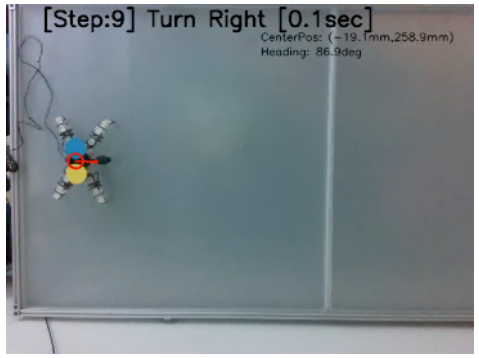
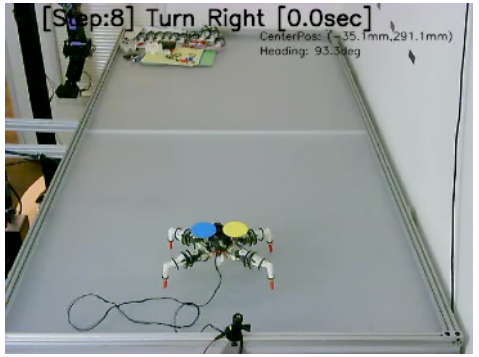
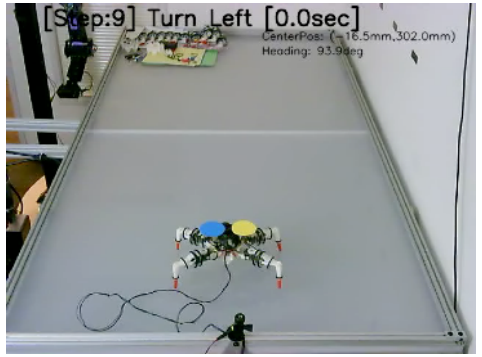
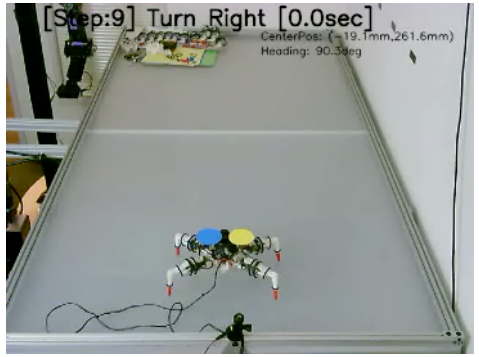
ROBOT INTELLIGENCE LAB



- Final motions of **Learning from Scratch** show that while Snapbot can turn left or right, it cannot go straight.

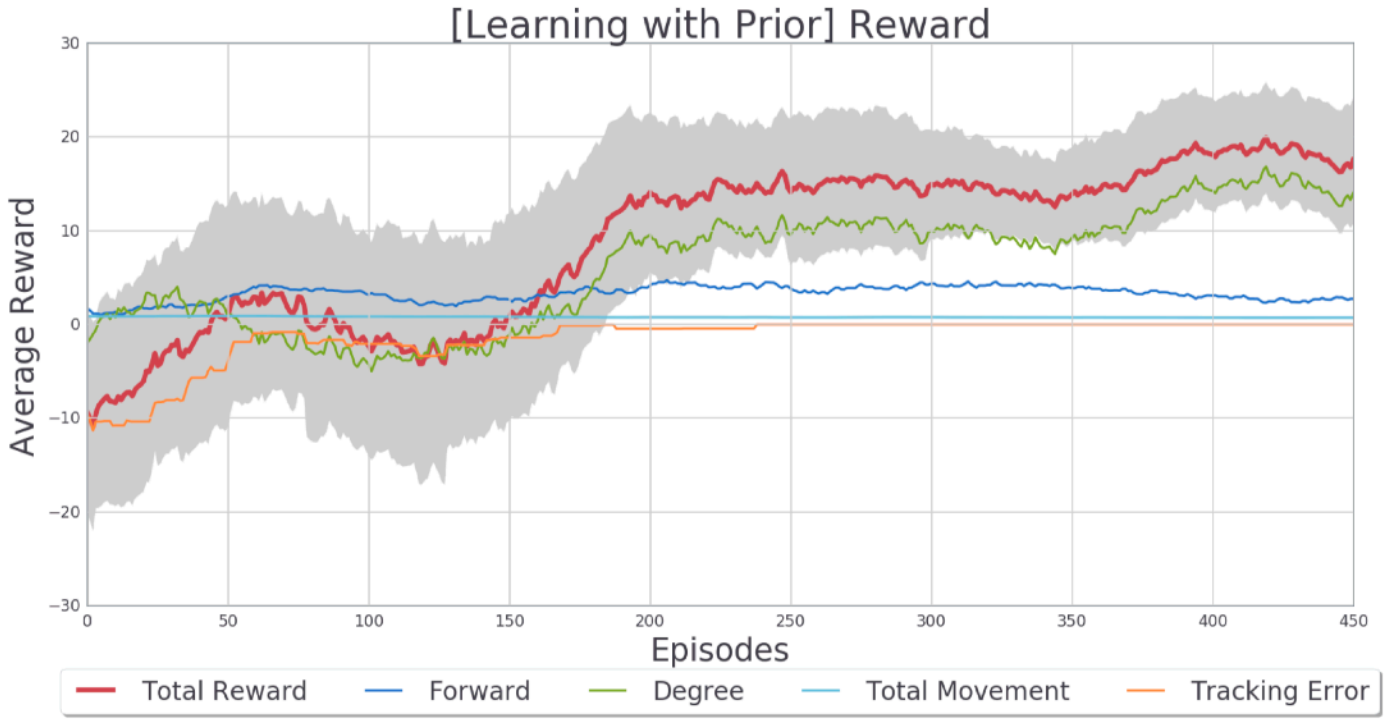
Experiments - Learning with Prior

ROBOT INTELLIGENCE LAB

Turn Left	Go Straight	Turn Right
		
		

- Final motions of **Learning with Prior** show that it can successfully go straight while turning left or right.

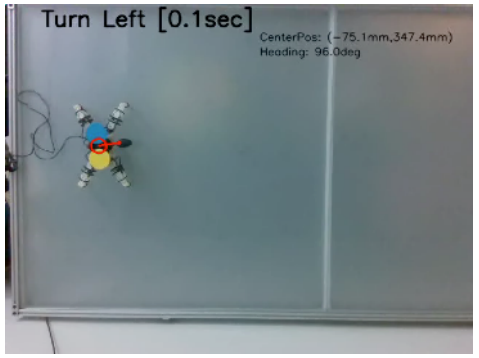
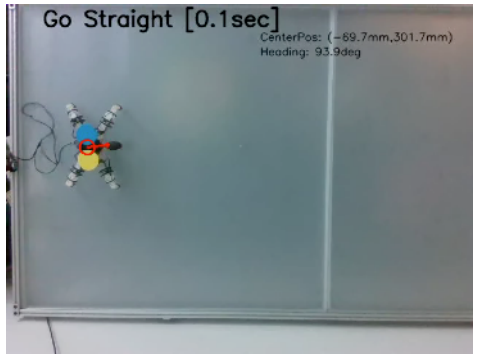
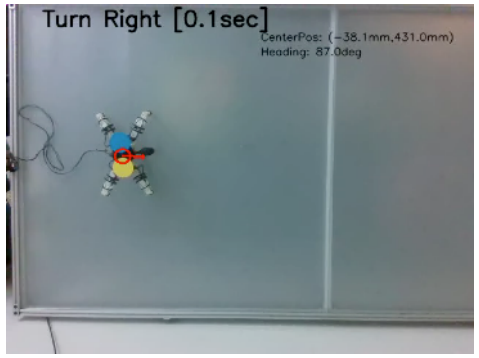
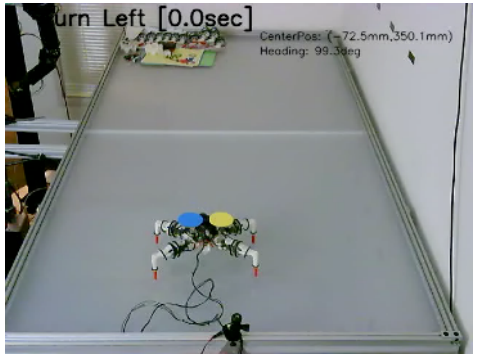
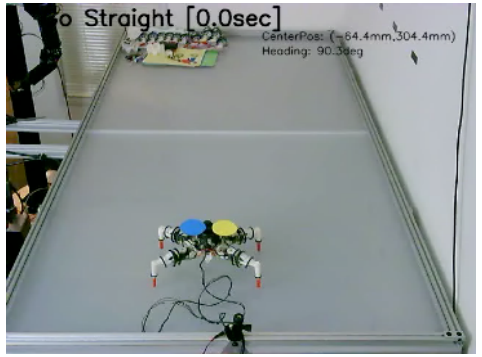
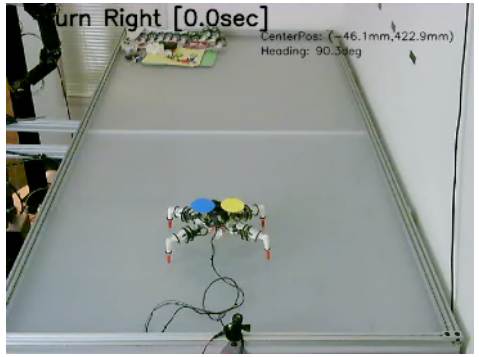
Experiments - Learning with Prior



- Final motions of **Learning with Prior** show that it can successfully go straight while turning left or right.

Experiments - Curriculum Learning

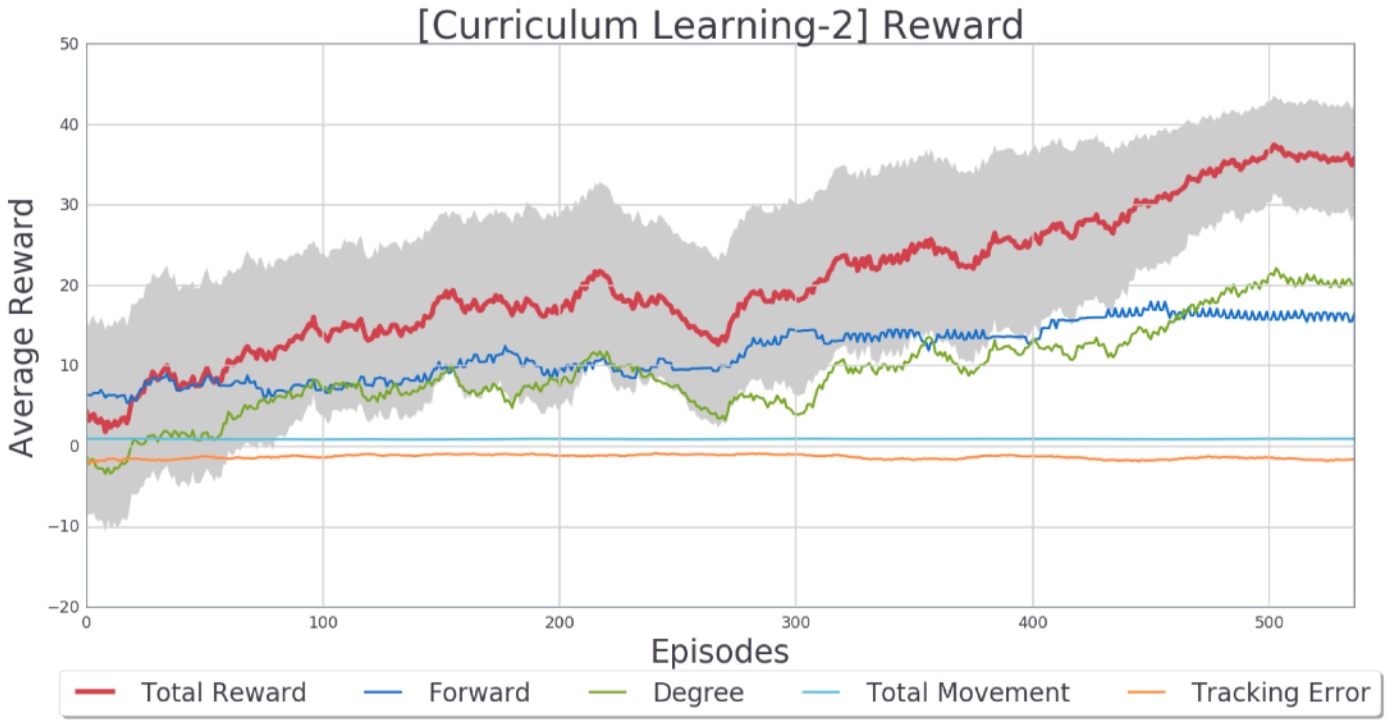
ROBOT INTELLIGENCE LAB

Turn Left	Go Straight	Turn Right
		
		

- Final motions of **Curriculum Learning** show that it can successfully go straight while turning left or right. Note that it can make more **diverse** motions!

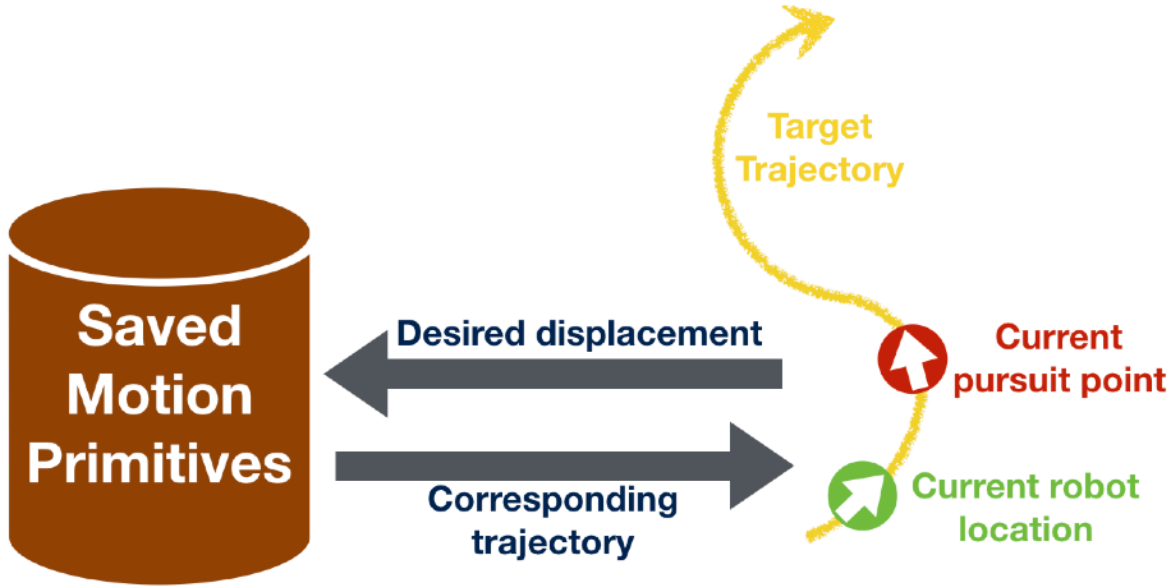
Experiments - Curriculum Learning

ROBOT INTELLIGENCE LAB



- Final motions of **Curriculum Learning** show that it can successfully go straight while turning left or right. Note that it can make more **diverse** motions!

Trajectory tracking experiments

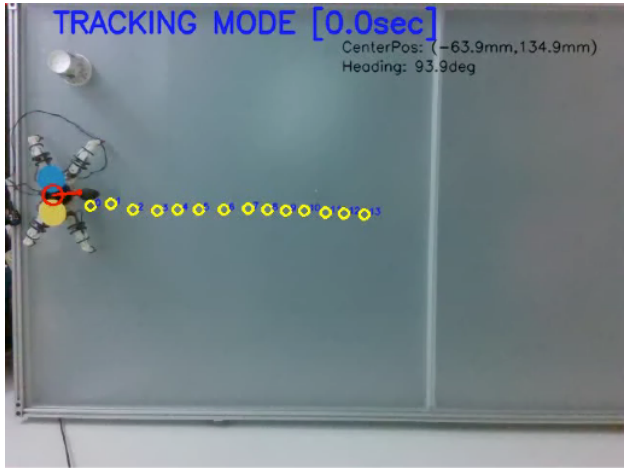


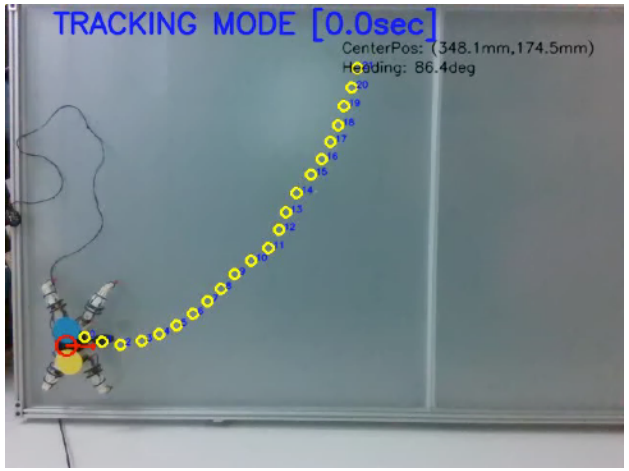
- We train nonparametric motion planners using trajectories collected from Learning with Prior and Curriculum Learning.

Trajectory tracking experiments

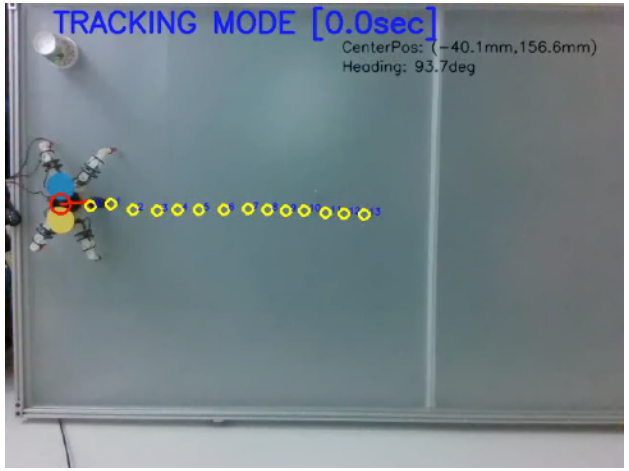
ROBOT INTELLIGENCE LAB

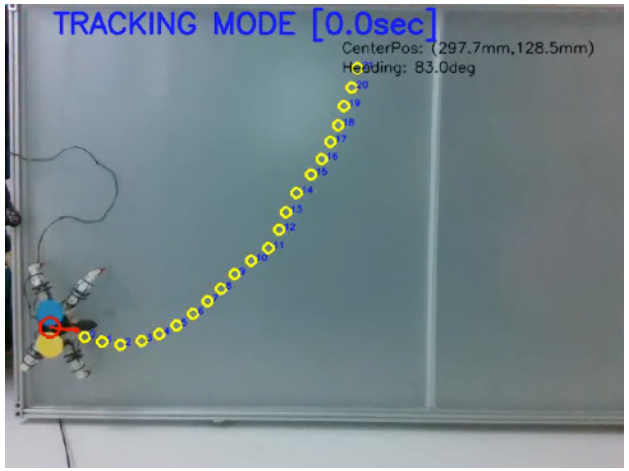
Learning with Prior Setting





Curriculum Learning

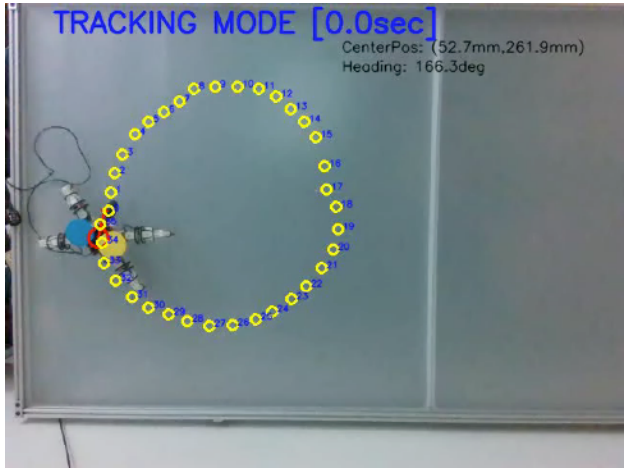


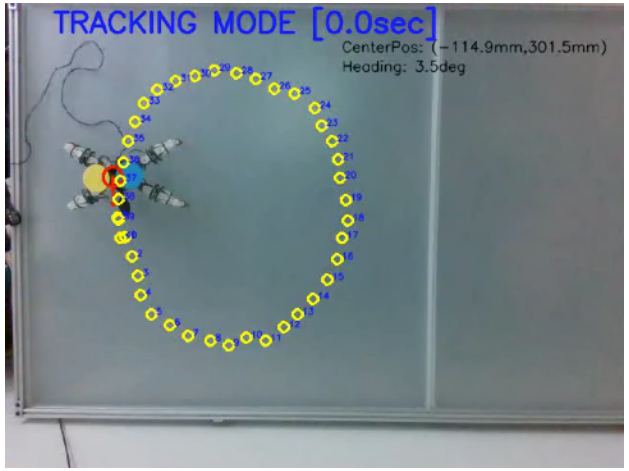


Trajectory tracking experiments

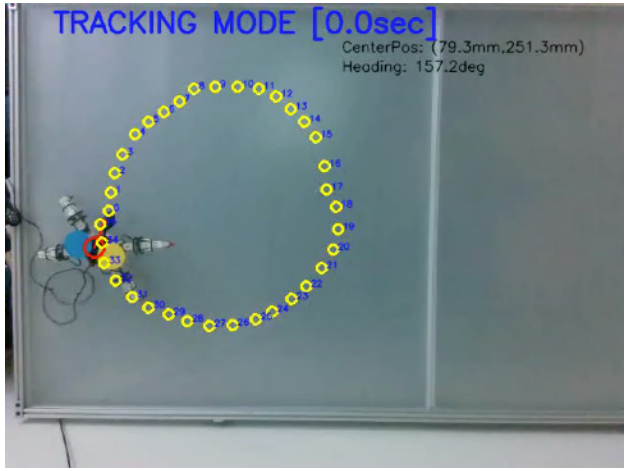
ROBOT INTELLIGENCE LAB

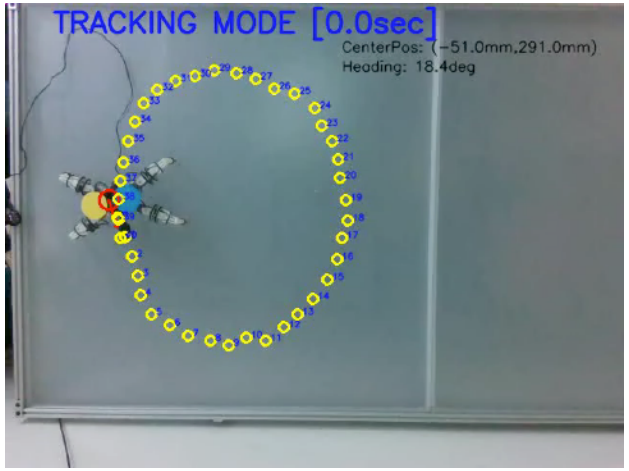
Learning with Prior Setting





Curriculum Learning



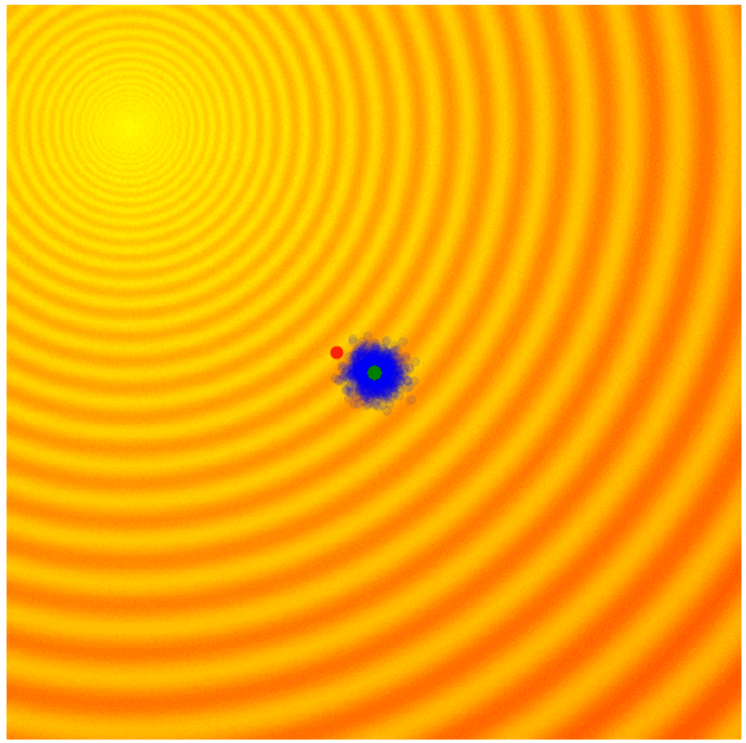


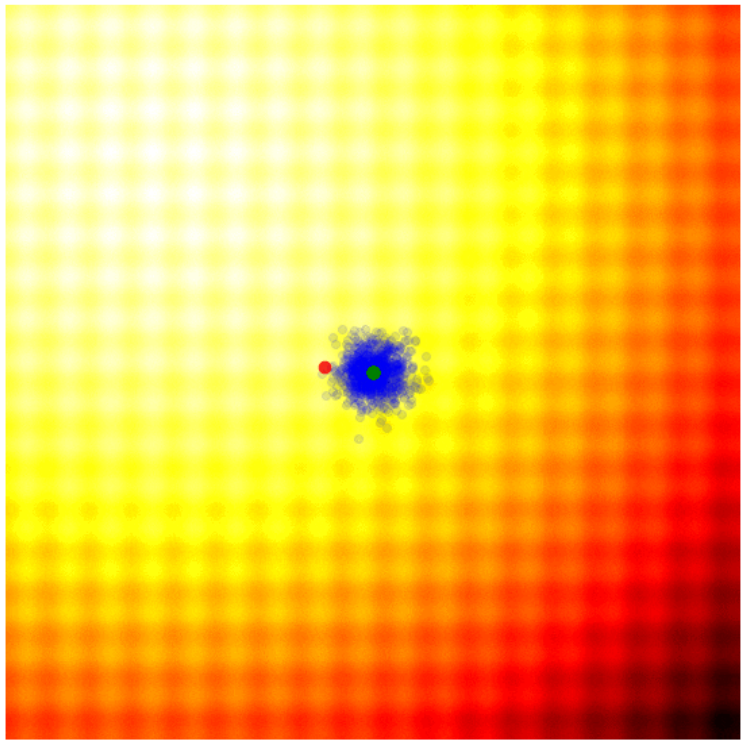
ROBOT INTELLIGENCE LAB

Cross Entropy Method (CEM)

Genetic Algorithm

ROBOT INTELLIGENCE LAB





- The green dot indicates the **elite population** from the previous generation, the blue dots are the offsprings to form the set of candidate solutions, and the red dot is the best solution so far.

https://blog.otoro.net/2017/10/29/visual-evolution-strategies/

Cross Entropy Method (CEM)

ROBOT INTELLIGENCE LAB

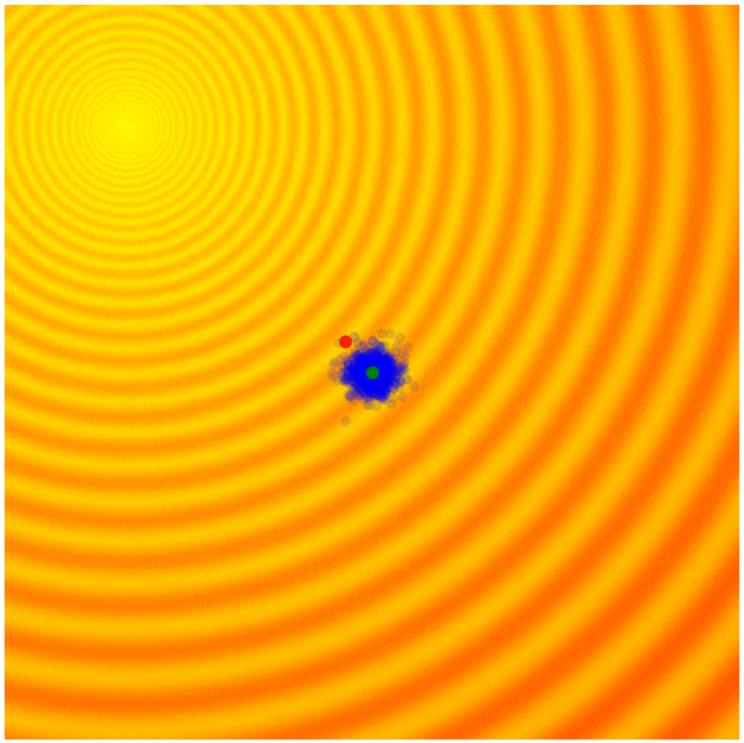
- Initialize $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$
- **for** iteration = 1, 2, ...
  - Collect n samples of $\theta_i \sim \mathcal{N}(\mu, \text{diag}(\sigma))$
  - Perform a noisy evaluation $R_i \sim \theta_i$
  - Select the top $p\,\%$ of samples (aka the **elite set**)
  - Fit a Gaussian distribution (with diagonal covariance) to the **elite set**, obtaining a new μ and σ
- **end for**
- Return the final μ

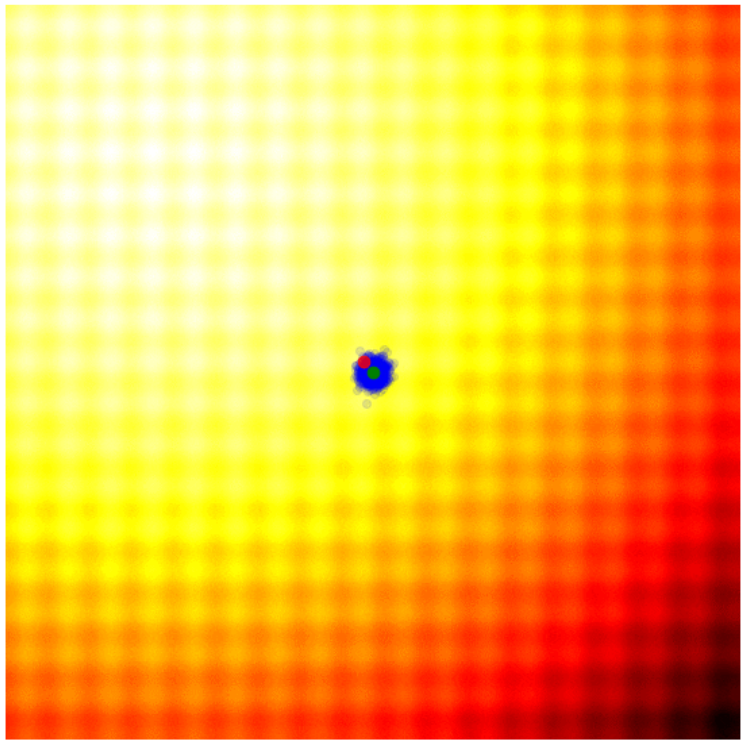
ROBOT INTELLIGENCE LAB

CMA-ES

"The CMA Evolution Strategy: A Tutorial," 2016

CMA-ES





- The green dot indicates the **elite population** from the previous generation, the blue dots are the offsprings to form the set of candidate solutions, and the red dot is the best solution so far.
- CMA-ES can **adaptively** increase or decrease the search space!

https://blog.otoro.net/2017/10/29/visual-evolution-strategies/

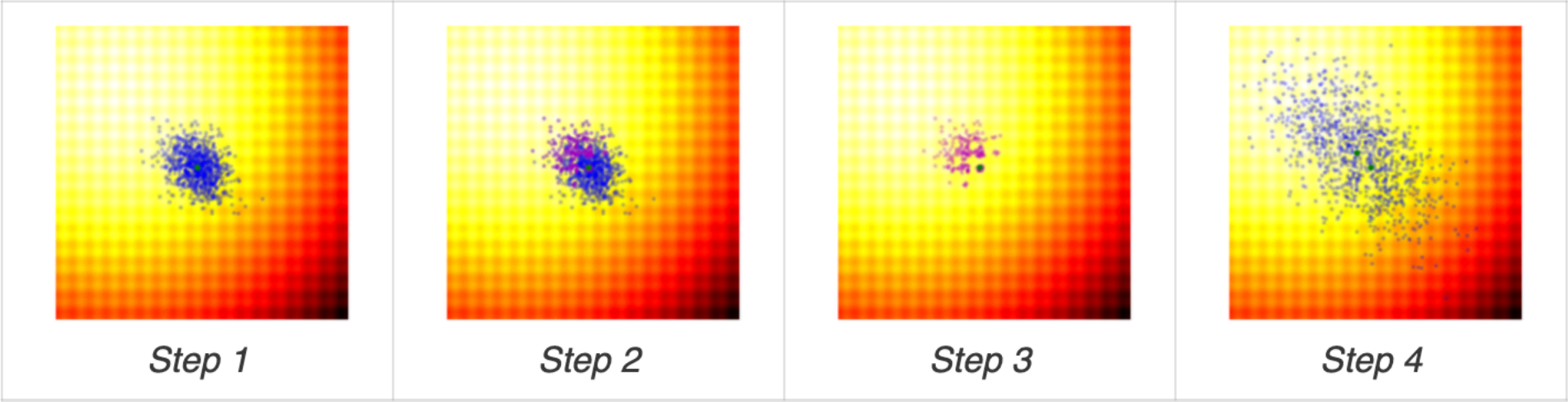
CMA-ES

ROBOT INTELLIGENCE LAB

- Initialize $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$
- **for** iteration g = 1, 2, ...
  - Collect n samples of $\theta_i \sim \mathcal{N}(\mu, \text{diag}(\sigma))$
  - Perform a noisy evaluation $R_i \sim \theta_i$
  - Select the top $p\,\%$ of samples (aka the **elite set**)
  - Fit $\mu^{(g)}$ to the **elite set** by $\mu^{(g)} = \frac{1}{N_{\text{best}}} \sum_{i}^{N_{\text{best}}} \theta_i$
  - The trick is to update the covariance using the previous mean $\mu^{(g-1)}$.
    - For 1-dimensional case, $\sigma_x^{2,(g+1)} = \frac{1}{N_{\text{best}}} \sum_{i=1}^{N_{\text{best}}} \left(x_i - \mu_x^{(g)}\right)^2$
- **end for**
- Return the final μ

https://blog.otoro.net/2017/10/29/visual-evolution-strategies/

The Effect of CMA



https://blog.otoro.net/2017/10/29/visual-evolution-strategies/

Augmented Random Search (ARS)

"Simple random search provides a competitive approach to reinforcement learning," 2018

Augmented Random Search

ROBOT INTELLIGENCE LAB

- Augmented Random Search (ARS) is a population-based, derivative-free reinforcement learning method.
  - Derivative-free methods such as a **cross-entropy method** (CEM) or **covariance matrix adaptation** (CMA) treat the return as as a black box function to be optimized in terms of the policy parameters.
  - From the current policy parameter θ^t, multiple parameters $\{\tilde{\theta}_i^t\}_{i=1}^N$ are sampled and the returns $\{\eta(\pi_{\tilde{\theta}_i^t})\}_{i=1}^N$ of the corresponding policies $\{\pi_{\tilde{\theta}_i^t}\}_{i=1}^N$ evaluated by rollouts.
- ARS utilizes a simple linear policy and searches over the space of matrices.

Basic Random Search

ROBOT INTELLIGENCE LAB

Algorithm 1 Basic Random Search (BRS)

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν

2: **Initialize:** $\theta_0 = \mathbf{0}$, and $j = 0$.

3: **while** ending condition not satisfied **do**

4: Sample $\delta_1, \delta_2, \ldots, \delta_N$ of the same size as θ_j, with i.i.d. standard normal entries.

5: Collect $2N$ rollouts of horizon H and their corresponding rewards using the policies

$$\pi_{j,k,+}(x) = \pi_{\theta_j + \nu\delta_k}(x) \quad \text{and} \quad \pi_{j,k,-}(x) = \pi_{\theta_j - \nu\delta_k}(x),$$

Positively perturbed policy

Negatively perturbed policy

with $k \in \{1, 2, \ldots, N\}$.

6: Make the update step:

Stepsize $\frac{\alpha}{N}$ $\sum_{k=1}^{N}$ Score of the perturbation $[r(\pi_{j,k,+}) - r(\pi_{j,k,-})]$ Perturbation δ_k

$$\theta_{j+1} = \theta_j + \frac{\alpha}{N} \sum_{k=1}^{N} [r(\pi_{j,k,+}) - r(\pi_{j,k,-})] \delta_k.$$

7: $j \leftarrow j + 1$.

8: **end while**

Augmented Random Search

ROBOT INTELLIGENCE LAB

- Drawbacks of Basic Random Search
  - Input normalization is often necessary for high-dimensional inputs.
  - Not all perturbations (directions) are useful.

Augmented Random Search

ROBOT INTELLIGENCE LAB

Algorithm 2 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, number of top-performing directions to use b ($b < N$ is allowed only for **V1-t** and **V2-t**)

Use top b search directions.

2: **Initialize:** $M_0 = \mathbf{0} \in \mathbb{R}^{p \times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n \times n}$, $j = 0$.

3: **while** ending condition not satisfied **do**

4: Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p \times n}$ with i.i.d. standard normal entries.

5: Collect $2N$ rollouts of horizon H and their corresponding rewards using the $2N$ policies

$$\textbf{V1:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$$

$$\textbf{V2:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\,\mathrm{diag}\,(\Sigma_j)^{-1/2}\,(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\,\mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$$

Input normalization

for $k \in \{1, 2, \ldots, N\}$.

6: Sort the directions δ_k by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$ denote by $\delta_{(k)}$ the k-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.

7: Make the update step:

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$$

where σ_R is the standard deviation of the $2b$ rewards used in the update step.

8: **V2** : Set μ_{j+1}, Σ_{j+1} to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[2]

9: $j \leftarrow j + 1$

10: **end while**


ROBOT INTELLIGENCE LAB