

Reinforcement Learning

Lecture 4. Policy-based Methods

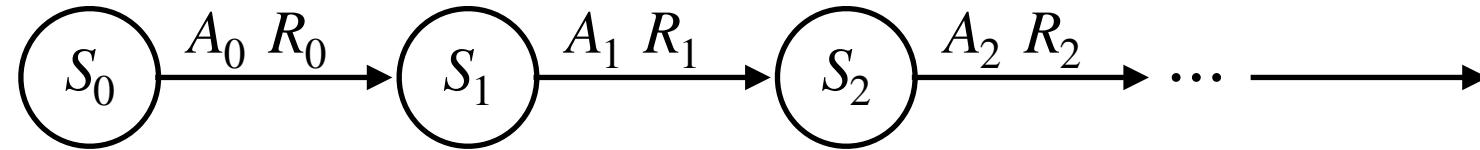
Sungjoon Choi, Korea University

Content

- Summary of Model-based and Model-free methods
- ~~Proving Policy Gradient Theorem~~
- Trust Region Policy Optimization (**TRPO**)
- Proximal Policy Optimization (**PPO**)
- Generalized Advantage Estimation (**GAE**)
- Soft Actor-Critic (**SAC**)

Summary of Model-based and Model-free Methods

Markov Decision Process



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

Markov Decision Process

- 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)
 - $P(s'|s, a) = P(S_{t+1} = s' | S_t = s, A_t = a)$
 - $P_{sas'} = P(s'|s, a)$
 - A reward function
 - $r(s) = \mathbb{E}[R_{t+1} | S_t = s]$
 - It depends on both state and action.
 - An initial state distribution d

Return

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

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

- Discount factor γ :



1

Worth Now



γ

Worth Next Step



γ^2

Worth In Two Steps

(State) Value Function

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

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots | S_t = s] \end{aligned}$$

(State-Action) Value Function

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

$$\begin{aligned} Q(s, a) &= \mathbb{E}[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots | S_t = s, A_t = a] \end{aligned}$$

Policy

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

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

- 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.

Optimality

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

Optimal Value and Policy

- 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

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

Value Iteration

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

Value Iteration

- 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

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

- 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.

- Bellman Optimality Equation

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

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

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

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

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

Q -Value Iteration

- 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).

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

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

• Bellman Equation

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

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

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

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

• Bellman Optimality Equation

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

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

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

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

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

Q -Value Iteration

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

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

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

- We now have an explicit form of the policy:

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

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

Policy Iteration

- **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:

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

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

- One-step lookahead:

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

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

Limitation of Model-based Methods

Value Iteration

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

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

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

- We now have an explicit form of the policy:

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

Policy Iteration

- 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:

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

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

- One-step lookahead:

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

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

What is the common **key limitation** of both methods?

Both methods require state transition probability $P(s' | s, a)$.

Model-free Methods

- The goal of a **model-based method** can be written as

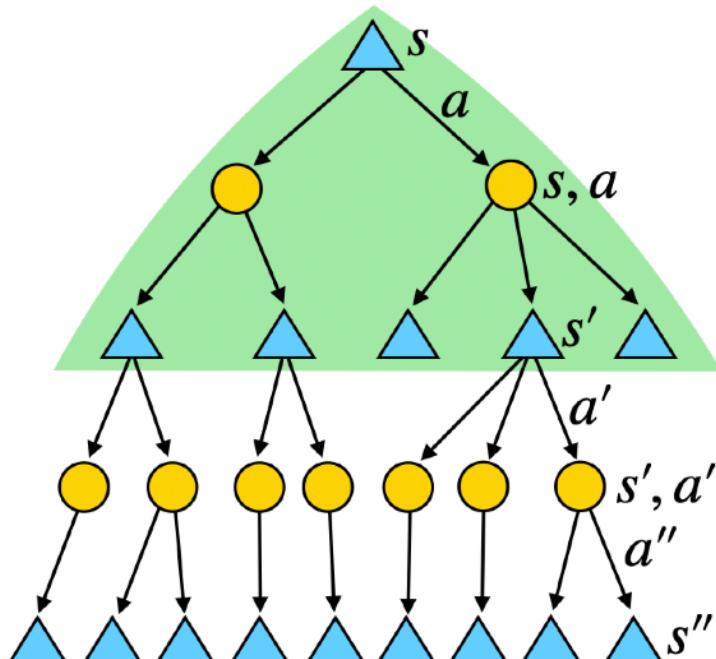
Given (S, A, R, P, d) , find π such that $\mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R_t | \pi \right]$ is maximized.

- The goal of a **model-free method** can be written as

Given (S, A, R) , find π such that $\mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R_t | \pi \right]$ is maximized.

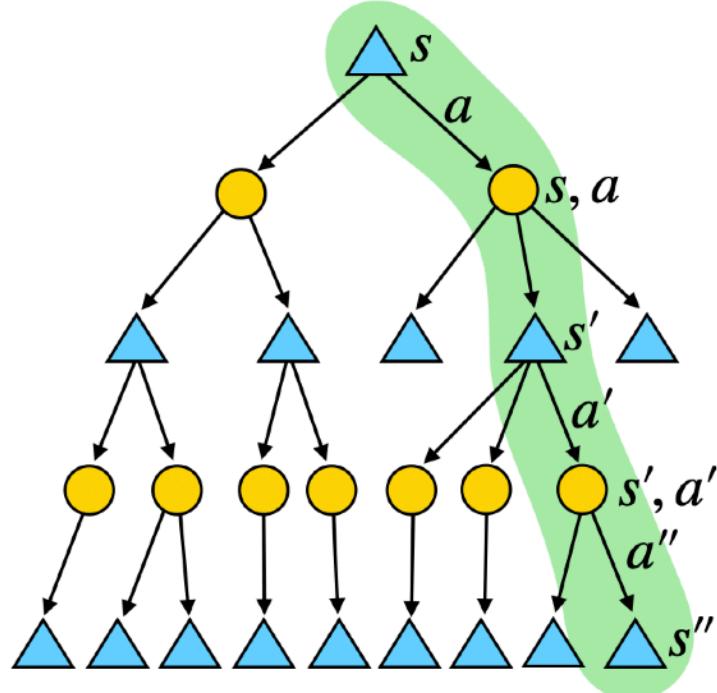
- Note that the **state transition probability P** and the **initial state distribution d** are omitted in RL.
- Hence, the main challenge is to approximate the **expectation** part!

Model-free Methods



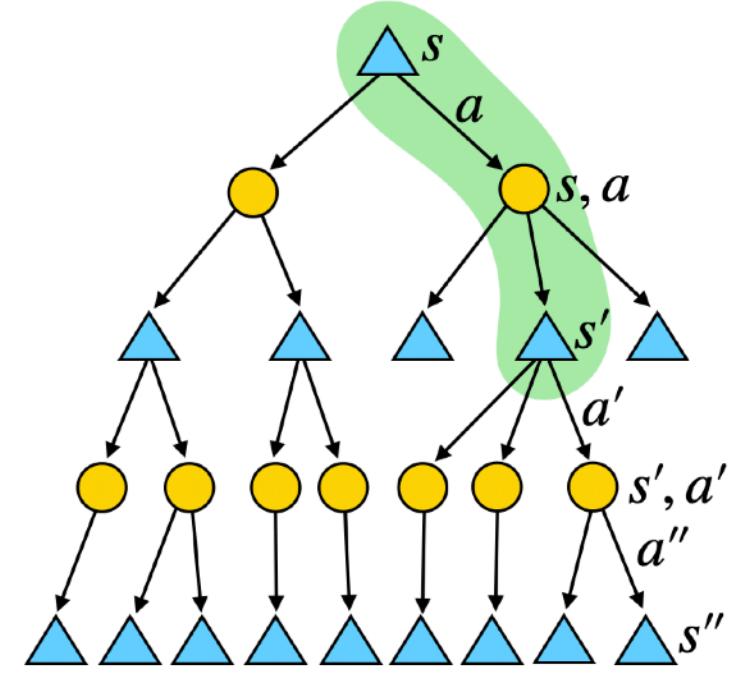
Dynamic Programming

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



Monte-Carlo Learning

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



Temporal Difference Learning

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

SARSA

- **Step 1: Policy evaluation**

- Estimate $\widehat{Q}(s, a)$ from samples using (S, A, R, S', A')

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

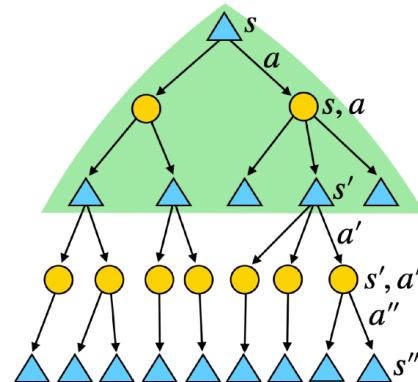
- **Step 2: Policy improvement**

- 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

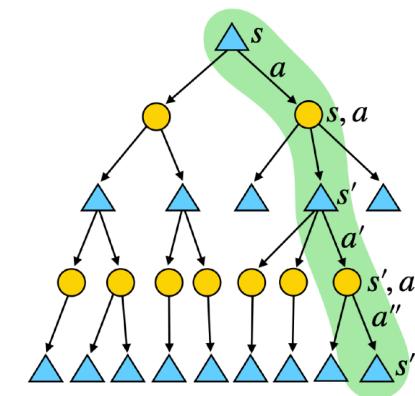
- Initialize $Q(s, a)$
- Repeat (for each episodes)
 - Sample an initial state s_0 .
 - Sample a_0 from an ϵ -greedy policy π .
 - Repeat (for each time step t)
 - Get reward r_{t+1} and next state s_{t+1} .
 - Sample a_{t+1} from the ϵ -greedy policy π .
 - Update $\widehat{Q}(s, a)$ using $(s_t, a_t,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

DP vs. MC vs. TD vs. SARSA



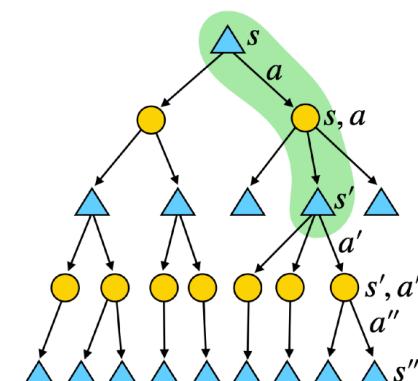
Dynamic Programming

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



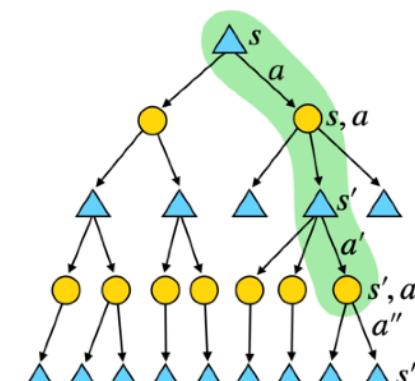
Monte-Carlo Learning

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



Temporal Difference Learning

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



SARSA

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

Q-Learning

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

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

- 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

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

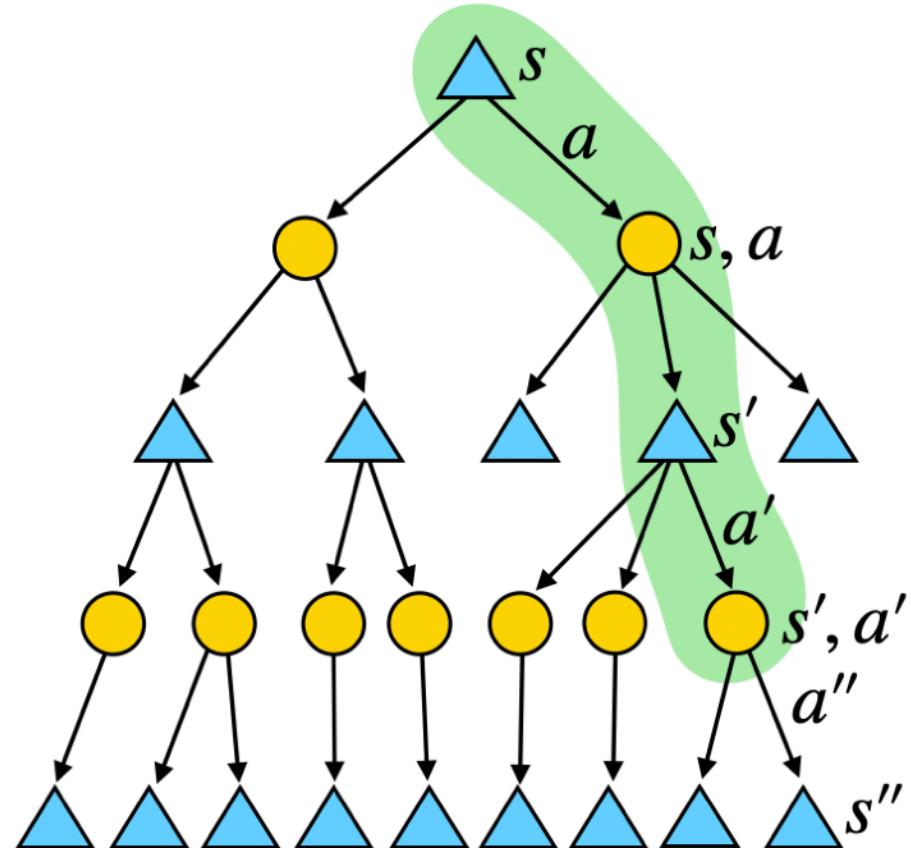
$$\widehat{Q}(s, a) \leftarrow \widehat{Q}(s, a) + \alpha \left(r(s, a, s') + \gamma \max_{a'} \widehat{Q}(s', a') - \widehat{Q}(s, a) \right)$$

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.

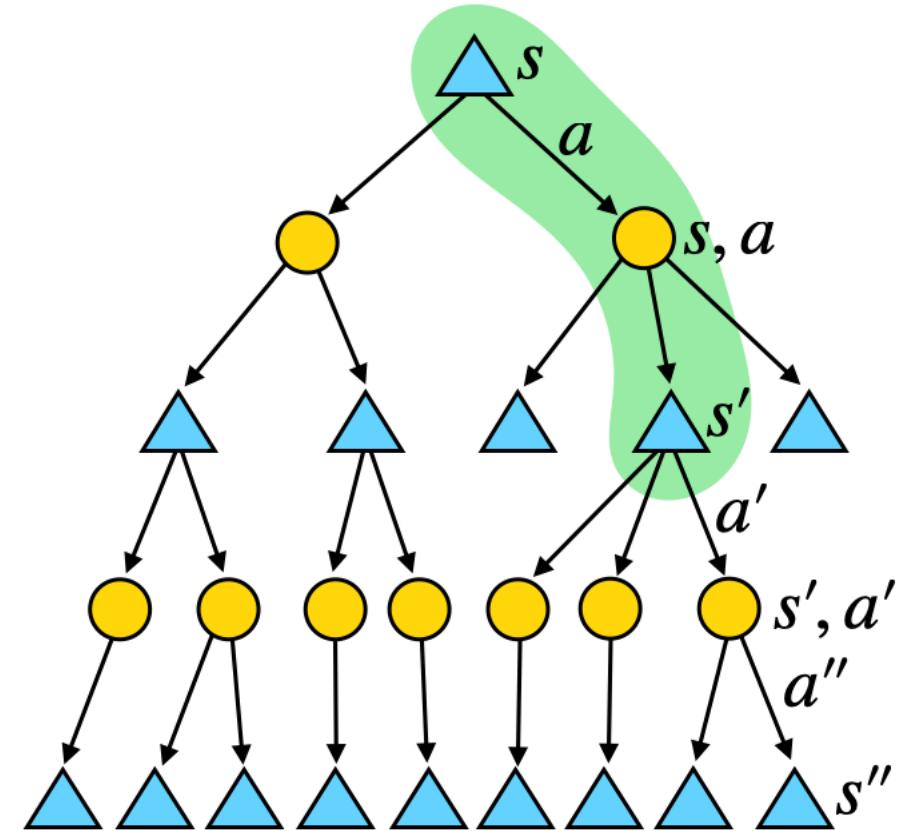
Q-Learning

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



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

Deep Q Network (DQN)

- Recall the original Q-Learning

$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha \left(\boxed{r(s, a, s') + \gamma \max_{a'} Q_k(s', a')} - \boxed{Q_k(s, a)} \right)$$

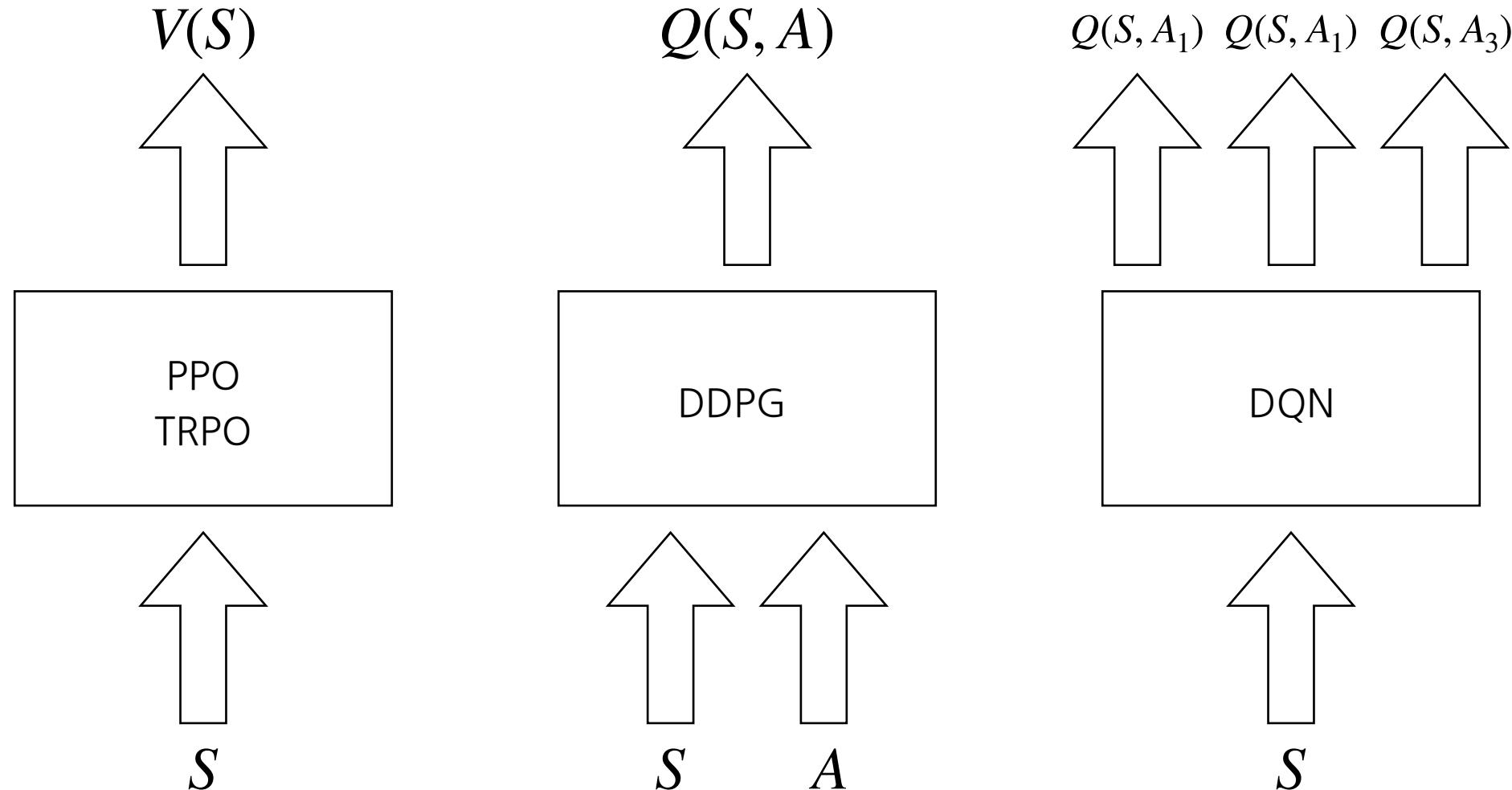
Target Prediction

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

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

Target Prediction

Deep Q Network (DQN)



Deep Q Network (DQN)

- (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)

- (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 parameters
- θ : current parameter to be updated

- 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())

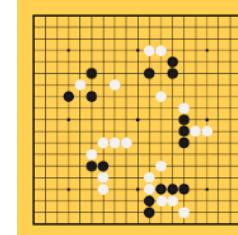
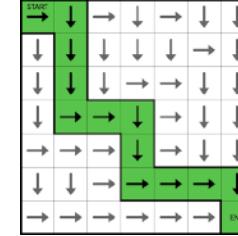
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()

    return loss
  
```

Rule of Thumb



- Discrete (and small) state space & discrete action space
 - E.g., grid world
 - In this case, the state-transition model can easily be defined.
 - Use **value iteration** or **policy iteration**.
 - Discrete (and large) state space & discrete action space
 - E.g., Go
 - In this case, the state-transition model is cumbersome to be defined
 - Use **Q-learning**.
 - Continuous state space & discrete action space
 - E.g., Atari games
 - In this case, the state-transition model is impossible to be defined.
 - Use **DQN**.
 - Continuous state space & continuous action space
 - E.g., Robotics
 - In this case, the state-transition model is impossible to be defined.
 - Use **PPO** or **SAC**.



Policy Gradient Theorem

Limitations of Previous Methods

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

Policy Optimization

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

Policy Optimization

- Policy gradient methods cast reinforcement learning into an optimization problem.
- Find θ that maximizes the return:

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

Policy Optimization

- Policy gradient methods cast reinforcement learning into an optimization problem.
- Find θ that maximizes the return:

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

- 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

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

- Policy Gradient Theorem:

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

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

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

State Visitation

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

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

State Visitation

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

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

- $\rho_{\pi_\theta}(s)$ 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)

- 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)

- 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 $\nabla_\theta V_{\pi_\theta}(s_1)$:

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

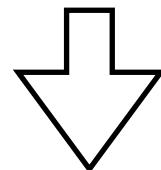
$$\nabla_\theta V_{\pi_\theta}(s_1) = \sum_a \nabla_\theta \pi_\theta(a | s_1) Q_{\pi_\theta}(s_1, a) + \pi_\theta(a | s_1) \nabla_\theta Q_{\pi_\theta}(s_1, a)$$

Proof of Policy Gradient (2/10)

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

Proof of Policy Gradient (2/10)

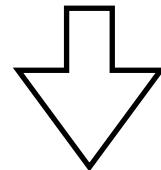
$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) + \pi_{\theta}(a | s_1) \nabla_{\theta} Q_{\pi_{\theta}}(s_1, a)$$



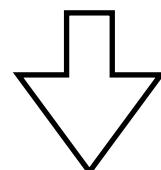
$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) + \pi_{\theta}(a | s_1) \nabla_{\theta} \left[r(s_1, a) + \gamma \sum_{s'} V_{\pi_{\theta}}(s') P(s' | s_1, a) \right]$$

Proof of Policy Gradient (2/10)

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



$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) + \pi_{\theta}(a | s_1) \nabla_{\theta} \left[r(s_1, a) + \gamma \sum_{s'} V_{\pi_{\theta}}(s') P(s' | s_1, a) \right]$$



$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) + \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \nabla_{\theta} V_{\pi_{\theta}}(s') P(s' | s_1, a) \right]$$

Proof of Policy Gradient (3/10)

$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) + \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \nabla_{\theta} V_{\pi_{\theta}}(s') P(s' | s_1, a) \right]$$

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

- By plugging in:

$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) + \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') + \pi_{\theta}(a' | s') \left[\gamma \sum_{s''} \nabla_{\theta} V_{\pi_{\theta}}(s'') P(s'' | s', a') \right] \right] P(s' | s_1, a) \right]$$

Proof of Policy Gradient (4/10)

$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_a \left[\nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) + \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') + \pi_{\theta}(a' | s') \left[\gamma \sum_{s''} \nabla_{\theta} V_{\pi_{\theta}}(s'') P(s'' | s', a') \right] \right] P(s' | s_1, a) \right] \right]$$

- $\nabla_{\theta} V_{\pi_{\theta}}(s_1)$ contains three terms

$$\begin{aligned} & \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) \\ & + \sum_a \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') \right] P(s' | s_1, a) \right] \\ & + \sum_a \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \pi_{\theta}(a' | s') \left[\gamma \sum_{s''} \nabla_{\theta} V_{\pi_{\theta}}(s'') P(s'' | s', a') \right] \right] P(s' | s_1, a) \right] \end{aligned}$$

Proof of Policy Gradient (5/10)

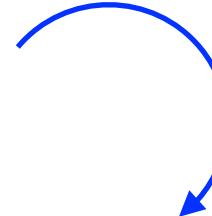
- If we focus on the **first** term

$$\sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a)$$

(orange box)

$$\sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s, a) P_{\pi_{\theta}}(S_0 = s | S_0 = s_1)$$

(orange box)



$\nabla_{\theta} V_{\pi_{\theta}}(s_1)$ contains three terms

$$\begin{aligned} & \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) \\ & + \sum_a \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') \right] P(s' | s_1, a) \right] \\ & + \sum_a \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_{\theta} V_{\pi_{\theta}}(s') \left[\gamma \sum_{s''} \nabla_{\theta} V_{\pi_{\theta}}(s'') P(s'' | s', a') \right] \right] P(s' | s_1, a) \right] \end{aligned}$$

$P_{\pi_{\theta}}(S_0 = s' | S_0 = s_1)$ is nonzero only when $s' = s_1$

Proof of Policy Gradient (6/10)

- If we focus on the **second** term

$\nabla_{\theta} V_{\pi_{\theta}}(s_1)$ contains three terms

$$\begin{aligned} & \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s_1, a) \\ & + \sum_a \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') \right] P(s' | s_1, a) \right] \\ & + \sum_a \pi_{\theta}(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_{\theta} V_{\pi_{\theta}}(s') \left[\gamma \sum_{s''} \nabla_{\theta} V_{\pi_{\theta}}(s'') P(s'' | s', a') \right] \right] P(s' | s_1, a) \right] \end{aligned}$$

$$\sum_a \pi_{\theta}(a | s_1) \gamma \sum_{s'} \sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') P(s' | s_1, a)$$

$$\sum_{s'} \sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') \gamma \sum_a P(s' | s_1, a) \pi_{\theta}(a | s_1)$$

$$\sum_{s'} \sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') \gamma P_{\pi_{\theta}}(S_1 = s' | S_0 = s_1)$$

Rearrange

State Transition Probability

Proof of Policy Gradient (7/10)

- If we focus on the **third** term

$$\sum_a \pi_\theta(a | s_1) \gamma \sum_{s'} \sum_{a'} \pi_\theta(a' | s') \gamma \sum_{s''} \nabla_\theta V_{\pi_\theta}(s'') P(s'' | s', a') P(s' | s_1, a)$$

Rearrange

$$\sum_{s''} \nabla_\theta V_{\pi_\theta}(s'') \gamma^2 \sum_a \sum_{s'} \sum_{a'} \pi_\theta(a | s_1) \pi_\theta(a' | s') P(s'' | s', a') P(s' | s_1, a)$$

State Transition Probability

$$\sum_{s''} \nabla_\theta V_{\pi_\theta}(s'') \gamma^2 P_{\pi_\theta}(S_2 = s'' | S_0 = s_1)$$

$\nabla_\theta V_{\pi_\theta}(s_1)$ contains three terms

$$\begin{aligned} & \sum_a \nabla_\theta \pi_\theta(a | s_1) Q_{\pi_\theta}(s_1, a) \\ & + \sum_a \pi_\theta(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_\theta \pi_\theta(a' | s') Q_{\pi_\theta}(s', a') \right] P(s' | s_1, a) \right] \\ & + \sum_a \pi_\theta(a | s_1) \left[\gamma \sum_{s'} \left[\sum_{a'} \nabla_\theta \pi_\theta(a' | s') \left[\gamma \sum_{s''} \nabla_\theta V_{\pi_\theta}(s'') P(s'' | s', a') \right] \right] P(s' | s_1, a) \right] \end{aligned}$$

Proof of Policy Gradient (8/10)

- Substituting the first, second and third terms:

$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s_1) Q_{\pi_{\theta}}(s, a) P_{\pi_{\theta}}(S_0 = s | S_0 = s_1) + \sum_{s'} \sum_{a'} \nabla_{\theta} \pi_{\theta}(a' | s') Q_{\pi_{\theta}}(s', a') \gamma P_{\pi_{\theta}}(S_1 = s' | S_0 = s_1) + \sum_{s''} \nabla_{\theta} V_{\pi_{\theta}}(s'') \gamma^2 P_{\pi_{\theta}}(S_2 = s'' | S_0 = s_1)$$

↓
Mathematical Induction

$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s) Q_{\pi_{\theta}}(s, a) \left(P_{\pi_{\theta}}(S_0 = s | S_0 = s_1) + \gamma P_{\pi_{\theta}}(S_1 = s | S_0 = s_1) + \gamma^2 P_{\pi_{\theta}}(S_2 = s | S_0 = s_1) + \gamma^3 P_{\pi_{\theta}}(S_3 = s | S_0 = s_1) + \dots \right)$$

$$\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s) Q_{\pi_{\theta}}(s, a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s | S_0 = s_1)$$

Proof of Policy Gradient (9/10)

• Plugging in $\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s) Q_{\pi_{\theta}}(s, a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s | S_0 = s_1)$ to $\nabla_{\theta} \eta(\pi_{\theta}) = \sum_s d(s) \nabla_{\theta} V_{\pi_{\theta}}(s)$

Proof of Policy Gradient (9/10)

- Plugging in $\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s) Q_{\pi_{\theta}}(s, a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s | S_0 = s_1)$ to $\nabla_{\theta} \eta(\pi_{\theta}) = \sum_s d(s) \nabla_{\theta} V_{\pi_{\theta}}(s)$

$$\nabla_{\theta} \eta(\pi_{\theta}) = \sum_s d(s) \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s' | S_0 = s)$$

Proof of Policy Gradient (9/10)

- Plugging in $\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s) Q_{\pi_{\theta}}(s, a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s | S_0 = s_1)$ to $\nabla_{\theta} \eta(\pi_{\theta}) = \sum_s d(s) \nabla_{\theta} V_{\pi_{\theta}}(s)$

$$\begin{aligned}
 \nabla_{\theta} \eta(\pi_{\theta}) &= \sum_s d(s) \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s' | S_0 = s) \\
 &= \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t \sum_s P_{\pi_{\theta}}(S_t = s' | S_0 = s) d(s)
 \end{aligned}$$

Rearrange

Proof of Policy Gradient (9/10)

• Plugging in $\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s) Q_{\pi_{\theta}}(s, a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s | S_0 = s_1)$ to $\nabla_{\theta} \eta(\pi_{\theta}) = \sum_s d(s) \nabla_{\theta} V_{\pi_{\theta}}(s)$

$$\begin{aligned}
 \nabla_{\theta} \eta(\pi_{\theta}) &= \sum_s d(s) \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s' | S_0 = s) \\
 &= \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t \sum_s P_{\pi_{\theta}}(S_t = s' | S_0 = s) d(s) \\
 &= \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s')
 \end{aligned}$$

By definition

Proof of Policy Gradient (9/10)

- Plugging in $\nabla_{\theta} V_{\pi_{\theta}}(s_1) = \sum_s \sum_a \nabla_{\theta} \pi_{\theta}(a | s) Q_{\pi_{\theta}}(s, a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s | S_0 = s_1)$ to $\nabla_{\theta} \eta(\pi_{\theta}) = \sum_s d(s) \nabla_{\theta} V_{\pi_{\theta}}(s)$

$$\begin{aligned}
 \nabla_{\theta} \eta(\pi_{\theta}) &= \sum_s d(s) \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s' | S_0 = s) \\
 &= \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t \sum_s P_{\pi_{\theta}}(S_t = s' | S_0 = s) d(s) \\
 &= \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s') \\
 &= \frac{1}{1 - \gamma} \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \rho_{\pi_{\theta}}(s')
 \end{aligned}$$

$\rho_{\pi_{\theta}}(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta}}(S_t = s)$

Proof of Policy Gradient (10/10)

$$\begin{aligned}\nabla_{\theta} \eta(\pi_{\theta}) &= \frac{1}{1 - \gamma} \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \rho_{\pi_{\theta}}(s') \\ &\propto \mathbb{E}_{s \sim \rho_{\pi_{\theta}}} \left[\sum_a \nabla_{\theta} \pi_{\theta}(a | s) Q_{\pi_{\theta}}(s, a) \right]\end{aligned}$$

- 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

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

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

Log Ratio Trick

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

- However, we should summate over all possible states and actions.
- We can use the log ratio trick to overcome this issue.

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

- To summarize

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

Log Ratio Trick

$$\begin{aligned}
 \nabla_{\theta} \eta(\pi_{\theta}) &= \frac{1}{1-\gamma} \sum_{s'} \sum_a \nabla_{\theta} \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \rho_{\pi_{\theta}}(s') \\
 &= \frac{1}{1-\gamma} \sum_{s'} \sum_a \pi_{\theta}(a | s') \frac{\nabla_{\theta} \pi_{\theta}(a | s')}{\pi_{\theta}(a | s')} Q_{\pi_{\theta}}(s', a) \rho_{\pi_{\theta}}(s') \\
 &= \frac{1}{1-\gamma} \sum_{s'} \sum_a \pi_{\theta}(a | s') \nabla_{\theta} \log \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \rho_{\pi_{\theta}}(s') \\
 &= \frac{1}{1-\gamma} \mathbb{E}_{a \sim \pi_{\theta}, s \sim \rho_{\pi}(S)} \left[\nabla_{\theta} \log \pi_{\theta}(a | s') Q_{\pi_{\theta}}(s', a) \right] \\
 &\approx \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q_{\pi_{\theta}}(s_t, a_t)
 \end{aligned}$$

- To summarize

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

Off-Policy vs. On-Policy Methods

- 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**?

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

Trust Region Policy Optimization (TRPO)

"Trust Region Policy Optimization", 2015

PG as an optimization problem

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

PG as an optimization problem

- Policy-based reinforcement learning is an optimization problem.
- The goal is to find θ that maximizes

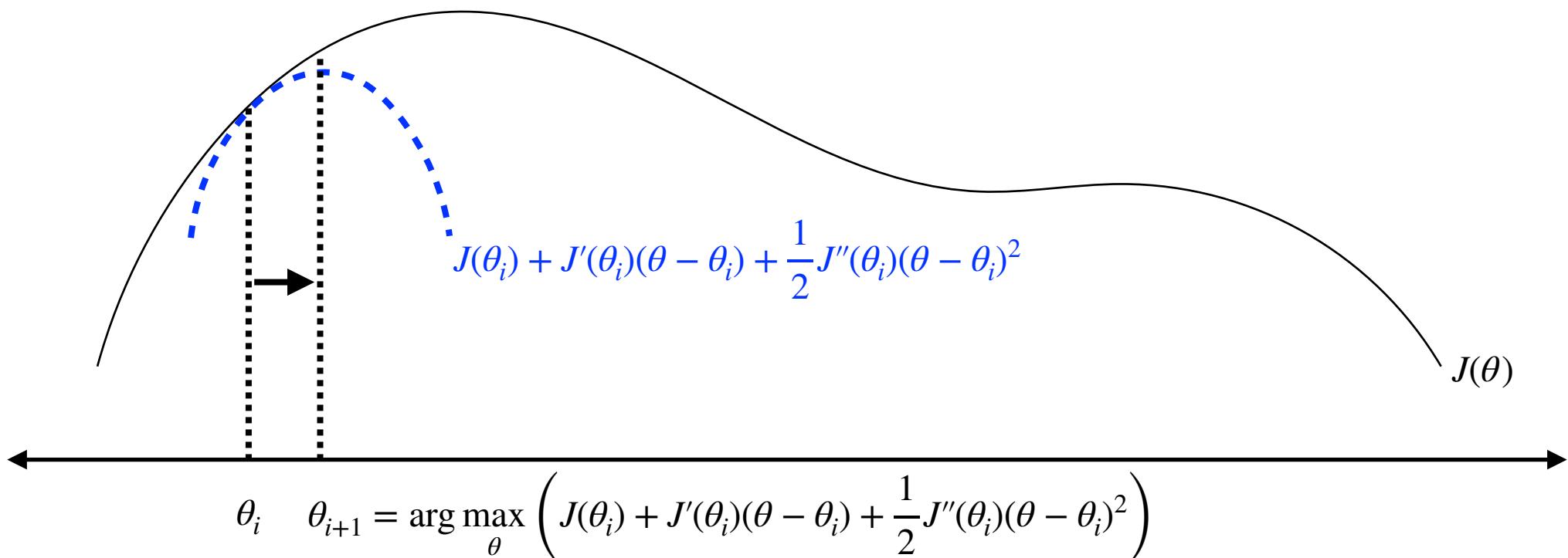
$$\eta(\pi_\theta) = \mathbb{E}_{s,a} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right]$$

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

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

Newton Method

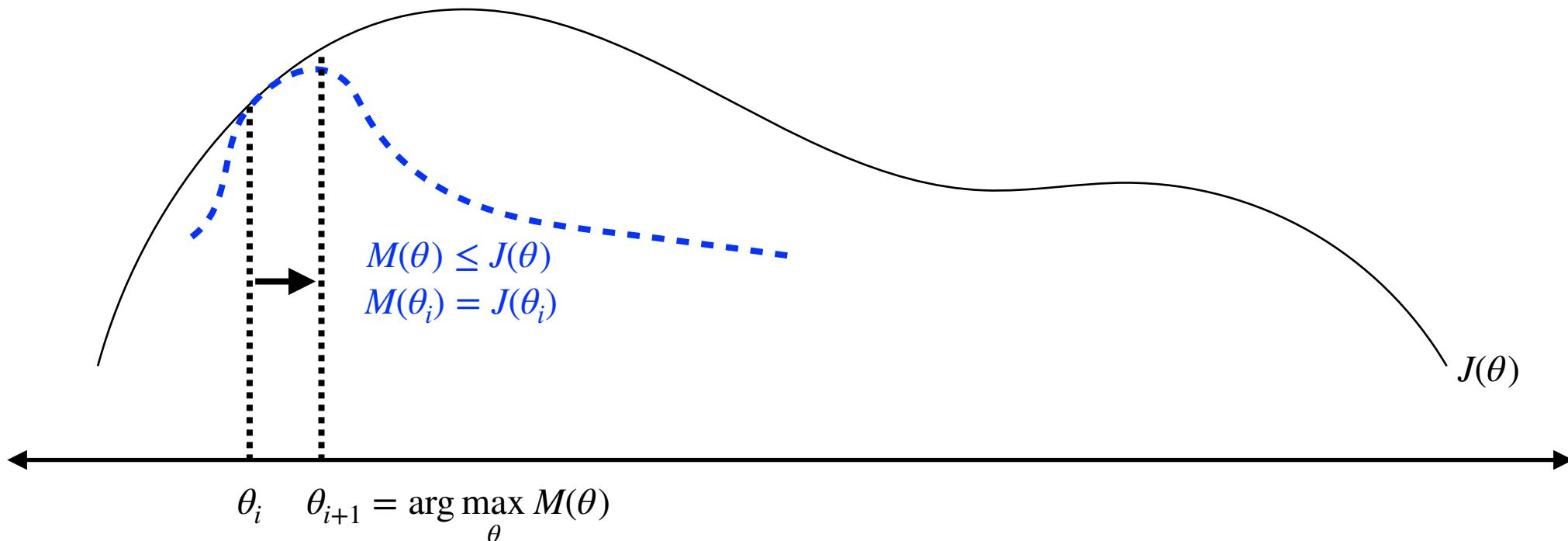
$$\max_{\theta} J(\theta)$$



$$\theta_i \quad \theta_{i+1} = \arg \max_{\theta} \left(J(\theta_i) + J'(\theta_i)(\theta - \theta_i) + \frac{1}{2}J''(\theta_i)(\theta - \theta_i)^2 \right)$$

Minorization Maximization

$$\max_{\theta} J(\theta)$$



Preliminaries

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

- 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:

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

Improvement of $\boldsymbol{\pi}'$ over $\boldsymbol{\pi}$

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

$$\rho_{\boldsymbol{\pi}}(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \dots$$

- Then the return improvement can be written as

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

Performance Improvement

- The return improvement

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

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

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

- 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

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

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

- Why?

Performance Improvement

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

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

- Why?

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

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

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

Performance Improvement

- Local approximation:

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

- 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:

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

Minorization for RL

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

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

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

where $D_{KL}^{max}(\pi, \pi') = \max_s D_{KL}(\pi(\cdot | s), \pi'(\cdot | s))$

- Then the following properties hold:

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

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

Minorization for RL

- Now, we optimize

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

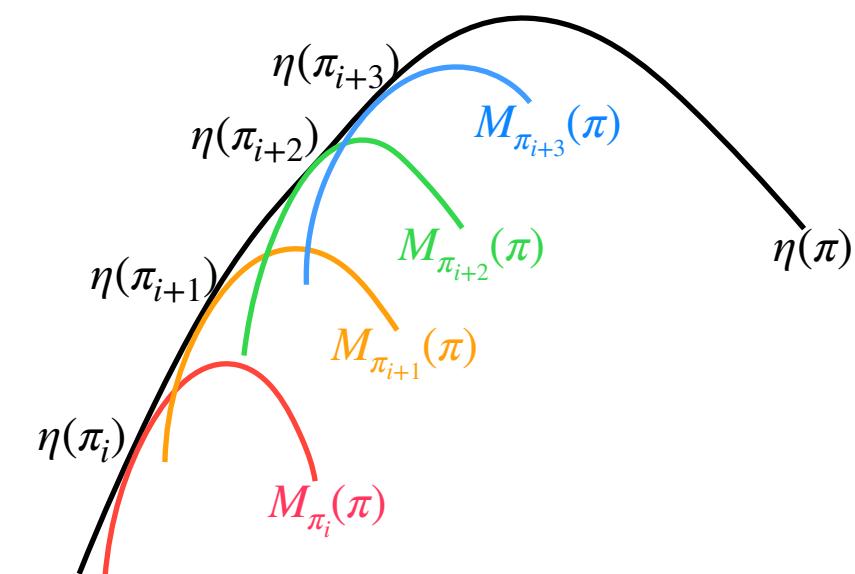
$$M_\pi(\boldsymbol{\pi}') = \eta(\pi) + L_\pi(\boldsymbol{\pi}') - c D_{KL}^{\max}(\pi, \boldsymbol{\pi}')$$

$$\max_{\theta_{i+1}} M_{\pi_{\theta_i}}(\boldsymbol{\pi}_{\theta_{i+1}}) = \eta(\pi_{\theta_i}) + L_{\pi_{\theta_i}}(\boldsymbol{\pi}_{\theta_{i+1}}) - c D_{KL}^{\max}(\pi_{\theta_i}, \boldsymbol{\pi}_{\theta_{i+1}})$$

- Lagrangian relaxation becomes

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

subject to $D_{KL}^{\max}(\pi_{\theta_i}, \boldsymbol{\pi}_{\theta_{i+1}}) \leq \delta$



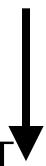
Trust Region Policy Optimization

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

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

- We approximate the KL divergence:

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



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

Trust Region Policy Optimization

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

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

- 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

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

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

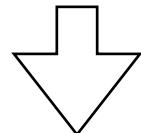
- We use the Monte Carlo Estimate of Q :

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

Trust Region Policy Optimization

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

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



Linear approximation to the loss and
quadratic approximation to the constraint

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

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

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

Trust Region Policy Optimization

- The final TRPO objective becomes:

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

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

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

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

- The update rule of the above problem is

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

Trust Region Policy Optimization

- The update rule of the above problem is

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

- However, the hessian matrix $\mathbf{H}(\theta_{old}) \in \mathbb{R}^{n \times n}$ where n is the number of parameters and the computational complexity of the inverse becomes $O(n^3)$.
- Instead of computing \mathbf{H}^{-1} , we solve the linear equation $\mathbf{H}\mathbf{x} = \mathbf{g}$ using a conjugate gradient method.

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

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

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

- Trust region method
 - The TRPO objective is

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

Clipped Surrogate Objective

- The objective of the TRPO is:

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

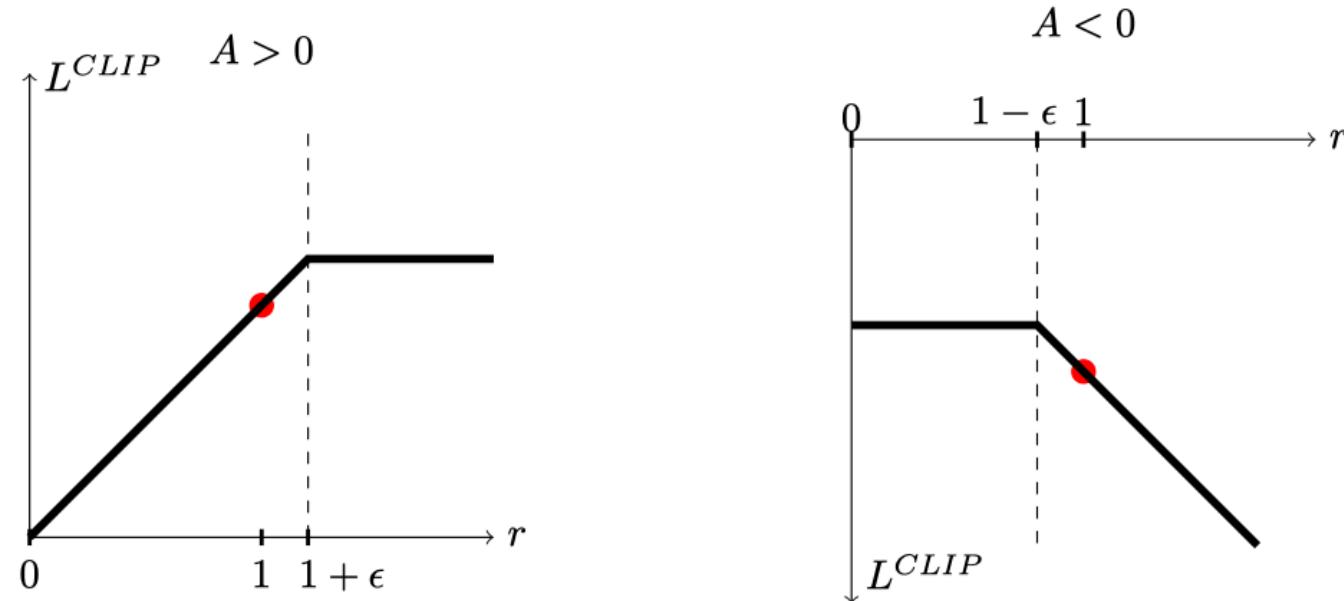
- The main objective of clipped surrogate is:

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

- 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

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



- 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)

- The TRPO objective is:

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

- The unconstrained objective of TRPO is:

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

- The adaptive KL penalty method for PPO is to adaptively change β by checking

$$d = \mathbb{E}_t [D_{KL}[\pi_{\theta_{old}}, \pi_{\theta}]]$$

- If $d < d_{targ}/1.5$, $\beta \leftarrow \beta/2$
- If $d > d_{targ} \times 1.5$, $\beta \leftarrow \beta \times 2$

PPO Implementation

```

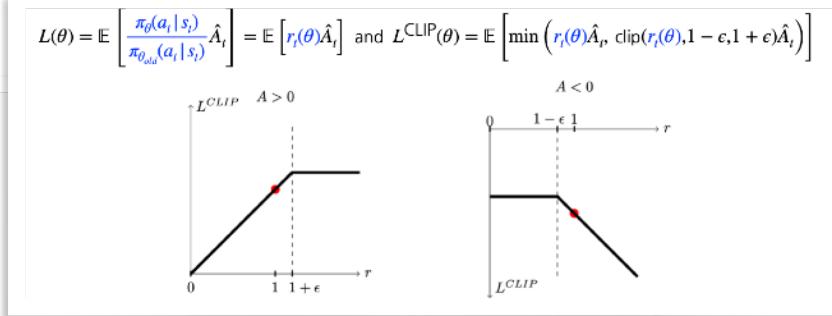
# Update policy
for _ in range(self.train_pi_iters):

    # Capped Surrogate Objective |
    _, 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

```



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

Generalized Advantage Estimation (GAE)

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

Advantage Function Estimation

- 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:

- $\hat{A}_t^{(1)} = \delta_t^V$
- $\hat{A}_t^{(2)} = \delta_t^V + \gamma \delta_{t+1}^V$
- $\hat{A}_t^{(3)} = \delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V$

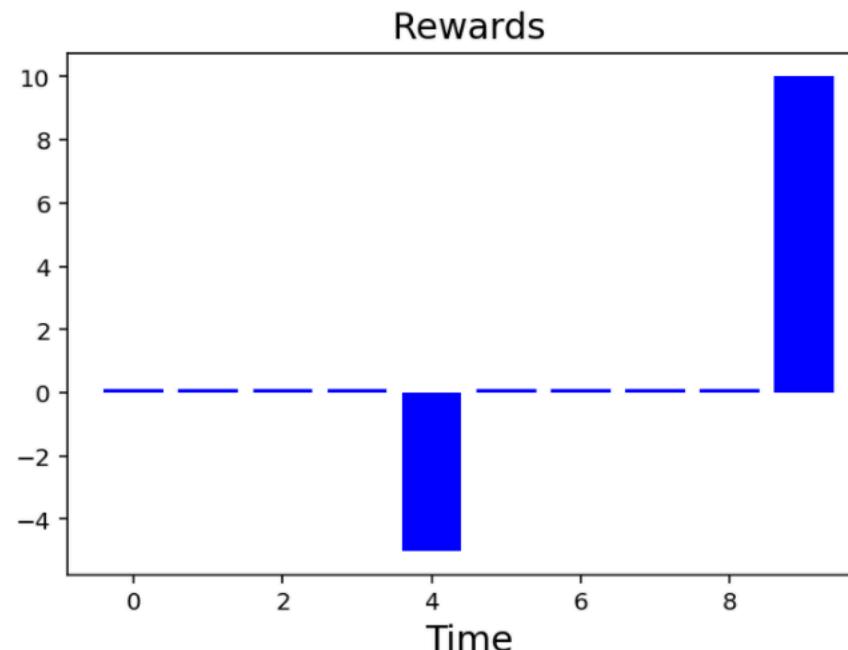
- Finally, we define the λ -exponentially-weighted average of \hat{A}_t :

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

Advantage Function Estimation

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

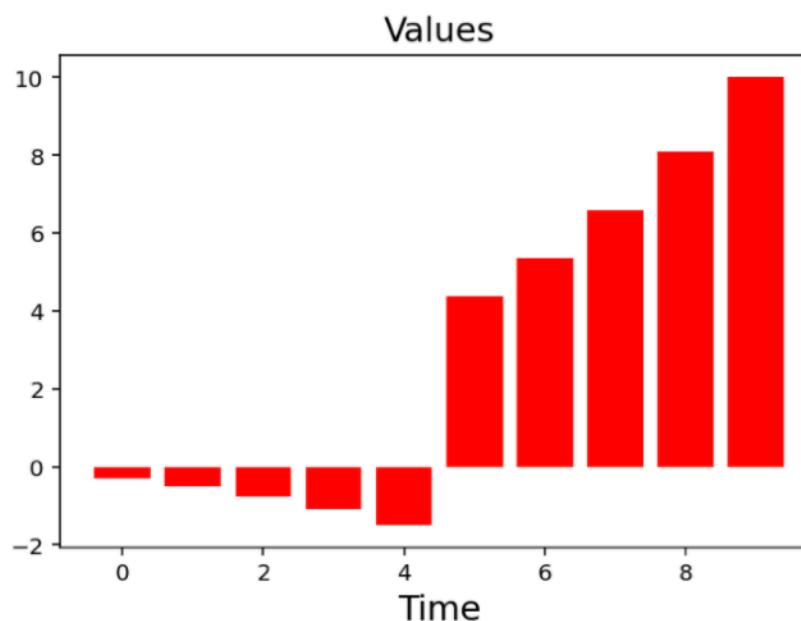
Values

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

```

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

```



Advantage Function Estimation

Generalized Advantage Estimates

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

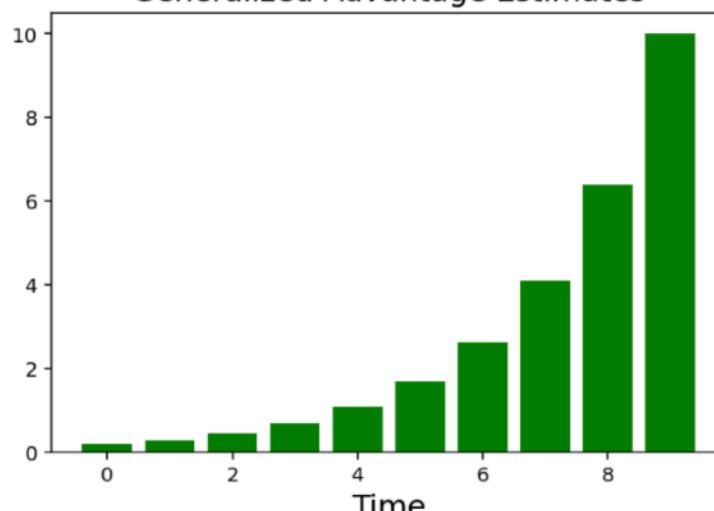
$$\hat{A}_t^{\text{GAE}(\gamma, \lambda)} = \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}^V \quad (16)$$

```

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()

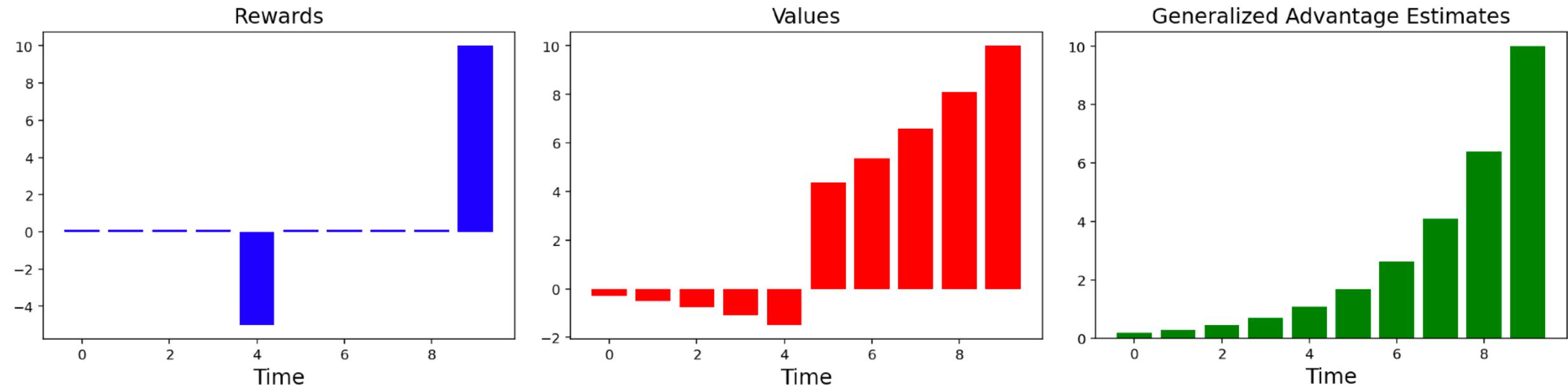
```

Generalized Advantage Estimates



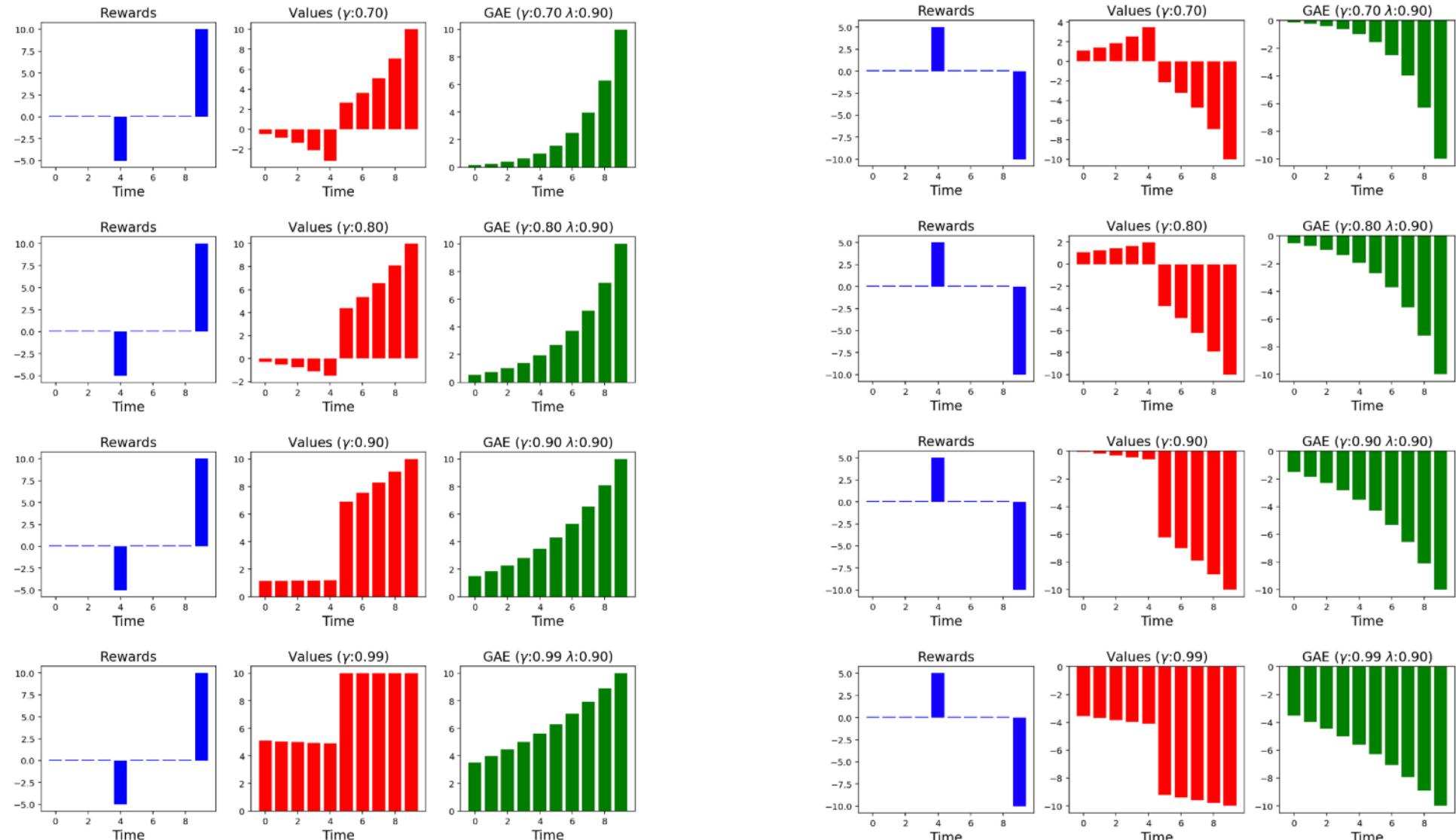
Advantage Function Estimation

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



<https://gist.github.com/sjchoi86/38c7a378cf482a1cde5630e5dde937e>

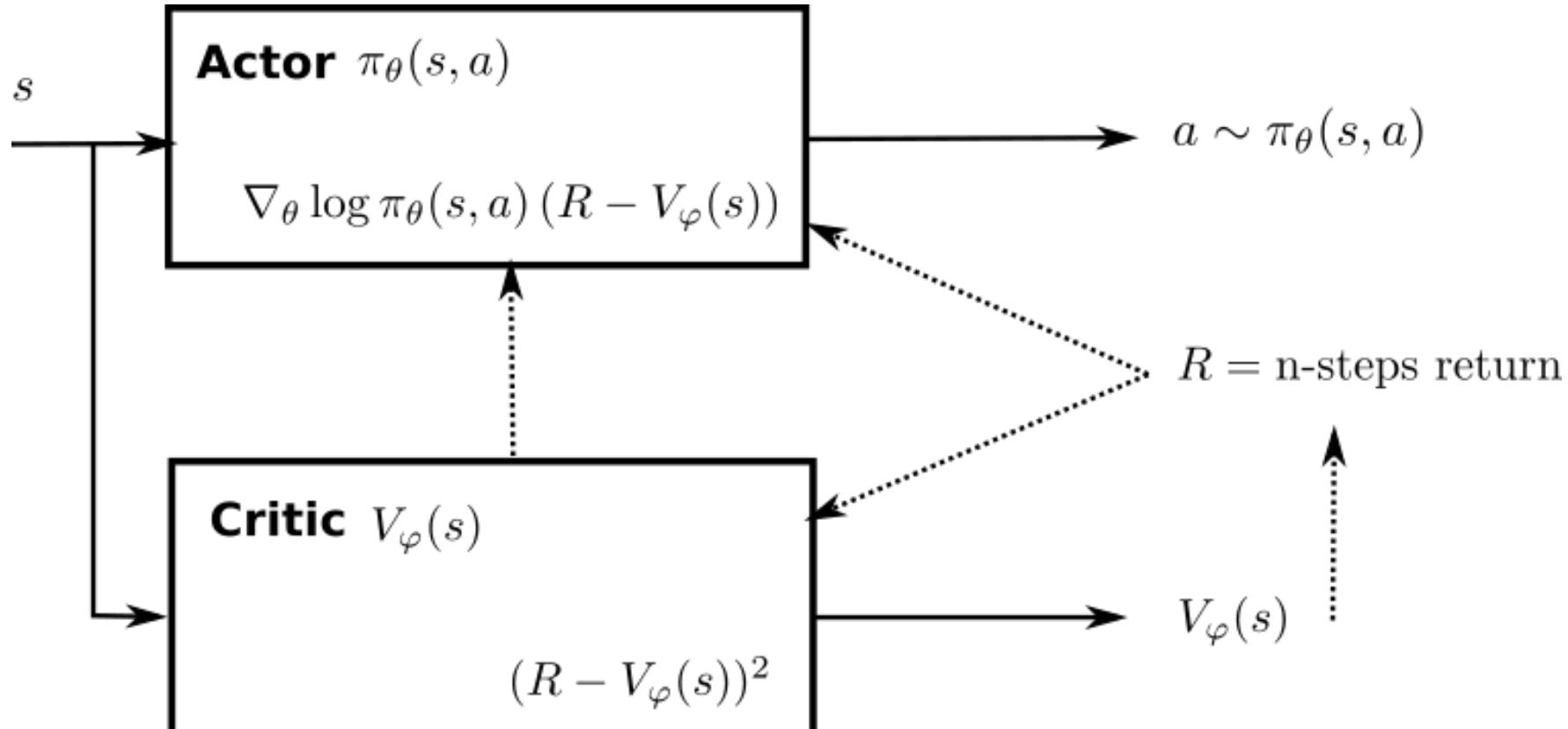
Advantage Function Estimation



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

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

- Standard RL objective

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

- Maximum Entropy RL objective

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

Maximum Entropy RL

- Maximum Entropy RL objective

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

- Policy evaluation step

- The Bellman backup operator for Max-Ent RL is:

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

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

- Policy improvement step

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

Soft Actor-Critic

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

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

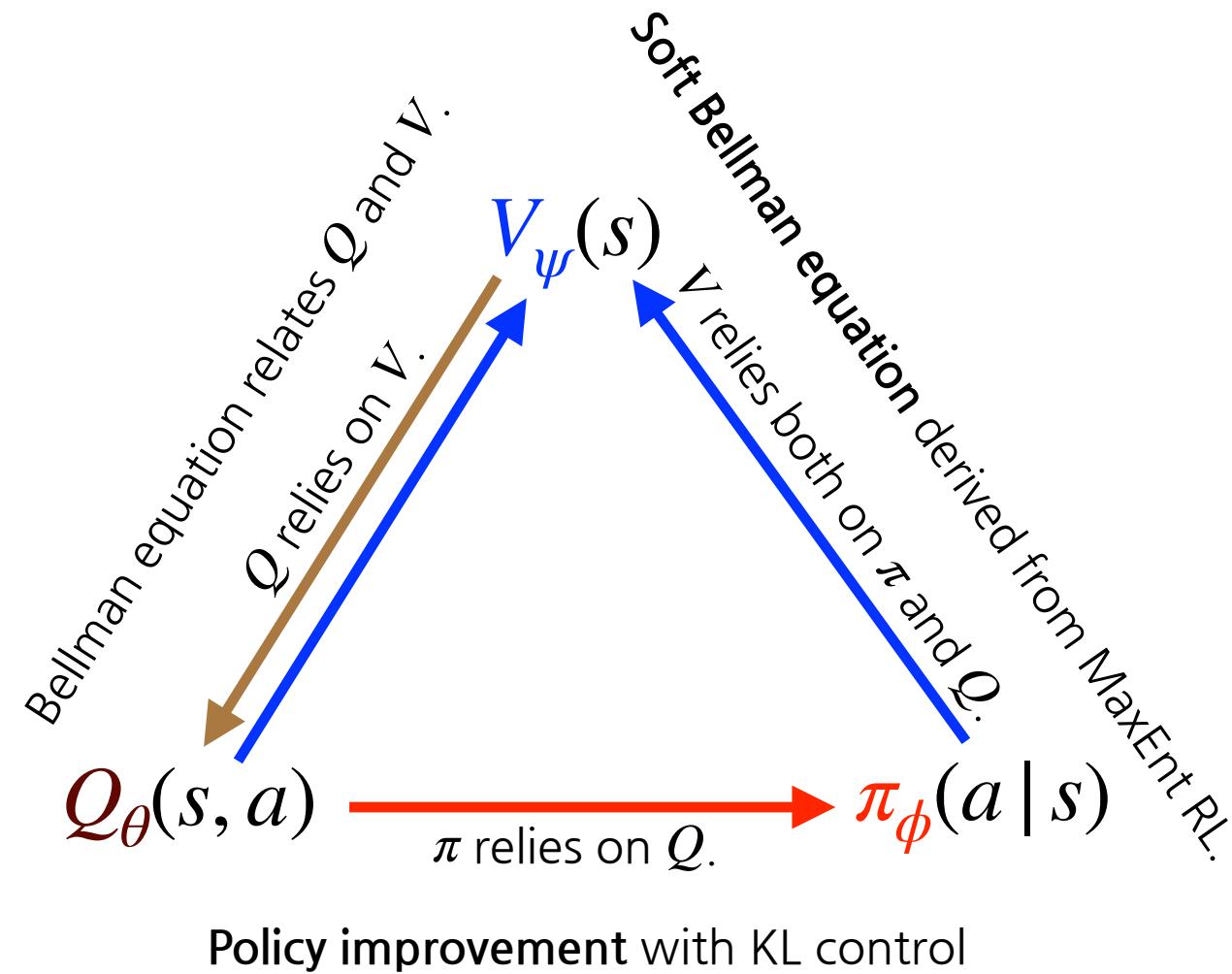
- For learning $\pi_\phi(a | s)$:

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

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

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

Soft Actor-Critic



SAC Implementation

```

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 $\pi_\phi(a | s)$:

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

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

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

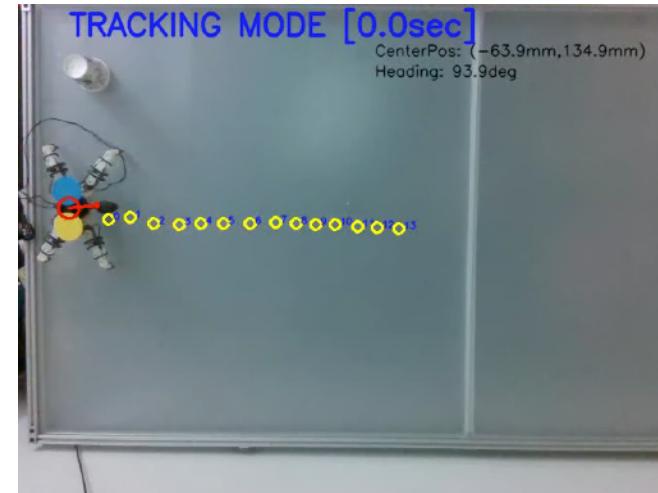
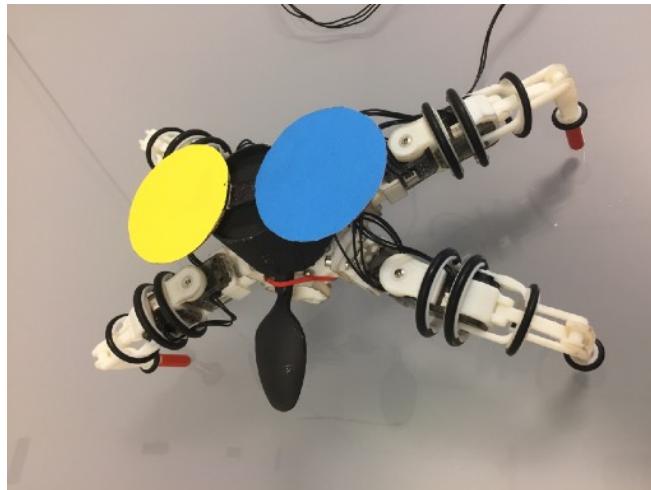
Summary

- Policy Gradient Theorem
 - Optimize the policy directly via $\nabla_{\theta}\eta(\pi_{\theta}) \approx \nabla_{\theta}\log\pi_{\theta}(a_t|s_t)Q_{\pi_{\theta}}(s_t, a_t)$
- 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 $TD(\lambda)$.
- Soft Actor-Critic (**SAC**)
 - Entropy-regularized RL with an actor-critic method.

Trajectory-based RL

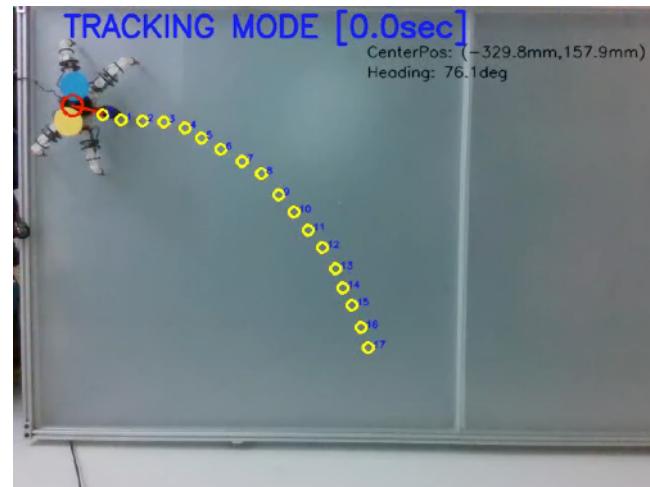
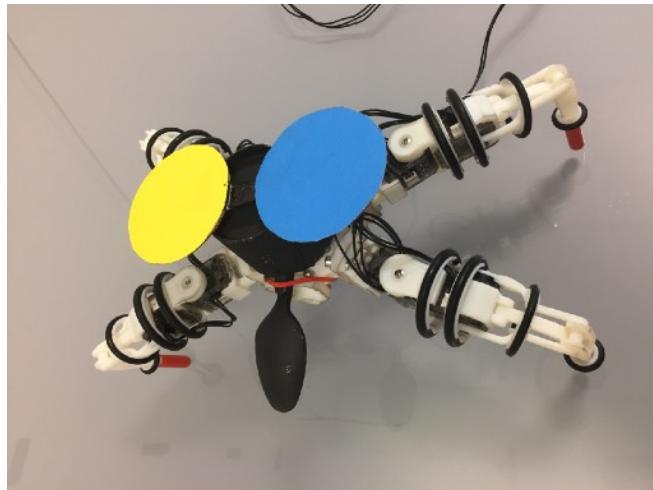
"Deep Latent Policy Gradient for Quadruped Locomotion with Snapbot," 2019

Motivation



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

Motivation

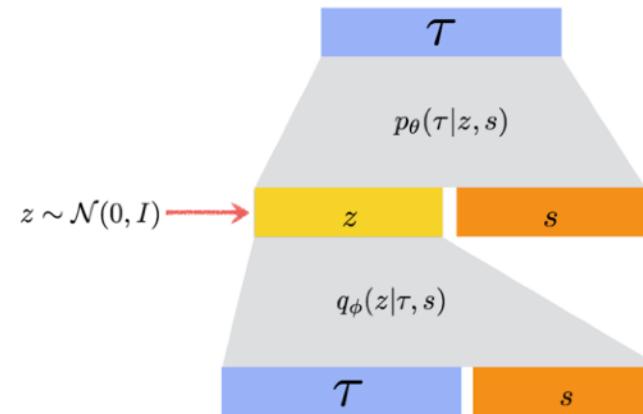


We focus on two known drawbacks of **reinforcement learning** when applying RL to real physical systems.

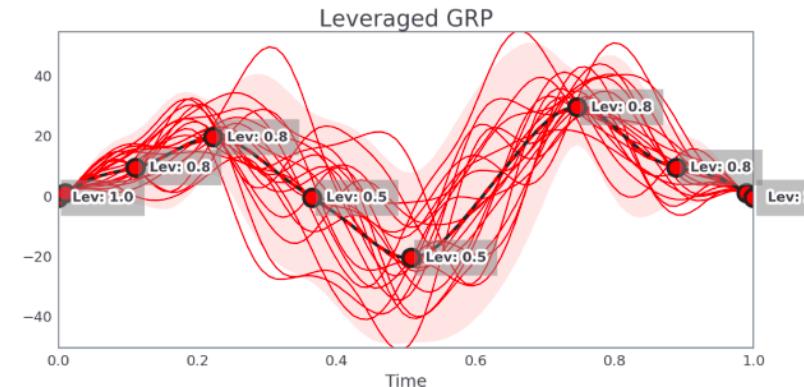
1. Inefficient **exploration** phase
2. Hard to design a proper **reward** function

Proposed method

Deep Latent Policy Gradient



Leveraged Gaussian Random Path



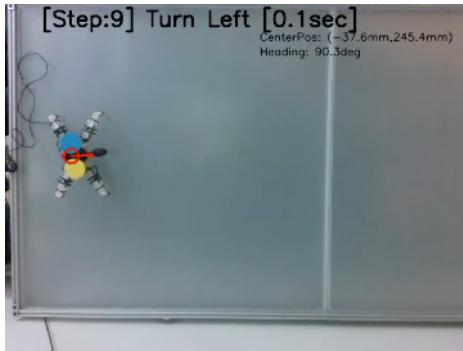
- We propose a deep latent policy gradient method for optimizing a stochastic policy function.
- We present a leveraged Gaussian random path which defines a probability distribution over smooth trajectories.

Experiments

- **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

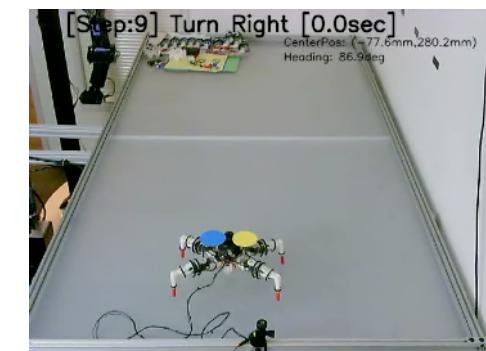
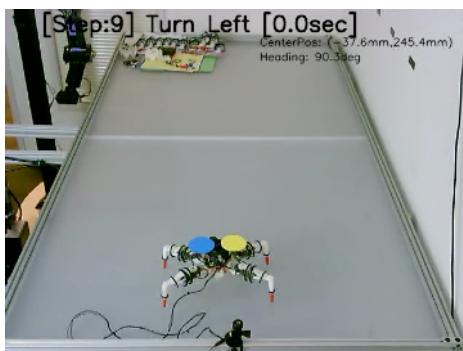
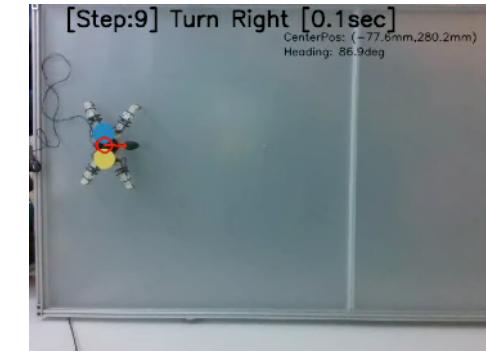
Turn Left



Go Straight

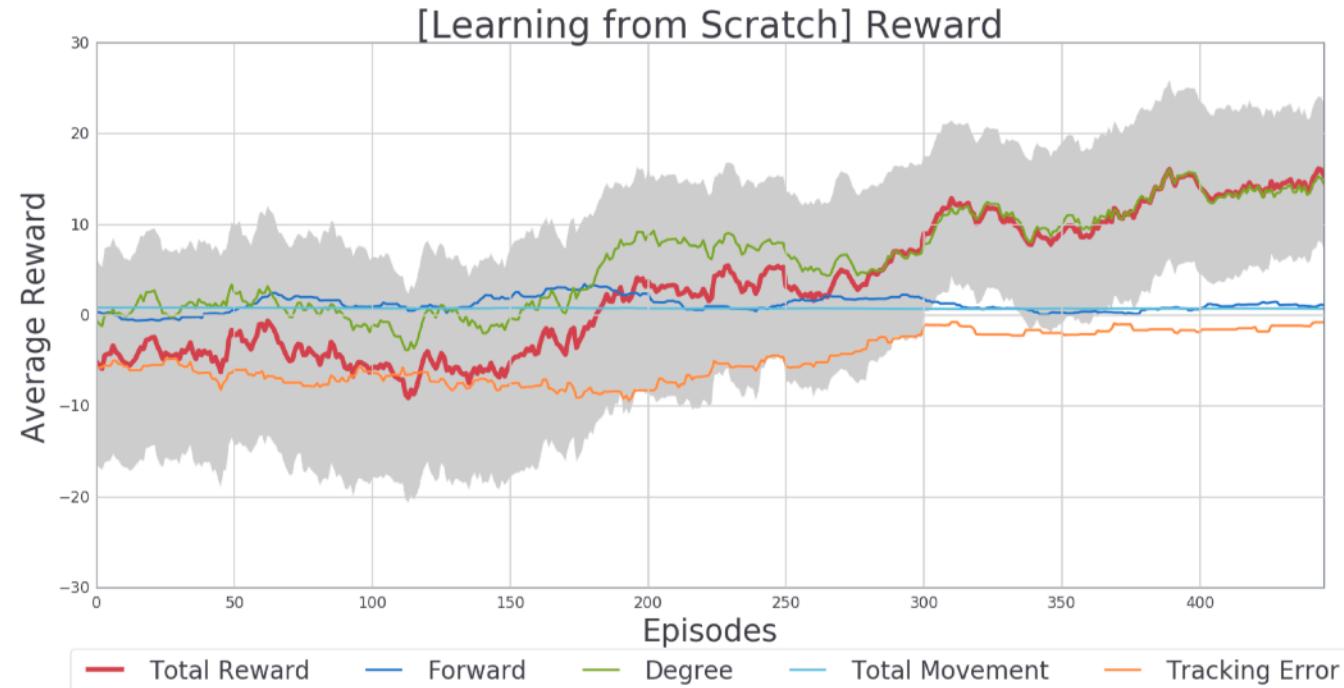


Turn Right



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

Experiments - Learning from Scratch



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

Experiments - Learning with Prior

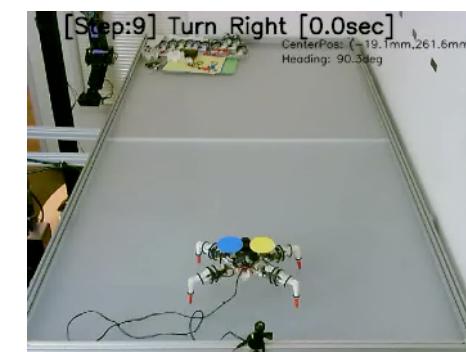
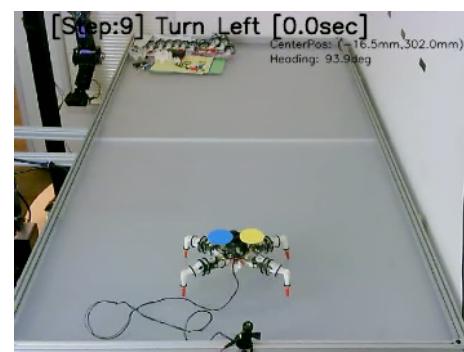
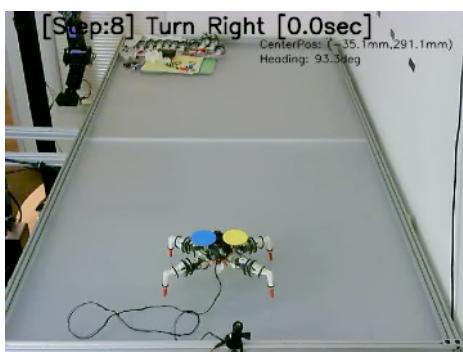
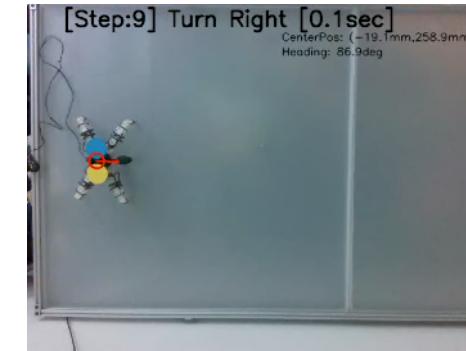
Turn Left



Go Straight

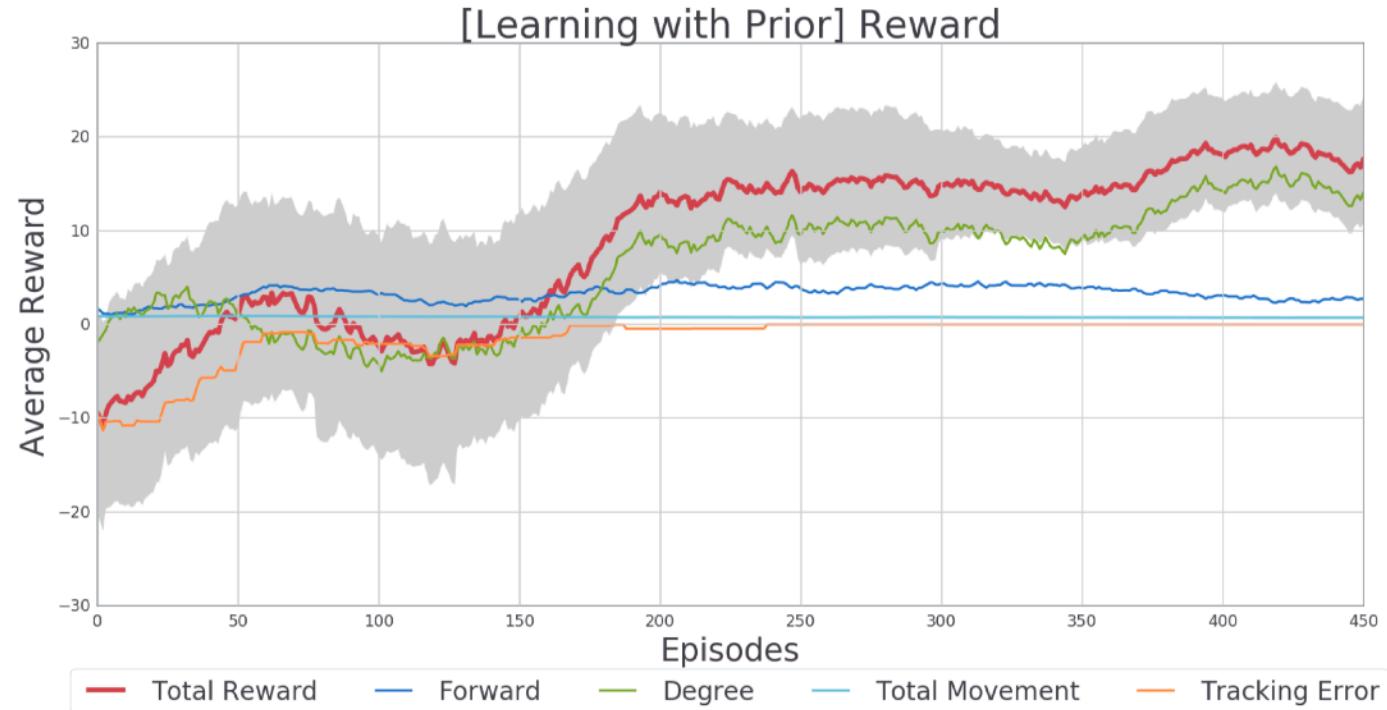


Turn Right



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

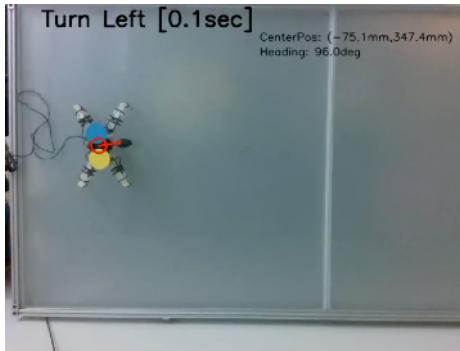
Experiments - Learning with Prior



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

Experiments - Curriculum Learning

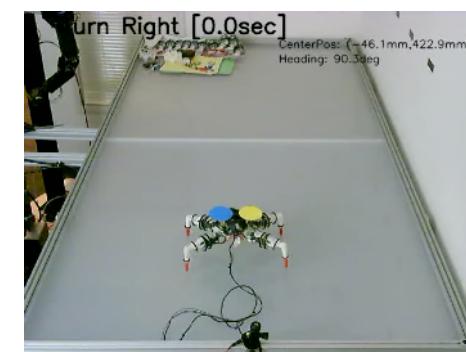
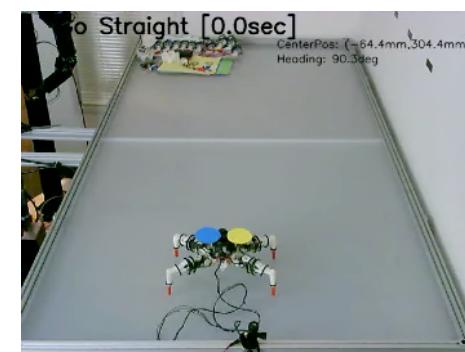
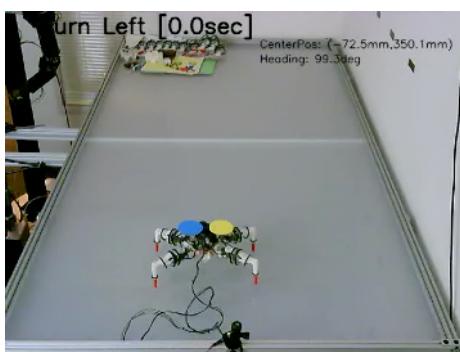
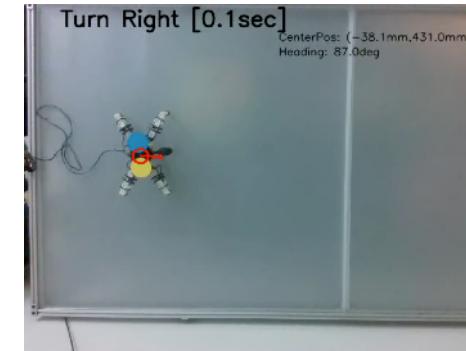
Turn Left



Go Straight

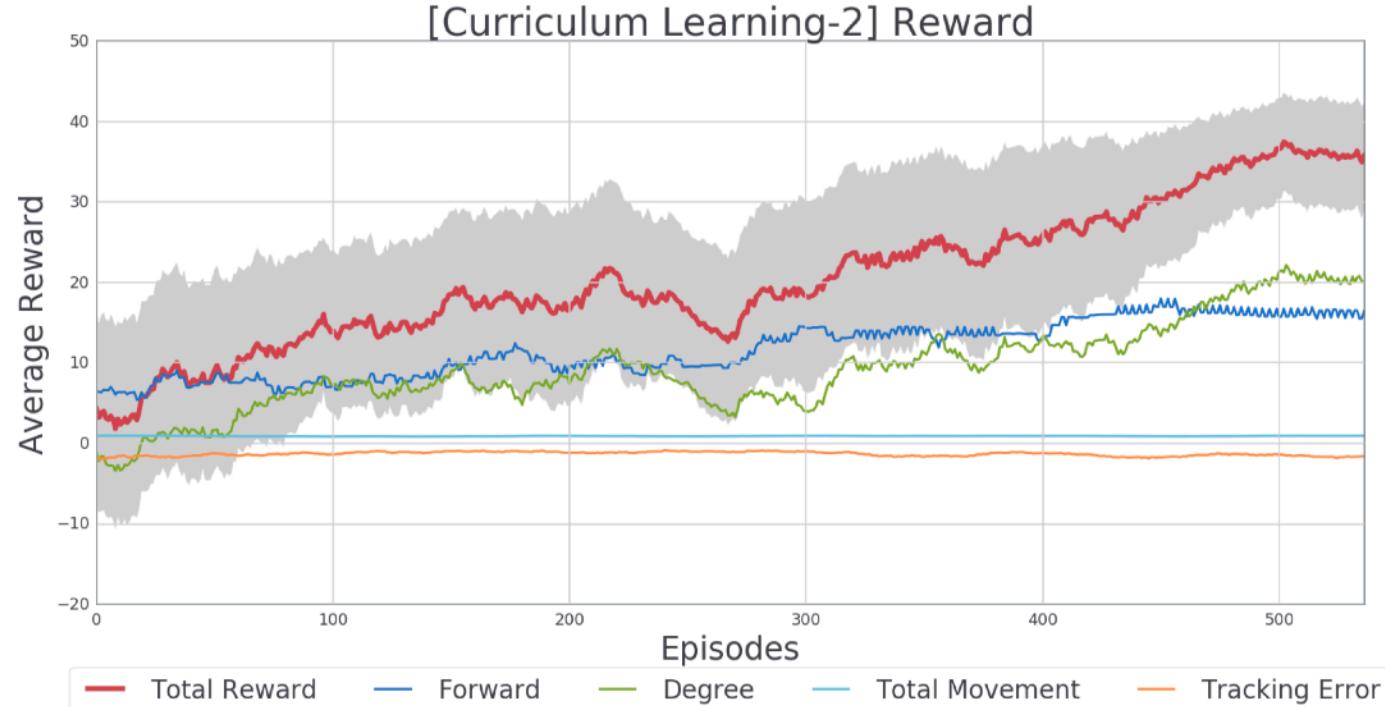


Turn Right



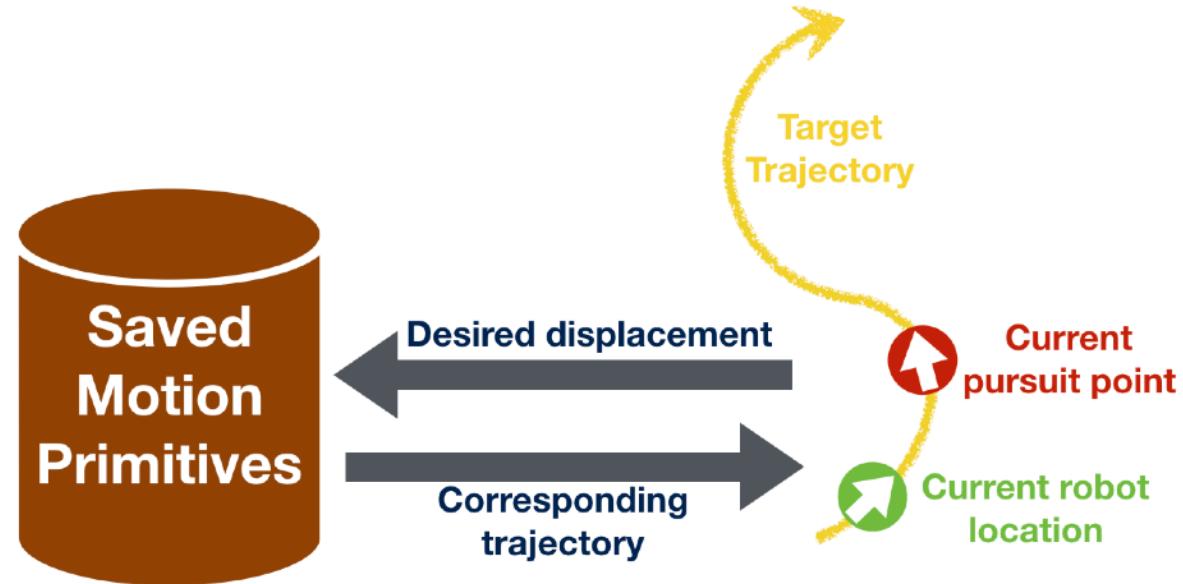
- Final motions of Curriculum Learning show that it can successfully go straight while turning left or right. Note that it can make more **diverse** motions!

Experiments - Curriculum Learning



- 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!

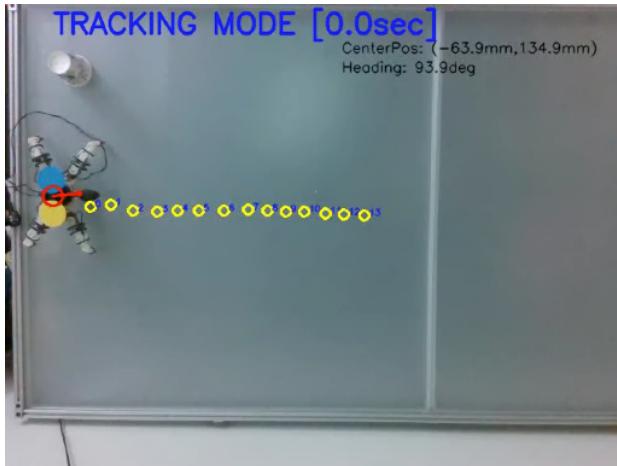
Trajectory tracking experiments



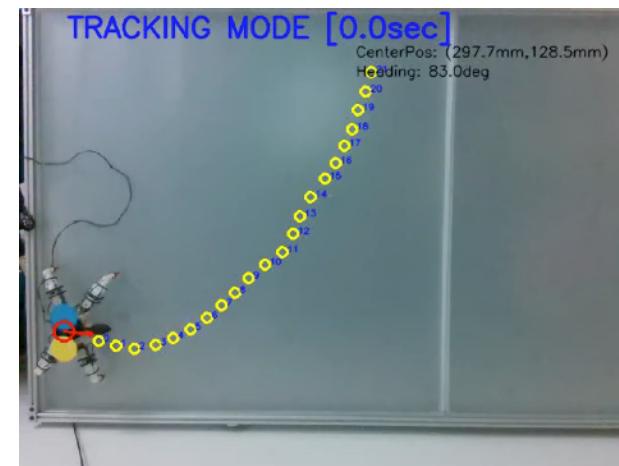
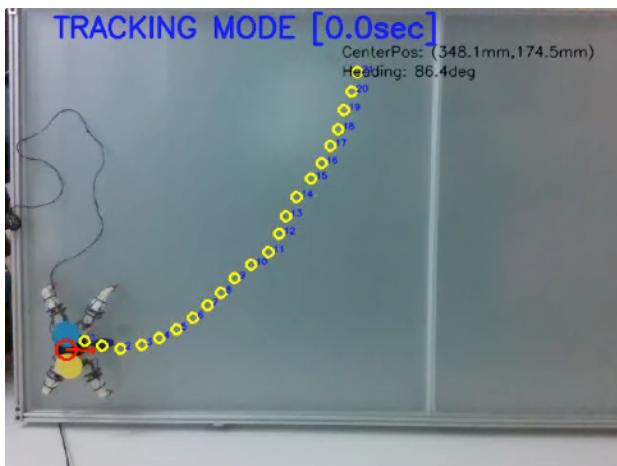
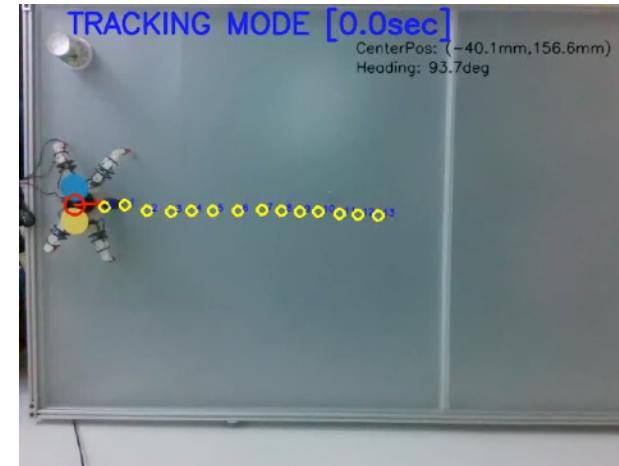
- We train nonparametric motion planners using trajectories collected from [Learning with Prior](#) and [Curriculum Learning](#).

Trajectory tracking experiments

Learning with Prior Setting

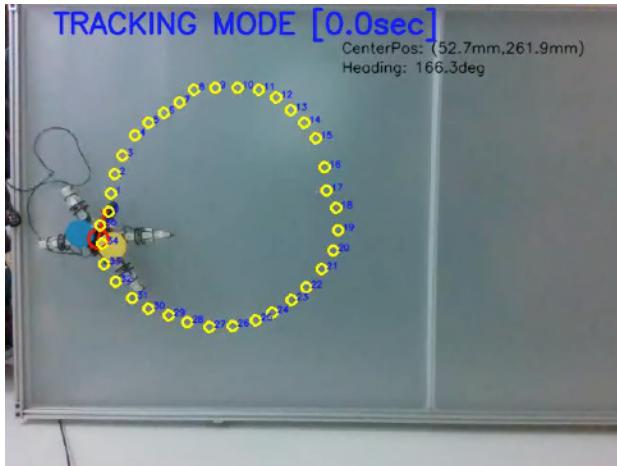


Curriculum Learning

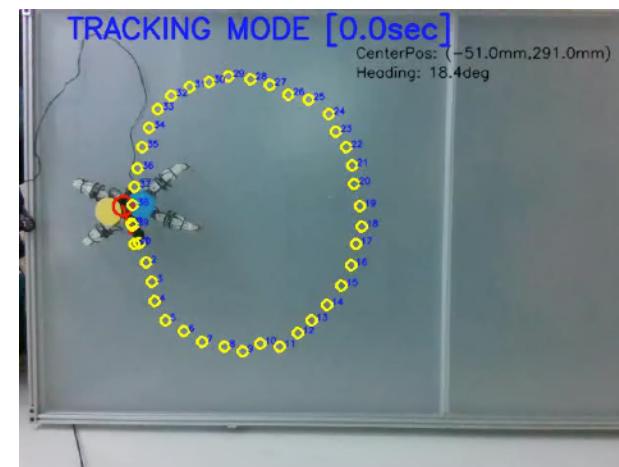
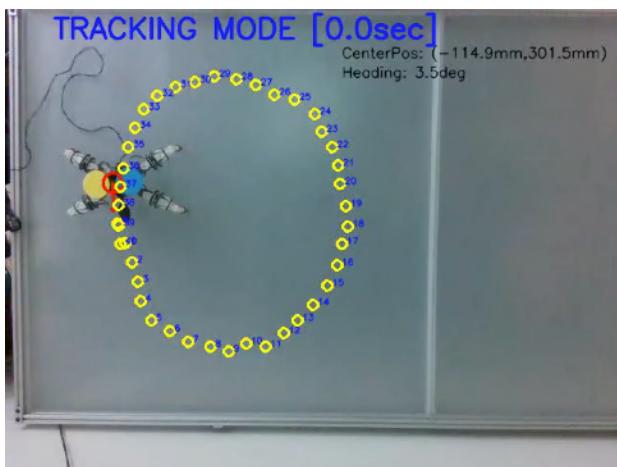
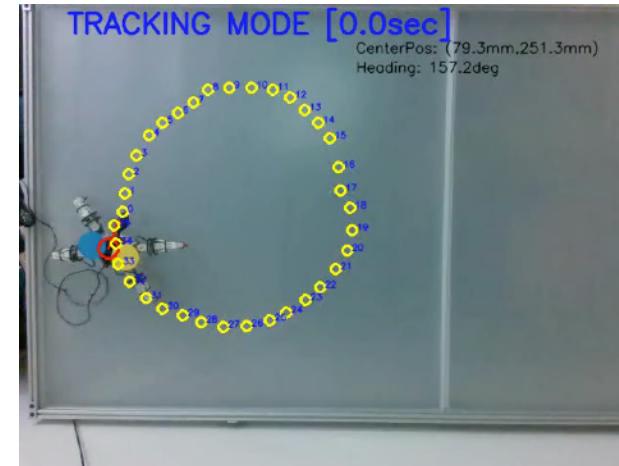


Trajectory tracking experiments

Learning with Prior Setting



Curriculum Learning





ROBOT INTELLIGENCE LAB