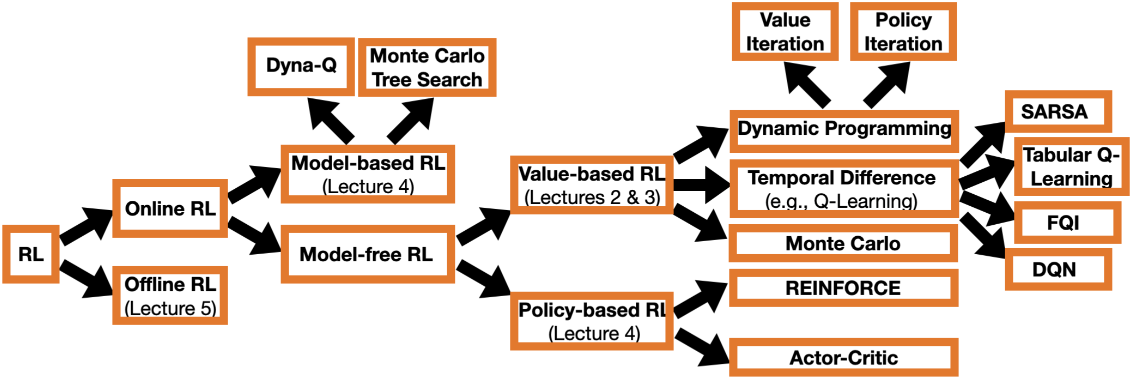
Reinforcement Learning

Lecture 4: Policy- and Model-based Learning

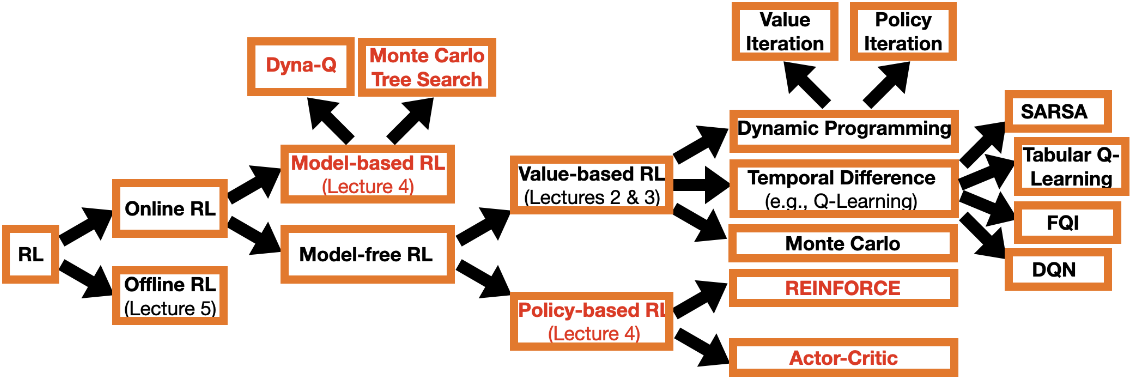
Chengchun Shi

Roadmap



RL
Online RL
Offline RL (Lecture 5)
Model-based RL (Lecture 4)
Model-free RL
Dyna-Q
Monte Carlo Tree Search
Value-based RL (Lectures 2 & 3)
Policy-based RL (Lecture 4)
Value Iteration
Policy Iteration
Dynamic Programming
Temporal Difference (e.g., Q-Learning)
Monte Carlo
REINFORCE
Actor-Critic
SARSA
Tabular Q-Learning
FQI
DQN

Roadmap (Cont'd)



Lecture Outline

Lecture Outline

Policy We Studied So Far

- **Greedy policy**:

$$\pi^{\text{opt}}(s) = \arg\max_{a} Q^{\pi^{\text{opt}}}(s, a)$$

- ϵ-**Greedy policy**:

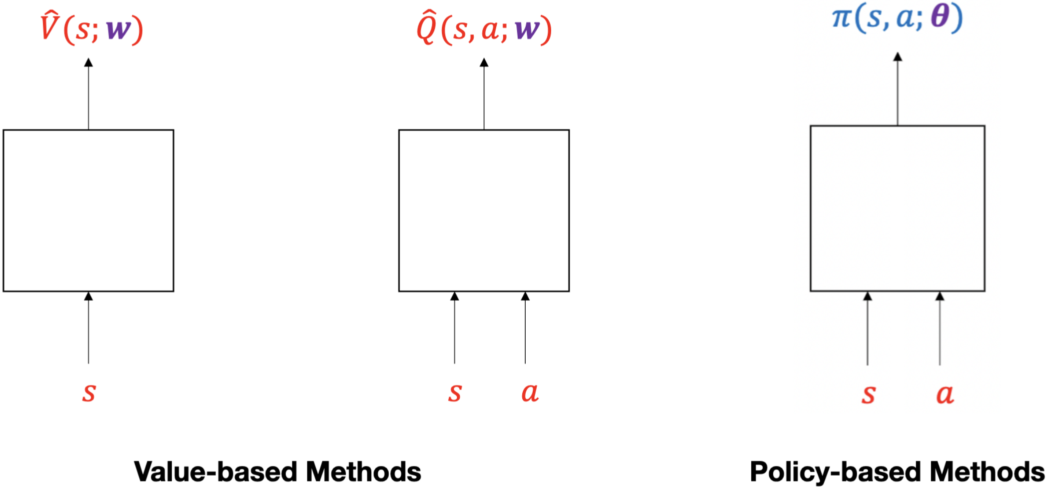
$$\begin{cases} \pi^{\text{opt}}(s), & \text{with probability } 1 - \epsilon \\ \text{random action}, & \text{with probability } \epsilon. \end{cases}$$

- **Value-based methods**: Policy Iteration, Value Iteration, Monte Carlo, SARSA, Q-Learning, etc.

Value-based v.s. Policy-based Methods

- **Value-based methods**: derive $\pi^{\mathbf{opt}}$ by learning an optimal Q-function (or value function)
- **Policy-based methods**: search $\pi^{\mathbf{opt}}$ within a restricted function class (e.g., linear, neural networks) that maximizes the value

Value-based v.s. Policy-based Methods (Cont'd)



Example: Linear Function Approximation

- Linear approximation of features $\phi(s, a)$
- State-action value function approximation

$$\boldsymbol{Q}(s, a; \boldsymbol{\theta}) = \phi^\top(s, a)\boldsymbol{\theta}$$

- Policy function approximation

$$\boldsymbol{\pi}(s, a; \boldsymbol{\theta}) = \frac{\exp(\phi^\top(s, a)\boldsymbol{\theta})}{\sum_{a'} \exp(\phi^\top(s, a')\boldsymbol{\theta})}$$

Value-based v.s. Policy-based Methods (Cont'd)

- **Pros** of policy-based methods:
  1. Suitable for learning general **stochastic** policies (value-based methods mainly designed for deterministic policies)
  2. More **robust** to model misspecification
  3. Scalable for **high-dimensional** or **continuous** action spaces (SARSA, Q-learning mainly designed for discrete action space)

- **Cons** of policy-based methods:
  1. Convergence to local minima
  2. May have large variance

Example I: Advantage of Stochastic Policy

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Consider iterated rock-paper-scissors
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (Nash equilibrium)

Example II: Robustness of Policy-based Method

- Q-function is more **difficult** to model compared to the optimal policy
- Example: optimal Q-function: $Q^{\pi^{\text{opt}}}(s, a) = \mathrm{g}(\phi^\top(s, a)\theta^*)$ for some monotonically increasing function $\mathrm{g} : \mathbb{R} \to \mathbb{R}$
- When g is not **linear** function, value-based method misspecifies Q-function model

$$\mathrm{g}(\phi^\top(s, a)\theta^*) \neq \phi^\top(s, a)\theta$$

- However, since g is a monotonically increasing function

$$\pi^{\text{opt}}(s) = \arg\max_{a} \mathrm{g}(\phi^\top(s, a)\theta^*) = \arg\max_{a} \phi^\top(s, a)\theta^*$$

- Policy-based methods correctly specify the optimal policy

$$\frac{\exp(\phi^\top(s, a)\theta)}{\sum_{a'} \exp(\phi^\top(s, a')\theta)} \to \mathbb{I}(a = \pi^{\text{opt}}(s))$$

when $\theta = k\theta^*$ and $k \to \infty$

Policy Objective Functions

- Average rewards:

$$J(\theta) = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}^{\pi(\bullet;\theta)} \left[\sum_{t=0}^{T-1} R_t \right] = \sum_{s,a} \nu^{\pi(\bullet;\theta)}(s)\pi(s,a;\theta)\mathcal{R}_s^a$$

  where $\mathcal{R}_s^a = \mathbb{E}(R_t | A_t = a, S_t = s)$
- For each π, the states $\{S_t\}_t$ forms a time-homogeneous Markov chain
- $\nu^{\pi(\bullet;\theta)}$ the stationary distribution of $\{S_t\}_t$ under $\pi(\bullet;\theta)$

Policy Objective Functions (Cont'd)

- Discounted rewards: given a discounted factor $\gamma \in [0, 1]$ and initial state distribution ν, maximize the expected discounted rewards:

$$J(\theta) = \mathbb{E}^{\pi(\bullet;\theta)}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right],$$

  or equivalently,

$$J(\theta) = \sum_{s} \nu(s) V^{\pi(\bullet;\theta)}(s)$$

- If $\gamma = 1$, the task is assumed to be episodic

Lecture Outline

Policy Gradient

- **Objective**: identify the maximizer of $\boldsymbol{J(\theta)}$
- **Method**: apply (stochastic) gradient ascent algorithm to update $\boldsymbol{\theta}$ (gradient descent to minimize $-\boldsymbol{J(\theta)}$)

$$\boldsymbol{\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J(\theta_t)}$$

Need to calculate the gradient $\boldsymbol{\nabla_\theta J(\theta)}$!

Policy Gradient Theorem

Theorem

For any differentiable policy $\pi(s, a; \theta)$ with respect to parameter θ, the policy gradient for average reward and discounted expected rewards objective is

$$\nabla_\theta J(\theta) = \sum_{s,a} \mu^{\pi(\bullet;\theta)}(s, a) \nabla_\theta \log(\pi(s, a; \theta)) Q^{\pi(\bullet;\theta)}(s, a)$$

- For average reward objective:
  $\mu^{\pi(\bullet;\theta)}$ is the stationary distribution of $\{(S_t, A_t)\}_t$ under $\pi(\bullet; \theta)$
- For discounted expected rewards objective:

$$\mu^{\pi(\bullet;\theta)}(s, a) = \sum_{t \geq 0} \gamma^t \mathrm{Pr}^{\pi(\bullet;\theta)}(S_t = s, A_t = a)$$

  Discounted state-action visitation probability

Policy Gradient Theorem (Cont'd)

Theorem

For any differentiable policy $\pi(s, a; \theta)$ with respect to parameter θ, the policy gradient for average reward and discounted expected rewards objective is

$$\nabla_\theta J(\theta) = \sum_{s,a} \mu^{\pi(\bullet;\theta)}(s, a) \nabla_\theta \log(\pi(s, a; \theta)) Q^{\pi(\bullet;\theta)}(s, a)$$

- For average reward objective:

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[\sum_{t \geq 0} (R_t - J(\theta)) \middle| S_0 = s, A_0 = a \right]$$

- For discounted expected rewards objective: Q-function defined as usual.
- Proof given in the appendix

Policy Score

- For any state-action pair $(\boldsymbol{s}, \boldsymbol{a})$, the term

$$\nabla_{\boldsymbol{\theta}} \log(\boldsymbol{\pi}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta}))$$

  is referred as the **policy score**

Example 1: Softmax Policy Gradient

- State-action pairs weighted by linear combination of features

$$\pi(s, a; \theta) = \frac{\exp(\phi^\top(s, a)\theta)}{\sum_{a'} \exp(\phi^\top(s, a')\theta)}$$

- The score function

$$\nabla_\theta \log \pi(s, a; \theta) = \phi(s, a) - \frac{\sum_{a'} \phi(s, a) \exp(\phi^\top(s, a)\theta)}{\sum_{a'} \exp(\phi^\top(s, a')\theta)}$$

or equivalently,

$$\nabla_\theta \log \pi(s, a; \theta) = \phi(s, a) - \mathbb{E}_{a' \sim \pi(s, \bullet; \theta)} \phi(s, a')$$

Example 2: Continuous Action Space

- Action space: set of real numbers $\mathcal{A} = \mathbb{R}$
- Policy approximator:

$$\pi(s, a, \theta) = \frac{1}{\sqrt{2\pi}\sigma(s; \theta)} \exp\left(-\frac{(a - \mu(s; \theta))^2}{2\sigma^2(s; \theta)}\right),$$

  where μ and σ are mean and deviation function approximators
- Linear function approximator with feature vectors $\phi_\mu(s)$ and $\phi_\sigma(s)$
  - $\mu(s; \theta) = \phi_\mu^\top(s)\theta_\mu$ and $\sigma(s; \theta) = \phi_\sigma^\top(s)\theta_\sigma$
  - $\nabla_{\theta_\mu} \log \pi(s, a, \theta) = \frac{a - \mu(s;\theta)}{\sigma^2(s;\theta)} \phi_\mu(s)$
  - $\nabla_{\theta_\sigma} \log \pi(s, a, \theta) = \frac{(a - \mu(s;\theta))^2 - \sigma^2(s;\theta)}{\sigma^2(s;\theta)} \phi_\sigma(s)$

Example 3: Bernoulli, Logistic Example

- Actions space: binary, $\{\mathbf{0}, \mathbf{1}\}$
- Policy approximator:

$$\boldsymbol{\pi}(\mathbf{1}, \boldsymbol{s}; \boldsymbol{\theta}) = \mathbf{1} - \boldsymbol{\pi}(\mathbf{0}, \boldsymbol{s}; \boldsymbol{\theta}) = \boldsymbol{p}(\boldsymbol{s}; \boldsymbol{\theta})$$

  where $\boldsymbol{p}(\boldsymbol{s}; \boldsymbol{\theta})$ is a function approximator
- Linear function approximator with feature vectors $\phi(\boldsymbol{s})$
  - Logistic function $\boldsymbol{\sigma}(\boldsymbol{x}) = [\mathbf{1} + \exp(-\boldsymbol{x})]^{-\mathbf{1}}$
  - For exponential soft-max policy $\boldsymbol{p}(\boldsymbol{s}; \boldsymbol{\theta}) = \boldsymbol{\sigma}(\phi^\top(\boldsymbol{s})\boldsymbol{\theta})$
  - $\nabla_{\boldsymbol{\theta}} \log(\boldsymbol{\pi}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta})) = (\boldsymbol{a} - \boldsymbol{\sigma}(\phi^\top(\boldsymbol{s})\boldsymbol{\theta}))\phi(\boldsymbol{s})$

Lecture Outline

REINFORCE: MC Policy Gradient Algorithm

- To maximize $J(\theta)$, we apply (stochastic) gradient ascent algorithm

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J(\theta_t)$$

- According to the policy gradient theorem,

$$\nabla_\theta J(\theta) = \sum_{s,a} \mu^{\pi(\bullet;\theta)}(s, a) \nabla_\theta \log(\pi(s, a; \theta)) Q^{\pi(\bullet;\theta)}(s, a)$$

- Focus on the average reward setting
- μ^π (stationary state-action distribution) is unknown: use empirical state-action distribution $\{(S_t, A_t)\}_t$ as an approx
- Q^π is unknown: use empirical return $G_t = \sum_{j=t}^{T} R_j$ as an approx

REINFORCE: Pseudocode

- **Initialization**: θ arbitrary
- **For each** episode $(S_0, A_0, R_0, \cdots, S_T, A_T, R_T)$ generated using policy $\pi(\bullet; \theta)$

    For $t = 0, 1, 2, \cdots, T$ **do**:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log(\pi(S_t, A_t; \theta)) G_t$$

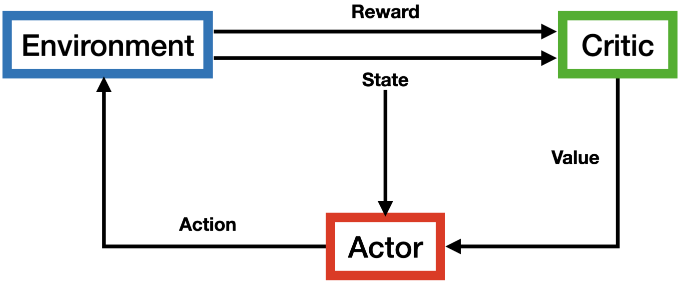
    end for

return θ

Actor-Critic Algorithm

- MC policy gradient algorithm may have a large **variance**
  - Return involves many state transitions, many actions and many rewards
- Solution sought by using **actor-critic algorithms**
- Actor-critic algorithms combine **policy gradient** with **value function estimation**

Actor-Critic Algorithm (Cont'd)



- **Critic** uses function approximator to learn value function
- **Actor** uses policy approximator to learn optimal policy

Actor-Critic Control

- **Critic**: estimates $Q^{\pi(\bullet;\theta)}(s, a)$ by a function approximator $\widehat{Q}(s, a; \omega)$
  - The critic performs **policy evaluation**
  - Standard methods can be applied: MC, TD
- **Actor**: updates policy parameter θ
  - The actor performs control using approximate policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{(s,a)\sim\mu} \nabla_\theta \log(\pi(s, a; \theta)) Q^\pi(s, a; \omega)$$

  - Parameter update
    - Average reward setting

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log(\pi(S_t, A_t; \theta)) \widehat{Q}(S_t, A_t; \omega)$$

    - Discounted reward setting

$$\theta \leftarrow \theta + \alpha \gamma^t \nabla_\theta \log(\pi(S_t, A_t; \theta)) \widehat{Q}(S_t, A_t; \omega)$$

Example: Actor-Critic with Linear Value Function

- Linear value function approximator

$$\widehat{Q}(s, a; \omega) = \phi^\top(s, a)\omega$$

- Focus on the discounted reward setting
- **Critic**: updates ω by linear TD

$$\omega_{t+1} = \omega_t + \eta\phi(S_t, A_t)(R_t + \gamma\phi^\top(S_{t+1}, A_{t+1})\omega_t - \phi^\top(S_t, A_t)\omega_t)$$

Pseudocode

- **Initialization**: s, θ, ω
- **For each** episode:

    Initialize $t = 0$

    Sample action a from $\pi(\bullet, s; \theta)$

    Repeat until s is terminal

        Receive reward r and next state s'

        Sample action a' from $\pi(\bullet, s; \theta)$

        $\theta \leftarrow \theta + \alpha\gamma^t \nabla_\theta \log(\pi(s, a; \theta))\phi^\top(s, a)\omega$

        $\omega \leftarrow \omega + \eta\phi(s, a)[r + \gamma\phi^\top(s', a')\omega - \phi^\top(s, a)\omega]$

        $a \leftarrow a'$ and $s \leftarrow s'$

        $t \leftarrow t + 1$

Bias-Variance Tradeoff

- **REINFORCE** uses Return G_t, an unbiased estimate of $Q^{\pi(\bullet;\theta)}(s, a)$
- **Actor-critic** uses $\widehat{Q}(s, a; \omega)$, a biased estimate of $Q^{\pi(\bullet;\theta)}(s, a)$
- REINFORCE gradient has **high variance** and **zero bias**
- Actor-critic gradient has **low variance** and **some bias**
- Similar to Pros & Cons of MC vs TD (Lecture 2)

Lecture Outline

Variance Reduction Using a Baseline

- Recall that policy parameter update

$$\theta \leftarrow \theta + \alpha\gamma^t \nabla_\theta \log(\pi(S_t, A_t; \theta))\widehat{Q}(S_t, A_t; \omega)$$

- For any θ, when $A_t \sim \pi(S_t, \bullet, \theta)$

$$\mathbb{E}[\nabla_\theta \log(\pi(S_t, A_t, \theta))|S_t] = \mathbf{0}$$

- For any baseline function $B(s)$, consider the update

$$\theta \leftarrow \theta + \alpha\gamma^t \nabla_\theta \log(\pi(S_t, A_t; \theta))[\widehat{Q}(S_t, A_t; \omega) - B(S_t)]$$

- The **mean** of gradient is the same without baseline
- However, the **variance** of the gradient would be smaller with a properly chosen B

Variance Reduction Using a Baseline (Cont'd)

- Consider the baseline that minimizes the variance of the gradient
- For any random variable $\boldsymbol{Z}$, the mean $\mathbb{E}\boldsymbol{Z}$ minimizes $\arg\min_{\boldsymbol{z}} \mathbb{E}(\boldsymbol{Z} - \boldsymbol{z})^2$
- To minimize variance of the gradient $\nabla_{\boldsymbol{\theta}} \log(\boldsymbol{\pi}(\boldsymbol{S_t}, \boldsymbol{A_t}; \boldsymbol{\theta}))[\widehat{\boldsymbol{Q}}(\boldsymbol{S_t}, \boldsymbol{A_t}; \boldsymbol{\omega}) - \boldsymbol{B}(\boldsymbol{S_t})]$, the baseline is set to the conditional mean of Q-function given the state
- i.e., $\boldsymbol{B}(\boldsymbol{s}) = \sum_{\boldsymbol{a}} \boldsymbol{\pi}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta})\widehat{\boldsymbol{Q}}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\omega})$, e.g., the estimated state-value

Policy Gradient Using Advantage Function

- Advantage function: $\boldsymbol{A}^{\pi(\bullet;\theta)}(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{Q}^{\pi(\bullet;\theta)}(\boldsymbol{s}, \boldsymbol{a}) - \boldsymbol{V}^{\pi(\bullet;\theta)}(\boldsymbol{s})$
- Policy gradient based on advantage function

$$\nabla_{\boldsymbol{\theta}} \boldsymbol{J}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\boldsymbol{\mu}^{\pi(\bullet;\theta)}} \nabla_{\boldsymbol{\theta}} \log(\boldsymbol{\pi}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta}))\boldsymbol{A}^{\pi(\bullet;\theta)}(\boldsymbol{s}, \boldsymbol{a})$$

- The advantage function reduces the variance of policy gradient

An Approach for Estimating Advantage Function

- The critic may compute estimators of both value functions

$$\widehat{\boldsymbol{Q}}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\omega}) \text{ for } \boldsymbol{Q}^{\boldsymbol{\pi}(\bullet;\boldsymbol{\theta})}(\boldsymbol{s}, \boldsymbol{a})$$

  and

$$\widehat{\boldsymbol{V}}(\boldsymbol{s}; \boldsymbol{\omega}) \text{ for } \boldsymbol{V}^{\boldsymbol{\pi}(\bullet;\boldsymbol{\theta})}(\boldsymbol{s})$$

  which can be done by standard methods such as TD learning

- The estimator of the advantage function

$$\widehat{\boldsymbol{A}}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\omega}) = \widehat{\boldsymbol{Q}}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\omega}) - \widehat{\boldsymbol{V}}(\boldsymbol{s}; \boldsymbol{\omega})$$

Another Approach

- $r + \gamma V^{\pi(\bullet;\theta)}(s') - V^{\pi(\bullet;\theta)}(s)$ is **unbiased** to $A^{\pi(\bullet;\theta)}(s, a)$

$$
\begin{aligned}
& \mathbb{E}[r + \gamma V^{\pi(\bullet;\theta)}(s') - V^{\pi(\bullet;\theta)}(s) | a, s] \\
= \; & \mathbb{E}[r + \gamma V^{\pi(\bullet;\theta)}(s') - Q^{\pi(\bullet;\theta)}(s, a) + Q^{\pi(\bullet;\theta)}(s, a) - V^{\pi(\bullet;\theta)}(s) | a, s] \\
= \; & Q^{\pi(\bullet;\theta)}(s, a) - V^{\pi(\bullet;\theta)}(s) = A^{\pi(\bullet;\theta)}(s, a)
\end{aligned}
$$

- As such,

$$
\nabla_\theta J(\theta) = \mathbb{E}_{(s,a)\sim\mu^{\pi(\bullet;\theta)}} \nabla_\theta \log(\pi(s, a; \theta))[r + \gamma V^{\pi(\bullet;\theta)}(s') - V^{\pi(\bullet;\theta)}(s)]
$$

- No need to estimate the advantage. It suffices to estimate the state-value and use the estimator to compute the policy gradient

Advantage Policy Gradient Methods

- The policy gradient

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{(s,a)\sim\mu^{\pi(\bullet;\theta)}} \nabla_{\theta} \log(\pi(s, a; \theta)) A^{\pi(\bullet;\theta)}(s, a)$$

- Gradient-based method

$$\theta \leftarrow \theta + \alpha\gamma^{t} \nabla_{\theta} \log(\pi(S_t, A_t; \theta)) \widehat{A}(S_t, A_t; \omega)$$

- Examples:
  - MC: $\widehat{A}(S_t, A_t; \omega) = G_t - \widehat{V}(S_t; \omega)$
  - TD: $\widehat{A}(S_t, A_t; \omega) = R_t + \gamma \widehat{V}(S_{t+1}; \omega) - \widehat{V}(S_t; \omega)$

Summary

Value Function Approximation \ Policy Function Approximation	No	Yes
No	Value-based (tabular)	REINFORCE
Yes	Value-based	Actor-Critic

- **Value-based**
  - SARSA
  - Tabular Q-learning
  - Fitted Q-iteration
  - Deep Q-network
- **REINFORCE**
  - No value function
  - Learn policy
- **Actor-critic**
  - Learn value
  - Learn policy
- **Advantage actor-critic**
  - Variance reduction

Lecture Outline

Recap: Planning vs Learning

Two fundamental problems in sequential decision making

- **Planning**
  - A model of the environment (e.g., state transition, reward function) is **known**
  - The agent performs computations with its model, **without** any external interaction
  - a.k.a. deliberation, reasoning, introspection, pondering, thought, search
- **Learning**
  - The environment is initially **unknown**
  - The agent **interacts** with the environment
  - The agent **learns** the optimal policy from experience

RL Algorithms We Have Covered So Far

- **Dynamic Programming** (Lecture 2): learn **value** from **model** (planning)
- **MC**, **TD** (Lectures 2 & 3): learn **value** from **experience** (learning)
- **Policy-based** (Lecture 4): learn **policy** from **experience** (learning)
- Today's lecture: **Model-based** RL
  - learn **model** from experience
  - use both learned model and experience to construct a **value** function or **policy**
  - combine learning with planning

What is a Model?

- A model $\mathcal{M}$ is a **representation** of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
- The state space $\mathcal{S}$ and action space $\mathcal{A}$ are usually known to us
- The discounted factor γ is **user-specified**
- Only need to learn the state transition $\mathcal{P}$

$$\mathcal{P}_{ss'}^{a} = \Pr(S_{t+1} = s' | S_t = s, A_t = a)$$

and reward function $\mathcal{R}$

$$\mathcal{R}_{s}^{a} = \mathbb{E}(R_t | S_t = s, A_t = a)$$

Model-Free v.s. Model-Based RL

- Model-based RL
  - Learn the model (e.g., reward $\mathcal{R}_s^a$ and transition $\mathcal{P}_{ss'}^a$) from experience
  - **Plan** value or policy from model or **integrate** planning with learning
- Model-free RL
  - **Learn** value or policy **without** learning the reward and transition function
  - Rely on Bellman optimality equation
  - Examples: MC, TD, Policy gradient

Model-Free v.s. Model-Based RL (Cont'd)

- **Pros** of model-based RL
- In some applications, we have a **perfect** model (e.g., Go, chess)
- Can handle **offline** data (more in the next lecture)

- **Pros** of model-free RL
- **Dimensional reduction**
- Easier to learn value than model
- # of parameters of $Q^{\pi^{\mathrm{opt}}}$: $|\mathcal{S}||\mathcal{A}|$
- # of parameters of $\mathcal{R}_s^a$: $|\mathcal{S}||\mathcal{A}|$
- # of parameters of $\mathcal{P}_{ss'}^a$: $|\mathcal{S}|^2|\mathcal{A}|$

Lecture Outline

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Model Learning

- **Goal**: estimate $\mathcal{R}_s^a$ and $\mathcal{P}_{ss'}^a$ from experience $\{S_0, A_0, R_0, \cdots, S_T\}$
- Using supervised learning

$$
\begin{aligned}
S_0, A_0 &\rightarrow R_0, S_1 \\
S_1, A_1 &\rightarrow R_1, S_2 \\
&\vdots \\
S_{T-1}, A_{T-1} &\rightarrow R_{T-1}, S_T
\end{aligned}
$$

- Learning $s, a \rightarrow r$ is a **regression** problem
- Learning $s, a \rightarrow s'$ is a **conditional density estimation** problem
- Loss function: least square/Huber loss, KL divergence
- Compute parameter that minimizes empirical loss

Models for Conditional Density Estimation

- Table lookup model
- Conditional kernel density estimation
- Gaussian process model [Williams and Rasmussen, 2006]
- Deep conditional generative learning[1]
    - mixture density network [Rothfuss et al., 2019]
    - normalising flows [Trippe and Turner, 2018]

[1] https://deepgenerativemodels.github.io/notes/index.html

Table Lookup Model

- Finite MDP model
- Count visits $N(s, a) = \sum_{t=0}^{T-1} \mathbb{I}(S_t = s, A_t = a)$ to each state-action pair

$$\widehat{\mathcal{P}}_{ss'}^{a} = \frac{1}{N(s,a)} \sum_{t=0}^{T-1} \mathbb{I}(S_t = s, A_t = a, S_{t+1} = s')$$

$$\widehat{\mathcal{R}}_{s}^{a} = \frac{1}{N(s,a)} \sum_{t=0}^{T-1} \mathbb{I}(S_t = s, A_t = a) R_t$$

- Alternatively
  - At each time step t, record experience tuple $\langle S_t, A_t, R_t, S_{t+1} \rangle$
  - To sample model, based on a state-action pair (s, a), randomly pick tuple matching $\langle s, a, \bullet, \bullet \rangle$

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Planning with Dynamic Programming

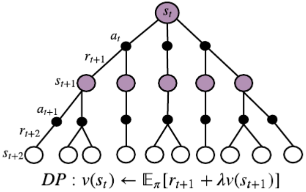
- Give a model $\langle \widehat{\mathcal{R}}, \widehat{\mathcal{P}} \rangle$
- Use dynamic programming algorithm
  - Policy iteration

$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \cdots \longrightarrow \pi^{opt} \longrightarrow V^{\pi^{opt}}$$

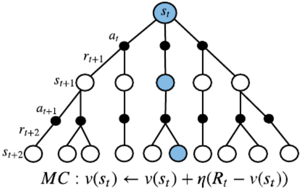
  - Value iteration

$$V^{\pi_0} \longrightarrow V^{\pi_1} \longrightarrow V^{\pi_2} \longrightarrow \cdots \longrightarrow V^{\pi^{opt}} \longrightarrow \pi^{opt}$$

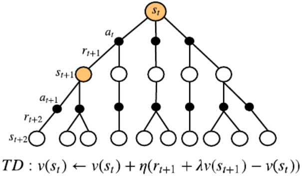
Difference From Model-Free Methods



Dynamic Programming (DP)



Monte Carlo (MC)



Temporal Difference (TD)

Planning with Model-Free RL

- A simple but powerful approach to planning
- Use the model only to **generate samples**
- **Sample** experience from model:

$$S' \sim \widehat{\mathcal{P}}^{A}_{S,\bullet} \quad \text{and} \quad R = \widehat{\mathcal{R}}^{A}_{S}$$

- Apply **model-free** RL to samples
  - deep Q-network
  - fitted Q-iteration
  - actor-critic
- This is often more **efficient** than dynamic programming-based method

Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Real and Simulated Experience

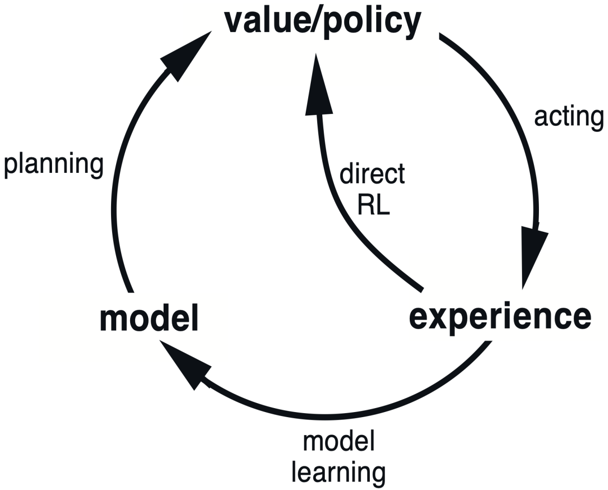
- We consider two sources of experience
- **Real experience**: Sampled from environment (true MDP)

$$\{S_0, A_0, R_0, \cdots, S_T\}$$

- **Simulated experience**: Sampled from model (estimated MDP)

$$S' \sim \widehat{\mathcal{P}}^{A}_{S,\bullet} \quad \text{and} \quad R = \widehat{\mathcal{R}}^{A}_{S}$$

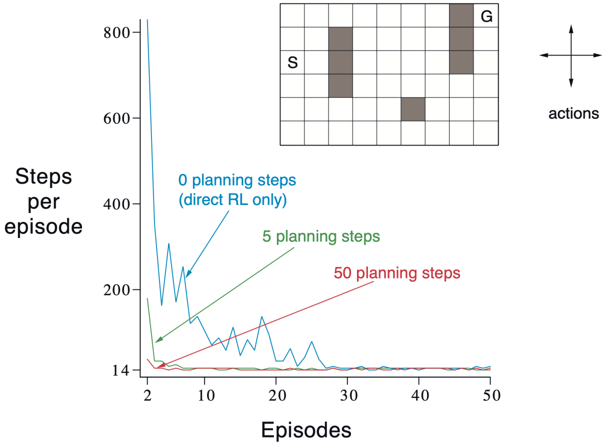
Dyna


value/policy
acting
planning
direct
RL
model
experience
model
learning

Dyna-Q Algorithm

- **Initialize** $\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})$ and **model**$(\boldsymbol{s}, \boldsymbol{a})$ for all $\boldsymbol{s}$ and $\boldsymbol{a}$
- **do** forever:
  (a) $\boldsymbol{s} \leftarrow$ current (non-terminal) state
  (b) $\boldsymbol{a} \leftarrow \varepsilon$-greedy$(\boldsymbol{s}, \boldsymbol{Q})$
  (c) Execute action $\boldsymbol{a}$; observe reward $\boldsymbol{r}$ and next state $\boldsymbol{s'}$
  (d) $\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}) \leftarrow \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}) + \alpha[\boldsymbol{r} + \gamma \max_{\boldsymbol{a'}} \boldsymbol{Q}(\boldsymbol{s'}, \boldsymbol{a'}) - \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})]$
  (e) **model**$(\boldsymbol{s}, \boldsymbol{a}) \leftarrow (\boldsymbol{r}, \boldsymbol{s'})$
  (f) Repeat $\boldsymbol{n}$ times:
      $\boldsymbol{s} \leftarrow$ random previously observed state
      $\boldsymbol{a} \leftarrow$ random action previously taking in $\boldsymbol{s}$
      $(\boldsymbol{r}, \boldsymbol{s'}) \sim$ **model**$(\boldsymbol{s}, \boldsymbol{a})$
      $\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}) \leftarrow \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}) + \alpha[\boldsymbol{r} + \gamma \max_{\boldsymbol{a'}} \boldsymbol{Q}(\boldsymbol{s'}, \boldsymbol{a'}) - \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})]$

Dyna-Q on a Simple Maze



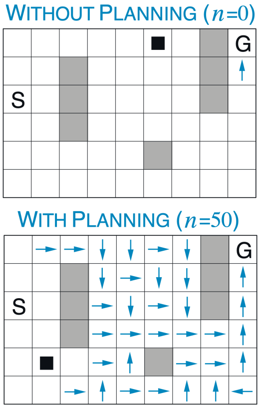


Figure: Policies found through 2nd episode. The arrows indicate greedy action; if no arrow is shown for a state, then all of its action values were equal.

- similar to "experience replay" in DQN
- use historical data more efficiently

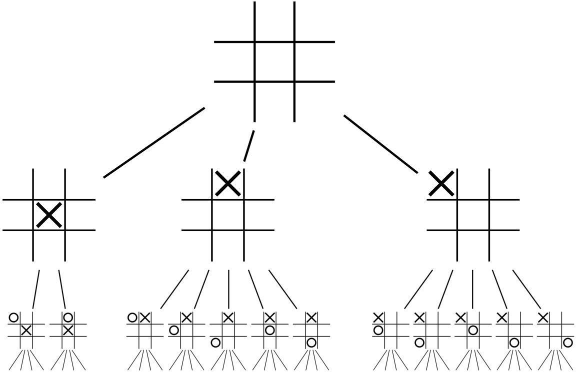
Model-based Methods

- First, we learn a **model** (reward and state transition functions) based on data
- Next, we can implement **planning** based on the learned model
- Alternatively, we can **integrate planning with learning** (Dyna)
- Finally, we can implement **Monte Carlo tree search** for decision-time planning

Two Ways of Planning

- **Background planning**
  - Planning is used well **before** an action is selected
  - Need to select actions fo each state, not **current** state
  - Examples: policy iteration and value iteration in Lecture 2
- **Decision-time planning**
  - Planning is started and completed **after** encountering each new state S_t
  - As a computation to **determine** A_t
  - On the next step planning begins anew with S_{t+1} to produce A_{t+1}, and so no

Game Trees



- Game trees: data structures to represent a game
- Exhaustive search can be **computationally intensive**
- Solutions bought by **Monte Carlo tree search**

Monte-Carlo Tree Search (Evaluation)

- Given a model $\mathcal{M}$
- Simulate K episodes from current states S_t using current policy π

$$\left\{ S_t, A_t^k, R_t^k, S_{t+1}^k, A_{t+1}^k, R_{t+1}^k, \ldots, S_T^k \right\}_{k=1}^{K} \sim \mathcal{M}, \pi$$

- Build a search tree containing visited states and actions
- Evaluate states $Q(s, a)$ by mean return of episodes from s, a

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{u=t}^{T} \mathbb{I}(S_u = s, A_u = a) G_u \to Q^{\pi}(s,a)$$

- After search is finished, select current action with maximum value in search tree

$$A_t = \operatorname*{arg\,max}_{a \in \mathcal{A}} Q(S_t, a)$$

Monte-Carlo Tree Search (Simulation)

- In MCTS, the simulation policy (rollout policy) π that simulates data **improves**
- Repeat (each simulation)
  - **Evaluate** states $Q(s, a)$ by Monte-Carlo evaluation
  - **Improve** simulation policy, e.g., by ε-greedy(Q)
  - **Monte-Carlo control** applied to **simulated experience**
- Converges to the optimal search tree, $Q(s, a) \rightarrow Q^{\pi^{\text{opt}}}(s, a)$

Lecture Outline

Case Study: the Game of Go

- Invented in China over 2500 years ago
- The **hardest** classic board game
- Much harder than chess:
    - Go has larger number of legal moves than chess ($\approx$250 v.s. $\approx$35)
    - Go involve more moves than chess ($\approx$ 150 v.s. $\approx$ 80)
    - Traditional game-tree search fails in Go

Rules of Go

- Two players place down white and black stones **alternately**
- Stones are **captured** according to simple rules

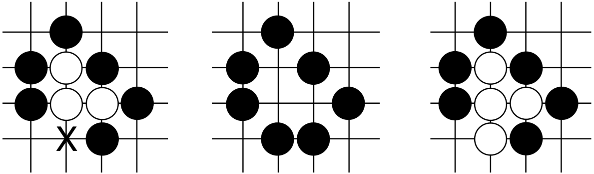


Figure: Left: the three white stones are not surrounded because point X is unoccupied. Middle: if black places a stone on X, the three white stones are captured and removed from the board. Right: if white places a stone on point X first, the capture is blocked.

- The game ends when neither player wishes to place another stone
- The player with more **territory** wins the game

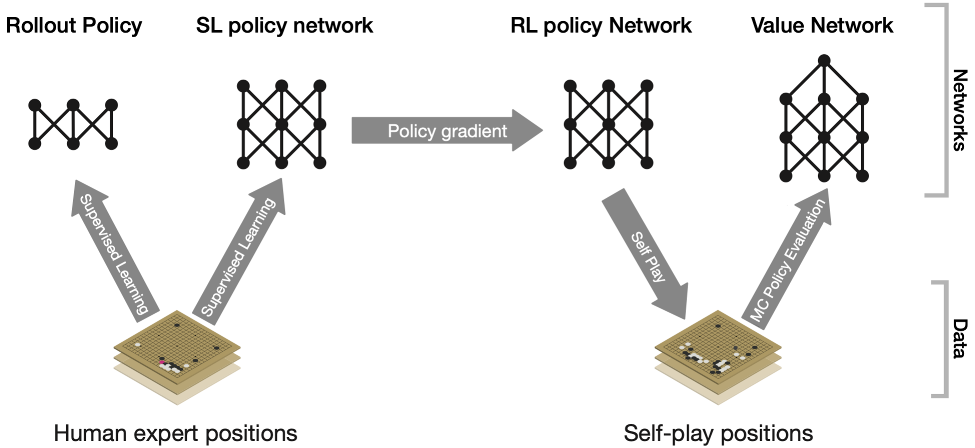
AlphaGo

THE ULTIMATE GO CHALLENGE
GAME 3 OF 3
27 MAY 2017
VS
AlphaGo
Winner of Match 3
Ke Jie
RESULT B + Res

AlphaGo Pipeline

- Based on a novel version of **Monte-Carlo tree search** (MCTS)
- Combined with a **policy** and a **value function** learned by RL with function approximation provided by deep CNN
- Simulate trajectories and generate the search tree using the **rollout** policy
- **Expand** search tree by selecting unexplored actions according to a **policy network**
- Policy network trained previously via supervised learning to predict moves contained in a database of nearly 30 million human expert moves, and updated via **self-play**
- Evaluate state-action value based on simulated returns (MC) and a **value** network
- Value network trained previously via RL

AlphaGo Pipeline (Cont'd)



Input of Neural Networks

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Policy Network

- Training the **SL policy network** took approximately 3 weeks using distributed implementation of SGD on 50 processors
- The SL policy network achieved **57**% accuracy; best accuracy achieved by other methods **44**%
- The **RL policy network** is trained on a million games in a single day
- The final RL policy won more than **80**% of games played against the SL policy
- It won **85**% of games played against a Go program using MCTS that simulated 100,000 games per move

Value Network

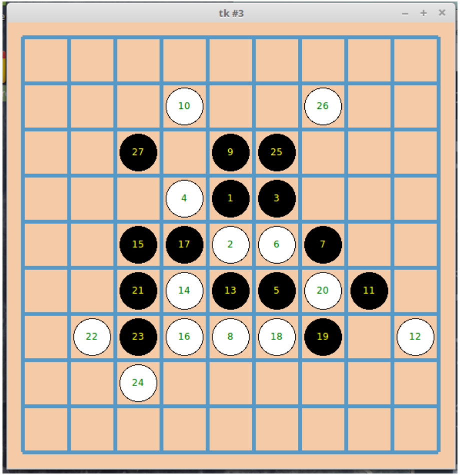
- The **value network** used **Monte Carlo policy evaluation** based on data obtained from a large number of self-play games played using the RL policy
- To avoid overfitting and instability, and to reduce the strong correlations between positions encountered in self-play, the dataset consists of **30** million positions, each chosen randomly from a unique self-play game
- Training was done using **50** million mini-batches each of **32** positions drawn from this data set
- Training took one week on **50** GPUs

Rollout Policy

- The **rollout policy** was learned prior to play by a simple linear network trained by supervised learning from a corpus of **8** million human moves
- In principle, the SL or RL policy networks could have been used in the rollouts, but the forward propagation through these deep networks took **too much time** for either of them to be used in rollout simulations
- The rollout policy network allowed approximately 1,000 complete game simulations per second to be run on each of the processing threads

AlphaGo Zero on Gomoku

https://github.com/initial-h/AlphaZero_Gomoku_MPI



Summary

- Model-based/Model free learning
- Integrating planning and learning
- Dyna-Q
- Background/Decision-time planning
- Monte Carlo Tree Search
- AlphaGo

References I

Jonas Rothfuss, Fabio Ferreira, Simon Walther, and Maxim Ulrich. Conditional density estimation with neural networks: Best practices and benchmarks. *arXiv preprint arXiv:1903.00954*, 2019.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Brian L Trippe and Richard E Turner. Conditional density estimation with bayesian normalising flows. *arXiv preprint arXiv:1802.04908*, 2018.

Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Appendix: Proof of Policy Gradient Theorem

- We focus on the discounted reward setting. Proofs in the average reward setting can be found in Sutton et al. [1999]
- Basic identities

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

$$(B) \qquad Q^\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a V^\pi(s')$$

$$(C) \qquad \nabla_\theta V^\pi(s) = \sum_a [\nabla_\theta \pi(a|s)] Q^\pi(s,a) + \sum_a \pi(a|s) [\nabla_\theta Q^\pi(s,a)]$$

$$(D) \qquad \nabla_\theta Q^\pi(s,a) = \gamma \sum_{s'} P_{s,s'}^a \nabla_\theta V^\pi(s')$$

Appendix: Proof (Cont'd)

$$\nabla_\theta V^\pi(s) \overset{(C)}{=} \sum_a [\nabla_\theta \pi(a|s)] Q^\pi(s,a) + \sum_a \pi(a|s)[\nabla_\theta Q^\pi(s,a)]$$

$$\overset{(D)}{=} \sum_a \pi(a|s)[\nabla_\theta \log(\pi(a|s))] Q^\pi(s,a) + \gamma \underbrace{\sum_{a,s'} \pi(a|s) P^a_{s,s'} \nabla_\theta V^\pi(s')}_{I}$$

Now, consider I. Similarly, we have

$$I = \sum_{a,s',a'} \pi(a|bs) P^a_{s,s'} \pi(a'|s')[\nabla_\theta \log(\pi(a'|s'))] Q^\pi(s',a')$$

$$+\gamma \sum_{a,s',a',s''} \pi(a|s) P^a_{s,s'} \pi(a'|s') P^{a'}_{s',s''} \nabla_\theta V^\pi(s'')$$

Appendix: Proof (Cont'd)

Recursively applying the first identity, we obtain

$$\nabla_\theta V^\pi(s) = \mu^{\pi(\bullet;\theta)}(s', a'; s) \nabla_\theta \log(\pi(s', a')) Q^\pi(s', a')$$

where

$$\mu^{\pi(\bullet;\theta)}(s', a'; s) = \sum_{t \geq 0} \gamma^t \pi(s', a') \mathrm{Pr}^{\pi(\bullet;\theta)}(S_t = s' | S_0 = s)$$

Questions