Artificial Intelligence

Taro Sekiyama

National Institute of Informatics (NII)
sekiyama@nii.ac.jp

Approaches to RL problems

1. **Value-based:** estimating value/Q-value functions
   A. **Model-based**
      - Dynamic programming

   B. ***Model-free***
      - Monte Carlo
      - Q-learning
2. **Policy-based:** estimating policies directly

Model-free approaches

- No need of knowledge of environments
- Common strategy: learning from ***experience***
  - Observation of what happens if we choose some action in some state
  - Finitely many observations are useful enough to estimate models
  - Note: Complete estimation of the models needs infinitely many observations

Experience

- Sequence of triples (s_i, a_i, t_i) found during interaction with environment
  - s_i the state in the i-th step
  - a_i the action taken in the i-the step
  - t_i the reward in the i-th step

Episode

- Experience from an initial state to a ***terminal*** state (i.e., desirable/undesirable state)
  - Maze: experiences from the starting point to the goal point
  - Go game: experiences of playing moves until the play finishes
- Many RL approaches try to maximize the cumulative reward of an episode
  - Note: not all RL problems have episodes

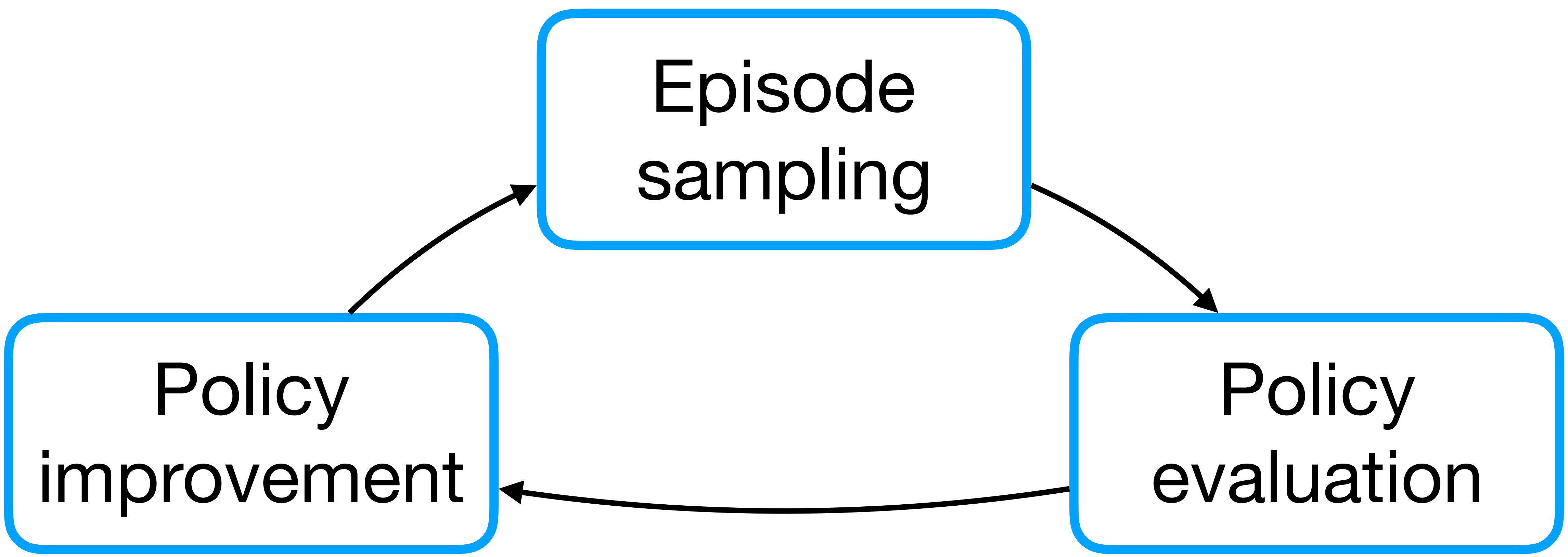
Approaches to RL problems

1. **Value-based:** estimating value/Q-value functions
   A. **Model-based**
      - Dynamic programming

   B. **Model-free**
      - *Monte Carlo*
      - Q-learning
2. **Policy-based:** estimating policies directly

Monte Carlo

- Estimating an optimal policy by sampling episodes
- Repeating the following process



Episode sampling

- Generating episodes following a policy

$$(s_0, a_0, t_0), (s_1, a_1, t_1), (s_2, a_2, t_2), \ldots, (s_{T-1}, a_{T-1}, t_{T-1}), s_T$$

  - $a_i = \pi(s_i)$
  - t_i is sampled from $r(\cdot \mid s_i, a_i)$
  - s_{i+1} is sampled from $p(\cdot \mid s_i, a_i)$
  - s_T is a terminal state

Policy evaluation

- Estimating value function V^π using the sampled episodes
- Algorithm

Initialize, for any $s \in S$,

  value function $V^\pi(s) := 0$,

  lists of cumulative rewards $R(s) := []$

For each episode $(s_0, a_0, t_0), \ldots, (s_{T-1}, a_{T-1}, t_{T-1})$

  Cumulative reward $G := 0$

  For each step i from $T - 1$ to 0

    $G := \gamma G + t_i$

    Append G to $R(s_i)$

    $V^\pi(s_i) :=$ the mean of the rewards $R(s_i)$

Policy improvement

- Generating a new policy π' from the estimated V^π
- Naive approach: generating a greedy policy

$$\pi'(s) = \arg\max_{a \in A} \mathbb{E}[V^\pi(\mathcal{S}) \mid \mathcal{S} \sim p(\cdot \mid s, a)]$$

- Problem: π' may converge to a local optimum
  - $\pi = \pi'$ but π' is not optimal

Policy improvement

- Generating a new policy π' from the estimated V^{π}
- Approach: **exploration** with small probability ϵ

$$\pi'(s) = \begin{cases} \arg\max_{a \in A} \mathbb{E}[V^{\pi}(\mathcal{S}) \mid \mathcal{S} \sim p(\,\cdot \mid s, a)] & \text{(with probability } 1 - \epsilon\text{)} \\ a & \text{(with probability } \epsilon / |A|\text{)} \end{cases}$$

  - Such π' is called a ϵ-***greedy*** policy
- Any action may appear in episodes with $\pi'(s)$
  - Infinitely many episodes enable investigation of all the states
  - Preventing policies from converging to local optimums

Problem with Monte Carlo

- Applicable only to ***episodic*** tasks
  - Not applicable to ***continuing*** tasks (tasks without terminal states)
- Not working well even for episodic tasks if episodes tend to be very long

Approaches to RL problems

1. **Value-based:** estimating value/Q-value functions
   A. **Model-based**
      - Dynamic programming
   B. **Model-free**
      - Monte Carlo
      - *Q-learning*
2. **Policy-based:** estimating policies directly

Q-learning

- Updating Q-value function Q step-by-step
- Applicable to both of episodic and continuing tasks
- Idea: Q is updated so that it satisfies the Bellman optimality equation

Review: Bellman optimality equation

For Q-value functions

$$Q(s,a) = \mathbb{E}[\mathscr{R} + \gamma \max_{a' \in A} Q(\mathcal{S}, a') \mid \\ \mathscr{R} \sim r(\cdot \mid s,a), \mathcal{S} \sim p(\cdot \mid s,a)]$$

- Q satisfies the equation iff Q is optimal
- The value of (s,a) is maximized when the "best" action a' is performed in the next state $\mathcal{S}$

Update of Q-value functions

More well-estimated Q-value

Q-value estimated so far

$$Q(s,a) := Q(s,a) + \eta\left[t + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Update with difference

where

- s : the current state
- a : the action taken in s by policy π
- t : the reward when doing a in s
- s' : the next state after doing a in s
- η : the learning rate

Algorithm

Initialize, for any $s \in S$, $a \in A$,

Q-value function $Q(s, a) := 0$

Loop

$s :=$ an initial state, $i := 0$

Loop until s becomes a terminal state or

i exceeds a given limitation

$a := \pi(s)$ (π is the ϵ-greedy policy derived from Q)

$s' :=$ the state after doing a in s

$t :=$ the reward obtain when doing a in s

$$Q(s, a) := Q(s, a) + \eta[t + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

$s := s',\ i := i + 1$

Approaches to RL problems

1. **Value-based:** estimating value/Q-value functions
   A. **Model-based**
      - Dynamic programming
   B. **Model-free**
      - Monte Carlo
      - Q-learning
2. ***Policy-based:*** estimating policies directly

Problem with value-based approaches

- The value of each state s is learned only from experiences for s
- Cannot be learned from states similar to s
  - **Ex:** Cart-pole problem
    - The same action should be taken in similar states
    - Experiences in a state are also useful for other similar states

Policy approximation

- Policy π_θ is approximated by some machine learning models with parameters θ
  - E.g., neural networks

😁 θ can be tuned so that the same action is taken in similar states

😁 All experiences contribute to optimizing θ

- Experiences in a state contribute to estimating values of other similar states

Policy gradient

- Objective: finding θ that maximizes the value function V^{π_θ}
- Idea: updating parameters θ so that the value function V^{π_θ} is increased
- Approach: repeating the two steps
  1. Sampling episodes and estimating V^{π_θ}
  2. Applying gradient ascent to increase V^{π_θ}

References

- Book “Reinforcement Learning — An Introduction (second edition)”
  by Sutton and Barto, 2018
  Online: http://incompleteideas.net/book/the-book.html