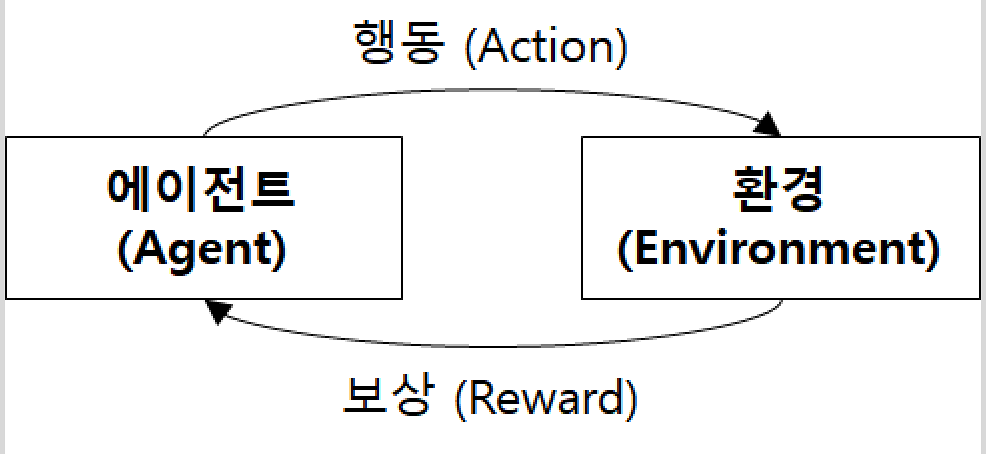
RL 스터디

1주차

강화학습 이란

- 어떤 환경 안에서 정의된 에이전트가 현재의 상태를 인식하여, 선택 가능한 행동들 중 보상을 최대화하는 행동 혹은 행동 순서를 선택하는 방법이다. – 위키피디아

- 여기서 환경은 MDP(POMDP)로 본다.



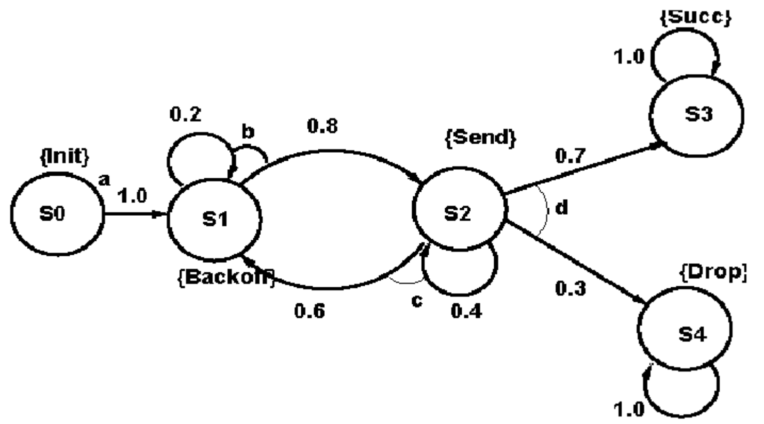
MDP, Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

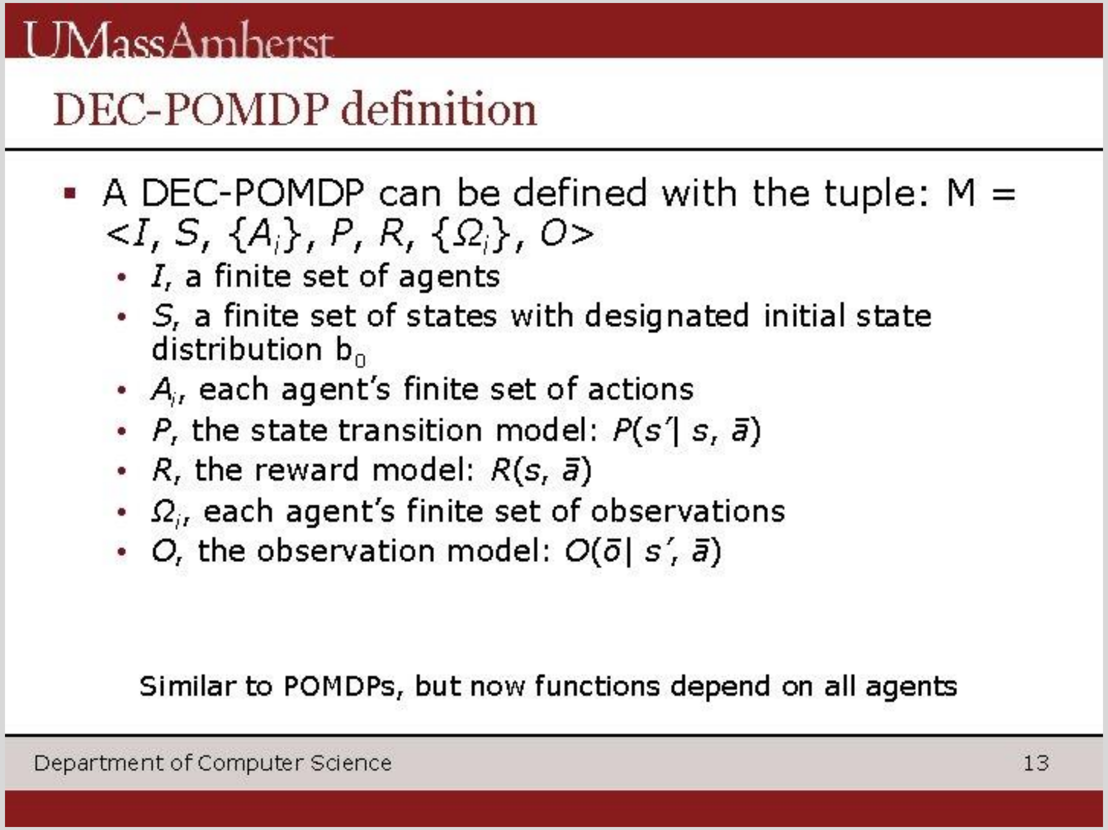
Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.



POMDP



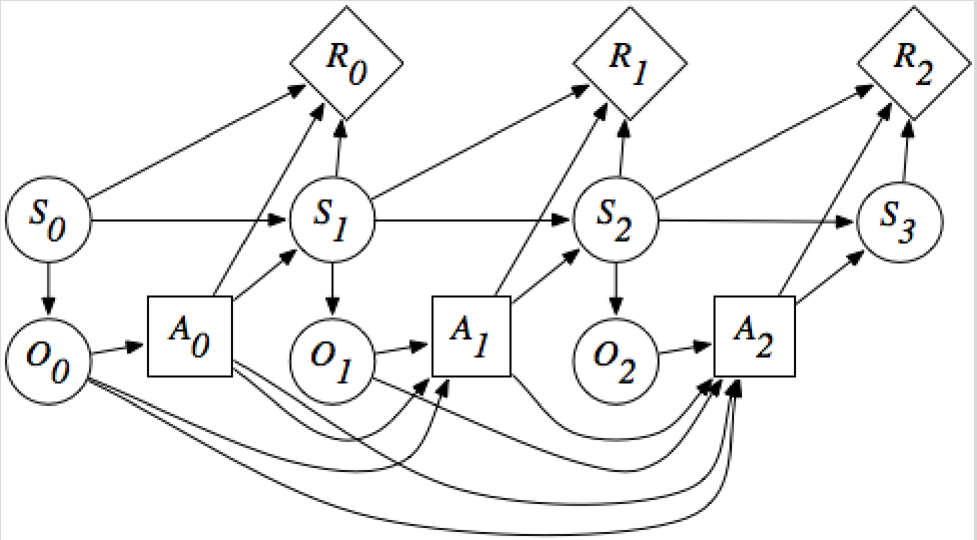
UMassAmherst

DEC-POMDP definition

- A DEC-POMDP can be defined with the tuple: M = $<I, S, \{A_i\}, P, R, \{\Omega_i\}, O>$
  - I, a finite set of agents
  - S, a finite set of states with designated initial state distribution b_0
  - A_i, each agent's finite set of actions
  - P, the state transition model: $P(s' | s, \bar{a})$
  - R, the reward model: $R(s, \bar{a})$
  - Ω_i, each agent's finite set of observations
  - O, the observation model: $O(\bar{o} | s', \bar{a})$

Similar to POMDPs, but now functions depend on all agents

Department of Computer Science 13



Value function and return

Definition

The *return* G_t is the total discounted reward from time-step t.

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

Value function and return

The value function $v(s)$ gives the long-term value of state s

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

Action Value function - Q

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Bellman equation

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$\begin{aligned} v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right] \end{aligned}$$

Bellman equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right]$$

Optimal Bellman equation

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_{\pi} v_\pi(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_{\pi} q_\pi(s, a)$$

optimization

- DP
- Monte Carlo
- Temporal Difference

DP – policy iteration

Dp는 완벽한 MDP 환경일때 최적의 정책을 결정하는데 도움을 주는 알고리즘
환경을 정확히 알지 못하기에 상호작용을 통해 최적의 정책을 결정

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
     $\Delta \leftarrow 0$
     Loop for each $s \in \mathcal{S}$:
       $v \leftarrow V(s)$
       $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
     old-action $\leftarrow \pi(s)$
     $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
     If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

DP – value iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\quad \Delta \leftarrow 0$
| $\quad$ Loop for each $s \in \mathcal{S}$:
| $\qquad v \leftarrow V(s)$
| $\qquad V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\quad \pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Monte Carlo Predict

- 확률적 샘플링을 통한 학습

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- Monte-Carlo policy evaluation uses *empirical* *mean* return instead of *expected* return

Algorithm 1: First-Visit MC Prediction

Input: policy π, positive integer *num_episodes*
Output: value function V ($\approx v_\pi$, if *num_episodes* is large enough)
Initialize $N(s) = 0$ for all $s \in \mathcal{S}$
Initialize Returns$(s) = 0$ for all $s \in \mathcal{S}$
for *episode* $e \leftarrow 1$ ***to*** $e \leftarrow$ *num_episodes* **do**
  Generate, using π, an episode $S_0, A_0, R_1, S_1, A_1, R_2 \dots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for *time step* $t = T-1$ ***to*** $t = 0$ *(of the episode e)* **do**
    $G \leftarrow G + R_{t+1}$
    if *state* S_t *is* ***not*** *in the sequence* $S_0, S_1, \dots, S_{t-1}$ **then**
      Returns$(S_t) \leftarrow$ Returns$(S_t) + G_t$
      $N(S_t) \leftarrow N(S_t) + 1$
  end
end
$V(s) \leftarrow \frac{\text{Returns}(s)}{N(s)}$ for all $s \in \mathcal{S}$
return V

Algorithm 2: Every-Visit MC Prediction

Input: policy π, positive integer *num_episodes*
Output: value function V ($\approx v_\pi$, if *num_episodes* is large enough)
Initialize $N(s) = 0$ for all $s \in \mathcal{S}$
Initialize Returns$(s) = 0$ for all $s \in \mathcal{S}$
for *episode* $e \leftarrow 1$ ***to*** $e \leftarrow$ *num_episodes* **do**
  Generate, using π, an episode $S_0, A_0, R_1, S_1, A_1, R_2 \dots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for *time step* $t = T-1$ ***to*** $t = 0$ *(of the episode e)* **do**
    $G \leftarrow G + R_{t+1}$
    Returns$(S_t) \leftarrow$ Returns$(S_t) + G_t$
    $N(S_t) \leftarrow N(S_t) + 1$
  end
end
$V(s) \leftarrow \frac{\text{Returns}(s)}{N(s)}$ for all $s \in \mathcal{S}$
return V

Monte Carlo Control

- 확률적 샘플링을 통한 학습

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

  $\pi \leftarrow$ an arbitrary ε-soft policy

  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

      Append G to $Returns(S_t, A_t)$

      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Temporal Difference

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A; observe reward, R, and next state, S'
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until S is terminal

Algorithm 1. Classical Temporal Difference Learning.

Initialize $Q(s, a)$ for all $s \in S, a \in A$ arbitrarily;
repeat (for each episode):
    Initialize s;
    repeat (for each step of episode):
        Select action a in state s based on ε-greedy policy;
        Execute action a, observe next state s', and receive reward $R^a_{ss'}$;
        Calculate TD error:
            $\delta_Q = R^a_{ss'} + \gamma \max_{a'} Q(s', a') - Q(s, a)$, ▷ Q-learning.
            $\delta_{SARSA} = R^a_{ss'} + \gamma Q(s', a') - Q(s, a)$; ▷ SARSA.
        Update the table entry: $Q(s, a) \leftarrow Q(s, a) + \alpha\delta$;
        $s \leftarrow s'$;
    until s is terminal.
until end of the episodes.

SARSA

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Figure 6.9: Sarsa: An on-policy TD control algorithm.

TD control

- N –step TD
- Lambda TD
- ……..

Off-policy VS on-policy

- On policy의 경우 자신의 정책에 따른 행동에 대해 평가하고 이를 바탕으로 학습
- Off policy는 행동하는 행동 정책과 이를 평가하는 타겟 정책이 다르다.

- On policy => 현재 정책에 의해 얻은 행동을 기반으로 학습 => 샘플 효율성 감소 + 국소 최적화에 빠질 수 있다.
- Off policy => 지금 행동과 정책을 평가하는 궤적이 달라 좀더 다양한 궤적을 평가 탐색을 늘리는데 도움 =>

샘플 효율성 증가 하지만 정책이 달라, bias를 줄이기 위해 important sampling 필요

Importance Sampling

Importance Sampling

- Estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

Monte Carlo Control

Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from μ to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \dots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value towards *corrected* return

$$V(S_t) \leftarrow V(S_t) + \alpha \left({\color{red} G_t^{\pi/\mu}} - V(S_t) \right)$$

- Cannot use if μ is zero when π is non-zero
- Importance sampling can dramatically increase variance

Monte Carlo Control

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
- $Q(s,a) \in \mathbb{R}$ (arbitrarily)
- $C(s,a) \leftarrow 0$
- $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
- $b \leftarrow$ any soft policy
- Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$
- $W \leftarrow 1$
- Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)} [G - Q(S_t, A_t)]$
  - $\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
  - If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
  - $W \leftarrow W \frac{1}{b(A_t|S_t)}$

TD

Importance Sampling for Off-Policy TD

- Use TD targets generated from μ to evaluate π
- Weight TD target $R + \gamma V(S')$ by importance sampling
- Only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right)$$

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Q-Learning

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$