MDP

천정민

서울과학기술대학교
SEOULTECH SEOUL NATIONAL UNIVERSITY OF SCIENCE & TECHNOLOGY

1 I. Introduction and Preview

2 II. Setting

3 III. Policy evaluation

I. Introduction and Preview

Taxonomy

Model	Component
MC	$\mathcal{S}, \mathbf{P}$
MRP	$\mathcal{S}, \mathbf{P}, R, \gamma$
DP	$\mathcal{S}, R, \gamma, \mathcal{A}$
MDP	$\mathcal{S}, \mathbf{P}, R, \gamma, \mathcal{A}$

- See how we have incremented the notions for
    - Stochasticity: $\mathbf{P}$
    - Reward: R and γ
    - Action: $\mathcal{A}$

Goal

- MRP
  - The aim was to find
    - $V(s)$ for infinite problem ($= V_t(s) = \mathbb{E}[G_t|S_t = s]$)
    - $V_t(s)$ for finite problem ($= \mathbb{E}[G_t|S_t = s]$)
- DP/MDP
  - Action is introduced and the actions are governed by policy. Following tasks are desired.
  - [1. Policy evaluation]
    - We need to be able to evaluate $V^\pi(s)$ for a fixed π.
    - This is also called as prediction in reinforcement learning.
  - [2. Optimal value function]
    - We want to be able to evaluate $V^{\pi^*}(s)$, where π^* is the optimal policy.
    - The quantity, $V^{\pi^*}(s)$, is the value function for optimal policy, or shortly as optimal value function.
  - [3. Optimal policy]
    - We want to find the optimal policy π^*.
    - This is also called as control in reinforcement learning.

II. Setting

A climbing skier's problem.

'A skier is climbing a steep mountain covered in snow. If the skier attempts to climb slowly (**normal mode**) she can advance 10 meters per minute, and if she attempts to climb fast (**speed mode**) she can advance 20 meters per minute with success probability of 0.9. The remaining probability of 0.1 implies that she falls and slides back by 10 meters. If she falls at the 0 meter point, she comes back to 0 meter point again.' 'In the **normal mode**, 1 unit of her energy is

consumed. In the **speed mode**, 1.5 unit of her energy is consumed. Her mission starts at the 0 meter point and ends as soon as she reaches to 70 meter point. She decides her climing mode on every minute. Her goal is to reach 70 meter point while consuming minimal amount of energy.' 'She know that there stands a helper at the 40 meter points. Every time she starts a

minute at the 40 meter points, the helper pushes her so that she will only consume 0 unit of her energy in the **normal mode** and consume 0.5 unit of her energy in the **speed mode**.'

'스키어가 눈으로 덮인 가파른 산을 오르고 있다. 천천히 오르려고하면 (**normal mode**) 분당 10 미터를 전진 할 수 있고, 빠르게 오르려고하면 (**speed mode**) 성공 확률 0.9로 분당 20 미터를 전진 할 수 있다. 남은 확률 0.1은 그녀가 넘어져서 10 미터 뒤로 미끌어지는 경우를 의미한다. 0 미터 지점에서 넘어지면 0미터 지점으로 돌아온다.' '**normal mode**에서는 1 단위의 에너지가 소비된다. **speed mode** 에서는

1.5 단위의 에너지가 소비된다. 그녀의 임무는 0 미터 지점에서 시작하여 70 미터 지점에 도달하는 즉시 종료된다. 그녀는 매분마다 클라이밍 모드를 결정하며, 그녀의 목표는 최소한의 에너지를 소비하면서 70 미터 지점에 도달하는 것이다.' '40미터 지점에는 조력자가 있다. 그녀가 40미터 지점에서 출발한다면,

조력자가 그녀를 밀어주기에 그녀는 **normal mode**에서는 0 단위, **speed mode**에서는 0.5 단위의 에너지만을 소비한다.'

MDP Formulation

- All components of the tuple in $(\mathcal{S}, \mathbf{P}, R, \gamma, \mathcal{A})$ needs to be defined.

State space ($\mathcal{S}$)

- Let S_t be the meter point where she stands at time t.
- $\mathcal{S} = \{0, 10, 20, 30, 40, 50, 60, 70\}$

Action space ($\mathcal{A}$)

- Let A_t be the climbing mode at time t.
- Let $\mathcal{A} = \{a_1, a_2\}$
  - a_1 implies the normal mode
  - a_2 implies speed mode.

Reward (R)

- Reward r_t depends on
  - the current state S_t (as in MRP) and
  - her action A_t. (as in DP)
- In other words, reward r_t is a function of S_t and A_t.
- In light of this, MDP defines the reward function $R(\cdot)$ as a bi-variate function, namely,

$$R(s, a) = \mathbb{E}[r_t | S_t = s, A_t = a]$$

- Notice the expectation operator above. In general, the reward function may depend not only on S_t and A_t but also some randomness. In the skier's problem, randomness does not affect the reward function.

- Specifically,
  - $R(s, a_1) = 1$, for all $s \neq 40$
  - $R(40, a_1) = 0$
  - $R(s, a_2) = 1.5$, for all $s \neq 40$
  - $R(40, a_2) = 0.5$
  - $R(70, \cdot) = 0$

(Return) (G_t)

- The return is $G_t = \sum_{i=t}^{\infty} \gamma^{t-i} r_i$ same as before.
- Since $\gamma = 1$, we simply have $G_t = \sum_{i=t}^{\infty} r_i$.

Transition probability ($\mathbf{P}$)

- As discussed, transition can be denoted as a functional form of

$$S_{t+1} = f(S_t, A_t, \text{some randomness})$$

- Remind that MC's transition probability matrix $\mathbf{P}$ was formed in the way that
  - i) S_t forms rows
  - ii) S_{t+1} forms columns
- Now with a newly introduced dimension A_t, one can't possibly represent the transition in a nice 2-dimensional array.

- Only if we fix actions for all states (or, equivalently, fix a policy), we can represent the transition in a matrix and a diagram.
- The next page presents transition diagram when
  1) a policy is fixed with normal mode action for all states
  2) a policy is fixed with speed mode action for all states.
  3) a policy is fixed with normal-speed randomization with equal probability for all states.

Normal Mode

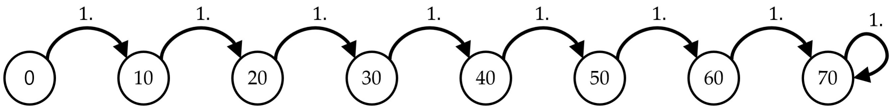

그림 1: Transition diagram if normal mode is chosen for every state

Speed Mode

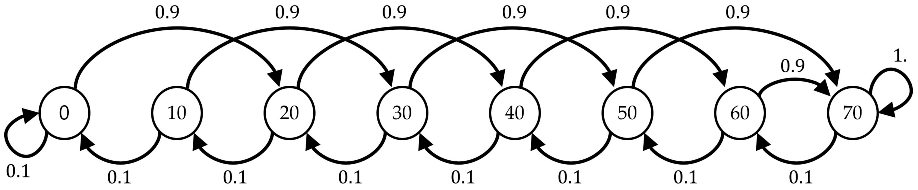

그림 2: Transition diagram if speed mode is chosen for every state

(Policy) (π)

- We shall define
  - π^{normal}: a policy choosing normal mode in all states.
  - π^{speed}: a policy choosing speed mode in all states.
  - π^{50}: a policy choosing normal mode or speed mode equally likely in all states.

- Given a fixed policy,
  - Policy
    - S_t determines A_t.
    - (If you know where you are, then you know what to do.)
  - Reward
    - S_t, along with determined A_t, determines reward function $R(s, a)$.
    - (If you know where you are, then you know what to do, and then you have a clear expectation how much you will be rewarded.)
  - With a fixed policy, S_t determines not only action but also reward!. It means that evaluating $V^{\pi}(s)$ is not any different from evaluation process of MRP.
  - **Under a fixed policy, evaluating value function in MDP is reduced to that in MRP.**

- Summary
  - [1. Policy evaluation] is to evaluate $V(s)$ for all states given a policy.
  - Based on the above discussion, this process is not any different from that of MRP.

III. Policy evaluation

Development

- From MRP, we had the following Bellman's equation.

$$\begin{aligned} V(s) &= R(s) + \gamma \sum_{\forall s'} \mathbf{P}_{ss'} V(s') \quad \texttt{(previous section)} \\ v &= R + \gamma \mathbf{P} v \quad \text{(in a vector form)} \end{aligned}$$

- We need come up with Bellman's equation for MDP as follows (Notation π must be added).

$$V^{\pi}(s) = R^{\pi}(s) + \gamma \sum_{\forall s'} \mathbf{P}^{\pi}_{ss'} V^{\pi}(s')$$

From $V^\pi(s) = R^\pi(s) + \gamma \sum_{\forall s'} \mathbf{P}^\pi_{ss'} V^\pi(s')$, we further need two remarks.

1. Reward function $R^\pi(s)$

- $R^\pi(s)$ is equal to $R(s, \pi(s))$ if $\pi(s)$ is an single action.
- However, she may randomly choose among multiple actions on a state.
- Including the random policy, it should be the sum of $R(s, a)$ weighted by the action distribution $\pi(a|s)$ as follows.

$$R^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) R(s, a) \tag{1}$$

- $\pi(s)$ returns a single action on the state s under the policy π.
- $\pi(a|s)$ returns the probability of choosing an action a on the state s under policy π.

From $V^{\pi}(s) = R^{\pi}(s) + \gamma \sum_{\forall s'} \mathbf{P}^{\pi}_{ss'} V^{\pi}(s')$, we further need two remarks.

2. Transition probability $\mathbf{P}^{\pi}_{ss'}$

- Likewise, if a policy allows randomized action, then this notation is not so straight-forward.
- Again, the probability should be weighted by the action distribution $\pi(a|s)$ as follows.

$$\mathbf{P}^{\pi}_{ss'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathbf{P}(s'|s, a), \tag{2}$$

where $\mathbf{P}(s'|s, a) = \mathbb{P}(S_{t+1} = s'|S_t = s, A_t = a)$.

Iterative estimation of state-value function for a given policy.

$$v_{i+1} \leftarrow R + \gamma \mathbf{P} v_i$$

- The pseudo code for MDP.

```
0: For a fixed \pi,
0: For all states s,
     define R^{\pi}(s) using Eq.(1)
0: For all states s,s',
     define P^{\pi}_{ss'} using Eq.(2)
1: Let epsilon <- 10^{-8}
2: Let v_0 <- zero vector
3: i <- 1
4: While ||v_i-v_{i-1}|| > epsilon # any norm
5:   v_{i+1} <- R + \gamma*P*v_{i}
6:   i <- i+1
7: Return v_{i+1}
```

Discussion

- With a different π, the current approach have to hard-code $R^\pi(s)$ and $P^\pi_{ss'}$, again.
- We need a more automated policy evaluation function such as `policy_eval()` that takes an arbitrary policy as an input and produces an estimate for state-value function as an output.

- Namely,

```
function policy_eval(\pi) {
  # 1. input
  #   \pi                 # dimension?
  # 2. preparation
  #   R <- reward_fn(\pi)  # dimension?
  #   P <- transition(\pi) # dimension?
  # 3. iterate until converge
  #   v_{i+1} <- R + \gamma*P*v_{i}
  # 4. output
  #   V^{\pi}
}
```

- Components
  1. $\pi : \mathcal{S} \to \mathcal{A}$
  2. $R^\pi : \mathcal{S} \to \mathbb{R}$
  3. $P^\pi : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$

Preparation for the components 1-3

1. $\pi : \mathcal{S} \to \mathcal{A}$
   - The input π should be an array whose size is $|S| \times |A|$.
   - For example, π^{speed} is as follows:

- A policy can be described in such a tabular form, as long as $|S|$ and $|A|$ are small enough.
- Modern reinforcement learning tackles large-scaled problems by functional approximation notably using deep neural network.

```python
import numpy as np
import pandas as pd

# 1. π : S → A
# The input π should be an array whose size is |S| × |A|
# For example, π^speed is as follows:

states = [str(i) for i in range(0, 80, 10)]
pi_speed = np.column_stack([
    np.zeros(len(states)),  # rep(0, length(states))
    np.ones(len(states))    # rep(1, length(states))
])

# Create DataFrame with row and column names
pi_speed = pd.DataFrame(pi_speed,
                        index=states,
                        columns=["normal", "speed"])

print(pi_speed.T)
```

```
##          0   10   20   30   40   50   60   70
## normal  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
## speed   1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
```

2. $R^\pi : \mathcal{S} \to \mathbb{R}$

$$R^\pi(s) = \sum_a \pi(s,a) R(s,a)$$

```
# Let 'R_s_a' be R(s,a)
R_s_a = np.array([
    [-1, -1, -1, -1, 0.0, -1, -1, 0],
    [-1.5, -1.5, -1.5, -1.5, -0.5, -1.5, -1.5, 0]
]).T

# Create DataFrame with row and column names
R_s_a = pd.DataFrame(R_s_a,
                     index=states,
                     columns=["normal", "speed"])

print(R_s_a)
```

```
##     normal  speed
## 0     -1.0   -1.5
## 10    -1.0   -1.5
## 20    -1.0   -1.5
## 30    -1.0   -1.5
## 40     0.0   -0.5
## 50    -1.0   -1.5
## 60    -1.0   -1.5
## 70     0.0    0.0
```

```
def reward_fn(given_pi):
    R_s_a_matrix = np.array([
        [-1, -1, -1, -1, 0.0, -1, -1, 0],
        [-1.5, -1.5, -1.5, -1.5, -0.5, -1.5, -1.5, 0]
    ]).T

    R_pi = np.sum(given_pi * R_s_a_matrix, axis=1)
    return R_pi

result = reward_fn(pi_speed.values)
print(result)
```

```
## [-1.5 -1.5 -1.5 -1.5 -0.5 -1.5 -1.5  0. ]
```

3. $P^{\pi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$

$$P^{\pi}_{ss'} = \sum_{a} \pi(s,a) P^{a}_{ss'}$$

```
P_normal = np.array([
    [0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 1]
])

P_normal = pd.DataFrame(P_normal, index=states, columns

P_speed = np.array([
    [0.1, 0, 0.9, 0, 0, 0, 0, 0],
    [0.1, 0, 0, 0.9, 0, 0, 0, 0],
    [0, 0.1, 0, 0, 0.9, 0, 0, 0],
    [0, 0, 0.1, 0, 0, 0.9, 0, 0],
    [0, 0, 0, 0.1, 0, 0, 0.9, 0],
    [0, 0, 0, 0, 0.1, 0, 0, 0.9],
    [0, 0, 0, 0, 0, 0.1, 0, 0.9],
    [0, 0, 0, 0, 0, 0, 0, 1]
])
P_speed = pd.DataFrame(P_speed, index=states, columns=states)
print(P_speed)
```

```
def transition(given_pi, states, P_normal, P_speed):
    P_out = np.zeros((len(states), len(states)))

    for i in range(len(states)):
        action_dist = given_pi[i, :]

        for j in range(len(states)):
            P_out[i, j] = (action_dist[0] *
                           P_normal.values[i, j] +
                           action_dist[1] *
                           P_speed.values[i, j])

    return P_out

P_out = transition(pi_speed.values, states,
        P_normal, P_speed)
print(P_out)
```

```
## [[0.1 0.  0.9 0.  0.  0.  0.  0. ]
##  [0.1 0.  0.  0.9 0.  0.  0.  0. ]
##  [0.  0.1 0.  0.  0.9 0.  0.  0. ]
##  [0.  0.  0.1 0.  0.  0.9 0.  0. ]
##  [0.  0.  0.  0.1 0.  0.  0.9 0. ]
##  [0.  0.  0.  0.  0.1 0.  0.  0.9]
##  [0.  0.  0.  0.  0.  0.1 0.  0.9]
##  [0.  0.  0.  0.  0.  0.  0.  1. ]]
```

Implementation, finally.

```
# Implementation, finally.
def policy_eval(given_pi):
    R = reward_fn(given_pi)
    P = transition(given_pi, states, P_normal, P_speed)
    gamma = 1.0
    epsilon = 10**(-8)
    v_old = np.zeros(8)

    while True:
        v_new = R + gamma * P @ v_old
        if np.max(np.abs(v_new - v_old)) < epsilon:
            break
        v_old = v_new

    return v_new.T

print(policy_eval(pi_speed.values))
```

```
## [-5.80592905 -5.2087811  -4.13926239 -3.47576467 -2.35376031 -1.73537603
##  -1.6735376   0.        ]
```

```
"0827-Jeongmin"
```