Lecture E2. MDP with Model 2

Sim, Min Kyu, Ph.D., mksim@seoultech.ac.kr



- 1. Recap
- II. Policy improvement
- III. Policy iteration

I. Recap

policy_eval()

```
_ gamma <- 1
states <- as.character(seq(0, 70, 10))
P normal <- matrix(c(0,1,0,0,0,0,0,0,0,</pre>
                       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),
   nrow = 8, ncol = 8, byrow = TRUE,
   dimnames = list(states, states))
• P speed <- matrix(c(.1, 0,.9, 0, 0, 0, 0, 0,</pre>
                      .1, 0, 0, 9, 0, 0, 0, 0,
                       0,.1, 0, 0,.9, 0, 0, 0,
                       0, 0, .1, 0, 0, .9, 0, 0,
                       0. 0. 0. 1. 0. 0. 9. 0.
                       0, 0, 0, 0, 1, 0, 0, 9,
                       0, 0, 0, 0, 0, 1, 0, 9,
                       0, 0, 0, 0, 0, 0, 0, 1).
   nrow = 8, ncol = 8, byrow = TRUE,
   dimnames = list(states, states))
```

```
transition <- function(</li>
   given pi, states, P normal, P speed) {
   P out <- array(0,
     dim = c(length(states), length(states)),
     dimnames = list(states, states))
   for (s in states) {
      action dist <- given pi[s.]
     P <- action dist["normal"]*P normal +
       action dist["speed"]*P speed
     P out[s,] <- P[s,]
   return(P out)
                         R(5,0)
• R s a <- matrix(</pre>
   c( -1, -1, -1, -1, 0.0, -1, -1, 0,
     -1.5, -1.5, -1.5, -1.5, -0.5, -1.5, -1.5, 0),
   nrow = length(states), ncol = 2, byrow = FALSE,
   dimnames = list(states, c("normal", "speed")))
reward fn <- function(given pi, R s a) {</pre>
   R pi <- rowSums(given pi*R s a)
   return(R pi)
                         RUST-
```

```
Vpolicy eval <- function(given pi) {
    R <- reward_fn(given pi, R s a = R s a)
    P <- transition(given pi, states = states, P normal = P normal, P speed = P speed)
    gamma <- 1.0
    epsilon <- 10^(-8)
    v old <- array(rep(0,8), dim=c(8,1))</pre>
    v new <- R + gamma*P%*%v old
    while (max(abs(v new-v old)) > epsilon) {
      v old <- v new
      v new <- R + gamma*P%*%v old
    return(v new)
  pi speed <- cbind(rep(0,length(states)), rep(1,length(states)))</pre>
  rownames(pi speed) <- states; colnames(pi speed) <- c("normal", "speed")</pre>
t(policy_eval(pi_speed))
                                       20
  ##
                   a
                            10
                                                 30
                                                                     50
  ## [1,] -5.805929 -5.208781 -4.139262 -3.475765 -2.35376 -1.735376 -1.673538
  pi 50 <- cbind(rep(0.5,length(states)), rep(0.5,length(states)))
  rownames(pi 50) <- states; colnames(pi 50) <- c("normal", "speed")</pre>
4t(policy_eval(pi_50))
  ##
                            10
                                       20
                                                 30
                                                          40
                                                                     50
                                                                               69 79
  ## [1,] -5.969238 -5.133592 -4.119955 -3.389228 -2.04147 -2.027768 -1.351388 0
```

Major components of approaching MDP

```
If you can't measure,
   you can't improve!
```

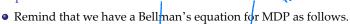
- 1. (policy evaluation) We need to be able to evaluate $(V^{\pi}(s))$ for a fixed π . This is called *policy evaluation*. This is also called as *prediction* in reinforcement learning.
- 2. (optimal value function) We want to be able to evaluate $V^{\pi^*}(s)$, where π^* is the optimal policy. The quantity, $V^{\pi^*}(s)$, is optimal policy's value function, or called 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
- Check your reasoning why the followings are possible.
 Optimal policy first: (optimal policy) + (policy evaluation) → (optimal value function)

 - Optimal value function first: (optimal value function) \rightarrow (optimal policy)
 - This note will develop the following approach.
 - (policy evaluation) + series of (policy improvement) \rightarrow (optimal policy)

II. Policy improvement

Development

Recap



$$\underline{V^\pi(s)} = R^\pi(s) + \gamma \sum_{\forall s'} \mathbf{P}^\pi_{ss'} V^\pi(s') \quad \text{(E1, p18)}$$

• It means that, given a π , its value is determined by immediate reward plus discounted sum of future rewards.

Motivation

• We shall first criticize the π_{peed} , whose policy evaluation is given as below.

t(policy_eval(pi_speed))

- From the state 60, current policy gives the estimate for the state-value function of -1.6735376, meaning that sticking to π_{speed} at state 60 has a value of -1.67.
- However, we know that switching to normal mode at state 60 is better, because switching to normal action at state 60 guarantees the arrival to the state 70 with additional energy spending of 1.0.

words remads.

Development

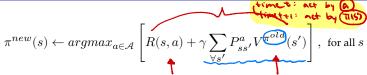
- ullet Under the current policy's (π_{speed}) value function, on the state 60,
 - Choosing normal mode gives

$$R + \gamma PV = -1.0 + 1.0 \cdot 0 = (-1.0)$$

$$R + \gamma \mathbf{P}V = -1.0 + 1.0 \cdot 0 = -1.0$$
• Choosing speed mode gives
$$R + \gamma \mathbf{P}V = -1.5 + (0.9 \cdot 0 + 0.1 \cdot -1.74) = -1.674$$

- $\bullet~$ Thus, π_{speed} should modify its action on the state 60. We've just improved the current policy π_{speed} for the state 60!
- This should be checked for all states as well as the state 60.
- This completes one step of **policy improvement**.
- Formally, policy improvement implies the following task of replacement:

$$\underline{\pi^{new}}(s) \leftarrow \underbrace{argmax_{a \in \mathcal{A}}}_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{\forall s'} P^a_{ss} \underbrace{V^{\underline{\pi^{old}}}(s')}_{ss} \right], \text{ for all } s$$



Meaning

- The term in the RHS, $R(s,a) + \gamma \sum_{\forall s'} P^a_{ss'} V^{\pi^{old}}(s')$ implies [the expected return of starting from states choosing an action a for this time step only, then following the policy π afterwards.].
- How is this quantity different from $V^{\pi}(s)$?

Defining q(s,a) action-value function

- The RHS makes an improvement using current policy (π) by varying only the action in this time step.
- Formally, q(s,a) is called **action-value function**, also famously known as *Q-function*.

$$q^{\pi}(s,a) := \underbrace{\mathbb{E}_{\underline{\pi}}[G_t|S_t = s, \underline{A_t = a}]}_{R(s,a) + \gamma \underbrace{\sum_{\mathbf{N}^s} P^a_{s'} V^{\pi^{old}}}_{\mathbf{N}^{old}}(s')}$$

$$\pi^{new}(s) \leftarrow argmax_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{\forall s'} P^a_{ss'} V^{\pi^{old}}(s') \right], \text{ for all } s$$
 Policy improvement

ullet Using this new notation of q(s,a), the policy improvement can be written as

$$\bigvee \pi_{new}(s) \leftarrow argmax_{a \in \mathcal{A}} q^{\sigma^{old}}(s, a), \text{ for all } s$$

- The improvement is called *greedy improvement* since it involves a myopic digression from the current policy, in a way that an action only on this time step is revised.
- It can be proved that *greedy improvement* is guaranteed to improve.

Implementation

$$\pi^{\textit{new}}(s) \leftarrow argmax_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{\forall s'} \mathbf{P}^{a}_{ss'} V^{\pi^{\textit{old}}}(s') \right], \text{ for all } s \in \mathcal{A}$$

```
V old <- policy eval(pi speed)
                                                   pi_new_vec <- apply(q_s_a, 1, which.max)</pre>
pi_old <- pi_speed
                                                   pi new <- array(0, dim = dim(pi old),
                      B+261
q_s_a <- R_s_a +
                                                                   dimnames = dimnames(pi old))
  cbind(gamma*P normal%*%V old,
                                                   for (i in 1:length(pi new vec)) {
        gamma*P speed%*%V old)
                                                     pi new[i, pi new vec[i]] <- 1
        g(s, a=nor) g(s. a=speed)
                                                   pi_new
         normal
      -6.208781 -5.805929
                                                   ##
                                                         normal speed
  10 -5.139262 -5.208781
                                                   ## 0
  20 -4.475765 -4.139262
                                                   ## 10
## 30 -3.353760 -3.475765
                                                   ## 20
                                COlumn Wile
## 40 -1.735376 -2.353760
                                                   ## 30
## 50 -2.673538 -1.735376
                                                   ## 40
     (-1.000000) -1.673538
                                                   ## 50
## 60
## 70
      0.000000 0.000000
                                                   ## 60
```

70

```
policy improve()
policy improve <- function(
 V old,
 pi old = pi old, R s a = R s a, gamma = gamma,
 P normal = P normal, P speed = P speed) {
 q s a <- R_s_a + cbind(gamma*P_normal%*%V_old,
                      gamma*P speed%*%V old)
 pi new vec <- apply(q s a, 1, which.max)
 %i new <- array(0, dim = dim(pi old),
               dimnames = dimnames(pi old))
 for (i in 1:length(pi_new_vec)) {
   pi new[i, pi new vec[i]] <- 1
 return(pi new)
```

• One step improvement from π^{speed}

```
pi_old <- pi_speed

V_old <- policy_eval(pi_old)

pi_new <- policy_improve(V_old,
 pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
 P_normal = P_normal, P_speed = P_speed)
```

```
(TI speed)
      normal speed
## A
## 10
## 20
## 30
## 40
## 50
## 60
##_70
pi_new
      normal speed
## 0
            0
## 10
   20
   30
                  0
## 40
## 50
## 60
## 70
                  a
            1
```

Summary

Policy improvement uses Bellman's equation of

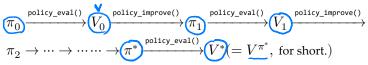
$$\pi^{new}(s) \leftarrow argmax_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{\forall s'} \mathbf{P}^a_{ss'} V^{\pi^{old}}(s') \right], \text{ for all } s$$

- For each state, policy improvement assesses whether the current policy leads to an optimal action that the current state-value function vouches.
- Policy improvement occurs in a greedy fashion. This greedy approach is guaranteed to improve.

III. Policy iteration

Development

- Given a policy <u>\pi</u>, <u>policy_eval()</u> evaluates its state-value function. Using the estimate of state-value function, <u>policy_improve()</u> improves the policy to the better one.
- If this process is iterated, then it is guaranteed to reach optimal policy. In other words, policy iteration is the process to reach the optimal policy described as follows.



- \bullet The iteration process terminates when π_i does not change any more, i.e. $\pi_i = \pi_{i+1}.$
- Note that policy evaluation is an approximate algorithm. For policy iteration purpose, policy evaluation cannot be, (and doesn't have to be as well), perfect.

Try do it over and over until no change - from π^{speed}

• Step 0

```
pi_old <- pi_speed
✓ pi old
  ##
         normal speed
              0
  ## A
                     1
  ## 10
                     1
  ## 20
  ## 30
              0
                     1
                     1
  ## 40
  ## 50
                     1
  ## 60
              0
                     1
  ## 70
              0
                     1
```

• Step 1

```
V old <- policy eval(pi old)
√pi_new <- policy_improve(V_old,
   pi old = pi old, R s a = R s a, gamma = gamma,
   P normal = P normal, P speed = P speed)
 pi_old <- pi_new
 pi_old
       normal speed
 ##
 ## 0
 ## 10
 ## 20
 ## 30
 ## 40
                  0
 ## 50
 ## 60
 ## 70
```

##

Step 2

```
V_old <- policy_eval(pi_old)
pi_new <- policy_improve(V_old,
    pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
    P_normal = P_normal, P_speed = P_speed)
pi_old <- pi_new
pi_old</pre>
```

##	0	0	1	
##	10	0	1	
##	20	0	1	
##	30	1	0	
##	40	1	0	
##	50	0	1	
##	60	1	0	
##	70	1	0	

normal speed

• Step 3

```
V_old <- policy_eval(pi_old)
pi_new <- policy_improve(V_old,
    pi_old = pi_old, R_s_a = R_s_a, gamma = gamma,
    P_normal = P_normal, P_speed = P_speed)
pi_old <- pi_new
pi_old</pre>
```

##		normal	speed
##	0	0	1
##	10	0	1
##	20	0	1
##	30	1	0
##	40	1	0
##	50	0	1
##	60	1	0
##	70	1	a

Policy iteration process (from π^{speed})

 Now we are ready to implement whole process as a single code block with a loop.

```
pi old <- pi speed
cnt <- 0
repeat{ # do-while in R
  print(paste0(cnt, "-th iteration"))
  print(t(pi old))
 V old <- policy eval(pi old)
  pi new <- policy_improve(V_old,
    pi old = pi old, R s a = R s a, gamma = gamma,
    P normal = P normal, P speed = P speed)
  if (all.equal(pi new, pi old)==TRUE) break
  pi old <- pi new
  cnt <- cnt + 1
print(policy_eval(pi new))
```

```
## [1] "0-th iteration"
          0 10 20 30 40 50 60 70
## normal 0
## speed
## [1] "1-th iteration"
         0 10 20 30 40 50 60 70
## normal 0
            1
## speed 1 0 1 0 0
## [1] "2-th iteration"
          0 10 20 30 40 50 60 70
## normal 0
## speed 1
           [,1]
## 0 -5.107744
## 10 -4.410774
## 20 -3.441077
## 30 -2,666667
## 40 -1.666667
## 50 -1.666667
## 60 -1.000000
## 70 0.000000
```

Policy iteration process (from π^{50})

• The process should work for other initial choice of π , despite of possibly different convergence rate.

```
pi old <- pi 50
cnt <- 0
repeat{ # do-while in R
  print(paste0(cnt, "-th iteration"))
  print(t(pi old))
 V old <- policy_eval(pi old)
  pi new <- policy improve(V old,
    pi old = pi old, R s a = R s a, gamma = gamma,
    P normal = P normal, P speed = P speed)
  if (all.equal(pi new, pi old)==TRUE) break
  pi old <- pi new
  cnt <- cnt + 1
print(policy_eval(pi new))
```

```
## [1] "0-th iteration".
            0 10 20 30
                          40
## normal 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
## speed 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
## [1] "1-th iteration"
          0 10 20 30 40 50 60 70
## normal 0
           1
## speed 1 0 1
## [1] "2-th iteration"
          0 10 20 30 40 50 60 70
## normal 0
                  0 0
## speed 1
           [,1]
##
      -5.107744
## 10 -4.410774
## 20 -3.441077
## 30 -2.666667
## 40 -1.666667
## 50 -1.666667
## 60 -1.00000
## 70 0.000000
```

Summary

- From a policy π , $q^{\pi}(s,a)$ implies an expected return of starting from the state s, choosing an action a for this time step only, then following the policy π afterwards.
- Policy improvement occurs by

$$\pi^{new}(s) \leftarrow argmax_{a \in \mathcal{A}} \ q^{\pi^{old}}(s, a)$$

, or, equivalently,

$$\pi^{new}(s) \leftarrow argmax_{a \in \mathcal{A}} \left[\underbrace{R(s,a) + \gamma \sum_{\forall s'} P^a_{ss'} V^{\pi^{old}}(s')}_{;} \right], \text{ for all } s$$

• Policy iteration is the iterative process from an arbitrary policy, policy evaluation and policy improvement take places until the policy converges. It is guaranteed to be converges to the optimal policy.

Exercise 1

Write python code to generate the same output in p.20.

"Success isn't permarnent, and failure isn't fatal. - Mike Ditka"