Lecture F2. MDP without Model 1

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

SEOULTECH 서울과학기술대학교 데이터사이언스학과

1. I. Development
2. II. Strategy
3. III. Preparation

I. Development

Recap on “Part E. MDP with model”

- Following is the list of sections of the lecture notes in `Part E`.
- Lecture note `E1` $\boxed{F1}$
  1. Introduction and Preview
  2. Setting
  3. Policy Evaluation 1 ✓
  4. Policy Evaluation 2 $V^\pi = R + \gamma P V^\pi$ ✓

  Pol. Eval 1 — mc ✓

  '' 2 ~ TD
- Lecture note `E2` $\boxed{F2}$
  1. Recap
  2. Policy improvement ✓ $\pi(s) = \underset{a}{\arg\max}\, R + \gamma P V$ Pol Improve
  3. Policy iteration ✓
- Lecture note `E3`
  1. Value improvement ✓ ✓ $V^* = \max(R + \gamma P V^*)$ ✓
  2. Value iteration
- In the previous lecture `F1`, we discussed two policy evaluation method (mc. TD) without knowledge of the model. Namely, MC policy evaluation and TD policy evaluation.
- Thus, today’s lecture must be on how to improve an policy without knowledge of the model, using the policy evaluation in the previous lecture `F1`.

Recap on policy improvement with knowledge of model

~~V(s).~~ q(s, a)

- In `F1`, we successfully evaluated $V^{\pi^{speed}}$ and $V^{\pi^{50}}$
- Let's first review our approach at `E2. Sec 2. Policy improvement.`
- In `E2, p8` we criticized π^{speed} as follows.

rewrite here

*From the state '~~60~~' '30', current policy gives the estimate for the state-value function of -1.67. (However), we know that switching to *normal mode* at state '~~60~~' is better alternative than current action of *speed mode*. Because it guarantess the arrival to the state '70' with additional energy spending of 1.0.*

$R(s) + \gamma P V(s')$ ✓

q(s, a)

- In other words, from state '~~60~~' 30,
  - *Choosing normal mode gives*

    ✓ $R + \gamma \mathbf{P} V = -1.0 + 1.0 \cdot 0 = -1.0$
  - *Choosing speed mode gives*

    ✓ $R + \gamma \mathbf{P} V = -1.5 + (0.9 \cdot 0 + 0.1 \cdot -1.74) = -1.674$
- Is this applicable to our current situation? NO.

✓ "0" "n" "-1" = "0" "n" "-1" = "10" .... = =

Issue 1 - can't evaluate $R + \gamma \mathbf{P} V$

- **Issue 1**: Even after evaluating V, unattainable is $R + \gamma \mathbf{P} V$ because we lack the knowledge of R and $\mathbf{P}$.
- To find an alternative, let's again review the policy improvement with knowledge of the model.
- In `E2, p20`, it states,

  Policy improvement occurs by

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

when unknown

, or, equivalently,

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

when MDP is known

- **Resolution to the Issue 1**: Use the first way of involving q-function, not the second way of involving value function. Let's go back to policy evaluation and evaluate $q^{\pi}(s,a)$, instead of $V^{\pi}(s)$.

Issue 2 - may not collect all combination of (s, a) for $q^{\pi}(s, a)$.

- Going back to π^{speed}, the agent always chooses speed mode at the state `60`. So there is no way we may collect sample of $q^{\pi^{speed}}(60, \text{“normal”})$ no matter how many episode we generates.
- In other words, from a deterministic policy π, we can't find an improvement toward the action that current π does not take.
- Choosing a random policy π^{50} could be an alternative? We can do one-step improvement from π^{50}, but it will become a determinstic policy again. Then, we are stucked again.
- Policy improvement without knowledge, thus cannot be done with 'greedy way'. It must allow some relaxation of not choosing the best out of experienced history. We call this as **exploration**.
- In other words, the improved policy must be a random policy. The improved policy needs to take every action a with some positive probability. By doing so, the improved policy will explore actions even if the action is suboptimal to the current policy evaluation.

- Then, how to improve policy with a guarantee of random policy?

- We introduce a ϵ-greedy action, which is to
    - choose the best action under the policy evaluation with probability $1 - \epsilon$
    - and to choose uniformly random action among all available actions with probability ϵ.

- Remind that **greedy action** with respect to the current values and then follow a policy π afterwards is an improvement over π.
- It can be proved that **ϵ-greedy action** with respect to the current values and then follow a policy π afterwards is an improvement over π.

- We say **exploitation** for choosing the best action under the policy evaluation.
- We say **exploration** for choosing the best action under the policy evaluation.
- In this light, ϵ-greedy action is to exploit with probability 1-ϵ and to explore with probability ϵ.

- We are to iterative policy, where each iteration involves an policy evaluation and policy improvement. Then, how should you choose the value of ϵ to reach the optimal policy?

1. We need ϵ to be large enough so that proper exploration take places among all possible actions.
2. On the other hand, remind that the optimal policy is guaranteed to be determinstic policy. We do not want produce the ultimate outcome that is randomizing action sets.

- Desired properties of ϵ is stated as below. In other words, π must be iteratively improved in a GLIE way.

GLIE $\sum_{n=1}^{\infty} \frac{1}{n} = \infty$ $\quad \sum_{n=1}^{\infty} \frac{1}{n^2} < \infty$ ✓ $\quad S_n = \sum a_n$ $\quad \boxed{\frac{a_n \to 0}{S_n \to \infty}}$

Definition 1

A policy π is called Greedy in the Limit of Infinite Exploration (GLIE) if

- (Exploration) All state-action pairs are visitied an infinite number of times.
- (Exploitation) The policy must be converged to the policy that is greedy with respect to the learned Q-function.

Remark 1

An ϵ-greedy policy where $\epsilon = \frac{1}{i}$ (i is the episode number) is GLIE. This is because

- $\sum_{i=1}^{\infty} \frac{1}{i} = \infty$ (exploration)
- $\frac{1}{i} \to 0$, as $i \to \infty$ (exploitation)

II. Strategy

Policy iteration process

$V^* = V^{\pi^*}$

- Policy iteration process is described as follows ($\underline{\texttt{E2,}}$ `p15`)

$V \leftarrow R + \gamma P V$

$$\pi_0 \xrightarrow{\texttt{policy\_eval()}} V_0 \xrightarrow{\texttt{policy\_improve()}} \pi_1 \xrightarrow{\texttt{policy\_eval()}} V_1 \xrightarrow{\texttt{policy\_improve()}}$$

$$\pi_2 \to \cdots \to \underset{\pi^*}{\cdots\cdots} \to \boxed{\pi^*} \xrightarrow{\texttt{policy\_eval()}} \boxed{V^*}(\underline{= V^{\pi^*},\ \text{for short.}})$$

- When model was known to the agent, policy evaluation was conducted until very accurate evaluation is done. This is because the process used fixed point theorem that is relatively fast.
- However, when model is not known (in note `F1`), policy evaluation using many sample paths take long time, because the sample paths need be generated prior to evaluation. Thus, general approach is use only small number of sample path (often, only one sample path) to do policy evaluation and then proceed to policy improvement.

$\pi' = \underset{a}{\operatorname{argmax}}\ q(s.a)$ (mc)

- This makes the general process of policy iteration to contain more higher number of iterations of policy evaluation and policy improvement.
- The MC policy iteration process is summarized as follows:
  0. Initialize $q(s,a)$ = 0
  1. Begin with a policy π
  2. Generate a sample path using the current π (`simul_path()`) → sample path
     π
  3. Evaluate the current policy π and update $q(s,a)$ (`pol_eval_MC()`) → q-s-a
  4. Improve the policy into a ϵ-greedy policy using $q(s,a)$ (`pol_imp()`) → π'
  5. Repeat 2-4 many times *(why not until policy converges?)*
- The TD policy iteration process is summarized as follows:
  0. Initialize $q(s,a)$
  1. Begin with a policy π
  2. Begin a new sample path from the state s
  3. Proceed a time step to generate subsequent a, r, s', a' (`simul_step()`) → sample-step
     π
  4. Evaluate the current policy π and update $q(s,a)$ (`pol_eval_TD()`) → q-s-a
  5. Improve the policy into a ϵ-greedy policy using $q(s,a)$ (`pol_imp()`) → π'
  6. Repeat 3-5 until the episode ends.
  7. Repeat 2-6 many times *(why not until policy converges?)*
- Rest of this note will create the source code `skier.R` that contains the functions above.

III. Preparation

`skiier.R` (1)

```
# Model
✓ states <- as.character(seq(0, 70, 10))
✓ P_normal <- matrix(c(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),
  nrow = 8, ncol = 8, byrow = TRUE,
  dimnames = list(states, states))
✓ P_speed <- matrix(c(.1, 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, 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))
```

```
✓ R_s_a <- matrix(
  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("n", "s")))
t(R_s_a)
```

```
##       0   10   20   30   40   50   60 70
## n -1.0 -1.0 -1.0 -1.0  0.0 -1.0 -1.0  0
## s -1.5 -1.5 -1.5 -1.5 -0.5 -1.5 -1.5  0
```

```
✓ q_s_a_init <- cbind(rep(0,length(states)),
                    rep(0,length(states)))
rownames(q_s_a_init) <- states;
colnames(q_s_a_init) <- c("n", "s")
t(q_s_a_init)
```

```
##   0 10 20 30 40 50 60 70
## n 0  0  0  0  0  0  0  0
## s 0  0  0  0  0  0  0  0
```

skiier.R (2)

```
# Policy
pi_speed <- cbind(rep(0,length(states)), rep(1,length(states)))
rownames(pi_speed) <- states;
colnames(pi_speed) <- c("n", "s")
t(pi_speed)
```

```
##   0 10 20 30 40 50 60 70
## n 0  0  0  0  0  0  0  0
## s 1  1  1  1  1  1  1  1
```

```
pi_50 <- cbind(rep(0.5,length(states)), rep(0.5,length(states)))
rownames(pi_50) <- states;
colnames(pi_50) <- c("n", "s")
t(pi_50)
```

```
##     0  10  20  30  40  50  60  70
## n 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
## s 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
```

skiier.R (3)

```r
# simul_path()
simul_path <- function(pi, P_normal, P_speed, R_s_a) {
  s_now <- "0"
  history_i <- s_now
  while (s_now != "70") {
    if (runif(1) < pi[s_now,"n"]) {
      a_now <- "n"
      P <- P_normal
    } else {
      a_now <- "s"
      P <- P_speed
    }
    r_now <- R_s_a[s_now, a_now]
    s_next <- states[which.min(cumsum(P[s_now,]) < runif(1))]
    history_i <- c(history_i, a_now, r_now, s_next)
    s_now <- s_next
  }
  return(history_i)
}
sample_path <- simul_path(pi = pi_speed, P_normal, P_speed, R_s_a)
sample_path
```

```
##  [1] "0"    "s"    "-1.5" "20"   "s"    "-1.5" "40"   "s"    "-0.5" "60"
## [11] "s"    "-1.5" "70"
```

skiier.R (4)

✓

```
# simul_step()
simul_step <- function(pi, s_now, P_normal, P_speed, R_s_a) {
  if (runif(1) < pi[s_now,"n"]) {
    a_now <- "n"
    P <- P_normal
  } else {
    a_now <- "s"
    P <- P_speed
  }
  r_now <- R_s_a[s_now, a_now]
  s_next <- states[which.min(cumsum(P[s_now,]) < runif(1))]
  if (runif(1) < pi[s_next,"n"]) {
    a_next <- "n"
  } else {
    a_next <- "s"
  }
  sarsa <- c(s_now, a_now, r_now, s_next, a_next)
  return(sarsa)
}
sample_step <- simul_step(pi = pi_speed, "0", P_normal, P_speed, R_s_a) # a.k.a. sarsa
sample_step
```

```
        s      a       r       s    a
## [1] "0"    "s"    "-1.5" "20"   "s"
```

skiier.R (5)

```r
## pol_eval_MC()
pol_eval_MC <- function(sample_path, q_s_a, alpha) {
  for (j in seq(1,length(sample_path)-1,3)) {
    s <- sample_path[j]
    a <- sample_path[j+1]
    G <- sample_path[seq(j+2, length(sample_path)-1, 3)] %>% as.numeric() %>% sum()
    q_s_a[s,a] <- q_s_a[s,a] + alpha*(G-q_s_a[s,a])
  }
  return(q_s_a)
}
q_s_a <- pol_eval_MC(sample_path, q_s_a = q_s_a_init, alpha = 0.1)
q_s_a
```

mc -tgt

```
##    n     s
## 0  0 -0.50
## 10 0  0.00
## 20 0 -0.35
## 30 0  0.00
## 40 0 -0.20
## 50 0  0.00
## 60 0 -0.15
## 70 0  0.00
```

skiier.R (6)

s a r s' a'

Sarsa

```r
## pol_eval_TD()
pol_eval_TD <- function(sample_step, q_s_a, alpha) {
  s <- sample_step[1]
  a <- sample_step[2]
  r <- sample_step[3] %>% as.numeric()
  s_next <- sample_step[4]
  a_next <- sample_step[5]
  q_s_a[s,a] <- q_s_a[s,a] + alpha*(r+q_s_a[s_next,a_next]-q_s_a[s,a])
  return(q_s_a)
}
q_s_a <- pol_eval_TD(sample_step, q_s_a_init, alpha=0.1)
q_s_a
```

TD target

```
##    n     s
## 0  0 -0.15
## 10 0  0.00
## 20 0  0.00
## 30 0  0.00
## 40 0  0.00
## 50 0  0.00
## 60 0  0.00
## 70 0  0.00
```

✓

skiier.R (7)

```
# pol_imp()
pol_imp <- function(pi, q_s_a, epsilon) { # epsilon = exploration_rate
  for (i in 1:nrow(pi)) {
    if (runif(1) > epsilon) { # exploitation
      pi[i, which.max(q_s_a[i,])] <- 1
      pi[i,-which.max(q_s_a[i,])] <- 0
    } else { # exploration
      pi[i,] <- 1/ncol(q_s_a)
    }
  }
  return(pi)
}
pi <- pol_imp(pi_speed, q_s_a, epsilon = 0)    ✓
t(pi)
```

```
##   0 10 20 30 40 50 60 70
## n 1  1  1  1  1  1  1  1
## s 0  0  0  0  0  0  0  0
```

Exercise 1

Prepare `skiier.py`

Exercise 2

Write the code for `mars.R` *for the Mars problem in Stanford's CS234.*

Exercise 3

Write the code for `mars.py` *for the Mars problem in Stanford's CS234.*

```
"It's not that I'm so smart, it's just that I stay with problems longer. - A. Einstein"
```