Lecture F1. MDP without Model - Policy Evaluation

Reinforcement Learning

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

SEOULTECH 서울과학기술대학교 데이터사이언스학과

I. Motivation

Recap on "MDP with model' ' part

- We had a full knowledge of its component $(\underline{S}, \underline{P}, \underline{R}, \underline{\gamma}, \underline{A})$.
- We had notation of
  - G_t
  - $R(s, a)$
  - $R^{\pi}(s)$ $= R(s, \pi(s))$
  - $\pi(s)$
  - $V^{\pi}(s)$
  - $q(s, a)$ state s에서 action a를 하고, 그 이후는 policy π를 따랐을때의 value fn.
  - Bellman's Eq:
- Our approach was
  - Policy evaluation
  - Policy improvement
  - Policy iteration
  - Value improvement
  - Value iteration

Setting

- In the previous part, we had the full knowledge of its component (S, P, R, γ, A).
- Often, this may not be possible in reality. In particular, probabilistic transition P, or $\mathbf{P}^{a}_{ss'}$ is unknown. In this case, we say that "We don't know the model.' '
- One possible approach is to start with estimating the transition mechanism. The other possible approach is to observe what the system tells us.
- In reinforcement learning, the model is called as **environment** that gives the reward to **agent**. The **agent** possesses a policy, and her action affects probabilistic transition. In other words, **the agent** interacts with **environment** by 1) changing probabilistic transition by making actions and 2) accepting rewards.
- Without fully knowing how **environment** works, our goal is still to find the best course of action. This process must be based on the accumulating the record, or **experience** by interaction with the environment.
- Thus, the algorithms to be discussed in this part is called **model-free algorithm**, including model-free policy evaluation, model-free policy iteration, and so on.

II. Simulator creation

About

- In reinforcement learning study, one may use existing simulator such as video game, robot, or machines. By interacting with the simulator (or environment), the agents earns record of her state, action, and immediate reward.
- Modern reinforcement learning books, especially short ones, focus on processing the records of her interaction in order to find the optimal strategy on "how to gain the most out of the environment' '.
- On the other hand, many real-world application of reinforcement learning must start from building a simulator that resemble as much aspects of the real environment as possible. Without capability of creating simulator, reinforcement learning research is restricted to realm where costly cheap simulator is pre-existing such as video game.
- This section creates the two MC simulators for the agent of π^{speed} and π^{50}.

History

- Our agent, the skier must gain serial experience of her **state, action, and reward**. The simulator must provide the experience. We call a full stochastic ~~episode~~ until termination as an **episode**, or *history*.

  path

- The skier with π^{normal} will experience, with 100% probability, history of the following ("n" for normal mode and "s" for speed mode, hereafter)

1st ep. $(\texttt{"0"}, n, -1, \texttt{"10"}, n, -1, \texttt{"20"}, n, -1, \texttt{"30"}, n, -1, \texttt{"40"}, n, 0, \texttt{"50"}, n, -1, \texttt{"60"}, n, 1, \texttt{"70"})$ termination

s. a. r. s a r

- Formally, history is the consecutive tuples of $\{s_t, a_t, r_t\}$ until the termination of her experience. The j-th history, among many repetitive simulation is defined as follows.

$$h_j = (s_{j,1}, a_{j,1}, r_{j,1}, s_{j,2}, a_{j,2}, r_{j,2}, \cdots, s_{j,L_j})$$

where L_j is the time length of j-th history.

Preparation

- From the previous lectures, we have created the following objects:
  - `states`
  - `P_normal`
  - `P_speed`
  - `R_s_a` R(s,a)
  - `pi_speed`
  - `pi_50`

env.

- These constitute the “environment” that our agent has no direct access to.

```r
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
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
```

```r
pi_speed <- cbind(rep(0,length(states)),
                  rep(1,length(states)))
rownames(pi_speed) <- states;
colnames(pi_speed) <- c("n", "s")
pi_50 <- cbind(rep(0.5,length(states)),
               rep(0.5,length(states)))
rownames(pi_50) <- states;
colnames(pi_50) <- c("n", "s")
```

```r
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
```

u ~ U(0,1)

```r
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
```

u ~ U(0,1)

Simulator for π^{speed} and π^{50}.

- We will create a 10,000 (i.e. `MC_N=10000`) episodes (i.e. full stochastic paths) for agent whose policy is π^{speed} and π^{50}.

Simulator for π^{speed}

```
pi <- pi_speed
set.seed(1234)
history <- list()
MC_N <- 10000
for (MC_i in 1:MC_N) {
  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
  }
  history[[MC_i]] <- history_i
}
history_speed <- history
```

$u \sim U(0,1)$

normal mode

s_now, a_now, r_now
s_next

U

```r
map(history_speed[1:20], function(x) paste0(x, collapse = ",")) %>% unlist() # map() from tidyverse
```

```
##  [1] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
##  [2] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
##  [3] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,50,s,-1.5,70"
##  [4] "0,s,-1.5,20,s,-1.5,10,s,-1.5,30,s,-1.5,50,s,-1.5,70"
##  [5] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
##  [6] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
##  [7] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
##  [8] "0,s,-1.5,0,s,-1.5,0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
##  [9] "0,s,-1.5,20,s,-1.5,40,s,-0.5,30,s,-1.5,50,s,-1.5,70"
## [10] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [11] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,50,s,-1.5,70"
## [12] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [13] "0,s,-1.5,0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [14] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [15] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [16] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [17] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [18] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [19] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
## [20] "0,s,-1.5,0,s,-1.5,20,s,-1.5,40,s,-0.5,60,s,-1.5,70"
```

-5.0

-5.0

v -6.5

$V^{\pi}(s)$

$V^{\pi_{speed}}(\text{state}=0)$

$= \frac{-5.0-5.0-6.5-\cdots}{10\cancel{00}0}$

$V^{\pi_{speed}}(\text{state}=40)$

$= \frac{-2-2-3.5\cdots}{9}$

Simulator for π^{50}

```
pi <- pi_50
set.seed(1234)
history <- list()
MC_N <- 10000
for (MC_i in 1:MC_N) {
  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
  }
  history[[MC_i]] <- history_i
}
history_50 <- history
```

```r
map(history_50[1:20], function(x) paste0(x, collapse = ",")) %>% unlist() # map() from tidyverse
```

```
##  [1] "0,n,-1,10,s,-1.5,30,s,-1.5,50,n,-1,60,s,-1.5,70"
##  [2] "0,s,-1.5,20,n,-1,30,n,-1,40,n,0,50,n,-1,60,n,-1,70"
##  [3] "0,n,-1,10,n,-1,20,s,-1.5,40,s,-0.5,30,n,-1,40,n,0,50,n,-1,60,n,-1,70"
##  [4] "0,s,-1.5,20,s,-1.5,40,n,0,50,n,-1,60,s,-1.5,70"
##  [5] "0,n,-1,10,n,-1,20,s,-1.5,40,n,0,50,n,-1,60,n,-1,70"
##  [6] "0,s,-1.5,0,n,-1,10,n,-1,20,n,-1,30,n,-1,40,n,0,50,n,-1,60,n,-1,70"
##  [7] "0,n,-1,10,n,-1,20,s,-1.5,40,n,0,50,n,-1,60,n,-1,70"
##  [8] "0,n,-1,10,n,-1,20,n,-1,30,n,-1,40,n,0,50,n,-1,60,n,-1,70"
##  [9] "0,n,-1,10,n,-1,20,n,-1,30,n,-1,40,s,-0.5,60,s,-1.5,70"
## [10] "0,n,-1,10,s,-1.5,30,n,-1,40,s,-0.5,60,s,-1.5,70"
## [11] "0,n,-1,10,n,-1,20,n,-1,30,s,-1.5,50,n,-1,60,s,-1.5,70"
## [12] "0,s,-1.5,20,s,-1.5,40,s,-0.5,60,n,-1,70"
## [13] "0,n,-1,10,s,-1.5,30,s,-1.5,50,n,-1,60,n,-1,70"
## [14] "0,n,-1,10,s,-1.5,30,n,-1,40,n,0,50,n,-1,60,n,-1,70"
## [15] "0,n,-1,10,s,-1.5,30,s,-1.5,50,s,-1.5,70"
## [16] "0,n,-1,10,n,-1,20,n,-1,30,n,-1,40,s,-0.5,30,s,-1.5,20,s,-1.5,40,n,0,50,s,-1.5,70"
## [17] "0,n,-1,10,s,-1.5,30,s,-1.5,50,s,-1.5,70"
## [18] "0,s,-1.5,20,n,-1,30,s,-1.5,50,n,-1,60,n,-1,70"
## [19] "0,s,-1.5,20,n,-1,30,s,-1.5,50,s,-1.5,70"
## [20] "0,n,-1,10,n,-1,20,n,-1,30,n,-1,40,s,-0.5,60,s,-1.5,70"
```

$$V^{\pi^{50}}(\text{state} = 20) = \frac{-7-3-4.5-4-4}{5} =$$

III. Policy Evaluation - Monte-Carlo

Motivation

- From the history objects `history_speed` and `history_50` above, how would you estimate $V^{\pi^{speed}}$ and $V^{\pi^{50}}$?
- Remind that

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s],$$

  where G_t is the discounted sum of future rewards.
- Let us look at the the first episode for $\pi^{speed}(s)$.
  - 0, s, -1.5, 20, s, -1.5, 40, s, -0.5, 60, s, -1.5, 70
  - We can tell the followings from the full stochastic path
    - From the state “0”, the return(G_t) is -1.5-1.5-0.5-1.5=-5.0
    - From the state “20”, the return(G_t) is -1.5-0.5-1.5=-3.5
    - ...
- The strategy is to investigate each history and take average of the return for each state.
- We will first exhibit the implementation, then formally summarizes the algorithm.

MC policy evaluation for π^{speed} (vectorized way)

```
pol_eval <- array(
  0, dim = c(length(states),2),
  dimnames = list(states, c("count", "sum")))
t(pol_eval)
```

j j+2

s a r s a r s a r s

```
##       0 10 20 30 40 50 60 70
## count 0  0  0  0  0  0  0  0
## sum   0  0  0  0  0  0  0  0
```

- We are to collect count and sum for each state, by going over all history of 10,000 episodes.

```
for (MC_i in 1:length(history_speed)) {
  history_i <- str_split(history_speed[[MC_i]], ",") %>% unlist()
  for (j in seq(1,length(history_i),3)) {
    pol_eval[history_i[j],"count"] <- pol_eval[history_i[j],"count"] + 1
    if (j < length(history_i)) {
      pol_eval[history_i[j],"sum"] <- pol_eval[history_i[j],"sum"] +
        history_i[seq(j+2,length(history_i)-1,3)] %>% as.numeric() %>% sum()
    } else { # terminal state
      pol_eval[history_i[j],"sum"] <- pol_eval[history_i[j],"sum"] + 0
    }
  }
}
```

not terminal state

terminal state

```
t(pol_eval)
```

```
##              0    10     20    30     40    50     60    70
## count    11201  1042  10272  1846   9485  2530   8579 10000
## sum     -64980 -5462 -42448 -6408 -22211 -4428 -14354     0
```

```
pol_eval[,"sum"]/pol_eval[,"count"]
```

```
##     0    10    20    30    40    50    60    70
## -5.80 -5.24 -4.13 -3.47 -2.34 -1.75 -1.67  0.00
```

← mc pol eval on π speed

−5.81 $\boxed{-5.21}$ −4.14 −3.48 −2.35 −1.74 −1.67 0

MC policy evaluation for π^{speed} (running estimate way)

- As the MC repetition proceeds, we will update the estimate by the following rule (A6, p13)

$$\hat{\theta}_{new} \leftarrow \hat{\theta}_{old} + \alpha(A_n - \hat{\theta}_{new}),$$

where $\theta = \mathbb{E}A$, $\alpha = 1/n$, and n is cumulative count.

- Or, the above expression can be written as:

$$\texttt{new\_est} \leftarrow \texttt{old\_est} + \alpha(\texttt{MC\_tgt-old\_est})$$

```r
pol_eval <- array(
  0, dim = c(length(states),2),
  dimnames = list(states, c("count", "est")))
t(pol_eval)
```

```
##       0 10 20 30 40 50 60 70
## count 0  0  0  0  0  0  0  0
## est   0  0  0  0  0  0  0  0
```

```r
for (MC_i in 1:length(history_speed)) {
  history_i <- str_split(history_speed[[MC_i]], ",") %>% unlist()
  for (j in seq(1,length(history_i),3)) {
    # update count
    pol_eval[history_i[j],"count"] <- pol_eval[history_i[j],"count"] + 1
    current_cnt <- pol_eval[history_i[j],"count"]
    # return is the new information, MC_tgt.
    if (j < length(history_i)) {
      MC_tgt <- history_i[seq(j+2,length(history_i)-1,3)] %>% as.numeric() %>% sum()
    } else { # terminal state
      MC_tgt <- 0
    }
    # update the last estimate using MC_tgt
    alpha <- 1/current_cnt
    pol_eval[history_i[j],"est"] <- pol_eval[history_i[j],"est"] +
      alpha*(MC_tgt- pol_eval[history_i[j],"est"])
  }
}
```

카운트

```r
t(pol_eval)
```

```
##              0       10       20      30      40      50      60    70
## count  11201.0  1042.00 10272.00 1846.00 9485.00 2530.00 8579.00 10000
## est       -5.8    -5.24    -4.13   -3.47   -2.34   -1.75   -1.67     0
```

Discussion

- Compare how similar the results in p~~22~~ 20 and p~~24~~ 22 are.

MC policy evaluation for π^{50} (vectorized way)

```r
pol_eval <- array(
  0, dim = c(length(states),2),
  dimnames = list(states, c("count", "sum")))
t(pol_eval)
```

```
##       0 10 20 30 40 50 60 70
## count 0  0  0  0  0  0  0  0
## sum   0  0  0  0  0  0  0  0
```

```r
for (MC_i in 1:length(history_50)) {
  history_i <- str_split(history_50[[MC_i]], ",") %>% unlist()
  for (j in seq(1,length(history_i),3)) {
    pol_eval[history_i[j],"count"] <- pol_eval[history_i[j],"count"] + 1
    if (j < length(history_i)) {
      pol_eval[history_i[j],"sum"] <- pol_eval[history_i[j],"sum"] +
        history_i[seq(j+2,length(history_i)-1,3)] %>% as.numeric() %>% sum()
    } else { # terminal state
      pol_eval[history_i[j],"sum"] <- pol_eval[history_i[j],"sum"] + 0
    }
  }
}
```

```
t(pol_eval)
```

```
##            0     10     20     30     40     50     60    70
## count  10854   5782   8137   7100   7522   7247   7137 10000
## sum   -64875 -29706 -33581 -24132 -15342 -14690 -9649     0
```

```
pol_eval[,"sum"]/pol_eval[,"count"]
```

```
##     0    10    20    30    40    50    60    70
## -5.98 -5.14 -4.13 -3.40 -2.04 -2.03 -1.35  0.00
```

MC policy evaluation for π^{50} (running estimate way)

- As the MC repetition proceeds, we will update the estimate by the following rule (A6, p13)

$$\hat{\theta}_{new} \leftarrow \hat{\theta}_{old} + \alpha(A_n - \hat{\theta}_{new}),$$

  where $\theta = \mathbb{E}A$, $\alpha = 1/n$, and n is cumulative count.

- Or, the above expression can be written as:

$$\texttt{new\_est} \leftarrow \texttt{old\_est} + \alpha(\texttt{MC\_tgt-old\_est})$$

```
pol_eval <- array(
  0, dim = c(length(states),2),
  dimnames = list(states, c("count", "est")))
t(pol_eval)
```

$q(s.a) \leftarrow q(s.a) + \alpha(MC\,tgt - q(s,a))$

```
##       0 10 20 30 40 50 60 70
## count 0  0  0  0  0  0  0  0
## est   0  0  0  0  0  0  0  0
```

```r
for (MC_i in 1:length(history_50)) {
  history_i <- str_split(history_50[[MC_i]], ",") %>% unlist()
  for (j in seq(1,length(history_i),3)) {
    # increment count
    pol_eval[history_i[j],"count"] <- pol_eval[history_i[j],"count"] + 1
    current_cnt <- pol_eval[history_i[j],"count"]
    # return is the new information, MC_tgt.
    if (j < length(history_i)) {
      MC_tgt <- history_i[seq(j+2,length(history_i)-1,3)] %>% as.numeric() %>% sum()
    } else { # terminal state
      MC_tgt <- 0
    }
    # update the last estimate with new information, MC_tgt.
    alpha <- 1/current_cnt
    pol_eval[history_i[j],"est"] <- pol_eval[history_i[j],"est"] +
      alpha*(MC_tgt-pol_eval[history_i[j],"est"])
  }
}
```

```r
t(pol_eval)
```

$V^\pi(s)$

```
##              0       10      20      30      40      50      60    70
## count 10854.00 5782.00 8137.00 7100.0 7522.00 7247.00 7137.00 10000
## est      -5.98   -5.14   -4.13   -3.4   -2.04   -2.03   -1.35     0
```

-5.97 -5.13 -4.12 -3.39

Summary

- Monte-Carlo policy evaluation. (running estimate)

```
For all state $s$, count(s) <- 0, old_est(s) <- 0
For each episode history_i
  For t=1,2,...,L_j
    count(s) <- count(s) + 1 when $s$ is encountered.
    MC_tgt <- collect all return after the state $s$.
    alpha <- 1/count(s)
    new_est(s) <- old_est(s) + alpha*(MC_tgt-old_est(s)) # the key part
    old_est <- new_est
  End
End
Return(new_est)
```

- This method is based on

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

and the algorithm iteratively performs

$$\underbrace{V_\pi(s)}_{new\ estimate} \leftarrow \underbrace{V_\pi(s)}_{old\ estimate} + \alpha \left(\underbrace{G_t}_{MC\_tgt\ (new\ info)} - \underbrace{V_\pi(s)}_{old\ estimate} \right),$$

where $\alpha = 1/count(s)$ in the plain setting.

IV. Policy Evaluation Temporal Difference

Recap

- The previous MC policy evaluation was motivated by

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

  and expectation was evaluated by MC repetitive episodes.

- In terms of algorithm, it used

$$\underbrace{V_\pi(s)}_{new\ estimate} \leftarrow \underbrace{V_\pi(s)}_{old\ estimate} + \alpha \left(\underbrace{G_t}_{MC\_tgt\ (new\ info)} - \underbrace{V_\pi(s)}_{old\ estimate} \right),$$

  where $\alpha = 1/count(s)$ in the plain setting.

- (G_t is called a MC target, and $G_t - V_\pi(s)$ is called a MC error.)

Drawback of MC

1. One caveat is we need a quantity of G_t, which can be obtained only when an episode terminates. Not all stochastic innovation terminates. You can think of 24-hour service or investment funds without matery.

2. Or, the period of updates is too long if an episode takes a long time. Are there ways that utilize newly coming information quickly?

Development

Spatio temporal

- Temporal difference method is motivated by

$$V_\pi(s) = \mathbb{E}_\pi[r_t + \gamma V_\pi(s') | S_t = s] \text{ (TD)}$$

$\sum_{i=1}^{\infty} \gamma^i r_{t+i}$

instead of

$$\boxed{V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]} \text{ (MC)}$$

$s \longrightarrow s'$

- In other words, temporal difference method does

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha(r_t + \gamma V_\pi(s') - V_\pi(s)) \text{ (TD)}$$

$r_t + \gamma r_{t+1} + \gamma^2 V_\pi(s'')$

instead of

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha(G_t - V_\pi(s)) \text{ (MC)}$$

- Naturally, $r_t + \gamma V_\pi(s')$ is called a TD target, and $r_t + \gamma V_\pi(s') - V_\pi(s)$ is called a TD error.
- TD updating occurs in every time step, instead of every episode (MC's updating frequency).

Pseudo code development

- Following is MC policy evaluation with running estimate (`F1,p25`)

$V^{\pi}(s)$

```
For all state $s$, count(s) <- 0, old_est(s) <- 0
For each episode history_i
  For t=1,2,...,L_j
    count(s) <- count(s) + 1 when $s$ is encountered.
    MC_tgt <- collect all return after the state $s$.
    alpha <- 1/count(s)
    new_est(s) <- old_est(s) + alpha*(MC_tgt-old_est(s))
    old_est <- new_est
  End
End
Return(new_est)
```

G_t

→ TD-tgt ← $r_t + V^{\pi}(s')$

r_t + old-est(s')

→ TD-tgt

- Q. What needs to be changed for TD estimate?
- A. `MC_tgt` needs to become `TD_tgt`.

The previous code - MC policy eval. (running) for π^{speed}.

```
for (MC_i in 1:length(history_speed)) {
  history_i <- str_split(history_speed[[MC_i]], ",") %>% unlist()
  for (j in seq(1,length(history_i),3)) {
    # update count
    pol_eval[history_i[j],"count"] <- pol_eval[history_i[j],"count"] + 1
    current_cnt <- pol_eval[history_i[j],"count"]
    # return is the new information, MC_tgt.
    if (j < length(history_i)) {
      MC_tgt <- history_i[seq(j+2,length(history_i)-1,3)] %>% as.numeric() %>% sum()
    } else { # terminal state
      MC_tgt <- 0
    }
    # update the last estimate with new information, MC_tgt.
    alpha <- 1/current_cnt
    pol_eval[history_i[j],"est"] <- pol_eval[history_i[j],"est"] +
      alpha*(MC_tgt-pol_eval[history_i[j],"est"])
  }
}
```

- Note that only the following part needs to be modified.

```
# return is the new information, MC_tgt.
if (j < length(history_i)) {
  MC_tgt <- history_i[seq(j+2,length(history_i)-1,3)] %>%
    as.numeric() %>% sum()
} else { # terminal state
  MC_tgt <- 0
}
```

$r_t + r_{t+1} + r_{t+2} + \cdots$

s, a, r, s', a', r'

1. Let's replace `MC_tgt` with `TD_tgt`,
   - `MC_tgt`: G_t
   - `TD_tgt`: $r_t + \gamma V_\pi(s')$
2. Since `history_i[j]` is the current state,
   - `history_i[j+2]` is its reward.
   - `history_i[j+3]` is the next state.

- The code needs to revised into

r_t $\quad$ $V^\pi(s')$

```
if (j < length(history_i)) {
  TD_tgt <- history_i[j+2] %>% as.numeric() + pol_eval[history_i[j+3],"est"]
} else { # terminal state
  TD_tgt <- 0
}
```

s'

TD policy evaluation for π^{speed}

```
pol_eval <- array(
  0, dim = c(length(states),2),
  dimnames = list(states, c("count", "est")))
t(pol_eval)
```

```
##       0 10 20 30 40 50 60 70
## count 0  0  0  0  0  0  0  0
## est   0  0  0  0  0  0  0  0
```

```r
for (episode_i in 1:length(history_speed)) {
  history_i <- str_split(history_speed[[episode_i]], ",") %>% unlist()
  for (j in seq(1,length(history_i),3)) {
    # update count
    pol_eval[history_i[j],"count"] <- pol_eval[history_i[j],"count"] + 1
    current_cnt <- pol_eval[history_i[j],"count"]
    # build TD target
    if (j < length(history_i)) {
      TD_tgt <- history_i[j+2] %>% as.numeric() +
        pol_eval[history_i[j+3],"est"]
    } else { # terminal state
      TD_tgt <- 0
    }
    # TD-updating
    alpha <- 1/current_cnt
    pol_eval[history_i[j],"est"] <- pol_eval[history_i[j],"est"] +
      alpha*(TD_tgt-pol_eval[history_i[j],"est"])
  }
}
```

```r
t(pol_eval)
```

```
##              0     10       20      30      40      50      60    70
## count 11201.00 1042.0 10272.00 1846.00 9485.00 2530.00 8579.00 10000
## est      -5.72   -5.2    -4.11   -3.46   -2.34   -1.73   -1.67     0
```

-5.8 -5.2 -4.14 -3.48 -2.35 -1.74

✓

TD policy evaluation for π^{50}

```
pol_eval <- array(
  0, dim = c(length(states),2),
  dimnames = list(states, c("count", "est")))
t(pol_eval)
```

```
##       0 10 20 30 40 50 60 70
## count 0  0  0  0  0  0  0  0
## est   0  0  0  0  0  0  0  0
```

```r
for (episode_i in 1:length(history_50)) {
  history_i <- str_split(history_50[[episode_i]], ",") %>% unlist()
  for (j in seq(1,length(history_i),3)) {
    # update count
    pol_eval[history_i[j],"count"] <- pol_eval[history_i[j],"count"] + 1
    current_cnt <- pol_eval[history_i[j],"count"]
    # build TD target
    if (j < length(history_i)) {
      TD_tgt <- history_i[j+2] %>% as.numeric() +
        pol_eval[history_i[j+3],"est"]
    } else { # terminal state
      TD_tgt <- 0
    }
    # TD-updating
    alpha <- 1/current_cnt
    pol_eval[history_i[j],"est"] <- pol_eval[history_i[j],"est"] +
      alpha*(TD_tgt-pol_eval[history_i[j],"est"])
  }
}
```

```r
t(pol_eval)
```

```
##              0      10      20     30      40      50      60    70
## count 10854.00 5782.00 8137.00 7100.0 7522.00 7247.00 7137.00 10000
## est      -5.89   -5.09   -4.11   -3.4   -2.04   -2.03   -1.35     0
```

V. Discussions

Discussion 1 - accuracy of policy evaluation

- Accuracy differs between knowing vs not knowing the model.
- **With the knowledge** of the model, π^{speed} was evaluated at `E1` as:

		0	10	20	30	40	50	60	70
`E1, Sec III`	Analytic	-5.81	-5.21	-4.14	-3.48	-2.35	-1.74	-1.67	0
`E1, Sec IV`	Fixed point	-5.81	-5.21	-4.14	-3.48	-2.35	-1.74	-1.67	0

- **Without the knowledge** of the model, π^{speed} is now evaluated as:

		0	10	20	30	40	50	60	70
occcurences		11201	1042	10272	1846	9485	2530	8579	10000
`Impl. 1 & 2`	MC	-5.80	-5.24	-4.13	-3.47	-2.34	-1.75	-1.67	0
`Impl. 4`	TD	-5.72	-5.20	-4.11	-3.46	-2.34	-1.73	-1.67	0

- Would it cause a problem?

Discussion 2 - MC, TD, and their variation

- The MC target is G_t.
- The TD target is $r_t + \gamma V_\pi(s')$.

- TD target seems to expand one time step. Will expanding two time steps be possible? In other words, will setting target of $r_t + \gamma r_{t+1} + \gamma^2 V_\pi(s'')$ work as well?
- The answer is yes, and it works for three time steps expansion such as $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V_\pi(s''')$ as well.

- If generalized, variations of targets can be listed as:
  - TD-0: $r_t + \gamma V_\pi(s')$ (This is original TD.) (방금한것)
  - TD-1: $r_t + \gamma r_{t+1} + \gamma^2 V_\pi(s'')$ ✓
  - TD-2: $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V_\pi(s''')$ ✓
  - TD-n: $r_t + \gamma r_{t+1} + \cdots + \gamma^{n+1} V_\pi(s^{'''(n+1)})$
  - TD-∞: $r_t + \gamma r_{t+1} + \cdots$ (This is MC.)

- The paper for the famous algorithm "A3C" uses 5-step TD on Atari agent.

Discussion 3 - Properties of w/ Model, MC, and TD.

episode가 terminate하지 않거나 발생안하나?

	w/ Model	MC	TD
Converges to true value?	Yes	Yes	Yes
Model free?	No	Yes	Yes
Non-episodic domain?	Yes	No	Yes
Unbiased estimate~~d~~ or?	N/A	Yes	No
Variance?	N/A	High	Low

- Remark
  - TD is biased.
  - MC has higher variance than TD.

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