Lecture F1. MDP without Model 1

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 (S, P, R, γ, A).
- We had notation of
  - G_t
  - $R(s, a)$
  - $R^\pi(s)$
  - $\pi(s)$
  - $V^\pi(s)$
  - $q(s, a)$
  - 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 probabilitic transition. In other words, **the agent** interacts with **environment** by 1) changing probabiliic transition by making actions and 2) accepting rewards.
- Without fully knowning 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, and so on.

II. Simulator creation

About

- In reinforcement learning study, one may use existing simulator such as video game, robot, or machinary. 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 most from 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-existant such as video game.
- The simulator may be physical device such as robot, or a computerized simulator that resembles the reality.
- This section creates two simulator for the skier's problem.

History

- Our agent, the skier must gain experience of series 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*.
- For example, the skier with π^{normal} will experience, with probability 1, one history must be ('n' for normal mode and 's' for speed mode, hereafter)

$$("0", n, -1, "10", n, -1, "20", n, -1, 30, n, -1, "40", n, 0, "50", n, -1, "60", n, 1, "70")$$

- Formally, history is the consecutive tuple of s_t, a_t, r_t until the termination of her experience. 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

```
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
```

```
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")
```

```
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
```

```
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
```

Simulator - 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
```

```
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"
```

Simulator - 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"
```

III. Monte-Carlo Policy evaluation

Motivation

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

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

- Let us look at the the first history $\pi^{speed}(s)$
  - 0, s, -1.5, 20, s, -1.5, 40, s, -0.5, 60, s, -1.5, 70
  - We can tell the followings
    - From the 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.

Implementation 1 - π^{speed} (vectorized)

```
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
```

```
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
    }
  }
}
```

```
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
```

Implementation 2 - π^{speed} (running estimate)

- (running estimate) 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.

```
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
```

```
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 info
    if (j < length(history_i)) {
      new_info <- history_i[seq(j+2,length(history_i)-1,3)] %>% as.numeric() %>% sum()
    } else { # terminal state
      new_info <- 0
    }
    # update the last estimate with new info
    alpha <- 1/current_cnt
    pol_eval[history_i[j],"est"] <- pol_eval[history_i[j],"est"] +
      alpha*(new_info- pol_eval[history_i[j],"est"])
  }
}
```

```
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
```

Implementation 3 - π^{50} (vectorized)

```
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
```

```
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
```

Implementation 4 - π^{50} (running estimate)

- (running estimate) 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.

```
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
```

```
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 info
    if (j < length(history_i)) {
      new_info <- history_i[seq(j+2,length(history_i)-1,3)] %>% as.numeric() %>% sum()
    } else { # terminal state
      new_info <- 0
    }
    # update the last estimate with new info
    alpha <- 1/current_cnt
    pol_eval[history_i[j],"est"] <- pol_eval[history_i[j],"est"] +
      alpha*(new_info- pol_eval[history_i[j],"est"])
  }
}
```

```
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      -5.98   -5.14   -4.13   -3.4   -2.04   -2.03   -1.35     0
```

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.
    new_info <- collect all return after the state $s$.
    alpha <- 1/count(s)
    new_est(s) <- old_est(s) + alpha*(new_info-old_est(s))
    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

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

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

IV. Temporal Difference Policy Evaluation

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

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

  where $\alpha = 1/\texttt{MC\_i}$ 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 when an episode terminates. Not all stochastic innovation terminates. You can think of 24-hour service or investment funds without materity.

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

Development

- Temporal difference method is motivated by

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

instead of

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

- In other words, temporal difference method does

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

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 MC's updating frequency of every episode.

Pseudo code development

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

```
 1: For all state $s$, count(s) <- 0, old_est(s) <- 0
 2: For each episode history_i
 3:   For t=1,2,...,L_j
 4:     count(s) <- count(s) + 1 when $s$ is encountered.
 5:     new_info <- collect all return after the state $s$.
 6:     alpha <- 1/count(s)
 7:     new_est(s) <- old_est(s) + alpha*(new_info-old_est(s))
 8:     old_est <- new_est
 9:   End
10: End
11: Return(new_est)
```

- Q. What needs to be changed for TD estimate?
- A. `new_info` was the MC target, it needs to be the TD target now.

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 info
    if (j < length(history_i)) {
      new_info <- history_i[seq(j+2,length(history_i)-1,3)] %>% as.numeric() %>% sum()
    } else { # terminal state
      new_info <- 0
    }
    # update the last estimate with new info
    alpha <- 1/current_cnt
    pol_eval[history_i[j],"est"] <- pol_eval[history_i[j],"est"] +
      alpha*(new_info-pol_eval[history_i[j],"est"])
  }
}
```

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

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

1. Let's replace `new_info` with `TD_tgt`.
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

```
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
}
```

Implementation 5 - 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
```

```
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"])
  }
}
```

```
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
```

Implementation 6 - 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
```

```
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"])
  }
}
```

```
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. I & II`	MC	-5.80	-5.24	-4.13	-3.47	-2.34	-1.75	-1.67	0
`Impl. IV`	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.)
  - 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:
  - 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 DP methd, MC, and TD.

	DP method	MC	TD
Converges to true value?	Yes	Yes	Yes
Model free?	No	Yes	Yes
Non-episodic domain?	Yes	No	Yes
Unbiased estimated?	N/A	Yes	No
Variance?	N/A	High	Low

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

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