


## #Session 3. Monte Carlo Policy Evaluation & Q-Learning

김진호


경희대학교 빅데이터연구센터  
경희대학교 소셜네트워크과학과

# Maze Problem

	(1,2)	(1,3)		<b>Trap</b>
(2,1)		(2,3)	(2,4)	(2,5)
(3,1)				(3,5)
(4,1)	(4,2)	(4,3)		
		(5,3)	(5,4)	<b>Goal</b>





- Trap을 피해 Goal에 도달하는 것
- 어떻게, 목표를 달성하는 Agent를 설계할 수 있을까?
- $S_t$  : Position of Agent at time  $t$ , ex. (1,1)
- $A_t$  : Possible Move on the  $S_t$ , ex. (Left, Down)
- $R_t$  : Goal = 100, Trap = -100, otherwise 0
- Policy?
  - $S_t$  에서 최적의 Action  $A_t^*$ 를 찾는 것

# Maze Problem

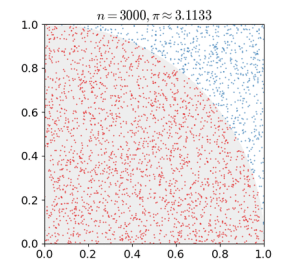
	(1,2)	(1,3)		<b>Trap</b>
(2,1)		(2,3)	(2,4)	(2,5)
(3,1)				(3,5)
(4,1)	(4,2)	(4,3)		
		(5,3)	(5,4)	<b>Goal</b>

- 지금까지 본 내용으로는,
  - 매 순간 최선을 다한 행동에 대해 평가를 하고,
  - 이를 학습해서 좋은 행동은 더 많이,
  - 이를 학습해서 나쁜 행동은 더 적게
- 매 순간 Action에 대한 Reward의 합을 통해,
  - 최종 Reward의 합이 최대가 되게!
- Reward는 Goal, Trap에 대해서만 발생한다!
  - 매 순간의 Reward는 어떻게 추산할까..

# Maze Problem

	(1,2)	(1,3)		<b>Trap</b>
(2,1)		(2,3)		(2,5)
(3,1)				(3,5)
	(4,2)	(4,3)		
		(5,3)		<b>Goal</b>

- Reward를 추산하기 힘드니! 겪어보자!
  - 일단 미로를 돌아다니게 해서
  - Goal에 도달하면 어쨌든, 좋은 행동
  - Trap에 도달하면 어쨌든, 나쁜 행동
- 충분히 많이 Sampling을 하면, 어느정도 되지 않을까?
  - Monte Carlo Simulation
  - Monte Carlo Policy Evaluation





MCPE

# Monte Carlo Policy Evaluation

Sampling을 통해 다양한 경로의 움직임을 학습하고,  
각 State에 대한 Policy의 가치를 학습하여, 최적의 Policy를 도출하자

# MCPE, Reward

	(1,2)	(1,3)		<b>Trap</b>
(2,1)		(2,3)	(2,4)	(2,5)
(3,1)				(3,5)
(4,1)	(4,2)	(4,3)		
		(5,3)		<b>Goal</b>

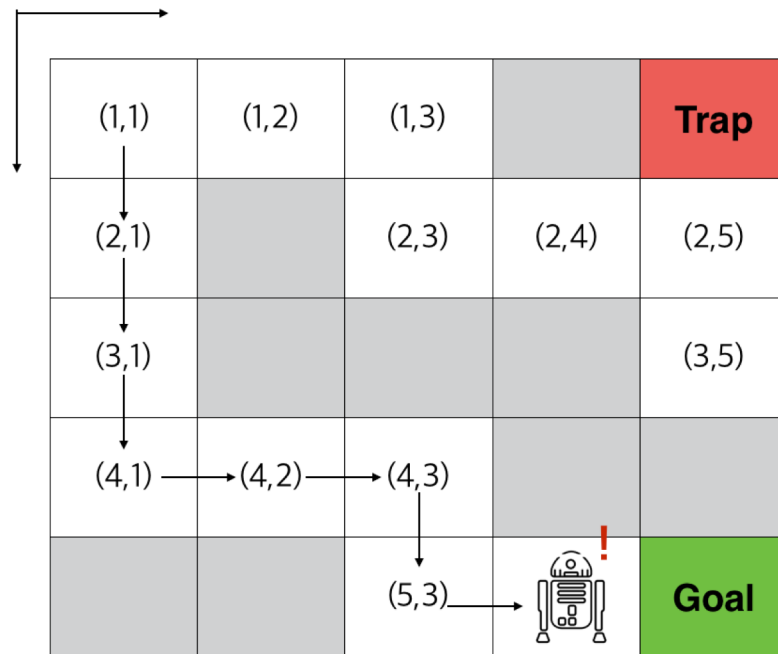
- Reward with Discount Factor  $\gamma$

- 100 steps만에 도착한 Agent
- 8 steps만에 도착한 Agent
- 차이를 주고 싶은데 ..

$$R = 100 - x$$

- $x$ 는 Agent가 goal에 도착했을 때, Step의 수
  - 100 steps만에 도착하였으면,  $R = 0$
  - 8 steps 만에 도착하였으면,  $R = 92$
- Trap에 도착한 경우,  $R \leftarrow -R = -100 + x$

# MCPE, Discount Factor



- Reward with Discount Factor  $\gamma$ 
  - 그림과 같이 도달한 경우,  $R = 92 (= 100 - 8)$
  - (1,1), Down의 가치와, (5,4), Left의 가치는 같을까?
- $V((5,4), L) = \gamma R = 0.7 \times 92 = 64.4$
- $V((5,3), L) = \gamma^2 R = 0.7^2 \times 92 = 45.08$
- $V((4,3), D) = \gamma^3 R = 0.7^3 \times 92 \approx 31.56$
- ...
- $V((1,1), D) = \gamma^8 R = 0.7^8 \times 92 \approx 5.30$



1

Worth Now



$\gamma$

Worth Next Step



$\gamma^2$

Worth In Two Steps

# MCPE, Value Table & Policy Table

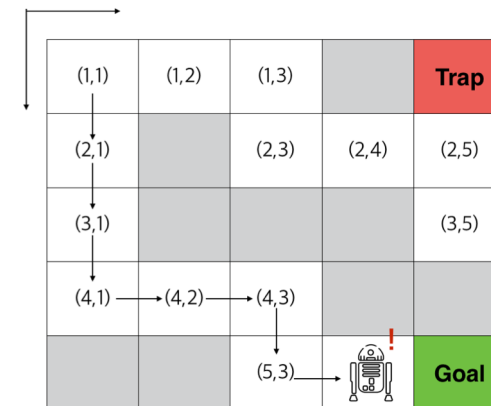
State	U	D	L	R
(5,4)	-	-	1	1
(5,3)	1	-	1	-
...				
(1,1)	-	1	1	-

$V((5,4),L) = 64.4$   
 $V((5,3),L) = 45.08$   
 $V((4,3),D) \approx 31.56$   
 ...  
 $V((1,1),D) \approx 5.30$

State	U	D	L	R
(5,4)	-	-	64.4	1
(5,3)	1	-	45.08	-
...				
(1,1)	-	5.30	1	-



State	U	D	L	R
(5,4)	-	-	0.98	0.02
(5,3)	0.03	-	0.97	-
...				
(1,1)	-	0.84	0.16	-



## MCPE, Pseudo Code

MC\_EVALUATION( $MDP, \pi, \gamma$ )

Inputs      policy  $\pi$   
             discount factor  $\gamma$   
Output      value function  $V^\pi$   
Initialize    $V = 0$

repeat

1. initialize  $s, \tau, \rho$
2. while  $s \notin T$  do
  - (a) let  $\tau = \tau \cup \{s\}$
  - (b) take action  $a = \pi(s)$
  - (c) observe reward  $r$  and next state  $s'$
  - (d) for all  $s \in \tau$ , let  $\rho(s) = \rho(s) + r$
  - (e) let  $s = s'$
3. for all  $s \in \tau$ ,  $V(s) = \gamma \cdot \rho(s)$

forever

# Python Code

# MCPE

- Goal: Learn  $v_\pi$  from episodes of experience under policy  $\pi$

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

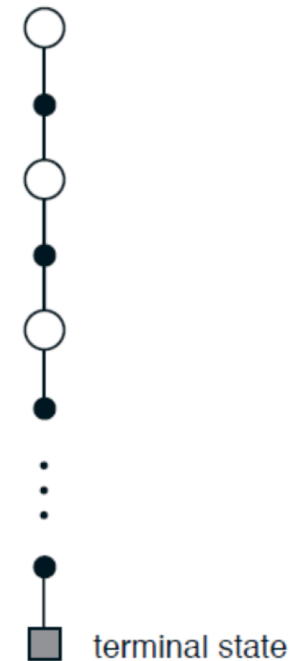
- Recall that the return is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

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

- Monte-Carlo policy Evaluation uses empirical mean return instead of expected return



# MCPE, First-Visit Monte-Carlo Policy Evaluation

- To evaluate state  $s$
- The first time-step  $t$  that state  $s$  is visited in an episode,
- Increment counter  $N(s) \leftarrow N(s) + 1$
- Increment total return  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = S(s)/N(s)$
- By law of large numbers,  $V(s) \rightarrow v_\pi(s)$  as  $N(s) \rightarrow \infty$

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

- Update  $V(s)$  incrementally after episode  $S_1, A_1, R_2, \dots, S_T$
- For each state  $S_t$  with return  $G_t$

$$\begin{aligned}N(S_t) &\leftarrow N(S_t) + 1 \\ V(S_t) &\leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))\end{aligned}$$

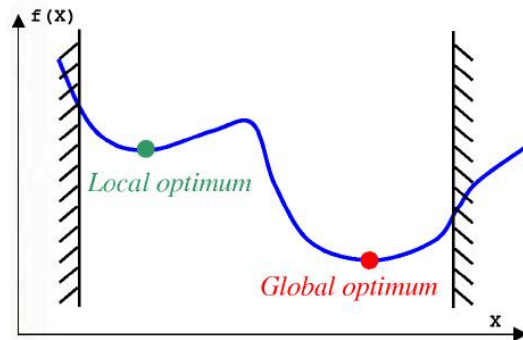
- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

# MCPE, w/ Exploration

Policy Table

State	U	D	L	R
(5,4)	-	-	0.98	0.02
(5,3)	0.03	-	0.97	-
...				
(1,1)	-	0.84	0.16	-



## • Problem of Greedy Policy Improvement

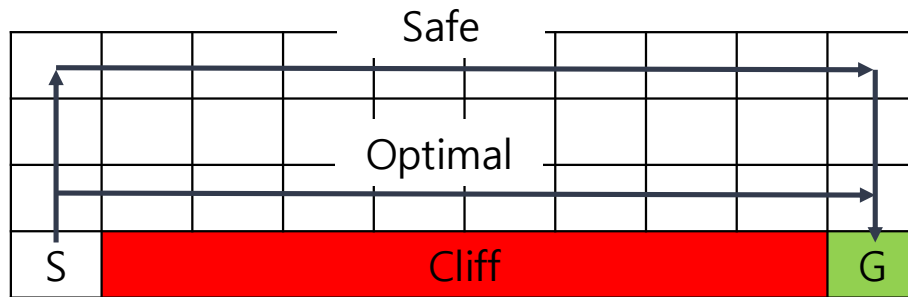
- Local Optimum

## • $\epsilon$ -greedy

- Simplest idea for ensuring continual exploration
- All  $m$  actions are tried with non-zero probability
- With probability  $1 - \epsilon$ , choose the greedy action
- With probability  $\epsilon$ , choose an action at random

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & \text{if } a^* = \arg \max_{a \in A} Q(s, a) \\ \frac{\epsilon}{m} & \text{otherwise} \end{cases}$$

# Q-Learning



	(1,2)	(1,3)		Trap
(2,1)		(2,3)	(2,4)	(2,5)
(3,1)				(3,5)
(4,1)	(4,2)	(4,3)		Open
		(5,3)	(5,4)	Goal

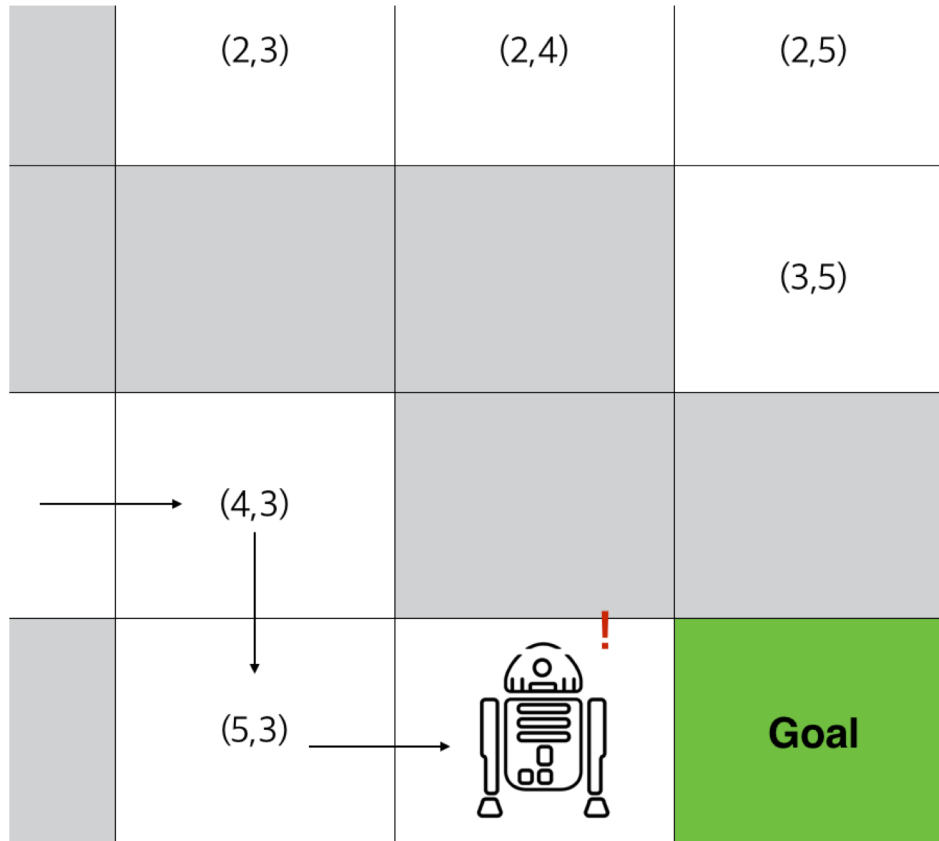
- MCPE는 과연 어떤 길을 찾아갈까?
- 최적화된 길을 가기 위해서는 어떤 방법이 좋을까?
  - MCPE는 Policy를 학습하니까..
  - 그리고, Policy에 의해 Action을 탐색하니까..
  - Policy에 의한 Action까지 학습하면 어떨까?!
- Q-Learning,
  - $Q(S_t, A_t)$ , 해당 state에서 해당 Action에 대한 값

# Q-Learning

주어진 State에서 주어진 Action에 의한 Reward의 기대값을 예측하는 함수 Q,  
Q를 학습함으로써, 최적의 정책을 학습하는 기법

.. Model FREE!! Algorithm

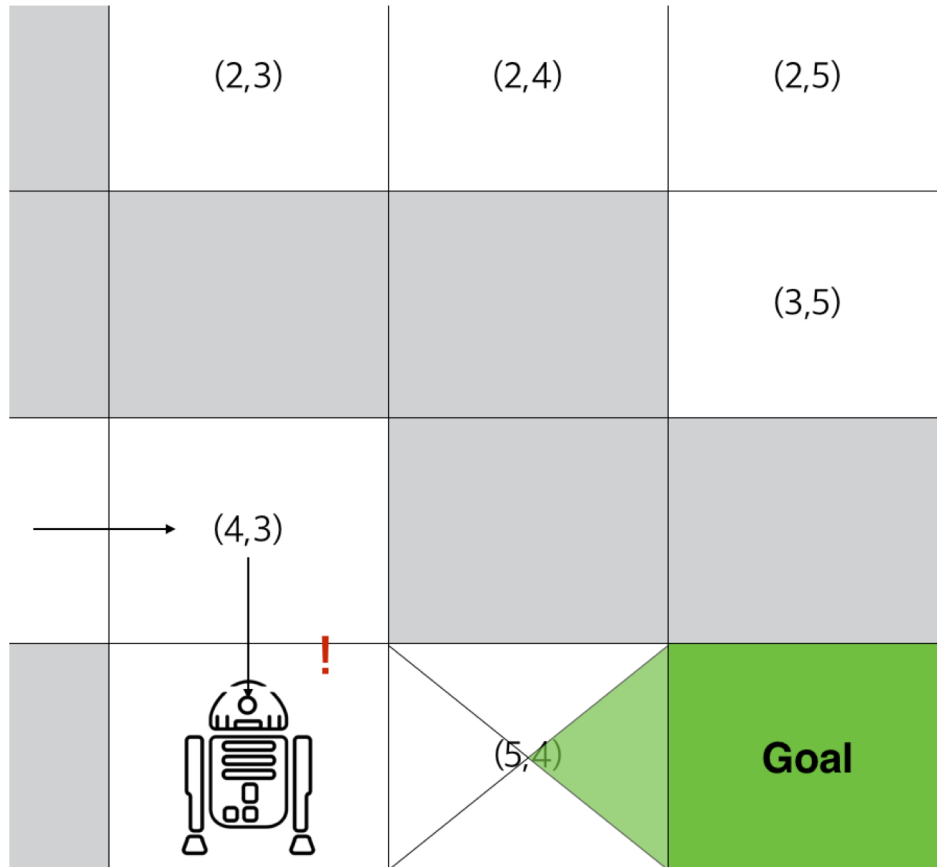
# Q-Learning



- 현재 위치에서,
  - 움직일 수 있는 후보지를 탐색한다.
  - 높은 Reward를 갖는 후보지로 이동한다.
  - 학습한다.
- (5,4)에서는 Right가 최적의 움직임이다.
  - $Q((5,4),R)$ 에 좋은 값을 넣어주자.
  - 그리고 따라간다.
- Goal에 도달하여 Episode 종료!

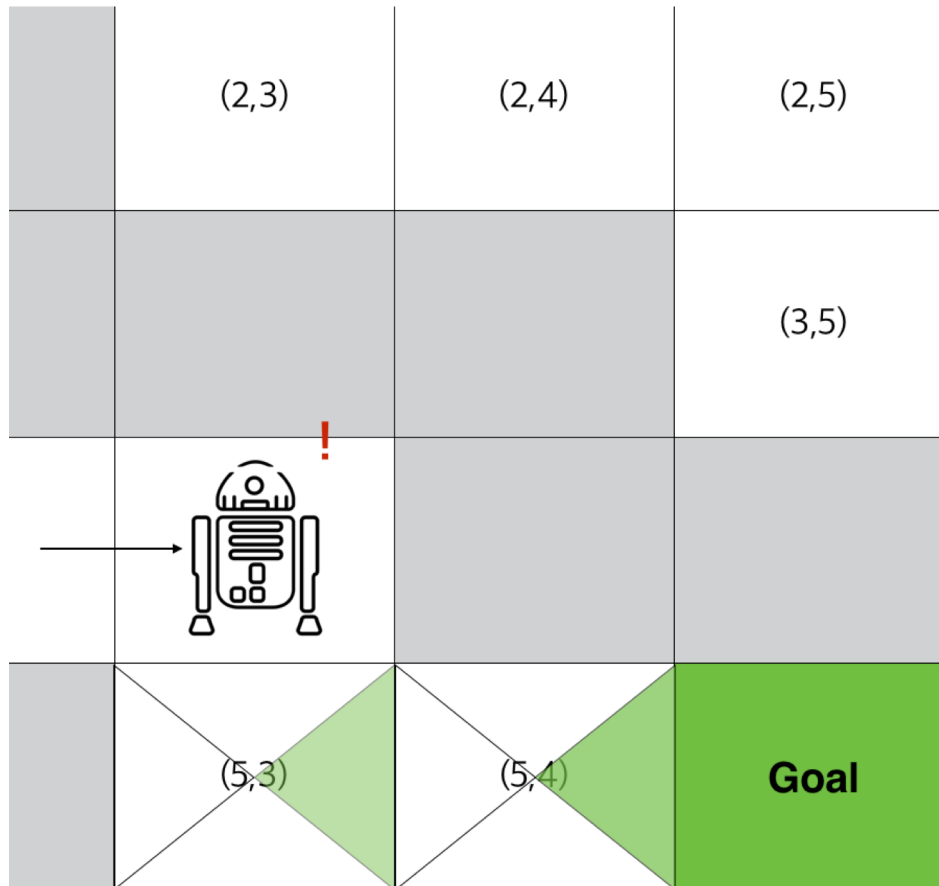


# Q-Learning



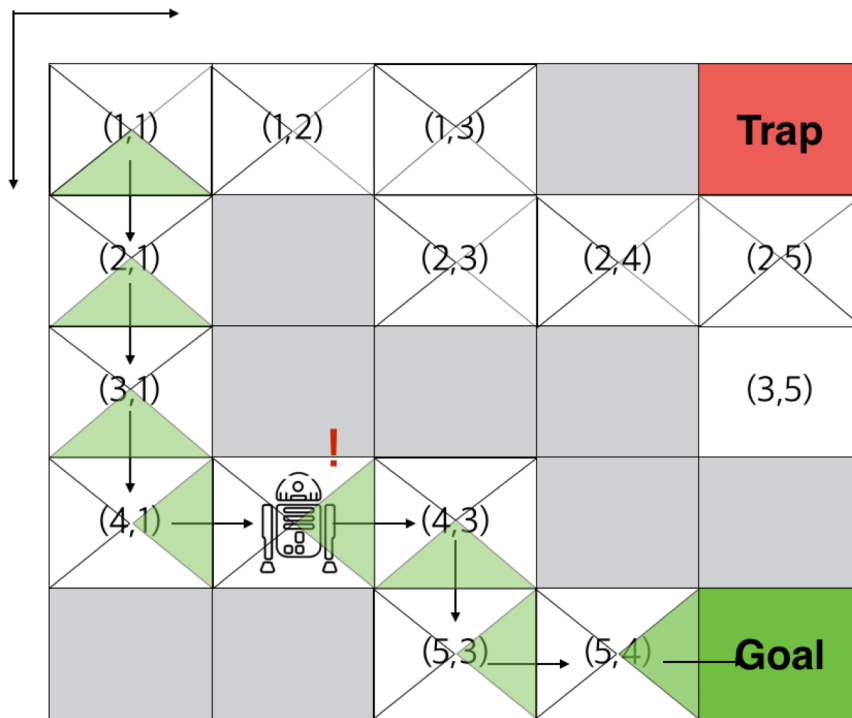
- 처음부터 돌다니면서,
  - 움직일 수 있는 후보지를 탐색한다.
  - 어차피 빈칸이니, 큰 값이 없어 랜덤하게..
- (5,3)에 도착하였더니, (5,4), R에 큰 값이 있다.
  - (5,4)로 갈 수 있도록,
  - $Q((5,3),R)$ 에 좋은 값을 넣어주자.
- (5,4)에 도착하였더니, 이미 갈 길이 정해져있다.
  - 골에 도달하였다!
  - Episode 종료

# Q-Learning



- 처음부터 돌다니면서,
  - 움직일 수 있는 후보지를 탐색한다.
  - 어차피 빈칸이니, 큰 값이 없어 랜덤하게..
- (4,3)에 도착하였더니, (5,3), R에 큰 값이 있다.
  - (5,3)로 갈 수 있도록,
  - $Q((4,3),D)$ 에 좋은 값을 넣어주자.
- (5,3)에 도착하였더니, 이미 갈 길이 정해져있다.
- (5,4)에 도착하였더니, 이미 갈 길이 정해져있다.
  - 골에 도달하였다!
  - Episode 종료

# Q-Learning



Trajectory	Q-Learning
$(5,4) \rightarrow (5,5)$	$Q((5,4),L) = \text{Reward} = 100$
$(5,3) \rightarrow (5,4) \rightarrow (5,5)$	$Q((5,3),L) = 0.7\max(Q((5,3),a)) = 70$
$(4,3) \rightarrow (5,3) \rightarrow (5,4) \rightarrow (5,5)$	$Q((4,3),D) = 0.7\max(Q((5,3),a)) = 49$
$(4,2) \rightarrow (4,3) \rightarrow (5,3) \rightarrow (5,4) \rightarrow (5,5)$	$Q((4,2),L) = 0.7\max(Q((4,3),a)) = 34.3$

- 최적화된 경로가 학습,  
 • (1,1)에서 goal인 (5,5)까지 최대 Reward를 갖는 state와 action을 따라간다.

# Q-Learning, Pseudo Code

Q\_LEARNING( $MDP, \pi, \epsilon$ )

Inputs      discount factor  $\gamma$

rate of exploration  $\epsilon$

Output      action-value function  $Q^*$

Initialize    $Q = 0$

1. initialize  $s, a$
2. while  $s \notin T$  do
  - (a) take action  $a$
  - (b) observe reward  $r$  and next state  $s'$
  - (c)  $Q_{(s,a)} = Q_{(s,a)} + [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
  - (d) choose action  $a'$
  - (e)  $s = s', a = a'$

## Python Code

# Q-Learning

- We now consider off-policy learning of action-values  $Q(s, a)$
- No importance sampling is required
- Next action is chosen using behaviour policy  $A_{t+1} \sim \mu(\cdot | S_t)$
- But we consider alternative successor action  $A' \sim \pi(\cdot | S_t)$
- And update  $Q(S_t, A_t)$  towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

# Q-Learning

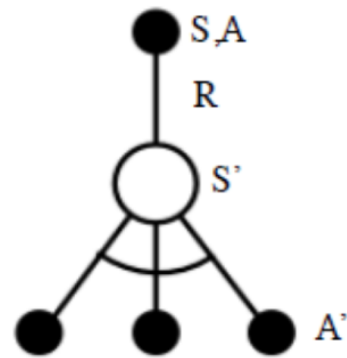
- We now allow both behaviour and target policies to **improve**
- The target policy  $\pi$  is greedy with respect to  $Q(s, a)$

$$\pi(S_{t+1}) = \arg \max_{a'} Q_{t+1}(S_{t+1}, a')$$

- The behaviour policy  $\mu$  is e.g.  $\varepsilon$ -greedy w.r.t.  $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ &= R_{t+1} + \gamma Q\left(S_{t+1}, \arg \max_{a'} Q(S_{t+1}, a')\right) \\ &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

# Q-Learning



$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ &= R_{t+1} + \gamma Q\left(S_{t+1}, \arg \max_{a'} Q(S_{t+1}, a')\right) \\ &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

Q-Learning control converges to the optimal action-value function,  $Q(S, A) \rightarrow q_*(s, a)$