RL Fundamental
+
Cooperative Multi-Agent RL Framework for Scalping Trading

발표자 : 조억

2020.01.17.

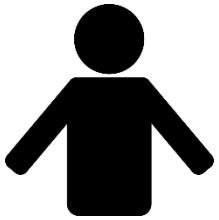
Data Mining Quality Analytics
hcai Human-Centered Artificial Intelligence Center

목적

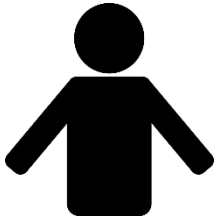
1. 강화학습 기초 개념부터 Deep Q Network까지 Grid World 예제 가지고 설명

2. 주식시장에 어떻게 환경과 에이전트에 대한 설계부터 구현했는지 과정 전달 드리고,

3. 주식 외 게임, 엘리베이터 등 환경 구현부터 에이전트 적용하며 느꼈던 점에 대해 공유



강화학습 처음이신 분



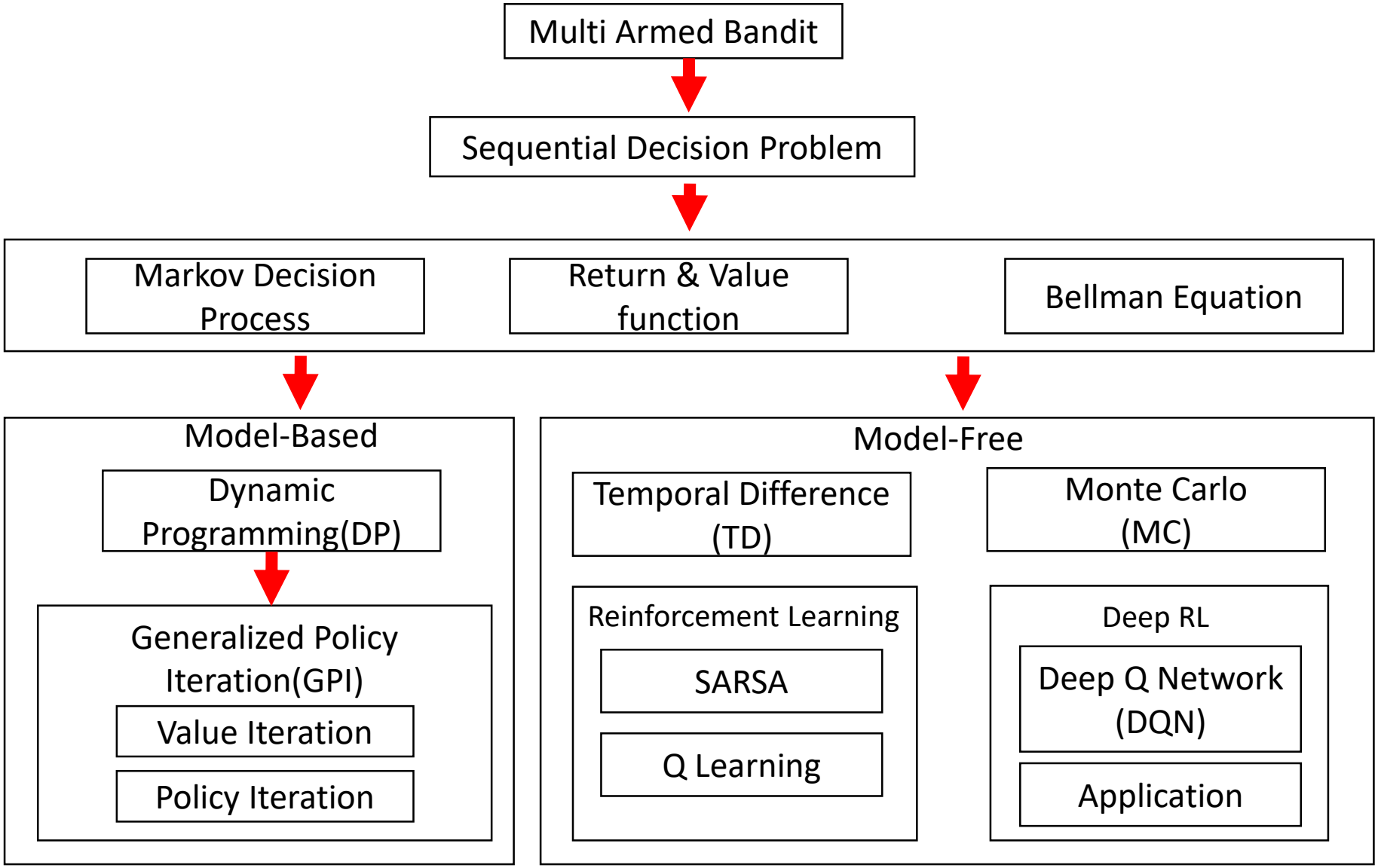
RL 기초지식과 경험 있는 분

강화학습에 대한 Application 적용 고민하는 분

목차

1. Reinforcement Learning

Reinforcement Learning



One Armed Bandit

강화학습으로 주식 시장을 정복하기 전에 우리는 간단한 카지노에 슬롯머신 게임을 생각해봅시다!!

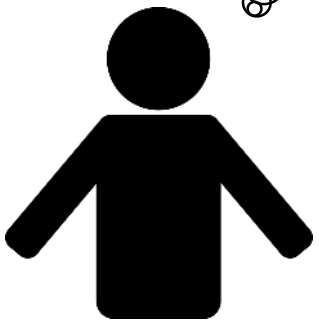
환경

한판 10 USD

에피소드

Action

에이전트



Action은 무조건 땡긴다.

언제까지?

1000$ 벌면 (WIN)
100$조차 다 잃거나(LOSE)

One Armed Bandit

Best case

보상

한방에 잭팟 터져 1000$를 벌거나

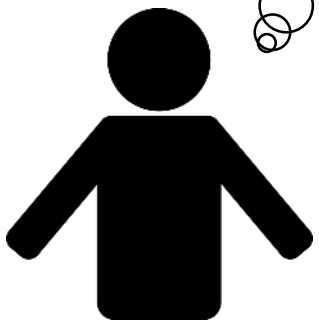
Worst Case

패널티

10판 (10 에피소드) 연속으로 0$(꽝)되서 100% 날리거나

보상이나 패널티를 1000~0으로 Scalar값으로 표현하며
리워드라고 부르자

Multi Armed Bandit

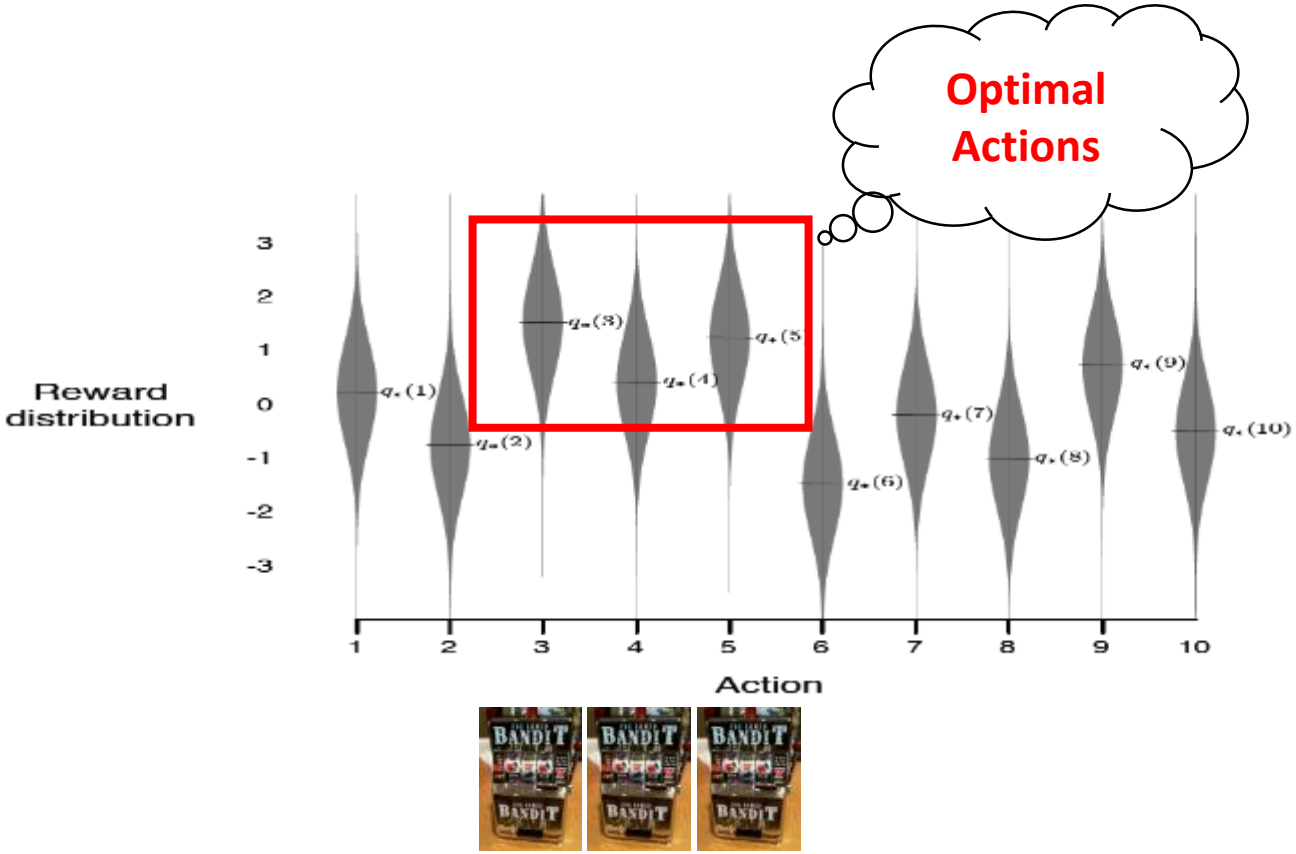


앗! 10개 중에 뭘 고르지?
=정책(Policy)

Action은 이제 10개의 슬롯머신 중 하나를 고르는 것

Multi Armed Bandit

각 슬롯 머신에 대한 보상을 아래와 같이 각각의 정규분포 샘플링한다고 가정



그렇다면, 우리가 기대하는건 3,4,5번의 슬롯머신을 에이전트가 택하는 것!

Multi Armed Bandit

자~ 이제 문제를 어떻게 풀어야 할까?!

Idea 1. (Action) Value Function

각 슬롯머신이 주는 보상(=**리워드**)을 모아서 기대값을 근사하는 **함수**가 필요가 있다.

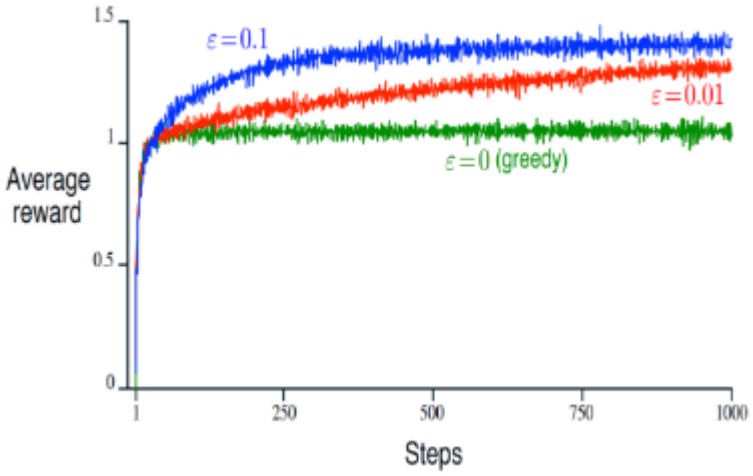
Idea 2. ϵ − greedy method

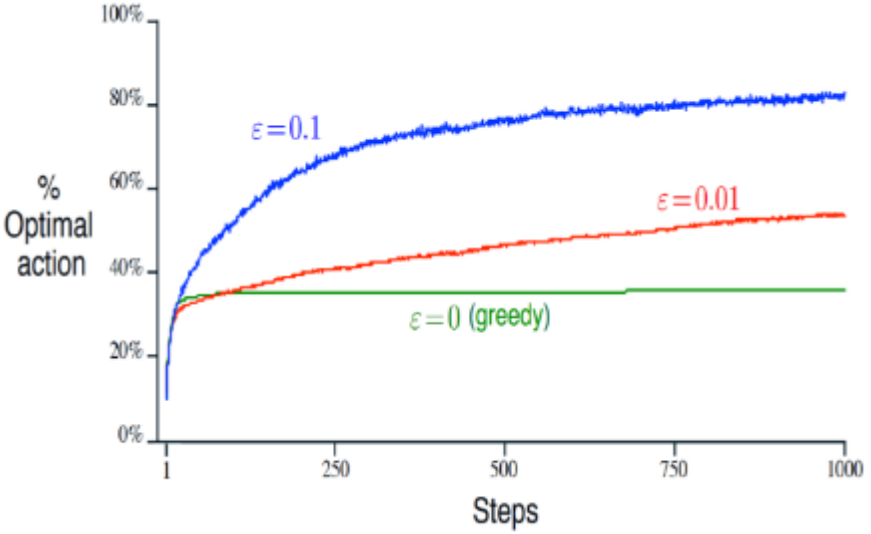
위 함수가 각 밴딧들의 샘플을 모을 수 있도록 10개 밴딧 골고루 선택할 필요가 있으며, 기대값에 근사가 되었다면 이제 그 함수를 계속 이용해야 한다.

- 학습하는 동안 **Exploration** : 다양한 경험을 위해 일부러 ϵ 의 확률만큼은 Non-Greedy actions한 액션을 확률적으로 뽑고 나머지 1- ϵ만큼은 Greedy action을 선택
- 학습이 완료되면 **Exploitation** : Greedy actions 중의 하나를 선택하는 것

Multi Armed Bandit

경험적으로 0.1의 확률(10%)로 Epsilon값을 주는 게 좋고 이는 이후 Q Learning에서도 보통 기본값으로 사용





Multi Armed Bandit

이 방법으로 우리는 현실 문제를 풀 수 있을까? No!

만약 슬롯 머신에서 각 머신이 주는 보상에 대해서 힌트 정보가 표시되는게 있다면? No, it can't

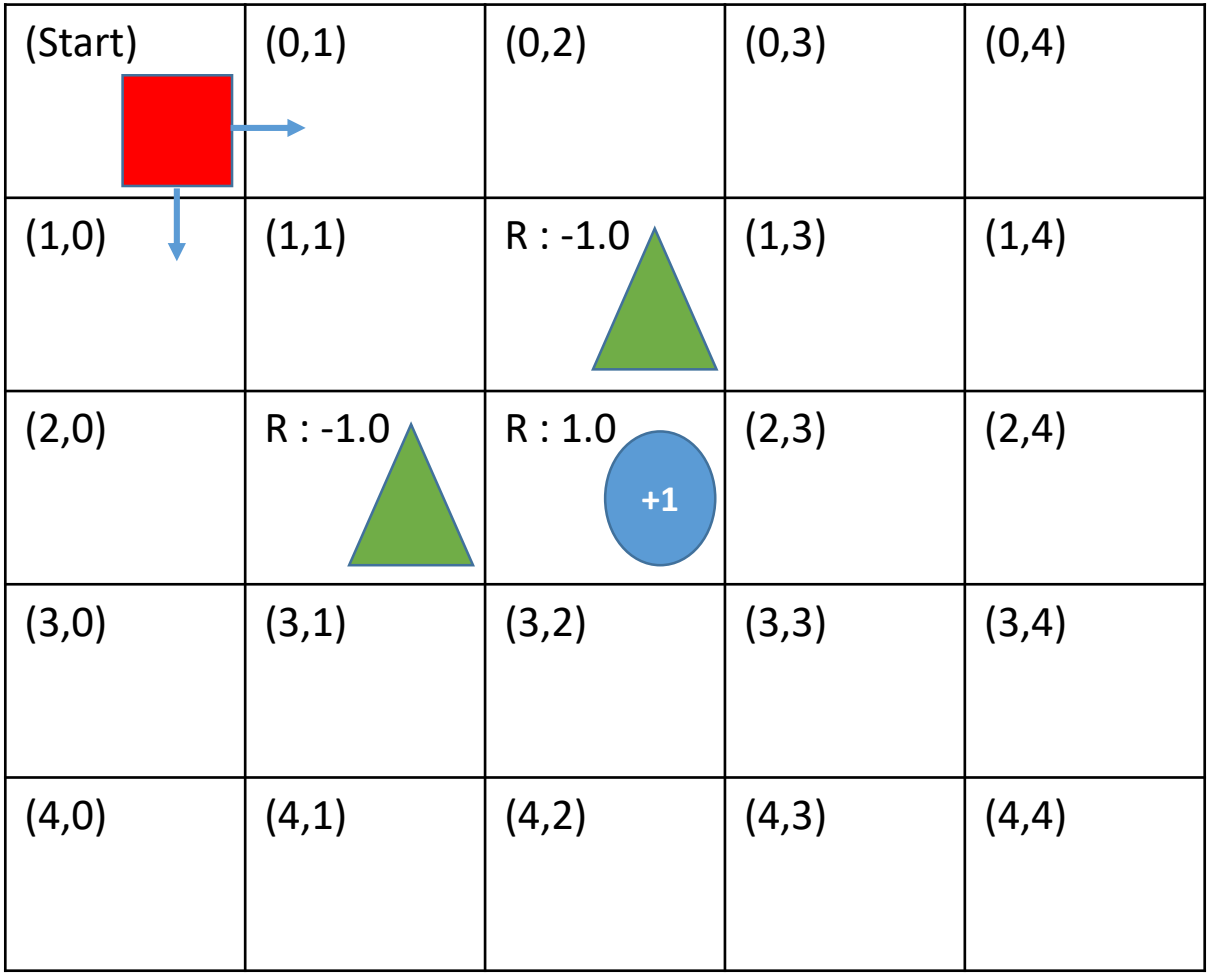
만약 한번의 결정이 아닌 특정 시간동안 여러 액션이 그 보상을 결정 짓는다면? No , it can't

	Multi armed bandit	Associative(=Context) search	Reinforcement Learning
액션 내릴 때 판단 기준 정보	X	O	O
시간 개념	X	X	O

Sequential Decision Problem

RL의 "Hello World"격인 Grid World라는 예제를 보자.

에이전트의 목표(Goal)은 시작지점에서 골인지점까지 장애물을 피해 최단 스탭으로 가야 한다.



Markov Decision Process

1. Sequential Decision Problem을 우리는 수학적으로 정의한 확률적 모델링
2. MDP 문제를 근사하여 푸는 다양한 솔루션 중의 하나가 Reinforcement Learning
   - ① 환경(Model)이 가지고 있는 Dynamics(불확실성)을 어떻게 정의를 할까?
   - ② 시간에 따라서 바뀌는 환경은 어떻게 해야 할 것인가? (**상태**)
   - ③ 일련의 의사결정은 어떻게 내려야 할까? (**시간**)

<구성요소>

$$\mathcal{MDP} = < S, A, P, R, \gamma >$$

S : 상태(State)
A : 행동(Action)
P : 상태변환확률(State transition probability)
R : 보상(Reward)
γ : 할인율(Discount factor)

<성질>

"Markov" property
(=memoryless property)

Markov Decision Process

상태(State)

State는 Agent가 매번 액션을 내릴 때 판단이 되는 상태의 집합 총칭

S = {(Start), (0,1), (0,2), (0,3) ,….(Goal),…. , (4,3),(4,4)}

MDP에선 $\tau = s_0, s_1, s_2, \ldots, s_T$

(Start)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	R : -1.0	(1,3)	(1,4)
(2,0)	R : -1.0	(Goal)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

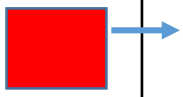
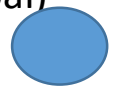
Markov Decision Process

액션(Action)

에이전트가 할 수 있는 행동의 집합 A = {위, 아래, 좌, 우}

시간 t에 취한 액션 $A_t = a$

밴딧에선 한판(=에피소드)가 하나의 시점(슬롯머신 팔을 댕기는 것) 이지만,

(Start) 	(0,1)	(0,2)	(0,3)	(0,4)
(1,0) 	(1,1)	R : -1.0 	(1,3)	(1,4)
(2,0)	R : -1.0 	(Goal) 	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Markov Decision Process

상태변환확률(State Transition Probability)

아래 선풍기를 틀어놨다고 치자. Deterministic에서 Stochastic한 상황

상태 (0,2)의 에이전트는 아래로 갈려고 했으나 오른쪽으로 바람에 밀려서 Action은 아래이나 다음 상태는 (0,3)

$$P^{a}_{ss'} = P[S_{t+1} = (0,3) \mid S_t = (0,2), A_t = DOWN] = 0.5$$
$$P^{a}_{ss'} = P[S_{t+1} = (1,2) \mid S_t = (0,2), A_t = DOWN] = 0.5$$

이 상태 변환 확률을 안다면 Model Based Learning, 모르면 Model Free Learning이라고 분류

(Start)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	R : -1.0	(1,3)	(1,4)
(2,0)	R : -1.0	(Goal)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

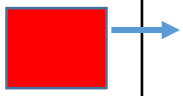
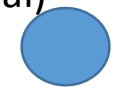
Markov Decision Process

리워드(Reward)

에이전트가 결국 추구하는 목표에 대한 힌트, 정보를 주는 스칼라 값

예) (1,2), (2,1)의 녹색 세모 모양이 있는 좌표(State)에 가면 -1의 리워드

(2,2)의 파란 동그라미 모양의 좌표(State) 에 가면 +1의 리워드

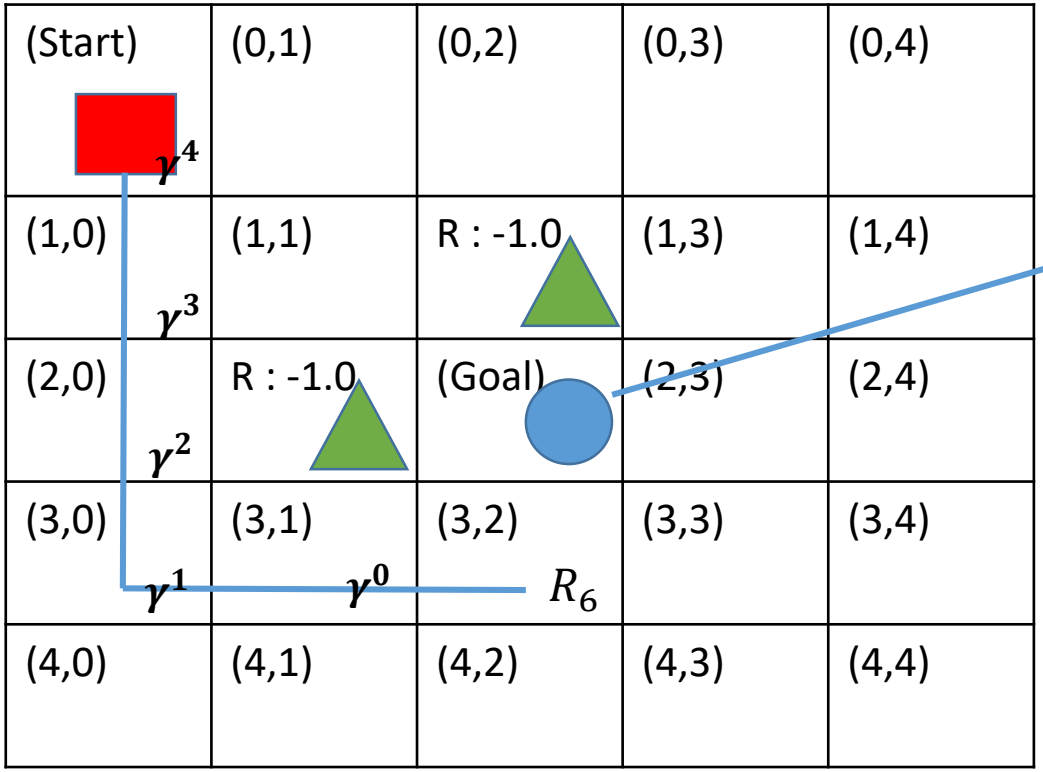
(Start) 	(0,1)	(0,2)	(0,3)	(0,4)
(1,0) 	(1,1)	R : -1.0 	(1,3)	(1,4)
(2,0)	R : -1.0 	(Goal) 	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Markov Decision Process

할인율(Discount factor)

0과 1사이의 값으로 현재 시점에서 미래 시점에서 받은 보상의 현재 가치를 계산할때 사용

이 값이 1 미만으로 설정하면 시간의 경과됨에 따라 할인이 들어가면서 최단 경로 찾기가 가능해짐



에피소드 종료!
스코어 계산 가능해짐

반환값(Return)

$$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \gamma^3 R_5 + \boldsymbol{\gamma^4 R_6}$$
$$G_2 = R_3 + \gamma R_4 + \gamma^2 R_5 + \gamma^3 R_6$$
$$G_3 = R_4 + \gamma R_5 + \gamma^2 R_6$$
$$G_4 = R_5 + \gamma R_5$$
$$G_5 = R_6$$

c.f.) $G_t \; of \; bandit = R_{t+1} + \mathrm{R_{t+2}} + \mathrm{R_{t+3}} + \; \ldots \; + \mathrm{R_T}$

Markov Decision Process

Markov Property

"Markov" property (=memoryless property) = the future only depends on the current state, not the history.

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, \dots, S_t]$$

(0,0)	(0,1)
(1,0)	(1,1)

만족

불만족

2. The Markov Property

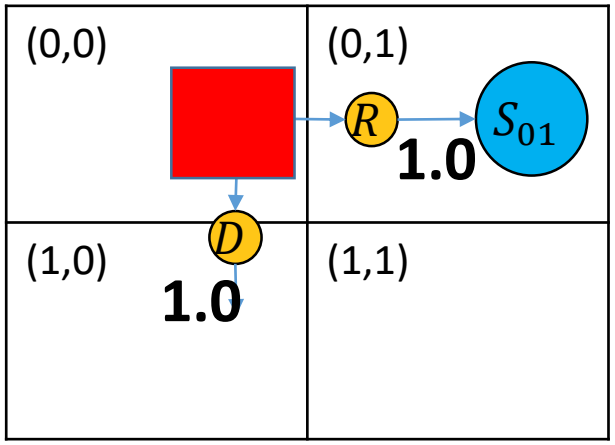
Note that although both transitions and rewards may be probabilistic, they depend only upon the current state and the current action: there is no further dependence on previous states, actions, or rewards. This is the *Markov property*. This property is crucial: it means that the current state of the system is all the information that is needed to decide what action to take—knowledge of the current state makes it unnecessary to know about the system's past.

It is important to note that the Markov properties for transitions and for rewards are not intrinsic properties of a real process: they are properties of the state-space of the model of the real process. Any process can be modelled as a Markov process if the state-space is made detailed enough to ensure that a description of the current state captures those aspects of the world that are relevant to predicting state-transitions and rewards.

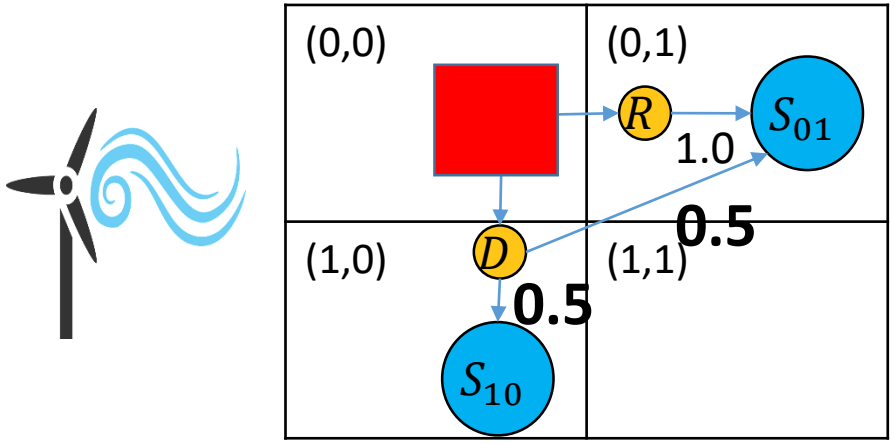
Q Learning저자 Watkins의 Markov property 해석

Markov Decision Process

MDP로 다양한 환경 표현할 수 있다.

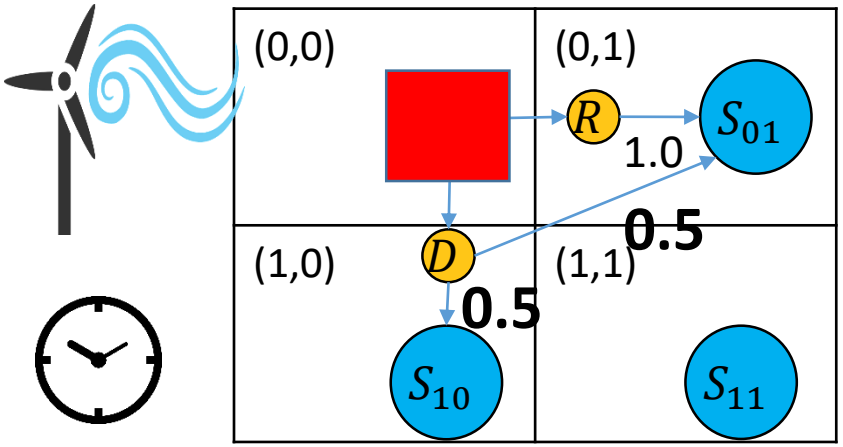


Deterministic Environment

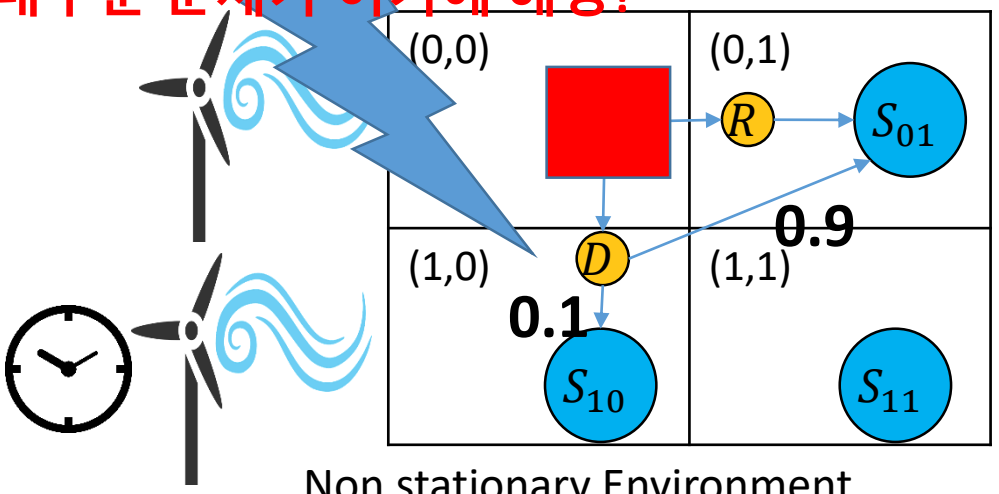


chastic Environment

대부분 문제가 여기에 해당!



Stationary Environment



Non stationary Environment

Return

MDP에선 단기적 보상만 고려하기 보단 에피소드 최종 점수(미래 누적 보상의 합) 최대화가 목표

Immediate Reward보단 최종적으로 받을 수 있는 보상의 합(반환값, Return)이 중요하다.

Bandit) $G_t = R_{t+1} + \mathrm{R_{t+2}} + \mathrm{R_{t+3}} + \ \ldots \ + \mathrm{R_T}$

MDP World) $G_t = R_{t+1} + \gamma R_{(t+2)} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

Where γ is a parameter, $0 < \gamma < 1$, called the discount rate

우리는 에피소드가 끝나는 시점에 Return을 알 수 있다.

Value Function

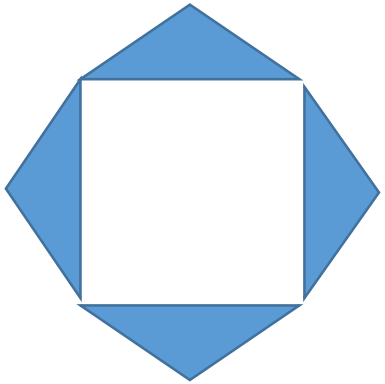
가치 함수(Value Function) : 특정 상태나 상태/액션에서 반환값에 대한 기대값(평균 가치)를 알려주는 함수

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

정책(Policy) : 각 상태에서 에이전트가 어떻게 판단을 해야할지 대한 기준을 주는 정보이며, 대부분 어떤 상태에서 어떤 액션을 선택할 확률

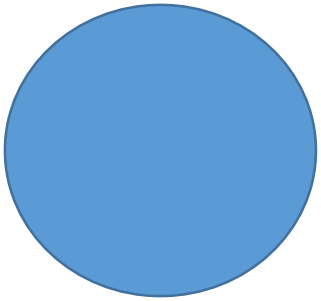
$$\pi(a|s) = P(A_t = a \,|\, S_t = s)$$

액션 가치 함수(Q function) : 가치 함수와 정책를 결합하여 특정상태에서 액션별 가치를 비교하기 위해서 Value function 대안으로 나온 함수



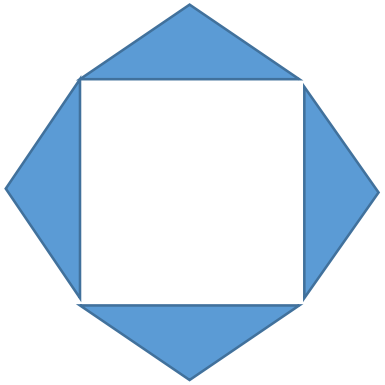
$\pi(s, a)$

주어진 π에서 액션 비교가능



$V(s)$

Action별 비교 불가



$Q_\pi(s, a)$

State + Action의 Action Value

Bellman Expectation Equation

가치 함수 / 큐 함수의 정의에서 벨만 기대 방정식(Bellman Expectation Equation)을 유도됨

가치 함수 Bellman Expectation Equation

$$v(s) = \mathbb{E}[G_t | S_t = s]$$
$$v(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s]$$
$$v(s) = \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \ldots) | S_t = s]$$
$$v(s) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}(R_{t+2} + \ldots) | S_t = s]$$
$$v(s) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}(G_{t+1}) | S_t = s]$$

$$\boldsymbol{v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]}$$

큐함수 Bellman Expectation Equation

$$q(s,a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$
$$q(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s, A_t = a]$$
$$q(s,a) = \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \ldots) | S_t = s, A_t = a]$$
$$q(s,a) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}(R_{t+2} + \ldots) | S_t = s, A_t = a]$$
$$q(s,a) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}(G_{t+1}) | S_t = s, A_t = a]$$

$$\boldsymbol{q(s,a) = \mathbb{E}[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]}$$

보통 정책 π마다 반환값에 영향을 받으므로 아래와 표기하기로 하자!

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

가치함수와 큐함수 관계

$$V_\pi(s) = \mathbb{E}_{a\sim\pi}[q_\pi(s,a) | S_t = s]$$
$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$

"이 방정식을 통해 현재 상태 S_t와 다음 상태 S_{t+1}의 가치 함수(큐함수) 관계 성립"

Bellman Optimality Equation

최고의 정책이 π^* 라면 그에 따른 Q함수도 최적 : $q^*(s,a) = \max_{\pi}[q_\pi(s)]$

최적 정책(optimal policy) ; $\pi^*(s) = argmax_{a \in A}[q^*(s,a)]$

π^*를 따르는 Value 함수와 Action Value함수와의 관계

$$v^*(s) = \max_a[q^*(s,a)|S_t = s]$$

$$v^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v^*(s_{t+1}|S_t = s]$$

가치함수에 대한 Bellman Optimality Equation

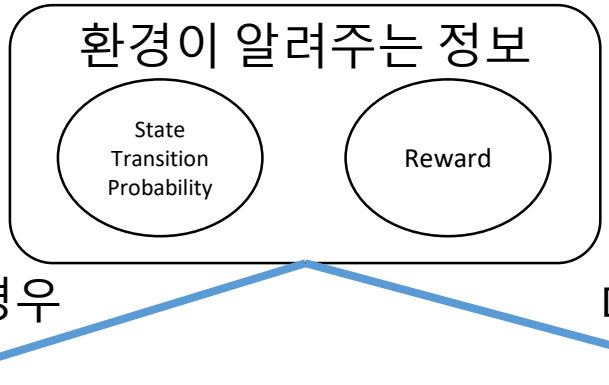
$$\boldsymbol{v^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1})|S_t = s, A_t = a]}$$

큐함수에 대한 Bellman Optimality Equation

$$\boldsymbol{q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') \mid S_t = s, A_t = a]}$$

How to get optimal policy?

v^*를 어떻게 구할 것인가?

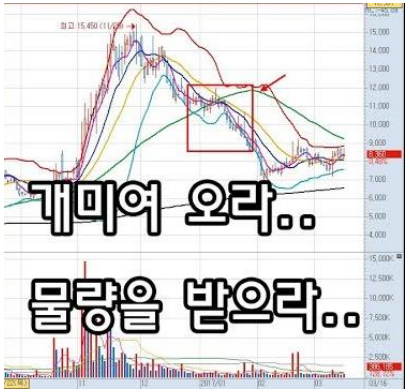


Model Free

"에이전트가 고생하며 샘플링"

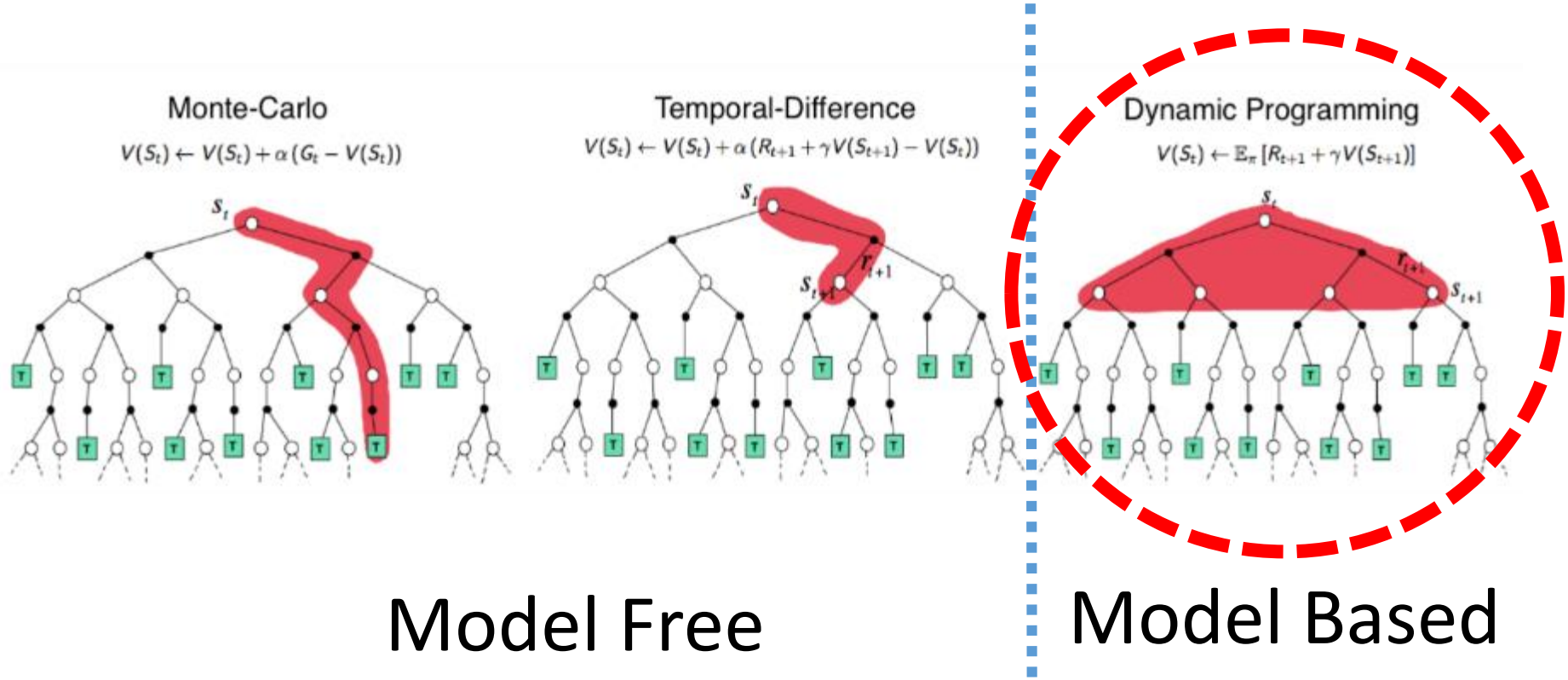
Model Based

"좀 더 우위에서 모든 경우 수 가능"



Markov Decision Process

Markov Decision Process 를 푸는 방법에는 아래 3가지가 있다. 가장 기초가 되는 Model Based learning인 Dynamic Programming 부터 살펴보자.



Model Free

샘플링

Model Based

모든 경우 수를 고려

Dynamic Programming

MDP 목표는 보상을 최대로 해주는 π^*이다.

어떤 임의의 정책 π에서 어떻게 π^* 로 업그레이드를 해나갈 수 있을까?

임의의 정책 π_k를 π_{k+1}로 업그레이드를 Iteration 반복하면서 가치함수를 벨만 기대/최적 방식으로 가치함수를 업데이트 계속 해보자

Dynamic Programming

Policy Iteration

임의의 π가 존재한다고 가정하고 Policy Iteration을 돌리는데 벨만 기대 방정식을 이용한다

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')]$$

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)[R_s^a + \gamma v_\pi(s')]$$

Deterministic 환경, $P_{ss'}^a = 1$

1. 정책 평가 : 위의 식을 가지고 오차가 $v_{k+1} - v_k$가 Δ 이하일때까지 반복

2. 정책 발전 : 1번 과정에서 나온 가치함수를 기준으로 아래 식대로 정책을 업데이트함

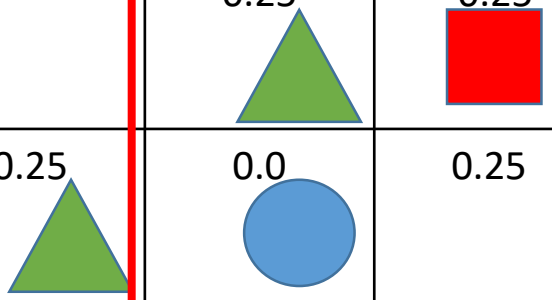
$$\pi_{k+1} \leftarrow argmax_{a \in A}[R_s^a + \gamma v_{k+1}(s')]$$

Dynamic Programming

Policy Iteration 중 Policy Iteration 예시

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)[R_s^a + \gamma v_\pi(s')]$$

0.0	0.0	-0.25	0.0	0.0
0.0		0.25	-0.25	0.0
-0.25	0.25	0.0	0.25	0.0
-0.0	-0.25	0.25	0.25	0.0
-0.0	0.0	0.0	0.0	0.0



1,3 좌표에서 에이전트가

위 : 0.25 * (0 + 0.9 X 0) = 0
아래 : 0.25 * (0 + 0.9 X 0) = 0
좌 : 0.25 * (-1 + 0.9 X 0) = -0.25
우 : 0.25 * (0 + 0.9 X 0) = 0

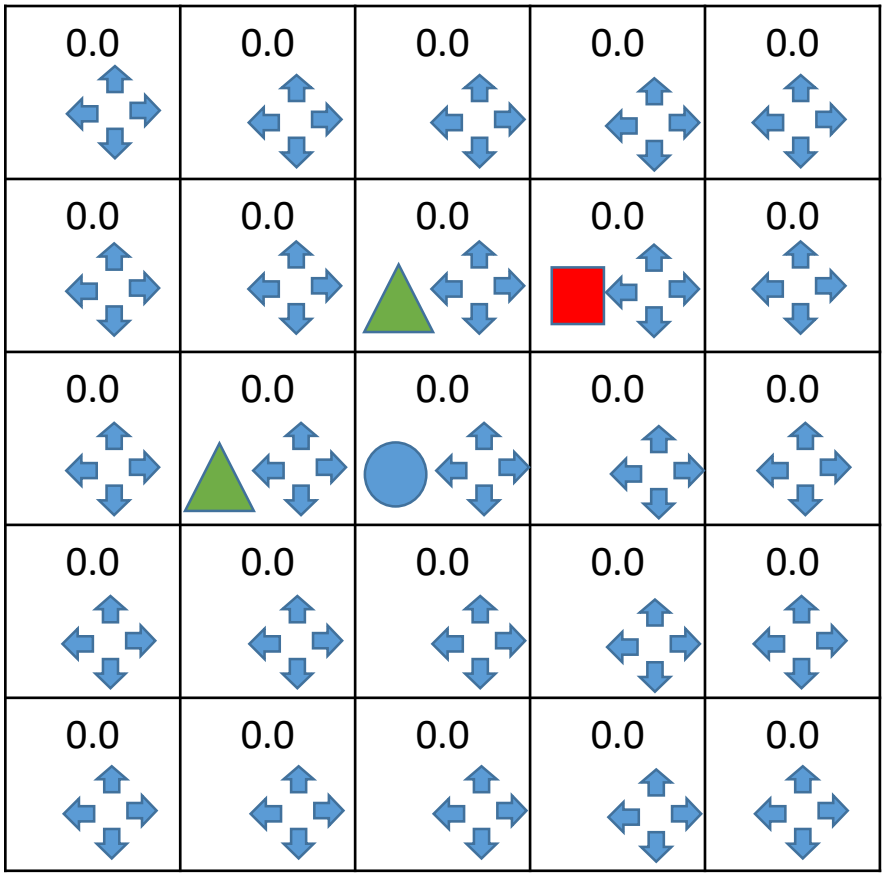
$v_{k+1}(s) = 0 + 0 - 0.25 + 0$
= -0.25

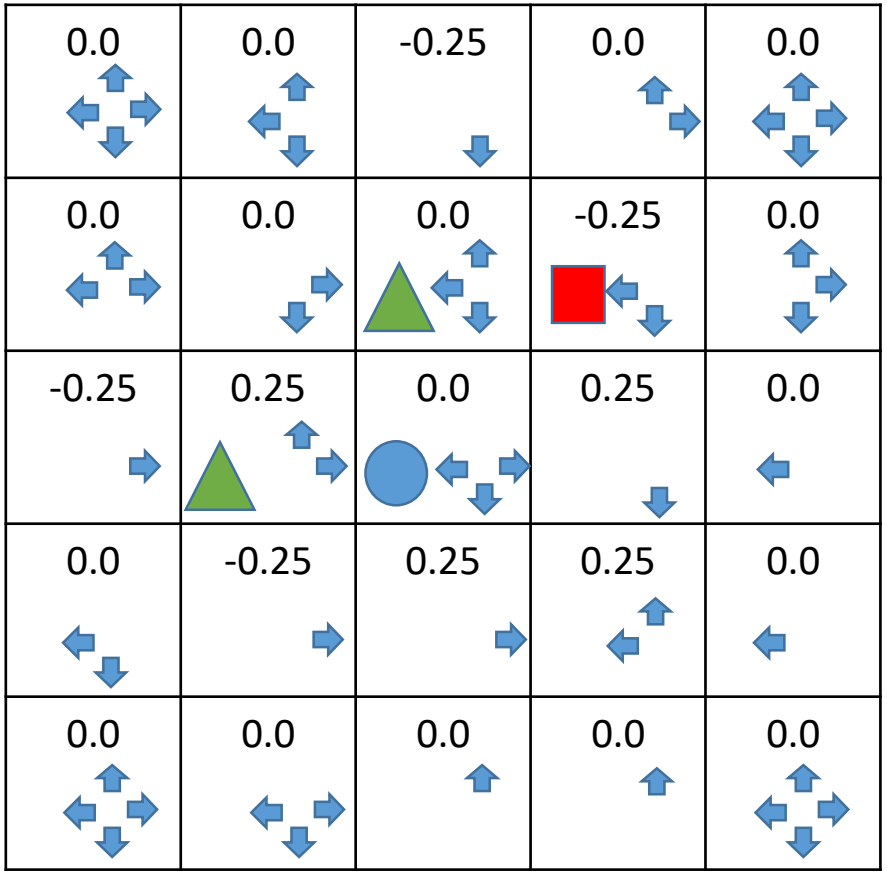
Dynamic Programming

Policy Improvement는 전 슬라이드의 업데이트된 가치 함수를 토대로 정책 탐욕적으로 발전시킨다.

$$\pi_{k+1}(s) \leftarrow argmax_{a \in A}[R_s^a + \gamma v_{k+1}(s')]$$

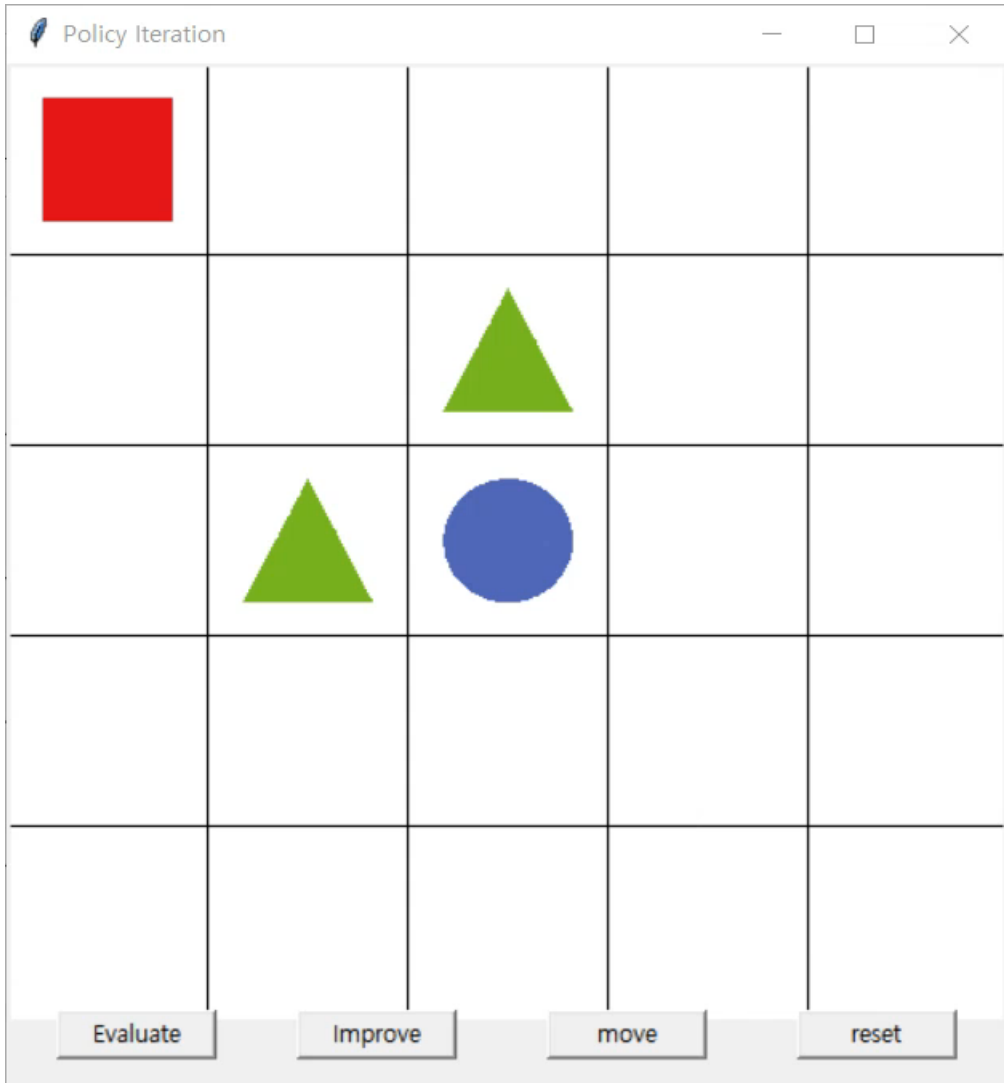
$\pi_k(s)$ → $\pi_{k+m}(s)$





Dynamic Programming

Policy Iteration 데모. 학습이 어느정도된 정책에 따라 에이전트는 최적의 행동으로 간다.



https://github.com/rlcode/reinforcement-learning-kr/tree/master/1-grid-world/1-policy-iteration

Dynamic Programming

Value Iteration

벨만 최적 방정식을 처음부터 만족한다고 가정하고 Value 함수 반복 업데이트

$$v^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}|S_t = s, A_t = a]$$

Policy Iteration에선 정책 평가와 발전이 따로 돌아가지만, 여기에선 하나의 이터레이션으로 돌아감.

$$v_k(s) \neq \max_a E[R_{t+1} + \gamma_k(S_{t+1})|S_t = s, A_t = a]$$

벨만 최적 방정식을 통해 가치함수를 업데이트하여 수렴 이후 greedy policy로 선택

Dynamic Programming

Value Iteration

모든 상태에 대해서 벨만 최적 방정식을 통해 1회 업데이트

$$for\ s \in S, v_{k+1} \leftarrow \max_a [R_s^a + \gamma \sum_{a \in A} P_{ss'}^a v_k(s')]$$

DP이기 때문에 가능
즉 Learning by Rollout이 아니라 Planning

쉬운 설명을 위해 $P_{ss'}^a$, 특정 action에서만 1로 떨어지는 Deterministic한 환경으로 고정!

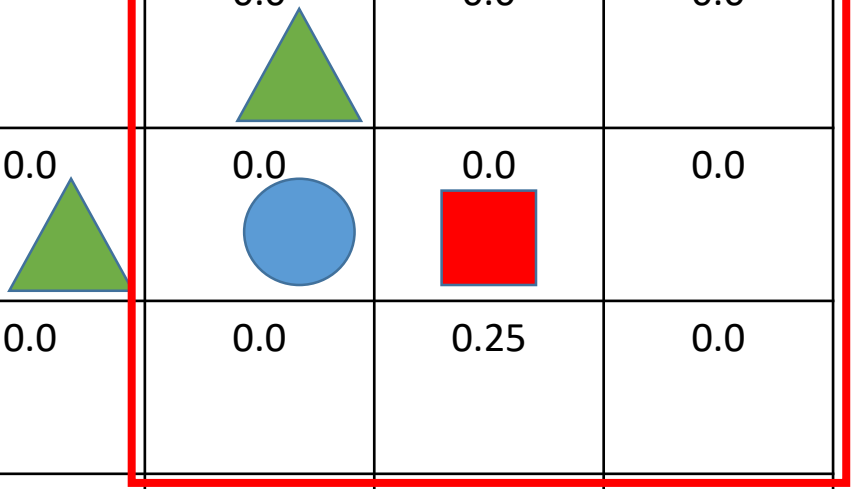
$$v_{k+1} \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

Dynamic Programming

Value Iteration의 경우는 어떨까? Policy Iteration보다 Iteration 수가 적어서 더 효율적

$$v_{k+1} \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

0.0	0.0	-0.25	0.0	0.0
0.0		0.0	0.0	0.0
-0.25	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.25	0.0
-0.0	0.0	0.0	0.0	0.0



1,3 좌표에서 에이전트가

위 : 0 + 0.9 X 0 = 0
아래 : 0 + 0.9 X 0 = 0
좌 : 1 + 0.9 X 0 = 0 = 1.0
우 : 0 + 0.9 X 0 = 0

$$v_{k+1} = max[0, 0, 1.0, 0] = 1.0$$

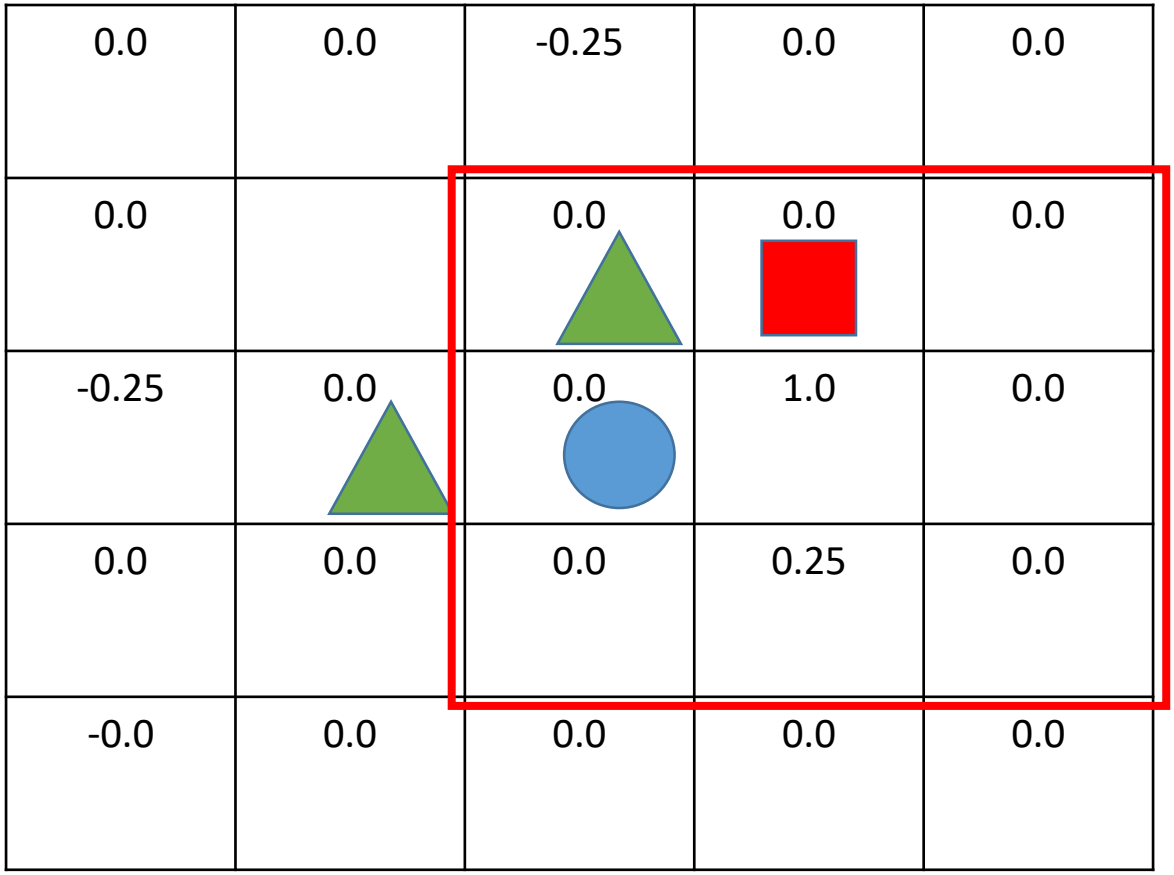
위 : 0.25 * (0 + 0.9 X 0) = 0
아래 : 0.25 * (0 + 0.9 X 0) = 0
좌 : 0.25 * (-1 + 0.9 X 0) = -0.25
우 : 0.25 * (0 + 0.9 X 0) = 0

Expectation을 위한 probability없음

Dynamic Programming

Value Iteration의 경우는 어떨까? Policy Iteration보다 Iteration 수가 적어서 더 효율적

$$v_{k+1} \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$



1,3 좌표에서 에이전트가

위 : 0 + 0.9 X 0 = 0
아래 : 0 + 0.9 X 1.0 = 0.9
좌 : -1 + 0.9 X 0 = 0 = -0.1
우 : 0 + 0.9 X 0 = 0

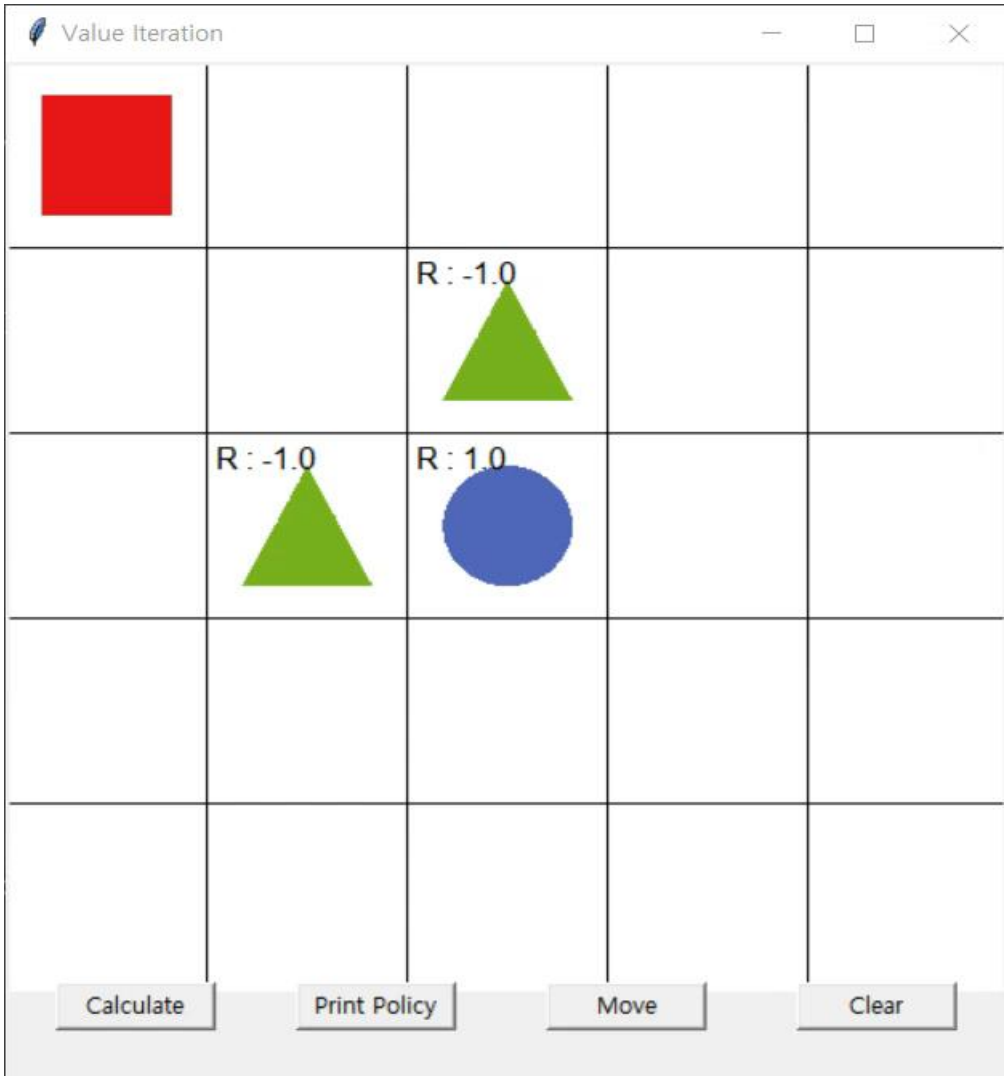
$$v_{k+1} = max[0, 0.9, -0.1, 0] = 0.9$$

위 : 0.25 * (0 + 0.9 X 0) = 0
아래 : 0.25 * (0 + 0.9 X 0) = 0
좌 : 0.25 * (-1 + 0.9 X 0) = -0.25
우 : 0.25 * (0 + 0.9 X 0) = 0

Expectation을 위한 probability없음

Dynamic Programming

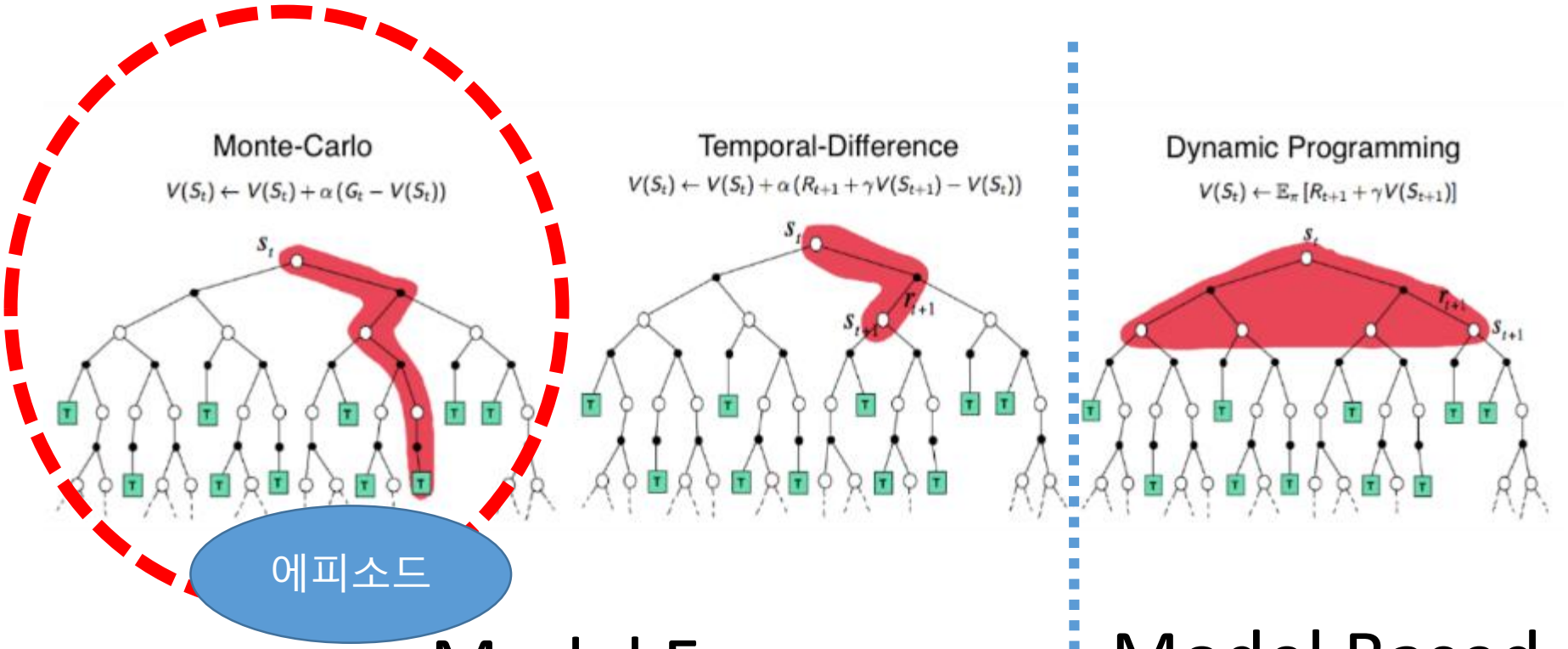
Value Iteration 데모. 학습이 어느정도된 정책에 따라 에이전트는 최적의 행동으로 간다.



https://github.com/rlcode/reinforcement-learning-kr/tree/master/1-grid-world/2-value-iteration

Markov Decision Process

Markov Decision Process 를 푸는 방법에는 아래 3가지가 있다. 가장 기초가 되는 Model Based learning인 Dynamic Programming 부터 살펴보자.



Model Free

샘플링하여 환경 정보를 에이전트가 근사

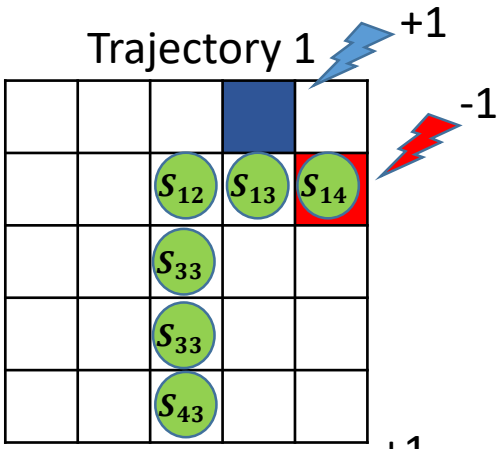
Model Based

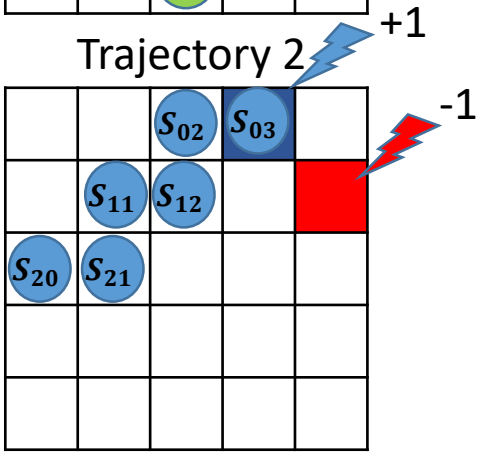
모든 경우 수를 고려

Monte Carlo Estimation

MC는 얘기했듯이 에피소드를 한번에 쭉 가고 마지막에 발생하는 반환값(환경에서 주는 확실한 값)을 사용하기 때문에 Unbiased estimator라고 할 수 있지만 학습 데이터가 호흡이 길어 Variance가 심하다. 적용은 아주 쉬움!

$$V(s_t) \leftarrow R_t$$





T	1	2	3	4	5	T
τ	s_{43}	s_{33}	s_{33}	s_{12}	s_{13}	s_{14}
G_t	0.59	0.65	0.729	0.81	0.9	1

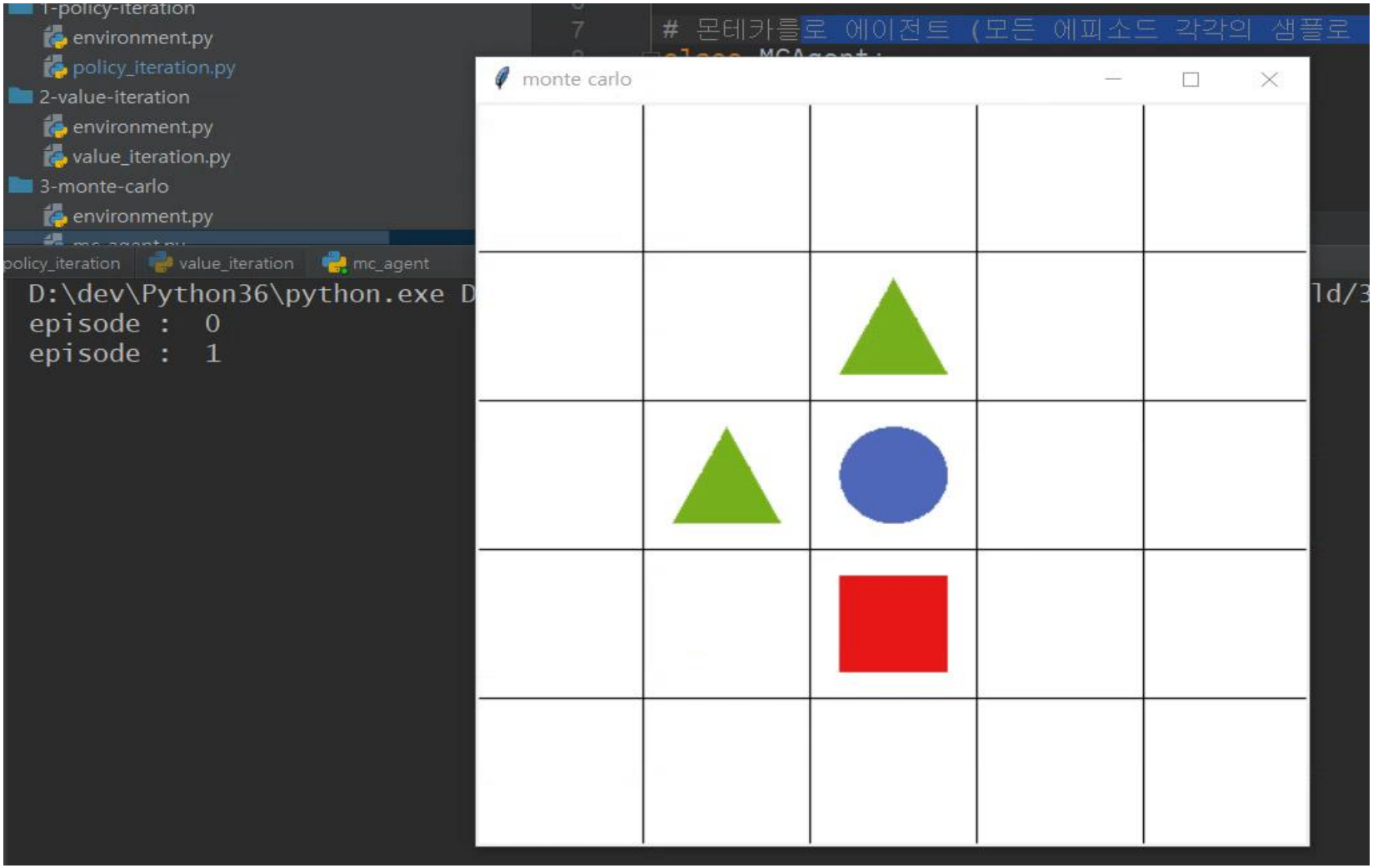
x γ x γ x γ x γ x γ

$V(s_{12}) = 0$

T	1	2	3	4	5	T
τ	s_{20}	s_{21}	s_{11}	s_{12}	s_{02}	s_{03}
G_t	0.59	0.65	-0.729	-0.81	-0.9	-1

Monte Carlo Estimation

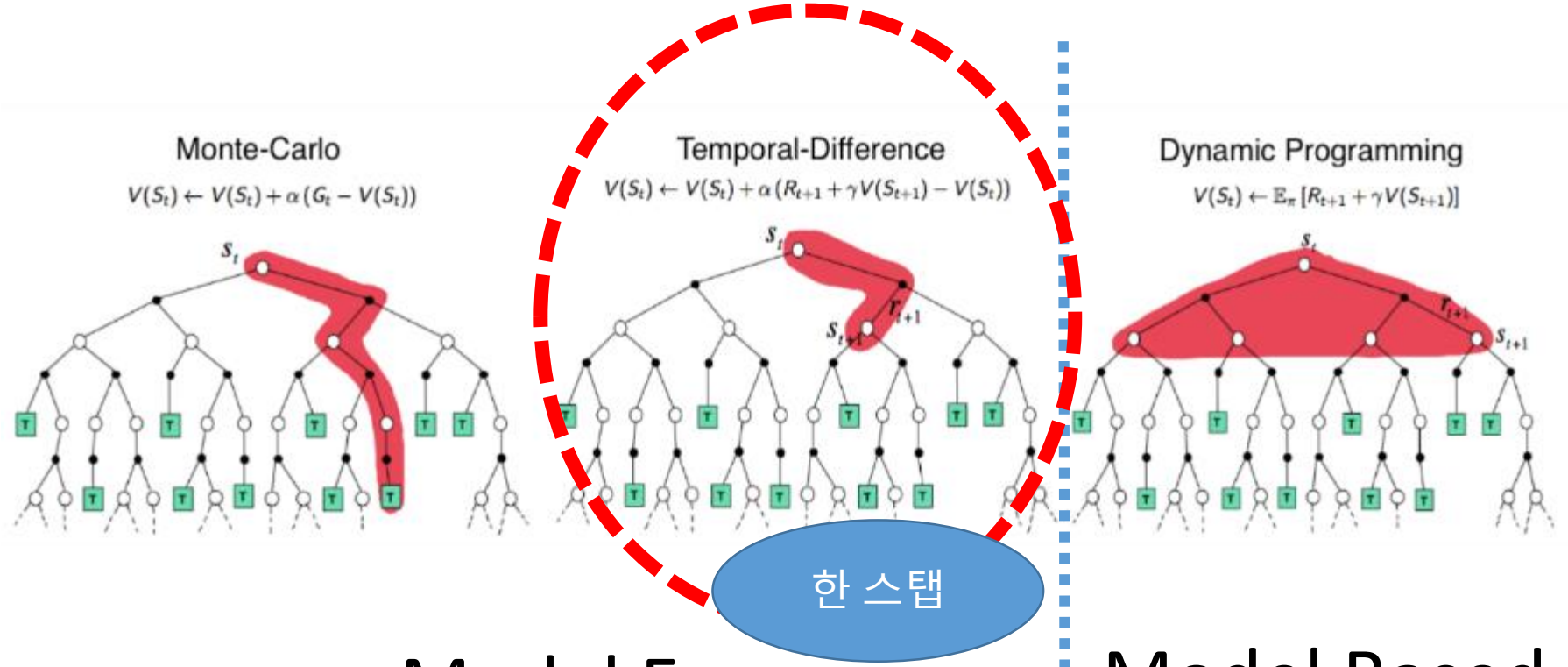
몬테카를로 예측 예제 실행 데모



https://github.com/rlcode/reinforcement-learning-kr/tree/master/1-grid-world/3-monte-carlo

Markov Decision Process

Markov Decision Process 를 푸는 방법에는 아래 3가지가 있다. 가장 기초가 되는 Model Based learning인 Dynamic Programming 부터 살펴보자.



Model Free

샘플링하여 환경 정보를 에이전트가 근사

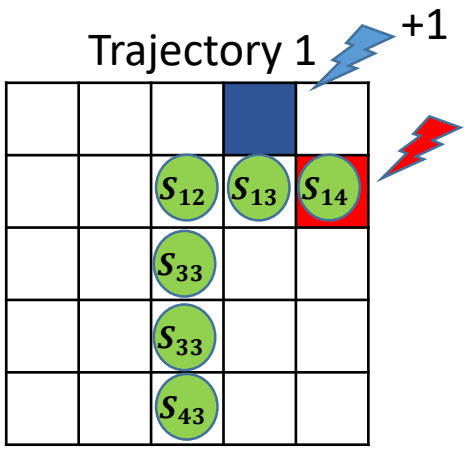
Model Based

모든 경우 수를 고려

Temporal Difference

Temporal Difference Variance는 적고 Bias에 취약하다. 그렇지만 아주 좋은 점이 있다 왜일까?

$$V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$$



T	1	2
τ	s_{43}	s_{33}
$V(S_t)$	0.59	0.65

T	2	3
τ	s_{33}	s_{33}
$V(S_t)$	0.65	0.729

T	3	4
τ	s_{33}	s_{12}
$V(S_t)$	0.729	0.81

T	4	5
τ	s_{12}	s_{13}
$V(S_t)$	0.81	0.9

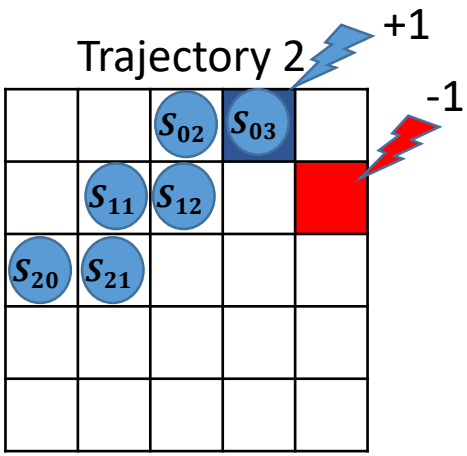
T	5	T
τ	s_{13}	s_{14}
$V(S_t)$	0.9	1

학습 도움이 될 수 있는 샘플은 Trajectory당 5개!

Temporal Difference learning

Temporal Difference Variance는 적고 Bias에 취약하다. 그렇지만 아주 좋은 점이 있다 왜일까?

$$V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$$



T	1	2
τ	s_{20}	s_{21}
$V(S_t)$	-0.59	-0.65

T	2	3
τ	s_{21}	s_{11}
$V(S_t)$	-0.65	-0.729

T	3	4
τ	s_{11}	s_{12}
$V(S_t)$	-0.729	-0.81

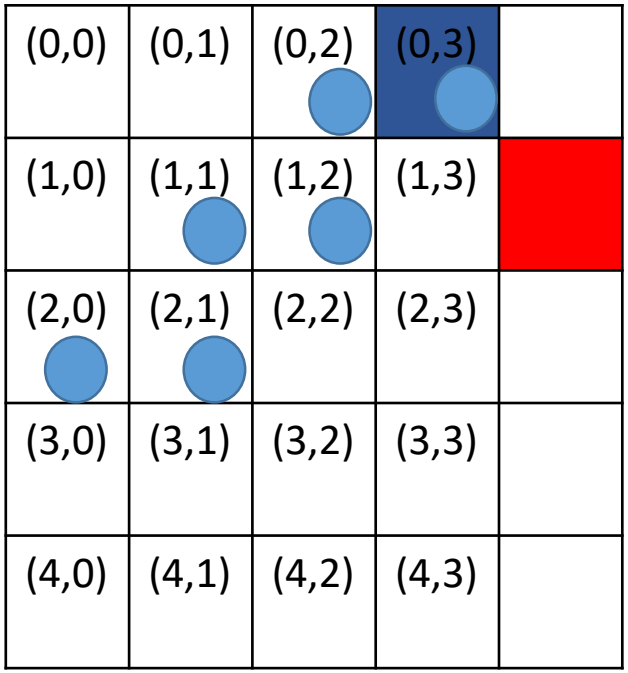
T	4	5
τ	s_{02}	s_{12}
$V(S_t)$	-0.81	-0.9

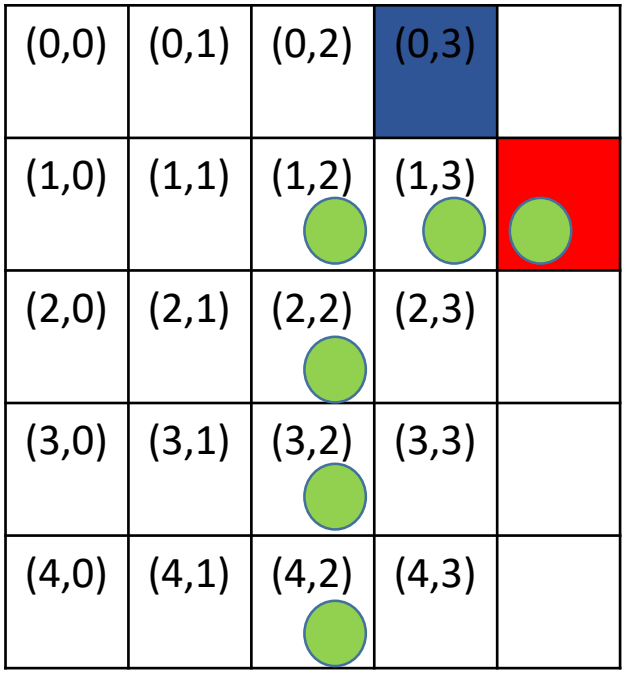
T	5	T
τ	s_{02}	s_{03}
$V(S_t)$	-0.9	-1

학습 도움이 될 수 있는 샘플은 Trajectory당 5개!

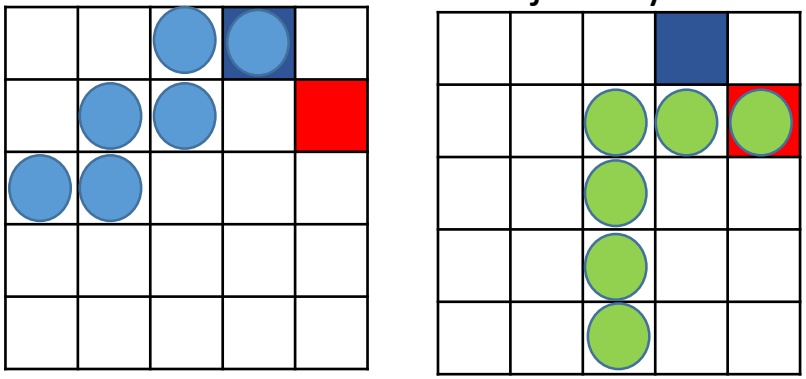
The Paths Perspective

MC는 2개의 실제 Trajectory를 가지고 **TD는 4개의 Trajectory**를 만들어 낼 수 있다.

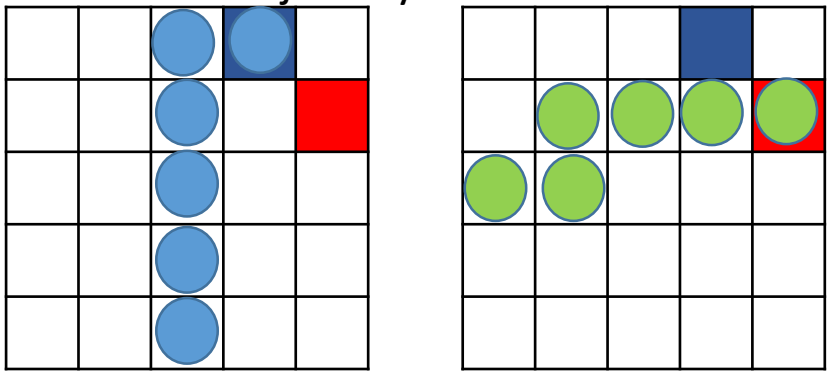




Trajectory 1



Trajectory 2



TD vs MC

Trajectory를 병합한다는 것은 어떤 의미일까?

주목할 점은 $V(s_{t+1})$는 s_{t+1}의 TD update 대상들이 대는 모든 것의 기대값

$$V(s_{t+1}) \approx \mathbb{E}[r'_{t+1} + \gamma V(s'_{t+2}) \approx \mathbb{E}[r'_{t+1}] + \gamma \mathbb{E}[V(s'_{t+2})]$$

우리는 이 식을 재귀적으로 TD update rule로 확장해볼 수 있다.

$$V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$$

$$\leftarrow r_t + \gamma \mathbb{E}[r'_{t+1}] + \gamma^2 \mathbb{E}[V(s''_{t+2})$$

$$\leftarrow r_t + \gamma \mathbb{E}[r'_{t+1}] + \gamma^2 \mathbb{E}\mathbb{E}[r''_{t+2}] + \cdots$$

MC	$V(s_t) \leftarrow r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ \ldots$	Unbiased but High Variance
TD	$V(s_t) \leftarrow r_t + \gamma \mathbb{E}[r'_{t+1}] + \gamma^2 \mathbb{E}\mathbb{E}[r''_{t+2}] + \ \ldots$	Biased but Low Variance

좋은 예측은 작은 분산으로 예측이 가능해야 한다는 점에서 볼때,
TD가 MC를 더 많은 아이템으로 가지고 기대값을 구할 수 있으므로 더 좋음.

$$Var[V(s)] \propto \frac{1}{N}$$

TD Learning

Q 함수를 사용하여 Temporal Difference 업데이트할 때 Model-Free Learning으로 **미래 시점인 $V(S_{t+1})$를 어떻게 예측하는지가 중요**한데, 3가지 방법에 대해 알아보자.

$$Q(s_t, a_t) \leftarrow r_t + \gamma V(s_{t+1})$$

① SARSA : 가장 간단하며, $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ 샘플 필요

$$V(s_{t+1}) = Q(s_{t+1}, a) \times a_{t+1}$$

Experience(Empirical Distribution)

② Expected SARSA : 기대값 이용하여 예측

$$V(s_{t+1}) = Q(s_{t+1}, a) \times \pi(s_{t+1}, a)$$

Expectation(True Policy Distribution)

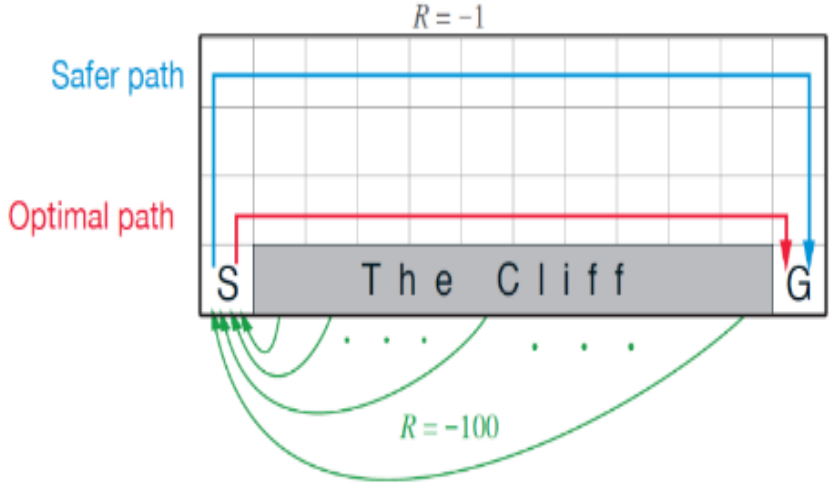
③ Q Learning : $V(s_{t+1})$을 예측을 현재 정책으로 사용. Off Policy가 가능

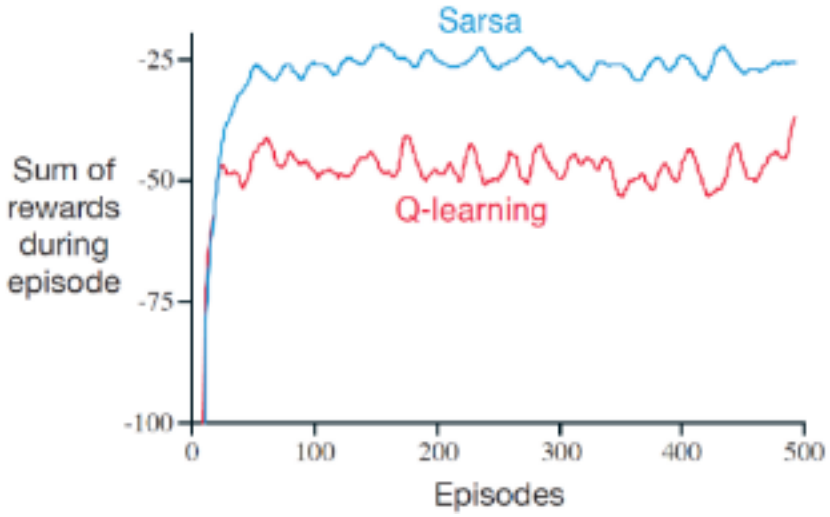
$$V^{\pi^*}(s_{t+1}) = Q^{\pi^*}(s_{t+1}, a) \times argmax_a Q^{\pi^*}(s_{t+1}, a)$$

TD Learning

Q Learning vs SARSA

SARSA가 Q Learning보다 Grid World 문제의 장애물등을 더 우회해서 가는 경향이 있음.

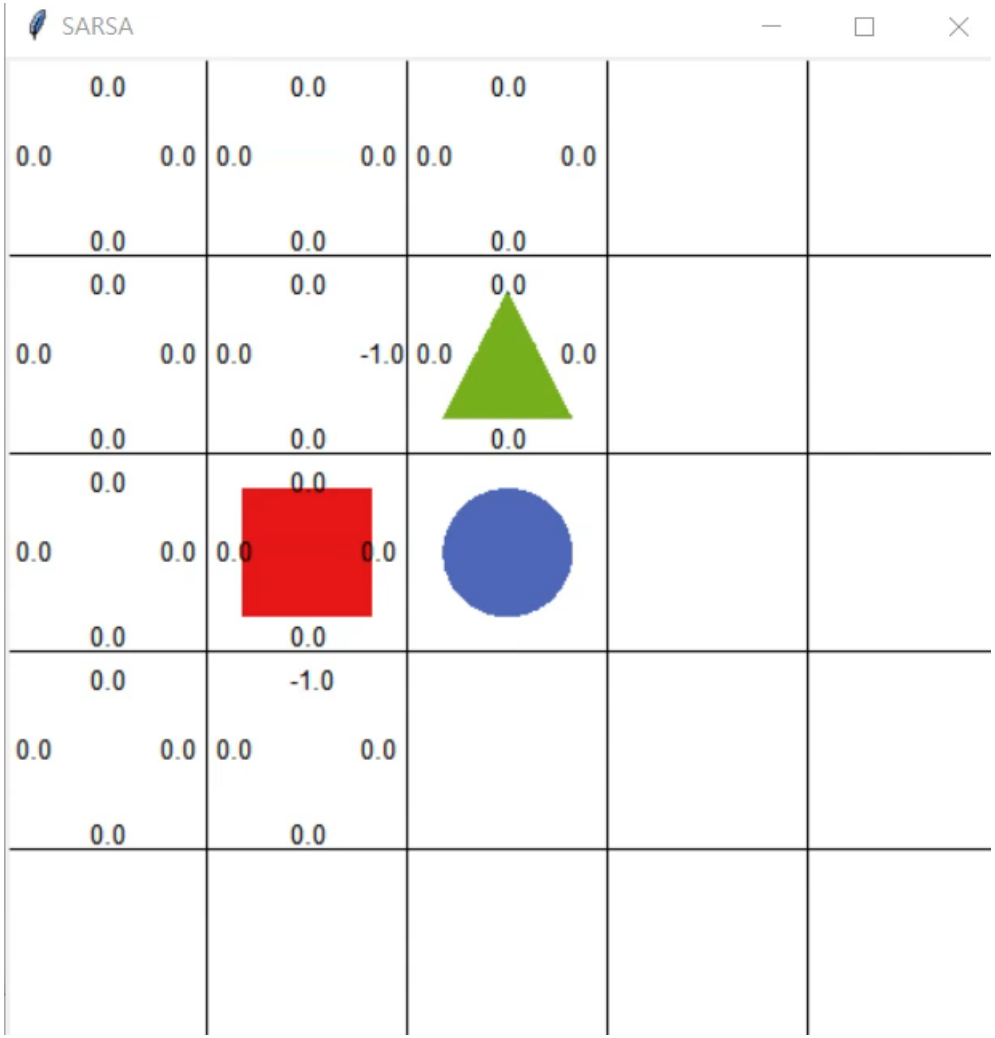




https://gitlab.com/verystrongjoe/master-rl-for-two-days/blob/master/day1/td/cliff_walking.py

TD Learning

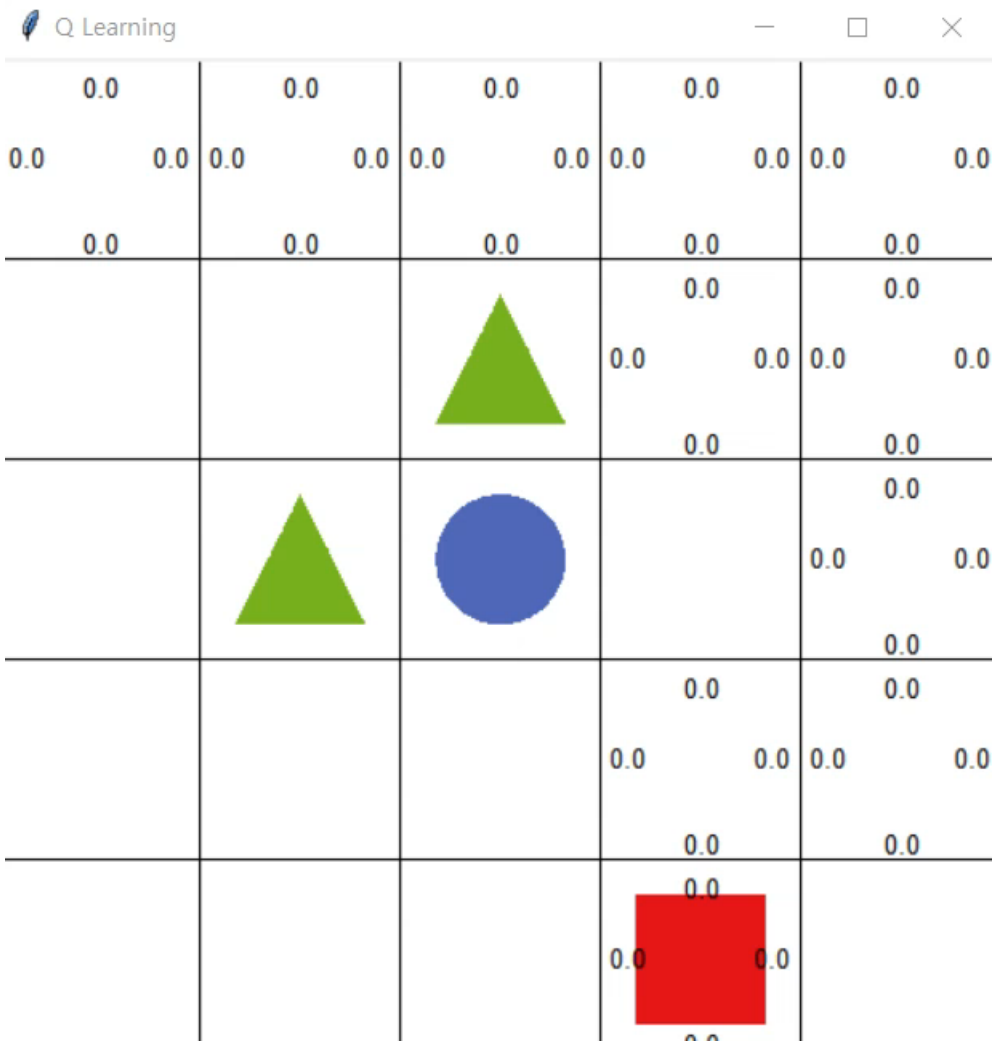
SARSA 데모



https://github.com/rlcode/reinforcement-learning-kr/tree/master/1-grid-world/4-sarsa

TD Learning

Q Learning 데모



https://github.com/rlcode/reinforcement-learning-kr/tree/master/1-grid-world/5-q-learning

Double Q Learning

Q Learning은 over-estimated하는 경향 있고 특히 noisy reward가 많게 되면 over optimistic!!

"예를 들면, 승섭이가 카지노에 가서 100개의 머신 중 38번째 머신에 앉아서 플레이를 했더니 잭팟이 터졌다. 보통 DQN 같은 경우 그 카지노에 대해서 아주 좋게 볼 것이다. 하지만 Double Q Learning은 또 하나의 Q를 두어서 실제로 그렇지 않는 케이스를 인지시켜주는 역할을 한다. "

"우왕! 이 카지노 대박"

"여보(another Q)! 아니야~ 운일 뿐이었어!"

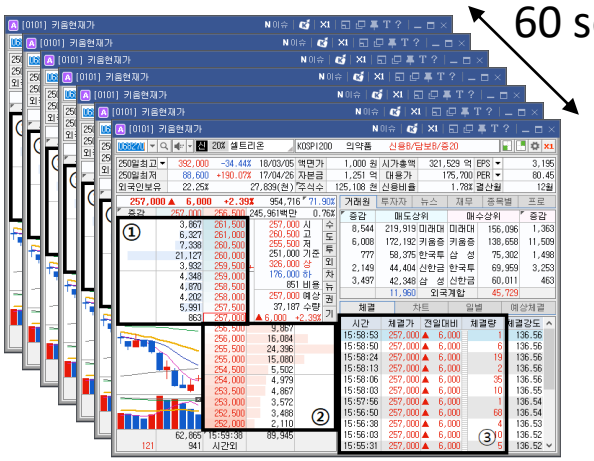
$$V^{\pi^{off}}(s_{t+1}) = Q^{\pi^{off}}(s_{t+1}, a) \times \pi^{off}(s_{t+1}, a)$$

$$V_B^{\pi^*}(s_{t+1}) = Q_B^{\pi^*}(s_{t+1}, a) \times Q_A^{\pi^*}(s_{t+1}, a)$$

Deep Q Network

Value function v_π과 q_π 들은 경험으로부터 학습해야 하는데, 환경에서 정의 가능한 State Set들이 너무 많은 경우는 어떻게 할까?

문제는 (State, Action) Pair 테이블 크기는 리소스 제약이 있고, 의미 있는 것은 극히 일부



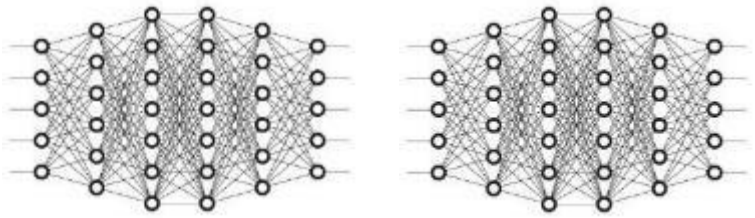
2x10x60x2 + 11x60

VS

0.0	0.0	-0.25	0.0	0.0
0.0		0.0	0.0	0.0
-0.25	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.25	0.0
-0.0	0.0	0.0	0.0	0.0

5x5

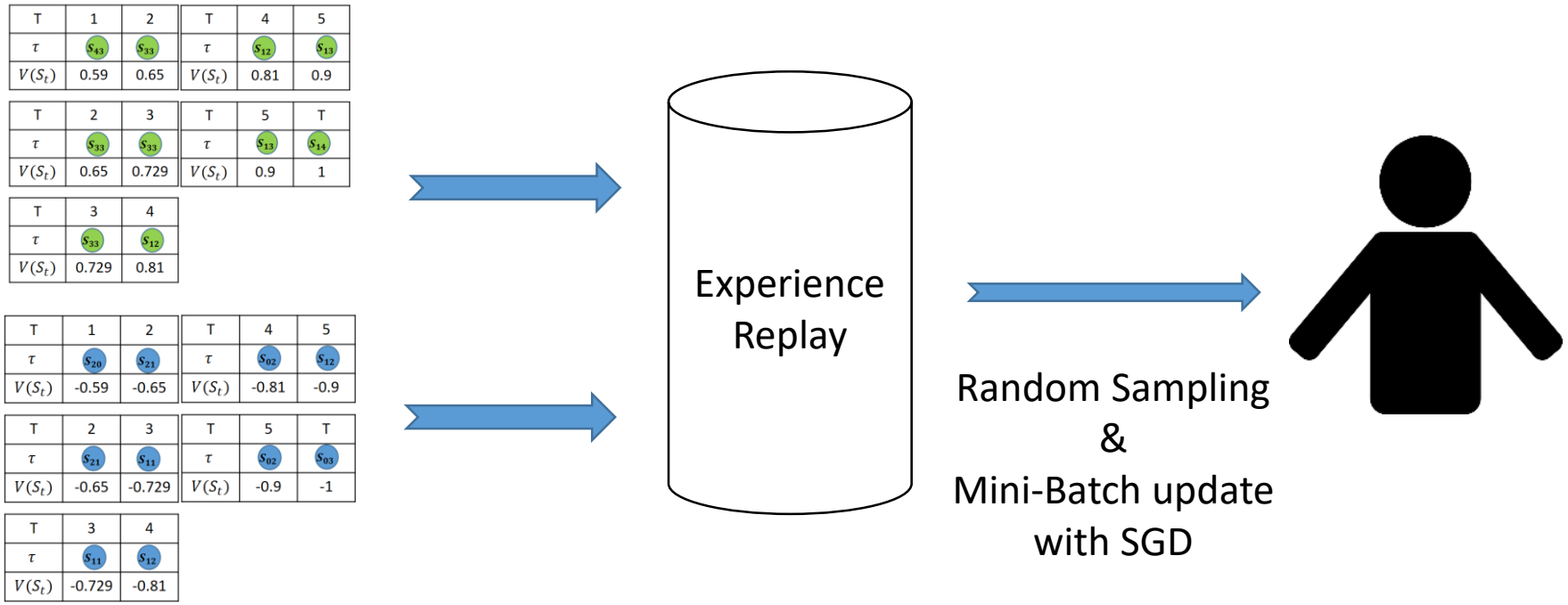
v_π q_π



Function Approximator: Linear Regression, Decision Tree, Neural Network 가능

Deep Q Network

Experience Replay에 TD sample을 넣어 샘플링 후 미니배치 학습해서 성능 향상



RL is beautiful

여태까지 우리는 강화학습을 실전에 적용할 준비를 마쳤고 이제 이런 가정을 한다.

1) 에이전트는 정확하게 환경의 불확실성을 예측을 할 수 있다.
2) 우리는 에이전트가 학습에 필요한 충분한 리소스를 가지고 있다.
3) 마지막으로 Markov property를 만족한다.

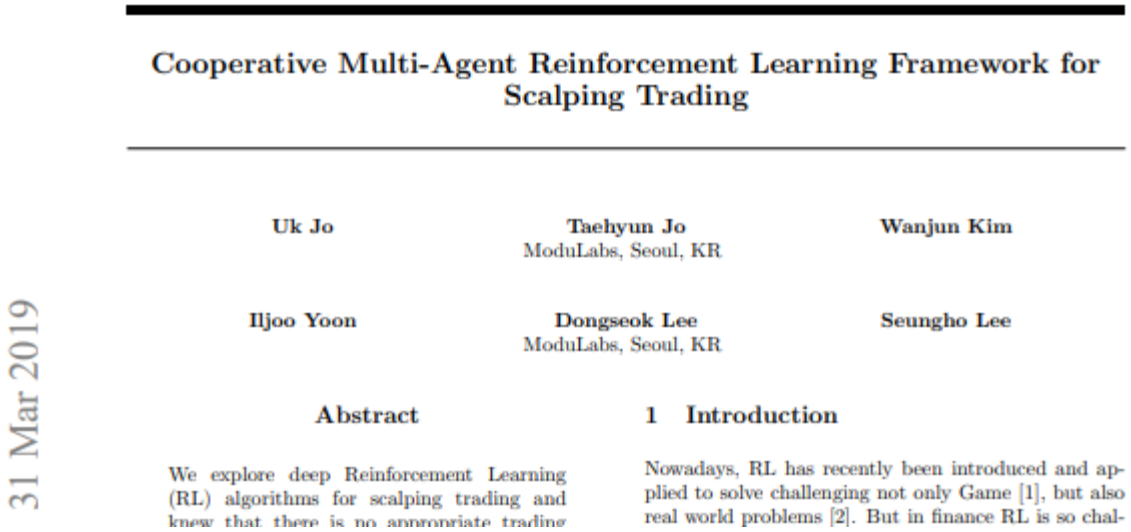
많은 강화학습 문제가 벨만 최적 방정식으로 깔끔하게 풀릴 수 있다. 강화학습의 특성상 우리는 좀 더 에이전트가 방문하는 상태와 액션에 집중함으로써 최적 정책을 발견 가능하다. 이게 다른 MDP를 문제를 푸는 방법과 차별화가 되는 포인트!!

2. Cooperative Multi-Agent Reinforcement Learning Framework for Scalping Trading

Introduction to Paper

※ **Cooperative Multi-Agent Reinforcement Learning Framework for Scalping Trading**

- 해당 논문은 모두의 연구소 팀 프로젝트로 시작
- 2019년 3월 Arxiv 등재



31 Mar 2019

Cooperative Multi-Agent Reinforcement Learning Framework for Scalping Trading

Uk Jo **Taehyun Jo** **Wanjun Kim**
ModuLabs, Seoul, KR

Iljoo Yoon **Dongseok Lee** **Seungho Lee**
ModuLabs, Seoul, KR

Abstract

We explore deep Reinforcement Learning (RL) algorithms for scalping trading and knew that there is no appropriate trading

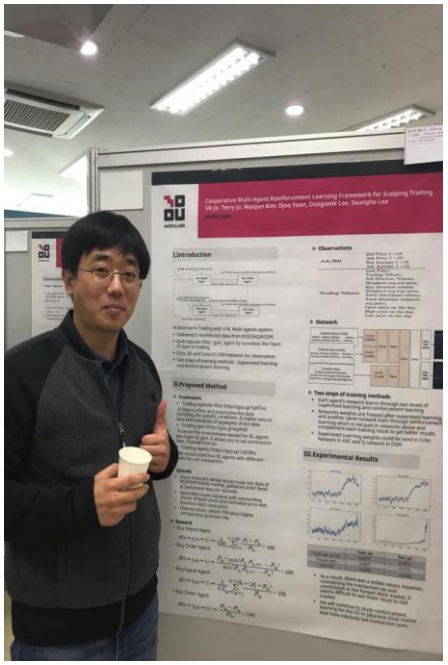
1 Introduction

Nowadays, RL has recently been introduced and applied to solve challenging not only Game [1], but also real world problems [2]. But in finance RL is so chal-

Motivation to Ideation

❖ 모두의 연구소 네트워킹 시간 3시간 동안 전문 트레이더 분들과 3시간을 시간가는 줄 모르고 대화를 나누는 게 발단

“강화학습은 정말 게임에만 맞을까? 주식에 적용을 해보자!!”



Motivation to Ideation

EMH (Effective Market Hypothesis)
+
Massively POMDP
+
Noise Market
+
Irrational Crowd Psychology

Is it impossible?

Motivation to Ideation

장기 투자가 아닌 "초단타 매매" 대상으로 데이터와 환경만 있다면 가능하지 않을까 부터 시작

변동성 심한 종목들 상한가 따라잡기하는 개미떼들이 모여 탐욕으로 얼룩진 상황

=> 전날 상한가를 기록했던 종목들 대상으로 다음날 아침 9시부터 9시 15분 처음 장이 열리고 15분간이 승부 타임이라 가장 변동이 심한 구간 (전문 트레이더분들의 의견 수렴)

Data Preparation

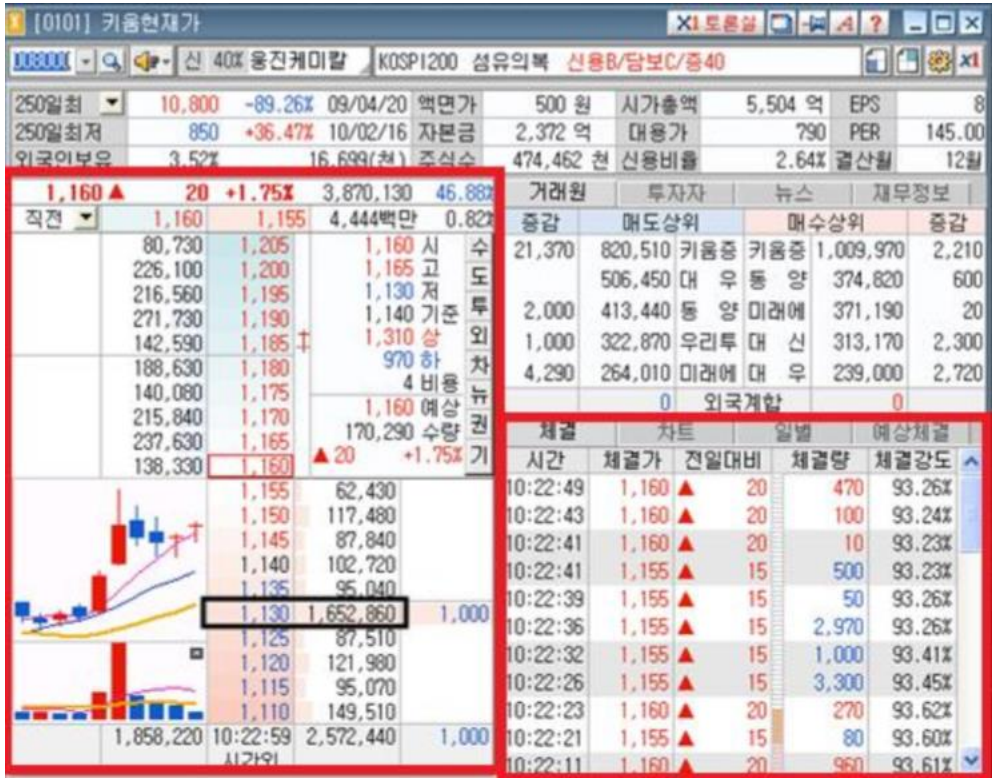
- 모든 증권사에서 틱 단위 데이터는 실시간 외에 수집 방법이 존재하지 않으므로 증권사에서 제공하는 API를 이용하여 데이터 자체 수집

- 키움과 한국 대한투자증권 2곳 대상으로 틱 단위의 데이터를 2018년 3월부터 2018년 9월 데이터를 전체 수집

출처: https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/
Cheng, Chi-Tung, et al. "Application of a deep learning algorithm for detection and visualization of hip fractures on plain pelvic radiographs." European radiology (2019): 1-9.

Data Preparation

❖ 매수/매도 호가 체결량 정보와 증권사에서 제공하는 체결 외 정보 등을 이용하여 상한가 따라잡기를 시도하는 스켈핑 하는 에이전트를 만들어보자



Data Preparation

복잡하게 보이지만 실제로 중요한 정보는 아래로 요약

① ② 매수/매도호가 (매도기준 설명)

- "셀트리온"이라는 주식을 "팔려고" 하는 주문의 집합

예)257,000원에 셀트리온 팔려고 하는 주식수는 863주
257,500원에 셀트리온 팔려고 하는 주식수는 5,991주
261,500원에 셀트리온 팔려고 하는 주식수는 3,867주
=> 실시간 셀트리온 257,000원에 매수 주문(최우선매도호가)

호가정보는 사람들이 얼마에 주문을 내느냐, 혹은 기존 주문을 취소/수정하느냐에 따라서 **실시간으로 변함**

- **즉, 호가창의 변화가 정보가 될 수 있음 (매도 호가에 물량이빠르게 쌓인다➔ 주식이 상승하기 어렵다? 팔려는 사람 多)**

③ 체결량(체결시간, 체결가격, 체결량)

- 거래가 이루어진 정보

예)15:18:52(오후 3시 18분 52초)에 257,500원 43주 거래
15:18:52 (오후 3시 18분 52초)에 257,500원 45주 거래

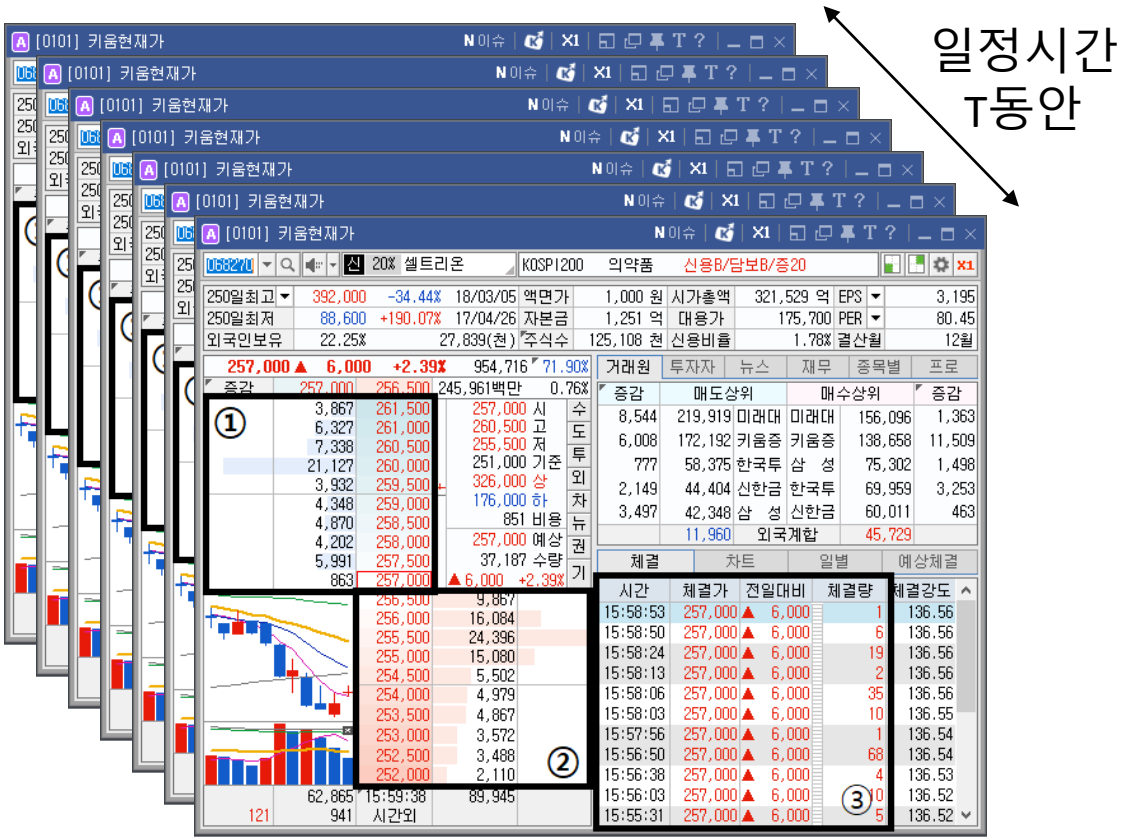
체결량이 빨간색/파란색으로 표시된 것

⇒ 거래가 체결 될때 매도호가에서 체결되었는가 혹은 매수호가에서 되었느냐에 따라

즉, 빨간색은 누군가 257,500원에 셀트리온을 사겠다고 즉시 주문을 낸 것 (최우선 매도호가에 주문)
파란색은 누군가 257,000원에 셀트리온을 팔겠다고 즉시 주문을 낸 것 (최우선 매수호가에 주문)

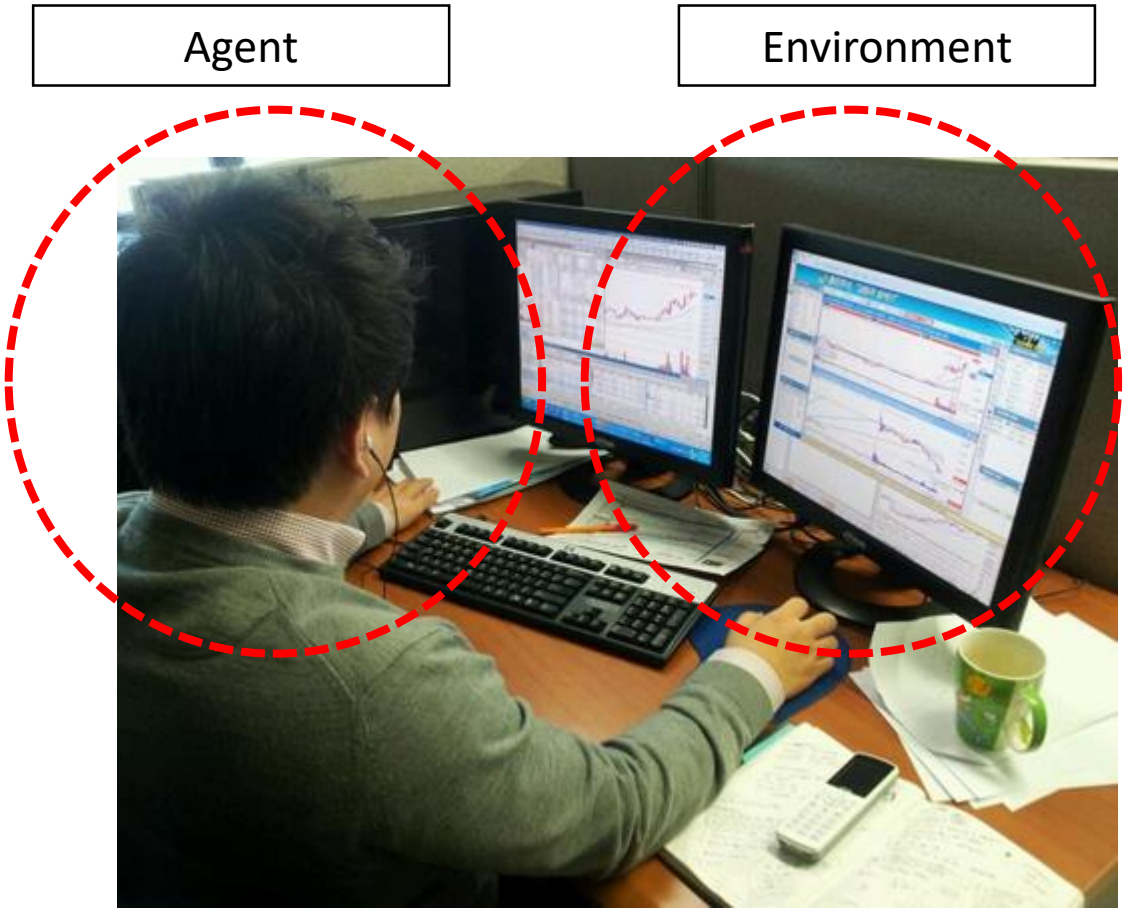
Data Preparation

복잡하게 보이지만 실제로 중요한 정보는 아래로 요약



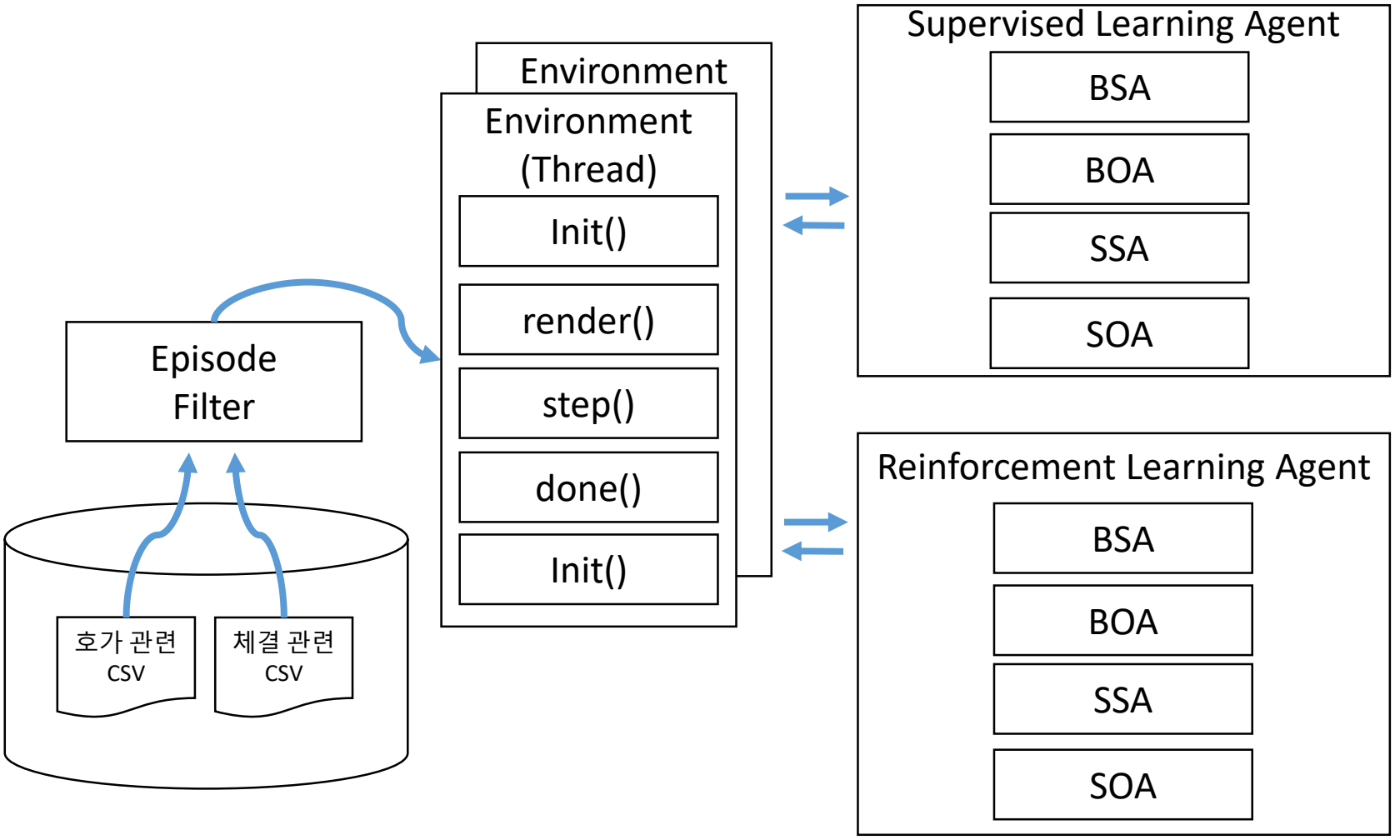
Trading Environment and Agent

데이 트레이더 특히 상한가 따라잡기등을 하는 인간 트레이더들을 에이전트로 생각하고 환경은 이제 데이트레이더들이 주로 보는 정보인 증권사 정보를 가지고

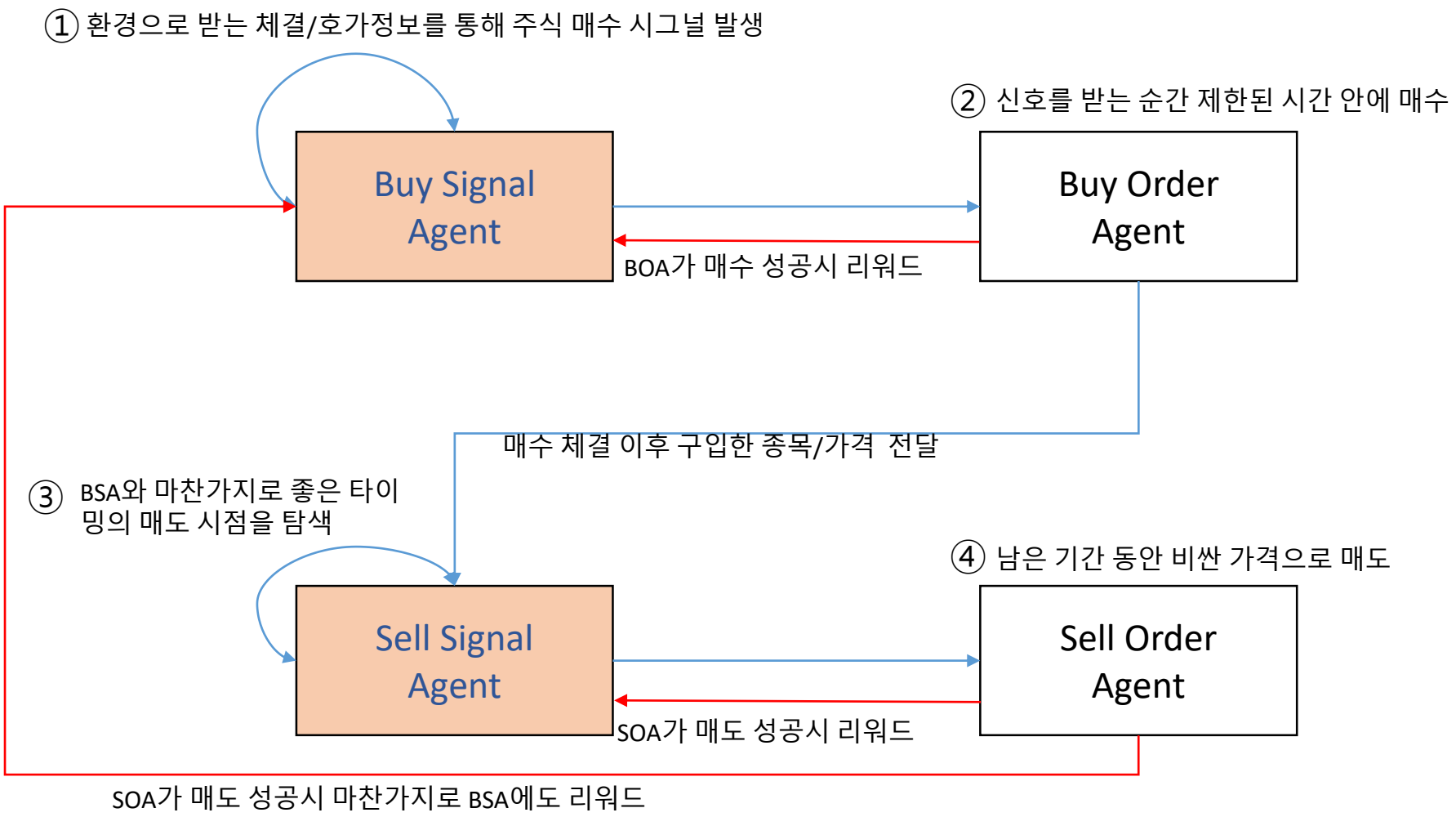


Trading Environment and Agent

우선 강화학습 환경과 에이전트를 One Screen view로 보면 아래와 같다.

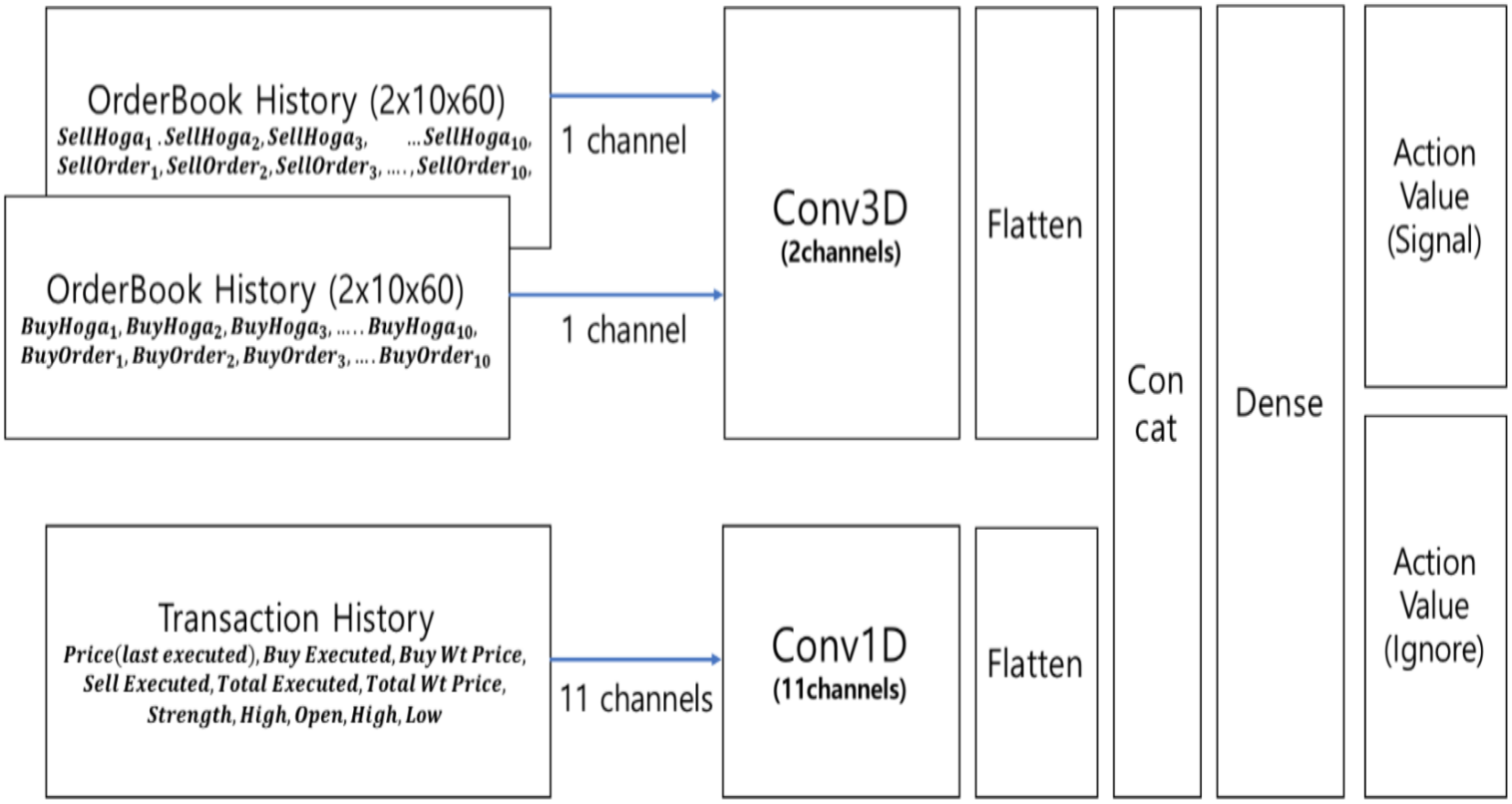


Trading Environment and Agent



Trading Environment and Agent

4개의 에이전트는 State를 받아서 별도의 Q 값을 나오게 되어있는데 아래와 같은 아키텍쳐를 공유한다.



Buy Signal Agent

60초간 사이 2% 의 가격 상승에 대해서 예측이 된다면 시그널을 발생하는 에이전트

S_t : 60초간의 시계열 데이터 (1차원 : 해당 종목 가격정보, 체결 관련 정보, 2차원 : 매수/매도호가 정보)

A_t : 매수 시그널 발생 여부

R_t : 공통으로 4개의 에이전트가 협업하여 발생한 수익에 대해서는 수익률에 비례하는 리워드 보상 받음.

추가적으로, 60초안의 2% 이상의 상승 시그널을 잡아내면 +2% ~ -2%를 1 ~ -1로 변환하여 리워드를 추가 부여하며, 뒤에 따라오는 BOA/SOA가 매수/매도를 제한 시간 안에 성공하면 추가 리워드 보상 받음

Buy Order Agent

이전 BSA에서 받은 시그널 받고 작동하게 되며, 제한 시간 안에 매수하여 가격

S_t : 60초간의 시계열 데이터 (1차원 : 해당 종목 가격정보, 체결 관련 정보, 2차원 : 매수/매도호가 정보)

A_t : 매수 여부

R_t : 공통으로 4개의 에이전트가 협업하여 발생한 수익에 대해서는 수익률에 비례하는 리워드 보상 받으며, 추가적으로 Action을 하여 매수가 성공했으면 –Action 자체가 추가 리워드고 SOA가 성공하면 마찬가지로 추가 리워드를 받게 됨.

Sell Signal Agent

매수 이후 총 제한 시간 동안 해당 종목에 대한 매도 시그널 발생하는 에이전트

S_t : 60초간의 시계열 데이터 (1차원 : 해당 종목 가격정보, 체결 관련 정보, 2차원 : 매수/매도호가 정보)

A_t : 매도 시그널 발생 여부

R_t : 공통으로 4개의 에이전트가 협업하여 발생한 수익에 대해서는 수익률에 비례하는 리워드 보상 받으며, 추가적으로 Action을 하여 매수가 성공했으면 추가 리워드, SOA가 성공하면 마찬가지로 추가 리워드 받게 됨.

Sell Order Agent

제한 시간 안에 가장 좋은 가격에 주식을 매도하는 에이전트

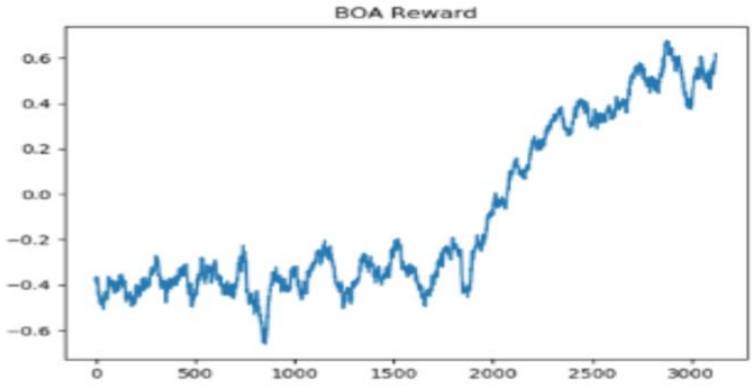
S_t : 60초간의 시계열 데이터 (1차원 : 해당 종목 가격정보, 체결 관련 정보, 2차원 : 매수/매도호가 정보)

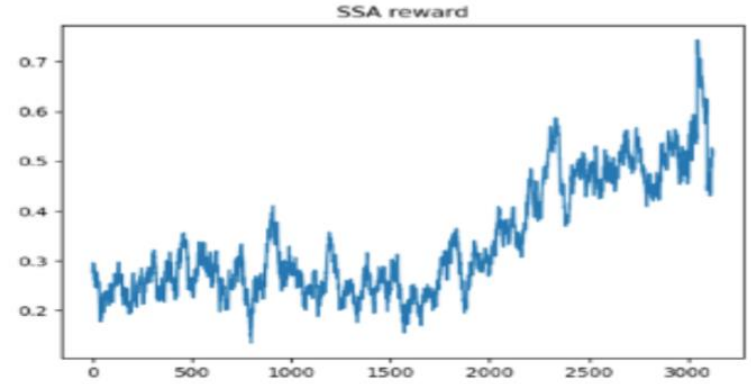
A_t : 매도 여부

R_t : 공통으로 4개의 에이전트가 협업하여 발생한 수익에 대해서는 수익률에 비례하는 리워드 보상

Results

각 에이전트의 리워드 그래프는 다음과 같다.





Results

해석 : 한국 기준의 세금(Worst case)를 고려해도 0.39%라는 수익율과 변동성이 상당히 있는 시황에서도 Max Draw Down(MDD)가 -4.36%를 기록했다는 점은 고무적.

	Train set	Test set
Profit per ep (%)	0.8070	0.3914
MDD (%)	-4.2640	-4.3656

Profit per ep(%) : 500개의 에피소드의 평균 수익율 (수수료 및 세금 = 0.3% 한국기준)
MDD(%) : Max Draw Down, 500개 에피소드 중 가장 큰 손실

출처: https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/
Cheng, Chi-Tung, et al. "Application of a deep learning algorithm for detection and visualization of hip fractures on plain pelvic radiographs." European radiology (2019): 1-9.

3. Conclusion

Trading Gym & Agent

Trading Gym for Quantitative trading

Intro

This is a trading gym for any agent to try trade for short term trading. We have collected enermous data for short term trading. We have been gathering for every Korean equities order and quote data every tick moment and also reflected data to our trading gym. **Trading Gym is a toolkit for developing and comparing reinforcement learning trading algorithms.**

Basics

There are two basic concepts in reinforcement learning: the environment (namely, the outside world) and the agent (namely, the algorithm you are writing). The agent sends actions to the environment, and the environment replies with observations and rewards.

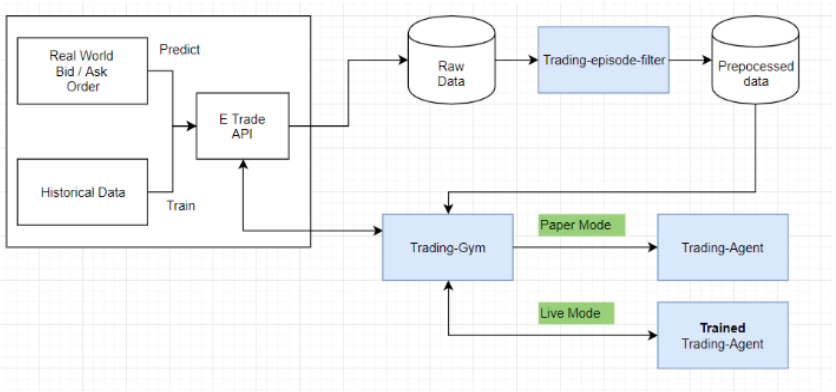
Trading Gym recreates market states based on trade orderbook and execution data. In this environment, you can make reinforcement learning agents learn how to trade.

The core gym interface is same as OpenAI Gym Env, which is the unified environment interface. There is no interface for agents; that part is left to you. The following are the `Env` methods you should know:

- reset(self): Reset the environment's state. Returns observation.
- step(self, action): Take an action in the environment by one timestep. Returns observation, reward, done, info.
- render(self, mode='human', close=False): Render one frame of the environment. Display gym's status based on a user configurations.

Architecture

It's is simple architecture that you motivate follow and run this repo easily.



https://github.com/6-Billionaires/trading-gym

Trading Agents for Quantitative trading

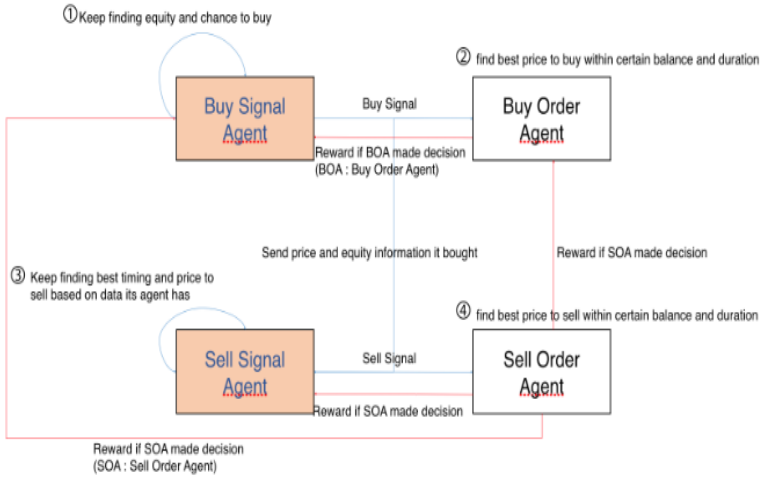
Intro

There are trading agents that learns to buy and sell stocks by connecting with trading gym. It consists of 4 agents having each own's role.

- Buy Signal Agent (BSA), which generates a buy signal.
- Buy Order Agent (BOA), which receives a buy signal and makes a purchase at the lowest price within a certain period of time
- Sell Signal Agent (SSA) which generates sell signal with receiving buyer price and quantity, market information
- Sell Order Agent (SOA) which receives sell signal and sells at highest price within a certain time.

Each Agents have an independent reward, but some parts of the rewards are shared with other Agents and eventually move to a single goal of 'Trading Return'.

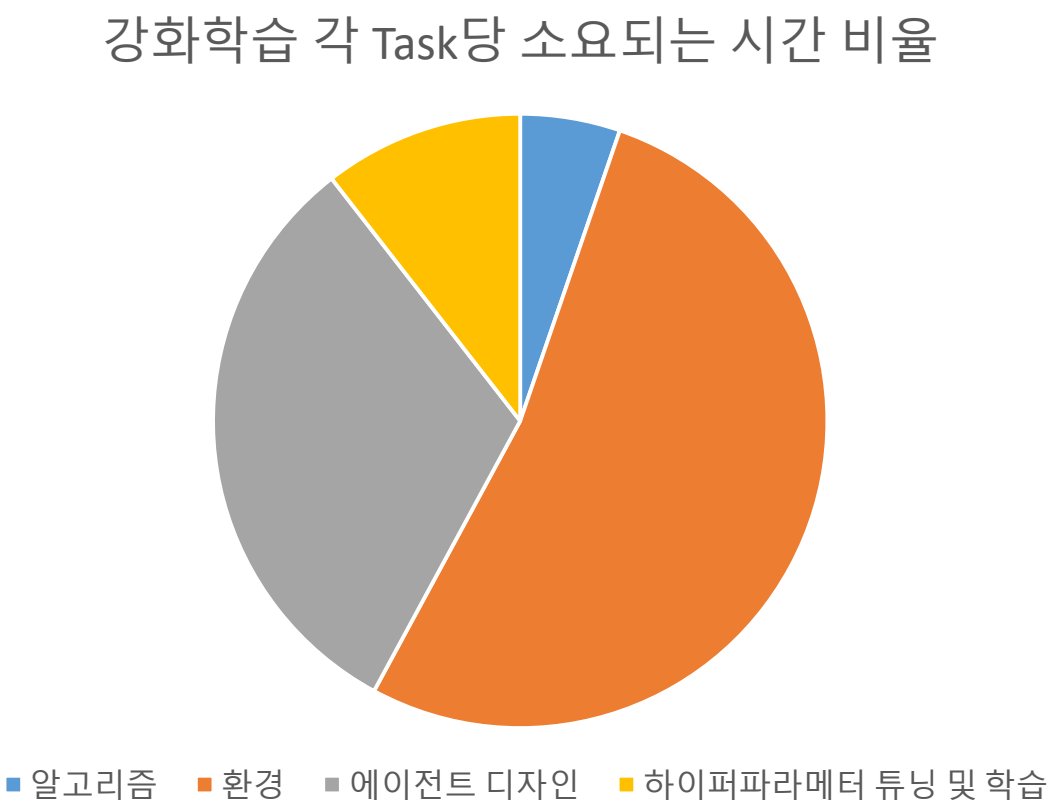
Architecture



https://github.com/6-Billionaires/trading-agent

Reinforcement Learning

에이전트를 위한 학습 환경 구현이 가장 큰 이슈, 그 다음이 에이전트 디자인(상태, 액션, 리워드)를 정하고 알고리즘을 고르게 되고 학습을 진행하는 것. 에이전트처럼 연구자도 똑같이 많은 좌절을 계속 맛보게 됨.



벌쳐 한 마리와 저글링 12마리와의 전투

500회 학습 후

저글링을 잘 못 피함
(**행동, 지형 모두 미학습**)

1000회 학습 후

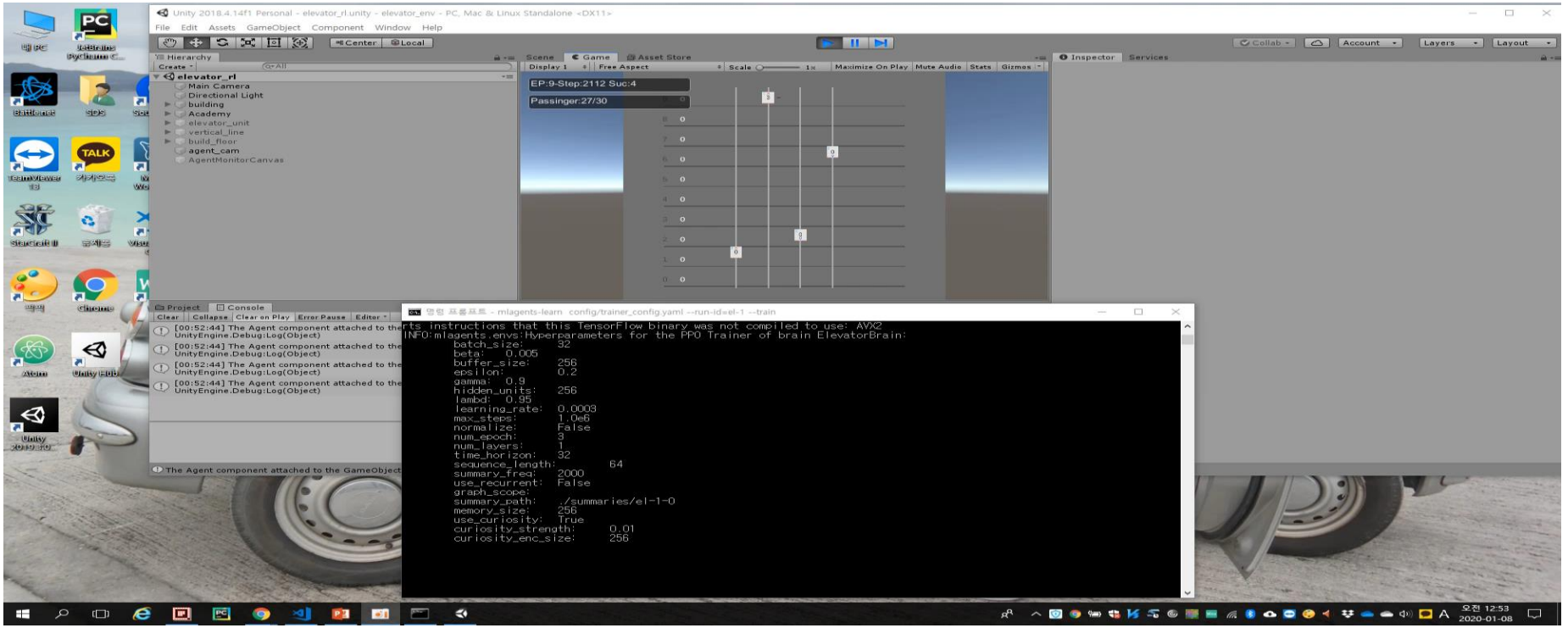
저글링 피하면서 공격 성공
지형적으로 구석에 몰려 실패
(**행동 학습, 지형 미학습**)

5000회 학습 후

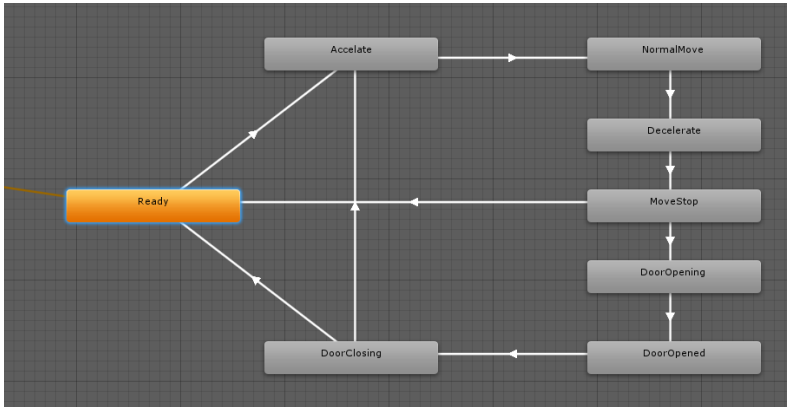
저글링을 피하면서 공격 성공
구석에 몰리지도 않고 미션 성공
(**행동, 지형 모두 학습**)

SDS SAIDA RL Framework (open source): https://github.com/TeamSAIDA/SAIDA_RL

Elevator RL



상태	설명
Ready,	문닫고 멈춰있는 상태
NormalMove,	위아래 어느쪽이든 정상적인 속도로 이동하는 상태
Decelerate,	다음층에 멈추기 위한 감속상태
MoveStop,	이동하는 중에 각층에 멈춤상태
DoorOpening,	문열는중
DoorOpened,	문열린 상태(이 상태에서 승객 승하차가 이루어짐)
DoorClosing,	문닫히는 동안.
Accelate,	이동에 대한 가속상태



Q & A