

인공지능 슈퍼마리오의 거의 모든 것

정원석



강화학습의 이론과 실습을 병행

인공지능 슈퍼마리오를 스스로 만들기 위한 거의 모든 것

소개



정원석 연구원

City University of New York -Baruch College
Data Science 전공

ConnexionAI 연구원

Freelancer Data Scientist

모두의연구소 강화학습 연구원

Github:
<https://github.com/wonseokjung>

Facebook:
<https://www.facebook.com/ws.jung.798>

Blog:
<https://wonseokjung.github.io/>

순서

1. Dynamic Programming
 - a. Policy iteration
 - b. Value iteration
2. Monte Carlo method
3. Temporal-Difference Learning
 - a. Sarsa
 - b. Q-learning
4. 딥러닝 프레임워크 케라스 소개 및 슈퍼마리오 환경 구축
5. DQN을 이용한 인공지능 슈퍼마리오 만들기

실습

1. Dynamic Programming

- a. Policy iteration
- b. Value iteration

2. Monte Carlo method

3. Temporal-Difference Learning

- a. Sarsa
- b. Q-learning

4. 슈퍼마리오 환경 구축 및 딥러닝 프레임워크 케라스 소개

5. DQN을 이용한 인공지능 슈퍼마리오 만들기

Model-based

Model-free

Deeplearning
+
RL

실습

1. Dynamic Programming

- a. Policy iteration
- b. Value iteration

2. Monte Carlo method

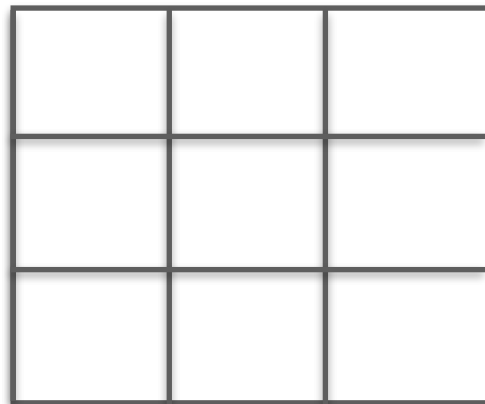
3. Temporal-Difference Learning

- a. Sarsa
- b. Q-learning

4. 슈퍼마리오 환경 구축 및 딥러닝 프레임워크 케라스 소개

5. DQN을 이용한 인공지능 슈퍼마리오 만들기

Grid world



환경 선택의 이유

Before Deeplearning Tabular

	A	B	C	D
1	Program	Region	Period	Viewers
2	Dat Man	North	Q1	91
3	Dat Man	South	Q1	87
4	Dat Man	West	Q1	99
5	Dat Man	East	Q1	102
6	Dan Tan	South	Q1	125
7	Dan Tan	West	Q1	140
8	Dan Tan	East	Q1	107
9	Dan Tan	North	Q1	130
10	Dob The Builder	West	Q1	79
11	Dob The Builder	South	Q1	85
12	Dob The Builder	East	Q1	91
13	Dob The Builder	North	Q1	70
14	Mr Maker	East	Q1	49
15	Mr Maker	North	Q1	50
16	Mr Maker	West	Q1	51
17	Mr Maker	South	Q1	69

After Deeplearning Image, text, voice...



고전 강화학습이 중요한 이유

Classic RL + DeepLearning = !

DQN

Q-learning + CNN \rightarrow DQN

폴리지 많은 문제들



각 Level의 State가 다르기 때문에 General agent를 만들기가 어렵다.

슈퍼마리오 논문

Curiosity-driven Exploration by Self-supervised Prediction

University of California, Berkeley

[ICML 2017](#)

실습자료는

https://github.com/wonseokjung/ReinforcementLearning_byWonseok

Code + Jupyter Notebook 주석 + 설치법 모두 제공

궁금한점이 있으시면 개인적으로 연락주세요!

Markov Decision Process

Total Reward

Return of Episode

Episode 동안 Return된 Reward 의 합

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

Total Reward with Discounted

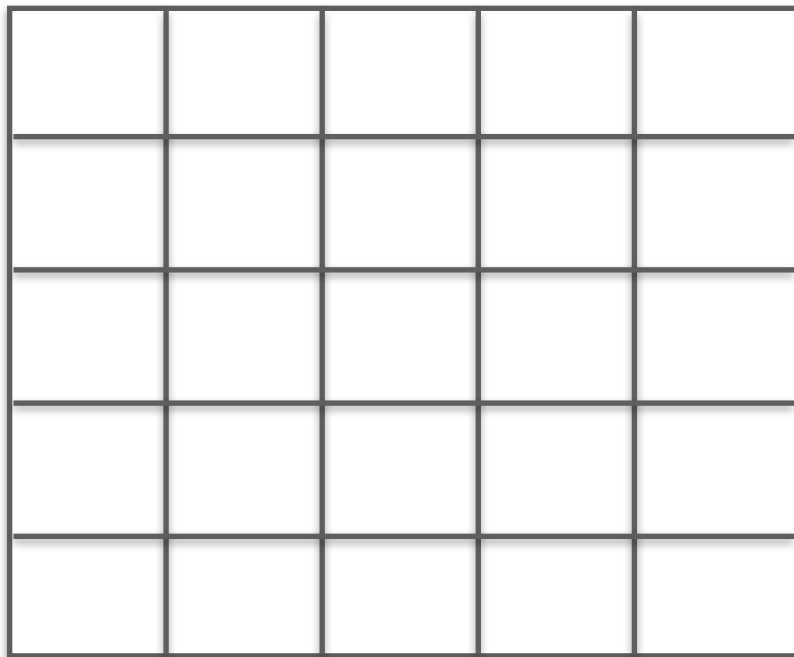
Discounted Return

Discounted factor가 적용된 Reward의 합

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

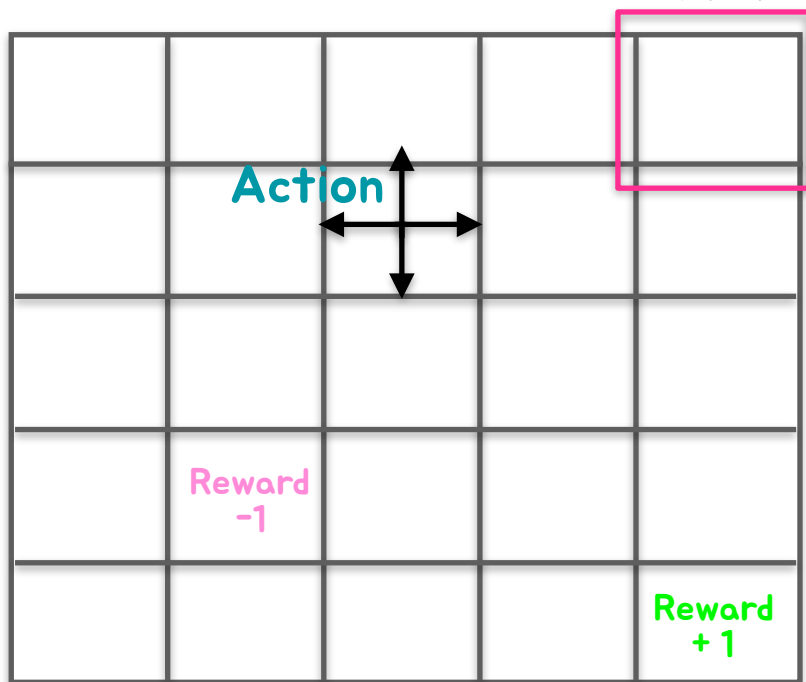
Grid World Environment

MDP 에서의 5 x 5 Grid world



Grid World Environment

MDP에서의 5 x 5 Grid world State



State : 그리드의 좌표

Action : 상, 하, 좌, 우

Reward : 함정 = -1, 목표 = 1

Transition Probability : 1

Discount factor : 0.9

State value

State-value function
(Policy를 따른 state-value function)

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S}$$

State-action value

Action Value function

(Policy를 따른 action-value function)

$$q_{\pi}(s, a) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$

A Fundamental property of value function

$$\begin{aligned}v_{\pi}(s) &\doteq \mathbb{E}_{\pi}[G_t \mid S_t = s] \\&= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_{\pi}[G_{t+1} | S_{t+1} = s'] \right] \\&= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_{\pi}(s') \right], \quad \text{for all } s \in \mathcal{S}.\end{aligned}$$

Bellman equation !

Optimal Policy를 찾자 - state

Value를 최대로 !

Optimal **state** value function

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s)$$

Optimal Policy 를 찾자 - state action

Value를 최대로 !

Optimal **state-action** value function

$$q_*(s, a) \doteq \max_{\pi} q_{\pi}(s, a)$$

Bellman equation + Optimality

Bellman optimality equation v^*

$$\begin{aligned}v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\&= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')].\end{aligned}$$

Bellman equation + Optimality

Bellman optimality equation q^*

$$\begin{aligned} q_*(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\ &= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]. \end{aligned}$$

흐름



Bellman optimal equation

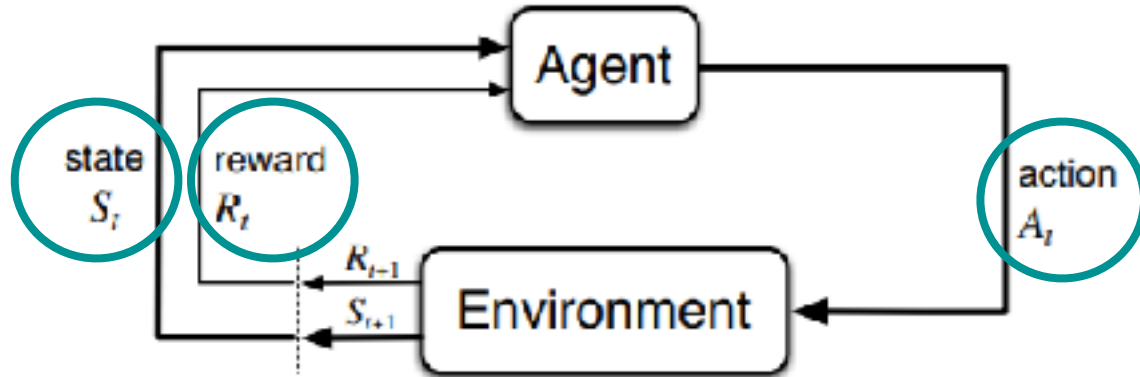
BellmanEquation

Optimal Policy

Bellman Equation + Optimal Policy

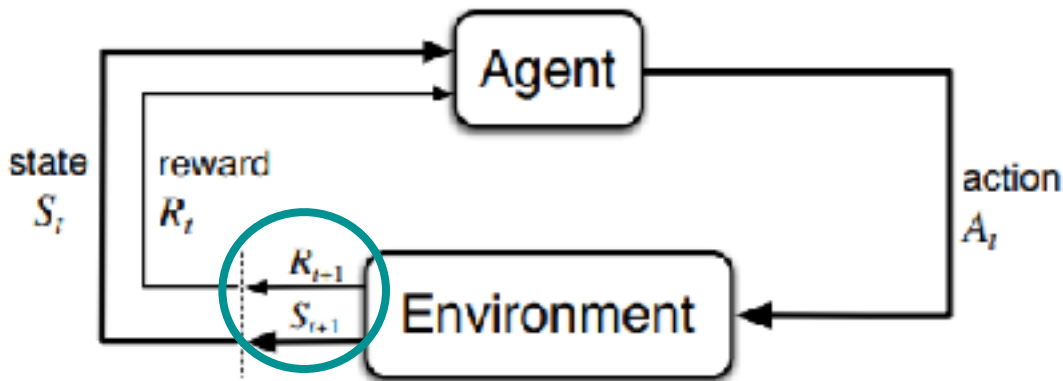
Dynamic Programming

조건



State, Reward, Action

조건



Transition Probability
또한 주어졌다.

$$p(s', r \mid s, a)$$

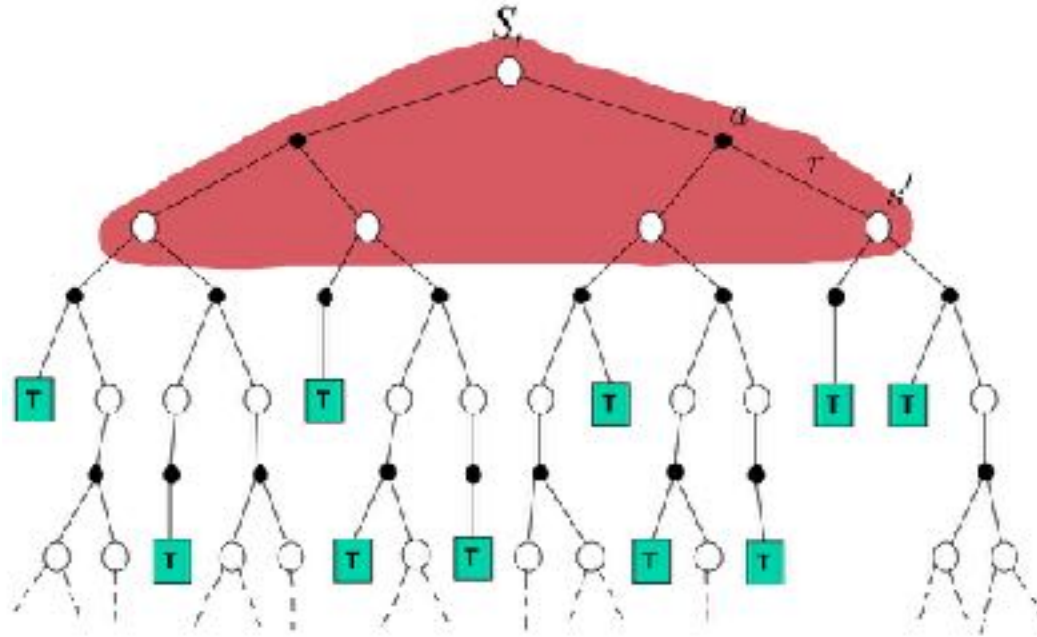
Dynamic Programming

Dynamic programming의 **Key idea!**

Value function을 사용하여 **보다 나은 Policy**
를 찾기위해 구조화 시키고 정리할수 있다.

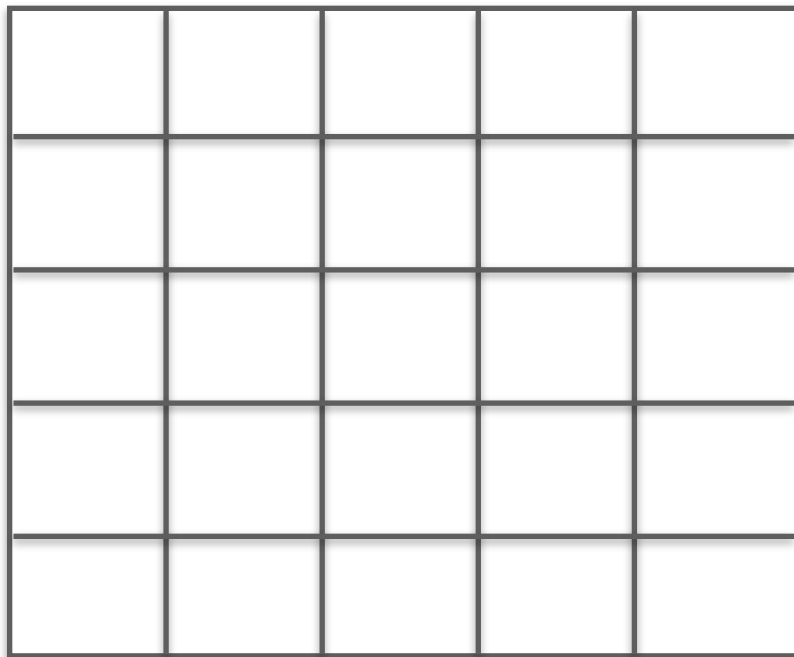
Dynamic programming

$$V(S_t) \leftarrow E_{\pi} [R_{t+1} + \gamma V(S_{t+1})]$$



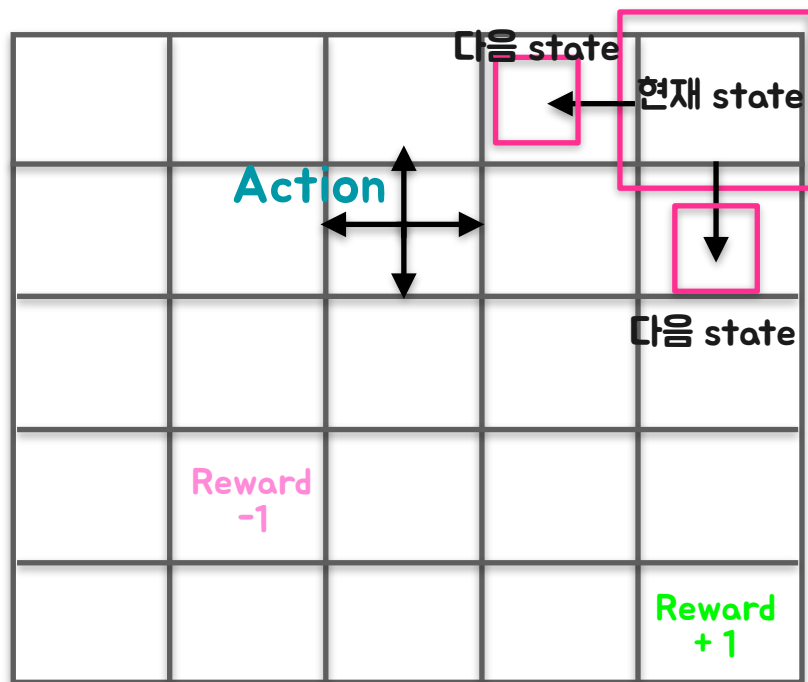
Grid World Environment

5 x 5 Grid world에서 Dynamic Programming



Grid World Environment

5 x 5 Grid world



State : 그리드의 좌표

Action : 상, 하, 좌, 우

Reward : 함정 = -1, 목표 = 1

Transition Probability : 1

Discount factor : 0.9

Update Rule

Bellman equation을 사용하여 업데이트한다.

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \text{ State!} \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_k(s')] \end{aligned}$$

두 종류의 Optimal Value functions

State Value

$$\boxed{v_*(s)} = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')]$$

Bellman optimality equations 충족

두 종류의 Optimal Value functions

Action Value

$$\boxed{q_*(s, a)} = E[R_{t+1} + \gamma \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]]$$
$$= \sum_{s', r} p(s', r \mid s, a) [r + \gamma \max_a q_*(s', a)]$$

Bellman optimality equations 충족

Dynamic Programming

두 종류의 최적 Value function

State-action Value function

$$\begin{aligned} q_*(s, a) &= E[R_{t+1} + \gamma \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]] \\ &= \sum_{s', r} p(s', r \mid s, a) [r + \gamma \max_a q_*(s', a')] \end{aligned}$$

Dynamic Programming

Policy Iteration

Value Iteration

Policy iteration

1. Policy를 따라 state-value를 계산하자!

Policy Evaluation



2. 더 좋은 Policy를 찾자!

Policy Improvement

최적의 Policy를
찾기위한 **두가지**
과정

Policy iteration- Policy Evaluation

Update Rule을 사용하여 Evaluation을 한다.

$$\begin{aligned} \boxed{v_{k+1}(s)} &\doteq \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \boxed{\pi(a|s)} \sum_{s', r} \boxed{p(s', r \mid s, a)} \left[\boxed{r} + \boxed{\gamma v_k(s')} \right] \end{aligned}$$

Value update

Policy

Transition Probability

Reward

Next State estimated value

Policy iteration- Policy Evaluation

Policy를 따라 state-value를 계산하자!

1. 모든 state를 $V(s) = 0$ 으로 초기화 시킨다.
2. 각 state를 Update Rule을 사용하여 $V(s)$ 를 업데이트 한다.

$$\sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_k(s') \right]$$

3. 업데이트하며 $V(s)$ 의 변화량이 매우 작을때 업데이트를 멈춘다.

Policy iteration- Improvement

Policy를 따라 Value function을 계산한 이유는 더 나은 Policy를 찾기위해서이다.

Greedy Policy

$$\begin{aligned}\pi'(s) &\doteq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \arg\max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_\pi(s')],\end{aligned}$$

Policy iteration- Improvement

Greedy Policy 적용

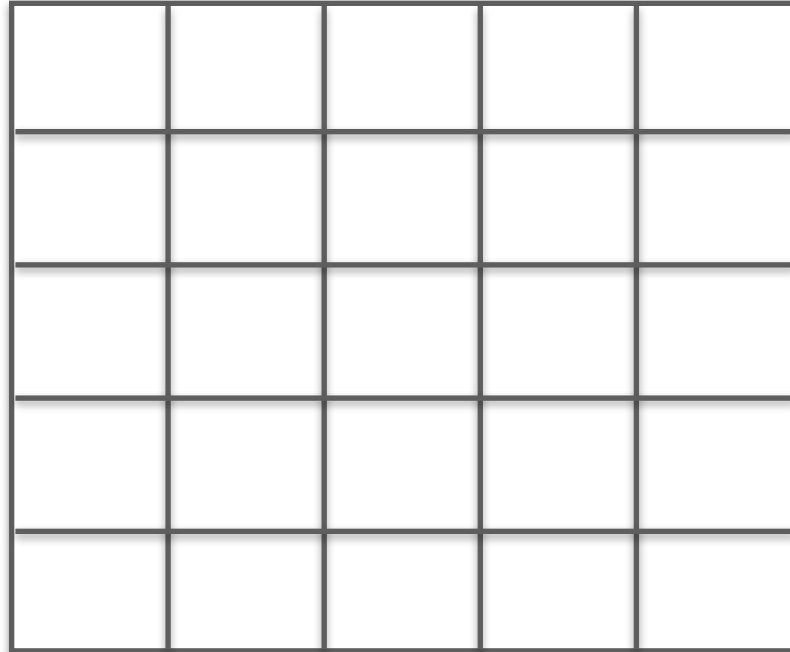
$$\begin{aligned} v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_{\pi'}(s') \right]. \end{aligned}$$

Policy iteration

Policy iteration은 Optimal policy를 찾을때까지
Policy Evaluation과 Policy Improvement를 반복한다.

Grid World Environment

5 x 5 Grid world



Grid World Environment - Policy iteration



State : 그리드의 좌표

Action : 상, 하, 좌, 우

Reward : 한칸 움직일때마다 -1

Transition Probability : 1

Discount factor : 0.9

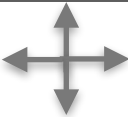
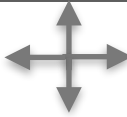
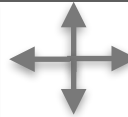
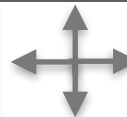
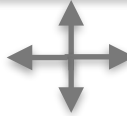
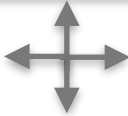
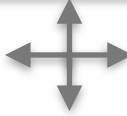
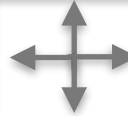
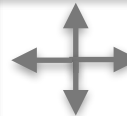
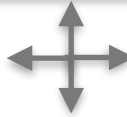
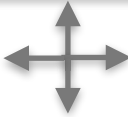
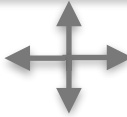
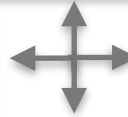
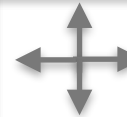
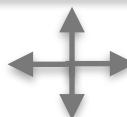

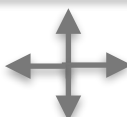
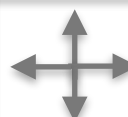
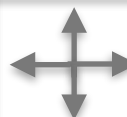
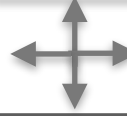
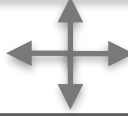
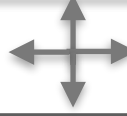
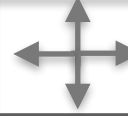
Grid World Environment - Policy iteration

k=0 일때 (초기화)

V_k

0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0

Greed Policy

Grid World Environment - Policy iteration

$k=1$

V_k

0.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	0.0

Greed Policy

	←	↕	↕	↕
↑	↕	↕	↕	↕
↕	↕	↕	↕	↕
↕	↕	↕	↕	↓
↕	↕	↕	→	

Grid World Environment - Policy iteration

$k=2$

V_k

0.0	-1.7	-2.0	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-2.0	-1.7	0.0

Greed Policy

	←	←	↕	↕
↑	↖	↕	↕	↕
↑	↕	↕	↕	↓
↕	↕	↕	↘	↓
↕	↕	→	→	

Grid World Environment - Policy iteration

$k=\text{inf}$

V_k

0.0	-90	-98	-99	-100
-90	-97	-98	-99	-99
-91	-98	-99	-98	-98
-99	-99	-98	-97	-90
-100	-99	-98	-90	0.0

Greed Policy

	←	←	←	↖↘
↑	↖↑	←	↖↘	↓
↑	↖↑	↕	↘↗	↓
↑	↖↗	→	↘↗	↓
↖↗	→	→	→	

Policy iteration

시연

Dynamic Programming

Policy Iteration

Value Iteration

Grid World Environment - Value iteration

k=수렴하면

V_k

0.0	-90	-98	-99	-100
-90	-97	-98	-99	-99
-91	-98	-99	-98	-98
-99	-99	-98	-97	-90
-100	-99	-98	-90	0.0

Greed Policy

	←	←	←	↖
↑	↖	←	↖	↓
↑	↖	↕	↘	↓
↑	↖	→	↘	↓
↖	→	→	→	

Value Iteration

반복하지 않는다!

$$v_{k+1}(s) \doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a]$$

$$= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_k(s')],$$

State, Action!

Value iteration

시연 + 코드설명

Model이 없다면??

실제로 경험을 해보며 환경과 상호작용을 해야한다.

Monte Carlo method

Monte Carlo

Monte Carlo method는 Dynamic programming처럼
모든 정보를 알고 시작 하는 것이 아닌
실제로 경험을 하며 **환경과 상호작용**을 한다.

Monte Carlo

실제로 경험을 하며 배우는 방법이 좋은 점은 environment의
정보가 없어도 실제로 경험을 하며 **optimal behavior**을 이루기
때문

Monte Carlo

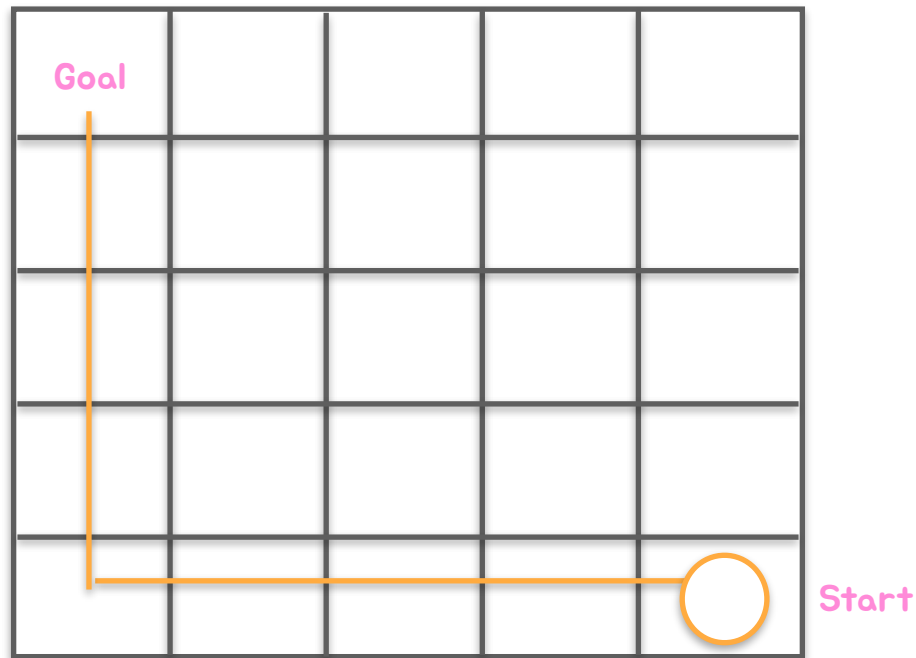
Monte Carlo는 **episode-by-episode**로 업데이트 한다

에피소드의 마지막 스테이트 **terminal state** 까지
가서 업데이트 한다.

Monte Carlo는 경험을 하며 return 된 sample을 이용하여
state-action value를 평균하여 업데이트한다.

Monte Carlo-GridWorld

끝까지 가본뒤 Update



Monte Carlo 시연

Temporal-Difference Learning

Temporal-Difference Learning

만약 강화학습을 대표할 수 있는 아이디어가 있다면 그것은 TD(temporal-difference) learning 일 것이다.

-Sutton

Temporal-Difference Learning

Monte Carlo + Dynamic programming

Monte Carlo 처럼 모델 없이 경험을 통하여 value를 측정하며
DP처럼 끝까지 가지 않아도 중간중간 value를 estimate하는것이 가능

Temporal-Difference Learning

Monte Carlo에서의 G_+ 를 알아야 업데이트 가능 :

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

현재 state에서 action을 선택하며 받을 Reward 과 다음 State에 discount factor가 적용된 state value를 estimate하며 update한다.

TD의 장점

1. Monte Carlo의 강점인 환경의 모델을 알지 못해도 사용가능
2. Dynamic programming에서 처럼 On-line 학습이다.
3. 끝까지 기다리지 않아도, 중간중간 update가 가능하기에 episode가 많이 길거나 혹은 continue한 model에서 사용하기 좋다.

TD의 아이디어

Temporal-Difference Learning이 Sarsa와 Q-learning의 바탕 아이디어가 되었다.

Temporal-Difference Learning

On-policy

Sarsa

Off-policy

Q-learning

Temporal-Difference Learning

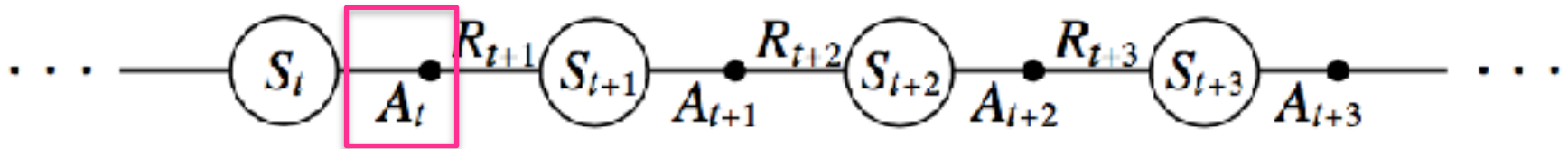
Sarsa

Q-learning

Sarsa

on-policy 방법을 사용하는 Sarsa

state-value function 대신 action-value function을 학습



Sarsa

다음 time step 에서 state와 action를 둘다 사용하여 action value를 estimate한다.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Sarsa-pseudo code

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Loop for each step of episode:

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

 until S is terminal

Sarsa-pseudo code

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Loop for each step of episode:

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

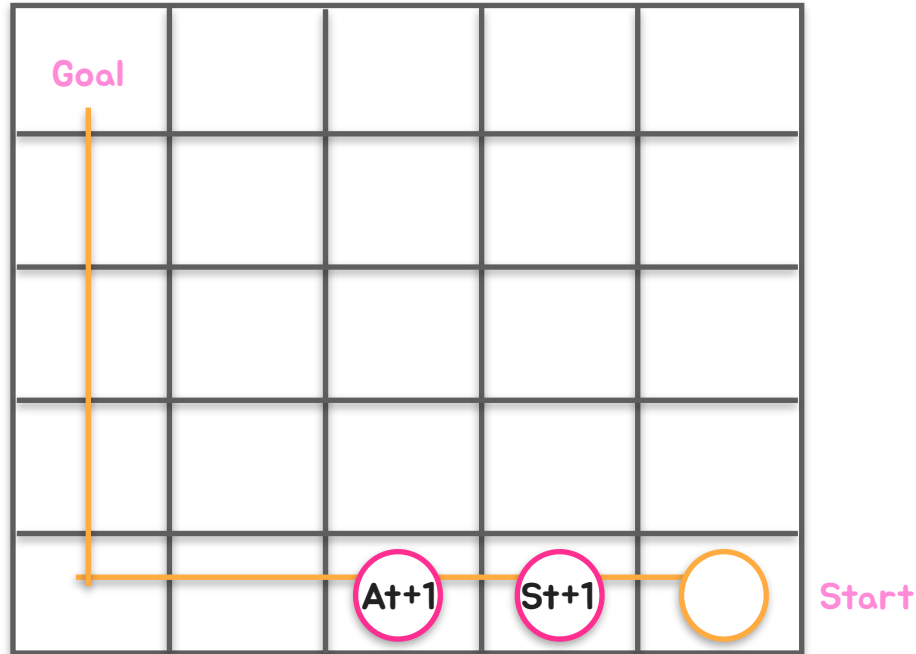
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

 until S is terminal

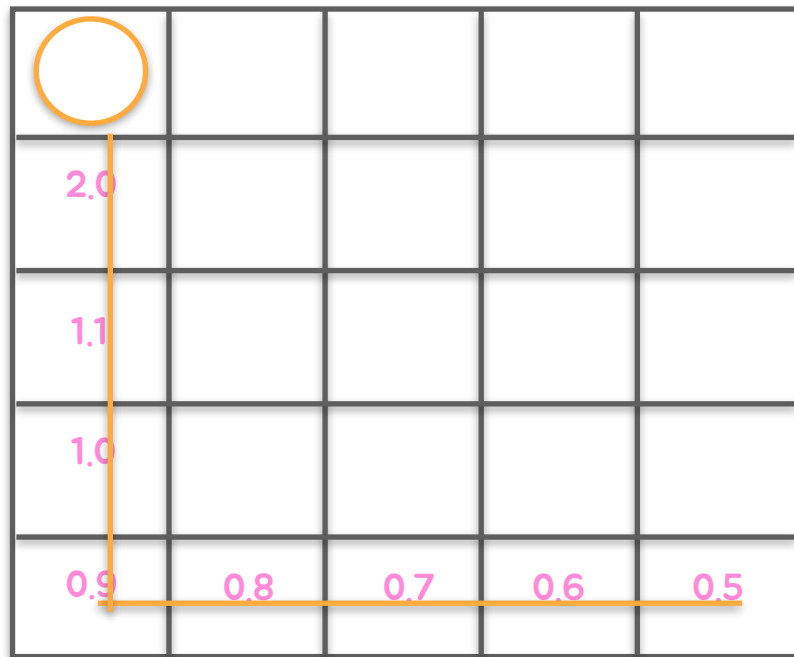
On-policy

Sarsa-gridworld



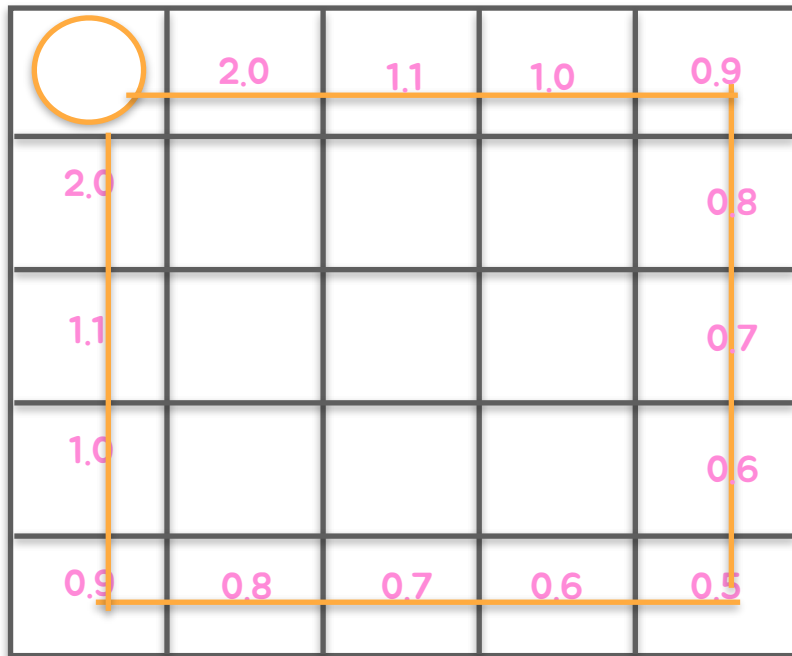
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Sarsa-gridworld



경험하지 않은 스테이트의 정보가 없다.

Sarsa-gridworld



경험을 여러번 해보며 action-value를 업데이트한다.
Policy는 On-policy

Sarsa
시연

Temporal-Difference Learning

Sarsa

Q-learning

Q-learning

Q-learning이라고 불리는 off-policy TD control인해 강화학습이 발전하는 계기가 되었다.
-(Watkins, 1989)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

exploration과 exploitation을 같이 한다.

Q-learning-pseudo code

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

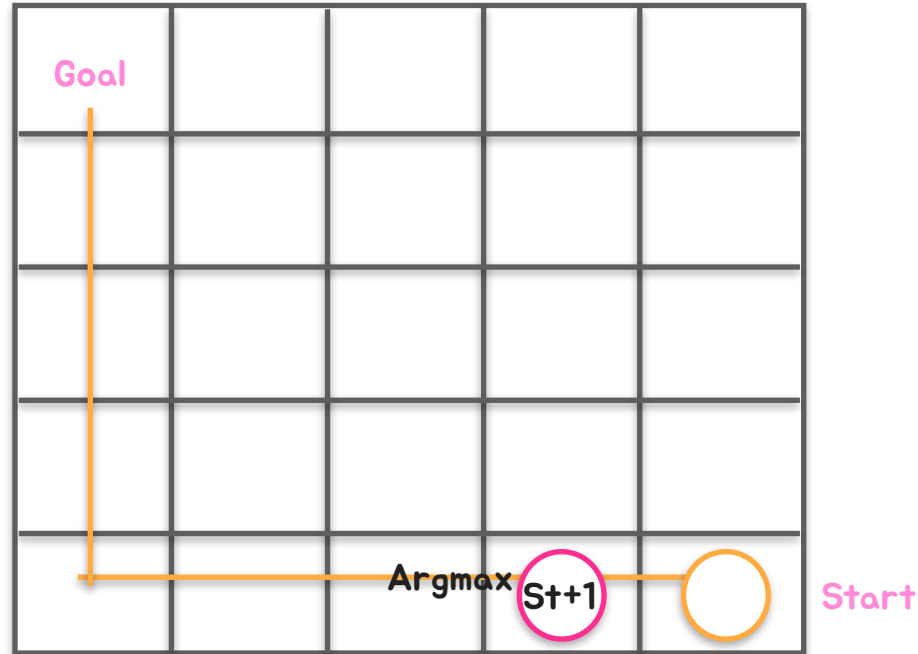
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal

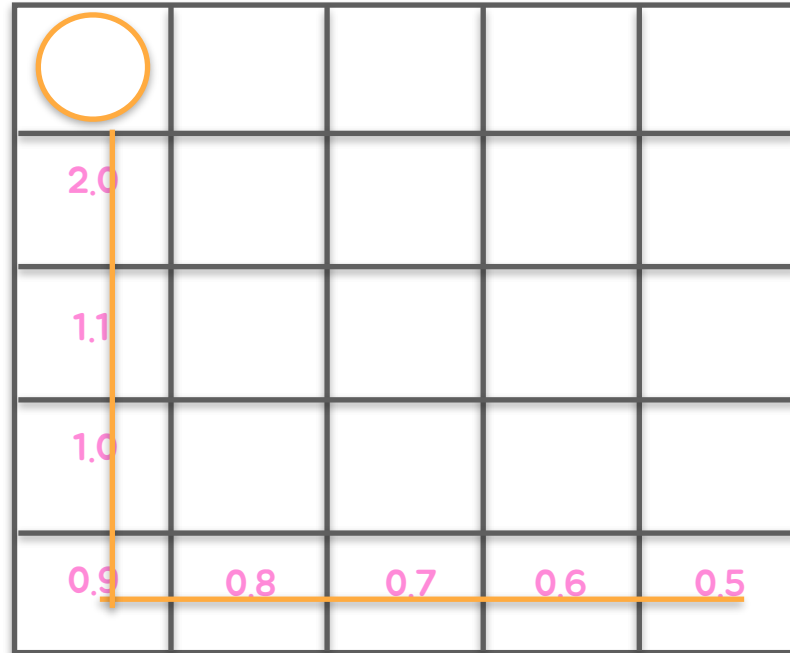
Off-policy

qlearning-gridworld



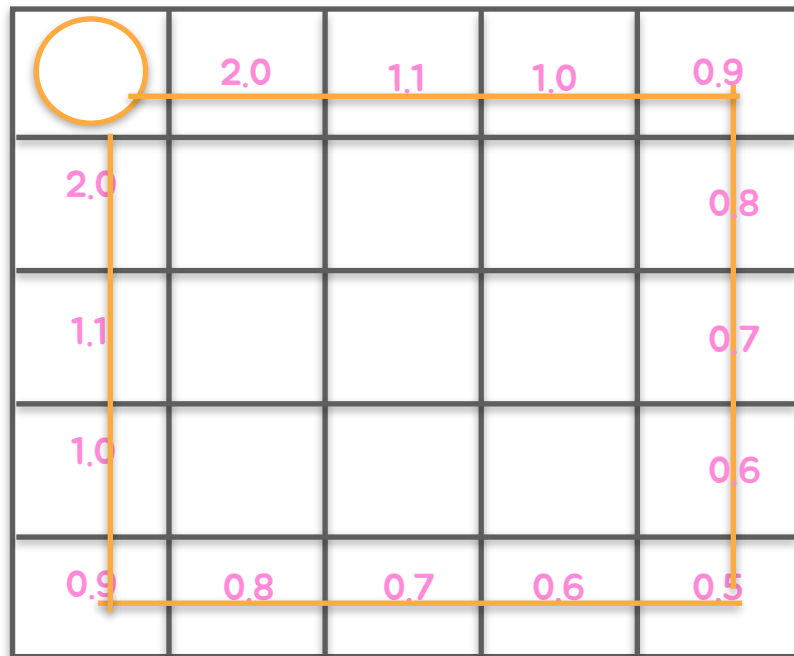
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Q-learning- gridworld



경험하지 않은 스테이트의 정보가 없다.

Q-learning- gridworld



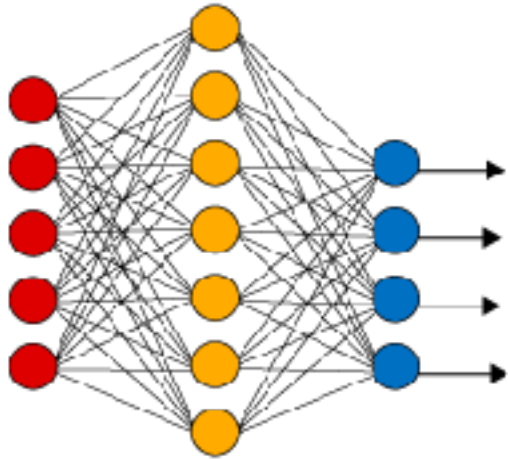
경험을 여러번 해보며 action-value를 업데이트한다.
Policy는 Off-policy

Q-learning 시연

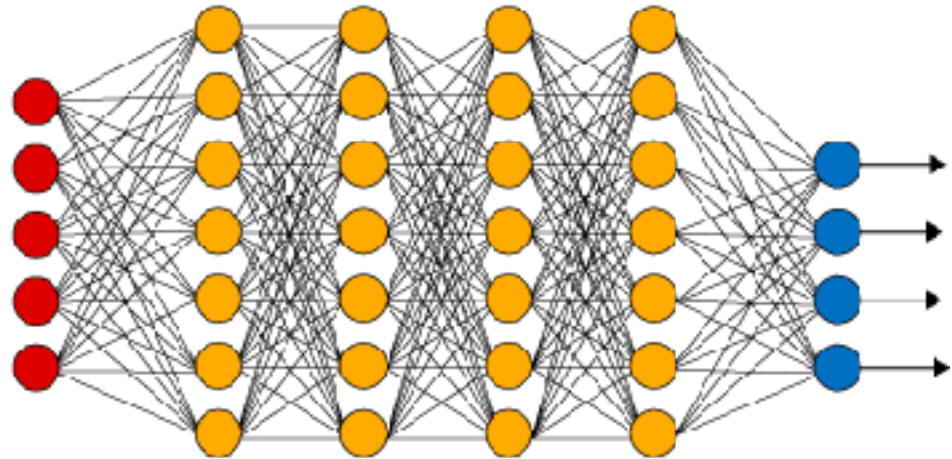
딥러닝 프레임워크 케라스 소개 슈퍼마리오 환경 구축

Deeplearning

Simple Neural Network



Deep Learning Neural Network

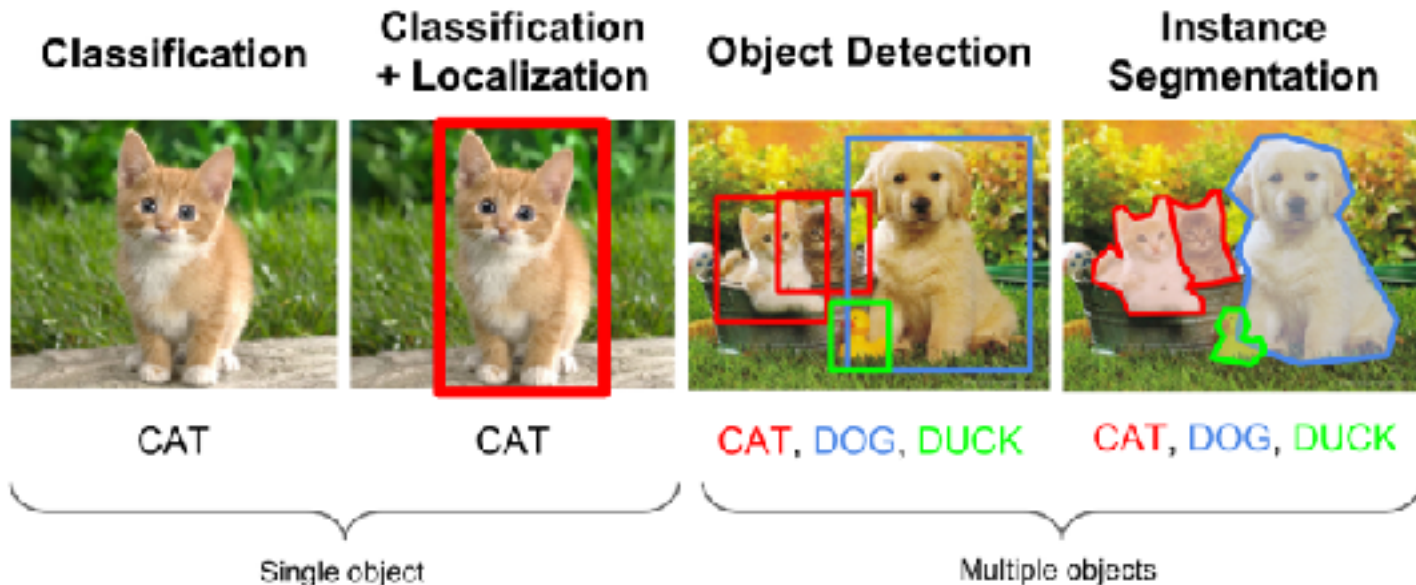


● Input Layer

● Hidden Layer

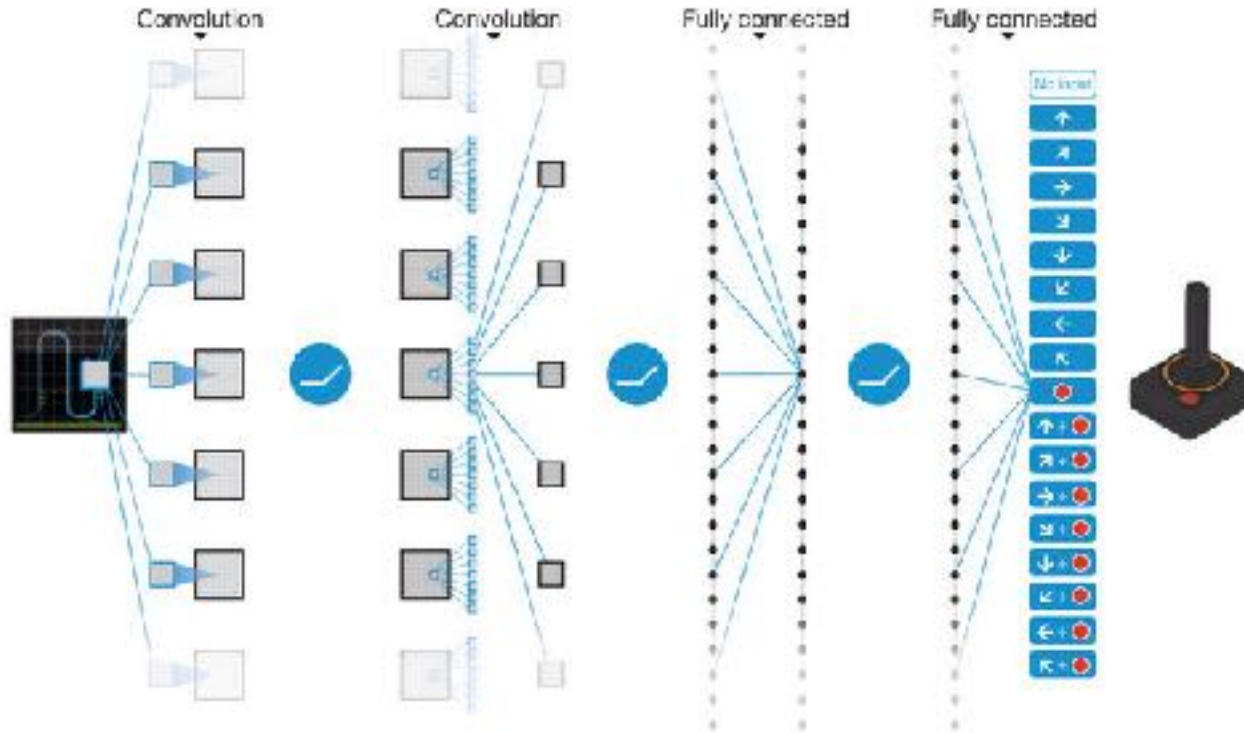
● Output Layer

Deeplearning으로 인해



사진을 input으로 받는것이 가능해짐

Deeplearning+Reinforcement Learning



Deep Reinforcement Learning

Deepmind, DQN

Deeplearning을 강화학습에 적용하여, 사람보다 플레이를 잘하는 인공지능을 만듦



<https://www.youtube.com/watch?v=V1eYniJ0Rnk>

DQN, Keras(breakout)

코드설명

1.Main

2.library

3.Function

DQN, Keras(breakout)

코드설명

1.Main

2.library

3.DQN

Main-1

1. 환경을 불러온다.

2. agent를 생성한다.

3. score, episode, global_step 를 정해준다.

4. 정해진 에피소드만큼 학습을 시작

- 목숨 다섯개가 주어진다.

- 환경의 초기값을 가져온다.

- 처음 일정구간동안 바가 action을 임의로 선택할수 있게 한다.

- 위의 환경 초기값을 이미지 전처리 해준다.

- 전처리해준 이미지로 네개의 히스토리를 만든다.

```
if __name__ == "__main__":  
    # 환경과 DQN 에이전트 설정  
    env = gym.make('BreakoutDeterministic-v4')  
    agent = DQNAgent(action_size=3)  
  
    scores, episodes, global_step = [], [], 0  
  
    for e in range(EPISODES):  
        done = False  
        dead = False  
  
        step, score, start_life = 0, 0, 5  
        observe = env.reset()  
  
        for _ in range(random.randint(1, agent.no_op_steps)):  
            observe, _, _, _ = env.step(1)  
  
        state = pre_processing(observe)  
        history = np.stack((state, state, state, state), axis=2)  
        history = np.reshape([history], (1, 84, 84, 4))
```

Main-2

* 게임이 끝날때까지 계속 돌게하는 while문을 만든다.

- render의 유무를 확인한다. (render을 유무에 따라 학습 속도가 달라진다.

- 글로벌 스텝을 하나씩 늘려준다.

- 스텝을 하나씩 늘려준다.

- 네개의 state(history)를 이용하여 action을 선택한다.

```
while not done:
    if agent.render:
        env.render()
    global_step += 1
    step += 1

    # 바로 전 4개의 상태로 행동을 선택
    action = agent.get_action(history)
    # 1: 정지, 2: 왼쪽, 3: 오른쪽
    if action == 0:
        real_action = 1
    elif action == 1:
        real_action = 2
    else:
        real_action = 3
```

Main-3

- 선택한 action으로 환경과 상호작용하며 환경에서 다음 스테이트, reward, done, info의 값을 받는다.

- 받은 다음 스테이트를 다시 전처리 해준다.

- history 에서 앞의 세개와 방금 받아온 state를 합쳐서 next_history로 선언

- q_max의 평균을 계산하기 위해서 현재의 model 로 부터 나온 Q 값의 max를 agent.avg_q_max에 더한다.

- 만약 dead인 경우 dead를 True로 바꾸고, start_life를 하나 줄여준다.

```
# 선택한 행동으로 환경에서 한 타임스텝 진행
```

```
observe, reward, done, info = env.step(real_action)
```

```
# 각 타임스텝마다 상태 전처리
```

```
next_state = pre_processing(observe)
```

```
next_state = np.reshape([next_state], (1, 84, 84, 1))
```

```
next_history = np.append(next_state, history[:, :, :, :3], axis=3)
```

```
agent.avg_q_max += np.amax(
```

```
    agent.model.predict(np.float32(history / 255.))[0])
```

```
if start_life > info['ale.lives']:
```

```
    dead = True
```

```
    start_life = info['ale.lives']
```

Main-4

- 다른 아타리 게임에서도 적용할 수 있도록 리워드의 범위를 -1~1로 한다.

- s, a, r, s' 를 리플레이 메모리에 저장한다.

- 리플레이 메모리가 시작점보다 높아지면 학습을 시작한다.

```
reward = np.clip(reward, -1., 1.)  
# 샘플 <s, a, r, s'>을 리플레이 메모리에 저장 후 학습  
agent.append_sample(history, action, reward, next_history, dead)  
  
if len(agent.memory) >= agent.train_start:  
    agent.train_model()
```

Main-4

- 일정시간마다 target model을 update한다.
- 만약에 죽으면 dead = false로 바꾸고 아니면 next history 값을 history가 받는다.
- 일정 에피소드마다 모델을 저장한다.

```
if global_step % agent.update_target_rate == 0:  
    agent.update_target_model()
```

```
if done:  
    # 각 에피소드 당 학습 정보를 기록  
    if global_step > agent.train_start:  
        stats = [score, agent.avg_q_max / float(step), step,  
                 agent.avg_loss / float(step)]  
  
        for i in range(len(stats)):  
            agent.sess.run(agent.update_ops[i], feed_dict={  
                agent.summary_placeholders[i]: float(stats[i])  
            })  
        summary_str = agent.sess.run(agent.summary_op)  
        agent.summary_writer.add_summary(summary_str, e + 1)
```

```
# 1000 에피소드마다 모델 저장  
if e % 1000 == 0:  
    agent.model.save_weights("./save_model/breakout_dqn.h5")
```

Main-4

- 일정시간마다 target model을 update한다.
- 만약에 죽으면 dead = false로 바꾸고 아니면 next history 값을 history가 받는다.
- 만약에 done이면 에피소드의 학습정보를 기록하여 출력한다.
- 일정 에피소드마다 모델을 저장한다.

```
if global_step % agent.update_target_rate == 0:  
    agent.update_target_model()
```

```
if done:  
    # 각 에피소드 당 학습 정보를 기록  
    if global_step > agent.train_start:  
        stats = [score, agent.avg_q_max / float(step), step,  
                 agent.avg_loss / float(step)]  
  
        for i in range(len(stats)):  
            agent.sess.run(agent.update_ops[i], feed_dict={  
                agent.summary_placeholders[i]: float(stats[i])  
            })  
        summary_str = agent.sess.run(agent.summary_op)  
        agent.summary_writer.add_summary(summary_str, e + 1)
```

```
# 1000 에피소드마다 모델 저장  
if e % 1000 == 0:  
    agent.model.save_weights("./save_model/breakout_dqn.h5")
```


DQN, Keras(breakout)

코드설명

1.Main

2.library

3.DQN

Import-1

1.필요한 라이브러리를 불러온다.

α.Keras

* CNN layer

* Dense layer

* optimizer

* 케라스에서의 딥러닝 모델

```
from keras.layers.convolutional import Conv2D
from keras.layers import Dense, Flatten
from keras.optimizers import RMSprop
from keras.models import Sequential
from skimage.transform import resize
from skimage.color import rgb2gray
from collections import deque
from keras import backend as K
import tensorflow as tf
import numpy as np
import random
import gym
import os
```

Import-2

b. 이미지 전처리

- * input으로 들어오는 이미지 크기 조절
- * RGB를 Gray로 만드는 라이브러리
- * replay memory만드는

c. Tensorflow

- * tensorflow backend
- * tensorflow

d. 기타

- * numpy
- * random
- * gym
- * os

```
from keras.layers.convolutional import Conv2D
from keras.layers import Dense, Flatten
from keras.optimizers import RMSprop
from keras.models import Sequential

from skimage.transform import resize
from skimage.color import rgb2gray
from collections import deque
from keras import backend as K

import tensorflow as tf
import numpy as np
import random
import gym
import os
```

DQN, Keras(breakout)

코드설명

1.Main

2.library

3.DQN

DQN-1

초기화

- render 의 유무
- model의 load 유무
- state 사이즈
- action 사이즈
- epsilon값
- epsilon의 시작점과 끝점 (decay를 위해)
- epsilon의 decay step 정의

```
self.render = False
self.load_model = False
# 상태와 행동의 크기 정의
self.state_size = (84, 84, 4)
self.action_size = action_size
# DQN 하이퍼파라미터
self.epsilon = 1.
self.epsilon_start, self.epsilon_end = 1.0, 0.1
self.exploration_steps = 1000000.
self.epsilon_decay_step = (self.epsilon_start - self.epsilon_end) \
    / self.exploration_steps
```

DQN-2

초기화

- 리플레이 메모리에서 뽑을 배치사이즈 정의
- 학습을 시작할 기준 정의
- 정답 모델로 업데이트 주기 정의
- discount factor
- 리플레이메모리 최대크기 정의
- 스타트할때 action을 임의로 정해주는 설정
- Deeplearning model
- Target model
- update target model

```
self.batch_size = 32
self.train_start = 50000
self.update_target_rate = 10000
self.discount_factor = 0.99
# 리플레이 메모리, 최대 크기 400000
self.memory = deque(maxlen=400000)
self.no_op_steps = 30
# 모델과 타겟모델을 생성하고 타겟모델 초기화
self.model = self.build_model()
self.target_model = self.build_model()
self.update_target_model()
```

DQN-3

초기화-2

- optimizer
- Tensorboard

```
self.optimizer = self.optimizer()

# 텐서보드 설정
self.sess = tf.InteractiveSession()
K.set_session(self.sess)

self.avg_q_max, self.avg_loss = 0, 0
self.summary_placeholders, self.update_ops, self.summary_op = \
    self.setup_summary()
self.summary_writer = tf.summary.FileWriter(
    'summary/breakout_dqn', self.sess.graph)
self.sess.run(tf.global_variables_initializer())

if self.load_model:
    self.model.load_weights("./save_model/breakout_dqn_trained.h5")
```

Save된 모델의 웨이트를 가져올때 사용

DQN-4

Keras로 딥러닝 모델 만들기

CNN Layers

```
def build_model(self):  
    model = Sequential()  
    model.add(Conv2D(32, (8, 8), strides=(4, 4),  
                     activation='relu',  
                     input_shape=self.state_size))  
    model.add(Conv2D(64, (4, 4), strides=(2, 2),  
                     activation='relu'))  
    model.add(Conv2D(64, (3, 3), strides=(1, 1),  
                     activation='relu'))
```

Dense Layer

```
    model.add(Flatten())  
    model.add(Dense(512, activation='relu'))  
    model.add(Dense(self.action_size))  
    model.summary()  
    return model
```


DQN-5

현재 model의 weight를 가져와서 target
model의 웨이트로 업데이트 하는 함수

```
def update_target_model(self):  
    self.target_model.set_weights(self.model.get_weights())
```

action을 선택하는 함수(policy) : 여
기서는 Epsilon greedy

```
def get_action(self, history):  
    history = np.float32(history / 255.0)  
    if np.random.rand() <= self.epsilon:  
        return random.randrange(self.action_size)  
    else:  
        q_value = self.model.predict(history)  
        return np.argmax(q_value[0])
```

DQN-6

Replay Memory

state, action, reward, next state를 리플레이 메모리에 저장해주는 함수

```
def append_sample(self, history, action, reward, next_history, dead):  
    self.memory.append((history, action, reward, next_history, dead))
```

DQN-7

Replay memory-2

리플레이 메모리에서 뽑아온 배치로 모델을 학습하는 함수

```
def train_model(self):  
    if self.epsilon > self.epsilon_end:  
        self.epsilon -= self.epsilon_decay_step  
  
    mini_batch = random.sample(self.memory, self.batch_size)
```

DQN-8

초기화

optimizer

Tensorboard

각 에피소드 당 학습 정보를 기록하

는 함수

```
def setup_summary(self):
    episode_total_reward = tf.Variable(0.)
    episode_avg_max_q = tf.Variable(0.)
    episode_duration = tf.Variable(0.)
    episode_avg_loss = tf.Variable(0.)

    tf.summary.scalar('Total Reward/Episode', episode_total_reward)
    tf.summary.scalar('Average Max Q/Episode', episode_avg_max_q)
    tf.summary.scalar('Duration/Episode', episode_duration)
    tf.summary.scalar('Average Loss/Episode', episode_avg_loss)

    summary_vars = [episode_total_reward, episode_avg_max_q,
                    episode_duration, episode_avg_loss]
    summary_placeholders = [tf.placeholder(tf.float32) for _ in
                           range(len(summary_vars))]
    update_ops = [summary_vars[i].assign(summary_placeholders[i]) for i in
                 range(len(summary_vars))]
    summary_op = tf.summary.merge_all()
    return summary_placeholders, update_ops, summary_op
```

DQN-9

초기화

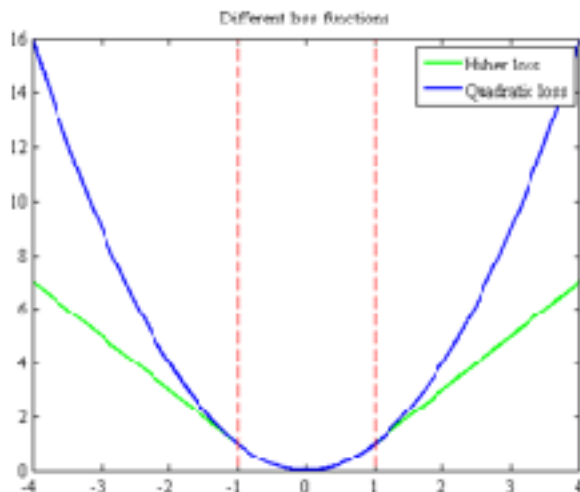
이미지 전처리를 위한 함수

```
# 학습속도를 높이기 위해 흑백화면으로 전처리
def pre_processing(observe):
    processed_observe = np.uint8(
        resize(rgb2gray(observe), (84, 84), mode='constant') * 255)
    return processed_observe
```

DQN-10

Optimizer 함수

여기서는 Huber Loss 사용



<https://goo.gl/images/XGsfYx>

```
def optimizer(self):
    a = K.placeholder(shape=(None,), dtype='int32')
    y = K.placeholder(shape=(None,), dtype='float32')

    prediction = self.model.output

    a_one_hot = K.one_hot(a, self.action_size)
    q_value = K.sum(prediction * a_one_hot, axis=-1)
    error = K.abs(y - q_value)

    quadratic_part = K.clip(error, 0.0, 1.0)
    linear_part = error - quadratic_part
    loss = K.mean(0.5 * K.square(quadratic_part) + linear_part)

    optimizer = RMSprop(lr=0.00025, epsilon=0.01)
    updates = optimizer.get_updates(self.model.trainable_weights, [], loss)
    train = K.function([self.model.input, a, y], [loss], updates=updates)

    return train
```

DQN, Keras(breakout)
시연

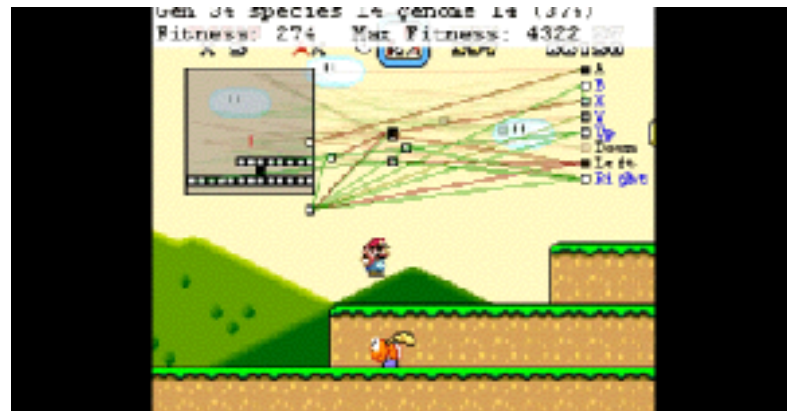
슈퍼마리오 환경 구축

Deep learning+Reinforcement Learning

Human



A.I.



다른 도메인에서도 적용이 가능하다!

여러 환경에 사용



하지만 환경에 따른 적절한 강화학습 알고리즘을 적용해야 한다.

예민한 강화학습



State 크기, action이 다르다.

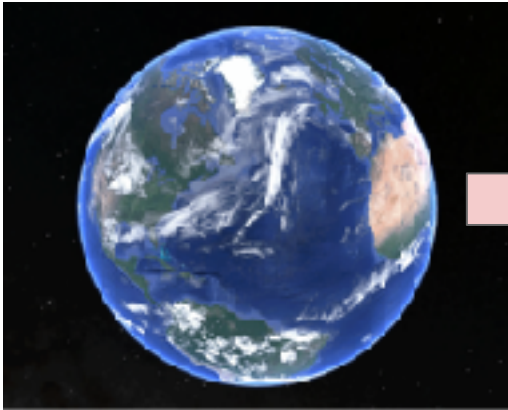
같은 알고리즘을 사용하더라도, Deep learning model, hyper parameter을 적절하게 사용해야 한다.

슈퍼마리오 설치에 필요한 네가지

Emulator



Environment



Programming Language



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity D

Initialize action-value function Q with random weights

for episode = 1, 2, 3, ... do

 Initialize sequence s_0 in \mathcal{S} and previous action $a_{t-1} = \text{null}$

 for $t = 1, 2, 3, \dots$ do

 With probability ϵ select a random action a_t

 otherwise select $a_t = \arg \max_a Q(s_t, a)$

 Execute action a_t in the environment and observe reward r_t and next state s_{t+1}

 Set $d_{t+1} = \text{true}$ if terminal condition $d_t = \text{true}$

 Store transition $(s_t, a_t, r_t, s_{t+1}, d_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1}, d_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } d_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a') & \text{for non-terminal } d_{j+1} \end{cases}$

 Perform a gradient descent step on $\{J_j = (y_j - Q(s_j, a_j))^2\}$ according to equation 3

 end for

end for

Algorithm

Programming Language - Python



1. <https://www.python.org/downloads/> - 3.5 version
2. <https://www.anaconda.com/download/> - Anaconda
3. <https://www.tensorflow.org/install/> - TensorFlow
4. <https://keras.io/#installation> - Keras



Emulator -FCUX

Emulator



<http://www.fceux.com/web/home.html>

Ubuntu

```
sudo apt-get update
```

```
sudo apt-get install fceux
```

MAC

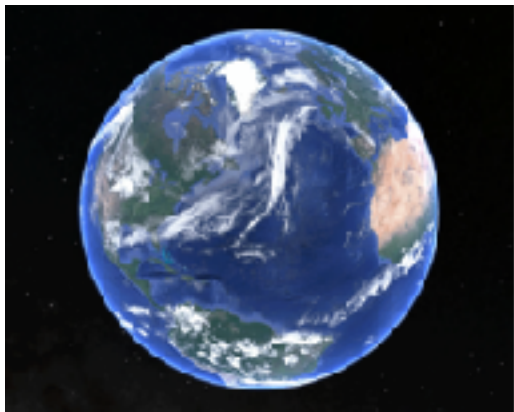
<https://brew.sh/> -homebrew website

Terminal open -> brew install fceux

```
sudo apt-get install fceux
```

OpenAI_Gym

Environment



OpenAI_Gym

<https://github.com/openai/gym>

```
pip3 install gym
```

```
git clone https://github.com/openai/gym.git
```

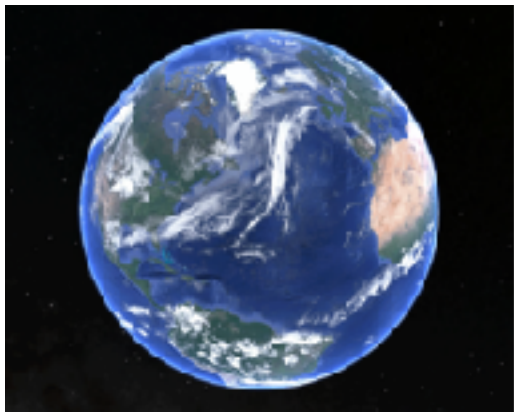
```
cd gym
```

```
pip install -e
```

OpenAI의 Gym을 사용하면 보다 쉽게 강화학습 실험이 가능하다.

OpenAI_Baselines

Environment



Baselines

<https://github.com/openai/baselines>

`pip3 install baselines`

```
git clone https://github.com/openai/baselines.git
```

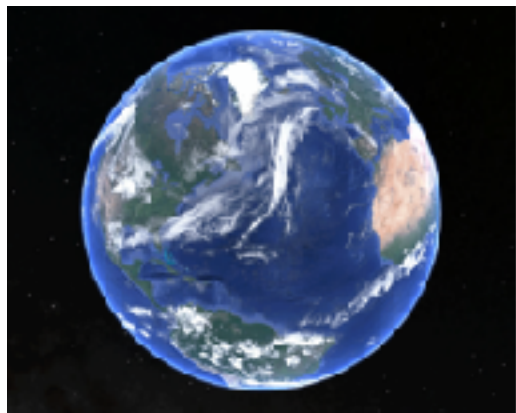
```
cd baselines
```

```
pip install -e .
```


SuperMario

Philip Paquette

Environment



<https://github.com/ppaquette/gym-super-mario>

```
pip3 install gym-pull
```

```
import gym
import gym_pull
gym_pull.pull('github.com/ppaquette/gym-super-mario')
env = gym.make('ppaquette/SuperMarioBros-1-1-v0')
```



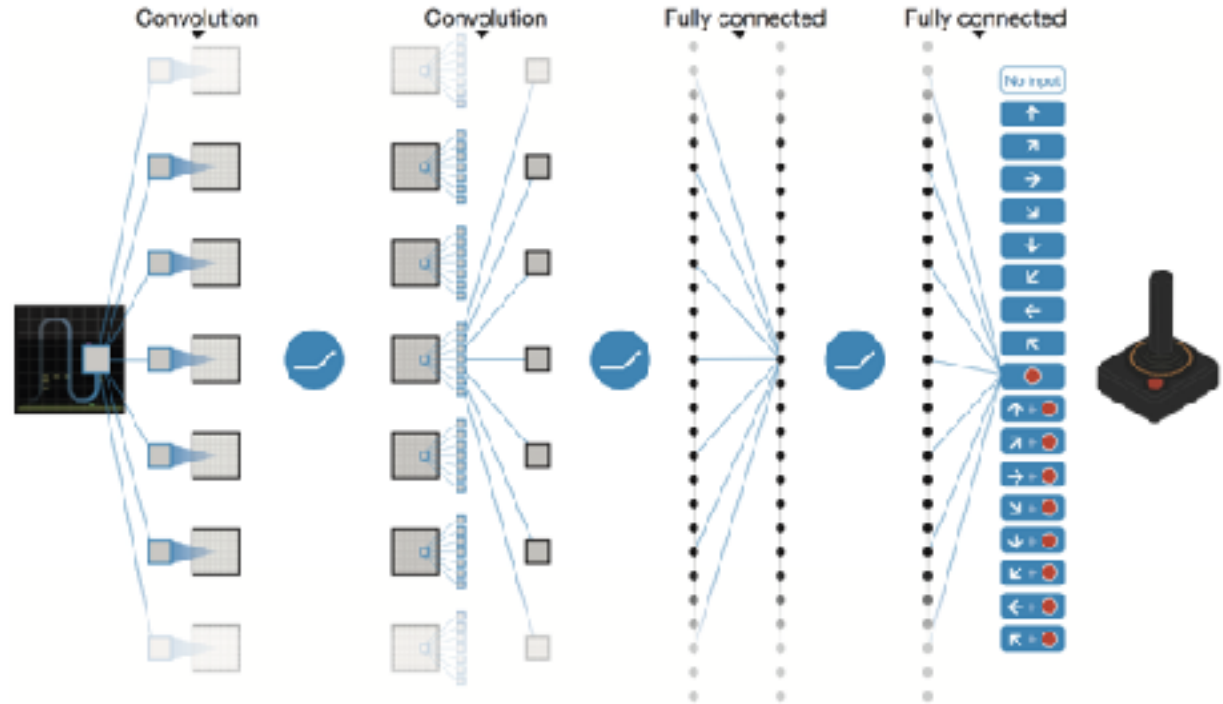
Algorithm-DQN

Algorithm



DeepMind

DEEP Q-NETWORK



설치에 문제가 생긴다면?

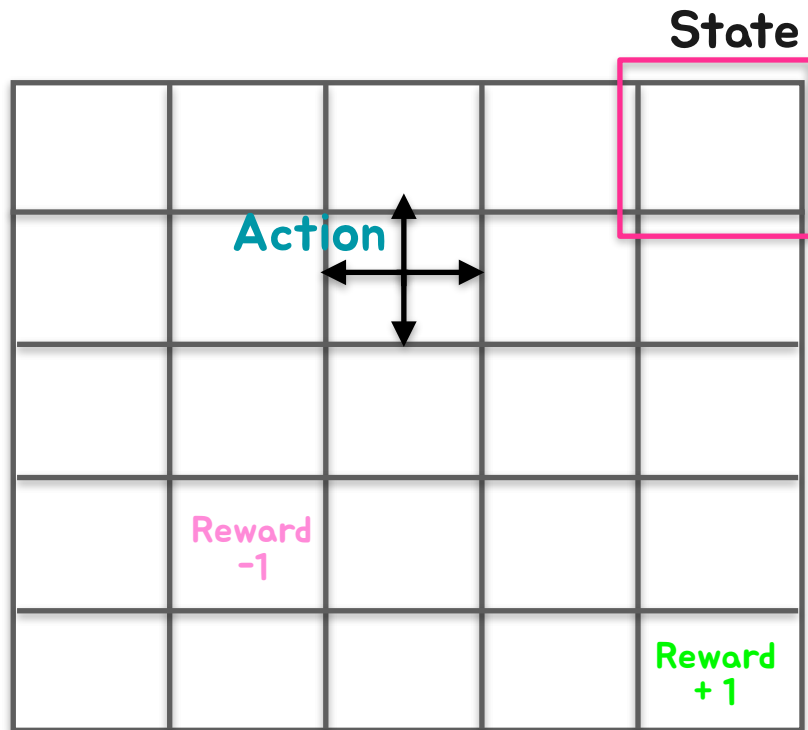
https://github.com/wonseokjung/KIPS_Reinforcement/tree/master/DQN

오늘 강화학습 실습강의 전용 github에 자세한 설치법 올려놓았습니다.

DQN을 이용한 인공지능 슈퍼마리오 만들기

그리드월드와 슈퍼마리오환경

그리드월드와 어떻게 다를까?



State : 그리드의 좌표

Action : 상, 하, 좌, 우

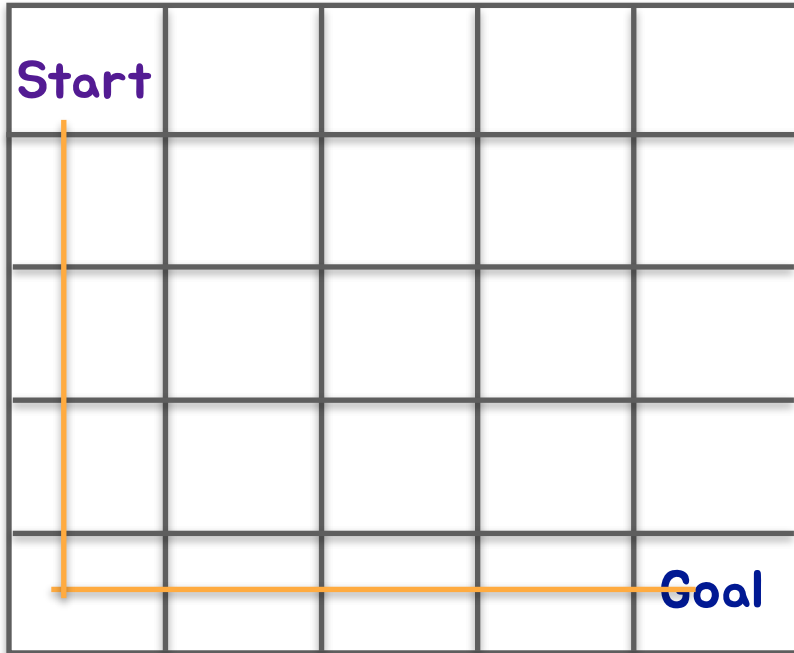
Reward : 함정 = -1, 목표 = 1

Transition Probability : 1

Discount factor : 0.9

Goal의 비교

그리드월드의 목표는 goal state로 가는것

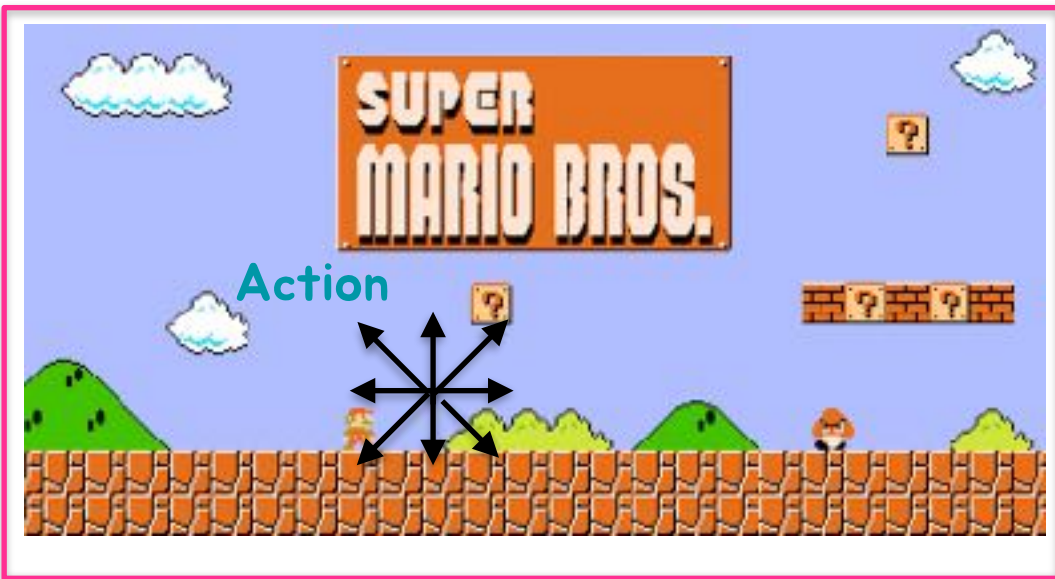


슈퍼마리오는 깃발을 잡는이 목표



슈퍼마리오에서의 환경

State



State : 화면

Action : 상, 하, 좌, 우, 점프, 달리기, action의 조합

Reward : 앞으로 전진할때 Reward +1, 뒤로가면 -1

Transition Probability : 1

Discount factor : 0.9

도착지인 깃발에 가까이 갈수록 높은 reward를 받는다.

계속되는 실패...

이슈

1. 마리오가 앞으로 전진하지 않으려고 하는 현상
2. State가 breakout보다 더 복잡하고 action이 많다.

Reward 설정



Penalty, Bonus reward추가

목표달성하지 못하면 -

시간이 지날때마다 -

깃발에서 멀어지면 -

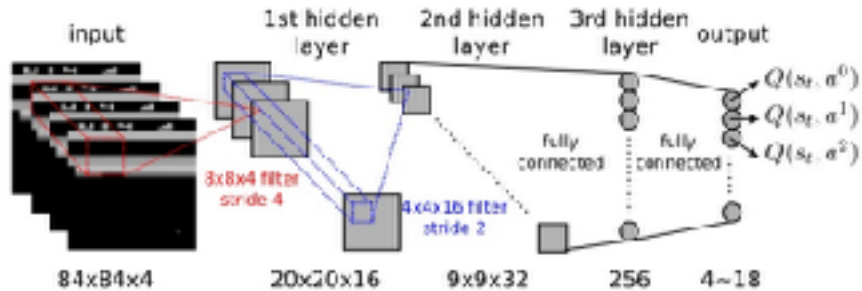
깃발에 가까워지면 +

목표에 도착하면 +

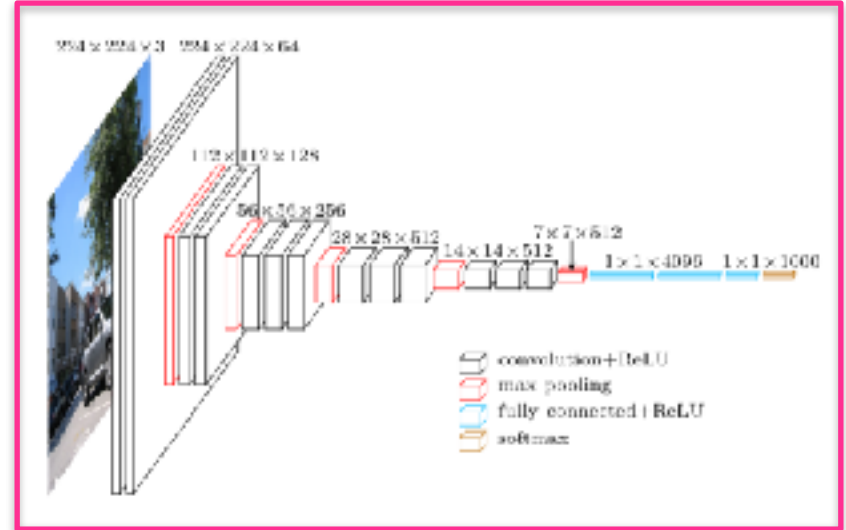
Deep learning model

VGG model and regular 비교

더 깊게 살펴보자



<https://goo.gl/images/s8XrCK>



<https://goo.gl/images/eoXooC>

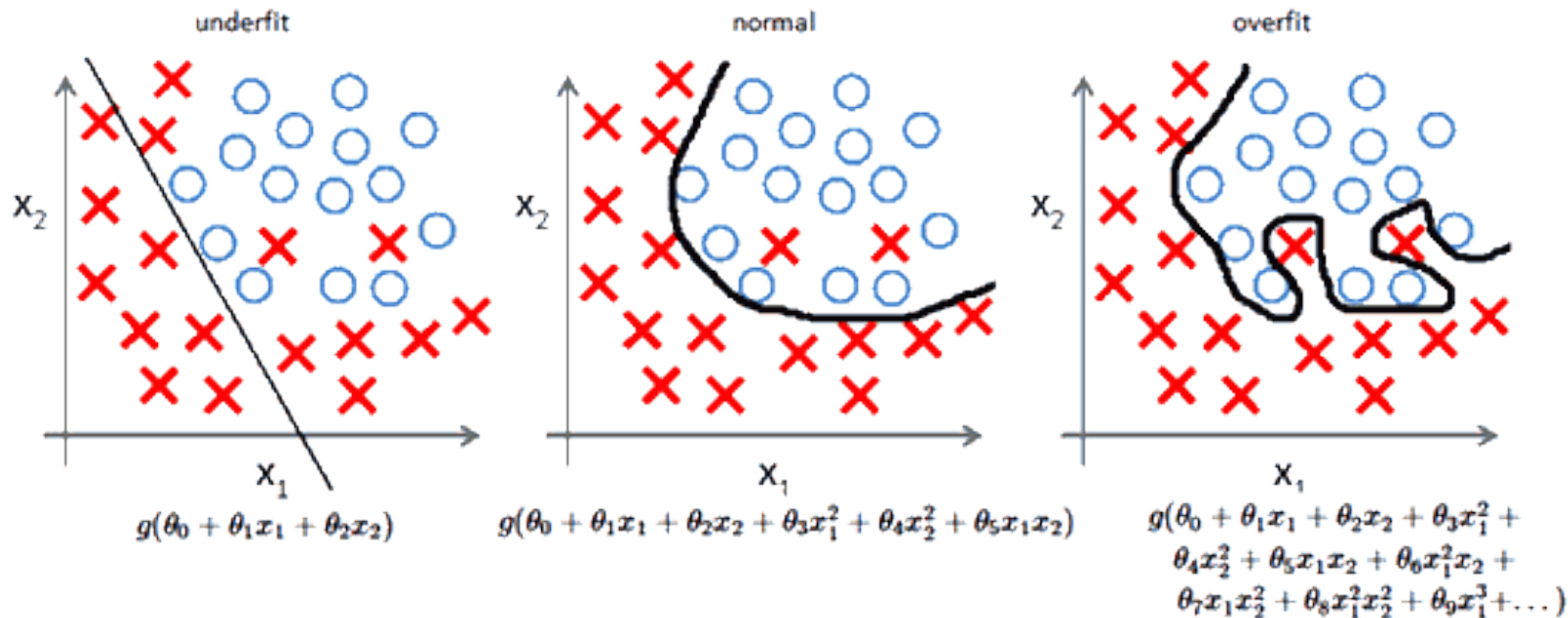
성공!!

DQN을 이용한 인공지능 슈퍼마리오 만들기

시연

레벨 1 을 클리어 하는 마리오는 만든 후..

다른 레벨에서 성능이 매우 떨어진다. Overfitting이 아닐까?



<https://goo.gl/images/6uDmqH>

강화학습의 연구는 활발히 진행중 !

Reward

Exploration

Algorithm

감사합니다.

Github:

<https://github.com/wonseokjung>

Facebook:

<https://www.facebook.com/ws.jung.798>

Blog:

<https://wonseokjung.github.io/>

References:

- * Reinforcement Learning: An Introduction Richard S. Sutton and Andrew G. Barto Second Edition, in progress MIT Press, Cambridge, MA, 2017
- * <https://github.com/rlcode/reinforcement-learning-kr>