강화학습의 이론과 실제

정원석

모두의연구소

강사소개

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

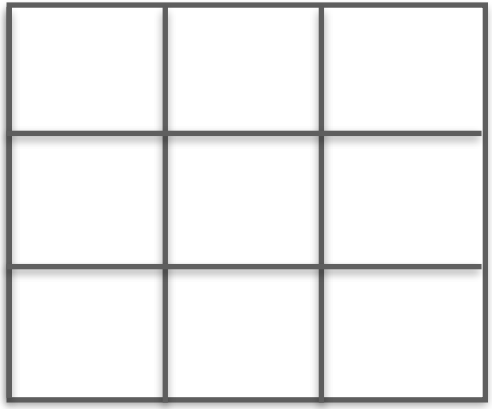
Model-free

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	Bat Man	North	Q1	91
3	Bat Man	South	Q1	87
4	Bat Man	West	Q1	99
5	Bat Man	East	Q1	102
6	Ben Ten	South	Q1	125
7	Ben Ten	West	Q1	140
8	Ben Ten	East	Q1	107
9	Ben Ten	North	Q1	133
10	Bob The Builder	West	Q1	79
11	Bob The Builder	South	Q1	85
12	Bob The Builder	East	Q1	91
13	Bob The Builder	North	Q1	73
14	Mr Maker	East	Q1	49
15	Mr Maker	North	Q1	50
16	Mr Maker	West	Q1	51
17	Mr Maker	South	Q1	59

After Deeplearning

Image, text, voice...

고전 강화학습이 중요한 이유

Classic RL + DeepLearning = !

DQN

Q-learning + CNN -> DQN

풀리지 않은 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/KIPS_Reinforcement

Code + Jupyter Notebook 주석 + 설치법 모두 제공

강의 끝난뒤에도 궁금한점이 있으시면 개인적으로 연락주세요!

Review

Markov Decision Process

Review

Return of Episode

Episode 동안 Return된 Reward 의 합

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

Review

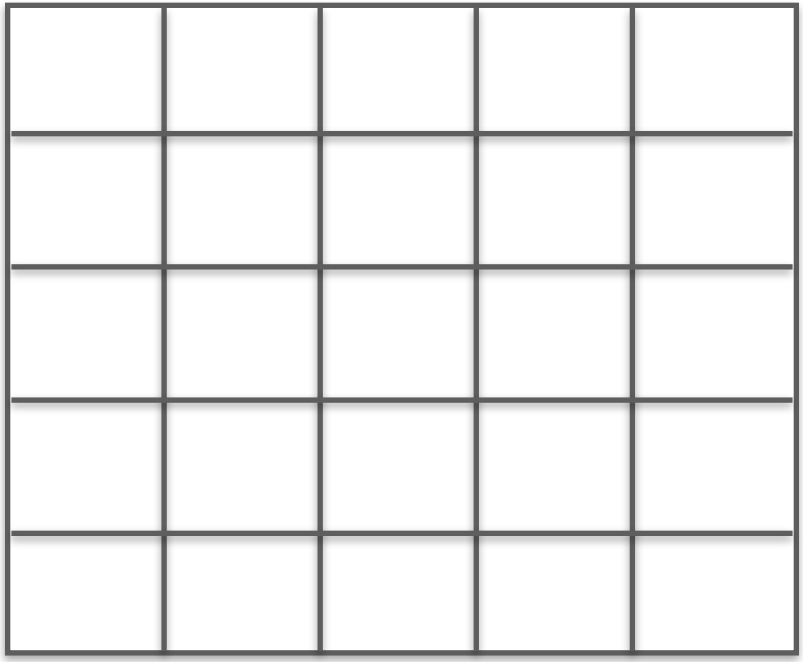
Discounted Return

Discounted factor가 적용된 Reward의 합

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

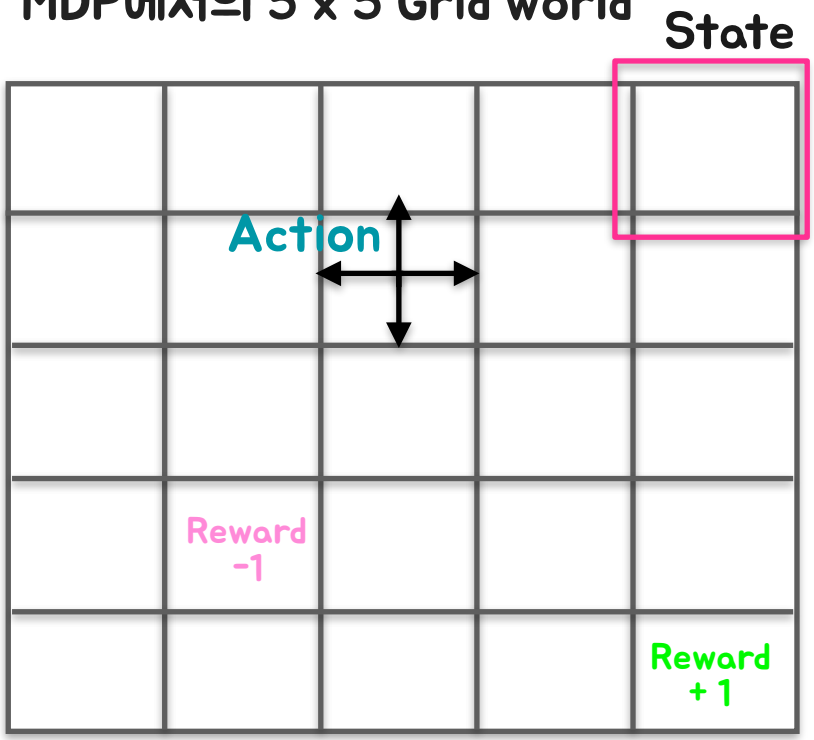
Grid World Environment

MDP 에서의 5 x 5 Grid world



Grid World Environment

MDP에서의 5 x 5 Grid world



State : 그리드의 좌표

Action : 상, 하 , 좌, 우

Reward : 함정 = -1, 목표 = 1

Transition Probability : 1

Discount factor : 0.9

Review

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

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

Review

Action Value function
(Policy를 따른 action-value function)

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

Review

A Fundamental property of value function

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

Bellman equation !

Optimal Policy -state

Value를 최대로 !

Optimal state value function

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

Optimal Policy - state action

Value를 최대로 !

Optimal state-action value function

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

Bellman Optimality equation

Bellman optimality equation v*

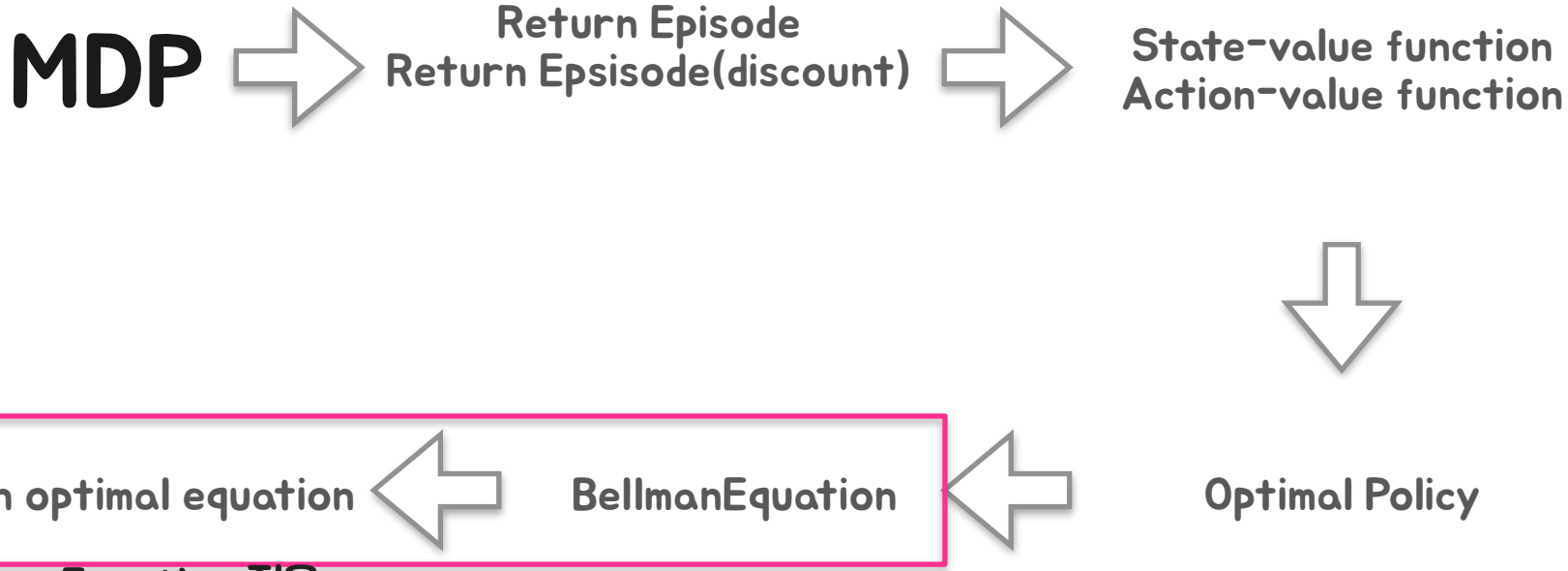
$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a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

Bellman Optimality equation

Bellman optimality equation q*

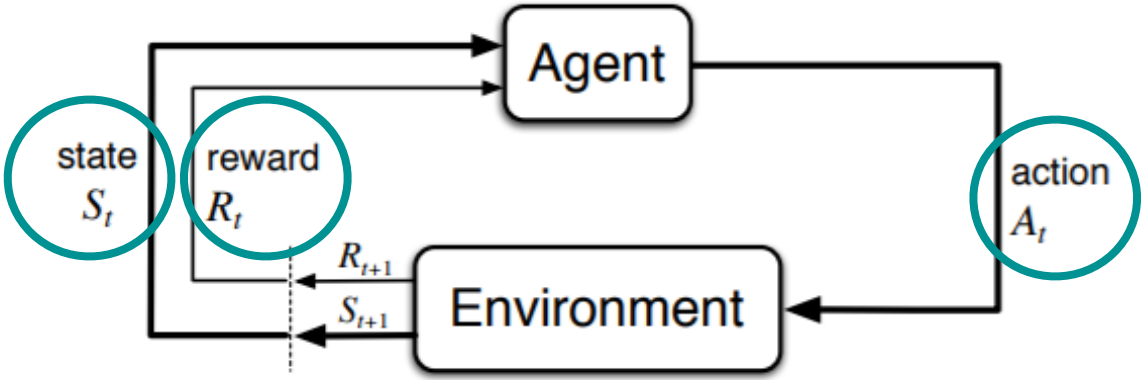
$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right].
\end{aligned}
$$

흐름


MDP
Return Episode
Return Epsisode(discount)
State-value function
Action-value function
Optimal Policy
BellmanEquation
Bellman optimal equation
Bellman Equation 적용

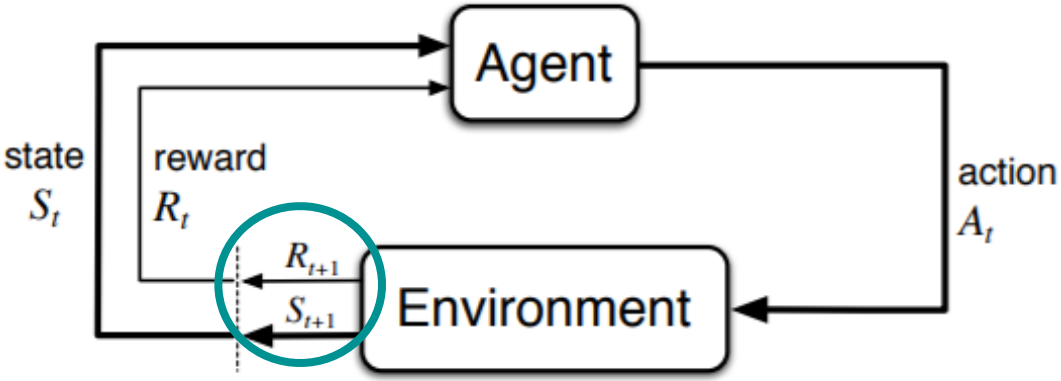
Dynamic Programming

조건



State, Reward, Action

조건



Transition Probability
또한 주어졌다.

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

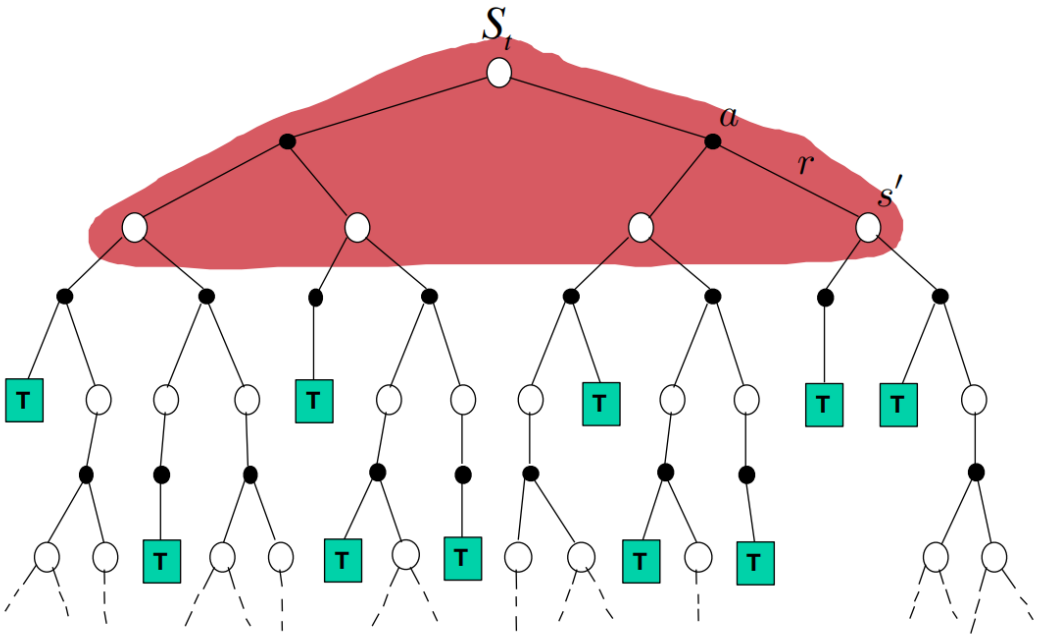
Dynamic Programming

Dynamic programming의 Key idea!

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

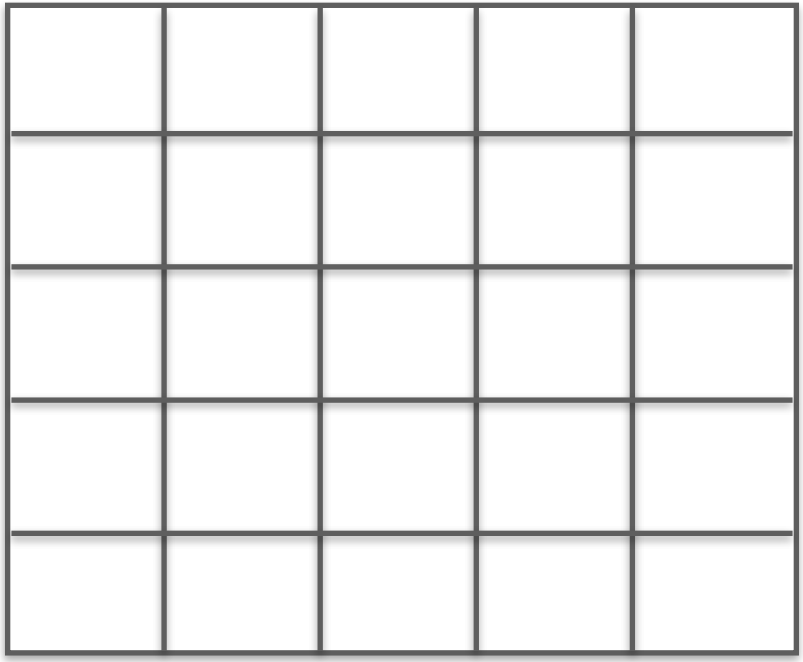
Dynamic programming

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



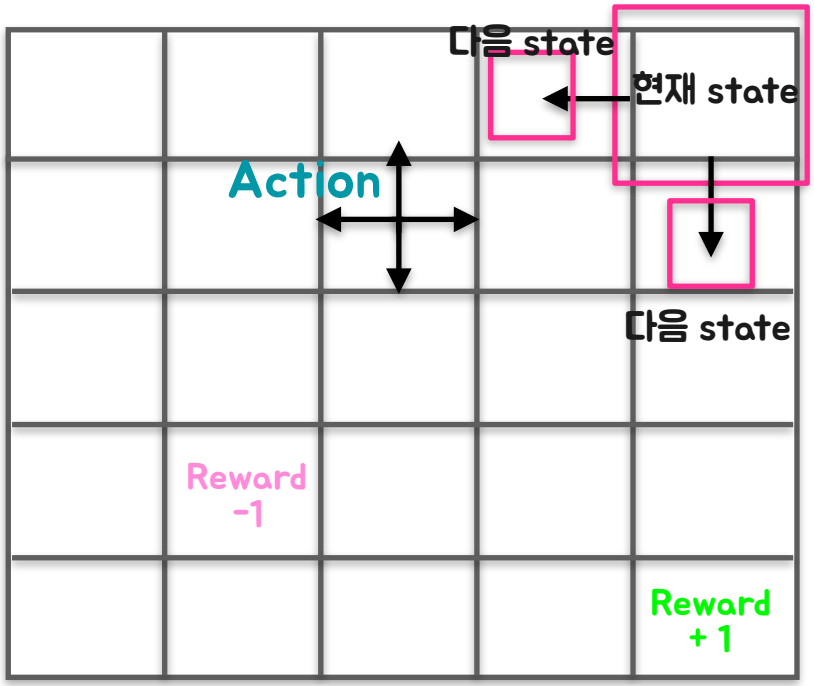
Grid World Environment

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.

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

State!

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

두 종류의 Optimal Value functions

State Value

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

Bellman optimality equations 충족

두 종류의 Optimal Value functions

Action Value

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

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

Bellman optimality equations 충족

Dynamic Programming

두 종류의 최적 Value function

State-action Value function

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

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

Dynamic Programming

Policy Iteration

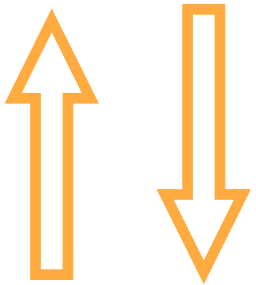
Value Iteration

Policy iteration

최적의 Policy를
찾기위한 **두가지**
과정

1.Policy를 따라 state-value를 계산하자!

Policy Evaluation



2. 더 좋은 Policy를 찾자!

Policy Improvement

Policy iteration- Policy Evaluation

Update Rule을 사용하여 Evaluation을 한다.

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

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

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.

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

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

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

Policy iteration- Improvement

Greedy Policy 적용

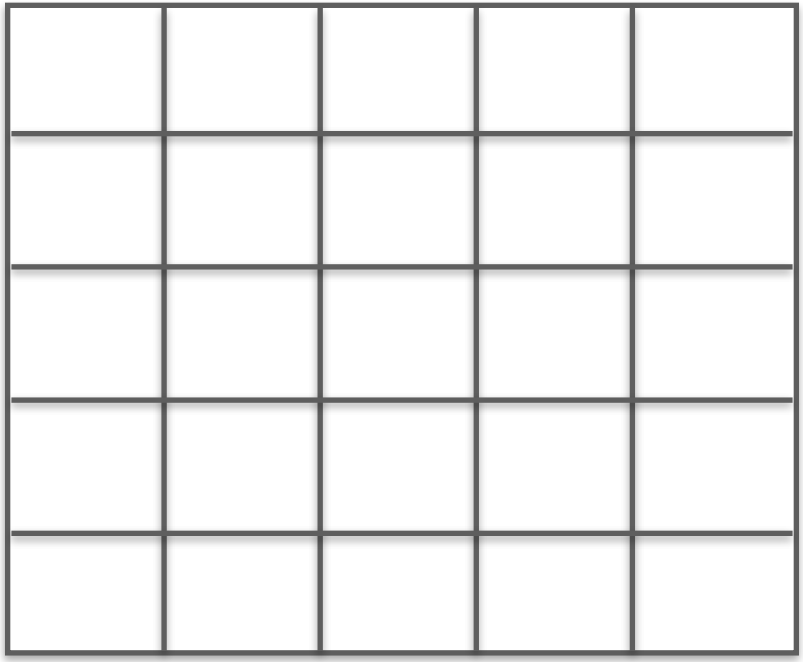
$$
\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_{\pi'}(s') \right].
\end{aligned}
$$

Policy iteration

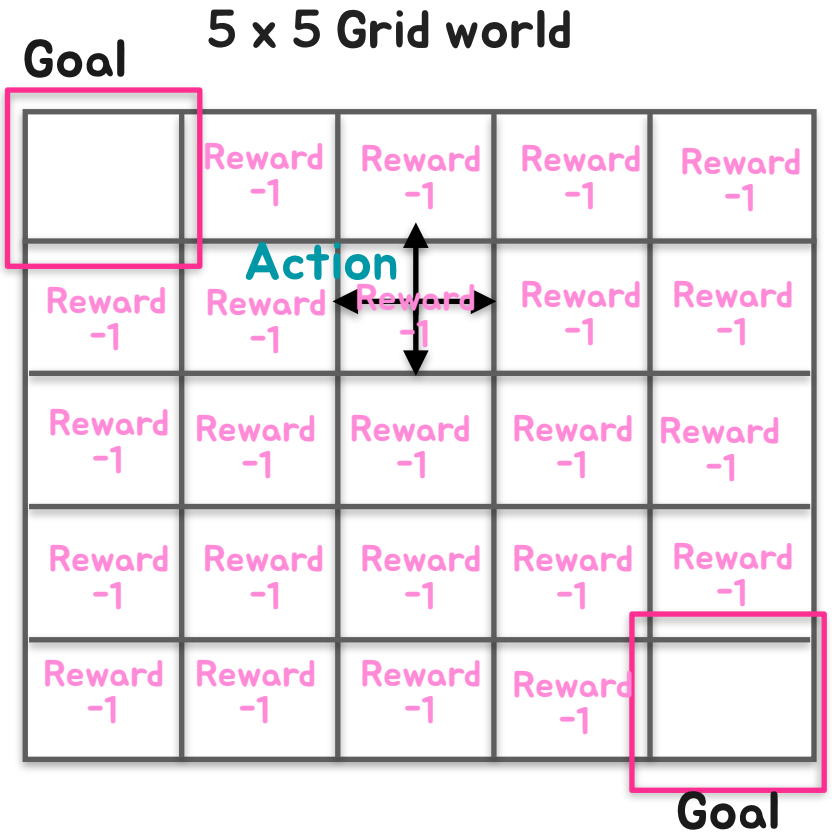
Policy iteration은 Optimal policy를 찾을때까지
Policy Evaluation과 Policy Improvement를 반복한다.

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

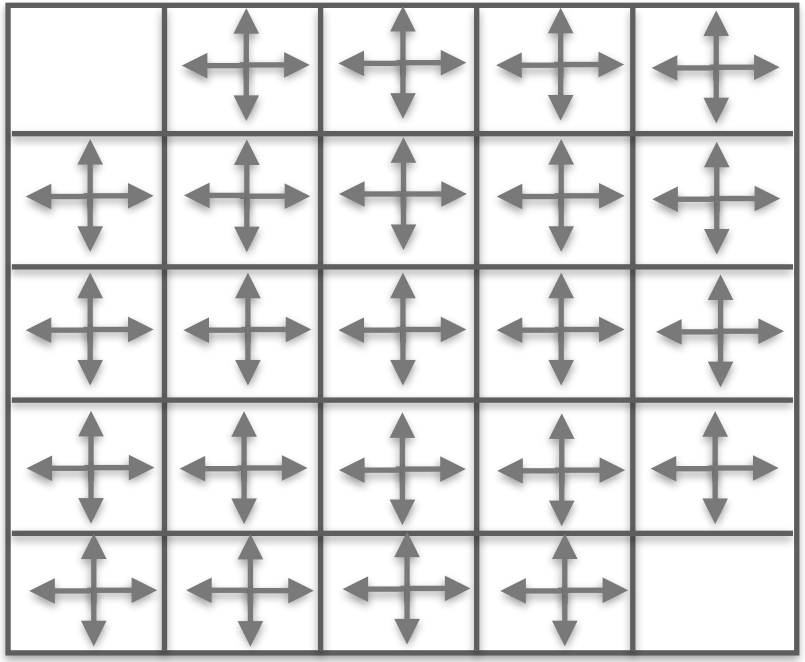
Grid World Environment - Policy iteration

k=0 일때 (초기화)

Vk

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



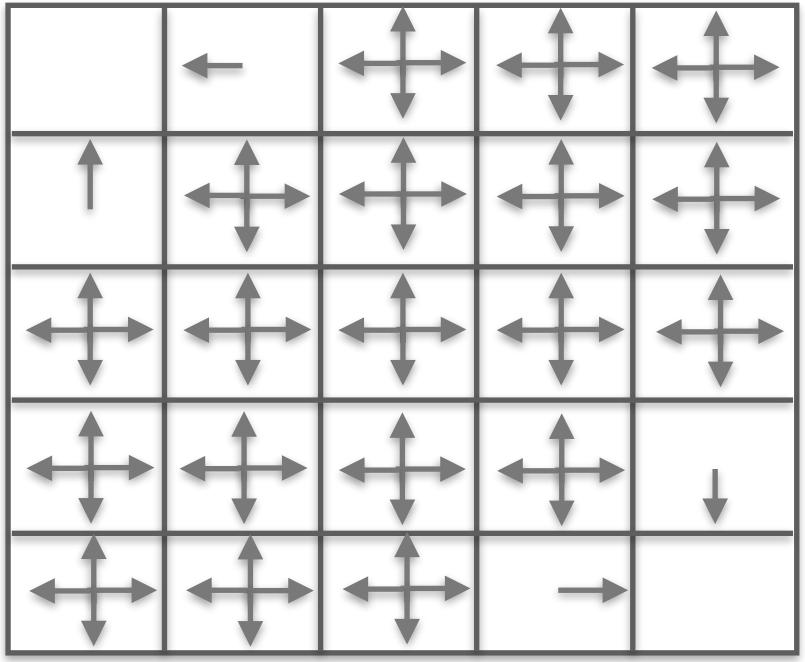
Grid World Environment - Policy iteration

k=1

Vk

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



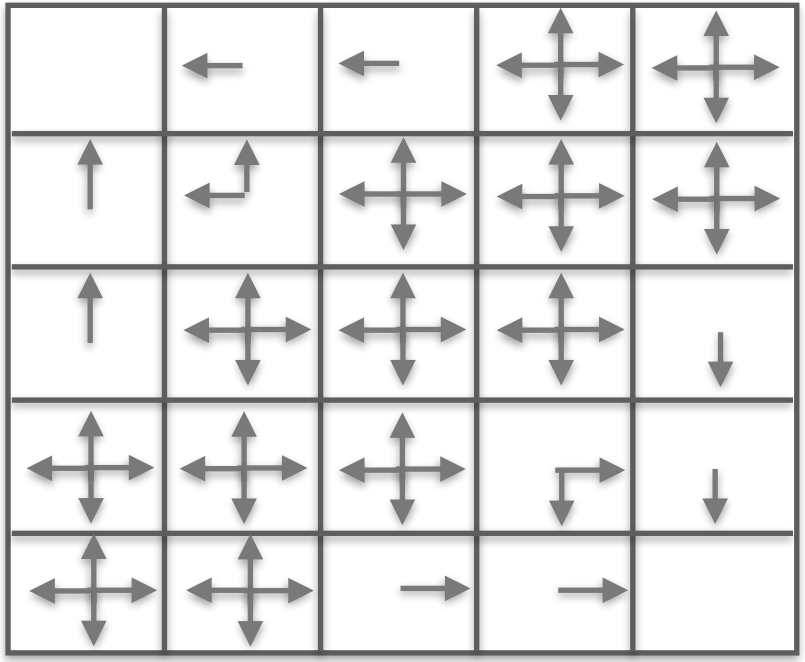
Grid World Environment - Policy iteration

k=2

Vk

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



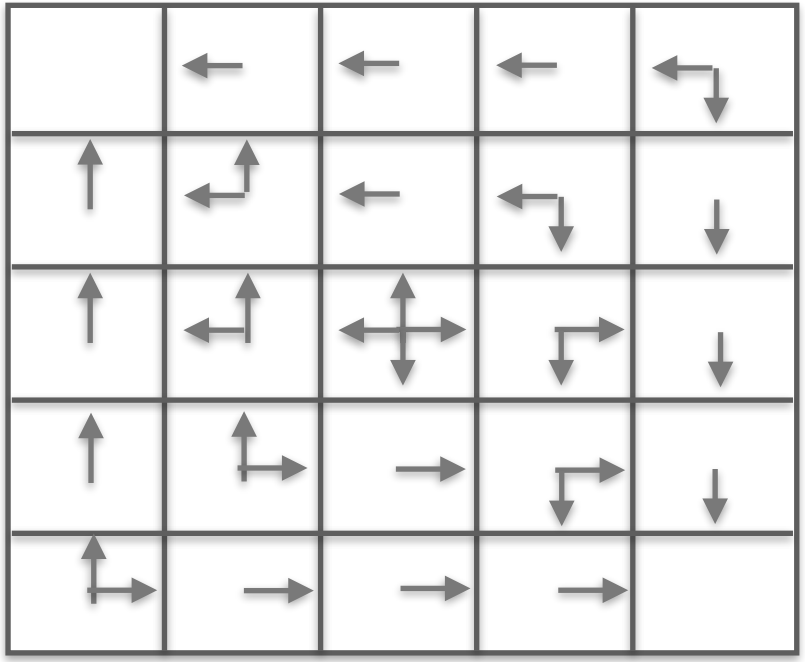
Grid World Environment - Policy iteration

k=inf

Vk

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 + 코드설명

Dynamic Programming

Policy Iteration

Value Iteration

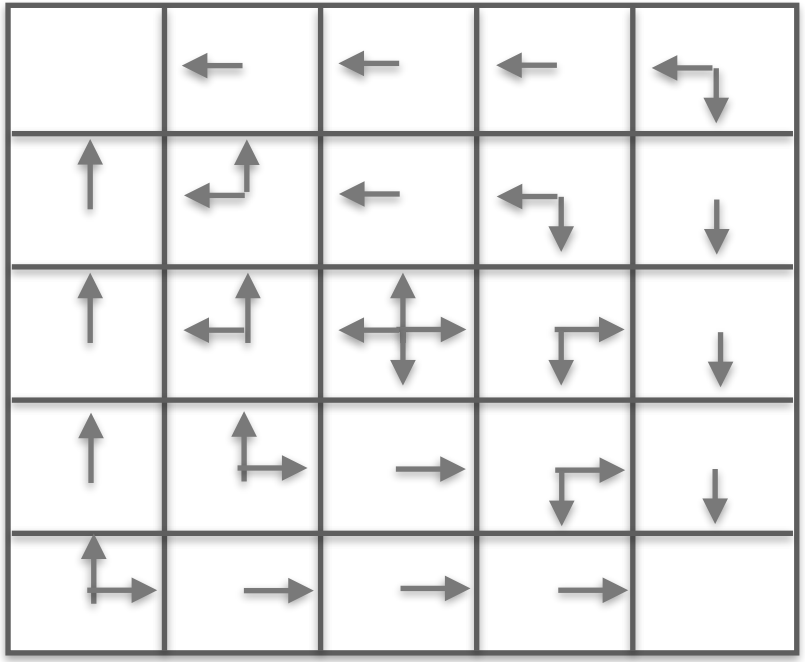
Grid World Environment - Value iteration

k=수렴하면

Vk

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!

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

State, Action!

Value iteration

시연 + 코드설명

Model이 없다면??

실제로 경험을 해보며 환경과 상호작용을 해야한다.

Monte Carlo method

Monte Carlo

Monte Carlo method는 Dynamic programing처럼
모든 정보를 알고 시작 하는 것이 아닌
실제로 경험을 하며 환경과 상호작용을 한다.

Monte Carlo

실제로 경험을 하며 배우는 방법이 좋은 점은 environment의 정보가 없어도 실제로 경험을 하며 optimal behavior을 이루기 때문

Monte Carlo

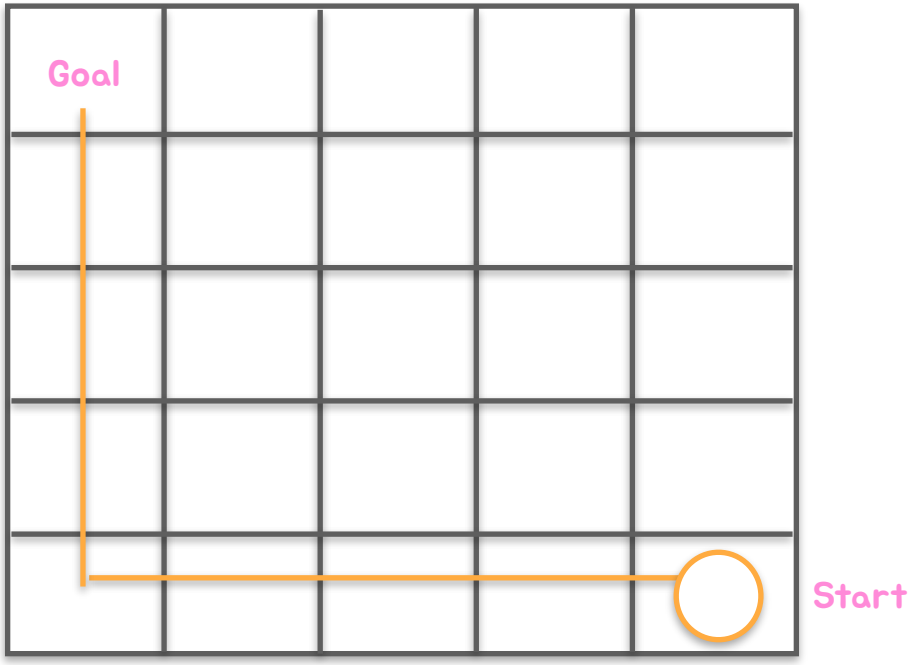
Monte Carlo는 episode-by-episode로 업데이트 한다

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 + 코드설명

Temporal-Difference Learning

Temporal-Difference Learning

만약 강화학습을 대표할 수 있는 아이디어가 있다면 그 것은 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 Carolo에서의 G_t를 알아야 업데이트 가능 :

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

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-Diffrenece Learning이 Sarsa와 Q-learning의 바탕 아이디어가 되었다.

Temporal-Diffrenece Learning

On-policy

Sarsa

Off-policy

Q-learning

Temporal-Diffrenece Learning

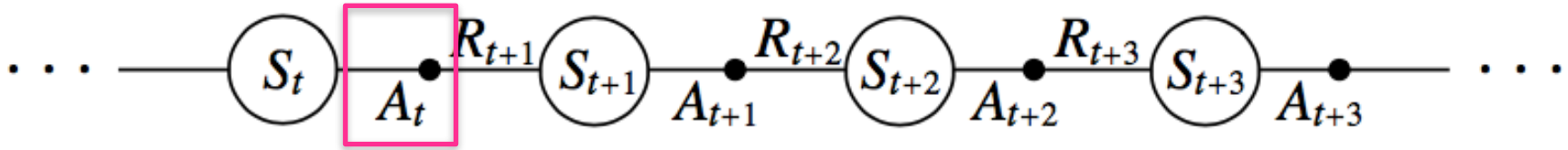
Sarsa

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.

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

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 $Q(\mathit{terminal}, \cdot) = 0$

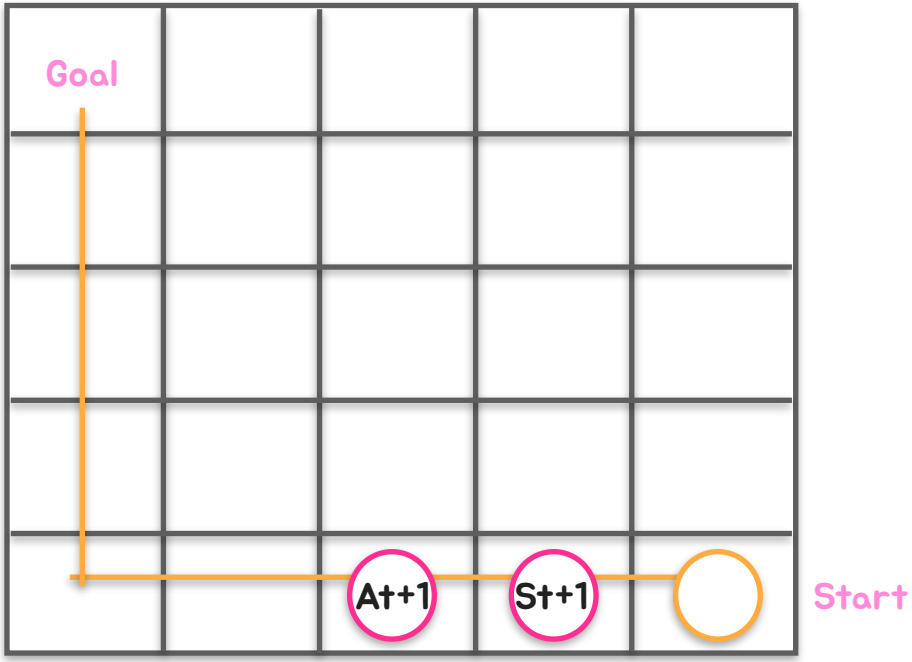
Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Sarsa-pseudo code

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

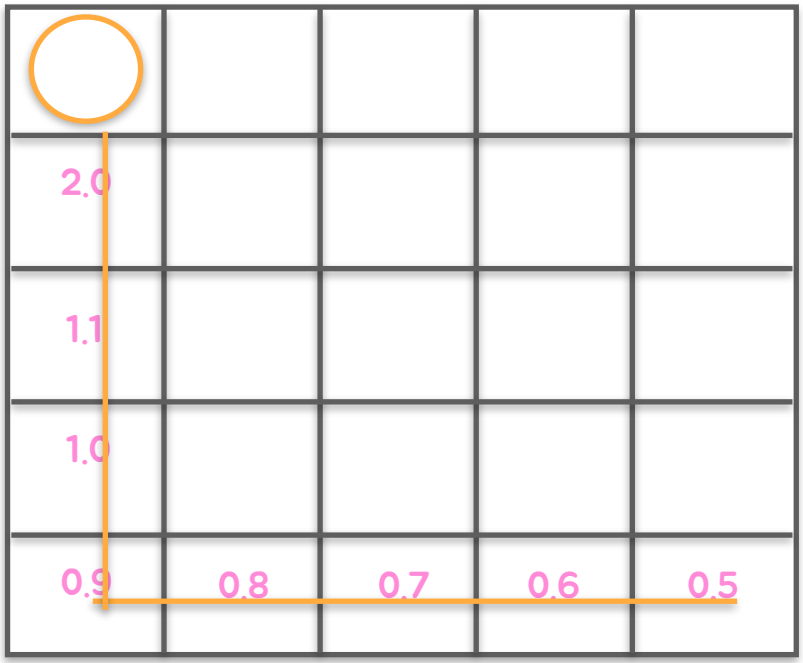
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Sarsa-gridworld



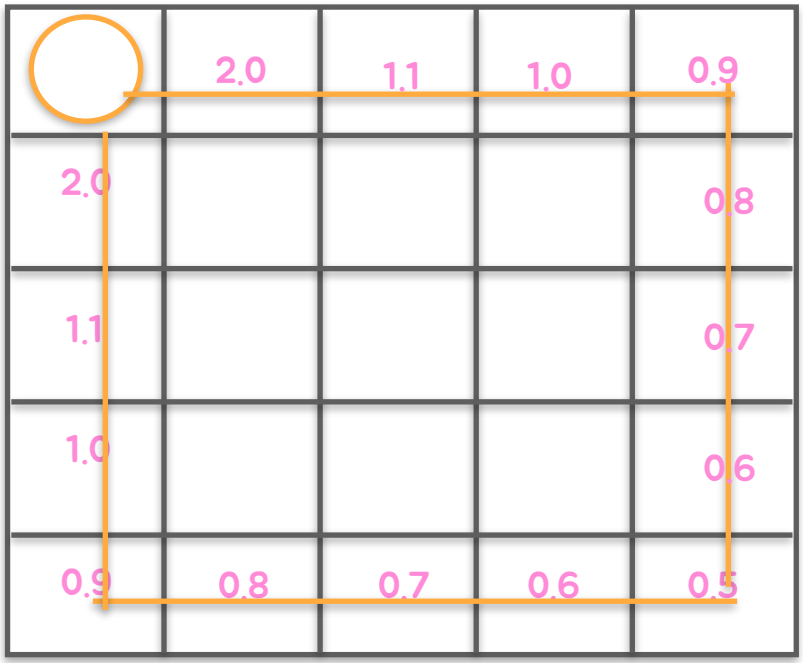
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Sarsa-gridworld



경험하지 않은 스테이트의 정보가 없다.

Sarsa-gridworld



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

Sarsa

시연 + 코드설명

Temporal-Diffrenece Learning

Sarsa

Q-learning

Q-learning

Q-lerning이라고 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.

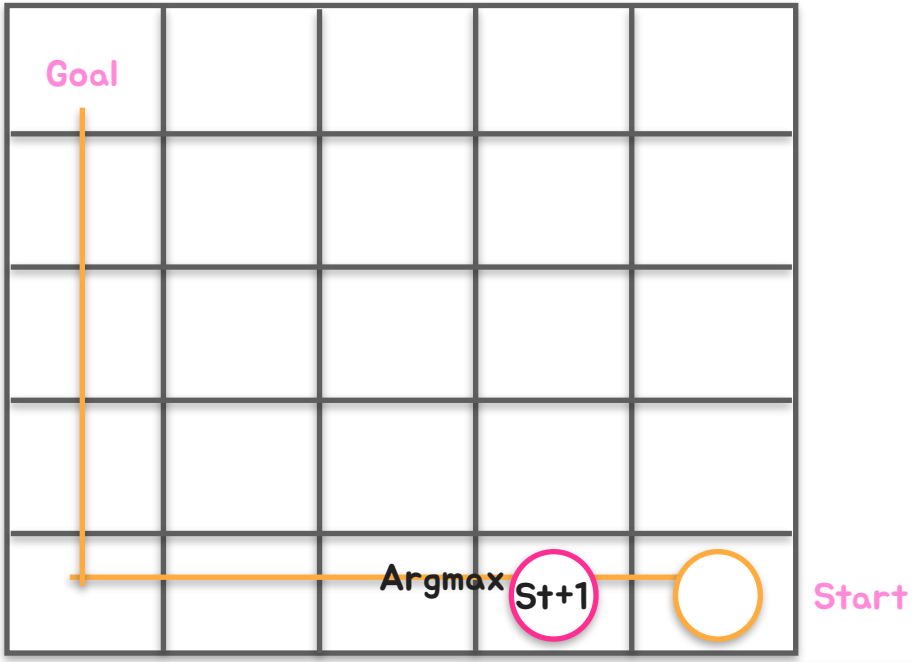
Q-learning-pseudo code

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

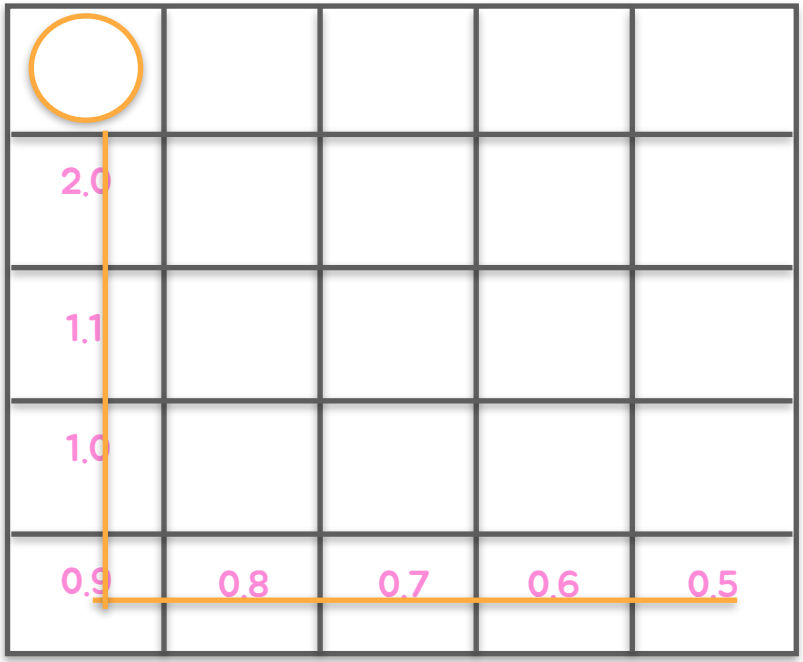
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

qlearning-gridworld



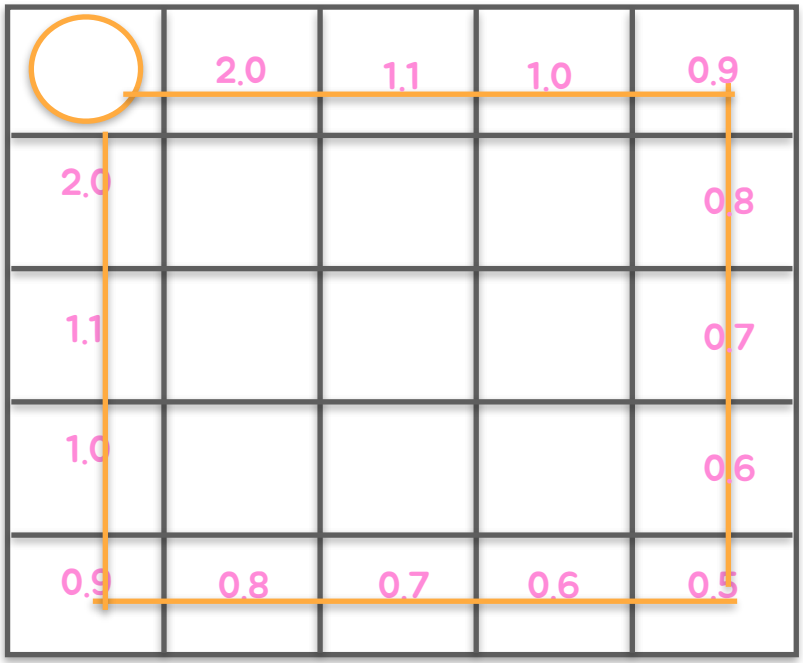
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Q-learning- gridworld



경험하지 않은 스테이트의 정보가 없다.

Q-learning- gridworld

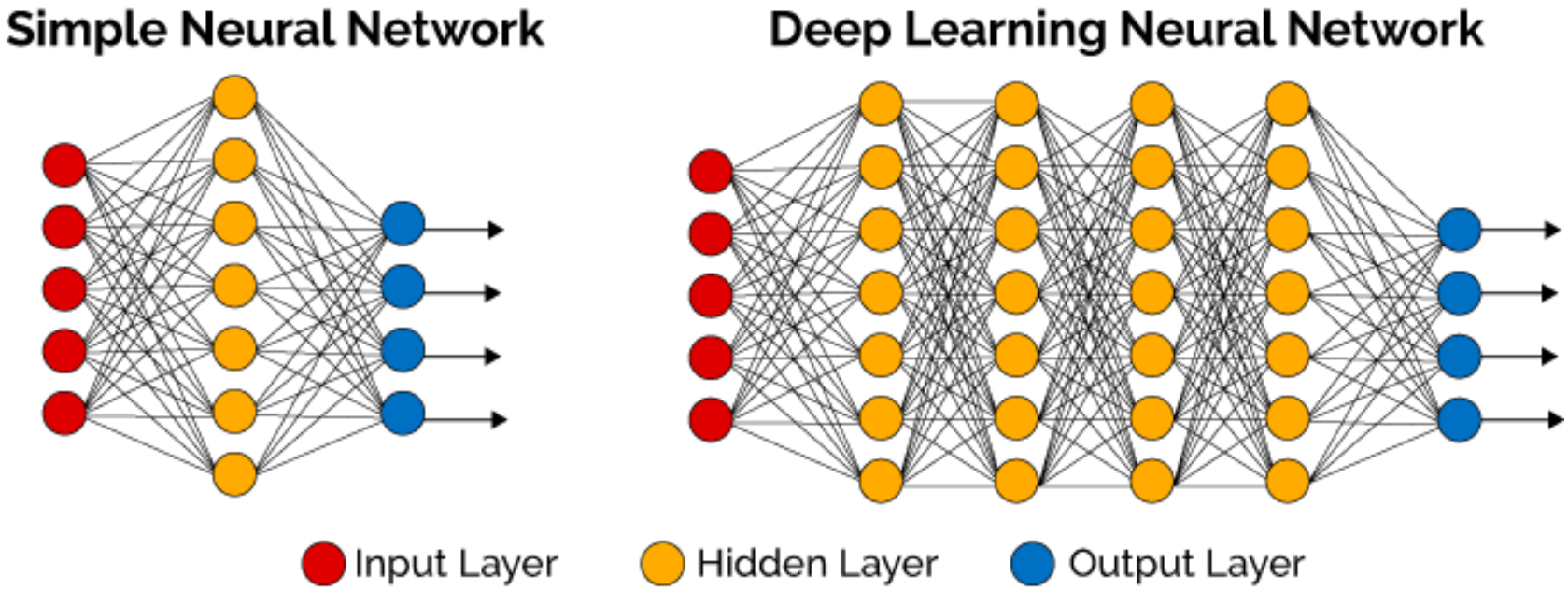


경험을 여러번 해보며 action-value를 업데이트한다.
Policy는 Off-policy

Q-learning
시연 + 코드설명

딥러닝 프레임워크 케라스 소개 슈퍼마리오 환경 구축

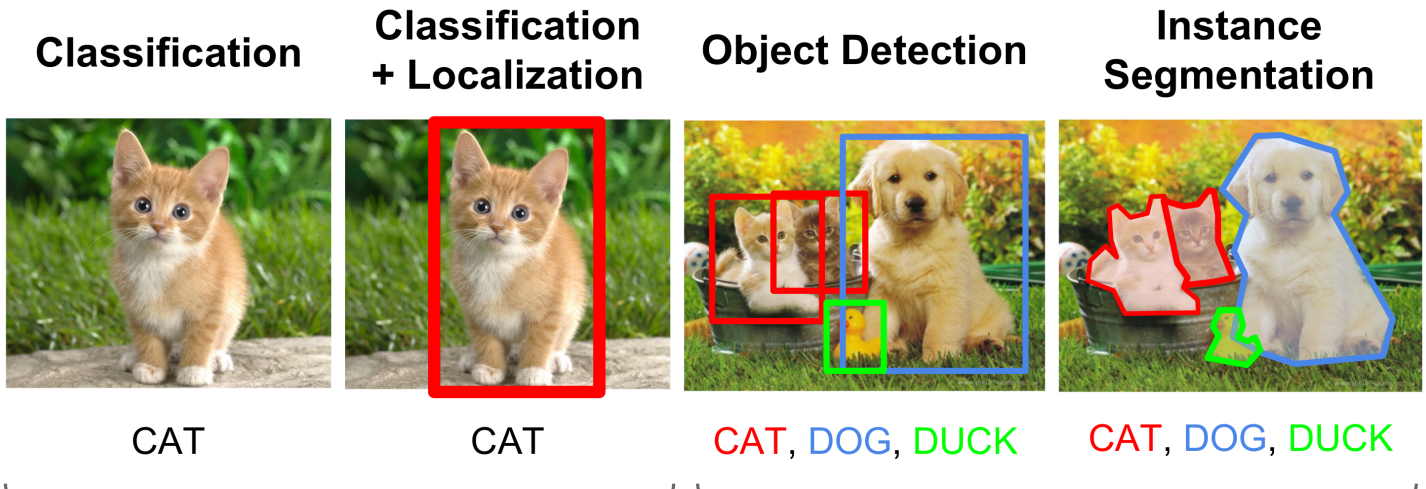
Deeplearning



Deeple

https://goo.gl/images/VA89CC

Deeplearning으로 인해

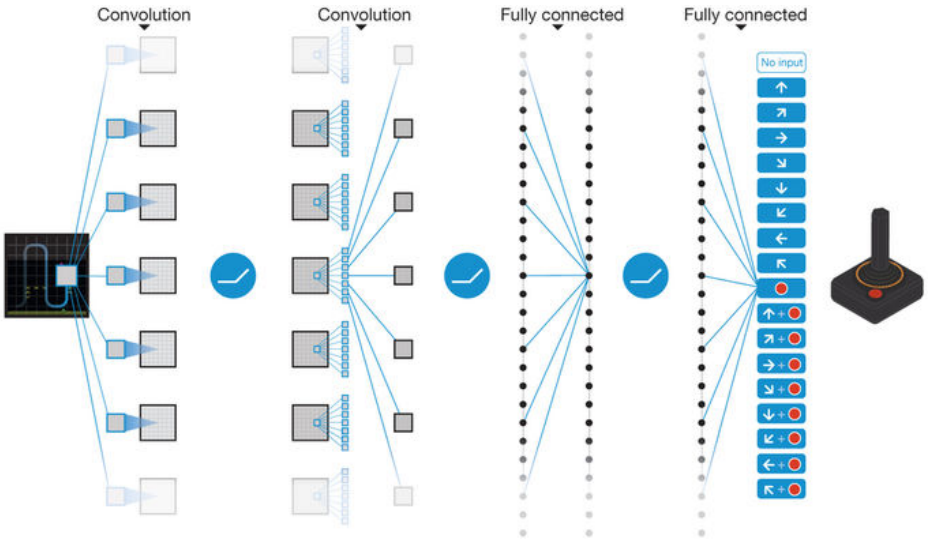


사진을 input으로 받는것이 가능해짐

https://chaosmail.github.io/deeplearning/2016/10/22/intro-to-deep-learning-for-computer-vision/

Deeplearning+Reinforcement Learning

Deep Reinforcement Learning



https://goo.gl/images/oNu5Gr

Deepmind, DQN

Deeplearning을 강화학습에 적용하여, 사람보다 플레이를 잘하는 인공지능을 만듬

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

DQN, Keras(breakout)

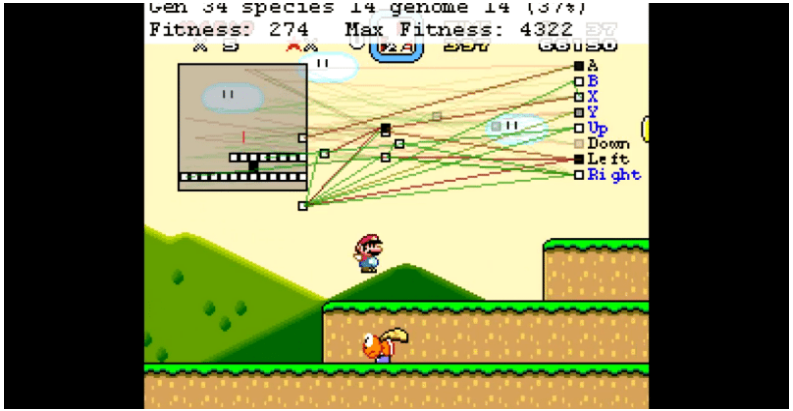
시연 + 코드설명

슈퍼마리오 환경 구축

Deeplearning+Reinforcement Learning

Human

A.I.



다른 도메인에서도 적용이 가능하다!

여러 환경에 사용

하지만 환경에 따른 적절한 강화학습 알고리즘을 적용해야 한다.

예민한 강화학습

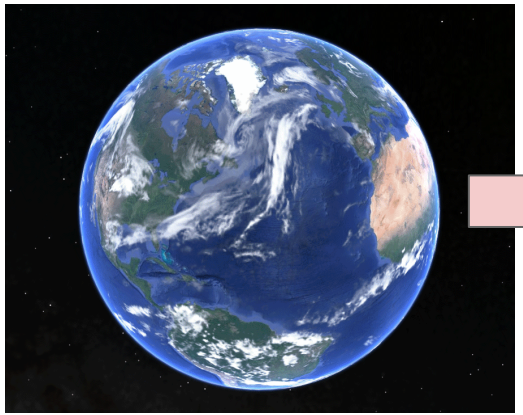
State 크기, action이 다르다.

같은 알고리즘을 사용하더라도, Deeplearning model, hyper parameter을
적절하게 사용해야 한다.

슈퍼마리오 설치에 필요한 네가지

Emulator

Environment



Programming Language

Algorithm 1 Deep Q-learning with Experienc

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with rand s
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and prepr quenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ 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

python NumPy + TensorFlow

Emulator -FCUX

Emulator

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

Ubuntu

sudo apt-get update

sudo apt-get install fceux

MAC

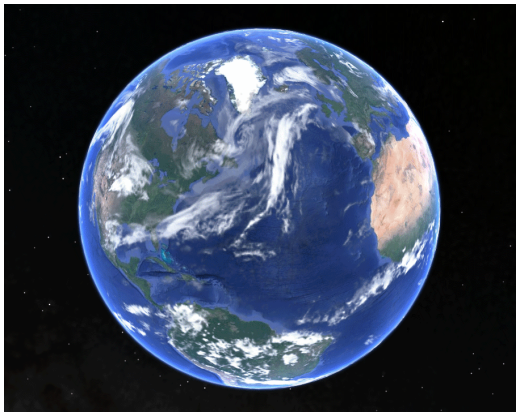
https://brew.sh/ -homebrew website

Terminal open -> brew install fceux

sudo apt-get install fceux

OpenAI_Gym

Environment



OpenAI_Gym

https://github.com/openai/gym

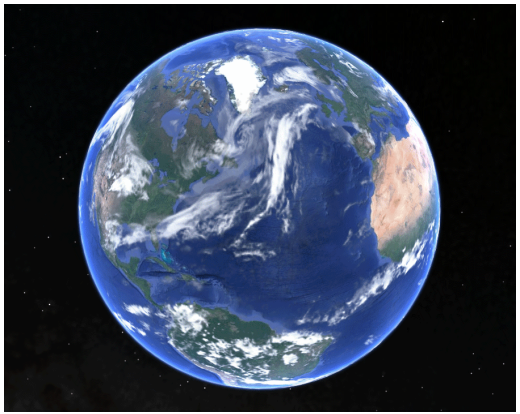
pip3 install gym

```
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

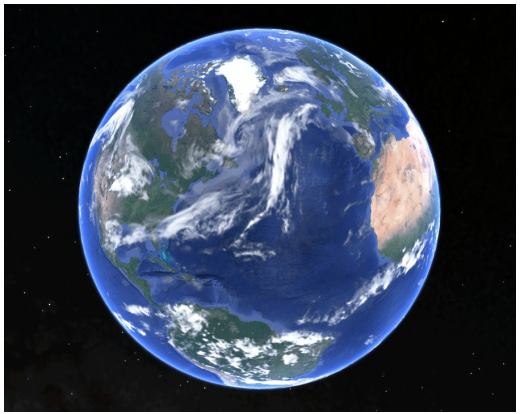
pip3 install baselines

```
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
```

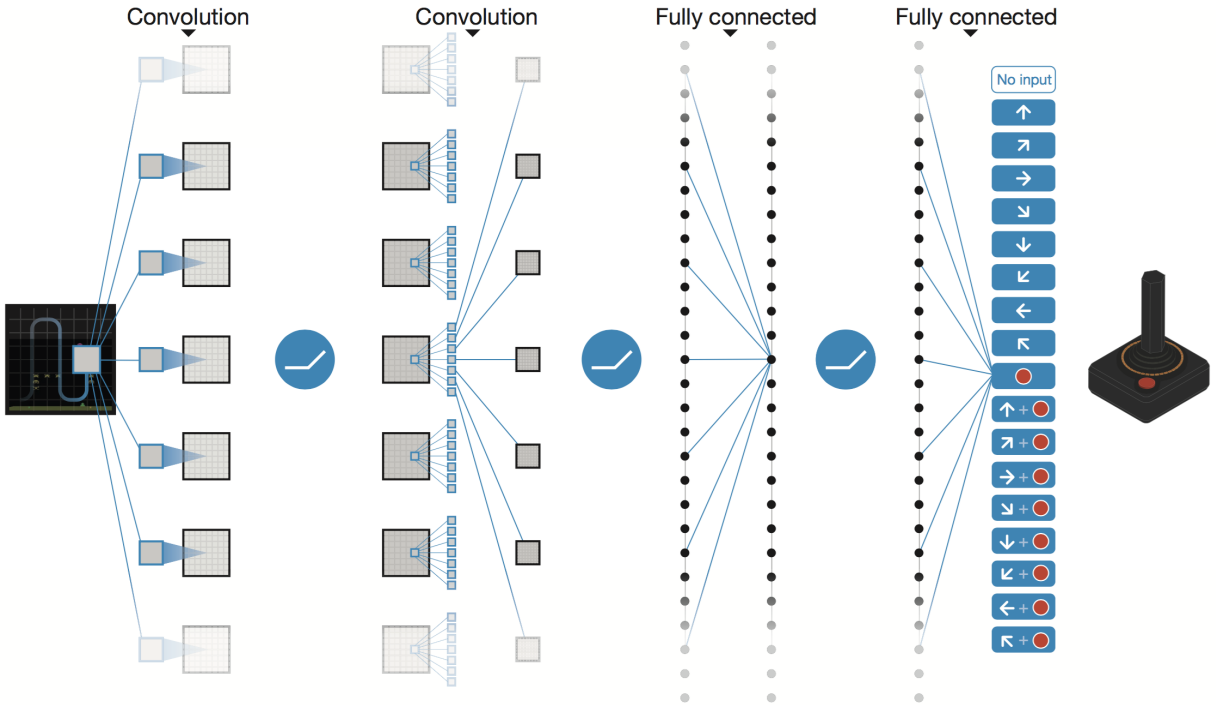
```
import gym
import gym_pull
gym_pull.pull('github.com/ppaquette/gym-super-mario')
env = gym.make('ppaquette/SuperMarioBros-1-1-v0')
```

Algorithm-DQN

Algorithm

DeepMind

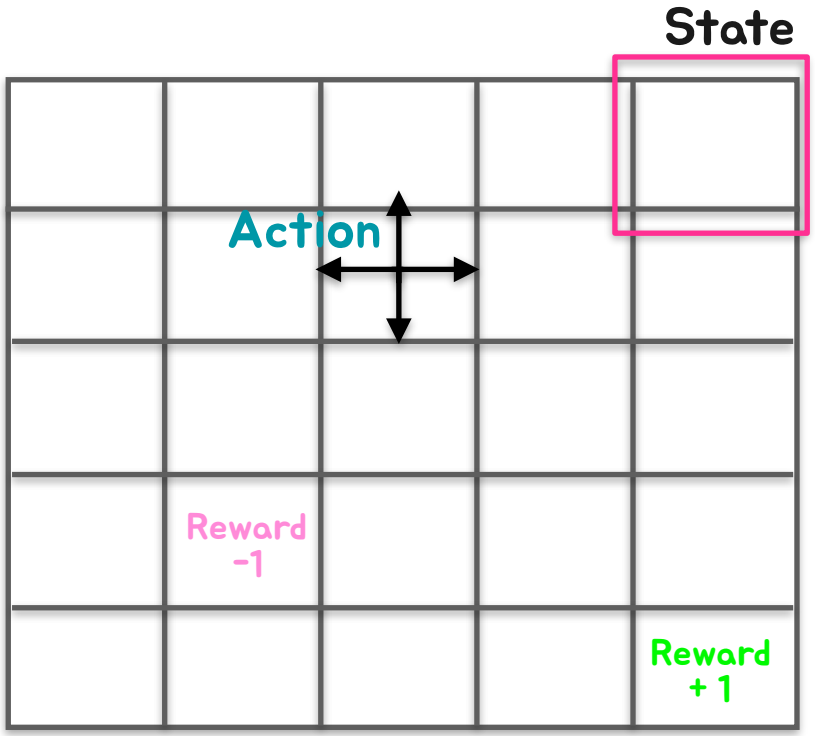
DEEP Q-NETWORK



DQN을 이용한 인공지능 슈퍼마리오 만들기

슈퍼마리오에서의 환경

그리드월드와 어떻게 다를까?



State : 그리드의 좌표

Action : 상, 하 , 좌, 우

Reward : 함정 = -1, 목표 = 1

Transition Probability : 1

Discount factor : 0.9

슈퍼마리오에서의 환경

State

State : 화면

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

Reward : 앞으로 전지할때 Reward +1, 뒤로가면 -1

Transition Probability : 1

Discount factor : Hyper parameter

도착지인 깃발에 가까이 갈수록 높은 reward를 받는다.

DQN을 이용한 인공지능 슈퍼마리오 만들기

시연 + 코드설명

감사합니다.