SAMSUNG SDS
강화학습 실습

실습 순서

1. 실습 환경 소개
2. 실습 과제 소개
3. DQN From Scratch
4. Lunar Lander 환경에서 DQN 실습

성균관대학교 SUNG KYUN KWAN UNIVERSITY

SAMSUNG SDS

실습환경소개

실습환경소개

2. 실습 코드 복사

```
(base) sdssudo@9a1c41aee5ac:~$ ls
Anaconda3-2021.05-Linux-x86_64.sh  Day2  anaconda3  sds01  sds03
Day1                               Day3  etc        sds02  sds04
(base) sdssudo@9a1c41aee5ac:~$ cp -r Day2/ sds01/
(base) sdssudo@9a1c41aee5ac:~$ ls sds01
Day2
```

→ cp -r Day2/ {자신의 id폴더}/

ex) cp -r Day2/ sds01/

실습환경소개

3. 폴더 이동 후 conda 환경 실행

```
(base) sdssudo@9a1c41aee5ac:~$ cd sds01/Day2
(base) sdssudo@9a1c41aee5ac:~/sds01/Day2$ ls
Answer  Template  requirements.txt
(base) sdssudo@9a1c41aee5ac:~/sds01/Day2$ conda activate day02
```

→ cd {자신의 id폴더}/Day2

→ cd conda activate day02

실습환경소개

4. Jupyter Notebook 실행

```
(day02) sdssudo@9a1c41aee5ac:~/sds01/Day2$ jupyter notebook --port 8201
[I 16:19:31.704 NotebookApp] [nb_conda_kernels] enabled, 4 kernels found
[W 2021-08-23 16:19:32.088 LabApp] Config option `kernel_spec_manager_class` not recognized by `LabApp`.
[W 2021-08-23 16:19:32.092 LabApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerA
pp. Be sure to update your config before our next release.
[W 2021-08-23 16:19:32.092 LabApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerA
pp. Be sure to update your config before our next release.
[W 2021-08-23 16:19:32.093 LabApp] Config option `kernel_spec_manager_class` not recognized by `LabApp`.
[W 2021-08-23 16:19:32.103 LabApp] Config option `kernel_spec_manager_class` not recognized by `LabApp`.
[I 2021-08-23 16:19:32.105 LabApp] JupyterLab extension loaded from /home/sdssudo/anaconda3/envs/day02/lib/python3.7/sit
e-packages/jupyterlab
[I 2021-08-23 16:19:32.105 LabApp] JupyterLab application directory is /home/sdssudo/anaconda3/envs/day02/share/jupyter/
lab
[I 16:19:32.111 NotebookApp] Serving notebooks from local directory: /home/sdssudo/sds01/Day2
[I 16:19:32.111 NotebookApp] Jupyter Notebook 6.4.3 is running at:
[I 16:19:32.111 NotebookApp] http://localhost:8201/?token=8d0da2b8349ca839d78ee55630ddc8d379210d027a2aab54
[I 16:19:32.111 NotebookApp]  or http://127.0.0.1:8201/?token=8d0da2b8349ca839d78ee55630ddc8d379210d027a2aab54
[I 16:19:32.111 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 16:19:32.116 NotebookApp] No web browser found: could not locate runnable browser.
[C 16:19:32.116 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/sdssudo/.local/share/jupyter/runtime/nbserver-4558-open.html
    Or copy and paste one of these URLs:
        http://localhost:8201/?token=8d0da2b8349ca839d78ee55630ddc8d379210d027a2aab54
     or http://127.0.0.1:8201/?token=8d0da2b8349ca839d78ee55630ddc8d379210d027a2aab54
```

→ jupyter notebook --port 82{아이디 번호}

ex) jupyter notebook --port 8201

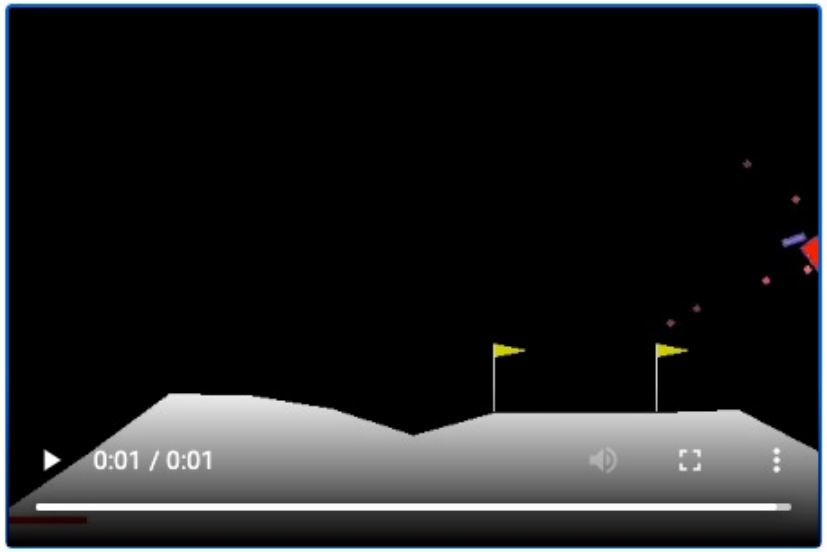
실습환경소개

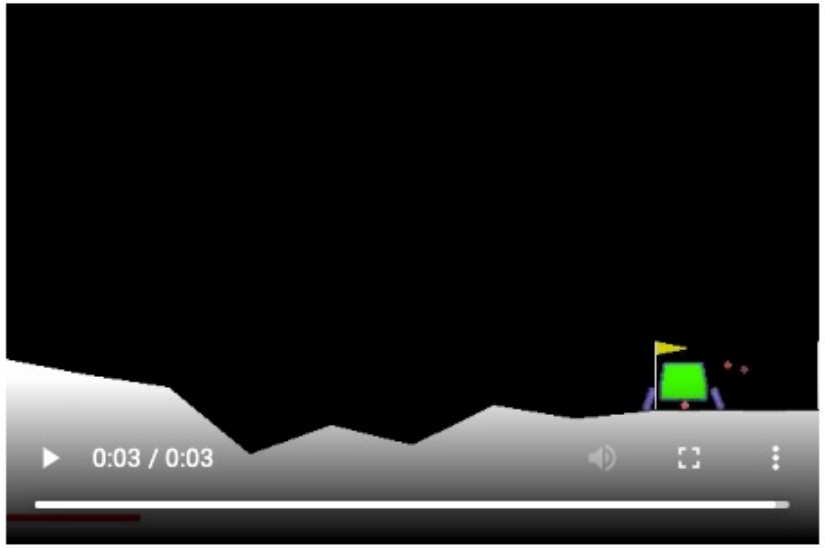
6. 접속 완료!

- 실습에 사용할 코드는 Template 폴더에 있습니다.

실습과제소개

실습과제소개





LunarLander 환경에 Advanced DQN 구현

제출
kim95175@gmail.com

실습과제소개

LunarLander 환경에 Advanced DQN 구현

- double DQN 또는 multistep (또는 둘다) 을 사용하여 150,000 step 학습하여 Best mean reward 50 이상인 log를 캡처하여 보내주세요.

- 자세한 안내는 수업 중 추가로 설명드릴 예정입니다.

제출은 실습 조교 이메일
kim95175@gmail.com


DQN
From
Scratch

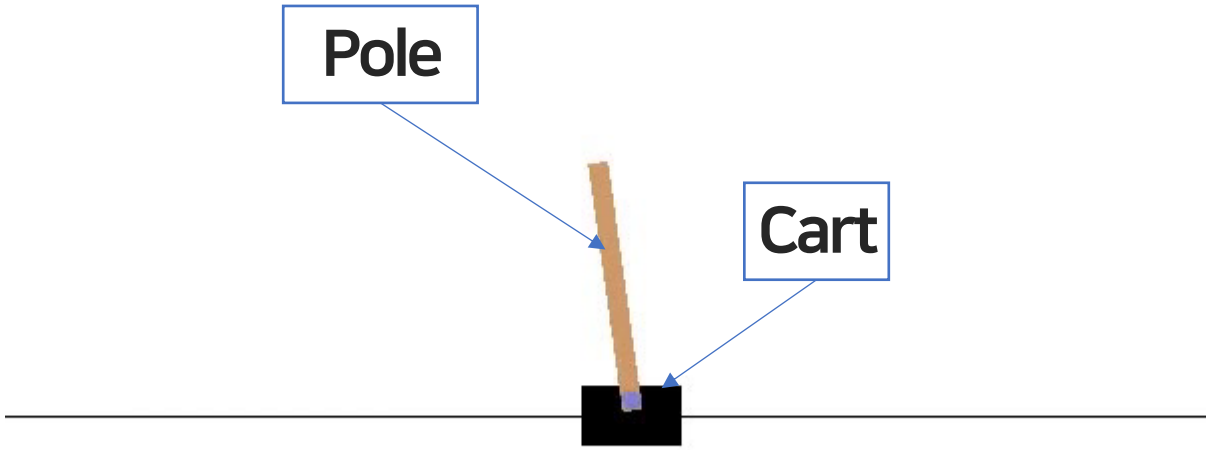
CartPole with DQN

학습 목표 :

- DQN 코드 구현
- Open AI gym CartPole-v0 환경에 적용

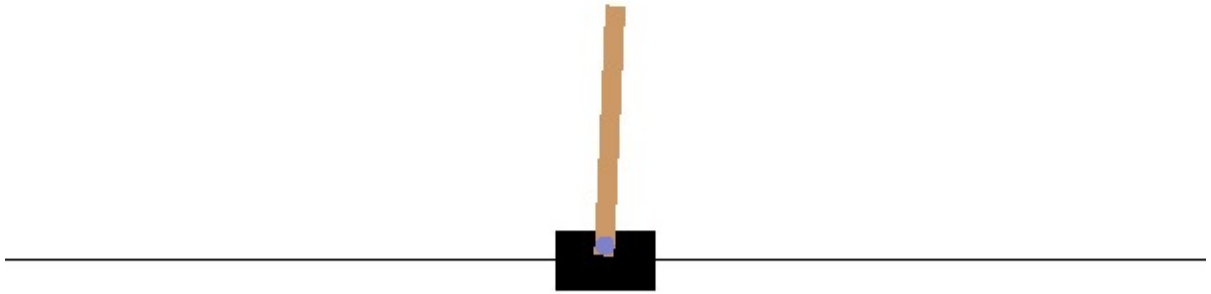
추가 내용 :

- Experience Replay 및 Target network 효과 확인
- Double DQN 및 Multistep 효과 확인



CartPole

- **Cart**를 움직여 **Pole**의 균형을 잡는 문제
- **Cart**를 왼쪽 또는 오른쪽으로 밀 수 있음
- 매 **step** 마다 **+1** 의 **reward**를 받음
- 막대가 수직으로부터 **15**도 기울어지거나,
  화면 끝으로 나가면 종료
- **Goal : Episode reward > 475**



State
= Box(4,)

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

- **4**개의 실수로 구성된 **4-dimensional box space (vector)**
- 각 숫자는 일정한 범위 **(min ~ max)** 사이의 연속된 값의 하나로 결정됨

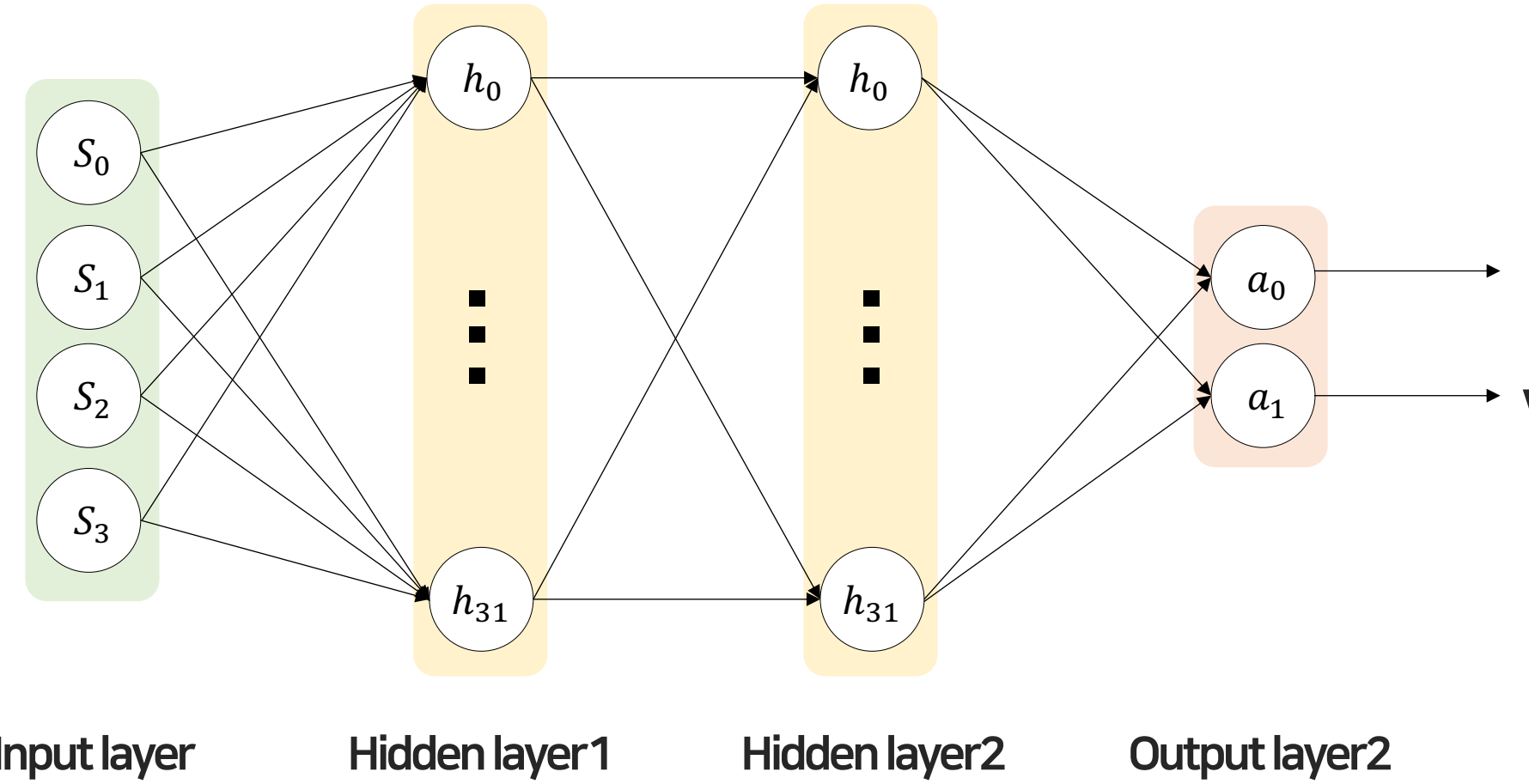
Action
= Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

- **0** 또는 **1** 로 결정되는 이산 **space**

Network Overview

State
= Box(4,)



State – action value(Q)

1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights θ
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
4. **For** episode $= 1, M$ **do**
5. Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6. **For** $t = 1,\mathrm{T}$ **do**
7. With probability ε select a random action a_t
8. otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
9. 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 $\phi_{t+1} = \phi(s_{t+1})$
10. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
11. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12. Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
13. Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
14. Every C steps reset $\hat{Q} = Q$

End For

End For

1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights θ
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
4. **For** episode $= 1, M$ **do**
5. Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6. **For** $t = 1$,T **do**
7. With probability ε select a random action a_t
8. otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
9. 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 $\phi_{t+1} = \phi(s_{t+1})$
10. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
11. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12. Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
13. Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
14. Every C steps reset $\hat{Q} = Q$

End For

End For

0. Initialize replay buffer, network, target network

Replay Memory

- **(s, a, r, s', done)**의 **Transition**을 저장
- 저장된 **Transition**을 랜덤하게 **Sample**
- 여기서 **Multistep** 구현 가능

```python
class ReplayMemory:
    def __init__(
        self,
        observation_shape: tuple = (),
        action_shape: tuple = (),
        buffer_size: int = 50000,
        num_steps: int = 1,
    ):
        self.observation_shape = observation_shape
        self.action_shape = action_shape
        self.buffer_size = buffer_size
        self.num_steps = num_steps


    def write(self, state, action, reward, next_state, done):
        pass


    def sample(self, num_samples: int = 1) -> Tuple[np.ndarray]:
        pass

```

Q Network

- **State** s를 입력으로 받아, **Action** a들에 대한 $Q(s,a)$ 값을 출력.
- 위 조건만 만족한다면, 어떤 네트워크라도 상관없음.

```python
class SimpleMLP(nn.Module):
    def __init__(self, input_size, output_size):
        super().__init__()


    def forward(self, x):
        pass
```

1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights θ
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
4. **For** episode $= 1, M$ **do**
5. Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6. **For** $t = 1$,T **do**
7. With probability ε select a random action a_t
8. otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
9. 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 $\phi_{t+1} = \phi(s_{t+1})$
10. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
11. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12. Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
13. Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
14. Every C steps reset $\hat{Q} = Q$

End For

End For

1. Select action using epsilon-greedy

ε - Greedy

- DQN에서 Exploration을 위한 ε-greedy

With probability ε select a random action a_t
otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

```
# Epsilon-Greedy로 Action 추출
epsilon = self.epsilon
self.update_epsilon()

# "7". With probability ε select a random action a_t
if None:
    pass  # 작성해주세요!
# "8". otherwise select a_t = argmax_a Q(φ(s_t), a; ϑ)
else:
    pass  # 작성해주세요!
```

1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights θ
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
4. **For** episode $= 1, M$ **do**
5. Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6. **For** $t = 1$,T **do**
7. With probability ε select a random action a_t
8. otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
9. 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 $\phi_{t+1} = \phi(s_{t+1})$
10. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
11. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12. Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
13. Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
14. Every C steps reset $\hat{Q} = Q$

End For

End For

2. Take step and store transition to replay buffer

환경 Interaction

- **ε-greedy를 통해** 뽑은 **Action**으로 환경과 상호작용 후, **Replay Memory**에 저장.

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 D

```
# "9". Execute action a_t in emulator and observe reward r_t and image x_(t+1)
next_ob, reward, done, info = env.step(action)

# "10". Store transition (φ_t, a_t, r_t, φ_(t + 1)) in D.
self.replay_memory.write(ob, action, reward, next_ob, done)
ob = next_ob  # 직전 State를 다시 저장해줍니다.
```

1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights θ
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
4. **For** episode $= 1, M$ **do**
5. Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6. **For** $t = 1$,T **do**
7. With probability ε select a random action a_t
8. otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
9. 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 $\phi_{t+1} = \phi(s_{t+1})$
10. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
11. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12. Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
13. Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
14. Every C steps reset $\hat{Q} = Q$

End For

End For

3. Train network

1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights θ
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
4. **For** episode $= 1, M$ **do**
5. Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6. **For** $t = 1$,T **do**
7. With probability ε select a random action a_t
8. otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
9. 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 $\phi_{t+1} = \phi(s_{t+1})$
10. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
11. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12. Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
13. Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
14. Every C steps reset $Q = Q$

End For

End For

Target Q network

Online Q network

3. Train network

Q-Network 학습

- Replay Memory에 저장된 Transition을 Random Batch로 뽑아 Q-value 학습.
- Double DQN은 Target Q 계산만 수정함으로써 간단히 구현 가능.

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)\,.$$

Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$

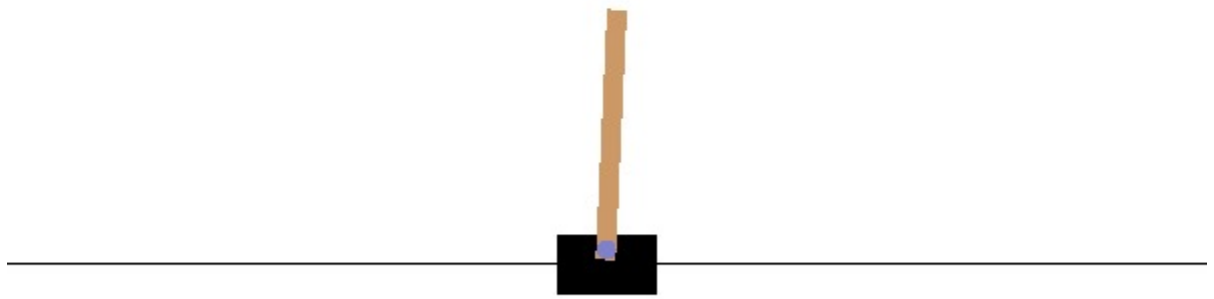
Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

```
# "12". set y_j = {...}
with torch.no_grad():
    if not self.double:  # "12"-1. y_j = r_j + γ * max_
        pass  # 작성해주세요!

    else:  # "Double". y_j = r_j + γ Q_hat(φ_j, argmax_
        pass  # 작성해주세요!


# "13". Perform a gradient step on (y_j - Q(φ_j, a_j; θ
pred = None  # 작성해주세요!
```

실습
DQN Implement



sds_day2_cartpole.ipynb

Lunar Lander with DQN

SAMSUNG SDS

LunarLander with DQN

LunarLander with DQN

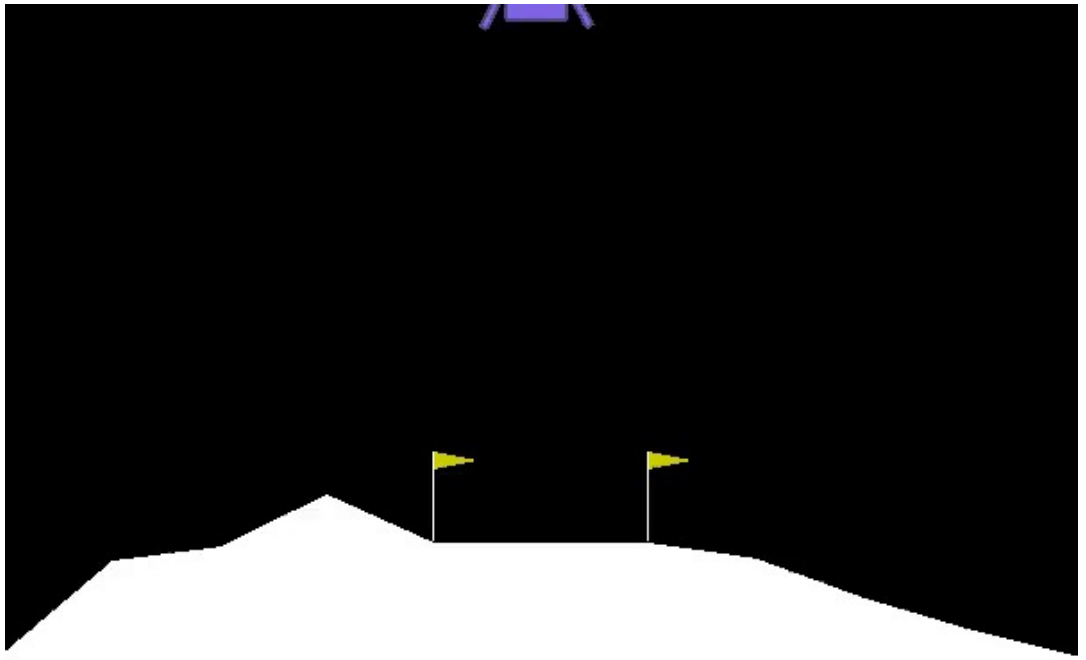
학습 목표 :

- LunarLander 환경에 DQN 코드 적용
- 모듈화된 코드의 구조 파악

LunarLander with DQN

Reward

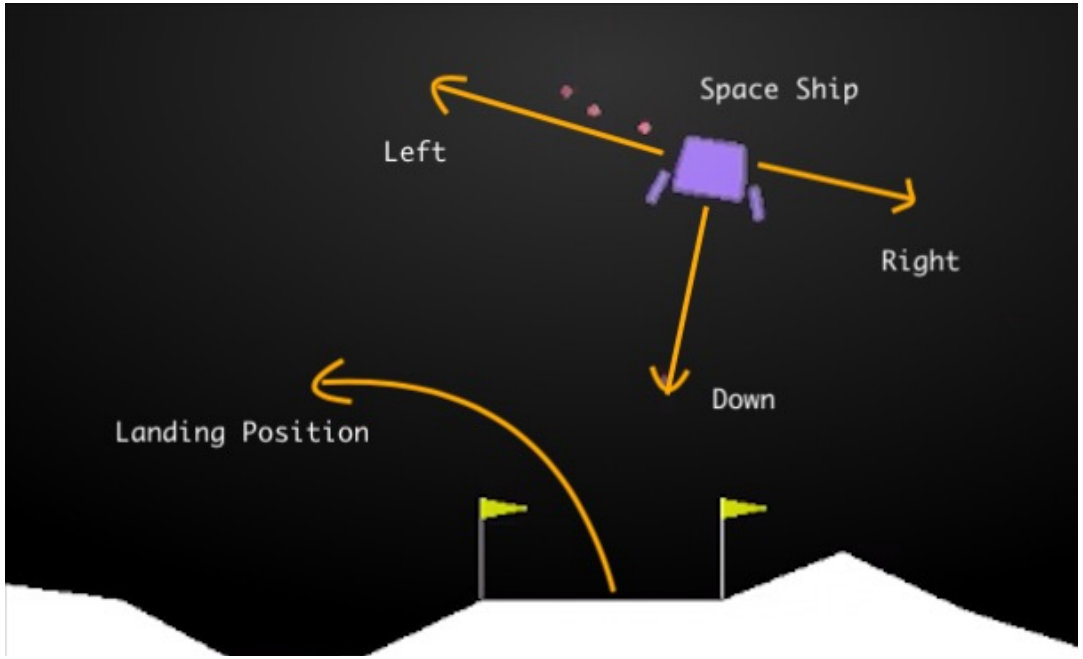
- **Spaceship**을 조정해 정해진 위치에 착륙 시키는 문제
- 지정된 곳에 착륙시 **reward +100~140**
- 착륙**(+100)** 또는 파손**(-100)**시 **episode** 종료
- 땅에 닿은 **leg** 하나 당 **+ 10**
- 엔진 분사 시 **frame** 당 **-0.3**
- **150k timsteps, Reward 50** 이상이면 성공



LunarLander with DQN

LunarLander

- **Spaceship**을 조정해 정해진 위치에 착륙 시키는 문제



State
= Box(9,)
(속도, 각도, leg 착륙여부…)

Action
= Discrete(6)

좌로 회전
회전 안함
우로 회전

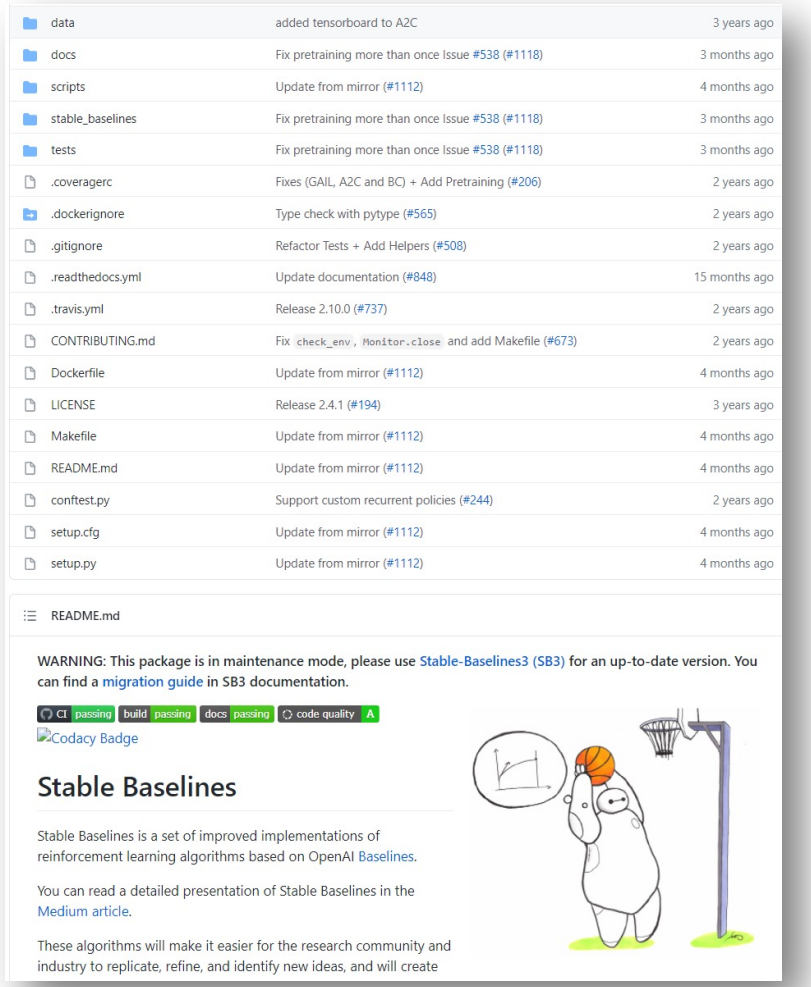
X

엔진 분사
엔진 분사x

실전적인 접근

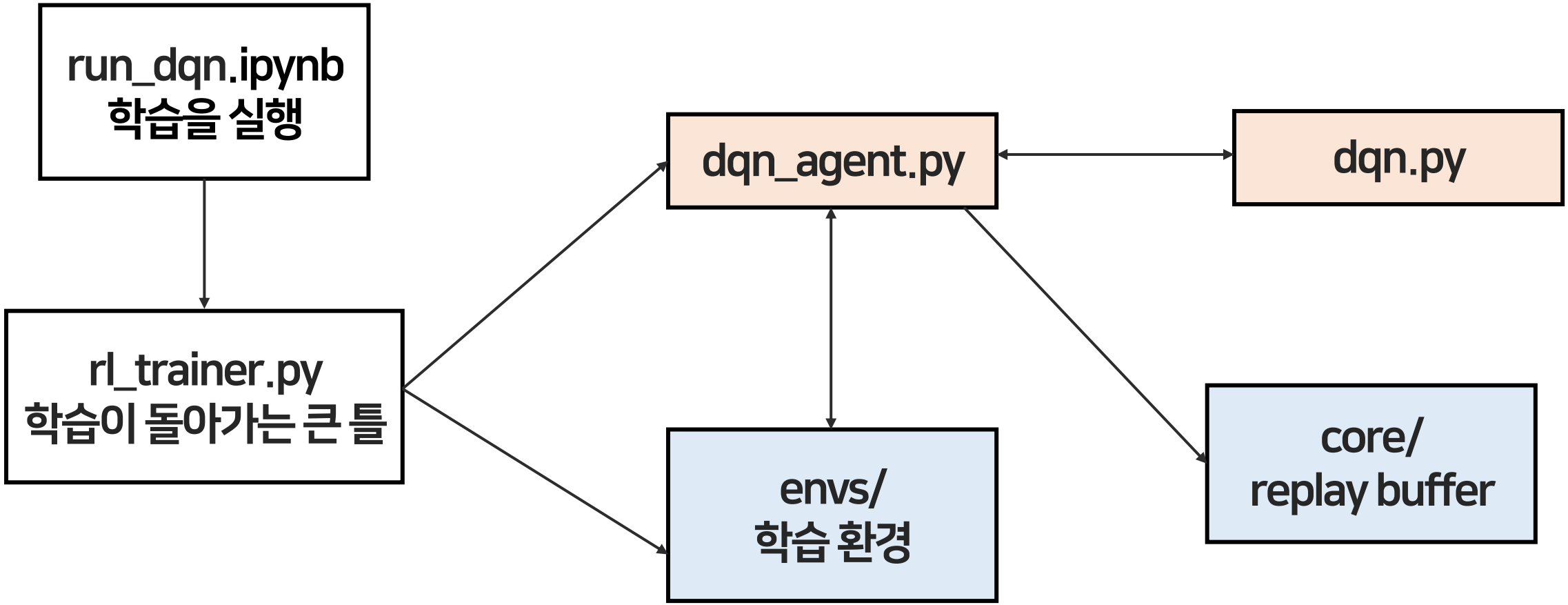
강화 학습 알고리즘을 실제로 적용하고자 할 때, 파일 하나로 구성된 경우는 없다.
=> 모듈화된 코드 사용

코드의 구조를 파악하고 자신의 의도에 맞춰 어느 부분을 수정해야 할지 찾아낼 수 있어야 함.



대표적인 RL 알고리즘 라이브러리 **stable-baseline**

코드 구성

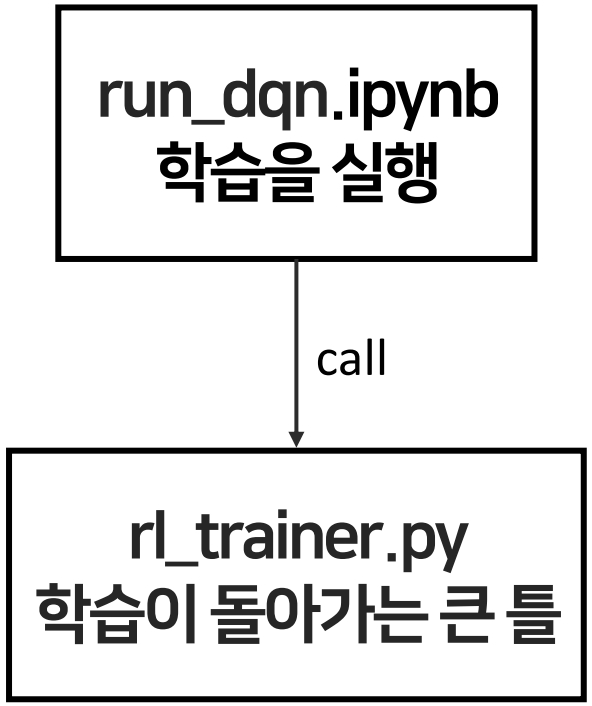

run_dqn.ipynb
학습을 실행
rl_trainer.py
학습이 돌아가는 큰 틀
dqn_agent.py
dqn.py
envs/
학습 환경
core/
replay buffer

코드 구성

run_dqn.ipynb
학습을 실행

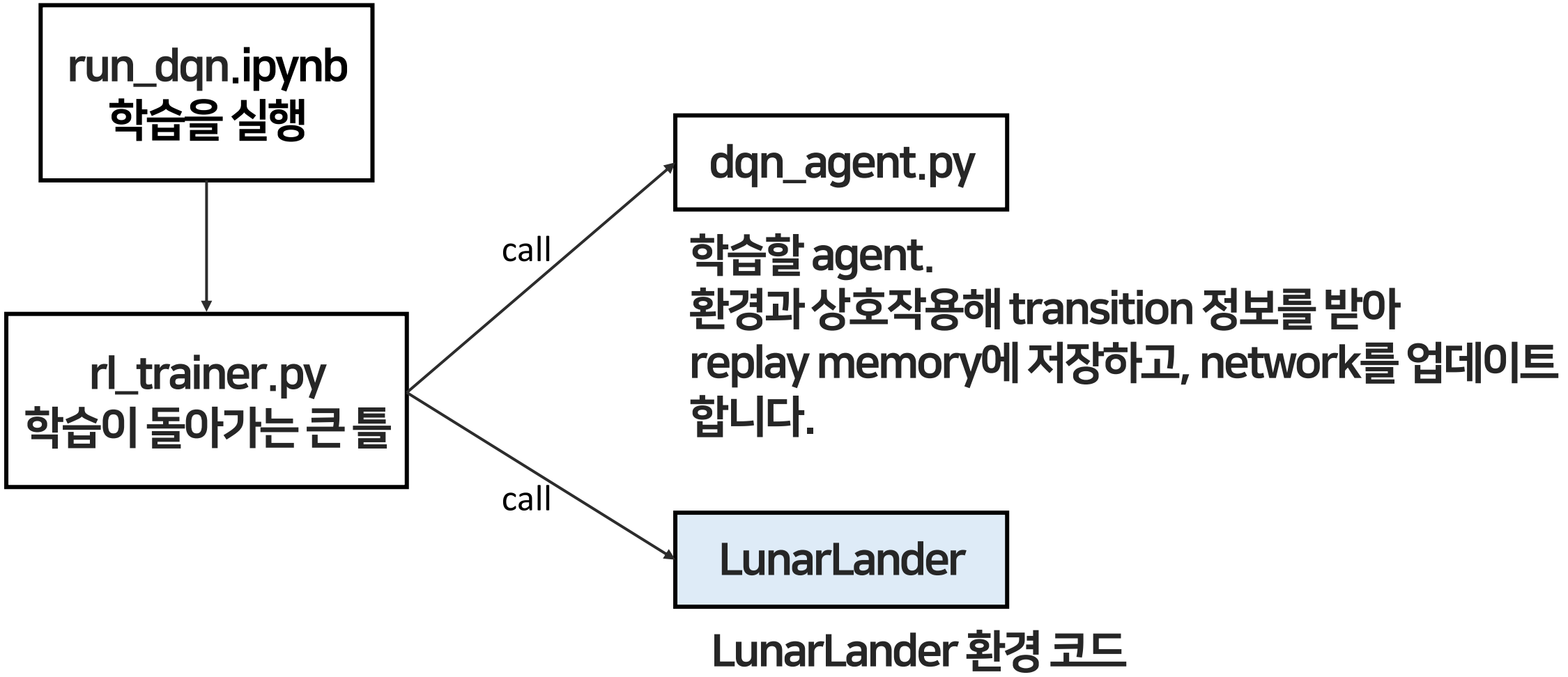
학습을 실행 시키는 코드
학습에 필요한 hyperparameter도 처리합니다.

코드 구성

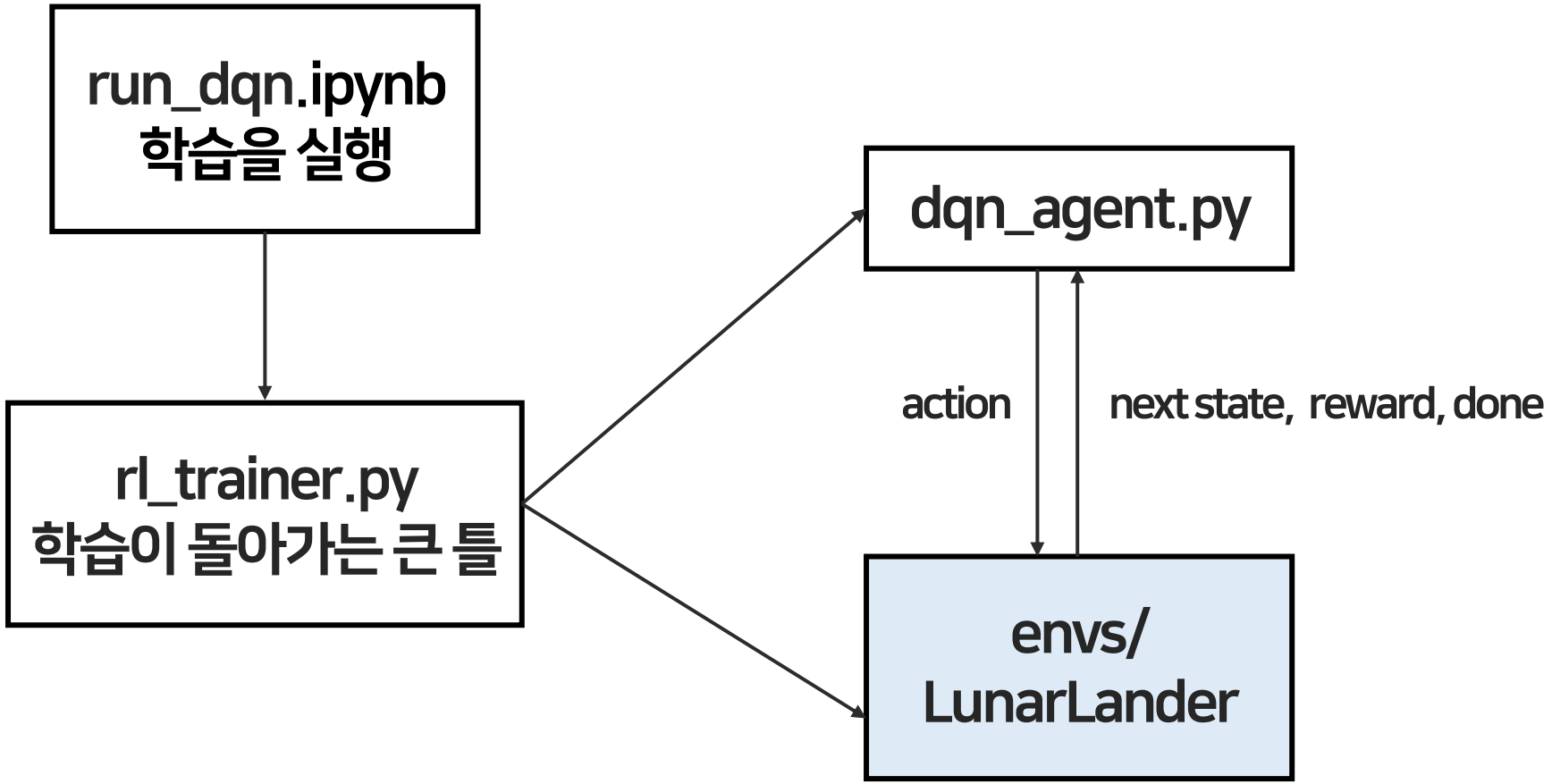


학습이 돌아가는 큰 틀이 되는 코드.
환경, agent 와 상호 작용하며 학습을 진행시킵니다.

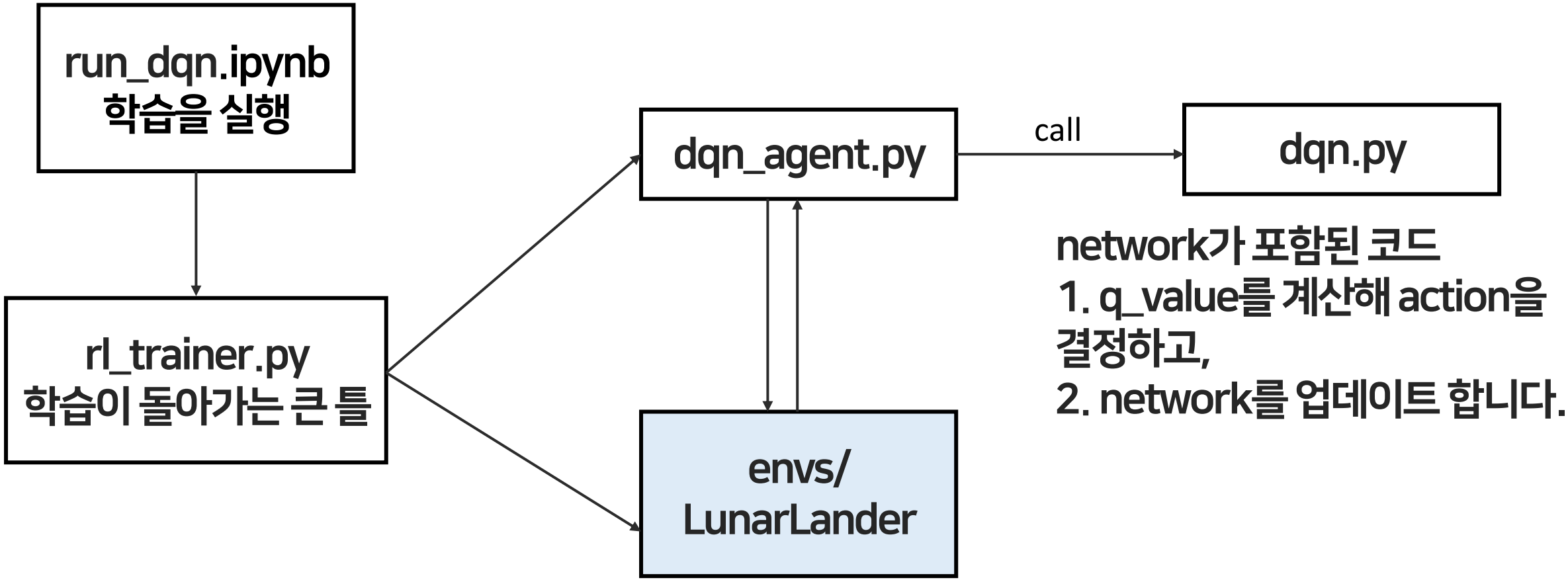
코드 구성


run_dqn.ipynb
학습을 실행
rl_trainer.py
학습이 돌아가는 큰 틀
call
call
dqn_agent.py
학습할 agent.
환경과 상호작용해 transition 정보를 받아
replay memory에 저장하고, network를 업데이트
합니다.
LunarLander
LunarLander 환경 코드

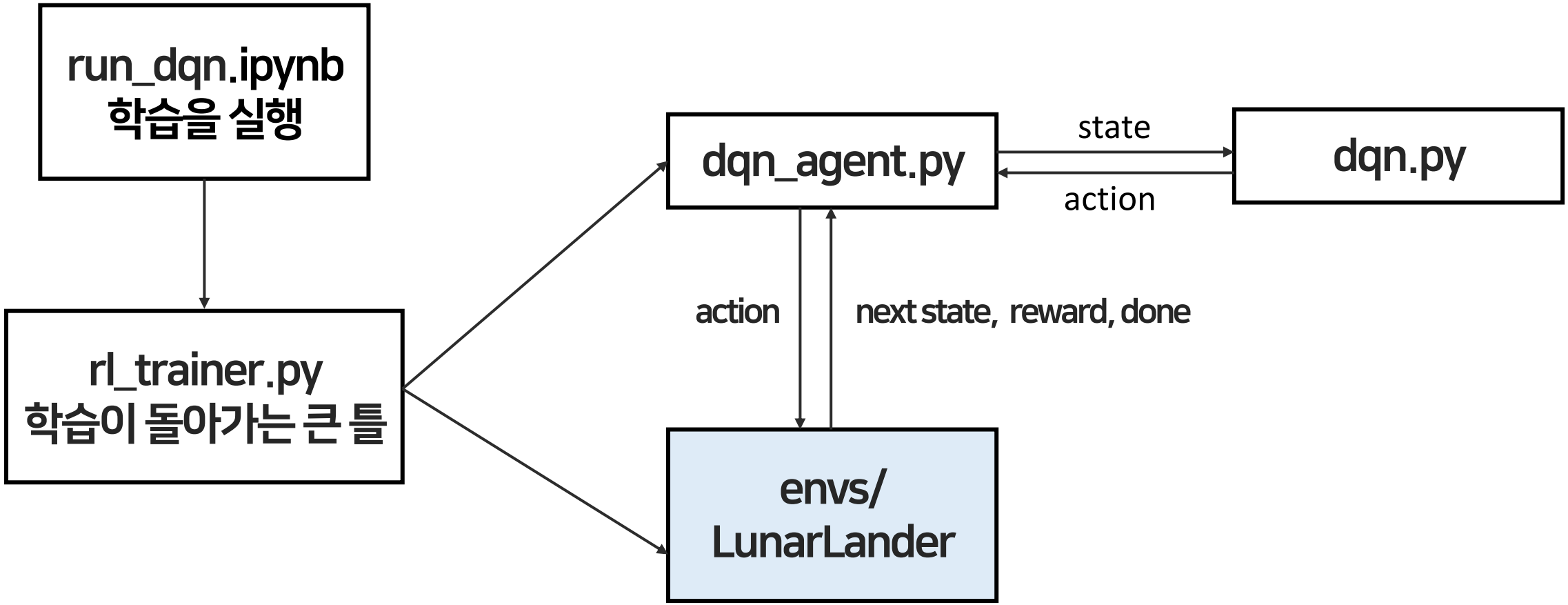
코드 구성


run_dqn.ipynb
학습을 실행
rl_trainer.py
학습이 돌아가는 큰 틀
dqn_agent.py
action
next state, reward, done
envs/
LunarLander

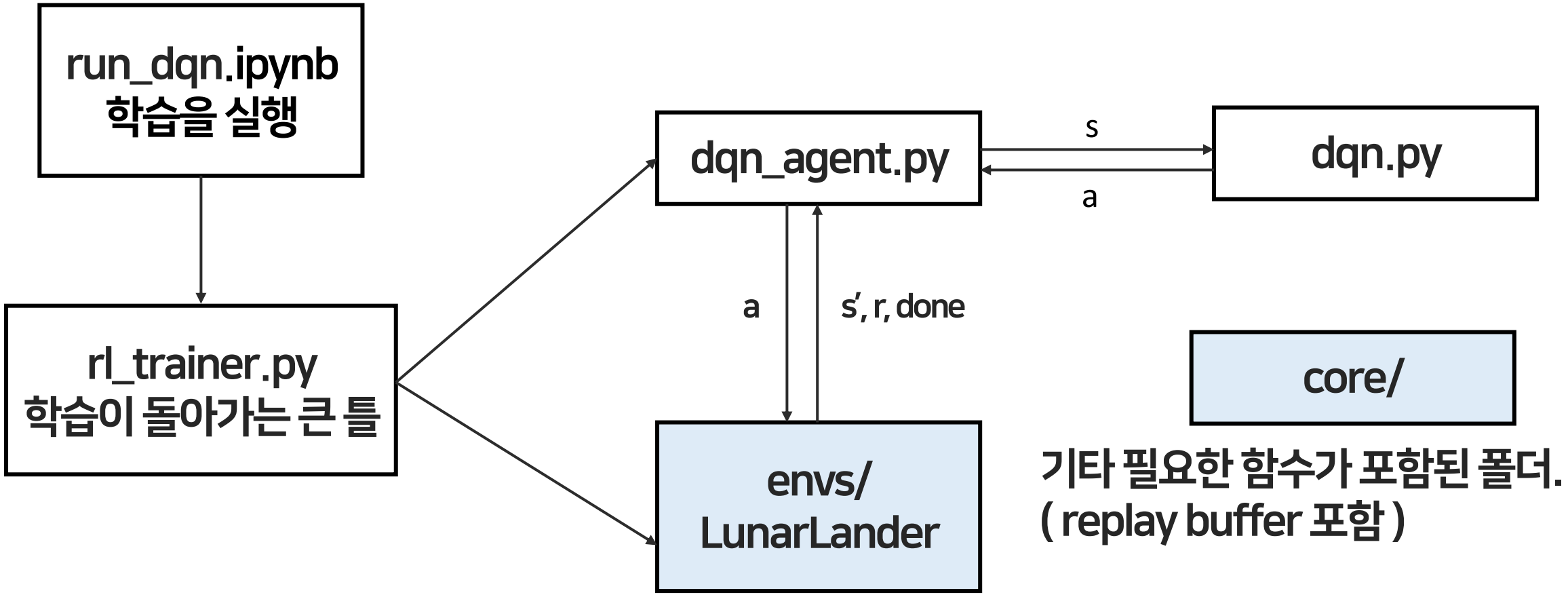
코드 구성


run_dqn.ipynb
학습을 실행
rl_trainer.py
학습이 돌아가는 큰 틀
dqn_agent.py
call
dqn.py
envs/
LunarLander
network가 포함된 코드
1. q_value를 계산해 action을 결정하고,
2. network를 업데이트 합니다.

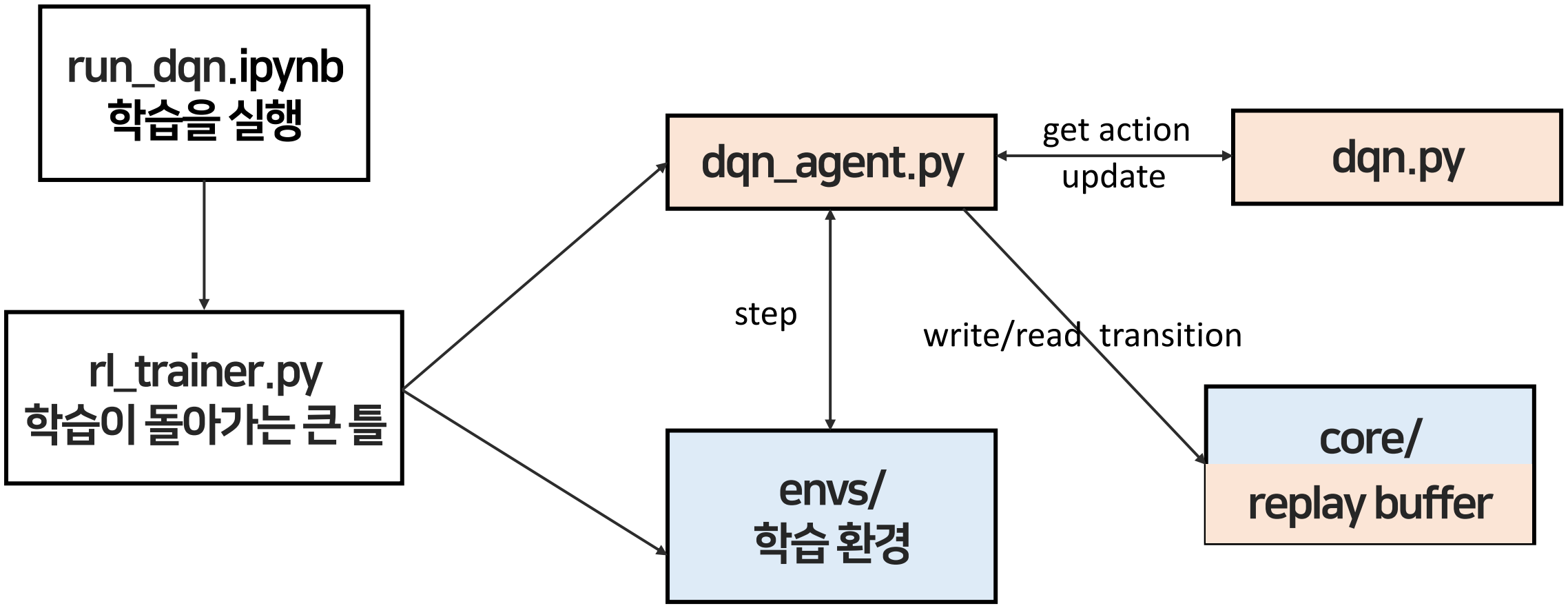
코드 구성


run_dqn.ipynb
학습을 실행
rl_trainer.py
학습이 돌아가는 큰 틀
dqn_agent.py
state
action
dqn.py
action
next state, reward, done
envs/
LunarLander

코드 구성


run_dqn.ipynb
학습을 실행
rl_trainer.py
학습이 돌아가는 큰 틀
dqn_agent.py
s
a
dqn.py
a
s', r, done
envs/
LunarLander
core/
기타 필요한 함수가 포함된 폴더.
(replay buffer 포함)

코드 구성 TODO


run_dqn.ipynb
학습을 실행
rl_trainer.py
학습이 돌아가는 큰 틀
dqn_agent.py
get action
update
dqn.py
step
write/read transition
envs/
학습 환경
core/
replay buffer

실습

Day2/Template/lunarlander/

Double DQN

Double DQN [2016 AAAI]

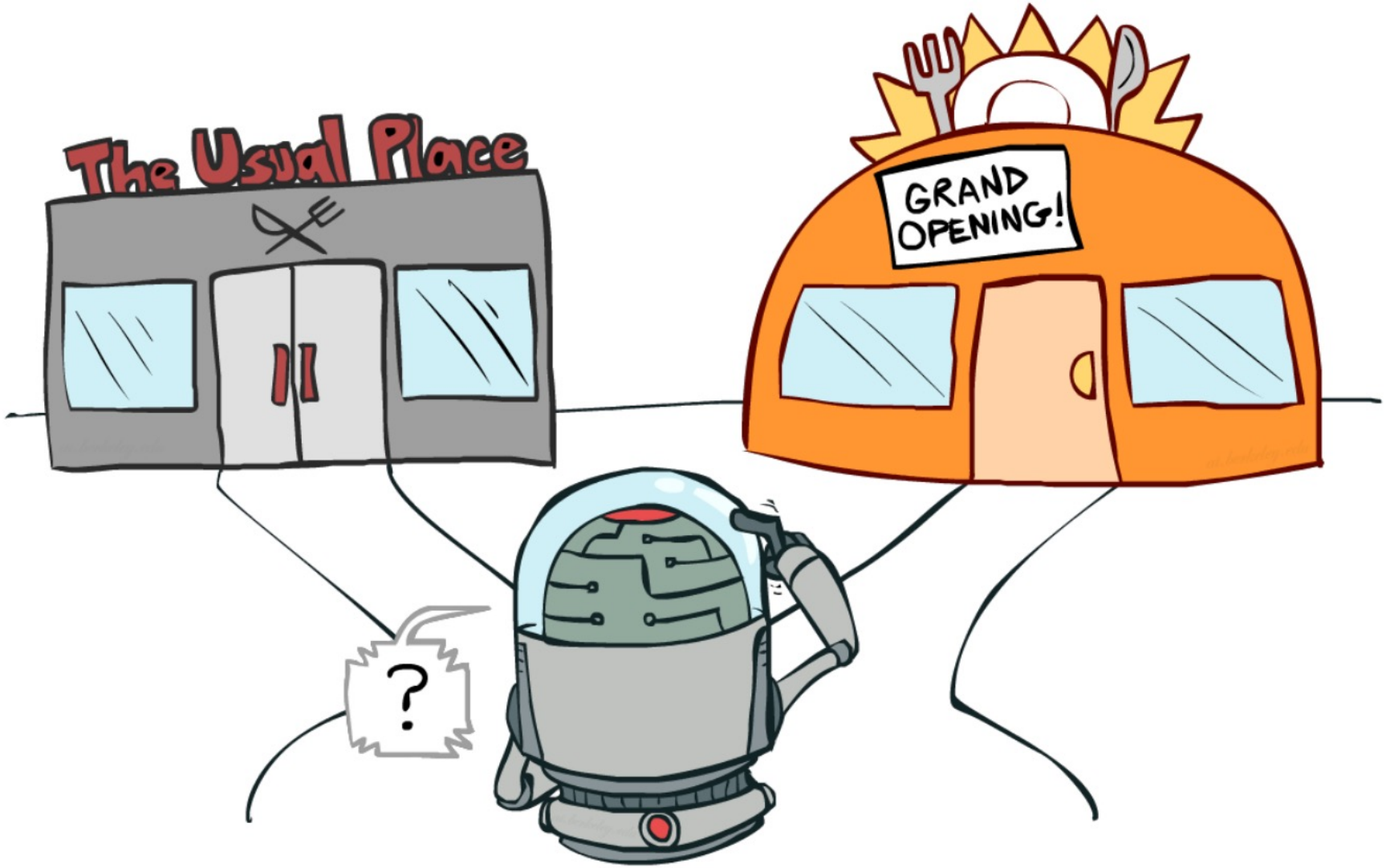
- Extend double Q-leaning idea to DQN
  - Current Q-network w is used to select actions
  - Target Q-network w^- is used to evaluate actions

$$R_{t+1} + \gamma \max_a Q(S_{t+1}, a; w^-) \quad \Rightarrow \text{DQN Target}$$

$$R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax}_a Q(S_{t+1}, a; w^-); w^-) \quad \Rightarrow \text{DQN Target}$$

$$R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax}_a Q(S_{t+1}, a; w); w^-) \quad \Rightarrow \text{Double DQN Target}$$

Day5 : DQN with exploration




Thank You