10주차 강화학습

Q-Table, DQN

해당 문제에 가장 좋은 행동 찾는 방안에 대한 모델?

가장 좋은 행동을 학습할 때 어떻게 할까?

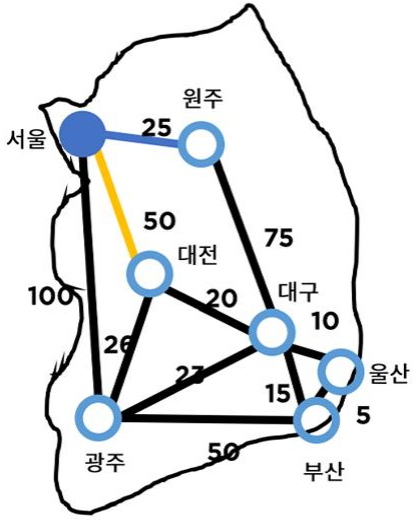
2: 현재의 의사결정이 미래에 영향을 미친다.

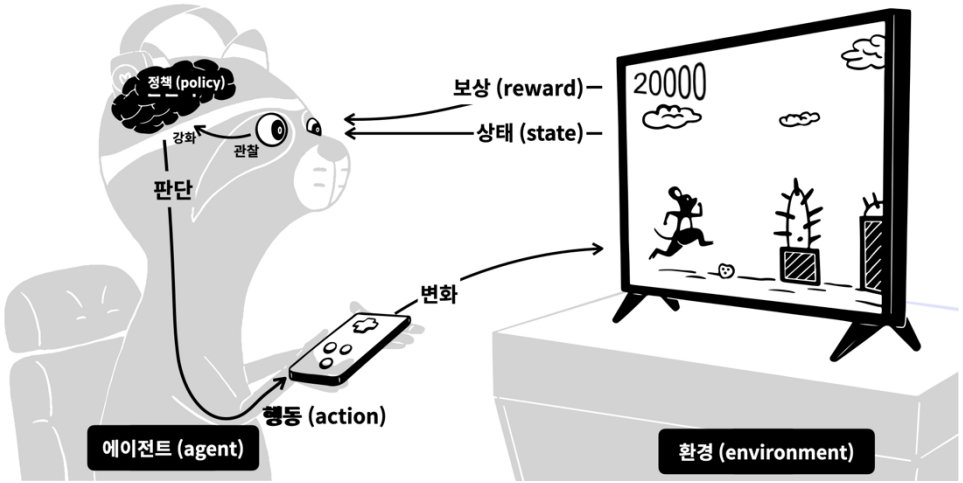
현재 에이전트의 위치 (서울) 에서,
- 선택 1 '원주'

"서울 -> 원주" 거리 = 25
"원주 -> 부산" 최소거리 = 90
"서울 -> 원주 -> ?? -> 부산" 의 최소거리 = 115

- 선택 2 '대전'

"서울 -> 대전" 거리 = 50
"대전 -> 부산" 의 최소거리 = 35
"서울 -> 대전 -> ?? -> 부산" 의 최소거리 = 85



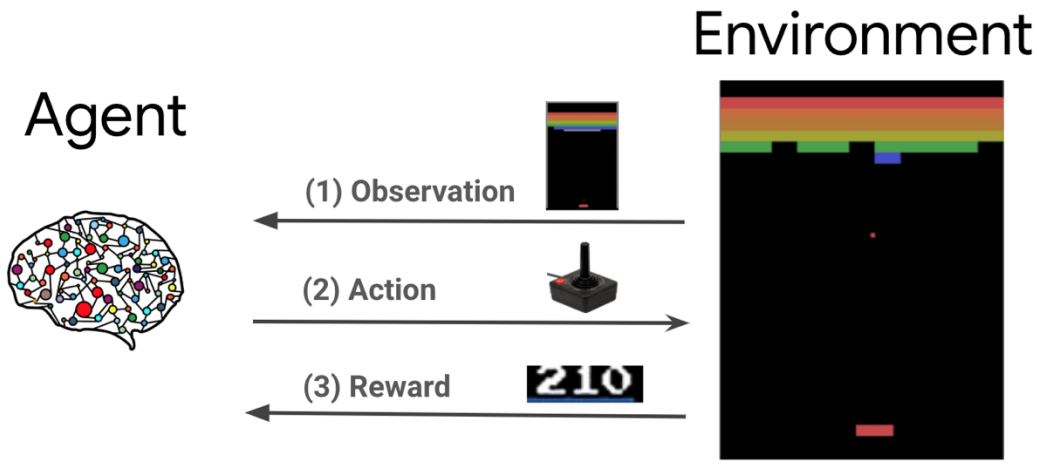


- 주어지는 것은 해당 문제는 정답이 아닌 현재 환경, 상태, 보상으로 이루어짐.
- 좋은 형태는 주어진 문제에 많은 보상을 받기 위해 행동하는 것.
- 이러한 형태를 강화학습이라 함.

강화학습 사례

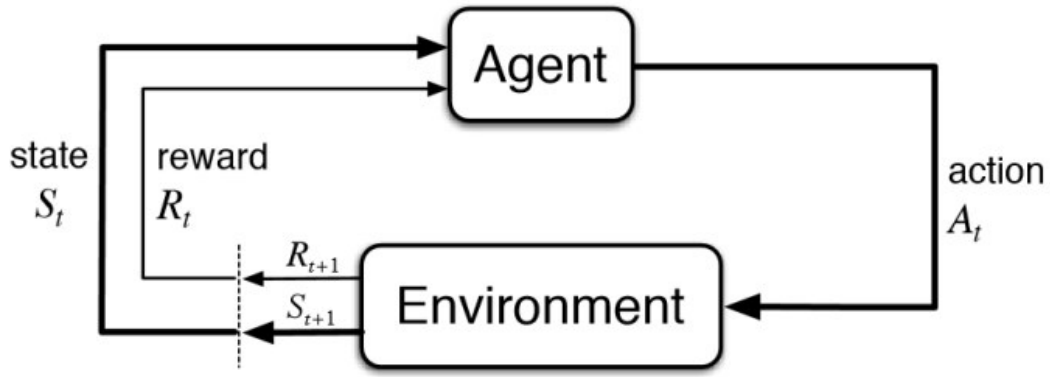
AlphaGo vs. 이세돌

강화학습 방식?



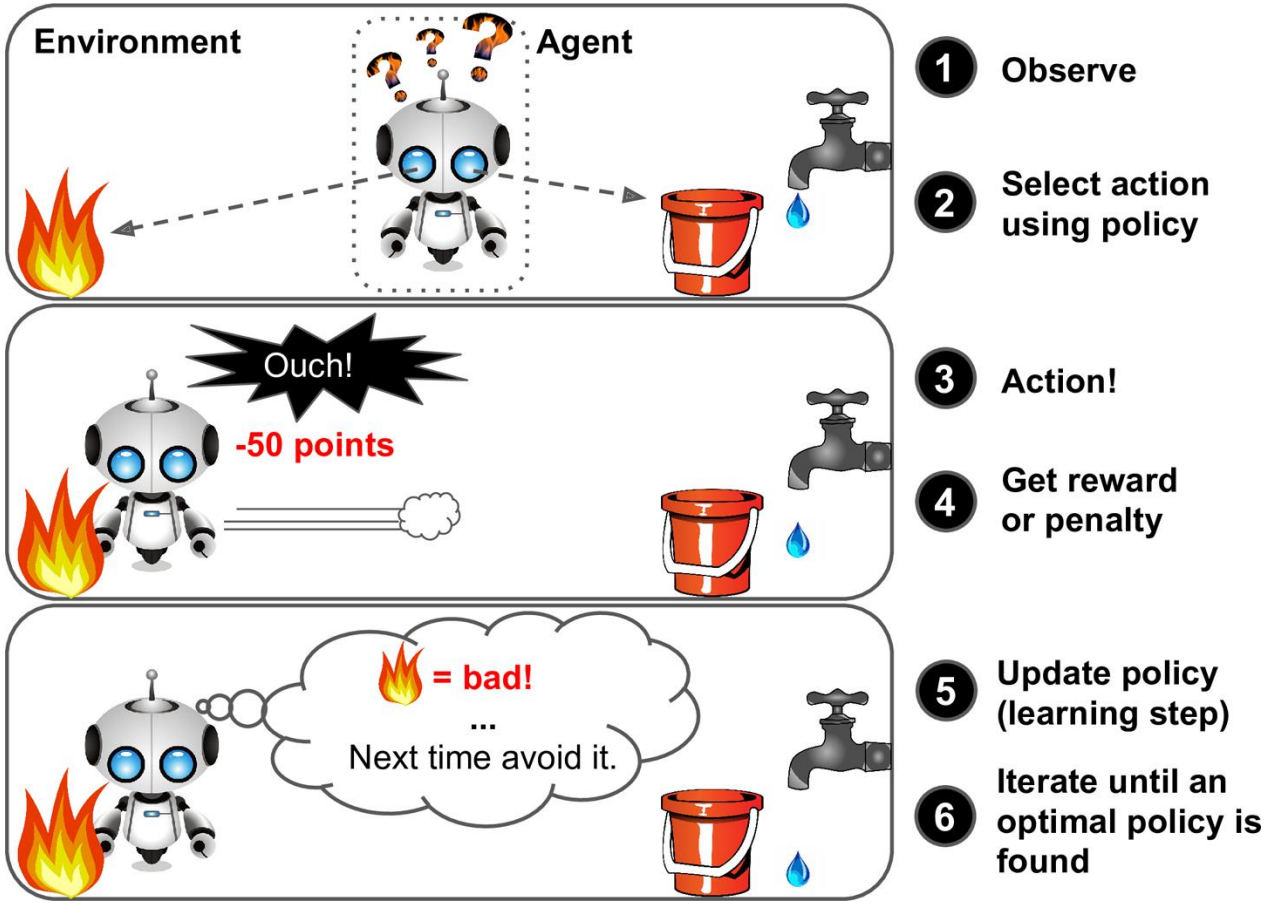
- 주어진 관찰했던 값을 토대로 action함.
- 그 행동에 했던 보상(점수)를 줌.

강화학습 구조



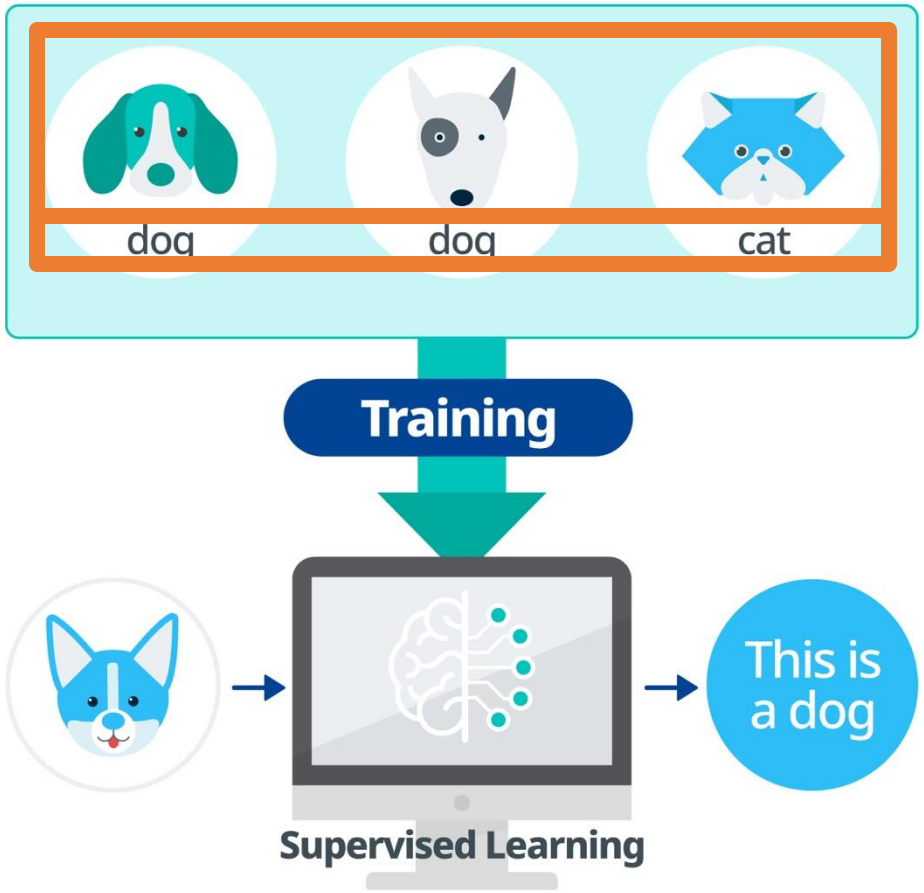
- 주어진 환경을 기반으로 Agent가 행동 취함.
  - 행동을 취하면 다음 환경으로 바뀜.
- 강화학습은 대부분 시계열과 관련이 되어 있음.

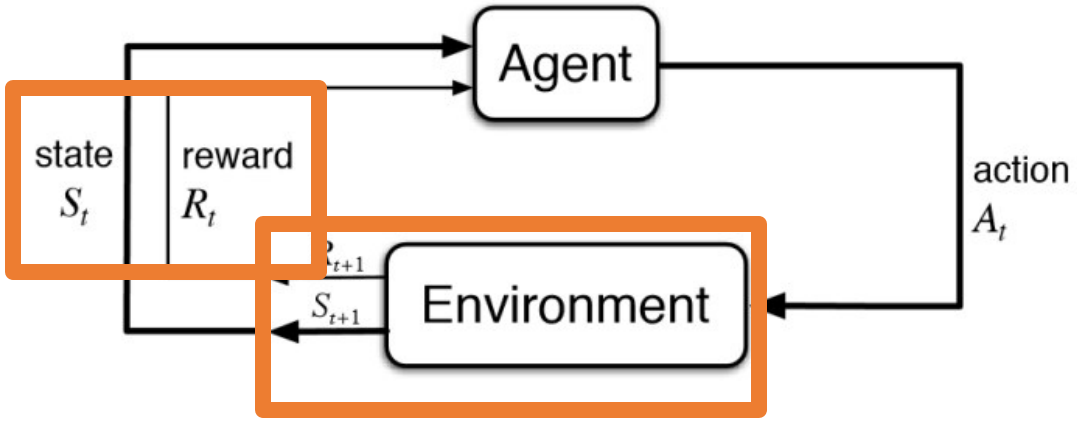
보상(점수) 주는 방법



- 잘 한 경우를 행동 취한 경우, 보상을 줌.
- 잘 못한 경우 보상 안주거나 벌 줌.
- 인공지능이 행동한 것과 행동했을 때 받은 점수를 기반으로 학습.

강화학습과 일반적인 머신러닝과 다른 점





강화학습 초기? (Q Table)

Game Board:

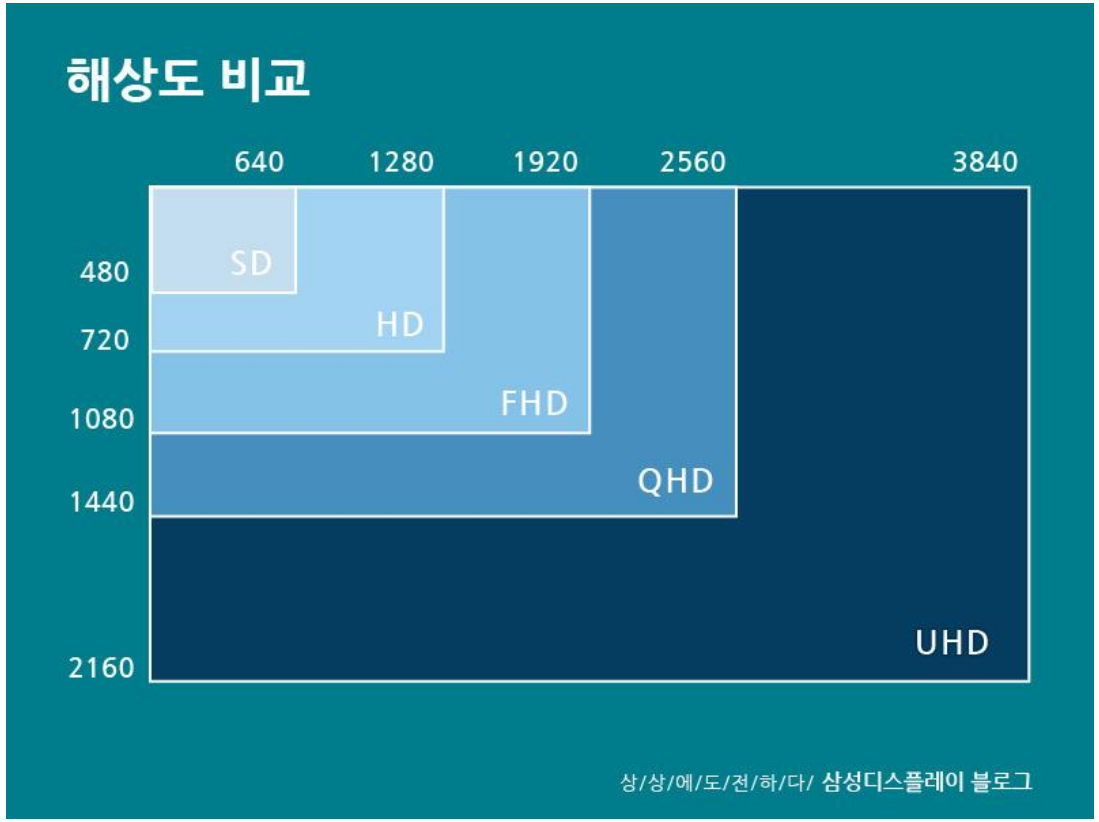
Current state (*s*): 000 010

Q Table: γ = 0.95

	000 100	000 010	000 001	100 000	010 000	001 000
⇧	0.2	0.3	1.0	-0.22	-0.3	0.0
⇩	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
⇨	0.21	0.4	-0.3	0.5	1.0	0.0
⇦	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

- 모델로 구동하는 것이 아닌, 특정 공식으로 이용하여 학습이 되는 형태
- 해당 위치에서 행동할 수 있는 것과 각 행동에 대한 가치(Q)가 가장 높은 것으로 행동
  - 가치가 모두 동일하면 랜덤으로 행동 (학습 초기가 그러한 구조)

Q Table 한계

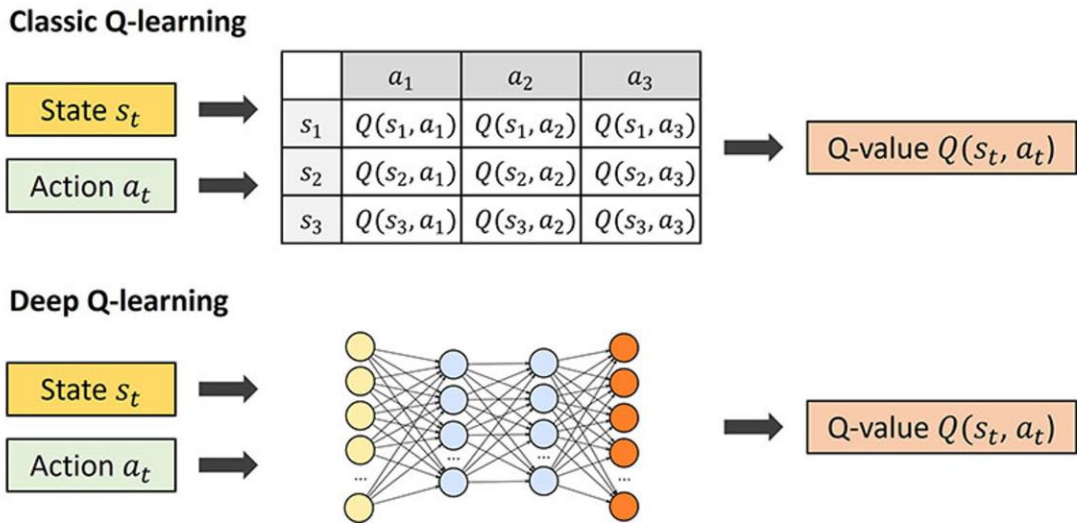


- FHD 해상도 환경에서 학습
- 각 픽셀에 행동할 수 있는 수치가 3개 있다고 가정.
- 모든 행동 할 수 있는 수치
  - 3^(1920(가로)*1080(세로)*3(색채널))

각 맵에 관해서 기록하는 것이 아닌, Agent가 기록하는 것.

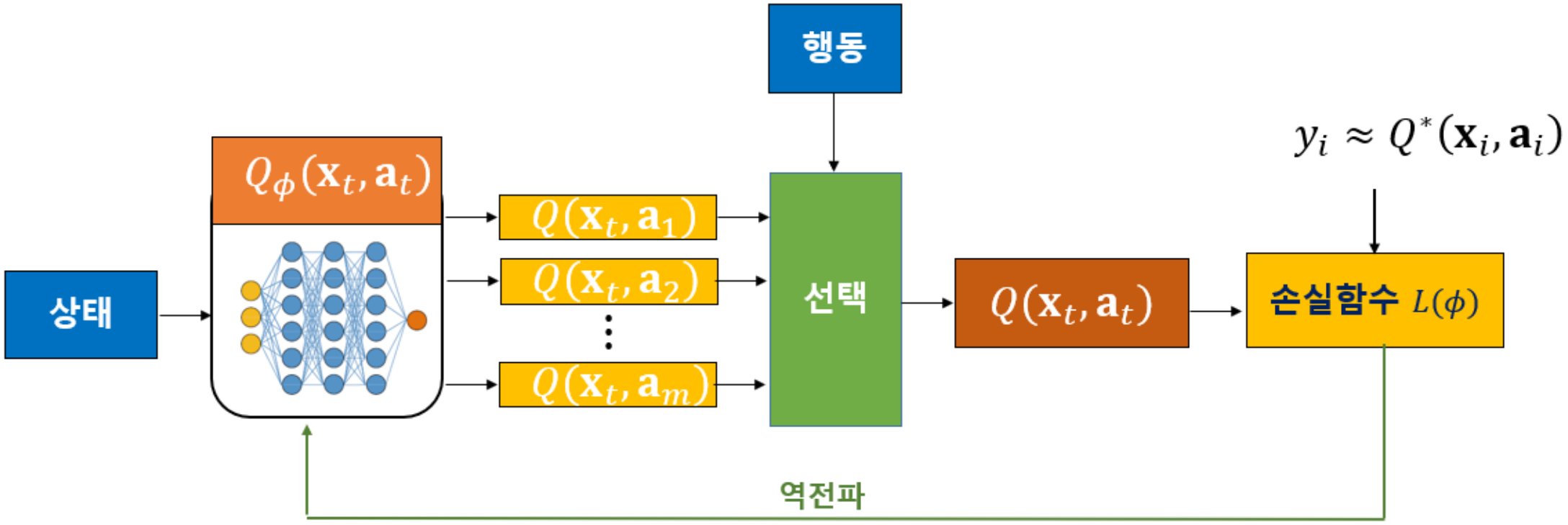
즉, 인공신경망으로 문제를 해결하자.

강화학습의 딥러닝, DQN

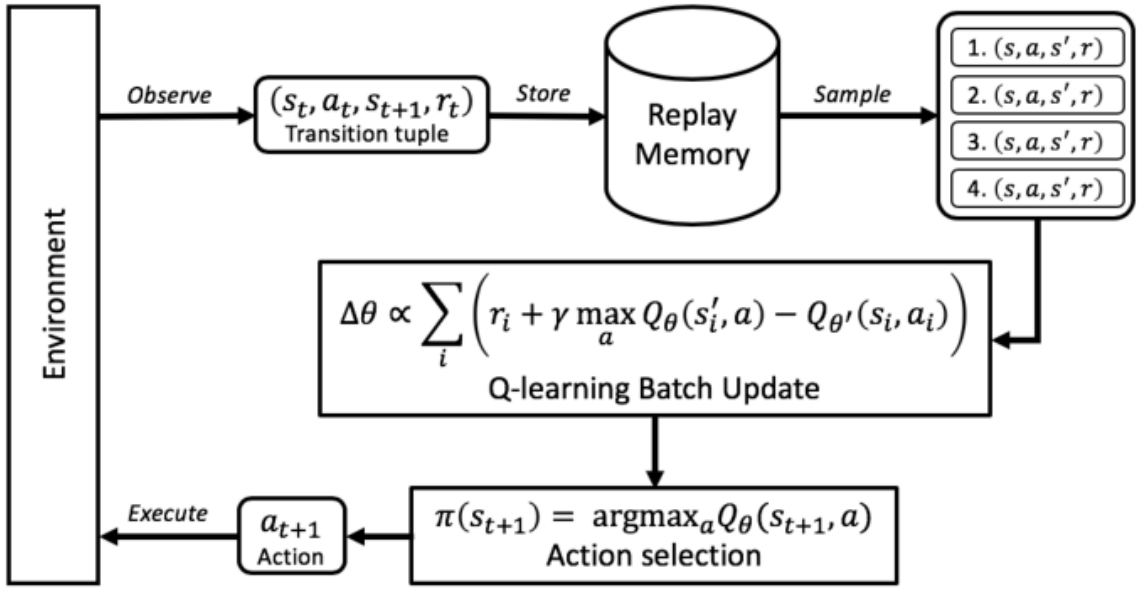


- 각 픽셀에 관해서 기록이 아닌 모델 기반으로 학습
  - 강화학습하는데 필요한 메모리 절감
  - 벨만 방정식을 이용한 손실함수 계산

DQN이 어떻게 구성되고, 행동 선택?

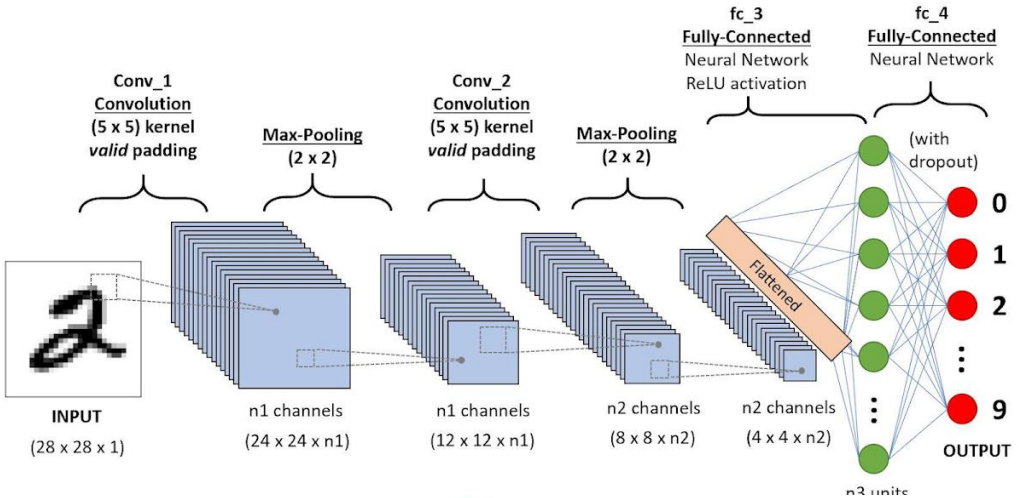


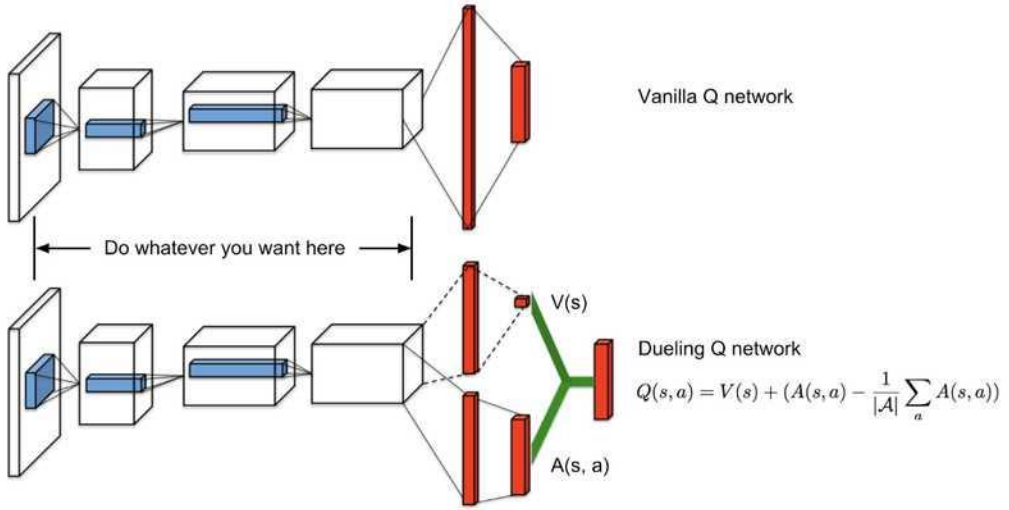
DQN 모델 성능 높이는 방안?



- 학습했던 행동들을 기억하는 것.
  - 학습했던 기억들을 기반으로 모델 학습.
  - 그러한 상태에서 현재 상태를 기반으로 행동 후, 학습.
- Replay Memory를 이용하여 학습을 수월하게 하는 방안 중 한가지.

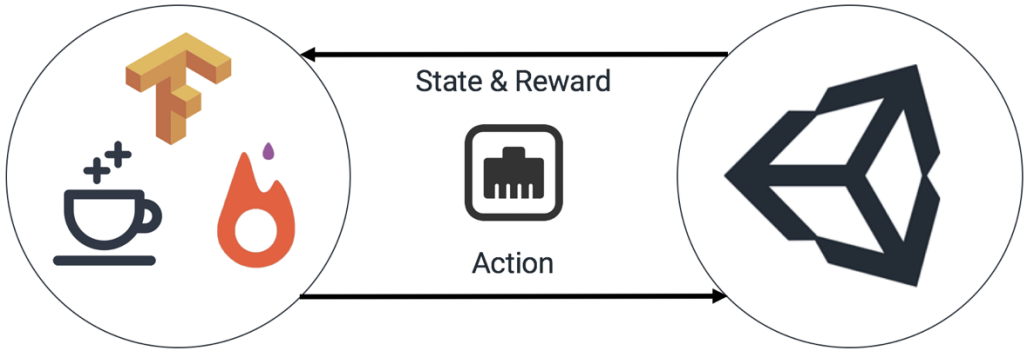
DQN 성능이 좋을까?

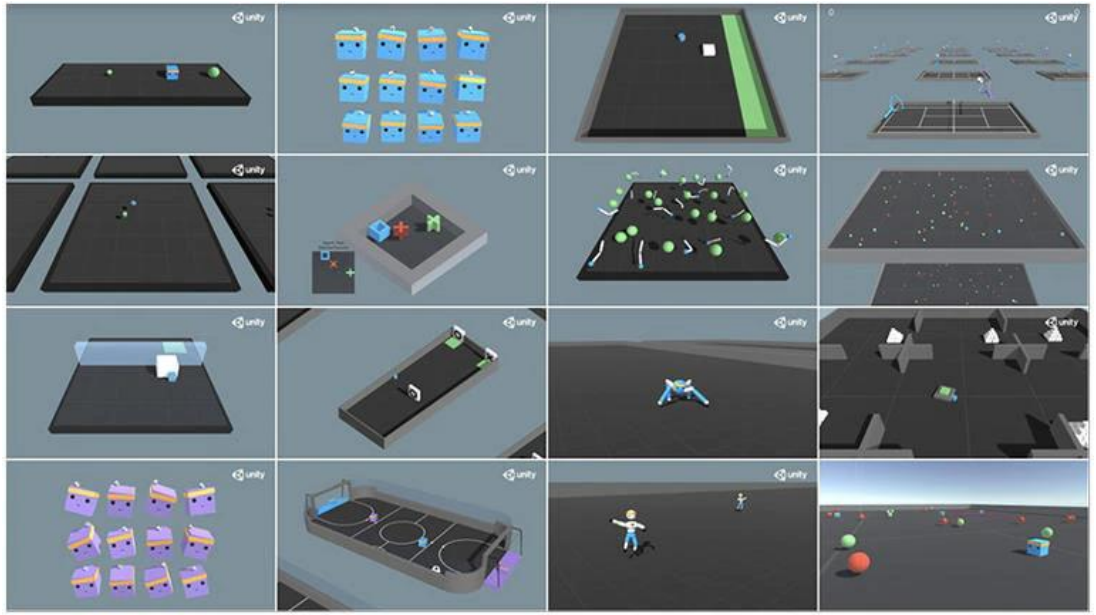




- 문제점이 존재
  - 샘플 상관관계
    - CNN 모델을 토대로 개선
  - 데이터 분포 변화
  - 항상 다른 목표 Y값
    - Double DQN를 이용하여 해결
  - DQN 특화된 모델(Dueling DQN)

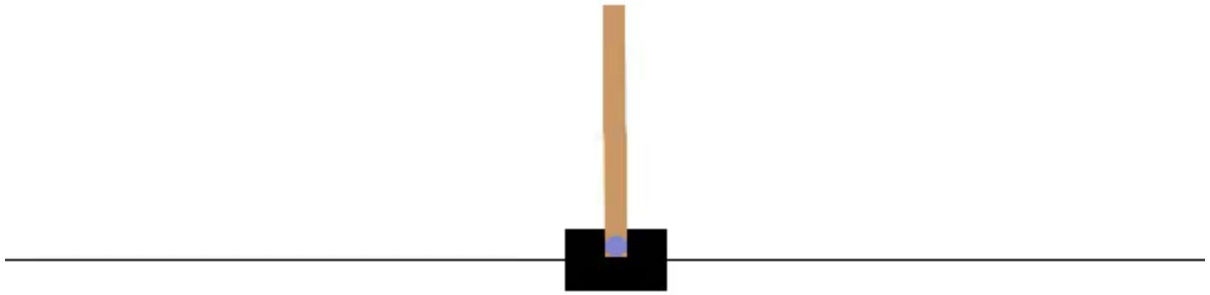
실제로 강화학습 만들고 싶다?





- 게임엔진과 Python 딥러닝 라이브러리와 연결해주는 것과, 딥러닝 라이브러리 통신으로 다른 프로그램 사이에 모델 학습 가능.

실습 : openAIGym - chartpole



실행 결과출력 & gym 설치

```
!apt-get install -y xvfb x11-utils
!pip install PyOpenGL
!pip install PyOpenGL-accelerate
!pip install gym[all]==0.17.*
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
!pip install colabgymrender
!pip install pyvirtualdisplay
!pip install gymnasium
```

필요한 라이브러리 import

- import gymnasium as gym
- import math
- import random
- import matplotlib
- import matplotlib.pyplot as plt
- from collections import namedtuple, deque
- from itertools import count

- import torch # pytorch
- import torch.nn as nn # 모델 & 레이어
- Import torch.optim as optim # 최적화
- Import torch.nn.functional as F # loss function

Gym 불러오기 & 화면 출력 설정

```
env = gym.make("CartPole-v1")
# matplotlib 설정
is_ipython = 'inline' in matplotlib.get_backend()
if is_ipython:
    from IPython import display

plt.ion()
# GPU를 사용할 경우
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

ReplayMemory 클래스 구현

```
Transition = namedtuple('Transition',
                        ('state', 'action', 'next_state', 'reward'))
class ReplayMemory(object):
    def __init__(self, capacity):
        self.memory = deque([], maxlen=capacity)

    def push(self, *args):
        """transition 저장"""
        self.memory.append(Transition(*args))

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

DQN 모델 작성(초기), Pytorch 예시

```
class DQN(nn.Module):

    def __init__(self, n_observations, n_actions):
        super(DQN, self).__init__()
        self.layer1 = nn.Linear(n_observations, 128)
        self.layer2 = nn.Linear(128, 128)
        self.layer3 = nn.Linear(128, n_actions)

    # 현재 관찰 수치를 보고 어떤 것으로 행동을 취할 것인지 계산한다.
    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        return self.layer3(x)
```

모델 학습 파라미터 1

```
# BATCH_SIZE는 리플레이 버퍼에서 샘플링된 트랜지션의 수입니다.
# GAMMA는 이전 섹션에서 언급한 할인 계수입니다.
# EPS_START는 엡실론의 시작 값입니다.
# EPS_END는 엡실론의 최종 값입니다.
# EPS_DECAY는 엡실론의 지수 감쇠(exponential decay) 속도 제어하며, 높을수록 감쇠 속도가 느립니다.
# TAU는 목표 네트워크의 업데이트 속도입니다.
# LR은 ``AdamW`` 옵티마이저의 학습율(learning rate)입니다.
BATCH_SIZE = 128
GAMMA = 0.99
EPS_START = 0.9
EPS_END = 0.05
EPS_DECAY = 1000
TAU = 0.005
LR = 1e-4
```

모델 생성 및 환경 세팅

```
# gym 행동 공간에서 행동의 숫자를 얻습니다.
n_actions = env.action_space.n
# 상태 관측 횟수를 얻습니다.
state, info = env.reset()
n_observations = len(state)

policy_net = DQN(n_observations, n_actions).to(device)
target_net = DQN(n_observations, n_actions).to(device)
target_net.load_state_dict(policy_net.state_dict())

optimizer = optim.AdamW(policy_net.parameters(), lr=LR, amsgrad=True)
memory = ReplayMemory(10000)

steps_done = 0
```

행동 선택 함수

```
def select_action(state):
    global steps_done
    sample = random.random()
    eps_threshold = EPS_END + (EPS_START - EPS_END) * ₩
        math.exp(-1. * steps_done / EPS_DECAY)
    steps_done += 1
    if sample > eps_threshold:
        with torch.no_grad():
            # t.max (1)은 각 행의 가장 큰 열 값을 반환합니다.
            # 최대 결과의 두번째 열은 최대 요소의 주소값이므로,
            # 기대 보상이 더 큰 행동을 선택할 수 있습니다.
            return policy_net(state).max(1)[1].view(1, 1)
    else:
        return torch.tensor([[env.action_space.sample()]], device=device, dtype=torch.long)
```

학습 상태 확인 그래프 함수

```
episode_durations = []

def plot_durations(show_result=False):
    plt.figure(1)
    durations_t = torch.tensor(episode_durations, dtype=torch.float)
    if show_result:
        plt.title('Result')
    else:
        plt.clf()
        plt.title('Training...')
    plt.xlabel('Episode')
    plt.ylabel('Duration')
    plt.plot(durations_t.numpy())
    # 100개의 에피소드 평균을 가져 와서 도표 그리기
    if len(durations_t) >= 100:
        means = durations_t.unfold(0, 100, 1).mean(1).view(-1)
        means = torch.cat((torch.zeros(99), means))
        plt.plot(means.numpy())

    plt.pause(0.001)  # 도표가 업데이트되도록 잠시 멈춤
    if is_ipython:
        if not show_result:
            display.display(plt.gcf())
            display.clear_output(wait=True)
        else:
            display.display(plt.gcf())
```

모델 학습 함수

```
def optimize_model():
    if len(memory) < BATCH_SIZE:
        return
    transitions = memory.sample(BATCH_SIZE)
    batch = Transition(*zip(*transitions))

    # 최종이 아닌 상태의 마스크를 계산하고 배치 요소를 연결합니다
    non_final_mask = torch.tensor(tuple(map(lambda s: s is not None,
                                            batch.next_state)), device=device, dtype=torch.bool)
    non_final_next_states = torch.cat([s for s in batch.next_state
                                       if s is not None])
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)

    # 이들은 policy_net에 따라 각 배치 상태에 대해 선택된 행동입니다.
    state_action_values = policy_net(state_batch).gather(1, action_batch)

    # 모든 다음 상태를 위한 V(s_{t+1}) 계산
    # 이것은 마스크를 기반으로 병합되어 기대 상태 값을 갖거나 상태가 최종인 경우 0을 갖습니다.
    next_state_values = torch.zeros(BATCH_SIZE, device=device)
    with torch.no_grad():
        next_state_values[non_final_mask] = target_net(non_final_next_states).max(1)[0]
    # 기대 Q 값 계산
    expected_state_action_values = (next_state_values * GAMMA) + reward_batch

    # Huber 손실 계산
    criterion = nn.SmoothL1Loss()
    loss = criterion(state_action_values, expected_state_action_values.unsqueeze(1))

    # 모델 최적화
    optimizer.zero_grad()
    loss.backward()
    # 변화도 클리핑 바꿔치기
    torch.nn.utils.clip_grad_value_(policy_net.parameters(), 100)
    optimizer.step()
```

모델 학습 진행

```
if torch.cuda.is_available():
    num_episodes = 600
else:
    num_episodes = 50

for i_episode in range(num_episodes):
    state, info = env.reset()
    state = torch.tensor(state, dtype=torch.float32, device=device).unsqueeze(0)
    for t in count():
        action = select_action(state)
        observation, reward, terminated, truncated, _ = env.step(action.item())
        reward = torch.tensor([reward], device=device)
        done = terminated or truncated

        if terminated:
            next_state = None
        else:
            next_state = torch.tensor(observation, dtype=torch.float32, device=device).unsqueeze(0)

        memory.push(state, action, next_state, reward)

        state = next_state

        optimize_model()

        # 목표 네트워크의 가중치를 소프트 업데이트
        target_net_state_dict = target_net.state_dict()
        policy_net_state_dict = policy_net.state_dict()
        for key in policy_net_state_dict:
            target_net_state_dict[key] = policy_net_state_dict[key]*TAU + target_net_state_dict[key]*(1-TAU)
        target_net.load_state_dict(target_net_state_dict)

        if done:
            episode_durations.append(t + 1)
            plot_durations()
            break

print('Complete')
plot_durations(show_result=True)
plt.ioff()
plt.show()
```

강화학습 결과 확인

```
def show_video():
  import base64
  from IPython import display as ipythondisplay
  import glob
  import io
  from IPython.display import HTML

  mp4list = glob.glob('video/*.mp4')
  if len(mp4list) > 0:
    mp4 = mp4list[0]
    video = io.open(mp4, 'r+b').read()
    encoded = base64.b64encode(video)
    ipythondisplay.display(HTML(data='''<video alt="test" autoplay
                loop controls style="height: 400px;">
                <source src="data:video/mp4;base64,{0}" type="video/mp4" />
             </video>'''.format(encoded.decode('ascii'))))
  else:
    print("Could not find video")
```

Colab 화면 가상 화면 만들기

```
def wrap_env(env):
  from gymnasium.wrappers.record_video import RecordVideo
  env = RecordVideo(env, './video')
  return env
```

현재 모델 학습 결과 확인

```
from colabgymrender.recorder import Recorder
from pyvirtualdisplay import Display
import numpy as np
from IPython import display

_display = Display(visible=False, size=(1400, 900))
_ = _display.start()
envs = wrap_env(gym.make('CartPole-v1',render_mode='rgb_array'))

done = False
state, info = envs.reset()


while not done:
  envs.render()
  state = torch.tensor(state, dtype=torch.float32, device=device).unsqueeze(0)
  for t in count():

      action = select_action(state)
      observation, reward, terminated, truncated, _ = envs.step(action.item())
      reward = torch.tensor([reward], device=device)
      done = terminated or truncated

      if terminated:
          next_state = None
          done = True
          break
      else:
          next_state = torch.tensor(observation, dtype=torch.float32, device=device).unsqueeze(0)

       # 다음 상태로 이동
      state = next_state
envs.close()
show_video()
```