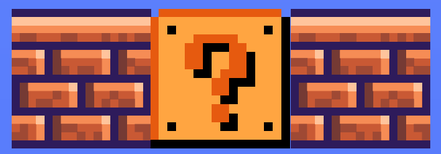
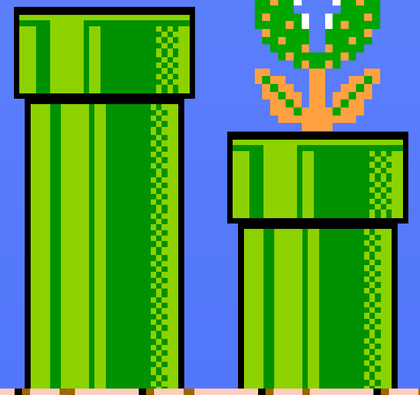
강화학습 기말 프로젝트

SUPER MARIO
REINFORCEMENT LEARNING

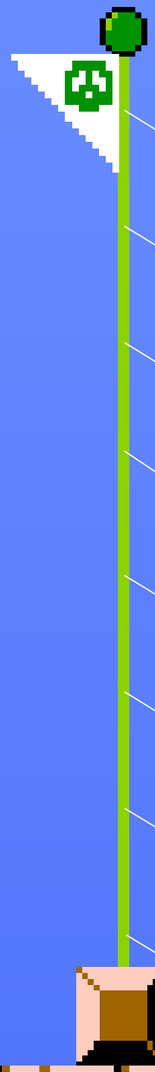


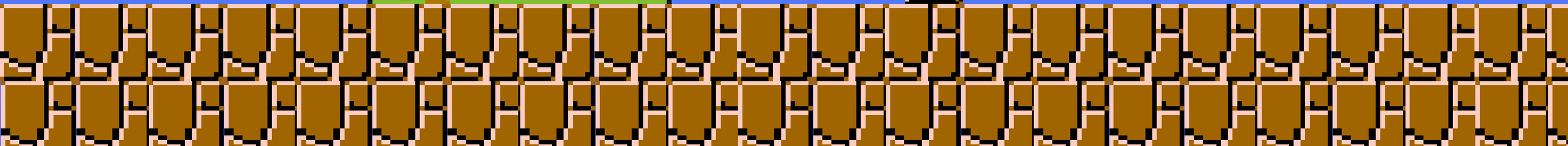
START



말25 / 맹의현, 진영인

Table Of Contents

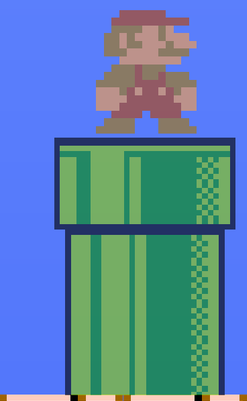





BACKGROUND

About Super Mario

Game Guide



← →	: 좌우 이동
↓	: 웅크리기
A	: 점프
B	: 대시, 불꽃 던지기

Agent(마리오)의 “Action, Reward”를 확실히 정의한다면

강화학습에 적당하지 않을까?

ENVIRONMENT

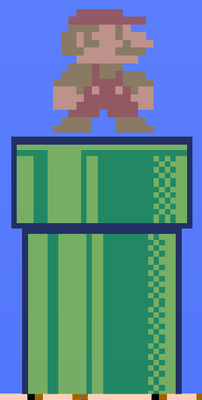
gym-super-mario-bros

```
%pip install gym-super-mario-bros
```

✓ 1.9s Python

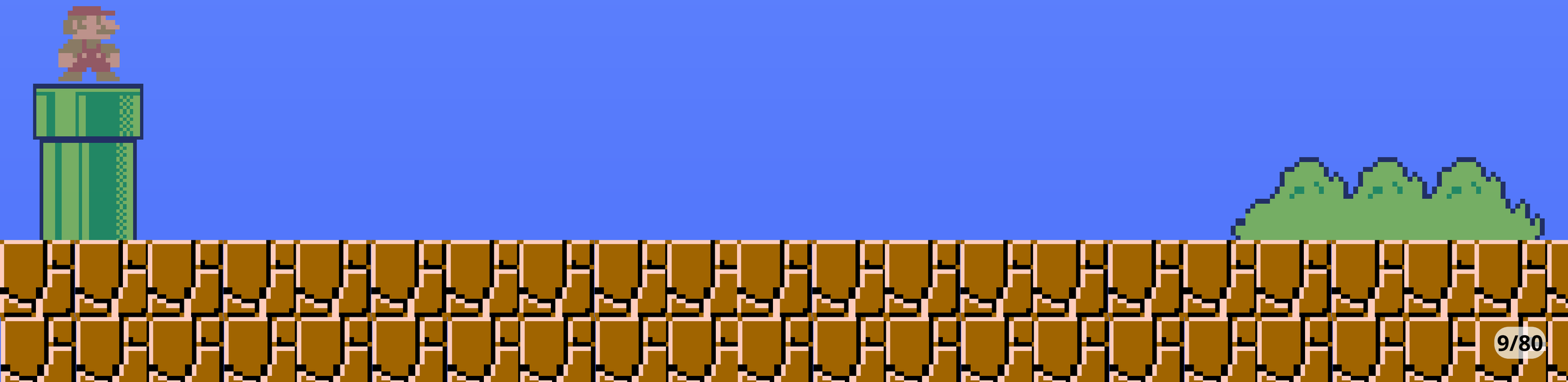
Super Mario Bros를 위한 Open AI Gym 환경

(nes-py emulator를 사용)



Environment Declaration

```
env = gym_super_mario_bros.make("SuperMarioBros-1-1-v0", render_mode='human', apply_api_compatibility=True)
```

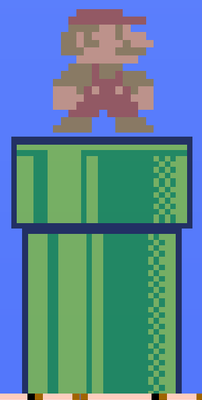


Environment Declaration

```
env = gym_super_mario_bros.make("SuperMarioBros-1-1-v0", render_mode='human', apply_api_compatibility=True)
```

→ SuperMarioBros-<**world**>-<**stage**>-<**version**>

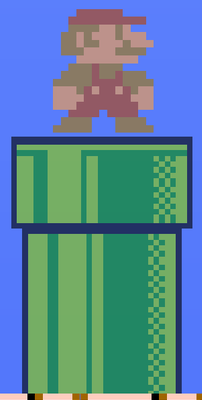
(1~8) (1~4) (1~4)



Environment Declaration

```
env = gym_super_mario_bros.make("SuperMarioBros-1-1-v0", render_mode='human', apply_api_compatibility=True)
```

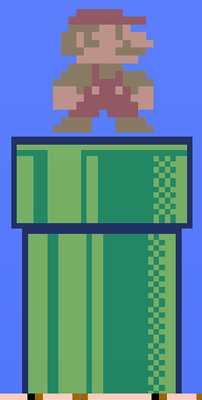
→ **게임 화면**을 볼 수 있도록 설정



Environment Declaration

```
env = gym_super_mario_bros.make("SuperMarioBros-1-1-v0", render_mode='human', apply_api_compatibility=True)
```

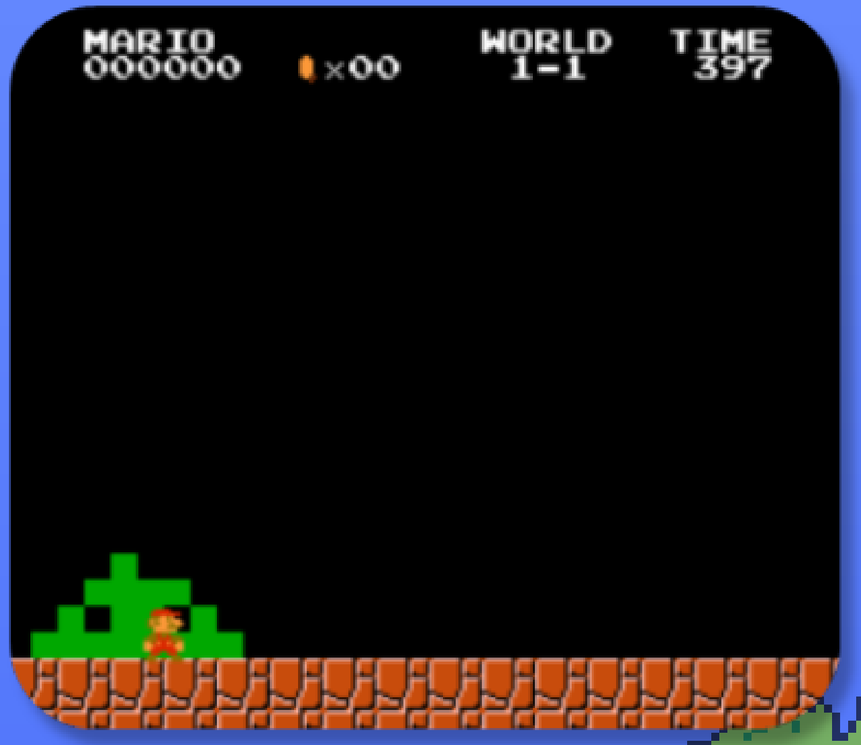
최신 Gym API와 **호환성 유지**



Versions

SuperMarioBros-v0
(Standard ROM)

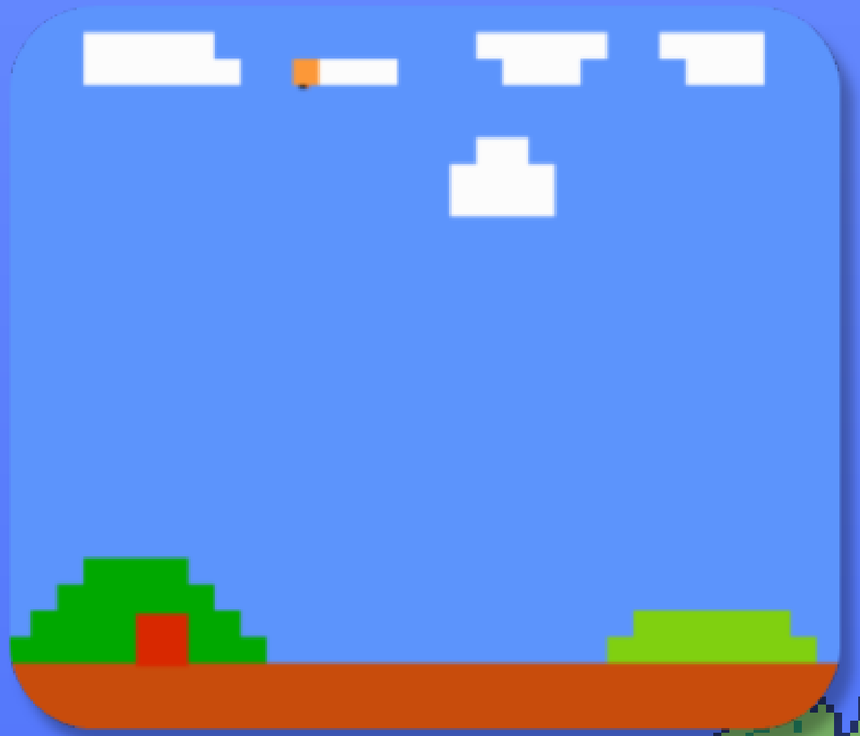
SuperMarioBros-v1
(Downsampled ROM)



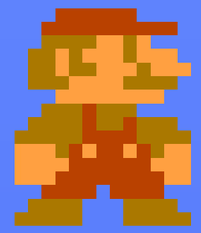
Versions

SuperMarioBros-v2
(Pixel ROM)

SuperMarioBros-v3
(Rectangle ROM)



Observation & Action Space



Agent

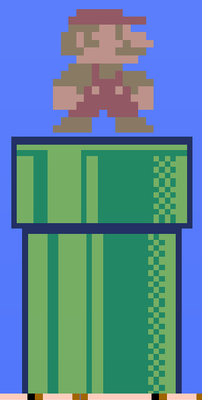
Observation,
Reward

Action

Environment

Action Space : Posible

0: Noop	5: Left + B	10: Right + A + B
1: Up	6: Left + A + B	11: A
2: Down	7: Right	12: B
3: Left	8: Right + A	13: A + B
4: Left + A	9: Right + B	



Action Space : Limitation

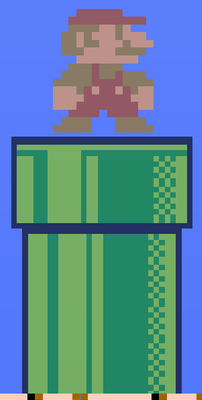
보유한 자원 내에서 효율적인 학습을 위해 **action space를 제한**

0: Right　　1: Right + A

```python
env = JoypadSpace(env, [["right"], ["right", "A"]])
```

Action Space : SkipFrame

```python
class SkipFrame(gym.Wrapper):
    def __init__(self, env, skip):
        """모든 `skip` 프레임만 반환합니다."""
        super().__init__(env)
        self._skip = skip

    def step(self, action):
        """행동을 반복하고 포상을 더합니다."""
        total_reward = 0.0
        for i in range(self._skip):
            # 포상을 누적하고 동일한 작업을 반복합니다.
            obs, reward, done, trunk, info = self.env.step(action)
            total_reward += reward
            if done:
                break
        return obs, total_reward, done, trunk, info
```

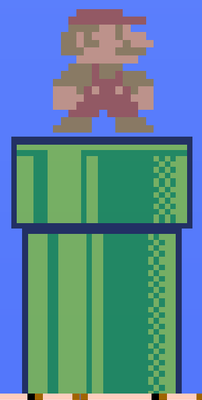


Action Space : SkipFrame

일정 frame만큼 동일한 **action을 반복 수행**하여
reward 누적 후 한 번에 반환

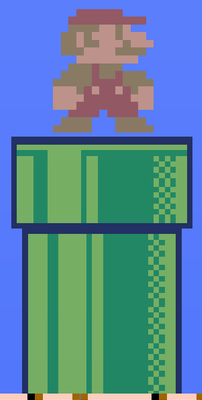
↓

모든 frame을 처리하지 않아 **처리 속도가 빨라짐**



Observation Space : GrayScale

```python
class GrayScaleObservation(gym.ObservationWrapper):
    def __init__(self, env):
        super().__init__(env)
        obs_shape = self.observation_space.shape[:2]
        self.observation_space = Box(low=0, high=255, shape=obs_shape, dtype=np.uint8)

    def permute_orientation(self, observation):
        # [H, W, C] 배열을 [C, H, W] 텐서로 바꿉니다.
        observation = np.transpose(observation, (2, 0, 1))
        observation = torch.tensor(observation.copy(), dtype=torch.float)
        return observation

    def observation(self, observation):
        observation = self.permute_orientation(observation)
        transform = T.Grayscale()
        observation = transform(observation)
        return observation
```

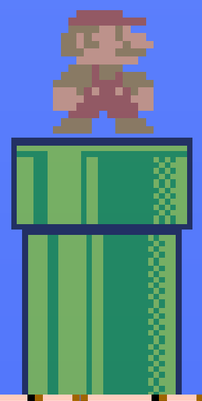


Observation Space : GrayScale

observation space를 **흑백으로 변환**해 색상 정보 제거

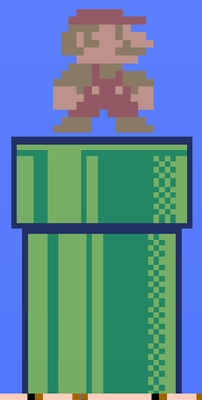
↓

데이터의 크기가 1/3으로 줄어 **학습 속도 향상**
모델이 색상에 영향 받지 않고 **모양과 움직임에 집중**



Observation Space : Resize

```python
class ResizeObservation(gym.ObservationWrapper):
    def __init__(self, env, shape):
        super().__init__(env)
        if isinstance(shape, int):
            self.shape = (shape, shape)
        else:
            self.shape = tuple(shape)

        obs_shape = self.shape + self.observation_space.shape[2:]
        self.observation_space = Box(low=0, high=255, shape=obs_shape, dtype=np.uint8)

    def observation(self, observation):
        transforms = T.Compose(
            [T.Resize(self.shape, antialias=True), T.Normalize(0, 255)]
        )
        observation = transforms(observation).squeeze(0)
        return observation
```

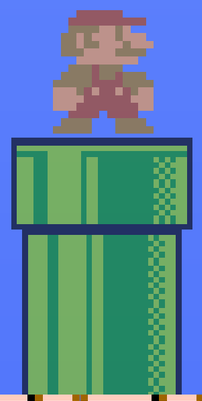


Observation Space : Resize

observation space를 **고정된 크기**(84 X 84)로 변환

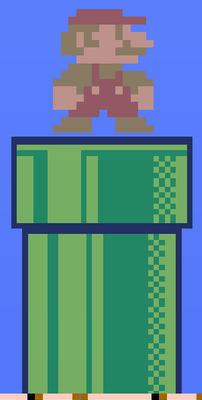
↓

데이터의 크기를 적절히 줄여 **학습 속도 향상**
모델의 **일관성 유지**



Apply Wrapper

```
# 래퍼를 환경에 적용합니다.
env = SkipFrame(env, skip=4)
env = GrayScaleObservation(env)
env = ResizeObservation(env, shape=84)
env = FrameStack(env, num_stack=4)
```



*** FrameStack( )** : 마지막 4개의 프레임을 스택으로 쌓아 observation 데이터로 활용


AGENT

Mario Agent

우리의 agent **“Mario”**의 기능

1. Act
현재 state를 기반으로 action policy에 따라 선택

2. Remember
cache를 통해 메모리에 경험을 추가하고,
recall을 통해 경험을 메모리에서 불러와 사용

3. Learn
최적의 action을 위한 Q-function 업데이트

Act

```python
def act(self, state):
    # 임의의 행동을 선택하기
    if np.random.rand() < self.exploration_rate:
        action_idx = np.random.randint(self.action_dim)

    # 최적의 행동을 이용하기
    else:
        state = state[0].__array__() if isinstance(state, tuple) else state.__array__()
        state = torch.tensor(state, device=self.device).unsqueeze(0)
        action_values = self.net(state, model="online")
        action_idx = torch.argmax(action_values, axis=1).item()

    # exploration_rate 감소하기
    self.exploration_rate *= self.exploration_rate_decay
    self.exploration_rate = max(self.exploration_rate_min, self.exploration_rate)

    # 스텝 수 증가하기
    self.curr_step += 1
    return action_idx
```

Act

```python
def act(self, state):
    # 임의의 행동을 선택하기
    if np.random.rand() < self.exploration_rate:
        action_idx = np.random.randint(self.action_dim)

    # 최적의 행동을 이용하기
    else:
        state = state[0].__array__() if isinstance(state, tuple) else state.__array__()
        state = torch.tensor(state, device=self.device).unsqueeze(0)
        action_values = self.net(state, model="online")
        action_idx = torch.argmax(action_values, axis=1).item()

    # exploration_rate 감소하기
    self.exploration_rate *= self.exploration_rate_decay
    self.exploration_rate = max(self.exploration_rate_min, self.exploration_rate)

    # 스텝 수 증가하기
    self.curr_step += 1
    return action_idx
```

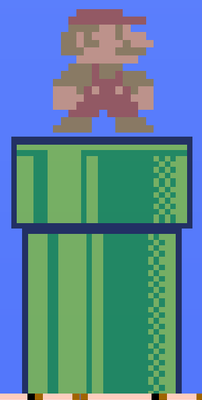
epsilon-greedy action 선택

Act

```python
    def act(self, state):
        # 임의의 행동을 선택하기
        if np.random.rand() < self.exploration_rate:
            action_idx = np.random.randint(self.action_dim)

        # 최적의 행동을 이용하기
        else:
            state = state[0].__array__() if isinstance(state, tuple) else state.__array__()
            state = torch.tensor(state, device=self.device).unsqueeze(0)
            action_values = self.net(state, model="online")
            action_idx = torch.argmax(action_values, axis=1).item()

        # exploration_rate 감소하기
        self.exploration_rate *= self.exploration_rate_decay
        self.exploration_rate = max(self.exploration_rate_min, self.exploration_rate)

        # 스텝 수 증가하기
        self.curr_step += 1
        return action_idx
```

*** Exploration**

일정 확률(ε) 이하로 **random action** 선택



Act

```python
    def act(self, state):
        # 임의의 행동을 선택하기
        if np.random.rand() < self.exploration_rate:
            action_idx = np.random.randint(self.action_dim)

        # 최적의 행동을 이용하기
        else:
            state = state[0].__array__() if isinstance(state, tuple) else state.__array__()
            state = torch.tensor(state, device=self.device).unsqueeze(0)
            action_values = self.net(state, model="online")
            action_idx = torch.argmax(action_values, axis=1).item()

        # exploration_rate 감소하기
        self.exploration_rate *= self.exploration_rate_decay
        self.exploration_rate = max(self.exploration_rate_min, self.exploration_rate)

        # 스텝 수 증가하기
        self.curr_step += 1
        return action_idx
```

*** Exploitation**

Q-network를 이용해 **최적의 action** 선택

Act

```python
    def act(self, state):
        # 임의의 행동을 선택하기
        if np.random.rand() < self.exploration_rate:
            action_idx = np.random.randint(self.action_dim)

        # 최적의 행동을 이용하기
        else:
            state = state[0].__array__() if isinstance(state, tuple) else state.__array__()
            state = torch.tensor(state, device=self.device).unsqueeze(0)
            action_values = self.net(state, model="online")
            action_idx = torch.argmax(action_values, axis=1).item()

        # exploration_rate 감소하기
        self.exploration_rate *= self.exploration_rate_decay
        self.exploration_rate = max(self.exploration_rate_min, self.exploration_rate)

        # 스텝 수 증가하기
        self.curr_step += 1
        return action_idx
```

학습이 진행됨에 따라 **exploration_rate(ε) 감소**

Remember : Cache

```
    def cache(self, state, next_state, action, reward, done):
        def first_if_tuple(x):
            return x[0] if isinstance(x, tuple) else x
        state = first_if_tuple(state).__array__()
        next_state = first_if_tuple(next_state).__array__()

        state = torch.tensor(state)
        next_state = torch.tensor(next_state)
        action = torch.tensor([action])
        reward = torch.tensor([reward])
        done = torch.tensor([done])

        # self.memory.append((state, next_state, action, reward, done,))
        self.memory.add(TensorDict({"state": state, "next_state": next_state, "action": action,
                                    "reward": reward, "done": done}, batch_size=[]))
```

Remember : Cache

```
    def cache(self, state, next_state, action, reward, done):
        def first_if_tuple(x):
            return x[0] if isinstance(x, tuple) else x
        state = first_if_tuple(state).__array__()
        next_state = first_if_tuple(next_state).__array__()

        state = torch.tensor(state)
        next_state = torch.tensor(next_state)
        action = torch.tensor([action])
        reward = torch.tensor([reward])
        done = torch.tensor([done])

        # self.memory.append((state, next_state, action, reward, done,))
        self.memory.add(TensorDict({"state": state, "next_state": next_state, "action": action,
                                    "reward": reward, "done": done}, batch_size=[]))
```

각 경험 요소를 **torch.tensor 형태**로 변환

Remember : Cache

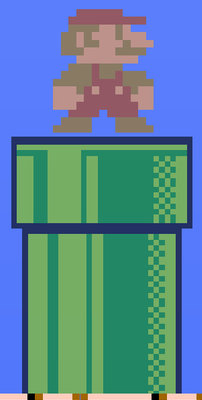
```python
def cache(self, state, next_state, action, reward, done):
    def first_if_tuple(x):
        return x[0] if isinstance(x, tuple) else x
    state = first_if_tuple(state).__array__()
    next_state = first_if_tuple(next_state).__array__()

    state = torch.tensor(state)
    next_state = torch.tensor(next_state)
    action = torch.tensor([action])
    reward = torch.tensor([reward])
    done = torch.tensor([done])

    # self.memory.append((state, next_state, action, reward, done,))
    self.memory.add(TensorDict({"state": state, "next_state": next_state, "action": action,
                                "reward": reward, "done": done}, batch_size=[]))
```

TensorDict 형식으로 데이터를 구성한 뒤, **버퍼에 저장**

Remember : Recall

```
    def recall(self):
        batch = self.memory.sample(self.batch_size).to(self.device)
        state, next_state, action, reward, done = (batch.get(key) for key in ("state", "next_state",
                                                                               "action", "reward", "done"))
        return state, next_state, action.squeeze(), reward.squeeze(), done.squeeze()
```

batch 크기만큼 **sampling** 후, 각 **경험 요소를 반환**




LEARN

DDQN이란?

Double Deep Q-Network

action의 선택(online 네트워크)과
평가(target 네트워크)를 분리

↓

Q-value의 신뢰성 향상
overestimate 문제 완화

"Deep Reinforcement Learning with Double Q-learning", Google DeepMind, 2015.12.8.
https://arxiv.org/pdf/1509.06461

MarioNet (DDQN)

```python
class MarioNet(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        c, h, w = input_dim

        if h != 84:
            raise ValueError(f"Expecting input height: 84, got: {h}")
        if w != 84:
            raise ValueError(f"Expecting input width: 84, got: {w}")

        self.online = self.__build_cnn(c, output_dim)

        self.target = self.__build_cnn(c, output_dim)
        self.target.load_state_dict(self.online.state_dict())

        # Q_target 매개변수 값은 고정시킵니다.
        for p in self.target.parameters():
            p.requires_grad = False

    def forward(self, input, model):
        if model == "online":
            return self.online(input)
        elif model == "target":
            return self.target(input)

    def __build_cnn(self, c, output_dim):
        return nn.Sequential(
            nn.Conv2d(in_channels=c, out_channels=32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(3136, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim),
        )
```

MarioNet (DDQN)

```python
class MarioNet(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        c, h, w = input_dim

        if h != 84:
            raise ValueError(f"Expecting input height: 84, got: {h}")
        if w != 84:
            raise ValueError(f"Expecting input width: 84, got: {w}")

        self.online = self.__build_cnn(c, output_dim)

        self.target = self.__build_cnn(c, output_dim)
        self.target.load_state_dict(self.online.state_dict())

        # Q_target 매개변수 값은 고정시킵니다.
        for p in self.target.parameters():
            p.requires_grad = False

    def forward(self, input, model):
        if model == "online":
            return self.online(input)
        elif model == "target":
            return self.target(input)

    def __build_cnn(self, c, output_dim):
        return nn.Sequential(
            nn.Conv2d(in_channels=c, out_channels=32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(3136, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim),
        )
```

input 차원이 **(c, 84, 84) 형식**이 아니면 error

MarioNet (DDQN)

```python
class MarioNet(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        c, h, w = input_dim

        if h != 84:
            raise ValueError(f"Expecting input height: 84, got: {h}")
        if w != 84:
            raise ValueError(f"Expecting input width: 84, got: {w}")

        self.online = self.__build_cnn(c, output_dim)

        self.target = self.__build_cnn(c, output_dim)
        self.target.load_state_dict(self.online.state_dict())

        # Q_target 매개변수 값은 고정시킵니다.
        for p in self.target.parameters():
            p.requires_grad = False

    def forward(self, input, model):
        if model == "online":
            return self.online(input)
        elif model == "target":
            return self.target(input)

    def __build_cnn(self, c, output_dim):
        return nn.Sequential(
            nn.Conv2d(in_channels=c, out_channels=32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(3136, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim),
        )
```

online 네트워크 정의(학습을 통해 **Q_online** 예측)

MarioNet (DDQN)

```python
class MarioNet(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        c, h, w = input_dim

        if h != 84:
            raise ValueError(f"Expecting input height: 84, got: {h}")
        if w != 84:
            raise ValueError(f"Expecting input width: 84, got: {w}")

        self.online = self.__build_cnn(c, output_dim)

        self.target = self.__build_cnn(c, output_dim)
        self.target.load_state_dict(self.online.state_dict())

        # Q_target 매개변수 값은 업데이트 X
        for p in self.target.parameters():
            p.requires_grad = False

    def forward(self, input, model):
        if model == "online":
            return self.online(input)
        elif model == "target":
            return self.target(input)

    def __build_cnn(self, c, output_dim):
        return nn.Sequential(
            nn.Conv2d(in_channels=c, out_channels=32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(3136, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim),
        )
```

target 네트워크 정의(일정 간격마다 **online 네트워크의 가중치 복사**)

➜ 학습 중에는 **Q_target 고정**

MarioNet (DDQN)

```python
class MarioNet(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        c, h, w = input_dim

        if h != 84:
            raise ValueError(f"Expecting input height: 84, got: {h}")
        if w != 84:
            raise ValueError(f"Expecting input width: 84, got: {w}")

        self.online = self.__build_cnn(c, output_dim)

        self.target = self.__build_cnn(c, output_dim)
        self.target.load_state_dict(self.online.state_dict())

        # Q_target 매개변수 값은 고정시킵니다.
        for p in self.target.parameters():
            p.requires_grad = False

    def forward(self, input, model):
        if model == "online":
            return self.online(input)
        elif model == "target":
            return self.target(input)

    def __build_cnn(self, c, output_dim):
        return nn.Sequential(
            nn.Conv2d(in_channels=c, out_channels=32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(3136, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim),
        )
```

target 네트워크는 학습하지 않으므로, **매개변수 고정**

MarioNet (DDQN)

```python
class MarioNet(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        c, h, w = input_dim

        if h != 84:
            raise ValueError(f"Expecting input height: 84, got: {h}")
        if w != 84:
            raise ValueError(f"Expecting input width: 84, got: {w}")

        self.online = self.__build_cnn(c, output_dim)

        self.target = self.__build_cnn(c, output_dim)
        self.target.load_state_dict(self.online.state_dict())

        # Q_target 매개변수 값은 고정시킵니다.
        for p in self.target.parameters():
            p.requires_grad = False

    def forward(self, input, model):
        if model == "online":
            return self.online(input)
        elif model == "target":
            return self.target(input)

    def __build_cnn(self, c, output_dim):
        return nn.Sequential(
            nn.Conv2d(in_channels=c, out_channels=32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(3136, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim),
        )
```

model 매개변수에 따라 online or taget 네트워크로 **Q값 예측**

MarioNet (DDQN)

```python
class MarioNet(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        c, h, w = input_dim

        if h != 84:
            raise ValueError(f"Expecting input height: 84, got: {h}")
        if w != 84:
            raise ValueError(f"Expecting input width: 84, got: {w}")

        self.online = self.__build_cnn(c, output_dim)

        self.target = self.__build_cnn(c, output_dim)
        self.target.load_state_dict(self.online.state_dict())

        # Q_target 매개변수 값은 고정시킵니다.
        for p in self.target.parameters():
            p.requires_grad = False

    def forward(self, input, model):
        if model == "online":
            return self.online(input)
        elif model == "target":
            return self.target(input)
```

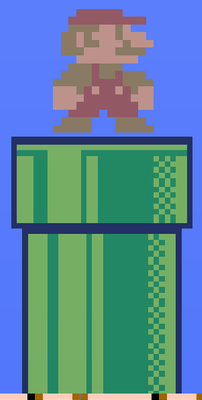
CNN 구조 정의

“**Conv layer** 3 + **Flatten layer** 1 + **FC layer** 2” 사용

```python
    def __build_cnn(self, c, output_dim):
        return nn.Sequential(
            nn.Conv2d(in_channels=c, out_channels=32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(3136, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim),
        )
```

TD_estimate & TD_target

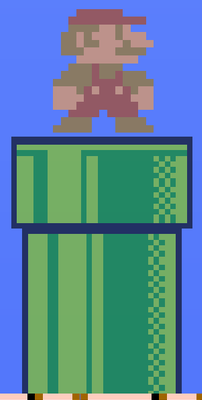
```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.gamma = 0.9

    def td_estimate(self, state, action):
        current_Q = self.net(state, model="online")[
            np.arange(0, self.batch_size), action
        ]  # Q_online(s,a)
        return current_Q

    @torch.no_grad()
    def td_target(self, reward, next_state, done):
        next_state_Q = self.net(next_state, model="online")
        best_action = torch.argmax(next_state_Q, axis=1)
        next_Q = self.net(next_state, model="target")[
            np.arange(0, self.batch_size), best_action
        ]
        return (reward + (1 - done.float()) * self.gamma * next_Q).float()
```



TD_estimate

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.gamma = 0.9

    def td_estimate(self, state, action):
        current_Q = self.net(state, model="online")[
            np.arange(0, self.batch_size), action
        ]  # Q_online(s,a)
        return current_Q

    @torch.no_grad()
    def td_target(self, reward, next_state, done):
        next_state_Q = self.net(next_state, model="online")
        best_action = torch.argmax(next_state_Q, axis=1)
        next_Q = self.net(next_state, model="target")[
            np.arange(0, self.batch_size), best_action
        ]
        return (reward + (1 - done.float()) * self.gamma * next_Q).float()
```

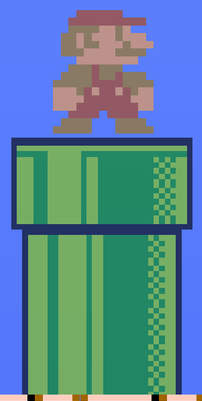
현재 state에서 선택된 **action**에 해당하는 **Q값** 추출



TD_target

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.gamma = 0.9

    def td_estimate(self, state, action):
        current_Q = self.net(state, model="online")[
            np.arange(0, self.batch_size), action
        ]  # Q_online(s,a)
        return current_Q

    @torch.no_grad()
    def td_target(self, reward, next_state, done):
        next_state_Q = self.net(next_state, model="online")
        best_action = torch.argmax(next_state_Q, axis=1)
        next_Q = self.net(next_state, model="target")[
            np.arange(0, self.batch_size), best_action
        ]
        return (reward + (1 - done.float()) * self.gamma * next_Q).float()
```

Q_target은 **역전파 알고리즘**을 수행하지 않음



TD_target

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.gamma = 0.9

    def td_estimate(self, state, action):
        current_Q = self.net(state, model="online")[
            np.arange(0, self.batch_size), action
        ]  # Q_online(s,a)
        return current_Q

    @torch.no_grad()
    def td_target(self, reward, next_state, done):
        next_state_Q = self.net(next_state, model="online")
        best_action = torch.argmax(next_state_Q, axis=1)
        next_Q = self.net(next_state, model="target")[
            np.arange(0, self.batch_size), best_action
        ]
        return (reward + (1 - done.float()) * self.gamma * next_Q).float()
```

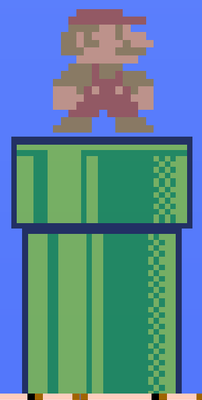
1. online 네트워크로 **다음 state**에서 **최대 Q값**을 가지는 **action** 선택

2. target 네트워크로 **최대 Q값을 평가**

Update Model

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=0.00025)
        self.loss_fn = torch.nn.SmoothL1Loss()

    def update_Q_online(self, td_estimate, td_target):
        loss = self.loss_fn(td_estimate, td_target)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        return loss.item()

    def sync_Q_target(self):
        self.net.target.load_state_dict(self.net.online.state_dict())
```

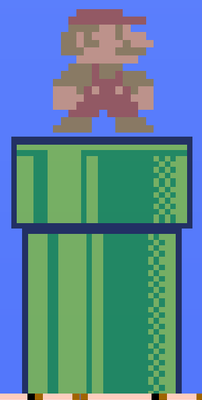
Python



Update Model

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=0.00025)
        self.loss_fn = torch.nn.SmoothL1Loss()

    def update_Q_online(self, td_estimate, td_target):
        loss = self.loss_fn(td_estimate, td_target)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        return loss.item()

    def sync_Q_target(self):
        self.net.target.load_state_dict(self.net.online.state_dict())
```

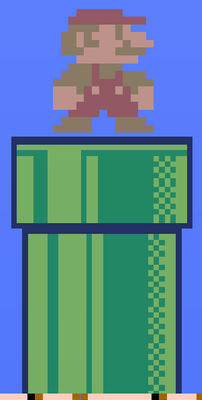
optimizer와 **loss function** 정의



Update Model

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=0.00025)
        self.loss_fn = torch.nn.SmoothL1Loss()

    def update_Q_online(self, td_estimate, td_target):
        loss = self.loss_fn(td_estimate, td_target)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        return loss.item()

    def sync_Q_target(self):
        self.net.target.load_state_dict(self.net.online.state_dict())
```

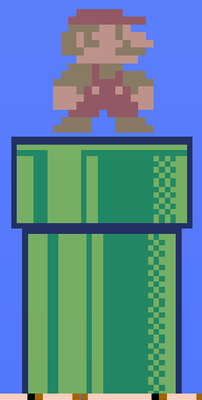
td_estimate와 td_target 간의 **loss를 계산**해 가중치를 업데이트



Update Model

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=0.00025)
        self.loss_fn = torch.nn.SmoothL1Loss()

    def update_Q_online(self, td_estimate, td_target):
        loss = self.loss_fn(td_estimate, td_target)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        return loss.item()

    def sync_Q_target(self):
        self.net.target.load_state_dict(self.net.online.state_dict())
```

target 네트워크의 가중치를 online 네트워크와 **동기화**



Combining Learning Methods

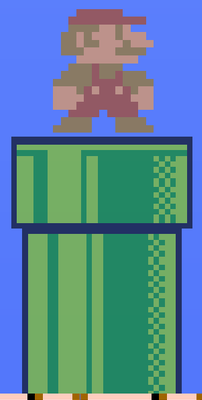
```python
    def learn(self):
        if self.curr_step % self.sync_every == 0:
            self.sync_Q_target()
        if self.curr_step % self.save_every == 0:
            self.save()
        if self.curr_step < self.burnin:
            return None, None
        if self.curr_step % self.learn_every != 0:
            return None, None

        # 메모리로부터 샘플링을 합니다.
        state, next_state, action, reward, done = self.recall()
        # TD 추정값을 가져옵니다.
        td_est = self.td_estimate(state, action)
        # TD 목표값을 가져옵니다.
        td_tgt = self.td_target(reward, next_state, done)
        # 실시간 Q(Q_online)을 통해 역전파 손실을 계산합니다.
        loss = self.update_Q_online(td_est, td_tgt)

        return (td_est.mean().item(), loss)
```

Python

Combining Learning Methods

```python
    def learn(self):
        if self.curr_step % self.sync_every == 0:
            self.sync_Q_target()
        if self.curr_step % self.save_every == 0:
            self.save()
        if self.curr_step < self.burnin:
            return None, None
        if self.curr_step % self.learn_every != 0:
            return None, None

        # 메모리로부터 샘플링을 합니다.
        state, next_state, action, reward, done = self.recall()
        # TD 추정값을 가져옵니다.
        td_est = self.td_estimate(state, action)
        # TD 목표값을 가져옵니다.
        td_tgt = self.td_target(reward, next_state, done)
        # 실시간 Q(Q_online)을 통해 역전파 손실을 계산합니다.
        loss = self.update_Q_online(td_est, td_tgt)

        return (td_est.mean().item(), loss)
```

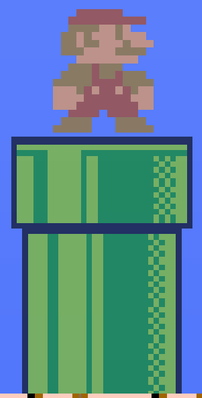
sync_every의 배수인 step마다 **target 네트워크 동기화**



Combining Learning Methods

```python
def learn(self):
    if self.curr_step % self.sync_every == 0:
        self.sync_Q_target()
    if self.curr_step % self.save_every == 0:
        self.save()
    if self.curr_step < self.burnin:
        return None, None
    if self.curr_step % self.learn_every != 0:
        return None, None

    # 메모리로부터 샘플링을 합니다.
    state, next_state, action, reward, done = self.recall()
    # TD 추정값을 가져옵니다.
    td_est = self.td_estimate(state, action)
    # TD 목표값을 가져옵니다.
    td_tgt = self.td_target(reward, next_state, done)
    # 실시간 Q(Q_online)을 통해 역전파 손실을 계산합니다.
    loss = self.update_Q_online(td_est, td_tgt)

    return (td_est.mean().item(), loss)
```

save_every의 배수인 step마다 **checkpoint 저장**



Combining Learning Methods

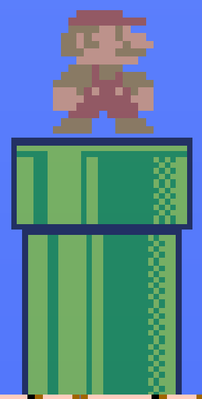
```
    def learn(self):
        if self.curr_step % self.sync_every == 0:
            self.sync_Q_target()
        if self.curr_step % self.save_every == 0:
            self.save()
        if self.curr_step < self.burnin:
            return None, None
        if self.curr_step % self.learn_every != 0:
            return None, None

        # 메모리로부터 샘플링을 합니다.
        state, next_state, action, reward, done = self.recall()
        # TD 추정값을 가져옵니다.
        td_est = self.td_estimate(state, action)
        # TD 목표값을 가져옵니다.
        td_tgt = self.td_target(reward, next_state, done)
        # 실시간 Q(Q_online)을 통해 역전파 손실을 계산합니다.
        loss = self.update_Q_online(td_est, td_tgt)

        return (td_est.mean().item(), loss)
```

최소한의 경험값(burnin)보다 현재 step이 작을때, 학습을 건너뜀

Combining Learning Methods

```
    def learn(self):
        if self.curr_step % self.sync_every == 0:
            self.sync_Q_target()
        if self.curr_step % self.save_every == 0:
            self.save()
        if self.curr_step < self.burnin:
            return None, None
        if self.curr_step % self.learn_every != 0:
            return None, None

        # 메모리로부터 샘플링 합니다.
        state, next_state, action, reward, done = self.recall()
        # TD 추정값을 가져옵니다.
        td_est = self.td_estimate(state, action)
        # TD 목표값을 가져옵니다.
        td_tgt = self.td_target(reward, next_state, done)
        # 실시간 Q(Q_online)을 통해 역전파 손실을 계산합니다.
        loss = self.update_Q_online(td_est, td_tgt)

        return (td_est.mean().item(), loss)
```

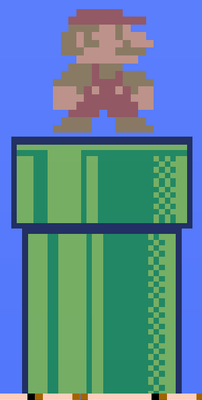
learn_every의 배수가 아닌 step의 경우, **학습을 건너뜀**



Combining Learning Methods

```python
def learn(self):
    if self.curr_step % self.sync_every == 0:
        self.sync_Q_target()
    if self.curr_step % self.save_every == 0:
        self.save()
    if self.curr_step < self.burnin:
        return None, None
    if self.curr_step % self.learn_every != 0:
        return None, None

    # 메모리로부터 샘플링을 합니다.
    state, next_state, action, reward, done = self.recall()
    # TD 추정값을 가져옵니다.
    td_est = self.td_estimate(state, action)
    # TD 목표값을 가져옵니다.
    td_tgt = self.td_target(reward, next_state, done)
    # 실시간 Q(Q_online)을 통해 역전파 손실을 계산합니다.
    loss = self.update_Q_online(td_est, td_tgt)

    return (td_est.mean().item(), loss)
```

경험 리플레이 메모리에서 **샘플링**



Combining Learning Methods

```python
    def learn(self):
        if self.curr_step % self.sync_every == 0:
            self.sync_Q_target()
        if self.curr_step % self.save_every == 0:
            self.save()
        if self.curr_step < self.burnin:
            return None, None
        if self.curr_step % self.learn_every != 0:
            return None, None

        # 메모리로부터 샘플링을 합니다.
        state, next_state, action, reward, done = self.recall()
        # TD 추정값을 가져옵니다.
        td_est = self.td_estimate(state, action)
        # TD 목표값을 가져옵니다.
        td_tgt = self.td_target(reward, next_state, done)
        # 실시간 Q(Q_online)을 통해 역전파 손실을 계산합니다.
        loss = self.update_Q_online(td_est, td_tgt)

        return (td_est.mean().item(), loss)
```

td_estimate와 td_target값을 계산해 **online 네트워크 업데이트**

Python


TRAIN

Training Agent

```python
# Mario 객체 초기화
mario = Mario(state_dim=(4, 84, 84), action_dim=env.action_space.n, save_dir=save_dir)

# 학습 진행
episodes = 500
for e in range(episodes):

    state = env.reset()

    while True:
        # 게임 화면 렌더링
        env.render()
        # 현재 상태에서 에이전트 실행하기
        action = mario.act(state)
        # 에이전트가 액션 수행하기
        next_state, reward, done, trunc, info = env.step(action)
        # 기억하기
        mario.cache(state, next_state, action, reward, done)
        # 배우기
        q, loss = mario.learn()
        # 기록하기
        logger.log_step(reward, loss, q)
        # 상태 업데이트하기
        state = next_state
        # 게임이 끝났는지 확인하기
        if done or info["flag_get"]:
            break

    logger.log_episode()

    # 일정 간격으로 기록 및 로그 저장
    if (e % 20 == 0) or (e == episodes - 1):
        logger.record(episode=e, epsilon=mario.exploration_rate, step=mario.curr_step)

# 환경 종료
env.close()
```

Training Agent

```
# Mario 객체 초기화
mario = Mario(state_dim=(4, 84, 84), action_dim=env.action_space.n, save_dir=save_dir)
```

새 **Mario 객체**를 생성해 **agent를 초기화**

```
# 학습 진행
episodes = 500
for e in range(episodes):

    state = env.reset()

    while True:
        # 게임 화면 렌더링
        env.render()
        # 현재 상태에서 에이전트 실행하기
        action = mario.act(state)
        # 에이전트가 액션 수행하기
        next_state, reward, done, trunc, info = env.step(action)
        # 기억하기
        mario.cache(state, next_state, action, reward, done)
        # 배우기
        q, loss = mario.learn()
        # 기록하기
        logger.log_step(reward, loss, q)
        # 상태 업데이트하기
        state = next_state
        # 게임이 끝났는지 확인하기
        if done or info["flag_get"]:
            break

    logger.log_episode()

    # 일정 간격으로 기록 및 로그 저장
    if (e % 20 == 0) or (e == episodes - 1):
        logger.record(episode=e, epsilon=mario.exploration_rate, step=mario.curr_step)

# 환경 종료
env.close()
```

Training Agent

```python
# Mario 객체 초기화
mario = Mario(state_dim=(4, 84, 84), action_dim=env.action_space.n, save_dir=save_dir)

# 학습 진행
episodes = 500
for e in range(episodes):

    state = env.reset()

    while True:
        # 게임 화면 렌더링
        env.render()
        # 현재 상태에서 에이전트 실행하기
        action = mario.act(state)
        # 에이전트가 액션 수행하기
        next_state, reward, done, trunc, info = env.step(action)
        # 기억하기
        mario.cache(state, next_state, action, reward, done)
        # 배우기
        q, loss = mario.learn()
        # 기록하기
        logger.log_step(reward, loss, q)
        # 상태 업데이트하기
        state = next_state
        # 게임이 끝났는지 확인하기
        if done or info["flag_get"]:
            break

    logger.log_episode()

    # 일정 간격으로 기록 및 로그 저장
    if (e % 20 == 0) or (e == episodes - 1):
        logger.record(episode=e, epsilon=mario.exploration_rate, step=mario.curr_step)

# 환경 종료
env.close()
```

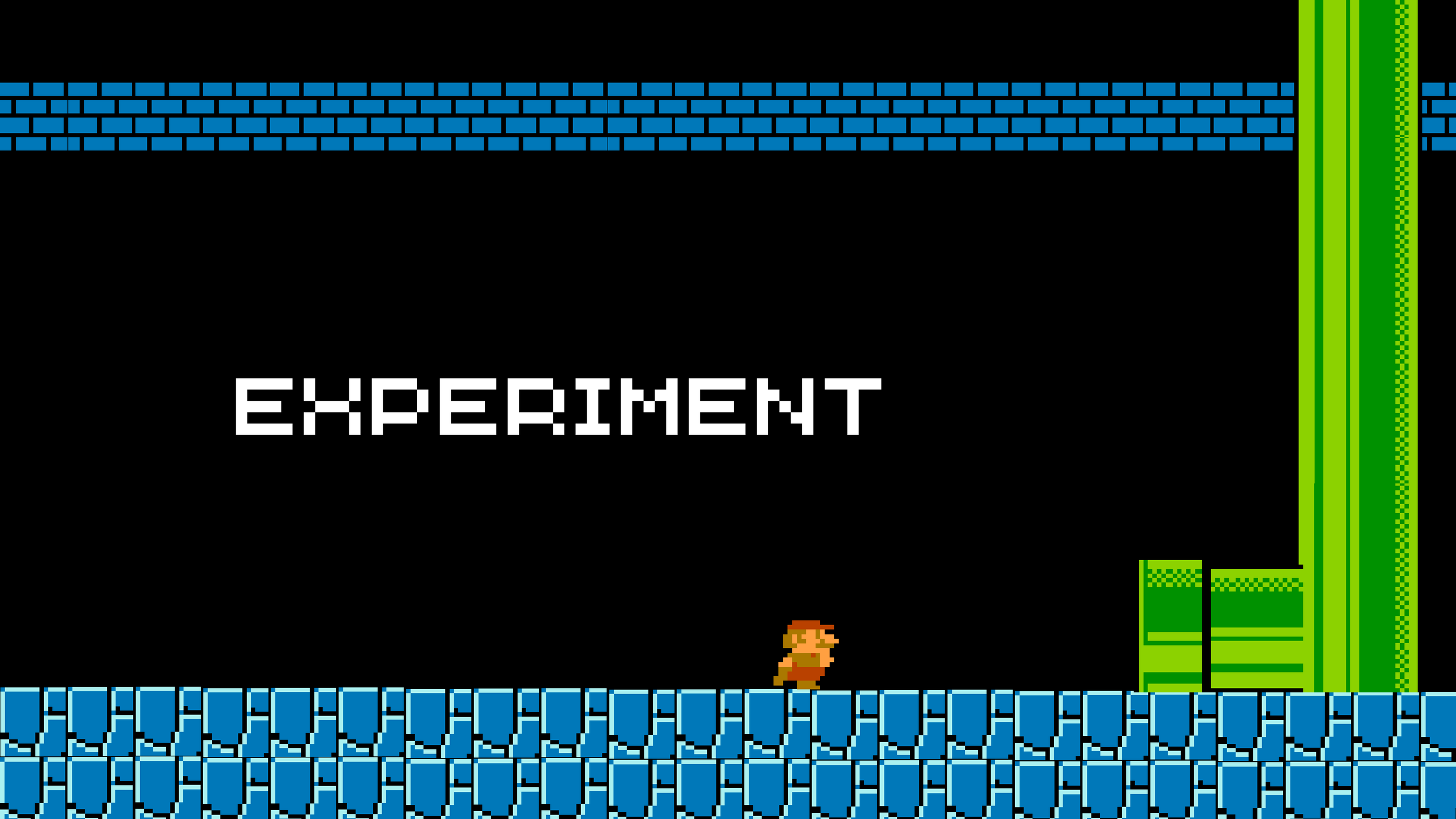
새 episode가 시작될 때, **환경을 초기화**

Training Agent

```python
# Mario 객체 초기화
mario = Mario(state_dim=(4, 84, 84), action_dim=env.action_space.n, save_dir=save_dir)

# 학습 진행
episodes = 500
for e in range(episodes):

    state = env.reset()

    while True:
        # 게임 화면 렌더링
        env.render()
        # 현재 상태에서 에이전트 실행하기
        action = mario.act(state)
        # 에이전트가 액션 수행하기
        next_state, reward, done, trunc, info = env.step(action)
        # 기억하기
        mario.cache(state, next_state, action, reward, done)
        # 배우기
        q, loss = mario.learn()
        # 기록하기
        logger.log_step(reward, loss, q)
        # 상태 업데이트하기
        state = next_state
        # 게임이 끝났는지 확인하기
        if done or info["flag_get"]:
            break

    logger.log_episode()

    # 일정 간격으로 기록 및 로그 저장
    if (e % 20 == 0) or (e == episodes - 1):
        logger.record(episode=e, epsilon=mario.exploration_rate, step=mario.curr_step)

# 환경 종료
env.close()
```

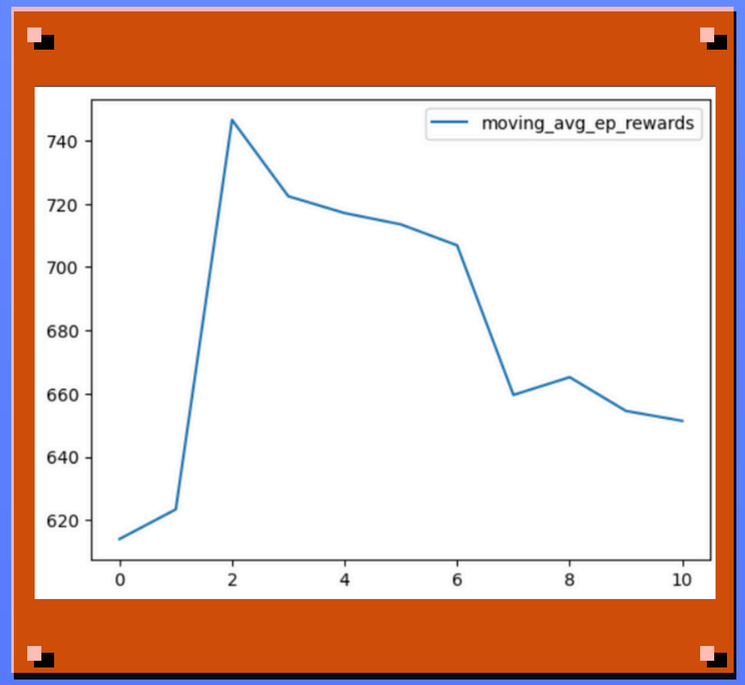
게임이 끝날 때까지 **반복 실행**

state → action → cache → learn → log → next_state

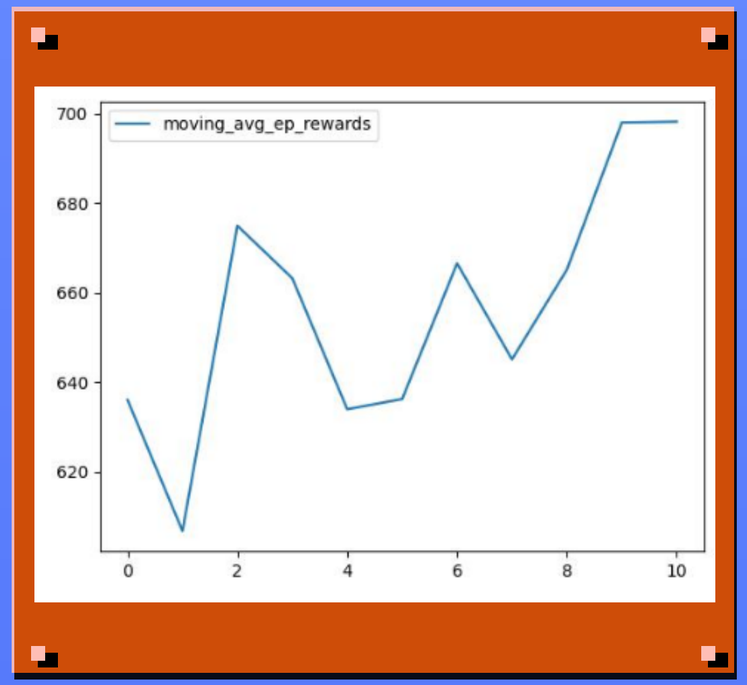

EXPERIMENT

Batch Size

batch_size = **32**



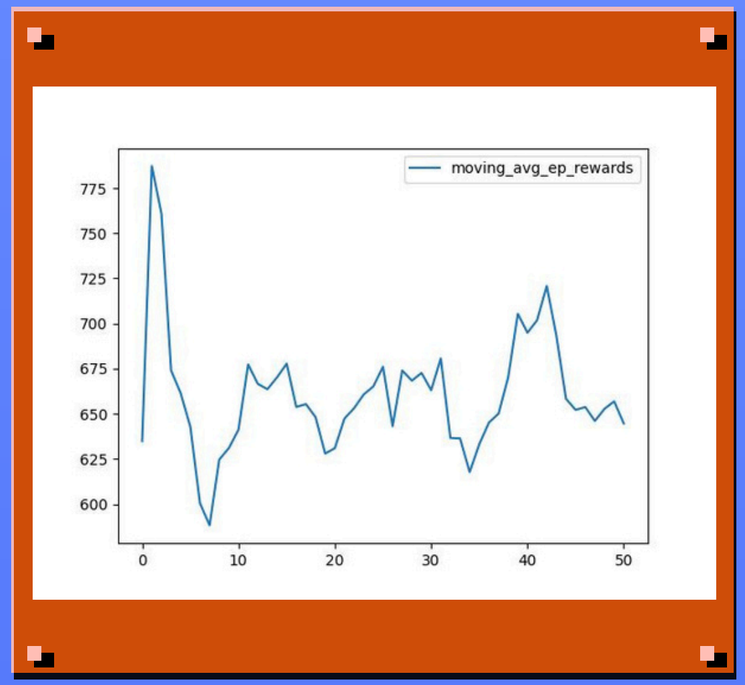
batch_size = **64**



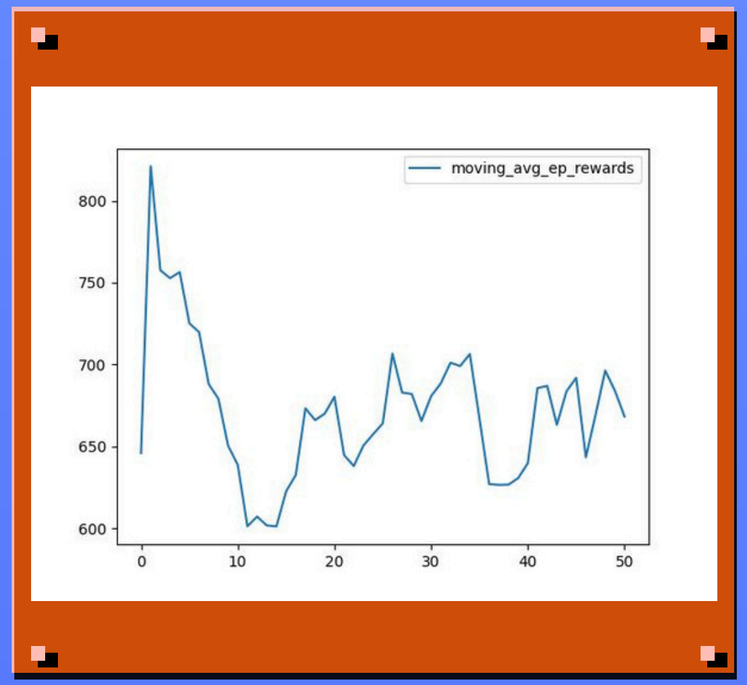
batch_size의 경우 **64**일 때, 보다 안정적으로 학습

Learn Every

learn_every = **10**



learn_every = **5**

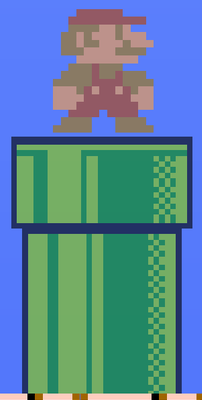


learn_every의 경우 **5**일 때, 약간의 성능 향상

Using Scheduler

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=0.00025)
        self.loss_fn = torch.nn.SmoothL1Loss()
        self.scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
            self.optimizer, mode="min", factor=0.5, patience=10
        )

    def update_Q_online(self, td_estimate, td_target):
        loss = self.loss_fn(td_estimate, td_target)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        # 스케줄러 업데이트
        self.scheduler.step()

        return loss.item()

    def sync_Q_target(self):
        self.net.target.load_state_dict(self.net.online.state_dict())
```

✓ 0.0s Python



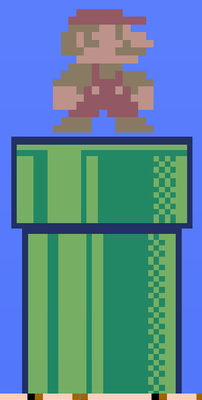
Using Scheduler

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=0.00025)
        self.loss_fn = torch.nn.SmoothL1Loss()
        self.scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
            self.optimizer, mode="min", factor=0.5, patience=10
        )

    def update_Q_online(self, td_estimate, td_target):
        loss = self.loss_fn(td_estimate, td_target)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        # 스케줄러 업데이트
        self.scheduler.step()

        return loss.item()

    def sync_Q_target(self):
        self.net.target.load_state_dict(self.net.online.state_dict())
```

✓ 0.0s

ReduceLROnPlateau 스케줄러 적용

→ 손실이 개선되지 않으면 학습률을 줄임



Python

Using Scheduler

```python
class Mario(Mario):
    def __init__(self, state_dim, action_dim, save_dir):
        super().__init__(state_dim, action_dim, save_dir)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=0.00025)
        self.loss_fn = torch.nn.SmoothL1Loss()
        self.scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
            self.optimizer, mode="min", factor=0.5, patience=10
        )

    def update_Q_online(self, td_estimate, td_target):
        loss = self.loss_fn(td_estimate, td_target)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        # 스케줄러 업데이트
        self.scheduler.step()

        return loss.item()

    def sync_Q_target(self):
        self.net.target.load_state_dict(self.net.online.state_dict())
```

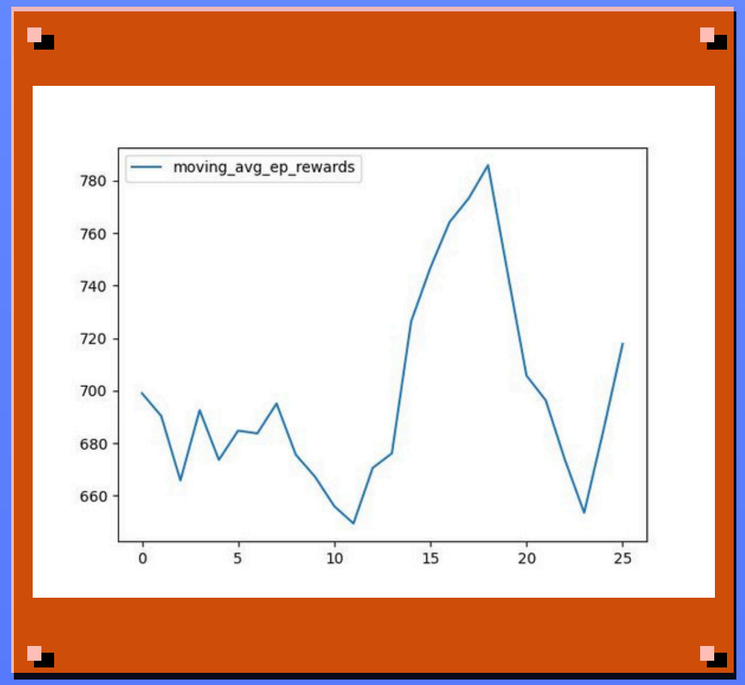
✓ 0.0s

계산된 loss값을 스케줄러에 전달해 **학습률 조정**

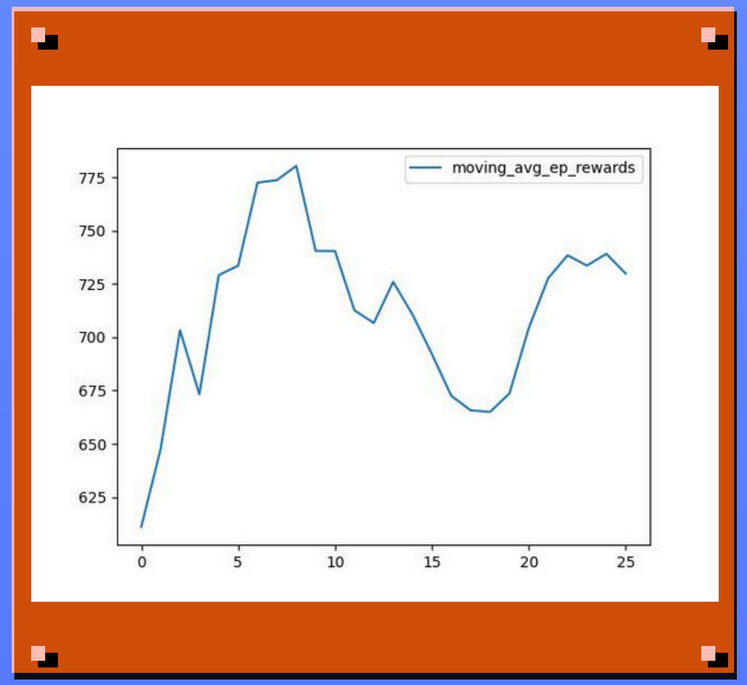
Python

Using Scheduler

Scheduler **Off**



Scheduler **On**



scheduler 적용 시, 보다 안정적으로 학습

Action Space Extention

Origin

0: Right

1: Right + A

Extension

0: Right

1: Right + A

2: Right + B

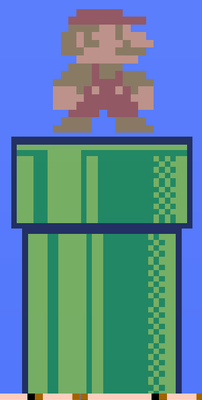
3: Right + A+ B

좀 더 **다양한 action**을 사용하도록 확장

Action Space Extention

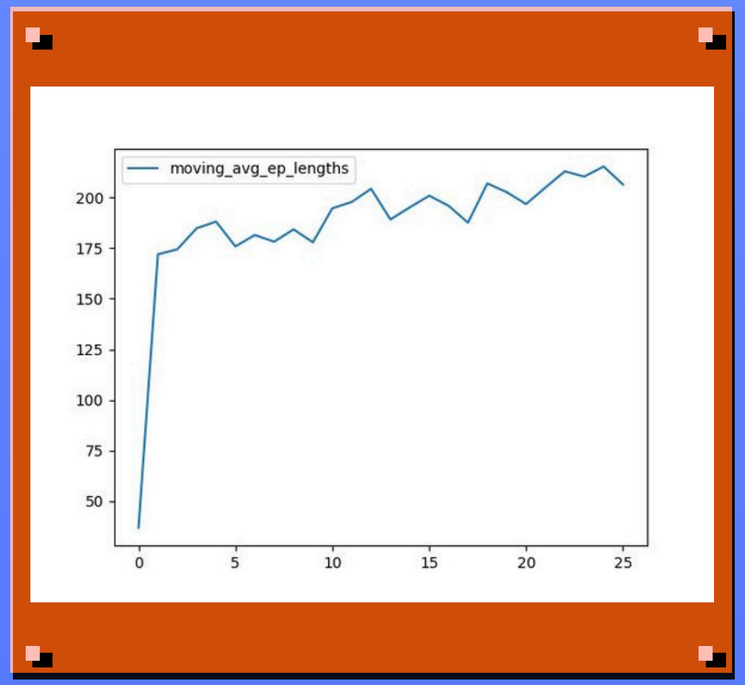
```
#   0. 오른쪽으로 걷기
#   1. 오른쪽으로 점프하기
env = JoypadSpace(env, [["right"], ["right", "A"]])
```

```
#   0. 오른쪽으로 걷기
#   1. 오른쪽으로 점프하기
#   2. 오른쪽 + 대시
#   3. 오른쪽 + 점프 + 대시
env = JoypadSpace(env, [
    ["right"], ["right", "A"], ["right", "B"], ["right", "A", "B"]
])
```

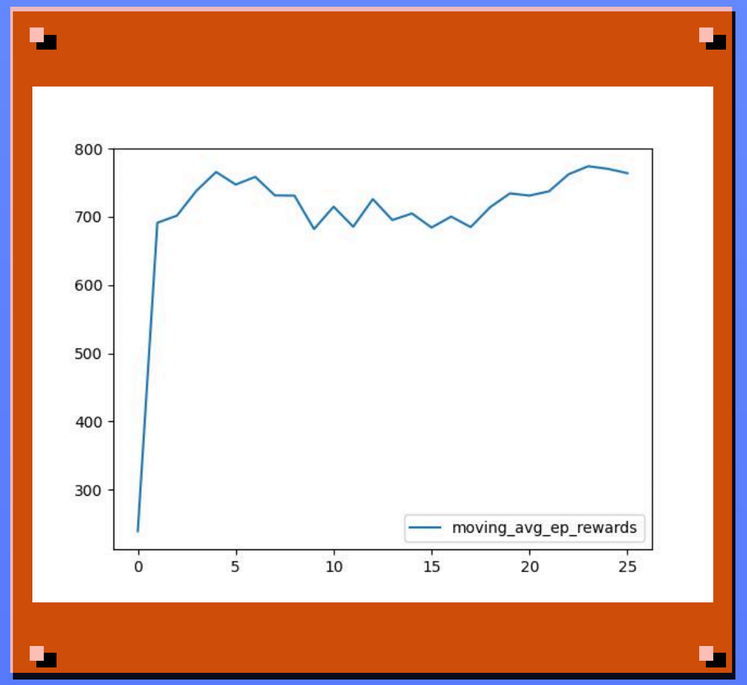


Action Space Extention

Distance



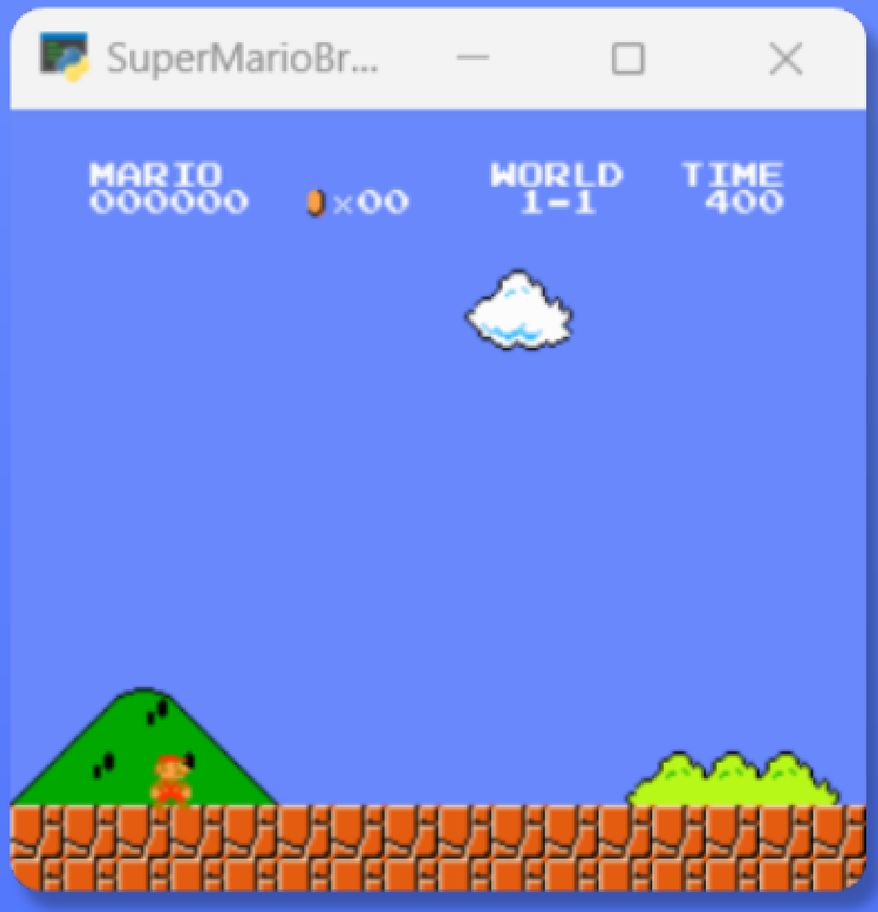
Reward



다양한 action 사용 시, 보다 **안정적**으로 학습됨을 확인

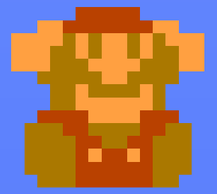

DEMONSTRATION
VIDEO

Demonstration video



DISCUSSION

Limit



보유 자원의 부족으로 인한 학습의 한계

Takeaway

But. 프로젝트를 진행하면서...

- agent와 environment의 상호작용
- agent가 학습하는 방법
- DDQN 알고리즘의 구조
- 실행 화면의 시각화

GitHub Link : https://github.com/maeng99/Super-Mario-RL

THANKS FOR WATCHING