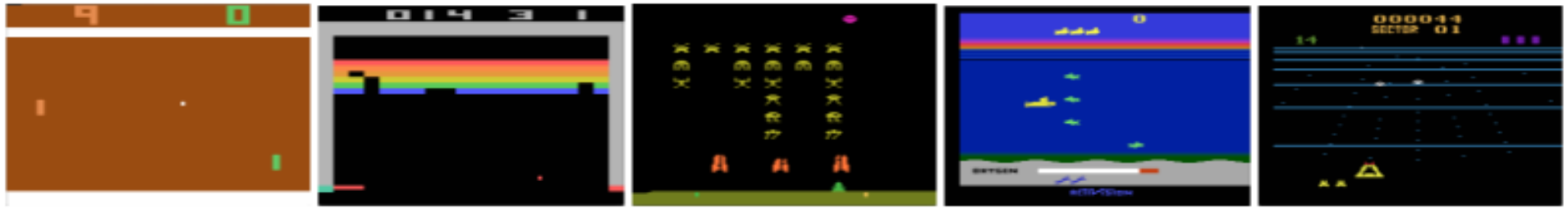
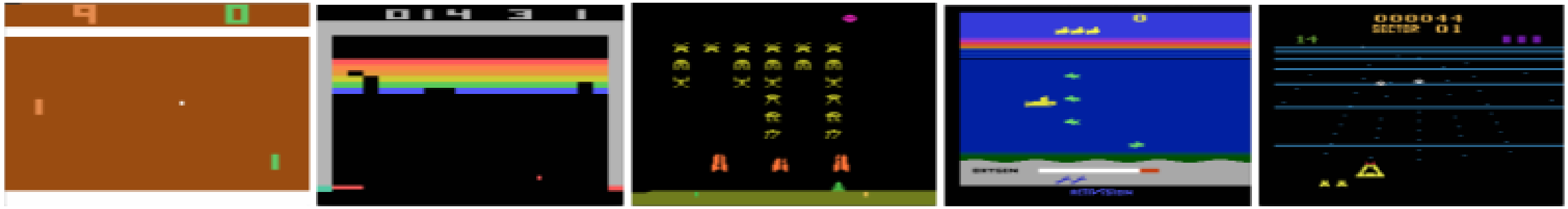
Playing Atari with Deep Reinforcement Learning

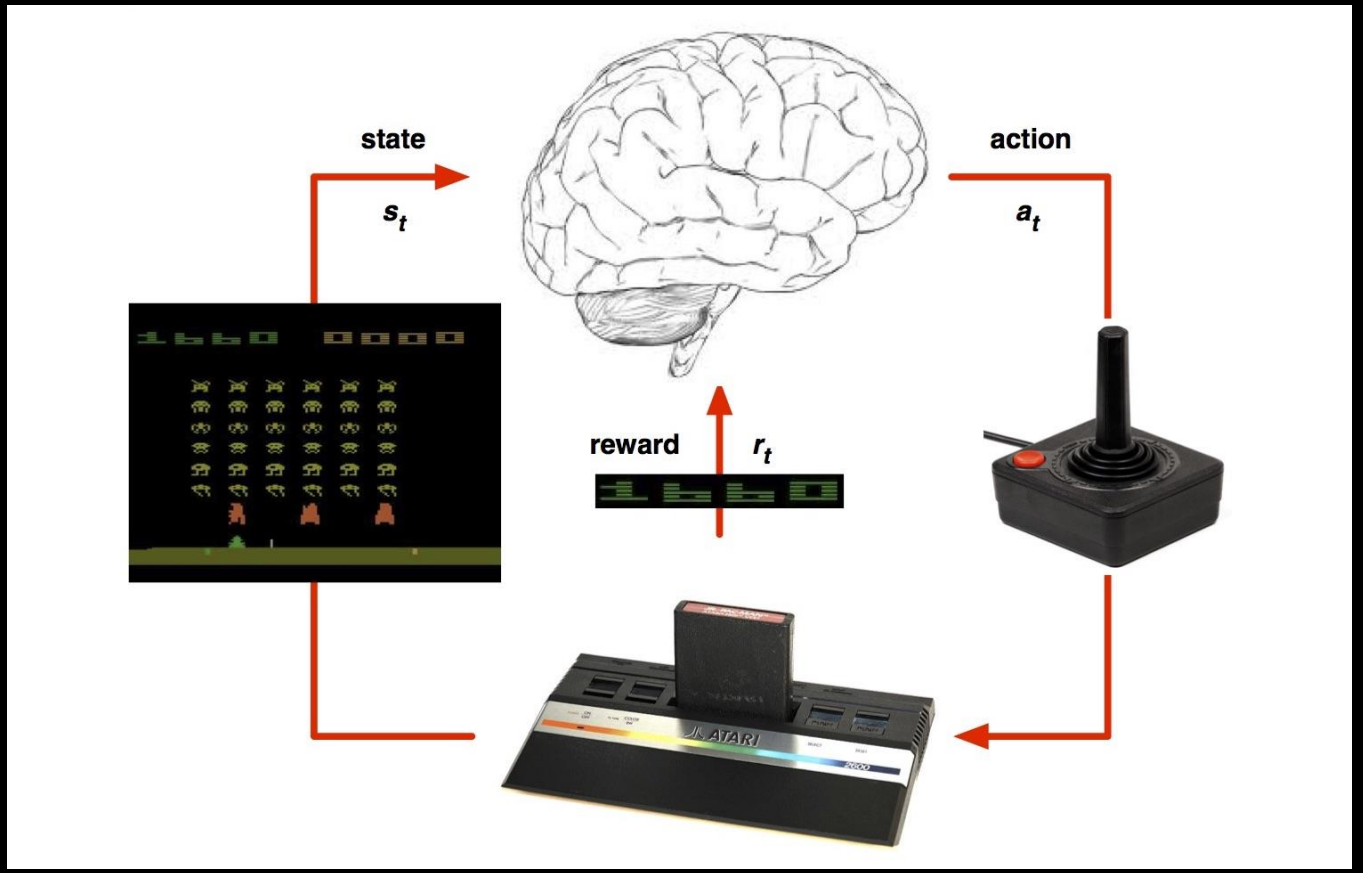


1. Introduction

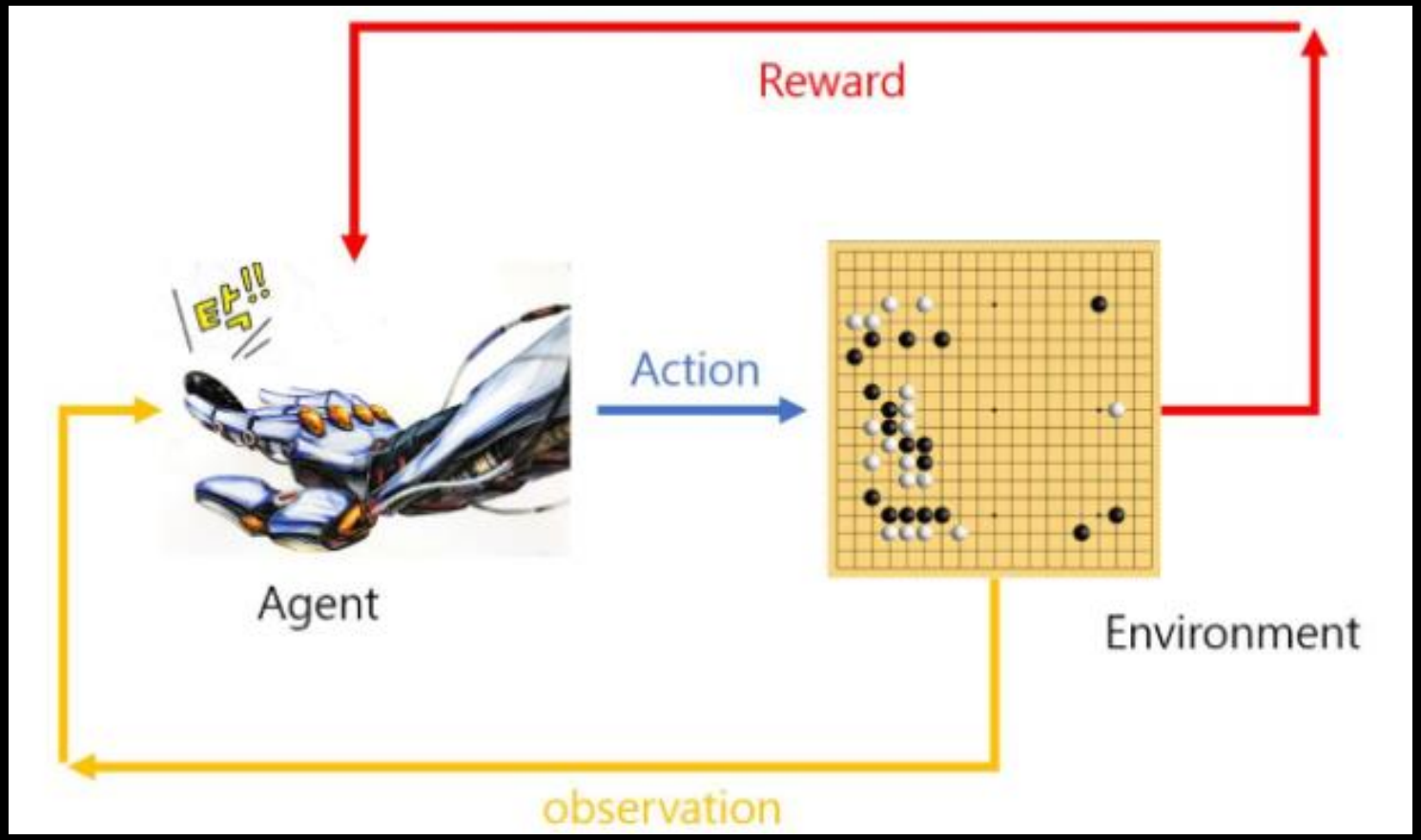


Visual Input: (210 × 160 RGB video at 60Hz)

1. Introduction



1. Introduction



2. Background

[Q-Value 한장 요약]

Part 01

Base Knowledge

[Q-Value Function]

- ✓ 어떤 상태 s에서 어떤 행동 a를 했을 때, 그 행동의 가치를 나타내는 Q(s,a) 함수를 사용
- ✓ 미래의 보상을 계산하기 위해 0~1 사이의 γ(Discount Factor)를 사용
- ✓ 현재 상태로부터 Δt 시간이 흐른 후에 얻는 보상r은 $\gamma^{\Delta t}$만큼 Discounted된 $r * \gamma^{\Delta t'}$
- ✓ Q-Value는 어떤 시간 t에서 전략 π를 따라 행동 a를 할 때 미래의 보상들의 총합의 기대값

$$Q_\pi(s,a) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s, A_t = a]$$

2. Background

[Q-Learning Algorithm 한장 요약]

Part 01

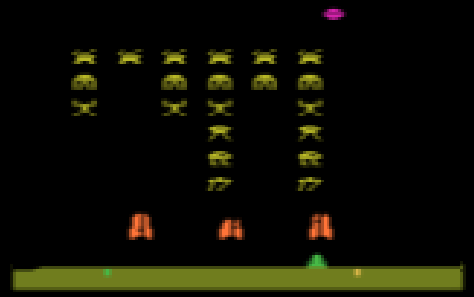
Base Knowledge

[Q-Learning]

- ✓ FMDP에서 Agent가 Expected sum of future reward를 극대화하도록 Policy를 학습하는 것
- ✓ Q는 고정된 값으로 시작하여, Agent의 Action a_t으로 얻은 Reward r_t를 통해 갱신됨
- ✓ 이전의 값과 새로운 정보의 Weighted Sum을 이용하는 Value Iteration 기법

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Bigg(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \Bigg)$$

3. DQN



[Q-Value 한장 요약]

Part 01

Base Knowledge

[Q-Value

✓ 어떤 상태 s에서 어떤 행동 a

치를 나타내는 Q(s,a) 함수를 사용

✓ 미래의 보상을 계산하기

)를 사용

✓ 현재 상태로부터 Δ

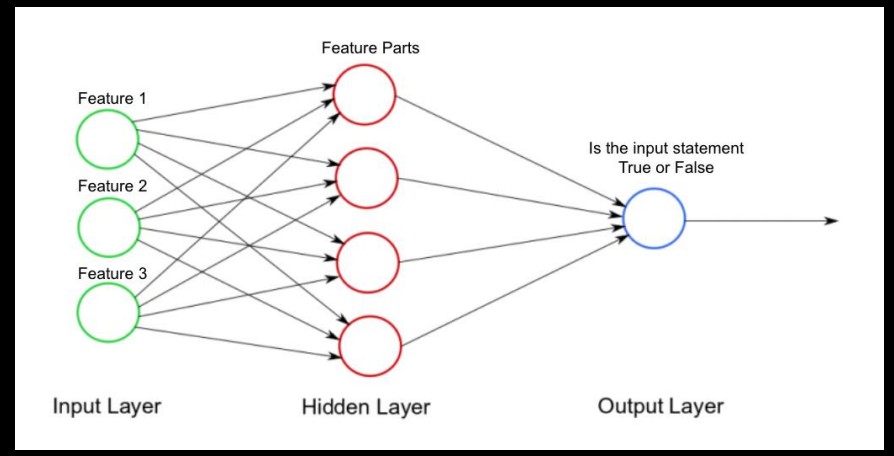
ounted된 $r * \gamma^{\Delta t'}$

✓ Q-Value는 어떤 시간 t

를 따라 행동 a

래의 보상들의 종합의 기대값

$$Q_\pi(s,a) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s, A_t = a]$$



3. DQN

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}.$$

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[R_t \middle| s_t = s, a_t = a, \pi\right].$$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma Q^*(s',a') \mid s,a\right].$$

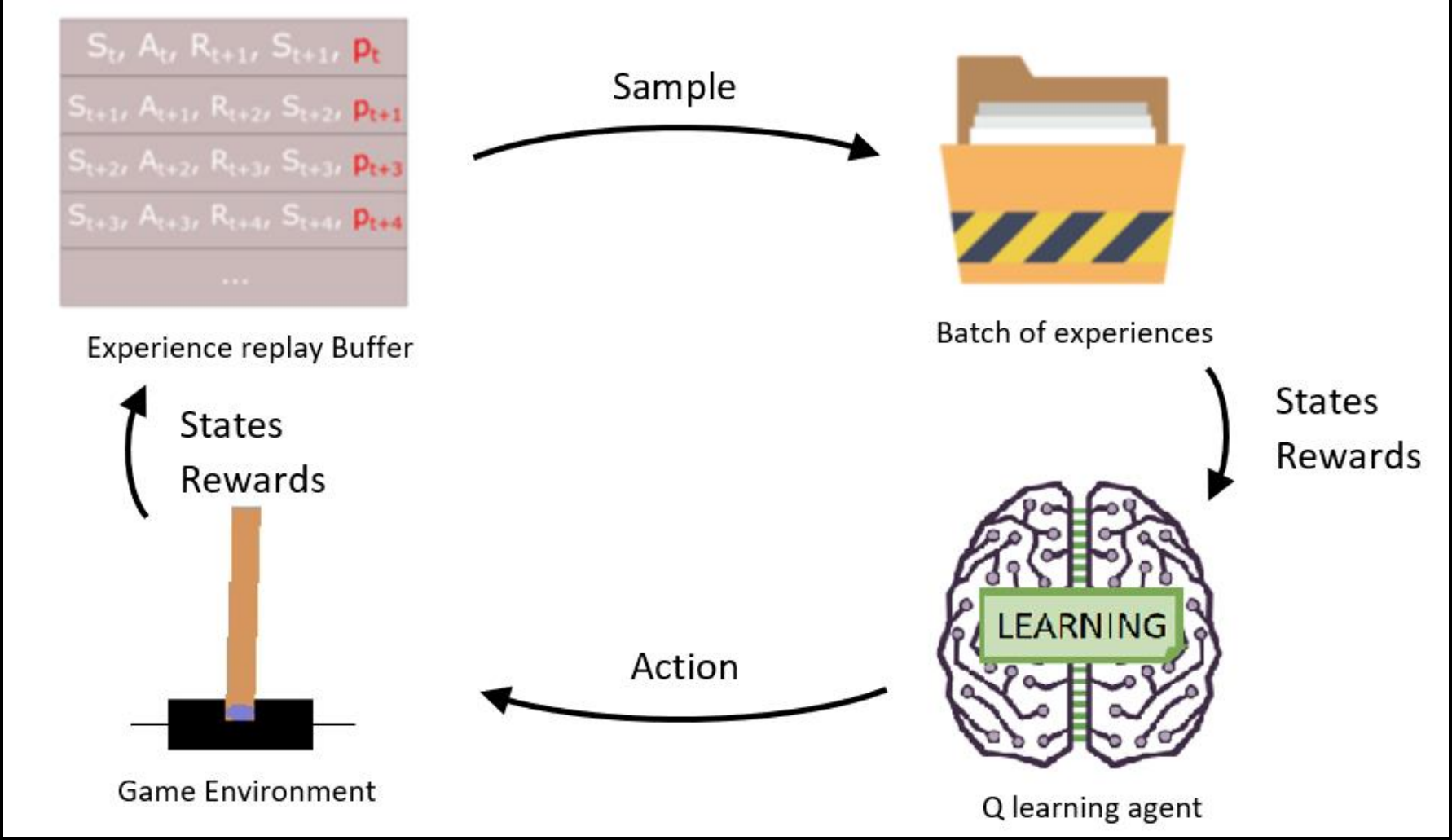
3. DQN

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

$$\text{where, } y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';w_{i-1}) \mid s,a\right].$$

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right].$$

Experience Replay



Target Network

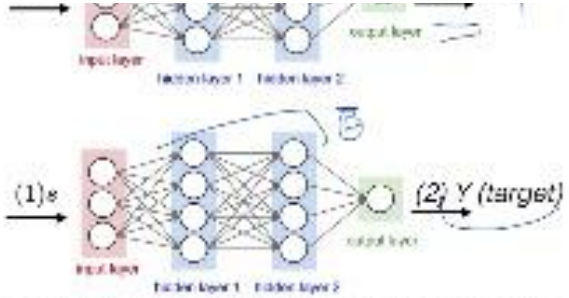
DQN의 해결방안

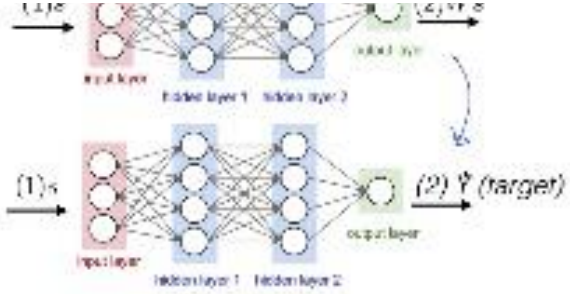
Target Network가 현재 학습과정에서 영향을 받지 않도록 분리

Solution 3: separate target network

Solution 3: copy network

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$





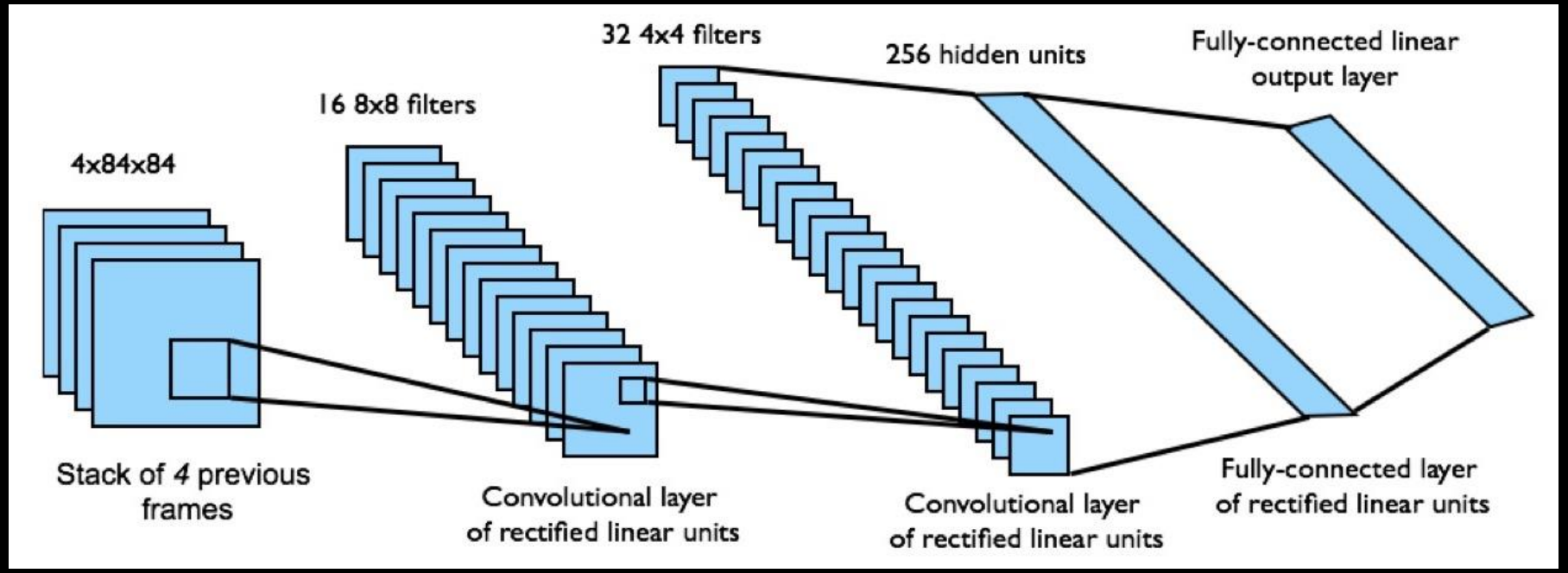
- Target(Y)의 Θ (Weight)와 예측값(Y^)의 Θ를 분리한다
- 학습할때는 예측값의 Θ만 업데이트하고,
- 일정한 시간 후에 target의 Θ 에 복사한다.

3. DQN

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  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

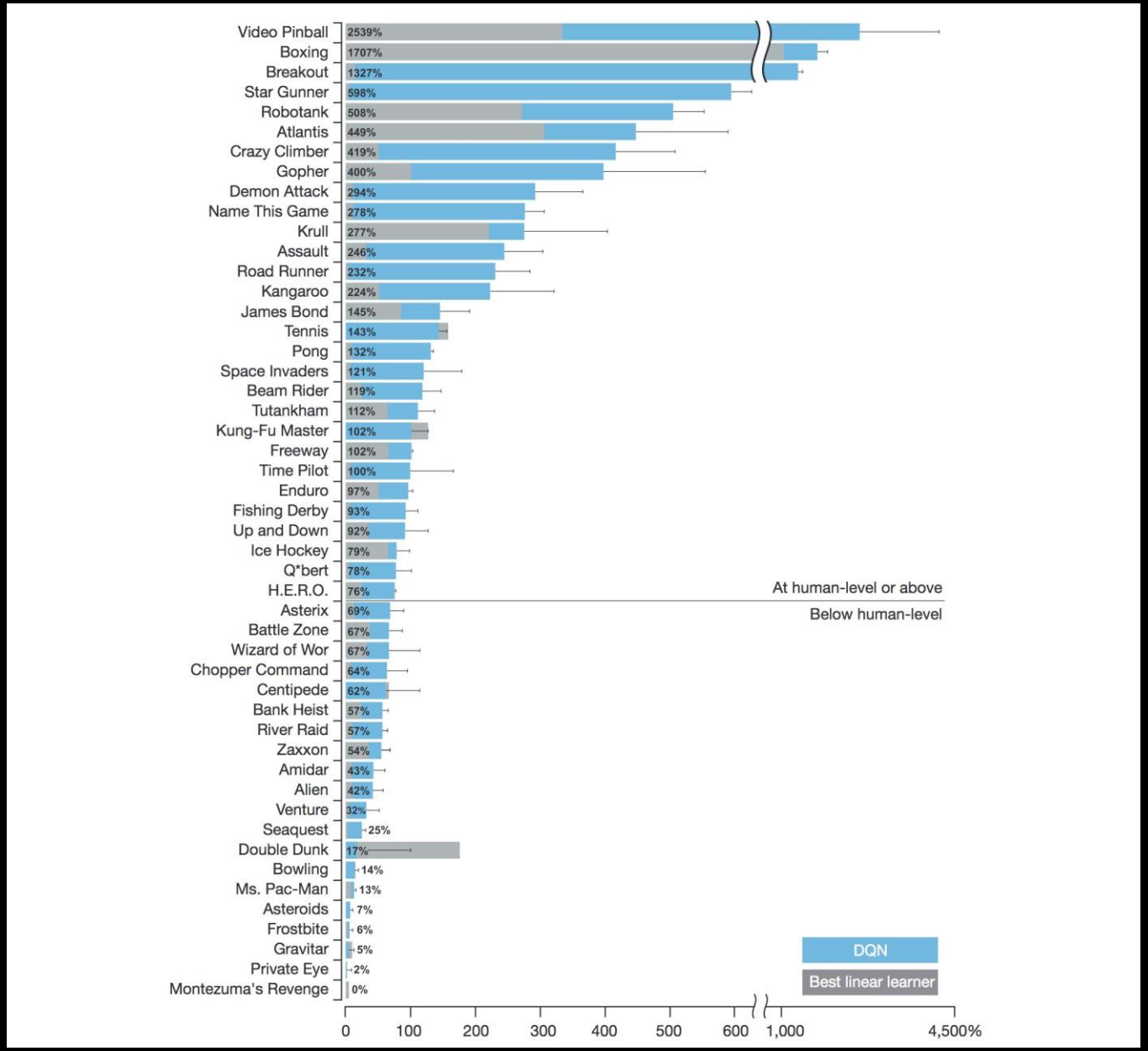
4. Result


32 4x4 filters
256 hidden units
Fully-connected linear output layer
16 8x8 filters
4x84x84
Stack of 4 previous frames
Convolutional layer of rectified linear units
Convolutional layer of rectified linear units
Fully-connected layer of rectified linear units

4. Result

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.



References

- https://mangkyu.tistory.com/61
- https://jamiekang.github.io/2017/05/07/playing-atari-with-deep-reinforcement-learning/
- https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
- http://sanghyukchun.github.io/90/