DQN Paper Review

2015, Human-level control through deep reinforcement learning
(피인용 4394회)

[쉽게 읽는 강화학습 논문 3화]

팡요랩

2019.01.06

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

The theory of reinforcement learning provides a normative account[1], deeply rooted in psychological[2] and neuroscientific[3] perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations. Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems[4,5], the former evidenced by a wealth of neural data revealing notable parallels between the phasic signals emitted by dopa-

agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a) (see Methods)[19].

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function[20]. This instability has several causes: the correlations present in the sequence

DQN 복습 – Table Lookup

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

DQN 복습 – Table Lookup

Off-Policy Control with Q-Learning

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned}
& R_{t+1} + \gamma Q(S_{t+1}, A') \\
=& R_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')) \\
=& R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')
\end{aligned}$$

DQN 복습 – With function approx.

Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector $\mathbf{w}$ minimising mean-squared error between approximate value fn $\hat{v}(s, \mathbf{w})$ and true value fn $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2\right]$$

- Gradient descent finds a local minimum

$$\begin{aligned}\Delta\mathbf{w} &= -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) \\ &= \alpha\mathbb{E}_\pi\left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla_\mathbf{w}\hat{v}(S, \mathbf{w})\right]\end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta\mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla_\mathbf{w}\hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

DQN 복습 – With function approx.

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

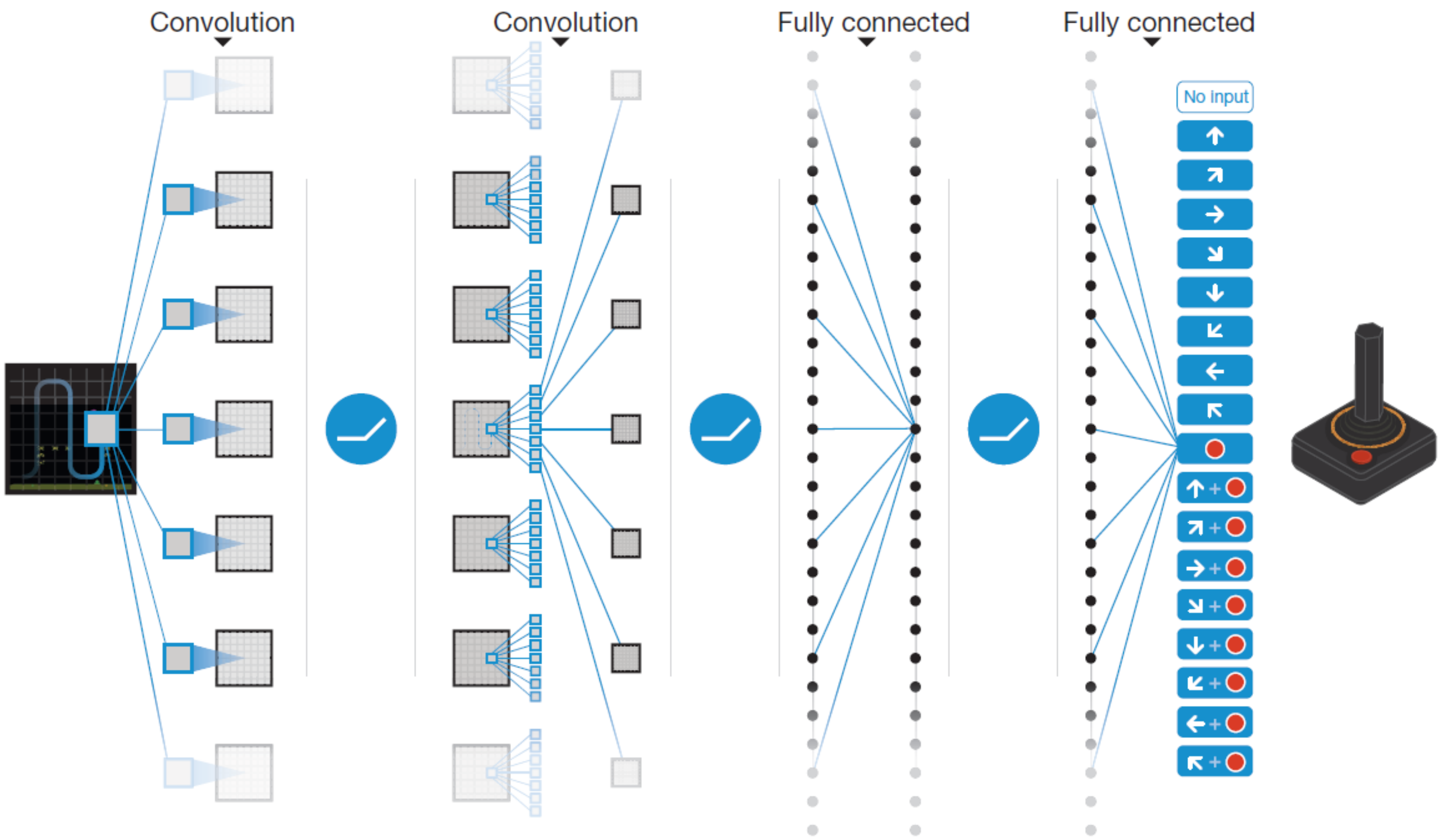
  - For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

아타리 게임에서의 Agent



학습 방법

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

$$\nabla_{\theta_i} L(\theta_i) \quad = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q\left(s',a';\theta_i^-\right) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

CNN 으로 Q 함수 agent 만들어서, 시뮬레이션 돌려서 나온 데이터로 학습하자. 끝!

학습의 불안정성

1. 강화학습에서는 nonlinear function approximator(ex : 뉴럴넷)을 사용하여 학습하면 학습이 불안정하고, 심지어는 발산하는 성질이 있는 것으로 알려져 있음.
2. 그 이유는 observation 의 시퀀스에 있는 correlation 때문.
3. Q 함수를 미세하게 수정 하였는데 policy는 급격하게 변할 수 있음.
4. 이는 데이터의 분포를 급격하게 바꾸며
5. action-values (Q) 와 target values $r+\gamma \max_{a'} Q(s', a')$ 사이의 관계도 급격하게 바꿈.

학습의 불안정성을 해결하는 아이디어

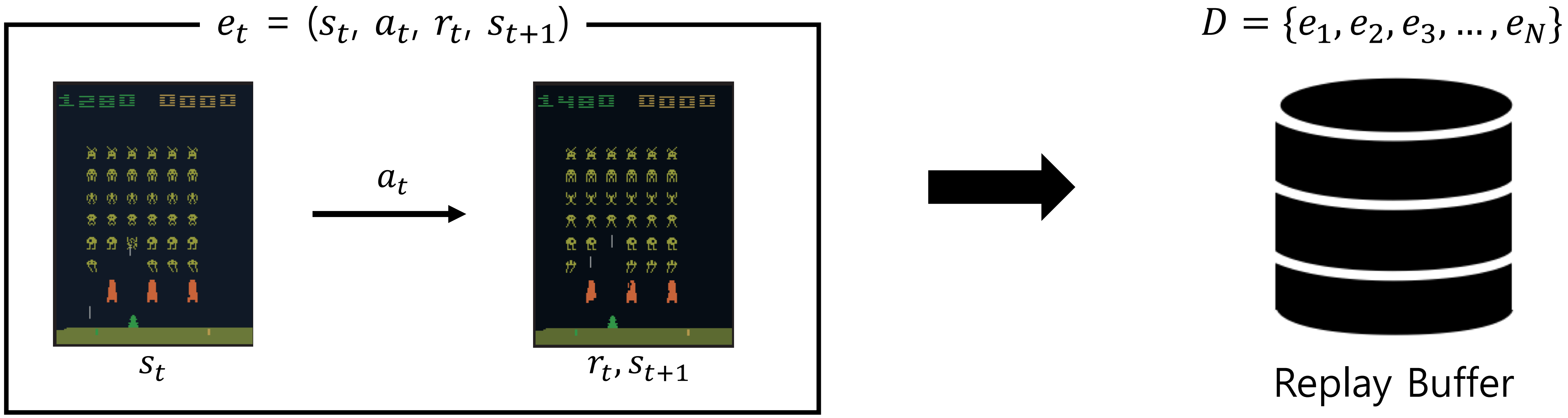
1. 강화학습에서는 nonlinear function approximator(ex : 뉴럴넷)을 사용하여 학습하면 학습이 불안정하고, 심지어는 발산하는 성질이 있는 것으로 알려져 있음.
2. 그 이유는 observation 의 시퀀스에 있는 correlation 때문.
3. Q 함수를 미세하게 수정 하였는데 policy는 급격하게 변할 수 있음.
4. 이는 데이터의 분포를 급격하게 바꾸며
5. action-values (Q) 와 target values $r + \gamma \max_{a'} Q(s', a')$ 사이의 관계도 급격하게 바꿈.

↓

Experience Replay

Target Network

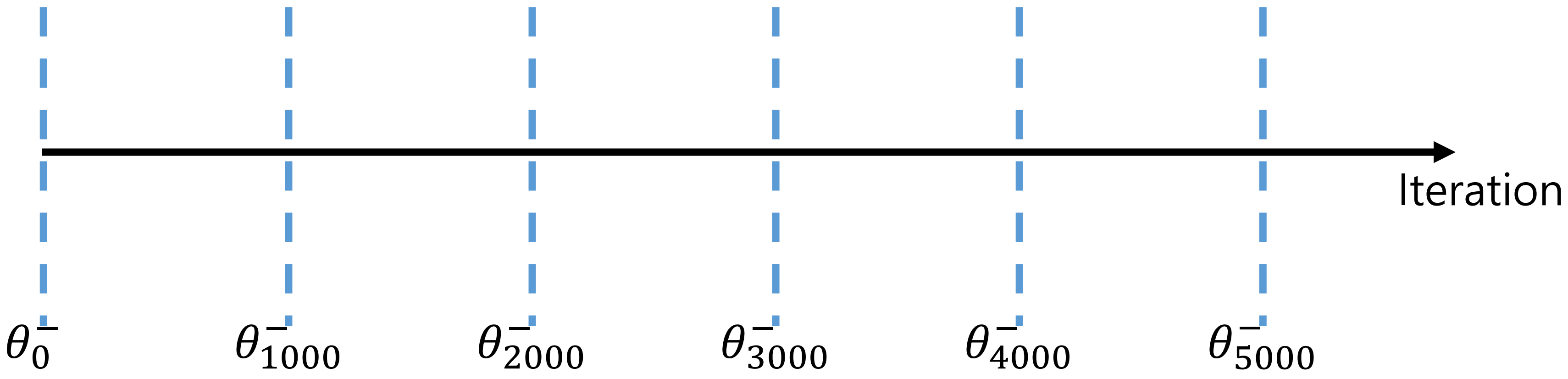
Experience Replay



1. 시뮬레이션을 돌리면서 매 틱마다 생성되는 Transition 튜플 (s_t, a_t, r_t, s_{t+1})을 Replay Buffer 에다가 저장해 놓는다.
2. Replay Buffer는 자동으로 가장 최신의 100만개의 튜플만 갖고 있는다.
3. 학습 시에는 이 100만개 중에서 uniform random 하게 임의로 32 개를 뽑아서, minibatch 를 구성하여 학습한다.

randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution

Target Network



1. The target network parameters are only updated with the Q-network parameters every C steps and are held fixed between individual updates

2. $Q_{\theta_i}(s,a)$함수를 학습할 때, i 번째 iteration 에서의 네트워크 파라미터를 θ_i 라 하자.

3. Target 값 $r+\gamma \max_{a'} Q(s', a')$ 을 계산할 때 쓰이는 $Q_{\theta_i}(s,a)$ 값에서 θ_i 를 고정! 해놓고 계산. 당분간 고정되어 있다 해서 위에 마이너스를 붙여서 표기함. => θ_i^-

학습 Detail

1. 원래 Input size는 210*160 인데 84*84 로 크기를 줄였다.
2. Input에 들어오는 이미지는 가장 최근 4장을 쌓아서 제공하였다. 즉 4*84*84 가 input.
3. 모델 구조는 3개의 conv layer 이후 2개의 fc layer로 구성되었다.
4. Atari 2600의 49개의 게임에 대해서 동일한 모델 구조, 동일한 hyperparameter를 사용하여 학습하였다.
5. 게임마다 reward scale이 달라서 모든 양의 리워드는 +1, 모든 음의 리워드는 -1로 clipping 하였다.
6. Behaviour policy는 $\varepsilon - greedy$를 썼으며 ε은 1.0에서 시작하여 100만 frame 동안 0.1까지 선형으로 줄어든다. 이후는 0.1로 고정.
7. 총 5천만 frame 을 학습에 썼다(게임 플레이 시간으로 38일).
8. Replay memory 는 최신 100만 frame을 사용.

학습 Detail

9. Frame skip = 4를 사용하였다. 한 번 decision 하면 최대 4번까지 같은 decision을 계속 보내는 방식. Agent가 같은 시간동안 거의 4배 많은 게임을 할 수 있게 해준다.

Hyperparameter	Value	Description
minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
replay memory size	1000000	SGD updates are sampled from this number of most recent frames.
agent history length	4	The number of most recent frames experienced by the agent that are given as input to the Q network.
target network update frequency	10000	The frequency (measured in the number of parameter updates) with which the target network is updated (this corresponds to the parameter *C* from Algorithm 1).
discount factor	0.99	Discount factor gamma used in the Q-learning update.
action repeat	4	Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.
update frequency	4	The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.
learning rate	0.00025	The learning rate used by RMSProp.
gradient momentum	0.95	Gradient momentum used by RMSProp.
squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
min squared gradient	0.01	Constant added to the squared gradient in the denominator of the RMSProp update.
initial exploration	1	Initial value of ε in ε-greedy exploration.
final exploration	0.1	Final value of ε in ε-greedy exploration.
final exploration frame	1000000	The number of frames over which the initial value of ε is linearly annealed to its final value.
replay start size	50000	A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.
no-op max	30	Maximum number of “do nothing” actions to be performed by the agent at the start of an episode.

Pseudo Code

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

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

    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 D

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

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

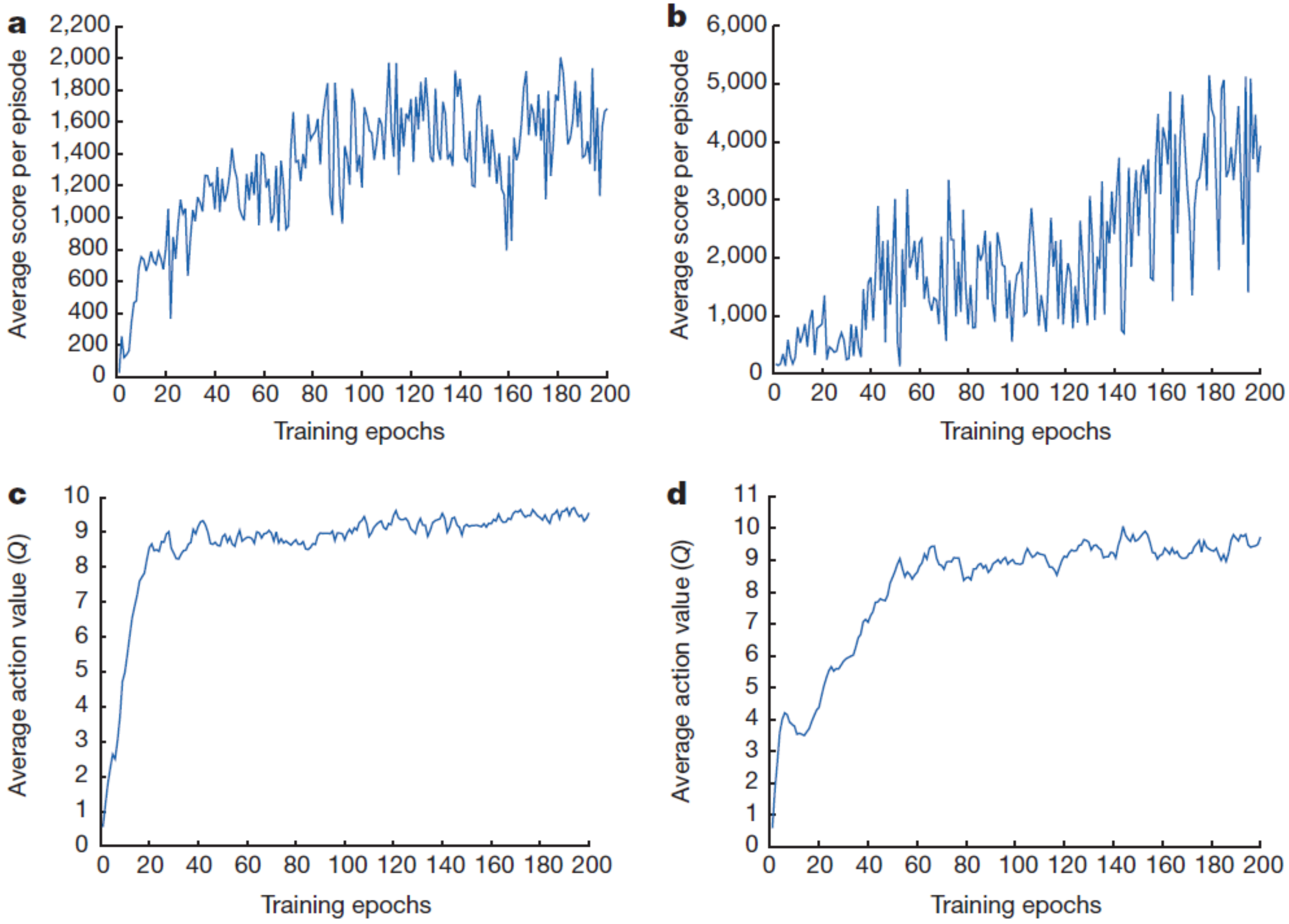
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

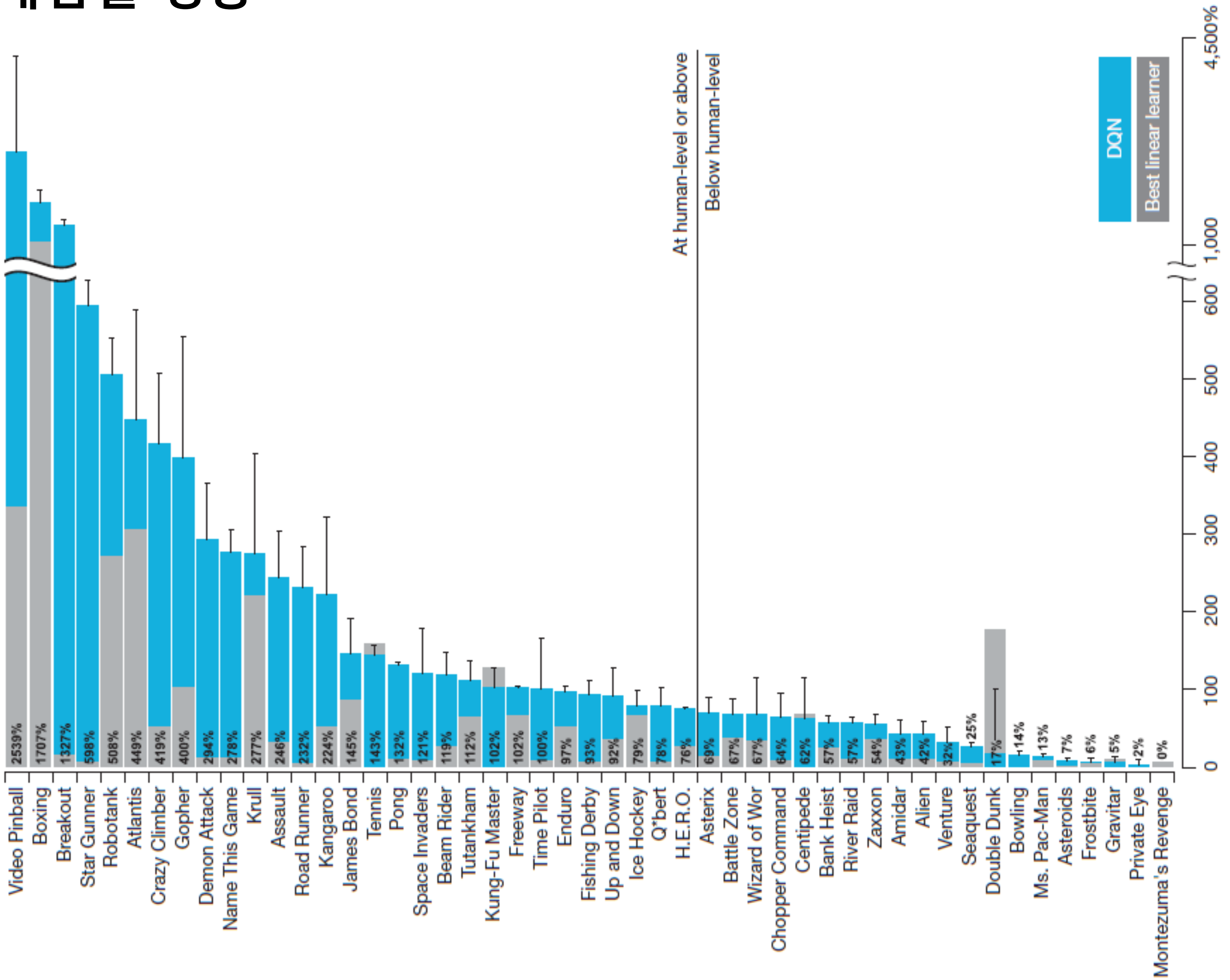
End For

학습 곡선



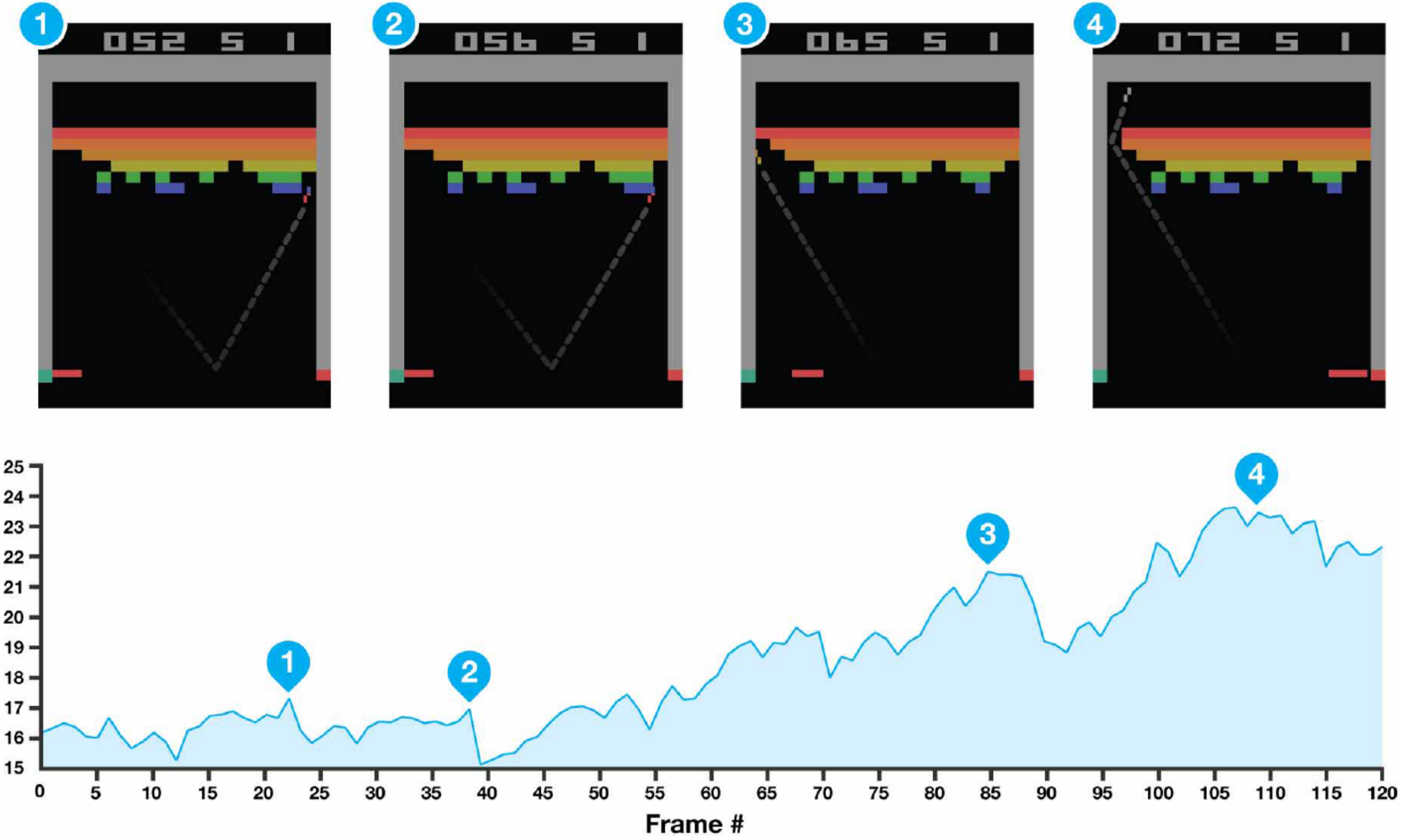
- a,c 는 space invade에 대한 그래프.
- b,d 는 seaquest 에 대한 그래프.

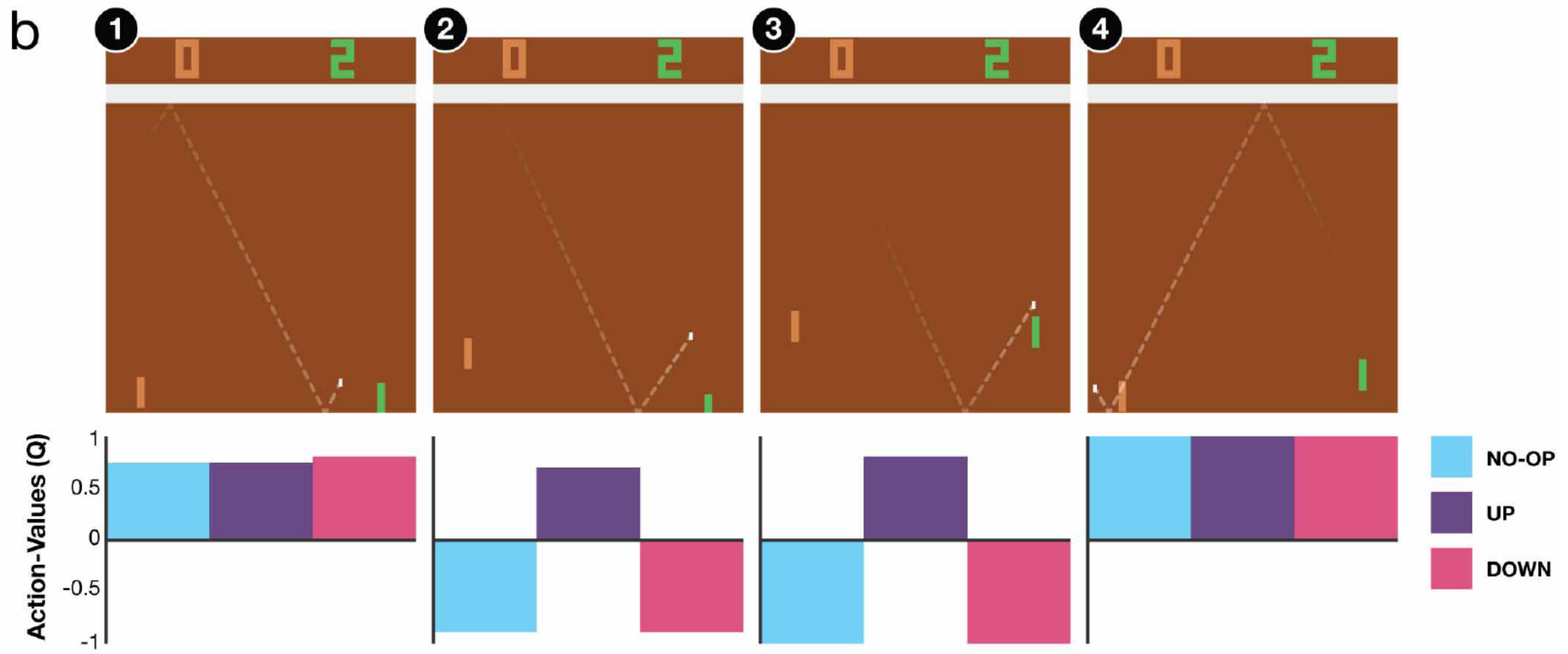
게임별 성능



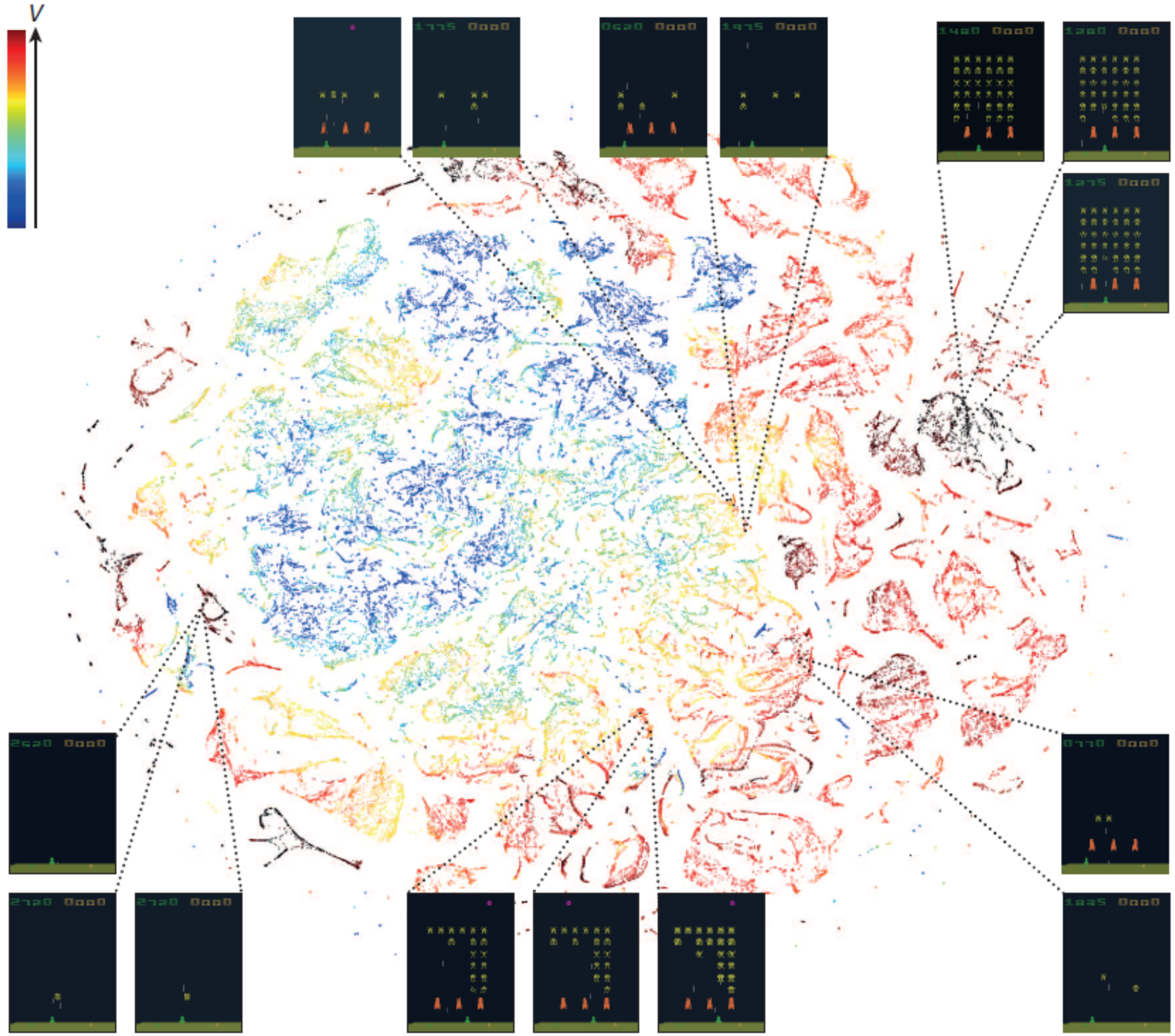
- Random play를 0%
- Professional human player를 100%
- 이때 DQN agent는 몇 %인지 게임별로 측정한 그래프.

a




b
1
2
3
4
0
2
Action-Values (Q)
1
0.5
0
-0.5
-1
NO-OP
UP
DOWN

Representation Visualization



- DQN agent가 2시간동안 play하여 방문한 state들을 기록함.
- 그 state들에 대해 마지막 fc layer에 있는 값들을 t-sne를 이용해 2차원에 plotting.

Replay, Target Q의 기여도 평가

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Neural Net의 기여도 평가

Game	DQN	Linear
Breakout	316.8	3.00
Enduro	1006.3	62.0
River Raid	7446.6	2346.9
Seaquest	2894.4	656.9
Space Invaders	1088.9	301.3