Deep reinforcement learning을 통해 얻어낸 인간 수준의 조작.

동물의 행동에 대한 심리학, 신경학적 접근에 깊은 기반을 두는 강화학습 이론은 agent(행동자)가 어떻게 주변 환경에 대한 통제를 최적화 할 수 있는가에 대한 보편적인 이론을 제공한다. 이 강화학습을 현실 수준의 복잡성을 가진 환경에 노출시킬 경우에 agent는 다양한 입력 신호를 통해 주변 환경을 인식해야 하고, 그 정보들을 통해 과거의 정보를 새로운 상황에 맞춰 보편화 해야 한다. 신기하게도, 사람과 다른 동물들은 이 문제를 강화학습과 계층적 감각 신호 분석 체계의 조화로 해결했는데, 이때 전자는 많은 신경학적 자료에서 나타나는 도파민 분비 뉴런의 간헐적 신호와 temporal difference reinforcement learning algorithm의 유사성으로 대표되고 있다. 한편, 여러 분야에서 성공적인 강화학습의 사례가 보고되었지만, 그 분야들은 환경을 이해하는데 유용한 특징들을 직접 만들어 줄 수 있거나, 환경을 완전히 관측할 수 있으며 간단한 수준의 정보만이 존재하는 분야들이었다. 여기서 우리는 최근 발전한 DNN 훈련법을 이용해 단대단 강화학습을 사용하여 다양한 입력 정보를 통해 행동 정책을 곧바로 학습할 수 있는 deep Q-network로 불리는 새로운 agent를 개발했고, 이 agent는 고전 아타리 게임 2600개로 시험되었다. 이 논문에서는 오직 화면 픽셀과 게임 점수만을 입력값으로 받는 deep Q-network가 현존하는 모든 알고리즘의 성능을 뛰어넘어 49개 게임에서 동일 알고리즘과 네트워크 구조, 하이퍼 파라메터를 사용해도 프로게이머 수준의 조작 능력을 갖춘 것을 확인했다. 이 연구는 고차원 입력 데이터와 행동간에 존재해 왔던 깊은 간극을 메울 수 있으며, 여러가지 어려운 분야에서 성공할 수 있는 최초의 인공 agent를 선보인다.

우리는 전 연구들에서는 잡아내지 못했던 보편 인공지능의 목표인 여러가지 상황에 대해 다양한 수준의 적응성을 가지는 단일 알고리즘을 만드는 작업에 착수했다. 이를 완수하기 위해서는 새로이 강화학습과 레이어를 거치며 점차 더욱 복잡한 특징값들을 추출해내어 미가공 자료로부터 물체 분류등의 작업을 학습할 수 있게 해주는 Deep Nural Network(DNN)을 같이 사용할 수 있는 deep Q-network(DQN)이라고 불리는 알고리즘을 설계했다. 이때 우리는 특별히 성공적이었던 Deep Convolutional Network라는 Huble 과 Wiesel의 세미나에서 언급된 대뇌의 시각 피질이 작동하는 원리에서 영감을 얻은 콘볼루션 레이어를 쌓아올려 시각 인식 과정을 모방하는 아키텍쳐를 사용했다. 이는 이미지 내의 국부간 연관성을 사용할 수 있게 해주었고, 시야의 변화 또는 크기의 변화와 같은 자연적인 변화에 내성을 지니게 해줬다.

우리는 agent가 작업을 수행하는 과정을 관측, 수행 및 보상의 과정으로 보았다. 이때 agent의 목표는 일련의 선택을 통해 가장 큰 보상이 돌아오는 행동을 수행하는것이다. 조금 더 학

술적으로 기술하자면, 우리는 Convolutional Neural Network(CNN)을 이용해 아래 'optimal action value'함수를 근사했다.

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

이 함수는 보상의 최대값인 r_t 가 공비 γ에 의해 t회만큼 깎이는 구조이며, 최대 r_t값은 행동 정책 π = P(a|s)를 통해(s=관측, a=행동) 도달할 수 있다.

강화학습은 흔히 인공 신경망이 행동값(위 식의 Q)함수를 나타내게 될 경우 불안정하거나, 심지어 발산하는 모습까지 보인다고 알려져 있다. 이는 크게 관측 시퀀스간 존재하는 연관성, 자그마한 Q에 대한 업데이트도 행동 정책에 큰 변화를 초래해 데이터 분포를 바꿔버릴 수 있는 가능성, 행동값(Q)과 표적값(r + γ maxQ(s', a'))과의 연관성등이 있다. 우리는 이를 해결하기 위해 생물체에게서 영감을 얻은 데이터를 랜덤하게 섞어버려서 관측시점간 연관성을 없애고, 데이터 분포에 따른 변화를 무듸게하는 Experince replay라는 개념과 Q값을 간헐적으로 업데이트 되는 표적값과 연속적으로 가깝게 조정하여 표적과의 연관성을 줄이는 두가지의 개념을 사용했다.

여타 neural fitted Q-iteration 과 같이 다른 안정적인 Q-learning의 경우 동일한 문제를 네트워크를 수백 이터레이션씩 반복해서 훈련시키며 문제를 해결하지만 이는 큰 인공신경망을 사용하기에는 과도히 비효율적이다. 우리는 행동값 함수의 근사 함수($Q(s,a;\theta_i)$ (θ_i = network weight at iteration i of Q-network))를 그림 1에서 나온 방식을 통해 파라미터화 했다. 이때 우리는 Exeprience replay를 하기 위해 agent의 '경험'자료를 $e_t = \{e_1 \ldots e_t\}$형식으로 시간 스텝 t에 맞춰 $D_t = \{e_1 \ldots e_t\}$ 형식의 데이터 세트로 만들었다.

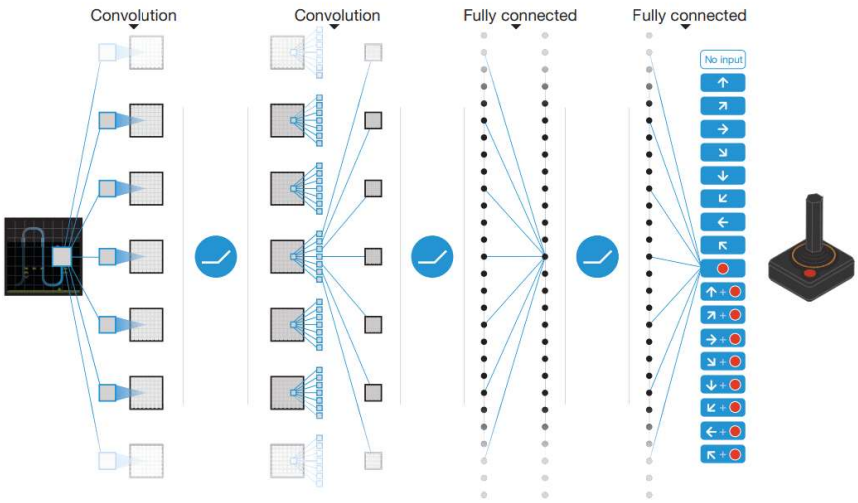


그림 1

학습을 진행하는 과정에서는 (s,a,r,s')~U(D)속하는 '경험'샘플에 대해 그림 2의 식을 따르는 Q-

learning update를 진행했다.

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

그림 2

그림 2에서 γ는 agent의 horizon(?)을 규정하는 0과 1 사이의 값이고, θ_i는 i회 반복된 상황에서의 파라미터값, θ_i^-는 C스텝마다 업데이트 되며, 업데이트가 되지 않았을 경우 상수로 유지되는 목표 네트워크 파라미터값이다.

우리의 DQN agent를 테스트하기 위해 우리는 49개의 다양한 목표를 제시하는 아타리 2600 플랫폼을 사용했다. 이 테스트 과정에서는 이 연구에서의 접근방법이 오직 입력 데이터가 이미지 데이터라는 사실과 각 게임마다 가능한 조작 목록만으로도 각 게임에서의 성공적인 행동정책을 학습할 수 있다는 사실을 보이기 위해 동일한 네트워크 구조, 하이퍼 파라메터와 학습 절차가 사용되었다. 특기할만하게도, 우리의 방법은 그림 3에서 볼 수 있듯 무거운 뉴럴 네트워크를 강화학습 신호와 SGD(Stocastic Gradient Descent)를 사용하여 훈련시킬 수 있었다.

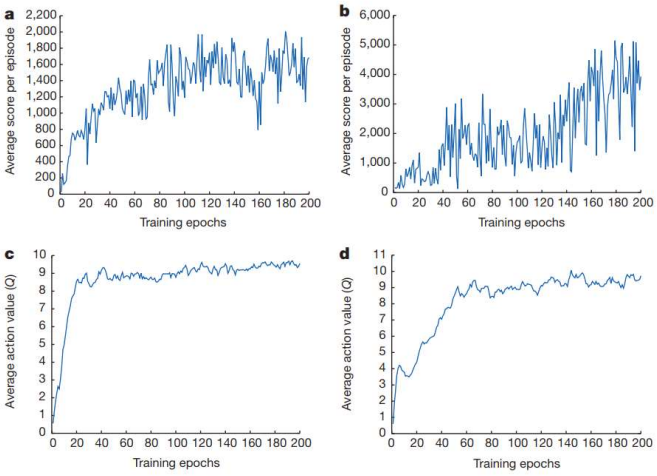


Figure 2 | Training curves tracking the agent's average score and average predicted action-value. **a**, Each point is the average score achieved per episode after the agent is run with ε-greedy policy ($\varepsilon = 0.05$) for 520 k frames on Space Invaders. **b**, Average score achieved per episode for Seaquest. **c**, Average predicted action-value on a held-out set of states on Space Invaders. Each point on the curve is the average of the action-value Q computed over the held-out set of states. Note that Q-values are scaled due to clipping of rewards (see Methods). **d**, Average predicted action-value on Seaquest. See Supplementary Discussion for details.

그림 3

이 연구에서 개발된 DQN은 결과값이 공개된 강화학습 관련 문헌에 언급된 가장 높은 점수를 기록한 방법과 비교되었으며, 통제된 환경에서 게임을 진행한 프로 테스터들의 점수와도 비교되었고, 또한 행동정책을 완전히 무작위로 결정하는 경우와도 비교되었는데, DQN은 이 중 가장 성능이 좋았던 강화학습 agent를 여타 방식에서 사용되었던 아타리 2600게임에 대한

선행지식 없이도 43개 게임에서 능가했으며 심지어 29개 게임에서는 프로 테스터들의 기록한 점수의 75%이상을 기록하는 등, 인간과 비교될만한 수준의 성능을 보였다.

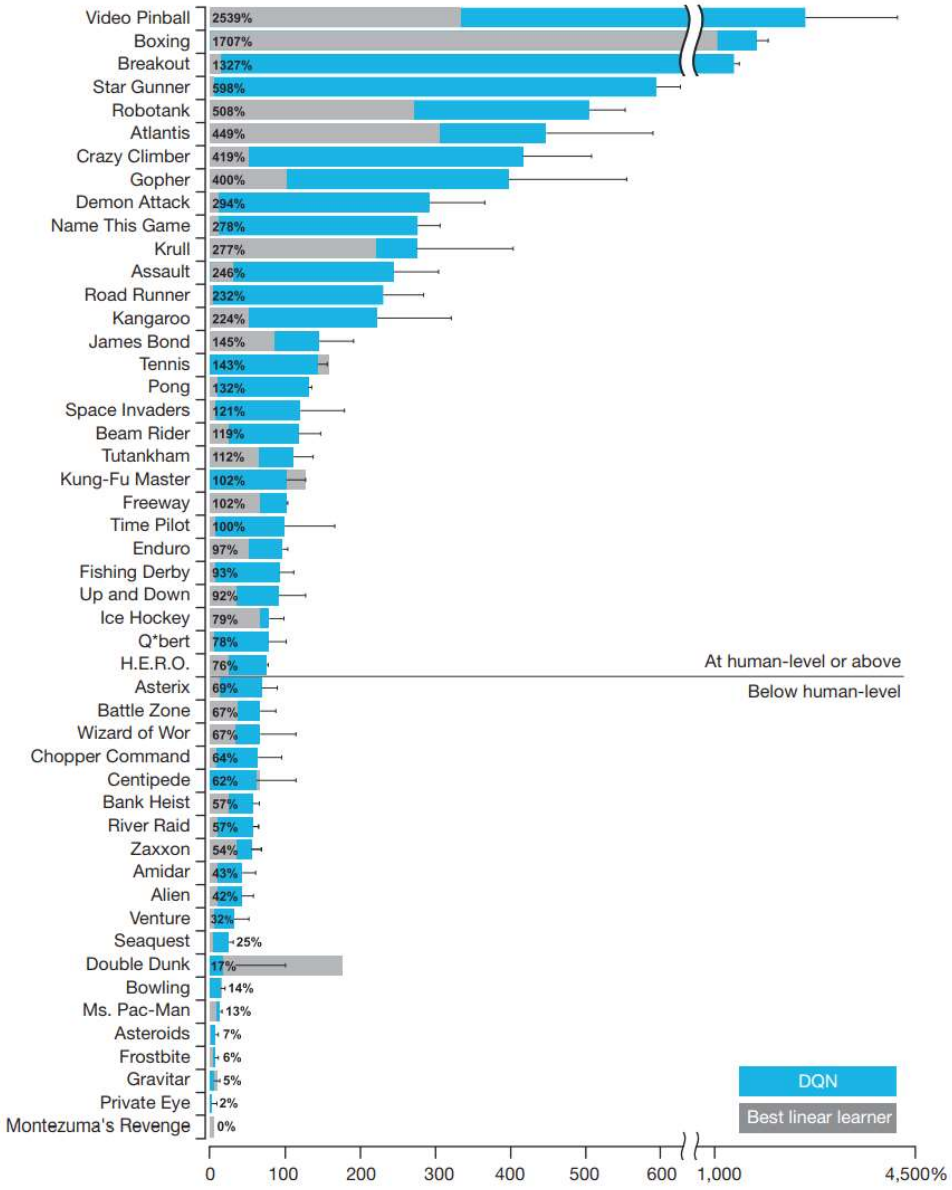


그림 4

여기에 더해 추가적인 실험에서는 DQN agent의 각 부분이 -replay memory, separate target Q-network, deep convloluted network architecture- 얼마나 중요한 역할을 수행하는지 각 부분을 비활성화 시킨 상태에서 성능에 어떤 수준의 지장이 초래되는지 확인해 봤다. (그림 5 참조)

Extended Data Table 3 | The effects of replay and separating the target Q-network

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

DQN agents were trained for 10 million frames using standard hyperparameters for all possible combinations of turning replay on or off, using or not using a separate target Q-network, and three different learning rates. Each agent was evaluated every 250,000 training frames for 135,000 validation frames and the highest average episode score is reported. Note that these evaluation episodes were not truncated at 5 min leading to higher scores on Enduro than the ones reported in Extended Data Table 2. Note also that the number of training frames was shorter (10 million frames) as compared to the main results presented in Extended Data Table 2 (50 million frames).

그림 5

우리는 그 뒤에 고차원 데이터를 시각화하기 위해 개발된 t-SNE로 불리는 기술을 사용하여 Space Invaders를 학습한 DQN에서 나타나는 표지값을 확인해 봤다.

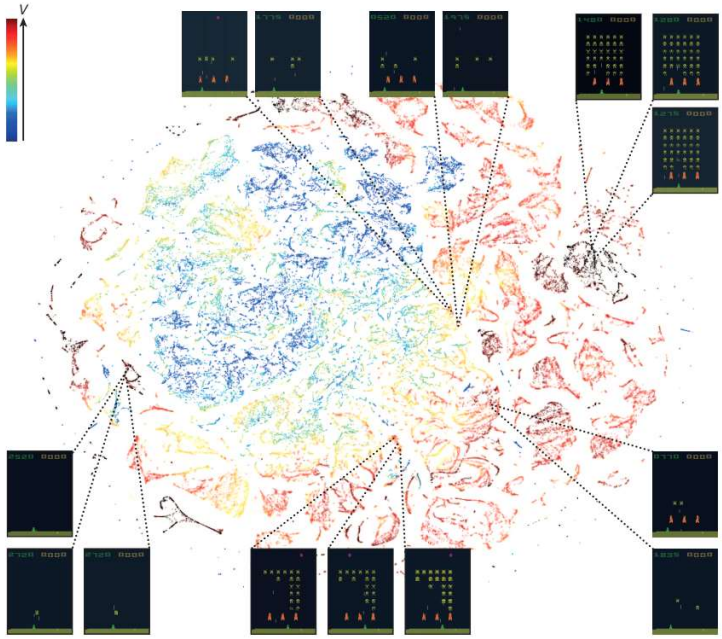
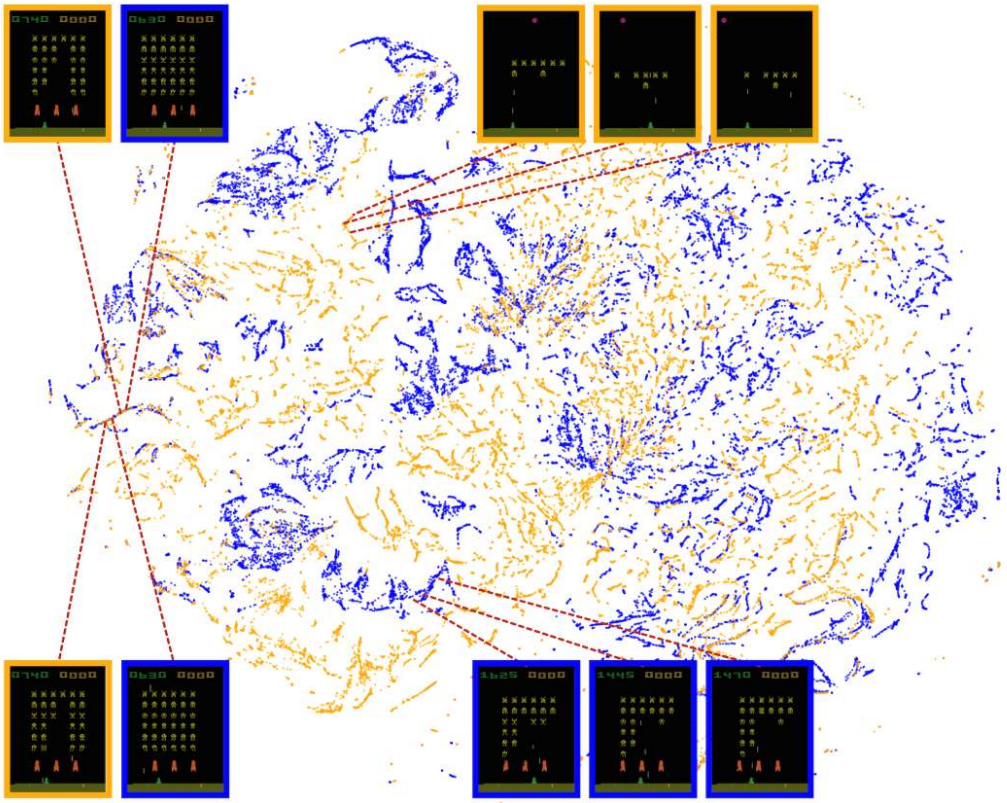


Figure 4 | Two-dimensional t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states experienced while playing Space Invaders. The plot was generated by letting the DQN agent play for 2 h of real game time and running the t-SNE algorithm[25] on the last hidden layer representations assigned by DQN to each experienced game state. The points are coloured according to the state values (*V*, maximum expected reward of a state) predicted by DQN for the corresponding game states (ranging from dark red (highest *V*) to dark blue (lowest *V*)). The screenshots corresponding to a selected number of points are shown. The DQN agent predicts high state values for both full (top right screenshots) and nearly complete screens (bottom left screenshots) because it has learned that completing a screen leads to a new screen full of enemy ships. Partially completed screens (bottom screenshots) are assigned lower state values because less immediate reward is available. The screens shown on the bottom right and top left and middle are less perceptually similar than the other examples but are still mapped to nearby representations and similar values because the orange bunkers do not carry great significance near the end of a level. With permission from Square Enix Limited.

그림 6

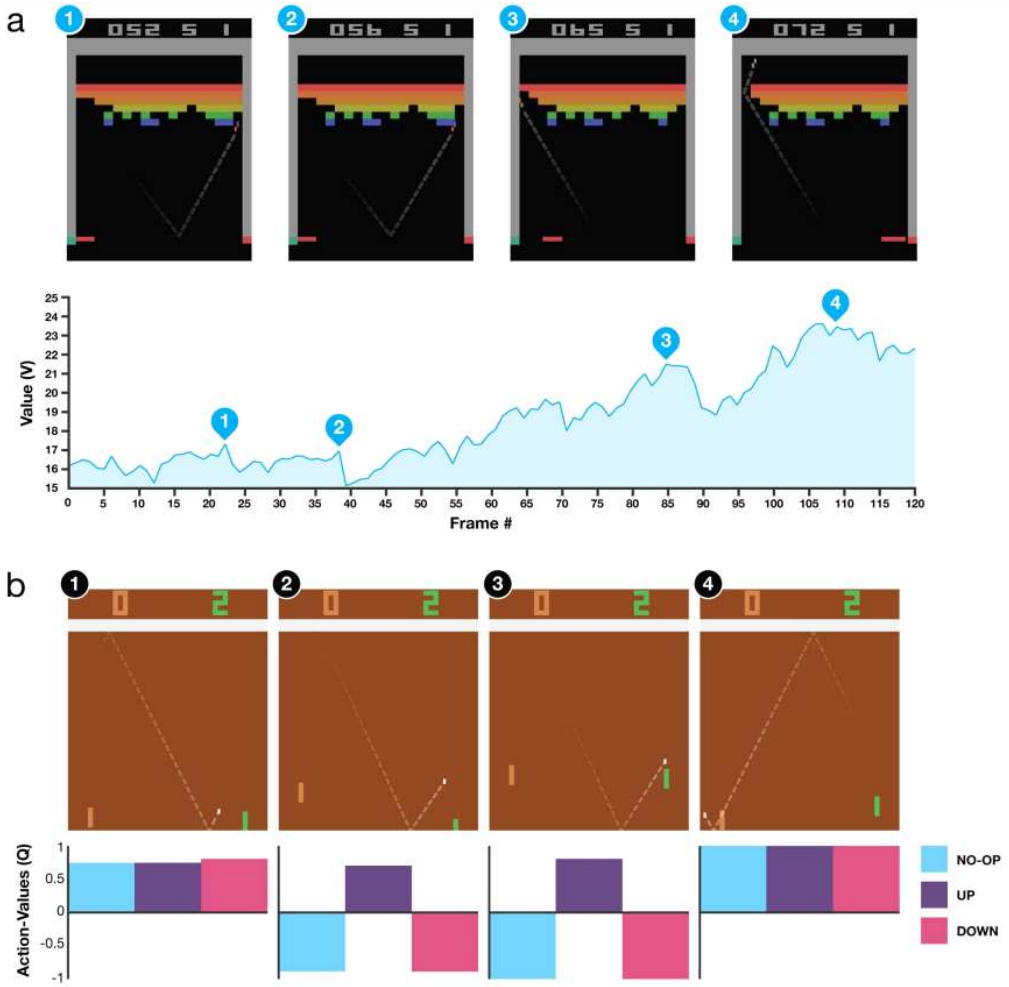
결과는 예상한대로 t-SNE에서 가깝게 위치한 점은 비슷한 표지값을 나타내고 있었다. 하지만 흥미롭게도 t-SNE알고리즘이 DQN의 변수중 기대보상값은 비슷하지만 시각적(?) (perceptually)으로는 다른 표지값을 가까이 배치하는 결과 역시 관측됐으며, 이는 네트워크가 복잡한 입력 데이터에서 적응적 행동을 나타내는 표지값을 학습할 수 있다는 이론을 뒷받침

한다. 또한, 우리는 입력값을 사람 또는 타 agent가 진행하는 게임 화면을 넣어주고, 최종 레이어의 표지값을 기록한 후, t-SNE알고리즘으로 시각화함으로 해서(그림 7) DQN이 학습한 표지값들이 다른 행동정책에 의해 생성된 데이터도 이해할 수 있다는 사실을 확인했다. (그림 8에서는 어떻게 DQN이 학습한 표지값이 상태와 행동값을 예측할 수 있는지 설명한다)



Extended Data Figure 1 | Two-dimensional t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states experienced during a combination of human and agent play in Space Invaders. The plot was generated by running the t-SNE algorithm[25] on the last hidden layer representation assigned by DQN to game states experienced during a combination of human (30 min) and agent (2 h) play. The fact that there is similar structure in the two-dimensional embeddings corresponding to the DQN representation of states experienced during human play (orange points) and DQN play (blue points) suggests that the representations learned by DQN do indeed generalize to data generated from policies other than its own. The presence in the t-SNE embedding of overlapping clusters of points corresponding to the network representation of states experienced during human and agent play shows that the DQN agent also follows sequences of states similar to those found in human play. Screenshots corresponding to selected states are shown (human: orange border; DQN: blue border).

그림 7



Extended Data Figure 2 | Visualization of learned value functions on two games, Breakout and Pong. a, A visualization of the learned value function on the game Breakout. At time points 1 and 2, the state value is predicted to be ~17 and the agent is clearing the bricks at the lowest level. Each of the peaks in the value function curve corresponds to a reward obtained by clearing a brick. At time point 3, the agent is about to break through to the top level of bricks and the value increases to ~21 in anticipation of breaking out and clearing a large set of bricks. At point 4, the value is above 23 and the agent has broken through. After this point, the ball will bounce at the upper part of the bricks clearing many of them by itself. **b**, A visualization of the learned action-value function on the game Pong. At time point 1, the ball is moving towards the paddle controlled by the agent on the right side of the screen and the values of all actions are around 0.7, reflecting the expected value of this state based on previous experience. At time point 2, the agent starts moving the paddle towards the ball and the value of the 'up' action stays high while the value of the 'down' action falls to −0.9. This reflects the fact that pressing 'down' would lead to the agent losing the ball and incurring a reward of −1. At time point 3, the agent hits the ball by pressing 'up' and the expected reward keeps increasing until time point 4, when the ball reaches the left edge of the screen and the value of all actions reflects that the agent is about to receive a reward of 1. Note, the dashed line shows the past trajectory of the ball purely for illustrative purposes (that is, not shown during the game). With permission from Atari Interactive, Inc.

그림 8

또한 DQN이 월등한 성능을 보이는 게임들이 대부분 근본적으로 다른것들(횡 스크롤링 슈터(River Raid)부터 복싱게임, 3D 레이싱 게임(Enduro)까지)이라는것 또한 흥미로운 사실이지만, 특정 게임에서 DQN이 보여준 장기적 전략을 보게 된다면(Breakout에서는 벽을 따라 구멍을 파고, 그 구멍으로 공을 올려보내는 등의 최적 전략을 학습했다.) 당연해 보이기도 한다. 그러나 몬테즈마의 복수와 같은 상황에 따라 각기 다른 장기적 전략을 요구하는 게임의 경우는 여타 agent와 다름 없이 DQN에게도 어려운 목표로 남아있다.

이 연구에서 우리는 최소한의 사전 정보와 화면 픽셀 정보, 게임 점수와 사람이 조작하는 커맨드와 동일한 커맨드 목록만을 가지고 동일 네트워크 구조, 하이퍼파라메터를 사용하는 단일 DQN구조가 다양한 환경에 대한 행동 정책을 학습할 수 있다는 사실을 보였다. 선행하는

연구들과는 달리 우리의 접근 방법은 '보상'이 지속적으로 CNN의 표지값을 값 예측에 적합한 명확한 특징값에 근사시키는 '단대단' 강화학습법을 적용했으며, 이 방식은 원시 시각피질이 '보는'방법을 배울때의 보상신호가 원시 시각피질에서 나타나는 표지값의 특성에 영향을 줄 수 있다는 이론에 기반한 방식이다. 특기할만한 점은, deep neural network에 강화학습을 적용시키는건 최근에 '경험'한 변화를 기록하고, 표지하는 기능을 포함하는 replay algorythm에 의지한다는 점이다. 이 과정은 포유류의 해마에서 휴지기(멍-한 상태 등)에 일어나는 현상과 유사하다. 향후 연구에서는 experience replay의 내용을 명료한 사건들로 주로 구성하는 등, 해마에서 일어나는 현상과 유사한 형식을 사용해 강화 학습에서 'prioritized sweeping'으로 불리는 방법을 시험해 보는것이 중요할것이다. 맺자면, 우리의 연구는 생물에서 영감을 얻은 개념들을 적용한 최첨단 기계학습이 다양한 목표를 학습할 수 있는 agent를 생성하는데 있어 얼마나 효율적인지를 보였다.

==========

방법론

Preprocessing 8비트 색공간을 갖는 210x160짜리 아타리 2600 프레임을 직접 계산에 사용하기에는 메모리랑 컴퓨팅 자원이 과도히 소모될 수 있기에 기본적인 전처리 과정을 거쳐 입력값의 크기를 줄이고, 에뮬레이터 특유의 특징을 없애는데 중점을 뒀다. 첫번째로 거친 전처리는 단일 프레임을 인코딩하기 위해 각 화면 픽셀 색의 최댓값을 인코딩 대상 프레임과 그 전 프레임에서 추출하는 과정이다. 이는 아타리 2600이 단일 프레임에 출력할 수 있는 sprite 갯수의 한계로 인한것으로서, 어떤 sprite는 홀수 프레임에만, 어떤 sprite는 짝수 프레임에만 나타나는 현상을 보정해준다. 두번째로 거치는 전처리는 프레임에서 luminance라고도 불리는 Y채널을 추출하여 84*84로 리사이징을 거치는 작업이다. 그림 9에서 설명되는 알고리즘이 이 전처리 과정을 최근 m프레임에 적용하고 m=4를 사용해 Q-함수에 집어넣을 입력값을 생성한다. (m값은 임의의 값이어도 동작한다)

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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

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

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

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

  End For

End For

Code availability. 소스코드는 https://sites.google.com/a/deepmind.com/dqn 에서 비 상업적 용도로만 사용할 수 있다.

Model architecture 뉴럴 네트워크를 사용할때 Q를 파라메터화 시키는데는 몇가지 방법이 있다. 이는 Q가 history-action쌍을 그 쌍에 대한 Q의 스칼라값으로 매핑하기 때문이다. 전에 진행됐던 몇몇 연구에서는 이 history-action쌍을 뉴럴 네트워크의 입력값으로 사용했는데, 이런 접근 방법은 한 쌍을 처리하기 위해서는 각각의 foward-pass를 설계해야 하고, 따라서 처리해야 하는 행동의 수에 따라 컴퓨팅 자원의 요구량이 선형적으로 증가한다는 큰 단점이 있게 된다. 우리는 대신 가능한 각 액션별로 개별적인 출력 유닛을 가지는 구조를 채택하는 한편, 뉴럴 네트워크에는 상태 지표값만을 입력해줬고, 출력값은 입력값에서 예상된 Q값에 상응하는 행동값으로 설정했다. 이러한 구조의 정점은 한번만 네트워크를 통해 Foward-pass를 돌려도 주어진 상황에서 필요한 모든 행동값을 계산할 수 있다는 점이다. 그림 1에서 개념적으로 설명된 구조는 전처리 과정에 의해 생성된 84*84*4의 크기를 가지는 이미지가 뉴럴네트워크의 입력값으로 들어가고, 첫번째 hidden layer는 32개의 필터를 8*8 격자에 4 stride로 convolve를 수행한 뒤, rectifier nonlinearity를 적용한다. 두번째 hidden layer에서는 64개의 필터를 4*4 격자에 2 stride로 convolce를 수행하고 첫번째와 동일하게 rectifier nonlinearity를 적용한다. 세번째 hidden layer에서는 64개의 필터를 3*3격자로 1 stride로 convolve한 뒤 recifier를 적용하고, 마지막 hidden layer는 fully-connected레이어이며, 512개의 rectifier 유닛들을 갖는다. 최종 출력 레이어 역시 fully-connected레이어고, 학습시키는 게임에 따라 4~18

개로 변하는 가능한 행동당 하나씩의 출력이 배정된 형식을 취했다.

Traning details 우리는 다른 방법들과의 결과를 비교해볼 수 있는 49개의 아타리 2600게임을 사용해 실험을 진행했고, 우리의 방법이 충분히 최소한의 사전지식만 가지고도 여러가지 게임을 학습할 수 있음을 보이기 위해 각기 다른 게임들에 맞춰 동일한 네트워크 구조, 학습 알고리즘, 하이퍼파라메터를 사용하는 네트워크를 각각 훈련시켰다. (그림 10 참조)

Extended Data Table 1 | List of hyperparameters and their values

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.

The values of all the hyperparameters were selected by performing an informal search on the games Pong, Breakout, Seaquest, Space Invaders and Beam Rider. We did not perform a systematic grid search owing to the high computational cost, although it is conceivable that even better results could be obtained by systematically tuning the hyperparameter values.

그림 10

게임을 사용해서 네트워크를 훈련시키는 과정에서 우리는 다른것들은 모두 그대로 놔두는 대신 게임의 보상 구조에 약간의 변경을 가했다. 이 변경점은 모든 보상값을 1로 변경하고 모든 페널티값을 -1로 변경해서 점수 변화가 없는 상황에 대한 보상은 0으로 통일한것이다. 이렇게 보상값을 정리하는것은 오류가 나타날 가능성을 줄이며 동일한 learning rate를 여러 게임에 적용시키는걸 용이하게 해주지만, 그와 동시에 보상의 크기를 인식할 수 없게 하기에 성능에 악영향을 미칠수도 있다. 게임들중 목숨 카운터가 있는 게임의 경우 에뮬레이터는 남은 목숨 수 역시 제공하는데, 이는 훈련을 진행할때 에피소드의 끝을 인식시키는데 사용됐다.

실험중에는 RMSProp (http://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf 참조) 알고리즘이 솔버로서 사용됐고, minibatch는 32로 설정됐다. 훈련중 행동정책은 처음 10000프레임까지는 1.0부터 0.1까지 감소하고, 그 뒤로는 0.1로 유지되는 ε(learning rate로 추정)에 ε-greedy하게 설정됐다. 우리는 총 5000000프레임을 약 38일에 거쳐 훈련을 진행하며 100000프레임에 달하는 replay memory를 사용했다.

아타리 2600을 사용해 훈련을 진행하며 우리는 약간의 전 연구에서 진행됐던 접근법도 사용

했는데, 에뮬레이터를 한 스텝 진행시키는건 agent가 행동을 결정하게 하는것보다 훨씬 적은 컴퓨팅 자원이 요구된다는걸 이용한 '프레임 건너뛰기'수법이다. 조금 더 자세히 설명하자면 agent가 매 프레임마다 행동을 결정짓게 하는게 아니라 k번째 프레임마다 결정짓게 하고, 동일한 결정은 건너뛴 프레임들에 동일하게 적용하는 방식이다. 이 방법은 agent가 대략 k배 더 많은 게임을 실행시간의 큰 증가 없이 경험할 수 있게 해줬다.

하이퍼파라메터들의 값과 최적화 파라메터들의 값들은 Pong, Breakout, Seaquest, Space Invaders, Beam Rider등의 게임에서 컴퓨팅 자원이 너무 많이 요구되는 systematic grid search가 아니라 informal search를 시행해서 정했고, 이 파라메터들은 여타 게임들에 대한 훈련을 진행할때도 동일하게 적용되었다. (파라메터의 값과 설명은 그림 10 참조) 실험을 진행할때의 환경은 입력 데이터가 이미제 이미지 데이터라는것과 게임별 점수, 행동의 결과를 알려주지 않은 가능한 행동 목록, 목숨 수만을 포함하는 최소한의 사전지식이 주어진채로 진행됐다.

Evaluation procedure 훈련된 agent들은 각 게임을 서로 다른 초기조건(각 에피소드의 시작에서 얼만큼 아무 조작도 안하는지)과 ε=0.05의 ε-greedy 행동정책을 가진채로 최대 5분까지 30회 플레이하게 함으로서 평가되었다. 이 서로 다른 초기조건은 평과 과정에서 overfitting이 일어나는것을 억제하기 위해 사용됐다. 비정상적으로 높은 기준선을 막기 위해 인간 플레이어가 가장 빨리 한 버튼을 연타할 수 있는 수준인 10Hz로 작동하도록 설정된 random agent가 만들어낸 점수가 기준선으로 설정되었고, 60Hz짜리도 실험되었지만 DQN이 인간 플레이어를 월등히 능가하는 성능을 보이는 6개 게임에서만 5%이상의 성능 향상을 보이는 등, 차이는 미미했다.

인간 테스터는 동일한 에뮬레이터를 사용해 통제된 환경 내에서 평가를 진행했다. 일시정지는 허용되지 않았으며, 저장과 불러오기 역시 불허됐다. 에뮬레이터는 60Hz로 구동됐으며, 소리 출력은 agent와의 형평성을 위해 비활성화 됐다. 인간 테스터의 점수는 최대 5분정도 지속되는 약 20 에피소드에 걸친 보상값의 평균으로 상정했으며, 각 게임을 플레이 하기 전에는 약 두시간의 연습시간이 주어졌다.

Algorythm 우리는 agent가 아타리 에뮬레이터 내의 '환경'과 상호작용하는 과정을 크게 행동, 관측, 보상의 시퀀스로 봤다. 각 time-step마다 agent는 가능한 행동 목록 A={1, ... ,K} 에서 행동 a_t를 선택하고, 그 행동은 에뮬레이터에 입력되어 내부상태와 게임 점수를 변화시킨다. 일반적으로는 환경은 무작위적일 수 있다.(?) 이때 에뮬레이터의 내부상태는 agent에게 공개되지 않고 대신 agent는 현재 화면의 픽셀값들을 벡터화한 $x_t \in \mathbb{R}^d$ 를 따르는 이미지를 관측하게 되고, 추가적으로는 게임 점수 변화를 뜻하는 r_t를 입력받는다.

여기서 상기하고 넘어가야 할 점은 일반적으로는 게임 점수가 그 전에 나왔던 화면들과 행동들에 의해 결정되기 때문에 행동에 대한 피드백이 몇천 time-step이 지나고 나서야 들어갈 수 있다는것이다.

agent가 현재 화면만을 관측하기 때문에 전체적인 목표는 일부만 관측할 수 있으며, 에뮬레

이터의 내부 상태는 피상적으로만 근사될 수 있다.(이는 오직 현재 화면만으로는 현 상황을 완전히 이해할 수 없다는것을 의미한다) 고로 행동과 관측 시퀀스들($s_t=x_1,a_1,x_2,\ldots,a_{t-1},x_t$)가 알고리즘의 입력값이 되고, 알고리즘은 이 입력값을 바탕으로 게임에 대한 전략을 학습해나간다. 모든 시퀀스들은 유한한 time-step내에 끝나는것으로 상정되어 있으며 이러한 formalism은 각 시퀀스가 한개의 상태를 나타내는 무겁지만 유한한 Markov decision process(MDP)에 힘을 실어줬다. 그에 따라 우리는 간단히 s_t를 시간 t에 대한 표지값으로 사용하는 평범한 MDP용 학습 방법을 사용할 수 있었다.

agent의 목표는 기대 보상이 가장 커지는 방향으로 에뮬레이터와 상호작용하는것이다. 우리는 여기서 기대 보상은 time-step에 따라 γ만큼 감소한다는 가정을 세웠고,(γ는 0.99로 고정됐다.) 미래의 보상값을 $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ 로 설정했다. 여기서 T는 게임이 끝나는 시점의 time-step이다. 또한 우리는 최적 행동값 함수 $Q^*(s,a)$를 어느 행동정책을 따르더라도 얻을 수 있는 최대 기대 보상값으로 설계 했고, 어느정도의 시퀀스 s가 진행되고 행동 a를 취한 뒤에는 $Q^*(s,a) = \max_\pi \mathbb{E}[R_t|s_t=s,a_t=a,\pi]$ 가 된다. 여기서 π는 행동에 대해 행동정책을 매핑하는것을 뜻한다.

The optimal action-value function obeys an important identity known as the

Bellman equation. This is based on the following intuition: if the optimal value $Q^*(s',a')$ of the sequence s' at the next time-step was known for all possible actions

a', then the optimal strategy is to select the action a' maximizing the expected value

of $r+\gamma Q^*(s',a')$

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

대부분의 강화학습 알고리즘의 기저에 깔려있는 아이디어는 Bellman 공식을 $Q_{i+1}(s,a)=\mathbb{E}_{s'}[r+\gamma \max_{a'} Q_i(s',a')|s,a]$를 따르도록 iterative하게 활용해 행동값 함수를 추정해 내는것이다. 이러한 value iteration 알고리즘들은 $Q_i \to Q^* \text{ as } i\to\infty$ 에 따라 최적 행동값 함수에 수렴해 가게 된다. 실제로는 이러한 기본적인 접근법은 비효율적인데, 이는 행동값 함수가 일반화되지 않고 각각의 시퀀스에만 최적화 되기 때문이다. 대신, 흔히 사용되는 방법으로는 $Q(s,a;\theta)\approx Q^*(s,a)$ 를 따르는 함수 근사를 이용해 행동값 함수를 추정하는 방법이 있다. 흔히 강화학습 커뮤니티에서는 선형 근사함수를 이용하지만 때로는 뉴럴 네트워크와 같은 비선형 함수근사를 이용하기도 한다. 우리는 θ의 '무게'를 갖는 뉴

럴 네트워크를 Q-네트워크로 불렀다. Q-네트워크는 θ_i 파라메터를 iteration i 에서 조정함으로서 bellman 공식에 존재하는 최적 표적값 $r+\gamma \max_{a'} Q^*(s',a')$가 근사된 표적값인 $y=r+\gamma \max_{a'} Q(s',a';\theta_i^-)$로 대체되는 mean-squared 오류를 몇 itreation 전의 파라메터인 θ_i^-를 사용해 피할 수 있다. 이는 iteration i 마다 변하는 loss function의 시퀀스 $L_i(\theta_i)$

$$L_i(\theta_i)=\mathbb{E}_{s,a,r}\left[(\mathbb{E}_{s'}[y|s,a]-Q(s,a;\theta_i))^2\right]$$
$$=\mathbb{E}_{s,a,r,s'}\left[(y-Q(s,a;\theta_i))^2\right]+\mathbb{E}_{s,a,r}[\mathbb{V}_{s'}[y]].$$

를 생성한다. 여기서 표적은 훈련이 시작되기 전부터 고정되는 supervised 학습과는 달리 네트워크의 weight 에 따라 변한다는것을 유념해둬야 할 필요가 있다. 각 최적화단계에서 우리를 전 iteration의 파라메터 θ_i^-를 고정시킨 상태에서 i번째 loss 함수를 최적화 하고, 이는 잘 정리된 최적화 문제만을 남긴다. 마지막 식은 표적이 θ_i에 대한 표적의 편차이고, 따라서 무시된다. 이때 weight에 대해 미분하면

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

이 식이 나오고, 대부분의 경우 이 전체 식을 모두 계산하기 보다는 stocastic gradient descent를 사용해 근사한다. 이때 Q-학습 알고리즘은 이런 구조에서 각 스텝마다 weight를 업데이트를 하며 단일 샘플 expectation을 사용하고, $\theta_i^-=\theta_{i-1}$로 놓을때 나타나게 된다.

유념해둬야 할것은 이 알고리즘은 모델이 필요치 않다는 점이다. 달리 적자면 이 알고리즘은 에뮬레이터에서 나온 샘플을 달리 보상값과 transition dynamics P(r,s'|s,a)를 예측하지 않고 직접 강화학습에 사용한다. 또한 이 알고리즘은 상태공간을 적당히 탐색할 수 있게 해주는 행동분산에 따라 greedy policy $\text{argmax}_{a'}Q(s,a';\theta)$를 학습하게 되는데, 이때 대부분의 경우(1-ε)에는 ε-greedy정책에 의해 행동이 결정되지만 그 외의 경우(ε)에는 무작위적인 행동을 나타내게 되므로 행동정책을 온전히 따르지도 않는다.