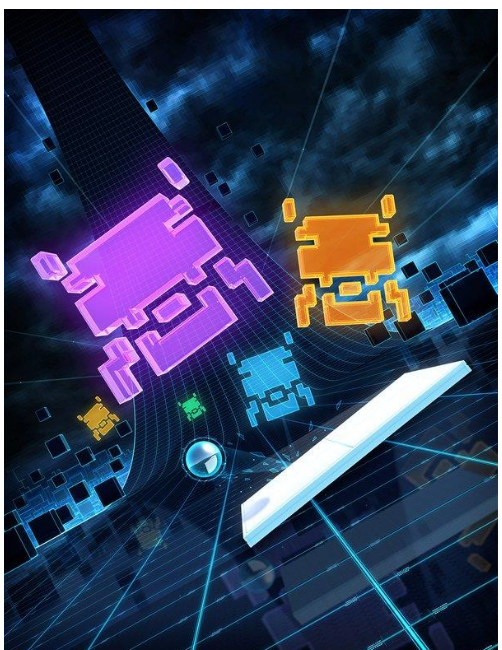
Human-Level Control through Deep Reinforcement Learning



Yu-Chun Chien, Chen-Yu Yen

Outline

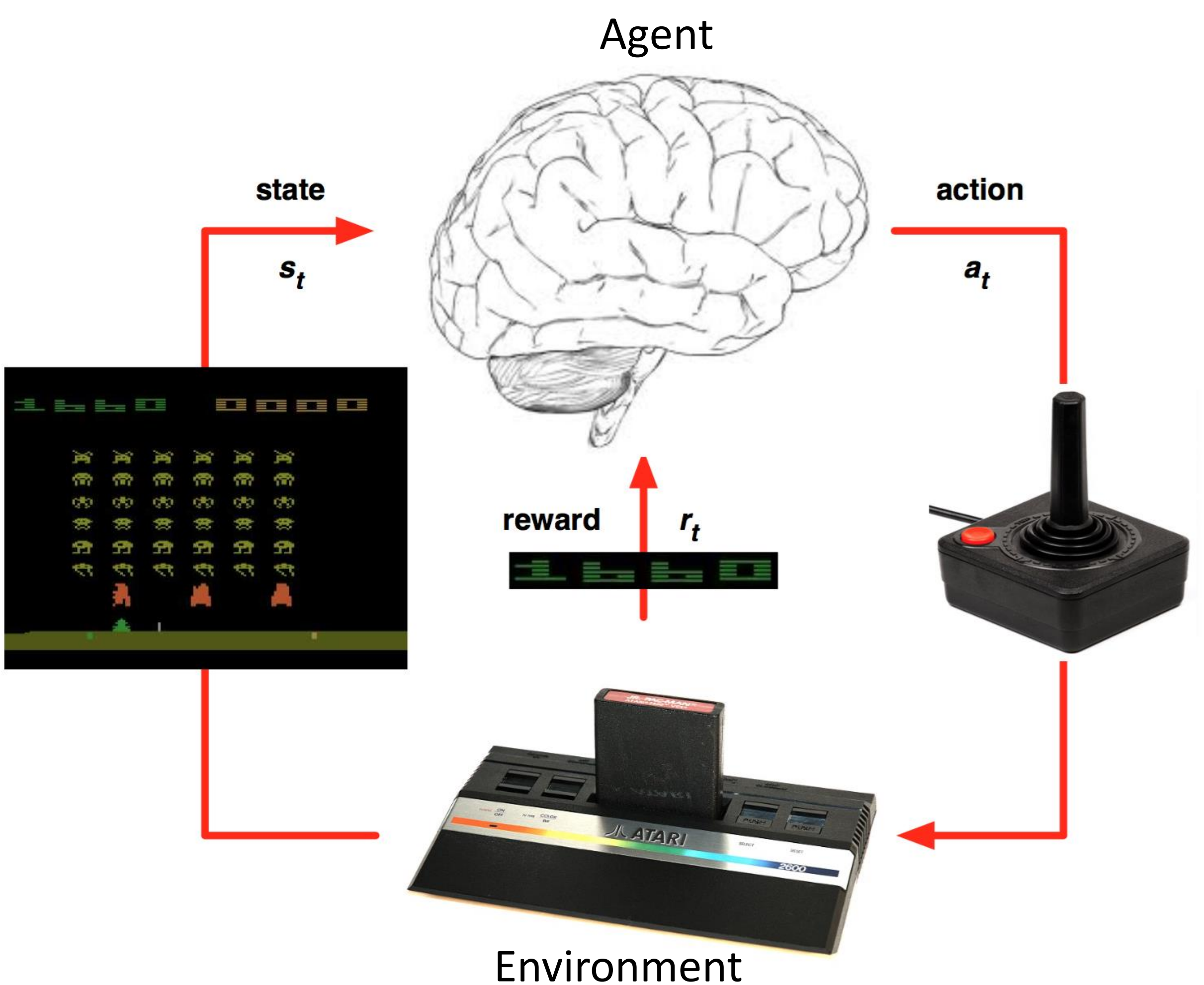
- Background
- Methods
- Evaluation
- Demo

Motivation

Space Invaders

DQN controls the green laser cannon to clear columns of space invaders descending from the sky and also destroys two pink motherships at the top of the screen

What is reinforcement learning (RL)



Policies and Value Functions

- Policy π is a behavior function selecting actions given states

$$\boldsymbol{a = \pi(s)}$$

- Value Function $Q^{\pi}(s, a)$ is expected total reward from state $\boldsymbol{s}$ and action $\boldsymbol{a}$ under policy $\boldsymbol{\pi}$

$$\boldsymbol{Q^{\pi}(s, a)} = \mathbb{E}[\boldsymbol{r_t} + \boldsymbol{\gamma r_{t+1}} + \boldsymbol{\gamma^2 \, r_{t+2}} + \cdots |\boldsymbol{s, a}]$$

Reinforcement Learning Approach

- Policy-based RL
  - Search for optimal policy $\boldsymbol{\pi}^*$
  - This is the policy achieving maximum future reward
- Value-based RL
  - Estimate the optimal value function $\boldsymbol{Q}^*$
  - This is the maximum value achievable under any policy

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

Bellman Equation

- Bellman expectation equation unrolls value function Q^{π}

$$Q^{\pi}(s,a) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s,a]$$
$$= \mathbb{E}_{s',a'}[r + \gamma Q^{\pi}(s',a') | s,a]$$

- Bellman optimal equation unrolls optimal value function Q^*

$$Q^*(s,a) = \max \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s,\ a_t = a,\ \pi\right]$$
$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

Solving Bellman optimal equation

- Represent value function by Q-network with weights θ

$$Q(s, a, \theta) \approx Q^{\pi}(s, a)$$

- Define objective function by mean-squared error in Q-values

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

RL + DL = Deep Q-Network (DQN)

- We seek a single agent which can solve any human-level task
  - RL defines the objective (Q-value function)
  - DL gives the mechanism
- Use deep network to represent value function

Stability Issues of DQN

Naïve Q-learning oscillates or diverges with neural nets

- Data is sequential and successive samples are correlated
- Policy changes rapidly with slight changes to Q-values
  - Policy may oscillate
  - Distribution of data can swing from one extreme to another
- Scale of rewards and Q-values is unknown
  - Gradients can be unstable when back-propagated

Stable Solutions for DQN

DQN provides a stable solution to deep value-based RL

1. Experience replay
2. Freeze target Q-network
3. Clip rewards to sensible range

Stable Solution 1: Experience Replay

To remove correlations, build dataset from agent's experience

- Take action a_t
- Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory ***D***
- Sample random mini-batch of transitions $(s, a\ , r, s')$ from replay memory ***D***
- Optimize MSE between Q-network and Q-learning targets

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

Stable Solution 2: Fixed Target Q-Network

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning target w.r.t old, fixed parameters

$$r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$$

- Optimize MSE between Q-learning targets and Q-network

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

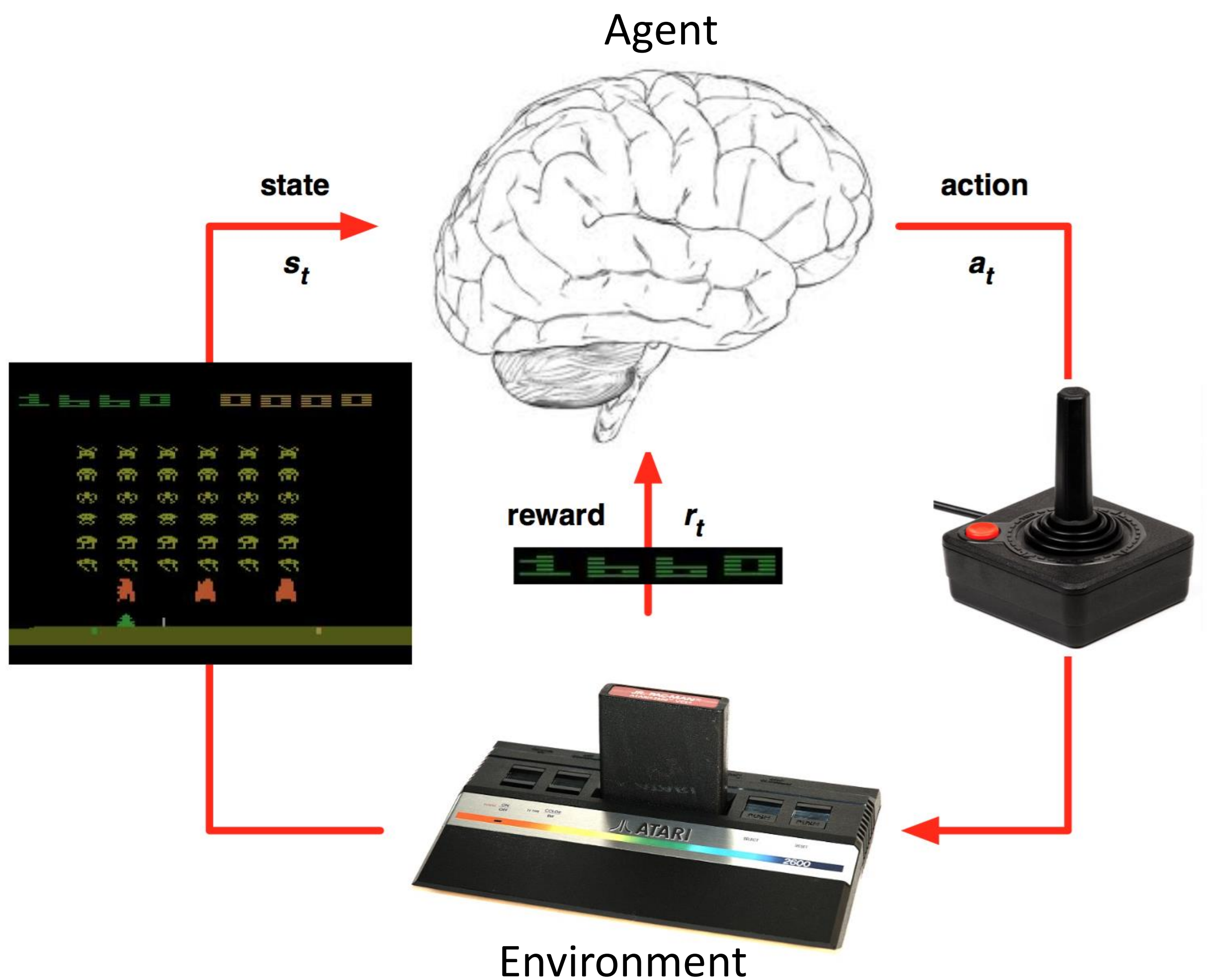
- Periodically update fixed parameters

Stable Solution 3: Reward/Value Range

To avoid oscillations, control the reward / value range

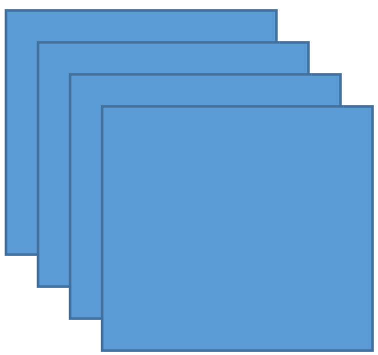
- DQN clips the rewards to [-1, +1]
    - Prevents too large Q-values
    - Ensures gradients are well-conditioned

DQN in Atari



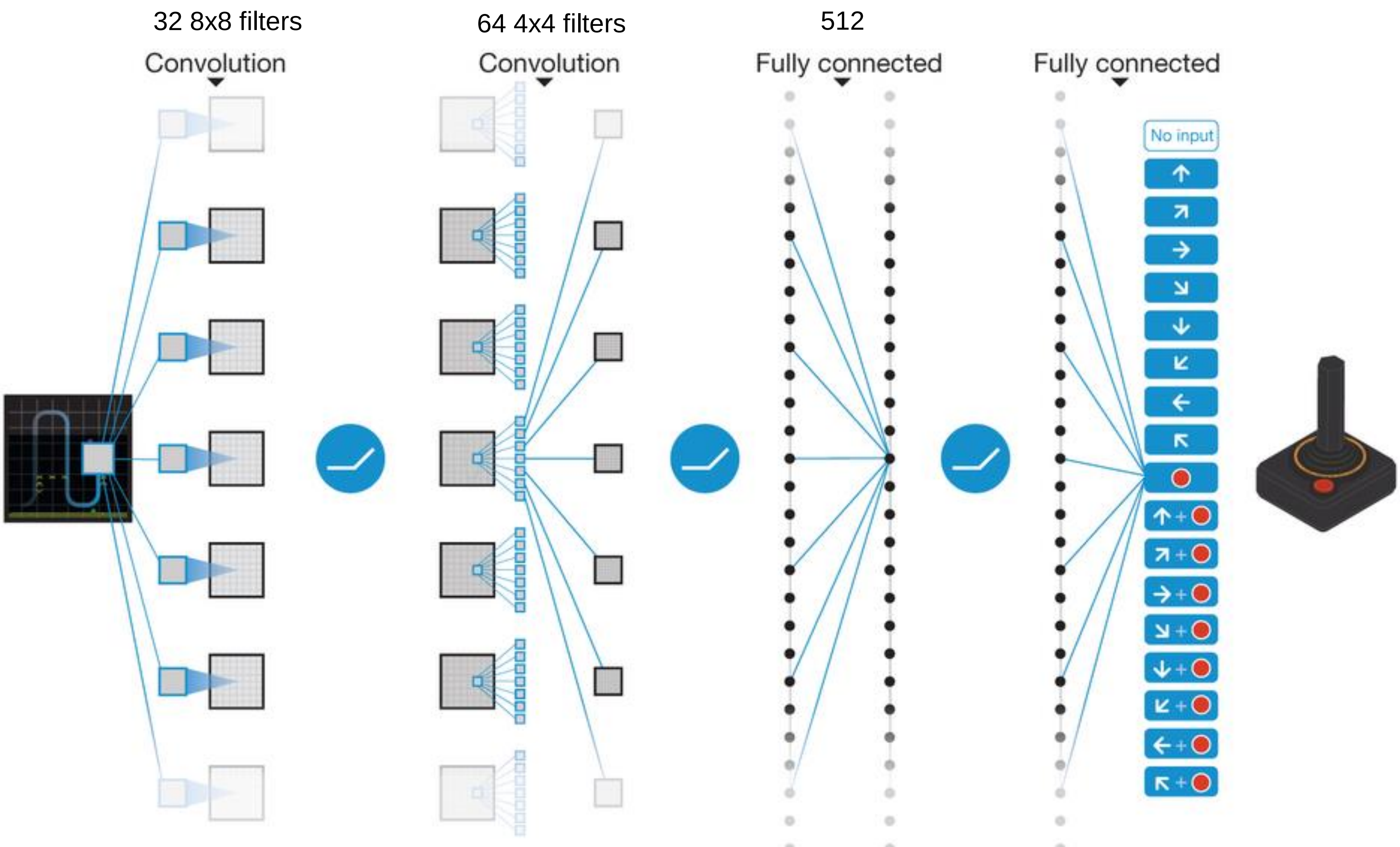
Preprocessing

- Raw images: 210x160 pixel images with 128-color palette
- Rescaled images: 84 x 84
- Input: 84 x 84 x 4 (4 most recent frames)



Model Architecture

- Input 84 x 84 x 4 stack frames
- Output 18 joystick positions



Training details

- 49 Atari 2600 games
- Use RMSProp algorithms with minibatches 32
- Use 50 million frames (38 days)
- Replay memory contains 1 million recent frames
- Agent select actions on every 4^{th} frames

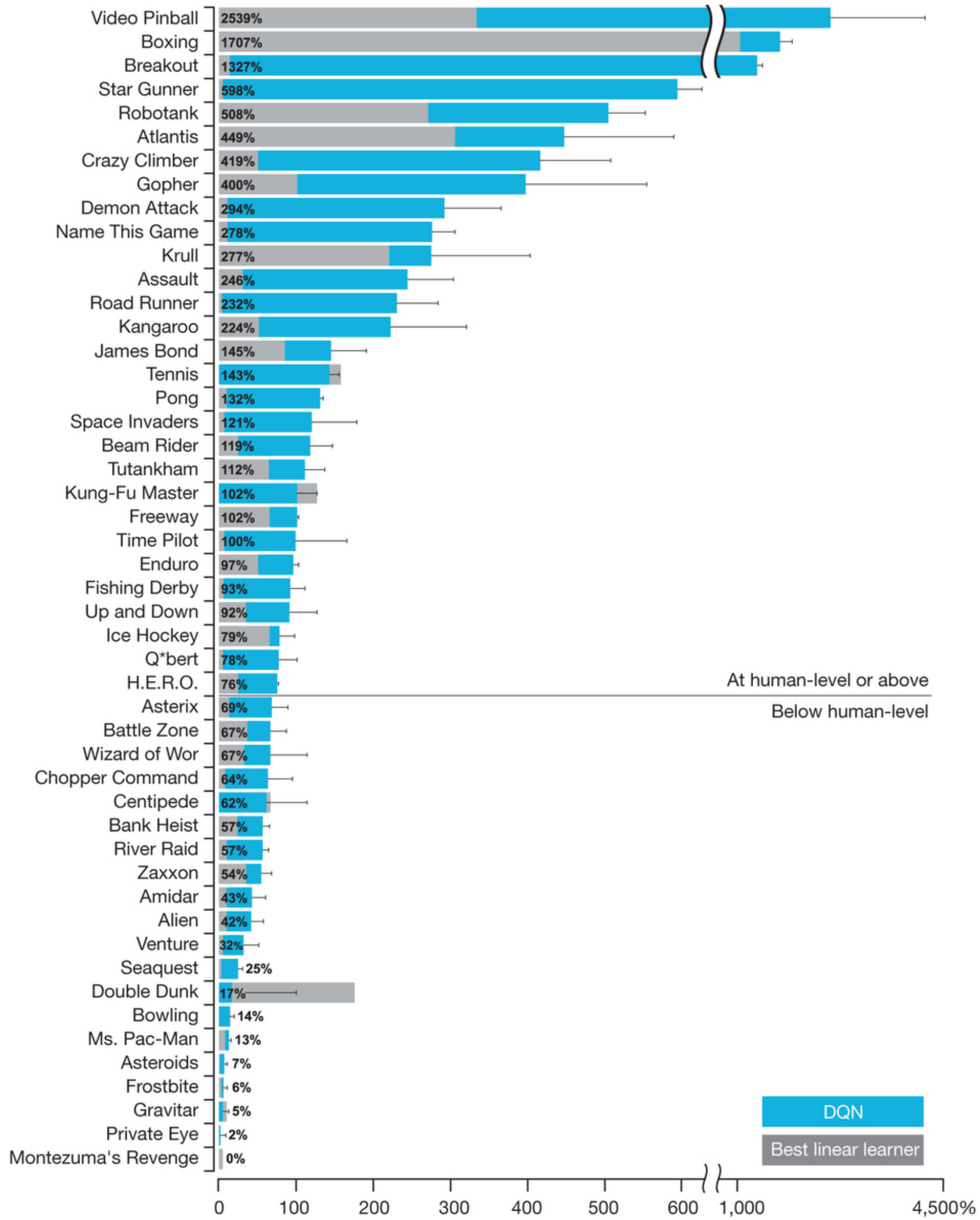
Evaluation

- Agent plays each games 30 times for 5 min with random initial conditions
- Human plays the games in the same scenarios
- Random agent play in the same scenarios to obtain base-line performance

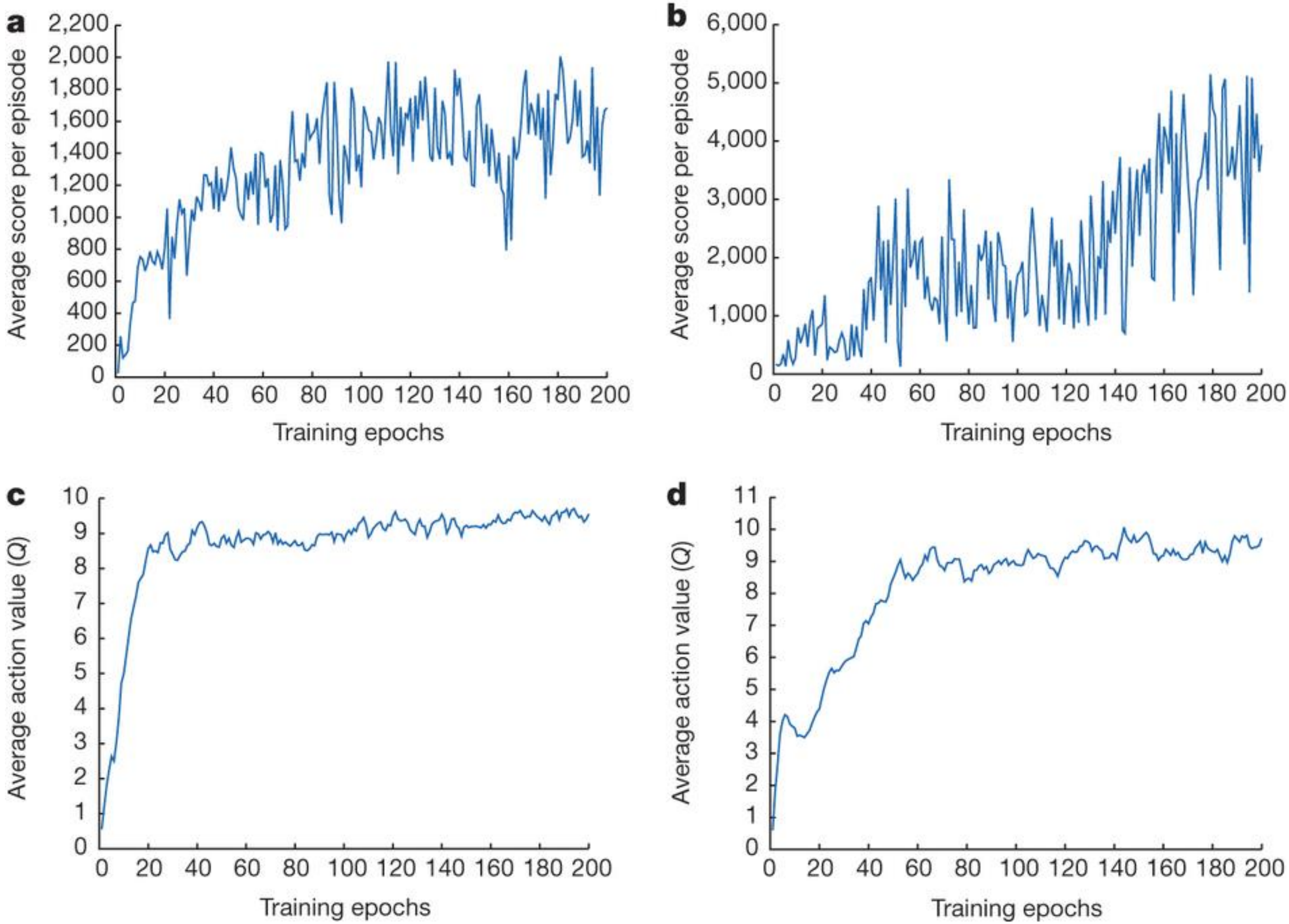
Algorithm

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

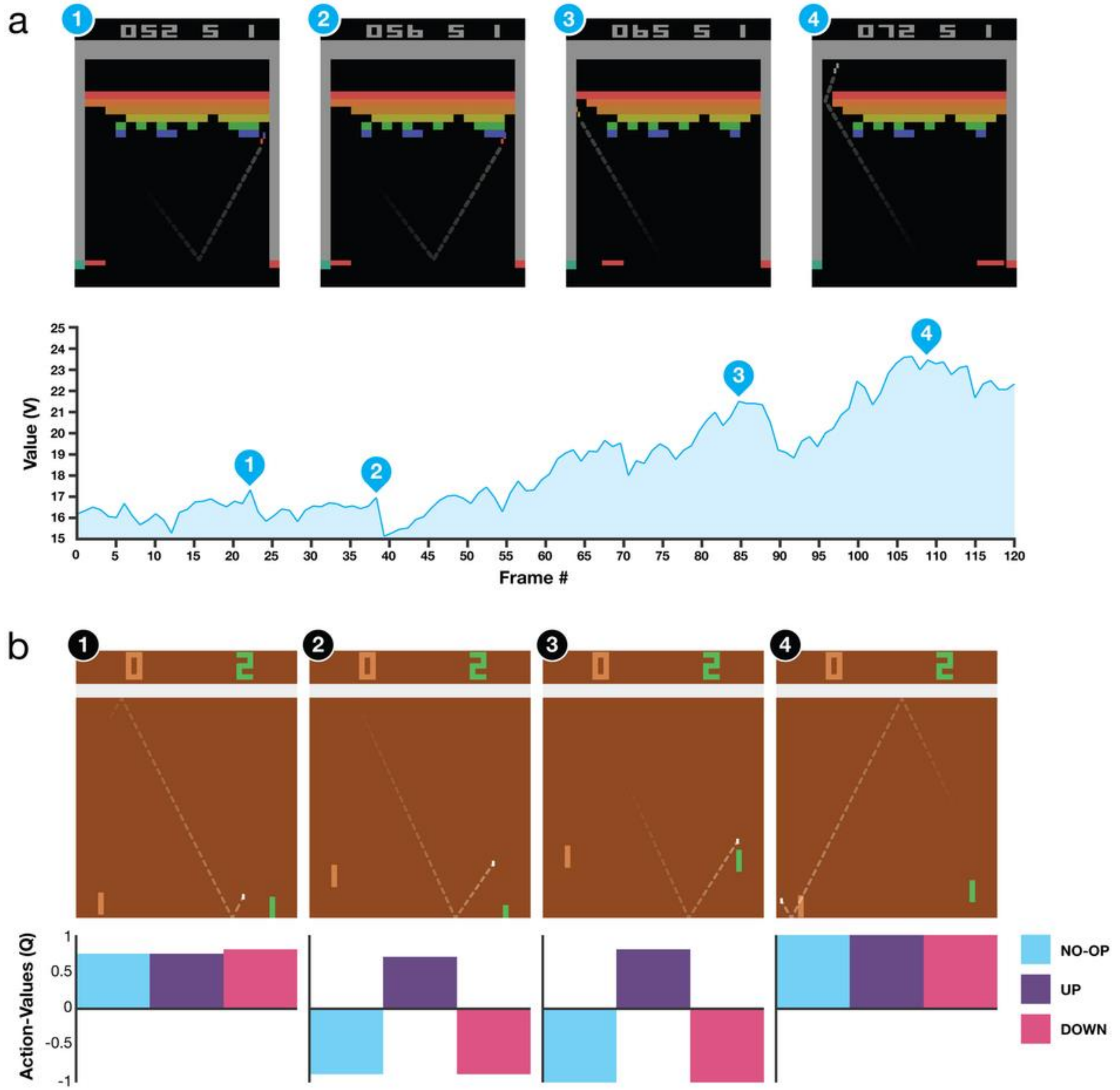
DQN Performance in Atari



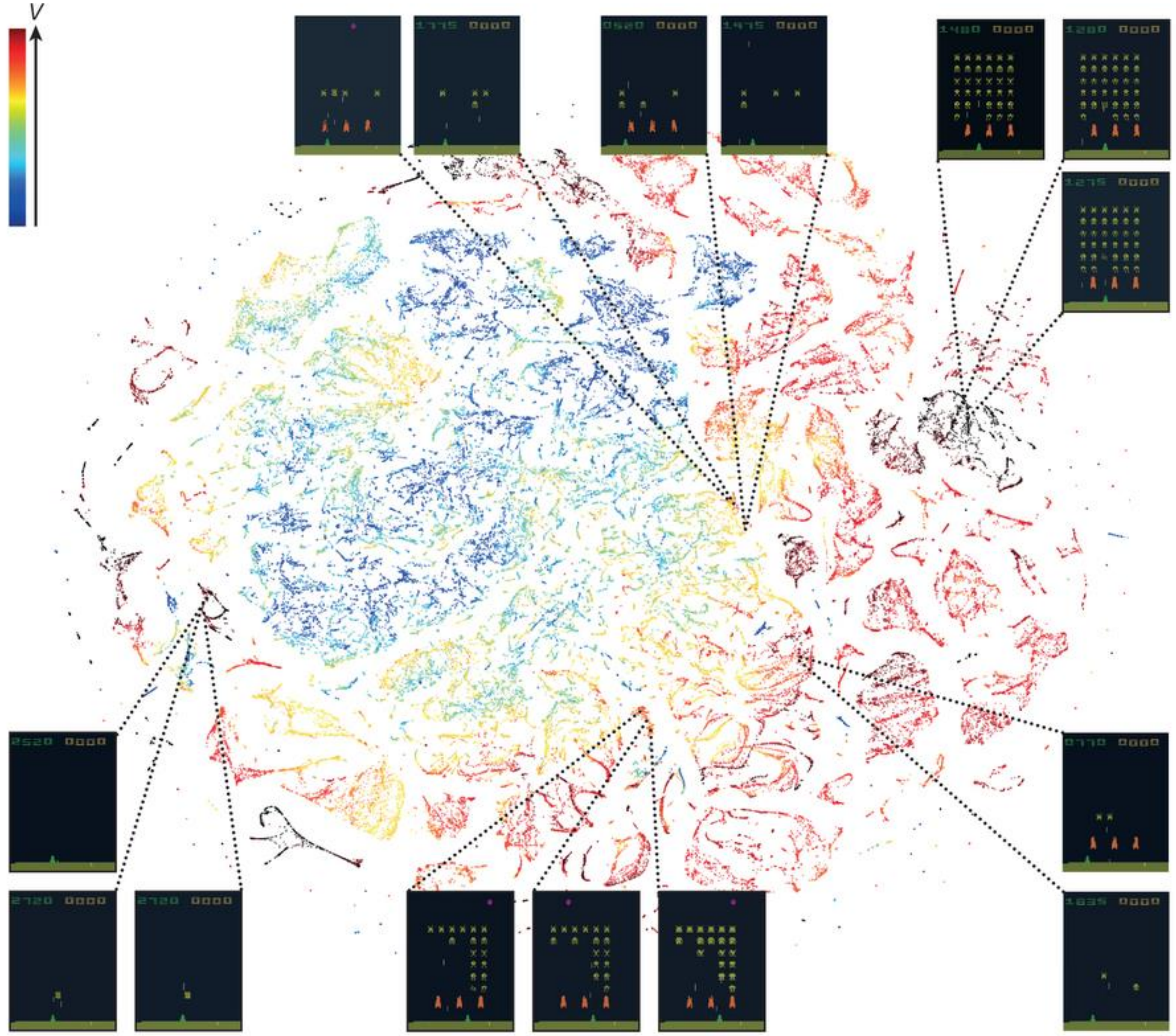
Evaluation


a
2,200
2,000
1,800
1,600
1,400
1,200
1,000
800
600
400
200
0
0 20 40 60 80 100 120 140 160 180 200
Training epochs
Average score per episode
b
6,000
5,000
4,000
3,000
2,000
1,000
0
0 20 40 60 80 100 120 140 160 180 200
Training epochs
Average score per episode
c
10
9
8
7
6
5
4
3
2
1
0
0 20 40 60 80 100 120 140 160 180 200
Training epochs
Average action value (Q)
d
11
10
9
8
7
6
5
4
3
2
1
0
0 20 40 60 80 100 120 140 160 180 200
Training epochs
Average action value (Q)

Visualization of value functions



Visualization of game states in the last hidden layer

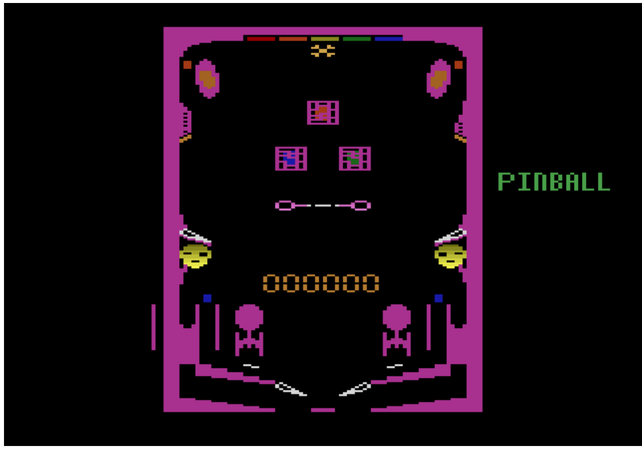


How effective are the stable solutions?

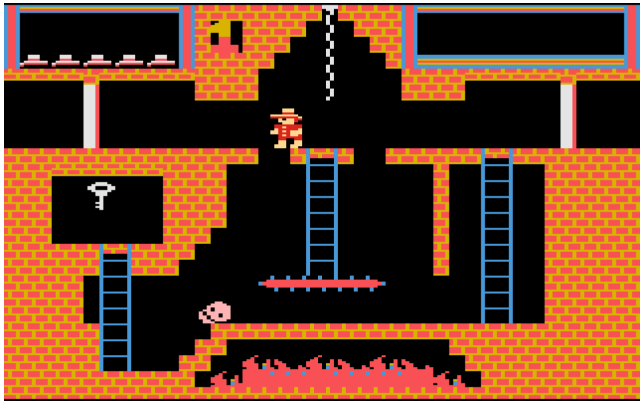
Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Strengths and weakness

- Strengths:
  - Quick-moving, short-horizon games
  - Pinball (2539%)



- Weakness:
  - Long-horizon games that do not converge
  - Walk-around games
  - Montezuma's revenge



Conclusion

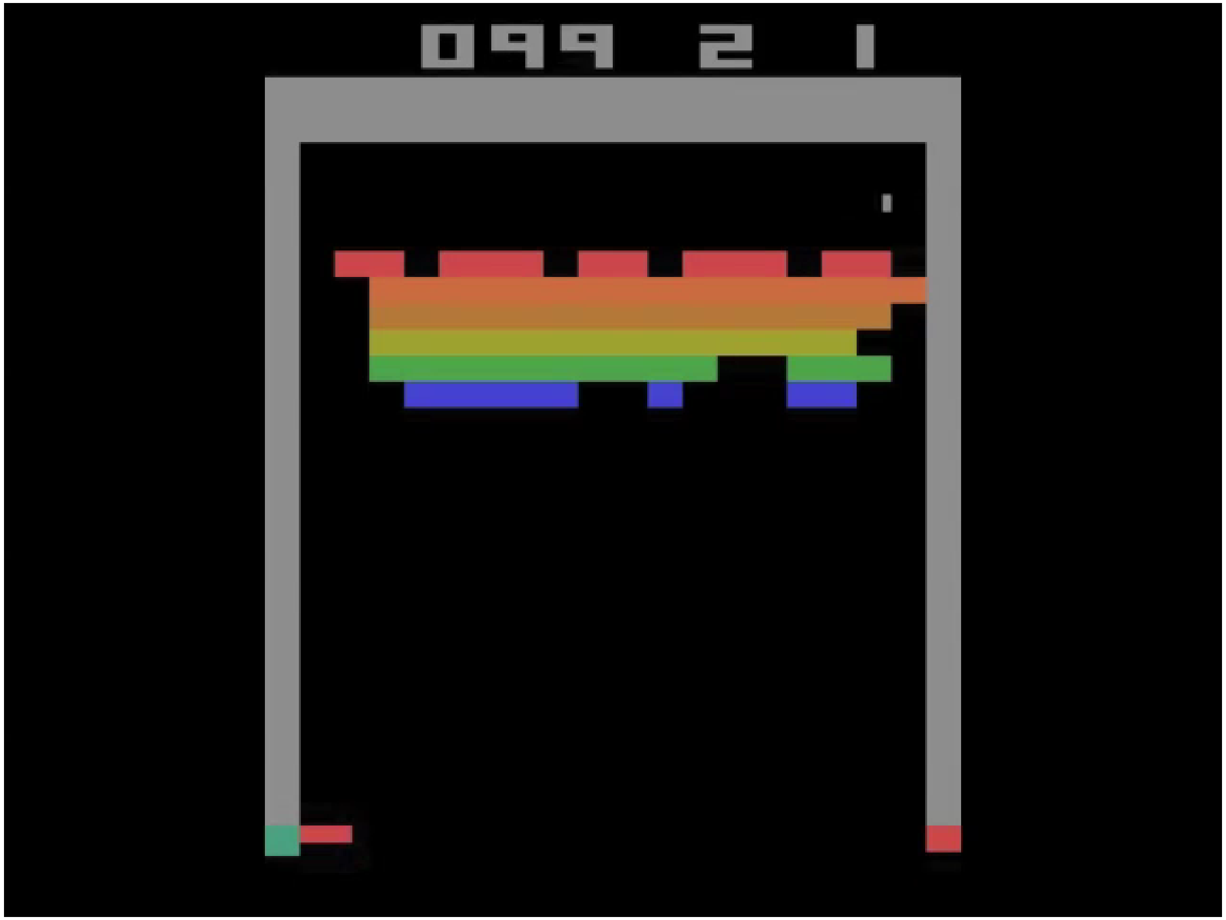
- Deep Q-network agent can learn successful policies directly from **high-dimensional** input using end-to-end reinforcement learning
- The algorithm achieve a level surpassing professional human games tester across **49 games**

Demo

http://www.nature.com/nature/journal/v518/n7540/fig_tab/nature14236_SV1.html

Demo

http://www.nature.com/nature/journal/v518/n7540/fig_tab/nature14236_SV2.html




Q
A

References

- Mnih, V. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015)
- ICLR 2015 Tutorial
- ICML 2016 Tutorial