Paper Discussion:

“Human-level Control through Deep Reinforcement Learning”

Mnih et al., Google DeepMind, Feb. 2015

Presented by Cesar A. Gomez

Machine Learning Journal Club

Department of Electrical and Computer Engineering

November 28, 2019

Content

- Introduction: Reinforcement Learning and the Q-learning algorithm
- The Deep Q-network method (DQN)
- Evaluation of DQN
- Open discussion

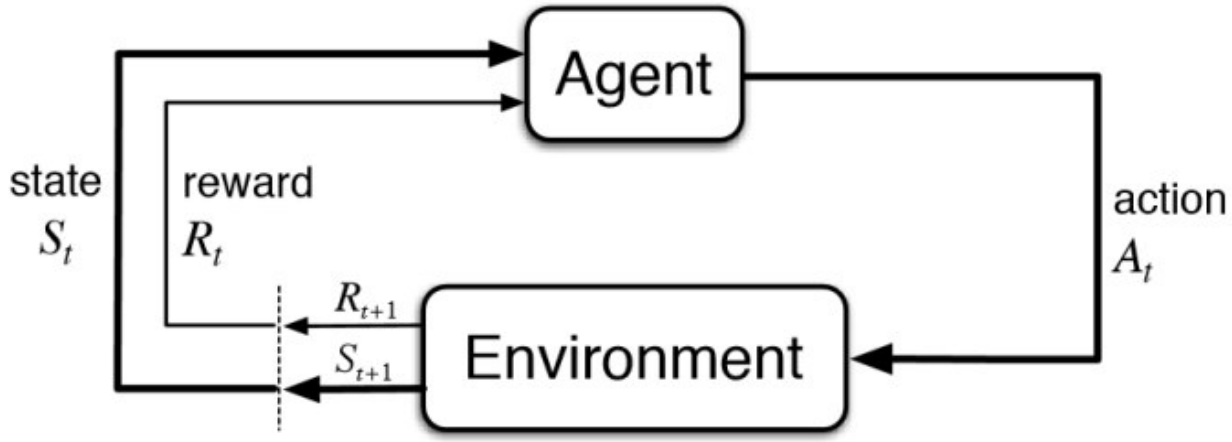
Introduction:

Reinforcement Learning and the Q-learning algorithm

Reinforcement Learning (RL)

Challenges

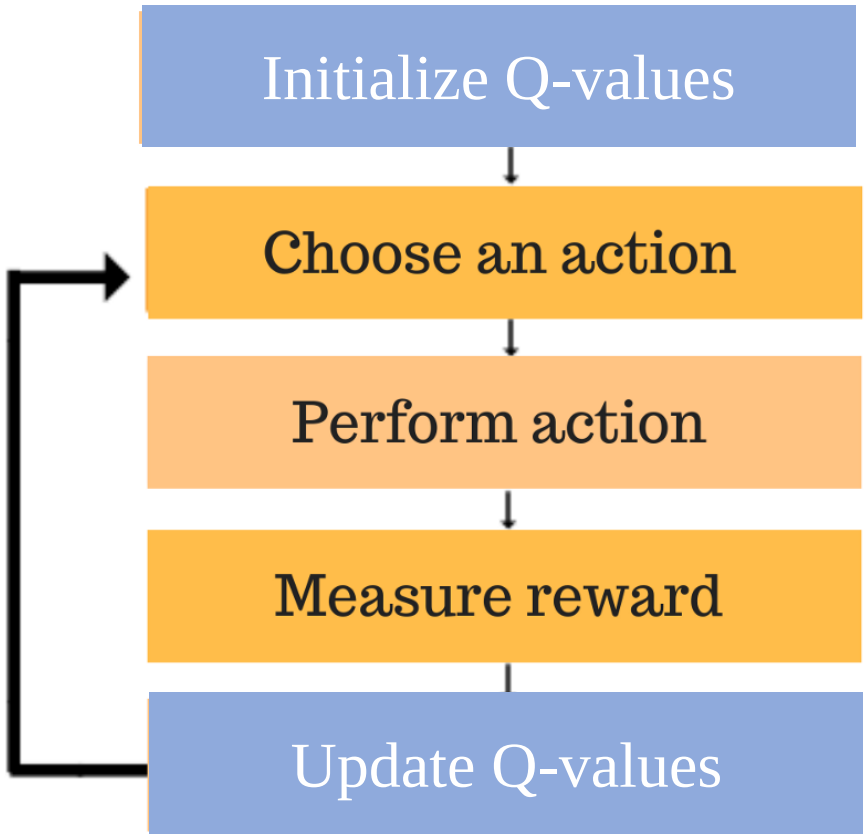
- How agents derive **efficient representations** of the environment from **high-dimensional inputs** and use them **to generalize** past experience to new situations
- Applicability **limited to** domains in which features can be **handcrafted** or with **fully observed**, low-dimensional state spaces



Source: doi: 10.1109/ICSMC.2009.5346114

Humans and other animals seem to solve this problem through a harmonious **combination of RL and hierarchical sensory processing** systems

Q-learning Algorithm



Learning rule:

possible actions — reward — next action

$$Q(s,a) := Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

current state — learning factor — discount factor — next state

Action-value (Q)-function

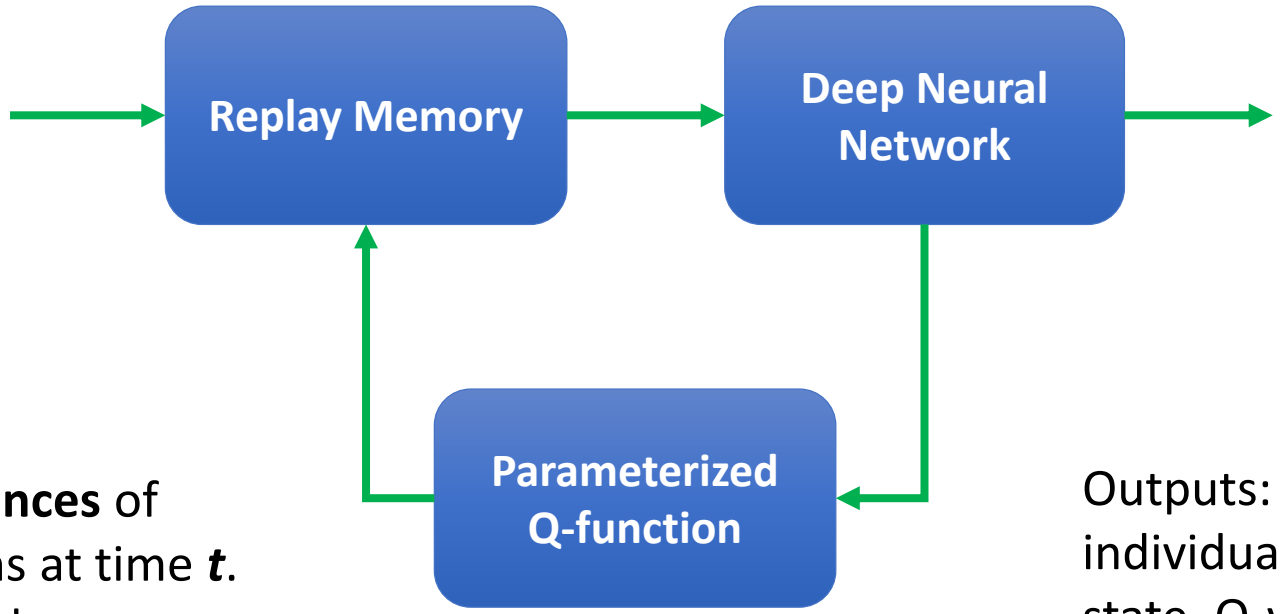
maximum sum of rewards

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

behaviour policy $\pi = P(a|s)$

Q-function is estimated **separately for each sequence**, without any generalization. Then, it is common to use a **linear function approximator** to estimate the Q-function

The Deep Q-network Method



Inputs: **complete sequences** of actions and observations at time ***t***. Finite **MDP** in which each sequence is a **distinct state**

Outputs: **predicted Q-values** of the individual actions for the input state. Q-values are computed for **all possible actions in a given state** with only a **single forward pass** through the network

Q-function Parametrization

Loss function for Q-learning update:

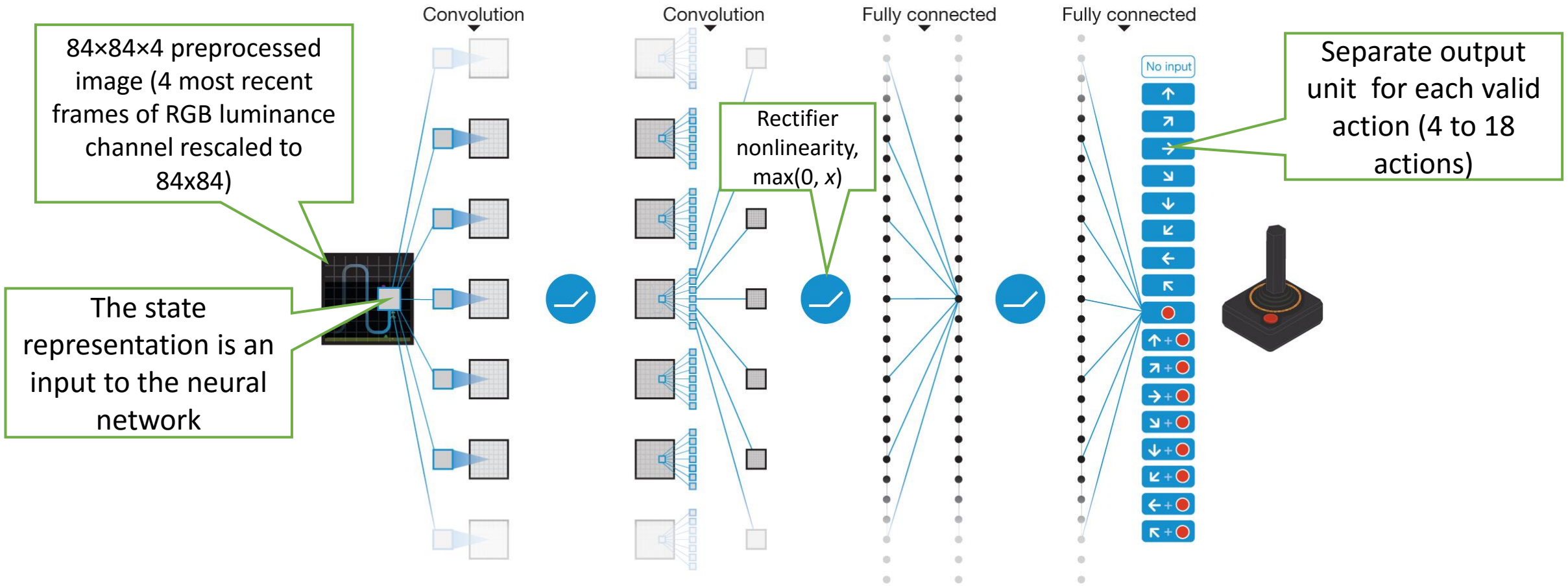
weights of the Q-network

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

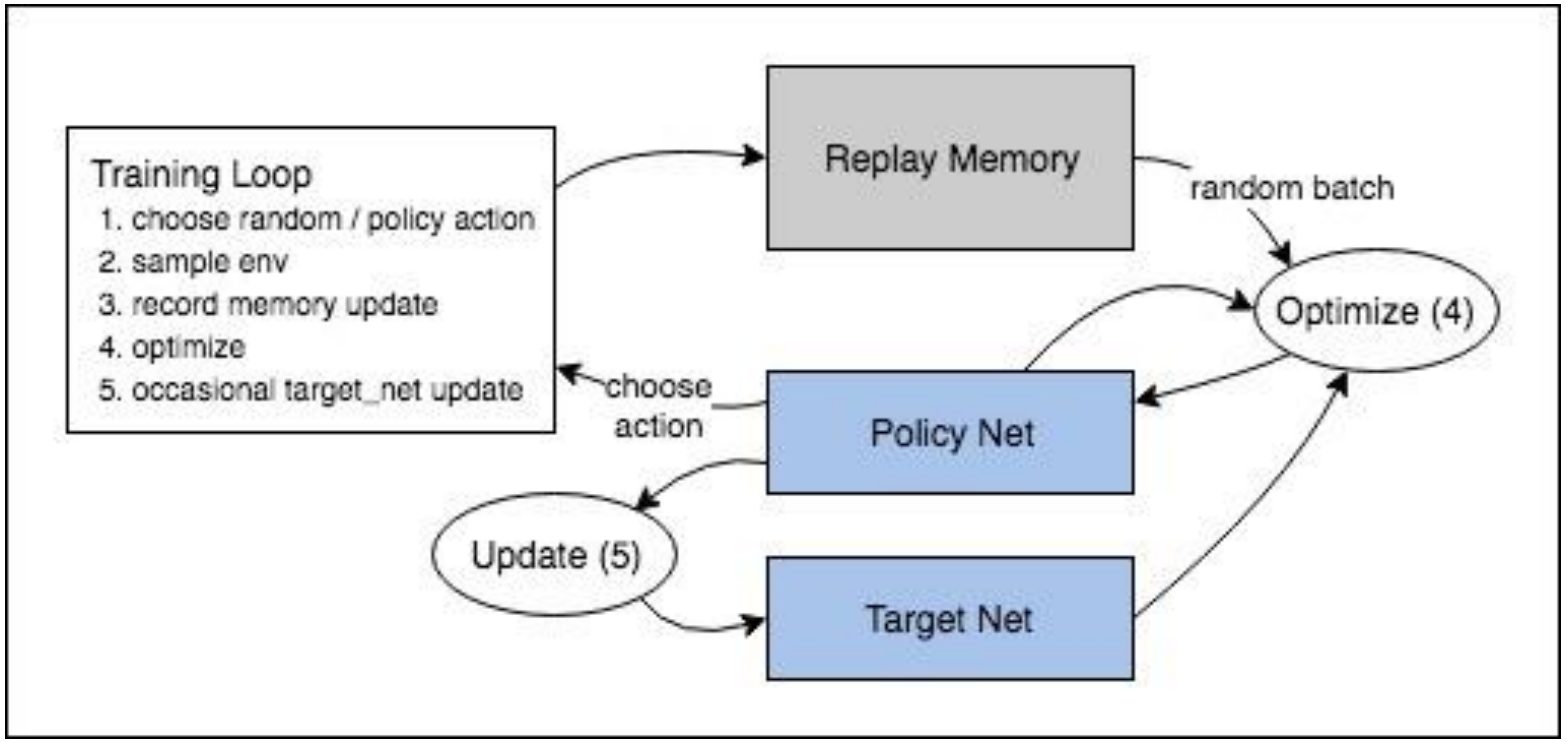
approximate target values

Targets depend on the network weights (not fixed before learning begins). At each stage of optimization, the weights from the previous iteration are fixed when optimizing the *i*th loss function

Q-network Architecture



Training Algorithm for DQN



https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

Training Algorithm for DQN (detailed)

DQN Agent Components

ε-greedy policy

To break correlations between samples

To improve stability of the network

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

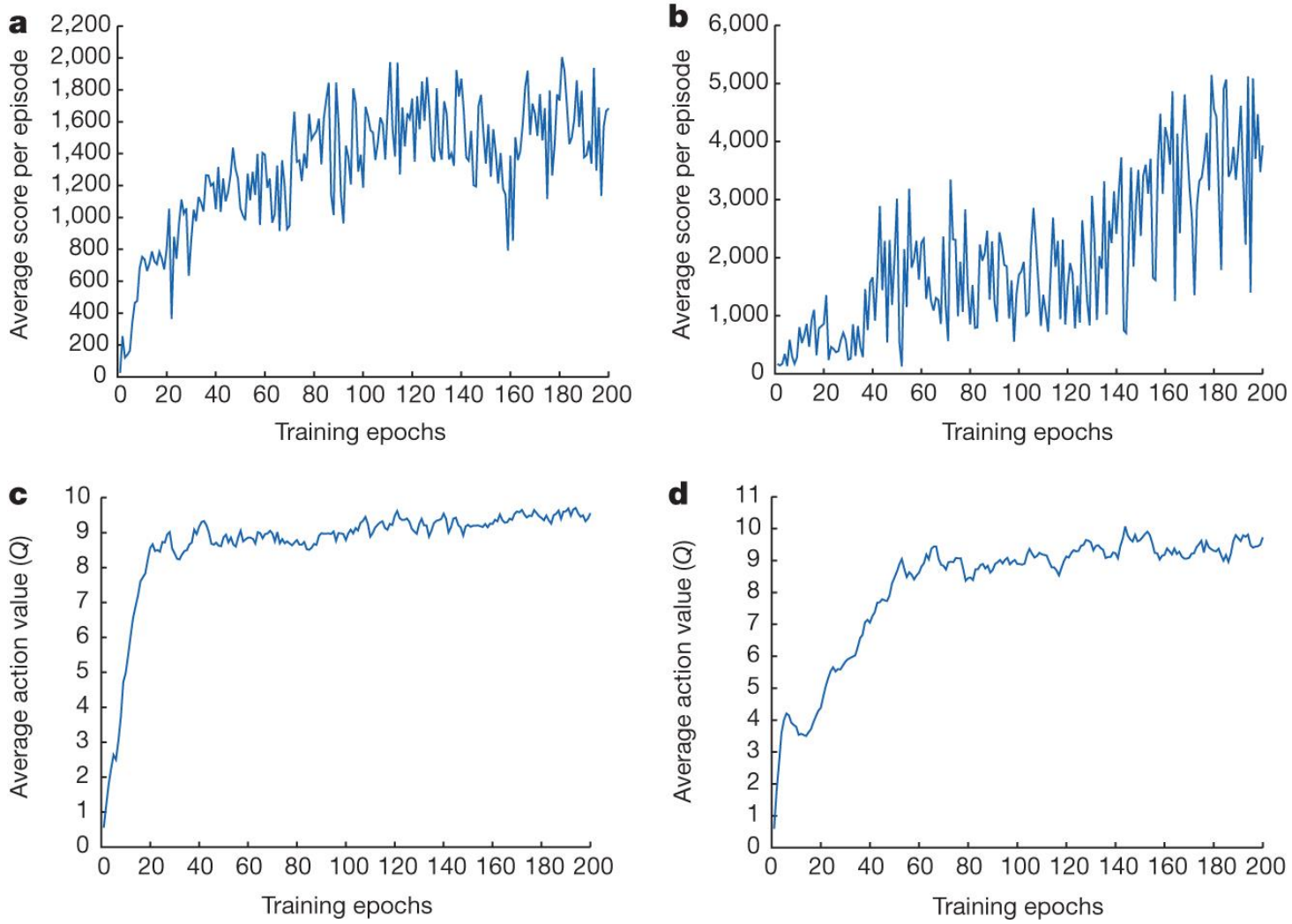
Evaluation of DQN

Demos

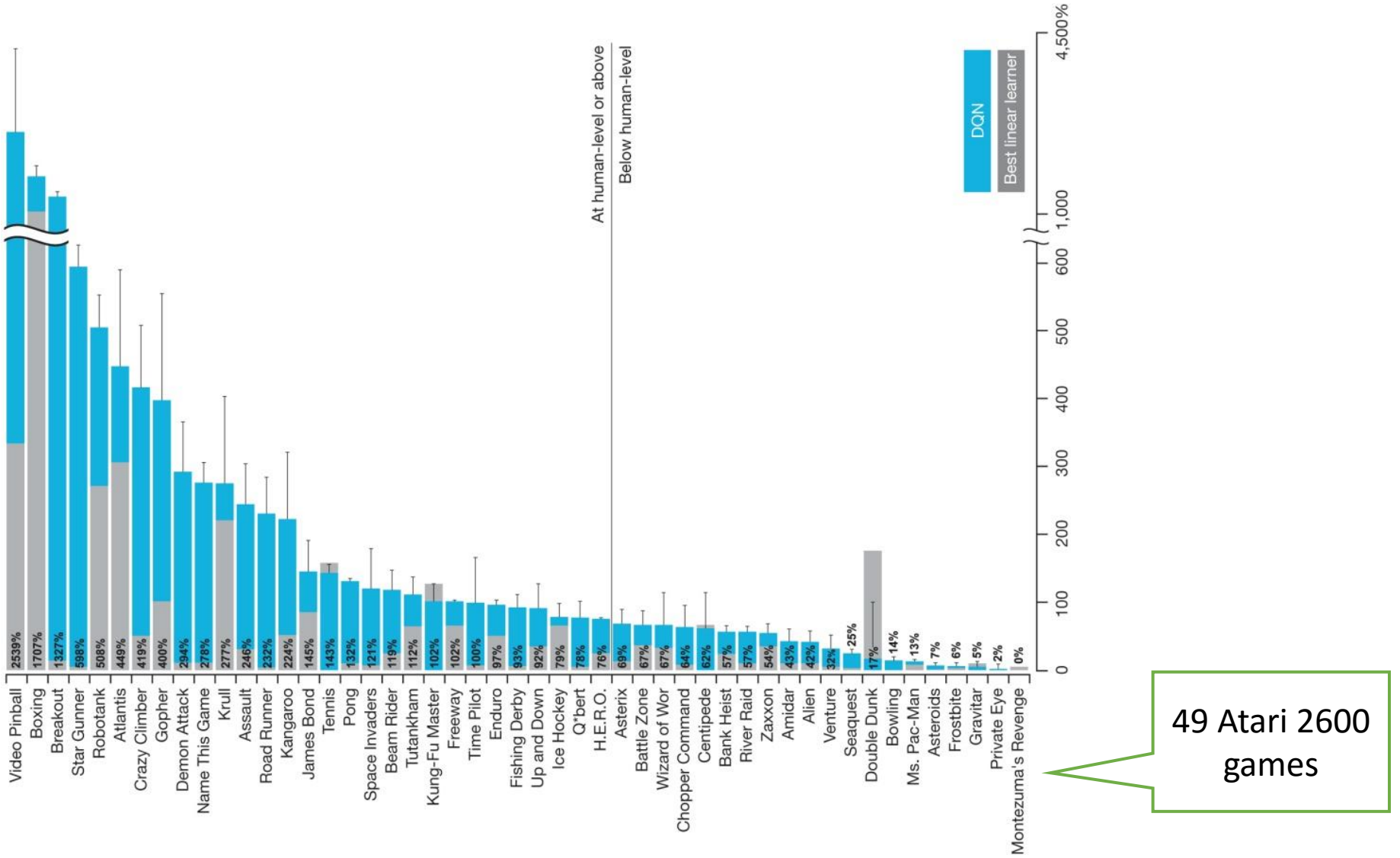
Improvement in the performance of DQN over training. After 600 episodes DQN finds and exploits the optimal strategy in this game, which is to make a tunnel around the side, and then allow the ball to hit blocks by bouncing behind the wall.

Performance of the DQN agent while playing the game of Space Invaders. The DQN agent successfully clears the enemy ships on the screen while the enemy ships move down and sideways with gradually increasing speed

Training Curves



DQN vs RL Best Methods

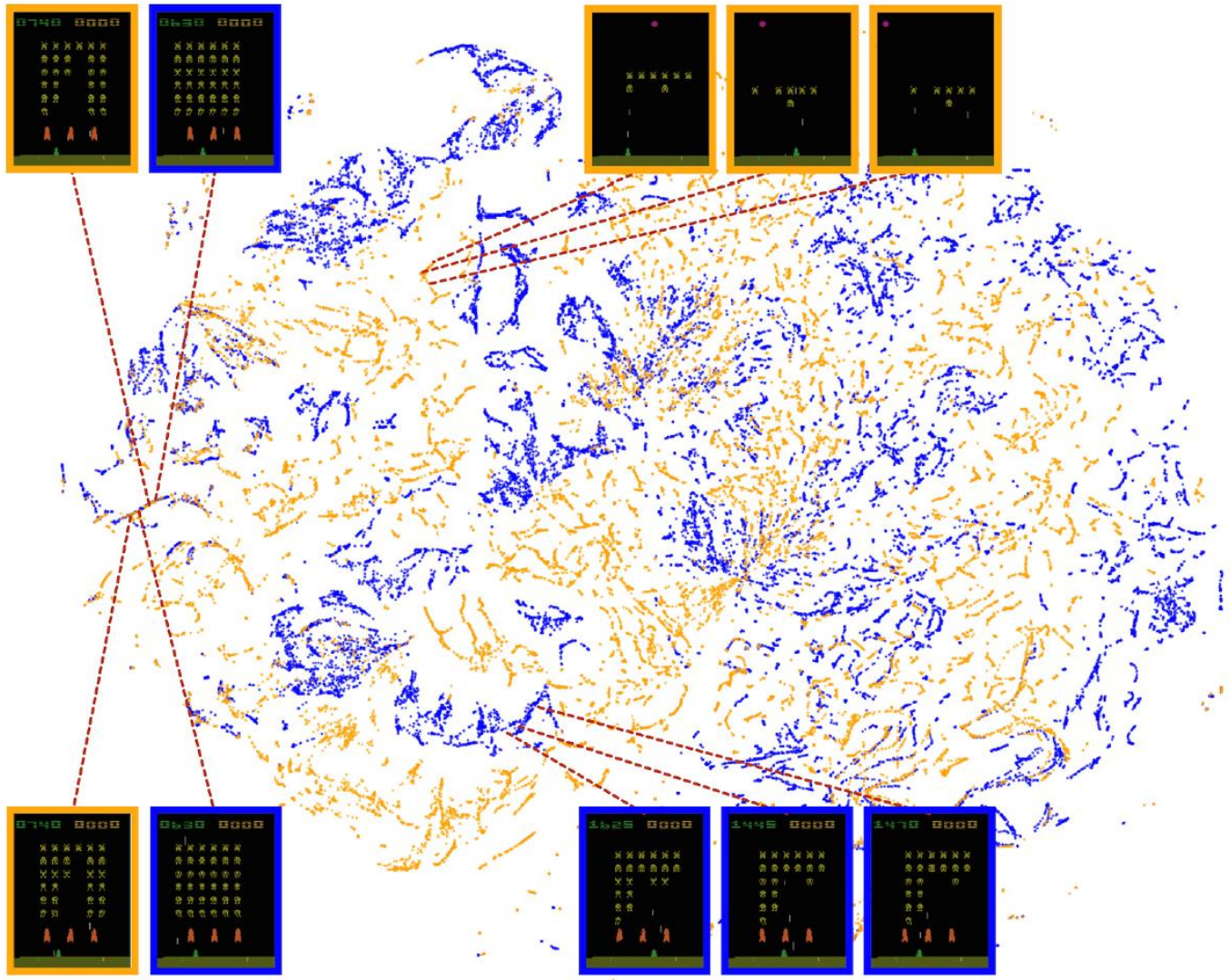


49 Atari 2600 games

Effects of DQN Agent Components

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

Representations



Two dimensional t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states

Human play

DQN play

Hyperparameters 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.

Open Discussion