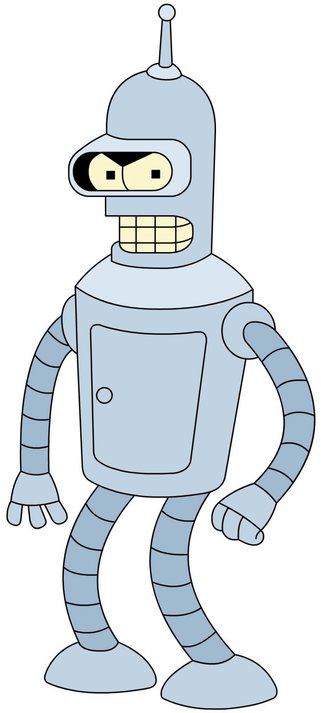
Reinforcement Learning

HSE, autumn - winter 2022

Lecture 4: Deep RL



Sergei Laktionov
slaktionov@hse.ru
LinkedIn

Background

1. Practical RL course by YSDA, week 4
2. Past iteration course, lecture 4
3. RL for Finance Book, Chapter 13

Recap: Q-Learning

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| < +\infty$

Recall the Bellman optimality equation for Q^* and apply it as an update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

TD-error

Bellman target

In this case, the learned action-value function, Q, directly approximates Q^*, the optimal action-value function, independent of the policy being followed.

Recap: Q-Learning

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| < +\infty$

Recall the Bellman optimality equation for Q^* and apply it as an update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

TD-error

Bellman target

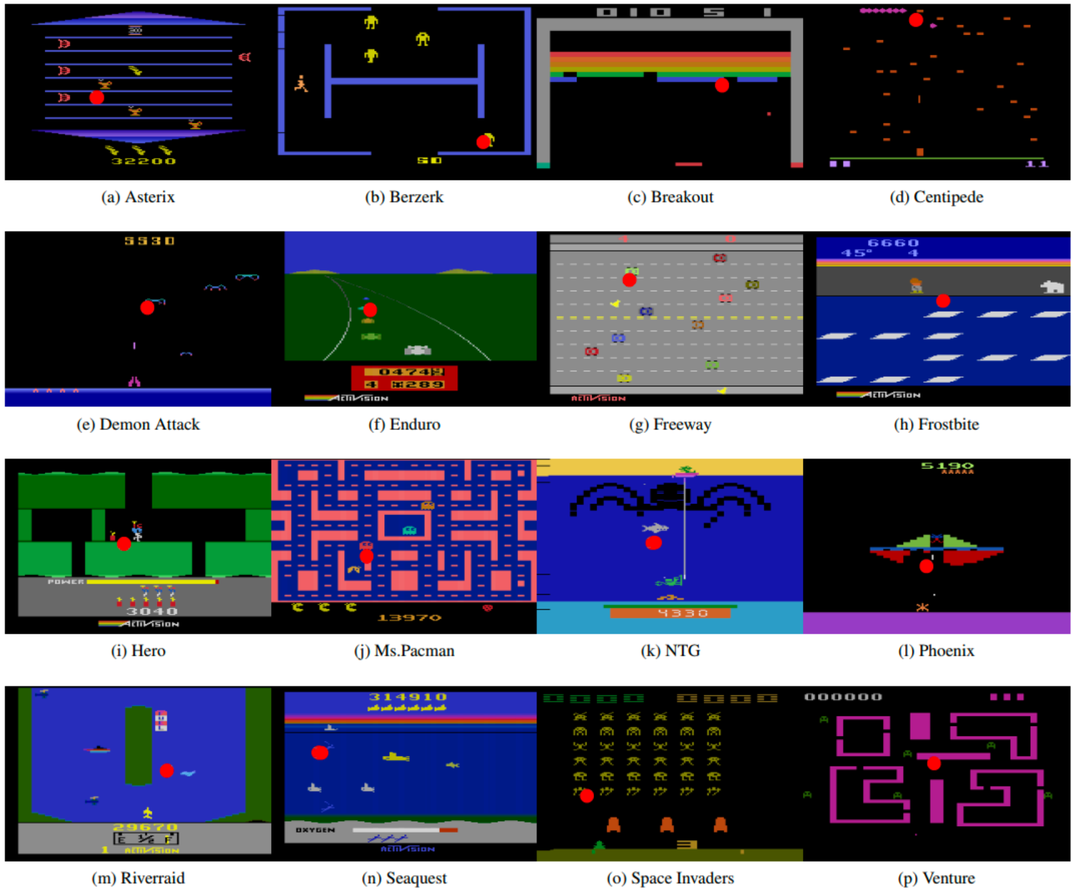
In this case, the learned action-value function, Q, directly approximates Q^*, the optimal action-value function, independent of the policy being followed.

Recall the trajectories are generated following the ε-greedy (behaviour) policy while the Q-function's update corresponds to the greedy (target) strategy. That is the reason why Q-learning is an off-policy method.

Atari

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| < +\infty$

1. More than 57 different games
2. Only frames are available
3. State space is actually finite but too large to apply tabular methods
4. Few actions are available



(a) Asterix (b) Berzerk (c) Breakout (d) Centipede

(e) Demon Attack (f) Enduro (g) Freeway (h) Frostbite

(i) Hero (j) Ms.Pacman (k) NTG (l) Phoenix

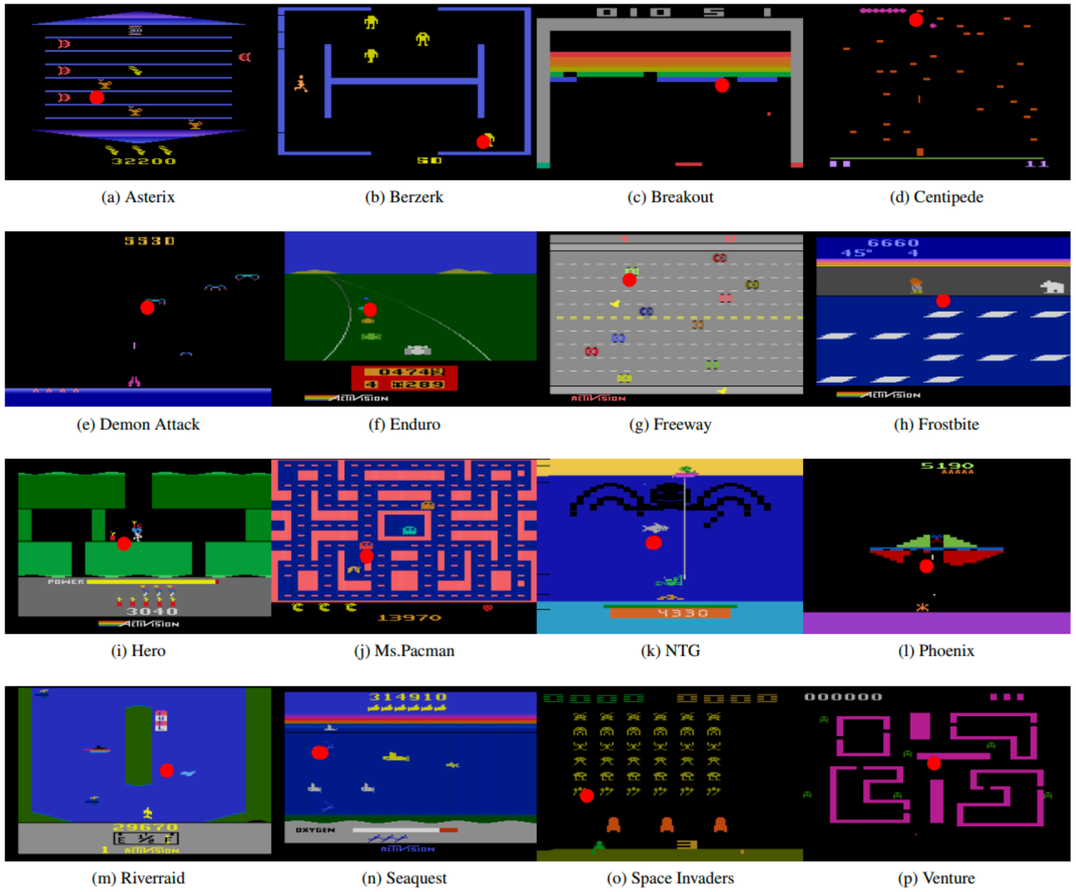
(m) Riverraid (n) Seaquest (o) Space Invaders (p) Venture

Source

Atari

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| < +\infty$

1. More than 57 different games
2. Only frames are available
3. State space is actually finite but too large to apply tabular methods
4. Few actions are available



(a) Asterix (b) Berzerk (c) Breakout (d) Centipede

(e) Demon Attack (f) Enduro (g) Freeway (h) Frostbite

(i) Hero (j) Ms.Pacman (k) NTG (l) Phoenix

(m) Riverraid (n) Seaquest (o) Space Invaders (p) Venture

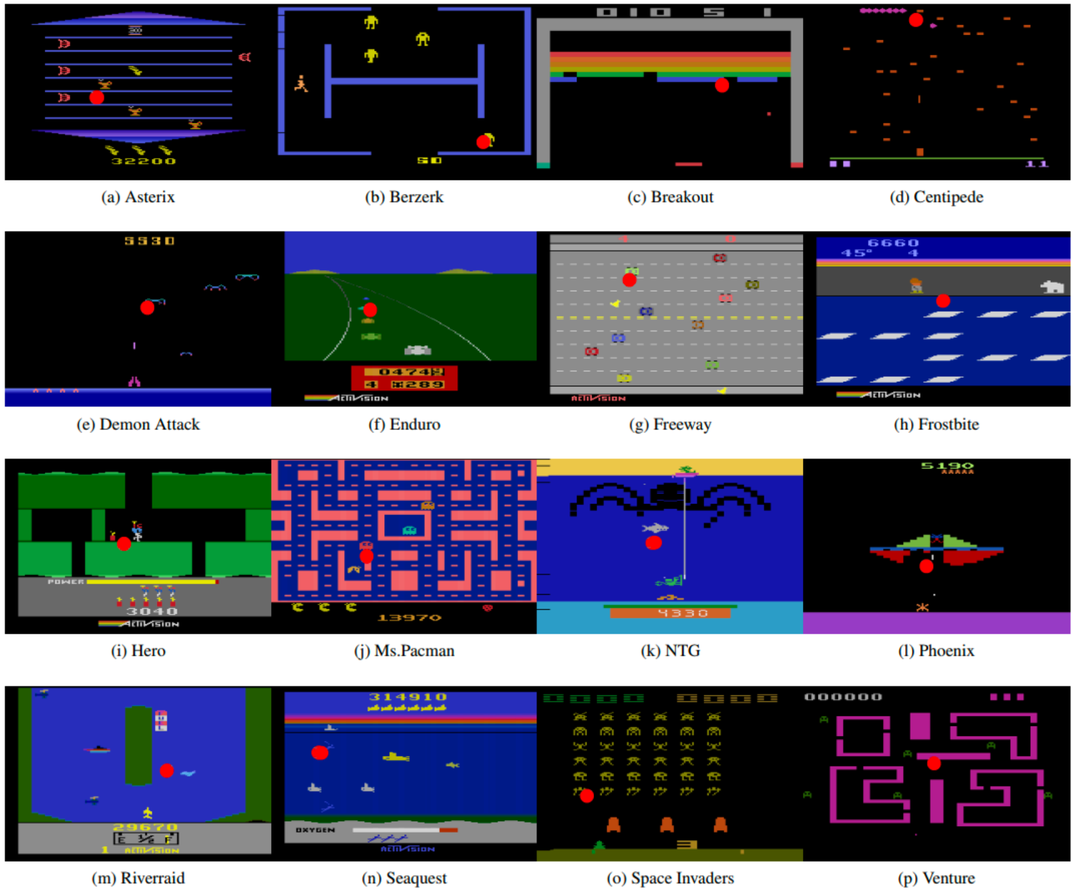
Source

The ultimate goal is to build a universal algorithm which can be applied to all of these environments without modifications and specific hyperparameters.

Atari

Setup: $|\mathcal{A}| < +\infty, |\mathcal{S}| < +\infty$

1. More than 57 different games
2. Only screenshots are observable
3. State space is actually finite but too large to apply tabular methods
4. Few actions are available



(a) Asterix (b) Berzerk (c) Breakout (d) Centipede

(e) Demon Attack (f) Enduro (g) Freeway (h) Frostbite

(i) Hero (j) Ms.Pacman (k) NTG (l) Phoenix

(m) Riverraid (n) Seaquest (o) Space Invaders (p) Venture

Source

Let's approximate action-value function with a neural network: $Q^*(s,a) \approx Q(s,a;\theta)$

Deep Q-Networks (DQN)

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

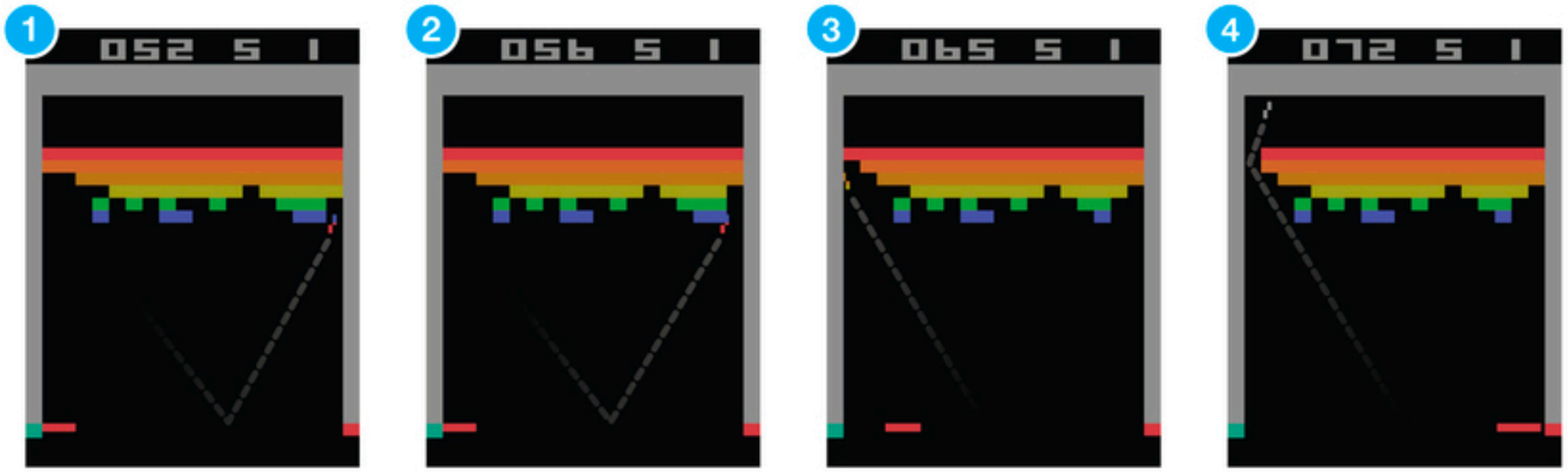
We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

Original paper

Is Atari Environment MDP?

Source

MDP from Frames



Source

Preprocessing

States:

- Crop image
- Grayscale
- Downsampling
- Stack 4 consecutive frames

Actions' frequency:

- MaxAndSkip
- Downsampling
- Sticky actions (we won't use)

Environment:

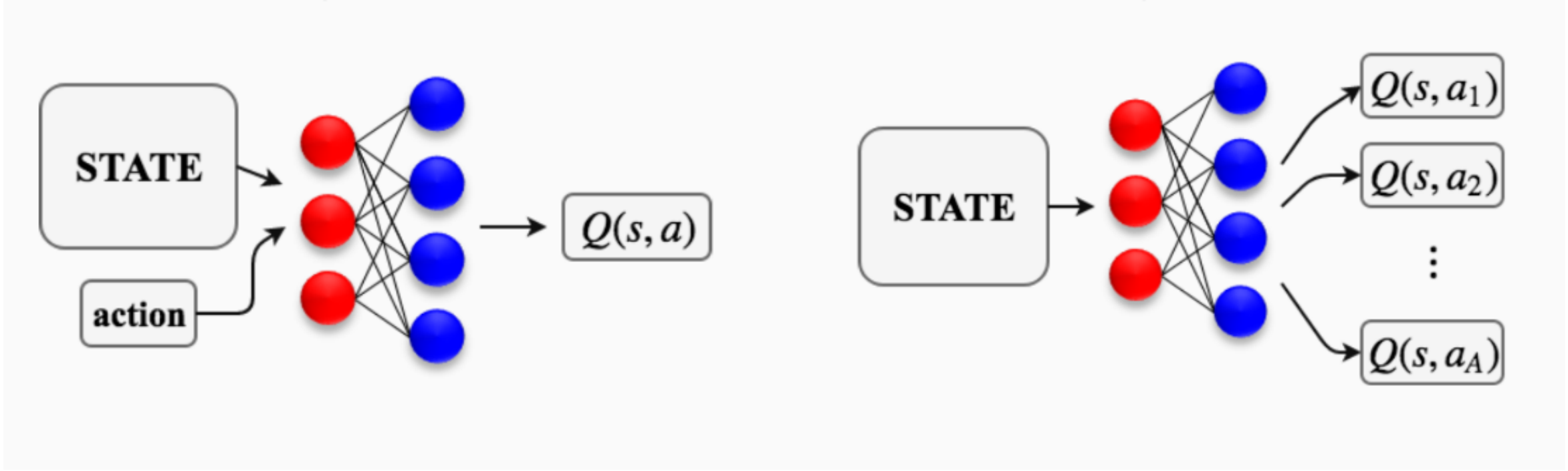
- EpisodicLife
- FireReset

Reward:

- ClipReward ($\{-1,0,1\}$)

Deep Q-network

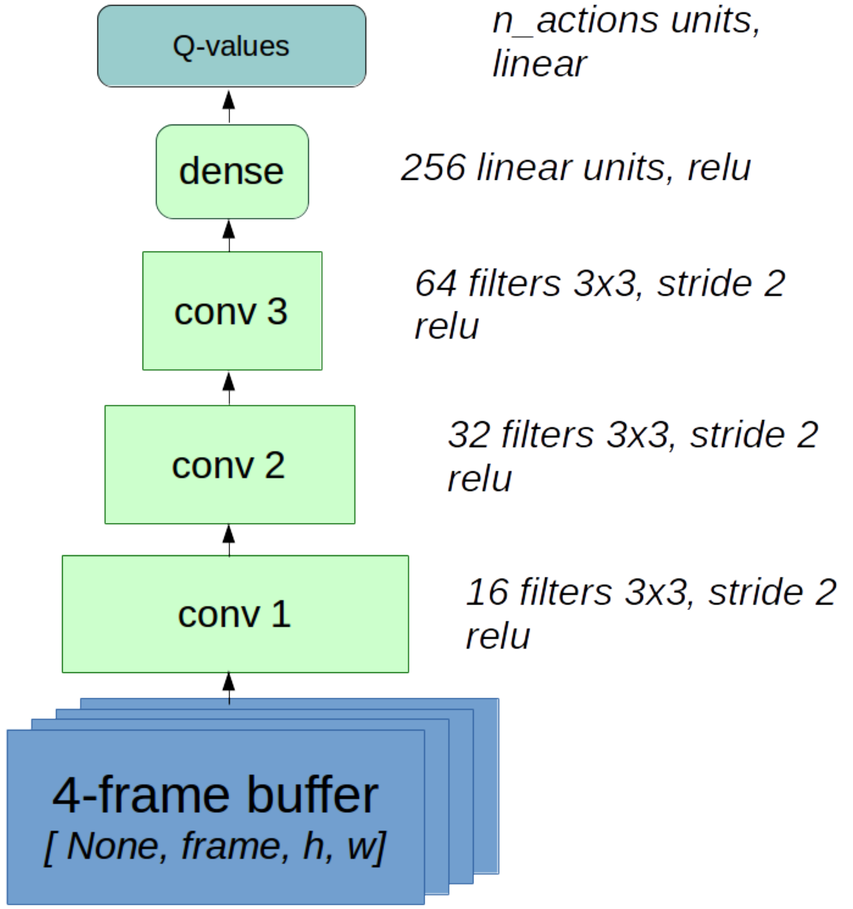
Choose your fighter:



Source

Deep Q-network

- No MaxPool
- No Dropout
- No BatchNorm



Source

Target

$Q^*(s,a) = \mathbb{E}[R_t + \gamma \max_a Q^*(S_{t+1}, a) \,|\, S_t = s, A_t = a]$ - Bellman equation

At each iteration i:

$y_i = \mathbb{E}_{s' \sim \mathscr{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a]$ - Bellman target, $\mathscr{E}$ is an environment

$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2]$ - MSE loss, $\rho(s, a)$ is the behaviour distribution

Gradients

$Q^*(s,a) = \mathbb{E}[R_t + \gamma \max_a Q^*(S_{t+1}, a) \,|\, S_t = s, A_t = a]$ - Bellman equation

At each iteration i:

$y_i = \mathbb{E}_{s' \sim \mathscr{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a]$ - Bellman target, $\mathscr{E}$ is an environment

$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2]$ - MSE loss, $\rho(s,a)$ is the behaviour distribution

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

Gradients

$Q^*(s,a) = \mathbb{E}[R_t + \gamma \max_a Q^*(S_{t+1}, a) \,|\, S_t = s, A_t = a]$ - Bellman equation

At each iteration i:

$y_i = \mathbb{E}_{s' \sim \mathscr{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a]$ - Bellman target, $\mathscr{E}$ is an environment

$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2]$ - MSE loss, $\rho(s,a)$ is the behaviour distribution

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

Gradients

$Q^*(s,a) = \mathbb{E}[R_t + \gamma \max_a Q^*(S_{t+1}, a) \,|\, S_t = s, A_t = a]$ - Bellman equation

At each iteration i:

$y_i = \mathbb{E}_{s' \sim \mathscr{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a]$ - Bellman target, $\mathscr{E}$ is an environment

$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2]$ - MSE loss, $\rho(s,a)$ is the behaviour distribution

TD-error

Step size

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

Bellman target

Gradients

$$Q^*(s,a) = \mathbb{E}[R_t + \gamma \max_a Q^*(S_{t+1}, a) \,|\, S_t = s, A_t = a]$$ - Bellman equation

At each iteration i:

$$y_i = \mathbb{E}_{s' \sim \mathscr{E}}[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|\, s, a]$$ - Bellman target, $\mathscr{E}$ is an environment

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2]$$ - MSE loss, $\rho(s,a)$ is the behaviour distribution

TD-error

Step size

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

Bellman target

Let’s apply SGD!

Target Network

If we use weights from the previous step targets are highly non-stationary.

Let's update weight of the network for target generation (target network) every K steps or apply soft-updates with parameter β:

A. $\theta^{-} \leftarrow \theta$ every K SGD iterations

B. $\theta^{-} \leftarrow (1-\beta)\theta^{-} + \beta\theta$ on each iteration

Target Network

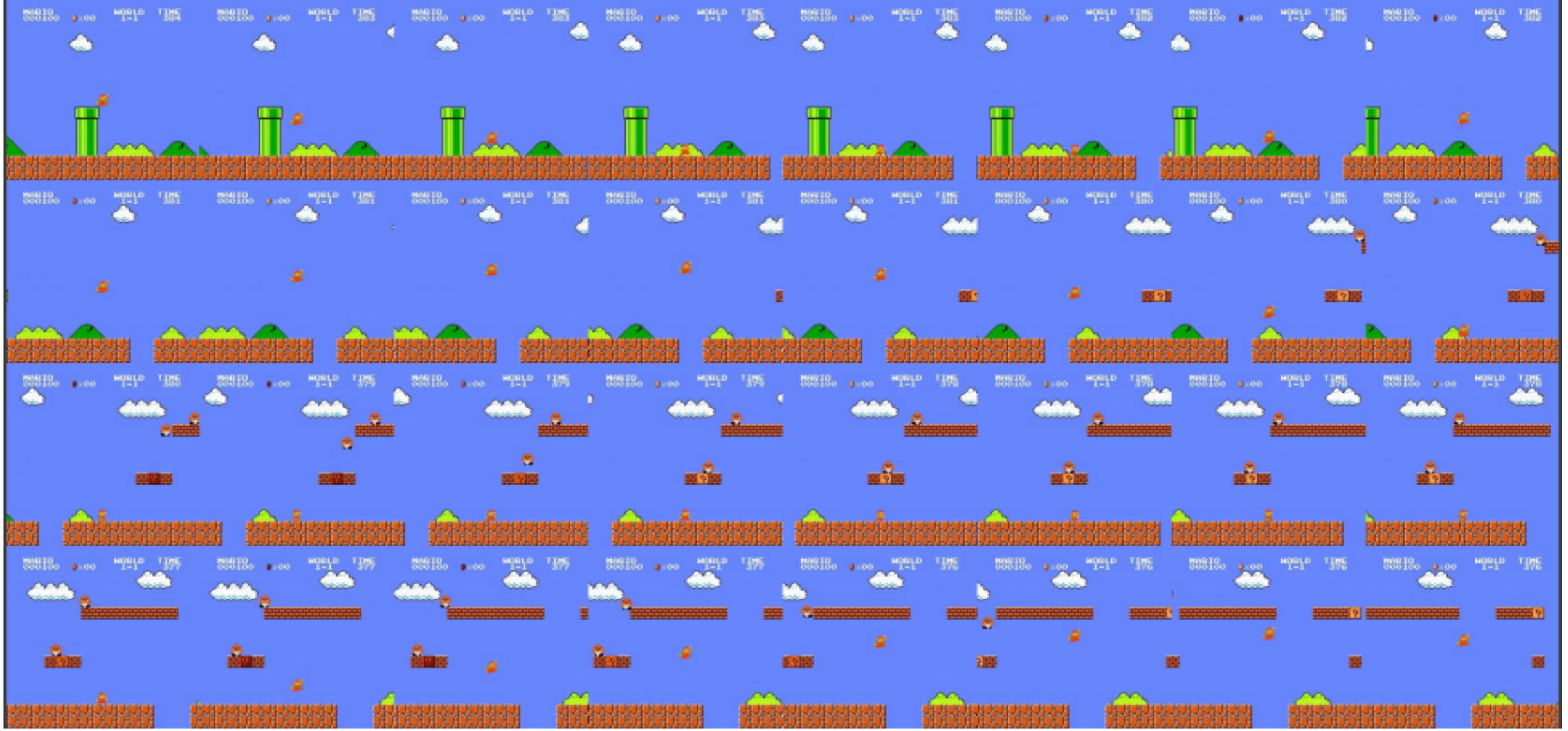
If we use weights from the previous step targets are non-stationary.

Let's update weight of the network for target generation (target network) every K steps or apply soft-updates with parameter β:

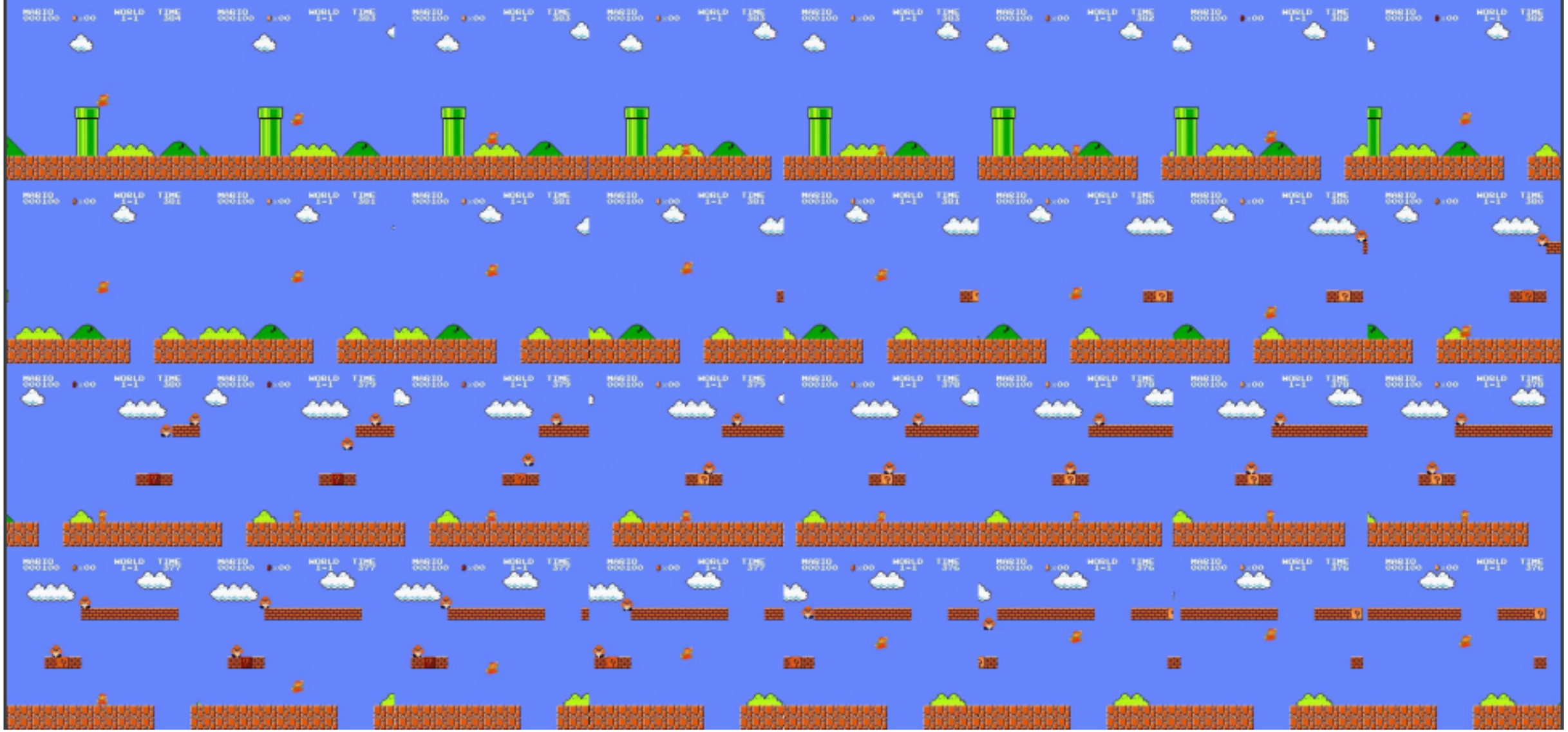
A. $\theta^- \leftarrow \theta$ every K SGD iterations

B. $\theta^- \leftarrow (1-\beta)\theta^- + \beta\theta$ on each iteration

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}_{s,a\sim\rho(.);s'\sim\mathscr{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta_t)\right) \nabla_{\theta_t} Q(s, a; \theta_t)\right]$$

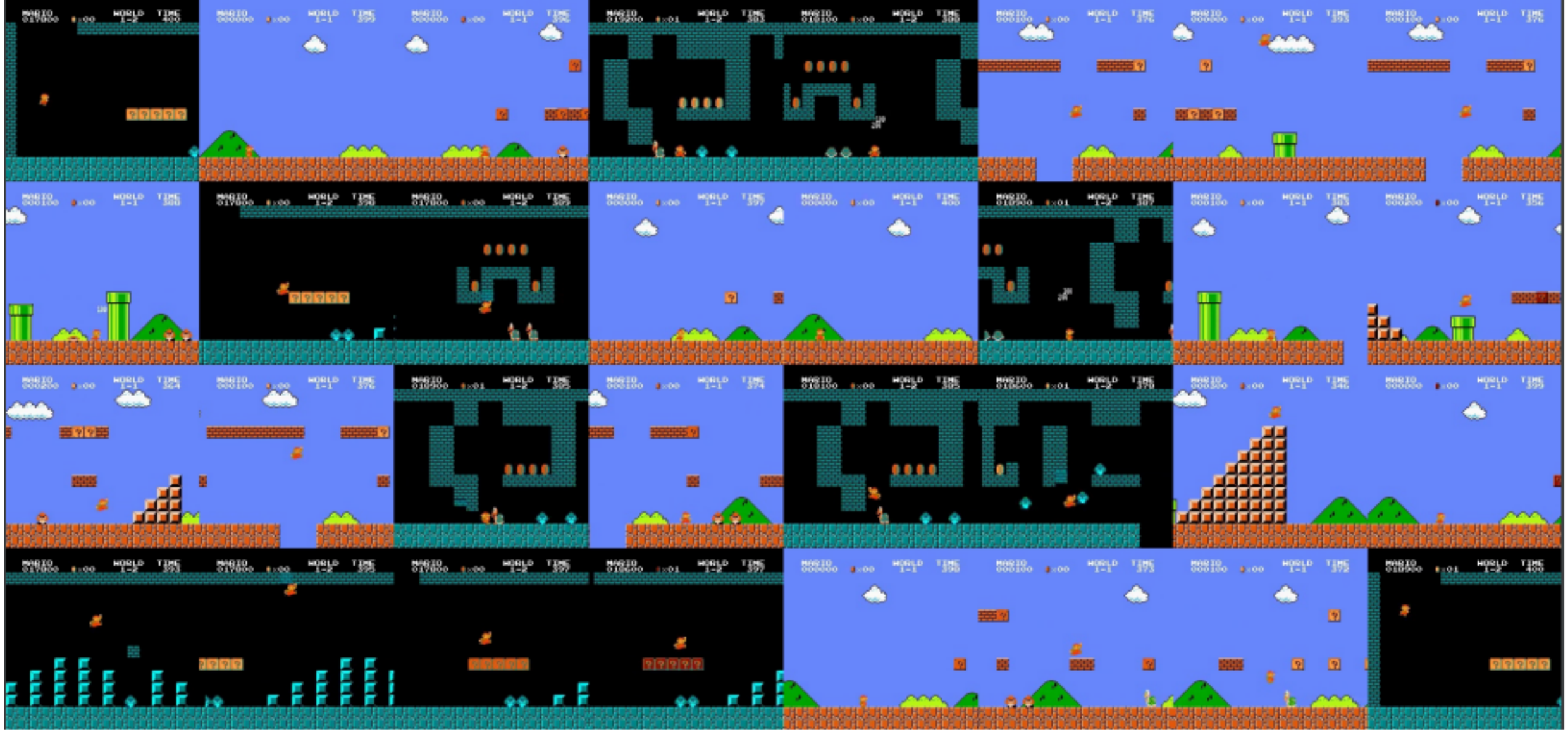


Source



Source

Consecutive samples are extremely correlated so batch’s elements are not i.i.d.



Let’s sample randomly from the Experience Replay Buffer
which stores played transitions

Exploration

Recall so-called ε-greedy policy:

$$\pi = \begin{cases} \text{select random action with probabily } \varepsilon \\ \text{select greedy action with probabily } 1 - \varepsilon \end{cases}$$

We use ε-greedy strategies to avoid being stuck in local optima

DQN Algorithm

Initialise $Q(s, a; \theta)$; $\theta^- = \theta$; replay buffer D is empty;

Observe s_0;

$t = 0,1,\ldots$

- Take action a_t sampled from the ε-greedy policy w.r.t. $Q(s, a; \theta_t)$;
- Observe $(r_t, s_{t+1}, \text{done}_{t+1})$, store $(s_t, a_t, r_t, s_{t+1}, \text{done}_{t+1})$ in D;
- Sample batch of transitions $(s_j, a_j, r_j, s_{j+1}, \text{done}_{j+1})_{j=1}^{B}$ from D (j and t are not connected)
- $y_j = r_j + \gamma(1 - \text{done}_{j+1}) \max_{a'} Q(s_{j+1}, a'; \theta^-)$;
- Perform a gradient descent step: $\theta_{t+1} = \theta_t - \frac{\alpha}{B} \sum_{j=1}^{B} \nabla_{\theta_t} (y_j - Q(a_j, s_j; \theta_t))^2$
- If k mod $K = 0$ update target network: $\theta^- \leftarrow \theta_t$

Rainbow

Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel DeepMind
Joseph Modayil DeepMind
Hado van Hasselt DeepMind
Tom Schaul DeepMind
Georg Ostrovski DeepMind

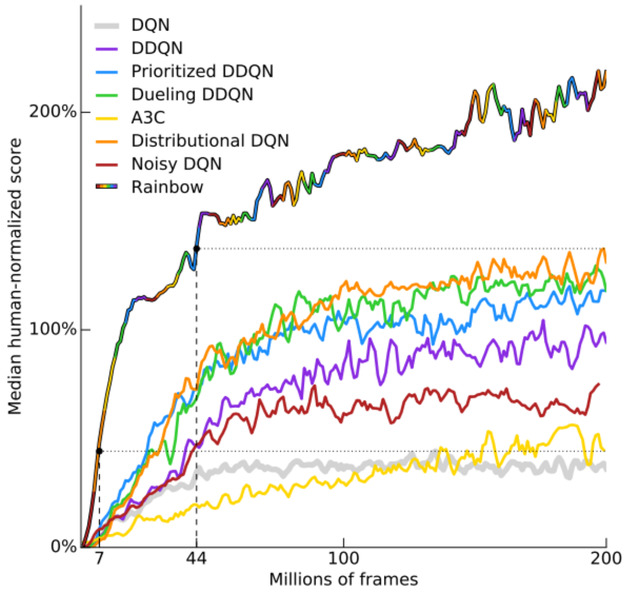
Will Dabney DeepMind
Dan Horgan DeepMind
Bilal Piot DeepMind
Mohammad Azar DeepMind
David Silver DeepMind

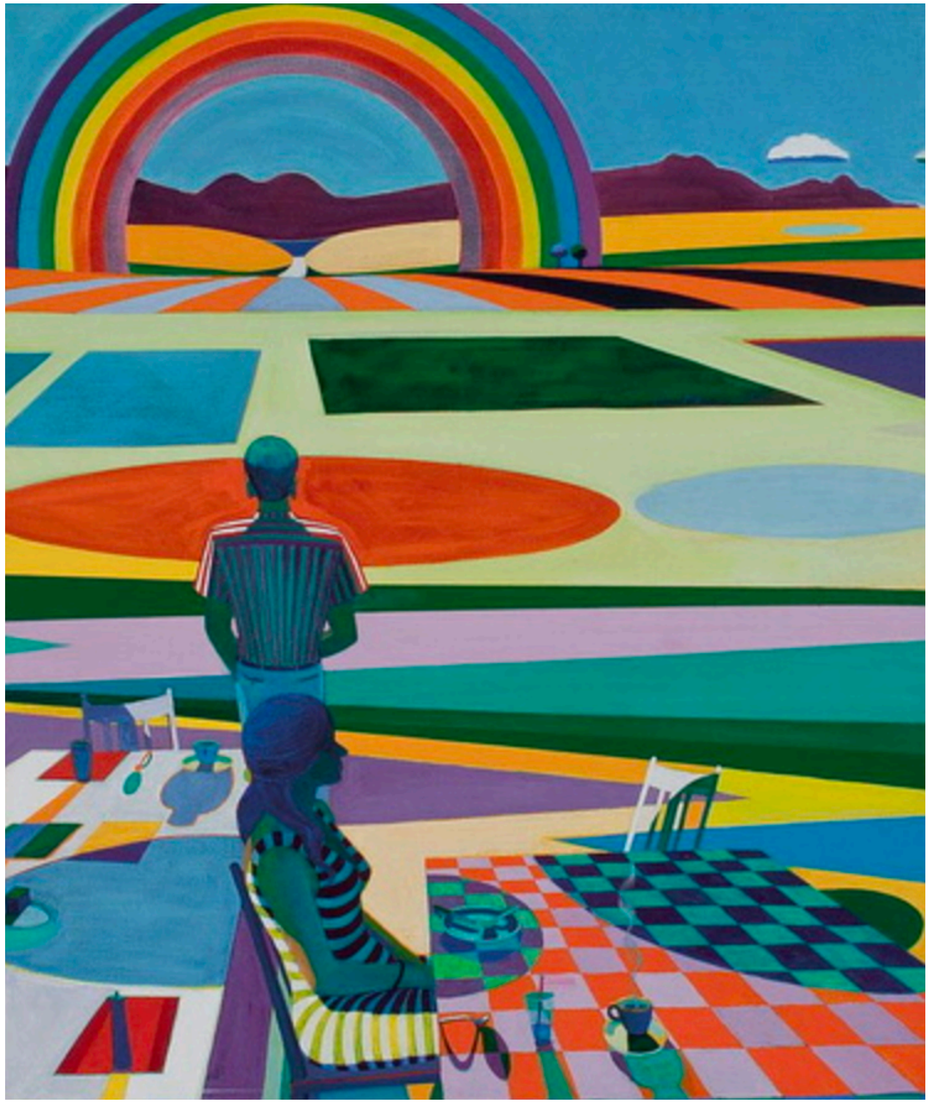
Abstract

The deep reinforcement learning community has made several independent improvements to the DQN algorithm. However, it is unclear which of these extensions are complementary and can be fruitfully combined. This paper examines six extensions to the DQN algorithm and empirically studies their combination. Our experiments show that the combination provides state-of-the-art performance on the Atari 2600 benchmark, both in terms of data efficiency and final performance. We also provide results from a detailed ablation study that shows the contribution of each component to overall performance.

Introduction

The many recent successes in scaling reinforcement learning (RL) to complex sequential decision-making problems were kick-started by the Deep Q-Networks algorithm (DQN; Mnih et al. 2013, 2015). Its combination of Q-learning with convolutional neural networks and experience replay enabled it to learn, from raw pixels, how to play many Atari games at human-level performance. Since then, many exten





Original paper

Source

Overstimation Bias

The overestimation of the Q-function by the algorithms which can be caused by several reasons:

1. Maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to a significant positive bias due to approximation error.

2. Due to using the same samples (plays) both to determine the maximizing action and to estimate its value.

Action selection

$$\max_{a'} Q(s', a') = Q(s', \text{argmax} Q(s', a'))$$

Action evaluation

Overstimation Bias

The overestimation of the Q-function by the algorithms which can be caused by several reasons:

1. Maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to a significant positive bias due to approximation error.

2. Due to using the same samples (plays) both to determine the maximizing action and to estimate its value.

Action selection

$$\max_{a'} Q(s', a') = Q(s', \text{argmax} Q(s', a'))$$

Action evaluation

Double DQN

Let's decouple action selection and action evaluation. Like in the tabular setting we can use two weakly correlated networks:

$$y_1 = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_1); \theta_2)$$

$$y_2 = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_2); \theta_1)$$

with independent buffers D_i, but it can be too expensive…

Double DQN

Let's decouple action selection and action evaluation. Like in the tabular setting we can use two weakly correlated networks:

$$y_1 = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_1); \theta_2)$$

$$y_2 = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_2); \theta_1)$$

with independent buffers D_i, but it can be too expensive…

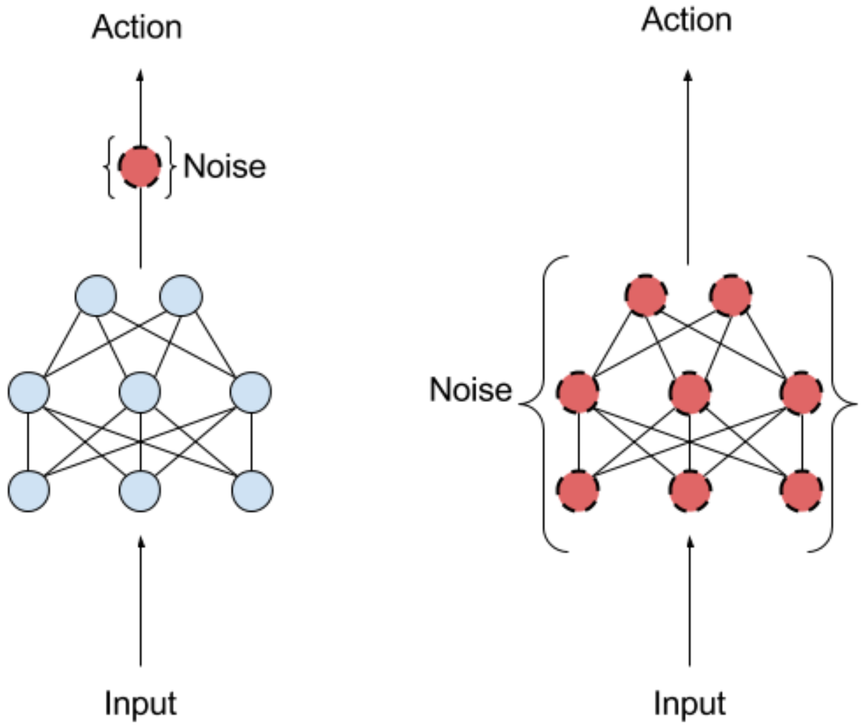
We can use target network as the second network:

$$y = r + \gamma Q(s', \text{argmax}_{a'} Q(s, a', \theta_t); \theta_t^-)$$

Noisy Networks

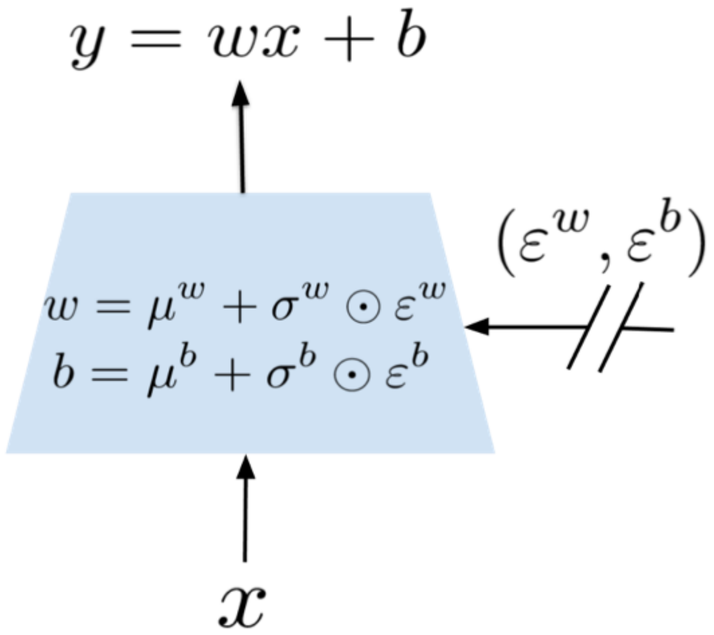
Issues:

- State-independent exploration
- ε-greedy is naive



Source

Noisy Networks



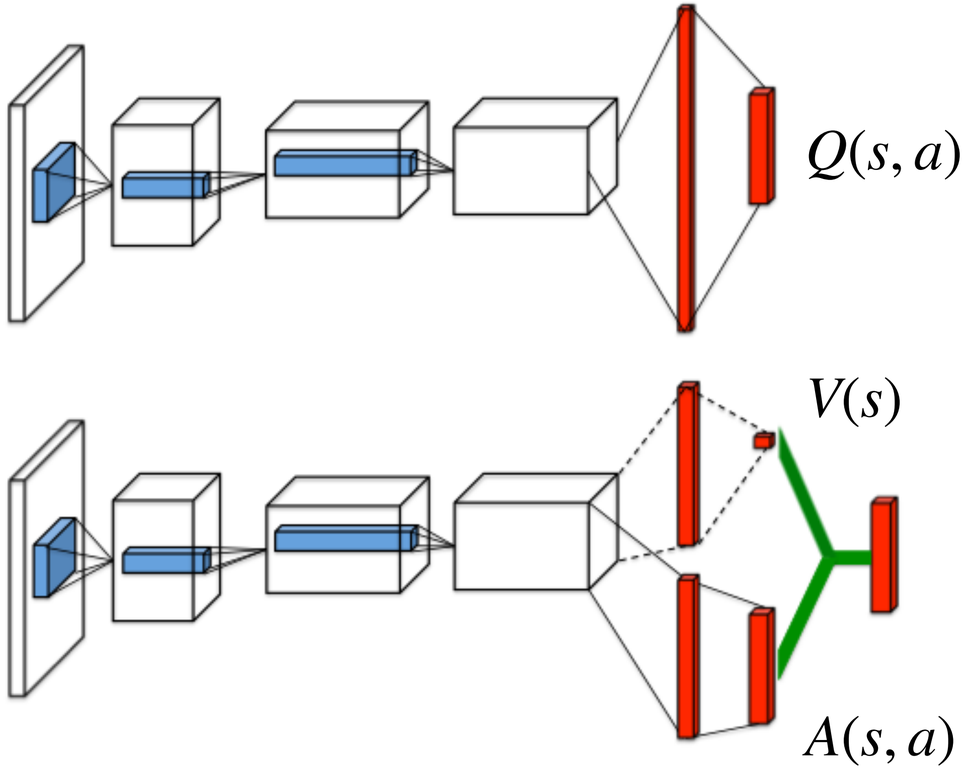
$$\epsilon^w, \epsilon^b \sim \mathcal{N}(0,1)$$

Source

Dueling DQN

$A(s,a) = Q(s,a) - V(s)$ is a relative measure of importance of each action, advantage.

$$Q(s,a) = V(s) + A(s,a)$$



Source

Multi-step DQN

$$y = \boxed{r + \gamma r' + \gamma r'' + \dots} + \gamma^n \max_{a^{(n)}} Q(s^{(n)}, a^{(n)})$$

We assume that all transitions are generated with greedy policy but it’s not the case.

Can’t be done off-policy, no theoretical guaranties!

Other Improvements

1. Prioritized Experience Replay
2. Distributional RL

Prioritized Experience Replay

The uniform sampling from the experience replay make an agent replay transitions at the same frequency that were originally experienced, regardless of their significance.

Prioritized Experience Replay

The uniform sampling from the experience replay make an agent replay transitions at the same frequency that were originally experienced, regardless of their significance.

$\delta_i = y_i - Q(s, a; \theta)$ - TD-error

$p_i = |\delta_i| + \epsilon, \epsilon > 0$

$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ is a the probability of sampling transition i

However we induce a bias in a Q-function approximation.

Prioritized Experience Replay

The uniform sampling from the experience replay make an agent replay transitions at the same frequency that were originally experienced, regardless of their significance.

$\delta_i = y_i - Q(s, a; \theta)$ - TD-error

$p_i = |\delta_i| + \epsilon, \epsilon > 0$

$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ is a the probability of sampling transition i

However we induce a bias in a Q-function approximation.

We can correct this bias by using Importance-Sampling (IS) weights

$w_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^\beta$ with beta annealing from 0 to 1 and $w_i\delta_i$ instead of δ_i.

Thank you for your attention!