REINFORCEMENT LEARNING WITH DQL

Ketkov S.
Kondratyev N.
Makarova O.
Pribytkina D.
Semenov S.

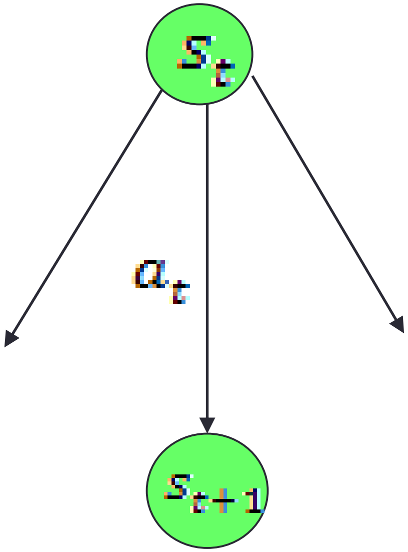
October, 2017

Abstract

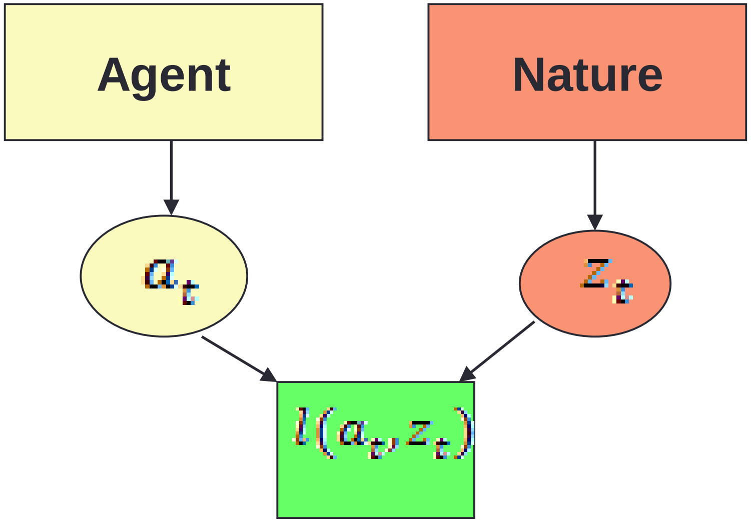
In this project we use a deep convolutional neural network for approximation of the optimal action-value function in reinforcement learning environment. In particular, we use ε-greedy policy based on Q-value function, while Q-values are calculated on the basis of reward rate with the help of neural network techniques. We tested our agent on the challenging domain of classic Atari 2600 games. We demonstrate that the deep Q-network agent, receiving only the pixels and the game score as inputs, achieves a level comparable to that of professional human players.

Preliminaries

We consider an agent (or a game player) that iteratively interacts with the environment. At each time period $t \in \{1,..,T\}$ the agent is at state $s_t \in S$ and chooses one of possible actions $a_t \in A$, where S and A are state and action spaces, respectively [1]. Usually, environment dynamics are given by an MDP model (see, e.g., [1]), where transition probabilities after choosing an action are given by Markov rule. We also predetermine initial and terminal states, that is the agent performs actions until reaching the terminal (finish) state.



Preliminaries



The interaction with environment takes place by means of rewards. Specifically, after choosing an action $a_t \in A(s_t)$ the nature forms a feedback z_t, and the agent suffers loss $l(a_t, z_t)$. We refer to [2] for different types of feedback and types of losses. Specifically, in this project we suppose that the agent gets to know the loss only after taking an action and the loss is deterministic.

Problem formulation

The agent’s goal is to maximize his total reward for *T* rounds, i.e., to maximize expectation of the following objective:

$$R_{total} = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ \dots$$

Parameter *T*, or time horizon may not be predetermined. Therefore, we use parameter ɣ in [0,1] to guarantee convergence of the sum. Here, $R(s_t)$ denotes the reward of taking action a_t at state s_t.

The main issue here is that it is necessary to find a compromise between exploration and exploitation (see, e.g., [1,2]), i.e., to understand whether to choose currently greedy or random actions.

Analysis of existing approaches

Here, we discuss simple and intuitive approaches to solving the general problem. The simplest way is to apply random policy, i.e., to choose random actions at each state. However, random approach does not take into account the structure of underlying problem. Another way is to use greedy policy, but this approach does not explore the action space. Therefore, some mixed policies can be used [1], such as:

- ε-greedy policy (we denote exploration probability by ε and with probability 1- ε follow a greedy policy);
- soft-max policy (we use Gibbs distribution, where the parameter corresponds to “degree” of randomness)

Q-learning (basic)

Here, we provide an introduction to reinforcement learning, based on Q-learning techniques. This approach is first introduced in [3] and is appeared to be one of the most popular approaches in practice [1].

First, we define Q-value function as:

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

Specifically, we maximize average cost of transition from current state to the terminal state. Most of existing approaches are aimed to approximate this function for each state and action. Then iteration of Q-learning algorithm is the following [1]:

$$Q(s,a) \leftarrow r + \gamma \max_{a' \in A(s')} Q(s',a')$$

Along with calculation of Q-values a step of ε-greedy policy is performed based on previous values of function Q.

Q-learning with DNN

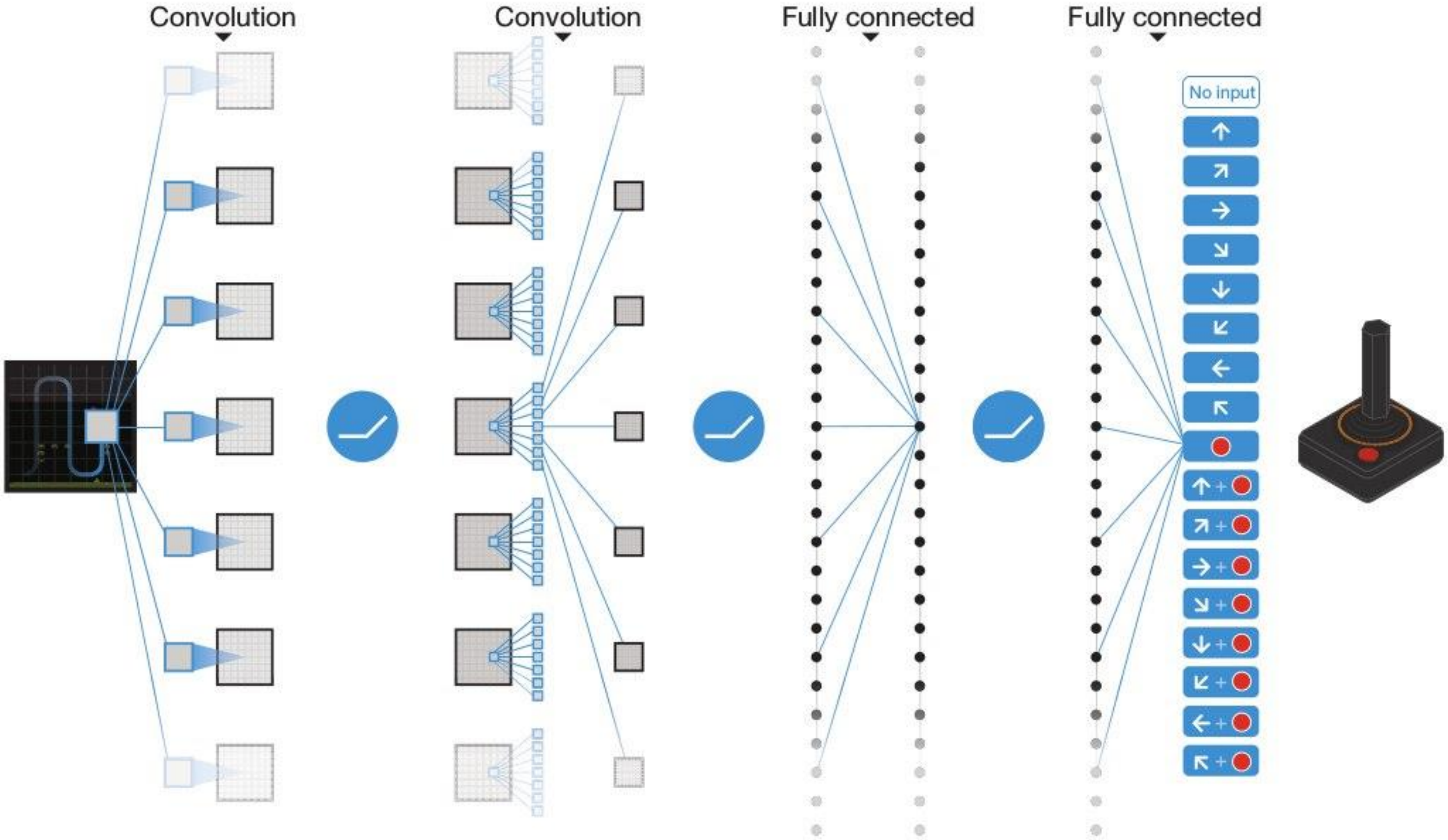
Simple Q-learning based on the table of Q-values is rather inefficient in the case where the cardinality of state space is large. Thus, some approximation techniques are used, for instance, approximation with deep neural networks [1]. Actually, we make Q-values dependent on some parameters that are weights in the neural network. Then, we formulate the loss function as follows [3]:

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

Using existing algorithms for neural networks (e.g., SGD and back propagation), now, we may approximate Q-values more effectively.

Architecture

Schematic illustration of the convolutional neural network*



* Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

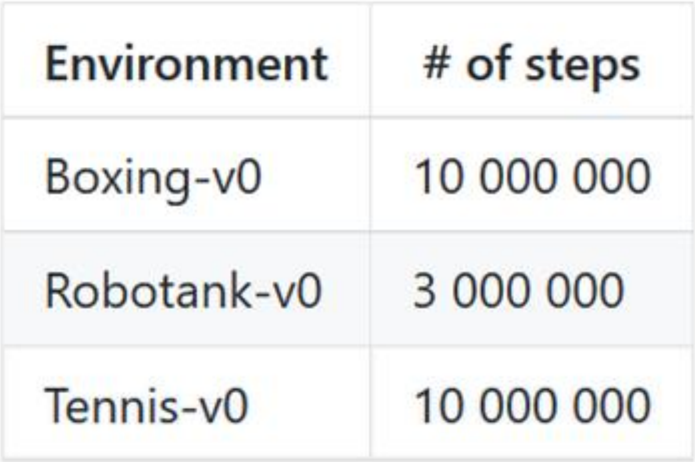
Description of experiment

We are using OpenAI Gym environments in this project. We provide trained models for the following environments: Atari Boxing, Atari Robotank, Atari Tennis.

All of these environments provides an RGB image of the screen as an observation on each step. The input image is an array of shape (210, 160, 3).

We used DQN algorithm implementation in Keras-RL package. Hyperparameters are described in Mnih et al. (2015).

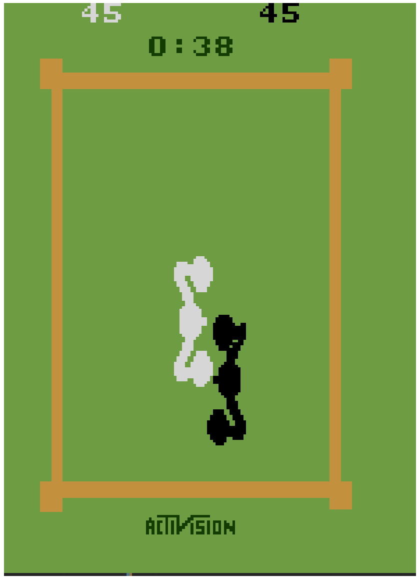
We trained four Open AI Gym Atari environments with the same hyperparameters and the same model architecture.



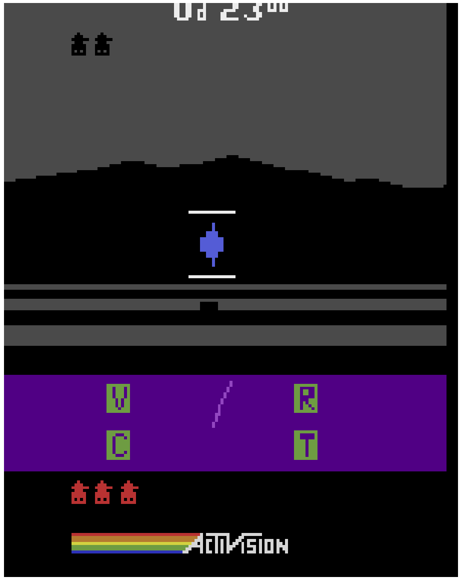
Environment	# of steps
Boxing-v0	10 000 000
Robotank-v0	3 000 000
Tennis-v0	10 000 000

Results

Boxing-v0
(average reward value: 23.7)



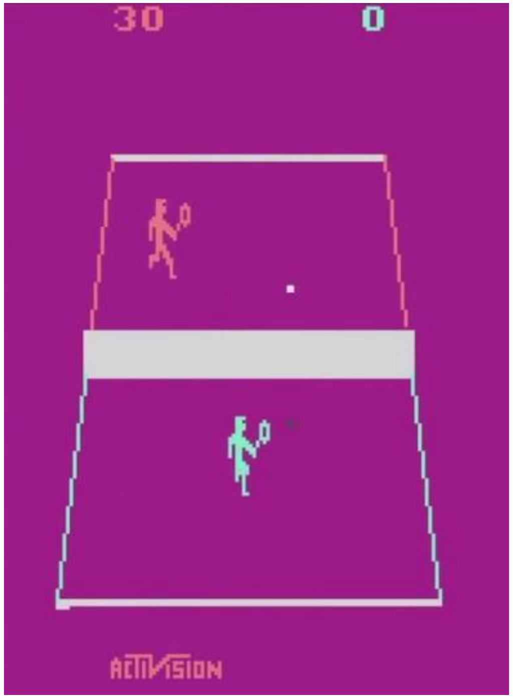
Robotank-v0
(average reward value: 8.4)



Results

Tennis-v0

The trained agent in Tennis-v0 environment, unfortunately, decided not to play tennis at all. So that it can avoid the negative reward. Proposed solution: to train agent on 50 million steps.



Further work

To get better results the number of iterations in training need to be increased to at least 50 million steps. Furthermore, some environments have best results with the agents, that were trained on 80 million steps.

Bibliography

1. *Sutton, R. & Barto, A.* Reinforcement Learning: An Introduction (MIT Press, 1998).
2. *Lugoshi, G. & Cesa-Bianchi, N*. Prediction, learning and games. New York: Cambridge University Press, 2006.
3. *Watkins, C. J. & Dayan, P.* Q-learning. Mach. Learn. 8, 279–292 (1992)
4. *Mnih, V., Kavukcuoglu, K., Silver, D.,* Human-level control through deep reinforcement learning. Nature (2015)