# Learning to Play Pong Video Game via Deep Reinforcement Learning

Ilya Makarov [1] (0000-0002-3308-8825), Andrej Kashin [1], and Alisa Korinevskaya [1]

National Research University Higher School of Economics,
School of Data Analysis and Artificial Intelligence,
3 Kochnovskiy Proezd, 125319 Moscow, Russia
iamakarov@hse.ru, kashin.andrej@gmail.com, korinalice@gmail.com

**Abstract.** We consider deep reinforcement learning algorithms for playing a game based on the video input. We discuss choosing proper hyperparameters in the deep Q-network model and compare with the model-free episodic control focused on reusing of successful strategies. The evaluation was made based on the Pong video game implemented in Unreal Engine 4.

**Keywords:** Deep Reinforcement Learning, Deep Q-Networks, Q-Learning, Episodic Control, Pong Video Game, Unreal Engine 4

## 1 Introduction

Reinforcement learning (RL) is a field of machine learning, that is dedicated to agents acting in the environment in order to maximize some cumulative reward. Actions of an agent are rewarded by the environment to reinforce correct behaviour of the agent. In this case the agent is able to automatically learn the optimal strategy. Methods of reinforcement learning appeared to be useful in many areas where AI is involved: robotics [1], industrial manufacturing [2], and video games [3]. RL usually solves sequential decision making problems [3]. We follow success of [4] experiment in order to test applicability of episodic control RL algorithm in Pong video game.

### 1.1 Reinforcement Learning Basics

An RL agent interacts with an environment over time. At each time step $t$ the agent is situated in a state $s_t$. The agent selects an action $a_t$ from some action space $\mathcal{A}$, following a policy $\pi(a_t|s_t)$. This policy is a probability distribution describing the agent behavior, i.e., a mapping from state $s_t$ to actions $a_t$. Then agent receives a scalar reward $r_t$ from the environment, and transitions to the next state $s_{t+1}$ according to the environment dynamics or model: reward function $R(s, a)$ and state transition probability $P(s_{t+1}|s_t, a_t)$ respectively. In an episodic RL problem [5], this process continues until the agent reaches a terminal

state and then restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the discounted, accumulated reward with the discount factor $\gamma \in (0, 1]$. The agent aims to maximize the expectation of such long term return from each state.

To determine agent preference over states, a value function is introduced as a prediction of the expected accumulated and discounted future reward. The action-value function $Q^{\pi}(s, a) = E[R_t | s_t = s, a_t = a]$ is the expected return for selecting action $a$ in state $s$ and following policy $\pi$ afterwards. An optimal action value function $Q^*(s, a)$ is the maximum action value achievable by any policy for state $s$ and action $a$. We could define a state value $V^{\pi}(s)$ function and the optimal state value $V^*(s)$ in the same way [4].

Temporal difference (TD) learning is the key idea in RL. It learns a value function $V(s)$ online directly from the experience with TD error, bootstrapping in a model-free and fully incremental way.

In TD learning the update rule has the following form:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[ r_t + \gamma V(s_{t+1}) - V(s_t) \right] \tag{1}$$

where $\alpha$ is a learning rate, and $r_t + \gamma V(s_{t+1}) - V(s_t)$ is called TD error.

Similarly, a Q-learning agent learns action-value function with the update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2}$$

In contrast to Q-learning which is off-policy approach, SARSA is an on-policy control method, with the update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3}$$

SARSA refines the policy greedily with respect to action values. TD-learning, Q-learning and SARSA converge under certain conditions. From optimal action-value function one can derive an optimal policy for RL agent.

### 1.2 Episodic Control

While Q-networks improves performance after gradient-based optimization procedures, model-free episodic control (EC) [5] is focused on reusing of successful strategies. Although the model-free episodic control algorithm is seemingly closer to the human learning, in the real world we rarely encounter exactly the same situation over and over again. But in games with the finite number of states, episodic control could run correctly.

In order to remember most successful strategy, EC agent makes use of additional structure – $Q^{EC}(s, a)$ table where the best action-values for each state are kept. At each time-step in the particular state of the environment the agent peeks the action with the maximal value of $Q^{EC}(s, a)$. At the end of each episode the value is updated in the following way:

$$Q^{EC}(s_t, a_t) \leftarrow \begin{cases} R_t & if (s_t, a_t) \notin Q^{EC} \\ max \left\{ Q^{EC}(s_t, a_t), R_t \right\} & \text{otherwise} \end{cases} \tag{4}$$

To carry out maximization's of $Q^{EC}(s, a)$ over all states, it is regarded as a nearest neighborhood model. Then of particular interest is the mapping $\phi$ of observations onto states as observations $o_t$ tend to have high dimensionality than drastically slows down KNN search.

## 2 Pong

To illustrate application of RL methods, we implemented them in the Pong Game environment [6] designed in Unreal Engine 4. Unreal Engine 4 (UE4, [7]) is a powerful suite of integrated tools for game developers to design and build games, simulations, and visualizations. Since its release in 1998 Unreal Engine has became a staple among the community.

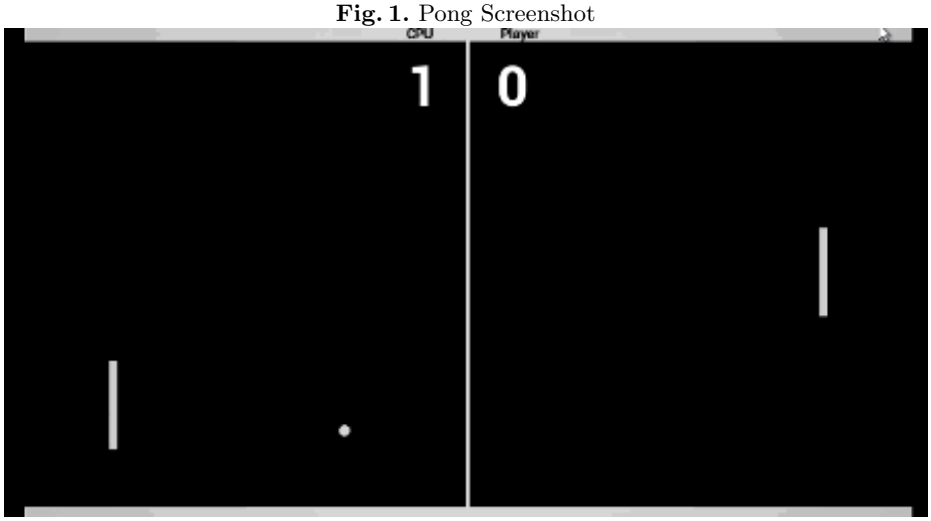
The main goal of this work is to prove that it is feasible to apply methods of RL inside UE4. We show that it is indeed possible to efficiently train and apply model built with the modern machine learning framework (in our case TensorFlow [8]) in UE4.

We achieve this by patching a plugin to support Python scripting inside UE4, implementing C++ module for capturing game screenshots, and creating TensorFlow-based Python controller for the player paddle.

### 2.1 Environment

We use physics based UE4 implementation of classical Pong environment.

**Fig. 1.** Pong Screenshot




This screenshot from the environment shows main elements of the game:

1. **Paddles**. Every player controls a solid rectangle called paddle that can reflect the ball.
2. **Ball**. The ball is a rigid body that moves with a constant speed and reflects from the walls and paddles.
3. **Walls**. The walls are present on the top and on the bottom of the screen.
4. **Goals**. The left and right sides of the screen represent goals. In order to score a point, the player needs to hit the opposite goal with the ball.
5. **Scores**. The top part of the screen shows two numbers that are equal to the number of points each player have scored.

The purpose of the game is to maximize the difference between your score and opponent score, while making it positive during time limit.

The human player uses raw pixels (screenshots, 5 frames per seconds) as an input during play, and outputs three types of actions using the keyboard:

- Key up: move paddle up
- Key down: move paddle down
- Idle: paddle stays at the same place

The game comes with the built-in AI controller that gets high-level features such as position and speed of the ball, position of his and opponent paddle as an input, and uses rule-based approach to control the paddle.

### 2.2 Python Scripting

In Unreal Engine 4, the standard way to implement game logic is through writing C++ modules or using internal visual scripting language called Blueprints. The C++ approach is more powerful and is used to implement core game logic and reusable modules, though in many cases it is overly verbose, and does not allow to quickly iterate on the solution due to slow compilation speeds. On the other hand, blueprints are simple to create and understand, provide faster development cycles, but yet not expressible enough in many cases.

When it comes to the modern machine learning engines, they are usually written in C++, while the provided public API comes in form of Python bindings. Although it is possible to use TensorFlow C++ API, it limits the reuse of openly available RL algorithms implemented in Python, and slows down the research process due to the nature of the C++ language.

Because of all these reasons we looked into alternative languages support for UE4 and identified two candidates: Python and Lua. Both languages are supported through third-party plugins available on GitHub, UETorch [9] for Lua and UnrealEnginePython [10] for Python.

As the authors were more familiar with Python and TensorFlow, it was chosen as the primary language.

The UnrealEnginePython plugin allows users to write Python scripts that interact with UE4 by being able to access and mutate the internal state of the game. This can be used to obtain current position of the ball and paddle, and giving the commands to move the paddle.During this work the original plugin was extended to support Python-based controllers in UE4.

### 2.3 RL Model Implementation

We implement AI controller using TensorFlow machine learning framework [11] to utilize GPU and multi-core CPU resources. The reward of +1 is given to the agent when it scores a point, and -1 when the opponent scores a point. The agent is trained with fixed 32 FPS. The screenshots are binarized and rescaled to resolution 80x80.

We use Deep Q-Network (DQN) [12] learning algorithm to train and control AI agent. Episodic Control was implemented with an embedding function represented as a random projection. It the simplest way to reduce dimensionality while preserving Euclidean distances.

## 3 Results

We have built a Deep RL based controller that outperforms the standard rule-based AI, while using raw pixels as an input. The training takes 6 hours on GPU and it takes 10 million iterations to beat the built-in scripted AI.

However, with regard to Episodic Control it was noticed that Euclidean distance is impractical way to measure frames' similarity. In case of binary images, Euclidean distance scores only sum of different pixels but not their relative proximity. In other words, using Euclidean distance it is impossible to choose frames where a ball is closer to its current position. Later, the authors found a result on too long applicability of episodic control in similar tasks [13]. This problem can be dealt with Variational Auto Encoder (VAE) [14] in the future work for more complex 3D game [15].

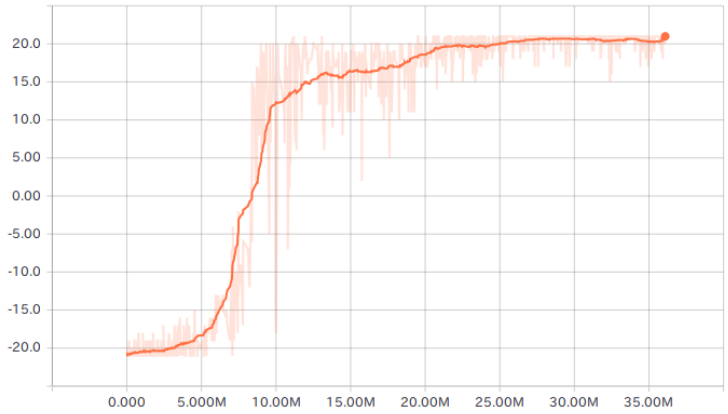



**Fig. 2.** Average score difference over a set of 21 games vs. number of iterations.

The code implementation is available on GitHub ([16], [17]).

## Acknowledgments

The work was supported by the Russian Science Foundation under grant 17-11-01294 and performed at National Research University Higher School of Economics, Russia.

## References

1. Smart, W.D., Kaelbling, L.P.: Effective reinforcement learning for mobile robots. In: Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on. Volume 4., IEEE (2002) 3404–3410
2. Mahadevan, S., Theocharous, G.: Optimizing production manufacturing using reinforcement learning. In: FLAIRS Conference. (1998) 372–377
3. Cole, N., Louis, S.J., Miles, C.: Using a genetic algorithm to tune first-person shooter bots. In: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753). Volume 1. (June 2004) 139–145 Vol.1
4. Mnih et al.: Playing atari with deep reinforcement learning. Review of Scientific Instruments **72**(12) (December 2013) 4477–4479
5. Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J.Z., Rae, J., Wierstra, D., Hassabis, D.: Model-free episodic control. Review of Scientific Instruments (14) (June 2016)
6. Atari: Atari arcade: Pong game [Online: Accessed: 2017-05-14].
7. Games, E.: Unreal engine technology. https://www.unrealengine.com/ [Online: Accessed: 2017-05-14].
8. Google Inc.: Tensorflow. https://www.tensorflow.org/ [Online: Accessed: 2017-05-14].
9. Lerer, A.: Uetorch. https://github.com/facebook/UETorch [Online: Accessed: 2017-05-14].
10. Ioris, R.D.: UnrealEnginePython. https://github.com/20tab/UnrealEnginePython [Online: Accessed: 2017-05-14].
11. Abadi, M., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015) Software available from tensorflow.org.
12. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540) (02 2015) 529–533
13. Pritzel, A., Uria, B., Srinivasan, S., Badia, A.P., Vinyals, O., Hassabis, D., Wierstra, D., Blundell, C.: Neural episodic control. CoRR **abs/1703.01988** (2017)
14. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. Review of Scientific Instruments (1) (May 2014)
15. Makarov, I., et al.: First-person shooter game for virtual reality headset with advanced multi-agent intelligent system. In: Proceedings of the 2016 ACM on Multimedia Conference. MM '16, New York, NY, USA, ACM (2016) 735–736
16. Kashin, A.: Tensorflow + unrealengine + python lab. https://github.com/akashin/HSE_AI_Labs/tree/master/Lab_4 [Online: Accessed: 2017-05-14].
17. Korinevskaya, A.: Episodic control for pong game. https://github.com/koral/episodic-control-pong-game [Online: Accessed: 2017-05-14].