Parallelization of Reinforcement Learning Algorithms for Video Games

Marek Kopel(✉) and Witold Szczurek

Faculty of Computer Science
and Management, Wroclaw University of Science and Technology, wybrzeze
Wyspiańskiego 27, 50-370 Wroclaw, Poland
marek.kopel@pwr.edu.pl

Abstract. This paper explores a new way of parallelization of Reinforcement Learning algorithms - simulation of environments on the GPU. We use the recently proposed framework called CUDA Learning Environment as a basis for our work. To prove the approach's viability, we performed experimentation with two main class of Reinforcement Learning algorithms - value based (Deep-Q-Network) and policy based (Proximal Policy Optimization). Our results validate the approach of using GPU for environment emulation in Reinforcement Learning algorithms and give insight into convergence properties and performance of those algorithms.

Keywords: Reinforcement learning · Video game · Parallel computing · Artificial intelligence

1 Introduction

The main disadvantage of Reinforcement Learning (RL) algorithms are their long training times and much compute required for those algorithms to provide good results. To increase innovation in the field, the researchers need to have faster turnaround times when developing new algorithms. Because most of today's improvements in processing speeds come from parallelization, it is the first consideration when improving algorithm convergence times.

Here we research the influence of parallelization of training environments on two well-known RL algorithms. The first one, from the family of value-based, the Deep Q Network (DQN) algorithm, and the second from policy gradient methods, the Proximal Policy Optimization (PPO) algorithm. We use the CUDA Learning Environment (CuLE) [5] framework that emulates ATARI environments on GPU, which provides a way to simulate many more environments compared to classical CPU emulation engines.

Thanks to that property, we can evaluate classical algorithms' performance in a highly parallel setting, similar to those used in cluster computing. Insight into the performance of those algorithms on such a highly parallel scale, which is at the same time available on most computing machines, should be a valuable tool in future RL research.

N. T. Nguyen et al. (Eds.): ACIIDS 2021, LNAI 12672, pp. 195–207, 2021.
https://doi.org/10.1007/978-3-030-73280-6_16

The rest of the paper is divided as follows:

- Background and related work - provides a background on Reinforcement Learning itself and methods of parallelization already employed in previous research.
- Experiments - Explains the aim of the experiments and describes hyperparameters for algorithms and measures used.
- Results - Describes the result of experiments and provides with the commentary regarding those.
- Conclusions - Concludes the work done and provides future research directions.

2 Background and Related Work

2.1 Reinforcement Learning

RL is a subtype of Machine Learning (ML). It is concerned with the sequential decision making of an actor in an environment. The decision making on behalf of the actor is made by a piece of software called "software agent" or just "agent" for short. It is often viewed as an optimization task and formalized as a Markov Decision Process (MDP) [16]. The agent is trying to maximize a reward signal that comes from the environment. It is often better for an agent to take actions that result in immediate rewards over actions that result in rewards that are further in time. This notion of an agent caring more about the present than the future is represented by a total discounted return (1). It is the sum of all rewards from time step t onward, discounted by a discount factor γ:

$$G_t = \sum_{k=0}^{\infty} \gamma^k * R_{t+k} \tag{1}$$

Total return G_t is used to estimate how good is some state in MDP. Function that calculates the value of some state is called value function (2). It is the base for a family of reinforcement learning algorithms called value-based methods.

$$v(s) = E[G_t|S_t = s] \tag{2}$$

Variation of the value function is an action-value function (3). It is an estimate not only for a given state in MDP but also for all actions chosen from this state. The action-value function is often more useful than the solely value function because it directly ties action to states. Q value optimization is basis for DQN algorithm [8] used in experiments in this paper.

$$q(a, s) = E[G_t|S_t = s, A_t = a] \tag{3}$$

Some algorithms instead of using value or action-value function, use so-called policy (4). It is essentially a mapping from states to actions without a notion of a given state's value. Algorithms optimizing policy functions are called policy gradient methods.

$$\pi(a|s) = Pr(A_t = a|S_t = s) \tag{4}$$

2.2 Methods for Solving RL Problems

There are two main classes of methods for solving RL problems: value-based and policy gradients. Value-based strategies optimize value function and implicitly build policy for acting in the environment based on optimal value function. Policy gradients methods optimize policy function directly. The hybrid approaches that utilize both of those functions are called Actor-Critic (AC) methods.

The first notable advancements in value-based RL were achieved with DQN algorithm. Since then, many variations of this algorithm have been developed to improve its sample efficiency. Double DQN [8], Dueling networks [17], Prioritized Replay [13], n-step learning [9], and Noisy Nets [7].

Double DQN overcomes the existing limitations of classical DQN, particularly the overestimation of the action-value function. Prioritized experience replay, instead of sampling data from replay buffer uniformly uses, prioritization to replay only essential transitions. Dueling Network is a type of neural network architecture proposed for the estimation of action-value function. This approach uses classical value function and advantage function to achieve better performance.

In n-step learning approach from classical RL for deep backups is combined with function approximation techniques used in DQN. In Noisy Nets, random parametric noise is added to neural network weights. It provides significantly better exploration than classical epsilon greedy techniques. State-of-the-art algorithmic improvements have been achieved by combining all of those techniques into one meta-algorithm called RAINBOW [10].

Algorithmic improvements to policy gradient methods included Trust Region Policy Optimization (TRPO) [14] and PPO [15]. Those methods use advantage estimates and, therefore, are considered AC methods. Another AC method, without special gradients as in TRPO or PPO, is Advantage Actor-Critic (A2C) [11], and it is asynchronous variation Asynchronous Advantage Actor-Critic (A3C). Recently published is a policy gradient method called Agent57 [2], which achieved super-human performance on all 57 ATARI games, which have not been done before.

2.3 Parallelization of RL Methods

Since RL methods' main disadvantage is long training times, it is essential to use available computing efficiently. Most of today's processing platforms uses parallel processing for scaling of computationally demanding workloads. Massively parallel architectures have been used to speed-up value-based algorithms. Gorilla [12] architecture uses multiple CPUs to simulate similar agents that collect data from episodes and then update a single server with neural network parameters.

In the domain of policy gradient algorithms: GA3C [1] is a GPU accelerated version of A3C. It uses multiple agents that are run on the CPU to collect data. GPU is used to train neural networks based on data gathered by agents. Data gathered by agents are stored in queues to help keep GPU busy. Including queues makes this algorithm offline and may lead to destabilization in learning, as GPU

optimizes neural networks in the direction of "old" data gathered by agents. Synchronized learning without queues had been implemented in PAAC [4]. It stabilizes learning at the cost of GPU underutilization.

As in DQN, massively parallel methods had been employed to achieve the speed-up of those algorithms. One such system is called IMPALA [6]. It uses many CPUs to accelerate the A2C algorithm. As part of system-level efforts, the work regarding the use of specialized chips called FPGA (Field Programming Gate Arrays) promises improvements in hardware efficiency and power consumption. FA3C [3] is state of the art A3C implemented on FPGA. It provides better performance than powerful GPU platforms such as NVIDIA Tesla P100.

As a framework for experimentation, we use CuLE [5], which enables the emulation of ATARI environments on GPU. Such emulation pattern is different from typical emulation engines, which uses CPU, and allows simulation of much more environments, which was crucial for experimentation done in this research. Such a workload assignment enables us to simulate much more environments and reduce communication between CPU and GPU. Because both simulation and training is done on GPU, this provides better GPU utilization.

3 Experiment

The experiments aimed to validate the new approach for parallelization and compare convergence properties of classical RL algorithms. To do this, we measured training times and corresponding frames per second for different numbers of parallel environments, which will be denoted n_e. Additionally, we compared GPU processors and memory utilization for CPU-GPU and GPU only training to compare both approaches' benefits.

3.1 Environment and Algorithms Parameters

Hyperparameters necessary for training are provided to the program using an initialization file. For the DQN, Adam optimizer is used instead of RMSProp as in [10], because it can help reduce algorithm sensitivity to the learning rate. The learning rate for DQN was chosen to be 0.0000625 and for the Adam optimizer 0,0015. Batch size across all runs of the algorithm was set to be 32 transitions. Due to memory constraints on the GPU device, memory capacity was set to be 50000 transitions. Experiments involving the PPO algorithm also use Adam optimizer, but the learning rate is set to 0.00001, while the PPO learning rate is 0.00025. Both algorithms limit the maximum length of episodes, which is set to 18000 transitions. Hyperparameters explained above and additional ones, along with system specifications, are included in appendices.

3.2 Measures

To evaluate tested algorithms' performance, we use the games' score, which for ATARI pong is a scalar value between −21 and 21. The comparison between algorithms' running times was made by finding the convergence curve's first step when the algorithm scored 19 points or higher. Such comparison promotes algorithm runs that converge faster. As a throughput measure during algorithms' training, we use frames per second (FPS) generated by the CuLE engine. To measure GPU's utilization and memory consumption, we use a program called "nvidia-smi," which was run as a daemon during training.

4 Results

4.1 Influence of Parallelization on Training Times

Figure 1 shows convergence curves for PPO. There are mean rewards gathered from the evaluation stage on the vertical axis, as described in the previous section. The horizontal axis shows wall time. For clarity's sake, time is represented on a logarithmic scale. Optimal performance is achieved with the n_e equal to 256. When a smaller number is used, the training times are significantly longer. When a bigger number is used, the training times are slightly higher than with 256 parallel environments.

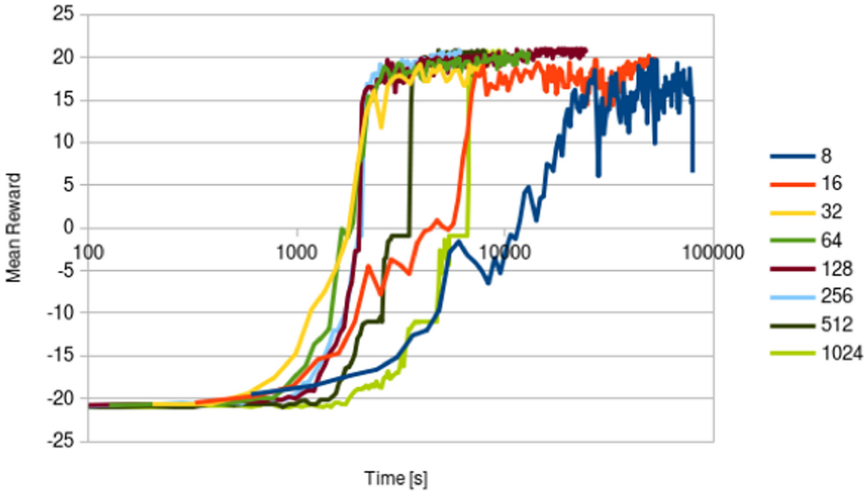

Fig. 1. PPO convergence curves for different n_e.

The same experiments were performed for the DQN algorithm. Convergence curves are shown in Fig. 2. Surprisingly training times increase with a higher n_e. The algorithm learning stops completely with the n_e equal to 512 and 1024.

This effect can be associated with the quality of data that is being gathered into a replay buffer. Each agent can write only a small number of frames that do not capture enough information for an algorithm to learn. This property shows that DQN is more susceptible to the n_e used in training and may not be as suitable for parallelization as policy gradient techniques. It may still be useful to run the DQN algorithm with a moderately high n_e because it can speed up the training time. Nevertheless, a large n_e impacts training negatively. For the DQN algorithm, it may be useful to have a variable n_e to provide a run-time controlled trade-off between convergence speed and an agent's performance.

In Fig. 3, aggregated results are shown for both PPO and DQN. The horizontal axis represents the time it took for an agent to score 19 or higher. The DQN algorithm takes much more time to converge to optimal policy than PPO. Additionally, DQN training times increase with the n_e, and the learning stops entirely with the n_e higher than 256 (those measurements are not shown in the figure). The PPO's best performance can be achieved with the number of agents equal to 256, as was stated earlier. For the number of parallel agents higher than 512, no significant speed-up can be achieved, similarly, for the smaller n_e.

Bad performance of DQN can be associated with throughput oriented nature of CuLE, as stated in [5]. The time-domain of each parallel environment is being explored less efficiently, which leads to no further improvements in convergence. Such behavior is that when more agents are playing in environments, they can gather only short episodes of play to a replay buffer. Those short episodes do not contribute enough information for a neural network to learn. Policy gradient algorithms seems more stable with a varying n_e than value-based algorithms. It may be because the policy is less susceptible to noise and perturbations, especially in the Proximal Policy Optimization algorithm, which takes special care for significant policy updates. Those experiments show that using many parallel

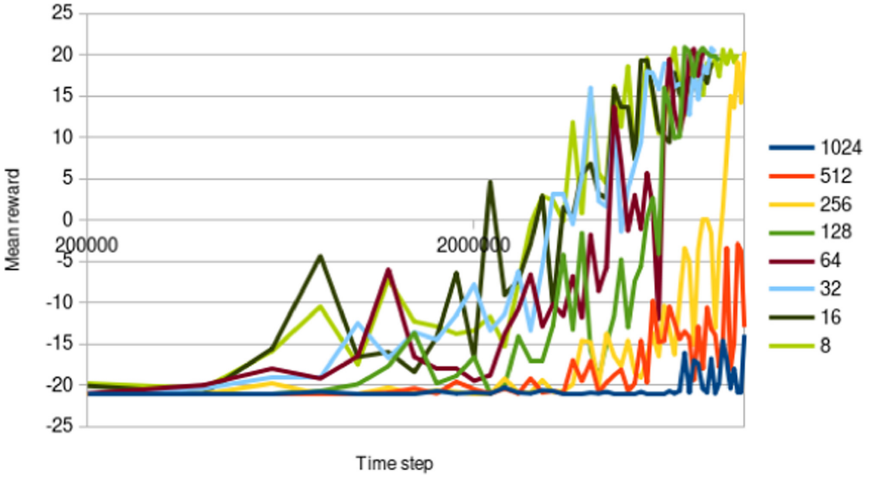

Fig. 2. DQN convergence curves for different n_e.

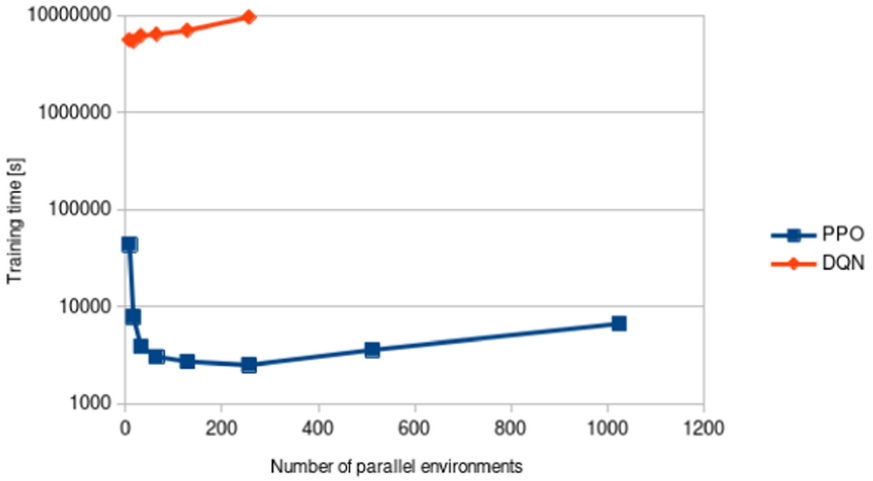

Fig. 3. PPO and DQN training times as a function of n_e (lower is better). Note that values of DQN that are not shown resulted in no convergence.

environments can significantly speed up the training times of RL algorithms. Performing both simulation and training on GPU enables simulation of much higher parallel games at once. Thanks to this, researchers can benefit from faster workflows when developing new algorithms for RL. Additionally, experiments show that the PPO algorithm can benefit more from parallelization than the DQN, which may be right for those algorithms' general family (value-based and policy gradients) though some further experimentation is needed to prove that.

4.2 Throughput Measurements

Here we compare throughput generated by the CuLE framework for DQN and PPO as a function of the number of training environments. The generated FPS differs between algorithms because, in DQN, parallel environments correspond to only the data acquisition stage, while in PPO, parallel agents are also responsible for advantage estimate computation. Though CuLE is capable of much higher throughput, it does not directly correlate with better convergence of algorithms. For example, the DQN algorithm performed worse with the increasing n_e, even though it can achieve higher throughput than PPO. Simultaneously, increasing the number of parallel agents in PPO corresponds to faster convergence times, though this speed up is not linear, and the throughput is worse than in the DQN case (Fig. 4).

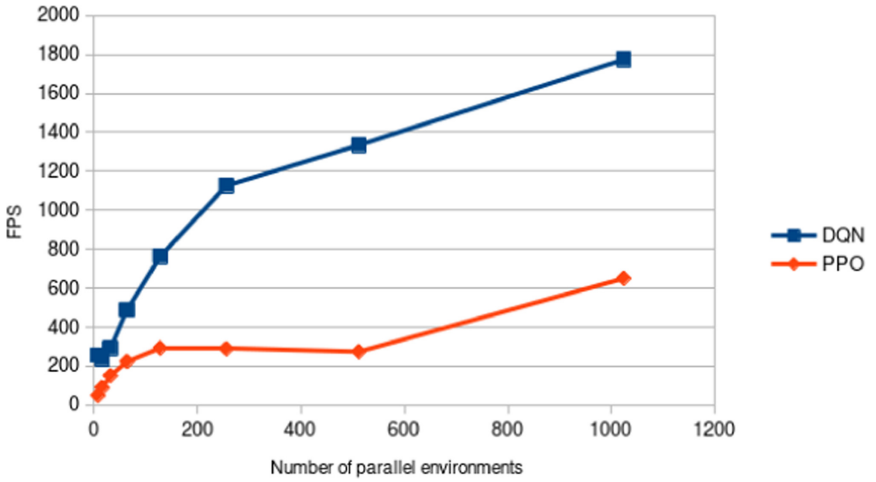

Fig. 4. PPO and DQN FPS for varying n_e (higher means better throughout).

4.3 System Performance Measurements

In this section, two parallelization approaches were tested (CPU-GPU vs. GPU only) to check which of them has better resource utilization. In the CPU-GPU approach, environments are simulated on processors, while learning is done on GPU. The second approach performs both steps on the GPU. This experiment does not require a comparison between DQN and PPO algorithms because CPU-GPU approaches and GPU-only approach should be independent of the algorithm being used. As such, for this experiment, the PPO algorithm was chosen,

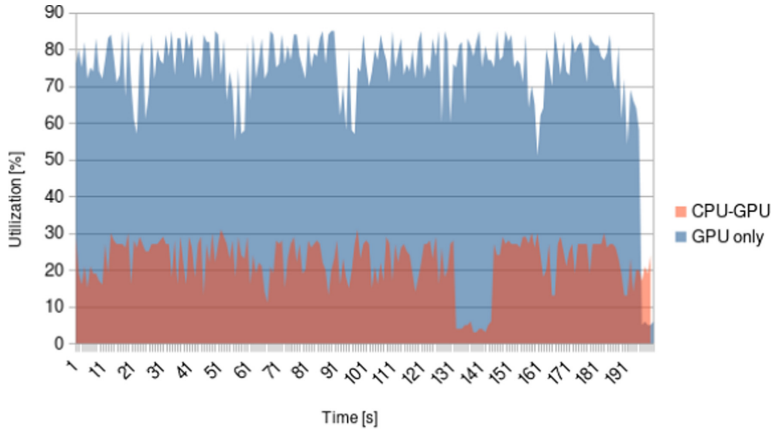

Fig. 5. Comparison of PPO GPU processors utilization in CPU-GPU and GPU only version.

with a constant n_e, namely 32. Only several iterations of running algorithms were measured because the system resources footprint should remain similar throughout the training process.

As a metric of resources, utilization GPU memory and processor utilization were measured. Figure 5 compares processors utilization for CPU-GPU and GPU only. As can be seen, GPU only approach outperforms CPU-GPU. Better compute utilization is achieved because this approach can keep GPU busy by switching between environment simulation and learning steps. A higher peaks of utilization seen in Fig. 5 corresponds to subsequent training results. Lower utilization intervals mark the end of each iteration of training. In the GPU-only approach, the system manages to perform several iterations during the measurement period. The end of iterations can be seen around 21, 61, 111, 141, 161 s of measurement. At the same time, the CPU-GPU approach manages to perform only the first iteration and starts second. The end of the first iteration is marked with a sudden drop in GPU utilization around 141 s of measurement. During this time, CPU emulated environments are being reset, and GPU remains almost idle. This experiment shows clearly that the GPU-only approach provides better system utilization and, thus, higher throughput. The GPU-only approach can perform around four iterations of training. At the same time, the CPU-GPU approach can only do one iteration of training.

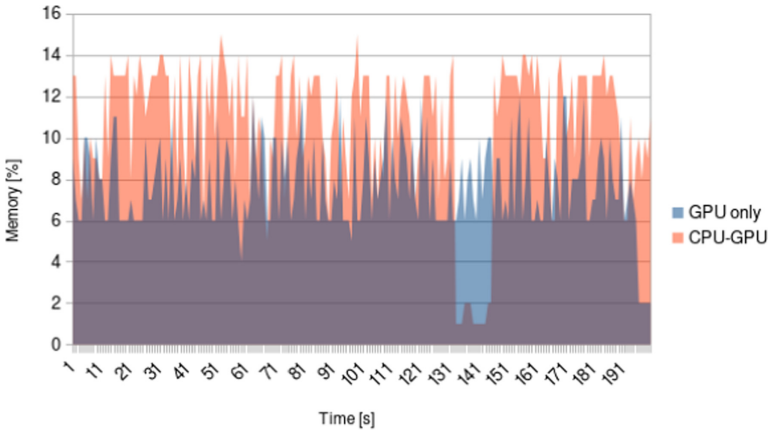

Fig. 6. Comparison of PPO GPU memory utilization in CPU-GPU and GPU only version.

In Fig. 6, the memory usage of both approaches is being compared. Surprisingly, the CPU-GPU approach uses more memory than the GPU only approach. Such a result may be because of memory copies that CPUs need to perform to transfer data to GPU. In GPU only approach, there is no need to make those copies, and thus less memory is used. Interestingly, in the CPU-GPU approach, an end of iteration can be seen on memory measurements. It can be marked at

around the same location as in utilization measurements, namely around 141 s of the experiment. The sudden drop in memory utilization comes from the fact that the algorithm resets environments emulated on the CPU, and during this time, GPU remains almost idle. Such precise iterations cannot be seen in GPU-only measurements. Presumably, even when iteration ends and environments are reset, GPU memory is still being utilized in the same percentage, but for a different purpose.

5 Conclusions

In this work, we researched the influence of parallelization on RL algorithms on the ATARI domain. The agents were tasked with learning to play ATARI games from raw visual input. As a base for our experimentation, we have chosen a recently proposed CuLE framework, enabling the emulation of ATARI environments on GPU. We tested two algorithms from the branch of model-free reinforcement learning methods, namely Deep Q-Networks and Proximal Policy Optimization. Those algorithms represent two categories of methods for solving RL problems: value-based and policy gradients.

Using GPU for simulation and training proved to be a viable strategy for reducing RL algorithms' training times on the ATARI domain. PPO algorithm was more stable with an increasing n_e than value-based DQN. For this part, we also measured FPS generated by the system during training. We showed that even though the DQN algorithm can provide better throughput, it does not correlate with better training times.

System performance measurement during the training of the PPO algorithm with CPU-GPU and GPU only was carried out. The GPU only approach benefited from better processor utilization than CPU-GPU and less memory consumption.

The future research direction for this work would be to perform more experimentation with a more diverse set of algorithms and different ATARI games. It could give more insight into this new method of parallelization - simulation of environments on GPU. Additionally, using a more diverse set of algorithms, one can test properties of the main methods used in Reinforcement Learning - value based methods and policy gradients. On the other hand, testing a more diverse set of games can provide insights into the new parallelization approach's capabilities. It may be so that more extended and more complex games suffer more from such an approach because of GPU thread divergence, which was suggested in the original work regarding the CuLE framework.

Acknowledgements. We want to thank the authors of CuLE for making their work freely available, which helped us do our research.

Appendicies

See Tables 1, 2 and 3.

A Hyperparameters

Table 1. PPO hyperparameters

Hyperparameter	Value
alpha	0.99
Clip ϵ	0.1
Adam ϵ	0.00001
Discount factor	0.99
Learning rate	0.00025
Max episode length	18000

Table 2. DQN hyperparameters

Hyperparameter	Value
Adam ϵ	0,0015
Batch size	32
Discount factor	0,99
Hidden layer size	512
History length	4
Learning rate	0,0000625
Max episode length	18000
Memory capacity	5000

B System specification

Table 3. Hardware and software specification

OS	ubuntu 18.04
Libraries	cuda 10.0, cudnn-7, python 3.6.9, pytorch 1.2.0
Graphics card	NVIDIA GeForce GTX 960M
CPU	Intel Skylake i5-6300HQ, 2,3 GHz
RAM	16 GB DDR3, 1600 MHz

References

1. Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., Kautz, J.: GA3C: GPU-based A3C for Deep Reinforcement Learning. CoRR abs/1611.06256 (Nov 2016)
2. Badia, A.P., et al.: Agent57: outperforming the Atari Human Benchmark. arXiv:2003.13350 [cs, stat], March 2020, http://arxiv.org/abs/2003.13350, arXiv: 2003.13350
3. Cho, H., Oh, P., Park, J., Jung, W., Lee, J.: FA3C: FPGA-accelerated deep reinforcement learning. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, pp. 499–513. Association for Computing Machinery, Providence, RI, USA, April 2019. https://doi.org/10.1145/3297858.3304058, https://doi.org/10.1145/3297858.3304058
4. Clemente, A.V., Castejón, H.N., Chandra, A.: Efficient parallel methods for deep reinforcement learning. arXiv:1705.04862 [cs], May 2017, http://arxiv.org/abs/1705.04862, arXiv: 1705.04862
5. Dalton, S., Frosio, I., Garland, M.: GPU-Accelerated Atari Emulation for Reinforcement Learning. arXiv:1907.08467 [cs, stat] (Jul 2019), http://arxiv.org/abs/1907.08467, arXiv: 1907.08467
6. Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., Kavukcuoglu, K.: IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. arXiv:1802.01561 [cs] (Jun 2018), http://arxiv.org/abs/1802.01561, arXiv: 1802.01561
7. Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al.: Noisy networks for exploration. arXiv preprint arXiv:1706.10295 (2017)
8. van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs], December 2015, http://arxiv.org/abs/1509.06461, arXiv: 1509.06461
9. Hernandez-Garcia, J.F., Sutton, R.S.: Understanding multi-step deep reinforcement learning: a systematic study of the dqn target. arXiv preprint arXiv:1901.07510 (2019)
10. Hessel, M., et al.: Rainbow: combining improvements in deep reinforcement learning. arXiv:1710.02298 [cs], October 2017, http://arxiv.org/abs/1710.02298, arXiv: 1710.02298
11. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. arXiv:1602.01783 [cs], June 2016, http://arxiv.org/abs/1602.01783, arXiv: 1602.01783
12. Nair, A., et al.: Massively parallel methods for deep reinforcement learning. arXiv:1507.04296 [cs] (Jul 2015), http://arxiv.org/abs/1507.04296, arXiv: 1507.04296
13. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized Experience Replay. arXiv:1511.05952 [cs], February 2016, http://arxiv.org/abs/1511.05952, arXiv: 1511.05952
14. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust Region Policy Optimization. arXiv:1502.05477 [cs], April 2017, http://arxiv.org/abs/1502.05477, arXiv: 1502.05477
15. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv:1707.06347 [cs], August 2017, http://arxiv.org/abs/1707.06347, arXiv: 1707.06347

16. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. Adaptive Computation and Machine Learning Series. The MIT Press Cambridge, Massachusetts (2018)
17. Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., de Freitas, N.: Dueling network architectures for deep reinforcement Learning. arXiv:1511.06581 [cs], April 2016, http://arxiv.org/abs/1511.06581, arXiv: 1511.06581