ISSN 2664-9969

UDC 004.67
DOI: 10.15587/2706-5448.2020.218427

Poloziuk K.,
Yaremenko V.

NEURAL NETWORKS AND MONTE-CARLO METHOD USAGE IN MULTI-AGENT SYSTEMS FOR SUDOKU PROBLEM SOLVING

The object of research is multi-agent systems based on Deep Reinforcement Learning algorithms and analysis of ways to establish interaction within the system, based on intelligent agents. Also, part of the material in this paper covers ways to organize the management and administration of agents at the meta-level: external controllers and tools to optimize their work, describing architectural solutions that should accelerate agents' training. The studied full-fledged multi-agent system would be flexible to expansion and would give effective acceleration in agent training and problem-solving quality.

In this paper, the following neural network models were considered: DQN, DDQN, PPO, TD (methods based on Q-Learning), an approach using a neural network with Monte-Carlo tree search. The presented models were tested on a Sudoku problem with a dataset of 5039 combinations, dimensions 2×2, 4×4, and 9×9. Several sets of agent rewards were used. The presentation of data during the learning and problem-solving process was described. Also was built a multi-agent system based on the model using a Monte-Carlo tree search.

According to the study results, it was revealed that for tasks in a complex environment, the models based on Q-Learning are practically ineffective (plots support the statement). The training process for these models is quite demanding on the characteristics of the workstation hardware. It was also determined that the Monte-Carlo tree search method does a good job. Even with a small number of iterations, it shows results better than other Deep Learning methods (45–50 % accuracy for 9×9). However, a significant drawback is a complexity of training the model, and the hardware requirements are too large for this kind of research.

Keywords: *DQN, DDQN, TD, PPO, neural network, deep learning, reinforcement learning, multi-agent system, MCTS, Q-Learning.*

Received date: 27.07.2020
Accepted date: 17.09.2020
Published date: 31.12.2020

1. Introduction

Reinforcement Learning is a promising machine learning sub-field used to solve complex problems with uncertain conditions or environments where the rules of existence are simpler to set by a system of fines and rewards than an explicit description of all the rules [1, 2]. The Reinforcement Learning approach is increasingly combined with Deep Learning and used to create a better combination of intelligent agents [1, 3].

The relevance of the work depends on the development of hardware, which allowed the scientific community to develop and train agents based on Deep Reinforcement Learning quickly, and therefore – to conduct a large number of studies with them, which led to many new studies in this field. Particular attention should be paid to research related to the use of intelligent agents in multi-agent systems.

This object of research is multi-agent systems based on Deep Reinforcement Learning algorithms, analysis of ways to establish interaction within the system, based on intelligent agents. Also, part of the material in this paper covers ways to organize the management and administration of agents at the meta-level: external controllers and tools to optimize their work, describing architectural solutions that should accelerate agents' training. *The aim of this research* is to explore and build a multi-agent system based on Deep Reinforcement Learning to solve problems in a complex environment.

2. Methods of research

Theoretical research consists of prototyping and analysis of the work of different architectures of Deep Neural Networks and different ways of training them to create intelligent agents and the subsequent creation of a multi-agent system based on the most successful approach. The considered architectures and methods include DQN, DDQN, TD, PPO, Neural MCTS [1, 4, 5]. The Sudoku problem's solution was considered with the help of a system of intelligent agents who communicate using a simple non-standardized protocol of communication such as «mailbox».

The study used three different types of environment (Sudoku boards), which were classified by size. Characteristics of environments and the number of representatives of each of them are given in Table 1.

The practical part of the research includes constructing an environment using the Python programming language and OpenAI Gym library and the implementation of various architectures and methods for Deep Learning using the PyTorch framework. The practical part was performed

iteratively, starting with agents' study in the most straightforward environments, increasing complexity if agents provided satisfying results.

Table 1

Environment characteristics

Type	Characteristic	Value
A	Width	2 cells
	Height	2 cells
	Number of instances	3 instances
B	Width	4 cells
	Height	4 cells
	Number of instances	36 instances
C	Width	9 cells
	Height	9 cells
	Number of instances	5000 instances

The study's driving force is the desire to train agents with the minimum number of steps and maximum accuracy to solve a Sudoku, like a human. With the help of deep training with reinforcement to achieve that, agents based on the evaluation system guessed how to solve the problem correctly and learned to think instead of thoughtlessly filling the cells.

The first step was to determine what agents know about the environment and what do not (Table 2).

Based on this, it is possible to determine the range of actions that agents are provided by the environment (Table 3) [6, 7].

Table 2

Restriction for agents in the environment

Agents do not know	Agents know
The line must be filled with numbers from 1 to *n* without repeat	About the existence of rows, columns, squares
The column should be filled with numbers from 1 to *n* without repeat	That cell belongs to the 1 row./1 col./1 sq.
The square should be filled with numbers from 1 to *n* without repeat	That they can put a number in an empty cell
–	That they can move the Sudoku board one step away from the current one

Table 3

Actions available to the agents

Action	Description
Number placement (1–9)	The agent tries to place the number on the field
Move in one of the directions (10–13)	The agent moves in one of the allowed directions (up, down, left, right)
Idle	The agent does not move

The agent in the environment functions as follows: at the initial moment, the agent is created with a starting position, which is chosen randomly. In the future, the agent, having «visual» access to the entire environment, can move through the cells of the environment and, analyzing the state of the environment, can fill the cells in which it is currently standing. The purpose of the agent is to fill the entire board with numbers (Sudoku solution).

The models in this section are subject to extensive modification by definition of the reward system, so below is a complete list of all the reward options considered during the experiments with which the Neural Networks were trained (Table 4).

Table 4

Rewards sets

Reward	Values from a set			
	1	2	3	4
Correct filling of an empty cell	5	5	5	5
Incorrect filling of an empty cell	–3	–3	0	–3
Attempt to fill an filled cell	–2	–2	–2	–2
Movement (up, down, left, right)	0	0	0	0
Idle (stop)	0	–2	–3	–3
Finding an empty cell	1	0	0	0
Sudoku solving	0	100	%	%
Exit the board	–1	–2	–3	–3
Reward for correct answers at the end of the work	0	0	1	1

This list represents the set of sets of awards with which training took place. Certain items in this list need to be explained separately. The sign «%» describes a situation where the award was given based on the percentage of correctly placed cells in Sudoku for the total number of initially empty cells.

The environment was presented in the form of 3 layers. Each layer is a set of columns, rows, or sections of Sudoku. This partitioning is the primary way to separate the system for several agents. The schematic image is presented in Fig. 1.

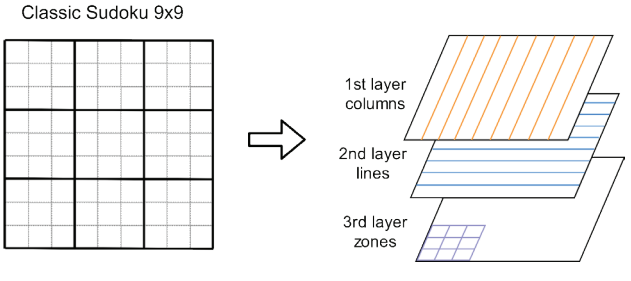

Fig. 1. Schematic partitioning of the Sudoku board

To obtain stable results and provide better understanding of the work, the following are the PC characteristics on which the study data were conducted (Table 5).

Table 5

Workstation characteristics

Characteristics	Description
CPU	Intel Kaby Lake, 4 core, 4.5 GHz
GPU	RTX 2080 TI, 11 GB VRAM, memory frequency 14 GHz, bus width 352 bits, transfer rate 616 GB per second

At the time of this study, it is possible to say that the station is equipped with a powerful GPU to solve machine learning problems, which can be obtained for personal use. CPU is much worse, but it should not significantly affect the learning process, as all computations are on the GPU side.

ISSN 2664-9969

3. Research results and discussion

Fig. 2–7 illustrates the general situation regarding the agent's behavior and its effectiveness when working in a type B environment. During the experiments, as a result of attempts to change the system of evaluation of the agent's actions, it was found that this trend is general and did not depend on the reward method's choice.

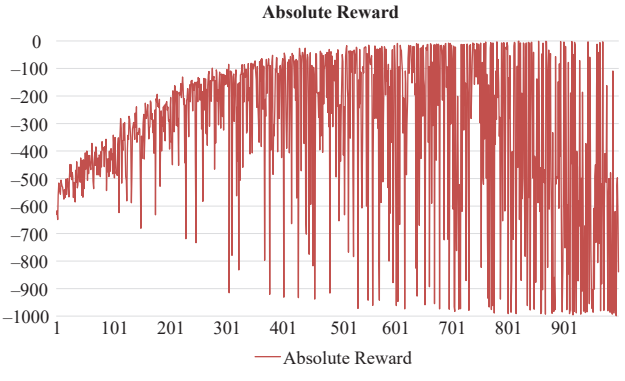

Fig. 2. Absolute rewards for DQN agent (1000 iterations, environment C)

Fig. 3. Average rewards for DQN agent (1000 iterations, environment C)

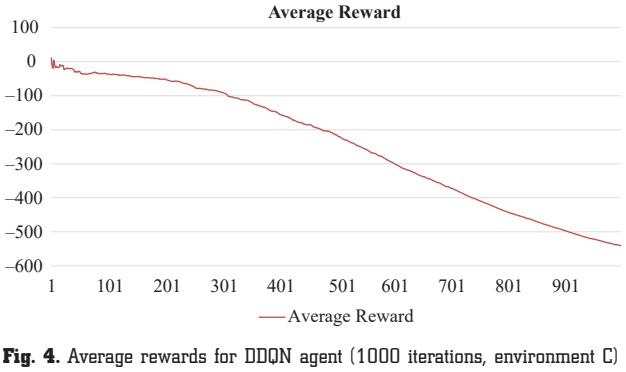

Fig. 4. Average rewards for DDQN agent (1000 iterations, environment C)

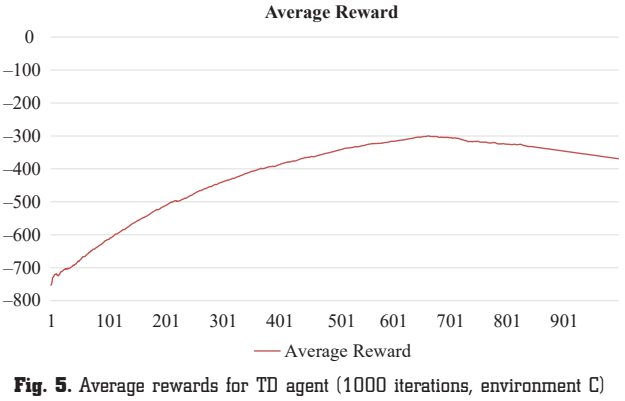

Fig. 5. Average rewards for TD agent (1000 iterations, environment C)

Fig. 6. Average rewards for PPO agent (1000 iterations, environment C)

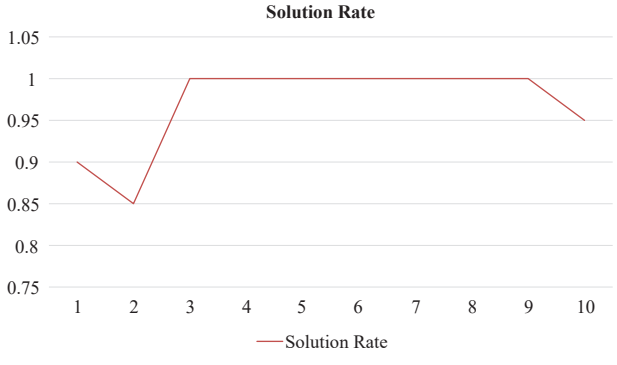

Fig. 7. Solved Sudoku rate by MCTS agent (10 iterations, environment B)

The plot of the absolute reward quite chaotically describes DQN agents' work (as can be seen from Fig. 2), which shows only a tendency to a sharp change in the absolute values of the reward, and this trend is present in all of the plots. On the other hand, the analysis of the average rewards shown in the plot (Fig. 3) shows that the agent studied and reduced the amount of punishment for one iteration. At some point in time, the average reward begins to fall as the absolute reward range increases. Studies were conducted with DQN, DDQN, TD, PPO.

As can be seen from Fig. 6, the plot converged to a plateau in the form of 0. The system has acquired an equilibrium at which the agent does not gain or lose points for its actions. This plot only shows that at best all previous experiments would have converged to such a situation, just in the case of PPO training was quite effective. The agent did not begin to forget the experience and learn to balance. It can be argued that solving this problem even with a single agent based on Q-Learning using Deep Neural Networks, is not possible.

However, the problem must be solved, so it was decided to solve this problem using Neural MCTS (Monte-Carlo Tree Search) and build a multi-agent system based on it with our own communication protocol [8–10]. Due to the lack of computing power and time, a limited number of experiments with type B were performed, as MCTS training for type B environment is already a time-consuming task.

The next experiment was to solve a 9 by 9 Sudoku. The same datasets were used as in the previous paragraphs. However, only 2 full iterations were performed, as the total processing time of one iteration was approximately 25 hours. This experiment can also be considered successful because the accuracy after 1000 episodes was 35 and 46 percent on the respective iterations. But the computing power of the available hardware prevented continuing experiments in this area.

The latest experiment is implementing a 9 by 9 Sudoku system using 27 agents, each responsible for a column, line, or square, and must share information with other agents to succeed in filling cells. The average training time of one of the agents is about 20 hours. The achieving of positive growth occurs from about 3 to 4 iterations and continues. But this experiment can't be completed due to the lack of computing power and time. The average accuracy after the execution of 3 iterations fluctuates at the level of 45–50 percent.

4. Conclusions

Based on the experiment results, it can be concluded that the construction of a multi-agent system based on the Deep Network with reinforcement is possible to solve problems in a complex environment with a large number of states. However, this task is quite tricky for classical methods on the basic Q-Learning: DQN, DDQN, TD, PPO, because:

1. Very complex environment, for example, Sudoku, is based on rather complex implicit rules. Finding these dependencies is too challenging for Q-Learning methods using Deep Networks [11].

2. The limitations of Q-Learning depend on the limitations of modern computers, rather than the method. This study does not refute Bellman's equation and postulates underlying it, just the number of states that the system must consider much larger than the number of records in the learning table that one agent can write.

3. The use of MCTS-based method shows good results but learning such a network is quite a time-consuming task, which increases with the size of the environment and the complexity of the task [4].

It can be argued that the results obtained during the work are quite encouraging. This study can be continued to obtain practical proof of the success of this kind of multi-agent system if the required capacity is available.

References

1. Wang, Y., Wu, F. (2019). *Multi-Agent Deep Reinforcement Learning with Adaptive Policies*. ArXiv, abs/1912.00949. Available at: https://arxiv.org/abs/1912.00949
2. Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J. et. al. (2017). Multiagent cooperation and competition with deep reinforcement learning. *PLOS ONE, 12 (4),* e0172395. doi: http://doi.org/10.1371/journal.pone.0172395
3. Simoes, D., Lau, N., Reis, L. P. (2019). Multi-Agent Deep Reinforcement Learning with Emergent Communication. *2019 International Joint Conference on Neural Networks (IJCNN)*. doi: http://doi.org/10.1109/ijcnn.2019.8852293
4. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G. et. al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529 (7587),* 484–489. doi: http://doi.org/10.1038/nature16961
5. Nguyen, T. T., Nguyen, N. D., Nahavandi, S. (2020). Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications. *IEEE Transactions on Cybernetics, 50 (9),* 3826–3839. doi: http://doi.org/10.1109/tcyb.2020.2977374
6. Kumar, S., Hakkani-Tür, D., Shah, P., Heck, L. (2017). *Federated control with hierarchical multi-agent deep reinforcement learning*. ArXiv. Available at: https://arxiv.org/abs/1712.08266v1
7. Hernandez-Leal, P., Kartal, B., Taylor, M. E. (2019). A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems, 33 (6),* 750–797. doi: http://doi.org/10.1007/s10458-019-09421-1
8. Foerster, J. N., Assael, Y. M., De Freitas, N., Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. Advances in Neural Information Processing Systems. *Neural information processing systems foundation*, 2145–2153.
9. Gupta, J. K., Egorov, M., Kochenderfer, M. (2017). Cooperative Multi-agent Control Using Deep Reinforcement Learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) Vol. 10642 LNAI*. Springer Verlag, 66–83. Available at: http://doi.org/10.1007/978-3-319-71682-4_5
10. Nguyen, N. D., Nguyen, T., Nahavandi, S. (2019). Multi-agent behavioral control system using deep reinforcement learning. *Neurocomputing, 359,* 58–68. doi: http://doi.org/10.1016/j.neucom.2019.05.062
11. Da Silva, F. L., Glatt, R., Costa, A. H. R. (2017). Simultaneously learning and advising in multiagent reinforcement learning. Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS. Vol. 2. *International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS),* 1100–1108

Poloziuk Kateryna, *Department of the System Design, National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Kyiv, Ukraine, e-mail: kate.polozyuk@gmail.com, ORCID: http://orcid.org/0000-0002-9892-5196*

Yaremenko Vadym, *Postgraduate Student, Assistant, Department of the System Design, National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Kyiv, Ukraine, e-mail: yaremenko.v.s@gmail.com, ORCID: http://orcid.org/0000-0001-8557-6938*