UDC 004.89, 004.912

M. Osypenko, V. Shymkovych, P. Kravets, A. Novatsky, L. Shymkovych

INTELLIGENT CONTROL SYSTEM WITH REINFORCEMENT LEARNING FOR SOLVING VIDEO GAME TASKS

Abstract: This paper describes the development of a way to represent the state and build appropriate deep learning models to effectively solve reinforcement learning video game tasks. It has been demonstrated in the Battle City video game environment that careful design of the state functions can produce much better results without changes to the reinforcement learning algorithm, significantly speed up learning, and enable the agent to generalize and solve previously unknown levels. The agent was trained for 200 million epochs. Further training did not improve results. Final results reach 75% win rate in the first level of Battle City. In most of the 25% of games lost, the agent fails because it chooses the wrong path to pursue an enemy that is closer to the base and therefore slower. The reason for this is the limitation of cartographic information. To further improve performance and possibly achieve 100% win rate, it is recommended to find a way to effectively include full information about walls and other map objects. The developed method can be used to improve performance in real applications.

Keywords: reinforcement learning, deep learning, state representation, neural network, Battle City.

Introduction

Reinforcement Learning (RL) is a learning approach in which an artificial intelligence (AI) agent interacts with its surrounding environment by trial-and-error method and learns an optimal behavioural strategy based on the reward signals received from previous interactions [1,2]. Many studies in the RL field focus on the algorithm itself, foregoing state representation choice, while utilizing ones that don’t require much engineering, such as RAM or pixels [3]. While this approach needs less time to implement, it limits the agent’s performance. With each new level or unique situation in an environment, it needs more time to learn and also has to remember previous interactions. It’s highly challenging to generalize on frames to previously unseen ones, so the agent essentially learns from scratch each new part.

Another important point is that most games in the Atari benchmark have no local optimums. For example, in Breakout, as long as the agent does not miss the ball, it has the opportunity to win the level. There is no way to make a play that produces an immediate reward but limits the agent’s ability to successfully finish an episode. And other Atari games are similar in this regard.

© O 0'Qu{r gpmq."X0'Uj {o mqx{ej ."R0'Mtcxgwu."C0'P qxcvum{."N0'Uj {o mqx{ej

To address the above-mentioned problems, we demonstrate our ideas in the Battle City environment. We aim to create state representation, build models, and choose an appropriate reinforcement learning algorithm that would allow the agent not only to get a higher score than some baseline but also to win a significant number of episodes.

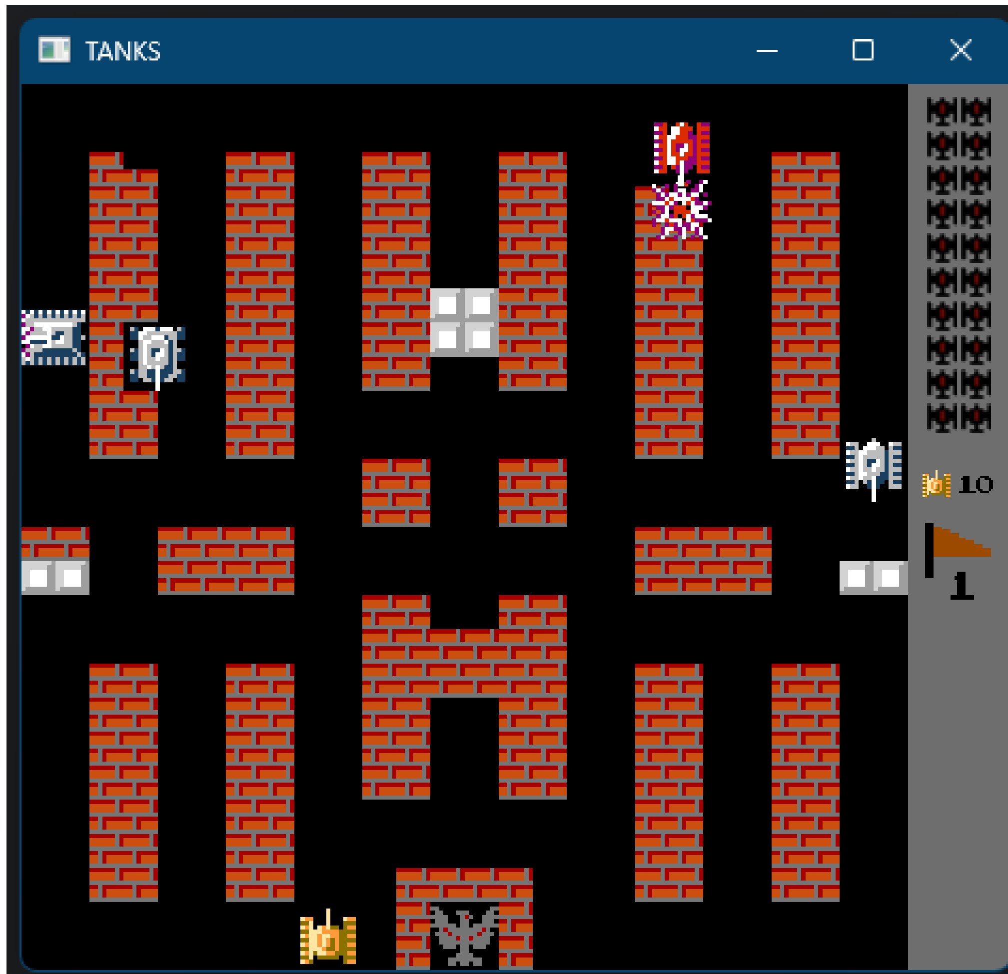

Figure 1. First level of Battle City

To win in the Battle City environment, the agent has to destroy 20 enemy tanks while keeping his base safe. This problem introduces a possible local optimum when the agent only focuses on getting rewards through enemy tanks without base defence.

Another important aspect of the chosen environment is that the agent has a lot of possible strategies that can lead to different long-term states, for instance, being in different parts of the map hundreds of steps later, as opposed to Atari games where decisions usually impact only the next few dozens of frames. Hence, we show the effectiveness of our ideas in this environment.

Related works

There are several works that demonstrate improvements in reinforcement learning methods on the Atari benchmark: A3C [4], PPO [5], SAC [6], etc. They all used pixels as state representation and didn't achieve completion of the games, which is definitely possible for humans and should be pursued.

Two prominent researches that created agents able to compete with top players are Open AI Five [7] and AlphaStar [8]. In both of them, handcrafted state features and neural networks are used. However, state representation and model construction are not the focus of the research, and corresponding decisions are only discussed briefly. Also, these games are rather complex and require clusters of computers to train the mentioned agents; thus, the results cannot be easily replicated by a casual researcher.

There is a recent study that focuses on different state representation for Atari games: Object-Centric Atari [9]. Such representation, however, doesn't include all the needed information, such as mazes in Pacman, which results in the inability of agent to solve certain environments.

Reinforcement learning algorithm

Our agent uses IMPALA (Importance Weighted Actor-Learner Architecture), which provides one of the best utilization of resources on a single machine. Learning is happening on actor-critic architecture with the off-policy V-trace correction method [10]. Generating transitions in the Battle City environment is computationally intensive compared to Atari games, so an off-policy algorithm, which can learn on the same transition multiple times, is preferred over an on-policy one.

State representation

The Battle City game consists of the following elements: map objects, i.e., walls, tanks, bullets, bonuses, base and information about lives and enemy tanks left.

Let's start with the main entities, since decisions should be made mostly depending on them. They include the player's tank, enemy tanks, bullets, base and bonuses. There are two possible ways to represent them. Either make a grid and position them on it or assemble them in a list. The first way, which is not much different from pixels, would introduce the above-mentioned problems, and considering that entities are sparse, we would waste a lot of space in the grid. Hence, we went with the second variant, list. Entities don't have a specific order, so a list is considered unordered. Each entity in a list includes whole information about the entity; more about that is in Table 1.

As for map objects, we could represent them the same way as main entities, but the only non-default fields would be type and position. Also, while there are less than 20 main entities at a time, the wall count can exceed two hundred, so storing them the same way as entities is suboptimal. We considered placing such objects in a grid with only their type, as

coordinates would be accounted for by position in the grid, but this representation proposes some challenges, mainly figuring out the correlation between the grid and the main objects. In respect to that, we decided to reduce information about map. These objects mainly determine how agent should path in the short term, and having knowledge about the presence or absence of a wall far from the agent shall not have many benefits. So, we place objects in a grid, but only consider a small part of it around the agent. This way, the agent won’t be flooded with redundant information but will still have the ability to correctly navigate. The correlation problem between grid and entities doesn’t arise as well since such a subgrid is always centred around the player's tank.

Table 1

Entity information

FIELD	DESCRIPTION
TYPE	Type of the entity. Tank, bullet, base or bonus.
TEAM	Entity team. Ally, enemy or neutral.
DIRECTION	Direction of the entity. Up, left, down, right or none.
POSITION	X and Y coordinates of the entity.
LIVES / ARMOR	Lives of the allied tank or amor for the enemy tank. For other entity types equals 1.
SPEED	Current speed of the entity.
TIME LEFT	Remaining bonus lifetime. For other entity types equals 0.

Finally, there is left general information that is displayed on the side of the game window: the number of enemy tanks and the player's lives. Since we considered lives in tank entity, we have a single number in this section, and we are fine with that, there is no need to change it to any other representation.

Reward function

The goal of the game is to destroy 20 enemy tanks while defending the base. There is an in-game score, but it is inconsistent; the player receives different amounts of points for different tanks and gets a lot of them for bonus collection, while in most cases it does not bring the agent closer to the goal. So, we simply give a reward 1 per enemy tank's destruction. Other reward functions were considered, such as negative reward for losing base or lives, but they either didn’t affect convergence or affected it negatively (the way to make this and consecutive findings is described in the Experiments section).

Action space

At any time, the player can move in any of four directions or stand still. Also, he can shoot or not shoot. Thus, we have actions of two types. If we combined them, we would have a total of 10 actions. But such a method would screw up entropy since agent could

converge on some simple behaviour, such as slowly going ahead and shooting. Entropy regulation should prevent that, but with action combinations, agent can have low entropy while doing so by simply alternating between 4 possible actions. Hence, we reduced action space to only 5 actions: going in 4 directions or shooting. It does limit agent in some way, but with a possible 20 actions per second, this is not an issue.

Model

Using a much richer class of function approximations, such as neural network approximation, where we directly use the states without requiring an explicit function specification [12, 13]. It can represent complicated nonlinear functions using different activation functions. ANN is one of the best choices for nonlinear function approximation, and in the last few years, deep neural networks (DNN) are becoming popular. The DNN is quite successful for problems such as speech recognition, object detection, and Natural Language Processing (NLP) [13-16]. DNN can automatically extract the low-dimensional features from high-dimensional input data like audio and pictures [17-23]. Deep reinforcement learning (DRL) is a powerful method to introduce efficient function approximation and enable RL to solve some of the most complex learning problems, such as playing video games directly from image pixels. Through a trial-and-error approach, DRL can create efficient autonomous RL agents that can learn optimal policies for complex real-world problems. It can also be implemented in works to estimate optimal policies directly from input images from the environment.

IMPALA is based on actor-critic architecture and requires two networks: policy and state-value. We went with the exact structure for both, the only difference being in the output layer, but it's worth noting that the policy network can be made simpler than the state-value one. This is explained by the fact that policy may represent simple behaviour, in our case, something like moving to the nearest tank and shooting when close, while the state-value function needs to output the possible reward of the corresponding policy with regard to the discount factor. In the latter case, much more precision is required.

Network types

There are three parts in our state representation: a list of entities, a small grid, and general information as a number. For the list, we need such a neural network that an identical set of items in different orders would produce the same results. This can be achieved with transformers [23]. Transformers are DNN that utilize a self-attention mechanism to capture contextual relationships within sequential data [24]. Unlike traditional neural networks and variants of Recurrent Neural Networks, such as Long Short-Term Memory, Transformer models excel at managing long dependencies among input sequence elements and facilitate parallel processing. Consequently, Transformer-based models have garnered significant attention from researchers in the field of artificial intelligence. This is due to their

tremendous potential and impressive accomplishments, which extend beyond NLP tasks to encompass various domains [16, 22], including Computer Vision, audio and speech processing. For a small grid, we can use either CNN or just the FC layer in case the subgrid is too small for CNN to convolute anything meaningful. And for general information, we can pass it through a small FC layer or just keep it as is.

After all parts are processed by the according network parts, we concatenate them and feed them into the output FC layers.

Preprocessing

For entity encodings, in addition to the fields mentioned in Table 1, we added distance by coordinates from entity to base, player, and enemy tanks. This change sped up convergence at least twice.

Simultaneously on the map can be no more than 4 enemy tanks, 1 bullet per enemy, and 2 bullets per player. Bonuses are not limited, so we take into account up to 3 of them. Thus, the maximum list size comes out to 16.

The game map is continuous. To create a grid, we divide the map into cells, each the size of one wall part. Then cells around the player are taken. The chosen subgrid size is 11x11, and the and the whole map is 26x26.

In the general information part, we added the coordinates of a player, though we did not observe the significant effect of this change on the convergence.

Architecture overview

The original transformer has sequence input and output [25]. While we need such input, for output, we would want hidden representations to be combined with others. In order to achieve this, we can modify the transformer.

The encoder part is left unchanged. For the decoder, we remove output sequence self-attention since we don’t have an output sequence; instead, we create a special token embedding for the query with an appropriate size. Encoder outputs with query tokens are passed to the multi-head attention layer with normalization and then fed forward in the in the same way as before. Also, we don’t apply softmax activation at the end.

Entity embeddings are obtained as a concatenation of separate embeddings of each categorical feature and non-categorical features.

The complete neural network architecture is presented in Figure 3, with its configuration below.

General information feed forward consists of two layers with 16 and 8 neurons. Entity embedding has 32, 16, and 8 sizes for type, team, and direction features, respectively. The modified transformer has three encoding layers and one decoder layer. Fully connected layers have 1024 neurons. The output size is 256. Grid feed forward has 3 layers with 256, 128, and 64 neurons. Output feed forward has 3 layers with 1024, 256 neurons, and the last

output has size 1 for the state-value network and the same size as the action space for the policy network. All activation functions are ReLU.

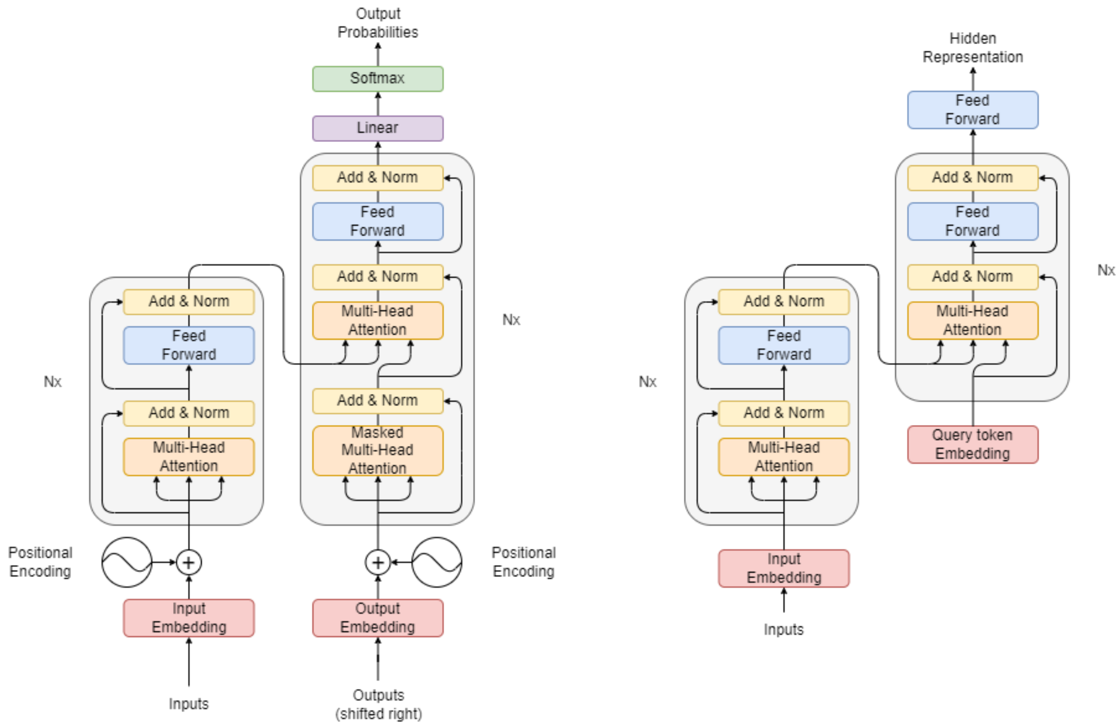

Figure 2. Transformer comparison.
Original is on the left, and modified is on the right

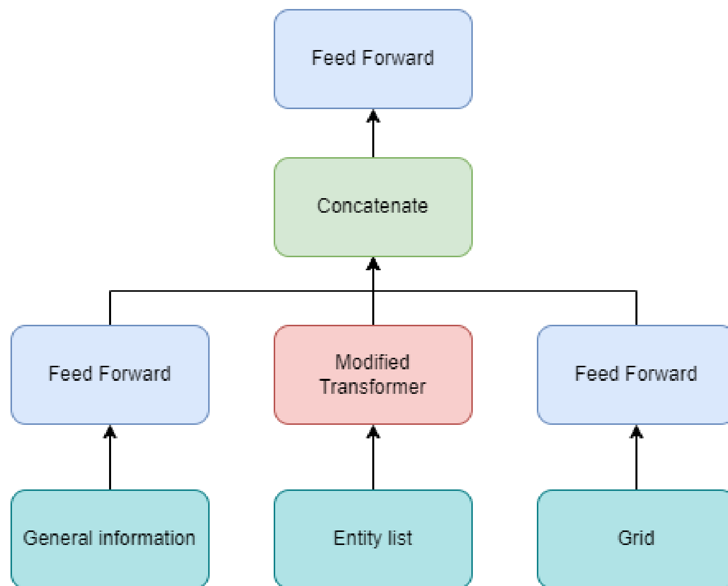

Figure 3. Network architecture

It is worth noting that network with approximately 2 times fewer neurons in each layer can converge as well, while network with 4 times fewer neurons will not.

ISSN 1560-8956

Experiments

Random agent performs poorly in the Battle City environment: he usually destroys his base within the first 100–200 steps while scoring 0 reward. However, with little training, he learns to move slightly forward so that he can’t accidentally break base. In this case, he scores on average 4 points out of 20 and has a 0.1% winrate. Also, such an average is achieved by policy when the agent only fires forward. We take this as a baseline.

The agent can quickly reach this baseline, but improving further takes a significant number of steps. With optimal parameters, around 500k steps are required to reach an average score of 6, while bad parameters can increase this number up to 2m steps. So, our standard training for parameter evaluation takes 3m steps.

On top of that, the training process is highly random. Even with exactly the same parameters, we can observe differences up to 3 points. The best training run had an average of 11 after 3m steps, while the worst run had an average of 8. Hence, for each set of parameters, we made at least a few training runs to get more accurate results. And even so, we can only evaluate significant hyper-parameters since the impact of insignificant ones can’t be distinguished from randomness by the number of runs made.

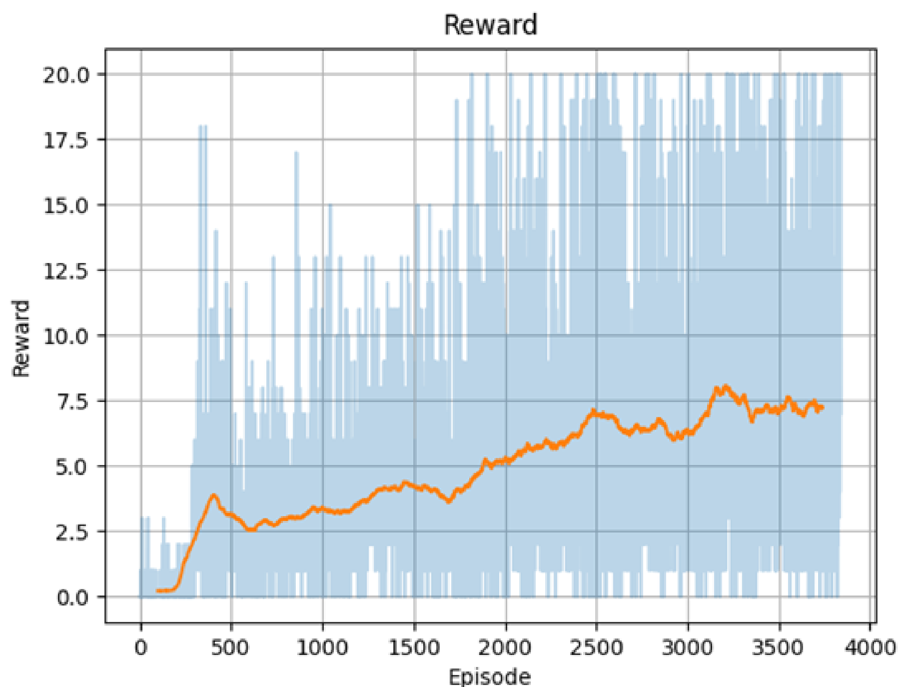

Figure 4. Reward graph after training for 3m steps that resulted in 8 average score

Training

After some time of hyper-parameter searching, we settled on the set presented in Table 2. Most parameter values are default ones or were discovered quickly, but gamma is worth discussing.

Table 2

Parameters

Parameter	Value
Trajectory length	20
Batch size	32
Experience Replay	10,000 trajectories
Gamma (Discount)	0.999
Entropy	0.003
Learning rate	1e-5 with manual decay to 1e-8

In the initial stages, lower gamma values (< 0.998) result in 2–3 times faster convergence. However, the final policy is suboptimal, and the agent reaches only ~45% winrate. With such low gamma, the agent cares only about the next few hundred steps, which is not enough to train to defend the base. Consequently, the agent aims to destroy tanks as fast as possible, and the base is often lost. On the other hand, larger gamma values (> 0.9999) result in much slower convergence, and it’s not clear if policy can converge at all with them.

Results

The agent was trained for 200m steps using the above parameters. Further training didn’t improve performance. The final policy achieves a 75% winrate on the first level of Battle City. This alone is good since Atari agents were not able to win frequently, but at the same time, our agent can generalize well and solve previously unseen levels.

While it was trained only on the first level, the agent reaches 16% winrate on the second level, which has a different map. It is almost five times lower than on the first level, but it is obviously better than our baseline and clearly shows generalization. Also, such a decrease in performance can be explained by the presence of steel walls that impact gameplay, in contrast to the first map, where they were placed on the edges and weren’t interacted with that often.

If we were to train our agent on more maps, each next one would require less time, and eventually the agent would be able to solve any map. This drastically differs from learning on pixels, where the agent just overfits to each observed situation and cannot generalize at all (the inability of the pixel agent to generalize on levels can be seen, for example, in the notebook [25] on the Ms. Pacman game). And it can speed up training by orders of magnitude.

In most of the 25% of lost games, the agent fails because he chooses the incorrect path to chase the enemy, who is approaching base and is slower as a result. This is caused by limited information about the map. To further increase performance and possibly achieve 100% winrate, it’s recommended to find a way to efficiently incorporate full information about walls and other map objects.

 ISSN 1560-8956

Figure 5. Second level of Battle City

Conclusion

We have demonstrated in the Battle City environment the way to represent the state and to build the corresponding model that results in largely increased performance of the reinforcement learning agent and allows it to generalize to previously unseen situations. The agent was trained for 200 million iterations. Further training did not improve the results. Final results reach 75% win rate in the first level of Battle City. In most of the 25% of games lost, the agent fails because he chooses the wrong path to pursue an enemy that is closer to the base and therefore slower. The reason for this is the limitation of map information. To further improve performance and possibly achieve 100% win rate, it is recommended to find a way to effectively include full information about walls and other map objects. Such techniques could then be applied to real-world problems to enhance existing reinforcement learning solutions.

REFERENCES

1. *Shakya, A. K., Pillai, G., & Chakrabarty, S.* (2023). Reinforcement learning algorithms: A brief survey. Expert Systems with Applications, 120495. https://doi.org/10.1016/j.eswa.2023.120495

2. *Kommey, B., Isaac, O. J., Tamakloe, E., & Opoku, D.* (2024). A Reinforcement Learning Review: Past Acts, Present Facts and Future Prospects. IT Journal Research and Development, *8*(2), 120–142. https://doi.org/10.25299/itjrd.2023.13474

3. *Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M.* (2013). Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602. https://doi.org/10.48550/arXiv.1312.5602

4. *Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K.* (2016, June). Asynchronous methods for deep reinforcement learning. International conference on machine learning. pp. 1928-1937. https://doi.org/10.48550/arXiv.1602.01783

5. *Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O.* (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347. https://doi.org/10.48550/arXiv.1707.06347

6. *Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., ... & Levine, S.* (2018). Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905. https://doi.org/10.48550/arXiv.1812.05905

7. *Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., ... & Zhang, S.* (2019). Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680. https://doi.org/10.48550/arXiv.1912.06680

8. *Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... & Silver, D.* (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, *575*(7782), pp. 350-354. https://doi.org/10.1038/s41586-019-1724-z

9. *Delfosse, Q., Blüml, J., Gregori, B., Sztwiertnia, S., & Kersting, K.* (2023). OCAtari: object-centric atari 2600 reinforcement learning environments. arXiv preprint arXiv:2306.08649. https://doi.org/10.48550/arXiv.2306.08649

10. *Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., ... & Kavukcuoglu, K.* (2018, July). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. International conference on machine learning. pp. 1407-1416. https://doi.org/10.48550/arXiv.1802.01561

11. *Sarker, I. H.* (2021). Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. SN Computer Science. Vol. 2(6), 420. https://doi.org/10.1007/s42979-021-00815-1

12. *LeCun, Y., Bengio, Y., & Hinton, G.* (2015). Deep learning. Nature, 521(7553), pp. 436-444. https://doi.org/10.1038/nature14539

13. *Shymkovych, Volodymyr, Anatoliy Doroshenko, Tural Mamedov, and Olena Yatsenko* (2022) Automated Design of an Artificial Neuron for Field-Programmable Gate

 ISSN 1560-8956

Arrays Based on an Algebra-Algorithmic Approach. International Scientific Technical Journal "Problems of Control and Informatics" vol. 67, no. 5, pp. 61-72. https://doi.org/10.34229/2786-6505-2022-5-6

14. *Bezliudnyi Y., Shymkovysh V., Doroshenko A.*(2021) Convolutional neural network model and software for classification of typical pests. Prombles in programming. Vol.4, pp. 95-102. https://doi.org/10.15407/pp2021.04.095

15. *Khurana, D., Koli, A., Khatter, K., & Singh, S.* (2023). Natural language processing: State of the art, current trends and challenges. Multimedia tools and applications. Vol. 82(3), pp. 3713-3744. https://doi.org/10.1007/s11042-022-13428-4

16. *Kravets P., Nevolko P., Shymkovych V., Shymkovych L.* (2020) Synthesis of High-Speed Neuro-Fuzzy-Controllers Based on FPGA. 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT). pp. 291-295. https://doi.org/10.1109/ATIT50783.2020.9349299

17. *Chai, J., Zeng, H., Li, A., & Ngai, E. W.* (2021). Deep learning in computer vision: A critical review of emerging techniques and application scenarios. Machine Learning with Applications. Vol. 6, 100134. https://doi.org/10.1016/j.mlwa.2021.100134

18. *Kravets, P., Novatskyi, A., Shymkovych, V., Rudakova, A., Lebedenko, Y., Rudakova, H.* Neural Network Model for Laboratory Stand Control System Controller with Parallel Mechanisms. In: Hu, Z., Dychka, I., He, M. (eds) Advances in Computer Science for Engineering and Education VI. ICCSEEA 2023. Lecture Notes on Data Engineering and Communications Technologies, Springer, Cham. 2023. Vol 181. pp. 47-58 https://doi.org/10.1007/978-3-031-36118-0_5

19. *Y.S. Hryhorenko, V.M. Shymkovysh, P.I. Kravets, A.O. Novatskyi, L.L. Shymkovysh, A.Yu. Doroshenko.* A convolutional neural network model and software for the classification of the presence of a medical mask on the human face. Problems in programming. 2023. Vol. 2. pp. 59-66. https://doi.org/10.15407/pp2023.02.059

20. *Yu, Y., Si, X., Hu, C., & Zhang, J.* (2019). A review of recurrent neural networks: LSTM cells and network architectures. Neural computation. Vol. 31(7), pp. 1235-1270. https://doi.org/10.1162/neco_a_01199

21. *Bezliudnyi Y., Shymkovych V., Kravets P., Novatsky A., Shymkovych L.* Pro-russian propaganda recognition and analytics system based on text classification model and statistical data processing methods. Адаптивні системи автоматичного управління: міжвідомчий науково-технічний збірник. 2023. № 1 (42), с. 15-31. https://doi.org/10.20535/1560-8956.42.2023.278923

22. *Kobchenko, V. R., Shymkovysh, V. M., Kravets, P. I., Novatskyi, A. O., Shymkovysh, L. L., & Doroshenko, A. Y.* (2024). An intelligent chatbot for evaluating the

emotional colouring of a message and responding accordingly. PROBLEMS IN PROGRAMMING, (1), 23-29.

23. *Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I.* (2017). Attention is all you need. Advances in neural information processing systems, 30. https://doi.org/10.48550/arXiv.1706.03762

24. *Islam, S., Elmekki, H., Elsebai, A., Bentahar, J., Drawel, N., Rjoub, G., & Pedrycz, W.* (2023). A comprehensive survey on applications of transformers for deep learning tasks. Expert Systems with Applications, 122666. https://doi.org/10.1016/j. eswa.2023.122666

25. https://colab.research.google.com/drive/1aSoqbO_wysvciYfDv4hJbL6ZS7lXFnMN

 ISSN 1560-8956