AGENT COMMUNICATION METHOD IN COOPERATIVE ENVIRONMENT BASED ON THE ARTIFICIAL NEURAL NETWORKS

Yaroslav Protsenko, Anton Paramonov

The problem of communication between cooperating agents in multiagent environments is considered in this paper. An algorithm is proposed that is based in reinforcement learning and recurrent neural networks. Main idea behind the algorithm is to use an additional recurrent network that translates information from internal state of one agent to internal state of another agent. Experimental evaluation is performed on model environment. Experimental results have shown that proposed method is potentially useful but requires additional investigation.

В роботі розглядається проблема комунікації коопеpуючих агентів у мультиагентних середовищах. Запропоновано алгоритм на основі підходів навчання з підкріпленням з використанням рекурентних нейронних мереж. Головна ідея алгоритму – це використання додаткової рекурентної мережі, яка виконує обмін інформацією між внутрішніми станами двох агентів під час комунікації. Обраний підхід заснований на застосуванні алгоритму A3C, рекурентної нейронної мережі Long Short-Term Memory (LSTM) для керування агентом та додаткової рекурентної мережі (мережі комунікації). Досліджено два варіанта архітектури нейронної мережі.

За першою версією агенти спочатку «спілкуються», а потім використовується результат у якості додаткових даних про середовище. Друга версія спочатку аналізує дані про середовище, а потім реалізує «спілкування» агентів, обмінюючись високорівневою інформацією. Проведено експериментальну оцінку запропонованого алгоритму на прикладі модельної задачі. Результати експерименту довели, що запропонований підхід покращує ефективність коопеpуючих агентів. Перевагою алгоритму є те, що він не потребує наявності складних та структурованих обчислювальних систем та може бути фізично реалізованим за допомогою дуже маленьких об'єктів, таких, як наприклад макромолекули.

В работе рассматривается проблема коммуникации кооперирующих агентов в мультиагентной среде. Предлагается алгоритм на основе подходов обучения с подкреплением с использованием рекуррентных нейронных сетей. Главная идея алгоритма – использование дополнительной нейронной сети, которая выполняет обмен информацией между внутренними состояниями двух агентов во время коммуникации. Предлагаемый подход основан на применении алгоритма A3C, рекуррентной нейронной сети Long Short-Term Memory (LSTM) для управления агентом и дополнительной рекуррентной сети (сети коммуникации). Исследовано два варианта построения архитектуры нейронной сети.

Первая версия сначала организовывает взаимодействие агентов, а затем использует результат в качестве дополнительных данных о среде. Вторая версия сначала анализирует данные о среде, а потом коммуницирует, обмениваясь высокоуровневой информацией. Проведена экспериментальная оценка предложенного алгоритма на примере модельной задачи. Результаты эксперимента показали, что предложенный алгоритм улучшает эффективность кооперирующих агентов. Преимуществом алгоритма можно считать то, что он не требует наличия сложных и структурированных вычислительных систем и может быть физически реализован с помощью очень маленьких объектов, таких, как например макромолекулы.

This paper deals with the problem of communication between cooperating agents in multi-agent environments with distributed decision making. In the last few years, the methods of creating intelligent agents that combine reinforcement learning and gradient methods have been significantly developed [1, 2], which have been successfully applied to fairly complex environments [3, 4]. A common feature of these achievements is that they use a centralized decision-making system that directly controls agents. However, there are a wide range of applications where centralized management is impractical or technically impossible. Such areas as traffic coordination, communications, trade, defense. Existing solutions to these problems now use explicitly programmed interaction protocols, sometimes in combination with some machine learning methods, to make decisions according to these protocols. This paper explores an alternative approach to building agent cooperation – the search for an effective agent communication protocol and efficient information encoding through reinforcement learning.

The proposed approach is based on the application of the Asynchronous Advantage Actor-Critic (A3C) algorithm [2], the Long Short-Term Memory (LSTM) recurrent neural network [5] for agent management (hereinafter referred to as *main recurrent unit*) and the additional recurrent network - the *communication network*. The communication network is also a separate LSTM unit. Each agent in the system observes a limited area of the environment. It has its own "copy of the network" and the internal state of the network. Information is exchanged when agents are in direct contact with the environment. In this case, the communication network updates the internal state of the agent, using the internal state of another agent, and vice versa. This network can be seen as a "language" that agents create during training. The scheme of agent interaction is shown in Figure 1.

The procedure of the agent's functioning is as follows: 1) reading directly observed information about the environment; 2) and 3) a communication network is activated for each agent (zero or more), which is also a recurrent network that reads the agent's state and processes the other agent's state as an input signal; 4) one iteration of the algorithm of the main recurrent block; 5) performing the action that was calculated by the interaction network in step 4. In this algorithm, agents in the field of view of each other, first exchange information, and then carry out the action in the environment.

The paper investigates the effectiveness of the implementation of two possible versions of this algorithm. The first version uses a communication network to update the internal state of the main LSTM unit during contact with other agents. Then the interaction between the agent and the environment is performed. The data flow diagram of the first version execution data is shown in Figure 2. The represented agents 1..N represent that they are in direct contact with the current agent at time T (ie, they are "visible" in the environment). The second version first reads information from the environment and then "communicates" with other agents. The results of communication with other agents are stored in a separate element of internal memory.

The agent controller is a recurrent network because it needs to store information between time cycles. Because of a variable number of interactions with other agents during one clock cycle of the controller, it is advisable to implement the communication network as a recurrent network. So the agent controller is a nested recurrent network.

The communication network at each agent interaction iteration complements the information stored in the LSTM state vector. The updated state vector is fed to the main recurrent block, which calculates the high-level feature vector at each time cycle. The output layer of the network is the perceptron, which receives the feature vector and generates the action logits vector (in the case of a discrete set of possible actions).

In the second version of the algorithm, the role of the main recurrent unit is reduced to analyzing the environment and providing high-level information about it. Accordingly, high-level analysis is now performed by the communication network. The communication network has its own state, which we call *signal*. During communication, two agents exchange signals and process them with their communication networks. A common feature of both implementations is that the agent is still a nested recurrent network. The potential advantage of such an organization is that there is no need to store information about the environment (low-level) and other agents (high-level) in one storage vector. Figure 3 shows a data flow diagram according to an alternative version of the algorithm.

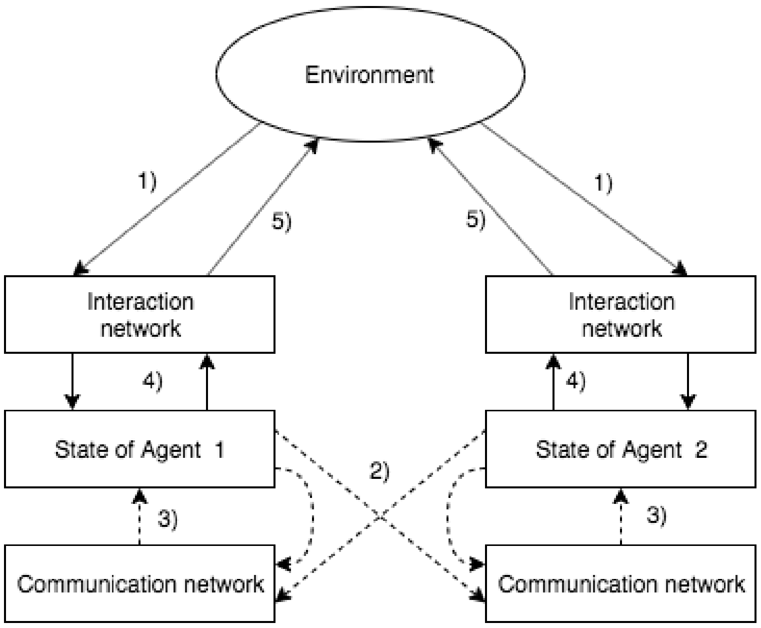


Figure 1. Diagram of interaction of agents based on recurrent neural networks.

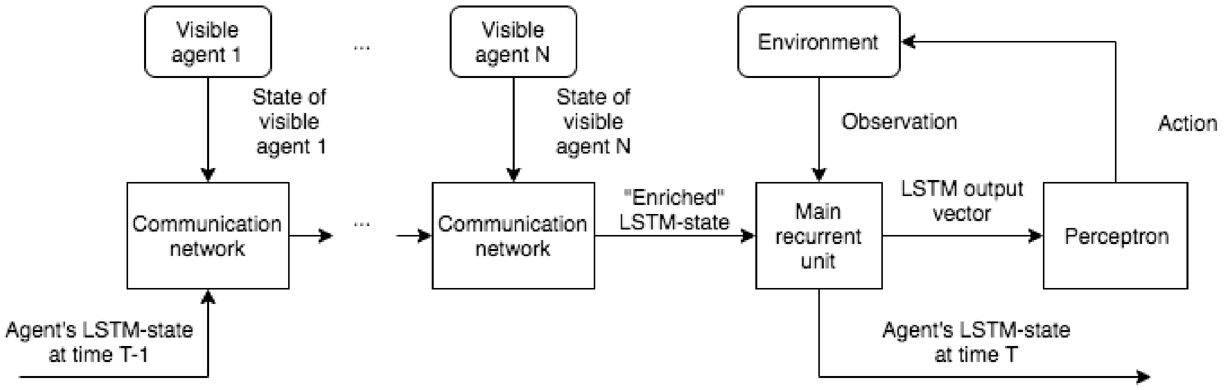


Figure 2. Data flow diagram of the first version of the algorithm.

To train the agent the A3C [2] algorithm is used with one modification. In the original version of the algorithm, the policy loss is defined as:

$$L = -log(\pi(s)) \cdot A(s) - \beta \cdot H(\pi) \tag{1}$$

where π is the probability of the agent performing the action in state s, A(s) is the estimated value of the advantage in state s, H(π) is the policy entropy, and β is a constant parameter.

Instead of the logarithm of the probabilistic distribution of actions, it is proposed to use a cross-entropy analogue as follows:

$$L = ((1 - \sigma) \cdot log(1 - \pi(s)) - \sigma \cdot log(\pi(s))) \cdot A(s) - \beta \cdot H(\pi) \tag{2}$$

where σ is the binary value (equal to 1 for the action actually performed and 0 for all other actions), and the gradient is calculated for all actions.

It is found that the proposed form of functionality helps to keep the values of the policy logos close to each other. That is, it does not allow the agent to fall into such a state when he always chooses the same action.

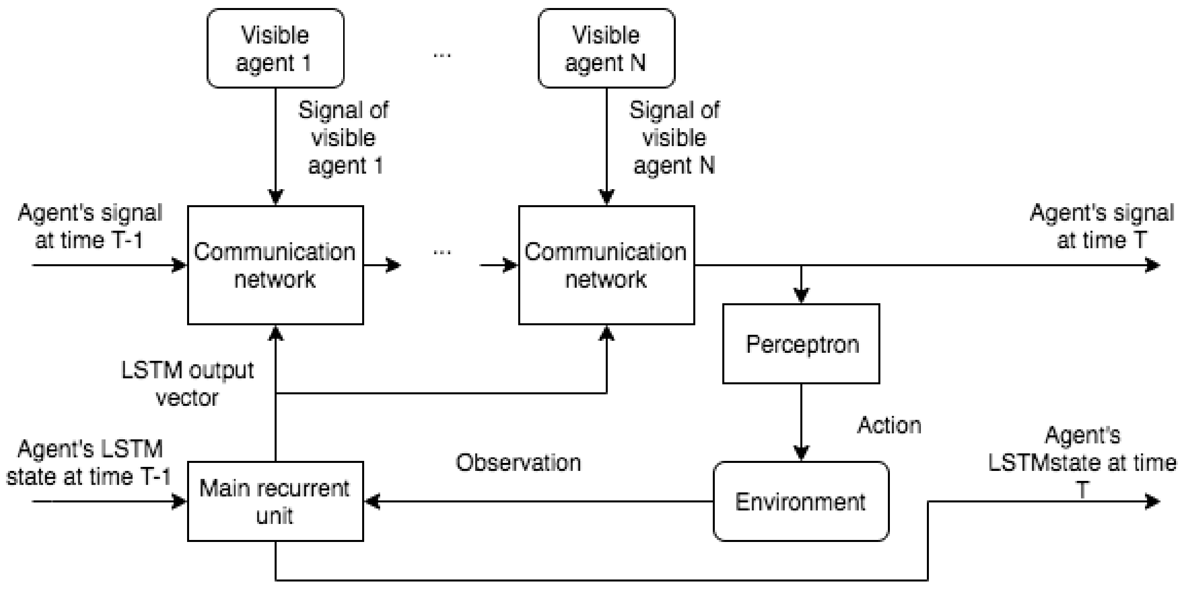


Figure 3. Data flow diagram according to the second version of the algorithm.

An experiment was performed for which we used a maze environment: a 5x5 cell field; there may be a wall between the cells that restricts movement and visibility. One of the cells is the "exit", in the other three cells are agents. Agents observe the horizontal and vertical distance to the nearest walls, the number of left and right corners in each of the four directions, and the direction and distance to the exit cell, if any. However, agents do not know their own coordinates. The agent's task is to explore the maze and find the exit cell. The interaction with the environment ends when the agent is in the exit cell. Guaranteed that there are paths between any two cells of the maze.

The purpose of the experiment is to determine whether an agent team with a communications network will completely leave the maze faster than a team without communication. Four agent versions are compared:

1. Agent without communication and without the ability to see other agents
2. Agent without communication, but able to "see" other agents - if other agents are visible, agents receive direction and distance from each other as part of sensory data
3. An agent capable of seeing other agents and with a communication network of version 1.
4. An agent capable of seeing other agents and with a communication network of version 2.

The agent team is considered to have successfully completed the maze when all agents reach the exit cells of the maze. Agents must minimize the number of moves that agents make before the team exits the maze.

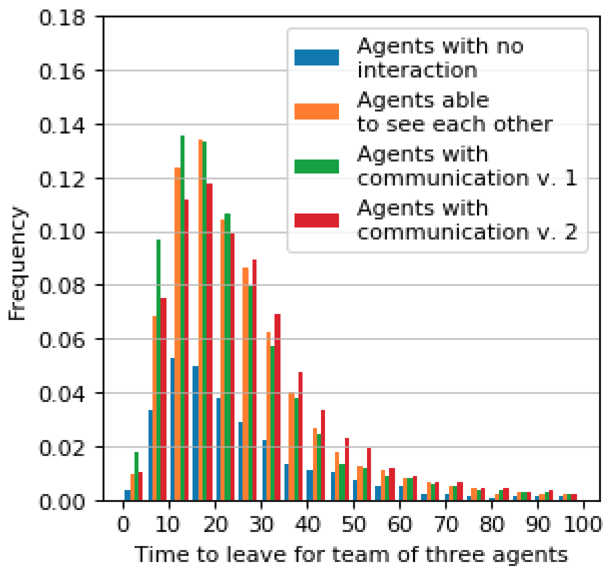
Using a reinforcement learning algorithm involves a reward function that agents will strive to maximize. It has been experimentally determined that the reward function must have "gradient" in some sense - that is, the reward should be given not only when the agent reaches the goal, but also when the agent is acting in the right direction. However, the reward function can be very simple: it is enough to point out actions that were in the “right direction” and which were in the wrong direction. The gradient descent algorithm was chosen according to the recommendations of the authors of the A3C algorithm [2]. The learning rate and others parameters were chosen experimentally.

Instead of running a large number of concurrent simulations and synchronization after each episode, only two concurrent worker threads are used and synchronized once in a few episodes. This solution is because of the use of the OpenMP platform [6] with Intel Xeon processors - two simulation threads run much faster than many threads, and periodic synchronization simulates the large number of threads. Another optimization of the experiment is the formation of a team with one agent during the training of "blind" agents, who can neither see the other nor communicate.

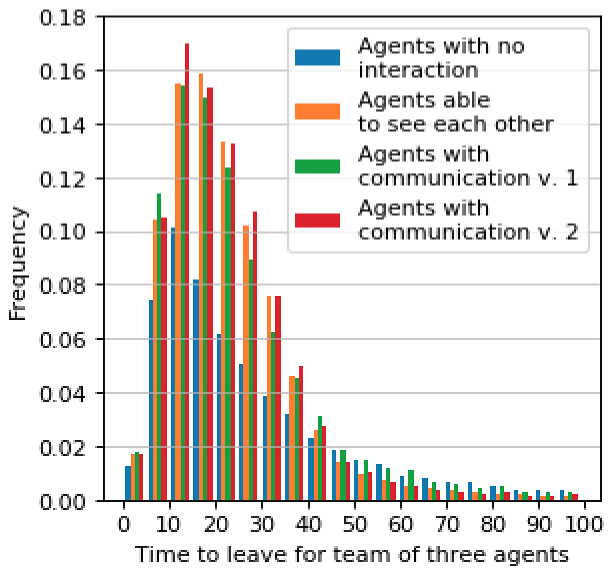
After 30,000 training episodes, training is stopped and performance of teams of three agents in all four configurations is tested. After that, another 20,000 training episodes were performed for each model and tested again. The test results represent the number of runs in each of the simulations. That is, four sets of results were generated for agents after 30,000 training episodes, and four more after 50,000 episodes (see Figure 4).

The agent performance statistics are summarized in Table 1. The table provides information on how many episodes of all 10,000 simulations were successful for the agent team (ie, all agents left the maze). The best result for each of the statistics are in the table.

During the training, the statistics of the effectiveness of agents were recorded over time. Figure 5 shows a graph of the average number of moves per training episode in different iterations of training. The average number is given for clarity, since the number of moves has a large variance.



(a)



(б)

Figure 4. Distribution of time to leave the maze for teams of agents of different architecture: (a) after 30 thousand training episodes; (b) after 50 thousand training episodes.

The graph shows that the blind agent (without communication and without the visibility of other agents) is performing better than others. However, it should be noted that the probability of the initial location of one agent near the exit in a random maze is much higher than the probability of the location of all three agents near the exit. Therefore, the agent of the simplest architecture was considered only to compare the work of the team against one.

The agent training history (see Figure 5) shows that initialization of the agent’s network with version 2 communication was unsuccessful. And at the stage of 30 thousand iterations of training the effect of a bad start has not yet been smoothed out. However, at 50,000 iterations, version 2 began to show a trend toward better results. Also, the statistics show that the ability to see other agents is a very strong heuristic when looking for a way out.

This experiment shows that the proposed algorithm (namely version 2) is capable of increasing the efficiency of the team of agents but needs further research to improve it. This algorithm can be used to find and explore optimal strategies in environments with further formalization and implementation in the form of an explicitly programmed algorithm. The advantage of the algorithm is that it does not require sophisticated and structured computing systems, and can potentially be implemented with very small objects, such as macromolecules.

Statistics of maze passing by teams of agents with different control network configurations in two stages of training.

Configuration	Percentage of successful passages, %	Average time to leave	Most frequent time to leave	I quartile of time to leave	Median of time to leave	III quartile of time to leave
After 30,000 training episodes						
Without seeing others and communicatio n	29.20	24.91	**11**	13	20	32
Able to see others	72.84	24.67	13	14	21	31
Communicati on of version 1	**75.61**	**23.34**	13	**12**	**19**	**29**
Communicati on of version 2	74.49	26.10	16	14	22	34
After 50,000 training episodes						
Without seeing others and communicatio n	56.76	26.62	11	**12**	21	35
Able to see others	87.08	**22.32**	15	13	20	**29**
Communicati on of version 1	87.21	23.64	**10**	**12**	20	30
Communicati on of version 2	**88.85**	**22.19**	12	13	**19**	**29**

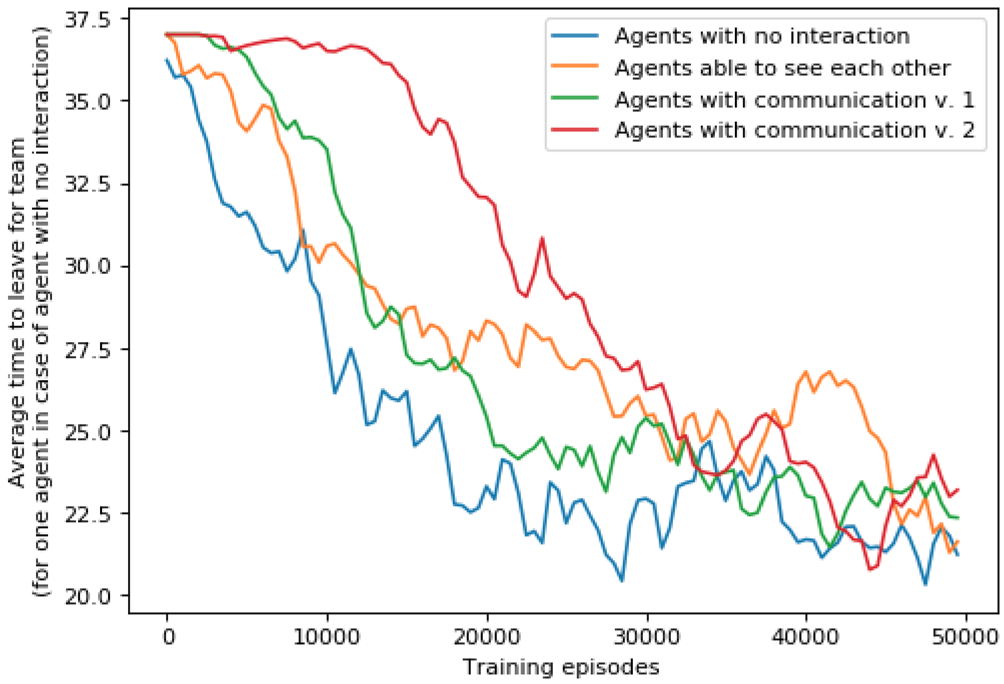


Figure 5. Movable average of the number of moves of agents at different training stages.

REFERENCES

1. Playing Atari with Deep Reinforcement Learning [online] / [V. Mnih, K. Kavukcuoglu,, D. Silver та ін.]. – 2013. – Available at: https://arxiv.org/abs/1312.5602.

2. Asynchronous Methods for Deep Reinforcement Learning [online] / Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza та ін.]. – 2016. – Available at: https://arxiv.org/abs/1602.01783.

3. OpenAI Five [online] – Available at: https://openai.com/five/.

4. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II [online]. – 2019. – Available at: https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii.

5. Sepp Hochreiter. Long Short-Term Memory [online] / Sepp Hochreiter, Jurgen Schmidhuber. – 1997. – Available at: http://www.bioinf.jku.at/publications/older/2604.pdf.

6. OpenMP [online] – Available at: https://www.openmp.org/