UDC 004.056.57:032.26

NOVOTARSKYI M.,
KUZMYCH V.

USAK METHOD FOR THE REINFORCEMENT LEARNING

In the field of reinforcement learning, tabular methods have become widespread. There are many important scientific results, which significantly improve their performance in specific applications. However, the application of tabular methods is limited due to the large amount of resources required to store value functions in tabular form under high-dimensional state spaces. A natural solution to the memory problem is to use parameterized function approximations. However, conventional approaches to function approximations, in most cases, have ceased to give the desired result of memory reduction in solving real-world problems. This fact became the basis for the application of new approaches, one of which is the use of Sparse Distributed Memory (SDM) based on Kanerva coding. A further development of this direction was the method of Similarity-Aware Kanerva (SAK). In this paper, a modification of the SAK method is proposed, the Uniform Similarity-Aware Kanerva (USAK) method, which is based on the uniform distribution of prototypes in the state space. This approach has reduced the use of RAM required to store prototypes. In addition, reducing the receptive distance of each of the prototypes made it possible to increase the learning speed by reducing the number of calculations in the linear approximator.

Keywords: reinforcement learning, Kanerva coding, function approximation, prototype, value function

1. Introduction

Given the increased interest in the application of intelligent systems in various fields of science and technology, the relevance of the development and practical implementation of reinforcement learning algorithms has increased. In recent years, we can see such significant progress in this area that we can say that reinforced learning already has the main features of a particular field of science. A number of university courses have been created [1,2,3,4], there is a basic textbook on reinforced learning [5], which the authors constantly keep up to date. Tabular reinforcement learning methods, namely Multi-armed Bandits [6], Finite Markov Decision Processes [7], Monte-Carlo Methods [8], Temporal-Difference Learning [9] and their modifications are already considered as classical approaches. They are, in most cases, used as part of the development of modern approaches to reinforcement learning. The reason for this is the fact that the solution of modern intellectual tasks requires large or multidimensional state spaces, which leads to significant problems in creating optimal policies.

The natural direction in which reinforcement learning methods are developed is that learning agents can use approximate functions, which makes it possible to significantly improve the productivity of learning in cases of large-scale state spaces [10]. In this case, as a rule, we use parameterized functions that ensure the successful operation of intelligent systems that describe the tasks of the real world. However, constructing an approximate function is not an easy task. Such construction in most cases requires pre-configuration, which includes manual allocation of state spaces based on expert evaluation or uses complex heuristic algorithms. It is known that to build effective heuristic algorithms it is also necessary at the first stage to perform an analysis of the properties of the state space for their correct separation. Therefore, the mentioned approaches to the construction of approximate functions have a disadvantage, which is associated with the problems of dynamic scaling of the state space. This scaling is extremely important for real-world tasks, as in most cases it causes a significant increase in RAM, the available size of which is a major constraint on the way to improving reinforced learning. One effective solution to this problem has been proposed in [11] in the form of a new sparse distributed memory (SDM). A little later, this approach was called "Kanerva coding" after its creator Pentti Kanerva, who first proposed it in 1988. The main advantage of the proposed approach is that it requires a much smaller amount of RAM in the case of increasing state space than standard methods with approximate functions [12].

The essence of Kanerva coding method is that we choose a certain set of prototypes, each of which is a copy of one of the possible states of the environment. Each of these prototypes has a dimension that coincides with the dimension of the state space. Then the selected s state is called adjacent to the p_i prototype if the bitwise difference between the s state and the p_i prototype is less than some predetermined threshold number. Each p_i prototype has a corresponding value θ_i, which is a component of the parameter vector. Then the approximate value of each state-action pair is determined by the sum: $\sum_i \theta_i \mu_i(s)$. With this approach, Kanerva coding eliminates the exponential growth of memory with increasing state space dimension. However, modern reinforced learning tasks increasingly require the use of state space with a dimension of several thousand. In this case, it is necessary to look for new approaches, some of which have been considered in detail and partially proposed in [13]. Among the proposed approaches, the Similarity-Aware Function Approximation method is original and effective. The basics of applications of this method are presented in [14]. This method generalizes the concept of similarity by introducing a new continuous metric, which increases the resolution of the method and thus reduces the number of prototypes that need to be used.

However, the method does not solve the main problem of choosing the optimal number of prototypes. Therefore, the effectiveness of this method can be seen only after a detailed preliminary analysis, which aims to find a fine line between the prototype starvation and over-generalization. In this paper, we propose an approach that allows the use in the linear approximator only those prototypes that are adjacent to the current state of the environment. This approach makes it possible to optimize the use of RAM, provided that large-scale state spaces are used in reinforcement learning.

2. Basics of related methods

Reinforced learning systems are based on a single concept that includes agents, environments, and states, as well as actions and rewards. The general scheme showing the main interactions of these entities is shown in Fig. 1.

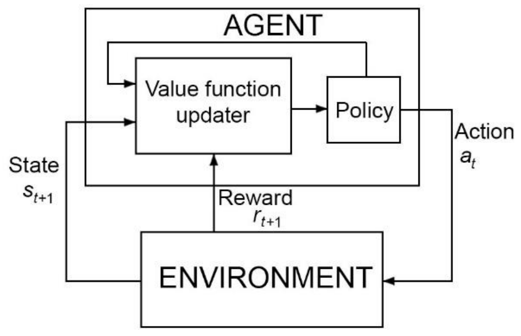

Fig. 1. Generalized structure of the reinforcement learning system

The main components of this system are the agent and the environment, which interact with each other. An agent is a certain entity that has the ability to produce actions that affect the environment. The environment is the world around the agent, which perceives the a actions and returns the r reward to the agent. The agent also has the ability to observe the current state, s, of the environment. Such a system is sometimes called a feedback system in which the role of feedback signals is played by signals of reward and state of the environment. The ways in which the agent chooses the next action are basic and are determined by the methods of his learning. At each point in time, the policy determines the behavior of the agent. It maps the set of all accumulated knowledge about the environment to the action to be performed. The sole purpose of the agent is to maximize the reward he can receive from the environment. The size of the reward signal determines the agent's policy choice. However, a policy that only takes into account the remuneration for the last action may conflict with the agent's overall goal if it leads to a significant reduction in remuneration in the future. The size of the reward signal determines the agent's policy choices. However, a policy that only considers reward for the last action may conflict with the agent's overall goal if it leads to significantly reduced reward in the future. Previous experience of interaction with the environment, which takes into account the value function, helps to solve this problem. Figure 1 shows that the input data for the policy is formed by the value function, which takes into account not only the current state and current reward, but also previous states and rewards with a certain discount.

2.1. Reinforced learning with function approximation

We will consider a reinforcement learning system, which includes an agent represented by an algorithm that has the property of training. At a t time, the agent performs an a_t action in accordance with a $\pi\left(a_t | s_t\right)$ policy that specifies the probability of performing the action under the s_t state of the environment. At the next $t+1$ time, the agent receives a r_{t+1} reward and the result of observation the s_{t+1} state of the environment. At each time step, the agent tries to modify his π policy so as to maximize the total amount of rewards in each episode using the expression:

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \tag{1}$$

where g_t is the total return of the agent, starting from t time; γ is the rate of the discount, which varies in $0 \le \gamma \le 1$ range; T is the maximum length of the history, which in the general case can be equal to infinity. Consider the principle of the agent, which combines Q-learning with the mechanisms of approximation for the successful operation of reinforcement learning systems using high-dimensional state spaces. Q-learning is based on the use of the Q-function in order to implement π policy. This function is also called the state-action value on-policy function. It is denoted as $Q^{\pi}\left(s,a\right)$. Using the Bellman optimality equation, we can find the $Q^{*}\left(s,a\right)$ optimal value as the maximum return; starting from the s state under condition we provide the actions and operations in accordance with the π policy:

$$Q^{*}\left(s,a\right) = \mathrm{E}\left[r + \gamma \max_{a'} Q^{*}\left(s',a'\right)\right]. \tag{2}$$

The reinforcement learning algorithm consists in realization of iterative process of the value function estimation:

$$Q_{i+1}\left(s,a\right) = \mathrm{E}\left[r + \gamma \max_{a'} Q^{*}\left(s',a'\right) | s,a\right]. \tag{3}$$

This sequence converges to the optimal value: $\lim_{i\to\infty} Q_i = Q^{*}$. However, the achievement of this result encounters problems of practical implementation due to the fact that the representation of the Q-function in tabular form leads to an exponential increase in the amount of RAM when using high-dimensional state spaces. To solve this problem, we present the Q -function as a parameterized function: $Q^{*}\left(s,a\right) \approx Q\left(s,a;\boldsymbol{\theta}\right)$, where $\boldsymbol{\theta}$ is parameter vector. Then the problem of representing the state-action function can be reduced to choosing values of a much smaller number of components of the $\boldsymbol{\theta}$ vector. In most cases, this uses a linear approximator, which is represented by the expression:

$$Q\left(s,a;\boldsymbol{\theta}\right) = \boldsymbol{\theta}^T \mathbf{x}(s,a) = \sum_{i=0}^{n-1} \theta_i x_i\left(s,a\right), \tag{4}$$

where $\mathbf{x}\left(s,a\right) = \left(x_0\left(s,a\right), x_1\left(s,a\right), \ldots, x_{N-1}\left(s,a\right)\right)$ is the feature vector, which represents the state-action function; $\boldsymbol{\theta}^T$ is the transposed parameter vector with the number of components equal to the number of features; n is the number of dimensions of the state space. Features in this case are $x_i\left(s,a\right)$ basic functions, because they form a linear basis to create the $Q\left(s,a;\boldsymbol{\theta}\right)$function approximation. There are different approaches to the selection of the feature vector components. One such approach is, for example, the stochastic gradient-descent method, for which $\mathbf{x}(s,a)=\nabla Q\left(s,a;\boldsymbol{\theta}\right)$. Then the iterative process of finding the vector of parameters is determined by the expression:

$$\boldsymbol{\theta}_{t+1}\leftarrow\boldsymbol{\theta}_t+\alpha\left[Q^{\pi}\left(s_t,a_t\right)-Q\left(s_t,a_t;\boldsymbol{\theta}\right)\right]\mathbf{x}\left(s_t,a_t\right). \tag{5}$$

As is known, for a stochastic gradient-descent method, this iterative process converges to a local minimum, which does not always coincide with the global minimum. Since in most cases it is necessary to find out a global minimum, it is important to find other function approximation techniques.

2.2. Algorithms for high-dimensional state spaces

Algorithms for high-dimensional state spaces are extremely relevant because in most cases they are the only possible approach to solve practical tasks of reinforcement learning. A well-known algorithm for large state spaces is the tile coding algorithm [5]. This algorithm has become widespread due to its simplicity and efficiency. An important disadvantage of this algorithm is that it requires pre-allocation of state space manually. Such preliminary selection significantly affects the further efficiency of the algorithm.

There are usually a large number of modifications of this algorithm that use different heuristic solutions to implement an efficient distribution of state space
[16, 17]. However, such heuristic algorithms require the use of a large number of computing resources to process current information on each of the state space dimension and pre-configure a large number of parameters that characterize the environment. Therefore, the effective use of these algorithms for reinforcement learning with the high-dimensional state space is difficult and available only after gaining some experience.

Radial basis functions (RBF) is the second known algorithm for the implementation reinforcement learning in large state spaces [5]. For this algorithm, the features have a continuous value in the range [0,1]. Typically, RBF-features are subject to normal distribution. The advantage of such features over binary ones is that the corresponding functional dependencies are differentiated, which expands the possibilities of management and analysis. The disadvantages of the RBF method repeat the same disadvantages as tile coding. These disadvantages include the increased computational complexity and the need for manual adjustment, which requires an expert level of use of this method.

The most promising algorithm has every reason to consider the Kanerva coding algorithm. Today, this is one of the algorithms that is characterized by the smallest increase in RAM with increasing dimensionality of the state space.

In this method, as in the previous ones, it is necessary to choose the vector of basic functions. They are represented by prototypes of states in this method. New $p=\left(\varphi_0,\varphi_1,...,\varphi_{n-1}\right)$ prototypes are created based on the components of the state space. The method uses similarity for each of the dimensions of the state space, which is equal to the code distance between the binary representation of the vectors of the prototype and the state. The $\mu\left(s,p_i\right)$ function is equal to one, provided that the s state is close to the p_i prototype and zero otherwise. This function is called the membership grade function. It can be represented by the expression:

$$\mu_i\left(s,p_i\right)=\begin{cases}1 \ \ if \ \sum\limits_{j=0}^{n-1}\left(\phi_j \ XOR \ \left(\varphi_j\right)_i\right)\le c,\\ 0 \ \ otherwise.\end{cases} \tag{6}$$

where j is the bit number, n is the state space dimensionality, c is the threshold value.

For Kanerva coding, there is also the problem of choosing the optimal set of prototypes, which in practice is solved by randomly selecting from the available set of states. Such a choice does not solve the problem of optimality. In this case, there may be significant disturbances in the distribution of receptive fields and, as a consequence, there is a decrease in the efficiency of representation of certain areas of the state space and the corresponding value functions. An adaptive adjacency method [15] has been proposed as one of the known ways to improve the coverage of state space by prototype receptive zones [15], but such an approach also cannot always solve the problem, but only reduces the percentage of failed coatings.

Also known methods that partially solve the problem of optimal coverage based on the use of fuzzy logic. But such approaches are a separate area, which we do not consider in this paper.

An original method called Similarity-aware Kanerva or SAK for short has been proposed recently.

2.3. SAK method

The SAK method significantly changes the principle of determining the similarity of states and prototypes and thus solves the problem of determining the fields of perception for the selected set of prototypes. The main difference of this approach is that the features are no longer binary, but are represented as continuous quantities, which are represented by real numbers [14].

For a feature with the j index, we determine the d_j distance between the p prototype and the s state by the expression:

$$d_j = \frac{|\phi_j - \varphi_j|}{range_j / \rho}, \tag{7}$$

where $range_j$ is the range of possible values of the j feature, ρ is a fixed factor, the value of which exceeds 1 and provides sensitivity to changes in the $|\phi_j - \varphi_j|$ difference.

Define the similarity grade for the j feature using the expression:

$$m_j(s,p) = e^{-d_j(s,p)}. \tag{8}$$

Fig.2 shows the graph of this function:

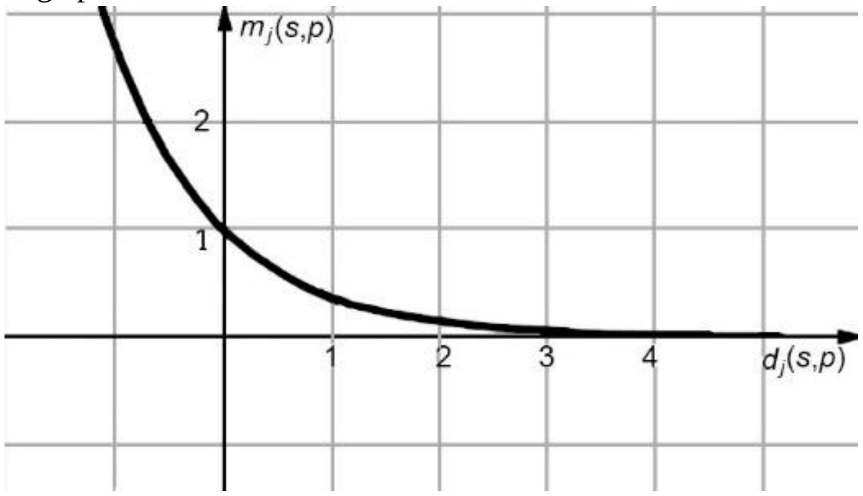

Fig. 2. Graph of the $m_j(s,p)$ function

The figure shows that the value of the $m_j(s,p)$ function is equal to 1, if the value of the difference between the dimension of the ϕ_j state and the φ_j prototype approaches zero with increasing distance between dimensions.

The $\mu(s,p)$ membership grade of states and prototypes is equal to the minimum similarity between their dimensions:

$$\mu(s,p)=\min_{j=0,1,\dots n-1} m_j(s,p). \tag{9}$$

Thus, the membership grade for states is a continuous value that varies in the range [0,1].

When using Kanerva coding, we can specify a fixed number of prototypes. In the case of using a linear approximator for Kanerva coding, the value of the state-action function approximation is determined from the expression:

$$Q(s,a;\theta)=\sum_{i=0}^{|P(s)|}\theta(p_i,a)\cdot\mu(s,p_i), \tag{10}$$

where $|P(s)|$ is the power of the indexed list of prototypes that are similar to the s state.

The $\theta(p_i,a)$ parameter value will be stored and updated for each p_i prototype when selected the a action by the agent. When using the Sarsa algorithm, the following update is determined from the expression [15]:

$$\theta_i(p_i,a)\leftarrow\theta_i(p_i,a)+\delta(s,p_i)\alpha(s,a)\left[r+\gamma\left(Q(s',a';\theta_{i-1})-Q(s,a;\theta_i)\right)\right], \tag{11}$$

where $\delta(s,p_i)=\dfrac{\mu(s,p_i)}{\sum_{k=0}^{|P(s)|}\mu(s,p_k)}$ is a current fraction of the membership grade, $\alpha(s,a)$ is the step size parameter.

Although the SAK algorithm significantly improves the basic parameters of reinforcement learning compared to the Kanerva coding algorithm, it does not solve the main problems of this type of algorithms. In particular, the algorithm uses random generation of prototypes. While it is known that using prototype tuning, actually increase the average solution rate from 67.9% to 97.1% [18]. In addition, the difficult question of choosing the optimal number of prototypes remains unclear. The choice of the optimal number of prototypes makes it possible to avoid the prototypes starvation due to the insufficient number of active prototypes for the value function approximation, as well as to avoid over-generalization of prototypes.

3. USAK algorithm

This paper proposes a Uniform Similarity-Aware Kanerva algorithm (USAK) that differs from SAK by adding a heuristic that improves the use of prototypes and proposes some modifications to include prototypes in the ActiveN list [14].

For an arbitrary j dimension, we determine the d_j distance between the p prototype and the s state from the expression:

$$d_j=\left|\phi_j-\varphi_j\right|. \tag{12}$$

The similarity grade for a j feature is determined from the condition of uniform placement of prototypes in the range of this feature.

$$m_j(s,p)=1-\frac{2b_jd_j}{\rho\cdot range_j}, \tag{13}$$

where ρ is the coefficient of overlap of the receptive zones of the prototype: $1<\rho\le1.5$, $range_j$ is the range of the j feature, b_j is the number of prototypes in the range of the j feature.

The graph of the function is shown in Fig.3.

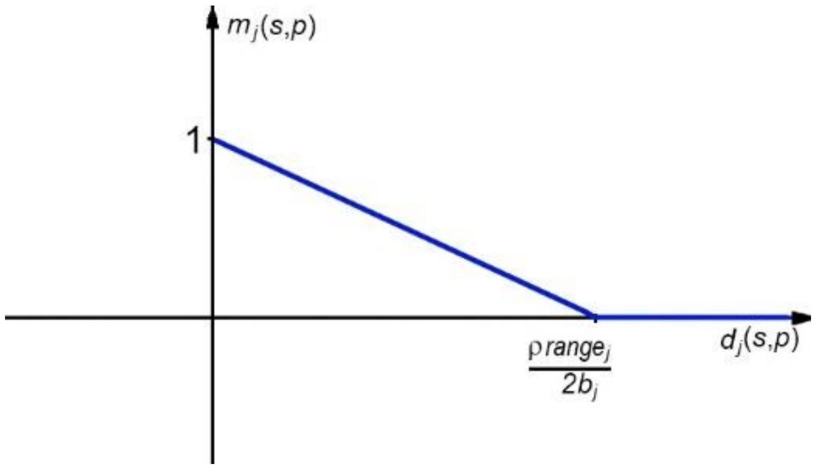

Fig. 3. Graph of the $m_j(s,p)$ function for the USAK algorithm

The graph in fig. 3 specifies a linear normalized value of the $m_j(s,p)$ similarity grade for the j feature. The value of $m_j(s,p)$ is equal to 1 if the value of the difference between the ϕ_j state dimension and the ϕ_j corresponding dimension of the prototype is zero. This $m_j(s,p)$ function linearly decreases to zero with increasing distance between dimensions and is equal to 0 at the perceptive limit of the prototype respectively to the j feature.

The $\mu(s,p)$ membership grade of the s state and the p prototypes is equal to the minimum similarity grade for their dimensions (9). Thus, the membership grade for states is a continuous quantity that varies linearly in the range [0,1]. Since the prototype has clear and limited perceptive zones for each of its features, in this case it is important to fully cover the state space with prototypes for each of the state dimensions. In this paper, an algorithm for uniform coverage of the state field is proposed.

In the general case, consider the n-dimensional state space with uniform coverage of this space by prototypes. Then there is no need to save all prototypes, because they can be dynamically generated by setting the values of their dimensions with a predetermined step. Consider the pseudocode of the USAK algorithm, which includes the dynamic generation of features of evenly distributed prototypes.

Algorithm 1 : Uniform Similarity-Aware Kanerva Coding

Input:

$\mathbf{b}=(b_0,b_1,...,b_j,...,b_{n-1})$: the number of prototypes for each dimension.

$\mathbf{range}=(range_0,range_1,...,range_j,...,range_{n-1})$: range of features for each dimension.

N : number of episodes.

M : the maximum number of iterations in the episode.

Output: $\boldsymbol{\theta}$ vector as a learning outcome.

```
def P_generator($s'$):
  for $i$ in range($B$)
     for $j$ in range(n):
          $\phi_j = s'(j)$
          $\varphi_j = i \times \frac{range_j}{b_j+1}$
          $d_j = |\phi_j - \varphi_j|$
```

```
            $m_j = 1 - \frac{2b_j d_j}{\rho \cdot range_j}$
            if $m_j > 0$ then $\mu.append(m_j)$ else $\mu.append(0)$
        $\mu_i = \min(\mu)$
        if $\mu_i > 0$ then $\mathbf{p}_{active}.append(i)$; $\boldsymbol{\mu}_{active}.append(\mu_i)$
    return $\mathbf{p}_{active}$, $\boldsymbol{\mu}_{active}$

def Teta_iter( $s_t, a_t, r_t, s_{t-1}, a_{t-1}$):
    for $i$ in range($|\mathbf{p}_{active}|$):
        $\delta(i) = \frac{\mu_i}{\sum_{k=0}^{|\mathbf{p}_{active}|} \mu_k}$
        $\theta_i = \theta_i + \alpha_t \left[ r_t + \gamma Q(s_t, a_t; \boldsymbol{\theta}_t) - Q(s_{t-1}, a_{t-1}, \boldsymbol{\theta}_{t-1}) \right] \delta(i)$
def Main():
    Determining the number of prototypes: $B = b_0 \times b_1 \times ... \times b_{n-1}$
    Initialization of the parameter vector: $\boldsymbol{\theta} = 0$, where $|\boldsymbol{\theta}| = B$.
    for episode in range(1,N):
        Randomly choose $s_0$ and $a_0$.
        for t in range(1,M):
            Perform $a_{t-1}$ in $s_{t-1}$.
            Get $r_t$ and $s_t$.
            Determine the $\mathbf{p}_{active}$ vector of active prototypes
            and the $\boldsymbol{\mu}_{active}$ distance vector:
            $\mathbf{p}_{active}, \boldsymbol{\mu}_{active} = \text{P\_generator}(s_t)$
            Determine the value of the $Q(s_t, a_t; \boldsymbol{\theta})$ function approximation:
            $Q(s_t, a_t; \boldsymbol{\theta}_t) = \sum_{i=0}^{|\mathbf{p}_{active}|} \theta_i \cdot \mu_{active}(i)$
            Update $\boldsymbol{\theta} : \boldsymbol{\theta}_{t+1} = \text{Teta\_iter}(s_t, a_t, r_t, s_{t-1}, a_{t-1})$
            Determine $(s_{t+1}, a_{t+1})$ using the $\varepsilon$-greedy method:
            $(s_{t+1}, a_{t+1}) = \arg\max_s Q(s_t, a_t, \boldsymbol{\theta}_t)$
                $s_{t-1} = s_t$, $a_{t-1} = a_t$
```

4. Practical implementation

Let us consider the work of the proposed method on the example created on the basis of the popular problem "WaterWorld", which was first proposed by Andrej Karpathy [19]. Its essence is that the agent must try to survive by avoiding collisions with objects that move freely in the environment. The author of this problem compares it with the problem of spacecraft navigation in the field of asteroids.

The agent receives +1 for survival for one time step and -100 points in case of collision with the simultaneous end of the current episode. To prevent collision, the agent has a set of sensors arranged evenly in a circle. The number of sensors can be changed during the experiments. Obviously, the dimensionality of the state space depends on their number. Each sensor allows you to determine the coordinates of the nearest object in the area of its observation, speed and direction of its movement. The agent also has additional sensors that determine its own coordinates, speed and direction of its movement.

The total number of features varied from 44 to 104, creating a state space with the appropriate dimensionality. The agent can perform 4 actions: left, right, up and down. Each of these actions is represented by a real number that indicates the modulus of velocity of the object in a certain direction. The experiments use agents that have from 4 to 20 sensors. Figure 4 shows a screenshot of the task, which shows the position of the agent with 18 sensors and the environment in which objects float freely.

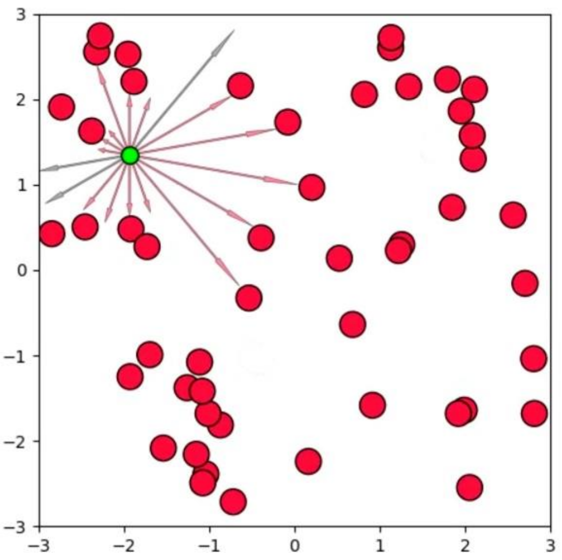

Fig.4. An agent in an environment with "hostile" objects

As shown in Figure 4, the agent is also able to distinguish the wall of the area from floating objects. The modulus of the inverse vector of the velocity of repulsion from the wall is proportional to the modulus of the vector of pushing on the wall. The purpose of the agent is to prolong its existence as much as possible, avoiding collisions with these "enemy objects".

The studies were performed using the USAK algorithm for the described problem. Such studies aimed to experimentally determine the indicators of its effectiveness in a given range of parameters.

Figure 5 shows the rewards for USAK algorithms with a different dimension of the state field. From this graph we can conclude that the speed of learning has no critical dependence on the dimensionality of the state speed. However, the advantage of the USAK algorithm is a 42% reduction in the number of calculations per episode due to the exclusion of remote prototypes from the calculations.

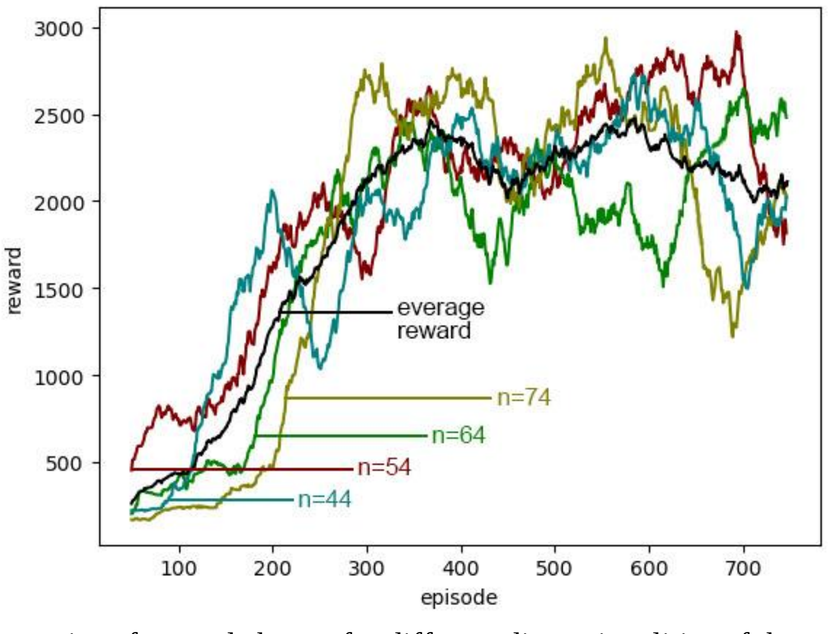

Fig. 5. Dynamics of reward change for different dimensionalities of the state space

Figure 6 shows the increase in occupied RAM over time. This growth does not depend significantly on the size of the state space, which indicates the effectiveness of the proposed approach to the uniform arrangement of prototypes that cover the state space. The advantage of this approach is that remote prototypes do not participate in the calculation, which leads to a linear increase for data to be stored. In this study, as in the previous case, we considered the state spaces with 44, 54, 64, and 74 dimensionalities. These dimensionalities of the state spaces are formed by agents with 8,10,12 and 14 sensors, respectively.

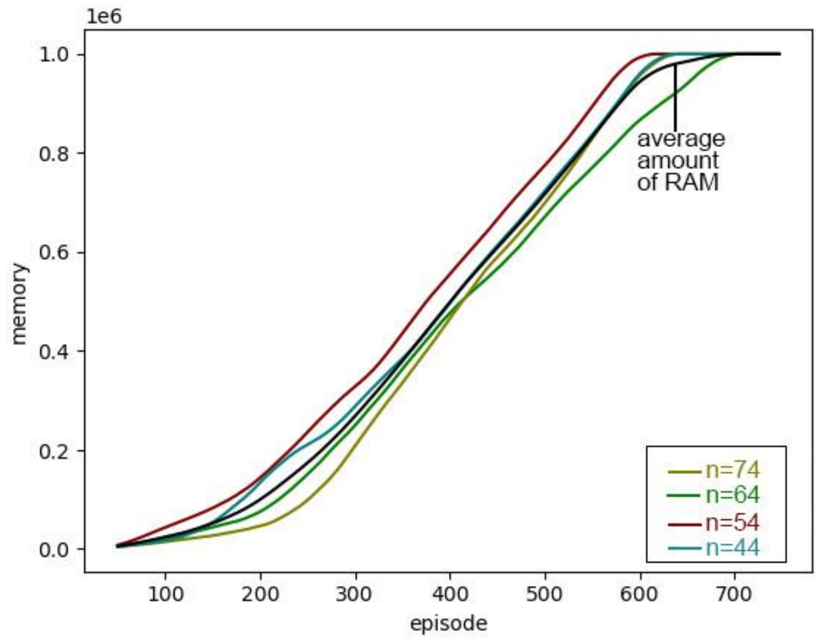

Fig. 6. Linear growth of occupied RAM as a function of time for state spaces of dimensionalities 44, 54, 64 and 74.

5. Conclusion

This paper discusses modern approaches to reducing the use of computer RAM in solving reinforced learning problems with high-dimensional state spaces. The USAK method is proposed, which improves the characteristics of the SAK method due to the uniform distribution of prototypes that cover the state space. Even distribution of prototypes has reduced the amount of RAM for their storage because such placement of prototypes allows you to generate prototypes automatically with the appropriate step of their dimensions. The second advantage of uniform placement of prototypes is that it is possible to use a predetermined fixed perceptivity of prototypes. This fact has a positive effect on the reduction of the volume of calculations in the linear approximation of the value function, because in this case only those prototypes that have a membership value greater than zero are taken into consideration. A number of experiments were conducted on the basis of the well-known problem "WaterWorld", which allowed to determine the advantages of this method, as well as to form their vision of ways to further improve it.

References

1. The Stanford University (2020), "CS234: Reinforcement Learning", available at: https://web.stanford.edu/class/cs234/
2. The University of Edinburg (2020), "Reinforcement Learning", available at: http://www.inf.ed.ac.uk/teaching/courses/rl/
3. The University of Alberta (2020), "Fundamentals of Reinforcement Learning", available at: https://www.classcentral.com/course/fundamentals-of-reinforcement-learning-14497

4. Silver D. (2020), “UCL Course on RL”, University College London, available at: https://www.davidsilver.uk/teaching/
5. Sutton R. S. and Barto A.G. (2018), “Reinforcement Learning: An Introduction”, Cambridge: The MIT Press, , available at: http://www.academia.edu/download/38529120/9780262257053_index.pdf
6. Slivkins A. (2019), “Introduction to Multi-Armed Bandits”, available at: https://arxiv.org/abs/1904.07272v5
7. Levin D.A. and Peres Y. (2017), “Markov chains and mixing times”, available at: https://www.statslab.cam.ac.uk/~beresty/teach/Mixing/markovmixing.pdf.
8. Wiering M. fand van Otello M. (2012), “Reinforcement Learning”, Berlin: Springer-Verlag.
9. Hester T. (2013), “TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains”, Berlin: Springer-Verlag.
10. Whiteson Sh. and Stone P. (2006), “Evolutionary Function Approximation for Reinforcement Learning“, Journal of Machine Learning Research, Vol. 7, pp. 877-917
11. Kanerva P. (2003), “Sparse Distributed Memory”, Cambridge: MIT Press.
12. Cheng Wu and Yiming Wang (2017), “Learning From Big Data: A Survey and Evaluation of Approximation Technologies for Large-Scale Reinforcement Learning”, IEEE, Computer and Information Technology (CIT), International Conference,-DOI: 10.1109/CIT.2017.11
13. Wei Li (2019), “Function Approximation-based Reinforcement Learning for Large-Scale Problem Domains”, PhD dissertation, Northeastern University, Boston, Massachusetts..
14. Wei Li and Meleis W. (2018) “Similarity-Aware Kanerva Coding for On-Line Reinforcement Learning”, Proceedings of the 2-nd International Conference on Vision, Image and Signal Processing.
15. Wei Li and Meleis W. (2018), “Adaptive Adjacency Kanerva Coding for Memory-Constrained Reinforcement Learning“, International Conference on Machine Learning and Data Mining in Pattern Recognition, pp.187-201.
16. Sherstov A. A. and Stone P. (2005),” Function Approximation via Tile Coding: Automating Parameter Choice”, International Symposium on Abstraction, Reformulation, and Approximation, pp.194-205.
17. Waskow S. J. and Bazzan A.L.C. (2010), “Improving Space Representation in Multiagent Learning via Tile Coding”, Brazilian Symposium on Artificial Intelligence, pp.153-162.
18. Cheng Wu (2010), “Novel Function Approximation Techniques for Large-scale Reinforcement Learning”, PhD dissertation, Northeastern University, Boston, Massachusetts.
19. Karpathy A. (2020), “REINFORCEjs. WaterWorld: DQN”, available at: https://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html