Application of the Neural Network Approach To Reinforced Learning Methods

Alexandr Eremeev[a], Alexandra Gerasimova[a] and Ilia Poliushkin[a]

[a] *National Research University "Moscow Power Engineering Institute", Krasnokazarmennaya 14, Moscow, 111250, Russian Federation*

Abstract

The paper investigates various modifications of reinforcement machine learning methods based on neural networks. The results of computer modeling (experiments) and comparative analysis of the above teaching methods in terms of their possible application in intelligent real-time systems, in particular, in intelligent decision support systems, are presented.

Keywords 1

Artificial intelligence, machine learning, reinforcement learning, neural networks, intelligent system, real time, decision support

1. Introduction

Among the methods of reinforcement learning (RL), the methods based on temporal differences (TD-methods) deserve special attention. [1]. The main advantage of TD methods is that there is no need to have preliminary data about the environment, since the necessary experience (knowledge) will be obtained directly in the process of interaction between the agent and the environment.

The well-known standard (classical) TD methods (TD (0), SARSA, Q-learning), as well as their modifications using traces of acceptability (TD (λ), SARSA (λ), Q (λ)), are simple enough to implement and can be used in intelligent real-time systems (IRTS), including intelligent real-time decision support systems (IRTDSS) [2]. However, a certain problem for these methods is finding the optimal value of the value function during training, since it is necessary to discretely determine all possible "state-action" pairs and analyze them repeatedly for a qualitative study of the environment. [3].

Decision-making problems cannot always be described discretely in the form of a Markov process and for this reason it is proposed to use artificial neural networks (ANNs) as an additional learning tool.

2. ANN application in reinforcement learning

2.1. Q-function approximation using neural networks

The application of the neural network approach in RL-learning, in particular, in TD-methods with an estimated Q-function (methods Q-learning, Q (λ)) is that the value of the Q-function is not calculated for each visited pair "state-action" (s, a), and the ANN was approximated. There are many modifications of Q-learning using ANNs of various architectures.

Applying ANNs to RL learning methods is called deep reinforcement learning (DRL). The general diagram of the DRL is shown in Figure 1 [3].

Some tasks require preprocessing to be able to use DRL methods (for example, to work with images, you need to use convolution). Then the agent, learning by means of the DRL method using the selected strategy, takes an action, receives a response from the environment in the form of a new

Russian Advances in Fuzzy Systems and Soft Computing: selected contributions to the 8-th International Conference on Fuzzy Systems, Soft Computing and Intelligent Technologies (FSSCIT-2020), June 29 – July 1, 2020, Smolensk, Russia
EMAIL: eremeev@appmat.ru; GerasimovaAl96@yandex.ru; poliushkinia@yandex.ru
ORCID: 0000-0002-3483-6979; 0000-0002-6747-8880; 0000-0002-5924-7986

state and reward, which are stored in the playback memory. The replay memory is used by the agent to adjust the ANN weights.

Within the framework of this article, various modifications of deep learning based on Q-learning using ANN will be considered.

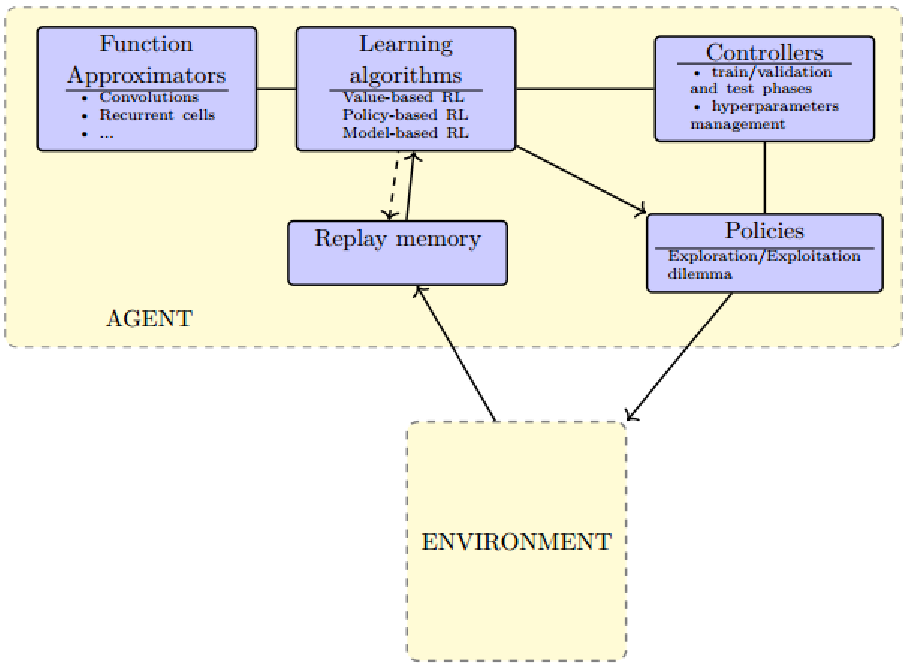

Figure 1: Scheme of Deep reinforcement learning

2.2. Deep Q-Network (DQN)

DQN method is a modification of the standard Q-learning method using ANN. This approach allows us to solve the problem that does not allow using a fairly simple tabular form of the Q-learning method due to the need to store in memory all the estimates of the Q-function for each state-action pair.

Let's look at an example. Let there be infinitely many states in the environment under study, each of which is described by some n-dimensional vector X = {x1, x2… xn}, where X is some state, X∈S is a set of possible states. Let m different actions be performed from each state X. To store such a large amount of information and to process it, appropriate resources are needed. Also, to obtain a solution to this problem, it is necessary to visit each pair of "state-action" a number of times to get a reliable estimate of the Q-function.

A Q-network is a neural network used to approximate the values of a Q-function:

$$Q(s, a;\ \theta)\ \approx\ Q*(s, a), \tag{1}$$

where $Q(s, a; \theta)$ includes a parameter θ that denotes ANN weights.

ANN training is based on the experience that the agent receives in the course of its interaction with the environment $<s_t, a_t, r_t, s_{t+1}>$ at time t. A subset of quadruples $<s_t, a_t, r_t, s_{t+1}>$ is selected from the agent's memory, and for each such quadruple, the value $Q(s,a)$ is calculated, and $Q(s, a; \theta)$ is predicted using the ANN.

The loss function is calculated taking into account the target value $r_t + \gamma \max_a Q(s_{t+1}, a)$ and the predicted value $Q(s, a; \theta)$, which the agent seeks to minimize. The loss function is calculated as follows:

$$loss\ =\ \left(rt\ +\ \gamma\max_a Q(s_{t+1},\ a;\ \theta) - \ Q(s_t,\ a_t;\ \theta)\right)^2. \tag{2}$$

The same Q-net is used to estimate the target and predicted values of the Q-function. The DQN method is based on the search for such an action that maximizes the value of the pair "state-action", and requires optimization of the value at each step of the iteration [4].

The DQN-based algorithm (DQN-algorithm) can be described as follows:

```
1. Initialize the Q-net arbitrarily;
2. Repeat (for each episode):
  2.1. Initialize $s$;
  2.2. Repeat (for each step of the sequence):
    2.2.1. Find $a$ by $s$ using the strategy derived from $Q$;
    2.2.2. Perform action $a$, find $r$, $s'$;
    2.2.3. Remember the set $<s, a, r, s'>$;
    2.2.4. If the amount of agent's “memory” exceeds $n$, then:
      For $n$ memory locations:
        - evaluation of the value $Q = r + \gamma \max_a Q(s', a; \theta)$;
        - correction of neural network weights;
    2.2.5. $s \rightarrow s'$;
  2.3. Until $s$ becomes a terminating state.
```

Thus, the application of Q-learning methods using ANN (on the example of the DQN-method) allows solving those problems to which classical Q-learning cannot be applied.

Note that the DQN method allows solving problems with an infinite number of states, but with a limited (discrete and small-sized) action space.

To handle the instability of the result, two heuristics are used [3]:

- the target Q-net is represented as $Q(s_{t+1}, a_{t+1}; \theta)$, where the parameter θ is updated every n steps. Since the target value remains unchanged for n steps, the instability does not propagate as quickly, thereby reducing the risk of algorithm divergence;
- in RT mode, the playback memory stores a set of tuples $<s_t, a_t, r_t, s_{t+1}>$, from which a subset of tuples is randomly selected to update the weights. This approach allows a wide range of subsets of state-action pairs to be updated. Also, updating by subsets gives less variance compared to methods in which the weights will be updated for each tuple and at each step, and also provides a larger update of parameters and a higher level of parallelism of the algorithm.

2.3. Double Deep Q-Network (DDQN)

DDQN-method - is a modification of the DQN-method, which uses two ANNs.

If we return to the standard Q-learning method, we will see that in the part $\max_a Q(s_{t+1}, a)$, the same estimate is used, which will affect the agent's choice of action in state s_{t+1}. However, in tasks with deferred reward, the agent may be more likely to choose actions that are overestimated due to inaccuracies or noise.

The DDQN method uses two ANNs - one to select the highest estimate and the other to obtain its value. Consequently, regardless of the source of errors in the Q-function (for example, noisy environment, randomness of processes in it, not stationarity), the use of two ANNs allows you to eliminate a positive bias in the assessment of the value function of the "state-action" pairs.

The target value of the assessment of the pair "state-action" at time t is represented as:

$$r_t + \gamma Q\left(s_{t+1}, argmax_{aQ(st+1,a;\,\theta)};\ \theta^{targ}\right). \quad (3)$$

This results in less overestimation of Q-learning values and increased stability and performance [3]. The Q^{targ} network is used to evaluate an action that is chosen according to a greedy strategy.

2.4. Actor-Critic (AC)

The DQN, DDQN methods were based on the approach using a value function (Q-function), the values of which influenced the agent's behavior strategy, and sometimes determined it (for example, a greedy strategy).

Let's consider an approach based on calculating the strategy gradient. The essence of this approach is to optimize the expected estimate of the "state-action" pairs, finding the best strategy, thanks to variations of the stochastic gradient ascent relative to the strategy parameters.

One of the known methods for solving learning problems using TD methods using a gradient strategy is the AC method. The Actor agent, as the name suggests, is used to select an action, and Critic is used to obtain an estimate of the value function (Q-function).

Actor and Critic can be represented by ANN (with nonlinear functions), and Actor uses the gradients obtained from the strategy, changing its parameters, and Critic with the parameter θ, estimates the approximation of the value function for the current strategy *π*. Using the set of tuples <*s*, *a*, *r*, *s'*> from the agent's replay memory, Critic updates the estimate of the value function *Q* to the target value Y^Q:

$$Y^Q_{t+1} = r + \gamma Q(s_{t+1}, a = Actor(s_{t+1}); \theta), \quad (4)$$

based on the adjustment typical of all TD methods.

3. Software implementation of algorithms and their testing

The high-level Python language was chosen as the programming language for the implementation of the considered methods and corresponding algorithms. Development was carried out in the Jupiter Notebook programming environment.

For testing and debugging, the classic problems of testing RL-learning methods are considered, namely, the problem of a mountain car (Mountain Car), stabilization of an inverted pendulum (Cart Pole), and Atari games (MsPacman and Assault).

The ε-greedy strategy is used to solve the exploration-exploitation dilemma. The ε value decreases over time. Figures 2-5 show graphs of the dependences of the average remuneration received by the agent, depending on the episode number. The X-axis is the episode number, the Y-axis is the number of points. Table 1 shows the comparative results of the methods.

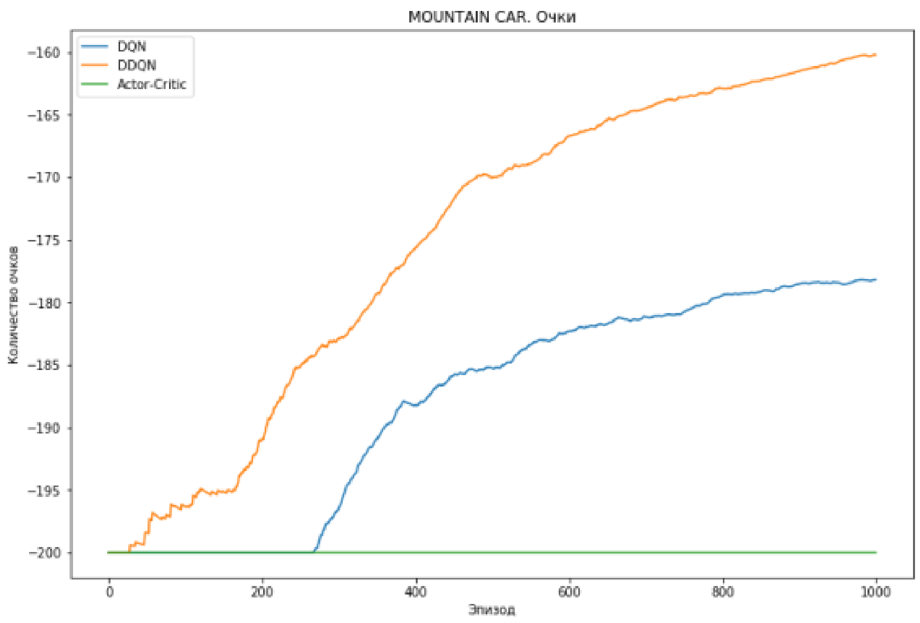

Figure 2: Mountain Car Challenge. Average points scored per episode

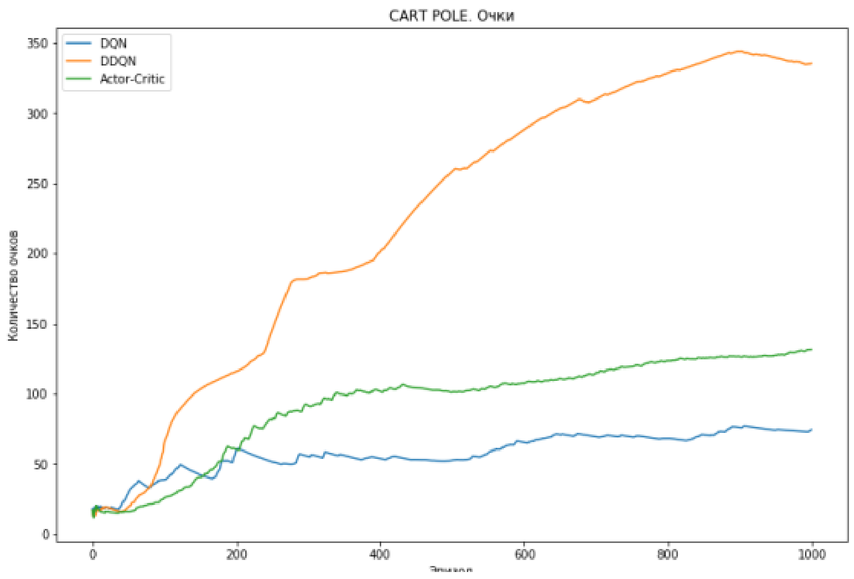

Figure 3: Cart Pole challenge. Average points scored per episode

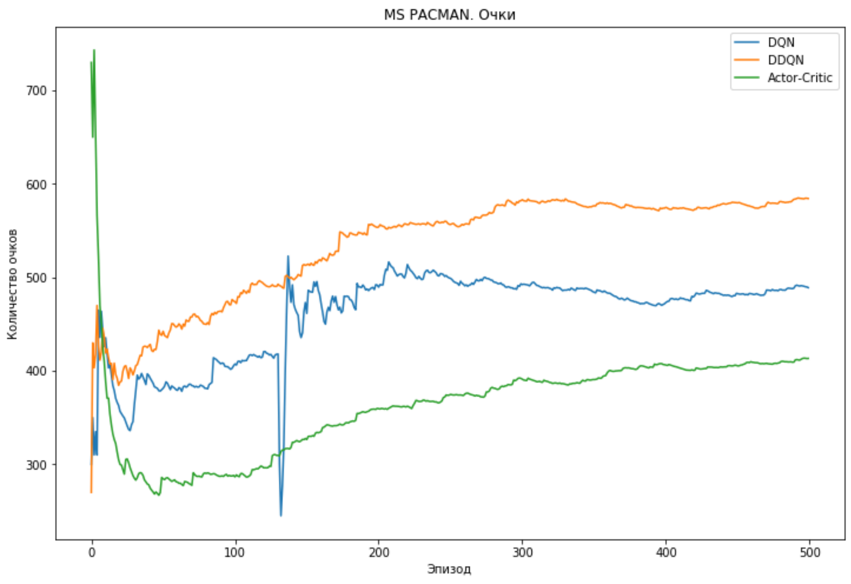

Figure 4: MsPacman challenge. Average points scored per episode

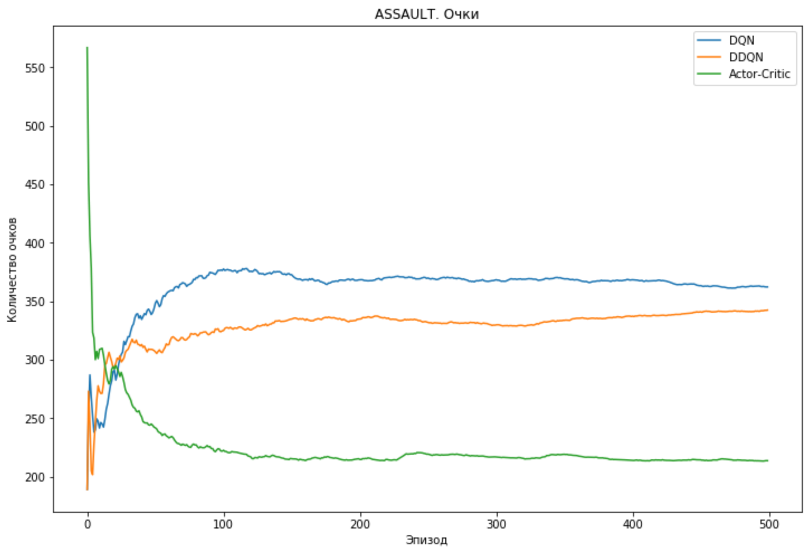

Figure 5: Assault challenge. Average points per episode

The peculiarity of the mountain car problem is that the agent does not receive punishment for remaining in the lowland. Positive reinforcement is achieved only when passing the summit. For this reason, the AC method did not cope with the task. The method is based on a gradient approach, therefore, at each iteration, it evaluates and corrects the current decision-making strategy. Since the terminal state with positive reinforcement is not reached for a long time, the strategy adjustment is minimal.

In the problem with an inverted pendulum, the AC-method showed itself better, since the timely obtained estimates helped the Actor agent to adjust the strategy for each estimate obtained.

All three methods dealt with Atari games. However, it should be noted here that both the DQN and DDQN methods scored about the same number of points, but the AC method stayed in the game a little longer on average, since the strategy adjustment did not occur in those moments when the agent's action did not bring any rewards.

Table 1
Agents learning outcomes

Algorithm	DQN	DDQN	Actor-Critic
Problem		Mountain Car	
Running time	10 min 7 sec	8 min 37 sec	8 min 48 sec
Average Score	-188.02	-175.09	-199.96
Problem		Cart Pole	
Running time	12 min 32 sec	18 min 18 sec	5 min 38 sec
Average Score	57.82	226.27	92.98
Problem		MsPacman	
Running time	41 min 25 sec	33 min 47 sec	27 min 50 sec
Average Score	466.08	533.20	363.34
Problem		Assault	
Running time	46 min 34 sec	37 min 55 sec	34 min 32 sec
Average Score	360.74	328.53	224.74

4. Conclusion

The work considers RL algorithms based on TD-methods using a neural network approach, namely DQN-, DDQN- and AC-methods, and analyzes the results of their work. It is shown that the use of ANN allows you to preserve the advantages of TD-methods, which combine the ability to learn from directly acquired experience, as well as the ability to update estimates at each step, based on other estimates, without waiting for the terminal state.

Methods using ANN make it possible to process high-dimensional continuous spaces. In the problems under consideration, the DDQN method showed better results than the DQN method. However, the DDQN method requires two ANNs, which increases both the time required for training and the amount of resources required to apply it.

For problems with a varied distribution of rewards, attention should also be paid to approaches using strategy gradients, since they use any experience gained from the agent's interaction with the environment and actions that are not related to the chosen strategy do not introduce significant shifts in the agent's trajectory.

Many problems, for the solution of which the IRTS and IRTDSS are used, are quite simply formulated in terms of "state-action-transition-reward", even if it is impossible to obtain a description of all probabilistic states of the environment. The considered methods can be used quite simply in such ISs, moreover, with sufficiently strict time constraints, since they do not require a large amount of computation to make a decision on the subsequent action and obtain a new state of the environment. However, online learning is acceptable for IRTS, as a rule, in the case of discrete tasks, and for continuous tasks, only if the computer power is high enough to correct the ANN weights in the allowable time.

At present, an integrated environment is being developed for the IRTDSS, which includes the considered reinforcement learning methods and flexible (anytime) algorithms for finding a solution. [5].

5. Acknowledgements

The work is funded by the RFBR (projects No. 18-01-00459 A, No. 20-07-00498 A).

6. References

[1] R. S. Sutton, A. G. Barto, Reinforcement learning, BINOM, Moscow, 2017.

[2] A. P. Eremeev, A. G. Gerasimova, A. A. Kozhuhov, Comparative analysis of reinforcement machine learning methods as applied to real-time systems, in: Proceedings of the International Scientific and Technical Congress “Intelligent Systems and Information Technologies 2019” “IS & IT-2019”. Scientific edition in 2 volumes. Taganrog: Publishing house of S.A. Stupin, 1 (2019).

[3] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, An Introduction to Deep Reinforcement Learning, in: Foundations and Trends in Machine Learning, 11(3-4) (2018).

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2019. URL: https://arxiv.org/abs/1509.02971.

[5] A. P. Eremeev, I. A. Poliushkin, N. A. Paniavin, Software Environment for Studying Intelligent Anytime Heuristic Search Methods, 2020 V International Conference on Information Technologies in Engineering Education (Inforino), Moscow, Russia, 2020, 1-4, doi: 10.1109/Inforino48376.2020.9111744.