# Comparative Evaluation for Effectiveness Analysis of Policy Based Deep Reinforcement Learning Approaches

Ziya Tan
Erzincan Binali Yıldırım University
Erzincan, Turkey
*Email: ziyatan [AT] erzincan.edu.tr*

Mehmet Karaköse
Fırat University
Elazığ, Turkey
*Email: mkarakose [AT] firat.edu.tr*

*Abstract*— **Deep Reinforcement Learning (DRL) has proven to be a very strong technique with results in various applications in recent years. Especially the achievements in the studies in the field of robotics show that much more progress will be made in this field. Undoubtedly, policy choices and parameter settings play an active role in the success of DRL. In this study, an analysis has been made on the policies used by examining the DRL studies conducted in recent years. Policies used in the literature are grouped under three different headings: value-based, policy-based and actor-critic. However, the problem of moving a common target using Newton's law of motion of collaborative agents is presented. Trainings are carried out in a frictionless environment with two agents and one object using four different policies. Agents try to force an object in the environment by colliding it and try to move it out of the area it is in. Two-dimensional surface is used during the training phase. As a result of the training, each policy is reported separately and its success is observed. Test results are discussed in section 5. Thus, policies are tested together with an application by providing information about the policies used in deep reinforcement learning approaches.**

***Keywords-Deep Reinforcement Learning; Deep Learning; Multi Agent***

## I. INTRODUCTION

Artificial intelligence helps us in many areas of our lives. It will be at the forefront in the future, especially with the developments in industry and health. Reinforcement learning(RL) is one of the popular algorithms in the field of artificial intelligence. With the combination of RL and deep learning(DL), a deep reinforcement learning approach has emerged. Due to the unsuccessful results of reinforcement learning in continuous environments, it has led researchers to a deep reinforcement learning approach. The policies used undoubtedly have an effect on the success of deep reinforcement learning approaches. Training will be successful with the policy used, taking into account factors such as the training environment, the problem presented, the qualifications of the agents, and the complexity of the target. We can group policies into three main categories as shown Fig.1. These are value-based, actor-critic and policy-based policies.

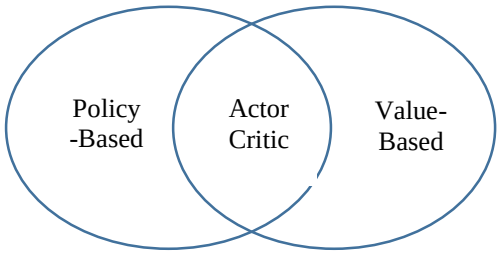


Fig. 1. Categories of policy in Deep Reinforcement Learning

In Policy-Based RL, the policy is randomly selected at the beginning and the value function of that policy is in the evaluation step. Then the new policy is found from the value function calculated in the optimization step. The process repeats until you find the most suitable policy. In this method, the policy is updated directly.

In value-based policies, the random value function is initially selected and then the new value function is calculated. This process is repeated until you find the optimum value function. The aim here is that the policy that follows the optimal value function is the optimal policy. This policy is updated indirectly through the value function.

Actor-critic policies are the approach that combines iterative learning methods used in value-based and policy-based methods. Also, actor-critic is accepted as the intersection of these two methods.

We present an application to compare the policies used in the deep reinforcement learning approach. In this application, the two agents try to get another object out of the circle it is in in a frictionless environment. Agents strike the object and apply a force. This force occurs according to Newton's laws of motion. Using four different policy algorithms (PPO, AC, DQN, PG), the training of two homogeneous agents was carried out in the same environment.

Deep reinforcement learning is a synthesis of deep learning and reinforcement learning methods. The policies chosen for use in the DRL directly affect the success of the system. An incorrectly chosen policy will cause the success rate to drop

significantly. From this point of view, the success of the policies used in different studies has been examined and the artificial neural networks used together are presented in a table. Fig. 2 shows the architecture of the DRL algorithm.

Some studies related to deep reinforcement learning; The

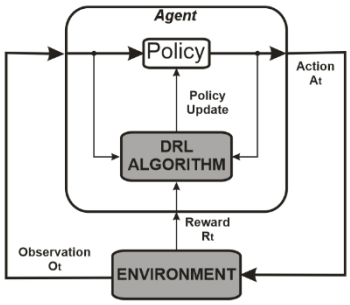


Fig. 2. Deep Reinforcement Learning architecture

DRL approach presented for the autonomous driving problem [1], the CNN-based data enhancement technique [2], the study investigating biological swarm behavior techniques [3], and the study that regulates the most accurate broadcast band to establish communication between vehicles using DRL are presented [4]. An end-to-end policy has been developed for navigating in a crowded environment using Proximal Policy Optimization (PPO) [5]. The comparative training results of LSTM, RNN and CNN algorithms with DQN in an environment with fixed obstacles are presented [6]. One of the most impressive results in reinforcement learning is DeepMind [7], where an agent performs superhuman by observing only screen pixels. They also developed an algorithm that succeeded in Go [8] competition for the first time. more specifically, it has been used to learn policies in problems such as object detection [9], captioning [10] or activity recognition [11] and image classification [12].

As a result of the articles examined, comparative results of the methods preferred in the problems discussed in these articles are presented. The policies used in different environments and problems have been examined. In addition, it is discussed why these policies are chosen. Whether the observation space is continuous or discrete is an important factor in policy choices. It is clear that it will contribute to DRL and policies, since there are few similar studies in the literature.

In section 2 provides information on deep reinforcement learning and policies. Section 3 discusses policy analysis and results. In chapter 4, the application environment in which collaborative agents are trained and the algorithms used are presented. Finally, in section 5, the result is given. Here, we summarize our major contributions as follows.

- To the best of our knowledge, this study examined recent studies of deep reinforcement learning in the literature. In addition, it offers a collaborative deep RL solution for the multi-agent problem.

- For the proposed problem, 4 different policies were trained separately and the results were reported.

-The use of deep reinforcement learning algorithms in which problems was explained with the analysis made. Also, the policies used with DRL are explained in detail.

## II. Deep Reinforcement Learning Algorithms Coupled With Policies

Neural networks play a major role in approximating the optimal value functions of reinforcement learning algorithms. For this reason, in many problems, especially in the field of robotics, artificial neural networks are used with reinforcement learning algorithms. In this section, DRL algorithms and policies used are presented in detail. In this context, there is a terminology used to describe the components of a DRL environment:

*Agent:* The decision-maker to train.

*Environment:* The general setting where the agents learn and decide what action to take.

*Action ($a$) :* One among the set of possible actions the agent can perform

*State ($s$):* Condition that the agent is in

*Reward ($r$):* The gain or loss the agent receives from the environment because of its own action

*Policy ($\pi$):* The strategy that the agent chooses to pursue. It represents a mapping between the set of situations and the set of possible actions.

*Off-policy:* Policy have an experience replay memory, so the agent can learn from previous data.

*On-policy:* The agents only learn about new data or observations.

*Model free:* It means that the agent receives data directly from the environment rather than making its own guess about the environment.

Machine learning is an area of artificial intelligence that has been developed since the 1960s. In machine learning, experience is gained from previous actions to increase success in solving a problem. Machine learning is examined under three main headings. These; supervised learning, unsupervised learning and reinforcement learning. The main goal in supervised learning is to make an inference from labeled data, in unsupervised learning, results are obtained from techniques such as prediction or clustering using unlabeled data. Reinforcement learning learns to improve his performance by interacting with his environment and using the reward-penalty system [13]. The agent chooses an action for each possible situation and completes its action. An agent maximizes the rewards he receives by repeating highly rewarding actions. In this process, it should not choose for new actions to discover the right actions. One of the most basic examples of strategies used to manage this decision is the ε-greedy [14] approach. The main goal in this strategy is for the agent to investigate the environment as much as possible in the first episodes, and

prefer exploitation to more than gradual exploration in the next steps. Q-function' defined as:(1)

$$Q_{\pi}(s,a)=E_{\pi}\{\sum_{k=0}^{H-1}\gamma^{k}\, r_{k+1}|s_0=s,a_0=a\} \quad (1)$$

In the RL problems, an agent interacts with environment represented as a series of sϵS states. The agent selects an action (at) from the action space (A) at each step ($s_t$) and receives a reward ($r_t$) for this action, repeating this action continuously in next states ($s_t$ + 1). The agent's goal is to maximize future cumulative rewards. Accordingly, the agent tries to improve its policy (π (a | s)) in which he chooses his actions to find out the most appropriate policy (π). One of the differential feature of reinforcement learning is the use of the reward signal to formulate a goal. With the reward it receives, the agent is not informed about what its next action will be, the ultimate goal is to maximize the total amount of reward in the long time. That is, the reward represents the signal that will lead to the final goal.

Formally, Markov Decision Processes (MDP) are sequential decision-making processes in which actions affect not only the next reward but also the next states. MDP is a discrete stochastic control process. Each problem that the agent aims to solve can be thought as a sequence of states (S1, S2, S3…) as shown eq.(2). Current state means transition s for the next state, it can occur only with a certain possibility.

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, S_3, \dots, S_t] \quad (2)$$

The actions of reinforcement learning algorithms are based on probability distributions of Q values. Real world problems have continuous or very large discrete state areas. In this case, it seems that every possible state is not taken into account. In DRL, neural networks are used as a nonlinear function approximation to overcome this problem. Especially in recent years, outstanding works have been achieved in studies in the field of deep learning (language processing, image processing, etc.). These developments have led to an increased interest in combining RL with neural networks. In DRL, neural networks take situations as input and output the possible action. Neural network architecture in DRL consists of at least two components. These; different number of hidden layers and action layers (pooling layer, fully connected layer, etc.) as shown Fig.3.

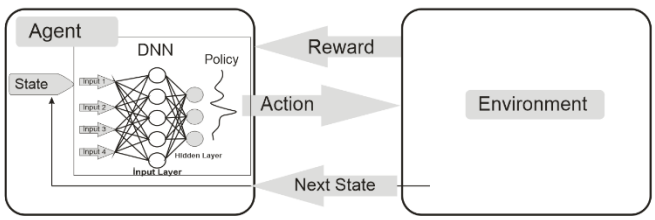


Fig. 3. The interaction of the environment with the Deep Reinforcement Learning

Model-free algorithms learn a policy or value function without making a prediction for the next state or reward. In model-based algorithms, the agent learns a model to select an action. These algorithms have been particularly successful in robotics.

Correct use of policies is very important for reinforcement learning. Therefore, it is necessary to pay attention to the nature of the problem in policy selection. Choosing the right neural network with policy will increase the success of training. In this section, information is given about the policies used in reinforcement learning. Table 1 shows the main characteristics of the policy types used in the DRL.

TABLE 1. MAIN CHARACTERISTIC OF THE POLICY IN DRL

| Policy | Method | Type | Action Space |
|---|---|---|---|
| Q-Learning | Off-policy | Value-Based | Discrete |
| DQN | Off-policy | Value-Based | Discrete |
| Sarsa | On-policy | Value-Based | Discrete |
| PG(Monte Carlo) | On-policy | Policy-Based | Discrete or Continuous |
| AC | On-policy | Actor-Critic | Discrete or Continuous |
| PPO | On-policy | Policy-Based | Discrete or Continuous |
| DDPG | Off-policy | Policy-Based | Continuous |
| TD3 | Off-policy | Policy-Based | Continuous |
| Soft AC | Off-policy | Actor-Critic | Continuous |

We can examine the policy algorithms used in reinforcement learning studies in the literature in three different methods. The training algorithms of the policies presented in this article are presented in the appendix.

### *A. Value-Based Methods*

In this method, the value function is used to show the success of the state and by learning the optimal value function, the optimal policy is learned. Deep-Q learning, Q-learning and Sarsa algorithms can be given as examples of value-based method.

#### *1) The Q-Learning algorithm*

Q-Learning [13] is a model free and off policy RL algorithm. Also, the observation space can be continuous or discrete. The action space is discrete. During the training phase, the agent explores the field of action using ϵ -greedy exploration. At each step it chooses a random action based on the ϵ value, otherwise it performs a greedy action according to the 1-ϵ function. To predict the value function, a Q-learning agent uses neural network Q (S, A), which is a table or function approximation. This network takes S (observation) and A (action) as input and returns long-term expectation of rewards as output. Once the training is complete, the trained value-function approximation Q (S, A) is also stored.

#### *2) Deep Q-Network (DQN)*

DQN is a model free and off policy DRL algorithm. The observation space can be discrete or continuous, while the action space is discrete. DQN trains a network to predict future rewards. During the training, neural network parameters are updated at each step, the training area is explored using ϵ-greedy, this action is repeated according to the determined

discount factor ratio, previous experiences are stored in the experience buffer, the neural network is updated according to the mini-batch experience randomly sampled from the buffer. Calculates the two function approximations to estimate the DQN value function.

*Q (S, A):* Observation (S) and action (A) are taken as input, and the value corresponding to long-term reward is output.

*Target Q '(S, A):* The neural network is periodically updated according to the parameter settings to stabilize the best result.

Both functions have the same structure and parameters.

*3) SARSA*

Sarsa is a model free and on policy RL algorithm [15]. The Sarsa can be trained as an observation space in discrete or continuous environments. It can be trained in discrete environments as an action space. During the training, the agent does the explore process using ϵ-greedy. To estimate the value function, a Q- table or function approximation Q (S, A) is calculated. After the training is completed, the trained value function is save in the network approximation Q (S, A).

*B. Policy-based Methods*

One of the main differences between value-based and policy-based methods is the methods they use during decision-making [11]. However, while value-based methods evaluate the best overall reward, policy-based methods focus on finding the most appropriate policy for training. Algorithms that implement a policy that decides which action to choose in each training step are called policy-based algorithms. Policy-based algorithms are more appropriate to be applied in stochastic environments or environments with high-dimensional actions, as they can represent continuous actions.

*1) Policy Gradient (PG)*

PG is a on policy and model free RL algorithm. It is also a policy-based algorithm that directly calculates the optimal policy that maximizes long-term reward. Agents trained with the PG algorithm can be trained in environments with discrete or continuous observations and actions space. During the training, it predicts the probabilities of realization of each action in the action environment and randomly chooses the actions according to the possibility distribution. It completes the first training episode without gaining any experience and updating policy parameters.

*2) Proximal Policy Optimization (PPO)*

PPO [5] is a model free and on policy RL algorithm. This algorithm is a type of PG training that uses stochastic gradient descent to sample data through environmental interaction. Agents trained with the PPO algorithm can be trained in environments with a discrete or continuous observation space and action space. During the training of the PPO agent, the possibilities of each action in the action space are estimated and randomly selects actions based on their possibility distribution. Also, many episode interact with the environment using the current policy before using mini-batch to update the parameters of neural networks. It uses two function approximations to predict the policy and value function:

*Actor μ(S):* The first network (actor) takes the S (observation) and returns the rates at which each action is performed in the action field.

*Critic V(S):* The second network(critic) takes, the S (observation), and returns its expectation corresponding to discounted long-term reward.

After the training is completed, the optimal policy parameters trained are stored in μ (S).

*3) Deep Deterministic PG (DDPG)*

The DDPG [16] is a model free and off policy RL algorithm. A DDPG algorithm uses an actor-critic framework that calculates the optimal policy, making the long-term reward the greatest. DDPG can be trained in continuous or discrete observation space. It can also be trained in the continuous action space. During the training phase, the actor-critic properties are updated in each learning step.

Past experiences are stored by the experience buffer. The algorithm updates the actor-critic, using experience as much as the mini-batch that it randomly selects from the buffer. DDPG uses 4 different function approximations to calculate the policy and value function [17].

*Actor μ(S):* The actor takes the S (observation) and performs an action to maximize the long-term reward.

*Target Actor μ'(S):* Periodically updates the target actor to improve the optimization stability of the agent.

*Critic Q(S, A):* Critic takes the S (observation) and A (action) as input and returns its value corresponding to the long-term reward.

*Target Critic Q'(S, A):* The agent updates target critic periodically to optimize its stability.

At the end of the training, the trained optimum policy actor is stored as μ (S).

*4) Twin-Delayed DDPG (TD3)*

TD3 is a model free and off policy RL algorithm. This learning algorithm adopts a method for reduce the overestimation in function approximation. This method is similar to the one implemented in the DDPG algorithm. It learns two different Q-value functions. Also TD3 prefer the minimum value function guess during policy updates [18].

The TD3 algorithm can be trained in environments with continuous or discrete observation space and continuous action space. A TD3 algorithm during the training, Updates the actor and critic parameters at each time step during learning. The agent use mini-batch of experience for updates the actor-critic.

*C. Actor-Critic Based Methods*

This methods are mixed methods that combine the benefits of policy based and value based approaches. The actor is responsible for choosing actions [19]. This evaluation

determines whether the expected situation is worse or better than the chosen action. Makes gradient-based learning in both networks. If $J(\theta):=E\pi\theta[r]$ equation represents a policy then $\theta$ is a DNN parameter. Since improvement can be costly and slow in continuous action environments, the DPG (Deterministic Policy Gradient) algorithm represents actions by parameterizing them as in the equation $\mu(s|\theta^{\mu})$ [17].

*1) Actor-Critic (AC)*

AC is a model free and on policy RL algorithm [20]. The purpose of agents using the AC algorithm is to train the actor to directly optimize and calculate critic future rewards. In addition, AC agents can be trained in environments that have an observation and action space that are continuous and discrete.

During training, an AC agent predicts their probability of performing every action in the action area. It then randomly chooses actions based on their probability distribution. The AC agents interact with the environment for multiple steps using the existing policy before updating the actor and critic parameters.

To prediction the policy function and value function, an AC agent uses two function approximators:

*Actor μ(S)*: First one is actor algorithm. It takes S (observation) and returns the probabilities of taking each action in the action space when in state S.

*Critic V(S)*: Second one is critic algorithm. It takes S (observation) and returns the corresponding anticipation of the discounted long-term reward.

The trained optimal policy is stored in μ(S) after the training.

## III. Policy Analysis Study

In the previous section, the policies frequently used in the literature are explained. In this section, methods used in articles on deep reinforcement learning published in recent years are analyzed. In these article, artificial neural networks and policies used especially in the applications are examined. Table 2 shows the types of problems of the studied article, which policy they prefer and which neural network they use.

When Table 2 is examined, it is seen that policy-based algorithms are generally preferred for image processing-based problems, and value-based algorithms are preferred for signal processing-based problems. Actor-Critic based policies are preferred in more complex continues environments.

Furthermore, preferred policies differ in problems where discrete and continuous environments are used. It has been observed that more value-based policies are used in discrete environments and actor-critic based policies are used in continuous environments.

In summary, LSTM has been successful as an artificial neural network in areas such as trading, speech recognition and autonomous driving. The memory usage strategy of the LSTM architecture has increased its success in these areas. In addition, the use of Q-learning algorithms with LSTM is preferred in such problems. The success of the DDPG policy algorithm in multi-agent environments is clear.

TABLE 2. Type of problems in articles

| Type of Problem | Used Policy | Used NN |
|---|---|---|
| Autonomous Driving [1] [21] [6] [22] | Sarsa, Q-Learning, DDPG | CNN, LSTM, RNN |
| Data Augmentation. [2] [23] | AC,PPO | CNN |
| Chip Placement [5] | PPO | CNN |
| Action Controller for Multiplayer Games [24] | Dual PPO | LSTM |
| Learning for Trading [25] [26] | DQN,PG,AC | LSTM, CNN |
| Energy Consumption Forecasting [27] | A3C,DDPG,RDPG | LSTM |
| Wind Speed Short Term Forecasting [28] | Q-Learning | LSTM |
| İmage Segmentation [29] [30] | Q-Learning, DDPG | CNN |
| Policies for Multi-Agent Control [31] [32] [33] [34] [35] | PG,DDPG,DQN | CNN |
| Intelligent System [36] [37] | Q-Learning, DDPG | LSTM,CNN |
| Minimalistic Attacks [38] [39] [40] | DQN,PPO,AC, DDPG,PG | CNN |
| New Policy Method [41] [42] | PPO | CNN |
| Learning of Speech [43] [44] [45] | Q-Learning, PPO | CNN,LSTM |
| Moving Obstacle Avoidance [46], [3] | PG, AC, PPO | CNN |
| Learning with Robust and Smooth Policy [47] | TRPO, DDPG | CNN |
| Object Detection in Large Images [48] [49] [50] [51] [52] | PG,DQN | CNN |
| Automatic Landing Control [53] [54] [55] | DDPG,AC,DQN | CNN |
| Cooperative Internet of UAVs [56] | AC | CNN |

## IV. Collaborative Multi-Agent Simulation

### A. Simulation Environment

By using deep reinforcement learning, successful studies have been carried out in multi-agent problems as well as in single agent problems. In this study, the interaction of two collaborative agents with a common goal is presented. The purpose of agents is to get an object out of its location as quickly as possible. Accordingly, agents move by applying force to the object in cooperation.


Fig. 4. Training and test environment

The results were observed by training the agents with 4 different algorithms. The 4 algorithms used (PPO, AC, DQN, PG) are explained in detail in Chapter 2. Fig. 4 shows the training area. The red circle represents agent A, the green circle represents agent B, and the blue circle represents the target object.

For this environment:

The 2-dimensional space is bounded from –12 m to 12 m in both the X and Y directions.

The contact spring stiffness and damping values are 100 N/m and 0.1 N/m/s, respectively.

The agents share the same observations for positions, velocities of A, B, and C and the action values from the last time step.

The simulation terminates when target object moves outside the circular ring.

At each time step, the agents receive the following reward:

$$r_A = r_{global} + r_{local,A}$$

$$r_B = r_{global} + r_{local,B}$$

$$r_{global} = 0.001dc$$

$$r_{local,A} = -0.005d_{AC} - 0.008u^2A$$

$$r_{local,B} = -0.005d_{BC} - 0.008u^2B$$

Here:
$r_A$ and $r_B$ are the rewards received by agents A and B, respectively.

$r_{global}$ is a team reward that is received by both agents as object C moves closer towards the boundary of the ring.

$r_{local,A}$ and $r_{local,B}$ are local penalties received by agents A and B based on their distances from object C and the magnitude of the action from the last time step.

$d_C$ is the distance of object C from the center of the ring.

$d_{AC}$ and $d_{BC}$are the distances between agent A and object C and agent B and object C, respectively.

$u_A$ and $u_B$ are the action values of agents A and B from the last time step.

Fig. 5 shows the working principle of the training simulation. Simulations were run on an Intel(R) Core(TM) i5-7200U CPU with 2.70 GHz clock rate and 8 GB of RAM.

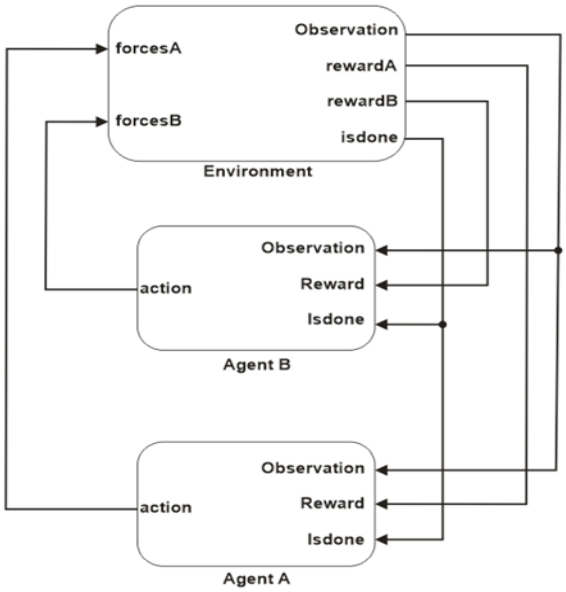


Fig.5. The working principle of training simulation

## B. *Experiment Results*

The parameters used in the experiments are determined the same. The training options specified in Table 3 are set to train the agents. The training is ran a maximum of 800 episodes with a maximum of 5000 time steps for each segment. Training is stopped when the average reward for 100 consecutive episodes is -10 or more. These experiments uses different policy agents with discrete action spaces.

TABLE 3. PARAMETERS OF TRAINING ENVIRONMENT

| Parameter | Value |
|---|---|
| Max Episodes | 800 |
| Time Steps Per Episode | 5000 |
| Score Averaging Window Length | 100 |
| Stop Training Value | -10 |

### 1) *Experiment I*

The PPO algorithm is used in the first experiment. PPO algorithms support actor and critic that use recurrent DNN as functions approximators. The training results of the two different agents are given in Fig.6 and 7.

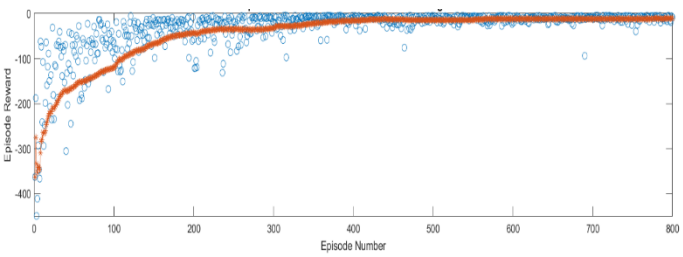


Fig. 6. Episode reward for PPO with agent A

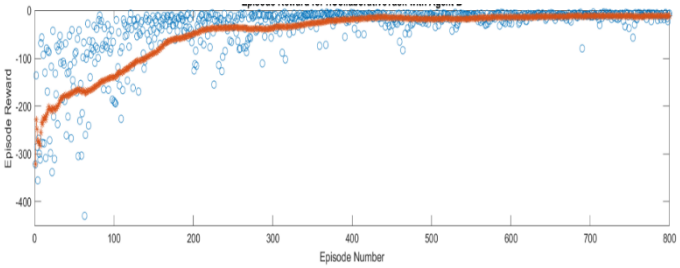


Fig. 7. Episode reward for PPO with agent B

Examining the training results, it is seen that the agents reached the ideal success point during the 800-episode training. The training lasted approximately 4 hours and 50 minutes. At the end of the training, agent A's average reward was -10.68, and agent B's average reward was -11.18.

*2) Experiment II*

The PG algorithm is used in the second experiment. PG algorithms use the REINFORCE algorithm either with or without a baseline. The training results of the two different agents are given in Fig.8 and 9. When the figures in Experiment 2 were examined, it is seen that PG agents could not learn regularly. One of the reasons for this may be that the maximum training episode is not sufficient. Experiments can be done to solve this problem in the next study. The training lasted approximately 9 hours . At the end of the training, agent A's average reward was -228.57, and agent B's average reward was -193.47.

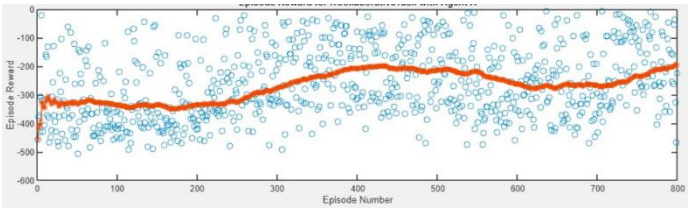


Fig.8. Episode reward for PG with agent A

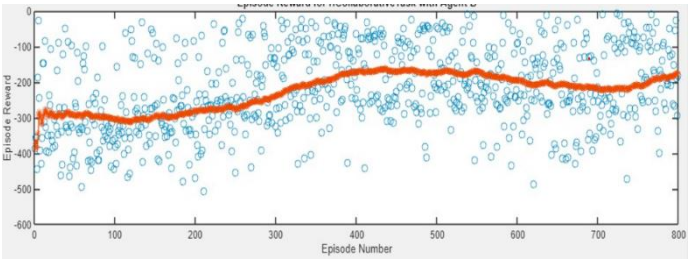


Fig.9. Episode reward for PG with agent B

*3) Experiment III*

The DQN algorithm is used in the third experiment. DQN algorithms support critic that use recurrent DNN as functions approximators. In the DQN algorithm, unlike the others, the average reward value was changed from -10 to +10 in order to terminate the training. This is because DQN trains a single network and the reward average may be higher in the first episodes than others. The training results of the two different agents are given in Fig.10 and 11. When the figures were examined, it is seen that the DQN algorithm has successfully completed the 800-part training. Continues as soon as possible and with maximum reward, especially after the 400th episode. The training lasted approximately 2 hours. At the end of the training, agent A's average reward was -3.65, and agent B's average reward was -1.7.

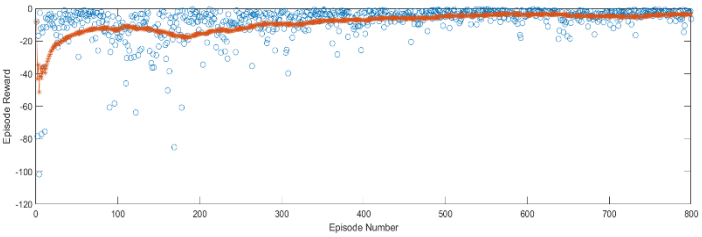


Fig. 10. Episode reward for DQN with agent A

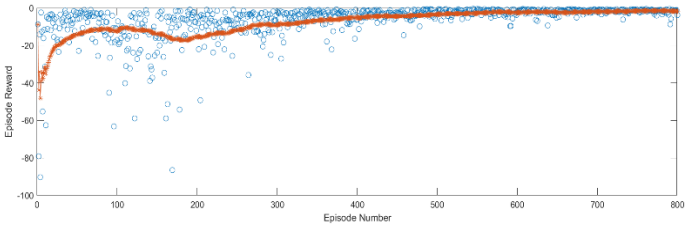


Fig. 11. Episode reward for DQN with agent B

*4) Experiment IV*

The AC algorithm is used in the third experiment. Agents trained with AC soon reached the ideal average reward line. The training lasted 2 hours and 20 minutes in total. In the AC algorithm, unlike the others, the average reward value was changed from -10 to +10 in order to terminate the training. The reason for this is to ensure that the training lasts 800 episodes. At the end of the training, agent A's average reward was -7.2, and agent B's average reward was -7.3. The training results of the two different agents are given in Fig.12 and 13.

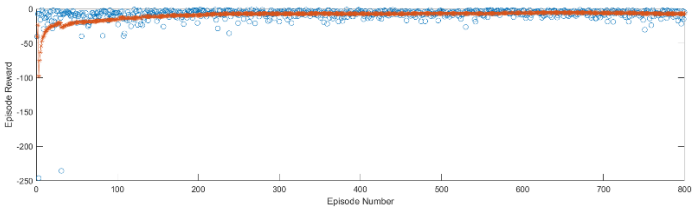


Fig.12. Episode reward for AC with agent A

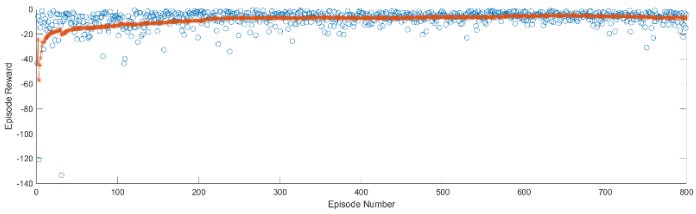


Fig.13. Episode reward for AC with agent B

### *C. Test Results of Collaborative Task problem*

According to the test results indicated in the Fig.14 the DQN agent was able to remove the blue object from the circle it was in as quickly as possible. In addition, DQN completed its training in a shorter time compared to other algorithms during the training phase. PG algorithm was not successful at the end of 800 episodes of training and could not complete its task in 50 seconds test period. PPO and AC algorithms have completed their tasks in about a period of time. The biggest feature that distinguishes DQN algorithm from others is that it is value based.

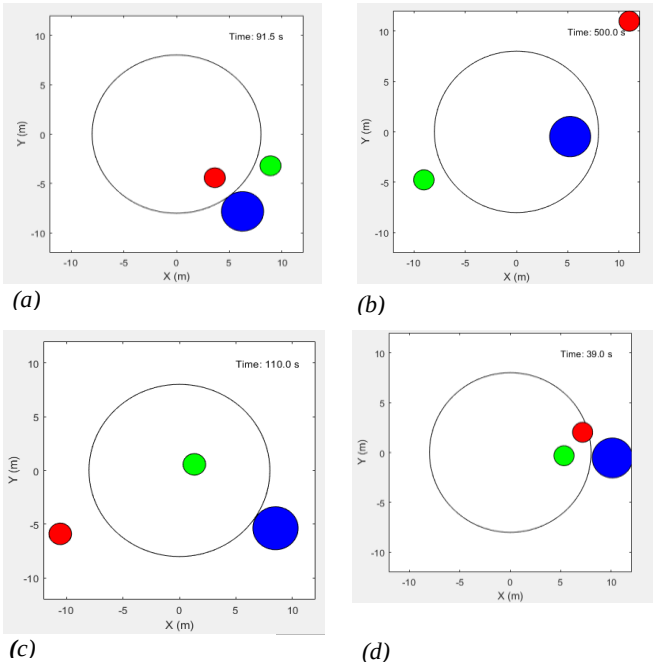


Fig.14. Test result of algorithms after training
( a- PPO, b- PG, c- AC, d-DQN )

All DRL algorithms proposed in Table 4 include model-free. So, none of the above are trying to estimate the objective function. Alternatively, these algorithms update their knowledge based on heuristic approach. The difference of DQN algorithm from other algorithms is that it is off-policy. Due to this difference, it has been more successful than other algorithms. PPO has improved its performance by changing the target function to reduce the complexity of implementation and computing. PPO agents get rid of the computation created by forced optimization as it suggests a clipped surrogate objective function.

TABLE 4.COMPARATIVE PROPERTIES OF THE ALGORITHMS USED IN THE EXPERIMENTS

| Policy | Method | Type | Model | Action Space | Observation Space |
|---|---|---|---|---|---|
| DQN | Off-policy | Value-Based | Model-free | Discrete | Continuous |
| PG(Monte Carlo) | On-policy | Policy-Based | Model-free | Discrete or Continuous | Continuous |
| AC | On-policy | Actor-Critic | Model-free | Discrete or Continuous | Continuous |
| PPO | On-policy | Policy-Based | Model-free | Discrete or Continuous | Continuous |

## V. CONCLUSION

Artificial intelligence is a field of study that aims to understand intelligence and create intelligent entities. The fastest progress in this area has been provided by the studies in the field of machine learning. In the studies conducted in the field of machine learning, reinforcement learning has been maintaining its popularity in recent years. With the technological developments in robotic and industry, artificial intelligence will be in our lives for many years.

Two different studies are presented in this article. Both studies emphasize the importance of policy choices in deep reinforced learning problems. First, DRL studies published in recent years have been examined. In these studies, what kind of problems are used and which policies are preferred for these problems have been investigated. According to researches, policy choices are directly related to the training environment and the problem. The importance of choosing the right policy for a successful outcome has emerged. It is seen that DDPG algorithms are used and successful, especially in the robotic field. Although policy choice is important, the choice of artificial neural networks used in deep reinforced learning is also important. It has been determined that LSTM architecture is used in language processing problems and CNN architecture is used in image processing problems. In future studies, especially the development of hybrid systems will increase the success in solving many problems.

In the second study, two different agents acting with a collaborative approach were trained with four different policy algorithms and their results were compared. As seen in the test results, the agent trained with the DQN algorithm successfully completed its task in 39 seconds, the agent trained with the ppo algorithm in 915 seconds, and the agent trained with the Ac algorithm in 111 seconds. However, the agent trained with the PG algorithm could not complete the task in the specified time. The fact that this problem is in discrete action space has been effective in the chosen policies. Accordingly, the DQN algorithm has been more successful than the others. Since he only needed to train one network, the training time was shorter than the others.

## REFERENCES

[1] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani and P. Pérez, "Deep Reinforcement Learning for Autonomous Driving: A Survey," arXiv:2002.00444, 2020.

[2] I. Kostrikov, D. Yarats and R. Fergus, "Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels," arXiv:2004.13649, 2020.

[3] T. Fan, P. Long, W. Liu and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," The International Journal of Robotics Research, vol. 39, no. 7, pp. 856-892, 2020.

[4] Z. Cao, H. Guo, W. Song, K. Gao, Z. Chen, L. Zhang ve X. Zhang, «Using Reinforcement Learning to Minimize the Probability of Delay Occurrence in Transportation,» IEEE Transactions on Vehicular Technology, cilt 69, no. 3, pp. 2424-2436, 2020.

[5] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, A. Babu, Q. V. Le and J. La, "Chip Placement with Deep Reinforcement Learning," arXiv:2004.10746, 2020.

[6] Z. Tan and M. Karaköse, "Comparative Study for Deep Reinforcement Learning with CNN, RNN, and LSTM in Autonomous Navigation," 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI), 2020.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-level control through deep reinforcement learning," nature, pp. 529-533, 2015.

[8] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre and V. Den, "Mastering the game of Go with deep neural networks and tree search," nature, vol. 529, p. 484, 2016.

[9] J. C. Caicedo and S. Lazebnik, "Active Object Localization With Deep Reinforcement Learning," Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 2488-2496, 2015.

[10] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention.," In International Conference on Machine Learning, pp. 2048-2057, 2015.

[11] S. Yeung, O. Russakovsky, G. Mori and L. Fei-Fei, "End-to-end learning of action detection from frame glimpses in videos," In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2678-2687, 2015.

[12] A. Maldonado-Ramirez, R. Rios-Cabrera and I. Lopez-Juarez, "A visual path-following learning approach for industrial robots using DRL," Robotics and Computer-Integrated Manufacturing, vol. 71, 2021.

[13] C. J. Watkins and P. Dayan, "Q-Learning," Machine Learning, vol. 3, no. 8, pp. 279-292, 1992.

[14] Z. Tan and M. Karaköse, "Optimized Deep Reinforcement Learning Approach for Dynamic System," 2020 IEEE International Symposium on Systems Engineering (ISSE), pp. 1-4, 2020.

[15] D. Zhao, H. Wang, K. Shao and Y. Zhu, "Deep reinforcement learning with experience replay based on SARSA," 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1-6, 2016.

[16] S. Fujimoto, H. v. Hoof and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," arXiv:1802.09477, 2018.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning.," arXiv preprint arXiv:1509.02971, 2015.

[18] Q. Shi, H.-K. Lam, C. Xuan and M. Chen, "Adaptive neuro-fuzzy PID controller based on twin delayed deep deterministic policy gradient algorithm," Neurocomputing, vol. 402, pp. 183-194, 2020.

[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine Learning, pp. 229-256, 1992.

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P, Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," In International conference on machine learning , pp. 1928-1937, 2016.

[21] S. Aradi, "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles," IEEE Transactions on Intelligent Transportation Systems, pp. 1-20, 2020.

[22] C. Chen, H. Wei, N. Xu, G. Zheng, M. Yang, Y. Xiong, K. Xu and Z. Li, "Toward A Thousand Lights: Decentralized Deep Reinforcement Learning for Large-Scale Traffic Signal Control," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 4, pp. 3414-3421, 2020.

[23] R. Raileanu, M. Goldstein, D. Yarats, I. Kostrikov and R. Fergus, "Automatic Data Augmentation for Generalization in Deep Reinforcement Learning," arXiv:2006.12862, 2020.

[24] W. Liang, W. Huang, J. Long, K. Zhang, K.-C. Li ve D. Zhang, «Deep Reinforcement Learning for Resource Protection and Real-Time Detection in IoT Environment,» IEEE Internet of Things Journal, cilt 7, no. 7, pp. 6392 - 6401, 2020.

[25] Z. Zhang, S. Zohren and S. Roberts, "Deep Reinforcement Learning for Trading," The journal of dinancial data science, vol. 2, no. 2, pp. 25-40, 2020.

[26] X. Wu, H. Chen, J. Wang, L. Troiano, V. Loi and H. Fujita, "Adaptive stock trading strategies with deep reinforcement learning methods," Information Sciences, vol. 538, pp. 142-158, 2020.

[27] T. Liu, Z. Tan, C. Xu, H. Chen and Z. Li, "Study on deep reinforcement learning techniques for building energy consumption forecasting," Energy and Buildings, vol. 208, 2020.

[28] H. Liu, C. Yu, H. Wu, Z. Duan and G. Yan, "A new hybrid ensemble deep reinforcement learning model for wind speed short term forecasting," Energy, vol. 202, 2020.

[29] N. Zeng, H. Li, Z. Wang, W. Liu, S. Liu, F. E. Alsaadi and X. Liu, "Deep-reinforcement-learning-based images segmentation for quantitative analysis of gold immunochromatographic strip," Neurocomputing, 2020.

[30] Z. Tian, X. Si, Y. Zheng, Z. Chen and X. Li, "Multi-Step Medical Image Segmentation Based on Reinforcement Learning," Journal of Ambient Intelligence and Humanized Computing, vol. 543, pp. 1-12, 2020.

[31] C. D. Hsu, H. Jeong, G. J. Pappas and P. Chaudhari, "Scalable Reinforcement Learning Policies for Multi-Agent Control," arXiv:2011.08055, 2020.

[32] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang and S. Russell, "Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, pp. 4213-4220, 2019.

[33] Z. Yan and Y. Xu, "A Multi-Agent Deep Reinforcement Learning Method for Cooperative Load Frequency Control of a Multi-Area Power System," Transactions on Power Systems, vol. 35, no. 6, pp. 4599-4608, 2020.

[34] H. Fu, H. Tang, J. Hao, Z. Lei, Y. Chen and C. Fan, "Deep Multi-Agent Reinforcement Learning with Discrete-Continuous Hybrid Action Spaces," arXiv preprint arXiv:1903.04959, 2019.

[35] W. Ding, S. Li, H. Qian and Y. Chen, "Hierarchical Reinforcement Learning Framework Towards Multi-Agent Navigation," in International Conference on Robotics and Biomimetics (ROBIO), Kuala Lumpur, Malaysia, 2019.

[36] M. Ausin, "Leveraging Deep Reinforcement Learning for Pedagogical Policy Induction in an Intelligent Tutoring System," Proceedings of the 12th International Conference on Educational Data Mining, pp. 168-177, 2019.

[37] A. Haydari and Y. Yilmaz, "Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey," IEEE Transactions on Intelligent Transportation Systems, pp. 1-22, 2020.

[38] X. Qu, Z. Sun, Y. S. Ong, A. Gupta and P. Wei, "Minimalistic Attacks: How Little it Takes to Fool Deep Reinforcement Learning Policies," IEEE Transactions on Cognitive and Developmental Systems, 2020.

[39] A. Russo and A. Proutiere, "Optimal Attacks on Reinforcement Learning Policies," arXiv:1907.13548, 2019.

[40] M. Lopez-Martin, B. Carro and A. Sanchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems," Expert Systems with Applications, vol. 141, 2020.

[41] J. H. Tianpei Yang, Z. Meng, Z. Zhang, Y. Hu, Y. Cheng, C. Fan, W. Wang, Z. W. Wulong Liu and J. Peng, "Efficient Deep Reinforcement Learning via Adaptive Policy Transfer," arXiv preprint arXiv:2002.08037, 2020.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv:1707.06347 [cs.LG], 2017.

[43] H. Cuayahuitl and S. Yu, "Deep reinforcement learning of dialogue policies with less weight updates," International Conference of the Speech Communication Association (INTERSPEECH),, 2017.

[44] V. S. Dorbala, A. Srinivasan and A. Bera, "Can a Robot Trust You? A DRL-Based Approach to Trust-Driven Human-Guided Navigation," arXiv:2011.00554, 2020.

[45] T. Rajapakshe, R. Rana, S. Latif, S. Khalifa and B. W. Schuller, "Pre-training in Deep Reinforcement Learning for Automatic Speech Recognition," arXiv:1910.11256, 2019.

[46] A. Garg, H.-T. L. Chiang, S. Sugaya, A. Faust and L. Tapia, "Comparison of Deep Reinforcement Learning Policies to Formal Methods for Moving Obstacle Avoidance," International Conference on Intelligent Robots and Systems (IROS), pp. 3534-3541, 2019.

[47] Q. Shen, Y. Li, H. Jiang, Z. Wang and T. Zhao, "Deep Reinforcement Learning with Robust and Smooth Policy," proceedings of the 37th International Conference on Machine Learning, vol. 119, pp. 8707-8718, 2020.

[48] B. Uzkent, C. Yeh and S. Ermon, "Efficient Object Detection in Large Images Using Deep Reinforcement Learning," Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), pp. 1824-1833, 2020.

[49] A. Pirinen and C. Sminchisescu, "Deep Reinforcement Learning of Region Proposal Networks for Object Detection," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6945-6954, 2018.

[50] M. B. Bueno, X. Giró-i-Nieto, F. Marqués and J. Torres, "Hierarchical Object Detection with Deep Reinforcement Learning," Deep Learning for Image Processing Applications, vol. 164, no. 3, p. 31, 2017.

[51] X. Kong, B. Xin, Y. Wang and G. Hua, "Collaborative Deep Reinforcement Learning for Joint Object Search," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1695-1704, 2017.

[52] X. L. Zequn Jie, J. Feng, X. Jin, W. F. Lu and S. Yan, "Tree-Structured Reinforcement Learning for Sequential Object Localization," arXiv preprint arXiv:1703.02710, 2017.

[53] C. Tang and Y.-C. Lai, "Deep Reinforcement Learning Automatic Landing Control of Fixed-Wing Aircraft Using Deep Deterministic Policy Gradient," 2020 International Conference on Unmanned Aircraft Systems (ICUAS), 2020.

[54] L. Cheng, F. Jiang and Z. Wang, "Real-time control for fuel-optimal Moon Landing Based on an Interactive Deep Reinforcement Learning Algorithm," Astrodynamics , vol. 3, pp. 375-386, 2019.

[55] Y. Xu, Z. Liu and X. Wang, "Monocular Vision based Autonomous Landing of Quadrotor through Deep Reinforcement Learning," in 37th Chinese Control Conference (CCC), Wuhan, 2018.

[56] J. Hu, H. Zhang, L. Song, R. Schober and H. V. Poor, "Cooperative Internet of UAVs: Distributed Trajectory Design by Multi-Agent Deep Reinforcement Learning," IEEE Transactions On Communications, vol. 68, no. 11, pp. 6807-6821, 2020.

## APPENDIX

The training algorithms of the policies presented in this article are given in this section.

- Initialize the network Q(S,A) with random values
- For each training episode:
  - Set the initial observation S.
  - Repeat the following for each step of the episode until S is a terminal state:
    - For the current observation S, select a random action A with probability ϵ. Otherwise, select the action for which the critic value function is greatest.

$$A = arg \max_{A} Q(S, A)$$

    - Execute action A. Observe the reward R and next observation S'.
    - If S' is a terminal state, set the value function target y to R. Otherwise, set it to

$$y = R + \gamma \max_{A} Q(S', A)$$

    - Compute the network parameter update.

$$\Delta Q = y - Q(S', A)$$

    - Update the network using the learning rate α.

$$Q(S, A) = Q(S, A) + \alpha * \Delta Q$$

    - Set the observation S to S'.

Fig. 15. Training algorithm of Q-Learning

- Initialize the network Q(S,A) with random parameter values $\theta_Q$, and initialize the target network with the same values: $\theta_{Q'} = \theta_Q$
- For each training time step:
  - For the current observation S, select a random action A with probability ϵ. Otherwise, select the action for which the critic value function is greatest.

$$A = arg \max_{A} Q(S, A|\theta_Q)$$

  - Execute action A. Observe the reward R and next observation S'
  - Store the experience (S,A,R,S') in the experience buffer.
  - Sample a random mini-batch of M experiences (Si, Ai,Ri,S'i) from the experience buffer.
  - If S'i is a terminal state, set the value function target yi to Ri. Otherwise, set it to;

(Double DQN)

$$A_{max} = arg \max_{A'} Q(Si', A'|\theta_Q)$$
$$y_i = R_i + \gamma Q'(Si', A_{max}|\theta_{Q'})$$

(DQN)

$$y_i = R_i + \gamma \max_{A'} Q'(Si', A'|\theta_{Q'})$$

  - Update the network parameters by one-step minimization of the loss L across all sampled experiences.

$$L = \frac{1}{M}\sum_{i=1}^{M}(y_i - Q(S_i, A_i|\theta_Q))^2$$

  - Update the target network parameters
  - Update the probability threshold ϵ

Fig. 16. Training algorithm of Deep Q- Learning

- Initialize the critic Q(S,A) with random values.
- For each training episode:
  - Set the initial observation S.
  - For the current observation S, select a random action A with probability ϵ. Otherwise, select the action for which the critic value function is greatest.

$$A = arg\max_{A} Q(S, A)$$

  - Repeat the following for each step of the episode until S is a terminal state:
    - Execute action A. Observe the reward R and next observation S'.
    - Select an action A' by following the policy from state S'.

$$A' = \max_{A'} Q(S', A')$$

    - If S' is a terminal state, set the value function target y to R. Otherwise, set it to:

$$y = R + \gamma \max_{A} Q(S', A')$$

    - Compute the critic parameter update.

$$\Delta Q = y - Q(S, A)$$

    - Update the critic using the learning rate α.

$$Q(S, A) = Q(S, A) + \alpha * \Delta Q$$

    - Set the observation S to S'.
    - Set the action A to A'.

Fig.17. Training algorithm of SARSA

- Initialize the actor μ(S) with random parameter values θμ.
- For each training episode, generate the episode experience by following actor policy μ(S). The agent takes actions until it reaches the terminal state ST. The episode experience consists of the sequence;

$$S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$$

$S_T$→State observation
$A_{T+1}$→ An action taken from that state
$S_{T+1}$→ Next
$R_{T+1}$→ reward received for moving $S_t$ to $S_{t+1}$

- For each state in the episode sequence, that is, for t = 1, 2, …, T-1, calculate the return Gt, which is the discounted future reward.

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} R_k$$

- Accumulate the gradients for the actor network by following the policy gradient to maximize the expected discounted reward.

$$d\theta_\mu = \sum_{t=1}^{T-1} G_t \nabla_{\theta_\mu} ln\mu(S_t|\theta_\mu)$$

- Update the actor parameters by applying the gradients.

$$\theta_\mu = \theta_\mu + \alpha d\theta_\mu$$

- Repeat steps 2 through 5 for each training episode until training is complete.

Fig.18. Training algorithm of PG

- Initialize the actor μ(S) with random parameter values θμ.
- Initialize the critic V(S) with random parameter values θV.
- Generate N experiences by following the current policy. The experience sequence is

$$S_{ts}, A_{ts}, R_{ts+1}, S_{ts+1}, \ldots, S_{ts+N-1}, A_{ts+N-1}, R_{ts+N}, S_{ts+N}$$

- Here, St is a state observation, $A_t$ is an action taken from that state, $S_{t+1}$ is the next state, and $R_{t+1}$ is the reward received for moving from $S_t$ to $S_{t+1}$.
- For each episode step t = ts+1, ts+2, …, ts+N, compute the return and advantage function
  - Compute the return Gt, which is the sum of the reward for that step and the discounted future reward

$$G_t = \sum_{k=t}^{ts+N} \left(\gamma^{k-t} R_k\right) + b\gamma^{N-t+1} V(S_{ts+N}|\theta_V)$$

  Compute the advantage function $D_t$

$$D_t = G_t - V(S_t|\theta_V)$$

- Learn from mini-batches of experiences over K epochs. To specify K, use the NumEpoch option. For each learning epoch:
  - Sample a random mini-batch data set of size M from the current set of experiences. Each element of the mini-batch data set contains a current experience and the corresponding return and advantage function values.
  - Update the critic parameters by minimizing the loss Lcritic across all sampled mini-batch data.

$$L_{kritik}(\theta_V) = \frac{1}{M} \sum_{i=1}^{M} (G_i - V(S_i|\theta_V))^2$$

  - Update the actor parameters by minimizing the loss $L_{actor}$ across all sampled mini-batch data.

$$L_{aktör}(\theta_\mu) = -\frac{1}{M} \sum_{i=1}^{M} min(r_i(\theta_\mu) * D_i, c_i, (\theta_\mu) * D_i)$$

- Repeat steps 3 through 5 until the training episode reaches a terminal state.

Fig.19. Training algorithm of PPO

- Initialize the critic Q(S,A) with random parameter values $\theta_Q$ and initialize the target critic with the same random parameter values: $\theta_{Q'} = \theta_Q$
- Initialize the actor μ(S) with random parameter values θμ, and initialize the target actor with the same parameter values: $\theta_{\mu'} = \theta_\mu$
- For each training time step:
  - For the current observation S, select action A = μ(S) + N, where N is stochastic noise from the noise model.
  - Execute action A. Observe the reward R and next observation S'.
  - Store the experience (S,A,R,S') in the experience buffer.
  - Sample a random mini-batch of M experiences. ($S_i$, $A_i$, $R_i$, $S'_i$) from the experience buffer.
  - If $S'_i$ is a terminal state, set the value function target $y_i$ to $R_i$. Otherwise, set it to

$$y = R_i + \gamma\, Q'(S'_i, \mu'(S'_i|\theta_\mu)|\theta_Q)$$

  - Update the critic parameters by minimizing the loss L across all sampled experiences.

$$L = \frac{1}{M}\sum_{i=1}^{M}(y_i - Q(S_i, A_i|\theta_Q))^2$$

  - Update the actor parameters using the following sampled policy gradient to maximize the expected discounted reward.

$$\nabla_{\theta_\mu} J \approx \frac{1}{M}\sum_{i=1}^{M} G_{ai}\, G_{\mu i}$$

$$A = \mu(S_i|\theta_\mu)\ iken\ \ G_{ai} = \nabla_A Q(S_i, A|\theta_Q)$$

$$G_{\mu i} = \nabla_{\theta_\mu}\mu(S_i|\theta_\mu)$$

  $\boldsymbol{G_{ai}}$ is the gradient of the critic output with respect to the action computed by the actor network
  $\boldsymbol{G_{\mu i}}$ is the gradient of the actor output with respect to the actor parameters.
  Both gradients are evaluated for observation $S_i$.

- Update the target actor and critic parameters depending on the target update method.

Fig. 20. Training algorithm of DDPG

- Initialize each critic $Q_k(S,A)$ with random parameter values $\theta_{Qk}$, and initialize each target critic with the same random parameter values: $\theta_{Qk'} = \theta_{Qk}$
- Initialize the actor μ(S) with random parameter values $\theta_\mu$, and initialize the target actor with the same parameter values: $\theta_{\mu'} = \theta_\mu$
- For each training time step:
  - For the current observation S, select action A = μ(S) + N, where N is stochastic noise from the noise model.
  - Execute action A. Observe the reward R and next observation S'.
  - Store the experience (S,A,R,S') in the experience buffer.
  - Sample a random mini-batch of M experiences ($S_i$,$A_i$,$R_i$,$S'_i$) from the experience buffer.
  - If S'i is a terminal state, set the value function target yi to Ri. Otherwise, set it to

$$y_i = R_i + \gamma * \min_k\left(Q_k(S'_i, clip(\mu'(S'_i|\theta_\mu) + \varepsilon)|\theta_{Qk}\right)$$

The value function target is the sum of the experience reward Ri and the minimum discounted future reward from the critics

  - At every time training step, update the parameters of each critic by minimizing the loss Lk across all sampled experiences.

$$L_k = \frac{1}{M}\sum_{i=1}^{M}(y_i - Q_k(S_i, A_i|\theta_{Qk}))^2$$

  - Every D1 steps, update the actor parameters using the following sampled policy gradient to maximize the expected discounted reward.

$$\nabla_{\theta_\mu} J \approx \frac{1}{M}\sum_{i=1}^{M} G_{ai}\, G_{\mu i}$$

$$G_{ai} = \nabla_A \min_k\left(Q_k(S_i, A|\theta_Q)\right), where\ A = \mu(S_i|\theta_\mu)$$

$$G_{\mu i} = \nabla_{\theta_\mu}\mu(S_i|\theta_\mu)$$

  - Every D2 steps, update the target actor and critics depending on the target update method.

Fig.21. Training algorithm of TD3

- Initialize the actor μ(S) with random parameter values θμ.
- Initialize the critic V(S) with random parameter values $\theta_V$.
- Generate N experiences by following the current policy. The episode experience sequence is

$$S_{ts}, A_{ts}, R_{ts+1}, S_{ts+1}, \ldots, S_{ts+N-1}, A_{ts+N-1}, R_{ts+N}, S_{ts+N}$$

Here, $S_t$ is a state observation, $A_t$ is an action taken from that state, $S_t+1$ is the next state, and $R_t+1$ is the reward received for moving from $S_t$ to $S_t+1$.

- For each episode step t = ts+1, ts+2, …, ts+N, compute the return Gt, which is the sum of the reward for that step and the discounted future reward.

$$G_t = \sum_{k=t}^{ts+N} \left(\gamma^{k-t} R_k\right) + b\gamma^{N-t+1} V(S_{ts+N}|\theta_V)$$

Here, b is 0 if $S_{ts+N}$ is a terminal state and 1 otherwise.

- Compute the advantage function Dt.
-

$$D_t = G_t - V(S_t|\theta_V)$$

- Accumulate the gradients for the actor network by following the policy gradient to maximize the expected discounted reward.

$$d\theta_\mu = \sum_{t=1}^{N} \nabla_{\theta_\mu} \ln\mu\left(S_t|\theta_\mu\right) * D_t$$

- Accumulate the gradients for the critic network by minimizing the mean squared error loss between the estimated value function V (t) and the computed target return Gt across all N experiences.

$$d\theta_V = \sum_{t=1}^{N} \nabla_{\theta_V} \left(G_t - V(S_t|\theta_V)\right)^2$$

- Update the actor parameters by applying the gradients.

$$\theta_V = \theta_V + \alpha d\theta_\mu$$

α is the learning rate of the actor

- Update the critic parameters by applying the gradients.

$$\theta_V = \theta_V + \beta d\theta_\mu$$

β is the learning rate of the actor

- Repeat steps 3 through 9 for each training episode until training is complete.

Fig. 22. Training algorithm of AC