Article

A Data-efficiency Training Framework for Deep Reinforcement Learning

Wenhui Feng, Chongzhao Han*, Feng Lian and Xia Liu

Ministry of Education Key Laboratory for Intelligent Networks and Network Security,
School of Automation Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China;
wenhfeng@stu.xjtu.edu.cn; czhan@mail.xjtu.edu.cn; lianfeng1981@mail.xjtu.edu.cn; liuhu95@stu.xjtu.edu.cn
* Correspondence: czhan@mail.xjtu.edu.cn

Abstract: Sparse reward long horizon task is a major challenge for deep reinforcement learning algorithm. One of the key barriers is data-inefficiency. Even in the simulation environment, it usually takes weeks to training the agent. In this study, a data-efficiency training framework is proposed, where a curriculum learning is design for the agent in the simulation scenario. Different distributions of the initial state are set for the agent to get more informative reward during the whole training process. A fine-tuning of the parameters in the output layer of the neural network for value function is conduct to bridge the gap between sim-to-real. An experiment of UAV maneuver control is conducted in the proposed training framework to verify the method more efficient. We demonstrate that data-efficiency is different for the same data in different training stages.

Keywords: deep reinforcement learning; data efficient; curriculum learning; transfer learning

1. Introduction

Reinforcement learning (RL) has been one of the hottest topics in artificial intelligence community since it led to many successes in learning proper policies for sequential decision-making problems. As it is inspired by human beings' trial-and-error learning process, RL algorithms usually experience many failures before they can achieve the optimal policies for the tasks. This learning paradigm makes the RL appropriate for simulation environments and many progresses have been reported, such as playing the Atari games [1], defeating the best human player at the game of Go [2, 3], beating humans in the imperfect information game poker [4, 5], and creating master recordings at the games of Quake III Arena 2 [6] and StarCraft [7]. Besides the above simulation field application, there are also a few results in real industrial fields being reported recently [8-18].

Despite the conspicuous RL successes in both simulation and real domain, several challenges exist that inhibit wider adoption of reinforcement learning for industrial control domain. One of the challenges is the trial-and-error learning paradigm of RL. The typical RL algorithms interact with the environment and seek the proper policies for the tasks from random policies. The initial random policies cannot fulfill the tasks and often lead the agents to failure. For the simulation domain, failure means restarting of the game and causes little damages. But failure often leads to great losses and even disasters in the industrial control community. For example, in automatic driving domain, the initial random policy may cause vehicle collision. So, the RL algorithm cannot be deployed in the vehicle until it is sufficiently trained in the simulation environment for the automatic driving domain. Similar situations will happen in the UAV, robot control, and many other industrial control fields. In order to address this challenge, sim-to-real transfer is introduced into the RL, which means to train a policy in the simulation of the real domain, and then use transfer learning to meet the gap between the simulation and the real fields [19, 20]. For tasks as complex as modern industrial control field, transfer learning from simulation to the real domain is a challenge question.

Another challenge obstructing the application of RL in real control field is the data efficiency. Deep neural network (DNN) is used both to represent the continuous, high dimension state space of the control task and to approximate the policy function. The combination of the DNN and the RL is deep reinforcement learning (DRL). Data efficiency is the notorious problem in DRL. On one hand, training a complex DNN needs a lot of data. On the other hand, during the training process of the DRL, the policy updating process keeps changing the stationary distribution of the state in the environment. Many data are necessary to evaluate the policy in each stationary distribution, and this means much more data are needed to train the DRL. In many real industrial fields, the time for training a DRL nets to get satisfied results is unacceptable. Take one of the most popular benchmarks for DRL algorithm, Atari Learning Environment [21, 22], as an example. DRL needs millions of frames of images to play the games acceptably well [23, 24], which corresponds to days of pay experience using the standard frame rate. However, human players can get the same level in just minutes [25].

The data for training DRL are different from those for training DNN in supervised learning (SL). The training data for SL are usually considered as independent and identically distributed. But the data for DRL is from interaction experiences between the agent and the environment. They have quite strong relativity. To circumvent this problem, experience replay is widely used in various DRL algorithms [26, 27].

Reward is the very important hint for the DRL algorithm to seek the optimal policy and distinguish the good policies from the bad ones. But in most environment, the agent can only get valid reward at the end of the episode when the agent reaches the terminal state [29]. At this time step, if the agent gets to the expected state, the task is accomplished and the policy will be reinforced. Otherwise, the algorithm will get into another episode and collect more data. As to the real industrial control tasks, the state spaces are continuous and high dimensional. The expected state is a point in the state space. The agent, led by the initial random policy, has little chance to reach the expected state. Thus, the DRL algorithm has no informative data to update the policy. This is the sparse reward problem in DRL community. Many works have been carried out to address this problem. They can be classified as follow: reward shaping, curriculum learning, hierarchical reinforcement learning, and, hindsight experience replay. Reward shaping [30-32] is to design an extra reward for each state-action pair in the episode using domain knowledge or experts experience to lead the agent to an optimal policy. Reward shaping needs in-depth insight to the environment and task to construct proper extra reward functions. Curriculum learning [33-36] is a methodology to optimize the order in which experience is accumulated by the agent, in order to accelerate the training process and increase the performance. Hierarchical reinforcement learning (HRL) [37-39] has recently shown its advantage in sample-efficient learning on the difficult long-horizon tasks. The core in HRL is to divide a complex problem into a hierarchy of more tractable subtasks. The high-level policy produces a subgoal for the policy in the low-level which can be achieved, so that the task can be solved efficiently. Another technic related closely to the HRL is hindsight experience replay (HER) [40, 41], which treats states in the history episodes as goal states.

In this work we apply the sim-to-real training paradigm to the DRL, and in the simulation-based training scenarios, we design curriculum for the agent by setting the different distributions of the initial state s_0 in different training stages. The proposed approach can only be applied in simulation environment and achieve better results in our experiments than other state-of-the-art curriculum learning methods. To bridge the gap between the simulation and reality scenarios, we fine-tune the output layer of the policy network and can get the proper policy through a few interactions with the reality environment. Specifically, the main contributions of this study are as follows:

1. Curriculum design for the agent in simulation environment.

We take advantage of the simulation environment, and set the initial state s_0 close to the goal in the early stage of the training. With the training process going on, the distances between s_0 and the goal state are set longer. Thus, we design the curriculum heuristically from easy to difficult for the agent.

2. Data efficiency analysis for the DRL.

The data fed to the DRL agent is different from those to the SL. The sequential data that the agent collects during learning are not only correlated with each other but also with the behavior policy. We analyze the data efficiencies in different training stage.

3. Transferring the learned policy from simulation scenario to reality one.

To meet the gap between the simulation-based environment and the reality one, we set up a training frame to transfer the learned policy from the simulation scenario to reality. During the simulation-based training process, we add random noise to the state transition function to model the uncertainty in the simulation environment. And in the reality environment, the parameters for the last layer of the neural network for policy or value function are fine-turned to remedy the gap between the sim-to-real.

The remainder of this paper is structured as follows: The related work is discussed in section2. Section 3 presents the detailed methods. Experiment of UAV control is introduced in Section 4. And the results are present in Section 5. Finally, Section 6 concludes this paper.

2. Related Work

2.1 Hindsight experience replay

To address the sparse reward problem in DRL, the technique of hindsight experience replay has been proposed [40]. The pivotal idea behind HER is, besides the goal for the agent to achieve, the algorithm can replay each episode with a different goal. Specifically, wherever the agent achieves at the end of the episode, the final state will be treated as a separate goal. The experience gathered in this episode may not help the agent to achieve the goal, but it still teaches the policy how to achieve the final state of the episode. So, HER is a multi-goal RL, and follow the principles from universal value function approximators [42], the policy and value function trained by HER takes as input not only the state, but also a goal. Although HER achieve the state-of-art performance in many RL benchmark environments, as we discuss above, HER maps a larger space (enlarged by the goal subspace) to policy and value function. This will instinctively need more data to approximate the function.

2.2 Multi-level Hierarchical Reinforcement Learning

Another proven and effective method to address the prohibitive data inefficiency problem of the DRL is multi-level HRL. This approach uses the divide-and-conquer idea to divide a problem into several short horizon subproblems. The high-level policy provides a subgoal for the low level to achieve, and the low level interacts with the environment directly to learn a policy to fulfill the task from the high-level policy [38]. The recent work finds that the primary advantage of HRL is that the low-level policies has improved their exploration capabilities [37]. Despite all their advantages, HRL has only heuristic method to design the reward function for the high-level actions, and it's usually very hard to training the multiple levels of policies simultaneously.

To design the reward function for the high-level actions, [43] combines the ideas of HER and HRL, and proposes nested policies and hindsight action transitions. The hierarchical Q-learning algorithm proposed by the authors can train the multi-policy immediately.

In contrast, our work transfers the long horizon sparse reward task into short horizon one by directly setting the initial state distributed closely to the goal in the simulation scenario. We attribute the inaccuracy of the simulation environment into the noise in the state transition functions. The results of experiments of UAV control show that our method is more data-efficient than the state-of-art HER and HRL methods.

3. Methods

In this section, we present our framework for training the data-efficient DRL. In the episodic task. The agent starts from some initial state s_0 to interact with the environment. At each time step t, the agent receives some state of the environment s_t and selects corresponding action a_t according to the policy $\pi(a_t \mid s_t)$. This process continues until the agent receives the terminal state s_T. If the terminal state s_T is the goal of the task, the agent successes in the task and receives a positive reward. Otherwise, the agent fails in the task and receives the zero reward as the other time steps. This is the typical scenario of the sparse reward RL. Even in simulation environment, this problem is prohibitively data-inefficient. We propose a data-efficient training framework for this problem.

3.1. *Curriculum in Simulation Scenario*

As we have discussed above, the DRL can rarely be deployed in the industrial control community directly. It is usually trained in simulation environment before it can be deployed in reality. In the simulation environment, we design the curriculum by setting the initial state s_0 distributed close to the goal to increase the probability for the agent to win the task and receive the informative reward. The informative reward can lead the learning process to get better policy. With better policy, the agent interacts with the environment more effectively and collect trajectories leading to the goal with more chance. So, in this framework, we can get the training process into the positive circle. As the training process goes on, the initial state s_0 is set farther from the goal so that the agent can explore more state space. Specifically, the algorithm counts the trajectories which lead to the goal in the end, and set the hyperparameter Γ as threshold value. When the count of the successful trajectories reaches Γ, the initial state s_0 is set to a new distribution farther from the goal. A new round of collecting data, counting the successful trajectories, training the policy goes on. Until the distribution of s_0 reaches the original distribution in the task, the agent can collect data to training the proper policy for the task.

3.2. *Experience Replay Buffers*

Experience replay can reduce the correlation of the data fed to the online training of the DRL to accelerate the training process. It has been the standard composition in the DRL [1,44].

As the unbalance data impose much difficulty in SL, there are rare data from successful trajectories in the experience replay buffer during the early stage of the training process in DRL. To cope with this situation, we set two experience replay buffers. The data from the trajectories which achieve the task in the end are saved in one buffer, denoted as B_w, and those from the failure trajectories saved in the other, denoted as B_l. As it can be shown, in the early stage of the training process, the algorithm needs more successful data to teach the agent which action will lead to the goal. The probability which the algorithm sample training data from the B_w is set relatively high at this training stage. As the training process goes on, the algorithm needs data from the loss trajectories to teach the agent which action will lead to failure. The probability which the algorithm sample data from B_w decay with the factor $\tau < 1$ during the training process until it equals to 0.5:

$$p(B_w) \leftarrow \max\{p(B_w) \times \tau, 0.5\} \tag{1}$$

The pseudocode for the DQN in our data-efficient training framework can be shown in the follow algorithm.

```
Algorithm. Data-efficient DQN
Initialize: experience replay buffer for win trajectories $B_w$
            experience replay buffer for lose trajectories $B_l$
            a temp buffer $B_t$
            the count of win trajectories $C = 0$
            the network for action-value function $Q$ with random weights $\theta$ and the
            target action-value function $\hat{Q}$ with random weights $\theta^- = \theta$
For episode = 1, $M$ do
    If $C < \Gamma$:
        Sample initial state $s_0$
    Else:
        Set the distribution of initial state $s_0$ farther from the goal, and sample $s_0$
    For t=1, T do
        With probability $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \arg\max_a Q(s_t, a_t; \theta)$
        Interact with the environment using $a_t$ and observe reward $r_t$ and the next
        state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in the temp buffer $B_t$
        If the episode terminates and wins the task:
            $C \leftarrow C + 1$
            Copy the temp buffer $B_t$ to $B_w$
        Elseif the episode terminates and lose the task:
            Copy the temp buffer $B_t$ to $B_l$
        Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $B_w$ with prob-
        ability $p(B_w)$, otherwise sample the transitions from $B_l$
        Set $y_j = \begin{cases} r_j & \text{if } j+1 \text{ is the end of the episode} \\ r_j + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with repect to the net-
        work parameters $\theta$
        Every $d$ steps reset $\hat{Q} = Q$
    End For
    Decay the probability of sample from win buffer $p(B_w)$ according to function (1)
End For
```

3.3. Transfer from Simulation to Reality

The data-efficient training method we discuss above can only be applied in simulation environments. To deploy the policy into realistic scenarios, transfer learning techniques must be used to bridge the gap between the sim-to-real [20]. In this study, we add random noise into the state the agent observed in the simulation-based training process. This approach is equivalent to meta learning trained over a distribution of tasks [45]. When the learned results are deployed in the reality environments, the input and hidden layers of the networks for the value function are frozen. The data collected from the interaction with the reality environment are used to training the output layer of the networks. These are the fine-tuning techniques widely used in the transfer learning in the SL community. Because only parameters of one layer of the network are tuned, only a few episodes are needed.

4. Experiment of UAV Maneuver Control

In this section, we conduct an experiment to control the UAV's maneuver in the air combat scenario. Because of the complexity of the air combat scenario, the conventional DRL needs prohibitively vast amount of data to training a reasonable policy. It will be shown that the data-efficient DRL training framework discussed above can be used to generate flight control commands to win the task in the air combat fields.

4.1. Problem Formulation

The aerial view of the air combat field is show in figure 1. The UAV starts from the origin of the field, denoted as S . The task for the UAV is to reconnoiter the enemy position, denoted as G , in the upper right of the field. The coordinates of the S and G are $(0km, 0km, 0km)$ and $(20km, 20km, 0km)$, respectively. There is an enemy airport between the origin S and the goal G , and aircrafts will take off to intercept the UAV at any space in the field. A policy must be learned to make a sequence decision on the maneuver for the UAV to detach the enemy aircraft and make its way to the goal G . The task will be regarded as being achieved if the UAV can approach the target within a distance of 100 meters. If the enemy aircraft can approach the UAV within the distance of 100 meters, the UAV is recked and loses the mission.

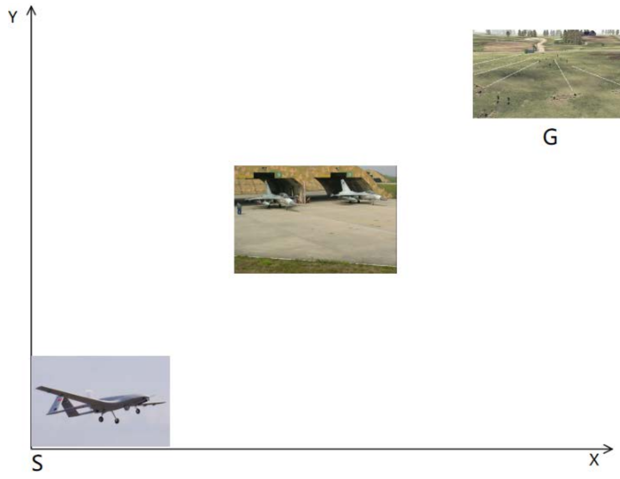

Figure 1. Air combat field

4.2 Dynamic Model for UAV

In order to control the maneuver of the UAV, the motion model must be established. We use a three-degree-of-freedom particle model to describe the UAV. The ground coordinate system is shown in figure 2. The ox axis points to the east, the oy axis points to the north, and the oz follows the right-hand rule of coordinate axis. The motion model of the UAV in the coordinate system is shown in

$$\begin{cases} \dot{x} = v\cos\gamma\sin\psi \\ \dot{y} = v\cos\gamma\cos\psi \\ \dot{z} = v\sin\gamma \end{cases} \tag{2}$$

where x, y, and z represent the position of the UAV in the coordinate system. v represents speed, and $\dot{x}$, $\dot{y}$, and $\dot{z}$ are the component values of the speed v on the three coordinate axes. The track angle γ denotes the angle between the velocity vector and the oxy plane.

The projection of v on the oxy plane is denoted as v'. The heading angle ψ is the angel between v' and oy axis. In the same coordinate system, the dynamic model for the UAV can be shown as

$$\begin{cases} \dot{v} = g(n_x - \sin\gamma) \\ \dot{\gamma} = \dfrac{g}{v}(n_z \cos\mu - \cos\gamma) \\ \dot{\psi} = \dfrac{g n_z \sin\mu}{v\cos\gamma} \end{cases} \tag{3}$$

where g is the acceleration of gravity. n_x is the overload in the velocity direction. n_z is the overload in the pitch direction. μ is the roll angle around the velocity v. $[n_x, n_z, \mu] \in R^3$ are the feasible basic control parameters in the UAV maneuver control model.

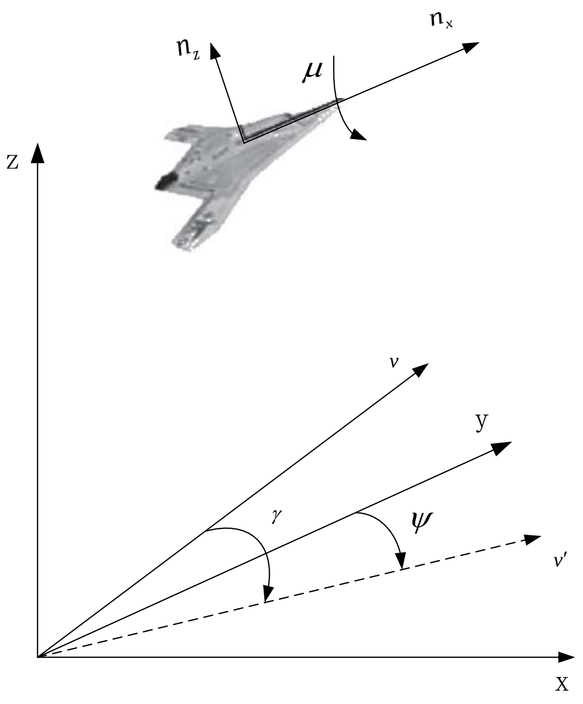

Figure 2. Ground coordinate system for UAV

4.3 State Space

The state space of this scenario has two components. One is the enemy aircraft states. Because we don't control the enemy aircraft, it can be seen as mass point. The states of the enemy aircraft have six dimensions to denote its position and velocity, denoted as $[x, y, z, \dot{x}, \dot{y}, \dot{z}]$. In the experiment, the enemy aircraft flies towards the UAV with a velocity of 150 m/s. The other component of the state space is the states for the UAV. Besides the position and velocity, the states of the UAV also include track angle γ and heading angle ψ, denoted as $[x, y, z, \dot{x}, \dot{y}, \dot{z}, \gamma, \psi]$. In order to facilitate the training of the neural network, each component of the states is normalized into the scale between $[0,1]$ before can be fed into the input layer.

4.4 Action Space

As we have discussed above, the action space for the UAV is $[n_x, n_z, \mu] \in R^3$. This is a continuous action space. In order to reduce the complex of the problem, we employ the idea from [46] and discretize the UAV action space into 15 actions. UAV can take 5 types of actions for direction: forward, right-turn, left-turn, upward, and downward. And it takes 3 velocity types of action for each direction: deceleration, maintain, and accelerate. It is shown in table 1.

Table 1. Maneuver library

No	Maneuver	Control Values		
		n_x	n_z	μ
1	forward maintain	0	1	0
2	forward accelerate	2	1	0
3	forward decelerate	-1	1	0
4	left turn maintain	0	8	-arccos(1/8)
5	left turn accelerate	2	8	-arccos(1/8)
6	left turn decelerate	-1	8	-arccos(1/8)
7	right turn maintains	0	8	arccos(1/8)
8	right turn accelerate	2	8	arccos(1/8)
9	right turn decelerate	-1	8	arccos(1/8)
10	upward maintain	0	8	0
11	upward accelerate	2	8	0
12	upward decelerate	-1	8	0
13	downward maintain	-1	8	π
14	downward accelerate	2	8	π
15	downward decelerate	-1	8	π

At each time step, the agent selects an action $a \in A$ according to the state. the agent control vector can be figured out by Table 1, which means:

$$A = \{a_1, a_2, \cdots, a_m\},\ m = 15 \tag{4}$$

$$a_i = [n_x, n_y, \mu],\ i = 1, 2, \cdots, 15 \tag{5}$$

4.5 Reward Function

If the agent can get to the target finally, the reward of 1 is granted to the agent. If the agent is regarded as crashing with enemy aircraft, it gets a reward of -10. Otherwise, the agent gets 0, which can be shown as follow.

$$r_t = \begin{cases} 1, & if \ \ \|s_t - G\|_2 \le 100 \\ -10, & if \ \ \|s_t - c_t\|_2 \le 100 \\ 0, & \text{otherwise} \end{cases}$$

Where s_t denotes the position of the UAV at timestep t, which are the seventh to the nineth dimensions of the state vector. c_t denotes the position of the enemy aircraft, which are the first three dimensions of the state vector.

4.6 The Distributions of the Initial State

In this study, we design the curriculum learning for the agent by setting the distribution of the initial state s_0 close to the goal in the early training stage. With this method, the agent has more chance to reach the goal and get the informative reward. As the training process goes on, the initial state is set farther away to the goal. Designing the curriculum in this way is directly and intuitively. Heuristically, we divided oxy plane of the air combat field into four parts as shown in figure 3. The initial state is set uniformly distributed in the No. 1 area in the combat field firstly. Then it is set in No.2 area, and so on, until it goes to the origin as the task. The threshold value Γ for counts of the successful trajectories for each area is set 50. That means if the agent collects 50 trajectories that fulfil the task, the initial state goes to the next area.

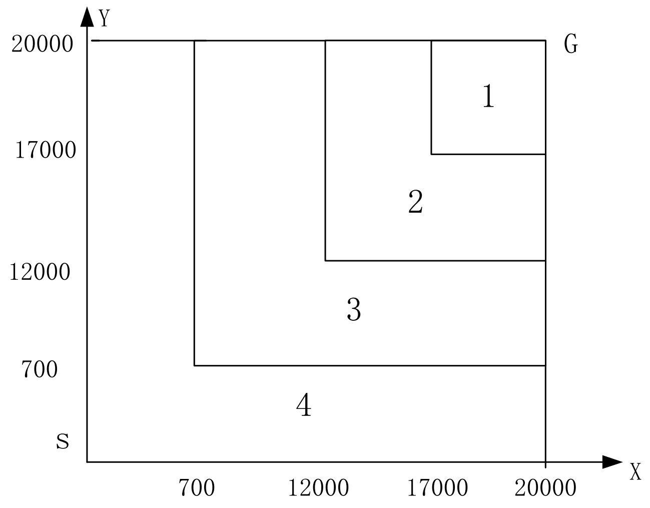

Figure 3. Distribution of the initial state setting in curriculum learning

4.7 Network Architecture for the Value Function

The algorithm of DQN is applied to training the agent. A fully connected neural network with three hidden layers is used to present the value function. The input layer has 14 units as the dimension of the state space, and the output layer has 15 units as that of the action space. The units for the hidden layers are 128, 512 and 64 respectively. The output layer is a linear transformation, and the activation functions for the other layers are ReLU. We summary all the hyperparameter used in the study in table 2.

Table 2. Hyperparameter for the experiment

Hyperparameter	value
threshold for win trajectories Γ	50
parameter for $\varepsilon - decay$	0.995
parameter for $p(B_w)$ decay τ	0.95
discounting rate γ	0.98
units for the NN	(14, 128, 512, 64, 15)
learning rate α	0.002
update the target net every d step	100
buffer size for the two experiments replays	5000
minibatch size	64
max velocity for the UAV	200 m/s
min velocity for the UAV	80m/s

5. **Results**

We compare our training framework with three state-of-art methods in DRL community. They are HER[40], HIRO[39], and MaxEnt_IRL[47]. We have discussed HER above. HIRO is hierarchical Reinforcement Learning with hindsight experiment replay, which using multi-layer policy to divide the task into multiple short horizon ones. As to the curriculum learning, [47] designs the curriculum without human interference. Figure 4 is the learning curves for these four methods. The result is averaged for 30 rounds.

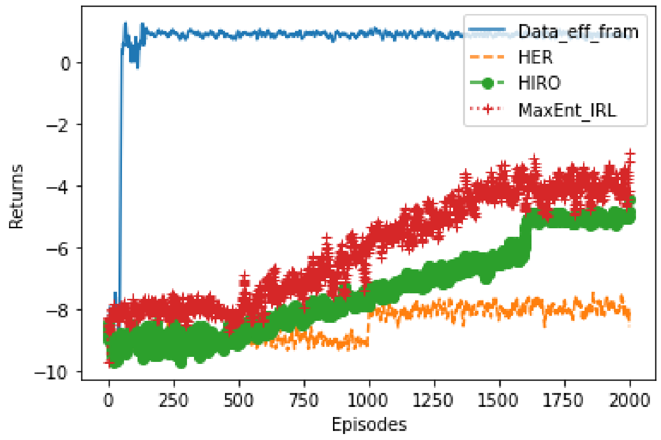

Figure 4. Training Curve for the four methods

It can be shown form the learning curves, the proposed method in this study can get a good result in 196 episodes. After interacting with the environment for about 70 episodes in the first curriculum, the agent does a good job in this easy scenario. Then the agent interacts in a new curriculum, the learning curve shows that the performance degrades significantly. But the agent can catch up within a few episodes. But the other methods cannot get convergent in 2000 episode.

As mentioned above, data-efficiency is different in different training stages. In the early stage, the agent needs data from successful trajectories to know how to get the task done. As the training processes go on, the agent needs to know which actions is wrong for the task. So, data from failure trajectories should be fed to the algorithm to get the agent to explore widely in the state space. We conduct another experiment in which more failure trajectories are fed to the algorithm in the first training stage. Then, we increase the successful trajectories during the training process. The learning curve and that of our proposed method can be shown in figure 5. It learns more slowly than our method does.

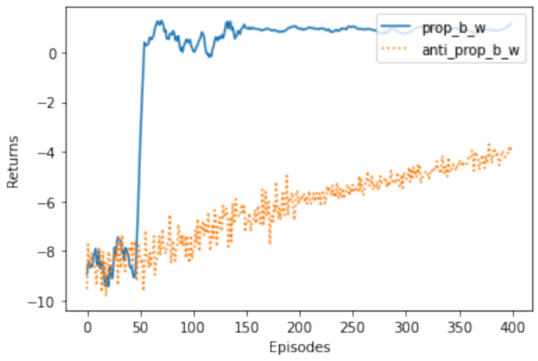

Figure 5. Data-efficiency in different training stage

The experiment is run on a computer with an Inter® Core™ i7-8700k CPU and 16G Ram, a NVIDIA GeForce Rtx 2060 graphics card is installed on this basis for Pytorch acceleration. We present the time cost and total return in table 3.

Table 3. Time cost and total return

Method	Time cost	Total return
Proposed data-efficient framework	28m	0.95
HER	43m	-7.3
HIRO	47m	-5.4
MaxEntIRL	21m	-3.2

6. Conclusion

In this paper, we propose a data-efficient training framework for DRL. We designed a curriculum for the agent in a directly way in the simulation environment. Because we set the distribution of the initial state heuristically, the agent is not only more likely to get the informative reward, but also explore the state spaces where it rarely explores in the trajectories led by the learned policy. This may be another reason that the proposed method can find the proper policy quickly and precisely.

Data-inefficiency is the notorious obstacle for DRL to be applied widely in industry. We find that data-efficiency is different between different training processes even for the same data. Data form the success trajectories are more efficient in the early training stage.

We use DQN as base training algorithm in our training framework. Next, we will apply more DRL algorithms into the proposed training framework.

References

1. Mnih, V.; Badia, A. P., Human-level control through deep reinforcement learning. *Nature*. 2015, 518, 529-533.
1. Silver, D.; Schrittwieser, J.; Simonyan, K., et al. Mastering the game of Go without human knowledge. *Nature*, 2017, 550, 354-359.
2. Silver, D.; Hubert, T.; Schrittwieser, J., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 2018, 362, 1140-1144.
3. Matej Moravík, et al. DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. *Scence*, 2017, 356, 508-513
4. Brown, N.; Sandholm, T. Safe and nested subgame solving for imperfect-information games. *Advances in Neural Information Processing Systems*, 2017, 690-700.
5. Jaderberg M.; Czarnecki W M.; Dunning I , et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 2019, 364, 859-865
6. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 2019, 575, 350–354.
7. Kober, J.; Bagnell, J. A. ; Peters, J. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 2013, 32, 1238-1274.
8. Levine S.; Finn C.; Darrell T.; Abbeel P. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Leaning Reseach*. 2016, 17, 1334-1373
9. Kalashnikov D.; Irpan A.; Pastor P, et al. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. *arXiv preprint arXiv: 1806.10293*, 2018
10. Pinto L.; Gupta A. Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, 3406–3413
11. Nagabandi A.; Konoglie K.; Levine S.; Kumar V. Deep Dynamics Models for Learning Dexterous Manipulation. *2020 Conference on Robot Learning*, PMLR 2020, 1101-1112
12. Kalashnikov D.; Varley J.; Chebotar Y, et al. MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale. *arXiv preprint arXiv: 2104.08212*, 2021
13. Gupta A.; Yu J.; Zhao TZ, et al. Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention. *arXiv preprint arXiv: 2104.11203*, 2021
14. Degrave J.; Felici F.; Buchli J, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*. 2022, 602, 414-419
15. Mirhoseini A.; Goldie A.; Yazgan M, et al. A graph placement methodology for fast chip design. *Nature*: *International weekly journal of science*. 2021, 594, 207-212.
16. Hu J.; Wang L.; Hu T, et al. Autonomous Maneuver Decision Making of Dual-UAV Cooperative Air Combat Based on Deep Reinforcement Learning. *Electronics*. 2022, 11, 467-N.PAG.
17. Kuutti S.; Bowden R.; Fallah S. Weakly Supervised Reinforcement Learning for Autonomous Highway Driving via Virtual Safety Cages. *Sensors*. 2021, 21, 2032-N.PAG
18. Rusu AA.;, Vecerik M.; Rothörl T, et al. Sim-to-Real Robot Learning from Pixels with Progressive Nets. *arXiv preprint arXiv: 1610.042868*, 2016
19. Zhao W; Queralta JP.; Westerlund T. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, 737-744.
20. Bellemare MG.; Naddaf Y.; Veness J.; Bowling M. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*. 2012, 47, 253-279
21. Machado MC.; Bellemare MG.; Talvitie E, et al. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*. 2018, 61, 523-562
22. Schulman J.; Wolski F.; Dhariwal P, et al. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv: 1707.06347*, 2017

23. Hessel M.; Modayil J.; van Hasselt H, et al. Rainbow: Combining Improvements in Deep Reinforcement Learning. *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018
24. Tsividis PA.; Tenenbaum JB.; Pouncy T, et al. Human learning in atari. *AAAI Spring Symposium - Technical Report*. 2017
25. Fedus W.; Ramachandran P.; Agarwal R.; Y. Bengio, et al., Revisiting Fundamentals of Experience Replay. *International Conference on Machine Learning*. 2020, 3061-3071,
26. Zhang S, Sutton RS. A Deeper Look at Experience Replay. *arXiv preprint arXiv: 1712.01275*, 2017
27. Silver D.; Singh S.; Precup D.; Sutton RS. Reward is enough. *Artificial Intelligence*. 2021, 299,
28. Sutton R.; Barto A. Reinforcement learning: An introduction. MIT press, 2018.
29. Ng A Y,; Harada D.; Russell S. Policy invariance under reward transformations: Theory and application to reward shaping. *Morgan Kaufmann Publishers Inc.* 1999.
30. Burda Y.; Edwards H.; Storkey A , et al. Exploration by Random Network Distillation. *arXiv preprint arXiv: 1810.128948*, 2018
31. Badia AP.; Sprechmann P.; Vitvitskyi A, et al. Never Give Up: Learning Directed Exploration Strategies.*arXiv preprint arXiv: 2002.060388*, 2022
32. Wang, X.; Chen, Y.; Zhu, W. A. Survey on Curriculum Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2022, 49, 4555–4576,.
33. Lin Z.; Lai J.; Chen X, et al. Learning to Utilize Curiosity: A New Approach of Automatic Curriculum Learning for Deep RL. *Mathematics* 2022, 10, 2227-7390
34. Yengera G.; Devidze R.; Kamalaruban P.; Singla A. Curriculum Design for Teaching via Demonstrations: Theory and Applications. *arXiv preprint arXiv: 2106.04696*, 2021
35. Zhipeng R.; Daoyi D.; Huaxiong L.; Chunlin C. Self-paced prioritized curriculumlearning with coverage penalty in deep reinforcement learning. *IEEE transactions on neural networks and learning systems*, 2018, 29, 2216-2226.
36. Gehring J.; Synnaeve G.; Krause A.; Usunier N. Hierarchical Skills for Efficient Exploration. *In 35th Conference in Neural Information Processing Systems, 2021.*
37. Vezhnevets AS.; Osindero S.; Schaul T, et al. FeUdal networks for hierarchical reinforcement learning. *In Proceedings of the 34th International Conference on Machine Learning*, 2017, 3540–3549
38. Nachum O.; Gu S.; Lee H.; Levine S. Data-Efficient Hierarchical Reinforcement Learning. *arXiv preprint arXiv: 1805.08296*, 2018
39. Andrychowicz M.; Wolski F.; Ray A, et al. Hindsight Experience Replay. *arXiv preprint arXiv: 1707.01495*, 2017
40. Vecchietti LF.; Seo M.; Har D. Sampling Rate Decay in Hindsight Experience Replay for Robot Control. *IEEE Transactions on Cybernetics*. 2022, 52, 1515-1526.
41. Schaul T., Horgan D.; Gregor K.; and Silver, D. Universal value function approximators. *In Proceedings of the 32nd International Conference on Machine Learning*, 2015, 1312–1320.
42. Levy A.; Konidaris G D.; Platt R , et al. Learning multi-level hierarchies with hindsight. *International Conference on Learning Representations 2019*
43. Zhang S.; Sutton R S . A Deeper Look at Experience Replay. *arXiv preprint arXiv: 1712.01275*, 2017
44. Wang J X.; Kurth-Nelson Z.; Tirumala D , et al. Learning to reinforcement learn. *arXiv preprint arXiv: 1611.05763, 2016*
45. Hu J.; Wang L.; Hu T. et al. Autonomous Maneuver Decision Making of Dual-UAV Cooperative Air Combat Based on Deep Reinforcement Learning. *Electronics*, 2022, 11(3):467-488.
46. Yengera G.; Devidze R.; Kamalaruban P, et al. Curriculum Design for Teaching via Demonstrations: Theory and Applications. *In 35th Conference in Neural Information Processing Systems, 2021.* 202