Monoceros: A New Approach for Training an Agent to Play FPS Games

Ruiyang Yang, Hongyin Tang, Beihong Jin*
State Key Laboratory of Computer Sciences, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
Email: *beihong@iscas.ac.cn

***Abstract*—In the deep reinforcement learning, the sparse reward problem directly impacts the quality of agent training. Existing methods have not been satisfactory, especially for the scenarios with high-dimensional state information. In this paper, we propose a new approach Monoceros to training a game agent. Monoceros can work for the scenarios with high-dimensional state information and alleviate the sparse reward problem during the agent training. Specifically, we present a composite reward function which combines both the knowledge implied in expert trajectories and manually-set reward functions. Moreover, we design a specific policy network to adapt to the high-dimensional information scenarios, and adopt the behavior clone as a pre-training strategy to accelerate the training process. Technically, Monoceros can be applied to train the agents to play First Person Shooter (FPS) games. We conduct extensive experiments on three scenarios in the VIZDoom platform. Experimental results show that in all the scenarios, the agent trained by Monoceros outperforms the agents trained by Arnold and GAIL, which are representative methods in the deep reinforcement learning and the imitation learning, respectively.**

I. Introduction

Deep reinforcement learning is usually used to train an agent so that the agent can achieve a given task in an optimal way. Specifically, while facing the task, deep neural networks are responsible for generating high-level representations for states in the environment, and by a reinforcement learning process, which can maximize the cumulative reward value obtained by the interactions between the agent and the environment, the agent can learn the best policy to complete the task. Because deep reinforcement learning requires continuous interactions with the environment and constant trials, it is not suitable to apply deep reinforcement learning directly to real living environments. Recently, computer games have become alternative environments, and corresponding platforms are developed for studying deep reinforcement learning methods. For example, for Doom, a first-person shooter game, the VIZDoom platform has been developed, where a variety of incomplete information in 3D scenarios is provided, to allow researchers to train and test agents.

However, although with the help of computer games, a lot of low-cost data can be obtained for training the agents by deep reinforcement learning approaches, the problem of "sparse reward" still emerges during the training, that is, only a small amount of actions can make the agent get rewards or punishments. This problem will negatively affect the effectiveness of the trained agent.

Currently, there are four ways to solve the "sparse reward" problem, that is, the reward shaping, the curriculum training, the hierarchical reinforcement learning and the inverse reinforcement learning. However, all these methods have shortcomings. Firstly, the reward shaping refers to modifying the reward settings in the game scenarios to achieve the goal of setting a reward for every action. Considering different complexities of different scenarios, setting proper reward functions is very difficult. If rewards were set unreasonably, then it would be difficult to drive the agent to learn along the direction of the better policy. Secondly, as the name implies, the curriculum training refers to training the agent in simple scenarios in the beginning, when the agent to be trained reaches a certain level, it will be put into complex scenarios to go on training. For the curriculum training, its most important thing is to get the well-designed curriculum. Therefore, similar to the reward shaping, it also suffers the disadvantage of unsatisfactory generalization. Thirdly, in the hierarchical reinforcement learning, tasks are generally divided into two levels. On the top level, the whole task is divided into parts, then each part is implemented on the bottom level. Currently, the hierarchical decomposition stays in the fashion of manual design, automated hierarchical decomposition and combination with domain prior knowledge are still under preliminary research. Fourthly, as for the inverse reinforcement learning, its core idea is to learn the reward functions from the expert trajectories with the expectation that the best policy generated by training the agent through reward functions is distributed near the expert policy. But for scenarios with high-dimensional state information, since they usually have complex structures, it is difficult to obtain complete information from these scenarios. Only relying on reward functions learned from expert trajectories is still difficult to train the agent to complete tasks with the best policy.

Aiming at the shortcomings of existing methods to solve "sparse reward", this paper proposes the Monoceros approach, which is an agent training approach for the scenarios with high-dimensional state information. The main contributions of this paper are summarized as follows.

- We combine the reward functions constructed by expert trajectories with manually-set reward functions to obtain the reward values that can understand the scenarios with high-dimensional state information. In addition, we de-

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

vise the internal structure of the distillation network and random network to adapt to the high-dimensional state information.
- We design the specific internal structure of policy network so as to improve the ability of perceiving and representing the scenarios with high-dimensional state information. In particular, for training the policy network from a reasonable starting point, we adopt the behavior clone instead of random initialization for the parameters in the policy network.
- We conduct a lot of experiments on the VIZDoom platform, the results show that agent trained by Monoceros outperforms ones by GAIL or Arnold in terms of survive duration and kill/death rate. Moreover, as a general solution, Monoceros can be applied to train the agents to play other FPS games, besides Doom.

The rest of the paper is organized as follows. Section II introduces the related work. Section III gives the problem formulation. Section IV describes the Monoceros approach in detail. Section V evaluates the Monoceros approach by experiments. Finally, the paper is concluded in Section VI.

II. Related Work

Our work is broadly related to the research under two nonorthogonal topics: deep reinforcement learning, imitation learning.

Early reinforcement learning methods cannot deal with the tasks in the complex high-dimensional state space, because the features that these methods need are extracted manually. The combination of abstract representations obtained from deep learning and reinforcement learning is expected to solve complex tasks in real life. Such desires push the research on deep reinforcement learning [1], [2]. The deep Q network (DQN) [3], [4] proposed by Google DeepMind team is a success example. DQN combines a deep convolutional neural network with Q learning of reinforcement learning. Taking the original images of the game as input, it reaches the level of human players playing Atari game. Further, the deep recurrent Q learning (DRQN) [5] is proposed to deal with the partial observation in some games.

However, it is difficult for these 2D games with a third-person perspective to imitate the scenes in real life. Researchers need a platform for trying different reinforcement learning methods. In particular, such a platform should obtain raw visual information from a semi-realistic 3D world with a first-person perspective. VIZDoom platform [6] for the classic FPS game named Doom is precisely the platform which meets the above requirements. Depending on ViZDoom, the Arnold agent [7], [8], which developed by the Terminators team of CMU, is an excellent one. Arnold has two neural networks, one of which is a DQN-based network for exploring the map to collect items and find enemies, and the other is a DRQN-based network for deciding actions. Meanwhile, the AI research team from Facebook combines A3C (Asynchronous Advantage Actor-Critic) with the curriculum training to train the F1 agent [9], [10]. They build a policy network which is based on the actor-critic model, and design various scenarios from simple to complex and train F1 according to the curriculum training method.

Imitation learning is to obtain optimal policies from expert trajectories. A well-known imitation learning is behavior clone [11]. It just records the expert trajectory and trains the agent by trajectory data. In general, for pursuing good experimental results, a large number of expert trajectory data are needed, which is not feasible in reality. The inverse reinforcement learning is to learn the reward function from expert trajectory data and uses the reward function to obtain the policy near the policy in expert trajectory data. The apprenticeship learning [12], proposed by Andrew Ng and Abbeel, is in this category. They apply the maximum margin algorithm to solve the reward function. Unfortunately, this algorithm not only has heavy calculation, but also brings the ambiguity, that is, same expert trajectory data will lead to different reward functions. In order to overcome the above disadvantages, maximum entropy based inverse reinforcement learning is proposed [13]. This method builds probabilistic models which fit the distribution of the expert trajectory data, and then chooses the model with the highest entropy (i.e., the least predicted risk) so as to avoid ambiguity.

But the inverse reinforcement learning is often not very applicable in actual training. There are two problems. Firstly, learning the reward function may require selecting features manually. Secondly, in the inverse reinforcement learning, some sub-loops contain the step of reinforcement learning. It introduces not only heavy calculation, but also convergence difficulties. For the first problem, a deep neural network can be used to approximate the reward function, which leads to the deep inverse reinforcement learning [14]. But using neural networks to approximate the reward function further increases the computational complexity. In order to solve the inapplicability of inverse reinforcement learning, GAIL (Generative Adversarial Imitation Learning) algorithm [15] directly extracts the policy from expert trajectory data, thus bypassing many intermediate steps of inverse reinforcement learning.

GAIL draws on the idea of the zero-sum game from GAN (Generative Adversarial Network) [16] and the two players of the game are a generator and a discriminator, respectively. The generator is used to generate the action sequence, and the discriminator is used to distinguish whether this action sequence is an expert one. Thus, the output of discriminator is actually equivalent to the reward function. The generator is trained by a deep reinforcement learning algorithm TRPO (Trust Region Policy Optimization) [17]. Using TRPO, the value of reward function will be monotone increasing or monotone nondecreasing when the policy is updated. However, TRPO needs to compute the conjugate gradient, which also produces heavy calculation and is difficult to implement. PPO (Proximal Policy Optimization) algorithm [18] is an improved version of TRPO algorithm. It can use gradient descent and has less computation in comparison with TRPO. Because the GAIL algorithm borrows the idea of GAN, it inherits training

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

instability and "mode collapse" problem. In order to solve the above problems, a new imitation learning framework named RED (Random Expert Distillation) [19] is proposed. RED treats the expert trajectory data as a support and solves support estimation by RND (Random Network Distillation) algorithm [20] and finally gets a fixed reward function.

Compared with the existing methods, Monoceros proposed in this paper gives a new way to construct the reward function. Specifically, the reward function learned from expert trajectory data is linearly combined with the manually-set reward function. Thus, during the training, the agent can obtain effective bonuses and penalties under the guidance of the new reward function. The resulting agent can be rather smart.

III. Problem Formulation

In this section, we give the problem to be addressed formally.

For a task needed to be performed by an agent (i.e., playing a game), we denote a set of high-dimensional states of scenarios that an agent faces by S and a set of actions that the agent can perform by A. Let trajectory τ be a sequence of (s, a) pairs, where $s \in S$, $a \in A$. Further, we denote the expert trajectory and the sampling trajectory by τ_E and τ_S, respectively.

Taking the Doom game as an example, each state in S is a three-dimensional tensor which represents the RGB information of the image, and the actions in A refer to the actions that a game player can perform. In the health gathering scenario, there are eight actions, that is, turn left, turn right, move forward, move backward, move forward and turn left, move forward and turn right, move backward and turn left, move backward and turn right. Each action can be represented by a one-hot encoding.

Policy π is actually a mapping between a state $s \in S$ and an action $a \in A$, denoted as $a = \pi(s)$. Our goal is to train the agent so that it can perform the task using the optimal policy π^*. Here, the optimal policy refers to the policy that maximizes the cumulative reward expectation, as shown in Equation (1).

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s, a, t)\right] \tag{1}$$

In Eq. (1), $r : S \times A \times \mathbb{T} \to \mathbb{R}$ is a reward function that changes over time $t \in \mathbb{T}$, $\gamma \in [0, 1]$ denotes a discount factor, T denotes the training duration for tasks (i.e., games).

Policy π can be implemented by a neural network. It means that the optimal policy can be obtained through optimizing the policy network. We denote the policy network as π_θ, where θ denotes all the parameters in the policy network.

Finally, we define a state value function to represent the expected reward under state s at time t, that is, $V(s_t) = \widehat{V}(s, t, \theta)$ where V can be estimated through a neural network, which is called the value network, and θ denotes neural network parameters.

IV. Monoceros Approach

In order to achieve desirable training results in high-dimensional information scenarios, we propose the Monoceros approach. As shown in Figure 1, Monoceros contains two parts: constructing rewards and training the policy network.

Following Monoceros, we first construct a reward function using the expert trajectory data. Next, we get a reward by combining the output of this reward function with the output of a manually set reward function. Besides, we initialize the parameters in the policy network by behavior clone. Then we train the policy network, Finally, the trained policy network is the agent that we need.

In particular, considering that the policy network can have a similar structure as the value network, we combine them into one network sharing parameters. Given a state, the policy network and the value network generate the action and the value, respectively.

A. Policy Network

We hope the trained policy network can learn the optimal policy, i.e., every time the policy is updated, the new policy ensures that the expectation of the cumulative rewards increase monotonously. In this subsection, we in turn present the composite loss function, the internal design and the pre-training strategy of the policy network.

In the policy network π_θ, let θ denote the current parameters and θ_{old} denote the parameters in last training iteration. $q_t(\theta) = Pr(a_t|s_t; \theta)/Pr(a_t|s_t; \theta_{old})$ is the ratio of the probability of a same state-action pair produced by the network with current parameters and the network with old parameters.

To compare the strategies, we introduce an advantage function which is shown in Eqs. (2)-(3) where $\lambda \in [0, 1]$ is a hyperparameter. The advantage function $\hat{A}_t$ is an estimation of the advantage value from t-th step. We estimate $\hat{A}_t$ by GAE (generalized advantage estimation).

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{2}$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3}$$

Following PPO, the loss function of the policy network is defined as follows.

$$L_t(\theta) = -\hat{\mathbb{E}}\left[L_t^C(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{4}$$

$$L_t^C(\theta) = \hat{\mathbb{E}}\left[\min\left(q_t(\theta)\hat{A}_t, \text{clip}(q_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{5}$$

$$L_t^{VF}(\theta) = \left(V(s_t) - V(\text{target}_t)\right)^2 |\pi_\theta \tag{6}$$

In Eq. (4) , $L_t^C(\theta)$ is the difference of the probability distributions between the new and old policies after updated. $L_t^{VF}(\theta)$ is the loss function of the value network. $S[\pi_\theta](s_t)$ is the entropy of the policy network and c_1,c_2 are coefficients. $L_t^{VF}(\theta)$ is introduced because the policy network and value network are sharing same parameters. $S[\pi_\theta](s_t)$ is added to ensure fully exploration.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

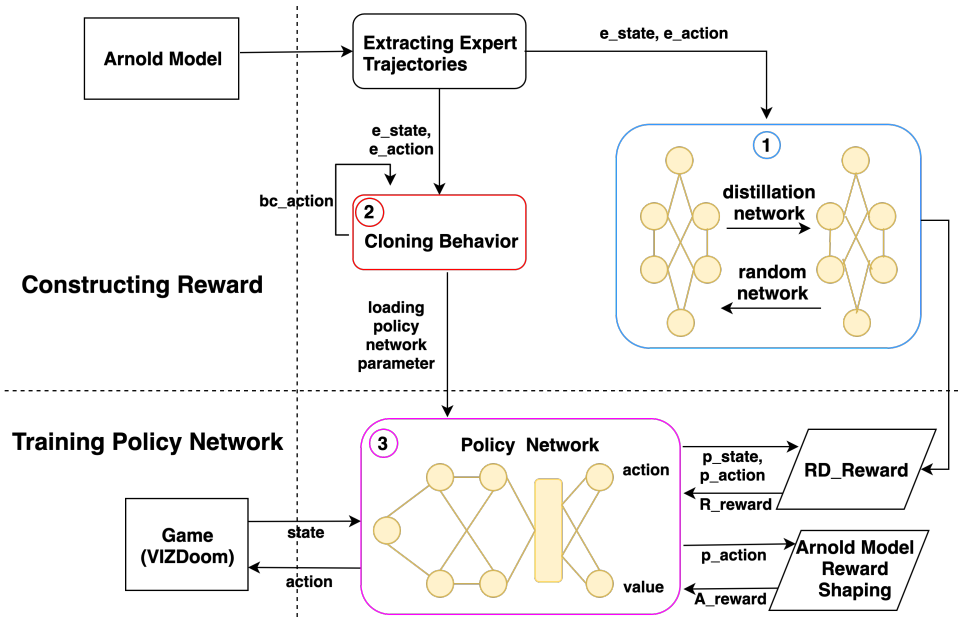


Fig. 1: Monoceros architecture

In Eq. (5), ϵ is a hyperparameter which is generally set to 0.2. $q_t(\theta)$ is usually limited to $[1-\epsilon, 1+\epsilon]$ to ensure that each update does not have much fluctuation. Function clip is for truncation, whose definition is as follows.

$$\text{clip}(\text{value}, 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon, \text{if value} < 1-\epsilon \\ 1+\epsilon, \text{if value} > 1+\epsilon \end{cases} \quad (7)$$

In Eq. (6), $V(\text{target}_t) = r_t + \gamma V(s_{t+1})$

Since we are aiming at the scenarios with high-dimensional state information, we design the specific structure for the policy network, partially referring to the typical network structure in DQN and A3C. As shown in Fig. 2, different from the typical structure which takes stacked grayscale images (4 frames as a unit usually) as input, our network receives single RGB image. According to the difference of inputs, our network consists of three convolutional layers at first, a fully-connected layer in the middle and two fully-connected layer at last. Specifically, the first convolutional layer has an 8*8 kernel, 4 stride and 32 filters. The second convolutional layer has a 4*4 kernel, 2 stride and 64 filters. The third convolutional layer has 3*3 kernel, 1 stride and 64 filters. The size of middle fully-connected layers is 512. Last two fully-connected layers output the action and the value, respectively.

B. Reward Function

Besides a composite loss function of the policy network, a reasonable reward needs to be designed.

We firstly train a reward function R_reward using the expert trajectory data. Details of the training procedure are given as follows. Referring to the RED algorithm, we build a distillation network $f_{\hat{\theta}}(s,a)$ and a random network $f_\theta(s,a)$. Note that the parameters of $f_\theta(s,a)$ are randomly initialized and remained fixed in the following steps. The distillation network and the random network are fed with same expert trajectory data and trained, with a goal that the output of $f_{\hat{\theta}}(s,a)$ is to approximate to the output of $f_\theta(s,a)$. It is emphasized that the trained network is only the distillation network. Therefore, given a new state-action pair, the difference of the output between $f_{\hat{\theta}}(s,a)$ and $f_\theta(s,a)$ measures the similarity of the current pair to the expert data ever seen during training. Large difference means that the current pair is not compatible with the expert policy π_E. So, we set the negation of the difference as a reward, which means the proximity to the expert policy. Eq. (8) gives the definition of R_reward, where σ_1 is a hyperparameter. As shown in Fig.1, RD_Reward represents the calculation module that gets the R_reward.

$$\text{R\_reward}(s,a) = \exp\left(-\sigma_1 (f_{\hat{\theta}}(s,a) - f_\theta(s,a))^2\right) \quad (8)$$

Next, the final reward function is shown in Eq. (9), where $\alpha, \beta \in (0,1)$ and A_reward is the reward that is manually set according to different scenarios with reference to reward shaping of Arnold Model.

$$\text{Reward}_m = \alpha \times \text{R\_reward} + \beta \times \text{A\_reward} \quad (9)$$

Reward_m combines the reward function reconstructed through expert trajectory and the reward function set by manual. Reward_m is consistent with our observation. We find that with limited expert trajectory data, the reward function reconstructed with limited iteration do not necessarily produce an appropriate reward. Meanwhile, the reward set by manual possibly has biases in the understanding of the scenarios which leads to an unreasonable setting. Therefore, we naturally combine them linearly to form a new reward function with

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

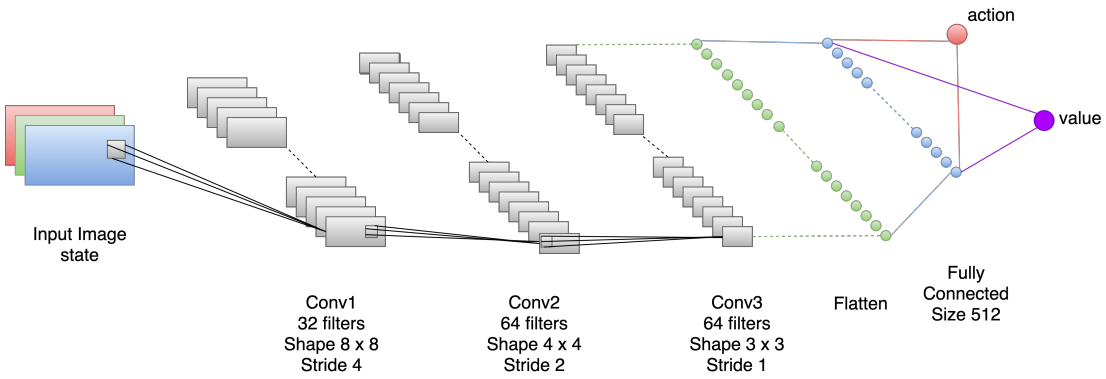


Fig. 2: Policy network structure

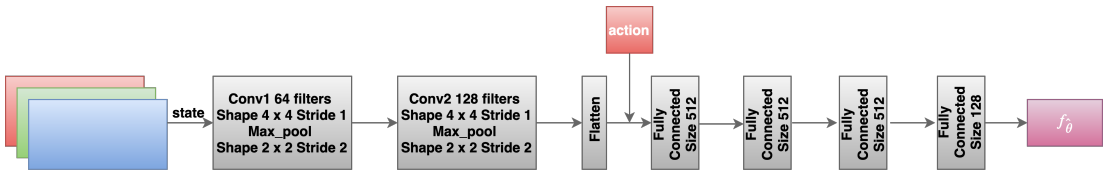


Fig. 3: Distillation network

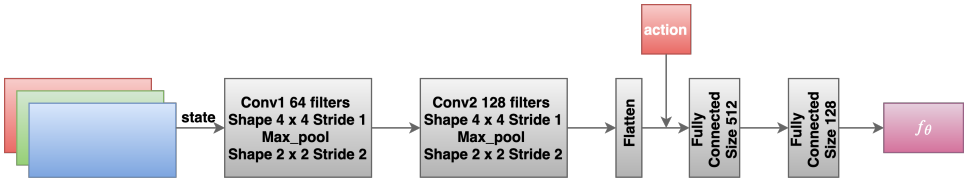


Fig. 4: Random network

certain ratios. The specific ratios can be determined through experiments on different scenarios.

Finally, for the high-dimensional information scenarios, we design the structures of distillation network and random network which are shown in Fig. 3 and Fig. 4, respectively.

V. Experiments

In this section, we conduct experiments to evaluate the "smartness" of the agent trained by different approaches.

A. Experimental Scenarios

We select three Doom game scenarios in the VIZDoom platform as experimental scenarios, that is, health gathering, health gathering (supreme) and limited death match (shotgun). Figure 5 shows the screen shots of three experimental scenarios. These scenarios consist of 35 frames per second images.

In the health gathering scenario, the agent needs to move in lava, losing health points all the time. In order to survive as long as possible in a round, the agent has to keep collecting health packs scattering in lava. The initial health point is set to 101. Each round has 2 minutes. Therefore, the maximum survival duration in one round is $35 \times 120 = 4200$ frames. In the health gathering (supreme) scenario, more wall obstacles are added to increase the difficulty of game. For these two scenarios, we measure the "smartness" of an agent by the survival duration per round. The longer the agent survives, the smarter it is.

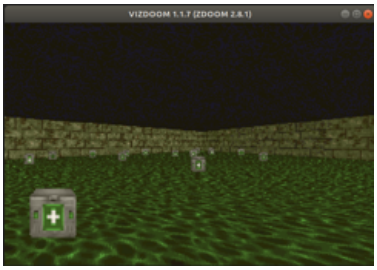
However, in the limited death match (shotgun) scenario, the agent needs to kill the enemies it encounters while moving in the field. The agent can gather health packs and ammos but it can only hold the weapons given at the beginning of the game. The initial values of health packs and ammos are set to 101 and 301, respectively. There are eight enemies when the scenario is initialized. Note both the enemies and agent will be resurrected once being killed within a round. At the end of each round (35fps $\times$ 900s = 31500 frames), we adopt the K/D (kill/death) rate, i.e., the number of killed enemies (kill) divided by the number of deaths (death), to measure how smart an agent is. The higher the K/D rate reaches, the smarter the agent is.

In addition, the VIZDoom platform provides several game resolutions. We adopt the 440*225 resolution which is 16:9. Such 16:9 resolutions provide a 108 degree field of view which are wider than 90 degree in 4:3. Therefore, the agent can receipt more visual information.

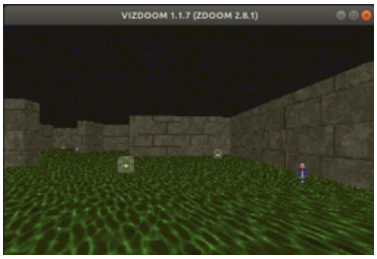
B. Experimental Setup

We select the Arnold model as the expert model to extract expert trajectory data since the model won the championship in the Visual Doom AI Competition 2017.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE



(a) Health gathering



(b) Health gathering (supreme)

(c) Limited death match (shotgun)

Fig. 5: Screen shots of three experimental scenarios

Hyperparameter	Description	Value
Policy Network		
γ	Cumulating reward discount ratio	0.95
c_1	Coefficient of the value function in the target function	1
c_2	Coefficient of the entropy loss function in the target function	0.01
pn_step	Training steps in each epoch	10
pn_minibatch	Mini-batch size in each training step	256
pn_learning_rate	Adam learning rate	5e-5
Reward Function		
rf_iter	Training steps	200
rf_minibatch	Mini-batch size in each training step of distillation network and random network	256
rf_learning_rate	Adam learning rate	1e-3
σ_1	Coefficient in R_reward	250000

TABLE I: Hyperparameters.

For both health gathering and health gathering (supreme) scenarios, we extract 50 rounds game data from Arnold as the expert trajectory data. The Arnold agent can survive until the end (i.e., it gets 4200 points) in each of the 50 rounds. For the limited death match (shotgun) scenario, we extract 20 rounds game data and select top-10 rounds with the highest K/D rate as the expert trajectory data.

Note that Arnold resizes the original color images of 440*225 to the ones of 108*60, and takes the resized ones as the training input. For the sake of fairness, we also adopt the images with the same size as input in our approach.

Table I lists the values of hyperparameters in Monoceros which keep the same in all three experimental scenarios. Some other hyperparameters are not the same since they are sensitive to different experimental scenarios. For example, "frame skips" is set to 4 for both health gathering and health gathering (supreme) scenarios, and set to 3 for the limited death match (shotgun) scenario.

Moreover, for the health gathering, health gathering (supreme) and limited death match(shotgun) scenarios, the reward parameters are set as $\alpha = 0.4/0.1/0.3$, $\beta = 0.6/0.9/0.7$, respectively.

C. Experimental Results

We first compare the behaviors of agents trained by Monoceros, GAIL and Arnold in the experimental scenarios mentioned above.

Table II shows the survive durations and standard deviations averaged by 20 round results in the health gathering and health gathering(supreme) scenarios. Table III shows the average K/D rates among 10 rounds as well as the standard deviations in the limited death match(shotgun) scenario. The percentages in parentheses in Tables II-III are the ratios relative to Monoceros. Obviously, the agent trained by Monoceros outperforms the other agents. We also note that in limited death match(shotgun) scenario, the K/D rate of Monoceros is only slightly better than Arnold. It may contribute to the fact that Monoceros does not use game internal data during training but Arnold does, that is, Arnold knows whether there are enemies in the current game screen. Besides, as shown in Figs. 6-8, Monoceros performs better than the others in nearly

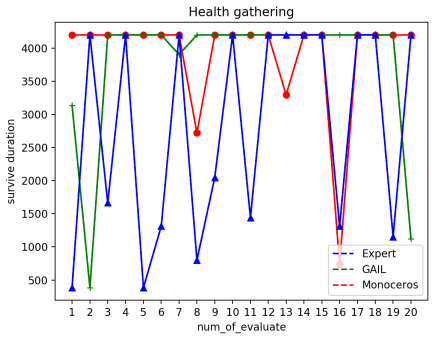


Fig. 6: Survive duration vs. round in the health gathering scenario

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

Scenario	Game Episode	Arnold	GAIL(PPO)	Monoceros
Health gathering	20	2835.8(72.5%) ±1550.16	3788.2(96.9%) ±1045.46	3908.6 ±815.38
Health gathering (supreme)	20	2438.7(79.45%) ±1101.46	1791.6(58.37%) ±995.19	3069.4 ±1032.03

TABLE II: Survive duration.

Scenario	Game Episode	Arnold	GAIL(PPO)	Monoceros
Limited Death Match (shotgun)	10	3.429(99.73%) ±0.087	2.562(74.52%) ±0.126	3.438 ±0.192

TABLE III: Kill/death rate.

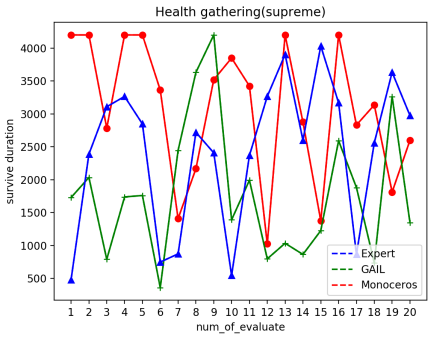


Fig. 7: Survive duration vs. round in the health gathering (supreme) scenario

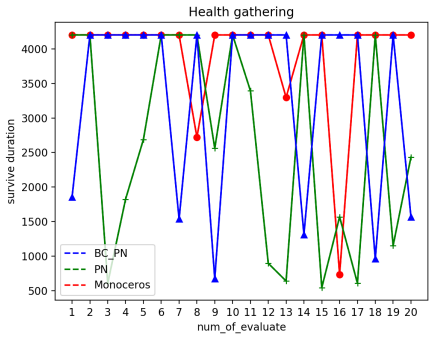


Fig. 9: Survive duration vs. round in the health gathering scenario

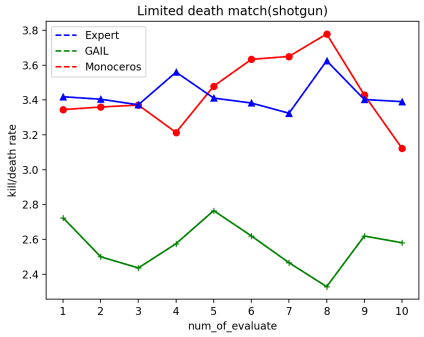


Fig. 8: Kill/death rate vs. round in the limited death match scenario

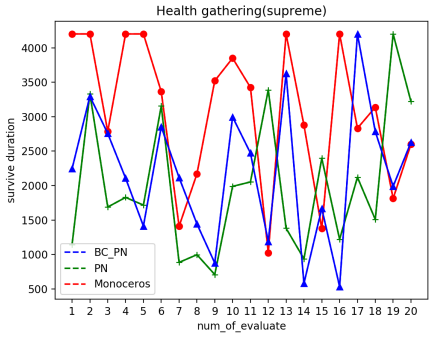


Fig. 10: Survive duration vs. round in the health gathering (supreme) scenario

every round.

Then we conduct an ablation study for the proposed Monoceros. To observe the effects of individual components, we compare Monoceros with the methods using only the policy network (PN) and PN + Behavior Clone (BC). As shown in Tables IV-V, PN+BC outperforms PN (only) and Monoceros performs better than PN+BC, which indicates that the behavior clone and the proposed reward function in our approach can benefit the performance in all of the experimental scenarios. In addition, Figs. 9-11 shows the detailed performance in each round.

Finally, in order to demonstrate the performance intuitively, we record the live gaming videos of the agents trained by Monoceros in the three scenarios which can be viewed online (health gathering & health gathering (supreme): https://www.youtube.com/watch?v=dZMbR8AWMIY. limited death match (shotgun): https://www.youtube.com/watch?v=xxeJaEHKg0M).

VI. Conclusion

As the research on deep reinforcement learning goes deeper, the sparse reward problem starts to trouble the agent training, particularly in some high-dimensional information scenarios. Existing methods, e.g., reward shaping, curriculum training, and hierarchical reinforcement learning, have to rely hugely on

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

Scenario	Game Episode	Monoceros	PN(only)	PN+BC
Health gathering	20	3908.6 ±815.38	2626.6 ±1479.83	3336.2 ±1388.77
Health gathering (superme)	20	3069.4 ±1032.03	1933.05 ±968.84	2188.25 ±972.57

TABLE IV: Survive duration.

Scenario	Episode	Monoceros	PN(only)	PN+BC
Limited death match (shotgun)	10	3.438 ±0.192	2.409 ±0.08	3.088 ±0.112

TABLE V: Kill/death rate.

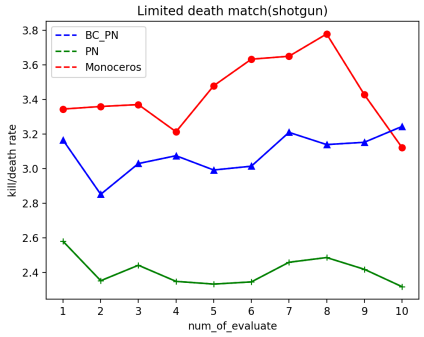


Fig. 11: Kill/death rate vs. round in the limited death match scenario

human factors. The methods from the inverse reinforcement learning can avoid human factors but only depend on the reward function learnt from expert trajectory data.

To overcome these deficiencies of existing methods, this paper proposes a new approach named Monoceros to training a game agent. Monoceros can deal with game scenarios with high-dimensional state information, and alleviate the sparse reward problem by presenting a new reward function. Experimental results show that the agent trained by Monoceros outperforms the ones trained by other state-of-the-art methods.

In the next research, we will further explore the other new design of the reward function to improve the generalization capacity of Monoceros.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (No. 61472408).

REFERENCES

[1] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, vol. abs/1701.07274, 2017.

[2] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *CoRR*, vol. abs/1811.12560, 2018.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. 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*, vol. 518, pp. 529–533, 02 2015.

[5] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR*, vol. abs/1507.06527, 2015.

[6] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 09 2016, pp. 1–8.

[7] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17, 2017, p. 2140–2146.

[8] D. S. Chaplot and G. Lample, "Arnold: An autonomous agent to play fps games," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17, 2017, p. 5085–8086.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, 2016, p. 1928–1937.

[10] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[11] D. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[12] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004.

[13] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, ser. AAAI'08, 2008, p. 1433–1438.

[14] C. Xia and A. E. Kamel, "Neural inverse reinforcement learning in autonomous navigation," *Robotics and Autonomous Systems*, vol. 84, pp. 1 – 14, 2016.

[15] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 4565–4573.

[16] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14, Cambridge, MA, USA, 2014, p. 2672–2680.

[17] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, 2015, p. 1889–1897.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[19] R. Wang, C. Ciliberto, P. V. Amadori, and Y. Demiris, "Random expert distillation: Imitation learning via expert policy support estimation," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019, pp. 6536–6544.

[20] Y. Burda, H. Edwards, A. J. Storkey, and O. Klimov, "Exploration by random network distillation," *CoRR*, vol. abs/1810.12894, 2018.