Accelerating Multi-Agent Learning via Centralized Counting and Efficient Hashing

Jianing Wang, Matthias Müller-Brockhausen, and Aske Plaat

Leiden Institute of Advanced Computer Science, Leiden University, Leiden, the Netherlands
j.wang.35@umail.leidenuniv.nl

Abstract. Exploration is crucial for learning in sparse reward environments such as continuous 2D Navigation or Communicative Navigation. The increased difficulty of multi- over single-agent tasks stems mainly from the increased number of entities requiring coordination and cooperation between each other. To improve cooperation during the exploration phase, we introduce an adaption of the Count-Based method that works centralized, containing all agents' information instead of decentralized. Moreover, we tune a hash function (SimHash) to reduce the high-dimensionality of the continuous navigation environment. With our method, we were able to be cut down training time by at least half.

Keywords: Multi-Agent Reinforcement learning · Exploration · 2D Navigation

1 Introduction

Learning can be associated with exploration and exploitation. Exploration refers to gaining new information and focusing on long-term gains. Exploitation utilizes current information to maximize short-term benefits. Efficacious exploration is crucial, especially in environments with sparse reward settings. The agents in these environments, with random exploration, barely achieve the tasks and receive learning signals, which is known as the sparse reward problem.

Figure 1 shows two 2D navigation environments with two agents and sparse reward settings. In the 2-Agent Navigation task, agents need to cooperate to cover both landmarks simultaneously. For the Communicative Navigation task, the speaker guides the listener towards the target landmark by uttering a communication signal. In both tasks, agents receive a learning signal only when the landmark is covered.

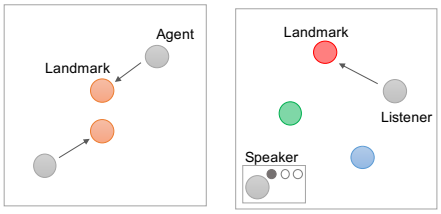


Fig. 1: Multi-agent environments with sparse reward settings: (a) 2-Agent Navigation, (b) Communicative Navigation.

The most common approach to deal with a sparse reward is by introducing an intrinsic reward as a bonus to encourage the agents to explore. For single-agent environments, this has been done extensively [21,26,27]. Since intrinsic reward methods work well in single-agent problems, we are interested in their performance in the multi-agent domain. Multi-agent reinforcement learning (MARL) has intrigued the interest of many researchers in recent years because many real-world applications are naturally modeled as multi-agent learning problems, such as team sport [11], multi-robot control [33], and autonomous vehicles [25]. Contrary to a single agent, cooperation is required to explore an environment efficiently with multiple agents.

To encourage the cooperation between agents, Jaques et al. [8] and Wang et al. [35] strengthen team coordination and communication by encouraging agents to choose actions with more social impact. Compared to their methods that focus on cooperation, we present an exploration method that encourages agents to explore the environment and collaborate with teammates simultaneously. Inspired by the centralized training method [14], we adopt a single-agent intrinsic reward method, Count-Based to Multi-Agent Count-Based (MACB), which considers the information of all the agents for counting. The idea behind the Count-Based method is that agents that can visit more different states in a limited time have a higher chance to find an optimal policy. By counting the occurrences of the joint observations and actions of all agents, the MACB method encourages the agent to visit the states that are new for itself and new for its teammates and therefore achieves simultaneous exploration for both environment and cooperation.

However, all joint observations and actions may only occur once because of the continuous state and action space, making it impossible to determine which state is relatively novel based on counting methods. To solve this problem, we consider using a hash function (SimHash) [32] to map similar state-action pairs to the same hash code before counting.

We evaluate our method with 2-Agent Navigation and Communicative Navigation, which are fully observable and partially observable, respectively. Our results show that the MACB method can help the agents receive the learning signals faster and therefore decrease the number of training episodes that the agents need to master the task by at least half. We also show that our method is easy to implement with existing multi-agent learning algorithms.

2 Related work

For simple RL problems, like MountainCar or CartPole, the basic exploration strategies guarantee finding the optimal decision [12,36]. The ϵ-greedy method [16,34] uses a probability of ϵ to randomly select an action for exploration and a probability of $(1-\epsilon)$ to choose an optimal action. Instead of choosing a random action with a certain probability, the noise-based methods [36] add random noise to action or parameter space directly [4,24]. Random exploration is easy to apply, but it is the least efficient strategy [15,33].

Intrinsic reward strategies are commonly used in hard to explore environments where the agents barely receive learning signals. The intrinsic reward strategies provide bonus rewards as learning signals to the agents through other criteria and therefore boost the progress of learning. Some studies [2,23,28] use the prediction error of different feature spaces to encourage the agents to visit the uncertain parts of the environment. Variational Information Maximizing Exploration (VIME) [6,17] encourages the agents to visit the states, minimizing the uncertainty of the environment dynamics distribution. Count-based methods such as MBIE [29] and MBIE-EB [30] encourage the agent to discover novel states using the state and action count.

Computer scientists took some more RL-related inspiration from the field of psychology [37] by introducing intrinsic rewards in order to encourage cooperative exploration in multi-agent problems. In Jaques et al. [8], the agents are encouraged to select the action which can influence the behavior of the other agents the most. The influence is calculated by how much the selected action can change the distribution over other agents' next actions. In Wang et al. [35], the agent is encouraged to visit the states where that influence the transition distribution of other agents the most. Iqbal et al. [7] use a hierarchical policy where the top-level agent chooses the best among five intrinsic reward functions, and the low-level agents follow this bonus to learn.

In the environments with continuous state and action space, some extensions of the Count-Based method in order to solve the high-dimensional state and action space problem include [1,20], where they propose using a density model to generate pseudo-counts and Tang et al. [32], where they use a hash function to decrease the dimensionality. Instead of the hash function, they also propose a learned hash model (an autoencoder [9,19]) to extract features from the state and reduce the dimensionality. Besides autoencoders, a convolutional neural network that can recognize the pattern of high-resolution images to solve classification tasks can also be used to extract the features [10].

3 Background

We consider an extension of MDP [31] called Markov Games (MGs) [13] to model the MARL problems. For an N agent RL problem, MGs are defined by a set of states S for all agents, sets of actions $A_1, ..., A_N$ and sets of observations $O_1, ..., O_N$ for each agents. The state transition function $P(s_{t+1}|s_t, x_t)$ considers actions from all the agents $x_t = (a_1^t, ..., a_N^t)$ and the state s_t is the concatenation of the observations of all the agents $s_t = (o_1^t, ..., o_N^t)$. We consider the cooperative tasks where all the agents receive the same reward $r_t = R(s_t, x_t)$. Agents aim to maximize the expected reward $R = \sum_{t=0}^{T}(\gamma^t r_t)$, where γ controls the effect that future rewards have on current decisions.

Centralized Critic Algorithm (MADDPG). The centralized critic technique is used to solve the non-stationarity problem in MARL [5,22]. The problem is that all individual policies continuously change during training, making it

impossible to explain the rewards received by each agent with their policies. The multi-agent deep deterministic policy gradient (MADDPG) [14] algorithm utilizes the information from other agents when training the action-value function (centralized critic) but only uses the local information when choosing actions (decentralized actor). Since each agent knows the information of other agents with a centralized critic, it can explain the changes of rewards caused by other agents. Specifically, each agent i has its own centralized action-value function $Q_i(s_t, x_t|\theta^{Q_i})$ which considers the states and actions from all the agents and aims to minimize the loss function:

$$\mathcal{L}(\theta^{Q_i}) = \mathbb{E}_{s_t, x_t, r_t, s_{t+1}}[(Q_i(s_t, x_t|\theta^{Q_i}) - y_t)^2], \tag{1}$$

where

$$y_t = r_t + \gamma \bar{Q}_i(s_{t+1}, \bar{\mu}_1(o_1^{t+1}), ..., \bar{\mu}_N(o_N^{t+1})). \tag{2}$$

The $\bar{Q}_i^{\mu}$ is a copy of Q_i^{μ} that slowly updates towards the critic. Each agent has its own actor $\mu_i(o_i)$, which only considers its local observations o_i. The gradient of the policies is given as:

$$\bigtriangledown_{\theta^{\mu_i}} J(\theta^{\mu_i}) = \mathbb{E}_{s\sim p^{\mu}, x\sim\mu}[\bigtriangledown_{a_i} Q_i(s, a_1, ..., a_i, ..., a_N)|_{a_i=\mu_i(o_i)} \bigtriangledown_{\theta^{\mu_i}} \mu_i(o_i)]. \tag{3}$$

Count-Based Exploration. The MADDPG method adds random action noise to achieve exploration. When an environment has a sparse reward setting, random exploration is the least efficient strategy and may cause the agents to repeatedly explore areas they have been before.

Instead of requiring the agents to complete a task, intrinsic reward methods give a bonus to the agents based on other criteria, such as visiting new states or gathering effective information. When training the policy, a new reward r_t' is used to update the action-value function. It includes an extrinsic reward r_t from the environment and an intrinsic reward r_t^+ [36]:

$$r_t' = r_t + \beta r_t^+ \tag{4}$$

where β is the bonus coefficient that balances exploration and exploitation. The Count-Based exploration strategy uses the state-action count to encourage the agent to visit new state-action. At the time t, the bonus r_t^+ equals the inverse square root count of the state-action pairs:

$$r_t^+(s_t, x_t) = \frac{1}{\sqrt{n(s_t, x_t)}} \tag{5}$$

where $n(s_t, x_t)$ is the number of times this state-action pair has occurred before. With the inverse count bonus, the agent is encouraged to visit the less-visited states. The count is stored in a tabular C.

4 Method

4.1 Multi-Agent Count-Based

The simplest way to adapt Count-Based to the multi-agent domain is assigning each agent i its own count table C_i that uses its local information $n(o_i, a_i)$ for counting. However, the agent may only focus on individual exploration by using the local information and neglect the search for different ways of cooperation with its teammates.

Inspired by the centralized training method, we propose a Multi-Agent Count-Based (MACB) strategy, which takes the joint observations and actions of all the agents $n(o_1, ..., o_N, a_1, ..., a_N)$ for counting and all the agents share a count table C. The joint observations unify all agents information in one central place, allowing cooperation by simultaneously exploring the environment. Sharing a count table can keep the exploration progress of all agents consistent, which helps them achieve the same learning process with the same amount of training.

However, if the environment is with continuous state and action space, all the joint observations and actions will only appear one time. The Count-Based method becomes meaningless if we cannot tell which joint is less visited. This further causes higher storage memory and searching time problems with the count table.

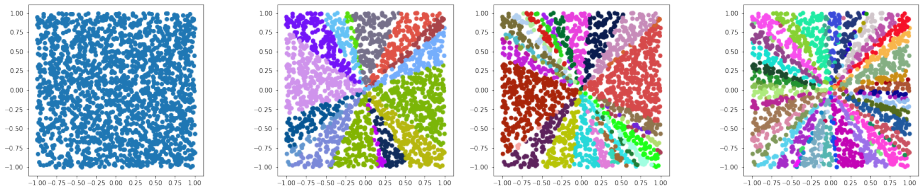
4.2 SimHash Function

Learning from [32], we utilize the SimHash [3] function to discretize a concatenated state-action pair $s||x$ into a k length hash code in the form of $\{-1, 0, 1\}^k$, and use the hash code for counting. The main idea is to map similar state-action pairs into the same hash code. The SimHash function $\phi(s||x)$ discretizes the state-action by the angular distance:

$$\phi(s||x) = sgn(A \cdot s||x) \in \{-1, 0, 1\}^k, \tag{6}$$

where A is a $k \times D$ matrix with i.i.d. entries sampled from a Gaussian distribution, where D is the size of the state-action $s||x$ and k is the length of the hash code which controls the granularity. To demonstrate how the SimHash function maps similar states into the same code, we randomly draw 2000 points in range $(-1, 1)$ and show the grouping results based on their position with $k = 8, 16, 32$. Figure 2 shows how the SimHash function groups 2-dimensional points angularly. With a larger k, the hash code is longer, and fewer state-action pairs map to the same code. If the hash code is too short, useful information can be lost, which can affect the learning process negatively. Therefore, a suitable k needs to be chosen for optimal results.

After decreasing the scale of state-action pairs using the SimHash function, we can use the corresponding hash code in the MACB strategy. The intrinsic reward



(a) 2000 points (b) $k = 8$ (c) $k = 16$ (d) $k = 32$

Fig. 2: Using a SimHash function to group 2000 points with $k = 8, 16, 32$. Points mapped to the same hash code are grouped in the same color.

is calculated by:

$$r_t^+(s_t, x_t) = \frac{1}{\sqrt{n(\phi(s_t||x_t))}} \tag{7}$$

The pseudo-code of MACB with the SimHash function is shown in Algorithm 1. We update the count of a joint state-action pair in table C after collecting a transition and calculate the new reward after sampling a random transition. The MACB strategy may fail if we update the count after sampling a transition because this transition can be sampled multiple times during training, which will cause the count to increase too quickly, and the intrinsic bonus will vanish after the first few episodes. In addition, the intrinsic reward should not be included in the replay buffer because identical transitions in the replay buffer will have different rewards, leading to inaccuracies as earlier transitions will not have the corresponding rewards for the current situation.

Algorithm 1: Multi-Agent Count-based (MACB)

Initialize multi-agent learning algorithm (e.g. MADDPG)
Initialize an empty hash tables C where the new key initialize with value 0
Initialize hyper-parameters β for trade-off and k for hash code granularity
Initialize matrix $A \in \mathbb{R}^{k \times D}$ with i.i.d. entries sample from a normal distribution
for *episode* $= 1$ *to* M **do**
  for $t = 0$ *to* T **do**
    Collect transition (s_t, x_t, r_t, s_{t+1}) and store in the replay buffers
    Compute hash code using SimHash function
     $\phi(s_t||x_t) = sgn(A \cdot s_t||x_t) \in \{-1, 1\}^k$
    Update the count in the table C, $n(\phi(s_t||x_t)) = n(\phi(s_t||x_t)) + 1$
    for *agent* $i = 1$ *to* N **do**
      Sample a minibatch of transitions (s_j, x_j, r_j, s_{j+1}) from replay buffers
      Compute hash code of each state-action pair $\phi(s_j||x_j)$
      Calculate the new reward $r'_j = r_j + \beta r_j^+$ where $r_j^+ = \frac{1}{\sqrt{n(\phi(s_j||x_j))}}$
      Update critic and actor using the new reward r'_j
    end
  end
end

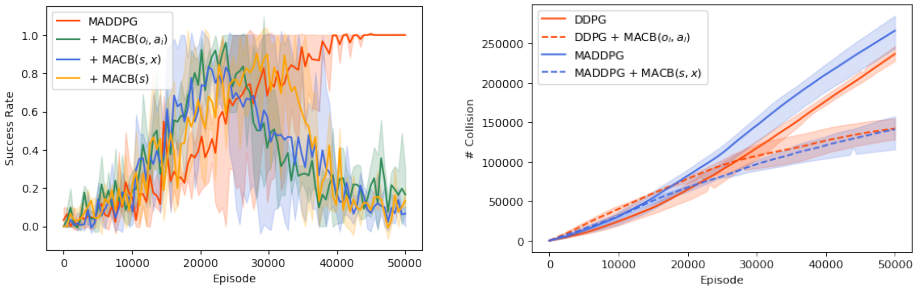
5 Experiments

We evaluate MACB in 2 different cooperative multi-agent tasks, 2-Agent Navigation and Communicative Navigation [18], as shown in Figure 1. Both of the environments are 2-dimensional with a continuous state and action space. 2-Agent Navigation is fully observable, which means that the agent can see all relevant information that it needs to make a decision. Communicative Navigation is a partially observed environment, where only the speaker knows which landmark is the target, and the listener has to decipher it based on the speaker's signal.

In both tasks, we set 20 timesteps for each episode. After an episode of training, we run ten more episodes without random action noise for evaluation. All the results are smoothed and averaged over three random seeds with a 75% confidence interval. Our code can be found in Github [1] and we include the hyper-parameters for the learning algorithm in Appendix A.

5.1 Performance in the fully observed environment

Figure 3a shows the average success rate of MADDPG with and without MACB on the partially observed 2-Agent Navigation task. Without the help of MACB, MADDPG learns gradually over episodes and fully grasps the problem at around 4×10^4 episodes. All 3 MACB variants accelerate learning and reach success rate convergence earlier (2 at 2×10^4 and 1 at 3×10^4 episodes). However, if we continue training after convergence, the success rate gradually decreases, which can be addressed by using Early Stopping.



(a) Success rate

(b) Number of collisions

Fig. 3: (a) The success rate in the 2-Agent Navigation problem with and without MACB exploration. (b) The number of collisions of DDPG and MADDPG with and without the MACB method. The MACB method can promote the learning process and decrease the number of collisions.

Since the environment is already fully observable, the performance of local [MACB(o_i, a_i)] vs global [MACB(s, x)] information does not differ much. Moreover, contrary to [32], state-action pair counting improves performance in our

[1] https://github.com/JianingWang99/CentralizedCountBased

experiments. The probable reason for this difference is that [32] applied it in a single-agent environment. Furthermore, in comparison to [32], where the state-action pair counting has almost the same result as only counting the state, our results show that counting the state-action pairs has a better performance. This may be because there are two agents in the environment and the action information is more important than in a single-agent environment.

We compare the accumulated number of collisions between DDPG and MADDPG with and without MACB in Figure 3b. According to [14], the MADDPG agents only have half of the number of collisions than DDPG agents, but our result shows that the number of collisions of the MADDPG is more than that of DDPG. However, after applying the MACB strategies, the accumulated number of collisions vastly decreases. These results indicate that the MACB exploration can help the agents find the optimal cooperation strategies within fewer episodes, and in turn, the total number of collisions decreases.

5.2 Performance in the partially observed environment

Figure 4 shows the success rate of MACB strategies in the partially observed environment with 2 different reward settings. With sparse rewards, the MADDPG agents learn slowly and only surpasses a 40% success rate after 4×10^4 episodes of training. While with the help of MACB(o_i, a_i) the success rate increases to around 70% at 4×10^4 episodes, the agents with MACB(s, x) can reach 100% success rate with only 1.5×10^4 episodes. Both MACB strategies have the same β (0.8) and k (512). This underlines the positive effect of centralization in partially observed environments.

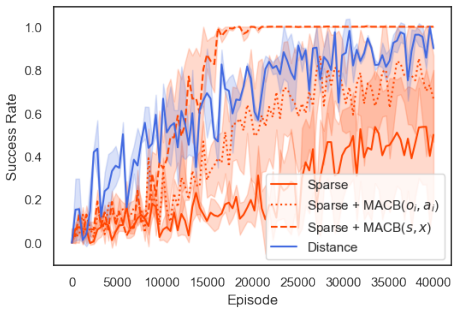


Fig. 4: The success rate of MADDPG and MACB strategies in the Communicative Navigation with sparse and distance reward settings.

In the sparse reward setting, the agent requires more time before receiving a steady learning signal. Distance rewards seem to mitigate this and enabling instantaneous learning. We can see that the MACB method can reduce the number of episodes that the agents require to receive learning signals. And once the agent starts learning, the success rate increases faster than the agent in the dense reward environment.

5.3 Trade-off between exploration and exploitation

Table 1 concludes the average success rate and the count-1 percentage (sr, c-1) after 4×10^4 episodes of training with different combinations of β and k. The

Count-1 percentage reflects how many state-action pairs only appear one time in the count table. We evaluate using the Communicative Navigation environment.

Without the help of MACB, the MADDPG agents reach a 47% success rate. With the help of MACB, most of the time, the success rate is higher than 47% with different hyper-parameters. When k is 32, the success rate just increases a little with different β values. The results indicate that a hash code will lose important and relevant information if k is too small, making it slower to learn the task. We also see that the count-1 percentage increases as k , and when k is 512, the count-1 percentage converges to 100%, and the success rates surpass 75% in most of the cases.

When tuning k, we can increase the value of k until its count-1 percentage converges at 100%. When k is too big, the hash function may lose its meaning and the search time and storage memory becomes high. With a larger value of k, the agents need to explore more state-action pairs and require a larger β. In addition, we control β smaller than 1 because we do not want the exploration bonus to overwhelm the extrinsic reward, as it would lead to the agents only exploring and never exploiting.

Table 1: The table concludes the success rate (sr) and count-1 percentage (c-1) of the MACB with MADDPG in the Communicative Navigation with different ratios of exploration (β) and length of hash code (k).

β / k	**0.0**	**0.05**	**0.2**	**0.4**	**0.8**
	sr, c-1	sr, c-1	sr, c-1	sr, c-1	sr, c-1
-	47%, -	-	-	-	-
32	-	59%, 67%	60%, 68%	56%, 71%	53%, 74%
64	-	**72%**, 91%	60%, 91%	34%,91%	**72%**, 94%
256	-	25%, 99%	**82%**, 99%	43%,99%	**81%**, 99%
512	-	**75%**,100%	48%,100%	**75%**,100%	**100%**,100%

6 Conclusion

Our work has succeeded in improving the exploration in multi-agent environments with a sparse reward setting, specifically: 2-Agent Navigation and Communicative Navigation (see Figure 1). By centralizing the count table of our Count-Based method, we have improved cooperation between agents. Moreover, we have successfully reduced the high-dimensionality of the continuous environment without losing training-relevant information by applying SimHash. After tuning its two parameters β and k, we were able to accelerate learning dramatically. For future work, there are two interesting paths to pursue. First, it remains to be seen what other multi-agent tasks our extensions work well on and how they perform in settings with more than just two agents. Second, solving the same task solely from an image input. SimHash requires knowledge about the entities' position, so an auto-encoder extracting them would be required. Alternatively, one could utilize a density model to predict the pseudo-counts of state-action pairs instead of using a table to record the counts directly.

References

1. Bellemare, M.G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R.: Unifying count-based exploration and intrinsic motivation. arXiv preprint arXiv:1606.01868 (2016)
2. Burda, Y., Edwards, H., Storkey, A., Klimov, O.: Exploration by random network distillation. arXiv preprint arXiv:1810.12894 (2018)
3. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. pp. 380–388 (2002)
4. Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al.: Noisy networks for exploration. arXiv preprint arXiv:1706.10295 (2017)
5. Gronauer, S., Diepold, K.: Multi-agent deep reinforcement learning: a survey. Artificial Intelligence Review pp. 1–49 (2021)
6. Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., Abbeel, P.: Vime: Variational information maximizing exploration. arXiv preprint arXiv:1605.09674 (2016)
7. Iqbal, S., Sha, F.: Coordinated exploration via intrinsic rewards for multi-agent reinforcement learning. arXiv preprint arXiv:1905.12127 (2019)
8. Jaques, N., Lazaridou, A., Hughes, E., Gulcehre, C., Ortega, P., Strouse, D., Leibo, J.Z., De Freitas, N.: Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In: International Conference on Machine Learning. pp. 3040–3049. PMLR (2019)
9. Krizhevsky, A., Hinton, G.E.: Using very deep autoencoders for content-based image retrieval. In: ESANN. vol. 1, p. 2. Citeseer (2011)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25**, 1097–1105 (2012)
11. Kurach, K., Raichuk, A., Stańczyk, P., Zajac, M., Bachem, O., Espeholt, L., Riquelme, C., Vincent, D., Michalski, M., Bousquet, O., et al.: Google research football: A novel reinforcement learning environment. arXiv preprint arXiv:1907.11180 (2019)
12. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
13. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Machine learning proceedings 1994, pp. 157–163. Elsevier (1994)
14. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. arXiv preprint arXiv:1706.02275 (2017)
15. McFarlane, R.: A survey of exploration strategies in reinforcement learning. McGill University (2018)
16. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International conference on machine learning. pp. 1928–1937. PMLR (2016)
17. Mohamed, S., Rezende, D.J.: Variational information maximisation for intrinsically motivated reinforcement learning. arXiv preprint arXiv:1509.08731 (2015)
18. Mordatch, I., Abbeel, P.: Emergence of grounded compositional language in multi-agent populations. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (2018)

19. Ng, A., et al.: Sparse autoencoder. CS294A Lecture notes **72**(2011), 1–19 (2011)
20. Ostrovski, G., Bellemare, M.G., Oord, A., Munos, R.: Count-based exploration with neural density models. In: International conference on machine learning. pp. 2721–2730. PMLR (2017)
21. Oudeyer, P.Y., Kaplan, F.: What is intrinsic motivation? a typology of computational approaches. Frontiers in neurorobotics **1**, 6 (2009)
22. Papoudakis, G., Christianos, F., Rahman, A., Albrecht, S.V.: Dealing with non-stationarity in multi-agent deep reinforcement learning. arXiv preprint arXiv:1906.04737 (2019)
23. Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. In: International Conference on Machine Learning. pp. 2778–2787. PMLR (2017)
24. Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P., Andrychowicz, M.: Parameter space noise for exploration. arXiv preprint arXiv:1706.01905 (2017)
25. Shalev-Shwartz, S., Shammah, S., Shashua, A.: Safe, multi-agent, reinforcement learning for autonomous driving. arXiv preprint arXiv:1610.03295 (2016)
26. Singh, S., Barto, A.G., Chentanez, N.: Intrinsically motivated reinforcement learning. Tech. rep., MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER SCIENCE (2005)
27. Singh, S., Lewis, R.L., Barto, A.G., Sorg, J.: Intrinsically motivated reinforcement learning: An evolutionary perspective. IEEE Transactions on Autonomous Mental Development **2**(2), 70–82 (2010)
28. Stadie, B.C., Levine, S., Abbeel, P.: Incentivizing exploration in reinforcement learning with deep predictive models. arXiv preprint arXiv:1507.00814 (2015)
29. Strehl, A.L., Littman, M.L.: A theoretical analysis of model-based interval estimation. In: Proceedings of the 22nd international conference on Machine learning. pp. 856–863 (2005)
30. Strehl, A.L., Littman, M.L.: An analysis of model-based interval estimation for markov decision processes. Journal of Computer and System Sciences **74**(8), 1309–1331 (2008)
31. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
32. Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., Abbeel, P.: # exploration: A study of count-based exploration for deep reinforcement learning. In: 31st Conference on Neural Information Processing Systems (NIPS). vol. 30, pp. 1–18 (2017)
33. Thrun, S.B.: Efficient exploration in reinforcement learning (1992)
34. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence. vol. 30 (2016)
35. Wang, T., Wang, J., Wu, Y., Zhang, C.: Influence-based multi-agent exploration. arXiv preprint arXiv:1910.05512 (2019)
36. Weng, L.: Exploration strategies in deep reinforcement learning. lilianweng.github.io/lil-log (2020), https://lilianweng.github.io/lil-log/2020/06/07/exploration-strategies-in-deep-reinforcement-learning.html
37. Wiersma, U.J.: The effects of extrinsic rewards in intrinsic motivation: A meta-analysis. Journal of occupational and organizational psychology **65**(2), 101–114 (1992)

A Appendix

A.1 Training Details

The networks for actors and critics are with two hidden layers with the size of 400 and 300. The activation function for both actor and critic is ReLU. After collecting 100 transitions, we begin updating the network parameters at each time step. An episode consists of 20 time steps. After an episode of training, we evaluate the algorithms with 10 more episodes without exploration action noises. All the results are averaged over 3 random seeds. The hyper-parameters used in the experiments are summarized in Table 2.

Table 2: Hyper-parameters used in experiments

Hyper-parameter	Value
Buffer size	10^6
Batch size	100
Time-step per Episode	20
Learning rate for optimizer	0.001
γ	0.99
τ	0.005