AIAA SCITECH 2022 Forum, 3-7 January 2022, San Diego, California USA & Virtual
DOI:10.2514/6.2022-2498

Self play with parameter sharing in n-player mixed competitive-cooperative games

George Marios Skaltsis* and Hyo-Sang Shin.†
School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield, MK43 OAL, U.K.

Antonios Tsourdos‡
School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield, MK43 OAL, U.K.

We introduce some parameter sharing multi-agent reinforcement learning schemes, combined with self-play for n-players mixed competitive-cooperative games. Except for the pure self-play scheme, an another one using the best policy, outperform the pure parameter sharing baseline, leading to better exploration of the state space and protecting from the performance deterioration observed with the baseline.

I. Introduction

THE last few years the usage and research of multi agent systems has increased a lot, since such systems can perform better than single agent systems in many scenarios, as well as the increase of computational power has facilitate the development and usage of multi-agent algorithms. The multi-agent systems have the advantage to perform complex tasks, exploiting the cooperation or competitiveness between the agents to achieve higher performance and credibility. Moreover, such systems are usually faster than single agent ones and can be more robust to agents malfunctions, something very important for critical tasks.

The agents used in such systems could be UAVs, UGVs or UUVs, other types or robots or even computer systems [1]. When using a system with multiple agents the problem of division of labour arises or in other words there is a need for task allocation techniques. The task allocation techniques goal could be to find what tasks will be assigned to which agents, the type of communication that might be used and generally how the agents should be coordinated in order to achieve optimal and robust overall performance [2],[3]. Some goals of task allocation procedure, contributing to the main goal of maximising the overall performance, might also include the minimization of the time the agents are inactive and the time needed for tasks to be executed, the maximisation of the number of completed tasks in certain time frames, as well as the reliability and robustness of the task allocation procedure.

The main approaches used to solve task allocation problems include market based approaches like CBBA and CNP based techniques, metaheuristics, heuristics, game theory based, as well as multi-agent reinforcement learning (MARL) approaches, that are the focus of this paper. Especially in the case of the UAVs, that is the focus of this work, some common applications include, search and rescue missions (SAR), military applications like ISR (intelligence, surveillance and reconnaissance) , target tracking and destruction , mobile edge computing (MEC), area coverage, autonomous coordination and others.

A. Reinforcement learning (RL) techniques

A relatively newer and very promising technique used in task allocation problems is reinforcement learning. In reinforcement learning the agents learn what actions to choose by interacting with other agents and the environment surrounding them, taking positive or negative rewards based on the actions they choose while being in a specific state of the environment. The reinforcement learning algorithms can be model based, in the case there is a model of the environment providing the transition probability of states, rewards etc or model free where there is no model of the environment and is also the focus of this work.[4]. The environment is usually formed as a Markov Decision Process (MDP) and the agents' goal is to optimise a reward function, improving their overall performance.

The model free methods can be divided in value based methods, policy based methods and actor-critic methods. The value based method try to compute an estimate of the value faction of the state $V(s)$ or the state-action value function

*Phd Researcher, School of Aerospace, Transport and Manufacturing, george.skaltsis@cranfield.ac.uk.
†Professor, School of Aerospace, Transport and Manufacturing, h.shin@cranfield.ac.uk.
‡Professor, School of Aerospace, Transport and Manufacturing, a.tsourdos@cranfield.ac.uk.

$Q(s, a)$ and use a fixed rule to create a mapping from these value functions to actions need to be chosen. A well known algorithm belonging to this category is the Q-learning method. Policy based methods do not need estimates of the value functions, but instead they compute probability distribution of actions over states $\pi_\theta = Pr[a|s]$, by using the parameterised policy π_θ and neural networks in deep learning case. Known policy based algorithms include policy gradients, REINFORCE and others. The actor-critic methods are a combination of the aforementioned methods, with the critic estimating a value function and the actor using a parameterised policy to update the action distribution as guidelined by the critic, using a policy gradient. Some actor-critic methods include TRPO, PPO which is the method used in this paper, A3C, DDPG and others [4].

Reinforcement learning methods need no specific mathematical models of the environment and are very good in dealing with disturbances, something crucial for dynamic environments like most of the real world applications. They generally have high efficiency, might be online implementable (for well trained networks) and have good behaviour to environmental disturbances. On the other side they can be computationally expensive for large scale systems [5],[6],[7]. Generally reinforcement learning techniques can have better performance than some metaheuristics like simulated annealing and greedy algorithms and optimisation based techniques like the Hungarian method. Nevertheless an existing problem remains the increase in the dimensionality and the computational cost of some methods[8].

The method used in this paper is Proximal Policy Optimization algorithm (PPO), that is an actor critic, on policy method originated in [9]. PPO is based on the same principle as Trust Region Policy Optimization (TRPO), aiming to find the biggest improvement step on a policy, using on policy data, but without getting too far, causing a performance deterioration. PPO achieves that in a simpler, less complex way than TRPO, using only first order optimisation. Generally PPO has the stability and reliability of TRPO, but it is simpler to implement and can have better overall performance.

B. Multiagent Reinforcement Learning (MARL)

Learning in a multi-agent environment is way more complex and difficult than in the single agent case. There are a lot of problem like the non-stationarity of the environment, since the other agents policies are changing and affect the environment, the curse of dimensionality as the number of agents increases, multi-agent credit assignment, global exploration etc. [10].

In MARL there are a lot of frameworks for learning. Independent learners or decentralised learning and execution uses single agent algorithms, even though the Markov property is violated, assuming the other agents as part of the environment and it is a method used frequently, mainly for scalability purposes. In centralised learning and execution a central agent is assumed that gathers all the observations , actions and rewards from all others agents, while all the agents take the same actions, something not suitable to larger environments. [11].

The most common paradigm is the centralised learning with decentralised execution, where a central agent is assumed that gathers information from all others agents, but the execution is decentralised, meaning that the policies use only local agents' information. A common approach to achieve that is for example, through a centralised critic and a decentralised actor or parameter sharing. In this paper we will use parameter sharing [12], [13]. Parameter sharing can use single agent RL methods in a multi-agent setting, by using a single shared policy for all the agents, using the experiences of all the agents simultaneously. Nevertheless, even though the learning is centralised the execution is not, because each of the agents has its own observations, including their respective index.

C. Stochastic games

Even though Multi-agent MDPs (MMDPs) extend the MDPs formulation to a multiagent setting, they assume that all agents receive the same reward. Therefore a wider extension of MDPs to a multiagent setup are stochastic games (SGs) that permit each agent to have a unique reward function [13]. A stochastic game is defined by the tuple $\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}$ where:

- $\mathcal{N}$ is the number of agents in the environment
- $\mathcal{S}$ is the number of the states
- $\mathcal{A}_i$ is the set of possible actions of agent i
- $\mathcal{P} : \times \Pi_i \in_{[\mathcal{N}]} \mathcal{A}_i \times \mathcal{S} \rightarrow [0, 1]$, is the transition function and $|\mathcal{N}|$ is the set of agents
- $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \ldots \mathcal{A}_N \times \mathcal{S} \rightarrow \mathbb{R}$, is the reward function for agent i

1. Partially-Observable Stochastic Games

In a partially observable stochastic game the agents have not access to the exact state of the environment, but only to an observation of that state through an observation function [14]. Therefore, a partially observable stochastic games is defined except for aforementioned terms of the stochastic game, with the terms Ω_i ,O, where:

- Ω_i is the set of possible observations for agent i
- $O_i : \mathcal{A}_i \times \mathcal{S} \times \Omega_i \rightarrow [0, 1]$, is the observation function

D. Self-play based techniques

The self-play scheme is a MARL framework that has shown very good results in two players zero sum games. In this scheme the agent or agents are trained with their past selves or policies that have been generated, during training, in the past. In recent approaches it is not necessary to use only the last policy, but a set of previous policies can be created, from which the self-play policy will be sampled. In the zero sum games case, the agents with this scheme try to defeat their older selves (or policies), achieving continual learning [15].

In the general case we can say that the self-play scheme envelops the general MARL loop and extends it by adding extra features in the beginning and the end of the loop. Defining as $\Pi = \Pi_1 \times \ldots \times \Pi_n$ the joint policy space, where Π_i is the policy space for agent i and pi is the current policy. Following the definition of self play for general sum partially observable stochastic games (POSG) in [15] the main components of self play are the *menagerie* $\boldsymbol{\pi^o}$, the *policy sampling distribution* Ω, and the *gating function* G. Specified by the tuple $< \Omega(\cdot|\cdot,\cdot), G(\cdot|\cdot,\cdot) >$:

- $\boldsymbol{\pi^o} \subseteq \Pi_i$; The **menagerie** is the set of the previous policies that can be used.
- $\Omega(\boldsymbol{\pi'} \in \Pi_{-i}|\boldsymbol{\pi^o} \subseteq \Pi_i, \pi \in \Pi_i) \in [0, 1]$; where $\boldsymbol{\pi'} \subseteq \boldsymbol{\pi^o}$; The **policy sampling distribution**, is a statistical distribution that chooses a policy from the menagerie that will be used in self play.
- $G(\boldsymbol{\pi^{o\prime}} \subseteq \Pi_i|\boldsymbol{\pi^o} \subseteq \Pi_i, \pi \in \Pi_i) \in [0, 1]$; The **curator** or **gating function**, of the menagerie, that is a function that chooses with what criterion a policy enters or is discarded from the menagerie.

In algorithm 1 we can see the reinforcement training loop with the addition of the self-play scheme for general sum games [15].

Algorithm 1: (POSG) RL Loop with Self-Play.

Input: *Environment*: $(S, A, O, \mathcal{P}(\cdot,\cdot|\cdot,\cdot), \mathcal{R}(\cdot,\cdot), \rho_0)$
Input: *Self-Play Scheme*: $(\Omega(\cdot|\cdot,\cdot), G(\cdot|\cdot,\cdot))$
Input: *Policy to be trained*: $\pi \in \Pi_i$

$\boldsymbol{\pi^o} = \{\pi\}$; // Menagerie initialization
for $e = 0, 1, 2, \ldots$ **do**
  $\boldsymbol{\pi'} \sim \Omega(\boldsymbol{\pi^o}, \pi)$; // Sample from menagerie
  $\boldsymbol{\pi} = \boldsymbol{\pi'} \cup \{\pi\}$;
  $s_0, \boldsymbol{o_0} \sim \rho_0$;
  for $t = 0, \ldots,$ *termination* **do**
    $\boldsymbol{a_t} \sim \boldsymbol{\pi}(\boldsymbol{o_t})$; // Classical MARL loop
    $s_{t+1}, \boldsymbol{o_{t+1}} \sim P(s_t, \boldsymbol{a_t})$;
    $\boldsymbol{r_t} \sim \boldsymbol{R}(s_t, \boldsymbol{a_t})$;
    $t \leftarrow t + 1$;
  end
  $\pi \leftarrow update(\pi)$;
  $\boldsymbol{\pi^o} \sim G(\boldsymbol{\pi^o}, \pi)$; // Curate menagerie
end
return π;

The single agent RL method used in this experiment is the Proximal Policy Optimisation algorithm (PPO). PPO is a state of the art, on policy, policy gradient method that has shown very good results in a lot of applications and is generally used as a baseline in a lot of research efforts [9]. The hyperparameters of the PPO algorithm used in the simulations are shown in table 1.

Hyperparameter	Value
sample_batch_size	200
train_batch_size	4000
sgd_minibatch_size	128
lambda	1
kl_coeff	0.2
entropy_coeff	0
num_sgd_iter	30
vf_clip_param	10.0
clip_param	0.3
batch_mode	truncate_episodes

Table 1 Hyperparameters for PPO algorithm

II. Simulation

A. Simulation environment

The environment used is the 'Adversarial Pursuit' environment from Pettingzoo [16], originated in the MAgent codebase [17]. This environment is a two dimensional partially observable one that has two type of agents, the red or predators that are larger and slower than the blue agents or preys. Also the number of predators is half the number of the preys. The predators goal is to navigate through the fixed grey obstacles and attack or tag the prey that tries to avoid them, since agents cannot be killed. The predators' also observation space is a circle of radius 5 while the prey has radius 4. The predators actions are to either move, attack or turn and the prey can only move or turn. The agents take +1 reward for tagging a prey while the prey takes -1. Also the predators take -0.2 on every attack regardless if they tag prey, in order to decrease the number of pointless attacks and encircle better the prey. The agent in each team must cooperate to defeat the other team making this is a mixed competitive-cooperative environment. The predators because are slower have to cooperate to encircle prey and tag it, something that requires implicit communication of the predators. Also, the reward is allocated individually only. The table 2 summarizes the key characteristics of the environment.

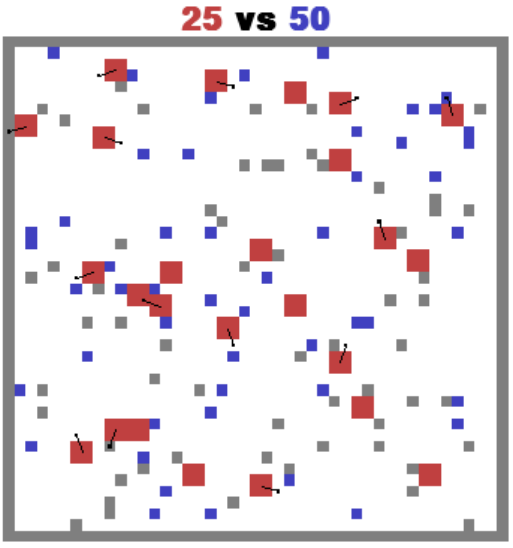


Fig. 1 The "Adversarial Pursuit environment"

This setting could be assumed as a simplified 2d version of a SAR application for UAVs (for example marine-time SAR, since there are moving targets), or a surveillance and target tracking and attacking problem for UAVs. The settings used is 25 predators with 50 preys. The simulation framework used is the RLlib framework [18] and Python.

Agent	Number	Observation range	reward	attack	attack penalty
predator	25	5	+1	Yes	-0.2
prey	50	4	-1	No	NA

Table 2 Hyperparameters of Adversarial Pursuit envirnmnet

B. Algorithms used

Being inspired from the self-play schemes in two players competitive zero-sum games we try to use here a similar approach for mixed competitive cooperative n-players games like the "Adversarial Pursuit" environment. In every case the PPO algorithm with parameter sharing is used as the learning framework. Two self-play schemes are used and in every case case there are two policies, one for each type of agents. As baseline is used the parameter sharing PPO for both types of agents.

The proposed methods are used in the predators team, while the prey keeps training in baseline PPO. The first framework used is the naive self-play, where a menagerie with 10 policies is used and in every iteration the current policy enters the menagerie and the oldest policy inside the menagerie is discarded. The second scheme, or maximum reward scheme, has also a menagerie size of 10, but here the mean policy reward is used in the gating function. The current policy enters the menagerie only if it has highest reward than the policy of the menagerie with the lowest reward. When this policy enters the menagerie the one with the lowest mean reward is discarded, so the gating function here is the max function. The sampling distribution used in both frameworks is the uniform distribution (the policies are sampled randomly).

III. Discussion

The goal of this paper is to find out the performance of the self-play proposed schemes in a mixed competitive-cooperative n-players game, since so far most of the self-play schemes used have been applied on competitive games. All graphs presented below, plot the mean episode reward of the environment and are averaged over 10 trials. The shaded region represents the standard deviation.

In Fig. 2, we see that both of the self play schemes have better performance than the baseline method. The max reward self-play scheme reaches very close to the ideal reward of -25, so the predators are able to tag almost all the prey and have a small number of tagging without purpose that punished with -0.2. The naive self-play scheme has very good performance as well, but slightly worse than the max reward scheme. The baseline method has the worst performance of all and shows performance collapse while the other methods are still improving. Some of the naive self-play trials demonstrated performance collapse but on a later stage than the baseline, while it was not observed in any of the trials of the max reward scheme.

We believe that this performance improvement with the self-play schemes might occur because the environment is highly dynamic and self-play adds randomness, improving the exploration of agents, leading to broader or different exploration space. Also, especially in the max reward scheme the goal of 'continual learning' is accomplished, by using every time the best policy, without creating bias to the policy space at the same time. In Fig.3 we see more clearly the deviation of the trials of the methods, with the max reward scheme to be the stablest one and the parameter sharing scheme to have the highest fluctuations.

Our goal is to further research the effect of the menagerie size, trying different sizes in order to conclude how the menagerie size affects performance. Moreover, since in this paper random sampling is used, it would be very interesting to see the performance with different more complex sampling distributions, or sampling the self-play policy from different past trials in order to improve exploration. Also, we intend to try different environments and hyperparameters, simulating environments with higher degree of cooperative behaviours, in order to find out the effect of self-play while increasing the level of cooperation in an environment.

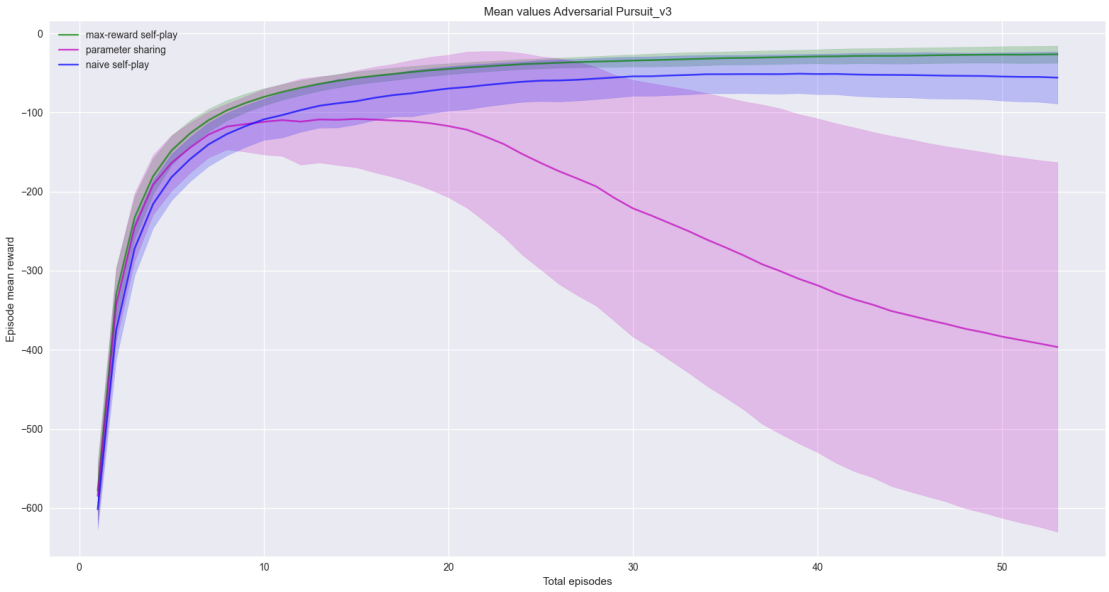
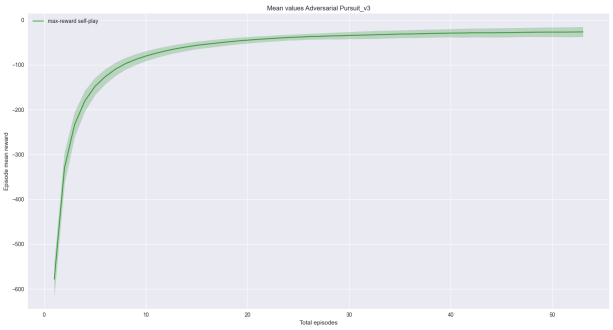
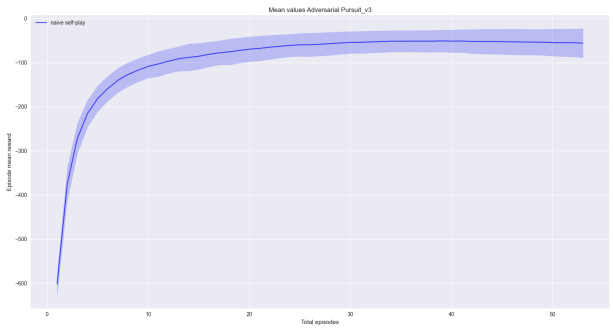


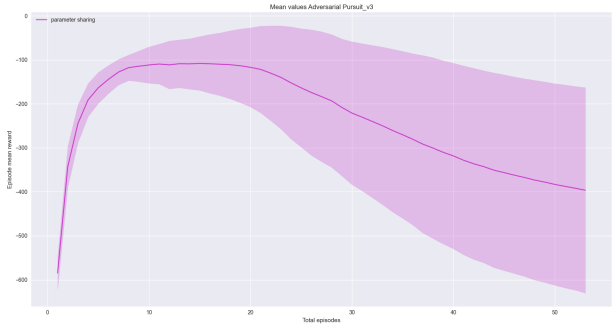
Fig. 2 Average mean episode reward trajectories. The plots are averaged over 10 runs with the shaded region representing the standard deviation



(a) Average episode reward and standard deviation for max reward self-play



(b) Average episode reward and standard deviation for naive self-play



(c) Average episode reward and standard deviation for parameter sharing

Fig. 3 Episode mean reward trajectory and standard deviation.

IV. Conclusion

In this research effort we experimented with the self-play usage to one n-player mixed competitive cooperative environment and proposed a simple and effective heuristic for the self-play scheme.The results show that self-play schemes improve performance in this kind of games and that our proposed max reward method performed better in terms of performance that pure self-play and the baseline parameter sharing. Our goal is to search for more effective and complex heuristics, regarding the gating function and the policy sampling distribution as well as to apply this methods to other environments with higher degree of cooperation, in order to evaluate the performance of self play with different degree of cooperation between the agents. Another interesting research direction is to find out more in detail the role of different menagerie sizes to the overall performance.

Acknowledgments

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-19-1-7032. Any opinions finding and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

References

[1] Rizk, Y., Awad, M., and Tunstel, E. W., "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys*, Vol. 52, No. 2, 2019. https://doi.org/10.1145/3303848.

[2] Khamis, A., Hussein, A., and Elmogy, A., "Multi-robot Task Allocation: A Review of the State-of-the-Art," *Studies in Computational Intelligence*, Vol. 604, 2015, pp. 31–51. https://doi.org/10.1007/978-3-319-18299-5_2, URL http://link.springer.com/10.1007/978-3-319-18299-5{_}2.

[3] Gerkey, B. P., and Matarić, M. J., "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, Vol. 23, No. 9, 2004, pp. 939–954. https://doi.org/10.1177/0278364904045564.

[4] Canese, L., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., and Spanò, S., "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences (Switzerland)*, Vol. 11, No. 11, 2021. https://doi.org/10.3390/app11114948.

[5] Tian, Y. T., Yang, M., Qi, X. Y., and Yang, Y. M., "Multi-robot task allocation for fire-disaster response based on reinforcement learning," *Proceedings of the 2009 International Conference on Machine Learning and Cybernetics*, Vol. 4, No. July, 2009, pp. 2312–2317. https://doi.org/10.1109/ICMLC.2009.5212216.

[6] Noureddine, D. B., Gharbi, A., and Ahmed, S. B., "Multi-agent deep reinforcement learning for task allocation in dynamic environment," *ICSOFT 2017 - Proceedings of the 12th International Conference on Software Technologies*, , No. Icsoft, 2017, pp. 17–26. https://doi.org/10.5220/0006393400170026.

[7] Dahl, T. S., MatariĆ, M. J., and Sukhatme, G. S., "A Machine Learning Method for Improving Task Allocation in Distributed Multi-Robot Transportation," *Complex Engineered Systems*, Vol. 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 307–337. https://doi.org/10.1007/3-540-32834-3_14, URL http://link.springer.com/10.1007/3-540-32834-3{_}14.

[8] Skaltsis, G. M., Shin, H.-S., and Tsourdos, A., "A survey of task allocation techniques in MAS," *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, pp. 488–497. https://doi.org/10.1109/ICUAS51884.2021.9476736.

[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," , 2017.

[10] Hernandez-Leal, P., Kartal, B., and Taylor, M. E., *A survey and critique of multiagent deep reinforcement learning*, Vol. 33, Springer US, 2019. https://doi.org/10.1007/s10458-019-09421-1, URL https://doi.org/10.1007/s10458-019-09421-1.

[11] Feriani, A., and Hossain, E., "Single and Multi-Agent Deep Reinforcement Learning for AI-Enabled Wireless Networks: A Tutorial," *IEEE Communications Surveys and Tutorials*, Vol. 23, No. 2, 2021, pp. 1226–1252. https://doi.org/10.1109/COMST.2021.3063822.

[12] Gupta, J. K., Egorov, M., and Kochenderfer, M., "Cooperative multi-agent control using deep reinforcement learning," *International Conference on Autonomous Agents and Multiagent Systems*, Springer, 2017, pp. 66–83.

[13] Terry, J. K., Grammel, N., Hari, A., Santos, L., and Black, B., "Revisiting Parameter Sharing In Multi-Agent Deep Reinforcement Learning," , 2021.

[14] Yang, Y., and Wang, J., "An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective," 2020, pp. 1–129. URL http://arxiv.org/abs/2011.00583.

[15] Hernandez, D., Denamganaï, K., Gao, Y., York, P., Devlin, S., Samothrakis, S., and Walker, J. A., "A Generalized Framework for Self-Play Training," *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–8. https://doi.org/10.1109/CIG.2019.8848006.

[16] Terry, J. K., Black, B., Jayakumar, M., Hari, A., Santos, L., Dieffendahl, C., Williams, N. L., Lokesh, Y., Sullivan, R., Horsch, C., and Ravi, P., "PettingZoo: Gym for Multi-Agent Reinforcement Learning," *arXiv preprint arXiv:2009.14471*, 2020.

[17] Zheng, L., Yang, J., Cai, H., Zhang, W., Wang, J., and Yu, Y., "MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence," *CoRR*, Vol. abs/1712.00600, 2017. URL http://arxiv.org/abs/1712.00600.

[18] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I., "RLlib: Abstractions for Distributed Reinforcement Learning," *International Conference on Machine Learning (ICML)*, 2018.

Self play with parameter sharing in n-player mixed competitive-cooperative games

Skaltsis, George Marios

2021-12-29

Skaltsis GM, Shin H-S, Tsourdos A. (2021) Self play with parameter sharing in n-player mixed competitive-cooperative games. In: AIAA SciTech 2022 Forum, 3-7 January 2022, San Diego, CA and Virtual Event

https://doi.org/10.2514/6.2022-2498