Parameter Sharing For Heterogeneous Agents in Multi-Agent Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

Parameter sharing, where each agent independently learns a policy with fully shared parameters between all policies, is a popular baseline method for multi-agent deep reinforcement learning. Unfortunately, since all agents share the same policy network, they cannot learn different policies or tasks. This issue has been circumvented experimentally by adding an agent-specific indicator signal to observations, which we term "agent indication." Agent indication is limited, however, in that without modification it does not allow parameter sharing to be applied to environments where the action spaces and/or observation spaces are heterogeneous. This work formalizes the notion of agent indication and proves that it enables convergence to optimal policies for the first time. Next, we formally introduce methods to extend parameter sharing to learning in heterogeneous observation and action spaces, and prove that these methods allow for convergence to optimal policies. Finally, we experimentally confirm that the methods we introduce function empirically, and conduct a wide array of experiments studying the empirical efficacy of many different agent indication schemes for graphical observation spaces.

1 Introduction

Reinforcement Learning (RL) is the intersection of machine learning and optimal control. It allows an agent in an environment to learn a policy which output actions given observations in order to maximize a reward signal defined by the environment. However, many real-life scenarios are more accurately modeled by multi-agent reinforcement learning (MARL), where multiple agents act and learn simultaneously.

MARL methods are further subdivided according to two main properties: whether the agents are cooperative or competitive, and if they are centralized or decentralized. In decentralized systems, each agent takes decisions and learns independently, without access to other agents' observations, actions, or policies. This is similar to real-life scenarios like groups of animals, as opposed to a 'hive-mind' controller acting as a single large RL agent (the centralized case). However, decentralized learning loses theoretical convergence guarantees Tan (1993) due to non-stationarity caused by other agents. Therefore, most modern MARL research follows the paradigm of Centralized Training, Decentralized Execution (CTDE) Lowe et al. (2017), where agents execute their individual policies separately, but have access to other agents' observations during training. Examples of CTDE include MADDPG Lowe et al. (2017), COMA Foerster et al. (2018), and QMIX Rashid et al. (2018).

Full parameter sharing (which we refer to as "parameter sharing") refers to where all policies are represented by a single neural network with the same shared parameters for each. This was introduced by Tan (1993) for classical RL and later concurrently introduced to cooperative multi-agent *deep* reinforcement learning by Gupta et al. (2017); Chu & Ye (2017).

Because parameter sharing only has one policy network or function, it's often assumed it can only handle agents with identical behavior in the environment, which given the apparent usefulness (Zheng et al., 2018; Yu et al., 2022; Chen et al., 2021) of parameter sharing is a profound limitation. This assumption has been relaxed for different types of agents, by adding an indication of the observing agent to the observations, allowing a single policy to serve multiple agents Foerster et al. (2016); Gupta et al. (2017). While this is

known to work empirically, our work's first contribution is formally defining "agent indication" and proving that it allows for convergence to optimal policies (3.2).

An additional limitation of parameter sharing even with agent indication is that agents can't have different action or observation spaces because the shared policy is of fixed size. This work's second contribution is formally introducing two padding based methods that are capable of resolving this for the first time, and similarly proving that these methods allow for convergence to optimal policies (3.2, 3.3). We additionally provide experimental validation that these methods function in 4.

One final practical issue is that there are arbitrarily many ways in which agent indication can be performed on graphical observations, and no literature currently exists on the best approach to this. This work takes an initial step towards this direction by thoroughly evaluating four simple approaches on five different diverse multi-agent environments *with hyperparameter tuning*, finding that two similar ones appear to function consistently upon the environments we ran experiments on.

2 Background and Related Work

2.1 Partially-Observable Stochastic Games

In multi-agent environments, MDPs can be extended to include a set of actions for each agent to create the Multi-agent MDP ("MMDP") model Boutilier (1996). However, this model assumes all agents receive the same rewards. The Stochastic Games model (sometimes called *Markov Games*), introduced by Shapley (1953), extends this by allowing a unique reward function for each agent. The Partially-Observable Stochastic Games ("POSG") model Lowe et al. (2017), defined below, extends the Stochastic Games model to settings where the state is only partially observable (akin to a POMDP). This is a more general model, and what we use in this paper. We define this model for our later use in Definition 1. A good in depth overview of the POSG model and intuition can be found in Terry et al. (2020a).

Definition 1 (Partially-Observable Stochastic Game)**.** A *Partially-Observable Stochastic Game* (POSG) is a tuple $\langle \mathcal{S}, N, \{\mathcal{A}_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\} \rangle$, where:

- $\mathcal{S}$ is the set of possible *states*.
- N is the *number of agents*. The *set of agents* is $[N]$.
- $\mathcal{A}_i$ is the set of possible *actions* for agent i.
- $U = \prod_{i \in [N]} \mathcal{A}_i$ is the set of possible *joint actions* over all agents.
- $P \colon \mathcal{S} \times \prod_{i \in [N]} \mathcal{A}_i \times \mathcal{S} \to [0, 1]$ is the (stochastic) *transition function*.
- $R_i \colon \mathcal{S} \times \prod_{i \in [N]} \mathcal{A}_i \times \mathcal{S} \to \mathbb{R}$ is the *reward function* for agent i.
- Ω_i is the set of possible *observations* for agent i.
- $O_i \colon \mathcal{S} \times \mathcal{A}_i \times \Omega_i \to [0, 1]$ is the *observation function*.

2.2 Parameter Sharing

Full parameter sharing (which we refer to as "parameter sharing"), where all policies are represented by a single neural network with the same shared parameters was first introduced by Tan (1993) for classical RL and later concurrently introduced to cooperative multi-agent *deep* reinforcement learning by Gupta et al. (2017); Chu & Ye (2017). This simple method has since been used to remarkable efficacy in various applications, such as Zheng et al. (2018); Yu et al. (2022); Chen et al. (2021). It's also worth noting that parameter sharing with agent indication is equivalent to naive self-play in competitive environments.

2.3 Coping with Heterogeneity

Agent indication for parameter sharing was first implemented in Foerster et al. (2016), and has been used in numerous derivative works. We are aware of no work studying the best method of this in graphical observations, theoretically studying methods for this, or attempting to cope with heterogeneous action or observation spaces outside of padding the action space of medivac in Samvelyan et al. (2019).

3 Theoretical Results

In this section, we call the agents of a POSG *homogeneous* if they can be reordered into any other order without changing the behavior of the POSG. Formally we chose to define this as meaning all agents have identical action spaces, observation spaces, and reward functions, and that the transition function is symmetric with respect to permutations on the actions input. If a POSG is not homogeneous, we definite it to be heterogeneous.

Parameter sharing has been traditionally seen as a technique that can only be used in games with homogeneous agents because of the fact that a single neural network is learned and shared among all agents. However, in this section we present a novel method of modifying the observation spaces, and prove that it can allow for the use of parameter sharing in the case of heterogeneous agents.

3.1 Disjoint Observation Spaces Allows for Learning an Optimal Policy

When agents are not homogeneous, it is not clear that parameter sharing can be applied. However, we note that a single policy can be used in cases where the observation spaces of the agents are disjoint. We state this claim and prove it, as a preliminary for the full "agent indication" technique described in the following section.

Lemma 1. *If $G = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\}\rangle$ is a POSG such that $\{\Omega_i\}_{i\in[N]}$ is disjoint (i.e., $\Omega_i \neq \Omega_j$ for all $i \neq j$), then any collection of policies $\{\pi_i\}_{i\in[N]}$ can be expressed as a single policy $\pi^{[N]}\colon \left(\bigcup_{i\in[N]} \Omega_i\right) \times \left(\bigcup_{i\in[N]} \mathcal{A}_i\right) \to [0,1]$ which, from the perspective of any single agent i, specifies a policy equivalent to π_i.*[1]

Proof. Let $\Omega = \bigcup_{i\in[N]} \Omega_i$ be the set of all observations across all agents, and similarly define $\mathcal{A} = \bigcup_{i\in[N]} \mathcal{A}_i$ to be the set of all actions available to agents. Note that while Ω is a union of N disjoint sets, it is not necessarily true that $\{\mathcal{A}_i\}_{i\in[N]}$ is disjoint and so $\mathcal{A}$ is not necessarily the union of disjoint sets.

Define $\Omega^{-1}\colon \Omega \to [N]$ as follows: $\Omega^{-1}(\omega)$ is the (unique) agent i for which $\omega \in \Omega_i$. Thus, for all $\omega \in \Omega$, we have that $\omega \in \Omega_{\Omega^{-1}(\omega)}$. Note that Ω^{-1} is well-defined specifically because the observation sets are disjoint, and thus each observation $\omega \in \Omega$ appears in exactly one agent's observation space.

Now, we define our single policy $\pi^{[N]}\colon \Omega \times \mathcal{A} \to [0,1]$. Let

$$\pi^{[N]}(\omega, a) = \begin{cases} \pi_{\Omega^{-1}(\omega)}(\omega, a) & \text{if } a \in \mathcal{A}_{\Omega^{-1}(\omega)} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

One can see from this definition that for any agent $i \in [N]$, for any $\omega \in \Omega_i$, and for any $a \in \mathcal{A}_i$, we have $\pi^{[N]}(\omega, a) = \pi_i(\omega, a)$. Thus, from the view of agent i, $\pi^{[N]}$ defines a policy consistent with its own policy π_i. □

Corollary 1. *For any Partially-Observable Stochastic Game $G = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\}\rangle$ with disjoint observation spaces, there exists a single policy $\pi^*\colon (\bigcup_{i\in[N]} \Omega_i) \times (\bigcup_{i\in[N]} \mathcal{A}_i) \to [0,1]$ which is optimal for all agents; i.e. $\forall i \in [N], \omega \in \Omega_i, a \in \mathcal{A}_i$, we have $\pi^*(\omega, a) = \pi_i^*(\omega, a)$, where π_i^* is an optimal individual policy for agent i.*

To briefly recap the intuition of the method we propose in this section—if by any means, the identity of an agent is indicated in the observation space passed to a full parameter sharing based learning method during

[1] Formally, for any agent $i \in [N]$, observation $\omega \in \Omega_i$, and action $a \in \mathcal{A}_i$, $\pi^{[N]}(\omega, a) = \pi_i(\omega, a)$.

training, then the policy will be able to learn to distinguish each agent and act in a manner adapted to that agent specifically, despite only being a single network.

3.2 Agent Indication Allows for Representing Optimal Policies

When observation spaces are not disjoint—that is, the raw observations from the environment do not indicate the identity of the agent—we can force them to be disjoint by "tagging" the observations with an identifier unique to each agent. This technique is referred to as "agent indication" and is described formally below.

Theorem 1. *For every POSG, there is an equivalent POSG with disjoint observation spaces.*

Proof. Let $G = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\} \rangle$ be a POSG with non-disjoint observation spaces. We define $G' = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, P, \{R_i\}, \{\Omega'_i\}, \{O'_i\} \rangle$, where Ω'_i and O'_i are derived from Ω_i and O_i respectively, as described below.

For each agent i, we define $\Omega'_i = \Omega_i \times \{i\} = \{(\omega, i) \mid \omega \in \Omega_i\}$. Intuitively, we "attach" information about the agent i to the observation. Now, for each agent $i \in [N]$, we define $O'_i \colon \mathcal{A}_i \times \mathcal{S} \times \Omega'_i \to [0, 1]$ as $O'_i(a, s, (\omega, i)) = O_i(a, s, \omega)$. This is equivalent to G in the sense that there is a family of bijections $f_i \colon \Omega_i \to \Omega'_i$ such that $\forall i \in [N], \forall a \in \mathcal{A}_i, \forall s \in \mathcal{S}, \forall \omega \in \Omega_i$, $O_i(a, s, \omega) = O'_i(a, s, f_i(\omega))$ (specifically, $f_i(\omega) = (\omega, i)$). □

Theorem 1 together with Corollary 1 shows that an optimal single policy $\pi^* \colon (\bigsqcup_{i \in [N]} \Omega_i) \times (\bigcup_{i \in [N]} \mathcal{A}_i) \to [0, 1]$[2] is consistent with the optimal policies of each agent: if $\pi^*_i \colon \Omega_i \times \mathcal{A}_i \to [0, 1]$ is an optimal individual policy for agent i, then $\pi^*((\omega, i), a) = \pi^*_i(\omega, a)$ for every action $a \in \mathcal{A}_i$.

In practice, many environments lend themselves to disjoint observation spaces (e.g., games with a third-person point of view), and Lemma 1 shows that one can use a single algorithm (e.g., a single neural network) to learn a single policy that behaves differently for each agent. Additionally, Theorem 1 says that for non-disjoint observation spaces, we can "attach" the identity of each agent to its observations (e.g. by superimposing an identifier that is distinct for each agent onto the observations before they are input to the learning algorithm), forcing the modified observations to be elements of disjoint observation spaces so that Lemma 1 applies. Corollary 1 says that there is no disadvantage to this approach in terms of the optimality of the learned policy. In situations where the representations of the observations of each agent do not have the same size, they can all be "padded" to the size of the largest and learning can proceed as normal. In other words, if you add 0s (or a similar value) to pad observation tensors with a smaller shape than others such that they're the same size, then this can be passed to a policy network and it can learn as normal. This padding allows neural networks or other policies of fixed dimension input to control agents with differing observation spaces.

3.3 Padding Heterogeneous Action Spaces Allows for Representing Optimal Policies

Consider an environment where each agent $i \in [N]$ has a different action space $\mathcal{A}_i$ and, critically, that for some pair of agents $i, i' \in [N]$ we may have $|\mathcal{A}_i| \neq |\mathcal{A}_{i'}|$. In the formal model, this is not a problem, but it does present a minor issue in implementation. Specifically, suppose we have a learning algorithm that has learned a policy $\pi \colon \omega \times \mathcal{A} \to [0, 1]$ for a single agent. In practice, the learning algorithm, given an observation ω, outputs the induced probability distribution $\pi(\omega, \cdot)$ often as a vector $\vec{a} \in [0, 1]^{|\mathcal{A}|}$ with ℓ_1-norm 1.

If each agent's behavior is learned as a separate policy, the learning algorithm for agent i will output a probability vector in $[0, 1]^{|\mathcal{A}_i|}$. However, if there are two agents i, i' with $|\mathcal{A}_i| \neq |\mathcal{A}_{i'}|$, using the same network for both agents i and i' appears to preclude the use of parameter sharing, as the output vectors have different dimensions.

We can address this as follows. Suppose we wish for our algorithm to learn a single policy $\pi^{[N]}$ for all agents, as in the previous section. One option is to have the algorithm output a probability vector in $[0, 1]^{|\mathcal{A}|}$, where $\mathcal{A} = \bigcup_{i \in [N]} \mathcal{A}_i$ as before. Now, when the algorithm outputs a vector $\vec{a} \in [0, 1]^{\mathcal{A}}$ for agent i, we can simply "clip" the vector and consider only the subvector corresponding to actions in $\mathcal{A}_i$. We can write this formally as $\vec{a}_{\mathcal{A}_i} \in [0, 1]^{\mathcal{A}_i}$.

[2] $\bigsqcup_i \Omega_i$ is the *disjoint union*: $\bigsqcup_{i \in [N]} \Omega_i := \bigcup_{i \in [N]} (\Omega_i \times \{i\}) = \bigcup_{i \in [N]} \{(\omega, i) \mid \omega \in \Omega_i\}$.

This can be quite space-inefficient, though, as in the worst case (when all agents have disjoint action spaces) $|\mathcal{A}| = \sum_{i \in [N]} \mathcal{A}_i$. Further, since an agent i will never perform an action $a \notin \mathcal{A}_i$, much space is wasted representing the output as a sparse vector (i.e. one with many zeros). A more practical alternative is to instead output a vector whose dimension is only as large as the largest action space, padding the vector with zeros for agents with smaller action spaces. Formally, the learning algorithm can simply output a vector $\vec{a} \in [0, 1]^{\alpha}$ for all agents, padding zeros at the end where necessary. An agent i with $|\mathcal{A}_i| < \alpha$ receiving a vector $\vec{a} \in [0, 1]^{\alpha}$ can simply consider the subvector $\langle \vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{|\mathcal{A}_i|} \rangle$. Essentially, the learning algorithm pads the action vector to length α, and each agent clips the vector they receive to length $|\mathcal{A}_i|$.

To briefly recap the intuition of the method we propose in this section—if you can "pad" action spaces so that a neural network is outputting actions of a fixed dimensionality and range, agents that can only accept less than this can either clip or throw out unneeded actions and the neural network can still learn fine. This padding allows neural networks or other policies of fixed dimension output to control agents with differing action spaces.

4 Experimental Results

One open question about agent indication is what method of indicating agent in graphical observation space performs the best empirically, or if they even significantly matter. While this will always be environment dependent and require tuning for optimal performance, we hope to offer a baseline for future implementation to be based upon,

1. *Identity*– Do nothing to the observation, included for control.
2. *Geometric*– Add an additional channel with alternating geometric checkered pattern for different types of agents.
3. *Binary*– Add additional channels, each of which is entirely black or white based on the type of an agent.
4. *Inversion*– Invert the color of the observation of certain types of agents and add it as a channel to the original observation. For agents that do not need inversion, duplicate original observation.
5. *Inversion with Replacement*– The same as *Inversion*, but the inverted observation (or the same duplicate observation for agents that do not need it) is used in place of the original observation.

Additionally, we wish to experimentally validate that the three of the methods we discussed in the prior section (agent indication and both forms of padding) empirically allow for learning.

4.1 Experimental Methodology

Fairly evaluating the performance of each of the methods of graphical agent indication that we posed in the prior section is challenging. What we chose to do is to conduct large hyperparameter tuning sweeps of all hyperparameters, with the agent indication method posed as a hyperparameter, for two the most popular parameter sharing methods (parameter shared PPO) across a diverse set of environments. Given this, we then trained the best hyperparameters for each environment repeatedly, allowing us to make claims about which agent indication method is associated with the best performing hyperparameters after automated tuning.

This experiment design accomplished several desirable things. By choosing different parameter sharing methods and environments, we're able to see if performance is consistently better with one method in a diverse set of scenarios. Additionally, by including the agent indication method as a hyperparameter in automated hyperparameter tuning, this allows for picking agent indication methods that perform the best overall, and not just with specific (likely non-optimal) hyperparameters. By repeatedly training hyperparameters found during automated tuning, we're also able to far more accurately characterize the performance of each set of hyperparameters.

4.2 Environment Selection

We selected five environments from this work, all from PettingZoo (Terry et al., 2020a), a library with over 50 diverse multi-agent reinforcement learning environments with a standardized API. PettingZoo has two "classes" of environments with graphical observations- the "Butterfly" class with custom made games with pygame rendering and pymunk physics with a diverse set of rules, as well as the multi-player Atari games introduced in Terry et al. (2020c). To maximize environment diversity, we chose every environment in the Butterfly class with heterogeneous agents (three), and two Atari games (the choice of which we will explain below).

The environments we chose were:

- *Cooperative Pong*- A Butterfly environment inspired by the Atari pong environment, where two differently shaped pistons work together to keep the ball in the air for as long as possible.
- *Knights Archers Zombies*- A Butterfly environment where knight and archer agents collectively work together to prevent zombies from crossing the bottom of the screen.
- *Prospector*- A Butterfly environment where banker and prospector agents work together to collect gold, give to one another and deposit it in banks
- *Pong*- classic two player Atari pong
- *Entombed Cooperative*- an Atari game where two identical agents work together to progress to the bottom of the screen

More comprehensive documentation of all these environments is available at `pettingzoo.ml`.

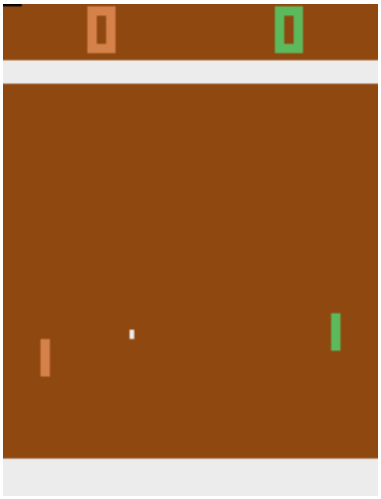
All of the butterfly class environments are cooperative environments (which is when parameter sharing is conventionally described as being used), with heterogeneous types of agents. Entombed cooperative was chosen because it's the only cooperative multiplayer game supported in PettingZoo (few were ever made), and because we wanted to see if adding agent indication so that the agents could be differentiated in an identical observation would impact learning performance. Pong was added as a simple test of the impact of agent indication in a simple and well known competitive environment (full parameter sharing is the same method as naive self play when applied to fully competitive games, though it's not often described as such). These environments are all pictured in Figure 1.

4.3 Additional Implementation Details

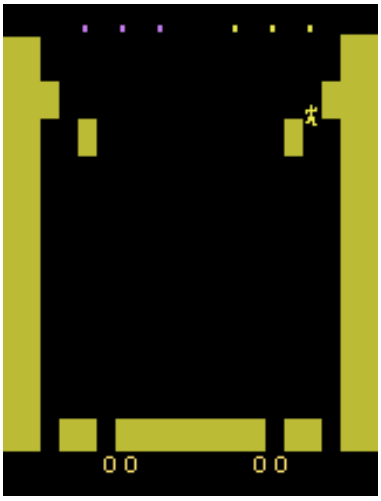
Each environment had standard preprocessing applied to the observations via SuperSuit Terry et al. (2020b), and the following number of training timesteps for each environment were used:

- *Cooperative Pong*- 4 million timesteps.
- *Knights Archers Zombies*- 10 million timesteps.
- *Prospector*- 100 million timesteps.
- *Pong*- 10 million timesteps.
- *Entombed Cooperative*- 10 million timesteps.

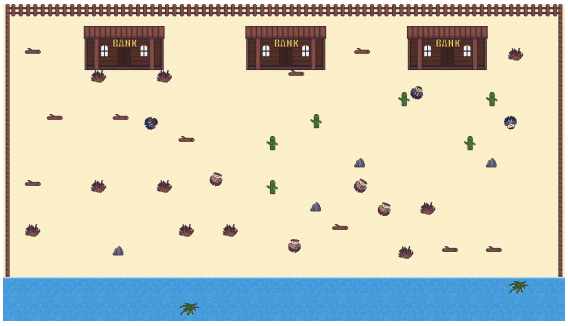
Additionally, the hyperparameter ranges we searched are described in Appendix A, and the evaluation of agents in Atari games was done by playing the games with built-in bots. For instance, in Pong environment, we evaluated our agent's performance by letting it play against built-in bot implemented in the game's single player mode. All of our learning code was implemented by Stable Baselines 3 (Raffin et al., 2021), and our hyperparameter search code was based upon RL Baselines3 Zoo (Raffin, 2020). All code used in our experiments is available at `https://anonymous.4open.science/r/parameter-sharing-paper-7C02/README.md`.



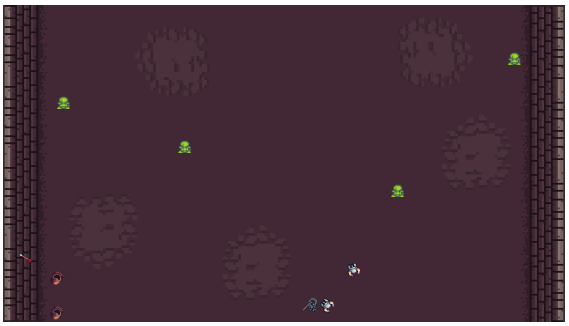
(a) Pong



(b) Entombed Cooperative



(c) Prospector



(d) Knights Archers Zombies

(e) Cooperative Pong

Figure 1: Images of the benchmark environments from Terry et al. (2020a).

Table 1: 10 best agent indication methods for Cooperative Pong. 10 additional evaluations were done for each of the hyperparameters.

Agent indication method	Avg. Reward
Inversion	-33.13
Inversion	-37.20
Inversion	-40.41
Inversion	-40.98
Inversion	-41.18
Inversion	-41.91
Inversion	-42.31
Inversion	-42.66
Inversion with Replacement	-42.85
Inversion	-43.22

4.4 Results

For each environment, the 10 best hyperparameters found during automated hyperparameter are shown. The reward shown in the table is the by the average best per agent reward over 10 additional evaluation runs (5 for prospector due to the run time), to give more accurate rankings. Tables 1-5 show the reward of these parameters ranked, along with the agent indication method from 1 included in the hyperparameter set.

Table 2: 10 best agent indication methods for KAZ. 10 additional evaluations were done for each of the hyperparameters.

Agent indication method	Avg. Reward
Inversion	0.87
Geometric	0.82
Inversion	0.67
Inversion with Replacement	0.66
Inversion	0.65
Inversion	0.61
Inversion with Replacement	0.58
Inversion with Replacement	0.53
Geometric	0.53
Geometric	0.51

Table 3: 10 best agent indication methods for Prospector. 5 additional evaluations were done for each of the hyperparameters.

Agent indication method	Avg. Reward
Binary	40.13
Binary	29.33
Binary	29.09
Inversion	12.23
Inversion	8.46
Inversion with Replacement	5.75
Inversion	4.21
Inversion with Replacement	3.95
Inversion with Replacement	3.14
Inversion with Replacement	2.40

Table 4: 10 best agent indication methods for Pong. 10 additional evaluations were done for each of the hyperparameters.

Agent indication method	Avg. Reward
Inversion	-18.5
Inversion	-19.1
Inversion	-19.6
Inversion	-20.3
Geometric	-20.9
Inversion with Replacement	-20.9
Inversion with Replacement	-21.0
Inversion with Replacement	-21.0
Inversion with Replacement	-21.0
Inversion with Replacement	-21.0

Table 5: 10 best agent indication methods for Entombed Cooperative. 10 additional evaluations were done for each of the hyperparameters.

Agent indication method	Avg. Reward
Geometric	6.3
Identity	5.8
Identity	5.7
Identity	5.5
Identity	5.4
Identity	5.3
Inversion with Replacement	4.9
Identity	4.6
Inversion	4.1
Inversion with Replacement	3.6

These table results show an arguably surprising result that environments appear to be highly sensitive to agent indication method. However, the inversion methods (with and without replacement) appeared to generally perform adequately across all environments, indicating that they are likely a good baseline and starting point when approaching new environments. Additionally, as prospector, an environment with heterogeneous action and observation spaces and differing agents, was able to be learned with parameter sharing using agent indication and the two padding methods this work introduces, this shows that these methods we introduce can empirically allow for learning. These results to not contradict our previous proofs, they just show that the specific method of agent indication can make converging to optimal policies more difficult, like any hyperparameter in machine learning can do.

5 Conclusion, Limitations and Future Work

This paper introduces novel methods for coping with heterogeneous action and observation spaces in multi-agent environments learned via full parameter sharing, and shows that these and the previously known method of agent indication (for coping with heterogeneous agent behaviors) are able to learn optimal policies. We show that these methods are also capable of working together to empirically allow for learning.

We further offer experiments on the efficacy of different methods of agent indication in graphical observation spaces, a previously unstudied problem, showing that it appears to be highly environment depending (which is arguably surprising), but that inversion based methods appear to serve as a good baseline and starting point when approaching new environments.

We also note that these experiments used the pettingzoo_env_to_vec_env_v0 and concat_vec_envs_v0 wrappers from SuperSuit (Terry et al., 2020b). These had bugs that would cause observations to be delayed by one frame under specific circumstances that were fixed in a recent SuperSuit version (3.3.2). We have no reason to believe that this bug would significantly affect our results, but we cannot state this with complete certainty. The experiments were not rerun given the very large computational costs and view of the low probability of errors.

We hope these results motivate future experimental work that treats full parameter sharing with much greater importance in multi-agent deep reinforcement learning.

References

Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pp. 195–210. Morgan Kaufmann Publishers Inc., 1996.

Dong Chen, Zhaojian Li, Yongqiang Wang, Longsheng Jiang, and Yue Wang. Deep multi-agent reinforcement learning for highway on-ramp merging in mixed traffic. *arXiv preprint arXiv:2105.05701*, 2021.

Xiangxiang Chu and Hangjun Ye. Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1710.00336*, 2017.

Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29: 2137–2145, 2016.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI*, 2018.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pp. 66–83. Springer, 2017.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pp. 6379–6390, 2017.

Antonin Raffin. Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304. PMLR, 2018.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. ISSN 0027-8424. doi: 10.1073/pnas.39.10.1095.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

J. K Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sulivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020a.

Justin K Terry, Benjamin Black, and Ananth Hari. Supersuit: Simple microwrappers for reinforcement learning environments. *arXiv preprint arXiv:2008.08932*, 2020b.

Justin K Terry, Benjamin Black, and Luis Santos. Multiplayer support for the arcade learning environment. *arXiv preprint arXiv:2009.09341*, 2020c.

Javier Yu, Joseph Vincent, and Mac Schwager. Dinno: Distributed neural network optimization for multi-robot collaborative learning. *IEEE Robotics and Automation Letters*, 2022.

Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

A Hyperparameter Search Range

A.1 PPO

Table 6: Hyperparameter search range for PPO algorithm. Continuous range is noted in square brackets.

Hyperparameter	Range
Batch size	8, 16, 32, 64, 128, 256, 512
Number of timesteps per update	8, 16, 32, 64, 128, 256, 512, 1024, 2048
Discount factor (Gamma)	0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999
Entropy coefficient	[0.00000001, 0.1]
Clip range	0.1, 0.2, 0.3, 0.4
Number of epochs	1, 5, 10, 20
GAE coefficient (Lambda)	0.8, 0.9, 0.92, 0.95, 0.98, 0.99, 1.0
Maximum gradient norm	0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 5
Value function coefficient	[0, 1]
Network architecture	(64, 64), (256, 256)
Activation function	tanh, relu