REVISITING PARAMETER SHARING IN MULTI-AGENT DEEP REINFORCEMENT LEARNING

Justin K. Terry
Swarm Labs
University of Maryland, College Park
justin.terry@swarmlabs.com

Nathaniel Grammel
Department of Computer Science
University of Maryland, College Park
ngrammel@umd.edu

Ananth Hari
Swarm Labs
Department of Electrical and Computer Engineering
University of Maryland, College Park
ananth.hari@swarmlabs.com

Luis Santos
Swarm Labs
Department of Mechanical Engineering
University of Maryland, College Park
luis.santos@swarmlabs.com

Benjamin Black
Swarm Labs
Department of Computer Science
University of Maryland, College Park
benjamin.black@swarmlabs.com

arXiv:2005.13625v5 [cs.LG] 11 Nov 2020

ABSTRACT

"Nonstationarity" is a fundamental problem in cooperative multi-agent reinforcement learning (MARL). It results from every agent's policy changing during learning, while being part of the environment from the perspective of other agents. This causes information to inherently oscillate between agents during learning, greatly slowing convergence. We use the MAILP model of information transfer during multi-agent learning to show that increasing centralization during learning arbitrarily mitigates the slowing of convergence due to nonstationarity. The most centralized case of learning is parameter sharing, an uncommonly used MARL method, specific to environments with homogeneous agents. It bootstraps single-agent reinforcement learning (RL) methods and learns an identical policy for each agent. We experimentally replicate our theoretical result of increased learning centralization leading to better performance. We further apply parameter sharing to 8 more modern single-agent deep RL methods for the first time, achieving up to 44 times more average reward in 16% as many episodes compared to previous parameter sharing experiments. We finally give a formal proof of a set of methods that allow parameter sharing to serve in environments with heterogeneous agents.

1 INTRODUCTION

Multi-agent reinforcement learning methods in cooperative environments seek to learn a policy (a function that takes observations and returns actions) that achieves the maximum expected total reward for all agents. As is common in MARL, the present work focuses on "decentalized" policies, where each agent has their own policy and can act without a central controller (Bu et al., 2008).

A nonstationary environment is one that changes during learning (Choi et al., 2000; Chades et al., 2012). While this can be beneficial when done intentionally (e.g., curriculum learning), it typically makes learning far more difficult as the policy's knowledge of the environment will become wrong over time (Bengio et al., 2009; Choi et al., 2000; Chades et al., 2012). Virtually all cooperative multi-agent reinforcement learning suffers from very slow convergence due to nonstationarity (Matignon et al., 2012; Hernandez-Leal et al., 2017; Papoudakis et al., 2019). Consider an environment with two agents, "Alice" and "Bob", who must learn to work together. Alice's policy must have knowledge of Bob's policy, which from her perspective is a part of the environment (and vice versa for Bob's

policy). At each step of learning, Alice learns about Bob's policy and the rest of the environment. Bob also then learns about the environment and Alice's policy, updating his policy. Alice's knowledge of Bob's policy is now slightly wrong, so Alice must now learn Bob's new policy and update her own, making Bob's knowledge of hers slightly wrong. This "ringing" of information can greatly slow convergence during learning, especially for highly coordinated tasks with many agents, and this specific form of nonstationarity is believed to be a fundamental reason why converging to good policies in multi-agent learning is so difficult Papoudakis et al. (2019).

The naive way for Alice and Bob to learn would be to concurrently learn separate policies for each with a single-agent RL method. As the above example illustrates, this is ineffective and experimentally is usually only able to learn toy environments (Matignon et al., 2012). This has motivated work on specialized multi-agent deep reinforcement learning (MADRL) methods, notably QMIX (Rashid et al., 2018), COMA (Foerster et al., 2018) and MADDPG (Lowe et al., 2017). These employ "centralization" techniques that allow Alice to learn about Bob's policy faster than just watching him in the environment (Papoudakis et al., 2019). Intuitively, this reduces the "ringing" delay, thus mitigating the nonstationarity.

MADDPG has a single shared critic network during learning, which acts on separate actor networks for each agent. Parameter sharing in MARL takes this to the extreme: learning a single shared policy simultaneously for multiple agents. This is clearly the most centralized case of learning, as there is only a single policy and no communication between policies. Note that centralization is used to refer to a handful of similar-but-distinct concepts in the MARL literature; here it refers to information transfer rates between policies during learning (formalized in subsection 2.3).

Parameter sharing, on the other hand, "bootstraps" single-agent RL methods to learn policies in cooperative environments, and was concurrently introduced by Gupta et al. (2017) for DDPG, DQN and TRPO and by Chu and Ye (2017) for a special case of DDPG. Their experimental results were not as remarkable as methods like MADDPG, and other similar methods have seen far more widespread adoption (Baker et al., 2019). However, DDPG, DQN, and TRPO were some of the first DRL methods; many newer ones exist which often have dramatically better performance and prior to our work have not been tested with parameter sharing.

On the surface, it seems like having one policy for every agent means that all agents must be identical (or "homogeneous"), and this was initially assumed to be the case (Gupta et al., 2017; Chu and Ye, 2017). However, by including the agent (or agent type) as part of the observation, a single policy can respond differently for each agent. This "agent indication" technique was first used by Gupta et al. (2017). Even with this, observation spaces of all agents must be the same size since there is a single neural network; however this can be resolved by "padding" the observations of agents to a uniform size. We can similarly "pad" action spaces to a uniform size, and agents can ignore the actions outside their "true" action space.

1.1 Main Contributions

In section 3, we use the MAILP model to mathematically formalize the above intuition that more centralization during learning mitigates nonstationarity in MARL and allows for faster convergence. Parameter sharing is the most centralized MARL method possible, so this theory also provides an explanation for experimental performance variations in MADRL methods: that "full" parameter sharing does better than fully independent single-agent learning (the most decentralized case). Furthermore, we prove a lower bound showing that cases of highly decentralized learning methods take a fantastically long time to converge due to nonstationarity.

In section 4, we empirically replicate the predictions of our theorems by applying both parameter sharing and fully independent learning applied to 11 single-agent DRL methods on the Gupta et al. (2017) benchmark environments, finding that parameter sharing consistently performs the best. Eight of the single-agent RL methods we apply parameter sharing to are more modern than those previously tested with parameter sharing. With these, we achieved the best documented performance on every environment from Gupta et al. (2017), achieving up to 44x more average reward in as little as 16% as many episodes compared to previously documented parameter sharing arrangements; we also outperform MADDPG and QMIX on all three environments. The policies learned by parameter sharing also orchestrate basic emergent behavior.

In section 5, we introduce the aforementioned "observation padding" and "action space padding" methods to allow parameter sharing to learn in environments with heterogeneous agents. We also offer proofs that these and agent indication allow parameter sharing to learn optimal policies. Based on a preprint version of this work, these methods have been successfully experimentally used in Terry et al. (2020b) and built into the SuperSuit set of multi-agent wrappers so that anyone can easily use them (Terry et al., 2020a).

2 BACKGROUND

2.1 REINFORCEMENT LEARNING

Reinforcement learning (RL) methods seek to learn a policy (a function which takes the observation and returns an action) that achieves the maximum expected total reward for an environment. Single-agent environments are traditionally modeled as a *Markov Decision Process* ("MDP") or a *partially-observable MDP* ("POMDP") (Boutilier, 1996). An MDP models decision making as a process where an agent repeatedly takes a single action, receives a reward, and transitions to a new state (receiving complete knowledge of the state). A POMDP extends this to include environments where the agent may not be able to observe the entire state.

In Deep Reinforcement Learning (DRL), a neural network is used to represent the policy. These methods generally fall into two categories: Q learning methods and policy gradient ("PG") methods. The first deep Q learning method was the Deep Q Network ("DQN") (Mnih et al., 2013), and the first widely used PG method was Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015). These methods have since been iterated on, arriving at the newer and more powerful methods that we use in this paper: "SAC" (Haarnoja et al., 2018), "IMPALA" (Espeholt et al., 2018), "TD3" (Fujimoto et al., 2018), "PPO" (Schulman et al., 2017), "TRPO" (Schulman et al., 2015), "A2C" from (Dhariwal et al., 2017) (a synchronous version of "A3C" from (Mnih et al., 2016)), "Rainbow DQN" (Hessel et al., 2018), "Ape-X DQN" and "ApeX DDPQ" (Horgan et al., 2018).

2.2 MULTI-AGENT REINFORCEMENT LEARNING

In multi-agent environments, Multi-agent MDPs ("MMDPs") (Boutilier, 1996) extend MDPs by allowing for a set of actions to accommodate multiple agents. However, MMDPs assume all agents receive the same reward. Stochastic Games (sometimes called *Markov Games*), introduced by Shapley (1953), extends this by allowing a unique reward function for each agent.

Partially-Observable Stochastic Games ("POSG") (Lowe et al., 2017), defined below, extend Stochastic Games to settings where the state is only partially observable (akin to a POMDP), and is the model we use throughout this paper.

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.
- $P \colon \mathcal{S} \times \prod_{i \in [N]} \mathcal{A}_i \times \mathcal{S} \to [0, 1]$ is the (stochastic) *transition function*.
- $R_i \colon \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N \times \mathcal{S} \to \mathbb{R}$ is the *reward function* for agent i.
- Ω_i is the set of possible *observations* for agent i.
- $O_i \colon \mathcal{A}_i \times \mathcal{S} \times \Omega_i \to [0, 1]$ is the *observation function*.

2.2.1 PARAMETER SHARING

The notion of parameter sharing exists is common throughout deep learning. In multi-agent reinforcement learning, it refers to a learning algorithm acting on behalf of every agent, while using and making updates to a collective shared policy (Gupta et al., 2017).

2.3 MULTI-AGENT INFORMATIONAL LEARNING PROCESSES

Introduced by Terry and Grammel (2020), the *Multi-Agent Informational Learning Process (MAILP)* model very generally describes the propagation of information through multi-agent systems during learning. In the model, each agent $i \in [N]$ has information $\mathcal{I}_i \in [0, 1]$. With an optimal policy, a certain fraction of its information must depend on the environment, while the rest of the information may depend on different agents or groups of agents, with the total amount of information summing to 1. The fraction of i's information that relates to a group of agents $\mathcal{G} \subseteq [N] \setminus \{i\}$ is referred to as the *coordination coefficient*, and is denoted by $\mathcal{C}_{i,\mathcal{G}}$; $\mathcal{C}_{i,env}$ denotes the fraction of i's information that relates to the environment. After each step of learning, the agent's information increases by:

$$\Delta^{\uparrow}\mathcal{I}_{i,\chi}(t) = \mathcal{K}_{i,\chi}\Lambda(\mathcal{C}_{i,\chi} - \mathcal{I}_{i,\chi}(t-1)), \tag{1}$$

where $\chi \in 2^{[N]\setminus\{i\}} \cup \{env\}$ represents either the environment (env) or a group of agents $\mathcal{G} \subseteq [N] \setminus \{i\}$; Λ is the *learning function* with the property that $\Lambda(x) \leq x$ for all $x \in [0, 1]$; and $\mathcal{K}_{i,\chi}$ is the *centralization coefficient*, which simply scales how fast information can transfer between entities during learning. Given arbitrary $\Lambda, \mathcal{C}, \mathcal{K}$, this can capture any learning process as long as the rate at which agents gain information stays the same or decreases as the agents gains more information.

The coordination and centralization coefficients between each combination of agents are collected into respective "tensor sets", which are properties of the environment, indexed as above. After each step, agents also lose some information about the other agents, which depends on how much they need to know about the other agent (the coordination coefficient) and how much the other agent learned, described in (1). This is nonstationarity: as the other agent learns new things, a portion of knowledge an agent has about them will inherently be wrong.

$$\Delta^{\downarrow}\mathcal{I}_{i,\mathcal{G}}(t) = \frac{\Delta^{\uparrow}\mathcal{I}_{\mathcal{G}}(t)\mathcal{I}_{i,\mathcal{G}}(t-1)}{\mathcal{I}_{\mathcal{G}}(t-1) + \Delta^{\uparrow}\mathcal{I}_{\mathcal{G}}(t)} \tag{2}$$

Summing (1) and (2) thus gives the total change in information for each agent after every step:

$$\mathcal{I}_i(t) = \mathcal{I}_i(t-1) + \Delta\mathcal{I}_i(t) = \mathcal{I}_i(t-1) + \Delta^{\uparrow}\mathcal{I}_{i,env}(t) + \sum_{\mathcal{G}} \left(\Delta^{\uparrow}\mathcal{I}_{i,\mathcal{G}}(t) - \Delta^{\downarrow}\mathcal{I}_{i,\mathcal{G}}(t)\right)$$

Learning is said to be complete when $\mathcal{I}_i(t) \geq 1 - \epsilon$ for all agents $i \in [N]$, where $\epsilon > 0$ is some small constant. $\mathcal{I}^0 := \mathcal{I}(0)$ is also used to denote the initial information that the agent has. We will often denote by $\mathcal{I}^{-1}(x)$ the earliest time step t in which $\mathcal{I}(t) \geq x$, so that we may write $\mathcal{I}^{-1}(1-\epsilon)$ to indicate the number of time steps needed to finish learning.

3 LEARNING CENTRALIZATION AND NONSTATIONARITY

Based on the intuition about nonstationarity outlined in section 1, we use the MAILP model to prove MARL converges slowly under normal circumstances due to nonstationarity, and that centralization during learning arbitrarily improves this (with parameter sharing having the most centralization).

As a baseline, we first analyze the convergence of a single agent environment. Here $\mathcal{C}_{env} = 1$, and as there are no other agents we drop the subscripts for $\mathcal{I}$. Our key result for the single-agent setting is Theorem 1, the proof of which is in Appendix A.

Theorem 1. *For single-agent learning with learning rate $\mathcal{K}_{env}$ and initial information $\mathcal{I}^0$, we have*

$$\mathcal{I}^{-1}(1-\epsilon) \geq \frac{\log(\epsilon) - \log(1-\mathcal{I}^0)}{\log(1-\mathcal{K}_{env})}$$

In the multi-agent setting, we derive a bound for an environment where each of the N agents' knowledge depends on the behaviors of each of the other $N-1$ agents in equal proportions. All agents pairs have the same coordination and centralization coefficients. This is typical of many real world environment requiring a high degree of coordination.

We use $\mathcal{C}_\star$ to denote the coordination value for each pair of agents ($\mathcal{C}_{i,j}$, where $i, j \in [N]$). Since $\mathcal{C}_{i,env} + \sum_{j\neq i}\mathcal{C}_{i,j} = 1$, we have that $\mathcal{C}_{i,env} = 1 - (N-1)\mathcal{C}_\star$. Further, as $\mathcal{C}_{i,env}$ is the same for all agents $i \in [N]$, we will simply denote this as $\mathcal{C}_{env}$. $\mathcal{C}_{env}$ is very small, so behavior predominantly

depends on the actions of policies of other agents. $\mathcal{K}_\star$ denotes the centralization coefficient between every pair of agents. Using these, the information update function for each agent looks like:

$$\mathcal{I}_i(t) = \mathcal{I}_{i,env}(t) + \sum_{j \in [N]_{-i}} \mathcal{I}_{i,j}(t) \tag{3}$$

To focus on nonstationarity and simplify the proof, we start with each agent having full knowledge of the environment ($\mathcal{I}_{i,env}(0) = (1-\epsilon)\mathcal{C}_{env}$). Because $\mathcal{C}_{env}$ is near 0, this is a reasonable simplification (and can only decrease the bound). Since all agents are identical, $\mathcal{I}_{i,j}(t)$ is equal for all $i, j \in [N]$. Thus, we denote the information an agent has about another agent at time t as $\mathcal{I}_\star, \star(t)$. Finally, we let $\mathcal{I}^0_{\star,\star} := \mathcal{I}_{\star,\star}(0)$. Given all this, we state Theorem 2, proof in Appendix A.

Theorem 2. *$\mathcal{I}_i^{-1}(1-\epsilon) \geq t^*$, where*

$$t^* = \frac{\log(\mathcal{C}_\star \epsilon / (\mathcal{C}_\star - \mathcal{I}^0_{\star,\star}))}{\log\left(1 - \mathcal{K}_\star + \frac{\mathcal{K}_\star \mathcal{I}^0_{\star,\star}}{\mathcal{C}_{env}/(n-1) + \mathcal{C}_\star(\mathcal{K}_\star + 1)}\right)}$$

This is a bound on the convergence rate of multi-agent learning. We illustrate this bound in Figure 1, which shows how low centralization rates increase convergence time. Compared to the single agent case (Theorem 1), we can see that nonstationarity can make multi-agent learning intractable. We can also see that for highly centralized learning, this problem becomes vanishingly small.

Since in parameter sharing, all agents share the same neural network, the agents always have the most up-to-date information about the behavior of all other agents. Thus, it achieves the theoretically best value $\mathcal{K}_\star = 1$. Theorem 3 (proof in Appendix A) describes the precise convergence rate when $\mathcal{K}_\star = 1$, which is not meaningfully worse than the single agent limit.

Theorem 3. *As $\mathcal{K}_\star \to 1$, the minimum convergence time t^* as given in Theorem 2 approaches*

$$\frac{\log(\mathcal{C}_\star \epsilon / (\mathcal{C}_\star - \mathcal{I}^0_{\star,\star}))}{\log\left(\frac{\mathcal{I}^0_{\star,\star}(n-1)}{1+(n-1)\mathcal{C}_\star}\right)}.$$

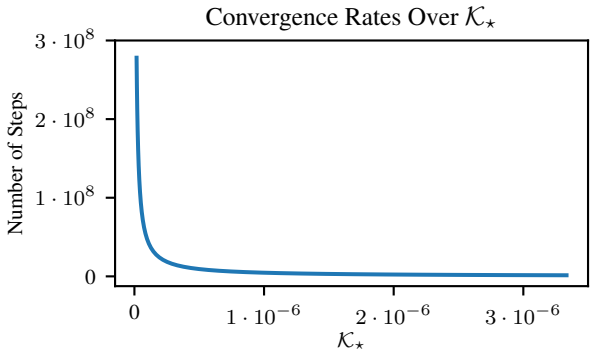
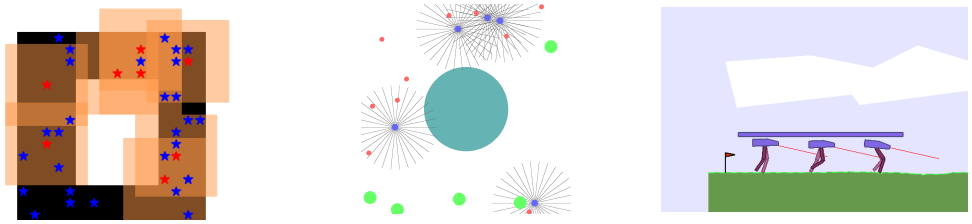


Figure 1: The number of steps needed for convergence for different values of $\mathcal{K}_\star$, as given by Theorem 2. We use $\mathcal{C}_{env} = 0.1$, $\epsilon = .001$, $\mathcal{I}^0 = 0.01$, $N = 3$. This illustrates how larger values of $\mathcal{K}$ (which correspond to higher degrees of centralization) can dramatically improve convergence times, and how smaller values of $\mathcal{K}$ can dramatically lengthen them.

The lower bound on convergence for multiple agents is larger than in the single-agent case. We note also that it is a *looser* bound than the single agent case. One can see from the proof of Theorem 1 that if $\Lambda(x) = x$, then the bound is *tight*. This is *not* the case for Theorem 2, which is not tight even when $\Lambda(x) = x$. Because of this, we have a strict gap on the convergence rate for multiple agents compared to a single agent; that is, we have an upper bound on convergence for a single agent, and a lower bound for the convergence of multiple agents.



(a) Pursuit (b) Waterworld (c) Multiwalker

Figure 2: Images of our benchmark environments, from Gupta et al. (2017).

4 Experimental Results

Parameter sharing has previously only been used experimentally with relatively simple DRL methods (plain DQN, plain DDPG, and TRPO) (Gupta et al., 2017; Chu and Ye, 2017). This raises the question of how parameter sharing performs when applied to more recent and advanced DRL methods. Accordingly, we tested parameter sharing with 11 DRL methods (8 for the first time in the literature) on the MARL benchmark environments from Gupta et al. (2017).

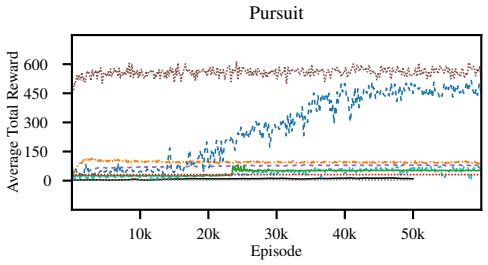
We use three MARL benchmark environments from Gupta et al. (2017), shown in Figure 2. In *pursuit*, 8 red pursuer agents must work together to surround and capture 30 randomly moving blue evader agents. The action space of each agent is discrete (four cardinal directions and "do nothing"). The observation space is a 7×7 box centered around an agent, depicted by the orange box. In *waterworld*, there are 5 purple agents, 5 green food targets and 10 red poison targets, and agents must learn to eat food and avoid colliding with poison or other agents. The action space of each agent is a 2-element vector indicating thrust in each direction, and the observation space is a 122-element vector of "antennae" outputs indicating the position of objects in the environment. In *multiwalker*, there is a package placed on top of 3 pairs of robot legs that must work together to move the package as far as possible to the right without dropping it. The action space of each agent is a 4-element vector of the torque to be applied to each leg joint, and the observation space of each agent is a 31-element vector of positions and velocities of elements of the walker with noise applied. More information about the environments is available in Appendix B.

Figure 3 shows the massive performance impact of newer DRL methods for parameter sharing both in terms of convergence time and average reward, achieving up to a 44x performance improvement on pursuit. These policies achieved the highest documented average total reward for these benchmark environments, and orchestrate basic emergent behavior. This unrealized potential in parameter sharing may account for why it was not been popularly used before. We reproduced the results from Gupta et al. (2017) to account for any discrepancies caused by our tooling rather than the algorithms, shown in Figure 3, finding no major differences.

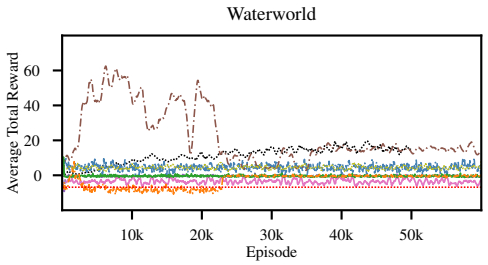
We then benchmarked the performance of all these single agent methods learning fully independently, the results of which are included in Figure 4. All methods performed worse than parameter sharing methods, reinforcing our theoretical results in section 3 on the importance of centralization in MARL. All learning was done with RLlib (Liang et al., 2017). All code, training log files, and the best trained policies for each method game combination are available at `github.com/parametersharingmadrl/parametersharingmadrl`. All hyperparameters are also included in Appendix D. We used the same MLP as Gupta et al. (2017), with a 400-followed by a 300-node hidden layer. Note that we computed rewards as the sum of all agent's reward after every step. Additionally, in Appendix C we compare the results of our best performing policies to MADDPG and QMIX, achieving superior performance for all three environments.

5 Parameter Sharing for Heterogeneous Agents

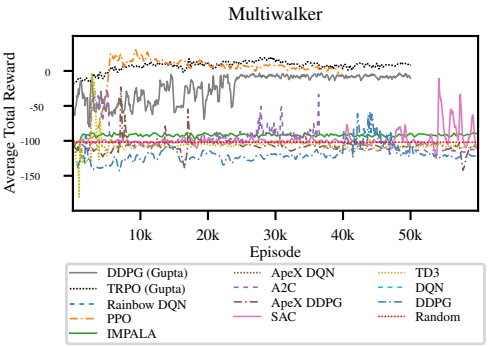
Given the apparent performance benefits of parameter sharing, it is desirable to be able to apply it to as many types of environments as possible. The two main limitations of parameter sharing are that it can only apply to cooperative environments (or cooperative sets of agents in an environment), and that it can only work for "homogenous" sets of agents (Gupta et al., 2017).



(a) Maximum average total reward was 621 after 24k steps of ApeX DQN. Maximum average total reward with a previously documented method was 14 after 40k steps of TRPO. Average reward for a random policy was 31.04.

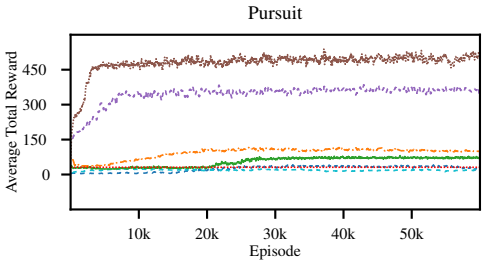


(b) Maximum average total reward was 74 after 6.3k steps of ApeX-DDPG. Maximum average total reward with a previously documented method was 19.5 after 39k steps of TRPO. Average reward for a random policy over 1000 plays was -6.82.

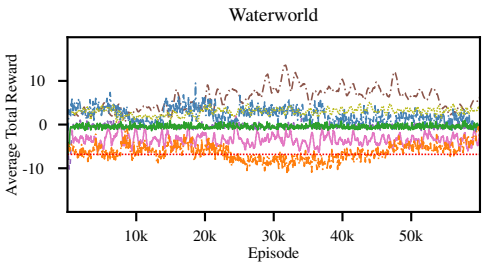


(c) Maximum average total reward was 41 after 9.2k steps of PPO. Maximum average reward with a previously documented method was 20 after 29k steps of TRPO. Average reward for a random policy was -102.05.

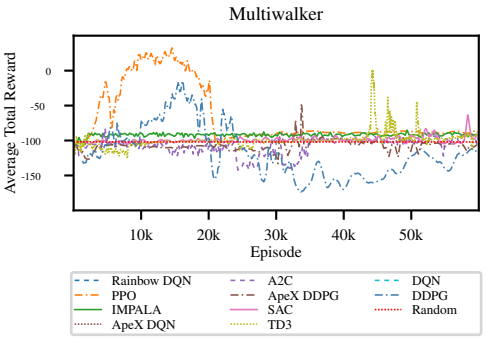
Figure 3: These illustrate the massive performance improvements of parameter sharing bootstrapping modern DRL methods for the first time, and the best documented average total rewards and convergence rates, on the Gupta et al. (2017) MARL benchmark environments. Average reward was computed over 1000 plays.



(a) Maximum average total average reward was 538 after 37.1k episodes using Apex-DQN.



(b) Maximum average total average reward was 18 after 31.8k episodes using ApeX-DDPG.



(c) Maximum average total average reward was 38 after 14.6k episodes using PPO.

Figure 4: These show the performance of the same DRL methods as Figure 3, but with fully independent learning instead of parameter sharing. The maximum average total reward, as well as convergence times, are worse, as predicted by our theoretical work in section 3.

A group of agents is *homogeneous* if the policy of one can be traded with another without affecting outcomes. Formally, homogeneous agents are agents which have identical action spaces, observation spaces, and reward functions, and for which the transition function is symmetric with respect to permutations of the actions. If agents are not homogeneous, they're said to be heterogeneous.

However, the challenge of heterogeneous groups of agents can be addressed, as per our intuition in section 1. The first challenge is that the policy must have an idea what agent (or kind of agent) it's seeing. This can be done by overlaying the agent on an image, appending the value to a vector observation, vel sim.

The following theorem, proven in Appendix A, states that an optimal policy can be learned when the observation spaces for agents are disjoint (and thus individual agents can clearly be distinguished).

Theorem 4. *For any POSG* $G = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\} \rangle$ *with disjoint observation spaces, there exists a single (shared) 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*.*

Now we formalize "agent indication" to prove that even in the case of non-disjoint observation spaces, a single shared policy can be learned by "tagging" observations with an indicator of the agent.

Theorem 5. *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, $f_i(\omega) = (\omega, i)$). □

The next issue is handling agents with differing heterogeneous observation sizes or action sizes. While this is not an issue theoretically (in terms of formulation of the POSG), it does pose important implementation issues. This can simply be resolved by "padding" the observations to to a uniform size. Similarly, if we have heterogeneous action spaces, we can utilize a method akin to an inverse of what we propose for homogenizing observation sizes. Assuming that all actions are converted into a common type, we can pad all the action spaces to the size of the largest, and discard actions outside of the "real" range in a reasonable manner. We formally define our methods for observation padding and prove that the methods allow for an optimal policy to be learned in Appendix A. Based on a preprint version of this work, all three methods are included as wrappers for multi-agent environments in SuperSuit (Terry et al., 2020a). Similarly based on a preprint, all three methods were successfully experimentally together and separately in (Terry et al., 2020b).

6 Conclusion, Limitations and Future Work

We derived a lower bound in the MAILP model on the convergence of single agent learning, and a larger lower bound on the convergence of multi-agent learning. Moreover, when $\Lambda(x) = x$, we have an upper bound on the convergence for single agent learning, demonstrating a proper gap in the convergence times in this case. We expect this gap to be even larger when $\Lambda(x) < x$. Furthermore, as parameter sharing is the most centralized form of learning, we show it bypasses nonstationarity in multi-agent reinforcement learning.

We further showed parameter sharing to be able to achieve up to 44 times more total average reward in as few as 16% as many episodes by bootstrapping modern DRL methods in the benchmark environments from Gupta et al. (2017), achieving the best documented results on those environments. We did this by using parameter sharing with 8 more modern DRL methods for the first time. We also outperformed MADDPG and QMIX on all environments tested. This shows parameter sharing to hold far more potential than previously realized for MARL.

We additionally show parameter sharing to be compatible with all forms of agent heterogeity. In the case of functional heterogeity, we prove that "agent indication" from Gupta et al. (2017) allows parameter sharing to learn policies for heterogeneous agents. In the cases of heterogenous action or observation spaces, we introduce methods of "observation padding" and "action space padding," and prove they all allow parameter sharing to converge to optimal policies. Based on a preprint

version of this work, these methods have been successfully experimentally and are included in a set of multi-agent wrappers so that anyone can easily use them.

Our work is limited in that, due to the computational resources required, we didn't compare a host of MADRL methods with learning centralization in between parameter sharing and fully independent learning (such as shared critic methods); the level of variance between runs at that level necessitate many runs of each method and hyperparameter searches to get clear results. Furthermore, our bound for Theorem 1 is only tight when $\Lambda(x) = x$ (leave exploring alternative forms of $\Lambda(x)$ as future work). We finally hope that, motivated by the work, parameter sharing is attempted in a variety of real world cooperative multi-agent scenarios.

REFERENCES

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

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

Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

Iadine Chades, Josie Carwardine, Tara Martin, Samuel Nicol, Regis Sabbadin, and Olivier Buffet. Momdps: A solution for modelling adaptive management problems. In *AAAI Conference on Artificial Intelligence*, 2012.

Samuel P. M. Choi, Dit-Yan Yeung, and Nevin Lianwen Zhang. An environment model for nonstationary reinforcement learning. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 987–993. MIT Press, 2000.

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

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

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

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 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*, pages 66–83. Springer, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. *arXiv preprint arXiv:1712.09381*, 2017.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

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*, pages 6379–6390, 2017.

Laetitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012. doi: 10.1017/S0269888912000057.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, and Stefano V Albrecht. Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint arXiv:1906.04737*, 2019.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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.

Justin K Terry and Nathaniel Grammel. Multi-agent informational learning processes. *arXiv preprint arXiv:2006.06870*, 2020.

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

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

A OMITTED PROOFS

A.1 CENTRALIZATION AND NONSTATIONARITY

We first state a rather more general lemma, whose utility will be clear later in our analysis.

Lemma 1. *Let $\alpha \geq 0$ and $C \geq 0$ be real constants, and $g(t)$ be a function such that $g(t) \leq g(t-1) + \alpha(C - g(t-1))$. Then, $g(t) \leq C - (1-\alpha)^t(C - g(0))$.*

Proof of Lemma 1. Let $g_0 = g(0)$. It is easy to see that $C-(1-\alpha)^0(C-g(0)) = C-(C-g(0)) = g(0)$. For the inductive step, we suppose $g(t-1) \leq C - (1-\alpha)^{t-1}(C - g(0))$ and proceed to bound $g(t)$:

$$\begin{aligned} g(t) &\leq g(t-1) + \alpha(C - g(t-1)) \\ &= g(t-1) + \alpha C - \alpha g(t-1) \\ &= g(t-1)(1-\alpha) + \alpha C \\ &\leq (1-\alpha)\left(C - (1-\alpha)^{t-1}(C - g(0))\right) + \alpha C \\ &= C - (1-\alpha)^t(C - g(0)) \end{aligned}$$

The final inequality in the above follows from the Inductive Hypothesis. □

We state the lemma as an inequality since it matches the way in which we will utilize it in the proofs of Theorem 1 and Theorem 2 below. However, note that the above proof also holds when we have strict equality; i.e. if $g(t) = g(t-1) + \alpha(C - g(t-1))$, then $g(t) = C - (1-\alpha)^t(C - g(0))$.

Proof of Theorem 1. We first write out the update equation for the agent. Note that since there are no other agents, only information about the environment needs to be learned. Thus, $\mathcal{I}(t) = \mathcal{I}_{env}(t)$. Recalling also that $\mathcal{C}_{env} = 1$, we get (4).

$$\begin{aligned} \mathcal{I}(t) &= \mathcal{I}(t-1) + \Delta^{\uparrow}\mathcal{I}(t) \\ &= \mathcal{I}(t-1) + \mathcal{K}_{env}\Lambda(1 - \mathcal{I}(t-1)) \end{aligned} \tag{4}$$

Since the function Λ has the property that $\Lambda(x) \leq x$, we have that $\mathcal{I}(t) \leq \mathcal{I}(t-1) + \mathcal{K}_{env}(1 - \mathcal{I}(t-1))$. Thus, we can apply Lemma 1 to get $\mathcal{I}(t) \leq 1 - (1 - \mathcal{K}_{env})^t(1 - \mathcal{I}^0)$.

Next, let $t^* = \frac{\log(\epsilon) - \log(1-\mathcal{I}^0)}{\log(1-\mathcal{K}_{env})}$ and consider $\mathcal{I}(t^*)$:

$$\begin{aligned} \mathcal{I}(t^*) &\leq 1 - (1 - \mathcal{K}_{env})^{t^*}(1 - \mathcal{I}^0) \\ &= 1 - (1 - \mathcal{K}_{env})^{\log_{(1-\mathcal{K}_{env})}\left(\frac{\epsilon}{1-\mathcal{I}^0}\right)}(1 - \mathcal{I}^0) \\ &= 1 - \left(\frac{\epsilon}{1 - \mathcal{I}^0}\right)(1 - \mathcal{I}^0) = 1 - \epsilon \end{aligned}$$

Thus, at time t^*, $\mathcal{I}(t^*) \leq 1 - \epsilon$, and so $\mathcal{I}^{-1}(1-\epsilon) \geq t^*$. □

Note that since the inequality only arises from upper bounding $\Lambda(x) \leq x$, and Lemma 1 holds for equality, Theorem 1 actually gives an exact result for convergence time in the case when $\Lambda(x) = x$ (note, however, that this is not a realistic Λ for real-world algorithms).

Proof of Theorem 2. We begin by considering the function $\mathcal{I}_{i,j}(t)$, which will be the same for all pairs of agents i, j.

$$\begin{aligned} \mathcal{I}_{i,j}(t) &= \mathcal{I}_{i,j}(t) + \Delta^{\uparrow}\mathcal{I}_{i,j}(t) - \Delta^{\downarrow}\mathcal{I}_{i,j}(t) \\ \Delta^{\uparrow}\mathcal{I}_{i,j}(t) &= \mathcal{K}_{\star}\Lambda(\mathcal{C}_{\star} - \mathcal{I}_{i,j}(t-1)) \\ \Delta^{\downarrow}\mathcal{I}_{i,j}(t) &= \frac{\Delta^{\uparrow}\mathcal{I}_j(t)\mathcal{I}_{i,j}(t-1)}{\mathcal{I}_j(t-1) + \Delta^{\uparrow}\mathcal{I}_j(t)} \end{aligned} \tag{5}$$

In order to further expand $\Delta^{\downarrow}\mathcal{I}_{i,j}(t)$, we must first write out $\Delta^{\uparrow}\mathcal{I}_j(t)$ as below.

$$\begin{aligned}\Delta^{\uparrow}\mathcal{I}_j(t) &= \sum_{j' \neq j} \Delta^{\uparrow}\mathcal{I}_{j,j'}(t) \\ &= \sum_{j' \neq j} \mathcal{K}_\star \Lambda(\mathcal{C}_\star - \mathcal{I}_{j,j'}(t-1))\end{aligned}$$

To allow us to simplify these expressions further, we note that since all agents are modeled identically in this setting, $\mathcal{I}_{i,j}$ is the same for all agents i, j. We will denote thus denote by $\mathcal{I}_{\star,\star}(t)$ this value (thus, $\mathcal{I}_{i,j}(t) = \mathcal{I}_{\star,\star}(t)$ for all $i, j \in [N]$). Thus, we get $\Delta^{\uparrow}\mathcal{I}_j(t) = (n-1)\mathcal{K}_\star \Lambda(\mathcal{C}_\star - \mathcal{I}_{\star,\star}(t-1))$.

This gives us the full expansion of $\Delta^{\downarrow}\mathcal{I}_{i,j}(t)$ below.

$$\Delta^{\downarrow}\mathcal{I}_{\star,\star}(t) = \frac{(n-1)\mathcal{K}_\star \Lambda(\mathcal{C}_\star - \mathcal{I}_{\star,\star}(t-1))\mathcal{I}_{i,j}(t-1)}{\mathcal{I}_j(t-1) + (n-1)\mathcal{K}_\star \Lambda(\mathcal{C}_\star - \mathcal{I}_{\star,\star}(t-1))} \tag{6}$$

We now consider the *net information change* $\Delta\mathcal{I}_{\star,\star}(t) = \Delta^{\uparrow}\mathcal{I}_{\star,\star}(t) - \Delta^{\downarrow}\mathcal{I}_{\star,\star}(t)$. By using (5) and (6), we get the below equation.

$$\begin{aligned}\Delta\mathcal{I}_{\star,\star}(t) &= \mathcal{K}_\star \Lambda(\mathcal{C}_\star - \mathcal{I}_{\star,\star}(t-1))(1 - \Phi(t-1)) \\ \Phi(t-1) &= \frac{\mathcal{I}_{\star,\star}(t-1)}{\mathcal{C}_{env}/(n-1) + \mathcal{I}_{\star,\star}(t-1) + \mathcal{K}_\star \Lambda(\mathcal{C}_\star - \mathcal{I}_{\star,\star}(t-1))}\end{aligned}$$

We now upper bound the denominator of this expression.

$$\begin{aligned}&\mathcal{I}_{\star,\star}(t-1) + \mathcal{K}_\star \Lambda(\mathcal{C}_\star - \mathcal{I}_{\star,\star}(t-1)) \\ &\leq \mathcal{I}_{\star,\star}(t-1) + \mathcal{K}_\star \mathcal{C}_\star - \mathcal{K}_\star \mathcal{I}_{\star,\star}(t-1) \\ &\leq \mathcal{K}_\star \mathcal{C}_\star + \mathcal{C}_\star = \mathcal{C}_\star(\mathcal{K}_\star + 1)\end{aligned}$$

This leads to the following bound on $\Delta\mathcal{I}_{\star,\star}(t)$, using the fact that $\mathcal{I}_{\star,\star}(t) \geq \mathcal{I}^0_{\star,\star}$.

$$\Delta\mathcal{I}_{\star,\star}(t) \leq \mathcal{K}_\star(\mathcal{C}_\star - \mathcal{I}_{\star,\star}(t-1))\left(1 - \frac{\mathcal{I}^0_{\star,\star}}{\frac{\mathcal{C}_{env}}{n-1} + \mathcal{C}_\star(\mathcal{K}_\star + 1)}\right)$$

This upper bound is now in a form that allows us to once again apply Lemma 1, to get (7).

$$\mathcal{I}_{\star,\star}(t) \leq \mathcal{C}_\star - (\mathcal{C}_\star - \mathcal{I}^0_{\star,\star}) \cdot \left(1 - \mathcal{K}_\star\left(1 - \frac{\mathcal{I}^0_{\star,\star}}{\frac{\mathcal{C}_{env}}{n-1} + \mathcal{C}_\star(\mathcal{K}_\star + 1)}\right)\right)^t \tag{7}$$

One can then verify from (7) that $\mathcal{I}_{\star,\star}(t^*) \leq \mathcal{C}_\star(1 - \epsilon')$. This, together with (3) gives the desired result:

$$\begin{aligned}\mathcal{I}_i(t^*) &= \mathcal{I}_{i,env}(t^*) + \sum_{j \in [N]_{-i}} \mathcal{I}_{i,j}(t^*) \\ &= (1-\epsilon)\mathcal{C}_{env} + (n-1)\mathcal{I}_{\star,\star}(t^*) \\ &\leq (1-\epsilon)\mathcal{C}_{env} + (n-1)\mathcal{C}_\star(1-\epsilon') \\ &= 1 - \epsilon\end{aligned}$$

□

Proof of Theorem 3. We examine the limit as $\mathcal{K}_\star \to 1$.

$$\begin{aligned}
&\lim_{\mathcal{K}_\star \to 1} \frac{\log(\mathcal{C}_\star \epsilon/(\mathcal{C}_\star - \mathcal{I}^0_{\star,\star}))}{\log\left(1 - \mathcal{K}_\star + \frac{\mathcal{K}_\star \mathcal{I}^0_{\star,\star}}{\mathcal{C}_{env}/(n-1) + \mathcal{C}_\star(\mathcal{K}_\star + 1)}\right)} \\
&= \frac{\log(\mathcal{C}_\star \epsilon/(\mathcal{C}_\star - \mathcal{I}^0_{\star,\star}))}{\log\left(\frac{\mathcal{I}^0_{\star,\star}}{\mathcal{C}_{env}/(n-1) + 2\mathcal{C}_\star}\right)} \\
&= \frac{\log(\mathcal{C}_\star \epsilon/(\mathcal{C}_\star - \mathcal{I}^0_{\star,\star}))}{\log\left(\frac{\mathcal{I}^0_{\star,\star}(n-1)}{1 + (n-1)\mathcal{C}_\star)}\right)}
\end{aligned}$$

□

A.2 Heterogeneous Agents

Lemma 2. *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 of Lemma 2. 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.

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

$$\pi^{[N]}(\omega, a) = \begin{cases} \pi_{\Omega^{-1}(\omega)}(\omega, a) & \text{if } a \in \mathcal{A}_{\Omega^{-1}(\omega)} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

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. □

Theorem 4 then follows immediately from Lemma 2.

A.2.1 Heterogeneous Observation and Action Spaces

Suppose a learning algorithm for agent $i \in [N]$ has learned a policy $\pi_i\colon \Omega_i \times \mathcal{A}_i \to [0,1]$. This is often implemented as a function $f_{\pi_i}\colon \mathbb{R}^{o_i} \to [0,1]^{l_i}$, where $l_i = |\mathcal{A}_i|$. Note the domain is $\mathbb{R}^{o_i}$, as observations are often represented as a fixed-size vector so that $\Omega \subseteq \mathbb{R}^{o_i}$ for a fixed integer o_i. Using these padding techniques, we can implement a shared policy for all agents as $f_{\pi^{[N]}}\colon \mathbb{R}^{\bar{o}} \to [0,1]^{\alpha}$, where $\bar{o} = \max_{i\in[N]} o_i$ (i.e. the largest observation size of any agent) and $\alpha = \max_{i\in[N]} |\mathcal{A}_i|$. To accomplish this with heterogeneous observation sizes, we "pad" an observation $\omega = (\omega_1, \omega_2, \ldots, \omega_{o_i})$ to produce a padded observation $\omega'_i = (\omega_1, \omega_2, \ldots, \omega_{o_i}, 0, 0, \ldots, 0) \in \mathbb{R}^{\bar{o}}$ of dimension $\bar{o}$. If we use the "agent indication" technique of Theorem 5, we also add the identity of agent i, yielding the padded observation $\omega' = (\omega_1, \omega_2, \ldots, \omega_{o_i}, 0, 0, \ldots, 0, i) \in \mathbb{R}^{\bar{o}}$, where now $\bar{o} = \max_{i\in[N]} o_i + 1$ to allow for the i at the end of the observation, as per Theorem 5. For the issue of heterogeneous action sizes, suppose $\mathcal{A}_i = \{a_1, a_2, \ldots, a_{l_i}\}$. The learning algorithm will return a vector $\vec{a} \in [0,1]^{\alpha}$ padded with zeros at the end, i.e. so that $\vec{a}_s = 0$ for all $s > l_i$. Agent i can then "trim" the result and consider only the subvector $(\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{l_i})$, taking action a_s with probability $\vec{a}_s$.

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

B DETAILED ENVIRONMENT DESCRIPTIONS

In the *pursuit* environment, shown in Figure 2a, there are 30 blue evaders and 8 red pursuer agents, in a 16×16 grid with an obstacle in the center, shown in white. The evaders move randomly, and the pursuers are controlled. Every time the pursuers fully surround an evader, each of the surrounding agents receives a reward of 5, and the evader is removed from the environment. Pursuers also receive a reward of 0.01 every time they touch an evader. The pursuers have a discrete action space of up, down, left, right and stay. Each pursuer observes a 7×7 grid centered around itself, depicted by the orange boxes surrounding the red pursuer agents. The environment ends after 500 steps.

In the *waterworld* environment, there are 5 agents (purple), 5 food targets (green) and 10 poison targets (red), as shown in Figure 2b. Each agent has 30 range-limited sensors, depicted by the black lines, to detect neighboring agents, food and poison targets, resulting in 212 long vector of computed values about the environment for the observation space. They have a continuous action space represented as a 2 element vector, which corresponds to left/right and up/down thrust. The agents each receive a reward of 10 when more than one agent captures food together (the food is not destroyed), a shaping reward of 0.01 for touching food, a reward of -1 for touching poison, and a small negative reward when two agents collide based on the force of the collision. The environment ends after 500 steps.

In the *multiwalker* environment, shown in Figure 2c, there is a package placed on top of 3 pairs of robot legs which you control. The robots must learn to move the package as far as possible to the right. Each walker gets a reward of 1 for moving the package forward, and a reward of -100 for dropping the package. Each walker exerts force on two joints in their two legs, giving a continuous action space represented as a 4 element vector. Each walker observes via a 32 element vector, containing simulated noisy lidar data about the environment and information about neighboring walkers. The environment ends after 500 steps.

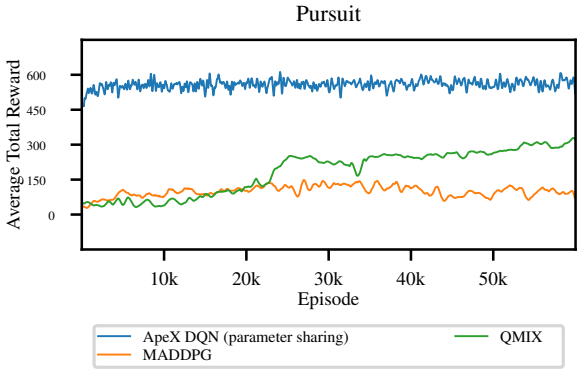
C EMPIRICAL COMPARISON TO MADDPG AND QMIX

We present additional results in Figure 5 comparing the performance of the best parameter sharing method against MADDPG and QMIX. We use the reference implementation of MADDPG from Lowe et al. (2017) and QMIX from Samvelyan et al. (2019) (improved implementation by authors of Rashid et al. (2018)). For QMIX, we also had to adapt the continuous action spaces to discrete by assigning discrete values to uniformly random action space samples. We adapted the network architecture to be the same in our other experiments, and hyperparameters were left to their default values in the reference codebase except those standard across all experiments. The full list of hyperparameters is in Appendix D. QMIX was adapted to continuous actions via binning, the original MADDPG supports it natively.

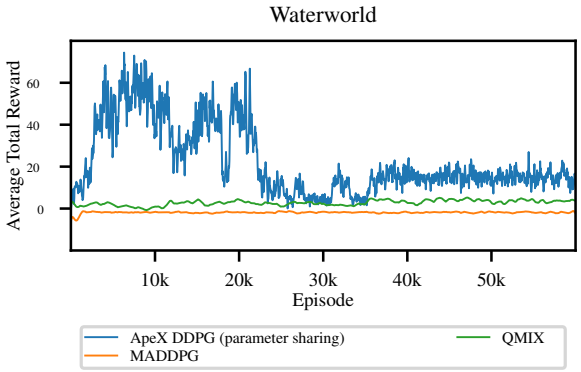
These results are insufficient to conclusively comment on the utility of the underlying strategies taken by QMIX and MADDPG that differ from vanilla parameter sharing. Doing so would require *very* large experiments involving different learning methods, environments, and hyperparameters. Rather our experiments offer a simple performance compression to more widely known methods, and demonstrate the apparent utility of vanilla parameter sharing from the perspective of a practitioner.

D HYPERPARAMETERS

Hyperparameters for various RL methods and for various games are given in Tables 1, 3, and 2. Some hyperparameter values are constant across all RL methods for all games. These constant values are specified in Table 4.



(a) QMIX reached a maximum average total reward of 363.2 after 59.6k episodes, MADDPG reached a maximum average total reward of 32.42 after 28.8k episodes. APEX DQN (parameter sharing) reached a maximum average total reward of 621 after 24k episodes



(b) QMIX reached a maximum average total reward of 8.09 after 45.3k episodes, MADDPG reached a maximum average total reward of -5.43 after 41k episodes. APEX DDPG (parameter sharing) reached a maximum average total reward of 74 after 6.3k episodes

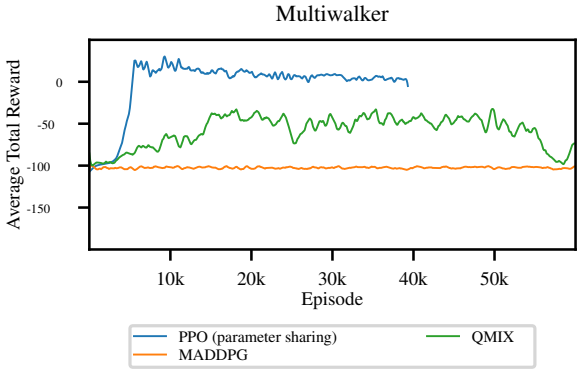


Figure 5: QMIX reached a maximum average total reward of -16.935 after 17.8k episodes, MADDPG reached a maximum average total reward of -101.85 after 44.2k episodes. PPO (parameter sharing) reached a maximum reward of 41 after average total 9.2k episodes

RL method	Hyperparameter	Value for Pursuit / Waterworld / Multiwalker
PPO	`sample_batch_size`	100
	`train_batch_size`	5000
	`sgd_minibatch_size`	500
	`lambda`	0.95
	`kl_coeff`	0.5
	`entropy_coeff`	0.01
	`num_sgd_iter`	10
	`vf_clip_param`	10.0
	`clip_param`	0.1
	`vf_share_layers`	`True`
	`clip_rewards`	`True`
	`batch_mode`	`truncate_episodes`
IMPALA	`sample_batch_size`	20
	`train_batch_size`	512
	`lr_schedule`	[[0, 5e-3], [2e7, 1e-12]]
	`clip_rewards`	`True`
A2C	`sample_batch_size`	20
	`train_batch_size`	512
	`lr_schedule`	[[0, 7e-3], [2e7, 1e-12]]

Table 1: Hyperparameters for Pursuit / Waterworld / Multiwalker

RL method	Hyperparameter	Value for Waterworld / Multiwalker
APEX-DDPG	sample_batch_size	20
	train_batch_size	512
	lr	0.0001
	beta_annealing_fraction	1.0
	exploration_fraction	0.1
	final_prioritized_replay_beta	1.0
	n_step	3
	prioritized_replay_alpha	0.5
	learning_starts	1000
	buffer_size	100000
	target_network_update_freq	50000
	timesteps_per_iteration	25000
Plain DDPG	sample_batch_size	20
	train_batch_size	512
	learning_starts	5000
	buffer_size	100000
	critics_hidden	[256, 256]
SAC	sample_batch_size	20
	train_batch_size	512
	Q_model	{hidden_activation: relu, hidden_layer_sizes: [266, 256]}
	optimization	{actor_learning_rate: 0.0003, actor_learning_rate: 0.0003, entropy_learning_rate: 0.0003,}
	clip_actions	False
	exploration_enabled	True
	no_done_at_end	True
	normalize_actions	False
	prioritized_replay	False
	soft_horizon	False
	target_entropy	auto
	tau	0.005
	n_step	1
	evaluation_interval	1
	metrics_smoothing_episodes	5
	target_network_update_freq	1
	learning_starts	1000
	timesteps_per_iteration	1000
	buffer_size	100000
TD3	sample_batch_size	20
	train_batch_size	512
	critics_hidden	[256, 256]
	learning_starts	5000
	pure_exploration_steps	5000
	buffer_size	100000

Table 2: Hyperparameters for Waterworld / Multiwalker

RL method	Hyperparameter	Value for Pursuit
APEX-DQN	sample_batch_size	20
	train_batch_size	512
	learning_starts	1000
	buffer_size	100000
	dueling	True
	double_q	True
Rainbow-DQN	sample_batch_size	20
	train_batch_size	512
	learning_starts	1000
	buffer_size	100000
	n_step	2
	num_atoms	51
	v_min	0
	v_max	1500
	prioritized_replay	True
	dueling	True
	double_q	True
	parameter_noise	True
	batch_mode	complete_episodes
Plain DQN	sample_batch_size	20
	train_batch_size	512
	learning_starts	1000
	buffer_size	100000
	dueling	False
	double_q	False
QMIX	buffer_size	3000
	critic_lr	0.0005
	gamma	0.99
	critic_lr	0.0005
	lr	0.0005
	grad_norm_clip	10
	optim_alpha	0.99
	optim_eps	0.05
	epsilon_finish	0.05
	epsilon_start	1.0
MADDPG	lr	0.0001
	batch_size	512
	num_envs	64
	num_cpus	8
	buffer_size	1e5
	steps_per_update	4

Table 3: Hyperparameters for Pursuit

Variable	Value set in all RL methods
# worker threads	8
# envs per worker	8
gamma	0.99
MLP hidden layers	[400, 300]

Table 4: Variables set to constant values across all RL methods for all RL games