Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning

Filippos Christianos
School of Informatics
University of Edinburgh
f.christianos@ed.ac.uk

Lukas Schäfer
School of Informatics
University of Edinburgh
l.schaefer@ed.ac.uk

Stefano V. Albrecht
School of Informatics
University of Edinburgh
s.albrecht@ed.ac.uk

Abstract

Exploration in multi-agent reinforcement learning is a challenging problem, especially in environments with sparse rewards. We propose a general method for efficient exploration by sharing experience amongst agents. Our proposed algorithm, called *Shared Experience Actor-Critic* (SEAC), applies experience sharing in an actor-critic framework by combining the gradients of different agents. We evaluate SEAC in a collection of sparse-reward multi-agent environments and find that it consistently outperforms several baselines and state-of-the-art algorithms by learning in fewer steps and converging to higher returns. In some harder environments, experience sharing makes the difference between learning to solve the task and not learning at all.

1 Introduction

Multi-agent reinforcement learning (MARL) necessitates exploration of the environment dynamics and of the joint action space between agents. This is a difficult problem due to non-stationarity caused by concurrently learning agents [27] and the fact that the joint action space typically grows exponentially in the number of agents. The problem is exacerbated in environments with sparse rewards in which most transitions will not yield informative rewards.

We propose a general method for efficient MARL exploration by *sharing experience* amongst agents. Consider the simple multi-agent game shown in Figure 1 in which two agents must simultaneously arrive at a goal. This game presents a difficult exploration problem, requiring the agents to wander for a long period before stumbling upon a reward. When the agents finally succeed, the idea of sharing experience is appealing: both agents can learn how to approach the goal from two different directions after a successful episode by leveraging their collective experience. Such experience sharing facilitates a steady progression of all learning agents, meaning that agents improve at approximately equal rates as opposed to diverging in their learning progress. We show in our experiments that this approach of experience sharing can lead to significantly faster learning and higher final returns.

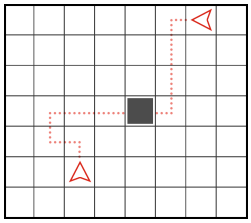


Figure 1: Two randomly-placed agents (triangles) must simultaneously arrive at the goal (square).

We demonstrate this idea in a novel actor-critic MARL algorithm, called *Shared Experience Actor-Critic* (SEAC).[1] SEAC operates similarly to independent learning [34] but updates the actor and critic parameters of an agent by combining gradients computed on the agent's experience with

[1] We provide open-source implementations of SEAC in `www.github.com/uoe-agents/seac` and our two newly developed multi-agent environments: `www.github.com/uoe-agents/lb-foraging` (LBF) and `www.github.com/uoe-agents/robotic-warehouse` (RWARE).

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

weighted gradients computed on other agents' experiences. We evaluate SEAC in four sparse-reward multi-agent environments and find that it learns substantially faster (up to 70% fewer required training steps) and achieves higher final returns compared to several baselines, including: independent learning without experience sharing; using data from all agents to train a single shared policy; and MADDPG [21], QMIX [29], and ROMA [37]. Sharing experience with our implementation of SEAC increased running time by less than 3% across all environments compared to independent learning.

2 Related Work

Centralised Training with Decentralised Execution: The prevailing MARL paradigm of centralised training with decentralised execution (CTDE) [25, 29, 21] assumes a training stage during which the learning algorithm can access data from all agents to learn decentralised (locally-executable) agent policies. CTDE algorithms such as MADDPG [21] and COMA [12] learn powerful critic networks conditioned on joint observations and actions of all agents. A crucial difference to SEAC is that algorithms such as MADDPG only reinforce an agent's own tried actions, while SEAC uses shared experience to reinforce good actions tried by any agent, without learning the more complex joint-action critics. Our experiments show that MADDPG was unable to learn effective policies in our sparse-reward environments while SEAC learned successfully in most cases.

Agents Teaching Agents: There are approaches to leverage expertise of *teacher* agents to address the issue of sample complexity in training a *learner* agent [8]. Such teaching can be regarded as a form of transfer learning [26] among RL agents. The *teacher* would either implicitly or explicitly be asked to evaluate the behaviour of the *learner* and send instructions to the other agent. Contrary to our work, most such approaches do focus on single-agent RL [5, 11]. However, even in such teaching approaches for multi-agent systems [6, 7] experience is shared in the form of knowledge exchange following a teacher-learner protocol. Our approach shares agent trajectories for learning and therefore does not rely on the exchange of explicit queries or instructions, introducing minimal additional cost.

Learning from Demonstrations: Training agents from trajectories [31] of other agents [41] or humans [35] is a common case of teaching agents. Demonstration data can be used to derive a policy [16] which might be further refined using typical RL training [13] or to shape the rewards biasing towards previously seen expert demonstrations [4]. These approaches leverage expert trajectories to speed up or simplify learning for single-agent problems. In contrast, SEAC makes use of trajectories from other agents which are generated by concurrently learning agents in a multi-agent system. As such, we aim to speed up and synchronise training in MARL whereas learning from demonstrations focuses on using previously generated data for application in domains like robotics where generating experience samples is expensive.

Distributed Reinforcement Learning: Sharing experience among agents is related to recent work in distributed RL. These methods aim to effectively use large-scale computing resources for RL. Asynchronous methods such as A3C [23] execute multiple actors in parallel to generate trajectories more efficiently and break data correlations. Similarly, IMPALA [10] and SEED RL [9] are off-policy actor-critic algorithms to distribute data collection across many actors with optimisation being executed on a single learner. Network parameters, observations or actions are exchanged after each episode or timestep, respectively, and off-policy correction is applied. However, all these approaches only share experience of multiple actors to speed up learning of a single RL agent and by breaking correlations in the data rather than addressing synchronisation and sample efficiency in MARL.

Population-play: Population-based training is another line of research aiming to improve exploration and coordination in MARL by training a population of diverse sets of agents [36, 18, 17, 20]. Leibo et al. [18] note the overall benefits on exploration when sets of agents are dynamically evolved and mixed. In their work, some agents share policy networks and are trained alongside evolving sets of agents. Similarly to Leibo et al. [18], we observe benefits on exploration due to agents influencing each other's trajectories. However, SEAC is different from such population-play as it only trains a single set of distinct policies for all agents, thereby avoiding the significant computational cost involved in training multiple sets of agents.

3 Technical Preliminaries

Markov Games: We consider Markov games for N agents (e.g. [19]) with partial observability (also known as partially observable stochastic games, e.g. [14]) defined by the tuple $(\mathcal{N}, \mathcal{S}, \{O^i\}_{i\in\mathcal{N}}, \{A^i\}_{i\in\mathcal{N}}, \Omega, \mathcal{P}, \{\mathcal{R}^i\}_{i\in\mathcal{N}})$, with agents $i \in \mathcal{N} = \{1, \dots, N\}$, state space $\mathcal{S}$, joint observation space $\mathcal{O} = O^1 \times \dots \times O^N$, and joint action space $\mathcal{A} = A^1 \times \dots \times A^N$. Each agent i only perceives local observations $o^i \in O^i$ which depend on the state and applied joint action via observation function $\Omega : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{O})$. The function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$ returns a distribution over successor states given a state and a joint action. $\mathcal{R}^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function for agent i which gives its individual reward r^i. The objective is to find policies $\pi = (\pi_1, ..., \pi_N)$ for all agents such that the discounted return of each agent i, $G^i = \sum_{t=0}^{T} \gamma^t r_t^i$, is maximised with respect to other policies in π, formally $\forall_i : \pi_i \in \arg\max_{\pi_i'} \mathbb{E}[G^i | \pi_i', \pi_{-i}]$ where $\pi_{-i} = \pi \setminus \{\pi_i\}$, γ is the discount factor, and T the total timesteps of an episode.

In this work, we assume $O = O^1 = \dots = O^N$ and $A = A^1 = \dots = A^N$ in line with other recent works in MARL [32, 29, 12, 22]. However, in contrast to these works we do not require that agents have identical reward functions, as will be discussed in Section 4.

Policy Gradient and Actor-Critic: Policy Gradient (PG) algorithms are a class of model-free RL algorithms that aim to directly learn a policy π_ϕ parameterised by ϕ, that maximises the expected returns. In REINFORCE [39], the simplest PG algorithm, this is accomplished by following the gradients of the objective $\nabla_\phi J(\phi) = \mathbb{E}_\pi \left[G_t \nabla_\phi \ln \pi_\phi(a_t|s_t) \right]$. Notably, the Markov property is not used, allowing the use of PG in partially observable settings. However, REINFORCE suffers from high variance of gradient estimation. To reduce variance of gradient estimates, actor-critic (AC) algorithms estimate Monte Carlo returns using a value function $V_\pi(s; \theta)$ with parameters θ. In a multi-agent, partially observable setting, the simplest AC algorithm defines a policy loss for agent i

$$\mathcal{L}(\phi_i) = -\log \pi(a_t^i | o_t^i; \phi_i)(r_t^i + \gamma V(o_{t+1}^i; \theta_i) - V(o_t^i; \theta_i)) \tag{1}$$

with a value function minimising

$$\mathcal{L}(\theta_i) = ||V(o_t^i; \theta_i) - y_i||^2 \text{ with } y_i = r_t^i + \gamma V(o_{t+1}^i; \theta_i) \tag{2}$$

In practice, when V and π are parameterised by neural networks, sampling several trajectories in parallel, using n-step returns, regularisation, and other modifications can be beneficial [23]. To simplify our descriptions, our methods in Section 4 will be described only as extensions of Equations (1) and (2). In our experiments we use a modified AC algorithm as described in Section 5.3.

4 Shared Experience Actor-Critic

Our goal is to enable more efficient learning by sharing experience among agents. To facilitate experience sharing, we assume environments in which the local policy gradients of agents provide useful learning directions for all agents. Intuitively, this means that agents can learn from the experiences of other agents without necessarily having identical reward functions. Examples of such environments can be found in Section 5.

In each episode, each agent generates one on-policy trajectory. Usually, when on-policy training is used, RL algorithms only use the experience of each agent's own sampled trajectory to update the agent's networks with respect to Equation (1). Here, we propose to also use trajectories of other agents while considering that it is *off-policy* data, i.e. the trajectories are generated by agents executing different policies than the one optimised. Correcting for off-policy samples requires importance sampling. The loss for such off-policy policy gradient optimisation from a behavioural policy β can be written as

$$\nabla_\phi \mathcal{L}(\phi) = -\frac{\pi(a_t|o_t;\phi)}{\beta(a_t|o_t)} \nabla_\phi \log \pi(a_t|o_t;\phi)(r_t + \gamma V(o_{t+1};\theta) - V(o_t;\theta)) \tag{3}$$

In the AC framework of Section 3, we can extend the policy loss to use the agent's own trajectories (denoted with i) along with the experience of other agents (denoted with k), shown below:

$$\begin{aligned} \mathcal{L}(\phi_i) = &-\log \pi(a_t^i|o_t^i;\phi_i)(r_t^i + \gamma V(o_{t+1}^i;\theta_i) - V(o_t^i;\theta_i)) \\ &- \lambda \sum_{k\neq i} \frac{\pi(a_t^k|o_t^k;\phi_i)}{\pi(a_t^k|o_t^k;\phi_k)} \log \pi(a_t^k|o_t^k;\phi_i)(r_t^k + \gamma V(o_{t+1}^k;\theta_i) - V(o_t^k;\theta_i)) \end{aligned} \tag{4}$$

Algorithm 1 Shared Experience Actor-Critic Framework

for timestep $t = 1 \dots$ **do**
  Observe $o_t^1 \dots o_t^N$
  Sample actions $a_t^1, \dots, a_t^N$ from $P(o_t^1; \phi_1), \dots, P(o_t^N; \phi_N)$
  Execute actions and observe $r_t^1, \dots, r_t^N$ and $o_{t+1}^1, \dots, o_{t+1}^N$
  for agent $i = 1 \dots N$ **do**
    Perform gradient step on ϕ_i by minimising Eq. (4)
    Perform gradient step on θ_i by minimising Eq. (5)
  end for
end for

Using this loss function, each agent is trained on both on-policy data while also using the off-policy data collected by all other agents at each training step. The value loss, in a similar fashion, becomes

$$\begin{aligned} \mathcal{L}(\theta_i) &= ||V(o_t^i; \theta_i) - y_i^i||^2 + \lambda \sum_{k \neq i} \frac{\pi(a_t^k | o_t^k; \phi_i)}{\pi(a_t^k | o_t^k; \phi_k)} ||V(o_t^k; \theta_i) - y_k^i||^2 \\ y_k^i &= r_t^k + \gamma V(o_{t+1}^k; \theta_i) \end{aligned} \tag{5}$$

We show how to derive the losses in Equations (4) and (5) for the case of two agents in Appendix C (generalisation to more agents is possible). The hyperparameter λ weights the experience of other agents; we found SEAC to be largely insensitive to values of λ and use $\lambda = 1$ in our experiments. A sensitivity analysis can be found in Appendix B. We refer to the resulting algorithm as *Shared Experience Actor-Critic* (SEAC) and provide pseudocode in Algorithm 1.

Due to the random weight initialisation of neural networks, each agent is trained from experience generated from different policies, leading to more diverse exploration. Similar techniques, such as annealing ϵ-greedy policies to different values of ϵ, have been observed [23] to improve the performance of algorithms.

It is possible to apply a similar concept of experience sharing to off-policy deep RL methods such as DQN [24]. We provide a description of experience sharing with DQN in Appendix D. Since DQN is an off-policy algorithm, experience generated by different policies can be used for optimisation without further considerations such as importance sampling. However, we find deep off-policy methods to exhibit rather unstable learning [15] compared to on-policy AC. We consider the generality of our method a strength, and believe it can improve other multi-agent algorithms (e.g. AC with centralised value function).

5 Experiments

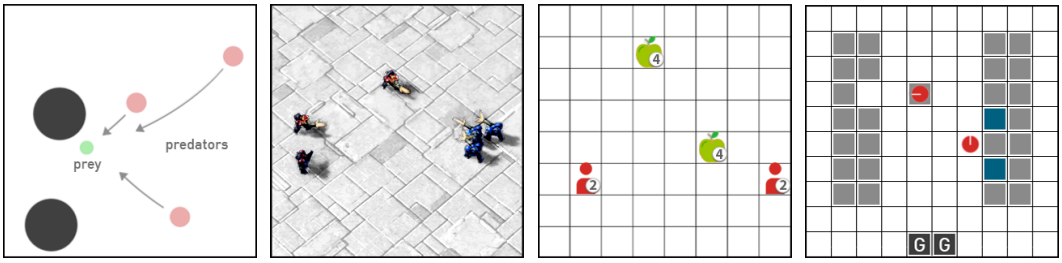
We conduct experiments on four sparse-reward multi-agent environments and compare SEAC to two baselines as well as three state-of-the-art MARL algorithms: MADDPG [21], QMIX [29] and ROMA [37].

5.1 Environments

The following multi-agent environments were used in our evaluation. More detailed descriptions of these environments can be found in Appendix A.

Predator Prey (PP), Fig. 2a: First, we use the popular PP environment adapted from the Multi-agent Particle Environment framework [21]. In our sparse-reward variant, three predator agents must catch a prey by coordinating and approaching it simultaneously. The prey is a slowly moving agent that was pretrained with MADDPG and dense rewards to avoid predators. If at least two predators are adjacent to the prey, then they succeed and each receive a reward of one. Agents are penalised for leaving the bounds of the map, but otherwise receive zero reward.

Starcraft Multi-Agent Challenge (SMAC), Fig. 2b: The SMAC [30] environment was used in several recent MARL works [29, 12, 37]. SMAC originally uses dense reward signals and is primarily designed to test solutions to the multi-agent credit assignment problem. We present experiments on a



(a) Predator Prey (b) SMAC - 3 marines (c) Level-Based Foraging (d) RWARE (10×11)

Figure 2: Environments used in our evaluation. Controlled agents are coloured red.

simple variant that uses sparse rewards. In this environment, agents have to control a team of marines each represented by a single agent, to fight against an equivalent team of marines controlled by the game AI. With sparse rewards agents receive a single non-zero reward at the final timestep of each episode: a victory rewards 1, while a defeat -1.

Level-Based Foraging (LBF), Fig. 2c: LBF [1, 3] is a mixed cooperative-competitive game which focuses on the coordination of the agents involved. Agents of different skill levels navigate a grid world and collect foods by cooperating with other agents if required. Four tasks of this game will be tested, with a varied number of agents, foods, and grid size. Also, a cooperative variant will be tested. The reported returns are the fraction of items collected in every episode.

Multi-Robot Warehouse (RWARE), Fig. 2d: This multi-agent environment (similar to the one used in [2]) simulates robots that move goods around a warehouse, similarly to existing real-world applications [40]. The environment requires agents (circles) to move requested shelves (coloured squares) to the goal posts (letter 'G') and back to an empty location. It is a partially-observable collaborative environment with a very sparse reward signal, since agents have a limited view area and are rewarded only upon successful delivery. In the results, we report the total returns given by the number of deliveries over an episode of 500 timesteps on four different tasks in this environment.

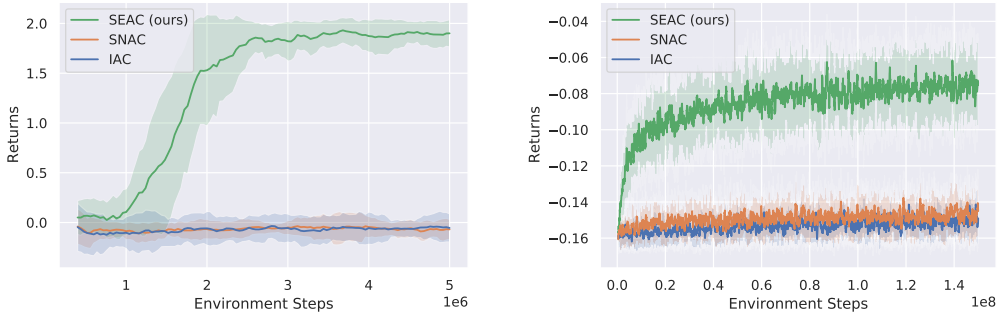
5.2 Baselines

Independent Actor-Critic (IAC): We compare SEAC to independent learning [34], in which each agent has its own policy network and is trained separately only using its own experience. IAC uses an actor-critic algorithm for each agent, directly optimising Eqs. (1) and (2); and treating other agents as part of the environment. Arguably, independent learning is one of the most straightforward approaches to MARL and serves as reasonable baseline due to its simplicity.

Shared Network Actor-Critic (SNAC): We also compare SEAC to training a single shared policy among all agents. During execution of the environment, each agents gets a copy of the policy and individually follows it. During training, the sum of policy and value loss gradients are used to optimise the shared parameters. Importance sampling is not required since all trajectories are on-policy. Improved performance of our SEAC method would raise the question whether agents simply benefit from processing more data during training. Comparing against this baseline can also show that agents trained using experience sharing are not learning identical policies but instead learn different ones despite being trained on the same collective experience.

5.3 Algorithm Details

For all tested algorithms, we implement AC using n-step returns and synchronous environments [23]. Specifically, 5-step returns were used and four environments were sampled and passed in batches to the optimiser. An entropy regularisation term was added to the final policy loss [23], computing the entropy of the policy of each individual agent. Hence, the entropy term of agent i, given by $H(\pi(o_t^i; \phi_i))$, only considers its own policy. High computational requirements in terms of environment steps only allowed hyperparameter tuning for IAC on RWARE; all tested AC algorithms use the same hyperparameters (see Appendix B). All results presented are averaged across five seeds, with the standard deviation plotted as a shaded area.



(a) PP, sparse rewards

(b) SMAC with three marines, sparse rewards

Figure 3: Mean training returns across seeds for sparse-reward variations of PP and SMAC-3m.

5.4 Results

Figures 3 to 5 show the training curves of SEAC, SNAC and IAC for all tested environments. For RWARE and LBF, tasks are sorted from easiest to hardest.

In the sparse PP task (Figure 3a) only SEAC learns successfully with consistent learning across seeds while IAC and SNAC are unable to learn to catch the prey.

In SMAC with sparse rewards (Figure 3b), SEAC outperforms both baselines. However, with mean returns close to zero, the agents have not learned to win the battles but rather to run away from the enemy. This is not surprising since our experiments (Table 1) show that even state-of-the-art methods designed for these environments (e.g. QMIX) do not successfully solve this sparsely rewarded task.

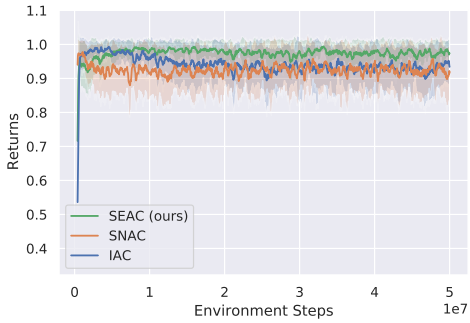
For LBF (Figure 4), no significant difference can be observed between SEAC and the two baseline methods IAC and SNAC for the easiest variant (Figure 4a) which does not emphasise exploration. However, as the rewards become sparser the improvement becomes apparent. For increased number of agents, foods and gridsize (Figures 4b to 4d), IAC and SNAC converge to significantly lower average returns than SEAC. In the largest grid (Figure 4c), IAC does not show any signs of learning due to the sparsity of the rewards whereas SEAC learns to collect some of the food. We also observe that SEAC tends to converge to its final policy in fewer timesteps than IAC.

In RWARE (Figure 5), results are similar to LBF. Again, the two baseline methods IAC and SNAC converge to lower average returns than SEAC for harder tasks. In the hardest task (Figure 5d), SEAC converges to final mean returns $\approx 70\%$ and $\approx 160\%$ higher than IAC and SNAC, respectively, and again converges in fewer steps than IAC.

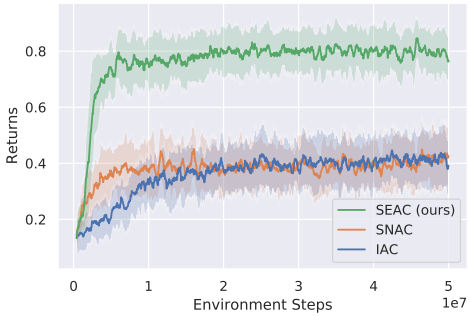
In Table 1 we also present the final evaluation returns of three state-of-the-art MARL algorithms (QMIX [29], MADDPG [21], and ROMA [37]) on a selection of tasks. These algorithms show no signs of learning in most of these tasks. The only exceptions are MADDPG matching the returns of SEAC in the sparse PP task and QMIX performing comparably to SEAC in the cooperative LBF task. QMIX and ROMA assume tasks to be fully-cooperative, i.e. all agents receive the same reward signal. Hence, in order to apply the two algorithms, we modified non-cooperative environments to return the sum of all individual agent returns as the shared reward. While shared rewards could make learning harder, we also tested IAC in the easiest variant of RWARE and found that it learned successfully even with this reward setting.

We also evaluate *Shared Experience Q-Learning*, as described in Appendix D, and Independent Q-Learning [34] based on DQN [24]. In some sparse reward tasks, shared experience did reduce variance, and improved total returns. However, less impact has been observed through the addition of sharing experience to this off-policy algorithm compared to SEAC. Results can be found in Appendix D.

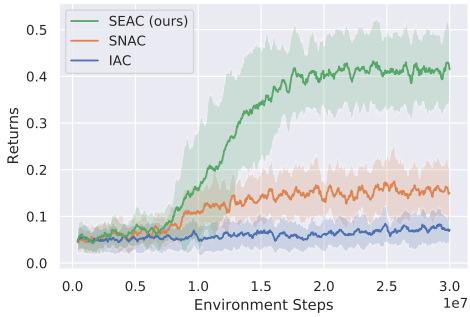
In terms of computational time, sharing experience with SEAC increased running time by less than 3% across all environments compared to IAC. More details can be found in Appendix B.



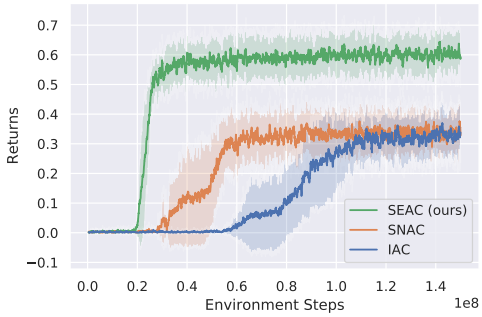
(a) LBF: (12 × 12), two agents, one food



(b) LBF: (10 × 10), three agents, three foods

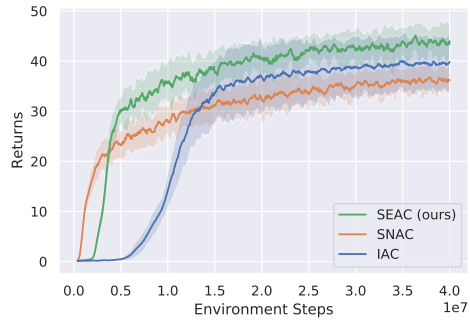


(c) LBF: (15 × 15), three agents, four food

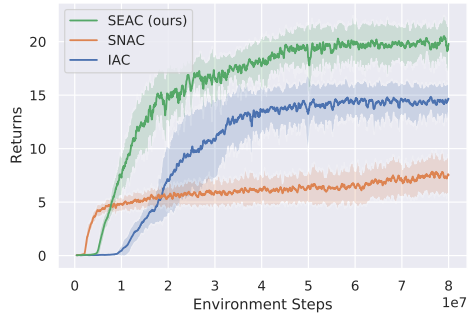


(d) LBF: (8 × 8), two agents, two foods, cooperative

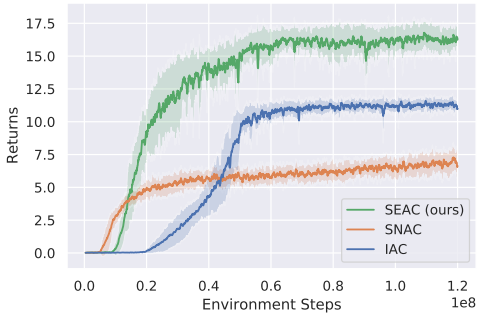
Figure 4: Mean training returns across seeds on LBF. Tasks are sorted from easiest to hardest.



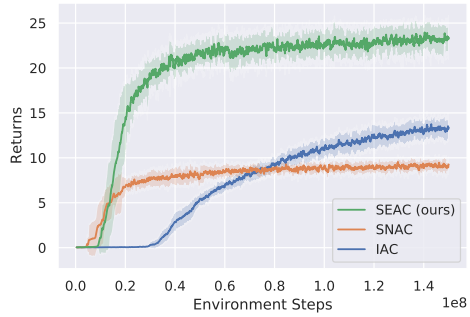
(a) RWARE: (10 × 11), four agents



(b) RWARE: (10 × 11), two agents



(c) RWARE: (10 × 11), two agents, hard



(d) RWARE: (10 × 20), four agents

Figure 5: Mean training returns across seeds on RWARE. Tasks are sorted from easiest to hardest.

Table 1: Final mean evaluation returns across five random seeds with standard deviation on a selection of tasks. Highest means per task (within one standard deviation) are shown in bold.

	IAC	SNAC	SEAC (ours)	QMIX	MADDPG	ROMA
PP (sparse)	-0.04 ±0.13	-0.04 ±0.1	**1.93 ±0.13**	0.05 ±0.07	**2.04 ±0.08**	0.04 ±0.07
SMAC-3m (sparse)	-0.13 ±0.01	-0.14 ±0.02	**-0.03 ±0.03**	**0.00 ±0.00**	**-0.01 ±0.01**	**0.00 ±0.00**
LBF-(15x15)-3ag-4f	0.13 ±0.04	0.18 ±0.08	**0.43 ±0.09**	0.03 ±0.01	0.01 ±0.02	0.03 ±0.02
LBF-(8x8)-2ag-2f-coop	0.37 ±0.10	0.38 ±0.10	**0.64 ±0.08**	**0.79 ±0.31**	0.01 ±0.02	0.01 ±0.02
RWARE-(10x20)-4ag	13.75 ±1.26	9.53 ±0.83	**23.96 ±1.92**	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
RWARE-(10x11)-4ag	**40.10 ±5.60**	36.79 ±2.36	**45.11 ±2.90**	0.00 ±0.00	0.00 ±0.00	0.01 ±0.01

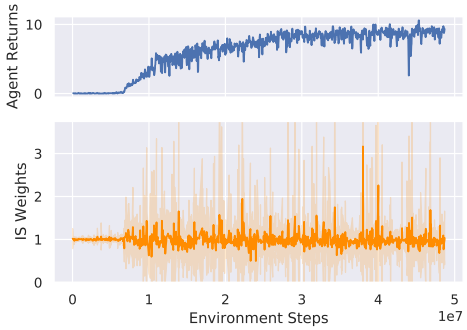


Figure 6: Importance weights of one SEAC agent in RWARE, (10×11), two agents, hard

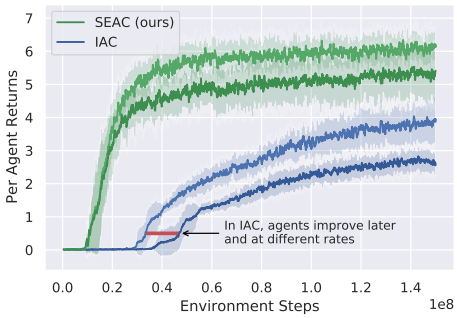


Figure 7: Best vs. Worst performing agents on RWARE, (10×20), four agents

5.5 Analysis

Similar patterns can be seen for the different algorithms across all tested environments. It is not surprising that IAC requires considerably more environment samples to converge, given that the algorithm is less efficient in using them; IAC agents only train on their own experience. This is further evident when noticing that in RWARE (Figs. 5a to 5d) the learning curve of SEAC starts moving upwards in roughly $1/N$ the timesteps compared to IAC, where N refers to the number of agents. Also, it is not surprising that SNAC does not achieve as high returns after convergence: sharing a single policy across all agents impedes their ability to coordinate or develop distinct behaviours that lead to higher returns.

We conjecture that SEAC converges to higher final returns due to agents improving at similar rates when sharing experiences, combined with the flexibility to develop differences in policies to improve coordination. We observe that SEAC is able to learn similarly quickly to SNAC because the combined local gradients provide a very strong learning direction. However, while SNAC levels off at some point due to the use of identical policies, which limit the agents' ability to coordinate, SEAC can continue to explore and improve because agents are able to develop different policies to further improve coordination. Figure 6 shows that encountered importance weights during SEAC optimisation are centred around 1, with most weights staying in the range $[0.5; 1.5]$. This indicates that the agents indeed learn similar but not identical policies. The divergence of the policies is attributed to the random initialisation of networks, along with the agent-centred entropy factor (Section 5.3). The range of the importance weights also shows that, in our case, importance sampling does not introduce significant instability in the training. The latter is essential for learning since importance weighting for off-policy RL is known to suffer from significant instability and high variance through diverging policies [33, 28].

In contrast, we observe that IAC starts to improve at a much later stage than SEAC because agents need to explore for longer, and when they start improving it is often the case that one agent improves first while the other agents catch up later, which can severely impede learning. Figure 7 shows that agents using IAC end up learning at different rates, and the slowest one ends up with the lowest final returns. In learning tasks that require coordination, an agent being ahead of others in its training can impede overall training performance.

We find examples of agents learning at different rates being harmful to overall training in all our tested environments. In RWARE, an agent that learns to fulfil requests can make the learning more difficult for others by delivering all requests on its own. Agents with slightly less successful exploration have a harder time learning a rewarding policy when the task they need to perform is constantly done by others. In LBF, agents can choose to cooperate to gather highly rewarding food or focus on food that can be foraged independently. The latter is happening increasingly often when an agent is ahead in the learning curve as others are still aimlessly wandering in the environment. In the PP task, the predators must approach the prey simultaneously, but this cannot be the case when one predator does not know how to do so. In the SMAC-3m task, a single agent cannot be successful if its team members do not contribute to the fight. The agent would incorrectly learn that fighting is not viable and therefore prefer to run from the enemy, which however is not an optimal strategy.

6 Conclusion

This paper introduced SEAC, a novel multi-agent actor-critic algorithm in which agents learn from the experience of others. In our experiments, SEAC outperformed independent learning, shared policy training, and state-of-the-art MARL algorithms in ten sparse-reward learning tasks, across four environments, demonstrating improved sample efficiency and final returns. We discussed a theme commonly found in MARL environments: agents learning at different rates impedes exploration, leading to sub-optimal policies. SEAC overcomes this issue by combining the local gradients and concurrently learning similar policies for all agents, but it also benefits from not being restricted to identical policies, allowing for better coordination and exploration.

Sharing experience is appealing especially due to its simplicity. We showed that barely any additional computational power, nor any extra parameter tuning are required and no additional networks are introduced. Therefore, its use should be considered in all environments that fit the requirements.

Future work could aim to relax the assumptions made for tasks SEAC can be applied to and evaluate in further multi-agent environments. Also, our work focused on the application of experience sharing to independent actor-critic. Further analysis of sharing experience as a generally applicable concept for MARL and its impact on a variety of MARL algorithms is left for future work.

Broader Impact

Multi-agent deep reinforcement learning has potential applications in areas such as autonomous vehicles, robotic warehouses, internet of things, smart grids, and more. Our research could be used to improve reinforcement learning models in such applications. However, it must be noted that real-world application of MARL algorithms is currently not viable due to open problems in AI explainability, robustness to failure cases, legal and ethical aspects, and other issues, which are outside the scope of our work.

That being said, improvements in MARL could lead to undue trust in RL models; having models that work well does not translate to models that are safe or which can be broadly used. Agents trained with these methods need to be thoroughly studied before being used in production. However, if these technologies are indeed used responsibly, they can improve several aspects of modern society such as making transportation safer, or performing jobs that might pose risks to humans.

Funding Disclosure

This research was in part financially supported by the UK EPSRC Centre for Doctoral Training in Robotics and Autonomous Systems (F.C.), the Edinburgh University Principal's Career Development Scholarship (L.S.), and personal grants from the Royal Society and the Alan Turing Institute (S.A.).

References

[1] Stefano V. Albrecht and Subramanian Ramamoorthy. "A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems". In: *International Conference on Autonomous Agents and Multi-Agent Systems*. Vol. 2. 2013, pp. 1155–1156.

[2] Stefano V. Albrecht and Subramanian Ramamoorthy. "Exploiting Causality for Selective Belief Filtering in Dynamic Bayesian Networks". In: *Journal of Artificial Intelligence Research* 55 (May 2016), pp. 1135–1178.

[3] Stefano V. Albrecht and Peter Stone. "Reasoning about Hypothetical Agent Behaviours and their Parameters". In: *International Conference on Autonomous Agents and Multi-Agent Systems*. 2017, pp. 547–555.

[4] Tim Brys, Anna Harutyunyan, Halit B. Suay, Sonia Chernova, Matthew E. Taylor, and Ann Nowé. "Reinforcement Learning from Demonstration through Shaping". In: *International Joint Conference on Artificial Intelligence*. 2015.

[5] Jeffery A. Clouse. "Learning from an Automated Training Agent". In: *Adaptation and Learning in Multiagent Systems*. Springer Verlag, 1996.

[6] Felipe L. Da Silva, Ruben Glatt, and Anna H. R. Costa. "Simultaneously Learning and Advising in Multiagent Reinforcement Learning". In: *International Conference on Autonomous Agents and Multi-Agent Systems*. 2017, pp. 1100–1108.

[7] Felipe L. Da Silva, Matthew E. Taylor, and Anna H. R. Costa. "Autonomously Reusing Knowledge in Multiagent Reinforcement Learning." In: *International Joint Conference on Artificial Intelligence*. 2018, pp. 5487–5493.

[8] Felipe L. Da Silva, Garrett Warnell, Anna H. R. Costa, and Peter Stone. "Agents Teaching Agents: A Survey on Inter-agent Transfer Learning". In: *Autonomous Agents and Multi-Agent Systems* 34.1 (2020).

[9] Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. "SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference". In: *International Conference on Learning Representations*. 2020.

[10] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Boron Yotam, Firoiu Vlad, Harley Tim, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *International Conference on Machine Learning*. Vol. 4. 2018, pp. 2263–2284.

[11] Anestis Fachantidis, Matthew E. Taylor, and Ioannis Vlahavas. "Learning to Teach Reinforcement Learning Agents". In: *Machine Learning and Knowledge Extraction* 1.1 (2019), pp. 21–42.

[12] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. "Counterfactual Multi-Agent Policy Gradients". In: *AAAI Conference on Artificial Intelligence, Innovative Applications of Artificial Intelligence, and AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI*. 2018, pp. 2974–2982.

[13] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. "Reinforcement Learning from Imperfect Demonstrations". In: *arXiv preprint arXiv:1802.05313* (2018).

[14] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. "Dynamic Programming for Partially Observable Stochastic Games". In: *AAAI*. Vol. 4. 2004, pp. 709–715.

[15] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. "Deep Reinforcement Learning and the Deadly Triad". In: *CoRR* abs/1812.0 (2018).

[16] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *Advances in Neural Information Processing Systems*. 2016, pp. 4565–4573.

[17] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio G. Castaneda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, et al. "Human-level Performance in 3D Multiplayer Games with Population-based Reinforcement Learning". In: *Science* 364.6443 (2019), pp. 859–865.

[18] Joel Z. Leibo, Julien Perolat, Edward Hughes, Steven Wheelwright, Adam H. Marblestone, Edgar Duéñez-Guzman, Peter Sunehag, Iain Dunning, and Thore Graepel. "Malthusian reinforcement learning". In: *International Joint Conference on Autonomous Agents and Multi-Agent Systems*. 2019, pp. 1099–1107.

[19] Michael L. Littman. "Markov Games as a Framework for Multi-Agent Reinforcement Learning". In: *Machine Learning Proceedings*. 1994, pp. 157–163.

[20] Qian Long, Zihan Zhou, Abhinav Gupta, Fei Fang, Yi Wu, and Xiaolong Wang. "Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning". In: *International Conference on Learning Representations*. 2020.

[21] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017, pp. 6382–6393.

[22] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. "MAVEN: Multi-Agent Variational Exploration". In: *Advances in Neural Information Processing Systems*. 2019, pp. 7611–7622.

[23] Volodymyr Mnih, Adria P. Badia, Lehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning". In: *International Conference on Machine Learning*. Vol. 4. 2016, pp. 2850–2869.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. "Human-level Control through Deep Reinforcement Learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[25] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. "Optimal and Approximate Q-Value Functions for Decentralized POMDPs". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 289–353.

[26] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2009), pp. 1345–1359.

[27] Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, and Stefano V. Albrecht. "Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning". In: *CoRR* abs/1906.0 (2019).

[28] Doina Precup. "Eligibility Traces for Off-policy Policy Evaluation". In: *Computer Science Department Faculty Publication Series* (2000), p. 80.

[29] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *International Conference on Machine Learning*. Vol. 10. 2018, pp. 6846–6859.

[30] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. "The StarCraft Multi-Agent Challenge". In: *International Joint Conference on Autonomous Agents and Multi-Agent Systems*. Vol. 4. 2019, pp. 2186–2188.

[31] Stefan Schaal. "Learning from Demonstration". In: *Advances in Neural Information Processing Systems*. 1997, pp. 1040–1046.

[32] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech M. Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, et al. "Value-Decomposition networks for cooperative multi-agent learning". In: *International Conference on Autonomous Agents and Multi-Agent Systems* (2018).

[33] Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning: An Introduction". In: *IEEE Transactions on Neural Networks*. Adaptive Computation and Machine Learning 9.5 (1998), pp. 1054–1054.

[34] Ming Tan. "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents". In: *Machine Learning Proceedings*. Elsevier, 1993, pp. 330–337.

[35] Matthew E. Taylor, Halit B. Suay, and Sonia Chernova. "Integrating Reinforcement Learning with Human Demonstrations of Varying Ability". In: *International Conference on Autonomous Agents and Multi-Agent Systems*. 2011, pp. 617–624.

[36] Jane X. Wang, Edward Hughes, Chrisantha Fernando, Wojciech M. Czarnecki, Edgar A. Duéñez-Guzmán, and Joel Z. Leibo. "Evolving Intrinsic Motivations for Altruistic Behavior". In: *International Conference on Autonomous Agents and Multi-Agent Systems*. 2019, pp. 683–692.

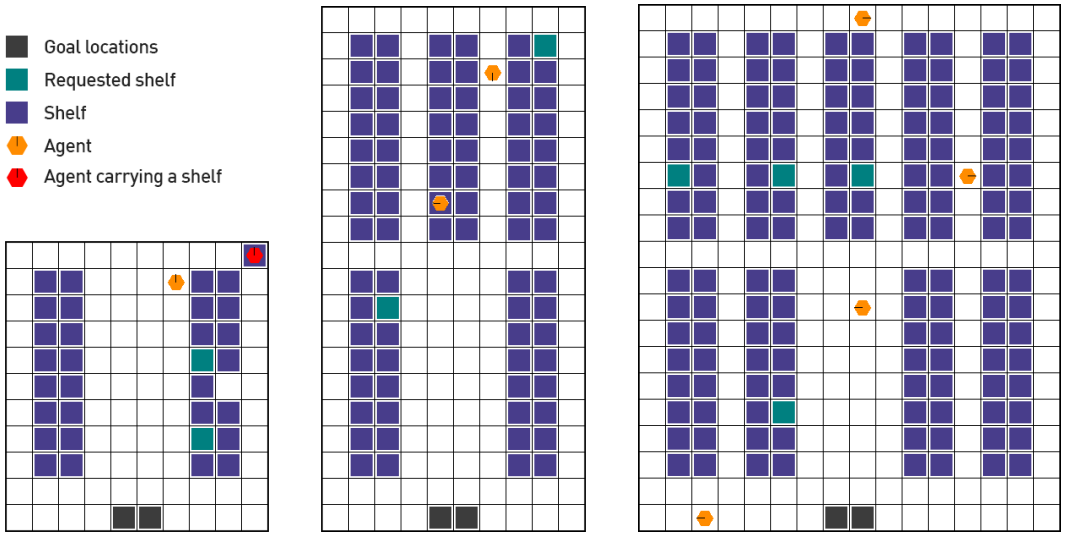
[37] Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. "ROMA: Multi-Agent Reinforcement Learning with Emergent Roles". In: *International Conference on Machine Learning*. 2020.

[38] Christopher J. C. H. Watkins and Peter Dayan. "Q-Learning". In: *Machine Learning* 8.3-4 (1992), pp. 279–292.

[39] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3-4 (1992), pp. 229–256.

[40] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses". In: *AI Magazine*. Vol. 29. 1. 2008, pp. 9–19.

[41] Matthieu Zimmer, Paolo Viappiani, and Paul Weng. "Teacher-student framework: a reinforcement learning approach". In: *AAMAS Workshop Autonomous Robots and Multirobot Systems*. 2014.



(a) Tiny size, two agents (b) Small size, two agents (c) Medium size, four agents

Figure 8: Three size variations of the multi-robot warehouse environment.

A Environments

A.1 Multi-Robot Warehouse

The multi-robot warehouse environment (Figure 8) simulates a warehouse with robots moving and delivering requested goods. In real-world applications [40], robots pick-up shelves and deliver them to a workstation. Humans assess the content of a shelf, and then robots can return them to empty shelf locations. In this simulation of the environment, agents control robots and the action space for each agent is

$$A = \{\text{Turn Left, Turn Right, Forward, Load/Unload Shelf}\}$$

Agents can move beneath shelves when they do not carry anything, but when carrying a shelf, agents must use the corridors visible in Figure 8.

The observation of an agent consists of a 3×3 square centred on the agent. It contains information about the surrounding agents (location/rotation) and shelves.

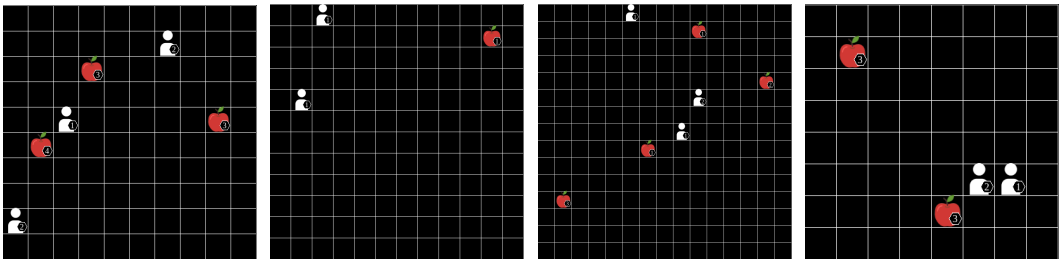
At each time a fixed number of shelves R is requested. When a requested shelf is brought to a goal location (dark squares in Fig. 8), another currently not requested shelf is uniformly sampled and added to the current requests. Agents are rewarded for successfully delivering a requested shelf to a goal location, with a reward of 1. A major challenge in this environments is for agents to deliver requested shelves but also afterwards finding an empty shelf location to return the previously delivered shelf. Agents need to put down their previously delivered shelf to be able to pick up a new shelf. This leads to a very sparse reward signal.

Since this is a collaborative task, as a performance metric we use the sum of the undiscounted returns of all the agents.

The multi-robot warehouse task is parameterised by:

- The size of the warehouse which is preset to either tiny (10×11), small (10×20), medium (16×20), or large (16×29).
- The number of agents N.
- The number of requested shelves R. By default $R = N$, but easy and hard variations of the environment use $R = 2N$ and $R = N/2$, respectively.

Note that R directly affects the difficulty of the environment. A small R, especially on a larger grid, dramatically affects the sparsity of the reward and thus exploration: randomly bringing the correct shelf becomes increasingly improbable.



(a) Foraging-10x10-3p-3f (b) Foraging-12x12-2p-1f (c) Foraging-15x15-3p-4f (d) Foraging-8x8-2p-2f-coop

Figure 9: Four variations of level based foraging used in this work.

A.2 Level-Based Foraging

The level-based foraging environment (Figure 9) represents a mixed cooperative-competitive game [1], which focuses on the coordination of the agents involved. Agents navigate a grid world and collect food by cooperating with other agents if needed.

More specifically, agents and food are randomly scattered in the grid world, and each is assigned a level. Agents can navigate in the environment and attempt to collect food placed next to them. The collection of food is successful only if the sum of the levels of all agents involved in collecting at the same time is equal to or higher than the level of the food. Agents are rewarded proportional to the level of food they took part in collecting. Episodes are terminated once all food has been collected or the maximum episode length of 25 timesteps is reached.

We are using full observability for this environment, meaning agents observe the locations and levels of all entities in the map. Each agent can attempt to move in all four directions and attempt to load adjacent food, for a total of five actions. After successfully loading a food, agents are rewarded:

$$r^i = \frac{FoodLevel * AgentLevel}{\sum FoodLevels \sum LoadingAgentsLevel}$$

This normalisation ensures that the sum of the agent returns on a solved episode equals to one.

Note that the final variant, Figure 9d, is a fully-cooperative environment. Food levels are always equal to the sum of all agents' levels, requiring all agents to load simultaneously, and thus sharing the reward.

B Additional Experimental Details

Our implementations of IAC, SEAC, and SNAC closely follow A2C [23], using n-step returns and parallel sampled environments. Table 2 contains the hyperparameters used in the experiments. Hyperparameters for MADDPG, QMIX and ROMA were optimised using a grid search over learning rate, exploration rate and batch sizes with the grid centred on the hyperparameters used in the original papers and parameter performance tested in all used environments.

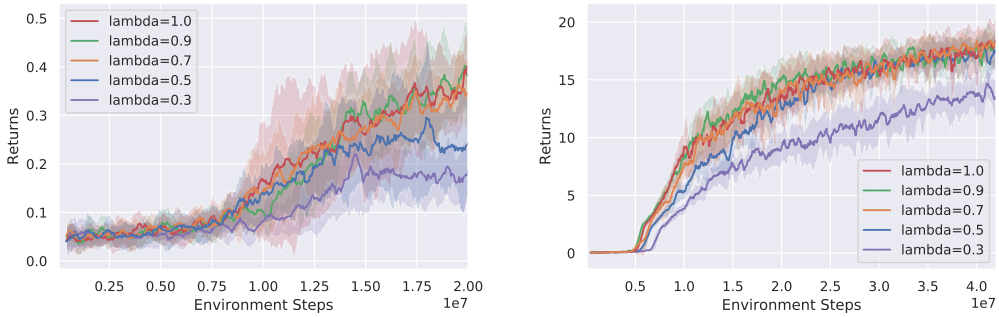
Table 2: Hyperparameters used for implementation of SEAC, IAC and SNAC

Hyperparameter	Value
learning rate	$3e^{-4}$
network size	64×64
adam epsilon	0.001
gamma	0.99
entropy coef	0.01
value loss coef	0.5
GAE	False
grad clip	0.5
parallel processes	4
n-steps	5
λ (Equations (4) and (5))	1.0

Table 3 contains process time required for running IAC and SEAC. Timings were measured on a 6^{th} Gen Intel i7 @ 4.6 Ghz running Python 3.7 and PyTorch 1.4. The average time for running and training on 100,000 environment iterations is displayed. Only process time (the time the program was active in the CPU) was measured, rounded to seconds. Often, the bottleneck is the environment and not the network update and as such, more complex and slower simulators, such as SMAC, show a lower percentage difference between algorithms.

Table 3: Measured mean process time (mins:secs) required for 100,000 timesteps.

	IAC	SEAC	% increase
Foraging-10x10-3p-3f-v0	2:00	2:04	3.86%
Foraging-12x12-2p-1f-v0	1:22	1:24	2.94%
Foraging-15x15-3p-4f-v0	2:01	2:06	3.90%
Foraging-8x8-2p-2f-coop-v0	1:21	1:24	3.78%
rware-tiny-2ag-v1	1:41	1:43	1.65%
rware-tiny-2ag-hard-v1	2:05	2:09	2.97%
rware-tiny-4ag-v1	2:49	2:53	2.25%
rware-small-4ag-v1	2:50	2:55	2.44%
Predator Prey	2:44	2:49	3.39%
SMAC (3m)	6:23	6:25	0.38%



(a) LBF (15×15), 3 agents, 4 foods

(b) RWARE (10×20), two agents

Figure 10: Training returns with different values of λ in SEAC

Figure 10 shows the training returns with respect to different λ values being applied in SEAC. We find that SEAC is not sensitive to tuning of the hyperparameter λ with similar performance across a wide range of values. Much lower values for λ closer to 0 lead to decreased performance, eventually converging to IAC for $\lambda = 0$.

For calculation of evaluation returns (Table 1), the best saved models per seed were selected and evaluated for 100 episodes. During evaluation, QMIX and ROMA use $\epsilon = 0$, while MADDPG and AC algorithms apply stochastic policies.

C SEAC Loss Derivation

We provide the following derivation of SEAC policy loss, as shown in Equation (4), for a fully observable two-agent Markov game

$$\mathcal{M} = \big(\mathcal{N} = \{1, 2\}, \mathcal{S}, (A^1, A^2), \mathcal{P}, (\mathcal{R}^1, \mathcal{R}^2)\big)$$

As per Section 3, let $\mathcal{A} = A^1 \times A^2$ be the joint action space and $A = A^1 = A^2$.

In the following, we use π_1 and π_2 to denote the policy of agent 1 and agent 2 which are conditioned on parameters ϕ_1 and ϕ_2, respectively. We use V^1 and V^2 to denote the state value function of agents 1 and 2 which are conditioned on parameters θ_1 and θ_2.

In order to account for different action distributions under policies π_1 and π_2, we use importance sampling (IS) defined for any function g over actions

$$\mathop{\mathbb{E}}_{a \sim \pi_1(a|s)} [g(a)] = \mathop{\mathbb{E}}_{a \sim \pi_2(a|s')} \left[\frac{\pi_1(a|s)}{\pi_2(a|s')} g(a) \right]$$

which can be derived as follows

$$\mathop{\mathbb{E}}_{a\sim\pi_1(a|s)}[g(a)] = \int_a \pi_1(a|s)g(a)da = \int_a \frac{\pi_2(a|s')}{\pi_2(a|s')}\pi_1(a|s)g(a)da = \mathop{\mathbb{E}}_{a\sim\pi_2(a|s')}\left[\frac{\pi_1(a|s)}{\pi_2(a|s')}g(a)\right]$$

Assumption 1 (Reward Independence Assumption: A1)**.** *We assume that an agent perceives the rewards as dependent only on its own action. Other agents are perceived as part of the environment.*

$$\forall s, s' \in \mathcal{S} : \forall a \in A : \hat{\mathcal{R}}^1(s, a, s') = \mathcal{R}^1(s, (a, \cdot), s')$$
$$\forall s, s' \in \mathcal{S} : \forall a \in A : \hat{\mathcal{R}}^2(s, a, s') = \mathcal{R}^2(s, (\cdot, a), s')$$

Assumption 2 (Symmetry Assumption: A2)**.** *We assume there exists a function $f : \mathcal{S} \mapsto \mathcal{S}$ such that*

$$\forall s, s' \in \mathcal{S} : \forall (a_1, a_2) \in \mathcal{A} : \mathcal{R}^1(f(s), (a_2, a_1), f(s')) = \mathcal{R}^2(s, (a_1, a_2), s')$$
$$\textit{and } \forall s, s' \in \mathcal{S} : \forall (a_1, a_2) \in \mathcal{A} : \mathcal{P}(s, (a_1, a_2))(s') = \mathcal{P}(f(s), (a_2, a_1))(f(s'))$$

Intuitively, given a state s, $f(s)$ swaps the agents: agent 1 is in place of agent 2 and vice versa.

Lemma 1 (Reward Symmetry: L1)**.** *From these two assumptions, it follows that for any states $s, s' \in \mathcal{S}$, and any action $a \in A$ the following holds:*

$$\hat{\mathcal{R}}^1(f(s), a, f(s')) = \hat{\mathcal{R}}^2(s, a, s')$$
$$\hat{\mathcal{R}}^2(f(s), a, f(s')) = \hat{\mathcal{R}}^1(s, a, s')$$

Proof.

$$\hat{\mathcal{R}}^1(f(s), a, f(s')) \overset{A1}{=} \mathcal{R}^1(f(s), (a, \cdot), f(s')) \overset{A2}{=} \mathcal{R}^2(s, (\cdot, a), s') \overset{A1}{=} \hat{\mathcal{R}}^2(s, a, s')$$
$$\hat{\mathcal{R}}^2(f(s), a, f(s')) \overset{A1}{=} \mathcal{R}^2(f(s), (\cdot, a), f(s')) \overset{A2}{=} \mathcal{R}^1(s, (a, \cdot), s') \overset{A1}{=} \hat{\mathcal{R}}^1(s, a, s')$$

□

During exploration, agent 1 and 2 follow policy π_1 and π_2, respectively. We will derive Equations (4) and (5) for training π_1 and V^1 using experience collected from agent 2. The derivation for optimisation of π_2 and V^2 using experience of agent 1 can be done analogously by substituting agent indices. Note that we only derive the off-policy terms of the SEAC policy and value loss. The on-policy terms of given losses are identical to A2C [23].

Agent 2 executes action a_2 in state s. Following Assumption 2, agent 1 needs to reinforce $\pi_1(a_2, f(s))$. Notably, in state $f(s)$, a_1 is sampled by π_2, so importance sampling is used to correct for this behavioural policy.

Proposition 1 (Actor Loss Gradient)**.**

$$\nabla_{\phi_1}\mathcal{L}(\phi_1) = \mathop{\mathbb{E}}_{a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}\left(\mathcal{R}^2(s, (\cdot, a_2), s') + \gamma V^1(f(s'))\right)\nabla_{\phi_1}\log\pi_1(a_2|f(s))\right]$$

Proof.

$$\begin{aligned}
\nabla_{\phi_1}\mathcal{L}(\phi_1) &= \mathop{\mathbb{E}}_{\substack{a_1\sim\pi_2\\ a_2\sim\pi_1}}\left[Q^1(f(s), a_2)\nabla_{\phi_1}\log\pi_1(a_2|f(s))\right] \\
&\overset{IS}{=} \mathop{\mathbb{E}}_{a_1,a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s))}Q^1(f(s), a_2)\nabla_{\phi_1}\log\pi_1(a_2|f(s))\right] \\
&= \mathop{\mathbb{E}}_{a_1,a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}\left(\mathcal{R}^1(f(s), (a_2, a_1), f(s')) + \gamma V^1(f(s'))\right)\nabla_{\phi_1}\log\pi_1(a_2|f(s))\right] \\
&\overset{A1}{=} \mathop{\mathbb{E}}_{a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}\left(\hat{\mathcal{R}}^1(f(s), a_2, f(s')) + \gamma V^1(f(s'))\right)\nabla_{\phi_1}\log\pi_1(a_2|f(s))\right] \\
&\overset{L1}{=} \mathop{\mathbb{E}}_{a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}\left(\hat{\mathcal{R}}^2(s, a_2, s') + \gamma V^1(f(s'))\right)\nabla_{\phi_1}\log\pi_1(a_2|f(s))\right] \\
&\overset{A1}{=} \mathop{\mathbb{E}}_{a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}\left(\mathcal{R}^2(s, (\cdot, a_2), s') + \gamma V^1(f(s'))\right)\nabla_{\phi_1}\log\pi_1(a_2|f(s))\right]
\end{aligned}$$

□

It should be noted that no gradient is back-propagated through the target $V^1(f(s')))$. In the same manner, the value loss, as shown in Equation (5), can be derived as follows.

Proposition 2 (Value Loss)**.**

$$\mathcal{L}(\theta_1) = \mathop{\mathbb{E}}_{a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}||V^1(f(s)) - \left(\mathcal{R}^2(s,(\cdot,a_2),s') + \gamma V^1(f(s'))\right)||^2\right]$$

Proof.

$$\begin{aligned}
\mathcal{L}(\theta_1) &= \mathop{\mathbb{E}}_{\substack{a_1\sim\pi_2\\ a_2\sim\pi_1}}\left[||V^1(f(s)) - \left(\mathcal{R}^1(f(s),(a_2,a_1),f(s')) + \gamma V^1(f(s'))\right)||^2\right] \\
&\overset{IS}{=} \mathop{\mathbb{E}}_{a_1,a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}||V^1(f(s)) - \left(\mathcal{R}^1(f(s),(a_2,a_1),f(s')) + \gamma V^1(f(s'))\right)||^2\right] \\
&\overset{A1}{=} \mathop{\mathbb{E}}_{a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}||V^1(f(s)) - \left(\hat{\mathcal{R}}^1(f(s),a_2,f(s')) + \gamma V^1(f(s'))\right)||^2\right] \\
&\overset{L1}{=} \mathop{\mathbb{E}}_{a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}||V^1(f(s)) - \left(\hat{\mathcal{R}}^2(s,a_2,s') + \gamma V^1(f(s'))\right)||^2\right] \\
&\overset{A1}{=} \mathop{\mathbb{E}}_{a_2\sim\pi_2}\left[\frac{\pi_1(a_2|f(s))}{\pi_2(a_2|s)}||V^1(f(s)) - \left(\mathcal{R}^2(s,(\cdot,a_2),s') + \gamma V^1(f(s'))\right)||^2\right]
\end{aligned}$$

□

D Shared Experience Q-Learning

D.1 Preliminaries and Algorithm Details

Deep Q-Networks: Deep Q-Networks (DQNs) [24] are used to replace the traditional Q-tables [38] by learning to estimate Q-values with a neural network. The algorithm uses an experience (replay) buffer D, which stores all experience tuples collected, circumventing the issue of time-correlated samples. Also, due to the instability created by bootstrapping, a second network with parameters $\bar{\theta}$ is used and updated by slowly copying the parameters of the network, θ, during training. The network is trained by minimising the loss

$$\mathcal{L}(\theta) = \frac{1}{M}\sum_{j=1}^{M}\left[(Q(s_j,a_j;\theta) - y_j)^2\right] \text{ with } y_j = r_j + \gamma\max_{a'}Q(s'_j,a';\bar{\theta}) \tag{6}$$

computed over a batch of M experience tuples (s,a,r,s') sampled from D.

During each update of agent i, previously collected experiences are sampled from the experience replay buffer D^i and used to compute and minimise the loss given in Equation (6). Independently

(a) RWARE (10×20), two agents (b) LBF: (8×8), two agents, two fruits, cooperative

Figure 11: Average total returns of SEQL and IQL for RWARE and LBF tasks

applying DQN for each agent in a MARL environment is referred to as *Independent Q-Learning* (IQL) [34]. For such off-policy methods, sharing experience can naturally be done by sampling experience from either replay buffer $o, a, r, o' \sim D^1 \cup \ldots \cup D^N$ and using the same loss for optimisation. We refer to this variation of IQL as *Shared Experience Q-Learning* (SEQL). In our experiments, we sample the same number of experience tuples $\frac{M}{N}$ from each replay buffer and the same sampled experience samples are used to optimise each agent. Hence, SEQL and IQL are optimised using exactly the same number of samples, in contrast to SEAC and IAC.

D.2 Results

Sharing experience in off-policy Q-Learning does improve performance, but does not show the same impact as for AC. We compare the performance of SEQL and IQL on one RWARE and LBF task to evaluate the impact of shared experience to off-policy MARL. Figure 11 shows the average total returns of SEQL and IQL on both tasks over five seeds. In the RWARE task, sharing experience appears to reduce variance considerably despite not impacting average returns significantly. On the other hand, on the LBF task average returns increased significantly by sharing experience and at its best evaluation even exceeded average returns achieved by SEAC. However, variance of off-policy SEQL and IQL is found to be significantly larger compared to on-policy SEAC, IAC and SNAC.