# Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning

# 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. We evaluate SEAC in a collection of sparse-reward multi-agent environments and find that it consistently outperforms two baselines and two 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 and the fact that the joint action space grows exponentially in the number of agents [18]. 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, which we show in our experiments can lead to significantly faster learning and higher final returns.

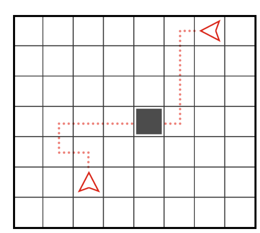


Figure 1: Two randomly-placed agents (triangles) must simultaneously arrive at the goal (square).

We demonstrate this idea in a novel actor-critic MARL algorithm, called *Shared Experience Actor-Critic* (SEAC). SEAC operates similarly to independent learning [25] but updates actor and critic parameters by combining gradients computed on the controlled agent's experience with weighted gradients computed on other agents' experiences. We evaluate SEAC in four sparse-reward multi-agent environments[1] and find that it learns substantially faster (up to 70% fewer required training steps) and achieves higher final returns compared to several baselines, including:

[1]We provide open-source implementations of two newly developed environments:
www.github.com/uoe-agents/lb-foraging, www.github.com/uoe-agents/robotic-warehouse

Preprint. Under review.

independent learning without experience sharing; using data from all agents to train a single shared policy; and MADDPG [12] and QMIX [20]. 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) [16, 20, 12] 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 [12] and COMA [8] learn powerful critic networks conditioned on joint observations and actions of all agents. A crucial difference to SEAC is that algorithms such as MADDPG and COMA 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 have been approaches to leverage expertise of *teacher* agents to address the issue of sample complexity in training a *learner* agent [5]. Such teaching can be regarded as a form of transfer learning [17] 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. However, even in such teaching approaches for multi-agent systems [4] 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 demonstration trajectories of other agents or humans is a common case [26, 22] of teaching agents. Demonstration data can be used to derive a policy which might be further refined using typical RL training [9] or to shape the rewards biasing towards previously seen expert demonstrations [3]. 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 [14] execute multiple actors in parallel to generate trajectories more efficiently and break data correlations. Similarly, IMPALA [7] and SEED RL [6] 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.

## 3 Technical Preliminaries

**Markov Games:** We consider partially observable Markov games for $N$ agents [11]. A Markov game is defined by the tuple $(\mathcal{N}, \mathcal{S}, \{O^i\}_{i\in\mathcal{N}}, \{A^i\}_{i\in\mathcal{N}}, \mathcal{P}, \{R^i\}_{i\in\mathcal{N}})$, with agents $i \in \mathcal{N} = \{1, \dots, N\}$, state space $\mathcal{S}$, 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 may depend deterministically or probabilistically on the current state. 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; $R^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function giving agent $i$'s individual reward $r^i$. Each agent $i$ seeks to maximise its discounted returns $G^i = \sum_{t=0}^{T} \gamma^t r_t^i$, with $\gamma$ and $T$ denoting the discount factor and total timesteps of an episode, respectively. $G_t^i$ denotes the returns for agent $i$ after timestep $t$.

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 [23, 20, 8, 13]. (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 $\pi_\phi$ parameterised by $\phi$, that maximises the expected returns. In REINFORCE [28], 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 $\theta$. 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 $\pi$ are parameterised by neural networks, sampling several trajectories in parallel, using $n$-step returns, regularisation, and other modifications can be beneficial [14]. 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 $\beta$ 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:

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

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

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

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 $\lambda$ weights the experience of other agents; we found SEAC to be largely insensitive to values of $\lambda$ 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

**Algorithm 1** Shared Experience Actor-Critic Framework

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

as annealing $\epsilon$-greedy policies to different values of $\epsilon$, have been observed [14] 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 [15]. 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 [10] compared to on-policy AC. Our preliminary results in Appendix D indicate that experience sharing does not lead to similar improvements in DQN compared to SEAC, but is able to reduce variance in more challenging tasks.

## 5 Experiments

We conduct experiments on four sparse-reward multi-agent environments and compare SEAC to two baselines as well as two state-of-the-art MARL algorithms, MADDPG [12] and QMIX [20].

### 5.1 Environments

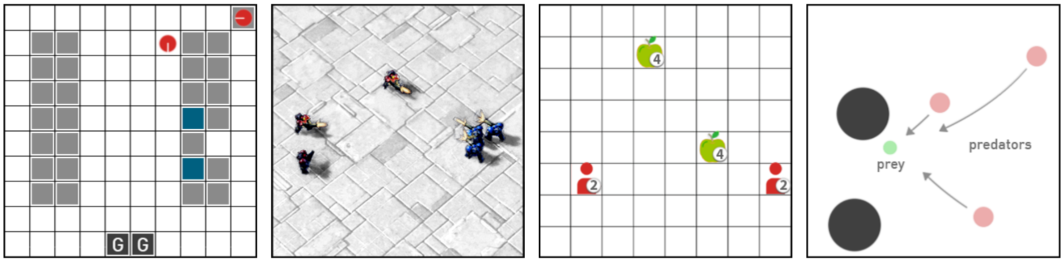
The following multi-agent environments were used in our evaluation. More detailed descriptions of these environments can be found in Appendix A.

**Multi-Robot Warehouse (RWARE), Fig. 2a:** This multi-agent environment simulates robots that move goods around a warehouse, similarly to existing real-world applications [29]. 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.

**Starcraft Multi-Agent Challenge (SMAC), Fig. 2b:** The SMAC [21] environment was used in several recent MARL works [20, 8]. 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 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 a victory rewards 1, while a defeat $-1$.

**Level-Based Foraging (LBF), Fig. 2c:** LBF [1, 2] 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.

**Predator Prey (PP), Fig. 2d:** Finally, we use the popular PP environment adapted from the Multi-agent Particle Environment framework [12]. 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.




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

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

### 5.2 Baselines

**Independent Actor-Critic (IAC):** We compare SEAC to independent learning [25], 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, optimising directly 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 policy and value loss gradients are summed, and 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 [14]. 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 [14]. 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.

### 5.4 Results

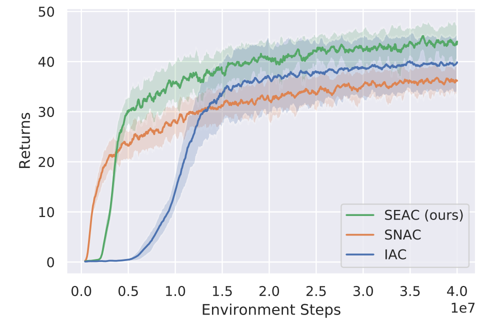
Figures 3 and 4 show the training curves of SEAC, SNAC and IAC for all tested environments. For RWARE and LBF, tasks are sorted from easiest to hardest.

In the RWARE (Figures 3a to 3d), the two baseline methods IAC and SNAC converge to significantly lower average returns than SEAC as difficulty increases. In the hardest task (Figure 3d), SEAC converges to final mean returns $\approx 70\%$ and $\approx 160\%$ higher than IAC and SNAC, respectively.

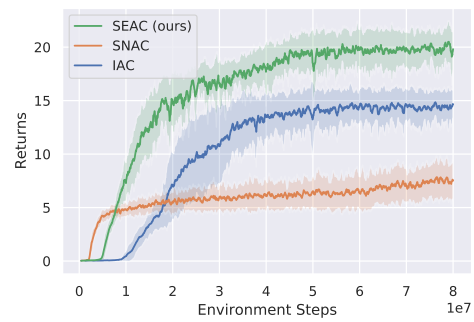
For LBF (Figures 3e to 3h), results are similar to RWARE. Again, easier variants which do not emphasise exploration show no significant differences, but as the rewards become sparser the improvement is starting to show. In the larger grid (Figure 3g) IAC does not show any signs of learning due to the sparsity of the rewards whereas SEAC shows significant learning.

In SMAC with sparse rewards (Figure 4a) 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.

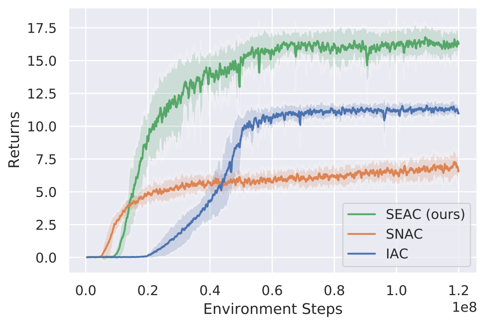
Finally, in the sparse PP task (Figure 4b) only SEAC learns successfully, with consistent learning across seeds.



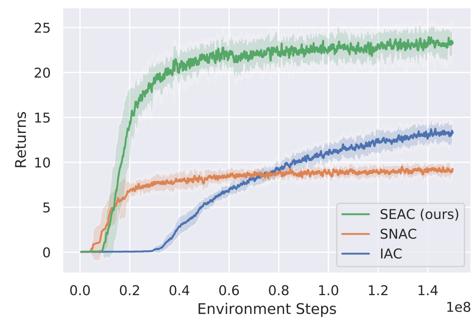

(a) RWARE: (10 × 11), four agents



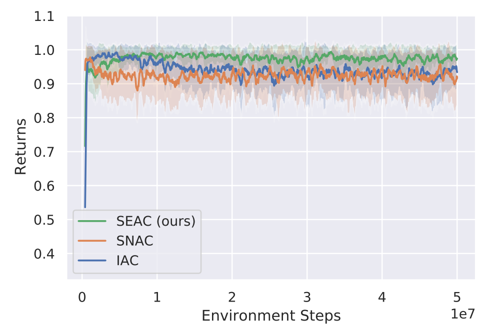

(b) RWARE: (10 × 11), two agents



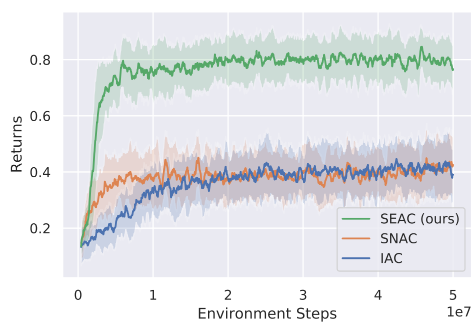

(c) RWARE: (10 × 11), two agents, hard



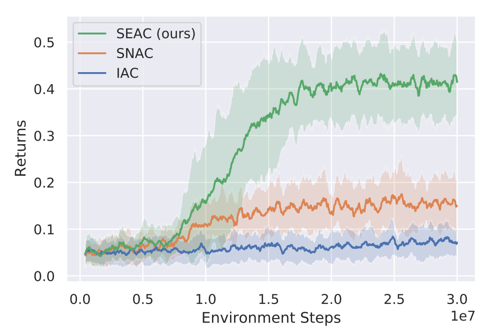

(d) RWARE: (10 × 20), four agents



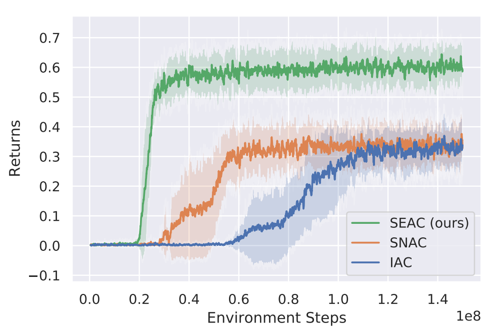

(e) LBF: (12 × 12), two agents, one food




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

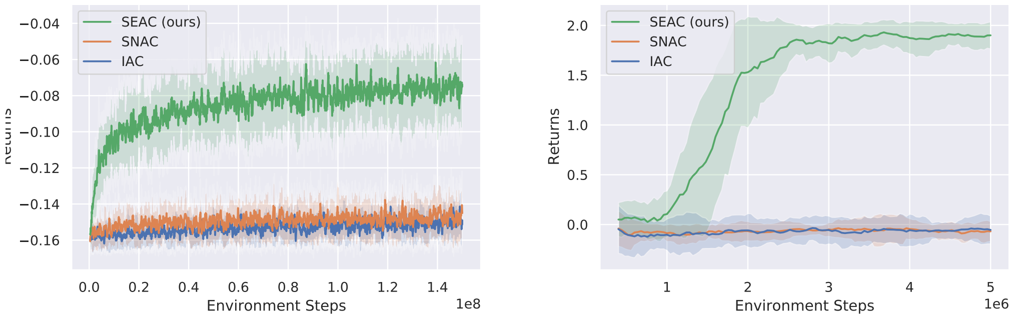



(g) LBF: (15 × 15), three agents, four food




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

Figure 3: Mean training returns across seeds on RWARE and LBF. Tasks in Figures 3a to 3d and Figures 3e to 3h are sorted from easiest to hardest.




(a) SMAC with three marines, sparse rewards

(b) PP, sparse rewards

Figure 4: Mean training returns across seeds for sparse reward variations of SMAC-3m and PP.

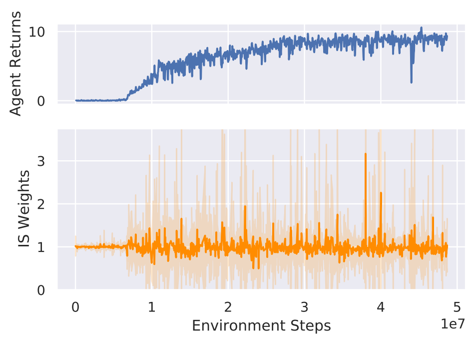



Figure 5: Importance weights of one SEAC agent in RWARE, $(10 \times 11)$, two agents, hard

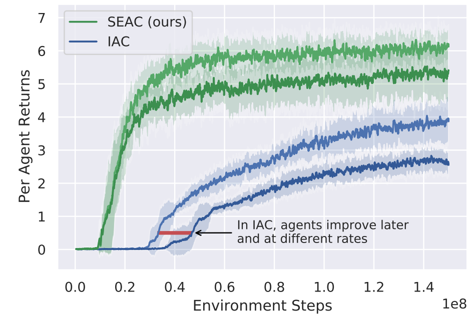



Figure 6: Best vs. Worst performing agents on RWARE, $(10 \times 20)$, four agents

We also evaluate *Shared Experience Q-Learning*, as described in Appendix D, and *Independent Q-Learning* based on DQN. In some sparse reward tasks, shared experience did reduce variance, but overall 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 Table 1 we also present the final returns of two additional state-of-the-art methods (QMIX [20] and MADDPG [12]) in deep MARL, on a selection of environments. These methods show no signs of learning in most of these tasks. QMIX assumes the task to be fully-cooperative, i.e. all agents receive the same reward signal. Hence, in order to apply QMIX, 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.

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

### 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. 3a to 3d) the learning curve starts moving upwards in roughly $1/N$ the timesteps of 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

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 bold.

| | IAC | SNAC | SEAC (ours) | QMIX | MADDPG |
|---|---|---|---|---|---|
| SMAC-3m (sparse) | -0.13 ±0.01 | -0.14 ±0.02 | **-0.03 ±0.03** | **0.00 ±0.00** | **-0.01 ±0.01** |
| RWARE-(10x20)-4ag | 13.75 ±1.26 | 9.53 ±0.83 | **23.96 ±1.92** | 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 |
| LBF-(15x15)-3ag-4f | 0.13 ±0.04 | 0.18 ±0.08 | **0.43 ±0.09** | 0.03 ±0.01 | 0.01 ±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 |

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 allowed to develop differences in their policies to further improve coordination. Figure 5 shows that encountered importance weights during SEAC optimisation are centred around one, with most weights staying in the range $[0.5; 1.5]$. This indicates that the agents indeed learn similar but not identical policies. It 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 [24, 19].

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 6 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 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 environment, the predators must approach the prey simultaneously, but this cannot be the case when one predator does not know how to. 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, but it also benefits from not having 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 is 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 in this work and evaluate in additional 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.

## 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 Peter Stone. "Reasoning about Hypothetical Agent Behaviours and their Parameters". In: *International Conference on Autonomous Agents and Multi-Agent Systems*. São Paulo, Brazil, 2017.

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

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

[5] Felipe Leno Da Silva, Garrett Warnell, Anna Helena Reali Costa, and Peter Stone. "Agents Teaching Agents: A Survey on Inter-agent Transfer Learning". In: *Autonomous Agents and Multi-Agent Systems* 34.1 (2020), p. 9.

[6] 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.

[7] 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.

[8] 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.

[9] 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).

[10] 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).

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

[12] 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*, pp. 6382–6393.

[13] 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.

[14] Volodymyr Mnih, Adria Puigdomenech 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.

[15] 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.

[16] 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 (May 2008), pp. 289–353.

[17] 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.

[18] 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).

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

[20] 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". In: *International Conference on Machine Learning*. Vol. 10. 2018, pp. 6846–6859.

[21] 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.

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

[23] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian 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: *arXiv preprint arXiv:1706.05296* (2017).

[24] R.S. Sutton and A.G. Barto. "Reinforcement Learning: An Introduction". In: *IEEE Transactions on Neural Networks*. Adaptive computation and machine learning 9.5 (1998), pp. 1054–1054.

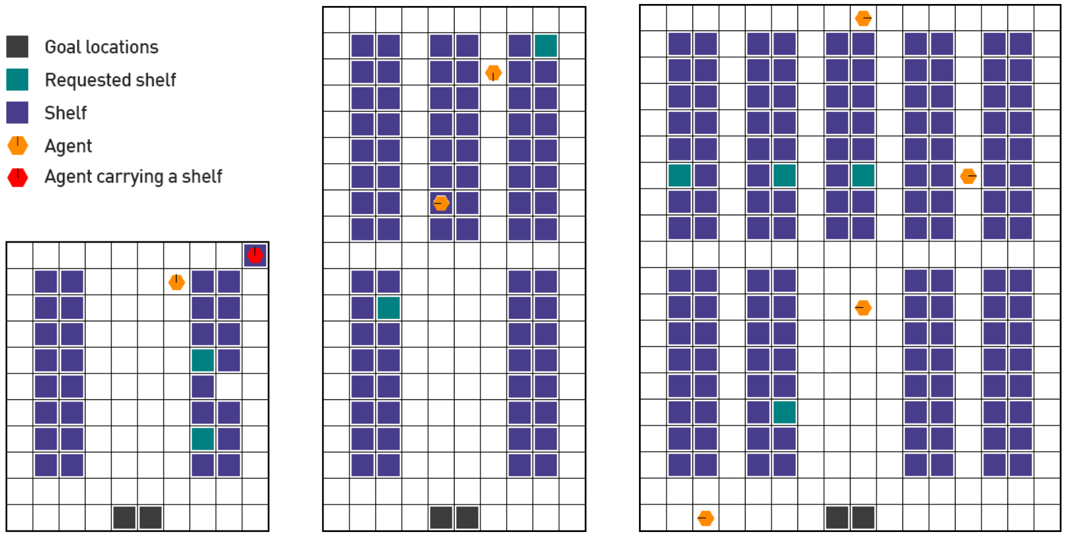
[25] Ming Tan. "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents". In: *Machine Learning Proceedings*. Elsevier, 1993, pp. 330–337.

[26] Matthew E Taylor, Halit Bener 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.

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

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

[29] 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.




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

Figure 7: Three size variations of the multi-robot warehouse environment.

# A Environments

## A.1 Multi-Robot Warehouse

The multi-robot warehouse environment (Figure 7) simulates a warehouse with robots moving and delivering requested goods. In real-world applications [29], 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 7.

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

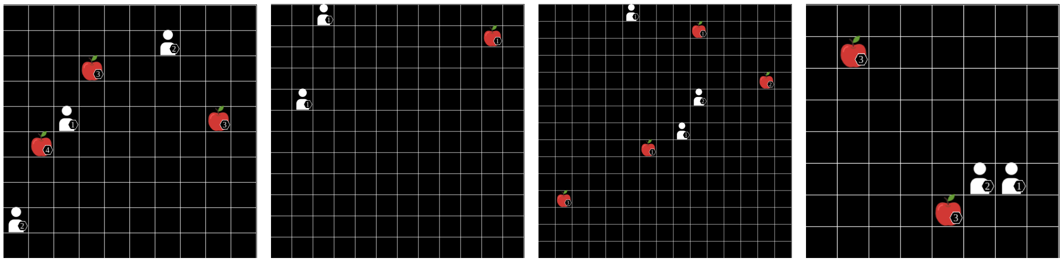
At each time a set number of shelves $R$ is requested. When a requested shelf is brought to a goal location (dark squares in Fig. 7), another shelf is uniformly sampled and added to the current requests. Agents are rewarded for successfully delivering a requested shelf to a goal location, with a reward 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. This leads 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 ($10 \times 11$), small ($10 \times 20$), medium ($16 \times 20$), or large ($16 \times 29$).
- The number of agents.
- 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 8: Four variations of level based foraging used in this work.

### A.2 Level-Based Foraging

The level-based foraging environment (Figure 8) 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 8d, 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 [14], using n-step returns and parallel sampled environments. Table 2 contains the hyperparameters used in the experiments.

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, slower simulators (SMAC) show a lower percentage difference between algorithms.

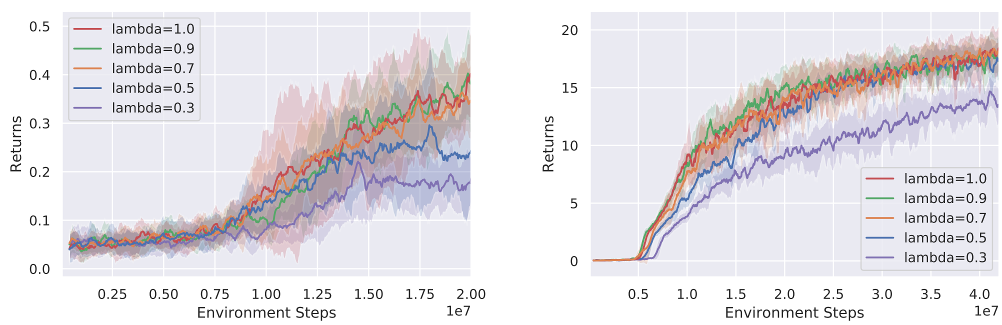
Table 2: Hyperparameters used for implementation of SEAC, IAC and SNAC

| Hyperparameter | Value |
|---|---|
| learning rate | $3e^{-4}$ |
| network size | $64 \times 64$ |
| 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 |
| $\lambda$ (Equations (4) and (5)) | 1.0 |

Figure 9 shows the training returns with different $\lambda$ values. The hyperparameter $\lambda$ is not sensitive to tuning. Much lower values lead to decreased performance.

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 9: Training returns with different values of $\lambda$ in SEAC

For calculation of evaluation returns (Table 1), the best saved models per seed were selected and evaluated for 100 episodes. During evaluation, Q-MIX used $\epsilon = 0$, while MADDPG and AC algorithms used the 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

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

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 $\pi_1$ and $\pi_2$ to denote the policy of agent 1 and agent 2 which are conditioned on parameters $\phi_1$ and $\phi_2$, respectively. We use $V^1$ and $V^2$ to denote the state value function of agents 1 and 2 which are conditioned on parameters $\theta_1$ and $\theta_2$.

In order to account for different action distributions under policies $\pi_1$ and $\pi_2$, 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.*

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

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

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

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:*

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

*Proof.*

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

□

During exploration, agent 1 and 2 follow policy $\pi_1$ and $\pi_2$ respectively. We will derive Equations (4) and (5) for training $\pi_1$ and $V^1$ using experience collected from agent 2. The derivation for optimisation of $\pi_2$ 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 [14].

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 $\pi_2$, so importance sampling is used to correct for this behavioural policy.

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

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

*Proof.*

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

□

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)**.**

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

*Proof.*

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

□

# D Shared Experience Q-Learning

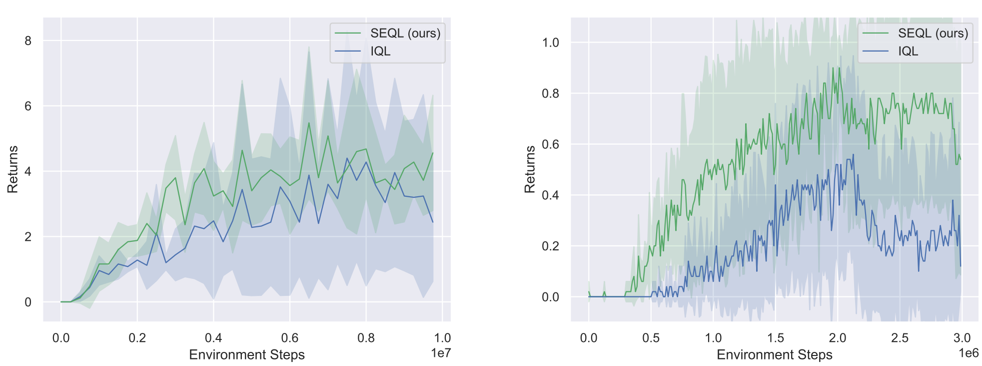
## D.1 Preliminaries and Algorithm Details

**Deep Q-Networks:** A Deep Q-Network (DQN) [15] is used to replace the traditional Q-table [27] by learning to estimate Q-values. 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, $\theta$ 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 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 10: 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) [25]. 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 10 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.