Comparative Evaluation of Cooperative Multi-Agent Deep Reinforcement Learning Algorithms

Georgios Papoudakis *
University of Edinburgh
g.papoudakis@ed.ac.uk

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

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

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

ABSTRACT

Multi-agent deep reinforcement learning (MARL) suffers from a lack of commonly-used evaluation tasks and criteria, making comparisons between approaches difficult. In this work, we evaluate and compare three different classes of MARL algorithms (independent learning, centralised multi-agent policy gradient, and value decomposition) in a diverse range of fully-cooperative multi-agent learning tasks. Our experiments can serve as a reference for the expected performance of algorithms across different learning tasks. We also provide further insight about (1) when independent learning might be surprisingly effective despite non-stationarity, (2) when centralised training should (and shouldn't) be applied and (3) which benefits value decomposition can bring.

KEYWORDS

Multi-Agent Reinforcement Learning, Deep Reinforcement Learning, Multi-Agent Systems, Cooperation

1 INTRODUCTION

Multi-agent reinforcement learning (MARL) addresses learning tasks consisting of multiple, concurrently learning agents. Recent years have seen a plethora of new MARL algorithms which integrate deep learning techniques [18]. However, comparison of MARL algorithms is difficult due to a lack of established benchmark tasks, evaluation protocols, and metrics. While several comparative studies exist for single-agent RL [12, 17, 45], we are unaware of such comparative studies for recent MARL algorithms. Albrecht and Ramamoorthy [2] compare several MARL algorithms but focus on the application of classic (non-deep) approaches in simple matrix games. Such comparisons are crucial in order to understand the relative strengths and limitations of algorithms, which may guide practical considerations and future research.

We contribute a comprehensive empirical comparison of seven MARL algorithms in a diverse set of tasks. We compare three classes of MARL algorithms: **independent learning**, which applies single-agent RL algorithms for each agent without consideration of the multi-agent structure [43]; **centralised multi-agent policy gradient** [15, 24]; and algorithms based on **value decomposition** [34, 41]. The two latter classes of algorithms follow the Centralised Training Decentralised Execution (CTDE) paradigm. These algorithm classes are frequently used in the literature either as baselines or building blocks for more complex algorithms [11, 14, 16, 19, 21, 33, 35, 40]. Algorithms are compared across two common-payoff matrix games and four multi-agent environments with a total of 23 different fully-cooperative learning tasks. We report average training and final returns, as well as several recently-proposed metrics to measure robustness and variability [6]. These results can serve as a reference for the expected performance of all evaluated algorithms across all 23 considered tasks. Additionally, we discuss under which conditions each class of algorithms can be applied effectively. Our main findings can be summarised as follows:

(1) Independent learning approaches are surprisingly effective in fully-observable environments and due to their simplicity serve as an effective baseline for MARL.
(2) Centralised training can assist coordination and help to overcome partial observability, but does not always improve returns achieved by the algorithm. In tasks with high-dimensional observations or significant sparsity of rewards, training large centralised critics can significantly complicate the training procedure.
(3) Decomposition of value functions is a powerful paradigm for multi-agent tasks, in particular if heterogeneous agents with different capabilities are trained.
(4) None of the algorithms consistently solve all matrix games optimally despite the simplicity of these tasks.

2 MATHEMATICAL FRAMEWORK

In this work, we consider the Dec-POMDPs [5, 31] framework to model cooperative multi-agent environments. A Dec-POMDP is defined by $(\mathcal{I}, \mathcal{S}, \mathcal{O}, \mathcal{A}, P, \Omega, R)$, where $\mathcal{I} = \{1, ...N\}$ is the set of agents and $\mathcal{S}$ is the set of states in the environments. We assume a partially observable setting, in which each agent $i \in \mathcal{I}$ has only access to a local observation $o_t^i \in \mathcal{O}$ from the set of observations $\mathcal{O}$ at each time step t. Furthermore, a Dec-POMDP consists of the set of joint actions $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2 \times ...\mathcal{A}^N$, where $\mathcal{A}^i$ is the set of actions of agent i, the state transition function $P(s'|s, \boldsymbol{a})$, which is the probability distribution over all possible next states given the current state and joint action $\boldsymbol{a}$, and the observation transition function $\Omega(\boldsymbol{o}|s, \boldsymbol{a})$, which is the distribution over all possible next joint observations, given the current state and joint action. The reward function is defined as $R(s, \boldsymbol{a}, s')$, and returns a single global reward shared among all agents. In some environments, the full state is

* Equal contribution

Table 1: Overview of algorithms and their properties.

	Centr. Training	Off-/On-policy	Value-based	Policy-based
IQL	✗	Off	✓	✗
IA2C	✗	On	✓	✓
MADDPG	✓	Off	✓	✓
COMA	✓	On	✓	✓
Central-V	✓	On	✓	✓
VDN	✓	Off	✓	✗
QMIX	✓	Off	✓	✗

unavailable. In these cases, we approximate the state of the environment using the joint observation of all agents $s \approx \{o^1, ..., o^N\}$.

3 ALGORITHMS

In this framework, we consider seven algorithms from the family of independent learning (IL) and centralised training decentralised execution (CTDE) paradigms. We split latter into centralised multi-agent policy gradient and methods for value decomposition. Table 1 lists all algorithms and their properties. In the following, policies and the value functions are conditioned on the history of agent observations. The history of all agents in the environment is denoted as $\mathbf{h}$. The history of agent i and the history of all agents except agent i are given by h^i and $\mathbf{h^{-i}}$, respectively. Similar notation applies to the actions a and observations of the agents.

3.1 Independent Learning

For IL, each agent is learning independently and perceives the other agents as part of the environment.

IQL: In Independent Q-Learning (IQL) [43], each agent has a decentralised state-action value function that is conditioned only on the local history of observations and actions of each agent. Each agent receives its local history of observations and updates the parameters of the Q-value network [28] by minimising the standard Q-learning loss [46].

$$\min_{w^i} \mathbb{E}_B \left[\frac{1}{2} \left(r^i + \gamma \max_{a'^i} Q_{w'^i}(h'^i, a'^i) - Q_{w^i}(h^i, a^i) \right)^2 \right] \quad (1)$$

IA2C: Independent synchronous Advantage Actor-Critic (IA2C) is a variant of the commonly-used A2C algorithm [10, 27] for decentralised training in multi-agent systems. Each agent has its own actor to approximate the policy and critic network to approximate the value-function. Both actor and critic are trained, conditioned on the history of local observations, actions and rewards the agent perceives, to minimise the A2C loss.

$$\min_{\theta^i, w^i} \mathbb{E}_B \left[\frac{1}{2} \left(r^i + \gamma V_{w'^i}(h'^i) - V_{w^i}(h^i) \right)^2 - \hat{A} \log \pi_{\theta^i}(a^i|h^i) \right] \quad (2)$$

where $\hat{A}$ is the advantage and can be computed using the *Generalised Advantage Estimation* (GAE) [37].

3.2 Centralised Training Decentralised Execution

In contrast to IL, CTDE allows sharing of information during the training phase, while policies are only conditioned on the agents' local observations enabling decentralised execution.

3.2.1 Centralised Multi-Agent Policy Gradient. One category of CTDE algorithms are centralised policy gradient methods in which each agent consists of a decentralised actor and a centralised critic, which is optimised based on shared information between the agents.

MADDPG: Multi-Agent DDPG (MADDPG) [24] is a variation of the DDPG algorithm [23] for MARL. All actors are only conditioned on the history of local observations to approximate the policy, while critics are trained on the joint observation and action of all agents to approximate the joint state-action value function. Each agent individually minimises the deterministic policy gradient loss [39].

$$\min_{w^i} \mathbb{E}_B \left[\frac{1}{2} \left(r^i + \gamma Q_{w'^i}(\mathbf{h'}, \mathbf{a'}) - Q_{w^i}(\mathbf{h}, \mathbf{a}) \right)^2 \right] \quad (3)$$

$$\min_{\theta^i} \mathbb{E}_B \left[-Q_{w^i} \left(\mathbf{h}, \langle \mathbf{a^{-i}}, \mu_{\theta^i}(h^i) \rangle \right) \right] \quad (4)$$

with $\mathbf{a'} = \langle \mu_{\theta'^1}(h'^1), ..., \mu_{\theta'^N}(h'^N) \rangle$. A common assumption of DDPG (and thus MADDPG) is differentiability of actions with respect to the parameters of the actor, so the action space must be continuous. Lowe et al. [24] apply the Gumbel-Softmax trick [20, 26] to learn in discrete action spaces.

COMA: In Counterfactual Multi-Agent (COMA) Policy Gradient, Foerster et al. [15] propose a modification of the advantage in the actor's loss computation to perform counterfactual reasoning for credit assignment in cooperative MARL.

$$\hat{A}^i = Q_{w^i}(\mathbf{h}, \boldsymbol{a}) - \sum_{\hat{a}^i} \pi_{\theta^i}(\hat{a}^i|h^i) Q_{w^i} \left(\mathbf{h}, \langle \mathbf{a^{-i}}, \hat{a}^i \rangle \right) \quad (5)$$

The advantage is defined as the discrepancy between the state-action value of the followed joint action and a counterfactual baseline. The latter is given by the expected value of the specific agent i following its current policy while the actions of other agents are fixed. The standard policy loss with this modified advantage is used to train the actor and the critic is trained using the TD-lambda algorithm [42]. Due to the architectural design of COMA, the approach is limited to fully-cooperative environments as it assumes the same reward signal for all agents.

Central-V: Central-V is an actor-critic algorithm in which the critic learns a joint state value function (in contrast, the critics in MADDPG and COMA are also conditioned on actions). It extends existing on-policy actor-critic algorithms, such as A2C [10, 27] or PPO [38], by applying centralised critics conditioned on the state of the environment rather than the individual history of observations. In this work, we use A2C for Central-V and the standard A2C loss is minimised during optimisation.

3.2.2 Value Decomposition. Another recent CTDE research direction is the decomposition of the joint state-action value function into individual state-action value functions of each agent.

VDN: Value Decomposition Networks (VDN) [41] aim to learn a linear decomposition of the joint Q-value. Each agent maintains a deep network to approximate its own state-action values. VDN decomposes the joint Q-value into the sum of individual Q-values.

$$Q_{tot}(\boldsymbol{h}, \boldsymbol{a}) \approx \sum_i Q_{w^i}(h^i, a^i) \tag{6}$$

The joint state-action value function is trained using the standard DQN algorithm [28, 46]. During training, gradients of the joint TD loss flow backwards to the network of each agent. Similar to COMA, VDN also assumes a shared reward signal among all agents and is, therefore, by design limited to fully-cooperative environments.

QMIX: QMIX [34] extends VDN to address a broader class of environments. To represent a more complex decomposition, a parameterised mixing network is introduced to compute the joint Q-value based on each agent's individual state-action value function.

$$Q_{tot}(\boldsymbol{h}, \boldsymbol{a}) = f_v(Q_{w^1}(h^1, a^1), ..., Q_{w^N}(h^N, a^N)) \tag{7}$$

A requirement of the mixing function is that the optimal joint action, which maximises the joint Q-value, is the same as the combination of the individual actions maximising the Q-values of each agent. This assumption is fulfilled by constraining the joint value function to be monotonically increasing with respect to each individual value function ($\frac{\partial Q_{\text{tot}}}{\partial Q^i} > 0$). QMIX is trained to minimise the DQN loss and the gradient is backpropagated to the individual Q-values, similarly to VDN.

4 MULTI-AGENT ENVIRONMENTS

We evaluate the algorithms in two finitely repeated matrix games [4] and four multi-agent environments within which we define a total of 23 different learning tasks. All tasks are fully-cooperative, i.e. all agents receive identical reward signals. These tasks range over various properties including the degree of observability (whether agents can see the full environment state or only parts of it), reward density (receiving frequent/dense vs infrequent/sparse non-zero rewards), and the number of agents involved. Figure 1 presents a task instance of each environment. Table 2 lists environments with properties, and we give brief descriptions below. We believe each of the following environments addresses a specific challenge of MARL. Full details of environments and learning tasks are provided in Appendix A.

Repeated Matrix Games: We consider two cooperative matrix games proposed by Claus and Boutilier [9]: the *climbing* and *penalty* game. The common-payoff matrices of the climbing and penalty game, respectively, are:

$$\begin{bmatrix} 0 & 6 & 5 \\ -30 & 7 & 0 \\ 11 & -30 & 0 \end{bmatrix} \qquad \begin{bmatrix} k & 0 & 10 \\ 0 & 2 & 0 \\ 10 & 0 & k \end{bmatrix}$$

where $k \leq 0$ is a penalty term. We evaluate in the penalty game for $k \in \{-100, -75, -50, -25, 0\}$. The difficulty of this game strongly correlates with k: the smaller k, the harder it becomes to identify the optimal policy due to the growing risk of penalty k. Both games are applied as repeated matrix games with an episode length of 25 and agents are given constant observations at each timestep. These matrix games are challenging due to the existence of local minima in the form of sub-optimal Nash equilibria [30]. Slight deviations from optimal policies by one of the agents can result in significant penalties, so agents might get stuck in risk-free (deviations from any agent does not significantly impede payoff) local optima. Therefore, these matrix games can be considered a benchmark for stability of convergence.

Table 2: Overview of environments and properties.

	Observability	Rew. Sparsity	Agents
MPE	Partial / Full	Dense	2-3
SMAC	Partial	Dense	2-9
LBF	Partial / Full	Sparse	2-3
RWARE	Partial	Sparse	2-4

MPE: The Multi-Agent Particle Environments (MPE) [29] consists of several two-dimensional navigation tasks. We investigate four tasks that emphasise coordination: Speaker-Listener, Spread, Adversary[1], and Predator-Prey[1]. Agent observations consist of high-level feature vectors including relative agent and landmark locations. The actions allow for two-dimensional navigation. All tasks but Speaker-Listener, which also requires binary communication, are fully observable. MPE tasks serve as a benchmark for agent coordination and their ability to deal with non-stationarity [32] due to significant dependency of the reward with respect to joint actions. Individual agents not coordinating effectively can severely reduce received rewards.

SMAC: The StarCraft Multi-Agent Challenge (SMAC) [36] simulates battle scenarios in which a team of controlled agents must destroy an enemy team using fixed policies. Agents observe other units within a fixed radius, and can move around and select enemies to attack. We consider five tasks in this environment which vary in the number and types of units controlled by agents. The primary challenge within these tasks is the agents' ability to accurately estimate the value of the current state under partial observability and a growing number of cooperating agents of diverse types across tasks.

LBF: In Level-Based Foraging (LBF) [3], agents must collect items which are spread randomly in a grid-world. Agents and items have skill levels, such that a group of one or more agents can collect an item if the sum of their levels exceed the item's level. We define five tasks in LBF which vary in world size, number of agents and items, and observability conditions. Two of these tasks require a high degree of cooperation in the sense that all agents must simultaneously participate to be able to collect any item, and for that

[1] Adversary and Predator-Prey are originally competitive tasks. The agents controlling the adversary and prey, respectively, are controlled by a pretrained policy obtained by training all agents with the MADDPG algorithm for 25000 episodes. For more details, see Appendix A.

(a) MPE Speaker-Listener (b) SMAC 3m (c) LBF 10x10-2p-8f (d) RWARE Tiny 2p

Figure 1: Task instances from the four environments.

reason are substantially harder.

RWARE: The Multi-Robot Warehouse environment (RWARE) represents a cooperative, partially-observable task with sparse rewards. RWARE simulates a grid-world warehouse in which agents (robots) must locate and deliver requested shelves to workstations and return them after delivery. Agents are only rewarded for completely delivering requested shelves and observe a 3×3 grid containing information about the surrounding agents and shelves. We define three tasks which vary in world size, number of agents and shelf requests. The sparsity of the rewards makes this a hard environment, since agents must correctly complete a series of action before receiving any reward. Additionally, observations are comparably high-dimensional.

5 EVALUATION PROTOCOL AND METRICS

5.1 Evaluation Protocol

To account for the improved sample efficiency of off-policy over on-policy algorithms and to allow for fair comparisons, we train off-policy algorithms for two million steps and on-policy algorithms for 20 million steps. While on-policy algorithms receive a factor of ten more environment samples, the samples are batched together and the same number of network updates are performed. By not reusing samples through an experience replay, on-policy algorithms are less sample efficient, but not generally slower (if the simulator is fast enough) and thus this empirical adjustment is fair. We evaluate off-policy algorithms every 50,000 steps and on-policy algorithms every 500,000 steps. As a result, a total of 40 evaluations are executed during the training of each algorithm. For matrix games, we train off-policy algorithms for 250,000 steps and on-policy for 2,500,000 steps and evaluate every 2,500 and 25,000 steps, respectively, for a total of 100 evaluations to enable more precise convergence analysis. Each algorithm is trained with several different hyperparameters using ten different seeds. Implementation details are described in Appendix E and best identified hyperparameters can be found in Appendix F.

5.2 Performance Metrics

Maximum returns: For each algorithm, we identify the evaluation and hyperparameter configuration that achieves the highest average evaluation returns across ten random seeds during training. Using these hyperparameters and trained models for each of the ten runs at the identified evaluation timestep, we evaluate and compute the average evaluation returns across 1000 episodes for each seed. We report the average returns and their standard deviation across the ten seeds. These values serve as a robust metric for maximum expected performance. Note that in SMAC, we report the average win rate instead of returns.

Average returns: We also report the average returns achieved throughout all evaluations during training using the same identified hyperparameters for each algorithm in Appendix D. Due to this metric being computed over all evaluations executed during training, it considers learning speed besides final achieved returns. To visualise convergence, we show the normalised average returns of all algorithms in each environment. Returns are normalised across all seeds and tasks of each environment in $[0, 1]$ range.

5.3 Reliability Metrics

RL is known to be sensitive to hyperparameters, seeds [17] and implementation details [13]. These factors equally apply to MARL. The presence of other learning agents in the environment can cause significant non-stationarity [32] which can lead to unstable policies during training. We adopt the following metrics, suggested by Chan et al. [6], to quantify the reliability of all MARL algorithms and present them as relative rankings. For more details, see [6].

Dispersion across Time (DT): We report the dispersion of returns achieved during training. The dispersion refers to the variability of returns and is computed using the inter-quartile range (IQR) of sliding windows over training returns. This metric aims to identify the smoothness of training progress, as high dispersion corresponds to short-term variability of training returns suggesting noisy training.

Short-term Risk across Time (SRT): The short-term risk aims to measure the worst-case drop in evaluation returns from one evaluation to the next during training. For each training run, the conditional value at risk (CVaR) of the difference in evaluation returns among successive evaluations is computed. The CVaR is given by the expected value of the distribution of such differences below its α-quantile. For our experiments, we use $\alpha = 0.05$. Unlike DT, this metric indicates instability in the form of sudden drops in returns rather than general variability throughout training. Such short-term drops might indicate instabilities caused by policy changes in

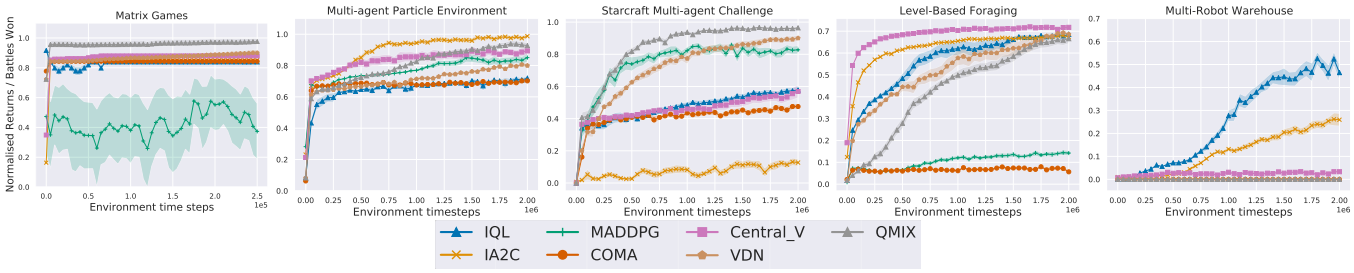

Figure 2: Normalised average returns of all algorithms in all environments showing the mean and the 95% interval.

a subset of agents.

Long-term Risk across Time (LRT): To measure long-term risk across training runs, the worst-case drop in evaluation returns during training is reported. Therefore, CVaR is applied to the drawdown [7] rather than adjacent evaluation returns. The drawdown is given by the drop in returns at each evaluation relative to the so-far best evaluation return. Whereas high SRT indicates high, sudden variability during training, this metric gives insight into long-term decay in returns. We found only marginal differences between algorithms with respect to this metric and no algorithm is found to significantly decay in returns beyond sudden variability or general instability.

Dispersion across Runs (DR): RL is known to be sensitive to random seeds [17]. Therefore, we also report the interquartile range as a robust metric for dispersion of final evaluation returns across multiple training runs with varying random seeds.

6 RESULTS

In this section, we will compare all three considered classes of MARL algorithms based on observed performance across all metrics and tasks. Section 7 provides further insights and analysis discussing when each class of MARL algorithms can be expected to be effective. Table 3 shows maximum evaluation returns with standard deviations across all tasks and algorithms. Figure 2 presents normalised average returns of the algorithms in all environments whereas Figure 3 provides rankings of all reliability metrics. Learning curves for all algorithms in each individual task are given in Appendix D.

6.1 Independent Learning

We find that IL algorithms, IQL and IA2C, perform adequately in all evaluation metrics on a range of environments and tasks despite their simplicity. However, performance of IL is limited in partially observable Speaker-Listener and across all SMAC tasks due to its inability to reason over joint information of agents. This effect becomes increasingly noticeable in harder SMAC tasks, where agents have to control a variety of melee and ranged units. In contrast to this, IL was found to be the most effective paradigm within RWARE despite partial observability. It appears that training centralised critics using joint information of all agents can significantly improve coordination, especially in partially observable tasks, but also be detrimental whenever training becomes difficult due to sparse rewards or high-dimensional observations.

IA2C: In the matrix games, IA2C slightly exceeded the risk-free joint policy in the climbing game, but was only able to learn the optimal policy in the simplest penalty game. The stochastic policy of IA2C appears to be particularly effective on fully-observable MPE tasks Predator-Prey, Spread, and Adversary as well as the majority of LBF tasks. This is also reflected in its reliability during training, scoring well in DT and SRT but experiencing large dispersion across runs for both these environments. It scores worse in most partially observable tasks, as stated above, but in contrast IA2C performs best in multiple RWARE tasks.

IQL: IQL consistently converges to sub-optimal, risk-free policies across all but the simplest matrix games. Surprisingly, it also achieves its highest evaluation performance early in training before its policy converged. IQL appears less effective in MPE, but achieves high returns on simpler SMAC tasks. In LBF, IQL also appears very reliable scoring the best out of all algorithms in terms of DT, SRT and being ranked among the best in DR.

6.2 Centralised Training Decentralised Execution

Centralised training aims to learn powerful critics over the joint observations and actions. This allows for reasoning over a larger information space, but potentially complicates the training process. We find that learning such critics is valuable in tasks which require significant coordination, such as the MPE Speaker-Listener and harder SMAC tasks due to partial observability. However, tasks such as MPE Spread, Adversary and Predator-Prey are fully observable, making independent learners competitive compared to this class of algorithms. Our results indicate that in RWARE, centralised training with critics over larger joint observation and action spaces can make the learning process more difficult without adding significant value.

6.2.1 Centralised Multi-Agent Policy Gradient. Centralised policy gradient methods vary significantly in performance.

MADDPG: Across all matrix games, MADDPG exhibits extreme instability during training and does not converge to any policy. Even in the simplest penalty game, MADDPG is unable to consistently learn the risk-free policy. It also exhibits low returns and reliability in discrete grid-world environments LBF and RWARE. We believe that these results are a direct consequence of the biased categorical

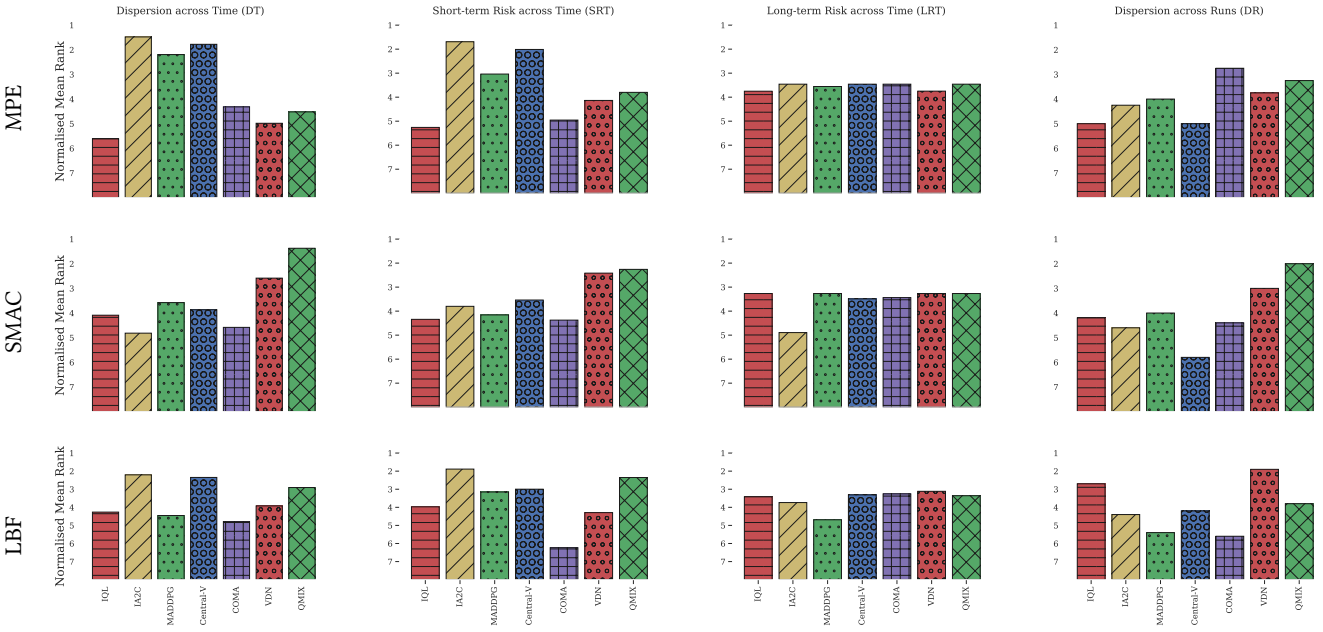

Figure 3: Reliability metrics for MPE (first row), SMAC (second row) and LBF (third row). Bars indicate the ranking, i.e. higher bars correspond to lower dispersion or risk.

reparametarisation using Gumbel-Softmax to apply MADDPG to discrete action spaces. In environments with continuous observations (MPE and SMAC), MADDPG exhibits high maximum and average returns. Given its high returns, MADDPG seems to suffer from comparably large dispersion across runs in MPE and SMAC, also suggesting some unreliability in these environments.

COMA: In the matrix games, COMA quickly converges to risk-free, sub-optimal policies in most cases. In general, COMA exhibits one of the lowest performances in most tasks and only performs competitively in few MPE and simpler SMAC tasks. Additionally, COMA shows poor reliability with particularly low rankings in DT, SRT and DR indicating high variance and noisy training with large variation across runs. We believe this instability occurs due to high variance of its counterfactual baseline causing gradients for updates to be unstable. To verify our claim, we conducted further experiments using reward standardisation showing significant improvements across multiple tasks. For more details, see Appendix B.

Central-V: Central-V is found to be the most reliable centralised policy gradient algorithm. In the climbing game, Central-V consistently identifies the sub-optimal equilibrium for payoff 7, outperforming algorithms stuck at risk-free policies, with the same being observed for the penalty game with $k = -25$. Compared to MADDPG, Central-V achieves significantly lower returns on all SMAC and some MPE tasks, due to Central-V's critic computing a centralised state value function instead of a centralised state-action value function. As a result, it does not take the joint action of all agents into account, limiting its ability to reason over coordinated actions. However, performance is not significantly affected by this in other environments and in LBF Central-V even outperforms MADDPG due to the application of a stochastic policy without the bias of the Gumbel-Softmax. Central-V also ranks high in terms of reliability across time in all environments. However, its evaluation performance seems to vary considerably across runs, ranking low in terms of DR.

6.2.2 Value Decomposition. Value decomposition is found to be an effective approach for most environments. In the majority of tasks across all environments except RWARE, VDN and QMIX outperform or at least match the highest returns of any other algorithm. This suggests that VDN and QMIX share the major advantages of centralised training with improved coordination. Additionally in SMAC, their performance is only matched by MADDPG and IQL in simpler tasks, but for harder SMAC tasks value decomposition appears to be very impactful and achieves higher returns than any other class of algorithms. Similarly, QMIX and VDN outperform all other algorithms in terms of reliability and achieved returns in matrix games. Generally, VDN and QMIX exhibit similar performance with minor improvements of QMIX over VDN in some tasks. Both rank fairly low in terms of dispersion and risk across time in MPE, indicating variance during the training process. However even in MPE, they appear to be stable across runs and generally rank high in all reliability metrics in SMAC. QMIX also ranks competitively in LBF. In RWARE, VDN and QMIX do not exhibit any learning, similar to most centralised policy gradient methods, indicating that these methods require sufficiently dense rewards to train their more complex architectures compared to independent learners.

VDN: Across all matrix games, VDN is able to perform beyond the sub-optimal equilibrium with convergence to optimal policies

Table 3: Results in all tasks across two matrix games and four multi-agent environments. The maximum returns during evaluation and the standard deviation among different seeds is presented. Ten seeds were used in all tasks and algorithms. The algorithms that achieve the highest returns as well as other algorithms with similar returns are presented in bold.

	Tasks \Algs.	IQL	IA2C	MADDPG	COMA	Central-V	VDN	QMIX
Matrix Games	Climbing	**165.80 ± 53.70**	140.40 ± 12.40	135.60 ± 77.80	129.40 ± 12.90	**175.00 ± 0.00**	**167.30 ± 66.70**	**176.50 ± 33.10**
	Pen. (0)	247.00 ± 0.50	**250.00 ± 0.00**	155.00 ± 112.20	**250.00 ± 0.00**	**250.00 ± 0.00**	**250.00 ± 0.00**	**250.00 ± 0.00**
	Pen. (-25)	165.50 ± 94.90	50.00 ± 0.00	35.00 ± 74.30	66.10 ± 61.90	70.00 ± 60.00	148.20 ± 88.70	**250.00 ± 0.00**
	Pen. (-50)	**163.10 ± 93.10**	50.00 ± 0.00	60.00 ± 97.00	50.60 ± 2.50	50.00 ± 0.00	**111.80 ± 82.00**	**149.20 ± 99.20**
	Pen. (-75)	**160.70 ± 91.30**	50.00 ± 0.00	**90.00 ± 106.80**	50.40 ± 8.50	50.00 ± 0.00	**71.80 ± 57.30**	**117.20 ± 88.40**
	Pen. (-100)	**158.40 ± 89.40**	50.00 ± 0.00	−165.00 ± 785.50	50.00 ± 0.00	50.00 ± 0.00	**78.80 ± 49.00**	**70.00 ± 6.00**
MPE	Speaker-Listener	−29.50 ± 3.20	−16.60 ± 7.10	−14.00 ± 1.70	−23.30 ± 3.10	−14.50 ± 3.60	−21.90 ± 7.10	**−12.30 ± 1.60**
	Spread	−214.10 ± 5.30	−132.30 ± 2.30	**−127.50 ± 2.70**	−183.10 ± 2.20	−132.30 ± 0.70	−213.30 ± 2.90	−158.90 ± 7.90
	Adversary	8.60 ± 0.90	**10.90 ± 0.40**	**11.50 ± 1.40**	9.30 ± 0.30	**11.30 ± 0.50**	9.60 ± 1.10	**10.30 ± 0.90**
	Predator-Prey	9.90 ± 5.10	**31.70 ± 4.10**	14.20 ± 8.80	0.70 ± 0.20	18.60 ± 9.00	20.50 ± 3.10	**29.60 ± 6.10**
SMAC	3m	**0.99 ± 0.01**	0.67 ± 0.35	**0.98 ± 0.01**	0.84 ± 0.10	0.82 ± 0.06	**0.99 ± 0.01**	**0.99 ± 0.01**
	8m	0.96 ± 0.03	0.11 ± 0.28	**0.98 ± 0.02**	0.95 ± 0.02	0.96 ± 0.03	**0.99 ± 0.01**	**0.99 ± 0.01**
	2s3z	0.70 ± 0.13	0.05 ± 0.14	0.96 ± 0.02	0.35 ± 0.13	0.77 ± 0.14	0.96 ± 0.02	**0.98 ± 0.01**
	3s5z	0.12 ± 0.12	0.00 ± 0.00	0.72 ± 0.08	0.01 ± 0.01	0.12 ± 0.18	0.69 ± 0.19	**0.95 ± 0.03**
	1c3s5z	0.19 ± 0.11	0.02 ± 0.07	0.72 ± 0.07	0.26 ± 0.07	0.23 ± 0.23	**0.91 ± 0.05**	**0.94 ± 0.04**
LBF	8x8-2p-3f	**1.0 ± 0.0**	0.98 ± 0.01	0.24 ± 0.06	0.18 ± 0.04	0.99 ± 0.00	0.97 ± 0.01	0.99 ± 0.01
	8x8-2p-2f-coop	**1.0 ± 0.0**	**1.0 ± 0.0**	0.00 ± 0.00	0.01 ± 0.02	0.99 ± 0.00	0.98 ± 0.02	0.82 ± 0.28
	8x8-3p-1f-coop	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	10x10-2p-8f	**0.54 ± 0.15**	**0.58 ± 0.04**	0.04 ± 0.02	0.09 ± 0.02	**0.60 ± 0.02**	**0.60 ± 0.07**	**0.57 ± 0.06**
	8x8-2p-3f, sight=2	0.97 ± 0.10	0.94 ± 0.01	0.46 ± 0.12	0.16 ± 0.07	**0.99 ± 0.01**	0.96 ± 0.02	0.97 ± 0.02
RWARE	tiny 2p	**2.46 ± 0.67**	**3.12 ± 1.25**	0.00 ± 0.00	0.00 ± 0.00	0.31 ± 0.16	0.00 ± 0.00	0.00 ± 0.00
	tiny 4p	3.52 ± 0.69	**4.28 ± 0.31**	0.00 ± 0.00	0.00 ± 0.00	0.66 ± 0.27	0.00 ± 0.00	0.00 ± 0.00
	small 2p	**1.56 ± 0.24**	0.05 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.03 ± 0.03	0.00 ± 0.00	0.00 ± 0.00

in some runs. While VDN and QMIX perform similarly in most environments, the difference in performance is most noticeable in MPE. It appears VDN's assumption of linear value function decomposition is mostly violated in this environment. In contrast, VDN and QMIX perform almost identical in most SMAC tasks and across all LBF tasks, where the global utility can apparently be represented by a linear function of individual agents' utilities.

QMIX: Across all matrix games, QMIX achieves the highest average episodic returns and exhibits high reliability. It is the only algorithm to exceed payoff of sub-optimal equilibrium in both climbing game and harder penalty games. While QMIX achieves high returns, we also observe that QMIX appears to continue learning towards the end of training, suggesting slower convergence than other algorithms. Across almost all other tasks, QMIX performs as one of the best algorithms. Its value function decomposition allows QMIX to perform slightly better than VDN, particularly in more challenging SMAC and all MPE tasks.

7 DISCUSSION

In this section, we highlight some key take-aways from our benchmark evaluation for cooperative MARL. In particular, we discuss the circumstances in which independent learning, centralised training and value decomposition can be applied effectively.

Independent learning can be effective in multi-agent systems. Why and when? It is often stated that IL is inferior to centralised training methods due to the environment becoming non-stationary from the perspective of any individual agent. This is true in many cases and particularly crucial when IL is paired with off-policy training from an experience replay buffer, as pointed out by Lowe et al. [24]. In our experiments, IQL trains agents independently using such a replay buffer and is thereby limited in its performance in the MPE environment. There, agents depend on the information about other agents and their current behaviour to choose well-coordinated actions and hence learning such policies from a replay buffer (where other agents differ in their behaviour) appears infeasible. However, this is not a concern to multi-agent

environments in general. In smaller SMAC and most tested LBF tasks, each agent can identify its intended behaviour by simply considering its local observation, without requiring extensive coordination with other agents. E.g. in LBF, agents "only" have to learn to pick up the food until it is collected. Of course, they will have to coordinate such behaviour with other agents, but naively going to food (especially when others are also close) and attempting to pick it up can be a viable local-minima policy, and hard to improve upon. Whenever more complicated coordination is required, such as simultaneously picking up an item with higher level, exploring and learning those joint actions becomes difficult. Similar challenges are faced in harder SMAC tasks, where agents are required to focus fire and learn techniques such as kiting.

IA2C on the other hand learns on-policy, so there is no such training from a replay buffer. It should be expected to perform much better in MPE as it does learn to regard the currently observed behaviour of other agents. As long as effective behaviour is eventually identified through time and exploration, IA2C can learn more effective policies than IQL despite also learning independently. However, IA2C is found to be less effective in SMAC.

Centralised information is required for most partially- observable environments. It seems unsurprising, but we note that the availability of joint information (observations and actions) over all agents serves as a powerful training signal to optimise individual policies whenever the full state is not available to individual agents. Comparing the performance in Table 3 of IA2C and Central-V, two almost identical algorithms aside from their critics, we notice that Central-V shows substantially higher returns in MPE Speaker-Listener and SMAC. Across all SMAC tasks, agent observations lack important information about other units outside of their vision due to partial observability. In contrast, IA2C and Central-V have access to the same information in fully-observable tasks such as most LBF and MPE tasks, leading to similar returns. However, RWARE appears to be an exception to this rule as we discuss next.

Centralised training can complicate training. While RWARE is partially observable, the information included in the joint observation in addition to individual observations is not essential for learning effective policies: unobserved agents further away do not influence an agent's immediate decision. Rather, we observe that it is very challenging for tested MARL algorithm to learn any effective policy due to considerably sparse rewards and higher observation dimensionality. In environments such as RWARE, where learning any policy achieving non-zero rewards is the concern, centralised critics can make learning even more difficult due to their larger network sizes. Hence, it should be carefully considered (1) whether centralised training can benefit the multi-agent system (due to required coordination or partial observability) but also (2) whether sufficient training signals are present to train the more complicated networks.

Value decomposition – VDN vs QMIX. Lastly, we address the differences observed in value decomposition applied by VDN and QMIX. Such decomposition offers an improvement in comparison to the otherwise similar IQL algorithm across most tasks. Both VDN and QMIX are different in their decomposition. QMIX uses a trainable mixing network to compute the joint Q-value compared to VDN which assumes linear decomposition. With the additional flexibility, QMIX aims to improve learnability, i.e. it simplifies the learning objective for each agent to maximise, while ensuring the global objective is maximised by all agents [1]. Such flexibility appears to mostly benefit convergence in harder SMAC tasks with heterogeneous agents (e.g. 1c3s5z), but also comes at additional expense seen in environments like LBF, where the decomposition did not have to be complex and the added complexity of a mixing network slows down convergence. The only environment where we can see significant differences in achieved returns between both value decomposition algorithms is MPE, where QMIX performs very well but VDN appears to learn suboptimal policies throughout most tasks. It appears that the dependency of rewards with respect to complicated interactions between agents in MPE tasks and the resulting non-stationarity benefits more complex decomposition.

8 CONCLUSION

We evaluated seven MARL algorithms in a total of 23 cooperative learning tasks, including combinations of partial/full observability, sparse/dense rewards, and a number of agents that range from two to nine. We compared algorithm performance in terms of achieved returns and reported reliability through several metrics designed to measure the robustness of learning and evaluation. Additionally, we further analysed the effectiveness of independent learning, centralised training and value decomposition and identify types of environments in which either strategy is expected to perform well. Our results indicate that (1) independent learning can perform competitively across a range of tasks despite their simplicity, suggesting that these algorithms should be considered as baselines and building blocks for MARL research, (2) centralised learning should be applied in most (but not all!) partially observable tasks, and (3) that value decomposition is a very effective paradigm throughout most types of environments. To conclude, while deep RL has improved considerably in recent years, MARL still requires much research effort. This work is limited to cooperative environments and commonly-used MARL algorithms. Competitive environments as well as solutions to a variety of MARL challenges such as exploration, communication, and opponent modelling need to be studied separately in the future. We hope that our work sheds some light on the relative strengths and limitations of existing MARL algorithms, to provide guidance in terms of practical considerations and future research.

REFERENCES

[1] Adrian K Agogino and Kagan Tumer. 2008. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *International Conference on Autonomous Agents and Multi-Agent Systems* (2008).

[2] Stefano V. Albrecht and Subramanian Ramamoorthy. 2012. Comparative Evaluation of MAL Algorithms in a Diverse Set of Ad Hoc Team Problems. *International Conference on Autonomous Agents and Multi-Agent Systems* (2012).

[3] Stefano V. Albrecht and Subramanian Ramamoorthy. 2013. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. *International Conference on Autonomous Agents and Multi-Agent Systems* (2013).

[4] Jean-Pierre Benoit and Vijay Krishna. 1984. Finitely repeated games. (1984).

[5] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* (2002).

[6] Stephanie CY Chan, Sam Fishman, John Canny, Anoop Korattikara, and Sergio Guadarrama. 2020. Measuring the Reliability of Reinforcement Learning Algorithms. *International Conference on Learning Representations* (2020).

[7] Alexei Chekhlov, Stanislav Uryasev, and Michael Zabarankin. 2005. Drawdown measure in portfolio optimization. *International Journal of Theoretical and Applied Finance* (2005).

[8] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *Advances in Neural Information Processing Systems Workshop on Deep Learning* (2014).

[9] Caroline Claus and Craig Boutilier. 1998. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI Conference on Artificial Intelligence* (1998).

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

[11] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. 2019. LIIR: Learning Individual Intrinsic Reward in Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems* (2019).

[12] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking deep reinforcement learning for continuous control. *International Conference on Machine Learning* (2016).

[13] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. 2019. Implementation Matters in Deep RL: A Case Study on PPO and TRPO. *International Conference on Learning Representations* (2019).

[14] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. 2016. Learning to communicate with deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems* (2016).

[15] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. *AAAI Conference on Artificial Intelligence* (2018).

[16] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. 2016. Opponent modeling in deep reinforcement learning. *International Conference on Machine Learning* (2016).

[17] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep Reinforcement Learning that Matters. *AAAI Conference on Artificial Intelligence* (2018).

[18] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. 2019. A survey and critique of multiagent deep reinforcement learning. *International Conference on Autonomous Agents and Multi-Agent Systems* (2019).

[19] Shariq Iqbal and Fei Sha. 2019. Actor-attention-critic for multi-agent reinforcement learning. *International Conference on Machine Learning* (2019).

[20] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[21] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro A Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. 2019. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. *International Conference on Machine Learning* (2019).

[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

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

[24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems* (2017).

[25] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning* (2013).

[26] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712* (2016).

[27] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. *International Conference on Machine Learning* (2016).

[28] 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. 2015. Human-Level Control Through Deep Reinforcement Learning. *Nature* (2015).

[29] Igor Mordatch and Pieter Abbeel. 2017. Emergence of Grounded Compositional Language in Multi-Agent Populations. *arXiv preprint arXiv:1703.04908* (2017).

[30] John Nash. 1951. Non-cooperative games. *Annals of mathematics* (1951), 286–295.

[31] Frans A Oliehoek, Christopher Amato, et al. 2016. *A concise introduction to decentralized POMDPs*. Springer.

[32] Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, and Stefano V Albrecht. 2019. Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning. *arXiv preprint arXiv:1906.04737* (2019).

[33] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. 2018. Modeling Others using Oneself in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:1802.09640* (2018).

[34] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. *International Conference on Machine Learning* (2018).

[35] Heechang Ryu, Hayong Shin, and Jinkyoo Park. 2020. Multi-Agent Actor-Critic with Hierarchical Graph Attention Network. *AAAI Conference on Artificial Intelligence* (2020).

[36] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. 2019. The StarCraft multi-agent challenge. *International Conference on Autonomous Agents and Multi-Agent Systems* (2019).

[37] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

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

[39] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. *International Conference on Machine Learning* (2014).

[40] Sainbayar Sukhbaatar, Rob Fergus, et al. 2016. Learning multiagent communication with backpropagation. *Advances in Neural Information Processing Systems* (2016).

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

[42] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine learning* (1988).

[43] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. *International Conference on Machine Learning* (1993).

[44] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. 2017. StarCraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782* (2017).

[45] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. 2019. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057* (2019).

[46] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* (1992).

A TASK SPECIFICATIONS

Below, we provide details and descriptions of the environments and tasks used for the evaluation.

A.1 Multi-Agent Particle Environment [29]

This environment consists of multiple tasks involving the cooperation and competition between agents. All tasks involve particles and landmarks in a continuous two-dimensional environment. Observations consist of high-level feature vectors and agents are receiving dense reward signals. The action space among all tasks and agents is discrete and usually includes five possible actions corresponding to no movement, move right, move left, move up or move down. All experiments in this environment are executed with a maximum episode length of 25, i.e. episodes are terminated after 25 steps and a new episode is started. All considered tasks are visualised in Figure 4.

MPE Speaker-Listener: In this task, one static speaker agent has to communicate a goal landmark to a listening agent capable of moving. There are a total of three landmarks in the environment and both agents are rewarded with the negative Euclidean distance of the listener agent towards the goal landmark. The speaker agent only observes the colour of the goal landmark. Meanwhile, the listener agent receives its velocity, relative position to each landmark and the communication of the speaker agent as its observation. As actions, the speaker agent has three possible options which have to be trained to encode the goal landmark while the listener agent follows the typical five discrete movement actions of MPE tasks.

MPE Spread: In this task, three agents are trained to move to three landmarks while avoiding collisions with each other. All agents receive their velocity, position, relative position to all other agents and landmarks. The action space of each agent contains five discrete movement actions. Agents are rewarded with the sum of negative minimum distances from each landmark to any agent and a additional term is added to punish collisions among agents.

MPE Adversary: In this task, two cooperating agents compete with a third adversary agent. There are two landmarks out of which one is randomly selected to be the goal landmark. Cooperative agents receive their relative position to the goal as well as relative position to all other agents and landmarks as observations. However, the adversary agent observes all relative positions without receiving information about the goal landmark. All agents have five discrete movement actions. Agents are rewarded with the negative minimum distance to the goal while the cooperative agents are additionally rewarded for the distance of the adversary agent to the goal landmark. Therefore, the cooperative agents have to move to both landmarks to avoid the adversary from identifying which landmark is the goal and reaching it as well. For this competitive scenario, we use a fully cooperative version where the adversary agent is controlled by a pretrained model obtained by training all agents using the MADDPG algorithm for 25,000 episodes.

MPE Predator-Prey: In this task, three cooperating predators hunt a forth agent controlling a faster prey. Two landmarks are placed in the environment as obstacles. All agents receive their own velocity and position as well as relative positions to all other landmarks and agents as observations. Predator agents also observe the velocity of the prey. All agents choose among five movement actions. The agent controlling the prey is punished for any collisions with predators as well as for leaving the observable environment area (to prevent it from simply running away without needing to evade). Predator agents are collectively rewarded for collisions with the prey. We employ a fully cooperative version of this task with a pretrained prey agent. Just as for the Adversary task, the model for the prey is obtained by training all agents using the MADDPG algorithm for 25,000 episodes.

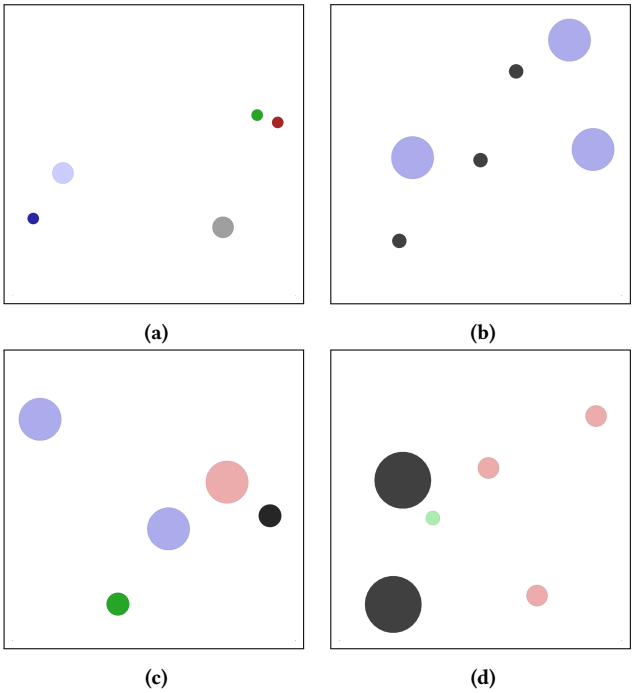

Figure 4: Illustration of MPE tasks (a) Speaker-Listener, (b) Spread, (c) Adversary and (d) Predator-Prey.

A.2 StarCraft Multi-Agent Challenge [36]

The StarCraft Multi-Agent Challenge is a set of fully cooperative, partially observable multi-agent tasks. This environment implements a variety of micromanagement tasks based on the popular real-time strategy game StarCraft II[2] and makes use of the StarCraft II Learning Environment (SC2LE) [44]. Each task is a specific combat scenario in which a team of agents, each agent controlling an individual unit, battles against an army controlled by the centralised built-in AI of the StarCraft game. These tasks require agents to learn precise sequences of actions to enable skills like *kiting* as well as coordinate their actions to focus their attention on specific opposing units. All considered tasks are symmetric in their structure, i.e. both armies consist of the same units. Figure 5 visualises each considered task in this environment. Experiments have been run using StarCraft II on version 2.4.6.2.69232.

SMAC 3m: In this scenario, each team is constructed by three space marines. These ranged units have to be controlled to focus

[2]StarCraft II is a trademark of Blizzard Entertainment™.

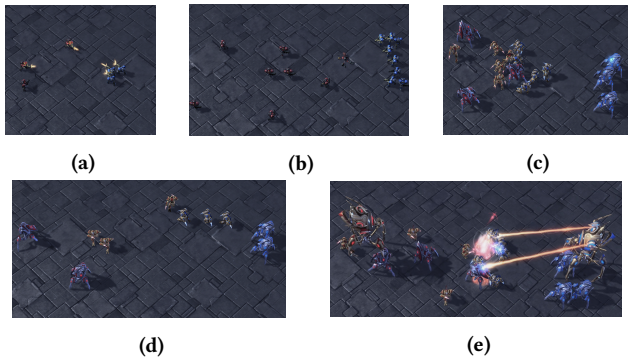

Figure 5: Illustration of SMAC tasks (a) 3m, (b) 8m, (c) 2s3z, (d) 3s5z and (e) 1c3s5z.

fire on a single opponent unit at a time and attack collectively to win this battle.

SMAC 8m: In this scenario, each team controls eight space marines. While the general strategy is identical to the 3m scenario, coordination becomes more challenging due to the increased number of agents controlling marines.

SMAC 2s3z: In this scenario, each team controls two stalkers and three zealots. While stalkers are ranged units, zealots are melee units, i.e. they are required to move closely to enemy units to attack. Therefore, own units still have to learn to focus their fire on single opponent units at a time. Additionally, stalkers are required to learn kiting to consistently move back in between attacks to keep a distance between themselves and enemy zealots to minimise received damage while maintaining high damage output.

SMAC 3s5z: This scenario requires the same strategy as the 2s3z task. Both teams control three stalker and five zealot units. Due to the increased number of agents, the task becomes slightly more challenging.

SMAC 1c3s5z: In this final scenario, both teams control one colossus in addition to three stalkers and five zealots. A colossus is a durable unit with ranged, spread attacks. Its attacks can hit multiple enemy units at once. Therefore, the controlled team has to coordinate to avoid many units to be hit by the enemy colossus at ones while enabling the own colossus to hit multiple enemies all together.

A.3 Level-Based Foraging [3]

The Level-Based Foraging environment consists of tasks focusing on the coordination of involved agents. The task for each agent is to navigate the grid-world map and collect items. Each agent and item is assigned a level and items are randomly scattered in the environment. In order to collect an item, agents have to choose a certain action next to the item. However, such collection is only successful if the sum of involved agents' levels is equal or greater than the item level. Agents receive reward equal to the level of the collected item. Figure 6 shows the tasks used for our experiments. By default, every agent can observe the whole map, including the positions and levels of all the entities and can choose to act by moving in one of four directions or attempt to pick up an item. In

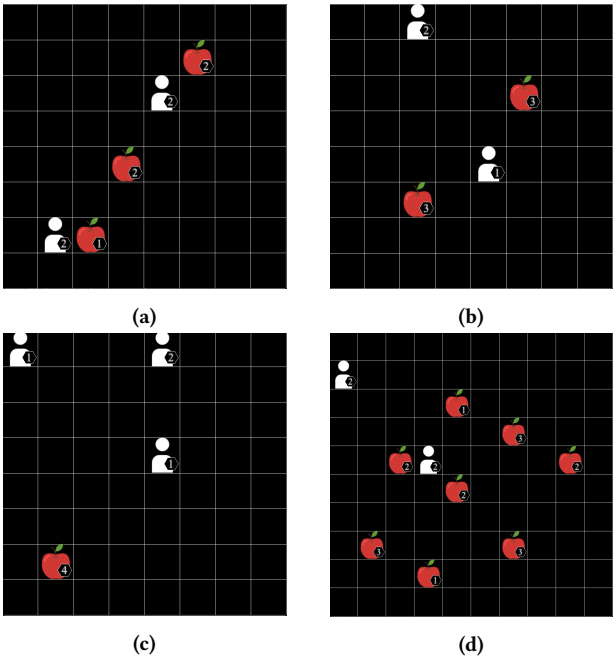

Figure 6: Illustrations of LBF tasks (a) LBF-8x8-2p-3f, (b) LBF-8x8-2p-2f-coop, (c) LBF-8x8-3p-1f-coop and (d) LBF-10x10-2p-8f.

the partially observable version, denoted with 'sight=2', agents can only observe entities in a 5×5 grid surrounding them.

LBF-8x8-2p-3f: An 8×8 grid-world with two agents and three items placed in random locations. Item levels are random and might require agents to cooperate, depending on the level.

LBF-8x8-2p-2f-coop: An 8×8 grid-world with two agents and two items. This is a cooperative version, so agents will always need to cooperate to collect an item due to the levels of placed items.

LBF-8x8-3p-1f-coop: An 8×8 grid-world with three agents and one item. This is a cooperative version, so all three agents will need to collect the item simultaneously.

LBF-10x10-2p-8f: A 10×10 grid-world with two agents and ten items. The time-limit (25 timesteps) is often not enough for all items to be collected. Therefore, the agents need to spread out and collect as many items as possible in the short amount of time given in each episode.

LBF-8x8-2p-3f, sight=2: Similar to the first variation, but partially observable. The agents' vision is limited to a 5×5 box centred around the agent.

A.4 Multi-Robot Warehouse

The multi-robot warehouse environment is a set of collaborative, partially observable multi-agent tasks simulating a warehouse operated by robots. Each agent controls a single robot aiming to collect requested shelves. At all times, N shelves are requested (where N is the number of agents). Each timestep a request is delivered to the goal location, a new (currently unrequested) shelf is uniformly sampled and added to the list of requests. Agents observe a 3×3

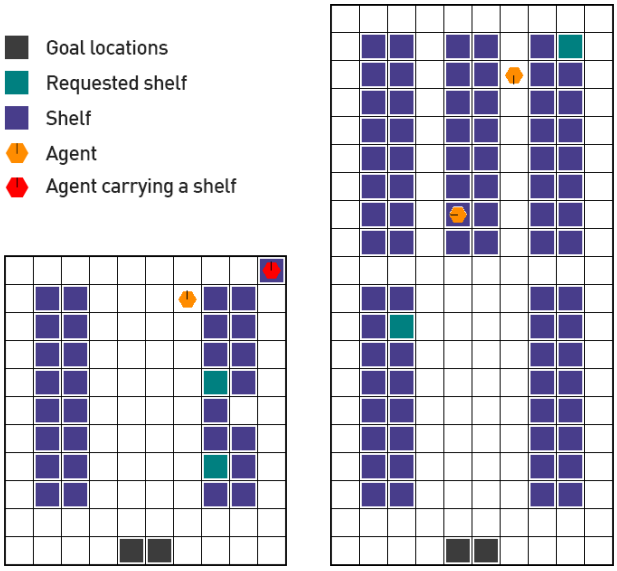

Figure 7: Illustrations of (a) tiny-2ag and (b) small-2ag.

grid including information about potentially close agents, given by their location and rotation, as well as information on surrounding shelves and a list of requests. The action space is discrete and contains of four actions corresponding to turning left or right, moving forward and loading or unloading a shelf. Agents are only rewarded whenever an agent is delivering a requested shelf to a goal position. Therefore, a very specific and long sequence of actions is required to receive any non-zero rewards, making this environment very sparsely rewarded. We use multi-robot warehouse tasks with warehouses of varying size and number of agents N (which is also equal to the number of requested shelves). Figure 7 illustrates the tiny and small warehouses with 2 agents.

RWARE-tiny-2p: A 10×11 grid-world with two agents.

RWARE-tiny-4p: A 10×11 grid-world with four agents.

RWARE-small-2p: A 10×20 grid-world with two agents. The rewards are much sparser due to the much larger world.

B COMA BASELINE ANALYSIS

We conducted further experiments to verify our hypothesis that COMA's counterfactual baseline is ineffective in reducing variance of the reward signal. In order to gain insight on this matter, we conduct experiments on four tasks: MPE Speaker-Listener, MPE Spread, LBF 8x8-2p-3f and SMAC 2s3z. For all these tasks, we ran experiments using five random seeds with the same hyperparameters as specified in Appendix F. Training was extended to 160M environment timesteps for MPE and LBF tasks and about 120M environment timesteps for SMAC 2s3z. We run these experiments once without any modification and once applying standardisation to each batch of reward signals during the optimisation by computing

$$\bar{r} = \frac{r - \mu(r)}{\sigma(r) + \delta}$$

where r refers to the batch of rewards, $\mu(r)$ and $\sigma(r)$ denote the mean and standard deviation across this reward batch. δ is a small

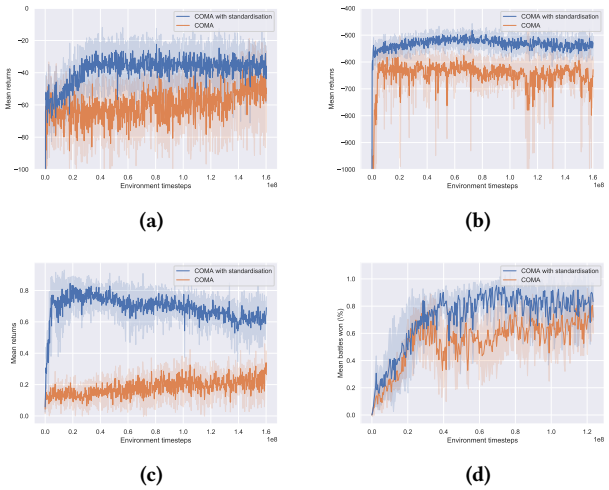

Figure 8: Average returns for COMA with and without standardised rewards in (a) MPE Speaker-Listener, (b) MPE Spread, (c) LBF 8x8-2p-3f and average win rate in (d) SMAC 2s3z.

constant introduced for numerical stability (we use $\delta = 1^{-10}$ for these experiments). Given our hypothesis, such standardisation would be expected to reduce the variance of the advantage leading to more stable optimisation. The average evaluation returns for COMA on both MPE and the LBF task as well as the average evaluation win rates for SMAC 2s3z with and without standardisation can be seen in Figure 8. Shading of all plots indicates a single standard deviation.

We find that average returns significantly increase and exhibit improved stability when reward standardisation is applied on all four tasks. This indicates the ineffectiveness of the applied baseline in reducing variance successfully and highlights the consistent benefit of applying such standardisation to rewards for actor-critic algorithms.

C AVERAGE PERFORMANCE

In Table 4, we present the mean returns during training. This is an alternative metric to Table 3 which reports maximum expected performance. In contrast, Table 4 also measures how fast agents learn, and how reliable the training performance is. We consider mean returns, as given in Table 4, to be a secondary metric due to its inability to represent if the task has been solved, but still an important element to understand the performance of algorithms.

D AVERAGE RETURN DURING TRAINING

In Figure 9, we present the returns of the algorithms during training. The returns are averaged across 10 training seeds and the shading presents the standard deviation. In on-policy algorithms, the true environment steps are higher by a factor of 10, due to the parallel computation.

Table 4: Results in all tasks of four different environments. The mean returns during training and the standard deviation among different seeds bis presented. Ten seeds were used in all tasks and algorithms.

	Tasks \Algs.	IQL	IA2C	MADDPG	COMA	Central-V	VDN	QMIX
Matrix Games	Climbing	123.8 ± 0.0	139.7 ± 17.4	90.0 ± 154.2	125.0 ± 0.0	168.2 ± 44.6	127.9 ± 2.3	162.6 ± 38.3
	Pen. (0)	238.1 ± 9.7	250.0 ± 22.2	85.0 ± 115.5	250.0 ± 0.0	250.0 ± 20.2	250.0 ± 0.0	249.9 ± 0.2
	Pen. (-25)	49.3 ± 0.0	50.0 ± 43.4	−108.9 ± 320.4	50.0 ± 0.0	70.0 ± 68.1	106.2 ± 62.0	250.0 ± 0.0
	Pen. (-50)	49.3 ± 0.0	50.0 ± 82.2	−200.1 ± 498.0	50.0 ± 0.0	50.0 ± 58.9	73.7 ± 42.3	149.0 ± 99.0
	Pen. (-75)	49.3 ± 0.0	50.0 ± 121.3	−364.2 ± 814.6	50.0 ± 0.0	50.0 ± 86.3	54.2 ± 12.1	95.3 ± 77.9
	Pen. (-100)	49.3 ± 0.0	50.0 ± 160.4	−429.9 ± 1009.7	50.0 ± 0.0	50.0 ± 113.9	54.7 ± 9.6	69.7 ± 59.1
MPE	Speaker-Listener	−42.9 ± 29.9	−21.6 ± 31.4	−16.7 ± 36.2	−27.3 ± 2.9	−16.8 ± 40.2	−33.5 ± 28.0	−18.9 ± 28.6
	Spread	−234.7 ± 66.0	−143.2 ± 31.3	−144.3 ± 37.0	−185.9 ± 5.5	−132.2 ± 32.7	−222.6 ± 33.7	−186.1 ± 39.4
	Adversary	7.5 ± 3.8	10.2 ± 4.5	10.9 ± 3.8	9.0 ± 0.6	10.8 ± 4.0	8.0 ± 3.8	8.5 ± 4.0
	Predator-Prey	5.3 ± 3.8	23 ± 11.8	7.3 ± 7.1	0.85 ± 0.4	13.2 ± 10.9	9.4 ± 6.7	15.7 ± 10.8
SMAC	3m	0.91 ± 0.19	0.36 ± 0.36	0.89 ± 0.23	0.79 ± 0.16	0.77 ± 0.15	0.92 ± 0.20	0.92 ± 0.19
	8m	0.83 ± 0.20	0.01 ± 0.05	0.87 ± 0.2	0.89 ± 0.20	0.92 ± 0.15	0.88 ± 0.21	0.91 ± 0.18
	2s3z	0.45 ± 0.21	0.01 ± 0.04	0.83 ± 0.23	0.18 ± 0.13	0.44 ± 0.25	0.81 ± 0.23	0.87 ± 0.22
	3s5z	0.04 ± 0.06	0.00 ± 0.00	0.57 ± 0.21	0.00 ± 0.01	0.03 ± 0.08	0.34 ± 0.30	0.75 ± 0.27
	1c3s5z	0.09 ± 0.07	0.00 ± 0.01	0.61 ± 0.21	0.17 ± 0.09	0.11 ± 0.13	0.57 ± 0.34	0.71 ± 0.29
LBF	8x8-2p-3f	0.90 ± 0.11	0.95 ± 0.12	0.26 ± 0.10	0.13 ± 0.06	0.96 ± 0.11	0.87 ± 0.18	0.76 ± 0.33
	8x8-2p-2f-coop	0.63 ± 0.35	0.91 ± 0.21	0.01 ± 0.00	0.00 ± 0.01	0.93 ± 0.18	0.44 ± 0.45	0.26 ± 0.38
	8x8-3p-1f-coop	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	10x10-2p-8f	0.44 ± 0.10	0.41 ± 0.11	0.07 ± 0.01	0.07 ± 0.02	0.43 ± 0.11	0.43 ± 0.15	0.40 ± 0.18
	8x8-2p-3f, sight=2	0.88 ± 0.06	0.93 ± 0.07	0.31 ± 0.09	0.13 ± 0.05	0.97 ± 0.07	0.87 ± 0.15	0.74 ± 0.30
RWARE	tiny 2p	1.44 ± 0.57	1.13 ± 0.22	0.00 ± 0.00	0.00 ± 0.00	0.17 ± 0.16	0.00 ± 0.00	0.00 ± 0.00
	tiny 4p	1.50 ± 1.24	1.55 ± 0.91	0.00 ± 0.00	0.00 ± 0.00	0.46 ± 0.27	0.00 ± 0.00	0.00 ± 0.00
	small 2p	0.69 ± 0.60	0.02 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.03 ± 0.03	0.00 ± 0.00	0.00 ± 0.00

E IMPLEMENTATION DETAILS

All algorithms use two hidden layers with ReLU [25] activation function. The number of hidden nodes in each layer range between 32 and 256 for different algorithms and environments. The results presented are based on our own implementations and implementations based on the Pymarl framework[3] [36]. We use the Adam optimiser [22] with the learning rate ranging between 0.00005 and 0.01 to optimise the objectives. Pymarl uses the RMSProp optimiser and GRU [8] networks for hidden layers. Also, in Pymarl all parameters are shared across agents.

Experimenting with MADDPG using GRU and shared parameters in the SMAC environment did not improve episodic returns. We use parallel environments [27] in on-policy algorithms and an experience replay [28] for off-policy algorithms to break the correlation between consecutive samples. In our implementations of IA2C and Central-V, we update the parameters using a batch size between 5 and 100, where each batch is used to compute n-step returns. In Pymarl implementations, the parameters are updated at the end of each episode. For exploration we use both epsilon-greedy and sampling from the soft policies.

F HYPERPARAMETERS

Below, we report the best hyperparameters identified for each algorithm on every task in Tables 5-11.

[3]Pymarl is available at https://github.com/oxwhirl/pymarl

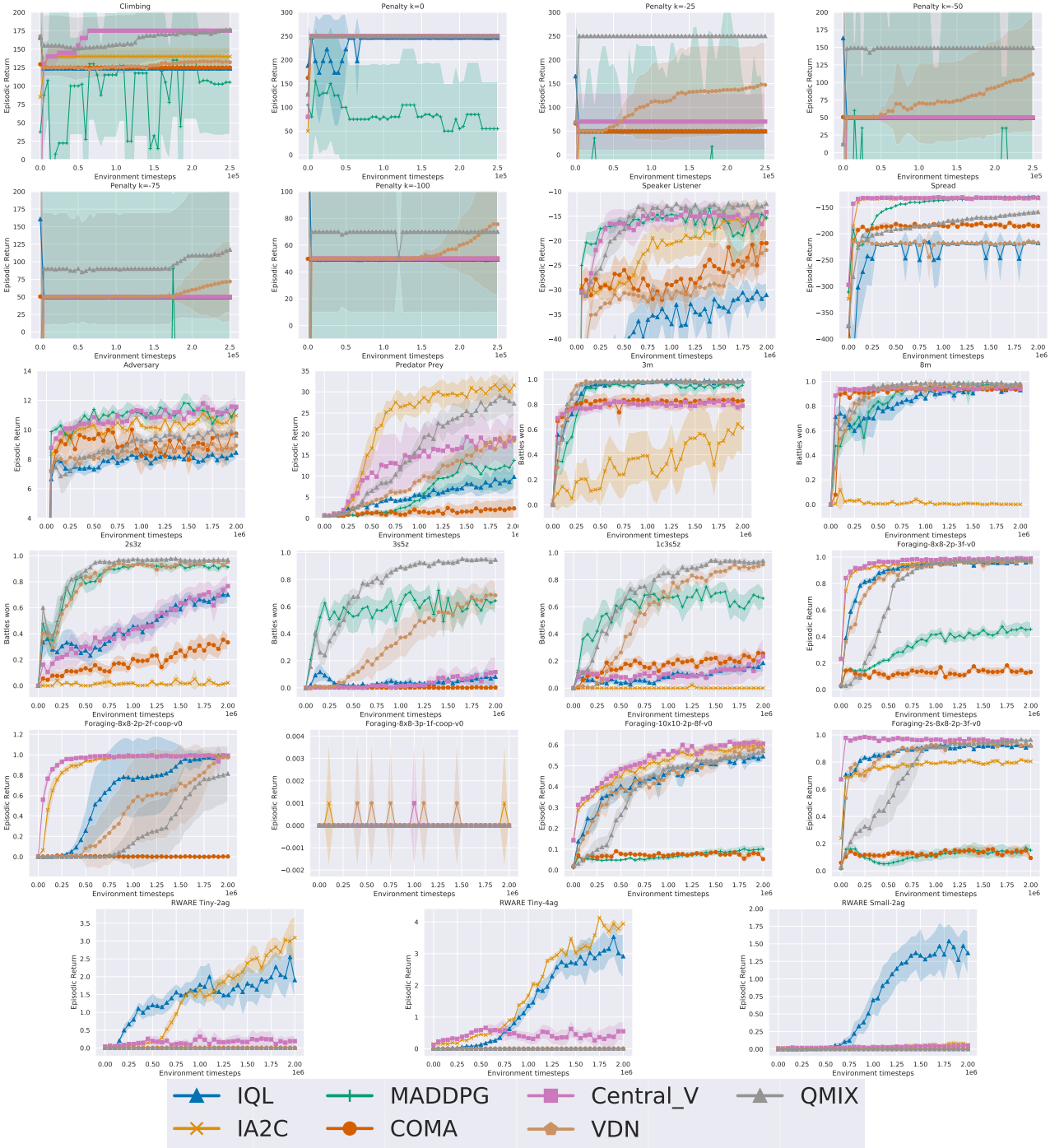

Figure 9: Average episodic returns and standard deviation during training for all seven algorithms in 17 tasks.

Table 5: Hyperparameters that lead to maximum returns in all tasks for IQL. Exploration decay for epsilon over stated steps.

	Tasks \Params.	Network Hidden	LR	Exploration	Batch size	Buffer size	Target Update	Parameter Sharing
Matrix Games	Climbing	(FC) 64	0.00005	1.0 → 0.05 in 2.5e+4	128 (steps)	1,000,000 (steps)	soft (0.05)	No
	Pen (0)	(FC) 64	0.00005	1.0 → 0.05 in 2.5e4	128 (steps)	1,000,000 (steps)	soft (0.05)	No
	Pen (-25)	(FC) 64	0.00005	1.0 → 0.05 in 2.5e+4	128 (steps)	1,000,000 (steps)	soft (0.05)	No
	Pen (-50)	(FC) 64	0.00005	1.0 → 0.05 in 2.5e+4	128 (steps)	1,000,000 (steps)	soft (0.05)	No
	Pen (-75)	(FC) 64	0.00005	1.0 → 0.05 in 2.5e+4	128 (steps)	1,000,000 (steps)	soft (0.05)	No
	Pen (-100)	(FC) 64	0.00005	1.0 → 0.05 in 2.5e+4	128 (steps)	1,000,000 (steps)	soft (0.05)	No
MPE	Speaker Listener	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	Spread	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	Predator Prey	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	Adversary	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
SMAC	3m	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	8m	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	2c3z	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	3s5z	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	1c3s5z	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
LBF	8x8-2p-3f	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	8x8-2p-2f-coop	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	8x8-3p-1f-coop	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	10x10-2p-8f	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
	8x8-2p-3f sight=2	(GRU) 64	0.0005	1.0 → 0.05 in 5e+4	32 (episodes)	5000 (episodes)	hard (200)	Yes
RWARE	tiny 2p	(FC) 256	0.00005	1.0 → 0.01 in 1.75e+6	128 (steps)	1,000,000 (steps)	soft (0.05)	No
	tiny 4p	(FC) 256	0.00005	1.0 → 0.01 in 1.75e+6	128 (steps)	1,000,000 (steps)	soft (0.05)	No
	small 2p	(FC) 128	0.00005	1.0 → 0.01 in 2e+6	128 (steps)	1,000,000 (steps)	soft (0.05)	No

Table 6: Hyperparameters that lead to maximum returns in all tasks for IA2C. In SMAC, epsilon-greedy policy is applied with stated epsilon decay. In all other environments, a soft policy of the actor network is applied by sampling from its policy. N-step returns are used for optimisation. Stated size of hidden network layers are for actor and critic networks. For SMAC, GRU layers are used only for the actor whereas the critic uses fully-connected layers in all environments.

	Tasks \Params.	Network Hidden	LR	Exploration	Entropy Coef	N steps	Parallel Environments	Parameter Sharing
Matrix Games	Climbing	(FC) 32	0.0003	Soft Policy	0.01	5	10	No
	Pen (0)	(FC) 32	0.0003	Soft Policy	0.1	5	10	No
	Pen (-25)	(FC) 32	0.0003	Soft Policy	0.01	5	10	No
	Pen (-50)	(FC) 32	0.0003	Soft Policy	0.01	5	10	No
	Pen (-75)	(FC) 32	0.0003	Soft Policy	0.1	5	10	No
	Pen (-100)	(FC) 32	0.0003	Soft Policy	0.01	5	10	No
MPE	Speaker Listener	(FC) 128	0.0007	Soft Policy	0.01	25	10	No
	Spread	(FC) 128	0.0003	Soft Policy	0.01	5	10	No
	Predator Prey	(FC) 128	0.0007	Soft Policy	0.01	25	10	No
	Adversary	(FC) 128	0.0003	Soft Policy	0.01	25	10	No
SMAC	3m	(GRU) 64	0.0005	0.5→0.01 in 1e+5	0.0	episode length	8	Yes
	8m	(GRU) 64	0.0005	0.5→0.01 in 1e+5	0.0	episode length	8	Yes
	2c3z	(GRU) 64	0.0005	0.5→0.01 in 1e+5	0.0	episode length	8	Yes
	3s5z	(GRU) 64	0.0005	0.5→0.01 in 1e+5	0.0	episode length	8	Yes
	1c3s5z	(GRU) 64	0.0005	0.5→0.01 in 1e+5	0.0	episode length	8	Yes
LBF	8x8-2p-3f	(FC) 64	0.0007	Soft Policy	0.01	25	10	No
	8x8-2p-2f-coop	(FC) 64	0.0003	Soft Policy	0.01	5	10	No
	8x8-3p-1f-coop	(FC) 128	0.0003	Soft Policy	0.01	25	10	No
	10x10-2p-8f	(FC) 128	0.0007	Soft Policy	0.01	25	10	No
	8x8-2p-3f sight=2	(FC) 128	0.0007	Soft Policy	0.01	25	10	No
RWARE	tiny 2p	(FC) 128	0.0007	Soft Policy	0.01	100	10	No
	tiny 4p	(FC) 128	0.0003	Soft Policy	0.01	50	10	No
	small 2p	(FC) 128	0.0003	Soft Policy	0.01	50	10	No

Table 7: Hyperparameters that lead to maximum returns in all tasks for MADDPG. The target networks are updated using soft updates with $\tau = 0.01$ and fully-connected hidden layers are used in all environments. Batch size and buffer capacity are given by individual steps, not episodes. Update frequency denotes the number of steps taken in the environment before each update.

	Tasks \Params.	Network Hidden	LR	Exploration	Batch Size	Buffer Size	Update Frequency
Matrix Games	Climbing	64	0.0005	Soft Policy	512	100000	25
	Pen (0)	64	0.0005	Soft Policy	512	100000	25
	Pen (-25)	64	0.0005	Soft Policy	512	100000	25
	Pen (-50)	64	0.0005	Soft Policy	512	100000	25
	Pen (-75)	64	0.0005	Soft Policy	512	100000	25
	Pen (-100)	64	0.0005	Soft Policy	512	100000	25
MPE	Speaker Listener	128	0.01	Soft Policy	1024	1000000	100
	Spread	128	0.001	Soft Policy	1024	1000000	100
	Predator Prey	128	0.001	Soft Policy	1024	1000000	100
	Adversary	128	0.01	Soft Policy	1024	1000000	100
SMAC	3m	128	0.005	1.0 → 0.05 in 5e+5	32	100000	1
	8m	128	0.005	1.0 → 0.05 in 5e+5	32	100000	1
	2c3z	128	0.005	1.0 → 0.05 in 5e+5	32	100000	1
	3s5z	128	0.005	1.0 → 0.05 in 5e+5	32	100000	1
	1c3s5z	128	0.005	1.0 → 0.05 in 5e+5	32	100000	1
LBF	8x8-2p-3f	128	0.001	1.0 → 0.05 in 6e+4	1024	1000000	100
	8x8-2p-2f-coop	128	0.001	1.0 → 0.05 in 6e+4	1024	1000000	100
	8x8-3p-1f-coop	64	0.001	1.0 → 0.05 in 6e+4	1024	1000000	100
	10x10-2p-8f	128	0.001	1.0 → 0.05 in 6e+4	1024	1000000	100
	8x8-2p-3f, sight=2	128	0.01	1.0 → 0.05 in 6e+4	1024	1000000	100
RWARE	tiny 2p	128	0.001	1.0 → 0.05 in 6e+4	1024	1000000	100
	tiny 4p	64	0.01	1.0 → 0.05 in 6e+4	512	1000000	100
	small 2p	64	0.001	1.0 → 0.05 in 6e+4	1024	1000000	100

Table 8: Hyperparameters that lead to maximum returns in all tasks for COMA. The learning rate is 0.0005. COMA is trained using 8 parallel environments in all tasks and exploration decay is given for epsilon over stated steps. Parameter sharing across agents is applied in all environments. Stated size of hidden network layers are for actor and critic networks. Hidden layers in the actor are GRU layers, with hidden layers of the critic being fully-connected layers.

	Tasks \Params.	Network Hidden	Exploration
Matrix Games	Climbing	64	0.5 → 0.01 in 1e+4
	Pen (0)	64	0.5 → 0.01 in 1e+4
	Pen (-25)	64	0.5 → 0.01 in 1e+4
	Pen (-50)	64	0.5 → 0.01 in 1e+4
	Pen (-75)	64	0.5 → 0.01 in 1e+4
	Pen (-100)	64	0.5 → 0.01 in 1e+4
MPE	Speaker Listener	64	0.5 → 0.01 in 1e+5
	Spread	64	0.5 → 0.01 in 1e+5
	Predator Prey	64	0.5 → 0.01 in 1e+5
	Adversary	64	0.5 → 0.01 in 1e+5
SMAC	3m	64	0.5 → 0.01 in 1e+5
	8m	64	0.5 → 0.01 in 1e+5
	2c3z	64	0.5 → 0.01 in 1e+5
	3s5z	64	0.5 → 0.01 in 1e+5
	1c3s5z	64	0.5 → 0.01 in 1e+5
LBF	8x8-2p-3f	32	0.5 → 0.01 in 1e+5
	8x8-2p-2f-coop	32	0.5 → 0.01 in 1e+5
	8x8-3p-1f-coop	32	0.5 → 0.01 in 1e+5
	10x10-2p-8f	32	0.5 → 0.01 in 1e+5
	8x8-2p-3f, sight=2	64	0.5 → 0.01 in 1e+5
RWARE	tiny 2p	32	1.0 → 0.05 in 1e+6
	tiny 4p	32	1.0 → 0.05 in 1e+6
	small 2p	32	1.0 → 0.05 in 1e+6

Table 9: Hyperparameters that lead to maximum returns in all tasks for Central-V. In SMAC, epsilon-greedy policy is applied with stated epsilon decay. In all other environments, a soft policy of the actor network is applied by sampling from its policy. N-step returns are used for optimisation. Stated size of hidden network layers are for actor and critic networks. For SMAC, GRU layers are used only for the actor whereas the critic uses fully-connected layers in all environments.

	Tasks \Params.	Network Hidden	LR	Exploration	Entropy Coef	N steps	Parallel Environments	Parameter Sharing
Matrix Games	Climbing	(FC) 32	0.0003	Soft Policy	0.1	5	10	No
	Pen (0)	(FC) 32	0.0003	Soft Policy	0.01	5	10	No
	Pen (-25)	(FC) 32	0.0003	Soft Policy	0.01	5	10	No
	Pen (-50)	(FC) 32	0.0003	Soft Policy	0.1	5	10	No
	Pen (-75)	(FC) 32	0.0003	Soft Policy	0.01	5	10	No
	Pen (-100)	(FC) 32	0.0003	Soft Policy	0.01	5	10	No
MPE	Speaker Listener	(FC) 128	0.0003	Soft Policy	0.01	5	10	No
	Spread	(FC) 64	0.0003	Soft Policy	0.01	5	10	No
	Predator Prey	(FC) 128	0.0003	Soft Policy	0.01	25	10	No
	Adversary	(FC) 64	0.0003	Soft Policy	0.01	25	10	No
SMAC	3m	(GRU) 64	0.0005	0.5 → 0.01 in 1e+5	X	episode length	8	Yes
	8m	(GRU) 64	0.0005	0.5 → 0.01 in 1e+5	X	episode length	8	Yes
	2c3z	(GRU) 64	0.0005	0.5 → 0.01 in 1e+5	X	episode length	8	Yes
	3s5z	(GRU) 64	0.0005	0.5 → 0.01 in 1e+5	X	episode length	8	Yes
	1c3s5z	(GRU) 64	0.0005	0.5 → 0.01 in 1e+5	X	episode length	8	Yes
LBF	8x8-2p-3f	(FC) 128	0.0007	Soft Policy	0.001	25	10	No
	8x8-2p-2f-coop	(FC) 128	0.0007	Soft Policy	0.001	25	10	No
	8x8-3p-1f-coop	(FC) 128	0.0003	Soft Policy	0.001	25	10	No
	10x10-2p-8f	(FC) 128	0.0007	Soft Policy	0.001	25	10	No
	8x8-2p-3f sight=2	(FC) 128	0.0003	Soft Policy	0.001	5	10	No
RWARE	tiny 2p	(FC) 128	0.0007	Soft Policy	0.01	100	10	No
	tiny 4p	(FC) 128	0.0007	Soft Policy	0.01	100	10	No
	small 2p	(FC) 64	0.0007	Soft Policy	0.01	100	10	No

Table 10: Hyperparameters that lead to maximum returns in all tasks for VDN. The learning rate is 0.0005, and the batch size is 32 episodes in all tasks with a buffer size of 5000 episodes. Exploration decay is given for epsilon over stated steps. Hidden layers of the network are GRU layers and parameter sharing across agents is applied in all environments.

	Tasks \Params.	Network Hidden	Hypernet embedding	Exploration
Matrix Games	Climbing	64	32	1.0 → 0.05 in 5e+3
	Pen (0)	64	32	1.0 → 0.05 in 5e+3
	Pen (-25)	64	32	1.0 → 0.05 in 5e+3
	Pen (-50)	64	32	1.0 → 0.05 in 5e+3
	Pen (-75)	64	32	1.0 → 0.05 in 5e+3
	Pen (-100)	64	32	1.0 → 0.05 in 5e+3
MPE	Speaker Listener	128	32	1.0 → 0.05 in 5e+4
	Spread	128	32	1.0 → 0.05 in 5e+4
	Predator Prey	128	32	1.0 → 0.05 in 5e+4
	Adversary	64	32	1.0 → 0.05 in 5e+4
SMAC	3m	64	64	1.0 → 0.05 in 5e+4
	8m	64	64	1.0 → 0.05 in 5e+4
	2c3z	64	64	1.0 → 0.05 in 5e+4
	3s5z	64	64	1.0 → 0.05 in 5e+4
	1c3s5z	64	64	1.0 → 0.05 in 5e+4
LBF	8x8-2p-3f	128	64	1.0 → 0.05 in 5e+4
	-2p-2f-coop	128	64	1.0 → 0.05 in 5e+4
	8x8-3p-1f-coop	64	32	1.0 → 0.05 in 5e+4
	10x10-2p-8f	64	32	1.0 → 0.05 in 5e+4
	8x8-2p-3f sight=2	64	32	1.0 → 0.05 in 5e+4
RWARE	tiny 2p	64	64	1.0 → 0.05 in 1e+6
	tiny 4p	64	64	1.0 → 0.05 in 1e+6
	small 2p	64	64	1.0 → 0.05 in 1e+6

Table 11: Hyperparameters that lead to maximum returns in all tasks for QMIX. The learning rate is 0.0005, and the batch size is 32 episodes in all tasks with a buffer size of 5000 episodes. Exploration decay is given for epsilon over stated steps. Hidden layers of the network are GRU layers and parameter sharing across agents is applied in all environments.

	Tasks \Params.	Network Hidden	Hypernet embedding	Exploration
Matrix Games	Climbing	64	64	1.0 → 0.05 in 5e+3
	Pen (0)	64	64	1.0 → 0.05 in 5e+3
	Pen (-25)	64	64	1.0 → 0.05 in 5e+3
	Pen (-50)	64	64	1.0 → 0.05 in 5e+3
	Pen (-75)	64	64	1.0 → 0.05 in 5e+3
	Pen (-100)	64	64	1.0 → 0.05 in 5e+3
MPE	Speaker Listener	128	32	1.0 → 0.05 in 5e+4
	Spread	128	32	1.0 → 0.05 in 5e+4
	Predator Prey	128	32	1.0 → 0.05 in 5e+4
	Adversary	64	32	1.0 → 0.05 in 5e+4
SMAC	33m	64	64	1.0 → 0.05 in 5e+4
	8m	64	64	1.0 → 0.05 in 5e+4
	2c3z	64	64	1.0 → 0.05 in 5e+4
	3s5z	64	64	1.0 → 0.05 in 5e+4
	1c3s5z	64	64	1.0 → 0.05 in 5e+4
LBF	8x8-2p-3f	128	64	1.0 → 0.05 in 5e+4
	8x8-2p-2f-coop	128	64	1.0 → 0.05 in 5e+4
	8x8-3p-1f-coop	64	32	1.0 → 0.05 in 5e+4
	10x10-2p-8f	64	32	1.0 → 0.05 in 5e+4
	8x8-2p-3f sight=2	128	32	1.0 → 0.05 in 5e+4
RWARE	tiny 2p	64	64	1.0 → 0.05 in 1e+6
	tiny 4p	64	64	1.0 → 0.05 in 1e+6
	small 2p	64	64	1.0 → 0.05 in 1e+6