On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning

Andrew Cohen[*†], **Ervin Teng**[*], **Vincent-Pierre Berges**[*], **Ruo-Ping Dong, Hunter Henry, Marwan Mattar, Alexander Zook, Sujoy Ganguly**

Unity Technologies

Abstract

The creation and destruction of agents in cooperative multi-agent reinforcement learning (MARL) is a critically under-explored area of research. Current MARL algorithms often assume that the number of agents within a group remains fixed throughout an experiment. However, in many practical problems, an agent may terminate before their teammates. This early termination issue presents a challenge: the terminated agent must learn from the group's success or failure which occurs beyond its own existence. We refer to propagating value from rewards earned by remaining teammates to terminated agents as the *Posthumous Credit Assignment* problem. Current MARL methods handle this problem by placing these agents in an *absorbing state* until the entire group of agents reaches a termination condition. Although absorbing states enable existing algorithms and APIs to handle terminated agents without modification, practical training efficiency and resource use problems exist.

In this work, we first demonstrate that sample complexity increases with the quantity of absorbing states in a toy supervised learning task for a fully connected network, while attention is more robust to variable size input. Then, we present a novel architecture for an existing state-of-the-art MARL algorithm which uses attention instead of a fully connected layer with absorbing states. Finally, we demonstrate that this novel architecture significantly outperforms the standard architecture on tasks in which agents are created or destroyed within episodes as well as standard multi-agent coordination tasks.

1 Introduction

In many real-world scenarios, agents must cooperate to achieve a shared objective. In these settings, single-agent reinforcement learning (RL) methods can fail or perform sub-optimally for various reasons, such as the partial observability inherent in multi-agent systems, exacerbated by increasing numbers of agents. Multi-agent reinforcement learning (MARL) promises to address these issues using the paradigm of *decentralized execution* and *centralized training* (Lowe et al. 2017). In this paradigm, agents act using *local* observations, but all *globally* available information is used during training.

The MARL literature (Lowe et al. 2017; Foerster et al. 2018; Long et al. 2020; Iqbal et al. 2021) often assumes that we will train a fixed number of agents. However, this is unsuitable for many practical applications of MARL. For instance, agents in a team-based video game may "spawn" (*i.e.,* be created) or "die" (*i.e.,* terminate before the other agents) within a single episode. Similarly, robots operating as a team may run out of battery, requiring that they terminate their trajectories before their teammates. In general, an agent can terminate early, meaning it no longer influences the environment or other agents mid-episode. Furthermore, one may also acquire additional agents mid-episode.

Typically, existing algorithms handle these situations by placing inactive agents in *absorbing states*. An agent remains in an absorbing state, irrespective of action choice, until the entire group of agents reaches a termination condition. Absorbing states enable existing algorithms to train cooperative agents to solve tasks with early termination without any architectural changes and also simplify environment and multi-agent API implementations. Furthermore, absorbing states enable decentralized POMDPs (Oliehoek and Amato 2016) and Markov Games (Littman 1994) to represent tasks with early termination without modification.

However, absorbing states introduce practical problems in training efficiency and resource use. Specifically, when using neural networks as function approximators, absorbing states introduce elements into the input distribution that, by the necessities of their construction, make the target function more challenging to approximate. Furthermore, absorbing states are not a scalable solution for large numbers of agents. Depending on the problem, a non-insignificant amount of resources may be used for agents that do not influence the environment. In an extreme case, if a group has more than 50% attrition, more resources are consumed for communication and storage for non-influential agents. These under-utilized resources are of particular concern when there are strict resource constraints.

The critical challenge posed by the early termination of an agent is credit assignment–which we call Posthumous Credit Assignment. Agents removed from the environment will not experience any rewards given to the group after termination. As such, they will not learn if their actions before termina-

[*]These authors contributed equally.

[†]Corresponding author andrew.cohen@unity3d.com

tion were valuable to the group. Absorbing states solve this by creating a pathway through the state space by which to propagate value from beyond an agent's early termination.

In this work, we present a novel architecture which uses attention instead of a fully connected layer with absorbing states for the state-of-the-art MARL algorithm COunterfactual Multi-Agent Policy Gradients (COMA) (Foerster et al. 2018). We refer to our proposed architecture as **M**ulti-**A**gent **PO**sthumous **C**redit **A**ssignment (MA-POCA). MA-POCA naturally handles agents that are created or destroyed within an episode but share a reward function. Working within the centralized training, decentralized execution framework, we need only enable the *critic* to handle a changing number of agents per timestep. By applying a self-attention mechanism (Vaswani et al. 2017) to only the active agent information before the critic, MA-POCA can scale to an arbitrary number of agents. Furthermore, the attention mechanism allows the critic to attribute the future expected value of the group to states with terminated agents *without absorbing states*. Lastly, the attention mechanism enables the implementation of the counterfactual baseline (Foerster et al. 2018) for agents with both continuous and discrete action spaces.

This work has three main contributions.

- We demonstrate that sample complexity increases with the quantity of absorbing states on a toy supervised learning task for a fully connected network, while attention is more robust to variable size input.
- We present a novel architecture, MA-POCA, which propagates rewards earned by remaining teammates to terminated agents without the use of absorbing states. Furthermore, because it does not rely on a fixed number of agents, MA-POCA also supports the *creation* of new agents during an episode.
- We present experiments on two standard multi-agent coordination tasks and two novel tasks in which agents can spawn or die. We show that MA-POCA provides improvement on the former and significantly outperforms the baselines on the latter.

2 Preliminaries

MARL notation

The setting we consider is a decentralized-POMDP (Oliehoek and Amato 2016) defined by: $(N, \mathcal{S}, \mathcal{O}, \mathcal{A}, P, r, \gamma)$ where $N \geq 1$ is the number of agents and $\mathcal{S}$ is the state space of the environment. $\mathcal{O}$ is the joint observation space of all agents $\mathcal{O} := \mathcal{O}^1 \times ... \times \mathcal{O}^N$ where $\mathcal{O}^i$ is the observation space of agent i. At time t, the environment is in state $s_t \in \mathcal{S}$ and $o_t^i \in \mathcal{O}^i$ is the local observation of agent i which is correlated with s_t. The environment state may contain information that is not available locally to any agent such as the total number of agents that are currently acting. $\mathcal{A}$ is the joint action space of all agents $\mathcal{A} := \mathcal{A}^1 \times ... \times \mathcal{A}^N$ where $\mathcal{A}^i$ is the action space of agent i. Note, the observation and action spaces for different agents do not need to be equal. Additionally, we use bold vectors to represent the joint quantities over agents *e.g.,* a joint action $\boldsymbol{a} = (a^1, \ ... \ , a^N)$ or joint observation $\boldsymbol{o} = (o^1, \ ... \ , o^N)$, where $\boldsymbol{a} \in \mathcal{A}$ and $\boldsymbol{o} \in \mathcal{O}$.

$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function where $P(s'|s, \boldsymbol{a})$ is the probability that the environment transitions to state s' given the current state s and joint action $\boldsymbol{a} \in \mathcal{A}$. $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the shared reward function where $r(s, \boldsymbol{a})$ is the reward received by all agents when the joint action $\boldsymbol{a} \in \mathcal{A}$ is taken and the environment is in state $s \in \mathcal{S}$.

Centralized Training, Decentralized Execution

In this work, we consider the Independent Actor with Centralized Critic (IACC) learning framework (Lyu et al. 2021) wherein a critic trained on joint information is used to update a set of independent actors in an actor-critic architecture (Konda and Tsitsiklis 2000). Independent Actor-Critic (IAC), which trains an independent critic and policy for each agent using only local information, and the Joint Actor-Critic (JAC), which trains a single joint policy and a joint critic, are competing approaches. In general, IAC does not perform well in tasks that require significant coordination because of the partial observability in using only local observations. Additionally, JAC is not practical in real world scenarios as a joint policy needs access to all agent observations at once to generate actions, essentially presuming perfect communications between agents and the policy node.

Let $\pi_i, \ 1 \leq i \leq N$ represent the policy of each independent actor. Given the environment state s_t and corresponding joint observation $\boldsymbol{o_t}$ and action $\boldsymbol{a_t}$, the joint policy $\boldsymbol{\pi}(\boldsymbol{a_t}|\boldsymbol{o_t})$ can be factored as $\boldsymbol{\pi}(\boldsymbol{a_t}|\boldsymbol{o_t}) = \prod_i \pi_i(a_t^i|o_t^i)$ since the agents act independently on local observations.

The centralized state value function for state s_t is defined as

$$V^{\boldsymbol{\pi}}(s_t) = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, \boldsymbol{a_{t+l}})\right] \tag{1}$$

and the centralized state-action value function as

$$Q^{\boldsymbol{\pi}}(s_t, \boldsymbol{a_t}) = r(s_t, \boldsymbol{a_t}) + \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{l=1}^{\infty} \gamma^l r(s_{t+l}, \boldsymbol{a_{t+l}})\right] \tag{2}$$

Counterfactual Baselines

Assuming a centralized Critic network where it takes all state and action from all actions as input

Counterfactual baselines leverage difference rewards (Wolpert and Tumer 2002) and introduce a per-agent baseline such that the advantage reflects the individual agent's contribution to the total reward (Foerster et al. 2018). Formally, the state action value function with the action of the individual agent marginalized out is used to compute the baseline

$$b_i(s, \boldsymbol{a}) = \mathbb{E}_{a' \sim \pi_i(\cdot|o_i)}[Q^{\boldsymbol{\pi}}(s, (\boldsymbol{a^{-i}}, a'))] \tag{3}$$

where $\boldsymbol{a^{-i}}$ is the joint action without the i'th entry. Then, the advantage of agent i is

$$\text{Adv}_i = Q^{\boldsymbol{\pi}}(s, \boldsymbol{a}) - b_i(s, \boldsymbol{a}) \tag{4}$$

and the update for agent i is

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\substack{s \sim \rho^{\pi} \\ a^i \sim \pi_i}}\left[\nabla_{\theta_i} \log \pi_i(a^i|o^i)(Q^{\boldsymbol{\pi}}(s, \boldsymbol{a}) - b_i(s, \boldsymbol{a}))\right] \tag{5}$$

$$b_i(s, \mathbf{a}_{-i}) = \sum_{a_i'} \pi_i(a_i'|o_i) \cdot Q(s, (a_i', \mathbf{a}_{-i}))$$

Intuition:

This is the **expected Q-value** if agent i had acted **according to its current policy**, and all other agents had **acted exactly as they did**.

Using this as the advantage function provides a shaped reward per agent that addresses the challenge of determining of how much an individual agent contributed to the shared reward of the group. Additionally, with the use of the counterfactual baseline, gradient descent still converges to the locally optimal policy (Foerster et al. 2018).

3 Challenges of Early Terminating Agents

In this section, we introduce the Posthumous Credit Assignment problem and discuss how it fits into decentralized POMDP framework. To the author's knowledge, this is first time the posthumous credit assignment problem has been explicitly mentioned in the literature. Then, we discuss how the decentralized POMDP framework can be extended to destroying agents via the use of an *absorbing state*. Absorbing states appear in the MARL literature (Samvelyan et al. 2019; Yu et al. 2021), but we provide an explicit discussion of them which the literature lacks. Finally, we discuss practical and theoretical issues introduced by absorbing states.

Posthumous Credit Assignment

In cooperative settings with shared rewards, an agent acts to maximize the expected future reward of the group. There are scenarios in which an individual agent's current actions enable the group to obtain reward at a later timestep, but result in the immediate termination of the agent itself (*e.g.,* a self-sacrificial event). From the perspective of a reinforcement learning agent, it has been removed from the environment and therefore will no longer receive the reward its group may obtain later. Additionally, the agent is unable to observe the state of the environment at the time the group receives the reward. Therefore, an agent must learn to maximize rewards that it cannot experience, presenting a critical credit assignment problem. We call this the *Posthumous Credit Assignment* problem.

Absorbing States

The decentralized POMDP framework is equipped to model tasks with the posthumous credit assignment problem via absorbing states. For each agent, let $o_i^{abs} \in \mathcal{O}_i$ be a unique absorbing state which agent i will occupy after it has reached a termination state and is no longer active in the environment. Note, this absorbing state needs to be *per agent* as agents with different observation spaces will enter absorbing states of different dimensions. Once agent i has entered o_i^{abs}, it will remain there, irrespective of actions, until the group has reached a termination condition and all agents reset to a new initial state. Thus, the following is true for o_i^{abs}

$$p(o_i^{abs}|o_i^{abs}, a_i) = 1, \ \forall a_i \in \mathcal{A}_i.$$

Additionally, when agent i is in state o_i^{abs}, the transition function will be independent of that agent's actions. Formally,

$$P(s'|s, \boldsymbol{a}) = P(s'|s, \boldsymbol{a}^{-i})$$

where $\boldsymbol{a}^{-i}$ is the joint action without the i'th entry. Thus, the introduction of an absorbing state transforms the original problem into a standard decentralized POMDP without the issue of posthumous credit assignment.

From this graph, it can be observed, the less float inputs -> more absorbing state -> more errors

but attention is better

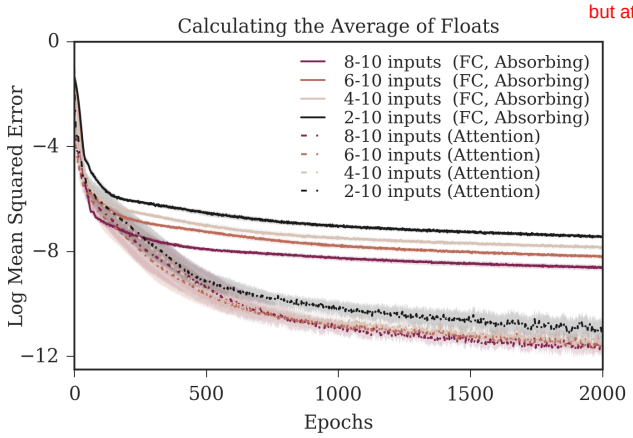


Figure 1: The sample efficiency of learning to compute the mean of a varying number of floats depends on the quantity of absorbing states as well as the representation of the input to the network. Attention outperforms absorbing states in both performance and robustness to larger variation in the input. Curves are mean and 95% confidence interval over 20 seeds.

However, though representing the setting of dying and spawning agents with the decentralized POMDP formalism and absorbing states is straightforward, issues arise when using absorbing states in practice. We argue that it is best to avoid absorbing states in general. The two major concerns are,

- the absorbing state representation complicates the learning dynamics of neural network-based function approximators;
- the complexity introduced and wasted computational resources consumed for agents that no longer impact the environment.

Absorbing State Representation The absorbing state representation is non-trivial since, in most algorithms, a function approximator, like a neural network, will ingest these state representations. Thus, the actual values and structure of the state are important as they act as input features. Additionally, the absorbing state must be disjoint from the observation space accessible by active agents. Otherwise, we would introduce (additional) partial observability since the same values could represent an active or inactive agent.

As a case study, we present a toy numerical example in Figure 1 to illustrate that fully connected layers with absorbing states are not a sample efficient input representation compared to attention networks. We train a neural network to estimate the mean of a variable number of uniformly sampled floats (up to 10) in the range $[0.25, 0.75]$. We compare two configurations (all hyperparameters are contained in Appendix D):

- the remaining values are substituted with a fixed absorbing state o_{abs} and the sample is shuffled to simulate

agents terminating early in the RL scenario. We train a fully connected network with 2 hidden layers of 32 units with ReLU activation functions;

- a self-attention (Vaswani et al. 2017) layer processes the variable input. We use a single layer entity embedding of size 32, a residual self-attention (RSA) block followed by a linear transformation into the output space. The implementation details of the RSA block are contained in Appendix B.

In Figure 1, we provide loss curves for learning to compute the mean of 2 through 10, 4 through 10, 6 through 10, and 8 through 10 floats using both fully connected layers with absorbing states and attention. We draw the number of inputs from a uniform random distribution within the ranges for each data point. In this experiment, the value of the absorbing state is $o_{abs} = 0.0$ but the trends are similar for other choices. Note that we *cannot* choose $o_{abs} \in [0.25, 0.75]$ as it will not be possible for the network to learn when a value should or should not be included in the mean. In Appendix A, we provide additional figures for the different values of $o_{abs} = -1.0,\ 1.0,\ 0.4$ showing that -1.0 and 1.0 are reasonable choices but 0.4 is not as it is contained in $[0.25, 0.75]$. Additionally, we provide an experiment when the number of absorbing states is fixed per sample to demonstrate that varying the number of floats is more challenging.

When using absorbing states, we observe that the sample complexity *increases* with the number of absorbing states. The runs with a greater maximum number of absorbing states take longer to converge. We also observe that attention significantly outperforms absorbing states in this task.

When extrapolating this result to the RL setting, the mappings learned by centralized value functions are much more complicated than this simple numerical example, likely amplifying the issues discussed. Furthermore, in this example, the presence of an absorbing state has exactly one meaning *i.e.,* do not include this input in the computation of the mean. However, in a MARL setting, the group's outcome can be positive, negative, or neutral following the early termination of an agent. Then, the variation in outcomes corresponds to high variance return targets for the single absorbing state o_{abs}. A possible alternative is to use multiple absorbing states, one for each outcome. However, it may not be possible to know which outcome will follow a given early termination or outcomes may be on a spectrum.

Implementation Complexity and Resource Constraints From a practical standpoint, absorbing states require both additional implementation complexity and increased resource overhead. These issues arise mainly in two areas: the sizing of the function approximator that represents the centralized value function and the communication and storage of redundant absorbing states.

In order to use absorbing states, we must size the centralized value function approximator to take as input all of the observations from *the absolute maximum* number of agents that can be active in the environment, regardless of how many are active at any given time. If the function approximator is a fully connected neural network, it must have inputs for all possible agents. In addition to adding sample complexity to the learning process, these extra parameters present an unnecessary computational overhead during training. Furthermore, in cases where the maximum number of agents is unknown, *e.g.,* when agents' actions can spawn additional agents, we must choose some arbitrarily large value for the maximum number of agents, exacerbating the sample complexity and computational overhead issues of the function approximator.

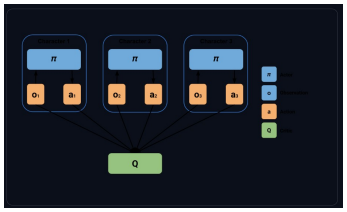
We must also consider the computational and resource overhead of the absorbing states themselves. Most implementations of absorbing states (*e.g.,* SMAC (Samvelyan et al. 2019)) add them as part of the state returned from the environment. This has the advantage of not requiring explicit knowledge of absorbing states by the algorithm implementation. However, as these absorbing states are treated in the same way as any other states, they must also be processed, stored, and communicated in the same way as other states. In distributed RL architectures which rely on distributed inference workers (Espeholt et al. 2018; Horgan et al. 2018), these states would need to be sent from these workers to the optimizer as part of trajectories, where they would introduce a communication overhead. In addition, they will exist in any buffers, queues, and, in the case of off-policy algorithms, replay stores, taking up unnecessary memory. These issues can be partially mitigated by moving the implementation of absorbing states from the environment to the algorithm (*e.g.,* padding states right before they are given as input to the centralized value function), at the cost of making the implementation of the algorithm more complex and less general across environments.

4 Methods

In this section, we propose a novel architecture for COMA (Foerster et al. 2018) called MA-POCA. MA-POCA uses self-attention (Vaswani et al. 2017) over *active* agents in the critic network, thereby addressing the issue of posthumous credit assignment without the need for absorbing states. Additionally, self-attention enables a network architecture that can efficiently compute counterfactual baselines for groups of homogeneous and heterogeneous agents. Note that, though the decentralized POMDP framework requires that the maximum number of agents N is known, the algorithm and network architecture of MA-POCA do not.

MA-POCA

MA-POCA learns a centralized value function to estimate the expected discounted return of the group of agents and a centralized agent-centric counterfactual baseline to achieve credit assignment in the manner of COMA. In architectures that use self-attention, it is common to have entity encoders which map entities to an embedding space before passing through the attention layer (Baker et al. 2020). In our setting, we consider *distinct* observation spaces as entities. For example, if two agents $i,\ j$ share the same observation space $\mathcal{O}_i = \mathcal{O}_j$, corresponding observations will be embedded with the *same encoder*. However, if $\mathcal{O}_i \neq \mathcal{O}_j$, we embed them with different encoders. Furthermore, we consider observations and observation-action pairs as separate entities;



shared critics and actors parameters

but the actor / policy networks that in observation that is relative to the actuating agent only

observation and actions are concatenated and then embedded.

Formally, let $g_i : \mathcal{O}_i \rightarrow E$ be an encoding network for observations $o_i \in \mathcal{O}_i$ where E is the embedding space. As stated previously, if $\mathcal{O}_i = \mathcal{O}_j$, then $g_i = g_j$.

MA-POCA Value Function

In this section, we discuss how to estimate the expected discounted return given in Eq. 1 for a group of agents wherein some may terminate early. Recall, in our setting, the number of active agents depends on t. Thus, let k_t denote the number of active agents at time step t such that $1 \leq k_t \leq N$ where N is the maximum number of agents that can be alive at any time.

In the MARL literature, there are generally two ways that centralized state or state-action value functions are conditioned on state:

- there is a separate vector containing global information (*i.e.,* the coordinates of all agents) that is unobserved by any individual agent (Foerster et al. 2018; Rashid et al. 2018);
- the joint observation of the agents $\mathbf{o_t} \in \mathcal{O}$ is used as this is a reasonable approximation to the global state (Long et al. 2020; Lowe et al. 2017).

It is also possible to use a hybrid. In this work, we only consider the joint observation of *active agents*, though the former would still require absorbing states or values in some capacity and can be treated by means similarly to what follows.

To handle a varying number of agents per timestep, we first encode the observations of all active agents $g_i(o_t^i)_{1 \leq i \leq k_t}$ and then pass the encodings through an RSA. The RSA block we use is architecturally similar to those used in the vanilla Transformer architecture (Vaswani et al. 2017) but without positional encodings (Baker et al. 2020). For more details on the self-attention mechanism, please see Appendix B. Then, the centralized state value function parameterized by ϕ has the form

how to train the shared critic

$$V_\phi(RSA(g_i(o_t^i)_{1 \leq i \leq k_t})) \tag{6}$$

and is trained with $TD(\lambda)$ (Sutton 1988)

$$J(\phi) = (V_\phi(RSA(g_i(o_t^i)_{1 \leq i \leq k_t})) - y^{(\lambda)})^2 \tag{7}$$

where

$$y^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

a little bit like using GAE and n-value bootstrapping

$$G_t^{(n)} = \sum_{l=1}^{n} \gamma^{l-1} r_{t+l} + \gamma^n V_\phi(RSA(g(o_{t+n}^i)_{1 \leq i \leq k_{t+n}})))$$

where k_{t+n} is the number of agents that are active at time $t+n$. Note, it is possible for k_{t+n} to be greater or less than k_t as any number of agents could have terminated early or been spawned at time step t. It is in this way that expected value from time $t+n$ may propagate to an agent that terminated at time t.

MA-POCA Counterfactual Baseline

Agents who try to maximize a shared reward function suffer from a credit assignment problem since it is hard to disentangle an agent's actions contributed to the group's return. (Foerster et al. 2018). Counterfactual baselines (Eq. 3) marginalize out the action of an individual agent in the centralized state-action value function enabling the computation of a shaped, per-agent advantage. Though the architecture used in the original implementation of COMA has a number of advantages, it is limited to problems with discrete actions and a fixed number of agents. In this work, we propose an alternative leveraging self-attention, that alleviates both of these constraints.

We consider observations and observation-action pairs to be distinct entities. Letting $f_i : \mathcal{O}_i \times \mathcal{A}_i \rightarrow E$ be an encoding network for observation-action pairs, the counterfactual baseline can be learned explicitly by estimating the expectation in Eq. 3 with Monte Carlo samples (Foerster et al. 2018). Thus, we can learn the counterfactual baseline for some agent j by learning a value function that is conditioned on the observation-action pairs of all agents i such that $1 \leq i \leq k_t$ $i \neq j$ but *only the observation* of agent j. Again using an RSA block and observation and observation-action entity encoders, the baseline parameterized by ψ for agent j has the form

how to train the shared baseline

the state value after paying attention to what everyone else does

$$Q_\psi(RSA(g_j(o_t^j), f_i(o_t^i, a_t^i)_{\substack{1 \leq i \leq k_t \\ i \neq j}}))$$

The objective for the baseline is

gj and fi acts as encoder

$$J(\psi) = (Q_\psi(RSA(g_j(o_t^j), f_i(o_t^i, a_t^i)_{\substack{1 \leq i \leq k_t \\ i \neq j}})) - y^{(\lambda)})^2$$

train the baseline network to be as close to the value of the state as possible (8)

which uses the *same target* $y^{(\lambda)}$ as the value function update in Eq. 7.

Note that a single joint observation $\boldsymbol{o} = (o^1, \ldots, o^N)$ (and action) generates up to N different samples on which to update Eq. 8, one for each j, $1 \leq j \leq N$. This is the key reason for using separate sets of parameters for the value function and baseline: in our training regime, the baseline is trained on the *permutations* of all agent observations to estimate the *per agent* baseline whereas the value function is not. This would mean a potential factor of N more samples used to compute the baseline versus the value function. We hypothesize that this would lead to baseline dominance and, experimentally, we found using separate networks to perform better.

Finally, the advantage for agent j to be used in the update in Eq. 5 is given by

how to trained the share policy

$$\text{Adv}_j = y^{(\lambda)} - Q_\psi(RSA(g_j(o_t^j), f_i(o_t^i, a_t^i)_{\substack{1 \leq i \leq k_t \\ i \neq j}}))$$

the advantage is the actual discounted return minus the baseline

5 Experiments

In this section, we evaluate MA-POCA empirically on four multi-agent environments and compare its performance to the state-of-the-art multi-agent algorithm COMA (Foerster et al. 2018) and the single-agent algorithm PPO (Schulman et al. 2017). Three of the environments are built using Unity's ML-Agents Toolkit (Juliani et al. 2020), and one

is taken from the Multi-Agent Particle Environments (Lowe et al. 2017). We choose to show the performance of PPO to illustrate that the environments require coordination to solve.

We show that, in standard cooperative tasks without dying or spawning, MA-POCA performs as well as or slightly better than COMA and that both outperform PPO. Furthermore, we show that MA-POCA significantly outperforms both baselines in tasks where agents die and/or spawn. Note, we developed our own environments due to the lack of existing environments with these features. Code for all algorithms and environments is available.[1]

Algorithm and Baselines

Architecturally, the implementation of COMA we use is similar to what is proposed in Section 4. The key difference being the use of an RSA block. Instead, all inputs are concatenated and fed into a fully connected neural network. In the case of agents that have terminated early or have not yet spawned, we use an absorbing state of all zeroes.

In both MA-POCA and COMA, we use separate networks to approximate the value function and baseline as per the discussion in Section 4. Note, this is not a problem with the original COMA implementation (Foerster et al. 2018) because their architecture depends on the assumption of strictly discrete actions which we do not make. However, the COMA architecture we use was suggested in the original work (Foerster et al. 2018).

We generate targets for the value function and baseline updates in Eqs. 7 and 8 and analogues for COMA using $TD(\lambda)$ (Sutton 1988) as done in (Foerster et al. 2018). We use the value function and baseline network to compute the advantage for the policy update as in Eq. 6. However, we do not use a target value function (Mnih et al. 2015) but instead use trust region clipping (Schulman et al. 2017) for the value function, baseline and policy updates which we found to work better in practice.

We use the implementation of Proximal Policy Optimization (PPO) (Schulman et al. 2017), and generalized advantage estimation (GAE) (Schulman et al. 2016) contained in the Unity ML-Agents Toolkit (Juliani et al. 2020).

Results

A brief description of the environments is provided in Figure 2; further details can be found in Appendix C. Figure 3 compares the mean and 95% confidence interval of episodic reward for MA-POCA, COMA, and PPO over 10 seeds each. Hyperparameters for all algorithms and experiments are contained in Appendix D.

In all four environments, PPO is unable to find the optimal policies and converges to a local optima. This is likely due to the partial observability introduced by exclusively decentralized training and acting. In (c) and (d), PPO has no mechanism to address the posthumous credit assignment problem so it is unable to learn from value beyond it's termination. For example, in (d) Dungeon Escape, the PPO agents are able to use the key when they observe it–however, they are not able to learn that killing the green dragon is the way to get the key as this removes them from the environment. Thus, they solve the problem roughly half the time when they accidentally collide with the green dragon. Also, note that, the reward is decreasing indicating that they are learning to avoid the green dragon (possibly, to hold out for the key when it drops).

The curves in (a) and (b) in the top row of Figure 3 contain results for the Collaborative Push Block and Simple Spreader environments which do not contain spawning or dying agents. In these environments, MA-POCA learns slightly faster than COMA. We hypothesize that the permutation invariance of attention gives MA-POCA an advantage as COMA's value network needs to learn that any permutation of a joint observation has the same value. Additionally, the cross-comparison of entities in attention may enable more robust modeling of the group value function.

The curves in (c) and (d) in the bottom row of Figure 3 contain results for the Baton Pass and Dungeon Escape environments which do contain spawning and/or dying agents. In these environments, MA-POCA significantly outperforms COMA with less variance between seeds. COMA eventually converges to the optimal policy. Of particular interest is that both PPO and MA-POCA are faster initially than COMA. We hypothesize COMA's inferior sample complexity is due to the inefficient input representation that absorbing states provide, as discussed in Section 3.

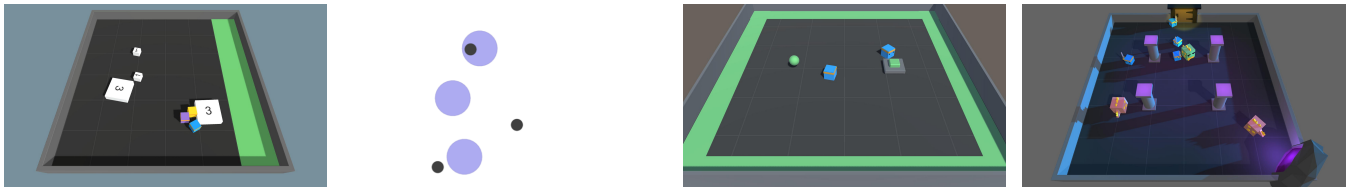
6 Related Work

Group-Centric MARL Value decomposition methods (Sunehag et al. 2017; Rashid et al. 2018; Iqbal et al. 2021) are an alternative to the counterfactual baseline used by COMA and MA-POCA to address multi-agent credit assignment. Value decomposition makes the assumption that the group value function is a monotonic function of the individual's value functions and thus assumes actions are independent which is not true in general.

Varying Agents in MARL Actor-Attention-Critic (Iqbal and Sha 2019) uses an attention-based state-action value function for each agent to estimate an individual's expected reward conditioned on the state of other agents, though the number of agents remains constant per episode. Similarly, Graph Policy Learning (Rahman et al. 2021) uses a graph neural network to condition an agent's value function and policy on the state of a varying number of teammates. However, as both approaches depend on the individual agent to condition the value function, they do not address posthumous rewards without absorbing states. Evolutionary Population Curriculum (Long et al. 2020) uses attention to handle a population of agents increasing in size but also uses distinct value function for each agent and so would also require absorbing states for posthumous credit assignment. Randomized Entity-Wise Factorization (Iqbal et al. 2021) applies attention dynamically group agents, but does not address a variable number of total agents.

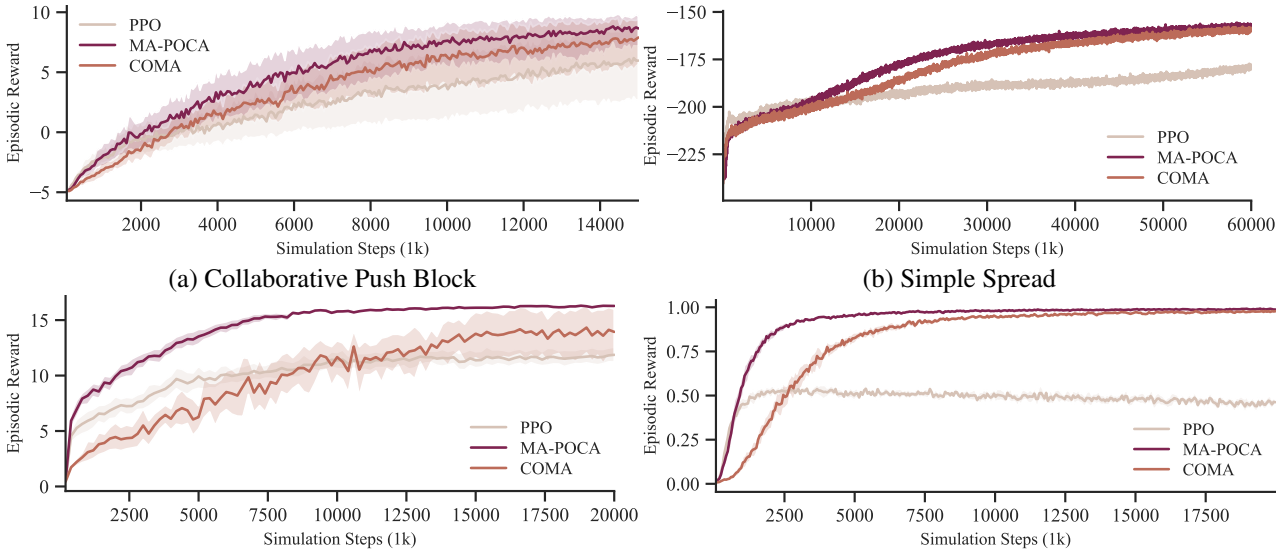
Environment Implementations The commonly used StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al.

[1] https://github.com/Unity-Technologies/paper-ml-agents/tree/main/ma-poca



(a) (b) (c) (d)

Figure 2: **(a) Collaborative Push Block**. Agents (blue, yellow, purple) must push white blocks to green area; larger blocks require more agents to push. **(b) Simple Spread**. Agents (large circles) must move to cover targets (small circles) without colliding with one another. **(c) Baton Pass**. Blue agents must grab green food and hit green button to spawn another agent, who can grab the next food, and so on. **(d) Dungeon Escape**. Blue agents must kill green dragon by sacrificing one of them to reveal a key. Teammates must pick up key and reach the door, while avoiding pink dragons.



(a) Collaborative Push Block (b) Simple Spread

(c) Baton Pass (d) Dungeon Escape

Figure 3: Comparison between MA-POCA, COMA, and PPO of cumulative reward per episode for **(a)** Collaborative Push Block, **(b)** Simple Spread, **(c)** Baton Pass, **(d)** Dungeon Escape. Results are averaged across 10 seeds. MA-POCA outperforms COMA in all four environments, and significantly so in environments that involve agents spawning or dying (**(c)** and **(d)**). PPO converges to a sub-optimal policy in all tasks.

2019) benchmarks return all zeros as an absorbing state for units that have died. Death masking (Yu et al. 2021) is a variant of absorbing state which appends an agent ID to a vector of all zeros. This is an instance of using different absorbing states for different outcomes as discussed in Section 3.

Support for Varying Agents To the author's knowledge, there are three APIs with explicit support for dying and spawning agents: PettingZoo (Terry et al. 2020), RLLib (Liang et al. 2018), and the Unity ML-Agents Toolkit (Juliani et al. 2020). RLLib also contains implementations of MARL algorithms though all would require absorbing states to be used with varying numbers of agents.

7 Conclusion

This paper explicitly identified the Posthumous Credit Assignment problem created when agents terminate early. The is currently handled in MARL by adding an absorbing state for agents that terminate early. Using a toy supervised learning problem, we empirically demonstrated a downside of using absorbing states. We then introduced MA-POCA, a novel architecture designed to train groups of agents to solve tasks in which individual agents may terminate early or spawn, without the use of absorbing states. MA-POCA naturally handles varying numbers of agents and achieves a counterfactual baseline via the use of self-attention. Finally, we demonstrate that MA-POCA outperforms COMA and PPO on two standard MARL tasks without dying or spawning agents. More importantly, MA-POCA significantly outperforms both on tasks with dying and spawning agents. Future work will extend other algorithms in the decentralized POMDP framework beyond absorbing states and investigate potential formalisms for problems where the maximum number of agents N is unknown.

References

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. arXiv:1607.06450.

Baker, B.; Kanitscheider, I.; Markov, T.; Wu, Y.; Powell, G.; McGrew, B.; and Mordatch, I. 2020. Emergent Tool Use From Multi-Agent Autocurricula. In *International Conference on Learning Representations*.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Yotam, B.; Vlad, F.; Tim, H.; Dunning, I.; Legg, S.; and Kavukcuoglu, K. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning*, volume 4, 2263–2284. ISBN 9781510867963.

Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2018. Counterfactual Multi-Agent Policy Gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Horgan, D.; Quan, J.; Budden, D.; Barth-maron, G.; Hessel, M.; van Hasselt, H.; and Silver, D. 2018. Distributed Prioritized Experience Replay. In *International Conference on Learning Representations*, 1–19.

Iqbal, S.; Schroeder de Witt, C.; Peng, B.; Böhmer, W.; Whiteson, S.; and Sha, F. 2021. Randomized Entity-Wise Factorization for Multi-Agent Reinforcement Learning. In *Proceedings of the 38th International Conference on Machine Learning*.

Iqbal, S.; and Sha, F. 2019. Actor-Attention-Critic for Multi-Agent Reinforcement Learning. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 2961–2970. Long Beach, California, USA: PMLR.

Juliani, A.; Berges, V.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; and Lange, D. 2020. Unity: A General Platform for Intelligent Agents. *arXiv preprint*, abs/1809.02627.

Konda, V.; and Tsitsiklis, J. 2000. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*.

Liang, E.; Liaw, R.; Moritz, P.; Nishihara, R.; Fox, R.; Goldberg, K.; Gonzalez, J. E.; Jordan, M. I.; and Stoica, I. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. *arXiv preprint arXiv:1712.09381*.

Littman, M. 1994. Markov-games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning*, 157–163.

Long, Q.; Zhou, Z.; Gupta, A.; Fang, F.; Wu, Y.; and Wang, X. 2020. Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning. In *International Conference on Learning Representations*.

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; and Mordatch, I. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Neural Information Processing Systems (NIPS)*.

Lyu, X.; Xiao, Y.; Daley, B.; and Amato, C. 2021. Contrasting Centralized and Decentralized Critics in Mult-Agent Reinforcement Learning. *arXiv preprint arXiv: 2102.04402*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529.

Oliehoek, F. A.; and Amato, C. 2016. *A Concise Introduction to Decentralized POMDPs*. Springer.

Rahman, M. A.; Hopner, N.; Christianos, F.; and Albrecht, S. V. 2021. Towards Open Ad Hoc Teamwork Using Graph-based Policy Learning. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 8776–8786. PMLR.

Rashid, T.; Samvelyan, M.; Schroeder de Witt, C.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. QMIX: Monotonic Value Function Factprisation for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*.

Samvelyan, M.; Rashid, T.; de Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G. J.; Hung, C.-M.; Torr, P. H. S.; Foerster, J.; and Whiteson, S. 2019. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M. I.; and Abbeel, P. 2016. High-Dimensional Continuous Control Using Generalzied Advantage Estimation. In *International Conference on Learning Representations*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint*, abs/1707.06347.

Sunehag, P.; Lever, G.; Gruslys, A.; Marian Czarnecki, W.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Z. Leibo, J.; Tuyls, K.; and Graepel, T. 2017. Value-Decomposition Networks For Cooperative Multi-Agent Learning. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*.

Sutton, R. S. 1988. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1): 9–44.

Terry, J. K.; Black, B.; Grammel, N.; Jayakumar, M.; Hari, A.; Sulivan, R.; Santos, L.; Perez, R.; Horsch, C.; Dieffendahl, C.; Williams, N. L.; Lokesh, Y.; Sullivan, R.; and Ravi, P. 2020. PettingZoo: Gym for Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2009.14471*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems*, 5999–6009.

Wolpert, D.; and Tumer, K. 2002. Optimal Payoff Functions for Members of Collectives. In *Modeling Complexity in Economic and Social Systems*, 355–369. World Scientific.

Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; and Bayen, A. 2021. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. *arXiv preprint arXiv:2103.01955*.

A Additional Experiments for Mean Computation

In this section, we provide additional figures for the different values of $o_{abs} = -1.0,\ 1.0,\ 0.4$ showing that -1.0 and 1.0 are reasonable choices (see Figure 4) but 0.4 is not as it is contained in $[0.25, 0.75]$ (see Figure 5). Additionally, we provide a figure for the curves when we fix the number of absorbing states (see Figure 6) to demonstrate that varying the number of floats per sample is more challenging. All hyperparameters are contained in Appendix D and are the same as the experiment discussed in the main text. In each figure, curves are the mean and 95% confidence interval over 20 seeds.

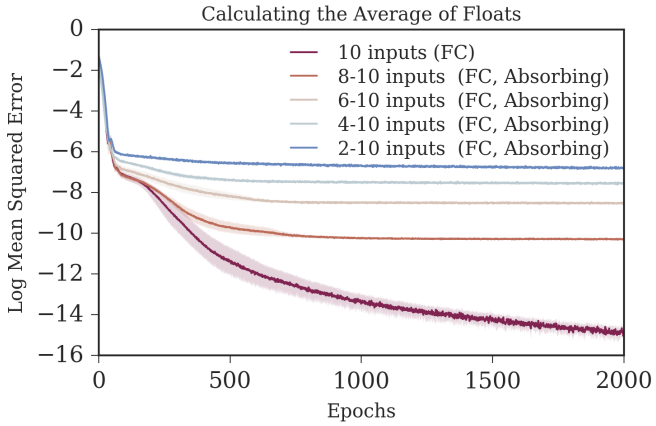


Figure 5: $o_{abs} = 0.4$. Since $o_{abs} \in [0.25, 0.75]$, this problem is partially observable and asymptotically the network cannot converge to solve the task.

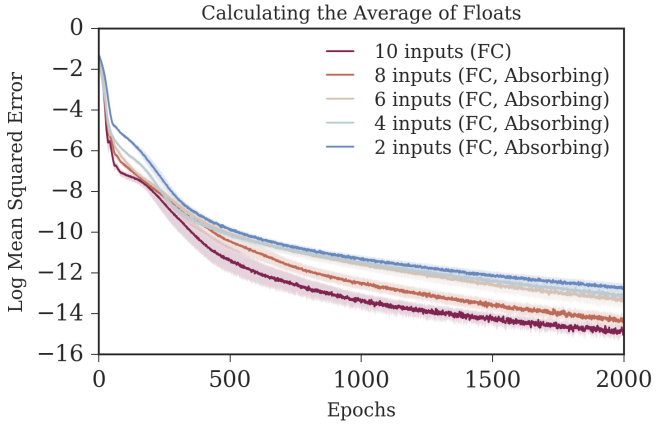


Figure 6: The number of absorbing states per sample is fixed but still shuffled. This version of the task is significantly simpler than when the number of absorbing states is varied per sample.

B Self-Attention

MA-POCA's centralized critic uses an RSA module in order to process a variable number of agents. The agent's observations are first embedded using a fully connected layer (Baker et al. 2020). Each agent's observation embedding is normalized using Layer Normalization (Ba, Kiros, and Hinton 2016) and then further embedded into Query : Q, Key : K and Value: V using a fully connected network. Q, K and V are fed into a scaled dot-product multi-head attention (Vaswani et al. 2017). The original observation embeddings are summed with the processed embeddings (the residual connection) and normalized again with Layer Normalization. The resulting embeddings are then averaged together to form a fixed size embedding. The same attention mechanism is used in the calculation of average experiments shown in Figure 1.

C Environments

Cooperative Push Block. In this environment, three agents must push 5 blocks of various size into a goal that is at the edge of a square stage. At the beginning of each episode, the blocks, agents, and goals are randomly placed in the stage. When one of the blocks hits the goal, all the agents receive a group reward corresponding to the size of the block. Small blocks are +1, medium +2, and large +3. Large blocks require all three agents to push together to move at any reasonable speed, 2 blocks require two agents to collaborate, and small blocks can be pushed by a single agent. The episode ends when all blocks are pushed into the goal or when 1000 steps have been taken by the agents. A small time penalty of -0.0025 per timestep is given to the agents to encourage them to finish quickly. Agents' observe by casting 21 rays in a 180° arc front of them, similar to a LIDAR. An agent is given the distance to an object that the ray collides with, as well as if it is an agent, a wall, the goal, or the type of block. Two sets of rays are given, one that is high enough to see the walls and goal over the blocks and agents, and one that is at agent-level.

Dungeon Escape. This environment contains five agents, a dragon holding a key (green), two dragons that don't hold a key (pink), a portal, and a door in a square stage. At the beginning of each episode, the agents, the dragons, and the door and portal are randomly placed in the stage. The agents' receive a +1 group reward if at least one of them to exit the stage through the door. To do so, this agent must first have a key, which is dropped by the green dragon. In order to slay the dragon and make it drop the key, an agent needs to run into it, which will cause both the agent and the dragon to be removed from the environment. The green dragon slowly moves towards the portal, while the pink dragons will attempt to eat the closest agent. The episode ends if the green dragon reaches the portal or an agent with the key reaches the door. In order for the group of agents to receive a reward, at least one of the agents *must learn to sacrifice itself* so that another can grab the key and go through the door, and one or more of them must learn to distract the pink dragons so that they do not attack the key-holding agent. Similarly to Cooperative Push Block, agents' observe by casting 15 rays

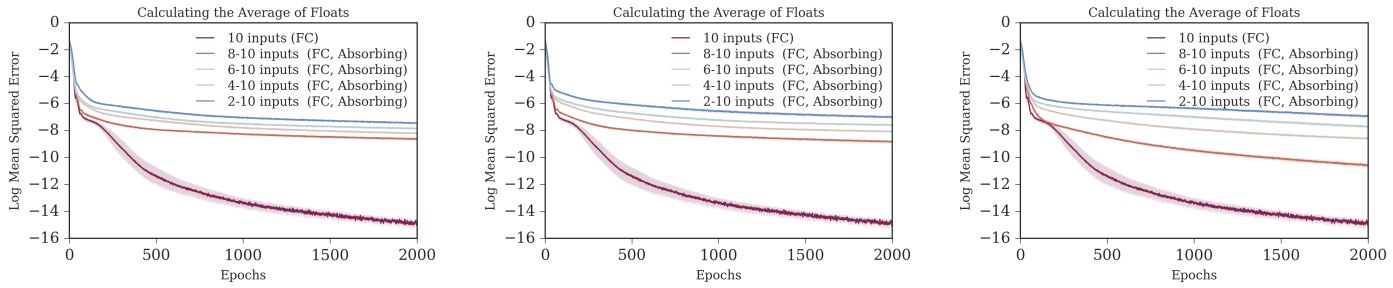


Figure 4: The sample efficiency of learning to compute the mean of a varying number of floats depends on the quantity of absorbing states. The performance for $o_{abs} = 0.0 - 1.0, 1.0$ are roughly the same and asymptotically all converge. Finally, without absorbing states (10 inputs (FC)) the task is trivial.

in a 120° arc front of them. Unlike Cooperative Push Block, only one set of rays are used as there are less elements to obstruct them.

Baton Pass. In this environment, a single agent is spawned along side an orb and a button. The button can only be pressed if the orb has already been collected. Pressing the button will create a new orb and spawn a new agent. Only the most recently spawned agent can collect the orb or press the button. This forces each agent to first collect the orb, then press the button and finally stay out of the way of the newly spawned agents. Indeed the agents are large and can block each other. Every time an orb is collected the whole group gets a +1 reward, the game ends when 20 orbs have been collected. Each agent also has the possibility to be despawned by touching an exit zone. Doing so will not grant any rewards or penalties but will free available space for other agents. At each time step, the agents will receive a penalty of 0.000125 times the number of currently existing agents. This is to force the agents to finish the task faster and encourage agents to despawn once they can no longer collect orbs or press the button. The agents perceive the environment with one set of 13 raycasts plus information about their velocity and their capacity to press the button or collect orbs. The agents can move forward, backwards and rotate.

Simple Spread. This environment is taken directly from set of Multi-Agent Particle Environments created by (Lowe et al. 2017). Agents are rewarded based on the minimum distance between each landmark (shown as dots) and any agent. Agents are penalized if they collide with other agents; the optimal strategy is to cover all the dots without colliding with each other. Agents observe the position of themselves, their teammates, and all 3 landmarks. All rewards are given as a group reward for the entire team of 3 agents.

D Hyperparameters

Common Hyperparameters for Reinforcement Learning Tasks

Table 1 shows the hyperparameters used for all environments. As our implementations of PPO, MA-POCA and COMA all use the same policy and value clipping and entropy bonus mechanisms, all hyperparameters apply to all three algorithms. Network size parameters apply to both the critic and policy. In PPO, λ is used for GAE whereas in MA-POCA and COMA it is used for $TD(\lambda)$.

Hyperparameter	Dungeon Escape Baton Pass Push Block	Simple Spread
Minibatch Size	1024	512
Buffer Size	10240	5120
Epochs per Update	3	3
Learning Rate	0.0003	0.0003
Optimizer	Adam	Adam
Entropy Bonus β	0.01	0.01
Clip Ratio ϵ	0.2	0.2
λ	0.95	0.95
Discount Factor γ	0.99	0.99
Hidden Units	256	128
Fully Connected Layers	2	2
Attention Entity Embedding Size*	256	128
Attention Entity Embedding Layers*	1	1
Number of Attention Heads*	4	4

Table 1: Hyperparameters for all algorithms for Collaborative Push Block, Baton Pass and Dungeon Escape (under the General Case column) and for Simple Spread. The hyperparameters marked with an * only apply to MA-POCA since PPO and COMA do not use an attention module.

Hyperparameters for Computing the Mean

For the fully connected network and for the self-attention networks. Hyperparameters that do not apply are denoted with a " / ".

Hyperparameter	Fully Connected	Attention
Minibatch Size	500	500
Learning Rate	0.001	0.001
Optimizer	Adam	Adam
Hidden Units	32	/
Layers	2	/
Entity Embedding Size	/	32
Entity Embedding Layers	/	1
Number of Attention Heads	/	4

Table 2: Hyperparameters for fully connected network and for self-attention network for the Compute the Mean task