National Technical University of Athens
School of Electrical and Computer Engineering
Division of

Reinforcement in Cooperative Games

Deep Learning Approaches

Diploma Thesis

of

KONSTANTINOS BARDIS

Supervisor: Stefanos Kollias
Professor

Athens, April 2022

National Technical University of Athens
School of Electrical and Computer Engineering
Division of

Reinforcement in Cooperative Games

Deep Learning Approaches

Diploma Thesis

of

KONSTANTINOS BARDIS

Supervisor: Stefanos Kollias
Professor

Approved by the examination committee on .

(Signature)	*(Signature)*	*(Signature)*
.....	.....	.....
Stefanos Kollias	Giorgos Stafilopatis	Girgos Stamou
Professor	Professor	Professor

Athens, April 2022

National Technical University of Athens
School of Electrical and Computer Engineering
Division of

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

(Signature)

.....
Konstantinos Bardis

Abstract

A Multi-Agent system is a system which necessitates the coordination and interaction between several decision-making entities (Agents) to accomplish a given task they otherwise would not be able to. There is a growing need of algorithms tailored to this setting, since modern systems are relying more and more on decentralized cooperation between several agents, which poses several additional challenges over the more well-studied single agent setting, like the non-stationarity induced by other agents' decisions, which behooves some important modifications to existing algorithms or the development of dedicated approaches from the ground-up. This diploma thesis aims to first study the modern literature on MARL, explain the challenges and opportunities it affords, and then to utilize several of these algorithms in multi-agent settings using the libraries PettingZoo and RLLib in cooperative settings.

Αυτή η Πτυχιακή Εργασία απασκοπεί στο να μελετήσει συστήματα πολλών πρακτόρων οι οποίοι επικοινωνούν και μαθαίνουν μέσω βαθειάς ενισχυτικής μάθησης προκειμένου να πετύχουν να φέρουν εις πέραν εργασίες που κανένας πράκτορας από μόνος του δεν θα μπορούσε. Τα τελευταία χρόνια, με την άνοδο της Τεχνητής Νοημοσύνης και την μεγαλύτερη εξάρτηση των σύγχρονων συστημάτων από αυτοματοποιημένες διαδικασίες και αλγορίθμους, κρίνεται όλο και πιο έντονη η ανάπτυξη συστημάτων που θα επιτρέπουν την αποκεντρωμένη, αποτελεσματική επικοινωνία και συνεργασία μεταξύ πρακτόρων σε πραγματικές συνθήκες. Η ιδιαιτερότητα της ανάγκης συντονισμού μεταξύ πολλών δρώντων καθιστά το πεδίο αυτό πιο πολύπλοκο σε σχέση με το πιο καλά μελετημένο πλαίσιο του ατομικού δράστη, και απαιτεί επιπρόσθετους, πιο εξειδικευμένους αλγόριθμους και προσεγγίσεις για την επίτευξη των στόχων του. Σε αυτήν την εργασία λοιπόν θα μελετήσουμε αρχικά τις προκλήσεις αλλά και τις ευκαιρίες που δίνει η βαθειά ενισχυτική μάθηση, ένα μέρος της σύγχρονης βιβλιογραφίας για τις προσεγγίσεις που θεωρούνται οι πλέον αποτελεσματικές και τέλος θα κάνουμε μια συγκριτική μελέτη μερικών εξάυτών με την βοήθεια των βιβλιοθηκών PettingZoo και RLLib.

Keywords

Multi-Agent, Reinforcement Learning, Neural Networks, RLLib, PettingZoo, Gym

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multi-Agent Reinforcement Learning (MARL) is a discipline that concerns itself with the study and development of systems where several agents interact with each other, either in a cooperative, adversarial or mixed manner, and learn optimal behaviors via trial and error, effectively serving as an extension to the more well-developed single agent optimal control problem.

There are several significant impediments to the progress of MARL. To begin with, in contrast to single agent situations where it usually is relatively easy to establish an overarching objective for the agent, typically to maximize some score or to minimize some cost, in multi-agent settings it is not so, since it can be quite unclear what to formalize as the overall multi-agent learning problem, as the returns of the agents can be correlated and incapable of being maximized independently, with two issues standing out: Stability of the learning process and consistent adaptation to the actions of the rest of the agents [6]. These two concepts are central to the very premise of MARL, which in contrast to more classical optimal control methods, can enable agents to adapt to unforeseen circumstances and learn in an online manner.

In addition, another fundamental problem inherent in MARL is the issue of non-stationarity of the environment from the perspective of each individual agent, as it changes due to the actions of the other agents, outside of the individual's control, eliminating formal convergence guarantees ([15], [16]). This non-stationarity creates problems for both value-based methods, such as Q-learning and its variants, and policy gradient methods, like PPO and ACTKR [60]; It precludes the former from utilizing experience replay in the usual manner, while it leads to unacceptably high variance in the latter [28]. Nevertheless, [31] empirically provide good evidence that such independent approaches can usually be effective in practice, an observation we will ourselves exploit in our experimental section.

Furthermore, multi-agent settings exacerbate the Curse of Dimensionality, since not only do individual agents have to deal with potentially high (or infinite) dimensional action and state spaces, as in Atari or Robotic locomotion problems, but also the growth of state and action variables as well as the number of agents can be exponential [6], calling for more sophisticated ways of circumventing that issue.

The fact there are several agents interacting with each other also intensifies the problem of learning under partially observable environments, also occasionally present in single agent settings yet lesser in intensity [7]. To more effectively model this problem and

enable the agents to learn, extensions of the standard Markov Decision Process (MDP) have been proposed, the main among them being Partially Observable Markov Decision Process (POMDP) [36] and then Dec-POMDP for decentralized, multi-agent settings. These will be discussed in more detail in the upcoming sections, along with popular algorithmic approaches to model them effectively, as they are the conceptual basis of MARL problems.

Credit assignment [33] is another issue central to reinforcement learning, even in single agent settings. It can defined as the problem of figuring out an action's actual influence towards future rewards, ringing notions of causality [32]. It is an issue that inevitably comes up whenever an agent interacts sequentially with an environment and receives rewards sparsely, wherein usually after a long trajectory of actions, rewards and observations, it often is not clear to what extent each of these actions contributed to the observed outcomes. Obviously, this problem is only more complicated in the MARL setting, particularly in cooperative settings with global rewards ([52], [44]).

Another issue befalling MARL research was, and to an extend still is, the shortage of standardized APIs for conducting reproducible research, unlike with the single Agent case, substantially increasing the engineering workload required to reuse existing code for research purposes. In fact, as per [54] while around 69 pip-installable libraries exist for the single agent formulation, only 5 MARL libraries with a significant number of customary users exist. In fact, we will be using the library provided by [54] called PettingZoo in our experiments, since it provides a convenient API supported by a formally sound way to represent MARL environments, a challenge to properly program in itself.

Notwithstanding that multitude of challenges, MARL is on track to becoming one of the most influential disciplines of machine learning, particularly as the reliance on autonomous AI systems increases. We can already witness some exciting areas that stand to be benefited strongly from the continuous maturation and adoption of MARL, a select few of which we will succinctly discuss.

Robotics seems like a natural application area for MARL, and with good reason; The ability of agents to adapt on the fly against ever changing and partially observable conditions is a central selling point of modern MARL algorithms, compared to planning methods which often require complete knowledge of the environment beforehand. One basic task is that of navigation, where a robot or team of robots must traverse a path between a given starting point and a destination, even one that is not fixed, under the constraint that it evades obstacles and potentially harmful interactions with other agents or people [19].

A related task is that of area sweeping tasks, when the agents have to comprehensively survey a given territory towards, for example, search and rescue missions, exploration or retrieval of otherwise inaccessible objects [30]. Other interesting robotics applications involve 'pursuit'- type of tasks, with some agents acting as "predators" attempting to capture some "prey" agents ([21], [22]) and "Multi-target observation" as an extension of the aforementioned exploration task wherein several agents have to keep track of a target simultaneously ([57], [14]).

Closely related to robotics are applications in UAVs (Unmanned Aerial Vehicles). For instance, [41] approximate a Nash Correlated equilibrium with a modified multi-agent

Q-learning algorithm for the task of optimal sensing coverage of an unknown field, while careful to minimize the overlapping sections of their field of views, a particularly useful objective for search and rescue operations. [43] utilized MADDPG, a multi-version variant of DDPG, to attack the multi-UAV target assignment and path planning (MUTAPP) problem and allow teams of drones to avoid obstacles and reach target areas in dynamic environments while avoiding collisions in real time.

Of course, a wide variety of applications exist outside of robotics-related domains. [5] utilize a modified version of a popular actor-critic algorithm (A2C) to rapidly identify and resolve conflicts between aircraft in high-density, stochastic and dynamic intersection and merging points with extremely high success rates (99.97 % and 100 % respectively). [45] utilize independent PPO in a MARL setting to control the energy exchanges in a factory, and compared it to a rule-based control strategy, with the MARL system enabling much faster reactions and often better performance.

[42] developed a MARL algorithm to regulate the energy exchange between a community of buildings, modelling each one as an agent and improving the energy status of the community as a whole. MARL has even been used in studying social behaviors in matrix games such as Prisoner's Dilemma [23] with experiments showing how conflict arises over the fight for common and scarce resources.

Besides the fantastically varied applications, MARL is also interesting because of its connection to Hierarchical Reinforcement Learning (HRL): More specifically, as in the case of Feudal RL [13], HRL can be interpreted as a special case of a multi-agent system, where the agents correspond to the multiple levels of the hierarchy. This formulation can aid in the design and implementation of even more sophisticated HRL architectures.

Having discussed the main challenges inherent in MARL, along with some opportune domains where it can truly shine, the structure of the Dissertation is the following: We will first present a comprehensive literature review of prior work in algorithmic approaches in MARL problems, with special emphasis on the domain of video games, which are considered among the most challenging and fruitful testsbeds due to the variety of skills they demand for success. In the next section we will analyze the framework of Dec-POMPDs and Stochastic Markov Games our algorithms usually operate under, and describe the methods we will use in our experiments. Our empirical section follows next, where we use said algorithms in a variety of environments and compare their performance, and lastly we cap off with directions for future research.

Chapter 2

Related Work

Several algorithms have been proposed in the last few years to deal with the problem of Multi-agent learning. Some of them are novel, explicitly designed for Multi-agent settings like COMA [16], while others are slightly modified versions of standard single-agent algorithms. In fact, as mentioned in the prior introductory section, there has been a number of successful applications using even vanilla single-agent algorithms for independent learning, making it a valid approach in practice.

There are two primary ways of categorizing MARL algorithms: Those that learn under a centralized manner and those that learn in a decentralized one. Centralized methods [10] adopt the framework of cooperative games [39] and directly augment the single-agent algorithms, enabling them to learn a single, "centralized" policy to produce the joint actions of all agents simultaneously. Decentralized methods on the other hand use the standard competitive formulation of a Markov Game by [27], and every agent optimizes their reward signal independently. That formulation, while potentially sufficient for some general-sum games, still is vulnerable to instability even in simple matrix games [15].

Nonetheless, more recent work lies in between those two antithetical approaches attempting to combine their strengths and mitigate their weaknesses: *Centralized Training and Decentralized Execution* (CTDE) and *Value Decomposition* (VD). CTDE improves upon Decentralized training by learning a centralized critic and adopting an actor-critic structure. VD joins all agents' individual Q-functions into a joint Q-function ([44], [52]), and has been considered as a de facto standard in MARL literature.

COMA [15], one of the most influential MARL algorithms, promotes the use of a centralized critic during training to estimate the Q-function by utilizing the joint action and full information available on the state, and decentralized actors to optimize each agent's policy, while conditioned solely on the action-observation history of each. It also incorporates a "counterfactual baseline", that uses the centralized critic to calculate an advantage function which is able to marginalize out a single agent's action while keeping the other agents' actions fixed, and compare that with the estimated return for the current joint action. This helps solve the multi-agent credit assignment problem, as only actions that immediately affect an agent's rewards are encouraged by the advantage function. COMA managed to significantly outperform other MARL actor-critic methods in the extremely challenging *StarCraft unit micromanagement* environment, known for its high stochasticity, high dimensional state-action space and delayed rewards.

Another algorithm following the CTDE paradigm was proposed in [28], dubbed MADDPG, essentially a multi-agent variant of DDPG (Deep Deterministic Policy Gradient). It works as an extension of the single-agent variant, by augmenting the critic with the extra information about the policies of other agents, with the actors confined to local awareness. This allows the policies to use extra information during training, but not at test time, enabling faster and more stable learning. MADDPG learns an augmented critic for every agent, does not assume a differentiable model of environment dynamics nor any structure on the inter-agent communication. They further train an ensemble of K different policies for each agent that cumulatively increases the robustness of the overall multi-agent system. Various experiments they conducted with Open AI's Multi-particle environment showed promising results, in both competitive and cooperative environments indicating a degree of generalization ability, though it is known to often have difficulty adapting in other environments, limiting its utility.

Incidentally, the authors of MADDPG note that a potential avenue for improving on the method's scalability, specifically the linear growth of the input space $\mathcal{Q}$ with N, the number of agents, would be to modify the Q-function to only consider agents in a certain neighborhood of an given agent. Mean Field Multi-Agent learning, proposed in [61], builds exactly on that idea. They approximate the interactions within a population of agents by the interaction of each agent with the average effect from the neighboring agents.To ensure that learning the Q-function is feasible even as the dimension of the joint action a grows with the number of agents N, they factorize it using only pairwise local interactions:

$$\mathcal{Q}^j(s, \mathbf{a}) = \frac{1}{N^j} \sum_{k \in N(j)} \mathcal{Q}^j(s, a^j, a^k) \tag{2.1}$$

This pairwise interaction can be approximated using the mean field theory developed by [51], and effectively allows the approximation of the Q-function independent of the number of agents,transforming a many-body problem to a two-body one. This decomposition also ameliorates the noise accumulation caused by the exploration of several agents, a problem not addressed in approaches with centralized critic Q-functions, as in COMA or MADDPG.

PPO [49] is considered a state-of-the-art algorithm for single agent problems. It is no surprise then that [63] attempt to adapt it to multi-agent problems, particularly cooperative games. MAPPO then follows the algorithmic structure of PPO by learning a policy π_θ and a value function $V_\phi(s)$ for each agent, which is only used during training for variance reduction, taking advantage of global information. Several implementation details were deemed critical to the success of the algorithm, such as Generalized Advantage Estimation (GAE) [48] with value normalization to stabilize value learning, incorporating agent-specific features in the global state under the condition that the state dimension does not explode, avoid mini-batching and too many epochs, tune the clipping ration ϵ, which balances stability and convergence speed, and use zero states with agent ID as the value input for dead agents.

A major selling point of the algorithm is that while it required minimal hyper-parameter tuning, no domain-specific algorithmic tweaks and could be trained even in a single desk-

top machine, it still managed to attain superior performance to other powerful algorithms in StarCraft, MPE and Hanabi domains. In fact, despite being an on-policy algorithm, which are known for their sample inefficiency, in most scenarios it achieved off-policy sample efficiency, thus enabling it to be trained in a single machine.The authors do however note that there are several limitations, such as using only discrete states, exclusively cooperative environments, and generally homogeneous agents.

As mentioned, MAPPO handles the issue of agent death via "death masking", a variant of an "absorbing state" which is a special, terminal state in a fully connected layer, where inactive agents (whether by addition or removal) are placed, irrespective of action choice, until the group of agents or episode terminates. However, using such states introduces significant problems as they not only complicate the training process of the function approximators but also waste quite substantial computational resources for agents that by definition do not affect the environment. That also gives rise to the problem of "Posthumous Credit Assignment", a situation when an agent must learn to maximize rewards it cannot experience. [11] propose a novel algorithm, dubbed MA-POCA, especially to deal with those issues.

MA-POCA proposes using a novel algorithm, based on the CTDE framework, where the centralized critic can handle a varying number of agents per time step by applying a self-attention mechanism [58] to only the active agents, which allows the critic to compute the future expected value of the group of states without requiring the use of absorbing states, and to implement counterfactual baselines [16] for both discrete and continuous action spaces for homogeneous or heterogeneous agents, both explicit limitations of MAPPO, thus allowing for the creation of new agents during the episode. MA-POCA is then shown to outperform both COMA and Independent PPO in a variety of environments, several of which were created using the Unity game engine specifically to that end.

[20] Also used an attention mechanism when learning a centralized critic, to enable more efficient and scalable learning based on the idea that in many environments and in the real world, it would be far more beneficial for an agent to actively choose which other agents to pay attention to at every time-step instead of just attending to everyone in the vicinity. Their approach is applicable in cooperative, competitive and mixed environments, scales linearly with the number of agents, can train policies with any reward scheme and different action spaces per agent, incorporates a variance reducing baseline and a set of centralized critics that dynamically attend to relevant information for each agent at each time step, thus making for a very versatile and scalable approach.

While the CTDE framework may be conveniently realizable in actor-critic and policy gradient settings, like in COMA, it is not at all straightforward in value-based methods such as Q-Learning, as already mentioned in the introduction, in part due to exponential growth in the combined action and observation spaces. One approach put forth by [52] for cooperative games trains individual agents with a novel additive value-decomposition network (VDN) such that the team value function is linearly decomposed into agent-wise value functions by back-propagating the total gradient Q through the neural networks representing those individual component value functions. The main assumption they make is that the individual agent's value functions can additively compose the joint action-

value function, which is also its main weakness since not every multi-agent problem is amenable to such decomposition and linear summation.

An alternative formulation by [44] works as an extension of VDN and occupies the middle ground between Independent Q-Learning, where each agent learns an individual action-value function Q_a, and a fully centralized state-action value function Q_{total} learned by COMA. The key insight, as detailed by the authors, is that a full factorization of the VDN value function is not necessary to extract effective policies; Rather, it is sufficient to ensure that the same result is reached from a global *argmax* of Q_{total} as with individual *argmax* operations performed of each Q_a. The condition sufficient for that result to hold is that a monotonicity constraint is imposed on the relationship between Q_{total} and each Q_a:

$$\frac{\partial Q_{total}}{\partial Q_a} \geqslant 0, \forall a \tag{2.2}$$

The QMIX algorithm manages to obtain much better results from both IQL and VDN in the Starcraft environment, at the cost of additional architectural complexity, since it used a neural network to combine the local functions rather than the simple linear summation of its predecessor VDN.

Chapter 3

Background

Markov Decision Processes (MDPs) are the theoretical paradigm underpinning most optimal control and single agent reinforcement learning algorithms and theory. That framework can also be extended to accommodate for multi-agent settings, which is a necessary modification since an MDP by itself is no longer adequate to describe the environment, given that actions from the other agents affect the environment dynamics.

One such extension was proposed by [27] in the form of Markov Games, and remains a foundational concept upon which MARL is based on. Borrowing notation from [7], we can define a Markov Game by the tuple $(N, S, \{A^i\}_{i \in N}, P, \{R^i\}_{i \in N}, \gamma)$ where

- $N = 1, 2, 3...N$ is the set of participating agents, where $N > 1$.
- S is the state space observed by all agents.
- A^i is the action space of the i-th agent and $A := A^1 \times A^2 \times ... \times A^N$ the joint action space.
- $P : S \times A \rightarrow \delta(S)$ is the transition probability to each state $s' \in S$ given a starting state $s \in S$ and a joint action $\mathbf{a} \in A$.
- $R^i : S \times A \times S \rightarrow \mathfrak{R}$ is the reward function of the i-th agent for a transition from state-action pair (s, a) to state s', representing the instantaneous reward received.
- $\gamma \in [0, 1]$ is the discount factor, a hyperparameter that controls the discount rate between current and future rewards.

Another generalization of the MDP that explicitly incorporates partial observability and thus allows for modelling highly complex cooperative and competitive sequential decision environments is the Decentralized POMPD (Dec-POMDP), an extension of Partially Observable MDP (POMDP) and itself a subset of Partially Observable Stochastic Games (POSG). As in the Markov Games specification, the objective is still the maximization of the expected return via the selection of an optimal joint policy.

Going by the formulation by [38], a Dec-POMDP is defined by the tuple $(N, S, O, A, P, r, \gamma$ where

- $N \geqq 1$ is the number of agents.

- S the state space of the environments.
- $O := O^1 \times O^2 \times ...O^N$ is the joint observation space of all agents, where O^i is the observation space of agent i; At time t the environment is at state $s_t \in S$ and $o_t \in O^i$ is the local observation of agent i which is correlated with s_t, as the environment's state can contain information not visible to an agent locally, such as the total number of agents active at any given moment.
- $A := A^1 \times A^2 \times ...A^N$ is the joint action space of all agents, again with A^i denoting the individual action space of agent i. Note that the observation and action spaces do not have to be equal.
- $P : S \times A \times S \rightarrow [0, 1]$ is the transition function where $P(s'|s, \mathbf{a})$ denotes the probability of the environment transitioning to state s' given current state s and joint action $\mathbf{a} \in A$.
- $r : S \times A \times S \rightarrow \Re$ is the central reward function, where $r(s, \mathbf{a})$ is the reward received by all agents when joint action $\mathbf{a} \in A$ is taken and the environment is in state $s \in S$.
- $\gamma \in [0, 1]$ is the discount factor.

Having discussed the theoretical framework that formalizes the decision-making process of the MARL problem, we will now overview the algorithms we will be utilizing in our experiments. These algorithms can support both discrete and continuous action spaces, a desirable property even though the environments we will experiment with use only discrete actions, and are state-of-the-art in standard reinforcement learning settings.

Utilizing parallel training to speed up learning in actor-critic methods is something that Asynchronous Advantage Actor-Critic (A3C) and its cousin Advantage Actor-Critic (A2C) do well. First proposed by [34], they essentially constitute a classic variant of policy gradient methods with parallelization capabilities that also have a stabilizing effect on training as they decorrelate the agents' data into a more stationary process since at any given time-step the parallel actor-learners will experience a variety of different states.It is also able to train on both discrete and continuous action spaces by default and also train via both feedforward and recurrent agents, making it quite a versatile agent.

In A3C the critics get synced with global parameters periodically while several actors get trained in parallel. The loss function to be minimized is the mean squared error between the Q-values and the state-values, or the advantage values $J_v(w) = (G_t - V_w(s))^2$, via gradient ascent to find the optimal w or set of weights for the neural networks. This function is used as the baseline in the policy gradient update rule. The outline for A3C, borrowed from the excellent survey by [59], that we will also follow for other descriptions as well, is the following:

1. Initialize global parameters ∂ and w as the actor and critic weight vectors respectively and similar thread-specific ones ∂' and w'.
2. Initialize time step $t = 1$.

3. While $T \leq T_{MAX}$:

   (a) Reset gradient: $d\partial = 0$, $dw = 0$

   (b) Synchronize thread-specific parameters with global ones as $\partial' = \partial$ and $w' = w$

   (c) Let $t_{start} = t$ and sample a starting state s_t

   (d) While $s_t \neq TERMNAL$ AND $t - t_{start} \leq t_{max}$:

      i. Choose $A_t \sim \pi_{\partial'}(A_t|S_t)$, receive new reward R_t and new state s_{t+1}

      ii. Update $t = t + 1$ and $T = T + 1$

   (e) Initialize the variable holding the return estimation

$$R = \begin{cases} 0 & \text{if } s_t \text{ is TERMINAL} \\ V_{w'}(s_t) & \text{otherwise} \end{cases}$$

   (f) For $i = t - 1, \ldots, t_{start}$:

   - $R \leftarrow \gamma R + R_i$, where R_i is a Monte Carlo measure of G_i
   - Accumulate gradients with respect to ∂':

$$d\partial \leftarrow d\partial + \nabla_{\partial'} \log \pi_{\partial'}(a_i|s_i)(R - V_{w'}(s_i))$$

   - Accumulate gradients with respect to w':

$$dw \leftarrow dw + 2(R - V_{w'}(s_i))\nabla_{w'}(R - V_{w'}(s_i))$$

   (g) Update asynchronously ∂ using $d\partial$ and w using dw

Thus, using this structure of executing updates, A3C efficiently enables parallelism when training agents simultaneously. The gradient accumulation step is a parallelized form of stochastic gradient descent since the weight matrices only get updated "a bit" towards the steepest descent direction.

While A3C's idea of having multiple workers each with their own weights interacting with their own copy of the environment enhances exploration, it is sub-optimal in the sense that the asynchronous nature of the updates from the global parameters pool can lead to some thread-specific agents working with old weights of prior policies leading to sub-optimal aggregated updates. To resolve this, A2C proposes instead to synchronously resolve this inconsistency via a central "coordinator" which has to wait for all the agents to have sent their updates before updating the global parameters, to ensure that in the next iteration all actors start from the same policy. Thus the training can be more cohesive and converge faster, due to the synchronized gradient update. A2C has also been shown to be more effective in utilizing GPUs and working with large mini-batches compared to A3C, while retaining equal or better performance to A3C, further adding to its overall efficiency. A graphical illustration of the differing architectures, also by [59], further elucidates those differences:

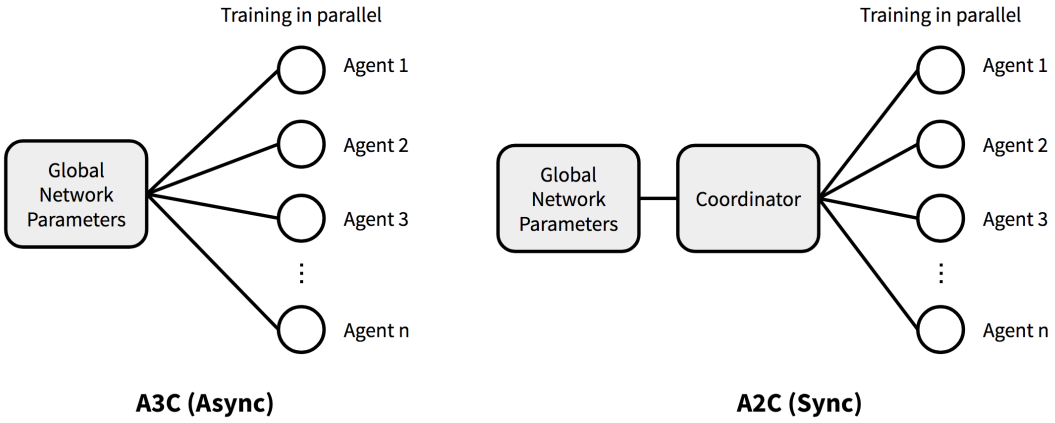

Figure 3.1. *Architectures of A3C vs A2C*

The next algorithm we will be utilizing is IMPALA, proposed by [2], having been designed expressly with the purpose of being able to handle multi-task environments and achieve very high throughput, able to scale to thousands of machines efficiently. It decouples acting and learning from the usual actor-critic setup and utilizes a novel mechanism called V-Trace for off-policy correction to stabilize learning, which along with its native scalable architecture allows it to be even more data efficient than A3C-based agents and more robust to both hyperparameter values and network configurations. The algorithm also incorporates several other optimization tricks native to TensorFlow [1] such as preparing the next batch of data for the learner while the computation is still underway or by compiling parts of the computational graph with XLA (a TensorFlow Just-In-Time compiler).

The following visualization 3.2, obtained from the original paper, clearly summarizes how this algorithm works with both a single learner and with multiply synchronous ones. In the first case, each actor generates trajectories and sends them to the learner via a queue, and the actor retrieves the latest policy updates from the learner before starting collection of the new trajectory. In the second case, policy parameters are distributed across multiple learners that work synchronously.

IMPALA actors, instead of transferring gradients with respect to the policy parameters to a central parameter server like in A3C, instead communicate by means of trajectories, namely tuples of sequences of (states, actions, rewards) to a centralized learner. Since this central learner has access to entire trajectories it can enable the parallel use of GPU resources to achieve the high throughput mentioned earlier. This however can lead to a significant lag between the behavior policy and the central learner one, behooving the use of "V-Trace" to correct for this inconsistency. Do note that the architecture allows for the presence of several learners to learn the target policy π by receiving trajectories from the set of actors.

V-trace is a novel off-policy actor-critic correction method, that is designed to correct the lag between actor and learner policies. To explain how this works, we again borrow notation from [59], and we first let the value function V_∂ be parameterized by ∂ and the policy π_ϕ parameterized by ϕ. We know the trajectories in the replay buffer are collected

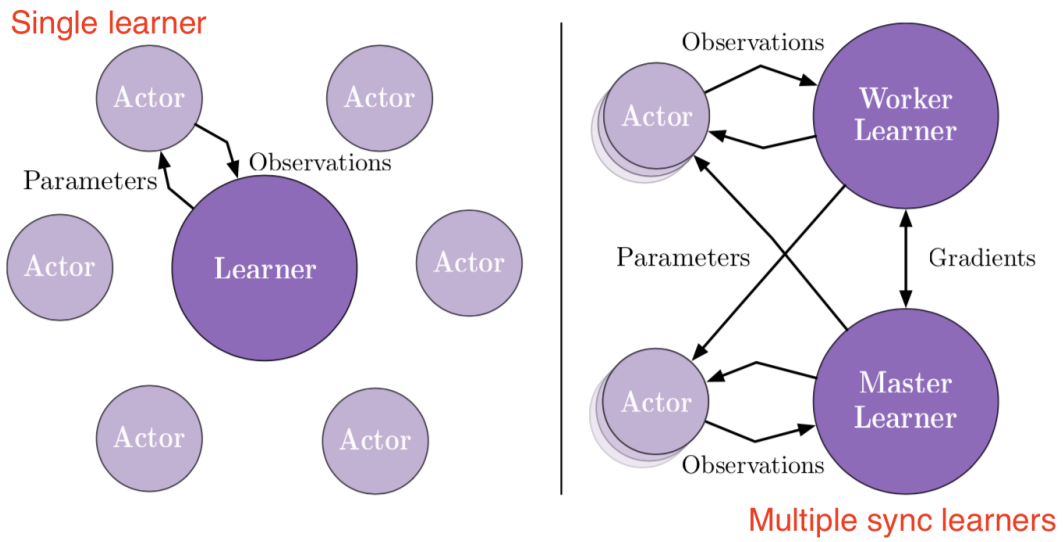

Figure 3.2. *IMPALA Learning Architecture*

by an older policy μ, which is the source of our problem as well. As training time t, given a sample from the replay buffer, a tuple of the form (s_t, a_t, s_{t+1}, r_t) the value function parameter ∂ is learned via a "Least Squares Loss" ($L2$) loss between the current value and a V-trace value target. The n-step V-trace target is defined as:

$$\begin{aligned} v_t &= V_\partial(s_t) + \sum_{i=t}^{t+n-1} \gamma^{i-t} \Big(\prod_{j=t}^{i-1} c_j \Big) \delta_i V \\ &= V_\partial(s_t) + \sum_{i=t}^{t+n-1} \gamma^{i-t} \Big(\prod_{j=t}^{i-1} c_j \Big) \rho_i (r_i + \gamma V_\partial(s_{i+1}) - V_\partial(s_i)) \end{aligned}$$

Where the red part $\delta_i V$ is a temporal difference for V whereas $\rho_i = \min(\bar{\rho}, \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)})$ and $c_j = \min(\bar{c}, \frac{\pi(a_j|s_j)}{\mu(a_j|s_j)})$ are truncated importance sampling (IS) weights. The product of $c_t, \ldots, c_{i-1}$ measures how much a temporal difference $\delta_i V$ observed at time i impacts the update of the value function at a previous time t. In the on-policy case, $\rho_i = 1$ and $c_j = 1$ and the V-trace target reduces to an on-policy n-step Bellman target.

$\bar{\rho}$ and $\bar{c}$ are two truncation constants with $\bar{\rho} \geqq \bar{c}$. $\bar{\rho}$ impacts the fixed-point of the value function we want to converge to while $\bar{c}$ impacts the speed of convergence. If $\bar{\rho} = \infty$, we converge to the value function of the target policy V^π; Else, if $\bar{\rho}$ is less than infinity our fixed point is the value function $V^{\pi_{\bar{\rho}}}$ of a policy $\pi_{\bar{\rho}}$, lying somewhere around μ and π. If at the limit $\bar{\rho}$ is close to zero, we obtain the value function of the behavior policy V^μ.

The value function parameter is thus updated in the following direction:

$$\Delta\partial = (v_t - V_\partial(s_t)) \nabla_\partial V_\partial(s_t)$$

whereas the policy parameter ϕ is updated through the policy gradient

$$\begin{aligned}\Delta\phi &= \rho_t \nabla_\phi \log \pi_\phi(a_t|s_t)(r_t + \gamma v_{t+1} - V_\vartheta(s_t)) + \nabla_\phi H(\pi_\phi) \\ &= \rho_t \nabla_\phi \log \pi_\phi(a_t|s_t)(r_t + \gamma v_{t+1} - V_\vartheta(s_t)) - \nabla_\phi \sum_a \pi_\phi(a|s_t) \log \pi_\phi(a|s_t)\end{aligned}$$

with $r_t + \gamma v_{t+1}$ being the estimated Q-value, from which a state-dependent baseline $V_\vartheta(s_t)$ is subtracted, and $H(\pi_\phi)$ an entropy bonus to encourage exploration. Thus the overall update is obtained by summing all these terms and scaling by the appropriate coefficients, which are hyperparameters of the algorithm. The combination of its scalable architecture along with the novel corrective mechanism of the V-trace and the software optimizations allowed IMPALA to outperform the A3C algorithm in terms of data efficiency, scalability and even final performance in a variety of challenging environments.

Another state-of-the-art algorithm we will be using that has achieved remarkable success on difficult environments is Soft Actor-Critic (SAC) by [17], another model-free, stochastic formulation for continuous environments, and specifically its variant built to handle discrete action spaces, put forth by [9] based on the same principles. It is based on the maximum entropy framework, which adds an entropy maximization term to the standard objective function, where the original can be recovered using a temperature parameter, which also happens to be an important hyperparameter of SAC. The maximum entropy polices framework is more robust against epistemic uncertainty and improves exploration by acquiring diverse behaviors, by incorporating the entropy measure of the policy into the reward function; In other words, the agent is expected to maximize random behavior (exploration) on the condition it still overcomes the task at hand.

SAC consists of three ingredients: An actor-critic architecture with separate stochastic policy and value function networks, an off-policy formulation that enables reuse of previously collected data for efficiency, and the aforementioned entropy maximization term to boost exploration and promote training stability. More specifically, while standard RL only maximizes the expected sum of rewards $\sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t)]$, the more flexible maximum entropy objective used by SAC which seeks to maximize both expected return and entropy simultaneously is given by

$$J(\vartheta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\vartheta}}[r(s_t, a_t) + \alpha\mathcal{H}(\pi_\vartheta(.|s_t))]$$

$\mathcal{H}$ denotes the entropy measure and α is the "temperature" hyperparameter, controlling the relative importance of the entropy term against the pure reward term, thus controlling the degree to which the policy is stochastic. The conventional RL objective can obviously be recovered in the limit as $\alpha \to 0$. The authors note that this formulation of the objective has several advantages: The first one, as mentioned several times already, is the explicit incentive given to exploratory behavior policies while ensuring that unpromising avenues are quickly abandoned. It also enables the policy to capture multiple nodes of near-optimal policy, which means that in problems where there are several actions that all seem attractive the policy will assign about equal probability mass to each of them. Lastly, it improves learning speed by virtue of more efficient exploration, and can be easily extended to infinite horizon problems via the use of a discount factor γ.

SAC aims to learn three functions:

- The target policy π_∂, parameterized by ∂.
- The soft Q-value parameterized by w, dubbed Q_w.
- The soft state-value function parameterized by ψ, V_ψ.

The soft Q-value and soft state-value functions are defined as

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho_\pi(s)}[V(s_{t+1})] \quad \text{; according to Bellman equation.}$$
$$\text{where } V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - a \log \pi(a_t|s_t)] \quad \text{; soft state value function.}$$

Thus,

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{(s_{t+1}, a_{t+1}) \sim \rho_\pi}[Q(s_{t+1}, a_{t+1}) - a \log \pi(a_{t+1}|s_{t+1})]$$

$\rho_\pi(s)$ and $\rho_\pi(s, a)$ denote the state and state-action marginals of the state distribution induced by the policy $\pi(a|s)$.

The soft state-value function is trained to minimize the mean squared error (MSE):

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}[Q_w(s_t, a_t) - \log \pi_\partial(a_t|s_t)])^2]$$
$$\text{with gradient: } \nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_w(s_t, a_t) + \log \pi_\partial(a_t|s_t))$$

where $\mathcal{D}$ is the replay buffer.

The soft Q-function is trained to minimize the soft Bellman residual:

$$J_Q(w) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}}[\frac{1}{2}(Q_w(s_t, a_t) - (r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho_\pi(s)}[V_{\bar{\psi}}(s_{t+1})]))^2]$$
$$\text{with gradient: } \nabla_w J_Q(w) = \nabla_w Q_w(s_t, a_t)(Q_w(s_t, a_t) - r(s_t, a_t) - \gamma V_{\bar{\psi}}(s_{t+1}))$$

where $\bar{\psi}$ can be an exponentially moving average of the value network weights, which has been shown to stabilize training ([35]).

SAC then updates the policy to minimize the KL-Divergence criterion:

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} D_{\text{KL}}\Big(\pi'(.|s_t) \| \frac{\exp(Q^{\pi_{\text{old}}}(s_t, .))}{Z^{\pi_{\text{old}}}(s_t)}\Big)$$
$$= \arg\min_{\pi' \in \Pi} D_{\text{KL}}(\pi'(.|s_t) \| \exp(Q^{\pi_{\text{old}}}(s_t, .) - \log Z^{\pi_{\text{old}}}(s_t)))$$

We can safely ignore the partition function $Z^{\pi_{\text{old}}}(s_t))$ that normalizes the distribution and is in general intractable, since it does not contribute to the gradient with respect to the new policy.

Taking the derivative with respect to ∂ then gives us the following objective:

$$J_\pi(\partial) = \nabla_\partial D_{\mathrm{KL}}(\pi_\partial(.|s_t)||\exp(Q_w(s_t,.) - \log Z_w(s_t)))$$

$$= \mathbb{E}_{a_t\sim\pi}\Big[-\log\big(\frac{\exp(Q_w(s_t,a_t) - \log Z_w(s_t))}{\pi_\partial(a_t|s_t)}\big)\Big]$$

$$= \mathbb{E}_{a_t\sim\pi}[\log \pi_\partial(a_t|s_t) - Q_w(s_t,a_t) + \log Z_w(s_t)]$$

Having computed those quantities, it then becomes easy to demonstrate the full SAC algorithm:

Algorithm 1 Soft Actor-Critic

Inputs: The learning rates, λ_π, λ_Q, and λ_V for functions π_θ, Q_w, and V_ψ respectively; the weighting factor τ for exponential moving average.

Initialize parameters θ, w, ψ, and $\bar{\psi}$.
for each iteration **do**
  (In practice, a combination of a single environment step and multiple gradient steps is found to work best.)
  for each environment setup **do**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \sim \rho_\pi(s_{t+1}|s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1}\}$
  for each gradient update step **do**
    $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$.
    $w \leftarrow w - \lambda_Q \nabla_w J_Q(w)$.
    $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$.
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$).

Figure 3.3. *Soft Actor-Critic Algorithm*

The algorithm also makes use of two Q-functions to mitigate positive bias in the policy improvement step, a frequent problem in earlier, more naive implementations of Q-learning algorithms without this dueling architecture, and use their minimum for the value gradient.Notably, it still can learn challenging tasks like the Humanoid with only one Q-function but using two speeds up training, particularly on hard tasks. The combination of the stochastic policies, maximum entropy objective and data efficiency make SAC stand out among many other algorithms as one of the most effective tools available.

The final algorithm we will be using is often used as a benchmark to measure new approaches and is none other than PPO (Proximal Policy Optimization), proposed by [49]. PPO builds upon TRPO (Trust region policy optimization), an algorithm proposed earlier by the same authors in [47]. Given that PPO is an evolution of TRPO, we will first briefly discuss the TRPO formulation, following again notation from [59], and then move on to PPO. The idea of TRPO then is to only take parameter updates close enough to the old policy, so that no rapid changes to it are made at each step, endangering stability of training. To quantify the degree of difference between the old and new policies the authors use the KL-Divergence constraint on the size of the policy update at each iteration.

Off-policy methods by definition use a behavior policy β to collect experience trajectories, usually by incorporating a high degree of exploration, and a target policy μ that

the algorithm tries to optimize for, usually a greedy one. The objective function then measures the total advantage over the state visitation distribution and action, while the mismatch between the training data distribution and the true policy state distribution is compensated by the IS (Importance Sampling) estimator

$$\begin{aligned} J(\partial) &= \sum_{s \in \mathcal{S}} \rho^{\pi_{\partial_{\text{old}}}} \sum_{a \in \mathcal{A}} (\pi_{\partial}(a|s) \hat{A}_{\partial_{\text{old}}}(s,a)) \\ &= \sum_{s \in \mathcal{S}} \rho^{\pi_{\partial_{\text{old}}}} \sum_{a \in \mathcal{A}} (\beta(a|s) \frac{\pi_{\partial}(a|s)}{\beta(a|s)} \hat{A}_{\partial_{\text{old}}}(s,a)) \quad \text{; Importance sampling} \\ &= \mathbb{E}_{s \sim \rho^{\pi_{\partial_{\text{old}}}}, a \sim \beta}[\frac{\pi_{\partial}(a|s)}{\beta(a|s)} \hat{A}_{\partial_{\text{old}}}(s,a)] \end{aligned}$$

Where ∂_{old} are the policy parameters before the update and thus known a-priori; $\rho^{\pi_{\partial_{\text{old}}}}$ is the discounted state distribution and $\beta(a|s)$ the behavior policy. Note that since the true rewards are usually unknown, we have to resort to an approximation of the true advantage function $A(.)$ denoted $\hat{A}(.)$.

Even when training on policy there can be a significant mismatch between the updates when they run asynchronously, as we also noted in the A3C algorithm. TRPO makes a note for that situation by labelling the behavior policy as $\pi_{\partial_{\text{old}}}(a|s)$, which transforms the objective function to

$$J(\partial) = \mathbb{E}_{s \sim \rho^{\pi_{\partial_{\text{old}}}}, a \sim \pi_{\partial_{\text{old}}}}[\frac{\pi_{\partial}(a|s)}{\pi_{\partial_{\text{old}}}(a|s)} \hat{A}_{\partial_{\text{old}}}(s,a)]$$

TRPO then aims to maximize the given objective function $J(\partial)$ under the aforementioned trust region constraint, to ensure that the newly updated policy is "close" to the old one to prevent any major discontinuities from arising, quantified by enforcing that the KL-Divergence be "small" enough, within a parameter δ:

$$\mathbb{E}_{s \sim \rho^{\pi_{\partial_{\text{old}}}}}[D_{\text{KL}}(\pi_{\partial_{\text{old}}}(.|s) || \pi_{\partial}(.|s)] \leq \delta$$

In that way, not only is the new policy constrained from diverging too much at any individual update, but also guaranteed to be a monotonic improvement.

PPO then comes as an improvement to TRPO in two major ways: It is much simpler to understand, and hence implement, and is characterized by better sample complexity (empirically), all the while without sacrificing anything in terms of effectiveness, outperforming other online policy gradient methods quite handily.The simplification works by using a clipped surrogate objective. More specifically, we first denote a probability ration between old and new policies as

$$r(\partial) = \frac{\pi_{\partial}(a|s)}{\pi_{\partial_{\text{old}}}(a|s)}$$

Then, the objective function of TRPO (on policy) becomes:

$$J^{\text{TRPO}}(\partial) = \mathbb{E}[r(\partial) \hat{A}_{\partial_{\text{old}}}(s,a)]$$

However, that is an unbounded objective function which does not guarantee that

updates will not change the policy significantly; Since ensuring stability during learning is the basic promise of trust region optimization, a constraint needs to be imposed to force $r(\partial)$ to stay within a small interval around 1: $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter. By incorporating this restriction, the objective function becomes

$$J^{\text{CLIP}}(\partial) = \mathbb{E}[\min(r(\partial)\hat{A}_{\partial_{\text{old}}}(s,a), \text{clip}(r(\partial), 1-\epsilon, 1+\epsilon)\hat{A}_{\partial_{\text{old}}}(s,a))]$$

The function $\text{clip}(r(\partial), 1-\epsilon, 1+\epsilon)$ does exactly what it implies, as it clips the ration between old and new policies to be within the interval we specified earlier. Then, the objective function of PPO takes the minimum between the original value and the clipped version, removing the incentive to increase the policy update too much for supposedly better rewards.

In case we are applying PPO on a network architecture with shared parameters for both policy (actor) and value (critic) functions, an approach actually employed quite often in cooperative games, such as the ones we will consider, the objective function should also be augmented with an error term on the value estimation (formula in red color) and an entropy term (formula in blue) to encourage exploration:

$$J^{\text{CLIP'}}(\partial) = \mathbb{E}[J^{\text{CLIP}}(\partial) - c_1(V_\partial(s) - V_{\text{target}})^2 + c_2 H(s, \pi_\partial(.))]$$

with both c_1 and c_2 being hyperparameters.

In spite of the very high performance of PPO in challenging environments like simulated robotic locomotion and Atari game playing, there are still blind-spots as evidenced by [18]:

- PPO can be unstable in continuous action spaces if rewards vanish outside bounded support.
- PPO can get stuck at suboptimal actions when up against discrete action spaces with sparse, high rewards.
- Policy can be sensitive to initialization when locally optimal actions lie close to the initialization parameters.

They then go on to suggest that using the Beta distribution for policy parameterization in continuous action spaces can be more robust to outliers and helpful to avoid failure modes, boosting the performance in MuJoCo environments, and using KL regularization as an alternative surrogate model helps especially when moving outside of current benchmarks and on the initialization issue.

Chapter 4

Experiments

In this section, we will be experimenting with a variety of environments provided by the PettingZoo library, with discrete action spaces. More specifically, we will work with the following:

Multi Particle Environments (MPE) is a set of communication-oriented environments where particle agents can (occasionally) move, communicate, see each other, push each other around and interact with fixed landmarks. They originally hail from OpenAI's MPE codebase [37], and while the originals were archived after they were used in the MAADPG paper by [28], up-to-date, maintained versions with various fixes (like making rewards consistent and cleaning up the observation spaces of certain environments) are included in PettingZoo and are the ones we will be utilizing.

There are both competitive and cooperative environments available, where the agents have to work together to achieve goals and receive a mixture of rewards based not only on their individual performance but also on the success of the other agents. We will be working with the latter subset, that includes:

- *Simple Reference*: The environment has 2 agents and 3 landmarks with different colors, and every agent wants to get closer to a target landmark, which however is known only by the other agent. Both are simultaneous listeners and speakers and rewarded locally by their distance to the target landmark and globally by the average distance of all agents to their respective targets.

- *Simple Speaker Listener*: Similar to *Simple Reference* except the one agent is the speaker that can speak but cannot move while the other is the listener that cannot move but has to navigate to the correct landmark.

- *Simple Spread*: An environment with N agents and landmarks, where agents must learn to cover all landmarks while avoiding collisions. All agents are globally rewarded based on how far the closest agent is to each landmark, while they are penalized is they collide with each other.

We will also be using a couple of Atari environments, since that suite was quite instrumental in the development of modern reinforcement learning in general. The original Arcade Learning Environment was developed by [53] while the multiplayer games were

introduced in [3]. While most of the games are of the competitive flavor, we have picked up a couple that are (mostly) cooperative:

- *Entombed* (cooperative) is an exploration game where the two agents need to work together to make it as far as possible down in the maze.
- *Space Invaders* encourages cooperation as the players can choose to maximize their score by collaborating to clear the levels. However, there can also be a competitive aspect if an agent decides to sabotage the other since there is a large bonus should the other player be hit by the enemies.

Lastly, we will also work with an environment unique to PettingZoo, part of the "Butterfly" suite, created with PyGame with visual Atari spaces. All of them require a high degree of coordination and thus learning emergent behaviors is crucial to achieving a successful policy. We will work with:

- *Cooperative Pong* is a game of simple pong where the objective is to keep the ball in play for the longest time. One agent is tiered cake-shaped as well, increasing the challenge of coordination.

It is also important to briefly discuss the theoretical framework used by the PettingZoo library to represent the games computationally, which builds on top of the Markov Games paradigm we talked about in length in the prior section. Motivated by the problems in POSG and EFG models [54] developed the *Agent Environment Cycle (AEC)* framework, which in similar function to Gym allows agents to sequentially observe their environment, take actions, receive rewards emitted from other agents and wait for the next agents to complete their respective turns.

Modelling games via an *AEC* API has several benefits, as noted by the authors: They allow for clearer attribution of rewards to different origins, enabling more effective learning; It more closely models how computer games are executed in code, particularly since the POSG-based APIs can be conceptually unclear for code-level implementations; It formally allows for rewards after every step as it is required in RL settings, whereas EFG formally only accounts for end-of-game rewards; It is simple enough to serve as a mental model even to non-experts, unlike the EFG paradigm which requires familiarity with relatively advanced game-theoretic concepts; It is much more flexible in dealing with varying agent populations, for example in cases of death or creation mid-episode and really is the least bad option for a universal API since simultaneous stepping requires the use of no-op actions which can be very burdensome whereas in sequential stepping the queuing of actions is not too inconvenient.

We will consider 2 types of Multi-Agent learning. The first one is fully Independent Learning, which as discussed in the introduction, despite suffering from various issues like non-stationarity from the perspective of any one agent due to the environment changing from the actions of other agents, empirically has been shown to work well in more than a few occasions. The second is going to be Parameter Sharing, which is an extreme

case of centralized training where all policies are represented by a single function approximator (in our case neural networks) with the same shared parameters [55]. Even though this does give rise to scalability issues and can be quite challenging to work with heterogeneous agents, it again has been shown to achieve state of the art performance, and can actually work well even with non homogeneous agents with appropriate padding, a technique we will ourselves use in some of our environments.

For the hyperparameter tuning of our algorithms along with every other core functionality related to training we will be using the excellent *RLLib* [25] and *Tune* [26] libraries, part of the open source *Ray* project for building distributed and scalable applications, providing us with all necessary tools to conduct our analysis.

The hyperparameter tuning algorithm we will be utilizing is a novel technique called *Heteroscedastic and Evolutionary Bayesian Optimisation solver* (*HEBO*) [12] that empirically was shown to be extremely effective by placing first in the NeurIPS 2020 Black-Box Optimization challenge, along with the *Asynchronous Successive Halving Algorithm* (*ASHA*) [24] for more efficient termination of unfruitful trials.

HEBO is a black-box Bayesian optimization algorithm that performs well under challenging conditions. The authors conducted extensive trials and came up with statistically important findings that most hyperparameter tuning tasks not only exhibit heteroscedasticity and non-stationarity, but also are ill-suited for individual acquisition functions (their role being to query novel input locations) which often conflict in their solutions, imposing the use of multi-objective formulations to significantly improve performance.

HEBO manages to effectively deal with those issues, as evidenced by its first place in the NeurIPS competition, primarily by combining non-linear input and output warping and adopting multi-objective acquisitions and secondarily by utilizing robust acquisition function formulations. More specifically, and without getting bogged down in details, the algorithm employs the following strategies:

To tackle heteroscedasticity, they use the *Box-Cox* transformation [4] on the outputs as a corrective mapping for non-normally distributed data, achieved by minimizing the negative *Box-Cox* likelihood function. Alternatively, when labels take on arbitrary values, they use the *Yeo-Johnson* transform [62] instead of *Box-Cox*.To account for non-stationarity, they use input warping in the vein of the *Kumaraswamy* transform as used in [50]. Together those corrections provide for an improved and more flexible surrogate model, one also relatively simple to implement as well.

To acquire a more robust objective, an essential step for early rounds of training where data is scarce and the model can be severely misspecified, that will allow the algorithm to avoid worst-case solutions and more skillfully differentiate between acquisition functions and surrogate models, they consider an expected formulation by borrowing ideas from domain randomization, as in [56] by considering an expected formulation. They then go on to prove that by only using the predictive mean and variance it is possible to approximate the expected objective efficiently.

The final component of *HEBO* is the use of a multi-objective acquisition function to seek a Pareto-front solution. This formulation facilitates the process of "hedging" between differing acquisitions so that no single objective function can dominate the overall

solution. The method they use to solve the objective is via an evolutionary solver, due to the discrete nature of hyperparameters in machine learning tasks that gradient-based solvers do not easily handle.

ASHA is a hyperparameter optimization algorithm designed to take advantage of aggressive early stopping and distributed training, as it scales linearly with the number of workers, enabling it to work effectively in tasks requiring a large number of cores. It is a very effective algorithm, as the authors demonstrated empirically that it managed to outperform several other state-of-the-art methods like Fabolas, PBT, BOHB and Vizier in a number of Neural Architecture Search benchmarks, namely designing CNNs for CIFAR-10 and RNNs for the Penn Treebank.

ASHA is inspired by the *Successive Halving Algorithm* (*SHA*) which is a principled way to conduct trials by allocating a larger fractions of resources to more promising configurations. ASHA is designed to be able to evaluate orders of magnitude more hyperparameter configurations than available parallel workers in small multiples of the wall-clock time required to train a single model. Intuitively, *ASHA* promotes configurations to the next rung whenever possible instead of waiting for a rung to complete before proceeding, and if no promotions are possible,it simply adds a configuration to the base rung so that more can be promoted to the upper rungs. We also present the pseudo-code for clarity purposes, taken directly from the relevant paper in 4.1.

In our experiments, we utilize a sophisticated neural network-based function approximator for our learners by [40], who built on top of the famed self-attention architecture, which has seen great success in Natural Language Processing (NLP) settings. They essentially adapted transformers, very large models that have proven their ability to effectively process long horizons of information for all sorts of tasks in NLP, to boost performance in partially observable RL domains. The proposed architecture, which we will be using ourselves, is dubbed *Gated Transformer-XL (GTrXL)* and manages to outperform LSTMs in the challenging multi-task DMLab-30 benchmark suite and achieve state-of-the-art performance, all the while being easy to train and implement and more expressive than its multi-layered LSTM competitors.

Self-attention architectures [58] are less susceptible to vanishing and exploding gradients than Recurrent Neural Networks (RNNs) as they, unlike them, avoid compressing the past into fixed-size hidden state chunks. Much work has also validated their superiority in a wide variety of domains like language modelling, machine translation and question answering, making them prime candidates for use where sequential information processing is central to learning, such as RL tasks. Nevertheless, LSTMs still generally remain the mainstream models for when memory is required for RL tasks, as most traditional transformer architectures are actually hard to implement and optimize, requiring tricks like complex learning rate schedules or specialized weight initialization schemes even in the supervised learning case that are not sufficient to enable transformers to solve even simple bandit tasks and tabular MDPs.

[40] attempt to solve the stability issues by introducing a novel gating mechanism to crucial points in the transformer's sub modules and reordering layer normalization. Pictorially, the final architecture, along with other transformer variants, can be illustrated

ALGORITHM 4.1: *Asynchronous Successive Halting (ASHA)*

input minimum resource r, maximum resource R, reduction factor η, minimum early-stopping rate s

function ASHA()
  repeat
    for for each free worker **do**
      $(\partial, k) = get\_job()$
      $run\_then\_return\_val\_loss(\partial, r\eta^{s+k})$
    end for
    for completed job (∂, k) with loss l **do**
      Update configuration ∂ in rung k with loss l
    end for
  until desired
end function
function *get_job*()
//Check if there is a promotable configuration
  for $k = [\log_\eta(R/r)] - s - 1, ..., 1, 0$ **do**
    candidates = *top_k*(rung k, |*rung* k| / η)
    promotable = {$t \in$ *candidates* : t not promoted}
    if |*promotable*| > 0 **then**
      return promotable[0], $k + 1$
    end if
//If not, grow bottom rung
    Draw random configuration ∂
    return $\partial, 0$
  end for
end function

as follows:

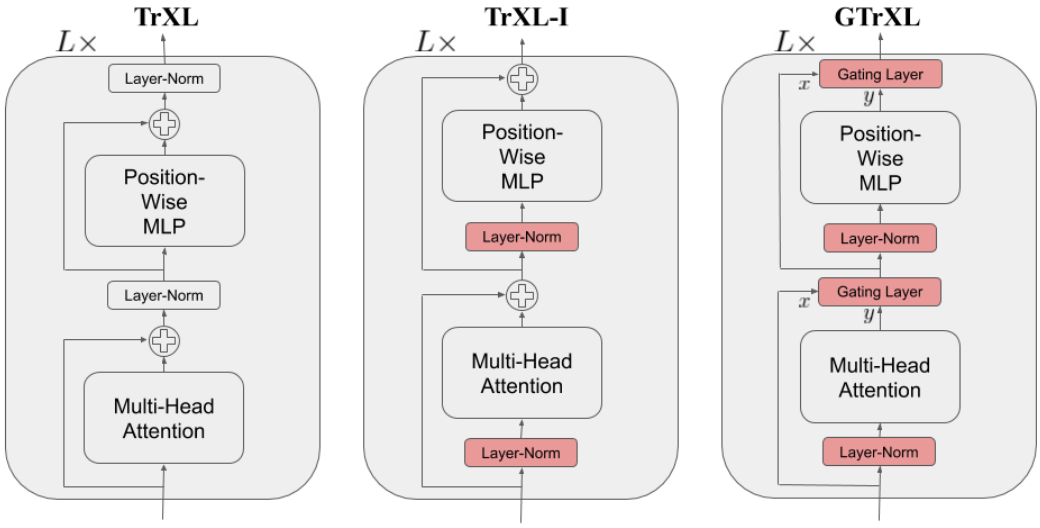

Figure 4.1. *Transformer Variants*

There are two main components of interest, namely the "Identity Map Reordering" technique and the modified gating mechanisms that contribute to the stabilization of the learning process, which we will briefly discuss.

The transformer variant *TrXL-I* using the "Identity Map Reordering" is depicted in the center of 4.1 and works by placing the normalization on only the input stream of the sub-modules, enabling an identity map from the input of the transformer at the first layer to the output at the last layer, in contrast to the vanilla transformer which incorporates a series of layer normalization operations that non-linearly transform the state encoding. The authors speculate that this tweak works well as it allows the agent to learn a Markovian Policy at the start of training process, by having the value and policy heads consume the state encoding without transforms. In other words, the network is initialied such that $\pi(.|s_t, ...s_1) \approx \pi(.|s_t)$ and $V^{\pi}(s_t|s_{t-1}, ..., s_1) \approx V^{\pi}(s_t|s_{t-1})$, under the assumption that the sub-modules at initialization produce values close to zero in expectation.

The second improvement is the replacement of the residual connections of the standard architecture with gating layers, which is illustrated in the right part of 4.1, with red denoting the modifications proposed. The final *GTrXL* layer block is thus defined as follows:

$$
\begin{aligned}
\overline{Y}^{(l)} &= \text{RelativeMultiHeadAttention}(\text{LayerNorm}([\text{StopGrad}(M^{(l-1)}), (E^{(l-1)})])) \\
Y^{(l)} &= g^{(l)}_{MHA}(E^{(l-1)}, \text{ReLU}(\overline{Y}^{(l)}) \\
\overline{E}^{(l)} &= f^{(l)}(\text{LayerNorm}(Y^{(l)})) \\
E^{(l)} &= g^{(l)}_{\text{MLP}}(Y^{(l)}, \text{ReLU}(\overline{E}^{(l)}))
\end{aligned}
$$

Where g denotes the gating function. The authors ablated a variety of gating layers, with the Gated Recurrent Unit (GRU) [8] being the most performant of those, which is also what we will be using in our experiments. It is described by the following equations:

$$
\begin{aligned}
r &= \sigma(W^{(l)}_r y + U^{(l)}_r x) \\
z &= \sigma(W^{(l)}_z y + U^{(l)}_z x - b^{(l)}_g \\
\hat{h} &= \tanh(W^{(l)}_g y + U^{(l)}_g (r \odot x)) \\
g^{(l)}(x, y) &= (1 - z) \odot x + z \odot \hat{h}
\end{aligned}
$$

Before commencing any actual learning on our environments, it is imperative that we establish a measure of baseline performance so that we can later compare it to our actual algorithms and see if they are learning anything, and if they do, assess the degree to which they do. To that end, we employ a random policy on all of our games for a total of 250 episodes each, a number large enough to give us a fairly accurate baseline. Table 4.1 shows those baseline results.

The environments are evaluated with their default settings, that can be found in their

Environment	Random policy score
Entombed	6.36
Space Invaders	844.86
Simple Spread	-118.52
Simple Reference	-56.47
Simple Speaker-Listener	-79.17
Cooperative Pong	-5.32

Table 4.1. *Baseline scores*

respective documentation pages in PettingZoo (for instance Atari) though we have taken some important preprocessing steps that will be analyzed per environment.

Having established our baselines, we will first look into the performances in the MPE environments, with the use of violin plots for the entirety of the rewards observed during the optimization runs, that essentially were runs with a much larger number of episodes following the hyperparameter optimization ones which worked with fewer episodes for computational efficiency reasons. To supplement our understanding, we will then peek at the "learning curves" of our algorithms per environment, that is the mean reward of all agents per iteration, to get an insight into how their training progressed across time.

It should also be noted that the number of episodes per environment was not held constant, again due to computational reasons, as the visual Atari environments were far slower to train than their lightweight MPE counterparts, thus shorter lines may be observed in the learning curve plots, without that however affecting the results in any meaningful way.

MPE environments in general received relatively light preprocessing. The Reference and Spread ones did not receive any preprocessing at all, besides being wrapped appropriately for use with the RLLib library. The Speaker-Listener however required some additional preprocessing, in the form of padding both the action and observation spaces of the heterogeneous agents to the same length, a necessary move for algorithms using shared learning techniques to work, as already discussed at length by [55], who rigorously proved not only that disjoint observation spaces allow for learning optimal policies but also that the padding of heterogeneous action spaces allow for convergence to optimal policies.

The results from the Reference environment are interesting, though mostly unsatisfactory. We can clearly see that most algorithms' median, as indicated by the internal box plot within the overarching violin plots, lie below or just above the red line, indicating the random policy baseline. Generally, the Shared and Independent policies are equally performant, with the shared policies generally having slightly better median returns, though exceptions do occur. We observe that there are rather long tails towards the negative ends, which implies that the algorithms took quite a while to figure out the game and start amassing positive (or less negative) rewards, or never managed to do well in the first place.

Regarding the A2C algorithm, a very interesting contrast between the independent and

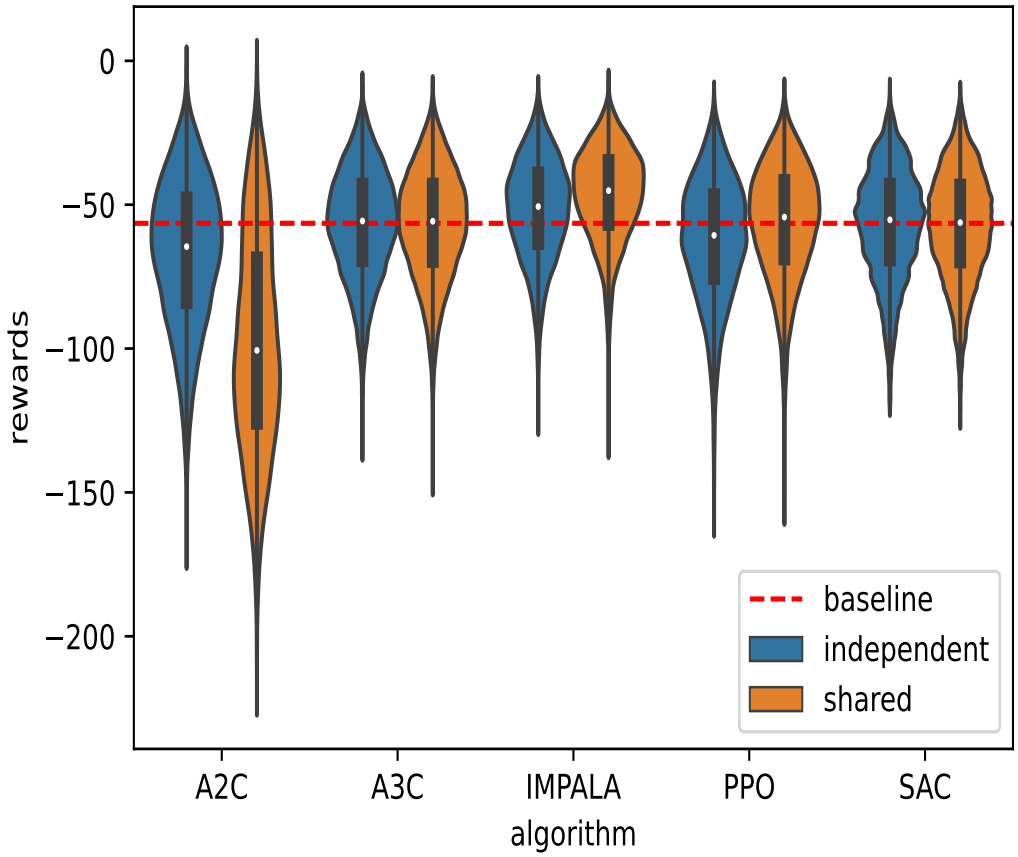

Figure 4.2. *Reference scores vs baseline*

shared learners occurs: The independent ones show a much higher median score and a more concentrated distribution around the baseline, while the shared learners have an almost uniform distribution with very elongated tails and a median far below the baseline, which can only mean that it effectively did not manage to learn anything useful even with the "optimal" hyperparameters.

The situation with the Asynchronous variant of the algorithm, A3C, is much smoother that the one observed with A2C. The distributions of the returns are very similar, and the median returns are almost exactly the same. It is however unfortunate that the median is very near the baseline, indicating a not very good fit, though there is a significant concentration of returns over the baseline implying that it does not fail entirely.

IMPALA admittedly seems the best fit algorithm, with both learner variants visibly over the baseline and the shared one being the best performing algorithm of the entire batch. It is also noticeable that the shared learner, even though follows roughly the same distribution, has longer tails of more negative rewards, indicating that it took a while longer to achieve relatively good performance over its independent variant, though ultimately doing better.

PPO shared again seems to outperform the independent one but only slightly, with

both variants having a similar distribution of rewards. The median of the independent learner hovers just below the baseline but in general both versions do not really seem to have achieve good performance. They also have the second longest tail into negative rewards territory, behind A2C's particularly problematic performance.

Lastly, SAC performance is very roughly even among the variants, with the independent slightly outperforming the shared, though to an almost negligible degree. It is interesting to note that they seem to have the tightest distribution of all algorithms, implying a sense of relative stability in their performance: Not too good but not venturing too long into sub-baseline territory.

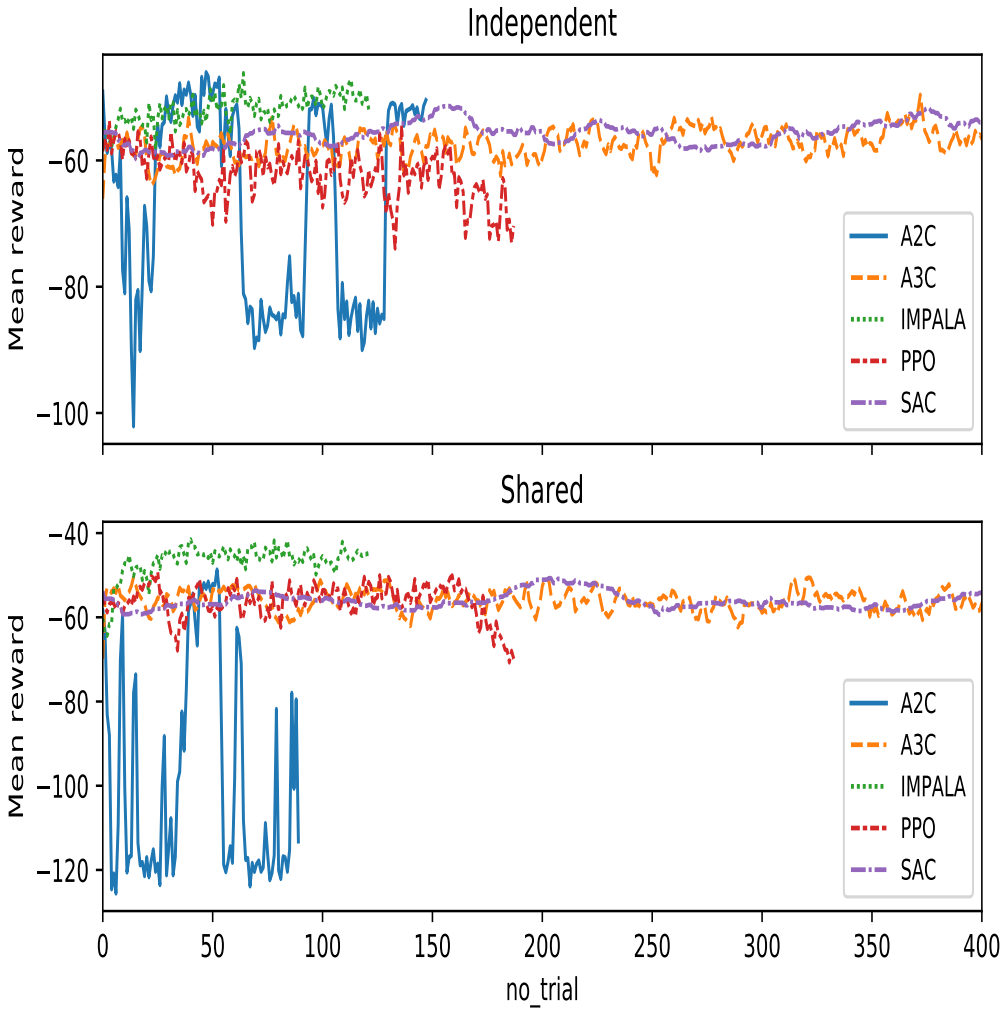

Figure 4.3. *Reference average reward over time*

Our learning curves also play intuitively nice, and corroborate our findings from the violin plots. We can generally witness the fact that none of the algorithms, of either variant, really ever managed to understand the environment well enough to consistently score ever-increasing rewards as the training was running and almost all of them languish in noisy straight lines, instead of increasing curves as we would ideally expect.

Of particular interest is A2C's extreme variance, in both its learning variants, compared to every single other method in this batch. It is the only one that manages to achieve rewards of up to -100 points as independent learners and a whooping -120 with shared learning, indicating that the latter, slightly more sophisticated MARL methodology, completely failed in this regard. Given that its Asynchronous variant, A3C, achieved a significantly stabler learning curve indicates that perhaps the asynchronous manner of updating in A3C accelerated and stabilized the learning process.

IMPALA seems to be the one algorithm doing slightly better than the rest with both its variants, though only by a short margin. In fact, we can observe a slightly upwards trend in its shared variant and some distance from the other shared learning algorithms, crowning it as the best such approach. The independent is also the best, though with even less of a trend and a shorter distance from some of its competitors.

PPO, in somewhat ironic fashion, can actually be observed having a clear downwards slope, particularly with its independent variant, which means that further training would only serve to further compromise performance. The shared learner is more stable for most of its training but can also clearly be seen degrading rapidly towards the end of its training, indicating that it under-fit the environment. There is some cause to believe that the hyperparameters were poorly configured, something that would require several more reruns to establish more confidently, as PPO unfortunately can be quite sensitive, despite it being designed to perform in a more robust manner.

SAC is also rather obtuse in its performance, despite receiving very extensive training, though notably it is also characterized as the algorithm with the least amount of variance in its training, as both learners are almost uniformly a straight line with very few ups and downs. It is also clear that it did not see any benefit from the longer training time, and would not have improved with more training either, failing in the same way as PPO.

The next graph 4.4 is about the Spread environment, where things in general look better than in the previous case. We see that almost all of our algorithms achieve median scores over the baseline, though not very far away from it. We can also observe that there are several long tails of negative returns, and that again there is equality between shared and independent learning variants, with shared learning only a bit superior.

A2C in this case seems much more principled, and actually presents the reverse situation from what we saw in the prior 4.2 figure: The shared variant is not only significantly better performing, and actually among the best performing configurations of the batch, but also it now has a much more centered distribution, with a significantly larger median value than the independent learner, which not only scores just below the baseline but also has by far the longest negative tail.

A3C also achieves median returns over the baseline for both learners, with the shared learner achieving slightly higher maximums, even though it does have a longer tail of negative values. The shared variant for this case achieved the highest median score of all algorithms in this game, albeit only marginally over A2C and in general not far away from the other well-performing algorithms.

IMPALA again achieves pretty good performance in both learning configurations, with the shared variant slightly outclassed by the independent learner, albeit with the most

concentrated distribution among all other trials, while the tail of the independent being almost double the length, indicating a much larger total number of rewards having strong negative rewards.

PPO shared again outclasses the independent variant, having a slightly higher median performance and maximum values, with also a shorter tail of extreme negative values, though only slightly so. SAC again has around even performance with both variants, with independent learning very slightly sporting a higher median, though both have very similar distributions and tails, almost identical, again indicating the stability of training across the different paradigms.

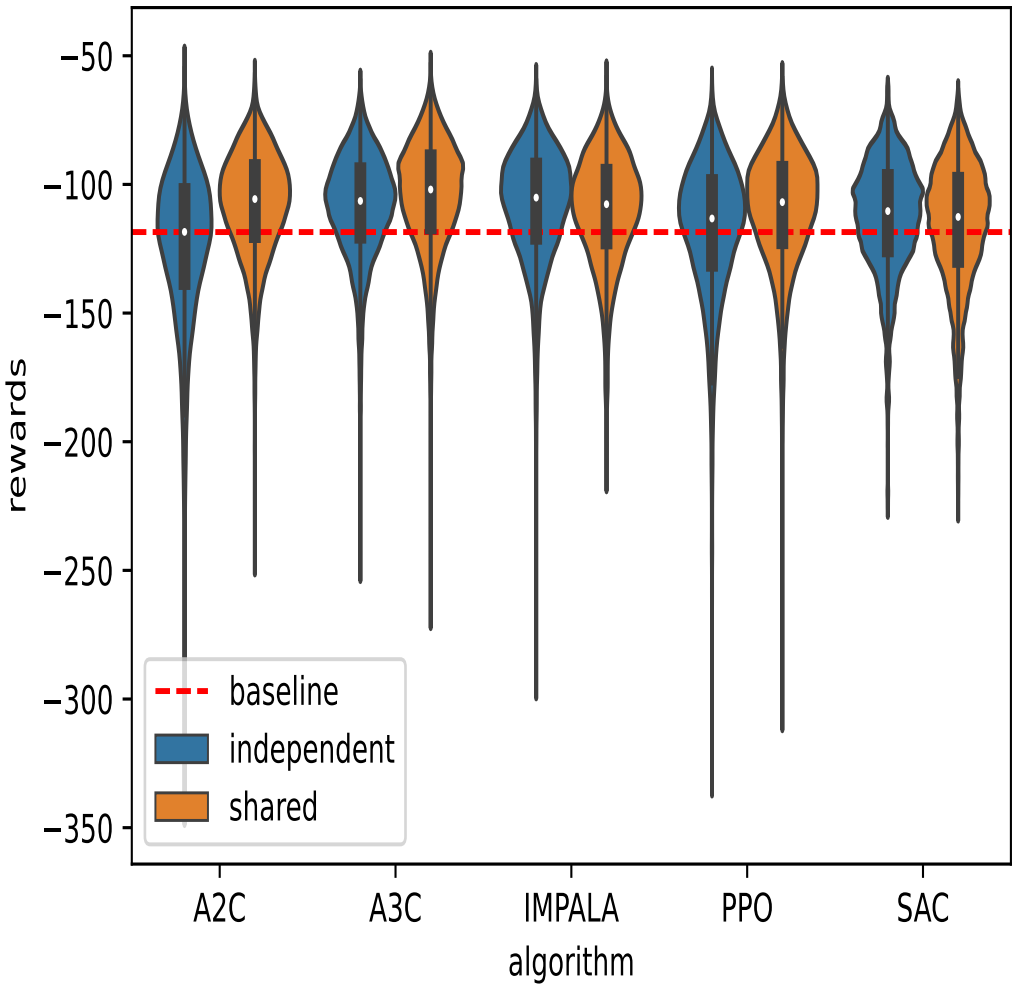

Figure 4.4. *Spread scores vs baseline*

The learning curves concur with our prior assessment from the violin plots. We can clearly see that no algorithms, of either variant, really achieve consistent learning as would be indicated by an upwards curve indicating a mean of increasing mean rewards over time. Instead, we get a very similar-looking plot as in 4.3, with all learning curves being mostly flat lines with varying degrees of noise.

A2C independent displays similar extreme variance as its counterpart in the previous environment, achieving by far the lowest score of -180, but funnily also the highest at -100. The shared variant interestingly seems to behave much more stably than its independent variant, and having variability in line with the rest of the algorithms, though not better performance.

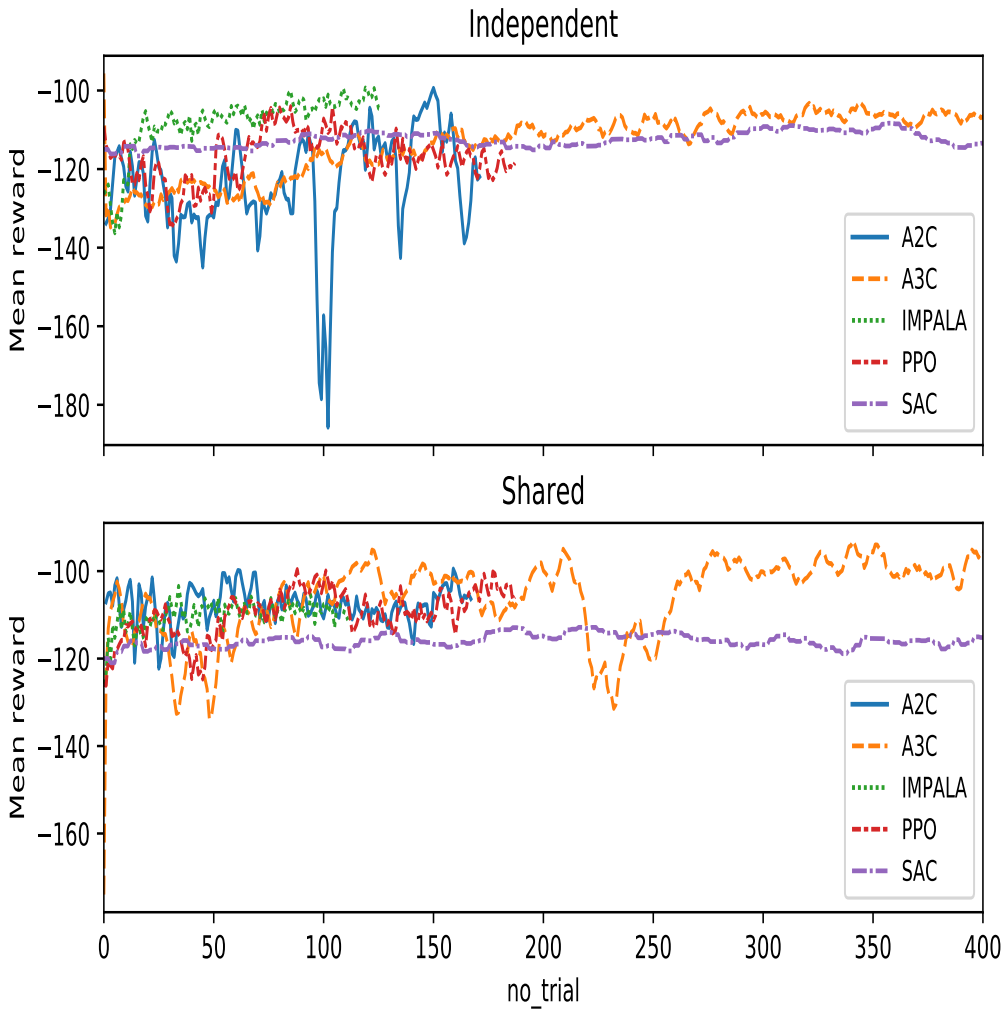

Figure 4.5. *Spread average reward over time*

A3C presents the reverse image: Its independent learner presents significant stability, and has one of the only 2 curves that seem to have a somewhat upward trending line with its independent variant, progressing very slowly but steadily. Its shared learner on the other hand has the largest variance of the batch, though interestingly also achieves the highest running score. Though quite noisy, it could be surmised that with additional training it could improve to a degree.

IMPALA independent also has the second weakly increasing trend, again achieving the highest score among the independent learners, with a similar but clearer trend with the one found in the Reference environment. Meanwhile its shared variant languished

somewhere among the middle point of its other competitors without achieving anything worthwhile.

PPO has colourful performance across its training, with significant variance and a changing trend, alternating between degrading and improving. It its independent variant it seems to hit a long stub after a while, indicating that more training would only further worsen it, while its shared variant seems to be slowly trending upwards, though with quite a lot of noise.

SAC is nearly identical to the one in the previous game, with markedly little variance and a curve with a clear lack of any trend, implying that despite its long train time it failed to learn anything of importance.

The next graph and final MPE environment is the Speaker-Listener, which effectively works the same way as Reference except that one agent is the 'speaker' and can speak but cannot move, while the other agent is the 'listener' that cannot speak, but must navigate to correct landmark.

The results in this environment are quite divergent from the previous two, where the tails were rather limited and the performances would more or less hover around the baseline. Here however we can see that all algorithms have extremely long tails, which we have actually chosen to truncate for visualization purposes, indicating a significant challenge to start accruing positive rewards. Thankfully, despite those extremely skewed distributions, we can see that some algorithms do manage to achieve rather good median scores, with again shared and independent learning approaches being roughly equally matched.

A2C here again does a reversal in its performance, more closely resembling the Reference game with both approaches having extremely negatively skewed distributions and medians well below the baseline, Nevertheless, they still have a fat piece of the density function over the baseline and reaching the highest maximal scores of the batch, possibly meaning that while it took a long while to learn the dynamics they eventually started to get much better, though possible requiring much additional training to move the median.

A3C again proves to be a better choice than its asynchronous variant as both learners achieve scores comfortably over the baseline, with the shared learner slightly having a higher median and a fatter positive distribution tail. IMPALA achieves almost identical performance-wise, with the two approaches being prety much the same, continuing on the trail of stability that it has established from the first game. A3C and IMPALA are also similar in their negative curves, which they are the thinnest among the batch.

PPO is an interesting case due to the very significant differences between the shared and independent learners: While both are objectively not particularly good with respect to the baseline, shared learning is very far away from the line, with the lowest median performance and its distribution itself is extremely elongated towards the negative rewards frontier, signalling that it failed substantially at learning a policy that produced enough 'good' rewards. Independent learning is much more concentrated over the line, implying that it might have done a somewhat good job.

SAC continues in the trend of equality between the learners, but with unsatisfactory median scores, both just below the random baseline. It also has rather far tails towards

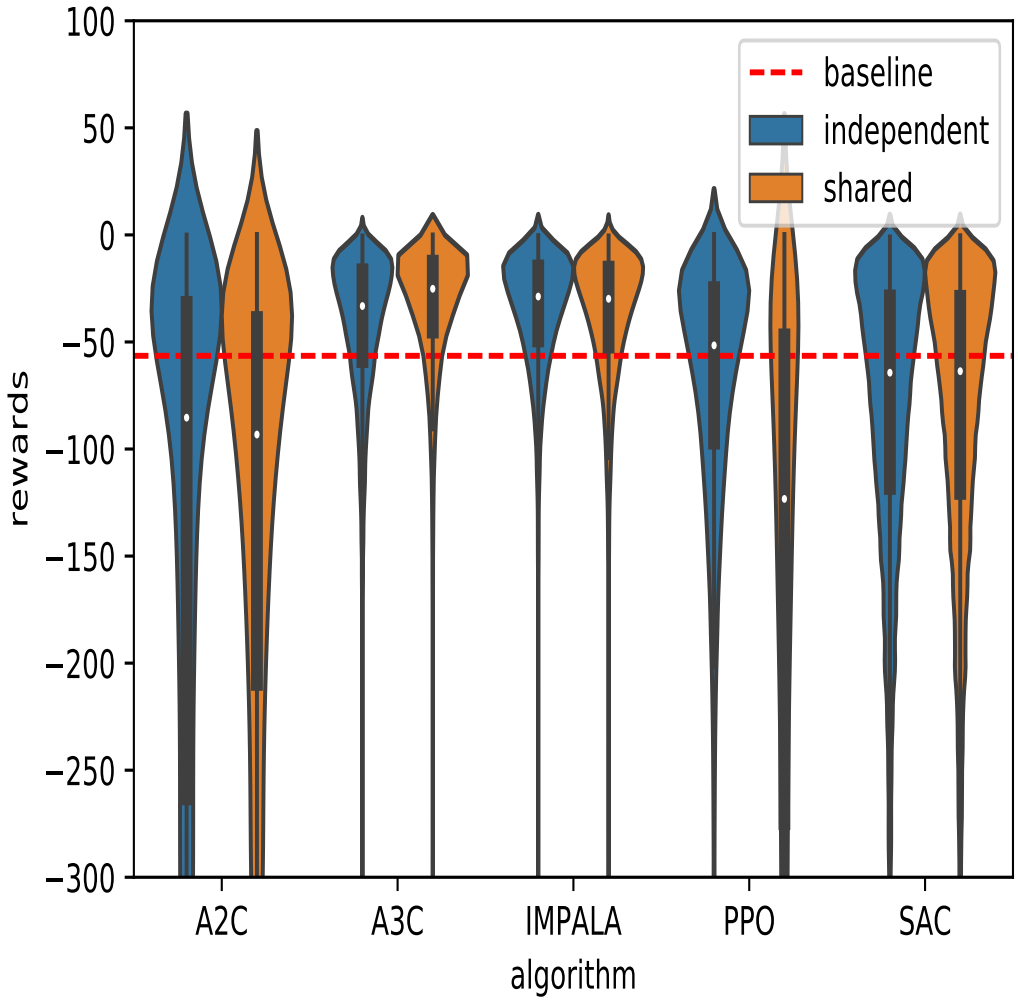

Figure 4.6. *Speaker-Listener scores vs baseline*

the negative end, second only to A2C's intense asymmetry, but still has a non-trivial mass of rewards over the baseline, indicating that it might have too used a lot more training iterations to build up more positive performance.

We then take a look at our learning curves for a more nuanced view. We can see again the expected correspondence with our findings from the plots up above, that again do not show any algorithms as being particularly well-adapted to our final MPE environment, with curves similar to what we have already witnessed; relatively straight, with varying degrees of noise, and without the coveted increasing trend. No clear winner emerges from the two variants, though independent learners tend to go less into extremely negative territory.

A2C again seems to be extremely volatile in both its versions, showing that this algorithm in general, at least given its specific set of tuned hyperparameters, could not handle the cooperative behaviors it should have come up with in the MPE environments, the result being extreme variance an overall poor performance. Interestingly, its indepen-

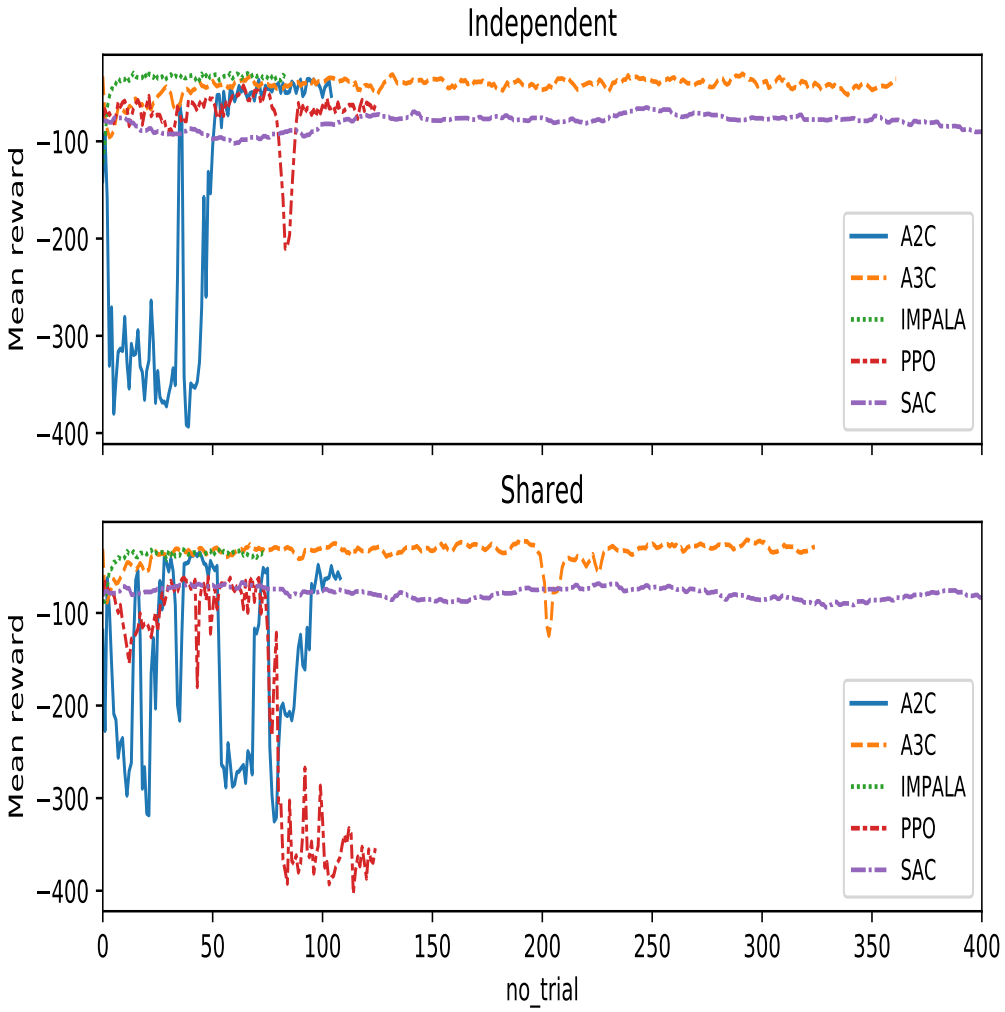

Figure 4.7. *Speaker-Listener average reward over time*

dent learner, following a chasm of performance directly prior, seems to have picked up a more confidently increasing line, on par to achieving the best performance among these learners. Its shared learner on the other hand never exhibits similar behavior, constantly going wildly up and down without ever establishing a stable foothold.

A3C, as we saw in the violin plots, establishes itself as the best algorithm in both variants, while also exhibiting stability and little variance during its training, except for a probably random fluke at some point in its shared learning. This does not really mean good performance in absolute terms however, as it very quickly plateaus at a score just over the baseline and does not seem to be able to do much better.

IMPALA exhibits similar behavior with A3C, again plateauing almost immediately but with a score roughly equivalent to A3C, and with very little variance. SAC is also following in the same trend as IMPALA and like its versions in other environments, achieving a relatively worse score with little variance in spite of its long training time.

Finally, PPO again negatively surprises with its performance, by having only the third

best score as an independent learner, though with relatively little variance minus a large drop just before it finishes its run, while its shared learner is by far the worst of the batch by having a clear downwards slide and achieving the lowest possible score, again implying that its sensitivity to changes has impacted its performance.

Moving on from the MPE games, we now investigate the performance on the Atari and Butterfly environments, which we "bundle" together since they are very similar in function and have received identical preprocessing. Essentially, they are all high-dimensional state-space visual environments, that feed the agent with images instead of a small list of positional information, like MPEs; Therefore, they also require a fair bit more processing for the algorithms to more efficiently process them, following conventions already set by research papers.

The Arcade Learning Environment (ALE), long considered instrumental in the development of modern reinforcement learning, is not without its flaws: As thoroughly argued by [29], ALE suffers from determinism, meaning that the underlying emulator is deterministic given the agent's actions, e.g the agent always starts at the same initial state and given a sequence of actions always gets the same results. This determinism can be exploited by agents who simply memorize effective sequences of actions and ignore the perceived state altogether.

To resolve this situation, the same authors [29] proposed the use of *sticky actions* to inject stochasticity into the ALE, that also help evaluate the robustness of learned policies. They have been designed to adhere to the following criteria:

- The stochasticity should only be minimally non-Markovian with respect to the environment. This entails that the action to be executed by the emulator should be conditioned only on the action chosen by the agent and on the previous action execute by the emulator.
- The difficulty of the environment should not be affected, meaning that the algorithms which do not exploit the determinism of the environment should not be additionally hindered by the new stochasticity.
- It should be easy to implement in ALE without requiring changes to the underlying emulator.

Sticky actions then introduce a *stickiness* parameter ψ that defines the probability the environment will execute the agent's previous action again, instead of the agent's new action. More formally, at time step t the agent decides to execute action a; However, the action A_t the environment in fact executes is

$$A_t = \begin{cases} a & \text{with prob } 1 - \psi \\ a_{t-1} & \text{with prob } \psi \end{cases}$$

Sticky actions also synergize well with other aspects of the ALE, like frame skipping, which we will also be doing as part of our preprocessing, as at each intermediate time step between the skipped frames there is a probability ψ of executing the prior action.

Sticky actions are also more reasonable than random delays as in the former case the agent can make a different decision at any time by sending a new action to the emulator, while in the latter the action taken has to be executed until the delay is passed, in effect preventing the agent from making another call in the nick of time.

Another issue with ALE is frame flickering, meaning often not every sprite every frame is rendered owing to hardware restrictions. For instance, sprites in some games like *Joust* or *Wizard of Wor* are rendered every two or three frames. The standard way of handling this issue is by computing the pixel-wise maximum of the previous n observations, the way we also followed.

Finally, other more basic preprocessing steps undertaken involve skipping frames for faster processing and less predictable sequences of states, resizing the observations for faster processing and down-scaling them, turning them into gray scale and stacking frames to allow the agent to see everything on the screen despite the flickering we mentioned above.

More specifically, for all our remaining environments, namely Entombed, Space Invaders and Cooperative Pong, we chose to take the maximum over the last 2 frames to deal with flickering, set $\psi = 0.25$ (sticky action parameter, probability of repeating the old action in lieu of the newest one) to inject stochasticity, skip over every 4 frames for faster processing, resizing the images to $84x84$ per [35] , stacking images up to batches of 4 and finally normalising to the range of $(-1, 1)$ for neural network training stability purposes.

Having discussed the preprocessing pipeline for our Atari games, we now turn our attention to the Cooperative Entombed environment performances. Here we can clearly see the seemingly peculiar structure that the violin plots follow, with most of them having 3 dense sections of returns instead of the more "traditional" KDE curve we had seen earlier. This can be explained by the nature of this particular game, as it can be described as an exploration-type of game with 5 stages, with agents only receiving rewards when the stage resets (usually via agent losing a life) or after progressing to the next state, and can only receive specified kinds of reward signals, like +4, +8 or +12 points, as can be seen by the dense concentrations around some of these specific values. Note that the violin plots we are studying do not give us a specific insight into how many levels the agents manage to complete during the training.

This is also the first time that we can clearly see that the shared learning paradigm clearly outperforms the independent learning one, with several instances of shared learning having an obvious advantage over its counterpart, which several times failed completely to obtain any meaningful rewards. This is intuitively satisfying in this case since the purpose of the game is to collaborate in order to progress through the stages.

More specifically, we can see that the performances of the algorithms A2C and A3C are very nearly identical. They both have median scores just below the random policy baseline, a rather disappointing find, and have roughly equal concentrations of the different types of rewards at all possible stages. It is interesting to note that the shared and independent variants are virtually indistinguishable, implying that these algorithms both failed by design to make much progress in this environment and the learning paradigm under which they operated probably would not have made much difference.

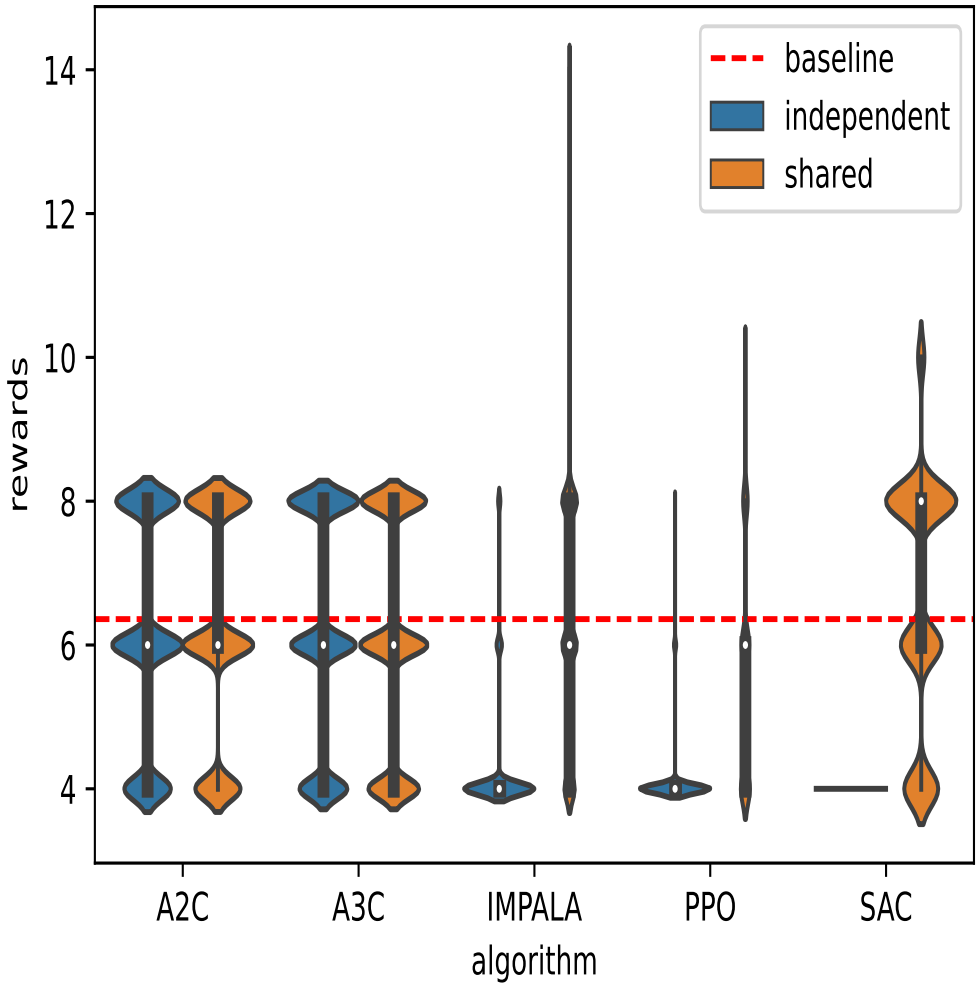

Figure 4.8. *Entombed scores vs baseline*

IMPALA presents another interesting image and is an algorithm which in independent learning mode seems to largely fail, with a median score at the lowest possible level and only scarcely receiving higher rewards, that only get rarer the higher their value, with a small concentration around 6 and a tiny one around 8. The shared learner does significantly better, with a median much closer to the baseline of roughly 6 points (though still below it) and with a noticeable tail over higher values, though scant after +8.

PPO behaves very much like IMPALA, with the shared learner achieving a much higher score than the independent learner, albeit still below the random baseline score. The independent PPO learner is almost exactly the same in behavior as its IMPALA counterpart, with almost all the mass of the returns around the lowest possible value of return. The shared one though interestingly cuts off sharply after the baseline, unlike the IMPALA one, indicating a much poorer overall fit and much lower ceiling.

Admittedly, the SAC algorithm presents by far the most interesting image of the trials in this game. Its independent learner variant is by far the poorest performer of the entire batch, being literally a degenerate distribution stuck at exclusively receiving the lowest possible reward, failing completely at learning anything of interest. Meanwhile, the shared variant achieved by far the best score of the batch, with a fat distribution around +8 and over the median.

We now will take a look at the learning curves, illustrated in 4.9. Given the whimsical nature of the rewards encountered in Entombed, these curves can actually reveal quite a bit more than the violin plots we saw earlier. Indeed, we can see some interesting patterns emerging. Shared methods seem to be outperforming the independent ones on grounds of several curves of the former lying in a higher area of rewards than the latter, in which several asymptotically or actually reached the lowest possible reward threshold.

More specifically, A2C now appears to have much more controlled variance, in line with that of all the other algorithms. It seems competitive against A3C in their independent variants, achieving consistently the second highest score, though neither of the curves show any signs of powerful learning. Its shared variant is also pretty competitive, again claiming second place and treading close to other competitors.

A3C is an intriguing case, as while it achieves the best performance during its training with independent learners, it achieves the worst with the shared learner, implying that the asynchronous update mechanisms and the shared learning did not work well in this Atari environment. It is a good question to see if this trend continues with our other 2 games, as we will shortly do.

IMPALA is another one algorithm that entirely fails to learn with the independent paradigm, following a constant downward slide and ending up flat-lining at the end, only achieving the lowest possible score. Its shared variant does comparably far better, but still remains quite unimpressive compared to its competitors. This is the first environment where we witness IMPALA perform poorly, so again it is an interesting question whether or not this was a fluke, perhaps owing to the specific hyperparameter set used, or a more general problem it may have with more complex, visual environments.

PPO continues its trend of disappointing performance, by not only being extremely slow to train as independent learning goes, as evidenced by its tiny curve compared to

the others, but also having very high variance as a shared learner and again achieving extremely low scores.

SAC, as we already saw in the violin plots, gives rise to an interesting dichotomy: While its independent variant is degenerate from the start at the lowest possible score, its shared variant achieves the highest score of all algorithms in the shortest amount of time. This, along with all the other algorithms that failed as independent learners, is quite satisfying intuitively as Entombed is a highly cooperative environment, where coordination is key to receiving rewards, something that is very hard to achieve with independent methods.

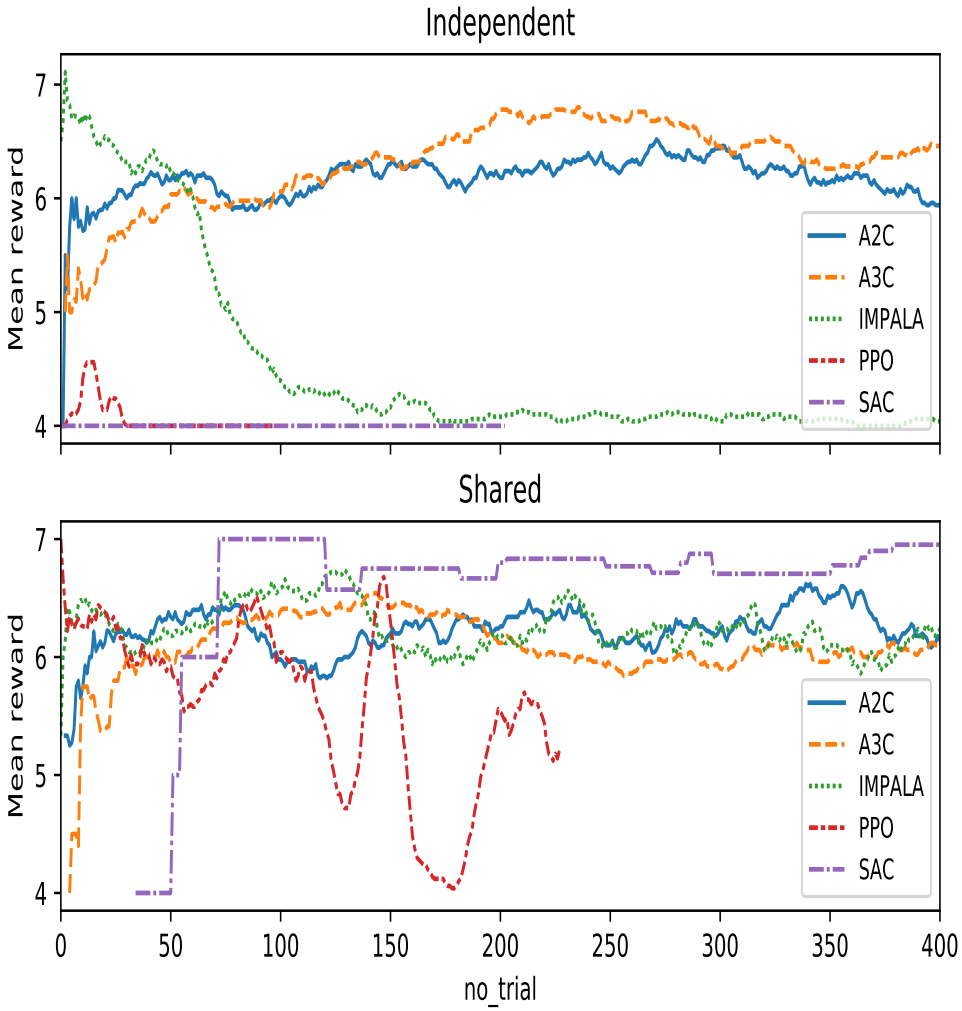

Figure 4.9. *Entombed average reward over time*

Our second Atari environment is going to be Space Invaders, in multiplayer version. The return density plot 4.10 has quite a few interesting things to tell us, with the match up being shared and independent learning returning back to being quite a stalemate, as we have cases when either one can prevail, even quite substantially, over the other. We can also observe here another by now rather familiar artifact, that of very long tails, although this time thankfully these tend to be towards the higher return side.

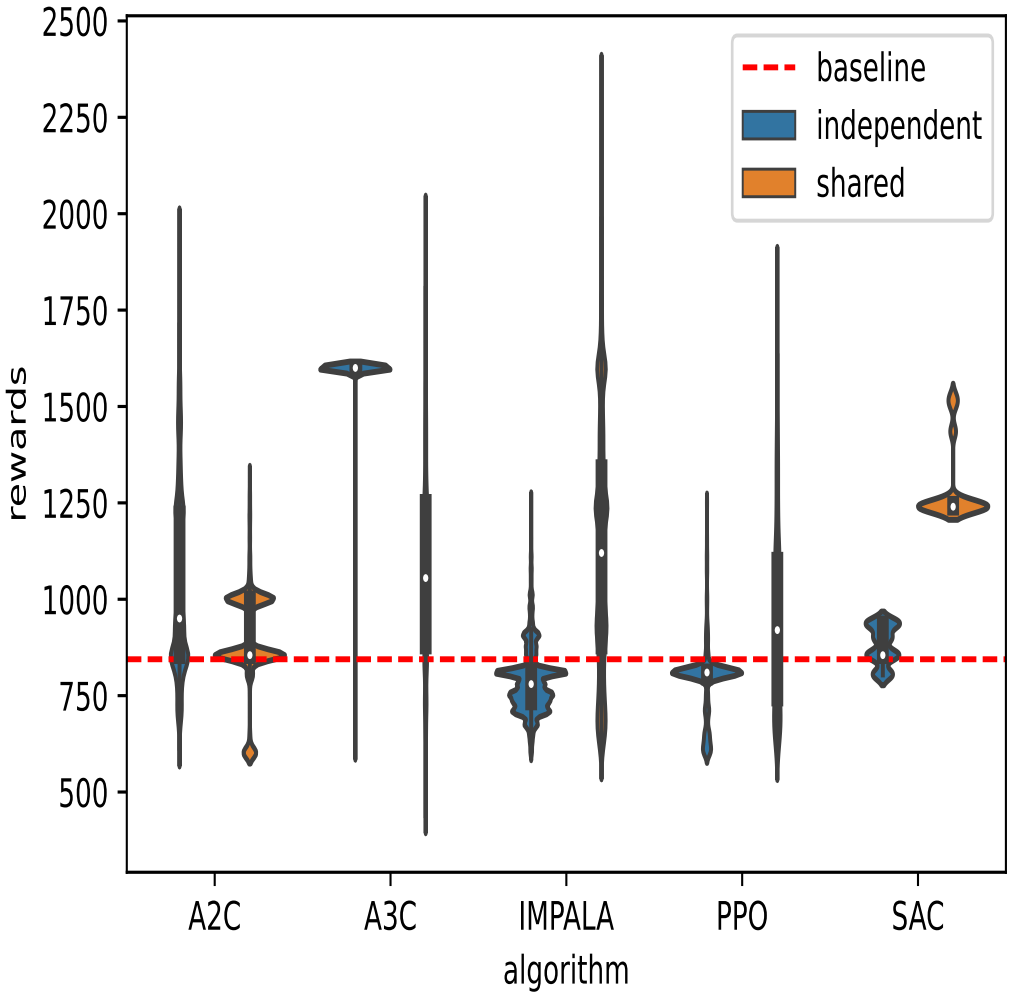

Figure 4.10. *Space Invaders scores vs baseline*

A2C presents relatively good performance, with median mass of returns for both learner variants being over the baseline. The independent learner seems to achieve visibly better performance than the shared variant, being closer to 1000 score than the shared which seems barely over the edge at roughly 800 points. Independent also has a much higher tail into very high score, which gets very thin over 1250 points though, indicating that it might have benefited more from even more extensive training.

A3C achieves all-round superior performance to A2C, and in fact the independent variant achieves by far the best median performance at roughly 1600 points. It has an extremely thin tail of returns below the median, indicating that the algorithm was in fact quite rapidly learning how to navigate the environment in a productive way and receiving a very large number of high performance rewards following a period of training. With significantly more training episodes, it potentially seems like it could solve the environment entirely, if it were to continue on that upwards path.

Meanwhile, the shared learner, even though comfortably higher than the baseline,

hovers in a much more most regime of roughly 1100 points, which does again indicate that it could potentially benefit from more extensive training to boost its performance. This hint is further reinforced by observing the much higher tail of the shared learner, indicating that it was in fact finding ways to experience high returns, higher in fact that the peak of the independent one.

In training IMPALA we again see a reversal of the performance between the two variants, where the shared learner is clearly much more superior. The independent one's entire mass of returns lies clearly below the baseline, with the median just short of the baseline and with a relatively short extreme tail upwards. This implies that the independent learner did not really converge to any useful policies and did not start receiving any really positive returns at any point, thus a longer training run would probably not have helped much.

This stands in somewhat contrast with what we see in the shared learner, which achieves a pretty good median of around 1100, far away from the baseline, and does have an almost uniform distribution of returns on its entire life. In fact, it has the highest maximum among the batch, implying that it could too benefit to a degree from more training or more advanced exploration techniques.

PPO with both of its variants continues its surprising trend of under-performance. Its independent variant achieves the second worst median performance, only exceeded by IMPALA's equivalent, while its shared variant does not do much better with a score only mildly over the random score baseline and a dense concentration of returns below it. This perspective provides us with the insight that it also largely failed to work well in this environment despite its long run time, and probably would require much more significant investment of computational resources if it were to get up to speed.

Lastly, we have SAC and another clear winner in the face of shared learning, by achieving the second highest score of the entire batch at a comfortable $\approx$ 1250 points and way over the baseline, while its independent variant languishes barely over the baseline at a mere $\approx$ 800 points median score. This also potentially means that SAC shared, given a much more significant training budget, could also conceivably solve the game with the given tuned hyperparameters.

As customary, we now take a look at the learning curves of our algorithms, in 4.11. Here, unlike Entombed, and again corroborating our violin plots, we can observe more balanced performance between the two paradigms of learning we employed, with both having algorithms achieving pretty high scores. In fact, this is the game with the highest scores versus the baseline, that we have experimented with so far.

We can clearly observe A2C behaving in the same manner as it did with the Entombed environment: Its independent learner being a very competitive sport, achieving clearly the second highest consistent stream of average rewards, whereas its shared variant achieving the worst scores and even falling into a global minimum of scores at a point, after which it plateaus. It is interesting to observe that its independent learner has a significant downturn towards the end of its training, which could make the points before candidates for early stopping.

A3C achieves the highest performance with a very clear edge over every other algo-

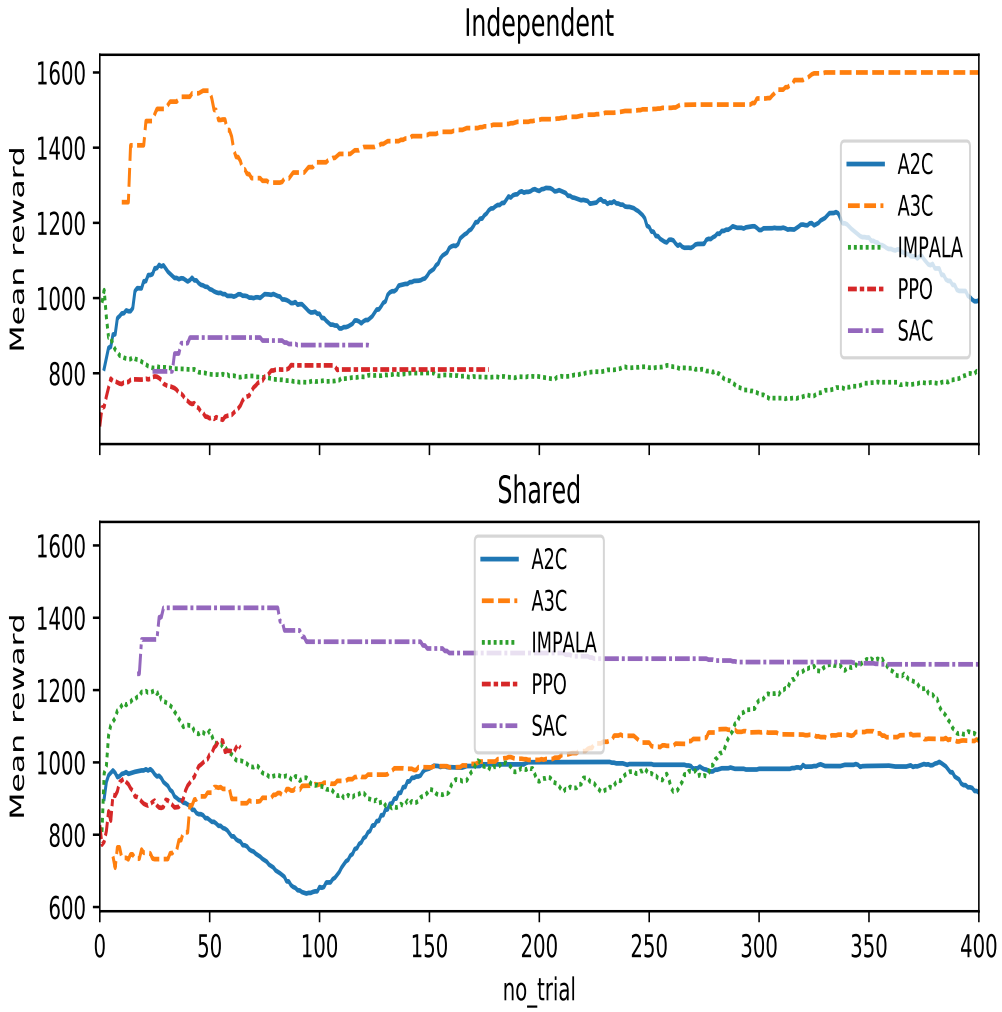

Figure 4.11. *Space Invaders average reward over time*

rithm in its independent variant, while its shared variant is only able to achieve mediocre performance, though far from the worst. This most likely disproves our earlier assumption that asynchronous updating and Atari games do not mix and match, though it does propose that it works better with independent learners.

IMPALA significantly under performs, achieving no learning and plateauing almost immediately in its independent configuration, while it does markedly better as a shared learner, being the runner-up and only second to SAC. This performance, along with its performance in Entombed, makes for a somewhat convincing case that it is IMPALA that does not work well at all with independent learning in complex environments like Atari, while with a more sophisticated architecture that explicitly allows for communication it is able to perform well.

The trend of PPO disappointing with its performance continues here, even in an environment that allowed for some pretty high scores for several other algorithms, particularly with its independent variant which also flat-lined along with IMPALA, almost merging with

it. Its shared variant does slightly better, with a somewhat increasing curve, but again one that took a very long time to train and was thus cut short for computational budget reasons. It is conceivable that were for this trend to continue, with a much larger budget, it could reach a competitive score after all, though that is far from certain, as we have already seen instances before of peaks followed by longer, permanent lows.

SAC is again an interesting case, with its independent learner being even slower to train than PPO's and achieving similarly disappointing performance, and its shared learner starting at a high point, but steadily continuing to retain the highest performance. Nonetheless, the overall trend seems to be diminishing, allowing for the scenario where it outperformed the other algorithms in part due to chance, such as lucky initialisation.

The last game we will be looking into in the vein of shared vs independent learning is a game unique to PettingZoo called Cooperative Pong, part of its unique set of Butterfly environments. As mentioned earlier, those games were created with the use of PyGame with visual Atari spaces, so in effect they function in a very similar manner to the original ALE games.They have been designed explicitly to require a high degree of coordination and thus are very challenging to learn without very specialized techniques.

We chose the Cooperative Pong environment for its relatively lightweight computational overhead and because it seems to require its agents to emerge with quite novel behaviors to succeed in it. Its state space consists of images, just like Atari's, with half of the screen available to each agent. The last figure 4.12 however paints a different, bleaker, picture of the performance of our algorithms compared to all other previous environments and particularly Atari ones, in which we saw several instances of promising performances.

In this environment, there is not really much to discuss. All algorithms, regardless of the variant of learning utilized, are below the baseline and almost equally matched in their sub-par performances. There are slight differences in the length of the tails of the various distributions, all of which significantly above the baseline, with A2C shared in particular having the far shorter one, meaning that it never even accidentally managed to reach any high rewards. We also observe that the SAC variants have some mass of returns over the baseline, perhaps implying that they could have benefited from a significantly expanded training regime, though that is a far from certain conclusion.

These results in general show that the emergent behaviors required to solve this particular environment were not even remotely close to being discovered by the configurations we tried, even after extensive hyperparameter tuning of a rather larger number of parameters for each algorithm, clearly illustrating the very high degree of difficulty inherent in a setting where a degree of "novelty" is crucial. This illustrates the need for more advanced MARL algorithms to be utilized, that allow the agents to communicate more effectively with one another or for more advanced exploration techniques to be involved.

We will now take a look at our final learning curves plot in 4.13, in case we get any additional insights to help us diagnose this apparent generalisation failure. These plots indeed convey a bit more information on how each algorithm fared during its training, compared to our rather uninformative violin plots that all showed a uniform performance.

Interestingly, A2C again has a variant with extreme variance, as has been evidenced

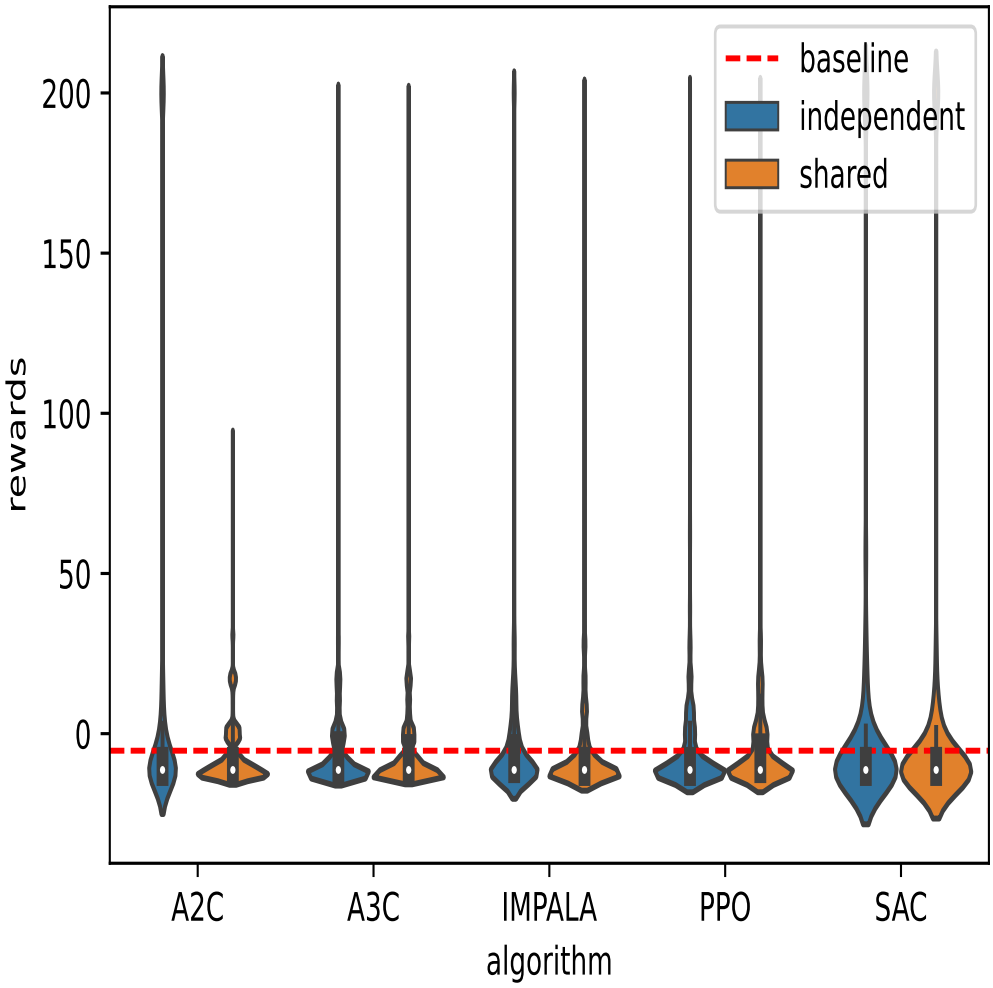

Figure 4.12. *Cooperative Pong scores vs baseline*

several times already, predominantly in MPE environments: Its independent learner has much larger variance in its mean rewards received during training, with the only comparable other algorithm being IMPALA. This does however mean that it also managed to score some pretty high scores, compared to most other competitors, though apparently it was not robust enough to retain them and instead kept swerving up and down. Its shared variant is indifferent, on the other hand, with small variance and about average performance.

A3C also seems markedly indifferent, though with stronger variance on its shared variant, where it did however manage to outperform slightly for almost the entire duration of the training its A2C cousin. Other than that, in both instances its performance remains stagnant, never showing any signs of receiving any better rewards than apparently only the lowest ones possible, in spite of it running a large number of trials too.

IMPALA initially shows some promise, particularly with its independent variant, but it quickly levels up after a brief period of high returns, comparable with those of A2C,

and then stays low for the rest of its training. Its shared variant never even experiences an initial boost, which could mean that the good initial performance of the former is again due to lucky initial conditions rather than anything else and learning did not grow afterwards.

PPO is again irrelevant, being both extremely slow in its execution and very mediocre when it does actually run, although it shows a bit of promise in its shared learning configuration, again implying that with a very large budget it could be effective, though that buget was not available to us for this project.

Finally, SAC seems to be by far the most promising algorithm in this apparently extremely challenging environment, an insight we also got a hint of from our violin plots. It seems to learn the fastest in both its iterations, with its independent variant rapidly closing the gap with A2C and quite possibly only limited by run-time to achieve the best performance, and is already clearly the winner in its shared setting, having one of the few clear instances we have seen so far of an upwards trending line, indicating that it was in fact learning the environment.

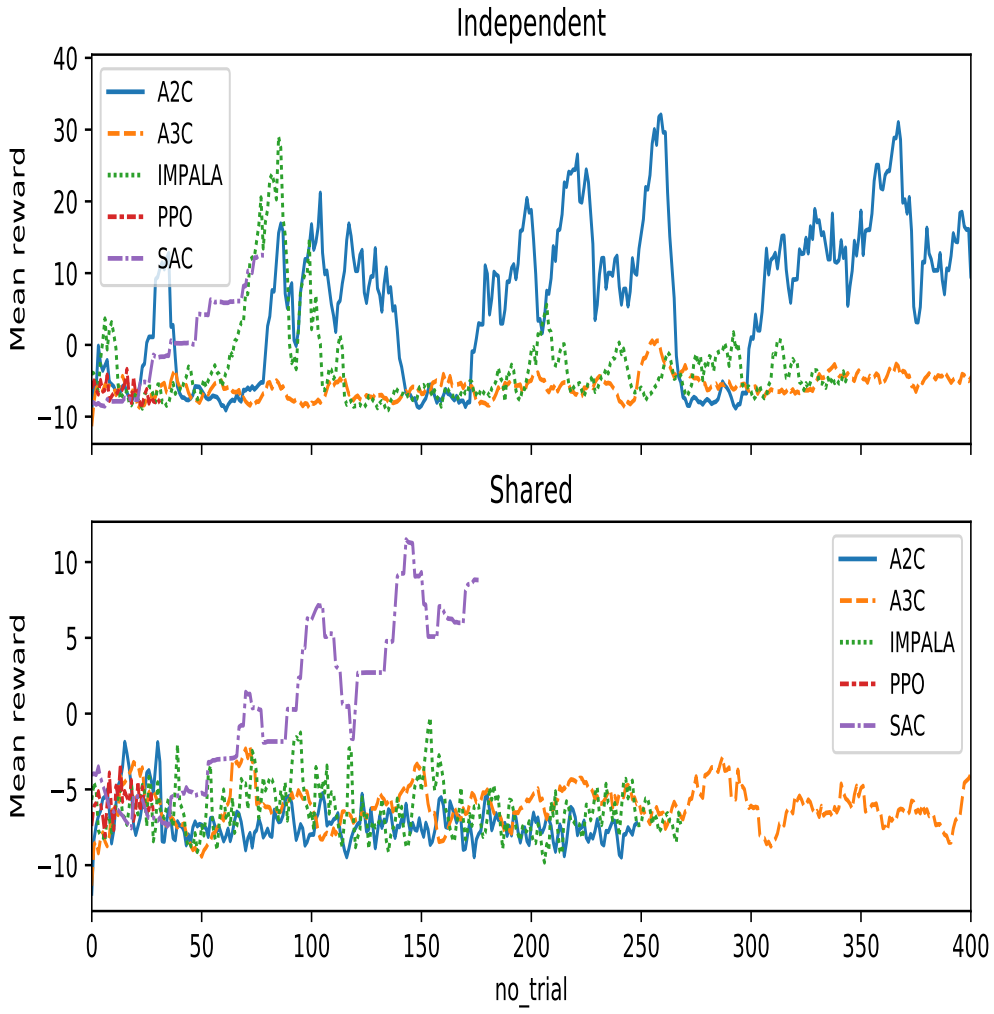

Figure 4.13. *Cooperative Pong scores average reward over time*

Chapter 5

Conclusion

In this work we took a tour around the field of Multi-Agent Deep Reinforcement Learning and the main challenges inevitably arising in real situations, making MADRL especially challenging. We then looked into algorithms and methodologies proposed to mitigate some of the issues in multi-agent settings and recapped the formulation of Markov games upon which most of these algorithms are built on. We then chose several cutting edge algorithms to run several experiments with on a variety of challenging environments using both Independent and Shared architectures, as despite their simplicity in concept and theoretical weaknesses do in fact often achieve state-of-the-art performance in both benchmark studies as well as application-wise, and compared their performances after running them through extensive hyperparameter tuning processes, to varying degrees of success.

We discovered that generally our relatively simple architectures could do a good job dealing with even complex environments, like Space Invaders, and that often both approaches were viable with some configurations. Still, we encountered several instances, most notably the Cooperative Pong environment, where both approaches were incapable of making any significant process, highlighting the need for more advanced, dedicated architectures that were designed explicitly for encouraging agent-to-agent interaction, like QMIX, to solve them.

There are several future research directions that are, or could be, extremely promising for MARL. One potentially excellent idea would be to imbue these algorithms with causal inference abilities, that would enable them to more skillfully understand their environment by building causal models instead of purely associative ones. While the intersection of Machine learning and Causal inference is still a nascent and under-explored field, there has been growing interest in this area, even by deep learning champions, as in [46]. Building causal models will also significantly aid in the generalisation abilities of these algorithms, often a very significant problem in RL research.

Another interesting approach taken from single agent settings would be to adopt procedural generation for environments as in ProgGen, that would more effectively test the generalisation abilities of the algorithms. This is especially important in RL where the more traditional train-test splits and cross-validation schemes are not applicable, and it can be hard to evaluate an algorithm in truly unseen situations.

Furthermore, a common problem of many modern MARL algorithms is their difficulty

in effectively dealing with incomplete and uncertain observations, partly because of issues with their scalability. Incorporating domain knowledge to agents can greatly aid in their developing effective solutions in realistic tasks, for example via informative reward functions which also reward promising behaviors and not just achievements, imitation or curriculum learning and hierarchical approaches, that as mentioned earlier, can effectively be formulated as a kind of multi-agent learning.

Another necessary component of work required for more effective MARL is the development of rigorous formulations for dynamic tasks with dynamic, adaptive agents, that provides for more formal guarantees about performance and stability. Generally, a good MARL objective should worry about both convergence and about effective coordination between agents.

Lastly, the application of game-theoretic approaches to consider the dynamics of the environment, besides only that of agents, could also provide important insight in the analysis of the learning process in its entirety, and allow for easier incorporation of prior knowledge for imperfect observations.

A

Appendix

In the appendix, we will detail the optimal values found per algorithm and per game, following the intensive hyperparameter tuning process undertaken with the help of RLLib's *HEBO* algorithm. We will also briefly discuss what a few of these hyperparameters do since there are several parameters unique to each algorithm, and it may not always be clear what their functionality is. It should be noted that only a small subset of the available options were searched for computational efficiency reasons, as searching a very large number of hyperparameters would require many more iterations to converge to local optima.

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.001	0.0044	0.0012	0.0025	0.0049	0.0043
N_2	0.0996	0.0996	0.2747	0.0996	0.0996	0.1872
N_3	25	75	25	50	25	100
N_4	512	512	128	256	512	256
N_5	T	F	F	F	F	F
N_6	0.0099	0.0107	0.0225	0.0104	0.0116	0.004
N_7	0.0232	0.0003	0.0225	0.0171	0.003	0.011

Table A.1. *Hyperparameter optimal values - SAC Independent*

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.0008	0.0042	0.0026	0.0012	0.0037	0.0007
N_2	0.0996	0.0996	0.0996	0.2747	0.6249	0.0996
N_3	75	10	100	25	75	10
N_4	64	512	128	128	256	256
N_5	T	F	F	F	T	F
N_6	0.0173	0.0077	0.0243	0.0225	0.0077	0.0104
N_7	0.0260	0.0042	0.0258	0.0225	0.0077	0.0050

Table A.2. *Hyperparameter optimal values - SAC Shared*

Where, in A.3

- "Gamma" is the discount factor of the MDP.
- "Target networks update frequency" is at how many steps we update the target network.

Parameter	Explanation
N_1	Learning Rate
N_2	Gamma (discount rate)
N_3	Target networks update frequency
N_4	Batch Size
N_5	Use prioritized replay? (T/F)
N_6	Actor leaning rate
N_7	Critic learning rate

Table A.3. *Hyperparameters - SAC*

- "Use prioritized replay?" is a Boolean value indicating whether or not to use prioritized experience replay buffer.

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.0071	0.0068	0.0042	0.0083	0.0061	0.0075
N_2	0.7634	0.0996	0.0996	0.5405	0.0996	0.6249
N_3	10	21	18	12	6	10
N_4	0.9067	0.9848	0.9719	0.9014	0.9227	0.9249
N_5	0.9319	0.5257	0.5288	0.7247	0.7483	0.8249
N_6	0.3132	0.4392	0.5480	0.9076	0.4522	0.8249
N_7	0.0097	0.0088	0.0016	0.0011	0.0005	0.0025

Table A.4. *Hyperparameter optimal values - PPO Independent*

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.0020	0.0025	0.0074	0.0061	0.0006	0.0024
N_2	0.0996	0.2747	0.0996	0.6084	0.0996	0.0996
N_3	29	22	9	4	20	11
N_4	0.9727	0.9750	0.9721	0.9772	0.9367	0.9375
N_5	0.9866	0.4750	0.7479	0.9938	0.3996	0.4225
N_6	0.4549	0.4750	0.8714	0.4082	0.9779	0.5061
N_7	0.0059	0.0075	0.0004	0.0064	0.0013	0.0043

Table A.5. *Hyperparameter optimal values - PPO Shared*

Parameter	Explanation
N_1	Learning Rate
N_2	Gamma (discount rate)
N_3	SGD Iterations
N_4	Lambda (GAE parameter)
N_5	KL-Coefficient
N_6	KL-Target
N_7	Entropy Coefficient

Table A.6. *Hyperparameters - PPO*

Where, in A.6

- "Lambda" is the GAE parameter, that together with "Gamma" control the Bias-Variance trade off of the trajectories and can be viewed as a form of reward shaping.
- "Entropy Coefficient" is the coefficient of the entropy regularizer, which is multiplied by the maximum possible entropy and added to the loss, helping prevent premature convergence of one action probability dominating the policy and preventing exploration.
- "KL-Coefficient" is the initial coefficient for KL-Divergence.
- "KL-Target" is the target value for KL-Divergence. The last two parameters are useful for the KL-Penalty implementation that prevents the policy from updating too abruptly and leading to an unrecoverable performance collapse.

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.0027	0.0021	0.0024	0.0080	0.0003	0.0071
N_2	0.0996	0.0996	0.0996	0.0996	0.0996	0.0996
N_3	0.999	0.9046	0.9818	0.9733	0.9192	0.9818
N_4	0.8278	0.7952	0.8115	0.3586	0.9164	0.8442
N_5	0.0035	0.0195	0.0089	0.0995	0.0025	0.0708

Table A.7. *Hyperparameter optimal values - A2C Independent*

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.0083	0.0009	0.0080	0.0066	0.0050	0.0050
N_2	0.0996	0.0996	0.0996	0.0996	0.0996	0.4498
N_3	0.9275	0.9768	0.9304	0.9822	0.9561	0.9499
N_4	0.9148	0.9853	0.4750	0.3582	0.3197	0.6499
N_5	0.0192	0.0578	0.0744	0.0204	0.0099	0.0500

Table A.8. *Hyperparameter optimal values - A2C Shared*

Parameter	Explanation
N_1	Learning Rate
N_2	Gamma
N_3	Lambda
N_4	Value function Loss coeff
N_5	Entropy Coefficient

Table A.9. *Hyperparameters - A2C*

Where, in A.15

- "Replay proportion fraction" is the fraction of the samples that will be replayed with respect to the new data samples.
- "Replay buffer slots" is the number of sample batches to store for replay.

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	$1.2E^{-5}$	$1E^{-5}$	$6.6E^{-5}$	0.0010	0.0085	0.0050
N_2	0.0996	0.8000	0.0996	0.0996	0.1896	0.0996
N_3	0.9049	0.8999	0.9080	0.9011	0.9203	0.9509
N_4	0.6131	0.3000	0.8349	0.3029	0.3855	0.6499
N_5	0.0514	0.0001	0.0723	0.0963	0.0256	0.0487

Table A.10. *Hyperparameter optimal values - A3C Independent*

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.0006	$8.31E^{-5}$	0.0002	0.0004	$2.48E^{-5}$	0.0037
N_2	0.0996	0.0996	0.0996	0.0996	0.0996	0.0996
N_3	0.9664	0.9121	0.900	0.9045	0.9303	0.9003
N_4	0.3000	0.6500	0.9644	0.3222	0.3397	0.9970
N_5	0.0732	0.0002	0.0001	0.0198	0.0040	0.0948

Table A.11. *Hyperparameter optimal values - A3C Shared*

Parameter	Explanation
N_1	Learning Rate
N_2	Gamma
N_3	Lambda
N_4	Value function Loss coeff
N_5	Entropy Coefficient

Table A.12. *Hyperparameters - A3C*

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.0003	0.0001	0.0003	1^{-6}	0.0003	0.0004
N_2	0.5373	0.0996	0.5373	0.8000	0.0996	0.0996
N_3	0.5500	0.6611	0.5500	0.3000	0.33886	0.3516
N_4	0.0876	0.0305	0.8762	0.001	0.0015	0.0262
N_5	0.1125	0.0287	0.1125	0	0.2700	0.0045
N_6	1	1	1	1	4	3

Table A.13. *Hyperparameter optimal values - IMPALA Independent*

Param/Env	Ref	Spread	S-L	Entombed	Space.Inv	Coop.Pong
N_1	0.0007	0.0001	0.0003	7.33^{-6}	0.0007	0.0010
N_2	0.6249	0.0996	0.0996	0.0996	0.6249	0.0996
N_3	0.4000	0.6268	0.5231	0.6625	0.4000	0.4736
N_4	0.0257	0.0925	0.0468	0.0831	0.0257	0.0126
N_5	0.2250	0.2946	0.0634	0.0016	0.2250	0.0177
N_6	3	3	4	4	3	3

Table A.14. *Hyperparameter optimal values - IMPALA Shared*

Parameter (SAC)	Explanation
N_1	Learning Rate
N_2	Gamma (discount rate)
N_3	Value function Loss coeff
N_4	Entropy coeff
N_5	Replay proportion fraction
N_6	Replay buffer slots

Table A.15. *Hyperparameters - IMPALA*

Bibliography

[1] Abadi, M., Isard, M., and Murray, D. G. A computational model for tensorflow: An introduction. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (New York, NY, USA, 2017), MAPL 2017, Association for Computing Machinery, p. 1–7.

[2] Architectures, A.-l., Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., and Jun, L. G. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures.

[3] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research 47* (jun 2013), 253–279.

[4] Box, G. E. P., and Cox, D. R. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological* (1964), 211–252.

[5] Brittain, M., and Wei, P. Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement Learning Approach.

[6] Buşoniu, L., Babuška, R., and De Schutter, B. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews 38*, 2 (2008), 156–172.

[7] Canese, L., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., and Spanò, S. Multi-agent reinforcement learning: A review of challenges and applications, 2021.

[8] Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches, 2014.

[9] Christodoulou, P. Soft Actor-Critic for Discrete Action Settings. 1–7.

[10] Claus, C., and Boutilier, C. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems.

[11] Cohen, A., Teng, E., Berges, V.-P., Dong, R.-P., Henry, H., Mattar, M., Zook, A., and Ganguly, S. On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning.

[12] Cowen-rivers, A. I., Wang, Z., and Maravel, A. M. An Empirical Study of Assumptions in Bayesian Optimisation. 1–15.

[13] Dayan, P. Feudal Reinforcement Learning. 271–278.

[14] Fernández, F., and Parker, L. E. Learning in large cooperative multi-robot domains. *International Journal of Robotics Research 16*, 4 (1997), 217–226.

[15] Foerster, J., Nardell, N., Farquhar, G., Afouras, T., Torr, P. H., Kohli, P., and Whiteson, S. Stabilising experience replay for deep multi-agent reinforcement learning. *34th International Conference on Machine Learning, ICML 2017 3* (2017), 1879–1888.

[16] Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (2018), 2974–2982.

[17] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *35th International Conference on Machine Learning, ICML 2018 5* (2018), 2976–2989.

[18] Hsu, C. C.-y., and Mendler-dünner, C. Revisiting Design Choices in Proximal Policy Optimization.

[19] Hu, J., and Wellman, M. P. Nash Q-Learning for General-Sum Stochastic Games. 1039–1069.

[20] Iqbal, S., and Sha, F. Actor-attention-critic for multi-agent reinforcement learning. *36th International Conference on Machine Learning, ICML 2019 2019-June* (2019), 5261–5270.

[21] Ishiwaka, Y., Sato, T., and Kakazu, Y. An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning. 245–256.

[22] Kok, J. R. Utile Coordination : Learning interdependencies among cooperative agents 1.

[23] Leibo, J. Z., and Lanctot, M. Multi-agent Reinforcement Learning in Sequential Social Dilemmas.

[24] Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., and Talwalkar, A. A System for Massively Parallel Hyperparameter Tuning.

[25] Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. RLlib: Abstractions for distributed reinforcement learning. In *35th International Conference on Machine Learning, ICML 2018* (2018), vol. 7, pp. 4768–4780.

[26] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. Tune: A Research Platform for Distributed Model Selection and Training.

[27] Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. *Machine Learning Proceedings 1994* (1994), 157–163.

[28] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems 2017-Decem* (2017), 6380–6391.

[29] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. H. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *ArXiv abs/1709.06009* (2018).

[30] Mataric, M. J. Reward functions for accelerated learning. In *Machine Learning Proceedings 1994*, W. W. Cohen and H. Hirsh, Eds. Morgan Kaufmann, San Francisco (CA), 1994, pp. 181–189.

[31] Matignon, L., Laurent, G. J., Fort-piat, N. L., Matignon, L., Laurent, G. J., Le, N., Independent, F.-p., Matignon, L., Laurent, G. J., and Fort-piat, N. L. E. Independent reinforcement learners in cooperative Markov games : a survey regarding coordination problems . To cite this version : HAL Id : hal-00720669 Independent reinforcement learners in cooperative Markov games : a survey regarding coordination prob- lems.

[32] Mesnard, T., Weber, T., Viola, F., Thakoor, S., Saade, A., Harutyunyan, A., Dabney, W., Stepleton, T., Heess, N., Guez, A., Moulines, É., Hutter, M., Buesing, L., and Munos, R. Counterfactual Credit Assignment in Model-Free Reinforcement Learning.

[33] Minsky, M. Steps toward artificial intelligence. *Proceedings of the IRE 49*, 1 (1961), 8–30.

[34] Mnih, V., Badia, A. P., Mirza, L., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016 4* (2016), 2850–2869.

[35] 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. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529–533.

[36] Monahan, G. E. State of the Art — A Survey of Partially Observable Markov Decision Processes : Theory , Models , and Algorithms.

[37] Mordatch, I., and Abbeel, P. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908* (2017).

[38] Oliehoek, F. A., and Amato, C. *A Concise Introduction to Decentralized POMDPs*. 2015.

[39] Panait, L., and Luke, S. Cooperative Multi-Agent Learning: The State of the Art. 1–39.

[40] Parisotto, E., Song, H. F., Rae, J. W., Pascanu, R., Gulcehre, C., Jayakumar, S. M., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M. M., Heess, N., and Hadsell, R. Stabilizing transformers for reinforcement learning. *37th International Conference on Machine Learning, ICML 2020 PartF168147-10* (2020), 7443–7454.

[41] Pham, H. X., La, H. M., Feil-seifer, D., and Nefian, A. Cooperative and Distributed Reinforcement Learning of Drones for Field Coverage.

[42] Prasad, A. Multi-agent Deep Reinforcement Learning for Zero Energy Communities.

[43] Qie, H. A. N., Shi, D., Shen, T., Xu, X., Li, Y., and Wang, L. Joint Optimization of Multi-UAV Target Assignment and Path Planning Based on Multi-Agent Reinforcement Learning. *IEEE Access 7* (2019), 146264–146272.

[44] Rashid, T., Samvelyan, M., and Schroeder, C. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning.

[45] Roesch, M., Linder, C., Zimmermann, R., Rudolf, A., Hohmann, A., and Reinhart, G. Smart grid for industry using multi-agent reinforcement learning. *Applied Sciences (Switzerland) 10*, 19 (2020), 1–20.

[46] Scholkopf, B., Locatello, F., Bauer, S., Ke, N. R., Kalchbrenner, N., Goyal, A., and Bengio, Y. Toward Causal Representation Learning. *Proceedings of the IEEE 109*, 5 (2021), 612–634.

[47] Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. Trust region policy optimization. *32nd International Conference on Machine Learning, ICML 2015 3* (2015), 1889–1897.

[48] Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016), pp. 1–14.

[49] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms. 1–12.

[50] Snoek, B. J., Larochelle, H., and Adams, R. P. PRACTICAL BAYESIAN OPTIMIZATION OF MACHINE LEARNING. 1–12.

[51] Stanley, H. E. Introduction to phase transitions and critical phenomena. *American Journal of Physics 40* (1971), 927–928.

[52] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* (2018), vol. 3, pp. 2085–2087.

[53] Terry, J. K., and Black, B. Multiplayer support for the arcade learning environment. *arXiv preprint arXiv:2009.09341* (2020).

[54] Terry, J. K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C., Williams, N. L., Lokesh, Y., and Ravi, P. PettingZoo: Gym for Multi-Agent Reinforcement Learning.

[55] Terry, J. K., Grammel, N., Son, S., and Black, B. Parameter Sharing For Heterogeneous Agents in Multi-Agent Reinforcement Learning.

[56] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. *IEEE International Conference on Intelligent Robots and Systems 2017-September* (2017), 23–30.

[57] Touzet, C. F. Robot Awareness in Cooperative Mobile Robot Learning. 87–97.

[58] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems* (2017), vol. 2017-Decem, pp. 5999–6009.

[59] Weng, L. Policy gradient algorithms. *lilianweng.github.io* (2018).

[60] Wu, Y., Mansimov, E., Liao, S., Grosse, R., and Ba, J. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *Advances in Neural Information Processing Systems* (2017), vol. 2017-Decem, pp. 5280–5289.

[61] Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. Mean field multi-agent reinforcement learning. *35th International Conference on Machine Learning, ICML 2018 12* (2018), 8869–8886.

[62] Yeo, I.-K., and Johnson, R. A new family of power transformations to improve normality or symmetry. *Biometrika 87* (12 2000).

[63] Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., and Wu, Y. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games.

List of Abbreviations

MARL	Multi-Agent Reinforcement Learning
MADRL	Multi-Agent Deep Reinforcement Learning
MADDPG	Multi-Agent Deep Deterministic Policy Gradients
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
SAC	Soft Actor-Critic
IMPALA	Importance Weighted Actor-Learner Architecture
PPO	Proximal Policy Optimization
COMA	Counterfactual Multi-Agent
VDN	Value-Decomposition Networks
MAPPO	Multi-Agent Proximal Policy Optimization
ACKTR	Actor Critic using Kronecker-Factored Trust Region
MA-POCA	Multi-Agent Posthumous Credit Assignment
CTDE	Centralized Training and Decentralized Execution
VD	Value Decomposition
MPE	Multi Particle Environment
GAE	Generalized Advantage Estimation
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
Dec-POMDP	Decentralized Partially Observable Markov Decision Process
POSG	Partially Observable Stochastic Game
SGD	Stochastic Gradient Descent
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
EFG	Extensive Form Games
ALE	Arcade Learning Environment