UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

MASTER THESIS IN CONTROL SYSTEMS ENGINEERING

CONTINUAL MULTI-AGENT REINFORCEMENT LEARNING APPLIED IN COOPERATIVE GAMES

Supervisor
Gian Antonio Susto

Master candidate
Olivia Casadei
2062673

ACADEMIC YEAR 2023-2024

November 25, 2024

Uomini forti, destini forti
Uomini deboli, destini deboli

Abstract

Reinforcement Learning combined with Continual Learning offers the potentiality to explore many of the challenges of the modern era in almost every field of the society, from financial and portfolio management to human-robot cooperation in the industry 4.0 but also in the healthcare and in smart grids and energy management. The intersection of reinforcement learning and continual learning in the multi-agent scenario has not been extensively investigated. It is therefore valuable to explore this crossover. In particular, this thesis analyzes the application of continual learning techniques to enhance the adaptability of multi-agent reinforcement learning systems in dynamic and cooperative environments. Among the issues that continual learning aims at overcoming stands catastrophic forgetting, namely the fact that agents lose previously acquired knowledge when learning new tasks or adapting to changing scenarios. This research aims to understand if it is possible to overcome this limitation, in particular in the scenario of fully cooperative games, by implementing and testing some architectures that allow agents to seamlessly transition between different teams while maintaining the ability to effectively perform in previously encountered ones.

Contents

List of Figures

Chapter 1

Introduction

If free will is what we consider essential to human nature, bestowing it to machines has been a pivotal step in the creation of an artificial intelligence that increasingly mirrors human capabilities. Reinforcement learning comes into being from this simple yet revolutionary idea: giving a machine free will, the ability to make decisions and to act and to learn from mistakes. The agent formalized by Sutton and Barto, capable of effectively interacting in an environment and successfully completing simple tasks, has made significant progress since the 1980s. The advent of GPUs and the resulting enhancement of neural network computing power enabled this agent to take on increasingly larger challenges, rising to compete against humans; in 2016 DeepMind's AlphaGo managed to prevail in a game until then considered impossible for artificial intelligence. In recent years, generative models have started to emulate abilities once believed to be exclusive to humans, such as creativity, deduction, and abstraction. In this journey toward living beings, quoting Darwin, "It is not the strongest species that survives, but the one that is most adaptable to change." it naturally follows to introduce the new challenge that artificial agents face: the ability to cooperate and to learn how to learn.

Continual Learning aims to fill this gap by setting as its goal to address the problem of catastrophic forgetting, which currently greatly limits the applicability of agents involved in sequential tasks that must simultaneously master more than one task and be able to shift from one to another without wasting time and resources relearning what they already apprehended.

Effectively interacting with its peers to successfully cooperate and achieve common goals is the objective of multi-agent reinforcement learning, this task faces challenges like the non-stationarity, credit assignment and scalability.

The motivation for this thesis arises from this context: to analyze, within the safe and well-established RL environment of games, an initial simple ap-

proach to continual learning techniques aimed at enhancing the cooperation capabilities of multi-agent systems. In particular, the focus will be on exploring ways to make it possible for an agent involved in a collaborative game to seamlessly switch teams and how to address the challenge of catastrophic forgetting.

The thesis is structured as follows: chapters 2, and 3 explore the theoretical concepts of reinforcement learning and multi-agent reinforcement learning with emphasis on the PPO algorithm which is currently one of the most popular and also the one considered in this work. In chapter 4 there is a quick overview on the fundamentals of neural networks. Chapter 5 presents the continual learning problem and the architecture implemented to tackle problems similar to the one proposed by this thesis. The environments considered (MPE and SMAC) are presented in chapter 5 along with the proposed approach and implementations, while the discussion of the early results obtained is proposed in chapter 6.

Chapter 2

Reinforcement Learning

Reinforcement Learning (RL)[1] is a branch of Machine Learning (ML) whose aim it to train an agent capable of proficiently interacting with the environment it is place into and to learn a strategy to optimize some kind of rewards functional to the objective given.

Its peculiarity revolves around the concept of leaning on the go interacting with the environment by trial and error. The mathematics behind ML is combined to the ability inherited from the living creatures to learn in order to survive and evolve: the consequences of their interaction in a potentially perilous environment are remembered and used in order to take better choices in the future.

RL differs from both unsupervised and supervised learning, which are the two major paradigms of ML. With respect to the first, the agent does not have access to the "correct/incorrect" label associated to its action but rather it apprehends such concepts from the reward that it obtains. But RL is not analogous to the latter either as the final goal is not to learn the hidden structure of the data it gathers but to acquire experience and knowledge of the environment finalized to its interactions with it.
Moreover the time variable plays a crucial role in the RL problem: the data is not static but it is rather a succession of actions and consequences where positive rewards may come delayed, another thing that the agent must learn to deal with.

Reinforcement Learning has garnered significant interest due to its broad applicability across diverse fields. From artificial intelligence to operations research and control engineering, it offers a powerful framework for solving complex problems that involve decision-making in dynamic environments.

[1] The theory concepts in this chapters are taken from [7]

2.1 Key Concepts

Here are some formal definitions of the key concepts of RL:

- Rewards: they serve as the numerical feedback signal for the agent to discern which actions are beneficial or not. The strategy that the RL aims to learn is thus the one that maximizes its cumulative rewards - the expected return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=1}^{T} \gamma^k R_{t+k+1} \tag{2.1}$$

  where γ is a discount factor to determine the agent's foresight. The discount factor serves also to make the return finite in case of continuing tasks (where $T \to \infty$), while episodic ones - like games - have a finite number of steps after which they terminate and the environment is reset. The rewards can be deterministic or stochastic if associated with some randomness with which they can present themselves to the agent, which is the case in most real-world applications where the response of the environment is unpredictable.

- Policy: the policy defines the agent's behavior at any given time. It is a function mapping from the states of the environment to the actions that the agent should take in such states. Formally, it is defined by the conditional probability of taking action a in state s:

$$\pi(a|s) = P(A = a \mid S = s) \tag{2.2}$$

  A policy can be either deterministic or stochastic, in such case for each state it provides a probability distribution over the possible actions. The goal of a RL problem is to have the agent learn the optimal policy $\pi^\star$, namely the one that maximizes the cumulative return G_t.

- Model: A model in RL refers to the knowledge (given or acquired) and representation of the internal dynamics of the environment in which the agents are embedded. In detail, to characterize how the environment will respond to the agent's actions, two ingredients are necessary: the state transition and the reward models.

  The use or not of the model knowledge allows to differentiate between two types of approaches to RL: model free or model based. In model free the agent learns by interacting directly by trial and error. The great advantage of not needing knowledge of the model, which is often complex

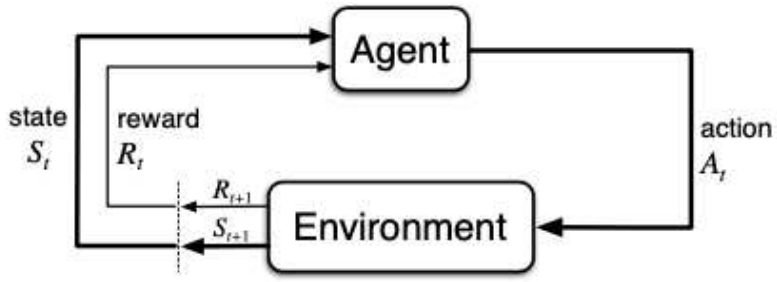


Figure 2.1: The agent-environment interaction in a MDP [1]

and difficult to learn if unknown, is contrasted with the dangers that it could expose itself or others if it roams freely in the environment. In model based the main challenge is to determine a representation of the model or an its approximation that is as reliable as possible. In any case, model estimation is often affected by uncertainty and assessing the sensibility of the system to errors is difficult, reasons why the model free approach is preferred.

2.2 MDPs

In order to model the process of sequential decision-making, the Markov Decision Process (MDP) is the ideal tool for this purpose: it is in fact a formal framework where states are affected by actions and with them the corresponding rewards, not only the immediate ones but also the future ones; its purpose is thus to find the best action sequence namely the optimal policy for the agent. Formally it is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$, $\mathcal{A}$ are respectively the finite sets of all the possible states and actions for the agent, $\mathcal{R}$ is the **reward function** and defines how the rewards are associated to states and actions,

$$\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] \tag{2.3}$$

while $\mathcal{P}$ is the **transition probability matrix** i.e. how the environment progresses from one state to another when a specific action is taken in probability terms, with each entry:

$$\mathcal{P}_{ss'}^a = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right] \tag{2.4}$$

Together, they completely identify the environment dynamics. $\gamma \in [0, 1]$ is a discount factor that determines how future rewards are weighted compared to immediate ones.

To determine the best policy π^*, it is necessary to define the **value-**

function for the state s, denoted as $v_\pi(s)$ which is the expected return of following π from the state s and it is an indication of how favorable it is for the agent to be in a particular state. Similarly, a function is needed to describe how advantageous it is to take a specific action in a given state, which leads to the definition of the **action-value function** q_π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \text{ for all } s \in \mathcal{S} \tag{2.5}$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right], \text{ for all } s \in \mathcal{S}\ ,\ a \in \mathcal{A}(s) \tag{2.6}$$

Note that both the v and q functions can be expressed in a recursive way through the definition of G_t 2.1:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \tag{2.7}$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a\right] \tag{2.8}$$

The last definition that needs to be provided is the one of optimality. It has been stated that the goal of the agent is to find the best way to act in the MDP namely the optimal policy π^*. To do so, the first step is to provide a sorting over policies: given two policies π_1 and π_2:

$$\pi_1 \geq \pi_2 \iff v_{\pi_1}(s) \geq v_{\pi_2}(s) \quad \forall s \in \mathcal{S} \tag{2.9}$$

The optimal policy is hence the one yielding the highest value-function and may not be unique, if more than one exist, they all share the same optimal value and action-value functions, defined respectively:

$$v^*(s) \doteq \max_\pi v_\pi(s), \text{ for all } s \in \mathcal{S} \tag{2.10}$$

$$q*(s,a) \doteq \max_\pi q_\pi(s,a), \text{ for all } s \in \mathcal{S},\ a \in \mathcal{A}(s). \tag{2.11}$$

2.3 Policy Gradient

There are two main approaches to solve the RL problem. The first involves leveraging action value based methods, which rely on the calculation or estimation of the V and Q functions to maximize returns using dynamic programming techniques, from which the policy is obtained indirectly. However, these methods are not suitable for real-world scenarios, since the actions and states space is often high dimensional, and to learn the approximation of the V and Q functions can be computationally expensive and it may lead to fault for its

inaccuracy.

The other main class of algorithms are policy based, where the idea is to go directly after the policy through gradient based optimization techniques. This provides better convergence properties, the capability to work with large-dimensional or continuous action spaces, and in addition allows stochastic policies to be learned. As a consequence, to perform the required optimization, the policy to be learned needs to be expressed in a parametric form such as a function $\pi_\theta(s, a)$ parameterized in θ - for example, the weights of a neural network, and its goodness is quantified according to a cost function $J(\theta)$. In the episodic case, the cost function is defined as

$$J(\theta) \doteq v_{\pi_\theta}(s_0) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [r(\tau)] \tag{2.12}$$

where p_θ and r are the transition probability and reward functions, and τ represents the possible trajectories that can be originated by s_0.

The optimization problem is solved by ascending the gradient of the policy with respect to θ: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$. In the computation of $\nabla_\theta J(\theta)$, the parameters influence the action selection and the distribution of states.

Stating explicitly the 2.12 and applying the gradient to $J(\theta)$ leads to

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau \tag{2.13}$$

where the $\nabla_\theta p_\theta(\tau)$ is not always possible to be calculated directly if the environment dynamics are not available, as is often the case. To work around the problem and to avoid calculating the derivatives of the transition probability, the Policy Gradient Theorem [8] offers an analytic expression for the gradient of the objective function:

$$\nabla_\theta J(\theta) \propto \sum_{s \in \mathcal{S}} \mu_\pi(s) \sum_{a \in \mathcal{A}} q_\pi(s, a) \nabla_\theta \pi_\theta(a|s) \tag{2.14}$$

where μ_π is the state distribution.

Now to estimate the gradient it is only necessary to have a formulation that contains samples from states and actions and can be obtained by 2.14, leading to

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \tag{2.15}$$

$$= \mathbb{E}_\pi \left[G_t \nabla_\theta \ln \pi(A_t|S_t, \theta) \right], \text{ (from 2.6)} \tag{2.16}$$

The update rule for θ that follows the 2.16 is the one adopted by the

REINFORCE algorithm and may introduce high variance. To make up for this, a baseline is introduced with the precaution that it does not depends on a. A reasonable to choice could be hence to adopt the value function or rather its estimate $\hat{v}(s, \phi)$, that inserted in the parameters update results in

$$\theta_{t+1} \doteq \theta_t + \alpha(G_t - \hat{v}(S_t; \phi)) \tag{2.17}$$

In this way, the estimate of the value function regards only the first state of each transition and hence does not provide insight into the effectiveness of the action. It could then be estimated directly $\hat{q}_\pi(s, a; \psi)$, from 2.15 by optimizing another set of parameters ψ. Two entities are now present: one to optimize the parameters responsible for the policy and another to suggest the direction for this optimization based on the estimate of the action-value function. Therefore, this strategy is called actor-critic.
Similarly to 2.17, the estimate of the state function can be leveraged as a baseline to reduce variance for $\hat{q}_\pi$, defining the **Advantage Function**, that quantifies how much better (or worse) the specific action a is compared to the average action that the agent could take.

$$A_{\pi_\theta}(s, a) = q_{\pi_\theta}(s, a) - v_{\pi_\theta}(s) \tag{2.18}$$

Combining the 2.8, the 2.18 and the fact that functions approximations are involved, it is possible to compute the policy gradient as

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\hat{A}_{\pi_\theta}(S_t, A_t; \phi) \nabla_\theta \ln \pi(A_t | S_t, \theta) \right] \tag{2.19}$$

with

$$\hat{A}_{\pi_\theta}(S_t, A_t; \phi) = R_t + \gamma \hat{v}(S_{t+1}; \phi) - \hat{v}(S_t; \phi) \tag{2.20}$$

2.3.1 Proximal Policy Optimization (PPO)

PPO, proposed by Schulman et al. in 2016 [9], leverages the policy gradient methods and is currently one of the most popular RL algorithms due to its robustness and sample efficiency compared to other actor-critic algorithms. Using the objective function whose gradient coincides to 2.19, it has been observed that the resulting policy changes drastically between updates, leading to instability in the learning process. To address this, the Trust Region Policy Optimization (TRPO) [10] algorithm introduces a trust region constraint, limiting the extent of each policy update to ensure stability. However, this approach relies on complex optimization methods, making TRPO computationally expensive and impractically slow.

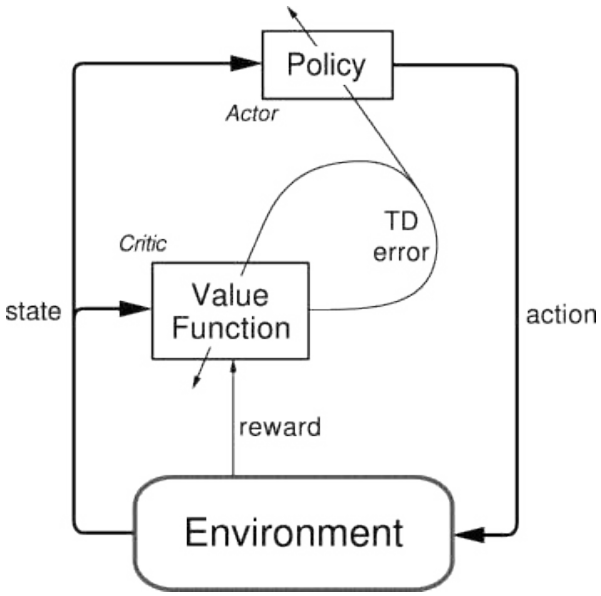


Figure 2.2: Actor Critic architecture

With respect to TRPO, where the deterrent for the new policy to be close to the old one is imposed by a hard constraint to which the objective function is subject, in [9] Schulman et al. demonstrate the effectiveness of introducing a clipped objective function instead.
The objective function to be differentiated to retrieve the policy gradient estimator is hence modified to ensure that consecutive policy updates are not too dissimilar. Defining the probability ratio as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{2.21}$$

the new objective function to be maximized becomes:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2.22}$$

with $\hat{A}_t$ a suitable advantage function. In particular, the policy is run for T time steps and hence the advantage function 2.18 is estimated over the T horizon with

$$\hat{A}_t^{GAE} = \sum_{n=0}^{T-1-t} (\gamma\lambda)^n \delta_{t+n} \tag{2.23}$$

where $\delta_t = R_t + \gamma V(S_{t+1}) - V(S_t)$ and λ is a smoothing parameter that balances the bias-variance trade off when the estimation are performed over a finite horizon $T > 1$. The loss function is then optimized with minibatch

stochastic gradient descent for K epochs.

Herby is reported the pseudocode for the PPO algorithm proposed in [9]:

Algorithm 1 PPO, Actor-Critic Style

for iteration $= 1, 2, \ldots$ **do**
  for actor $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in the environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L with respect to θ, using K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Chapter 3

MultiAgent RL

Multi-agent reinforcement learning (MARL)[1] involves training multiple autonomous agents within a shared environment, where each agent independently learns a policy to achieve specific goals. These agents can either collaborate toward a shared objective, such as cooperative tasks or team-based games, or compete in an adversarial setting, where agents' goals are in conflict. In either scenario, each agent must consider the actions and strategies of the others, adapting its behavior to maximize its own outcomes. To navigate these interactions effectively, agents often rely on some form of communication to share critical information or coordinate actions, particularly in cooperative settings.

MARL builds on principles of RL by placing multiple agents in a shared environment, where they learn through trial-and-error interactions, using the environment's feedbacks to refine their policies, like in single agent RL.

Conversely, each agent's actions influence not only its own rewards and trajectories, but also to the environment's dynamics and the other agents' opportunities, resulting in a non-stationarity of the environment, the more serious the more numerous the agents that populate it.

As a result, complexity is added to the learning process as agents must continually adapt to each other changing behavior besides optimizing their own policy.

On the other hand, MARL offers the advantage of *divide et impera*, such as in the case of an high dimensional decision making problem that can be decomposed into smaller subproblems managed by multiple agents that aim to solve a collaborative task.

[1] The theory concepts in this chapters are taken from [2]

3.1 First Steps

As in single agent RL, the MDP is the mathematical framework used to model the decision-making problem. For multi-agent RL this concept generalizes to the stochastic game. A stochastic game is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}_1, \dots \mathcal{R}_n \rangle$ where

- $\mathcal{S}$ is the discrete set of states
- $\mathcal{A}_1, \dots, \mathcal{A}_n$ are the discrete set of actions that agents can take. Together they form the joint action set $\mathbf{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$
- $\mathcal{P}$ is the state transition probability function
- $\mathcal{R}_1, \dots \mathcal{R}_n$ are the reward functions. If the agents share the same rewards, namely $\mathcal{R}_1 = \cdots = \mathcal{R}_n$ then the the game is fully cooperative, while if $\sum_i \mathcal{R}_i = 0$ the game is zero-sum or fully competitive. If neither happens, the game is said to be mixed.

The idea is to reduce the problem to a single-agent framework, using either central learning or independent learning. In **central learning**, a single centralized policy, denoted as π_c, dictates the actions for all agents collected in a single joint action $a = \{a_1, ..., a_n\} \in \mathbf{A}$. This approach requires that individual rewards are collected into a scalar joint one. However, when agents have competing interests—such as in zero-sum games—this centralization can create conflicts, as a shared reward signal may not adequately reflect individual incentives. Additionally, as the number of agents grows, the action space expands exponentially, making it increasingly challenging to manage and optimize the joint action.

Independent learning, in contrast, allows each agent to develop its peculiar policy based solely on its observations, without a centralized controller. This approach addresses some of central learning's limitations, particularly by eliminating the need to aggregate rewards or the dimensionality problem of the action space. However, because each agent perceives the actions of others as part of the environment's dynamics, it is forced to learn under high non stationarity as the environment is continually changing as agents update their policies. This can lead to instability in learning and inability to convergence to a solution. Within independent learning, two approaches are possible when comes to learning the policy: self-play and mixed play.

In **self-play**, all agents operate under the same learning algorithm. A specific form of self-play, known as policy self-play, takes this further by having agents learn to play against versions of their own policy. This approach is particularly effective in game environments, where agents benefit from training

against opponents with the same policy, identifying and overcoming weaknesses of their own strategy. Moreover, the fact that multiple agents concur to the optimize the same policy enables a faster training.

Mixed play, in contrast, involves agents using different learning algorithms, creating a broader variety of strategies in the environment. This diversity encourages each agent to become more robust, as it must adapt to a range of policies that may vary in complexity and strategy at the expenses of an augmented architectural complexity.

3.2 MultiAgent Deep RL

Deep learning serves as a powerful function approximator, helping address the scalability challenge of MARL algorithms. Additionally, by sharing networks and experience data, agents can significantly improve sample efficiency, reducing the amount of experience needed to learn effectively.

MARL approaches can be categorized based on training and execution phases, which determine how agents interact and learn from each other. The training phase considers the observation space available to agents during learning; this space might be centralized or shared among agents, allowing them to leverage broader information, such as joint state-action pairs, which can enhance cooperative learning. In the execution phase, the focus shifts to the decision making process and in particular what information the agent relies on to choose the action to be taken. This distinction between training and execution phases reflects the degree of coordination or autonomy expected. The three main categories for MARL algorithms according to their training and execution modes are:

- Centralized training and execution (CTE)
- Decentralized training and execution (DTE)
- Centralized training and decentralized execution (CTDE)

In **Centralized training and execution** both the learning process and the resulting policies are influenced by the actions of other agents. They often share a range of information with each other that is considered common knowledge, including local observations, learned models, value functions, and sometimes even their own policies. The problem becomes finding a super single agent which coordinates actions for all agents in an optimal manner. Central learning is a specific case of CTE and consequently shares the same limitations listed above, especially in scalability.

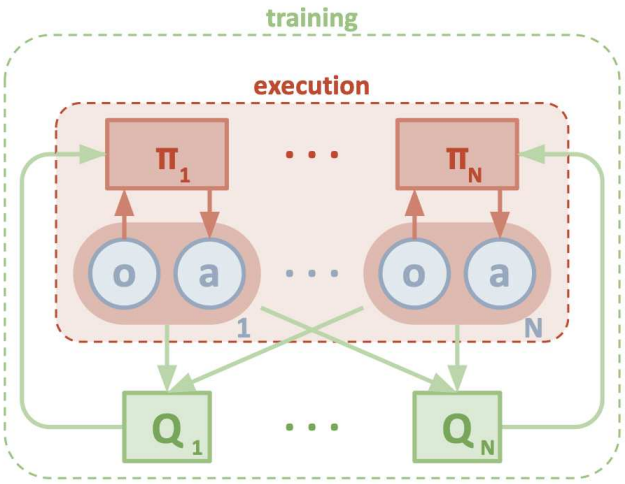


Figure 3.1: Training and Execution diagram for MARL [2]

In **Decentralized training and execution** both the training of agents and the policies they develop occur independently, making this approach suitable when agents cannot communicate directly with one another. In this setup, each agent relies solely on its own observations and rewards, without access to shared information or coordination with others. However, this decentralized approach introduces the major drawback of independent learning: the non-stationarity, that burdens the learning, as agents must continuously adapt to unpredictable changes caused by the evolving strategies of other agents. To address this challenge, decentralized MARL can be enhanced with *agent modeling* techniques, which allow agents to estimate or predict the likely behavior of others within the environment. By incorporating agent models, each agent can attempt to infer the intentions and future actions of its peers. This predictive capability mitigates some of the instability introduced by non-stationarity.

The most popular and effective paradigm in MARL is **Centralized training with Decentralized execution**. It addresses the limitations of fully centralized or fully decentralized approaches by allowing agents to share some common knowledge during training, which benefits in terms of speed and stability. Maintaining independence during action selection in the execution phase on the other hand makes the approach scalable and suitable for environments where communication may not be feasible during execution. This paradigm is particularly effective within the *actor-critic* framework. In CTDE, the critic is centralized, meaning it has access to the combined observations and eventually actions of all agents, allowing it to provide accurate value or q-value estimates by leveraging shared knowledge. Meanwhile, each agent has its own actor and

individual policy to follow, enabling decentralized execution and keeping the dimensionality of the action space manageable.

Generally, the problems that fall into the CTDE can be modeled through the decentralized partially observable Markov decision processes (DEC-POMDP). It is defined by a tuple $\langle \mathcal{S}, \mathbf{A}, \Omega, \mathcal{P}, \mathcal{R}, \mathcal{O}, n, \gamma \rangle$, where

- $\mathcal{S}$ is the state space
- $\mathbf{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the set of joint actions
- $\Omega = \Omega_1 \times \cdots \times \Omega_n$ is the set of joint information.
- $\mathcal{O}$ is the observation probability function

3.3 Centralized Policy Gradient Methods

The policy gradient theorem seen in section 2.3 can be extended to the multiagent case. In [11], Lowe et al. provided a formulation for the gradient of the objective function for agent i in the multi agent case in the context of CTDE, namely given N the number of agents each parametrized by θ_i and characterized by policy π_i, then

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s \sim p_\mu, a_i \sim \pi_i} \left[\nabla_{\theta_i} \log \pi_i(a_i|o_i) Q^{\pi_i}(\mathbf{x}, a_1, \ldots, a_N) \right]. \tag{3.1}$$

$Q_i^\pi(\mathbf{x}, a_1, ..., a_n)$ is the centralized q-value function for agent i where $\mathbf{x}$ contains the common knowledge available such as shared observations. The fact that Q_i^π is learned separately allows each agent to have its own reward structure obviating the central learning problem related to the scalar reward. To obtain Q_i^π the centralized critic approach can be leveraged: to approximate either the value or q-value of the policy of agent i it makes use of the common knowledge (shared observations and also actions of other agents in the case of q).

Among the algorithms that leverage the centralized critic strategy it is worth mentioning the MADDPG proposed by Lowe et al. [11], the MATPRO ([12]) and HATPRO, the latter proposed by Kuba and co. in [13] to adapt the TRPO to multiagent setting, COMA [ref], and the MAPPO.

Relevant to this thesis is the implementation of MAPPO proposed by [5], so here are reported the details of the algorithm.

3.3.1 MAPPO

The MAPPO is proposed as an extension of PPO to the multi-agent setting: a policy π_θ and a value function $V_\phi(s)$ are learned simultaneously by two neural

networks. The assumption of full cooperation is made, meaning that rewards are shared among all agents. As anticipated under the CTDE paradigm, the value function is utilized solely during the training phase but with respect to global states provided either from the environment or concatenating all the local observations, the net designed by Yu et al. has access to both the centralized state (provided by the environment) and local observations. To prevent the overlapping of information, repeated features are removed from the vector input $\mathbf{s}_i$.

The objective for the centralized critic is

$$\begin{aligned} L(\phi) = \frac{1}{Bn} \sum_{i=1}^{B} \sum_{k=1}^{n} \max \Bigg[& \left(V_\phi(s_i^{(k)}) - \hat{R}_i \right)^2, \\ & \left(\text{clip} \left(V_\phi(s_i^{(k)}), V_\phi^{\text{old}}(s_i^{(k)}) - \epsilon, V_\phi^{\text{old}}(s_i^{(k)}) + \epsilon \right) - \hat{R}_i \right)^2 \Bigg] \end{aligned} \tag{3.2}$$

where B refers to the batch size and n to the number of agents, while $\hat{R}_i$ is the discounted reward to go and s_i the shared observations. Yu et al. reinforced the learning stability by standardizing the targets of the value function, observing a reduction also of the training variance.

The actor network, characterized by θ parameters on the other hand takes the agent observations $\mathbf{o}^t$ and outputs a distribution over actions in discrete action spaces from which an action is sampled in continuous action spaces.
It aims at minimizing the following loss function

$$\begin{aligned} L(\theta) &= L^{PPO}(\theta) + \sigma \frac{1}{Bn} \sum_{i=1}^{B} \sum_{k=1}^{n} S \left[\pi_\theta(o_i^{(k)}) \right] \\ L^{PPO}(\theta) &= \frac{1}{Bn} \left[\sum_{i=1}^{B} \sum_{k=1}^{n} \min \left(r_{\theta,i}^{(k)} A_i^{(k)}, \text{clip} \left(r_{\theta,i}^{(k)}, 1-\epsilon, 1+\epsilon \right) A_i^{(k)} \right] \right] \end{aligned} \tag{3.3}$$

where S is the policy entropy, and σ is the entropy coefficient hyperparameter.

Chapter 4

Deep Learning

In reinforcement learning, exact representation methods face significant limitations in solving large-scale problems, as the state space may be vast or continuous, often requiring excessive memory resources to store values for each state. Moreover relying solely on visited states for updates leads to poor generalization and inefficiency as well as greatly increasing the training time. To address these challenges, function approximation techniques are used to enable the agent to generalize from previously visited states to novel, similar ones.

The approximating function is parametrized in θ, whose dimensionality is greatly smaller than the state space, and it is used to represent bot the value and q-value functions but can be extended to the policy as well.

Neural networks are particularly effective function approximators for RL because of their ability to model complex, high-dimensional relationships in data and largely applied in almost every model.

4.1 FeedForward Neural Networks

Deep feedforward networks, often referred to as feedforward neural networks are fundamental models in deep learning. The primary objective of a feedforward network is to approximate a target function, f^*. For instance, in a classification task, $y = f^*(x)$ maps an input x to a category y. A feedforward network constructs a mapping $y = f(x, \theta)$ and learns the optimal parameter values θ, that produce the most accurate approximation on this target function.

The function f in FFNNs is constructed as a composition of multiple simpler functions, reflecting a multi-layered architecture that enables increasing levels of abstraction. Each neuron within this framework computes a weighted

[0]The theory concepts in this chapters are taken from [14].

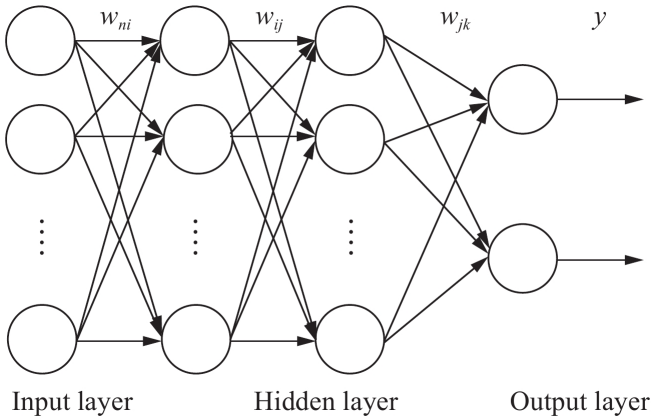


Figure 4.1: Feed Forward Neural Network

sum of its inputs x_i:

$$y = f(\sum_i \theta_i x_i + b) \tag{4.1}$$

which is then transformed through a nonlinear activation function. A single layer of an FFNN stacks multiple neurons. In [15], it has been established that a single-layer FFNN can approximate any continuous function with arbitrary accuracy, provided that an appropriate activation function is utilized. A single layer comes at the cost of a number of hidden neurons exponential in the dimension of the input space, hence multiple layers are stacked together with fewer neurons attaining a hierarchical feature representation. The last layer is the output function and is chosen according to the problem alongside with the loss function.

The training process of a feedforward neural network involves a forward pass, during which the input is propagated through the network to produce an output. This output is then used in the loss function, which quantifies the difference between the predicted and actual values. To optimize the network, the gradients of the loss with respect to each weight and bias are computed using the chain rule of derivatives. These gradients are backpropagated from the last layer to the first, allowing for the systematic updating of weights throughout the network.

Here is reported the ReLU activation function (fig. 4.2) that is the one that will be used in the project:

$$ReLu(z) = \max(0, z) \tag{4.2}$$

It is the basis of modern NN training as it has the simplicity of a linear activation function but is not linear and it encourages sparsity in the outputs

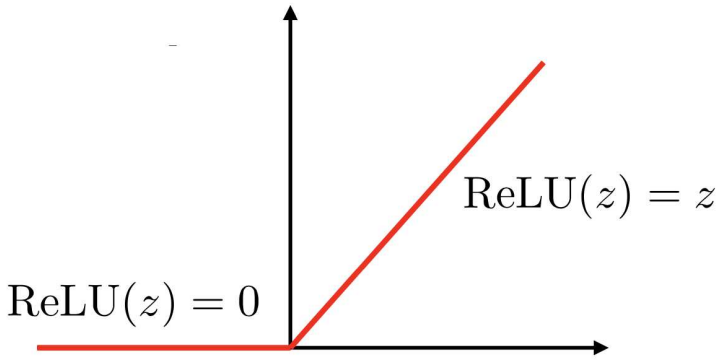


Figure 4.2: ReLU activation function

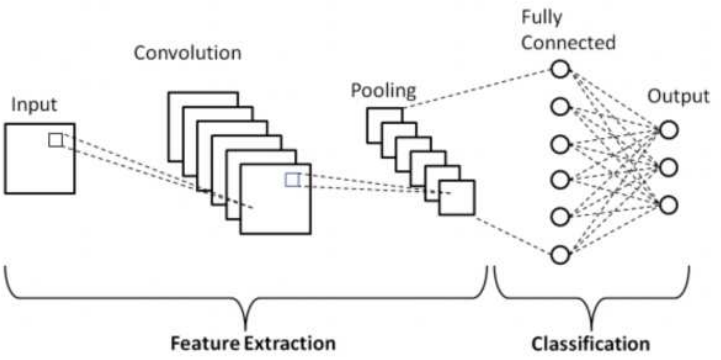


Figure 4.3: Convolutional Neural Network

4.2 Convolutional Neural Networks

A great limitation of FFNNs is that they lose structural information in the input, particularly evident in tasks involving images. To address this issue, in CNNs the matrix multiplication is replaced with convolution operations. Considering as input a tensor for instance representing an image, a filter is slid over it and the convolution is computed. Additionally, the activation function is still applied after the convolution operation to introduce nonlinearity. This approach preserves the spatial relationships within the data, which are crucial when dealing with images. The learnable weights of the model are contained within the filter.

Among the strengths of CNNs there is the local connectivity, that allows the net to learn patterns hierarchically, starting with simple, low-level features in the initial layers and progressively building up to more complex structures in deeper layers. Moreover, the fact that each single filter that is is applied across different regions of the input maintains the same weights helps significantly

reduce memory requirements and prevents overfitting by limiting the number of parameters the model must learn. Additionally, translation equivariance ensures that the network's response to an object in the input remains consistent even when the object is shifted, allowing CNNs to detect features reliably regardless of their position in the input.

The typical architecture of a CNN net is to alternate convolutional layers with pooling layers that serve to reduce the dimensionality of the tensors and contribute to the hierarchical learning. The last layer is typically a feedforward layer as it is more convenient in the output extraction based on the features learned. The process of learning - feedforward pass and backpropagation of the gradient of the loss function is analogous to the one described for the feedforward net.

Chapter 5

Continual Learning

So far, agents in reinforcement learning have primarily been designed to tackle a single, fixed task or a set of multiple but predefined tasks, each with a specific data distribution and objective carried out altogether. In contrast, the human brain learns from a continuous stream of experiences and encounters new tasks on a regular basis. This ongoing exposure to novel information doesn't overwrite past knowledge; rather, the human brain efficiently integrates new skills and concepts while preserving previously acquired ones. Emulating this ability with artificial agents poses a significant challenge in machine learning, as it would allow to develop systems that can learn adaptively and maintain permanently long-term knowledge while incorporating new skills seamlessly without the need to start over the learning process. For this reason continual learning is also referred to as lifelong learning or incremental learning.

Training a model in machine learning is fundamentally an optimization problem, where the goal is to adjust the model's parameters to achieve the best possible performance on a specific task. When learning the new task, the parameters optimized for the first one are simply used as initialization points for the new training process. However, as optimization proceeds on this second task, the resulting optimal parameters typically do not align with those found previously, degrading the model's performance on earlier tasks, a phenomenon known as catastrophic forgetting.

Addressing catastrophic forgetting inherently involves tackling the plasticity-stability dilemma [16]: the challenge of allowing a model to remain adaptable (plasticity) to new tasks while retaining knowledge of previously learned ones (stability). The final goal is to obtain a model that can generalize effectively across tasks to handle the distribution shifts that often occur as new tasks are introduced. However, re-training on an ever-expanding dataset that includes data from all past tasks is often infeasible due dimensionality and storage issues

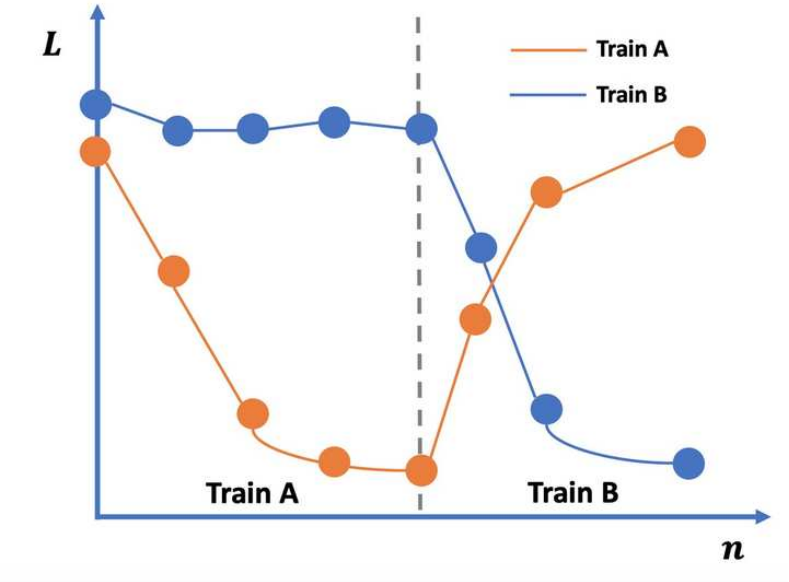


Figure 5.1: Catastrophic Forgetting for two tasks

and privacy concerns.

The proposed approaches in literature explored up to now can be grouped together into five main categories that are enlisted in fig. 5.2 with their sub-categories. To the purposes of this thesis will be reported the following:

- Replay-based approaches
- Regularization-based approaches

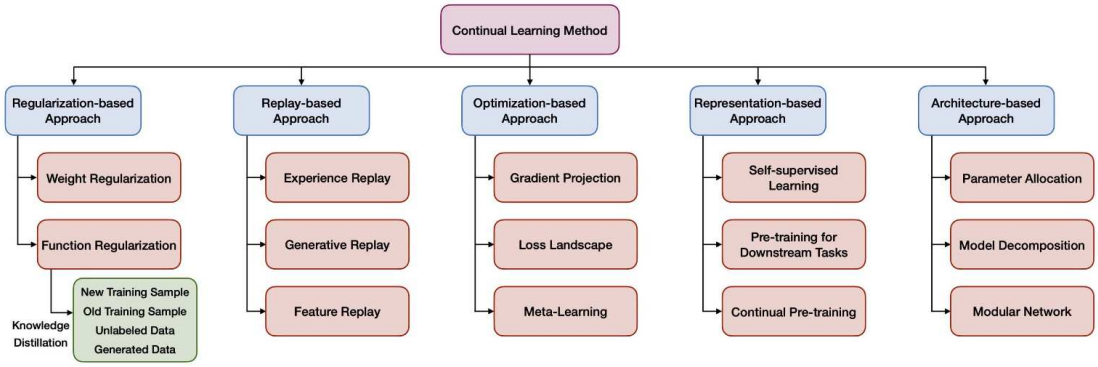


Figure 5.2: State of Art on Continual Learning approaches [3]

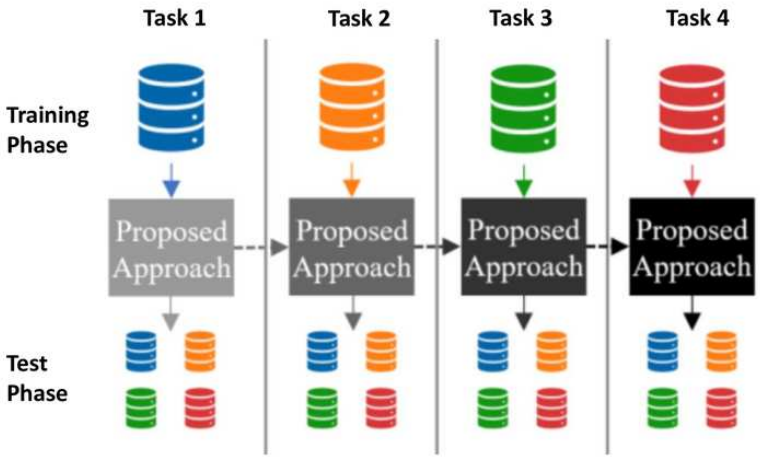


Figure 5.3: Continual Learning process

5.1 Replay-Based Approaches

Replay methods represent the most intuitive and simple idea to prevent the model to forget knowledge previously acquired: a subset of data from previously encountered tasks are stored in a buffer. Such samples, randomly selected, are provided to the model along with the new data with the aim at helping the model consolidate previous learned knowledge, hopefully preventing it from being swept away by the new task.

Replay methods can be classified into three main categories based on the type of content they replay:

- Experience Replay
- Generative Replay
- Feature Replay

Experience replay consist in storing a few training samples of past tasks in a memory buffer. The significant challenge lies in determining with samples to keep as the size must remain small to be resource-efficient in the accumulation of tasks. The sample selection can be done randomly as in [17], to create a buffer that contains the same number of samples for each task encountered, while more advanced techniques make use of optimization to promote sample diversity, like in [18]. The way the collected samples are used for training is equally important. The simplest strategy is to adopt the samples as constraints that can be task specific [19] or global [20] for the optimization of the new task. Then again there are more complicated strategies such as making

adaptive constraints on the forgetting rate of samples for a targeted training [21], or to use knowledge distillation [22].

In **Feature replay** methods on the other hand the samples stored in the buffer are feature level instead of data level and this allows to overcome privacy issues that arise in keeping track of data directly. The main problem with these methods consists in the representation shift, the counterpart for catastrophic forgetting, namely that the internal representations or features learned by a neural network change as it is trained and the stored older ones become incompatible with the model's current interpretation. [3]

Generative Replay, also known as pseudo-rehearsal involves producing samples from the generative models of the old tasks rather than from a buffer and therefore this method is mostly used in the development of continual learning for said models.

5.1.1 Regularization-Based Approaches

Regularization approaches aim at preventing catastrophic forgetting by introducing a soft constraint on the model update in the form of a regularization term in the loss function. In this way the two main drawbacks of the rehearsal strategy are alleviated: the fact that samples are not required benefits both the memory and privacy concerns.

In [23] this approach is divided into two branches, data and prior focus methods.

Learning without Forgetting [24] belongs to the **data-focus** category as the regularization happens on the loss between outputs. In particular, when the network is training for the new task it is enforced to remain similar to the previous model by leveraging the distillation loss [25] evaluated on the outputs of the old net and on the one in training. In classification problems, the loss function applied is the cross-entropy

$$L^{CE} = -\sum_{i=1}^{N} p_i \log(q_i) \tag{5.1}$$

where p_i are the correct outputs (provided by the old model), while q_i are the probabilities predicted by the new net.

Among **prior-focus** method is worth mentioning the EWC (Elastic Weight Consolidation) method [4], proposed by Kirkpatrick et al. in 2017 for supervised learning an reinforcement learning tasks. When considering two tasks A and B generally the assumption that the parameter space for A and B have some sort of overlapping holds and that the desired solution to fit both the

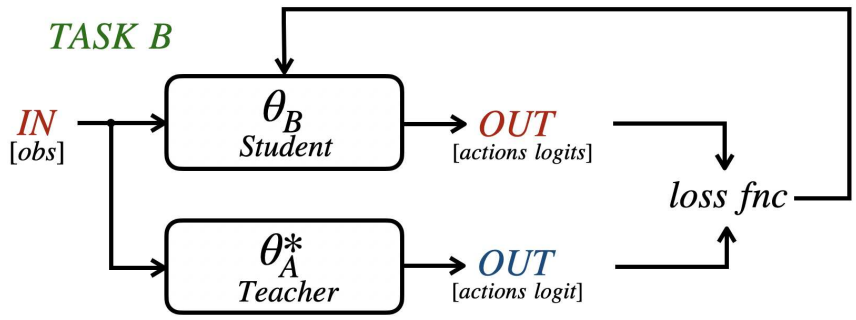


Figure 5.4: Knowledge Distillation [3]

tasks should be in the intersection. But how to correctly move towards such intersection? The idea is that each weight has a difference importance for each task, and hence a suitable solution should penalize more the changes that affect important weights for A leaving freedom to the less influent ones. To do so the metric used by EWC keeps into account how much the weight changes affect the loss function for A and is derived from the probabilistic interpretation of the loss function, in particular from the posterior distribution $p(\theta|\mathcal{D}) = p(\theta|\mathcal{D}_A, \mathcal{D}_B)$ reason for which this methods belongs to the prior-focus category. Applying Bayes rule and the *log*, the final aim becomes finding the θ parameters that maximize the

$$log\ p(\theta|\mathcal{D}) = log\ p(\theta|\mathcal{D}_A) + log\ p(\mathcal{D}_B|\theta) - log\ p(\mathcal{D}_B) \tag{5.2}$$

From the 5.2,the EWC loss function becomes:

$$L^{EWC}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i}) \tag{5.3}$$

considering that the first term of 5.2 must be approximated with the Fisher information matrix and is treated as a regularization term, while the log-likelihood of B (negative) is the standard loss function to be minimized to optimize B. The main drawback of this approach is that the complexity scales linearly with the number of tasks.

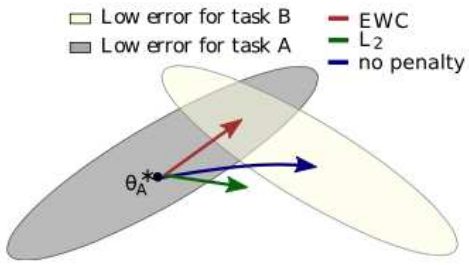


Figure 5.5: How the weights should be shifted from task A to task B to retain past knowledge [4]

Chapter 6

Problem Formulation

Continual learning is an emerging and promising field of research, as it presents a range of unresolved challenges that are crucial for the development of more adaptive and scalable artificial intelligence systems. Specifically, in the context of multi-agent reinforcement learning, it offers the potential to enable agents to continually adapt to environments and entities within it that dynamically change.

The problem addressed in this thesis is inspired by the paper [26], which focuses on designing a training paradigm for fully cooperative multi-agent games with the ultimate goal is for these agents to be able to effectively coordinate with new agents they have never encountered during training. In their setup, the teams differ from each other in terms of architecture and initialization seed, leading to the emergence of diverse strategies and policies among the agents. To tackle this challenge, they employ off-policy MARL algorithms, namely IQL and VDN. After applying continual learning techniques, the agents are tested for their ability to coordinate with unseen agents in few-shot and zero-shot scenarios, evaluating their adaptability and generalization in cooperative settings.

A similar challenge is tackled by Yuan et al. in [27]. However, their focus lies in how the groups of teammates are constructed. These groups are generated using a genetic procedure that ensures each new set of teammates has policies distinct from those encountered previously, thereby ensuring comprehensive coverage of the entire policy space. This approach aims at guaranteeing the ability for the agents to adapt to each possible team configuration. Like the approach in Nekoei et al., Yuan et al. also utilize off-policy algorithms such as VDN and QMIX.

Instead of working off-policy, this thesis explores the advantages of on-policy approaches in multi-agent reinforcement learning. On-policy methods,

unlike off-policy ones, allow agents to learn directly from their current experiences, which can lead to more stable and efficient learning. The remarkable results obtained by Yu et al. [5] in the application of on-policy methods for cooperative multi-agent tasks highlight their effectiveness in improving agent performance and coordination.
Building on these insights, the problem tackled in this thesis is to address the following challenge in a fully cooperative multi-agent scenario: how to apply a continual learning approach that enables agents to cooperate with different, unseen agents without losing their ability to collaborate effectively with their own team. This involves developing methods that ensure agents can retain their learned coordination skills while also adapting to new team configurations.

6.1 Framework

The experiments are carried out in two different frameworks: MPE environment and Starcraft II.

6.1.1 MPE

The Multi Particle Environment setting is based on the cooperative partially observable Markov game framework developed by OpenAI. With respect to the scenario illustrated in [28], the assumption of identical action and observation spaces across all agents is removed and agents do not share the same policy.
The environment operates within a finite discrete horizon, with each episode lasting 25 steps. It consists of N agents and L landmarks situated in a two-dimensional world with continuous space

The agents' objective is to collaboratively perform a task that maximizes a shared return, which is equally distributed among all agents. Each agent and landmark has a position and intrinsic properties, such as color and shape, which collectively define the state. The actions available to the agents include moving through the space, remaining stationary, and to produce a communicating signal with each other. The communication variables are abstract, with their meanings being assigned by the agents during the training process. Additionally, each agent has unique, individual goals that are not shared with others, thus necessitating communication. Each agent also maintains a private memory bank that they can use at their discretion. The observation space of each agent consists of the physical state of all other individuals, incoming communications, along with its own memory and goals.

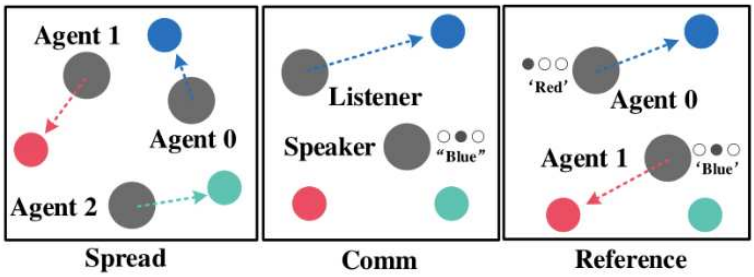


Figure 6.1: Scenarios for MPE environment, [5]

In particular, among the scenarios of the MPE world, the ones considered are:

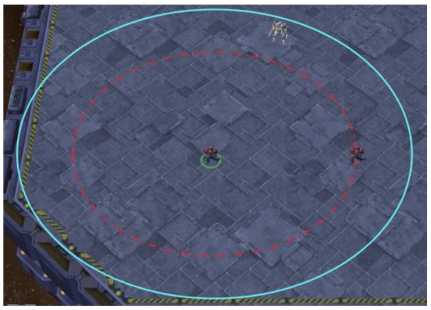
- Cooperative communication (`comm`) where one agent acts as the "speaker", remaining stationary and whose only possible action is which communication signal emit according to the goal. The other agent functions as the "listener" and hence cannot communicate but must navigate to the correct landmark based on the speaker's guidance.
- Simple reference (`reference`): same as the `comm` scenario but both agents are simultaneous speakers and listeners, namely each agent has a specific target landmark, but the target of one agent is only known to the other agent. The agents share a collective reward, meaning their success depends on both agents reaching their respective target landmarks. To achieve this, the agents must learn to communicate the goal of the other agent while also navigating to their own landmark.
- Coverage (`spread`): in this scenario there are N agents and N landmarks, the collective rewarded is based on how close any agent is to each landmark, promoting coverage of all landmarks. However, agents are penalized if they collide with one another.

and are graphically represented into

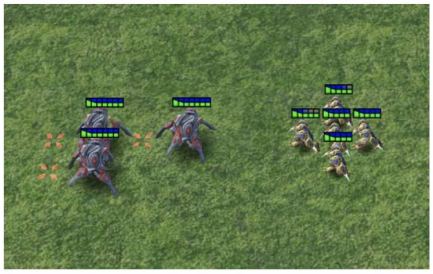
6.1.2 SMAC

The second framework considered is the Starcraft II micromanagement challenge.

StarCraft is a complex real-time strategy (RTS) game that demands a combination of micromanagement and macromanagement skills. Micromanagement involves the precise control of individual units to maximize their efficiency in combat and resource collection, while macromanagement requires



(a) Sight and shooting range for the agent



(b) An example of a map of the SMAC challenge

Figure 6.2: SMAC environment [6]

high-level planning, such as managing resources, building structures, and planning long-term strategic goals. StarCraft presents unique challenges for multi-agent reinforcement learning for its characteristics: the primary challenge is that it is an imperfect information game; players face a 'fog of war' which oblige active exploration of the map as unvisited regions are obscured. Additionally, StarCraft's gameplay emphasizes the difficulty of temporal credit assignment, since early decisions may yield consequences only apparent later in the game. This delayed reward structure tests an agent's ability to plan for both short-term and long-term outcomes. More details of the game dynamics can be found in [29].

Samvelyan et al. propose a series of mini-challenges set in the Starcraft environment mentioned above.

The Starcraft MultiAgent Challenge (SMAC) [6] is a simplified set of tasks that has been developed to focus on micromanagement skills within controlled combat scenarios, removing the macromanaging complexity of the integral game. In these scenarios, agents assembled in teams operate within restricted minimaps and are tasked with defeating a similarly structured enemy team. The primary objective is hence to maximize damage inflicted on enemies while minimizing health loss across allied units. To achieve this, agents must develop several essential skills like focus fire, where agents coordinate to target and eliminate specific enemy units efficiently. Furthermore, if the team includes different unit types, agents must leverage each unit's unique strengths, optimizing the team's overall effectiveness. Additionally, agents are expected to coordinate attacks from multiple angles, exploit advantageous terrain, and execute kiting maneuvers - strategically moving to maintain a safe distance while provoking enemies to follow.

The enemy units are controlled by the game's built-in AI.

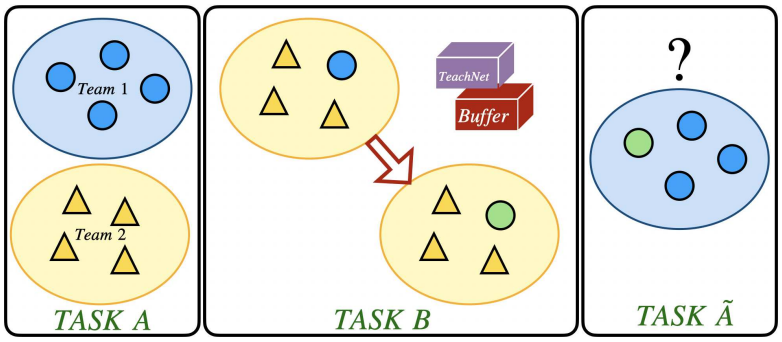


Figure 6.3: Problem formulation diagram: The first task requires the teams to learn to play from scratch, with each team consequently developing its own policy and an internal communication strategy. Subsequently, an agent from one team is placed in the other team, allowing it to adapt to the behavior of its new teammates. Once the learning process is complete, the agent is returned to its original team to evaluate whether it has forgotten its initial knowledge

From the RL point of view, the reward consists in the damage dealt plus a bonus for winning the game.

Each agent can observe within a restricted field of view, in which it can assess information of nearby units, including both allies and enemies. This makes the environment partially observable. The information each agent receives is represented as a feature vector that includes key details about each unit within its sight range. Specifically, this vector contains data on each unit's type, distance from the agent, current health and shield levels, the most recent actions taken by allied units, and relevant terrain features within the field of view. This limited perspective requires agents to make strategic decisions based on incomplete information, as they lack a full understanding of the entire map's layout and enemy positions outside their immediate surroundings. During centralized training however a global state is available containing information about all the units on the map. The action space is discrete and includes options for movement in specific directions, the choice to attack and which enemy to target if it is within shooting range or the option to do nothing.

6.2 Proposed Approach

The challenge tackled in this thesis consists in applying continual learning in the framework of the fully cooperative multiagent environments MPE and SMAC in order to train an agent making it able to change team and without damaging the new team performance while retaining knowledge of the previous team it was in. The problem can be broken down to the following steps as shown in 6.3:

hyperparameters	common	MPE	SMAC
GAE λ	0.95		
γ	0.99		
FC layer dim	64		
num of FC	2		
activation	ReLU		
lr		$7e-4$	$5e-4$
PPO epochs		15	$5, 15$
clip ϵ			0.2

Table 6.1: Hyperparameters of MPE and SMAC

1. Train two teams to obtain satisfactory results in their task. When considering an agent within its team it is referred as Task A.

2. Replace one agent with a player from another team and train the new formation. Such agent will be called *Active Agent* as it is the only one whose parameters are updated, while the rest of the teammates are frozen in order to prevent them from modifying their policy to adapt. When the Active Agent is training with the unfamiliar team, it is referred as Task B. Task B is carried out in three paradigms:

   a) Naive Training, namely no expedients are taken to prevent the agent to forget and serves as a benchmark to compare the performance of the continual learning with.

   b) Buffer Replay: alongside with learning with the unfamiliar team, the agent is provided with samples taken from task A, in order to keep track of his initial team

   c) LwF: while training the new net for the agent, the old frozen net of the agent during task A serves as teacher to enforce some kind of memory of its previous behavior.

3. Put back the agent in its initial team (Task $\tilde{A}$) to assess the loss or not of knowledge

Hereby is the detailed implementation of the R-MAPPO algorithm proposed by Yu et al. in [5] used as a starting point to conduct continual learning experiments.

The hyperparameters and the net configurations for the experiments are enlisted in tab. 6.1

The first continual learning approach implemented is the rehearsal mode, following [30] for the choice of saving the output logits - the unnormalized

Algorithm 2 MAPPO

Initialize θ (parameters for policy π) and ϕ (parameters for critic V) using Orthogonal initialization
Set learning rate α
while step $\leq \text{step}_{\max}$ **do**
  Set data buffer $D = \{\}$
  for $i = 1$ to batch_size **do**
    Initialize empty trajectory $\tau = [\,]$
    for $t = 1$ to T **do**
      for each agent a **do**
        $(p_t^{(a)}, h_{t,\pi}^{(a)}) = \pi(o_t^{(a)}, h_{t-1,\pi}^{(a)}; \theta)$
        Sample action $u_t^{(a)} \sim p_t^{(a)}$
        $(v_t^{(a)}, h_{t,V}^{(a)}) = V(s_t^{(a)}, h_{t-1,V}^{(a)}; \phi)$
      end for
      Execute actions u_t, observe r_t, s_{t+1}, and o_{t+1}
      $\tau \mathrel{+}= [s_t, o_t, h_t^{\pi}, h_{t,V}, u_t, r_t, s_{t+1}, o_{t+1}]$
    end for
    Compute advantage estimate $\hat{A}$ on τ using GAE
    Compute reward-to-go $\hat{R}$ on τ and normalize
    Split trajectory τ into chunks of length L
    for each chunk $l = 0, 1, \ldots, T//L$ **do**
      $D = D \cup (\tau[l : l + T], \hat{A}[l : l + L], \hat{R}[l : l + L])$
    end for
  end for
  for each mini-batch $k = 1, \ldots, K$ **do**
    $b \leftarrow$ random mini-batch from D with all agent data
    Adam update θ on $L(\theta)$ with data from b
    Adam update ϕ on $L(\phi)$ with data from b
  end for
end while

probabilities or preferences for selecting each action - to the net instead of the true output. In contrast to the approach proposed that randomly saves samples during the training phase, the chosen method involves sampling from the task local optimum to ensure more flexibility in the buffer creation. Since the training phase in particular for the SMAC environment required at least $5 * 10^5$ steps, it is inefficient to run the process each time the new buffer must be created in order to retrieve only a small fraction of them. Moreover samples from the early stage of training hence associated to poor performances could lead the Active Agent into favoring disadvantageous actions. The size of the buffer was chosen of length 250 (equivalent to 10 episodes of 25 steps) for the MPE scenario and 400 for SMAC (corresponding to a single episode). The samples taken randomly from the buffer are provided to the agent during each PPO epoch and the difference between the old and the new outputs takes part in the loss function as a regularization term.

MAPPO follows an actor-critic structure with CTDE, since the goal is to make the single agent recall the past knowledge, it is meaningful to regularize the decentralized part (the actor) the actor, whose policy does not rely on the shared knowledge. The samples are hence composed of the observations and the available actions, both inputs of the actor net and by the logits correspondent to the actions that the net outputs. In every PPO update, the input sample is given to the net in training and the output sample is compared with the output predicted by the net with a $l1$ loss function, as shown in fig. 6.4

The loss function that is backpropagated in the actor net becomes:

$$L(\theta) = L^{PPO}(\theta) + \lambda L^{Buf}(\theta) \tag{6.1}$$

with λ an hyperparameter that regulate the trade-off between learning the new task and remembering the previous one.

To implement LwF, the opposite approach was taken: instead of using data from the old task, the optimal network previously found for the Active Agent was used as a teacher network to perform knowledge distillation on the network currently in training. In this case, both an $l2$ loss on the logits, similar to the buffer replay case, and a cross-entropy loss (as in [24]) were tested. Cross-entropy loss, typically used in classification tasks, can also be applied in this context since the actions are discrete and therefore encoded similarly to class labels. Once again, the term is used as a soft constraint, weighted by a hyperparameter λ to tune.

The last thing that needs to be specified is how different teams were obtained; in particular by using different seeds in the initialization of the nets as in [26] it is safe to assume that the policy learned will be different enough to

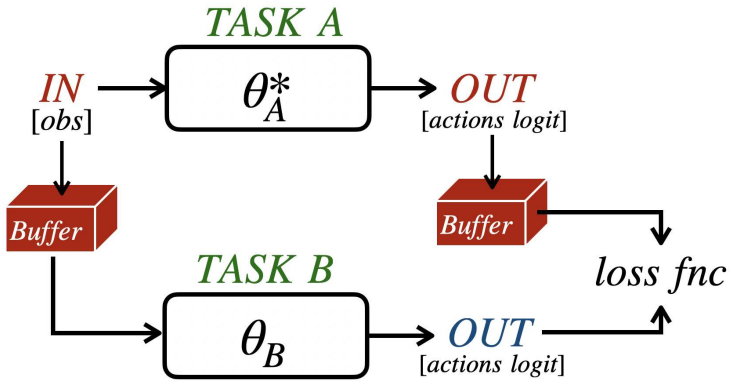


Figure 6.4: Creation and use of the replay buffer

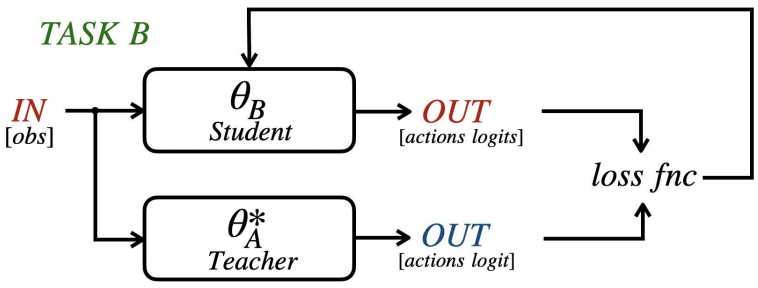


Figure 6.5: Policy distillation from the net of task A

sustain the hypothesis on which the problem is based on - that each team plays in its own way. Conversely, with respect to Nekoei et al. the architecture used is the same for each team and player. Anyway, in the Naive training phase it is possible to obtain a further assessment of this fact, as when one agent is replaced the overall team performances significantly drop.

Furthermore, it is worth mentioning that the number of steps the training (both of task A and B) depends on the environment and on the particular scenario. This holds especially for SMAC as the difficulty of the map and the number of players influences the convergences speed and hence, in order not to waste time and resources, the training length is chosen ad hoc for each map.

The main objective of the experiments is to understand if these simple techniques prevent the Active Agent to incur into catastrophic forgetting and successively to investigate the influence of the architecture and the hyperparameters to obtain an optimal configuration.

Chapter 7

Results and Discussion

To evaluate the performance of continual learning methods in comparison to naive training - where no strategies are used to retain knowledge from previous tasks - the analysis is conducted in two phases. First, the training process for task B is presented, specifically when an already trained agent operates with a new team. Both scenarios, with and without the implementation of CL methods, are considered.

Subsequently the agent's performance is assessed after being reintegrated into its original team. In this test scenario, the neural networks of both the agent and its teammates are frozen, meaning that no further learning occurs during this phase. Therefore, the number of episodes in which this test is carried out is lower than the previous one. This phase is referred to as $\tilde{A}$ which signify that the agent is playing with its original team, but with potential modifications resulting from the experiences gained during task B.

The nominal performance A and B reported in the plots as references represents the optimal values reached by team 1 and team 2 when trained from scratch. Such references are useful in both the training and testing phases: during task B, they serve to determine whether the performance of the second team is impacted by the presence of the foreign agent and to evaluate whether continual learning processes introduces performance degradation or interference. On the other hand, the optimal value achieved by team 1 before one of its players was modified serves as reference for evaluating the forgetting.

7.1 MPE Results

Some useful information regarding the MPE environment are provided: the plots represent the rewards obtained during tasks B and A explained in the previous section with respect to the time steps, that are for convenience grouped

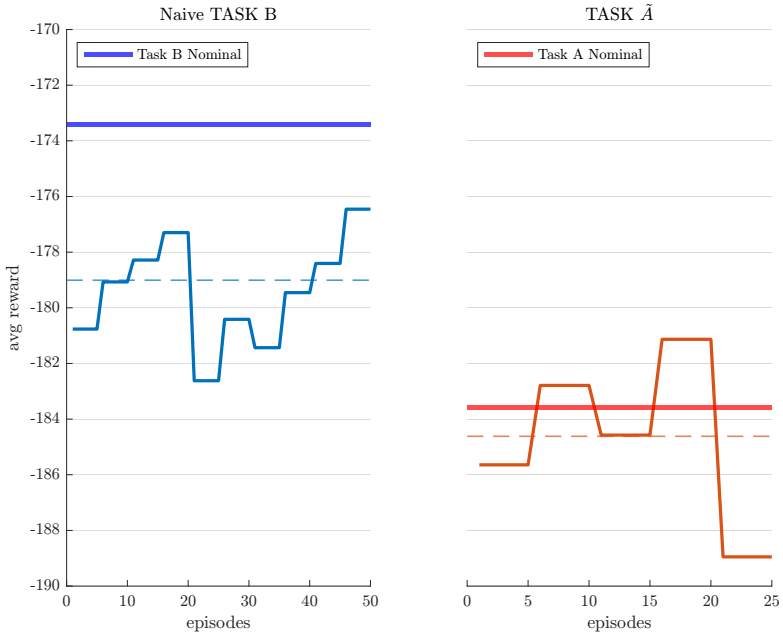


Figure 7.1: **`spread`** *scenario, Naive Training*

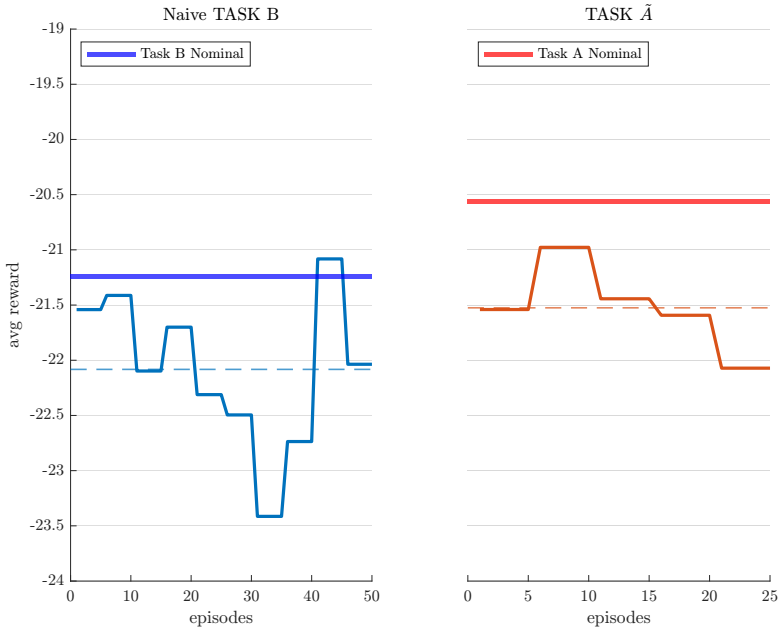


Figure 7.2: **`reference`** *scenario, Naive Training*

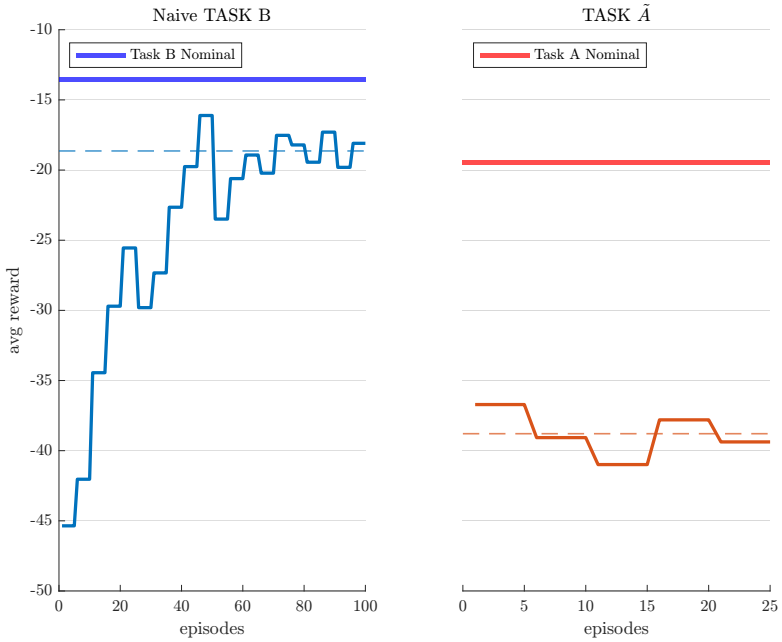


Figure 7.3: `comm` scenario, Naive Training

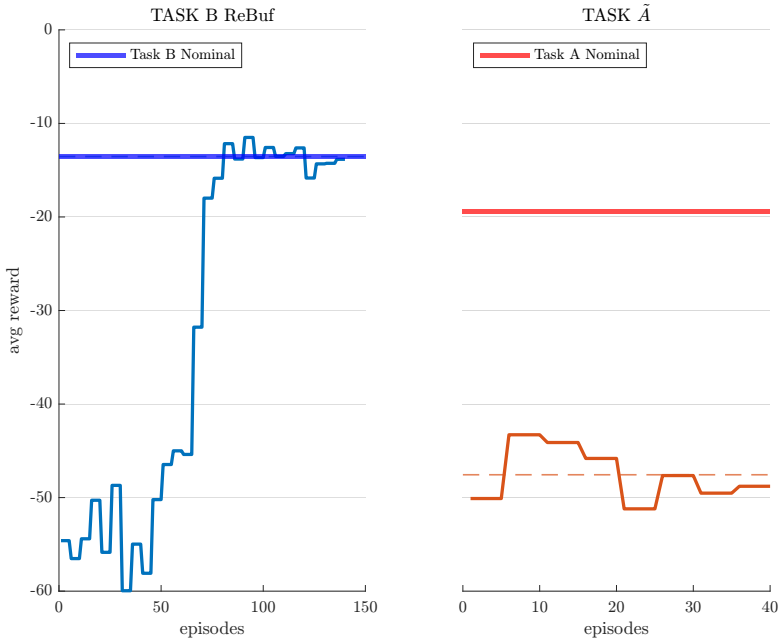


Figure 7.4: `comm` scenario, Training with Buffer Replay and $\lambda = 1 * 10^{-4}$

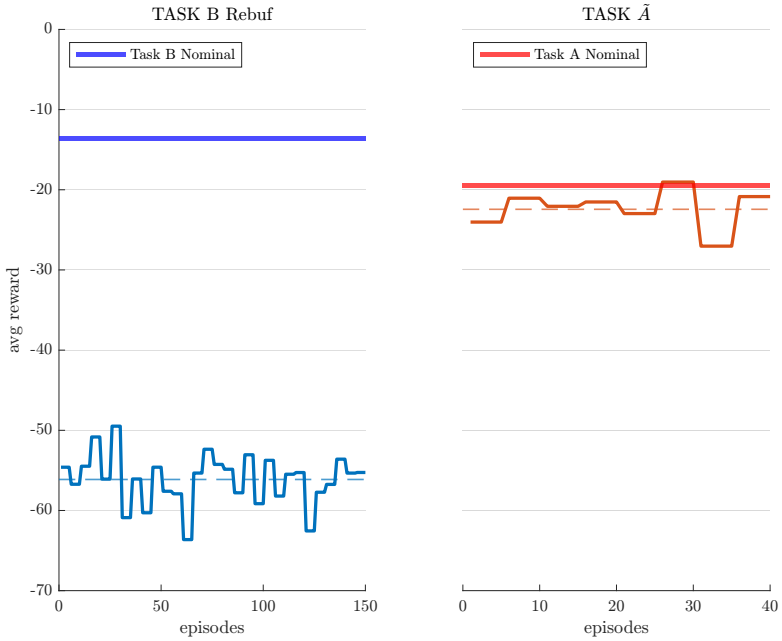


Figure 7.5: `comm` scenario, Training with Buffer Replay and $\lambda = 1 * 10^{-3}$

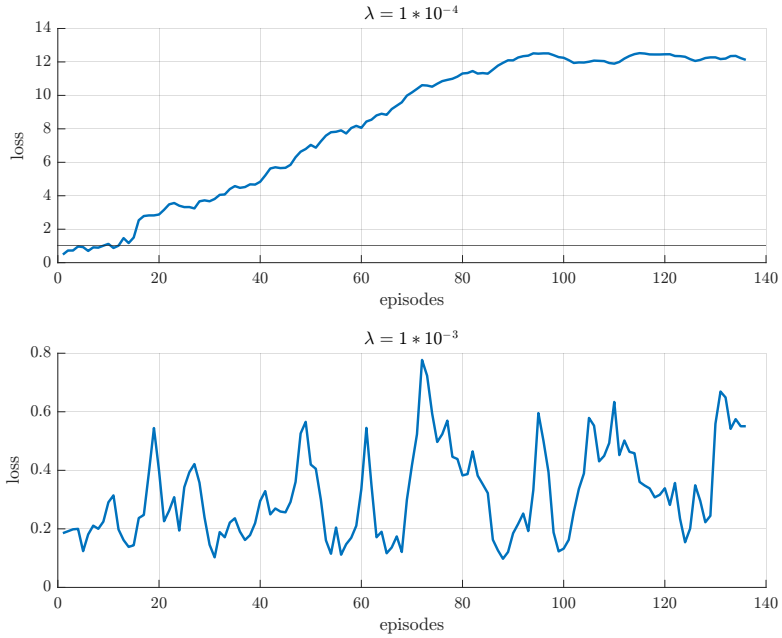


Figure 7.6: `comm` scenario, replay loss l1

into episodes to make the plot easier to read. Each episode is 25 step long; 128 parallel threads are run, the rewards gathered each 5 episodes (125 steps) in each thread are averaged so that the plot does not result noisy and the instantaneous oscillations would not provide any relevance to the understanding of the trend.
Moreover, the 15 epochs of PPO update are performed at the end of each episode, so for as concerns the buffer replay technique that requires a sample to be provided at each PPO epoch, the size chosen initially for the buffer is of 250 samples, in a ratio 1/15 in the case of a 250 episode long training, that was sufficient to ensure the proper functioning of the method.

From the figures 7.1 (`spread`) and 7.2(`reference`) it is possible to state that these two scenarios do not require any continual learning method to improve as there is no forgetting by shuffling the teams. The reason behind this fact may reside in the nature of the task:

- in the `spread` scenario the particles need to reach the marks avoiding collisions but since they possess also the knowledge regarding the position of other agents, therefore the optimal policy they learn is probably universal and independent of the actions of others, hence not requiring cooperation.

- in the `reference` scenario, on the other hand, both the agents are able to speak and move and have access to an observation space wider than the `comm` scenario, where one particle is essentially blind and cannot feedback about what the other is doing.

In the `comm` scenario on the other hand, communication is essential as one agent is able only to speak and has no other information of the whereabouts of the other agent to leverage for the action selection. Both agents must rely completely on the communication to reach the goal, hence the recalls of previous knowledge are not able to compensate forgetting the previous task. In the end the agent will be able to cooperate with either of the teammates, but not simultaneously, and it is possible to appreciate this fact by the figures 7.5 and 7.4. The λ factor is responsible for how much influence past knowledge has on performing the current task, so for a smaller value (i.e. $\lambda = 1 * 10^{-4}$) the agent learns to play with the new companion at the expenses of its original team, while with $\lambda = 1 * 10^{-3}$ the recalls prevent the agent to learn properly the new task.

This fact can also be confirmed by fig. 7.6, which shows the trend of the cost functions when the regularization λ parameters varies.

7.2 SMAC Results

In the SMAC environment, several scenarios among the ones proposed by [5] were tested, posing particular focus on three maps that demonstrated satisfying results in terms of training convergence. The optimal configuration implemented in [5] involves 8 parallel threads and the performing of the PPO update at the end of each episode, which is 400 steps long. Moreover the number of epochs is reduced to 5 in the case of hard maps. Even though the training process requires a larger number of steps with respect to the MPE scenario, the environment is also more complex due to the higher number of agents with broader observation and action spaces. For this reason, the size of the buffer for the SMAC environment is chosen equal to 400. The performances are assessed over the incremental win rate, that is evaluated at the end of each episode through the difference between the battles won in the current and in the previous episode, accumulated from the start.

The maps selected among the ones classified as easy are the `3s_vs_4z` and the `8m`, while for the hard ones the `3s5z`. Figures 7.7, 7.8 and 7.9 show the results for the Naive experiment; note the visible degradation of the team performances when the agent is reinserted after being trained with other companions (task $\tilde{A}$), especially in the `3s5z` and `3s_vs_4z`. In the `3s_vs_4z` map a more critical deterioration of the performances occurs with respect to the other maps as the number of agents is significantly smaller; in the `8m` and in the `3s5z` the other agents compensate for the one that has forgotten how to cooperate.

Hereby are present the results obtained with the continual learning techniques.

7.2.1 Buffer Replay

The rehearsal method provided the best performances on all three scenarios, as shown in fig 7.10, 7.11, 7.12. For the choice of the regularization parameter the best option is to use $\lambda = 1 * 10^{-3}$ as it can be seen in particular in fig. 7.11 and 7.12, which illustrates how performance varies when different values of λ are picked. In both maps the increase in the recall is negligible with respect to the learning of the new task, in particular in the `8m` map. An experiment on the maps 3s5z and 8m to reduce the size of the buffer to the 50%. For the `8m` and `3s_vs_4z` maps the different size did not affect the performance neither of the task B nor of the task $\tilde{A}$. In fig. 7.13 is reported the example for the latter. On the contrary, it is interesting to note that in the case of the `3s5z` map, even though achieving comparable final outcomes in both tasks, using a

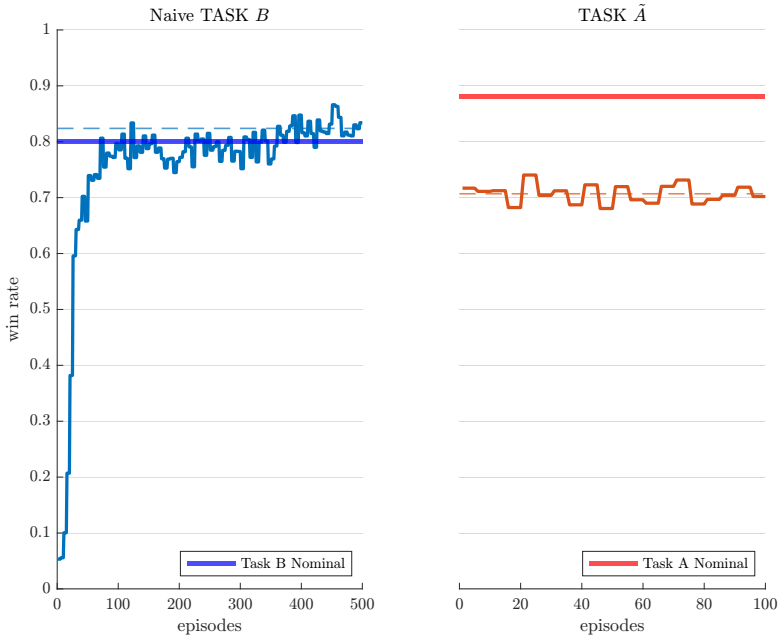


Figure 7.7: map: `8m`, Naive Training

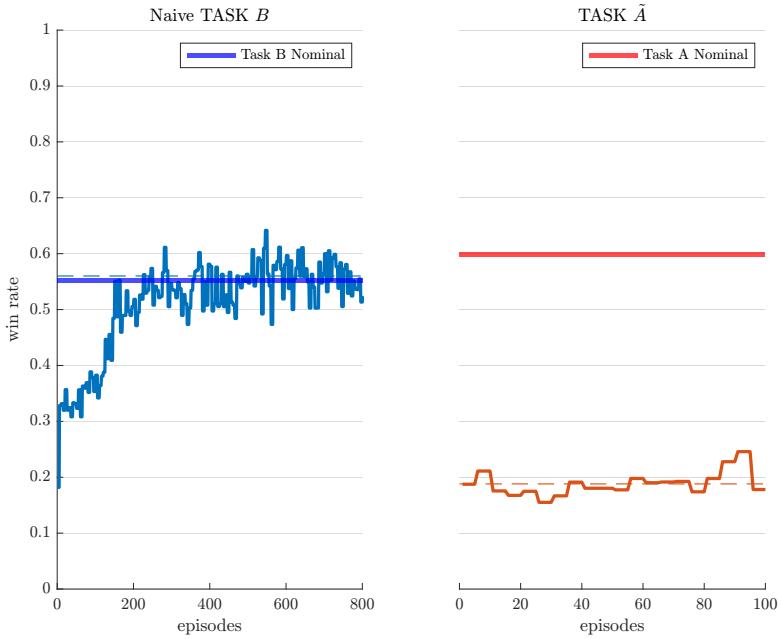


Figure 7.8: map: `3s5z`, Naive Training

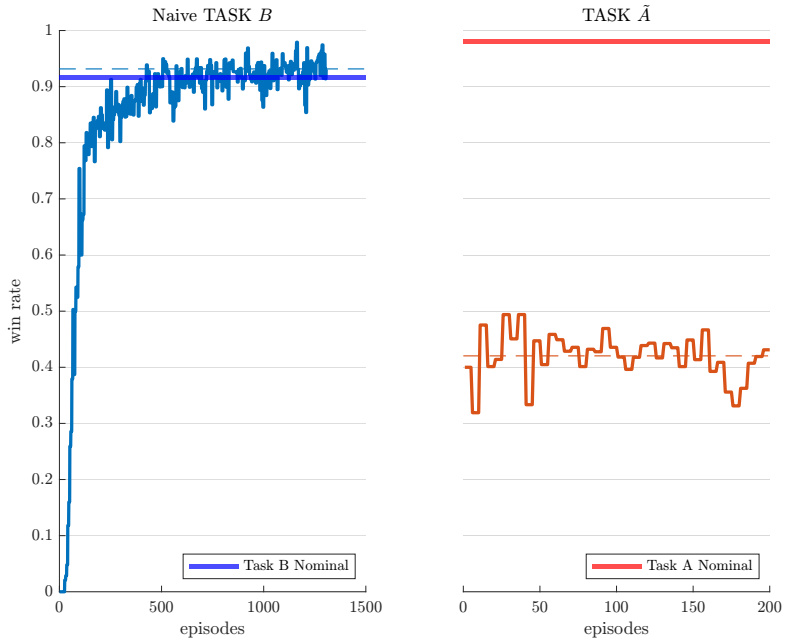


Figure 7.9: map: `3s_vs_4z`, Naive Training

reduced number of samples resulted in an increased convergence speed. This can also be observed in the loss functions, where it is noticeable that with the reduced buffer, the replay-related errors stabilize at the same level in both cases but exhibit a higher peak in the case of the 50% buffer. This phenomenon does not occur in other maps, likely because, being simpler maps, they already have a good convergence speed.

7.2.2 LwF

The implementation of the LwF on the other hand does not yield satisfying results for the continual learning task, in neither of the maps. The $l2$ loss function was preferred to the Cross Entropy because as it can be observed in fig. 7.18, the forgetting of the old task in systematic, fact that happens even with stronger values of λ.

With the LwF method, it is evident that the trade-off between present and past performance becomes more pronounced. As shown in fig. 7.16, similar to the `comm` scenario, it is not possible to achieve good performances on both tasks simultaneously. This can be attributed to the architectural difference: unlike the buffer approach, which encourages the network to maintain similarity to its previous configuration, in the distillation process the agent is asked to replicate the same behavior under observations that might not have been encountered in the previous task. As a result, instead of "training together with its old

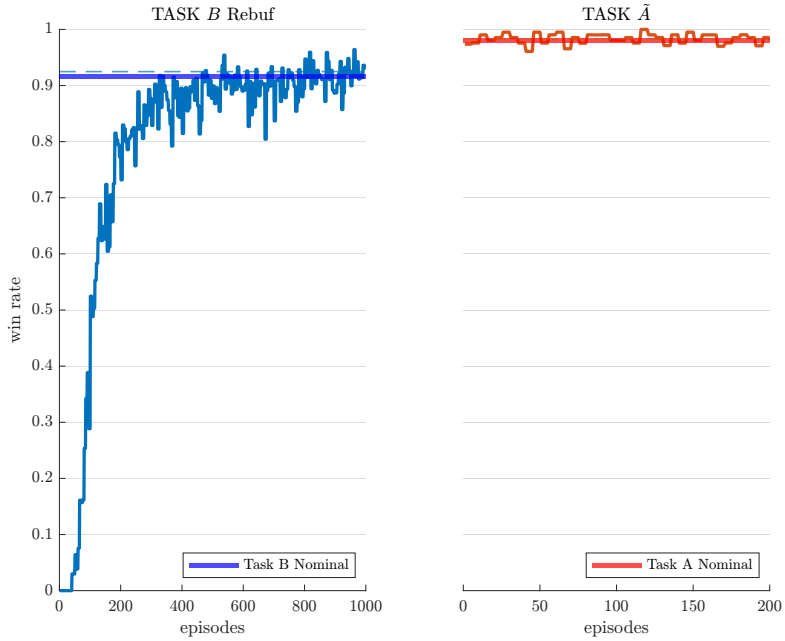


Figure 7.10: map: `3s_vs_4z`*, Buffer with* $l1$ *loss,* $\lambda = 1 * 10^{-3}$

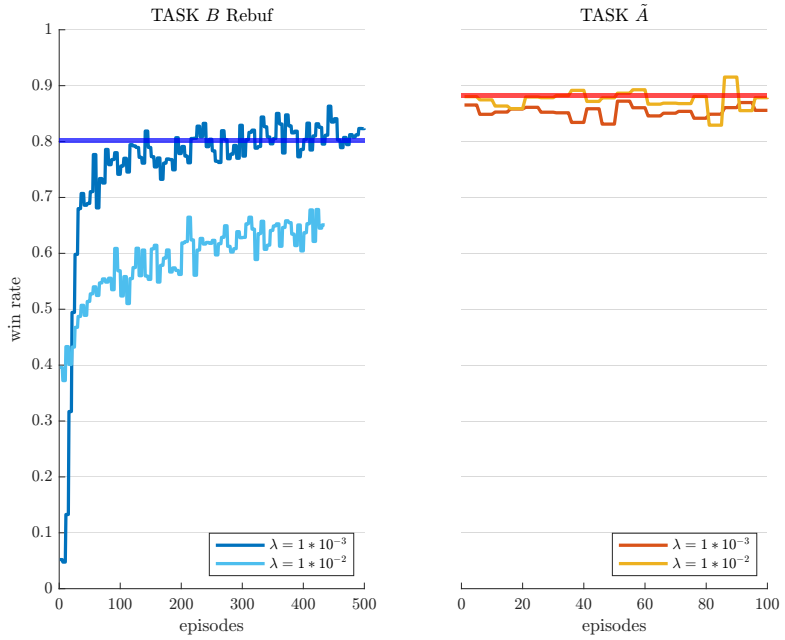


Figure 7.11: map: `8m`*, Buffer with* $l1$ *loss, comparison between different values of* λ

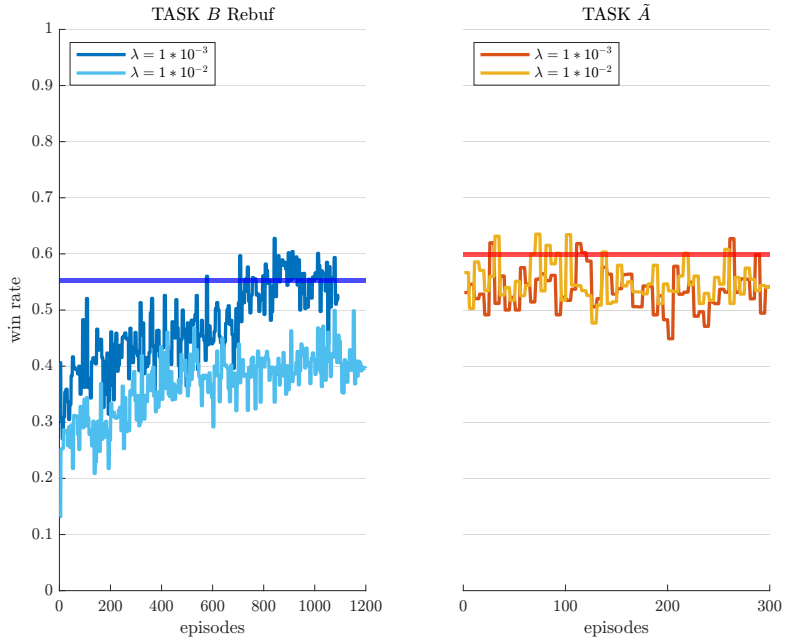


Figure 7.12: map: `3s5z`*, Buffer with* $l1$ *loss, comparison between different values of* λ

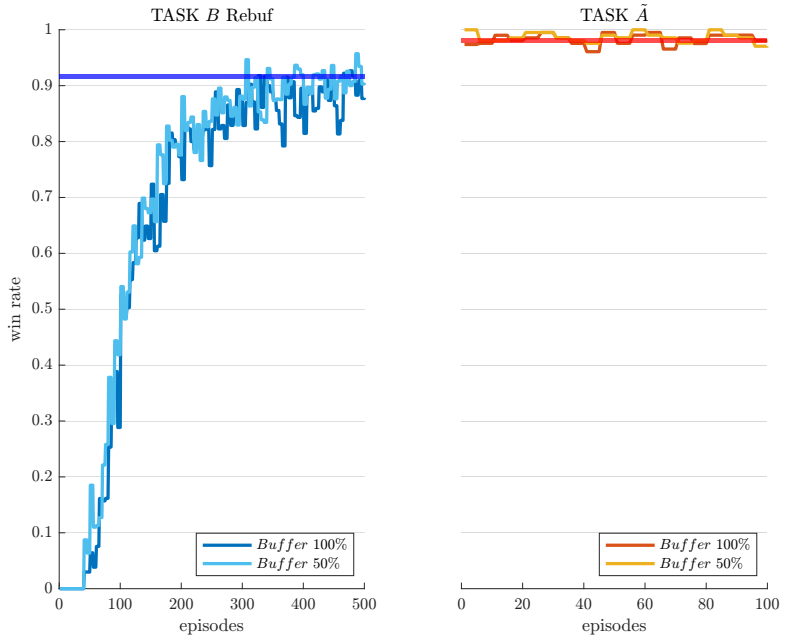


Figure 7.13: map: `3s_vs_4z`*, comparison between different values of the buffer size*

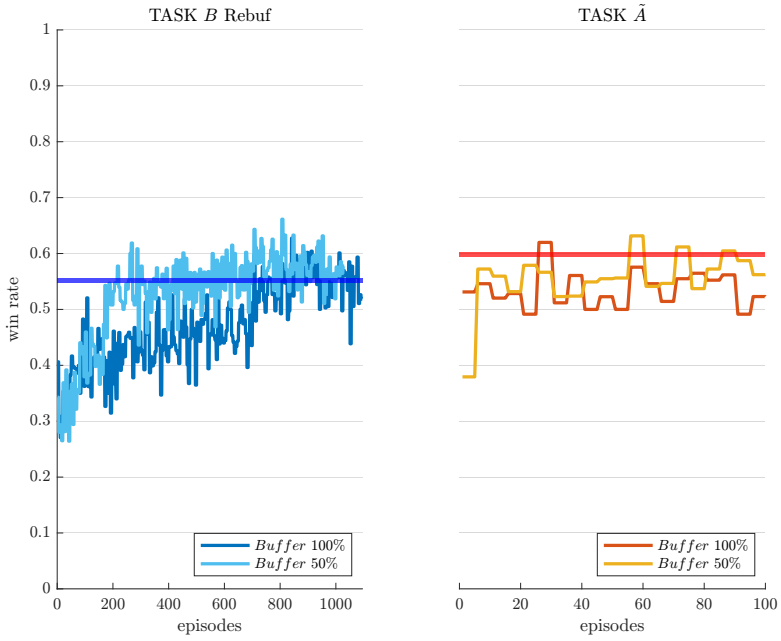


Figure 7.14: map: `3s5z`*, comparison between different values of the buffer size*

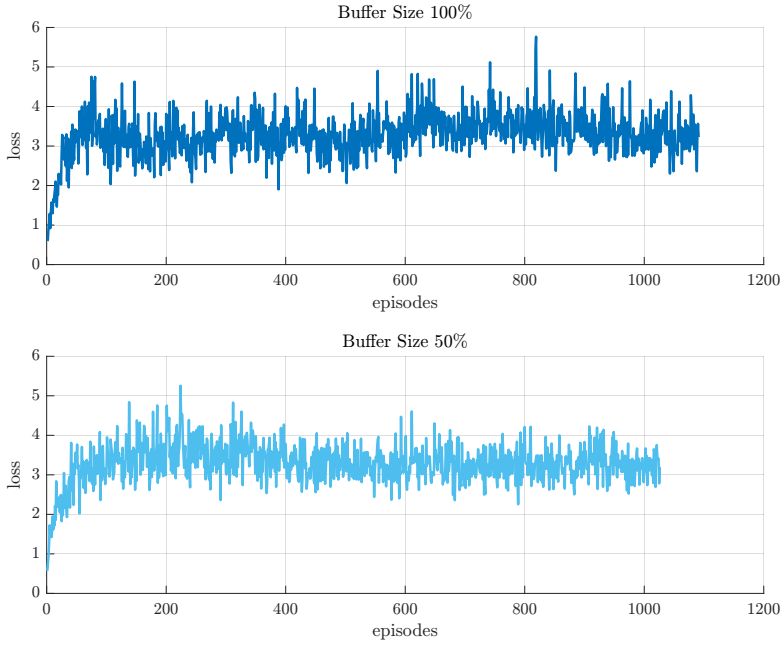


Figure 7.15: map: `3s5z`*, comparison between l1 losses with different buffer sizes*

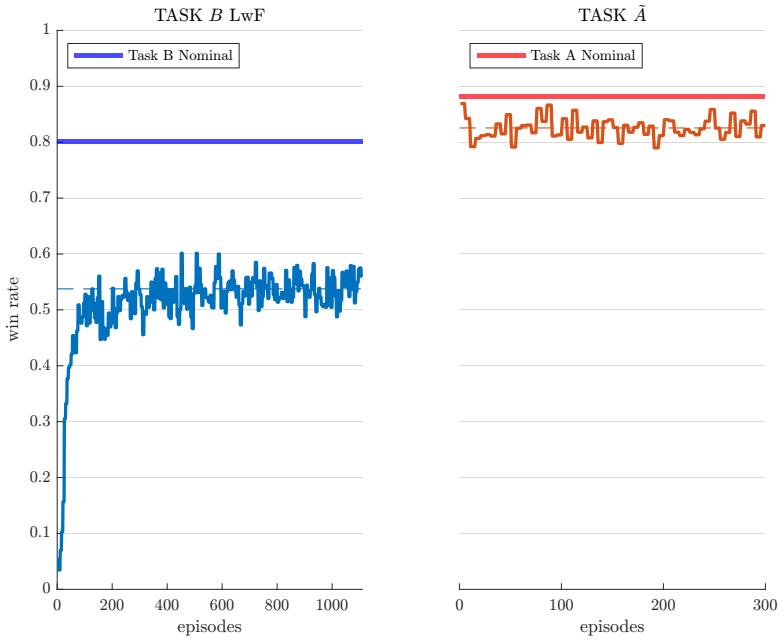


Figure 7.16: map: `8m`, LwF training with $l2$ loss, $\lambda = 1 * 10^{-2}$

self" as in the buffer method, the new agent is potentially misled into making decisions that may not beneficial with the new team.

For the choice of λ, the same considerations can be done as in the buffer case, as it can be observed a similar pattern in the stability-plasticity trade off, for example in fig. 7.17.

In conclusion, the best approach to perform continual learning is to use the replay method, with reduced size of the buffer and with $\lambda = 1 * 10^{-3}$. The comparison between all the architecture is reported in fig. 7.19

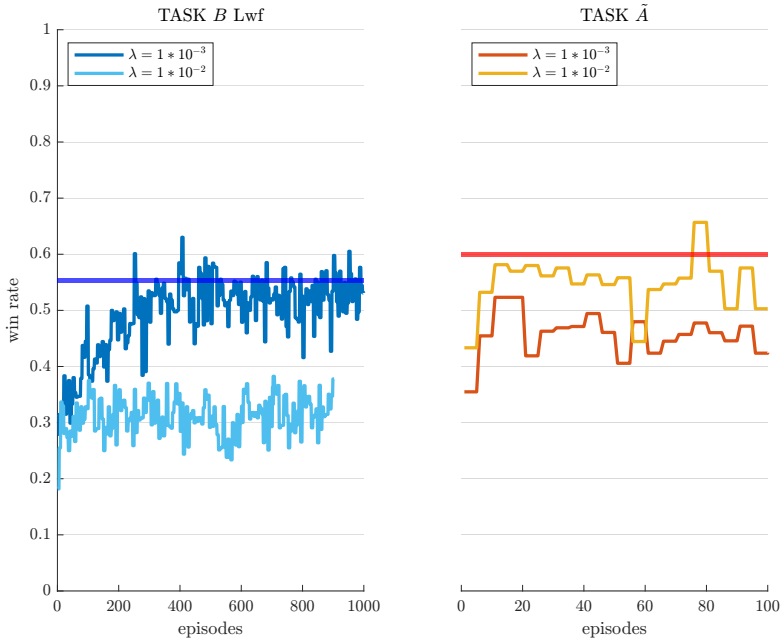


Figure 7.17: map: `3s5z`, LwF with l2 loss, comparison between different values of λ

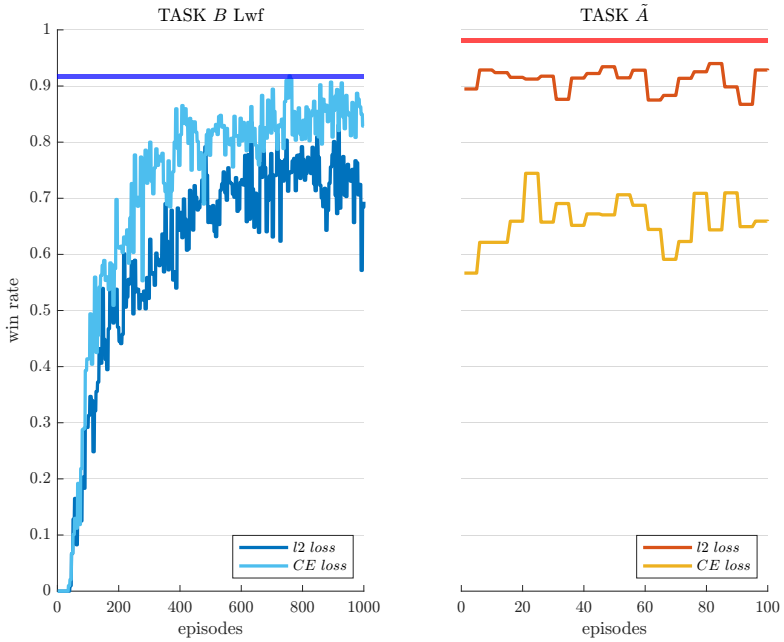


Figure 7.18: map: `3s_vs_4z`, LwF with l2 loss, comparison between different values of λ

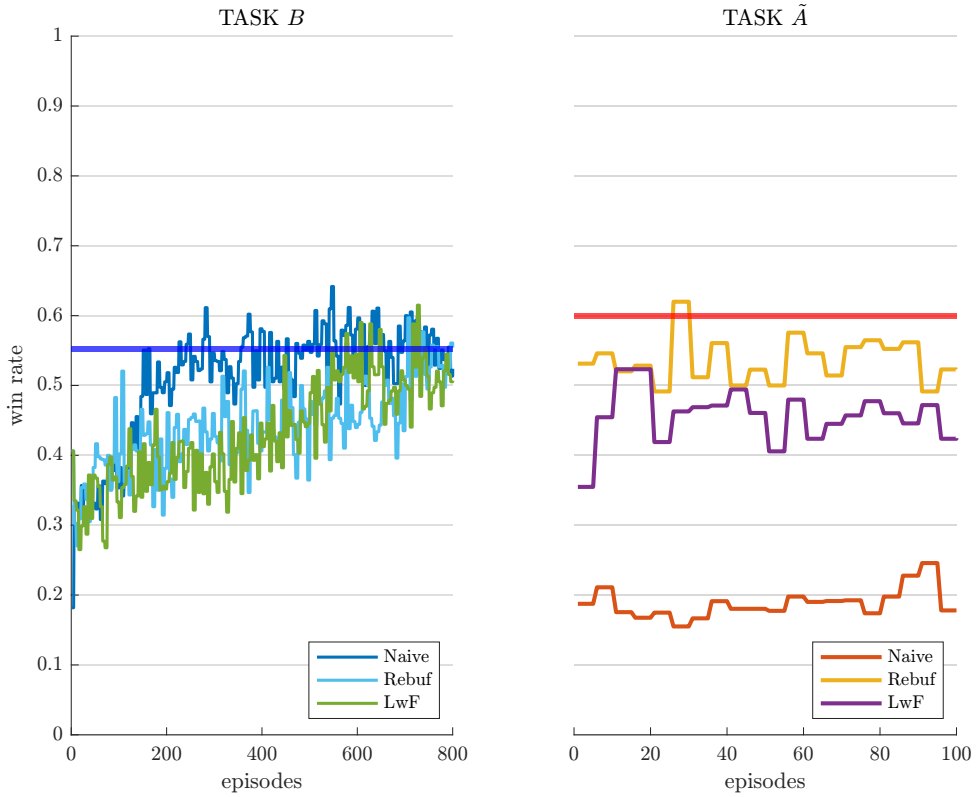


Figure 7.19: map: `3s5z`*, Comparison between the architectures*

Chapter 8

Conclusions

The purpose of this thesis was to approach the complex, yet full of unexplored potential, world of continual multi agent reinforcement learning in the context of fully collaborative games. In particular, two environments were explored, MPE with simple scenarios of collaborations between a few agents to perform navigation tasks, and then on Starcraft, an RTS game that due to its partially unobservable environment, complex reward structure and the fact that it requires a real time strategy has become successful in the RL world. To test the potential of continual learning and apply it to multi agent, the following problem was chosen: first, train different teams with different strategies in the aforementioned environments; then, shuffle the teams and test whether simple continual learning architectures are sufficient to prevent catastrophic forgetting.

A replay method was implemented that exploits a buffer of samples from old tasks to refresh the memory during training, and the effects of changing the buffer size and the regularization parameter were studied. The other method used was knowledge distillation, which uses the neural network model from the old task to help retaining prior knowledge. However, this method has not found satisfactory results, so in the end, buffer replay, the simpler approach, remains the best choice.

Future developments of this work may be directed toward testing other configurations where more than one agent is substituted to study the effects of collaboration and catastrophic forgetting on a batch of individuals, it may also be useful to try prolonged learning on more than one task. To improve the efficiency of the replay method, attention could be focused on sample selection methods, to ensure that only those actually relevant to the agent are stored. Moreover, through a probabilistic investigation aimed at raising the internal variance of the buffer, redundant samples could be discarded to reduce

memory usage. With an incremental number of tasks the implementation of the replay buffer proposed in this thesis becomes inconvenient given the growing requirement for the samples to be saved, so other continual learning techniques need to be implemented, perhaps exploiting features and similarities between tasks to grant greater generalization ability.

Bibliography

[1] Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic. *CoRR*, abs/1205.4839, 2012.

[2] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024.

[3] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(8):5362–5383, 2024.

[4] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017.

[5] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre M. Bayen, and Yi Wu. The surprising effectiveness of MAPPO in cooperative, multi-agent games. *CoRR*, abs/2103.01955, 2021.

[6] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge, 2019.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[8] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function

approximation. *Advances in neural information processing systems*, 12, 1999.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[10] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[11] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020.

[12] Hepeng Li and Haibo He. Multi-agent trust region policy optimization, 2023.

[13] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning, 2022.

[14] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[15] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[16] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *CoRR*, abs/1802.07569, 2018.

[17] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference, 2019.

[18] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning, 2019.

[19] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning, 2022.

[20] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem, 2019.

[21] Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximally interfered retrieval, 2019.

[22] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning, 2017.

[23] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2021.

[24] Zhizhong Li and Derek Hoiem. Learning without forgetting, 2017.

[25] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[26] Hadi Nekoei, Akilesh Badrinaaraayanan, Aaron Courville, and Sarath Chandar. Continuous coordination as a realistic scenario for lifelong learning, 2021.

[27] Lei Yuan, Lihe Li, Ziqian Zhang, Feng Chen, Tianyi Zhang, Cong Guan, Yang Yu, and Zhi-Hua Zhou. Learning to coordinate with anyone, 2023.

[28] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations, 2018.

[29] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017.

[30] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline, 2020.