Università degli Studi di Verona

Dipartimento di Informatica - S.S.D. INF/01

Doctoral Thesis - XXXIV Cycle

Enhancing Exploration and Safety in Deep Reinforcement Learning

Author:
Enrico Marchesini

Advisor:
Prof. Alessandro Farinelli

ISBN: T.B.D.

Enhancing Exploration and Safety in Deep Reinforcement Learning
Enrico Marchesini
Tesi di Dottorato
Verona, 10 Marzo 2022

Whenever someone asks me if reinforcement learning can solve their problem,
I tell them it can't. I think this is right at least 70% of the time.

Alex Irpan (2018)

Abstract

A Deep Reinforcement Learning (DRL) agent tries to learn a policy maximizing a long-term objective by trials and errors in large state spaces (Sutton and Barto, 2018). However, this learning paradigm requires a non-trivial amount of interactions in the environment to achieve good performance. Moreover, critical applications, such as robotics (OpenAI et al., 2019), typically involve safety criteria to consider while designing novel DRL solutions. Hence, devising safe learning approaches with efficient exploration is crucial to avoid getting stuck in local optima, failing to learn properly, or causing damages to the surrounding environment (Garcıa and Fernández, 2015).

This thesis focuses on developing Deep Reinforcement Learning algorithms to foster efficient exploration and safer behaviors in simulation and real domains of interest, ranging from robotics to multi-agent systems. To this end, we rely both on standard benchmarks, such as SafetyGym (Ray et al., 2019), and robotic tasks widely adopted in the literature (e.g., manipulation (Gu et al., 2017), navigation (Tai et al., 2017)). This variety of problems is crucial to assess the statistical significance of our empirical studies and the generalization skills of our approaches (Henderson et al., 2018).

We initially benchmark the sample efficiency versus performance trade-off between value-based and policy-gradient algorithms. This part highlights the benefits of using non-standard simulation environments (i.e., Unity (Juliani et al., 2018)), which also facilitates the development of further optimization for DRL. We also discuss the limitations of standard evaluation metrics (e.g., return) in characterizing the actual behaviors of a policy, proposing the use of Formal Verification (FV) (Liu et al., 2019) as a practical methodology to evaluate behaviors over desired specifications.

The second part introduces Evolutionary Algorithms (EAs) (Fogel, 2006) as a gradient-free complimentary optimization strategy. In detail, we combine population-based and gradient-based DRL to diversify exploration and improve performance both in single and multi-agent applications. For the latter, we discuss how prior Multi-Agent (Deep) Reinforcement Learning (MARL) approaches hinder exploration (Rashid et al., 2018), proposing an architecture that favors cooperation without affecting exploration.

Contents

Chapter 1

Introduction

Life forms on earth are born inheriting character and behavioral traits, yet they do not know how to behave effectively in their native environment. For example, in the human race, a child must learn to walk, a teenager to drive, and a parent to care for children. Meanwhile, they must also learn how to interact safely with their surroundings because changing the environment to avoid hazards is not always possible (e.g., covering the edges of furniture for children).

Decision-making is thus the foundation brick that enables life forms, including artificial ones, to explore and learn novel behaviors. Deciding how to act also represents the common factor for all the situations where a subject acquires and uses a particular skill while avoiding hazards. In addition, every decision influences the surrounding environment either with short or long-term consequences, and the multitude of actions coming from different individuals cause the scenario to be uncertain. Hence, collecting and adapting past experiences to unseen situations is crucial for achieving the objective without risks.

Drawing inspiration from behaviorist psychology, Reinforcement Learning (RL) models decision-making problems using agents that act in an environment to learn a policy, solving a specific task. Hence, agents simulate the typical behavior of biological (or artificial) subjects, interacting with the environment in a trial and errors fashion to maximize a long-term objective called *return* (Sutton and Barto, 2018). Consequently, the typical issues we raised in the context of life forms (i.e., exploring efficiently to learn robust behaviors while fostering safety) are also two critical challenges in modern RL.

Due to the tremendous performance of non-linear function approximators to deal with large state-action spaces, research in the Reinforcement Learning area recently focused on using Deep Neural Networks (DNNs). This field of research known as Deep Reinforcement Learning (DRL) has recently driven astonishing progress in a wide variety of domains, ranging from robotics (OpenAI et al., 2019; Gu et al., 2017; Tai et al., 2017; Tan et al., 2018) to games (Silver et al., 2018a; Mnih et al., 2013a; Vinyals et al., 2019). The peculiar ability to generalize the decision-making process in unseen situations makes this learning framework suitable for high-dimensional sequential problems. A key example is the ability of DRL to outclass prior supervised and unsupervised learning paradigms, attaining superhuman performance in Chess, Shogi, and Go within a single self-play system that mastered these games at a superhuman level (Silver et al., 2018b). However, despite the impressive successes, Deep Reinforcement Learning suffers from the issues of decision-making systems and introduces other limitations associated with the use of DNNs (e.g., convergence to local

optima). Moreover, the large amount of environment interactions required to learn makes sample efficiency and safety the crucial problems in recent research for practical applications. Hence, devising safer learning approaches that efficiently explore the environment to learn robust behaviors represents one of the core issues that hinder a broader application of Deep Reinforcement Learning-based techniques to real-world problems.

1.1 Contributions

This thesis focuses on improving exploration and safety of learned behaviors by augmenting standard gradient-based optimization with Evolutionary Algorithms (EAs) (Fogel, 2006; Montana and Davis, 1989) and Formal Verification (FV) (Liu et al., 2019; Singh et al., 2019) techniques. We begin by summarizing the challenges we address that contain the main body of our work.

1.1.1 Benchmarking Exploration

The non-trivial number of interactions with the environment required by Deep Reinforcement Learning algorithms to achieve human (or superhuman) performance is one of the main factors that hinder its applicability outside of research demos (Pathak et al., 2017; Ostrovski et al., 2017). For example, the first algorithm capable of outperforming average human performance on all the suite of 57 Atari games required several orders of magnitude more iterations than the ones of a human player (Badia et al., 2020). This inefficiency intuitively becomes more evident in the context of multi-agent applications, where the cardinality of the state-action space to explore grows exponentially in the number of agents.

The sample efficiency problem and how we collect and re-use past samples represent the first challenge in our work. In more detail, recent Deep Reinforcement Learning literature for robotic control mainly focuses on policy-gradient (or actor-critic) algorithms with continuous action spaces. However, it is well-known that value-based algorithms with discrete action spaces offer a better sample efficiency, possibly resulting in shorter training times. We summarize such efficiency in Figure 1.1 that highlights a clear distinction in terms of sample efficiency among the variety of Reinforcement Learning techniques.

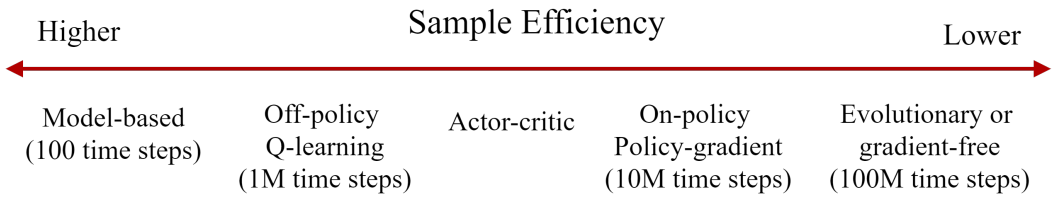

FIGURE 1.1: Schematic overview of Reinforcement Learning sample efficiency according to Dorner (2021).

In this direction, we benchmark (on-policy) policy-gradient and (off-policy) value-based Deep Reinforcement Learning approaches in robotic applications to show their efficiency and discuss how the experience collection and management process can be improved to increase the performance of existing algorithms.

Moreover, we note that the typical evaluation of a DRL system relies (possibly) on non-informative metrics (e.g., cumulative reward or success rate). Hence, we argue that such evaluations do not guarantee the desired behaviors of the trained policies in scenarios involving safety or high-cost equipment (e.g., robotics). To this end, we employ Formal Verification (Corsi et al., 2021) to characterize the Deep Neural Network's decision-making process.

1.1.2 Enhancing Exploration

Improving the way samples are collected, memorized, and retrieved can drastically improve performance. However, Deep Reinforcement Learning algorithms also suffer from convergence to local optima due to the typical non-linear nature of Deep Neural Networks and the lack of diverse exploration when operating in high-dimensional spaces. While local optima are intrinsic to using these function approximators, the latter is an active topic of interest.

Several studies address the lack of exploration relying on sensitive domain-specific hyper-parameters (e.g., curiosity-driven and count-based exploration (Pathak et al., 2017; Ostrovski et al., 2017)). The sensitivity to such hyper-parameters, however, represents a significant issue for Deep Reinforcement Learning as it typically results in brittle convergence properties and poor performance in practical tasks (Haarnoja et al., 2018; Henderson et al., 2018).

Conversely, Evolutionary Algorithms (Fogel, 2006) have been recently employed as a promising gradient-free optimization alternative over gradient-based DRL. The implicit redundancy of these population-based approaches has the advantages of enabling diverse exploration and improving robustness, leading to a more stable convergence. In particular, Genetic Algorithms (GAs) (Montana and Davis, 1989) achieved competitive returns compared to gradient-based Deep Reinforcement Learning (Such et al., 2017), and are characterized by lower computational cost (see Figure 1.2 for the general flow of a standard GA). These gradient-free approaches, however, struggle to solve high-dimensional problems having poor generalization skills and, as shown in Figure 1.1, are significantly less sample efficient than gradient-based methods. In more detail, genetic approaches do not use gradient information to guide the learning process. Hence they require a higher number of interactions to match standard DRL returns. However, the lack of gradient computation also makes Genetic Algorithm interactions significantly faster.

In contrast to this dual perspective that uses either gradient-based Deep Reinforcement Learning or gradient-free Evolutionary Algorithms, we follow an emergent research direction that proposes the combination of the two families of approaches. This combination takes inspiration from the physical world, where evolution and learning cooperate in assimilating the best of both solutions (Simpson, 1953). In this direction, we discuss the design of novel frameworks that merges the beneficial aspects of GAs with (possibly) any DRL algorithm.

Moreover, we extend our discussion to the Multi-Agent (Deep) Reinforcement Learning (MARL) domain, where value decomposition algorithms represent a recent research direction to improve performance and favor cooperative behaviors (Sunehag et al., 2018; Rashid et al., 2018, 2020; Son et al., 2019). In this context, enhancing the exploration skills of the agents is crucial as the typical non-stationarity environment

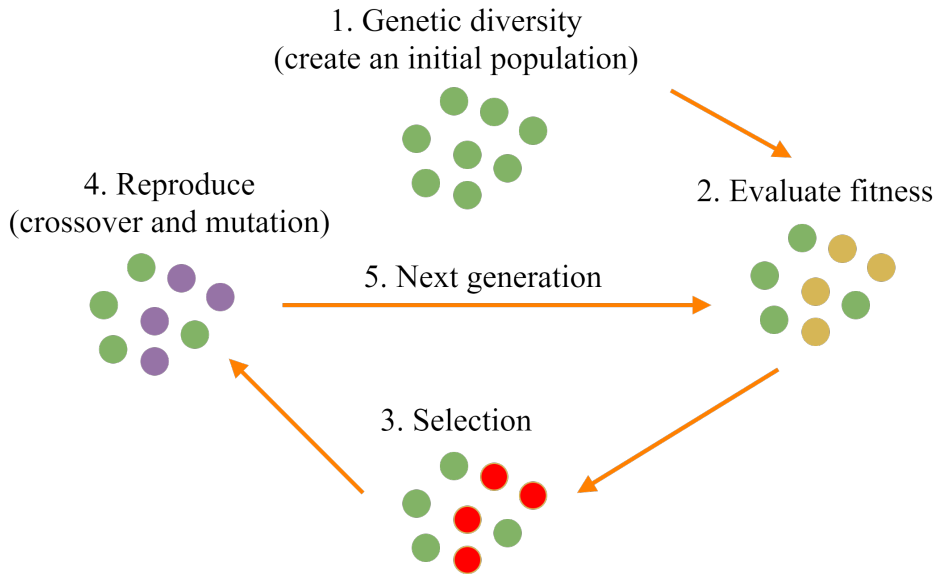

FIGURE 1.2: Typical flow of a Genetic Algorithm. After creating an initial population, a GA evaluates and selects the best individuals in the population according to a specified criteria. Individuals are then combined via evolutionary operators (i.e., crossover and mutation) to form a new population. This process repeats until a convergence criteria is met.

makes it harder to explore the environment and foster cooperative behaviors efficiently. Hence, we discuss the design of value decomposition algorithms and propose an evolutionary search to enhance performance further.

1.1.3 Fostering Safer Behaviors

Learning how to interact within the surrounding environment safely is the crucial component that allows life forms to grow and evolve while learning novel behaviors. Fostering a safer interaction with the environment is vital to avoid causing damages to the surroundings, nearby subjects, or the agent itself. Safe Deep Reinforcement Learning is thus a crucial research area, complementary to the issues related to exploration that we have to address for promoting broader applicability of DRL systems to real-world scenarios.

In this direction, several learning paradigms incorporate auxiliary objectives, similar to the reward function, to model safe (or unsafe) situations (Garcıa and Fernández, 2015) and maintain safety specifications separate from the task objective (e.g., the return). Such additional signal, which the literature typically refers to as *cost*, is then exploited to foster safety using different techniques.

For example, Multi-Objective (Deep) Reinforcement Learning (MORL) aims at learning policies to optimize the different criteria simultaneously (Liu et al., 2015; Yang et al., 2019). Hence, it is possible to change the underlying optimization process to minimize the additional cost function. However, explicitly learning behaviors over multiple preferences (i.e., optimizing multiple signals) is challenging, and prior work either converges to an average policy over the objectives or presents poor scalability to high-dimensional spaces (Roijers et al., 2014; Vamplew et al., 2011).

Conversely, a recent Safe Deep Reinforcement Learning trend employs constrained processes (Altman, 1999; Stooke et al., 2020) to incorporate the safety signal in the optimization process via constraints. The idea here is to minimize the additional cost signal (that models unsafe behaviors) while maximizing the return (Chow et al., 2018; Liu et al., 2020). However, constraints hinder exploration and cause a significant trade-off in performance as the goal of such approaches is to limit the cost accumulation at the expense of low returns, which in practice result in poorly learned behaviors (Ray et al., 2019).

A different approach aims at estimating the probability of incurring into unsafe states (i.e., a failure), given the current state-action pair, by training a so-called Safety Critic (Thananjeyan et al., 2020; Bharadhwaj et al., 2021; Thananjeyan et al., 2021). However, these additional estimators could return misleading information, especially in the early stages of the training where the critics are pre-trained on offline data that should cover a wide variety of undesired behaviors to ensure correct estimations. In addition, computing the probability of failure at each step causes non-negligible overhead in the action sampling process. The decision-making of Deep Reinforcement Learning systems must exhibit a tractable computational demand to avoid hindering the application to real-world physical platforms that require high-frequency control (e.g., robotics).

We investigate the Safe DRL field under a different perspective by employing existing Formal Verification techniques to characterize the decision-making process of a Deep Neural Network over desired specifications (Liu et al., 2019), which are typically provided in a safety-critical context. We initially benchmark the performance of Deep Reinforcement Learning algorithms over this additional aspect. Hence, we discuss the design of novel evolutionary operators and training processes that exploits FV to complement existing DRL approaches and foster safer behaviors.

1.1.4 Summary of the Contributions

In summary, the following points highlight the contribution that this thesis makes to the state-of-the-art:

1. In the context of benchmarking exploration:

   - We develop novel robotic simulation environments based on Unity (Juliani et al., 2018),[1] showing that this game engine offers significantly lower training times over traditional simulators such as Gazebo, a standard choice for robotic platforms controlled with the Robot Operating System (ROS).[2] We use two practical tasks:

     (a) Trajectory generation for a manipulator (Gu et al., 2017).

     (b) Mapless navigation for a mobile robot (Tai et al., 2017).

     These real-world inspired tasks are used to show that it is possible to transfer the model trained in Unity on Gazebo and on real robotic platforms (using Robot Operating System). Crucially, this is possible without additional tuning or training.

[1] www.unity.com
[2] www.ros.org

- We benchmark policy-gradient, actor-critic, and value-based approaches in the practical mapless navigation domain, which is a widely adopted benchmark in recent Deep Reinforcement Learning literature for robotics (Tai et al., 2017; Zhang et al., 2017; Kretzschmar et al., 2016; Wahid et al., 2019; Chiang et al., 2019). Our results show that value-based DRL approaches with discrete action spaces are a more sample efficient alternative over policy-gradient algorithms while resulting in comparable or better performance in tasks that employ physical control. We further enhance the results of these approaches (i.e., returns and training time) using asynchronous parallel training phases and multi-batch memories to employ the visited samples efficiently.
- We leverage Formal Verification to show that standard evaluation metrics (e.g., success rate) do not provide guarantees on the actual behavior of the trained policies. In this direction, we also show that optimizing the training process with, for example, scaling discount factors and supervised exploration also leads to models with better behaviors (e.g., smoother trajectories).

2. Regarding the exploration enhancement:
   - We empirically evaluate prior combinations of Deep Reinforcement Learning and Evolutionary Algorithms in a navigation scenario with a discrete action space. We exploit this evaluation to discuss why the combination of previous work (Khadka and Tumer, 2018; Bodnar, 2020; Pourchot and Sigaud, 2019) with value-based algorithms leads to detrimental performance, obtaining lower return than using the only value-based algorithm.
   - We develop a general framework that allows the combination of EAs with policy-gradient, actor-critic, and value-based algorithms, ensuring to match the performance of the DRL approach in a worst-case scenario (i.e., avoiding drawbacks on the return). Furthermore, we show that our framework addresses the sensitivity of Deep Reinforcement Learning algorithms to hyper-parameters, significantly improving the returns in settings that result in pathological performance of the only DRL component (e.g., a specific seed initialization).
   - We extend the discussion to the Multi-Agent (Deep) Reinforcement Learning setting, where the environment complexity makes it challenging to explore the environment and foster cooperative behaviors efficiently. Furthermore, we discuss the limitation of prior MARL approaches and propose a novel architecture that obtains superior performance and is compatible with our combined approach.

3. Finally, to motivate our perspective on fostering safer behaviors:
   - We use Formal Verification to evaluate the safety of policy-gradient and value-based approaches over desired specifications that model safe behaviors. To this end, we use aquatic navigation as a highly challenging evaluation task due to the non-stationary environment and the uncertainties of the robotic platform. In this scenario, it is crucial to consider the safety aspect by analyzing the behavior of the trained network.

- We introduce a safe-oriented population-based framework characterized by gradient-informed evolutionary operators that work on top of existing Deep Reinforcement Learning algorithms to bias policies toward safety without multi-objective or constrained optimization. Furthermore, we confirm the benefits of our safety-oriented operators in biasing the policies, using Formal Verification.
- We propose to relax the formal guarantees of Formal Verification to analyze the model behaviors tractably in the training loop. Hence, we integrate this component in our safe-oriented evolutionary framework to introduce a form of prior knowledge on safety (i.e., the desired specifications of FV) in the training loop.

1.2 Thesis Outline

When taken together the above contributions pave the way towards the use of DRL approaches in realistic scenarios (e.g., robotics applications). However, the main criticism of Deep Reinforcement Learning is that despite the impressive successes in specific areas, it is known to be hard and does not always lead to valuable results (Alex Irpan, 2018). For example, DRL generally overfits, it is typically unstable and relies on reward functions that are difficult to design. Initialization with different random seeds has thus a crucial role in DRL, and running enough experiments is the current known way to address such issues (Henderson et al., 2018).

For this reason, before discussing the thesis outline, in what follows we clarify our vision on Deep Reinforcement Learning research that brings us to the current manuscript. We initially focused on understanding the pros and cons of state-of-the-art Deep Reinforcement Learning approaches in widely considered benchmarks. This approach helps to become familiar with the main concepts and terminology in DRL, focusing on current algorithms to deeply understand their insights, applicability, and limitations. Such extensive analysis naturally shifted our attention to considering the same aspects for the environments where evaluation metrics are typically collected. In this context, we broadly characterize the limitations of current benchmarks, designing novel problem settings to make progress on them. Thus, among the wide range of topics available in the literature (e.g., exploration, sim-to-real transfer, multi-agent), we chose to push the boundaries of exploration, which is the crucial component responsible for the performance of (possibly) any Reinforcement Learning model. In this direction, we explored novel ways to address the limitation of state-of-the-art approaches, by proposing evolutionary approaches and novel value decomposition methodologies. Hence, Safe Deep Reinforcement Learning represented the natural path for our research due to its close connection with exploration, the critical importance in real-world applications (such as the ones investigated in the first part, i.e., robotic navigation in static and dynamic environments and manipulation) and, more generally, the future of this research field.

Regarding the thesis outline, we begin with a review chapter that covers basic knowledge of Deep Neural Network and Deep Reinforcement Learning which are necessary to understand the following parts and lay the conceptual notations for the manuscript. Therefore, we divide the thesis into three main parts that cover the contributions of our work, which we briefly summarize in the following:

Benchmarking Exploration

Part I focuses on the importance of choosing the appropriate approach for a particular task, benchmarking different DRL algorithms (i.e., value-based, policy-gradient, and actor-critic) in robotics simulators. We propose further optimizations (i.e., asynchronous parallel training, multi-batch memory, scaling discount factor) that improve the overall return while reducing training time.

- Chapter 3 (*Unity Simulation for Robotics*) discusses state-of-the-art results in Deep Reinforcement Learning applied to robotic platforms, that recent work achieved mainly using ROS-based simulations and transferring the policy on real platforms (Zhao et al., 2020; Ding et al., 2020). Hence, we present three robotic environments:

  1. Trajectory generation for a commercial manipulator (Section 3.3).
  2. Mapless navigation for an indoor mobile robot (Section 3.4.1).
  3. Navigation for an outdoor aquatic drone (Section 3.4.2).

  These environments use novel simulation software, Unity (Juliani et al., 2018). In particular, we show that it is possible to export a model trained in a Unity environment, to a conventional ROS simulator (i.e., Gazebo and RViz) and the real robot, without additional training. This work has previously been presented in Marchesini et al. (2019); Marchesini and Farinelli (2020a).

- Chapter 4 (*Benchmarking Sample Efficiency*) presents an evaluation of existing Deep Reinforcement Learning algorithms and further optimizations into the robotic scenarios. Our goal is to improve the performance of existing approaches and highlight the importance of choosing a particular algorithm, depending on task-related requirements (e.g., smoother navigation, shorter trajectories). In particular, we show that value-based DRL could be a viable and more efficient alternative over policy-gradient and actor-critic algorithms for tasks of practical interest. This work has previously been presented in Marchesini and Farinelli (2020a); Marzari et al. (2021).

- Chapter 5 (*Evaluating Decision-Making*) shows the limitations of standard evaluation parameters (e.g., cumulative reward), which are not informative enough in evaluating the behaviors of a trained model. To this end, we discuss the designing of formal metrics to measure the safety of Deep Neural Network using prior Formal Verification approaches. This work has previously been presented in Corsi et al. (2020); Pore et al. (2021).

Enhancing Exploration

Part II addresses the limitations of Deep Reinforcement Learning in the context of achieving efficient exploration in single and multi-agent domains. To this end, we discuss two different approaches:

1. We focus on Evolutionary Algorithms to complement prior gradient-based algorithms with population-based approaches.
2. We propose a value decomposition solution to favor cooperative behaviors in multi-agent settings.

In more detail:

- Chapter 6 (*Combining Deep Reinforcement Learning with Evolutionary Algorithms*) highlights the recent trend of merging the benefits of gradient-based and gradient-free approaches. We discuss the current methodologies designed in this field and, following the insights of Chapter 4, we highlight their significant performance trade-off when applied to value-based Deep Reinforcement Learning algorithms.
- Chapter 7 (*Genetic Soft Updates for Policy Evolution*) presents our combined framework that, in contrast to prior work, is compatible with any DRL approach. We show that our population-based solution generates diverse experiences and finds better policies, improving the robustness of the overall approach to detrimental initialization that would cause pathological performance to the only Deep Reinforcement Learning algorithm. This work has previously been presented in Marchesini and Farinelli (2020b); Marchesini et al. (2021).
- Chapter 8 (*Global Dueling Q-Learning*) extends our work to the Multi-Agent (Deep) Reinforcement Learning domain. In detail, we discuss how current MARL algorithms hinder exploration, proposing a Deep Neural Network architecture to overcome such limitations. Moreover, we extend this approach with our insights on the combination of DRL with Evolutionary Algorithms. This work has previously been presented in Marchesini and Farinelli (2021a,b).

Exploring for Safer Behaviors

Part III analyzes the effects of prior Safe Deep Reinforcement Learning work (Ray et al., 2019; Stooke et al., 2020; Liu et al., 2020; Thananjeyan et al., 2020) on exploration, proposing a different perspective to foster safer behaviors using Evolutionary Algorithms and Formal Verification.

- Chapter 9 (*Quantifying Safe Behaviors*) uses our evaluation metrics based on FV to quantify the number of correct decisions that a trained Deep Neural Network makes over desired specifications. We evaluate policy-gradient and value-based approaches, along with their combination with EAs, in a complex non-stationary scenario (i.e., aquatic navigation) to discuss the benefits of each solution in the context of Safe Deep Reinforcement Learning. We also employ FV to confirm further that our evolutionary framework biases the exploration process in the direction of more robust policy regions with higher returns. This work has previously been presented in Marchesini et al. (2021a); Corsi et al. (2021).
- Chapter 10 (*Safety-Oriented Search*) proposes a different perspective to foster safety, using Evolutionary Algorithms with gradient-informed mutations designed to explore safer behaviors. We also explore an approximated form of Formal Verification to inject prior knowledge on desired safety specifications in the training loop without significant overhead. This work has previously been presented in Marchesini et al. (2021b); Corsi et al. (2020).

We conclude in Chapter 11 (*Conclusions and Future Directions*) with a broad discussion of progress and possible future research directions in the context of Deep Reinforcement Learning exploration and safety.

1.3 Related Work

In this section, we provide a high-level overview of the most related work to this thesis. We seek to provide the reader with a general overview of the state-of-the-art while clarifying our line of research. We refer to the individuals' chapters for a much more thorough review of related work.

State-of-the-art results in Deep Reinforcement Learning have been achieved mainly using simulation and transferring the policy on real platforms (Juliani et al., 2018; OpenAI et al., 2019; Zhao et al., 2020; Ding et al., 2020). For this reason, simulation environments are the key component that allows the design and evaluation of novel approaches. For example, Arcade Learning Environment (ALE) (Machado et al., 2018) is still a widely adopted evaluation benchmark for value-based algorithms (Mnih et al., 2013b; van Hasselt et al., 2016; Wang et al., 2016; Hessel et al., 2018; Bellemare et al., 2017), while the MuJoCo (Todorov et al., 2012) physics-based tasks are a benchmark for policy-gradient solutions (Lillicrap et al., 2016; Fujimoto et al., 2018; Schulman et al., 2017; Haarnoja et al., 2018). However, benchmarking environments are constrained by the limitations of the simulation platforms that can not always provide meaningful challenges to novel learning systems. Hence, exploring novel frameworks for the development of testing environments is crucial (Juliani et al., 2018). Despite the significant progresses achieved in simulation, several works (Mania et al., 2018; Henderson et al., 2018; Colas et al., 2019) criticized the importance of choosing benchmarks to evaluate DRL research. It is well known that reproducibility, proper experimental techniques, and reporting procedures for the results are crucial, yet often underestimated points in recent literature.

The lack of diverse exploration in high-dimensional spaces is one of the core issues in Deep Reinforcement Learning that brings together the problems related to robust evaluations and broader applications to more complex scenarios. Moreover, Multi-Agent (Deep) Reinforcement Learning tasks aggravate the space dimensionality issue further, introducing other problems related, for example, to the cooperation/competition of the agents and their credit assignment (Sunehag et al., 2018; Rashid et al., 2018). Standard exploration strategies (e.g., ϵ-greedy action selection (Sutton and Barto, 2018), noisy networks (Fortunato et al., 2017), stochastic policies (Schulman et al., 2017)) are inadequate when considering sparse reward or action/observation spaces that are too large. Intrinsic motivation approaches have been adopted to encourage better exploration (Hong et al., 2018; Ostrovski et al., 2017; Conti et al., 2018; Pathak et al., 2017). However, such methods typically rely on sensitive task-specific hyper-parameters that hinder convergence properties and reproducibility further. In contrast, Evolutionary Algorithms (Fogel, 2006; Montana and Davis, 1989) naturally offer diverse and robust exploration due to their population-based formulation. For this reason, EAs have been adopted to complement gradient-based Deep Reinforcement Learning, merging the benefits of both solutions (Khadka and Tumer, 2018; Khadka et al., 2019; Pourchot and Sigaud, 2019; Colas et al., 2018).

Hence, we focus on the field of Safe DRL, where exploration and safety evaluations are critical (Ray et al., 2019). In this context, we analyze how prior work based on constrained approaches limits exploration, resulting in a significant performance trade-off (i.e., reward versus safety) (Stooke et al., 2020; Chow et al., 2018; Achiam et al., 2017; Liu et al., 2020). We propose a novel direction to avoid constraints, demonstrating the benefits of informed evolutionary operators devoted to safety while

using recent verification techniques (Wang et al., 2018b,c) both to provide a formal evaluation of the policy behaviors and to inject safety specifications into the agent.

1.4 Publications

Most of the contributions discussed in Section 1.1.4 appeared in top-level conferences and journals. In more detail:

1. The content of Part I regarding Unity simulations (Chapter 3), benchmarking Deep Reinforcement Learning algorithms and further optimizations (Chapter 4), and using different metrics to evaluate the trained models (Chapter 5), has been published in Marchesini et al. (2019); Marchesini and Farinelli (2020a); Marzari et al. (2021); Corsi et al. (2020); Pore et al. (2021).
2. The contributions of Part II, where we describe how to enhance exploration combining DRL with Evolutionary Algorithm (Chapter 7), and its extension to multi-agent systems (Chapter 8) instead have been published in Marchesini et al. (2021); Marchesini and Farinelli (2020b, 2021a,b).
3. The contents of Part III regarding the use of Formal Verification as a benchmark evaluation metrics for Safe Deep Reinforcement Learning (Chapter 9), the safe-oriented framework with gradient-informed evolutionary operators and FV (Chapter 10) have been published, or currently under submission, in Marchesini et al. (2021a); Corsi et al. (2021, 2020); Marchesini et al. (2021b).

The mentioned publications, as well as other contributions related to the field of model-based planning under uncertainty (which we do not discuss in the thesis as it is unrelated to the exploration and safety thematics), are listed in the following:

- E. Marchesini, D. Corsi, A. Farinelli, "Exploring Safer Behaviors for Deep Reinforcement Learning". AAAI Conference on Artificial Intelligence, 2022.
- E. Marchesini, A. Farinelli, "Enhancing Deep Reinforcement Learning Approaches for Multi-Robot Navigation via Single-Robot Evolutionary Policy Search". IEEE International Conference on Robotics and Automation (ICRA), 2022.
- L. Marzari, D. Corsi, E. Marchesini, A. Farinelli, "Enhancing Deep Reinforcement Learning Mapless Navigation Using Curriculum Learning". ACM/SIGAPP Symposium On Applied Computing (SAC), 2022.
- E. Marchesini, D. Corsi, A. Farinelli, "Genetic Soft Updates for Policy Evolution in Deep Reinforcement Learning". International Conference on Learning Representations (ICLR), 2021.
- D. Corsi, E. Marchesini, A. Farinelli, "Formal Verification of Neural Networks for Safety-Critical Tasks in Deep Reinforcement Learning". Conference on Uncertainty in Artificial Intelligence (UAI), 2021.
- E. Marchesini, D. Corsi, A. Farinelli, "Formal Analysis of Decision-Making Models for Aquatic Navigation using Combined Deep Reinforcement Learning". IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021.

- E. Marchesini, A. Farinelli, "Centralizing State-Values in Dueling Networks for Multi-Robot Reinforcement Learning Mapless Navigation". IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021.
- A. Pore, E. Marchesini, D. Corsi, D. Dall'Alba, A. Casals, A. Farinelli, P. Fiorini, "Safe Reinforcement Learning using Formal Verification for Tissue Retraction in Autonomous Robotic-Assisted Surgery". IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021.
- D. Corsi, E. Marchesini, A. Farinelli, "Evaluating the safety of deep reinforcement learning models using semi-formal verification". arXiv, 2021.
- A. Castellini, E. Marchesini, A. Farinelli, "Partially Observable Monte Carlo Planning with State Variable Constraints for Mobile Robot Navigation". Engineering Applications of Artificial Intelligence (EAAI), 2021.
- M. Zuccotto, M. Piccinelli, A. Castellini, E. Marchesini, A. Farinelli, "Learning state-variable relationships in POMCP: a framework for mobile robots". Submitted at Frontiers, 2021.
- E. Marchesini and A. Farinelli, "Discrete Deep Reinforcement Learning for Mapless Navigation". IEEE International Conference on Robotics and Automation (ICRA), 2020.
- E. Marchesini and A. Farinelli, "Genetic Deep Reinforcement Learning for Mapless Navigation". International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2020.
- E. Marchesini, D. Corsi, A. Farinelli, P. Fiorini, "Formal Verification for Safe Deep Reinforcement Learning in Trajectory Generation". International Conference on Robotic Computing (IRC), 2020.
- A. Castellini, E. Marchesini, and A. Farinelli, "Explaining the influence of prior knowledge on POMCP policies". European Conference on Multi-Agent Systems (EUMAS), 2020.
- E. Marchesini, D. Corsi, A. Benfatti, A. Farinelli, and P. Fiorini, "Double Deep Q-Network for Trajectory Generation of a Commercial 7DOF Redundant Manipulator". International Conference on Robotic Computing (IRC), 2019.

Chapter 2

Background

This chapter lays the main concepts and provides key notation used in the following parts of the thesis. In detail, we briefly introduce the use of non-linear function approximators such as Deep Neural Networks to handle high-dimensional spaces. Hence, we frame the main components of a Reinforcement Learning and Deep Reinforcement Learning problems. This part recalls the main concepts and formalization of seminal works and courses in the field such as (Sutton and Barto, 2018; Goodfellow et al., 2016; Silver, 2020; Levine, 2020; Achiam, 2021) and serves to provide a general overview of the family of problems we consider. We continue by highlighting the field of Evolutionary Algorithms and giving a more detailed description of Genetic Algorithms, which are then broadly employed in Part II and III of the thesis. We also provide a brief overview of Formal Verification, focusing on reachability-based methods that are used as a practical tool in our contributions. We conclude the chapter with an overview of state-of-the-art methodologies that address exploration and safety problems in DRL, which are the core challenges that we face in this manuscript.

2.1 Deep Neural Networks

Pivotal progress in a variety of open challenges such as autonomous driving (Codevilla et al., 2019), Natural Language Processing (NLP) (Brown et al., 2020), and protein folding (Jumper et al., 2021), have been driven by the advances of learning methods based on Neural Networks (NNs).

These function approximators take inspiration from biological neural networks that constitute the brain of life forms. Due to their similarity to biological neurons, a Neural Network is a collection of neurons, which are interconnected units (typically) organized in layers of parameterized non-linear transformations. Networks with more layers are thus called Deep Neural Networks and can handle massive amounts of data (Goodfellow et al., 2016). Moreover, the various layers may perform different transformations on their inputs, based on the so-called *activation function* computed in each neuron. Such activations are typically non-linear functions and Figure 2.1 shows three examples of common activations. Hence DNNs are referred to as non-linear function approximators. In more detail, a given input signal flows from the first layer (i.e., input layer) to the last one (i.e., output layer), usually after layers of intermediate computations (i.e., hidden layers) (Goodfellow et al., 2016). Crucially, each layer is parametrized by learnable parameters that serve to approximate the function of interest.

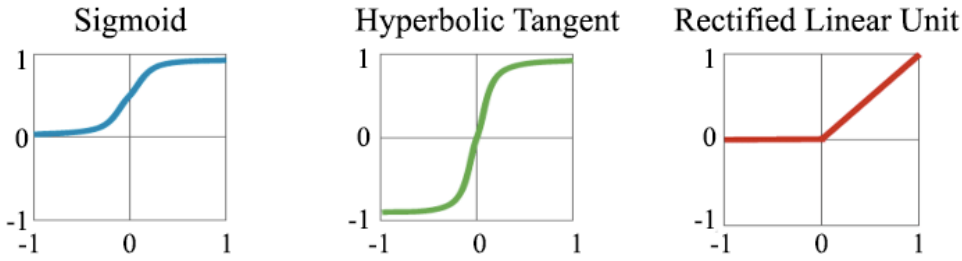

FIGURE 2.1: Graphical overview of three common non-linear activation functions.

This high-level description denotes standard Multi-Layer Perceptron (MLP), where there are not backwards connections, typical of Recurrent Neural Networks (RNNs) (Sepp Hochreiter, 1997) (which we discuss in Section 8 as a practical way to deal with partial-observability in multi-agent applications), or application-specific architectures such as Convolutional Neural Networks (CNNs) (Yann LeCun, 1995). The following chapters mostly consider standard MLPs (or deep feedforward networks (Goodfellow et al., 2016)) that we refer by the letter f with a subscript that refers to its parameters θ, i.e., f_θ.

2.1.1 Gradient Descent

Gradient Descent (GD) (Lemarechal, 2012) is the standard algorithm for learning such parameters θ. As such, we briefly describe it in the following and depicts the general idea in Figure 2.2.

Similarly to training any machine learning model, we first have to define a *loss function* $L(f_\theta)$ (also referred to as a cost or objective function in the literature (Goodfellow et al., 2016)). Such objective measures the overall performance of the DNN by quantifying how "far" are the network's outputs $\hat{y}$ over the desired outputs y. Given the differentiable (or piece-wise differentiable) nature of the non-linear function approximator, GD consists in iteratively updating the parameters by following the

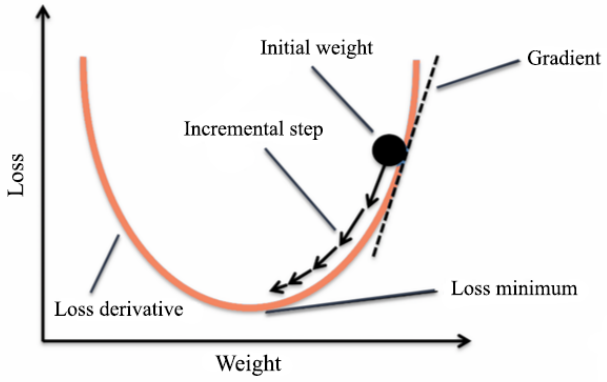

FIGURE 2.2: High-level overview of the Gradient Descent algorithm. Courtesy of Clairvoyant.

direction of the gradient of $L(f_{\theta_i})$. For example, in the case of Gradient Descent, $\theta_{i+1} = \theta_i - \alpha \nabla_{\theta_i} L(f_{\theta_i})$, where α is the learning rate that controls the magnitude of each update, and i represent the $i - th$ iteration. Hence, by following the negative (or positive) direction of such gradient, the function will minimize (or maximize) such desired objective.

We note that in the context of training a Deep Neural Network, the stochastic approximation of GD, namely Stochastic Gradient Descent (SGD) (Robbins, 2007) and its variants (e.g., Adam (Diederik P. Kingma, 2015)), is the standard de-facto. However, the choice of these algorithms (Zachary C. Lipton, 2018) is often uncertain as discussed in Chapter 7, where a different combination of our evolutionary gradient-free component and the gradient-based algorithm leads SGD and Adam to quite different performances.

2.2 Reinforcement Learning

Starting from the general definition of Reinforcement Learning in Chapter 1, where an agent learns by trials and errors in its environment given a reward signal, we now provide notations and theoretical fundamentals of this learning paradigm, summarized in Figure 2.3.

A RL agent performs a sequential decision-making task in its environment to collect data and learn the desired policy. In particular, at every step t, the agent observes the current state of the environment s_t, deciding and executing an action a_t. This mapping between observations and probabilities of performing a certain action is called *policy* (denoted with π).[1] A time step later, the agent receives a so-called *reward* signal r_{t+1} as a consequence of its action, which indicates the quality of the interaction. Hence, rewards are the main responsible for driving the learning process (**?**). This continuous agent-environment interaction leads to a (so-called) *trajectory* $\tau = (s_0, a_0, r_1, s_1, a_1, \dots)$.

The environment thus changes following its dynamics and due to the agent's policy, or to the presence of other (unobserved) agents' policies (which typically cause non-stationarity issues in Multi-Agent problems).

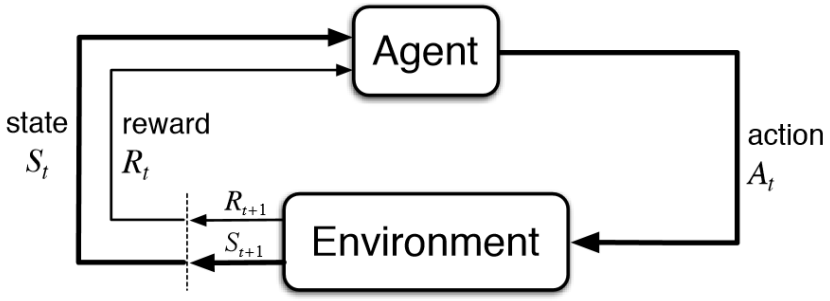

FIGURE 2.3: The typical agent–environment interaction in Reinforcement Learning. Courtesy of Sutton and Barto (2018).

[1] Refer to Sutton and Barto (2018) for a more comprehensive definition.

Formally, a Markov Decision Process (MDP) (Bellman, 1957) is the classical mathematical framework to shape this kind of sequential decision-making problems. In more detail, a MDP is represented as a 5-tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$, where:

- $\mathcal{S}$ is the set of possible states in the environment;
- $\mathcal{A}$ is the set of possible actions of the agent;
- $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function that returns the a scalar (i.e., immediate reward) after transitioning from state $s \in \mathcal{S}$ with action $a \in \mathcal{A}$.
- $P\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability to the next state $s' \in \mathcal{S}$ given the previous state $s \in \mathcal{S}$ and performed action $a \in \mathcal{A}$.
- $\gamma \in [0, 1)$ is a discount factor used to trade-off the importance between immediate and future rewards.

In the literature, there are different formalizations for a Markov Decision Process, based on the nature of the task. For example, Chapter 8 considers a Multi-Agent (Deep) Reinforcement Learning settings, where each agent receives only partial observations and acts in a decentralized decision-making paradigm. As such, the decentralized partial-observable version of a MDP, namely a Decentralized Partially Observable Markov Decision Process (POMDP) (Oliehoek and Amato, 2016), represents a natural way of describing this class of problems.

Finally, the goal of a Reinforcement Learning agent is to maximize the cumulative reward over many interactions, namely the *return* (Sutton and Barto, 2018). In the simplest case, given the sequence of the agent's rewards in a trajectory τ, we define the return G_τ as a naive sum over the rewards:

$$G_\tau = r_{t+1} + r_{t+2} + \cdots + r_T \tag{2.1}$$

However, this notation applies only to episodic tasks, where the concept of final time step T causes a reset of the environment to a specific initial state and conditions. We refer to Chapter 3 for practical examples of the design of an environment and its components.

In contrast, a typical real-world scenario does not naturally break into episodes, and the return (2.1) could be infinite. For this reason, we introduce the *discount factor* $\gamma \in [0, 1)$ and the goal becomes to maximize the expected discounted return, defined as:

$$G_\tau = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \tag{2.2}$$

Discounting is thus a crucial component as if $\gamma < 1$, the return's infinite sum has value as long as the reward sequence is bounded. In more detail, if $\gamma = 0$, the agent only cares about maximizing immediate rewards, being "myopic". Hence, the agent's objective is to learn how to act to maximize the immediate reward and maximize (2.2) just by maximizing each immediate reward separately. However, maximizing the only immediate reward can reduce access to future rewards so that the return may be (in general) reduced. Conversely, as γ approaches 1, future rewards are more important, and the agent becomes more "farsighted".

2.2.1 Objective

So far, we generally discussed the agent's goal of maximizing its expected return. In the following, we formally detail this optimization problem.[2] First, we express the expected return over trajectories as:

$$J_\pi = \mathop{\mathbb{E}}_{\tau\sim\pi}[G_\tau] \tag{2.3}$$

where $\tau \sim \pi$ indicates that the trajectory is distributed according to agent's policy. Our goal is to search for an optimal policy π_* that maximizes J_π (2.3). Formally:

$$\pi_* = \arg\max_{\pi\in\Pi} J_\pi \tag{2.4}$$

where Π represents the space of admissible policies. Hence, given a policy π and a state s, we can measure the agent's objective (i.e., the expected return) by using value functions. There exists different value functions in the literature, but state-value $V_\pi(s)$ and action-value functions $Q_\pi(s,a)$ are the main responsible for quantifying how good it is to be in a particular state ($V_\pi(s)$) or perform the given action in that state ($Q_\pi(s,a)$) and then following the policy. Formally:

$$\begin{aligned} V_\pi(s) &= \mathop{\mathbb{E}}_{\tau\sim\pi}[G_\tau|s_0 = s] = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)|s_0 = s\right] \\ Q_\pi(s,a) &= \mathop{\mathbb{E}}_{\tau\sim\pi}[G_\tau|s_0 = s, a_0 = a] = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)|s_0 = s, a_0 = a\right] \end{aligned} \tag{2.5}$$

where the value of a terminal state, in the case of episodic tasks, is always zero.

Value functions thus define a partial ordering over the policies. As such, we can define a policy π to be better than (or equal) another policy π' whether its expected return is greater than (or equal) to that of π' $\forall s \in \mathcal{S}$; formally:

$$\pi \geq \pi' \iff V_\pi(s) \geq V_{\pi'}(s)\ \forall s \in \mathcal{S} \tag{2.6}$$

Starting from Equation 2.4, we remark that the optimal policy may not be unique. When this applies, optimal policies share the same value function(s), which are naively referred to as optimal state-value or action-value functions. Formally:[3]

$$V_{\pi_*}(s) = \max_{\pi\in\Pi} V_\pi(s)\ \forall s \in \mathcal{S} \tag{2.7}$$

$$Q_{\pi_*}(s,a) = \max_{\pi\in\Pi} Q_\pi(s,a)\ \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{2.8}$$

Intuitively, state and action-value functions are deeply interconnected as the latter represents the value of starting in a particular state s and performing an action a, and then following a policy π. We can thus formalize the so-called *advantage* function

[2] Note that our concepts and notations follow the seminal work of Sutton and Barto (2018).
[3] We may omit parameters s, a for notation simplicity.

$A_\pi(s,a)$ that describe the advantage (or disadvantage) of taking action a over sampling an action from π (Wang et al., 2016). Formally:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{2.9}$$

State-of-the-art Deep Reinforcement Learning algorithms employ the advantage function to decompose action values (Wang et al., 2016; Hessel et al., 2018). We exploit the insights of the advantage function in Chapter 8 to model our multi-agent network architecture.

2.2.2 Algorithms

In this section, we provide a brief overview of the taxonomy of Reinforcement Learning algorithms. Note that this is not an exhaustive taxonomy. We only discuss the main concepts used throughout the thesis.

Value-based and Policy-gradient Methods

In the context of RL, we identify two distinct (but complementary) families of approaches, which we briefly summarize in the following:

- Value-based: this class of algorithms aims to build a value function used to define a policy (e.g., Q-learning (Watkins and Dayan, 1992)).
- Policy-gradient: these methods directly optimize the task objective, such as the expected return, by directly finding a policy using variants of SGD over the policy parameters (e.g., REINFORCE (Sutton et al., 2000)).

Moreover, a policy-gradient algorithm typically requires an estimate of a value function for the current policy. To this end, an actor-critic architecture consists of an actor that models the policy and a critic that estimates a value function (Konda and Tsitsiklis, 2000).

Evolutionary Algorithms

Policy-gradient solutions belong to a broader class of policy-based methods that also includes Evolutionary Algorithms (Fogel, 2006). EAs and in particular Genetic Algorithms are a black-box optimization process inspired by Charles Darwin's theory of natural evolution. A GA reflects the process of natural selection, depicted in Figure 2.4 where the fittest individuals are selected for reproduction in order to produce better offspring of the next generation.

In more detail, a Genetic Algorithm (Montana and Davis, 1989) (summarized in Algorithm 1) evolves an initial population $\mathcal{P}$ of n individuals, each one represented by a set of parameters θ (or genome). Each θ_i $(i = \{0,..,n-1\})$ is then evaluated to produce a per-individual fitness $\mathcal{P}$-fitness$_{\theta_i}$, used by a selection operator to choose the best genome.[4] The idea of the selection phase is thus to select the fittest individuals to pass their behaviors to the next generation. After the selection, crossover and mutation operators generate a diversified set of individuals to form the new population. Such operators ensure to maintain diversity and prevent premature convergence. The

[4] We denote the fitness of the entire population as $\mathcal{P}$-fitness.

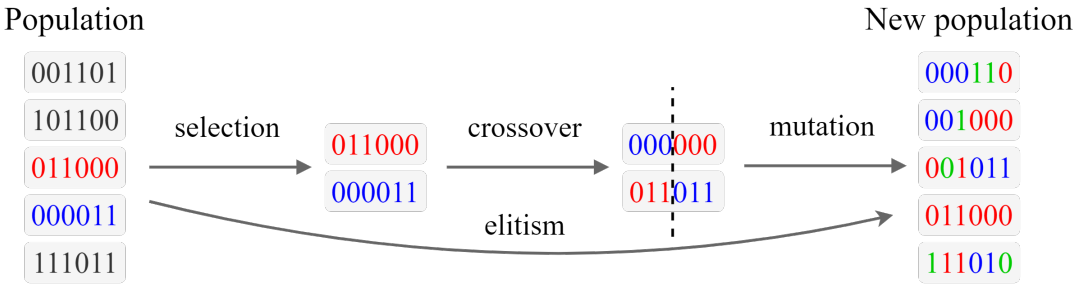

FIGURE 2.4: The typical flow of a Genetic Algorithm (Montana and Davis, 1989).

entire process continues until the population has converged (does not produce offspring significantly different from the previous generation, or we matched the desired performance).

Algorithm 1 Genetic Algorithm

1: $\mathcal{P} \leftarrow$ Initialize n individuals with random parameters θ_i $(i = \{0, \dots, n-1\})$
2: **while** terminal condition is not met **do**
3: $\quad \mathcal{P}$-fitness $\leftarrow$ Evaluate population $\mathcal{P}$
4: $\quad \mathcal{P}_* \leftarrow$ Select the best individuals according to $\mathcal{P}$-fitness
5: $\quad \mathcal{P} \leftarrow$ Re-initialize the n individuals with evolutionary operators on $\mathcal{P}_*$
6: **end while**

This class of algorithms has been recently employed as a promising gradient-free optimization alternative to Deep Reinforcement Learning. The redundancy of these population-based approaches has the advantages of enabling diverse exploration and improving robustness, leading to a more stable convergence. In particular, the same naive Genetic Algorithm previously described has shown competitive results compared to the more computationally complex gradient-based Deep Q-Network (Such et al., 2017) and also presents the advantage of requiring a significantly lower computational cost. However, these gradient-free approaches struggle to solve high-dimensional problems with poor generalization skills and are significantly less sample efficient than gradient-based methods.

Off-policy and On-policy Learning

In the literature, Reinforcement Learning algorithms are further categorized according to their use of the policies (Sutton and Barto, 2018):

- Off-policy: this class of methods evaluates or improves a policy different from that used to generate the data, namely a behavior policy.
- On-policy: in contrast, these solutions evaluate (or improve) the same policy used to make decisions.

Consequently, an off-policy setup makes it straightforward to learn from trajectories that are not necessarily obtained under the current policy. Hence, an experience replay allows re-using samples of the different behavior policies. On the contrary, on-policy methods usually introduce a bias when used with trajectories that are not obtained under the current policy (unless a correction mechanism, such as importance

sampling, is used). Off-policy methods are thus more sample efficient as they use (and re-use) any experience.

2.3 Deep Reinforcement Learning

The field of Deep Reinforcement Learning combines Reinforcement Learning and the impressive performance of Deep Neural Networks, enabling to approximate decision-making tasks characterized by high-dimensional state (or action) spaces.

The typical flow in the training of a Deep Reinforcement Learning model remains the same of Figure 2.3, where an agent interacts in an environment to collect new experiences (samples), that are used to learn either policies or value functions in an on-policy or off-policy fashion. The insights for the actual learning process are the same described in Section 2.1.

2.3.1 Deep Q-Network

One of the first Deep Reinforcement Learning algorithms is the off-policy Deep Q-Network (DQN) (Mnih et al., 2013b). This represents the ideal example for explaining the ideas behind DRL because it is the natural extension of classic tabular Q-learning (Watkins and Dayan, 1992) with Deep Neural Networks. The idea is to expand the tabular Q-Learning algorithm to the Deep Reinforcement Learning setup by replacing the regular Q-table with a Deep Neural Network because high-dimensional state (or action) spaces do not allow to store such table (or do it efficiently). Hence, rather than mapping a state-action pair to action values, a neural network maps input states to Q-values, denoted with $Q_\theta(s, a)$.

The original DQN implementation is highlighted in Algorithm 2.[5]

Algorithm 2 Deep Q-Network

Initialize a replay buffer B to store samples
Initialize the network's parameters θ with random weights
for epoch $= 1$ **to** ∞ **do**
  for t $= 1$ **to** T **do**
    With probability ϵ select a random action a_t, otherwise $a_t = \max_a Q_\theta(s_{t+1}, a)$
    Execute a_t in the environment, observing reward r_t and next state s_{t+1}
    Store the sample (s_t, a_t, r_t, s_{t+1}) in B
    Randomly sample a minibatch b of samples (s_k, a_k, r_k, s'_k) from B
    Set $y_k = \begin{cases} r_k, & \text{for terminal } s'_k \\ r_k + \gamma \max_a Q_\theta(s'_k, a), & \text{for non-terminal } s'_k \end{cases}$
    Perform a gradient descent step on $(y_k - Q_\theta(s_k, a_k))^2$
  end for
end for

As in vanilla Q-Learning, the agent needs to update the model (i.e., the DNN) parameters in order to maximize the expected cumulative reward. To this end, the DQN algorithm approximates the following form of Bellman's equation:

[5] Note that different implementations of *vanilla* DQN typically use a target network to stabilize the learning process. However, for notation simplicity, here we discuss the implementation that only considers the replay buffer.

$$Q_\theta(s_t, a_t) = (1 - \alpha)Q_\theta(s_t, a_t) + \alpha(r_t + \gamma \max_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a)) \tag{2.10}$$

which tells the agent how to update the current perceived value (at time step t) with the estimated optimal future reward, which assumes that the agent takes the best current known action:

$$a_t = \arg\max_{a \in \mathcal{A}} Q_\theta(s_t, a) \tag{2.11}$$

Hence, following the Deep Learning setup, we define the loss function to update the weights in order to approximate $Q_\theta(s, a) \forall s \in \mathcal{S}, a \in \mathcal{A}$ following Equation 2.10. In more detail, Deep Q-Network minimizes the loss function defined as the squared difference between the target $y = r + \gamma \max_a Q_\theta(s', a)$ (which is equal to the only reward in the case of a terminal state) and the estimated value $\hat{y} = Q_\theta(s, a)$. In practice, such loss is averaged over a given a batch of samples b, where each sample is a tuple (s, a, r, s'). Formally:

$$L(f_\theta) = \frac{1}{|b|} \sum_{k=0}^{|b|} \left[\Big((r_k + \gamma \max_a Q_\theta(s'_k, a)) - Q_\theta(s_k, a_k) \Big)^2 \right] \tag{2.12}$$

where s'_k denotes the next state observed in the environment after performing action a_k in state s_k. This notation is equivalent to s_t and s_{t+1} for the state at time step t and its next state, but refers to the k-th sample of the batch.

We finally note that several optimizations have been developed over the years to improve the naive Deep Q-Network algorithm. Namely, Double DQN (van Hasselt et al., 2016), Dueling DQN (Wang et al., 2016), Distributed DQN (Bellemare et al., 2017), Noisy DQN (Fortunato et al., 2017). We discuss these enhancements and their combination (i.e., the Rainbow algorithm (Hessel et al., 2018)) in more detail once they are used in the manuscript.

2.4 Formal Verification

Formal verification for DNN involves checking whether desired input-output relationships (properties) hold (Liu et al., 2019). For example, it is possible to examine the neighborhood of a given input x_0, to find the maximum possible disturbance ρ_0 that satisfy the following assertion:

$$\mathbf{x} \in \mathcal{X} \Rightarrow \mathbf{y} = f_\theta(\mathbf{x}) \in \mathcal{Y} \tag{2.13}$$

where $\mathcal{X} = \{\mathbf{x} : \|\mathbf{x} - x_0\|_2 \leq \rho_0\}$ and $\mathcal{Y}$ is a feasible output set. This formalization, however, encodes the input space of the properties as a hyperrectangle, limiting the application of Equation 2.13 to general scenarios. This has been addressed as follows to represent different geometries (e.g., polytopes):

$$\text{If } x_0 \in [l_0, u_0] \wedge \ldots \wedge x_n \in [l_n, u_n] \Rightarrow y \in [l_y, u_y] \tag{2.14}$$

where $x_i \in \mathbf{x}$ $(i = \{0, \ldots, n\})$ are the inputs of the DNN, and $y \in \mathbf{y}$ is an output. We consider reachability methods (Wang et al., 2018b,c; Weng et al., 2018) to verify properties in the form of Equation 2.14. This class of approaches computes an over-approximation of the exact output reachable set propagating the domain $\mathcal{X}$ through the network. In detail, the lower and upper bound propagation of each input (i.e., $I_i = [l_i, u_i]$) is approximated first by computing the *pre-activation* bounds with the following linear mapping, which is standard in the literature (Liu et al., 2019):

$$\begin{aligned} l_{new} &= \max(\theta, 0) * l + \min(\theta, 0) * u \\ u_{new} &= \max(\theta, 0) * u + \min(\theta, 0) * l \end{aligned} \tag{2.15}$$

then, the upper and lower bound of the activation are computed as follows by propagating these values through the activation function, which assume having monotonic activation function σ applied to an interval $I = [l, u]$:[6]

$$\sigma(I) = \begin{cases} [\sigma(l), \sigma(u)] & \textit{if monotonically increasing} \\ [\sigma(u), \sigma(l)] & \textit{if monotonically decreasing} \end{cases} \tag{2.16}$$

Equations 2.15, 2.16 are applied layer-by-layer and node-wise to compute the output reachable set $\Gamma(\mathcal{X}, f_\theta) := \{\mathbf{y} : \mathbf{y} = f_\theta(\mathbf{x}),\ \forall \mathbf{x} \in \mathcal{X}\}$. Hence, a property in the form of Equation 2.13 (or more generally Equation 2.14) is considered satisfied if it belongs to the output set, i.e., $\Gamma(\mathcal{X}, f_\theta) \subseteq \mathcal{Y}$.

In particular, the following chapters employ Neurify (Wang et al., 2018b) and ProVe (Corsi et al., 2021) as state-of-the-art reachability-based verification methods. We note that this thesis considers Formal Verification tools as a practical way to evaluate decision-making behaviors of DNNs. Hence, a detailed discussion of the various verification approaches and their methodologies is out of scope for this manuscript.

2.5 Enhancing Exploration and Safety

Chapter 1 discusses the issues of gradient-based approaches related to exploration and safety, highlighting the contribution that EAs and value decomposition could give to the field of DRL. After Part I, where we discuss the pros and cons of different DRL algorithms and further optimization, the benefits of combining EAs and gradient-based DRL in the contexts of exploration and safety will cover most of our work. In particular, both Part II and Part III propose a variety of optimizations based on combining the two families of algorithms to enhance exploration towards policy regions that achieve higher rewards and foster safer behaviors.

2.5.1 Deep Reinforcement Learning Exploration

The problem of exploration in RL is crucial as the absence of exhaustive information is one of the issues behind the requirement of a considerable number of trials to achieve good performance. Moreover, the lack of a diverse exploration in high-dimensional spaces causes convergence to local optima and hinders the agent from discovering

[6] This is standard in verification literature (Zhang et al., 2018), as the most common activation, e.g., ReLU, Tanh, are monotonic.

novel behaviors that could lead to higher returns. The brittle convergence guarantee of DRL algorithms, which is strongly related to the choice of initialization and hyper-parameters, is also the main issue that limits a broader adoption of such learning techniques to real-world scenarios (Henderson et al., 2018). Hence, devising robust learning approaches while improving sample efficiency is the first challenge we address enhancing existing solutions with EA (Chapters 6, 7).

Moreover, multi-agent domains typically require efficient exploration to cope with the agents' high-dimensional state and action spaces while favoring cooperative (or competitive) behaviors. To this end, value decomposition algorithms have been recently proposed as an efficient way to address these issues. However, the structural constraints used to guarantee the action value decomposition induce the agents to limit their exploration abilities, which can hinder performance (Chapter 8).

2.5.2 Deep Reinforcement Learning Safety

However, exhaustive exploration is a typical problem in practical applications such as the ones we consider in Chapter 3 because it is generally not possible to guarantee or quantify safe behaviors. Intuitively, this lack of safety represents a crucial issue when working with high-cost hardware or in a human-populated scenario.

Besides naive solutions that aim at fostering safety with simple penalties in the reward, the use of Safety Critics (Thananjeyan et al., 2020; Bharadhwaj et al., 2021; Thananjeyan et al., 2021) represents a recent research direction that relies on estimating the probability of incurring into unsafe states, given a state-action pair. However, such approaches could potentially return misleading information for policy improvement when the Safety Critic estimator is not trained on a wide variety of unsafe behaviors or rely on data collected by human supervision, which may be challenging to collect.

In contrast, a different direction formalizes the problem of Safe DRL through a Constrained Markov Decision Process (CMDP), where an auxiliary cost function, similar to the reward, denotes unsafe states. The objective of a constrained DRL agent is thus to maximize the long-term reward subject to constraints on the costs. However, constrained approaches have several drawbacks, such as the careful tuning of the threshold for the constraints, where high values mean that they are too permissive, or conversely, too restrictive (Garcıa and Fernández, 2015). Hence, constrained DRL is also not devoid of short-term fatal consequences as empirical evidence shows that they typically fail at satisfying the imposed constraints (Ray et al., 2019).

We further elaborate these Safe DRL directions and their formalization in Part III, where we propose a different perspective on the problem that uses EAs and Formal Verification to address their limitations.

Part I

Benchmarking Exploration

Chapter 3

Unity Simulation for Robotics

A key factor responsible for the significant advances in Deep Reinforcement Learning research and algorithm design (Mnih et al., 2013b; Lillicrap et al., 2016; Schulman et al., 2017; Fujimoto et al., 2018; Haarnoja et al., 2018) is the increasing presence of challenging and scalable simulation platforms (Juliani et al., 2018; Todorov et al., 2012; Machado et al., 2018). However, reproducing the improvements offered by such novel methods is seldom straightforward, and the non-determinism in benchmark environments is one of the core factors that hinder reproducibility.

This chapter discusses the benefits of non-standard simulation software that offers high-performing physics engines to ensure a realistic and informative simulation for robotics. In this direction, we present three robotic environments that will be used through the thesis to discuss our contributions and the current limitations of DRL algorithms.

3.1 Introduction

State-of-the-art results in Deep Reinforcement Learning have been achieved mainly using simulation and transferring the policy on real platforms (Juliani et al., 2018; OpenAI et al., 2019; Zhao et al., 2020; Ding et al., 2020). Although the general discussion around simulation appears underdeveloped over the algorithmic counterpart, simulation environments are the key component that allows the DRL community to test and evaluate novel ideas. Mujoco (Todorov et al., 2012), for example, offers physics-based simulation for a variety of tasks that range from video games to complex locomotion and has been adopted to benchmark a variety of algorithms (Lillicrap et al., 2016; Fujimoto et al., 2018; Schulman et al., 2017; Haarnoja et al., 2018). Hence, the quality of environments is of critical importance.

However, benchmarking environments are constrained by the limitations of the simulators because they can not always provide meaningful challenges to novel learning systems. It is also not obvious which properties of an environment make it a useful benchmark. In this direction, recent game engines offer a robust yet flexible and easy-to-use platform for unlimited environment creation, enabling the simulation of visually realistic worlds with complex physics and interactions between multiple agents. Along this line, Unity has recently released a toolkit that enables rapid prototyping and development of simulation environments (Juliani et al., 2018).[1]

[1] www.unity3d.com/company

For these reasons, we investigate Unity as an alternative way to develop high-quality environments, focusing our attention on the robotic domain, where the uncertainty of the platforms and the environment dynamics represent a complex challenge for modern for Deep Reinforcement Learning algorithms. To this end, our contributions are as follows:

- We present three robotic environments:
  1. Trajectory generation for the commercial manipulator Panda.[2]
  2. Mapless navigation for the indoor mobile robot TurtleBot3.[3]
  3. Navigation for the outdoor aquatic drone of the INTCATCH2020 European project.[4]
- We show that it is possible to export a model trained in our Unity environments, to conventional Robot Operating System simulators (i.e., Gazebo and RViz) and the real robots, without additional training.

We begin our discussion by analyzing a set of desired properties important for designing a robotic simulation environment. Then, we introduce popular simulation platforms in DRL research and highlight their limitations.

3.2 Robotic Environments

We rely on the taxonomy and definitions of Juliani et al. (2018) to provide a general understanding on the main components that are crucial do design a robotic environment. The term simulation environment refers to the scenario in which an agent (e.g., a robot) acts, influencing its surroundings and being influenced by its dynamics. In robotics, the essential components that form the environmental complexity are mostly related to sensors and physics, which we briefly discuss in the following.

Sensors: process large amounts of data coming from various sources (e.g., auditory, visual) and theyr non-linear approximatio nature is the main responsible behind the recent advances in the field of Deep Learning (Goodfellow et al., 2016). In particular, visual and depth information are vital for many real-world decision-making problems that range from self-driving cars to robotics (Zhu et al., 2017). It is thus crucial that modern benchmark environments accurately simulate realistic sensors based on the manufacturer's specifications.

Physics: The uncertainty and the dynamics of an environment represent the other key factor that drove many advances in the field of Deep Reinforcement Learning. Complex interactions between the robot and the environment are strongly dependent on the need for the simulation scenario to compute and replicate the real-world dynamics. Having physically realistic simulation is moreover of fundamental importance in order to naturally address the typical sim-to-real problem that arises when transferring a simulated trained policy to the actual scenario and platform (Zhao et al., 2020; Ding et al., 2020).

Furthermore, researchers must consider practical constraints when designing environments for DRL experimentation. For example, simulated environments must be

[2] www.franka.de
[3] www.turtlebot.com
[4] www.intcatch.eu

flexible and allow a variety of setups to control the simulation and ensure a correct generalization of a trained model.

3.2.1 Simulation Environments

A wide variety of simulators have been developed over the years either for domain-specific tasks, as a collection of different environments packaged together to form a benchmark, or as general platforms to create environments with arbitrary components (e.g., sensors and physical interactions). Follows a brief overview of the most famous simulation environments responsible for key development in Deep Reinforcement Learning:

- Games and Video Games: typically consists of high-dimensional environments with fixed rules and dynamics that return partial (or global) information to the agent. They can model single or multi-agent scenarios but are usually black-boxes from the agent's perspective as they can not be customized to foster generalization. Examples of these include ALE (Machado et al., 2018), StarCraft II (Vinyals et al., 2019), Dota 2 (OpenAI et al., 2019) as well as classic board games such as Chess, Shogi, and Go (Silver et al., 2018b).
- Toy examples: are mostly related to classical control tasks that involve simple dynamics and limited state and action spaces. Typical tasks include CartPole or MountainCar (Sutton and Barto, 2018) that are representative examples of the suite of Gym environments (Brockman et al., 2016).
- Physics interaction: represents tasks where the physical component introduces significant challenges for the learning algorithms and typically requires continuous control. Key examples of this are the Mujoco locomotion tasks (Todorov et al., 2012; Lillicrap et al., 2016) or the DeepMind Lab (Beattie et al., 2016) that are widely considered as a benchmark for Deep Reinforcement Learning algorithms. This class of simulation environments is perhaps key to advancements in both the Safe DRL and robotic fields as it allows to model realistic scenarios where the robots can acquire precise sensor information of the surroundings (Juliani et al., 2018; Ray et al., 2019).

In this direction, Unity provides a general-purpose game engine that supports various platforms (e.g., smartphones, computers, virtual reality) with an underlying accurate physics simulation. For this reason, we believe this is an ideal candidate simulation platform for Deep Reinforcement Learning research. The engine's flexibility enables the creation of tasks across the previously mentioned classes, ranging from simple 2D grid world problems to complex 3D strategy games or multi-agent competitive games.

In the following, we present our three robotics environments that employ the Unity game engine to design a physically realistic simulation. Crucially, these environments are also compatible with the widely adopted Gym interface (Brockman et al., 2016) to promote their applicability. To further motivate the importance of this general-purpose simulation software, we show that its accurate simulation of sensors and physics allows transferring the trained models on conventional robotics simulators (e.g., Gazebo) and the real platforms without any additional training.

3.3 Trajectory Generation for Panda

Target Platform: We build our environment on the seven degrees of freedom Panda manipulator, depicted in Figure 3.1, and rely on the Panda's Denavit-Hartenberg (D-H) parameters to verify whether the performed trajectory in our environment reflects the kinematics of the robot.

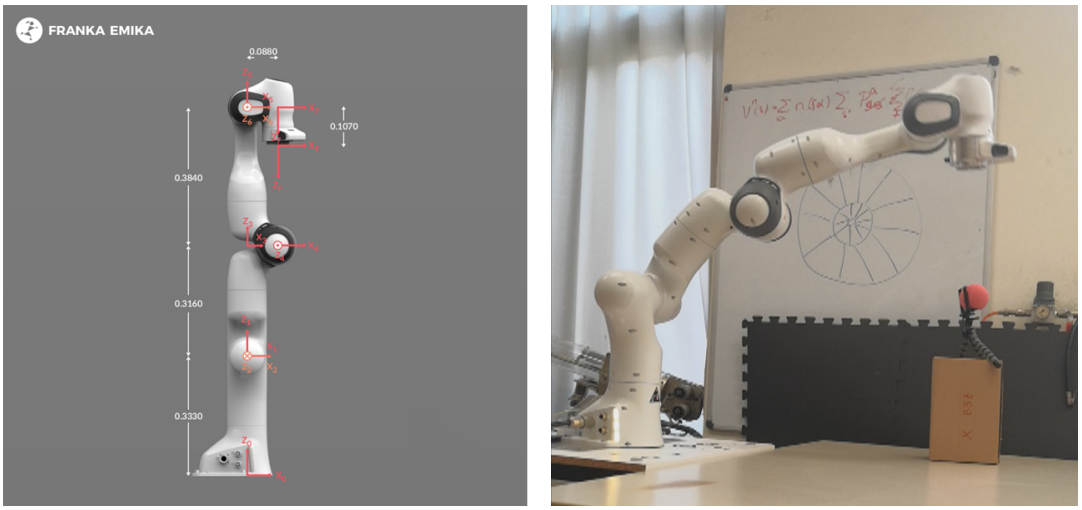

FIGURE 3.1: The kinematic chain of the Panda (image from frankaemika.github.io) (left). The real Panda manipulator used in our experiments (right).

Task Encoding: Given the manufacturer specifications, we encode the manipulator work-space considering a range of ± 120 degree for the first six joints, excluding the last one in the wrist, which is responsible for the grasping phase that we do not consider in the trajectory generation task. The observation space includes nine values; one for each considered joint and the last three for the target, which coordinates are represented as a triple $(x, y, z) \in \mathbb{R}^3$ sampled uniformly in the whole work-space of the robot. These values are normalized in range $[0, 1]$. We encoded the output space of the environment to represent an action that moves each joint by 2 degrees. However, we note that given the generalization skills of Deep Reinforcement Learning agents, it is possible to generate a trajectory using different values. Formally:

$$\mathcal{S} = [j_{0:5}, g_{x,y,z}] \qquad \mathcal{A} = [v_{0:5}] \tag{3.1}$$

where $j_{0:5}$ are the current position of the six joints, $g_{x,y,z}$ are the goal's coordinates, and $v_{0:5}$ maps each action to a joint movement (i.e., if $v_i > 0$ then j_i moves 2 degrees clockwise, else if $v_i < 0$ then j_i moves 2 degrees anti-clockwise, $\forall i = \{0, \ldots, 5\}$). A decisive factor that allows us to use a discrete action space is the functions provided by the Franka APIs that allow us to interpolate the discrete joint steps, to create a smooth trajectory with a desired joint velocity. Discrete actions are further motivated by the fact that recent work (Tavakoli et al., 2018; Zahavy et al., 2019; Dulac-Arnold et al., 2015) shows that discrete action space algorithms are a competitive alternative over continuous action spaces in control tasks.

Reward Function: A natural choice for a dense reward is the distance from the end-effector to the target. However, we noticed that there might be configurations close to the goal but can not reach the desired final configuration. Giving a high reward to such configurations would provide false positive information, ruining the training phase. To avoid this, the agent receives a sparse reward in case of reaching the target, and for timeouts:

$$r_t = \begin{cases} -1 & \text{timeout is reached} \\ 0 & \text{intermediate step} \\ 1 & \text{end-effector has reached the target} \end{cases} \tag{3.2}$$

Moreover, we remark that the different parameters of the environment (e.g., joint rotations, reward) are easily configurable to match the desired specifications.

Unity Simulation: Our simulated Unity environment, depicted in Figure 3.2, is realized using primitive 3D objects of the engine (e.g., spheres, cubes) for the plane and the targets. We use the manufacturer model of the Panda for our robotic agent, simulating the joint connections using Unity's native joints system. Unity's polar coordinates return the current joints and target positions that form the observation space with respect to the modeled environment. Finally, default physics parameters are considered to simulate gravity, frictions, and collisions.

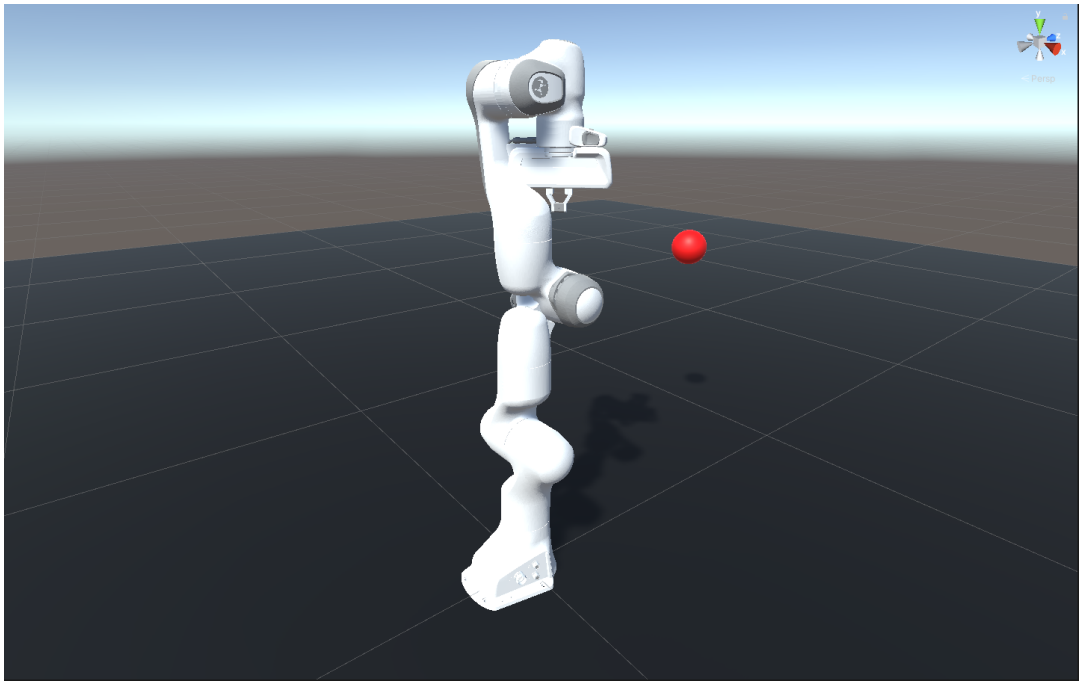

FIGURE 3.2: Overview of the Unity environment for the Panda.

3.4 Mapless Navigation

Similarly to manipulators, mobile robots are widely considered in Deep Reinforcement Learning literature (Tai et al., 2017; Zhang et al., 2017; Kretzschmar et al., 2016) due to the variety of practical applications. In particular, we focus on the mapless

navigation problem, a well-known benchmark in recent literature (Wahid et al., 2019; Chiang et al., 2019), which aims at navigating the robot towards random targets using local observation and the goal's position, without a map of the surrounding environment or obstacles. We present two navigation-based scenarios:

1. An indoor one with both continuous and discrete action spaces, characterized by static obstacles and walls.
2. A highly dynamical aquatic navigation scenario with both continuous and discrete action spaces to cope with the uncertainty of the operational environment.

3.4.1 Indoor Navigation for TurtleBot3

We first introduce the indoor navigation scenario with fixed obstacles, characterized by a discrete and continuous action space and a dense reward function.

Target Platform: We build our environment on the differential drive mobile robot TurtleBot3, depicted in Figure 3.3, which is a widely used platform in previous work focusing on robot navigation (Tai et al., 2017; Tai et al., 2016).

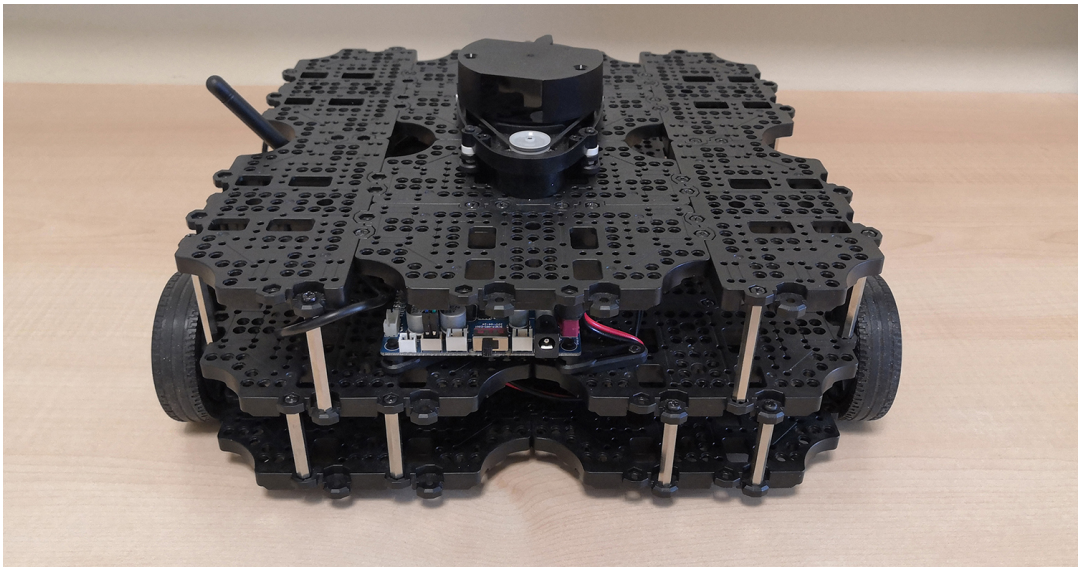

FIGURE 3.3: The real TurtleBot3 mobile robot used in our experiments.

Task Encoding: Given the specifications of the TurtleBot3, we consider an angular velocity of max 90 deg/s. The discrete action space encodes actions directly mapped into the angular and linear velocities of the robot (i.e., $[v_{ang_{0:\dots}}, v_{lin_{0:\dots}}]$ are the possible discrete velocities for the action space), while a continuous action space directly maps actions to the motor velocities (i.e., $[v_{ang}, v_{lin}]$). The decision-making frequency of the simulated robot is 20Hz. However, in the following evaluations, a Deep Reinforcement Learning trained agent computes on average ≈ 60 $actions/s$, providing a significant margin of improvement to increase the control frequency further and deal with a faster robot or dynamic obstacles. The laser sensor mounted on the robot is an LDS-01 and provides sparse scan values sampled between -90 and 90 degrees in a fixed angle distribution. The maximum update rate of this component, provided by the manufacturers, is 5Hz. The target goal coordinates are randomly chosen in the

environment's area and are guaranteed to be obstacle-free. Similar to the manipulator scenario, such coordinates are expressed as a triple $(x, y, z) \in \mathbb{R}^3$. Formally, the observation and action spaces of the task are:

$$\mathcal{S} = [d, h_g, lid_{0:\dots}] \qquad \mathcal{A} = [v_{ang_{0:\dots}}, v_{lin_{0:\dots}}] \wedge [v_{ang}, v_{lin}] \tag{3.3}$$

where d is the current distance of the robot from the target goal, h_g is its relative heading over the goal, and $lid_{0:\dots}$ are the sparse lidar values that can be configured according to the task. Moreover, all the other parameters that characterize the environment are configurable. Finally, to analyze the generalization skills of our models, the configuration of the obstacles is fixed and compact around the robot's initial position.

Reward Function: There are three conditions for the reward: two sparse values in case of reaching the target or crashing, which terminates an episode (resetting the robot to its starting position), and a dense part used during the travel:

$$r_t = \begin{cases} -1 & \text{if crashes or timeout} \\ 1 & \text{if reaches the target} \\ \mu(d_{t-1} - d_t) & \text{otherwise} \end{cases} \tag{3.4}$$

where $d_{t-1} - d_t$ indicates the distance between robot and goal between two consecutive time steps, and μ is a multiplicative factor. In the following experiments $\mu = 15$. However, we note that the tuning of μ causes different behaviors of the robot:

- $\mu = 15$ leads to straighter movements, and smoother trajectories as the reward for actions with zero angular velocity are more significant.
- $\mu = 10$ causes non-smooth paths because more unnecessary actions with non-zero angular velocity are selected.

Unity Simulation: We modeled different Unity environments for the training and testing of Deep Reinforcement Learning policies. Figure 3.4 shows two of them that, similar to the manipulator environment, are realized using primitive 3D objects of the Unity engine (e.g., spheres, cubes) for walls, obstacles, and the targets. For our robotic agent, we use the manufacturer model of the TurtleBot3, and the current scan values, robot position, and target positions that form the observation space are returned by Unity's polar coordinates with respect to the modeled environment.

3.4.2 Outdoor Navigation for Aquatic Drone

Robotic navigation, however, is not limited to indoor environments; instead, wild (outdoor) environments pose more significant challenges for learning algorithms (e.g., uneven terrain, unexpected obstacles). Hence, we introduce an aquatic navigation scenario characterized by a physically realistic water surface with dynamic waves and floating objects. Similar to indoor navigation, among the variety of applications for aquatic drones (Codd-Downey and Jenkin, 2017; Karapetyan et al., 2018), navigation is a crucial aspect to enable autonomous behaviors. For example, autonomous water quality monitoring represents an efficient alternative to the more traditional manual sampling (Castellini et al., 2020).

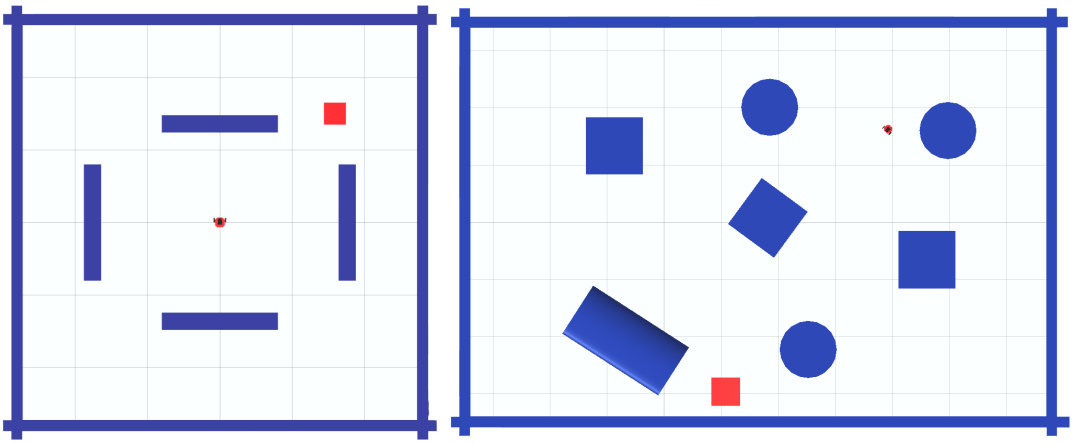

FIGURE 3.4: Two representative overviews of a training (left) and testing (right) environments for the TurtleBot3 navigation task.

Target Platform: We consider the drones of the EU-funded Horizon 2020 project INTCATCH as a robotic platform, depicted in Figure 3.5.[5] Since the platform is not standard, here are some specifications of the robot: the drone is a differential drive platform based on a hull equipped with two in-water propellers that can be deployed in shallow water, with a max velocity of $3m/s$. The onboard sensors (e.g., GPS, compass) provide the localization and orientation information, while a laser sensor returns the distances between the boat and obstacles.

FIGURE 3.5: The aquatic drones of the INTCATCH 2020 European project.

Task Encoding: Given the drone's specifications, the robot's decision-making frequency is set to $10Hz$, as aquatic navigation does not require high-frequency controls.

[5] www.intcatch.eu

We designed both discrete and continuous action space settings to provide a more comprehensive evaluation for this non-standard scenario. The discrete action space considers values that map to the power of the two motors to drive the boat (i.e., $[v_{0:...}]$ that ranges from left steering to forward only movement and right steering). In the continuous setup, two outputs control each motor velocity (i.e., $[v_{left}, v_{right}]$). Moreover, the observation space contains sparse laser scans sampled in the range $[-90, 90]$ degrees in a fixed angle distribution and the polar target coordinates, a similar setting considered in the previous environment. Formally:

$$\mathcal{S} = [d, h_g, lid_{0:...}] \qquad \mathcal{A} = [v_{0:...}] \wedge [v_{left}, v_{right}] \tag{3.5}$$

Reward Function: The reward function is equivalent to the indoor navigation one, where $r_t = \mu(d_{t-1} - d_t)$ is a dense value during the travel and computes the euclidean distance between the robot and the goal at two consecutive time steps. Two sparse values explain the cases of reaching the target ($r_t = 1$) or crashing ($r_t = -1$), which terminates an episode (resetting the drone to a starting location and spawning a new target).

Unity Simulation: The game engine allows us to reproduce the water behavior by triangulating a plane and displacing the generated vertices to simulate the desired wave condition. The plane triangulation can be adjusted to address the trade-off between a more fine-grained simulation of the waves and a higher computational demand imposed on the machine. We considered this, as other particle-based methods such as FleX are not suitable to simulate high-dimensional water surfaces.[6] Trivially, the amount of generated vertices depend on the hardware. Moreover, Unity integrated collision and force algorithms (e.g., gravity, friction, viscosity) are suitable to simulate the water surface, the aquatic drone, and their interactions.

As in the indoor scenario, we modeled different environments for the training and testing of Deep Reinforcement Learning policies, depicted in Figure 3.6. In more detail, obstacles are blue shapes, and the goal is a red sphere. To speed up the training process in such a complex dynamic environment, we consider Unity materials for the rendering pipeline (as they are less computational demanding). However, it is possible to adjust the scene characteristics (e.g., materials, lights, shadows) for a more realistic scene.

3.5 From Unity to the Real Robots

Finally, we show that it is possible to export a model trained in our Unity environment to the Robot Operating System and the real robot. To this end, we note that only the Panda and the TurtleBot3 have a native ROS interface. Hence we only consider these two for our transfer to ROS and the robots. In particular, we modified the source of observations for the trained policies as in the Robot Operating System testing, the Panda and the TurtleBot3 models retrieves information from RViz and Gazebo, respectively. The critical outcome of these validations is that the movement of the actual robots shows a very close correspondence with the original Unity simulation environment.

[6] www.developer.nvidia.com/flex

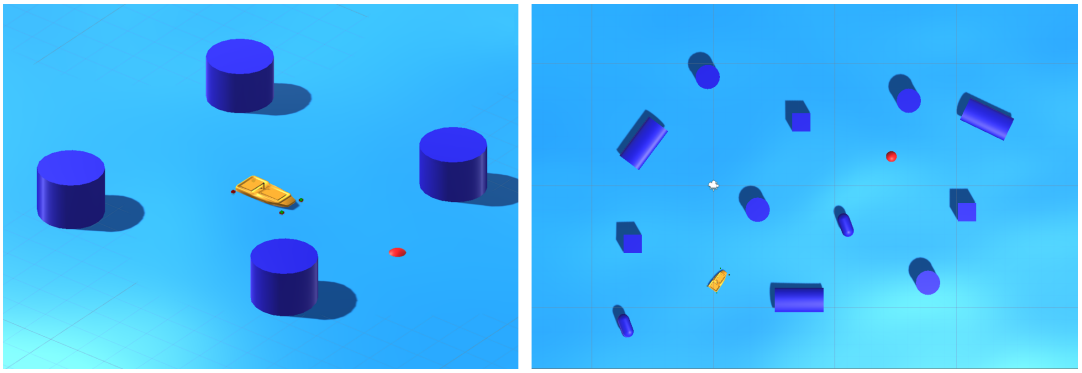

FIGURE 3.6: Two representative overviews of a training (left) and testing (right) environments for the aquatic navigation task.

Panda Transfer: with a correct localization of the goal in the Panda scenario, we do not encounter additional sources of noise that hinder the performance of our trained models.

Figure 3.7 shows an explanatory episode for a trained trajectory generation model in the two setups: the RViz simulator (top) and the real Panda (bottom).

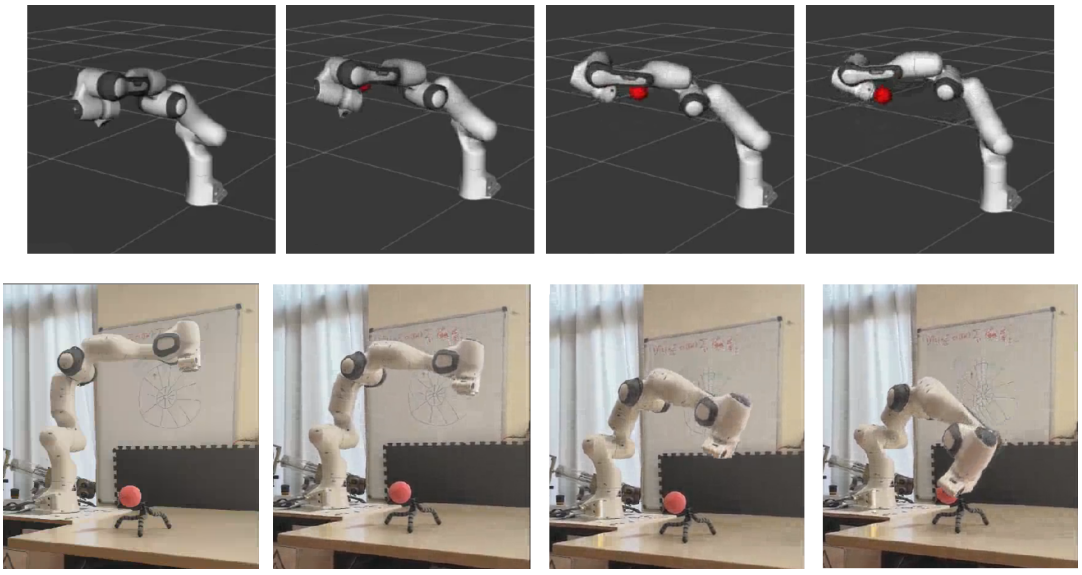

FIGURE 3.7: Overview of an explanatory trajectory of a trained model in the RViz visualizer (top), and the real Panda (bottom).

TurtleBot3 Transfer: To further motivate the use of Unity, Figure 3.8 shows the average difference between training a navigation policy in Unity and the same environment in Gazebo for the TurtleBot3. It is clear that the optimization offered by the game engine significantly speedup the training phases (i.e., ≈ 4 time faster) even among different standard implementations of various Deep Reinforcement Learning algorithms such as Double DQN (van Hasselt et al., 2016), Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016), and Proximal Policy Optimization (PPO) (Schulman et al., 2017).

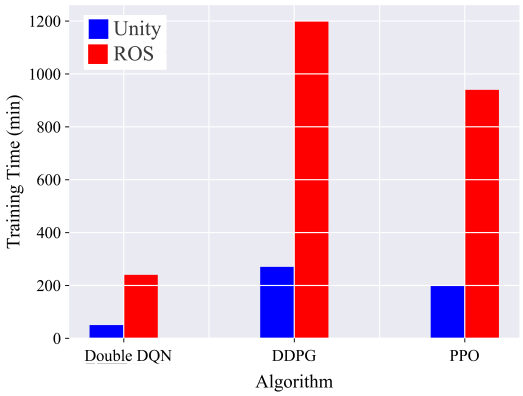

FIGURE 3.8: Average training time between different algorithms in the same Unity and Gazebo environment.

In the transfer to the robot, we use the manufacturer ROS package to retrieve the laser sensor values and Adaptive Monte Carlo Localization (AMCL) to localize the robot. However, given the update rate of the TurtleBot3 laser sensor (i.e., 5Hz), we note that the robot localization retrieved with AMCL is imprecise, and the traditional *movebase* motion planner, to which we compare, could fail without fine-tuning its parameters. In contrast, DRL trained models were able to generalize over this additional source of noise and have no problem during the navigation.

Figure 3.9 shows an explanatory trajectory for a trained navigation model in the three setups: the Unity environment to show the simulated laser scans (top), the Gazebo simulator (middle), and the real TurtleBot3 (bottom).

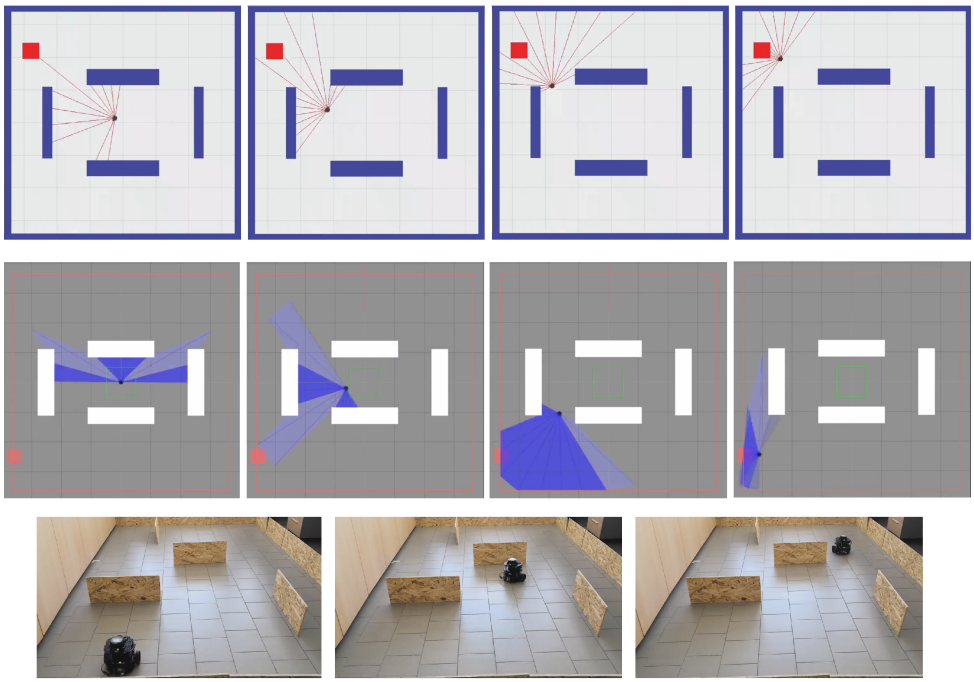

FIGURE 3.9: Overview of an explanatory trajectory of a trained model in the Unity environment (top), the Gazebo simulator (middle), and the real TurtleBot3 (bottom).

3.6 Discussion

In this chapter, we investigated Unity as a viable general-purpose framework to design fast and easy-to-change environments for Deep Reinforcement Learning, presenting three robotic environments that will be used through this thesis. Crucially, we showed that DRL policies trained in Unity work in standard simulation tools and on the real robots, without additional training.

Chapter 4

Benchmarking Sample Efficiency

The non-determinism in benchmark environments, along with the intrinsic variance of Deep Reinforcement Learning algorithms, make state-of-the-art DRL results tough to interpret and reproduce.

This chapter discusses the importance of choosing a particular algorithm, depending on task-related requirements. To this end, we use our robotic environments to benchmark existing Deep Reinforcement Learning techniques and present further optimization (e.g., asynchronous training, scaling discounts) to enhance their performance.

4.1 Introduction

Without significant metrics and clear standardization of experimental evaluations, it is difficult to determine whether the claimed improvements of novel algorithms are meaningful (Henderson et al., 2018; Colas et al., 2019). For example, the well-known suite of MuJoCo locomotion tasks is the de-facto benchmark for the majority of policy-gradient Deep Reinforcement Learning algorithms proposed over the years. However, Mania et al. (2018) recently argued that such environments are not as complex as previously thought, obtaining state-of-the-art performance using a simple random search of static linear policies.

In this direction, we note that since the release of the first continuous control policy-gradient DRL algorithm, DDPG (Lillicrap et al., 2016), the vast majority of works on autonomous robotics focused exclusively on these approaches (Tai et al., 2017; Xie et al., 2018; Gu et al., 2017; OpenAI et al., 2019), employing continuous action spaces. Such methods followed the idea that value-based DQN (Mnih et al., 2013b) can not deal with high-dimensional action spaces. However, discrete action space solutions (and generally value-based algorithms) typically result in shorter training time being more sample efficient. Hence, different methods are renewing the interest in using discrete action spaces, showing that value-based Deep Reinforcement Learning algorithms can also handle high-dimensional actions. In more detail, Tavakoli et al. (2018) proposes an adaptation of Dueling DQN (Wang et al., 2016) with Double DQN (van Hasselt et al., 2016) that achieves competitive results in locomotion benchmarks (Brockman et al., 2016; Todorov et al., 2012). More recently, de Wiele et al. (2020) designed a DQN-based algorithm to handle enormous discrete and continuous action spaces, with several other value-based methods that confirm these results (Zahavy et al., 2019; Dulac-Arnold et al., 2015).

We begin by presenting the central insights of the Deep Reinforcement Learning algorithms that we consider in our evaluation. Then, we present the experimental setup

for the indoor mapless navigation environment we consider here, discussing the pros and cons of different algorithms and further optimizations applied to these domains.

4.2 Preliminaries

We formalize the indoor mapless navigation as a Reinforcement Learning problem, defined over a Markov Decision Process, as described in recent DRL literature (Tai et al., 2017; Zhang et al., 2017). In more detail, at each time step $t = \{0, \dots, T\}$, the robot chooses and executes an action a_t according to the state s_t, observing the new state of the environment s_{t+1} and receiving a reward r_t. The goal is the same described in Section 2.2 that is to maximize the total discounted reward from step t onward, given by Equation 2.3.

For our evaluation, we use the value-based Double DQN (van Hasselt et al., 2016), the policy-gradient PPO (Schulman et al., 2017), and the actor-critic DDPG (Lillicrap et al., 2016), which we briefly describe in the following.

Double Deep Q-Networks The classic target Q-value in the DQN algorithm, i.e., $r_t + \gamma \max_a Q_\theta(s_{t+1}, a)$, involves taking the max over the next state values. Intuitively, suppose the estimation is incorrect (i.e., it is underestimated or overestimated), which is typical in the early stages of the training. In that case, a bias is introduced in the learning process. Since DQN involves learning estimates from estimates (i.e., *bootstrapping*), such underestimation (or overestimation) may lead to significant convergence issues. van Hasselt et al. (2016) illustrated this bias issue in the Atari environments, proposing to avoid the maximization bias by disentangling the updates from biased estimates. Hence, Double DQN uses two separate Q-value estimators, one for action selection parametrized by θ, and the other for the action evaluation parametrized by θ'. Formally, the target Q-value in the Double DQN algorithm becomes:

$$r_t + \gamma Q_\theta(s_{t+1}, \max_a Q_{\theta'}(s_t, a)) \tag{4.1}$$

The update of the action evaluation model, namely the target model, is typically performed by periodically copying the parameters of θ, or smoothing such transition through Polyak averaging (Polyak and Juditsky, 1992), which in practice show better performance (Lillicrap et al., 2016):

$$\theta' = \beta\theta + (1 - \beta)\theta' \tag{4.2}$$

where β is a hyper-parameter representing the interpolation factor in Polyak averaging for the target model.

Deep Deterministic Policy Gradient In contrast, DDPG is an actor-critic algorithm that concurrently learns a value function and a policy. It uses off-policy samples and the Double DQN insights to learn the Q-function (i.e., the critic, parametrized by θ_Q with target model θ'_Q as in Double DQN), which DPPG uses to learn the policy (i.e., the actor, parametrized by θ_π with target model θ'_π)

Deep Deterministic Policy Gradient is specifically designed for continuous action spaces and interleaves the learning of the two components. Due to the continuous

nature of the action space, we can not exhaustively evaluate such space (i.e., compute the maximum over actions as in Equation 2.11). To address this, DDPG set up a gradient-based learning rule for a deterministic policy $\pi(s)$, using the estimated value function.

In more detail, the critic is trained similarly to Double DQN, where the main difference is that the target is computed using the deterministic policy of the actor; formally:

$$r_t + \gamma Q_{\theta'_Q}(s_{t+1}, \pi_{\theta'_\pi}(s_{t+1})) \tag{4.3}$$

Instead, the actor is trained by maximizing the mean value given by the critic for the actions taken by the actor network. Hence, we update the actor to produce actions that achieve the maximum predicted value, for a given state. Formally, given a sample at time step t, we update the actor using the sampled policy gradient:

$$\nabla_{\pi(s_t)} Q_{\theta_Q}(s_t, \pi(s_t)) \nabla_{\theta_\pi} \pi_{\theta_\pi}(s_t) \tag{4.4}$$

And the target networks are updated using Polayk averaging:

$$\begin{aligned} \theta'_\pi &= \beta\theta_\pi + (1-\beta)\theta'_\pi \\ \theta'_Q &= \beta\theta_Q + (1-\beta)\theta'_Q \end{aligned} \tag{4.5}$$

Proximal Policy Optimization In a different direction, Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) introduced the concept of *trust region search strategy* to ensure that the updated policy does not differ too much from the current one, hence limiting the exploration only to nearby policy spaces. To this end, authors propose to use the *KL-Divergence* to measure the distance between the current policy $\pi_{\theta_{old}}$ and the new policy π_θ and ensure that such distance is always fewer than a given threshold δ. Therefore, the objective function aims at maximizing the discounted cumulative reward while ensuring the limited divergence between the two policies; formally:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \quad \text{s.t.} \quad D_{KL}(\pi_{\theta_{old}} \,||\, \pi_\theta) < \delta \tag{4.6}$$

However, TRPO is a computationally demanding algorithm that relies on second-order optimization with the additional overhead of a hard constraint.

Schulman et al. (2017) improved TRPO with Proximal Policy Optimization, a first-order optimization problem that measures how much the two policies differ by encoding the probability ratio of the new (updated) policy π_θ and the old policy at time step t (i.e., π_{θ_t}) as follows:

$$\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)} \tag{4.7}$$

Hence, given the advantage of performing a specific action a under policy π_{θ_t} at state s (i.e., $A_{\pi_{\theta_t}}(s,a)$) the clipped objective function, which is the best performing version of PPO according to the original work, is formalized as follows:

$$\mathbb{E}\left[\min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)} A_{\pi_{\theta_t}}(s,a), \text{clip}\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)}, 1-\epsilon_{clip}, 1+\epsilon_{clip}\right) A_{\pi_{\theta_t}}(s,a)\right)\right] \tag{4.8}$$

In the above equation, the ratio between the policies always clips in a small interval around 1, and such interval indicates the maximum allowed changes for the new policy, regulated by a hyper-parameter ϵ_{clip} set to 0.2 in the original implementation.

4.3 Experiments

We benchmark the described algorithms in the TurtleBot3 mapless navigation domain, described in Section 3.4.1. Since this is the first time in the thesis we present a comprehensive training and testing experiment in a Unity environment, Gazebo, and the robot, we provide additional information and overviews to clarify the entire process, which is summarized in Figure 4.1.

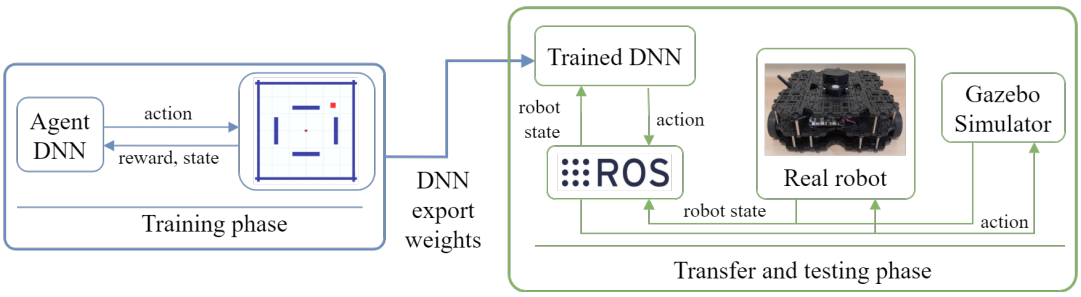

FIGURE 4.1: Overall architecture of our training and testing procedure in Unity, ROS, and the robot.

We aim at showing that value-based Deep Reinforcement Learning approaches with discrete action spaces could represent a viable and more sample-efficient alternative over policy-gradient and actor-critic algorithms. Moreover, value-based DRL results in comparable or better performance in tasks that employ physical control.

Training Setup: All the considered algorithms share the same input layer structure: 13-sparse laser scans and the target position (Tai et al. (2017) considers a similar setting). One computation of the network represents an action that directly maps into the angular velocity of the robot in the case of Double DQN (with $[-90, -45, 0, 45, 90]$ deg/s as possible angular velocities). In contrast, it is used end-to-end in the case of PPO and DDPG, multiplying such output by a hyper-parameter to shape the angular velocity in a range $[-90, 90]$ deg/s. In these experiments, we choose a constant linear velocity $= 0.15$ m/s. However, given the capability of generalization of the method, once the network is trained, it is possible to modify this value to obtain different behaviors. We did explore other encodings for the problem, increasing the number of sparse scan ranges up to 25 and decoupling the network's output in two streams to compute different linear and angular velocities. However, the higher complexity of the problem causes longer training times, but the success rate was similar. Hence, we

preferred a coarse structure of the action space to maintain a fast training phase. We summarize the input and output structure of the Double DQN algorithms in Figure 4.2

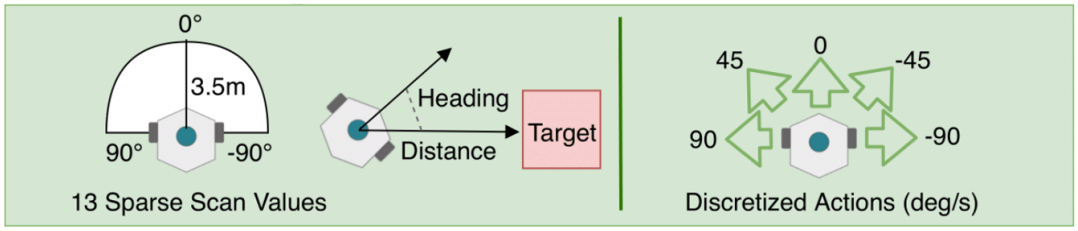

FIGURE 4.2: Overview of the input and output values for the Double DQN function approximator.

To determine the size of the hidden layers, we performed tests on different network dimensions (Chen and Chang, 1996). In particular, we performed multiple trials with different random seeds and network sizes. Figure 4.3 highlights the results of the chosen network architecture, noting that DDPG and PPO hidden layers have the same structure.

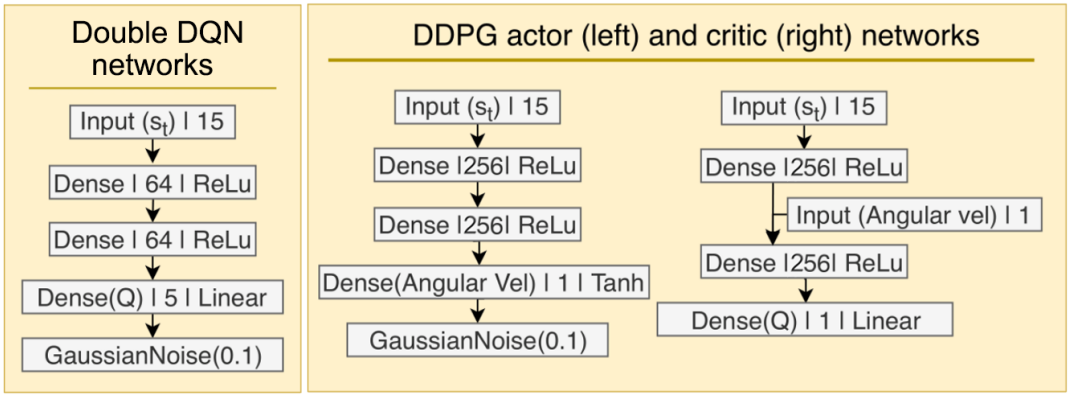

FIGURE 4.3: Network architectures for Double DQN, DPPG and PPO. Each layer is represented by type, dimension, and activation function.

Experimental Setup: Data are collected on a RTX 2080, using the standard hyper-parameters of the original algorithmic implementations (van Hasselt et al., 2016; Lillicrap et al., 2016; Schulman et al., 2017). Given the importance of the statistical significance of the results (Colas et al., 2019), we report mean and standard deviation collected over ten independent runs with different random seeds. Such setting motivates slightly different results over our published results that we evaluated over a few seeds (Marchesini and Farinelli, 2020a).

4.4 Empirical Evaluation

For each algorithm, we consider the following evaluation metrics:

- Success rate: how many successful obstacles-free trajectories are performed on a batch of one hundred episodes.

- Training time.
- Path length.

In particular, we note that after the training phase, the models were able to learn to navigate, exploiting the minimal information in the input layer of the network, generalizing three core navigation components: (i) robot starting position, (ii) target position, and (iii) velocity. The laser scan-based navigation also allows the TurtleBot3 to navigate in unknown environments with different obstacles, crucial for motion planning.

Training Results: Figure 4.4 shows that the discrete action space Double DQN (DDQN) offers better performances in terms of success rate over the same number of interactions. The training stabilizes at over 95% of success rate after about 3000 epochs corresponding to 50 minutes of training. To further support the viability of value-based algorithms, Figure 3.8 shows that to reach similar performance (i.e., 95% of success rate), the policy-gradient and actor-critic algorithms require at least four times the training time of DDQN.

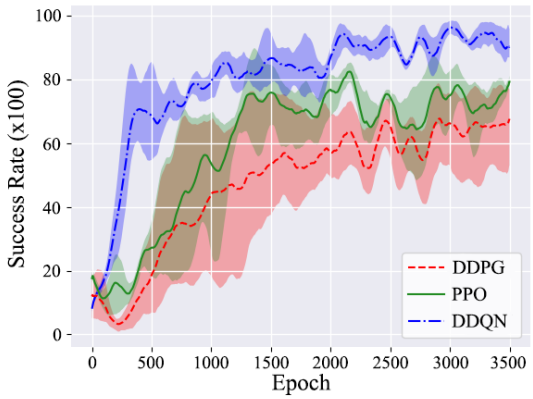

FIGURE 4.4: Average success rate between DDQN, DDPG, and PPO.

Moreover, we did run experiments using DDPG and PPO with the discrete action space for a more comprehensive evaluation. However, such implementations gave us no significant improvements both in terms of success rate and training time. Moreover, we note that in our relatively simple motion planning scenario, the four Deep Neural Network used by DDPG lead to the significant time-consuming training phase. The same considerations hold for more recent approaches such as TD3 (Fujimoto et al., 2018), an improved version of DDPG that uses two critics, hence six Deep Neural Networks.

Testing Results: Figure 4.5 shows our additional tests for Double DQN and DDPG in the training environment (on the left) and in a testing environment of size $3.5 \times 10.5m$ (on the right) that present previously unseen obstacles (e.g., cylinders). These figures report the trajectories generated by two exemplary executions of the trained models aiming at providing a visual representation of the behaviors for the different models and are consistent with our evaluations. Given a very similar behavior of DDPG and PPO, we show the path of just one of the two.

In particular, we notice a very close correspondence of the robot motion in the training scenario. However, the value-based algorithm offers a shorter path, $\approx 14m$ versus

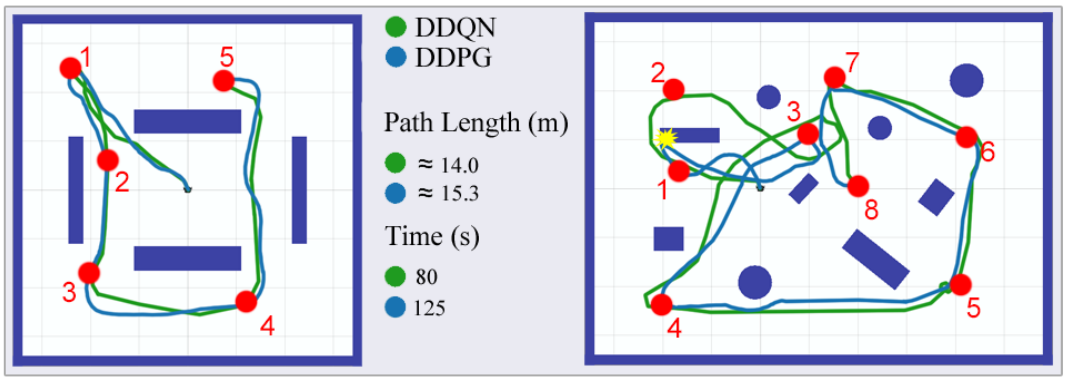

FIGURE 4.5: Explanatory comparison between the traveled path of a trained DDQN and DDPG policies in the training (left) and testing (right) environments.

$\approx 15.5m$, and takes less (simulation) time ($80s$ versus $125s$). Moreover, our testing environments show another crucial aspect of our evaluation, where the DDPG model fails to reach the second target goal (even with a training phase that is four times longer). We relate the failure of DDPG to the slow update rate of the laser sensor, which negatively affects the generalization capabilities of more complex algorithms such as DDPG. Our evaluation found that the discretization offered by DDQN deals better with the lag introduced by the lidar sensor over DDPG and PPO. Notice that this issue is not related to the learning algorithm or the chosen network architecture as the standard movebase motion planner (included in the ROS distribution) also fails for some obstacle configurations.[1]

4.5 Further Optimizations

We further enhance the results of these approaches (i.e., returns and training time) using asynchronous parallel training phases and multi-batch memories to employ the visited samples efficiently, which we detail in the following sections.[2]

4.5.1 Asynchronous Parallel Simulation

The nature of Unity is to manage and optimize the efficiency of multiple concurrent game threads, so we exploited the original asynchronous fashion of the game engine for the training process. In more detail, we separate the experience collecting process (i.e., the interactions in the environment) in a separate thread over the one that updates the networks' parameters. Moreover, we use multiple independent instances of the training environment to collect samples with parallel agents simultaneously. Figure 4.6 on the left shows the effectiveness of the asynchronous parallel DDQN version compared to the original one in a setting with 1, 2, and 4 parallel environments.

[1] We refer to the video attached to Marchesini and Farinelli (2020a) for a summary of our evaluation.

[2] These improvements were implemented in all the algorithms evaluated in Section 4.3.

4.5.2 Multi-Batch Memory

Given the mixed reward structure of the indoor navigation environment, we use a modified version of the Priority Experience Replay (PER) (Schaul et al., 2016) to accelerate training times. We modified the original priority system by introducing three memory batches: one for the successful interactions (i.e., the ones with $r_t = 1$) of size 5000, one for the unsuccessful ones (i.e., the ones with $r_t = -1$) of the same size and one with the other experiences of size 20000. Therefore, we update the models' weights with a batch of experiences composed of the same amount of samples taken from the three batches. We train our models with an equal-size batch of experiences from all the possible behaviors shaped by the reward function with this implementation. Figure 4.6 on the right shows the difference in terms of success rate between the original PER and this version, applied to the DDQN algorithm.

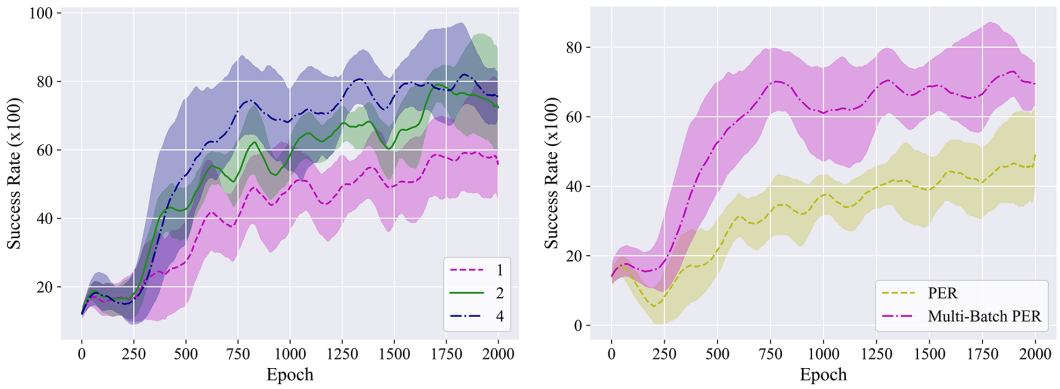

FIGURE 4.6: Average success rate between a 1, 2, and 4 thread implementation of DDQN (left) and with the standard PER and our multi-batch PER (right).

4.6 Related Work

The significant progress made by Deep Reinforcement Learning in solving challenging problems across various domains have been recently argued by several works (Mania et al., 2018; Henderson et al., 2018; Colas et al., 2019). In detail, Mania et al. (2018) represented one of the first works that criticize the importance of choosing benchmarks to evaluate DRL research. Along this line, Henderson et al. (2018) investigated challenges related to three main topics of interest: (i) reproducibility, (ii) proper experimental techniques, and (iii) reporting procedures for the results. In particular, the authors illustrated the variance in reported evaluations compared over original baselines implementations. Following a similar direction, Colas et al. (2019) proposed a comprehensive guide to foster rigorous comparison across different algorithms, reviewing the importance of statistical tests and empirically comparing them.

Hence, given the importance of the statistical significance of the results, through this thesis, we will exhaustively report the details of each experimental setup, including, for example, details about the chosen metrics, number of runs, mean and standard deviation of the results.

4.7 Discussion

In this chapter, we investigated the performance of value-based, policy-gradient, and actor-critic Deep Reinforcement Learning algorithms in the context of robotic mapless navigation. Our goal is to discuss the importance of choosing a particular algorithm depending on task-related requirements, in contrast to the typical trend of employing a specific algorithm for a specific task without considering different solutions. To this end, we use our robotic environments to benchmark the different families of algorithms, presenting further optimization to enhance their performance.

In more detail, the asynchronous parallel training and a multi-batch memory further improved the navigation performance, achieving a success rate of over 95% in a short time (i.e., 50 minutes) on a budget computational platform for the value-based approach. In contrast, training times of policy-gradient and actor-critic algorithms were significantly longer (i.e., ≈ 4.5 and ≈ 3.5 hours), and the resultant navigation policies performed worse than the discrete one in terms of success rate and path length.

Chapter 5

Evaluating Decision-Making

Choosing informative metrics is fundamental to determine whether algorithmic improvements are meaningful (Henderson et al., 2018; Colas et al., 2019).

This chapter discusses the limitations of standard evaluation metrics (e.g., average reward, success rate) in providing information regarding the actual behaviors of a trained Deep Reinforcement Learning policy. To this end, we use Formal Verification techniques to formally guarantee the decision-making process of a DRL model over a set of desired specifications. We also discuss further optimizations (e.g., scaling discounts) and how they lead to policies with higher performance.

5.1 Introduction

Applying Deep Reinforcement Learning to learn different behavioral skills for various platforms typically involves some desired behaviors (e.g., limited workspace, safety constraints) and high-cost hardware. Therefore, it is crucial to evaluate the correct behavior of a trained model before deploying the system in real applications (Liu et al., 2019). However, ensuring a provable behavior of a non-linear function approximator such as a Deep Neural Network is an active research topic. The lack of such formal guarantees represents one of the main issues that prevent the wider use of DRL systems for building trustworthy commercial solutions.

In this chapter, we characterize the behaviors of a Deep Reinforcement Learning model to show two essential aspects:

1. The limitations of standard evaluation metrics.
2. The effects of an improved training process on both the performance and decision-making of a model.

Regarding the former, we also aim at improving performance and reducing the training time by presenting a scaling discount factor and the use of a mixed exploration policy, based on a directional controller (Xie et al., 2018).

We evaluate our approach on the Panda environment, detailed in Section 3.3, where our optimizations allow us to increase the complexity of the scenario, reaching random targets in the whole workspace of the robot within millimeter precision. In contrast, previous approaches on DRL trajectory generation for manipulators consider only random targets generated in a limited fixed workspace within centimeter precision (Gu et al., 2017).

Moreover, for what concerns the Formal Verification of Deep Neural Networks, we note that several research efforts have been devoted to this problem (Liu et al., 2019; Wang et al., 2018c,b). However, prior work can not directly verify a Deep Reinforcement Learning model that encodes decision-making policies (e.g., which joint has to move to keep the manipulator in its workspace). The reason is related to their focus on verifying whether the bound of a single output of the network lies in a given interval (e.g., a motor velocity never exceeds fixed bounds). In contrast, a DNN for decision-making typically requires the analysis of multiple outputs (e.g., choose the action that maximizes a return). Hence, starting from existing verification tools, we describe the formalization of properties (i.e., input-output relationships) for decision-making, focusing on verifying the behavior of the Panda's trajectory generation. In more detail, we employ interval analysis to verify the relationships between two or more network outputs, allowing the verification of DRL models, where the output nodes correspond to actions, and the trained network chooses the one with the highest value.

5.2 Preliminaries

Similar to the indoor mapless navigation scenario in Section 4.3, we formalize the trajectory generation task for the Panda manipulator as a Reinforcement Learning problem, defined over a Markov Decision Process (Gu et al., 2017).

To evaluate the actual behaviors of a trained decision-making model, we argued that standard metrics such as the total reward or the success rate over independent epochs are not informative. Along this line, Szegedy et al. (2015) shows that human-imperceptible perturbations in the input space of a DNN may result in a significant difference in the output prediction, which is an open problem in literature (Madry et al., 2018). In particular, Sandy Huang and Abbeel (2017) highlighted that neural network policies, trained with state-of-the-art DRL algorithms, are also vulnerable to adversarial inputs. These specific input configurations are challenging to detect with empirical testing phases and are consequently undetectable using standard metrics, underlining the limits of the traditional evaluation approach.

Following the recent trend in Formal Verification for Deep Neural Networks, we propose to design a set of input-output relationships (or properties) that encode desired behaviors for a decision-making agent. To this end, we introduce the main concepts and notations useful to understand a typical verification process, which will be detailed in the following sections. Starting from the standard formalization for a property (Equation 2.13), we define an "area" as a set of inputs limited by an upper and lower bound. A "subarea" is then a further subdivision of the same input set used to reduce the overestimation during the computation (Wang et al., 2018b). The propagation of an area (or a sub-area) through a DNN generates an "output-bound", a set of bounds for each output node that quantifies the limits on the output values of the network, given an input in the provided area.

Figure 5.1 shows an example of these concepts, where two input nodes with area $([a_0, b_0], [a_1, b_1])$ are propagated through the network to compute the output-bound $[c, d]$ of the single output node. To further clarify the process, we employ two different bound representations. First, Figure 5.1 on the left shows a simple network with two inputs and one output. Otherwise, we visualize the Deep Neural Network function and bounds as a 2-dimensional graph on the right, with the inputs on the x-axis and

the computed output bounds $[c, d]$ on the y-axis.[1]. The red curve represents the actual values assumed by the output node.

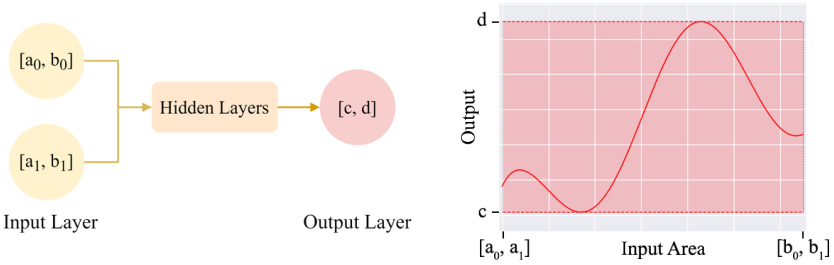

FIGURE 5.1: Explanatory overview of the bound analysis of a generic MLP with two inputs and one output.

5.3 Decision-Making Properties

We present our design for properties that encode desired behaviors for a trained decision-making agent and an overview of a typical verification algorithm.

Given the formulation provided by Liu et al. (2019), a property for a DNN formalizes an input-output relationship in the form of Equation 2.13. Such a relationship aims at verifying if an output of the network lies in a specific interval and applies to many problems related to robotics and deep learning in general (e.g., the velocity limit of a motor or the probability in a classification task).

To address this issue, we propose a different formulation, specifically designed for decision making problems; formally:

$$p : \text{If } x_0 \in [a_0, b_0] \land ... \land x_n \in [a_n, b_n] \Rightarrow y_j > y_i \tag{5.1}$$

We refer to these properties as *decision-making properties* as they can be used to ensure that a given action (e.g., y_j) is always preferred over the others for a given input configuration. Following the insights on Formal Verification of Section 2.4, we exploit this proposition to prove (or deny) a variety of properties in the context of trajectory generation for the Panda, which can be verified using prior verification approaches (Wang et al., 2018b).

As a practical example, consider a simplified navigation scenario encoded by a Deep Neural Network with:

- Inputs $x_i \in [0, 1]$ with $i = \{0, \dots, 3\}$, representing the normalized distance from an obstacle in the four cardinal directions (1 translates in a distance $\geq 1m$ in that direction), where x_0 is the *right* distance and x_1 is the *left* distance.
- Outputs $y_0 = right$, $y_1 = left$, representing the directions where the agent can turn.

where we could be interested in a natural language decision-making property as:

[1] We note that a network generally has more than one input, so we represent a tuple with one value for each input, but the order in the tuple is required only for visualization purposes.

$p_{\rightarrow}$: If an obstacle is close to the right and other directions are obstacle-free, always turn left (i.e., choose y_1).

Assuming prior knowledge on the task, typically available in practical applications, we know the minimum distance from an obstacle that allows avoiding a collision when turning in the opposite direction in the worst-case scenario, i.e., at maximum speed. This information, which we assume to be $0.07m$, leads to the following formal notation for the property in the form of Equation 5.1:

$$p_{\rightarrow} : \text{If } x_0 \in [0, 0.07] \wedge x_1, x_2, x_3 \in \mathcal{D} \Rightarrow y_0 < y_1, \text{where } \mathcal{D} = (0.07, 1] \tag{5.2}$$

Hence, to verify the relation between two (or more) outputs, we rely on the interval algebra of Moore (1963). In particular, supposing $y_0 = [c, d]$ and $y_1 = [e, f]$ we have the preposition:

$$d < f \Rightarrow y_0 < y_1 \tag{5.3}$$

To further explain the issues of previous verification approaches that prevent a direct application to decision-making, Figure 5.2 shows a simplified visual example of a typical output analysis. After computing the output bounds, a typical decision-making scenario presents $max(y_1) > min(y_0)$, and it is not possible to assert if the desired property holds. Figure 5.2 on the left shows an example of this behavior, where $d \not< c$, (i.e., the bounds overlaps). In this situation, verification frameworks can not formally verify the property (i.e., we do not have enough information to state if the input-output relation holds or not). We overcome this issue by computing the propagation phase for a subset of the input area, obtaining a more accurate estimation of the output function shape (Figure 5.2 in the middle). This leads to right Figure 5.2, where the use of smaller bounds allows to check if $y_1(x) < y_0(x)$ for any $\mathbf{x} \in \mathcal{X}$. Hence we verified that $y_1 < y_0$ (i.e., the network always chooses the action represented by y_0 inside the input area specified by the property). Furthermore, $y_2 \not< y_1$ (the agent can choose y_2 in that input domain).

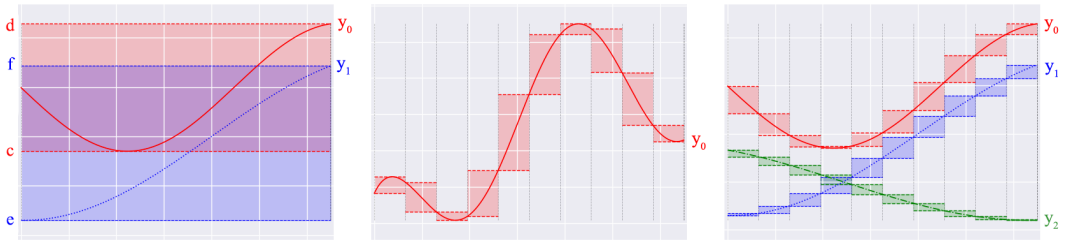

FIGURE 5.2: Explanatory output analysis of a decision-making problem with two outputs and one subdivision (left); the estimation of an output function shape, using multiple subdivisions (middle); and the output analysis with three outputs and multiple subdivisions (right).

Algorithm 3 shows a pseudo-code description of the overall approach that relies on the bound estimation process of prior work (Wang et al., 2018b). In detail, given as input: (i) the trained neural network to test, (ii) the input area to perform the analysis, (iii) the property to verify, and (iv) the number of desired subdivisions (which is a hyper-parameter):

- Line 1-2: we initially perform the split of the original input area. This function uses a simple heuristic that splits the input area with the largest bound as it performs better than a random strategy in our initial evaluation.[2]. We also initialize an array that will contain all the states that lead to a violation of the tested property.
- Lines 3-5: We then use the input propagation of Neurify (Wang et al., 2018b) on the smaller sub-areas to compute the output bounds. This is the most computational demanding section of the process due to the non-trivial number of sub-areas. However, each propagation is strongly independent of the others, enabling parallel computation on Graphics Processing Units (GPUs).
- Lines 6-12: Hence, it is possible to evaluate the desired property for each generated sub-area, which can produce three different results: (i) the property is verified; (ii) the property is denied or (iii) in this area we can not conclude anything on the property (i.e., more subdivisions are required to solve the property).
- Line 13: Finally, the verification procedure ends, providing additional information. If the returned array "violation-areas" is empty, we know that the property is verified, and no violations are found in the input area. Otherwise, the array contains the input configuration that causes a wrong evaluation from the network, which can be quantified to evaluate the actual behaviors of the model over the properties.

Algorithm 3

Given:

- a DRL trained model to test f_θ
- the input domain (area) where we are interested in verifying the property $\mathcal{D}$, and the number of subdivisions n_{sub}
- the property p to verify

1: sub-areas ← *split-area*($\mathcal{D}$, n_{sub}) ▷ e.g., using an heuristic
2: violation-areas ← [ ]
3: **for** sub-area **in** sub-areas **do**
4: output-bounds ← *get-out-bound*(f_θ, sub-area) ▷ e.g., using Neurify, ProVe
5: **end for**
6: **for** output-bound **in** output-bounds **do**
7: is-violation ← *check-property*(output-bound, p) ▷ e.g., using Eq. 5.3
8: Append sub-area to violation-areas if is-violation is true
9: **if** is-violation is unknown on output-bound **then**
10: **return** False, [ ] ▷ i.e., increase n_{sub} and repeat
11: **end if**
12: **end for**
13: **return:** True, violation-areas

5.4 Experiments

We evaluate the behaviors of the Panda in the trajectory generation task using a Deep Reinforcement Learning trained policy. Since this is the first time in the thesis we

[2]This is a crucial step to reduce the number of the subdivision to perform. Therefore this is an exciting area for future research

present a comprehensive training and verification experiment from the manipulator environment to the robot, we provide additional information and overviews to clarify the entire process, which is summarized in Figure 5.3.

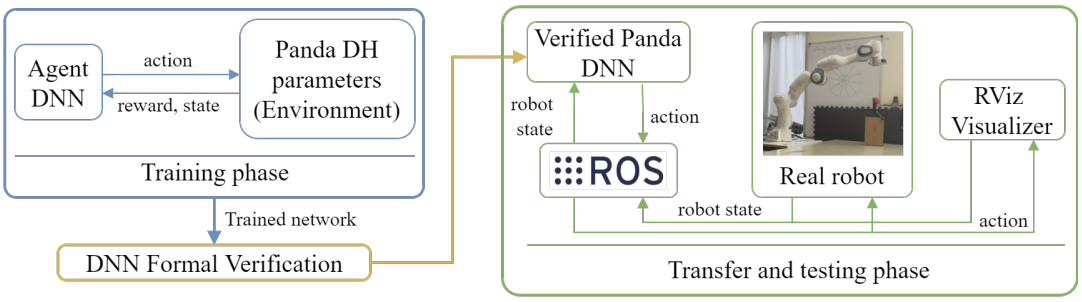

FIGURE 5.3: Overall architecture of our training and verification procedure from Unity to the robot.

We aim at showing that formal verification is a viable methodology to evaluate the behaviors of trained decision-making models. In particular, we show how further optimization for the training process makes the resultant policy more robust, obtaining a lower number of violations over the desired behaviors.

Training Setup: We consider a Double DQN algorithm for the training of the trajectory generation task presented in Section 3.3, which we enhance with the following optimizations.

Linear Scaling Discount Factor

The joint steps that compose the trajectory of the Panda tend to be minimized in the advanced stages of the training as the network learns to reach the target minimizing the steps to obtain the positive reward. Discount is thus crucial, and prior work considers to upscale the discount, starting to learn by maximizing rewards on a short-term horizon and progressively giving more weight to delayed rewards (François-Lavet et al., 2015). In contrast, our approach works oppositely. To address the sparsity of the reward, we initially give more importance to delayed rewards and then to short-term ones, hence scaling the discount factor linearly downwards to achieve higher returns.

Exploring with a Directional Controller

Knowing the kinematics of the manipulator, we designed a mixed exploration method that replaces part of the random choices of the standard ϵ-greedy strategy with correct actions, providing the network with correct samples. For example, given j_0 the current position of $joint_0$ and j_{0f} its final pose when generating the target, if $j_0 < j_{0f}$ then a clockwise action is selected. Moreover, training the network using the controller as policy when the end-effector reaches a minimum distance from the target ($5cm$ in our experiments) enables us to reach a millimeter precision $< 0.1cm$. This further motivates the results of Chapter 4 as state-of-the-art continuous Deep Reinforcement Learning for trajectory generation achieved trajectories within a $5cm$ error (Gu et al., 2017).

Experimental Setup: Similar to the experiments in Section 4.3, data are collected on an RTX 2080, using the standard hyper-parameters of the original algorithmic implementations of Double DQN (van Hasselt et al., 2016) and ten independent runs with different random seeds. Considering the sparsity of the reward in this environment, described in Section 3.3, our best performing implementation of the linear scaling discount factor considers an initial $\gamma = 0.99$. We then gradually scale down to $\gamma = 0.8$ for each improvement measured in the success rate (i.e., a decay of 0.0019 for each successful trajectory).

5.5 Empirical Evaluation

We consider the success rate as standard evaluation metrics that measures the number of successful trajectories over a batch of one hundred epochs.[3] An epoch ends when a correct trajectory is generated or a fixed timeout of actions is reached (300 in our experiments).

Figure 5.4 on the left shows that the scaling discount factor offers more stability during the training over a fixed discount $\gamma = 0.95$ (i.e., the best performing one in our evaluation). This result considers the same setup of prior work with an error between the target and the end-effector of $5cm$ (Gu et al., 2017), and an ϵ-greedy exploration policy. In more detail, the training stabilizes at over 95% of successes after about 20000 iterations. This corresponds to 3 hours of training with our computational platform. Crucially, the scaling discount presents a success rate that is on average 10-15% better over the fixed discount, keeping the same trajectory dimension in terms of joint steps. Hence, the following results will consider only training performed with a scaling discount. Moreover, Figure 5.4 on the right depicts the similarity between two explanatory trajectories to the same target position: one generated by a trained Double DQN model, and the other one computed with the inverse kinematic solver of Robotic Toolbox[4].

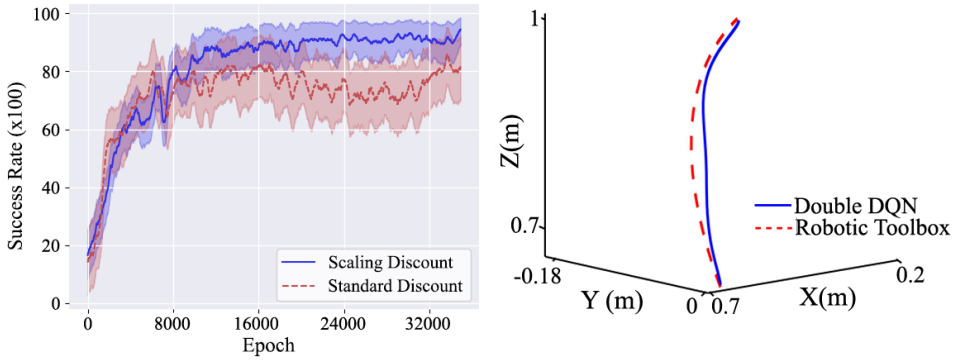

FIGURE 5.4: Average success rate between a Double DQN with our scaling discount and a fixed one (left). Explanatory trajectory generated by the trained Double DQN model and the Robotic Toolbox in Matlab (right).

Figure 5.5 on the left shows the performance with the mixed policy based on the directional controller. A preliminary evaluation shows the best results using the controller

[3] Note that the average reward would provide the same results, due to the sparse reward signal of the environment.

[4] petercorke.com/wordpress/toolboxes/robotics-toolbox

actions in 25% of the ϵ cases (a greater value of controller actions causes a drop in performance). This result offers over 90% of success rate after about 30000 iterations, reducing the trajectory error to $2cm$. By providing more correct trajectories in the first stages of the training, we obtain superior performance (i.e., lower error) faster (i.e., 2 hours). As detailed in the previous section, when the controller is used in the last part of the trajectory (i.e., when the target's distance is $2cm$), it is possible to train the same Double DQN model to reach the target with millimeter precision (Figure 5.5 on the right).

FIGURE 5.5: Average success rate with our mixed exploration phase with an error of $2cm$ between the end-effector and the target (left). Average success rate with the directional controller for the last part of the trajectory with an error of $0.1cm$ (right).

5.5.1 Evaluating Behaviors with Formal Verification

While the success rate provides a clear indication of the performance of the trained Double DQN model in reaching random targets, it does not provide any information on important features of the trajectory executed by the arm.

Assuming an industrial application, we could be interested in verifying that the manipulator always operates inside its workspace. Therefore, we require that the decision-making properties, formalized as Equation 5.1, describe the following behavior. Suppose the current position of a joint j_i equals one of its domain limits, regardless of the configuration of the other joints and the target's position. In that case, the robot must not rotate j_i in the wrong direction. Trivially, an action that rotates j_i causes the robot to exit from the workspace.

To this end, we want to verify that the model never selects a specific action over a set of others. In detail, the trajectory generation environment for the Panda has one output for each joint ($v_{0:5}$), hence we have to verify the property over multiple outputs. Equation 5.1 is thus in form:

$$p : \text{If } j_0 \in I_{j_0} \wedge ... \wedge j_5 \in I_{j_5} \wedge g_{x,y,z} \in I_g \Rightarrow v_0 < [v_{1:5}]$$

where I denotes the input area for specific input. A property in this form verifies whether the network chooses to move the joint j_0 (that corresponds to output v_0). To avoid such behavior, we require an area (or sub-area) with a lower bound of $[v_{1:5}]$ greater than the upper bound of v_0. To summarize, it is sufficient that one of the other actions has a greater value to ensure that the agent never chooses the specified output.

Following the environment specification, a property that checks whether a joint exit from its workspace is formalized as follows:

$$p_{0L} : \text{If } j_0 \in [1,1] \wedge \{j_{1:5}, g_{x,y,z}\} \in \mathcal{D} \Rightarrow \mathrm{v}_0 < [v_{1:5}], \text{ where } \mathcal{D} = [0,1)$$

Property p_{0L} represents a configuration where the position of j_0 equals its limit on the left (i.e., a normalized value 1). For this reason, whatever values the other network inputs assume, the output corresponding to the action j_0 rotates left must be lower than at least one of the others. Hence, for each joint j_i we consider two properties, one for the left limit (p_{iL}) and one for the right limit (p_{iR}). To show that our optimizations lead to a network that behaves better according to the desired specifications, we evaluate these properties on two trained networks: one trained with a standard Double DQN (*Standard*) and one with the scaling discount and the mixed exploration (*Optimized*). As evaluation metrics, we use the percentage of input configurations (i.e., sub-areas) that cause a property violation during our formal analysis, calculated over the size of the original input area. We note that such metric is an upper bound of the actual violations during the execution of the trained model, and this can be considered a worst-case analysis. Our analysis assumes an equally distributed probability for a state to belong to one of the sub-areas. However, when executing the trajectory, some input configurations appear more frequently, while property violations are usually restricted to the limits of the workspace. To summarize, during the testing phase of a trained model, a violation may never appear because undesirable behaviors belong to a restricted sub-set of input configurations that rarely occur.

Table 5.1 shows the results of our evaluation, which confirm that the network trained with our optimizations better behaves over the properties. In more detail, the model trained with standard Double DQN has a significantly higher percentage of violations for p_{0R} and p_{3L} (where $> 65\%$ of input configurations lead to a violation of the property). Furthermore, even if the optimized model has a higher failure rate in specific properties, the average number of undesirable behaviors is significantly lower (0.12% in contrast to 12.33%).

TABLE 5.1: Property verification results. For each property we show the percentage of property violations for the *Standard* and the *Optimized* models.

Property	***Standard*** **(%)**	***Optimized*** **(%)**
p_{0L}	0.00	0.17
p_{0R}	78.76	0.00
p_{1L}	0.00	0.12
p_{1R}	0.01	0.00
p_{2L}	0.31	0.01
p_{2R}	0.00	0.03
p_{3L}	68.33	0.50
p_{3R}	0.27	0.28
p_{4L}	0.04	0.10
p_{4R}	0.04	0.03
p_{5L}	0.09	0.08
p_{5R}	0.08	0.12
Average	12.33	0.12

5.6 Related Work

In this section, we first discuss DRL approaches for manipulator's trajectory generation. Hence, we briefly discuss the broad field of formal verification that we employ to verify our trained models' behaviors.

5.6.1 Learning Trajectory Generation

Deep Reinforcement Learning has been previously employed on seven degrees of freedom robotic arms (Gu et al., 2017; Marchesini et al., 2019). These methodologies, however, most consider a limited target area or fixed targets during the test on the real robot (e.g., a door handle). An exhaustive evaluation of DRL algorithms to perform the trajectory generation of a similar manipulator is presented in Gu et al. (2017). In particular, authors consider only continuous action space algorithms, which limitations have been discussed in Section 4.3. Moreover, their random target position is sampled uniformly from a cube of size $0.2m$ centered around a point, with a precision of $5cm$. In contrast, Marchesini et al. (2019) shows that it is possible to use discrete action space algorithms for the trajectory generation problem, but considering an error of $5cm$ between the target and the end-effector. Following this, we optimized Double DQN to generate a trajectory in the whole considered workspace of the manipulator and reduce the error between the target position and the end-effector to $< 0.1cm$, using our directional controller.

5.6.2 Formal Verification for Deep Neural Networks

Traditionally, the validation of neural networks relies on a large evaluation of a set of input points and the analysis of the corresponding outputs. This determines whether they belong to the desired set of bounds (Liu et al., 2019). However, since the input space is continuous and Deep Neural Networks are vulnerable to adversarial examples (Szegedy et al., 2015), a practical evaluation strategy can not provide reliable results (Papernot et al., 2016). An entire family of formal approaches extends the Boolean Satisfiability or the Satisfiability Modulo Theories to find configurations that falsify assertions in the form of Equation 2.13 (Katz et al., 2017; Dutta et al., 2017; Bunel et al., 2017). A different line of research aims at reducing the formal verification to an optimization problem, trying to falsify a set of safety properties (Lomuscio and Maganti, 2017; Tjeng et al., 2017; Raghunathan et al., 2018). However, all these approaches suffer from a scalability problem on large networks that prevents their application on real complex problems (Wang et al., 2018c). A promising method to address this issue relies on Moore's interval algebra (Moore, 1963). In particular, ExactReach (Xiang et al., 2017) and MaxSense (Xiang et al., 2018) represent the first approaches in this direction. While the former is an exact method that does not scale to large networks, the latter proposes an approach to partition the input domain but suffers from a severe loss of information. Neurify (Wang et al., 2018b) addresses these issues, exploiting the interval analysis propagation of ExactReach, to obtain a scalable and reliable approach. For our behavioral analysis, we use Neurify by splitting the input space subdivision into independent sub-intervals, enabling a direct computation of the number of violations over the properties.

5.7 Discussion

In this chapter, we investigated further optimizations to improve the performance of the Double DQN algorithm (i.e., a scaling discount factor and a mixed exploration policy). Moreover, we discussed the limits of standard evaluation metrics in the evaluation of the actual decision-making process of a trained Deep Reinforcement Learning model. The reported results show that our approach can verify whether a trained DRL model respects a set of properties that describes a desired set of behaviors. Moreover, the empirical evaluation highlighted that a model trained with the proposed optimization tends to learn the task faster with a higher success rate, which intuitively results in a lower percentage of violations.

5.8 Conclusions

In this first part of the thesis, we focused on providing a comprehensive overview of recent issues related to the choice and the evaluation of Deep Reinforcement Learning algorithms.

We started by presenting three robotics environments that we will use through the thesis as practical evaluation scenarios. Crucially, we showed the benefits of using non-standard simulation software, such as Unity, to improve the physics simulation while exporting the trained models on the real robots without further training.

Hence, we highlighted the issues related to the choice of the training algorithms, benchmarking value-based, policy-gradient, and actor-critic approaches in our mapless navigation domain, presenting further optimizations (i.e., the asynchronous parallel training and the multi-batch memories). We also showed that value-based DRL approaches could be a more sample-efficient alternative over policy-gradient algorithms, resulting in comparable or better performance in tasks that employ physical control.

Finally, we discussed the limitations of evaluating these methodologies with standard metrics (e.g., the average reward) as they are not informative on the actual decision-making of the models. To this end, we leveraged existing Formal Verification approaches to characterize the actual behavior of the trained policies over desired specifications. Moreover, we evaluated further optimizations (i.e., a scaling discount factor and a directional controller for exploration) within FV to show that optimizing the training process also leads to models with better behaviors.

Part II

Enhancing Exploration

Chapter 6

Combining Deep Reinforcement Learning with Evolutionary Algorithms

Exploration in high-dimensional spaces is a crucial challenge in Deep Reinforcement Learning. Despite its importance, typical exploration strategies use naive heuristics such as ϵ-greedy action selection (e.g., DQN (Mnih et al., 2013b)) or Gaussian control noise (e.g., DDPG (Lillicrap et al., 2016)). Moreover, the lack of diverse exploration in such complex domains leads DRL algorithms to a premature convergence to local optima (Pathak et al., 2017; Ostrovski et al., 2017).

This chapter discusses an emergent research direction that combines gradient-based DRL methods with gradient-free population-based approaches, typical of Evolutionary Algorithms (Fogel, 2006). EA, in particular, have the advantages of enabling diverse exploration and improving robustness, leading to a more stable convergence. In detail, we employ the indoor mapless navigation task to highlight the issues of prior combined approaches, which hinders their combination with value-based Deep Reinforcement Learning.

6.1 Introduction

The ability to adapt to the surrounding environment by generalizing from a massive amount of training experiences is the key factor behind the recent success of Deep Reinforcement Learning (Mnih et al., 2013a; Silver et al., 2018a; OpenAI et al., 2019). However, the sample efficiency of DRL algorithms is not the only limitation that hinders broader applications of such learning techniques to more complex scenarios. These solutions have to cope with the uncertainties and the dynamics of the operational environment. Hence, Deep Reinforcement Learning also suffers from convergence to local optima, mainly caused by the lack of diverse exploration when operating in high-dimensional spaces.

Classic exploration strategies in value-based algorithms with discrete action spaces are usually limited to ϵ-greedy action selection, which selects the action that maximizes the current Q-value with probability 1 - ϵ or uses a random action. Another approach considers injecting noise in the action selection to increase the diversity seen by the agent. In contrast, policy-gradient algorithms with continuous action space sample actions from the current stochastic policy, typically translating into sampling a Gaussian distribution that adds to a deterministic base policy. These naive exploration

strategies applied to DRL algorithms achieved non-trivial performance in a variety of tasks (Mnih et al., 2013b; Lillicrap et al., 2016; Schulman et al., 2017; Fujimoto et al., 2018). However, they are inadequate when rewards are sparse or the observation and action spaces to explore are simply too large. A typical example of this issue is Montezuma's Revenge Atari game, where also the variants of DQN fail to achieve scores at the level of a novice human due to inadequate exploration (Mnih et al., 2013b; van Hasselt et al., 2016; Wang et al., 2016; Mnih et al., 2016).

Several studies address the exploration problem to encourage better exploration. To this end, intrinsic motivation seeks to maximize an additional task-independent intrinsic reward function while optimizing to the reward signal (Barto et al., 2013; Oudeyer and Kaplan, 2013). Typical examples are:

- Empowerment: it measures the level of control the agent has over its future.
- Surprise: an agent takes more reward to act differently over its understanding of the environment.
- Novelty: the agent takes more reward to explore new states, which is related to surprise (Ostrovski et al., 2017).

Barto et al. (2013) and Oudeyer and Kaplan (2013) present an exhaustive taxonomy and comparison of intrinsic motivation approaches. However, we note that these strategies typically rely on sensitive task-specific hyper-parameters, which is another significant issue in Deep Reinforcement Learning. The sensitivity to hyper-parameters is, in fact, the main responsible for brittle convergence properties and poor performance in practical tasks of DRL (Haarnoja et al., 2018; Henderson et al., 2018).

In contrast, Evolutionary Algorithms (Fogel, 2006) have been recently employed as a promising gradient-free optimization alternative to Deep Reinforcement Learning. The redundancy of these population-based approaches has the advantages of enabling diverse exploration and improving robustness, leading to a more stable convergence. In particular, Genetic Algorithms (GA) (Montana and Davis, 1989) show competitive results compared to gradient-based DRL (Such et al., 2017) and have low computational cost. However, these gradient-free approaches struggle to solve high-dimensional problems, have poor generalization skills, and are significantly less sample efficient than gradient-based methods.

Hence, an emergent research direction proposes the combination of gradient-free and gradient-based methods following the physical world, where evolution and learning cooperate in assimilating the best of both solutions (Simpson, 1953). The first mixed approach, Evolutionary Reinforcement Learning (ERL) (Khadka and Tumer, 2018), relies on the actor-critic architecture to inject information into an evolutionary population. At the same time, both the gradient-free and gradient-based training phases proceed in parallel. Similarly, Proximal Distilled ERL (PDERL) (Bodnar, 2020) extends ERL with different evolutionary methods. CEM-RL (Pourchot and Sigaud, 2019) brings this research field into the family of distributed approaches, combining a portfolio of TD3 (Fujimoto et al., 2018) learners with the Cross Entropy Method (CEM) (Duan et al., 2016).

These combined approaches, however, also present two significant limitations, which we discuss through this chapter:

- The parallel training phases of the Deep Reinforcement Learning and evolutionary components (Khadka and Tumer, 2018; Bodnar, 2020), or the multitude of learners (Pourchot and Sigaud, 2019) result in significant overhead.
- The combination strategy of prior work does not ensure better performance than the DRL agent as it does not prevent detrimental behaviors (e.g., performance drop).
- The actor-critic formalization of previous mixed approaches allows them to a straightforward evaluation in continuous locomotion benchmarks (Brockman et al., 2016; Todorov et al., 2012). However, this also hinders their combination with value-based Deep Reinforcement Learning.

The latter is crucial as recent work (Matheron et al., 2019) investigated the unsatisfactory performance of actor-critic solutions in deterministic tasks that, in contrast, can be effectively addressed with value-based DRL. Hence, in this chapter, we discuss the poor performance of value-based implementations of prior combined approaches (Khadka and Tumer, 2018; Bodnar, 2020) in our discrete action space indoor mapless navigation task.

6.2 Preliminaries and Related Work

Following the trend of using Evolutionary Algorithms, or Evolutionary Strategy (ES), as an alternative optimization process over Deep Reinforcement Learning (Such et al., 2017; Salimans et al., 2017), an emergent research field proposed the combination of gradient-free population-based approaches and gradient-based solutions.

In detail, ERL (Khadka and Tumer, 2018) considers an actor-critic DDPG agent (Lillicrap et al., 2016) and a concurrent EA training that generates a population of individuals, which are mutated and selected based on their fitness. The DRL agent trains in parallel from the samples generated by both the training phases. It is thus periodically injected into the running population, which is also used to collect the training performance. The mutation function of ERL ensures that, in a certain number of episodes, the gradient-based policy outperforms its evolutionary siblings, introducing the gradient-based benefits into the population. Hence, such a mutation pattern biases the selection process of the next generation and its performance. In their experiments, authors highlight an efficient transfer of information between the two families of algorithms, outperforming DDPG in well-known locomotion benchmarks (Brockman et al., 2016; Todorov et al., 2012). However, both the usage of all the experiences in the buffer and forcing the Deep Reinforcement Learning agent to perform better than the evolutionary population bias the training and cause detrimental behaviors, which we will show in the following section.

Inspired by ERL, several combinations have been proposed (Bodnar, 2020; Colas et al., 2018; Pourchot and Sigaud, 2019; Khadka et al., 2019). While GEP-PG (Colas et al., 2018) is a precursor of a combined approach, where a curiosity-driven approach fills the buffer of the agent, PDERL (Bodnar, 2020) addresses the EA component of ERL. In particular, PDERL introduces novel evolutionary operators to compensate for the simplicity of the genetic representation (as investigated by Lehman et al. (2018), where authors address destructive behaviors of biologically-inspired variation operators applied to neural networks, which causes catastrophic forgetting). We also mention CERL (Khadka et al., 2019) and CEM-RL (Pourchot and Sigaud, 2019) as

they are extensions of ERL for distributed training with multiple active learners (i.e., gradient-based agents), which leads to a significant overhead for the training process.

However, these works share a common baseline as they all rely on actor-critic Deep Reinforcement Learning and build on the insights of ERL. Hence, we choose ERL and PDERL to show the poor performance of these approaches when combined with value-based algorithms.

6.3 Experiments

We evaluate the performance of prior combined approaches in the indoor mapless navigation task with a discrete action space, which makes it suitable for value-based algorithms. We aim to show the limitations of such prior algorithms to provide a comprehensive evaluation that will introduce and highlight the contribution of our combined framework in the next chapter.

Training Setup: We consider the Rainbow algorithm (Hessel et al., 2018) as the gradient-based component for ERL and PDERL. Rainbow represents a state-of-the-art value-based approach that combines all the improvements developed for DQN over the years (van Hasselt et al., 2016; Wang et al., 2016; Schaul et al., 2016; Mnih et al., 2016; Bellemare et al., 2017). In contrast, for the gradient-free evolutionary component, we refer to the original authors' implementations (Khadka and Tumer, 2018; Bodnar, 2020).

Experimental Setup: Similar to previous chapters, data are collected on an RTX 2080, using the standard hyper-parameters of the original algorithmic implementations (Hessel et al., 2018; Khadka and Tumer, 2018; Bodnar, 2020) and ten independent runs with different random seeds. In order to get reproducible and consistent results in this comparison, the random seed is fixed across a single run of every algorithm (because there may exist a sequence of targets that favor a run or a better network initialization), while it varies in different runs. Consequently, a specific run of every algorithm executes the same sequence of targets and initializes the networks with the same weights.

6.4 Empirical Evaluation

Similar to previous chapters, we consider the success rate as evaluation metrics that measures the number of successful trajectories over a batch of one hundred epochs.

As previously discussed, Figure 6.1 shows that a direct combination of ERL with a value-based algorithm can not cope with the issues of such Deep Reinforcement Learning approaches. Indeed, the ERL combination with Rainbow results in detrimental performance. Given these results, we performed an additional evaluation with the improved version of ERL, PDERL, which introduces novel genetic operators to improve ERL robustness. These improved genetic operators achieved slightly better performance over the naive ERL version. However, PDERL still provides detrimental performance over the standard Rainbow.

Hence, we conjecture that the detrimental behavior of previous mixed approaches is related to their Deep Reinforcement Learning injection pattern and the bias in the selection operator, rather than the simplicity of their gradient-free component. Furthermore, in a task such as mapless navigation, the parallel training phases of

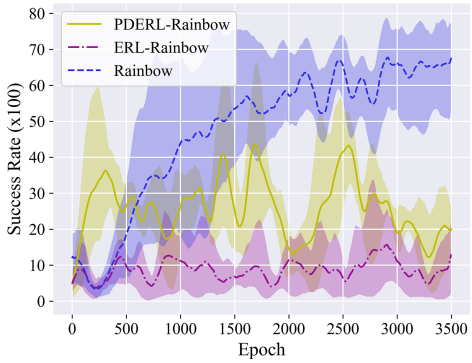

FIGURE 6.1: Average success rate between Rainbow and its combination with ERL and PDERL.

the gradient-based and gradient-free components do not provide a robust evaluation. Intuitively, the gradient-free population evaluates over different random targets, which can be easier to reach, hence selecting a genome that does not represent an overall better navigation policy. Follows that the episodes of the population component should be accurately modeled to ensure a robust evaluation in a practical task such as navigation.

6.5 Discussion

In this chapter, we investigated the performance of value-based implementations of previous combined approaches, ERL and PDERL. We discussed the limitations of such frameworks, which typically return detrimental performance when combined with value-based algorithms. This is crucial as previous chapters show the importance of value-based Deep Reinforcement Learning both in standard benchmarks and in practical applications.

For this reason, the next chapter investigates a generally applicable framework that allows the combination of Evolutionary Algorithm with both policy-gradient (or actor-critic) and value-based algorithms, ensuring to match the performance of the gradient-based component in a worst-case scenario.

Chapter 7

Genetic Soft Updates for Policy Evolution

Evolutionary Algorithms (Fogel, 2006) represent a natural way to improve the exploration abilities of Deep Reinforcement Learning agents using genetic operators such as mutations and crossovers. In particular, the combination of gradient-free EAs and gradient-based DRL show promising results in enhancing the performance of existing actor-critic algorithms, addressing the typical issue related to the lack of diverse exploration (Khadka and Tumer, 2018; Bodnar, 2020). However, their combination strategy, described in the previous chapter, leads to significant issues when combined with value-based approaches (Marchesini et al., 2021), which typically present more brittle convergence properties over policy-gradient (and actor-critic) algorithms (van Hasselt et al., 2018).

For this reason, this chapter presents a generally applicable framework that allows complementing any Deep Reinforcement Learning algorithm with Evolutionary Algorithms, improving the performance of prior work with every considered gradient-based approach. Crucially, our combination strategy matches the performance of the gradient-based component in a worst-case scenario, enabling the usage of value-based solutions. Moreover, we show that our framework addresses the sensitivity of DRL algorithms to hyper-parameters, significantly improving the returns in settings where the only Deep Reinforcement Learning component presents pathological performance (e.g., a specific seed initialization).

7.1 Introduction

Our framework, which we refer to as Soft Updates for Policy Evolution (Supe-RL), enables us to combine the characteristics of Evolutionary Algorithm (in particular, a Genetic Algorithm) with any DRL algorithm, addressing the limitations of previous approaches. The general idea is to benefit from the high sampling efficiency of gradient-based Deep Reinforcement Learning while incorporating gradient-free GA to generate diverse experiences and find better policies.

Figure 7.1 provides a general overview of the methodology. In more detail, Supe-RL based algorithms perform a periodic genetic evaluation applying a Genetic Algorithm to the agent network's weights. A selection operator uses a fitness metric to evaluate the population, choosing the best performing genome (i.e., the weights of the network) used to update the weights of the Deep Reinforcement Learning agent. In contrast to previous work, our genetic evaluation is only performed periodically, drastically

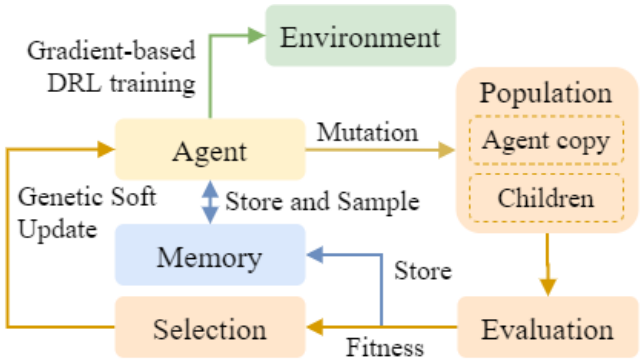

FIGURE 7.1: High-level overview of the Supe-RL framework.

reducing the overhead. Furthermore, our soft update combination strategy, detailed in Section 7.2 allows direct integration of a GA to any DRL algorithm as it is similar to performing a gradient step towards a better policy, avoiding detrimental behaviors. As previously discussed, this allows the combination with value-based approaches, which benefit from the variety of optimizations developed for DQN, namely the Rainbow algorithm (Hessel et al., 2018). Crucially, our genetic component influences the Deep Reinforcement Learning agent policy only if one of its mutated versions performs better in a subset of additional evaluation epochs. Hence, with a sufficient number of epochs, we obtain a reasonable estimation of the overall performance of the population.

To provide a comprehensive evaluation of our approach, we consider both the discrete action space indoor mapless navigation and the continuous action space aquatic navigation, which allows testing Supe-RL with different DRL algorithms. To further confirm the superior performance of our framework, we also employ standard benchmarks (i.e., MuJoCo locomotion (Brockman et al., 2016; Todorov et al., 2012)) to compare over prior work (Khadka and Tumer, 2018).

7.2 Genetic Soft Updates for Policy Evolution

The central insight of Supe-RL is to soft update a Deep Reinforcement Learning agent towards better policy regions, enabling the combination with any DRL algorithm. Crucially, given the more brittle convergence properties of value-based DRL, in this section, we discuss the general procedure to complement a Deep Reinforcement Learning approach with our framework. Therefore, we separately discuss other tricks that allow Supe-RL to work with any gradient-based algorithm.

Algorithm 4 reports a general description of a typical execution of our framework, which proceeds as follows:

- Lines 1-8: the Deep Neural Network weights of a DRL agent drl_a, also referred to as genome or θ_a, are initialized with random values. Following a standard training setup, the agent then starts to collect experiences interacting with its environment. Such experiences are stored in a memory buffer to train θ_a.
- Lines 9-12 (Algorithm 5): periodically, i.e., every $e_{\mathcal{P}}$ epochs, Supe-RL generates a population $\mathcal{P}$ of children, each one characterized by a different genome θ_p (with $p = \{0, \ldots, |\mathcal{P}|\}$). The agent's weight θ_a are used to create n individuals

Algorithm 4 Supe-RL pseudocode

Given:
- the size n of the population $\mathcal{P}$
- the periodicity $e_{\mathcal{P}}$ for the genetic evaluation
- a gradient-based DRL agent in environment *env* initialized with seed s

1: **for** epoch = 1 **to** ∞ **do**
2: **for** t = 1 **to** T **do**
3: Following the agent's policy, select action a_t
4: Execute a_t, observing state s_{t+1}, reward r_t
5: Store (s_t, a_t, r_t, s_{t+1}) in the agent's memory buffer
6: **end for**
7: Sample a random mini-batch of transitions from the memory buffer
8: Perform a gradient descent step updating the agent's weights θ_a
9: **if** epoch % $e_{\mathcal{P}}$ == 0 **then**
10: **Start the genetic evaluation** ▷ typically in a parallel fashion
11: $\mathcal{P}$, envs$_{\mathcal{P}}$ $\leftarrow$ *generate-children*(n, θ_a, env)
12: $\mathcal{P}$-fitness $\leftarrow$ *genetic-evaluation*$(\mathcal{P}$, envs$_{\mathcal{P}})$ ▷ possibly storing the samples in the agent's buffer
13: θ_* $\leftarrow$ Select best θ_p according to $\mathcal{P}$-fitness ▷ $\forall \theta_p \in \mathcal{P}$
14: $\theta_a \leftarrow$ Update using θ_*
15: **End the genetic evaluation**
16: **end if**
17: **end for**

(children) applying Gaussian noise $\mathcal{G} \sim \mathcal{N}(0, mut_v)$ to the parameter vector: $mut_p\theta_a + \mathcal{G}$, where mut_p is the mutation probability for each weight. In contrast, the mutation function of ERL multiplies the randomly chosen weights by $\mathcal{N}(0, mut_v)$, i.e., $mut_p\theta_a\mathcal{G}$. Such mutation operator acts in a similar fashion of a dropout layer, biasing the Deep Reinforcement Learning agent to perform better than the evolutionary population in the long term. In practice, ERL mutates 10% of the network weights in each episode (or epoch), multiplying them by $\mathcal{N}(0, 0.1)$ (plus a mutation with standard deviation 10 or a reset mutation in a small percentage of cases). Given that their evolutionary component is running in parallel with the gradient-based agent, we noticed that the weights in the population tend to 0, hence causing a detrimental behavior.

- Lines 12-13: the population and a copy of θ_a are then independently tested over a set of evaluation episodes that shares the same goals to find the overall best performing individual θ_* based on the fitness.[1] In this phase, we can also store a portion of diverse experiences in the memory buffer of drl_a to further exploit the diversity of the population-based component.
- Line 14: if the selected genome belongs to one of the children, θ_a are updated towards the mutated version, and the training phase continues with the new weights. In contrast, if the best score belongs to drl_a, the training phase running in parallel continues.

[1] The fitness definition is domain-specific; in the case of navigation, it is computed as the number of targets the agent reaches over the evaluation episodes.

Algorithm 5 *generate-children*(n, θ_a, env)

$\mathcal{P} \leftarrow$ Initialize with $n+1$ copies of θ_a
envs$_\mathcal{P} \leftarrow$ Initialize with $n+1$ copies of env
for each $\theta_p \in |\mathcal{P}| - 1$ **do**
  Randomly sample a mut_p percentage of weights from θ_p
  for each $w \in mut_p \theta_p$ **do**
    $w \leftarrow w + \mathcal{G}$ ▷ where $\mathcal{G} \sim \mathcal{N}(0, mut_v)$
  end for
end for
return $\mathcal{P}$, envs$_\mathcal{P}$

Since the evaluation does not require any interaction among the population, we instantiate an independent copy of the environment for each individual in a separate thread and test them in parallel. The parallel computation drastically reduces the overhead for the drl_a training process.[2] Hence, our approach has both the advantage of searching for a better-performing policy exploiting mutations that resembles noisy exploration (Fortunato et al., 2017), and enriching the memory buffer with new diversified experiences. As detailed in the following empirical evaluation, our genetic evaluation strategy leads to better policies significantly reducing training time, with Supe-RL resulting approximately two times faster than ERL in the same scenario.

Crucially, in contrast to previous combined approaches, Supe-RL based algorithms are designed to improve the performance of drl_a as our combination strategy does not bias the choice of the better-performing children in the long term. Moreover, we do not transfer any information in the worst-case scenario where the gradient-based agent is always the best genome in the genetic evaluation. Hence a Supe-RL based training will match the performance of the baseline Deep Reinforcement Learning algorithm.

Limitations: We note that Supe-RL and previous combined approaches are specially designed for training in simulation, where it is possible to parallelize the learning process. Hence, when training on a real scenario, it is not generally possible to evaluate the population in parallel, and combined approaches could significantly increase the training time. However, having access to a simulation environment is a common assumption in Deep Reinforcement Learning literature, where significant results have been achieved mainly using simulation and transferring the policy on real platforms (Juliani et al., 2018; Zhao et al., 2020; Ding et al., 2020; OpenAI et al., 2019).

7.2.1 Methodology

Our practical implementations of Supe-RL consider a classic gradient-free Genetic Algorithm (Montana and Davis, 1989), described in Section 2.2.2, with two gradient-based approaches. In particular, we evaluated a variety of different Deep Reinforcement Learning algorithms that work in both discrete and continuous action spaces. Among Rainbow, PPO, DDPG, and TD3, we chose the best-performing ones: (i) Rainbow as a value-based algorithm for the discrete indoor navigation, and (ii) PPO as policy-gradient (or actor-critic) one for the continuous aquatic navigation. We refer to the resultant combinations as GRainbow and GPPO, respectively.

[2] The multi-thread nature of the Unity game engine makes this parallel testing phase straightforward and efficient.

Value-based Genetic Soft Updates

The genetic evaluation of a value-based agent presents additional challenges due to the typical instability of the training approaches. Such brittle convergence and the poor scalability on high-dimensional action spaces are known drawbacks of DQN-based algorithms. Nonetheless, we remark that recent works show the benefits of this family of approaches solutions in these challenging settings (Tavakoli et al., 2018; Zahavy et al., 2019; Dulac-Arnold et al., 2015; de Wiele et al., 2020), further motivating the requirement for a combined approach that is compatible with value-based Deep Reinforcement Learning.

We evaluated different methodologies to update the gradient-based agent with the better performing child:

1. The first method substitutes θ_a with θ_*, approaching the new set of weights on the target network via Polyak averaging (or soft update, according to the literature (Lillicrap et al., 2016)). We tried different settings for the target network, but a soft update of 10% of the weights showed us the best performance. However, this method leads to an unusual optimizer choice, which is crucial considering the high volatility of value-based algorithms. In particular, we note that the well-known Adam (Diederik P. Kingma, 2015) is widely considered in the literature due to the self-adaptable learning rate that typically requires minimal hyper-parameter tuning. However, after the switch of drl_a, Adam leads to a marginal performance drop. We motivate this as the learning rate of Adam at the epoch of the switch is "adjusted" for the old θ_a, hence requiring additional training epochs to embrace the new, better-performing agent. This first Rainbow implementation of Supe-RL, which we refer to as GRainbow, requires careful tuning of an SGD optimizer. Nonetheless, results in Section 7.4 show that GRainbow outperforms both Rainbow and the GA components.

2. Given the main limitation of GRainbow in hand-tuning SGD, which requires several trials, we considered Tessler et al. (2019) that improves stability by copying the agent model to the target, only when the former performs better than the latter. Hence, we flipped the GRainbow update approach by soft updating drl_a towards the best genome and then switching the target network with such genome (which guarantees better performance on the target). This simple trick resulted in a more stable switch operation, enabling us to use Adam and improve the performance of GRainbow.

3. Finally, a natural improvement of the previous methods considers employing the soft update technique for both networks. We refer to the resultant combination as Soft GRainbow (SGRainbow). In more detail, we soft update both the agent's models towards the weights of the best performing child, smoothing the transition of the networks' weights towards its better-mutated version. The update rule for the drl_a networks is thus:

$$\begin{aligned} \theta_a &= \beta\theta_* + (1-\beta)\theta_a \\ \theta'_a &= \beta\theta_* + (1-\beta)\theta'_a \end{aligned} \tag{7.1}$$

where we recall that θ_a, θ'_a are the weights of the agent's network and target network, and β is the update factor of the Polyak averaging. We consider these

slight periodical changes in the drl_a weights as a gradient step towards a nearby set of parameters that performs better. Hence, this switching strategy represents the core mechanism that allows Supe-RL to work and improve the performance of value-based Deep Reinforcement Learning while also reducing the variance across different runs.

Therefore, a useful side message is that using Stochastic Gradient Descent seems a better alternative than Adam when drl_a "jumps" due to target network instabilities. At the same time, Adam works better if the transition of the target network is more stable.

Policy-gradient Genetic Soft Updates

In contrast to value-based Supe-RL implementations, the better convergence properties of policy-gradient (and actor-critic) Deep Reinforcement Learning algorithms do not lead to any particular issue related to the optimizer's choice. Hence, we considered the soft update strategy of SGRainbow. Trivially, we do not use the population visited samples in the update of GPPO, due to the on-policy nature of the baseline PPO algorithm.

7.3 Experiments

We evaluate our framework in six different environments to provide a comprehensive overview of its performance over different scenarios. In detail, we use the discrete action space indoor navigation for GRainbow (and SGRainbow), the continuous action space aquatic navigation, and four benchmark locomotion tasks (Brockman et al., 2016; Todorov et al., 2012) for GPPO. We compare the gradient-free and gradient-based baselines (i.e., Rainbow, PPO, GA) and ERL, the most closely related work. Moreover, we will also present several ablation experiments to confirm the superior performance of Supe-RL and to analyze its core components (e.g., whether to use or not the samples from the genetic evaluations, the effects of different values for β).

Training Setup: We consider the original authors' implementations for the gradient-free component of the algorithms over which we compare here and in the following additional experiments: ERL, PDERL, CEM-RL (Khadka and Tumer, 2018; Bodnar, 2020; Pourchot and Sigaud, 2019). In particular, the results of prior approaches with value-based Deep Reinforcement Learning have been discussed in the previous chapter. Hence, this section presents their performance in the continuous action space tasks, using PPO as the gradient-based component (as it showed better performance in our preliminary evaluation).

Experimental Setup: Following the experiments of previous chapters, data are collected on an RTX 2080, using the hyper-parameters of the original authors' implementations for ERL, PDERL, CEM-RL, and ten independent runs with different random seeds. Concerning our baselines and Supe-RL based approaches, we consider the following hyper-parameters:[3]

- Genetic Evaluation and GA: we use the same genetic hyper-parameters for the baseline Genetic Algorithm, and the gradient-free component of GRainbow, SGRainbow, and GPPO. In particular, the population size is $n = 10$,

[3]For a fair comparison, the baselines (i.e., Rainbow, PPO, and GA) use the same hyper-parameters when trained alone and in the combined approaches.

and the number of epochs in the evaluation to compute the fitness ranges from 10 to 20 across tasks. The mutation probability of a network weight is $mut_p = \{0.75, 0.4, 0.1\}$ (based on the current success rate of the drl_a) and the mutation value is $mut_v = 0.1$.

- GRainbow, SGRainbow, and Rainbow: here, we discuss the relevant hyper-parameters for GRainbow and SGRainbow, referring to Hessel et al. (2018) for further details. Based on preliminary evaluations, we use a PER (Schaul et al., 2016) memory buffer with size 30000, batches of 64, with the original priority decaying parameters. The soft update of the target network (Lillicrap et al., 2016) for Rainbow uses $\beta = 0.01$. Furthermore, given the well-structured reward function, we notice that a simple ϵ-greedy exploration strategy with a decay of 0.99 for each epoch led to a faster training phase compared to using noisy exploration (Fortunato et al., 2017). Finally, a genetic evaluation collects an average of 30000 total samples; adding a random 10% of these interactions in the agent's priority buffer showed improved performance.
- GPPO, ERL, PDERL, CEM-RL, and PPO: we consider the clipped PPO implementation recommended in the authors' work (Schulman et al., 2017), to which we remind the interested reader for further details. We use a small buffer of 256 samples with mini-batches of size 64 for the on-policy gradient-based component, while we use the original algorithm parameters for ERL, PDERL, CEM-RL.

7.4 Empirical Evaluation

The goal of our empirical evaluation is to investigate whether Supe-RL based approaches combine the benefits of a Genetic Algorithm with both value-based and policy-gradient (or actor-critic) Deep Reinforcement Learning while maintaining minimal overhead for the training.

All the trained models (except for the GA and the previous value-based combined approaches) can navigate generalizing: starting and target position and velocity. Furthermore, the lidar allows navigating in unknown environments with different obstacles, while the boat maintains similar performance in different wave conditions. We consider both the average success rate and reward as main evaluation metrics in our Unity environments. To further confirm the beneficial effects of combined approaches, we also show the average number of steps and perform an additional evaluation in the testing scenarios described in Section 3.4.1, 3.4.2. In contrast, the only reward is used in locomotion benchmarks, which is referred to with the general term *Performance* in the literature (Lillicrap et al., 2016; Schulman et al., 2017; Fujimoto et al., 2018; Haarnoja et al., 2018).

Indoor Mapless Navigation: In the discrete action space task, we compare the GA, Rainbow, GRainbow, and SGRainbow, referring to Section 6.4 for the results of the value-based implementations of ERL and PDERL.

Figure 7.2 shows the average success rate on the left and the average reward on the right. In particular, similarly to prior combined approaches, the standalone Genetic Algorithm can not cope with the complexity of the task, where the algorithm needs to generalize the navigation skills while exploiting the laser values to avoid obstacles. In contrast, our periodical genetic evaluation with a soft update strategy that simulates

a gradient step towards a better policy shows promising results. In more detail, we note that the naive GRainbow implementation with the Stochastic Gradient Descent optimizer outperforms Rainbow (i.e., $\approx 80\%$ of successes over $\approx 60\%$). Moreover, the soft genetic update of SGRainbow further improves the performance while reducing the variance, reaching $\approx 90\%$ successes in about 2000 epochs. Such results correspond to 60 minutes of training with our computational hardware. In contrast, Rainbow and GRainbow reached $\approx 80\%$ and $\approx 60\%$ successes in similar training time.

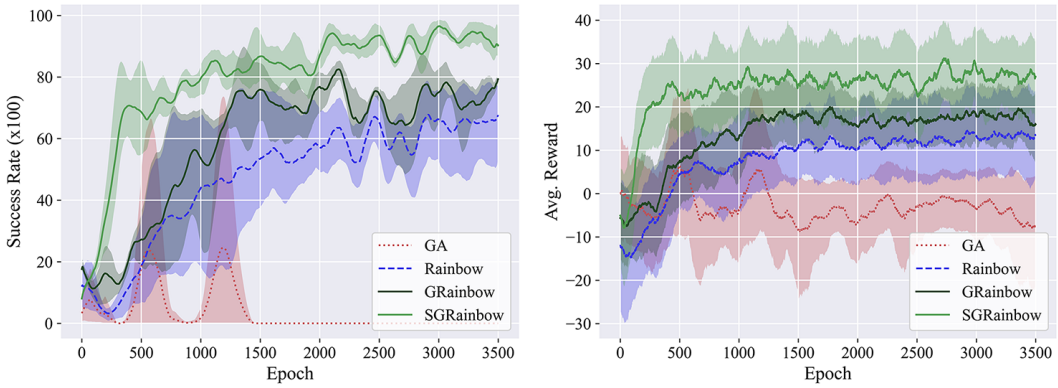

FIGURE 7.2: Average success rate (left) and average reward (right) for GA, Rainbow, GRainbow, SGRainbow in the discrete action space indoor navigation environment.

Furthermore, Figure 7.3 shows the average number of steps performed by SGRainbow and Rainbow during the training. These results confirm the superior performance of SGRainbow in learning more efficient navigation skills, as it achieves higher returns performing shorter paths (i.e., fewer steps).

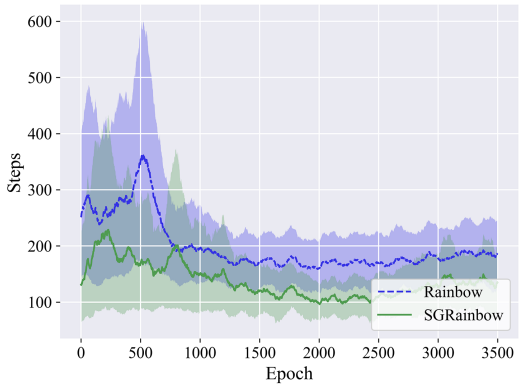

FIGURE 7.3: Average number of steps for Rainbow and SGRainbow during the training.

To further confirm the superior performance of SGRainbow over Rainbow and GRainbow (which are the only trained policies able to navigate in the environment), we performed an additional experiment in the TurtleBot3 testing scenario that presents previously unseen obstacles. Table 7.1 reports the average reward, steps, and time

(in seconds) of this evaluation. For a fair comparison, we chose a set of targets reachable by every model. Although these results confirm the superior performance of SGRainbow, we note that rewards are similar (as the agents navigate towards the target, collecting positive rewards). At the same time, time and number of steps differ significantly ($\approx 31\%$ and $\approx 37.5\%$ for SGRainbow over Rainbow).

TABLE 7.1: Performance of Supe-RL value-based approaches in the TurtleBot3 testing scenario.

Algorithm	Avg. Reward	Avg. Steps	Avg. Time (s)
Rainbow	36.2	370	42
GRainbow	37.8	303	34
SGRainbow	39.1	269	29

Outdoor Aquatic Navigation: In the continuous action space task, we compare the GA, PPO, and its combined versions, ERL-PPO and GPPO.

Figure 7.4 shows the average success rate on the left and the average reward on the right. In particular, the Supe-RL based algorithm, GPPO, offers better performances considering our evaluation metrics also in the continuous action space domain. In more detail, GPPO reaches over $\approx 98\%$ of average success rate in about 1300 epochs that correspond to 110 minutes of training, while PPO, similarly to ERL-PPO, was able to reach $\approx 82\%$ of average success rate in ≈ 1700 epochs (160 and 210 minutes of training, respectively). However, we note that Figure 7.4 presents a significant variance across the runs with different seeds. A more detailed analysis of the collected metrics revealed an initialization seed that resulted in pathological performance for the majority of the algorithms. We will investigate this in the following sections.

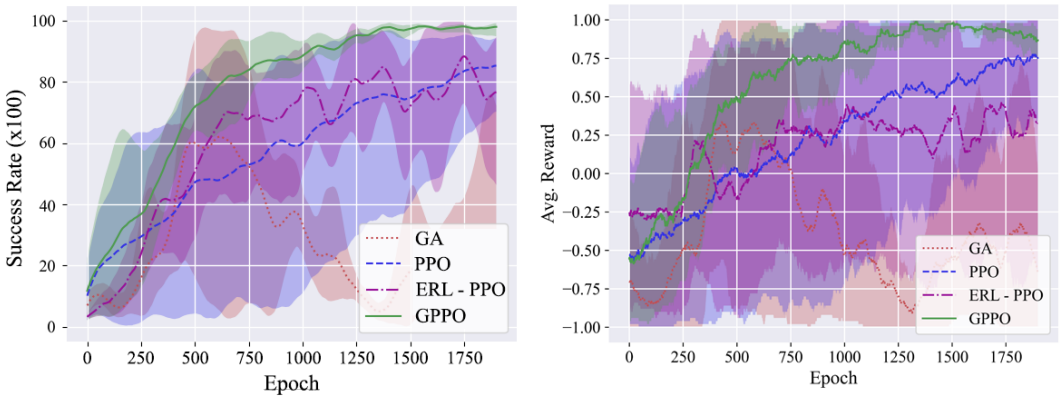

FIGURE 7.4: Average success rate (left) and average reward (right) for GA, PPO, ERL-PPO, GPPO in the continuous action space outdoor aquatic navigation environment.

Furthermore, Figure 7.5 shows the average number of steps performed by GPPO and PPO, which confirms the superior performance of GPPO similarly to the indoor navigation evaluation, where fewer steps translates into shorter paths and higher rewards.

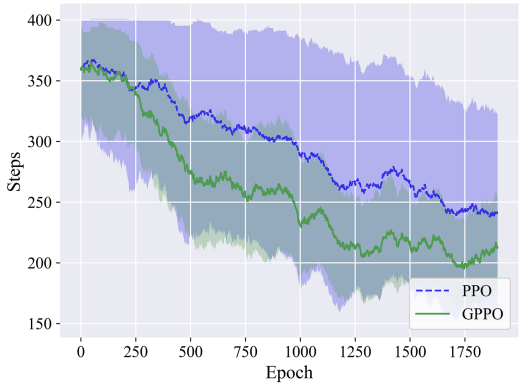

FIGURE 7.5: Average number of steps for PPO and GPPO during the training.

Finally, Table 7.2 reports the performance of PPO, ERL-PPO, and GPPO (i.e., the algorithms that achieved good navigation skills) in the previously unseen testing scenario. Crucially, GPPO confirms its superior performance also in combination with policy-gradient (or actor-critic) algorithms.

TABLE 7.2: Performance of Supe-RL policy-gradient (actor-critic) approaches in the aquatic testing scenario.

Algorithm	**Avg. Reward**	**Avg. Steps**	**Avg. Time (s)**
PPO	4.5	98	27
ERL-PPO	4.6	96	27
GPPO	4.8	89	22

7.4.1 Robustness over Detrimental Initialization

A more detailed analysis of our results reveals that the high variance of GA, PPO, and ERL in the aquatic navigation task is due to a specific seed that results in a pathological performance of the approach. Moreover, such a problem also occurs in the indoor navigation task, but on a smaller scale.

In detail, Figure 7.6 reports the average success rate (for the aquatic navigation on the left and the indoor task on the right) of the only detrimental initialization seed that causes GA, Rainbow, PPO, and ERL to pathological performance (compared to the other experiments with different seeds). Crucially, Supe-RL based approaches (i.e., GPPO, GRainbow, and SGRainbow) do not present poor performance, reaching similar results to the other training phases.

This analysis is not as statistically relevant as previous results because it is not averaged over multiple seeds. However, we believe that this is an essential aspect of our framework. For this reason, the next chapter employs Formal Verification to provide guarantees on the improvements of Supe-RL in the context of safety. Moreover, for a

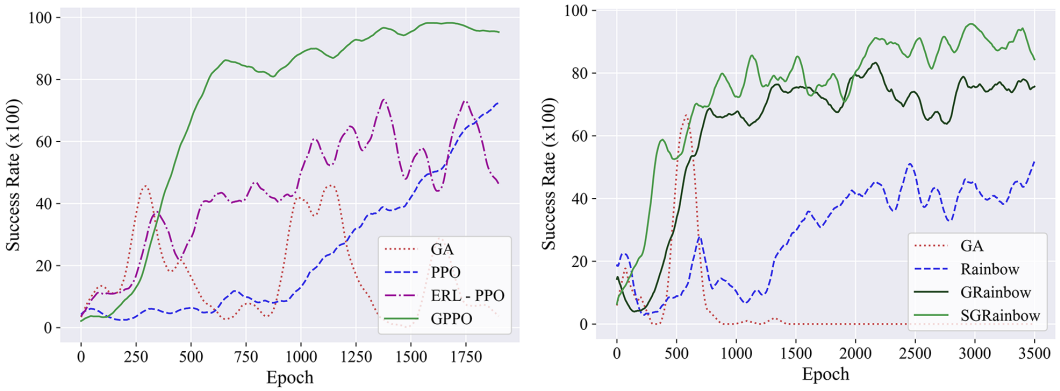

FIGURE 7.6: Average success rate considering the training phase of the only detrimental initialization seed in the aquatic navigation task (left) and in the indoor one (right).

more comprehensive overview of our evaluations in the robotic tasks, Figure 7.7 reports the average success rate for all the experiments without such detrimental seed, where Supe-RL based algorithms still outperform all the considered baselines.

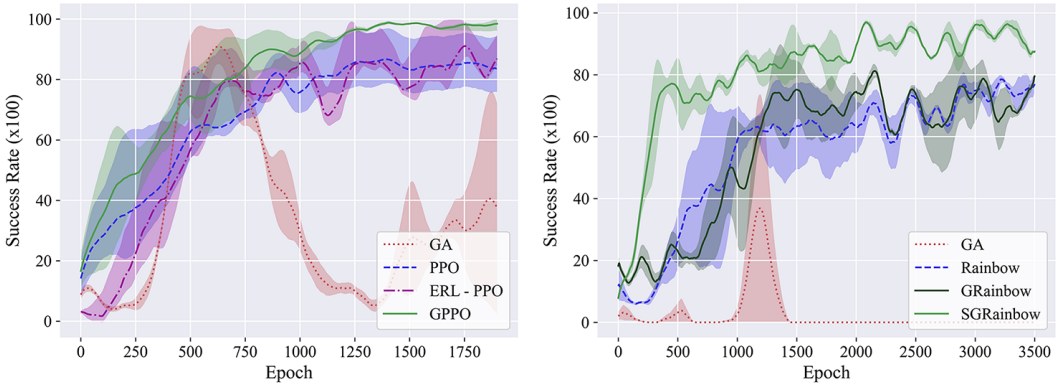

FIGURE 7.7: Average success rate considering all the training phases without the detrimental initialization seed in the aquatic navigation task (left) and in the indoor one (right).

7.4.2 Ablation Studies and Additional Experiments

For a broader evaluation, we present a set of additional and ablation experiments to test our claims on the superior performance of Supe-RL based approaches.

Value-based Experiments

We begin with the value-based implementation, showing the performance of GRainbow with SGD and Adam and the performance difference in the tuning of the soft update parameter β for SGRainbow. We also present an ablation experiment on the influence of the population samples in the buffer of the Deep Reinforcement Learning agent.

Evaluating Adam and SGD: In Section 7.2.1 we discussed two different implementations for GRainbow that differ in the soft update strategy and allow us to consider the Adam optimizer over SGD, which typically requires careful tuning. Figure 7.8 on the left shows the performance difference between GRainbow with Adam, which outperforms the manually tuned version with Stochastic Gradient Descent.

Tuning the Soft Update: We performed multiple evaluations over different seeds to find the best value for the soft update parameter β that regulates the flow of information from the best set of weights in the population to the gradient-based agent. Figure 7.8 in the middle clearly shows that SGRainbow outperforms the naive GRainbow, and reports the average success rate of our experiments with $\beta = \{0.15, 0.3, 0.6\}$.

Using Population Samples: Given our best performing value-based Supe-RL algorithm (i.e., SGRainbow with $\beta = 0.3$), we performed an ablation experiment to test the influence of introducing the samples collected in the genetic evaluation phase into the Deep Reinforcement Learning agent's memory buffer. In particular, Figure 7.8 on the right shows a non-negligible performance improvement when storing part of the population samples into the memory buffer.

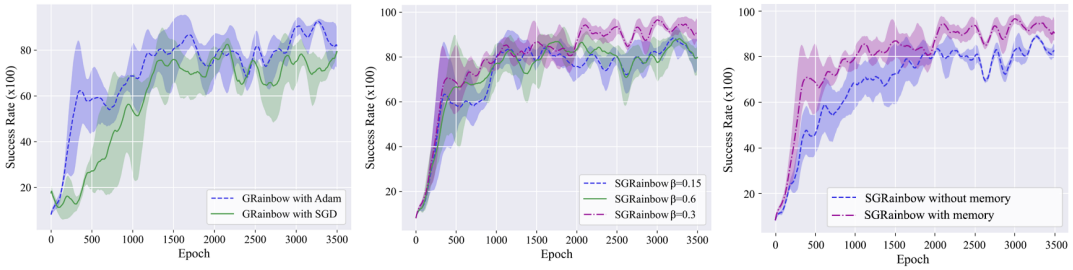

FIGURE 7.8: Average success rate in the additional value-based Supe-RL experiments: SGD and Adam GRainbow (left). Effects on the soft update factor tuning for SGRainbow (middle). Ablation on introducing part of the population visited samples in agent's memory buffer (right).

Policy-gradient Experiments

In this section, we show a further comparison of GPPO in standard benchmarks environments and over CEM-RL (Pourchot and Sigaud, 2019), a distributed combined approach.

Evaluation in Locomotion Benchmarks: We performed additional experiments on Reacher-v2, HalfCheetah-v2, Hopper-v2, and Ant-v2 MuJoCo locomotion tasks (Brockman et al., 2016; Todorov et al., 2012) with GPPO and ERL. These environments consist in learning locomotion behaviors that allow the different simulated agents to move forward, receiving a positive reward signal.

For a fair comparison, we consider the data collection specifics detailed in (Khadka and Tumer, 2018). Our ERL implementation with PPO returned comparable results to the original ones presented in (Pourchot and Sigaud, 2019; Khadka et al., 2019; Khadka and Tumer, 2018). Crucially, Figure 7.9 shows that our GPPO offers comparable or better performance across all the considered tasks. Furthermore, results in the Hopper

environment further highlights the detrimental behavior of ERL, which we discussed in the previous chapter.

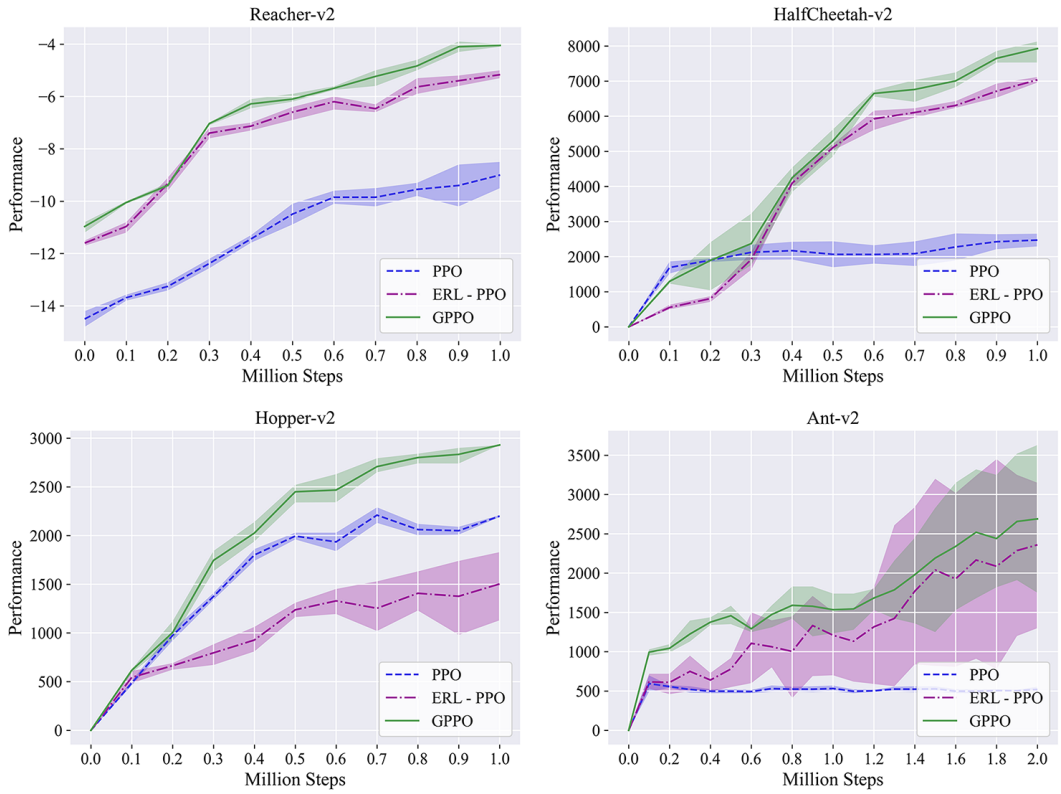

FIGURE 7.9: Performance of PPO, ERL-PPO and GPPO in the MuJoCo locomotion benchmarks: Reacher-v2, HalfCheetah-v2, Hopper-v2, and Ant-v2.

Comparison with a Combined Distributed Approach: As detailed in the previous section, the recent field of combined approaches presents distributed solutions that use a portfolio of gradient-based Deep Reinforcement Learning learners, such as CERL (Khadka et al., 2019) and CEM-RL (Pourchot and Sigaud, 2019). Intuitively, this results in significant overhead in the training process.[4] Nonetheless, we compare GPPO with a PPO version of CEM-RL (CEM-PPO) due to its superior performance over other combined approaches. In contrast, Supe-RL based approaches employ one DRL learner, and the population is only used for policy evaluation.

Figures 7.10 shows the average success rate and training time in the continuous aquatic navigation scenario. As expected, CEM-PPO required significantly more time for the training (250 minutes over 110). In particular, CEM-PPO reaches a 98% success rate (the performance of GPPO at epoch 1200) in approximately 600 episodes. However, it required 125% more wall-clock time with respect to the time required by GPPO to reach a similar performance.

[4]Note that the CEM-RL authors also confirm the significant overhead in Sec. 5.2.2 of the original paper (Pourchot and Sigaud, 2019), which states that their evaluations are on limited timesteps, due to computational demands.

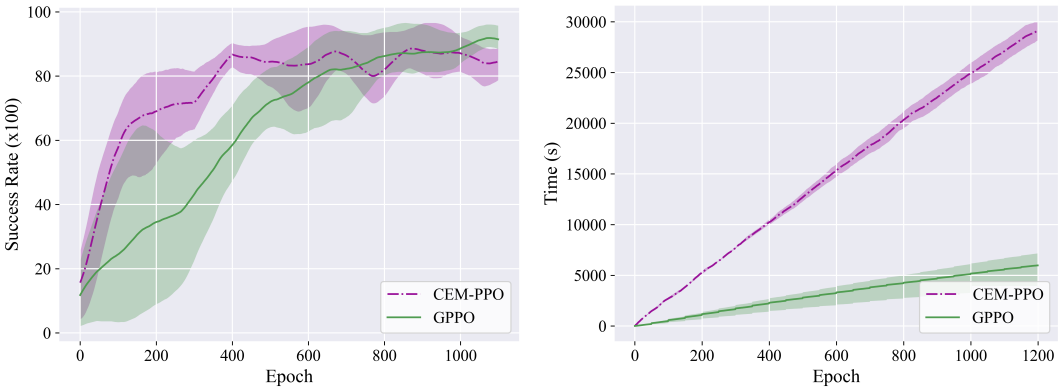

FIGURE 7.10: Average success rate (left) and training time (right) for GPPO and CEM-PPO in the continuous action space outdoor aquatic navigation environment.

7.5 Discussion

This chapter presents our combined framework that exploits the robustness of population-based Genetic Algorithm to improve value-based and policy-gradient Deep Reinforcement Learning agents. We evaluated Supe-RL based algorithms in our navigation scenarios using state-of-the-art algorithms as gradient-based baselines (i.e., Rainbow and PPO). Our extensive empirical evaluation, along with the ablation studies, shows that Supe-RL significantly outperforms DRL baselines (Rainbow and PPO), the GA, and other recent combined approaches, which also showed poor performance when combined with value-based Deep Reinforcement Learning. Crucially, Supe-RL is the first framework that successfully combines Genetic Algorithms and DRL in the field of value-based methods.

Chapter 8

Global Dueling Q-Learning

So far, we have discussed problems related to exploration in single-agent environments, where the task complexity derives from the dynamics of the environment and its high-dimensional observation and action spaces. In contrast, Multi-Agent (Deep) Reinforcement Learning studies how multiple agents interact in a shared environment. Hence, it is the branch of RL that studies how agents interact with the environment and with each other others, which can either cooperate, compete, or collectively act to accomplish a specific task.

In this chapter, we restrict our attention to cooperative settings, where all the agents share a common goal and cooperate to achieve it. In particular, we discuss how current MARL algorithms hinder exploration, proposing a different perspective on the agents' networks architecture and training schema to overcome such limitations.

8.1 Introduction

Multi-Agent (Deep) Reinforcement Learning (Tuyls and Weiss, 2012) considers multiple learning agents (or robots) that have to optimize either an individual or a joint reward signal accumulated over time. In particular, we focus on multi-agent navigation as it recently gained attention due to the wide range of applicability to real-world scenarios (e.g., search (Baxter et al., 2007) and self-driving (Wang et al., 2018a)). Furthermore, we can leverage our insights from the previous chapters on robotic navigation.

Multi-agent navigation has been previously considered as a cooperative MARL problem and requires the learning of an efficient collision avoidance policy (Long et al., 2018). Among the different approaches (Tuyls and Weiss, 2012; Oroojlooyjadid and Hajinezhad, 2019; L. Busoniu and Schutter, 2008), *centralized learning* aims at reducing the task to a single-agent problem, which is solvable with standard DRL algorithms. However, these centralized solutions require comprehensive knowledge about all the agents' observations (e.g., positions, velocities) and their workspace (e.g., a grid map) that form a joint observation. However, centralized methods do not scale well within the number of agents and consistently fail in practice due to the "lazy" agent problem (Sunehag et al., 2018). Intuitively the lazy agent consists of an individual discouraged from learning the policy as its exploration can hinder other successful agents.

Conversely, *independent learning* consists of individual agents that have to face a non-stationary problem. Such an issue is due to the dynamics of the environment

that changes within the other unobserved policies. In practice, we address the non-stationarity by using recurrent units such as Long Short-Term Memory (LSTM) (Sepp Hochreiter, 1997), but the lack of shared information does not favor cooperative behaviors. However, independent learning is typically used as a baseline as it represents a good-performing yet straightforward solution for a variety of Multi-Agent (Deep) Reinforcement Learning tasks (Lowe et al., 2017). Both the scalability and the non-stationarity issues have been considered by the recent Centralized Training Decentralized Execution (CTDE) paradigm. In particular, CTDE has been used to design value-based MARL algorithms (Sunehag et al., 2018; Rashid et al., 2018; Son et al., 2019) that showed state-of-the-art results on several multi-agent tasks (e.g., multi-agent particle envs (Lowe et al., 2017)). In more detail, CTDE involves training agents' policies that use global information in a centralized way and rely only on local observations for the action selection, enabling decentralized execution.

In more detail, to address the combinatorial action space and the size of the joint observations, VDN (Sunehag et al., 2018), QMIX (Rashid et al., 2018), Weighted QMIX (Rashid et al., 2020) and QTRAN (Son et al., 2019) propose different mechanisms to factor a joint (or global) action-value function Q_G. However, these approaches introduce structural constraints (e.g., additivity, monotonicity) to ensure that Q_G factors into the agent individuals Q_i-values (with $i = \{1, \ldots, n\}$, where n is the number of agents) that are used for the decentralized action selection process. Such constraints severely limit the joint action-value function class that VDN and QMIX can represent, while QTRAN showed poor practical performance although its robust theoretical factorization guarantees (Son et al., 2019).

To address these limitations, we propose a value-based Centralized Training Decentralized Execution approach, which we refer to as multi-agent Global Dueling Q-learning (GDQ). In contrast to prior work, GDQ exploits the state-values V_i of each agent i to estimate a joint state-value for the system, which is used in the update of the agents' weights.[1] In more detail, GDQ centralizes the individual state-values of the agents to compute a joint state-value V_G using a standard value-based Deep Reinforcement Learning algorithm (van Hasselt et al., 2016), which learns using the cumulative reward signal. A high-level overview of GDQ's architecture is depicted in Figure 8.1.[2] Our goal is to avoid the issues in the factorization of Q_G, separating each agent's state-value V_i and advantage A_i functions. Such decomposition is possible as Dueling DQN (Wang et al., 2016) showed that the combination of V_i and A_i results in the agent's action-value Q_i, from which an agent samples its action. Intuitively, the state-value function measures how good it is to be in a state, whether the advantages are the relative importance of the actions. Such values are thus combined to form the per-agent action value, i.e., $Q_i = V_i + A_i$, which is used to choose a particular action in the state. Hence, we use the agents' state values as input for the centralized value-based model that learns the joint state-value V_G, which replaces V_i in the estimation of the temporal difference target of each agent. It also has the advantage of addressing the overestimation problem of DQN-based algorithms that typically requires a target network (van Hasselt et al., 2016). Our idea is to inject information on

[1] In these high-level descriptions, we omit the state s and action a parameters for notation simplicity.

[2] Note that in a practical multi-agent application, the input for each agent is its recent history, and the hidden layers contain (at least) one recurrent layer to deal with the partially observable and non-stationary environment.

the global environment state in each agent update rule, such that the agents optimize their behaviors while considering the state of the system.

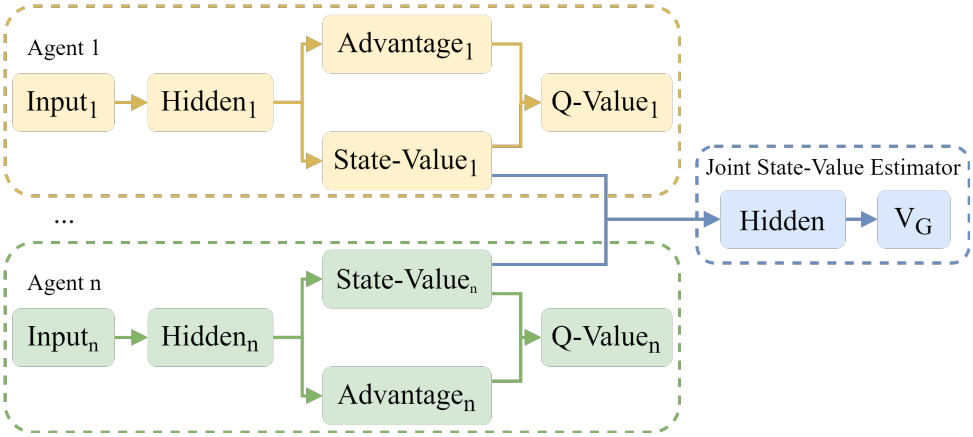

FIGURE 8.1: High-level overview of GDQ. Each agent computes its Q_i-values for a decentralized execution. Agents' Dueling Networks compute the state-values V_i that are used as input for the joint state-value estimator.

We first evaluate the performance of GDQ on the benchmark *Cooperative navigation* task of the multi-agent particle envs to confirm the superior performance of GDQ over prior value-based Centralized Training Decentralized Execution algorithms (i.e., VDN and QMIX). Hence, we extend the Turtlebot3 Unity environment of Section 3.4.1 to the multi-agent setting to highlight the scalability and the beneficial effects of GDQ in increasingly complex practical situations.

8.2 Preliminaries and Related Work

Similar to previous CTDE approaches, we describe the multi-agent navigation scenario as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) (Oliehoek and Amato, 2016). In detail, a Dec-POMDP extends a classic POMDP, formalized in Section 2.2, for scenarios with a decentralized decision-making process of a multitude of agents. Hence, this extension offers an important framework for cooperative sequential decision making under uncertainty, such as the task we consider for our evaluation.

8.2.1 Multi-agent Navigation

We previously discussed the importance of robotic navigation in recent Deep Reinforcement Learning literature (Tai et al., 2017; Zhang et al., 2017; Kretzschmar et al., 2016). Given its wide range of applications, such as search and rescue (Baxter et al., 2007), or collision avoidance solutions (Long et al., 2018), this class of problems has been naturally extended to the Multi-Agent (Deep) Reinforcement Learning domain.

Similar to prior CTDE baselines (Sunehag et al., 2018; Rashid et al., 2018), we remark the beneficial aspects of value-based DRL (which has been exhaustively addressed in previous chapters). Hence, we employ this family of algorithms in the design of GDQ.

8.2.2 Centralized Training Decentralized Execution

Centralized Training Decentralized Execution has recently attracted significant research attention as a core Multi-Agent (Deep) Reinforcement Learning paradigm (Sunehag et al., 2018; Rashid et al., 2018; Son et al., 2019; Lowe et al., 2017). The main idea introduced by the first CTDE approach, the actor-critic Multi-Agent DDPG (MADDPG) (Lowe et al., 2017), is that the individuals learn using centralized information (e.g., Q_G) or the global state (e.g., positions and goals of the other robots) during the training phase. However, the decision-making process of each agent is strictly independent as it is based only on the local action-observation history, guaranteeing a decentralized execution.

Starting from MADDPG (and the actor-critic COMA (Foerster et al., 2018)), multi-agent research shifted its focus to developing CTDE value-based approaches, that aim at satisfying the Individual Global Max (IGM) requirement. In detail, IGM states that the optimal joint action induced from Q_G is equivalent to the collection of the individuals Q_i-values. Formally, this asserts that $Q_G(\boldsymbol{\tau}, \mathbf{a})$ (where $\boldsymbol{\tau}$ is the joint action-observation history, and $\mathbf{a} \equiv [a_i]_{i=1}^n$ is the joint action) is factorizable if and only if exists $[Q_i : \mathcal{T} \times \mathcal{A} \to \mathbb{R}]_{i=1}^n$ such that $\forall \boldsymbol{\tau} \in \mathcal{T}$:

$$\arg\max_{\mathbf{a} \in \mathcal{A}} Q_G(\boldsymbol{\tau}, \mathbf{a}) = \begin{pmatrix} \arg\max_{a_1 \in \mathcal{A}} Q_1(\tau_1, a_1) \\ \vdots \\ \arg\max_{a_n \in \mathcal{A}} Q_n(\tau_n, a_n) \end{pmatrix} \tag{8.1}$$

we can summary Equation 8.1 saying that a global arg max on Q_G returns the same result as a set of arg max on each Q_i, with $i = \{1, \ldots, n\}$.

Two main different factorization strategies for satisfying the IGM have been proposed:

1. *Additivity* by VDN (Sunehag et al., 2018).
2. *Monotonicity* by QMIX (Rashid et al., 2018, 2020).

These factorizations are sufficient for the IGM principle. However, they limit the representation expressiveness of the joint action-value function Q_G (Mahajan et al., 2019). QMIX also considered additional *hypernetworks* (Ha et al., 2017) to add additional global state information to the robot individual observations, introducing overhead. QTRAN (Son et al., 2019) uses a relaxation of the IGM constraint in the form of a linear constraint, which despite the theoretical guarantee, result in poor performance.

In contrast to this research trend that aims at factoring Q_G (and using additional global state information to favor cooperation), GDQ is more closely related to the independent learning paradigm. The reason is that GDQ's agents do not share any additional global observations to favor cooperation. However, they only rely on an additional estimation of a joint state-value V_G, which uses the individuals V as input (i.e., a form of centralized training). Crucially, we use V_G only in the temporal difference target computation of each agent. Hence GDQ is also based on the Centralized Training Decentralized Execution paradigm.

8.3 Global Dueling Q-Learning

In this section, we present the multi-agent Global Dueling Q-learning algorithm. In particular, we discuss the benefits of computing a joint state-value over prior CTDE approaches that factorize the joint action-value to satisfy the IGM, which is the main limitation of such methods.

Multi-Agent Global Dueling Q-learning: GDQ uses a Double DQN (van Hasselt et al., 2016) based algorithm for each agent, enhanced with PER (Schaul et al., 2016), n-step returns (Hessel et al., 2018), and the dueling architecture (Wang et al., 2016). The insights of Dueling Networks represent the foundation of our value-based CTDE approach.

In detail, the agents' Q-networks maintain two streams to represent both the individual state-value function $V(s)$, and the individual advantage function $A(s,a)$.[3] The streams are combined with an aggregation layer to produce the agent's action-value function $Q(s,a)$, which in its simplest form computes the Q-values as follows:

$$Q(s,a) = V(s) + A(s,a) \tag{8.2}$$

The use of a separate representation for the state-value and the action advantages allows to learn whether the considered state s is either valuable or not. Crucially, $V(s)$ does not influence the action selection process of an agent, which is strictly dependent on the advantage values; formally:

$$\arg\max_{a \in \mathcal{A}} Q(s,a) \equiv \arg\max_{a \in \mathcal{A}} A(s,a) \tag{8.3}$$

Furthermore, we note that the action-value decomposition improves the sample efficiency of the training process (Sutton and Barto, 2018; Wang et al., 2016). For example, it is crucial to know which action to take in specific states (e.g., avoiding a collision). However, various actions can be negligible in other states (e.g., where obstacles are far from the agents). This is mainly related to two factors:

1. The direct estimation of the Q-values (as in standard DQN) requires calculating the value of each action at each state, which is a non-negligible overhead when the whole set of possible actions do not affect the environment in a relevant way. For example, in a navigation scenario, we could move to the left or the right only when there is a risk of collision. Otherwise, the choice of action has no major effect on what happens. As detailed by prior work (Wang et al., 2016), this also aggregates similar actions, which further improves the sample efficiency.
2. In a dueling architecture, where the state-value is estimated independently by a separate stream of the network, it is possible to learn $V(s)$ more efficiently as its separate estimation stream updates with every update of the Q-values.

In practice, Equation 8.2 is unidentifiable, in the sense that we can not uniquely recover the state-value and the advantages function from given Q-values. For example, if we multiply the state-value and divide the action advantages by the same constant value, we obtain the same Q-values. This causes poor practical performance and has been

[3] Here we refer only to local state/action information, where s could be the local observation history in a partially observable scenario.

addressed using different aggregation functions to combine the two streams. Hence, we consider the best operator identified in the original work (Wang et al., 2016) for the aggregation layer of each agent; formally:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A(s,a)|} \sum_{a' \in A(s,a)} a' \right) \tag{8.4}$$

where $|A(s,a)|$ is the cardinality of the advantage function's stream, i.e., the number of possible actions.

Hence, given the weights of each agent's network θ_{a_i} and a batch of samples b, where each sample is a tuple (s, a, r, s', V'_G), each agent minimize the following loss function:

$$L(f_{\theta_{a_i}}) = \frac{1}{|b|} * \sum_{k=0}^{|b|} \left[\left(y_{\theta_{a_i}}(r_k, s'_k, V'_{G_k}) - Q_{\theta_{a_i}}(s_k, a_k) \right)^2 \right] \tag{8.5}$$

where V'_G is the estimated joint next state-value, which we describe in the next section.

In more detail, the target is computed by replacing the next state-value, with the one computed by our centralized joint state-value estimator:

$$\begin{aligned} y_{\theta_{a_i}}(r, s', V'_G) &= r + \gamma \ \arg\max_{a \in \mathcal{A}} Q_{\theta_{a_i}}(s', a, V'_G) \\ Q_{\theta a_i}(s', a, V'_G) &= V'_G + A_{\theta_{a_i}}(s', a) \end{aligned} \tag{8.6}$$

As previously discussed, this does not influence the action selection process of an agent as it only depends on the advantage function. However, it serves to inject learned information on the global state value. The other natural contribution of introducing V'_G is that it naturally mitigates the overestimation bias of the standard DQN. Hence, it is possible to remove the requirements of a target network for each agent.

Joint State-Value Estimator: We use a standard Double DQN algorithm to compute the joint state-value V'_G for the agents. This additional estimator takes as input all the agents' next state-values $[V_1(s'), \ldots, V_n(s')]$ computed by their separate stream to learn an estimation of the joint state-value V'_G. The training procedure for this centralized component is the same as the Double DQN algorithm. However, it considers state values instead of action values and the cumulative reward of the agents.

In more detail, a memory buffer stores at the concatenated agents' state-values $\mathbf{v} = [V_1(s), \ldots, V_n(s)]$, the next state-values $\mathbf{v}' = [V'_1(s'), \ldots, V'_n(s')]$, and the agents' cumulative rewards $\mathbf{r} = r_1 + \cdots + r_n$.[4] Hence, given the weights of both the joint state-value estimator θ_c and its target network θ'_c, and a batch b of samples from its memory (where each sample is a tuple $(\mathbf{r}, \mathbf{v}, \mathbf{v}')$, the centralized component aims at minimizing the following loss function:

[4] Intuitively, next state-values require an additional forward step of the agents' networks.

$$L(f_{\theta_c}) = \frac{1}{|b|} * \sum_{k=0}^{|b|} \left[\left(y_{\theta'_c}(\mathbf{r}, \mathbf{v}') - V_{\theta_c}(\mathbf{v}) \right)^2 \right] \tag{8.7}$$

where $V_{\theta_c}(\mathbf{v})$ is computed with a forward pass of the joint state-value estimator on input $\mathbf{v}$, and the target $y_{\theta'_c}$ is:

$$y_{\theta'_c}(\mathbf{r}, \mathbf{v}') = \mathbf{r} + \gamma \, V_{\theta'_c}(\mathbf{v}') \tag{8.8}$$

Although this chapter focuses on cooperative Multi-Agent (Deep) Reinforcement Learning, we believe the decoupled nature of the state-value stream could also address competitive MARL by using a separate joint state-value estimator for each agent group. We intend to explore this direction for future work.

8.4 Experiments

We evaluate GDQ on a preliminary experiment on the multi-agent benchmark Cooperative navigation task of the *multiagent particle envs* suite. Therefore, we present the multi-agent navigation scenario used to test GDQ performance on a growing number of agents.

8.4.1 Multi-Agent Navigation Scenario

We consider a setup similar to the single-robot navigation scenario presented in Section 3.4.1, adapted for the multi-agent setup.

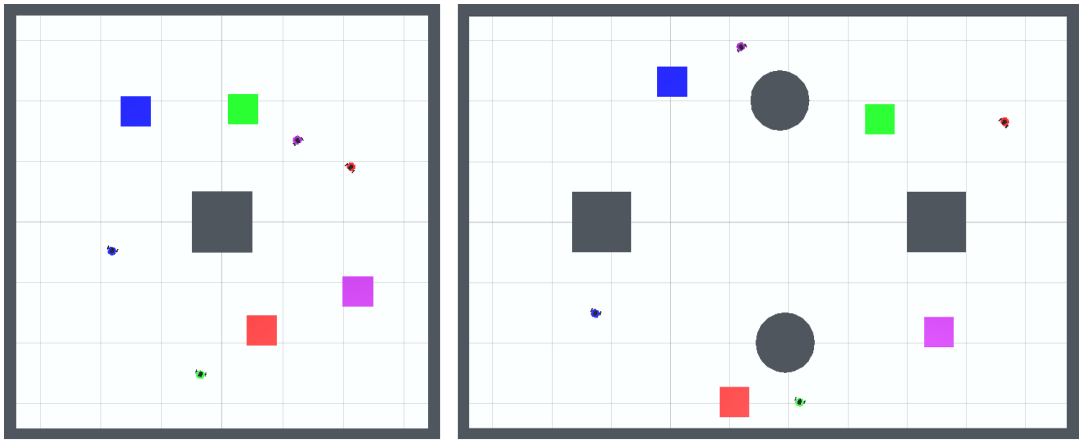

FIGURE 8.2: Training environment for our MARL navigation (left) Testing environment for the trained policies (right). Explanatory view with $n = 4$ robots in different colors.

In more detail, we consider an indoor navigation environment with $n = \{2, 4, 8\}$ robots and fixed obstacles, where each robot has to navigate to its target, avoiding collisions. The decision-making frequency of the robot in both the training and testing environments is set to 20Hz, to reflect the update rate of the equipped LDS-01 lidar sensor. This training environment is depicted in Figure 8.2. To detect collisions among the robots and the obstacles, we used Unity's mesh collider system, applied

to the 3D models of the environment. This allows triggering a collision event when an intersection between the bounds of two or more colliders is detected. The target goals of each randomly spawn in the scenario and are guaranteed to be obstacle-free and with a minimum distance between each other (set to $0.5m$). Each agent receive a separate reward r_t at each timestep t that is structured as follows:

$$r_t = \begin{cases} \mu(d_{t-1} - d_t) - 0.005, \text{ if } d_t > 0.1 \\ 1, \text{ if } d_t <= 0.1 \\ -1, \text{ if a collision is detected} \end{cases} \tag{8.9}$$

In detail, we have two sparse values in case of reaching the target within a threshold distance $d_t = 10cm$ from the robot's goal (i.e., $r_t = 1$), or crashing within the obstacles or other robots (i.e., $r_t = -1$) which terminates an episode resetting the robot to its starting position and generating a new set of goals. A dense component is used during the travel: $\mu(d_{t-1} - d_t) - 0.005$, where d_{t-1}, d_t is euclidean distance between a robot and its goal at two consecutive time steps and μ is a multiplicative factor that is used for normalization. A penalty of 0.005 is applied at each timestep to encourage the robots to perform the shortest path possible while avoiding collisions.

Training Setup: We consider the original authors' implementations for the MARL algorithms over which we compare: VDN (Sunehag et al., 2018), QMIX (Rashid et al., 2018) (as Weighted QMIX (Rashid et al., 2020) achieved comparable performance in our preliminary evaluation), and an Independent Learner approach based on Rainbow (Hessel et al., 2018) (IQL).

Given the partially observable nature of multi-agent navigation, the action-value functions are defined over individual observation histories. Hence we incorporate recurrent units (i.e., an LSTM layer (Sepp Hochreiter, 1997)) in all the agent' networks.

All the considered algorithms share the same input layer structure: 35-sparse laser scans sampled in $[-120, 120]$ degrees in a fixed angle distribution and the individual target position. To cope with the complexity of learning an efficient collision avoidance policy, the output layer considers the Branching Architecture (Tavakoli et al., 2018) to output two streams of Q-values: one for angular velocities $v_{ang} \in [-90, -45, 0, 45, 90]\ deg/s$ and one for linear velocities $v_{lin} \in [0, 0.05, 0.1, 0.15, 0.2]\ m/s$. Hence, following the GDQ architecture, each network maintains three separate streams: one for the advantage of v_{ang}, one for the advantage of v_{lin}, and one to estimate the state-value. The two advantage streams are then combined with the state-value using Equation 8.4.

Similar to the benchmark in Chapter 4, we performed an initial evaluation in the training environment with multiple trials on different network sizes and seeds. The outcome led us to use the network architectures depicted in Figure 8.3: one ReLU hidden layer with 128 neurons, a ReLU LSTM layer of size 64, plus an additional ReLU hidden layer of 64 neurons for each stream. The joint state-value estimator, is a simple feed-forward network with two ReLU hidden layers of 64 neurons each and a Linear output.

Experimental Setup: We collect data on an RTX 2080, using the hyper-parameters of the original authors' implementations for the baselines. Each experiment in the multi-agent navigation (i.e., with $n = \{2, 4, 8\}$ robots) runs over ten independent runs

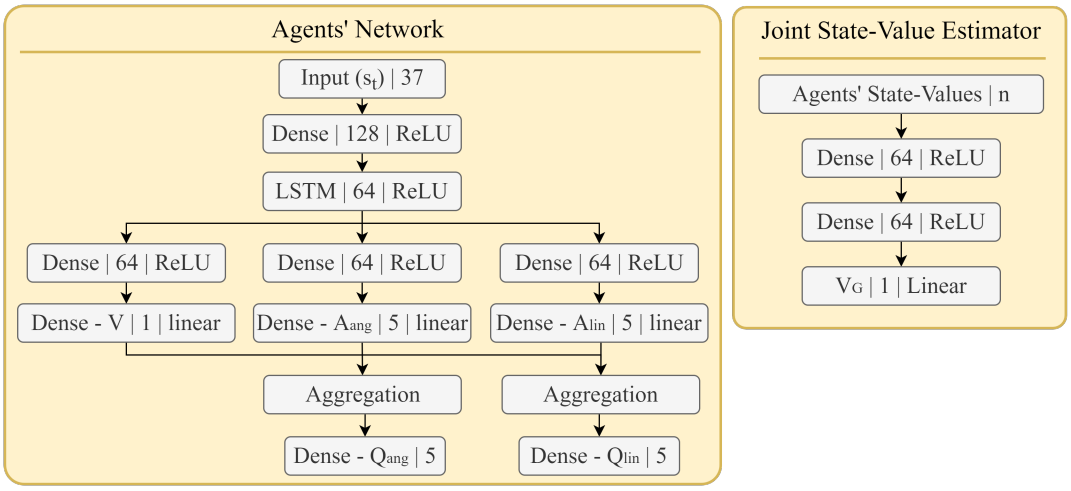

FIGURE 8.3: Agents' network architecture (left). Joint state-value estimator network architecture (right).

with different seeds, for statistical significance of the collected data (Henderson et al., 2018).

8.5 Empirical Evaluation

Our goal is to show the benefits of avoiding structural constraints on the network architectures that limit exploration. Hence we aim at showing the superior performance of GDQ.

To this end, given the multi-agent setting of our scenarios, we consider both the average reward and the average percentage of collisions of the agents as evaluation metrics. Given the navigation nature of the tasks, we also report the average traveled distance, either as actual distance in meters or as the number of steps.

8.5.1 Standard Navigation Benchmark

To evaluate the performance of GDQ over previous similar CTDE baselines, we consider the Cooperative Navigation as a standard benchmark example. In particular, we compare with VDN, QMIX, and IQL based on Rainbow, as Weighted QMIX resulted in comparable performance over QMIX (in our preliminary experiments), and QTRAN typically results in poor performance (Son et al., 2019).

For this benchmark experiment, all the networks share the previous architecture and Adam optimizer with default learning rate (for algorithmic specific hyper-parameters, we refer the interested reader to the original works (Sunehag et al., 2018; Rashid et al., 2018; Hessel et al., 2018) as we used the authors' implementation). In this collaborative problem, three agents should cover the same number of landmarks and are rewarded based on how far each one is from each landmark, while they are penalized if they collide with other agents.

Results in Table 8.1 show the average reward, distance from the landmarks, and collision percentage. The reported metrics confirm the superior performance of GDQ over the considered baselines as it obtains the highest average reward (i.e., minimum

average distance) while also favoring information exchange, resulting in the lowest percentage of collisions. Moreover, IQL obtained good performance despite the simplicity of the approach, while the superior performance of QMIX over VDN and IQL confirm the original authors' results (Rashid et al., 2018).

TABLE 8.1: Average reward, distance from the landmarks, and collision % for IQL, VDN, QMIX, and GDQ.

Algorithm	Avg. Reward	Avg. Distance (m)	Avg. Collisions (%)
IQL	-2.41 ± 0.31	1.39 ± 0.11	34.2 ± 2.2
VDN	-2.45 ± 0.21	1.43 ± 0.13	32.5 ± 1.9
QMIX	-2.23 ± 0.24	1.25 ± 0.09	25.6 ± 1.7
GDQ	-2.06 ± 0.21	1.02 ± 0.10	18.7 ± 1.7

8.5.2 Unity Multi-Agent Navigation

Given the results in the preliminary evaluation in a standard benchmark, the multi-agent navigation task compares the only two best-performing algorithms: QMIX and GDQ. This evaluation investigates whether GDQ can successfully address a robotic task of practical interest while favoring cooperation.

Each experiment with $n = \{2, 4, 8\}$ robots reports the following curves that show mean and standard deviation over the runs, smoothed over 100 epochs. All the metrics are averaged over the n robots. We also test the two MARL algorithms in a previously unseen scenario to evaluate their generalization skills.

Training Results: Figure 8.4 shows the results of GDQ and QMIX, where the robots have to learn how to reach different target positions, avoiding collisions with the fixed obstacles and with each other. In detail, while the results are comparable when considering only two robots, with a growing number of robots, GDQ offers better performances. In more detail, with $n = 4$, the percentage of successes stabilizes at over ≈90% in ≈1700 epochs, i.e., 150 minutes of training. QMIX, in contrast, reaches ≈80% successes in the same epochs that correspond to 190 minutes of training. It naturally follows that GDQ offers better performance even in terms of reward. The evaluation with $n = 8$ confirms the superior performance of GDQ.

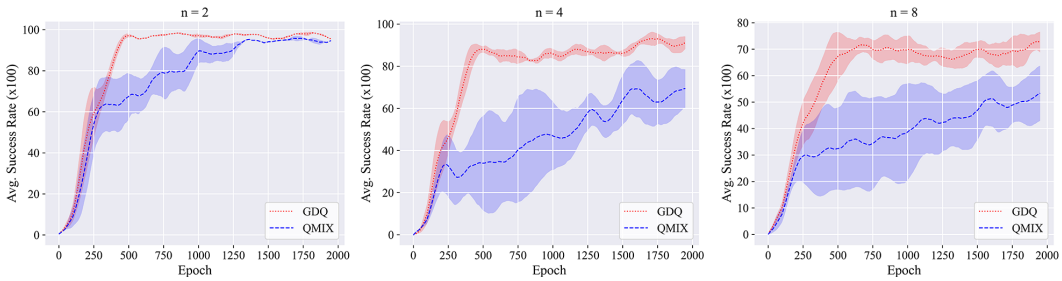

FIGURE 8.4: Average success rate of the GDQ and QMIX algorithms in the multi-agent navigation training phases.

Testing Results: Crucially, after the training phase, the trained models can navigate the environment in a decentralized fashion, exploiting the minimal information in the network's input layer. Moreover, the individuals generalize crucial aspects of robot navigation such as (i) robot starting position, (ii) target position, (iii) velocity. The laser scan-based navigation also allows our robots to navigate previously unknown environments with different obstacles, which is crucial for motion planning. To confirm the successful generalization of our models, we run an additional evaluation with $n = \{4, 8\}$ in a previously unseen scenario, depicted in Figure 8.2 on the right. Results in Table 8.2 confirm the trend highlighted during the training, where GDQ offers superior performance over the baseline in terms of average reward and path length when the same sequence of goals is considered for both algorithms. In this evaluation, we also collect the percentage of collisions detected, which further confirms a better generalization of the task with multiple robots in the case of GDQ.

TABLE 8.2: Average reward, steps and collision percentage in the testing scenario.

Algorithm	n	Reward	Steps	Collisions (%)
QMIX	4	26.3 ± 3.1	295 ± 23	1.4 ± 2.6
GDQ	4	30.1 ± 2.7	242 ± 18	0.8 ± 1.8
QMIX	8	18.2 ± 3.9	361 ± 29	34.2 ± 4.1
GDQ	8	24.3 ± 2.8	285 ± 22	17.3 ± 2.3

8.6 Genetic Global Dueling Q-Learning

In addition to GDQ that avoids structural constraints of prior Centralized Training Decentralized Execution, we also aim at improving exploration for Multi-Agent (Deep) Reinforcement Learning in multi-robot mapless navigation by analyzing the problem from a different perspective. Specifically, in this context, a navigation policy can be decoupled into two (so-called) sub-policies:

1. Navigation skills to reach the target.
2. Collision avoidance behaviors to avoid other robots.

Following relevant literature, both policies could be learned as a single navigation policy (Tai et al., 2017; Zhang et al., 2017), however, we argue that learning the two skills by training a single policy implicitly hinders sample efficiency and exploration and the motivation is straightforward. A collision between robots ends a training epoch because robots should be returned to a non-collision state (i.e., the environment is typically reset). However, the robots could have continued to explore the current path and acquire novel skills that are useful to learn their end goal of reaching the target.

To this end, we propose an Evolutionary Policy Search (EPS), depicted in Figure 8.5, that works on top of existing MARL approaches. Our goal is to bias the current multi-robot policies with basic navigation skills easily acquired with EPS in a single robot environment. Intuitively, improving navigation skills in a stationary environment with a single robot is much simpler than learning them from scratch in a complex

non-stationary environment such as the multi-robot task (and collisions rarely occur, improving the exploration).

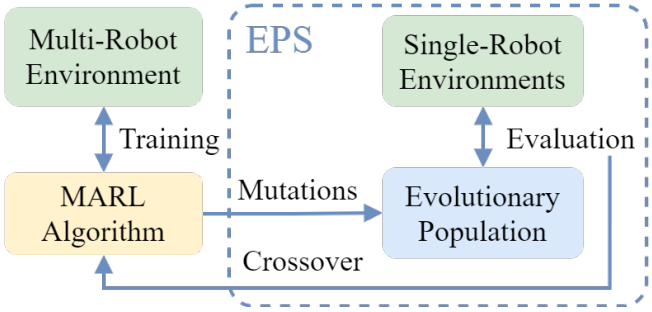

FIGURE 8.5: Overview of our Evolutionary Policy Search.

In more detail, we propose using an evolutionary population that is periodically generated from the robots' policies using mutation operators to explore different regions of the policy space. These mutated versions of the policies are evaluated independently over a set of trials in a single-robot environment. Hence, we select the individual with the highest return, which means better navigation behaviors, and inject its skills into the MARL policies using a crossover operator. The beneficial effects of this information transfer have been highlighted in the previous chapter and by previous combinations of Evolutionary Algorithms and gradient-based Deep Reinforcement Learning (Khadka and Tumer, 2018; Bodnar, 2020). However, these frameworks were limited to single-agent scenarios and designed to improve the overall return. In contrast, EPS facilitates the learning of basic navigation skills online (i.e., during the MARL training) using our decoupled formalization to improve the performance and sample efficiency of existing approaches.

The general flow of our Evolutionary Policy Search is summarized in Algorithm 6:

- Line 1: the chosen Multi-Agent (Deep) Reinforcement Learning baseline runs on the multi-robot task following the algorithm's specifications.
- Lines 2-6: periodically, we leverage Evolutionary Algorithms (Fogel, 2006) and gradient-based mutations (Lehman et al., 2018) to generate a population of mutated versions of the robots' policies.
- Lines 7-8: to this end, we sample a batch b from the memory buffer to compute the per-weight sensitivity λ of the network's outputs over its weights θ_e (more details in the following section). This is used to generate the population of n individuals $\mathcal{P}$ with weights $\theta_{\mathcal{P}}$, to which we add a copy of the original network's weights.
- Line 10: we evaluate $\mathcal{P}$ in a fixed set of epochs on multiple independent instances of a single-robot environment that collect the individual's average reward that represents our fitness score $\mathcal{P}$-fitness.
- Line 11: $\mathcal{P}$-fitness is then used to select the best set of weights θ_*, i.e., the one that achieves higher return:

$$\theta_* = \arg\max_{\theta_p}(\mathcal{P}\text{-fitness}) \tag{8.10}$$

Algorithm 6 Evolutionary Policy Search

Given:

- network with shared weights θ_e at current epoch
- the size n of the population $\mathcal{P}$
- periodicity $e_{\mathcal{P}}$ for the evolutionary policy search
- a running MARL algorithm ▷ e.g., GDQ, IQL

1: During the training of the multi-agent navigation task, the robots proceed according to the chosen algorithm.
2: **if** epoch % $e_s == 0$ **then**
3: **Start the evolutionary search** ▷ typically in a parallel fashion
4: $\mathcal{P} \leftarrow$ Initialize with $n+1$ copies of θ_e ▷ each with weights θ_p
5: $\text{envs}_{\mathcal{P}} \leftarrow$ Initialize with $n+1$ copies of single-robot environments
6: Compute $\mathcal{G} \leftarrow \mathcal{N}(0, mut_v)\ \forall$ weight $\in \theta_p,\ \forall p \in \mathcal{P}$
7: $b \leftarrow$ Sample a batch of visited states
8: Compute sensitivity λ as Equation 10.6 using b
9: $\theta_p \leftarrow \theta_p + \frac{\mathcal{G}}{\lambda}\ ,\ \forall p \in \mathcal{P}$
10: $\mathcal{P}$-fitness $\leftarrow$ *genetic-evaluation*($\mathcal{P}$, $\text{envs}_{\mathcal{P}}$)
11: $\theta_* \leftarrow$ Select best navigation policy using $\mathcal{P}$-fitness as Equation 8.10
12: Combine θ_e with θ_* using Equation 8.11
13: **End the evolutionary search**
14: **end if**
15: Continue the MARL training until the next EPS

- Line 12: Hence, we inject the best behaviors highlighted in the population in the policies of the multi-robot scenario, using a mean crossover operator based on Polyak averaging, which showed better performance in our preliminary experiments:

$$\theta_e = \beta\theta_* + (1-\beta)\theta_e \tag{8.11}$$

where β is a hyper-parameter that controls the amount of information injected from the best individual and the MARL algorithm. We noticed that high values of β (i.e., ≥ 0.4) affect the Multi-Agent (Deep) Reinforcement Learning network detrimentally due to a large amount of information on navigating in the stationary single-agent environment that is injected. Conversely, when $\beta < 0.4$, we found a beneficial transfer of information with a significant performance improvement, which benefits EPS.

8.6.1 Gradient-based Mutations

Perturbing the weights of a Deep Neural Network via simple Gaussian noise can lead to disruptive policy changes (Lehman et al., 2018). Hence, we use gradient information to design mutations that avoid such detrimental behaviors, normalizing the Gaussian perturbation by a per-weight sensitivity λ. We consider Gaussian noise $\mathcal{G}$ as a baseline for the perturbations and normalize it with our sensitivity λ, which we compute using past visited states in the memory buffer. We then apply the resultant gradient-based mutations to the population weights. Formally, we use the per-weight magnitude of the gradient of the outputs $\mathbf{y} = f_{\theta_e}(b)$ (where b is a randomly sampled batch of past

visited states, and f_{θ_e} is the function represented by the network with weights θ_e), to estimate the sensitivity λ to that weight with a first-order approximation:

$$\lambda = \sum_{\mathbf{y}} \left(\frac{\sum_{s \in b} abs(\nabla_{\theta_e} f_{\theta_e}(s))}{|b|} \right) * \frac{1}{|\mathbf{y}|} \tag{8.12}$$

where each sample of b contributes equally to λ to reduce the overall changes to the policy.[5] We refer to Chapter 10 for further details about gradient-based mutations. However, we note that this is the first application of this operator in a Multi-Agent (Deep) Reinforcement Learning context.

Limitations: We note that EPS shares a limitation with prior work (Khadka and Tumer, 2018; Khadka et al., 2019; Pourchot and Sigaud, 2019; Bodnar, 2020), requiring a simulator to perform the evolutionary search. However, state-of-the-art results in Deep Reinforcement Learning, robotics, and combined approaches in general, are mainly achieved using simulation and transferring the policy on real platforms (Zhao et al., 2020; Ding et al., 2020). Moreover, our formalization of EPS in Algorithm 6 requires weight sharing in the considered MARL baseline. While this is typically used in Multi-Agent (Deep) Reinforcement Learning (Lowe et al., 2017; Rashid et al., 2020, 2018), we note that EPS is also compatible with the scenario where each robot maintains its separate set of weights. In this case, it is sufficient to instantiate the population $\mathcal{P}$ with the copies of each robot's set of weights (i.e., the population size equals the number of robots) and compute the sensitivity separately using individuals experiences. Finally, since EPS works on top of existing algorithms, it (possibly) applies to any MARL baseline.

8.6.2 Empirical Evaluation

We evaluate the augmented evolutionary Global Dueling Q-learnings on the same setup of GDQ, starting with a preliminary experiment on the multi-agent benchmark Cooperative navigation and then our multi-agent navigation scenario with up to twelve agents.

Note that for a fair comparison, we include the additional epochs required by EPS in all the results as this is typical in combined approaches (Khadka and Tumer, 2018). However, we note that the training time overhead is negligible due to parallelization (i.e., each single-robot environment is strictly independent and, in our experiments, EPS-based approaches train with an overhead of $\approx 4 \pm 3\%$ of the considered Multi-Agent (Deep) Reinforcement Learning baseline).

Standard Navigation Benchmark

We first investigate the benefits of EPS applied to an independent learners (IQL) algorithm based on the state-of-the-art value-based algorithm Rainbow (Hessel et al., 2018), and GDQ that showed superior performance over prior value-based MARL. We refer to these implementations as EPS-IQL, and EPS-GDQ, respectively.

Results in Table 8.3 show the average reward and percentage of collisions of IQL, EPS-IQL, GDQ, EPS-GDQ considering the entire training phase. These preliminary

[5] We use the absolute value of $\nabla_{\theta_e} f_{\theta_e(s)}$ as we are interested in the magnitude (not the sign) of the slope.

experiments confirm the idea behind EPS, where searching concurrently for better core navigation behaviors leads to better performance (i.e., higher reward and fewer collisions).

TABLE 8.3: Avg. reward and collision % of the training phases for IQL, EPS-IQL, GDQ, EPS-GDQ.

Algorithm	Avg. Reward	Avg. CollisionS (%)
IQL	-2.42 ±0.32	29.2 ±2.5
EPS-IQL	-2.16 ±0.25	25.1 ±2.6
GDQ	-2.12 ±0.24	18.2 ±1.5
EPS-GDQ	-1.95 ±0.19	16.3 ±1.7

Unity Multi-Agent Navigation

For each experiment with $n = \{2, 4, 8, 12\}$ robots, we plot the following curves that report mean and standard deviation over the multiple runs, smoothed over 100 epochs. In more detail, we plot the average reward as the main evaluation metric, which indicates the navigation performance. We also discuss the relative success rate that indicates how many successful collision-free trajectories (i.e., we consider a success when all the robots in the environment reach their target) are performed over 100 epochs.

Training Results: Figure 8.6 shows the results of IQL, QMIX, GDQ, and EPS-GDQ in the navigation tasks with a growing number of robots, where the goal is to learn how to reach different target positions while avoiding collisions with the walls and with each other.

Similar to the GDQ evaluation, we note that results are comparable when considering two robots, while the benefits of EPS become evident with a growing number of robots. In more detail, with $n = 4$ EPS-GDQ stabilizes at ≈35 average reward (i.e., ≈ 90%) in ≈100000 steps. The standard GDQ and QMIX, in contrast, reach ≈88% and ≈90% successes in 175000 and 110000 steps, respectively. The performance improvement of EPS-GDQ is evident with $n = \{8, 12\}$ robots, where QMIX does not learn how to factorize the global action-value in so few interactions with the environment. In contrast, EPS offers a clear performance advantage over GDQ, especially in the initial training phases where core navigation behaviors are learned faster.

TABLE 8.4: Avg. reward and collisions % for QMIX, GDQ, EPS-GDQ in the evaluation in a previously unseen scenario.

Algorithm	Avg. Reward	Collision (%)
QMIX	26.3 ± 3.1	1.4 ± 2.6
GDQ	30.1 ± 2.7	0.8 ± 1.8
EPS-GDQ	32.4 ± 2.5	0.9 ± 1.0

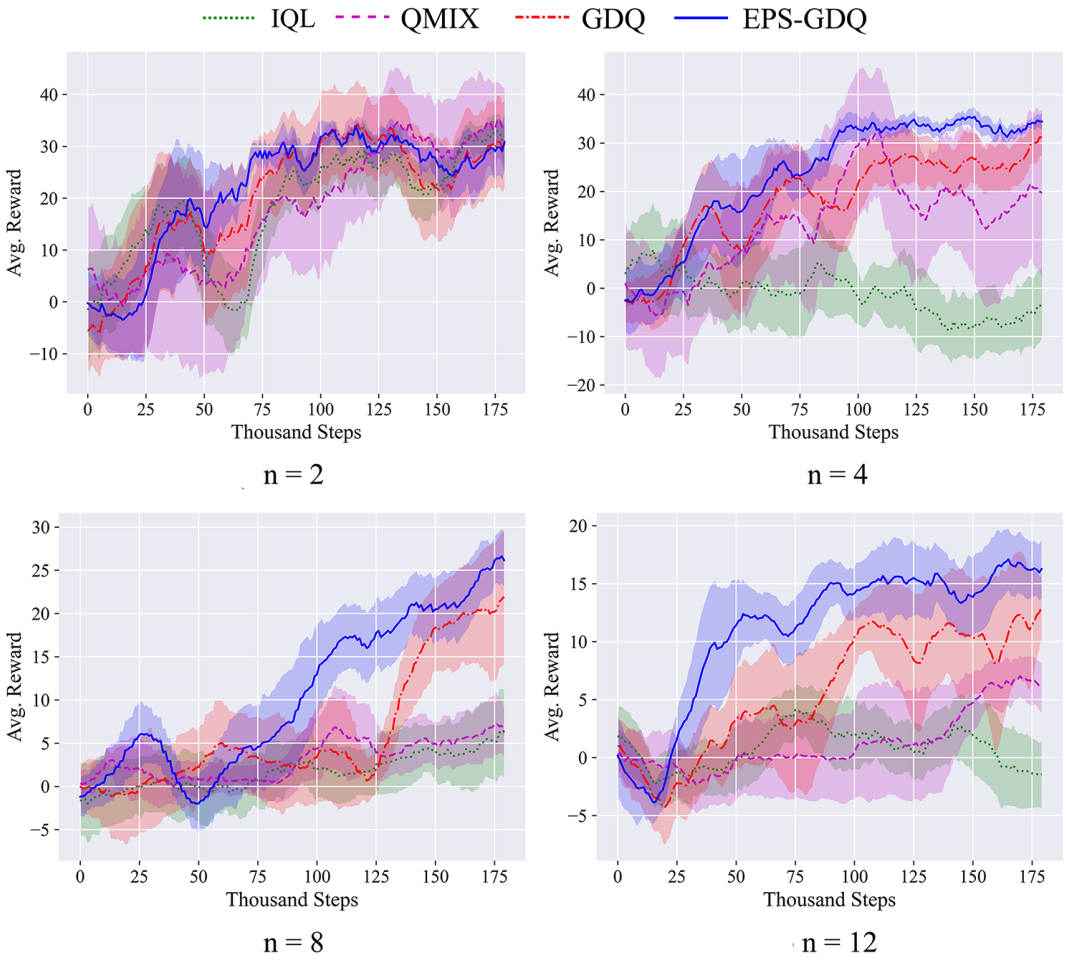

FIGURE 8.6: Average reward for IQL, QMIX, GDQ, and EPS-GDQ in our multi-robot navigation scenarios with $n = \{2, 4, 8, 12\}$ robots.

Testing Results: To test whether our models generalize over crucial aspects of robotic navigation, we perform an additional experiment with four robots in a previously unseen scenario.[6] For a fair evaluation, we selected a sequence of targets that were reachable by all the models. Results in Table 8.4 confirm the trend highlighted during the training, where EPS-GDQ outperforms the baseline in terms of average reward. We also measured the percentage of collisions detected, further confirming the performance improvement obtained with our evolutionary search, which returns comparable collisions over GDQ, but with a higher reward.

8.6.3 Ablation Study

To further confirm our idea that EPS improves exploration and sample efficiency of prior algorithms, we performed an additional ablation study in the scenario with four robots. In detail, we initially trained a navigation policy in the single-robot scenario. Hence, we initialize the network's weights of GDQ with the resultant set of weights of the pre-training (which we refer to as Pre-GDQ) in a similar fashion to transfer

[6] We consider four robots due to the good performance of the models trained with the MARL baselines.

FIGURE 8.7: Average reward for the single-agent pre training (left) and Pre-GDQ (right) initialized with a single-robot policy that reached ≈20 avg. reward. Pre-GDQ reaches ≈30 avg. reward in around 210000 steps.

learning. Our goal is to verify whether the EPS periodical search offers improved performance over the more intuitive solution of Pre-GDQ.

We performed three experiments with different initialization at different stages of the pre-training (i.e., at around 50, 75, and 95% of success rate for the single-robot pre-training, which corresponds to $\approx 12, 20, 30$ average reward, respectively). Figure 8.7 shows the average reward over different runs for Pre-GDQ initialized at ≈75% successes. Moreover, Table 8.5 shows the performance of our trials with different initializations (where we indicate the steps in thousands). Crucially, every experiment confirms our intuition behind EPS as our EPS-GDQ achieves ≈35 average reward in 100000 steps (i.e., approximately two times fewer steps than the best performing Pre-GDQ).

TABLE 8.5: Average results with different pre training for GDQ. EPS-GDQ outperform Pre-GDQ in any inizialitation, achieving ≈35 avg. reward in ≈100000 steps.

Pre Training		**Pre-GDQ**		**EPS-GDQ**	
Steps	Reward	Steps	Reward	Steps	Reward
75	12	200	25		
110	20	210	30	100	35
170	30	280	34		

8.7 Discussion

We presented Global Dueling Q-learning, a novel value-based Centralized Training Decentralized Execution approach for multi-agent scenarios that exploit state-values information to favor cooperation.

Our empirical evaluation shows that GDQ significantly outperforms prior CTDE approaches in multi-robot robotic navigation, especially with a growing number of robots. Crucially, we only consider the individual's state values and no additional centralized information (such as in QMIX). This also allows us to scale the number of robots without significant overhead for the training process (each robot only adds an input node in our centralized joint state-value estimator).

We also extended Global Dueling Q-learning within the Evolutionary Policy Search, a novel approach for multi-robot navigation that works on top of existing algorithms, maintaining their Centralized Training Decentralized Execution fashion. The idea is to perform a periodical evolutionary search to find better core navigation behaviors to inject into the MARL training. We evaluated our framework in a multi-robot robotic navigation scenario with up to 12 robots. This empirical evaluation shows that EPS-GDQ significantly improves the performance of prior Multi-Agent (Deep) Reinforcement Learning approaches, especially with a growing number of robots.

8.8 Conclusions

In this second part of the thesis, we focused on enhancing exploration in single and multi-agent domains, combining gradient-based Deep Reinforcement Learning algorithms with Evolutionary Algorithms.

We started by discussing the recent field that combines gradient-free and gradient-based methods. Crucially, we showed the limitations of prior combined work that hinder its applicability with value-based DRL. Hence, we proposed an evolutionary framework that copes with such limitations, applicable to any Deep Reinforcement Learning algorithm, that showed robustness over detrimental hyperparameters setting.

Finally, we discussed the exploration issues of prior value decomposition Multi-Agent (Deep) Reinforcement Learning algorithms and proposed a framework that uses a joint state-value to cope with such problems. We also extended this solution using evolutionary approaches to decompose the desired policy and foster more robust navigation performance.

Part III

Exploring for Safer Behaviors

Chapter 9

Quantifying Safe Behaviors

Practical applications, such as robotics usually involve high-cost hardware. Hence, the behavior of the trained policy must be evaluated to avoid potentially dangerous situations.

This chapter discusses the design of safety metrics that quantify the number of correct decisions the network chooses over pre-defined safety specifications. We also employ combined training algorithms described in Chapter 7 to assess whether policies trained with combined approaches result in more robust and safer behaviors.

9.1 Introduction

Following the insights of Chapter 5, we remark that it is crucial to evaluate the correct behavior of a trained model before and design a methodology to quantify the amount of network's safe decisions (Liu et al., 2019). To this end, we formally define a metric that quantifies the number of correct decisions that a trained Deep Reinforcement Learning policy takes over pre-defined safety specifications. We also use the state configurations where such undesirable settings occur to discuss how it is possible to exploit this information to design a controller to avoid such unsafe behaviors.

In detail, we use interval analysis (Moore, 1963) based verification to verify the relations between two or more network outputs in the case of value-based DRL or the output values in the case of continuous control. In contrast to prior work that mainly considers such analysis from dependent input intervals, our approach subdivides the analysis into small input intervals independent of each other. This allows a straightforward parallelization and the computation of the percentage of undesirable situations. We discussed this verification setup in Chapter 5.

To analyze the safety of the trained models, we employ policies trained for aquatic navigation due to the physically realistic water surface with dynamic waves that makes the task particularly challenging. In this context, we introduce a set of properties that describe core behaviors that an autonomous controller should respect in our task. Moreover, we use the combined approaches presented in Chapter 7, namely the value-based SGRainbow and the actor-critic GPPO, to train the navigation policies. Our goal is to show that the beneficial transfer of information between EA and gradient-based DRL results in more robust policies. Crucially, the reduced number of unsafe behaviors of the combined policies allow us to design a simple controller that guarantees the correct behavior of the network.

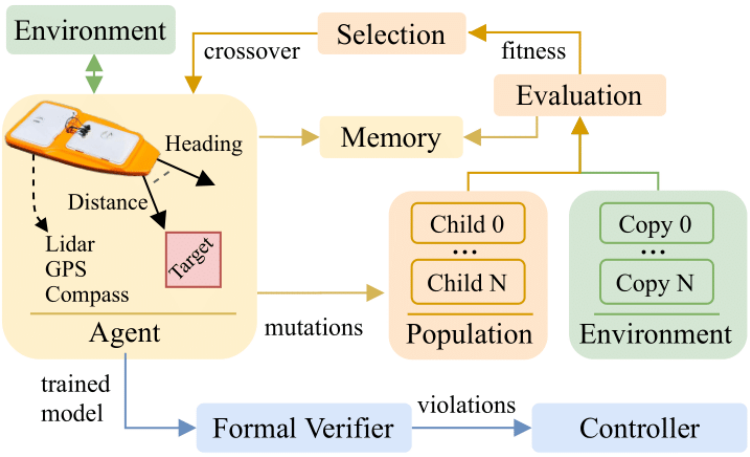

FIGURE 9.1: Overall schematic of our experimental setup.

We note that the methodologies for this chapter have already been presented in the previous chapters. Hence. Figure 9.1 highlights a summary of the overall setup for these experiments.

9.2 Experiments

Designing a set of properties for our aquatic navigation scenario that presents a variable set of obstacles is a challenging problem. Given the dynamic and non-stationary nature of such an environment, it is not possible to formally guarantee the safety of the drone in any possible situation. For this reason, we focus on ensuring that the agent makes rational decisions (i.e., it selects the best action given available information) to provide an initial benchmark for the safety of this class of problems. Hence, we selected three core properties that represent a possible safe behavior of our agent about possible obstacles:

- $p_{\leftarrow}$**:** If there is an obstacle near to the left, whatever the target is, go straight or turn right.
- $p_{\rightarrow}$**:** If there is an obstacle near to the right, whatever the target is, go straight or turn left.
- $p_{\uparrow}$**:** If there are obstacles near both to the left and to the right, whatever the target is, go straight.

To formally rewrite these properties as Equation 2.14, 5.1, it is necessary to detail further the structure of the input and the output layers of our Deep Neural Network, and formally define the concept of "near" for an obstacle. In both the value-based and policy-gradient setup, the input layer contains 17 inputs: (i) the lidar collect 15 values normalized $\in [0, 1]$, (ii) the heading of the target with respect to the robot heading ($\in [-1, 1]$), and (iii) the distance of the target from the robot ($\in [0, 1]$). The output nodes correspond to specific actions in the value-based case: from v_0 to v_6, each node represents the action "strong right, right, weak right, forward, weak left, left, and strong left". While for the policy-gradient scenario we have two outputs v_{left} for the left motor and v_{right} for the right one (for simplicity we only consider forward movements, hence v_{left} and v_{right} can assume values $\in [0, 1]$)

Moreover, we measured that an obstacle is close to the agent if its distance is less than 0.35 on the front, and less than 0.24 on the two sides (normalized in the corresponding input domain), considering the max velocity of the agent (hence, these values allow us to turn in the opposite direction to a fixed obstacle without colliding). Given this, we obtain the following input domains for our properties:

- $I_{p_{\leftarrow}} : x_0, ..., x_5 \in [0.00, 0.24] \wedge x_5, ..., x_{14}, x_{15}, x_{16} \in \mathcal{D}$
- $I_{p_{\rightarrow}} : x_{10}, ..., x_{14} \in [0.00, 0.24] \wedge x_0, ..., x_{10}, x_{15}, x_{16} \in \mathcal{D}$
- $I_{p_{\uparrow}} : x_0, ..., x_5, x_{10}, x_{14} \in [0.00, 0.24] \wedge x_6, ..., x_6, x_{15}, x_{16} \in \mathcal{D}$

where $\mathcal{D}$ is the complete domain of the corresponding node where the obstacle is not near. Finally, we formalize the previous decision properties for the value-based combined SGRainbow and the actor-critic combined GPPO as follow:

- $p_{\leftarrow,SGRainbow}$: If $I_{p_{\leftarrow}} \Rightarrow [v_4, v_5, v_6] < [v_0, v_1, v_2, v_3]$
- $p_{\rightarrow,SGRainbow}$: If $I_{p_{\rightarrow}} \Rightarrow [v_0, v_1, v_2] < [v_3, v_4, v_5, v_6]$
- $p_{\uparrow,SGRainbow}$: If $I_{p_{\uparrow}} \Rightarrow [v_0, v_1, v_5, v_6] < [v_2, v_3, v_4]$
- $p_{\leftarrow,GPPO}$: If $I_{p_{\leftarrow}} \Rightarrow v_{left} - v_{right} > k$
- $p_{\rightarrow,GPPO}$: If $I_{p_{\rightarrow}} \Rightarrow v_{right} - v_{left} > k$
- $p_{\uparrow,GPPO}$: If $I_{p_{\uparrow}} \Rightarrow |v_{left} - v_{right}| < k$

where k is a constant value for the minimum motors power difference allows a rotation that avoids a collision.

Finally, in contrast to prior work that typically returns SAT if the property is SATisfied or UNSAT if it is UNSATisfied for at least one input (Wang et al., 2018b; Katz et al., 2017), we aim at quantifying the safety of the individuals over the properties. To this end, we propose the following *violation* metric v to quantify the number of violations.

Definition 9.1 (Violation metric) Given a (safety) property $p := \mathbf{x} \in \mathcal{X} \Rightarrow \mathbf{y} = f_\theta(\mathbf{x}) \in \mathcal{Y}$ on f_θ, and its reachability set $\Gamma(\mathcal{X}, f_\theta) := \{\mathbf{y} : \mathbf{y} = f_\theta(\mathbf{x}),\ \forall \mathbf{x} \in \mathcal{X}\}$. Given $\mathcal{X}_{\text{SAT}}, \mathcal{X}_{\text{UNSAT}} \subseteq \mathcal{X}$ such that $\Gamma(\mathcal{X}_{\text{SAT}}, f_\theta) \subseteq \mathcal{Y}$ and $\Gamma(\mathcal{X}_{\text{UNSAT}}, f_\theta) \cap \mathcal{Y} = \emptyset$.[1] We define the violation metric as:

$$v = \frac{|\mathcal{X}_{\text{UNSAT}}|}{|\mathcal{X}|}. \tag{9.1}$$

9.3 Empirical Evaluation

Table 9.1 shows the results for each property, considering the average violation of the ten best performing models (according to their success rate) for the best three random seeds initialization. We show that converged models that achieve a similar return could suffer from input corner cases in a very different way. For example, models trained with seed 1 present a violation of ≈9.3 on seed 1 for $p_{\leftarrow,SGrainbow}$, while models with seed 3 have a violation of 0 on the same property. Results confirm our insights of Chapter 4, showing that the violation and the success rate are not necessarily related. Although we tested the safety on the best models, we found a high violation rate in

[1] Note that $\mathcal{X}_{\text{SAT}} \cup \mathcal{X}_{\text{UNSAT}} = \mathcal{X}$ and $\mathcal{X}_{\text{SAT}} \cap \mathcal{X}_{\text{UNSAT}} = \emptyset$.

TABLE 9.1: Mean and the variance of the violation metric (%) for the best three random seeds initialization.

Property	**Seed 1**	**Seed 2**	**Seed 3**
$p_{\leftarrow,SGrainbow}$	9.3 ± 2.4	5.3 ± 3.1	0.0 ± 0.0
$p_{\rightarrow,SGrainbow}$	3.0 ± 2.3	4.1 ± 2.2	0.0 ± 0.0
$p_{\uparrow,SGrainbow}$	7.1 ± 1.4	6.9 ± 2.7	3.4 ± 0.2
$p_{\leftarrow,GPPO}$	1.3 ± 0.3	1.3 ± 0.7	1.3 ± 0.2
$p_{\rightarrow,GPPO}$	0.2 ± 0.2	0.0 ± 0.0	0.7 ± 0.5
$p_{\uparrow,GPPO}$	3.6 ± 1.4	6.3 ± 2.6	3.1 ± 0.1

some cases. For this reason, we remark that this safety analysis is a necessary step to evaluate a policy before its deployment in a real-world environment.

Moreover, to further confirm our claims on the beneficial transfer of information of mixed approaches and the robustness highlighted in Section 7.4, we performed an additional evaluation using formal verification. In more detail, we report the violation percentage, computation time, and memory returned by the verification tool to test our safety properties. Table 9.2 reports the average collected metrics considering the best ten performing models of the best seed initialization. Models trained with mixed approaches (i.e., SGRainbow, GPPO) present fewer violations in every considered property over the standard gradient-based baseline (i.e., Rainbow and PPO). Furthermore, there is also a significant improvement over the computation time and memory required by the verifier. This confirms our claims on the policy improvement of mixed approaches as they evaluate with a significant difference in the output values, which translates into fewer bounds re-computations for the verifier.

TABLE 9.2: Verification results for combined approaches and standard gradient-based DRL.

	Violation (%)			**Time (s)**			**Memory (MB)**		
Algorithm	$p_{\leftarrow}$	$p_{\rightarrow}$	$p_{\uparrow}$	$p_{\leftarrow}$	$p_{\rightarrow}$	$p_{\uparrow}$	$p_{\leftarrow}$	$p_{\rightarrow}$	$p_{\uparrow}$
Rainbow	2.21	9.11	0.0	79.7	75.5	92.6	3.74	3.96	6.92
SGRainbow	0.0	4.75	0.1	66.7	74.1	68.9	2.18	2.91	4.1
PPO	0.9	1.2	4.2	3.4	56	124	0.1	2.6	5.8
GPPO	1.3	0.7	3.1	3.1	3.2	3.6	0.1	0.1	0.14

Behavioral Controller: Due to the low violation rate of the combined models, it is possible to design a simple controller to guarantee the correct behavior of the network. To illustrate this, we describe the process to decide whether a behavioral controller can address the unsafe behaviors in our navigation task.

Given the decision-making frequency of the robot set to $10Hz$ and the violation presented in Table 9.2, a complete search through the data structure that contains all the sub-areas that cause a violation always requires less than 0.09s. This means that, with our hardware setup, we can verify if the input state leads to a violation at each iteration without lags in the robot operations. Consequently, in an ideal scenario where

there are no communication lags or other overheads, we can avoid all the decisions derived from input configurations that violate our desired safety properties.

9.4 Discussion

We formally presented the violation metric as a practical way to quantify the number of unsafe behaviors over pre-defined safety specifications. We evaluated the behaviors of aquatic navigation models to show the importance of such metrics. Moreover, we exploited such violation value to confirm further the robustness of prior combined approaches over standard gradient-based Deep Reinforcement Learning.

Chapter 10

Safety-Oriented Search

This chapter considers Reinforcement Learning problems where an agent attempts to maximize a reward signal while minimizing a cost function that models unsafe behaviors.

In contrast to prior work employing constrained optimization, we propose a Safety-Oriented Search that complements Deep Reinforcement Learning algorithms to bias the policy toward safety within an evolutionary cost optimization. We leverage evolutionary exploration benefits to design a novel concept of safe mutations that use visited unsafe states to explore safer actions. We further characterize the behaviors of the policies over desired specifics with a sample-based bound estimation, which makes prior verification analysis tractable in the training loop. Hence, driving the learning process towards safer regions of the policy space.

10.1 Introduction

We consider the class of problems where unsafe behaviors are specified with an auxiliary cost signal to maintain safety specifications separate from the task objective (e.g., the long-term reward) (Garcıa and Fernández, 2015). In the literature, Constrained Markov Decision Process (Altman, 1999) are used to formalize such problems due to the intuitive way of constraints (on the cost) to encode safety criteria (Liu et al., 2020; Stooke et al., 2020). However, constrained DRL often violates the constraints introduced in the optimization and naturally limits exploration (Ray et al., 2019). Conversely, efficient exploration is crucial to avoid getting stuck in local optima or failing to learn proper behaviors, obtaining low returns (Hong et al., 2018; Ostrovski et al., 2017; Pathak et al., 2017). Such a trade-off between having efficient exploration to achieve good performance, and the limitation induced by constraints, suggest investigating alternative ways to overcome the typical contrast in the design of Safe Deep Reinforcement Learning algorithms. In particular, such algorithms are either based on the *optimality criterion* (e.g., introduce the concept of risk in the optimization), or on the *exploration process* (e.g., avoid undesirable situations) (Garcıa and Fernández, 2015).

To this end, we propose Safety-Oriented Search (SOS) to combine *exploration process* and *optimality criterion* into a unique framework, depicted in Figure 10.1, that works on top of existing DRL algorithms. Our goal is to bias policies toward safety without multi-objective or constrained optimization without formalizing the problem as a CMDP. We leverage Evolutionary Algorithms for SOS to augment Deep Reinforcement Learning, proposing a novel concept of Safe Mutations (SMs). SMs exploit

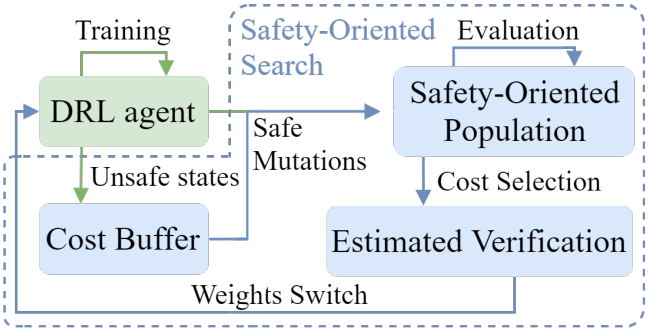

FIGURE 10.1: High-level schematics of SOS.

the visited states deemed unsafe according to the cost to approximate the per-weight sensitivity of the actions over such undesired situations. Then, such sensitivity is used to compute safety-informed perturbations that locally bias the agent policy to explore different behaviors (i.e., actions) in the proximity of the unsafe states (i.e., the *exploration process*). In more detail, an evolutionary population is periodically generated from the DRL policy using Safe Mutations, and it is evaluated independently over a set of trials to select the subset of individuals with returns comparable to the Deep Reinforcement Learning agent and a lower cost.

Assuming that this subset improves the additional cost, we note that such signal is typically sparse (i.e., it is not trivial to shape the risk associated with every state in a high-dimensional space). Hence, the cost function does not fully characterize the behaviors of the policy, sharing the issues of using sparse rewards (Hong et al., 2018). For this reason, we argue that a tractable characterization of the behaviors of a Deep Neural Network is instrumental in evaluating the safety of a policy (Liu et al., 2019). To this end, we employ our violation metric to quantify the number of correct decisions that a policy chooses over desired safe specifications. We then select the policy in the subset with the lowest violation that will replace the agent (i.e., the *optimality criterion*). However, Formal Verification (Wang et al., 2018b; Weng et al., 2018) is known to be computationally demanding and can not be used directly in the training loop without assumptions that can not be satisfied in practice (e.g., having an optimal policy (Lutjens et al., 2020)). We propose to relax the formal guarantees of prior bound-estimation methods (Wang et al., 2018b) with a sample-based estimation to make this analysis tractable. Crucially, our Estimated Verification (EV) impacts the training within a negligible overhead. To summarize, we foster safer behaviors with periodical Evolutionary Algorithm evaluations while the Deep Reinforcement Learning training process optimizes the reward objective.

10.2 Preliminaries

A Constrained Markov Decision Process (Altman, 1999) is a Markov Decision Process with an additional set of constraints $\mathcal{C}$ based on $C_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ ($i = \{1, \dots, n\}$) cost functions (similar to the reward) and $\mathbf{h} \in \mathbb{R}^n$ thresholds for the constraints. The C_i-*return* is defined as $J_\pi^{C_i} := \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t C_i(s_t, a_t)]$, where $\gamma \in (0, 1)$ is the discount, $\tau = (s_0, a_0, \dots)$ is a trajectory, $\pi = \{\pi(a|s) : s \in \mathcal{S}, a \in \mathcal{A}\}$ denotes a policy in state $\mathcal{S}$ and action $\mathcal{A}$ spaces. Constraint-satisfying (feasible) policies $\Pi_\mathcal{C}$, and optimal policies

π_* are thus defined as:

$$\Pi_{\mathcal{C}} := \{\pi \in \Pi : J_\pi^{C_i} \leq h_i,\ \forall i\}, \quad \pi_* = \arg\max_{\pi \in \Pi_{\mathcal{C}}} J_\pi \tag{10.1}$$

where $J_\pi := \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$ is the expected discounted return that we aim at maximizing in a standard MDP; Π are the stationary policies, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. Without loss of generality, we consider the case of one cost function (as in recent constrained Deep Reinforcement Learning literature (Ray et al., 2019; Stooke et al., 2020; Liu et al., 2020)) and we will discuss later how SOS could handle multiple cost functions.

10.2.1 Gradient-based Mutations

Mutating a policy with simple Gaussian noise can lead to disruptive changes (Lehman et al., 2018) that can be naively address uses zero-mean and low standard deviation (Martin H. and de Lope, 2009). Otherwise, if we define a genome as a Deep Neural Network parametrized by θ that represents a function $f_\theta : \mathcal{D}_{\mathbf{x}} \to \mathcal{D}_{\mathbf{y}}$ (input $\mathbf{x} \in \mathcal{D}_{\mathbf{x}}$ and output $\mathbf{y} \in \mathcal{D}_{\mathbf{y}}$), and a vector of states $\mathbf{s}$, we can express the average divergence of the outputs $\mathbf{y}$ as a result of a perturbation δ as:

$$d(f_\theta, \delta) = \frac{\|f_\theta(\mathbf{s}) - f_{\theta+\delta}(\mathbf{s})\|_2}{|\mathbf{s}|} \tag{10.2}$$

where $f_\theta(\mathbf{s})$ are the forward propagations of the states through the DNN. A more flexible way to avoid disruptive mutations assumes using a differentiable Deep Neural Network to approximate $d(f_\theta, \delta)$ with gradient information (Lehman et al., 2018). In detail, it considers the following first-order Taylor expansion to model an output $y_j \in \mathbf{y}$ ($j = \{0, \ldots, |\mathbf{y}|\}$) as a function of perturbations δ over the states $\mathbf{s}$:

$$y_j(f_\theta, \delta) = f_\theta(\mathbf{s})_j + \delta \nabla_\theta f_\theta(\mathbf{s})_j \tag{10.3}$$

In later sections, we discuss how Safe Mutations specializes gradient-based mutations (Equation 10.3) to explore safer behaviors.

10.3 Safety-Oriented Search

SOS proposes two mechanisms to foster safety:

1. A novel concept of Safe Mutations to augment DRL with a policy search devoted to exploring safer behaviors.
2. An Estimated Verification that relaxes the guarantees of FV to characterize the behaviors of the policies during the training tractably.

The general flow of SOS is presented in Algorithm 7:

- We augment the Deep Reinforcement Learning agent training with a *cost-buffer* B_c which stores the visited unsafe states (according to the cost).
- Lines 2-9: Periodically, we sample a batch b from B_c to compute the per-weight safety-informed sensitivity λ of the agent outputs over its weights θ_a. This is used by SM to generate a population of n individuals (DNNs) $\mathcal{P} = \{p_1, \ldots, p_n\} \cup \{p_a\}$ with weights $\theta_{\mathcal{P}}$ (p_a is a copy of the agent), voted to explore for safer behaviors.

- Line 10: $\mathcal{P}$ is evaluated in a set of epochs to collect the individuals average reward R_p and cost C_p that define the fitness score $\mathcal{P}$-fitness $= (R_p, C_p)\ \forall p \in \mathcal{P}$
- Line 11: Such fitness is used to select a subset of genomes $\mathcal{P}'$ as:[1]

$$\mathcal{P}' = \{p_j \in \mathcal{P} : \mathcal{P}\text{-fitness}_j \geq \mathcal{P}\text{-fitness}_a,\ j = \{1, \dots, n\}\} \cup \{p_a\} \tag{10.4}$$

  where $\mathcal{P}\text{-fitness}_j \geq \mathcal{P}\text{-fitness}_a \Rightarrow R_{p_j} \geq R_{p_a} \wedge C_{p_j} \leq C_{p_a}$. By choosing an appropriate number of evaluation epochs for the population, we define the individuals in $\mathcal{P}'$ to be safer than p_a as they have higher (or equal) rewards and lower (or equal) costs.
- Line 12: Although $\mathcal{P}'$ improves the additional cost, such a sparse metric does not characterize the behaviors of the policies. For this reason, SOS selects the "safest" genome $\theta_* \in \mathcal{P}'$ using the Estimated Verification detailed in the related section.
- Line 13: Hence, if θ_* is a SM perturbed policy (i.e., $\theta_* \neq p_a$) it substitutes p_a to continue the training, otherwise the training that is running in parallel, continues.

We note that in a worst-case scenario, we match the performance of the baseline Deep Reinforcement Learning agent as we would never switch its policy. Summarizing, SOS proposes a periodical search devoted to safety-oriented exploration to improve the DRL policy, simulating a small gradient step toward a "safer" (better) policy.

Algorithm 7 Safety-Oriented Search

Given:

- a DRL agent with weights θ_a at the current epoch in environment *env*
- a verifier V_{reach} and desired safety-properties P_s ▷ e.g., Neurify, ReluVal
- a cost-buffer B_c filled with unsafe samples
- periodicity $e_{\mathcal{P}}$ for SOS and population size n
- scale mut_v for the Gaussian noise $\mathcal{G}$ and threshold λ_{max}

1: While the standard training of the agent proceeds:
2: **if** epoch % $e_{\mathcal{P}} == 0$ **then**
3: **Start the safety-oriented search** ▷ typically in a parallel fashion
4: $\mathcal{P} \leftarrow n + 1$ copies of θ_a (each $p \in \mathcal{P}$ has weights θ_p)
5: $\text{envs}_{\mathcal{P}} \leftarrow$ Initialize with $n + 1$ copies of *env*
6: $b \leftarrow$ Sample an unsafe batch from B_c
7: Compute $\mathcal{G} \leftarrow \mathcal{N}(0, mut_v)\ \forall$ weight $\in \theta_p,\ \forall p \in \mathcal{P}$ (i.e., baseline noise)
8: $\lambda \leftarrow$ Equation 10.6 using b, replacing values $\leq \lambda_{max}$ with λ_{max}
9: $\theta_p \leftarrow \theta_p + \frac{\mathcal{G}}{\lambda},\ \forall p \in \mathcal{P}$
10: $\mathcal{P}$-fitness $\leftarrow$ *genetic-evaluation*($\mathcal{P}$, $\text{envs}_{\mathcal{P}}$)
11: Select a "safer" subset $\mathcal{P}'$ using $\mathcal{P}$-fitness as Equation 10.4
12: $v_{\mathcal{P}'} \leftarrow V_{reach}(\tilde{\Gamma}_{\mathcal{P}'}, P_s)$ using Def. 9.1 and 10.2
13: $\theta_a \leftarrow \min_{\theta_{\mathcal{P}'}} v_{\mathcal{P}'}$
14: **End the safety-oriented search**
15: **end if**
16: Continue the training of the agent until the next SOS

[1] Note that ordering among fitness tuples is feasible as its components $R, C \in \mathbb{R}$.

10.3.1 Safe-Informed Evolutionary Operators

Section 8.6.1 discussed how perturbing the weights of a Deep Neural Network via simple Gaussian noise can lead to disruptive policy changes (Martin H. and de Lope, 2009). Gradient information (sensitivity) can be used to design mutations that avoid such detrimental behaviors, normalizing the perturbation by a per-weight sensitivity (Lehman et al., 2018).

We leverage the cost function to design our Safe Mutations, avoiding disruptive changes to the Deep Reinforcement Learning policy while biasing it to explore safer behaviors. We consider a baseline Gaussian noise $\mathcal{G} \sim \mathcal{N}(0, mut_v)$ for the perturbations and normalize it with our safety-informed sensitivity λ. The resultant mutations δ_{SM} are applied to the agent weights θ_a to generate $\mathcal{P}$. One way to compute λ considers the gradient of the actual divergence (Equation 10.2) (Lehman et al., 2018):

$$\begin{aligned} \nabla_{\theta_a} d(f_{\theta_a}, \mathcal{G}) &\approx \nabla_{\theta_a} d(f_{\theta_a}, 0) + H_{\theta_a}(d(f_{\theta_a}, 0))\mathcal{G} \\ \lambda_{f_{\theta_a}} &= abs(\nabla_{\theta_a} d(f_{\theta_a}, \mathcal{G})) \end{aligned} \tag{10.5}$$

where H_{θ_a} is the Hessian of divergence with respect to θ_a. However, this requires second-order approximations, and therefore it is computationally demanding. To address this, we use the per-weight magnitude of the gradient of the outputs $\mathbf{y} = f_{\theta_a}(b)$, where b is a batch of unsafe states randomly sampled from B_c, to estimate the sensitivity λ to that weight with a first-order approximation:

$$\begin{aligned} \lambda_{f_{\theta_a}} &= \sum_{\mathbf{y}} \left(\frac{\sum_{\mathbf{s}} abs(\nabla_{\theta_a} f_{\theta_a}(\mathbf{s}))}{|\mathbf{s}|} \right) \frac{1}{|\mathbf{y}|} \\ \delta_{SM}(f_{\theta_a}) &= \frac{\mathcal{G}}{\lambda_{f_{\theta_a}}} \end{aligned} \tag{10.6}$$

where each unsafe experience equally contributes to λ to reduce the overall changes to the policy.[2] In practice, we note that using a threshold λ to limit the mutation rescaling (i.e., λ_{max}) leads to better performance. To summarize, our idea is to design safety-oriented gradient information using visited unsafe states to bias the policy to explore different actions in the proximity of such situations.

10.3.2 Relaxing Formal Verification

We leverage formal verification for Deep Neural Networks to evaluate the subset of safer genomes $\mathcal{P}'$ and characterize their behavior over a set of given properties in the form of Equation 5.1. In particular, we use the violation of Chapter 9 for our evaluation.

Despite recent advances in the field of formal analysis for DNNs (Wang et al., 2018b; Weng et al., 2018), existing tools require non-negligible computation time to approximate the reachable set using Equations 2.15, 2.16 (Liu et al., 2019). Hence, these approaches can not be directly applied to verify the properties (and compute v) during the training. We propose the following empirical strategy to estimate the reachability set, using feedforward steps of the Deep Neural Network. We thus apply the verification phase of an existing framework (Corsi et al., 2021) to the estimated bounds,

[2] We use the absolute value due to the interest in the magnitude, and not the sign, of the slope.

obtaining our Estimated Verification method that enables SOS to perform the verification in the training loop.

Definition 10.1 (Estimated Reachability Set) Given a (safety) property $p := \mathbf{x} \in \mathcal{X} \Rightarrow \mathbf{y} = f_\theta(\mathbf{x}) \in \mathcal{Y}$ on f_θ, and a set $\mathcal{X}' \subseteq \mathcal{X}$ of m samples. We define the reachability set:

$$\tilde{\Gamma}(\mathcal{X}', f_\theta) := \{[\min(f_\theta(\mathcal{X}')_y, \max(f_\theta(\mathcal{X}')_y)] \; \forall y \in \mathbf{y}\}$$

Crucially, given a discretization value ϕ for the input space of a property, our estimation returns the exact reachability set using $m = \frac{|\mathcal{X}|}{\phi}$ different samples. For example, in practical tasks such as robotics, ϕ could be the precision of the sensor. However, our interest is to characterize the behaviors in the proximity of unsafe states. Hence we further exploit the cost-buffer B_c to compute a cost-oriented reachability set:

Definition 10.2 (Estimated Cost Reachability Set) Given a (safety) property $p := \mathbf{x} \in \mathcal{X} \Rightarrow \mathbf{y} = f_\theta(\mathbf{x}) \in \mathcal{Y}$ on f_θ, and the cost-buffer B_c. We define the cost reachability set:

$$\tilde{\Gamma}(\mathcal{X} \cap B_c, f_\theta) := \{[\min(f_\theta(\mathcal{X})_y \cap B_c, \max(f_\theta(\mathcal{X})_y \cap B_c)] \; \forall y \in \mathbf{y}\}$$

10.3.3 Limitations of Safety-Oriented Search

Our evolutionary search assumes to have access to a simulation environment. This is common in Deep Reinforcement Learning, where significant results have been achieved mainly using simulation and transferring the policy on real platforms (Juliani et al., 2018; Zhao et al., 2020). We also assume a single cost function as in prior constrained DRL literature (Ray et al., 2019). However, it would be possible to handle multiple cost functions by using crossover operators, similarly to (Khadka et al., 2019). Verification assumes knowing desired specifications to design the properties, typically available in tasks with safety requirements. Commonly with prior formal verification (Liu et al., 2019), such properties are hand-designed, hence as the complexity of the task increases, the input space typically grows, and writing safety properties may be unfeasible. To this end, we believe that producing compact state representations to reduce complexity (Cuccu et al., 2019) could be an exciting topic for future investigation.

10.4 Experiments

We use Safety Gym (Ray et al., 2019), a recent benchmark for our class of problems (Hunt et al., 2021; Stooke et al., 2020). Safety Gym models several navigation tasks (i.e., reach a Goal, press a Button, Push a box) for various robots (i.e., Point, 2-wheels Car, 12-joint Doggo) and difficulty levels (i.e., 0, 1, 2). Each has various hazards and densities (randomly placed at each epoch) that induce the additional cost upon collisions. The reward is shared across tasks, with a dense component to promote movements toward the goal and a large sparse value for reaching it. Lidar scans sense the hazards and the goal and comprise the observation space and position and velocity sensors. We consider the six tasks recommended by the authors, namely PointGoal1, PointGoal2, PointButton1, PointPush1, CarGoal1, DoggoGoal1, that comprise at least one environment for each task, robot, and difficulty.

We designed three properties to characterize the policy behaviors to ensure that the agent chooses rational actions in the visited states. Given the cardinality of the inputs, we report natural language property descriptions, which are shared across the tasks. We remind to later sections for an overview of the formal definition in the form of Equation 5.1:

- $p_{\uparrow}$: If the robot has hazards too close on the front, it must turn in any direction or move backward.
- $p_{\rightarrow}$: If the robot has hazards too close on the right and the front, it must turn left or move backward.
- $p_{\leftarrow}$: If the robot has hazards too close on the left and the front, it must turn right or move backward.

We remark that our goal is to provide a high-level overview of core navigation skills to quantify the overall behaviors of the agent. Hence, despite using only three properties, their high-level formalization was sufficient to cover over $98.7 \pm 0.5\%$ of the visited unsafe states.

We investigate an on-policy version of SOS based on PPO (SOS-PPO) against PPO, CPO, Lagrangian-PPO (L-PPO), and IPO. We also report additional experiments with a TD3 implementation of SOS (SOS-TD3) and the indoor mapless navigation scenario. We compare with constrained Deep Reinforcement Learning as it is the most closely related to addressing safety using cost functions. We evaluate the effectiveness of SOS-based algorithms at minimizing the cost signal with the evolutionary search while preserving returns. Conversely, constrained DRL hinders exploration (to satisfy cost constraints) at the expense of low returns, leading to a significant performance trade-off (Ray et al., 2019).

Training Setup: We use prior implementations of CPO, PPO, TD3, and L-PPO, which improves the constraint satisfaction, and our version of IPO, which we carefully tuned since authors did not release code.[3] Similar to Ray et al. (Ray et al., 2019), our experiments share the same network architecture. We achieve comparable results over prior work by considering separate feedforward multi-layer perceptron policy and value networks of size (64, 64) with Tanh activations.

Experimental Setup: The experimental setup is the same as in previous chapters. In particular, we remark that the additional epochs required by SOS are included in all the results for a fair comparison. However, the overhead of training time is negligible due to parallelization (i.e., both the search and the Estimated Verification are independent for each individual, and SOS-PPO trains with an overhead of $4 \pm 2\%$ over PPO). Given the importance of the statistical significance of the results (Henderson et al., 2018; Colas et al., 2019), we report mean and standard deviation collected over ten independent runs with different random seeds. This motivates slightly different results over the original implementations evaluated over a few seeds.

Concerning the baselines and SOS implementations, we consider the following hyper-parameters:

- SOS-PPO and SOS-TD3: for the on-policy SOS-PPO, we considered the clipped PPO which was recommended in the original work (Schulman et al., 2017) and to which we remind the interested reader for further details. We use a buffer of

[3] We refer to the original papers for specific details about the approaches and hyper-parameters.

256 elements with mini-batches of size 64. For the additional experiments of the off-policy SOS-TD3, we use the TD3 implementation of the authors (Fujimoto et al., 2018), considering their set of hyper-parameters. We remind the interested reader of the original implementations of the considered baselines, and to their paper, for further details (Schulman et al., 2017; Fujimoto et al., 2018; Stooke et al., 2020; Liu et al., 2020; Ray et al., 2019).

- Safety oriented evaluation: we use the same set of hyper-parameters for our on-policy and off-policy search. In detail, the size n of the population is 10. The number of evaluation episodes to compute the cost-oriented fitness tuple is set to 15 as the resultant normalized fitness was comparable with the one collected using more evaluation episodes. Our mutation operator applies to all the agent weights, and the baseline Gaussian noise $\mathcal{G}$ has zero-mean and standard deviation 0.1. To compute the Safe Mutations, we sample a mini-batch of 250 unsafe states from the cost-buffer, setting the sensitivity threshold λ_{max} = 0.01. After evaluating different seeds in the Goal environments, these values were chosen and shared for all the experiments. For the Estimated Verification, we use $m = 60$ samples as they showed comparable performance over formal verification. We set the discretization value $\phi = 0.001$ to reflect the precision of the lidar of a real robotic platform (i.e., the LDS-01 of the TurtleBot3).

10.5 Empirical Evaluation

For each task, we consider the following plots:

1. Average reward.
2. Auxiliary cost with a dashed line for the cost threshold of constrained DRL (for a fair comparison, the threshold is the cost reached by SOS at convergence).
3. Pareto frontier of return versus cost at convergence, which drastically improves (up and to the left) with SOS.

Figure 10.2 shows the results in five tasks (arranged in columns) for the complete SOS-PPO implementation. This uses Estimated Verification to compute the violation metric for the selection, by verifying $p_{\uparrow}$, $p_{\rightarrow}$, $p_{\leftarrow}$ using the cost reachability set (Definition 10.2).

For the baselines, we obtain the same trend of prior work (Ray et al., 2019), where the objective of maximizing the reward while limiting the cost present a meaningful trade-off:

- PPO obtains high returns by taking unsafe actions.
- L-PPO is the most reliable in enforcing the constraint but typically achieves poor returns.
- CPO achieves attractive rewards due to its approximation errors, which prevent it from satisfying the constraints.
- IPO returns a similar result to CPO but has the advantage of being a first-order method.

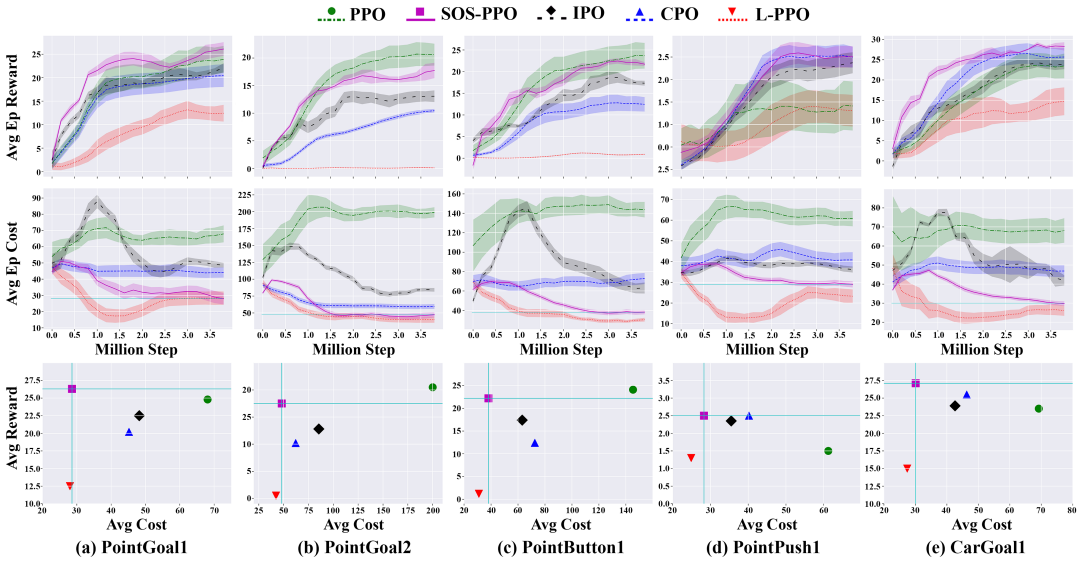

FIGURE 10.2: Comparison of PPO, SOS-PPO, IPO, CPO, L-PPO in Safety Gym. Each column (i.e., each task) shows the average reward and cost during the training, and Pareto frontier at convergence.

- Conversely, SOS-PPO successfully maintains comparable returns over PPO while drastically reducing the long-term cost, attaining cost values similar to L-PPO at convergence.

A detailed analysis of the returns of L-PPO highlights that its constrained policies are prone to perform poor behaviors (e.g., L-PPO often learns to stand still or move in circles for entire epochs to avoid collisions). This further motivates the introduction of the EV in the training loop to characterize the policies' behavior and select the safest individual using our violation metric.

Considering the limitation of formal verification, we note that writing properties for the DoggoGoal1 task without knowing the kinematics of the robot is not trivial. For this reason, Figure 10.3 shows the results of a SOS-PPO implementation without the Estimated Verification part. Hence, in these experiments, the best individual $\theta_* \in \mathcal{P}'$ is the one that has minimum cost.

In this environment, we note that L-PPO maintains the imposed cost limit but fails at learning the locomotion for this complex 12-joint robot. In contrast, our approach achieves comparable rewards over CPO and significantly reduces costs, similar to previous results.

10.5.1 Ablation Studies and Additional Experiments

We present a set of additional and ablation experiments to confirm our claims on the superior performance of SOS-based approaches and their components.

Mapless Navigation Additional Experiments

For a more comprehensive evaluation of our work, here we discuss and report the results of SOS in the indoor mapless navigation task. The setup is the same as prior

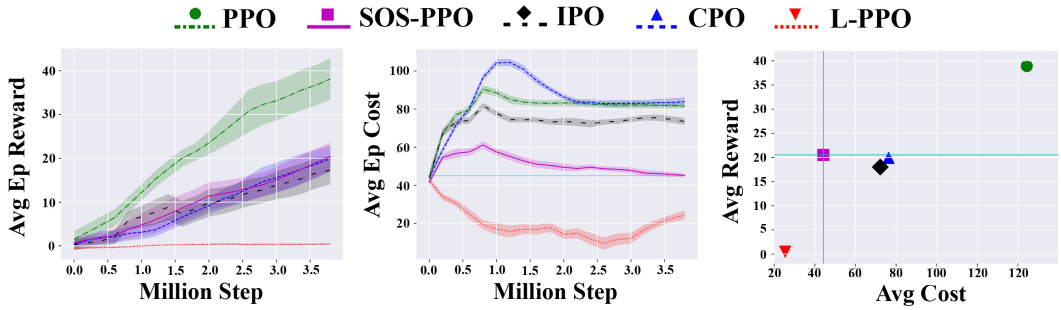

FIGURE 10.3: Comparison in the DoggoGoal task with a SOS-PPO implementation that selects the best individual in $\mathcal{P}'$ that has minimum cost instead of using the EV part.

experiments in this domain. A simulated TurtleBot3 has to learn how to navigate an indoor environment with obstacles to reach random targets, using only local observations (e.g., laser scans). The cost function triggers upon collision with an obstacle, and the logical properties of the verification part are comparable to the ones previously considered.

Figure 10.4 shows the Pareto frontier of reward versus cost at convergence on the right. Crucially, it significantly improves (up and to the left) with SOS, further confirming the results of our work.

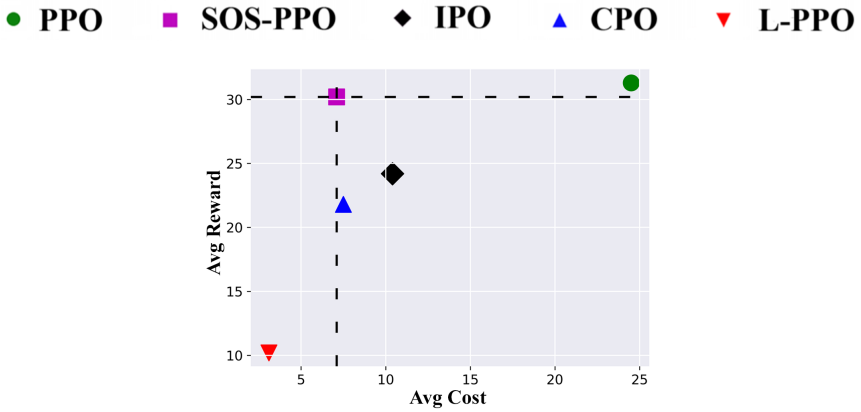

FIGURE 10.4: Comparison in our robotic mapless navigation task. Pareto frontier of average reward versus average cost at convergence.

Off-Policy Additional Experiments

We performed additional experiments in PointGoal1 and PointGoal2 to show that SOS can also be combined with off-policy Deep Reinforcement Learning. Figure 10.5 confirms the results observed in the on-policy setting, where SOS-TD3 achieves a similar reward to the original TD3, while correctly biasing the policy toward safer regions (i.e., with lower cost). We note that it could be possible to augment the DRL agent buffer with the population's diverse experiences to improve sample efficiency further.

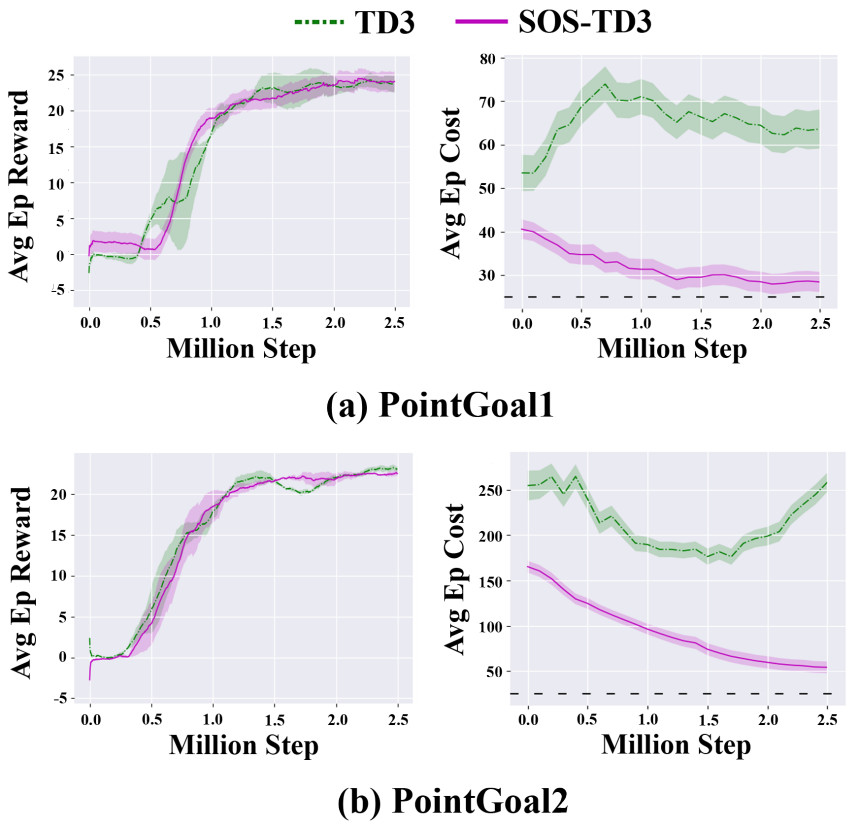

FIGURE 10.5: Comparison with TD3 and its SOS implementation (SOS-TD3) in PointGoal1 (a) and PointGoal2 (b). For each task we report the average reward and average cost. SOS-TD3 preserve the reward, while minimizing the auxiliary cost even in the off-policy scenario.

Safe Mutation and Estimated Verification

We further analyze SOS to show that:[4]

- Safe Mutations avoids disruptive changes to the policy while biasing exploration towards safety.
- The selection based on Estimated Verification successfully characterizes the policies' behaviors, resulting in better performance.

Figure 10.6 shows the plots generated by fitting a Gaussian kernel density model on the action selected over several epochs. This explanatory overview reports the behaviors (i.e., chosen actions) of the DRL policy and two perturbed versions with SM and simple Gaussian noise $\mathcal{G}$. The first row shows the effects of the mutations in the first stages of the training, the second shows the middle stages of the training (i.e., around 1.5 million steps), while the last shows the last epoch of the training phase of the PPO agent. SM locally biases the agent policy, maintaining a similar action distribution (i.e., behaviors). In contrast, Gaussian noise causes disruptive changes, resulting in very different and typically worst behaviors.

[4]Ablation studies are performed in PointGoal1 as we obtained similar results across the Safety Gym tasks.

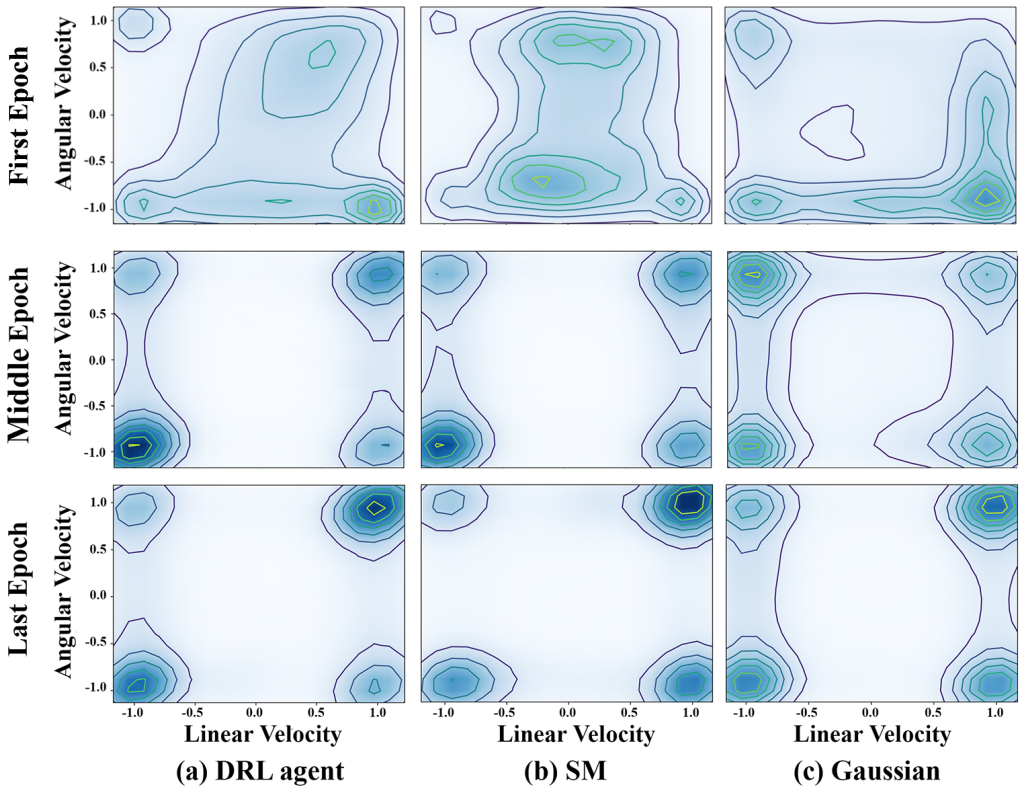

FIGURE 10.6: Overview of the DRL policy (a) and its mutation with SMs (b) or simple Gaussian noise (c) in the first, middle, and final epochs of the training (depicted in the first, second and third row, respectively). SMs locally biases the agent policy, maintaining a similar action distribution.

This is further supported by the top of Figure 10.7, which shows the average reward and cost collected by two SOS-PPO implementations. In detail, such implementations use (i) Gaussian noise (i.e., without SM and the cost-buffer) and (ii) standard output-gradient mutations (i.e., with the sensitivity computed on random samples). Hence, these ablation studies have no information on the additional cost to generate the population and can not cope with the safety aspect of the tasks, resulting in lower returns and higher costs over PPO.

Moreover, Figure 10.7 on the bottom shows the same metric for a SOS-PPO implementation that selects $\theta_* \in \mathcal{P}$ without EV, i.e., it selects the individual with comparable reward to the Deep Reinforcement Learning agent and with minimum cost. Crucially, these results confirm the importance of our framework in the scenarios where prior knowledge can not be included for the design of the properties for Estimated Verification as it maintains superior performance over the baselines. However, we note that this supports our claims on the importance of integrating EV in the selection process to characterize the actual behaviors of the policies. In particular, EV allows discovering safer policies since the early stages of the training, improving both the maximization of the reward and the cost minimization.

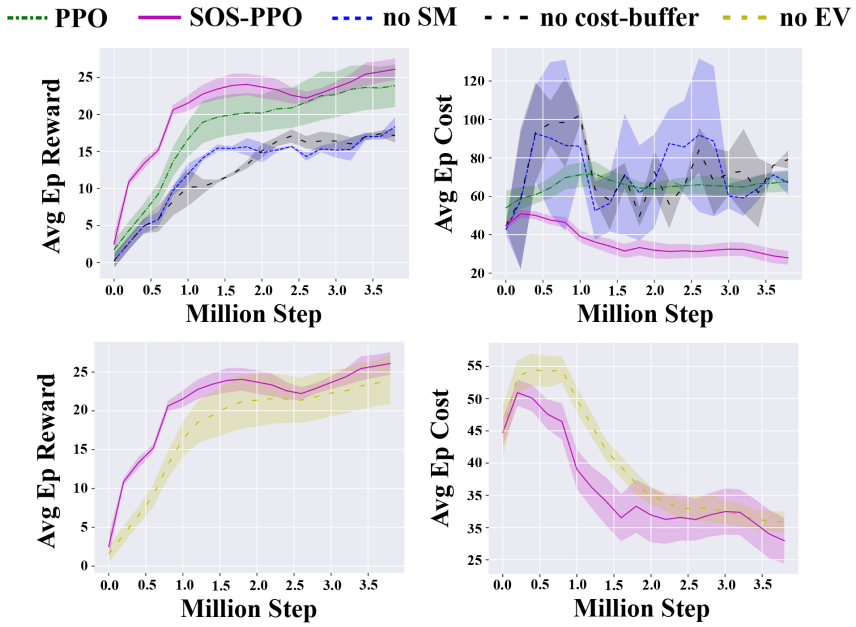

FIGURE 10.7: Ablation study of SOS-PPO implementations with (i) Gaussian mutations and (ii) output-gradient mutations without cost-buffer (top). Ablation study of SOS-PPO that selects the best individual that have minimum cost, instead of using EV (bottom).

Approximation of the Estimated Verification

We also analyze Estimated Verification to show that:

- The time required by prior Formal Verification approaches prevents their application in the training loop.
- EV provides comparable results (i.e., violation) over formal approaches.

Figure 10.8 on top shows the cumulative time required to verify our three properties using prior formal verification (i.e., ProVe (Corsi et al., 2021)), and EV with different sample sizes m. We remark that the analysis is periodically performed in the training loop for each safety-oriented search. Crucially, EV drastically reduces the computational time by more than 73.8× for each verification. Hence, in our experiments, Estimated Verification is the only feasible solution as the average training time of SOS in PointGoal1 requires a couple of hours, while it would require more than 48 hours using formal verification.

Finally, our estimation also returns similar results over formal verification due to the comparable violation results. Figure 10.8 on the bottom shows an explanatory computation (during the training) of the violation metric for our three safety properties. Note that we only report the curve of EV that uses the cost reachability set (Definition 10.2) as it has comparable performance over the estimated reachability set (Definition 10.1) in approximately half the time. Results highlight that with the bit of tuning of the sample size m, we obtain an accurate estimation of the violation (i.e., in our experiments, with $m = 60$ we have an error between the formal violation and the estimated one below 0.5%).

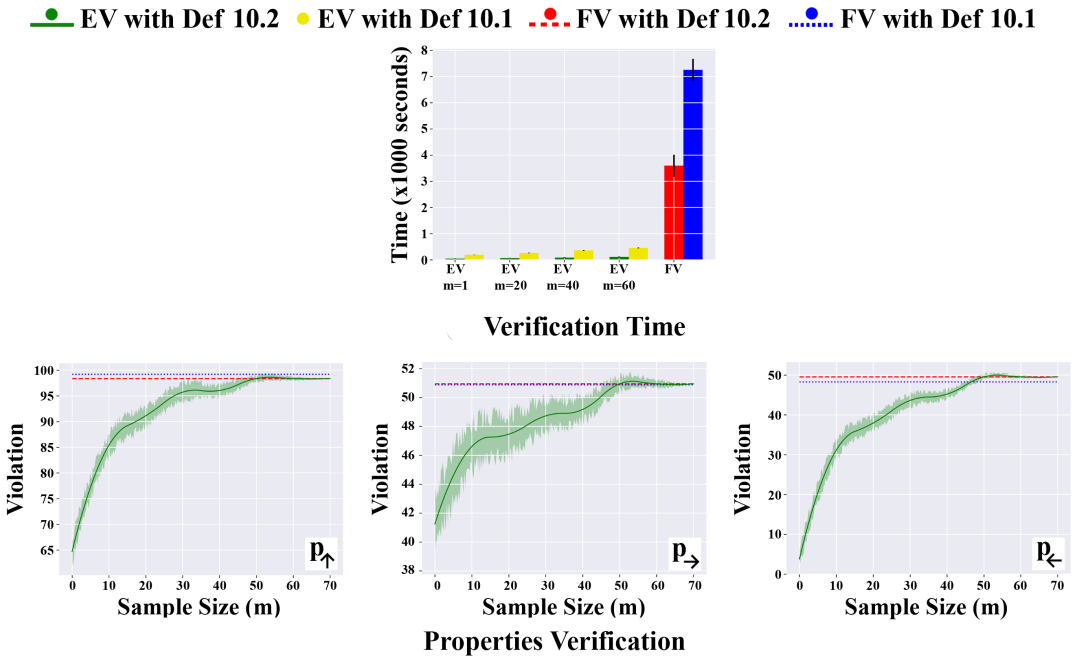

FIGURE 10.8: Comparison between formal analysis and the Estimated Verification: computation time (with different values of m for EV) (top). EV estimation of the violation metric over Formal Verification (bottom).

Formal Definition of Safety Properties

Here we report the proper encoding of the properties used by EV and formal verification in the Safety Gym tasks and in our evaluation. Notice that for some properties we could be interested in the verification of the relationship between an output node $y_i = [\underline{y_i}, \overline{y_i}]$, and a constant value c. For example, the Point robot has two outputs that control linear and angular velocities. In this scenario, the steering action depends on the sign of a single output value (e.g., if < 0 turn left, if > 0 turn right). Hence, we use our property formalization 5.1 by setting the target output to $y_j = [c, c]$, where c is the desired constant:

Considering the observation space of the tasks and the natural language description of the properties in the main paper, we provide an explanatory subset of the considered properties: $p_{\uparrow}$ and $p_{\rightarrow}$ for PointGoal and $p_{\leftarrow}$, for CarGoal:

$$
\begin{aligned}
&p_{\uparrow,\text{PointGoal}} : \text{If } x_0, \ldots, x_2 \in [-4, 10] \wedge x_3, \ldots, x_{18} \in [0, 1] \wedge x_{19}, \ldots, x_{21} \in [-3, 3] \wedge \\
&x_{22}, x_{23} \in [0.7, 1] \wedge x_{24}, \ldots, x_{35} \in [0, 1] \wedge x_{36}, x_{37} \in [0.7, 1] \wedge x_{38}, \ldots, x_{40} \in [-0.5, 0.5] \\
&\wedge x_{41}, \ldots, x_{56} \in [0, 1] \wedge x_{57}, \ldots, x_{59} \in [-0.7, 0.7] \\
&\Rightarrow y_1 < [-0.2, -0.2] \vee y_1 > [0.2, 0.2] \vee y_0 < [0, 0]
\end{aligned}
$$

$p_{\leftarrow,\text{PointGoal}}$: If $x_0, \dots, x_2 \in [-4, 10] \land x_3, \dots, x_{18} \in [0, 1] \land x_{19}, \dots, x_{21} \in [-3, 3] \land$
$x_{24}, \dots, x_{27} \in [0.7, 1] \land x_{28}, \dots, x_{37} \in [0, 1] \land x_{38}, \dots, x_{40} \in [-0.5, 0.5] \land$
$x_{41}, \dots, x_{56} \in [0, 1] \land x_{57}, \dots, x_{59} \in [-0.7, 0.7]$
$\Rightarrow y_1 < [-0.2, -0.2] \lor y_0 < [0, 0]$

$p_{\rightarrow,\text{CarGoal}}$: If $x_0, \dots, x_2 \in [-13, 13] \land x_3, \dots, x_5 \in [-9, 9] \land x_6, \dots, x_{15} \in [-1, 1] \land$
$x_{15}, \dots, x_{30} \in [0, 1] \land x_{31}, \dots, x_{34} \in [-3, 3] \land x_{35}, \dots, x_{50} \in [0, 1] \land$
$x_{44}, \dots, x_{47} \in [0.7, 1] \land x_{51}, \dots, x_{53} \in [-0.5, 0.5] \land x_{54}, \dots, x_{69} \in [0, 1] \land$
$x_{70}, \dots, x_{72} \in [-1, 1]$
$\Rightarrow y_0 > y_1 \lor (y_0 < [0, 0] \land y_1 < [0, 0])$

Notice that the prior knowledge in the input domain is included by measuring the minimum distance required to avoid an obstacle at max speed for the different robots. This is similar to Chapters 5, 9.

10.6 Related Work

Safety critics (Thananjeyan et al., 2020; Bharadhwaj et al., 2021; Thananjeyan et al., 2021) rely on estimating the probability of incurring into unsafe states, given a state-action pair. However, such approaches could potentially return misleading information for policy improvement, especially in the early stages of the training, where safety critics are pre-trained on offline data. Such offline demonstration represents unsafe samples which should cover a wide variety of unsafe behaviors for a robust pre-training. This may be challenging, and a possible alternative is to use data collected by human policies or human supervision. Moreover, these methods also introduce overhead in the action sampling process. Each step has to compute different samples (e.g., the action, the probability of failure), which can hinder their application to the physical hardware that requires high-frequency control.

In contrast, we compare SOS with constrained Deep Reinforcement Learning as it is more related to our approach. In more detail, CPO (Achiam et al., 2017) has near-constrained satisfaction guarantees, but the Taylor approximations lead to inverting a Fisher matrix, possibly resulting in infeasible updates and demanding recovery steps. Similarly, PCPO (Yang et al., 2020) also has theoretical guarantees for constraint satisfaction but uses second-order approximations and has mixed improvements over CPO (Zhang et al., 2020). Lyapunov-based algorithms (Chow et al., 2018, 2019), in contrast, combine a projection step with action-layer interventions, similarly to the safety layer of (Dalal et al., 2018). However, the cardinality of Lyapunov constraints equals the number of states, resulting in a non-negligible implementation cost.

Lagrangian methods (Ray et al., 2019; Stooke et al., 2020) reduce the complexity of prior approaches, with promising constraints satisfaction. These methods transform the equality-constrained problem defined over a real vector $\mathbf{x}$:

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t.} \quad g(\mathbf{x}) = 0$$

with a dual variable that form the Lagrangian:

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + l_\lambda g(\mathbf{x})$$

where l_λ is the Lagrange multiplier. Gradient-based algorithms then iteratively update the primal and dual variables:

$$\begin{aligned} -\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, l_\lambda) &= -\nabla_{\mathbf{x}} f(\mathbf{x}) - l_\lambda \nabla_{\mathbf{x}} g(\mathbf{x}) \\ \nabla_{l_\lambda} \mathcal{L}(\mathbf{x}, l_\lambda) &= g(\mathbf{x}) \end{aligned}$$

where l_λ acts as a learned penalty and is used to satisfy the constraint. This adapts to the constrained setting (Geibel and Wysotzki, 2005; Altman, 1998) representing a well-known constrained DRL approach due to its simplicity and good cost-limit satisfaction (Ray et al., 2019; Stooke et al., 2020). Similarly, IPO (Liu et al., 2020) reduces the constrained problem into an unconstrained one by augmenting the objective with logarithmic barrier functions, which provide sub-optimal solutions.

García and Fernández (Garcıa and Fernández, 2015) shows that constrained approaches have several drawbacks, such as the careful tuning of the threshold h as high values mean that they are too permissive, or conversely, too restrictive. Furthermore, such approaches rely on the strong assumption that typically can not be satisfied in practice (e.g., having an optimal policy) to provide theoretical guarantees on the constraints' satisfaction. Hence, constrained Deep Reinforcement Learning is also not devoid of short-term fatal consequences as empirical evidence shows that they typically fail at satisfying the imposed constraints (Ray et al., 2019), which is also related to the non-linear approximation nature of Deep Neural Networks.

Furthermore, constraints naturally limit exploration, causing getting stuck in local optima or failing to learn properly (Conti et al., 2018; Hong et al., 2018). In contrast, we leverage Evolutionary Algorithms to design SOS as prior combinations of DRL and EAs show a beneficial transfer of information between the two approaches (Khadka and Tumer, 2018; Khadka et al., 2019; Marchesini et al., 2021). These methods, however, use the evolutionary component only for improving the return and can not be trivially extended to address the safety component.

10.7 Discussion

We summarize our contributions in Safety-Oriented Search, a framework that combines Evolutionary Algorithms and Deep Reinforcement Learning using novel concepts of Safe Mutations and Estimated Verification to minimize an auxiliary cost signal to improve the policy safety while preserving the return. In detail:

- SM proposes the design of an informed mutation operator that preserves the policy while biasing exploration towards the desired objective (e.g., safety).
- EV enables to characterize the behaviors during the training, providing a significant speedup for the verification process, with comparable performance over prior formal verification.

Safety-Oriented Search is compatible with on-policy and off-policy Deep Reinforcement Learning. Our results in the Safety Gym benchmarks confirm that SOS successfully addresses the trade-off between return and cost, achieving comparable returns to unconstrained algorithms and comparable cost values to constrained DRL.

SOS has several potential impacts on society as it addresses safety, a crucial aspect of practical Deep Reinforcement Learning applications. While the SMs show that it is possible to augment exploration toward the desired objective and successfully transfer beneficial information into a DRL agent, the Estimated Verification can characterize the behaviors of a policy into the training loop. Hence it could be employed to design Safe Deep Reinforcement Learning algorithms. Nonetheless, the broader field of network verification, to which EV belongs, also presents negative consequences as an incorrect formalization of the properties could result in undesired behaviors.

Chapter 11

Conclusions and Future Work

In this thesis, we considered Deep Reinforcement Learning approaches aiming at improving efficiency and safety. Specifically, we proposed several methods to enhance the exploration abilities of the DRL agents. To conclude, we recap our contributions and discuss progress and frontiers in the field.

Benchmarking Exploration. A significant part of this thesis is devoted to discuss the importance that simulation environments have on the performance and development of Deep Reinforcement Learning algorithms. In the literature, the discussion on simulation appears underdeveloped over the algorithmic counterpart. We believe that the importance of this topic is somewhat underestimated, and we argue that the quality of environments is of critical importance to foster progress in the field of DRL.

In this direction, we explored an alternative way to develop high-quality environments, focusing on tasks where the agents' uncertainty and environment dynamics can genuinely represent a complex challenge for modern solutions. This follows the criticism on the simplicity of standard evaluation benchmarks discussed by Mania et al. (2018), which showed how benchmarks environments are not as complex as previously thought, obtaining state-of-the-art performance using random searches of linear policies. Hence, we presented three robotics-based domains to provide a more comprehensive set of tasks to evaluate the following contributions. In detail, we developed the robotic scenarios using Unity, a physically realistic simulation environment known for its applications to the game developing world. Crucially, we showed that Unity is a viable alternative over more computationally demanding solutions. It allows exporting the trained policies on the real robots without additional training or tuning. The increasing employment of Unity as a simulation environment for drl research further confirmed our intuition.[1].

Intuitively, future work in this domain involves developing novel simulation software that represents real challenges to open questions in the Deep Reinforcement Learning field. In this context, multi-agent applications represent an ideal domain, given the wide range of open problems in this field. The complexity of Multi-Agent (Deep) Reinforcement Learning translates into several different benchmark environments. Each developed explicitly for a particular challenge. For example, game-based settings (e.g., Starcraft) are the benchmark de facto to test cooperative MARL algorithms. However, these algorithms can not typically cope with different MARL open problems such as communication among agents. Hence, our vision is to develop a suite of environments of increasing difficulty, each comprising a multitude of open problems for Multi-Agent

[1] www.github.com/Unity-Technologies/Unity-Robotics-Hub

(Deep) Reinforcement Learning. We intend to exploit existing board games to this end, following the recent interesting insights given by the Hanabi challenge (Bard et al., 2019).

Benchmarking Sample Efficiency. Along a similar direction, we then argued the importance of choosing a particular algorithm, depending on task-related requirements.

To this end, we exploited the previous robotic environments to benchmark Deep Reinforcement Learning algorithms and present further optimization to enhance their performance.

We note that the incrementalist perspective of improving on an existing approach may be less appealing for a researcher. However, re-implementing and improving prior work exposes new researchers to how existing algorithms are brittle and could be improved. Furthermore, while it is hard to suggest directions for future work in this context, some of the most impressive achievements in machine learning have come from this incremental vision (Achiam, 2021).

Evaluating Decision-Making. Chapter 5 concludes Part I of the thesis discussing the importance of evaluation metrics in Deep Reinforcement Learning.

Our vision follows recent work that argued how choosing informative metrics is fundamental to determine whether algorithmic improvements are meaningful (Henderson et al., 2018). Hence, we analyzed this problem under the decision-making point of view, as standard evaluations based on reward and number of successes do not convey enough information to assess the decision-making skills of DRL policies. To this end, we proposed the use of prior Formal Verification tools to formally verify the behaviors of a policy in (possibly) the entire state space over desired specifications.

In this context, Colas et al. (2019) provides a starting point towards a standardization of the procedures for Deep Reinforcement Learning evaluations. However, we believe researchers should put a considerable effort towards this topic to ensure a shared baseline to evaluate novel approaches. Standardization would also provide a sound and objective way to properly assess various DRL methods, which is a crucial issue given the increasing number of Deep Reinforcement Learning approaches that are developed by the research community.

Combining Deep Reinforcement Learning with Evolutionary Algorithms. Part II of the thesis then focuses on the issues of exploring high-dimensional spaces, which is a crucial challenge both in single and multi-agent applications.

In this direction, Chapters 6 and 7 highlight the dual perspective between gradient-based Deep Reinforcement Learning and gradient-free Evolutionary Algorithms. In particular, we proposed to augment the broad family of DRL algorithms with a population search aimed at enhancing exploration while improving the robustness of the policies. Our framework improved over prior work in terms of performance and applicability. It showed promising performance, generalization skills, and robustness when applied to value-based, policy-gradient, and actor-critic solutions.

We discuss possible future directions on this topic in what follows, where we further employ Evolutionary Algorithms to achieve different objectives.

Global Dueling Q-Learning. We then analyzed the issues related to exploration in Multi-Agent (Deep) Reinforcement Learnings.

In detail, we analyzed the limitations induced by prior work to factorize a global action-value function and favor cooperation at the expense of limiting the agents' exploration. To this end, we proposed a different perspective that employs the insights of Dueling Networks (Wang et al., 2016) to avoid structural constraints for the factorization while fostering cooperative behaviors. We further extended this approach showing that evolutionary searches represent an efficient alternative way to learn different skills concurrently, improving sample efficiency and performance.

The policy decomposition and the separate estimation of the state values open several directions for future work. For example, we note that the insights of prior work on value decomposition networks (Sunehag et al., 2018; Rashid et al., 2018, 2020) could be applied to our network architecture to foster cooperation further. Moreover, our decomposition allows us also to explore competitive multi-agent setups, which is another exciting field of research. Nonetheless, our primary goal for the future involves the previously discussed design of a comprehensive suite of multi-agent environments based on complex board games to foster the development of efficient multi-agent approaches that are generally applicable to (possibly) any settings. In our vision, such general methodologies would deal with the different multi-agent challenges (e.g., communication, cooperation, credit assignment, partial information) within a single solution.

Quantifying Safe Behaviors. The design of an evaluation metric in safety-critical contexts begins Part III. This is crucial as practical applications, such as robotics usually involve high-cost hardware or human cooperation. Hence, the behavior of a Deep Reinforcement Learning policy must be evaluated to avoid unsafe situations.

We formally defined a violation metric to quantify the number of unsafe decisions that a trained policy chooses over safety specifications defined a priori. To this end, we employed Formal Verification to compute our violation results and confirm the beneficial effects of Evolutionary Algorithms to gradient-based DRL.

Safety-Oriented Search. Finally, Chapter 10 ends the thesis contributions by analyzing the problems of safety and efficient exploration into a unique framework.

In this direction, we argued the limitations of prior work on Safe Deep Reinforcement Learning that typically employ constrained optimization to guarantee safety. However, the non-linear approximation nature of Deep Neural Networks hinders the satisfaction of such constraints and results in a significant trade-off between safety and return. Hence, we proposed a different perspective based on safe-informed evolutionary operators and approximated verification to bias a gradient-based Deep Reinforcement Learning policy to explore safer behaviors. Crucially, our contribution allowed us to obtain a comparable return over the DRL component while significantly improving the overall safety of the resultant policy.

Considering the promising results on the variety of tasks we considered for our combination of Deep Reinforcement Learning and Evolutionary Algorithms, we believe this represents another promising field for future developments. In particular, the objective-informed evolutionary operator could be a critical component for the design of the general MARL approach due to its significant performance in biasing the policy toward the desired objective, without explicit multi-objective optimization. Moreover, the approximated version of Formal Verification enables to characterize the decision-making process of an agent in an online fashion.

Overall, we believe that novel Safe DRL solutions with formal safety guarantees is an exciting future perspective. In particular, we argue that the trade-off related to developing effective and safe Deep Reinforcement Learning methods require a paradigm shift in the way such methods are designed and evaluated. We believe that the combination of sample efficient DRL approaches and Formal Verification, proposed in this thesis, is a first important step in this direction. Nonetheless, a significant amount of research work is required to achieve this long-term vision.

Acronyms

ALE Arcade Learning Environment. 10, 29

AMCL Adaptive Monte Carlo Localization. 37

CEM Cross Entropy Method. 64

CMDP Constrained Markov Decision Process. 23, 109, 110

CNN Convolutional Neural Network. 14

CTDE Centralized Training Decentralized Execution. 84–86, 93, 99, 100

D-H Denavit-Hartenberg. 30

DDPG Deep Deterministic Policy Gradient. 36, 40

Dec-POMDP Decentralized Partially Observable Markov Decision Process. 85

DNN Deep Neural Network. 1, 3, 5, 7–9, 13–15, 20, 22, 44, 49–51, 58, 70, 95, 104, 110, 111, 113, 124, 129

DQN Deep Q-Network. 19–21, 40, 64, 66

DRL Deep Reinforcement Learning. 1–11, 13, 18–20, 22, 23, 27–33, 35–40, 42, 46, 47, 49, 50, 53, 54, 58, 59, 63–67, 69–75, 79–82, 84, 85, 94, 96, 100, 103, 107, 109–115, 118, 120, 123–125, 127–130

EA Evolutionary Algorithm. 2, 3, 6, 8–11, 13, 18, 22, 23, 63, 65, 67, 69, 100, 109, 110, 124, 128, 129

ERL Evolutionary Reinforcement Learning. 64, 67

ES Evolutionary Strategy. 65

EV Estimated Verification. 110, 114–117, 119–121, 124, 125

FV Formal Verification. 2, 3, 5–9, 11, 13, 23, 49, 50, 59, 110, 121, 128–130

GA Genetic Algorithm. 3, 4, 13, 18, 19, 69, 70, 72, 74, 75, 82, 135

GD Gradient Descent. 14, 15

GDQ Global Dueling Q-learning. 84, 85, 96, 99, 100

GPU Graphics Processing Unit. 53

IGM Individual Global Max. 86

LSTM Long Short-Term Memory. 84

MADDPG Multi-Agent DDPG. 86

MARL Multi-Agent (Deep) Reinforcement Learning. 3, 6, 9, 10, 16, 83–86, 89, 90, 92–96, 100, 127–129

MDP Markov Decision Process. 16, 40, 50, 110

MLP Multi-Layer Perceptron. 14

MORL Multi-Objective (Deep) Reinforcement Learning. 4

NLP Natural Language Processing. 13

NN Neural Network. 13

PDERL Proximal Distilled ERL. 64, 67

PER Priority Experience Replay. 46, 87

POMDP Partially Observable Markov Decision Process. 16, 85

PPO Proximal Policy Optimization. 36, 41

RL Reinforcement Learning. 1, 2, 13, 15, 16, 18, 19, 22, 40, 50, 83, 109, 135

RNN Recurrent Neural Network. 14

ROS Robot Operating System. 5, 8, 28, 35

SGD Stochastic Gradient Descent. 15, 74, 76, 80

SM Safe Mutation. 109–111, 113, 116, 119, 124

SOS Safety-Oriented Search. 109

TRPO Trust Region Policy Optimization. 41

List of Figures

List of Tables

Bibliography

Achiam, J. (2021). *Exploration and Safety in Deep Reinforcement Learning*. PhD thesis, EECS Department, University of California, Berkeley.

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *International Conference on Machine Learning (ICML)*.

Alex Irpan (2018). Deep Reinforcement Learning Doesn't Work Yet.

Altman, E. (1998). Constrained markov decision processes with total cost criteria: Lagrangian approach and dual linear program. In *Mathematical methods of Operations Research*.

Altman, E. (1999). Constrained markov decision processes. In *CRC Press*.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning (ICML)*.

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., Dunning, I., Mourad, S., Larochelle, H., Bellemare, M. G., and Bowling, M. (2019). The hanabi challenge: A new frontier for AI research. In *arXiv*.

Barto, A., Mirolli, M., and Baldassarre, G. (2013). Novelty or surprise? In *Frontiers in Psychology*.

Baxter, J. L., Burke, E. K., Garibaldi, J. M., and Norman, M. (2007). Multi-robot search and rescue: A potential field based approach. In *Autonomous Robots and Agents*.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., and Petersen, S. (2016). Deepmind lab. In *arXiv*.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A Distributional Perspective on Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

Bellman, R. (1957). A markovian decision process. In *Indiana University Mathematics Journal*.

Bharadhwaj, H., Kumar, A., Rhinehart, N., Levine, S., Shkurti, F., and Garg, A. (2021). Conservative safety critics for exploration. In *International Conference on Learning Representations (ICLR)*.

Bodnar, Ben Day, P. L. (2020). Proximal Distilled Evolutionary Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. In *arXiv*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Bunel, R., Turkaslan, I., Torr, P. H. S., Kohli, P., and Kumar, M. P. (2017). A unified view of piecewise linear neural network verification. In *arXiv*.

Castellini, A., Bloisi, D., Blum, J., Masillo, F., and Farinelli, A. (2020). Multivariate sensor signals collected by aquatic drones involved in water monitoring: A complete dataset. In *Data in Brief*.

Chen, C.-T. and Chang, W.-D. (1996). A feedforward neural network with function shape autotuning. In *Neural Networks*.

Chiang, H. L., Faust, A., Fiser, M., and Francis, A. (2019). Learning Navigation Behaviors End to End. In *RA-L*.

Chow, Y., Nachum, O., Duenez-Guzman, E., and Ghavamzadeh, M. (2018). A lyapunov-based approach to safe reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Chow, Y., Nachum, O., Faust, A., Ghavamzadeh, M., and Duéñez-Guzmán, E. A. (2019). Lyapunov-based safe policy optimization for continuous control. In *International Conference on Machine Learning (ICML)*.

Codd-Downey, R. and Jenkin, M. (2017). On the utility of additional sensors in aquatic simultaneous localization and mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Codevilla, F., Santana, E., Lopez, A. M., and Gaidon, A. (2019). Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Colas, C., Sigaud, O., and Oudeyer, P. (2018). GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms. In *International Conference on Machine Learning (ICML)*.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. (2019). A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. In *arXiv*.

Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K. O., and Clune, J. (2018). Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Corsi, D., Marchesini, E., and Farinelli, A. (2020). Evaluating the safety of deep reinforcement learning models using semi-formal verification. In *arXiv*.

Corsi, D., Marchesini, E., and Farinelli, A. (2021). Formal verification of neural networks for safety-critical tasks in deep reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Corsi, D., Marchesini, E., Farinelli, A., and Fiorini, P. (2020). Formal verification for safe deep reinforcement learning in trajectory generation. In *International Conference on Robotic Computing (IRC)*.

Cuccu, G., Togelius, J., and Cudré-Mauroux, P. (2019). Playing atari with six neurons. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., and Tassa, Y. (2018). Safe exploration in continuous action spaces. In *arXiv*.

de Wiele, T. V., Warde-Farley, D., Mnih, A., and Mnih, V. (2020). Q-learning in enormous action spaces via amortized approximate maximization. In *arXiv*.

Diederik P. Kingma, J. B. (2015). Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*.

Ding, Z., Lepora, N. F., and Johns, E. (2020). Sim-to-real transfer for optical tactile sensing. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Dorner, F. (2021). Measuring progress in deep reinforcement learning sample efficiency. In *arXiv*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking Deep Reinforcement Learning for Continuous Control. In *International Conference on Machine Learning (ICML)*.

Dulac-Arnold, G., Evans, R., Hasselt, H. V., Sunehag, P., Lillicrap, T. P., Hunt, J. J., Mann, T. A., Weber, T., Degris, T., and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. In *arXiv*.

Dutta, S., Jha, S., Sanakaranarayanan, S., and Tiwari, A. (2017). Output range analysis for deep neural networks. In *arXiv*.

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*.

Fogel, D. (2006). Toward a new philosophy of machine intelligence. In *Evolutionary computation 3. ed.*

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2017). Noisy Networks for Exploration. In *International Conference on Learning Representations (ICLR)*.

François-Lavet, V., Fonteneau, R., and Ernst, D. (2015). How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*.

Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. In *Journal of Machine Learning Research (JMLR)*.

Geibel, P. and Wysotzki, F. (2005). Risk-sensitive reinforcement learning applied to control under constraints. In *Journal of Artificial Intelligence Research (JAIR)*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Ha, D., Dai, A. M., and Le, Q. V. (2017). Hypernetworks. In *International Conference on Learning Representations (ICLR)*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning (ICML)*.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*.

Hong, Z., Shann, T., Su, S., Chang, Y., and Lee, C. (2018). Diversity-driven exploration strategy for deep reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Hunt, N., Fulton, N., Magliacane, S., Hoang, N., Das, S., and Solar-Lezama, A. (2021). Verifiably safe exploration for end-to-end reinforcement learning. In *HSCC*.

Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2018). Unity: A general platform for intelligent agents. In *arXiv*.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. In *Nature*.

Karapetyan, N., Moulton, J., Lewis, J. S., Quattrini Li, A., O'Kane, J. M., and Rekleitis, I. (2018). Multi-robot dubins coverage with autonomous surface vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. (2017). Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification*.

Khadka, S., Majumdar, S., Nassar, T., Dwiel, Z., Tumer, E., Miret, S., Liu, Y., and Tumer, K. (2019). Collaborative evolutionary reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Khadka, S. and Tumer, K. (2018). Evolutionary Reinforcement Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Kretzschmar, H., Spies, M., Sprunk, C., and Burgard, W. (2016). Socially compliant mobile robot navigation via inverse reinforcement learning. In *The International booktitle of Robotics Research*.

L. Busoniu, R. B. and Schutter, B. D. (2008). A comprehensive survey of multi-agent reinforcement learning. In *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*.

Lehman, J., Chen, J., Clune, J., and Stanley, K. O. (2018). Safe Mutations for Deep and Recurrent Neural Networks Through Output Gradients. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Lemarechal, C. (2012). Cauchy and the gradient method.

Levine, S. (2020). Cs 285 at uc berkeley: Deep reinforcement learning.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Liu, C., Arnon, T., Lazarus, C., Barrett, C., and Kochenderfer, M. J. (2019). Algorithms for verifying deep neural networks. In *arXiv*.

Liu, C., Xu, X., and Hu, D. (2015). Multiobjective reinforcement learning: A comprehensive overview. In *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Liu, Y., Ding, J., and Liu, X. (2020). IPO: interior-point policy optimization under constraints. In *AAAI Conference on Artificial Intelligence*.

Lomuscio, A. and Maganti, L. (2017). An approach to reachability analysis for feed-forward relu neural networks. In *arXiv*.

Long, P., Fan, T., Liao, X., Liu, W., Zhang, H., and Pan, J. (2018). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Lutjens, B., Everett, M., and How, J. P. (2020). Certified adversarial robustness for deep reinforcement learning. In *Conference on Robot Learning (CoRL)*.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*.

Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. (2019). MAVEN: multi-agent variational exploration. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Marchesini, E., Corsi, D., Benfatti, A., Farinelli, A., and Fiorini, P. (2019). Double Deep Q-Network for Trajectory Generation of a Commercial 7DOF Redundant Manipulator. In *International Conference on Robotic Computing (IRC)*.

Marchesini, E., Corsi, D., and Farinelli, A. (2021a). Benchmarking safe deep reinforcement learning in aquatic navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Marchesini, E., Corsi, D., and Farinelli, A. (2021b). Exploring safer behaviors for deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Marchesini, E., Corsi, D., and Farinelli, A. (2021). Genetic soft updates for policy evolution in deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Marchesini, E. and Farinelli, A. (2020a). Discrete deep reinforcement learning for mapless navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Marchesini, E. and Farinelli, A. (2020b). Genetic deep reinforcement learning for mapless navigation. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Marchesini, E. and Farinelli, A. (2021a). Centralizing state-values in dueling networks for multi-robot reinforcement learning mapless navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Marchesini, E. and Farinelli, A. (2021b). Enhancing deep reinforcement learning approaches for multi-robot navigation via single-robot evolutionary policy search. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Martin H., J. A. and de Lope, J. (2009). Learning autonomous helicopter flight with evolutionary reinforcement learning. In *Computer Aided Systems Theory*.

Marzari, L., Corsi, D., Marchesini, E., and Farinelli, A. (2021). Enhancing deep reinforcement learning mapless navigation using curriculum learning. In *Submitted at ACM/SIGAPP Symposium On Applied Computing (SAC)*.

Matheron, G., Perrin, N., and Sigaud, O. (2019). The problem with ddpg: understanding failures in deterministic environments with sparse rewards. In *arXiv*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013a). Playing Atari with Deep Reinforcement Learning. In *Workshop of Conference on Neural Information Processing Systems (NeurIPS)*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013b). Playing Atari with Deep Reinforcement Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Montana, D. and Davis, L. (1989). Training Feedforward Neural Networks Using Genetic Algorithms. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Moore, R. E. (1963). Interval arithmetic and automatic error analysis in digital computing. In *Stanford University*.

Oliehoek, F. A. and Amato, C. (2016). A concise introduction to decentralized pomdps. In *SpringerBriefs in Intellingece Systems*.

OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d. O., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. In *arXiv*.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019). Solving rubik's cube with a robot hand. In *arXiv*.

Oroojlooyjadid, A. and Hajinezhad, D. (2019). A review of cooperative multi-agent deep reinforcement learning. In *arXiv*.

Ostrovski, G., Bellemare, M., Oord, A., and Munos, R. (2017). Count-Based Exploration with Neural Density Models. In *International Conference on Machine Learning (ICML)*.

Oudeyer, P.-Y. and Kaplan, F. (2013). How can we define intrinsic motivation? In *HAL*.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016). The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroSP)*.

Pathak, D., Agrawal, P., Efros, A., and Darrell, T. (2017). Curiosity-Driven Exploration by Self-Supervised Prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

Polyak, B. T. and Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging.

Pore, A., Corsi, D., Marchesini, E., Dall'Alba, D., Casals, A., Farinelli, A., and Fiorini, P. (2021). Safe reinforcement learning using formal verification for tissue retraction in autonomous robotic-assisted surgery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Pourchot, A. and Sigaud, O. (2019). CEM-RL: Combining evolutionary and gradient-based methods for policy search. In *International Conference on Learning Representations (ICLR)*.

Raghunathan, A., Steinhardt, J., and Liang, P. (2018). Certified defenses against adversarial examples. In *arXiv*.

Rashid, T., Farquhar, G., Peng, B., and Whiteson, S. (2020). Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. (2018). QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Ray, A., Achiam, J., and Amodei, D. (2019). Benchmarking safe exploration in deep reinforcement learning. In *arXiv*.

Robbins, H. E. (2007). A stochastic approximation method. In *Annals of Mathematical Statistics*.

Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. (2014). A survey of multi-objective sequential decision-making. In *Journal of Artificial Intelligence Research (JAIR)*.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. In *arXiv*.

Sandy Huang, Nicolas Papernot, I. G. Y. D. and Abbeel, P. (2017). Adversarial attacks on neural network policies. In *International Conference on Learning Representations (ICLR)*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning (ICML)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. In *arXiv*.

Sepp Hochreiter, J. S. (1997). Long short-term memory. In *Neural Computation*.

Silver, D. (2020). Ucl course on rl.

Silver, D., Huang, A., Maddison, C., and et al (2018a). Mastering the game of go with deep neural networks and tree search. In *Nature*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018b). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. In *Science*.

Simpson, G. G. (1953). The baldwin effect. In *Evolution*.

Singh, G., Gehr, T., Püschel, M., and Vechev, M. (2019). An abstract domain for certifying neural networks. In *ACM Program. Lang.*

Son, K., Kim, D., Kang, W. J., Hostallero, D., and Yi, Y. (2019). QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Stooke, A., Achiam, J., and Abbeel, P. (2020). Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning (ICML)*.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. In *CoRR*.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. (2018). Value-decomposition networks for cooperative multi-agent learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Szegedy, C., Goodfellow, I., and Shlens, J. (2015). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*.

Tai, L., Li, S., and Liu, M. (2016). A deep-network solution towards model-less obstacle avoidance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Tai, L., Paolo, G., and Liu, M. (2017). Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V. (2018). Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. In *Robotics: Science and Systems (RSS)*.

Tavakoli, A., Pardo, F., and Kormushev, P. (2018). Action Branching Architectures for Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*.

Tessler, C., Merlis, N., and Mannor, S. (2019). Stabilizing off-policy reinforcement learning with conservative policy gradients. In *arXiv*.

Thananjeyan, B., Balakrishna, A., Nair, S., Luo, M., Srinivasan, K., Hwang, M., Gonzalez, J. E., Ibarz, J., Finn, C., and Goldberg, K. (2021). Recovery rl: Safe reinforcement learning with learned recovery zones. In *RA-L*.

Thananjeyan, B., Balakrishna, A., Rosolia, U., Li, F., McAllister, R., Gonzalez, J. E., Levine, S., Borrelli, F., and Goldberg, K. (2020). Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks. In *RA-L*.

Tjeng, V., Xiao, K., and Tedrake, R. (2017). Evaluating robustness of neural networks with mixed integer programming. In *arXiv*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Tuyls, K. and Weiss, G. (2012). Multiagent learning: Basics, challenges, and prospects. In *AI Magazine*.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. (2011). Empirical evaluation methods for multiobjective reinforcement learning algorithms. In *Machine Learning*.

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep reinforcement learning and the deadly triad. In *Conference on Neural Information Processing Systems (NeurIPS)*.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. In *AAAI Conference on Artificial Intelligence*.

Vinyals, O., Babuschkin, I., Czarnecki, W., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J., Jaderberg, M., and Silver, D. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. In *Nature*.

Wahid, A., Toshev, A., Fiser, M., and Lee, T. E. (2019). Long range neural navigation policies for the real world. In *CoRR*.

Wang, P., Chan, C., and de La Fortelle, A. (2018a). A reinforcement learning based approach for automated lane change maneuvers. In *IEEE Intelligent Vehicles Symposium*.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. (2018b). Efficient formal safety analysis of neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. (2018c). Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*.

Wang, Z., de Freitas, N., and Lanctot, M. (2016). Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. In *Machine Learning*.

Weng, T.-W., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Boning, D., Dhillon, I. S., and Daniel, L. (2018). Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning (ICML)*.

Xiang, W., Tran, H., and Johnson, T. T. (2018). Output reachable set estimation and verification for multilayer neural networks. In *IEEE Transactions on Neural Networks and Learning Systems*.

Xiang, W., Tran, H.-D., and Johnson, T. T. (2017). Reachable set computation and safety verification for neural networks with relu activations. In *arXiv*.

Xie, L., Wang, S., Rosa, S., Markham, A., and Trigoni, N. (2018). Learning with Training Wheels: Speeding up Training with a Simple Controller for Deep Reinforcement Learning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Yang, R., Sun, X., and Narasimhan, K. (2019). A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Yang, T.-Y., Rosca, J., Narasimhan, K., and Ramadge, P. J. (2020). Projection-based constrained policy optimization. In *International Conference on Learning Representations (ICLR)*.

Yann LeCun, Y. B. (1995). Deep learning. In *The handbook of brain theory and neural networks*.

Zachary C. Lipton, J. S. (2018). Troubling trends in machine learning scholarship. In *International Conference on Machine Learning (ICML)*.

Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., and Mannor, S. (2019). Learn what not to learn: Action elimination with deep reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. (2018). Efficient neural network robustness certification with general activation functions. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Zhang, J., Springenberg, J. T., Boedecker, J., and Burgard, W. (2017). Deep reinforcement learning with successor features for navigation across similar environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Zhang, Y., Vuong, Q., and Ross, K. W. (2020). First order constrained optimization in policy space. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Zhao, W., Queralta, J. P., and Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *IEEE Symposium Series on Computational Intelligence*.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*.