Bachelor's Degree in Data Science and Engineering

Final Thesis Report
Thesis defense: July 1, 2021

Distributed Deep Reinforcement Learning in an HPC system and deployment to the Cloud

Author: Miquel Escobar Castells

Director: Jordi Torres Viñals
Co-director: Raúl García Fuentes

June 20, 2021

Abstract

Combining Reinforcement Learning and Deep Learning is the most challenging Artificial Intelligence research and development area at present. Scaling these types of applications in an HPC infrastructure available in the cloud will be crucial for advancing the massive use of these technologies. The purpose of this project is to develop and test various implementations of Deep RL algorithms given a selected case study, using Barcelona Supercomputing Center's (BSC) CTE-POWER cluster, and make a deployment to the cloud of the training pipeline to analyze its costs and viability in a production environment.

Keywords: Reinforcement Learning, RL, Deep Reinforcement Learning, DRL, HPC, cloud, Ray, AWS, SageMaker.

Resumen

Combinar el aprendizaje por refuerzo con el aprendizaje profundo es, a día de hoy, el desafío más grande en el sector de desarrollo e investigación en inteligencia artificial. Escalar este tipo de aplicaciones mediante supercomputadores o servicios en la nube es crucial para avanzar en el uso masivo de estas tecnologías. El objetivo de este proyecto es desarrollar y testear varias implementaciones de algoritmos de Deep Reinforcement Learning (DRL) sobre el caso de estudio seleccionado, usando el cluster CTE-POWER del Barcelona Supercomputing Center (BSC), así como hacer un despliegue a la nube de los entrenamientos para analizar sus costes y viabiliadad en un entorno de producción.

Palabras clave: aaprendizaje por refuerzo, RL, aprendizaje por refuerzo profundo, DRL, HPC, cloud, Ray, AWS, SageMaker.

Resum

Combinar l'aprenentatge per reforç amb l'aprenentatge profund és, a dia d'avui, un dels reptes més grans en el sector d'investigació en intel·ligència artificial. Escalar aquest tipus d'aplicacions mitjançant supercomputadors o serveis al núvol és crucial per avançar en l'ús massiu d'aquestes tecnologies. L'objectiu d'aquest projecte és desenvolupar i testejar vàries implementacions d'algorismes de Deep Reinforcement Learning (DRL) sobre el cas d'estudi seleccionat, utilitzant el clúster CTE-POWER del Barcelona Supercomputing Center (BSC), així com fer un desplegament al núvol dels entrenaments per analitzar els seus costos i viabilitat en un entorn de producció.

Paraules clau: aprenentatge per reforç, RL, aprenentatge per reforç profund, DRL, HPC, cloud, Ray, AWS, SageMaker.

Acknowledgements

I would first like to thank my director, Jordi Torres, for the proposal of this project and for giving me the opportunity to join such an incredible research group, as well as for your guidance all the way through.

I would also like to express my most sincere gratitude to my colleagues at the Emerging Technologies for Artficial Intelligence group: to Raúl, for your direct contributions and support during my time in the group; to Juan Luis and Oriol, who provided feedback and counsel along the way.

I would also like to thank my parents, my grandmother, my brother and sister for your unconditional support and wise counsel. Finally, I would like to end by thanking my friends Jordi A., Víctor, Pau, Pol, Jacobo, Lucas, Francesc, Jordi C., Arnau, and Lucía, for their encouragement and support all through my studies.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

The project is carried out at Barcelona Supercomputing Center (BSC-CNS), as part of the research in Reinforcement Learning conducted in the Emerging Technologies for Artificial Intelligence group. The investigation group has been focusing the last few years in the field of Deep Learning, and is now shifting into the study and application of the thriving area of Reinforcement Learning. In fact, as a direct result of the new line of research, the group manager recently published a book on the topic [38], which has been profoundly utilized for the development of this project.

The purpose of this project is to use BSC's CTE-Power cluster to develop and test various implementations of Deep Reinforcement Learning algorithms given a case study, and posteriorly perform a deployment to the cloud of the obtained models to analyze its costs and viability in a production environment.

1.2 Background

In order to allow determining the originality and scope of the contributions made by the project author, the origin of the main ideas is described in this section.

The project idea comes from the general line of research conducted in BSC's Emerging Technologies for Artificial Intelligence investigation department, which is focused on state-of-the-art Reinforcement Learning techniques, specifically in the scope of Deep Reinforcement Learning. Thus, the project initial ideas were mostly proposed by the group manager and director of this thesis.

The project in itself starts from scratch, even though the knowledge that the group has already acquired in the field of Reinforcement Learning, both in terms of theoretical and mathematical background, as well as in relation to various software frameworks, will take part in the development of the project.

1.3 Justification

Reinforcement Learning is a domain in Artificial Intelligence which has been gaining popularity among multiple fields, all the way from robotics to Natural Language Processing or self-driving cars. In fact, Deepmind [35] contemplates the possibility that all intelligence can always be represented as the maximization of reward, and that Reinforcement Learning is enough to reach a more human-like general AI.

Even so, in Reinforcement Learning it can be difficult to fit some problems and their data to the corresponding environment and agent/s, as these are extremely data-hungry, sensitive to hyperparameter tuning and relatively unstable. If not carefully managed, this can easily lead to overcomplicating simple problems.

Consequently, one of the major challenges that Deep Reinforcement Learning encounters is computation costs and training time, as very complex problems can require enormous amounts of iterations, executions and hyperparameter tuning. That is why, in order to tackle this, High

Performance Computing and cloud services come into play, as they can help reduce the computer costs and the training time of the more complex algorithms.

1.4 Objectives

The main goals of the project are listed below.

- Successfully train and test Deep Reinforcement Learning algorithms for the selected case study (also referred as environment).
- Implement and train the chosen algorithms in an HPC system (the CTE-Power cluster), mainly focusing on distributed and parallel computing.
- Perform a benchmarking in terms of results and training time in the CTE-Power cluster, comparing and analyzing the use of CPU, GPU and distributed computing.
- Develop and deploy a training pipeline using the chosen cloud service provider, and given the Reinforcement Learning environment.

1.5 Stakeholders

The main beneficiary of this work is Barcelona Supercomputing Center, more specifically the Emerging Technologies for Artificial Intelligence research group [4].

A secondary beneficiary is the Reinforcement Learning community, given that the code and results obtained from the work are public and available to the scientific community, containing clear implementations for different training methodologies in a supercomputer, as well as a complete pipeline for training and inference in the cloud.

2 Project planning

2.1 Methodology

In this section the methodology followed to implement the project is explained.

2.1.1 Management

In order to properly manage the course of the project, a minimum of one weekly meeting has taken place with the director and the entire team, where the progress of this week is exposed and any inquiries or proposals are discussed. There also exists constant feedback for any other issues by means of the various contact tools provided by BSC, such as Slack or the internal mail.

2.1.2 Development

The development of the project is done using a git repository, which is hosted by GitHub at `https://github.com/miquelescobar/ddrl`, in order to access from the CTE-POWER cluster node the modifications made locally, or the other way around. All training results (for both live and finished executions) are stored in CTE-POWER's node, and subdivided into the different environments, algorithms and hyperparameters. This data can be easily shared afterwards, as well as processed to analyze any necessary metrics and/or results. All the code corresponding to the Amazon SageMaker service can also be found in the repository.

2.1.3 Documentation

The documentation of the project can be differentiated into three different levels of abstraction, each one being resolved using a specific methodology:

1. **Work explanation**: in the highest abstraction level, there is the documentation related to the general purpose and characteristics of this work, as well as directions to all relevant resources. This can be found in this same report.
2. **Repository**: at a lower level, the documentation related to the code repository, hosted in GitHub, can be found in its README files, which contain all necessary information about the directories and files of the repository itself, as well as the corresponding instructions for reproducibility (from the installation of the dependencies to the execution of the code).
3. **Code**: at the lowest level, the code of the project has been documented with the use of docstrings (in all modules, classes and methods), as all of it is written in Python.

2.2 Risks

The potential risks of the project are listed below.

- **CTE-POWER cluster maintenances**: periodic maintenance breaks take place in the CTE-POWER cluster, preventing the users from accessing the machines. Thus, the breaks can delay some scheduled or prepared executions.
- **Insufficient resources assignation**: each of the trainings is executed within a limited resources allocation, that is, hardware limitations (a certain number of CPUS and/or GPUS). If, for instance, the required memory at a given moment is more than the one available, the execution will stop with an error. It is for that reason that the job's resource assignation must be calculated carefully.
- **Operating system incompatibilities**: the operating system running in CTE's machine is Red Hat Enterprise Linux Server 7.5 alternative, which might not be compatible with certain dependencies which might operate correctly in other operating systems.
- **Hyperparameters sensitivity**: most of Reinforcement Learning algorithms can be extremely sensitive to a slight hyperparameter variation. That requires an exhaustive trial of multiple values and combinations in order to ensure the best possible results with the algorithm are obtained.
- **DRL algorithms overfitting tendencies**: not only are DRL algorithms extremely sensitive to a hyperparameter variations, but there also exists a large risk of overfitting [42] and unstability during the training process, which can translate into a poor convergence and suboptimal results [15].

2.3 Work packages

The work packages and their details are listed below. The modifications that have been mode from the initial planning are written in red.

Task	Task title	Task description	Start date	Due date
Work Package 1 (WP1)				
T1.1	Planification	Initial planification of the work plan, including the Gantt diagram.	15/02/2021	12/03/2021
T1.2	Meetings	The periodic meetings with the director and team to evaluate the state of the project.	15/02/2021	Weekly on Monday
Work Package 2 (WP2)				
T2.1	Study of Reinforcement Learning theory	Study of the theory and mathematical background of Reinforcement Learning.	15/02/2021	22/03/2021
T2.2	Learning software frameworks	Learn about the available software frameworks for RL, and select the one to work with.	22/02/2021	22/03/2021
T2.3	Work environment configuration	The configuration of the work environment (Git repository, virtual machines and cloud configuration, etc.)	22/03/2021	29/03/2021

T2.4	Study cases selection	Selection of the study cases, that is, the RL environments to work with.	29/03/2021	05/04/2021
Work Package 3 (WP3)				
T3.1	Algorithms implementation	Implementation of various Deep RL algorithms training for the selected software framework.	05/04/2021	19/04/2021
T3.2	Training	Training of the implemented algorithms in the chosen RL environments.	19/04/2021	~~10/05/2021~~ 24/05/2021
T3.3	Algorithms selection	Selection of the best algorithms based on their performance.	~~19/04/2021~~ 10/05/2021	~~10/05/2021~~ 24/05/2021
T3.4	Results analysis	Analysis of the obtained results mainly in terms of accuracy and execution training time.	~~26/04/2021~~ 10/05/2021	~~30/04/2021~~ 30/05/2021
T3.5	Results visualizations	Generation of visualizations of the results, from plots to 2D/3D representations of the trained agent model interacting with the chosen environment.	~~26/04/2021~~ 10/05/2021	~~30/04/2021~~ 30/05/2021
Work Package 4 (WP4)				
T4.1	Cloud provider selection	Evaluation and analysis of all cloud services provider options, and selection of one of them.	30/05/2021	03/05/2021
T4.2	Development of training pipeline	Implementation of the model's training pipeline within the selected cloud provider and its services.	03/05/2021	~~17/05/2021~~ 24/05/2021
T4.3	Training execution	Execution of training in the cloud with the previously selected algorithms.	~~17/05/2021~~ 24/05/2021	~~24/05/2021~~ 31/05/2021
T4.5	Results Analysis	Analysis of the obtained results in the cloud, mainly in relation to execution and training times.	31/05/2021	07/06/2021
T4.6	Comparison to CTE	Comparison of the implementation in the cloud to that of the CTE-POWER's cluster.	31/05/2021	07/06/2021

Table 1: Detailed work packages and tasks.

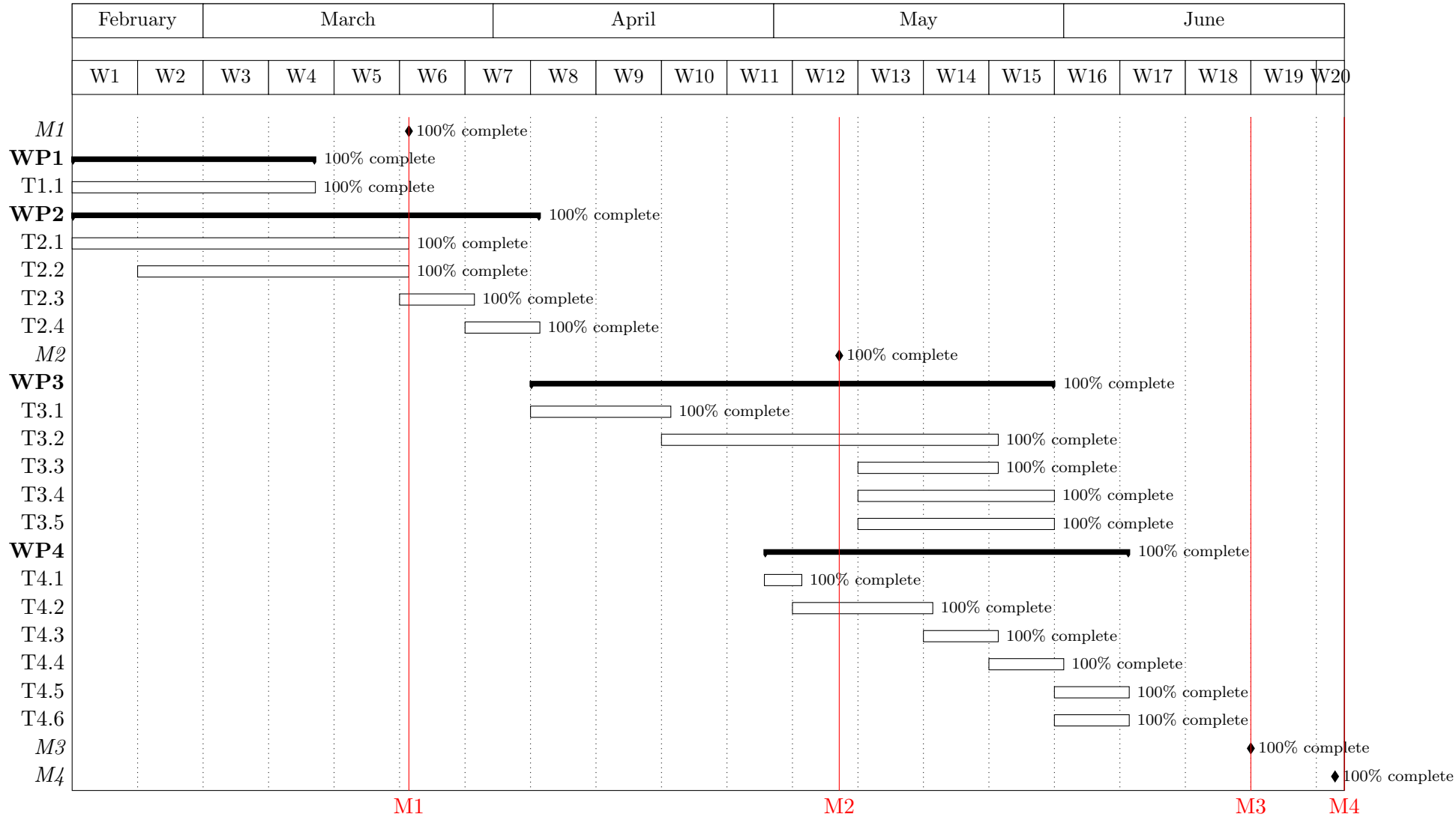
2.4 Milestones

The milestones of the project, corresponding to the three reports that must be delivered as well as the thesis defense presentation, are listed in Table 2.

Milestone	Milestone title	Milestone description	Due date
M1	Initial report	Delivery of the initial report to the thesis director.	22/03/2021
M2	Follow-up report	Delivery of the follow-up report to the thesis director.	07/05/2021
M3	Final report	Delivery of the follow-up report to the thesis director.	20/06/2021
M4	Thesis defense	Presentation of the thesis defense to the jury.	01/07/2021

Table 2: Detailed milestones of the project.

2.5 Gantt diagram



3 Problem definition

Once the objectives of the project are defined, this section aims to deepen into the lower level specifications of the proposed challenge. Firstly, a Reinforcement Learning framework must be chosen, that should have built-in parallelization and distributed computing tools or should at least be compatible with external ones. Then, developing a deployment to the cloud requires the selection of the cloud services provider. And lastly, a case study must be chosen, that is, a RL environment with specific properties that fit all of the project requirements.

3.1 Selection of the software framework

In order to implement a solution, the first requirement is to select the software framework for its development.

The programming language used is **Python**, for two main reasons: firstly, because it is the language of which the author of the project has a higher experience with and knowledge of; secondly, it is the language being used in the research group; and lastly and more importantly, Python is the go-to choice for data science, machine learning and artificial intelligence projects. It is an object-oriented programming language with a high degree of readability. Furthermore, it supports the main libraries for machine learning, such as **PyTorch**, **Scikit-learn**, **Keras**, **TensorFlow** or **RLlib**, and there exist also powerful toolkits for data analysis and visualization, such as **Matplotlib** or **Pandas**, which are useful for the analysis of the obtained results.

There are numerous Python libraries and frameworks for Reinforcement Learning, that might contain support for developing algorithms, built-in implementations of algorithms as well as the ability to train agents for defined environments. The most broadly used ones are **Keras-RL**, Open AI Baselines, **Stable Baselines**, Acme and **Ray**.

The selected framework is **Ray**, being **RLlib** one of its libraries. It was the framework selected by the research group in order to conduct all RL projects. Some of the reasons it was chosen before other options are the following:

- **RLlib** natively supports both **TensorFlow** and **PyTorch**, and most of its internals are platform agnostic. This allows for much more flexibility than other libraries.
- Its library **Ray Tune** is a package designed for hyperparameter optimization that fully supports distributed algorithms, and is fully integrated with **RLlib**.
- The developers of the framework, the AMPLab at the University of California Berkeley, have a solid background of successful projects, such as Apache Spark [41]. This can be an indicator of **Ray** becoming one of the most used frameworks in this field in the near future.

3.2 Selection of the cloud services provider

One of the goals of the project is to reproduce in the cloud the trainings performed in the CTE-POWER cluster. In order to do so, a cloud services provider must be selected.

There exist multiple options, but as shown in Figure 7 most of the applications are built in three of them: Amazon Web Services, Microsoft Azure and Google Cloud Platform. The

three of them were considered, but the final selection was Amazon Web Services, based on the following grounds:

- A robust service for machine learning in **AWS Sagemaker** with Reinforcement Learning specific implementations.
- Its support and documentation for **Ray**.
- The knowledge and experience acquired by the research group using AWS in previous projects.
- A free-tier that provides some of its services for free up to a certain limit.

Furthermore, an AWS credit of 300 USD was received from Amazon for the development of the project.

3.3 Selection of the environment

Before implementing the solution, the problem must be carefully and detailedly defined. Since the goal of this project is to successfully train DRL algorithms both in a supercomputing environment (see 4.3) and a cloud service (see 4.4), and provide a solution easily scalable and reproducible given other problems, the case study or environment must be selected taking into consideration various factors.

One of them is the **degree of difficulty**: it should be a complex enough environment such that only some algorithms and hyperparameter configurations are able to solve it, but at the same time already proven solvable in a reasonable amount of time (due to both the project's time limitations and the intention of performing numerous trainings). Other factors to consider are the use it has in the scientific community, the **quality** of the implementation, the viability of its software **dependencies** and the **compatibility** with the selected frameworks.

Provided that we are using Python and the Open AI Gym toolkit [3] for the RL environments, the selected environment must be compatible with it. There exist multiple open source environments implementations that are compatible with Open AI Gym, even the toolkit itself has some built-in environments, which are the first that will be explored. Apart from this, other toolkits have also been explored and tested. All of them are listed below, with a description of their characteristics.

Open AI Gym. The built-in environments in Open AI Gym go all the way from Atari games, to simulated robotics or continuous control tasks. Unfortunately, the better suited environments for this project were the continuous control tasks running on Mujoco physics simulator, which only allows a free trial of 30 days before having to pay for a license.

Unity Machine Learning Agents Toolkit (ML-Agents). The second explored toolkit was the Unity Machine Learning Agents [17]. It is an open source project with defined games and simulations using Unity's software, and also has a wrapper that converts the environments to gym (the format required by Open AI Gym). The quality of the implementations and the difficulty and variety of the environments is great, and the fact that Unity is used in a huge amount of games and applications makes it a great fit. But on the downside, the execution of the environment simulation must be done with either the Unity Hub app running or by transforming the environment into a binary file (that must be OS specific) that executes it. Unfortunately,

the former is not feasible when using servers and the latter depends on the operating system, which rises a lot of compatibility issues.

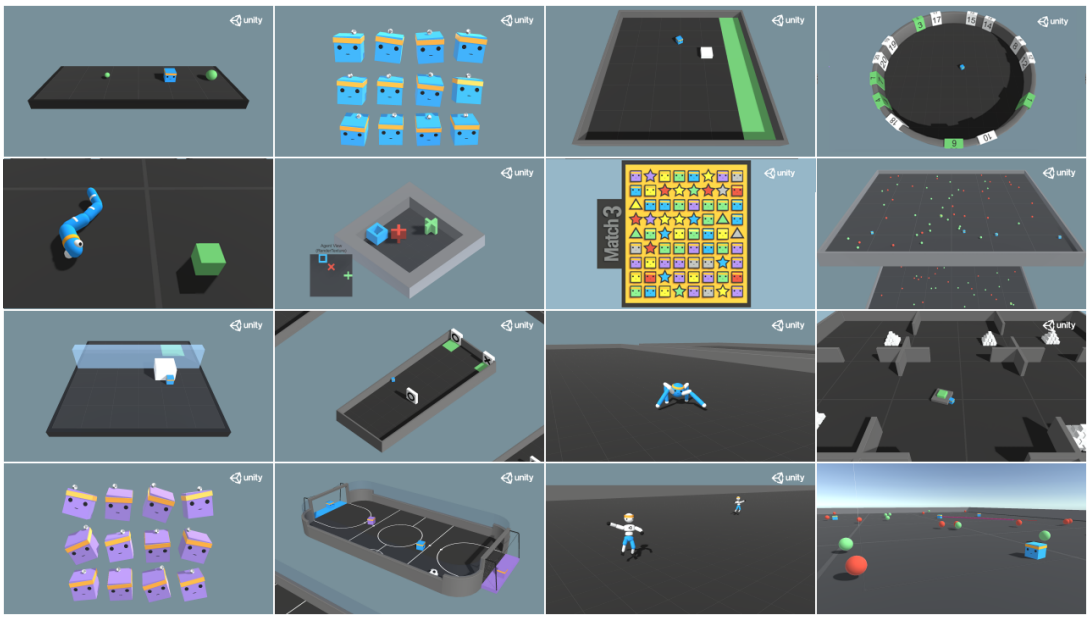


Figure 1: The sample of Unity's environments rendered. Source: Unity Technologies

PyBullet Gymperium. [11] This toolkit tackles the problem mentioned in **Open AI Gym**, providing an open source and free implementations of the continuous control Mujoco environments using the physics simulation Bullet. On the downside, it is not an official PyBullet [9] repository, and the provided renderization functions are very limited resulting in quite poor images.

PyBullet Physics SDK. This toolkit is very similar to **PyBullet Gymperium**, but it actually is the official Python SDK for the physics simulator Bullet. In it, we can find open source and free implementations, among which we find some of the continuous control Mujoco environments. Given that the renderization functions are much better, this toolkit was the final selection.

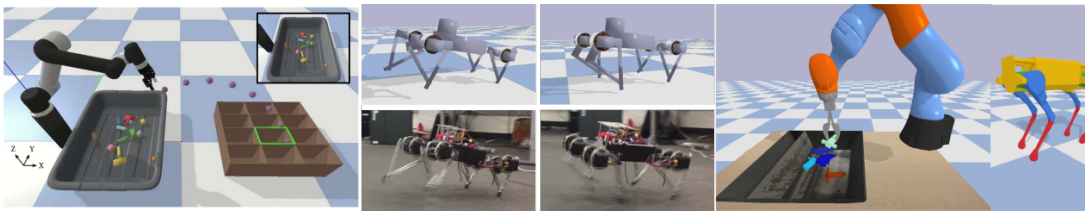


Figure 2: A sample of PyBullet's environments rendered. Source: PyBullet

Taking into consideration all the points mentioned above, the final selection was PyBullet's Physics SDK **HumanoidBulletEnv-v0** environment implentation, which is shown in Figures 3 and 4. It is one of the most complex out of Mujoco's continuous control environments. It is also broadly used in the scientific community together with all the other Mujoco environments, mainly for benchmarking purposes.

The agent consists of a 3D humanoid figure composed by 13 links and 17 joints, and the goal of the agent is to advance in the x dimensions as fast as possible, without falling. The

action and observation spaces are described below.

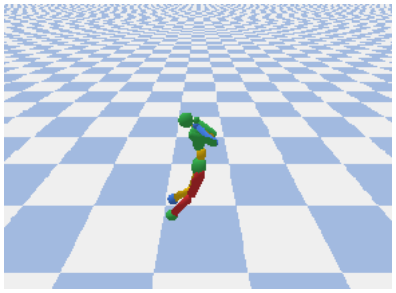


Figure 3: Renderization of the humanoid environment at a random time step.

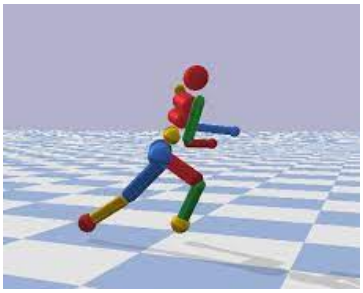


Figure 4: Another renderization of the humanoid environment at a random time step.

Observation space. The observation space is formed by 44 variables: 3 of them correspond to the orientation of the agent for each spatial dimension, 3 other to the velocity of the torso, 2 are the Euler angles for the x and y dimension (also known as roll and pitch), 17 for the relative x positions for each of the joints, another 17 for the velocities on the x dimension for each of the joints, and 2 variables indicating whether or not each feet is touching the floor. See Table 9 in appendix A.1 for more details on the observation space variables.

Action space. The action space is composed of 17 motors, one for each joint in the humanoid figure. See Table 10 in appendix A.2 for more details on the agent actuators.

Reward function. The reward function must be defined according to the environment goal, which is to advance in the x direction. It does also take accountability for states in which the robot falls (it is considered *not alive*), the energy cost of the performed actions, joints getting stuck between each other are and whether or not the feet are colliding with any other body part. The reward function is computed given the current and previous states as shown in Equation 1.

$$\begin{aligned} R(s_t, s_{t-1}) \; = \; & \text{alive\_bonus}(s_t) \; + \; \text{progress}(s_t, s_{t-1}) \; + \; \text{energy\_cost}(s_t) \; + \\ & \text{joints\_at\_limit\_cost}(s_t) \; + \; \text{feet\_collision\_cost}(s_t) \end{aligned} \tag{1}$$

The alive_bonus(s_t) function returns -1 when the state of the environment indicates that there is no physical possibility that the robot does not fall, and 1 otherwise. The progress(s_t, s_{t-1}) function measures the speed of the robot, as $\frac{\text{target\_dist}(s_t) \; - \; \text{target\_dist}(s_{t-1})}{\text{time}}$. The energy_cost$(s_t)$ is a penalty that measures the amount of energy the robot uses, which is measured by adding the powers used at all the joints: this aims to reduce noise and unnecessary movement. Finally, the feet_collision_cost(s_t) returns -1 when at least one of the two feet is colliding with another joint, link or object, and 1 otherwise.

4 Previous work

4.1 Reinforcement Learning

Reinforcement Learning is a domain in artificial intelligence, and more specifically, in machine learning, that seeks to train a learner (the *agent*) to achieve a defined goal (the *reward*) by performing a series of actions within a defined *environment*. At any given time step t the agent selects an *action* a_t given an *observation* of the environment's current state s_t, in order to maximize the cumulative *reward* r_t, which is returned by the environment. Intuitively, the main idea behind Reinforcement Learning is to let the agent acquire knowledge and learn from experience, that is, by allowing it to interact with the environment through its available actions, and obtaining a reward which is directly related that the goal that wants to be achieved. The core concepts of Reinforcement Learning are the following:

Agent. The agent A is the solution to the problem. It is a function that learns to take the optimal actions given the environment's state.

Environment. The environment E is the representation of the problem the agent tries to solve. It is formed by a set of states S, known as *state space*, which can be infinite. It must have a *transition function* defined, which given an action a_t performed by the agent and the current state s_t returns the next state s_{t+1}, and a reward function, which maps the action to the reward r_t;.

State. At any given time step t the environment is represented by a set of variables. A state is each possible combination of values for this set of variables.

Observation. An observation is the information of a state s_t that the agent can observe and has knowledge of. From this point onwards, we will assume that the observation is equal to the state (that is, the agent can observe all the available information) for simplicity purposes.

Action. The agent can select an action a out of the environment's action space $A(s)$ for each specific state. It is represented as a set of variables, which can be discrete or continuous. The action taken by the agent affects the environment, which updates its state s_{t+1} and returns a reward r_t.

Reward. The reward r is the feedback given by the environment to the agent when the latter performs an action. It is computed by the *reward function* $R(s, a)$, which takes as an input the state and the action. The agent seeks to maximize this value. The definition of the reward function can be complex, and must return high values for positive behaviour and lower values for negative behaviour. We must also take into consideration the concept of delayed reward, that is, when we only can know if the taken action is correct in a future time step.

Policy. The policy π is the criterion or method that the agent utilizes in order to select the optimal action (the action that maximizes the reward) given the environment's state. It can be either deterministic $\pi(s) = a$ or stochastic $\pi(a|s) = P(a_t = a|s_t = s)$.

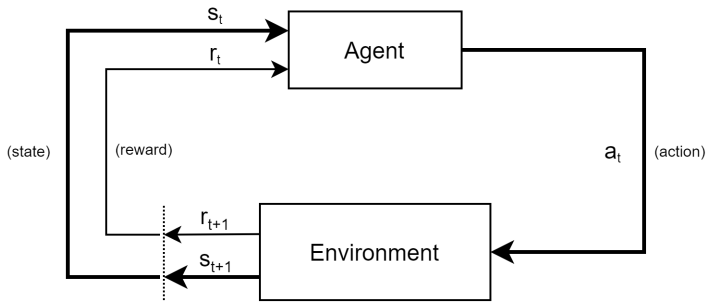


Figure 5: The Reinforcement Learning cycle. The agent performs an action, and the environment returns a reward together with the new state.

Mathematically speaking, a lot of environments can be represented as a Markov Decision Process (MDP): it is formed by a finite set of states S and the transition probabilities $P(s_{t+1}|s_t, a) = P(s'|s, a)$ that depend upon the actions $A(s)$ the agent can take. Thereafter, the agent's task lies on estimating the expected cumulative reward G of its actions in order to select the one that maximizes it.

4.1.1 Criterion of optimality.

There are three main models of optimal behaviour, that is, the function we are looking to optimize in order to maximize the notion of cumulative reward, also known as *return*, which basically establishes how far into the future the policy must take into account when selecting the present action. A common approach is making use of a discount factor γ, to discount the weight of future rewards with time, as shown in Equation 2.

$$G_{t'} = \mathbb{E}\left[\sum_{t=t'}^{t=\infty} \gamma^t r_t\right] \tag{2}$$

Another usual approach is the *finite-horizon*, which only considers the future h steps, all with the same weight, as shown in Equation 3.

$$G_{t'} = \mathbb{E}\left[\sum_{t=t'}^{t=t'+h} r_t\right] \tag{3}$$

An alternative is the *average-reward* model [https://arxiv.org/abs/2010.08920], which considers all future rewards (infinite horizon) by taking the average, as shown in Equation 4.

$$G_{t'} = \lim_{h\to\infty} \mathbb{E}\left[\frac{1}{h}\sum_{t=t'}^{t=t'+h} r_t\right] \tag{4}$$

State-value function. Given a policy we can define the *state-value* function $V_\pi(s)$ as the expected reward given a state, as shown in Equation 5.

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{5}$$

Action-value function. And the *action-value* function $Q_\pi(s, a)$, also known as *Q-function*, as the expected reward of performing an action a given a state s and a policy π, as shown in Equation 6. As explained in the algorithms section, the Q-function can be customized to represent a modification of the obtained reward depending on the algorithm, for instance with the intention of encouraging exploration of the agent.

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{6}$$

Bellman equations. Assuming an infinite-horizon with discount factor γ model, from Equation 2 we obtain:

$$G_t = \sum_{i=0}^{T} \gamma^i r_{t+i} = r_t + \gamma G_{t+1} \tag{7}$$

And thus, applying (7) to (5) and (6), we obtain:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \sum_a \pi(a|s) \cdot \sum_{s' \in S} P(s'|s, a) \cdot [R(s, a) + \gamma V_\pi(s')] \tag{8}$$

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \sum_{s' \in S} P(s'|s, a) \cdot [R(s, a) + \gamma V_\pi(s')] \tag{9}$$

We can rewrite then (8) as:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \sum_a \pi(a|s) \cdot Q_\pi(s, a) \tag{10}$$

Policy optimization.

Conceptually, the policy that achieves the optimal value and action-value functions V_* and Q_* is the following:

$$V_*(s) = \max_\pi V_\pi(s) \tag{11}$$

$$Q_*(s, a) = \max_\pi Q_\pi(s, a) \tag{12}$$

Advantage function.

A common approach for most algorithms in DRL consists on using the advantage concept in the objective function. The advantage function, defined in Equation 13, compares the Q-value of an action to the average Q-value of all the actions given the same state, extracting the difference. Temporal difference methods [36], which are used by most of DRL algorithms, seek to estimate the advantage function and use this for the objective function to train the agent.

$$A(s, a) = Q(s, a) - V(s) \tag{13}$$

4.1.2 Exploration vs exploitation

The training algorithms differ from supervised and unsupervised learning methodologies in that they do not use any pre-labeled data as the former, neither unlabeled data as the latter, as there is no previous data. They actually learn by interacting with the environment and analyzing the obtained rewards given their corresponding state and performed action, which is used as the training data. That is in fact why one of the biggest dilemmas in Reinforcement Learning is the trade-off between exploitation (using the already acquired knowledge) and exploration (of non-inspected options) [18].

On the one hand side, the agent must perform actions that based upon prior experience it knows will return the highest rewards, in order to find the optimal path. But on the other end, it must explore alternative and unknown options with the aim of collecting this prior experience. Since both strategies are exclusive, that is, at a given time step the agent must chose one or the other, it makes it a very difficult task to find the ideal balance.

In order to tackle this, there exist various methods to take the decision for the agent. The most broadly used exploration strategy is the ϵ-Greedy Exploration, which consists on selecting, at each time step, an exploratory (random) action with probability ϵ, where $\epsilon < 1$, and selecting the learned optimal action with probability $1 - \epsilon$, as shown in Equation 14.

$$\begin{cases} \text{random action from A(s)} & \text{if } \xi < \epsilon \\ \max_a Q_*(s,a) & \text{otherwise,} \end{cases} \tag{14}$$

There exist some other alternatives to this method that play around with the ideas of reducing the value of ϵ with time [5] [37].

The simplest approach to this is the decaying ϵ-Greedy, which uses the decay rate γ to reduce ϵ over time, as shown in Equation 15.

$$\epsilon_t = \gamma^t \epsilon_0, \text{ where } 0 < \gamma < 1 \tag{15}$$

Another common method consist on using softmax [29], which sets the policy's probability of selection for each action by ranking the value-action function values using a Boltzmann distribution, as shown in Equation 16.

$$\pi(s,a) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a'} e^{\frac{Q(s,a')}{\tau}}} \tag{16}$$

4.1.3 Off-policy vs on-policy

One of the most common classifications of Reinforcement Learning algorithms consists of dividing them into off-policy and on-policy methods.

Off-policy algorithms estimate the *Q-value* function independently from their policy. That is, they perform actions (they are exploring, as the actions are not selected by the policy) and store the retrieved data (state-action pairs and the resulting rewards). Then, separately, a policy is trained based upon the observed Q-values.

On the other hand, on-policy algorithms update the Q-values directly using the current policy's action. The same policy that is being trained to select the optimal actions is also responsible for the retrieval of the state-action pairs and corresponding rewards that are used for its own training.

This actually translates into off-policy algorithms being more sensible to hyperparameters, biased, and prone to divergence, but at the same time they have lower variance and thus are more sample-efficient.

4.1.4 Model-based vs model-free

Another usual classification of Reinforcement Learning methods consists on separating them contingent on if they are model-based or model-free.

Model-based algorithms are those that, in order to train their policy, they *model* the environment, that is, they try to approximate the transition $P(s'|s,a)$ and reward $R(s,a)$ functions, and then use these approximations to estimate the optimal policy.

Contrarily, a model-free algorithm does not seek to approximate a model of the environment (the transition and reward functions), but instead estimates the policy directly from experience. It trains a value function that given a state returns the optimal action to perform.

4.1.5 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a subarea of Reinforcement Learning that combines deep neural networks with the reinforcement learning architecture.

Given an environment in which the amount of states were too large (or even infinite), the estimation of the V-function, Q-value table or the policy is not feasible. That is when deep neural networks come into play: they can behave as function approximators, capable of receiving a flexible input (which represents the observation and/or action spaces) and output an estimation of the value. In DRL algorithms, we find that the neural networks can behave as estimators of:

- **State-value function ($V_\pi(s)$)**: also known as V-function (see Equation 5), which receives as input an state s and returns the expected reward given a policy π.
- **State-action value function ($Q_\pi(s,a)$)**: also known as Q-function, which receives as input an action a and state s and returns the expected reward given a policy.
- **Policy ($\pi(s)$)**: the value function, which receives as input an state and returns the optimal action in order to maximize the cumulative reward.

This approach allows RL algorithms to be applied to much more complex environments with successful results.

Most of the research and RL applications as of today require the use of deep neural networks, as it is the case with this project. For instance, environments with continuous actions are more suitable to a DRL approach and would require a discretization of the actions' domains if a dynamic programming approach were to be taken.

4.1.6 Algorithms

Several algorithms are used in this project, each of them being different and might be more suitable for certain environments or more sensitive to some of the hyperparameters. In this section, a detailed description is made for each one of them.

4.1.6.1 Deep Q-Network (DQN)

The Deep Q-Network algorithm was introduced by [28], and consists of approximating the Q-function, that returns a value for each of the environment state-action pairs (which is the expected reward by selecting that action in that specific state) by means of a deep neural network. Thus, the optimal deterministic action to perform can be determined following the policy shown in Equation 17, which can be directly obtained from the Q-function estimation.

$$\pi(s) = \arg\max_a \ Q_\theta(s, a) \tag{17}$$

In order to do so, the algorithm uses two networks: the main one (called Q-network), and a target neural network, that is a copy of the former and acts as the agent to perform episode steps and obtain rewards. This data is then employed to update the weights of the main neural network after each episode, with samples being drawn uniformly from the replay buffer, and the gradient is computed using mini-batches in order to update the Q-network. Every k steps (which is an hyperparameter), the target network is updated to the Q-network. This is done to minimize the volatility of the neural network used for the exploration and compilation of training data by only updating it every k steps, maintaining stability for the periods in which it retrieves the data.

The network is trained to minimize the loss function shown in Equation 18, which is the squared error between the network's estimated Q-value and the observed reward.

$$L = (r_t + \gamma \arg\max_a \ Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s, a))^2 \tag{18}$$

On the downside, the DQN algorithm is only applicable to discrete action spaces, as it requires an efficient evaluation of the Q-function approximation which is not achievable in continuous action spaces unless a discretization is performed.

4.1.6.2 Deep Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient algorithm [23] uses an actor-critic architecture. The critic Q_ϕ is responsible of learning the Q-function, while the actor π_θ defines the policy that is used to select the optimal actions. The loss function of the critic is the same one used in DQN, as shown in Equation 19.

$$L_C = (r_t + \gamma Q_\phi(s_{t+1}, \pi\theta(s_{t+1})) - Q_\phi(s, a))^2 \tag{19}$$

To train the actor, the loss function uses the critic network Q_ϕ which estimates the Q-value of the state and the action selected by the policy, as shown in Equation 20.

$$L_A = -Q_\phi(s, \pi_\theta(s)) \tag{20}$$

4.1.6.3 Twin Delayed DDPG (TD3)

Twin Delayed DDPG (TD3) [13] is an off-policy algorithm. It can be described as the improved version of DDPG, which can be unstable and prone to brittling with respect to hyperparameters, due to the critic network overestimating Q-values. TD3 tackles this issue by reducing DDPG's overestimation bias in Q-values, by doing the following:

- The addition of a second critic network (contrary to DDPG's only having one). TD3 takes the minimum out of the two estimates of the Q-value, as shown in Equation 21, where $d = 1$ whenever s_{t+1} is a terminal state.

$$Q(s_t, a_t) = r_t + \gamma(1 - d) \min_{i=1,2} Q_{\phi i}\left(s_{t+1}, \pi_\theta(s_{t+1})\right) \tag{21}$$

- The delay of the policy updates. TD3 updates the actor network less frequently than the critics (by default, one actor network update per every two critic updates). The policy is learned by maximizing only $Q_{\phi 1}$, as shown in Equation 22.

$$\max_\theta \ \mathbb{E}\left[Q_{\phi i}(s, \pi_\theta(s))\right], \quad \forall s \in S \tag{22}$$

- Adding noise to the target action. This prevents the actor from taking advantage from Q-values estimation errors. As shown in Equation 23, clipped noise is added to the policy's selected action, with the hyperparameters c and σ, and takes into account the domain $[a_{min}, a_{max}]$ of the action.

$$A(s) = clip\left(\pi_\theta(s) + clip(\epsilon, -c, c),\ a_{min}, a_{max}\right), \quad \text{where } \epsilon \sim N(0, \sigma) \tag{23}$$

These modifications translate into a significantly better performance than the DDPG baseline.

4.1.6.4 Soft Actor Critic (SAC)

Soft Actor Critic (SAC) [16] [15] is an off-policy algorithm that stands out for not only looking for a maximization of the reward, but also seeking to maximize the entropy, which is basically a measure of the randomness within the policy. In other words, it seeks to obtain the highest possible reward while acting as randomly as possible.

At each time step, the reward is modified by adding a bonus proportional to the entropy of the policy. Thus, the new cumulative reward is defined as shown in Equation 24. The hyperparameter α controls the explore-exploit tradeoff, a higher value translating into more exploration while a lower value indicates more exploitation.

$$G_t = r_t + \gamma\left(r_{t+1} + \alpha H(\pi(\cdot|s_{t+1}))\right) \tag{24}$$

And by applying the definition of entropy to 24:

$$G_t = r_t + \gamma\left(r_{t+1} - \alpha \log(\pi(a_{t+1}|s_{t+1}))\right) \tag{25}$$

The algorithm is composed by two critic networks $Q_{\phi 1}$ and $Q_{\phi 2}$, and a policy π_θ.

Learning Q-values. The Q-value function is defined then as $Q_{\phi i}(s_t, a_t) = G_t$. Then the

loss function is 26.

$$L(\phi_i) = \left(Q_{\phi i}(s_t, a_t) - (r_t + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi j}(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\theta(a_{t+1}|s_{t+1}) \right)) \right) \tag{26}$$

Learning the policy. The policy π_θ seeks to maximize the expected G_t, thus it is optimized by maximizing Equation 27.

$$\begin{aligned} & \max_\theta \ \mathbb{E}\left[Q_{\pi\theta}(s,a) + \alpha H(\pi_\theta(\cdot|s)) \right] \\ = & \max_\theta \ \mathbb{E}\left[Q_{\pi_\theta}(s,a) - \alpha \log(\pi_\theta(a|s)) \right] \end{aligned} \tag{27}$$

This method contains a state-of-the-art built-in balance between exploration and exploitation, and avoids an overfitting to the first found solutions, by allowing the policy to find new paths different from the currently successful ones. I was first introduced by [16].

4.1.6.5 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [34] is a policy gradient method for Reinforcement Learning. It is an on-policy algorithm. The main characteristic is its low complexity in comparison to other deep reinforcement learning algorithms.

The idea is to train a network to choose the action that maximizes the expected advantage ponderated by the ratio between the current policy's probability of that action and that of the old one, but at the same time limiting the magnitude of the policy update at each step by clipping this ratio between $1-\epsilon$ and $1+\epsilon$, being ϵ an hyperparameter, as shown in Equation 28.

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta k}(a|s)} A^{\theta_{\pi k}}(s,a), \ \text{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta k}(a|s)}, 1-\epsilon, 1+\epsilon) A^{\theta_{\pi k}}(s,a) \right) \tag{28}$$

4.2 Software frameworks

Multiple software frameworks have been used for the development of this project. All of them are thoroughly explained in this section.

4.2.1 Ray

Ray is an open source Python library which provides an API for simplified distributed computing. It easily allows to scale and parallelize applications. Furthermore, there exist multiple libraries that work on top of Ray Core and supply various machine learning functionalities, such as reinforcement learning algorithms (see 4.2.1.1), hyperparameter tuning (see 4.2.1.2), distributed training (RaySGD) and scalable and programmable serving (Ray Serve).

There also exist many integrations with other libraries and platforms, such as Dask [33],

Scikit-learn [31] and Hugging Face, among others.

4.2.1.1 RLlib

RLlib [21] is an open-source library for Reinforcement Learning that offers an API with various tools. It actually supports integrations with **Tensorflow** and **PyTorch**, as most of its functions support both frameworks, even though most of its internals are framework agnostic. This is very relevant as, differently from other frameworks, it does not limit its practice to users familiar with one of them.

4.2.1.2 Tune

Tune [22] is a Python library mainly focused on providing hyperparameter tuning functionalities and experiment execution for machine learning. It allows the user to execute intelligent hyperparameter optimization with very few unobtrusive lines of code, supporting various machine learning frameworks (such as **PyTorch**, **Tensorflow**, XGBoost [6] and MXNet [7]).

Furthermore, it has implemented the state-of-the-art algorithms for hyperparameter tuning, such as Hyperopt [2], BOHB [12], Hyperband [20], Dragonfly [19], Nevergrad [32], Optuna [1], ZOOPt [25] or HEBO [10], among others.

4.2.2 Tensorflow

Tensorflow [26] is an open source Python library for machine learning, initially developed by Google and released to the public on 2015. Its main uses are for training and inference of machine learning models, but it also provides multiple add-on tools for various tasks, such as TensorBoard, Tensorflow Playground or What-If Tool.

4.2.3 PyTorch

PyTorch [30] is also a free and open source machine learning library that is based on the Torch [8] library, mainly developed by Facebook's AI Research lab (FAIR). It is a library optimized for tensor computing in CPUs and GPUs and is based on an automatic differentiation system.

An extensive number of tools are built on it, such as TorchElastic, TorchServe or PyTorch Lightning, which is a high-level interface of the framework.

4.2.4 Docker

Docker is an open source platform that uses virtualization for an easy way to build, deploy and run applications by means of containers. These *contain* all the required packages, dependencies, system tools and runtime, defined in what is known as an image, making it possible for them to run in any environment. It is similar to the concept of a virtual machine, but it differs in that it does not need to create an entire operating system as it can use the same Linux system's kernel.

Figure 6 shows a high-level schema of how Docker applications work, all the way from building to deployment and execution, while also unveiling the simple syntax of its CLI.

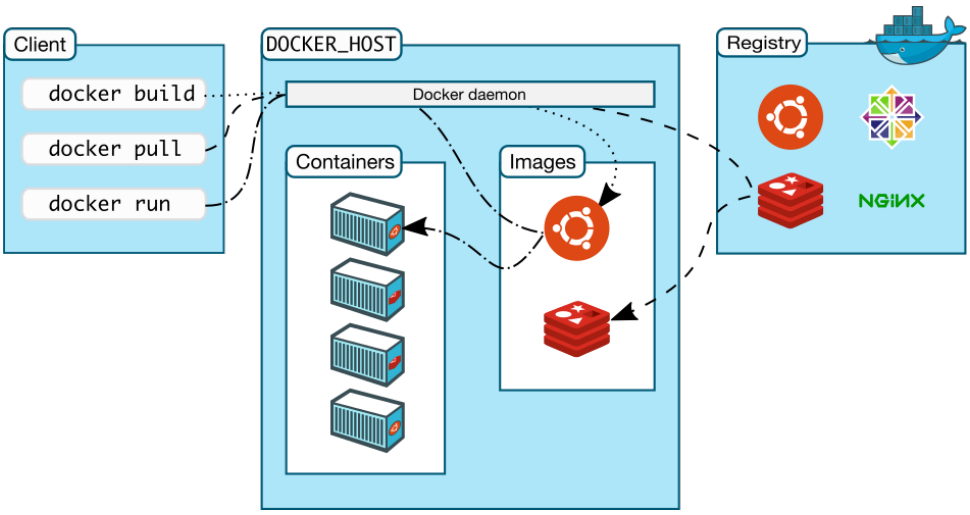


Figure 6: High level abstraction of a Docker application life cycle. Source: Docker documentation

.

4.3 CTE-POWER Cluster

CTE-POWER is a cluster based on IBM Power9 processors, with a **Red Hat Enterprise Linux Operating System** and an Infiniband interconnection network. It is formed by a total of 54 nodes, 2 of them being login nodes and 52 compute nodes. Each of them has a total of 20 cores with 4 threads/core, 512 GB of memory, 3.8 TB of local storage and 4 GPUs NVIDIA volta. For more technical details, see `https://www.bsc.es/marenostrum/marenostrum/technical-information`.

4.4 Amazon Web Services

Amazon Web Services is a branch of Amazon that provides cloud computing services to all kinds of users, all the way from individuals to enterprises and governments. These services seek to deliver abstraction of technical infrastructure, paying only for what is used, and ease of implementation for what would otherwise be very complex applications.

AWS not only provides these computing services but it also automatically manages billing, security and logging, among others. All services are billed based on usage, but each service measures usage in varying ways: some are pay-as-you-go while others charge depending on the pre-configured availability. Out if the cloud computing services providers, AWS has a dominant market share of 33% of all cloud (IaaS, PaaS) while the next two competitors Microsoft Azure and Google Cloud have 20%, 9% respectively according to Synergy Research Group [14].

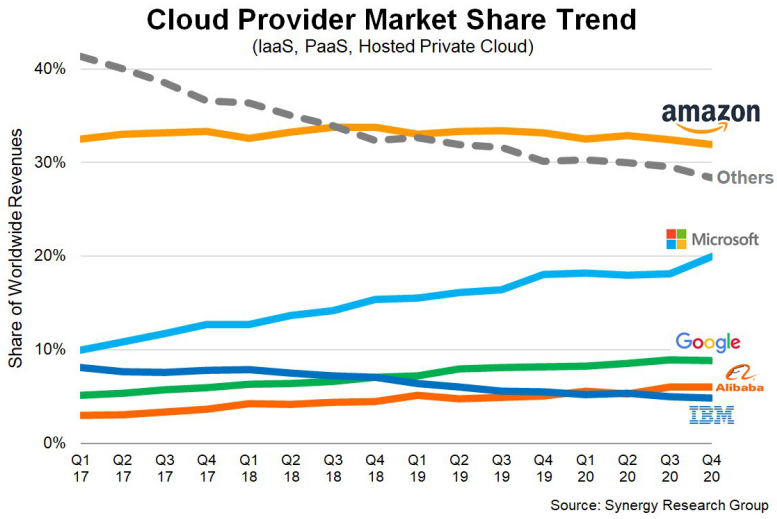


Figure 7: Cloud market share, as of the end of 2020.

4.4.1 Amazon SageMaker

Amazon SageMaker is a cloud machine learning service that was initiated in 2017. It allows users to experiment, develop and train machine learning models in the cloud (using virtual machines, see 4.4.3), and also provides the developers to store the resulting trained models' artifacts in the cloud, to be able to make a deployment into inference afterwards.

SageMaker provides a Python SDK that operates at a high level of abstraction for users to define their processing or training jobs, as well as their models, and execute them while analyzing the metrics and results. On its highest level of abstraction, SageMaker actually allows developers to select pre-trained models that can be deployed as-is. In a slighlty lower level, SageMaker has an extensive number of prepared algorithms can be trained on custom data. Developers can also create their own machine learning models from scratch, and make use of Docker images to customize the training environment by including all required dependencies and configurations of the instance that handles the training job (more on this in the AWS Implementation section).

Regardless of the level of abstraction, SageMaker enables multiple integrations with other AWS services, such as storages (see 4.4.2) and monitorization (see 4.4.5), to facilitate the development of complete machine learning training and deployment pipelines.

4.4.2 Amazon Simple Storage Service (S3)

Amazon Simple Storage Service is a web service that offers *object-based storage* [27], which is different than other traditional storage techniques such as file systems, which manages the data using a file hierarchy, or block storage, which manages it dividing it into blocks. By treating the data as objects, directories (file hierarchy) as themselves do not actually exist, but instead each object contains its own data, optional metadata and a unique identifier, which in the case of S3 is known as *key* and is the path or name of the file (from which a directory structure can be easily deduced). In S3, the stored objects can be organized into different buckets, each one containing up to 5 terabytes of objects that must be uniquely identifiable by their key.

This flexible data storage system is very well suited for a wide range of applications, such

as data lakes, data archives or heterogeneous sets of files corresponding to the same project. It also allows the service to be highly-available, scalable and have a very low latency. In the scope of this project, S3 fits very well the requirements for training jobs' storage, as it provides an easy interface to store a diversified array of files, such as the model checkpoints, metrics and generated videos of the interaction between the agent and the environment, among others. More on this in section AWS Implementation.

4.4.3 Amazon Elastic Computing (EC2)

AWS most widely used and known service is Elastic Compute Cloud (EC2), which offers a virtual cluster of all types of computers constantly available (the user can select number of CPUs, GPUs, RAM memory, hard-disk storage, operating system, network properties, etc). The user can specify, configure, create, launch, stop and terminate server-instances as needed, and only pays for the time the instances are up and running.

It is also worth noting that some of the other services are in fact built on EC2, including AWS SageMaker. In this project, the SageMaker training jobs definitions allow the selection of the EC2 instance/s that will host and run the training script, as explained in AWS Implementation. It is always a good practice to find the optimal balance between enough computational resources and the price of the instance.

4.4.4 Elastic Container Registry (ECR)

Elastic Container Registry is a Docker image registry fully managed by AWS. ECR supports public and private container image repositories, and is integrated with a wide range of AWS services that can use Docker images for their execution. It provides the functionality shown in the Regsitry section of Figure 6.

It also incorporates a very handy CLI, compatible with any OS, that simplifies the management of images and versions by providing methods to push, pull and configure the Docker images.

In the context of the project, ECR is used to manage the Docker images employed for the training jobs, which contain the specific dependencies and requirements for the correct execution of the job.

4.4.5 Amazon CloudWatch

Amazon CloudWatch is a service that provides real-time monitorization for all the applications running in AWS, all the way from cpu and memory usage of active EC2 instances to collecting and tracking metrics and logs of SageMaker training jobs. The formats and graphs used to show the data in CloudWatch's web interface can be easily edited, and custom metrics can be defined by making use of all the data registered on each particular service.

On top of that, the service also allows the configuration and trigger of alarms given the occurrence of a defined event. For instance, this is very useful for applications that require auto-scaling, which must automatize the deployment of new resources if the currently available ones are in path to saturation.

In this project, CloudWatch is used to monitor the logs of SageMaker training jobs, in order to be able to debug warnings and errors, as well as to set up real-time tracking of the EC2 instance and algorithm metrics. This is explained in more detail in AWS Implementation.

5 Implementation

There are two independent and clearly separated implementations in this work, one corresponding to CTE-POWER's training scripts, and the other corresponding to the AWS cloud scripts and configuration. All the code is in a git repository, which is hosted in GitHub at https://github.com/miquelescobar/ddrl. In it, one can find all the code, results, figures, plots, videos and documentation generated during the development of the project. The README file provides a comprehensive guide for reproducibility, explaining each of the directories in detail as well as the dependencies installation process.

One of the aims of this work is to deeply understand the differences between the implementation in a private computing cluster (in this case, the CTE-POWER cluster) and the implementation in an external cloud services provider (in this case, Amazon Web Services). Listed below are the main properties, advantages and disadvantages for each of the two options.

CTE-POWER cluster

+ Very little learning overhead for using it, as it only requires extra knowledge of **Slurm Workload Manager** [40].
+ It is "free" to use for the members of research groups in BSC.
– Cluster maintenance breaks prevent the users from accessing and using the machines, and can last from a few hours to a few days.
– The operating system, **Red Hat Enterprise Linux Server 7.5** is the same for all the machines, and might cause problems with some dependencies that work differently in other OS. Furthermore, the average user does not have access to the configuration of the machine and all the dependencies must be installed and maintained by a specialized professional.
– There is no open access to the Internet from the machines. This precludes the setup of any web service or communication that could be needed in an application, and thus requires another machine to host the setup.

Amazon Web Services

– It requires a considerable amount of previous work to familiarize and learn how AWS works, and more specifically, how to use AWS SageMaker.
– All the offered services are billed, thus an efficient use is required to not go over budget.
+ There is an extensive variety of server instances to use, with all possible resource combinations.
+ There is also a great variety of operating systems and pre-built Docker images that include some of the most common dependencies installed. Anyhow, these can always be customized by the user by means of the Docker image making the service compatible with basically any system.
+ The SageMaker service is fully integrated with other AWS services, making it easy to monitor, store data or deploy the resulting model, among others.

In summary, the two options have very different properties but at the same time each of them fulfils a clear and differentiated purpose. From the point of view of research, the CTE-POWER cluster is very adequate to carry out trials executing as many jobs as necessary, experiment with as many algorithms, environments and configurations as desired, since the costs are so low. On the other hand, AWS can be used for those cases in which not much experimentation is required (the task is clearly defined and the algorithm is known previously or is already trained in CTE-POWER, for instance) but the model or results must be integrated into an application, and thus requires some communication and coordination with other modules.

5.1 Training in RLlib

Both approaches have in common the framework used with the built-in algorithms, which is Ray. The algorithms can be trained following different architectures. The algorithms selected for the chosen environment (TD3, SAC, PPO) all follow the same architecture, shown in Figure 8. In terms of resource configuration, Ray easily allows to specify the amount and properties of the rollout workers, that is:

- The amount of workers for the Trainer module.
- The amount of sampling rollout workers (responsible for sampling new data).
- The number of CPUs and GPUs for each of the worker types.

,

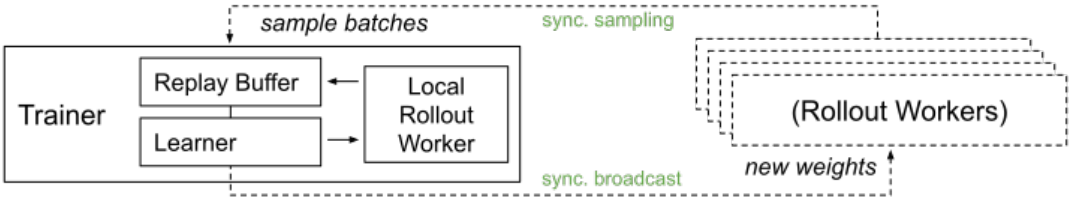


Figure 8: Architecture of a Ray training process, for TD3, SAC and PPO algorithms.

Whenever the required resources exceed those of one node, we can recur to distributed computing. The training algorithm maintains the architecture shown in Figure 8, but there is one more step required. A master process must initialize Ray in one **head node** (must know the addresses of the worker nodes to communicate with all of them) and one or more **worker nodes** (all must know the address of the head node to communicate with it) that initialize the required Ray workers as usually. Then, the head node is responsible for executing the training script, storing the checkpoints and metrics. The Ray workers in the worker nodes are responsible for executing their given tasks (learning or sampling). Figure 9 shows a high-level representation of this architecture.

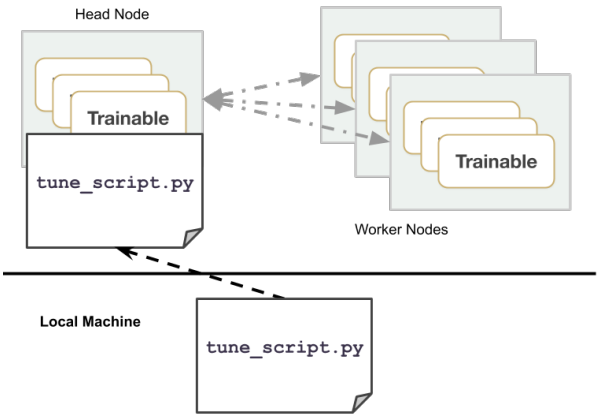


Figure 9: Architecture of distributed computing in Ray.

Thus, the mention of the **training scripts** in the following sections refer to a script that executes this architecture, and the configuration of the training includes the resource allocation.

5.2 Implementation in CTE-POWER Cluster

For training the algorithms using the CTE-POWER, Python scripts are implemented and posteriorly executed using the Slurm Workload Manager [40], with which the allocated resources and properties of the job can be configured. It is very important that the allocated resources are enough for those demanded in the configuration of the training script, otherwise the script will detect that there are not enough resources and the execution will result in an error.

Furthermore, all the required dependencies are satisfied by loading the corresponding environment modules in the cluster, which include the required versions of the necessary software frameworks and/or packages, for both Reinforcement Learning algorithms and training (Ray, Tensorflow, Pytorch) and Reinforcement Learning environments implementations (OpenAI Gym, PyBullet).

At the same time, the training scripts allow parametrization for choosing the environment, algorithm and hyperparameters (there can be more than one parallel combination, see 4.2.1.2). The results are stored at the defined directory, as well as the configured checkpoints during training.

The schema in Figure 10 shows the implemented workflow for the training process in the CTE-POWER cluster.

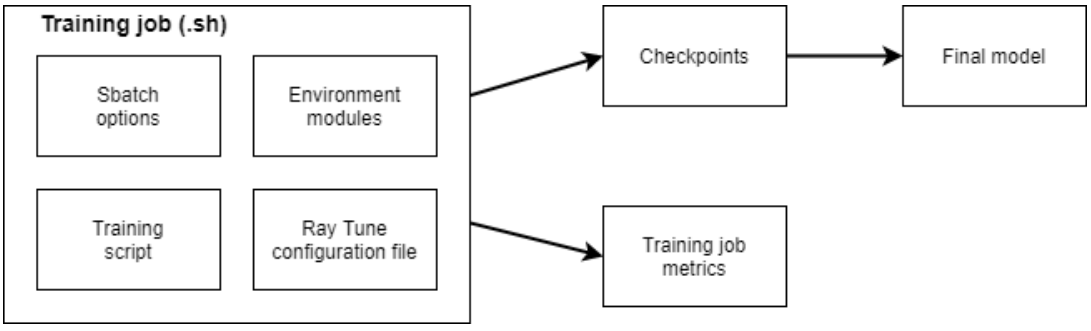


Figure 10: The architecture of the solution implementation in CTE-POWER cluster.

A shown in the schema, a training job is defined and executed from a bash script, which requires four independent modules. The **`sbatch` options** consist of a series of parameters that configure the job (from the resources allocation to the output of the logs, among others). The **environment modules** refer to the layers that the job requires to operate, which are software packages (the Python version, the required libraries, etc.) defined by the system administrators. The **training script** is the command that will be executed, and must handle all the training process. The **Ray Tune configuration file** are the parameters that will be passed to the training, such as the environment, the algorithm, stopping conditions and hyperparameters.

When the four modules above are defined or referred to in the bash file, the training job can be executed using the `sbatch` command. During the execution, the training job stores the corresponding checkpoints and training metrics.

Once the job finishes, we are left with a final model as well as with a complete metrics files with an iteration granularity level. In order to generate the videos of the interactions with the trained model, the corresponding renderization scripts can be executed in a computer with the required dependencies installed, since CTE-POWER computation nodes do not have the capabilities to generate videos from the renderization of environment states.

5.3 Implementation in Amazon Web Services

Some of the trainings made in CTE-POWER, mainly the most successful ones, are reproduced in the cloud for comparison purposes, using the AWS SageMaker service.

In order to train an algorithm for a given environment, the instance type, number of instances, timeout, dependencies and training script can be specified to create an AWS SageMaker training job, which can then be executed. During the execution, configurable checkpoints and renderized videos are stored in the provided S3 bucket, and the configured metrics can be tracked live using either AWS CloudWatch or by setting up a TensorBoard. Once the execution is finsihed, due to any of the predefined stopping conditions, the final model artifact is uploaded to that same S3 bucket.

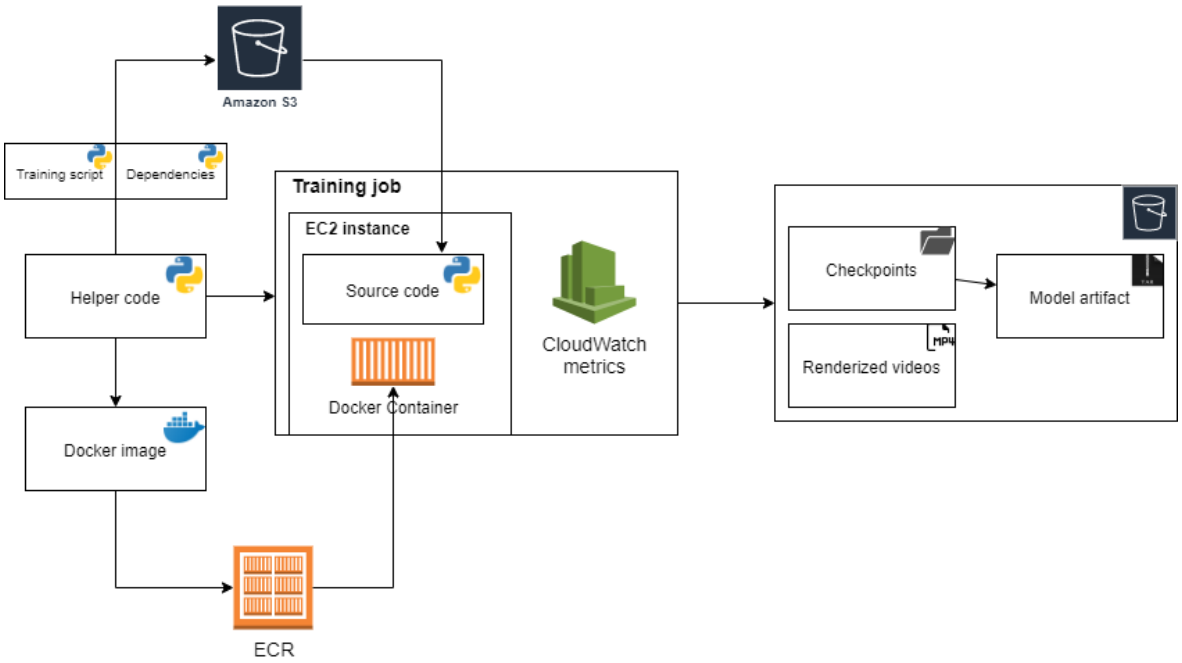


Figure 11: The architecture of the solution implementation in AWS SageMaker.

As can be seen in Figure 11, the training job requires the helper code, which configures the job, uploads the necessary files, builds and pushes the Docker container and starts the execution. From there onwards, the control of the pipeline is left completely to the SageMaker service. Each of the modules that appear in the architecture schema are explained below.

The **helper code** consists of a script (in a Jupyter Notebook) with a series of commands that configure the parameters for the **training job** (see RLEstimator class documentation in AWS Sagemaker Python SDK). It is also responsible for buildng and pushing the corresponding **Docker image** to **ECR**. Finally, the script submits the **training job** in order for it to start.

The **helper code** also must upload the **training script**, which is the entry point of the **training job**, as well as the required **dependencies** to **S3** (only Python code, any other dependency must be installed in the **Docker image**).

The training job **instance** retrieves the **Docker image** from **ECR** and runs the **Docker container** process. The process downloads the **training script** and **dependencies** from **S3** and uses them as the **source code** to execute.

During the execution, the output generated by the job is the checkpoints folder, as well as the **renderized videos** of the agent at each checkpoint and the **CloudWatch metrics**, which were previously configured in the **helper code**. At the end of the job, a compressed file is generated with the **model artifact**, that is, all the properties and the weights of the final model.

Finally, the model can be deployed into inference by referencing the resulting model artifact. SageMaker provides an easy interface to automatically create the endpoint, formatting the responses and setting up the serverless service responsible for listening and answering the received petitions at the endpoint. This inference environment can be configured to suit the availability and desired requirements of the inference service.

6 Evaluation

In this section the results of all the executed trainings for the selected environment are analyzed. The focus is made in analyzing the execution time (we observe how parallelism and distributed computing reduces it significantly) and the obtained rewards per each algorithm and hyperparameter combination.

Refer to the GitHub repository at https://github.com/miquelescobar/ddrl/results/, where the plotting scripts, figures, renderized videos and summarized training metrics can be found.

6.1 Training results

The trained algorithms are TD3, SAC and PP0. This section shows the trainings with best results for each of the algorithms.

6.1.1 TD3

A detailed explanation of the algorithm can be found in 4.1.6.3. Even though tries to solve the drastic overestimation problem of the DDPG algorithm, it is still considered an unstable algorithm that is prone to overestimate Q-values, since it does nothing explicit to control the policy updates.

The best performing training is presented in Figure 12, and corresponds to the hyperparameter configuration shown in Table 3.

Hyperparameter	**Symbol**	**Value**
Tau	τ	0.002
Horizon	h	1000
Actor learning rate		0.003
Critic learning rate		0.003
Discount factor	γ	0.99
Batch size		256
Actor hidden layers		[512, 512]
Critic hidden layers		[512, 512]
Activation function		ReLU
Target noise		0.2
Target noise clip	c	0.5

Table 3: Hyperparameters with best results for TD3.

We can observe in Figure 12 some sporadic spikes in volatility as well as three clear performance collapses (at around timesteps $1.5e6, 3e6$ and $4e6$. The latter are probably caused by the algorithm visiting what are known as *catastrophic states* [24], which are states of the environment that the agent has never visited before or it did but a long time ago (known as *catastrophic forgetting*), and thus the Q-value estimation is erroneous which at the same time can lead with a high probability to new catastrophic states. Even so, we observe that the algorithm is quick to recover by training the newly found states.

This is a clear example of one of the biggest problems in RL, the trade-off between exploration and exploitation (see 4.1.2). Since TD3 does not tackle this issue explicitly, the consequences are greater than in other algorithms (such as SAC or PPO).

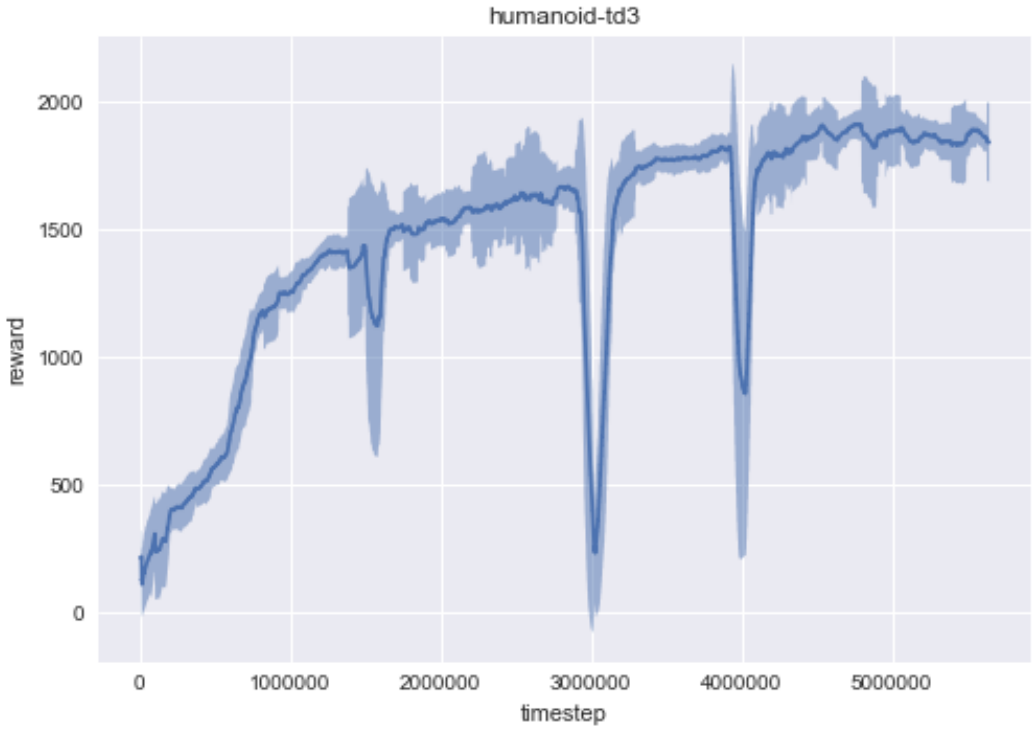


Figure 12: TD3 algorithm average reward per time step for the best performing configuration.

6.1.2 SAC

A detailed explanation of the algorithm can be found in 4.1.6.4. Similarly to TD3, SAC uses two critic networks and one actor, but the main difference is in the definition of the Q-value. We have seen that TD3 can suffer from overestimation problems, and in order to sort this problem out, SAC modifies the Q-value by adding to the cumulative reward the policy's entropy for the given state as a bonus, thus encouraging policies with higher entropy, that is, that select actions in a more "random" manner.

As we observe in the plot in Figure 13, that shows the best performing SAC training (with the hyperparameters configuration presented in Table 4), the training seems a lot more smoother and less volatile.

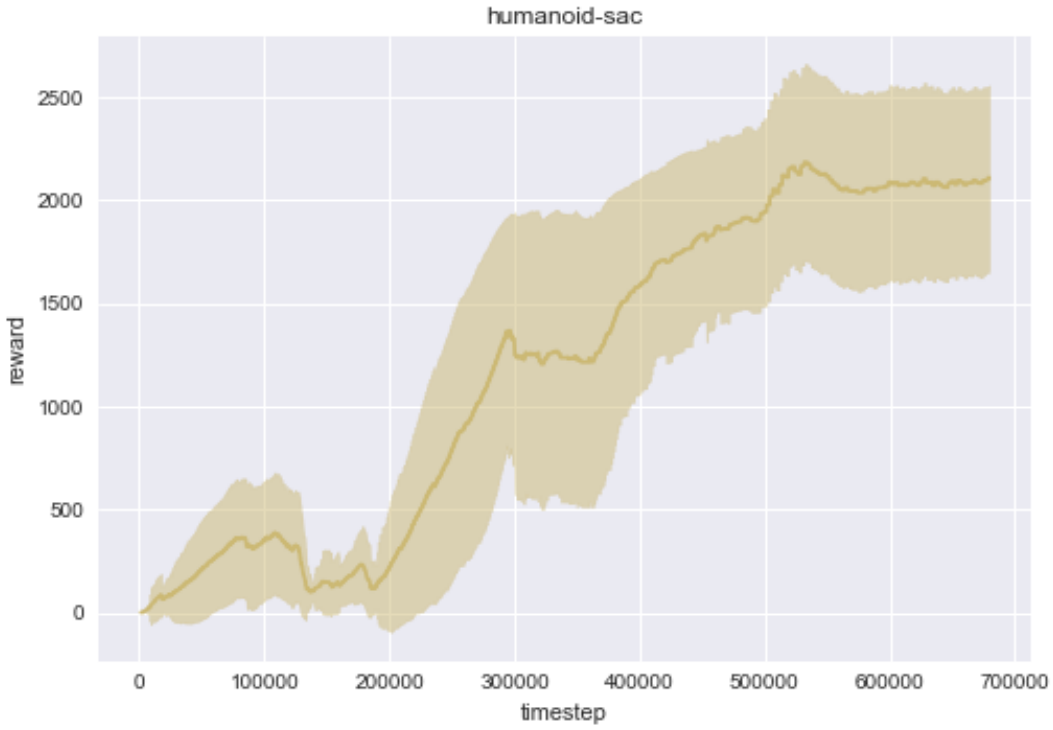


Figure 13: SAC algorithm average reward per time step for the best performing configuration.

Hyperparameter	**Symbol**	**Value**
Tau	τ	0.005
Horizon	h	1000
Actor learning rate		0.001
Critic learning rate		0.001
Discount factor	γ	0.99
Batch size		256
Actor hidden layers		[512, 512]
Critics hidden layers		[256, 256]
Activation function		ReLU
Initial alpha	α	1.0

Table 4: Hyperparameters with best results for SAC.

6.1.3 PPO

A detailed explanation of the algorithm can be found in 4.1.6.5. Differently from TD3 and SAC, PPO tackles the stability issue with a much more invasive methodology: it reduces the objective by clipping the ratio $\frac{\pi_\theta(a|s)}{\pi_{\theta new}(a|s)}$, and thus the update of the policy is constrained at each iteration. This translates into a much more stable training, as can be seen in Figure 14, and increases the probability of convergence.

Observing the reward evolution plot in Figure 14, it is noticeable that the amount of time steps is much higher than in the TD3 and SAC trainings, more specifically one order of mag-

nitude higher. That is due to the lower sample efficiency of on-policy methods, that directly translates into a higher variance, as explained in Section 4.1.3. Even so, the average time it takes per time step is considerably lower than TD3 and SAC, thus compensating the low sample efficiency.

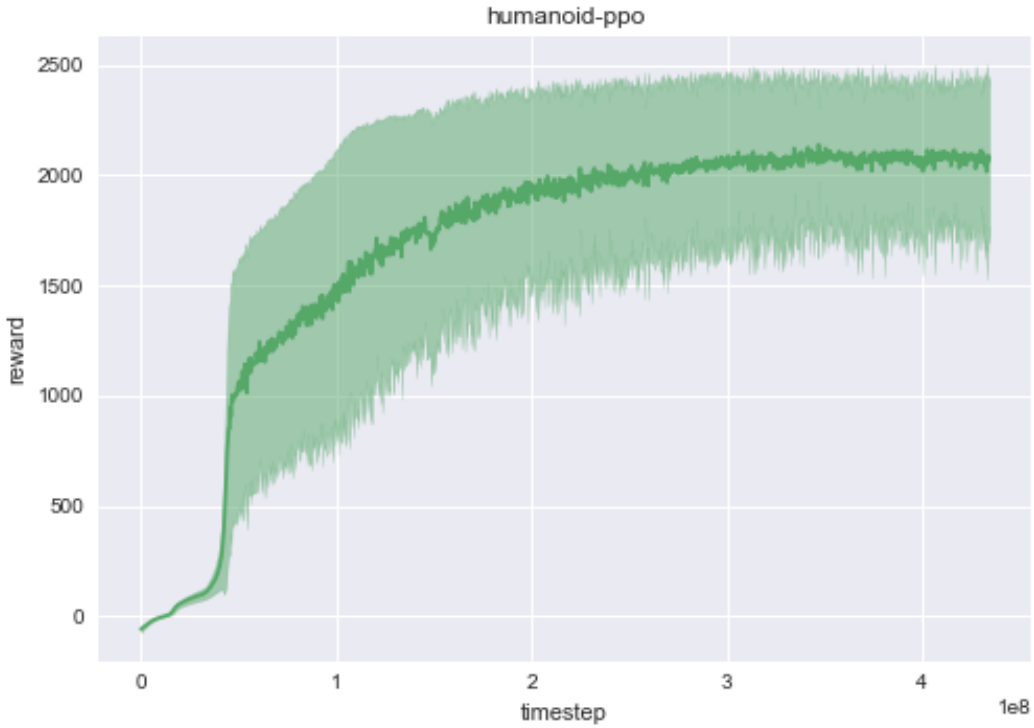


Figure 14: PPO algorithm average reward per time step for the best performing configuration.

The hyperparameters that obtained the highest reward are shown in Figure 5.

Hyperparameter	**Symbol**	**Value**
Horizon	h	1000
Actor GAE lambda	λ	1.0
Learning rate		0.0005
Discount factor	γ	0.99
Batch size		4000
SGD minibatch size		256
Clip parameter	ϵ	0.3

Table 5: Hyperparameters with best results for PPO.

6.2 Execution time

The execution time is a critical metric to monitorize during training of algorithms. Time is always related to cost: in the CTE-POWER cluster resources are limited and a training job does not free them until it is finished, thus reducing execution time is key; in AWS SageMaker you are actually paying for all the seconds that the training job servers are up and running.

Consequently the best resource configuration is the one that achieves the best time and cost efficiencies, that is, the best ratio between the total execution time and the total cost of the execution.

In Figure 8 we observe the architecture and resource allocation for the training jobs execution. In this section, we analyze the effects that different resource allocation and workers configurations have in the training execution time.

6.2.1 TD3

For the TD3 algorithm, we can observe in Figures 15 and 16 the execution time as a function of the number of workers used. As the number of workers increases, there is a clear reduction of the time spent per iteration. The speedup obtained between the execution with just one worker and the one with 64 and a GPU is around $\times 13$.

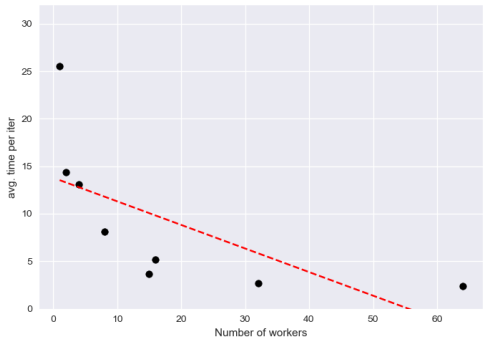


Figure 15: Training iteration time by number of workers (TD3).

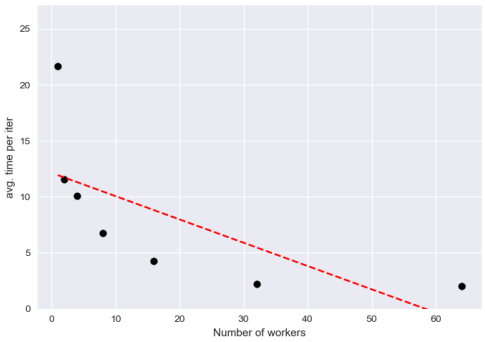


Figure 16: Training iteration time by number of workers, with GPU (TD3).

Num workers	GPU	CPUs / worker	Iteration time (s)	Speedup
1	0	1	25.56	1.00
1	1	1	21.69	1.18
2	0	1	14.38	1.78
2	1	1	11.58	2.21
4	0	1	13.10	1.95
4	1	1	10.09	2.53
8	0	1	8.12	3.15
8	1	1	6.76	3.78
15	0	1	3.63	7.04
16	0	1	5.20	4.92
16	1	1	4.24	6.04
32	0	1	2.68	9.52
32	1	1	2.19	11.69
64	0	1	2.35	10.88
64	1	1	1.99	12.86

Table 6: Iteration time and speedup per resource configuration (TD3).

6.2.2 SAC

For the SAC algorithm, we see that the executions with only one worker perform incredibly slower than the rest. Only by adding three more workers, the speedup is of around ×3. Using 64 workers and GPU, the speedup gets up to ×36 with respect to one worker without GPU.

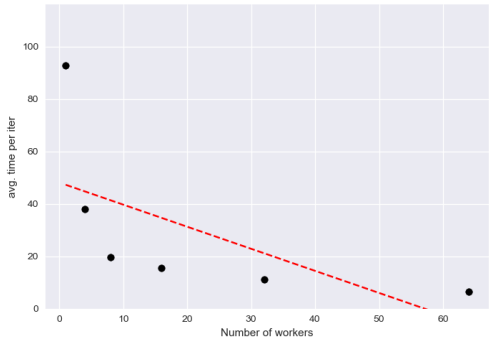


Figure 17: Training iteration time by number of workers (SAC).

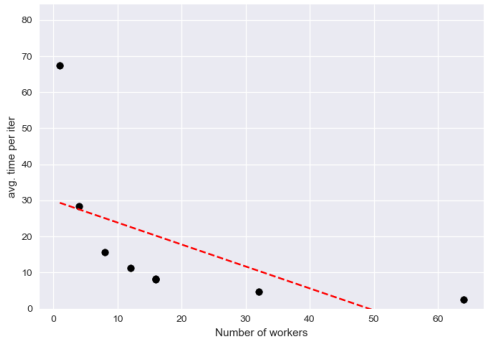


Figure 18: Training iteration time by number of workers, with GPU (SAC).

Num workers	GPU	CPUs / worker	Iteration time (s)	Speedup
1	0	1	92.87	1.00
1	1	1	67.49	1.38
4	1	1	28.46	3.26
4	0	1	37.83	2.45
8	1	1	15.65	5.93
8	0	1	19.48	4.77
12	1	1	11.32	8.20
16	1	1	8.04	11.56
16	0	1	15.46	6.01
16	1	1	8.18	11.35
32	1	1	4.72	19.67
32	0	1	11.17	8.31
64	1	1	2.55	36.45
64	0	1	6.37	14.57

Table 7: Iteration time and speedup per resource configuration (SAC).

6.2.3 PPO

In the PPO algorithm, the tendency is maintained and the parallelization of the work is very successful. The speedup gets as high as ×16 when using 64 workers with GPU instead of a single one.

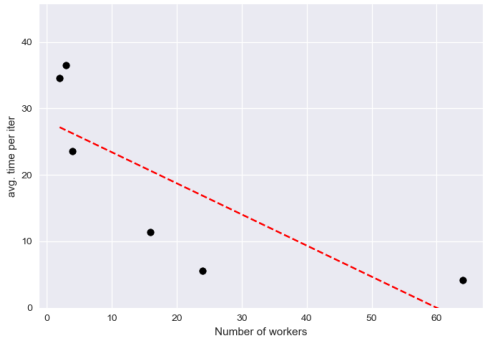


Figure 19: Training iteration time by number of workers (PPO).

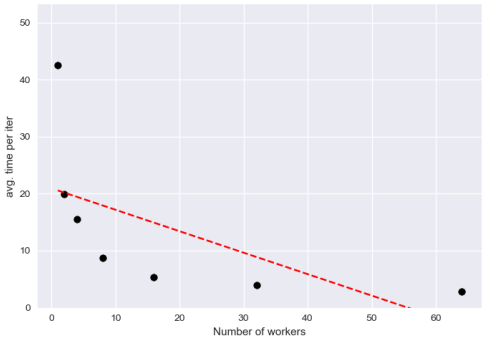


Figure 20: Training iteration time by number of workers, with GPU (PPO).

Num workers	GPU	CPUs / worker	Iteration time (s)	Speedup
1	1	1	42.59	1.00
2	0	1	34.54	1.23
2	1	1	19.88	2.14
3	0	1	36.55	1.17
4	0	1	23.53	1.81
4	1	1	15.53	2.74
8	1	1	8.76	4.86
16	0	1	11.34	3.76
16	1	1	5.33	8.00
24	0	1	5.57	7.65
32	1	1	3.91	10.88
64	0	1	4.13	10.30
64	1	1	2.78	15.30

Table 8: Iteration time and speedup per resource configuration (PPO).

The PPO implementation in RLlib also keeps track of the time spent in each of the modules (see Figure 8): sampling, loading (passing the data from the sampling workers to the learning workers), learning (optimizing the agent) and updating (passing the new policy from the learning to the sampling workers). As we could expect, the vast majority of the time is spent in the sampling and learning modules.

In Figures 21 and 22 we observe how the proportion of the total time corresponding to sampling gets drastically reduced when adding new workers. On the other hand, the time spent in the learning task is maintained relatively constant and thus it becomes responsible for most of the spent time when the sampling is highly parallelized. Another interesting insight is that the loading and updating tasks increment inn due to the number of workers: the data must be transferred among all workers, thus if these are scaled the tasks' time increases accordingly.

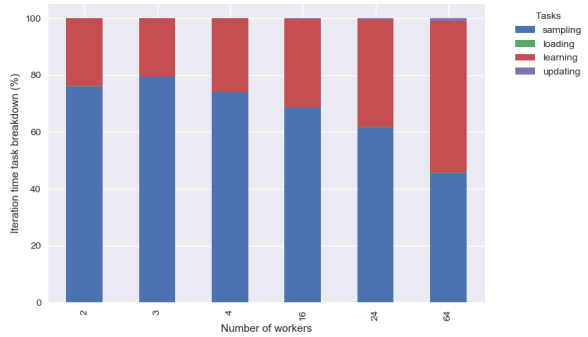


Figure 21: PPO training iteration time by task.

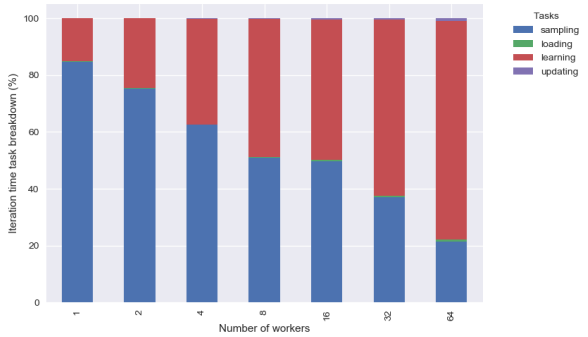


Figure 22: PPO training iteration time by task, with GPU.

6.3 Examples

Renderized videos of the trained agents interacting with the **HumanoidBulletEnv-v0** envirionment can be found in the GitHub repository, at `https://github.com/miquelescobar/ddrl/evaluation/videos/`. As we would expect, the agent walks better, faster and without falling when its obtained reward during training is higher.

Figure 23 shows the sequence of an agent that gets into a position (in the last frame) where it is perceived as *not alive* (see Equation 1), thus being clearly a failed attempt.

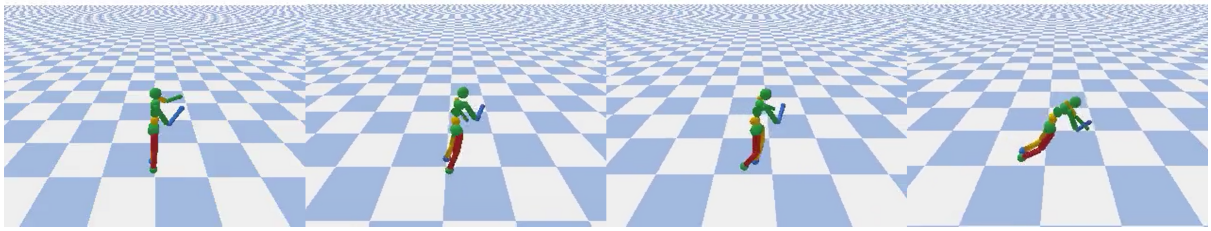


Figure 23: Sequence in which the agent becomes *not alive* in the last frame.

In the second sequence (Figure 24), we observe one of the solutions found by the agent, which consists of maintaining the whole body practically straight and perpendicular to the floor except for the two arms, which are used to maintain equilibrium.

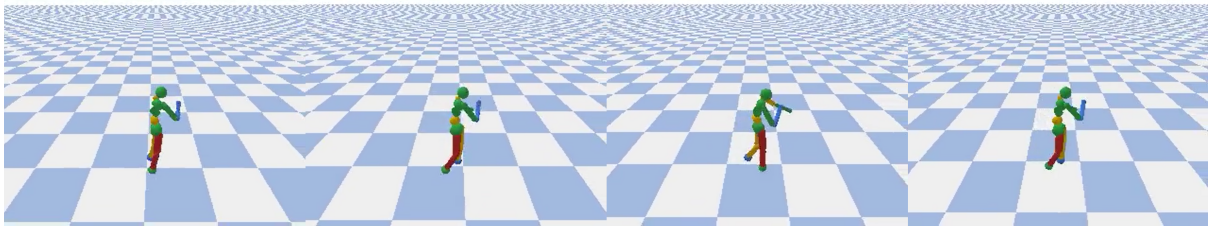


Figure 24: Sequence in which the agent uses the arms to maintain equilibrium.

Another interesting approach one of the agents made was to actually bend back the head and the upper body to not fall onwards while running, as can be seen in Figure 25.

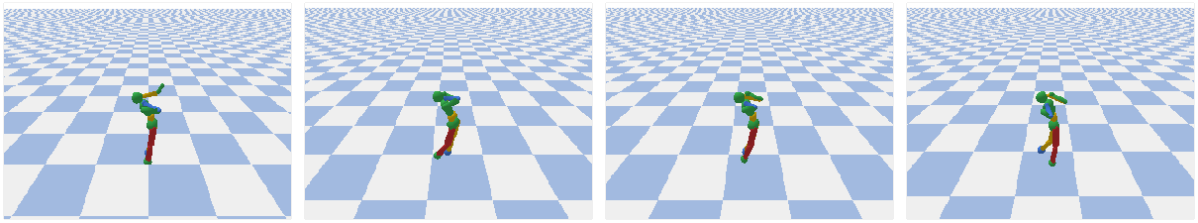


Figure 25: Sequence in which the agent compensates its inertia by bending the head back.

7 Conclusions

7.1 Personal conclusions

During the development of this project, I have learned a lot about Reinforcement Learning, HPC and cloud computing at a theoretical and practical level. Furthermore, I have gained experience in the field of research, by working with a great investigation group with talented colleagues.

Another aspect which this project has helped me improve is the management of tasks and time, as well as the handling of unexpected situations or errors. The weekly meetings with the team played a huge role in handling these situations, as being able to receive educated opinions from multiple co-workers made it easy to evaluate and consider a wide range of options.

Overall, I strongly believe that this work will be of use for future research and applications developed in the group, which adds a positive sensation of completeness to the project.

7.2 Objectives fulfilment

In accordance to the defined objectives in section 1.4, the project results can be considered a success since all four objectives have been satisfied:

- The Evaluation section shows that the selected environment has been solved with a successful agent.
- A reproducible implementation for DRL algorithms training in the CTE-POWER cluster can be found in the repository. The implementation allows to configure the desired resource allocation, parallelization and distributed computing.
- A benchmarking in terms of results and training time has been made. See Evaluation for a more detailed explanation.
- A reproducible and parametrizable training pipeline deployment script for AWS SageMaker can be found and documented in the repository.

7.3 Future work

The field of Reinforcement Learning is constantly expanding, either with the publication of new state-of-the art algorithms, the development of new frameworks or the great amount of projects that start using it. Taking this into consideration, below are listed some ideas of what parts of the project could be further explored.

- Compare the performance of the selected framework Ray to other existing frameworks.
- Implement and train DRL algorithms for applications in fields such as autonomous driving, medical images, NLP, trading, etc., and compare their performance to state-of-the-art machine and deep learning models.

- Adapting the training scripts for CTE-POWER cluster, both for parallelized and distributed computing, into another HPC system.
- Applying distributed computing executions using the cloud, as opposed to only single-node executions.
- The development of an application that uses a DRL agent trained and deployed in the cloud.
- The development of a toolkit (for instance, in the form of a CLI or a Python library) that automizes the configuration and training execution of parallelized and distributed DRL algorithms.

References

[1] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[2] J. Bergstra, D. Yamins y D. D. Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, I–115–I–123.

[3] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[4] BSC. *Emerging Technologies for Artificial Intelligence research group*. https://www.bsc.es/discover-bsc/organisation/scientific-structure/emerging-technologies-artificial-intelligence.

[5] Olivier Caelen y Gianluca Bontempi. *Improving the Exploration Strategy in Bandit Algorithms*. Dec. 2007. DOI: 10.1007/978-3-540-92695-5_5.

[6] Tianqi Chen y Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[7] Tianqi Chen et al. *MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems*. 2015. arXiv: 1512.01274 [cs.DC].

[8] R. Collobert, K. Kavukcuoglu y C. Farabet. "Torch7: A Matlab-like Environment for Machine Learning". In: *BigLearn, NIPS Workshop*. 2011.

[9] Erwin Coumans y Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2021.

[10] Alexander I Cowen-Rivers et al. "HEBO: Heteroscedastic Evolutionary Bayesian Optimisation". In: *arXiv preprint arXiv:2012.03826* (2020). winning submission to the NeurIPS 2020 Black Box Optimisation Challenge.

[11] Benjamin Ellenberger. *PyBullet Gymperium*. https://github.com/benelot/pybullet-gym. 2018–2019.

[12] Stefan Falkner, Aaron Klein y Frank Hutter. "BOHB: Robust and Efficient Hyperparameter Optimization at Scale". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy y Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 1437–1446. URL: http://proceedings.mlr.press/v80/falkner18a.html.

[13] Scott Fujimoto, Herke van Hoof y David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI].

[14] Synergy Research Group. *Cloud Market Ends 2020 on a High while Microsoft Continues to Gain Ground on Amazon*. Tech. rep. Synergy Research Group, February 2, 2021. URL: https://www.srgresearch.com/articles/cloud-market-ends-2020-high-while-microsoft-continues-gain-ground-amazon.

[15] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. 2019. arXiv: 1812.05905 [cs.LG].

[16] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG].

[17] Arthur Juliani et al. *Unity: A General Platform for Intelligent Agents*. 2020. arXiv: 1809.02627 [cs.LG].

[18] Leslie Pack Kaelbling, Michael L. Littman y Andrew W. Moore. "Reinforcement Learning: A Survey". In: *CoRR* cs.AI/9605103 (1996). URL: https://arxiv.org/abs/cs/9605103.

[19] Kirthevasan Kandasamy et al. *Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly*. 2020. arXiv: 1903.06694 [stat.ML].

[20] Lisha Li et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2018. arXiv: 1603.06560 [cs.LG].

[21] Eric Liang et al. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2018. arXiv: 1712.09381 [cs.AI].

[22] Richard Liaw et al. *Tune: A Research Platform for Distributed Model Selection and Training*. 2018. arXiv: 1807.05118 [cs.LG].

[23] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[24] Zachary C. Lipton et al. *Combating Reinforcement Learning's Sisyphean Curse with Intrinsic Fear*. 2018. arXiv: 1611.01211 [cs.LG].

[25] Yu-Ren Liu et al. *ZOOpt: Toolbox for Derivative-Free Optimization*. 2018. arXiv: 1801.00329 [cs.LG].

[26] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[27] Michael Mesnier, Gregory Ganger y Erik Riedel. "Object-based storage". In: *Communications Magazine, IEEE* 41 (Sept. 2003), pp. 84–90. DOI: 10.1109/MCOM.2003.1222722.

[28] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[29] Ling Pan et al. *Reinforcement Learning with Dynamic Boltzmann Softmax Updates*. 2019. arXiv: 1903.05926 [cs.LG].

[30] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[31] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[32] J. Rapin y O. Teytaud. *Nevergrad - A gradient-free optimization platform*. https://GitHub.com/FacebookResearch/Nevergrad. 2018.

[33] Matthew Rocklin. "Dask: Parallel computation with blocked algorithms and task scheduling". In: *Proceedings of the 14th python in science conference*. 130-136. Citeseer. 2015.

[34] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[35] David Silver et al. "Reward is enough". In: *Artificial Intelligence* 299 (2021), p. 103535. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2021.103535. URL: https://www.sciencedirect.com/science/article/pii/S0004370221000862.

[36] Richard Sutton. "Learning to Predict by the Method of Temporal Differences". In: *Machine Learning* 3 (Aug. 1988), pp. 9–44. DOI: 10.1007/BF00115009.

[37] Michel Tokic. *Adaptive -Greedy Exploration in Reinforcement Learning Based on Value Differences*. Sept. 2010. DOI: 10.1007/978-3-642-16111-7_23.

[38] Jordi Torres. *Introducción al aprendizaje por refuerzo profundo. Teoría y práctica en Python.* Barcelona: WATCH THIS SPACE Book Series. Kindle Direct Publishing, 2021. ISBN: 9798599775416.

[39] Wikipedia. *Euler angles — Wikipedia, The Free Encyclopedia.* http://en.wikipedia.org/w/index.php?title=Euler%20angles&oldid=1028147008. [Online; accessed 20-June-2021]. 2021.

[40] Andy B. Yoo, Morris A. Jette y Mark Grondona. "SLURM: Simple Linux Utility for Resource Management". In: *Job Scheduling Strategies for Parallel Processing.* Ed. by Dror Feitelson, Larry Rudolph y Uwe Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. ISBN: 978-3-540-39727-4.

[41] Matei Zaharia et al. "Apache Spark: A Unified Engine for Big Data Processing". In: *Commun. ACM* 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782. DOI: 10.1145/2934664. URL: https://doi.org/10.1145/2934664.

[42] Chiyuan Zhang et al. *A Study on Overfitting in Deep Reinforcement Learning.* 2018. arXiv: 1804.06893 [cs.LG].

A Environment specificatios

A.1 Observation space

The observation space is composed by the Euler angles α and β [39], the differences between the current and target values (x, y, z) which are constant (in fact, target z is defined as the initial value of z), the collision of the feet, and finally the relative positions and the speeds of all the joints.

Name	Type	Interval	Description
α	float	$[-\pi, \pi]$	The Euler angle α
β	float	$[-\pi, \pi]$	The Euler angle β
x_target_difference	float	$[-\pi, \pi]$	The difference between current x the constant target x
y_target_difference	float	$[-\pi, \pi]$	The difference between current y the constant target y
z_target_difference	float	$[-\pi, \pi]$	The difference between current z the initial value of z
velocity_x	float	$[-1, 1]$	The velocity of the robot at the x axis (comparing s_t to s_{t-1} positions)
velocity_y	float	$[-1, 1]$	The velocity of the robot at the y axis (comparing s_t to s_{t-1} positions)
velocity_z	float	$[-1, 1]$	The velocity of the robot at the z axis (comparing s_t to s_{t-1} positions)
feet1_contact	bool	$\{0, 1\}$	Whether or not the right feet is in contact with any other link, joint or object.
feet2_contact	bool	$\{0, 1\}$	Whether or not the left feet is in contact with any other link, joint or object.
abdomen_x_position	float	$[-1, 1]$	The relative position to the body of abdomen_x joint.
abdomen_y_position	float	$[-1, 1]$	The relative position to the body of abdomen_y joint.
abdomen_z_position	float	$[-1, 1]$	The relative position to the body of abdomen_z joint.
right_hip_x_position	float	$[-1, 1]$	The relative position to the body of right_hip_x joint.
right_hip_y_position	float	$[-1, 1]$	The relative position to the body of right_hip_y joint.
right_hip_z_position	float	$[-1, 1]$	The relative position to the body of right_hip_z joint.
left_hip_x_position	float	$[-1, 1]$	The relative position to the body of left_hip_x joint.
left_hip_y_position	float	$[-1, 1]$	The relative position to the body of left_hip_y joint.
left_hip_z_position	float	$[-1, 1]$	The relative position to the body of left_hip_z joint.
right_knee_position	float	$[-1, 1]$	The relative position to the body of right_knee joint.
left_knee_position	float	$[-1, 1]$	The relative position to the body of left_knee joint.

right_shoulder1_position	float	$[-1, 1]$	The relative position to the body of right_shoulder1 joint.
right_shoulder2_position	float	$[-1, 1]$	The relative position to the body of right_shoulder2 joint.
left_shoulder1_position	float	$[-1, 1]$	The relative position to the body of left_shoulder1 joint.
left_shoulder2_position	float	$[-1, 1]$	The relative position to the body of left_shoulder2 joint.
right_elbow_position	float	$[-1, 1]$	The relative position to the body of right_elbow joint.
left_elbow_position	float	$[-1, 1]$	The relative position to the body of left_elbow joint.
abdomen_x_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
abdomen_y_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
abdomen_z_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
right_hip_x_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
right_hip_y_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
right_hip_z_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
left_hip_z_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
right_knee_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
left_knee_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
right_shoulder1_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
right_shoulder2_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
left_shoulder1_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
left_shoulder2_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
right_elbow_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.
left_elbow_speed	float	$[-1, 1]$	The speed of the joint, mapped between -1 and 1.

Table 9: Properties of all dimensions in the observation space.

A.2 Action space

The action space is formed by 17 actuators, each corresponding to one joint of the robot. All of them can be set to any value in the $[-1, 1]$ interval. The power is the maximum force that the joint can apply (for instance, a knee is stronger than the shoulder). The resulting applied

force to each joint is thus computed as joint_input $\times$ power.

Name	Type	Domain interval	Power
abdomen_x	float	$[-1, 1]$	100
abdomen_y	float	$[-1, 1]$	100
abdomen_z	float	$[-1, 1]$	100
right_hip_x	float	$[-1, 1]$	100
right_hip_y	float	$[-1, 1]$	300
right_hip_z	float	$[-1, 1]$	100
left_hip_x	float	$[-1, 1]$	100
left_hip_y	float	$[-1, 1]$	300
left_hip_z	float	$[-1, 1]$	100
right_knee	float	$[-1, 1]$	200
left_knee	float	$[-1, 1]$	200
right_shoulder1	float	$[-1, 1]$	75
right_shoulder2	float	$[-1, 1]$	75
left_shoulder1	float	$[-1, 1]$	75
left_shoulder2	float	$[-1, 1]$	75
right_elbow	float	$[-1, 1]$	75
left_elbow	float	$[-1, 1]$	75

Table 10: Properties of all dimensions in the action space.