BACHELOR´S THESIS / ENGINEERING 2024

Exploring Generalization in Deep Reinforcement Learning for Control Tasks

Àlex Montoya Pérez

Universitat Pompeu Fabra Barcelona

Exploring Generalization in Deep Reinforcement Learning for Control Tasks

TREBALL DE FI DE GRAU DE

Àlex Montoya Pérez

Anders Jonsson & Sergio Calo Oliveira

Grau en Enginyeria en Informática

Curs 2023 - 2024

Universitat Pompeu Fabra Barcelona

Escola d'Enginyeria

A la meva gent.

Agraïments

En primer lloc, als meus directors del TFG, especialment a Sergio Calo per acceptar amb els braços oberts la meva proposta i donar-me l'oportunitat d'aprendre d'ell i amb ell. Gràcies per fer que aquest procés no sols fos educatiu, sinó també gratificant i memorable, motivant-me així a continuar explorant en aquest camp.

A la meva família, que sempre m'ha donat suport en cada idea que se m'ha acudit, adaptant-se a les meves decisions perquè pugui perseguir les meves metes. Gràcies pel seu amor incondicional, la seva paciència infinita i per creure en mi fins i tot en els moments en què jo dubtava.

A Yuriel, la meva parella i part de la família, qui és el meu suport constant en tot el que em proposo. Gràcies per estar sempre al meu costat, entendre els meus sacrificis i celebrar els meus èxits. La teva presència ha fet aquest camí més suportable i especial. Agraeixo també a la seva família per acollir-me com un més i per confiar sempre en mi.

Als meus companys d'universitat, per la seva companyia, comprensió i moments compartits que han fet d'aquest camí una experiència inoblidable.

Als meus amics, per la seva amistat incondicional malgrat els meus horaris i falta de disponibilitat, gràcies per cadascun dels moments al vostre costat.

I finalment, a mi mateix, a l'Àlex del 2020 ple de determinació i motivació, que no tem als nous reptes i sempre busca créixer i ser millor. A aquell Àlex que es va atrevir a fer un gran gir en la seva vida, conscient del sacrifici que implicava, però sense rendir-se mai. Aquest assoliment és un tribut a aquesta perseverança i a l'esperit de superació que portem dins.

Abstract

Deep reinforcement learning (DRL) has achieved significant success in addressing various tasks; however, trained agents often struggle to generalize beyond the environments in which they were initially trained. This thesis investigates the generalization capabilities of DRL algorithms through a series of independent experiments on control tasks of varying complexity. These tasks range from simple scenarios like Mountain Car to more complex challenges such as Pick and Place, utilizing the Gymnasium and Gymnasium-Robotics suites with the MuJoCo physics engine. Each experiment involves training a model using Proximal Policy Optimization (PPO) within a specific environment and subsequently evaluating its performance in slightly modified environments. The impact of hyperparameters on generalization is also examined. The objective is to identify the strengths and limitations of DRL agents in adapting learned policies to similar but altered environments.

Resum

L'aprenentatge profund per reforç (DRL) ha aconseguit un èxit significatiu en la resolució de diverses tasques; tanmateix, els agents entrenats sovint tenen dificultats per generalitzar més enllà dels entorns en els quals van ser inicialment entrenats. Aquesta tesi investiga les capacitats de generalització dels algorismes de DRL a través d'una sèrie d'experiments independents en tasques de control de diversa complexitat. Aquestes tasques varien des d'escenaris simples com Mountain Car fins a desafiaments més complexos com Pick and Place, utilitzant els conjunts d'eines Gymnasium i Gymnasium-Robotics amb el motor de física MuJoCo. Cada experiment consisteix a entrenar un model utilitzant Proximal Policy Optimization (PPO) dins d'un entorn específic i posteriorment avaluar el seu rendiment en entorns lleugerament modificats. També s'examina l'impacte dels hiperparàmetres en la generalització. L'objectiu és identificar els punts forts i les limitacions dels agents de DRL en l'adaptació de polítiques apreses a entorns similars però alterats.

Resumen

El aprendizaje profundo por refuerzo (DRL) ha logrado un éxito significativo en la resolución de diversas tareas; sin embargo, los agentes entrenados a menudo tienen dificultades para generalizar más allá de los entornos en los que fueron inicialmente entrenados. Esta tesis investiga las capacidades de generalización de los algoritmos de DRL a través de una serie de experimentos independientes en tareas de control de diversa complejidad. Estas tareas varían desde escenarios simples como Mountain Car hasta desafíos más complejos como Pick and Place, utilizando los conjuntos de herramientas Gymnasium y Gymnasium-Robotics con el motor de física MuJoCo. Cada experimento consiste en entrenar un modelo utilizando Proximal Policy Optimization (PPO) dentro de un entorno específico y posteriormente evaluar su rendimiento en entornos ligeramente modificados. También se examina el impacto de los hiperparámetros en la generalización. El objetivo es identificar las fortalezas y limitaciones de los agentes de DRL en la adaptación de políticas aprendidas a entornos similares pero alterados.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Reinforcement Learning (RL) [Sutton and Barto, 2018] is an exciting and autonomous learning approach where an intelligent agent strives to master tasks through trial and error in an initially unknown environment. Unlike standard machine learning (ML) methods [Mukhamediev, 2015], which often require pre-existing datasets, RL liberates itself from this limitation by enabling agents to acquire knowledge directly from their interactions with the environment.

Deep Reinforcement Learning (DRL) [Francois et al., 2018] has emerged as an important family of techniques that may support the development of intelligent systems capable of learning to achieve goals in a variety of complex real-world environments [Mnih et al., 2016], representing a groundbreaking paradigm within the field of AI. Unlike traditional RL, which relies on tabular methods or linear function approximation, DRL leverages the power of Artificial Neural Networks (ANNs) [Mijwil, 2021], inspired by the human brain, with neurons acting as the fundamental building blocks which allows DRL to handle high-dimensional state and action spaces effectively.

A desirable characteristic of such intelligent systems is the ability to function in diverse environments, including those never encountered before. However, DRL algorithms are typically trained and evaluated on a fixed environment. These evaluations focus on the algorithms' ability to optimize a policy in a specific complex environment, rather than their ability to learn representations that generalize to previously unseen circumstances.

Despite the considerable advancements in the field, each study within this domain adopts disparate sets of environments, variations, and experimental methodologies. This lack of uniformity underscores the need for a methodical, empirical investigation into generalization within DRL, characterized by well-defined environments, metrics, and baselines. The primary contribution of this thesis lies in its empirical scrutiny of the generalization efficacy of DRL algorithms across environments commonly employed in prior DRL generalization research. Through a meticulously designed series of independent experiments encompassing control tasks of varying complexities and more sophisticated challenges like Pick and Place, this study utilizes both the Gymnasium and Gymnasium-Robotics suites, leveraging the MuJoCo physics engine to facilitate comprehensive analysis and understanding.

1.1 Objectives

The primary objective of this thesis is to conduct a thorough analysis of the generalization capabilities of DRL algorithms through a series of meticulously designed controlled experiments. By examining how these algorithms perform when faced with environments that differ slightly from their training settings, this study aims to identify both the strengths and limitations of DRL agents in adapting their learned policies to new, yet related, contexts. Additionally, the research will scrutinize the impact of various hyperparameters on the training and generalization performance of these agents, providing a deeper understanding of the factors that influence their ability to generalize. Ultimately, the insights gained from this investigation will contribute to enhancing the robustness and adaptability of DRL agents, paving the way for their successful deployment in practical, real-world applications where dynamic and unpredictable environments are the norm.

1.2 Thesis Organization

Chapter 2 introduces fundamental concepts of RL, Artificial Neural Networks (ANNs), and DRL, alongside discussing generalization in DRL and tools like OpenAI Gym and the MuJoCo physics engine.

Chapter 3 outlines the experimental setup, including environments, programming languages, and computing clusters. It details DRL agent implementation and specific experiments conducted to evaluate their generalization.

Chapter 4 presents experiment results, analyzing DRL agent performance in training and modified environments, and exploring the impact of hyperparameters.

Chapter 5 interprets results, comparing them with existing literature and discussing implications for DRL and real-world applications.

Chapter 6 summarizes key findings, highlights contributions, acknowledges limitations, and suggests future research directions.

Appendices offer additional environment details, extended result analyses, and supplementary information.

Chapter 2

BACKGROUND THEORY

This chapter provides the foundational knowledge necessary to investigate the generalization capabilities of DRL algorithms. It presents essential concepts such as RL principles, ANNs and key DRL techniques like Experience Replay Buffer (ERB) and Actor-Critic methods. Additionally, it delves into the challenge of generalization in DRL and introduces tools such as the OpenAI Gym frameworks and the MuJoCo physics engine. This chapter serves as a cornerstone for the empirical exploration conducted in the thesis, with the aim of enhancing the adaptability of DRL agents across diverse environments.

2.1 Reinforcement Learning

The essence of RL centers on an ongoing journey, reminiscent of a beginner acquiring a fresh set of skills. In a similar manner, the RL agent traverses its environment, taking actions and receiving rewards, as seen in Figure 2.1. This constant cycle serves as a melting pot in which the agent hones its decision-making abilities and deepens its comprehension of the environment's nuances.

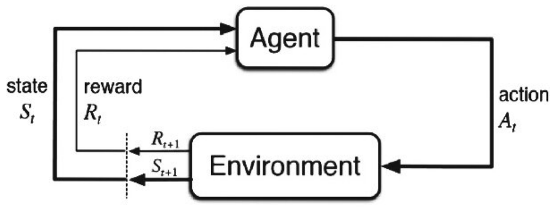

Figure 2.1: The agent-environment interaction. Source:[Sutton and Barto, 2018]

The distinctive feature of RL, setting it apart from traditional ML paradigms, lies in its emphasis on experiential learning. Rather than passively ingesting knowledge from annotated datasets, RL agents actively engage with their environments, usually modeled as a Markov Decision Process (MDP), see Section 2.1.1, accumulating insights through firsthand experience. This self-driven learning process underscores RL's autonomy and adaptability, enabling agents to learn optimal strategies through iterative experimentation and exploration. RL focuses on finding an optimal policy [Comanici and Precup, 2010] rather than analyzing data.

2.1.1 Markov Decision Processes

Markov Decision Processes (MDPs) [Otterlo and Wiering, 2012] serve as mathematical frameworks to formalize decision-making problems encountered in environments where outcomes are uncertain and influenced by the actions taken by a decision-maker, or agent. MDPs offer a structured approach for sequential decision-making under uncertainty, rendering them particularly adept for modeling and resolving RL challenges [Wang and Snooks, 2021].

Components of MDPs

MDPs are tuples $(S, A, T, R, , \gamma)$ which defines a set of states, actions, transition probabilities, rewards, and a discount factor, which together encapsulate the dynamics of the decision-making process.

- **States (S):** Set of environmental states, denoted as S, is defined as a finite collection $\{s_1, \ldots, s_N\}$ where N represents the size of the state space, i.e., $|S| = N$. A state serves as a unique representation encompassing all pertinent information within a problem's context. For instance, in a Pick and Place scenario, a state might include the positions of objects to be picked and the location of the robot arm. In subsequent chapters, we will explore the use of features to describe states.

- **Actions (A):** The action set A comprises a finite collection $\{a_1, \ldots, a_K\}$, where K denotes the size of the action space, i.e., $|A| = K$. Actions are instrumental in governing the system state. For example, in a Pick and Place scenario, actions might include "pick object", "move arm left", "move arm right", "place object", etc. The subset of actions applicable in a specific state $s \in S$ is denoted as $A(s)$, with $A(s) \subseteq A$.

- **Transition Function (T):** When an action $a \in A$ is applied in state $s \in S$, the system transitions to a new state $s_{t+1} \in S$ based on a probability distribution over possible transitions. This transition function, denoted as $T : S \times A \rightarrow \Delta(S)$, where $\Delta(S)$ is the set of all probability distributions over the set S, specifies the probability distribution of the next state given the current state and action. Specifically, $T(s, a, s_{t+1})$ represents the probability of transitioning from state s to state s_{t+1} after performing action a.

- **Reward Function** (**R**): Assigns rewards to states or actions. While $R : S \to \mathbb{R}$ gives rewards for states, $R : S \times A \times S \to \mathbb{R}$ assigns rewards for actions in states, and $R : S \times A \times S \to \mathbb{R}$ assigns rewards for state transitions. These formulations guide the learning objective in the MDP. For instance, in a Pick and Place scenario, successful object placements may receive positive rewards, failed attempts negative rewards, and intermediary states could have non-zero rewards to define sub-goals for learning.

- **Discount Factor (**γ**):** Represents the difference in importance between short and long term rewards. It influences the value of future rewards relative to immediate rewards, shaping the agent's decision-making process over time. The discount factor γ can take values from the interval (0, 1], where a value closer to 0 makes the agent prioritize immediate rewards, while a value closer to 1 makes the agent value future rewards more heavily.

Markov Decision Process Property

In a MDP, the agent starts in an initial state s_0 and chooses actions based on a policy, see Section 2.1.2. Upon taking an action a_t, the environment transitions to a new state s_{t+1} according to the transition probabilities, and the agent receives a reward r_t, as seen in Figure 2.1. This process repeats iteratively, with the agent aiming to maximize the expected cumulative reward over time.

An state s_t is said to have the Markov property if, and only if, the equality 2.1 holds:

$$\begin{aligned} &P(s_{t+1} = s_{t+1}, r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0) \\ &= P(s_{t+1} = s_{t+1}, r_{t+1} = r | s_t, a_t). \end{aligned} \tag{2.1}$$

In other words, the environment's reaction at time $t + 1$ relies solely on the state of the preceding time step t, rather than considering the complete history of states, actions, and rewards. Consequently, it can be inferred that the current state s_t encapsulates information regarding past occurrences in earlier time steps. All states in the environment must hold the Markov property to be considered as a MDP.

2.1.2 Policies and Value Functions

A policy π represents a strategy that dictates the agent's behavior in an environment. It defines the actions the agent takes in each state s_t. Formally, a policy can be either deterministic or stochastic. A deterministic policy is a function $\pi : S \to A$, mapping each state $s \in S$ to a specific action $a \in A$. A stochastic policy is a function $\pi : S \to \Delta(A)$, where $\Delta(A)$ is the set of all probability distributions over the action set A, mapping each state $s \in S$ to a probability distribution over actions.

State Value Function

For a given policy π, the state value function $V^\pi(s)$ represents the expected return when starting in state s and following policy π thereafter, as seen in Equation 2.2[1].

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[R_t \mid s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right] \\ &= \sum_{s_{t+1}} T(s, \pi(s), s_{t+1})\Big(R(s, a, s_{t+1}) + \gamma V^\pi(s_{t+1})\Big). \end{aligned} \tag{2.2}$$

State-Value Function

Similarly, a state-action value function $Q^\pi(s, a)$, Equation 2.3, calculates the expected return from starting in state s, taking action a, and following policy π.

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t \mid s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \\ &= \sum_{s_{t+1}} T(s, a, s_{t+1})\Big(R(s, a, s_{t+1}) + \gamma V^\pi(s_{t+1})\Big). \end{aligned} \tag{2.3}$$

Goal in a MDP

The goal in a MDP is to find the best policy, maximizing the value function for all states, Equation 2.4. An optimal policy π^* ensures that its optimal value function $V^{\pi^*}(s)$ satisfies the property $V^{\pi^*}(s) \geq V^\pi(s)$ for all states.

$$\begin{aligned} &V^{\pi^*}(s) = \max_\pi V^\pi(s) = \max_a Q^{\pi^*}(s,a), \\ &Q^{\pi^*}(s,a) = \max_\pi Q^\pi(s,a). \end{aligned} \tag{2.4}$$

[1]Note that $\mathbb{E}$ is used for the expected value under policy π, representing the expected return from starting in state s, taking action a, and following policy π.

The **Bellman Optimality Equation** [Bellman, 1958] characterizes the optimal solution, Equation 2.5, stating that the value of a state under an optimal policy equals the expected return for the best action in that state.

$$\pi^*(s) = \arg\max_a \sum_{s \in \mathcal{S}} T(s,a,s_{t+1}) \Big(R(s,a,s_{t+1}) + \gamma V^*(s_{t+1}) \Big) = \arg\max_a Q^{\pi^*}(s,a). \tag{2.5}$$

The $\pi^*(s)$ selects the best action using the $V^{\pi^*}(s)$, while the $Q^{\pi}(s,a)$ combines rewards and values of possible next states without requiring a forward-reasoning step. This allows for learning Q-functions instead of V-functions in model-free approaches when transition and reward functions are unknown. The relationship between $Q^{\pi^*}(s,a)$ and $V^{\pi^*}(s)$, Equation 2.6, allows for optimal action selection based on $Q^{\pi^*}(s,a)$ without consulting the MDP model, simplifying the decision-making process.

$$\begin{aligned} Q^*(s,a) &= \sum_{s_{t+1} \in \mathcal{S}} T(s,a,s_{t+1})[R(s,a,s_{t+1}) + \gamma V^*(s_{t+1})] \\ V^*(s) &= \max_a Q^*(s,a). \end{aligned} \tag{2.6}$$

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) [Mijwil, 2021] serve as models inspired by the human brain, with neurons acting as the fundamental building blocks. While the human brain comprises approximately 100 billion neurons, each forming connections with thousands to hundreds of thousands of other neurons, ANNs mimic this structure, illustrated in Figure 2.2, by organizing neurons into layers and establishing connections between them.

In the context of RL, see Section 2.1, ANNs are widely employed to approximate value functions or policies, see Section 2.1.2. Mathematically, an ANN can be represented as a function $f : \mathbb{R}^n \to \mathbb{R}^m$, where n is the dimensionality of the input space and m is the dimensionality of the output space. The function f maps input vectors to output vectors, thus enabling the network to learn complex mappings between states and actions.

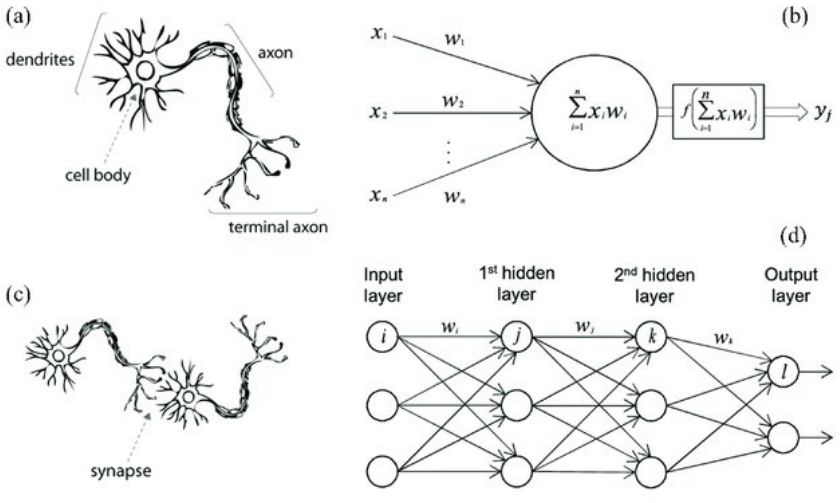

Figure 2.2: A biological neuron in comparison to an artificial neural network: (a) human neuron; (b) artificial neuron; (c) biological synapse; and (d) ANN. Figure extracted from [Meng et al., 2020].

An ANN typically comprises an input layer, one or more hidden layers, and an output layer. Let L denote the total number of layers in the network, with $L = 0$ representing the input layer, $L-1$ representing the output layer, and 1 to $L-2$ representing the hidden layers. Each neuron i in a layer l computes its output y_i^l based on the weighted sum of inputs from the previous layer, followed by an activation function σ, as seen in Equation 2.7:

$$y_i^l = \sigma\left(\sum_{j=1}^{n^{l-1}} w_{ij}^l y_j^{l-1} + b_i^l\right), \tag{2.7}$$

where w_{ij}^l represents the weight of the connection between neuron i in layer l and neuron j in layer $l-1$, and b_i^l is the bias term associated with neuron i in layer l.

During training, the network adjusts its weights and biases to minimize a predefined loss function $\mathcal{L}$, Equation 2.8, which quantifies the disparity between the predicted output and the actual target. This is typically done using optimization algorithms such as Gradient Descent [Ruder, 2016]. The weights are updated iteratively according to the gradient of the loss function with respect to the weights:

$$w_{ij}^l \leftarrow w_{ij}^l - \alpha \frac{\partial \mathcal{L}}{\partial w_{ij}^l}, \tag{2.8}$$

where α denotes the learning rate, controlling the magnitude of weight updates.

The training process aims to find the optimal set of weights and biases that minimize the prediction error on the training data, thereby enabling the network to generalize well to unseen data.

ANNs play a crucial role in RL by approximating complex value functions or policies necessary for decision-making in various tasks. Their ability to learn from data and generalize to new situations makes them powerful tools for solving a wide range of problems in robotics and artificial intelligence.

2.3 Deep Reinforcement Learning

Traditional RL methods, see Section 2.1, often struggle to cope with the complexity of real-world scenarios due to their limited capacity to generalize across diverse states and actions.

DRL addresses this limitation by integrating ANN into the learning process, as Figure 2.3 illustrates. ANNs, excel at approximating complex functions and extracting meaningful representations from inputs playing a pivotal role in capturing the intricate relationships between environmental states, actions, and rewards, enabling agents to learn efficient and adaptable control policies.

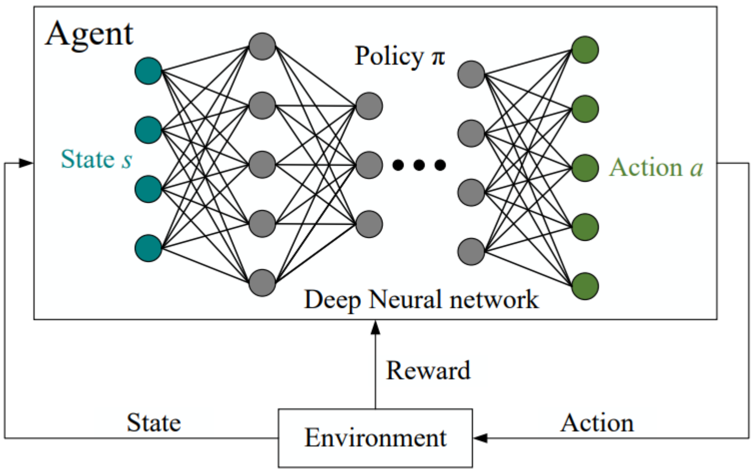

Figure 2.3: Deep Reinforcement Learning Interaction. Source: [Yi and Liu, 2023].

In the subsequent sections, we delve deeper into Experience Replay Buffer (ERB) strategy, see Section 2.3.1, the mechanics of Deep Q Network (DQN), see Section 2.3.2, and Actor-Critic algorithms, see Section 2.3.3, elucidating their underlying principles, training procedures, and empirical performance.

2.3.1 Experience Replay Buffer

The Experience Replay Buffer (ERB) [Li et al., 2022], introduced as a cornerstone technique in DRL has evolved into a fundamental practice across various DRL algorithms, including DQN, see Section 2.3.2, and Actor-Critic methods, see Section 2.3.3. Its integration is attributed to its profound impact on enhancing learning performance.

Experience Replay Buffer (ERB) operates by storing past experiences in a memory buffer, creating a reservoir of diverse interactions between the agent and its environment. These stored experiences are typically tuples of the form (s, a, r, s_{t+1}), where s represents the state, a the action taken, r the reward received, and s_{t+1} the subsequent state. During training, the agent selectively samples from this buffer, leveraging the stored experiences to learn from a wide range of scenarios. By revisiting and learning from a multitude of previously encountered situations, the agent gains the ability to generalize better and navigate through complex environments more effectively. This approach not only promotes efficient data utilization but also mitigates the issues of correlated data, thereby fostering stable and accelerated learning.

Nevertheless, while ERB offers benefits, it also brings forth several challenges. Firstly, efficiently storing and managing the memory buffer becomes crucial as it can rapidly expand in size, demanding considerable resources. Secondly, striking a balance between exploration and exploitation is paramount; relying too heavily on replayed data can dampen the agent's curiosity and diversity. Thirdly, adapting to non-stationary environments poses a significant hurdle, as outdated data reproduction can impede the agent's ability to adapt and generalize effectively.

2.3.2 Deep Q Network

The traditional Q-learning algorithm in RL [Jang et al., 2019] struggles to handle large-scale or continuous space MDP due to the curse of dimensionality associated with complex networks. To address this challenge, DeepMind introduced the Deep Q-Network (DQN) algorithm [Fernandez-Fernandez et al., 2023], which approximates the Q-Table using an ANN, as Figure 2.4 illustrates. By inputting the current state s_t, the DQN algorithm predicts the Q-value of each action efficiently, thus eliminating the need for the cumbersome Q-table.

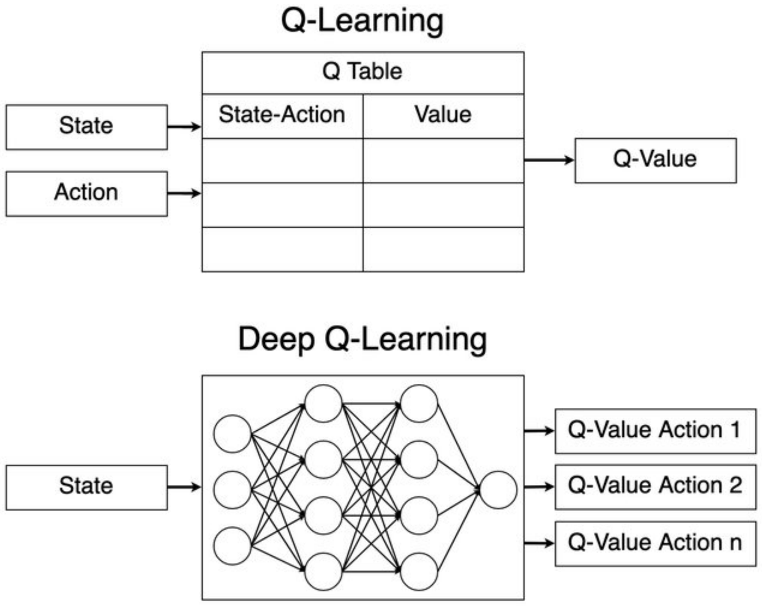

Figure 2.4: Q-Learning vs. Deep Q-Learning. Source: [Sebastianelli et al., 2021].

The agent is responsible for learning, while the environment presents specific problems for the agent to interact with. The primary goal of the DQN algorithm is to enable the agent to learn the optimal action selection and maximize subsequent rewards. The agent's role includes action selection a_t and neural network training, while the environment updates the state s_t and calculates rewards r_t.

The DQN algorithm employs two multilayer perceptron ANN:

- **Eval-net:** ANN responsible for calculating the actual Q-Value based on the current state-action pairs. It uses the following formula:

$$Q_{eval}(s_t, a_t; \theta^E), \tag{2.9}$$

  where θ^E represents the parameters of the evaluation network.

- **Target-net:** ANN responsible for estimating the Q-Value based on the current state-action pairs. It uses the following formula:

$$Q_{target}(s_{t+1}, a; \theta^T), \tag{2.10}$$

  where θ^T represents the parameters of the target network.

The agent selects the next action based on these networks, and experiences (s_t, a_t, r_t, s_{t+1}) are stored in an ERB, see Section 2.3.1, and randomly sampled for training eval-net. Eval-net's parameters are continually updated based on a loss function, Equation 2.11, while target-net's parameters are copied from eval-net every k iterations to ensure algorithm convergence.

The loss function used for training the evaluation network is defined as the mean squared error (MSE) between the predicted Q-values and the target Q-values:

$$\mathcal{L}(\theta^E) = \mathbb{E}_{(s,a,r,s_{t+1})\sim rand(ERD)} \left[\left(Q_{eval}(s, a; \theta^E) - y \right)^2 \right], \tag{2.11}$$

where θ^E are the parameters of the evaluation network, $rand(ERD)$ represents a minibatch sampled randomly from the ERB, and y is the target Q-value, typically computed using the target network.

The target Q-value used in the loss function is calculated as follows:

$$y = r + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^T), \tag{2.12}$$

where r is the reward received after taking action a in state s, s_{t+1} is the next state, γ is the discount factor, and θ^T are the parameters of the target network.

The parameters θ^E of the evaluation network are updated using gradient descent to minimize the loss function:

$$\theta^E_{t+1} = \theta^E_t - \alpha \nabla_{\theta^E} \mathcal{L}(\theta^E), \tag{2.13}$$

where α is the learning rate.

The parameters θ^T of the target network are updated periodically by copying the parameters from the evaluation network:

$$\theta^T_{t+1} = \theta^E_t. \tag{2.14}$$

This update is performed every k iterations.

2.3.3 Actor Critic

Actor-critic algorithms [Konda and Tsitsiklis, 2001] represent a specific class of Temporal Difference (TD) methods within RL. Unlike traditional TD methods where a single value function represents both policy and value estimates, actor-critic methods maintain separate structures for these components creating the following two ANN:

- **Actor:** Responsible for decision-making, the actor selects actions according to the prevailing policy. Its primary task involves exploring the action space to optimize cumulative rewards. Through ongoing policy refinement, the actor dynamically adjusts to changes in the environment.

- **Critic:** Tasked with assessing the actor's decisions, the critic estimates the value function and evaluates the actor's actions. offering feedback on their efficacy. Playing a crucial role, the critic directs the actor towards actions likely to yield greater returns, thereby enhancing the learning process as a whole.

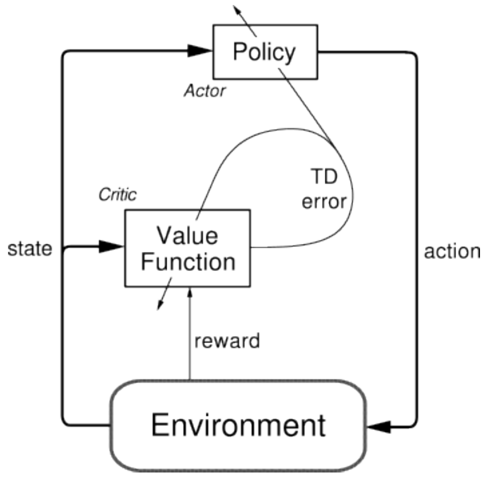

Figure 2.5: The Actor-Critic architecture. Source: [Sutton and Barto, 2018].

The actor-critic framework, Figure 2.5, assesses and offers feedback on the ongoing execution of the current policy by the actor. This assessment relies on computing a Temporal Difference (TD) error, often referred to as the advantage function, denoted as $A_t(s_t, a_t)$ and represented as Equation 2.15.

$$A_t(s_t, a_t) = R(s_t, a_t) + V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t). \tag{2.15}$$

The formulation of the actor's policy gradient, usually modelled with a parameterized function respect to θ, denoted π_θ, stems from the REINFORCE algorithm [Williams, 1992], which updates the policy parameters in a Monte-Carlo style, detailed in [Wang and Snooks, 2021], sampling the total return from the entire trajectory. However, in actor-critic methods, a bootstrap approach is employed. This introduces a significant alteration to the advantage function, incorporating a baseline $b(s_t)$ as depicted below in Equation 2.16.

$$b(s_t) = V^{\pi_\theta}(s_t). \tag{2.16}$$

The advantage function serves as the primary feedback signal, guiding learning for both the actor and the critic. When its positive, it suggests that the tendency to choose action should be reinforced in future selections. Conversely, a negative advantage function implies that this inclination should be diminished.

As previously discussed, the actor's learning process is built upon the policy gradient approach denoted by π_θ. This method employs an expression for the policy gradient of the actor, $\nabla_\theta J(\theta)$, as represented by Equation 2.17. This equation delineates the policy gradient of the actor concerning its parameters θ, utilizing the advantage function $A_t(s_t, a_t)$ over a sequence of states s_t and corresponding actions a_t.

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) A^{\pi_\theta}(s_t, a_t). \tag{2.17}$$

Proximal Policy Optimization

OpenAI advanced beyond the foundational REINFORCE algorithm by introducing Trust Region Policy Optimization (TRPO) in 2015 [Schulman et al., 2017]. Their enhancements were notably tailored for robotic applications, showcasing their algorithm in tasks such as swimming and walking gaits, along with demonstrating its effectiveness in various Atari games. Shortly thereafter, OpenAI introduced Proximal Policy Optimization (PPO) in 2017. PPO offered enhanced performance compared to TRPO while streamlining its methodology. It boasts greater sample efficiency than TRPO, demonstrating reduced sensitivity to hyperparameter tuning.

PPO is a type of Actor Critic algorithm which operates by progressively refining the policy through trial and refinement. Initially, the algorithm adopts a starting policy denoted by $\pi_\theta(a, s)$, parameterized by θ, and then gathers data by engaging with the environment and executing actions guided by the current policy. This data is then utilized to adjust the policy in a manner that maximizes the anticipated reward. Mathematically, this can be expressed as Equation 2.18:

$$\theta_{\text{new}} = \underset{\theta}{\text{argmax}} \sum_{t=0}^{T} \mathbb{E}\left[\min\left(r_t(\theta)\ A^{\pi_{\theta_{\text{old}}}}(s_t, a_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\right)\right], \tag{2.18}$$

where θ_{new} represents the updated policy parameters, $\pi_{\theta_{\text{old}}}$ is the old policy, $A^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$ denotes the advantage function, $r_t(\theta)$ is the ratio of probabilities between the new and old policies, as seen in Equation 2.19, and ϵ is a small threshold controlling the range of policy updates.

$$r_t(\theta) = \frac{\pi_\theta(a_t, s_t)}{\pi_{\theta_{\text{old}}}(a_t, s_t)}. \tag{2.19}$$

The fundamental concept underlying PPO is to maintain the policy changes within a small range to ensure stability. This is achieved through the implementation of a clipping mechanism, which restricts the extent of policy modifications. By comparing the old and new policies, this function limits the adjustments to the new policy if it deviates significantly from the old one. Consequently, this approach ensures that the updated policy remains closely aligned with its predecessor, thereby mitigating potential instability issues. Mathematically, the clipped objective function is defined as Equation 2.20:

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)A^{\pi_{\theta_{\text{old}}}}(s_t, a_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta_{\text{old}}}}(s_t, a_t)\right)\right], \tag{2.20}$$

where $L^{CLIP}(\theta)$ represents the clipped surrogate objective function, $r_t(\theta)$ is the probability ratio at time step t, and ϵ is the clipping parameter.

2.4 Generalization in Deep Reinforcement Learning

Generalization in DRL refers to the capacity of an agent to execute proficiently in situations resembling, though not exactly mirroring, those it encountered during training. The pivotal aspect of achieving generalization lies in the assurance that the acquired strategies can be seamlessly deployed in unfamiliar environments or environments subject to modifications over time. This extension of practical utility stands as a cornerstone for augmenting the adaptability and real-world efficacy of DRL algorithms, thereby enriching their versatility and applicability across diverse domains.

2.4.1 Zero-Shot Generalization

Zero-shot generalization (ZSG) [Kirk et al., 2021] in the context of DRL refers to the ability of a RL agent to perform effectively in tasks it has never encountered before, without any additional training on those specific tasks. The term *zero-shot* implies that the agent can generalize its learned policies to entirely new environments or scenarios, akin to transferring knowledge from known tasks to unseen tasks without any direct exposure or training on the new tasks.

Let's consider a scenario where the RL agent has been trained on a set of source tasks represented by a distribution $\mathcal{D}_{\text{source}}$. During training, the agent learns a policy parameterized by θ_{source} and extracts feature representations from the environment denoted by ϕ_{source}.

The goal of zero-shot generalization is to enable the agent to perform effectively on unseen target tasks represented by a distribution $\mathcal{D}_{\text{target}}$ without directly training on these tasks. To achieve this, the learned policy parameters θ_{source} and feature representations ϕ_{source} need to be adapted to the target tasks.

One approach to zero-shot generalization involves fine-tuning the learned policy parameters using a limited amount of data from the target tasks. Let $\mathcal{D}_{\text{fine-tune}}$ denote the distribution of data from the target tasks used for fine-tuning. The objective is to update the policy parameters θ_{source} to θ_{target} such that the agent's performance on the target tasks improves.

Mathematically, this fine-tuning process can be formulated as an optimization problem:

$$\theta_{\text{target}} = \arg\min_{\theta} \mathcal{L}_{\text{target}}(\theta), \tag{2.21}$$

where $\mathcal{L}_{\text{target}}(\theta)$ represents the loss function on the target tasks. This loss function can be defined based on the difference between the agent's predicted actions and the optimal actions in the target tasks.

Similarly, the feature representations ϕ_{source} can be adapted to ϕ_{target} using a similar optimization process. Let $\mathcal{L}_{\text{feat}}(\phi)$ denote the loss function for feature adaptation:

$$\phi_{\text{target}} = \arg\min_{\phi} \mathcal{L}_{\text{feat}}(\phi), \tag{2.22}$$

where $\mathcal{L}_{\text{feat}}(\phi)$ captures the discrepancy between the source and target feature distributions.

While zero-shot generalization offers benefits such as reduced training time and resource requirements, it also poses challenges. Ensuring that the learned representations are sufficiently generalizable and robust to variations encountered in unseen tasks is crucial. Additionally, the effectiveness of zero-shot generalization may vary depending on the similarity between the source and target tasks and the complexity of the environments involved.

2.5 OpenAI Gym Frameworks

OpenAI Gym [Towers et al., 2023] is a versatile toolkit designed for developing and comparing RL algorithms. It offers a wide array of environments that serve as benchmarks for RL research. These environments are categorized into different domains, see Figure 2.6, each providing unique challenges and complexities.

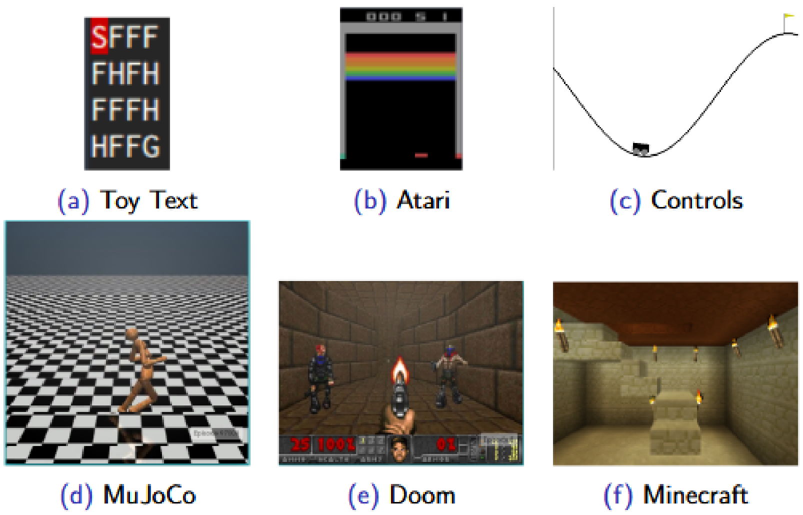

Figure 2.6: Example of different OpenAI Gym environments. Source:[OpenAI Gym Tutorial, 2018].

The diverse range of environments provided by OpenAI Gym enables comprehensive evaluation and comparison of RL algorithms. Researchers can test the robustness and generalization capabilities of their algorithms across different types of tasks and domains. For instance, this thesis investigates the generalization capabilities of DRL algorithms by experimenting with a variety of environments from the Classic Control and gymnasium robotics categories. Refer to Appendix A for a detailed description of the primary environments used.

2.5.1 Classic Control

Classic control environments are based on well-established problems in control theory and are often used as introductory tasks for RL. These environments typically involve simple, low-dimensional state and action spaces, making them ideal for learning fundamental RL concepts and testing basic algorithms. They include environments such as:

- **Acrobot-v1:** A two-link pendulum system where the goal is to swing the end of the lower link to a given height.

- **CartPole-v1:** A balancing task where a pole is attached to a moving cart, and the objective is to prevent the pole from falling over.

- **MountainCarContinuous-v0:** A continuous control version of the Mountain Car task, where a car must ascend a steep hill using continuous force application.
- **Pendulum-v1:** A task involving a pendulum that needs to be swung up to an upright position.

2.5.2 Gymnasium Robotics Environments

Utilizing the MuJoCo physics engine, see Section 2.6, OpenAI Gym environments offer fast and accurate simulation capabilities. An example of useful environments are the Fetch environments [Plappert et al., 2018] from Gymnasium Robotics [de Lazcano et al., 2023], specifically based on the *7-DoF Fetch* robotics arm with a two-fingered parallel gripper.

The Fetch environments, Figure 2.7, encompass a variety of tasks tailored to different robotic manipulation challenges:

- **FetchReach-v2:** In this task, Fetch is tasked with precisely moving its end-effector to a designated goal position.
- **FetchPush-v2:** Here, Fetch is required to move a box by pushing it until it reaches a specified goal position.
- **FetchSlide-v2:** In this scenario, Fetch must strike a puck across a lengthy table, ensuring it slides and comes to rest at the desired goal location.
- **FetchPickAndPlace-v2:** One of the core tasks, Fetch is challenged to grasp a box from a table using its gripper and transport it to a specified goal position above the table.

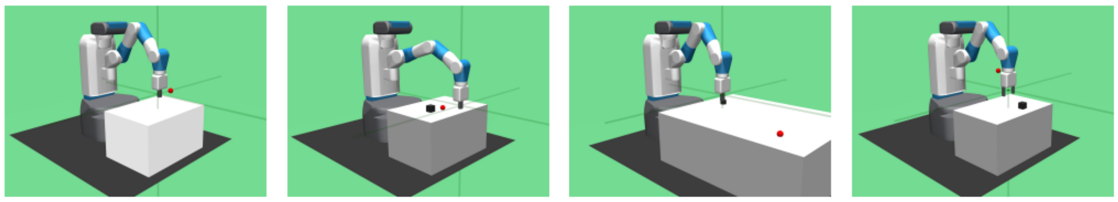

Figure 2.7: Fetch environments: *FetchReach*, *FetchPush*, *FetchSlide*, and *FetchPickAndPlace*, based on the *7-DoF Fetch* robotics arm. Source:[Plappert et al., 2018].

2.6 Physics Engine Simulator (MuJoCo)

The MuJoCo physics engine simulator[Todorov et al., 2012] serves as a crucial tool for training robotic systems and will be highly beneficial for applying the different environments discussed in this thesis. Renowned for its accuracy and speed, MuJoCo facilitates research in robotics, biomechanics, graphics, and animation, Figure 2.8. Its features include precise simulation in generalized coordinates, robust inverse dynamics, and support for various actuators and solvers. With an intuitive XML model format and interactive 3D visualization, MuJoCo offers efficient simulation and optimization for robotic applications, making it an ideal environment for testing and implementing RL and DRL algorithms.

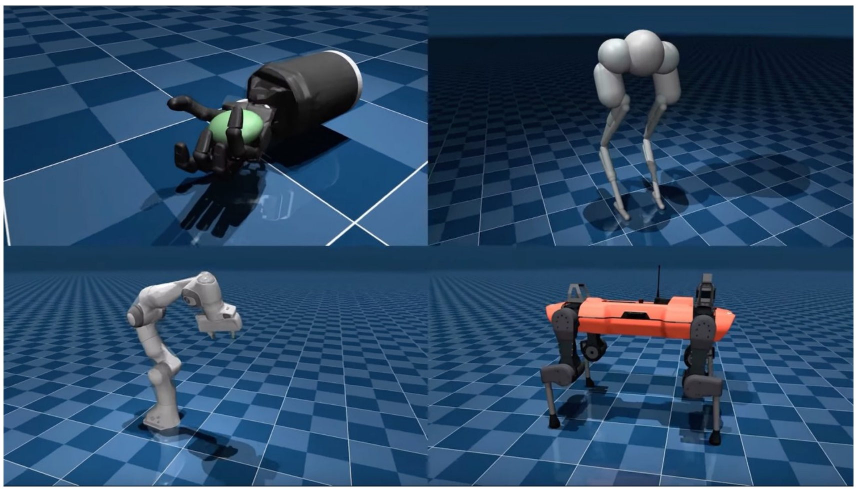

Figure 2.8: Collection of Simulated Robots in MuJoCo. Source:[Todorov et al., 2012].

Chapter 3

METHODOLOGY

The methodology explains how the construction of the experimental framework has been, beginning with the selection of programming languages and various tools essential for enabling the agent to interact with the different control task and Pick and Place environments. This step ensures that the agent can effectively undergo training and learn from its experiences within the chosen environment.

Once the necessary languages and tools are identified, the focus shifts to understanding the architecture of the DRL agent. This involves delineating its structure, specifying training procedures, and fine-tuning hyperparameters to optimize learning efficiency. The objective throughout this process is to equip the agent with the capabilities required to acquire and refine its policies effectively.

Following the establishment of the training setup, attention turns to investigating the generalization capabilities of the trained agent. This phase involves engaging in controlled experimentation to assess the agent's adaptability across diverse similar scenarios. The aim is to evaluate its ability to generalize learned policies to novel environments, providing insights into the agent's robustness and versatility.

Repository and Resources

All the implementation, environments, results, and resources used in this research are available in a GitHub repository. You can access it through the following link: https://github.com/ialexmp/DRL-Generalization.

The repository contains detailed information about the requirements, setup instructions, and steps to replicate the experiments. This resource is intended to facilitate further research and enable other researchers to build upon the work presented in this thesis.

3.1 Environment

This section encompasses a range of essential components, including the programming language chosen for development, the different environments sourced from OpenAI Gymnasium and MuJoCo, and the differents components of the MDP.

3.1.1 Programming Language

Python has been chosen as the primary programming language due to its versatility, user-friendly syntax, and extensive library ecosystem, which are well-suited for RL, ANN and DRL development. Python's simplicity and readability make it accessible to developers of all skill levels and facilitate collaboration throughout the research process. Key libraries such as NumPy, PyTorch, and Gymnasium have played crucial roles in enabling the implementation of various algorithms and environments for this project.

Despite the undeniable performance and efficiency advantages offered by languages like C and C++, Python's high-level nature and rich library support make it the preferred choice for rapid prototyping and development in the AI domain. While C and C++ may excel in raw performance, their complex syntax and limited library ecosystem can impede the agility and productivity required for successful AI and RL projects.

3.1.2 Exploring Gymnasium Environments

Conducting separate experiments with distinct environments for training DRL agents, each utilizing the PPO algorithm, see Section 2.3.3, serves as a crucial approach for gaining insights into the adaptability and efficacy of DRL methodologies across diverse tasks. These isolated experiments allow for a focused examination of how each agent learns and performs within its designated environment. Through this approach, a comprehensive understanding of the agents' learning dynamics, strengths, and limitations can be acquired. In this thesis, the following environments sourced from the Gymnasium and Gymnasium Robotics suite, see Section 2.5, will be tested.

- **Acrobot**: This environment involves a two-link pendulum system where the goal is to swing the free end of the pendulum above a designated height. The discrete action space consists of applying torque to the joint between the two pendulum links. The complexity lies in coordinating the movement of the two links to generate enough momentum to swing the free end above the required height while minimizing energy consumption. For detailed information about the environment settings, refer to Appendix A.1.

- **CartPole**: The task is to balance a pole attached to a cart by applying forces to the cart. The discrete action space usually includes applying a force to either move the cart left or right. The challenge is to balance the pole for as long as possible without letting it fall beyond a certain threshold angle. For detailed information about the environment settings, see Appendix A.2.
- **Mountain Car Continuous**: This environment consists of a car placed in a valley with the goal of accelerating it to reach the top of a hill. The action space involves applying continuous acceleration to the car, allowing it to traverse the uneven terrain. The complexity lies in navigating the car up the steep hill while overcoming the effects of gravity and inertia, requiring precise control to reach the goal. For detailed information about the environment settings, see Appendix A.3.
- **Pendulum**: The task is to swing a pendulum upright by applying torque to the free end. The action space typically involves applying continuous torque to the pendulum, allowing it to swing back and forth. The complexity arises from the need to control the torque applied to stabilize the pendulum in an upright position while minimizing oscillations. For detailed information about the environment settings, see Appendix A.4.
- **Custom Pick and Place**: This customized environment, derived from the Gymnasium Robotics toolkit, involves a robot manipulator tasked with moving a block to a target position on a table surface. The action space encompasses controlling the joints of the robot manipulator to perform various movements such as reaching, grasping, lifting, and placing the block. The complexity lies in coordinating the movements of multiple joints to execute precise actions required to manipulate the block and navigate obstacles on the table surface. For detailed information about the environment settings, see Appendix A.5.

3.1.3 High Performance Computing Cluster

To expedite the training process of the DRL Agent in the Custom Pick and Place environment, the High-Performance Computing (HPC) cluster, SNOW, provided by Universitat Pompeu Fabra, was utilized. The robust infrastructure of SNOW played a pivotal role in achieving timely results. Leveraging the cluster's computational power enabled the execution of large-scale simulations, parameter optimization, and experimentation using the PPO algorithm with exceptional efficiency and speed, surpassing the capabilities of standard workstations or servers. Consequently, this facilitated quicker convergence and the refinement of a more resilient and efficient PPO algorithm.

3.2 Deep Reinforcement Learning Agent

Navigating Continuous Action Spaces in RL

In the realm of RL, see Section 2.1, an action space defines the feasible actions an agent can execute at any given time step. While discrete action spaces present agents with a predefined set of choices, continuous action spaces pose a challenge due to their potentially infinite range of values. Even within a defined range of -1.0 to 1.0 for each dimension, the potential values are effectively infinite owing to the continuous nature of decimal increments.

One common approach involves attempting to adapt DQN, see Section 2.3.2 to handle continuous action spaces by directly outputting values for each action or defining probability distributions for actions parameters, facilitating built-in exploration within neural network architectures. However, algorithms based on the Actor-Critic framework, which includes methods like PPO, see Section 2.5 often outperform DQN in scenarios with continuous action spaces. These methods focus on policy optimization rather than Q-value estimation, leading to more effective learning strategies and improved performance in such environments.

Thesis Orientation:

Recognizing the inherent complexity of continuous action spaces, and in alignment with the requirements of the thesis, it has been opted to focus on DRL algorithms that are compatible with continuous actions, such as the Actor-Critic algorithms. This strategic decision allows for precise control over the robot's end effector movements in three-dimensional space and enables manipulation of its gripper function for tasks such as grasping and object manipulation within the environment for the Pick and Place environment. Similarly, for the different continuous control tasks investigated in this research, this strategic approach facilitates nuanced control over the agents' actions, ensuring effective adaptation to the continuous state spaces inherent in tasks like balancing a pendulum or navigating a mountain car.

3.2.1 Agent Implementation

Proximal Policy Optimization (PPO) stands as a cornerstone algorithm within the Actor-Critic framework, see Section 2.3.3, specifically tailored to address the challenges presented by continuous action spaces in RL task by optimizing the policy function to maximize expected rewards.

Actor and Critic Networks

The actor network and the critic network are crafted to facilitate policy learning and value estimation, respectively.

- **Actor Neural Network**: Implemented with fully connected layers (FC layers) to process input observations and produce action distributions. Specifically, the *forward()* method of the actor network defines the flow of information through the network. It applies rectified linear unit (ReLU) activation functions, Equation 3.1, to the output of each FC layer, culminating in the final layer where the sigmoid activation function $\sigma(x)$, Equation 3.2, is used to produce action distributions, represented in Equation 3.3 and Equation 3.4 respectively. Furthermore, depending on the nature of the action space, either a multivariate normal distribution is instantiated when actions are continuous, enabling the generation of stochastic actions, or a categorical distribution is utilized when actions are discrete, facilitating probabilistic action selection

$$\text{ReLU} = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0, \end{cases} \tag{3.1}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.2}$$

- **Critic Neural Network**: Similarly, it is constructed with FC layers to estimate the value of state-action pairs. In the *forward()* method of the critic network, the input observations pass through FC layers with ReLU activation functions before arriving at the output layer, which produces a single value representing the estimated value (or advantage) of the given state-action pair.

$$y = \text{ReLU}\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{3.3}$$

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{3.4}$$

where y represents the output of the fully connected layer, σ and ReLU denotes the activation function and $\sum_{i=1}^{n} w_i x_i + b$ is the weighted sum of inputs plus bias, which is passed through the activation function to produce action distributions.

Training Procedure

The training procedure manages the optimization of the actor and critic networks using the PPO algorithm. It unfolds as follows:

- **Rollout Generation:** Each training iteration triggers the rollout method to engage with the environment, generating episodes. These episodes encompass sequences of observations, actions, log probabilities, and rewards. Rollouts persist until the specified number of timesteps per batch is exhausted.

- **Advantage Estimation:** Following rollout generation, advantages are computed via the critic network. Advantages capture the delta between observed rewards and the estimated values of state-action pairs. This step is pivotal for informing policy updates and refining the actor network's performance.

- **Policy Update:** The actor network's policy parameters undergo updates to maximize cumulative rewards while stabilizing training dynamics. This entails minimizing the PPO objective function, Equation 2.17, which entails computing the ratio of probabilities, Equation 2.19, under the current and prior policies and applying a clipping mechanism to bound policy updates.

- **Value Function Update:** Concurrently, the critic network's parameters evolve to minimize the mean squared error (MSE), Equation 3.5, between predicted and actual rewards. This enhances the fidelity of value estimations, furnishing more informative feedback to the actor network and fostering training stability.

  $$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{R}_i - R_i)^2 \tag{3.5}$$

  Where N is the total number of samples, $\hat{R}_i$ represents the predicted reward and R_i is the actual reward in sample s_i

- **Training Loop:** The training loop iteratively orchestrates multiple training cycles. Each cycle encompasses the generation of rollouts, advantage estimation, policy and value function updates, and subsequent iterations until convergence. The number of training cycles per batch can be tailored to accommodate computational constraints and convergence criteria.

- **Status Tracking and Logging:** Throughout the training procedure, real-time status updates are logged. These updates illuminate the ongoing action (e.g., rollout, training cycle), the cumulative rewards amassed, and the losses incurred during training.

Through meticulous orchestration of these steps, the training procedure systematically refines the actor and critic networks, equipping RL agents with enhanced capabilities to learn optimal policies and value functions for Pick and Place robotics tasks.

Hyperparameters Fine-Tuning

Fine-tuning of hyperparameters is indispensable for optimizing the performance and convergence of the PPO algorithm. These hyperparameters act as knobs that govern various aspects of the learning process, dictating how the agent interacts with the environment, updates its policy, and estimates the value function. By adjusting and refining these hyperparameters, we can tailor the learning dynamics to suit the specific characteristics of the task at hand, leading to more efficient training and improved performance. The following hyperparameters are adjusted and refined during the training process:

- **Timesteps per Batch**: The number of timesteps considered in each batch during training.
- **Max Timesteps per Episode**: The maximum number of timesteps allowed for each episode before termination.
- **Gamma** (γ): Discount factor determining the importance of future rewards in the agent's decision-making process.
- **Epsilon** (ϵ): Clipping parameter used in the PPO objective function to limit policy updates.
- **Alpha** (α): Learning rate, controlling the magnitude of parameter updates during optimization.
- **Training Cycles per Batch**: The number of training cycles executed per batch, each encompassing multiple iterations of rollout generation, advantage estimation, policy updates, and value function updates.

Model and Plot Saving

At regular intervals, the models, state, and additional data are automatically saved to a designated directory within the ”Results” folder. This structured organization ensures easy access and management of experimental artifacts. The saved contents include the actor and critic models, their respective losses, rewards logs, hyperparameters, and generated plots depicting the training progress. These saved artifacts serve as invaluable resources for further analysis and ensure the reproducibility of experimental results.

3.3 Training Protocol and Experiments

To optimize the performance of the DRL agent employing the PPO algorithm across diverse environments, a series of experiments were devised, systematically tuning hyperparameters to ensure the development of highly effective models. Hyperparameter configuration plays a pivotal role in shaping the agent's learning dynamics, convergence speed, and overall performance. Therefore, multiple parameters were adjusted and tested to optimize the learning process. This included exploring different values for timesteps per batch, and maximum timesteps per episode to evaluate their impact on learning dynamics and convergence. Additionally, fixed values were set for gamma (γ) and epsilon (ϵ) to prioritize long-term rewards and regulate policy updates, respectively. The learning rate, alpha (α), was varied to assess its influence on the speed and stability of learning. Finally, the number of training cycles per batch was examined to understand its effects on training efficiency and convergence. Through systematic experimentation and adjustment of these hyperparameters, the DRL agent was trained to effectively navigate and learn from diverse control task environments.

For the control task environments (Acrobot, Mountain Car Continuous, CartPole and Pendulum), which share similar basic complexities, experiments were designed with comparable hyperparameter ranges to ensure consistency across the training process.

In the case of the Pick and Place scenario, an extensive series of experiments were conducted due to the high complexity of this environment, making it impractical to execute solely on a computer CPU. Therefore, all experiments related to this environment were carried out on a GPU cluster provided by Universitat Pompeu Fabra, as detailed in Section 3.1.3.

A summary of the most notable hyperparameter values used in these experiments is presented in Table 3.1 for better visualization.

Hyperparameter	**Control Task**	**Pick and Place**
Timesteps per Batch (TB)	500, 750, 1000	10K, 50K, 100K, 150K, 200K
Max Timesteps per Episode (MTE)	750, 1000	1K, 5K, 10K, 15K
Gamma (γ)	0.99	0.99
Epsilon (ϵ)	0.1, 0.2	0.1, 0.2
Alpha (α)	0.01, 0.001, 0.0001, 0.00001	0.01, 0.001, 0.0001, 0.00001
Training Cycles per Batch (TCB)	5, 10	5, 10

Table 3.1: Hyperparameters Used for Training.

3.4 Generalization Protocol and Experiments

To assess the generalization capabilities of the trained agents, see Section 3.3, adjustments were made to specific parameters within their original training environments. These modifications were designed to replicate realistic scenarios where environmental conditions vary, necessitating the adaptation of learned policies by the agents. Table 3.2 provides an overview of these environment modifications, detailing the modified features (MF), their trained values (TV), the values tested (VT) during modification (each tested over 200 episodes), the step size between tested values (SV), and the maximum number of steps required to achieve the goal (MS). For a comprehensive description of each custom environment modification, refer to Appendix B.

TE	MF	TV	VT	SV	MS
Acrobot-v1	Center of Mass	0.5	[0.1, 3.0]	0.2	400
	Link Length	1.0	[0.25, 4.0]	0.25	400
	Link Mass	1.0	[0.25, 4.5]	0.25	400
CartPole-v1	Force Magnitude	10	[5.0, 15.0]	1.0	10000
	Gravity	9.8	[5.0, 15.0]	1.0	10000
	Pole Length	0.5	[0.2, 0.8]	0.1	10000
	Pole Mass	0.1	[0.1, 0.6]	0.05	10000
MountainCar Continuous-v0	Car Friction	0.0025	[0.001, 0.005]	0.0005	300
	Initial Position	[-0.04, -0-06]	[-0.6, 0.3]	0.1	300
	Car Power	0.0015	[0.0005, 0.003]	0.0005	300
Pendulum-v1	Delta Time	0.05	[0.01, 0.15]	0.01	8000
	Gravity	9.8	[5, 15]	1	8000
	Pendulum Length	1.0	[0.5, 1.6]	0.1	8000
	Pendulum Mass	1.0	[0.25, 2.50]	2.5	8000

Table 3.2: Summary of modifications made to test generalization. MF = Modified Feature of the trained environment, TV = Trained Value, VT = Values Tested with the modified feature, SV = Step size between the values tested, MS = Maximum number of Steps to reach the goal.

Chapter 4

EXPERIMENT EVALUATION

This chapter presents the outcomes of diverse experiments conducted with the DRL Agent, see Section 3.2, across a range of control tasks and Pick and Place environments, see Section 3.1.2. Subsequently, the top-performing models are subjected to testing in slightly modified environments to assess their adaptability to novel conditions. This analysis aims to illuminate the agents' capacity to adapt to new environments and provide insights into their generalization capabilities.

4.1 Training Results

This section outlines the primary findings from the training experiments detailed in Section 3.3. These experiments aimed to evaluate the DRL Agent's performance across different control tasks and Pick and Place environments. Additional detailed results can be found in the appendix B.1.

4.1.1 Classic Control Tasks

All experiments concerning the Classic Control environment followed a standardized procedure. Initially, preliminary experimentation identified the sensitivity of specific hyperparameters, notably the learning rate (α). Subsequent experiments systematically explored these critical parameters consisting of a fixed value of 1 million timesteps while varying the α value from 0.01 to 0.00001 to ascertain the optimal value.

One example of this procedure is the experiment with the Pendulum environment, where the results, illustrated in Figure 4.1, indicate that while an $\alpha = 0.01$ showed rapid initial improvement, it resulted in high variability and instability, which are important indicators of both training stability and the model's potential for generalization. High variability in rewards can signify that the agent is still exploring and has not converged to a stable policy. This exploration might lead to fluctuating performance, which is undesirable for reliable training and even more critical when evaluating the model's ability to generalize to new environments.

A rate of 0.001 achieved the highest rewards, but the blue line representing an α of 0.0001 demonstrated a stable and steady improvement, making it a promising choice for longer-term performance.

Finally, while an α of 0.00001 ensures stability, the slow progress suggests it may not be practical even with many more timesteps. Therefore, for the final experiment, $\alpha = 0.0001$ is selected due to its balance of stability and consistent improvement, though $\alpha = 0.001$ could also be a good option for potentially higher rewards and stability.

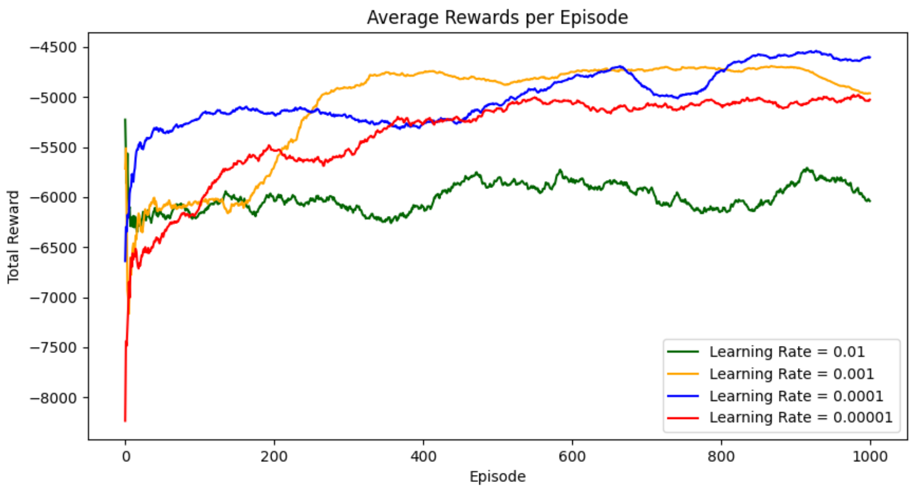

Figure 4.1: Pendulum: Average Rewards Across Various α Values.

Subsequently, with the learning rate fixed, further experiments were conducted to ascertain the number of timesteps needed to converge. The Figure 4.2 illustrates the total rewards per episode for the Pendulum-v1 environment over 10 million timesteps, corresponding to 10,000 episodes. Initially, there is a steep increase in total rewards, indicating rapid learning and adaptation by the agent as it quickly learns basic control tasks to keep the pendulum upright. However, this phase is marked by high variability, reflecting the agent's exploration of various actions. In the intermediate phase, spanning approximately 1000 to 5000 episodes, the total rewards continue to increase but at a slower rate, suggesting more refined learning. Variability decreases, indicating that the agent's actions are becoming more consistent. Around 5000 episodes, performance plateaus with total rewards oscillating around a stable average, indicating the agent has reached a level where further significant improvements are minimal. The average rewards line, smoothed with a window of 100 episodes, highlights this overall trend of initial improvement followed by stabilization. The reduced width of the shaded area around the average rewards line shows decreased variability, signifying more predictable and reliable performance by the agent.

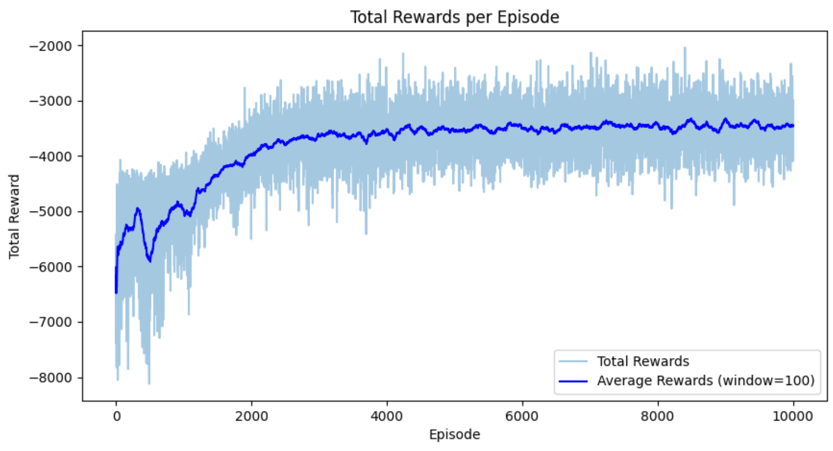

Figure 4.2: Pendulum: Total Training Rewards per Episode.

Overall, the agent demonstrates successful learning, achieving consistent performance and a robust policy, validating the chosen learning rate and training duration. Based on these observations, the final number of timesteps for the final model will be set to 5 million timesteps (5000 episodes). This duration is justified as it allows the agent to reach and maintain stable performance, avoiding unnecessary computational resources while still achieving optimal learning and policy refinement. After conducting this series of experiments for each of the Control task environments, all results explained in detail in the Appendix B, these were synthesized and analyzed to determine the most effective combination of hyperparameters. Table 4.1 illustrates the final hyperparameters for the best-performing models identified in each experiment.

Best Training Results

Environment	**T**	**TB**	**MTE**	α	γ	ϵ	**TCB**	**Time**
Acrobot-v1	1M	1000	1000	0.00001	0.99	0.2	5	0:11:33
CartPole-v1	2M	1000	1000	0.001	0.99	0.2	5	0:20:10
MountainCar-Continuous-v0	3M	1000	1000	0.001	0.99	0.2	5	0:32:05
Pendulum-v1	5M	1000	1000	0.0001	0.99	0.2	5	1:04:39

Table 4.1: Best Training for different environments. T = Number of Timesteps, TB = Times per Batch, MTE = Maximum Timesteps per Episode, TCB = Training Cycles per Batch.

4.1.2 Custom Pick and Place Tasks

As discussed in Appendix B.2, the original environment presented significant challenges due to its high-dimensional state and action spaces and complex physical interactions. To address these difficulties, several features were simplified, resulting in a custom Pick and Place environment. Despite these simplifications, the environment remained challenging, requiring precise coordination of multiple joints and effective navigation around obstacles.

Over the course of more than a month, numerous long-running experiments were conducted on the Universitat Pompeu Fabra Cluster, see Section 3.1.3. Despite these extensive efforts, the agent failed to learn the Pick and Place task effectively.

One of the most significant experiments is illustrated in Figure 4.3, which shows the total rewards per episode over approximately 14 days of training and 200 million timesteps. The total rewards per episode exhibit substantial fluctuations, indicating that the model struggled to learn effectively. The average rewards, represented by the blue line, remained relatively low and stable throughout the experiment, suggesting limited improvement in performance despite extensive training.

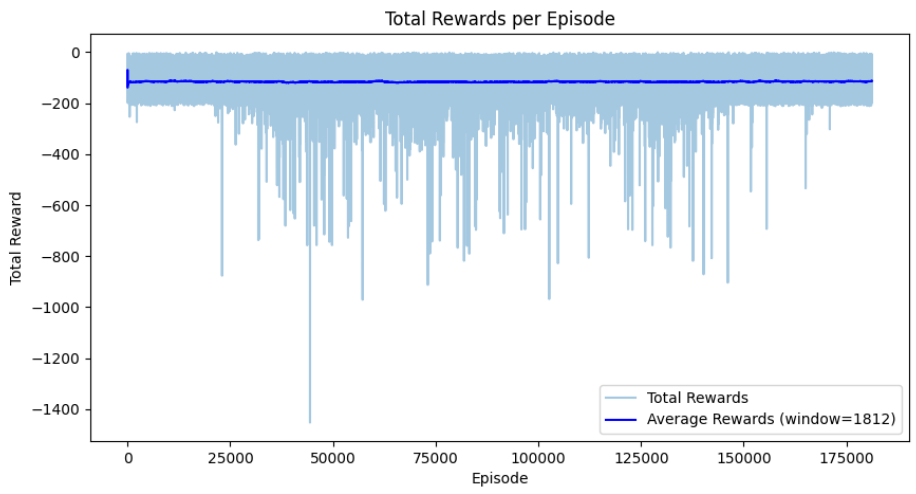

Figure 4.3: Pick and Place: Total Training Rewards per Episode.

These results indicate that even with significant computational resources and extensive training, the model was unable to learn an effective policy for the custom Pick and Place environment. This underscores the challenges of training DRL agents in complex environments, where task intricacies and high-dimensional action spaces can hinder learning and generalization.

4.2 Generalizations Results

This section highlights the most significant findings from the generalization experiments detailed in Section 3.4. These experiments were designed to evaluate the adaptability of the trained agents to changes in critical environmental parameters. Each of the following subsections details the specific modifications implemented in the environments and the corresponding performance results. The detailed results of all experiments are available in the Appendix B.

4.2.1 Modified Acrobot: Center of Mass Positions

In the Acrobot environment, the center of mass (CoM) positions of the links were altered to evaluate the agent's robustness to changes in physical parameters, limiting each trial to a maximum of 400 steps to achieve the goal. This study yielded two distinct scenarios: the blue line illustrates the average rewards obtained when adjusting the CoM of Link 1 while keeping Link 2 at its trained position, while the orange line represents the average rewards obtained when adjusting the CoM of Link 2 while keeping Link 1 at its trained position.

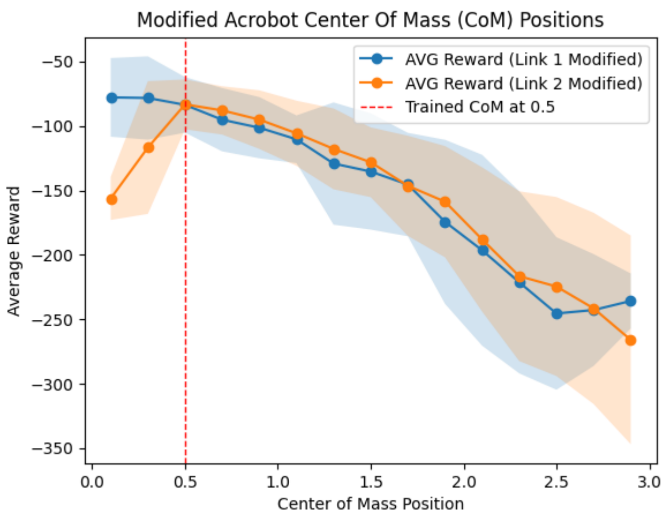

Figure 4.4: Acrobot: Average Rewards with modified Center of Mass Positions.

Figure 4.4 illustrates that as the CoM position of Link 1 increases from 0.0 to 3.0, there is a significant decrease in average rewards. This trend suggests a decline in the agent's performance, which initially remains relatively stable but deteriorates as the CoM moves further from the trained position (0.5), indicated by a red dashed line. A similar pattern is observed for modifications to Link 2, where performance begins to decline sharply after the initial CoM position, highlighting the agent's sensitivity to changes in the physical properties of both links.

The shaded areas around the lines represent the variability in the results. The increasing variability as the CoM position moves further from the trained position indicates greater uncertainty in the agent's performance under these altered conditions. Despite this variability, the overall trend shows a clear decline in performance.

The observed performance drop can be attributed to the fact that the agent's policy, developed during training, is optimized for the specific dynamics of the original CoM configuration. Changes in the CoM positions alter the dynamics of the system, affecting the state transitions and the control torques required to achieve the desired movements. These changes make the previously learned policy less effective in adapting to the new dynamics, resulting in reduced performance.

However, despite these challenges and considering the reward structure of the environment, the agent is still able to reach the goal within less than 400 steps after these modifications. This resilience suggests that while the agent's performance degrades with significant changes to the CoM, it retains a degree of robustness that allows it to adapt and achieve its objective, albeit with reduced efficiency.

4.2.2 Modified CartPole: Force Magnitude

In this set of experiments, the generalization capability of a DRL agent trained in the CartPole environment was tested by modifying the force magnitude applied to the cart. The objective was to simulate an environment anomaly that affects the cart's ability to exert force, thereby evaluating the agent's robustness to such changes.

In the first experiment, the force magnitude for the action that pushes the cart to the left (action = 0) was altered, while the force magnitude for the right action (action = 1) was kept at the trained value. This setup aimed to simulate a scenario where an anomaly affects the cart's capacity to push leftward with varying intensities. In the second experiment, the setup was reversed by modifying the force magnitude for the right action (action = 1), while maintaining the trained force magnitude for the left action (action = 0). This scenario tested the agent's performance when the cart's ability to push rightward was compromised.

Both experiments were conducted over a maximum of 10,000 steps to evaluate the agent's performance under different force magnitudes. Figure 4.5 presents the average rewards achieved by the agent when subjected to different force magnitudes.

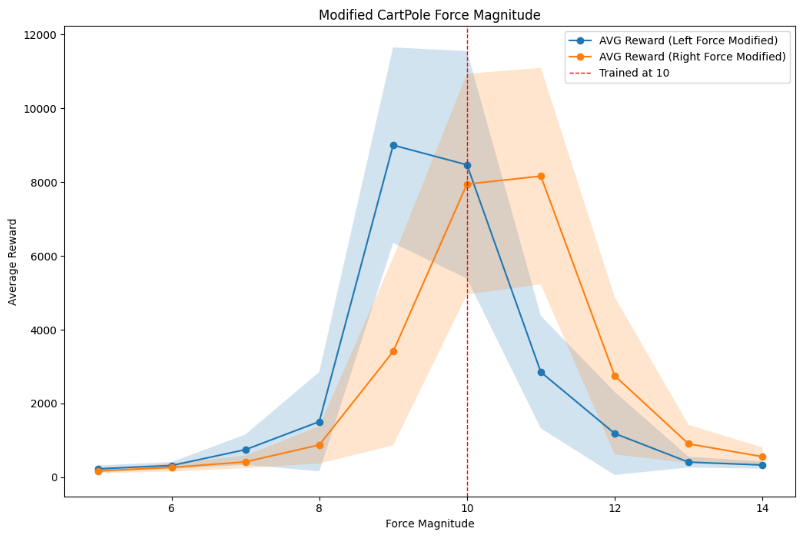

Figure 4.5: CartPole: Average Rewards with modified Force Magnitude.

When the left force magnitude was increased or decreased from the trained value of 10 (dashed red line), the agent's performance varied significantly. The average reward peaked around the trained force magnitude, indicating optimal performance at or near the trained value. Significant deviations from the trained magnitude resulted in a steep decline in average reward, highlighting the agent's sensitivity to changes in force magnitude, underscoring the agent's inability to generalize to these modifications.

Similarly, changes in the right force magnitude also led to variations in performance. The agent achieved the highest average reward near the trained force magnitude of 10. Deviations from this value resulted in a noticeable drop in performance, although the decline was slightly less steep compared to the left force modification.

Both modifications demonstrate the agent's reliance on the trained force magnitude for optimal performance. The blue and orange shaded areas represent the confidence intervals, showing variability in performance across different trials. The peak performance near the trained force magnitude suggests that the agent's policy is finely tuned to the specific dynamics experienced during training.

The significant drop in performance away from the trained force magnitude highlights the challenges in generalizing to new dynamics. However, the presence of a peak around the trained value suggests that while the agent is finely tuned to the training conditions, there is a narrow range within which it can still perform optimally

4.2.3 Modified Mountain Car Continuous : Friction

In the Mountain Car Continuous environment, experiments were conducted to evaluate the agent's ability to generalize by modifying the friction coefficient of the car. The goal was to assess how variations in friction affect the agent's performance and determine the robustness of the learned policy, with a maximum of 300 steps allowed. The reward structure in this environment includes a negative reward at each step, penalizing the agent for taking actions of large magnitude. Additionally, if the mountain car reaches the goal, a positive reward of +100 is added to the negative reward for that step.

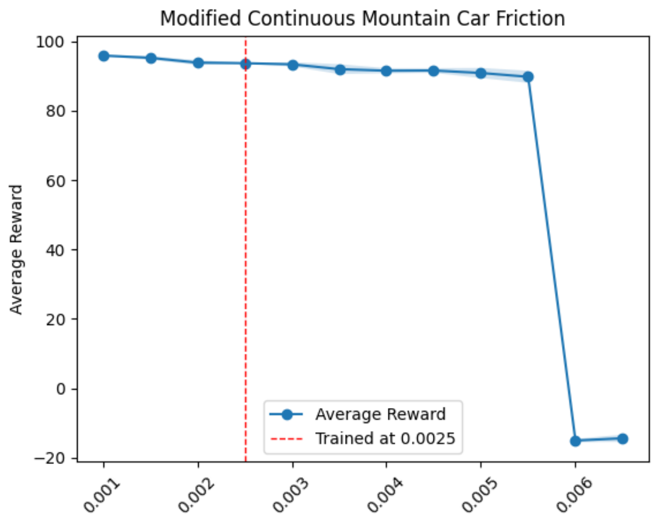

Figure 4.6: Mountain Car Continuous: Average Rewards with modified Friction (300 steps).

The agent was initially trained with a friction coefficient of 0.0025, indicated by the red dashed line in the Figure 4.6. At this friction coefficient, the agent achieves an average reward close to 100, which serves as a baseline for comparing performance at other friction levels.

For friction coefficients below the trained value (0.0010 to 0.0020), the agent maintains a high average reward, close to 100, indicating that it successfully reaches the goal in these conditions. This suggests that the agent can adapt well to slightly lower friction settings.

As the friction coefficient increases beyond the trained value (0.0030 to 0.0060), the agent's performance remains relatively stable initially, with average rewards still close to 100, indicating successful goal-reaching behavior. However, at a friction coefficient of approximately 0.0050, the agent's performance drops sharply, with average rewards plummeting to negative values. This significant decline suggests that the agent struggles to reach the goal under these higher friction conditions, reflecting its limited generalization capability to environments with substantially different friction levels.

The observed failure to reach the goal at friction coefficients equal to or greater than 0.006 is likely due to the car's power limitations. The increased friction appears to exceed the car's power capacity, preventing it from overcoming the higher resistance and successfully reaching the goal. To verify this hypothesis, longer experiments with a maximum of 5000 steps were conducted. Figure 4.7 demonstrates that even with an extended number of steps, the model still fails to reach the goal, corroborating the intuition that the car's power is insufficient to adapt to the higher friction levels.

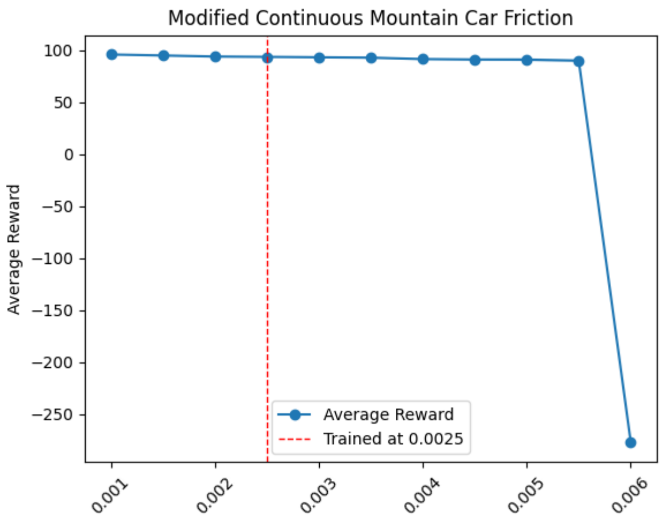

Figure 4.7: Mountain Car Continuous: Average Rewards with modified Friction (5000 steps).

The low variability in performance, indicated by the narrow shaded area around the average reward line, suggests that the agent's behavior is consistent across different runs. This consistency further supports the conclusion that high friction levels significantly impair the car's ability to complete the task, highlighting a fundamental limitation in the agent's power capacity.

4.2.4 Modified Pendulum: Length

In the Pendulum environment, experiments were conducted to test the agent's generalization capability by modifying the pendulum length. The objective was to analyze how changes in pendulum length affect the agent's performance, thereby evaluating the robustness of the learned policy. The agent was initially trained with a pendulum length of 1.0 (dashed red line), and its performance was observed under these modified conditions with a maximum of 8000 steps to reach the goal.

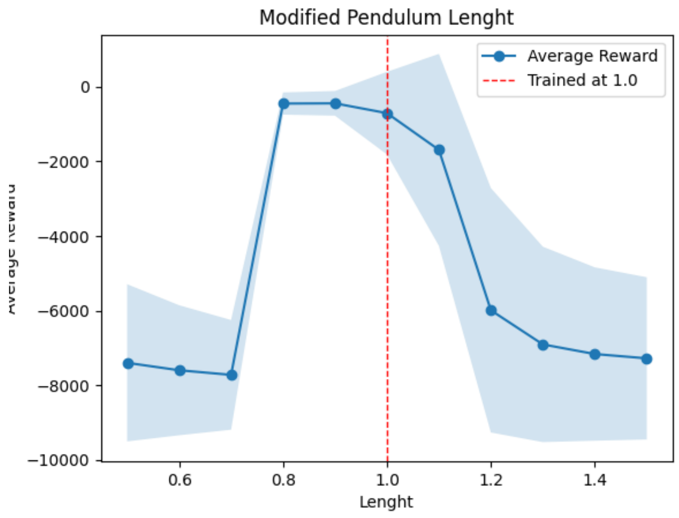

Figure 4.8: Pendulum: Average Rewards with modified Length.

Figure 4.8 illustrates that the agent's performance is sensitive to changes in the pendulum length. The rewards reflect this sensitivity, given that each step in the environment incurs a reward of -1, leading to a minimum possible reward of -8000. Initially, for pendulum lengths shorter than 1.0, the average reward was around this minimum, suggesting that the agent performed poorly and took the maximum number of steps without achieving the goal. However, as the length approached 1.0, the agent's performance improved significantly. At a length of approximately 0.9, the average reward improved, indicating that the agent could achieve the goal in fewer steps.

At the exact training length of 1.0, the agent performed optimally, achieving the highest average reward with a small number of steps. This peak in performance highlights the agent's effective learning and adaptation to the specific dynamics of the training environment. Near this length, the agent maintained good performance, suggesting a reasonable degree of generalization to similar but slightly varied conditions.

As the pendulum length increased beyond 1.0, the agent's performance declined noticeably. The average reward dropped, reaching values significantly lower as the length increased further from the training length. At lengths around 1.4, the average reward approached -8000 again, indicating that the agent struggled to adapt and took the maximum number of steps without achieving the goal. This significant decrease in performance with increased length suggests that the agent's learned policy is not robust to substantial changes in the environment's dynamics.

Overall, these results underscore the agent's limited generalization capability. While it can adapt to some extent to changes in pendulum length, its performance peaks near the trained length and deteriorates rapidly with larger deviations.

Chapter 5

DISCUSSIONS AND CONCLUSIONS

The Discussions and Conclusions chapter provides a thorough analysis of DRL experiments, emphasizing the assessment of generalization capabilities across tasks of diverse complexity. Initially hypothesized was the potential synergy between RL and deep learning (DL), leveraging RL's strengths in sequential decision-making and DL's capabilities in robust function approximation. This chapter critically examines how well DRL agents adapted and generalized their learned policies across the varying environments tested in this research, shedding light on both successes and challenges encountered in practical applications.

5.1 Discussions

The discussion chapter delves into the analysis of results derived from experiments in DRL with a focus on generalization capabilities. Two types of environments were employed: control tasks, which have relatively low complexity, and the Pick and Place scenario, characterized by higher complexity. The primary objective was to examine how well the DRL agents could learn and generalize across these varying tasks.

In the control tasks, environments such as Acrobot, CartPole, Continuous-MountainCar, and Pendulum were utilized. These tasks presented simpler, well-defined challenges. The DRL agents, trained using the PPO algorithm, demonstrated the ability to effectively learn and adapt within these environments. The hyperparameter tuning was a critical aspect of these experiments, with variables such as timesteps per batch, maximum timesteps per episode, γ, ϵ, and α being meticulously adjusted to optimize performance.

The success in these control tasks indicated that the DRL agents were capable of mastering the dynamics of relatively straightforward environments. The training results for these tasks, as documented in the best training outcomes, showed that the agents could achieve high performance with consistent rewards over time. This ability to optimize performance through systematic hyperparameter tuning underscores the importance of such configurations in the DRL process .

However, the Pick and Place task, representing a more complex scenario, posed significant challenges. Despite extensive efforts, the DRL agent struggled to learn this task effectively. This environment, characterized by high-dimensional state and action spaces and intricate physical interactions, proved to be a formidable challenge for the PPO algorithm. Multiple long-running experiments conducted over more than a month on the Universitat Pompeu Fabra Cluster highlighted the difficulties faced by the agent. Even after simplifying certain features to create a custom environment, the agent failed to develop an effective policy. The total rewards per episode fluctuated substantially, and the average rewards remained low and stable, indicating limited learning progress despite the extensive training .

The failure in the Pick and Place task can be attributed to several factors. Firstly, the complexity of coordinating multiple joints and navigating around obstacles requires precise control and adaptability, which the agent could not achieve. Secondly, the high-dimensional action space increases the difficulty of learning an effective policy within a reasonable timeframe. These results highlight the inherent challenges in applying DRL to complex, real-world tasks and underscore the need for further research and potential algorithmic advancements to overcome these obstacles.

Generalization capabilities were also a critical aspect of the research. To evaluate this, modifications were introduced to certain parameters in the environments where the agents were initially trained. These modifications tried to simulate realistic scenarios requiring the agents to adapt their learned policies. The experiments revealed varied outcomes. In some cases, the agents displayed a degree of robustness and adaptability to changes in parameters suggesting that the trained agents can generalize to new variations of the environments to some extent .

However, the results also indicated limitations in the agents' generalization abilities. For instance, in environments where the parameters were significantly altered beyond the ranges seen during training, the agents struggled to maintain performance. This observation points to the need for enhanced training methodologies that better equip agents to handle a wider range of variations and unexpected changes in their operational environments.

5.2 Conclusions

In conclusion, this thesis provides a comprehensive assessment of DRL through a detailed analysis of experimental outcomes. The research underscores both the successes and challenges encountered in training DRL agents across tasks of varying complexity. While the agents demonstrated robust performance in controlled environments, achieving high proficiency levels through meticulous hyperparameter tuning, they faced substantial difficulties when confronted with the complexities inherent in the Pick and Place task. This disparity highlights the current limitations of DRL methodologies in adapting to real-world scenarios with high-dimensional state and action spaces.

Furthermore, the generalization experiments conducted underscore the imperative for advancing training methodologies to enhance the adaptability and robustness of DRL agents. These findings contribute significantly to the broader understanding of DRL, offering insights into the factors influencing agent performance and laying the groundwork for future research endeavors. Addressing the identified challenges is crucial for advancing the practical application of DRL in real-world domains, promising innovative solutions to complex decision-making problems.

Chapter 6

FUTURE WORK

The findings of this study highlight several critical areas for future research to enhance the performance and generalization capabilities of DRL agents. Future experiments should test DRL agents in a broader array of environments, including more complex and dynamic scenarios. Expanding beyond standard control tasks to include environments with higher-dimensional state spaces, multi-agent interactions, and varied physical dynamics will provide deeper insights into the generalization abilities of DRL algorithms.

Addressing the challenges posed by complex tasks, such as the Pick and Place scenario, could involve exploring hierarchical reinforcement learning (HRL), which decomposes tasks into simpler sub-tasks, and curriculum learning, where the agent is trained on progressively more challenging tasks. Meta-learning, or ”learning to learn,” can help agents adapt quickly to new tasks by leveraging prior experience, enabling better generalization to new environments. Domain randomization, which involves training agents in a variety of simulated environments with randomized parameters, can also enhance robustness by forcing agents to learn policies that perform well across different variations.

Transfer learning can be used to leverage knowledge from one task or environment to improve performance on another related task, potentially accelerating learning and achieving better generalization. Advanced hyperparameter optimization techniques, such as Bayesian optimization and automated machine learning (AutoML), can be employed to find optimal hyperparameter configurations more efficiently.

Leveraging advancements in computational power and parallel processing will enable more extensive experimentation and faster training cycles. Utilizing distributed training frameworks and cloud-based resources can facilitate the development and testing of more complex models and algorithms, significantly reducing training times and allowing for more comprehensive exploration of different techniques and environments.

Incorporating human feedback into the training process, through methods like reward shaping and imitation learning, can guide the agent's learning and improve performance on complex tasks. Exploring and developing new DRL algorithms that address the specific challenges identified in this study, such as exploration-exploitation balance, sample efficiency, and stability in learning, can also overcome the limitations of current DRL approaches.

Future research in DRL should adopt a multi-faceted approach, including expanded environment testing, advanced training methodologies, leveraging computational advancements, and exploring novel algorithms. By addressing the limitations and challenges identified in the current study, these future experiments and techniques can develop more robust, adaptable, and generalizable DRL agents, paving the way for more effective real-world applications.

Bibliography

[Bellman, 1958] Bellman, R. (1958). Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239.

[Comanici and Precup, 2010] Comanici, G. and Precup, D. (2010). Optimal policy switching algorithms for reinforcement learning. volume 2, pages 709–714.

[de Lazcano et al., 2023] de Lazcano, R., Andreas, K., Tai, J. J., Lee, S. R., and Terry, J. (2023). Gymnasium robotics.

[Fernandez-Fernandez et al., 2023] Fernandez-Fernandez, R., Victores, J. G., and Balaguer, C. (2023). Deep robot sketching: An application of deep q-learning networks for human-like sketching. *Cognitive Systems Research*, 81:57–63.

[Francois et al., 2018] Francois, V., Henderson, P., Islam, R., Bellemare, M., and Pineau, J. (2018). An introduction to deep reinforcement learning.

[Jang et al., 2019] Jang, B., Kim, M., Harerimana, G., and Kim, J. W. (2019). Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667.

[Kirk et al., 2021] Kirk, R., Zhang, A., Grefenstette, E., and Rocktäschel, T. (2021). A survey of generalisation in deep reinforcement learning. *CoRR*, abs/2111.09794.

[Konda and Tsitsiklis, 2001] Konda, V. and Tsitsiklis, J. (2001). Actor-critic algorithms. *Society for Industrial and Applied Mathematics*, 42.

[Li et al., 2022] Li, M., Kazemi, A., Laguna, A. F., and Hu, X. S. (2022). Associative memory based experience replay for deep reinforcement learning. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, New York, NY, USA. Association for Computing Machinery.

[Meng et al., 2020] Meng, Z., Hu, Y., and Ancey, C. (2020). Using a data driven approach to predict waves generated by gravity driven mass flows. *Water*, 12.

[Mijwil, 2021] Mijwil, M. (2021). Artificial neural networks advantages and disadvantages. *Mesopotamian Journal of Big Data*, 2021:29–31.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

[Mukhamediev, 2015] Mukhamediev, R. (2015). Machine learning methods: An overview. *CMNT*, 19:14–29.

[OpenAI Gym Tutorial, 2018] OpenAI Gym Tutorial (2018). Mark Gao's AI Blog.

[Otterlo and Wiering, 2012] Otterlo, M. and Wiering, M. (2012). Reinforcement learning and markov decision processes. *Reinforcement Learning: State of the Art*, pages 3–42.

[Plappert et al., 2018] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

[Sebastianelli et al., 2021] Sebastianelli, A., Tipaldi, M., Ullo, S., and Glielmo, L. (2021). A deep q-learning based approach applied to the snake game.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

[Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

[Towers et al., 2023] Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., and Younis, O. G. (2023). Gymnasium.

[Wang and Snooks, 2021] Wang, D. and Snooks, R. (2021). *Artificial Intuitions of Generative Design: An Approach Based on Reinforcement Learning*, pages 189–198.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

[Yi and Liu, 2023] Yi, J. and Liu, X. (2023). Deep reinforcement learning for intelligent penetration testing path design. *Applied Sciences*, 13(16).

Appendix A

DETAILED ENVIRONMENT DESCRIPTIONS

Gymnasium provides a versatile toolkit for single-agent RL, offering a diverse range of environments, from straightforward tasks to complex scenarios. These environments, represented as instances of Python classes known as Environments (*Env*), adhere to the principles of Markov Decision Processes (MDPs), fostering interactions between RL agents and simulated environments. Gymnasium environments offer a standardized interface for training and evaluation which comprises four essential functions: *make*, *reset*, *step*, and *render*.

These functions are integral to the RL workflow, enabling the initialization of environments, resetting them to their initial states, executing actions, receiving observations and rewards, and rendering visualizations of the environment.

Initializing environments in Gymnasium is straightforward using the *make* function. For instance, invoking *gym.make('CartPole-v1')* creates an instance of the CartPole environment, ready for interaction. Gymnasium allows users to customize environments by specifying parameters during initialization, enabling them to tailor the environment to their specific needs. Additionally, Gymnasium provides Wrappers, which modify or augment environment functionalities, facilitating further customization and experimentation.

Interactions with environments follow the classical agent-environment loop of RL, as figure 2.1 illustrates. This iterative process involves initializing the environment and iteratively taking actions, receiving observations and rewards until reaching a terminal state. The action space defines the set of possible actions that an agent can take, while the observation space describes the information available to the agent about the state of the environment. These spaces are crucial for developing RL algorithms, as they determine the input and output of the agent-environment interaction. Gymnasium environments provide clear definitions of action and observation spaces through attributes such as *env.action_space* and *env.observation_space*, enabling users to understand the expected inputs and outputs of the environment.

Gymnasium supports a wide array of space types essential for RL algorithm development, including Box, Discrete, Dict, Tuple, MultiBinary, and MultiDiscrete.

A.1 Acrobot Environment

In this environment, the Acrobot consists of two links connected linearly to form a chain, with one end of the chain fixed. The joint between the two links is actuated. The goal is to apply torques on the actuated joint to swing the free end of the linear chain above a given height while starting from the initial state of hanging downwards. Figure A.1 illustrates the Acrobot environment and the goal of the task.

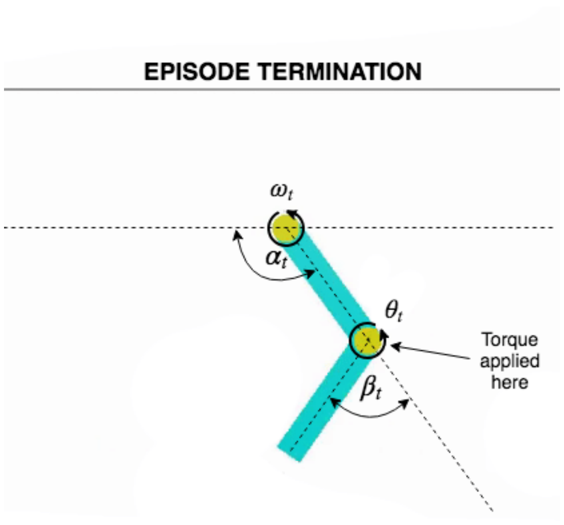

Figure A.1: Acrobot environment showing the two-link pendulum system with the actuated joint. The goal is to swing the free end above the designated height.

Action Space

The action space is discrete and deterministic, representing the torque applied on the actuated joint between the two links. The agent can choose from three possible actions:

1. **Apply -1 torque:** Exerts a negative torque on the actuated joint.

2. **Apply 0 torque:** Exerts no torque on the actuated joint.

3. **Apply 1 torque:** Exerts a positive torque on the actuated joint.

This action space allows the agent to influence the motion of the Acrobot by selecting the appropriate torque to apply at each timestep.

Observation Space

The observation space provides information about the two rotational joint angles and their angular velocities. It is represented as an *ndarray* with shape $(6,)$:

- **Cosine of θ_1:** The cosine of the angle of the first joint.
- **Sine of θ_1:** The sine of the angle of the first joint.
- **Cosine of θ_2:** The cosine of the angle of the second joint relative to the first link.
- **Sine of θ_2:** The sine of the angle of the second joint relative to the first link.
- **Angular velocity of θ_1:** The angular velocity of the first joint.
- **Angular velocity of θ_2:** The angular velocity of the second joint.

These observations enable the agent to understand the current state of the Acrobot, including the positions and movements of the links.

Reward Definition

The reward system is designed to guide the agent towards achieving the goal of swinging the free end above the designated height. Rewards are defined as follows:

- **Step Reward:** Each step incurs a reward of -1 until the goal is achieved.
- **Goal Achievement:** Achieving the target height results in termination with a reward of 0.

The primary objective is to reach the goal in as few steps as possible, motivating the agent to learn efficient control strategies.

Starting State

The starting state of the Acrobot is initialized with some initial stochasticity. Each parameter in the underlying state (θ_1, θ_2, and the two angular velocities) is uniformly initialized between -0.1 and 0.1. This means both links start pointing downwards with slight variations in their positions and velocities.

Episode End

An episode in the Acrobot environment ends if one of the following conditions is met:

- **Termination:** The free end of the Acrobot reaches the target height, defined as $-cos(\theta_1) - cos(\theta_2 + \theta_1) > 1.0$.

- **Truncation:** The episode length exceeds 500 steps.

These criteria ensure that episodes terminate when the goal is achieved or when the task becomes excessively long without success.

A.2 CartPool Environment

In this environment, a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum (pole) is placed upright on the cart, and the goal is to balance the pole by applying forces to the left or right on the cart. Figure A.2 illustrates the CartPole environment and the goal of the task.

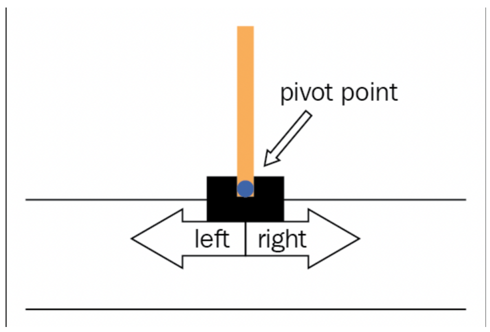

Figure A.2: CartPole environment showing the cart and the attached pole. The goal is to balance the pole by applying forces to the left or right on the cart.

Action Space

The action space is discrete and consists of two possible actions, representing the direction of the fixed force applied to the cart:

1. **Push cart to the left:** Action 0.
2. **Push cart to the right:** Action 1.

This action space allows the agent to influence the motion of the cart by selecting the appropriate force to apply at each timestep.

Observation Space

The observation space provides information about the positions and velocities of the cart and pole. It is represented as an *ndarray* with shape $(4,)$:

- **Cart Position:** The position of the cart along the track.
- **Cart Velocity:** The velocity of the cart.
- **Pole Angle:** The angle of the pole relative to the vertical.
- **Pole Angular Velocity:** The angular velocity of the pole.

These observations enable the agent to understand the current state of the CartPole system, including the positions and movements of the cart and pole.

Reward Definition

The reward system is designed to guide the agent towards achieving the goal of balancing the pole. Rewards are defined as follows:

- **Step Reward:** A reward of +1 is given for every step taken, including the termination step.
- **Alternate Reward:** If *sutton_barto_reward=True*, a reward of 0 is given for every non-terminating step and -1 for the terminating step.

The primary objective is to keep the pole upright for as long as possible, motivating the agent to learn effective control strategies.

Starting State

The starting state of the CartPole is initialized with some initial stochasticity. Each observation parameter (cart position, cart velocity, pole angle, and pole angular velocity) is uniformly initialized between -0.05 and 0.05. This means the cart and pole start in a near-upright position with slight variations in their positions and velocities.

Episode End

An episode in the CartPole environment ends if one of the following conditions is met:

- **Termination:** The pole angle exceeds ±12°.
- **Termination:** The cart position exceeds ±2.4 units.
- **Truncation:** The episode length exceeds 500 steps.

These criteria ensure that episodes terminate when the pole falls over or the cart moves out of the allowable range, or when the episode becomes excessively long without failure.

A.3 Mountain Car Continuous Environment

In this environment, a car is placed at the bottom of a sinusoidal valley with the goal of reaching the top of the right hill. The car can be accelerated in either direction using continuous actions. Figure A.3 illustrates the Mountain Car Continuous environment and the goal of the task.

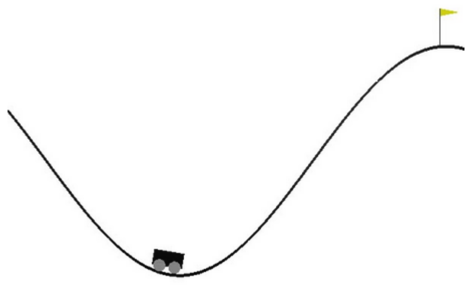

Figure A.3: Mountain Car Continuous environment showing the car in a valley. The goal is to accelerate the car to reach the top of the right hill.

Action Space

The action space is continuous and consists of a single action representing the directional force applied to the car. The action is an *ndarray* with shape $(1,)$, representing the force applied to the car. The action is clipped in the range $[-1, 1]$ and multiplied by a power of 0.0015.

This action space allows the agent to control the car's acceleration in a continuous manner, influencing its movement along the track.

Observation Space

The observation space provides information about the car's position and velocity. It is represented as an *ndarray* with shape $(2,)$:

- **Position:** The position of the car along the x-axis.
- **Velocity:** The velocity of the car.

These observations enable the agent to understand the current state of the car's movement within the valley.

Transition Dynamics

The transition dynamics of the car's movement are defined by the following equations:

$$\text{velocity}_{t+1} = \text{velocity}_t + \text{force} \times \text{power} - 0.0025 \times \cos(3 \times \text{position}_t) \quad \text{(A.1)}$$

$$\text{position}_{t+1} = \text{position}_t + \text{velocity}_{t+1} \quad \text{(A.2)}$$

where the force is the action clipped to the range $[-1, 1]$ and the power is a constant 0.0015. Collisions at either end of the track are inelastic, setting the velocity to 0 upon collision with the wall. The position is clipped to the range $[-1.2, 0.6]$ and the velocity is clipped to the range $[-0.07, 0.07]$.

Reward Definition

The reward system is designed to penalize large actions and reward reaching the goal:

- **Step Reward:** A negative reward of $-0.1 \times \text{action}^2$ is received at each timestep to penalize large magnitude actions.
- **Goal Reward:** If the car reaches the goal (position ≥ 0.45), a positive reward of +100 is added to the negative reward for that timestep.

This reward structure encourages the agent to reach the goal efficiently while avoiding large, unnecessary actions.

Starting State

The starting state of the car is initialized with some initial stochasticity. The position of the car is uniformly initialized between $[-0.6, -0.4]$, and the starting velocity of the car is always set to 0. This ensures the car starts near the bottom of the valley with a small random variation in its position.

Episode End

An episode in the Mountain Car Continuous environment ends if one of the following conditions is met:

- **Termination:** The position of the car is greater than or equal to 0.45 (the goal position on top of the right hill).
- **Truncation:** The length of the episode exceeds 999 steps.

These criteria ensure that episodes terminate when the car reaches the goal or when the episode becomes excessively long without success.

A.4 Pendulum Environment

This environment, involves a pendulum attached at one end to a fixed point, with the other end being free. The goal is to apply torque to the free end to swing the pendulum into an upright position, with its center of gravity directly above the fixed point. Figure A.4 illustrates the Pendulum environment and the goal of the task.

Action Space

The action space is continuous and consists of one action representing the torque applied to the free end of the pendulum:

1. **Torque:** The torque applied, which can range from -2.0 to 2.0 Nm.

This action space allows the agent to influence the motion of the pendulum by applying the appropriate torque at each timestep.

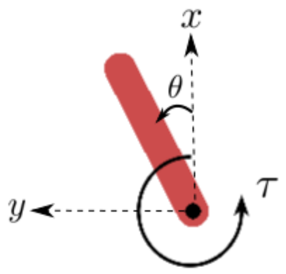

Figure A.4: Pendulum environment showing the pendulum and the applied torque. The goal is to swing the pendulum upright by applying torque to the free end.

Observation Space

The observation space provides information about the state of the pendulum. It is represented as an *ndarray* with shape $(3,)$:

- **x = cos(θ):** The cosine of the pendulum's angle.
- **y = sin(θ):** The sine of the pendulum's angle.
- **Angular Velocity:** The angular velocity of the pendulum.

These observations enable the agent to understand the current state of the pendulum, including its position and movement.

Reward Definition

The reward system is designed to guide the agent towards achieving the goal of swinging the pendulum upright. The reward function is defined as follows:

$$r = -(\theta^2 + 0.1 \cdot \theta_{\text{dt}}^2 + 0.001 \cdot \text{torque}^2) \tag{A.3}$$

where θ is the pendulum's angle normalized between $[-\pi, \pi]$ (with 0 being the upright position). The reward function penalizes the square of the angle, the square of the angular velocity, and the square of the applied torque, encouraging the agent to minimize these quantities to achieve the upright position.

The minimum reward that can be obtained is $-(\pi^2 + 0.1 \cdot 8^2 + 0.001 \cdot 2^2) = -16.2736044$, while the maximum reward is 0, which occurs when the pendulum is upright with zero velocity and no torque applied.

Starting State

The starting state of the pendulum is initialized with a random angle in $[-\pi, \pi]$ and a random angular velocity in $[-1, 1]$. This means the pendulum starts in a random position with slight variations in its velocity.

Episode Truncation

An episode in the Pendulum environment truncates after 200 timesteps, providing a fixed-length horizon for the agent to achieve the task.

A.5 Custom Pick and Place Environment

The environment selected for training the RL agent is a customization of the environment introduced in the research paper titled *Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research*, derived from the Gymnasium Robotics toolkit [de Lazcano et al., 2023]. While originally based on the "*FetchPickAndPlace-v2*" environment, see Section 2.5, the modifications aimed to streamline the task for enhanced training efficiency.

In this customized environment, the *7-DoF Fetch* manipulator is tasked with manipulating a block to a designated target position on a table surface. Figure A.5 illustrates how the environment and the manipulator looks.

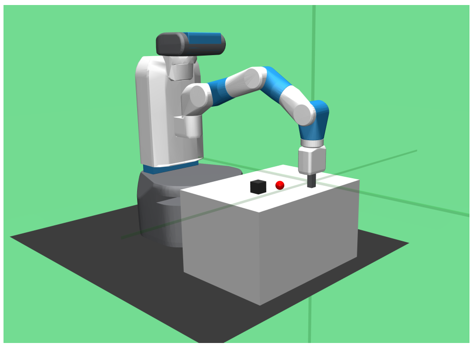

Figure A.5: The task is to grasp a box and move it to the target location on the table surface.

Unlike the original environment, certain aspects have been simplified to facilitate training. Notably, the target is always positioned on the table surface, eliminating the possibility of mid-air targets. Furthermore, the reward system has been configured to provide dense rewards based on the Euclidean distance between the achieved and desired goal positions.

Control of the robot remains consistent with the original environment, relying on small displacements of the gripper in Cartesian coordinates. The underlying MuJoCo framework handles inverse kinematics computations to facilitate gripper movements. Additionally, the gripper retains its functionality to open or close, essential for grasping operations in Pick and Place tasks.

Similar to the original environment, the customized version maintains the continuous nature of the task, requiring the robot to sustain the block in the target position indefinitely. To ensure smooth control, the robot operates at a frequency of $f = 25Hz$, with actions applied over consecutive simulator steps, each with a time step of $\Delta t = 0.002s$, before returning control to the robot.

Despite the modifications for training simplicity, the custom environment retains the challenge and realism necessary for effective reinforcement learning experiments. Its tailored scenarios for robotic manipulation tasks offer ample opportunities for algorithm development and evaluation, as explored further in this study. Furthermore, leveraging the Mujoco Python binding, overseen by the MuJoCo team at DeepMind, ensures access to cutting-edge simulation capabilities, essential for simulating realistic robotic environments.

Action Space

The action space is represented as a Box space with parameters indicating its type, range, shape, and data type. Specifically, it is denoted as a Box(-1.0, 1.0, (4,), float32), signifying a continuous space with a range from -1.0 to 1.0 in each dimension. This space comprises four dimensions, each corresponding to a specific action component:

1. **Displacement in the x direction (dx):** Controls the displacement of the end effector along the x-axis, with values ranging from -1 to 1.

2. **Displacement in the y direction (dy):** Governs the displacement of the end effector along the y-axis, with values between -1 and 1.

3. **Displacement in the z direction (dz):** Manages the displacement of the end effector along the z-axis, with values ranging from -1 to 1.

4. **Gripper control:** Regulates the opening and closing of the gripper. This component represents the positional displacement per timestep of each finger of the gripper, with values ranging from -1 to 1.

These components allows the agent to interact with the environment by selecting actions that influence the movement and behavior of the robot's end effector, which enables the robot to perform tasks such as grasping and placing objects within the environment.

Observation Space

The observation space provides goal-aware information about the robot's end effector state and its intended goal. It comprises a dictionary containing key details derived from Mujoco bodies, including sites attached to relevant entities such as the block and the end effector.

The observation space is structured around three main keys:

- **Observation:** This *ndarray* of shape (25,) contains kinematic information about both the block object and the gripper. It includes details such as the end effector's position in global coordinates, the block's position and rotation, relative block position with respect to the gripper, joint displacement of the gripper fingers, and various velocity components.
- **Desired Goal:** This key signifies the final goal to be achieved, represented as a 3-dimensional ndarray containing the desired final block position in Cartesian coordinates (x, y, z). The goal position may be located either on the table surface or elevated above it, facilitating Pick and Place trajectories.
- **Achieved Goal:**This key reflects the current state of the block, simulating its position as if it had already achieved the goal. It serves as a valuable resource for goal-oriented learning algorithms like Hindsight Experience Replay (HER), providing insight into the current state of the environment relative to the desired outcome.

Reward Definition

The reward space defines the feedback mechanism used to guide the agent's learning process. In this environment, two types of rewards can be used: sparse and dense. Sparse rewards provide a binary feedback (-1 or 0), where -1 is given if the block hasn't reached its final target position, and 0 if it has. Dense rewards, on the other hand, provide a continuous feedback based on the Euclidean distance between the achieved and desired goal positions. These reward signals help the agent learn the optimal policy for completing the task effectively.

Appendix B

COMPREHENSIVE ANALYSIS OF EXPERIMENTAL RESULTS

This appendix provides a comprehensive examination of the results not covered in Chapter 4.1. The first section details the training procedures and outcomes of the DRL agent, see Section 3.2 , across four control tasks: Acrobot, CartPole, Mountain Car Continuous, and Pendulum.

This includes a thorough explanation of the methodologies employed, the hyperparameters used, and the performance metrics achieved. The second section presents an analysis of the performance of the top-performing models from each control task when applied to custom environments with slight modifications. This analysis evaluates the robustness and adaptability of these models to environmental changes.

B.1 Training Results

All training procedures for the control tasks followed a consistent methodology, as detailed extensively in Section 3.3, with Pendulum results provided as a representative example in Section 4.1. This section comprehensively explains the results pertaining to the Acrobot, Mountain Car Continuous, and CartPole environments.

B.1.1 Acrobot

In the Acrobot environment experiment, all tested learning rates (α) performed quite well, but optimization was crucial for achieving the best balance between rapid learning and stability. The results, as depicted in Figure B.1, show the performance across various learning rates and 1 million timesteps.

In this specific case, the variability of the average reward was closely studied being an important indicator of both training stability and the model's potential for generalization. An $\alpha = 0.01$ showed rapid initial improvement but was marked by high variability and instability. An $\alpha = 0.001$ achieved high rewards but exhibited some variability. The $\alpha = 0.0001$ demonstrated stable and steady improvement, making it the best choice for long-term performance. Conversely, while $\alpha = 0.00001$ ensured stability, its slow progress rendered it impractical for extended training periods.

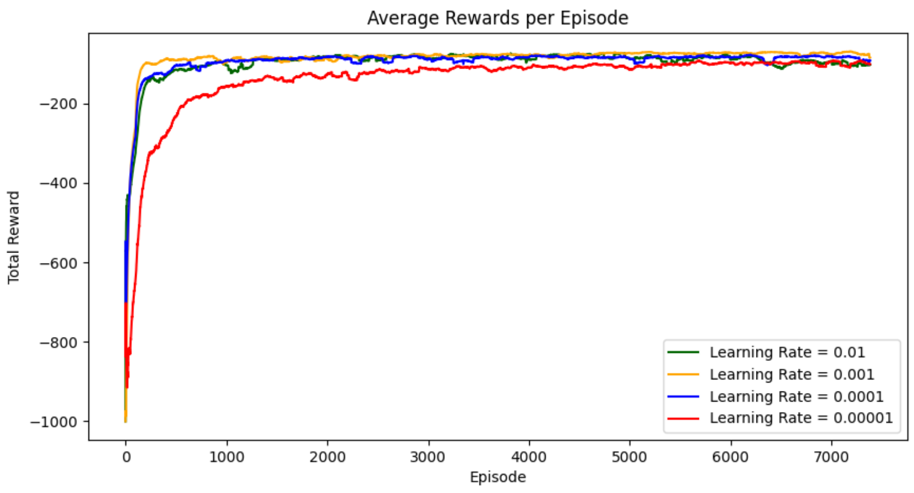

Figure B.1: Acrobot: Average Rewards Across Various α Values.

Based on these observations, $\alpha = 0.0001$ was selected for its balance of stability and consistent improvement. Further experiments with the selected learning rate aimed to determine the necessary number of timesteps for convergence. Figure B.8 illustrates the total rewards per episode over 1M timeteps.

Initially, there was rapid learning and adaptation within the first 1,000 episodes, characterized by a steep increase in total rewards. However, this period also showed some downward peaks in the average reward, particularly up to the established value of the maximum number of timesteps (1,000).

These downward peaks could indicate that the agent is still exploring suboptimal actions, leading to temporary drops in performance. In the intermediate phase, spanning from approximately 1,000 to 5,000 episodes, the total rewards continued to increase at a slower rate, with decreasing variability indicating more consistent actions. Around 5,000 episodes, the performance plateaued, with total rewards oscillating around a stable average, indicating minimal further improvement.

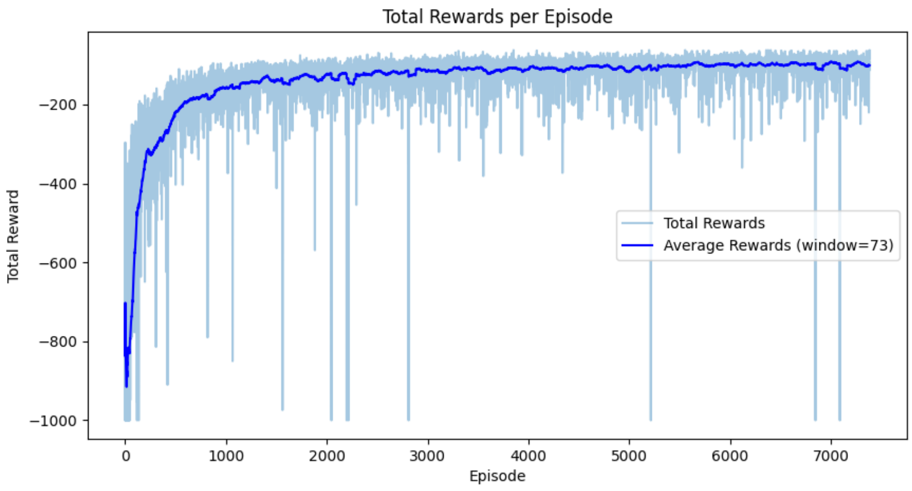

Figure B.2: Acrobot: Total Training Rewards per Episode.

The average rewards line, smoothed with a window of 100 episodes, highlights the trend of initial improvement followed by stabilization. The reduced width of the shaded area around the average rewards line indicates decreased variability, signifying more predictable and reliable performance.

Overall, the agent demonstrated successful learning, achieving consistent performance and a robust policy with $\alpha = 0.0001$. The final number of timesteps for the model will be set to 1 million, ensuring stable performance while optimizing computational resources.

B.1.2 CartPole

In the CartPole environment experiment, various learning rates (α) were tested to assess their impact on the agent's performance. The results, depicted in Figure B.3, show the performance across different learning rates.

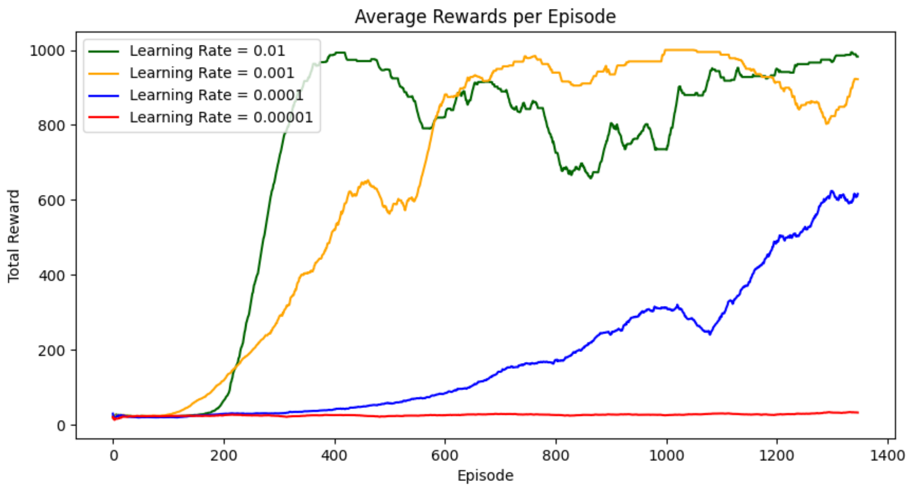

Figure B.3: CartPole: Average Rewards Across Various α Values.

The learning rate of $\alpha = 0.01$ resulted in rapid initial learning but exhibited significant instability and variability in the rewards. High variability in rewards can signify that the agent is still exploring and has not converged to a stable policy. This exploration might lead to fluctuating performance, which is undesirable for reliable training and is even more critical when evaluating the model's ability to generalize to new environments. The $\alpha = 0.001$ provided a better balance, achieving high rewards with moderate variability. In contrast, $\alpha = 0.0001$ showed more stable and steady improvement but had a lower learning capacity. This is more notable when decreasing the alpha value to $\alpha = 0.00001$; despite its stability, it was too slow to be practical for training within a reasonable time frame.

Based on these results, $\alpha = 0.001$ was selected for its stability and consistent improvement. However, $\alpha = 0.0001$ could also be a viable choice if prioritizing stability, though it requires additional computational resources and time.

Further experiments with the selected learning rate aimed to determine the necessary number of timesteps for convergence. Figure B.4 illustrates the total rewards per episode over 2 million timesteps.

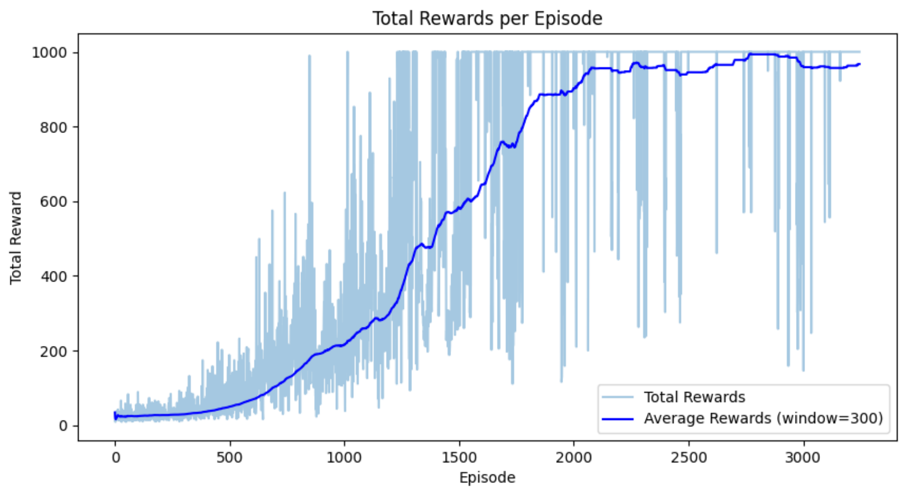

Figure B.4: CartPole: Total Training Rewards per Episode.

Initially, there was slow learning and adaptation within the first 500 episodes, characterized by a slight increase in total rewards. During the intermediate phase, from approximately 1,000 to 2,000 episodes, the total rewards increase at a higher rate, with notable variability. This increased variability could be attributed to the agent's attempts to fine-tune its policy and explore potentially better strategies. Around 2,000 episodes, the performance plateaued, with total rewards stabilizing around a consistent average.

The average rewards line, smoothed with a window of 100 episodes, shows the trend of initial improvement followed by stabilization. Significant variability is evident in the total rewards (represented by the blue shaded area). This variability can be attributed to the inherent exploratory nature of the PPO algorithm. During training, the agent sometimes deviates from optimal actions to explore the action space, which can lead to temporary drops in performance. As training progresses, the agent becomes better at selecting optimal actions, reducing variability and improving overall performance. The smoothed average rewards line demonstrates this trend, highlighting the agent's increasing stability and consistency over time.

Overall, the agent successfully learned and achieved consistent performance with $\alpha = 0.001$. The final number of timesteps for the model was set to 2 million, ensuring stable performance while optimizing computational resources.

B.1.3 Mountain Car Continuous

In the Mountain Car Continuous environment experiment, various learning rates (α) were tested to assess their impact on the agent's performance. The results, depicted in Figure B.5, show the performance across different learning rates.

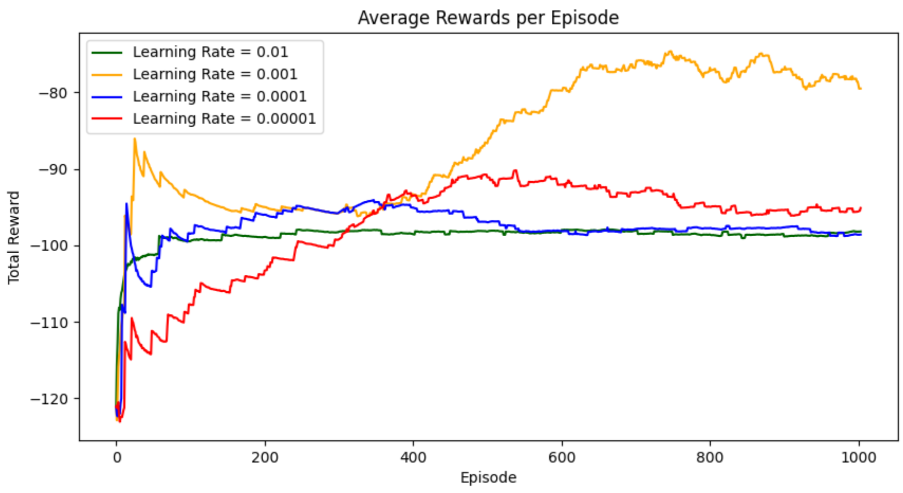

Figure B.5: Mountain Car Continuous: Average Rewards Across Various α Values.

The learning rate of $\alpha = 0.01$ shows a slow start with minimal initial improvement. Around the 200th episode, the performance slightly improves but remains largely stagnant with a slight negative trend. This indicates that $\alpha = 0.01$ may be too high, causing the agent to struggle with stability and failing to make significant progress in learning an effective policy. The $\alpha = 0.001$ l demonstrates the most promising performance. After a brief initial drop in rewards, there is a rapid and consistent improvement starting around the 50th episode. The agent continues to improve until approximately the 600th episode, where it reaches a plateau with the highest average reward among all tested learning rates. The consistent upward trend and stabilization suggest that $\alpha = 0.001$ offers a good balance between exploration and learning speed, leading to effective policy optimization. The learning rate $\alpha = 0.0001$ shows slow and steady improvement, with rewards stabilizing around the 200th episode and minimal improvement thereafter, indicating that while stable, it may be too slow for practical training as the agent struggles to achieve higher rewards. The learning rate $\alpha = 0.00001$ demonstrates the slowest and least effective learning, with an initial significant decrease in rewards followed by gradual improvement, but overall low and stable performance after the 200th episode, suggesting it is too small for effective progress within the training period.

Further experiments with the selected learning rate aimed to determine the necessary number of timesteps for convergence. Figure B.6 illustrates the total rewards per episode over 1 million timesteps.

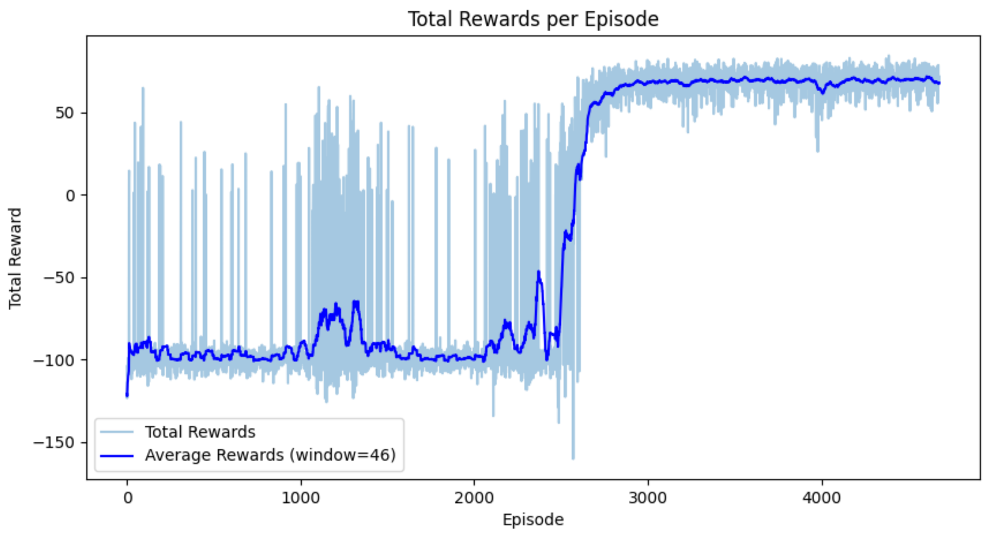

Figure B.6: Mountain Car Continuous:Total Training Rewards per Episode.

Initially, there was high variability in the rewards within the first 2500 episodes, characterized by significant fluctuations in performance. This high variability is typical in the early stages of training when the agent is exploring the environment and refining its policy. As training progressed, the agent's performance began to stabilize, and the variability in rewards decreased. By around 2500 episodes, the rewards became more consistent, indicating that the agent had learned a more stable and effective policy.

The average rewards line, smoothed with a window of 46 episodes, shows the trend of initial improvement followed by stabilization. The significant variability in total rewards (represented by the blue shaded area) gradually diminishes as the agent's policy becomes more refined and reliable. The smoothed average rewards line highlights this trend, showcasing the agent's increasing stability and consistency over time.

Overall, the agent successfully learned and achieved consistent performance with $\alpha = 0.001$. The final number of timesteps for the model was set to 3 million, ensuring stable performance while optimizing computational resources.

B.2 Generalization Results

This section provides a detailed explanation of the results obtained from various generalization experiments conducted in control task environments (Acrobot, CartPole, Mountain Car Continuous, and Pendulum). Table 3.2 summarizes the experiments conducted.

B.2.1 Acrobot

This section discusses the various generalization results obtained from experiments concerning the links' length and mass. Additionally, section 4.2.1 provides a detailed account of an experiment specifically focused on the center of mass.

Link Length Modifications

In the Acrobot environment, the lengths of the links were altered to evaluate the agent's robustness to changes in physical parameters, limiting each trial to a maximum of 400 steps to achieve the goal. This study yielded two distinct scenarios: the blue line illustrates the average rewards obtained when adjusting the length of Link 1 while keeping Link 2 at its trained length, while the orange line represents the average rewards obtained when adjusting the length of Link 2 while keeping Link 1 at its trained length.

Figure B.7 illustrates the average rewards achieved by the agent in each of the scenarios.

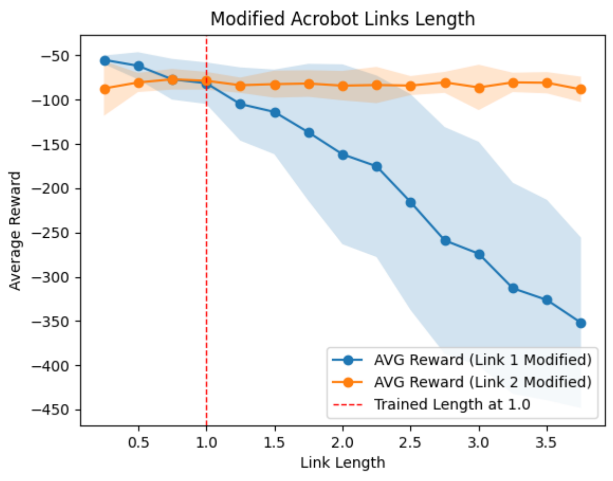

Figure B.7: Acrobot: Average Rewards Across with modified Link Lengths.

The plot illustrates that as the length of Link 1 increases from 0.5 to 3.5, there is a significant decrease in average rewards. This trend suggests a decline in the agent's performance, which initially remains relatively stable but deteriorates sharply as the link length moves further from the trained length (1.0), indicated by a red dashed line. A similar pattern is observed for modifications to Link 2, where performance remains relatively stable across the range of lengths tested.

The observed performance drop for Link 1 can be attributed to the fact that the agent's policy, developed during training, is optimized for the specific dynamics of the original link length configuration. Changes in the link lengths alter the dynamics of the system, affecting the state transitions and the control torques required to achieve the desired movements. These changes make the previously learned policy less effective in adapting to the new dynamics, resulting in reduced performance. However, despite these challenges and considering the reward structure of the environment, the agent is still able to reach the goal within less than 400 steps for most of the modified link lengths.

For Link 2, the agent's performance remains relatively stable, suggesting that the learned policy is more robust to changes in the length of Link 2 compared to Link 1. This indicates that the dynamics and control requirements for Link 2 are less critical to the agent's overall performance, or that the range of length modifications tested did not significantly impact the system's dynamics.

Overall, these results highlight the agent's sensitivity to changes in physical properties, particularly the length of Link 1, and underscore the importance of robustness in the design and training of reinforcement learning agents for control tasks with variable physical parameters.

Link Mass Modifications

In this experiment, the masses of the Acrobot's links were altered to evaluate the agent's robustness to changes in physical parameters, similar to the CoM position modification study. Each trial was limited to a maximum of 400 steps to achieve the goal. This study yielded two distinct scenarios: the blue line illustrates the average rewards obtained when adjusting the mass of Link 1 while keeping Link 2 at its trained mass, and the orange line represents the average rewards obtained when adjusting the mass of Link 2 while keeping Link 1 at its trained mass.

Figure B.8 shows the agent's performance as the mass of each link varies from 0.5 to 4.0 units, with the original trained mass indicated by the red dashed line at 1.0 units. The blue line shows that when the mass of Link 1 is close to its trained value (1.0), the agent maintains a relatively stable performance with minor fluctuations in rewards.

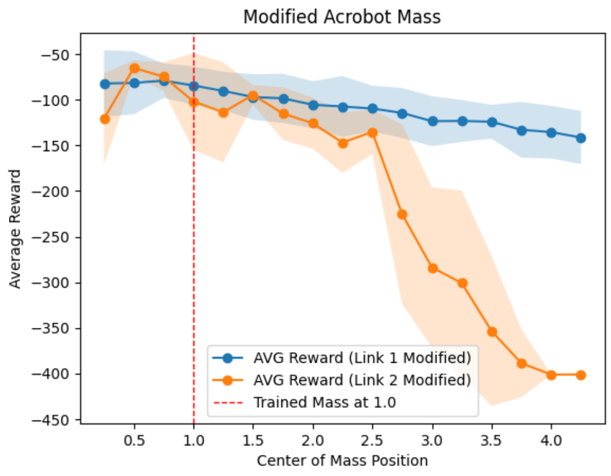

Figure B.8: Acrobot: Average Rewards with modified Mass.

However, as the mass of Link 1 increases beyond 1.5 units, there is a noticeable decline in the average rewards, which continues gradually, indicating that the agent's performance is increasingly sensitive to the changes in Link 1's mass. In contrast, the orange line indicates that the agent's performance declines sharply as the mass of Link 2 increases beyond 1.0 units. The drop in average rewards is more significant compared to Link 1 modifications, suggesting a higher sensitivity to changes in the mass of Link 2. Additionally, the variance increases significantly with the mass of Link 2, reflecting greater inconsistency in the agent's ability to achieve the goal under varying mass conditions.

The observed performance drop for both scenarios can be attributed to the fact that the agent's policy, developed during training, is optimized for the specific dynamics of the original mass configuration. Changes in the masses of the links alter the dynamics of the system, affecting the state transitions and the control torques required to achieve the desired movements. These changes make the previously learned policy less effective in adapting to the new dynamics, resulting in reduced performance. However, despite these challenges and considering the reward structure of the environment, the agent is still able to reach the goal within less than 400 steps in most cases, indicating a degree of robustness in its policy.

B.2.2 CartPole

This section discusses the various CartPole generalization results obtained from experiments concerning gravity, pole length, and pole mass. Additionally, section 4.2.2 provides a detailed account of an experiment specifically focused on the force magnitude.

Gravity Modifications

In this scenario, the gravity value was modified between 5 and 15 values, with steps of 1, to evaluate the agent's robustness to changes in physical parameters.

Figure B.9 shows that when the gravity is close to the trained value of 9.8 (indicated by the red dashed line), the agent achieves relatively high rewards, indicating effective performance in keeping the pole upright. As the gravity value decreases below 9.8, the agent's performance remains relatively stable, with average rewards fluctuating but generally staying within a high range. This stability suggests that the agent is able to maintain the pole upright effectively under lower gravity conditions, although some variability in performance is observed, as indicated by the shaded region representing the variance.

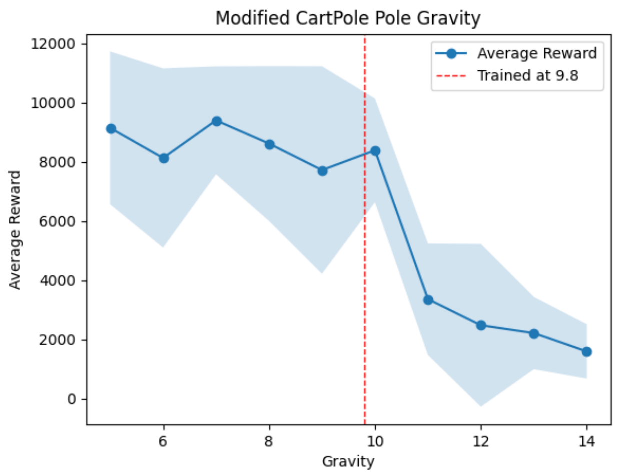

Figure B.9: CartPole: Average Rewards with modified gravity.

When the gravity increases beyond 9.8, the agent's performance begins to decline significantly, with a noticeable drop in average rewards. This sharp decline suggests that the agent's policy is more sensitive to higher gravity values, likely due to the increased difficulty in balancing the pole under such conditions. The increased variability in rewards at higher gravity values reflects greater inconsistency in the agent's ability to maintain balance under these more challenging conditions.

The observed performance drop can be attributed to the agent's policy being optimized for the specific dynamics of the original gravity configuration. Changes in gravity alter the system's dynamics, affecting the state transitions and the control efforts required to keep the pole upright. These changes make the previously learned policy less effective in adapting to the new dynamics, resulting in reduced performance. Despite these challenges, the agent demonstrates a degree of robustness, maintaining relatively high rewards within a range of gravity values, particularly those lower than the trained value.

Pole Length Modifications

On this occasion, the length of the pole was modified between 0.2 and 0.8, with steps of 0.1, to evaluate the agent's robustness to changes in physical parameters.

Figure B.10 shows that when the pole length is close to the trained value of 0.5 (indicated by the red dashed line), the agent achieves relatively high rewards, indicating effective performance in keeping the pole upright. As the pole length decreases below 0.5, the agent's performance improves, with average rewards increasing and reaching a peak around 0.5. This trend suggests that the agent's policy is well-suited for shorter pole lengths, where maintaining balance might be easier due to the lower inertia.

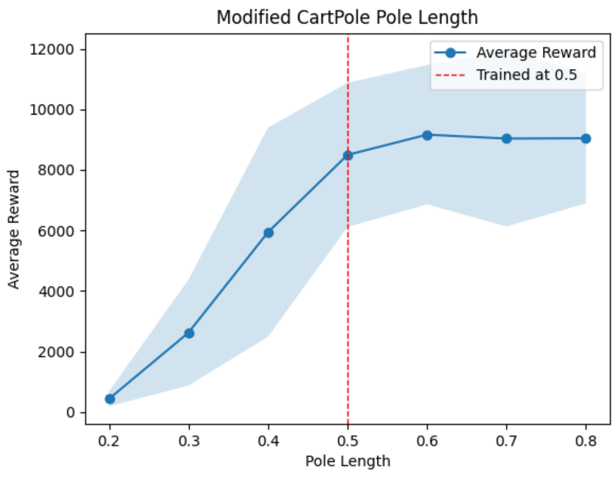

Figure B.10: CartPole: Average Rewards with modified Pole Length.

As the pole length increases beyond 0.5, the agent's performance begins to stabilize, with average rewards remaining relatively high and showing less variability. The shaded region representing the variance indicates that the agent's performance becomes more consistent with longer pole lengths, maintaining high rewards with reduced fluctuations.

The observed performance trends can be attributed to the agent's policy being optimized for the specific dynamics of the original pole length configuration. Changes in pole length alter the system's dynamics, affecting the state transitions and the control efforts required to keep the pole upright. These changes make the previously learned policy less effective in adapting to the new dynamics at significantly different lengths. Despite these changes, the agent demonstrates a strong degree of robustness, maintaining high rewards across a range of pole lengths, particularly those around the trained value of 0.5 and higher.

Pole Mass Modifications

At this time, the mass of the pole was modified between 0.01 and 0.6, with steps of 0.05, to evaluate the agent's robustness to changes in physical parameters. Figure B.11 shows that when the pole mass is close to the trained value of 0.1 (indicated by the red dashed line), the agent achieves relatively high rewards, indicating effective performance in keeping the pole upright. As the pole mass decreases below 0.1, the agent's performance remains relatively stable, with average rewards fluctuating but generally staying within a high range. This stability suggests that the agent is able to maintain the pole upright effectively with lighter pole masses.

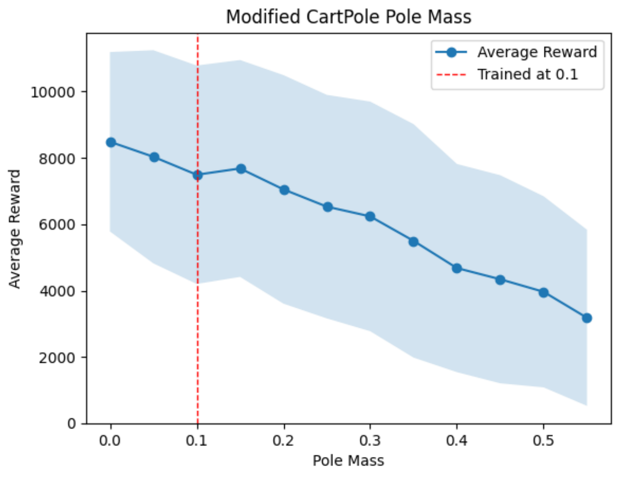

Figure B.11: CartPole: Average Rewards with modified Pole Mass.

However, as the pole mass increases beyond 0.1, the agent's performance begins to decline gradually, with a noticeable decrease in average rewards. The decline in performance is more pronounced with heavier pole masses, indicating that the agent's policy is more sensitive to increased pole mass. The increased variability in rewards, as indicated by the shaded region representing the variance, shows that the agent's performance becomes less consistent with heavier pole masses, reflecting greater difficulty in maintaining balance under these conditions.

The observed performance drop can be attributed to the agent's policy being optimized for the specific dynamics of the original pole mass configuration. Changes in pole mass alter the system's dynamics, affecting the state transitions and the control efforts required to keep the pole upright. These changes make the previously learned policy less effective in adapting to the new dynamics, resulting in reduced performance. Despite these challenges, the agent demonstrates a degree of robustness, maintaining relatively high rewards within a range of pole masses, particularly those around and below the trained value of 0.1.

B.2.3 Mountain Car Continuous

This section discusses the various Mountain Car Continuous generalization results obtained from experiments concerning the initial car position and the car's power. Additionally, section 4.2.3 provides a detailed account of an experiment specifically focused on friction.

Initial Car Position Modifications

In the Mountain Car Continuous environment, the initial position of the car was altered to evaluate the agent's robustness to changes in starting conditions. In the original environment, the initial position is established randomly between values -0.04 and -0.06, in the valley of the mountain. For this custom environment, the initial position has been set to fixed values on the x-axis between -0.6 and 0.3 with steps of 0.1. This setup allows an analysis of the model's capabilities to adapt when starting from different points on the hill. Each trial is limited to a maximum of 300 steps to reach the goal.

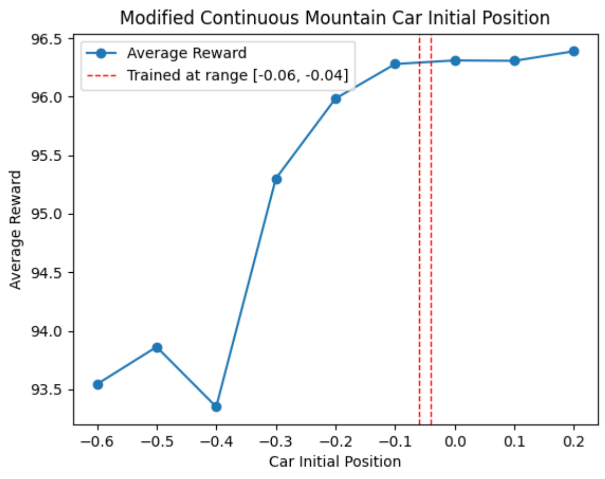

Figure B.12: Mountain Car Continuous: Average Rewards Across Various α Values.

Figure B.12 illustrates that when the car starts at positions between -0.6 and -0.4, there is a noticeable drop in average reward, indicating that the agent finds it more challenging to reach the goal efficiently from these starting points. The performance improves significantly as the initial position moves closer to -0.2, and remains relatively high and stable from -0.2 to 0.2. This suggests that the agent is better able to adapt and perform when starting from positions nearer to the trained range of -0.04 to -0.06, indicated by the red dashed lines.

The no variability in the average rewards across different trials suggests that the agent's performance is consistent for each fixed initial position. This consistency underscores the agent's robustness and reliability in adapting to different initial conditions within the tested range.

The observed performance drop for initial positions further from the trained range can be attributed to the fact that the agent's policy, developed during training, is optimized for starting conditions within the valley of the mountain. When starting from further up the hill, the agent encounters different state transitions and control challenges, making the previously learned policy less effective. However, the agent's ability to still achieve high rewards when starting from positions closer to the trained range demonstrates its capacity to leverage the environment's reward structure effectively.

Power Car Modifications

In this case, the power of the car was altered to evaluate the agent's robustness to changes in the car's engine power. The model was originally trained with a car power of 0.0015. In this custom environment, the power has been modified to values between 0.0005 and 0.003 with steps of 0.00025. This setup simulates scenarios where the car experiences variations in power due an avery, either losing or gaining engine strength. Each trial is limited to a maximum of 300 steps to reach the goal.

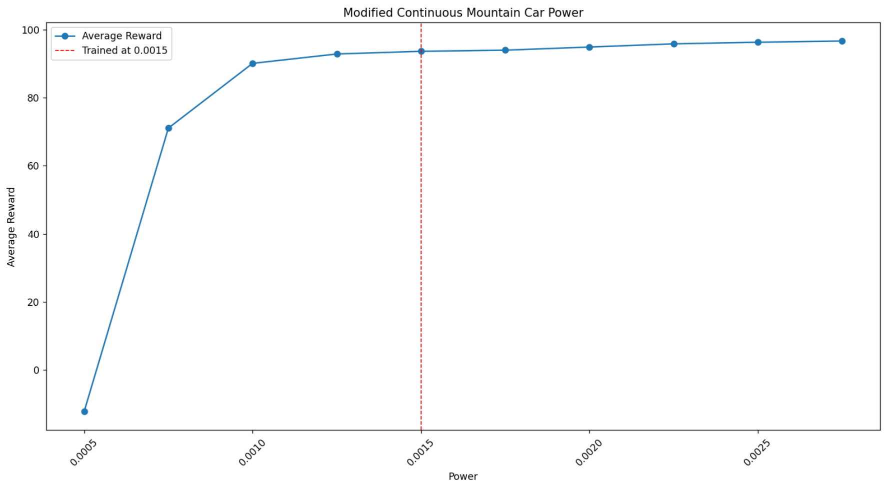

Figure B.13: Mountain Car Continuous: Average Rewards with modified Power (300 steps).

Figure B.14 shows that the average reward increases as the car power increases from 0.0005 to 0.003. When the power is set to 0.0005, the average reward is the negative, indicating that the agent was not able to reach the goal efficiently with lower power. As the power increases, the agent's performance improves significantly, with the highest rewards achieved at the power settings closest to and above the trained value of 0.0015, indicated by the red dashed line.

The trend suggests that the agent's policy, developed during training, is well-suited to handle scenarios with the trained power value and higher. The increase in performance with higher power settings indicates that the agent can better leverage the increased power to reach the goal more efficiently.

To further investigate whether the agent could reach the goal with a power value of 0.0005 given more time, a longer experiment with a maximum of 500 steps was conducted. The results of this experiment are illustrated in Figure B.14. This experiment demonstrates that the agent was able to reach the goal successfully but with more difficulty, which makes sense given the reduced power making it harder to climb the hill.

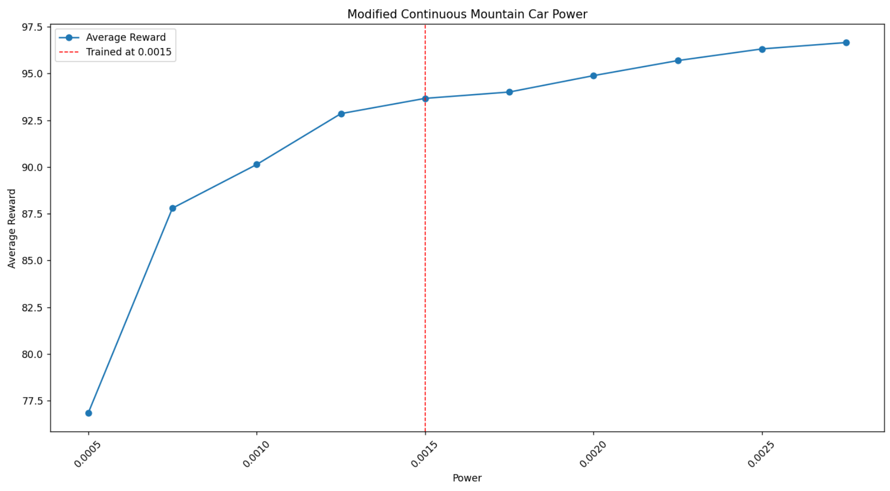

Figure B.14: Mountain Car Continuous: Average Rewards with modified Power (500 steps).

The increased difficulty and lower performance at reduced power levels highlight the agent's challenges in adapting to scenarios where the car's engine is weaker than the trained condition. Conversely, the agent's ability to maintain and even improve performance with power settings above the trained value underscores its capacity to leverage increased power effectively.

B.2.4 Pendulum

This section discusses the various Pendulum generalization results obtained from experiments concerning delta time, gravity, and pendulum mass. Additionally, section 4.2.4 provides a detailed account of an experiment specifically focused on pendulum length.

Delta Time Modified

Experiments were conducted to evaluate the agent's generalization capabilities by modifying the delta time, which simulates varying time intervals. The agent was initially trained with a delta time of 0.05, and the tested values ranged from 0.01 to 0.15 with steps of 0.01

The results, as illustrated in Figure B.15, show how changes in delta time affect the agent's performance. The agent achieves the highest average reward around the trained delta time of 0.05, indicating optimal performance under the conditions it was trained in. For delta times between 0.02 and 0.09, the agent maintains a relatively high average reward, demonstrating a good degree of generalization within this range. This suggests that the agent can adapt to moderately different time intervals without significant performance degradation.

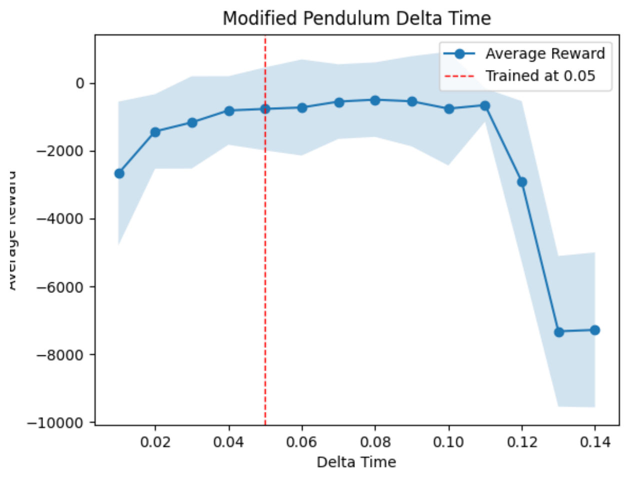

Figure B.15: Pendulum: Average Rewards with modified Delta Time.

However, as the delta time increases beyond 0.09, there is a noticeable decline in performance. Particularly, at delta times of 0.12 and higher, the average reward drops sharply, approaching highly negative values. This indicates that the agent struggles to reach the goal under these conditions, reflecting its limited ability to generalize to significantly different delta times. The sharp decline in performance at higher delta times suggests that the agent's policy is highly sensitive to the dynamics introduced by larger time intervals, which substantially alter the environment's behavior.

The variability in performance, as indicated by the shaded area around the average reward line, highlights the consistency of the agent's behavior across different runs. While there is some variability, particularly at extreme delta times, the overall pattern shows that the agent's performance is relatively stable within the mid-range of delta times but becomes less predictable and more variable as the delta time deviates significantly from the trained value.

These findings underscore the importance of training reinforcement learning agents in diverse conditions to enhance their robustness and ability to adapt to varying temporal dynamics. This is particularly relevant in real-world scenarios where time intervals may fluctuate, and ensuring that agents can handle these variations is crucial for their effective deployment.

Gravity Modified

At this time, the agent's generalization capabilities were evaluated by modifying the gravity. The agent was initially trained with a gravity value of 9.8, and the tested values ranged from 6 to 15 with steps of 1. The results, as illustrated in the provided plot, show how changes in gravity affect the agent's performance. The agent achieves the highest average reward near the trained gravity of 9.8, indicating optimal performance under the conditions it was trained in. For gravity values between 8 and 11, the agent maintains a relatively high average reward, demonstrating a reasonable degree of generalization within this range. This suggests that the agent can adapt to moderately different gravity settings without significant performance degradation.

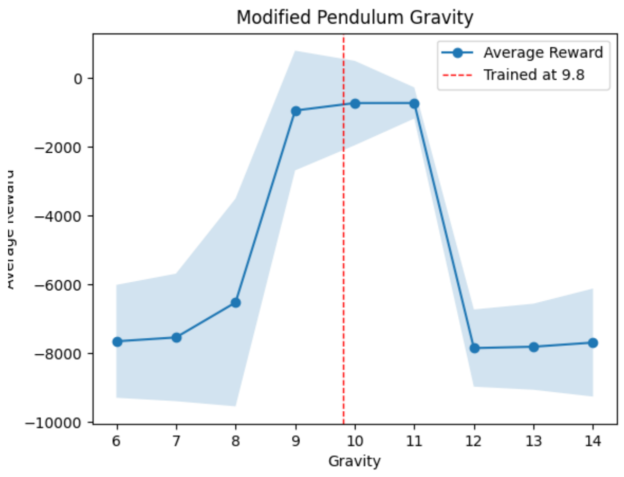

Figure B.16: Pendulum: Average Rewards with modified Gravity.

However, as the gravity increases beyond 11, there is a noticeable decline in performance. Particularly, at gravity values of 12 and higher, the average reward drops sharply, approaching highly negative values. This indicates that the agent struggles to reach the goal under these conditions, reflecting its limited ability to generalize to significantly different gravity levels. The sharp decline in performance at higher gravity values suggests that the agent's policy is highly sensitive to the dynamics introduced by stronger gravitational forces, which substantially alter the environment's behavior.

Conversely, for gravity values significantly lower than the trained value (6 to 8), the agent's performance is also suboptimal, with average rewards dropping to negative values. This indicates difficulties in reaching the goal under these lower gravity conditions, further reflecting the agent's limited generalization ability.

The variability in performance, as indicated by the shaded area around the average reward line, highlights the consistency of the agent's behavior across different runs. While there is noticeable variability, particularly at extreme gravity values, the overall pattern shows that the agent's performance is relatively stable within the mid-range of gravity but becomes less predictable and more variable as the gravity deviates significantly from the trained value.

Pendulum Mass Modified

In this case, experiments were conducted to evaluate the agent's generalization capabilities by modifying the mass of the pendulum. The agent was initially trained with a pendulum mass of 1.0, and the tested values ranged from 0.25 to 2.5 with steps of 0.25

Figure B.17 illustrates how the changes in pendulum mass affect the agent's performance. The agent achieves the highest average reward near the trained mass of 1.0, indicating optimal performance under the conditions it was trained in. For masses between 0.5 and 1.25, the agent maintains a relatively high average reward, demonstrating a reasonable degree of generalization within this range. This suggests that the agent can adapt to moderately different pendulum masses without significant performance degradation.

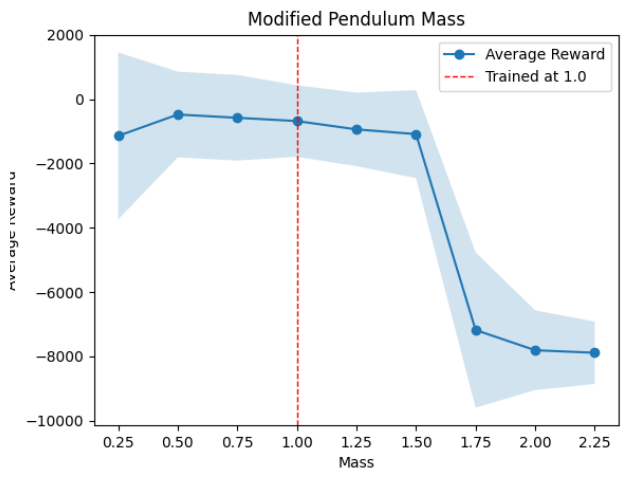

Figure B.17: Pendulum: Average Rewards with modified mass.

However, as the mass increases beyond 1.25, there is a noticeable decline in performance. Particularly, at masses of 1.5 and higher, the average reward drops sharply, approaching highly negative values. This indicates that the agent struggles to reach the goal under these conditions, reflecting its limited ability to generalize to significantly different pendulum masses. The sharp decline in performance at higher masses suggests that the agent's policy is highly sensitive to the dynamics introduced by larger pendulum masses, which substantially alter the environment's behavior.

The variability in performance, as indicated by the shaded area around the average reward line, highlights the consistency of the agent's behavior across different runs. While there is some variability, particularly at extreme masses, the overall pattern shows that the agent's performance is relatively stable within the mid-range of masses but becomes less predictable and more variable as the mass deviates significantly from the trained value.