Pontifícia Universidade Católica
do Rio de Janeiro

Renata Garcia Oliveira

Online Ensembles for Deep Reinforcement Learning in Continuous Action Spaces

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Engenharia Elétrica of the PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Engenharia Elétrica.

Advisor: Prof. Wouter Caarls

Rio de Janeiro
September 2021

Renata Garcia Oliveira

Online Ensembles for Deep Reinforcement Learning in Continuous Action Spaces

Thesis presented to the Programa de Pós–graduação em Engenharia Elétrica of the PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Engenharia Elétrica. Approved by the Examination Committee:

Rio de Janeiro, September the 14th, 2021

Renata Garcia Oliveira

Master in Electrical Engineering from the Federal Center for Technological Education Celso Suckow da Fonseca (CEFET–RJ – 2017), specializing in the area of intelligent methods to support decision making. Graduated in Computer Engineering from the Federal University of Pernambuco (UFPE–2008) and Post-Graduation from Candido Mendes University (UCAM–2012). Currently a public servant of the Ministry of Defense in the Department of Airspace Control (DECEA), working in the area of Information Technology since 2010.

Bibliographic data

Garcia Oliveira, Renata

Online Ensembles for Deep Reinforcement Learning in Continuous Action Spaces / Renata Garcia Oliveira; advisor: Wouter Caarls. – 2021.

88 f: il. color. ; 30 cm

Tese (doutorado) - Pontifical Catholic University of Rio de Janeiro, Departamento de Engenharia Elétrica, 2021.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Métodos de apoio à Decisão – Teses. 3. Aprendizado por Reforço. 4. Gradiente da política determinística profunda. 5. Comitê de Ações Contínuas. 6. Aprendizado por Comitê. 7. Otimização de Hiperparâmetro. I. Caarls, Wouter. II. Pontifical Catholic University of Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título.

CDD: 620.11

Acknowledgments

To Prof. Wouter Caarls, for accepting and welcoming me, as well as for all the support and understanding throughout the thesis development.

To my parents Renato and Rose, who always dedicated themselves to raise their children well.

To my husband Eduardo for always being by my side, making my daily life very pleasant and happy.

To my friends, Adalberto Igor, Errison, Evelyn, Gabriel, Franklin, Iam, Nicolae, Pêrci, Yessica, for the excellent relationship at PUC-Rio as well as for the countless help, with the laboratory, the study room, the computational tools and the licenses, plus the support and company during lunch, dinner and in the moments of relaxation. All of this was fundamental for the development of this thesis; for all of that, my most sincere thanks.

To my brothers, Diego and Bianca, for always being by my side, encouraging me.

To friends and co-workers who were patient with my available free hours.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Garcia Oliveira, Renata; Caarls, Wouter (Advisor). **Online Ensembles for Deep Reinforcement Learning in Continuous Action Spaces**. Rio de Janeiro, 2021. 88p. Tese de Doutorado – Departamento de Engenharia Elétrica, Pontifical Catholic University of Rio de Janeiro.

This work seeks to use ensembles of deep reinforcement learning algorithms from a new perspective. In the literature, the ensemble technique is used to improve performance, but, for the first time, this research aims to use ensembles to minimize the dependence of deep reinforcement learning performance on hyperparameter fine-tuning, in addition to making it more precise and robust. Two approaches are researched; one considers pure action aggregation, while the other also takes the value functions into account. In the first approach, an online learning framework based on the ensemble's continuous action choice history is created, aiming to flexibly integrate different scoring and aggregation methods for the agents' actions. In essence, the framework uses past performance to only combine the best policies' actions. In the second approach, the policies are evaluated using their expected performance as estimated by their value functions. Specifically, we weigh the ensemble's value functions by their expected accuracy as calculated by the temporal difference error. Value functions with lower error have higher weight. To measure the influence on the hyperparameter tuning effort, groups consisting of a mix of different amounts of well and poorly parameterized algorithms were created. To evaluate the methods, classic environments such as the inverted pendulum, cart pole and double cart pole are used as benchmarks. In validation, the Half Cheetah v2, a biped robot, and Swimmer v2 simulation environments showed superior and consistent results demonstrating the ability of the ensemble technique to minimize the effort needed to tune the the algorithms.

Keywords

Reinforcement Learning; Deep Deterministic Policy Gradient; Continuous Action Ensemble; Ensemble Learning; Hyperparameter Optimization.

Resumo

Garcia Oliveira, Renata; Caarls, Wouter. **Conjuntos Online para Aprendizado por Reforço Profundo em Espaços de Ação Contínua**. Rio de Janeiro, 2021. 88p. Tese de Doutorado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho busca usar o comitê de algoritmos de aprendizado por reforço profundo (deep reinforcement learning) sob uma nova perspectiva. Na literatura, a técnica de comitê é utilizada para melhorar o desempenho, mas, pela primeira vez, esta pesquisa visa utilizar comitê para minimizar a dependência do desempenho de aprendizagem por reforço profundo no ajuste fino de hiperparâmetros, além de tornar o aprendizado mais preciso e robusto. Duas abordagens são pesquisadas; uma considera puramente a agregação de ação, enquanto que a outra também leva em consideração as funções de valor. Na primeira abordagem, é criada uma estrutura de aprendizado online com base no histórico de escolha de ação contínua do comitê com o objetivo de integrar de forma flexível diferentes métodos de ponderação e agregação para as ações dos agentes. Em essência, a estrutura usa o desempenho passado para combinar apenas as ações das melhores políticas. Na segunda abordagem, as políticas são avaliadas usando seu desempenho esperado conforme estimado por suas funções de valor. Especificamente, ponderamos as funções de valor do comitê por sua acurácia esperada, calculada pelo erro da diferença temporal. As funções de valor com menor erro têm maior peso. Para medir a influência do esforço de ajuste do hiperparâmetro, grupos que consistem em uma mistura de diferentes quantidades de algoritmos bem e mal parametrizados foram criados. Para avaliar os métodos, ambientes clássicos como o pêndulo invertido, *cart pole* e *cart pole* duplo são usados como *benchmarks*. Na validação, os ambientes de simulação Half Cheetah v2, um robô bípede, e o Swimmer v2 apresentaram resultados superiores e consistentes demonstrando a capacidade da técnica de comitê em minimizar o esforço necessário para ajustar os hiperparâmetros dos algoritmos.

Palavras-chave

Aprendizado por Reforço; Gradiente da política determinística profunda; Comitê de Ações Contínuas; Aprendizado por Comitê; Otimização de Hiperparâmetro.

Table of contents

1 Introduction 16
1.1 Motivation 16
1.2 Ensemble Methods 17
1.3 Objectives 18
1.4 Contributions 19
1.5 Organization 20

2 Ensemble Reinforcement Learning 21
2.1 Reinforcement Learning (RL) 21
2.1.1 Q-Learning Algorithm 23
2.2 Deep Reinforcement Learning (Deep RL) 24
2.2.1 Deep Q-Learning 24
2.2.2 Deep Deterministic Policy Gradient (DDPG) 25
2.3 Ensembles in Discrete Action Spaces 27
2.3.1 Action Ensembles in Discrete Action Spaces 27
2.3.1.1 Majority Voting 27
2.3.1.2 Rank Voting 28
2.3.1.3 Boltzmann Multiplication 28
2.3.1.4 Boltzmann Addition 28
2.3.2 Q-Ensembles in Discrete Action Spaces 28
2.4 Ensembles in Continuous Action Spaces 30
2.4.1 Action Ensembles in Continuous Action Spaces 30
2.4.1.1 Mean 32
2.4.1.2 Density Based 32
2.4.1.3 Data Center 33
2.4.2 Q-Ensembles in Continuous Action Spaces 33
2.5 Policy Selection in Training Phase 34
2.5.1 Pre-training 34
2.5.2 Alternately Persistent 34
2.5.3 Parallel Environment 34

3 Ensemble Aggregation Methods 36
3.1 Online Training 36
3.2 Classical Aggregations in DDPG with Continuous Action Spaces 36
3.2.1 Evaluating Ensemble Performance 37
3.3 History-Based Framework (HBF) 38
3.3.1 HBF Implementation 40
3.3.2 Performance Measure in History-Based Framework 44
3.4 Online Weighted Q-Ensemble (OWQE) 45
3.4.1 Inference 45
3.4.2 Online Training Weights 47
3.4.3 Performance Measure in Online Weighted Q-Ensemble 49

4 Simulations 50

4.1 Model Architecture 50
4.1.1 Initial Configuration of the Architecture 50
4.2 Ensemble Groups 51
4.2.1 *3 Best or 3 Good Group* 52
4.2.2 Action Aggregation Groups 52
4.2.2.1 *Good Group* 53
4.2.2.2 *Mid Group* 53
4.2.2.3 *Bad Group* 53
4.2.3 Value Aggregation Groups 53
4.2.3.1 *1 Good 1 Bad Group* 53
4.2.3.2 *1 Good 3 Bad Group* 53
4.2.3.3 *1 Good 7 Bad Group* 54
4.3 Strategy Experiments of History-Based Framework 54
4.4 Strategy Experiments of Online Weighted Q-Ensemble 55
4.5 Environment 56
4.5.1 Inverted Pendulum Swing-up (PD) 57
4.5.2 Cart Pole (CP) 57
4.5.3 Cart Double Pole (CDP) 58
4.5.4 Half Cheetah v2 (HC) 59
4.5.5 Swimmer v2 (SW) 60

5 Results 63
5.1 Individual Results 63
5.2 Classical Aggregations in DDPG with Continuous Action Spaces 64
5.3 History-Based Framework (HBF) 68
5.3.1 Ensemble of Pre-Trained Algorithms 68
5.3.2 Ensemble Online Training 70
5.3.3 Half Cheetah Test 70
5.3.4 Discussion 71
5.4 Online Weighted Q-Ensembles (OWQE) 73
5.4.1 Learning Curve 75
5.4.2 Action Preference and Q-Weights 75
5.5 Discussions 78

6 Conclusions and Future Proposals 80

Bibliography 82

A Appendix — Single Run 87

List of figures

Figure 2.1 Reinforcement Learning. 22
Figure 2.2 Deep Deterministic Policy Gradient (DDPG) Algorithm. 26
Figure 2.3 Majority Voting. 27
Figure 2.4 Cycling Leaning Rate. 31
Figure 2.5 Illustration of Density Based Aggregation. 32

Figure 3.1 Classical Ensemble Aggregation Strategy. 37
Figure 3.2 Generic History-Based Framework (HBF). 39
Figure 3.3 History-Based Framework – Scoring. 40
Figure 3.4 History-Based Framework – Moving Average. 40
Figure 3.5 History-Based Framework – Ensemble Aggregation. 41
Figure 3.6 History-Based Framework Implementation. 43
Figure 3.7 Illustration of Regret Metric. 45
Figure 3.8 Q Matrix Generation in DDPG Online Weighted Q-Ensemble 46
Figure 3.9 Final Q Value Generation in DDPG Online Weighted Q-Ensemble 47
Figure 3.10 Online Weighted Q-Ensemble. 48

Figure 4.1 Illustration of the Inverted Pendulum Swing-up Model. 58
Figure 4.2 Illustration of the Cart Pole Model. 59
Figure 4.3 Illustration of the Cart Double Pole Model. 60
Figure 4.4 Illustration of the Half Cheetah v2 Model. 60
Figure 4.5 Illustration of the Swimmer v2 Model. 61

Figure 5.1 Alternately Persistent and Online Learning of *mostly good* Group in Inverted Pendulum Swing-up. 67
Figure 5.2 Alternately Persistent and Online Learning of *mostly good* Group in Cart Pole. 67
Figure 5.3 Alternately Persistent and Online Learning of *mostly good* Group in Half Cheetah v2. 67
Figure 5.4 Half Cheetah Performance for the *Good* Hyperparameterizations in Alternately Persistent. 68
Figure 5.5 Illustration of DC and DC-ED-DC Learning Curves of PD and CP environment. 71
Figure 5.6 Illustration of DC and DC-ED-DC Learning Curves of CDP and HC environment. 72
Figure 5.7 Performance of Weighted DDPG Ensemble Model. 74
Figure 5.8 Learning Curve of Online Weighted Q-Ensemble. 76
Figure 5.9 Counter Action and Q-Weights of Online Weighted Q-Ensemble. 77

List of tables

Table 4.1 Table with the Hyperparameters' Description used to Build each Algorithm of the Ensemble. 51
Table 4.2 Composition of Test Configurations – Classical Aggregations. 52
Table 4.3 Composition of Test Configurations – History-Based Framework. 52
Table 4.4 Ensemble Strategies Configuration of History-Based Framework. 54
Table 4.5 Variables of Inverted Pendulum Swing-up Training. 57
Table 4.6 Variables CP Training. 59
Table 4.7 Variables Cart Double Pole Training. 59
Table 4.8 Variables of Half Cheetah v2 Training. 61
Table 4.9 Variables of Swimmer v2 Training. 62

Table 5.1 Parameterizations Chosen by Environment and Their Performance in Action Ensemble. 64
Table 5.2 Parametrizations by Environment and Their Performance used in Value Ensemble. 64
Table 5.3 Comparison Performance of Classical Aggregations Ensemble in DDPG with Continuous Action Space in Alternately Persistent. 65
Table 5.4 Comparison Performance of Classical Aggregations Ensemble in DDPG with Continuous Action Space in Online Learning. 66
Table 5.5 Pendulum Results for Different Ensemble Strategies (31 runs). 69
Table 5.6 Cart Pole Results for Different Ensemble Strategies (31 runs). 69
Table 5.7 Cart Double Pole Results for Different Ensemble Strategies (31 runs). 69
Table 5.8 Half Cheetah v2 Validation Results (10 runs). 69
Table 5.9 Average Relative Regret Result – OWQE. 73
Table 5.10 Computational Effort Reference in Online Weighted Q-Ensemble. 79

Table A.1 Parametrizations by Environment and Their Performance in Action Ensemble. 87
Table A.2 Parametrizations by Environment and Their Performance used in Online Weighted Q-Ensemble. 88

List of Abreviations

ALE	— Arcade Learning Environment
B	— Best
CP	— Cart Pole
CDP	— Cart Double Pole
DB	— Density Based
DS	— Density Scores
DC	— Data Center
DCR	— Data Center Ranking
DDPG	— Deep Deterministic Policy Gradient
DDQN	— Double Q-Learning
DQN	— Deep Q-Learning
Deep RL	— Deep Reinforcement Learning
Double DQN	— Double Deep Q-Learning
DPG	— Deterministic Policy Gradient
ED	— Euclidean Distance
GRL	— Generic Reinforcement Learning Library
HBF	— History-Based Framework
HC	— Half Cheetah v2
M	— Mean
MA	— Moving Average
MDP	— Markov Decision Process
OU	— Ornstein-Uhlenbeck
OWQE	— Online Weighted Q-Ensemble
PD	— Inverted Pendulum Swing-up
RL	— Reinforcement Learning
SW	— Swimmer v2

List of Symbols

$A(s)$	The set of all actions available in state s.
$\bar{a}$	Average or central element of a group.
a	An action.
$\mathcal{B}$	The minibatch.
$\mathcal{D}$	The replay memory or replay buffer.
d_i	The density of each i action, a sum of Gaussian functions centered at the proposed actions.
e	The e^{th} ensemble.
ed_i	Squared Euclidean Distance of i^{th} action.
g	The g^{th} group of ensembles.
i	The i^{th} aggregation.
i, j, k	Indexes in a summation or within a group.
N	Number of ensembles, or number of actions of the ensemble, or number q-value of the ensemble.
$\mathcal{N}$	Noise.
$P(s' \mid s, a)$	Transition's probability to state s', from state s taking action a.
$p(s_t, a)$	The preference value of action selection in state s_t and action a.

Q, Q_t	The array estimates of action-value function Q^π.
$Q^\pi(s,a), Q(s,a)$	The value of taking action a in state s under policy π.
$\mathbf{Q}$	Q matrix.
q_{w}	The final Q-value vector q_{w}.
q_{w_j}	The final Q-value q_{w_j} related action a_j.
$\mathbb{R}$	The set of real numbers.
$\mathcal{R}$	The set of all possible rewards, a finite subset of $\mathbb{R}$.
r	A reward.
$r(a)$	The rank of an action a.
$\mathcal{R}_{\mathrm{HBF}}$	The Performance Measure in History-Based Framework.
$\mathcal{R}_{\mathrm{OWQE}}$	The Performance Measure in Online Weighted Q-Ensemble.
rs	The reward scale hyperparameter.
R_t	The return following time t.
S	The set of all nonterminal states.
s, s'	The states.
$T, T(t)$	The final time step of an episode, or of the episode including time step t.
t	Discrete time step or play number
V, V_t	The array estimates of state-value function V^π.
$V^\pi(s), V(s)$	The value of state s under policy π (expected return).
W	The weights vector in Online Weighted Q-Ensemble.
W_{raw}	The raw weights vector in Online Weighted Q-Ensemble.

y_i	The target of sample (s_i, a_i, r_i, s'_i) in a minibatch.
α	The step-size parameter (learning rate).
α_h	The step-size parameter (historical learning rate).
γ	The discount-rate parameter.
$\mu, \mu(s), \mu(s \mid \theta)$	The deterministic policy being learned.
π	The policy (decision-making rule).
$\pi(a \mid s), \pi(s, a)$	The probability of taking action a in state s under *stochastic* policy π.
$\pi(s)$	The action taken under the *deterministic* policy π.
π^*	The optimal control policy.
$\rho, \rho(s_t, a_t, s_{t+1})$	This is the reward (or expected reward) of a transition.
$\sigma(q_{ij})$	The Q-value q_{ij} normalized by softmax function σ.
$\sigma(s, a)$	The empirical standard deviation.
τ	The step-size parameter (soft target update rate).
θ, θ'	The deep network parameters.
υ, υ^β	A different (stochastic) behavior policy.

“Você não pode ensinar nada a ninguém, mas pode ajudar as pessoas a descobrirem por si mesmas.”

Galileu Galilei, *Revista Galileu.*

1
Introduction

This chapter aims to introduce the work presented in this thesis, presenting the motivation, the ensemble methods, the objective of the work, the contributions and, finally, the organization of the thesis chapters.

1.1
Motivation

Reinforcement learning (RL) [1] is an area of machine-learning that enables an agent to learn in an interactive environment by trial and error. Using feedback from its own actions and experience, it seeks to maximize rewards, optimizing its control policy and achieving the required goals in the process. RL is widely used for controlling environments in a model-free setting (without knowledge of how their environments will change in response to an action) by maximizing a reward signal in order to achieve a goal and it is based on a mathematical framework known as a Markov Decision Process (MDP), which describes how an environment reacts to an agent's control input. In order to scale RL techniques, Deep RL (based on deep neural networks) learns its own representations of environment and can thereby learn control policies for many common domains such as motor control [2], disease planning and pandemic prediction [3, 4].

Deep Reinforcement Learning requires several pre-defined hyperparameters, which are responsible for guiding the learning process through interaction with an environment. There are two types of parameters: Model Parameters are automatically estimated from the input data and saved with the final trained model; the Model Hyperparameters, on the other hand are manually defined in advance, and are used in the learning process to help in the estimation of the model's parameters. These hyperparameters are learning rates, discount factor, reward scale, neural network layer size, replay steps, replay memory size, minibatch size, activation function, among others. Since hyperparameters are not automatically trained or adjusted during training, an optimal learning result depends on manual fine-tuning.

The search for the best hyperparameters can be performed by grid search [5], an exhaustive search technique in which a specific value of a given hyperparameter in the model gradually varies, while all others are frozen. After this process is repeated for all hyperparameters, the maximum performance result is chosen, as well as their hyperparameter configuration. This procedure is computationally expensive because it requires many long sequential training runs. Hence, there are several other hyperparameters optimization methods; recent studies use bayesian optimization [6], bandit-based approach [7], genetic algorithm [8] and sequential decision [9]. Other methods for learning optimal parameters are population-based training (PBT) [10] and CERL [11], which use multiple learners as a population to train neural networks inspired by genetic algorithm, copying the best hyperparameters across the population. Tuning parameters is hard and some approaches, such as the population based ones, consider multiple parallel environments, which makes their real world use more complex.

In contrast to hyperparameter tuning for supervised learning, where the same data set can be re-used over and over, RL demands new environment data acquisition for each configuration, since it is necessary to run the control policy being optimized in the environment, which takes time. In addition to that, applying this process in the real world takes even longer, and it risks causing damage to the robot [12, 13]. Based on these assumptions and on the need to optimize the use of the amount of environment interactions, the goal is to learn from a single environment; in addition, the idea is to use ensemble in continuous action spaces, seeking to improve performance and robustness.

Faced with the challenges of finding the hyperparameters in reinforcement learning, and also with the requirement to make training more compatible with real world testing, this work aims to bring a new approach to the use of ensembles in Deep Reinforcement Learning. Another proposition is to investigate if it is possible to train sets of different sets of hyperparameters to reduce the hyperparameter tuning effort.

1.2 Ensemble Methods

Ensemble learning is a general approach to machine learning that seeks better performance by combining predictions from multiple models. Although there are a seemingly unlimited number of ensembles that can be developed for a predictive modeling problem, it is widely found in the literature, for example: tree ensembles using mean predictions; use of different algorithms using the same data set acquisition; and ensembles that use weighted mean predictions.

Ensembles were first used in RL before the advent of deep learning techniques [14, 15, 16] to increase performance, and recent efforts have shown that ensemble aggregations of deep neural networks perform better than a single algorithm as well [17]. There are two types of ensembles in RL; the Action Aggregation, which performs aggregation of agents' actions, and Value Aggregation, that performs aggregation of value functions.

Studies of Action Aggregation in RL started with majority vote decision [14, 15] and average decision [18], with some studies using Deep Q-Learning (DQN) [19] and Deep Deterministic Policy Gradient (DDPG) [17] algorithms applying Deep RL. In Value Aggregation, average function value [15] and majority vote decision [20] are used.

While ensembles in Deep RL demonstrated good results when used to increase performance, there have been a few studies approaching the RL ensemble behavior when using different hyperparameters, aimed at reducing the tuning effort [5]. This thesis, therefore, aims to investigate whether ensembles of different sets of hyperparameters can be trained to decrease the hyperparameter tuning effort [5, 21, 22].

1.3 Objectives

This work aims to employ the ensemble, a traditional method used for performance improvement, to research its use in minimizing the fine-tuning efforts of the underlying algorithm's hyperparameters, as this tuning process is very costly and time-consuming.

Initially, the performance of existing ensemble strategies in the specific configurations of reducing the effort of hyperparameter tuning for deep reinforcement learning should be executed as a baseline definition. Once the performance of classical ensemble strategies in continuous action spaces has been characterized, how those strategies behave in different environments and with different hyperparameter sets should be investigated. Specifically, we aim to investigate the use of ensembles containing algorithms that did not converge nor achieve good performance, as that will certainly be the case when unknown sets of hyperparameters are combined.

Next, new ensemble algorithms should be proposed in both types of ensemble models used in RL. For Action Aggregation, we propose a History-Based Framework (HBF) which focuses on achieving an action aggregation strategy with minimum hyperparameter tuning effort. The framework seeks to improve performance in various environments even with an ensemble that contains non-converging algorithms. The performance of the framework in comparison to the baselines in order to verify the performance in these cases is enhanced.

Given the possibility of using the information from the value function to choose actions, a value aggregation strategy in development with the same goal of reducing hyperparameter tuning effort. We propose Online Weighted Q-Ensemble (OWQE), a weighing approach that adjusts the critics' weights parameters by minimizing the temporal difference error of the ensemble. Again, its performance in comparison to the baselines.

1.4 Contributions

The research presented the following contributions:

- A study of classical aggregation ensembles in reinforcement learning with Deep Deterministic Policy Gradient and online learning [5];
- An investigation of how performance-enhancing algorithms address the new purpose of reducing tuning effort of hyperparameters;
- A method of evaluating the ensemble of algorithms and groups to measure the proposed aggregation techniques. To measure the sensitivity to the hyperparameter settings, groups that mix different amounts of good and bad DDPG parameterizations;
- The development of the History-Based Framework the first study to seek optimized learning techniques for deep ensemble reinforcement as a way to reduce the hyperparameter tuning effort with differently parameterized algorithms;
- The creation of the History-Based Framework (HBF) implementation for continuous actions to allow multiple aggregation compositions, in order to use the configuration that best suits the environment [21]. The DDPG implementation with differently parameterized settings are trained online in the simulated rigid body dynamics named as MuJoCo environment [23] in the OpenAI Gym toolkit [24];
- The HBF's contributions lie in the maintenance of a temporal moving average of policy scores and selecting the actions of the best scoring policies;
- The creation of Online Weighted Q-Ensemble (OWQE) as a value aggregation strategy to use a weighing approach that adjusts the critics' weights by minimizing the temporal difference error of the ensemble; and
- The use of classic control environments to evaluate the ensembles and the validation of the presented distinct ensemble approaches with more complex environments such as Half Cheetah v2, and Swimmer v2 of the MuJoCo environment [23] in the OpenAI Gym toolkit [24].

1.5 Organization

This dissertation is organized in 6 chapters, organized as follows:

Chapter 2: describes the reinforcement learning algorithms and background on the RL ensemble aggregation.

Chapter 3: explains the methodology developed for the analysis of the hyperparameters fine-tuning reduction effort, while also presenting the newly created ensemble methods: the action and the value aggregation strategies, History-Based Frameworkfor Action Ensemble and the Online Weighted Q-Ensemble.

Chapter 4: describes the configurations used in the experiments, the hyperparameters, and also the groupings performed to assemble the ensembles.

Chapter 5: presents the results from the experimentation structure described in Chapter 4, and provides analysis and discussions of each methodology as well.

Chapter 6: presents conclusions achieved after observing the results of the experiments, and points out where the solutions of this work can be applied in future work.

2
Ensemble Reinforcement Learning

This chapter summarizes related work on Ensemble RL. First, general reinforcement learning algorithms will be described in order to present the theory and the step-by-step evolution of the algorithms until arriving at the description of the algorithm used in the experiments, Deep Deterministic Policy Gradient (DDPG). Subsequently, the related work of Ensemble RL will be presented, as well as the conventional continuous action ensemble aggregation strategies.

2.1
Reinforcement Learning (RL)

Reinforcement Learning (RL) [1] is widely used for controlling environments in a model-free setting in order to maximize a reward signal for achieving a goal. RL is a third machine learning paradigm, in addition to supervised learning, where the learned data is previously classified by labels, or unsupervised learning, where characteristics are extracted from unlabeled data. In RL, the agent seeks to achieve a goal from interaction with an environment. The samples of this interaction are partially labeled by the reward signal. Partially, because the rewards only give short-term feedback while the objective is to optimize the sum of these rewards.

The active decision-making agent learns by interacting with its environment, where the actions are picked using a mapping from situations to actions. This situations are numerically captured through sensors and called as state. After the agent runs an action, a numerical reward signal is assigned and used to update the resulted next state.

RL is based on a mathematical framework known as Markov Decision Process (MDP) which describes how an environment reacts to an agent's control input. A Markov Decision Process is a 4-tuple $\langle S, A, P, \rho \rangle$, where S is a set of states called the state space, A is a set of actions called the action space, $P(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the transition function (the probability that action $a \in A$ in state $s \in S$ at time t will lead to state $s' \in S$ at time $t + 1$, or the dynamics) and, $\rho(s_t, a_t, s_{t+1}) \in \mathcal{R}$ is the reward (or expected reward) of such a transition. The agent receives a reward at each time step, after transitioning from state s to state s', due to action a. The state and action spaces may be finite or infinite.

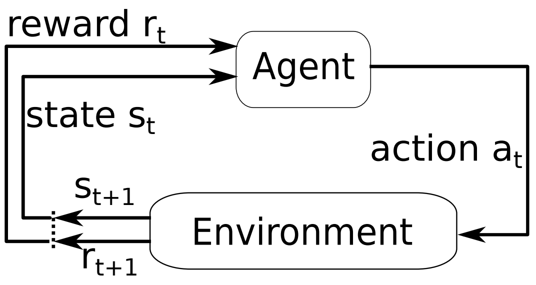

Figure 2.1: Reinforcement Learning.

Figure 2.1 illustrates how the interaction functions in RL to solve sequential decision problems. An active decision-making agent interacts at discrete time steps and a certain state $s_t \in S$, taking an action $a_t \in A$ which results in a reward $r_{t+1} \in \mathcal{R}$ and a next state s_{t+1}. The objective of the RL is to accumulate the maximum expected return (sum of rewards, see Eq. (2-1)), while it interacts with the environment by updating the policy π, which represents a mapping from states to action probabilities used by the agent. The policy $\pi(a|s)$ represents the probability of taking action a in state s.

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T \tag{2-1}$$

The maximum expected return $\mathbb{E}\{R_t\}$ has two different definitions, one of them for episodic tasks, which are defined by tasks with a finite amount of time T, such as playing a game; in that case, Eq. (2-1) represents the return. The other definition is related to continuing tasks, which do not have a naturally episodic definition, in that case a discounted factor ($\gamma \in [0,1]$) is used to trade off the importance of immediate and later rewards in order to avoid infinite sums, Eq. (2-2).

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2-2}$$

Since the agent's actions, that follow π, affect the future states of the environment, the expected return starting from a certain state s depends on the policy. This is captured in the value function $V^\pi(s)$, Eq. (2-3). Another value function, $Q^\pi(s,a)$, considers the expected return starting in state s and taking a as the first action, and only then following π, Eq. (2-4).

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left\{R_t|s_t = s\right\} \qquad = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} \tag{2-3}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left\{R_t|s_t = s, a_t = a\right\} \quad = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} \tag{2-4}$$

Algorithms that use Eq. (2-3) or (2-4), for example, can therefore reason about the possible utility of taking different actions in the same state, in general choosing the one that maximizes the return:

$$\pi(a \mid s) = \begin{cases} 1, & \text{if } a = \arg\max_{a'} Q(s,a') \\ 0, & \text{otherwise} \end{cases} \tag{2-5}$$

2.1.1 Q-Learning Algorithm

Q-Learning is the most iconic classical reinforcement learning algorithm. It is known as off-policy, because it learns the value function (expected return) of the optimal control policy π^*, or the target policy, while following some other (exploratory) behavior policy υ. that is required for convergence of the algorithm to the optimal solution; otherwise, it would get stuck in a local maximum.

Q-Learning is an iterative algorithm that in every control step updates the state-action values by Eq. (2-6) using a temporal difference error that measures the difference between the expected value given the next state $r' + \gamma \max_{a'} Q(s', a')$ and the current estimate $Q(s, a)$.

Q-Learning uses a learning rate $\alpha \in [0, 1]$ which weighs the relative importance of old and new estimates, effectively implementing an exponentially weighted low-pass filter.

$$\begin{aligned} Q(s,a) &= (1-\alpha)Q(s,a) + \alpha\left[r' + \gamma\max_{a'} Q(s',a')\right] \\ &= Q(s,a) + \alpha\left[r' + \gamma\max_{a'} Q(s',a') - Q(s,a)\right] \end{aligned} \tag{2-6}$$

Real world tasks tend to be continuous and normally are too large to learn all states and its possible actions. To tackle these problems, a parameterized value function $Q(s, a; \theta_t)$ can be learned using a linear or nonlinear function approximator, rewriting the Eq. (2-6) into Eq. (2-7), where the parameters are updated towards a target value y, using stochastic gradient descent on the difference between the target and the current value $Q(s, a; \theta)$.

$$\begin{aligned} y &= r + \gamma\max_{a} Q(s',a;\theta) \\ \theta' &= \theta + \alpha\left[y - Q(s,a;\theta)\right]\nabla_{\theta}Q(s,a;\theta) \end{aligned} \tag{2-7}$$

Classically, the value function is represented by a linear-in-parameters approximator, since it provides convergence guarantees. The use of a neural network to represent the parameterized value function may make the algorithm unstable. Specifically, since the targets are always changing, the value function may diverge to infinity.

2.2 Deep Reinforcement Learning (Deep RL)

Reinforcement Learning (RL) demands a lot of interaction with the environment to solve high-dimensional continuous state-space problems. Conventional function approximators do not have the ability to learn the representations required to effectively abstract such spaces, requiring careful manual feature extraction. To become more efficient, most research in the last few years has focused on the combination of RL and deep learning techniques.

Deep learning is widely known for being able to automatically learn representations from data, thus avoiding the manual (and error-prone) feature extraction while simultaneously reducing the required interaction time. However, several adjustments had to be made in order to avoid the general instability of neural networks in a reinforcement learning setting [25].

2.2.1 Deep Q-Learning

Deep Q-Learning (DQN) [25] was the first combination of Deep Learning and the Q-Learning algorithm. It was applied in the Arcade Learning Environment (ALE) using discrete actions. ALE is a framework designed to allow reinforcement learning agents to play arbitrary Atari 2600 games, with over 500 original games played in a single game screen of 210x160 pixels with 128-color palette and a maximum of 18 action inputs to the game via a digital joystick.

DQN uses a replay memory ($\mathcal{D}$) in each iteration, which stores samples obtained from the environment during training. This allows them to be continually re-used for value function updates, a process called experience replay. Consecutive state transitions have a strong correlation, so sampling from the memory is randomized in order to reduce the variance of the updates. The advantage of sampling non-correlated data from the model is to avoid "forgetting" the data.

In addition, DQN also uses a target network, which is a copy of the main network, that is used to calculate the target values in Eq. (2-7). Its parameters θ' are copied from the main network parameters θ every τ steps; otherwise, the parameters values are frozen. DQN uses this new target shown in Eq. (2-8), which stabilizes the learning because the targets change more slowly, reducing divergence.

The main network is updated as in Q-learning, but using standard deep learning techniques such as ADAM [26] instead of simple gradient descent. The loss function is given in Eq. (2-9), where the expectation is taken over a minibatch $\mathcal{B} \subset \mathcal{D}$ of experienced transitions. The sample y_i, used in $\mathbb{E}_{(s,a,r,s')}$, have the sample $i \in \mathcal{B}$, where $(s, a, r, s') \in i$.

$$y_i = r + \gamma \max_{a'} Q(s', a'; \theta') \tag{2-8}$$

$$L(\theta^Q) = \mathbb{E}_{(s,a,r,s')} \left[(y_i - Q(s, a; \theta))^2 \right] \tag{2-9}$$

2.2.2 Deep Deterministic Policy Gradient (DDPG)

DDPG is an actor-critic algorithm applicable in high-dimensional continuous action tasks [27]. It is an approach based on the off-policy Deterministic Policy Gradient (DPG) algorithm [28]. The critic $Q(s, a)$ learns to approximate the Bellman Equation as in deep Q-learning [25]. DDPG optimizes the critic by minimizing the loss (Eq. (2-10) and (2-11)), where the function approximator is parameterized by θ^Q. The s and a is the state and action in the transition, r is reward and s' represents the next state.

The main network also is updated as in Q-learning, but using standard deep learning techniques such as ADAM [26] instead of simple gradient descent. The loss function is given in Eq. (2-10), where the expectation is taken over a minibatch $\mathcal{B} \subset \mathcal{D}$ of experienced transitions. The sample y_i is applied in target network $Q(\cdot|\theta^{Q'})$ and used in $\mathbb{E}_{(s,a,r,s')}$, have the sample $i \in \mathcal{B}$, where $(s, a, r, s') \in i$.

$$L(\theta^Q) = E_{(s,a,r,s'))} \left[\left(y_t - Q(s, a|\theta^Q) \right)^2 \right] \tag{2-10}$$

where

$$y_t = r_t + \gamma Q(s', \mu(s')|\theta^{Q'}) \tag{2-11}$$

The Bellman error of the value function of the target policy is the performance objective, averaged over the state distribution of the behavior policy. It is learned from a parameterized actor function $\mu(s|\theta^\mu)$, which deterministically maps states to a specific action. Equations (2-12) and (2-13) presents the actor update, which takes a step in the positive gradient criteria of the critic, with respect to the actor parameters. Applying the chain rule, J

represents the expected return from the start distribution and υ^{β} a different (stochastic) behavior policy.

$$J = \mathbb{E}_{s_t \sim \upsilon^{\beta}} \left[Q(s,a)|_{s=s_t, a=\mu(s_t)} \right] \tag{2-12}$$

$$\begin{aligned} \nabla_{\theta_\mu} J &= \mathbb{E}_{s_t \sim \upsilon^{\beta}} \left[\nabla_{\theta_\mu} Q(s,a)|_{s=s_t, a=\mu(s_t)} \right] \\ &= \mathbb{E}_{s_t \sim \upsilon^{\beta}} \left[\nabla_a Q(s,a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^{\mu}|_{s=s_t}) \right] \end{aligned} \tag{2-13}$$

Taking advantage of the off-policy data, experiments are stored for use as a Replay Buffer, inducing sample independence. The samples s, a, r, s' are generated by exploring sequentially the environment and storing them in the replay buffer ($\mathcal{D}$). Every time step, a minibatch is sampled uniformly from the replay buffer, updating the actor and critic. The buffer size is big enough to avoid overfiting [29]. Figure 2.2 illustrates this process.

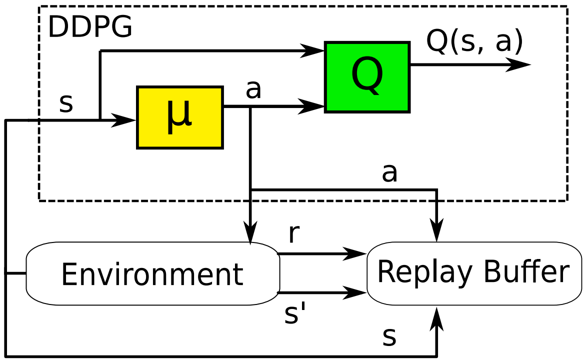

Figure 2.2: Deep Deterministic Policy Gradient (DDPG) algorithm [27], its interaction with the environment and the storage of transactions in the Replay Buffer.

Note that y_t in Eq (2-11) depends on $\theta^{Q'}$, a target network used to stabilize the network learning with a time delay. DDPG introduces a "soft" target update, so the target network actor and critic network weights update is performed slowly. The slow update is done using a exponentially decaying average as in Eq. (2-14). This technique stabilizes the problem of learning the action-value function and brings the problem closer to the case of supervised learning [27].

$$\theta'^{(t)} = \tau \theta'^{(t-1)} + (1-\tau)\theta'^{(t)} \text{ with } \tau \in [0,1] \tag{2-14}$$

The exploration in continuous action spaces is a challenge in RL. A noise $\mathcal{N}$ is added to sample the policy $\pi = \mu(s_t|\theta_t^{\mu}) + \mathcal{N}$, using the Ornstein-Uhlenbeck process [30] to generate temporally correlated exploration for exploration efficiency in physical environments that have momentum ($\theta_{\mathcal{N}} = 0.15$ and $\sigma_{\mathcal{N}} = 1$). The Ornstein-Uhlenbeck process models the velocity of a

Brownian particle with friction, which results in temporally correlated values centered around zero [27]. Below is a presentation of some related works using ensembles in both discrete and continuous action spaces, applying either action or value ensembles.

2.3 Ensembles in Discrete Action Spaces

In discrete action spaces, the individual algorithms in the ensemble can be combined by looking only at a limited available action set.

2.3.1 Action Ensembles in Discrete Action Spaces

For discrete action ensembles, each algorithm outputs a list of action preferences, which are then combined through methods such as majority voting, rank voting, Boltzmann multiplication, and Boltzmann addition [14]. Its efficiency was shown in maze problems of different complexities. The aggregations were tested with several classical RL algorithms, using both tabular representations and neural network function approximations.

2.3.1.1 Majority Voting

In this method, each algorithm votes for a single action, the one with the highest preference. The discrete action that received the most votes is the chosen action. The possible actions for each state are $A(s_t) = a_1, \cdots, a_m$. Figure 2.3 shows an ensemble composed by 3 RL agents: two of them choose the action a_2 while one chooses the action a_1, so the Majority Voting method chooses the action a_2. If two or more actions tie in the choice, the action will be randomly chosen among them.

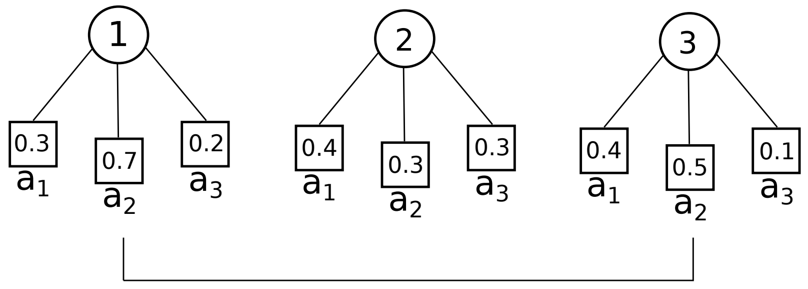

Figure 2.3: Majority Voting.

2.3.1.2
Rank Voting

In rank voting, action selection preferences are not used directly but instead ranked and weighted.

The action selection policy of this algorithm is π_t^i, where $\pi_t^i = \pi_t^i(s_t, a_t)$. Eq. (2-15) presents the sum of rankings as a means of choosing the action of the ensemble. The rank is $r_t^i(a[j]) \geq r_t^i(a[k])$ if $\pi_t^i(a[j]) \geq \pi_t^i(a[k])$. An example of how the ranking can be assembled is that the most preferred action is associated with the highest weight m, the second most preferred one with the weight m-1, and this goes down to the least preferred action with weight 1.

$$p(s_t, a[i]) = \sum_i r_t^i(a[i]) \tag{2-15}$$

2.3.1.3
Boltzmann Multiplication

This method seeks for a selection of actions by multiplying all the action selection preferences of the ensemble as in Eq. (2-16), thereby treating the preferences as probabilities. The highest value of $p(s_t, a)$ indicates the action to be selected, in case of a tie the action is chosen at random.

$$p(s_t, a[i]) = \prod_j \pi_t^j(s_t, (a[i]) \tag{2-16}$$

2.3.1.4
Boltzmann Addition

Similar to the Rank Voting method, it sums all action selection preferences of the ensemble. Note that, this method is a specificity of Rank Voting, in which $r_t^j = \pi_t^j$, presented in Eq. (2-17).

$$p(s_t, a[i]) = \sum_j \pi_t^j(s_t, (a[i]) \tag{2-17}$$

2.3.2
Q-Ensembles in Discrete Action Spaces

In Q-ensembles, multiple value functions are learned over a single shared action set. These are then combined to make a common decision about which action is more preferable. As such, instead of considering only action preferences such as done in action ensembles, Q-ensembles consider expected returns as a basis for ensemble aggregation.

One of the earliest uses of Q-ensembles was the combination of multiple regression trees into a random forest [31]. In this Fitted Q Iteration (FQI) approach, the reinforcement learning problem was reformulated as a sequence of standard supervised learning problems, where themselves were solved using an ensemble method. FQI and a variant that uses a neural network instead of a random forest (Neural Fitted Q Iteration, NFQ) were precursors to DQN, and in fact use the same loss function.

The most popular Q-ensemble method (used in FQI) is Q-average aggregation, where the individual Q values are simply averaged. Neural networks ensembles [15] used Neural Fitted Q-iteration and Q-averages aggregation in the pole balancing environment. Comparing majority voting action aggregation and the Q-averaging value aggregation, the former improved performance, plus the ensembles used different network topologies. A predicted state-values $V(s)$ uses average value state ensemble and compared with majority voting [18]. The tests were performed in 20x20 mazes (5 agents) and tic-tac-toe (3 agents). The ensemble cases performed better than the ones using a single algorithm, but no clear advantage of action or state-value ensembles was found.

The Ensemble DQN is presented [19] solving in parallel K DQN losses, with the average being the Q-value ensemble, Q^E, as proposed in Eq. (2-18).

$$Q^E(s,a) = \frac{1}{K}\sum_{i=1}^{K} Q(s,a;\theta^i) \tag{2-18}$$

Different discount factors were employed in the Q-learning ensemble modules [32]; in this scenario, the policy episode is selected through the ϵ-Greedy selection strategy, due to maximization of the average reward. Multiple discounting reinforcement learning brings an interesting study of optimality with a tree MDP and a grid world MDP. The ensemble is composed by Q-learning algorithms with different γ, which varies between $(0.01, 0.99)$. The tests use 99 γ-modules [32]. The ϵ-Greedy selection presents a decay over episodes. In a performance test, it was found that an ensemble of 12 (MDP tree) and 6 (grid world) algorithms was needed to achieve the highest average.

Inspired by Bayesian reinforcement learning, upper confidence bounds (UCB) exploration via Q-ensemble [20] was tested using Arcade Learning Environment Atari games, where using the UCB presents a slight improvement. The average of the normalized learning curve of all games for each algorithm was used and compared with the ensemble majority vote and Double DQN [33]. The calculation of the UCB is based on $\{Q_K\}$ outputs, so the empirical standard deviation (σ) and empirical mean(μ) of $\{Q_k(s_t,a)\}_{k=1}^{10}$ are used to choose the action. Besides that, another hyperparameter, λ, is necessary for tuning the algorithm, Eq. (2-19). The UCB is composed by an empirical standard deviation $\sigma(s_t,a)$ of $\{Q_k(s_t,a)\}_{k=1}^{10}$ and an empirical mean $\mu(s_t,a)$ of $\{Q_k(s_t,a)\}_{k=1}^{10}$; the λ should maximize this UCB, which results agent's chosen action.

$$a_t \in \arg\max_a \{\mu(s_t, a) + \lambda\sigma(s_t, a)\} \tag{2-19}$$

Another algorithm that uses Atari suite, in addition to performing validation in an n-state MDP environment is called Enhancing Reinforcement in Q-Ensembles. The share-in-learning knowledge update [34] considers the estimate of the best Q value. The loss function is presented in Eq. (2-20). The result showed improvement in the game score for some of the six Atari Games indicating faster learning through robust estimates; however, the quantity of algorithms used in the simulations was not presented, only the necessity to tune this number was highlighted.

$$\begin{aligned} L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma Q_k(s', \arg\max_{a'} Q_{\text{best}}(s', a'; \theta_i); \theta_i^-) \\ - Q_k(s, a; \theta_i))^2] \end{aligned} \tag{2-20}$$

Recently, Distributional Reinforcement Learning uses an ensemble of i DQN agents, each independently trained but with the same architecture and hyperparameters, running in different Atari 2600 Game environment [35]. The ensemble selects its action by a type of average of the estimated Q-values of each agent. Instead of directly estimating Q-values, it uses a finite support $\mathbf{z}$ of 51 points and learns discrete probability distributions $\phi(x, a; \theta)$ over $\mathbf{z}$ via softmax transfer. The empirical study uses a subset of Atari 2600 games, the method showed improvement when compared to the single run.

2.4 Ensembles in Continuous Action Spaces

In continuous action spaces, the individual algorithms do not share a limited action set over which statistics can be calculated; instead, actions need to be combined in a continuous space.

2.4.1 Action Ensembles in Continuous Action Spaces

In continuous action ensembles, binning, data center, and density-based [16] were presented using the cart-pole swing-up benchmark as well as a gas turbine simulation. The density-based aggregation is an interpretation of majority voting for the continuous action spaces. This variant showed a better performance result using the policy gradient neural rewards regression algorithm.

Seeking a sample efficient way [36] of training any aggregation method chosen from the literature into one environment, a cycling learning rate with defined maximum and minimum learning rate values was able to converge. Figure 2.4 shows the changes in the learning rate. When the learning rate returns to the beginning of the cycle (eg: 0.01), there is a drop in performance, even though the learning curve shows more significant variance when the learning rate resets.

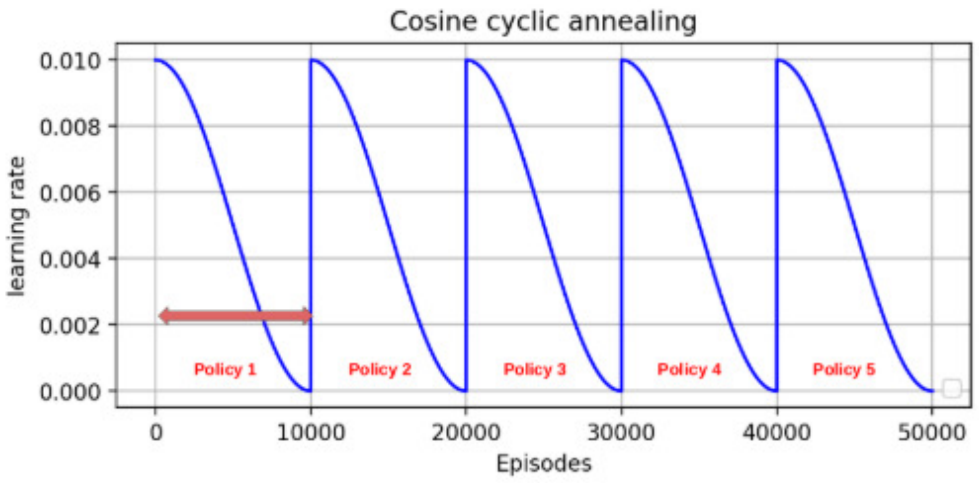

Figure 2.4: Cycling Leaning Rate. Source: [36]

The Bootstrapped aggregated multi-DDPG [17] (BAMDDPG), is structured with Multiple DDPG networks for optimizing controllers (policies) in continuous actions spaces, using three DDPG networks, each one with a separate simulation environment. The limitation of this approach is the difficulty in carrying out training in real world, which is a single environment, with minimal intervention. Although BAMDDPG also considered alternating the training of the different policies in the same environment, the performance behavior was not made explicit, as the presented results uses multiple environments; therefore, the mean action aggregation was used. Tests carried out with 3 and up to 5 sub-policies demonstrated promising results; however, the ones with more than 10 sub-policies showed degradation in the cumulative reward.

Recently, a History-Based Framework for online continuous action ensembles in Deep RL optimized the technique of continuous action ensembles to improve performance and robustness as well as to avoid parallel environments, in order to make the system applicable to real-world robotic applications [5, 21]. In this study, ensembles were used for the first time to decrease the adjustment effort of the algorithms' hyperparameters.

Below is introduced the ensemble aggregation of pre-learned policies [16], in addition to the simple mean aggregation, these three techniques form a base case in experiments.

2.4.1.1 Mean

Mean aggregation simply returns the arithmetic mean of the ensemble actions, or the sum of all values divided by the total number of values. If we are dealing primarily with well-tuned hyperparameters algorithms, this should lead to a good ensemble action.

2.4.1.2 Density Based

Density-based aggregation was created as an extension of majority voting to continuous actions spaces. The density of each action d_i is a sum of Gaussian functions centered at the proposed actions, as shown in Figure 2.5. The final chosen action is the one with the highest density in the action space.

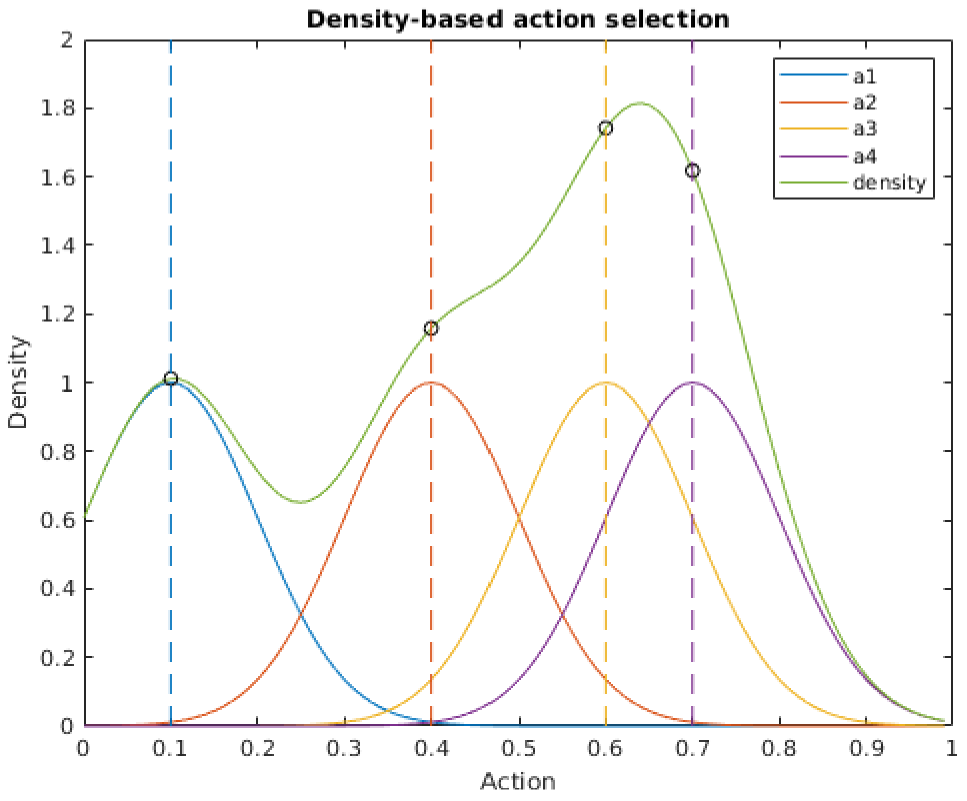

Figure 2.5: Illustration of Density Based Aggregation.

Equation (2-21) calculates the action density, with the exponent being the sum of the difference between the N actions of the ensemble and the actual action a_i, considering a m-dimensional action vector. Contemplating an environment with more than 1 action, the different dimensions in the action space will usually not have same scale, and as a result they are normalized for the interval $[-1, 1]$. The distance (basis function width) parameter r must be chosen empirically. As such, this is not a parameter-free algorithm.

$$d_i = \sum_{j=1}^{N} e^{\frac{-\sum_{l=1}^{m}\left(a_{il}-a_{jl}\right)^2}{r^2}} \tag{2-21}$$

2.4.1.3 Data Center

As opposed to Density Based aggregation, data center aggregation is a parameter-free algorithm. In the first step, the Euclidean distance is calculated for each action with respect to the mean of the action ensemble. The action with the longest distance is removed and the procedure is repeated until only two actions remain. Finally, the average (Mean Aggregation) of the last two actions is the chosen one, as seen in Algorithm 1.

Algorithm 1: Datacenter Pseudocode.

```
while $|\mathcal{A}| > 2$ do
    $\mu = (\sum a_i)/|\mathcal{A}|$
    for each element $a_i$ in $|\mathcal{A}|$ do
        $d_i = ||a_i - \mu||$
    end
    j = $\arg\max_i d_i$
    $\mathcal{A} \leftarrow \mathcal{A} \setminus \{a_j\}$
end
$data\_center = (\sum a_i)/2$
```

2.4.2 Q-Ensembles in Continuous Action Spaces

In continuous action Q-ensembles, the chosen actions consider the value function, which must be estimated as part of some actor-critic method. In the area of Deep RL, a DDPG actor ensemble has been used to choose a good action applying an average critic ensemble, which means the outputs of the critic networks are combined by taking average [37]. It is noteworthy that each critic's output is calculated using each action. Equation (2-18) introduces the Q-Ensemble, but instead of using the predefined actions set, here we use the actions of the actors ensemble.

A good performance with Q-ensembles was achieved with 10 DDPG instances on bipedal walking, resulting in increased learning speed and fewer falls. On the other hand, an approach using a confidence vector ($c_t^i(s_t, r_t) \in (0, 1]$) was used employing the Q-ensemble; however, instead of using a DDPG ensemble, an approach inspired on DDPG architecture with several critical heads was taken [38]. Both works showed significant improvement compared to the single run.

2.5 Policy Selection in Training Phase

In order to perform ensemble aggregation, the individual algorithms need to be trained. This can be done in many ways, but the most popular ones are pre-training, alternate training, and parallel environments.

These training modes reflect how the policy will be chosen in the ensemble, leading or not to a better convergence of the learning curve. Considering the use of a single environment in RL, it is important to pursue the self-adaptation feature of the algorithm while learning the environment. After the training phase, during the inference phase, the algorithm will always choose the aggregated action of the ensemble.

2.5.1 Pre-training

Pre-training often appears in ensembles composed of algorithms prior to the Deep RL ensemble. All algorithms are previously trained for later use in the ensemble, which demands a large computational effort. During action ensemble selection there is no more training/learning of the algorithms, even if there is training or tuning of the parameters associated with the ensemble's functioning.

2.5.2 Alternately Persistent

Comparing to pre-trained, the alternately persistent should have efficient data learning due to the use of shared replay memory. In recent literature cases, when a single environment is used, [36, 17] the training phase applies the Alternately Persistent mode, where at each training episode, a single policy is selected to be executed in a round-robin fashion. At each step of the evaluation, the action aggregation technique is applied.

2.5.3 Parallel Environment

Another approach used is the one with multiple environments being trained independently and in parallel [17]. Thus, there is no concern about selecting different policies during training since each algorithm will have the agent's policy and its environment. Each trained policy will participate, in the validation or testing phase, in the ensemble's aggregation algorithm, in order to select the ensemble's action.

3
Ensemble Aggregation Methods

In this chapter, Section 3.1 starts out by describing a new method for policy selection in the training phase. Section 3.2 proposes the application of known continuous action aggregation strategies from classical literature to DDPG ensembles with the intent to reduce the hyperparameter fine-tuning effort. Section 3.3 proposes a new history-based framework to improve the performance of the classical aggregation strategies. This research uses only the actions generated by the actors. Finally, in Section 3.4, a further analysis of ensembles to fine-tuning the hyperparameters is done which also takes the critics' output into account.

3.1
Online Training

The selection policies described in Section 2.5 consider that the ensemble algorithms are fine-tuned and focus on increasing the performance of the ensemble. As the objective of this work involves joining non-converging and converging algorithms while not using multiple environments, it was necessary to use another method of policy selection during training. Using Alternately Persistent training, for example, allocates the same amount of time for good and bad policy developing algorithms. As such, even algorithms that will never learn a good policy are allowed to act, which may damage to the robot or environment when applied to real world.

In our proposed Online Training mode, we perform ensemble action selection throughout the training, so that the ensemble can select the policy actions that are most promising. At each step in every training episode, the action aggregation technique is applied, as the idea is to take advantage of a time-related learning action once all DDPG are learning together.

3.2
Classical Aggregations in DDPG with Continuous Action Spaces

To establish a baseline, we first considered action aggregation strategies that have already been developed and successfully used in literature, applying them to DDPG ensembles with the intent to verify the behavior of these aggregations when used with ensembles that have different hyperparameter settings.

Figure 3.1 illustrates the classical ensemble aggregation strategy, where the *(1) Ensemble Aggregation* uses the aggregations methods described in Section 2.4.1, receiving the *(A) Action Ensemble* and the output *(F) Final Action*. The tested aggregation methods are Mean aggregation (Section 2.4.1.1), Density Based (Section 2.4.1.2) and Data Center (Section 2.4.1.3).

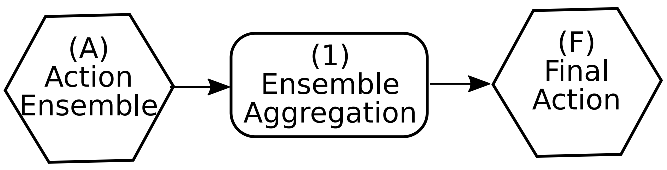

Figure 3.1: Classical Ensemble Aggregation Strategy.

3.2.1 Evaluating Ensemble Performance

Firstly, it is interesting to compare the simulations considering the policy selection modes during training. As the focus is on using a single environment, two selection modes are used, Alternately Persistent (Section 2.5.2) and Online Training (Section 3.1). This policy selection mode comparison is essential to the analysis of how much it influences the chosen ensemble group and final aggregation performance, as well as whether there is variability across environments.

Furthermore, there is no comparative methodology in the literature for ensembles that minimize hyperparameter tuning effort. The main difference is that for the normal intent of performance improvement, all individual algorithms can be expected to perform at least reasonably well, while reducing hyperparameter tuning some will have very bad performance. Therefore, it was necessary to create a comparison methodology that, in addition to comparing the ensembles, also uses the *Best Single* as a performance reference. For this, some single algorithms were generated and evaluated in each test environment, with their performances being ordered. The best and worst ones were used in the ensemble composition, with the best performance of the environment being used as benchmark for comparison against *Best Single*, used as continuous action aggregation baselines.

For this, 3 ensemble groups were built. The first one demonstrates the performance of aggregations according to the more traditional use of ensemble techniques, and the other two seek to build ensembles with more and fewer non-converging algorithms, to find out how aggregations manage the best action policy choice.

As such, the three groups (*3 best*, *good* and *bad*) were created. Section 4 organizes the details of the experimental settings.

3.3 History-Based Framework (HBF)

Ordinarily, ensemble aggregation considers pre-learned policies, and this study aims to do it with an online training mode. Given that the policies are learned, simple aggregation of actions is not expected to perform well. Assuming that the policy performance is time-correlated, the proposed framework uses the historical performance to make its decisions; therefore, it seeks to take advantage of mutual learning while using the moving average historical performance. As such, policies with bad performance in the past can be avoided. To accomplish this, the framework introduces a *scoring function* to measure the individual policies' performances.

The ensemble framework was created to select the best ensemble action. Figure 3.2 presents a generic view of HBF. First, a *(G.1) Scoring* function running the *(G.A) Action Ensemble* to assign weights, represented by *(G.B) Scores*, referring to the quality of each action used in the aggregation function. Figure 3.3 presents an example of ordering the action weights (c_2, c_1, c_3). Those *(G.B) Scores* serve as inputs to the *(G.2) Moving Average*, which will generate the HBF's *(G.C)* $\overline{Scores}$. The main idea is to take advantage of the learning process iteration to pursue better policies.

The *(G.C)* $\overline{Scores}$ is a variable that, throughout the learning process, stores a qualitative weighting of actions. This variable, or memory structure, aims to create a measure that qualifies those policies generating better actions than others. This is important for the ensemble, as it means avoiding the selection of policy actions that do not have a history of selecting good actions.

Figure 3.4 presents the *(G.B) Scores* delivery to the *(G.2) Moving Average* function. The intention of Moving Average step is to reduce the fast switching in the algorithm's agents' choice, noticed empirically, enabled the learning of the ensemble. Although considering that the ensemble can make a better trade-off between exploration and exploitation [14], it has been found empirically the difficulty for the ensemble aggregations to maintain consistency when simply selecting the action with the highest-scoring in the ensemble. Thus, the importance of creating a history of the scores was realized, favoring recent actions and reducing the action chattering between the algorithms.

In this comparison, it is interesting to note that in Figure 3.4 the score $\bar{c_2}$

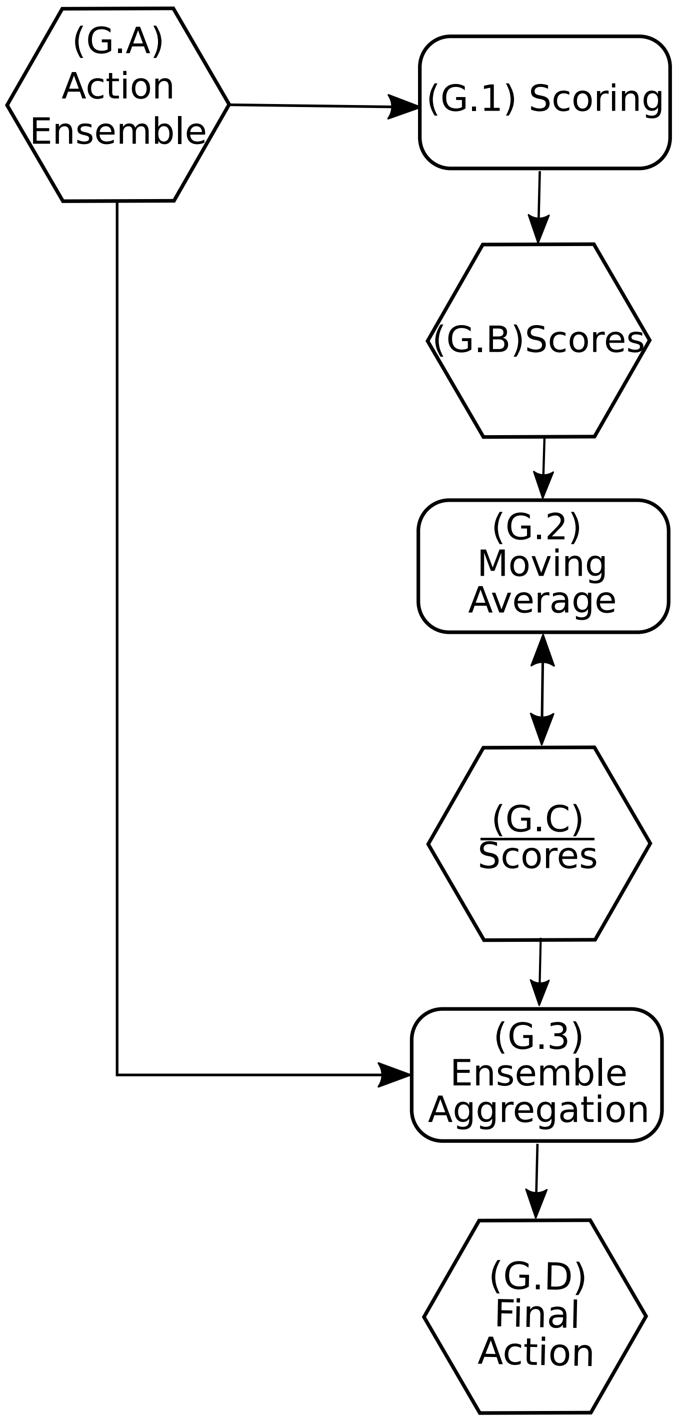

Figure 3.2: Generic History-Based Framework (HBF).

has the highest value before and after the *(G.2) Moving Average* function. On the other hand, the score c_1, even though it has the second-best score, wasn't enough to result in $\bar{c_1} > \bar{c_3}$. The intention of this presentation of the Generic HBF is to offer the mechanism intuition, and the updated implementation is presented in Section 3.3.1.

Figure 3.5 presents the last stage of the Generic HBF. In this step, the *(G.C)* $\overline{Scores}$ are used to aggregate the actions by executing the aggregation

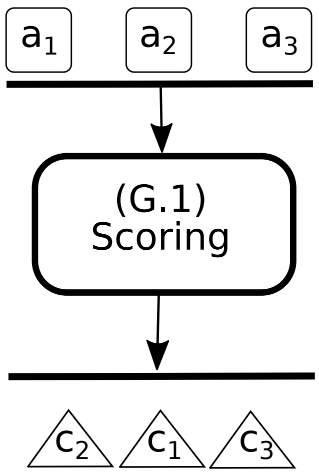

Figure 3.3: History-Based Framework — Example of three agent actions ($\{a_1, a_2, a_3\}$ — *(G.A) Action Ensemble*) that are the input to the *(G.1) Scoring* function and present in their output. The actions are sorted according to the scoring, referring to the aggregation used and the *(G.B) Scores* values ($\{c_2, c_1, c_3\}$).

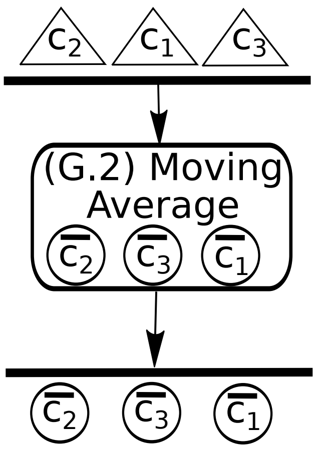

Figure 3.4: History-Based Framework — Example of actions and their scores serve as input to *(G.2) Moving Average*, presenting *(G.C) $\overline{Scores}$* as output, a permanent structure that seeks to qualify policies through their actions.

function (*(G.3) Ensemble Aggregation*), resulting in the *(G.D) Final Action*. As opposed to classical ensemble aggregations, this function has access to the policies' averaged scores, which enables it to make a more informed decision.

3.3.1 HBF Implementation

After presenting the Generic History-Based Framework in Figure 3.2, our specific implementation will now be presented step-by-step, concretely explaining the methods, such as the *(G.1) Scoring* and the *(G.3) Ensemble Aggregation*. In order to explore the best way to define the historical performance,

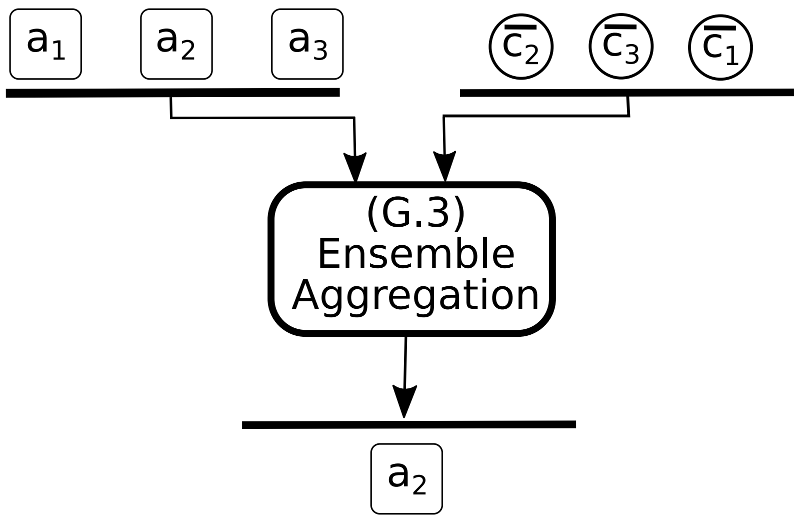

Figure 3.5: History-Based Framework — Example of the last step of the Generic HBF that receives the actions and the ordered *(G.C)* $\overline{Scores}$. The actions are selected according to the scores then aggregated (*(G.3) Ensemble Aggregation*) resulting in *(G.D) Final Action* of the ensemble.

the framework allows different scoring functions. Before explaining it, it is important to highlight the History-Based Parameter and Active Set Aggregation mechanisms.

- **History-Based Parameter**: The History of the Scores works like a low-pass filter by allowing low frequencies to pass without difficulty, and attenuates (or reduces) the amplitude of frequencies higher than the cutoff frequency. The $\overline{scores}$ parameter accumulates the performances at each time step, using the historical learning rate α_h as shown in Eq. (3-1).

$$\overline{scores} = \alpha_h \cdot scores + (1 - \alpha_h) \cdot \overline{scores} \tag{3-1}$$

  Where α_h is the parameter of an exponential moving average filter in the same way that the learning rate α is used in the classical Q-learning algorithm. It has an exponential adjustment in which a lower α_h discounts newer observations more, lowering the cutoff frequency of the low-pass filter. In addition the Eq. (3-1) allows to store only one value and not a vector of scores acquired over time, for example.

- **Active Set Aggregation**: Action Filtering takes advantage of performance history to filter out inherently bad policies. Since the $\overline{scores}$ qualify which policies tend to be good, this allows applying a filter that separates the considered good and bad actions. Performing these separation tasks in online mode, while the algorithms are learning, adds a dynamic and flexible mechanism to the algorithm, contributing to its learning.

So, the actions are ranked according to $\overline{scores}$, and only some are selected to be used in the final aggregation step. The best ones compose a new action ensemble, and a new aggregation is performed to select the best action from the active set. The percentage of selected actions is chosen empirically.

Figure 3.6 presents a complete view of the HBF. The hexagonal structures represent the input or output data of the methods, which are illustrated in the rectangular structures.

The framework allows different scoring functions to explore the best way to define historical performance. Such a scoring method may itself require the definition of an ensemble center, so at the beginning of Figure 3.6, an *(1) Ensemble Aggregation* uses methods described in Section 2.4, to find *(B) Center Action*. This step is only necessary when the Squared Euclidean Distance is used in the next function — *(2) Scoring*.

The *(A) and (B)* data serve as input to *(2) Scoring*, which calculates a weighting of values and, a score, for each action — *(C) Scores*. The following methods are specifically used or adapted for this scoring step: Data Center Ranking, Density Scores and Squared Euclidean Distance. If the framework is used to implement the aggregations already presented in literature, no function is chosen here. In that case, all actions have the same weight, which is the baseline. The *(2) Scoring* can be instantiated by these functions:

– **Data Center Ranking**: The Data Center Ranking uses the same concept as the data center (Section 2.4.1.3), which is used to the selection of the center element. During the selecting process, the one farthest from the center is set to be removed. Data Center Ranking returns the step at which the particular policy's action was removed, where the largest value is the center itself.

– **Density Scores**: The density of each action (d_i) is presented in Section 2.4.1.2. In this scoring, the scores assigned to each action are exactly d_i. As such, the biggest value is assigned to the center.

– **Squared Euclidean Distance**: Although this is a simple approach, it is very effective and widely used in literature [39]. Equation (3-2) presents the square of the difference between a policy action and some chosen aggregate action $\bar{a}$, which can be calculated using any of the ensemble aggregation techniques described in Section 3. The values are multiplied by -1 such that the best action has the highest value.

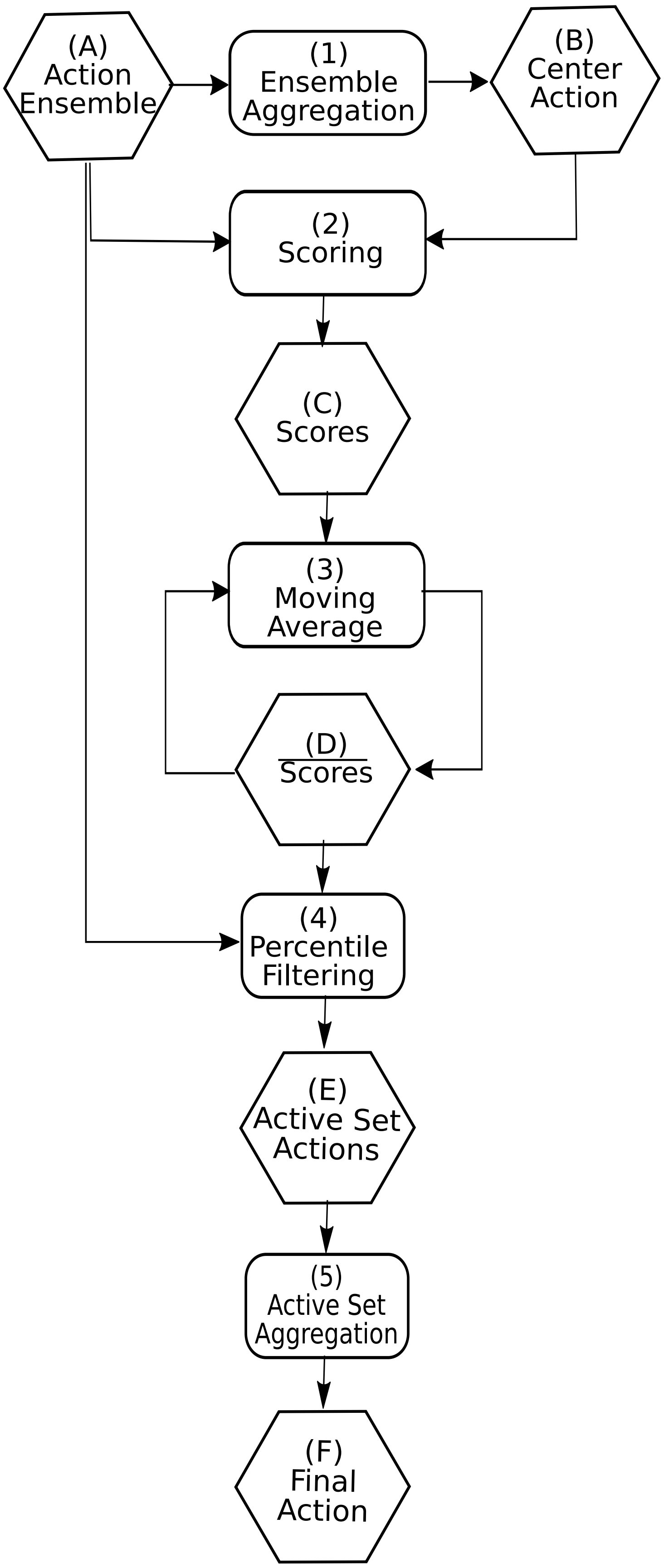

Figure 3.6: History-Based Framework Implementation.

$$ed_i = \sum_{j=1}^{N}(a_{ij} - \bar{a}_j)^2, \qquad (3\text{-}2)$$

where i is a chosen action, j is index from 1 to N (number of ensemble actions) and $\bar{a}$ the average or central element of a group pointed by ensemble aggregation techniques.

The *(3) Moving Average* uses the *(C) Scores* to update the *(D)* $\overline{Scores}$, history-based variable, presented in Equation (3-1). Section 3.3.1 describes in detail the Moving Average function.

The next function of the framework is the *(4) Percentile Filtering*, where a specified percentage of the actions in the *(A) Action Ensemble* is selected based on the order of scores to form the *(E) Active Set Actions.* The idea is to use the scores' history to filter the best actions, safeguarding those policies that perform well over time. This is the main way in which the scores are used to influence the final actions. In general, for all aggregations the shortest distance usually means better value, so a descending score is used in *(4) Percentile Filtering.* In the case of Density Based aggregation, the highest density represents the best action, so the order is ascending.

Finally, the *(F) Final Action* is selected by the *(5) Active Set Aggregation.* The aggregations used here are Best, Data Center, Density Based and Mean. The best aggregation selects the best action based on the $\overline{scores}$, Eq (3-1); in this configuration flow , the *(4) Percentile Filtering* function is not relevant.

The code used in the framework can be found online[1].

3.3.2 Performance Measure in History-Based Framework

To measure the overall performance of the framework setups, in order to select the best overall strategy used for validation, the average relative regret is calculated.

As seen in Fig 3.7, regret is the area between the curve that represents the optimal controller and the curve to be evaluated. This metric compares the best performing aggregation with the others; therefore, larger values mean worse overall performance.

To calculate the average relative regret as in Eq. (3-3), the episode return $p_{e,g,i}$ of agent i on environment e and ensemble group g is subtracted from the best result on that environment/group, and normalized by that same maximum. This is then summed over all environments.

This metric was chosen to compare the different forms of action aggregation between different environments built with HBF, as it has the property of

[1]https://github.com/renata-garcia/grl

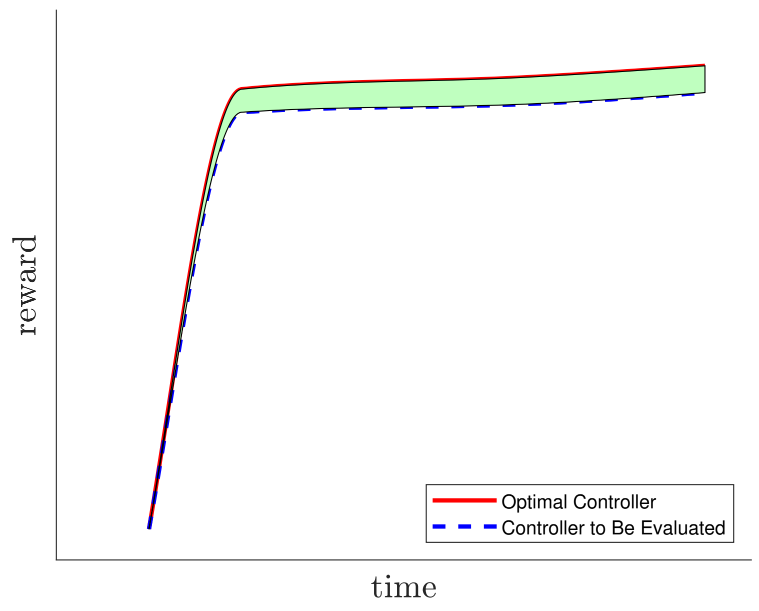

Figure 3.7: Illustration of regret metric which is the integrated difference in performance between the optimal controller and controller to be evaluated (green area).

being invariant to multiplication of rewards by a constant factor. The strategy with the least regret is the one chosen for validation.

$$\mathcal{R}_{\mathrm{HBF}}(i) = \sum_e \sum_g \left| \frac{\max_j p_{e,g,j} - p_{e,g,i}}{\max_j p_{e,g,j}} \right| \tag{3-3}$$

3.4 Online Weighted Q-Ensemble (OWQE)

In the HBF, there is no use of value function information. The methodology to be presented seeks to process this information to contribute to the best choice of the action ensemble. Our Q-ensemble model builds upon the Actor-Critic Ensemble method, by weighing the critics' predictions. Such a weighing, similar to conventional classifier Boosting [40], aims to emphasize the input of the critics that better estimate the environment when selecting the ensemble action. Considering that the DDPG ensemble hyperparameters are chosen based on user-experience without having their performance guaranteed, it is important that critics with bad performance do not destabilize the final policy.

3.4.1 Inference

Figure 3.8 presents the Q matrix generation in DDPG Online Weighted Q-Ensemble. The ensemble is composed of n DDPG agents, each composed of a critic $Q_i = Q(\cdot;\zeta_i)$ and an actor $\mu_j = \mu(\cdot;\theta_i)$, $i, j \in 1 \dots n$. At state s at a given time step t, each actor μ_j calculates an action a_j. Subsequently, each of the actions is evaluated by all critics, generating the corresponding Q-values $Q(s, a_j; \zeta_i)$. The generation of these values results in a matrix $\mathbf{Q}$ with elements q_{ij}.

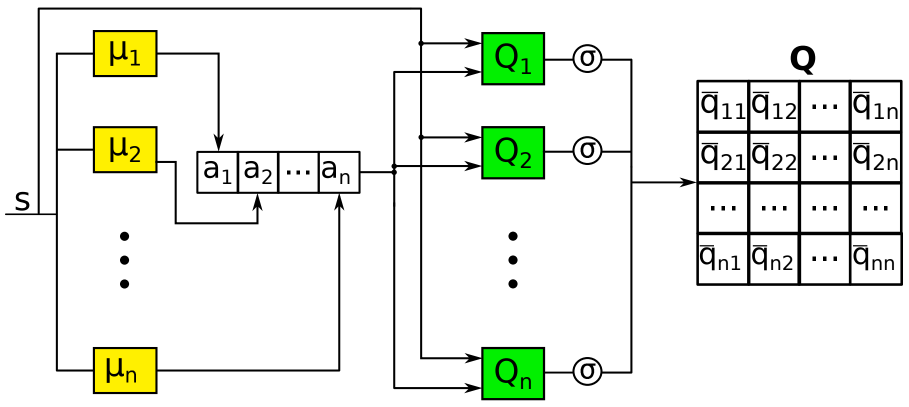

Figure 3.8: Q matrix generation in DDPG Online Weighted Q-Ensemble. Each critic Q_i evaluates the actions suggested by all actors μ_j, and the resulting values are normalized using the softmax function σ.

The DDPG networks are updated independently, since they use different hyperparameters. This creates a challenge when analyzing the $\mathbf{Q}$ matrix to select the ensemble action because their Q-value magnitudes may not be directly comparable. We propose the use of a softmax function σ to normalize the values of a critic for the different actions, Eq. (3-4). The $\sigma(q_{ij})$ normalizes q-value, q_{ij}, over all q-values related to that Q_i value function. Initially, it was thought to normalize with respect to actions: that is, to get all the q-values generated with an action a_j. However, this approach would maintain the distortions, as each value function has a generalization of the knowledge of the environment, and its result about different actors' actions, μ_j, is very close to each other.

$$\sigma(q_{ij}) = \frac{e^{q_{ij}}}{\sum_{k=1}^{n} e^{q_{ik}}} \tag{3-4}$$

Figure 3.9 presents how the weights form the q-values matrix, $\mathbf{Q}$. The raw weights W_{raw} are normalized with the same softmax function σ to ensure they form a proper distribution, resulting in W. Then, we calculate the weighted average using weight vector W of length n to calculate the critic ensemble prediction for all actions, Eq. (3-5), where $W = \frac{1}{n}\mathbf{1}$ results in the standard Q value averaging.

$$q_{\text{w}} = W^T\mathbf{Q} \tag{3-5}$$

Due to the normalization of the $\mathbf{Q}$ matrix, this procedure is equivalent to interpreting each critic as defining a softmax policy over the suggested

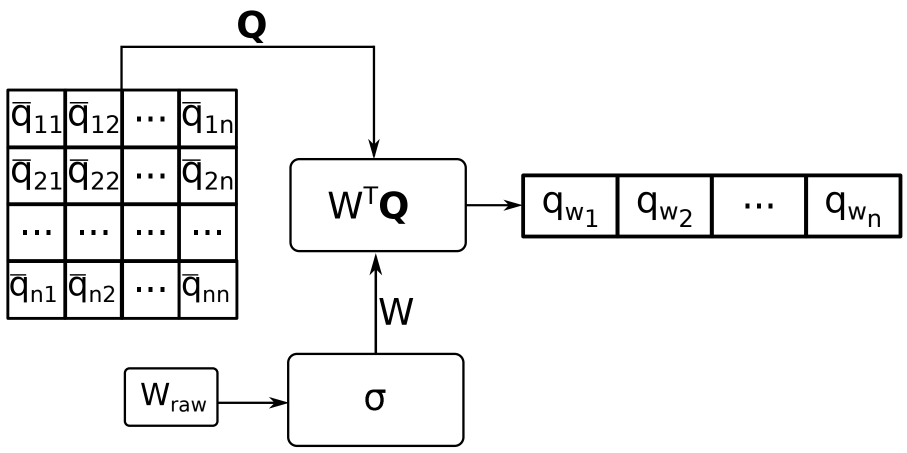
Q
$\bar{q}_{11}$ $\bar{q}_{12}$ ··· $\bar{q}_{1n}$
$\bar{q}_{21}$ $\bar{q}_{22}$ ··· $\bar{q}_{2n}$
··· ··· ··· ···
$\bar{q}_{n1}$ $\bar{q}_{n2}$ ··· $\bar{q}_{nn}$
$W^T\mathbf{Q}$
q_{w_1} q_{w_2} ··· q_{w_n}
W
W_{raw}
σ

Figure 3.9: Final Q value generation in DDPG Online Weighted Q-Ensemble. The final Q-value q_{w_j} of each action a_j is the sum of the critics' values $\{\bar{q}_{1j}, \bar{q}_{2j}, \cdots, \bar{q}_{nj}\}$, weighted by their respective weights w_i.

actions, and averaging the resulting action probabilities (also called Boltzmann addition [14]). The final ensemble action, a_t, is the one with the highest probability, Eq. (3-6).

$$a = \mu_z(s), \tag{3-6}$$

where $z = \arg\max q_w$.

Figure 3.10 presents the complete image of the Online Weighted Q-Ensemble Model explained step-by-step.

3.4.2 Online Training Weights

All DDPG agents are trained in a single environment, using a shared replay buffer. The behavior policy v^β is either derived from the ensemble action (online training) or from each actor in sequence on a per-episode basis (alternate training). The former can be expected to learn faster, while the latter ensures at least some near on-policy transitions for all actors, increasing robustness.

The raw weights, W_{raw}, are initialized uniformly, and passed through a softmax layer before being used for the weighting. Therefore, at the beginning of the training, the critic weights W remain close to the uniform distribution. During training, we minimize the temporal difference (TD) error of the critic ensemble by minimizing the loss

$$\begin{aligned} \mathcal{L}(W_{\mathrm{raw}}) &= \sum_{(s,a,r,s')\in\mathcal{D}} \sum_{i=1}^{n} w_i \delta_i^2 \\ \delta_i &= \quad r_i + \gamma_i Q(s', \mu(s'; \theta_i'); \zeta_i') - Q(s, a; \zeta_i). \end{aligned} \tag{3-7}$$

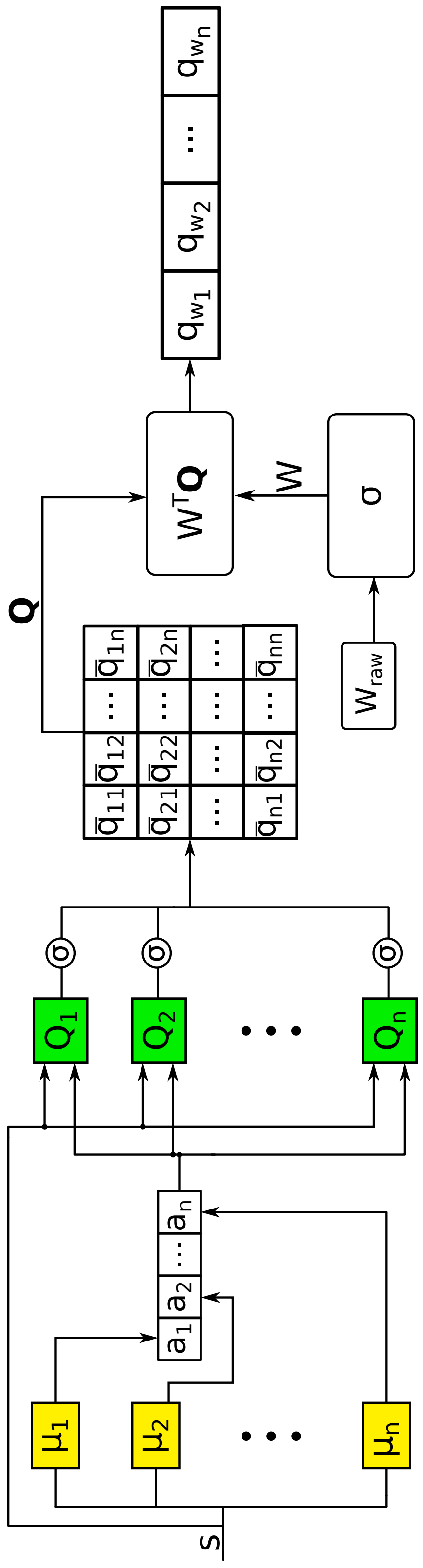

Figure 3.10: Online Weighted Q-Ensemble. Each critic Q_i evaluates the actions suggested by all actors μ_j, and the resulting values are normalized using the softmax function σ. The final Q-value q_{w_j} of each action a_j is the sum of the critics' values $\{\bar{q}_{1j}, \bar{q}_{2j}, \cdots, \bar{q}_{nj}\}$, weighted by their respective weights w_i.

Minimizing Eq. (3-7) reduces the weights of the critics with higher squared TD error, which can be assumed to have a worse value function prediction. Since $\sum_i w_i = 1$, due to the softmax function applied to W_{raw}, this increases the better critics' weight.

The code used in the Online Weighted Q-Ensemble can be found online[2].

3.4.3 Performance Measure in Online Weighted Q-Ensemble

This metric compares the different forms of aggregation between different environments, and unlike the performance metric presented in Section 3.3.2, it has the property of being invariant to the addition of a constant (instead of just constant multiplication), which allows better robustness in comparing environments that have performance values at different scales. To be invariant to both addition of and multiplication by a constant means that the variance within an environment, with performances ranging from $1000-1100$, will have the same weight as one with performances from $0-500$.

Therefore, to measure the overall performance of the aggregations used, the average relative regret is calculated over all environments and ensemble groups:

$$\mathcal{R}_{\text{OWQE}}(i) = \sum_e \sum_g \left| \frac{\max_j p_{e,g,j} - p_{e,g,i}}{\max_j p_{e,g,j} - \min_j p_{e,g,j}} \right| \tag{3-8}$$

where $p_{e,g,i}$ is the performance of aggregation i in environment e for ensemble group g.

[2]https://github.com/renata-garcia/wce_ddpg

4
Simulations

This chapter describes how the simulations were setups, Section 4.1 explains the model architecture used through methodologies. Section 4.2 shows the organization of the Ensemble Groups that allowed evaluating the methodologies. The Strategy Experiments of History-Based Framework and Online Weighted Q-Ensemble are presented in Sections 4.3 and 4.4. Lastly Section 4.5 presents the environments, that run in a fast physics simulator.

4.1
Model Architecture

Initially, the environments were executed with the architecture model described as follows in Section 4.1.1, which was used as a basis to generate the configurations of the others 31 DDPG algorithms. The total of 32 DDPG algorithms were executed in the environments and ordered according to the last episode's average performance. This procedure was used to assemble the sets for each experiment.

In the case of the Online Weighted Q-Ensemble Model, the same methodology was followed with the exception of two hyperparameters. In order to prevent any bias in the TD Error calculation, the discount factor γ and reward scale rs values were frozen at 0.99 and 0.01 respectively, and other parameters were varied.

Hyperparameters influence the speed of learning and the ability to generalize the algorithm, as they are variables related to the network structure and its training. The value ranges used to compose the trained DDPG algorithms are shown in Table 4.1. These ranges were based on the DDPG algorithm presented for continuous action environments [27]. In the next section, the initial configuration of the architecture is described.

4.1.1
Initial Configuration of the Architecture

The default actor and critic networks have two hidden layers with 400 and 300 neurons, respectively, as recommended in the original DDPG paper [27]. The neural networks learn using Adam Optimizer [26], with the actor and

Table 4.1: Table with the Hyperparameters' Description used to Build each Algorithm of the Ensemble.

Hyperparameters	**Value**	**Description**
discount factor	0.98 or 0.99	Discount factor used in the Q-learning update.
reward scale	0.01, 0.1 or 1	Scaling factor applied to the environment's rewards.
soft target update rate	0.01	Update rate of the target network weights.
update interval	10 or 100	Steps number, or frequency, with which the soft target update is applied.
learning rate	0.001 and 0.0001	Update rate used by AdamOptimizer.
replay steps	64, 128 or 256	Number of minibatches used in a single step.
minibatch size	16, 64 or 128	Number of transitions from the environment used in batch to update the network.
layer size	50, 100, 200, 300 and 400	Number of neurons in regular densely-connected NN layers
activation function	relu or softmax	Activation function of the hidden layers.
replay memory size	1000000	Size of the replay memory array that stores the agent's experiences in the environment.
observation steps	1000	Observation period to start replay memory using random policy.

the critic learning rates of 10^{-4} and 10^{-3}, respectively. The discount factor is $\gamma = 0.99$, reward scale is $rs = 0.1$ and the soft target updates are $\tau = 0.001$ with an update interval of 1. The exploration noise uses an Ornstein-Uhlenbeck (OU) process [30], which is specific to explore physical environments that have momentum. The OU parameters are $\theta_{\mathcal{N}} = 0.15$ and $\sigma_{\mathcal{N}} = 1.0$ for all simulations discussed in this work. The importance of this technique is to avoid temporally correlating the exploration, because pure white noise would not lead to significant deviation. From these DDPG initial values, some modifications were made in order to improve the performance in each environment. The number of inputs of the actor and critic networks, as well as the number of outputs of the actor depend on the environments, as described in Section 4.5.

4.2 Ensemble Groups

In order to verify the efficiency of ensembles, we used an approach testing groups of good and bad parametrizations with different algorithms (DDPG instantiated with different parameters). For this, 32 DDPG agents with distinct hyperparameters were individually tested. Good hyperparameters in one environment will not necessarily remain in the others; as such, the classification of a parameterization as either good or bad depends on the environment.

Therefore, each environment was extensively executed to find the best performance, performing a narrow grid search for each hyperparameter. Upon finding a high performance for one variable, it was frozen and a grid variation was performed on another variable, until the process ended. This is computationally quite costly, yet it is important in the discovery of the best single DDPG baseline for comparisons.

The groups focus on measuring the interference of bad hyperparameters in the methods. For this, 2 sets of groups were created, one for the action aggregation and the other for the value aggregation, with the *3 best* group being used as baseline. The average end performance of 31 (10 in case of Half Cheetah and Swimmer) learning runs was used to order the individual parameterizations.

4.2.1 *3 Best or 3 Good Group*

After running the 32 single DDPG for each specific environment, the three best hyperparameter parametrizations (algorithms) are grouped. Sometimes called *3 Best Group* and sometimes called *3 Good Group*.

4.2.2 Action Aggregation Groups

After ranking the individual performances, the 12 best ones were classified as good, and the 12 worst as bad. Table 4.2 summarizes the composition used in Classical Aggregations in DDPG with Continuous Action Space, while Table 4.3 summarizes the composition of the groups formed in the History-Based Framework tests.

Table 4.2: Composition of test configurations. Groups of DDPG parameterizations used in Classical Aggregations, with Continuous Action Space.

parameterizations	***3 best***	***good***	***bad***
best	3	12	4
worst	0	4	12

Table 4.3: Composition of test configurations. Groups of DDPG parameterizations used in the History-Based Framework tests.

parametrizations	***good***	***mid***	***bad***
best	12	8	4
worst	4	8	12

4.2.2.1 *Good Group*

This ensemble is composed by 16 DDPG, as the idea is to gather 75% of the group, 12 good algorithm parameterizations and reserve 25%, 4 bad parameterizations that are not well configured, to result in a non-convergence of learning. The idea is to verify the ensemble's learning strategy behavior when the individual parameterizations are not exclusively good, therefore certifying the learning resilience.

4.2.2.2 *Mid Group*

This ensemble is also composed by 16 DDPG, but splitting the algorithm parameterizations group in half, with 50% being the good parameterizations and the other 50% the bad ones (non-convergence of learning).

4.2.2.3 *Bad Group*

Opposite to the previous compositions, 75% of this group is composed by bad parameterizations and 25% of good parameterizations. As the majority is composed of bad parameterized non-coverging algorithms, it becomes an immense challenge to carry out learning, based on those good algorithms.

4.2.3 Value Aggregation Groups

4.2.3.1 *1 Good 1 Bad Group*

The ensemble is composed of 2 DDPG, one well parameterized and the other poorly parameterized, which results in the non-convergence of learning. The idea is to verify the learning strategy behavior when there is a high performance distinction between two algorithms.

4.2.3.2 *1 Good 3 Bad Group*

The ensemble is composed of 4 DDPG, one well and three poorly parameterized, which result in the non-convergence of learning. The idea here is to verify the learning strategy behavior when 75% of the algorithms do not converge.

4.2.3.3
1 Good 7 Bad Group

The ensemble is composed of 8 DDPG, one well and seven poorly parameterized, and once more the learning convergence is avoided verifying the learning strategy behavior when a large percentage of algorithms do not converge.

4.3
Strategy Experiments of History-Based Framework

Since many ensemble strategies for continuous reinforcement learning use pre-learned algorithms [16], such simulations were also performed for comparison purposes. The base case considers just the data center, density based and mean strategies.

Table 4.4 introduces the history-based framework instances (Section 3.3) with the first set of chosen DDPG being the base case strategies highlighted at the top of the table. A composition of acronyms was created, in order to enable the identification of the History-Based Framework configuration.

- DC: Data Center;
- DB: Density Based;
- M: Mean;
- DCR: Data Center Ranking;
- DS: Density Scores; and
- ED: Squared Euclidean Distance.

Table 4.4: Ensemble Strategies Configuration of History-Based Framework.

	strategy	ensemble aggregation	percentile filtering	scoring	a_h	active set aggregation
BASE	DC	—	—	—	1.0	data center
	DB	—	—	—	1.0	density based
	M	—	—	—	1.0	mean
	DCR-B	—	—	data center ranking	0.01	best
	DS-B	—	—	density scores	0.01	best
	M-ED-B	mean	—	euclidean distance	0.01	best
	DCR-DC	—	25%	data center ranking	0.01	data center
	DS-DB	—	25%	density scores	0.01	density based
	M-ED-M	mean	25%	euclidean distance	0.01	mean
	M-ED-DC	mean	25%	euclidean distance	0.01	data center
	M-ED-DB	mean	25%	euclidean distance	0.01	density based
	DC-ED-DC	data center	25%	euclidean distance	0.01	data center

Initially a class of instances was chosen, performing only history-based score filtering while choosing the best-scoring policy at the end of the process.

These three strategies are indicated in the table by: DCR-B, DS-B and M-ED-B.

Another experiment was to perform active set aggregation. For this, after history-based filtering 25% of the actions with the best moving average, scores are selected and used to calculate the final action. It is important to point out that other values, such as 50% and 75%, were empirically tested in the history-based filtering configuration, however, only with 25% it was empirically noticed that the selective grouping contributes to the improvement of performance.

These three history-based strategies are indicated in the table as DCR-DC, DS-DB, and M-ED-M. For consistency, we applied the same style of aggregation function during *scoring* and *active set aggregation*. Finally, other experiments were performed, based on the observed performance of the previous strategies. These three strategies are presented at the end of table: M-ED-DC, M-ED-DB and DC-ED-DC.

In the comparative results always the mean and the 95% confidence interval over 31 runs (10 in case of Half Cheetah) are shown, considering that randomly observed samples larger than 30 may be supposed to be normally distributed [40, p.280]. The strategies presented in bold numbers are those whose mean is within the 95% confidence interval of the best ensemble result, while strategies with a mean within or above the 95% confidence interval of the best single policy are marked with an asterisk.

In order to measure the overall performance of the framework setups to select the strategy used for validation in the Half Cheetah v2 environment, the average relative regret is calculated as in Eq. (3-3).

4.4 Strategy Experiments of Online Weighted Q-Ensemble

To validate our value aggregation model, its performance was tested by ablating the two differences with respect to Q-value averaging: using a weighted average and Boltzmann addition. This results in the following combinations:

- *Softmax TDError*: uses the model presented in Section 3.4;
- *TDError*: skips the softmax normalization of the Q-values presented in Eq. (3-4);
- *Softmax Average*: maintains $W = \frac{1}{n}\mathbf{1}$, but otherwise implements the model of Section 3.4; and
- *Average*: standard Q-value averaging.

The average policy ensemble, with independently trained DDPG agents and without a Q-ensemble, recently presented good performances with 3 fine-tuned hyperparameter sets [17] in a single environment 2D robot arm simulator. Based on this result, one ensemble group with 3 fine-tuned DDPG (*3 Best*) instances is used to validate the model.

However, the Weighted DDPG Ensemble Model seeks to minimize the effort of fine-tuning in an ensemble, and to that end 3 more ensemble groups were created mixing good (fine-tuned) and bad (not fine-tuned and non-converging) DDPG hyperparameters: *1 Good and 1 Bad*; *1 Good and 3 Bad*; and *1 Good and 7 Bad*.

In addition, two types of training mode are used in order to expand the model validation. In the alternate training mode, at the beginning of each training episode, the policy is chosen alternately between each of the algorithms of the ensemble [17], Section 2.5.2. In the online training mode and in the ensemble testing phase, the ensemble action is chosen at each episode step, Section 3.1.

As also occurred in the simulation of the HBF, simple control problems were chosen as environments, and two harder robotic tasks were picked to evaluate scalability. In all cases, the episodes start at the resting point of the environment, the observations are in trigonometric format and there are 1000 observation steps before starting training. The ranges of the hyperparameters are given in Table 4.1, except the discount factor γ and the reward scale rs which are frozen in 0.99 and 0.01 respectvelly.

The specific environments used are the Inverted Pendulum Swing-up and Cart-Pole environments from the Generic Reinforcement Learning Library (GRL) [1], Half Cheetah v2 and Swimmer v2 from the OpenAI Gym framework [24] with the MuJoCo environments [23]. Swimmer v2 was used as a final validation for hyperparameter randomization. For this environment, 30 random configurations of ensembles, formed with 8 DDPG, were executed. The network architecture and the hyperparameters were randomly generated within the limits already used (Table 4.1).

In order to measure the overall performance of the framework setups to select the strategy used for validation in the Swimmer v2 environment, the OWQE average relative regret is calculated as in Eq. (3-8).

4.5 Environment

The environments used in the experiments are Inverted Pendulum Swing-up, Cart Pole, Cart Double Pole, Half Cheetah v2, and Swimmer v2. The

[1]GRL Library (Generic Reinforcement Learning Library) (https://github.com/wcaarls/grl).

first three ones are provided by the Generic Reinforcement Learning Library (GRL) [2]. The last two are used from the OpenAI Gym framework [24] with the MuJoCo environments [23]. The training environments consider random initialization to avoid overfitting to a single starting condition.

4.5.1 Inverted Pendulum Swing-up (PD)

The pendulum is a free pole, attached only by an axis that, if there is no external force, remains in the stable equilibrium at the down position, as shown in Figure 4.1. The goal is to learn the necessary torque to swing the pole, rotate it on the axis and maintain balance on the unstable equilibrium of the top position [41]. It does not have enough torque to accomplish this in one swing, so it has to learn to swing back and forth to gain energy. The pole's angle is observed in terms of its sine and cosine, in order to avoid the $[-\pi, \pi]$ wrapping problem (more details of training and variables characteristics can be found in Table 4.5). The reward function is represented by (4-1), where x are the state variables, pendulum angle α and angular velocity $\dot{\alpha}$, while u is the control action in volts.

$$\rho(x,u) = -x^T Q_{rew} x - R_{rew} u^2 \tag{4-1}$$

where

$$Q_{rew} = \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}, R_{rew} = 1$$

Table 4.5: Variables of Inverted Pendulum Swing-up Training.

condition	values
init:	random position of pole.
goal:	swinging pendulum up so it maintains upright.
observation variables:	(1) pendulum position $\sin(\alpha)$. (2) pendulum position $\cos(\alpha)$. (3) velocity $\dot{\alpha}$ of the pole.
control action:	(1) motor voltage u.
α range:	$[-\pi, \pi]$.
$\dot{\alpha}$ range:	$[-12\pi, 12\pi]$.
u range:	$[-3, 3]$ V.

4.5.2 Cart Pole (CP)

The Cart Pole problem consists of a cart that moves back and forth along a no friction track with the task of balancing a pole attached by an

[2]GRL Library (Generic Reinforcement Learning Library) (https://github.com/wcaarls/grl).

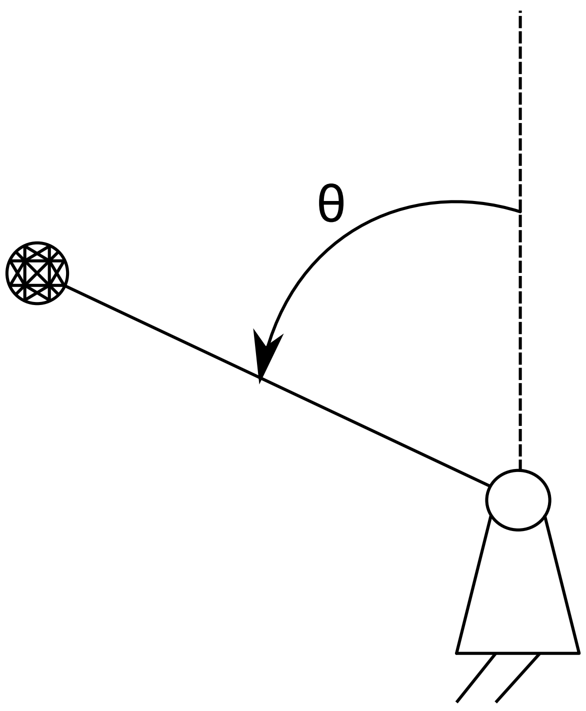

Figure 4.1: Illustration of the Inverted Pendulum Swing-up Model.

un-actuated joint as shown in Figure 4.2. The goal is to learn how to swing up and balance the pole just by moving the car around the track [42, 43]. The observation variable, as in the inverted pendulum swing-up environment, considers information derived from the angle, now along with the cart position. Table 4.6 shows the training details and variable characteristics. The reward function is represented in Equation (4-2).

$$\rho(x,\theta,\dot{x},\dot{\theta}) = -2x^2 - 0.1\dot{x}^2 - \theta - 0.1\dot{\theta}^2 \tag{4-2}$$

4.5.3 Cart Double Pole (CDP)

As in the cart pole, a pole is attached by an un-actuated joint to a cart, which moves along a no friction track [44]. But this time, another pole is attached by an un-actuated joint to the first pole as shown in Figure 4.3. The poles start near upright, and the goal is to prevent the system from falling over, with the reward function being represented by Equation (4-3). The observation variable considers information derived from both angles and the cart position. The training and variables characteristics are found in Table 4.7. The episode ends prematurely when the pole is more than 15 degrees from the vertical, or when the cart moves more than 2.4 meters from the center.

$$\rho(x,\theta_1,\theta_2) = 6.2 - |x| - |\theta_1| - |\theta_2|; \tag{4-3}$$

Table 4.6: Variables CP Training.

condition	**values**
init:	car in middle and pole in a random position.
goal:	swing pole up and maintain it upright.
observation variables:	(1) position of the cart on the track (x). (2) angle of the pole with the vertical (θ). (3) cart velocity ($\dot{x}$). (4) angular velocity ($\dot{\theta}$).
control action:	(1) force u applied to the cart.
θ range:	$[-\pi, \pi]$.
$\dot{\theta}$ range:	$[-5\pi, 5\pi]$.
x range:	$[-2.4, 2.4]$.
$\dot{x}$ range:	$[-10, 10]$.
u range:	$[-15, 15]$ V.

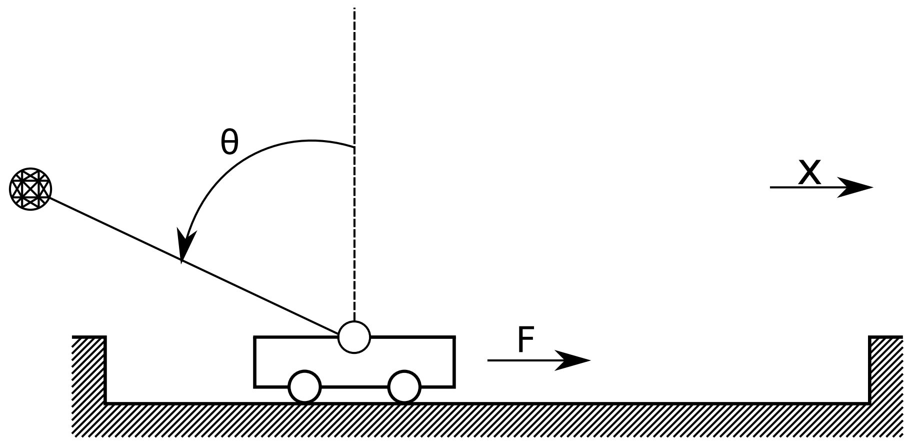

Figure 4.2: Illustration of the Cart Pole Model.

Table 4.7: Variables Cart Double Pole Training.

condition	**values**
init:	car in middle and pole in upright position.
goal:	maintain it upright.
observation variables:	(1) position of the cart on the track (x). (2) angle of the pole 1 with the vertical (θ_1). (3) angle of the pole 2 with the vertical (θ_2). (4) cart velocity ($\dot{x}$). (5) angular velocity of pole 1 ($\dot{\theta}$). (6) angle velocity of pole 2 ($\dot{\theta_2}$).
control action:	(1) force u applied to the cart.
θ_1 and θ_2 range:	$[-\pi, \pi]$.
$\dot{\theta_1}$ and $\dot{\theta_2}$ range:	$[-5\pi, 5\pi]$.
x range:	$[-2.4, 2.4]$.
$\dot{x}$ range:	$[-10, 10]$.
u range:	$[-20, 20]$ V.

4.5.4 Half Cheetah v2 (HC)

Half Cheetah is a walking animal in a 2D environment [23, 45]. Figure 4.4 shows the six joint points of the half cheetah to which torque can be applied in

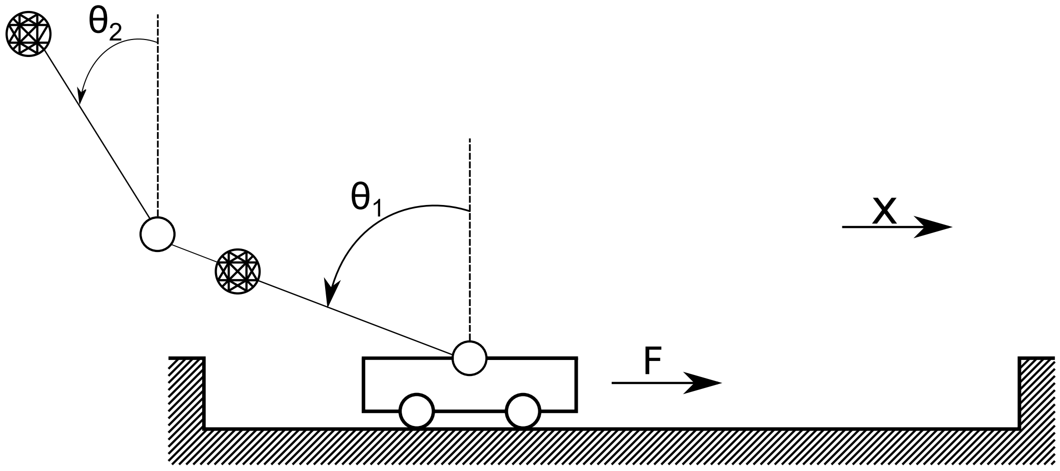

Figure 4.3: Illustration of the Cart Double Pole Model.

order to control this two-legged walking robot. For each leg, the three degrees of freedom correspond to thighs, shins and feet. The seventeen observation variables and six actions are described in more depth in Table 4.8.

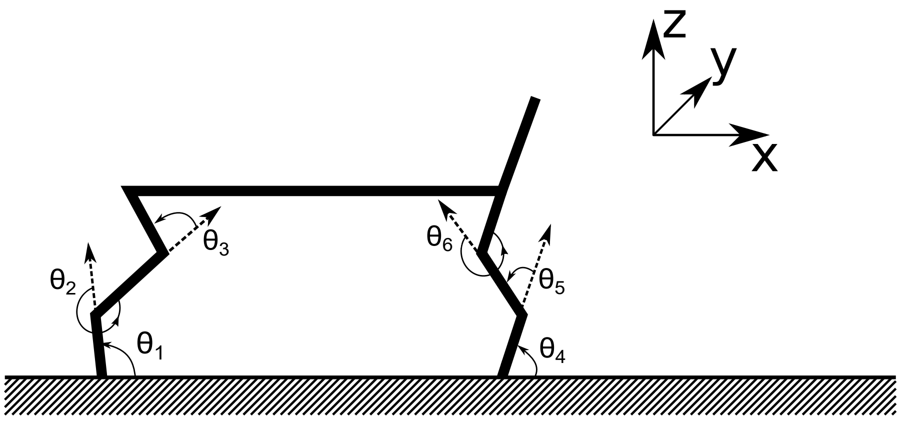

Figure 4.4: Illustration of the Half Cheetah v2 Model.

4.5.5 Swimmer v2 (SW)

Swimmer is a three-link robot in a viscous fluid in a 2D environment [23]. This simulation's goal is to make it swim forward as fast as possible. Figure 4.5 shows the two joint point of the swimmer which receive torque. The eight observation variables and two actions are described in Table 4.9 [46].

Table 4.8: Variables of Half Cheetah v2 Training.

condition	**values**
init:	random variables.
goal:	learn to walk alone as fast as possible.
observation variables:	(1) x position, range $[0, 1]$ meter.
	(2) y angle, range $[-\pi, \pi]$.
	(3) z position, range $[0, 1]$ meter.
	(4) hind foot angle (θ_1), range $[-\pi, \pi]$.
	(5) hind shin angle (θ_2), range $[-\pi, \pi]$.
	(6) hind thigh angle(θ_3), range $[-\pi, \pi]$.
	(7) front foot angle (θ_4), range $[-\pi, \pi]$.
	(8) front shin angle (θ_5), range $[-\pi, \pi]$.
	(9) front thigh angle (θ_6), range $[-\pi, \pi]$.
	(10) x velocity, range $[-1, 1]$ m/s.
	(11) y angular velocity, range $[-10, 10]$ m/s.
	(12) z velocity, range $[-1, 1]$ m/s.
	(13) hind foot angular velocity ($\dot{\theta_1}$), range $[-10, 10]$ m/s.
	(14) hind shin angular velocity ($\dot{\theta_2}$), range $[-10, 10]$ m/s.
	(15) hind thigh angular velocity ($\dot{\theta_3}$), range $[-10, 10]$ m/s.
	(16) front foot angular velocity ($\dot{\theta_4}$), range $[-10, 10]$ m/s.
	(17) front shin angular velocity ($\dot{\theta_5}$), range $[-10, 10]$ m/s.
	(18) front thigh angular velocity ($\dot{\theta_6}$), range $[-10, 10]$ m/s.
control action:	(1) torque hind thigh, range $[-1, 1]$ N m.
	(2) torque hind shin, range $[-1, 1]$ N m.
	(3) torque hind foot, range $[-1, 1]$ N m.
	(4) torque front thigh, range $[-1, 1]$ N m.
	(5) torque front shin, range $[-1, 1]$ N m.
	(6) torque front foot, range $[-1, 1]$ N m.

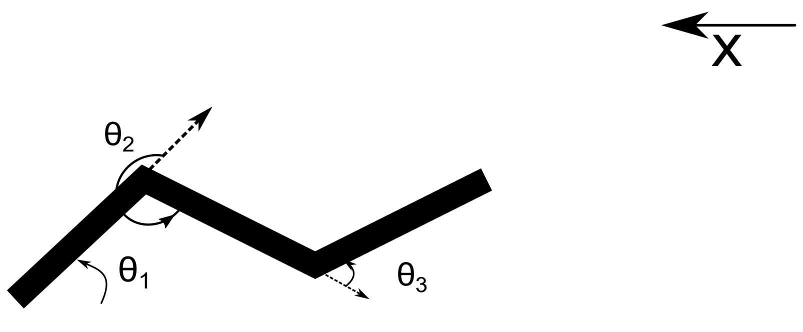

Figure 4.5: Illustration of the Swimmer v2 Model.

Table 4.9: Variables of Swimmer v2 Training.

condition	**values**
init:	random variables.
goal:	swim forward as fast as possible.
observation variables:	(1) x position, range $[0, 1]$.
	(2) x velocity, range $[-1, 1]$.
	(3) angle 1 (θ_1), range $[-\pi, \pi]$.
	(4) angle 2 (θ_2), range $[-\pi, \pi]$.
	(5) angle 3 (θ_3), range $[-\pi, \pi]$.
	(6) angular velocity 1 ($\dot{\theta_1}$), range $[-10, 10]$.
	(7) angular velocity 2 ($\dot{\theta_2}$), range $[-10, 10]$.
	(8) angular velocity 2 ($\dot{\theta_3}$), range $[-10, 10]$.
control action:	(1) torque 1, range $[-1, 1]$ N m.
	(2) torque 2, range $[-1, 1]$ N m.

5
Results

This chapter summarizes the results simulated. Section 5.1 presents the algorithm's individual results which compose the ensemble's validation groups according to their performance. In Section 5.2 results are presented comparing classical aggregations using DDPG. Sections 5.3 and 5.4 present the results and their discussions of the History-Based Framework and Online Weighted Q-Ensemble.

5.1
Individual Results

The performance execution of the individual action ensemble experiment algorithms proposed in the action aggregation study is presented in Table 5.1, where the performance values used to form groups from the best and the worst algorithms are shown. All environments have high different performance variations, with the exception being the Cart Double Pole, probably due to its initial state being already close to the equilibrium state. After sorting the 32 algorithms by performance, only the 12 best and 12 worst are presented, as the others are not used in the ensemble groups.

Furthermore, Table 5.2 presents the performance of the individual algorithms in Q-ensemble experiments, displaying the three best and the 7 worst performances per episode. The algorithm hyperparameter values are shown previously in Table 4.1, note that these performance values are different from Table 5.1 because the experiments do not consider varying the discount factor and reward scale. Furthermore, these values were generated in a new framework created from scratch in Python Language[1] on the other hand the values in Table 5.1 were generated in GRL framework[2]. All individual performance simulations are presented in Appendix A, yet in the Tables cited 5.1 and 5.2, only the performances of the algorithms selected to participate in ensemble compositions are presented.

[1]https://github.com/renata-garcia/wce_ddpg
[2]https://github.com/wcaarls/grl

Table 5.1: Parameterizations Chosen by Environment and Their Performance in Action Ensemble.

	Inverted Pendulum	Cart Pole	Cart Double Pole	Half Cheetah
BEST	−792±14	−344±86	616±1	1419±73
	−794±15	−378±112	602±10	1407±333
	−797±16	−472±153	588±19	1378±85
	−801±22	−484±303	585±34	1275±69
	−803±18	−509±204	585±15	1267±300
	−808±19	−538±284	575±36	1255±115
	−815±20	−613±191	558±56	1247±150
	−816±24	−682±387	538±31	1235±107
	−818±26	−769±351	531±57	1208±150
	−820±22	−952±363	529±59	1194±122
	−824±23	−984±208	500±36	1145±203
	−827±22	−1033±269	495±44	1135±127
WORST	−3508±0	−2860±737	343±93	775±286
	−3508±0	−3071±577	330±53	758±194
	−3508±0	−3366±574	284±31	708±205
	−3508±0	−4037±1686	282±37	629±325
	−3508±0	−4274±2297	275±100	602±350
	−3508±0	−4572±84	269±35	525±351
	−3523±6	−4599±72	266±32	483±295
	−3585±47	−4645±61	262±30	262±258
	−3605±29	−4695±17	241±28	215±250
	−3653±76	−4704±15	230±42	138±245
	−3897±120	−4709±12	180±87	−14±24
	−3962±107	−4712±13	30±11	−29±10

Table 5.2: Parametrizations by Environment and Their Performance used in Value Ensemble.

	Inverted Pendulum	Cart Pole	Half Cheetah
BEST	−754±13	−239±34	4084±1271
	−766±13	−271±61	3790±0
	−769±20	−293±64	3194±1551
WORST	−1293±773	−1424±300	1623±556
	−1318±109	−1445±292	1414±263
	−1653±727	−1498±256	1397±266
	−1691±811	−1617±326	1292±323
	−2280±814	−1716±1008	873±723
	−2846±977	−2000±1437	−240±0
	−4030±964	−2364±929	−583±94

5.2 Classical Aggregations in DDPG with Continuous Action Spaces

This section presents the comparison results of the DDPG execution using the mean aggregation and the continuous action aggregations [14] presented in literature. The groups used were *3 best*, tested with Mean aggregation, *good* and *bad*, with the latter being used in order to verify the ensemble resilience in the use of non-converging algorithms. The other aggregations used were Data Center (Section 2.4.1.3) and Density Based (Section 2.4.1.2).

The performance values of each experiment represent the average of the last 10 episode returns in the learning runs. In the Tables 5.1 and 5.2, the final result is presented with an average performance over 30 runs and its 95% confidence interval, except the Half Cheetah environment that always uses a 10 runs average due to its computational complexity. The best single DDPG algorithm is presented as a benchmark for performance comparison.

The executions of the ensemble, which showed an intersection of the mean and its confidence interval with *Best* DDPG, are highlighted in bold. The experiments were run several times to achieve an expected reliability for comparison purposes [40].

Analyzing Tables 5.3 and 5.4, it can be seen that the *3 best* group in Alternately Persistent training performs very well with any chosen strategy; even the simple Mean strategy works properly in this learning mode, but it does not in the Online Learning.

However, with the Data Center strategy in Online Learning, the *3 best* performs equal to or even better than Alternately Persistent, and with a narrower confidence interval, indicating a more stable learning. This stability becomes interesting when the results of the *good* group compared: for this case, the Data Center Online Learning always converges, although the ensemble includes non-learning-capable algorithms.

Table 5.3: Comparison Performance of Classical Aggregations Ensemble in DDPG with Continuous Action Space in Alternately Persistent.

		DDPG	Alternately Persistent		
			3 best	*good*	*bad*
PD	Best	-792 ± 14			
	Mean		**-839**±43	−1613±174	−4041±118
	Data Center		**-795**±11	**-804**±10	−1784±406
	Density Based		**-922**±136	−1080±161	−2361±259
CP	Best	-344 ± 86			
	Mean		**-416**±154	−962±273	−2550±334
	Data Center		**-573**±312	−901±228	−2472±347
	Density Based		**-477**±172	−1551±205	−2770±370
HC	Best	1419 ± 73			
	Mean		**3474**±822	**2205**±634	551±412
	Data Center		**2321**±618	**2040**±423	1332±83
	Density Based		**2205**±459	**1441**±195	1041±110

Besides that, in the *3 best* and *good* ensembles the Data Center strategy shows very good results when compared with the *best* DDPG, with the exception being the *good* group in the Cart Pole environment in Alternately

Table 5.4: Comparison Performance of Classical Aggregations Ensemble in DDPG with Continuous Action Space in Online Learning.

		DDPG	Online Learning		
			3 best	*good*	*bad*
PD	Best	−792 ± 14			
PD	Mean		−1786±369	−2836±280	−4252±109
PD	Data Center		**-794**±11	−854±24	−2221±326
PD	Density Based		−1412±229	−1607±266	−2834±249
CP	Best	−344 ± 86			
CP	Mean		−1333±336	−2103±243	−3409±290
CP	Data Center		**-291**±58	**-484**±167	−1760±345
CP	Density Based		**-344**±120	−1110±193	−1378±194
HC	Best	1419 ± 73			
HC	Mean		314±302	−53±17	−100±20
HC	Data Center		**2159**±342	**1942**±198	**1284**±294
HC	Density Based		**1424**±415	613±392	555±267

Persistent training mode. Data Center aggregation in the *bad* group of the Half Cheetah environment even shows performance almost on par with the best single strategy for both training modes.

On the other side, Density Based aggregation does not stand out from the single's best performance in training. Although it converges in all environment in the *3 best* group with Alternately Persistent training, its mean is always below the best other strategy.

To show the learning behavior, we present the learning curves of different aggregation techniques with the *good* group in all environments and both training modes. The Alternately Persistent learning is presented in Figure 5.1, with the Inverted Pendulum presenting distinct performances for each aggregation, while, in Figure 5.2, Cart Pole struggled to reach a better performance. In Figure 5.3, Half Cheetah showed a huge performance leap in the early seconds between the Data Center and Mean, yet both ended with an equal performance.

In Online learning, Data Center presented the best performance in all environments, mostly because the other aggregations present worse behavior in this mode. Although they start comparably, the performance becomes more erratic as time goes on.

Figure 5.4 compares the *best* algorithm learning curve with the *good* group, which was trained with Alternately Persistent learning in the Half Cheetah environment. Data Center learns comparably in the beginning, but surpasses the *best* performance later on. The Mean aggregation worked just as well at the end, even though it presented a low capacity in the initial stage of learning. Both show the possibility of further performance improvement if learning were extended.

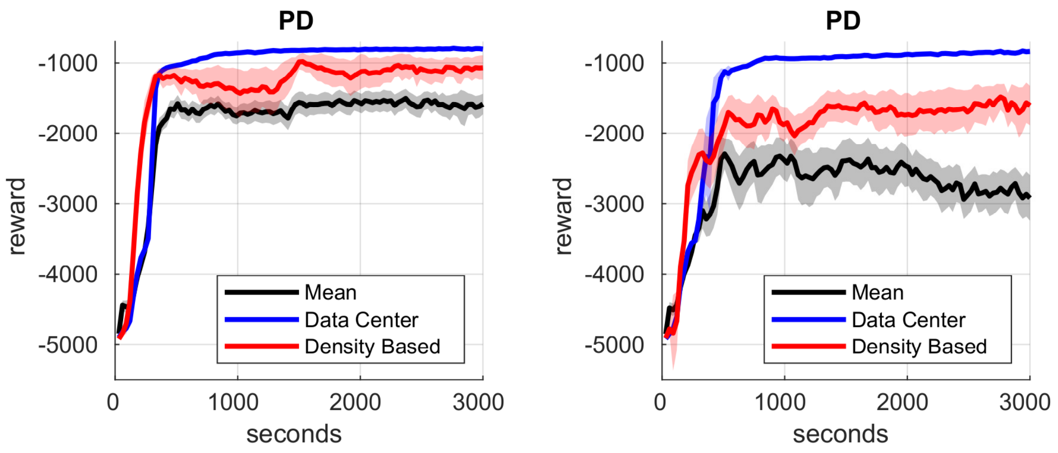

Figure 5.1: Alternately Persistent and Online Learning of *mostly good* Group in Inverted Pendulum Swing-up.

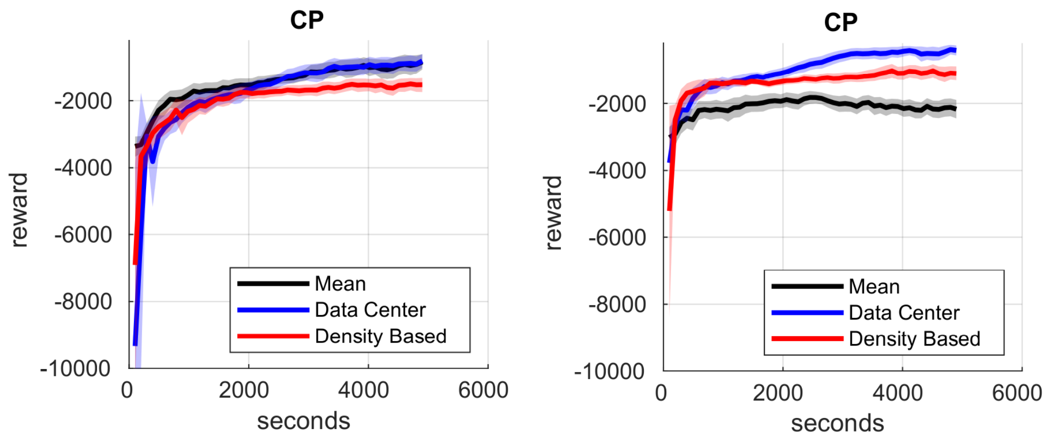

Figure 5.2: Alternately Persistent and Online Learning of *mostly good* Group in Cart Pole.

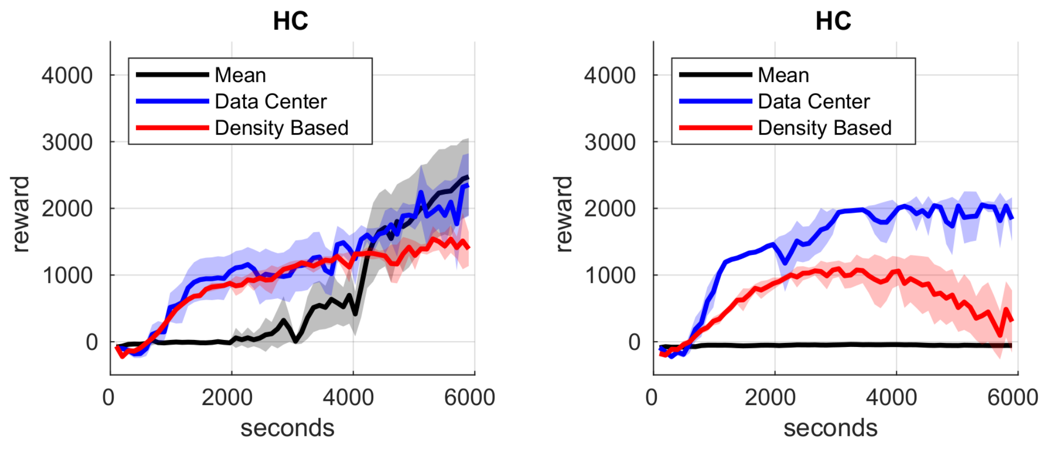

Figure 5.3: Alternately Persistent and Online Learning of *mostly good* Group in Half Cheetah v2.

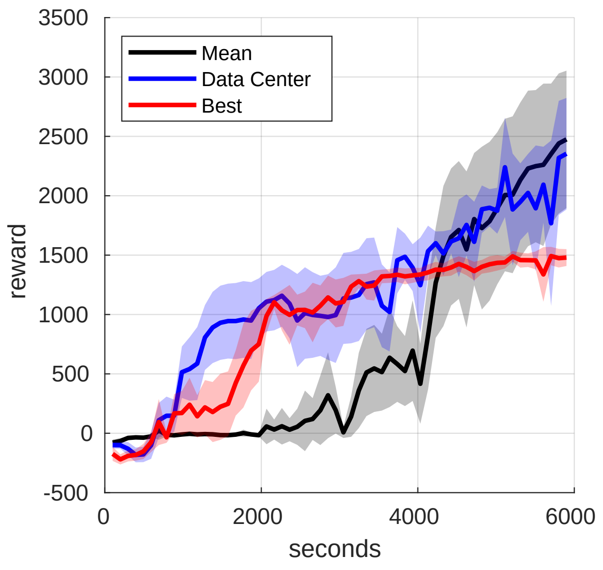

Figure 5.4: Half Cheetah v2 Performance for the *good* Hyperparameterizations in Alternately Persistent Learning.

5.3 History-Based Framework (HBF)

This section presents the results of experiments performed on the History Based Framework.

5.3.1 Ensemble of Pre-Trained Algorithms

In ensemble research, it is common to use previously trained algorithms, and this work follows this tradition: below are presented the tables containing pre-learned algorithm simulations. Table 5.5 presents the inverted pendulum environment, while Tables 5.6 and 5.7 present cart pole and cart double pole. All of them show that many of the presented ensemble strategies have an equal or better performance than the best base case (without history).

In the *good* group, almost all the strategies performed better than the best single policy, with the exception being the cart double pole environment, that, in this case, only DC-ED-DC stood out. Analyzing the tables, it is seen that the best results are found predominantly in the new strategies created by the History-Based Framework.

Looking at the *mid* group, it is interesting to note that some strategies maintain performance, although half of the algorithms participating in the ensemble do not learn. For the *bad* group, only the cart double pole maintains convergence, probably because, in this test case, the episodes start near the the equilibrium end point.

Table 5.5: Pendulum Results for Different Ensemble Strategies (31 runs).

	strategy	pre-learned learning			online learning		
		good	*mid*	*bad*	*good*	*mid*	*bad*
BASE	DC	**-779**±11*	-894±89	-2327±367	-850±25	-1075±80	-2244±321
BASE	DB	-846±75	-1406±189	-2553±279	-1594±264	-2633±342	-2810±254
BASE	M	-1416±115	-2851±116	-4043±62	-2835±286	-3574±172	-4234±108
	DCR-B	**-775**±10*	-874±176	-2388±428	-804±17*	-841±59	-2477±413
	DS-B	-794±22*	-1188±325	-3339±206	-1783±343	-2233±376	-3468±79
	M-ED-B	-1067±353	-1490±509	-3359±518	-810±18	-1358±442	-2343±584
	DCR-DC	**-770**±10*	-876±180	-2386±430	**-791**±13*	-1008±176	-2341±375
	DS-DB	-796±33*	-1471±398	-3433±135	-846±24	-1105±94	-2346±379
	M-ED-M	-793±20*	-884±63	**-2174**±486	-894±159	-1139±251	-2369±374
	M-ED-DC	**-778**±13*	**-786**±12*	**-1765**±489	**-790**±11*	**-783**±12*	**-1677**±444
	M-ED-DB	-782±11*	-799±13*	**-1713**±486	**-793**±10*	-861±77	**-1448**±306
	DC-ED-DC	**-775**±11*	-864±182	**-2066**±471	-806±14	-974±201	-1913±400

Table 5.6: Cart Pole Results for Different Ensemble Strategies (31 runs).

	strategy	pre-learned learning			online learning		
		good	*mid*	*bad*	*good*	*mid*	*bad*
BASE	DC	-392±51*	-507±162	**-1062**±247	-463±170	**-783**±261	-1684±364
BASE	DB	-931±138	-1189±144	-1582±183	-1077±206	-1317±279	-1322±214
BASE	M	-516±109	-878±222	-1637±233	-2153±254	-2805±249	-3411±294
	DCR-B	-306±54*	-571±230	-1180±326	-434±143	**-831**±319	-1895±435
	DS-B	-860±148	-1373±173	-2190±240	-920±272	-2020±379	-2316±365
	M-ED-B	-379±115*	-985±326	-2349±542	-379±99*	-1382±410	-2483±392
	DCR-DC	-275±54*	-550±220	-1141±313	-727±351	**-812**±303	-1450±440
	DS-DB	-380±54*	-510±157	**-1062**±249	-462±164	**-735**±288	-1338±289
	M-ED-M	-268±27*	**-363**±160*	**-924**±247	-632±158	-1851±431	-2624±504
	M-ED-DC	-244±20*	**-315**±144*	**-1083**±418	-296±94*	**-913**±441	-2474±604
	M-ED-DB	-263±24*	**-352**±157*	**-953**±269	**-236**±24*	**-766**±377	-1659±531
	DC-ED-DC	**-228**±15*	**-362**±170*	**-837**±291	-340±121*	**-677**±367	**-912**±291

Table 5.7: Cart Double Pole Results for Different Ensemble Strategies (31 runs).

	strategy	pre-learned learning			online learning		
		good	*mid*	*bad*	*good*	*mid*	*bad*
BASE	DC	609±10	554±27	**485**±32	565±42	392±60	**551**±33
BASE	DB	222±62	225±65	183±57	278±61	234±49	296±65
BASE	M	535±42	435±45	317±35	88±12	89±11	84±11
	DCR-B	603±18	502±42	436±43	**612**±2	**598**±12	**577**±26
	DS-B	292±64	258±58	218±53	523±51	368±52	477±48
	M-ED-B	577±27	477±51	406±47	552±41	419±52	287±48
	DCR-DC	**613**±3	509±39	435±39	**611**±2	**595**±12	**560**±27
	DS-DB	610±7	555±26	**499**±33	247±31	530±48	462±57
	M-ED-M	603±10	546±38	**479**±40	485±36	292±40	258±29
	M-ED-DC	**611**±3	557±33	**494**±40	606±7	571±25	467±45
	M-ED-DB	606±8	561±29	**488**±39	606±6	**599**±10	534±31
	DC-ED-DC	**614**±3*	**587**±24	**500**±41	606±5	**600**±10	**560**±27

Table 5.8: Half Cheetah v2 Validation Results (10 runs).

strategy	pre-learned learning			online learning		
	good	*mid*	*bad*	*good*	*mid*	*bad*
DC	1324±144	**1043**±249	1002±146	1945±222*	**1870**±201*	1325±322
DC-ED-DC	**1501**±39*	**817**±459	**1166**±83	**2057**±72*	**1962**±229*	**1515**±174*

5.3.2 Ensemble Online Training

In the ensemble online training, the individual DDPG policies are trained from scratch during execution, learning together. Table 5.5 presents the inverted pendulum environment, while Table 5.6 and 5.7 present the cart pole and cart double pole, again showing that many of the presented ensemble strategies have an equal or better performance than the best base case, mainly for the *good* group.

In the *mid* group, cart pole does not present good convergence. The *bad* group of cart pole presents performance compared to *mid* for the DC-ED-DC strategy. This environment performs better when policies are more consistent. The DC-ED-DC strategy presented the lowest error as calculated by Eq. (3-3), having a 2.3% average relative regret.

Figures 5.5 and 5.6 show the learning curves of the DC and DC-ED-DC strategies.

5.3.3 Half Cheetah Test

Half Cheetah [23] is a complex environment, chosen to validate the strategy of the online continuous action ensembles. It seeks the goal of a bipedal walk, and its environment is tested with DC (best base case strategy) and DC-ED-DC (best strategy with the lowest error method as described in the Section 4.3) for the comparison of results. Table 5.8 presents the pre-learned and the online learning ensemble strategies' performance in the Half Cheetah environment respectively. The noteworthy result is that Half Cheetah learns consistently better for online learning when compared to the best individual simulations. Considering the best single parameterization in our set, the DC-ED-DC online learning strategy shows 45% improvement in the *good* group (2057$\pm$72 vs. 1419$\pm$73), proving that. Overall, the DC-ED-DC is consistently better, or at least equal to *mid* group, the base case DC. Besides that, training policies online presents much better results than using pre-learned policies.

Finally, to test the decreased hyperparameter tuning effort in our approach, random continuous values of the hyperparameters' intervals described in Section 3.3 were generated. Note that, due to the choice's randomness, the chosen parameters naturally tend to fall more in the middle than at the range's extremes, generating more acceptable average values (Table 4.1). For a random test, 30 ensembles with 16 parameterizations were generated and run once. In the previous experiments, the formation for groups always ensured that at least 4 algorithms (*good groups*) were not able to converge, opposed to the random

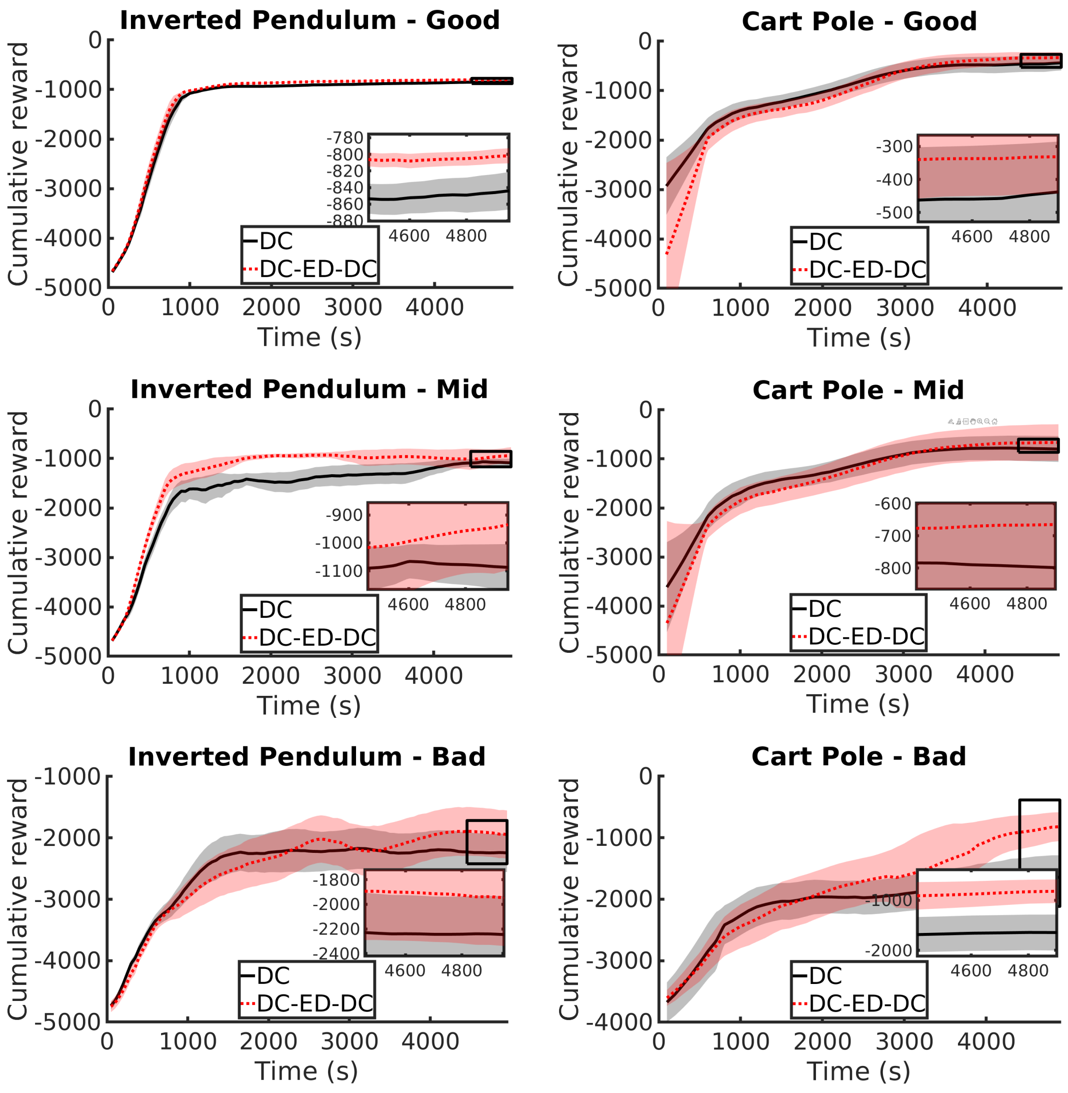

Figure 5.5: Illustration of DC and DC-ED-DC Learning Curves of PD and CP environment.

generation ones, where no such control was made. The random Half Cheetah v2 hyperparameter ensemble, using the DC-ED-DC strategy, resulted in the mean performance of 3684±626.

5.3.4 Discussion

Although in the literature the Density Based aggregation strategy shows better results in the performed experiments [14, 15, 16], the case studies analyzed in this thesis have shown more consistent Data Center results. One reason may be the need to parameterize Density Based algorithm, which may have negatively influenced the performance. An other consideration is that both the off-policy algorithm and the environments are different, yet there are good results in the combined framework strategies that include the Density

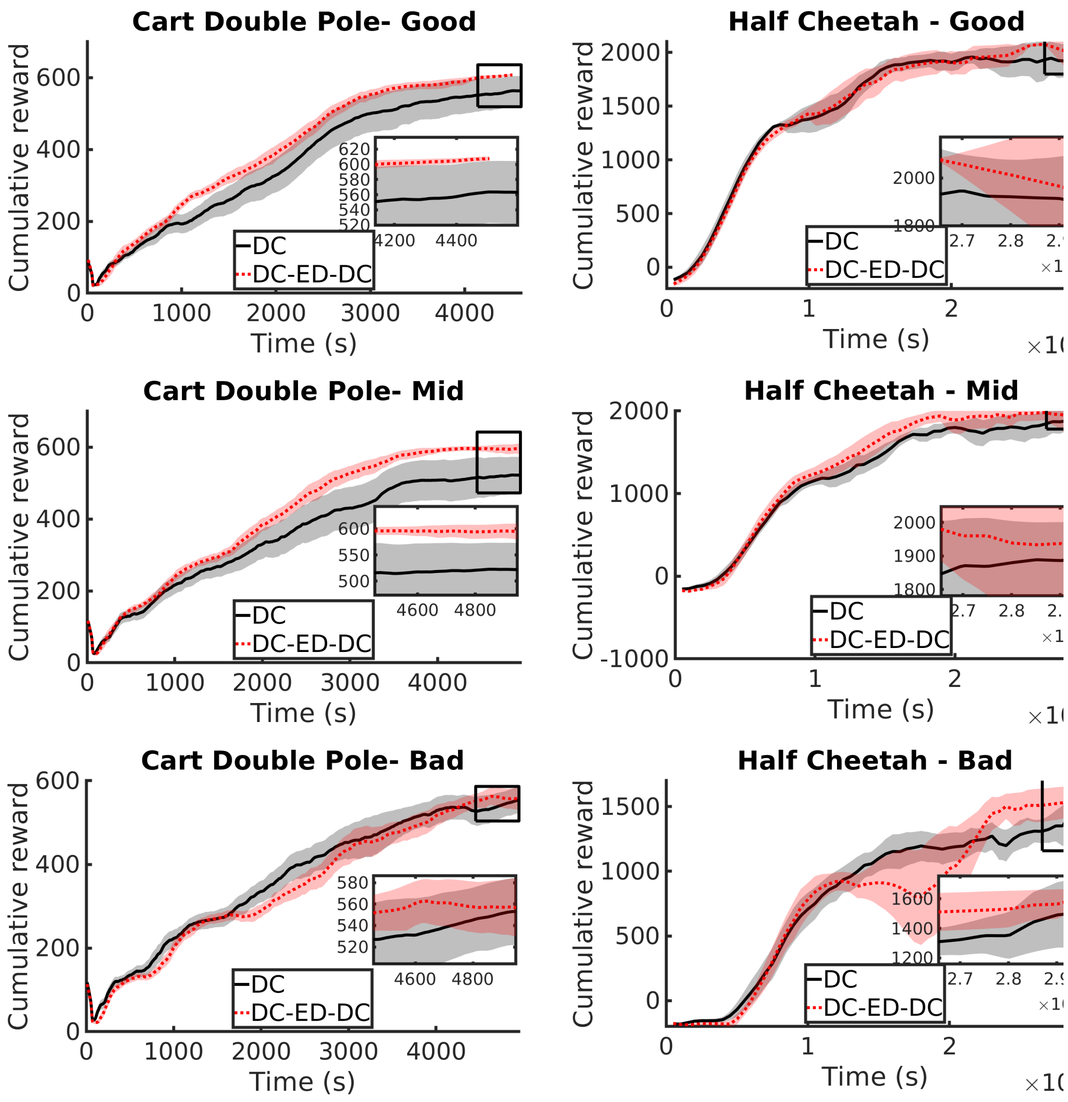

Figure 5.6: Illustration of DC and DC-ED-DC Learning Curves of CDP and HC environment.

Based algorithm.

Regarding the separation of the tests into groups, the *good* group presented better results, as expected. Another point is that the best ensemble strategies in the *good* group responded considerably better than the best individual algorithm, while the Density Based strategy performed less consistently; that means that the best ensemble results improve robustness and accuracy as expected [47]. Although the best strategy varies with the group and environment, DC-ED-DC showed more consistent overall performance than all other strategies, including the baselines.

Looking at the other groups, the *mid* group was, for some cases, able to keep up with the expected *good* performance, allowing significant leeway in the choice of hyperparameter realizations. In the *bad* group, the cart double pole and Half Cheetah environments presented a surprising performance, comparable to the best tuned run. The former environment already started in the final position equilibrium, while the latter showed the ability to maintain excellent performance over the best individual algorithm, mainly because of the 45% improvement demonstrated in the *good* group.

The final fully randomized validation test demonstrated the performance of hyperparameters found randomly within the composition expected in a realistic DDPG implementation. In the Half Cheetah environment, it can be seen that the strategies performed surprisingly well compared to the single run.

5.4 Online Weighted Q-Ensembles (OWQE)

In this section, the simulation results of the OWQE are presented. Figure 5.7 shows the final performance and its confidence interval. Each bar graph compares the performance on the 4 ensemble groups for the different aggregations, with separate graphs showing distinct environments and training modes.

It is observed that the online training mode almost always outperforms alternate training; furthermore, there is no significant variation between the aggregations in the *3 Best* ensemble performance. Ensembles with a majority of bad parameterizations perform worse, which is especially evident in the more complex Half Cheetah v2 environment.

In general, the *Softmax TDError* aggregation performs better, or within the confidence interval, than the other aggregations with the *1 Good 3 Bad* performance, on par with the single best and *3 Best* ensemble even in the Half Cheetah v2 environment.

Regarding the single ablations, there is no obvious trend as for which has the better performance. In fact, sometimes they perform worse than simple Q-averaging. However, looking at the average relative regret presented in Table 5.9, the critic weighing (*TDError* aggregation) has a larger influence than Boltzmann averaging (*Softmax Average*), although it is clear that both are required for our model's final performance.

Table 5.9: Average relative regret result – OWQE – Eq. (3-3) over the pendulum, cart pole and Half Cheetah v2 environments and all ensemble groups, using online training.

Average	***Softmax Average***	***TDError***	***Softmax TDError***
5.5059	6.0599	4.9232	2.3890

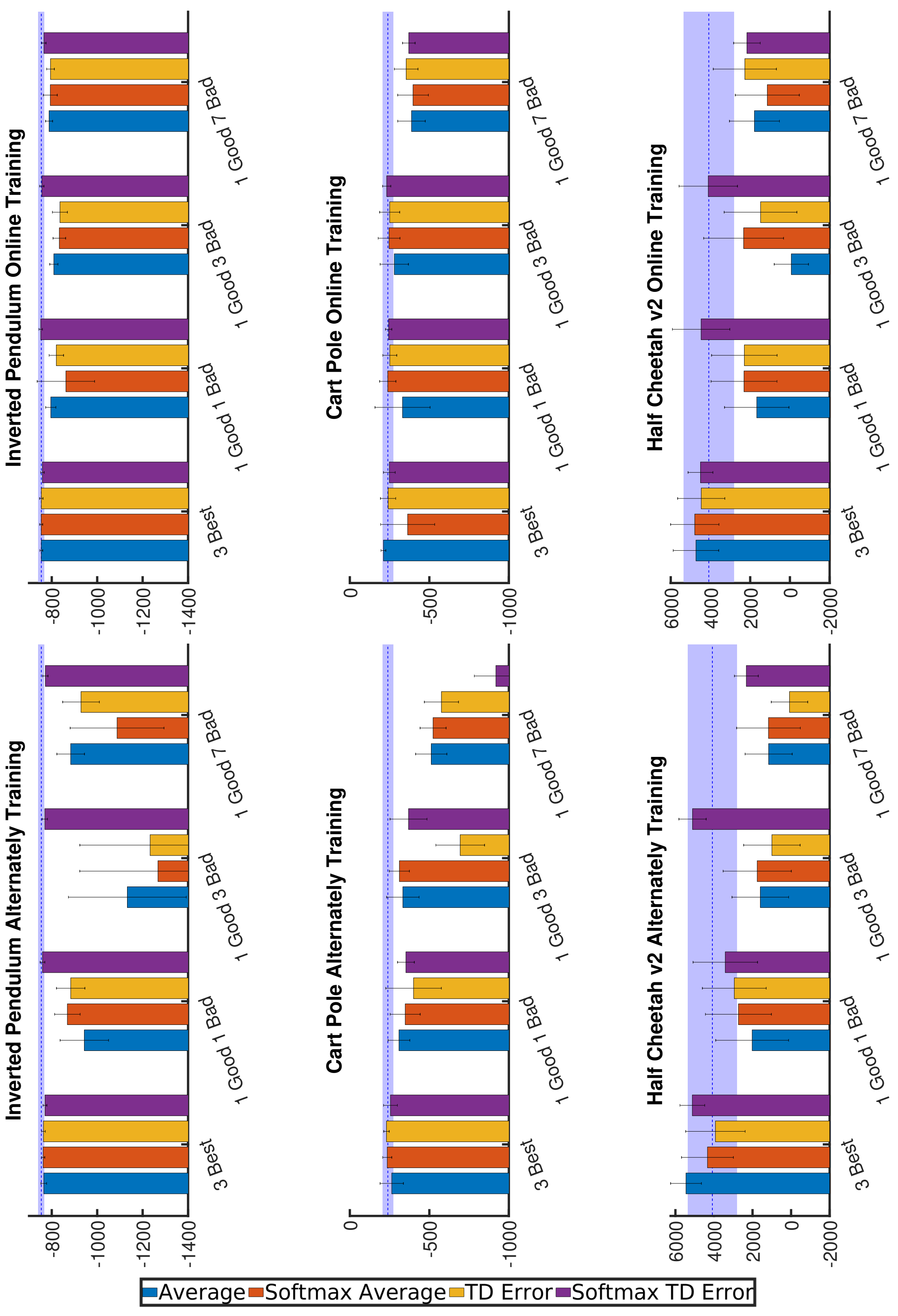

Figure 5.7: Performance of Weighted DDPG Ensemble Model comparing groups (*3 Best, 1 Good 1 Bad, 1 Good 3 Bad and 1 Good 7 Bad*) and Q-Aggregation (*Average, Softmax Average, TD Error, Softmax TD Error*). Columns separate the Alternately and Online training and the lines present the environments: inverted pendulum, cart pole, Half Cheetah v2 and swimmer v2. The graphic bar also shows the error bar of the 95% confidence interval. The horizontal line marks the mean of 10 single run and the shadows area represents the 95% confidence interval

The Swimmer v2 validation uses 30 different ensembles, each with 8 randomly generated parameterizations. The performance of the full model (*Softmax TDError* aggregation) was compared with simple Q-averaging, using online training. The mean and confidence intervals are 85 ± 13 for averaging, and 110 ± 18 for our model, showing a significant improvement.

5.4.1 Learning Curve

Figure 5.8 shows the online training mode learning curves of the *1 Good 1 Bad* and *1 Good 3 Bad* ensembles for the *Average* and *Softmax TDError* aggregations. In the inverted pendulum, *Softmax TDError* learns a bit faster than *Average*, while in the cart-pole this behavior is reversed. In both environments learning is stable with good end performance, with *Softmax TDError* having higher mean and lower final variance.

In Half Cheetah v2, the performance difference is huge. Both curves show high variance, but in both ensemble groups *Softmax TDError* performs much better. Specifically, *Average* does not manage to learn in the *1 Good 3 Bad* ensemble, while *Softmax TDError* maintains almost the same performance as in the *1 Good 1 Bad* group.

5.4.2 Action Preference and Q-Weights

To better understand our model's behavior, an investigation of how the actions are chosen was made. Figure 5.9 presents the behavior of the Half Cheetah v2 MuJoCo environment, where in the left column are the assigned weights to each critic — Q-Weights ($\mathbf{W}$), and in the right the percentage of different actors' actions chosen in the episode. All bad parameterizations have the same color, while the good ones are highlighted.

At the beginning of the learning process, the $\mathbf{W}$ are uniformly distributed, and at the end the weights tend to choose the critic with the lowest calculated TD Error. The same process happens with the counted actions; the beginning of the learning process has shown an equal distribution of the chosen actions, while at the end the choice of actions is influenced by the critics' acquired Q-Weights.

The first row shows the *3 Best* group, where each parameterization has a different color. Intuitively, the expectation is that both weights and action choices should stay equally distributed, since all parameterizations have roughly the same individual performance. Initially, this is not the case, as some choices may learn faster than others, but, at least for the actions, the end result

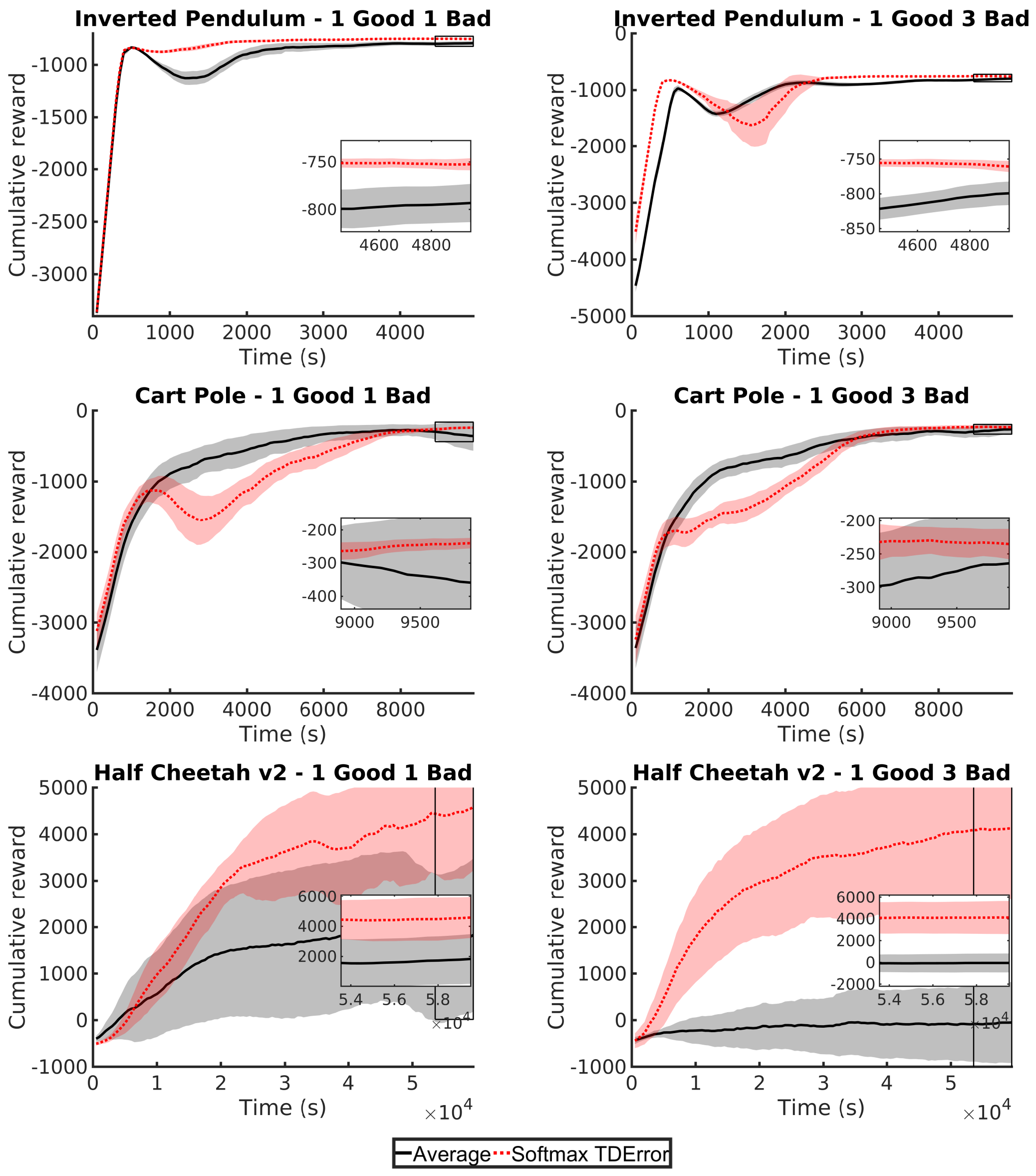

Figure 5.8: Learning Curve of Online Weighted Q-Ensemble with *1 Good 1 Bad, and 1 Good and 3 Bad* and Q-Aggregation (*Average, Softmax TD Error*). All cases are online training, lines present the environments: inverted pendulum, cart pole and Half Cheetah v2. The graphic also shows the 95% confidence interval.

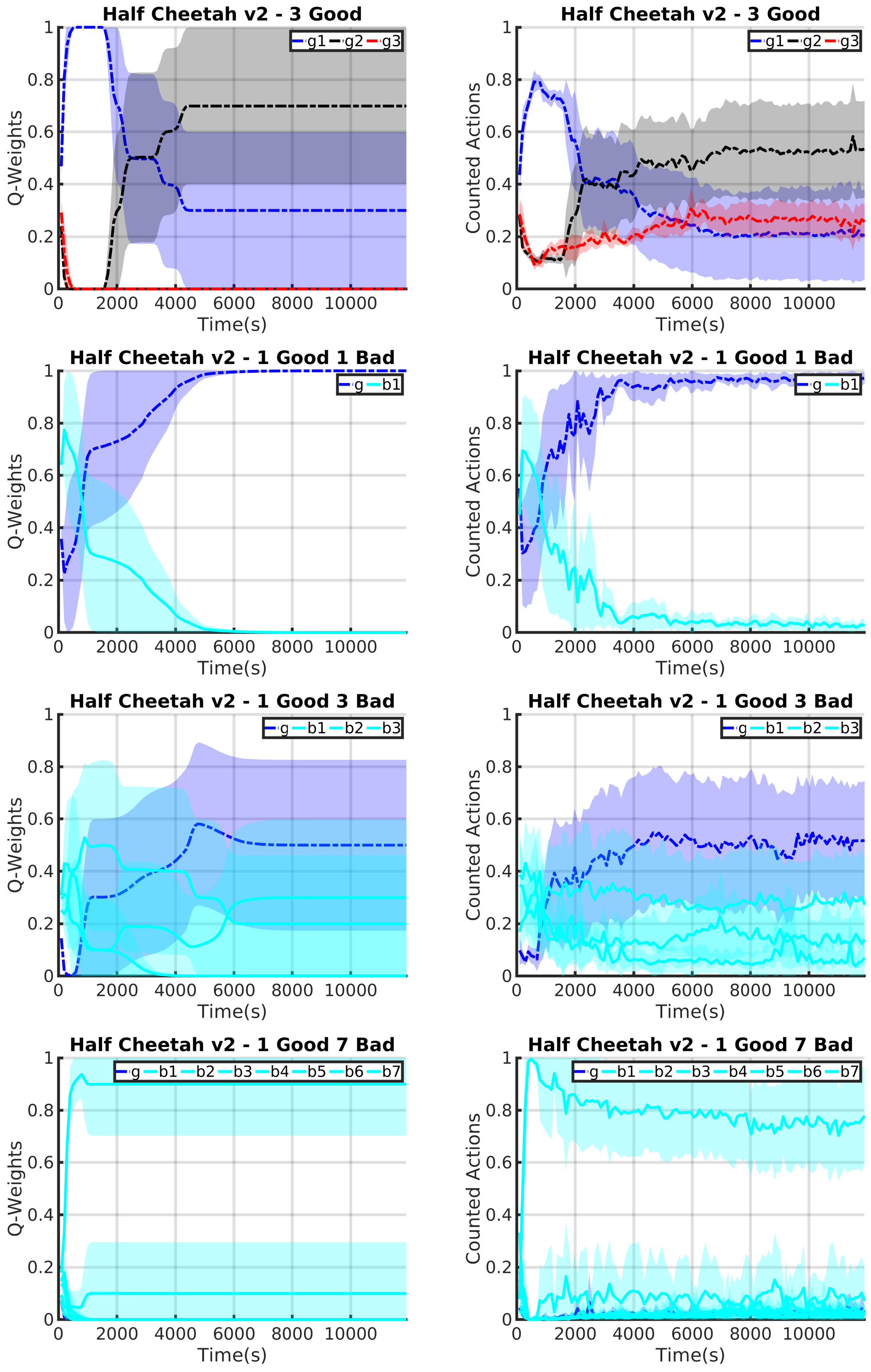

Figure 5.9: Counter Action and Q-Weights of Online Weighted Q-Ensemble with *3 Best, 1 Good 1 Bad, 1 Good and 3 Bad, and 1 Good and 7 Bad* and Q-Aggregation *Softmax TD Error*). All cases are online training in Half Cheetah v2 environment. The graphic also shows the 95% confidence interval.

is as expected. Note that one critic has a very low weight, but it's actor's action is chosen normally (red line). This shows that having a higher TD-error critic does not always imply on a worse actor.

The second row presents the *1 Good 1 Bad* group, which is composed of two opposing models algorithm, with an expectation that the good algorithm will be chosen from the beginning. Indeed, there is a very clear distinction from the Q-Weights, that is reflected in the choice of actions with little noise. The third row of Figure 5.9 presents the *1 Good and 3 Bad* group, which behaves similarly to the *1 Good 1 Bad*, but with more challenges, as there are more bad parameterizations to compete with.

Finally, the *1 Good and 7 Bad* group shown in the last row struggles to find the good agent. The Q-Weights do not manage to converge to the best individual critic, nor is its action chosen more often than the others. However, the results in Figure 5.7 show that the ensemble still reaches an adequate (although not optimal) performance.

Overall, the best individual agent has both lower weight and its actions are generally chosen less in larger ensembles. Even so, there is an improvement in the final performance when *Softmax TDError* is used.

5.5 Discussions

In the results presented, it was clear that the classic strategies ensembles are only good for 3 best. In the History-Based Framework, an action ensemble , there is a behavior change because aggregations that work well in good groups are presented, that is, when there are 25% of non-converging algorithms. Adding the q-value information in the selection of the ensemble's action by Online Weighted Q-Ensemble, it was noticed that it achieves good results even with a large majority of non-converging algorithms. For this, groups with only 1 convergent algorithm were tested with varied numbers of non-convergent algorithms.

All experiments are intended to demonstrate the use of the amount of environment interactions. Therefore, there are no investigations into the cost-benefit of computational effort. All experiments were performed on workstations with or without a dedicated video card for processing. Table 5.10 presents a reference of computational effort, the machine used in this example is an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, with a GeForce GTX 1080 8GB and 16 GB Ram Memory.

Table 5.10: Computational effort reference in Online Weighted Q-Ensemble using a workstation configured with an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, with a GeForce GTX 1080 8GB and 16 GB Ram Memory.

Environment	**Group**	**Aggregation**	**Time (min)**
Inverted Pendulum Swing-up	Single	—	22
Inverted Pendulum Swing-up	*1 Good 1 Bad*	*Softmax TDError*	53
Inverted Pendulum Swing-up	*1 Good 7 Bad*	*Softmax TDError*	107
Half Cheetah v2	Single	—	145
Half Cheetah v2	*1 Good 1 Bad*	*Softmax TDError*	482
Half Cheetah v2	*1 Good 7 Bad*	*Softmax TDError*	1970

6
Conclusions and Future Proposals

The comparison with the classical aggregation in continuous action spaces demonstrates that it is possible to improve the grid search performance with the use of a DDPG ensemble. When testing an ensemble with *3 Best* DDPG, the best aggregation cannot be highlighted; however, when we use the *good group*, it is realized that the Data Center strategy presents good performance and the narrowest confidence interval. When using algorithms with unreliable hyperparameter tuning, the *bad group*, the Data Center demonstrated its capability to learn as well as fine-tuning a single agent, but only in the Half Cheetah environment. The baseline algorithms are good for ensembles of good parameterizations, but performance almost always goes downhill if bad ones are included.

The study regarding continuous action ensembles with the History-Based Framework (HBF) propose learning control policies from scratch, demonstrating that the hyperparameter tuning need a reduction in DDPG algorithms when tested with the Half Cheetah environment. In order to achieve this result, a history-based framework considering the ensemble's historical performance was introduced, with a capability of capturing different compositions of ensemble strategies, presenting the ones that performed best. The best strategy, DC-ED-DC, takes the squared Euclidean distance of each action from the data center's action center and performs the moving average by updating the $\overline{scores}$, selecting 25% of the best scoring ones and applying the data center algorithm again. A comparison was made with state-of-the-art ensemble strategies, and it demonstrated that the chosen strategy outperforms the baseline algorithms. It also demonstrated the advantage in simulation of using an ensemble algorithm over individual algorithms. In the classical environments, pre-learned strategies mostly showed better results than online learning; however, evaluating in the Half Cheetah environment, it is shown that the approach to online learning policies made a very significant difference, improving performance for both the base case and the best strategies (DC-ED-DC). The HBF was designed to work with online learning and a single environment, since it has advantages in the system's applicability in real-world robotic applications.

Adding the q-value information in the selection of the ensemble's action, the Online Weighted Q-Ensemble (OWQE) was designed to decrease the hyperparameter tuning effort, using q-values continuous action spaces. Based on previous works, which use an average of Q-ensembles in an actor-critic setting, we introduced a weighing approach that adjusts the critics' weights by minimizing the temporal difference error of the ensemble as a whole. Additionally, instead of combining the Q-values directly, they were applied through a softmax layer, in order to focus on relative preferences rather than absolute values. In both simple and complex robotic simulation environments, our model showed better results than the standard Q-value averaging, and managed to maintain performance comparable to the best individual run even if the ensemble included up to 3-7 bad parameterizations. Validation using ensembles with 8 randomized parameterizations also showed superior performance compared to the q-value averaging. Such as before, it was aimed at the system's applicability in real-world robotic applications, so the tests used a single environment.

In future works, it would be interesting to extend the simulations to more environments, evolving the validations for use in real robots. Other interesting points to be further expanded in possible subsequent works are the acceleration of learning from the beginning curves and the extension of tests with further algorithms, such as TD3 and SAC. One of the perceived limitations of the HBF was the lack of q-values (value functions) usage; therefore, the OWQE was developed, in order to fill this gap and improve the ensemble's performance. Upcoming work on OWQE may include the gamma and reward scale hyperparameters optimizations. Furthermore, it learns in the OWQE *1 Good 7 Bad group* evaluations, but with great variations in performance when compared between environments. As the real world has only one real environment, this entire study was directed to it, but, for future researches, there are interesting possibilities, e.g., the improvement of performances for all environments or the use of multi environments in order to accelerate learning techniques, such as transfer learning.

Bibliography

[1] SUTTON, R. S.; BARTO, A. G.. **Introduction to Reinforcement Learning**. MIT Press, Cambridge, MA, USA, 2018.

[2] LIU, R.; NAGEOTTE, F.; ZANNE, P.; DE MATHELIN, M. ; DRESP-LANGLEY, B.. **Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review**. Robotics, 10(1):22, 2021.

[3] WATTS, J.; KHOJANDI, A.; VASUDEVAN, R. ; RAMDHANI, R.. **Optimizing individualized treatment planning for parkinson's disease using deep reinforcement learning**. In: 2020 42ND ANNUAL INTERNATIONAL CONFERENCE OF THE IEEE ENGINEERING IN MEDICINE & BIOLOGY SOCIETY (EMBC), p. 5406–5409. IEEE, 2020.

[4] KHALILPOURAZARI, S.; DOULABI, H. H.. **Designing a hybrid reinforcement learning based algorithm with application in prediction of the covid-19 pandemic in quebec**. Annals of Operations Research, p. 1–45, 2021.

[5] OLIVEIRA., R. G.; CAARLS., W.. **Comparing action aggregation strategies in deep reinforcement learning with continuous action**. In: ANAIS DO XXIII CONGRESSO BRASILEIRO DE AUTOMáTICA - VOLUME 2 NO 1: CBA 2020,. SBA, 2020.

[6] BERTRAND, H.. **Hyper-parameter optimization in deep learning and transfer learning: applications to medical imaging**. PhD thesis, Université Paris-Saclay, 2019.

[7] LI, L.; JAMIESON, K.; DESALVO, G.; ROSTAMIZADEH, A. ; TALWALKAR, A.. **Hyperband: A novel bandit-based approach to hyperparameter optimization**. Journal of Machine Learning Research, 04 2018.

[8] CARDEÑOSO FERNANDEZ, F.; CAARLS, W.. **Parameters tuning and optimization for reinforcement learning algorithms using evolutionary computing**. In: INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS AND COMPUTER SCIENCE (INCISCOS). HTTPS://DOI.ORG/10.1109/INCISCOS.2018.00050, 2018.

[9] WISTUBA, M.; SCHILLING, N. ; SCHMIDT-THIEME, L.. **Sequential model-free hyperparameter tuning.** In: 2015 IEEE INTERNATIONAL CONFERENCE ON DATA MINING. HTTPS://DOI.ORG/10.1109/ICDM.2015.20, Nov 2015.

[10] JADERBERG, M.; DALIBARD, V.; OSINDERO, S.; CZARNECKI, W. M.; DONAHUE, J.; RAZAVI, A.; VINYALS, O.; GREEN, T.; DUNNING, I.; SIMONYAN, K. ; OTHERS. **Population based training of neural networks.** CoRR, abs/1711.09846, 2017.

[11] KHADKA, S.; MAJUMDAR, S.; NASSAR, T.; DWIEL, Z.; TUMER, E.; MIRET, S.; LIU, Y. ; TUMER, K.. **Collaborative evolutionary reinforcement learning.** In: PROCEEDINGS OF THE 36TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, volumen 97 de **Proceedings of Machine Learning Research**, p. 3341–3350, Long Beach, California, USA, 2019. PMLR.

[12] MEIJDAM, H. J.; PLOOIJ, M. ; CAARLS, W.. **Learning while preventing mechanical failure due to random motions.** In: 2013 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, p. 182–187. IEEE, 2013.

[13] TAN, J.; ZHANG, T.; COUMANS, E.; ISCEN, A.; BAI, Y.; HAFNER, D.; BOHEZ, S. ; VANHOUCKE, V.. **Sim-to-real: Learning agile locomotion for quadruped robots.** arXiv preprint arXiv:1804.10332, 2018.

[14] WIERING, M. A.; VAN HASSELT, H.. **Ensemble algorithms in reinforcement learning.** IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 38:930–936, 2008.

[15] HANS, A.; UDLUFT, S.. **Ensembles of neural networks for robust reinforcement learning.** In: 2010 NINTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND APPLICATIONS, p. 401–406. IEEE, 2010.

[16] DUELL, S.; UDLUFT, S.. **Ensembles for continuous actions in reinforcement learning.** In: ESANN, 2013.

[17] WU, J.; LI, H.. **Deep ensemble reinforcement learning with multiple deep deterministic policy gradient algorithm.** Mathematical Problems in Engineering, 6:1–12, 2020.

[18] FAUSSER, S.; SCHWENKER, F.. **Ensemble methods for reinforcement learning with function approximation**. In: INTERNATIONAL WORKSHOP ON MULTIPLE CLASSIFIER SYSTEMS. Springer, 2011.

[19] ANSCHEL, O.; BARAM, N. ; SHIMKIN, N.. **Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning**. p. 176–185, 2017.

[20] CHEN, R. Y.; SIDOR, S.; ABBEEL, P. ; SCHULMAN, J.. **Ucb exploration via q-ensembles**. arXiv preprint arXiv:1706.01502, 2017.

[21] OLIVEIRA., R. G.; CAARLS., W.. **A history-based framework for online continuous action ensembles in deep reinforcement learning**. In: PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON AGENTS AND ARTIFICIAL INTELLIGENCE - VOLUME 2: ICAART,, p. 580–588. INSTICC, SciTePress, 2021.

[22] OLIVEIRA., R. G.; CAARLS., W.. **A online weighted q-ensembles for deep reinforcement learning in continuous action spaces**. In preparation, 2021.

[23] TODOROV, E.; EREZ, T. ; TASSA, Y.. **Mujoco: A physics engine for model-based control**. In: 2012 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, p. 5026–5033. IEEE, 2012.

[24] BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J. ; ZAREMBA, W.. **Openai gym**. CoRR, 2016.

[25] MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S. ; HASSABIS, D.. **Human-level control through deep reinforcement learning**. Nature, 518(7540):529–533, 02 2015.

[26] KINGMA, D. P.; BA, J.. **Adam: A method for stochastic optimization**, 2017.

[27] LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D. ; WIERSTRA, D.. **Continuous control with deep reinforcement learning**. Proceedings of International Conference on Learning Representations, 2016.

[28] SILVER, D.; LEVER, G.; HEESS, N.; DEGRIS, T.; WIERSTRA, D. ; RIEDMILLER, M.. **Deterministic policy gradient algorithms**. In: PROCEEDINGS OF THE 31ST INTERNATIONAL CONFERENCE ON INTERNATIONAL CONFERENCE ON MACHINE LEARNING. HTTP://DL.ACM.ORG/CITATION.CFM?ID=3044805.3044850, 2014.

[29] LIU, R.; ZOU, J.. **The effects of memory replay in reinforcement learning**. In: 56TH ANNUAL ALLERTON CONFERENCE ON COMMUNICATION, CONTROL, AND COMPUTING (ALLERTON). IEEE, 2018.

[30] UHLENBECK, G. E.; ORNSTEIN, L. S.. **On the theory of the brownian motion**. Physical review, 36:823, 1930.

[31] ERNST, D.; GEURTS, P. ; WEHENKEL, L.. **Tree-based batch mode reinforcement learning**. Journal of Machine Learning Research, 6:503–556, 2005.

[32] REINKE, C.; UCHIBE, E. ; DOYA, K.. **Average reward optimization with multiple discounting reinforcement learners**. In: INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING, p. 789–800. Springer, 2017.

[33] VAN HASSELT, H.; GUEZ, A. ; SILVER, D.. **Deep reinforcement learning with double q-learning**. In: PROCEEDINGS OF THE AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, volumen 30, 2016.

[34] MENON, R. R.; RAVINDRAN, B.. **Shared learning: Enhancing reinforcement in q-ensembles**. arXiv preprint arXiv:1709.04909, 2017.

[35] LINDENBERG, B.; NORDQVIST, J. ; LINDAHL, K.-O.. **Distributional reinforcement learning with ensembles**. Algorithms, 13(5), 2020.

[36] SAPHAL, R.; RAVINDRAN, B.; MUDIGERE, D.; AVANCHA, S. ; KAUL, B.. **Seerl: Sample efficient ensemble reinforcement learning**. arXiv preprint arXiv:2001.05209, 2020.

[37] HUANG, Z.; ZHOU, S.; ZHUANG, B. ; ZHOU, X.. **Learning to run with actor-critic ensemble**. arXiv preprint arXiv:1712.08987, 2017.

[38] ZHENG, Z.; YUAN, C.; LIN, Z.; CHENG, Y. ; WU, H.. **Self-adaptive double bootstrapped ddpg**. In: PROCEEDINGS OF THE TWENTY-SEVENTH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI-18, p. 3198–3204. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[39] KRISLOCK, N.; WOLKOWICZ, H.. **Euclidean distance matrices and applications**. In: HANDBOOK ON SEMIDEFINITE, CONIC AND POLYNOMIAL OPTIMIZATION. Springer, 2012.

[40] FREUND, Y.; SCHAPIRE, R. E. ; OTHERS. **Experiments with a new boosting algorithm**. In: ICML, volumen 96, p. 148–156. Citeseer, 1996.

[40] TRIOLA, M. F.. **Elementary Statistics Technology Update**. Pearson, 11 edition, 2015.

[41] BUSONIU, L.; BABUSKA, R.; DE SCHUTTER, B. ; ERNST, D.. **Reinforcement learning and dynamic programming using function approximators**. CRC press, 2017.

[42] BARTO, A. G.; SUTTON, R. S. ; ANDERSON, C. W.. **Neuronlike adaptive elements that can solve difficult learning control problems**. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13(5):834–846, Sep. 1983.

[43] NAGENDRA, S.; PODILA, N.; UGARAKHOD, R. ; GEORGE, K.. **Comparison of reinforcement learning algorithms applied to the cart-pole problem**. In: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATIONS AND INFORMATICS (ICACCI). IEEE, 2017.

[44] ZHONG, W.; ROCK, H.. **Energy and passivity based control of the double inverted pendulum on a cart**. In: PROCEEDINGS OF THE 2001 IEEE INTERNATIONAL CONFERENCE ON CONTROL APPLICATIONS (CCA'01), p. 896–901. IEEE, 2001.

[45] WAWRZYNSKI, P.. **Learning to control a 6-degree-of-freedom walking robot**. In: EUROCON 2007-THE INTERNATIONAL CONFERENCE ON "COMPUTER AS A TOOL", p. 698–705. IEEE, 2007.

[46] COULOM, R.. **Reinforcement learning using neural networks, with applications to motor control**. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2002.

[47] ZHANG, C.; MA, Y.. **Ensemble machine learning: methods and applications**. Springer, 2012.

A
Appendix — Single Run

Table A.1: Parametrizations by Environment and Their Performance in Action Ensemble.

	Inverted Pendulum	**Cart Pole**	**Cart Double Pole**	**Half Cheetah**
BEST	−792±14	−344±86	616±1	1419±73
	−794±15	−378±112	602±10	1407±333
	−797±16	−472±153	588±19	1378±85
	−801±22	−484±303	585±34	1275±69
	−803±18	−509±204	585±15	1267±300
	−808±19	−538±284	575±36	1255±115
	−815±20	−613±191	558±56	1247±150
	−816±24	−682±387	538±31	1235±107
	−818±26	−769±351	531±57	1208±150
	−820±22	−952±363	529±59	1194±122
	−824±23	−984±208	500±36	1145±203
	−827±22	−1033±269	495±44	1135±127
MIDDLE	−873±173	−1202±300	481±79	1119±115
	−875±30	−1207±495	476±64	1112±140
	−906±172	−1300±377	473±70	1100±126
	−986±53	−1391±496	451±87	1096±170
	−996±61	−1931±453	427±39	1084±388
	−1245±314	−2048±415	423±52	991±162
	−1723±442	−2626±390	380±58	883±165
	−3386±269	−2637±540	357±59	809±241
WORST	−3508±0	−2860±737	343±93	775±286
	−3508±0	−3071±577	330±53	758±194
	−3508±0	−3366±574	284±31	708±205
	−3508±0	−4037±1686	282±37	629±325
	−3508±0	−4274±2297	275±100	602±350
	−3508±0	−4572±84	269±35	525±351
	−3523±6	−4599±72	266±32	483±295
	−3585±47	−4645±61	262±30	262±258
	−3605±29	−4695±17	241±28	215±250
	−3653±76	−4704±15	230±42	138±245
	−3897±120	−4709±12	180±87	−14±24
	−3962±107	−4712±13	30±11	−29±10

Table A.2: Parametrizations by Environment and Their Performance used in Online Weighted Q-Ensemble.

	Inverted Pendulum	**Cart Pole**	**Half Cheetah**
BEST	−754±13	−239±34	4084±1271
	−766±13	−271±61	3790±0
	−769±20	−293±64	3194±1551
MIDDLE	−781±67	−386±220	3094±647
	−781±16	−409±148	2651±0
	−791±30	−480±279	2551±706
	−801±28	−529±309	2548±1181
	−805±31	−725±872	2158±765
	−808±29	−738±865	1988±656
	−885±61	−1263±167	1897±257
	−925±45	−1337±264	1795±199
	−964±49	−1350±210	1722±215
	−1190±50	−1401±169	1712±232
WORST	−1293±773	−1424±300	1623±556
	−1318±109	−1445±292	1414±263
	−1653±727	−1498±256	1397±266
	−1691±811	−1617±326	1292±323
	−2280±814	−1716±1008	873±723
	−2846±977	−2000±1437	−240±0
	−4030±964	−2364±929	−583±94