TECHNICAL UNIVERSITY OF CRETE

Technical University of Crete
School of Electrical and Computer Engineering
Intelligent Systems Laboratory

A Multi-Modal Q-Learning Approach using Normalized Advantage Functions and Deep Neural Networks

Diploma Thesis by

Christos P. Petridis

Submitted in partial fulfilment of the requirements for the Diploma of Electrical and Computer Engineer at the Technical University of Crete

To be defended publicly on Friday, 23th of August at 09:00
against the thesis committee consisting of

Associate Professor Michail G. Lagoudakis
School of Electrical and Computer Engineering

Associate Professor Georgios Chalkiadakis
School of Electrical and Computer Engineering

Professor Aggelos Bletsas
School of Electrical and Computer Engineering

This page was intentionally left blank

Abstract

Reinforcement Learning, a branch of Machine Learning geared towards the development of Autonomous Agents, presents a rapid evolution in recent years as a means of solving sequential decision problems. The development of robust Deep Neural Networks has also played a crucial role to this success. The combination of these two areas eventually led to Deep Reinforcement Learning, a state-of-the-art field which demonstrated already a great potential and tremendous results in continuous control tasks. In order to contribute to this effort, the present thesis investigates an extension of the Normalized Advantage Functions (NAFs) to multi-modal representations, such as multiple quadratics and RBFs (Radial Basis Functions). More specifically, we focus on a continuous variant of the well-known Q-learning algorithm with experience replay, combined with the NAF representation and deep neural networks. The original NAF representation is by design unimodal, given that the quadratic advantage function offers only one mode, which means that loss in performance may occur due to the inability to explore and capture complex representations with multiple modes. To tackle this problem, this thesis proposes two multi-modal representations as a simple solution. The first one uses multiple quadratic terms, whereas the second one uses RBFs. In each case, the formulation of the action advantage is accomplished by two different methods. The first one uses the sum of equally weighted advantage terms, which are derived as outputs of the neural network. The second method uses the argmax operator over the advantage terms. Both of these methods avoid any direct interaction with the neural network, thus making the proposed architectures more efficient. In order to evaluate our implementation, simulation tests were run on an open-source platform, called RoboSchool, which is integrated into the broader OpenAI Gym framework, and provides different

environments for testing reinforcement learning algorithms. In our case, we used six environments (pendulum, inverted pendulum, inverted double pendulum, humanoid, ant, walker2d), which support different simulated robots and consist of continuous control tasks. Our results showed a significant improvement in performance and efficiency of the proposed multi-modal algorithm compared to the original unimodal one, nevertheless at the cost of some increase in computation time. We observed that the outcome for each task differs as it depends on the values of several hyper-parameters, with batch normalization, learning rate and exploration noise being the most sensitive ones. This thesis is a first step towards a full-scale extension to multi-modal representations and their application to more complex environments yielding even more robust solutions to continuous control tasks.

Διπλωματική εργασία του φοιτητή

Χρήστου Π. Πετρίδη

με θέμα

Μια Πολυ-Τροπική Προσέγγιση Q-Μάθησης μέσω Κανονικοποιημένων Συναρτήσεων Κέρδους και Βαθέων Νευρωνικών Δικτύων

Περίληψη

Η Ενισχυτική Μάθηση (Reinforcement Learning), ως κλάδος της Μηχανικής Μάθησης που προσανατολίζεται στην ανάπτυξη αυτόνομων πρακτόρων, παρουσιάζει μια ταχεία εξέλιξη τα τελευταία χρόνια ως τρόπος επίλυσης προβλημάτων ακολουθιακών αποφάσεων. Η ανάπτυξη αξιόπιστων Βαθέων Νευρωνικών Δικτύων (Deep Neural Networks) έχει επίσης διαδραματίσει καθοριστικό ρόλο στην επιτυχία αυτή. Ο συνδυασμός αυτών των δύο περιοχών τελικά οδήγησε στη Βαθιά Ενισχυτική Μάθηση (Deep Reinforcement Learning), ένα πεδίο τελευταίας τεχνολογίας που κατέδειξε ήδη μεγάλες δυνατότητες και αξιοθαύμαστα αποτελέσματα σε προβλήματα συνεχούς ελέγχου. Για να συμβάλλουμε σε αυτή την προσπάθεια, η παρούσα διπλωματική εργασία διερευνά την επέκταση των Κανονικοποιημένων Συναρτήσεων Κέρδους (Normalized Advantage Functions - NAFs) σε πολυτροπικές αναπαραστάσεις, όπως πολλαπλά quadratics

και RBFs (Radial Basis Functions). Ειδικότερα, εστιάζουμε σε μια συνεχή παραλλαγή του γνωστού αλγόριθμου Q-learning με επανάληψη εμπειρίας σε συνδυασμό με την αναπαράσταση NAF και τα βαθιά νευρωνικά δίκτυα. Η αρχική αναπαράσταση NAF είναι από το σχεδιασμό της μονοτροπική, δεδομένου ότι η quadratic advantage function προσφέρει μόνο ένα mode, πράγμα που σημαίνει ότι μπορεί να προκύψει απώλεια απόδοσης εξαιτίας της αδυναμίας εξερεύνησης και αποτύπωσης σύνθετων αναπαραστάσεων με πολλαπλά modes. Για να αντιμετωπίσει αυτό το πρόβλημα, αυτή η διπλωματική εργασία προτείνει δύο πολυτροπικές αναπαραστάσεις ως απλή λύση. Η πρώτη χρησιμοποιεί πολλαπλά quadratics, ενώ η δεύτερη χρησιμοποιεί RBFs. Σε κάθε περίπτωση, η διαμόρφωση του action advantage επιτυγχάνεται με δύο διαφορετικές μεθόδους. Η πρώτη χρησιμοποιεί το άθροισμα εξίσου σταθμισμένων όρων advantage, οι οποίοι παράγονται ως έξοδοι του νευρικού δικτύου. Η δεύτερη μέθοδος χρησιμοποιεί τον τελεστή argmax πάνω στους όρους advantage. Και οι δυο μέθοδοι αποφεύγουν οποιαδήποτε άμεση αλληλεπίδραση με το νευρωνικό δίκτυο, καθιστώντας έτσι τις προτεινόμενες αρχιτεκτονικές αποτελεσματικότερες. Προκειμένου να αξιολογηθεί η υλοποίησή μας, πραγματοποιήθηκαν δοκιμές προσομοίωσης σε μια πλατφόρμα ανοιχτού κώδικα, που ονομάζεται RoboSchool, η οποία ενσωματώνεται στο ευρύτερο πλαίσιο OpenAI Gym και παρέχει διαφορετικά περιβάλλοντα για τον έλεγχο των αλγορίθμων ενισχυτικής μάθησης. Στην περίπτωσή μας χρησιμοποιήσαμε έξι περιβάλλοντα (εκκρεμές, αντεστραμμένο εκκρεμές, αντεστραμμένο διπλό εκκρεμές, ανθρωποειδές, ant, walker2d), τα οποία υποστηρίζουν διαφορετικά προσομοιωμένα ρομπότ και αποτελούνται από προβλήματα συνεχούς ελέγχου. Τα αποτελέσματά μας έδειξαν σημαντική βελτίωση στις επιδόσεις και την αποτελεσματικότητα του προτεινόμενου πο-

λυτροπικού αλγόριθμου σε σύγκριση με τον αρχικό μονοτροπικό αλγόριθμο, ωστόσο με το κόστος κάποιας αύξησης του υπολογιστικού χρόνου. Παρατηρήσαμε ότι το αποτέλεσμα για κάθε εργασία διαφέρει καθώς εξαρτάται από τις τιμές αρκετών υπερπαραμέτρων, με τις batch normalization, learning rate και exploration noise να είναι οι πιο ευαίσθητες. Η παρούσα διπλωματική εργασία είναι ένα πρώτο βήμα προς μια πλήρη επέκταση σε πολυτροπικές αναπαραστάσεις και την εφαρμογή τους σε πιο σύνθετα περιβάλλοντα αποφέροντας ακόμη πιο αξιόπιστες λύσεις σε προβλήματα συνεχούς ελέγχου.

Acknowledgements

I could not imagine that this thesis would last about a year. During that period, a lot of things happened in my life that led me to work harder than I expected. This journey taught me to fight even if the initial plan deviates. Hence, it is very important for me to dedicate some lines in this document in order to thank the people that stood by my side and I finally completed my thesis.

The first person that I would like to thank is my supervisor, Vice Rector of TU Crete and Assoc. Professor Michail G. Lagoudakis. His guidelines, his experiences that he shared with me and our scientific discussions from undergraduate courses to this work helped me to gain the appropriate knowledge to complete my Diploma Thesis.

I would also like to thank my best friends, Anastasios and Athanasios, who have supported and encouraged me since our childhood. I also thank my fellow students, Antonios, Konstantinos and Nikolaos, as we spent many hours of studying, attending classes and hanging out in beautiful places in Chania during these six years.

Especially, I would like to thank the person that was able to tolerate me during these years; my girlfriend Anastasia. I am really grateful for her patience, her constant support and her willingness to stand by me in good and bad times. Thank you for making my life easier and happier!

Lastly and most importantly, I dedicate this thesis to my family, my father Panagiotis, my mother Marianthi and my sister Charikleia, for teaching me the meaning of strength, commitment and integrity. I owe all my successes and achievements to you.

Αφιερώνεται στους γονείς μου,

Παναγιώτη και Μαριάνθη.

Contents

List of Tables

List of Figures

Acronyms and Abbreviations

2D — Two Dimensional

3D — Three Dimensional

AI — Artificial Intelligence

ANN — Artificial Neural Network

CNN — Convolutional Neural Network

DL — Deep Learning

DNN — Deep Neural Network

DRL — Deep Reinforcement Learning

MDP — Markov Decision Process

NAF — Normalized Advantage Function

NN — Neural Network

OU — Ornstein-Uhlenbeck

RBF — Radial Basis Function

RL — Reinforcement Learning

Chapter 1

Introduction

In recent years, Artificial Intelligence (AI) is a field that has captured our attention due to its revolutionizing impact in our lives. One of its goal is to create fully-automated agents that interact with the environment to learn optimal behaviours and to make the best decisions. The implementation of such systems is neither easy nor quick. Instead, it requires a lot of time and effort to build reliable systems which must be responsive and learn effectively. In this context, Reinforcement Learning (RL), as a mathematical framework for autonomous learning, has the potential for great success [3]. RL refers to algorithms whose goal is to learn how to maximize their returns through trial and error in sequential decision problems; for example, get the maximum winning points in a game over many actions. The rapid evolution of deep learning led deep reinforcement learning (DRL) to achieve great success in problems highly challenging. One of the characteristics of DRL is that the deep neural nets represent value and policy functions thus many DL methods can easily be used. For example, some algorithms such as deep Q-network (DQN) [21] and trust region policy optimization (TRPO) [42] have achieved scores that approximate human-level performances. Hence, these algorithms tend to become the core of artificial intelligence.
However, a gap is observed when applying DRL methods and their theory despite the fact that they have demonstrated tremendous success empirically.

More specifically, the majority of theoretical work related to reinforcement learning refers to state and action spaces which are finite and sometimes focuses on cases which present a linear value function. Under these circumstances, linear regression and convex optimization are methods that can easily explain reinforcement learning from statistical and algorithmic point of view. Instead, in deep neural networks which are nonlinear function approximators, it is extremely hard to analyse the theoretical perspective of reinforcement learning as it requires to solve a nonconvex statistical optimization problem.
The acquisition of new deep learning tools in combination with the need to make robust reinforcement algorithms or to evolve the old ones led the community to pay attention to the field of deep reinforcement learning. The problem of applying the Q-learning algorithm to stochastic domains, i.e. to solve continuous control tasks with high-dimensional action spaces, was an interesting challenge which led to the NAF [2] algorithm. Based on that, we took a step further and tried to extend this algorithm so as to be able to exploit its full functionality. To achieve this, we approached this algorithm from a different perspective and we focused on importing the idea of multi-modularity. To date, this approach has not been explored sufficiently thus we decided to delve more into this problem.

1.1 Thesis Contribution

This work proposes an idea on how to approach and implement multimodal representations. The basic idea was to reimplement, extend and test new multi-modal representations on the existing NAF algorithm in order to seek an optimal mode and not to collapse to the first one it finds. We need to point out that this thesis has an educational direction as all of the experiments were done under laboratory conditions and are not tested on real physical systems. The basic aim was to give the opportunity to one who is interested in this field to delve more into our proposed idea, to get in touch with an aspect of deep reinforcement learning and, why not, to come up with new, more robust

and state-of-the-art techniques.

1.2 Thesis Outline

The structure of this thesis is as follows: In Chapter 2 we will refer to the essential theoretical background regarding Reinforcement Learning and its policies, the Markov Decision Process, the architecture of the Neural Networks and their implementation and the Advantage Updating algorithm. In Chapter 3 we will discuss in detail the development in the fields of Reinforcement Learning and Deep Learning and we will provide information about the related work undergoing in continuous state and action spaces and the development of the advantage functions. In Chapter 4 we will provide a detailed analysis of how we approach our problem, how to choose our methods and what our principal idea was. In Chapter 5 we will refer to the methods we chose and we will explain in detail the process of their implementation. In Chapter 6 we will describe the environments and the software platforms that we used in order to test and evaluate our methods. In Chapter 7 we present in detail the results of our work while in Chapter 8 we will discuss the importance of our results, we will propose some ideas for future work and we will cite the conclusions of this work.

Chapter 2

Background

2.1 Learning

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) is a discrete-time mathematical modelling framework for decision making, particularly useful when the outcome of a process is in part a result of the agents actions and in part random. They have found extensive use in areas such as economics, control, manufacturing, and RL.

A MDP can be described as a 5-tuple (S, A, P, R, γ), where:

- $S = s_1, s_2, \ldots, s_n$ is the finite state space of the process. The state $\mathbf{s}$ is a description of the status of the process at a given time.
- $A = a_1, a_2, \ldots, a_m$ is the finite action space of the process. The set of actions are the possible choices an agent has at a particular time.
- P is a Markovian transition model, where $P(s, a, s')$ is the probability of making a transition to state s' when taking action a in state s. A

Markovian transition model means that the probability of making a transition to state s' when taking action a in state s depends only on s and a and not on the history of the process.

- R is the reward function (real number) of the process. It is Markovian as well and can be the instant or the expected instant reward (for stochastic rewards) at each time step. The expected reward $\mathbf{R}$ for a state-action pair (s, a), is defined as:

$$\mathbf{R}(s, a) = \sum_{s' \in S} P(s, a, s') R(s, a, s')$$

- $\gamma \in (0, 1]$ is the discount factor. When $\gamma = 1$ a reward retains its full value independently of when it is received. As γ becomes smaller, the importance of rewards in the future is diminished exponentially by γ^t.

A MDP is often augmented to include also D as (S, A, P, R, γ, D), where:

- D is the initial state distribution. It describes the probability that each state in S will be the initial state. In some problems most states have a zero probability, while few states (possibly only one) are candidates for being an initial state.

In a MDP, the optimization objective is to maximize (or minimize) the expected total discounted reward, which is defined as:

$$E_{s \sim D; a_t \sim ?; s_t \sim P} \left(\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right)$$

through appropriate action choices at each time step.

Bellman Equation

The goal of an agent is to get the highest expected reward from any state it may be located in. In order to succeed, the optimum value function must be

achieved, i.e. to maximize the sum of cumulative rewards [4].
Using the Bellman equation, the **state value function** can be split in two parts: an immediate reward and a discounted value annotated $\mathbf{R_{t+1}}$ and $\gamma\mathbf{G_{t+1}}$ respectively.

$$\begin{aligned} V(s) = E[G_t \mid S_t = s] &= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s] \\ &= E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots) \mid S_t = s] \\ &= E[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= E[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s] \end{aligned}$$

According to the above equation, the state-action value function $\mathbf{Q}$ can be defined as:

$$Q(s,a) = E[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

2.1.2 Policies

The term policy refers to the process of mapping states to actions. A policy π defines the response, which may be either deterministic or stochastic, of an agent in the environment for any state and it has the ability to completely determine the agent's behaviour. Thus, $\pi(s)$ is the action chosen by the agent, following the policy π.

In addition, there is a kind of policy that yields the highest expected utility. This policy is called optimal policy $\boldsymbol{\pi^*}$. This policy targets to maximize the expected total discounted reward over the entire state space and under all conditions. It is known that for every MDP there exists at least one optimal policy. But this does not mean that this policy is unique. Hence, equal expected total discounted reward can be produced through different actions. To sum up, the definition of state-action value function $Q_\pi(s,a)$ for a policy

π is shown below

$$Q_\pi(s,a) = E_{a_t \sim \pi; s_t \sim P}\left(\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right)$$

and indicates, over all possible combinations of states and actions, the expected, discounted total reward when action a is taken at state s by following the policy π.

2.1.3 Reinforcement Learning

Reinforcement Learning is the procedure of learning in an environment by interacting with it [5, 3, 6]. The agent usually does not know anything about neither the environment -as it does not have any model of the underlying MDP- nor what the results of its actions are. On top of that, the possibility of a stochastic environment leads the agent to yield different outcomes for the same situation. Thus, RL is considered as the third tier in the field of machine learning due to the fact that there is no teacher to give information about good or bad behaviours as it happens in supervised learning whereas the goal in unsupervised learning is to find similarities and differences between data points.

In this context, the goal of an autonomous agent is to learn how to maximize the cumulative reward over time derived from the consequences of its actions. Usually, this reward is discounted because when it is obtained in early stages, it is considered more valuable than a later one. This can be considered as a measure for the agent to find a quick solution and not to waste time in states that provide negative rewards.

In RL, there are two related problems: Prediction and Control. As far as prediction is concerned, the main goal is to find a way to predict the total reward given a fixed policy. However, in control problems, the goal of the agent is to learn how he can maximize the total reward by finding a good policy. It is often to see these two problems together, as the prediction algorithm tries to evaluate a policy whereas the control algorithm improves

it.
In figure 2.1, a typical framing of a RL scenario is depicted: an **agent** receives the **state** and the **reward** derived from the previous state transition. Based on these, the agent takes an action and interacts with the environment. The environment receives the **action**, returns the **new state** and the **new reward** to the agent and this process is executed repeatedly.

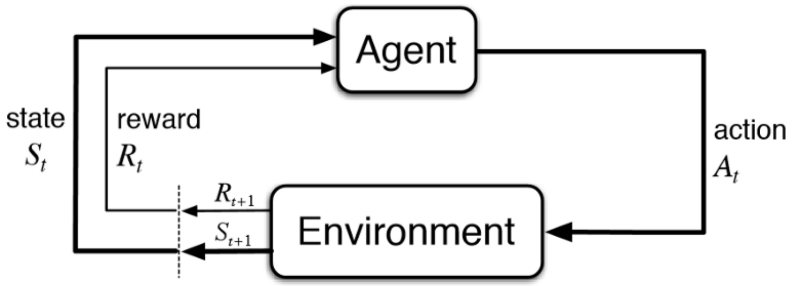


Figure 2.1: Agent and environment interaction in RL (Sutton and Barto, 1998)

Main Elements of RL

A RL agent could consist of one or more of the following elements:

- **Policy:** π indicates how an agent behaves. More specifically, it is a function that maps each state to an action, such that:

  $$\pi : S \rightarrow A$$

  where

  $$S : \text{state space}$$

  and

  $$A : \text{action space}$$

  Further, a policy function can be either Deterministic ($A_t = \pi(s_t)$) or Stochastic ($\pi(a \mid s) = P[a_t = a \mid s_t = s]$)

- **Return:** As γ is the discount factor of future rewards, the return G_t, which is also called expected total discounted reward at step t, is given by:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{m=0}^{\infty} \gamma^m r_{t+m+1}, \qquad \gamma \in (0, 1]$$

Model-free RL

Model in reinforcement learning is often refer to the transition dynamics of the environment:

$$p(s', r \mid s, a) \quad \forall s, a$$

In general, model-free means that the agent tries to maximize the expected reward only from real experience, without a model/prior experience. After taking an action, it does not know which state it will be in but it matters only for the reward associated with that state. More specifically, next states and available actions are only observed based on what the agent experiences. Model-free methods are generally less computational-heavy compared to model-based methods as they try to obtain the optimal policy for the given task and not to learn the entire dynamics of the environment.

Q-learning

Q-learning [7] is a model-free, off-policy algorithm. The use of this algorithm can be either in an online or offline setting. Its main function is based on samples of the form (s, a, r, s') so as to estimate the state-action value function of an optimal policy. Each sample is a minimal piece of agent-environment interaction; s is a state experienced by the agent at some time step, a is the action chosen and executed by the agent in that state, r is the reward that was received during the transition, and s' is the next state observed at the next time step. As it is showed below, the difference update

equation is:

$$Q(a,s) = Q(a,s) + \alpha[R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)]$$

where α: is the learning rate which determines to what extent newly acquired information overrides old information.
The algorithm's advantage is the limited computational demands at every step, whereas its drawback is the large amount of required steps so as to converge. However, under uncertain conditions, it allows an agent to act optimally by using an action-value representation that it has learned, satisfying the Markov property. Figure 2.2 depicts the Q-learning algorithm.

Q Learning.
Input: $D, \gamma, Q_0, \alpha_0, \sigma, \pi$. Learns $\hat{Q}^*$ from samples
// D: Source of samples (s, a, r, s')
// γ: Discount factor
// Q_0: Initial value function
// α_0: Initial learning rate
// σ: Learning rate schedule
// π: Exploration policy
$\tilde{Q} \leftarrow Q_0; \alpha \leftarrow \alpha_0; t \leftarrow 0$
for each (s, a, r, s') in D **do**
$\tilde{Q}(s,a) \leftarrow \tilde{Q}(s,a) + \alpha\left(r + \gamma \max_{a' \in A} \tilde{Q}(s',a') - \tilde{Q}(s,a)\right)$
$\alpha \leftarrow \sigma(\alpha, \alpha_0, t)$
$t \leftarrow t + 1$
end for
return $\tilde{Q}$

Figure 2.2: Q-learning algorithm [7]

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) or Neural Networks (NN) are computing systems vaguely inspired by the biological neural networks that constitute the brain. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signals additional artificial neurons connected to it.

2.2.1 Network Architecture

A very simple NN could be considered as a linear transformation of some data points X:

$$X_1 = WX + B$$

When the values of some column in X affect all the values in the same column in X_1 then the *input layer* X and the *layer* X_1 are said to be *fully connected*. As it is described above, the convention of characterizing NN with layers has its origins in physical neurons as they are arranged in layers. The aforementioned transformation can be implemented only in linear transformations. Thus, learning non-linear functions requires the addition of a non-linear *activation* function:

$$X_1 = f(WX + B)$$

When learning is about more complex functions, transformations can be recursively stacked:

$$X_2 = f(W_1 X_1 + B_1)$$
$$\dots$$
$$X_{k+1} = f(W_k X_k + B_k)$$

In the context of an optimization algorithm, the function used to evaluate a candidate solution is referred to as the objective function. Typically, with neural networks, we seek to minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply the "loss".

More specifically, a loss function is a function that maps a set of parameter values for the network onto a scalar value that indicates how well those parameter accomplish the task the network is intended to do. The data are often the input in the NN with the corresponding target values. This characterizes the error and leads to the desired behaviour of the NN. It is easy to find the loss function as all parts are differentiable and thus, its derivatives can be calculated with respect to all the parameters $W, W_1, \dots, W_k$ and $B, B_1, \dots, B_k$. Knowing about that, we can minimize the loss function using the gradient descent by an effective way called *back-propagation* [8]. As far as the structure of the NN is concerned, the first layer which is the X is commonly referred as *input layer*, the last layer as the *output layer* and all the other intermediate layers are called *hidden layers*. In Figure 2.3, the architecture of an ANN is depicted.

2.2.2 Common Activation Functions

In a NN, the purpose of an activation function is to redefine one node's output and to map output values in certain intervals which depends on what activation function is used.

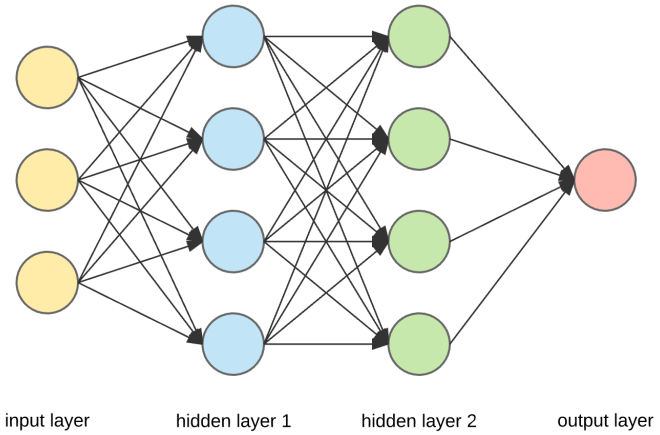


Figure 2.3: Artificial Neural Network

Linear function

The function is a line or linear as shown in Figure 2.4. Therefore, the output of the functions will not be confined between any range. It does not help with the complexity or various parameters of usual data that is fed to the neural networks whereas its range is between $(-\infty, \infty)$. The equation is:

$$\phi(z) = cz, \quad \text{c : constant}$$

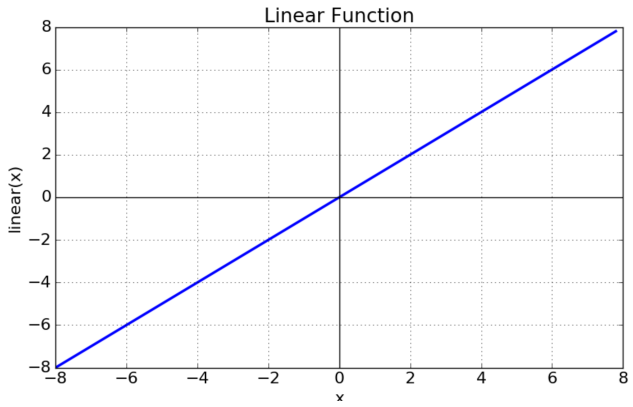


Figure 2.4: Linear Function

Sigmoid function

The sigmoid function curve looks like a S-shape, as shown in Figure 2.5. The main reason why we use sigmoid function is because it produces values in the interval $[0, 1]$. Therefore, it is especially used for models which require the prediction of the probability as an output. The drawback of this function is that it can cause a NN to get stuck during training time.

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

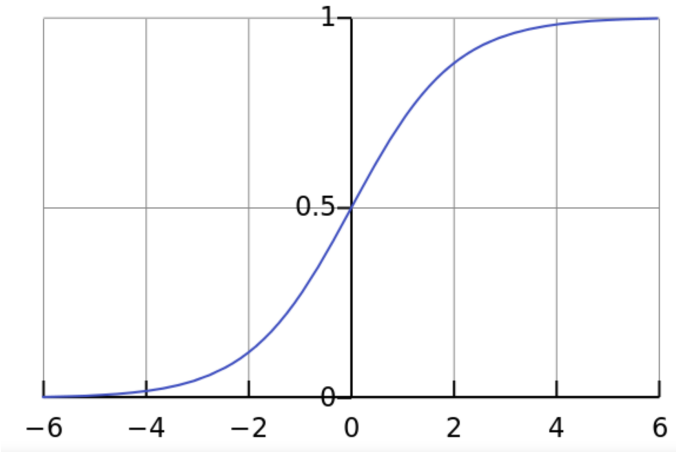


Figure 2.5: Sigmoid(or Logistic) Function

Tan-h function

The tanh (or hyperbolic tangent) function is also like sigmoid but it acts better. The range of the tanh function is between $(-1, 1)$. As Figure 2.6 shows, tanh is also sigmoidal (s-shaped). The advantage of this function is that the negative inputs will be mapped truly negative and the zero inputs

will be mapped near zero.

$$\phi(x) = \frac{2}{1 + e^{-2x}} - 1$$

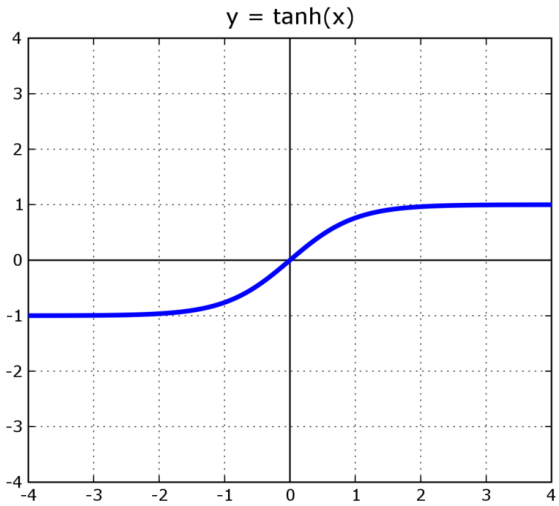


Figure 2.6: Tan-h Function

Softmax function

For classification networks, a very common function that is used in the output layer is the softmax function as shown in Figure 2.7. A basic property of the softmax is that all output values will be mapped in the range $[0, 1]$ and sum up to one, making it an ideal choice for different classes or categories to be interpreted as probabilities. The mathematical definition for a set $x_1, \ldots, x_K$ is shown below:

$$\text{softmax}(x_k \mid x_1, \ldots, x_K) = \frac{e^{x_k}}{\sum_{i=1}^{K} e^{x_i}}$$

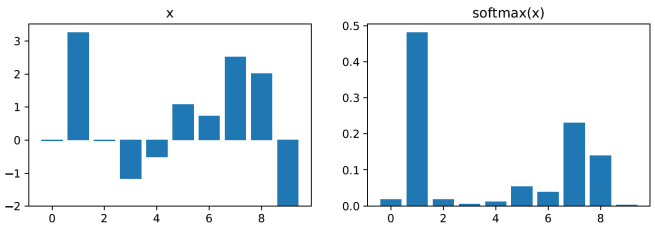


Figure 2.7: Softmax Function

ReLU(Rectified Linear Units) function

To date, the ReLU function, shown in Figure 2.8 is the most common activation function since it is used in almost all the CNNs or DL. When the input is below 0, the linear unit with the rectified output is 0 whereas if the input is greater than 0, the result equals the input. However, the disadvantage is that any negative input given to the ReLU activation function outputs into zero, which means that the resulting graph is affected because the negative values are not mapped appropriately. The mathematical expression is:

$$\phi(x) = \max(0, x)$$

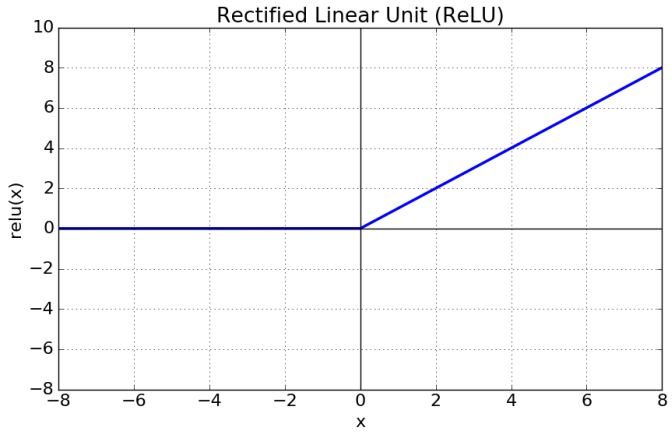


Figure 2.8: ReLU Function

Leaky ReLU

The Leaky ReLU activation function, shown in Figure 2.9, is very similar to the ReLU but it has one difference; the negative from the inputs are replaced by a multiple of the input with a small constant factor α. In this way, this activation function can solve the "dying ReLU" problem as it is called, setting the range of Leaky ReLU in $(-\infty, \infty)$. The mathematical expression is shown below:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

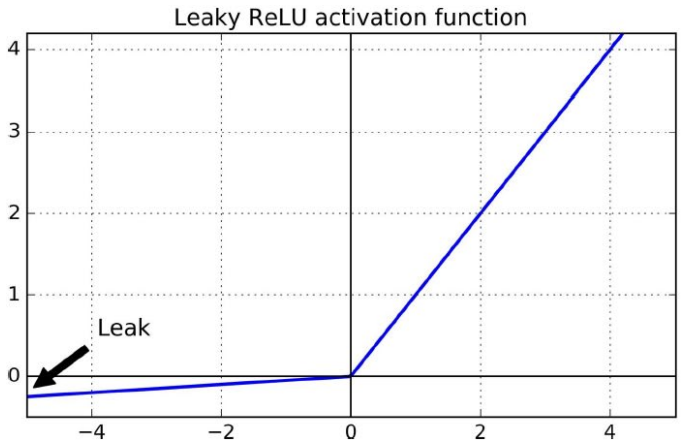


Figure 2.9: Leaky ReLU Function

In general, a neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. For instance, if there is not a non-linear activation function, the network will behave as having a single layer, no matter how many layers it actually has.

In addition to the aforementioned, some activation functions have the ability to tackle the problem of the vanishing gradient. In other words, this problem occurs when the gradient has so small values in prior layers of a deep neural

network that finally does not have significant improvement in the weight values of the earlier layers.

2.2.3 Optimizers

A common method used for optimizing neural networks is called Adam [9]. The estimation of first and second order of the gradients $g_t = \nabla_{\theta_t} f(\theta_{t-1})$ gives the ability of updating the parameters of the loss function $f(\theta)$. The current timestep is annotated with the letter t and its initial value is equal to one. Its use helps to bias-correct the estimation of first and second order:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) g_t^2$$

$$\mathbb{E}[g] \approx \hat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\mathbb{E}[g^2] \approx \hat{u_t} = \frac{u_t}{1 - \beta_2^t}$$

Here, $\beta_1, \beta_2 \in [0, 1)$ are hyper parameters. The updates are based on a step size α according to:

$$\theta_t = \theta_{t-1} - \frac{\hat{m_t}}{\sqrt{\hat{u_t} - \epsilon}}, \quad \epsilon > 0$$

Adam could be considered as a generalization of RMSProp, setting $\beta_1 = 0$ [10]. This means that the estimation of the second order is used only. Such an optimizer has successful implementation on training on-policy algorithms [11]. Some other commonly optimizers used earlier include Adadelta [12], Adagrad [13] and momentum [14].

2.2.4 Batch Normalization

It is known that the weights of a layer take different values during the training time, thus next layers have to adjust to the changes. To be more specific, any layer that comes next, needs to incorporate these changes, a process known as *internal covariance shift*. A simple solution to this problem is the use of normalization layers, called *batch normalization* [15]. The purpose of these layers is to bleach the activation functions of the previous layer. This process is done by element-wise subtraction of the mini-batch mean and divide by the square root of the variance. For every mini-batch the statistics are computed, which is considered as a regularizer and in some cases, there is no need for dropout. According to the authors, this happens because the representation of one sample will be shifted independently relative to the other samples of the mini-batch. Sometimes, a problem can occur when bleaching the outputs of a previous layer because it may lead to a decrease on what the next layer can represent. For this problem to be overcome, the authors proposed a process where parameters will be trained so as to ensure that the normalization layer, in case it is needed, will produce the identity function. During inference, the process of normalization is done on population estimates of the mean and variance. These population estimates are inferred using running mean and variance estimates attained during training. The use of batch normalization, in many cases, improved the test errors in state-of-the-art neural networks.

2.3 The Advantage Updating Algorithm

The use of *advantage updating* is a great tool for reinforcement learning in continuous time. The advantage updating algorithm is placed among other reinforcement learning algorithms and it stores two types of information. Firstly, the value $V(s)$ is stored for every state s which represents the total discounted expected return, starting in state s and taking optimal actions. Secondly, the *advantage* $A(s, \alpha)$ is stored for every state s and action α.

This value shows to what extent the expected total discounted reinforcement is increased when an action α has been taken based on the current action which is considered to be the best so far. After the system converges to the optimality, the true value of each state is considered the value function $V^*(s)$. As far as the advantage $A^*(s,a)$ is concerned, it is zero when the action a is the optimal due to the fact that a presents no advantage related to itself whereas it takes negative values when any suboptimal action is chosen which makes sense because a suboptimal action could not have a positive or zero advantage relative to the best action. Given an action a, the $Q^*(s,a)$ performs as the utility of the action, the change in $\Delta V^*(s,a)$ performs as the incremental utility whereas the advantage $A^*(s,a)$ performs the utility of the action based on the optimal action. Below, the advantage function A^* in terms of the V^* is defined:

$$A^*(s,a) = \frac{1}{\Delta t}\left[R_{\Delta t}(s,a) + \gamma^{\Delta t}\sum_{s'} P(s,a,s')V^*(s')\right] - V^*(s)$$

The correlation between the advantages and the Q values can be defined as:

$$A^*(s,a) = \frac{1}{\Delta t}\left[Q^*(s,a) - \max_{a'} Q^*(s,a')\right]$$

During the learning, the value function and the advantage function are needed. Instead, after convergence to the optimality, only the advantage function is capable of extracting the policy. This policy is optimal in state s for any action α that maximizes $A^*(s,a)$ and defined as:

$$A_{\max}(s) = \max_a A(s,a)$$

In case that the above quantity is zero in every state, then the advantage function is considered *normalized*.

Chapter 3

Related Work

In this section, we will talk about related work in the fields of reinforcement learning and deep reinforcement learning. Then, an overview of recent research in the domain of continuous space and actions will be presented and be discussed.

3.1 RL and DL

Many challenging sequential decision-making problems had not find solutions for many years. However, in recent years, RL managed to become increasingly popular due to the fact that it has succeeded to tackle a great amount of these problems. To achieve this, DL techniques had to be combined with RL [16, 17, 18]. The combination of these two fields has generated a new field called Deep Reinforcement Learning (DRL) and its use is most successful in problems which have high dimensional state-space. Some other RL approaches in the past faced designing difficulties on how to choose features [19, 20]. Instead, DRL has the ability to succeed in extremely complicated tasks with no or lower prior knowledge and this happens due to the fact that it can learn different levels of abstractions from raw data. For instance, a DRL agent can learn policies from raw inputs of thousands pixels so as to

play and compete humans in Atari games [21, 22]. In addition to this, related methods have been applied in order to perform robotic control tasks both simulated and real-world [23, 24, 25]. These breakthroughs can offer the possibility of imitating, even in high dimensional space, the ability of human problem solving, which was difficult to think of a few years ago.
Deep RL has done several notable works in gaming, such as reaching superhuman level in Atari games [21], mastering in Go [26] or winning top professionals in Poker [27, 28]. In real-world applications, DRL has a great potential as well, such as self-driving cars [29], robotics [30, 31, 32], smart grids [33] and finance [34]. Notwithstanding all the innovations so far, more challenges come out in order to apply DRL algorithms. For instance, it is not straightforward to explore the environment effectively or to find a good behaviour and generalize it in another similar context. Hence, many algorithms have been recommended for the DRL framework, depending on a wide range of settings of sequential decision-making problems.
The aforementioned success stories in the field of gaming are very important because it is showed that the DRL has a great potential in a wide range of complex and diverse tasks, especially when working with high dimensional inputs are required. However, deep RL systems have already begun to be used in real environments. For instance, [35] refers to how Facebook exploits DRL for pushing notifications or how it loads videos in a faster way using smart pre-fetching.
While there are fields that someone could say that supervised learning can stand alone, i.e sequence prediction [36, 37], it can be proved that RL can also be applicable to those ones. In [38] is showed that the design of a right neural architecture for applications which use supervised learning can constitute a RL problem. But, it is known that these tasks are able to be managed by evolutionary strategies as well [39, 40].
As we can see, DRL has a lot of applications in many different fields and thus, the field of computer science could not be an exception. In other words, classic and fundamental algorithmic problems have the potential to be solved by DRL, e.g the travelling salesman problem [41]. As this problem is NP-complete, the chance of tackling it using DRL leads to the conclusion that

many other NP-complete problems could be solved if their structure allows it.

3.2 Continuous State and Action Domains

In domains with discrete actions, i.e. Atari games, most of RL techniques focus on value estimation and Q-learning [21]. Instead, this differs in continuous domains because explicit representation of a policy is required. For instance, in a policy gradient algorithm. In general, the value function estimate is crucial and has a lot of benefits. However, these benefits cost in the architecture of a system, especially when we talk about continuous deep RL. Thus, the exploitation of such advantages requires the use of two networks. This happens due to the fact that both the policy and the value function must be represented by separate networks [42, 43]. As it may look like a limitation, an approach described in [2], shows how some features of Q-learning can be used into continuous domains and lead to the implementation of a single network whose outputs are the policy and the value function. A similar Q-function representation were in dueling networks [1] as shown in Figure 3.1, but it is not related to continuous action domains. Also, this alternative Q-learning technique has a correlation with the work presented in [44] but it is observed a difference in the Q-function update.

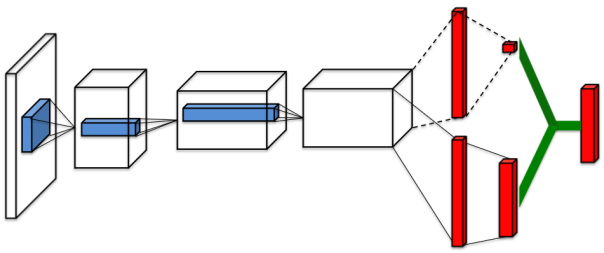


Figure 3.1: The dueling Q-network(source: [1])

Bearing in mind that we investigate and expand the work of [2], our empirical

evaluation shows that our approach boosts even more the continuous Q-learning algorithm and gives, in some benchmarks, better results than the continuous actor-critic methods and the initial unimodal approach.

3.3 Advantage Functions

The separation of value and advantage functions is not a new entry in the literature. The first reference was done by Baird [45]. In his work, called advantage updating algorithm, the equation about the Bellman residual update was split in two update terms. The one term was referred to a state value function and the other one for its associated advantage function. The results of advantage updating demonstrated a faster convergence than Q-learning in continuous domains [46]. However, the advantage learning algorithm that came after showed only a single advantage function [47]. A new architecture developed in [1]. In this work, the NN consists of a single deep model. The output of this model is a combination both of the value and of the advantage $A(s, a)$ yielding a state-action value $Q(s, a)$. This innovation makes the dueling architecture flexible to be combined with many model-free reinforcement learning algorithms. However, [48] is referred as the starting point of the use of advantage functions in policy gradients. A more recent work in this field is the estimation of advantage values on-line as a way to limit the variance of policy gradient algorithms [42].

Chapter 4

Our Approach

In the aforementioned Chapters, an extended presentation of DRL capabilities was described. The rapid development of Deep Neural Networks (DNN) and the need of more robust RL techniques have led to a great interest of further research on this field. We saw that DRL has made an impressive progress in discrete domains, such as Atari Games [49], reaching human level scores and under certain circumstances, DRL techniques outperform top professionals. This success has made the community to explore new methods in a more complicated domain, called continuous domain. The research in this domain focuses on algorithms that solve continuous state spaces. However, real-world tasks consist both of continuous state spaces and continuous action spaces, making the procedure of developing innovative and state-of-the-art algorithms even harder and more complicated. In this context, we decided to investigate and compare a method that tries to solve problems in continuous state and action spaces. Thus, we selected the Normalized Advantage Function (NAF) algorithm as our initial method in order to expand and improve it. To be more specific, the idea of NAF algorithm was born as an efficient way to enable Q-learning for being applied in continuous action spaces combined with deep neural networks (DNNs). As it is known, Q-learning [50] is a tabular method applied to problems in which the state and action space is small enough for approximate value functions to be represented as arrays and

tables. Hence, Q-learning becomes ideal for discrete domains but it cannot solve continuous state spaces.

The first approach to overcome this obstacle subjects to an improved version of Q-learning, called Deep Q-learning [49]. In this work, experience replay [51] and target networks were used and this led domains with continuous state spaces to be solved. However, it has not managed to solve domains which consist of continuous action spaces as well. A new approach that tried to solve this issue was the Dueling Networks [1], but again, its implementation could not be put into practice for continuous domains. Although the last approach was very promising yet not yielding the willing outcome, NAF algorithm [2], which our investigation is based on, has a lot of similarities with dueling networks but NAF algorithm can be applied to continuous action spaces. Understanding that this problem is offered for more investigation, we decided to conduct a further research regarding a limitation that NAF has.

Our goal was to explore the multimodal capabilities of this algorithm for educational purposes. As it will be explained further below, NAF algorithm is based on a unimodal representation. Practically, this means that the quadratic advantage function leads to a single good mode and has no ability to represent multiple good modes, which may exist. In other words, as the quadratic form is not very expressive, the algorithm can find a local optimum but it may not be the global one. From this perspective, our investigation focuses on exploring some other ways for NAF to exploit its full functionality.

Reviewing the literature, we concluded that there are no references about multimodal approaches regarding NAF algorithm and that makes our effort even more difficult yet interesting. It is worth mentioning that literature includes papers about multimodal approaches proposing energy-based policies [52] but it is not related to our work as they explore multimodularity using different methods. As it is shown, our approach to investigate the multimodal representations is quite innovative and interesting. The biggest problem that we faced was the limitations of computational resources which will be explained further in Section 6.5.

The basic idea of our approach is about finding the best parametric forms that can describe the multimodal representations. In this context, we implemented two different expressions. The first one was to use multiple quadratics which would be selected either by an equal contribution of weights or by using the argmax. In order to test more methods, we decided to also implement multimodularity using Radial Basis Functions (RBFs). In this case, the advantage term would be selected by the same method mentioned above. In Chapter 5, we describe the steps of our implementation in detail while in Chapter 7 we present our results.

Chapter 5

Implementation

5.1 NAF Description

The basic algorithm which our research was focused on is the extension of Q-learning. However, when we deal with continuous state spaces, this algorithm does not have a solution using tabular methods, and practically, even some discrete state space are too large to be solved by this method. So, there was the need of an extension of Q-learning to continuous state and action spaces. In this context, [2] proposed the normalized advantage functions, a simple solution to tackle continuous task. They proved through extensive experiments that their method is much more effective than other actor-critic methods and at the same time, it learns faster and delivers more accurate policies. However, this approach even if it is simple yielding very good results, lacks of multimodularity, a case that we investigate in this thesis. Before starting exploring how to implement our proposed extension, we firstly need to analyse the NAF algorithm and experiment with it.

5.2 NAF Architecture

As we mentioned in Section 5.1, the Q-learning algorithm could not be used directly in continuous action spaces and in order this problem to be overcome an alternative method was proposed, the Normalized Advantage Functions. To be more specific, this method combines the idea of Q-learning with deep neural networks as a means of representations. To achieve this, the Q-function $Q(s, \alpha)$ had to be in such a form that during the Q-learning update, the maximum of Q-function could be ascertained analytically and easily. As it is known, many representations can be used for the maximum to be found. However, this method uses a deep neural network as a representation tool. The NN generates two separate terms; the first one is the value function term $V(s)$ while the second one is the advantage term $A(s, a)$ and the sum of this functions leads to Q-function. The advantage term A has a specific form because it consists of non-linear (quadratic) features of the states. Thus, the advantage term can be described as a quadratic function. Now, we can define the mathematical expression both of Q-function and the advantage term $A(s, \alpha)$. So, the Q-function is defined as:

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

where

$\boldsymbol{s}$: a state of the environment

$\boldsymbol{a}$: the action to be taken

As far as the quadratic function of A is concerned, it is defined as:

$$\boldsymbol{A(s,a) = -\frac{1}{2}(a - \mu(s))^T P(s)(a - \mu(s)}$$

Looking at the aforementioned equation, we need to explain some terms in detail. Firstly, the matrix $\boldsymbol{P}$ is a matrix that depends on the state s and it is a positive-definite square matrix. Because of its form, the advantage term A

is maximized when $\alpha = \mu(s)$. The $\boldsymbol{\mu}$ plays one of the most important roles as it represents a greedy policy function. So, the Q takes its maximum value when α is the greedy action. By doing this, when we have an optimal Q, it is not required to know the optimal action, since we have the optimal value of μ.
Another term which needs to be defined from the mathematical expression of the advantage term A is the matrix $\boldsymbol{P}$. Hence, it can be expressed as:

$$\boldsymbol{P}(s) = \boldsymbol{L}(s)\boldsymbol{L}(s)^T$$

Here, $\boldsymbol{L}$ is also a matrix and specifically, it is a lower-triangular matrix. The entries of this matrix come as the output of a linear neural network, where the diagonal values are both exponentiated and strictly positive.

$$L(s) = \begin{bmatrix} e^{\sigma_{11}(s)} & \dots & 0 \\ \vdots & \ddots & \vdots \\ \sigma_{m1} & \dots & e^{\sigma_{mn}(s)} \end{bmatrix}$$

Having presented the definition of the main terms, we did not mention how to estimate the functions $\boldsymbol{V}$ and $\boldsymbol{\mu}$. Hence, these functions are produced by a deep neural network. Below, the Figure 5.1 shows the architecture of this network.

As we can see, the input of the NN is only the state $\boldsymbol{s}$. The NN consists of two hidden layers both of them with 200 fully connected units using rectified linear units (ReLU) as activation function. The outputs of the NN is the matrix $\boldsymbol{L}$ which has already been defined, the mean $\boldsymbol{\mu}$ which is produced by a fully connected layer using a tanh activation so as actions are bounded for safety reasons. Finally, the term $\boldsymbol{V}$ is also an output of the NN, produced by a fully connected layer without a specific activation function. Figure 5.2 shows how the Q-function is derived from the output of the NN.
The NAF algorithm is shown in Figure 5.3. It is worth mentioning that the algorithm uses a replay buffer in order to collect experiences and use

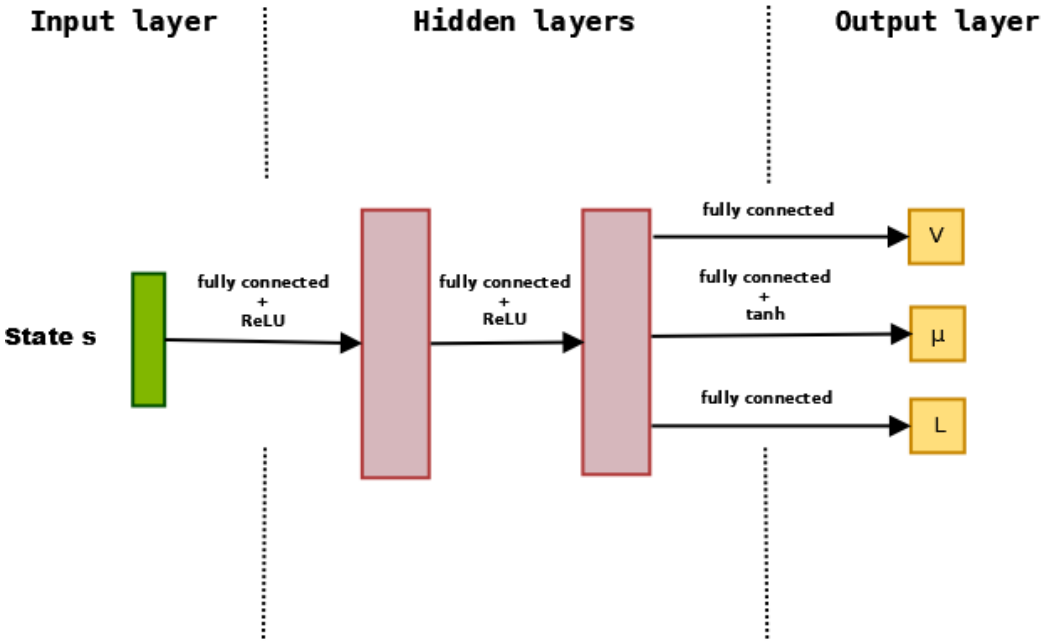


Figure 5.1: The Neural Network design for NAF. This Figure depicts only the basic components of the NN.

them a means to learn faster and more accurately. In this way, the algorithm optimizes its learning process. Moreover, noise is added to the policy μ for exploration purposes. The use of the replay buffer and the exploration noise are crucial parts of the algorithm's functionality thus they are analysed in detail.

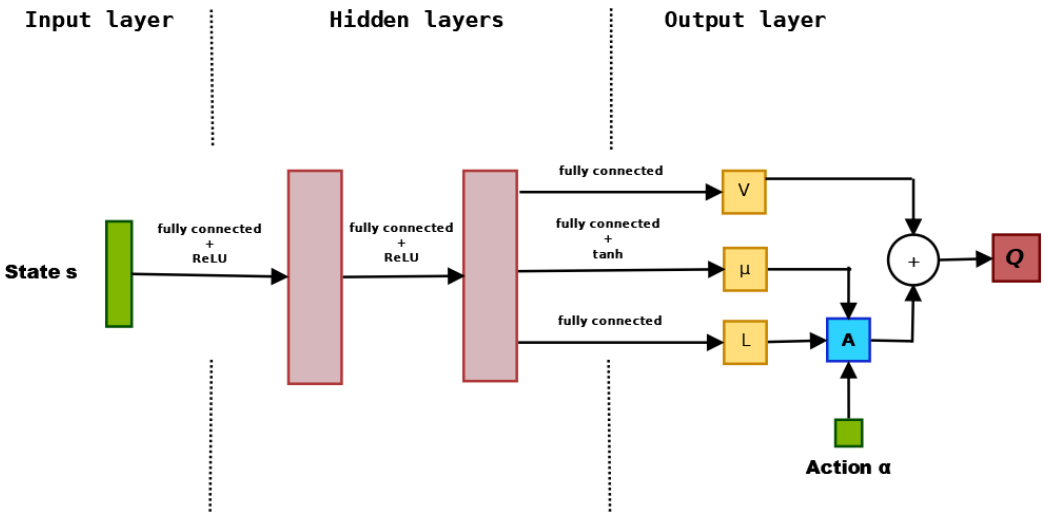


Figure 5.2: The NN and its full functionality. The Figure depicts how the outputs V, L and μ are used to form the Q-function.

Algorithm 1 NAF algorithm

Randomly initialize network $Q(x, u|\theta)$
Initialize target network Q', $\theta' \leftarrow \theta$
Initialize replay buffer $R \leftarrow \emptyset$
for episode $= 1$ to M **do**
  Initialize random process $\mathcal{N}$ for action exploration
  Receive initial exploration state x_1
  for $t = 1$ to T **do**
    Select action $\mathbf{u_t} = \mu(\mathbf{x_t}|\theta) + \mathcal{N}_\mathbf{t}$
    Execute $\mathbf{u_t}$ and observe r_t and $\mathbf{x}_{t+1}$
    Store transition $(\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1})$ in R
    for iteration $= 1$ to I **do**
      Sample a random minibatch of m transitions from R
      Set $y_i = r_i + \gamma V'(\mathbf{x}_{i+1}|\theta')$
      Update θ by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(\mathbf{x}_i, \mathbf{u}_i|\theta))^2$
      Update target network $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
    end for
  end for
end for

Figure 5.3: NAF algorithm as it described in [2]

5.3 Noise Exploration

In reinforcement learning algorithms, exploration plays a crucial role. It is known that a simple way of exploration and, most of the times, the most common is to randomize the actions based on a distribution. This procedure is done by choosing some random actions or when Gaussian noise is added in continuous state and action spaces. However, when we talk about randomness, an obstacle can show up: how to choose the magnitude of the randomness of noise for the exploration. This problem becomes even bigger in domains with high dimensionality where different actions require different values of exploration. Moreover, in some domains, independent Gaussian noise cannot have a positive impact as a means of exploration. For instance, there is a simulated task called swimming snake (see Figure 5.4) where the motion of different body joints must be coordinated for synchronization to be achieved. By adding Gaussian noise, correlation between action dimensions cannot be achieved. In order to overcome this problem and to implement the most appropriate exploration noise they decided to use the **Ornstein-Uhlenbeck** (OU) process [53] . In this way, we were able to produce a temporally correlated noise sequence [43]. An OU process requires three parameters to defined: $\boldsymbol{\sigma}, \boldsymbol{\theta}$ and $\boldsymbol{\mu}$. According to what tasks we have to deal with, we choose respectively the appopriate values for these parameters. As we saw that our algorithm was not affected by these values, we selected fixed parameter values so as to achieve the biggest fairness to our experiments. In this context, the values set to:

$$\sigma = 0.3$$
$$\theta = 0.15$$
$$\mu = 0$$

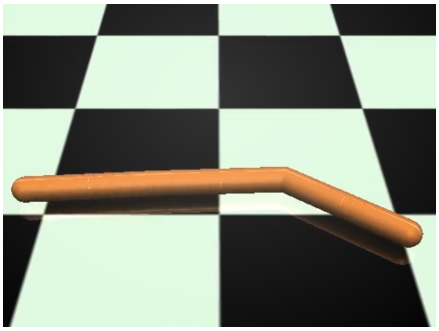


Figure 5.4: An instance of Swimmer task (source: OpenAI Gym)

5.4 Replay Buffer

In reinforcement learning, it is known that we receive sequential samples from interactions with the environment. This means that a network "sees" too many samples of one kind and forgets the others. For instance, an agent can reach the second level of a game. It looks or behaves completely differently, so it forgets how to play the first level. To avoid this inconvenience, we store all experience in a replay buffer. This allows us to train on more independent samples. We choose a batch of transitions from the buffer at random and train on that. This, eventually, helps break the temporal correlation of training samples.

5.5 The Idea of Multimodularity

Reinforcement learning is always an interesting field of research, let alone in combination with deep learning. Many state-of-the-art algorithms have been produced and one of them is the NAF algorithm. As we mentioned in Section 5.1, NAF is a new method of applying the Q-learning algorithm in domains with continuous state and actions spaces. This is a big step in research, particularly the combination of Q-learning with deep neural networks. In the beginning, we focused on investigating different covariance matrices for the advantage function term which was analysed extensively. Some of them

were the covariance matrix used in the original form, an identity covariance and a diagonal covariance matrix. The original form of the algorithm gives a lot of freedom for further research and experimentation thus we started testing different matrices. However, before exploring new forms of matrices, we reimplement the basic algorithm as close as possible to the original one and we observed that it was able to solve easily simple environments such as Pendulum, InvertedPendulum and InvertedDoublePendulum. Instead, we could not solve environments such as Humanoid or Walker2d due to the limitations of computational resources.
While we were testing different forms of covariance matrices, we observed that in most cases only the original covariance matrix had the best results and worth to be used in the majority of tasks. However, during this procedure, a new idea came out from the tests and analysis of the baseline algorithm. We observed that the quadratic form of the advantage term has the properties of a unimodal distribution. In other words, this means that the algorithm can find a local maximum which it may be a sub-optimal and thus it will not seek the optimal one. This observation led us to expand our research and to try to find a way on how the algorithm can seek more modes until to find the optimal. Hence, we concluded that we had to find a multimodal distribution and implement an extension of the basic algorithm. In this context, we started to pursue ways on how to modify the advantage term so as to be transformed into a multimodal expression.

5.6 Methods and Selection

To date, in literature, multimodularity is referred as the combination of different modalities for new features to be learnt. In this way, deep networks will be trained easily and will be more efficient solving more complex tasks. For instance, learning features for one modality, such as video, can have better results, if during the feature learning time, multiple modalities, such as audio or text, are present. Although the majority of multimodularity refers to this approach, in our case, we are looking for something completely differ-

ent. Conducting research on how to alter our advantage term so as to behave in a multimodal way, we ended up to test two methods. The first one was to investigate the use of multiple quadratics. The second one was to implement a different approach using the Radial Basis Functions (RBFs). Since we analysed if our theory could be implemented, we concluded that either of these methods could offer us the multimodal representations that we are looking for. Each method is described in detail in Sections 5.6.1 and 5.6.2. Figure 5.5 depicts a simple chart of the multimodal methods we tested.

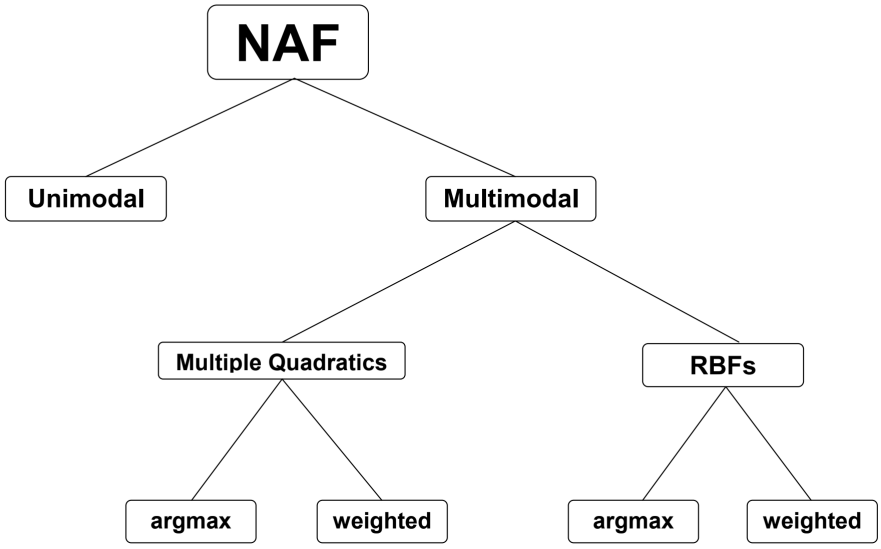


Figure 5.5: A general chart that depicts what path we followed for implementing our research

5.6.1 Multiple Quadratics

As we explained in Section 5.2, the advantage term is in a quadratic form. Also, it is a unimodal distribution which means that this expression of the advantage term could not seek any other modes since the quadratic function has only one, specifically at μ. In other words, the unimodal distribution finds a sub-optimal mode in most of the times. To change this and to give

to our algorithm the ability to seek the best mode, we propose a multimodal approach using the sum of multiple quadratics. However, the weighted sum of quadratics is not the only solution. We also propose the use of argmax which is a simple method both to be implemented and to yield good results. Let's see the analysis of our proposition.

Weights

It becomes clear from the definition that multiple quadratics require to export not only a unique advantage term but more than two. So, the new advantage term will have this form:

$$A(s, \alpha) = -\frac{1}{2} \sum_i w_i((\alpha - \mu_i(s))^T P(s)(\alpha - \mu_i(s)))$$
$$A(\mu, s) = \frac{1}{\sum w_i} \sum_i w_i A(\mu_i, s)$$

where i is the number of unimodal advantage terms that we use.
However, observing the aforementioned expression, we concluded that we needed a way on how to weigh the contribution of each term. Our first thought was the weights to be one of the outputs of our deep neural network. This option would give us the ability to produce weights with different contribution for each advantage term, giving priority and selecting the term with the biggest positive impact. At a first glance, that sounded ideal for implementation but when we tested it two problems arose. The first one was there was not a standard metric on how to generate the weights, meaning that the contribution of the terms would not be accurate and sometimes we would not choose the best one. The second and bigger problem that we faced was the stability of our neural network. The addition of another parameter which means more parameters to be trained and the need for more outputs led our network to instability and malfunction. If we consider that all of our environments consist of continuous action and state spaces with high dimensionality, it becomes clear that any deviation from the initial architecture

could lead the network to collapse. In our tests, most of times the network was unstable and could not perform in a proper way. These limitations forced us to abandon this idea and to focus on something more realistic and feasible under these circumstances. Thus, we empirically decided each term to have an equal contribution to the final advantage term. In this way, we avoid any interaction with the neural network regarding the selection of the weights and keeping our network stable. Also, in order not to output many advantage terms from our network, we set a fixed number to use, namely 4, each with 25% contribution. Hence, the final advantage term is defined as:

$$\boldsymbol{A = A_1(\mu_1, \sigma) + A_2(\mu_2, \sigma) + A_3(\mu_3, \sigma) + A_4(\mu_4, \sigma)}$$

According to aforementioned definition, it becomes clear that the NN had to change slightly. The change is about the need for three extra outputs as the advantage terms became four. Although we made these changes, we ensured the stability of our NN and its proper functionality. Figure 5.6 shows the new version of the NN.

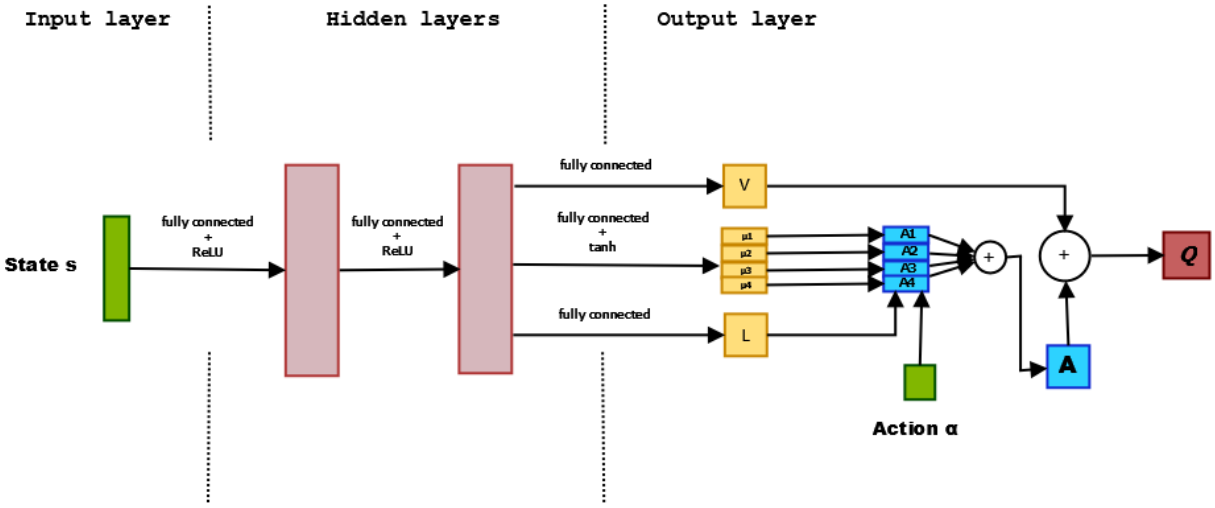


Figure 5.6: The new architecture of NN using the technique of weights.

Argmax

Our research showed that the method described in Section 5.6.1 is an effective method with a positive outcome. But, in order to find a more simple yet efficient and unbiased method we used the argmax as a way to choose the best advantage term directly. The argmax, as it is known, refers to the inputs in our case at which each advantage term output is as large as possible. Taking this into consideration, we concluded that even we had many advantage terms as output from the NN, only one of them would be the most appropriate. This characteristic feature of argmax is very important because we can avoid the use of weights and how to choose them, making the selection of the best mode even more unbiased. The mathematical expression of the advantage term is defined as:

$$\boldsymbol{A} = \text{argmax}(\boldsymbol{A_1}, \boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4})$$

In Figure 5.7 is shown the functionality of our NN using the argmax process so as the advantage term to be selected.

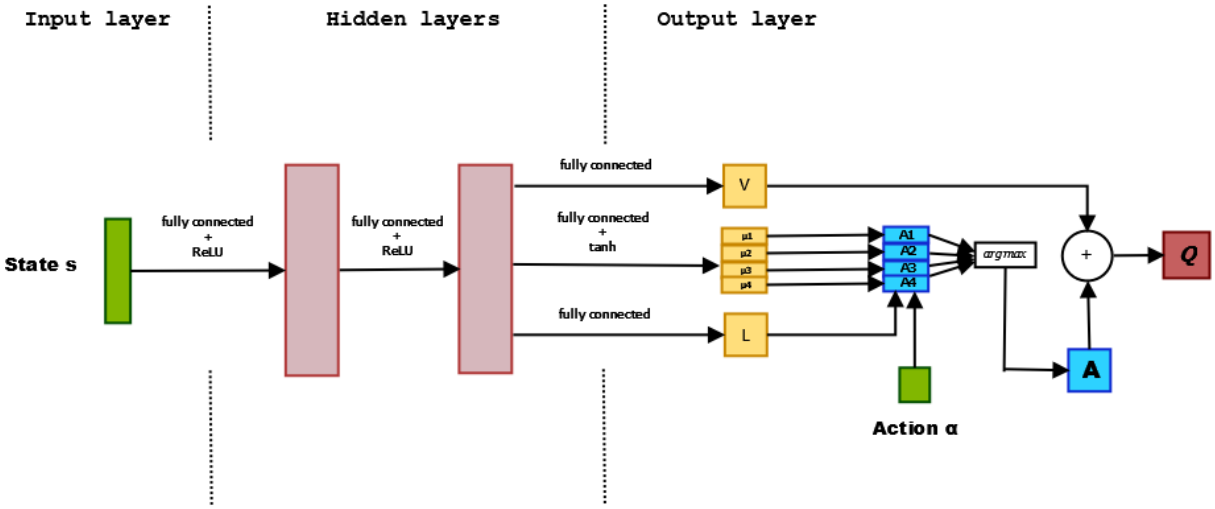


Figure 5.7: The new architecture of NN using the technique of argmax.

5.6.2 Radial Basis Functions

At a first glance, multiple quadratics seem to be a decent and quite simple approach on how to create a multimodal representation and adjust it to our case. However, we would like to make a step further and to investigate another multimodal approach, hoping to find a more efficient and robust solution.
In this context, we chose to try the Radial Basis Functions(RBFs). To be more specific, RBFs, also called multivariate Gaussian distribution are a general form of a univariate normal distribution but in higher dimensions. This property gave us the idea that RBFs would be a good fit to our research problem due to the fact that continuous state and action spaces in our domains deal with high dimensions.
Generally speaking, one of the characteristics of a RBF is that its response depends monotonically on a central point. Taking this into consideration, we thought that having a fixed point which is the greedy action policy of the agent, we could take a number of different centres which in our case, is the mean $\boldsymbol{\mu}$ value and choose the one which is closer to the greedy action $\boldsymbol{\alpha}$. Thus, we could compare many modes and finally select the optimal one.
It is known that a radial function is a Gaussian and we have a scalar input. So, the function is defined as:

$$h(x) = exp\left(-\frac{(x-c)^2}{r^2}\right)$$

where

$$c : \text{centre}$$
$$r : \text{radius}$$

So far, we have extensively explained that the advantage term has a multimodal form and consists of a vector of actions. Taking this into consideration, it becomes clear that even our hypothesis to use RBF is correct, we could

not apply a univariate distribution because the actions are not a scalar but a vector of different actions. Hence, we had to choose a multivariate Gaussian distribution. The mathematical expression of the RBF is defined as:

$$\boldsymbol{N(\mu, \Sigma)(x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} exp\left(-\frac{1}{2}\left((x-\mu)^T \Sigma^{-1}(x-\mu)\right)\right)}$$

where

$$\mu \in R^n : \text{mean value}$$
$$\Sigma : \text{the covariance matrix}$$

Looking at the definition, it becomes clear that the argument of the exponential function $-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$ is a quadratic form with respect to the vector x. This feature allowed us to investigate the multimodal representations of our algorithm.
Below, we can see a graphical representation both of a univariate and a multivariate distribution.

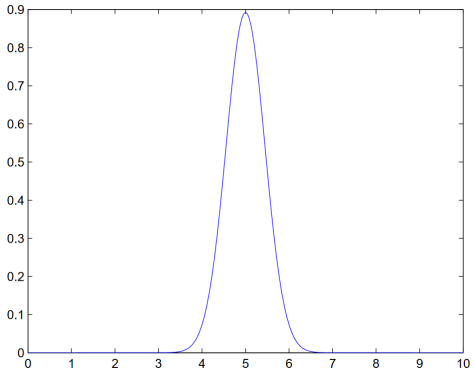


Figure 5.8: A univariate Gaussian distribution.

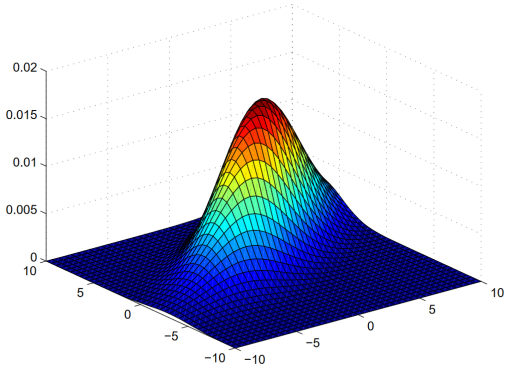


Figure 5.9: A multivariate Gaussian distribution.

The next steps that we follow are the same ones as we described in Section 5.6.1. In this case we tested again both the sum of multiple advantage terms in a multivariate representation and the selection of the multivariate distribution that gives the optimal and global response by using the argmax. In order to have a fair comparison between our two tested methods, we decided to use only four advantage terms. Hence, regarding the sum of multiple advantage terms which are equally weighted the final advantage term is defined as:

$$A = \boldsymbol{N_1}(\mu_1, \Sigma) + \boldsymbol{N_2}(\mu_2, \Sigma) + \boldsymbol{N_3}(\mu_3, \Sigma) + \boldsymbol{N_4}(\mu_4, \Sigma)$$

Instead, in the case of the argmax, the advantage term is defined as:

$$A = \text{argmax}(N_1, N_2, N_3, N_4)$$

All these methods, as it is shown in Chapter 7 are tested and evaluated in a great number of simulated tasks under certain circumstances. The performance, the efficiency and the results are presented and discussed in detail.

Chapter 6

Experimental Setup

The rapid evolution of DRL forced the community to develop platforms for testing and evaluating the new algorithms that arose. For instance, in supervised learning, large labelled datasets, such as ImageNet [54], have led to a great progress in this field. Instead, in the field of RL a large and diverse collection of environments are required. Also, the lack of standardization of the tasks used in publications which means that it becomes difficult to reproduce published research and compare outcomes from different papers made the RL research slow down. For this problem to be fixed, different platforms have been developed. In this thesis we decided to use the OpenAI Gym and more specifically, the Roboschool.

6.1 Roboschool

Gym [55] is a toolkit for developing and comparing reinforcement learning algorithms. A very important feature is that it is compatible with any numerical computation library, such as TensorFlow [56] which we also used and we will describe it later in this chapter. Although the gym library is a set of test problems -environments- that can work out our RL algorithms, we limited the selection of the tasks and chose what we needed. In this context,

we used Roboschool. The Roboschool is an open-source software for robot simulation, integrated with OpenAI Gym. Roboschool provides a set of different environments for controlling robots in simulation. These environments are extremely challenging and provide a free alternative solution to MuJoCo [57] implementations. It is worth mentioning that Roboschool is based on the Bullet Physics Engine [58], an open-source physics library that has been used by other simulation software such as Gazebo and V-REP [59].

6.2 Environments

Roboschool provides twelve environments with tasks which are challenging and realistic. In this thesis, we selected six of them in order to test our algorithm. The selection is based on what computational resources we had and on covering a wide range of different tasks. Figure 6.1 shows an instance of each environment, while Table 6.1 gives a short description of each domain.

Roboschool domains	
Domain	Description
Pendulum	Try to keep a frictionless pendulum standing up.
InvertedPendulum	Continuous control version of classic cartpole problem. Keep a pole upright by moving the 1D cart.
InvertedDoublePendulum	Two-link continuous control version of classic cartpole problem. Keep two-link pendulum upright by moving the one-dimesional(1D) cart.
Humanoid	Make a three-dimensional(3D) bipedal robot walk forward as fast as possible, without falling over.
Ant	The four-legged ant should move toward the fixed target from a fixed starting position and posture.
Walker2d	Agent should move forward as quickly as possible with a bipedal walker constrained to the plane without falling down or pitching the torso too far forward or backward.

Table 6.1: List of domains.

6.3 Programming Language of Choice

In the field of Machine Learning and generally whatever deals with Artificial Intelligence (AI) and Data Science problems, Python is the first choice as the programming language. The whole coding infrastructure of our implementation was done using Python and its scientific and numerical libraries, such as numpy and matplotlib to name a few. This choice is based on the fact that Python is a high-level language with simple syntax, object-oriented, powerful enough to execute our algorithm and computationally efficient.

6.4 TensorFlow

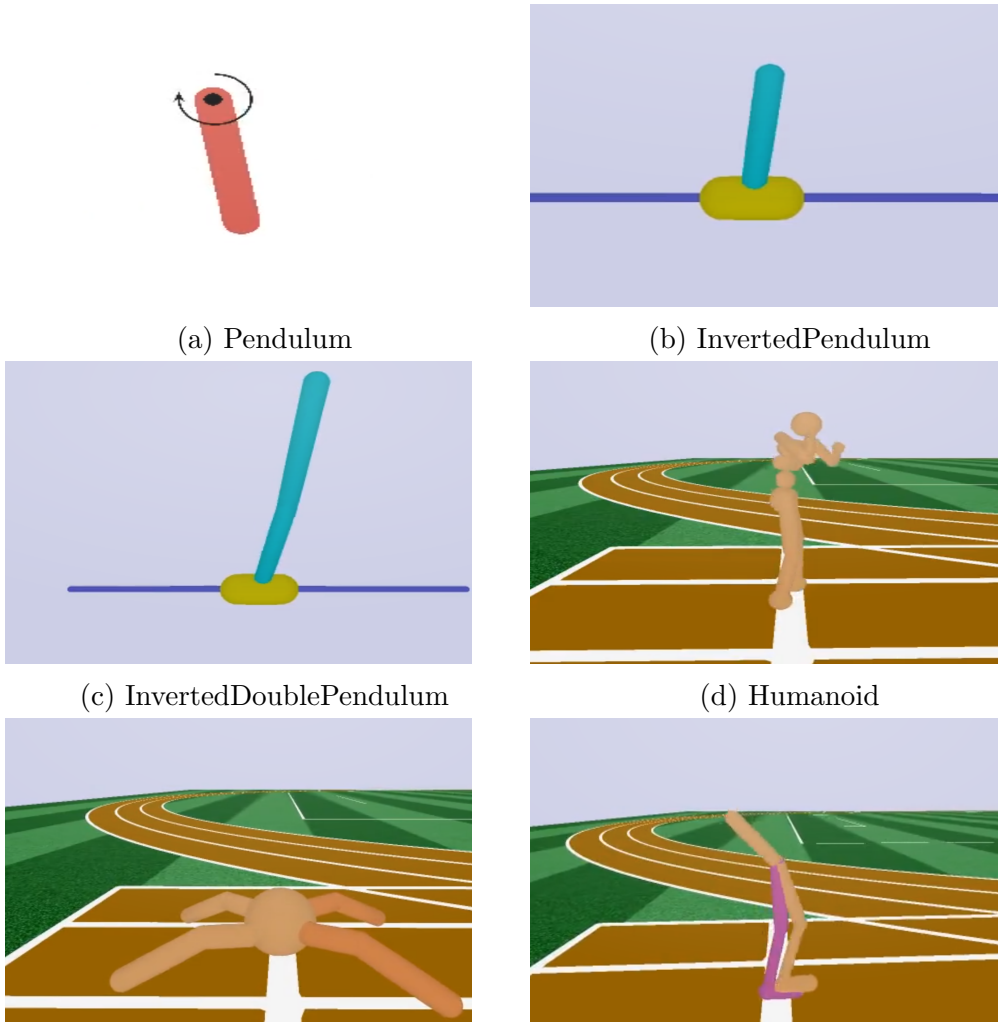
The design and the implementation of a neural network is not so simple as it may sound. It requires a lot of attention on how to build it and what tools to use. In this context, TensorFlow [56] is the best solution so far. In our case, we had to implement a complex neural network because of the high dimensionality both of inputs and outputs. In order to avoid exploding gradient descents and frequent system failures we decided to use the TensorFlow. It is an open source software library for numerical computation using data flow graphs. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. TensorFlow has been the fundamental tool in our complicated implementation and in even more demanding numerical computations.

6.5 Operating System, Hardware Specification and their Impact

An important role in our implementation and its results is played by the computational resources we have. We tried our application to be independent from the operating system. However, the open source libraries and platforms we used are based mainly on Linux distribution thus we use one of them which was the Ubuntu 16.04 version. We selected this version because it was easy to install and execute the numerical methods which the implementation required.
However, many domains require strong computational resources because they have to deal with high-dimensional tasks. Unfortunately, that was a limitation for us because we benchmarked our application in a 32-bit system with a i3 dual core CPU. This limitation led us not to test many combinations and we restricted ourselves to the basic ones.

6.6 Code Structure

As we mentioned in Section 6.3, the programming language of our code is Python. Python is a very simple language and does not require header files like C/C++ for example. Hence, our code is structured in four files: the first file includes the whole infrastructure of our neural network, declaring the structure of the input, output and hidden layers. Also, it includes the implementation of the replay buffer for storing the experience during the training time. The second python file includes the hyperparameters such as the learning rate, the batch size, the number of hidden layers to name a few. This part of code is arranged in such a way that only these parameters need to be changed by any user and test their combinations. The most important part of this file is the NAF algorithm itself implemented as close as possible to the original one. The third file includes the whole implementation of the exploration. Exploration is a very crucial part of the training process thus we decided to distinguish it from the other parts. The fourth and final file is about the data visualization of our outcomes. It is used to depict the outcome for each combination of the methods and the domains that we tested.



(a) Pendulum

(b) InvertedPendulum

(c) InvertedDoublePendulum

(d) Humanoid

(e) Ant

(f) Walker2d

Figure 6.1: Graphical depictions of the six Roboschool domains we used.

Chapter 7

Results

7.1 The Evaluation Process

In Chapters 5 and 6 we provided a detailed analysis of how we implemented our methods and what domains we used. During the experimentation procedure, we received some valuable results that gave us the ability to compare our algorithm and our proposed approach on how to deal with multimodal representations in the field of Deep Reinforcement Learning. In order to evaluate our methods, we firstly use a random agent as our baseline. It is very important to have a starting point of comparison which is totally independent and based on randomness. We used this agent in every environment. The next was to evaluate the unimodal approach of the NAF algorithm but as we implemented it. As a first comparison step, we compared our implementation to our baseline, i.e. the random agent. Again, this method was applied to every environment. The final step was to evaluate our two proposed methods, the multiple quadratics and the RBFs, where each one used two techniques, one using weights for each advantage term and the second using the argmax. Since we applied our proposed methods and tested them to the six domains, we extracted valuable curves for each methodology. The comparisons are based on all applied methods evaluated for each domain.

Sections 7.2 and 7.3 describe in detail the parameters we used both in our experiments and in the architecture of the neural network.

7.2 Parameters of the Experimentation Process

Generally, in the field of Machine Learning we need to train models in neural networks for them to learn and act independently. However, the training of a model requires some significant parameters to be set. In our work, we firstly had to choose the number of timesteps and episodes that we would use in our environments. During our research, we concluded that, due to the different domains where in some of them we had to deal with 3D simulated robots thus high-dimensional actions, we had to use a large number of episodes, a suggestion proposed in literature as well. However, our limited computational resources and the pressure of time forced us to a manageable number of episodes. In this context, we train our models for 3000 episodes, each one with a maximum of 1000 timesteps which means that the whole training procedure could reach the 3M timesteps. An exception to this was the environment 3D simulated robot, called Humanoid. Because of its nature and in order to achieve better results, we run our experiments in 5000 episodes with a maximum of 1000 timesteps, leading to a maximum of 5M timesteps. We noticed that these numbers would lead our system to run for a long time because we used a CPU for calculations rather than a GPU. Moreover, in order to extract more accurate and fair results, we run each experiment 5 times and we use the average value of these five iterations as the final reward value for each episode.

7.3 Neural Network Hyperparameters

Apart from the parameters that we need to choose for the experimentation procedure, the hyperparameters of a neural network plays the most important role in the whole procedure. Table 7.1 shows the values of hyperparameters for our network.

Hyperparameters	
Parameter	**Range of value**
Episodes	3000
Timesteps	1000(max)
Training iterations	5
Learning rate(ADAM)	**0.01**, 0.001, **0.0001**
γ	0.99
τ	0.01
ϵ	0.1, **0.3**, 1
Hidden layers	2
Hidden size	**32**, 64, **128**, 256
Buffer memory	$10e6$
Batch size	32, **64**, **100**, 200
Batch normalize	True, False

Table 7.1: Parameters of our system. Bold font indicates the most used value of the corresponding parameter.

The selection of these values are not in random. Instead, this process requires many trials in order to find the most suitable value for each occasion. In this work, we decided to set fixed values for each hyperparameter, especially when the domains are similar. For instance, environments such as Pendulum and InvertedPendulum, had better performance when learning rates close to 0.01 were used. Instead, tasks which needed more expressiveness and had high-dimensionality actions performed better when learning rate close to 10^{-4} was applied.
The different values of learning rate led us to conclude that our system was sensitive to some hyperparameters. So, we concluded that the learning rate,

the use or not of batch normalization and the values of the exploration noise are affected our system. It becomes clear that we had to make any possible combination for the ideal structure of our system but the many limitations prevented us and we finally used fixed values which were more suitable for the majority of the environments.
As we can see in Table 7.1, hidden size and batch size can take different values and not fixed. That was an exception we did because when the task was easier to be solved, small number of hidden and batch size were required so there was no need our network to be more complicated which many times led it to collapse. Instead, environments with high dimensionality required larger sizes in order good training models to be achieved.

7.4 Comparisons

Having run many experiments and conducted a great number of tests, it is time to present and compare our results. As we have already mentioned many times, we tested our methods in six different domains, each of them with its own characteristics and we developed two methods, multiple quadratics and RBFs tested with two different action selection approaches each of them. Also, we used a random agent as baseline and we implemented the unimodal representation as well. Hence, we organized our comparisons as follows: firstly, we will present the results for unimodal representation and will compare it with the baseline. Secondly, we will compare the outcome in each domain both of multiple quadratics and RBFs. In addition, a comparison between the methods of weights and argmax will be presented. Finally, for each task, we will present the result of all combinations and we will discuss it.

7.4.1 Baseline Agent vs Unimodal Approach

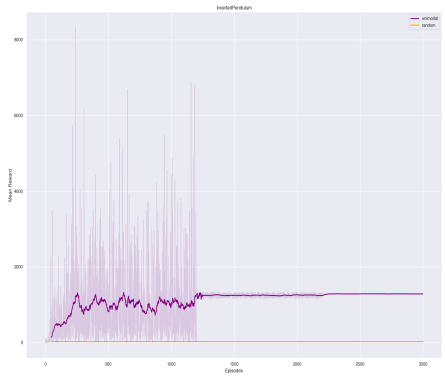
Figure 7.1 shows the mean rewards that each agent achieved in every domain. More, the average raw reward for each episode is depicted(the faded color), too. At a first glance, it becomes clear that our first goal was successful as the unimodal agent outperforms the random agent in all environments.

In all domains, we see that the random agent has a steady behaviour and practically it cannot make progress. Besides, it takes random moves so it is normal not to perform well. On the other hand, the unimodal approach that we implemented based on - as close as possible - the instructions of the NAF algorithm behaves very well compared to the random agent which was the first goal. This means that generally there is an improvement and our agent performs well. It is worth mentioning some important observations:
In Figure 7.1b we see that, at the beginning, the agent in unimodal form has a deviation between the raw values of the reward but after some episodes we see that the mean and the raw reward converge. This is a very important feature of our approach because it indicates that our agent after a number of training episodes, it becomes stable. This behaviour is also observed in Figures 7.1e and 7.1d.
In Figures 7.1e and 7.1d we observed that our agent achieves high rewards at early episodes. This is a huge improvement because these two tasks simulate a 3D robot which means that the agent has to deal with high dimensional actions.
All in all, our agent presents very good results compared to the baseline agent and we notice improvements.

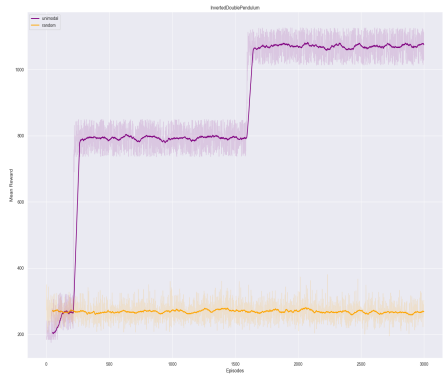
7.4.2 Multiple Quadratics Method

Figure 7.2 depicts the results of our first proposed method. This method consists of two other techniques, the use of weighted terms and the selection of the advantage term using the argmax. In this section, we compare these two techniques and discuss the most important outcomes.

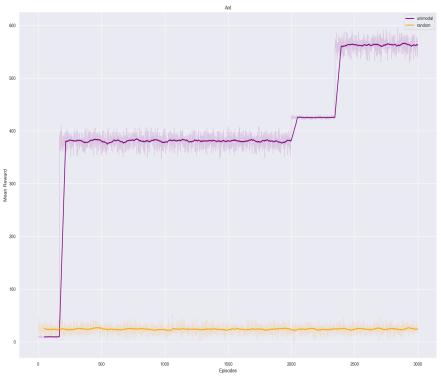
(a) Pendulum



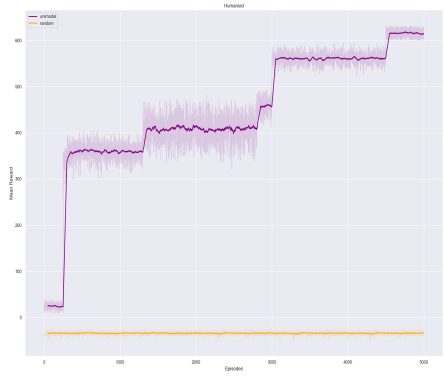
(b) InvertedPendulum



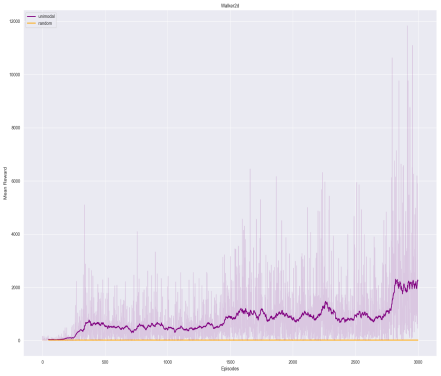
(c) InvertedDoublePendulum



(d) Ant



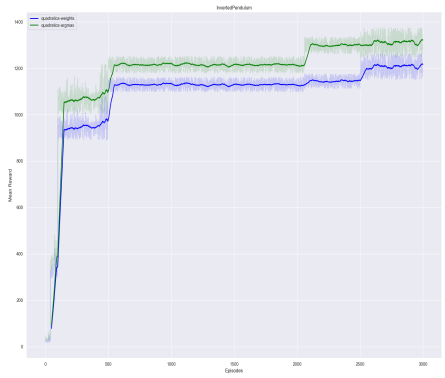
(e) Humanoid



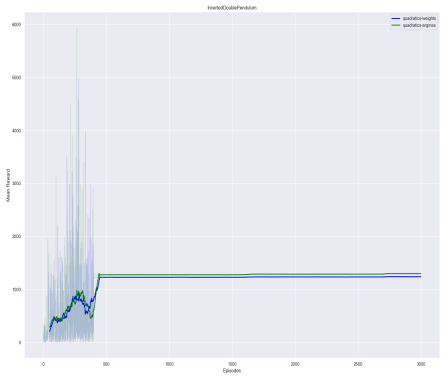
(f) Walker2d

Figure 7.1: Performance of the unimodal approach vs a random agent. The faded color depicts the average raw reward that the agents achieved in every episode.

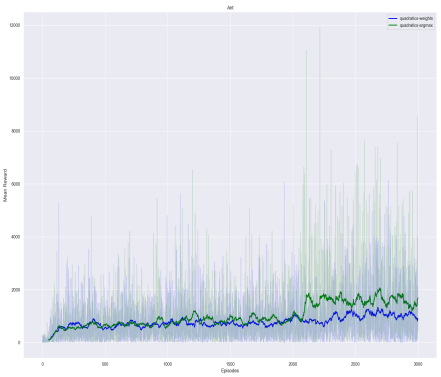
(a) Pendulum



(b) InvertedPendulum



(c) InvertedDoublePendulum



(d) Ant

(e) Humanoid

(f) Walker2d

Figure 7.2: Performance of the agent using Multiple Quadratics method. The faded color depicts the average raw reward that the agents achieved in every episode.

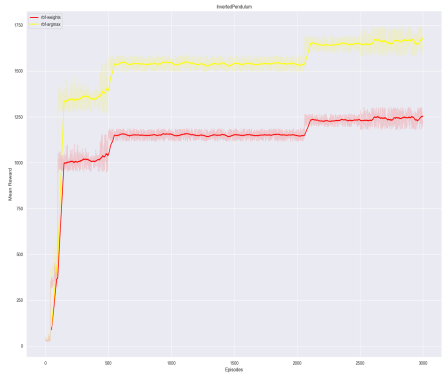
Is has become clear that the more high dimensional actions a domain has, the more complicated it is to be solved. We mention this because we observe in Figures 7.2d, 7.2e, and 7.2f that the agent performs slightly different regardless which technique it uses. Hence, we can conclude that in 3D tasks, our multimodal approach using multiple quadratics has almost the same behaviour. This does not mean that our agent does not perform well. Instead, it achieves high rewards but the two options of multiple quadratics have similar performance.
In Figures 7.2a and 7.2c we can see that both techniques behave similarly. But, the most important thing in this behaviour is that both of them, after a certain training time, converge and become stable. This observation is very obvious because as we can see, the mean rewards values and the raw rewards values are almost the same or there is a tiny deviation between them. Hence, we conclude that both techniques behave properly in such domains.
Overall, we see that the approach with multimodal distribution, in particular with multiple quadratics using weights and argmax can provide great performance and good results regardless of which technique we use.

7.4.3 RBFs Method

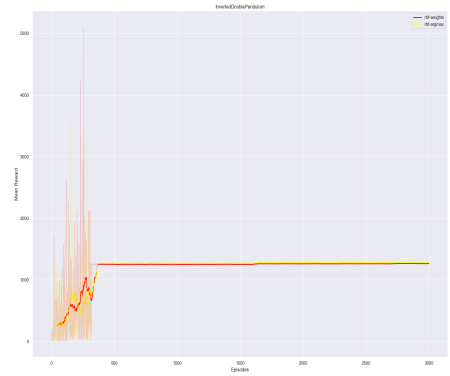
As we presented the results of the Multiple Quadratics method, Figure 7.3 depicts the outcome of RBFs method in each domain. We implemented the RBFs method following the applied procedure in Multiple Quadratics. Hence, we discuss the results of the weights and argamax techniques and we compare their performance.

Looking at the general picture, we come to a conclusion about these two techniques. They behave almost similarly and there are only slight differences in the values of mean reward. Thus, they perform both very well. The only significant difference in their performance is observed in Figure 7.3b. In this task, we see that using the argmax technique for implementing the RBFs method and approaching a good multimodal representation outweigh

(a) Pendulum

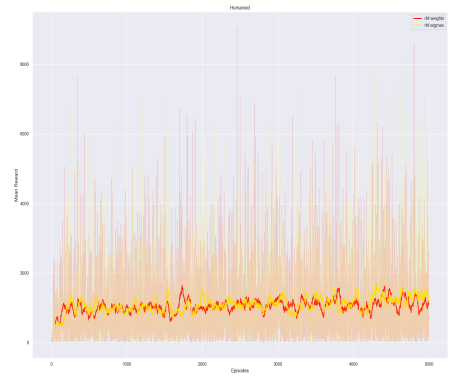


(b) InvertedPendulum

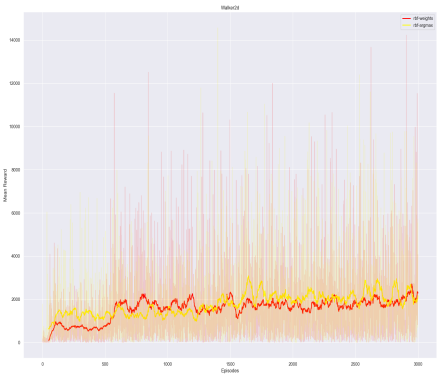


(c) InvertedDoublePendulum

(d) Ant



(e) Humanoid



(f) Walker2d

Figure 7.3: Performance of the agent using RBFs method. The faded color depicts the average raw reward that the agents achieved in every episode.

the use of the weighted technique. This is clear when we see closely that there is a constant difference in the rewards from nearly the beginning of the training time.
In Figures 7.3a, 7.3b and 7.3c, we can also observe that the deviation between the mean reward and the average raw reward is almost minimal. This leads us to conclude that in 2D domains such as Pendulum, and Inverted-DoublePendulum the weighted and argmax techniques can make the agent to converge fast and to have a steady good performance.
Instead, Figures 7.3d, 7.3e and 7.3f show that there is a big deviation in the values of mean and average raw rewards. This means that despite the high rewards that present, they may lead the agent to instability and, in the end, it may not make a significant progress.
To conclude, in five out of six domains, the two techniques perform the same well and we can apply any of them. In addition, we found that when we deal with 3D simulated robots, there is a divergence in rewards which may lead the agent to fail in the long run. However, in 2D tasks, our techniques can be efficient and our agent is trained very well.

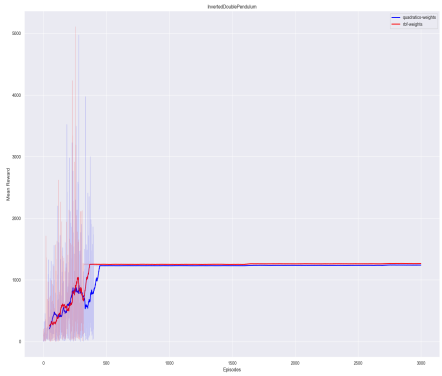
7.4.4 Comparison of the Weighted Technique

It is useful to compare the technique of weights and its behaviour in our two different methods which we used for our approach. In this way, we can examine its performance in the collection of the selected environments. Figure 7.4 shows a comparison of the use of weights between the two methods and we can extract meaningful results.

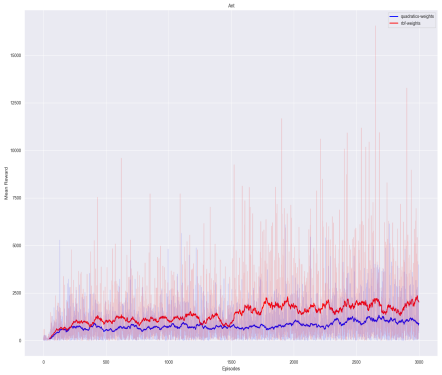
In general, it becomes clear from the results in all domains that the implementation of RBFs with weights slightly outperforms the use of weights in Multiple Quadratics. This leads us to choose the RBFs method in order to achieve better results and to be able to keep our whole system stable.
Specifically, we observe that in domains with lower dimensional actions this technique achieves a convergence in rewards in both methods. Figures 7.4a, 7.4b and 7.4c can prove this declaration. Hence, the agent learns faster and it

(a) Pendulum

(b) InvertedPendulum



(c) InvertedDoublePendulum



(d) Ant

(e) Humanoid

(f) Walker2d

Figure 7.4: Performance of the agent using weights in Multiple Quadratics and RBFs. The faded color depicts the average raw reward that the agents achieved in every episode.

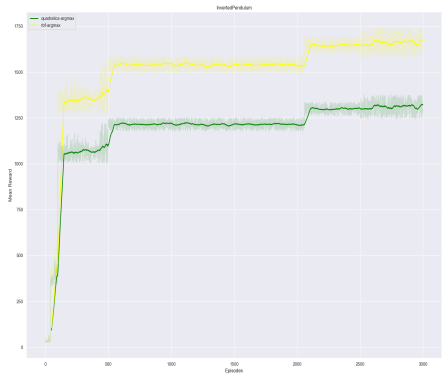
yields as high as possible. However, the required convergence is not observed in Figures 7.4d, 7.4e and 7.4f where the domains consist of simulated robots with high dimensional actions. The weighted technique still performs well and yields very good results regarding the rewards but the constant divergence between the mean reward and the reward of each episode may provide a poorer outcome to similar tasks.
Lastly, it is worth mentioning that our agent, in Figures 7.4a and 7.4c , shows to converge and be stabilized at the early stages of the training time(in $\approx 500^{th}$ episode), a feature that is very significant for training models.

7.4.5 Comparison of the Argmax Technique

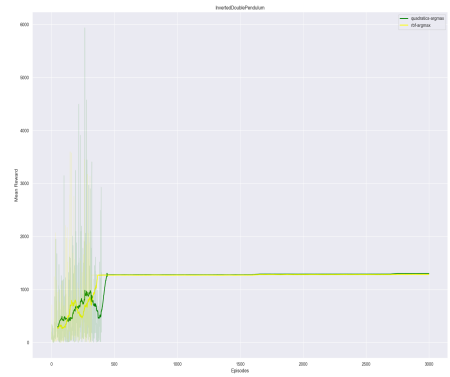
As we mentioned in Section 7.4.4, a comparison of each technique that we used in our methods can extract significant verdicts. The Figure 7.5 summarizes the performance of the argmax which we used to our approach and applied to the six domains.

Looking at the Figure 7.5, we can directly understand that the argmax outperforms when it is used by the RBFs. The differences may not be so big but they cannot be ignored as the training model must be efficient and effective in the end of training process.
To be more specific, Figure 7.5b depicts the difference between the two methods. It becomes clear that the RBF can yield more efficiently than Multiple Quadratics when they use the argmax. Moreover, we can observe that in both methods there is a convergence in mean and raw reward values, leading to a stable system and to a well trained model. This convergence is also observed in Figures 7.5a and 7.5c. The conclusion of these three Figures shows that when we have to deal with 2D simulated robots, the preferred approach can be any of the two methods we implemented because there are tiny differences in performance which are not affect the training model in overall.

(a) Pendulum

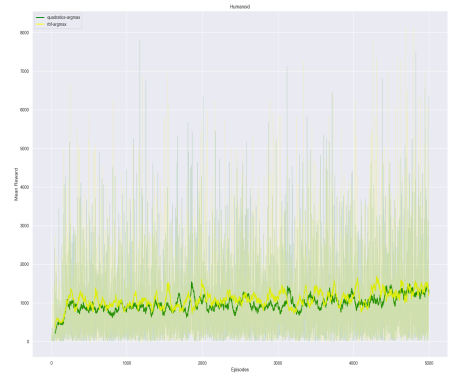


(b) InvertedPendulum

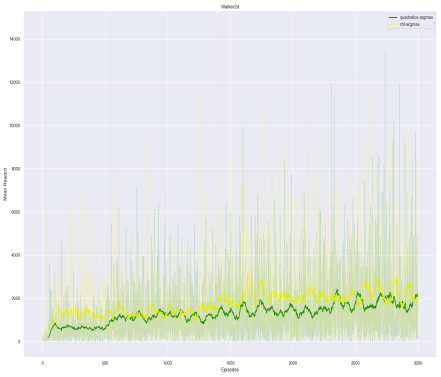


(c) InvertedDoublePendulum

(d) Ant



(e) Humanoid



(f) Walker2d

Figure 7.5: Performance of the agent using argmax in Multiple Quadratics and RBFs. The faded color depicts the average raw reward that the agents achieved in every episode.

However, this is not happen when the tasks have to train 3D simulated robots. In these cases, as it is shown in Figures 7.5d, 7.5e and 7.5f the best method is to choose the RBFs because it performs better than Multiple Quadratics with argmax. The only disadvantage that is observed is that the mean and raw reward values present a divergence which is possible to lead our agent not to be trained as well as it could be.

7.4.6 Overall Figures and Comparisons

In Sections 7.4.1, 7.4.2, 7.4.3, 7.4.4 and 7.4.5 we presented the results of subsets regarding our final algorithm and its verdict. However, it is really important to show and discuss the outcome of our approach in every domain and compare the random agent, the unimodal representation and our multimodal approach so as to understand the improvements that we have achieved. Below, we present the figure of each domain and we benchmark each method.

Pendulum

Figure 7.6 depicts the performance of each representation over 3000 episodes. The mean reward is the rolling mean over 50 episodes and that gives us a better visual perspective of our implementation.
The first and most important conclusion is that all of our methods outperform the baseline agent which fluctuates around -750. Therefore, there is an initial improvement of our proposed approaches.
The second conclusion is that our multimodal approach is far better than any other representation, both the baseline agent and the unimodal distribution. This is also very significant because the goal of this work is to improve the existed algorithm and it seems that we have achieved it regarding this domain.
However, these are the more general conclusions of our implementation. So, it is crucial to mention more specific facts of our experiments. For instance,

we can observe that our proposed methods, both the multiple quadratics and the RBFs, in $\approx 500^{th}$ episode, start to converge and have tiny deviations during the training time. This is the most important outcome because it shows us that our methods are pretty stable regarding this domain. But, we need to mention here that our agents present an unstable behaviour during the first 500 episodes. Lastly, it becomes clear that the method of RBFs using the argmax presents the biggest improvement even it has small differences with the other three methods. Hence, we could say that, in similar domains, we would choose any of the multimodal representations.

Figure 7.6: The performance of all approaches applied to Pendulum environment.

InvertedPendulum

Figure 7.7 depicts the performance of each representation over 3000 episodes. The mean reward is the rolling mean over 50 episodes and that gives us a

better visual perspective of our implementation.
The first and most important conclusion is that all of our methods outperform the baseline agent which fluctuates around 50. Therefore, there is an initial improvement of our proposed approaches.
The second conclusion is that one of our multimodal approaches is far better than any other representation, the baseline agent and the unimodal distribution. This is also very significant because the goal of this work is to improve the existed algorithm and it seems that we have achieved it regarding this domain.
However, these are the more general conclusions of our implementation. So, it is crucial to mention more specific facts of our experiments. For instance, we can observe that our proposed method of RBFs using argmax leads the agent to learn really fast and to be extremely stable and this makes it outperform any other agent in long term. It also very important to mention that all of our proposed methods present a stable performance and there is not big fluctuations thus they are stable, too. Moreover, we observe that two of our methods, the multiple quadratics using weights and the RBFs using weights do not perform better from the unimodal distribution. Hence, we received poorer performance in this domain for these two methods and we would not choose them to solve similar tasks. A possible explanation to this poor performance is that as the task is a bit more complicated than the Pendulum, the equal selection of weights does not offer any improvement in order to solve this task.
Overall, we can say that our method succeeded our initial goal which was to improve the behaviour of the agent and this happens even if it is done only by one method.

InvertedDoublePendulum

Figure 7.8 depicts the performance of each representation over 3000 episodes. The mean reward is the rolling mean over 50 episodes and that gives us a better visual perspective of our implementation.

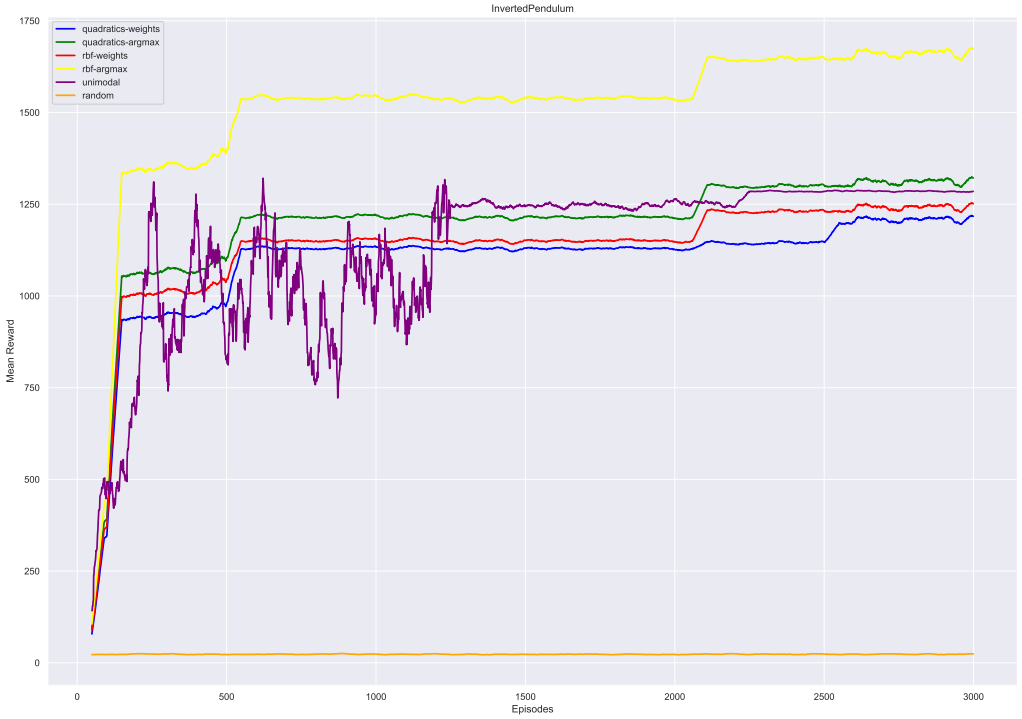


Figure 7.7: The performance of all approaches applied to InvertedPendulum environment.

The first and most important conclusion is that all of our methods outperform the baseline agent which fluctuates around 270. Therefore, there is an initial improvement of our proposed approaches.
The second conclusion is that the multimodal approaches that we propose outperform by far both the unimodal distribution and the baseline agent thus we consider that our methods improves the performance of the agent and extend the initial algorithm.
However, these are the more general conclusions of our implementation. So, it is crucial to mention more specific facts of our experiments. One of the most important observation that we need to mention is the stability of all of our methods. We see that they converge fast and there are no fluctuations, providing a stable performance to our implementation. Also, the Figure shows that during the first 500 episodes, our methods have ups and downs regarding the reward values which means that at the first stages of training

there were an unstable behaviour that forced our agent not to converge faster and not to act steadily. Lastly, in this domain, the principal method with the best performance is the use of multiple quadratics with argmax. It has a little difference with the others but this small leading makes it the first choice for applying to similar environments.

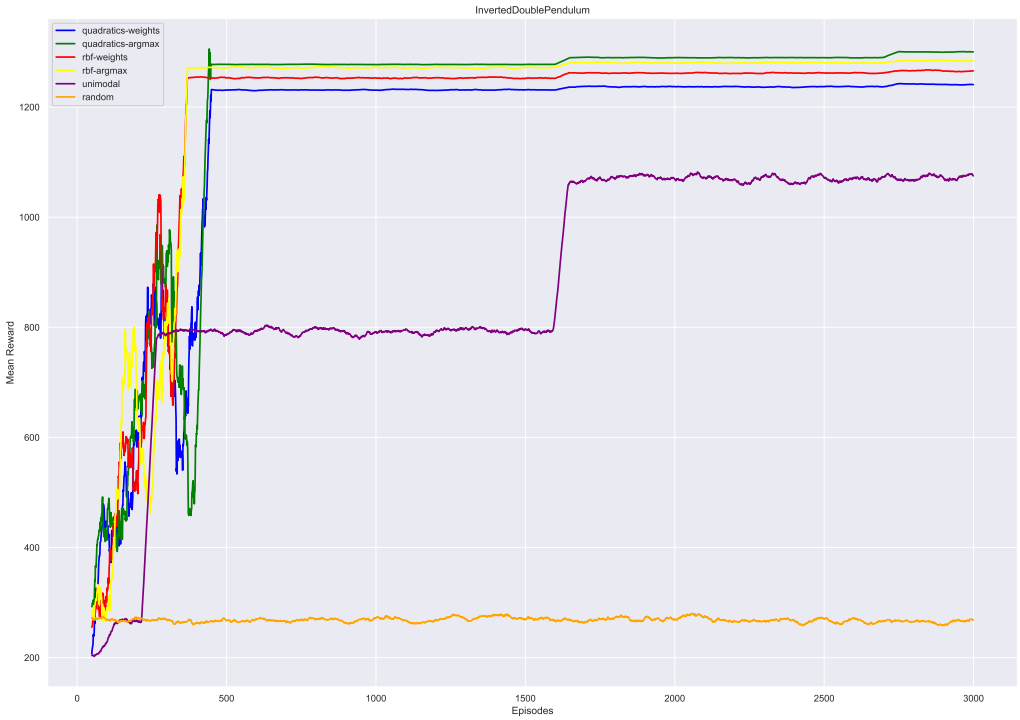


Figure 7.8: The performance of all approaches applied to InvertedDoublePendulum environment.

Ant

Figure 7.9 depicts the performance of each representation over 3000 episodes. The mean reward is the rolling mean over 500 episodes and that gives us a better visual perspective of our implementation.
The first and most important conclusion is that all of our methods outperform the baseline agent which fluctuates around 50. Therefore, there is an initial improvement of our proposed approaches.

The second conclusion is that all of our multimodal approaches are far better than any other representation, the baseline agent and the unimodal distribution. This is also very significant because the goal of this work is to improve the existed algorithm and it seems that we have achieved it regarding this domain.
However, these are the more general conclusions of our implementation. So, it is crucial to mention more specific facts of our experiments. In this domain, it becomes clear that the RBFs method outperforms the use of multiple quadratics and this is obvious from the beginning of the training time. Thus, a task with a 3D simulated robot requires a more expressive form in order to perform as well as possible. It is also obvious that the two techniques that are use in the RBFs have similar behaviour and there are only small differences regarding the rewards. However, we observed that in the end of the training time, the use of argmax performs much better not only than all methods but also than RBFs which use weights. This lead us to consider that a RBFs method is more suitable in domains with high dimensional actions.

Humanoid

In Figure 7.10 is shown the performance of each representation over 5000 episodes. The mean reward is the rolling mean over 500 episodes and that gives us a better visual perspective of our implementation.
The first and most important conclusion is that all of our methods outperform the baseline agent which fluctuates around -50. Therefore, there is an initial improvement of our proposed approaches.
The second conclusion is that all of our multimodal approaches are far better than any other representation, the baseline agent and the unimodal distribution. This is also very significant because the goal of this work is to improve the existed algorithm and it seems that we have achieved it regarding this domain.
However, these are the more general conclusions of our implementation. So,

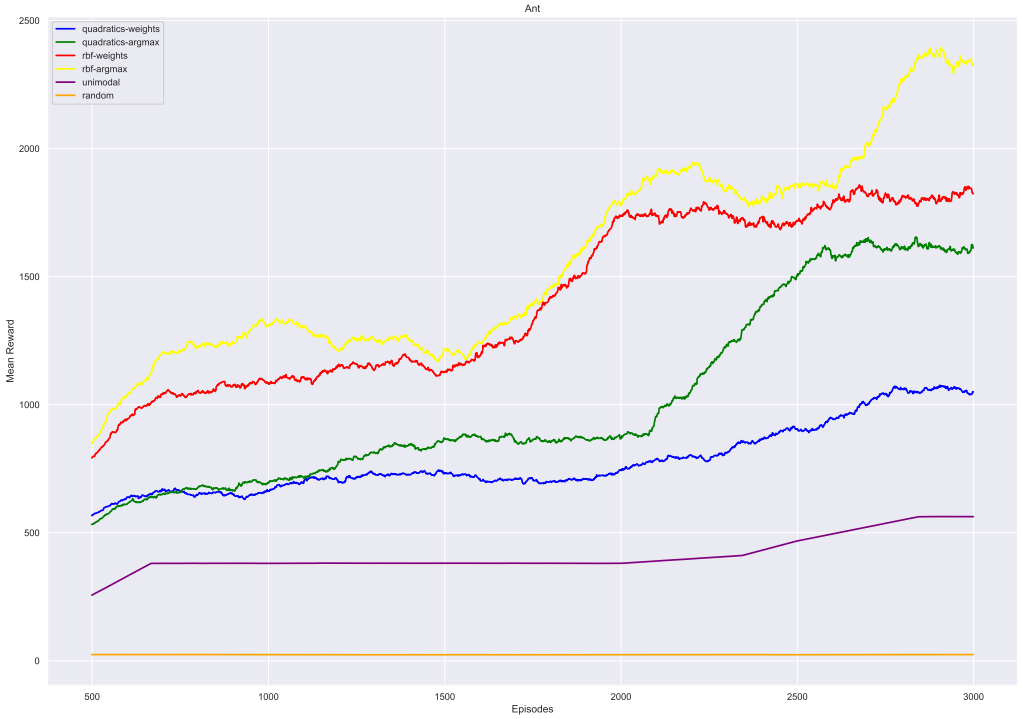


Figure 7.9: The performance of all approaches applied to Ant environment.

it is crucial to mention more specific facts of our experiments. Firstly, we see that the RBFs using argmax achieve the highest reward during almost the whole training time and this makes it more suitable for similar tasks. Secondly, we observe that the RBFs have in general, better and more stable performance than multiple quadratics thus we consider that in task with 3D simulated robots it is more efficient to use these methods. Moreover, multiple quadratics using argmax present a better achievement at the end of training time than RBFs which use weights. This leads us to conclude that the technique of argmax improves much more the performance of a 3D simulated robot, yielding higher rewards in high dimensional domains. Lastly, it is important to mention that the unimodal representation in such tasks which have continuous state and actions space presents a very performance regarding multimodal approaches.

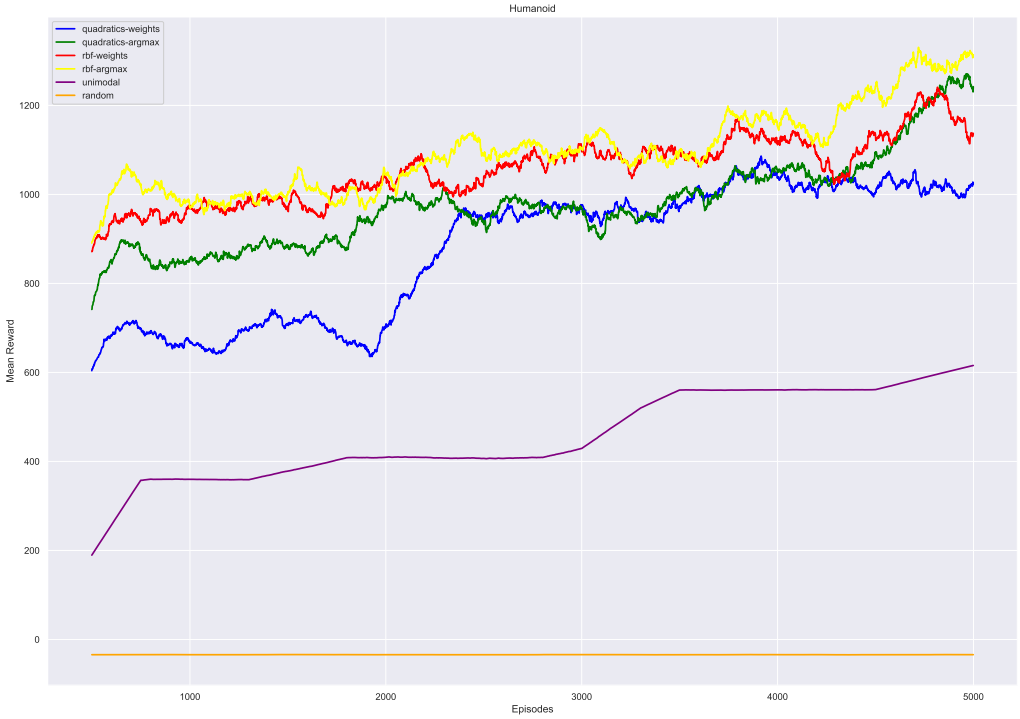


Figure 7.10: The performance of all approaches applied to Humanoid environment.

Walker2d

In Figure 7.11 is shown the performance of each representation over 3000 episodes. The mean reward is the rolling mean over 500 episodes and that gives us a better visual perspective of our implementation.
The first and most important conclusion is that all of our methods outperform the baseline agent which fluctuates around 30. Therefore, there is an initial improvement of our proposed approaches.
The second conclusion is that all of our multimodal approaches are better than any other representation, the baseline agent and the unimodal distribution. We need to note here that in the of training time, the unimodal representation presents a rapid increase. This means that if we had more resources to run experiments for a longer time we might see eventually the unimodal to overcome one of the multimodal approaches or to collapse. How-

ever, the goal of this work is to improve the existed algorithm and it seems that we have achieved it under these circumstances.

At a first glance, these are very important conclusions but we need to discuss some more specific observations that derive. Firstly, it becomes clear again that the RBFs methods, especially the one which use argamax, outperform the multiple quadratics methods and the unimodal representation, achieving higher rewards and thus they become more suitable for similar domains. Also, it is worth mentioning that the RBFs methods yield higher rewards than multiple quadratics thus they present an overall better performance and behave more efficiently in such domains. Lastly, it is shown a fluctuation for all methods. We believe that this happens duo to high-dimensionality of the tasks thus it makes the agent not to converge and have a stable performance.

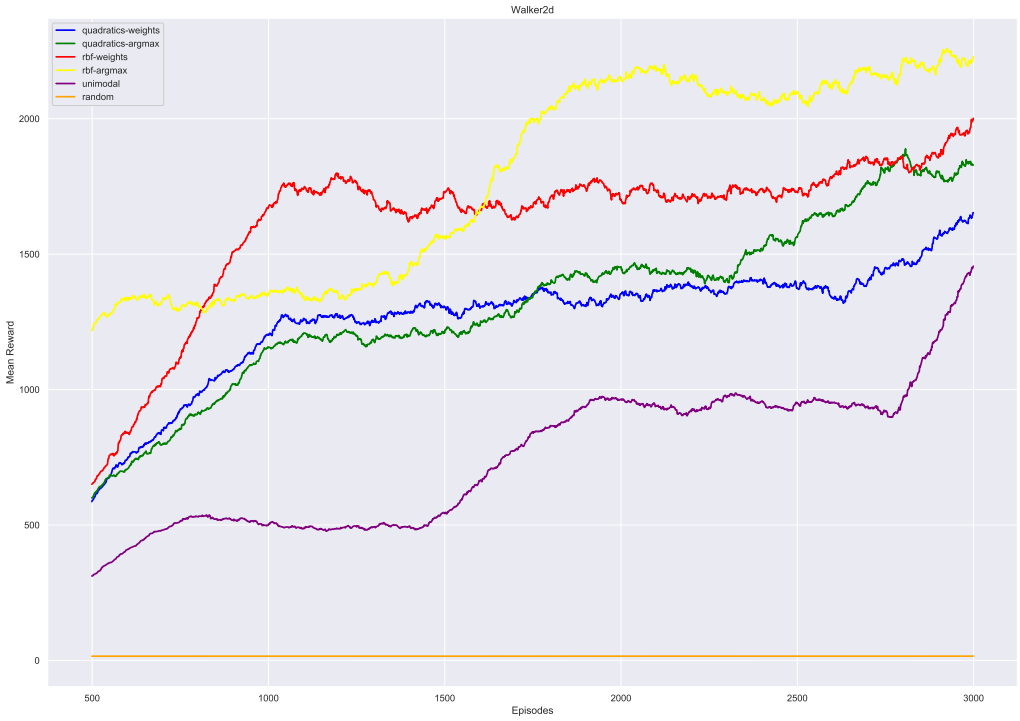


Figure 7.11: The performance of all approaches applied to Walker2d environment.

Chapter 8

Conclusion

8.1 Discussion

Our results show that the proposed approach of multimodal representations does outperform the original unimodal NAF algorithm. In each tested task, we received a better outcome which makes it clear that our new version has succeeded its initial goal.
What is noteworthy is that under limited computational resources and with no extensive literature references about this issue, we managed to prove that the reimplementation of Q-learning in continuous actions and state spaces is feasible and can produce a robust and alternative solution in high dimensional spaces. Delving more into these results, we can conclude that under laboratory conditions we achieved high scores in every simulated task, even it was a 2-D or 3-D robot, and most importantly, we showed that the unimodal approach does not exploit the whole functionality of NAF algorithm, a situation that we were able to alter.
To date, literature does not have a standard way to measure the performance of RL algorithms for each environment. Moreover, many of these tasks are considered unsolved thus there is no an official way to compare our results. However, we can extract valuable insights comparing the two distributions

and that is what we did. We observed high scores of our implementation with respect to the original distribution which means that our benchmarks are based on fair comparisons. Hence, we understand the importance of our results as they can be used as a means of future comparisons for any similar algorithm that may test the multimodularity in such domains.
Although we produced very good results, we did not test them extensively due to computational limitations. This is a very important issue because we could extract even better results and eventually apply our implementation to a real physical system. The acquisition of a powerful machine that can run simultaneously many combinations of hyper-parameters for each environment can be further tested either using our implementation or exploring a new version. In addition, we observed that our approach had a slightly increased computation cost compared to the unimodal representation. This fact is rational because we export more advantage terms for implementing the multimodal representations and this leads to a time-consuming behavior, delaying the execution process and increasing the size of the NN.

8.2 Future Work

The idea of multimodularity can be explored even more as it is an open field for research. This work examined only a small portion of how to deal with high-dimensionality in continuous state and action spaces. There are many aspects that can be reimplemented, improved or extended. For instance, this work could be tested in all of the environments that OpenAI Gym provides and maybe in other similar platforms. Therefore, a more general model could be built which would be ready to be tested in real-world tasks. Apart from that, another aspect for further investigation would be the experimentation with every possible combination of the most appropriate hyperparameters. In this context, the new algorithm could be applied to a range of tasks regardless the features of each task, thus it would be easier to be adopted by any physical system.
As far as the implementation is concerned, there is a lot of room for adjusting

new methods for exploring multimodal representations. New ideas can come out and be compared with the ones we propose on this thesis. Moreover, the existing neural network could be extended in order to produce more outputs, such as the weights of the multiple quadratics and RBFs thus the algorithm would be optimized.
An interesting idea for the future would the extensive test of these methods both to more complicated simulated tasks but also to real-life systems. In this way, we could understand if our methods have a significant impact to real world problems and the idea of multimodularity could be implemented not only using the NAF algorithm, but other baseline algorithms for deep reinforcement learning such as DDPG algorithm.

8.3 Conclusions

It may sound trivial but our results, from our point of view and comparing them with the literature, are very decent and if we consider that we proposed a new approach that does not have extensively been explored, then our impact to this field of research has a positive sign. Of course, our experiments were performed in laboratory conditions and our methods were not tested in real-world systems. However, our results indicate that there is an important improvement in performance regarding our proposed methods for multimodal representations regarding the published methods so far. This makes us believe that our approach can trigger the community for further research in this field. It is important to highlight that we conducted our research and run our experiments under computational limitations as we did not have the appropriate resources. Hence, a more state-of-the-art system could achieve the results much faster and we could test even more parameters of our model. Therefore, it becomes clear that given enough computational resources and extending and improving these methods or implementing more robust ideas, the verdict will be much more outstanding regarding the field of Deep Reinforcement Learning and its application to continuous action spaces thus in physical systems.

Bibliography

[1] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015.

[2] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," 2016.

[3] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[4] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957. [Online]. Available: http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[6] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996. [Online]. Available: http://people.csail.mit.edu/lpk/papers/rl-survey.ps

[7] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104451

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[10] T. Tieleman and G. Hinton., "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." COURSERA:Neural networks for machine learning, 4(2), 2012.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[12] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[13] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2021068

[14] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, Jan. 1999. [Online]. Available: http://dx.doi.org/10.1016/S0893-6080(98)00116-6

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol.

abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[16] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[17] J. Schmidhuber, "Deep learning in neural networks: An overview," *CoRR*, vol. abs/1404.7828, 2014. [Online]. Available: http://arxiv.org/abs/1404.7828

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Machine Learning*, vol. 49, no. 2, pp. 291–323, Nov 2002. [Online]. Available: https://doi.org/10.1023/A:1017992615625

[20] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *CoRR*, vol. abs/1207.4708, 2012. [Online]. Available: http://arxiv.org/abs/1207.4708

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[22] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[23] R. Hafner and M. Riedmiller, "Reinforcement learning in feedback control," *Machine Learning*, vol. 84, no. 1, pp. 137–169, Jul 2011. [Online]. Available: https://doi.org/10.1007/s10994-011-5235-x

[24] T. de Bruin, J. Kober, K. Tuyls, and R. Babuska, "The importance of experience replay database composition in deep reinforcement learning," 01 2015.

[25] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1–9. [Online]. Available: http://proceedings.mlr.press/v28/levine13.html

[26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[27] N. Brown and T. Sandholm, "Libratus: The superhuman ai for no-limit poker," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 5226–5228. [Online]. Available: https://doi.org/10.24963/ijcai.2017/772

[28] M. Moravcík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. H. Bowling, "Deepstack: Expert-level artificial intelligence in no-limit poker," *CoRR*, vol. abs/1701.01724, 2017. [Online]. Available: http://arxiv.org/abs/1701.01724

[29] Y. You, X. Pan, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," *CoRR*, vol. abs/1704.03952, 2017. [Online]. Available: http://arxiv.org/abs/1704.03952

[30] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, Jan. 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=2946645.2946684

[31] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," *CoRR*, vol. abs/1704.05588, 2017. [Online]. Available: http://arxiv.org/abs/1704.05588

[32] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," *CoRR*, vol. abs/1710.06542, 2017. [Online]. Available: http://arxiv.org/abs/1710.06542

[33] V. François-Lavet, "Contributions to deep reinforcement learning and its applications in smartgrids." 2017.

[34] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653–664, March 2017.

[35] J. Gauci, E. Conti, Y. Liang, K. Virochsiri, Y. He, Z. Kaden, V. Narayanan, and X. Ye, "Horizon: Facebook's open source applied reinforcement learning platform," *CoRR*, vol. abs/1811.00260, 2018. [Online]. Available: http://arxiv.org/abs/1811.00260

[36] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. C. Courville, and Y. Bengio, "An actor-critic algorithm for sequence prediction," *CoRR*, vol. abs/1607.07086, 2016. [Online]. Available: http://arxiv.org/abs/1607.07086

[37] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba, "Sequence level training with recurrent neural networks," *CoRR*, vol. abs/1511.06732, 2016.

[38] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *CoRR*, vol. abs/1611.01578, 2016. [Online]. Available: http://arxiv.org/abs/1611.01578

[39] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017. [Online]. Available: http://arxiv.org/abs/1703.00548

[40] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *CoRR*, vol. abs/1703.01041, 2017. [Online]. Available: http://arxiv.org/abs/1703.01041

[41] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *CoRR*, vol. abs/1611.09940, 2016. [Online]. Available: http://arxiv.org/abs/1611.09940

[42] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015.

[43] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[44] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On stochastic optimal control and reinforcement learning by approximate inference (extended abstract)," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI '13. AAAI Press, 2013, pp. 3052–3056. [Online]. Available: http://dl.acm.org/citation.cfm?id=2540128.2540576

[45] L. C. Baird III, "Advantage updating." 1993.

[46] M. E. Harmon, I. Leemon C. Baird, and A. H. Klopf, "Reinforcement learning applied to a differential game," *Adaptive Behavior*, vol. 4, no. 1, pp. 3–28, 1995. [Online]. Available: https://doi.org/10.1177/105971239500400102

[47] M. E. Harmon and L. C. Baird III, "Multi-player residual advantage learning with general function approximation."

[48] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, pp. 1057–1063. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009657.3009806

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[50] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[51] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992699

[52] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *CoRR*, vol. abs/1702.08165, 2017.

[53] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Phys. Rev.*, vol. 36, pp. 823–841, Sep 1930. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.36.823

[54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[55] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016.

[56] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[57] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," 2012.

[58] E. Coumans, "Bullet physics simulation," in *ACM SIGGRAPH 2015 Courses*, ser. SIGGRAPH '15. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2776880.2792704

[59] M. F. E. Rohmer, S. P. N. Singh, "V-rep: a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.