# $Q$-Learning with a Degenerate Function Approximation


Alejandro Gabriel Agostini

Institut de Robòtica i Informàtica Industrial

CSIC - UPC

A thesis advised by:

Enric Celaya Llover

A thesis for the degree of

*Doctor of Philosophy*

Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
PhD Program in Artificial Intelligence

Barcelona, 2011

Universitat Politècnica de Catalunya

Departament de Llenguatges i Sistemes Informàtics

PhD Program:
Artificial Intelligence

This thesis was completed at the:

Institut de Robòtica i Informàtica Industrial, CSIC-UPC

Cover illustration by Alejandro Gabriel Agostini

# Acknowledgements

It has been a long journey, at times rambling, others fuzzy, but finally very fruitful. In hindsight, I found a lot of people that, in one way or another, have provided me with their valuable support.

The realization of my PhD wouldn't have been possible without the support of all the persons at the Institut de Robòtica i Informàtica Industrial. The institute and its people already constitute an important part of my life. I will always be grateful of them.

I would like to thank Enric Celaya for his lessons about research, mainly for teaching me to evaluate each idea exhaustively and rigorously. I appreciate his constant availability to deeply discuss any idea and his capability of analysing in detail every document I left on his desk. I have enjoyed very much our long conversations scrutinizing the secrets of machine learning, our complex mental journeys searching for hidden treasures, sometimes non-existent. Now that the thesis is finished, I hope we can continue having these rich conversations.

I am also grateful with Carme Torras for her support and valuable advice. I have learned a lot from her. I appreciate very much her availability and her positive predisposition. She gave me the opportunity to participate in important research projects that have contributed significantly to my scientific education.

I would also like to thank Ulises Cortés for his help at the beginning of my PhD studies, to Àngela Nebot for her good advice and kindness, to Mercè Juan Badia for all her help in administrative issues, and to all the persons in the department of Llenguatges i Sistemes Informàtics that have provided me support.

I dedicate this thesis to Irene, for her love and unconditional support, for coping with my moods, for listening to my endless scientific deliriums, and, overall, for trying to understand them. I also dedicate this thesis to my family, which always supported me in an unconditional way, beyond my successes and failures. Infinite thanks to all of them.

Finally, I would like to thank Mauricio, Diego, Cristian, Saúl, Tere, José Luis, Carlos, Jordi, Josep María, Ricardo, Vanesa, and all the persons who provided me with their help when I needed it.

# Abstract

In this thesis we propose an approach for generalization in continuous domain RL that, instead of using a single function approximator, tries many different function approximators in parallel, each one defined in a different region of the domain, that compete to provide the inference at a given input. Using a competitive strategy increases the opportunities of a having a good approximator among the evaluated ones, avoiding the restriction of being attached to the performance of a single approximator. The proposed approach permits to reduce the number of experiences needed for convergence, and to produce more stable convergence profiles, with respect to using a single function approximator. The reduction in the number of experiences is obtained since, at each evaluated point, there is usually a competing approximator that generalizes better than the single global one. The more stable convergence profile is obtained since, if one approximator gets its approximation degraded, it will be supported by other approximators that usually perform better, keeping a good overall performance of the system. For the selection of the best approximator in a point, the approach associates to each approximator a relevance function which quantifies the quality of its approximation at a given input. The approximator with highest relevance in the input is selected for the inference. The relevance function is defined using a parametric estimation of the sample variance, and a parametric estimation of the sample density in the input space, which are used to quantify the accuracy and the confidence in the approximation at that input, respectively. All the parametric estimations involved in the competition, i.e. the cumulative reward, the sample variance, and the sample density in the input space, are obtained from a probability density model in the input-output space embedded in each approximator. From

this joint density model it is possible to obtain the conditional probability distribution of the cumulative reward values conditioned to a situation and an action. From this distribution we can obtain an estimation for the expected value of the cumulative reward at that situation and action, which is used for the inference of the approximator, and the variance of the cumulative reward values, which is used in the relevance function. The sample density in the input space is obtained from the number of samples collected in the approximator domain and the probability distribution in the input space obtained by marginalizing the output variable in the joint density model. The density model is represented with a Gaussian Mixture Model. The parameters of the model are updated using a new approach of online Expectation-Maximization which uses the density information to produce a forgetting based on the new information provided rather than on time, preventing the typical distortion occurring when a time-dependent forgetting is used to forget past entries in the approximation of a non-stationary function.

# Contents

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

Why humans, and animals in general, are so capable of learning to act, and to make decisions in situations never experienced? This is still an open question that tries to find an answer in different fields like neurophysiology, psychology, and, more recently, Artificial Intelligence (AI) [76]. The progresses made in the last decades in computer science motivated the researchers to model animal learning in computers, not only to better understand the underlying mechanisms of animal behaviour, but also to develop artificial agents capable of automatically learn to perform tasks.

The underlying idea of many learning to act paradigms is that an agent learns the cause-effects resulting from action executions and applies this knowledge to make decisions during a task. The learning of cause-effects is approached in different ways by the different learning paradigms, ranging from the learning of correlations between events, behaviours, and rewards, corresponding to the postulates of instrumental conditioning [79], to the learning of complex cognitive cause-effect relations as those postulated by Piaget in his theory of cognitive development [65].

Usually, in AI, the "cause" part of a cause-effect relation consists in a description of a situation in which the agent might be, and an action executable in this situation. The "effect" part, instead, codes the consequences of causes, either by a description of the situation reached after the action execution, a description of task-related evaluative feedback, or reward, obtained from the environment after the action execution, or a combined description of evaluative feedback and situation. The coding of effects involving situations are mainly used in deliberative approaches, where the agent rea-

sons about sequences of cause-effects that would permit to achieve a goal. Examples of these approaches are the symbolic reasoning strategies, like logic-based planners [44], and the planning methods based on Dynamic Programming [44]. On the other hand, the paradigms which code effects in the form of reward values are usually reactive approaches, where the agent selects at each situation the most rewarded action, without further deliberation. The most outstanding reactive approaches are those of Reinforcement Learning [40, 81], which are widely used for learning to act from experience.

Disregarding the type of cause-effects learned, there are two main aspects to tackle for the learning of cause-effect relations. On the one hand, the mechanisms through which the consequences of actions will be learned should be defined. This is the main concern of behavioural learning approaches, like those of Reinforcement Learning. On the other hand, it should be determined how this knowledge is represented so as to use it for decision making in new situations. This problem is faced using techniques of machine learning for knowledge representation [21, 31, 51], which permit to generalize the knowledge of the observed cause-effects to predict the effects of actions in unforeseen situations.

This thesis is about a new technique for generalization in Reinforcement Learning.

## 1.1 Reinforcement Learning

Reinforcement Learning (RL) is a task-directed learning paradigm for learning to act from interaction with the environment, which is suitable for problems where it is difficult to teach an agent what action to execute at each situation but results more simple to let the agent learn autonomously how to select the best action for the task at each situation. The learning takes place from experiences consisting in executing actions in certain situations, and receiving from the environment, at each action execution, a numerical feedback that barely indicates how good, or bad, an action is in the context of the task.

RL was created from the efforts of modelling the animal learning paradigms of classical and instrumental conditioning [82, 92]. Classical conditioning postulates, in a broad sense, that animals learn to correlate an event, denoted as unconditioned stimulus, with a preceding event, called conditioned stimulus. The reference experiment of

classical conditioning was carried out by Pavlov [61]. Pavlov demonstrated how a dog is able to correlate the presence of food with the ring of a bell, if, repeatedly, the ring of a bell is presented just before the food. Instrumental conditioning, on the other hand, postulates that animals are capable of learning correlations between specific behaviours in certain situations with rewards and punishments. The first in demonstrating this animal capability was Edward Thorndike [88], postulating in his "Law of Effects" that animals strongly connect situations with more rewarded behaviours, and that these behaviours are more likely to occur when these situations are presented. The correlation between behaviours, situations, and rewards or punishments was lately studied and quantified by Skinner under the name of Instrumental Conditioning [79]. Skinner measured the strength of these correlation by placing animals inside a chamber, and stimulating them in a controlled way with rewards or punishments signals as a consequence of their behaviours.

The initial efforts of modelling in computers the classical conditioning [84], and the posterior extensions of this model to be used in control applications [18, 82], gave rise to the formalization of the so successful temporal-difference (TD) learning methods [81, 83], which constitutes one of the major milestones in the formalization of RL. TD methods are basically prediction methods suitable for multi-step prediction problems where the outcome to be predicted comes at the end of a sequence of observations. TD also provides a way of dealing with the temporal credit assignment problem [82] of determining which observations in the sequence deserve more credit in producing the outcome. These features make TD very suitable to be used in RL problems, which are usually multi-step prediction problems, and where the assignment of credits to the actions that are responsible for the success or failure of a task results crucial for the learning to succeed. The definitive formalization of the TD was given by Sutton in 1988 [83]. Until that work, TD was applied in RL to predict that a low or high reinforcement is forthcoming with the presence of a state. This knowledge alone could not be used for control applications as it does not provide any information associated to actions. Thus, in order to use TD in control problems, it needed either the model of state transitions with actions, that would permit to follow the track of most rewarded states, or to explicitly code the function that maps situations to actions, stored in an *actor*, that indicates which action to execute at each state.

The use of TD for control application was lately approached and improved by the work of Watkins [92]. This work constitutes the other major milestone in RL, and was elaborated as a new model of Instrumental Conditioning inspired in the *optimality argument*. This argument claims that evolution provided animals with the ability of learning optimal behaviours, in the same way as it provided sophisticated bodies adapted to survive in their niches. Watkins stated that the optimality argument indicates that the function of instrumental conditioning is to learn to behave optimally, and proposed the use of tools from optimal control, in particular from Dynamic Programming [19], to model such an optimal behaviour. However, his aim was to model the learning mechanisms of instrumental conditioning in a simple way, using just experience of actions in situations, as it occurs in actual animal learning, avoiding the complicated calculations involved in conventional DP methods. Thus, using TD as the more elaborated model for animal learning, he proposed a new TD method called *Q-Learning* that, instead of estimating the forthcoming rewards associated to states for a given behaviour, estimates directly the highest possible forthcoming rewards associated to actions in states for the optimal behaviour. In this way, Q-Learning provided a tool for approximating incrementally the DP equations [1] , bringing together TD and Optimal Control, and establishing the basement for the definitive formalization of RL in terms of Dynamic Programming and Markov Decision Processes [81, 92].

## 1.2 Generalization in Reinforcement Learning

During the initial stages of RL the main efforts were focused on the formalization of the RL methods, rather than in the way the knowledge is represented, in order to simplify the interpretation of the experiments result. Thus, the formalization of the RL methods were carried out assuming that the number of states and actions was finite, and that the utility function, that maps states, or state-action pairs, to a prediction of forthcoming rewards, is represented in a plain tabular way. Each state, or state-action, had associated an utility value, which was updated independently of each other, making necessary that all the entries in the table should be experienced many times in order to learn the utility function.

[1] Watkins was a bit surprised that such a simple formulation for the incremental approximation of the dynamic programming equations was not proposed before for RL, to a point that he assumed as very likely that a similar formulation so far existed.

The increasing interest in applying RL methods to more complex control applications, with large or infinite number of states and actions, provoked that the tabular representation were no longer applicable, as it may require an infeasible number of experiences for the learning to take place. This limitation generated the necessity of a more abstract state representation that permits to generalize the knowledge of the value at one state, or state-action pair, to other similar states, or state-action pairs, and, hence, reducing the number of experiences required for learning.

Some of the early works of RL already approached the problem of generalization. One of the first methods for generalization in RL was proposed in the work of Barto and Sutton [18] for the control task of the pole-balancing. They based the approach on the former work BOXES by Michie and Chambers [49], which consisted in performing a partition of the state space, and calling to each part a *box*. Each box provides an aggregation of states, where all the states in a box received the same value, and where the experience at each state in a box is generalized to the other states aggregated with it. Michie and Chambers pointed out that there are many box configurations that would prevent learning to occur, and suggested the possibility of creating boxes by splitting and merging them during run-time, without an initial restriction in the boundaries. This idea was lately developed under the name of variable resolution techniques [53, 54].

The necessity of generalization in RL was also remarked in the PhD theses of Sutton [82] and Watkins [92]. Sutton distinguished between two credit assignment problems: temporal credit assignment and structural credit assignment. While temporal credit assignment deals with the identification of which of the taken actions deserve more credits in obtaining the final outcome, structural credit assignment deals with the assignments of credits to the internal decision mechanisms that are responsible for the action selection. Structural credit assignment deals with the way in which the reward function is represented, and how the credit of correct decisions in a state is distributed to other similar states. Despite he did not approach the structural credit assignment problem, he pointed out that both credit assignment problems are separable, and that the results obtained in his work can be combined with techniques that face the structural credit assignment using different strategies of knowledge representation. Watkins, on the other hand, also pointed out the necessity of generalization in RL,

and mentioned the convergence problems that may arise if the generalization of the experience in a state is transmitted to other states that would obtain a different value if they were experienced. He did not tackle theoretically the problem of generalization in RL, but he implemented a generalization method based on the CMAC function approximation [15], and analysed the convergence profile of this single implementation.

Many techniques for generalization in RL have been proposed during the last decades [30, 37, 53, 54, 59, 68, 70, 73] which, in general, consist in a combined learning strategy that, on the one hand, learn the values resulting from actions execution, and, on the other hand, generalize this knowledge using techniques borrowed from other fields of machine learning like pattern recognition [21] and concept learning [51].

Disregarding the generalization method used, there are some problems inherent to RL that the generalization methods should be able to deal with. During the learning process the value function estimation varies, not only as a consequence of the policy evaluation process, that estimates the value function for the current policy, but also as a consequence of the policy improvement mechanisms, that change the underlying policy towards the optimal one. These mechanisms make the estimation of the value function highly non-stationary. Another difficulty for generalization methods in RL is that data arrive sequentially, i.e. one at a time, and are sampled along trajectories dictated by the underlying dynamics of the environment, and influenced by the action selection strategy. This produces a very biased sampling of the points used for the estimation, which may in turn produce large distortions in the estimations and instabilities in the convergence. Thus, the generalization methods should be robust to the non-stationary target function and biased sampling problem in order for learning to take place.

In general, methods which perform local regressions [7, 30, 59] deal better with the biased sampling and non-stationarity problems as they restrict the adaptation of the function to smaller regions, at the expense of the generalization performed, establishing a trade-off between locality and generalization. Another class of methods that copes well with the mentioned problems are *fitted value iteration* methods [37, 68, 70]. Fitted value iteration methods are batch, memory-based, approaches which perform one step of function approximation over a fixed set of points, which should provide samples representative enough for the learning to take place [1] , and one step of value adaptation,

[1] In spite that providing relevant samples in advance permits to deal well with the problem of biased

always on these points, through a conventional RL method. These two processes, fitting the value function and the value function adaptation on the set of points, are carried out intertwined until some convergence criterion is fulfilled.

The aforementioned problems of generalization in RL may appear in different degrees at different regions of the domain, and at different stages of the learning process. Then, the function approximator selected should permit, at least, to cope with the approximation requirement of the most demanding region, no matter how simple the approximation requirements result in other regions of the domain. The uncertainties related to the actual shape and complexity of the value function at all the regions of the domain and during the whole learning process make the selection of the proper function approximation very complicated. This selection is done, in many cases, empirically, by evaluating the performance of different approximators until one that seems to work properly is found.

However, no matter how well we define the function approximation method to use, if we use a single function approximator, we cannot avoid the overall performance of the system to be subject to the performance of that approximator. A failure in the approximation in any region of the domain may produce a failure in the whole learning process.

## 1.3 Our Proposal

In this thesis we propose an incremental approach for generalization in continuous domain RL that, instead of using a single function approximator, tries many different function approximators in parallel, each one defined in a different region of the domain. We argue that this strategy may increase the opportunities of a having a good approximator among the evaluated ones, avoiding the restriction of being attached to the performance of a single approximator. This argument is hold under the assumption that a rich enough set of approximators is generated, and that there is a mechanism for selecting the approximator that best generalizes at each evaluated point.

sampling, finding the proper set of samples may be rather complicated in complex application, and constitutes one of the major limitations of fitted value iteration methods.

Our working hypothesis is that the competitive strategy of function approximators will permit to reduce the number of experiences needed for convergence, and to produce more stable convergence profiles, with respect to using a single global function approximator. The reduction in the number of experiences might be obtained since, at each evaluated point, there will be usually an approximator that generalizes better than a single global approximator. The more stable convergence profile might be obtained since, if one approximator gets its approximation degraded, it will be supported by other approximators that usually perform better, keeping a good overall performance of the system.

The idea of using a competitive structure is inspired in the work of Edelman [32], which claims that the robustness and flexibility of the brain is given by its competitive nature. This idea is formalized under the theory of Neuronal Group Selection, which states that the brain is organized in neuronal groups that are activated in parallel with a stimulus and that only one produces the response in a competitive, winner-takes-all, fashion. One fundamental postulate of this theory is the *degeneracy* concept, which entails that non-isomorphic groups of neurons must be isofunctional so as to match a large variety of stimuli while having similar functionality, which increases the chances of a good response.

## 1.4 Objective

The main objective of the thesis is to devise a method for generalization in Reinforcement Learning using a competitive strategy of function approximators which is able to reduce the amount of experiences needed for the learning of the reward function, and to produce more stable convergence profiles, than using a single global function approximator.

## 1.5 Contributions

The contributions of the thesis are the following:

1. A new incremental, model-free, approach for function approximation in continuous domain RL which uses tools for density estimation to achieve accurate ap-

proximations of the value function in continuous domains [6–8]. The density information permits, in addition to the function approximation, to measure the accuracy with which the approximation is done through an estimation of the variance in the approximation (a remarkable feature uniquely attributed so far to Gaussian Processes methods [33]). Even more, the density information also permits to measure the confidence with which the estimations are done in the evaluated point, which is used to regulate the exploration-exploitation strategy. The approach outperforms state of the art methods in the benchmark application of the inverted pendulum with limited torque.

2. A new approach of on-line Expectation-Maximization (EM) for the estimation of a mixture density model parameters [7, 8]. The approach uses the density information to forget past experiences in the same proportion of the new experiences provided, rather than as a function of time. This prevents useful experiences of regions weakly related to the current sample to be wrongly discarded, providing an updating mechanism which is robust to the biased sampling problem.

3. A new incremental, model-free, approach for generalization in continuous domain RL using a competitive function approximation strategy [10, 11]. Each approximator consists in a probability density model defined in a specific region of the domain that provides a continuous function approximation of the value function. The criterion to select the best approximator in a point consists in selecting the approximator that minimizes the variance of the values (accuracy), and maximizes the density of samples (confidence) in the evaluated point. New approximators are generated every time the best approximator in a point does not fulfil the approximation requirement. The proposed system produces a faster convergence rate and a more stable convergence profile with respect to a single non-parametric function approximator using probability density estimation [8], and with respect to state of the art approaches in the benchmark applications of the inverted pendulum, the mountain-car, and the cart-pole balancing.

Some of the ideas that motivated the development of the approaches proposed in this thesis were originated from our previous contributions for generalization in RL [1–5]. On the other hand, some of the concepts presented in this thesis were also

used to develop a supervised learning method that rapidly learns planning operators from few action experiences [12–14, 94], with good results in real robot applications [13]. The approach is based on the same underlying concept of a competitive strategy, where many alternatives of cause-effect explanations are evaluated in parallel, and the most successful ones are used to generate basic operators for planning. The success of a cause-effect explanation is evaluated by a probabilistic estimate that takes into account the density of samples as a measure of the confidence in the estimation. Since supervised learning is out of the scope of this thesis we refer the readers to the cited works for further details on this method.

## 1.6 Outline of the Document

The structure of the document is the following. Next chapter presents an overview of the formalization of the paradigm of Reinforcement Learning, an introduction to the problem of generalization in RL, and a description of the state of the art approaches that address this problem. Then, in chapter 3, we detail the advantages of using a competitive strategy for function approximation, and present the mechanisms of a competitive strategy called degenerate function approximation (DFA). In chapter 4 we propose a new approach for FA in RL based on probability density estimations. This approach will be lately combined with the DFA approach in the formalization of the method for FA in RL using a competitive strategy. This is the thread of chapter 5. The thesis document ends with a conclusion chapter.

# Chapter 2

# Reinforcement Learning

Reinforcement Learning (RL) [40, 81] is a learning paradigm where an agent should learn to take (optimal) decisions for the execution of a task, using only its own experience and a task-related evaluative feedback, known as immediate reward, provided by the environment after each action execution.

RL was mainly originated from the efforts of modeling the instrumental conditioning of animal learning theory [18, 82, 84, 92], which claims that animals learn associations between actions in certain situations and rewards or punishments. Analogously, many RL methods provide mechanisms that permit an artificial agent to learn associations between task-related reward values with situations, or with actions in situations, that indicate how good a situation, or an action in a situation, is with respect to a task in the long run. This information is used by the agent to make decisions, selecting at each situation the most rewarded action, or the action that would lead to the most rewarded next situation.

The formalization of RL is carried out in terms of Markov Decision Processes (MDP) [60], as models of the environment, and Dynamic Programming (DP) [19], that provides the theoretical background to find optimal behaviours using such models. However, contrarily to DP, RL does not assume the existence of a previous MDP model of the environment, which implies that the only way of knowing the consequences of actions is by an exhaustive exploration of all of them at all the situations. This is not possible in applications with many states and actions, demanding to generalize each experience to infer the consequences of actions in other situations.

This chapter briefly reviews the theoretical background of RL, first introducing, in the next section, the basic concepts of MDP and DP used for the formalization of RL, following, in section 2.2, with a description of the most known RL methods, to lately present, in section 2.3, the main aspects concerning the problem of generalization in RL, exemplifying some of the state of the art approaches.

## 2.1 Markov Decision Processes and Dynamic Programming

An MDP assumes that the environment is described using a set of discrete states, $\mathcal{S}$, a set of actions executable at those states, $\{a = \mathcal{A}(s), s \in \mathcal{S}\}$, and the one-step dynamics of the environment which is described with two probability models. The first one is the probability of transition from any initial state $s$ to any final state $s'$, after the execution of an action $a$,

$$\mathcal{P}^a_{ss'} = P[s_{t+1} = s' | s_t = s, a_t = a]. \tag{2.1}$$

The second is a model of the probability distribution of an immediate reward $r$ associated to each state transition. In the case of DP, this probability distribution is only parametrized with its expected value,

$$\mathcal{R}^a_{ss'} = E[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s']. \tag{2.2}$$

Note that the probability of state transitions (2.1) only depends on the state observed at time $t$, under which circumstance the process is said to fulfill the *Markov property*. If this property is not fulfilled, i.e. if the probability of state transition also depends on states visited before time $t$, then the problem is said to be non-Markovian, and the theory here developed is no longer applicable [1].

A MDP also assumes the existence of a *decision maker*, which selects the action to execute at each state. Normally, the decision maker is a control strategy, which in

[1] However, in many cases, it is possible to transform a non-Markovian process into a Markovian one, under a redefinition of a state which involves past observations [46].

the field of DP and RL is referenced as the *action policy*, and indicates the probability $\Pi(s, a)$ of selecting an action $a$ at state $s$. In the scope of this introduction we will assume that the action policy is deterministic and described with a function that indicates which action is selected at each state, $a = \pi(s)$.

In general, the agent goal is to maximize the reward it receives in the long run. At each particular run the agent gets rewards depending on the action policy followed. The cumulative sums of the rewards obtained from following a particular action policy $\pi$, starting at a state $s$, is calculated as,

$$\begin{aligned} R^{\pi}(s) &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \end{aligned} \tag{2.3}$$

where the discount factor $\gamma \in [0, 1)$ regulates the influence of future rewards in the sum. Equation (2.3) is defined as the *return* which values may vary at different runs depending on the probabilities of transitions (2.1) and the reward obtained after each action execution. Under these circumstances, a good prediction of the cumulative sum of rewards starting at state $s$ and then following policy $\pi$ is the expected value of the return,

$$V^{\pi}(s) = \mathrm{E}[R^{\pi}(s)]. \tag{2.4}$$

The function $V^{\pi}(s)$ is known as the *value function* for policy $\pi$. The values $V^{\pi}(s)$ can be calculated using the set of equations,

$$V^{\pi}(s) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left( \mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right), \tag{2.5}$$

where $a = \pi(s)$, and $s' = s_{t+1}$, is the state perceived at time $t + 1$. Equations (2.5) are known as the *Bellman equations* [19]. The expected return can also be associated to an action $a$ in state $s$,

$$Q^{\pi}(s,a) = \mathrm{E}[R^{\pi}(s,a)] \tag{2.6}$$
$$= \sum_{s'} \mathcal{P}^{a}_{ss'} \left( \mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right), \tag{2.7}$$

where $a$ is any of the possible actions in $s$. The function $Q^{\pi}(s,a)$ is known as the *action-value function*, or simply the *Q-function*, and is the expected return that would be obtained after executing action $a$ at state $s$ and then following the policy $\pi$. Note that the difference between the Bellman's equations in (2.5) and (2.7) is that the action referenced in (2.5) corresponds to the followed policy, $a = \pi(s)$, while in (2.7) references any possible action in $s$.

The action-value function $Q^{\pi}(s,a)$ is useful in control applications as it provides a straight way of knowing the result of executing an action, permitting to select at each state the action with highest expected return. The policy resulting from selecting the action with highest expected return at each state is known as the *greedy policy*.

We are now in position to define the *optimal action policy*, $\pi^*$, as the one that permits the agent to get the highest possible expected return. In terms of the value function, the optimal policy $\pi^*$ fulfils the inequality $V^{\pi^*}(s) \geq V^{\pi}(s)$, for all $s \in \mathcal{S}$, and for all the policies $\pi$ in the policy space. The value function $V^*(s)$ is called the *optimal value function*. The optimal value function can be computed through the maximization equations,

$$V^{\pi^*}(s) = \max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'} \left( \mathcal{R}^{a}_{ss'} + \gamma V^{\pi^*}(s') \right), \tag{2.8}$$

which are known as the *Bellman optimality equations* [19]. Combining (2.7) and (2.8) we can see that an optimal value function $V^{\pi^*}(s)$ has associated an *optimal action-value function* $Q^{\pi^*}(s,a)$ through the equations,

$$V^{\pi^*}(s) = \max_{a} Q^{\pi^*}(s,a). \tag{2.9}$$

On the other hand, from equation (2.6) we have,

$$\begin{aligned} Q^{\pi^*}(s,a) &= \mathrm{E}[R^{\pi^*}(s,a)] \\ &= \mathrm{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}] \\ &= \mathrm{E}[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2}] \\ &= \mathrm{E}[r_{t+1} + \gamma V^{\pi^*}(s_{t+1})]. \end{aligned} \tag{2.10}$$

Replacing (2.9) in (2.10) we get the expression,

$$Q^{\pi^*}(s,a) = E[r_{t+1} + \gamma \max_{a'} Q^{\pi^*}(s_{t+1}, a')] \tag{2.11}$$

$$= \sum_{s'} \mathcal{P}^a_{ss'} \left( \mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^{\pi^*}(s', a') \right). \tag{2.12}$$

Equations (2.12) are the Bellman optimality equations to calculate the optimal action-value function.

Broadly speaking, DP proposes methods to find a solution to the Bellman optimality equations. The basic mechanism of DP consists in an iteration between two processes: policy evaluation, and policy improvement. *Policy evaluation* is the process to calculate the value function, or the action-value function, for a given fixed policy $\pi$. *Policy improvement*, instead, is a process that produces an improved policy $\pi'$ from the learned value function $V^{\pi}(s)$, or action-value function $Q^{\pi}(s,a)$, of the previous policy $\pi$. One way of policy improvement is to define the improved policy $\pi'$ as the greedy policy resulting from selecting at each state $s$ the action with highest $Q^{\pi}(s,a)$. Once the new policy $\pi'$ is defined, the policy evaluation process is carried out again to determine $V^{\pi'}(s)$ or $Q^{\pi'}(s,a)$. The iteration between these two processes is called *policy iteration*, and is repeated until no further improvements of the value function is obtained, moment in which the underlying policy is the optimal, $\pi^*$.

In policy iteration, the policy evaluation process can be stopped before its convergence to carry out the policy improvement, keeping the convergence guarantees [81]. A special case of policy iteration, known as *value iteration*, is when the policy evaluation is stopped after only one sweep of updating at all states. For instance, equations

(2.12) implements the value iteration process, where the one-step policy evaluation is combined with the policy improvement in the same formula.

## 2.2 Reinforcement Learning

Reinforcement Learning pursues the same goal as DP: the learning of the optimal policy in an environment modelled as a Markov Decision Process. Even more, many RL methods and DP share the same strategy to learn the optimal policy of learning the optimal value function $V^{\pi^*}(s)$, or action-value function $Q^{\pi^*}(s, a)$, and then deriving the optimal policy from it. However, contrarily to DP, in RL it is not assumed that the model of the environment is provided in advance, and the learning of the optimal functions is performed using only the agent own experience. These RL methods are fully specified in terms of Dynamic Programming and Markov Decision Processes, and may be considered as sampled versions of the DP methods.

The lack of knowledge about the world dynamics implies that, the only way of knowing the consequences of executing actions is the exhaustive exploration of all the actions in all the states. On the other hand, the agent not only should learn the consequences of actions (policy evaluation) but also use this knowledge to improve the policy followed so far (policy improvement). Since these two processes are necessary to learn the optimal policy, there is a trade-off between exploring actions so as to learn the consequences of actions, and exploiting the knowledge so as to improve the policy. This problem of deciding when to explore and when to exploit is known as the *exploration-exploitation trade-off* [81] which is inherent to RL applications.

RL methods that use the strategies of DP to find the optimal policy can be classified in two groups: those that first approximate a MDP model of the environment and afterwards apply conventional DP techniques to find the optimal expected returns; and those that directly estimate the value functions, or action-value functions, using an incremental approximation of the Bellman equations. The former methods are denoted as *model-based RL* approaches, while the later are known as *model-free RL* approaches. In the case of model-based RL methods, as well as in DP methods, the updating of the value function is done with a *full backup*, i.e. considering all the possible transitions at the evaluated state, through (2.1) and (2.2). Model-free RL methods, instead, update

the value associated to a state, or state-action, using just the information provided by the experienced transition, i.e. with a *sample backup*. The incremental formula to do sample backup in RL is Temporal-Difference [83].

### 2.2.1 Temporal-Difference

Temporal-difference (TD) is essentially a prediction tool suitable for multi-step prediction problems, where the value to predict comes at the end of a sequence of events. The idea behind TD is that the predictions are updated using the difference between two predictions in the sequence, rather than the difference between the actual outcome and a prediction as it is normally carried out in supervised learning [1] . TD takes benefit from the information contained in the temporal correlation of the observations instead of treating them as temporally independent, and updates the prediction at each observation based on the predictions of other observations in the sequence. This permits to achieve convergence using less examples than supervised learning methods [83]. Another remarkable feature of TD is that it provides a way of dealing with the temporal credit assignment problem [82] of determining the influence of each of the executed actions in the expected return. All these features make TD a very suitable strategy to be used for the incremental approximation of the expected returns in model-free RL.

The most basic formula of TD in RL is TD(0) which is used to update incrementally the value function for a given fixed policy $\pi$ [82, 83] [2],

$$V_{i+1}^{\pi}(s) = V_i^{\pi}(s) + \eta_i \left( r(s,a) + \gamma V_i^{\pi}(s') - V_i^{\pi}(s) \right), \tag{2.13}$$

where $i$ indicates the iteration step, $r(s,a)$ is the reward obtained after executing action $a$ in $s$, $s'$ is the next state obtained after the action execution, $\gamma$ is the discount factor, and $\eta_i$ is a step-size parameter, or *learning coefficient*, that determines the proportion

[1] However, for single-step prediction problems, TD and conventional supervised learning methods result equivalent.

[2] Sutton mentioned in his work [83] that, in order to define a TD method, it is necessary to establish the desired relationship between predictions at different times in the sequence, and, using this relation, to construct an updating rule that permits to correct the predictions to the proper values. For the case of the incremental estimation of the expected return, he suggested the updating rule used finally in the TD(0) equation: $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$.

of updating of the old estimation using the correction factor,

$$\delta = r(s,a) + \gamma V_i^{\pi}(s') - V_i^{\pi}(s). \tag{2.14}$$

The correction factor $\delta$ is usually referenced as the *TD error* or *Bellman error* [26].

In order to assure convergence to the value function $V^{\pi}(s)$, the coefficient $\eta_i$ should fulfil the requirements for stochastic approximations [43],

$$\sum_{i=1}^{\infty} \eta_i = \infty, \tag{2.15}$$

and,

$$\sum_{i=1}^{\infty} \eta_i^2 < \infty. \tag{2.16}$$

Condition (2.15) guarantees that the learning coefficient keeps a large enough value to permit the correction of past estimations with new incoming samples. Condition (2.16), in turn, ensures that the learning coefficient decreases to zero in the limit to permit convergence.

Equation (2.13) performs the updating of the expected return of only one state: the one which was observed immediately before the action execution. However, TD also permits a more efficient use of the experience by updating the value function of more than one state, using what is known as *eligibility traces* [81]. Eligibility traces consists in keeping a *trace* over the most recently experienced states, associating to each state an additional parameter $e(s)$ called eligibility. The eligibility of a state indicates how "eligible" is a state to be updated with the current TD error $\delta$. It permits to spread the current experience over past states which also conduct to the current reinforcement event, though in lesser proportion than the state experienced just one step before. The eligibility of a state, hence, should indicate how long ago a state was experienced, and is usually calculated as [81],

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t, \end{cases} \tag{2.17}$$

where $\gamma$ is the discount factor, and $\lambda$ is a trace-decay parameter that regulates the "forgetting" effect of past observations. The eligibility decays with a factor $\gamma\lambda$ when the state is not experienced, and gets its value increased by one when the state is the one currently experienced. The resulting TD formula using eligibility traces is simply (for clarity, from hereafter we omit the iteration reference),

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \eta \left(r(s,a) + \gamma V^{\pi}(s') - V^{\pi}(s)\right) e(s), \tag{2.18}$$

which is known as TD($\lambda$). TD($\lambda$) differs from TD(0) in the factor $e(s)$, and in that the TD($\lambda$) should be applied to all the states instead of just the state one-step before the reinforcement event.

### 2.2.2 Actor-Critic

The TD methods presented in the previous section provide an incremental version of the policy evaluation mechanism, since it permits to find the value function for a given policy $\pi$, incrementally, using action experiences. However, in order to find the optimal value function, it is necessary to perform also policy improvement. In this case, TD(0) or TD($\lambda$) may require either the MDP model of the environment, or a complementary module called an *actor*, in order to complete the policy iteration process. In case the MDP model is provided, it would be used to evaluate which of the possible actions in a state may produce the most rewarded next state and thus update the values following the greedy strategy. On the other hand, if an actor is provided, it would be used to explicitly code the probabilities of action execution at each state which are updated towards the optimal policy with the information provided by the TD method: if TD predicts an improvement of the reward with an action execution (positive TD error) then this action gets its probability of being executed increased, while this probability is decreased in the converse case.

The structure using a TD method and an actor is known as the *Actor-Critic* architecture [18], where the role of the TD is to criticize the action selected by the actor by means of the TD error. This architecture was extensively studied in Sutton's thesis [82], and currently constitutes one of the alternatives for the implementation of a control system using RL [81].

**Algorithm 1** SARSA

Initialize $Q(s, a)$
observe current state $s$
select action $a$ in $s$ according to current action policy
**loop**
  execute $a$, get $r(s, a)$, and observe new state $s'$
  choose $a'$ in $s'$ using, for instance, the $\epsilon$-greedy strategy (policy improvement)
  generate $q(s, a) = r(s, a) + \gamma Q(s', a')$
  $Q(s, a) \Leftarrow Q(s, a) + \eta\,(q(s, a) - Q(s, a))$ (policy evaluation)
  $s \Leftarrow s'$
  $a \Leftarrow a'$
**end loop**

### 2.2.3 SARSA

SARSA [75] is an algorithm that provides an alternative to carry out policy iteration using TD but without the need of an actor. It uses the TD formula to estimate the action-value function for a given policy, instead of the value function as in (2.13),

$$Q^{\pi}(s, a) \leftarrow Q^{\pi}(s, a) + \eta \left(r(s, a) + \gamma Q^{\pi}(s', a') - Q^{\pi}(s, a)\right), \tag{2.19}$$

where $r(s, a)$ is the reward obtained after executing any action $a$ in $s$, and $a' = \pi(s')$. Given that the updating of the action-value function is done for a given fixed policy, SARSA, as well as TD(0), are said to be *on-policy* approaches.

The estimation of the action-value function simplifies the process of policy improvement. While the policy evaluation step is carried out with formula (2.19), the policy improvement step can be done by simply selecting in the resulting state $s'$, an action $a'$ using an $\epsilon$-greedy strategy, as explained in algorithm 1.

Due to the exploration-exploitation requirements, a random component must be added to explore all the actions and hence have confident estimations of the action-value function. This could be carried out, for instance, by adding a small random component $\epsilon$ to the greedy action selection, which consists in selecting a greedy action with probability $1 - \epsilon$, and a random action with probability $\epsilon$. This action selection strategy is known as $\epsilon$-greedy.

There is one important issue to take into account w.r.t. the added random component. Since the updating of the action-value function (2.19) should be consistent with the followed policy, the random component, e.g. $\epsilon$, should decrease to zero in the limit to avoid large deviations from the improved policy, and, thus, to permit convergence [81].

The complete algorithm for SARSA [1] is presented in algorithm 1.

### 2.2.4 $Q$-Learning

Perhaps the most known and used paradigm of RL is $Q$-Learning [92]. $Q$-Learning estimates directly the solution to the Bellman Optimality Equations (2.12) using a sample backup of the form,

$$Q(s,a) \leftarrow Q(s,a) + \eta \left( r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right), \tag{2.20}$$

where the sample used for the updating, $r(s,a) + \gamma \max_{a'} Q(s',a')$, resembles the sample used for the calculation of the expected optimal return in (2.11), except for the actual optimal action-value $Q^{\pi^*}$, which is replaced by its estimation $Q(s',a')$.

$Q$-Learning directly estimates the optimal action-value function, disregarding the policy followed, which makes it an *off-policy* RL approach. In SARSA, for instance, the exploration component should be small and restricted as it affects the adaptation of the $Q$-function towards the optimal. On the contrary, in $Q$-Learning, the exploration strategy is not restricted, and does not affect the updating of the $Q$-function, permitting faster value adaptations toward the optimal and a more flexible definition of the exploration strategy. However, the off-policy nature of the learning allows for abrupt changes in the policy followed and, hence, in the behaviour of the agent. This may be highly undesirable in applications where abrupt changes in the behaviour may damage the agent, as in many real robot applications, in which cases SARSA, or the actor-critic architecture, would be preferred.

[1] The name SARSA was extracted from the notation of the elements in the sequence $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ which is used in the TD formula for the action-value estimation.

**Algorithm 2** $Q$-Learning

Initialize $Q(s, a)$
observe current state $s$
**loop**
  select action $a$ in $s$ according to exploration-exploitation strategy
  execute $a$, get $r(s, a)$, and observe new state $s'$
  estimate maximum $Qmax = \max_{a'} Q(s', a')$
  generate $q(s, a) = r(s, a) + \gamma Qmax$
  $Q(s, a) \Leftarrow Q(s, a) + \eta\,(q(s, a) - Q(s, a))$
  $s \Leftarrow s'$
**end loop**

Note that, in $Q$-Learning, the policy improvement process is implicitly carried out in the updating formula (2.20) as in the value iteration process [1] hence, at least theoretically, there is no need to explicitly carry out exploitation in the exploration-exploitation strategy. Nevertheless, in some applications, exploitation may be mandatory to guide the agent through the regions which are relevant for the learning to take place, as in control applications where the goal states are only reachable after following the control strategy learned so far.

The complete algorithm for $Q$-Learning is depicted in algorithm 2.

### 2.2.5 Policy Search

Policy search methods [26, 42, 57] directly search for policies that maximize the long-term returns in the policy space, and may not even involve an estimation of the value function [2] . They carry out an explicit representation of the policy using a set of parameters that map into the policy space. This representation permits controlling the way in which the policy is updated, preventing discontinuous changes in the policy that may occur when it is obtained from the action-value function by just following the greedy strategy.

Policy search methods are suitable for problems in which the changes in the behaviour of the agent needs to be smooth and carefully controlled, like in real-platform applications with high risk of damage, which cannot be guaranteed by simply following

[1] In fact, $Q$-Learning constitutes a sample version of the value iteration process.
[2] These methods are also known as *Policy-based* methods, while those that use a value function estimation to derive a policy, e.g. $Q$-Learning, are referenced as *Value-based* methods.

the greedy policy. They are also preferred in those applications where the approximation of the policy may result simpler than the approximation of the value function, provided some initial knowledge that permits assuming this. Policy search methods are also attractive since they permit a continuous variation of the policy, a condition that allows granting convergence when combined with some FA methods [62].

One known instantiation of policy search method is the actor-critic structure that combines a policy approximation with a value function approximation, this last used to search for the optimal parameters of the policy approximation. Since the actor-critic structure performs both, a policy approximation and value function approximation, it is considered as a bridge between Policy-based and Value-based methods.

#### 2.2.5.1 Policy Gradient

Policy Gradient methods [85] belong to the Policy Search class of algorithms, where the policy $\pi_\theta$ is explicitly represented with a set of parameters $\theta$. In this case, as well as in many other RL methods, the method consists in the policy evaluation and policy improvement steps. However, in the policy evaluation step, instead of approximating the expected reward as a function of the state and action, like in most of the RL algorithms, the method approximates the expected reward with respect to the policy parameters $\theta$,

$$J(\theta) = E\left[\sum_{t=1}^{\infty} \gamma^{(t-1)} r_t \,|\pi_\theta\right]. \tag{2.21}$$

Having the expected reward as a function of the policy parameters usually allows for a more simple and compact representation than the expected reward depending on the state and action variables. In the policy improvement step, the parameters of the policy are updated in the direction of the gradient ascent of $J(\theta)$,

$$\Delta\theta = \alpha \nabla_\theta J(\theta), \tag{2.22}$$

where $\alpha$ is the learning rate.

The main concern of Policy Gradient methods is to estimate the gradient $\nabla_\theta J(\theta)$. In this regard, one of the most efficient approaches is the Natural gradient method [41]

that follows the steepest direction towards a local maximum of $J(\theta)$. Therefore, the Natural gradient moves the policy towards the best policy given the expected reward learned so far, and not just towards a better policy, as obtained when the standard gradient method is used.

The Natural policy gradient formulation was used to create the Natural Actor-Critic method (NAC) [63, 64] that constitutes one of the most outstanding state of the art policy-based RL approaches. In short, the NAC updates the policy parameters using the Natural policy gradient, while the critic obtains the gradient and the parameters associated to the expected value approximation using an adaptation of the $LSTD(\lambda)$ algorithm [25].

## 2.3 Generalization in RL

The basic formulation of RL assumes that the value function, or the action-value function, is explicitly represented by storing values for each state, or state-action, in a plain tabular representation. This implies that all the possible actions in all the states must be experienced many times for learning to take place. This is impractical in applications where the number of states and actions is too large, and impossible when it is infinite. Unfortunately, many of the interesting applications of RL are of this kind, demanding a different representation that permits to use each experience to update the values of more than one state, or state-action, reducing the number of experiences needed for learning. This is the problem of *generalization* in RL.

The generalization problem has been widely treated in the field of machine learning, mainly for concept learning [51] and pattern classification [21, 31]. These techniques use a finite set of input-output samples of an unknown function, commonly referenced as the *target function*, to perform an approximation of this function using a method of *function approximation* (FA) [27]. The approximation permits to infer values of the target function in points of the domain with unknown output. In RL, the target function is the value function, or the action-value function, and the input-output training points are the samples used in the sample backups. For instance, in $Q$-Learning the samples

are of the form,

$$< (s,a), r(s,a) + \gamma \max_{a'} Q(s',a') > \tag{2.23}$$

where $(s,a)$ is the input vector, and $r(s,a) + \gamma \max_{a'} Q(s',a')$ is the output. Note that, the values $Q(s',a')$ are estimated from the FA made so far. We will use the notation $\hat{Q}(s',a') \approx Q(s',a')$ for the values inferred by the FA method in $(s',a')$.

The idea of FA in RL, and in machine learning in general, is to represent the target function using a set of parameters which permits an abstract representation of the points in the domain, where a change in any of these parameters affects the inference at many inputs. The parameters could be of different types depending on the function representation used for the approximation like, for instance, the weights of a Neural Network [16], or the decision values in a decision-tree representation [67]. Depending on whether the number of parameters is fixed or variable during the learning process, the methods are classified into *parametric* or *non-parametric*. Parametric methods [21] use a fixed set of parameters which configures a fixed mapping from the parameter space into the function space. Hence, a parametric FA method can represent a limited number of functions which depends on the nature and number of the parameters used. Contrarily, non-parametric methods are able to change the number of parameters in the representation, thus being more flexible than parametric ones as they may permit to approximate a function at any precision disregarding its shape and complexity.

### 2.3.1 Problems of Function Approximation in RL

Any method for FA in machine learning can be used in RL, but not all of them will permit a good generalization. Due to the recursive nature of the expected returns estimation, e.g. (2.13), (2.19), and (2.20), and to the variations of the exploration-exploitation strategy caused, for instance, by the changes in the values associated to the actions, the function to be approximated is largely *non-stationary*. On the other hand, the samples used by the FA method are obtained along trajectories dictated by the dynamics of the environment, and also by the exploration-exploitation strategy. This makes the sampling to be *highly biased* and non-stationary, inducing regions that are more frequently sampled to have better estimations than regions sparsely explored.

This is an inconvenience since, for a good control strategy, the reward function should be well represented even in those regions sparsely visited, or visited long time ago, in which a wrong decision may imply a failure in the control task.

The biased sampling and the non-stationarity problems compromise the convergence of the function approximation methods. Therefore, FA methods in RL should adopt special strategies to mitigate the undesired effects of these problems. For instance, works involving the class of methods called *fitted value iteration* [33, 37, 70] avoid the problem of biased sampling by assuming that a fixed set of relevant [1] states, or state-actions, is provided in advance. The approximation of the value function is performed by *fitting* a function to those points in the set, focusing the error analysis only on those states. Some other methods [48, 81, 91] deal with the biased sampling problem through weighting the approximation error at each state, or state-action, by its probability of occurrence according to a fixed followed policy. In [47], in turn, the convergence proof of a linear function approximation in RL is carried out by assuming ergodicity, i.e., by assuming that all the points in the domain are visited infinite times and, hence, neglecting the biased sampling problem. Another way of dealing with the biased sampling problem is the one proposed in [62] for policy-iteration algorithms. Conceptually, they demonstrated that, if the agent behaviour changes smoothly with the changes in the value function estimates, the visitation frequency of the states, or state-action pairs, varies smoothly, and so it varies the new value function to be estimated. These smooth variations in the policy and in the value function permits granting convergence.

In general, local function approximation methods, as state-aggregation techniques (section 2.3.2) or radial basis functions (section 2.3.3), deal better with the problems of non-stationarity and biased sampling. They limit the updating of the value function to the surrounding of the sample, preventing large distortion of the approximation in regions far from the experienced point. However, the locality of the updating also implies a restriction in the generalization done, establishing a trade-off between locality and generalization.

Another characteristic inherent to FA in RL is that both, the function approximation and the updating of the value function using backups, take place intertwined, and

[1] The set of points should be relevant in the sense that they have to permit to learn something about the underlying MDP.

their performances are mutually dependent. Most of the convergence analysis of these two intertwined processes is done in terms of *nonexpansive* functions [1] [26, 37]. Broadly speaking, the recursive application of a function approximator with a DP/RL operator may converge to an estimation of the optimal value function, within a bounded error, if the function approximator is nonexpansive. A nonexpansive function approximator is one that does not exaggerate the difference between the approximation carried out in an input for two different target functions. This suggests that a nonexpansive FA may prevent exaggerated changes in the approximations when the approximated function is non-stationary. In particular, all the function approximators that are *averagers*, i.e. that the inferred values are produced by a weighted average of values of the target function, are nonexpansive. Examples of averagers are local weighted averaging, k-nearest-neighbour, linear interpolation, and state-aggregation techniques. It is important to remark that the averagers interpolate from the observed values, providing inferences in between the ranges of the observations, rather than extrapolate, preventing large divergence in the approximation, and, hence, favouring convergence.

Another important issue to take into account when designing a FA method for RL is that, during the process of FA and RL, the intermediate value functions at the different stages usually have different structures, and could be even more complex than the optimal one [24]. Since the complexity of the intermediate value functions, and of the final optimal value function, is usually unknown in advance, they cannot be expected to fit into a predefined parametric model. In general, non-parametric methods deal better with these complexity variations than parametric ones.

In this thesis we will focus on applications with continuous domain. In the following sections we present some of the most used strategies for FA in continuous domains, describing remarkable state of the art approaches.

### 2.3.2 State-Aggregation Techniques

The most simple approach for function approximation in RL problems with continuous domains is to represent the state space with a finite partition, and then treat the problem as in the tabular case, considering each part as a more abstract state representation. State-aggregation techniques use a partition of the state space where each

[1] A function that fulfills the inequality $\|f(a) - f(b)\| \leq \|a - b\|$ is said to be nonexpansive.

part is an aggregation of states, hence the name, which assigns to all the states in the part the same value. Every time a state in a part is experienced, the sample is used to update all the states in the part. The key idea of these techniques is to use the coarsest parts possible that permit to learn the target value function.

Some remarkable works of state-aggregation techniques are Variable Resolution Dynamic Programming (VRDP) [53, 55, 69], and U-Tree [46]. The original VRDP [53] is a model-based RL approach which learns the transition probabilities between parts in the partition and then applies conventional DP considering each part as a state. The resolution is increased on demand of a better approximation of the value function by splitting those parts that belong to trajectories with higher accumulated rewards. An extension of VRDP techniques to model-free problems is proposed in [69], where the resolution is increased in those regions of the state-space where the variation of the estimated value function is high. In turn, U-Tree is also a model-based approach of RL which uses a decision-tree representation of a state-action space with discrete variables. Each branch in the tree is a compact representation of a sequence of states and actions which is coded considering only the relevant values of the variables [1] at each observed state, or executed action in the sequence. Each variable value conforms a node in the tree. At each leaf, the tree stores experienced instances of the corresponding sequence, the values of expected returns associated to each possible action at the final state, and an estimation of the MDP model of state transitions and rewards. U-tree is a clear example of a RL method that transforms a non-Markovian problem into Markovian using memory. This is possible since the coded sequence can be considered as a new state definition which is indeed Markovian.

One of the most important problems to face when using state-aggregation techniques is the over refinement of the domain. The over refinement may be caused by different reasons. For instance, the changes in the value functions estimation during the learning process may produce a high granularity in some regions, though may result useful for a proper approximation in one stage of the learning, may be useless for later stages. Another source of over refinement is stochasticity. In stochastic problems the splitting of a state may be carried out by a large approximation error generated by an outlier generated randomly with low probability. The over refinement problem may

[1] The relevancy of a variable value is with respect to its correlation with a the obtained rewards.

produce that the amount of experiences required to reach an acceptable approximation, even though reduced, remains intractable. One simple way of facing the proliferation of parts is to generate parts in a general to specific fashion, depending not only on the approximation error, but also on the number of experiences collected in the part involved in the inference. If the number of experiences is low then more experiences should be collected before generating new parts to allow for confident estimations of the approximation error [69].

### 2.3.3 Linear Combination of Basis Functions

Another technique of FA in continuous domain RL is to use a linear model, where the output is a linear combination of basis functions (BFs) [26], also referenced as *features*. For instance, in terms of an action-value function approximation $\hat{Q}(s,a)$, the linear model of BFs has the form,

$$\hat{Q}(s,a) = \sum_{i=1}^{K} \theta_i \, F_i(s,a,\xi_i), \tag{2.24}$$

where $F_i$, $i = 1, \ldots, K$, are the BFs, and $\theta_i$ are the parameters that weight the influence of each BF in the inference. The parameters $\xi_i$ specify the shape and size of the $i$th BF, and can be fixed in advance or adjusted during learning. The BFs map the original state-action space, into a more abstract feature space that provides a compact representation of the action-value function, now fully described in terms of the parameters $\xi_i$ and $\theta_i$.

There are many different strategies of FA in RL using a linear combination of basis functions [26]. The most simple strategy is to predefine the BFs, fixing in advance the number of BFs and their parameters $\xi_i$, and then apply a linear method to find the parameters $\theta_i$. This strategy has been widely used in RL thanks to the simplicity and efficiency of linear methods, and to their known convergence properties when using, for instance, the method of gradient descent [47]. In fact, almost all useful convergence results for FA in RL are for linear, or simpler, FA methods. An advantage of the linear methods is that there is only one minimum for the approximation error, which corresponds to the optimal values of the parameters. Any method that guarantees a convergence to a minimum of the error, or a point close to that minimum, guarantees the

convergence to the optimal set of parameters, or close to the optimal set. However, since the BFs are fixed in linear methods, the repertory of functions that can be approximated is also limited. Given the lack of knowledge about the shape of the unknown target value function, this limitation may require a lot of effort in the definition of the BFs to use. The alternative is to let the BFs be found automatically, while learning proceeds [26].

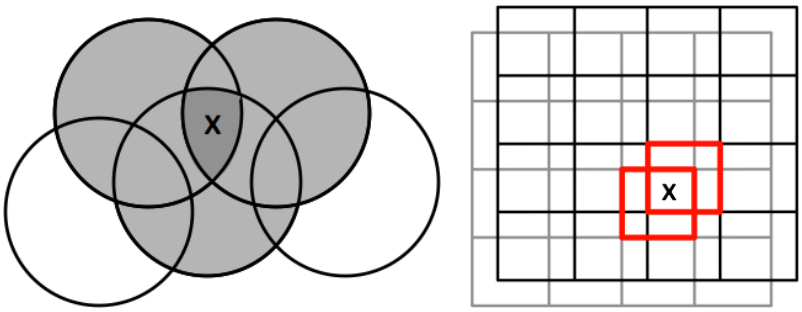
The automatic learning of the BFs may be carried out by either predefining the number and nature of the BFs, and letting the parameters $\xi_i$ and $\theta_i$ be adjusted during learning [48], or by varying the number and nature of the BFs and adjusting their parameters $\xi_i$ on demand for a better approximation in a non-parametric approach [20, 38, 59, 93]. This last strategy is the most flexible, permitting to approximate arbitrary functions. However, the convergence of these non-parametric approaches are much more difficult to analyse than linear methods, and, in many cases, cannot be guaranteed [26].

For the sake of illustration, we now summarize two BFs which are frequently used in the literature: binary BFs, and radial BFs. The explanation of the following FA methods will be carried out using the general notation $\mathbf{x}$ to reference the input vector, which in RL may represent either a state, or a state-action.

### Binary Basis Functions

One simple example of basis functions are the *binary basis functions*, where the output of the BF is 1 when the analyzed point, say $\mathbf{x}$, is included in its domain, and 0 otherwise. Figure 2.1 illustrates two examples of binary BFs. Figure 2.1(a) shows an example of a *coarse coding* representation [81], where the BF domains consist in circles that together configure a covering of the input space. Figure 2.1(b), in turn, illustrates a simple example of a *multi-partition scheme* representation [15, 23, 81, 93]. The figure depicts an scheme with two partitions, where each part defines a domain of a BF. Note that, state-aggregation techniques, described in section 2.3.2, can be considered as a special case of a multi-partition scheme, where the number of partitions is one, and the parameter $\theta_i$ for the $i$th BF is the value function estimation at the corresponding part.

When using binary BFs, the final approximation achieved is more dependent on the number of elements involved in the inference rather than on their sizes: the higher the




(a) Coarse Coding (b) Multi-partition Scheme

Figure 2.1: Examples of binary BFs in a 2D domain. Figure 2.1(a) presents a coarse coding representation, while figure 2.1(b) illustrates a multi-partition scheme. In both cases the BFs used for the inference, i.e. those containing the point $x$, are marked.

number of elements the better the approximation. This permits to use BFs that cover wide regions of the input space, favouring generalization.

### Radial Basis Functions

Radial basis functions (RBF), provide a continuous mapping from the original state space, or state-action space, into a feature space with values usually in the range $[0, 1]$.

RBF produces values depending on the radius (hence the name) of the input $\mathbf{x}$ with respect to the center of the BF. One common example of RBF is the Gaussian shaped BF,

$$F_i(\mathbf{x}, \xi_i) = \exp\left(-\frac{1}{2}\left[\mathbf{x} - \mathbf{c}_i\right]^T \Sigma_i^{-1} \left[\mathbf{x} - \mathbf{c}_i\right]\right), \tag{2.25}$$

where the parameters $\xi_i = (\mathbf{c}_i, \Sigma_i)$ consist in the vector indicating the center of the BF, $\mathbf{c}_i$, and the covariance matrix $\Sigma_i$ that determines the shape and size of the BF. The center $\mathbf{c}_i$ plays the role of a prototypical point with an influence that is maximal at $\mathbf{c}_i$ and that decays smoothly according to $\Sigma_i$ when moving away from the center. This smooth variation in the inference of a RBF permits to carry out a smooth, differentiable, approximation of the value function, providing a more suitable representation for FA in continuous domains than binary BF.

### 2.3.4 Neural Networks

Neural Networks (NN) [16] have been extensively used for FA in supervised learning [51, 76] due to their capabilities of approximating non-linear functions, the well known methods for its parameters estimation, and their capabilities of dealing with high dimensional domain applications. However, when applied for FA in RL, and disregarding some successful applications [87], NNs may perform poorly [24, 89]. One problem of NNs is that the global nature of its approximation may produce large distortions in regions far from the experienced sample, which is worsen if the sampling is biased. Another drawback is that NNs need many iterations to get a good approximation, involving intensive computations, which may imply an impractical amount of time for learning to take place when samples arrive one by one.

Despite the mentioned disadvantages, there are some methods that try to exploit the benefits of NNs for FA in RL. For instance, the global nature of their approximation may result very useful for generalization, since the updating of large areas may accelerate convergence significantly. One of the state of the art approaches of NNs in RL is Neural Fitted $Q$ Iteration (NFQ) [70]. NFQ basically combines a multi-layer perceptron approximation with $Q$-Learning. This method faces the problem of distortions produced by a global approximation by first storing a set of representative samples uniformly distributed in the state-action space, and then fitting a function to them in a batch mode. This strategy permits a balanced updating of the approximation at different regions of the state-action space. Since the updating is carried out using a set of representative samples, the NFQ method is cast into the *fitted value iteration* class of algorithms [37]. Another important feature of NFQ is that the batch nature of the learning permits to apply a backpropagation approach called Rprop [72] which has better convergence properties than conventional backpropagation methods.

### 2.3.5 Gaussian Processes

Gaussian Processes (GP) [68] is a non-parametric approach for continuous FA that combines the flexibility of non-parametric methods with Bayesian inference in a tractable way. In general terms, a GP uses a *prior* probability distribution over functions, defined by the user, and a set of observations $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ , considered as the *evidence*, to

calculate a posterior probability distribution over functions from which the inference is derived. The specification of a prior is important because it specifies the properties of the functions to be considered. The data set is described with a matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ that represents the observed inputs $\mathbf{x}_i$, $i = 1, \ldots, n$, and a vector $\mathbf{y} = [y_1, \ldots, y_n]$ that represents the corresponding outputs of the target function.

The probabilistic treatment of the data provides a very flexible representational tool, capable of approximating value functions in a wide spectrum of RL applications. One point to remark is that, in addition to the function estimation in a given point, GP also provides the uncertainty in this estimation at each evaluated point [1]. Recently, many works have exploited the advantages of GP for FA in RL [28, 33, 73, 74]. Some of them use a model-free approach [28, 33], combining, for example, SARSA with GP [33]. Some others, instead, use a model-based approach [28, 73], generating a separate model for the environment dynamics, and for the value function.

In the sequel we present the outlines of a GP approximation. For a deep insight about GP please refer to [68]. A GP is completely specified by a mean function $m(\mathbf{x})$, and a covariance function, $k(\mathbf{x}, \mathbf{x}')$, also called kernel,

$$m(\mathbf{x}) = \mathrm{E}\left[f(\mathbf{x})\right], \tag{2.26}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathrm{E}\left[\left(f(\mathbf{x}) - m(\mathbf{x})\right)\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right], \tag{2.27}$$

where $f(\mathbf{x})$ indicates not a single value of a function, but a sequence of points corresponding to one possible function, being each element of the sequence a point of the function. Because of this, $f$ is not referred as a random variable, which would be instantiated in single values, but as a real *process*, which is instantiated with sequences of values belonging to a function. In this view, the real process $f$ is generated from a GP, $\mathcal{GP}$, as,

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{2.28}$$

[1] Although this uncertainty is said to be another remarkable contribution of GP, as far as we know, it has not been used for any purpose. Nevertheless, some works suggested that this information could be used, for instance, to enrich the repertory of exploration-exploitation strategies in RL [33].

The initialization of the mean and covariance functions defines the prior probability distribution over functions. The mean function $m(\mathbf{x})$ is set, in most of the cases, to be 0 everywhere. Instead, the covariance function is usually defined as the squared exponential (sometimes referenced also as radial basis function),

$$k_\theta(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2}\left(\mathbf{x} - \mathbf{x}'\right)^T \Lambda^{-1} \left(\mathbf{x} - \mathbf{x}'\right)\right), \tag{2.29}$$

where $\sigma^2$ reflects the variability of the latent function $f$, and $\Lambda = diag(l_1, \ldots, l_d)$ is the length-scale which determines the decay of the covariance with the distance between the inputs $\mathbf{x}$ and $\mathbf{x}'$. The parameters $\theta$ of the covariance function, which in the exemplified kernel are $\theta = \left(\sigma^2, \Lambda\right)$, are denoted as *hyper-parameters*, and are determined so as to maximize the log-likelihood of the data $\log p(y|\mathbf{X}, \theta)$ using, for instance, evidence maximization [45]. GP does not require an explicit parametrization of the function. Instead the set of hyper-parameters $\theta$ implicitly defines the function representation. Since the hyper-parameters $\theta$ are tuned to fit a set of observations, when applied for FA in RL, GP approximation are said to belong to the *fitted value iteration* class of algorithms [37].

The setting to 0 of the values of the mean function implies that the FA capabilities of the GP rely completely on the covariance function. In fact, learning in GP is the problem of finding suitable properties of the covariance function [68] [1].

Once determined the hyper-parameters $\theta$, it is possible to calculate the distribution of values of functions at a given input $\mathbf{x}^*$, and from this distribution derive the estimation of the function at $\mathbf{x}^*$, $f^*$, and the uncertainty in this estimation. To see how this is carried out note that, a GP is defined as a collection of random variables for which any finite number of them configured a joint Gaussian distribution [68]. Each random variable in the joint distribution is either related to an input point $\mathbf{x}$, either observed or evaluated, or to an output value $y$, associated to each input. From the joint Gaussian distribution, it is possible to derive the distribution of output values $f^*$

[1] Despite one of the benefits attributed to GP is that it is not attached to a certain class of functions [68], this attachment exists implicitly in the definition of the prior, which indeed defines the nature and richness of the functions that can be approximated.

for a given input $\mathbf{x}^*$ through the conditional probability,

$$(f^*|\mathbf{x}^*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}(\bar{f}^*, \sigma^2_{f^*}), \tag{2.30}$$

where the rest of the possible input-output, i.e. the ones not belonging to the observed samples $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ or to the evaluated point $\mathbf{x}^*$, are marginalized. The mean value $\bar{f}^*$ is used for the inference of the function at $\mathbf{x}^*$, and the variance $\sigma^2_{f^*}$ as a measure of the uncertainty in the estimation.

Finally, to illustrate how the inference is calculated, let assume that some noisy observations $y = f(x) + \epsilon$ are provided, where $\epsilon$ representing an iid Gaussian noise with variance $\sigma_n^2$ [1] . In this case, the mean and variance in (2.30) are calculated as,

$$\bar{f}^* = k(\mathbf{x}^*, \mathbf{X})(K + \sigma_n^2 I)^{-1}\mathbf{y}, \tag{2.31}$$

and,

$$\sigma^2_{f^*} = k(\mathbf{x}, \mathbf{x}^*) - k(\mathbf{x}^*, \mathbf{X})(K + \sigma_n^2 I)^{-1}k(\mathbf{X}, \mathbf{x}^*), \tag{2.32}$$

respectively, where $K$ is the kernel matrix with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

## 2.4 $Q$-Learning with Function Approximation

$Q$-Learning (section 2.2.4) is usually preferred over other RL methods, like SARSA or Actor-Critic, since it combines fast convergence properties with simplicity. This section presents the general mechanisms involved in $Q$-Learning with FA. In particular we will focus on the online version of FA in $Q$-Learning for continuous domain applications, used along this thesis, where the updating of the FA is carried out stepwise, using only the current experience. For clarity, we will refer to such approach as online *memory-free* FA in $Q$-Learning.

[1] One point to remark concerning the random component in the observations is that the shape of random component needs to be known in advance for the calculation of the conditional mean (2.31) and variance (2.32). This restriction may imply serious difficulties to the GP methods when dealing with observations with unknown stochastic components.

$Q$-Learning with FA in continuous domains has been approached by many works [26, 37, 47, 70, 81, 86]. Most of the recently proposed approaches belong to the fitted value iteration algorithms [37, 70]. Fitted value iteration algorithms, base their convergence proofs on the concept of nonexpansive functions, and the existence of a set of points that are *representative enough* so as to provide information about the underlying MDP [26, 37]. Having a representative enough set of points may be rather complicated in many interesting practical applications where the model of the environment is not available and samples can only be observed while interacting with the environment.

To obtain a representative set of samples, the simplest approach is to get samples from the whole state space by chaining a number of random actions [34]. However, when the problem grows in complexity, the probability of executing a random sequence that drives the system to the interesting regions of the workspace may be too low to be achieved in practical time. In such cases it is necessary to exploit the knowledge already obtained with previous interactions [34], which conducts to the problem of biased sampling. In [70], the biased sampling problem is avoided by assuring that all datapoints are used for update the same number of times. This is made possible by remembering a dense enough set of transitions and performing full updates in batch mode. In fact this is a common trait of all fitted value iteration algorithms. From a computational point of view, this approach is very computationally intensive, since all datapoints are used a large number of times until convergence is reached.

The alternative is to update the function representation by just using the currently experienced transition, without memorizing samples, as carried out in the original algorithm of $Q$-Learning (see algorithm 2). This relaxes the necessity of selecting and storing a relevant set of samples, and the computational costs of updating all their values at each iteration.

However, in the memory-free case, the biased sampling problem becomes an important issue to deal with, and special attention must be paid to avoid large distortions in the approximation in regions far from the experienced sample. Many approaches restrict the undesired effects of biased sampling by focusing the approximation resources only at those points that are visited when following a fixed policy [47, 86]. But this restriction is impractical, since, on the one hand, the agent cannot exploit the knowledge acquired so far to improve its behaviour, and, on the other hand, following a particular

**Algorithm 3** $Q$-Learning with an Online Memory-free Function Approximation

Initialize representation of the Q-Function, $\hat{Q}(s,a)$
observe current state $s$
**loop**
  select action $a$ according to exploration-exploitation strategy
  execute $a$, get $r(s,a)$, and observe new state $s'$
  estimate maximum $Qmax = \max_{a'} \hat{Q}(s',a')$
  generate $q(s,a) = r(s,a) + \gamma Qmax$
  update $\hat{Q}(s,a)$ using sample $< (s,a), q(s,a) >$
  $s \Leftarrow s'$
**end loop**

policy may prevent the agent to visit those regions that are indeed relevant for the learning of the optimal $Q$-function.

Under these circumstances, most applications of online memory-free $Q$-Learning follow the action policy dictated by the exploration-exploitation strategy, at the expense that convergence is no longer guaranteed [50, 90]. In this case convergence may occur for each particular case depending, among other things, on how well the FA method deals with the biased sampling problem. The method for online memory-free FA in $Q$-Learning is a natural extension of the original $Q$-Learning algorithm, and is depicted in algorithm 3.

# Chapter 3

# Degenerate Function Approximation

## 3.1 Introduction

One of the main difficulties encountered when selecting a function approximation (FA) method for generalization in RL is that the target value function is unknown, and that it varies in shape and complexity along the learning process [37]. The value function may be rather smooth at some regions, while presenting large variations at others. In most of the cases these variations are unpredictable, demanding the FA method to be flexible enough to cope with the variable approximation requirements at all the regions of the domain, and during the whole learning process.

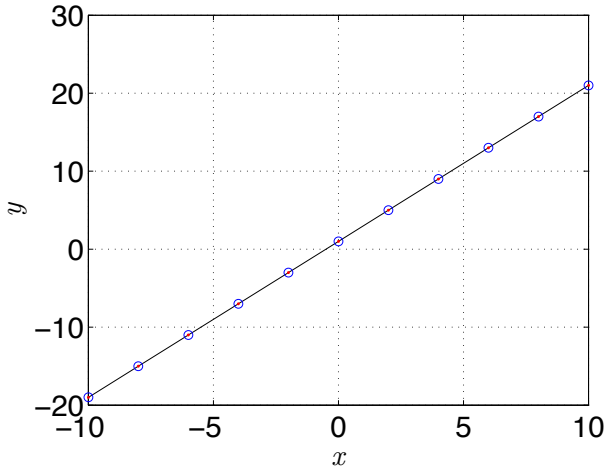
The flexibility of the FA method should permit to cope with the approximation requirements at most demanding regions, no matter how simple the target function may be at others. This may produce that, if the target function presents many regions with simple shapes, the FA method uses, for the approximation at those regions, a function much more complex than actually needed. In this case, the amount of experiences required to achieve a desired approximation would be much larger than the amount of experiences needed to achieve the same approximation but with a simpler, though appropriate, function. This is so since the larger the complexity of the function used, the larger the number of experiences required to balance all its parameters. This fact is illustrated in figure 3.1, where a straight line is approximated using two different

polynomials, one of degree 1, and another, more complex, of degree 2 [1]. To illustrate the amount of experiences needed for convergence, we calculate, at each iteration of the learning process, the mean absolute error (MAE) over 1000 test inputs randomly selected, and plot, in figure 3.1(b), the evolution of the MAE for both cases. It is easy to see that the approximation carried out with the simpler polynomial (red line), convergences much faster than the one performed with the more complex polynomial (blue line).

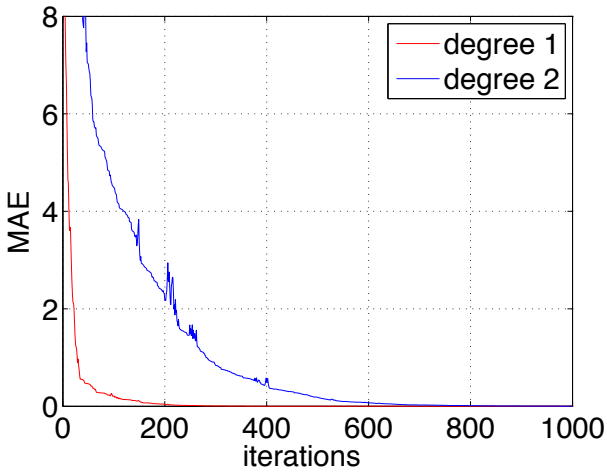
The previous result suggests that, if we define a partition of the domain where each part delimits a region where the target function has a simple shape, and define each of these parts as the domain of a simple, and independent, function approximator, then the number of experiences required to achieve a certain approximation with the simple FAs would be less than the number of experiences required by a more complex FA defined in the entire domain. For instance, figure 3.2(a) presents a portion of a sinus-like target function (black line), which is approximated, on the one hand, with a degree 4 polynomial (blue circles), as the least degree polynomial that permits an acceptable approximation, and, on the other hand, with four FAs which use a polynomial of degree 1 (red dots), and which are defined in smaller regions. The convergence profiles of the approximation carried out with the polynomial of degree 4 (blue), and the approximation carried out by the four simpler FAs (red), are shown in figure 3.2(b). As shown in the figure, the simpler FAs achieve the same approximation error in a shorter time than the complex one.

Note that, the improvement in the convergence when using linear FAs occurs despite the fact that the four FAs involve a total of 8 parameters. This is so since, in the linear FAs case, the parameters of each FA are adjusted independently, where each linear FA needs to balance only two parameters, against the 5 parameters involved in the quartic FA. In addition, it is important to consider that each linear FA has achieved such approximation with one fourth of the total number of samples, while the quartic polynomial was tuned using all of them, which also shows the efficiency of using many simple FAs instead of a unique, complex one.

[1] The method used for the incremental updating of the parameters is the gradient descent method, using samples drawn randomly from an uniform probability distribution.

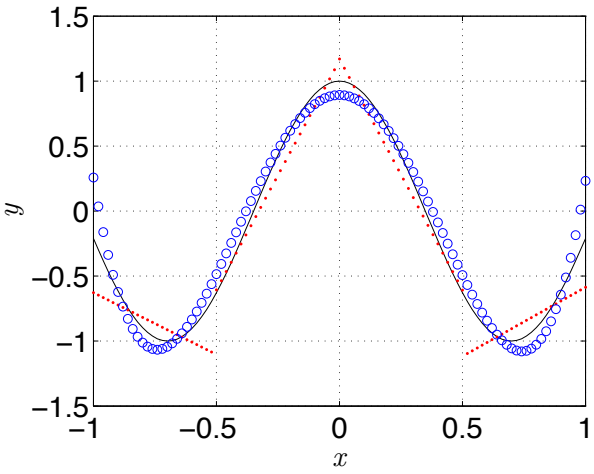



(a) Target function approximations.

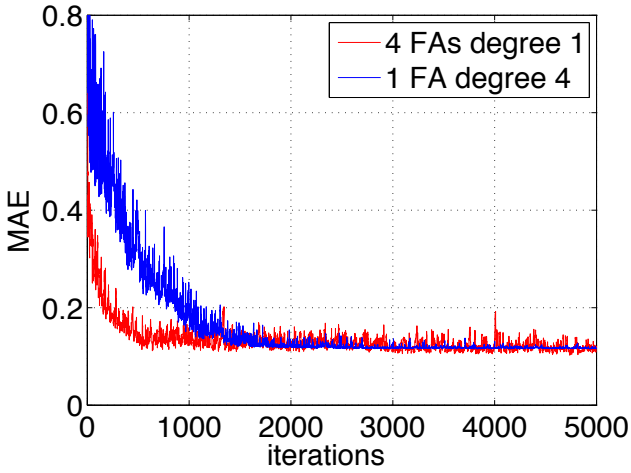



(b) Convergence profiles.

Figure 3.1: Comparison of the learning rate between two FA with different complexities. Figure 3.1(a) shows the target function to be approximated and the approximation reached, at sparse test samples, for the polynomial of degree one (red dots), and for the polynomial of degree 2 (blue circles), showing that both reach the same accuracy. Figure 3.1(b) shows the different convergence profiles of the approximation carried out with the polynomial of degree one (red line), and with the polynomial of degree two (blue line).




(a) Target function approximations.




(b) Convergence profiles.

Figure 3.2: Comparison between a FA consisting in a polynomial of degree 4 versus four linear FAs. Figure 3.2(a) shows the approximations achieved, where the black line is the target function, the blue circles show the approximation of the degree 4 polynomial, and the red dots correspond to the approximations of the simpler, degree 1, polynomials. Figure 3.2(b) contrasts the convergence profile of the complex FA, in blue, with the approximation performed by the simpler FAs, in red.

### 3.1.1 A Competitive Strategy for Function Approximation: General Concept

Even though many simpler FAs may perform better than a unique complex one, there is still the need of finding those regions in which this simpler FAs will perform well. For instance, in figure 3.2(a), the shape of the target function in the central regions in which the linear FAs were defined, can be reasonably approximated with a straight line. However, it is evident that the approximation with a straight line is fairly poor in the lateral regions of the domain where the target function presents a more complex shape. It is clear that a better choice of these regions would result in a better approximation. Then, in order to actually exploit the advantages of using simple FAs, we would need to define a searching strategy that permits to find those regions in which the target function can be well approximated with the function of choice.

One searching strategy could be the one used by the variable resolution (VR) approaches (section 2.3.2) to find the parts in which the target function is well approximated with a constant value. Basically, in these approaches, the searching is performed sequentially by splitting existing parts. We may, thus, use this strategy to find the regions in which the target function is well approximated by the simple function selected, e.g. a constant value.

However, this searching strategy has some drawbacks. For instance, in multi-dimensional spaces the newly generated parts inherit all except one of the borders of their parent. This may imply that, depending on how the splitting of parts is carried out from the beginning of the learning process, some parts become more difficult to be generated than others. In this case, the approach may need to perform many splitting steps until a fairly enough amount of different parts are tried, so as to find those where the target function may be sufficiently approximated. Another limitation is that, at each point, only one part is tried at a time. If a part does not produce good results, then a new process of splitting is carried out, and the system must wait until a sufficient amount of experiences are collected in the newly generated parts to be able to evaluate further specializations.

An alternative would be to replace the partition of the domain by a covering, in which different domains for the FAs may overlap. In this way, the shapes of the new parts may be freely defined, and many different domains around each point may be tried

simultaneously, increasing the opportunities of finding a region in which the function can be approximated by a constant value. Thus, the inference at a point could be provided in a competitive way by the FA that better approximates the target function.

#### 3.1.1.1 An Illustrative Example

To illustrate the idea of using a competitive strategy we perform a simple experiment in which a multi-step target function is approximated using two strategies: one where the domains of the FAs are searched as in the VR techniques, i.e. using a partition of the target function domain, and the other using the competitive strategy described in the previous section. The function selected for the FAs is just the constant-valued function, what indeed makes the strategy using a partition the typical VR technique. In both cases, the value used for a given FA is calculated as the mean value of the observations collected so far in the FA domain. For the sake of this illustration, we assume that the target function is known. This allows us to select, in the competitive strategy, the FA that better approximates the target function at a point as the one whose constant approximation is closer to the value of the target function at that point.

The approximation, in both cases, is performed incrementally, generating new FAs when a large approximation error is detected. For the case of the VR technique, the part containing the evaluated point is split in two halves, initializing the approximation of the half containing the point with the value of the function observed at that point, while the other half keeps the approximation of the old part. For the case of the competitive strategy, new FAs are generated in parallel, with domains defined by segments with end points chosen randomly in the total range of the input space around the experienced input. The initialization of the approximation of the new parallel FAs also consists in the observed value of the function.

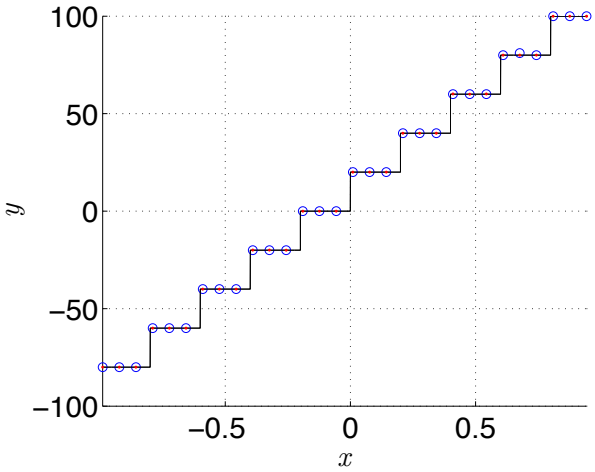
To show how the number of parallel FAs improves the chances of a good approximation, we implement, for the case of the competitive strategy, two different experiments: one generating 2 FAs, and the other generating 10 FAs once a large approximation error is detected. Figure 3.3 presents the result of all the experiments. Figure 3.3(a) shows, in black, the multi-step target function, and the approximations, at some test points, carried out by the VR strategy (blue circles), and by the competitive strategy with 2 FAs generation (red dots). Figure 3.3(b) presents, in turn, the convergence

profiles of the VR technique (blue), and the two cases of competitive strategy, with a generation of 2 FAs (red), and with a generation of 10 FAs (green). It can be seen that the competitive strategies converge much faster than the VR technique, achieving a smaller approximation error in a shorter time. In particular, the competitive strategy with a generation of 10 parallel FAs converge faster than the competitive strategy with a generation of 2 FAs, which suggests that the larger the number of parallel FAs, the better the generalization.

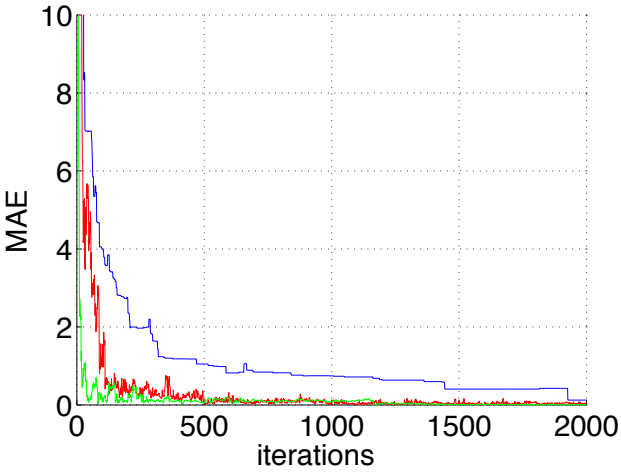
To illustrate the generalization capabilities of both approaches, the VR and the competitive strategy, we present, in figure 3.4, the number of parts used by both approaches for the approximation of the target function [1]. A vertical line indicates the transition between parts. After 2000 iterations, the VR strategy generated 173 parts for the approximation of the multi-step target function (figure 3.4(a)), while the competitive strategy, with 2 FAs generation, configures a partition with only 33 parts (figure 3.4(b)). Even more, although the number of parts configured by the competitive strategy is 33, the actual number of FAs used to approximate the target function is 26. This is possible since the parallel structure permits that the same FA is used for the inference at disjoint regions of the domain.

In addition to the potential benefits so far explained, there are some other interesting features of the competitive strategy. Since at each point there will usually be more than one FA, each providing its own inference of the value function, the competitive strategy is not limited by the accuracy of a single approximator, as happens in the VR approaches and in most of the approaches for FA in RL. If one approximator gets its approximation degraded, it will be supported by other approximators that may perform better, keeping a good overall performance of the system, and producing more stable convergence profiles. This makes the competitive approach attractive in RL since it permits to deal well with non-stationary target functions: when a drastic change of the target function occurs at some point of the domain, it is not necessary that the formerly best approximator at this point completely reshape its approximation before the output becomes accurate; instead, a different approximator that managed to be more accurate at that point will provide the output as soon as it proves to

[1] Note that, despite the domains of the FAs used in the competitive strategy do not configure a partition of the target function domain, the points of the domain of each FA for which the FA is the best for the inference, indeed configure a partition.

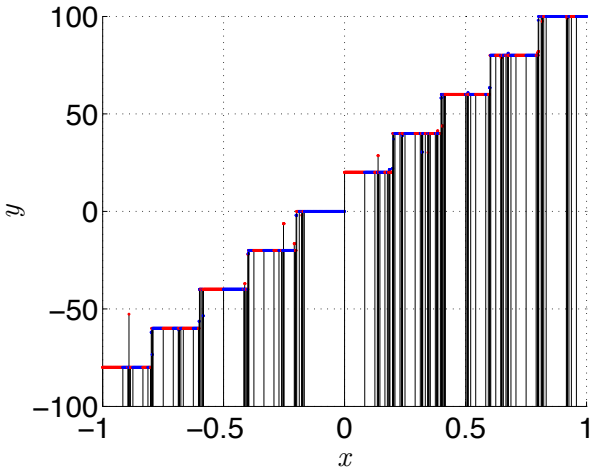



(a) Target function approximations.

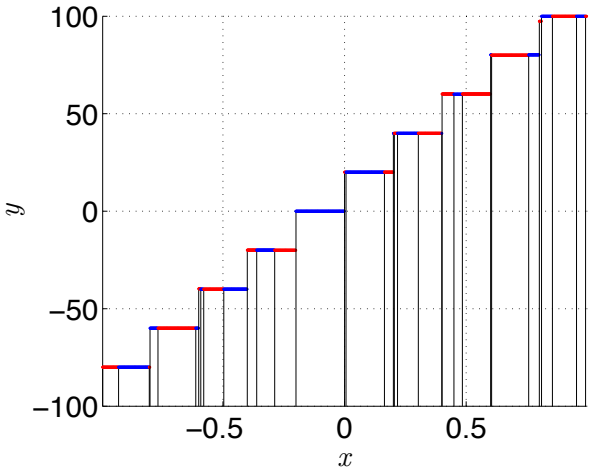



(b) Convergence profiles.

Figure 3.3: Comparison between a VR strategy and the competitive strategy in the approximation of a multi-step function. Figure 3.3(a) shows the target function depicted in black, while the approximations carried out by the VR strategy and the competitive strategy, are depicted in red and blue, respectively. In figure 3.3(b), the convergence profile of the VR approach is depicted in blue, while the ones corresponding to the competitive strategy are painted in red, for the case of 2 FAs generation, and in green for the case of 10 FAs generation.




(a) Detail of the number of parts used by the VR strategy.




(b) Detail of the number of parts used by the competing strategy.

Figure 3.4: Detail of the number of parts used by the VR strategy, and the competitive strategy, for the approximation of a multi-step function. The vertical lines delimit each part.

be more accurate there. An alternative approach would be to use for the inference a weighted average of the estimations of the approximators covering the point, rather than a winner-takes-all strategy. However, to provide an accurate output, a weighted average strategy would need to wait until all the approximators with a non-negligible weight get their approximations reshaped, while with a winner-takes-all strategy it would be enough that one approximator performs well there. Another point to remark is that the competitive strategy deals well with the problem of over-refinement, explained in section 2.3.2, as the generation of FAs does not slow the convergence, but could even improve it since more FAs represents more chances of having a good one among them, though at the expense of a larger computational cost.

In this chapter we propose a competitive strategy for function approximation in continuous domains. Since the approach developed generates parallel FAs on demand for a better approximation, it is non-parametric. Next section presents the formalization of a function representation using a competitive strategy, that will be used, in section 3.3, to formalize the idea of using competing function approximators for the approximation of an arbitrary target function. During that section, the formalizations presented are illustrated with some examples. The chapter ends with a brief conclusion.

## 3.2 Degenerate Function Representation

Before entering into the details about the mechanisms for function approximation using a competitive strategy, we provide a formal definition of the function that will be used to approximate a given target function.

We define a *competitor* function, $\Phi_i(x, \xi_i)$, as a parametrized function with parameters $\xi_i$, defined in a D-dimensional continuous domain $X_i \subset \Re^D$. Associated to each competitor $\Phi_i$, there is a *relevance* function, $\Gamma_i(x, \nu_i)$, which is a parametrized function with parameters $\nu_i$, also defined in the domain $X_i$.

We define a *degenerate* function [1] , $\mathcal{F}(x, \mathbf{\Phi}, \mathbf{\Gamma})$, as a function configured by a set of competitors $\mathbf{\Phi} = \left\{\Phi_1, \Phi_2, \ldots, \Phi_{|\mathbf{\Phi}|}\right\}$, and the corresponding set of relevance functions

[1] The denomination *degenerate* is borrowed from the *degeneracy* concept of the theory of Neuronal Group selection [32], in which the idea of using a competitive structure is based.

$\mathbf{\Gamma} = \left\{\Gamma_1, \Gamma_2, \ldots, \Gamma_{|\mathbf{\Phi}|}\right\}$. The function is defined in a domain $X = \bigcup_{i=1}^{|\mathbf{\Phi}|} X_i$, where the competitor domains form a covering of $X$.

The output of the function $y = \mathcal{F}(x, \mathbf{\Phi}, \mathbf{\Gamma})$, at a given input $x \in X$ is determined as follows: first, a *winner* competitor, $\Phi_w$, is selected such that

$$w = \underset{i \in I_x}{\operatorname{argmax}} \Gamma_i(x, \nu_i), \tag{3.1}$$

where $I_x = \{i | x \in X_i\}$ references the set of indices of the *active* competitors for $x$,

$$\mathbf{\Phi}_x = \{\Phi_i | x \in X_i\}. \tag{3.2}$$

Then, the value $y$ is obtained as

$$\begin{aligned} y &= \mathcal{F}(x, \mathbf{\Phi}, \mathbf{\Gamma}) \\ &= \Phi_w(x, \xi_w). \end{aligned} \tag{3.3}$$

## 3.3 Degenerate Function Approximation

We assume that there is an arbitrary unknown target function $f(x)$ that should be approximated, from which we can only observe its values at particular points obtained sequentially, $(x_t, y_t)$, $y_t = f(x_t)$, where $t$ accounts for the $t$-th time step. Our aim is to devise a non-parametric, online memory-free function approximation using a degenerate function representation. To this end we need to define the functions to be used for the competitors $\mathbf{\Phi}$, their respective relevance functions $\mathbf{\Gamma}$, that should indicate how good the approximation of the respective competitors are at an evaluation point, and the method that will be used to find the parameters $\xi_i$, and $\nu_i$, so as to get a good approximation of the target function,

$$f(x) \approx \mathcal{F}(x, \mathbf{\Phi}, \mathbf{\Gamma}). \tag{3.4}$$

In addition, since the approach will be non-parametric, we need to define the mechanisms for the generation of competitors on demand for a better approximation [1]. We

[1] It is also possible to make the approach non-parametric by changing not only the number of

call a method for function approximation that uses a degenerate function a *degenerate function approximation* (DFA).

In the definition of a DFA system, we could use any type of function for the competitor $\Phi(x, \xi)$, e.g. constant. The type of function selected will influence the generalization performed by the DFA, depending on the size of the regions of the target function that can be well approximated with $\Phi(x, \xi)$. However, if the relevance function selected does not permit a good distinction between a good or a bad approximation, the performance of the DFA system will be poor, disregarding the competitor selected. Hence, a correct definition of the relevance function is crucial for the good performance of the DFA system. Next section is focused on this aspect.

### 3.3.1 Relevance Function

To make the exposition clear, we will first assume that the target function is deterministic, and that the competitor and the relevance functions are constant-valued functions. We consider as the constant value for the competitor function the sample mean [22],

$$\Phi_i(x, \xi_i) = \bar{Y}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} y_j, \tag{3.5}$$

where $n_i$ is the total number of samples observed so far in $X_i$. To define the relevance function, note that the variance of $f$ in $X_i$, $\sigma_i^2 = E_{X_i}\left[(f(x) - E_{X_i}[f(x)])^2\right]$, is a good indicator of how good $E_{X_i}[f(x)]\,(\approx \bar{Y}_i)$ approximates $f$ in $x$. Hence, the inverse of the variance $\sigma_i^2$ would be a good value for the relevance. However, as we only have a limited amount of samples of $f$, the variance can not be determined precisely and should be estimated. We estimate the variance using the (unbiased) sample variance $S_i^2$ [22],

$$S_i^2 = \frac{1}{n_i - 1} \sum_{j=1}^{n_i} \left(y_j - \bar{Y}_i\right)^2. \tag{3.6}$$

Since the variance is estimated using a limited amount of information, its estimation carries uncertainties which vary with the number of samples $n_i$. To take into account

competitors but also the number of parameters $\xi$ and $\nu$. However, we will keep the number of these parameters fixed, varying only the number of competitors, to keep the approximation at each competitor as simple as possible.

these uncertainties we use a known result from Statistic [22],

$$\frac{1}{\sigma_i^2}\sum_{j=1}^{n_i}\left(y_j-\bar{Y}_i\right)^2 \sim \chi^2\left(n_i-1\right). \tag{3.7}$$

Equation (3.7) states that the expression on the left has a $\chi^2$ distribution with $n_i - 1$ degrees of freedom. Using (3.6) and (3.7) we obtain,

$$\frac{\left(n_i-1\right)S_i^2}{\sigma_i^2} \sim \chi^2\left(n_i-1\right). \tag{3.8}$$

This allows us to build a confidence interval for the true variance $\sigma_i^2$ that only depends on the sample variance and the number of samples experienced in the competitor domain [1]. For this, we use the definition of the $\alpha$-quantile $\chi^2_\alpha(h)$,

$$P(Y > \chi^2_\alpha(h)) = \alpha. \tag{3.9}$$

Equation (3.9) means that, if a random variable $Y$ has a $\chi^2$ distribution with $h$ degrees of freedom, the probability that $Y$ takes values greater than $\chi^2_\alpha(h)$ is $\alpha$. Fixing $\alpha$ to a convenient value, for instance $\alpha = 0.95$ to make things concrete, we can obtain an interval for $Y$ that is 95 % certain to include its true value. Taking $Y = \frac{(n_i-1)S_i^2}{\sigma_i^2}$ we have,

$$P\left(\frac{\left(n_i-1\right)S_i^2}{\sigma_i^2} > \chi^2_\alpha\left(n_i-1\right)\right) = P\left(\sigma_i^2 < \frac{\left(n_i-1\right)S_i^2}{\chi^2_\alpha\left(n_i-1\right)}\right) = \alpha. \tag{3.10}$$

The upper bound,

$$\frac{\left(n_i-1\right)S_i^2}{\chi^2_\alpha\left(n_i-1\right)}, \tag{3.11}$$

represents the highest possible value (with $\alpha$ confidence) of the actual unknown variance given the current uncertainties in the estimation. Less confident estimations will

[1] In principle, the method to calculate the confidence interval for the variance through the $\chi^2$ distribution assumes normally distributed samples. However, it is known [22] that this method is also valid when samples are obtained from any other distribution, different from the normal. In this case, the values provided would indicate approximated confidences, rather than accurate ones, that would become more precise as more samples are considered.

have higher upper bounds. The value (3.11) provides a measure of the quality in the approximation that balances the estimation accuracy and the confidence in that estimation. Hence, we adopt the inverse of the upper bound of the confidence interval for the variance as the relevance of a competitor $\Phi_i$,

$$\Gamma_i(x, \nu_i) = \frac{\chi^2_\alpha \, (n_i - 1)}{(n_i - 1) \, S_i^2}. \tag{3.12}$$

This definition of the relevance prevents that weak estimations with low variances are favoured against those with slightly higher variances but with much more confident estimations.

In the above development we assumed a deterministic target function. We now abandon this assumption and let the target function be stochastic. In this case, the sample variance is not uniquely determined by the approximation quality of a competitor, but also reflects the intrinsic variability of the samples due to the non-determinism. However, since these additional source of variability are essentially associated to a given point $x$, it will affect in a similar measure all the competitors covering it, and will not influence in the competition.

#### 3.3.1.1 An Illustrative Example

In order to illustrate the implementation of the DFA approach using constant competitor and relevance functions, we consider the same simple problem of approximating a multi-step function presented in section 3.1.1.1, but this time without assuming that we know the target function. Instead, we only get information of the target function at some points sampled sequentially. In the example, we consider constant-valued functions for the competitor and its relevance, calculated using formulas (3.5), and (3.6), respectively.

**Online, Memory-free, Updating**

The averages involved in (3.5) and (3.6) have the form,

$$\bar{f}_n = \frac{\sum_{j=1}^{n} f_j}{n} \tag{3.13}$$

where $f_j$ is the $j$-th observation of the function in the given domain. It is easy to see that the incremental, memory-free, calculation of this generic average is,

$$\bar{f}_n = \left(1 - \frac{1}{n}\right) \bar{f}_{n-1} + \frac{1}{n} f_n. \tag{3.14}$$

In the previous incremental formula (3.14), all the samples are weighted with the same factor $1/n$, no matter how long ago they were observed. This would be appropriate in cases where all the experiences are equally important for the average. However, when the estimations depend on quantities that are expected to improve along time, old observations are less accurate and should progressively be replaced by newer, more accurate ones. For instance, the sample variance (3.6) is calculated using the squared difference between the sample mean and the target function. Since the sample mean is improving along time, later observations of this error are more accurate than old ones, and a forgetting must be considered. This forgetting is also necessary in cases where the target function is non-stationary, like in RL applications. In this case, the variations of the target function also require to discard old, possibly outdated, values by including a forgetting effect in the updating.

To implement such forgetting, we make use of the incremental updating formula for stochastic approximations [43], which has the same form of the incremental formula (3.14), but considering, instead of factor $1/n$, a generic definition of a learning coefficient $\eta(n)$ that can be regulated to produce the desired forgetting effect. Using the stochastic approximation approach, equation (3.14) is replaced by,

$$\bar{f}_n = (1 - \eta(n))\bar{f}_{n-1} + \eta(n) f_n. \tag{3.15}$$

To produce a forgetting, the factor $\eta(n)$ should be defined to avoid a fast dropping to zero of the learning coefficient, what may produce the average to get stuck before reaching the correct solution. This may be accomplished by establishing the condition $\eta(n) > 1/n$ which makes new incomes more relevant than old ones, with a forgetting effect determined by the particular setting of $\eta$. However, the definition of the learning coefficient $\eta(n)$ is not unrestricted if we want to guarantee convergence to a solution. To this end, the function $\eta(n)$ should be defined in a way that it fulfils the stochastic

approximation requirements (2.15) and (2.16). A known set of functions which fulfil the stochastic approximation conditions are those of the form,

$$\eta(n) = \frac{1}{an + b}, \tag{3.16}$$

where $0 < a < 1$, and $b > 0$. This formula, even though simple, permits a wide variety of behaviours for the learning coefficients. For instance, making $a = 1$ and $b = 0$, we get a learning coefficient $\eta(n) = 1/n$, which is suitable for cases where no forgetting should occur. For values $a < 1$, the forgetting increases as the parameter $a$ decreases. In the extreme case of $a = 0$ and $b = 1$, the forgetting of past values would be complete, and the estimations would consist only in the current observations. Hence, the parameter $b$ limits the influence of the parameter $a$ in the learning coefficient behaviour, and regulates the forgetting effect mainly for lower values of $n$.

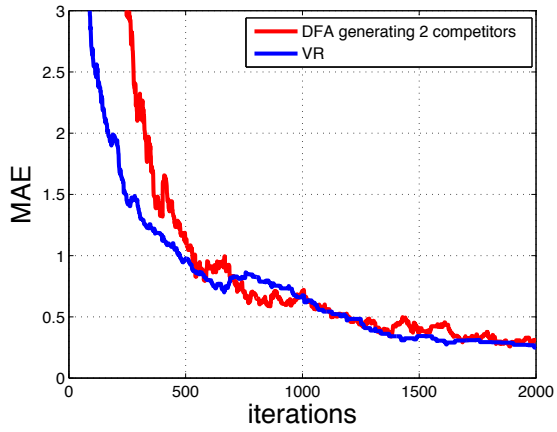
### Competitor Generation

The generation of new competitors for this simple example is carried out as in section 3.1.1.1, by generating a predefined number of competitors with random domains each time a large approximation error is encountered. However, in this case, instead of allowing the generation of competitors with domains as large as the domain of the target function, we restrict the size of the domain of each new competitor to the size of the domain of the winner competitor. The segment corresponding to the domain of a new competitor is defined using a left endpoint randomly generated in $[x - \Delta, x]$, and a right endpoint randomly generated in $[x, x + \Delta]$, where $x$ is the evaluated point, and $\Delta$ is half of the length of the segment corresponding to the domain of the winner competitor.

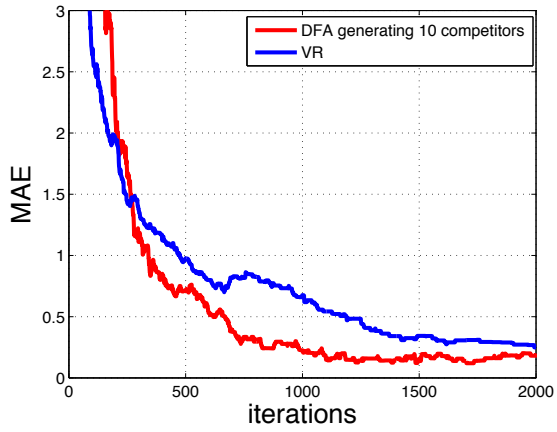
### Results

As the results of the example we use the average of 10 runs of the mean absolute error (MAE) obtained at each iteration [1] . The results, compiled in figure 3.5, show how the convergence rate of the DFA method improves with the number of competitors generated every time a large approximation error is detected.

[1] In the example presented in section 3.1.1.1 we showed the results of a single run just to illustrate the advantages of using a competitive strategy.




(a) VRFA versus DFA with 2 competitors generation.




(b) VRFA versus DFA with 10 competitors generation.

Figure 3.5: Comparison between the performance of the VR strategy and the performance of the DFA with constant-valued competitor and relevance.

For the case of generating 2 competitors (figure 3.5(a)), the performance of the DFA is comparable with that of the variable resolution approach. The VR approach generated, in average, 118 parts, while the DFA approach configured a smaller average of 54 parts. In this case, the random generation strategy used by the DFA method is not able to generate enough competitors whose approximation capabilities permit to overcome the approximation of the VR approach. The performance of the DFA approach is improved significantly for the case of generating 10 competitors (figure 3.5(b)), surpassing the performance of the VR approach. The average number of parts configured by the DFA method with 10 competitors generation is 64. These results show that the system improves its performance with the number of competitors generated, supporting our hypothesis that trying more FAs in parallel increases the opportunities of generalization.

Note that the performance of the DFA method in the case of an unknown target function is worst than the performance shown in section 3.1.1.1. This is expected since, in that case, we use the unrealistic assumption that the target function is known, and, hence, which approximation has the highest relevance at each point. In this case, the relevance has to be estimated with incoming samples.

#### 3.3.1.2 Point-Dependent Relevance

In the developments presented before we considered constant-valued estimations for the competitor function and the relevance function. However, the DFA system permits to regulate the precision of the relevance of a competitor by providing more parameters $\nu_i$ to the relevance function $\Gamma_i(x, \nu_i)$. In general, the sample variance $S_i^2$ (3.6), can be approximated with a multi-parametric estimation $S_i^2(x, \tau_i)$, with parameters $\tau_i$. Using this estimation, we can estimate the sample variance for samples experienced in a region $\tilde{X}_i \subset X_i$ as,

$$\tilde{S}_i^2 = \frac{\int_{\tilde{X}_i} S_i^2(x, \tau_i)\, p_i(x, \varsigma_i) dx}{\int_{\tilde{X}_i} p_i(x, \varsigma_i) dx}, \tag{3.17}$$

where $p_i(x, \varsigma_i)$ is a parametric estimation of the probability density function in $X_i$. In

a similar way, we can estimate the number of samples in $\tilde{X}_i$ as,

$$\tilde{n}_i = n_i \int_{\tilde{X}_i} p_i(x, \varsigma_i) dx. \tag{3.18}$$

Now, replacing the sample variance and number of samples in $X_i$ in the upper bound formula (3.11), with their counterparts in $\tilde{X}_i$ we get,

$$\frac{(\tilde{n}_i - 1)\, \tilde{S}_i^2}{\chi_\alpha^2\, (\tilde{n}_i - 1)}, \tag{3.19}$$

which leads to express the relevance function for a point $x \in \tilde{X}_i$ as,

$$\Gamma_i(x, \nu_i) = \frac{\chi_\alpha^2\, (\tilde{n}_i - 1)}{(\tilde{n}_i - 1)\, \tilde{S}_i^2}. \tag{3.20}$$

With this relevance formula we can regulate the precision of the relevance of a competitor at a point $x$ by adjusting the size of the region $\tilde{X}_i$ to an appropriate value depending on the nature of the generalization performed and the regularity of the variance function. In order to simplify the calculations of the integrals in (3.17) and (3.18), we assume that the region $\tilde{X}_i$ is small enough for the probability density function $p_i(x, \varsigma_i)$ and the variance function $S_i^2(x, \tau_i)$ to be nearly constant, which permits to estimate the relevance (3.20) as,

$$\Gamma_i(x, \nu_i) \approx \frac{\chi_\alpha^2\, (h_i)}{h_i\, S_i^2(x, \tau_i)}, \tag{3.21}$$

where $h_i = \tilde{V}_i\, n_i\, p_i(x, \varsigma_i) - 1$. $\tilde{V}_i$ is the volume of $\tilde{X}_i$ that, in practice, is defined empirically for each particular problem.

### Influence of Point-dependent Relevance in Competitor Selection

A precise relevance permits a better use of the competitor approximation capabilities since it may permit to select a competitor whose approximation is closer to the target function in a point, no matter how large the approximation error may be at other regions of its domain. To illustrate this idea, we perform another simple experiment of approximating a plain step function (figure 3.6). In the experiments we compare the

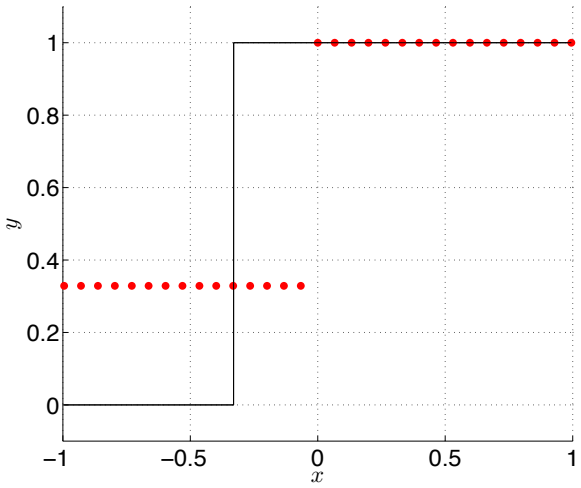
approximation carried out by two DFA approaches, one with relevance functions using a constant estimation of the variance (3.6), and the other using a linear estimation of the variance,

$$S_i^2(x, \tau_i) = A_i x + B_i, \tag{3.22}$$

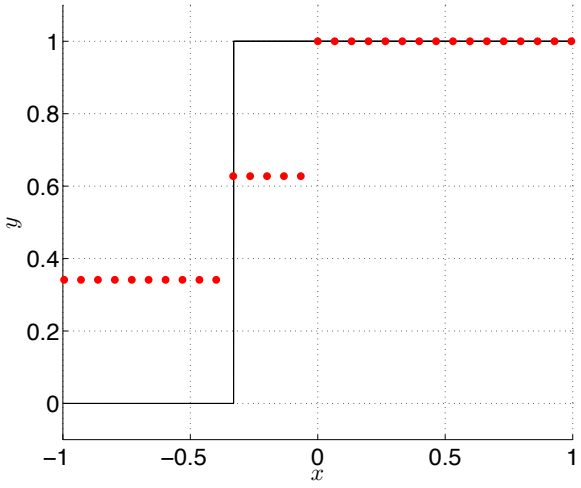
where $\tau_i = \{A_i, B_i\}$. In both cases, we use the sample mean (3.5) as the competitor functions, and assume an uniform probability density function with value $p_i(x, \varsigma_i) = 1/V_i$, where $V_i$ is the volume of the competitor domain. For the experiment we use a DFA with three competitors: two non-overlapped competitors, each one with a domain consisting in a half of the target function domain, and one competitor that covers the entire target function domain. For illustrative purposes, we keep the number of competitors fixed.

Figure 3.6(a) shows the approximation of the step function achieved by the DFA approach for the case of a constant relevance function (MAE=0.22). Figure 3.6(b) presents, in turn, the results corresponding to the case of a relevance function with a linear estimation of the variance (MAE=0.17). The improvement in the approximation is achieved thanks to the more precise relevance calculation, which permits to perform a more efficient selection of the best competitor at a point [1]. The relevance of each competitor for the case of a linear variance estimation is depicted in figure 3.7(b), while the corresponding variance estimations are presented in figure 3.7(a). Note that, in the case of a linear estimation of the variance, the largest competitor, whose relevance is depicted in red, becomes more relevant than the left half competitor immediately to the right of the step. This is so since the approximation of the largest competitor is closer to the target function in this region than the more specific left half competitor. Contrarily, in the constant relevance case, the left half competitor wins over the largest competitor in the region immediately to the right of the step, when actually the largest competitor performs a better approximation there. This is so since the global constant relevance does not permit to discriminate between regions well or bad approximated within the

[1] It is important to remark that, despite a more precise relevance permits to make a better selection of the competitors, the parameters of the relevance function should be estimated from incoming samples and, as we have already shown, the higher the number of parameters to be balanced, the higher the number of experiences required to balance them. This establishes a trade-off between the accuracy in the competitor selection and the convergence speed.

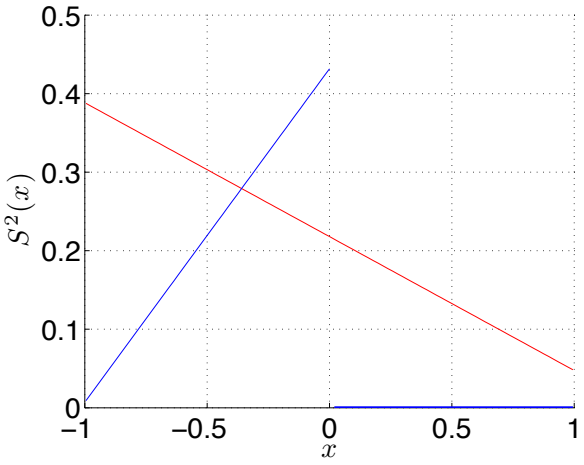



(a) Approximation using a constant estimation of the variance.

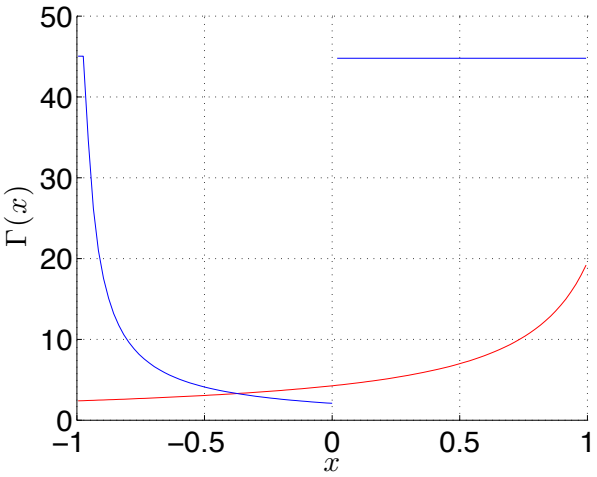



(b) Approximation using a linear estimation of the variance.

Figure 3.6: Comparison between a DFA method using a constant estimation of the variance, and using a linear estimation of the variance, in the approximation of a step function. The target function is depicted in a black line, while the approximation of the DFA method is shown in red dots.




(a) Variance estimations in the competitors domains.




(b) Relevance of the competitors.

Figure 3.7: Relevance estimation, and their corresponding variance estimations, of the competitors. The results for the larger competitor are presented in red. The results for the smaller competitors covering the left and right half of the target function domain are presented in blue. Note that the variance of the right half competitor tends to zero as learning proceed, which makes the relevance to tend to infinite. For clarity in the figure we limit the minimum value of the variance to a value close to zero.

competitor domain. This implies that, in order to achieve any desired precision in the approximation when using constant relevance, we would need to generate more specific competitors until we find those which perform a good enough approximation in all its domain, so as to surpass the average approximation quality of other competitors. For instance, in the example, the DFA with constant relevance may achieve a more precise approximation by generating two more competitors resulting from the splitting of the competitor on the left half.

#### 3.3.1.3 Relevance Estimation under Biased Sampling

In the case of a biased sampling, the relevance estimation when using a global constant relevance becomes much more inaccurate, and the system may not be able to achieve any desired precision by just generating competitors. This is so because, under a biased sampling, the relevance of a competitor reflects mainly the quality in the approximation at frequently sampled regions, providing a weak indication of this quality at weakly sampled ones. Therefore, the necessity of a more informative relevance estimation is increased when biased sampling occurs, in order to permit an approximation to any desired precision.

To illustrate the effect of biased sampling, we use the same example of the step target function, but now, instead of providing observations from a uniform sampling, we generate such observations using a biased sampling. To implement this, we obtain samples from the right half with a probability of 0.75, and from the left half with a probability of 0.25. Figure 3.8(a) presents the approximation achieved by the DFA approach with constant relevance, while figure 3.8(b) shows the approximation achieved by the DFA approach using a linear variance estimation for the relevance.

Note that, in the case of a constant relevance, the largest competitor has an estimation closer to 1 than in the uniform sampling case (figure 3.6(a)), since it is fed mostly from samples on the right half, where the target function has a value of 1. Its relevance value is also increased since the average approximation quality in the sampled points is better. When an inference has to be done on the left half, the relevance of the largest competitor surpasses the relevance of the left half competitor, producing an incorrect competitor selection (overall MAE=0.3). Conversely, if we now use a linear estimation of the variance, the competitor selection becomes much better, permitting,

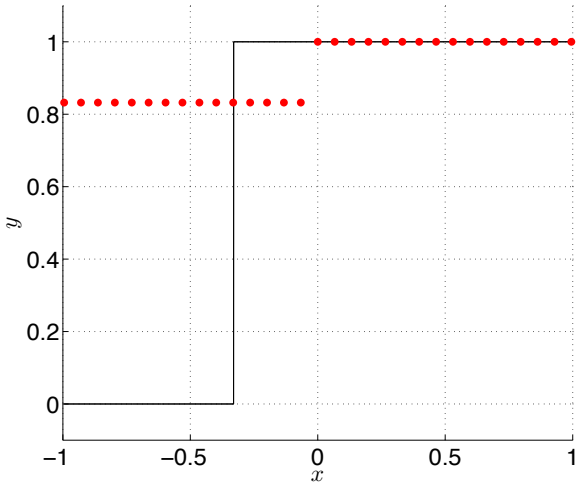
in this case, to select the best competitor for the inference at practically the entire domain (MAE=0.15) (figure 3.6(a)). The linear approximation of the variances, and their corresponding relevance functions, now for the case of a biased sampling, are shown in figure 3.9(a), and 3.9(b), respectively.

#### 3.3.1.4 An Illustrative Example: Point-dependent Relevance Case

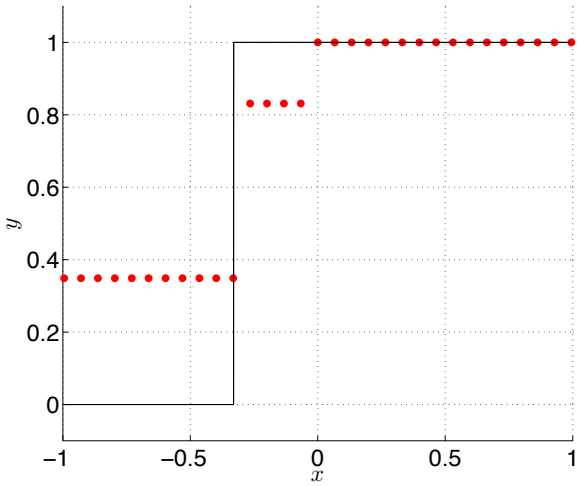
In order to evaluate more thoroughly the gain in the approximation when considering a more accurate relevance, we carried out the same experiments presented in section 3.3.1.1, but this time considering a linear estimation of the variance in the relevance functions. Figure 3.10 presents the results.

The experiment with a linear estimation of the variance and two competitors generation (green line) converges faster, and to a more precise approximation, than the constant variance estimation case (red line). This is not the case for the experiments with 10 competitors generation, where both strategies, using a constant variance estimation and using a linear variance estimation, have similar performances. This is because the amount of competitors generated in this last case is very high, and so there are many chances of generating competitors that cover a region where the target function is constant. Under these circumstances there is no much gain of using a linear variance estimation since a constant variance provides already a very precise indication of the quality in the approximation at all the points of the competitor domain. This is also the case in the previous example of the step target function approximation, where the competitor defined on the right half of the domain approximates exactly the target function (figures 3.6 and 3.8).

Indeed, in the example of a multi-step target function, using a linear estimation of the variance only makes sense when there are not enough competitors whose domains cover a region where the target function is constant. In this case, a linear variance estimation permits to better distinguish between good and bad approximated regions of the competitor domain than a plain constant variance estimation, and, hence, to better select the best competitor in a point. This is also the case of the previous example for the competitors covering the left half of the domain (figures 3.6 and 3.8), and the reason of the better performance of the DFA with a linear variance estimation in the experiments with 2 competitors generation (figure 3.10(a)).

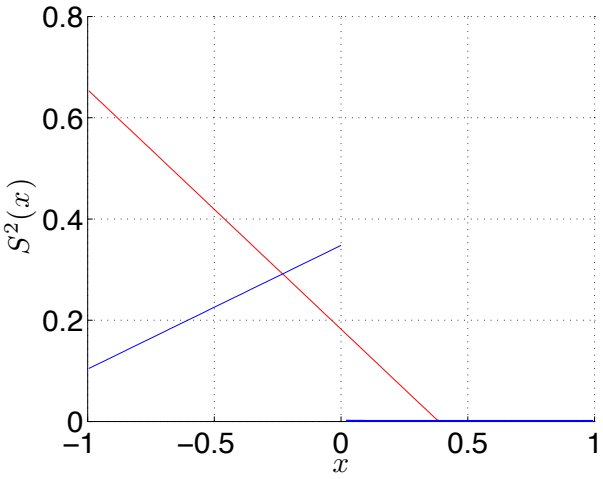



(a) Approximation under biased sampling using a constant relevance .

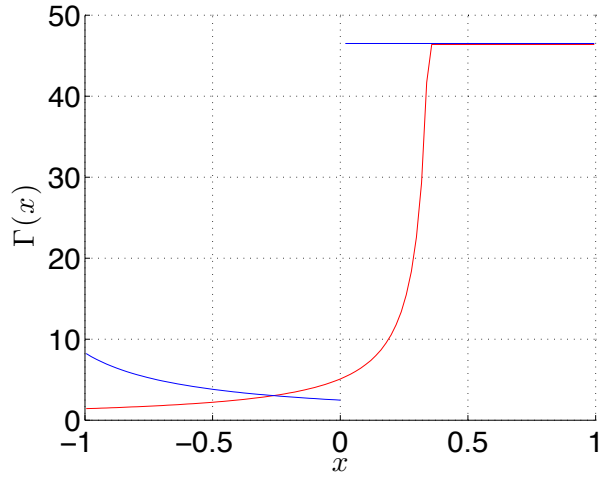



(b) Approximation under biased sampling using a linear estimation of the variance for the relevance.

Figure 3.8: Comparison between a DFA method using a constant estimation of the variance, and using a linear estimation of the variance, in the approximation of a step function under biased sampling. The target function is depicted in a black line, while the approximation of the DFA method is shown in red dots.

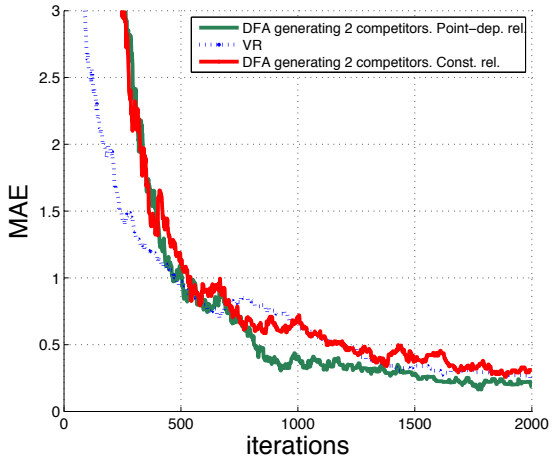



(a) Variance estimations in the competitors domains.

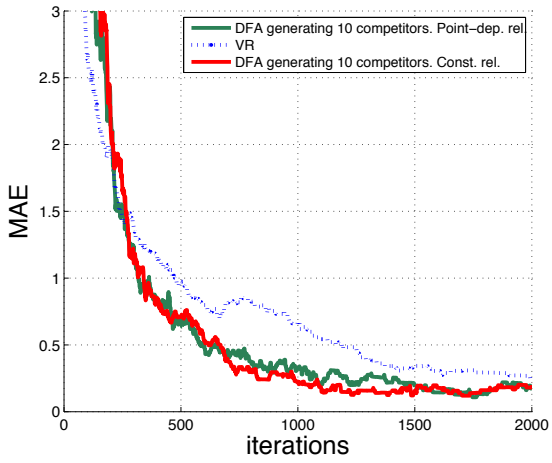



(b) Relevance of the competitors.

Figure 3.9: Variance estimation, and their corresponding relevance values, of the three competitors used for the example, now fed using a biased sampling. The results for the larger competitor covering the entire domain of the target function are presented in red. In blue, the results for the smaller competitors covering the left and right half of the domain, respectively. For consistency, we make the variance to be zero when the linear estimation gives a value below zero. In the figure, this limit is set to a value close to zero to permit a clear visualisation of the relevance functions.




(a) VRFA versus DFA with 2 competitors generation.




(b) VRFA versus DFA with 10 competitors generation.

Figure 3.10: Comparison among the performances of the VR strategy, the DFA approach with constant-valued relevance, and of the DFA approach with point-dependent relevance.

### 3.3.2 Competitor Management

In order to approximate an arbitrary function to any desired precision, the number of competitors should be varied on demand for a better approximation. The idea is to find, through generating competitors, the regions of the domain in which the target function can be well approximated with the function used by each competitor. This implies that the generalization capabilities of the DFA will depend on whether the approximated function presents wide regions of this kind, and the ability of the method to create competitors which capture these regularities. Any generation mechanism can be employed for this purpose, as far as the new competitors enrich the approximation capabilities of the system.

The strategy adopted for the generation of competitors may produce competitors that, generated once to solve the approximation needs, become less useful for the approximation as a consequence, for instance, of the non-stationarity of the target function. Despite a large amount of competitors improves the chances of generalization, an elimination strategy of less useful competitors can be also implemented to avoid an excessive computational cost in storing and processing all the competitors.

### 3.3.3 Algorithm for the DFA

The complete algorithm for the degenerate function approximation is presented in algorithm 4. Note that, in this algorithm, the updating of the parameters $\xi_i$ and $\nu_i$ may be done using any strategy, either memory-based or memory-free. However, in our case, we use an online, memory-free, strategy in every application of the DFA.

## 3.4 Conclusions

In this chapter we have introduced the advantages of using a competitive strategy for function approximation, and presented the mechanisms of a competitive strategy called degenerate function approximation (DFA). This strategy uses a set of function approximators, named as competitors, that compete to provide the inference in a point. The competitor selected for the inference is obtained using a relevance function that

**Algorithm 4** Degenerate Function Approximation

initialize the degenerate function, $\mathcal{F}(x)$, using a set of competitors $\mathbf{\Phi}$ whose domains determine a covering of the target function domain
**loop**
  get observation $< x, y >$
  get the active competitors $\mathbf{\Phi}_x$, and the corresponding relevance functions $\mathbf{\Gamma}_x$
  select the winner competitor in $x$, $\Phi_w(x)$
  calculate the approximation error $e(x) = (y - \Phi_w(x))^2$
  update parameters $\xi_i$ of competitors $\Phi_i \in \mathbf{\Phi}_x$ using sample $< x, y >$
  update parameters $\nu_i$ of relevance functions $\Gamma_i \in \mathbf{\Gamma}_x$ using sample $< x, e(x) >$
  **if** there is a need of competitor generation **then**
    generate new competitors and their corresponding relevance functions
  **end if**
  **if** the criterion for elimination is fulfilled **then**
    eliminate less useful competitors and their respective relevance functions
  **end if**
**end loop**

quantifies the quality in the approximation of each competitor. So far, we have presented the formalization of this competitive strategy, illustrating how its mechanisms are articulated using simple representations for the competitor and relevance functions.

Next chapter presents an approach that provides, from a single model, a multivariate non-parametric approximation of the sample variance, the sample density, and the target function. This approach, in its parametric version, will lately be embedded in each competitor (see chapter 5), permitting a very specific relevance estimation, and the definition of competitor functions with arbitrary shapes and complexities.

# Chapter 4

# $Q$-Learning using Probability Density Estimations

## 4.1 Introduction

In the previous chapter we introduced the benefits of using a competitive strategy for function approximation in RL, and the formalisms of a competitive strategy called degenerate function approximation (DFA). The DFA strategy consists in a set of competing FAs, referenced as *competitors*, and a set of *relevance* functions, used to select the best competitor for the inference at each evaluated point. In this chapter we present a new approach for FA in RL based in a probability density estimation. This approach will permit to use for each competitor a multi-parametric function approximation, and to provide a precise calculation of the relevance of each competitor (3.21).

The approach consists in a multivariate probability density estimation of the experienced samples in the joint input-output space, from which it is possible to easily derive a multi-parametric approximation of the target function, a multi-parametric estimation of the variance of the samples, as well as a precise estimation of the density of samples experienced so far.

We chose to use a probability density model since it provides all the information we need to enrich the DFA system. Density estimations are receiving increasing interest in the field of machine learning [21], since they keep all the information contained in the data. Despite being more demanding than simple function approximation (due to the fact that they embody more information), their use for function approximation has

been advocated by different authors [35, 36]. One reason is that simple and well known tools, like the Expectation-Maximization (EM) algorithm, can be used to rapidly obtain accurate estimations of the density function.

The work presented in this chapter will focus on the evaluation and formalization of a probability density estimation approach for function approximation in RL. This is done to test the feasibility of the approach in solving the problem of FA in RL before using such density model for the instantiation of the competitor and relevance functions in the DFA system. We will first introduce the mechanisms to use this density estimation for an online memory-free approximation of a target function, to lately propose a strategy that combines these mechanisms with those of $Q$-Learning for the function approximation of the action-value function.

When using the probability density estimation for the problem of FA in $Q$-Learning, the idea is to represent the density distribution of the observed samples in the joint space of states, actions, and $q$-values, $p(s, a, q)$. With this approach, it is possible to obtain, for each given state and action, the probability distribution of $q(s, a)$ as the conditional probability $p(q|s, a)$. From this distribution we can obtain an estimation for the action-value, $\hat{Q}(s, a)$, as the expected value of $q(s, a)$. In addition, we can obtain, from the probability distribution $p(q|s, a)$, the variance of $q(s, a)$ [1].

As a further benefit of using density estimations, it is possible, by marginalization on the state-action variables, to obtain the local sampling density in a point $(s, a)$. This information will be used to precisely estimate the number of samples influencing the statistics (3.18). The precise information about the density of samples, together with the probability distributions of the $q$-values for a given $(s, a)$, can be used to define a wide spectrum of exploration strategies. In this chapter we propose one exploration strategy based on these quantities.

For the estimation of the density distribution we use a Gaussian Mixture Model (GMM). This is a simple approach with large literature supporting its consistency and efficiency for the estimation of multivariate densities [21]. The parameters of the GMM are estimated with the EM method [29], which is originally defined for batch mode, i.e. when all the samples are provided in advance. According to the requirements of our

[1] So that our approach also presents what has been argued to be an important feature of GPs [28, 33].

approach, we devise an online, memory-free, version of the EM. The density estimation provides the information necessary to regulate the forgetting of past entries required in online, memory-free, updating formulas. In our proposed updating formula, this forgetting is concentrated only in the explored region, according to the amount of data collected near the sample, and not as a function of time, which produces an inconsistent forgetting in regions where no new information is provided. This prevents the undesired distortion of the estimations in regions far from the experienced samples caused by the biased sampling problem.

The outline of the chapter is the following. We first provide a brief introduction to the Gaussian Mixture Model (GMM). Then, we present the outline of the Expectation-Maximization algorithm (EM) for the calculation of the GMM parameters, and present our proposal for the online, memory-free, version of the EM. Afterwards, the use of the GMM for an online, memory-free, function approximation is presented, and how this FA method is combined with $Q$-Learning for the approximation of the action-value function. The method is lately tested on the benchmark task of controlling an inverted pendulum with limited torque, where comparisons with state of the art methods are carried out. The chapter finishes with a conclusion section.

## 4.2 The Gaussian Mixture Model

A Gaussian Mixture Model [21] is a weighted sum of multivariate Gaussian probability density functions, and is used to represent general probability density distributions in multidimensional spaces. It is assumed that the samples of the distribution to be represented have been generated through the following process: first, one Gaussian is randomly selected with *a priori* given probabilities, and then, a sample is randomly generated with the probability distribution of the selected Gaussian. According to this, the probability density function of generating sample $\mathbf{x}$ is:

$$p(\mathbf{x};\Theta) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(\mathbf{x};\mu_i,\Sigma_i), \tag{4.1}$$

where $K$ is the number of Gaussians of the mixture; $\alpha_i$, usually denoted as the mixing parameter, is the prior probability, $P(i)$, of Gaussian $i$ to generate a sample; $\mathcal{N}(\mathbf{x};\mu_i,\Sigma_i)$ is a multidimensional normal Gaussian function with mean vector $\mu_i$ and covariance

matrix $\Sigma_i$; and $\Theta = \{\{\alpha_1, \mu_1, \Sigma_1\}, ..., \{\alpha_K, \mu_K, \Sigma_K\}\}$ is the whole set of parameters of the mixture. By allowing the adaption of the number $K$ of Gaussians in the mixture, any smooth density distribution can be approximated arbitrarily close [35]. The parameters of the model can be estimated using a maximum-likelihood estimator (MLE). Given a set of samples $\mathbf{X} = \{\mathbf{x}_t; t = 1, \ldots, N\}$, the likelihood function is given by

$$\mathcal{L}[\mathbf{X}; \Theta] = \prod_{t=1}^{N} p(\mathbf{x}_t; \Theta). \tag{4.2}$$

The maximum-likelihood estimation of the model parameters is the $\Theta$ that maximizes the likelihood (4.2) for the data set $\mathbf{X}$. Direct computation of the MLE requires complete information about which mixture component generated which instance. Since this information is missing, the EM algorithm, described in the next section, is often used.

## 4.3 The Expectation-Maximization algorithm

The Expectation-Maximization (EM) algorithm [29] is a general tool that permits to estimate the parameters that maximize the likelihood function (4.2) for a broad class of problems when there are some missing data. The EM method first produces an estimation of the expected values of the missing data using initial values of the parameters to be estimated (E step), and then computes the MLE of the parameters given the expected values of the missing data (M step). This process is repeated iteratively until a convergence criterion is fulfilled.

For the case of an estimation for the GMM, the process starts with an initialization of the mean vectors and covariance matrices of the Gaussians. The E step consists in obtaining the probability $P(i|\mathbf{x}_t)$ for each Gaussian $i$ of generating instance $\mathbf{x}_t$, that we denote by $w_{t,i}$,

$$w_{t,i} = P(i|\mathbf{x}_t) = \frac{P(i)p(\mathbf{x}_t|i)}{\sum_{j=1}^{K} P(j)p(\mathbf{x}_t|j)} = \frac{\alpha_i \mathcal{N}(\mathbf{x}_t; \mu_i, \Sigma_i)}{\sum_{j=1}^{K} \alpha_j \mathcal{N}(\mathbf{x}_t; \mu_j, \Sigma_j)}, \tag{4.3}$$

where $t = 1, .., N$ and $i = 1, .., K$. The maximization step consists in computing the MLE using the estimated $w_{t,i}$. It can be shown [31] that, for the case of a GMM, the

mixing parameters, means, and covariances are given by

$$\alpha_i = \frac{1}{N}\sum_{t=1}^{N} w_{t,i}, \tag{4.4}$$

$$\mu_i = \frac{\sum_{t=1}^{N} w_{t,i}\mathbf{x}_t}{\sum_{t=1}^{N} w_{t,i}}, \tag{4.5}$$

and,

$$\Sigma_i = \frac{\sum_{t=1}^{N} w_{t,i}(\mathbf{x}_t\mathbf{x}_t')}{\sum_{t=1}^{N} w_{t,i}} - \mu_i\mu_i', \tag{4.6}$$

respectively.

### 4.3.1 Online EM

Estimating a probability density distribution by means of the EM algorithm involves the iteration of E and M steps on the complete set of available data, that is, the mode of operation of EM is in batch. However, we chose to use an online, memory-free, approach, where data arrive sequentially, and none of them is stored. This prevents the use of the batch EM algorithm, and requires an online, memory-free, version of it. Several online EM algorithms have been proposed for the Gaussian Mixture Model applied to clustering or classification of stationary data [17, 80].

The approach proposed in [80] in not strictly an on-line EM algorithm. It applies the conventional batch EM algorithm onto separate data streams corresponding to successive episodes. For each new stream, a new GMM model is trained in batch mode and then merged with the previous model. The number of components for each new GMM is defined using the Bayesian Information Criterion, and the merging process involves similarity comparisons between Gaussians. This method involves many computationally expensive processes at each episode and tends to generate more components than actually needed. The applicability of this method to RL seems limited, not only for its

computational cost, but also because, due to the non-stationarity of the $Q$-estimation, old data should not be taken as equally valid during all the process.

The work of [17] performs incremental updating of the density model using no historical data and assuming that consecutive data vary smoothly. The method maintains two GMMs: the current GMM estimation, and the previous GMM of the same complexity after which no model updating (i.e. no change in the number of Gaussians) has been done. By comparing the current GMM with the historical one, it is determined if new Gaussians are generated or if some Gaussians are merged together. Two observed shortcomings of the algorithm are that the system fails when new data is well explained by the historical GMM, and when consecutive data violate the condition of smooth variation.

In [78], an on-line EM algorithm is presented for the Normalized Gaussian Network (NGnet), a model closely related to the GMM. This algorithm is based on the works of [56, 58]. In [58] a method for the incremental adaptation of the model parameters using a forgetting factor and cumulative statistics is proposed, while in [56] the method in [58] is evaluated and contrasted with an incremental version which performs steps of EM over a fixed set of samples in an incremental way. The method proposed in [78] uses foundations of both works to elaborate an on-line learning algorithm to train a NGnet for regression, where weighted averages of the model parameters are calculated using a learning rate that implicitly incorporates a time dependent forgetting factor to deal with non-stationarities. Inspired by this work, we developed an on-line EM algorithm for the GMM. Our approach uses cumulative statistics whose updating involves a forgetting factor explicitly.

#### 4.3.1.1 Our Proposal

The averages of equations (4.5), and (4.6), involved in the M step of the EM algorithm, have the form

$$\bar{f}_n = \frac{\sum_{j=1}^{n} w_j f_j}{\sum_{j=1}^{n} w_j}, \tag{4.7}$$

where $0 \leq w_j \leq 1$ indicates the proportion of the $j$-th sample $f_j$ used to calculate the average. This formula is a generalization of the average formula (3.13), where the samples are provided in their full proportions. For the cumulative sums used in (4.7) we introduce the notation,

$$[f]_n = \sum_{j=1}^{n} w_j f_j, \tag{4.8}$$

whose incremental version is simply,

$$[f]_n = [f]_{n-1} + f_n w_n. \tag{4.9}$$

Using this notation, the average (4.7) may be expressed as,

$$\bar{f}_n = \frac{[f]_n}{[1]_n}, \tag{4.10}$$

which allows an incremental calculation of it, by first updating the cumulative sums for the numerator and denominator, and then performing their ratio. In our online EM approach, we will adopt a similar strategy to calculate incrementally the averages involved in the M step. To do this, for each new income, we calculate the activation of each Gaussian using (4.3), though just for the incoming sample, and then update incrementally all the cumulative sums involved in the M steps,

$$[1]_{t,i} = \sum_{\tau=1}^{t} w_{\tau,i}, \tag{4.11}$$

$$[\mathbf{x}]_{t,i} = \sum_{\tau=1}^{t} w_{\tau,i}\mathbf{x}_\tau, \tag{4.12}$$

$$\left[\mathbf{x}\mathbf{x}'\right]_{t,i} = \sum_{\tau=1}^{t} w_{\tau,i}(\mathbf{x}_\tau \mathbf{x}'_\tau). \tag{4.13}$$

For the simple case where all the samples are equally valid (no forgetting), the averages (4.4), (4.5), and (4.6), can be calculated incrementally as,

$$\alpha_{t,i} = \frac{[1]_{t,i}}{\sum_{j=1}^{K} [1]_{t,j}}, \tag{4.14}$$

$$\mu_{t,i} = \frac{[\mathbf{x}]_{t,i}}{[1]_{t,i}}, \tag{4.15}$$

and

$$\Sigma_{t,i} = \frac{[\mathbf{x}\mathbf{x}']_{t,i}}{[1]_{t,i}} - \mu_{t,i}\mu'_{t,i}, \tag{4.16}$$

respectively.

However, when all the samples are not equally valid, like in non-stationary cases, we need to consider some forgetting in the cumulative sums in order to discard past, possibly outdated, values. The usual approach to do this is to consider a forgetting factor $\lambda_t$ in the following way

$$[[f]]_t = \lambda_t [[f]]_{t-1} + f_t w_t, \tag{4.17}$$

where the double brackets are used to distinguish the cumulative sum with a forgetting factor $\lambda_t$, with respect to the sum without forgetting (4.9). $\lambda_t$, which ranges in [0,1], is a time-dependent discount factor introduced for forgetting the effect of old values. Observe that, for values of $\lambda_t < 1$, the influence of old data decreases progressively, so that they are forgotten along time. This forgetting effect is attenuated when $\lambda_t$ approaches 1: in this case, old and new data have the same influence in the sum, and equation (4.17) becomes equal to equation (4.9). As learning proceeds and data values become more stable, forgetting them is no more required and $\lambda_t$ can be made to progressively approach 1 to reach convergence.

Using one such time-discounted sum is the strategy followed in [78] to include a forgetting in the calculation of the averages in their online EM approach [1]. However,

[1] In [78] time-discounted weighted average, instead of sums, are defined by normalizing the

we argue that such forgetting is only justified when samples are provided in their full proportions, i.e. $w = 1$, since when samples are provided in a proportion less than 1, the formula (4.17) carries out a forgetting larger than the new information provided, producing an unlearning effect.

For instance, when the proportion of a sample provided for the updating is null, i.e. $w_t = 0$, equation (4.17) is

$$[[f]]_t = \lambda_t \, [[f]]_{t-1} \,, \tag{4.18}$$

showing that the accumulator will wrongly decay to 0 when no new information is provided.

Therefore, we need to devise a forgetting formula that only forgets past experiences in the same proportion of the new experiences provided. To address this problem, we use the sampling density information to modify the updating formula (4.17). The idea is that the new updating formula should produce a forgetting of old estimations as long as new information is provided, and not just as a function of time. Since the proportion $w_t$ is indeed a measure of how much information is provided, we will regulate the forgetting according to this proportion.

For the sake of formalization, we introduce the *mass-discounted sum* $\{f\}_t$, for which the *mass-discounted mean* of $f$ will be obtained,

$$\bar{f}_t = \frac{\{f\}_t}{\{1\}_t}. \tag{4.19}$$

The definition of $\{f(x)\}_t$ is made according to the following modification of (4.17),

$$\{f\}_t = \Upsilon_t \left( \{f\}_{t-1}, w_t \right) = \Lambda_t(w_t)\{f\}_{t-1} + \Omega_t(w_t) f_t, \tag{4.20}$$

sum $[[f]]_{t,i} = \sum_{\tau=1}^{t} \left( \prod_{s=\tau+1}^{t} \lambda_s \right) f_{\tau,i} w_{\tau,i}$, whose incremental version is (4.17), with a factor $\eta_T = \left( \sum_{t=1}^{T} \prod_{s=t+1}^{T} \lambda_s \right)^{-1}$. When using these normalized sums in the expression of their estimations involved in the M step, all these $\eta_T$ cancel out, making the time-discounted weighted average equivalent to the average calculated from the cumulative sums.

where $w_t$ denotes the proportion of the sample $f_t$ used for the updating, $\Upsilon_t(f, w)$ is a two parameters function defined as shown using the two auxiliary single parameter functions $\Lambda_t(w)$ and $\Omega_t(w)$, which we have to define appropriately so that the forgetting effect depends on the weight $w_t$ used in the updating step. To determine these functions, we must first specify the conditions they must fulfil. Of course, we expect that when performing a complete update, that is, when $w_t = 1$, the effect of updating (4.20) is equivalent to that of (4.17). Therefore,

$$\Upsilon_t\left(\{f\}_{t-1}, 1\right) = \lambda_t \{f\}_{t-1} + f_t. \tag{4.21}$$

Noting that this equation must hold for any assignment of the independent values $\{f\}_{t-1}$ and $f_t$, we must have,

$$\Lambda_t(1) = \lambda_t, \tag{4.22}$$

$$\Omega_t(1) = 1. \tag{4.23}$$

To completely determine these two functions, we need to establish their behaviour for arbitrary values of $w$. For consistency, if at time $t$ we perform an update with weight $w = w_1 + w_2$, the result should be the same as if we apply two consecutive updates, both at time $t$, with weights $w_1$ and $w_2$, respectively,

$$\Upsilon_t\left(\{f\}_{t-1}, (w_1 + w_2)\right) = \Upsilon_t\left(\Upsilon_t\left(\{f\}_{t-1}, w_1\right), w_2\right). \tag{4.24}$$

Equation (4.24) is a *functional equation* for $\Upsilon_t$, and it defines implicitly corresponding functional equations for its constitutive function $\Lambda_t$ and $\Omega_t$. Developing the left-hand side of (4.24) using (4.20) gives,

$$\Upsilon_t\left(\{f\}_{t-1}, (w_1 + w_2)\right) = \Lambda_t(w_1 + w_2)\{f\}_{t-1} + \Omega_t(w_1 + w_2) f_t. \tag{4.25}$$

Now, developing the right-hand side of (4.24),

$$\begin{aligned}
\Upsilon_t\left(\Upsilon_t\left(\{f\}_{t-1}, w_1\right), w_2\right) &= \Lambda_t(w_2)\Upsilon_t\left(\{f\}_{t-1}, w_1\right) + \Omega_t(w_2) f_t \\
&= \Lambda_t(w_2)\left(\Lambda_t(w_1)\{f\}_{t-1} + \Omega_t(w_1) f_t\right) + \Omega_t(w_2) f_t \\
&= \Lambda_t(w_2)\Lambda_t(w_1)\{f\}_{t-1} + \left(\Lambda_t(w_2)\Omega_t(w_1) + \Omega_t(w_2)\right) f_t,
\end{aligned}$$

so that equation (4.24) reduces to,

$$\begin{aligned} \Lambda_t\left(w_1+w_2\right)\{f\}_{t-1} &+ \Omega_t\left(w_1+w_2\right) f_t \\ = \Lambda_t(w_2)\Lambda_t(w_1)\{f\}_{t-1} &+ \left(\Lambda_t(w_2)\Omega_t(w_1)+\Omega_t(w_2)\right) f_t. \end{aligned}$$

As before, we conclude that the factors of the independent values $\{f\}_{t-1}$ and $f_t$ must be the same at both sides,

$$\Lambda_t\left(w_1+w_2\right) = \Lambda_t(w_2)\Lambda_t(w_1) \tag{4.26}$$

$$\Omega_t\left(w_1+w_2\right) = \Lambda_t(w_2)\Omega_t(w_1)+\Omega_t(w_2). \tag{4.27}$$

The solution of the functional equation (4.26) is well known to be of the form $\Lambda_t(w) = A^w$, and the value $A$ can be determined using the condition (4.22),

$$A = \Lambda_t(1) = \lambda_t. \tag{4.28}$$

Hence,

$$\Lambda_t(w) = \lambda_t^w. \tag{4.29}$$

Now, using this result in (4.27), and making $w_1 + w_2 = 1$, the equation (4.27) leads to,

$$\Omega_t(1) = \Omega_t(w_1+w_2) = \lambda_t^{w_2}\Omega_t(w_1)+\Omega_t(w_2) = 1, \tag{4.30}$$

and also,

$$\Omega_t(1) = \Omega_t(w_2+w_1) = \lambda_t^{w_1}\Omega_t(w_2)+\Omega_t(w_1) = 1. \tag{4.31}$$

Solving the system of two equations established by (4.30) and (4.31) we get

$$\Omega_t(w_1) = \frac{1-\lambda_t^{w_1}}{1-\lambda_t}. \tag{4.32}$$

This completes equation (4.20) and allows us to write its sought form, now including the reference to the Gaussian, $i$,

$$\{f\}_{t,i} = \lambda_t^{w_{t,i}}\{f\}_{t-1,i} + \frac{1-\lambda_t^{w_{t,i}}}{1-\lambda_t} f_{t,i}. \tag{4.33}$$

With the updating formula (4.33) [1] , the power $w_{t,i}$ prevents undesired changes in the parameters of the Gaussians which are not responsible of generating the observed values. Thus, if we make $w_{t,i} = 0$ in (4.33), what we get is

$$\{f\}_{t,i} = \{f\}_{t-1,i}, \tag{4.34}$$

so that the values of the statistics of the inactive units remain unchanged. On the other hand, in the limit case of $w_{t,i} = 1$, corresponding to a full activation of unit $i$, the effect of the new updating formula is the same as before,

$$\{f\}_{t,i} = \lambda_t\{f\}_{t-1,i} + f_{t,i}. \tag{4.35}$$

To evaluate the forgetting effect for activations in (0,1), we perform a simple test using three different sequences of samples, and their corresponding activations, of the form $\text{seq}_i = (\{w_0, f_0\}, \ldots, \{w_t, f_t\})$,

$$\begin{aligned}
\text{seq}_1 &= (\{1,10\},\{0.2,1\},\{0.2,1\},\{0.2,1\},\{0.2,1\},\{0.2,1\},\{1,1\}),\\
\text{seq}_2 &= (\{1,10\},\{0.5,1\},\{0.5,1\},\{1,1\}),\\
\text{seq}_3 &= (\{1,10\},\{1,1\},\{1,1\}).
\end{aligned}$$

All the sequences provide the same cumulative quantity of 12, with an average, without forgetting, of 4. The difference among the sequences is that the middle terms provide the same amount of information split in different proportions. For instance, in sequence $\text{seq}_1$, the amount 1 is split in five time steps, each with a proportion of $w = 0.2$.

[1] Note that, the expression $(1-\lambda_t^{w_{t,i}})/(1-\lambda_t)$ in the second term in (4.33) is undetermined when $\lambda_t = 1$. In order to determine its value when $\lambda_t = 1$, we apply the l'Hopital's rule, deriving the numerator and the denominator with respect to $\lambda$. This yields $(-w_{t,i}\lambda_t^{w_{t,i}-1})/(-1)$, which, for $\lambda_t = 1$, gives the value that should be used for this expression in such a case, $(-w_{t,i}1^{w_{t,i}-1})/(-1) = w_{t,i}$.

We compare the averages of the samples provided by each sequence, calculated using the time-discounted sums (4.17), and using the mass-discounted sums (4.33). The forgetting factor $\lambda$ is set to $\lambda = 0.9$ in both cases.

For the case of the average calculated using the mass-discounted sums, we obtain, for all the sequences, the same average of $\bar{f} = 3.69$. This corroborates that the forgetting is based on the quantity of information provided, rather than on time, since it does not matter on how many time steps the middle quantity is split. In contrast, the averages calculated using a time-discounted sum (4.17) are,

$$\begin{aligned}\bar{f}_{\text{seq}_1} &= 3.11,\\ \bar{f}_{\text{seq}_2} &= 3.54,\\ \bar{f}_{\text{seq}_3} &= 3.69,\end{aligned}$$

reflecting that forgetting depends on time: the larger the number of time steps in which the middle amount 1 is split, the larger the forgetting effect. The undesired forgetting effect is evidenced more clearly if we split the middle quantity of 1 in 100 time-steps, each with a proportion of 0.01. In this case, the average using space-discounted sums is again $\bar{f} = 3.69$, while the average calculated using time-discounted sums is merely $\bar{f} = 1.00$, evidencing a strong time-dependent forgetting. In general, the average calculated with time-discounted sums would decrease to one as the amount 1 is split in more time steps.

We will use the mass-discounted cumulative sums for the calculation of the parameters involved in the M step of the online EM,

$$\alpha_{t,i} = \frac{\{1\}_{t,i}}{\sum_{j=1}^{K} \{1\}_{t,j}}, \tag{4.36}$$

$$\mu_{t,i} = \frac{\{\mathbf{x}\}_{t,i}}{\{1\}_{t,i}}, \tag{4.37}$$

and

$$\Sigma_{t,i} = \frac{\{\mathbf{x}\mathbf{x}'\}_{t,i}}{\{1\}_{t,i}} - \mu_{t,i}\mu_{t,i}'. \tag{4.38}$$

## 4.4 Function Approximation using Probability Density Estimations

In this section we will present an online memory-free function approximation using a probability density estimation, represented with a GMM. We make the approach non-parametric by permitting Gaussian generation on demand for a better approximation. We assume that there is an arbitrary unknown target function $f(x)$ defined in a $D$-dimensional continuous domain $X \subset \Re^D$, that should be approximated, from which we can only observe its values at particular points obtained sequentially, $(x_t, y_t)$, where the output $y_t = f(x_t)$ is supposed to be uni-dimensional [1] , and $t$ accounts for the $t$-th time step.

In this case, the samples provided to the density model are of the form $(x_t, y_t)$, and the joint probability distribution estimated with a GMM is

$$p(x, y; \Theta) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(x, y; \mu_i, \Sigma_i). \tag{4.39}$$

From this probability distribution, we can obtain an approximation of the target function as

$$f(x) \approx E\left[y|x; \Theta\right] = \mu(y|x; \Theta). \tag{4.40}$$

To compute this, we must first obtain the distribution $p(y|x; \Theta)$. In the sequel we omit the reference to the parameters $\Theta$ for clarity in the formulas. Decomposing the covariances $\Sigma_i$ and means $\mu_i$ in the following way:

$$\mu_i = \begin{pmatrix} \mu_i^x \\ \mu_i^y \end{pmatrix}, \tag{4.41}$$

$$\Sigma_i = \begin{pmatrix} \Sigma_i^{xx} & \Sigma_i^{xy} \\ \Sigma_i^{yx} & \Sigma_i^{yy} \end{pmatrix}, \tag{4.42}$$

[1] We suppose a uni-dimensional output since it suffices for the type of functions we will use. However, all the formulations that follow can be easily extended to the multidimensional case.

the probability distribution of $y$, for the given input $x$, can then be expressed as [1] ,

$$p(y|x) = \sum_{i=1}^{K} \beta_i(x) \mathcal{N}\left(y; \mu_i(y|x), \sigma_i(y)\right), \tag{4.43}$$

where

$$\mu_i(y|x) = \mu_i^y + \Sigma_i^{yx} \left(\Sigma_i^{xx}\right)^{-1} \left(x - \mu_i^x\right), \tag{4.44}$$

$$\sigma_i^2(y) = \Sigma_i^{yy} - \Sigma_i^{yx} \left(\Sigma_i^{xx}\right)^{-1} \Sigma_i^{xy}, \tag{4.45}$$

and

$$\beta_i(x) = \frac{\alpha_i \mathcal{N}(x; \mu_i^x, \Sigma_i^{xx})}{\sum_{j=1}^{K} \alpha_j \mathcal{N}(x; \mu_j^x, \Sigma_j^{xx})}. \tag{4.46}$$

From (4.43) we can obtain the estimation of $f(x)$ as the conditional mean, $\mu(y|x)$, of the mixture at a point $x$ as

$$f(x) \approx \mu(y|x) = \sum_{i=1}^{K} \beta_i(x) \mu_i(y|x). \tag{4.47}$$

### 4.4.1 Gaussian Management

Since the main purpose of our GMM is to approximate a target function $f$ to any desired precision, the generation of new Gaussians is principally driven by the need to better approximate the set of observed $y$ values, more than better approximate the density of samples in the joint space. There are many aspects to take into account in Gaussian generation to avoid large distortion in the approximation when new Gaussians are added, and to permit an improvement in the approximation in the short term. Here we propose one alternative that, even though simple, permits to improve the approximation fairly fast and with little distortion.

[1] In general, the probability density model results very useful in the approximation of stochastic functions since it provides an estimation of the probability distribution of the output variable $y$ for any given input $x$.

A new Gaussian is generated when the two following conditions are satisfied: 1) The estimation error of the observed sample is larger than a predefined value $e_c$,

$$(f(x) - \mu(y|x))^2 \geq e_c, \tag{4.48}$$

and 2) the number of samples accumulated in $\{1\}_i$ for all the Gaussians with non-negligible probability density $p_i(x) = \mathcal{N}(x; \mu_i^x, \Sigma^x x_i)$, is above a confidence threshold $n_c$. The imposition of this condition prevents the generation of Gaussians when there are other Gaussians with chances of improving the approximation when more samples are experienced. We use the marginal probability $p_i(x)$, instead of the joint probability $p_i(x, y)$, since we are interested in testing the amount of inputs captured by the Gaussian in the input space: note that, a low $p_i(x, y)$ may be caused by a the non-stationarity of the target function which shifts the output $y$ far from the Gaussian mean, which makes the joint probability $p_i(x, y)$ to decrease.

The Gaussians with higher marginal probabilities are determined as,

$$\mathcal{I} = \{i | p_i(x) / \sum_{j=1}^{K} p_j(x) > \beta_c\}, \tag{4.49}$$

where the threshold for the relative marginal probability, $\beta_c$, ranges in [0,1]. Then, the minimum number of samples is obtained as,

$$n_{min} = \min_{i \in \mathcal{I}} \{1\}_i. \tag{4.50}$$

Finally, the evaluation of the second criterion is expressed as,

$$n_{min} > n_c. \tag{4.51}$$

Whenever both criteria are fulfilled, a Gaussian is generated with parameters given by,

$$\{1\}_{K+1} = 1, \tag{4.52}$$

$$\mu_{K+1}(x, y) = (x_t, y_t), \tag{4.53}$$

**Algorithm 5** Function Approximation with a GMM Probability Density Estimation

initialize the GMM with a set of $K_{ini}$ Gaussians.
**loop**
  get observation $< x, y >$
  calculate the activation $w_i$ of each Gaussian in $(x, y)$ (4.3) (E step)
  update the GMM parameters given by (4.36), (4.37), and (4.38) using (4.33) (M step)
  calculate $\mu(y|x)$ (4.47)
  calculate the approximation error $e = (y - \mu(y|x))^2$
  get the minimum number of samples $n_{min}$ (4.50)
  **if** $n_{min} < n_c$ & $e < e_c$ **then**
    generate new Gaussian (section 4.4.1)
  **end if**
**end loop**

$$\Sigma_{K+1} = C\ diag\{d_1, ..., d_D, d_y\}, \tag{4.54}$$

where $d_i$ is the total range size of the variable $i$; $D$ is the dimension of the domain; and $C$ is a positive value defining the dispersion of the new Gaussian.

### 4.4.2 Algorithm for the FA using Probability Density Estimations

The complete algorithm for the online memory-free function approximation using probability density estimations is presented in algorithm 5.

## 4.5 Variance Estimation using Probability Density Estimations

One of the most remarkable features of a joint density model is that it permits to calculate, point to point, the variance of the samples. For the case of a density model represented with a GMM, the sample variance is calculated as,

$$\sigma^2(y|x) = \sum_{i=1}^{K} \beta_i(x)(\sigma_i^2(y) + (\mu_i(y|x) - \mu(y|x))^2). \tag{4.55}$$

The value $\sigma^2(y|x)$ reflects the point to point approximation error with which the FA carried out from the GMM (4.47) approximates a given target function, and, also, the

intrinsic variability of the data when the target function is stochastic.

## 4.6 Estimating the Number of Samples in a Region

Another benefit of using a probability density estimation for FA is that we can estimate the number of samples in a region $\tilde{X} \subset X$ surrounding $x$. As done before (section 3.3.1.2), we calculate the number of samples in a region $\tilde{X}$ as,

$$\tilde{n} = n \int_{\tilde{X}} p(x)dx. \tag{4.56}$$

To simplify the calculations of the integral in (4.56), we assume that for regions $\tilde{X}$ in which the density of samples is homogeneous the value for the probability density estimation is constant, and equal to $p(x)$, which permits to estimate the number of samples as,

$$\tilde{n} \approx \tilde{V} n \, p(x; \Theta), \tag{4.57}$$

where

$$n = \sum_{i=1}^{K} \{1\}_i \tag{4.58}$$

is the total number of samples contemplated in the density model,

$$p(x; \Theta) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(x; \mu_i^x, \Sigma_i^{xx}) \tag{4.59}$$

is the probability density estimation in $X$ provided by the GMM, and $\tilde{V}$ is the volume of the region $\tilde{X}$. This number of samples will be used to credit the estimations $\mu(y|x)$ and $\sigma^2(y|x)$.

## 4.7 $Q$-Learning using Probability Density Estimations

In this section we present a method for FA in $Q$-Learning in continuous state and action spaces using probability density estimations represented with a GMM (referenced from

now on as the GMMRL method). In this case, the samples provided to the GMM are of the form $(s, a, q)$, where $s$ is an observed state, $a$ is the executed action at that state, and $q$ is the estimated cumulative reward calculated as,

$$q(s,a) = r(s,a) + \gamma \max_a(\hat{Q}(s',a)), \tag{4.60}$$

where $r(s,a)$ is the immediate reward obtained after executing action $a$ in state $s$, $s'$ is the state observed after the action execution, and $\hat{Q}(s',a)$ is the approximation of the action-value function at $(s',a)$ calculated from the GMM, $\hat{Q}(s',a) = \mu(q|s',a)$ (4.47). To compute the value $q(s,a)$ we need to solve the maximization problem $\max_a(\hat{Q}(s',a))$. The analytical solution to this problem is complicated, but an approximated value can be obtained by numerical techniques. In our implementation, we adopt the simple strategy of computing the values $\hat{Q}(s',a)$ for a finite number of actions, and then taking the value of the action that provides the largest $Q$ as the approximated maximum.

### 4.7.1 Exploration-Exploitation Strategy

From the action-value function approximation it is possible to exploit what has been learned so far by taking the action with highest $\hat{Q}(s,a)$ in every situation $s$ the system comes across,

$$\begin{aligned} a &= \pi(s) \\ &= \operatorname*{argmax}_{a'} \hat{Q}(s,a'). \end{aligned} \tag{4.61}$$

However, for the learning of the optimal action values, exploratory actions should also be taken. This is the exploration-exploitation trade-off presented in section 2.2. A simple exploration strategy is the plain $\epsilon$-greedy policy, that takes actions from the greedy policy (4.61) with probability $1-\epsilon$, and exploratory actions with probability $\epsilon$. A more sophisticated strategy is the Boltzmann exploration [81] which defines the probability of executing an action according the values $\hat{Q}(s,a)$ of each evaluated action,

$$p(a|s) = \frac{e^{\hat{Q}(s,a)/T}}{\sum_{a'} e^{\hat{Q}(s,a')/T}}, \tag{4.62}$$

where $T$ is a positive parameter, called the temperature, that is made to decrease with time, and which regulates the importance of the values $\hat{Q}(s,a)$ in the probability: the higher the temperature the lower the importance of $\hat{Q}(s,a)$. A high temperature is preferred at the beginning of the learning process to decrease the importance of still poor estimations of $\hat{Q}(s,a)$.

Another exploration strategy is proposed in [39], which uses the uncertainty in the estimation of $\hat{Q}(s,a)$. In this strategy the upper bound of the confidence interval for the unknown value $Q(s,a)$ is used as the inferred value $\hat{Q}(s,a)$ in the evaluation of $(s,a)$. This favours the exploration of actions that have some chances of getting a large $q(s,a)$ in the experienced states. With this strategy, actions for which the estimation of $Q(s,a)$ is uncertain will have large confidence intervals and may be preferred in detriment of actions with smaller upper bounds for $Q(s,a)$, hence contributing to reduce the uncertainty at less explored regions.

### Our proposal

Our proposal for exploration is similar to that in [39], as it is also based on the uncertainties of the estimations $\hat{Q}(s,a)$. However, instead of using a deterministic strategy of using the upper bound of the confidence interval for $Q(s,a)$, our strategy allows the system to predict a $\hat{Q}(s,a)$ at random, with a probability distribution that corresponds to the current knowledge we have about $Q(s,a)$.

We know from Statistics that the variable defined as

$$\frac{\bar{Y}-\mu}{\sqrt{S^2/n}}, \tag{4.63}$$

has a $t$-distribution with $n-1$ degrees of freedom [22], where $\bar{Y}$ is the sample mean calculated from a set of observations as in (3.5), $\mu$ is the unknown mean, $S^2$ is the sample variance, estimated as in (3.6), and $n$ is the number of samples used for the estimations. Replacing the quantities in (4.63) for their corresponding counterparts in the estimation of the mean value $Q(s,a)$ we have

$$\frac{\hat{Q}(s,a)-Q(s,a)}{\sqrt{S^2(s,a)/n(s,a)}} \sim t(n(s,a)-1), \tag{4.64}$$

where $Q(s,a)$ is the actual (unknown) expected value of $q(s,a)$, $\hat{Q}(s,a) = \mu(q|s,a)$ is the point-dependent sample mean (4.47), $S^2(s,a) = \sigma^2(q|s,a)$ is the point-dependent sample variance (4.55), and $n(s,a)$ is the number of samples in the surrounding of $(s,a)$ estimated as in (4.57). Then, our strategy for exploration consists in selecting a value $t_{rand}$ of this random variable according to the $t$-distribution with $n(s,a)-1$ degrees of freedom, and finding the corresponding $\hat{Q}_{rand}(s,a)$ as,

$$\hat{Q}_{rand}(s,a) = \hat{Q}(s,a) - t_{rand}\frac{S(s,a)}{\sqrt{n(s,a)}}. \tag{4.65}$$

Once a $\hat{Q}_{rand}(s,a)$ value has been assigned to every evaluated action in the current situation, the action to be executed is selected using the greedy policy (4.61),

$$a = \underset{a'}{\operatorname{argmax}}\, \hat{Q}_{rand}(s,a'). \tag{4.66}$$

This point-dependent exploration strategy automatically regulates the exploration according to the needs of each precise region around $(s,a)$. More uncertain estimations of $Q(s,a)$ will have higher variance $S^2(s,a)$, and $\hat{Q}_{rand}(s,a)$ will have a higher range of variation. This favours the exploration of regions with higher uncertainties in $\hat{Q}(s,a)$. On the other hand, less explored regions will have low values of $n(s,a) \approx \tilde{V}\, n\, p(s,a)$, which also increases the range of variation of $\hat{Q}_{rand}(s,a)$. Hence, equation (4.65) also favours the exploration of less sampled regions. Finally, this exploration strategy permits to exploit more certain knowledge so as to better conduct the exploration to interesting regions. Note that, the exploitation of more certain actions would not be possible by just using the upper bound of the confidence interval for $Q(s,a)$, as carried out in the work [39].

## 4.8 Performance Evaluation

To evaluate the performance of the of GMMRL method we use the classical benchmark problem of swinging up and stabilizing an inverted pendulum with limited torque [30]. The task consists in swinging the pendulum until reaching the upright position and then stay there indefinitely. The optimal policy for this problem is not trivial since, due to the limited torques available, the controller has to swing the pendulum several

times until its kinetic energy is large enough to overcome the load torque and reach the upright position. Thanks to its simplicity but still challenging control task, this benchmark is widely used to test the performance of state of the art methods for function approximation in RL [26, 28, 70, 73]. The scenario for the inverted pendulum is depicted in figure 4.1.

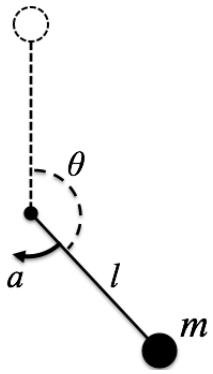



Figure 4.1: Inverted pendulum benchmark.

The dynamics of the inverted pendulum is modelled as,

$$ml^2\ddot{\theta} = -\mu\dot{\theta} + mgl\sin\theta + a, \tag{4.67}$$

where $\theta$ is the angular position, $l$ is the length of the link, $m$ the mass of pendulum (assumed to be concentrated at the distal end with respect to the joint), $g$ the gravity constant, $\mu$ the friction factor, and $a$ the input torque. In our simulations we use the values $l = m = 1$, $g = 9.8$, $\mu = 0.01$, and an input torque $a$ ranging in $[-5, 5]$. For the simulation we use the Euler method with a differential for time of $dt = 0.001$ seconds and an actuation interval of 0.1 seconds.

The state-action space is three-dimensional, configured by the angular position $\theta$, the angular velocity $\dot{\theta}$, and the action. We take advantage of the symmetry of the problem by identifying states with inverted angular position and velocity: $(\theta, \dot{\theta}, a) \sim (-\theta, -\dot{\theta}, -a)$. As the reward signal we simply take minus the absolute value of the angle of the pendulum from its top position: $r(\theta, \dot{\theta}) = -|\theta|$ which ranges in the interval $[-\pi, 0]$. In this application, the probability density model is defined in the four-dimensional joint space $\mathbf{x} = (\theta, \dot{\theta}, a, q)$.

To solve the maximization $\max_a(\hat{Q}(s', a))$ we use a set of actions obtained from a uniform sampling of the action variable with period 0.1. The discount coefficient $\gamma$ is set to 0.85.

### Reference Methods

Three performance comparisons are carried out.

The first one is by contrasting the results of the FA with a GMM using the two different discounted sums in the online EM: the mass-discounted sum formula (4.33), and the time-discounted sum formula (4.17). This is done to show the gain in considering a forgetting based on the new information provided, rather than on time.

The second is done by comparing our results with those of the Neural Fitted Q-Iteration [71] (section 2.3.4), as it is one of the most remarkable methods in continuous function approximation for $Q$-Learning that uses the benchmark of the inverted pendulum.

Finally, the third comparison is done with the results of [77], since their approach shares many similarities with ours. They use a normalized Gaussian network (NGnet) [52] to approximate the action-value function $Q(s, a)$, and another one to approximate the policy function, in an actor-critic approach. The function representation of the NGnet is,

$$y = \sum_{i=1}^{K} \left( \frac{\mathcal{N}_i(x; \mu_i, \Sigma_i)}{\sum_{j=1}^{K} \mathcal{N}_j(x; \mu_i, \Sigma_i)} \right) (W_i x + b_i)\,, \tag{4.68}$$

where $\mathcal{N}_i$ is a Gaussian, defined in the domain, and $\{W_i, b_i\}$ are the parameters of an hyperplane. The approach proposed in [77] consists in finding the set of parameters $\{\mu_i, \Sigma_i, W_i, b_i\}$, $i = 1, \ldots, K$, to better approximate the action-value function, using an online EM version [78] with a time-dependent forgetting factor. The approach is also non-parametric since it implements a unit generation strategy. Note that, in their approach the approximation of the target function carried out by each unit, is explicitly represented by the hyperplane with parameters $\{W_i, b_i\}$.

### Initialization and Settings

The common settings of our system consist in the following. We provide the system with 10 initial Gaussians. The elements of the mean $\mu_i$ of the mixture component $i$ are selected randomly in the range of the corresponding variable, except for the $q$ variable that is initialized to the maximum possible value to favour exploration of unvisited regions [1]. The initial covariance matrices $\Sigma_i$ are diagonal and the variance for each variable is set to 20% of the total span of its range. Each Gaussian is initialized with an accumulated number of samples $\{1\}_i = 0.1$. This small value makes the component $i$ to have little influence in the estimation while there is no, or little, updating. Note that, the rest of the accumulators are derived from the Gaussian mean and covariance, and the accumulator $\{1\}_i$. For instance, the initial accumulator for the sampled points $(s, a, q)$ would be $\{(s, a, q)\}_i = \mu(s, a, q)_i\{1\}_i$. The discount factor $\lambda_t$ for the computation of the discounted sums takes values from the equation,

$$\lambda_t = 1 - 1/(at + b), \tag{4.69}$$

where $a$ determines its growth rate toward 1, and $b$ fixes the initial value of $\lambda_t$, and regulates the influence of $a$. In our experiments we set $a = 0.001$, and the value of $b$ depends on whether the updating formula is using mass-discounted sums (4.33) or time-discounted sums (4.17). For the case of using mass-discounted sums the value $b$ is set to $b = 10$, and when the time-discounted sums are used, $b$ is set to $b = 1000$, to compensate for the effect of the exponent $w_{t,i} < 1$ in sparsely sampled regions. Finally, the thresholds for the approximation error, for the minimum number of samples, and for the relative marginal probability, are set to $e_c = 0.025 * \text{rang}_q$, $n_c = 200$, and $\beta_c = 0.1$, respectively, where $\text{rang}_q$ is the total span of the $q$ variable.

### Results: Time-Discounted Forgetting versus Mass-Discounted Forgetting

For the comparison between the performance of the FA with a GMM using time-discounted sums and mass-discounted sums, we carried out the experiments using episodes of 500 iterations. At the beginning of each episode, the pendulum is placed

[1] The maximum value for the $q$ variable can be calculated from equation (4.60) as a sum of an infinite series, instantiating the reward, at each term, to the maximum possible value.

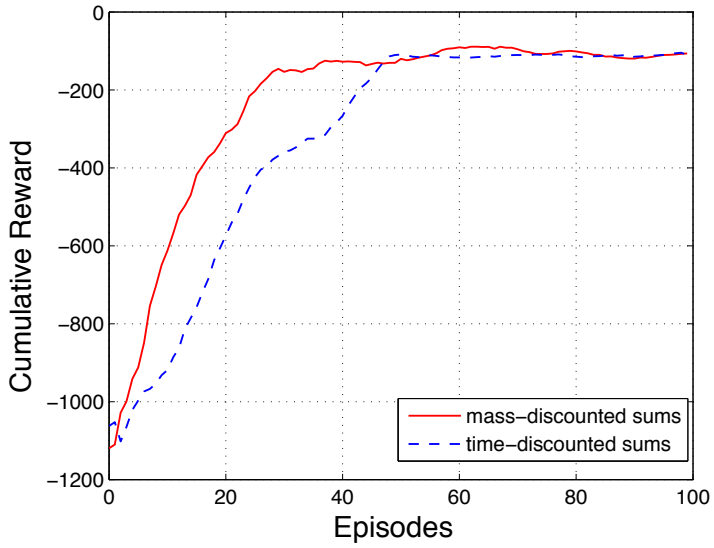



Figure 4.2: Comparison between the performance of the GMMRL approach using time-discounted sums and using mass-discounted sums.

in the hang-down position. At the end of each episode, a test of 500 iterations is performed exploiting the policy learned so far. As the result of the test we take the sum of the rewards obtained at each iteration. For each experiment we calculate the average of 15 independent runs of 100 test episodes.

Figure 4.2 shows the results of the comparison. The results indicate that the proposal using the mass-discounted sums (in red) achieves convergence significantly faster than when using time-discounted sums (in blue). The worst performance of the method using time-discounted sums is produced by the undesired forgetting of the experiences far from the currently sampled region. This problem is accentuated at early stages of the learning, when the sampling is very variable due to a large variations in the exploration strategy produced by policy adaptations. This effect is greatly reduced when the mass-discounted sums are used, as shown by the red plot.

Figures 4.3 and 4.4 illustrate two projections of the Gaussians of a typical GMM obtained after a training episode with mass-discounted sums. It can be seen that they are not equally distributed along the whole configuration space, but concentrate in the most common trajectories of the system, what constitutes an efficient use of resources.

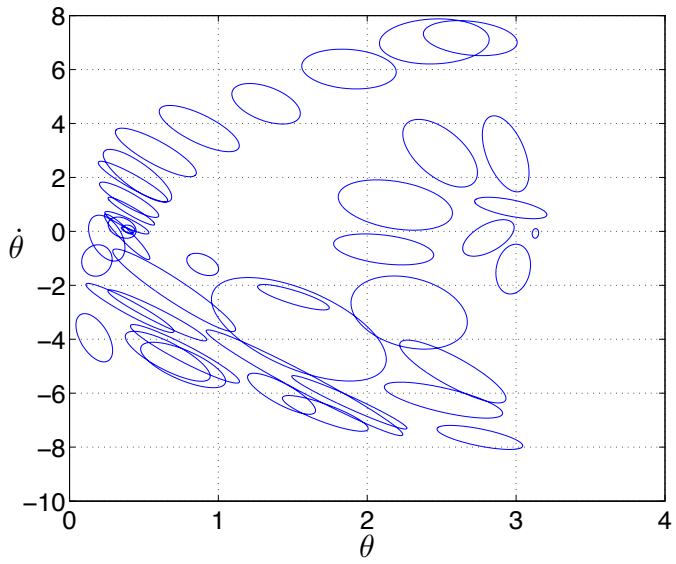



Figure 4.3: Distribution of Gaussians in a projection of the joint space to the state space.

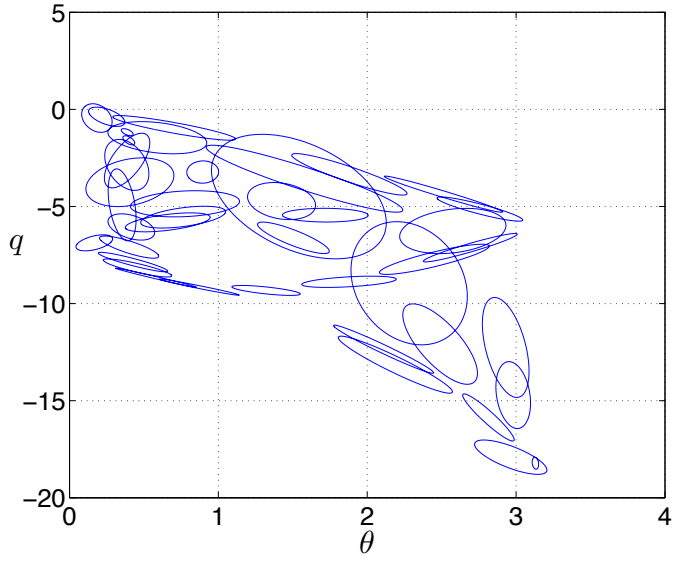



Figure 4.4: Distribution of Gaussians in a projection of the joint space to the $(\theta, q)$ space.

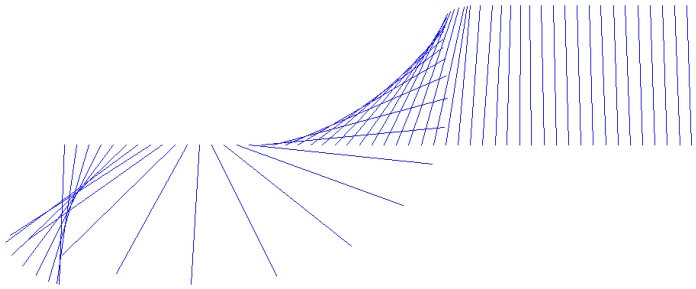


Figure 4.5: A stroboscopic sequence obtained from placing the pendulum in the hang-down position.

Finally, to illustrate the control reached we present in figure 4.5 a control from the position of the pendulum hanging down.

### Results: Comparison with Neural Fitted Q-Iteration

Since the NFQ strategy [71] works in a batch mode, we compare the results by taking into account the total number of times the system needs to be updated to achieve the control. Riedmiller reports that the swing-up and balance task required 100 iterations of the NFQ algorithm, each one requiring 1000 epochs of batch learning with the Rprop learning algorithm to train the neural net with an unspecified number $D$ of samples. This gives a total of $100000 \times D$ sample updates. In our approach using mass-discounted sums (red plot in figure 4.2), good control is obtained after approximately $28 \times 500 = 14000$ updates, which is significantly better. This proves that our approach provides an efficient strategy for FA in $Q$-Learning. However, for a rigorous comparison with NFQ, we would need to test the system in the same real platform used in the work [71].

### Results: Comparison with NGnet

After the comparison with [71], we evaluate more exhaustively the performance of the FA with a GMM comparing its results with those of the NGnet presented in [77]. To do this, we use the same setting for the experiments of that work, using episodes of 70 iterations. After each episode the pendulum is randomly placed inside an arch centred

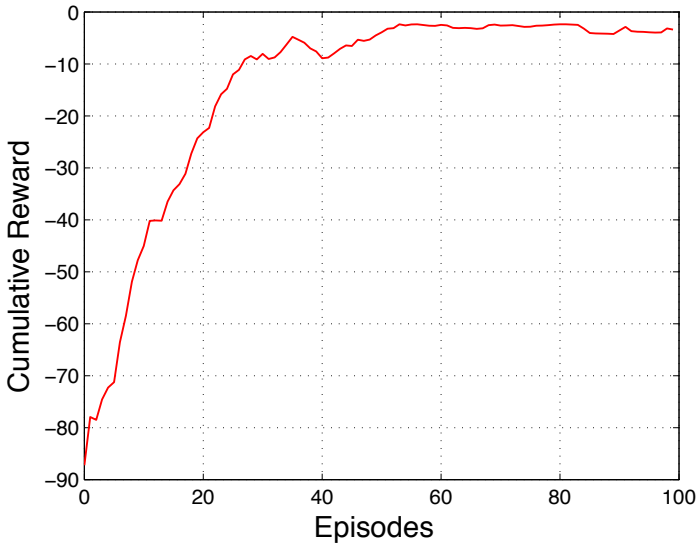



Figure 4.6: Results of the experiments with mass-discounted updating in the inverted pendulum task, using the same experiments set-up as in [77].

in the upright position. The length of the arch is steadily incremented with each episode. The performance is evaluated computing the average for 15 experiments of 100 episodes each, of the total accumulated reward per episode obtained by exploiting the policy learned so far.

The result of the GMMRL approach is shown in figure 4.6. With the GMMRL the convergence takes place in around 28 episodes, with an average number of Gaussians of 40. In [77], the reported number of episodes to reach convergence is 40, and the number of components of the NGnet needed to approximate the critic is 108 (for a fair comparison, the components of the actor of the NGnet are not taken into account). We would like to note that our best result in the 15 runs using mass-discounted sums converged in 11 espisodes with 29 Gaussians.

## 4.9 Conclusions

We proposed a new approach for function approximation in $Q$-Learning for continuous state-action spaces, in which a Gaussian Mixture Model, that estimates the probability density in the joint state-action-$q$value space, is used for function approximation. From

this joint distribution we can obtain, not just the expected value of $q$ for a given state and action, but a full probability distribution $p(q|s,a)$ of the $q$ variable at each given $(s,a)$, that is used to define a directed exploration-exploitation strategy.

As a further benefit, from the density estimation in the joint space we can also obtain the sample density in the state-action space. This information is used to remedy the problem of biased sampling inherent to Reinforcement Learning. For this, we modified the incremental updating rule of an on-line EM algorithm in order to avoid forgetting data of less frequently sampled regions, even when exploration is repetitively done near the goal configurations.

Tests performed on a classical RL problem, the swing-up and balance of an inverted pendulum, show that our approach improves the results of previous works. The comparison between our basic approach, using the time-discounted updating formula (4.17), and the proposed formula of mass-discounted updating (4.33), shows that the approach is effective in reducing the perturbing effect of biased sampling.

Finally, the point-dependent estimations of the mean, variance, and number of samples obtained from the joint density model will be used for the definition of a DFA approach for FA in $Q$-Learning. This is the main thread of next chapter.

# Chapter 5

# $Q$-Learning with a Degenerate Function Approximation

## 5.1 Introduction

This chapter presents the method that implements the main idea of this thesis: the use of competing function approximators to face the problem of generalization in RL. Our principal argument is based on the fact that trying in parallel different ways of approximating the value function increases the chances of having a good approximator among the competing ones, increasing, in turn, the chances of generalization.

The idea of a competitive strategy for function approximation was implemented in chapter 3, through a new approach called degenerate function approximation (DFA). The DFA approach tries in parallel different function approximations, named as *competitors* $\Phi_i(x, \xi_i)$, which are defined in different regions of the target function domain. The DFA approach selects, at each point $x$, the competitor with the best estimated quality in the approximation among all the competitors. To perform this selection, we associate to each competitor a relevance function (3.21), which quantifies the approximation quality of the competitor at each point. The relevance of a competitor in a point is calculated taking into account the accuracy of its approximation, through a point-dependent sample variance $S_i^2(x, \tau_i)$, and the confidence in this approximation, through a point-dependent estimation of the sample density in the input space $p_i(x, \varsigma_i)$.

In chapter 3 we also analysed the behaviour of the DFA system when samples are provided with a biased distribution, which is a characteristic problem of FA in RL, and

saw that, in order to produce a correct selection of the competitor under biased sampling, the relevance of a competitor should be represented precisely, providing a good point-dependent estimation of the quality in the approximation. This is because, under biased sampling, a constant-valued relevance for all the competitor's domain reflects mainly the approximation quality at the most frequently sampled regions, providing a weak indication of this quality at sparsely sampled ones.

In chapter 4 we devise a new approach for FA in RL based on a probability density estimation. From the probability density model, we can easily derive, through the conditional probabilities, a point-dependent estimation of the sample mean (4.47), as well as a point-dependent estimation of the sample variance (4.55). In addition, from the density model we can estimate the number of samples involved in the estimation of the mean and variance at each point (4.57). All these estimations are easily updated using a novel online, memory-free, version of the Expectation-Maximization algorithm (section 4.3.1).

In this chapter we bring together the GMM approach of chapter 4 and the DFA approach of chapter 3, to define a method for FA in $Q$-Learning using a competitive strategy. This is implemented by embedding a GMM in each competitor. This permits to estimate the quality in the approximation at a given point, calculating the relevance using the point-dependent estimation of the variance (4.55), and the point-dependent estimation of the number of samples (4.57). Even more, the GMM permits to define a multi-parametric competitor function through the point-dependent sample mean (4.47).

The organization of this chapter is as follows. Next section explains the outlines of a FA approach consisting in a DFA using probability density estimation. Then, in section 5.3, the mechanisms for FA in $Q$-Learning using the DFA are presented, to lately test these mechanisms in three different difficulty tasks: the swing-up and balancing of an inverted pendulum with limited torque, the mountain-car, and the cart-pole balancing.

## 5.2 Degenerate Function Approximation using Probability Density Estimations

Before entering into the details of how a competing strategy implemented with the DFA approach is used for function approximation in RL, we instantiate the DFA approach

using the estimations derived from the probability density estimation in the joint space. To this end, we embed in each competitor $\Phi_i$, a GMM defined in the domain of the competitor, $X_i$. We refer to such a GMM as $p_i(x, y; \Theta_i)$, where $\Theta_i$ are the GMM parameters $\Theta_i = \{\{\alpha_{i,1}, \mu_{i,1}, \Sigma_{i,1}\}, ..., \{\alpha_{i,K}, \mu_{i,K}, \Sigma_{i,K}\}\}$. From this model we define the competitor function as (4.47),

$$\Phi_i(x, \xi_i) = \mu_i(y|x; \Theta_i), \tag{5.1}$$

where $\mu_i(y|x; \Theta_i)$ is obtained as the expected value of the conditional probability $p_i(y|x; \Theta_i)$ (4.43). In turn, the estimated relevance function (3.21), rewritten here for convenience,

$$\Gamma_i(x, \nu_i) \approx \frac{\chi^2_\alpha(h_i)}{h_i \, S_i^2(x, \tau_i)}, \tag{5.2}$$

where $h_i = \tilde{V}_i \, n_i \, p_i(x, \varsigma_i) - 1$, is defined using the estimation of the variance as in (4.55),

$$S_i^2(x, \tau_i) = \sigma_i^2(y|x; \Theta_i), \tag{5.3}$$

and the estimation of the number of samples as in (4.57),

$$\tilde{n}_i \approx \tilde{V}_i \, n_i \, p_i(x; \Theta_i), \tag{5.4}$$

where

$$p_i(x; \Theta_i) = \sum_{j=1}^{K} \alpha_{i,j} \mathcal{N}(x; \mu_{i,j}^x, \Sigma_{i,j}^{xx}), \tag{5.5}$$

is the probability density in the domain derived from marginalizing the variable $y$ in $p_i(x, y; \Theta_i)$ (4.59), $n_i$ is the total number of samples represented in the density model (4.58),

$$n_i = \sum_{j=1}^{K} \{1\}_{i,j}, \tag{5.6}$$

and $\tilde{V}_i$ is the volume of the region $\tilde{X}_i$ surrounding $x$, which is defined empirically so as to have homogeneous values for $\mu_i(y|x;\Theta_i)$ and $\sigma_i^2(y|x;\Theta_i)$ in that region. Note that, in this case, the parameters for all the point-dependent estimations of the DFA system, i.e. $\xi_i$, $\tau_i$, and $\varsigma_i$, are those from the corresponding GMM, $\Theta_i$.

Equations (5.3) and (5.4) provide precise estimations of the variance and number of samples, respectively, which permits, in turn, a point-dependent estimation of the quality in the approximation through the relevance (5.2), making the competitor selection robust to the biased sampling problem. In addition, the multi-parametric definition of the competitor (5.1) permits approximating a rich set of functions in the competitor domain.

### 5.2.1 Competitor Management using Probability Density Estimations

So far we have focused in the definition of the DFA regarding the competitors and relevance functions, and little attention was given to the mechanisms for competitor generation. In this section we propose an approach to generate competitors on demand for a better approximation, when the competitors provided so far do not suffice. Note that the approximation may be refined either by generating new Gaussians at each competitor, by generating new competitors, or by combining both strategies. Given that a representation with few parameters adapts faster than a complex representation, the number of Gaussians per competitor is kept fixed to favour more rapid convergence. We also propose a strategy to eliminate competitors that had been generated to improve the approximation, but later became less useful for the system as a consequence, for instance, of the non-stationarity of the target function.

The purpose of competitor generation is to find, incrementally, the regions of the domain in which the target function can be well approximated with the GMM provided to each competitor. This implies that, the generalization capabilities of the DFA will depend on whether the approximated function presents wide regions of this kind, and the ability of the method to create competitors which capture these regularities.

For the competitor generation we adopt the optimistic assumption that the target function presents large regions of the domain which are well approximated with $K$

Gaussians, initializing the system with few competitors with large domains [1]. $K$ is the fixed amount of Gaussians provided to the GMM of each competitor.

#### 5.2.1.1 Evaluating the Approximation

In order to evaluate the necessity of improving the approximation with the generation of new competitors, we should define the criteria that permit to distinguish between a poor approximation caused by the lack of competitors, and a poor approximation produced by other reasons, like, for instance, the lack of experience. Note that, the DFA may have enough competitors to fulfil the approximation requirements, but still produce large approximation error due to the lack of samples used in the updating of their parameters.

In the same way, there would be many other sources of large approximation errors different from the lack of competitors. For instance, in RL, a large error may be produced by the variations of the target function during the learning process, that may cause new observations to be different from previous ones, at the same regions. Depending on how flexible is the method used for the parameters adaptation, the poor approximation may be improved by just waiting for a new adaptation of the parameters, rather than generating new competitors. Another source of error may be the eventual stochastic nature of the target function. This may produce large errors in punctual estimations as a consequence of outliers generated randomly from the underlying probability distribution.

In our approach we will assume that a large error is caused either by a lack of experience, or by a lack of competitors, since the other mentioned sources of error are difficult to quantify. This incomplete evaluation of the approximation error may incur in an over-generation of competitors. Fortunately, the over-generation of competitors may just imply more chances of having a good competitor among the competing ones, without affecting negatively the convergence rate. Note that, contrarily to our case, the over-generation of parameters is actually a problem for most of the non-parametric FA approaches, since it represents an increment in the number of experiences required for convergence. Despite DFA is in principle benefited from the over-generation of

[1] Actually, the initial number of competitors could be arbitrarily chosen, as far as all the points of the target function domain are covered, at least, by one competitor.

competitors, we control the proliferation of them using a mechanism for competitor elimination. This is done to prevent an excessive computational cost for storing and processing all of them.

In order to evaluate when a poor approximation is caused by the lack of experience, we adopt the criterion of defining a threshold, $n_c$, for the minimum number of experiences that should be collected in a competitor domain to consider the estimations of its parameters as confident,

$$n_{min} > n_c, \tag{5.7}$$

where

$$n_{min} = \min_{\Phi_i \in \mathbf{\Phi_x}} n_i, \tag{5.8}$$

where $\mathbf{\Phi_x}$ is the set of active competitors. In turn, to evaluate the accuracy in the approximation we set a threshold for the minimum error allowed in the approximation of the winner competitor, $e_c$,

$$(f(x) - \mu_w(y|x))^2 \geq e_c. \tag{5.9}$$

If both thresholds, $n_c$ and $e_c$, are surpassed then we assume that the source of error is the lack of competitors and the generation mechanism is triggered. Many generation strategies can be applied. We propose two complementary ways of generating new competitors: generation of a competitor from a combination of two existing ones, and generation of a competitor from an specialization of the winner competitor.

#### 5.2.1.2 Generation from Combining Two Competitors

One strategy for competitor generation is to generate a competitor based on the information provided by two active competitors (3.2). These competitors are the competitor which has the least prediction error at the experienced point, and the winner competitor. The domain of the new competitor will be formed by the intersection of the domains of the two selected competitors. We choose this strategy as we consider that the region delimited by the intersection of the domain of the competitor with least error and the

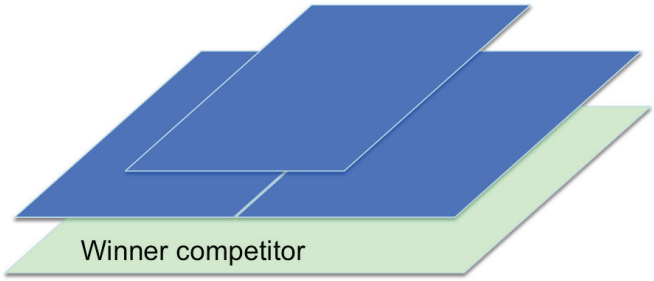



Figure 5.1: Illustration of the generation from splitting for a 2D domain. The domain of the winner competitor is painted in green, and those for the new generated competitors are shown in blue.

winner competitor has more chances to be well approximated by the GMM provided to the new competitor. Generation is carried out whenever the resulting domain does not coincide with a domain of an active competitor. Otherwise, the generation is only carried out from specializing the winner competitor.

#### 5.2.1.3 Generation from Specializing the Winner Competitor

Generation from specializing the winner competitor is carried out by splitting the domain of the winner competitor in three overlapped domains of half the size of the domain of the winner competitor, as illustrated in figure 5.1.

The splitting is performed along the dimension of the domain of the winner competitor where the samples are more disperse [1]. The dispersion along dimension $d$ is calculated from the marginal variance in $d$ from the GMM of the winner competitor,

$$\sigma_d^2 = \sum_{j=1}^{K} \alpha_j (\sigma_{d,j}^2 + (\mu_{d,j} - \mu_d)^2), \tag{5.10}$$

where $\mu_{d,j}$ is the expected value of the probability obtained by marginalizing in $p_w(x, y)$

[1] It may occur that, all the domains tried from splitting are already considered for other competitors domain. In this case, the splitting is carried out along the next dimension with largest dispersion. In the unusual, though possible, case that the splitting is already tried for all the dimensions, then the next most relevant competitor, i.e. the successor of the winner, is selected for splitting.

all the variables except $d$, and

$$\mu_d = \sum_{j=1}^{K} \alpha_j \mu_{d,j}. \tag{5.11}$$

Generation from splitting the winner competitor complements the generation from combination of two active ones, as it permits to increase the resolution to any extent, which would not be possible by only generating from the combination of two competitors.

#### Initialization of the GMM of New Competitors

Every new competitor is initialized with a GMM of $K$ Gaussians. In case the competitor domain contains the experienced sample, one Gaussian is initialized with a mean vector equal to the experienced sample $(x_t, y_t)$. The rest of the Gaussians are initialized with mean vectors with components for the domain variables provided by points selected randomly in the input space, and with component for the output variable given by the inference of the DFA system at the selected inputs.

The covariance matrix of each new Gaussian is initialized to a diagonal matrix, with components for the domain variables given by a percentage $C$ of the span of the variable inside the competitor domain. The component for the output variable is the variance of the output, estimated from the DFA system, at the inputs selected to create the mean vectors.

Finally, the accumulator of the number of samples of each new Gaussian is initialized to 1.

### 5.2.1.4 Elimination of Competitors

The strategy adopted for the generation of competitors is not exempt of producing competitors which do not improve the accuracy in the approximation. Some of the new competitors may be equally, or less, accurate than other existing ones. Some others may improve the accuracy at early stages after their generation, but turn out to be less precise in later stages of the learning process due, for instance, to the non-stationarity of the target function. However, the inaccuracy of a competitor does not

**Algorithm 6** Degenerate Function Approximation with GMMs

initialize the degenerate function, $\mathcal{F}(x)$, using a set of competitors $\mathbf{\Phi}$ whose domains form a covering of the target function domain.
initialize the GMMs of each competitor with a set of $K$ Gaussians.
**loop**
  get observation $(x, y)$
  get the active competitors $\mathbf{\Phi}_x$
  update the parameters of the GMM of each active competitor using online EM (section 4.3.1)
  get the minimum number of samples $n_{min}$ (5.8)
  select the winner competitor in $x$, $\Phi_w(x)$
  calculate approximation error $e = (y - \Phi_w(x))^2$
  **if** $n_{min} < n_c$ & $e < e_c$ **then**
    generation from combination
    generation from splitting
  **end if**
  **if** modulo(iterations/$it_{\text{red}}$) $= 1$ **then**
    eliminate redundant competitors from $\mathbf{\Phi}_x$
  **end if**
**end loop**

necessarily imply that the competitor is useless for the system. For example, even though a competitor has large approximation error, it may be the best the system has so far at some points of the domain.

Then, to control the proliferation of competitors we do not eliminate competitors which are inaccurate, but competitors which have not won in the last $n_a$ times they were active. At a predefined number of iterations $it_{\text{elim}}$, we eliminate all the competitors that fulfil the elimination criterion.

### 5.2.2 Algorithm for the DFA using Probability Density Estimations

The complete algorithm for the online, memory-free, degenerate function approximation using probability density estimations is presented in algorithm 6.

## 5.3 $Q$-Learning with a Degenerate Function Approximation

We are now in position to describe the mechanisms for FA in $Q$-Learning in continuous state and action spaces with a DFA using probability density estimations, named as the DFARL method. As usual, the samples of the target function will consist on the $q$-values derived from the sampled version of the Bellman equation,

$$q(s,a) = r(s,a) + \gamma \max_{a'}(\hat{Q}(s',a')), \tag{5.12}$$

where $r(s,a)$ is the immediate reward obtained after executing action $a$ in state $s$, $s'$ is the state observed after the action execution, and $\hat{Q}(s',a')$ is the approximation of the action-value function at $(s',a')$ provided by the DFA system, $\hat{Q}(s',a') = \Phi_w(s',a') = \mu_w(q|s',a')$. To compute the value $q(s,a)$ we need to solve the maximization problem $\max_{a'}(\hat{Q}(s',a'))$. We adopt the same strategy as before of computing the values $\hat{Q}(s',a')$ for a finite number of actions, and then taking the value of the action that provides the largest $Q$ as the approximated maximum.

### 5.3.1 Exploration-Exploitation Strategy

For action exploration we use the strategy as in section 4.7.1, selecting for each action to be evaluated a value $t_{rand}$ obtained randomly from a $t$-distribution with $n_w(s,a)-1$ degrees of freedom, and finding the corresponding value,

$$\hat{Q}_{rand}(s,a) = \hat{Q}(s,a) - t_{rand}\frac{S_w(s,a)}{\sqrt{n_w(s,a)}}, \tag{5.13}$$

where $\hat{Q}(s,a) = \Phi_w(s,a)$ is the value inferred by the DFA system at $(s,a)$, $S_w^2(s,a)$ is the variance estimation at $(s,a)$ , and $\tilde{V}_w n_w p_w(s,a)$ is an estimation of the number of samples $n_w(s,a)$ characterized by the statistical values at $(s,a)$. Once a $\hat{Q}_{rand}(s,a)$ value has been assigned for every evaluation $(s,a)$, the action to be executed is selected using the greedy policy as

$$a = \operatorname*{argmax}_{a'} \hat{Q}_{rand}(s,a'). \tag{5.14}$$

## 5.4 Performance Evaluation

In order to assess the generality and efficiency of the approach we will consider three different tasks that cover the basic type of difficulty in control problems [70]: *avoidance*, which consists in keeping the system into a valid region of the state space; *goal reaching*, where the system is requested to reach a goal area of the state space, finishing the task when it gets there; and *regulation*, where the system is also requested to reach a goal but, contrary to finishing the task when this happens, it should stay there using active control.

To implement these tasks we will use three standard benchmarks of Reinforcement Learning. The avoidance control task will be implemented with the cart-pole balancing benchmark that consists in keeping a cart and a pole into an acceptable working region by movements of the cart. The goal reaching task will be implemented by the mountain-car benchmark, where a car should reach a goal position at the top of a mountain, point in which the task is finished. Finally, the regulator task will be implemented by the inverted pendulum benchmark, where an under-actuated pendulum should be balanced from the downward position until it reaches the upward position in which it should be stabilized using active control. These benchmarks permit to assess not only the generality and efficiency of the DFARL method but also its scalability since the cart-pole balancing benchmark has two more dimensions than the other benchmarks.

### 5.4.1 Comparison with Other Methods

In order to compare the performance of our approach with that of other RL approaches we should consider some factors. Since our approach is memory-free and model-free, its convergence speed is only directly comparable with those of model-free and memory-free approaches. However, for those approaches that are memory-based, e.g. fitted value iteration methods, an indirect comparison is possible by comparing, for instance, the number of times the system needs to be updated until a good performance is achieved. Model-based approaches are not considered for comparison.

In particular, the performance of our approach will be directly compared, in all the benchmarks, with the best configurations found empirically for the methods of a GMMRL approach, presented in chapter 4, and a memory-free version of the Variable

Resolution (VR) (see appendix A), this last to provide a general efficiency reference. In addition to these comparisons, we will provide comparisons with other remarkable state of the art approaches at each particular benchmark, comparisons that will be direct or indirect depending on whether they are memory-free or not.

### 5.4.2 A Regulation Problem: the Inverted Pendulum

The inverted pendulum benchmark [30] is depicted in figure 5.2. The problem definition in this benchmark is the same used in section 4.8 to evaluate the GMMRL method.

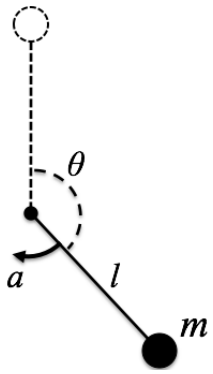



Figure 5.2: Inverted pendulum benchmark.

#### 5.4.2.1 Learning Methods Set-up

Table 5.1 compiles the values of the learning parameters for each of the learning methods. The parameters related to a GMM in the column of the DFARL method correspond to the parametric GMMRL at each competitor.

The covariances of the initial Gaussians for the DFARL and the GMMRL approaches are determined as

$$\Sigma_{K+1} = C\ diag\{d_1, ..., d_D, \text{rang}_q\}, \tag{5.15}$$

where $d_i$ is the total span of the range of variable $i$, for the case of the GMMRL, and the span of the range of variable $i$ inside the competitor domain, for the case of the

Table 5.1: Learning parameters for the Inverted Pendulum.

| Parameter | DFARL | GMMRL | VR |
|---|---|---|---|
| # initial Gaussians $K$ | 10 | 10 | – |
| Initial samples per Gaussian $\{1\}_j$ | 0.1 | 0.1 | – |
| Dispersion factor of initial Gaussians $C$ | 0.5 | 0.2 | – |
| Initial mean vectors for Gaussians $\mu_j$ | random | random | – |
| Learning rate parameter $a$ | 0.001 | 0.001 | 0.1 |
| Learning rate parameter $b$ | 10 | 10 | 10 |
| Reference volume $\tilde{V}$ | 1 | 1 | – |
| Approximation error threshold $e_{c,g,v}$ | $0.025 * \text{rang}_q$ | $0.025 * \text{rang}_q$ | $0.025 * \text{rang}_q$ |
| Minimum number of samples $n_{c,g,v}$ | 50 | 200 | 50 |
| Relative marginal probability $\beta_c$ | – | 0.1 | – |
| Last activations threshold $n_a$ | 1000 | – | – |
| Iterations to check elimination $it_{\text{elim}}$ | 1000 | – | – |

DFARL; $D$ is the dimension of the domain; and $C$ is a positive value defining the dispersion of the new Gaussian.

The initialization of the competitors in the DFARL approach consists in a set of 9 competitors whose domains have half of the size of the input space. They are generated by splitting each variable in three overlapped segments of half of the size of its total span, and then combining them with the total span of the rest of the variables to form the competitors' domains. Figure 5.3 presents an example of competitors generated in this way for the case of a 2D input space.

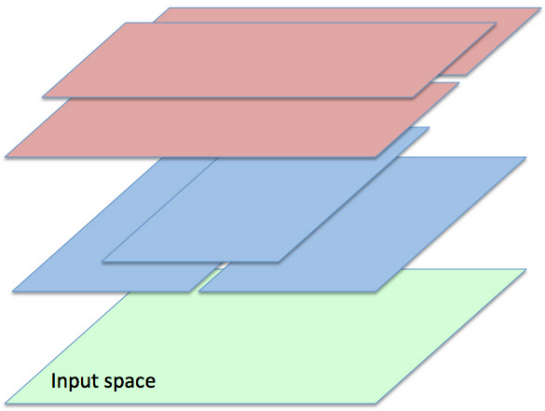



Figure 5.3: Example of the domains of the initial competitors in a 2D input space.

#### 5.4.2.2 Experiments Set-up

The experiments set-up is the same used in section 4.8 for the evaluation of the GMMRL method. In this set-up, the experiments are performed using episodes of 500 iterations. At the beginning of each episode, the pendulum is placed in the hang-down position. At the end of each episode, a test of 500 iterations is performed exploiting the policy learned so far, also starting from the hang-down position. As the result of the test we take the sum of the rewards obtained at each iteration. For each experiment we show the average of 15 independent runs of 100 test episodes.

#### 5.4.2.3 Results

The results, depicted in figure 5.4, show that the DFARL method has a higher convergence rate than the GMMRL approach, and a clearly superior performance than the VR approach. This is a remarkable result since the GMMRL method has already a very good performance with respect to other approaches (see section 4.8).

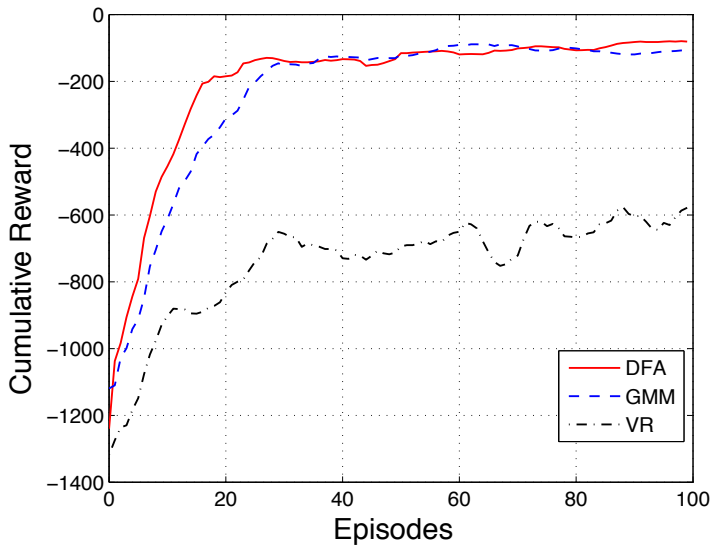



Figure 5.4: Comparison of the performances of the DFARL, GMMRL, and VR approaches.

To exemplify the performance of the GMMRL approach when it is provided with the same amount of initial Gaussians as in the DFARL approach, we carried out an

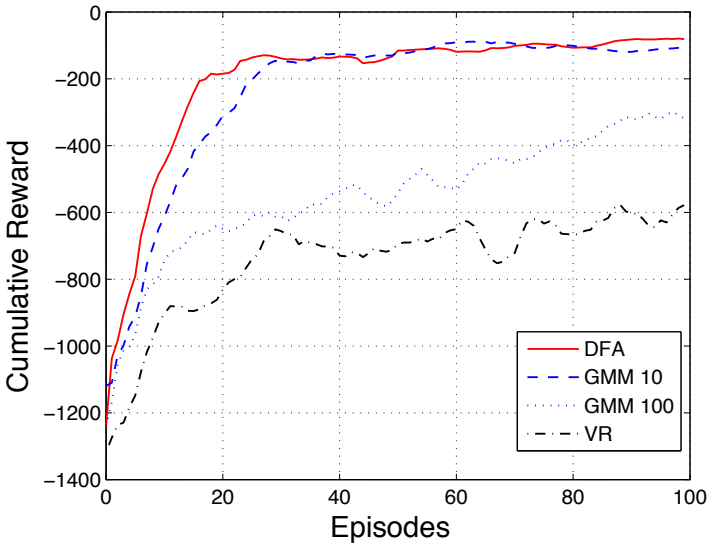



Figure 5.5: Comparison of the performance of a GMMRL with 10 initial Gaussians, a GMMRL with 100 initial Gaussians, the DFARL, and the VR methods.

experiment initializing the GMMRL with the same total number of Gaussians considered initially by the DFARL, i.e. $K = 10 \times 10 = 100$. The results, presented in figure 5.5, show that the convergence rate of the GMMRL with 100 initial Gaussians, is lower than when using 10 initial Gaussians. For the GMMRL with 100 initial Gaussians the performance is worst likely due to the fact that the system must simultaneously balance a large number of parameters (in fact, 100 Gaussians is more than required, since the GMMRL with 10 initial Gaussians reaches convergence with an average of about 65 Gaussians). We would like to remind that, in the DFARL, each competitor constitutes an independent parametric GMMRL method with convergence properties of a GMMRL with 10 Gaussians. Increasing the number of competitors, and therefore the total number of Gaussains, does not slow down the convergence of the DFARL approach since each competitor must only balance their own parameters, independently of those of other competitors.

To provide an idea about the proliferation of competitors, the DFARL approach achieves a good performance with an average of 201 competitors, and an average of 2 competitors eliminated. We present in figure 5.6 an example of 2D histograms showing the number of overlapped competitors in 2-dimensional projections of the 3-dimensional

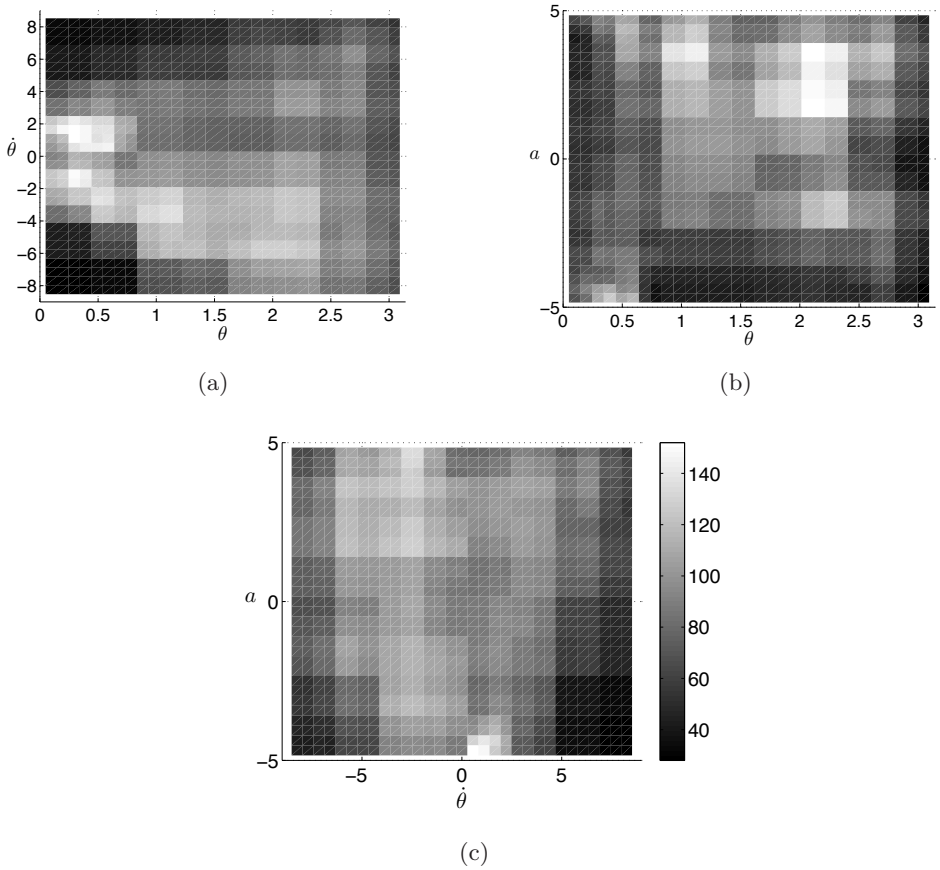



Figure 5.6: Number of overlapped competitors in the projected spaces $(\theta, \dot{\theta})$, $(\theta, a)$, and $(\dot{\theta}, a)$, for the inverted pendulum task.

state-action space. In general, we can observe a proliferation of competitors in most visited regions, and in regions where a more precise control is required. For instance, in figure 5.6(a) we can observe a larger proliferation of competitors in the region close to the goal state, $s = (0, 0)$, where a more precise control is necessary to keep the pendulum balanced. In this figure, and in figure 5.6(b), we can also observe that more competitors were generated in regions with angular position $\theta \in [2, 2.5]$. This region is crucial for the control task since, when the pendulum reaches this region after being swung from the downward position, it has already enough potential energy to reach the upward position by applying a high action in the opposite direction.

### 5.4.3 A Goal Reaching Problem: the Mountain-Car

The benchmark problem of the mountain-car [81] consists in driving an underpowered car up a steep mountain road (figure 5.7). The difficulty of the problem is that the force exerted by the gravity is larger than the force of the engine of the car. Then, in order to climb the mountain, the car needs to first move away from the mountain up to the opposite slope, and then apply maximum acceleration to accumulate enough inertia to reach the tip of the mountain.

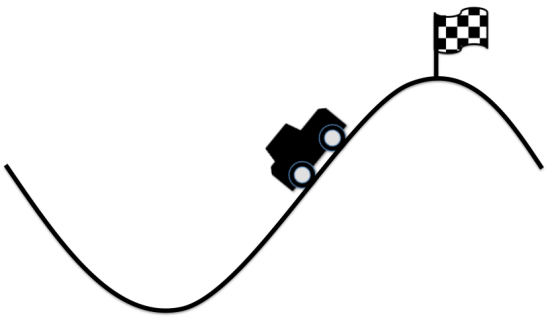


Figure 5.7: Mountain-car benchmark.

The equations modelling the dynamics of the car are,

$$p_{t+1} = \text{bound}\left[p_t + v_{t+1}\right], \tag{5.16}$$

$$v_{t+1} = \text{bound}\left[v_t + 0.001a_t + (-0.0025)\cos(3p_t)\right], \tag{5.17}$$

where $p_t$ is the car position at time $t$, $v_t$ is the car velocity, and $a_t$ is the discrete action consisting in the car acceleration forward, $a_t = 1$, backward, $a_t = -1$, or no acceleration $a_t = 0$. The operator bound keeps the position and velocity in the intervals $[-1.2, 0.5]$ and $[-0.07, 0.07]$, respectively. The car arrives at the goal position whenever the condition $p_t = 0.5$ is achieved, in which case the reward is $r = 0$ and the episode terminates. In any other case the reward is $r = -1$. When the car reaches the left position bound, its velocity is set to 0. For this task the discount coefficient $\gamma$ is set to 0.999.

#### 5.4.3.1 Learning Methods Set-up

Table 5.2 summarizes the values of the parameters used for each of the learning methods.

Table 5.2: Learning parameters for the Mountain-car.

| Parameter | DFARL | GMMRL | VR |
|---|---|---|---|
| # initial Gaussians $K$ | 10 | 10 | – |
| Initial samples per Gaussian $\{1\}_j$ | 0.1 | 0.1 | – |
| Dispersion factor of initial Gaussians $C$ | 0.2 | 0.2 | – |
| Initial mean vectors for Gaussians $\mu_j$ | random | random | – |
| Learning rate parameter $a$ | 0.001 | 0.001 | 0.1 |
| Learning rate parameter $b$ | 20 | 10 | 20 |
| Reference volume $\tilde{V}$ | 1 | 1 | – |
| Approximation error threshold $e_{c,g,v}$ | $0.005 * \text{rang}_q$ | $0.005 * \text{rang}_q$ | $0.005 * \text{rang}_q$ |
| Minimum number of samples $n_{c,g,v}$ | 100 | 200 | 100 |
| Relative marginal probability $\beta_c$ | – | 0.1 | – |
| Last activations threshold $n_a$ | 5000 | – | – |
| Iterations to check elimination $it_{\text{elim}}$ | 5000 | – | – |

In this problem, we use an independent DFARL approach for each of the discrete actions, defined in the 2-dimensional state space $(pos, vel)$. Each DFARL system is initialized with 22 competitors. 16 competitors are generated as parts of a $4 \times 4$ grid that covers the state space. The rest of competitors are generated by splitting each range of the state variables in three overlapped segments, and building competitors whose domains are defined using one of these segments, and the whole range for the rest of the variables, as exemplified in figure 5.3.

#### 5.4.3.2 Experiments Set-up

To permit a more exhaustive evaluation of the DFARL approach we will use the same experimental set-up as in [93] to test the Adaptive Tile Coding method (ATC) in the same benchmark of the mountain-car. The ATC is a novel online memory-free function approximation method that is simple and computationally efficient, and that proved to work efficiently in this benchmark. In short, the ATC represents the input space using a multi-partition scheme (see section 2.3.2). The approach is non-parametric since it

automatically searches for the appropriate granularity at each partition following a general to specific strategy, splitting parts in two halves along a dimension selected according to one of two alternative splitting criteria: a value criterion, that estimates how much the value function will change if the part is split along that dimension, and a policy criterion, that estimates how much the policy will change if the splitting occurs.

In the experiments, we implement training episodes of 300 iterations, or until the car reaches the goal position. After 600 updates, a test episode exploiting the policy learned so far is carried out during 300 iterations, or until goal is reached. As the result of the test we take the sum of the rewards obtained at each iteration. Both, training and test episodes are initiated in a state selected randomly. For each experiment we show the average of 15 independent runs.

#### 5.4.3.3 Results

The results are presented in figure 5.8. The performance of the DFARL approach is significantly better than the performance of the global GMMRL and VR methods.

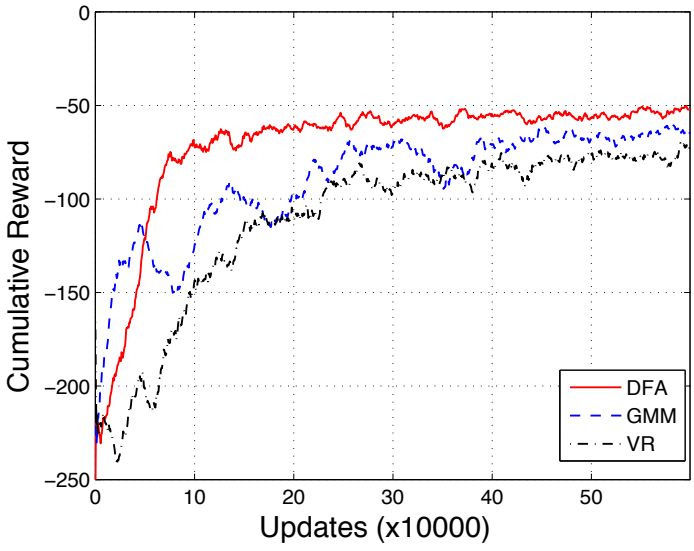



Figure 5.8: Comparison of the performance of the DFARL, GMMRL, and VR methods in the mountain-car task.

Regarding the number of competitors generated, considering the three actions, the

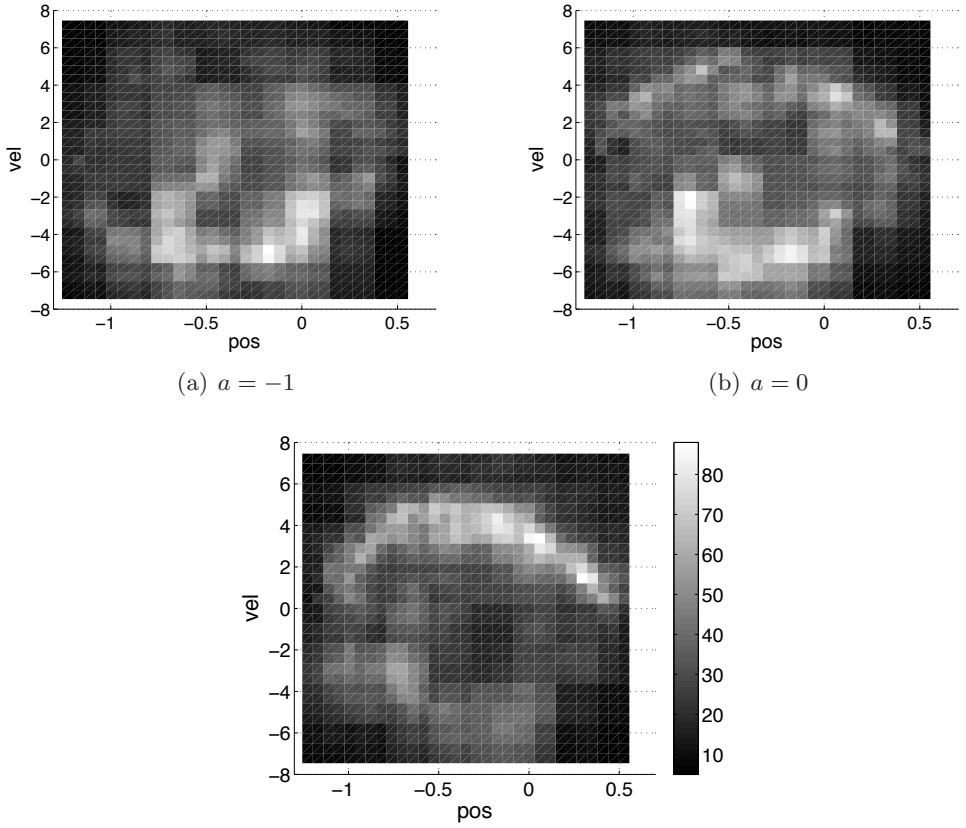
DFARL achieves a good performance with an average of 1509 competitors and an average of 390 competitors eliminated. To show the distribution of the competitors generated, we provide in figure 5.9 the 2D histograms of the competitors generated for each of the three actions considered. In general, we can observe a larger proliferation of competitors at the most visited regions, and in regions more crucial for the task. For instance, in the histogram for action 1 (figure 5.9(c)), we observe that more competitors were generated in regions with positive velocities, mainly at positions close to the goal, where a positive action would help the car to reach the top of the mountain. For the action -1 (figure 5.9(a)), instead, the proliferation of competitors takes place mainly at negative velocities in positions around the valley, where a negative action would help the robot to gain enough potential energy in climbing the opposite mountain, to afterwards reach the goal with maximum positive acceleration.

To more exhaustively evaluate the DFARL approach, we compare the results obtained with those reported in [93] for the Adaptive Tile Coding algorithm. For comparison, we adopt the same evaluation criteria they use to evaluate the ATC, of considering that a good policy is achieved when the reward accumulate in test episodes reaches the -100 value, and that a near optimal policy is achieved when the accumulated reward is close to -50. From the results reported in [93], the ATC achieves a good policy in about 400,000 updates, and raises above -60 at approximately 2,800,000 updates. Our approach achieves a good policy in only 62,000 updates, and raises over -60 at 360,000 updates.

### 5.4.4 An Avoidance Problem: the Cart-Pole Balancing

The Cart-Pole Balancing benchmark [18] consists in a pole mounted on cart that has to be stabilized inside an acceptable region by the motions of the cart (figure 5.10). The cart is free to move in an acceptable range within a 1-dimensional bounded track, and the pole moves in a vertical plane parallel to this track.

The equations for the model of the cart-pole system are




(a) $a = -1$

(b) $a = 0$

(c) $a = 1$

Figure 5.9: Number of overlapped competitors in the state space for actions -1, 0, and 1, in the mountain-car benchmark.

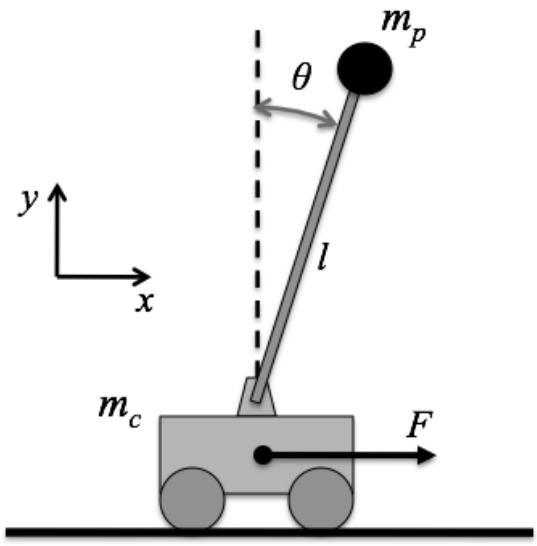



Figure 5.10: Cart-pole balancing benchmark.

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left[-F - m_p l\dot{\theta}\sin\theta + \mu_c \text{sgn}(\dot{x})\right] - \frac{\mu_p\dot{\theta}}{m_p l}}{l\left[\frac{4}{3} - \frac{m_p\cos^2\theta}{m_c+m_p}\right]}, \tag{5.18}$$

$$\ddot{x} = \frac{F + m_p l\left[\dot{\theta}\sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c \text{sgn}(\dot{x})}{m_c + m_p}, \tag{5.19}$$

where $x$ is the cart position, $\theta$ is the angular position, $m_p$, is the mass of the pole, $l$ is the length of the pole, $m_c$ is the mass of the cart, $F = a$ is the control action, consisting in the acceleration applied to the cart, $g$ is the gravity constant, and $\mu_c$ and $\mu_p$ are the friction coefficients of the cart and the pole, respectively. The parameters of the model are set to $m_p = 0.15kg$, $m_c = 1.0kg$, $l = 0.75m$, and $g = 9.81m/s^2$. The friction coefficients are set to 0. The action takes values within the interval $[-50, 50]N$. To solve the maximization $\max_a(\hat{Q}(s', a))$ in (5.12), we use a set of actions obtained from a uniform sampling of the action variable with period 2. The actuation frequency is 60 Hz. The discount coefficient is set to $\gamma = 0.95$. The acceptable region of the pole is defined by $-\pi/6 \leq \theta \leq \pi/6$, and the one corresponding to the cart is defined by $-1.5m \leq x \leq 1.5m$. The reward function selected varies linearly from 0 at the bound

of the acceptable angle for the pole, to 1 at the vertical position. Reward is -1 when either the pole or the cart leaves its acceptable region.

The cart-pole benchmark is of interest since it involves a 5-dimensional state-action space, $(s, a) = (x, \dot{x}, \theta, \dot{\theta}, a)$, two more than the inverted pendulum and mountain-car, and serves to illustrate the scalability of the algorithm.

#### 5.4.4.1 Learning Methods Set-up

Table 5.3 presents the values of the parameters used for each the learning methods for the problem of the Cart-Pole Balancing.

Table 5.3: Learning parameters for the Cart-Pole balancing.

| Parameter | DFARL | GMMRL | VR |
|---|---|---|---|
| # initial Gaussians $K$ | 20 | 20 | – |
| Initial samples per Gaussian $\{1\}_j$ | 0.1 | 0.1 | – |
| Dispersion factor of initial Gaussians $C$ | 0.2 | 0.2 | – |
| Initial mean vectors for Gaussians $\mu_j$ | random | random | – |
| Learning rate parameter $a$ | 0.001 | 0.001 | 0.1 |
| Learning rate parameter $b$ | 25 | 15 | 25 |
| Reference volume $\tilde{V}$ | 50 | 50 | – |
| Approximation error threshold $e_{c,g,v}$ | $0.025 * \text{rang}_q$ | $0.025 * \text{rang}_q$ | $0.025 * \text{rang}_q$ |
| Minimum number of samples $n_{c,g,v}$ | 100 | 100 | 50 |
| Relative marginal probability $\beta_c$ | – | 0.1 | – |
| Last activations threshold $n_a$ | 1000 | – | – |
| Iterations to check elimination $it_{\text{elim}}$ | 1000 | – | – |

We provide the DFARL system with an initial set of 15 competitors generated by splitting each range of a variable in the state-action space in three overlapped segments, and building competitors whose domains are defined using one of these segments, and the whole range for the rest of the variables, as illustrated in figure 5.3.

#### 5.4.4.2 Experiments Set-up

The set-up for the experiments is basically the same used in [63, 64]. A training episode starts in a random state obtained from a normal distribution with mean vector $\mu_0 = (0, 0, 0, 0)$ and covariance matrix $\Sigma_0 = diag(0.1, 0.1, 0.1, 0.1)$. Each training

episode lasts 5 seconds or until the system fails in keeping the pole, or the cart, in the acceptable region. After 10 seconds of simulated training time, a test episode is performed. Each test episode consists in averaging the sum of rewards obtained during runs of 5 seconds of simulated time, or until failure, starting at four different positions of the pole: $-10\,^\circ, -5\,^\circ, 5\,^\circ, 10\,^\circ$, with the cart centred on the track. For each experiment we calculate the average of 15 independent runs.

#### 5.4.4.3 Results

Figure 5.11 presents the results of the experiments where the maximum possible reward, estimated by exhaustive manual tuning, is also shown. Note that the different performance between the DFARL method and the GMMRL is even larger than in the case of previous comparisons. We attribute the superiority the DFARL over the GMMRL and VR methods to its better generalization capabilities which become more important as the dimensionality of the state-action space increases. GMMRL required, in average, a final amount of 400 Gaussians to reach convergence, while DFARL required, in average, about 222 competitors with 20 Gaussians each. Note that, even if the total number of Gaussians in DFARL is much larger, many competitors are updated in parallel at each iteration, and each of them has to adjust a much smaller set of parameters than the GMMRL and in a smaller region of the domain.

Regarding the number of competitors involved in the learning, a good performance is achieved with an average of 222 competitors, as mentioned above, and an average of 64 competitors eliminated. To provide an idea of the distribution of the generated competitors, we present in figure 5.12 the number of competitors overlapped in projections in the $(x, \dot{x})$, $(\theta, \dot{\theta})$, and $(x, \theta)$ spaces. As in previous cases, it can be seen a higher number of overlapped competitors at frequently visited regions, like around the $(0, 0, 0, 0)$ state, and in those regions where the control of the system is more challenging. For instance, in the $(x, \dot{x})$ projection (figure 5.12(a)), there is a proliferation of competitors in positions of the cart around $\pm 0.5$ with velocities that would make the system to fail, reaching positions close to the border, if no corrective action is applied (for instance, positive velocities for positive positions of the cart). Note that a more detailed analysis of the proliferation of competitors using 2D projections is actually

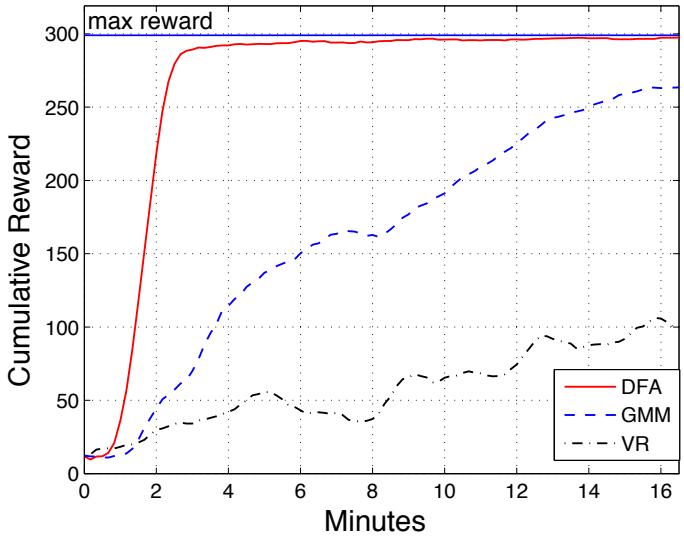



Figure 5.11: Comparison of the performance of the DFARL, GMMRL, and VR methods in the cart-pole balancing task.

very complicated for this task since they only provide a poor information about the actual distribution of competitors in the 5-dimensional space.

#### 5.4.4.4 Evaluation in the Stochastic Case

To assess the validity of the DFARL approach in stochastic domains, and to compare with the state of the art approach of Natural Actor Critic (NAC) [63], we evaluate its performance using a stochastic model of the dynamics described by $p(s_{t+1}|s_t, a_t) = \mathcal{N}(s_{t+1}, \sum_T)$, where $s_{t+1}$ is obtained as usual from equations ( 5.18) and (5.19), and $\Sigma_T = 0.01\Sigma_0$.

**Learning Methods Set-up**

The setting of the learning parameters is basically the same used for the deterministic case (table 5.3), with the only difference of the dispersion of the initial Gaussians $C$ for the DFARL approach, which is set to 0.5 instead of 0.2. This is so to prevent competitors generated from outliers to prematurely win. Note that a larger $C$ diminishes the

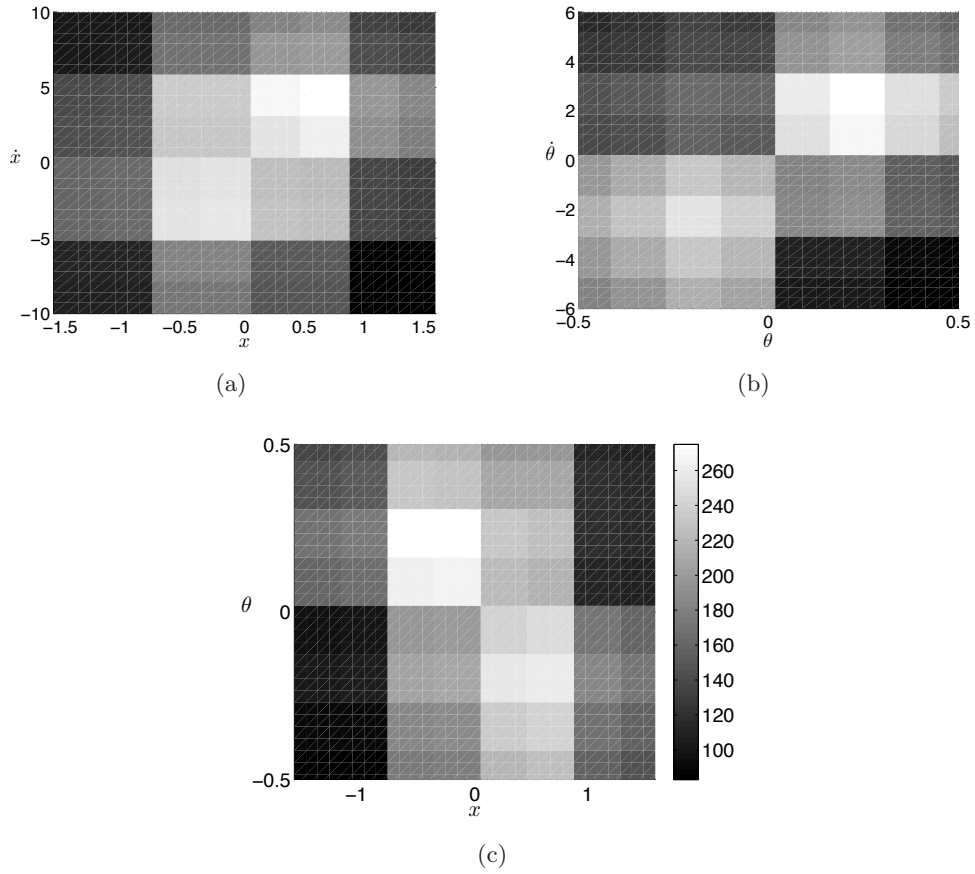



Figure 5.12: Number of overlapped competitors in the projected spaces $(x, \dot{x})$, $(\theta, \dot{\theta})$, and $(x, \theta)$, in the benchmark of the cart-pole balancing.

initial value of the sample density at the input space, $p_i(x)$, and so its initial relevance (5.2).

### Results

The results of the experiments for the stochastic case are shown in figure 5.13. In the figure we also indicate an estimation of the maximum possible cumulative reward that may be obtained with the stochastic model. This reference was calculated from averaging the cumulative reward of 10 test runs in the stochastic model using a set of competitors generated in the previous experiments, whose performance for the deterministic case is near optimal. As seen from the figure, the DFARL approach also outperforms the GMMRL and VR approaches.

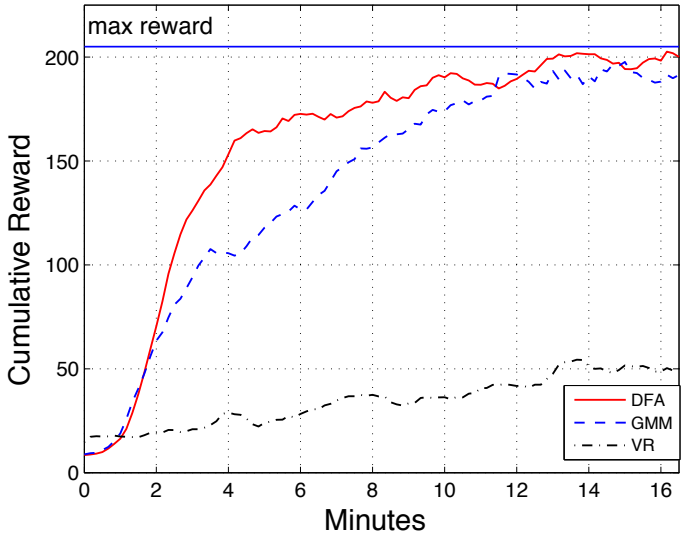



Figure 5.13: Comparison of the performance of the DFARL, GMMRL, and VR approaches using stochastic state transitions in the cart-pole balancing task.

Finally, to provide a reference with respect to the state of the art, we observe that DFARL approach for the stochastic case reaches a good performance after about 4 minutes of simulated time, while [63], using a Natural Actor-Critic (NAC) approach, reported convergence after 10 minutes. Despite these results are remarkable, they should not be interpreted as a direct comparison with the NAC since it is a policy

search approach and since, in [63], it is only provided the curve for the expected return over updates in the training phase, which is not an indicator of the actual control performance of the system but of the convergence of the parameters of the policy approximation. Instead, in our case, we provide the sum of cumulative rewards obtained in test episodes as an indicator of the control performance. Therefore, the results presented should be interpreted only as an indicator that our approach has an acceptable learning speed, in a range comparable to the state of the art.

### 5.4.5 Discussion

In all the tasks, the DFARL method achieves a near optimal performance and presents faster learning speed and more stable convergence profiles than the GMMRL and VR approaches, demonstrating its generality and robustness. These results support our claim that it is advantageous to use a collection of function approximators defined on overlapping regions of the whole domain that compete between them to provide the best estimation at each point, instead of using a single global approximator of arbitrary complexity.

In particular, the results highlight two characteristics of the DFARL method: locality and scalability.

#### Locality

Locality is important in the mountain-car task since it permits to better keep the informative reward obtained only when the goal is reached, avoiding it from being forgotten in the long run when many samples are observed with negative reward. The superior performance of the DFARL approach in this task is partly caused by its capability of keeping the approximation at each competitor unaltered by samples outside the competitor domain. In contrast, the GMMRL gets this relevant information more easily forgotten since the approximation at the goal region is also affected from samples at other regions of the domain. This may explain the more unstable convergence profile of the GMMRL method in the mountain-car compared with that of the other two problems.

Note that locality is also a feature of the VR approach, where each part is only updated from samples inside the part. This explains the good performance of the VR approach in the mountain-car task in comparison with that of the other two problems.

#### Scalability

The scalability of the DFARL approach is evidenced from the very good results obtained in the cart-pole balancing task, which has two more dimensions than the other tasks. In problems with large domains, a global approximator needs to balance a large number of parameters to achieve a good approximation at all the regions. This demands a lot of training experiences. In contrast, in the DFARL approach, each competitor has to deal with a reduced approximation problem since the approximation is carried out in a small region, usually much smaller than the entire domain of the problem, using a reduced set of parameters. This results in that each competitor requires much less experiences to converge. This, added to the fact that many competitors are updated in parallel with each experience, makes the DFARL approach appealing for high-dimensional problems.

#### On the Computational Cost

As expected, the computational cost of the DFARL method is higher than that of the GMMRL method. For instance, the amount of Gaussians stored for the DFARL approach is much higher than the Gaussians stored in the GMMRL. To give a more quantitative idea, the approximated number of Gaussians stored by the DFARL approach when a good performance is achieved is 2000, 15000, and 4400 for the inverted pendulum, mountain-car (considering all the actions), and cart-pole tasks, respectively. In these tasks, the corresponding number of Gaussians for the GMMRL approach is 45, 600, and 400. In this way, the number of Gaussians stored in the experiments by the DFARL is, at least, one order of magnitude larger than the GMMRL case. However, in the performed experiments, the increase in computation time was always less than one order of magnitude. This relatively small difference in the computation time is explained by the fact that, at each training instance, only the Gaussians of the active competitors are considered, whose number is only few times higher than the total number of Gaussians of the GMMRL approach. For instance, in the cart-pole benchmark, the average number of active competitors per training instance is 60, which leads to

an amount of 1200 Gaussians. This number is just three times higher than the 400 Gaussians used by the GMMRL approach.

# Chapter 6

# Conclusions

The main conclusions of the thesis are the following:

- In function approximation (FA) problems, trying many different approximators in parallel increases the opportunities of obtaining a good approximation than trying only one. If the best approximator is selected at each input, the competitive strategy achieves a good approximation faster and with a more stable convergence profile than a single function approximator. These properties are enhanced as more approximators are considered for the competition.

- A key aspect in the competitive strategy is the proper definition of the relevance function that, associated to each approximator, would permit to select the best approximator in a point. In this regard, in order to perform a correct selection of the best approximator at a given input, the relevance function should be point-dependent, instead of global.

- All the point-dependent estimations necessary to implement the competitive strategy for FA can be obtained from a probability density estimation in the joint input-output space. These point-dependent estimations are: an estimation of the approximated function, an estimation of the variance in the approximation, and an estimation of the density of samples in the input space. The last two estimations are used for a point-dependent estimation of the relevance of the approximator.

- A probability density estimation represented with a Gaussian Mixture Model (GMM) is suitable to be used for FA in Reinforcement Learning (RL). The approach, called GMMRL, has a performance comparable with the state of the art. Despite estimating the density in the joint input-output space is more demanding than learning a simple function approximation, the approach works efficiently since it allows using the simple and efficient Expectation-Maximization algorithm to rapidly obtain accurate estimations of the parameters of the model.

- The competitive strategy for FA in RL, the DFARL, which uses a GMM in each approximator, significantly outperforms the GMMRL approach and other state of the art approaches.

- Biased sampling is inherent to online RL and constitutes a serious problem for most FA methods. Therefore, for a FA method to succeed in RL, it has to properly deal with the biased sampling problem. We note that:

    * Conventional online updating formulas, which use a time-discounted forgetting, are not robust to the biased sampling problem since they replace old values based on time. In this case, the regions sampled long time ago get their estimations forgotten, no matter how well they are approximated, unlearning the approximation at those regions.

    * The probability density estimation with a GMM permits implementing an updating formula robust to the biased sampling problem with a forgetting based on the new information provided rather than on time, preventing the undesired effects produced by time-dependent forgetting.

    * Locality of the competitors in DFARL makes the approach even more robust to the biased sampling problem since the approximation at each competitor remains unaltered when the sampling occurs outside the competitor domain.

- Using a probability density estimation for FA in RL permits obtaining the probability distribution for the cumulative reward at each input, which can be used to devise different information-based strategies for exploration-exploitation.

## 6.1 Future Work

There are several possible improvements to the GMMRL and DFARL methods that could be explored. Below we enumerate some of them.

### Gaussian Management

For the generation of new Gaussians in the GMMRL approach, we have only considered the generation by adding new Gaussians since it is simple and showed to work efficiently, with little distortion in the approximation. An open research in this aspect is how to generate Gaussians by splitting existing Gaussians, without distorting the approximation achieved so far. Another possible aspect to tackle is Gaussian elimination, which is necessary to control an excessive proliferation of Gaussians.

### Competitor Management

One of the most direct improvements that can be made to the DFARL approach may be the definition of a better strategy for competitor generation to search for regions that can be well approximated by the competitor functions. So far we have proposed a simple general to specific strategy that starts with coarse competitors domains that are subsequently split or combined based on heuristic rules. However, many other strategies for competitor generation can be explored. For instance, the system could search for regions of the domain where the estimation of the variance is higher, and generate many alternative competitors there. Another possibility would be to follow simultaneously both, a general-to-specific strategy and a specific-to-general strategy, which may increase the chances of finding the proper granularity when the value function is unknown.

### Managing Competitors and their Gaussians at the Same Time

We could combine the improvement in the approximation obtained with the generation of competitors with the improvement of the approximation capabilities of each individual competitor by simultaneously performing competitor and Gaussian management. For instance, we could initialize each competitor with a small number of Gaussians and

allow the generation of new ones until some predefined number, e.g. ten, to keep the approximation simple. A more challenging possibility would be to make the approximation of each competitor non-parametric, by removing the limit in the number of Gaussians. This would rise the interesting question of deciding when it would be better to improve the system by creating a new competitor or by increasing the complexity of the existing ones.

### Trying Different Function Approximators

Another line to explore with the DFARL approach would be to use other types of function approximations for the competitor function, different from the GMMRL, at the expense of needing to separately approximate the relevance function. Depending on each specific problem, some FA methods may perform better than others, and usually the designer has no means to know beforehand which one to select. In this case, the DFARL approach could be made to allow trying different FA in parallel so as to select the best one at every region of the input space.

### Dimensionality Reduction

In many high-dimensional applications, the value function may largely depend on the values of some domain variables, which are said to be *relevant* variables, while presenting no dependency, i.e. no variations, along other *irrelevant* ones. Moreover, the relevancy of a variable may be different at different regions of the domain. When this characteristic is present in a domain, and when the number of relevant variables in each region is small with respect to the total number of variables, the domain is said to be categorizable [66]. Categorizable domains are interesting since irrelevant variables could be neglected in the function used for the approximation, which could drastically reduce the dimensionality problem. However, finding which variables are relevant in which regions is not an easy problem. Some strategies have been developed to exploit the categorization property in discrete domains with suggesting preliminary results [1–5, 66].

We believe that the DFARL approach provides a suitable base to extend these methods to continuous domain applications. This could be implemented by generating competitors that try approximations considering different set of variables, which are

presumably relevant for the approximation. Note that it may be possible to find large regions that can be well approximated with a simple function, but which require to consider all the variables since the approximated function presents variations along all of them. On the contrary, we may find regions where many variables become irrelevant, but the function present complex variations along the relevant ones, being not possible to perform a good approximation with a simple representation. Then, the competitors generated may not only try approximations considering different variables, but also considering different function complexities.

# Appendix A

# $Q$-Learning with a Variable Resolution Function Approximation

This appendix describes the VR algorithm used for comparison in the experiments of chapter 5. We implement an online memory-free version of the Variable Resolution approach. Variable Resolution is a state aggregation technique (section 2.3.2) that represents the value function, or the action-value function, using a partition of the state-action space, where each part aggregates the states, or state-action pairs, covered by that part. The algorithm is non-parametric since it automatically varies the resolution of the parts according to the approximation requirement.

In the partition, a part $i$ has associated an estimation of the action-value $Q_i = \hat{Q}(s,a)$ that is assigned to all the state-action pairs $(s,a)$ covered by the part. $Q_i$ estimates the average of the experienced $q$-values in the part calculated as

$$q(s,a) = r(s,a) + \gamma \max_a \hat{Q}(s',a) \tag{A.1}$$

where $r(s,a)$ is the immediate reward obtained after executing action $a$ in state $s$, $s'$ is the state observed after the action execution, and $\hat{Q}(s',a)$ is the approximation of the action-value function at $(s',a)$, provided by the part containing $(s',a)$. In the online memory-free approach the values $q(s,a)$ are used to update $Q_i$ using the incremental

formula for discounted average estimations,

$$Q_i = Q_i + \eta_i \left(q(s,a) - Q_i\right) \tag{A.2}$$

where $\eta_i$ is the learning coefficient. In our implementation, we use as learning coefficient a function of the form,

$$\eta_i = \frac{1}{a\,\nu_i + b}, \tag{A.3}$$

where $\nu_i$ is the number of samples experienced so far in the part, and $a \in (0, 1]$ and $b \leq 0$ are predefined parameters.

For the action exploration we use the same strategy as in section 4.7.1, of obtaining an action-value $\hat{Q}_{rand}(s,a)$ for each evaluated action through the formula (4.65), rewritten here for convenience,

$$\hat{Q}_{rand}(s,a) = \hat{Q}(s,a) - t_{rand}\frac{S(s,a)}{\sqrt{n(s,a)}}, \tag{A.4}$$

where $t_{rand}$ is a value obtained randomly from a $t$-distribution with $n(s,a) - 1$ degrees of freedom, and then, selecting the action with highest $\hat{Q}_{rand}(s,a)$ for its execution. To allow the computation of (A.4), we need an estimation of the variance of $q$ of the state-action pairs covered by the part, $S_i^2 = S^2(s,a)$, and an estimation of the number of samples $n_i = n(s,a)$. The variance $S_i^2$ is estimated with an incremental formula analogous to A.2,

$$S_i^2 = S_i^2 + \eta_i \left((q(s,a) - Q_i)^2 - S_i\right). \tag{A.5}$$

For the estimation of the number of samples $n_i$ we may just use a counter of the number of samples experienced so far inside the corresponding part. However, this would be correct only if the estimations $Q_i$ and $S_i^2$ are calculated without forgetting, i.e. when all the samples experienced in the past are equally important, in which case the learning coefficient would just be $1/\nu_i$. On the contrary, when a forgetting of past estimations is considered, old values have lesser influence in $Q_i$ and $S_i^2$ and should not be completely considered in $n_i$. It can be shown [9] that the actual number of samples used in $Q_i$

and $S_i$ is, in the general case, $n_i = 1/\eta_i$. We will use this estimation for the number of samples $n_i$.

**Partition Management**

In the implementation of the VR approach we will use an initial representation consisting in a grid formed by $N^D$ parts, where $N$ is the number of segments in which the range of each variable is divided, and $D$ is the dimension of the state-action space.

To evaluate the approximation, we adopt the same criteria as in the cases of DFARL and GMMRL, of considering that a poor approximation is carried out when the approximation error is high, above a predefined threshold,

$$(q(s,a) - \hat{Q}(s,a))^2 \geq e_v, \tag{A.6}$$

and when the number of samples experienced in the part fulfils

$$\nu_i > n_v, \tag{A.7}$$

so as to consider the estimations as confident.

When both criteria are surpassed, the part containing the experienced $(s,a)$ is split in two halves along the dimension where the length of the part is highest in relation to the total span of the corresponding variable.

# Bibliography


[1] A. Agostini and E. Celaya. Learning in Complex Environments with Feature-Based Categorization. In *Proc. of the 8th Conference on Intelligent Autonomous Systems (IAS8). (Amsterdam, Netherlands)*, pages 446–455, 2004.

[2] A. Agostini and E. Celaya. Learning Model-Free Motor Control. In *Proc. of the 16th European Conference on Artificial intelligence (ECAI'04). (Valencia, Spain)*, pages 947–948, 2004.

[3] A. Agostini and E. Celaya. Trajectory Tracking Control of a Rotational Joint using Feature-Based Categorization Learning. In *Proc. International Conference on Intelligent Robots and Systems (IROS'04). (Sendai, Japan)*, pages 3489–3494, 2004.

[4] A. Agostini and E. Celaya. Feasible Control of Complex Systems using Automatic Learning. In *Proc. of the 2nd International Conference on Informatics in Control, Automation and Robotics (ICINCO'05). (Barcelona, Spain)*, pages 284–287, 2005.

[5] A. Agostini and E. Celaya. Generalization in Reinforcement Learning with a Task-Related World Description using Rules. Technical Report IRI-TR-06-01, Institut de Robòtica i Informàtica Industrial, CSIC-UPC, 2006.

[6] A. Agostini and E. Celaya. Exploiting Domain Symmetries in Reinforcement Learning with Continuous State and Action Spaces. In *Proc. 8th International Conference on Machine Learning and Applications (ICMLA'09). (Miami, EE.UU.)*, pages 331–336, 2009.

[7] A. Agostini and E. Celaya. Reinforcement Learning for Robot Control Using Probability Density Estimations. In *Proc. of the 7th International Conference on*

*Informatics in Control, Automation and Robotics (ICINCO'10). (Madeira, Portugal)*, pages 160–168, 2010.

[8] A. Agostini and E. Celaya. Reinforcement Learning with a Gaussian Mixture Model. In *Proc. International Joint Conference on Neural Networks (IJCNN'10). (Barcelona, Spain)*, pages 3485–3492, 2010.

[9] A. Agostini and E. Celaya. Stochastic Approximations of Average Values using Proportions of Samples. Technical Report IRI-TR-11-02, Institut de Robòtica i Informàtica Industrial, CSIC-UPC, 2010.

[10] A. Agostini and E. Celaya. A Competitive Strategy for Function Approximation in Q-Learning. In *Proceeding of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11). Barcelona, Spain. Accepted*, 2011.

[11] A. Agostini and E. Celaya. A Competitive Strategy for Function Approximation in Reinforcement Learning (journal paper). *Submitted*, 2011.

[12] A. Agostini, E. Celaya, C. Torras, and F. Wörgötter. Action Rule Induction from Cause-Effect Pairs Learned Through Robot-Teacher Interaction. In *Proc. of the first International Conference on Cognitive Systems, CogSys 2008. (Karlsruhe, Germany)*, pages 213–218, 2008.

[13] A. Agostini, C. Torras, and F. Wörgötter. Integrating Task Planning and Interactive Learning for Robots to Work in Human Environments. In *Proceeding of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11). Barcelona, Spain. Accepted*, 2011.

[14] A. Agostini, F. Wörgötter, E. Celaya, and C. Torras. On-Line Learning of Macro Planning Operators using Probabilistic Estimations of Cause-Effects. Technical Report IRI-TR-08-05, Institut de Robòtica i Informàtica Industrial, CSIC-UPC, 2008.

[15] J.S. Albus et al. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of dynamic systems, measurement and control*, 97(3):220–227, 1975.

[16] J.A. Anderson and J. Davis. *An introduction to neural networks*. MIT Press, 1995.

[17] O. Arandjelovic and R. Cipolla. Incremental learning of temporally-coherent gaussian mixture models. In *Technical Papers - Society of Manufacturing Engineers (SME)*, 2005.

[18] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on systems, man, and cybernetics*, 13(5):834–846, 1983.

[19] R. Bellman. Dynamic Programming. *NJ, Princeton UP*, 1957.

[20] A. Bernstein and N. Shimkin. Adaptive-resolution reinforcement learning with polynomial exploration in deterministic domains. *Machine Learning*, 81(3):359–397, 2010.

[21] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[22] G. Blom. *Probability and statistics: theory and applications*. Springer-Verlag, 1989.

[23] A. Bonarini, A. Lazaric, and M. Restelli. Reinforcement Learning in Complex Environments Through Multiple Adaptive Partitions. *Lecture Notes in Computer Science*, 4733:531–542, 2007.

[24] J. Boyan and A.W. Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995.

[25] J.A. Boyan. Least-Squares Temporal Difference Learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56, 1999.

[26] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming using Function Approximators*. Taylor & Francis CRC Press, 2010.

[27] E.W. Cheney. *Introduction to approximation theory*. Amer Mathematical Society, 1998.

[28] M.P. Deisenroth, C.E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.

[29] A.P. Dempster, N.M. Laird, D.B. Rubin, et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[30] K. Doya. Reinforcement learning in continuous time and space. *Neural Comput.*, 12(1):219–245, 2000.

[31] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley and Sons, Inc, New-York, USA, 2001.

[32] G.M. Edelman. *Neural Darwinism: The theory of neuronal group selection*. Basic Books New York, 1987.

[33] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208, 2005.

[34] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[35] M. Figueiredo. On Gaussian Radial Basis Function Approximations: Interpretation, Extensions, and Learning Strategies. *In Proc. of the International Conference on Pattern Recognition*, 2:618–621, 2000.

[36] Z. Ghahramani and M. Jordan. Supervised learning from incomplete data via an em approach. In *Proceeding of Advances in Neural Information Processing Systems (NIPS'94)*, pages 120–127. San Mateo, CA: Morgan Kaufmann, 1994.

[37] G.J. Gordon. Stable Function Approximation in Dynamic Programming. Technical Report CS-95-130, CMU, 1995.

[38] J. Johns and S. Mahadevan. Constructing basis functions from directed graphs for value function approximation. In *Proceedings of the 24th international conference on Machine learning*, pages 385–392, 2007.

[39] L.P. Kaelbling. *Learning in embedded systems*. The MIT Press, 1993.

[40] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence*, 4(1):237–285, 1996.

[41] S. Kakade. A natural policy gradient. In *Advances in neural information processing systems*, volume 2, pages 1531–1538, 2002.

[42] S. Kalyanakrishnan and P. Stone. An empirical analysis of value function-based and policy search reinforcement learning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 749–756, 2009.

[43] HJ Kushner and GG Yin. *Stochastic approximation algorithms and applications*. Springer-Verlag, 1997.

[44] S.M. LaValle. *Planning algorithms*. Cambridge Univ Pr, 2006.

[45] D.J.C. MacKay. Comparison of approximate methods for handling hyperparameters. *Neural Computation*, 11(5):1035–1068, 1999.

[46] A. Mccallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, The University of Rochester, 1996.

[47] F.S. Melo, S.P. Meyn, and M.I. Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*, pages 664–671. ACM, 2008.

[48] I. Menache, S. Mannor, and N. Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.

[49] D. Michie and RA Chambers. BOXES: An experiment in adaptive control. *Machine intelligence*, 2(2):137–152, 1968.

[50] J.D.R. Millán, D. Posenato, and E. Dedieu. Continuous-action Q-learning. *Machine Learning*, 49(2):247–265, 2002.

[51] T. Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), October 1997.

[52] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Comput.*, 1(2):281–294, 1989.

[53] A.W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 333–337, 1991.

[54] A.W. Moore and C.G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3):199–233, 1995.

[55] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine learning*, 49(2):291–323, 2002.

[56] R. Neal and G. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Proceedings of the NATO Advanced Study Institute on Learning in graphical models*, pages 355–368, 1998.

[57] A.Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[58] Steven J. Nowlan. *Soft competitive adaptation: neural network learning algorithms based on fitting statistical mixtures*. PhD thesis, Pittsburgh, PA, USA, 1991.

[59] D. Ormoneit and Ś. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2):161–178, 2002.

[60] C.H. Papadimitriou and J.N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[61] I.P. Pavlov and G.V. Anrep. *Conditioned reflexes: An investigation of the physiological activity of the cerebral cortex*. Dover, 1960.

[62] T.J. Perkins and D. Precup. A convergent form of approximate policy iteration. *Advances in neural information processing systems*, pages 1627–1634, 2003.

[63] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[64] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003.

[65] J. Piaget. The origins of intelligence in children New York. *International Universities Press, Inc*, 1952.

[66] J.M. Porta and E. Celaya. Reinforcement learning for agents with many sensors and actuators acting in categorizable environments. *Journal of Artificial Intelligence Research*, 23(1):79–122, 2005.

[67] J.R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[68] C.E. Rasmussen and C.K.I. Williams. *Gaussian processes for machine learning*. Springer, 2006.

[69] S.I. Reynolds. Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 783–790, 2000.

[70] M. Riedmiller. Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the European Conference on Machine Learning*, pages 317–328, 2005.

[71] M. Riedmiller. Neural Reinforcement Learning to Swing-up and Balance a Real Pole. In *Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3191–3196, 2005.

[72] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE international conference on neural networks*, pages 586–591, 1993.

[73] A. Rottmann and W. Burgard. Adaptive autonomous control using online value iteration with Gaussian processes. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 3033–3038, 2009.

[74] A. Rottmann, C. Plagemann, P. Hilgers, and W. Burgard. Autonomous blimp control using model-free reinforcement learning in a continuous state and action

space. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1895–1900, 2007.

[75] GA Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

[76] S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice hall, 2009.

[77] Masa-aki Sato and Shin Ishii. Reinforcement learning based on on-line em algorithm. In *Proceedings of the 1998 conference on Advances in neural information processing systems (NIPS'99)*, pages 1052–1058, Cambridge, MA, USA, 1999. MIT Press.

[78] Masa-Aki Sato and Shin Ishii. On-line em algorithm for the normalized gaussian network. *Neural Comput.*, 12(2):407–432, 2000.

[79] B.F. Skinner. *Science and human behavior*. Free Pr, 1965.

[80] M. Song and H. Wang. Highly efficient incremental estimation of gaussian mixture models for online data stream clustering. In *Proceedings of SPIE: Intelligent Computing: Theory and Applications III*, pages 174–183, Orlando, FL, USA, 2005.

[81] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[82] R.S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD dissertation, University of Massachusetts, Department of Computer Science, 1984.

[83] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[84] R.S. Sutton and A.G. Barto. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological review*, 88(2):135–170, 1981.

[85] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 2000.

[86] C. Szepesvári and W.D. Smart. Interpolation-based Q-learning. In *Proceedings of the twenty-first international conference on Machine learning (ICML'04)*, pages 791–798, Bannf, Canada, 2004.

[87] G. Tesauro. Practical issues in temporal difference learning. *Machine learning*, 8(3):257–277, 1992.

[88] E.L. Thorndike. The law of effect. *The American Journal of Psychology*, 39(1):212–222, 1927.

[89] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[90] C.F. Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22(3-4):251–281, 1997.

[91] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[92] C. Watkins. *Learning from delayed rewards*. PhD thesis, Ph. D Dissertation. Department of Psychology, University of Cambridge, Cambridge, UK, 1989.

[93] S. Whiteson. *Adaptive representations for reinforcement learning*. Springer Verlag, 2010.

[94] F. Wörgötter, A. Agostini, N. Krüger, N. Shylo, and B. Porr. Cognitive Agents - A Procedural Perspective relying on the Predictability of Object-Action-Complexes (OACs). *Robotics and Autonomous Systems*, 57(4):420–432, 2009.