UNIVERSITAT POLITÈCNICA DE CATALUNYA

Computer Science Department

PhD in *Artificial Intelligence*

Regularized Approximate Policy Iteration using kernel for on-line Reinforcement Learning

PhD Thesis of

Gennaro Esposito

gesposit @ cs.upc.edu

PhD Supervisor : Prof. Mario Martín (UPC)

2015

En una isla solitaria desembarca un fugitivo
en busca de libertad y esperanza que encuentra a través del amor.
Un amor que es incapaz de mirarle a los ojos.
Que ignora que es amado.
Y la ilusión es la que le da vida, a través de la muerte
Es la ilusión de Morel

Contents

List of Figures

List of Tables

Thesis overview

Framework

By using Reinforcement Learning (RL), an autonomous agent interacting with the environment can learn how to take adequate actions for every situation in order to optimally achieve its own goal. RL provides a general methodology able to solve uncertain and complex decision problems which may be present in many real-world applications. RL problems are usually modeled as a Markov Decision Processes (MDPs) deeply studied in the literature. The main peculiarity of a RL algorithm is that the RL agent is assumed to learn the optimal policies from its experiences without knowing the parameters of the MDP. The key element in solving the MDP is learning a value function which gives the expectation of total reward an agent might expect at its current state taking a given action. This value function allows to obtain the optimal policy. Hence, the agent will choose the action to take by using this value function. RL algorithms estimate the value function by observing data generated on-line by the interaction with the environment. Various value function estimation techniques have been proposed. Among others, one well known and used technique is the Temporal Difference (TD) algorithm [88]. However, in case of problems with very large or even continuous state spaces, generalization seems to be necessary. Common approach to generalization in RL is to use function approximation methods to estimate the value function. Approaches to generalization can be classified as parametric or non-parametric depending on the assumption that the function can be described by using an "a priori" fixed set of parameters or not.

On one hand, several parametric methods have been considered in the literature for RL generalization, such as neural networks [41], linear architectures [48], [95], wavelets [55], [54] and splines [94]. The main advantage of these methods is that they have fast and easy learning mechanisms. However, they present the inherent problem that in some cases the solution of the problem might not be expressed with the given architecture and the number of chosen parameters. So, while RL algorithms require finding the optimal solution or at least converge to some good enough policy, this may not be guaranteed (in general) by parametric methods.

On the other hand, non-parametric methods like decision trees [26] or kernel-based methods like Support Vector Machines (SVM) ([81], [23]) or Gaussian processes [72] for pattern classification and regression, have been used in RL domain. The main advantage is that these methods do not depend on a pre-fixed set of parameters and it might be possible to obtain the optimal solution. However, the on-line feature of RL demands that the function approximation method used should be incremental and non-monotonic, features that (in general) might not be fulfilled by non-parametric methods.

In this thesis we will focus in value function approximation using Support Vector Regression (SVR) which is the regression method for the SVM paradigm. A model-free approach for approximate value iteration is presented in [68] which uses kernel smoothers. Some authors [85], [84], [53] developed specialized kernels exploiting state space manifold structure while others [29] used Gaussian processes for computing the value function by an approximate value iteration algorithm. Tobias and Daniel [93] proposed a Least Squares Temporal Difference (LSTD) approach based on SVM. In [13] a kernel-based approximate Dynamic Programming (DP) using Bellman Residual Minimization (BRM) is presented aiming at solve the cost to go function for approximate DP problems. However, their method is not model-free as it requires the knowledge of the transition probability function and no statistical guarantees about the convergence are presented. Moreover, it is also a batch algorithm and cannot be directly applied to RL problem. [51] apply an incremental SVR to the approximation of action value function in an actor-critic basis without Policy Iteration (PI). However the paper suffer for a lack of rigorous statistical analysis of their method. Dietterich [30] investigated a linear programming approach using kernels and DP approximating the value function with SVR and minimizing the TD error. However, their approach is unable to solve online learning problem because it estimates a value function after visiting all states using a uniform random behavior policy. In their approach, an RL agent can neither cumulate its experiences continuously nor adapt itself to the changing environment readily. On the contrary, an RL agent should be able to learn from data obtained sequentially from interaction with the environment.

In all the reported works, while somehow the algorithms may be attractive for some specific problems or having interesting practical implementations, the lack of theoretical guarantee is a fundamental flaw.

Contributions

In this thesis we study the capacity of SVR using kernel methods to adapt and solve complex RL problems in large or continuous state space. SVR can be studied using a geometrical interpretation in terms of optimal margin [23] or can be seen as a regularization problem given in a Reproducing Kernell Hilbert Space (RKHS) [82]. SVR have good properties over the generalization ability and as they are based a on convex optimization problem, they do not suffer from sub-optimality. SVR are non-parametric showing the ability to automatically adapt to the complexity of the problem. Accordingly, applying SVR to approximate value functions sounds to be a good approach. One important aspect which makes SVR suitable for RL problems is that they can be solved both in batch mode when the whole set of training sample are at disposal of the learning agents or incrementally. Incremental SVM was originated by the work of [70] while for SVR by [57]. Within SVR incrementality enables the addition or removal of training samples very effectively. Basically incremental SVR finds the appropriate Karush-Kuhn-Tucker (KKT) conditions for new or updated data by modifying their influences into the regression function maintaining consistence in the KKT conditions for the rest of data used for learning. In RL problems an incremental SVR should be able to approximate the action value function leading to the optimal policy. Accordingly, computation load should be lower, learning speed faster and generalization more effective than other existing methods.

The overall contribution coming from of our work is to develop, formalize, implement and study a new RL technique for generalization in discrete and continuous state spaces with finite actions. Our method uses the Approximate Policy Iteration (API) framework with the BRM criterion which allows to represent the action value function using SVR. This approach for RL is the first one we know using SVR compatible to the agent-interaction-with-the-environment framework of RL (contrasting with batch approaches), which shows his power by solving a large number of benchmark problems, including very difficult ones, like the bicycle driving and riding control problem. In addition, unlike most RL approaches to generalization, we develop a proof finding theoretical bounds for the convergence of the method to the optimal solution under given conditions.

We called the proposed algorithm $API - BRM_\varepsilon$ which estimates the action value functions by solving an optimization problems with regularized objective functions in RKHS where it easy to choose the kernel function and consequently the function space. $API - BRM_\varepsilon$ is an instance of API algorithm built using BRM [8] The idea is that a small Bellman Error (BE) may yield a good approximation to the policy evaluation function, which in turn may imply a good final performance. In particular, we demonstrated how the problem of finding the optimal policy minimizing the Bellman Residuals (BR) can be cast as a regression problem using SVR and an appropriate RKHS. An important novelty is represented by the incremental SVR based algorithm in the context of API using BRM. Some novelties also come as technical contributions represented by the extension from the square loss to the $\varepsilon-$insensitive loss and presented in lemma 4.3, theorem 4.1 and proposition 3.2. One important result is the finite-sample bound on the performance of the resulting policy depending on the mixing rate of the trajectory, in the approximation power of the function set, in the capacity of the function set and finally on the discounted concentrability of the future state distribution. This work is the first attempt to give some theoretical justification of using SVR non-parametric regularized optimization problem, to solve the approximation problem of action value function in RL. One major technical difficulty of the proof is that one has to deal with dependent samples. Novelty of this work is the finite-sample analysis using a β-mixing case for the batch algorithm which has relevance in practical online learning. The main condition we ask is that the trajectory should be sufficiently representative and rapidly mixing. The mixing condition is essential for efficient learning and in particular we use the exponential β-mixing condition. The algorithm eventually converges to the optimal policy using β-mixing distributed data samples. We also require that the states in the trajectory follow a stationary distribution. Another contribution of this work is the experimental analysis of a non-parametric approximation algorithm for the generalization problem in RL using PI and kernel methods. Experimental evidence and performance of $API - BRM_\varepsilon$ for well known RL benchmarks are also presented.

Some interesting properties of $API - BRM_\varepsilon$ algorithm are:

- $API - BRM_\varepsilon$ is quite efficient for problems where sampled experience is sparse. The algorithm is based on API, a very powerful framework met with success mostly among planning problems. It also open new research directions for the use of kernel based API in the context of learning.

- $API - BRM_\varepsilon$ is a model free algorithm that can be easily adapted to model based learning. In absence of generative models samples must be collected from the actual

process in real-time then the algorithm may work in an online fashion. Eventually when there is a generative model available the offline variant of the algorithm can be used.

- $API - BRM_\varepsilon$ is an API algorithm which makes a good use of function approximation implicitly constructing an approximate model using kernels. The algorithm place approximation directly in the value function and uses samples to perform the necessary operations without going through any kind of model eliminating a potential source of error.

- $API - BRM_\varepsilon$ solving for the SVR a convex optimization problem, does not suffer from sub optimality looking for the global optimal solution of the approximation problem. As non-parametric learning method has the ability to automatically adapt to the complexity of the problem. Both properties rely on the use of SVR with $\varepsilon-$insensitive loss function, which is essentially a convex quadratic programming optimization problem.

- $API - BRM_\varepsilon$ uses incremental SVR which allows for the estimation and approximation of state action value functions in RL. PI can be done implicitly any time a new experience is obtained. $API - BRM_\varepsilon$ complexity strongly depends on the cost one has to pay in order to solve the SVR which is essentially a quadratic problem optimization. SVR can be solved in batch mode when the whole set of training sample are at disposal to the learning agents or in incremental mode enabling the addition or removal of training samples effectively.

- The approach taken by $API - BRM_\varepsilon$ makes full use of all samples at once either they are i.i.d. or strongly mixing. In contrast traditional RL algorithms use stochastic approximation where each sample is processed only once and contributes with small changes. Usually a very large number of samples is required. The experience replay technique of storing samples and making multiple passes over them might be used to partially overcome this problem which is no longer necessary with $API - BRM_\varepsilon$.

- In traditional RL algorithms the accuracy of the approximation at different states depends on the time and order state visitation. If the learning rate is high the algorithm risks oscillatory or divergent behavior. If learning rate is kept small the learning becomes extremely slow. $API - BRM_\varepsilon$ has no risk of overshooting, oscillation, or divergence because it has no learning parameters to tune up and does not take gradient steps. Compared to standard PI there are some similarities and some significant differences.

- $API - BRM_\varepsilon$ has an alternative representations from batch to incremental, from offline to online, showing effective generalization ability through SVR which makes use of the Structural Risk Minimization theory and his extension to mixing processes.

Thesis Roadmap

This document is organized as follows:

- in Chapter 1 we describe the Reinforcement Learning problem starting from the finite Markov Decision Process and analyzing Dynamic Programming and Model Free algorithms and convergence properties.
- Chapter 2 focuses on the Generalization problem in RL in the contest of continuous MDP and also introducing parametric and non-parametric value function approximation architectures.
- Chapter 3 introduces the kernel based API problem and the tools necessary to understand our method which is finally presented in Chapter 4 we analyze our method describing the properties of the solution.
- In Chapter 5 we focus on the statistical theoretical guarantees for the convergence of $API - BRM_\varepsilon$ providing a bound using mixing processes in regularized regression.
- Chapter 6 reports results and performance in the experiments realized using $API - BRM_\varepsilon$ on several well known RL benchmarks.
- Finally conclusion and future work are presented. To complement the presented material we include an Appendix containing:
- Chapter A describes kernel methods from a general point of view and its connection with regularization methods.
- Chapter B reports some generalization bounds coming from the statistical learning theory.
- Chapter C illustrates SVM and SVR analyzing batch and incremental solutions in the geometrical formulation.

Chapter 1

Reinforcement Learning

1.1 Introduction

Machine Learning (ML) research area is part of the broader field of Artificial Intelligence (AI). ML algorithms allow computer to learn from observed data. Such kind of goals can be fulfilled building a step by step procedure followed by the computer to reach the desired result. The problem within this approach is that one should know a solution to the problem and eventually implement it. Moreover the approach requires the programmer to foresee possible situations the program may encounter which is sometimes not feasible. Often it is easier to use ML algorithms based on observations allowing the computer to find itself solutions. Learning continues using the experience and eventually might find solutions unknown to the programmer. In this work we discuss some advances related to RL which may be considered a subset of ML. The material for this section was referenced from [88],[77], [11], [97].

In RL the focus is primary on learning algorithms by means of interaction with an environment using a trial-and-error mechanism. RL lies between supervised learning (learning with an expert who provides examples of correct behavior) and unsupervised learning (learning with no assistance; trying to find structure in the data). In RL the reward signal reinforces good decision making and penalizes bad ones. As such, RL represents a large class of problems, where an interactive agent learns to act in its environment by trial-and-error. There are two classes of problems in the context of RL: prediction problems, in which the agent learns to predict the long-term accumulated reward of a fixed decision policy for different starting states; and, control problems, in which the agent learns a decision policy that maximizes the long-term cumulative reward for any starting state. Control is, in general, harder than prediction. However, many times, prediction and control are interleaved; in order to achieve better control, the agent has to be able to predict the outcome of its current control policy. Basically there are two main approaches to reinforcement learning. Model-based learning (or, *indirect control*) uses samples to learn a model of the process and then calls one of the standard planning algorithms on the learned model to generate a good decision policy. However, building an accurate model of a complex process from samples can be very difficult and is certainly error-prone. Model-free learning (or, *direct control*), on the other hand, uses the samples to learn a good decision making policy directly without ever building a model. The focus of this

thesis is on model-free RL for control problems. Despite the amount of recent research in this area, existing algorithms that use exact representations have not been widely applied on real-world problems, mainly because the required resources grow extremely quickly as a function of the size of the problem. As a result, exact (but impractical) solutions are commonly abandoned in favor of approximate (but practical) solutions. This has sparkled interest in approximate methods. However, it has also raised the question of stability, beyond the question of efficiency. Research on efficient and stable approximate methods has focused mainly on the prediction problem, leaving the problem of efficient and stable methods for control a largely open question.

1.2 Reinforcement Learning Definitions

ML is about designing and automatically learning algorithms from available data and can be thought as three different categories: *Unsupervised Learning* where the objective is to find patterns, regularities or clusters a in a set of unlabeled data. Examples include dimensionality reduction and clustering. *Supervised Learning:* where the goal is to find a function mapping inputs to outputs and with labeled data, examples may include handwritten text and prediction or classification of handwritten numerals; *Reinforcement Learning:* here the task is to find optimal strategies of behavior for an artificial agent using only reinforcement signals indicating how the agent is able to perform. Examples include robotics, control tasks and games. Hence, RL is a subfield ML whose task is essentially learning functions from data. RL can be interpreted as being somewhere between these supervised and unsupervised learning. Typically one assumes there exists some measure which can be observed informing how well the agent is doing his job. Hence, the agent obtain more information by trial and error. The method is appealing because might be easier to construct a reinforcement scheme than building good policies from scratch. Humans sometimes learn using a mechanism resembling more RL rather than supervised learning. Henceforth the idea of using punishments and incentives is often encountered in real life. A learning agent has a set of sensors to observe the state of its environment and a set of actions it can perform to alter this state. Basically the agent task is to learn a control strategy called a policy for choosing actions in order to achieve its goals. In RL we assume that the agent goals can be defined using a reward function assigning measurable numerical value to each distinct actions it may take in each distinct state. The reward function may be built into the agent or only known to an external expert who provides the reward for some actions performed by the agent. Hence we may think that an agent in RL consists of a learning algorithm to adapt to policy of behavior. Most other elements like the body of the agent or the source of the reinforcements, can be thought as part of the environment. The behavior policy changes with use of the learning algorithm and can be thought as a function mapping situations to actions.

1.3 Markov Decision Processes

RL can be used to find optimal solutions for many problems, but of course these problems should be modeled in a way that the algorithms can be applied.

1.3.1 Finite MDP

Definition 1.1. *(**Finite MDP**) A **Finite MDP** can be defined as a tuple (S,A,P,R,γ), with the following definitions for its contents:*

- *S is a finite set of states, where $s_t \in S$ denotes the state the agent is in at time t.*
- *A is a finite set of available actions in state s, where $a_t \in A$ denotes the action the agent performs at time t.*
- *$P : S \times A \times S \to [0,1]$ is a transition function where $P^{s'}_{s,a}$ denotes the probability of ending up in state s' when performing action a in state s.*
- *$R : S \times A \times S \to \mathbb{R}$ is a reward function where $R^{s'}_{s,a}$ denotes the expected reward when the agent transitions from state s to state s' after performing action a. The actual reward that is witnessed by the agent after performing action a_t in transition to state s_{t+1} may contain noise and is denoted as r_{t+1}, where $\mathbb{E}[r_{t+1}|(s,a,s') = (s_t,a_t,s_{t+1})] = R^{s'}_{s,a}$*
- *$\gamma \in [0,1)$ is a discount factor.*

The *MDP* is sometimes called the environment to contrast it with the inner workings of the agent. An agent in RL is usually assumed to be very simple, consisting mainly of an action selection policy $\pi : S \times A \to [0,1]$, where $\pi_t(s,a)$ denotes the probability that the agent will select action a to perform if it is in state s at time t.

A *deterministic policy* π is a policy where $\forall s : \pi(s,a) = 1$ for exactly one $a \in A$ and $\pi(s,b) = 0$ for all other $b \in A$ while a *stationary policy* is a policy that does not change over time, i.e. where $\forall t : \pi_t = \pi$ With a slight abuse of notation, we will use $\pi(s)$ to refer to the probability distribution or the probability mass function of the actions in state s. We will then use $a \sim \pi(s)$ to indicate that action a is chosen according to the probability function in state s. One interaction of an agent with a *MDP* consists of the agent observing the present state s_t and choosing an action a_t to perform according to its policy. The *MDP* then transitions to a new state s_{t+1} with probability $P^{s_{t+1}}_{s_t,a_t}$ and returns a reward r_{t+1} with expected value $R^{s_{t+1}}_{s_t,a_t}$. Usually, any physical presence of the agent itself is also part of the environment.

MDP by definition fulfill the *Markov property*: a stochastic process has the Markov property if the conditional distribution of the next state of the process depends only on the current state of the process. For an *MDP*, this property implies that the transitions P and the rewards R do not depend on the states the agent visited in the past. Formally this can be expressed as $\mathbb{E}[s_{t+1}|s_0,a_0,...,s_t,a_t] = \mathbb{E}[s_{t+1}|s_t,a_t]$ and $\mathbb{E}[r_{t+1}|s_0,a_0,...,s_t,a_t] = \mathbb{E}[r_{t+1}|s_t,a_t]$.

Problems in which the Markov property does not hold for the observable states and actions are modeled with Partially Observable *MDPs*. In *POMDPs* the assumption is that there is some *MDP* describing the problem while the agent cannot observe the full state. As a result the chain of observations might not fulfill the Markov property while the underlying process is still Markovian. Most RL algorithms assume Markov property while there is a separate set of algorithms especially designed for *POMDPs*. In a *deterministic MDP* transitions and reward are deterministic and $P^{s'}_{s,a} = 1$ for exactly one state s' while

become zero for all other states and $\forall\, t\ :\ r_{t+1} = R^{s^{t+1}}_{s_t,a_t}$. In a *stationary MDP* every element in the tuple (S,A,P,R,γ) is fixed and independent on the time step An example is a problem where there is more than one learning agent and the agents can only observe the behavior of the other agents. Taking the perspective of any one agent and assuming the other agents can also affect the environment, the transition and reward functions can then change over time.

Many RL tasks are episodic in nature. Whereas in continual learning tasks, the RL agent is placed in some possibly random starting state, and is then allowed to wander indefinitely. In episodic tasks the state space is assumed to contain a terminal (or absorbing) state into which the agent is assured to transition after a finite possibly random duration of time. In episodic RL tasks when such a state is reached, the episode terminates and the agent is placed in a new usually random state to begin another episode. It is then usually assumed that the *MDP* reaches a terminal state with probability one in the limit otherwise the value of a state can in principle be unbounded. Note that we only need a single terminal state for any episodic problem, since multiple terminal states would be indistinguishable from each other. If terminal states with different values are desired, one can model this as an equivalent MDP with a single terminal state with different rewards on the incoming transitions. The definition of an episodic MDP does not imply that an agent actually reaches the terminal state, whether this happens may depend on the policy. In a large portion of the RL literature, it is assumed that a *MDP* is given and then the goal is to find a suitable algorithm to solve it. However, any problem must be modeled as an *MDP* before it can be solved while there is no general method do it. Often best model will be dependent on which algorithm one likes to use and therefore the best algorithm for the job depends on the model. Algorithms performance depends on the number of available actions, so sometimes one limits this number in the modeling phase. Similar to the state space, one may encounter the choice between them using a model with continuous or discrete actions. A large majority of algorithms in the field of RL assume discrete, finite action sets. We will see later on how to deal with continuous space MDP while keeping the finite action hypothesis whole over our work. The transition function may be the result of interactions with a physical or simulated system and usually arises quite naturally when the models for the state and action spaces are selected. However, these transitions can also be a consideration in the modeling phase. A very important consideration in the modeling phase is the choice of a reward function. Remarkably, the reward function in the large majority of RL literature is assumed to be given. Some problems have properties that leads naturally to a reward function. When selecting a reward function it is important to make sure that (together with the discount factor) it has the following two properties:

- optimizing the discounted cumulative reward should result in the intended behavior and

- reward function should have the Markov property. While this may seems trivial in some practical cases in which this requirement is not met.

The discount factor determines the value of an action or a state, together with the reward function. Most RL algorithms optimize the discounted cumulative reward. Even if we compare different algorithms in terms of their performance on average rewards per step, instead of in terms of how they optimize the discounted rewards, in many cases algorithms that use the discounted cumulative reward paradigm outperform algorithms that explicitly try to optimize the average rewards.

1.3.2 Value Function

In RL, the value of a state or an action plays a central role. In some cases, one may be interested in the value of a certain policy of behavior in a given problem. In most cases, one wants to optimize the total return in terms of cumulative rewards. In this section, we formalize the notion of value in terms of the structure of the *MDP*. This allows us to talk about the value of a state, action or policy unambiguously in the remainder of this dissertation. In the next section we discuss some methods to learn these values, but first we introduce our notation and definitions.

When we are given an *MDP* and a policy π, it is possible to determine the value V^π of following this policy when starting in state s. Given the policy π this value is defined as the cumulative discounted reward:

$$V^\pi(s) \quad = \quad \mathbb{E}[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i} | s_t = s, \pi] \tag{1.1}$$

Sometimes one is interested in the value of some given policy, but more often we will be interested in maximizing the value. The optimal value of a state is the maximal possible value that can be obtained with any policy. We denote this optimal value with V^*, where $V^*(s) = \max_\pi V^\pi(s)$. The goal is then to find the optimal, stationary policy π^* that maximizes the value for each state. By definition $V^{\pi^*}(s) = V^*$. The value defined in section 1.3.2 can also be defined recursively:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} | s_t = s, \pi] \\ &= \mathbb{E}[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, \pi] \\ &= \sum_a \pi(s,a) \cdot \sum_{s'} P^{s'}_{s,a} \cdot \left(R^{s'}_{s,a} + \gamma V^\pi(s') \right) \end{aligned} \tag{1.2}$$

This then specifies a linear system of $|S|$ quations that can be solved to find the value of each state. Naturally, as we are dealing with finite MDP, the sets of states and actions are of finite size. Similarly, the optimal value function can be described with a recursive definition:

$$V^*(s) = \max_a \sum_{s'} P^{s'}_{s,a} \cdot \left(R^{s'}_{s,a} + \gamma V^*(s') \right) \tag{1.3}$$

Unfortunately, this system of Equations is non-linear due to the max operator which makes it harder to solve analytically. section 1.3.2 is known as the *Bellman Optimality Equation*. V^* can be found solving section 1.3.2 and the optimal policy is given by

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s'} P^{s'}_{s,a} \cdot \left(R^{s'}_{s,a} + \gamma V^*(s') \right) \tag{1.4}$$

while using V^π a *policy improvement* can be evaluated as

$$\pi'(s) = \arg\max_{a \in A} \sum_{s'} P^{s'}_{s,a} \cdot \left(R^{s'}_{s,a} + \gamma V^\pi(s') \right) \tag{1.5}$$

One may also iteratively performs policy evaluation followed by policy improvement, a sequence of policies that are guaranteed to converge to the optimal policy π^* is obtained [11].

1.3.3 Action Value Function

Similar to state values, we can look at the value of a certain action in a state. A value of an action a in a state s under a policy π is defined as the expected cumulative discounted reward when performing that action and following policy π afterward. For historical reasons, action values are often called $Q-values$ and we denote these $Q(s,a)$, which is defined as

$$Q^{\pi}(s,a) = \mathbb{E}\Big[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i} | s_t = s, a_t = a\Big] \tag{1.6}$$

which can also be defined recursively:

$$Q^{\pi}(s,a) = \sum_{s'} P^{s'}_{s,a} \cdot \Big(R^{s'}_{s,a} + \gamma \sum_{a'} \pi(s',a') \cdot Q^{\pi}(s',a')\Big) \tag{1.7}$$

Storing action values requires more space than storing state values: $|S \times A|$ compared to $|S|$. However, action values have the important advantage over state values that when they are found the optimal policy can be easily constructed, simply by selecting the action with the highest value in each state. In contrast, when only state values are known one must solve section 1.3.2 for each state in order to find this optimal action. This difference is especially important for model-free algorithms that approximate the state or action values, since then section 1.3.2 can not be solved since P and R are not known.

1.4 Dynamic Programming

Dynamic programming (DP)is a collection of methods that assumes the knowledge of a full model of the environment and use this model to determine values or optimal policies. DP techniques can be seen as precursors of many of the RL techniques. In this section, we shortly discuss some of the different methods to solve *MDPs*. We will only look at methods that in some way use state or action values as defined in the former section. In principle, it is also possible to search for an optimal policy directly, if some proper criterion to be optimized is formulated. The two main DP techniques we will discuss are *Value Iteration* (VI) and *Policy Iteration* (PI).

In DP is that we can iteratively apply some update to the values of states or actions which results in a better approximation of the true state. The obtained approximation can be better because the resulting value is closer to the true value of the current policy, as defined by the Bellman equations, or because it is closer to the optimal value, as defined in the Bellman optimality equations. When considering simulation based RL, we may introduce updates that only improve the values in expectancy instead of on each step. All these updates have in common that they adapt the value function. These updates are therefore operators mapping functions to functions. We can define the operator T^{π} : $\mathbb{R}^{\mathbb{S}} \to \mathbb{R}^{\mathbb{S}}$ as the mapping

$$(T^{\pi} V^{\pi})(s) = \sum_{a} \pi(s,a) \cdot \sum_{s'} P^{s'}_{s,a} \cdot \Big(R^{s'}_{s,a} + \gamma V^{\pi}(s')\Big) \tag{1.8}$$

where $\mathbb{R}^{\mathbb{S}}$ denotes a space of bounded real-valued functions over a set $\mathbb{S}$. A *fixed point* of an operator T is a function f, such that $T f = f$. For operator T^{π} as defined in section 1.4 the fixed point is V^{π}, since by definition

$$(T^{\pi} V^{\pi})(s) = V^{\pi}(s) \tag{1.9}$$

or more concisely $T^{\pi} V^{\pi} = V^{\pi}$ and we may also define the symbolic operators

$$(\Pi_a)\hat{Q} = \sum_a \pi(s,a)\hat{Q} \quad (P_s)\hat{Q} = \sum_{s'} P^{s'}_{s,a}\hat{Q}$$

as

$$T^{\pi}V^{\pi} = \Pi_a P_s \Big(R_{s,a} + \gamma V^{\pi} \Big) \tag{1.10}$$

which in principle can be solved as

$$V^{\pi} = \Big(I - \gamma \Pi_a P_s \Big)^{-1} \Pi_a P_s R_{s,a} \tag{1.11}$$

Similarly we can define T^* as

$$\forall \quad s \in S: \quad (T^* V)(s) = \max_a \sum_{s'} P^{s'}_{s,a} \cdot \Big(R^{s'}_{s,a} + \gamma V(s') \Big) \tag{1.12}$$

and note that its fixed point is V^*, since follows that $T^* V^* = V^*$. An operator $T : \mathbb{R}^X \to \mathbb{R}^X$ is called a *contraction mapping* with factor κ if for any two functions $f, g \in \mathbb{R}^X$ the following equation holds

$$\|T f - T g\| \leq \kappa \|f - g\| \tag{1.13}$$

where $\|f\|$ is a sup-norm defined as $sup_{x \in X}|f(x)|$ and the domain X is implicitly given by the function f. If $\kappa = 1$, the mapping is called a non-expansion. If $\kappa < 1$, the contraction has a unique fixed point, defined by the equation $T f = f$. Furthermore, this fixed point is guaranteed to be reached by repeatedly applying the mapping. Consider the distance $\|T^n f - T^n g\|$, where T^n stands for applying the operator n times, in general we have

$$\|T^n f - T^n g\| \leq \kappa \|T^{n-1} f - T^{n-1} g\| \leq \kappa^n \|f - g\| \tag{1.14}$$

Since this holds for arbitrary functions $f \in \mathbb{R}^X$, this also holds when f is the fixed point. By definition of the fixed point, we then have $T^n f = f$ and we get

$$\|f - T^n g\| \leq \kappa^n \|f - g\| \tag{1.15}$$

Since $g \in \mathbb{R}^X$ also arbitrary, this means that for any function $g \in \mathbb{R}^X$, when $\kappa < 1$ we have

$$\lim_{n \to \infty} \|f - T^n g\| = 0 \tag{1.16}$$

This shows that repeatedly applying a contraction mapping on any function in its domain results in convergence to the fixed point of the contraction mapping in the limit. It is easy to prove that if you have a contraction mapping T_1 with factor κ_1 and a contraction mapping T_2 with factor κ_2, the combined operator $T_1 T_2$ is also a contraction mapping, with factor $\kappa_1 \kappa_2$. Contraction mappings are important in the theoretical analysis of RL and dynamic programming algorithms. If we know that an algorithm can be seen as a contraction mapping with a factor lower than one and we know that the fixed point of the contraction is the Bellman optimality equation, then we know that applying the algorithm will leads to optimal values in the limit. If the contraction factor is known, an upper bound on the rate of convergence can be given.

1.4.1 Value Iteration

Value iteration is an iterative algorithm that can be used to find the optimal value function, and thus the optimal policy. The idea is to repeatedly apply the operator T^*, as defined in on some initial finite value function. We know that T^* has a unique fixed point in V^* that is obtained in the limit, if we can show that the operator is a contraction with some factor $\kappa < 1$. This is indeed the case:

$$\begin{aligned} \| \quad & T^*V - T^*V' \| \\ = \quad & \max_{s\in S} |\max_a \sum_{s'} P^{s'}_{s,a}\Big(R^{s'}_{s,a} + \gamma V(s')\Big) - \max_a \sum_{s'} P^{s'}_{s,a}\Big(R^{s'}_{s,a} + \gamma V'(s')\Big)| \\ \leq \quad & \max_{s\in S} \max_a |\sum_{s'} P^{s'}_{s,a}\Big(R^{s'}_{s,a} + \gamma V(s')\Big) - \sum_{s'} P^{s'}_{s,a}\Big(R^{s'}_{s,a} + \gamma V'(s')\Big)| \\ \leq \quad & \max_{s\in S} \max_a \max_{s'} |\Big(R^{s'}_{s,a} + \gamma V(s')\Big) - \Big(R^{s'}_{s,a} + \gamma V'(s')\Big)| \\ \leq \quad & \max_{s'\in S} |\gamma V(s') - \gamma V'(s')| = \gamma \|V - V'\| \end{aligned} \tag{1.17}$$

We identify $\kappa = \gamma$ and therefore value iteration converges to the optimal value function in the limit if $\gamma < 1$. There are some other requirements on the MDP, such as that any state must be reachable. In some specific cases, it can even be guaranteed that the converges occurs after a finite number of iterations. In this algorithm we introduce two possible stopping criteria. The algorithm stops if either the maximal amount of any value is changed in the last iteration is lower than some threshold ε, or if the number of iterations transcends K. After the algorithm has terminated, we can find the (approximate) optimal policy with

$$\pi(s,a) = \begin{cases} 1/M \ \ if \ (W^\pi V)(s,a) = (T^*V)(s) \\ 0 \ \ otherwise. \end{cases} \tag{1.18}$$

where M is the number of actions that are optimal in state s according to V. Value iteration is only applicable if the transition function P and reward function R are known. Even if this is the case, the algorithm can be slow if the state space is too large.

1.4.2 Policy Iteration

Policy iteration uses T^π instead of T^*, with π is the current policy. It can be shown that this operator converges to the fixed point V^π. The idea is to find this value function or an approximation thereof, and then use it to improve the current policy. Then, the value function will no longer be accurate and the procedure repeats itself. The algorithm solve a $|S| \times |S|$ system of linear equations. This can be done in $O(|S|^3)$ time, but this may be too costly if the state space is large. Alternatively, it is also possible to use an iterative method, as in the value iteration algorithm that at iteration k computes

$$\forall s \in S: \quad V_{k+1}(s) = (T^{\pi_k} V_k)(s) = \sum_{s'} P^{s'}_{s,a}\Big(R^{s'}_{s,a} + \gamma V_k(s')\Big) \tag{1.19}$$

for all $s \in S$ until either $\|V_{k+1} - V_k\|$ is smaller than some threshold. If the threshold condition is met at iteration k, $V_{k+1} = V_k$. There are two ways in which the policy evaluation

step can be relaxed. First, only some of the state values may be updated. The most extreme example of this is when only a single state is updated between each two policy improvement steps. Second, if more than one state is updated, one may update the values only partially towards V^{π_k}, for instance by performing a fixed number of iterations. The most extreme example of this is when only one update is performed. PI generates an improving sequence of policies which implies that $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$ for all s and all $k \geq 0$. In practice, PI may converge in a fairly low amount of iterations, although like value iteration it can be prohibitively slow is the state space is reasonably large and it requires knowledge of R and P.

1.5 Model-Free Value Learning

DP algorithms have the major disadvantage that they require a model. A good model of the problem might not be available, although we may have access to a simulated or real physical system interacting with the agent. As a possible extension to DP methods mitigating these issues, one can use *asynchronous updates* not updating the whole state space in every update. Another possibility is to simulate stochastic updates in place of the models.

In *asynchronous updates* the whole MDP is taken as a given input and the algorithms process this to output the policy. In several problems, parts of the state space are more interesting than others. This could be since they might be visited more often by good policies, or eventually rewards are higher. In this case it is worth to consider only updating a subset of the states in each iteration. As extreme case one only updates the value of a single state at a time.

In *stochastic updates* the method gives the result of a single experience of the agent when it performs an action a in a state s. They are usually combined with asynchronous updates, since if we learn by looking at the experience of an agent, they are processed by the learning algorithm corresponding to the state the agent is. Learning algorithm will not necessarily process each experience at once but sometimes process part of the information before a terminal state is reached or the whole state space is visited.

Given a policy π estimating $V^{\pi}(s)$ can be done through equation

$$V^{\pi}(s) = \mathbb{E}[\sum_{i=1}^{T-t} \gamma^{i-1} r_{t+i} | s_t = s, \pi] \tag{1.20}$$

or

$$V^{\pi}(s) = \mathbb{E}[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, \pi] \tag{1.21}$$

assuming episodic MDP such that in at most T steps the terminal state is reached. The two definitions can than be turned into two different updates.

1.5.1 Monte Carlo Updates

In *Monte Carlo updates* one uses the actual sum of discounted rewards obtained until the end of the current episode. The current value of each state s can then be updated with

this value in order to get a better estimate of the value of the policy that was followed. Since the rewards may contain noise and the state transitions stochastic, one to averages different runs from each state. Denoting the start of episode k with t_k and its end with T_k the update to the value of the state then becomes

$$V_{k+1}(s_t) = (1 - \alpha_k(s_t))\, V_k(s_t) + \alpha_k(s_t) \left(\sum_{i=1}^{T_k - t} \gamma^{i-1} r_{t+i} \right) \tag{1.22}$$

$\alpha_k(s) \in [0,1]$ is a *learning rate parameter* and $t_k \leq t < T_k$. In this way section 1.5 was turned into an update using the learning rate to average over the different outcomes we observe from a state in different episodes. In not deterministic MDP exists the possibility that a state is visited more than once in a single policy and to make the update using only the summed rewards after the first visit of the state or the summed rewards after each visit [69]. If we assume for simplicity that no state is visited more than once the return after each episode is clearly an unbiased estimate for V^π. Consider a learning rate of $\alpha_k(s_t) = 1/n_k(s_t)$, where $n_k(s)$ is the number of times state s will have been updated after the current update. Then, for each state $V_k(s)$ will be equal to the average over all the returns after visiting state s in the first k episodes. If the number of times each state is visited increases, the variance of $V_k(s)$ decreases and the in the limit V_k converges: $\lim_{k\to\infty} V_k = V^\pi$. A disadvantage of Monte Carlo methods is that the returns can have considerable variance. If we reach a state that was already visited many times, we might want to use its value, instead of the considerably noisier rewards that actually result from performing the policy from that state onwards.

1.5.2 Temporal Difference Learning

In *Temporal Difference* (TD) learning each state value gets updated with a one step Monte Carlo update, using the actual return and the value of the next state. For Monte Carlo methods section 1.5 was turned into an update while in TD learning we use section 1.5. When the model is not known one samples an experience consisting of a state transition and a reward. Such a sample may be noisy so the update must averages over the samples with a learning rate. This results in TD learning expressed by:

$$V_{t+1}(s_t) = (1 - \alpha_t(s_t))V_t(s_t) + \alpha_t(s_t)(r_{t+1} + \gamma V_t(s_{t+1})) \tag{1.23}$$

which can be viewed as a minimizing the expected TD error $\mathbb{E}[\delta_t | s_t = s]$, where the TD error δ_t is defined as

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{1.24}$$

TD learning can be shown to converge in the sense that $\lim_{t\to\infty} V_t = V^\pi$ as long as the learning rates are chosen such that

$$\forall\, s \in S \quad \sum_{t=0}^{\infty} \alpha_t(s) = \infty \quad \sum_{t=0}^{\infty} \alpha_t^2(s) < \infty \tag{1.25}$$

Such conditions on the learning rates are referred to *Robbins-Monro conditions* [74]. First condition ensures that the whole possible value function space stays reachable no matter how poor some samples are. Second condition ensures that updates become small enough in the limit to ensure stability.

1.5.3 Bias and Variance

We have discussed two different methods to approximate V^π without using a model: Monte Carlo methods and TD learning. We have specified some cases in which TD learning is preferred, such as in non-episodic tasks since Monte Carlo waits until the end of an episode before updating. Monte Carlo estimates are unbiased estimates for V^π which can be easily checked as $V^\pi(s)$ is defined through the expected value of the discounted future return when following policy π from state s. Monte Carlo methods simply sample trajectories and are therefore unbiased. TD learning introduces bias by using the state value of the next state in its estimate. This value will be content less and arbitrary and will therefore introduce a bias in the initialization of the values which is especially true when this state has not been visited yet. Monte Carlo methods can suffer from considerable variance. Each update uses a sample from the whole trajectory following the updating state and therefore variances of the consecutive random rewards are combined. In other words, if we assume the variance of each reward is σ^2, then the variance of a Monte Carlo update is between σ^2 and $\sigma^2/(1-\gamma)^2$ depending on how many steps the episode on average takes from the state under consideration. For TD learning the variance is then always σ^2, since it only uses a single stochastic reward for its updates. Moreover, the bias of all state values decreases under TD learning, ensuring that the updates become better over time. In contrast, in Monte Carlo methods the expected target of the update has the same variance independent on the number of updates that have already occurred. See [97] for more details on the subject.

1.6 Stochastic Iterative Algorithms

Optimization problems or system of equations are often solved by means of iterative algorithms. However in many situations the information needed to carry out the iterations is not directly available and one has to deal with the presence of noise. Stochastic iterative algorithms are variants of deterministic iterative algorithms able to operate in presence of noise. Following [11] suppose we are interested in solving a set of equations of the form

$$(T\,\Psi) = \Psi \tag{1.26}$$

where T is an operator $T : \mathbb{R}^n \to \mathbb{R}^n$ whose form may be not known precisely such that its exact evaluation is difficult. However we may have access to the random variable $\sigma = T\,\Psi + w$ where w is a random noise term. Therefore one possible stochastic setting to solve section 1.6 may be:

$$\Psi_{t+1} = (1-\alpha_t)\Psi_t + \alpha_t\,(T\,\Psi_t + w_t) \tag{1.27}$$

where α is a positive step-size parameter usually chosen to be smaller than one and we are using t to index the different iterations. The resulting algorithm is called a stochastic approximation algorithm where smaller α reduce the sensitivity to the noise w. On the other hand, the algorithm leads slower progress with smaller α and it is also possible using variable step-size. If the algorithm converges to Ψ^*, T is continuous at Ψ^* then the limit must satisfy $(T\Psi^*) = \Psi^*$ so that Ψ^* is the fixed point of the operator T the convergence is guaranteed to take place.

A more concrete setting can be obtained considering that what we really are interested is to solve is the equation

$$\mathbb{E}[g(\Psi, v)] = \Psi \tag{1.28}$$

where g is a known function and the expectation is taken with respect to the conditional distribution $P(v|\Psi)$ of the random variable v. If the distribution $P(v|\Psi)$ is known it is possible to generate random samples $\bar{v}$ of v and use them to estimate $E[g(\Psi, v)]$. Now considering a single sample $\bar{v}$, one way to solve section 1.6 in a stochastic setting can be the use of the Robbins-Monro approximating algorithm of the form

$$\Psi_{t+1} = (1 - \alpha_t)\Psi_t + \alpha_t \, g(\Psi_t, \tilde{v}_t) \tag{1.29}$$

where $\tilde{v}_t$ is a single random variable generated according to the distribution $P(v|\Psi)$. Now recalling the scheme in section 1.6 we may write Robbins-Monro as

$$\Psi_{t+1} = (1 - \alpha_t)\Psi_t + \alpha_t \left(E[g(\Psi_t, v_t)] + g(\Psi_t, \tilde{v}_t) - E[g(\Psi_t, v_t)]\right) \tag{1.30}$$

then we see that putting $T\,\Psi_t = E[g(\Psi_t, v_t)]$ the operator and $w_t = g(\Psi_t, \tilde{v}_t) - E[g(\Psi_t, v_t)$ the zero mean noise term we are dealing with a special case of the algorithm in section 1.6. Assume also that

$$\Psi_{t+1}(i) = \Psi_t(i) \quad \forall t \notin N^i \tag{1.31}$$

where $\Psi_t = (\Psi_t(1), ..., \Psi_t(n)) \quad \forall \Psi_t \in \mathbb{R}^n$ and $\Psi_t(i)$ indicating the ith component. In this framework it is crucial for the step-size to be chosen according to some general assumption in order to guarantee the convergence and in particular we assume the step-size $\alpha_t(i)$ non negative and $\alpha_t(i) = 0$ for $t \notin N^i$ where N^i be an infinite set of integers indicating the set of times at which the update of Ψ_t is performed. The noise term $w_t(i)$ as statistically independent from Ψ_t with variance σ^2. Allowing $\alpha_t(i)$ to decrease to zero the effect of the noise w_t on the variance of $\Psi_{t+1}(i)$ becomes vanishing. Therefore we assume that the following conditions hold with probability one:

$$(a) \quad \sum_{t=0}^{\infty} \alpha_t(i) = \infty \tag{1.32}$$

$$(b) \quad \sum_{t=0}^{\infty} \alpha_t^2(i) < \infty \tag{1.33}$$

Moreover during the update of the algorithm the entire history can be represented by the increasing $\sigma - fields$

$$P_t = \{\Psi_0(i), ..., \Psi_t(i), w_0(i), ..., w_t(i), \alpha_0(i), ..., \alpha_t(i), |i = 1, ..., n\} \tag{1.34}$$

and by construction $P_{t+1} \subseteq P_t$.

1.6.1 Convergence Analysis

In some cases an algorithm may converges to some fixed point. For instance, it is good to know if it can be proven that the value function converges to the optimal value function for a certain algorithm. More general results have been presented in the literature [11]. The following lemma proposed by [80] applies to stochastic processes such as Markov chains obtained by interaction of an algorithm with an MDP:

Lemma 1.1. *[80] Consider a stochastic process* $(\alpha_t, \Delta_t, F_t)$*, where* $(\alpha_t, \Delta_t, F_t) : X \to \mathbb{R}$ *satisfy the equations*

$$\Delta_{t+1}(x_t) = (1 - \alpha_t(x_t))\Delta_t(x_t) + \alpha_t(x_t)F_t(x_t) \tag{1.35}$$

where $x_t \in X$ *and* $t = 0, 1, 2, ...$ *Let* P_t *be a sequence of increasing* $\sigma - fields$ *such that* α_0 *and* Δ_0 *are* $P_0 - measurable$ *and* α_t*,* Δ_t *and* F_t *are* $P_t - measurable$*,* $t \geq 1$*. Assume that the following hold:*

1. *the set X is finite*
2. $\alpha_t(x_t) \in [0,1], \quad \sum_{t=0}^{\infty} \alpha_t(x_t) = \infty,$
   $\sum_{t=0}^{\infty} \alpha_t^2(x_t) < \infty \quad w.p.1 \quad and \quad \forall x \neq x_t : \alpha_t(x_t) = 0,$
3. $\|E\{F_t|P_t\}\| \leq \kappa\|\Delta_t\| + c_t$
4. $Var\{F_t(x_t)|P_t\} \leq K(1 + \kappa\|\Delta_t\|^2)$*, where K is some constant*

where $\|\cdot\|$ *denotes a maximum norm. Then* Δ_t *converges to zero with probability one.*

The proof can be found in [80].

The same result holds if we consider $F_t = (T\Psi_t)(x_t) + w_t(x_t)$ and assuming the following conditions hold with probability one:

$$\mathbb{E}[w_t(x_t)|P_t] = 0 \tag{1.36}$$

$$\mathbb{E}[w_t^2(x_t)|P_t] \leq K(1 + \kappa\|\Psi_t(x_t)\|^2) \tag{1.37}$$

and also assuming that the mapping T is a weighted maximum norm pseudo contraction operator. This means that if we define the weighted maximum norm as:

$$\|\Psi_t\|_\xi = \max_i \left(\frac{|\Psi_t(i)|}{\xi(i)}\right) \tag{1.38}$$

(when all $\xi(i) = 1$ we have the maximum norm denoted as $\|\cdot\|_\infty$) and an operator $T : \mathbb{R}^n \to \mathbb{R}^n$ is a weighted maximum pseudo contraction norm if if there exists some $\Psi_t^* \in \mathbb{R}^n$ and a positive vector $\xi = (\xi(1), ..., \xi(n))$ and a constant $\gamma \in [0,1)$ such that

$$\|(T\Psi) - \Psi^*\|_\xi \leq \gamma\|\Psi_t - \Psi^*\|_\xi \quad \forall \Psi_t \tag{1.39}$$

If all these conditions are present Ψ_t converges to Ψ^* with probability one.

The idea of the lemma 1.1 is usually to apply it with $\mathbb{X} = \mathbb{S} \times \mathbb{A}$, and $\Delta = Q - Q^*$. The maximum norm specified in the lemma can then be understood as satisfying the following equation:

$$\|\Delta_t\| = \max_s \max_a |Q_t(s,a) - Q^*(s,a)| \tag{1.40}$$

Often, the first, second and fourth assumption are easily met and to apply the lemma one only has to show that the contraction in the third assumption holds.

1.7 Learning Action Values

One may also define operators implementing the Bellman equations for action values as $T^\pi : \mathbb{R}^{S\times A} \to \mathbb{R}^{S\times A}$ with fixed point Q^π through

$$(T^\pi Q)(s,a) = \sum_{s'} P^{s'}_{s,a} \cdot \left(R^{s'}_{s,a} + \gamma \sum_{a'} \pi(s',a') \cdot Q(s',a') \right) \tag{1.41}$$

which using the symbolic operators Π_a and P_s can be written as

$$T^\pi Q^\pi = P_s \left(R_{s,a} + \gamma \Pi_a Q^\pi \right) \tag{1.42}$$

which can be solved as

$$Q^\pi = \left(I - \gamma P_s \Pi_a \right)^{-1} P_s R_{s,a} \tag{1.43}$$

Similarly we can define T^* as

$$(T^* Q)(s,a) = \sum_{s'} P^{s'}_{s,a} \cdot \left(R^{s'}_{s,a} + \gamma \max_{a'} Q(s',a') \right) \tag{1.44}$$

T^* is also a contraction mapping with fixed point Q^*. In fact $T^* Q^* = Q^*$ is the Bellman optimality equation for action values and therefore Q^* is a fixed point for T^*. Furthermore:

$$\begin{aligned}
\| & \; T^* Q - T^* Q' \| \\
= & \; \max_{s\in S} \max_{a\in A} |(T^* Q)(s,a) - (T^* Q')(s,a)| \\
= & \; \max_{s\in S} \max_{a\in A} |\sum_{s'} P^{s'}_{s,a} \left(\gamma \max_{a'} Q(s',a') - \gamma \max_{a'} Q'(s',a') \right)| \\
\leq & \; \max_{s'\in S} |\gamma \max_{a'} Q(s',a') - \gamma \max_{a'} Q'(s',a') \Big)| \\
\leq & \; \max_{s\in S} \max_{a} \gamma |Q(s',a') - Q'(s',a')| = \gamma \|Q - Q'\|
\end{aligned} \tag{1.45}$$

This holds for all Q, Q' and since $T^* Q^* = Q^*$ implies

$$\|T^* Q - Q^*\| = \|T^* Q - T^* Q^*\| \leq \gamma \|Q - Q^*\| \tag{1.46}$$

showing the convergence of Q to Q^* under the max norm with a factor of at most γ when applying operator T^*. This means we could implement value iteration with action values to approximate Q^*, with ensured convergence in the limit.

In the value based approaches for solving RL problems, one aims to find the fixed point of the Bellman operator $Q^\pi = T^\pi Q^\pi$ for policy evaluation problem or the Bellman optimality operator $Q^* = T^* Q^*$. To find the optimal value function it is necessary to know how to represent an action value function Q, evaluate $T^\pi Q^\pi$ or $T^* Q^*$ and finally find the calculate the fixed point of T^π or T^* operators. When $S \times A$ is a small finite space and Q can be represented by a finite number of real values. On the contrary we must approximate Q with a given approximating function. This process is called function

approximation which ca ne addressed by approximation and statistical learning theory. A reasonable way to evaluate $T^{\pi}Q^{\pi}$ or $T^{*}Q^{*}$ is to approximately estimate them by random sampling from $P^{s'}_{s,a}$. Finally there are several approaches to find the fixed point of the Bellman operators which will be described later on.

T^{π} is a contraction mapping with factor γ and fixed point Q^{π}. This can be used as the policy evaluation step in a PI algorithm with action values. The policy improvement step can then be accomplished by using a policy that is greedy in the action values.

A *greedy policy* is defined as follows: an action a is greedy in a state s for an action value function Q if $Q(s,a) = \max_{a'} Q(s,a')$. A policy π is greedy when in all states s the action selection probability $\pi(s,a)$ is equal to zero for all non-greedy actions. Multiple actions may be greedy in a given state. Any policy that always selects one of these actions is called greedy, regardless of how the probabilities of selecting these multiple greedy actions are distributed. Although it is convenient to have action values, DP with state action values suffers from the same limitations as DP with state values. The most important of these are the requirement of a model and the computational requirements for larger state and action spaces. In RL algorithms one usually makes a distinction between methods following the policy they are learning called *on-policy* and those learning from behavior generated by different policy called *off-policy*.

1.7.1 Exploration Techniques

We defined the greedy policy as the one that chooses the highest valued action in each state. Such kind of policy can be constructed by setting the probability of all actions corresponding to the highest action value to zero. However, this policy is of limited use in combination with model-free algorithms since it is only able to explore a small part of the state space. In general, one should balance the exploitation of the knowledge obtained which can be done by choosing greedy actions with the exploration of the state and action space in order to find new interesting actions. As a result one needs a policy able to explore. An extreme example the use of a random policy selecting random actions on each step. The traded off between exploration and exploitation can be managed using a parameter $\varepsilon \in [0,1]$, representing the probability of using the random policy. A greedy action is then selected with probability $(1-\varepsilon)$ yielding to the $\varepsilon - greedy$ exploration. Hence, we call a policy $\varepsilon - greedy$ if it selects a random action with probability $\varepsilon \in [0,1]$ and a greedy action with probability $(1-\varepsilon)$.

A requirement for the convergence of some algorithms is that the exploration policy is greedy in the limit with infinite exploration. A policy π is greedy in the limit with respect to some action value function Q_t if for all states s it holds that $\lim_{t\to\infty} \sum_a \pi_t(s,a) Q_t(s,a) = \max_a Q_t(s,a)$ An example of a family of policies greedy in the limit with infinite exploration in a finite MDP is an $\varepsilon - greedy$ policy where $\varepsilon_t = 1/n_t(s_t)^x$ with $n_t(s_t)$ denoting the number of times the state s_t was visited in the first t time steps and $x \in (0,1]$. In practice, $\varepsilon - greedy$ exploration works fine. Eventually the main problem is that it does not differentiate between potentially good actions that are not greedy at the moment and actions that are known to be worthless. A better type of exploration can take into account the values of different actions. An action with a larger value should have a larger probability of being selected and actions that are known to have very low values may be neglected. One way to do this is to use a so-called Boltzmann distribution that yields the

following policy:

$$\pi_t(s,a) = \frac{e^{Q_t(s,a)/\tau}}{\sum_b e^{Q_t(s,b)/\tau}} \tag{1.47}$$

This definition fulfills the desired properties and requirements we have for a policy: the action selection probabilities sum to one and higher values correspond to larger selection probabilities. The τ parameter is called the temperature and regulates how greedy the policy is. When τ decreases towards zero, the policy becomes more greedy and when it increases toward infinity, the policy becomes more random.

Similar to the state values we can transform the Bellman equations for action values into iterative sampled updates as:

$$Q^\pi(s,a) = \mathbb{E}\Big[\sum_{i=1}^{T-t} \gamma^{i-1} r_{t+i} | s_t = s, a_t = a, \pi\Big] \tag{1.48}$$

$$Q^\pi(s,a) = \mathbb{E}\Big[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a, \pi\Big] \tag{1.49}$$

$$Q^*(s,a) = \mathbb{E}\Big[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a, \pi\Big] \tag{1.50}$$

section 1.7.1 can be sampled with Monte Carlo methods whose disadvantage is that all not selected actions in an episode are not updated. This implies that if on average there are M actions per state, one might need M times as many episodes to reach the same accuracy as when using state values. The difference in convergence rate is also dependent on whether there are actions with a zero (or low) probability of being selected which might take quite long to be approximated with reasonable accuracy.

Two TD algorithms can be derived from the Bellman Equation 1.49 and section 1.7.1. Assume Q_t as an increasingly good approximation for Q^π sampling section 1.7.1 and updating with a learning rate $\alpha_t(s_t,a_t) \in [0,1]$ to average out stochastic noise gives:

$$Q_{t+1}(s_t,a_t) = (1-\alpha_t(s_t,a_t))Q_t(s_t,a_t) + \alpha_t(s_t,a_t)(r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1})) \tag{1.51}$$

This update is known as SARSA and was investigated by [76], [42], [91]. The name is because it uses an experience sample consisting of the tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ for each update. It can be shown that this algorithm converges to Q^π, provided that all actions are selected according to a fixed policy π and restrictions on the learning rates. SARSA is an on-policy algorithm. SARSA may also converge to Q^* under a policy slowly becoming greedy [80].

Another possibility proposed by [103] and called Q-learning can be obtained sampling the Bellman Equation 1.50 assuming Q_t is an increasingly good approximation of Q^* as

$$Q_{t+1}(s_t,a_t) = (1-\alpha_t(s_t,a_t))Q_t(s_t,a_t) + \alpha_t(s_t,a_t)(r_{t+1} + \gamma \max_a Q_t(s_{t+1},a)) \tag{1.52}$$

Q-learning can be shown to converge to the optimal value function Q^*. Q-learning learns about the optimal policy regardless of the policy that is being followed. For this reason Q-learning is an off-policy as it learns about one policy while following another policy. For

Q-learning the conditions for convergence are that in every state every action is eventually selected an infinite amount of times and the learning rates are chosen such that

$$\forall\,(s,a) \in S \times A \quad \sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \quad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty \tag{1.53}$$

In a stochastic environment Q-learning will need to sample each action in every state an infinite number of times to fully average out the noise, but in many cases the optimal policy is learned long before the state action values are highly accurate. In a determinist environment it is optimal to set the learning rate equal to one and Q-learning reduces to a form of value iteration, since it performs an asynchronous update equal to section 1.7.1. Although the off-policy capability of Q-learning is appealing, it is also source of instability. Operating in an on-policy mode and updating state action pairs according to the same distribution they would be experienced under π is stable and convergent near the best possible solution. However, if state action pairs are updated according to different distribution, for example generated following the greedy policy, the estimated values diverges to infinity. It is worth noting that the Q-learning update rule section 1.7.1 combines two strong ideas: the notion that the knowledge is made of expectations and the TD which is a method of prediction well suited for online and incremental learning. Basic rule section 1.7.1 can also be derived from the formula of the expected value of a discrete random variable. In fact, the expected value μ of a discrete random variable x with possible oucomes $x_1, x_2, ..., x_n$ and probabilities $P(x_i)$ can be calculated as $\mu = \sum_{i=1}^{n} x_i P(x_i)$ thus for a discrete variable with only two possible values becomes $\mu = (1-\alpha)x_1 + \alpha x_2$ where α is the probability of the second value x_2. Hence considering x_1 as the previously stored expected value μ and x_2 as the new observation x we may write $\mu = (1-\alpha)\mu + \alpha x$. Therefore we come to the conclusion that the learning rate α basically express the probability that the random variable μ get the value of the observation x .

1.7.2 Expected SARSA

Q-learning updates uses the max operator which causes the estimation policy π_t^e to be greedy which guarantees the $Q_t(s,a)$ values converge to $Q^*(s,a)$. The behavior policy of Q-learning is usually exploratory and based on $Q_t(s,a)$. For SARSA the behavior policy and the estimation policy are the same making it an on-policy algorithm. Hence, it will not converge to optimal Q^* values as long as explorations occurs. However, by annealing exploration over time, SARSA will converge to optimal Q^* values just like Q-learning. SARSA's convergence guarantee requires every state to be visited infinitely often, the behavior policy and therefore the estimation policy are stochastic to ensure enough exploration. One of the main effect is that there can be large variance in SARSA updates since a_{t+1} is not selected deterministically. Variance can occur in any TD method due to stochasticity of the environment through the transition probability $P_{sa}^{s'}$ and reward $R_{sa}^{s'}$. Within a model-free method it is possible to reduce stochasticity choosing a low learning factor α. In SARSA other variance may come from policy stochasticity which is known to the agent.

Expected SARSA is an on-policy method which can be seen as a variant of SARSA and exploiting the knowledge of the policy being used to prevent stochasticity. To do that

the update is not based on $Q_t(s_{t+1}, a_{t+1})$ but instead on $\mathbb{E}[Q_t(s_{t+1}, a_{t+1})]$. As SARSA can be viewed as an averaging sample of Bellman Equation 1.49, A closer look shows that sampling of the policy is not necessary. Reward and transition are sampled because one does not want to store a model for the reward and transition function. However, a known policy is required as it is used to select the next action. Policy π at time t is always known and there is no need to sample the action actually taken while one can instead use the update:

$$\begin{aligned} Q_{t+1}(s_t, a_t) &= (1 - \alpha_t(s_t, a_t))Q_t(s_t, a_t) + \\ &\alpha_t(s_t, a_t)(r_{t+1} + \gamma \sum_a \pi_t(s_{t+1}, a)Q_t(s_{t+1}, a)) \end{aligned} \tag{1.54}$$

First mention of this algorithm in an exercise in the book by [88]. It was more recently studied in detail and found in general to be an improvement over SARSA by [98]. It can be proved that in general Expected SARSA has the same bias but lower variance than the SARSA algorithm. Lower variance means that α could be increased to speed up the learning process. Expected SARSA can be also viewed as an on-policy version of Q-learning. In fact since $Q_t(s,a)$ is an estimate of $Q^\pi(s,a)$ its expectation can be seen as the estimate $V_t(s)$ for $V^\pi(s)$ through $V_t(s) = \sum_a \pi_t(s,a)Q_t(s,a)$. If the policy we have $\pi_t(s,a) = 0$ for all a except a^* for which $Q_t(s,a)$ has its maximal value. Hence, in case of greedy policy we have $V_t(s) = \max_a Q_t(s,a)$ showing that the Q-learning update is just a special case of Expected SARSA when the estimation policy is greedy. The proof of Expected SARSA convergence can be found in [97] according to the follwong theorem:

Theorem 1.1. *[Expected SARSA Convergence from [97]] Expected SARSA defined by the update section 1.7.2 converges to the optimal value function whenever the following assumptions hold:*

1. *S and A are finite*
2. $\alpha_t(s_t, a_t) \in [0,1], \quad \sum_t \alpha_t(s_t, a_t) = \infty, \quad \sum_t \alpha_t^2(s_t, a_t) < \infty \quad w.p.1$ *and* $\quad \forall (s,a) \neq (s_t, a_t) : \alpha_t(s,a) = 0$
3. *the policy is greedy in the limit with infinite exploration*
4. *the reward function is bounded*

Chapter 2

Generalization Problem in RL

2.1 Introduction

Goal of RL is to find the optimal policy. In the case of value function methods, this goal is reduced to find the optimal or near optimal value function. Under this methodology, when the state space is relatively small, the value function can be represented exhaustively by a lookup table with entries corresponding to each state But in practice, the state space can be very large, infinite or continuous. Tabular representation becomes impractical due to computational requirements. RL algorithms require exact representations of the value functions and policies. In general, an exact value function representation can be accomplished by storing different estimates of the reward for every state-action pair when Q-functions are used (or for every state in the case of V-functions). Different actions have to be stored for every state to represent policies exactly. In case the variables have a very large (or continuous) domains, one cannot make use of exact representations and value functions or policies need to be represented approximately. As most problems of possible interest have large or even continuous state and action spaces, approximation is necessary. Approximators can be separated into two main types: parametric and non-parametric. Parametric approximators can be seen as mappings from a parameter space into the space of functions they want to represent. Number of parameters and the form of the mapping are given a priori, while the parameters can be found using data about the target function. In contrast, the structure of a non-parametric approximator is derived from the data. Despite its name, a non-parametric approximator typically still has parameters. In this case the number of parameters and their values are determined from the data. Kernel-based approximators for example define one kernel for each point, and the target function may be represent as a weighted linear combination of such kernels, where the weights are the parameters The material for this section was referenced from [88],[10], [18], [103], [12], [48], [3].

2.2 Generalization in RL

In the formalization of the RL methods presented so far we assumed that the number of states and actions was finite, and that the value functions can be represented in a tabular way. Each state, or state-action, has associated a value, which has to be updated inde-

pendently of each other. This makes necessary that all the entries in the table have to be experienced many times in order to learn. Tabular representation are no longer applicable with RL methods applied to more complex control applications having large or infinite number of states and actions as it may require a number of experiences infeasible for the learning to take place. As a result of this limitation, more abstract state representation are necessary permitting to generalize the known value at one state, or state-action pair, to other similar states, or state-action pairs reducing the number of experiences required for learning.

Barto and Sutton [10] proposed one of the first methods for generalization in RL for the cart pole balancing control task. The need of generalization in RL was also commented in [86] and Watkins [103] where, among other things, was pointed out the necessity of generalization in RL. One important aspect comes with the convergence problems arising if the generalization of the experience in a state is transferred to other states that produces a value different from the one they experienced. Several authors proposed different techniques to generalize in RL in the past few years [32], [38], [62], [63], [68], [72], [73], [75]. Generally speaking they consist in combining the learning strategy in such a way values resulting from actions execution are able to generalize this knowledge using techniques coming from pattern recognition [15] or concept learning [60]. Methods performing local regressions like [4],[68] shows better behavior with the biased sampling and non-stationarity problems as they reduce the function capacity to adapt to smaller regions. This comes as a compromise for generalization performed through a trade-off between the necessary locality and the generalization issue. Another class of methods are fitted value iteration methods [38], [72],[73] which are batch and memory-based approaches performing a single step of the function approximation over a set of points. This gives samples representative for the learning and one step of value adaptation always on these points using a standard RL method. Fitting the value function and the value function adaptation on the set of points are carried out until some convergence criterion is fulfilled.

Typical problems in RL generalization may appear in different degrees at different regions of the domain, and at different stages of the learning process. Function approximator selected should allow to cope with the approximation requirement of the most demanding region, independently on the complexity of the approximation in other regions of the domain. During the learning process the estimation of the value function changes. This is a consequence of the policy evaluation process estimating the value function for the current policy. It is also a consequence of the policy improvement mechanisms changing the policy towards the optimal one. As a result the estimation of the value function becomes non-stationary. Moreover, as in RL data arrive one at a time sampled along trajectories which depends on the dynamics of the environment and are influenced by the action selection strategy. This is another difficulty for generalization methods which may produce biased sampling with large distortions in the estimations and instabilities in the convergence. Generalization should be robust to the non-stationary target function and biased sampling problem in order for learning to take place.

Shape and complexity of the value function uncertainties in the various regions of the domain during the learning process can make the selection of the correct function approximation quite complicated. In many cases selection has to done empirically through the evaluation of the performance of different approximators. Nevertheless, using a single function approximator one cannot avoid that the overall performance of the system to be

related to the approximator performance. A deficiency in the approximation in several part of the domain might turn into a failure in the whole learning process.

Various approximation architectures have been investigated for use in RL such as Neural Networks [41], Linear Architectures [48], [95], Wavelets [55], [54] and Splines [94]. Kernel-based methods like SVM ([81], [23]) or Gaussian Processes [72] for pattern classification and regression have been also used in RL domain. Candidates for value function approximation are SVR which is the regression method using SVM paradigm. A model-free approach for approximate value iteration is presented in [68] which uses kernel smoothers. Some authors [85], [84], [53] developed specialized kernels exploiting state space manifold structure while others [29] used Gaussian processes for computing the value function by an approximate value iteration algorithm. Tobias and Daniel [93] proposed a Least Squares Temporal Difference (LSTD) approach based on SVM. In [13] a kernel-based approximate DP using BRM is presented. Dietterich [30] investigated a linear programming approach using kernels and DP approximating the value function with SVR minimizing the TD error. However, their approach is unable to solve online learning problem because it estimates a value function after visiting all states using a uniform random behavior policy. In their approach, an RL agent can neither cumulate its experiences continuously nor adapt itself to the changing environment readily. On the contrary, an RL agent should generally be able to learn from data obtained sequentially from interaction with the environment. In all the reported works, while somehow the algorithms may be attractive for some specific problems or having interesting practical implementations, the lack of theoretical guarantee is a fundamental flaw.

2.2.1 Generalization And Discretization Issues

There are limitations and advantages of using function approximation in RL. Consider the tabular storage of each action value. Storing each value in a separate cell in a table has a clear advantage: since each cell is separate, there is no interference among them and the approximation of the value of each action for the cMDP rely on the algorithm used and the policy. Anyway, cell separation has some disadvantage. If the state space is large before each action is visited it can take a very long time. Furthermore, with noisy cMDP each action has to be visited several times to average out the noise. To avoid very slow convergence rates, it should be necessary to generalize over states and actions similar. Generally speaking a learned function can misinterpret observed data (poor generalization) when the function approximator is too inflexible. In this case the function approximator underfit the data. On the contrary with a too flexible function approximator it can represent nearly any function and many parameters might be tweaked with the risk that the function approximator will represent the general structure and the noise eventually present in the data. In this case then function approximator overfit the data. generalization error becomes large in both cases on a large portion of the value space. In RL a function approximator underfitting the data might learn quickly but is incapable of each a good performance or may not reach reasonable performance at all. A flexible function approximator might overfit the data which could be especially true when only a small amount of data has been observed. Usually this may result in slow learning since the estimations for states not observed will be quite poor. A suitably, trained and flexible function approximator might reach very good final performance levels as more experiences are obtained. Convergence

to good approximation in the limit can be guaranteed in some cases. Finally a more flexible approximator will give a better final approximation even though it might take very long before acceptable performance is obtained.

In practical applications of RL, states and actions are defined by continuous parameters and the sets S and A have to be considered large or infinite, while learning the value function requires function approximation. One often used method to find features for a linear function approximator is to simply divide the continuous space into separate segments and then to attach one or more features to each segment. These features are then active if the value of a point in the continuous space is considered that falls into the corresponding segment. Prominent examples of discretizing methods often used in RL are coarse coding and tile coding [88],[87]. In tile coding [88] space variable is partitioned into tiles and such partition is called a tiling. Tile code uses several overlapping tilings maintaining for each tiling the its weights. Approximate value for a given point can obtained considering the sum of the weights of the tiles (one per tiling). With a given a training sample method try to adjust the weights of the tiles trying to reduce the error on the samples set. Tile coding can be considered a piecewise constant approximator.

To explain the concept of coarse coding consider an RL problem with continuous and bi-dimensional state space. A possible feature can be chose as corresponding to circles or rectangles in the state space. In case the state is inside a circle the corresponding feature has the value 1 otherwise the feature is 0 (binary feature). In a given a state, binary features 1 indicate that the state lies within a given circle and thus coarsely code its location. The representation of a state with overlapping features is known as coarse coding. In coarse coding generalization from state to state depends on the number of their features whose receptive fields overlap. In case states have one feature in common, there will be generalization between them. If circles are small, generalization will be over a short distance. If they are large there will be over a large distance. Shape of the features determine the nature of the generalization and in linear function approximation methods this is determined by the sizes and shapes of the features receptive fields. Features with large receptive fields give broad generalization, but might also seem to limit the learned function to a coarse approximation, unable to make discriminations much finer than the width of the receptive fields. Initial generalization from one point to another may be controlled by the shape and size of the receptive fields.

2.3 MDPs in continuous spaces

Much of the discussion on finite MDPs still holds in the continuous case. The main difference is that the state space S will generally be an infinitely large bounded set. One may discuss continuous state and continuous state action MDPs. In the first case only the state space is continuous, but the action space is finite. In the second case both the state and the action space are continuous. Wherever we write continuous one can also assume that results essentially (not formally) hold for very large finite spaces. Techniques for large finite state spaces are very similar to the continuous one. For practical real-world applications it would nice to deal with continuous action MDPs. In this case one major difficulty is that extending convergence results to the continuous action requires maximizing the action at each state which except in special cases cannot be done exactly.

So hereafter we will work in continuous space and finite action domains. For a space Ω with a σ-algebra σ_Ω define $\mathcal{M}(\Omega)$ the set of all probability measure over σ_Ω and $\mathcal{B}(\Omega)$ the space of bounded measurable function w.r.t. σ_Ω and $\mathcal{B}(\Omega, L)$ the space of bounded measurable function with bound $0 < L < \infty$

Definition 2.1. *(**Continuous State cMDP**) A continuous state and finite action discounted **cMDP** can be defined as a tuple (S, A, P, γ), with the following definitions for its contents:*

- *S is a measurable state space where $s_t \in S$ denotes the state the agent is in at time t.*
- *A is a finite set of available actions where $a_t \in A$ denotes the action the agent performs at time t.*
- *$P : S \times A \to \mathcal{M}(\mathbb{R} \times S)$ is a mapping that when evaluated at $(s, a) \in S \times A$ gives the distribution over $\mathbb{R} \times S$ denoted as $P(r, s'|s, a)$.*
- *Marginals of P can be defined as $\mathcal{P}(\cdot|s, a) = \int_{\mathbb{R}} P(dr, \cdot|s, a)$ denotes the probability of ending up in state s' when performing action a in state s (transition probability)*
- *$\mathcal{R}(\cdot|s, a) = \int_S P(\cdot, ds'|s, a)$ is a reward function denoting the expected reward when the agent transitions from state s to state s' after performing action a.*
- *The immediate expected reward is defined as $r(s, a) = \int_{\mathbb{R}} r\mathcal{R}(dr|s, a) = \mathbb{E}[\hat{r}|s, a]$ while $\hat{r}$ his the empirical approximation of r.*
- *$\gamma \in [0, 1)$ is a discount factor.*

At stage t an action $a_t \in A$ is selected by the agent controlling the process and in response the pair (r_t, s'_t) is drawn from the distribution $P(r, s'|s_t, a_t)$ i.e $(r_t, s'_t) \sim P(r, s'|s_t, a_t)$ where r_t is the reward the agent receives and s'_t the next cMDP state. The procedure continue leading to a random trajectory $\xi_t = \{s_1, a_1, r_1, s_2, a_2, r_2, ...\} \in \Xi$ where Ξ denotes the space of all possible trajectories.

For a cMDP an agent consists mainly of an action selection policy such that $a_t = \pi(s_t)$. A stationary stochastic policy maps states to distributions over the action space $\pi_t : S \to \mathcal{M}(A)$ with $\pi_t(a|s)$ denoting the probability that the agent will select the action a to perform in state s at time t. We will use $\pi(s)$ to refer to the probability distribution or the probability mass function of the actions in state s. Stochastic policies are also called soft when they do not commit to a single action per state. $\pi(a|s)$ stands for the probability that the soft policy chooses action a in state s. An $\varepsilon-$greedy policy is a soft policy which for some $0 \leq \varepsilon \leq 1$ picks deterministically a greedy action with probability $1 - \varepsilon$ and a uniformly random action with probability ε. We will then use $a \sim \pi(\cdot|s)$ to indicate that action a is chosen according to the probability function in state s.

A continuous state and finite action cMDP encodes the temporal evolution of a discrete-time stochastic process controlled by the agent through the policy π. A probability model for the cMDP consists of a sample space Ω, a $\sigma-$algebra of Borel measurable subsets of Ω denoted as $\mathcal{B}(\Omega)$ and a probability measure on $\mathcal{B}(\Omega)$, As a results one choose $\Omega = S \times A \times S \times A.....S \times A \times S$ and a typical element $\omega \in \Omega$ consists in a sequence of

states and actions $\omega = \{s_0, a_0, s_1, a_1, ...\}$ which is called a sample path. The random variables (s_t, a_t) taking values in $(S \times A)$ as $(s_t(\omega), a_t(\omega))$. This means that when the observed sequence of states and action is ω the random variable (s_t, a_t) denotes the state-action at the time t.

A policy π inducing the transition probability kernel of following a policy π defined as $P^{\pi}(ds'|s) = \mathcal{P}(ds'|s, \pi(s))$. For a measurable subset $B \subseteq S \times A$ define

$$(P^{\pi})(B|s,a) = \int \mathcal{P}(ds'|s,a)\mathbb{I}_{\{(s',\pi(s'))\in B\}}$$

and the m-step transition probability kernels are defined inductively by

$$(P^{\pi})^m(B|s,a) = \int \mathcal{P}(ds'|s,a)(P^{\pi})^{m-1}(B|s',\pi(s'))$$

while for a probability measure $\rho \in \mathcal{M}(S \times A)$ define the left-linear operator

$$(\rho P)(B) = \int \rho(ds,da)P(ds',da'|s,a)\mathbb{I}_{\{(s',a')\in B\}}$$

.

Given the policy π the cMDP reduces to a Markov chain $\mathcal{M}^{\pi} = (S, R^{\pi}, P^{\pi})$ with reward function $R^{\pi} = r(s, \pi(s))$ and transition kernel $P^{\pi}(\cdot|s) = P(\cdot|, s, \pi(s))$ with a stationary distribution ρ if it admits one.

A stationary policy and an cMDP induce what is called a Markov Reward Process (MRP) which is determined by the pair $\mathcal{M} = (S, P)$, meaning that the transition probability kernel P assigns a probability measure over $S \times \mathbb{R}$ to each state. An MRP gives rise to the stochastic process $Z'_t = (s_t, r_{t+1})$ where $(r_{t+1}, s_{t+1}) \sim P(\cdot, \cdot|s_t, a_t)$. Note that $Z_t = (s_t, r_t)$ is a time-homogeneous Markov process with r_0 is an arbitrary random variable. Instead $Z'_t = (s_t, r_{t+1})$ is a second-order Markov process. Given a stationary policy π and the cMDP $\mathcal{M} = (S, A, P)$ the transition kernel of the MRP (S, P^{π}) induced by π and $\mathcal{M}$ is defined using $P^{\pi}(\cdot|s) = \sum_{a\in A} \pi(a|s)\mathcal{P}(\cdot|s,a)$. In a MRP given the history $\mathcal{H}_t = \{s_0, r_1, s_1, ..., s_t\}$ the distribution of the states s_{t+1} is completely determined by s_t while the distribution of r_{t+1} is completely determined by (s_t, s_{t+1}) given the history $\mathcal{H}_{t+1}$. Besides two different settings are possible considering π_b as a behavior policy. In the *off-policy* scenario the data comes in the form of triplets $\mathcal{H}_t = \{(s_0, \tilde{r}_1, \tilde{s}_1), (s_1, \tilde{r}_2, \tilde{s}_2)..., (s_{t-1}, \tilde{r}_t, \tilde{s}_t)\}$, where the distribution of (r_{t+1}, s_{t+1}) is independent of $\mathcal{H}_t$ given s_t and is equal to the transition kernel P. Further, it is assumed that $\{s_t\}_{t\geq 0}$ is a Markov process. A more general setting may be obtained when $\tilde{s}_t = s_t$ which is more general and is called *on-policy*.

Definition 2.2. ***(Value Function)*** *For an agent following the policy π considering the sequence of rewards $\{r_t : t \geq 1\}$ when the cMDP is started in the state action $(s_1, a_1) \sim \nu(s,a) \in \mathcal{M}(S \times A)$ the action value function Q^{π} is defined as*

$$Q^{\pi}(s,a) \quad = \quad \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_1 = s, a_1 = a, \pi] \tag{2.1}$$

It is worth noting that if $|r(s,a)| < R_{max}$ then $Q_{max} = R_{max}/(1-\gamma)$ for any π. The optimal value function is defined as

$$Q^*(s,a) = \sup_{\pi} Q^{\pi}(s,a) \quad \forall\, (s,a) \in S \times A \tag{2.2}$$

A policy $\pi = \hat{\pi}(\cdot, Q)$ is greedy w.r.t. an action value function Q if $\forall\, s \in S$ we choose $\pi(s) = \arg\max_{a \in A} Q(s,a)$

Definition 2.3. ***(Bellman Operator)*** *For a cMDP the Bellman operator $T^\pi : \mathcal{B}(S \times A) \to \mathcal{B}(S \times A)$ for action value function is defined as*

$$(T^\pi Q)(s,a) = r(s,a) + \gamma \int \mathcal{P}(ds'|s,a) Q(s', \pi(s')) \tag{2.3}$$
$$= \int P(dr, ds'|s,a)(r + \gamma \textstyle\sum_{a' \in A} \pi(a'|s') Q(s',a')) = \mathbb{E}[r + \gamma \textstyle\sum_{a' \in A} \pi(a'|s') Q(s',a')]$$

where $\mathcal{B}(\cdot)$ represents the space of bounded measurable functions. Given a policy π a fixed point for the Bellman operator is given by the action value function $Q^\pi = T^\pi Q^\pi$ while the Bellman optimality operator for action value function is defined as

$$(T^* Q)(s,a) = \int P(dr, ds'|s,a)(r + \gamma \max_{a' \in A} Q(s',a')) \tag{2.4}$$

In RL (differently than DP) cMDP model is not fully available or may be difficult to represent. Typically, the state space, the action space, and the discount factor are available, while the transition model and the reward function are not known in advance. Hence, RL algorithms rely on information coming from interaction between the agent and the process or from a generative model of the process and including observations of states, actions, and rewards. A *generative model* of an cMDP is a simulator of the process that takes a state s and an action a as inputs and generates a reward r and a next state s' sampled according to the dynamics of the cMDP. Observations are usually organized in tuples (s,a,r,s') meaning that at some time step the process was in state s, action a was taken by the agent, a reward r was received, and the resulting next state was s'. Samples (s,a,r,s') can be collected from actual episodes of interaction with the process and the learning agent does not have much control on the distribution of samples over the state space, since the process cannot be reinitialized. On the opposite, with samples coming from a generative model the agent has full control of the distribution over the state space and queries can be made for any arbitrary state. In both cases, the action choices of the learning agent are not restricted by the process, but only by the learning algorithm that is running. Samples may also come from stored experiences of other agents on the same cMDP.

Given a cMDP and a fixed policy π let D_n be a dataset $D_n = \{(s_1, a_1, r_1, s'_1, ...\}$ of observations and define the ordered multiset $Z_n = \{(s_1,a_1), ..., (s_n,a_n)\}$. The collected data D_n can be used to define the empirical operators and can be thought of as the empirical approximation to the true operators.

Definition 2.4. ***(Empirical Bellman operators)*** *Given the policy π consider the D_n and Z_n data sets the Empirical Bellman Operator $\hat{T}^\pi : Z_n \to \mathbb{R}^n$ is defined as*

$$(\hat{T}^\pi Q)(s_t, a_t) = r_t + \gamma \sum_{a' \in A} \pi(a'|s'_t) Q(s'_t, a') \quad t = 1, ..., n \tag{2.5}$$

while the Empirical Bellman Optimality Operator $\hat{T}^* : Z_n \to \mathbb{R}^n$ is defined as

$$(\hat{T}^* Q)(s_t, a_t) = r_t + \gamma \max_{a' \in A} Q(s'_t, a') \quad t = 1, ..., n \tag{2.6}$$

which provide an unbiased estimate of the Bellman operator where given the policy π *for any fixed bounded measurable deterministic function* $Q : S \times A \to \mathbb{R}$ *it holds that* $\mathbb{E}[\hat{T}^{\pi}Q(s_t,a_t)|s_t,a_t] = T^{\pi}Q(s_t,a_t)$ *and also* $\mathbb{E}[\hat{T}^{*}Q(s_t,a_t)|s_t,a_t] = T^{*}Q(s_t,a_t)$ *for* $1 \leq t \leq n$.

The empirical Bellman operators get an n-element list Z_n and return an n-dimensional real-valued vector of the single-sample estimate of the Bellman operators applied to the value function Q at the selected points. It is worth noting that in the expression 2.5 of the empirical Bellman operator we used the average operator over the policy $\sum_{a' \in A} \pi(a'|s'_t)Q(s'_t,a')$. Using this average helps exploiting the knowledge of the policy which may prevent large variance in the learning process. This can be actually thought as an extension of the Expected SARSA algorithm [99] used in tabular RL methods for the action value function approximation case. Applying the same criteria we may extend this to the empirical Bellman operator using an estimation of the transition matrix $P^{s'}_{s_t,a_t}$ through the collected data so that the *Average Empirical Bellman Operator* can be defined as

$$(\hat{T}^{\pi}Q)(s_t,a_t) = \sum_{s' \in S} P^{s'}_{s_t,a_t}\Big(r_t + \gamma \sum_{a' \in A} \pi(a'|s'_t)Q(s'_t,a')\Big) \quad t = 1,...,n \tag{2.7}$$

even though in continuous state space the estimation of $P^{s'}_{s_t,a_t}$ based on data is a quite difficult task. It is useful to recall here that given a continuous function space $f \in \mathcal{H}$ and a probability distribution $\nu \in \mathcal{M}(Z)$ the $L^p(\nu)-$norm $(1 \leq p,q < \infty)$ can be defined as

$$\|f\|^q_{p,\nu} = (\int |f(z)|^p d\nu(z))^{\frac{q}{p}}$$

The $L^p(\infty)-$norm is defined as

$$\|f\|^q_{p,\infty} = (\sup |f(z)|^p)^{\frac{q}{p}}$$

Considering the random $\mathcal{Z}$-valued sequence $Z_n = \{z_1,...,z_n\}$ with $z_t \sim \nu(z)$ the empirical $p(n)$-norm can be defined as

$$\|f\|^q_{p,n} = (\frac{1}{n}\sum_{t=1}^{n} |f(z_t)|^p)^{\frac{q}{p}}$$

having the property

$$\mathbb{E}[\|f\|^q_{p,n}] = \|f\|^q_{p,\nu}$$

Finally, sometimes we might use a shorthand for the empirical version of the expectation operator applied over a function $f(Z)$ defining the empirical operator

$$\mathbb{E}_n[f] = \frac{1}{n}\sum_{t=1}^{n} f(Z_t)$$

.

2.4 Parametric And Non-Parametric Function Approximation

Two major classes of function approximators can be identified, namely parametric and non-parametric approximators. Parametric approximators are mappings from a parameter space into the space of functions they aim to represent value functions or eventually policies. The functional form of the mapping and the number of parameters are typically established in advance and do not depend on the data. Parameters of the approximator are tuned using data about the target function. Consider a Q-function approximator parameterized by an r-dimensional vector $\mathbf{w}$. The approximator is denoted by an approximation mapping $F : \mathbb{R}^r \to \mathcal{D}$, where $\mathbb{R}^r$ is the parameter space and Q is the space of Q-functions. Every parameter vector $\mathbf{w}$ provides a compact representation of the corresponding approximate Q-function $\hat{Q}(s,a) = F(\mathbf{w},s,a)$ where F denotes the Q-function evaluated at the state-action pair (s,a). So, instead of storing distinct Q-values for every pair (s,a), which would be impractical in many cases, it is only necessary to store r parameters. When the state action space is discrete, r is usually much smaller than $|S| \cdot |A|$ providing a more compact representation. However, since the set of Q-functions representable by F is only a subset of Q, an arbitrary Q-function can generally only be represented up to a certain approximation error, which must be accounted for. In general, the mapping F can be non-linear in the parameters. However, linearly parameterized approximators are often preferred, because they make it easier to analyze the theoretical properties of the resulting algorithms. A linearly parameterized Q-function approximator employs r basis functions (BFs) $\phi_1, ..., \phi_r : S \times A \to \mathbb{R}$ and an (r+1)-dimensional parameter vector $\Theta = (\mathbf{w}, b)$ and the approximate Q-values are computed through:

$$\hat{Q}(s,a) = \sum_{j=1}^{r} \phi_j(s,a) w_j + b = \langle \Phi(s,a), \mathbf{w} \rangle + b$$

where $\Phi(s,a) = [\phi_1(s,a), ..., \phi_r(s,a)]^T$ is the vector of BFs. The notation $\langle \cdot, \cdot \rangle$ is used to denote the standard inner product. BFs function are also called features in the literature [12].

Non-parametric approximators, despite their name, still have parameters. However, unlike in the parametric case, the number of parameters, as well as the form of the non-parametric approximator, are derived from the available data. Kernel-based approximators are typical representatives of the non-parametric class. Consider a kernel-based approximator of the Q-function. In this case, the kernel function is a function defined over two state-action pairs, $\kappa : S \times A \times S \times A \to \mathbb{R}$ as $(s,a,s_j,a_]) \mapsto \kappa(s,a,s_j,a_j)$ that must also satisfy certain additional conditions [81]. Under these conditions, the function κ can be interpreted as an inner product between feature vectors of its two arguments in a high-dimensional feature space. Using this property, powerful approximators can be obtained by only computing the kernels, without ever working explicitly into the feature space. Assume that a set of state-action samples is available, for this set of samples, the kernel-based approximator takes the form:

$$\hat{Q}(s,a) = \sum_{j=1}^{n_s} \phi_j(s,a) w_j + b = \langle \Phi(s,a), \mathbf{w} \rangle + b$$

where $w_1, ..., w_r$ are the parameters. This form looks similar to the linearly parameterized approximator. However, there is a crucial difference between these two approximators. In the parametric case, the number and form of the BFs were defined in advance, and therefore led to a fixed functional form of the approximator. In contrast, in the non-parametric case, the number of kernels and their form, and thus also the number of parameters and the functional form of the approximator, are determined from the samples. One situation in which the kernel-based approximator can be seen as a parametric approximator is when the set of samples is selected in advance. Then, the resulting kernels can be identified with predefined BFs: $\phi_j(s,a) = \kappa(s,a,s_j,a_j),\ j = 1, ..., n_s$ and the kernel-based approximator is equivalent to a linearly parameterized approximator. However, in many cases, such as in online RL, the samples are not available in advance. .

Parametric approximators have to be flexible enough to accurately model the target functions solely by tuning the parameters because they are designed in advance. Highly flexible, non-linearly parameterized approximators such as neural networks are available. However, when used to approximate value functions, general non-linear approximators make it difficult to guarantee the convergence of the resulting algorithms, and indeed can sometimes even lead to divergence. Often, linearly parameterized approximators must be used to guarantee convergence. Such approximators are specified by their BFs. When prior knowledge is not available to guide the selection of BFs, a large number of BFs must be defined to evenly cover the state-action space. This is impractical in high dimensional problems.

Non-parametric approximators are highly flexible but as their shape depends on the data, it may change while the algorithm is running, making it difficult to provide convergence guarantees. Non-parametric approximators may adapts their complexity to the amount of available data. This is beneficial in situations where data is costly or difficult to obtain. It may become a disadvantage when a large amount of data is used due to the computational and memory demands of the approximator growing with the number of samples.

Either we are using a parametric or non-parametric approximator we may call the representation of the given function $Q(s,a)$ into the approximating architecture $\hat{Q}(s,a)$ as a *projection* of $Q(s,a)$ into the space of the possible functions might be represented with the given architecture $\hat{Q}(s,a)$. Projection is actually the method of finding the appropriate parameters maximizing the accuracy of the approximation according to certain criteria and with respect to the target function.

2.5 Linear And Non Linear Function Approximation

Apart from the parametric and non-parametric distinction, another distinction comes is we use linear or non-linear function approximation. Many tabular methods can be interpreted as optimizing a Bellman error through a gradient descent update. Here the idea is that if we can minimize the Bellman error $BE = \|Q - TQ\|$ we come closer to the desired fixed point $Q = TQ$. The simplest of non-tabular functions is a linear function as it can be interpreted as a linear function. The feature vector is of the same size as the state space and for each state precisely one element of the feature vector is equal to one, while the rest of the elements is equal to zero. Linear function approximation is easier to analyze

and implement, and there are global convergence guarantees that can not be given when non-linear function approximators are used.

When TD learning is used to estimate the value of a given stationary policy under on policy updates, the value function converges when the feature vectors are linearly independent [88], [32]. Main drawback of linear function approximation compared to the non-linear one is that none needs good informative features. Usually these may require some knowledge of the domain. Even if convergence in the limit to an optimal solution is guaranteed, the solution is only optimal in the sense that it is the best possible linear function of the given features. As a result poor features imply bad solutions. Additionally, while the theoretical guarantees are less convincing, nice empirical results have been obtained by combining RL algorithms with non-linear function approximators (such as neural networks).

Non-linear function approximation has the problem that it can get stuck in local optima and also different functions can be represented which means that the founded solution may be of better quality than the linear function approximators. In a non-linear function approximator, the value function is represented by some predetermined parametrized function, such that $\hat{Q}(s,a) = F(\mathbf{w},s,a)$. Here $\mathbf{w}$ is the parameter vector that is not necessarily of the same size as the feature vector. F could be a neural network and $\mathbf{w}$ will be a vector with all the weights of the network. We assume F fixed and that the value of the state is dependent on the state action through the feature vector $\Phi(s,a)$ and the time step through $\mathbf{w}$.

An example is represented by the use of gradient descent update [9], [18] which follows the direction of the negative gradient over some metric to be minimized. The idea is that the gradient of a parameterized function to its parameters points in the direction in which the function increases, according to a first-order Taylor expansion. Under the assumption that the function is smooth, changing the parameters an infinitesimally small amount in the direction of the negative gradient should then result in a new function that has an infinitesimally smaller value at the given input. The assumption behind the gradient descent algorithms is that this usually holds when a larger step in this direction is taken. Any update that follows the direction of the negative gradient is called a gradient descent update: Let $f : \mathbb{R}^r \times \mathbb{R}^N \to \mathbb{R}^M$ denote a parametrized function and define $f_t : \mathbb{R}^N \to \mathbb{R}^M$ to be the function that corresponds to a parameter vector $\mathbf{w} \in \mathbb{R}^r$ such that $\forall s \in \mathbb{R}^r : f_t(s) = f(\mathbf{w},s)$. In a gradient descent update, the parameters of f_t are updated so that the target of the update lies in the direction of the negative gradient. The update to the parameters then is

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} f_t(s) \tag{2.8}$$

where $\eta \in [0,1]$ is a step size and $s \in S$ is some input and $\nabla_{\mathbf{w}} f_t(s)$ is the gradient to the parameters. Then we can obtain a new function $f_{t+1} : S \to S$ which is defined by $f_{t+1}(s) = f(\mathbf{w}_{t+1},s)$. It can be seen from this definition that a gradient descent update is always defined for a given input, which is denoted by s in the definition. The update eq. (2.8) is defined on the whole parameter vector. Usually, the function that we want to minimize through a gradient descent procedure is some error measure. The goal is then to minimize this error as much as possible. If instead we want to maximize rather than minimize a function, the minus in update eq. (2.8) is replaced with a plus and the procedure is called gradient ascent. Many applications of gradient descent involve a fixed

set of inputs and target outputs to which the parameters of the function should be adapted. Although it can be slow to converge, one simple way to partially address the issue of local optima is to use stochastic iterative gradient descent. This algorithm uses only one data point at a time. This may then prevent getting stuck in a local optimum, since each update does not necessarily move the function into the direction of the gradient. It can not be guaranteed in general that the global optimum is found, but convergence to a local optimum of the error function can be guaranteed if the data points are chosen at random and the step size is chosen according to the Robbins-Monro conditions described in the previous chapter. Unfortunately, this subset may not span the full parameter space, which is why this is called a local optimum. An additional advantage of stochastic gradient descent over batch learning is that it is straightforward to extend online stochastic gradient descent to an adapting target function. In other words, one does not have to have an independent and identically distributed training set. These features make online gradient methods very suitable for RL.

2.6 Value Function Approximation in RL

The accuracy of an approximate solution $\hat{Q}^{\pi}$ generated by an RL algorithm is important to the control the performance achieved by the algorithm. For a continuous state cMDP a simple criterion for evaluating the accuracy of the solution is to evaluate the Bellman Error (BE) expressed in terms of $L_p(\nu)-$norm as

$$BE^q_{p,\nu} = \|Q - T^{\pi}Q\|^q_{p,\nu} = (\int |Q(s,a) - T^{\pi}Q(s,a)|^p d\nu(s,a))^{\frac{q}{p}} \tag{2.9}$$

where the individual terms $Q^{\pi}(s,a) - T^{\pi}\,Q^{\pi}(s,a)$ are called Bellman Residuals (BR). Designing a RL algorithm attempting to minimize the Bellman Error in section 2.6 for a cMDP seems to be a quite reasonable approach. In fact, finding the exact solution correspond to achieve a zero error. However, managing directly the minimization it is a difficult task as this requires the BR to be computed for every state in the state space. A common approach is to solve the problem only in a small set of representative sample states $\hat{S} \times \hat{A} \subseteq S \times A$ using some prior knowledge about important states in the system by means of simulations and using approximation of the Bellman Error over the sample states o $BE^q_{p,\nu} \approx \hat{BE}^q_{p,\nu}$. Such class of algorithms is known as Bellman Error Methods and a function approximation architecture must be used for the set of candidate functions. Choice of the architecture is a relevant aspect in the algorithm design. Parametric function representation may be a solution for this problem. However, as in most cases exact function representation is either unavailable value function approximation has to be used. Value function approximation is a difficult problem where experiences and theories have shown the difficulty to choose an efficient and stable function approximator with convergence guarantee in common RL algorithms.

Applications of RL in real-world problems have revealed many open research problems of value function approximation. From one side it is very difficult to choose the right parametric form for the value function approximator. In principle value function approximator should have a parametric form able to represent the optimal value function and all of the intermediate ones. In situation where the knowledge of the domain is not enough to

help locating the exact form of the value function, it is better to choose flexible value function approximator structures. However, researchers have found that most RL algorithms do not converge may not converge when combined with value function approximator. Another question is that not so much is known about convergence and error bounds of standard RL algorithms when used with value function approximators. In experimental applications divergence or oscillation can be observed in this kind of algorithm. Sometimes divergence results from the inability to remember of the function approximator. As an example in standard RL algorithm (like TD, Q-learning or SARSA) using neural networks during the initial training performance improve very nicely. However, having found a reasonable good policy the approximate version of the algorithm stop generating bad actions [41]. As a result, the neural network value function approximator behaves in such a way large numbers of states are predicted to have the same value. Hence, the algorithm start choosing random actions because all actions seems to be equally good. Bad experiences should lead these value function RL algorithm to improve approximation as the process repeats itself. This problem could be solved by keeping memory of bad and good problem so that the function approximator does not forget them.

Theoretical analysis given by [38], [39] provides insights into function approximators. They reports results about the stability of one class of function approximators, the so called averagers used with standard RL algorithms like TD. Systematic analysis of variants of different RL algorithms shows that the only convergence results are restricted to linear function approximators and piecewise constant approximators based on state aggregation. Hence, one may acknowledge that the convergence behavior of most combinations of RL algorithms and valuefunction approximators has to be considered an open and unknown issue.

2.7 Approximate Policy Iteration

Policy iteration (PI) [11] is a method of discovering the optimal policy for any given MDP, providing an iterative procedure in the policies space. PI discovers the optimal policy by generating a sequence of monotonically improving policies. PI scheme is based on the observation that T^{π} is a monotonic, a quasi linear and a contraction operator in the L_{∞} norm with contraction rate γ, and for any initial vector Q successive applications of T^{π} converge to the action value function Q^{π} of policy π. Each iteration consists of two phases: *policy evaluation* which computes the state-action value function Q_k of the current policy π_k by solving the linear system of the Bellman equations and *policy improvement* defining the improved greedy policy π_{k+1} over Q^{π_k} through

$$\pi_{k+1} = \arg\max_{a \in A} Q_k(s,a) \tag{2.10}$$

The action-value function Q_k is typically chosen to be such that $Q_k \approx T^{\pi_k} Q_k$, i.e., it is an approximate fixed point of T^{π_k}. Policy π_{k+1} is as good as π_k if not better. Policy evaluation and policy improvement steps are repeated until there is no change in the policy reaching a convergence on the final policy. fig. 2.1 shows a block diagram of PI scheme showing the actor critic architecture and the dependencies among the various components. Convergence guarantees to the optimal policy of PI scheme, strongly relies on the tabular

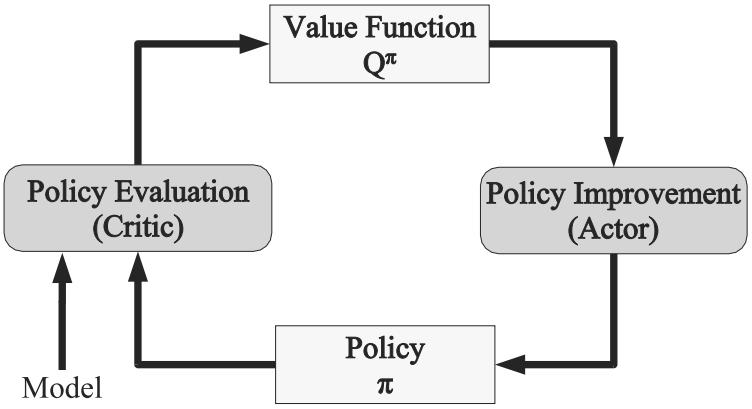

Figure 2.1: Policy Iteration scheme showing the actor critic architecture (from [48])

representation of the value function, exact solution of the Bellman equations, and tabular representation of each policy. In continuous space and finite action cMDP exact representations are infeasible and approximation methods must be used. In the framework of PI, approximations can be introduced into the value function or the policy representation. In this context API algorithms are also referred to as *actor-critic architectures* and represented as in fig. 2.2 borrowed from [48]. The actor *performs* actions by interacting with its environment while the critic *evaluates* the actions and gives feedback to the actor, leading improvement in the performance of next actions. Approximations in the PI framework may be introduced for the representation of the value function replacing its tabular representation by a parametric or non-parametric function approximator. Also the tabular representation of the policy $\pi(s)$ might be replaced by a parametric or non-parametric representation. In the API scheme value function projection and policy evaluation are essentially mixed into one procedure as there is no intermediate representation of the value function facilitating their separation. since there is no intermediate representation for a complete policy, the same also is true for policy improvement and policy projection. This clearly shows the difficulty involved in the use of function approximation within policy iteration algorithms. An important factor for a successful approximate algorithm is the choice of the approximation architecture. An obvious question is whether the sequence of policies and value functions generated by an API algorithm converges to a policy (if convergence is fulfilled) and value functions are close to the optimal ones. An answer is given by the following general theorem adapted from [11] showing that API can be considered a well defined algorithm. If the error in policy improvement, the projection and the error in policy evaluation are bounded, API generates policies having performance close to the optimal one. Further, this difference tends to zero as the errors come close to zero.

Theorem 2.1. *Let $\hat{\pi}_0, \ldots, \hat{\pi}_{K-1}$ be the sequence of policies generated by an API algorithm and let $\hat{Q}^{\hat{\pi}_0}, \ldots, \hat{Q}^{\hat{\pi}_{K-1}}$ be the corresponding approximate value functions. Let ε and δ be positive scalar numbers bounding the error in all the approximations over all iterations to value functions and policies respectively. If $\forall\, k = 0, 1, \ldots, \|\hat{Q}^{\hat{\pi}_k} - Q^{\hat{\pi}_k}\|_\infty \leq \varepsilon$, and*

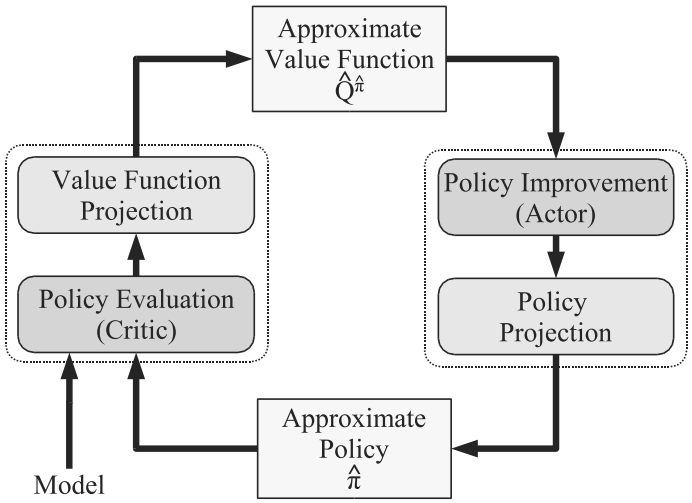

Figure 2.2: Approximate Policy Iteration scheme showing the actor critic architecture (from [48])

$\forall k = 0, 1, ..., \|T^{\hat{\pi}_{k+1}} \hat{Q}^{\hat{\pi}_k} - T^* \hat{Q}^{\hat{\pi}_k}\|_\infty \leq \delta$, *Then this sequence eventually produces policies whose performance is at most a constant multiple of* ε *and* δ *away from the optimal performance:* $\lim_{k\to\infty} \sup \|\hat{Q}^{\hat{\pi}_k} - Q^*\|_\infty \leq \frac{\delta+2\gamma\varepsilon}{(1-\gamma)^2}$

This generic theorem and experimental evidence provides that API is a quite interesting algorithm. In case of parametric approximation the crucial factor for a successful approximate algorithm is the choice of the parametric approximation architecture and the choice of the parameter adjustment method. For the non-parametric case the approximate is successful depending on the capacity of the kernel. This will be described in details in the next chapter.

2.8 Parametric Value Function Approximation

In the actor critic architecture for any state use the representation of the value function, the policy is not physically stored anywhere. Hence, we have to compute action values for the whole set of actions to evaluate the policy and to derive the greedy action performing the maximization at each state. As a consequence, for each query to the policy, one eliminates approximations and errors in representation and policy improvement paying the price of some extra optimization. Closing the loop in the procedure require the evaluation of a policy using data samples producing as a result the approximate value function. In Least Squares Policy Iteration (LSPI) algorithm of [48] this step is performed by Least Squares Temporal Difference Learning (LSTD) an algorithm efficiently learning the approximate action value function by using a linear architecture.

To derive LSPI consider the operator implementing the Bellman equations for action

values T^π with fixed point Q^π defined as

$$(T^\pi Q)(s,a) = \int P(dr,ds'|s,a)(r+\gamma \sum_{a' \in A} \pi(a'|s')Q(s',a')) \tag{2.11}$$

defining stochastic operators

$$(\Pi_a)\hat{Q} = \sum_a \pi(s,a)\hat{Q} \quad (P_s)\hat{Q} = \int P(dr,ds'|s,a)\hat{Q}$$

the resulting linear system

$$\Big(I - \gamma P_s \Pi_a\Big) Q^\pi = P_s R_{s,a} \tag{2.12}$$

which in principle can be solved analytically as

$$Q^\pi = \Big(I - \gamma P_s \Pi_a\Big)^{-1} P_s R_{s,a} \tag{2.13}$$

or iteratively to obtain exact Q^π values while the optimal Bellman operator is defined as

$$(T^* Q)(s,a) = \int P(dr,ds'|s,a)(r+\gamma \max_{a'} Q(s',a')) \tag{2.14}$$

In parametric value function approximation the parameters may be adjusted appropriately so that the approximate values are close enough to the original values, and, therefore, $\hat{Q}^\pi$ can be used in place of the exact value function Q^π. The characterization accepts a variety of interpretations in this context and it does not necessarily refer to a minimization of some norm. Value function approximation should be regarded as a functional approximation rather than as a pure numerical approximation, where functional refers to the ability of the approximation to play closely the functional role of the original function within a decision making algorithm. The difficulty associated with value function approximation, beyond the loss in accuracy, is the choice of the projection method. This is the method of finding appropriate parameters that maximize the accuracy of the approximation according to certain criteria and with respect to the target function. Typically, for ordinary function approximation, this can be done using a training set of examples of the form $\{(s,a),\, Q^\pi(s,a)\}$ providing the value $Q^\pi(s,a)$ of the target function at certain sample points (s,a) likewise in supervised learning. Unfortunately, in the context of decision making, the target function Q^π is not known in advance, and must be inferred from the observed system dynamics. The implication is that policy evaluation and projection to the approximation architecture must be mixed together. This is usually achieved by trying to find values for the free parameters so that the approximate function has certain properties that are similar to those of the original value function. Now consider a linear architectures approximating the value function Q^π using a linear parametric combination of r Basis Functions (BF) as

$$\hat{Q}^\pi(s,a) = \sum_{j=1}^{r} \phi_j(s,a) w_j + b = \langle \Phi(s,a), \mathbf{w} \rangle + b$$

where the $(\mathbf{w}, b)$ are the parameters. The BFs $\phi_j(s,a)$ are fixed but arbitrary and in general, non-linear functions of s and a. BSs ϕ_j are assumed to be linearly independent to ensure that there are no redundant parameters and that the matrices involved in the computations are full rank. In general $r \ll |S||A|$ and the basis functions ϕ_j have compact descriptions. As a result, the storage requirements of a linear architecture are much smaller than those of the tabular representation. Typical linear approximation architectures are polynomials of any degree (each basis function is a polynomial term) and radial basis functions (each basis function is a Gaussian with fixed mean and variance). Linear architectures have their own virtues: they are easy to implement and use, and their behavior is fairly transparent, both from an analysis standpoint and from a debugging and feature engineering standpoint. It is usually relatively easy to get some insight into the reasons for which a particular choice of features succeeds or fails. This is facilitated by the fact that the magnitude of each parameter is related to the importance of the corresponding feature in the approximation (assuming normalized features).

Let Q^π be the value function of a policy π given as a column vector of size $|S| \cdot |A|$ and $\hat{Q}^\pi$ the vector of the approximate action values as computed by a linear approximation architecture with parameters (w,b) and basis functions $\phi_j, j = 1,2,...,r$. Define $\Phi(s,a) = (\phi_1(s,a),...,\phi_r(s,a))^T$ to be the column vector of size r where each entry j is the corresponding basis function ϕ_j computed at (s,a). Now defining the $|S||A| \times (r+1)$ matrix $\Phi = [\Phi(s_1,a_1),...,\Phi(s_{|S|},a_{|A|})\ 1]^T$ and the vector $\mathbf{w} = [w\ b]^T$ then Q^π can be expressed in matrix form as $Q^\pi = \Phi\mathbf{w}$. Each row of Φ contains the value of all basis functions for a certain pair (s,a) and each column contains the value of a certain basis function for all pairs (s,a). If the basis functions are linearly independent, then the columns of Φ are linearly independent as well. We seek to find a combined policy evaluation and projection method that takes as input a policy π and the model of the process and outputs a set of parameters $(\mathbf{w}, b)$ such that is $\hat{Q}^\pi$ a good approximation to Q^π. Now recall that the action value function Q^π is the solution of the Bellman equation

$$Q^\pi = P_s R_{s,a} + \gamma P_s \Pi_a Q^\pi$$

An obvious approach in deriving a good approximation is to require that the approximate value function satisfies the Bellman equation as closely as possible. Substituting the approximation $\hat{Q}^\pi$ in place of Q^π yields an overconstrained linear system over the $r+1$ parameters (w,b)

$$\begin{aligned}\hat{Q}^\pi &\approx P_s R_{s,a} + \gamma P_s \Pi_a \hat{Q}^\pi \\ (\hat{Q}^\pi - \gamma P_s \Pi_a \hat{Q}^\pi) &\approx P_s R_{s,a} \\ (\Phi - \gamma P_s \Pi_a \Phi)\mathbf{w} &\approx P_s R_{s,a}\end{aligned}$$

Solving this system of overcontrained equations in the least-squares sense yields a solution [48]

$$\mathbf{w} = \Big((\Phi - \gamma P_s \Pi_a \Phi)^T (\Phi - \gamma P_s \Pi_a \Phi)\Big)^{-1} (\Phi - \gamma P_s \Pi_a \Phi)^T P_s R_{s,a}$$

minimizing the L_2 norm of the BR (the difference between the left-hand side and the right-hand side of the Bellman equation) and can be called the BR minimizing approximation

to the true value function. Note that the solution $\mathbf{w}$ of the system is unique since the columns of Φ (the basis functions) are linearly independent by definition.

Implementing API with parametric approximation can be also done using the BRM approach [90] computing the approximate state value functions from the model of the process. In fact, the action value function Q^π is also the fixed point of the Bellman operator $T^\pi Q^\pi = Q^\pi$ and another way to find a good approximation is to force the approximate value function to be a fixed point under the Bellman operator as $T^\pi \hat{Q}^\pi \approx \hat{Q}^\pi$ For that to be possible, the fixed point has to lie into the space of approximate value functions which is the space spanned by the basis functions. Even though $\hat{Q}^\pi$ lies in that space by definition, $T^\pi \hat{Q}^\pi$ may, in general, be out of that space and must be projected. Considering the orthogonal projection $(\Phi(\Phi)^T(\Phi)^{-1}(\Phi)^T)$ which minimizes the L_2 norm, we seek an approximate value function $\hat{Q}^\pi$ that is invariant under one application of the Bellman operator T^π followed by orthogonal projection

$$\hat{Q}^\pi \approx (\Phi(\Phi^T\Phi)^{-1}\Phi^T)(T^\pi \hat{Q}^\pi)$$
$$\hat{Q}^\pi \approx (\Phi(\Phi^T\Phi)^{-1}\Phi^T)(P_s R_{s,a} + \gamma P_s \Pi_a \hat{Q}^\pi)$$

Note that the orthogonal projection to the column space of Φ is well-defined because the columns of Φ (the basis functions) are linearly independent by definition. Manipulating the equation above, one may derive an expression for the desired solution that amounts to solve a $(r+1 \times r+1)$ linear system, where r is the number of basis functions [48]

$$\mathbf{w} = \Big(\Phi^T(\Phi - \gamma P_s \Pi_a \Phi)\Big)^{-1} \Phi^T P_s R_{s,a}$$

is guaranteed to exist for all values of γ ([47],[44]). Since the orthogonal projection minimizes the L_2 norm, the solution $\mathbf{w}$ yields a value function $\hat{Q}^\pi$ which can be called the least-squares fixed-point approximation to the true value function. Clearly, the solutions found by these two methods will be different since their objectives are different, except in the case where the true value function Q^π lies in the Φ plane; in that case, both methods are in fact solving the Bellman equation and their solutions are identical. If Q^π does not lie in the Φ plane, there is no clear evidence that any of the two methods will find a good solution or even the solution (w,b) that corresponds to the orthogonal projection of Q^π to the plane. Munos [66] provides a theoretical comparison of the two approximation methods in the context of state value-function approximation when the model of the process is known concluding that the least-squares fixed point approximation is less stable and less predictable compared to the BRM approximation depending on the value of the discount factor.

2.9 Non-parametric Value Function Approximation Using Kernels

In [30] Diettrich makes use of two different techniques using Linear Programming (LP) ([106], [2]) with kernel to find the value function approximation for the RL problems. The first formulation is based on Bellman Error and SVR while the other one uses the advantage with respect to the best action trying to ensure that good action should have

an advantage over bad ones. Both formulations try to minimize the support vectors number fitting the data of the value function. Unlike other approximation architecture, kernel methods can be adapted to the final complexity of the approximator fitting the value function. Virtually all existing work on value function approximation and policy gradient methods starts with a parameterized formula for the value function or policy and then seeks to find the best policy that can be represented in that parameterized form. This can give rise to very difficult search problems for which the Bellman equation is of little or no use. However, rather than fixing the form of the function approximator and searching for a representable policy, one may instead identify a good policy and then search for a function approximator that can represent it. This approach exploits the ability of mathematical programming to represent a variety of constraints including those that derive from supervised learning, from advantage learning and from the Bellman equation. By combining the kernel trick with mathematical programming, one may obtain a function approximator that seeks to find the smallest number of support vectors sufficient to represent the desired policy. This sidesteps the difficult problem of searching for a good policy among those policies representable by a fixed function approximator. A way to do that is to consider a continuous state cMDP and assume that estimation of the average transition probability and rewards are known and available to the agent. Let $\tilde{S}$ be a set of training states for which we have an approximation $\tilde{V}(s)$ to the optimal value function $V^*(s)$, $s \in \tilde{S}$. Moreover, also assume the availability of a policy π consistent with $\tilde{V}(s)$. The goal is to construct a parameterized approximation $\hat{V}(s)$ that can be applied to all states in the cMDP to yield a good policy π via one step look-ahead search. In the experiments reported in [30], the set $\tilde{S}$ contains states that lie along trajectories from a small set of training starting states $\tilde{S_0}$ to terminal states. A successful learning method should be able to generalize to give a good policy for new starting states not in $\tilde{S_0}$. To represent states for function approximation, let $\Phi(s)$ denote a vector of features describing the state s. In the *Linear Programming* ([27], [100]) formulation of the function approximation problem is introduced, expressing each of these formulations in terms of a generic fitted function approximator $\hat{V}(s)$ and then implement $\hat{V}(s)$ as the dot product of a weight vector $\mathbf{w}$ with the features vector $\Phi(s)$ such that $\hat{V}(s) = \langle \Phi(s), \mathbf{w} \rangle$ using a given kernel and considering the expansion of $\mathbf{w}$ as a weighted sum of the training points as $\mathbf{w} = \sum_{s \in \tilde{S}} \alpha_s \Phi(s)$ In all linear programming formulations linear objective functions are used and all slack variables are constrained to be non negative. The first formulation treats the value function approximation problem as a *supervised learning* problem and applies the $\varepsilon - insensitive$ loss function to fit the function approximator. The objective function seeks to minimize these absolute deviation errors. A key idea of support vector methods is to combine this objective function with a penalty on the norm of the weight vector. The $1 - norm$ of the weight vector is minimized because it is possible to implement it via linear programming which brings to the following optimization problem

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \|\mathbf{w}\|_1 + C \sum_{s \in \tilde{S}} (\xi_s + \xi_s^*) \\ s.t. \quad & \langle \Phi(s), \mathbf{w} \rangle + \xi_s \geq \tilde{V}(s) - \varepsilon \\ & \langle \Phi(s), \mathbf{w} \rangle - \xi_s^* \leq \tilde{V}(s) + \varepsilon \\ & \xi_s, \xi_s^* \geq 0 \quad \forall s \in \tilde{S} \end{aligned} \tag{2.15}$$

In the *advantage learning formulation* linear programming is also used and the focus is on the minimal constraints that must be satisfied to ensure that the greedy policy computed from $\tilde{V}$ will be identical to the greedy policy computed from $\hat{V}$ requiring that the backed up value of the optimal action a^* be greater than the backed up values of all other actions a in a given state s. In this way there is one constraint and one slack variable $\xi_{s,a^*,a}$ for every action executable in state s except for the chosen optimal action $a^*(s) = \pi(s)$. The backed up value of a^* must have an advantage of at least ε over any other action a considering s' or s'' successors state of s applying respectively the action a or a^* we have

$$\begin{array}{ll} \min\limits_{\mathbf{w},\xi} & \|\mathbf{w}\|_1 + C\sum_{s\in\tilde{S},a^*,a\in\tilde{A}(s)} \xi_{s,a^*,a} \\ s.t. & \sum_{s''\in\tilde{S}} P_{sa^*}^{s''}(R_{sa^*}^{s''} + \langle\Phi(s''),\mathbf{w}\rangle) + \xi_{s,a^*,a} \geq \\ & \sum_{s'\in\tilde{S}} P_{sa}^{s'}(R_{sa}^{s'} + \langle\Phi(s'),\mathbf{w}\rangle) + \varepsilon \\ & \xi_{s,a^*,a} \geq 0 \quad \forall s,s',s''\in\tilde{S} \quad a^*,a\in\tilde{A}(s) \end{array} \tag{2.16}$$

Experimental evidence applying both linear programming methods on a maze problem shows that these methods are able to learn and generalize well with advantage learning formulation easier and more reliable to train.

Advantage updating was also explored in several interesting directions in [7] where they replaces the single value function approximator $Q(s,a)$, with two approximators. One for values and one for the advantages. Suppose the agent is in state s and is considering possible actions a^* and a with backed-up values $Q(s,a^*)$ and $Q(s,a)$. Then the quantity

$$A(s,a^*,a) = Q(s,a^*) - Q(s,a)$$

is called the advantage of action a^* over action a in state s . In [7] Baird uses only the advantages with respect to the best actions a^* so $A(s,a) = A(s,a^*,a)$ and the advantage of the best action a^* is 0, while the advantages of all of the other actions are positive. The value of the state $V(s)$ is stored separately. So the predicted value of executing action a in state s is the sum $V(s) + A(s,a)$. The motivation for this approach is the following. Suppose that the current estimated value for state s is very wrong. It is still possible that the values of the best actions relative to alternative actions are correct, and thus the agent is still going to make a correct choice of action according to V. The separation of V and A allows the advantage learning algorithm to update V without disturbing the relative values of the actions. In contrast, other algorithms, such as Q-learning, cannot do this without temporarily damaging the relative values of the actions.

Chapter 3

Kernel Based Approximate Policy Iteration

3.1 Introduction

Starting from this chapter we consider the problem of finding an optimal policy in continuous space finite actions cMDP given the trajectory of some behavior policy. The idea is to present here and in the next two chapters an API algorithm where in successive iterations the action value functions of the intermediate policies are obtained approximating the value functions using SVR and then the optimal policy. The algorithm repeatedly computes an evaluation function of the policy of the previous step and then uses this evaluation function to compute the next improved policy. To avoid the need of learning a model, action value functions are computed, making the policy improvement step simple, similarly as in the parametric approximation of LSPI algorithm of [48]. LSPI rely on LSTD while our algorithm makes use of a regularized version of the BRM described in section 2.8. The idea of using BR in PI goes back to [8] who proposed it for computing approximate state value functions given the model of a finite-state and action cMDP. Seeking small Bellman Error may yield to a good approximation of the policy evaluation function, which in turn may imply a good final performance. To introduce our regularized API-BRM method using SVR we still have to introduce several elements useful both for the theoretical analysis and the practical implementation of the algorithm. Henceforth, we start with a discussion about data collection and processing. Hence, we introduce some important elements about RKHS and error evaluation in a regularized approximation framework. The main result of this work presented in chapter 4, is a finite-sample bound on the performance of the resulting policy depending on the mixing rate of the trajectory, in the approximation power of the function set, in the capacity of the function set and finally on the discounted concentrability of the future state distribution. One major technical difficulty of the proof is that one has to deal with dependent samples. The main condition we ask is that the trajectory should be sufficiently representative and rapidly mixing. We also require that the states in the trajectory follow a stationary distribution. The mixing condition is essential for efficient learning and in particular we use the exponential β-mixing condition. The particular mixing condition assumed allows us to derive polynomial decay rates for the estimation error as a function of the sample size. An im-

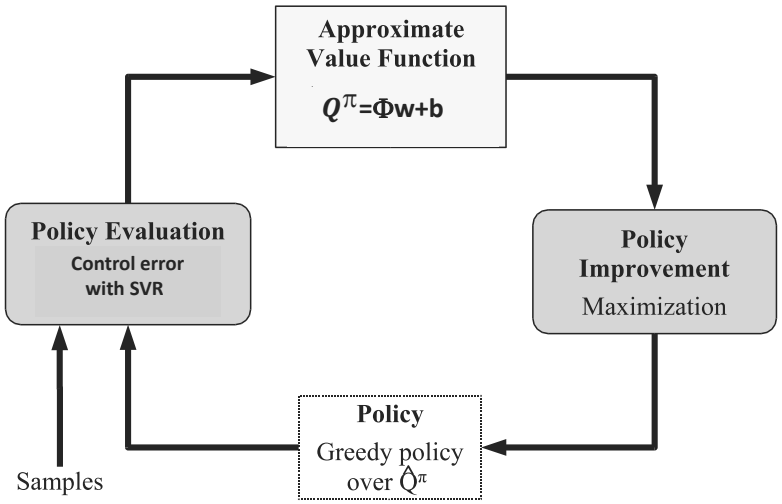

Figure 3.1: Approximate policy iteration using SVM

portant contribution of the present work relates to the finite-sample analysis for the batch algorithm in the β-mixing case which can be used in practical online learning. Furthermore our algorithm is based on exact incremental SVR in the context of BRM with API. Some technical lemmas are also proved in order to study the statistical property of our method requiring the extension to ε-insensitive losses while in standard regression analysis quadratic losses are normally used. A heuristic and practical implementation can be derived from our algorithm where policy improvements can be performed on the basis of incompletely evaluated value functions and is presented in chapter 5.

3.2 Finite Data Sampling

An important aspect of any method solving RL problems is the way that data are collected and processed. *Data collection* setting can be categorized as *online* or *offline* and the *data processing* method can be categorized as *batch* or *incremental*. The online sampling setting is when the agent chooses the action sequence $a_t \sim \pi_t(\cdot|s_t)$ and directly influences how the data stream is generated $D_n = \{(s_1, a_1, s'_1, r_1), ..., (s_n, a_n, s'_n, r_n)\}$ which we may assume as a stationary process that in general could be non-i.i.d. The stochastic functions $\pi_t(\cdot|s)$, whenever evaluated for the states s' in D_n define the stochastic process $\Pi_n = \{\pi_1(\cdot|s'_1), ..., \pi_n(\cdot|s'_n)\}$. Assuming controlled mixing condition for the non-i.i.d. process, allows for the generalization of strictly i.i.d. scenarios. Accordingly we may define the ordered multisets $\{(s_1, a_1), ..., (s_n, a_n)\}$ and $\{(r_1, s'_1), ..., (r_1, s'_1)\}$ with $a_t = \pi_{b_t}(s_t)$ and $s_t \sim \nu_s(s)$ and $\nu_s(s) \in \mathcal{M}(S)$ while $(r_t, s'_t) \sim P(r, s'|s_t, a_t)$. Here π_b is a behavior stationary Markov policy producing $(s_t, a_t) \sim \nu(s, a)$ with $\nu(s, a) \in \mathcal{M}(S \times A)$ the resulting state action distribution. In the online setting the behavior policy should be the same as the learning policy (fully optimistic) or can be updated (improved) once every several transitions (partially optimistic). On the contrary in the offline setting the agent does not have control on how the data are generated and the agent is provided with a data set D_n which

can be assumed in general non-i.i.d. according to some unknown distribution. In the offline setting the behavior policy is usually stochastic and might be unknown to the agent. Generally speaking different heuristics can be used to define the behavior policy π_b for example selecting a fixed stochastic stationary policy π_b generates i.i.d. samples; or using a policy based on the most recent estimate of the action-value function $\hat{Q}$ which can be the $\varepsilon-$greedy policy w.r.t. $\hat{Q}$. Data processing while learning may use a batch or an incremental algorithm. A batch algorithm processes the whole data set D_n and can freely access any element at any time. An incremental algorithm continues to learn whenever a new data sample is available while the computation in principle might not directly depend on the whole data set D_n but could rely on the last sample. Boundary between incremental and batch algorithm may be not so clear and, as a matter of fact, an incremental algorithm might be considered as a special case of a batch one whenever data are processed in a specific temporal ordering. In batch RL problems, the learner cannot make any assumptions on the sampling procedure of the transitions. Sampling could be done using a random policy, uniform in the state action space or along non connected trajectories. Using only this information, the learner has to come up with a policy that will then be used by the agent to interact with the environment. During this application phase the policy is fixed and no further improved as new observations come in. Since the learner itself is not allowed to interact with the environment, and the given set of transitions is usually finite, the learner cannot be expected to always come up with an optimal policy. There is a distinct separation of the whole procedure into three aspects: exploring the environment and collecting state transitions and rewards, learning a policy, and finally apply the learned policy. Exploration has a relevant impact on the quality of the learned policies. Transition probabilities of the system should be present into the distribution of transitions of the provided batch to allow the derivation of good policies. A simple way to achieve this result consists in collecting the training examples from the system itself, by simply interacting with it. In any case when sampling comes from the real system the covering of the state space by the transitions used for learning becomes an important issue. When important regions (for example states close to the goal) are not represented by any samples, learn a good policy from the data it is not possible, since important information is missing. In practice this might be a real problem because a purely random policy is often not able to achieve an adequate covering of the state space. This could be especially true in the case of attractive starting states and hard to reach desirable states. It is often necessary to already have a rough idea of a good policy in order to be able to explore interesting regions that are not in the direct vicinity of the starting states. Alternating exploration phases (where a set of training samples is grown by interacting with the system) and exploitation phases (where the batch set of observations is used) and called *growing batch learning problem*. Alternations number between episodes of learning and exploration can be in the whole range of being as close to the pure batch approach to recalculating the policy after every few interactions. Such growing batch approach represents in practice the modeling of choice when applying batch RL algorithms to real systems. Since from the interaction perspective the growing batch approach is very similar to the pure online approach, the distinction between online and offline, might be not that useful anymore.

3.3 Reproducing Kernel Hilbert Spaces

A kernel function $\kappa(\cdot,\cdot)$ plays an important role in any kernel-based learning method mapping any two elements from a space of input patterns to the real numbers taken the state space of the cMDP, and can be thought of as a similarity measure on the input space. In the derivation of kernel methods, the kernel function arises naturally as an inner product in a high-dimensional features space. Hence the kernel satisfy several important properties of an inner product: it must be symmetric and positive semidefinite, meaning that the associated Gram matrix $K_{ij} = \kappa(s_i, a_i, s_j, a_j)$ must be positive semidefinite. A kernel that satisfies these properties is said to be admissible. The key idea of the kernel technique is to invert the chain of arguments, choosing a kernel rather than a mapping before applying a learning algorithm. Of course, not any symmetric function κ can serve as a kernel. One fundamental property comes with Mercer's kernels (adapted from [82])

Proposition 3.1. *(Mercer's Kernel)*
The function $\kappa : (S \times A) \times (S \times A) \to \mathbb{R}$ is a Mercer kernel if and only if for each $n \in \mathbb{N}$ and $Z_n = \{(s_1,a_1),...,(s_n,a_n)\}$, the $n \times n$ matrix $\mathbf{K}_{ij} = k((s_i,a_i),(s_j,a_j))$ is positive semidefinite.

Given a kernel function his smallest feature space which can serve as a canonical feature space is the RKHS defined as (adapted from [25]):

Definition 3.1. ***(Reproducing Kernel Hilbert Spaces)*** *Consider a subset of measurable functions $\mathcal{F} : S \to \mathbb{R}$ and a subset of vector values measurable functions $\mathcal{F}^{|A|} : S \times A \to \mathbb{R}^{|A|}$ such that*

$$\mathcal{F}^{|A|} = \{(Q_1,...,Q_{|A|}) : Q_i \in \mathcal{F}, i = 1,...,|A|\} \tag{3.1}$$

called the hypothesis space $\mathcal{H}$. A natural choice for $\mathcal{H}$ is within the framework of RKHS $\mathcal{H} : S \times A \to \mathbb{R}$ which is an Hilbert space defined in $S \times A$ with the inner product $\langle \cdot,\cdot \rangle$ and characterized by a symmetric positive definite function $\kappa : (S \times A) \times (S \times A) \to \mathbb{R}$ called reproducing kernel, continuous in $S \times A$ such that for each $(s,a) \in S \times A$ the following reproducing property holds:

$$\forall\,(s,a) \in S \times A \quad Q(s,a) = \langle Q(\cdot), \kappa(\cdot,s,a) \rangle_{\mathcal{H}} \quad \kappa(\cdot,s,a) \in \mathcal{H} \tag{3.2}$$

assuming as measurable function space $\mathcal{F}^{|A|} = \mathcal{H}$. $\mathcal{H}$ is the closure of the linear span of the set of functions $\Phi_{span} = \{\, \Phi(s,a) = \kappa(\cdot,s,a) \mid (s,a) \in S \times A \}$ considering the map from $\Phi : S \times A \to C^0 S \times A$ which denotes the function that assigns the value $\kappa(s_t,a_t,s,a)$ to $(s_t,a_t) \in S \times A$ and $C^0(S \times A)$ the space of continuous functions on $S \times A$.

RKHS spaces have the important property that norm convergence implies point-wise convergence. The full power of RKHS can be expressed by the following Theorem (adapted from [78]):

Theorem 3.1. *(Representer Theorem) Let κ be a Mercer Kernel and D_n be a training set while $R_{emp} : Z \times \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be any arbitrary function. Now let be $R_{reg} : \mathbb{R} \to [0,\infty)$ be a strictly monotonically increasing function. Define $\mathcal{H}_K$ as RKHS induced by κ. Then any $Q \in \mathcal{H}_K$ minimizing the regularized risk $R_{reg}(Q) = R_{emp}(Q) + \lambda \|Q\|^2_{\mathcal{H}}$ admits a representation of the form $Q(\cdot) = \sum_{t=1}^{n} \beta_t \kappa(\cdot,s_t,a_t)$*

Let $\bar{\kappa} = \sup_{(s,a)\in S\times A}\sqrt{\kappa(s,a,s,a)}$ then the reproducing property tells us that $\|Q\|_\infty \leq \bar{\kappa}\|Q\|_{\mathcal{H}} \quad \forall Q \in \mathcal{H}$.

Using the Representer Theorem the function in $\mathcal{H}_K$ can be expressed as linear combination of the elements in the span $\Phi_{span} = \{\Phi(s,a) = \kappa(\cdot,s,a) \mid (s,a) \in S\times A\}$ which can be expressed as $Q(s,a) = \sum_t \beta_t \kappa(s,a,s_t,a_t)$. Hereafter we introduce the regularized regression using RKHS as functions spaces and quadratic norm into the Hilbert space $\|Q\|^2_{\mathcal{H}}$ as regularizer term. In fact, as we want that the regularizer measures the complexity of the action value function $Q(s,a)$, more complex functions should have larger regularizer which means larger norm. Moreover, we have to point out that dealing with finite action space one should define the norm of $Q(\cdot,a) \quad a \in A$ in a proper way. As a mild condition the complexity of Q should upper bound the complexity of $Q(\cdot,a)$ for all $a \in A$ and in RHKS this can be achieved defining $\|Q(s,a)\|^2_{\mathcal{H}} = \sum_{a\in A}\|Q(\cdot,a)\|^2_{\mathcal{H}}$.

A useful concept related to the complexity of the function space is represented by the covering numbers (adapted from [40]):

Definition 3.2. *(**Covering Number**) Let $\varepsilon > 0$ and $\mathcal{H}$ be a set of real-valued functions defined on X and ν_x a probability measure on X. Every finite collection of $N_\varepsilon = \{f_1,...,f_{N_\varepsilon}\}$ defined on X with the property that for every $f \in \mathcal{H}$ there is a function $f' \in N_\varepsilon$ such that $\|f - f'\|^q_{p,\nu_x} \leq \varepsilon$ is called $\varepsilon-$cover of $\mathcal{H}$ w.r.t. $\|\cdot\|^q_{p,\nu_x}$. Let $\mathcal{N}_p(\varepsilon,\mathcal{H},\|\cdot\|^q_{p,\nu_x})$ be the size of the smallest $\varepsilon-$cover of $\mathcal{H}$ w.r.t. $\|\cdot\|^q_{p,\nu_x}$. If no finite $\varepsilon-$cover exists $\mathcal{N}(\varepsilon,\mathcal{H},\|\cdot\|^q_{p,\nu_x}) = \infty$. Then $\mathcal{N}_p(\varepsilon,\mathcal{H},\|\cdot\|^q_{p,\nu_x})$ is called an $\varepsilon-$covering number of $\mathcal{F}$ and $\log\mathcal{N}_p(\varepsilon,\mathcal{H},\|\cdot\|^q_{p,\nu_x})$ is called the metric entropy of $\mathcal{H}$. Considering the empirical norm based $\|\cdot\|^q_{p,n}$ on the sequence of random variable $X_n = \{X_1,...,X_n\}$ we may define the empirical covering number as $\mathcal{N}_p(\varepsilon,\mathcal{H},\|\cdot\|^q_{p,n})$*

Given an admissible kernel κ and the RKHS $\mathcal{H}$ assume that for each $R > 0$ let be $\mathcal{H}_R = \{f \in \mathcal{H} : \|f\|_{\mathcal{H}} \leq R\}$ then exists a constant $C > 0$ and $0 < \alpha < 1$ such that for any (R,ε) the following metric entropy condition is satisfied (from [82])

$$\log\mathcal{N}_p(\varepsilon,\mathcal{H}_R,\|f\|^q_{p,n}) \leq C\left(\frac{R}{\varepsilon}\right)^{2\alpha} \tag{3.3}$$

Since some of the presented results involve expectations of suprema over uncountable sets it is useful to introduce the following notion [82]:

Definition 3.3. *(**Charatheodory Set**) Let (T,d) be a metric space and (Z,σ_Z) a measurable space. A family of measurable maps $\{f_t\}_t \in T$ is called a Caratheodory family if $t \mapsto f_t(z)$ is continuous for all $z \in Z$. Moreover, if T is separable or complete, we say that $\{f_t\}_t \in T$ is separable or complete, respectively. A measurable set $\mathcal{F} \subset Z$ (separable or complete) is a Caratheodory set if there exists a (separable or complete) metric space (T,d) and a Caratheodory family $\{f_t\}_t \in T$ such that $\mathcal{F} = \{f_t : t \in T\}$. Note that, by the continuity of $t \mapsto f_t(z)$, Caratheodory sets satisfy*

$$\sup_{f\in\mathcal{F}} f(z) = \sup_{t\in T} f_t(z) = \sup_{t\in Z} f_t(z), \quad z \in Z \tag{3.4}$$

for all dense $Z \subset T$. For separable Caratheodory sets $\mathcal{F}$, there exists a countable and dense $Z \subset T$, and hence the map $z \mapsto \sup_{t\in T} f_t(z)$ is measurable for such $\mathcal{F}$. Also for a complete Charatheodory set $\mathcal{F}$ the map $(z,t) \mapsto f_t(z)$ is measurable.

3.4 Regularized Non-Parametric Regression

In this section we introduce some insights about regularized regression in RKHS. Let $X \subset \mathbb{R}^d$ be a measurable space and $Y \subseteq \mathbb{R}$ a Polish space (separable completely metrizable topological space). In non-parametric regression the goal is to estimate a functional relationship between an input random variable X and an output random variable Y under the assumption that the joint distribution $P(X,Y)$ is (almost) completely unknown. To solve this problem, one typically assumes a set of observations (X_i,Y_i) from i.i.d. random variables (x_i,y_i), $i = 1,...,n$, all having distribution P with the corresponding Borel $\sigma-$algebra and consider the finite sequence of gathered samples $Z_n = \{(X_1,Y_1),...,(X_n,Y_n)\}$. Eventually, within a stationary process we may also try to cope with mixing scenarios.

The learning goal aims to build a predictor $f : X \to \mathbb{R}$ on the basis of these observations such that $f(x)$ is a good approximation of y. To formalize this aim, one assumes a loss function ℓ, that is, a continuous function $\ell : \mathbb{R} \times \mathbb{R} \to [0,\infty)$ assessing the quality of a prediction $f(x)$ for an observed output y by $\ell(y,f(x))$. Here it is commonly assumed that the smaller $\ell(y,f(x))$ is, the better the prediction is. The quality of a predictor f is then measured by the risk

$$R_{\ell,P}(f) = \mathbb{E}[\ell(y,f(x))] = \int_{X,Y} \ell(y,f(x)) d\nu_x(x) dP(y|x)$$

which is a random variable defined by the average loss obtained by predicting with f. Following the interpretation that a small loss is desired, one tries to find a predictor with risk close to the optimal risk

$$R^*_{\ell,P} = \inf\{ R_{\ell,P}(f) \,|\, f : X \to \mathbb{R}\}.$$

Moreover one may also define the inner risk as

$$\mathcal{C}_{\ell,P}(f(x)) = \int_Y \ell(y,f(x)) dP(y|x).$$

We are interested in the cost function having the property:

$$R_{\ell,P}(f) - R^*_{\ell,P} = \int_X (\mathcal{C}_{\ell,P}(f(x)) - \mathcal{C}^*_{\ell,P}) d\nu_x(x)$$

Consequently one can analyze $R_{\ell,P}(f) - R^*_{\ell,P}$ looking at the inner risk excess $\mathcal{C}_{\ell,P}(f(x)) - \mathcal{C}^*_{\ell,P}$ investigating its measure with respect to ν_x.

In regression problems, the loss function is typically distance based and putting $t = f(x)$ it only relies on the variable $\eta = y - t$ and the mapping $\eta \mapsto \ell(\eta)$ is convex $\forall\, \eta \in \mathbb{R}$ which implies ℓ is a Lipschitz continuous function

$$\begin{gathered} \forall N > 0 \quad \exists L_N \, s.t. \, |\ell(y,t_1) - \ell(y,t_2)| \le L_N |t_1 - t_2| \\ \forall t_1,t_2 \in [-N,N] \,, \forall y \in [-M,M] \\ \exists C_0 \, s.t. \, \forall y \in [-M,M] \quad \ell(y,0) \le C_0 \end{gathered}$$

where L_N, C_0 values depend on the specific form of the loss function. For regression using the inequality $||a|^p - |b|^p| \le p|a-b|\max(a,b)|^{p-1}$ we consider the following losses:

$$\begin{aligned} &square \quad \ell_2(y,t) = |y-t|^2 \quad [L_N = 2N + M \, C_0 = M^2] \\ &absolute\, value \quad \ell_1(y,t) = |y-t| \quad [L_N = 1 \, C_0 = M] \\ &\varepsilon - insensitive \quad \ell_\varepsilon(y,t) = \max(0, |y-t| - \varepsilon) \quad [L_N = 1 \, C_0 = M] \end{aligned}$$

In the supervised setting when we put $t = f(x)$ with a slight abuse of notation we write the loss function as $\ell(f(x)) = \ell(f(x),x)$. The classical loss function for regression is the square loss $\ell_2(y,t)$ mainly because it simplifies the mathematical treatment and leads naturally to estimates which can be computed rapidly. Besides it is well-known that the squares risk is minimized by the *conditional mean* of Y given x [40] i.e.

$$f^*_{\ell_2,P}(x) = \underset{f:X\to\mathbb{R}}{\arg\min}\, R_{\ell_2,P}(f) = \int_Y y dP(y|x) = \mathbb{E}[Y|X=x].$$

Though this loss seems mathematically rather easy to handle and the corresponding learning algorithms are often computational feasible, it is well-known that minimizing an empirical risk based on the squares loss is a method which is quite sensitive to outliers. Therefore, a number of more robust surrogate loss functions have been proposed. Hence, from a practical point of view, there are situations in which a different loss is more appropriate.

Moreover, $\varepsilon-$insensitive loss function $\ell_\varepsilon(y,t)$ promises algorithmic advantages in terms of sparseness compared to the absolute and square losses when used in the kernel-based regression. Besides if the conditional distributions of Y given x $P(\cdot|x)$ are known to be symmetric it can be shown [82] that using $\varepsilon-$insensitive loss the only minimizer of $R_{\ell_\varepsilon,P}(f)$ is represented by the the *conditional median* of Y given x

$$f^*_{\ell_\varepsilon,P}(x) = \underset{f:X\to\mathbb{R}}{\arg\min}\, R_{\ell_\varepsilon,P}(f) = f^*_{1/2,P}(x)$$

Another possibility which could be explored is the so called *quantile regression* [82] which tries to estimate the conditional quantile meaning the set valued function:

$$f^*_{\tau,P}(x) = \{\, t \in \mathbb{R}\, P((-\infty,t]|x) \geq \tau \quad and \quad P((t,\infty]|x) \geq 1-\tau\} \tag{3.5}$$

where $\tau \in (0,1)$ is a fixed constant and $P(\cdot|x)$, $x \in X$ is the conditional probability. This is strictly related to the $\tau-$pinball loss function defined as

$$\ell_\tau(y,t) = (\tau-1)(y-t) \quad if \quad (y-t) < 0 \quad and \quad \tau(y-t) \quad if \quad (y-t) > 0 \tag{3.6}$$

Empirical methods estimating quantiles using the *pinball loss* may obtain functions f_D for which $R_{\ell_\tau,P}(f_D)$ is close to $R^*_{\ell_\tau,P}$ with high probability. In general this only implies that f_D is close to $f^*_{\tau,P}$ in a weak sense meaning that we may obtain $f_{D_n} \to f^*_{\tau,P}$ in probability for all sequences $\{f_{D_n}\}$ with $R_{\ell_\tau,P}(f_{D_n}) \to R^*_{\ell_\tau,P}$. As a result the approximate risk minimizers approximate the unique risk minimizer in probability. However under realistic assumptions on the distribution P one may have an inequality of the form

$$\|f - f^*_\tau\|_{2,\nu} \leq c_p[R_{\ell_\tau,P}(f_D) - R^*_{\ell_\tau,P}] \tag{3.7}$$

where c_P depends on the distribution P (see [82]) so that we may hope to control the error of f_D in a wider sense. A similar argument may be invoked also for the $\varepsilon-$insensitive loss function.

Since the distribution P generating the input/output pairs is unknown, the risk $R_{\ell,P}$ is unknown and consequently we cannot directly find f. To resolve this problem, it is tempting to replace the risk $R_{\ell,P}(f)$ in by its empirical counterpart (the empirical risk)

$$R_{\ell,D}(f) = \mathbb{E}_n[\ell(y,f(x))] = \frac{1}{n}\sum_{t=1}^{n} \ell(y_t, f(x_t)).$$

Unfortunately even though the law of large numbers shows that $R_{\ell,D}$ is an approximation of $R_{\ell,P}$ for each single f, solving $\inf_{f:X\to\mathbb{R}} R_{\ell,D}(f)$ does not in general lead to an approximate minimizer of $R_{\ell,P}(\cdot)$. This is an example of overfitting in which the learning method produces a function that models too closely the output values in Z_n bringing to poor performance on future data. One common way to avoid overfitting is to choose a small set of functions $f \in \mathcal{H}$ that is assumed to contain a reasonably good approximation of the solution. Then, instead of minimizing $R_{\ell,D}$ over all functions, one minimizes only over $\mathcal{H}$ solving $\inf_{f\in\mathcal{H}} R_{\ell,D}(f)$.

Building a good non-parametric predictor f can be achieved using kernel-based regression, which finds a minimizer $f_{P,\lambda}$ of the regularized empirical risk

$$f_{P,\lambda} = \arg\min_{f\in\mathcal{H}} \{ R_{\ell,P}(f) + \lambda \|f\|^2_{\mathcal{H}} \} \tag{3.8}$$

where $\lambda > 0$ is a regularization parameter to reduce the danger of overfitting, $\mathcal{H}$ is a RKHS of a kernel $\kappa : X \times X \to \mathbb{R}$ and $\ell(y,\cdot) : \mathbb{R} \to [0,\infty)$ is convex for all $y \in Y$. Because section 3.4 is strictly convex in f, the minimizer $f_{P,\lambda}$ is uniquely determined and a simple gradient descent algorithm can be used to find it. However, for specific losses such as $\varepsilon-$insensitive more efficient algorithmic approaches are used in practice. Using the sequence of collected samples Z_n the corresponding empirical problem can be formulated as

$$f_{D,\lambda_n} = \arg\min_{f\in\mathcal{H}} \{ R_{\ell,D}(f) + \lambda \|f\|^2_{\mathcal{H}} \} \tag{3.9}$$

Following the interpretation that a small risk is desired, one tries to find a predictor whose risk is close to the optimal risk $R^*_{\ell,P}$ which is a much stronger requirement than convergence in probability of $R_{\ell,P}(f)$ to $R_{\ell,P,\mathcal{H}} = \inf_{f\in\mathcal{H}} R_{\ell,P}(f)$ as is not obvious whether $R_{\ell,P,\mathcal{H}} = R^*_{\ell,P}$ or not even for large Hilbert spaces $\mathcal{H}$. Throughout this work we assume that for some $M \geq 0$ the distribution $P(\cdot|x)$ is almost everywhere supported on $[-M,M]$ that is $|y| \leq M$ almost surely with respect to P which means that for the loss functions we are taking into account always $|f^*_{\ell,P}| \leq M$ (truncation assumption). The efficiency of the algorithm in section 3.4 can be measured by the difference between f_{D,λ_n} and the regression function $f^*_{\ell,P}$. One can see [40] that using the squares loss function $\ell_2(y,t)$ the approximation error can be expressed as:

$$\|f_{D,\lambda_n} - f^*_{\ell_2 P}\|^2_{2,\nu_x} = R_{\ell_2,P}(f) - R^*_{\ell_2,P} = \mathbb{E}[(y - f_{D,\lambda_n}(x))^2 | Z_n] - \mathbb{E}[(y - f^*_{\ell_2,P}(x))^2] \tag{3.10}$$

for any distribution P on $X \times Y$. On the contrary using the $\varepsilon-$insensitive loss function $\ell_\varepsilon(y,t)$ we can only lower bound the error [82] using the Liptschitz continuity property of ℓ_ε as

$$|\, R_{\ell_\varepsilon,P}(f_{D,\lambda_n}) - R^*_{\ell_\varepsilon,P}\,| = |\, \mathbb{E}[\ell_\varepsilon(y, f_{D,\lambda_n}(x))|\, Z_n] - \mathbb{E}[\ell_\varepsilon(y, f^*_{\ell_\varepsilon,P}(x))]\,| \leq \|f_{D,\lambda_n} - f^*_{\ell_\varepsilon,P}\|_{1,\nu_x}. \tag{3.11}$$

Hence, using $\varepsilon-$insensitive losses one typically obtains functions f_{D,λ_n} for which $R_{\ell_\varepsilon,D}(f)$ is close to $R^*_{\ell_\varepsilon,P}$ with high probability. In general this only implies that f_{D,λ_n} can be close to $f^*_{\ell_\varepsilon,P}$ in a weak sense. In fact using convex loss functions one may obtain $f_{D,\lambda_n} \to f^*_{\ell,P}$ in probability for all sequences f_{D,λ_n} with $R_{\ell_\varepsilon,D}(f) \to R^*_{\ell_\varepsilon,P}$ and the approximate risk minimizers approximate the unique risk minimizer $f^*_{\ell,P}$ in probability [82]. Nevertheless

we may get a stronger notion of approximation if we make some realistic assumptions on the probability distribution P assuming an inequality of the form

$$\begin{aligned} \|f_{D,\lambda_n} - f^*_{\ell_\varepsilon,P}\|_{2,\nu_x} &\leq c_P \Big[R_{\ell_\varepsilon,P}(f_{D,\lambda_n}) - R^*_{\ell_\varepsilon,P} \Big] \\ &= c_P \Big[\mathbb{E}[\ell_\varepsilon(y, f_{D,\lambda_n}(x))|\, Z_n] - \mathbb{E}[\ell_\varepsilon(y, f^*_{\ell_\varepsilon,P}(x))] \Big]. \end{aligned} \tag{3.12}$$

The constant $c_P \geq 1$ depends on the distribution P and section 3.4 can be used to bound the approximation error. It is out of the scope of this work to further analyze under which conditions on P section 3.4 holds which we will take as an assumption when dealing with ℓ_ε losses.

Estimation of the error using section 3.4 for ℓ_2 losses or section 3.4 for the ℓ_ε losses depends on P and $\mathcal{H}$. We should expect that the minimizer of the regularized empirical error f_{D,λ_n} to be a good approximation of the minimizer $f^*_{\ell,P}$ of $R_{\ell,P}(f)$ as $n \to \infty$ and $\lambda = \lambda(n) \to 0$ which is actually true if $f^*_{\ell,P}$ can be approximated using functions from $\mathcal{H}$ and measured by the regularization error defined as

$$\mathcal{A}(\lambda) = \inf_{f \in \mathcal{H}} \{ \|f - f^*_{\ell,P}\|^q_{p,\nu_x} + \lambda \|f\|^2_{\mathcal{H}} \} \tag{3.13}$$

where $p = q = 2$ for ℓ_2 losses and $q = 1$ and $p = 2$ for ℓ_1 of ℓ_ε losses. Now the regularization function can be written as

$$f_\lambda = \arg\min_{f \in \mathcal{H}} \{ \|f - f^*_{\ell,P}\|^q_{p,\nu_x} + \lambda \|f\|^2_{\mathcal{H}} \} \tag{3.14}$$

Since the minimization in section 3.4 rely on the discrete quantity $R_{\ell,D}(f)$ the approximation $f^*_{\ell,P}$ by f_{D,λ_n} involves the capacity of the function space $\mathcal{H}$ which can be measured using the covering numbers. If we define the loss space as $\mathcal{L}_\mathcal{H} = \{\, \ell(y, f(x)) \quad x \in X,\, y \in Y,\, f \in \mathcal{H}\}$ the covering number of $\mathcal{L}_\mathcal{H}$ and $\mathcal{H}$ can be easily related using the Lipschitz condition $|\ell(y, f(x_1)) - \ell(y, f(x_2))| \leq L_M |f(x_1) - f(x_2)|$ and it can be shown ([59]) that the following condition holds :

$$\mathcal{N}_p(\varepsilon, \mathcal{L}_\mathcal{H}(Z^n), \|\cdot\|^q_{p,n}) \leq \mathcal{N}_p(\varepsilon / L_M, \mathcal{H}(X^n), \|\cdot\|^q_{p,n})$$

where for the ℓ_ε and ℓ_1 norm $L_M = 1$.

The following proposition from [104] and extended to take into account both ℓ_2 and ℓ_ε losses gives an estimation of the excess error for the regularization function f_{D,λ_n} by decomposition

Proposition 3.2 (Extended from [104])**.** *Let $f_\lambda \in \mathcal{H}$ and f_{D,λ_n} defined as in section 3.4. Then results*

$$c_P[R_{\ell,P}(f_{D,\lambda_n}) - R^*_{\ell,P}] \leq c_P[R_{\ell,P}(f_{D,\lambda_n}) + \lambda \|f_{D,\lambda_n}\|^2_{\mathcal{H}} - R^*_{\ell,P}]$$

which can be bounded by $\mathcal{E}_{D,\mathcal{H}} + \mathcal{D}(\lambda)$ defining the regularization error as

$$\mathcal{D}(\lambda) = c_P[R_{\ell,P}(f_\lambda) + \lambda \|f_\lambda\|^2_{\mathcal{H}} - R^*_{\ell,P}]$$

and the estimation error as

$$\mathcal{E}_{D,\mathcal{H}} = c_P[R_{\ell,P}(f_{D,\lambda_n}) - R_{\ell,D}(f_{D,\lambda_n}) + R_{\ell,D}(f_\lambda) - R_{\ell,P}(f_\lambda)]$$

with $c_P = 1$ for ℓ_2 losses and $c_P \geq 1$ for ℓ_ε losses if section 3.4 holds. Defining the two variables

$$\zeta_1(x,y) = \ell(y, f_{D,\lambda_n}(x)) - \ell(y, f^*_{\ell,P}(x))$$

and

$$\zeta_2(x,y) = \ell(y, f_\lambda(x)) - \ell(y, f^*_{\ell,P}(x))$$

the sample error can be written as

$$\mathcal{E}_{D,\mathcal{H}} = c_P[\mathbb{E}[\zeta_1] - \mathbb{E}_n[\zeta_1]] + c_P[\mathbb{E}_n[\zeta_2] - \mathbb{E}[\zeta_2]]$$

Proof is omitted for brevity but can be easily obtained using a similar procedure used in [104].

The rate of the regularization error $\mathcal{D}(\lambda)$ is important for bounding both estimation and regularization error. The decay of λ and $n \to \infty$ determines the size of the hypothesis space and hence the sample error estimate. If the RKHS is large enough we may consider that the approximation error $\mathcal{D}(\lambda) = 0$ as $\lambda \to 0$ and $n \to \infty$ as shown for bounded spaces X and Y and the squared losses in [40] and we only have to bound the estimation error $\mathcal{E}_{D,\mathcal{H}}$.

3.5 Regularization and Support Vector Regression

Theoretical results in [24] show that kernel regression methods using a loss function with bounded first derivative in combination with a bounded and rich enough continuous kernel like RBF kernel are not only consistent and computational tractable, but also offer attractive robustness properties. SVR shows good performance in practical applications and a robust theoretical justification in terms of universal consistency and learning rates when training samples come from an i.i.d. process [83]. Sometimes i.i.d. assumption might not be strictly justified in real-world problems. Some ML applications (system diagnosis, market prediction, speech recognition) are not i.i.d. processes. Moreover, samples are often gathered from different sources and might not be identically distributed. SVR in such non-i.i.d. scenarios have no theoretical justification but they are sometimes applied successfully. Nevertheless, for any process that satisfies laws of large numbers, a sequence of regularization parameters exists such that the corresponding SVR can be considered consistent. Universal consistency for stationary processes has general negative results on this kind of sequence and regularization parameters cannot be adaptively chosen and should rely on stochastic properties of the process. However, if the process satisfies certain mixing properties an adequate regularization sequence can be chosen.

Whenever $\varepsilon-$insensitive loss functions are used in SVR the classical geometrical formulation can be obtained looking at the regularization problem

$$\frac{1}{n}\sum_{t=1}^{n} \ell_\varepsilon(y_t, f(x_t)) + \lambda \|f\|^2_{\mathcal{H}} \tag{3.15}$$

which is equivalent (meaning that the same function minimizes both functionals) [34] to solve the following optimization problem in which an additional set of parameters is introduced

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\xi^*} \quad & C\sum_{t=1}^{n}(\xi_t+\xi_t^*)+\tfrac{1}{2}\|\mathbf{w}\|_{\mathcal{H}_0}^2 \\ s.t. \quad & f_{\mathbf{w},b}(x_t)-y_t \le \varepsilon+\xi_t \\ & y_t - f_{\mathbf{w},b}(x_t) \le \varepsilon+\xi_t^* \\ & \xi_t,\xi_t^* \ge 0 \quad \forall (x_t,y_t)\in Z_n \end{aligned} \tag{3.16}$$

The model function is expressed by $f_{\mathbf{w},b}(x_t)=\langle\Phi(x_t),\mathbf{w}\rangle+b$, $\lambda=\frac{1}{2nC}$ and ξ,ξ^* are slack variables. Eventually the problem can be solved through the technique of Lagrange multipliers. Hereafter with a slight abuse of notation we keep referring only to the Hilbert space $\mathcal{H}$ (without explicitly mention $\mathcal{H}_0$) and using both geometric and regularized version of SVR as the case may be.

The main difference between the regularized and the geometric formulation of SVR can be found on the meaning of the constant b [82]. The geometrical approach considers an Hilbert space $\mathcal{H}_0$ and define a function $f_{\mathbf{w},b}$ in terms of an affine hyperplane specified by $(\mathbf{w},b)$. In the regularized formulation one directly considers an RKHS $\mathcal{H}$ with the functions contained in it. However the two approaches are equivalent whenever we fix b and the functions $\langle\Phi(\cdot),\mathbf{w}\rangle$ with $\mathbf{w}\in\mathcal{H}_0$ form an RKHS $\mathcal{H}$ whose norm can be computed by

$$\|f\|_{\mathcal{H}}=\inf\{\|\mathbf{w}\|_{\mathcal{H}_0} : \mathbf{w}\in\mathcal{H}_0 \quad with \quad f=\langle\Phi(\cdot),\mathbf{w}\rangle\} \tag{3.17}$$

The offset term makes a real difference and, in general, the decision functions produced by both approaches might be different.

Some final remarks about SVR: the algorithm is based on solid theoretical guarantees, the solution returned is sparse, and it allows a natural use of positive definite symmetric kernels, which extend the algorithm to non-linear regression solutions. SVR also admits favorable stability properties as already mentioned previously. However, one drawback of the algorithm is that it requires the selection of two parameters, C and ε. These can be selected via cross-validation but this requires a relatively large validation set. Some heuristics are often used to guide the search for their values: C is searched near the maximum value of the labels in the absence of an offset $(b=0)$ and for a normalized kernel, and ε is chosen close to the average difference of the labels. Moreover, the value of ε determines the number of support vectors and the sparsity of the solution.

3.6 Regularized API With Bellman Residuals Minimization

As already illustrated in section 2.8, a way to implement API is through BRM. API proceeds at iteration k evaluating π_k choosing Q_k such that the Bellman Residuals to be small (i.e. is the approximate fixed point of T^{π_k}). API calculates $\pi_{k+1}=\hat{\pi}(\cdot,Q_k)$ producing the sequence $Q_0\to\pi_1\to Q_1\ldots.$. Hence for the sequence $\{Q_k\}_{k=0}^{K-1}$ the Bellman Residuals can be defined at each iteration as $\varepsilon_k^{BR}=Q_k-T^{\pi_k}Q_k$.

Algorithm 1 Offline $API - BRM_\varepsilon$

Require: $(k, \kappa, \lambda, \gamma, n)$
  initialize $Q_0(s,a)$ arbitrarily for all (s,a)
  for all k **do**
    repeat
      $\pi_k(\cdot) \leftarrow \hat{\pi}(\cdot, \hat{Q}_{k-1})$
      generate/update training sample set D_n^k and Π_n^k
      $\hat{Q}_k \leftarrow Batch - API - BRM_\varepsilon(\Pi_k, D_n^k, \kappa, \lambda)$
    until k=K
  end for
  return

Algorithm 2 Online $API - BRM_\varepsilon$

Require: $(\kappa, \lambda, \gamma, \varepsilon_k\, function)$
  initialize $Q_0(s,a)$ arbitrarily for all (s,a)
  for all time step k **do**
    update greedy policy $\pi_k(\cdot) \leftarrow \hat{\pi}(\cdot, \hat{Q}_{k-1})$
    choose action $a_k = \{\pi_k(\cdot) \quad w.p.\, 1-\varepsilon_k \quad \vee \quad \text{uniform random action} \quad w.p.\, \varepsilon_k\}$
    apply a_k and measure next state s_{k+1} and reward r_{k+1}
    update training sample set $D_k \leftarrow D_{k-1} \cup (s_k, a_k, r_{k+1}, s_{k+1})$ and $\Pi_k \leftarrow \Pi_{k-1} \cup \pi_k(\cdot, s_{k+1})$
    $\hat{Q}_k \leftarrow Incremental - API - BRM_\varepsilon(\Pi_k, D_k, \kappa, \lambda)$
  end for
  return

Hereafter, we focus on two different ways to implement API using BRM to find the approximating value function. Assuming data collection done in the offline sampling setting and the data processing in batch mode, a way to implement API is presented in Algorithm 1. In this case the behavior policy π_{b_k} is different from the training policy π_k and we assume the data are non i.i.d. distributed unless π_{b_k} is known to be completely random. On the other way whenever we consider that interaction with the environment is possible, we may collect data using the online sampling setting. Data processing is still possible in both batch or incremental mode but in the Algorithm 2 we present an incremental mode which can be more efficient in practical situations. In this case behavior and learning policy can be assumed optimistically to be the same or eventually to change every few iteration. Exploration is done using an ε-greedy policy. We assume that sample are in general non-i.i.d. distributed.

Standard BRM algorithm as in [8] uses least squares regression to approximate the action minimizing the Bellman Error given the distribution of the input data ν defined as

$$L_{BRM_2}(Q,\pi) = \|Q - T^\pi Q\|_\nu^2 = \int |Q(s,a) - T^\pi Q(s,a)|^2 d\nu(s,a). \tag{3.18}$$

Using the sample data set D_n and the policy process Π_n the finite-sample empirical estimate $\hat{L}_{BRM_2}(Q, \Pi_n, D_n)$ given the loss function ℓ_2 can be written as

$$\hat{L}_{BRM_2}(Q, \Pi_n, D_n) = \mathbb{E}_n[|Q(s_t,a_t) - \hat{T}^\pi Q(s_t,a_t)|^2]. \tag{3.19}$$

BRM problem can be solved in the context of regularized API method using a RKHS by taking linear combinations of the form $Q(s,a) = \sum_{t=1}^{n} \beta_t \kappa(s_t, a_t, s, a)$ where $\beta_t \in \mathbb{R}$. Accordingly the regularized BRM problem can be written as

$$\hat{Q} = \arg\min_{Q \in \mathcal{H}} \{\hat{L}_{BRM_2}(Q, \Pi_n, D_n) + \lambda_n \|Q\|^2_{\mathcal{H}}\}. \tag{3.20}$$

Unfortunately this approach brings to a biased estimate of $L_{BRM_2}(Q,\pi)$. In fact when evaluating the quadratic terms in the summation gives the unwanted additional variance term:

$$\mathbb{E}[|Q(s_t,a_t) - \hat{T}^\pi Q(s_t,a_t)|^2 | s_t, a_t, \pi_t] = |Q(s_t,a_t) - T^\pi Q(s_t,a_t)|^2 + Var[r_t + \gamma \sum_{a' \in A} \pi(a'|s'_t) Q(s'_t, a')].$$

As a result

$$\mathbb{E}[\,\mathbb{E}_n[|Q - \hat{T}^\pi Q|^2] \,|\, \Pi_n, D_n] = \|Q - T^\pi Q\|^2_\nu + \mathbb{E}[\mathbb{E}_n[T^\pi Q - \hat{T}^\pi Q|^2] \,|\, \Pi_n, D_n]$$

leading to $\mathbb{E}[\hat{L}_{BRM_2}(Q, \Pi_n, D_n) | \Pi_n, D_n] \neq L_{BRM_2}(Q,\pi)$. To overcome this problem a common suggestion is to use uncorrelated, or double sampling $\hat{L}_{BRM_2}(Q, \Pi_n, D_n)$. Accordingly for each state and action it means that at least two next states should be generated for each (s_t, a_t). In principle reuse of samples close in space is also possible but this approach requires the definition of a proximity function.

An alternative approach presented in [6] defines a modified BRM optimization problem apt to cancel the unwanted variance term by introducing an auxiliary function h and a new loss function defined as

$$L_{BRMh_2}(Q, h, \pi) = \|Q - T^\pi Q\|^2_\nu - \|h - T^\pi Q\|^2_\nu. \tag{3.21}$$

It can be shown [6] that the empirical version of this loss is an unbiased estimator and one has to reformulate the problem as a coupled optimization expressed as:

$$\begin{aligned} \hat{h} &= \arg\min_{h \in \mathcal{H}} \{\, \mathbb{E}_n[|h - \hat{T}^\pi Q|^2] + \lambda_h \|h\|^2_{\mathcal{H}} \,\} \\ \hat{Q} &= \arg\min_{Q \in \mathcal{H}} \{\, \mathbb{E}_n[|Q - \hat{T}^\pi Q|^2] - \mathbb{E}_n[|h - \hat{T}^\pi Q|^2] + \lambda_Q \|Q\|^2_{\mathcal{H}} \,\}. \end{aligned} \tag{3.22}$$

We may evaluate the solution of the regularized optimization problem in the section 3.6 which may be expressed in closed form using the Representer Theorem. The optimal functions can be expressed as a linear combination of the kernel functions centered into the training samples. However this regression problem is less intuitive and more difficult to solve also having a dense solution meaning that the whole set of D_n participate to the solution. Furthermore it can be solved as batch problem while one can not take profit of the incrementality and it also needs a model for the cMDP. Some theoretical convergence guarantees of this method are reported in [6] (parametric approximation) and in [35] (non-parametric approximation) working for model based batch solution and using i.i.d. data samples

3.7 Regularized $API - BRM_\varepsilon$ Algorithm Formulation

As all the ingredients have been presented, starting from this section we are ready to introduce our method aiming to solve the BRM problem using SVR. $API - BRM_\varepsilon$ can

be formulated considering the BR for a continuous state and finite actions cMDP as $BR(s,a) = Q(s,a) - T^{\pi}Q(s,a) = D^{\pi}Q(s,a) - r(s,a)$ where the approximating function to be considered is

$$D^{\pi}Q(s,a) = Q(s,a) - \gamma \int \mathcal{P}(ds'|s,a) \sum_{a' \in A} \pi(a'|s')Q(s',a')$$

i.e

$$D^{\pi}Q(s,a) = \mathbb{E}[\, Q(s,a) - \gamma \sum_{a' \in A} \pi(a'|s')Q(s',a')|s,a] = \mathbb{E}[\, \hat{D}^{\pi}Q(s,a,s')|s,a]$$

while $r(s,a) = \mathbb{E}[\hat{r}|s,a]$ and the BRM using the $\varepsilon-$insensitive loss can be written as

$$L_{API-BRM_{\varepsilon}}(Q,\pi) = \mathbb{E}[\ell_{\varepsilon}(D^{\pi}Q - r)] = \mathbb{E}[\ell_{\varepsilon}(Q - T^{\pi}Q)] \tag{3.23}$$

Using the data sets Π_n, D_n the empirical estimate becomes:

$$\hat{L}_{API-BRM_{\varepsilon}}(Q,\Pi_n,D_n) = \mathbb{E}_n[\ell_{\varepsilon}(\hat{D}^{\pi}Q - \hat{r})] = \mathbb{E}_n[\ell_{\varepsilon}(Q - \hat{T}^{\pi}Q)] \tag{3.24}$$

with $\hat{D}^{\pi}Q(s_t,a_t,s'_t) = Q(s_t,a_t) - \gamma \sum_{a'_t \in A} \pi_t(a'_t|s'_t)Q(s'_t,a'_t)$. Hence the $API - BRM_{\varepsilon}$ optimization problem becomes

$$\hat{Q} = \arg\min_{Q \in \mathcal{H}} \{\, \hat{L}_{API-BRM_{\varepsilon}}(Q,\Pi_n,D_n) + \lambda_n \|Q\|^2_{\mathcal{H}} \,\} \tag{3.25}$$

where the regularization term use the norm in the Hilbert space $\mathcal{H}$. $API - BRM_{\varepsilon}$ shows a remarkable sparsity property in the solution which essentially relies on the training support vectors. $\hat{L}_{API-BRM_{\varepsilon}}(Q,\Pi_n,D_n)$ is an almost unbiased estimator of $L_{API-BRM_{\varepsilon}}(Q,\pi)$ as results evaluating the expectation over the empirical losses

$$\begin{aligned} &\mathbb{E}[\hat{L}_{API-BRM_{\varepsilon}}(Q,\Pi_n,D_n)|\Pi_n,D_n] = \mathbb{E}[\,\mathbb{E}_n[\ell_{\varepsilon}(\hat{D}^{\pi}Q - \hat{r})]|\Pi_n,D_n] \\ &= \mathbb{E}[\,\mathbb{E}_n[\ell_{\varepsilon}(Q - \hat{T}^{\pi}Q)]|\Pi_n,D_n] \geq \mathbb{E}[\,\ell_{\varepsilon}(D^{\pi}Q - r)] = L_{API-BRM_{\varepsilon}}(Q,\pi) \end{aligned} \tag{3.26}$$

where we used the Jensen's inequality $\ell(\mathbb{E}[X]) \leq \mathbb{E}[\,\ell(X)]$ holding for any convex function ℓ (we used $D^{\pi}Q(s,a) = \mathbb{E}[\,\hat{D}^{\pi}Q(s,a,s')|s,a]$, $r(s,a) = \mathbb{E}[\hat{r}|s,a]$ and $T^{\pi}Q = \mathbb{E}[\hat{T}^{\pi}Q]$). In practice the empirical estimate can be biased whenever slacks are presents i.e. the errors on the regression function are above the fixed threshold ε. It is unbiased when the error is contained in the resolution tube of the SVR. Nevertheless the choice of the SVR parameters C and ε gives a way to control this effect so the bias cannot be considered random. In the experimental section we show how our method is able to identify a near optimal policy without being affected by the bias which can be obtained using a fine tuning of the approximation parameters. In the next section analyzing the implementation of $API - BRM_{\varepsilon}$ we will show how this may affect the solution. Moreover if we want to evaluate the error according to the expression 3.11 we may write

$$\left| \mathbb{E}[\ell_{\varepsilon}(\hat{D}^{\pi}Q - \hat{r})|\Pi_n,D_n] - \mathbb{E}[\ell_{\varepsilon}(r - \hat{r})] \right| \leq \|Q - T^{\pi}Q\|_{1,\nu} \tag{3.27}$$

and eventually if the condition of section 3.4 are met we might also write

$$\|Q - T^{\pi}Q\|_{2,\nu} \leq c_P[\mathbb{E}[\ell_{\varepsilon}(\hat{D}^{\pi}Q - \hat{r})|\Pi_n,D_n] - \mathbb{E}[\ell_{\varepsilon}(r - \hat{r})]] \tag{3.28}$$

which can be used to bound the sample error of BRM using SVR.

3.8 API with SVR

In order to gain insights into the regularized $API-BRM_\varepsilon$ we analyze the SVR solution in the dual solving the geometrical version of the SVR and the corresponding constrained optimization problem using the Lagrangian multipliers. At the first we consider the infinite sample formulation and then we move to the finite sample version. Consider the subset of observed samples D_n and express the approximation of the value function using a linear architecture as

$$Q(s,a) = \langle \Phi(s,a), \mathbf{w} \rangle + b \tag{3.29}$$

where $\mathbf{w} = (w_1, ..., w_d)^T$ is the weight vector and $\Phi(s,a) = (\phi_1(s,a), ..., \phi_d(s,a))^T$ the features vector of the point (s,a) from which we may build the kernel function $\kappa(s_t,a_t,s,a) = \langle \Phi(s_t,a_t), \Phi(s,a) \rangle$. The action value function belongs the Hilbert space $Q \in \mathcal{H}$ so it does the weight vector $\mathbf{w} \in \mathcal{H}$. Using the Representer Theorem we also know that the function in $\mathcal{H}$ can be expressed as linear combination of the elements in the span $\Phi_{span} = \{\Phi(s,a) = \kappa(\cdot,s,a) \quad (s,a) \in S \times A\}$ which can be expressed as $Q(s,a) = \sum_t \alpha_t \kappa(s,a,s_t,a_t)$. Using the definition of the Bellman operator $T^\pi Q$ the BR at each training point for a fixed policy π we may write:

$$\begin{aligned} BR(s_t,a_t) &= Q(s_t,a_t) - T^\pi Q(s_t,a_t) = D^\pi Q(s_t,a_t) - r(s_t,a_t) \\ &= Q(s_t,a_t) - \gamma \int \mathcal{P}(ds'|s_t,a_t) \cdot \textstyle\sum_{a' \in A} \pi(a'|s') \cdot Q(s',a') - r(s_t,a_t) \end{aligned} \tag{3.30}$$

and substituting the functional form of section 3.8 yields

$$BR(s_t,a_t) = \langle \Phi(s_t,a_t), \mathbf{w} \rangle - \gamma \int \mathcal{P}(ds'|s_t,a_t) \cdot \sum_{a' \in A} \pi(a'|s') \cdot \langle \Phi(s',a'), \mathbf{w} \rangle + (1-\gamma)b - r(s_t,a_t)$$

expressing the BR using the weight $\mathbf{w}$ and the features mapping $\Phi(\cdot)$. Policy and the cMDP dynamic are not included into the Hilbert space $\mathcal{H}$. An alternative way to express the BR is through the combination of the two terms using the Bellman feature mapping

$$\Psi^\pi(s_t,a_t) = \Phi(s_t,a_t) - \gamma \int \mathcal{P}(ds'|s_t,a_t) \textstyle\sum_{a' \in A} \pi(a'|s') \Phi(s',a') \tag{3.31}$$

which takes into account the structure of the cMDP dynamics and the policy. BR are now expressed as

$$BR(s_t,a_t) = \langle \Psi^\pi(s_t,a_t), \mathbf{w} \rangle + (1-\gamma)b - r(s_t,a_t)$$

and with the Bellman feature vector $\Psi^\pi(s_t,a_t)$ we may build the Bellman kernel

$$\tilde{\kappa}(s_t,a_t,s,a) = \langle \Psi^\pi(s_t,a_t), \Psi^\pi(s,a) \rangle$$

The function $D^\pi Q$ and weight vector $\mathbf{w}$ belong to the Bellman Hilbert space $\mathcal{H}_{\Psi^\pi}$. Using the Representer Theorem the function in $\mathcal{H}_{\Psi^\pi}$ can be expressed as linear combination of the elements in the span $\Psi_{span} = \{\Psi(s,a) = \tilde{\kappa}(\cdot,s,a) \quad (s,a) \in S \times A\}$ which can be expressed as $D^\pi Q(s,a) = \sum_t \beta_t \tilde{\kappa}(s,a,s_t,a_t)$. With this choice the policy as well as the MDP dynamic are directly incorporated into the Hilbert space $\mathcal{H}_{\Psi^\pi}$. Under simple assumptions one may shows that if the set of vectors $\mathcal{V}_\Phi = \{\Phi(s_t,a_t) \,|(s_t,a_t) \in S \times A\}$ are linearly independent then the set of vectors $\mathcal{V}_\Psi = \{\Psi^\pi(s_t,a_t) \,|(s_t,a_t) \in S \times A\}$ are also linearly independent (see also [13] for a similar discussion).

Theorem 3.2. *Assume the kernel* $\kappa(s_t,a_t,s,a) = \langle \Phi(s_t,a_t), \Phi(s,a) \rangle$ *is non degenerate then for any fixed policy* π *the Bellman kernel* $\tilde{\kappa}^\pi(s_t,a_t,s,a) = \langle \Psi^\pi(s_t,a_t), \Psi^\pi(s,a) \rangle$ *is also non degenerate.*

Proof. Considering the real space vector $\mathcal{V}_\Phi$ spanned by the set of vectors $\Phi(s_t,a_t)$ then the corresponding space vector $\mathcal{V}_\Psi$ spanned by the set of vectors $\Phi(s_t,a_t)$ comes from a linear combination of vectors in $\mathcal{V}_\Phi$ as

$$\Psi^\pi(s_t,a_t) = \Phi(s_t,a_t) - \gamma \int \mathcal{P}(ds'|s_t,a_t) \sum_{a' \in A} \pi(a'|s') \Phi(s',a') = \mathcal{D}^\pi \Phi(s_t,a_t) \quad (3.32)$$

where $\mathcal{D}^\pi$ is a linear operator mapping vectors $\Phi(s_t,a_t) \in \mathcal{V}_\Phi$ to vectors $\Psi^\pi(s_t,a_t) \in \mathcal{V}_\Psi$. Now if we look at this operator it has full rank because its eigenvalues depends on the combination of the unity and stochastic matrices and the factor $\gamma < 1$ leading to the fact that the kernel of the $\mathcal{D}^\pi$ operator must have $dim(Ker(\mathcal{D}^\pi)) = 0$. As a result $dim(\mathcal{V}_\Phi) = dim(\mathcal{V}_\Psi)$ so the vectors $\Psi^\pi(s_t,a_t) \in \mathcal{V}_\Psi$ are linearly independent. From the linear independence of the vectors $\Psi^\pi(s_t,a_t) \in \mathcal{V}_\Psi$ it follows that if the kernel κ is non degenerate then also the corresponding Bellman kernel $\tilde{\kappa}$ is non degenerate. □

Hence, while the kernel κ corresponding to the features mapping $\Phi(\cdot)$ is given by

$$\kappa(s_t,a_t,s,a) = \langle \Phi(s_t,a_t), \Phi(s,a) \rangle \quad (s_t,a_t),\ (s,a)\ \in S \times A \quad (3.33)$$

the Bellman kernel $\tilde{\kappa}$ corresponding to the features mapping $\Psi^\pi(\cdot)$ is given by

$$\tilde{\kappa}^\pi(s_t,a_t,s,a) = \langle \Psi^\pi(s_t,a_t), \Psi^\pi(s,a) \rangle \quad (s_t,a_t),\ (s,a)\ \in S \times A \quad (3.34)$$

and the connection between the two kernels can be formally expressed using the linear operator $\mathcal{D}^\pi$ as

$$\begin{aligned} \tilde{\kappa}^\pi(s_t,a_t,s,a) &= \langle \Psi^\pi(s_t,a_t), \Psi^\pi(s,a) \rangle = \langle \mathcal{D}^\pi \Phi(s_t,a_t), \mathcal{D}^\pi \Phi(s,a) \rangle \\ \langle \Phi(s_t,a_t), (\mathcal{D}^\pi)^T \mathcal{D}^\pi \Phi(s,a) \rangle &\leq \lambda_1 \langle \Phi(s_t,a_t), \Phi(s,a) \rangle = \lambda_1 \kappa(s_t,a_t,s,a) \end{aligned} \quad (3.35)$$

with $\lambda_1 \leq 1$ max eigenvalue of $(\mathcal{D}^\pi)^T \mathcal{D}^\pi$. Plugging the operator $(\mathcal{D}^\pi)$ into the above expression we have

$$\begin{aligned} \tilde{\kappa}^\pi(s_t,a_t,s,a) &= \langle \Psi^\pi(s_t,a_t), \Psi^\pi(s,a) \rangle \quad (3.36) \\ &= \langle \Phi(s_t,a_t) - \gamma \int \mathcal{P}(dy'|s_t,a_t) \textstyle\sum_{b' \in A} \pi(b'|y') \Phi(s',a'), \\ &\quad \Phi(s,a) - \gamma \int \mathcal{P}(ds'|s,a) \textstyle\sum_{a' \in A} \pi(a'|s') \Phi(s',a') \rangle \\ &= \kappa(s_t,a_t,s,a) - \gamma \int \mathcal{P}(dy'|s_t,a_t) \textstyle\sum_{b' \in A} \pi(b'|y') \kappa(y',b',s,a) \\ &\quad - \gamma \int \mathcal{P}(ds'|s,a) \textstyle\sum_{a' \in A} \pi(a'|s') \kappa(s',a',s_t,a_t)) \\ &\quad + \gamma^2 \int \mathcal{P}(ds'|s_t,a_t) \textstyle\sum_{a' \in A} \pi(a'|s') \int \mathcal{P}(dy'|s,a) \sum_{b' \in A} \pi(b'|y') \kappa(s',a',y',b') \end{aligned}$$

Defining the auxiliary functions

$$\hat{\kappa}^\pi(s_t,a_t,s,a) = \kappa(s_t,a_t,s,a) - \gamma \int \mathcal{P}(dy'|s_t,a_t) \sum_{b' \in A} \pi(b'|y') \kappa(y',b',s,a)$$

we may also write

$$\tilde{\kappa}^{\pi}(s_t,a_t,s,a) = \hat{\kappa}^{\pi}(s_t,a_t,s,a) - \gamma \int \mathcal{P}(ds'|s_t,a_t) \sum_{a' \in A} \pi(a'|s') \hat{\kappa}^{\pi}(s,a,s',a')$$

Moving to the finite sample version consider the empirical Bellman operator $\hat{T}^{\pi}Q$ where the BR can be written as

$$\begin{aligned} \hat{BR}(s_t,a_t,s'_t) &= Q(s_t,a_t) - \hat{T}^{\pi}\, Q(s_t,a_t) = \hat{D}^{\pi} Q(s_t,a_t,s'_t) - \hat{r}_t \\ &= Q(s_t,a_t) - \gamma \textstyle\sum_{a'_t \in A} \pi(a'_t|s'_t) \cdot Q(s'_t,a'_t) - \hat{r}_t \end{aligned} \tag{3.37}$$

and substituting the functional form of section 3.8 yields

$$\hat{BR}(s_t,a_t,s'_t) = \langle \Phi(s_t,a_t), \mathbf{w} \rangle - \gamma \sum_{a'_t \in A} \pi(a'_t|s'_t) \cdot \langle \Phi(s'_t,a'_t), \mathbf{w} \rangle + (1-\gamma) b - \hat{r}_t$$

expressing the BR using the weight $\mathbf{w}$ and the features mapping $\Phi(\cdot)$. Then proceeding as before using the empirical Bellman features mapping

$$\hat{\Psi}^{\pi}(s_t,a_t,s'_t) = \Phi(s_t,a_t) - \gamma \textstyle\sum_{a'_t \in A} \pi(a'_t|s'_t) \Phi(s'_t,a'_t) = \hat{\mathcal{D}}^{\pi} \Phi(s_t,a_t) \tag{3.38}$$

with $\hat{\mathcal{D}}^{\pi}$ the empirical counterpart of $\mathcal{D}^{\pi}$ results

$$\Psi^{\pi}(s_t,a_t) = \mathbb{E}[\hat{\Psi}^{\pi}(s_t,a_t,s'_t)] = \mathbb{E}[\hat{\mathcal{D}}^{\pi} \Phi^{\pi}(s_t,a_t)] = \mathcal{D}^{\pi} \Phi^{\pi}(s_t,a_t)$$

The same kind of argument presented in theorem 3.2 applies also for the empirical operator $\hat{\mathcal{D}}^{\pi}$. Hence the empirical BR may be expressed as

$$\hat{BR}(s_t,a_t,s'_t) = \langle \hat{\Psi}^{\pi}(s_t,a_t,s'_t), \mathbf{w} \rangle + (1-\gamma) b - \hat{r}_t$$

with the empirical Bellman feature vector $\hat{\Psi}^{\pi}(s_t,a_t,s'_t)$ we may build the Bellman kernel

$$\tilde{\kappa}^{\pi}(s_t,a_t,s'_t,s,a,s') = \langle \hat{\Psi}^{\pi}(s_t,a_t,s'_t), \hat{\Psi}^{\pi}(s,a,s') \rangle$$

The function $\hat{D}^{\pi}Q$ as well as weight vector $\mathbf{w}$ belongs to the empirical Bellman Hilbert space $\mathcal{H}_{\hat{\Psi}^{\pi}}$. It is clear that by construction for any function $\hat{D}^{\pi}Q \in \mathcal{H}_{\hat{\Psi}^{\pi}}$ results $D^{\pi}Q = \mathbb{E}[\hat{D}^{\pi}Q] \in \mathcal{H}_{\Psi^{\pi}}$. Assuming that the true action value function $T^{\pi}Q(s,a)$ belongs to the Hilbert space $\mathcal{H}$ generated by the kernel κ, we also need to collect enough data in order to build a Bellman kernel $\tilde{\kappa}^{\pi}$ able to capture the dynamics of the cMDP. In this way solving the regression problem with BRM_{ε} in the limit $\varepsilon \to 0$ is equivalent to solve the Bellman equation of the cMDP. The connection between the empirical kernels can be found using the linear operator $\hat{\mathcal{D}}^{\pi}$ as

$$\begin{aligned} \tilde{\kappa}^{\pi}(s_t,a_t,s'_t,s,a,s') &= \langle \hat{\Psi}^{\pi}(s_t,a_t,s'_t), \hat{\Psi}^{\pi}(s,a,s') \rangle = \langle \hat{\mathcal{D}}^{\pi} \Phi(s_t,a_t), \hat{\mathcal{D}}^{\pi} \Phi(s,a) \rangle \\ \langle \Phi(s_t,a_t), (\hat{\mathcal{D}}^{\pi})^T \hat{\mathcal{D}}^{\pi}\, \Phi(s,a) \rangle &\leq \hat{\lambda}_1 \langle \Phi(s_t,a_t), \Phi(s,a) \rangle = \hat{\lambda}_1 \kappa(s_t,a_t,s,a) \end{aligned} \tag{3.39}$$

with $\hat{\lambda}_1 \leq 1$ max eigenvalue of $(\hat{\mathcal{D}}^{\pi})^T \hat{\mathcal{D}}^{\pi}$. Plugging the operator $\hat{\mathcal{D}}^{\pi}$ in the above expression we have

$$\begin{aligned} \tilde{\kappa}^{\pi}(s_t,a_t,s'_t,s,a,s') &= \langle \hat{\Psi}^{\pi}(s_t,a_t,s'_t), \hat{\Psi}^{\pi}(s,a,s') \rangle \\ &= \langle \Phi(s_t,a_t) - \gamma \textstyle\sum_{a'_t \in A} \pi(a'_t|s'_t) \Phi(s'_t,a'_t), \\ &\quad \Phi(s,a) - \gamma \textstyle\sum_{a' \in A} \pi(a'|s') \Phi(s',a') \rangle \\ &= \kappa(s_t,a_t,s,a) - \gamma \textstyle\sum_{a'_t \in A} \pi(a'_t|s'_t) \kappa(s'_t,a'_t,s,a) \\ &\quad - \gamma \textstyle\sum_{a' \in A} \pi(a'|s') \kappa(s',a'), s_t, a_t) \\ &\quad + \gamma^2 \textstyle\sum_{a' \in A} \pi(a'|s') \sum_{a'_t \in A} \pi(a'_t|s'_t) \kappa(s',a',s'_t,a'_t) \end{aligned} \tag{3.40}$$

Defining the auxiliary functions

$$\hat{\kappa}^{\pi}(s_t,a_t,s'_t,s,a) = \kappa(s_t,a_t,s,a) - \gamma \sum_{a'_t \in A} \pi(a'_t|s'_t)\kappa(s'_t,a'_t,s,a) \tag{3.41}$$

we may also write

$$\tilde{\kappa}^{\pi}(s_t,a_t,s'_t,s,a,s'_t) = \hat{\kappa}^{\pi}(s_t,a_t,s'_t,s,a) - \gamma \sum_{a' \in A} \pi(a'|s')\hat{\kappa}^{\pi}(s',a',s_t,a_t,s'_t) \tag{3.42}$$

3.9 $API - BRM_{\varepsilon}$ Dual Batch Solution

The weighting vector $\mathbf{w}$ can be found minimizing the BR $|BR(s_t,a_t)| \leq \varepsilon$ and assuming $Q(s,a) = \langle \Phi(s,a), \mathbf{w} \rangle + b$ with $Q \in \mathcal{H}$ formulating a regularized regression problem using an SVR as

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\xi^*} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2_{\mathcal{H}} + C\sum_{t=1}^{n}(\xi_t + \xi_t^*) \\ s.t. \quad & r(s_t,a_t) - \langle \Phi(s_t,a_t), \mathbf{w} \rangle + \gamma \int \mathcal{P}(ds'|s_t,a_t) \cdot \sum_{a' \in A} \pi(a'|s') \langle \Phi(s_t,a_t), \mathbf{w} \rangle - (1-\gamma)b \leq \varepsilon + \xi_t \\ & -r(s_t,a_t) + \langle \Phi^{\pi}(s_t,a_t), \mathbf{w} \rangle - \gamma \int \mathcal{P}(ds'|s_t,a_t) \cdot \sum_{a' \in A} \pi(a'|s') \langle \Phi(s_t,a_t), \mathbf{w} \rangle + (1-\gamma)b \leq \varepsilon + \xi_t^* \\ & \xi_t, \xi_t^* \geq 0 \quad t = 1,...,n \end{aligned} \tag{3.43}$$

Using the Bellman kernel we way also find the same weighting vector $\mathbf{w}$ using the features mapping $\Psi^{\pi}(s,a)$ and searching for a solution of the regression function $D^{\pi}Q(s,a) = \langle \Psi^{\pi}(s,a), \mathbf{w} \rangle + b$ with $D^{\pi}Q \in \mathcal{H}_{\Psi^{\pi}}$ and solving the SVR problem

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\xi^*} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2_{\mathcal{H}_{\Psi^{\pi}}} + C\sum_{t=1}^{n}(\xi_t + \xi_t^*) \\ s.t. \quad & r(s_t,a_t) - \langle \Psi^{\pi}(s_t,a_t), \mathbf{w} \rangle - (1-\gamma)b \leq \varepsilon + \xi_t \\ & -r(s_t,a_t) + \langle \Psi^{\pi}(s_t,a_t), \mathbf{w} \rangle + (1-\gamma)b \leq \varepsilon + \xi_t^* \\ & \xi_t, \xi_t^* \geq 0 \quad t = 1,...,n \end{aligned} \tag{3.44}$$

Once the Bellman kernel $\tilde{\kappa}^{\pi}(s_t,a_t,s,a)$ and the rewards $r(s_t,a_t)$ are provided can be solved in principle using any standard SVM package. The main difference within the two formulations depends on how we express the regularizing term where $\|\mathbf{w}\|^2_{\mathcal{H}} = \|\mathbf{w}\|^2_{\mathcal{H}_{\Psi^{\pi}}}$. From a practical point of view we may consider the Hilbert space $\mathcal{H}_{\Psi^{\pi}}$ richer as it contains not only the function capacity of $\mathcal{H}$ but also the dynamics of the MDP and the policy π.

Now we may solve the SVR in the dual using standard Lagrange multipliers technique based on the formulation in section 3.9 which is more compact but whenever necessary we may exploit the equivalence of the two formulations through the connection between the Hilbert spaces $\mathcal{H}$ and $\mathcal{H}_{\Psi^{\pi}}$. Solving the problem in section 3.9 requires introducing the multipliers $\alpha,\alpha^*,\eta,\eta^*$ so we can write the corresponding Lagrangian as:

$$\begin{aligned} \mathscr{L}_{QP} = & \frac{1}{2}\|\mathbf{w}\|^2_{\mathcal{H}_{\Psi^{\pi}}} + C\sum_{t=1}^{n}(\xi_t + \xi_t^*) - \sum_{t=1}^{n}(\eta_t\xi_t + \eta_t^*\xi_t^*) \\ & - \sum_{t=1}^{n}\alpha_t[\varepsilon + \xi_t - r(s_t,a_t) + \langle \Psi^{\pi}(s_t,a_t), \mathbf{w} \rangle + (1-\gamma)b] \\ & - \sum_{t=1}^{n}\alpha_t^*[\varepsilon + \xi_t^* + r(s_t,a_t) - \langle \Psi^{\pi}(s_t,a_t), \mathbf{w} \rangle - (1-\gamma)b] \end{aligned} \tag{3.45}$$

the dual variables have to satisfy positive constraints

$$\alpha_t, \alpha_t^*, \eta_t, \eta_t^* \geq 0. \tag{3.46}$$

It follows from the saddle point conditions that the partial derivatives of $\mathscr{L}_{QP}$ with respect to the primal variables $(\mathbf{w}, b, \xi, \xi^*)$ have to vanish for optimality:

$$\frac{\partial \mathscr{L}_{QP}}{\partial \mathbf{w}} = 0 \quad \longrightarrow \quad \mathbf{w} = \sum_{t=1}^{n} (\alpha_t - \alpha_t^*) \Psi^{\pi}(s_t, a_t) \tag{3.47}$$

$$\frac{\partial \mathscr{L}_{QP}}{\partial b} = 0 \quad \longrightarrow \quad \sum_{t=1}^{n} (\alpha_t - \alpha_t^*) = 0 \tag{3.48}$$

$$\frac{\partial \mathscr{L}_{QP}}{\partial \xi_t} = 0 \quad \longrightarrow \quad C - \alpha_t - \eta_t = 0 \tag{3.49}$$

$$\frac{\partial \mathscr{L}_{QP}}{\partial \xi_t^*} = 0 \quad \longrightarrow \quad C - \alpha_t^* - \eta_t^* = 0 \tag{3.50}$$

$$\tag{3.51}$$

Substituting section 3.9,section 3.9,section 3.9,section 3.9 into section 3.9 and eliminating the dual variables $\eta_t = C - \alpha_t$ and $\eta_t^* = C - \alpha_t^*$ yields to the dual optimization problem

$$\begin{aligned} \min_{\alpha, \alpha^*, b} \quad & \tfrac{1}{2} \textstyle\sum_{t=1}^{n} \sum_{p=1}^{n} K_{tp}^{\pi} (\alpha_t - \alpha_t^*)(\alpha_p - \alpha_p^*) + \varepsilon \sum_{t=1}^{n} (\alpha_t + \alpha_t^*) \\ & - \textstyle\sum_{t=1}^{n} r(s_t, a_t)(\alpha_t - \alpha_t^*) + b(1-\gamma) \sum_{t=1}^{n} (\alpha_t - \alpha_t^*) \end{aligned} \tag{3.52}$$

$$s.t. \qquad \textstyle\sum_{t=1}^{n} (\alpha_t - \alpha_t^*) = 0 \quad \alpha_t, \alpha_t^* \in [0, C] \quad \forall t = 1, ..., n$$

where $K_{tp}^{\pi} = \tilde{\kappa}^{\pi}(s_t, a_t, s_p, a_p) = (\Psi_t^{\pi})^T \Psi_p^{\pi}$ is the Bellman Error kernel matrix where

$$(\Psi^{\pi})^T = [\Psi^{\pi}(s_1, a_1), \ldots \Psi^{\pi}(s_n, a_n)]^T.$$

The problem in section 3.9 can be written in more compact form as

$$\min_{\alpha, \alpha^*, b} \quad \tfrac{1}{2} (\alpha - \alpha^*)^T K^{\pi} (\alpha - \alpha^*) + \varepsilon(\alpha + \alpha^*) - R^T (\alpha - \alpha^*) + b(1-\gamma)(\alpha - \alpha^*) \tag{3.53}$$

$$s.t. \qquad I_n^T (\alpha - \alpha^*) = 0 \quad \alpha_t, \alpha_t^* \in [0, C] \quad \forall t = 1, ..., n$$

with $R = [r(s_1, a_1) \ldots r(s_n, a_n)]^T$ and I_n the identity matrix with n elements or even in more compact form defining $\beta = [\alpha \quad \alpha^*]$ as

$$\min_{\beta, b} \quad \tfrac{1}{2} \beta^T Q_K^{\pi} \beta + (\mathbf{c}^T + b\mathbf{d}^T)\beta \tag{3.54}$$

$$s.t. \qquad [I_n \quad -I_n]^T \beta = 0 \quad \beta_t \in [-C, C] \quad \forall t = 1, ..., n$$

where we put $\mathbf{d}^T = [(1-\gamma) I_n \quad -(1-\gamma) I_n]$ and $\mathbf{c}^T = [\varepsilon I_n + R \quad \varepsilon I_n - R]$ and the kernel matrix

$$Q_K^{\pi} = \begin{bmatrix} K^{\pi} & -K^{\pi} \\ -K^{\pi} & K^{\pi} \end{bmatrix}. \tag{3.55}$$

Moreover the Karush-Kuhn-Tucker (KKT) conditions require that at the point of the solution the product between the dual variables and constraints has to vanish:

$$\begin{aligned} \alpha_t[\varepsilon+\xi_t - r(s_t,a_t) + \langle \Psi^\pi(s_t,a_t),\mathbf{w}\rangle + (1-\gamma)b] &= 0 \\ \alpha_t^*[\varepsilon+\xi_t^* + r(s_t,a_t) - \langle \Psi^\pi(s_t,a_t),\mathbf{w}\rangle - (1-\gamma)b] &= 0 \\ (C-\alpha_t)\xi_t &= 0 \\ (C-\alpha_t^*)\xi_t^* &= 0 \end{aligned} \tag{3.56}$$

As a result only samples (s_t,a_t) with corresponding $\alpha_t = C$ or $\alpha_t^* = C$ lie outside the $\varepsilon-$insensitive tube. Secondly $\alpha_t\alpha_t^* = 0$ i.e. there can never be a set of dual variables α_t, α_t^* simultaneously non zero. The same kind of argument exposed for the SVR can be applied to the calculation of the b parameter. Once the dual variables are known defining $\beta_t = \alpha_t - \alpha_t^*$ the weight vector is given by

$$\mathbf{w} = \sum_{t=1}^{n} \beta_t \Psi^\pi(s_t,a_t)$$

Hence, the dual problem has Lagrangian

$$\mathscr{L}_{PP} = \frac{1}{2}\sum_{t=1}^{n}\sum_{p=1}^{n} K_{tp}^\pi \beta_t\beta_p + \varepsilon\sum_{t=1}^{n}(\alpha_t+\alpha_t^*) - \sum_{t=1}^{n}(\alpha_t-\alpha_t^*)r(s_t,a_t) \tag{3.57}$$
$$+\hat{b}(1-\gamma)\sum_{t=1}^{n}(\alpha_t-\alpha_t^*) - \sum_{t=1}^{n}(\delta_t\alpha_t + \delta_t^*\alpha_r^*) + \sum_{t=1}^{n}[\upsilon_t(\alpha_t - C) + \upsilon_t^*(\alpha_t^* - C)]$$

where $\delta_t, \delta_t^*, \upsilon_t, \upsilon_t^*$ and $\hat{b}$ are the Lagrange multipliers. Optimizing this Lagrangian leads to the following KKT conditions:

$$\frac{\partial\, \mathscr{L}_{PP}}{\partial\, \alpha_t} = 0 \rightarrow \quad \sum_{p=1}^{n} K_{tp}^\pi\beta_p + \varepsilon - r(s_t,a_t) + \hat{b}(1-\gamma) - \delta_t + \upsilon_t = 0 \tag{3.58}$$

$$\frac{\partial\, \mathscr{L}_{PP}}{\partial\, \alpha_t^*} = 0 \rightarrow -\sum_{p=1}^{t} K_{tp}^\pi\beta_p^* + \varepsilon + r(s_t,a_t) - \hat{b}(1-\gamma) - \delta_t^* + \upsilon_t^* = 0 \tag{3.59}$$

$$\delta_t \geq 0 \quad \delta_t\alpha_t = 0 \quad \delta_t^* \geq 0 \quad \delta_t^*\alpha_t^* = 0 \tag{3.60}$$

$$\upsilon_t \geq 0 \quad \upsilon_t(\alpha_t - C) = 0 \quad \upsilon_t^* \geq 0 \quad \upsilon_t^*(\alpha_t^* - C) = 0. \tag{3.61}$$

Note that b is equal to $\hat{b}$ at optimality and defining $(g_t^* = -g_t + 2\varepsilon)$

$$g_t = \sum_{p=1}^{n} K_{tp}^\pi\beta_p + b(1-\gamma) - r(s_t,a_t) + \varepsilon$$

. As the $\alpha_t\alpha_t^* = 0$ condition holds the β_t parameter completely determine both α_t and α_t^*. Recalling that from KKT we have $\mathbf{w} = \sum_{t=1}^{n}\beta_t\Psi^\pi(s_t,a_t)$ the margin function may be expressed as

$$\begin{aligned} h(s_t,a_t) &= \langle \Psi^\pi(s_t,a_t),\mathbf{w}\rangle + b(1-\gamma) - r(s_t,a_t) \\ &= \sum_{p=1}^{n}\beta_p\langle \Psi^\pi(s_t,a_t), \Psi^\pi(s_p,a_p)\rangle + b(1-\gamma) - r(s_t,a_t) \\ &= \sum_{p=1}^{n} K_{tp}^\pi\beta_p + b(1-\gamma) - r(s_t,a_t) \end{aligned} \tag{3.62}$$

It is also worth noting that the following conditions descend from the KKT:

$$\begin{aligned} \delta_t &= h(s_t,a_t)+\varepsilon+\upsilon_t = g_t+\upsilon_t \quad \alpha_t[h(s_t,a_t)+\varepsilon+\upsilon_t]=0 \\ \delta_t^* &= -h(s_t,a_t)+\varepsilon+\upsilon_t^* = g_t^*+\upsilon_t^* \quad \alpha_t^*[-h(s_t,a_t)+\varepsilon+\upsilon_t^*]=0 \end{aligned} \tag{3.63}$$

and combining $h(s_t,a_t)$ definition with the KKT conditions we obtain:

$$\begin{cases} h(s_t,a_t) \geq \varepsilon \;\; \beta_t = -C,\, \alpha_t = 0,\, \alpha_t^* = C \\ h(s_t,a_t) = \varepsilon \;\; -C < \beta_t < 0,\, \alpha_t = 0,\, 0 < \alpha_t^* < C \\ -\varepsilon \leq h(s_t,a_t) \leq \varepsilon \;\; \beta_t = 0,\, \alpha_t = 0,\, \alpha_t^* = 0 \\ h(s_t,a_t) = -\varepsilon \;\; 0 < \beta_t < C,\, 0 < \alpha_t < C,\, \alpha_t^* = 0 \\ h(s_t,a_t) \leq -\varepsilon \;\; \beta_t = C,\, \alpha_t = C,\, \alpha_t^* = 0 \end{cases} \tag{3.64}$$

Hence, the conditions in section 3.9 allows for the identification of three subsets of training set and can be classified as:

- the set E Error support vectors $E = \{t\ s.t.\ |\,|\beta_t| = C\}$
- the set S Margin support vectors $S = \{t\ s.t.\ |\,0 < |\beta_t| < C\}$
- the set R Remaining samples $R = \{t\ s.t.\ |\,|\beta_t| = 0\}$

3.10 Bellman Kernel Characterization

Finally we may investigate the connection of the SVR solution between the Hilbert spaces $\mathcal{H}$ and $\mathcal{H}_{\Psi^\pi}$ for the infinite sample formulation. Consider the weighting vector $\mathbf{w} = \sum_{t=1}^n \beta_t \Psi^\pi(s_t,a_t)$ if we plug

$$\Psi^\pi(s_t,a_t) = \Phi(s_t,a_t) - \gamma \int \mathcal{P}(s'|s_t,a_t) \sum_{a' \in A} \pi(s'|a')\Phi(s',a')$$

may write

$$\begin{aligned} \mathbf{w} &= \textstyle\sum_{t=1}^n \beta_t [\Phi(s_t,a_t) - \gamma \int \mathcal{P}(s'|s_t,a_t) \sum_{a' \in A} \pi(s'|a')\Phi(s',a')] \\ &= \textstyle\sum_{t=1}^n \beta_t \Phi(s_t,a_t) - \gamma \sum_{t=1}^n \beta_t \int \mathcal{P}(s'|s_t,a_t) \sum_{a' \in A} \pi(s'|a')\Phi(s',a') \end{aligned} \tag{3.65}$$

giving the weighting vector $\mathbf{w}$ in terms of a linear combination of vectors in the span of $\Phi(s,a)$. According to the expression section 3.10, while in the Hilbert space $\mathcal{H}_{\Psi^\pi}$ the KKT conditions tells us that the vector $\mathbf{w}$ is a linear combination of $\{\Psi^\pi(s_t,a_t) : t = 1,..,n\}$. Hence in general in the Hilbert space $\mathcal{H}$ the whole set of functions $\Phi(s_t,a_t)$ may be necessary to represent the same vector $\mathbf{w}$.

Now to avoid clutter with notation we move to the empirical version of $API - BRM_\varepsilon$ using the finite sample formulation

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\xi^*} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2_{\mathcal{H}_{\hat{\Psi}^\pi}} + C\textstyle\sum_{t=1}^n (\xi_t + \xi_t^*) \\ s.t. \quad & \hat{r}_t - \langle \hat{\Psi}^\pi(s_t,a_t,s_t'), \mathbf{w} \rangle - (1-\gamma)b \leq \varepsilon + \xi_t \\ & -\hat{r}_t + \langle \hat{\Psi}^\pi(s_t,a_t,s_t'), \mathbf{w} \rangle + (1-\gamma)b \leq \varepsilon + \xi_t^* \\ & \xi_t, \xi_t^* \geq 0 \quad t = 1,...,n \end{aligned} \tag{3.66}$$

and the solution of the regression problem may be expressed as

$$\hat{D}^{\pi}\hat{Q}(s,a,s') = \langle \hat{\Psi}(s,a,s'), \mathbf{w} \rangle + (1-\gamma)b = \sum_{t=1}^{n} \beta_t \tilde{\kappa}^{\pi}(s,a,s',s_t,a_t,s'_t) \tag{3.67}$$

where the Hilbert norm can be written as

$$\|\hat{D}^{\pi}\hat{Q}\|^2_{\mathcal{H}_{\hat{\Psi}\pi}} = \sum_{i,j=1}^{n} \beta_i \beta_j \tilde{\kappa}^{\pi}(s_i,a_i,s'_i,s_j,a_j,s'_j) \tag{3.68}$$

and the action value function

$$\begin{aligned} \hat{Q}(s,a) &= \langle \Phi(s,a), \mathbf{w} \rangle + b = \textstyle\sum_{t=1}^{n} \beta_t \hat{\kappa}^{\pi}(s,a,s_t,a_t,s'_t) \\ &= \textstyle\sum_{t=1}^{n} \beta_t [\kappa(s,a,s_t,a_t) - \gamma \sum_{a'_t \in A} \pi(a'_t|s'_t)\kappa(s,a,s'_t,a'_t)] \end{aligned} \tag{3.69}$$

So defining

$$\begin{gathered} (s_i,a_i) = (s_t,a_t) \quad and \quad \alpha_i = \beta_t \quad if \quad j = 0 \\ (s_i,a_i) = (s'_t,a'_j) \quad and \quad \alpha_i = -\gamma\beta_t\pi(a'_j|s'_t) \quad if \quad j > 0 \\ i = (t-1)(1+|A|) + j + 1 \quad t = 1,..,n \quad j = 0,...,|A| \end{gathered} \tag{3.70}$$

we may express the weighting vector as

$$\mathbf{w} = \sum_{i=1}^{(1+|A|)n} \alpha_i \Phi(s_i,a_i). \tag{3.71}$$

section 3.10 tells that the weighting vector $\mathbf{w}$ is the result of a linear combination of $(1+|A|)n$ vectors $\Phi(s_i,a_i) \in \mathcal{H}$. This essentially depends on the choice to keep the average over the policy (which reduces the variance on this term) and might be reduced to $2n$ vectors using the max policy (as in Q-learning) or one of the possible actions (as in SARSA). The action value function is expanded as

$$\hat{Q}(s,a) = \langle \Phi(s,a), \mathbf{w} \rangle + b = \sum_{i=1}^{(1+|A|)n} \alpha_i \kappa(s,a,s_i,a_i) \tag{3.72}$$

and the Hilbert norm

$$\|\hat{Q}\|^2_{\mathcal{H}} = \sum_{i,j=1}^{(1+|A|)n} \alpha_i \alpha_j \kappa(s_i,a_i,s_j,a_j) \tag{3.73}$$

with $\|\hat{D}^{\pi}\hat{Q}\|^2_{\mathcal{H}_{\Psi\pi}} = \|\hat{Q}\|^2_{\mathcal{H}}$.

3.11 Incremental Equivalent To Batch $API - BRM_{\varepsilon}$

SVR can be also solved very efficiently using an incremental algorithm (see [70], [49] for SVM and [57] for the extension to SVR) which updates the trained SVR function whenever a new sample z_c is added to the training set D_n. The basic idea is to change the coefficient β_c corresponding to the new sample z_c in a finite number of discrete steps until it meets the KKT conditions while ensuring that the existing samples in D_n continue to satisfy the KKT conditions at each step. Moreover to build an exact incremental SVR one needs to define three primitive actions:

- add a new vector $D' = D \cup \{z_c\}$
- remove an existing vector $D' = D \setminus \{z_c\}$
- update an existing vector $D' = D \setminus \{z_c\} \cup \{z'_c\}$

In each case the resulting incremental SVR should be the same that would be training from the scratch using the whole final set of data as done in batch mode.

Now to transform batch $API - BRM_\varepsilon$ formulation into an incremental version we have to arrange the primal problem in section 3.9 or equivalently the dual problem in section 3.9. This has to be done considering that in batch mode the data samples have been collected at some time while interacting with the cMDP. In the incremental version data sample are generated one step t at the time using some behavior policy. Assume now that into the data set

$$D_n = \{(s_1, a_1, r_1, s'_1), ..., (s_n, a_n, r_n, s'_n)\} = \{(s_t, a_t, r_t, s'_t) \quad t = 1, ..., n\}$$

any given point $(\bar{s}_i, \bar{a}_i)$ may be repeated a number n_i of times such as $n = \sum_{i=1}^{N} n_i$ where N is number of actually different samples. For the sake of the argument imagine we arranged the repeated samples as

$$\begin{aligned} s_1 = ... = s_{n_1} = \bar{s}_1 \quad & and \quad a_1 = ... = a_{n_1} = \bar{a}_1 \\ s_{n_1+1} = ... = s_{n_1+n_2} = \bar{s}_2 \quad & and \quad a_{n_1+1} = ... = a_{n_1+n_2} = \bar{a}_2 \\ & ..... \\ s_{n_{N-1}+1} = ... = s_{n_N} = \bar{s}_N \quad & and \quad a_{n_{N-1}+1} = ... = a_{n_N} = \bar{a}_N \end{aligned}$$

Now for each $(\bar{s}_i, \bar{a}_i)$ the total number of transitions $n_i = \sum_{j=1}^{n_i} n_{ij}$ are also present in D_n

$$\{\vec{s}'_{ij} = s'_t \quad \bar{r}_{ij} = r_t \quad t = (i-1)n_i + 1, ..., (i-1)n_i + n_{i+1}, \quad i = 1, ...N, \; j = 1, ..., n_i\}$$

In practice we are considering D_n organized in repeated experiences as

$$D_n = \{(\bar{s}_i, \bar{a}_i, \bar{r}_{ij}, \vec{s}'_{ij}) \quad i = 1..., N, \quad j = 1, ..., n_i, \quad n_i = \sum_{j=1}^{n_i} n_{ij}, \quad n = \sum_{i=1}^{N} n_i\}$$

which means N different points $(\bar{s}_i, \bar{a}_i)$ repeated n_i times with n_{ij} different transactions $(\bar{r}_{ij}, \vec{s}'_{ij})$. If D_n is large enough we may build an unbiased estimation of the transition probability as $P^{\vec{s}'_{ij}}_{\bar{s}_i, \bar{a}_i} \approx \frac{n_{ij}}{n_i}$. Hence the feature mapping $\Psi^{\pi_i}(\bar{s}_i, \bar{a}_i)$ defined for the batch $API - BRM_\varepsilon$ may be written as

$$\bar{\Psi}^{\pi}(\bar{s}_i, \bar{a}_i) \approx \sum_{\vec{s}'_{ij}} P^{\vec{s}'_{ij}}_{\bar{s}_i, \bar{a}_i} \hat{\Psi}^{\pi_{ij}}(\bar{s}_i, \bar{a}_i, \vec{s}'_{ij}) = \frac{1}{n_i} \sum_{j=1}^{n_i} n_{ij} \hat{\Psi}^{\pi_{ij}}(\bar{s}_i, \bar{a}_i, \vec{s}'_{ij}) \tag{3.74}$$

where with $\pi_{ij}(a' | \vec{s}'_{ij})$ we indicated the probability for selecting a given action in the state $\vec{s}'_{ij}$ which has to be thought as a stochastic matrix. Now considering dual formulation of the batch $API - BRM_\varepsilon$ as

$$\min_{\bar{\beta}} \qquad \frac{1}{2} \bar{\beta}^T Q_K^{\pi_t} \bar{\beta} + (\mathbf{c}^T + b\mathbf{d}^T) \bar{\beta} \tag{3.75}$$

$$s.t. \qquad [I_N \quad -I_N]^T \bar{\beta} = 0 \quad \bar{\beta}_t \in [-C, C] \quad \forall (\bar{s}_t, \bar{a}_t) \in S \times A$$

where we put $\mathbf{d}^T = [(1-\gamma)I_N \quad -(1-\gamma)I_N]$ and $\mathbf{c}^T = [\varepsilon I_N + \bar{R} \quad \varepsilon I_N - \bar{R}]$ with

$$\bar{R} = [\bar{r}(\bar{s}_1,\bar{a}_1)\dots\bar{r}(\bar{s}_N,\bar{a}_N)]^T = [\sum_{j=1}^{n_1}\frac{n_{1j}}{n_1}\bar{r}_{1j}\dots\sum_{j=1}^{n_N}\frac{n_{Nj}}{n_N}\bar{r}_{Nj}]^T$$

and $\bar{r}_{ij} = r^{\bar{s}'_{ij}}_{\bar{s}_i,\bar{a}_i}$ and the kernel matrix

$$\bar{Q}^{\pi}_K = \begin{bmatrix} \bar{K}^{\pi} & -\bar{K}^{\pi} \\ -\bar{K}^{\pi} & \bar{K}^{\pi} \end{bmatrix} = \begin{bmatrix} \bar{\Psi}^{\pi} \\ -\bar{\Psi}^{\pi} \end{bmatrix} \begin{bmatrix} (\bar{\Psi}^{\pi})^T & (-\bar{\Psi}^{\pi})^T \end{bmatrix} \tag{3.76}$$

The features mapping are used to build the matrix $\bar{K}^{\pi} = (\bar{\Psi}^{\pi})^T\bar{\Psi}^{\pi}$ where

$$\bar{\Psi}^{\pi} = [\bar{\Psi}^{\pi}(\bar{s}_1,\bar{a}_1)\dots\bar{\Psi}^{\pi}(\bar{s}_N,\bar{a}_N)]^T = [\sum_{j=1}^{n_1}\frac{n_{1j}}{n_1}\hat{\Psi}^{\pi_{1j}}(\bar{s}_1,\bar{a}_1,\bar{s}'_{1j})\dots\sum_{j=1}^{n_N}\frac{n_{Nj}}{n_N}\hat{\Psi}^{\pi_{Nj}}(\bar{s}_N,\bar{a}_N,\bar{s}'_{Nj})]^T$$

Hence, the matrix product can be decomposed as

$$\bar{\beta}^T\begin{bmatrix}\bar{\Psi}^{\pi} \\ -\bar{\Psi}^{\pi}\end{bmatrix} = \begin{bmatrix}\bar{\alpha} & \bar{\alpha}^*\end{bmatrix}\begin{bmatrix}\Psi^{\pi} \\ -\Psi^{\pi}\end{bmatrix} = \begin{bmatrix}\bar{\alpha}_1 \dots \bar{\alpha}_N & \bar{\alpha}^*_1 \dots \bar{\alpha}^*_N\end{bmatrix}\begin{bmatrix}\bar{\Psi}^{\pi}(\bar{s}_1,\bar{a}_1) & \dots & \Psi^{\pi}(\bar{s}_N,\bar{a}_N) \\ -\bar{\Psi}^{\pi}(\bar{s}_1,\bar{a}_1) & \dots & -\Psi^{\pi}(\bar{s}_N,\bar{a}_N)\end{bmatrix} \tag{3.77}$$

and for each product

$$\bar{\alpha}_i\bar{\Psi}^{\pi}(\bar{s}_i,\bar{a}_i) = \sum_{j=1}^{n_i}\bar{\alpha}_i\frac{n_{ij}}{n_i}\hat{\Psi}^{\pi_{ij}}(\bar{s}_i,\bar{a}_i,\bar{s}'_{ij}) = \sum_{j=1}^{n_i}n_{ij}\hat{\alpha}_{ij}\hat{\Psi}^{\pi_{ij}}(\bar{s}_i,\bar{a}_i,\bar{s}'_{ij}) \tag{3.78}$$

where we put $\hat{\alpha}_{ij} = \frac{\bar{\alpha}_i}{n_i}$ (and $\hat{\alpha}^*_{ij} = \frac{\bar{\alpha}^*_i}{n_i}$) which suggests that the same solution of the optimization problem may be obtained by a particular kind of SVR where we control that the same Lagrangian multipliers have been used for any repeated sample $(\bar{s}_i,\bar{a}_i,\bar{r}_{ij},\bar{s}'_{ij})$. This can be easily understood considering that the set of events having the same $(\bar{s}_i,\bar{a}_i)$ but different $(\bar{r}_{ij},\bar{s}'_{ij})$, from the point of view of the feature mapping $\bar{\Psi}^{\pi}(\bar{s}_i,\bar{a}_i)$ are linearly dependent and as a consequence must share the same lagrangian multipliers $(\bar{\alpha}_i,\bar{\alpha}^*_i)$.

$$\begin{aligned}
\min_{\mathbf{w},b,\bar{\xi},\bar{\xi}^*} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2_{\mathcal{H}_{\Psi^{\pi}}} + C\textstyle\sum_{i=1}^{N}\sum_{j=1}^{n_i}n_{ij}(\bar{\xi}_{ij}+\bar{\xi}^*_{ij}) \qquad (3.79)\\
s.t. \quad & \bar{r}_{ij} - \langle\hat{\Psi}^{\pi_{ij}}(\bar{s}_i,\bar{a}_i,\bar{s}'_{ij}),\mathbf{w}\rangle - (1-\gamma)b \le \varepsilon + \bar{\xi}_{ij} \quad n_{ij} \quad times\\
& -\bar{r}_{ij} + \langle\hat{\Psi}^{\pi_{ij}}(\bar{s}_i,\bar{a}_i,\bar{s}'_{ij}),\mathbf{w}\rangle + (1-\gamma)b \le \varepsilon + \bar{\xi}^*_{ij} \quad n_{ij} \quad times\\
& \bar{\xi}_{ij},\bar{\xi}^*_{ij} \ge 0 \quad i = 1,\dots,N, \quad j = 1,..,n_i
\end{aligned}$$

when a new point is added to the SVR we must check if it is already present in order to apply the constraints on the $\hat{\alpha}_{ij}$. In the empirical case this check will be only necessary for discrete state domains, while in the general case of continuous state domain it is quite unlikely that the MDP will make transitions on exactly the same point. Moreover it is not necessary to store the probability transition matrix $P^{\bar{s}'_{ij}}_{\bar{s}_i,\bar{a}_i} = \bar{P}^{ij}_i$ as the incremental SVR algorithm implicitly takes this task into account.

Analysis of the properties of the incremental solution for problem section 3.11 follows the methodology reported in [57] while some technical details can also found in Appendix 3.12.

A possible generalization of $API - BRM_{\varepsilon}$ using different loss functions are presented also in Appendix 3.13 together with a methodology involving the solution of the approximation problem in the primal.

3.12 Appendix 3A: $API - BRM_\varepsilon$ Incremental Solution

Consider the formulation of the incremental $API - BRM_\varepsilon$ reported in problem section 3.11. Looking for the solution we may proceed as in the batch case, the weight vector $\mathbf{w}$ of the problem in section 3.11 can be expanded as:

$$\mathbf{w} = \sum_{k=1}^{N} \sum_{h=1}^{n_k} \hat{\beta}_{kh} \bar{P}_k^{kh} \hat{\Psi}^{\pi_{hk}}(\bar{s}_k, \bar{a}_k, \bar{s}'_{hk}) \tag{3.80}$$

and the approximated action value function will be expressed as

$$\begin{aligned} Q(\bar{s}_i, \bar{a}_i) &= \langle \Phi(\bar{s}_i, \bar{a}_i), \mathbf{w} \rangle + b = \textstyle\sum_{k=1}^{N} \sum_{h=1}^{n_k} \hat{\beta}_{kh} \bar{P}_k^{kh} \langle \Phi(\bar{s}_i, \bar{a}_i), \hat{\Psi}^{\pi_{hk}}(\bar{s}_k, \bar{a}_k, \bar{s}'_{hk}) \rangle + b \\ &= \textstyle\sum_{k=1}^{N} \sum_{h=1}^{n_k} \hat{\beta}_{kh} \bar{P}_k^{kh} \hat{K}_{ikh}^{\pi} + b \end{aligned} \tag{3.81}$$

where the auxiliary function $\hat{K}^{\pi}$ is defined as

$$\begin{aligned} \hat{K}_{ikh}^{\pi} &= \langle \Phi(\bar{s}_i, \bar{a}_i), \hat{\Psi}^{\pi_{hk}}(\bar{s}_k, \bar{a}_k, \bar{s}'_{hk}) \rangle = \hat{\kappa}(\bar{s}_i, \bar{a}_i, \bar{s}_k, \bar{a}_k, \bar{s}'_{kh}) \\ &= \kappa(\bar{s}_i, \bar{a}_i, \bar{s}_k, \bar{a}_k) - \gamma \textstyle\sum_{a'} \pi_{kh}(a' | \bar{s}'_{kh}) \kappa(\bar{s}_i, \bar{a}_i, \bar{s}'_{kh}, a') \end{aligned} \tag{3.82}$$

and the regression function

$$\begin{aligned} \hat{D}^{\pi} \hat{Q}_r(\bar{s}_i, \bar{a}_i) &= \textstyle\sum_{j=1}^{n_i} \bar{P}_i^{ij} \hat{Q}_r^{\pi}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij})) = \sum_{j=1}^{n_i} \bar{P}_i^{ij} \langle \hat{\Psi}^{\pi_{ij}}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}), w \rangle + (1-\gamma) b \\ &= \textstyle\sum_{k=1}^{N} \sum_{h=1}^{n_k} \sum_{j=1}^{n_i} \hat{\beta}_{kh} \bar{P}_i^{ij} \bar{P}_k^{kh} \langle \hat{\Psi}^{\pi_{ij}}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}), \hat{\Psi}^{\pi_{hk}}(\bar{s}_k, \bar{a}_k, \bar{s}'_{hk}) \rangle + (1-\gamma) b \\ &= \textstyle\sum_{k=1}^{N} \sum_{h=1}^{n_k} \sum_{j=1}^{n_i} \hat{\beta}_{kh} \bar{P}_i^{ij} \bar{P}_k^{kh} \tilde{K}_{ijkh}^{\pi} + (1-\gamma) b \end{aligned} \tag{3.83}$$

using $\hat{D}^{\pi} \hat{Q}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}) = \langle \hat{\Psi}^{\pi_{ij}}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}), \mathbf{w} \rangle + (1-\gamma) b$ and the Bellman kernel $\tilde{K}^{\pi}$ is defined with elements as

$$\begin{aligned} \tilde{K}_{ijkh}^{\pi} &= \langle \hat{\Psi}^{\pi_{ij}}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}), \hat{\Psi}^{\pi_{hk}}(\bar{s}_k, \bar{a}_k, \bar{s}'_{hk}) \rangle = \tilde{\kappa}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}, \bar{s}_k, \bar{a}_k, \bar{s}'_{kh}) \\ &= \hat{\kappa}(\bar{s}_i, \bar{a}_i, \bar{s}_k, \bar{a}_k) - \gamma \textstyle\sum_{a'} \pi_{ij}(a' | \bar{s}'_{ij}) \hat{\kappa}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}, a') \end{aligned} \tag{3.84}$$

One that aspect we have to take into account regards the policy π_{ij} used to build the feature vector $\hat{\Psi}^{\pi_{ij}}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij})$. In this case the function $\pi_{ij}(\cdot, s'_t)$ remains implicitly stored into the kernel matrix of the machine $\hat{K}^{\pi} = (\hat{\Psi}^{\pi})^T \hat{\Psi}^{\pi}$ where

$$\hat{\Psi}^{\pi} = [\hat{\Psi}^{\pi_{11}}(\bar{s}_1, \bar{a}_1, \bar{s}'_{11}) \ldots \hat{\Psi}^{\pi_{Nn_N}}(\bar{s}_N, \bar{a}_N, \bar{s}'_{Nn_N})]^T$$

which according to the API Algorithm 2 should be evaluated as the greedy policy coming form the previously known action value function approximation as $\pi_t(\cdot) \leftarrow \hat{\pi}(\cdot, \hat{Q}_{t-1})$. This means that whenever we add a new sample into the incremental $API - BRM_\varepsilon$ the history of the greedy policies $\Pi_n = \{\pi_{11}, \ldots, \pi_{Nn_N}\}$ used in any next state transition s'_t is implicitly stored into the machine. Hence the action value function $\hat{Q}_t$ updated into the policy evaluation step takes into account this policy history. Nevertheless the new policy $\hat{\pi}(\cdot, \hat{Q}_t)$ is still the greedy policy of the last approximation of the action value function $\hat{Q}_t$ coming from the $API - BRM_\varepsilon$ problem.

In order to show how the incremental SVR algorithm finds solution of the optimization problem in section 3.11, consider the sample (s_t, a_t) with the initially set value of β_t and

gradually change (increase o decrease) its value preserving KKT conditions. Adding a sample means that for a given state s_t applying the action a_t we reach the state s'_t and getting the reward r_t. Therefore two situations are possible: either the point (s_t, a_t) was visited before and is already present as $(s_t, a_t, r_t, s'_t) = (\bar{s}_i, \bar{a}_i, \bar{r}_{ij}, \bar{s}'_{ij})$ (for a given i and j) with a given set of transitions $(\bar{r}_{ij}, \bar{s}'_{ij}) \quad j = 1, ..., n_i$ or not. In the first case the estimated transition probability is $\bar{P}^{\bar{s}'_{ij}}_{\bar{s}_i,\bar{a}_i} = \bar{P}^{ij}_i = \frac{n_{ij}}{n_i}$ is going to change during the incremental step. In the last one the sample is a completely new we have to start building the transition matrix from scratch. According to the analysis of the batch $API - BRM_\varepsilon$ we may write the margin function for the samples in D_n before the incremental update expressed as

$$h(\bar{s}_i, \bar{a}_i) = \sum_{k=1}^{N} \sum_{h=1}^{n_k} \sum_{j=1}^{n_i} \hat{\beta}_{hk} \bar{P}^{ij}_i \bar{P}^{kh}_k \langle \Psi^{\pi_{ij}}(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}), \hat{\Psi}^{\pi_{kh}}(\bar{s}_k, \bar{a}_k, \bar{s}'_{kh}) \rangle +$$
$$b(1-\gamma) - \bar{r}(\bar{s}_i, \bar{a}_i) = \sum_{k=1}^{N} \sum_{h=1}^{n_k} \hat{\beta}_{hk} \bar{P}^{ij}_i \bar{P}^{kh}_k \tilde{K}^{\pi}_{ijkh} + b(1-\gamma) - \bar{r}(\bar{s}_i, \bar{a}_i) \tag{3.85}$$

Hence, after adding a completely new point $(s_c, a_c, r_c, s'_c) = (\bar{s}_c, \bar{a}_c, \bar{r}_{cq}, \bar{s}_{cq})$ according to the KKT the margin function becomes

$$h'(\bar{s}_i, \bar{a}_i) = \sum_{k=1}^{N} \sum_{j=1}^{n_i} \sum_{h=1}^{n_k} \hat{\beta}'_{hk} \bar{P}^{ij}_i \bar{P}^{kh}_k \tilde{K}^{\pi}_{ijkh} + \hat{\beta}'_{cq} \sum_{j=1}^{n_i} \bar{P}^{ij}_i \tilde{K}^{\pi}_{ijcq} + b(1-\gamma) - \bar{r}(\bar{s}_k, \bar{a}_k) \tag{3.86}$$

so the changes in the margin function during the update of a new training point according to the KKT as

$$\Delta h(s_i, a_i) = \sum_{k=1}^{N} \sum_{h=1}^{n_k} \sum_{j=1}^{n_i} \Delta\hat{\beta}_{hk} \bar{P}^{kh}_k \bar{P}^{ij}_i \hat{K}^{\pi}_{ikh} + \Delta\hat{\beta}_{cq} \sum_{j=1}^{n_i} \bar{P}^{ij}_i \hat{K}^{\pi}_{ijcq} + + \Delta b(1-\gamma) \tag{3.87}$$

In case we are updating a point already present the transition probability $\bar{P}^{ch}_c$ will also change to $\bar{P}'^{ch}_c$ for $h = 1, ..., n_c$ and we have

$$h'(s_i, a_i) = \sum_{\substack{k=1 \\ k \neq c}}^{N} \sum_{h=1}^{n_k} \sum_{j=1}^{n_i} \hat{\beta}'_{hk} \bar{P}'^{kh}_k \bar{P}'^{ij}_i \tilde{K}^{\pi}_{ijkh} + \hat{\beta}'_{cq} \sum_{\substack{h=1 \\ h \neq q}}^{n_c} \sum_{j=1}^{n_i} \bar{P}'^{ch}_c \bar{P}'^{ij}_i \tilde{K}^{\pi}_{ijch}$$
$$+ \hat{\beta}'_{cq} \sum_{j=1}^{n_i} \bar{P}'^{cq}_c \bar{P}'^{ij}_i \tilde{K}^{\pi}_{ijcq} + b(1-\gamma) - \bar{r}(\bar{s}_i, \bar{a}_i)$$

As a result we may evaluate the changes in the margin function during the update of a new training point according to the KKT as

$$\Delta h(s_i, a_i) = \sum_{k=1}^{N} \sum_{h=1}^{n_k} \sum_{j=1}^{n_i} \Delta\hat{\beta}_{hk} \bar{P}^{kh}_k \bar{P}'^{ij}_i \hat{K}^{\pi}_{ikh} + \Delta\hat{\beta}_{cq} \sum_{\substack{h=1 \\ h \neq q}}^{n_c} \sum_{j=1}^{n_i} \bar{P}'^{ch}_c \bar{P}'^{ij}_i \hat{K}^{\pi}_{ickh}$$
$$+ \Delta\hat{\beta}_{cq} \sum_{j=1}^{n_i} \bar{P}'^{cq}_c \bar{P}'^{ij}_i \tilde{K}^{\pi}_{ijcq} + \Delta b(1-\gamma)$$

while from the equality condition coming from b we have

$$\Delta\hat{\beta}_{cq} + \sum_{k=1}^{N} \Delta\hat{\beta}_{hk} = 0 \tag{3.88}$$

which are the equations connecting the changes in the $\hat{\beta}_{hk}$ coefficients due to the updating of the sample sample. From the expressions section 3.12 and section 3.12 we may understand that the kernel matrix enters into the update of the lagrangian coefficients β not directly but combined with probabilities and we may define the average kernel matrix as:

$$\bar{Q}^k_i = \sum_{h=1}^{n_k} \sum_{j=1}^{n_i} \bar{P}^{ij}_i \bar{P}^{kh}_k \hat{K}^{\pi}_{ijkh} \tag{3.89}$$

Considering for the moment only the case where we are inserting a new training point which was not present before. Hence, we may define the index of the samples in the support vectors set S as $S = \{s_1, s_2, ..., s_l\}$ and the section 3.12 and section 3.12 can be represented in matrix form as

$$\begin{bmatrix} 0 & 1 & ... & 1 \\ (1-\gamma) & \bar{Q}_{s_l}^{p_l} & ... & \bar{Q}_{s_1}^{p_l} \\ . & . & . & . \\ (1-\gamma) & \bar{Q}_{s_l}^{p_1} & ... & \bar{Q}_{s_l}^{p_l} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\beta_{s_1} \\ . \\ \Delta\beta_{s_l} \end{bmatrix} = - \begin{bmatrix} 1 \\ \bar{Q}_{s_1}^{k} \\ . \\ \bar{Q}_{s_l}^{k} \end{bmatrix} \Delta\beta_k \tag{3.90}$$

that is

$$\begin{bmatrix} \Delta b \\ \Delta\beta_{s_1} \\ . \\ \Delta\beta_{s_l} \end{bmatrix} = \hat{\beta}\Delta\beta_k \tag{3.91}$$

where

$$\hat{\beta} = \begin{bmatrix} \beta \\ \beta_{s_1} \\ . \\ \beta_{s_l} \end{bmatrix} = -\mathbf{R} \begin{bmatrix} 1 \\ \bar{Q}_{s_1}^{k} \\ . \\ \bar{Q}_{s_l}^{k} \end{bmatrix} \tag{3.92}$$

where

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & ... & 1 \\ (1-\gamma) & \bar{Q}_{s_l}^{p_l} & ... & \bar{Q}_{s_1}^{p_l} \\ . & . & . & . \\ (1-\gamma) & \bar{Q}_{s_l}^{p_1} & ... & \bar{Q}_{s_l}^{p_l} \end{bmatrix}^{-1} \tag{3.93}$$

Define the non support vector set $N = E \cup R = \{n_1, ..., n_l\}$ and combining the above equations leads to:

$$\begin{bmatrix} \Delta h(s_{n_1}, a_{n_1}) \\ \Delta h(s_{n_1}, a_{n_2}) \\ . \\ \Delta h(s_{n_l}, a_{n_l}) \end{bmatrix} = \hat{\gamma}\Delta\beta_k \tag{3.94}$$

with $\hat{\gamma}$ defined as

$$\hat{\gamma} = \begin{bmatrix} 1 \\ \bar{Q}_{n_1}^{k} \\ . \\ \bar{Q}_{n_l}^{k} \end{bmatrix} + \begin{bmatrix} 0 & 1 & ... & 1 \\ (1-\gamma) & \bar{Q}_{n_1}^{p_l} & ... & \bar{Q}_{n_l}^{p_l} \\ . & . & . & . \\ (1-\gamma) & \bar{Q}_{n_l}^{p_1 q_1} & ... & \bar{Q}_{n_l}^{p_l} \end{bmatrix} \hat{\beta} \tag{3.95}$$

Given $\Delta\beta_k$ one can update $\beta_t, t \in S$ and b according to section 3.12 and update $\Delta h(s_t, a_t), t \in N$ according to section 3.12. Moreover section 3.9 suggests that $\beta_t, t \in N$ and $h(s_t, a_t), t \in S$ are constant if the set S stays unchanged. Hence it only necessary to solve the problem

on how to find the appropriate value $\Delta\beta_k$. On the other side, when we are updating the $API-BRM_\varepsilon$ with a sample already present into the training set, we keep the average kernel matrix section 3.12 for the incremental update of the lagrangian coefficients β_t. Now the section 3.12 may also be obtained removing the point (s_i,a_i) from the training set, updating the transition probability $\bar{P}_c^{cq}$ considering that we have a new point (s_c,a_c,r_{cq},s'_{cq}) and then adding it again to the training set now taking into account the updated probability $\bar{P'}_c^{cq}$ and therefore the update of the average kernel matrix $\bar{Q}_i^c$.

All the above incremental equations are valid while the vectors do not migrate from set R,E or S to another one. This suggest a way to cope with the problem of finding the correct $\Delta\beta_k$ choosing it to be the largest value that either can maintain the set S unchanged or eventually leads to the termination of the incremental algorithm. As a matter of fact the first step is to determine if the change $\Delta\beta_k$ should be positive or negative according to the section 3.9 $sign(\Delta\beta_k) = sign(-h(s_k,a_k))$ Then, eventually, one has to find out the bound on $\Delta\beta_k$ imposed by each sample in the training set D'. Considering only the case $\Delta\beta_k > 0$ (the opposite case $\Delta\beta_k < 0$ is quite similar), for a new (multi) sample (s_c,a_c,r_{cq},s'_{cq}) we have two cases:

- [1] $h(s_c,a_c)$ changes from $h(s_c,a_c) < -\varepsilon$ to $h(s_c,a_c) = -\varepsilon$ then the new sample is added to the set S and the algorithm terminates
- [2] if β_c increases from $\beta_c < 0$ up to $\beta_c = 0$ the new sample is added to the set E and the algorithm terminates

for each sample $t \in S$

- [3] if β_i changes from $0 < |\beta_i| < C$ to $|\beta_i| = C$ then sample (s_i,a_i) migrates from S to E. If $\beta_i = 0$ sample (s_i,a_i) migrates from S to R

for each sample $t \in E$

- [4] if $h(s_i,a_i)$ changes from $|h(s_i,a_i)| > \varepsilon$ to $|h(s_i,a_i)| = \varepsilon$ then sample (s_i,a_i) migrates from E to S

for each sample $t \in R$

- [5] if $h(s_i,a_i)$ changes from $|h(s_i,a_i)| < \varepsilon$ to $|h(s_i,a_i)| = \varepsilon$ then sample i migrates from R to S.

The book-keeping procedure then is to trace for each sample into the training set D' against these five cases and to determine the allowed $\Delta\beta_c$ for each sample according to the section 3.12 and section 3.12 while the final $\Delta\beta_k$ is defined as the one with the minimum absolute value among the possible $\Delta\beta_k$.

The migration process needs that the matrix $\mathbf{R}$ must be updated whenever the set S changes its composition. It is possible to do the task efficiently without explicitly computing the matrix inverse. When the sample $k \in S$ is removed the updated $\mathbf{R^{new}}$ can be obtained as follows:

$$\begin{aligned} \mathbf{R^{new}} &= \mathbf{R_{I,I}} - \frac{\mathbf{R_{I,k}}\mathbf{R_{k,I}}}{R_{k,k}} \\ \mathbf{I} &= \{1,...,k,k+2,...,s_l+1\} \end{aligned} \tag{3.96}$$

while when a new sample is added to set S the update can be found through:

$$\mathbf{R}^{\mathbf{new}} = \begin{bmatrix} & \mathbf{R} & & 0 \\ & & & . \\ & & & . \\ 0 & . & . & 0 \end{bmatrix} + \frac{1}{\hat{\gamma}_p} \begin{bmatrix} \mu \\ 1 \end{bmatrix} \begin{bmatrix} \mu & 1 \end{bmatrix} \tag{3.97}$$

where μ and $\hat{\gamma}_p$ are defined as

$$\mu = -\mathbf{R} \begin{bmatrix} 1 \\ \bar{Q}_1^p \\ . \\ \bar{Q}_1^p \end{bmatrix} \quad \hat{\gamma}_i = \bar{Q}_p^p + \begin{bmatrix} 1 \\ \bar{Q}_1^p \\ . \\ \bar{Q}_l^p \end{bmatrix} \mu \tag{3.98}$$

when the sample (s_p, a_p) was moved from E to R. Whenever the sample (s_i, a_i) is added to S μ and $\hat{\gamma}_p$ can be obtained from the section 3.12 and section 3.12. An initial SVR solution in general can be obtained from a batch SVR solver often being the most efficient approach. If we want to unlearn an existing sample from the training set D we can also provide a decremental algorithm. If the sample to unlearn $i \in R$ clearly does not contribute to the SVR solution and its removal does not require adjustments. However, if on the other hand (s_i, a_i) has $\beta_i \neq 0$ one can gradually reduce β_i while ensuring all the other samples in the training set to satisfy the KKT conditions. As a result the decremental algorithm follows the same strategy of the incremental one taking into account:

- the direction of the change of β_i from $sign(\Delta\beta_i) = sign(h(s_i, a_i))$
- there is no case [1] because the removed (s_k, a_k) does not need to satisfy the KKT conditions
- the condition of case [2] becomes: β_i changing from $|\beta_i| > 0$ to $|\beta_i| = 0$.

3.13 Appendix 3B: $API - BRM_\varepsilon$ Primal Solution

Now we focus on the solution of the $API - BRM_\varepsilon$ optimization problem in the primal looking for the approximation action value function for each iteration of the API algorithm. According to the previous section we are able to process the data in batch mode solving the optimization problem in section 3.9 or eventually in incremental mode solving the optimization problem in section 3.11. We have already shown the connection between the two formulations when solving the problem in the dual. However SVR can also be solved directly in the primal formulation [21], [52]. Consider the data set

$$D_n = \{(\bar{s}_t, \bar{a}_t, \bar{r}_t, \bar{s}'_t) \quad t = 1, ..., n\} = \{(\bar{s}_i, \bar{a}_i, \bar{r}_{ij}, \bar{s}'_{ij}) \quad i = 1..., N, \quad j = 1, ..., n_i, \quad n_i = \sum_{j=1}^{n_i} n_{ij}, \quad n = \sum_{i=1}^{N} n_i\}$$

as described in the previous section. Now the slack variables of ξ_t, ξ_t^* in section 3.9 and using the Jensen's inequality can be written as

$$\begin{aligned} &\tfrac{1}{n}\textstyle\sum_{t=1}^{n}(\xi_t + \xi_t^*) = \tfrac{1}{n}\sum_{i=1}^{N} n_i(\bar{\xi}_i + \bar{\xi}_i^*) = \tfrac{1}{n}\sum_{i=1}^{N} n_i \ell_\varepsilon(\hat{D}^\pi Q(\bar{s}_i, \bar{a}_i) - \bar{r}(\bar{s}_i, \bar{a}_i)) \\ &= \tfrac{1}{n}\textstyle\sum_{i=1}^{N} n_i \ell_\varepsilon(\sum_{j=1}^{n_i} \bar{P}_i^{ij}(\hat{D}^\pi Q(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}) - \bar{r}_{ij})) \leq \tfrac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{n_i} n_{ij} \ell_\varepsilon(\hat{D}^\pi Q(\bar{s}_i, \bar{a}_i, \bar{s}'_{ij}) - \bar{r}_{ij}) \\ &= \tfrac{1}{n}\textstyle\sum_{t=1}^{n} \ell_\varepsilon(\hat{D}^\pi Q(s_t, a_t, s'_t) - \hat{r}_t) = \mathbb{E}_n[\ell_\varepsilon(\hat{D}^\pi Q(s_t, a_t, s'_t) - \hat{r}_t)] \end{aligned} \tag{3.99}$$

Hence BRM_ε in the primal requires solving the optimization problem

$$\hat{Q} = \arg\min_{Q\in\mathcal{H}} \{\mathbb{E}_n[\ell_\varepsilon(\hat{D}^\pi Q(s_t,a_t,s_t') - \hat{r}_t)] + \lambda_n \|Q\|_{\mathcal{H}}^2\} \tag{3.100}$$

and $\hat{D}^\pi Q(s_t,a_t,s_t') = Q(s_t,a_t) - \gamma \sum_{a_t'\in A} \pi(a_t'|s_t') Q(s_t',a_t')$ where we have dropped b for simplicity which does not affect the generalization performance of SVR. Now according to the Representer Theorem the optimal function can be expressed as a linear combination of the kernel functions $\tilde{\kappa}^\pi(\eta,\zeta)$ with $\eta,\zeta = (s,a,s')$ centered into the training samples as:

$$\hat{D}^\pi Q(s,a,s') = \sum_{t=1}^{n} \beta_t \tilde{\kappa}^\pi(s_t,a_t,s_t',s,a,s') \tag{3.101}$$

assuming $Q(s,a) = \sum_{t=1}^n \beta_t \hat{\kappa}^\pi(s_t,a_t,s_t',s,a)$ and $Q(s',a') = \sum_{t=1}^n \beta_t \hat{\kappa}^\pi(s_t,a_t,s_t',s',a')$. Using these expansion in BRM_ε minimization problem section 3.13 this reduces to finding the expansion coefficients vector β^* such that

$$\beta^* = \arg\min_{\beta} \{\mathbb{E}_n[\ell_\varepsilon(\mathbf{K}_t^\pi \beta - \hat{r}_t)] + \lambda_n \beta^T \mathbf{K}^\pi \beta\} \tag{3.102}$$

where $\mathbf{K}_{ij}^\pi = \{\tilde{\kappa}(\eta_i,\zeta_j)\}$ is the Bellman kernel matrix and $\mathbf{K}_t^\pi$ the $t-th$ row of $\mathbf{K}^\pi$. In principle as long as $l_\varepsilon(\cdot)$ is differentiable one may optimize the section 3.13 by a gradient descent algorithm. In fact the finite Newton algorithm can be proved to converge in a finite number of steps and can efficiently solve the SVR. However this is not directly applicable to the linear loss function case since $l_\varepsilon(\cdot)$ is not differentiable everywhere. One alternative may be to slightly modify the loss function to overcome the problem defining $\ell_{\varepsilon,\Delta}$ as in [52]

$$\ell_{\varepsilon,\Delta}(z) = \begin{cases} 0 & if \quad |z| \le \varepsilon \\ (|z|-\varepsilon)^2 & if \quad \varepsilon < |z| < \Delta \\ (\Delta-\varepsilon)(2|z|-\Delta-\varepsilon) & if \quad |z| \ge \Delta \end{cases}$$

assuming $\Delta > \varepsilon$ the first order derivative can be written as

$$\frac{\partial \ell_{\varepsilon,\Delta}(z)}{\partial z} = \begin{cases} 0 & if \quad |z| \le \varepsilon \\ 2sign(z)(|z|-\varepsilon) & if \quad \varepsilon < |z| < \Delta \\ 2sign(z)(\Delta-\varepsilon) & if \quad |z| \ge \Delta \end{cases}$$

The properties of this loss function are controlled by the two parameters (ε,Δ) which is able to encompass different type of losses as reported in table 3.1.

Introducing the new definition of the loss function into section 3.13 we have the following nonlinear SVR problem

$$\beta^* = \arg\min_{\beta} \{\mathbb{E}_n[\ell_{\varepsilon,\Delta}(\mathbf{K}_t^\pi \beta - \hat{r}_t)] + \lambda_n \beta^T \mathbf{K}^\pi \beta\} \tag{3.103}$$

Now define the residual vectors:

$$\begin{aligned} \tilde{\mathbf{r}}_t^\pi(\beta) &= \mathbf{K}_t^\pi \beta - \hat{r}_t \\ \tilde{\mathbf{r}}^\pi(\beta) &= \mathbf{K}^\pi \beta - \mathbf{r} \end{aligned} \tag{3.104}$$

Δ	ε	**Loss type**	**sparse**
$0 < \Delta < \infty$	0	Huber	no
$\Delta = \infty$	0	quadratic	no
$\Delta \to \varepsilon$	0	linear	no
$\Delta = \infty$	$0 < \varepsilon < \infty$	quadratic insensitive	yes
$\Delta \to \varepsilon$	$0 < \varepsilon < \infty$	linear insensitive	yes

Table 3.1: Loss function derived from $\ell_{\varepsilon,\Delta}(z)$

where $\mathbf{r}$ is the reward vector. Moreover defining the sign vectors $\mathbf{s}^\pi(\beta) = [s_1^\pi(\beta), ..., s_n^\pi(\beta)]^T$ and $\bar{\mathbf{s}}^\pi(\beta) = [\bar{s}_1^\pi(\beta), ..., \bar{s}_n^\pi(\beta)]^T$ by

$$\mathbf{s}_t^\pi(\beta) = \begin{cases} 1 & if \quad \varepsilon < \tilde{\mathbf{r}}_t^\pi(\beta) < \Delta \\ -1 & if \quad -\Delta < \tilde{\mathbf{r}}_t^\pi(\beta) < -\varepsilon \\ 0 & otherwise \end{cases}$$

and

$$\bar{\mathbf{s}}_t^\pi(\beta) = \begin{cases} 1 & if \quad \tilde{\mathbf{r}}_t^\pi(\beta) \geq \Delta \\ -1 & if \quad \tilde{\mathbf{r}}_t^\pi(\beta) \leq -\Delta \\ 0 & otherwise \end{cases}$$

and putting $w_t^\pi(\beta) = (s_t^\pi(\beta))^2$ we define the active matrix $\mathbf{W}^\pi(\beta) = diag[w_1^\pi(\beta), ..., w_n^\pi(\beta)]$ one may define the function $L_{\varepsilon,\Delta}^\pi(\beta)$ as

$$L_{\varepsilon,\Delta}^\pi(\beta) = (\tilde{\mathbf{r}}^\pi(\beta))^T \mathbf{W}^\pi(\beta)\tilde{\mathbf{r}}^\pi(\beta) - 2\varepsilon(\tilde{\mathbf{r}}^\pi(\beta))^T \mathbf{s}^\pi(\beta) \tag{3.105}$$
$$+2(\Delta-\varepsilon)(\tilde{\mathbf{r}}^\pi(\beta))^T \mathbf{s}^\pi(\beta) + \lambda\beta^T \mathbf{K}^\pi\beta + \varepsilon^T \mathbf{W}^\pi(\beta)\varepsilon - (\Delta^2-\varepsilon^2)(\bar{\mathbf{s}}^\pi(\beta))^T \bar{\mathbf{s}}^\pi(\beta)$$

and write the section 3.13 as

$$\beta^* = \arg\min_\beta L_{\varepsilon,\Delta}^\pi(\beta) \tag{3.106}$$

The function $L_{\varepsilon,\Delta}^\pi(\beta)$ in the optimization problem in section 3.13 is piecewise quadratic and convex function with a unique minimizer and continuously differentiable with respect to β. Although $L_{\varepsilon,\Delta}^\pi(\beta)$ is not twice differentiable it is possible to define a generalized Hessian matrix which allows for the use of the finite Newton algorithm. Eventually the finite Newton algorithm applied iteratively solve the problem in section 3.13 proceed as follows

1. fix $k = 0$ and choose a starting point β^0
2. if β^k is the minimizer of section 3.13 stop
3. compute the Newton step $\mathbf{h}^\pi$
4. choose step size ρ and set $\beta^{k+1} = \beta^k + \rho\mathbf{h}^\pi$

For any given value of β a point (s_t, a_t, s'_t) is a support vector if $|\tilde{\mathbf{r}}_t^\pi(\beta)| > \varepsilon$ and let $SV1 = \{t \,|s_t^\pi(\beta) \neq 0\}$ denote the index set of supports lying in the quadratic part of the loss function, $SV2 = \{t \,|\bar{s}_t^\pi(\beta) \neq 0\}$ the index set of support vectors lying in the linear part of the loss function and n_{nSV} the index set of the non support vectors. The gradient of $L_{\varepsilon,\Delta}^\pi(\beta)$ with respect to β is:

$$\begin{aligned}\nabla L_{\varepsilon,\Delta}^\pi(\beta) = 2(\mathbf{K}^\pi)^T \mathbf{W}^\pi(\beta)\tilde{\mathbf{r}}^\pi(\beta) - 2\varepsilon(\mathbf{K}^\pi)^T \mathbf{s}^\pi(\beta) \\ +2(\Delta-\varepsilon)(\mathbf{K}^\pi)^T \bar{\mathbf{s}}^\pi(\beta) + 2\lambda \mathbf{K}^\pi \beta\end{aligned} \tag{3.107}$$

while if we define the set A^π such that

$$A^\pi = \{\beta \in \mathbb{R}^n \,|\, \exists t \quad |\tilde{r}_t^\pi(\beta)| = \varepsilon \quad or \quad |\tilde{r}_t^\pi(\beta)| = \Delta\} \tag{3.108}$$

the Hessian exists for $\beta \notin A^\pi$ while for $\beta \in A^\pi$ can be arbitrarily defined to one of its limits and we have the generalized Hessian as:

$$\nabla^2 L_{\varepsilon,\Delta}^\pi(\beta) = 2(\mathbf{K}^\pi)^T \mathbf{W}^\pi(\beta)\mathbf{K}^\pi + 2\lambda \mathbf{K}^\pi \tag{3.109}$$

Hence the Newton step at the $k-th$ iteration is given by

$$\mathbf{h}^\pi = -(\nabla^2 L_{\varepsilon,\Delta}^\pi(\beta^k))^{-1} \nabla L_{\varepsilon,\Delta}^\pi(\beta^k) \tag{3.110}$$

Reordering the training samples such that the first n_{SV1} are supports of the quadratic part of the loss function, then we have n_{SV2} supports of the linear part and finally the n_{nSV} the non support training samples the section 3.13 can be written as:

$$\begin{bmatrix}\mathbf{h}_{SV1}^\pi \\ \mathbf{h}_{SV2}^\pi \\ \mathbf{h}_{nSV}^\pi\end{bmatrix} = \begin{bmatrix}\mathbf{K}^\pi{}_{SV1,SV1} + \lambda \mathbf{I}_{SV1,SV1} & \mathbf{K}^\pi{}_{SV1,SV2} & \mathbf{K}^\pi{}_{SV1,nSV} \\ \mathbf{0} & \lambda \mathbf{I}_{SV2,SV2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \lambda \mathbf{I}_{nSV,nSV}\end{bmatrix}^{-1} \times \begin{bmatrix}\mathbf{r}_{SV1} + \varepsilon[\mathbf{s}^\pi(\beta^k)]_{SV1} \\ -(\Delta-\varepsilon)[\bar{\mathbf{s}}^\pi(\beta^k)]_{SV2} \\ \mathbf{0}\end{bmatrix} - \begin{bmatrix}\beta_{SV1}^k \\ \beta_{SV2}^k \\ \beta_{nSV}^k\end{bmatrix}$$

which can be simplified as

$$\begin{bmatrix}\mathbf{h}_{SV1}^\pi \\ \mathbf{h}_{SV2}^\pi \\ \mathbf{h}_{nSV}^\pi\end{bmatrix} = \begin{bmatrix}(\mathbf{K}^\pi{}_{SV1,SV1} + \lambda \mathbf{I}_{SV1,SV1})^{-1}(\mathbf{r}_{SV1} + \varepsilon[\mathbf{s}^\pi(\beta^k)]_{SV1} + (\Delta-\varepsilon)\lambda^{-1}\mathbf{K}^\pi{}_{SV1,SV2}[\bar{\mathbf{s}}^\pi(\beta^k)]_{SV2}) - \beta_{SV1}^k \\ -(\Delta-\varepsilon)\lambda^{-1}[\bar{\mathbf{s}}^\pi(\beta^k)]_{SV2} - \beta_{SV2}^k \\ -\beta_{nSV}^k\end{bmatrix}$$

allowing the evaluation of the solution in a finite number of steps at a cost of $O(n_{SV1}^3)$ using for example the Cholesky factorization. Accordingly we may extend the batch or the incremental methods presented into the previous section in order to cope with the modified $\varepsilon-$insensitive loss function $\ell_{\varepsilon,\Delta}$. We keep this exercise as future work.

Chapter 4

$API - BRM_\varepsilon$ Theoretical Analysis

4.1 Introduction

Theoretical guarantees on the RL algorithm's performance can be described using the Bellman Error $BE = \|Q^* - Q^\pi\|_{p,\rho}^q$ as a performance loss measure. In fact, we may consider testing a RL algorithm giving a greedy policy π with respect to an estimated action value function. If the agent initial state is distributed according to some performance measuring distribution $\rho \in \mathcal{M}(S)$ one may use the $L_{p,\rho}^q$ norm of the Bellman Error evaluate the expected difference between the value following the optimal policy π^* and the one obtained following the policy π. In this case, the probability distribution ρ should represent the importance of the different regions of the state space and the initial state distribution. Hence, in this section we analyze the statistical properties of the regularized $API - BRM_\varepsilon$ providing a finite sample bound of the performance loss $\|Q^* - Q^{\pi_K}\|_{p,\rho}^q$ where π_K is the greedy policy w.r.t. Q^{K-1} after K PI as depicted in Algorithm 1 with ρ is a performance evaluation measure. This chapter is rather technical and the theoretical analysis proceeds as follows:

- in section 4.2 we introduce the mixing processes and discuss the implications of using non-i.i.d. data samples proving some technical lemmas
- in section 4.3 we make some technical assumption and discuss possible implications
- in section 4.4 we prove Theorem 4.2 which gives an upper bound on the statistical performance of the $API - BRM_\varepsilon$ policy evaluation step
- in section 4.5 we report Theorem 4.3 about the effect of the error propagation for the ε-insensitive loss function in the BR sequence
- in section 4.6 in Theorem 4.4 we prove an upper bound on the performance loss $\|Q^* - Q^{\pi_K}\|_{1,\rho}$ of $API - BRM_\varepsilon$
- and finally in section 4.7 we discuss the results and characteristics of the statistical bound

In practice we study the policy evaluation error of $API - BRM_\varepsilon$ using the SVR regularization scheme where we suppose that given any policy π one may obtain Q^π by solving the

regularized problem in section 3.7 with a given π_k at step k of the API procedure. Theorem 4.2 provides an upper bound on the Bellman error defined as $\|Q-T^\pi Q\|_{1,\nu}$. Hence one may show how the Bellman errors of the policy evaluation procedure propagate through the $API-BRM_\varepsilon$ in Theorem 4.4.

In the evaluation of the theoretical bound, we adapt to $API-BRM_\varepsilon$ some ideas from by [35]. However, for $API-BRM_\varepsilon$ we assume the data samples to be β-mixing distributed and make use of non-parametric function approximation within the BRM approach based on $\varepsilon-$insensitive loss function. In the work of [35] they assume i.i.d. data samples, non-parametric approximation function and the alternative formulation of BRM approach (as described in the optimization problem section 3.6) based on quadratic loss function. As a result the differences between the two methods are remarkable. In $API-BRM_\varepsilon$ we take advantage of the sparsity property as well as of the incrementality enabling the addition or removal of training samples very effectively. An upper bound of the performance loss using a parametric function approximation is presented in [6] where they use β-mixing sequences and the alternative formulation of BRM approach (as described in the optimization problem section 3.6) using quadratic loss function. However, they work with finite dimensional space while we use non-parametric function spaces which are essentially infinite dimensional. As a result our work is the first attempt to give some theoretical justification of using regularized non-parametric approximation function using SVR to solve the generalization problem in RL with action value function.

4.2 Mixing Processes

A sequence of random variables is called a *time series* in the statistics literature and a (discrete time) *stochastic process* in the probability literature. Let X be a measurable space and $Y \subseteq \mathbb{R}$ be closed. Furthermore let $(\Omega, \sigma_\Omega, \nu)$ be a probability space and $\mathcal{Z} = (Z)_{i\geq 1}$ be a stochastic process such that $Z_i : \quad \Omega \to X \times Y$ for all $i \geq 1$. For $n \geq 1$ we write $Z_n = \{(X_1,Y_1),...,(X_n,Y_n))\} = \{Z_1,...,Z_n)\}$ for a training set of length n distributed according to the first n components of $\mathcal{Z}$. Moreover we assume $\mathcal{Z}$ to be a *stationary process* i.e. the $(X \times Y)^n$-valued random variables $(Z_{i_1},...,Z_{i_n})$ and $(Z_{i_1+i},...,Z_{i_n+i})$ have the same distribution for all $n,i,i_1,...,i_n \geq 1$. We further write P for the distribution of one (and thus all) Z_i for all measurable $A \subseteq X \times Y$ we have $P(A) = \nu(\{\omega \in \Omega : Z_i(\omega) \in \sigma_\Omega\})$. Knowing a set of samples Z_n drawn from the probability distribution ν our goal is to find a good approximation of a function f. However, approximating a function from sparse samples is an ill-posed problem. To deal with it we may use the regularization theory. Learning rates measure the quality of the function approximation and certainly play an important role in learning theory. Mostly the assumption that the samples were drawn independently from the unknown probability $P(X,Y)$ is made in learning theory. However, independence in some context can be a restrictive concept and to learn from stationary processes whose components are not independent it is necessary to replace the independence assumption by a notion that still guarantees certain concentration inequalities. The *mixing* condition allows a firm mathematical foundation of the notion of near independence and permits one to derive a more robust theory. In RL the mixing scenario can be easily encountered in the context of online policy scenario as the actual distribution may be dependent on some of the previous observations. In general we prefer to drop the assumption on the in-

dependence of sample sequence and study the error analysis of regularized regression for the strongly exponentially mixing sequence. Differently from the i.i.d. case, the learning performance of the algorithms with mixing sequence is not measured directly by the sample size (or the number of observations). Instead, it is related to the effective number of observations. On the other hand we also rely on the approximation property and capacity of the RKHS.

To learn from stationary processes whose component are not independent [83] suggests that it is necessary to replace the independence assumption by a notion that still guarantees certain concentration inequalities. Thus, the index t or time, does not affect the distribution of a variable Z_t in a stationary sequence. This does not imply independence however. In particular, for $i < j < k$, $P(Z_j|Z_i)$ may not equal $P(Z_k|Z_i)$, that is, conditional probabilities may vary at different points in time. The following is a standard definition giving a measure of the dependence of the random variables Z_t within a stationary sequence. We recall some standard mixing coefficients and their basic properties from [105]. The $\alpha-$mixing is based on the $\alpha-$mixing coefficients:

$$\alpha_n = \sup\{\, |\mu(A\cap B) - \mu(A)\mu(B)|\, i \geq 1, A \in \mathcal{A}_1^i \quad and\, B \in \mathcal{A}_{i+n}^{\infty}\}, \quad n \geq 1 \tag{4.1}$$

where $\mathcal{A}_1^i$ and $\mathcal{A}_{i+n}^{\infty}$ are the $\sigma-$algebras generated by $(Z_1,...,Z_i)$ and $(Z_{i+n},Z_{i+n+1},...)$ respectively. Moreover the process $\mathcal{Z}$ is called strong $\alpha-$mixing if for some $\bar{\alpha}_0,\bar{\alpha}_1,\bar{\alpha}_2 > 0$ we have $\alpha_n \leq \bar{\alpha}_0 \exp(-\bar{\alpha}_1 n^{\bar{\alpha}_2})$. Accordingly the β-mixing coefficients are defined as

$$\beta_n = \frac{1}{2}\sup\{\sum_{i,j=1}^{\infty} |\mu(A_i\cap B_j) - \mu(A_i)\mu(B_j)|\, : A_i \subset \mathcal{A}_1^i\, B_j \subset \mathcal{A}_{i+n}^{\infty}\, partitions\} \tag{4.2}$$

and the process is said to be β-mixing if $\beta_n \to 0$ with $n \to \infty$ and exponentially β-mixing if for some constant $\bar{\beta}_0,\bar{\beta}_1 > 0$ we have $\beta_n \leq \bar{\beta}_0 \exp(-\bar{\beta}_1 n)$ It is obvious from the definitions that all mixing coefficients equal 0 if $\mathcal{A}$ and $\mathcal{B}$ are independent. Both α_n and β_n measure the dependence of an event on those that occurred more than n units of time in the past. Moreover β-mixing imply α-mixing in the non-i.i.d. scenario. In the most general version, future samples depend on the training sample Z_n and thus the generalization error or true error of the hypothesis trained on Z_n must be measured by its expected error conditioned on the sample Z_n. The stability of a learning algorithm generally assumes the i.i.d. property. Replacing an element in a sequence with another has no effect on the expected value of a random variable defined over that sequence. Hence, the following equality holds, $\mathbb{E}[\ell(Z_1,...,Z_i,..,Z_n)|Z_n] = \mathbb{E}[\ell(Z_1,...,Z',...,Z_n)|Z_n,Z']$ for a random variable ℓ that is a function of the sequence of random variables $Z_n = (Z_1,...,Z_n)$. However, if the points in that sequence Z_n are dependent, this equality may not hold anymore. The main technique working with this problem, is based on the independent block sequence which consists on eliminating several blocks of contiguous points from the original dependent sequence, leaving us with some remaining blocks of points. Instead of these dependent blocks, we then consider independent blocks of points, each with the same size and the same distribution (within each block) as the dependent ones.

Several lemmas can be proved using α or β-mixing distribution, the expected value of a random variable defined over the dependent blocks is close to the one based on these independent blocks. As a result using independent blocks brings us back to a situation similar to the i.i.d. case, with i.i.d. blocks replacing i.i.d. points [105],[61]. The blocking

device of [105] may be built partitioning the set $\{1,...,n\}$ determined by the choice of an integral block length a_n with $n = 2\mu_n a_n$ for appropriate $\mu_n, a_n \in \mathbb{N}$. The partition will have $2\mu_n$ blocks of integral block length a_n such that $n - 2a_n \leq 2\mu_n a_n \leq n$ and a residual block for $1 \leq j \leq \mu_n$ we have

$$\begin{aligned} H_j &= \{i : 2(j-1)a_n + 1 \leq i \leq (2j-1)a_n\} \quad head \\ T_j &= \{i : (2j-1)a_n + 1 \leq i \leq 2ja_n\} \quad tail \\ R &= \{2\mu_n a_n + 1, ..., n\} \quad residual \end{aligned} \tag{4.3}$$

The structure of the blocks goes as $H_1, T_1, H_2, T_2, ..., H_{\mu_n}, T_{\mu_n}, R$ where we define $H = \cup_{1 \leq j \leq \mu_n} H_j$. The samples in every second block are replaced by ghosts samples whose joint marginal distribution is kept the same as that of the original samples. For the new random variables the new blocks are now independent of each other. Consider the sequence of random variables $\bar{Z}'(H) = \{Z'_i \, i \in H\}$ such that $\bar{Z}'(H)$ is independent of $\bar{Z}_n$ and the blocks $\bar{Z}'(H_j)$ with $j = 1, .., \mu_n$ are i.i.d. each block having the same distribution as a block from the original sequence. Let $\mathcal{F}$ be some space of measurable real-valued functions with a domain $\mathcal{Z}$. Now for any $f \in \mathcal{F}$ we may derive a the function space $\bar{\mathcal{F}} = \{\bar{f} : f \in \mathcal{F}\}$ and $\bar{f} : \mathcal{Z}^{a_n} \to \mathbb{R}$. Now we quote the following result:

Lemma 4.1. *[[105] Lemma 4.1] Suppose $(Z_1, ..., Z_n) \in \mathcal{Z}$ is a stationary β-mixing process with mixing coefficients β_n and $\bar{Z}' \in \mathcal{Z}$ the block independent samples For any measurable function $f : \mathcal{Z}^{\mu_n a_n} \to \mathbb{R}$ we have $\mathbb{E}[f(\bar{Z}(H)) - f(\bar{Z}'(H))] \leq \|f\|_\infty (\mu_n - 1)\beta_{a_n}$*

Now we report a technical Lemma which apply for the β-mixing process as stated in [5] which will come in handy in the next sections:

Lemma 4.2. ***(Relative Deviation Inequality)*** *Lemma 2 in [5] Consider a $\mathcal{Z}-$valued stationary β-mixing sequence $Z = \{Z_t\}_{t=1}^n$ and a permissible class $\mathcal{F}$ of real functions f with domain Z. Assume that for some $M > 0$ we have $\sup_{f \in \mathcal{F}} \|f\|_\infty \leq M$. Then fix $n \in \mathbb{N}$ and $\varepsilon, \eta > 0$. Let $\bar{Z}'(H)$ be a $(\mu_n, a_n)-$independent blocks sequence with a residual block R satisfying $|R| \leq \frac{\varepsilon\eta}{6M}$. Then*

$$\begin{aligned} &\mathbb{P}\Big\{ \sup_{f \in \mathcal{F}} \Big| \frac{\frac{1}{n}\sum_{i=1}^n f(Z_i) - \mathbb{E}[f(Z)]}{\eta + |\mathbb{E}[f(Z)]|} \Big| > \varepsilon \Big\} \\ &\leq 2\mathbb{P}\Big\{ \sup_{f \in \mathcal{F}} \Big| \frac{\frac{1}{\mu_n}\sum_{j=1}^{\mu_n} \bar{f}(H'_j) - \mathbb{E}[\bar{f}(H_1)]}{a_n\eta + |\mathbb{E}[\bar{f}(H_1)]|} \Big| > \tfrac{2}{3}\varepsilon \Big\} + 2\beta_{a_n}\mu_n \end{aligned} \tag{4.4}$$

The following Lemma generalizing for $p, q = 1, 2$ Lemma 3 in [5] relates the covering number of $\mathcal{N}_p(\varepsilon, \mathcal{F}, \|z\|_{p,n}^q)$ with $\mathcal{N}_p(\bar{\varepsilon}, \bar{\mathcal{F}}, \|\bar{z}(H)\|_{p,\mu_n}^q)$:

Lemma 4.3. ***(Covering Number)*** *[Extension of Lemma 3 in [5]] For any $(z_1, ..., z_n) \in \mathcal{Z}^n$ and for $p = 1, 2$ we have*

$$\mathcal{N}_p(\varepsilon, \bar{\mathcal{F}}, \|\bar{z}(H)\|_{p,\mu_n}^p) \leq \mathcal{N}_p\Big(\Big[\frac{2(1 - \frac{|R|}{n})}{4a_n^p} \Big]^{\frac{1}{p}} \varepsilon, \mathcal{F}, \|z\|_{p,n}^p \Big) \tag{4.5}$$

Proof. Consider that for any function $f : \mathcal{Z} \to \mathbb{R}$ we may bound $\|\bar{f}\|_{p,\bar{z}(H)_{\mu_n}}^p$ in terms of $\|f\|_{p,z_n}^p$. In fact considering that $2a_n\mu_n = n - |R|$ and using the Jensen's inequality and the

fact thar $H \subset \{1,...,n\}$ we have

$$\|\bar{f}\|^p_{p,\bar{z}(H)_{\mu_n}} = \frac{1}{\mu_n}\sum_{j=1}^{\mu_n}\Big|\sum_{i\in H_j} f(z_i)\Big|^p \leq$$

$$\frac{a_n^p}{\mu_n a_n}\sum_{i\in H}|f(z_i)|^p \leq \frac{4a_n^p}{2(1-\frac{|R|}{n})}\|f(z)\|^p_{p,z_n} \tag{4.6}$$

Now considering that for any $f_1, f_2 \in \mathcal{F}$ using the previous inequality we have $\overline{f_1 - f_2} = \bar{f}_1 - \bar{f}_2$ hence we get $\|\bar{f}_1 - \bar{f}_2\|^p_{p,\bar{z}(H)_{\mu_n}} \leq \Big(\frac{4a_n^p}{2(1-\frac{|R|}{n})}\Big)\|f_1 - f_2\|^p_{p,z_n}$ therefore any $\Big[\frac{2(1-\frac{|R|}{n})}{4a_n^p}\Big]^{\frac{1}{p}}\varepsilon$ cover of $\mathcal{F}$ is an $\varepsilon-$cover of $\mathcal{F}$. □

We may introduce a slightly modified version of Theorem 4 in [5] and working for both cases $p = 1,2$ (which generalizes for exponentially β-mixing stochastic processes Theorem 19.3 in [40] and stated only for $p = 2$ and i.i.d. processes):

Theorem 4.1. ***(Relative Deviation Concentration Inequality)****[Theorem 4 in [5]] Consider a $\mathcal{Z}-$valued stationary β-mixing sequence $\bar{Z} = \{Z_t\}_{t=1}^n$ and a permissible class $\mathcal{F}$ of real valued functions f with domain $\mathcal{Z}$. Let $n \in \mathbb{N}$ and $K_1, K_2 \geq 1$ and choose $\eta > 0$ and $0 < \varepsilon < 1$. Hence assume that for any $f \in \mathcal{F}$ the following conditions hold:*

- $\mathbf{C_1}$ - $\|f\|_\infty \leq K_1$
- $\mathbf{C_2}$ - $\mathbb{E}[f^2(Z)] \leq K_2\mathbb{E}[f(Z)]$

further we consider the (a_n, μ_n) independent blocks with residual block R assuming that the following conditions also hold:

- $\mathbf{C_3}$ - $\sqrt{n}\sqrt{1-\varepsilon}\sqrt{\eta} \geq 576\max\{2K_1a_n, \sqrt{2a_nK_2}\}$
- $\mathbf{C_4}$ - $\frac{|R|}{n} \leq \frac{\varepsilon\eta}{6K_1}$ *and* $|R| \leq n/2$
- $\mathbf{C_5}$ - *For all $z_1,...,z_n \in \mathcal{Z}$ and all $\delta > \frac{\eta a_n}{8}$ we have*

$$\frac{\sqrt{\mu_n}\varepsilon(1-\varepsilon)\delta}{96\sqrt{2}a_n\max\{K_1,2K_2\}} \geq \int_{\frac{\varepsilon(1-\varepsilon)\delta}{16a_n\max\{K_1,2K_2\}}}^{\sqrt{\delta}}\Big(\log\mathcal{N}_p\Big(\frac{u}{(4a_n^p)^{\frac{1}{p}}}, \mathcal{F}, \|z\|_{p,n}\Big)\Big)^{\frac{1}{p}}du$$

Then there exists universal constants $c_1, c_2 > 0$ such that

$$\mathbb{P}\Big\{\sup_{f\in\mathcal{F}}\Big|\frac{\frac{1}{n}\sum_{t=1}^n f(Z_t) - \mathbb{E}[f(Z)]}{\eta + \mathbb{E}[f(Z)]}\Big| > \varepsilon\Big\} \leq c_1\exp\Big(-c_2\frac{\mu_n a_n\eta\varepsilon^2(1-\frac{2}{3}\varepsilon)}{\max\{a_n^2K_1^2, a_nK_2\}}\Big) + 2\beta_{a_n}\mu_n \tag{4.7}$$

The constants can be set to $c_1 = 120$ and $c_2 = \frac{1}{2^{12}3^4}$.

The proof of this theorem can be found in [40] for the quadratic norm $p = 2$ and i.i.d. samples. The proof of the generalization to the β-mixing processes for the quadratic norm $p = 2$ case can be found in [5]. Extension to $p = 1,2$ norms follows same argument as in the proof of lemma 4.3 and is omitted for brevity. Note that the difference of using

$p=1,2$ norms essentially is reflected into the covering number expression in condition $\mathbf{C_5}$ meaning the $\varepsilon-$cover for the two cases should be different.

Lastly, as some proofs presented in this work rely on it, we introduce in this section a useful tool called *peeling device* [96]. Intuitively, the peeling device considers some function of larger and larger balls centered around a given one, it will allow us to peel off sets of increasingly smaller probability. To introduce the peeling lets define the functions $\tau : \mathcal{F} \to [\rho, \infty)$ with $\rho > 0$ and a strictly increasing sequence $\{\sigma_l\}_{l\geq 0}$ starting with $\sigma_0 = 0$ but growing to infinity. We can peel the function space $\mathcal{F}$ into $\mathcal{F} = \cup_{l\geq 1}\mathcal{F}_{\sigma_l}$ where $\mathcal{F}_{\sigma_l} = \{f \in \mathcal{F} : \sigma_{l-1} \leq \tau(f) \leq \sigma_l, l = 1,2,..\}$. Then for any $\rho > 0$ and a residual $X_n(f)$ stochastic process indexed by $\mathcal{F}$ (meaning that implicitly depends on the functions f) we have:

Definition 4.1. ***(Peeling Device)****[[96]] Consider the function space $\mathcal{F}$ and let be $X_n(f)$ a stochastic process indexed by $\mathcal{F}$. Now consider the function $\tau : \mathcal{F} \to [\rho, \infty)$ with $\rho > 0$ with goal to have a probability upper bound on the weighted process $|X_n(f)|/\tau(f)$. Let $\{\sigma_l\}_{l\geq 0}$ be a strictly increasing sequence with $\sigma_0 = 0$ and $\lim_{l\to\infty} \sigma_l = \infty$. The function space $\mathcal{F}$ can be peeled off into smaller function space $\mathcal{F} = \cup_{l\geq 1}\mathcal{F}_{\sigma_l}$ with $\mathcal{F}_{\sigma_l} = \{f \in \mathcal{F} : \sigma_{l-1} \leq \tau(f) \leq \sigma_l, l = 1,2,..\}$ For any $a > 0$ results*

$$\mathbb{P}\Big\{\sup_{f\in\mathcal{F}} \frac{|X_n(f)|}{\tau(f)} > a\Big\} \leq \sum_{l\geq 1} \mathbb{P}\Big\{\sup_{f\in\mathcal{F}_{\sigma_l}} \frac{|X_n(f)|}{\tau(f)} > a\Big\} \leq \sum_{l\geq 1} \mathbb{P}\Big\{\sup_{f\in\mathcal{F},\tau(f)<\sigma_l} |X_n(f)| > a\sigma_{l-1}\Big\} \quad (4.8)$$

which is called peeling device where each $l = 1,2,..$ denotes the level of peeling.

This result lets us get probability inequalities for the weighted process from probabilities for the original process.

4.3 $API-BRM_\varepsilon$ Technical Assumptions

To analyze the statistical performance of $API-BRM_\varepsilon$ procedure we make the following assumptions:

Assumption 4.1. ***(MDP Regularity)*** *The set of states $S \subset \mathbb{R}^d$ is a compact subspace of the d-dimensional Euclidean space. The random immediate reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$ as well as the expected immediate rewards $r(s,a)$ are bounded by R_{max}.*

Results are stated for RL problems whose state space is a compact subset of $\mathbb{R}^d$ but other generalization to other spaces are possible. The reward boundness is a reasonable assumption in many practical problems.

Assumption 4.2. ***(Data Sampling)*** *Trajectory should be sufficiently representative and rapidly mixing. Samples are drawn from a stationary distribution $\nu \in \mathcal{M}(S \times A)$ and $\{(s_t, a_t, r_t, s'_t)\}_{t=1}^n$ is the sample path induced by a stochastic stationary behavior policy π_b. $\{s_t, a_t\}_{t=1}^n$ is strictly stationary and exponentially β-mixing with the actual rate given by the parameters $\bar{\beta}_0, \bar{\beta}_1$ with $(s_t, a_t) \sim \nu(s,a)$ $a_t \sim \pi_b(s_t)$ and $s'_t \sim \mathcal{P}(\cdot|s_t, a_t)$*

One might simplify the proof asking for i.i.d. data samples and proceed as in [35] eventually asking for fresh independent samples drawn at each PI step. The mixing assumption has to be considered a more realistic thinking of an agent interacting with the

environment. Hence, we stay with data samples generated in the single trajectory scenario where samples are not independent anymore but under certain conditions of the cMDP the process (s_t, a_t) gradually forgets its past (see [6] and [5]). Mixing processes represent a way to quantify this forgetting were the independent blocks technique can be used [105]. Globally exponentially stable unforced dynamical systems subjected to finite-variance continuous density input noise give rise to exponentially β-mixing Markov processes [45]. This class encompasses many dynamical systems common in the system identification and adaptive control. Moreover given the Borel $\sigma-$algebra $\sigma(\mathbb{R})$ and the distribution ν, a strictly stationary Markov chain $X = \{X_k, k \in \mathbb{Z}\}$ satisfy the geometric ergodicity if for each $x \in \mathbb{R}$ there exist functions $a, c : \mathbb{R} \to (0, \infty)$ such that

$$\forall n \geq 1 \quad , \forall B \in \mathcal{B}(\mathbb{R}), \quad |P(X_n | X_0 = x) - \nu(B)| \leq a(x) e^{-c(x)n}$$

It can be shown that for Markov processes, geometric ergodicity implies exponential β-mixing [28]. Hence, for such processes there is no loss of generality assuming exponential β-mixing. Finally it can be also shown that if the process $\{s_t\}_{t=1}^n$ is β-mixing the Markov process $\{(s_t, a_t, r_t)\}_{t=1}^n$ is also β-mixing with the same rate [20].

Assumption 4.3. ***(Function Space Capacity)*** *For any $R > 0$ let be $\mathcal{H}_R = \{f \in \mathcal{H} : \|f\|_{\mathcal{H}} \leq R\}$ then there exists $C > 0$ and $0 \leq \alpha < 1$ such that for any $u > 0$ and all $z_1, ..., z_n \in \mathcal{Z}$ the following metric entropy condition is satisfied:*

$$\log \mathcal{N}_1(u, \mathcal{H}_R, \|z\|_n) \leq C \left(\frac{R}{u}\right)^{2\alpha} \tag{4.9}$$

As we only deal with RKHS function space this assumption works fine. The α value enters into the convergence bound as the metric entropy measures the minimum number of balls with radius u required to completely cover a ball with radius R in $\mathcal{H}$. It is worth to point out that in case of finite function spaces the metric entropy condition has $\alpha = 0$ and $C = \log |\mathcal{H}|$ (see [82]).

Assumption 4.4. ***(Function Space Boundness)*** *The subset $\mathcal{F}^{|A|}$ is a separable and complete Caratheodory set with $R_{max} \leq Q_{max} \leq \infty$.*

We require all the functions in $\mathcal{F}^{|A|}$ to be bounded in order to have bounded solution for the regularized BRM_ε problem. Eventually a truncation operator should be applied to fulfill the assumption but we miss an explicit indication to avoid clutter. Hence to avoid measurability problems related to the use of taking supremum over uncountable function spaces it is also necessary to play with separable and complete Caratheodory set (see [82]).

Assumption 4.5. ***(Function Approximation Property)*** *The action value function of any policy belongs to $\mathcal{F}^{|A|}$*

Assuming this we are considering a function space large enough to include the true action value function and therefore we don't have to worry about the function approximation error. As described in section 3.4, when the *approximation error* exists, the result of bound described in the following apply only for the *estimation error*. Results regarding the behavior of the approximation error for small RKHS are discussed in [107]. Model selection procedures could used to balance estimation and approximation errors leading to optimal learning rates [46]. The way model selection should be implemented and analyzed is outside the scope of this work.

4.4 $API-BRM_\varepsilon$ Policy Evaluation Error

In this part we focus on the k-iteration of $API-BRM_\varepsilon$ and we prove the following theorem which provides an upper bound on the statistical performance of the $API-BRM_\varepsilon$ policy evaluation step.

Theorem 4.2. *(BRM_ε **Policy Evaluation Error**) For any fixed policy π let $\hat{Q}$ be the solution of the regularized $API-BRM_\varepsilon$ optimization problem*

$$\hat{Q} = \arg\min_{Q\in\mathcal{H}} \{ \hat{L}_{API-BRM_\varepsilon}(Q,\Pi_n,D_n) + \lambda_n \|Q\|^2_{\mathcal{H}} \} \tag{4.10}$$

with the choice of $\lambda_n = \left(\frac{1}{n\|Q^\pi\|^2_{\mathcal{H}}}\right)^{\frac{1}{1+2\alpha}}$. If the assumptions 4.3,..,4.5 hold, there exists $c(\delta)>0$ such that for $n\in\mathbb{N}$ and $0<\delta<1$ with n sufficiently large we have

$$\|\hat{Q} - T^\pi \hat{Q}\|_{2,\nu} \leq c(\delta) n^{-\frac{1}{1+2\alpha}} \tag{4.11}$$

and

$$c(\delta) = c_1 (\|Q^\pi\|^2_{\mathcal{H}})^{\frac{\alpha}{1+2\alpha}} \left[\frac{\log(\max(n,c_2)/\delta)}{\bar{\beta}_1}\right]^3$$

with probability at least $1-\delta$. In particular when $\alpha=0$ the above bound holds for $n \geq c_3 \exp(\bar{\beta}_1)$ while in case of $\alpha>0$ it holds when

$$n \geq \max(c_3 \exp(\bar{\beta}_1), 1/\|Q^\pi\|^2_{\mathcal{H}})$$

and also

$$\frac{1}{n}\left(\frac{c_4 \log(\max(n,c_2)/\delta)}{\bar{\beta}_1}\right)^{\frac{4}{\alpha}} \leq \|Q^\pi\|^2_{\mathcal{H}} \tag{4.12}$$

where $c_3, c_4 > 0$ depending only on Q_{max}.

Proof of this Theorem can be found in Appendix 4.9.

Hence we proved that there is a way to control the estimation error for any fixed policy π. Given the solution of the regularized $API-BRM_\varepsilon$ regularization problem $\hat{Q}$ if the assumptions 4.3,..,4.5 hold with the choice $\lambda_n = \left(\frac{1}{n\|Q^\pi\|^2_{\mathcal{H}}}\right)^{\frac{1}{1+2\alpha}}$ there exists $c(\delta)>0$ such that for $n\in\mathbb{N}$ and $0<\delta<1$ with n sufficiently large we have

$$\|\hat{Q} - T^\pi \hat{Q}\|_{2,\nu} \leq c(\delta) n^{-\frac{1}{1+2\alpha}} \tag{4.13}$$

with probability at least $1-\delta$.

4.5 $API-BRM_\varepsilon$ Error Propagation

In this subsection we study the effect of the ε-insensitive loss function in the BR sequence $\varepsilon_k^{BR} = |Q_k - T^{\pi_k} Q_k|$ on the performance loss $\|Q^* - Q^{\pi_K}\|_{1,\rho}$ π_K after K PI steps on the resulting policy π_k. The task of analyzing the error propagation and reduction in an API

algorithm has been performed by [66] and with some more details in [36] and [35]. Basically when we use an API algorithm generating a sequence of greedy policies π^{k-1} w.r.t. the approximate value function Q^k the resulting policy (remembering Q^* the optimal value defined in definition 2.2) one wishes to control the effects of the BR sequence $\varepsilon_k^{BR} = |Q_k - T^{\pi_k} Q_k|$ on the performance loss $\|Q^* - Q^{\pi_K}\|_{1,\rho}$ π_K after K PI steps. This in practice depends on the cMDP dynamics and in particular the performance loss relies on the difference between the sampling distribution ν and the future state action distribution in the form of $\rho P^{\pi_1} P^{\pi_2} ... P^{\pi_K}$ left linear operators. Essentially the necessary result can be obtain specializing for the norm $p = 1$ case the Theorem 3 in [36]. Before stating the theorem we need to define the following concentrability coefficients:

Definition 4.2. *(**Expected Concentrability Distribution**) Given $\nu, \rho \in \mathcal{M}(S \times A)$ $m \geq 0$ and an arbitrary sequence of stationary policies $\{\pi_k\}_{k=1}^m$ let $\rho P^{\pi_1} P^{\pi_2} ... P^{\pi_m} \in \mathcal{M}(S \times A)$ denote the future state-action distribution obtained when the first state-action is distributed according to ρ and then we follow the sequence of policies $\{\pi_k\}_{k=1}^m$. Define the concentrability coefficients:*

$$c_{PI_1,\rho,\nu}(m_1, m_2; \pi) = \sqrt{\mathbb{E}\Big[\Big|\frac{d(\rho (P^{\pi^*})^{m_1} (P^{\pi^*})^{m_2})}{d\nu}(s,a)\Big|^2\Big]} \tag{4.14}$$

with $(s,a) \sim \nu$. If the future state distribution $\rho (P^{\pi^*})^{m_1} (P^{\pi^*})^{m_2}$ is not absolutely continuous w.r.t. ν we assume $c_{PI_1,\rho,\nu}(m_1, m_2; \pi) = \infty$

Hence we are ready to present the result on error propagation in API from [36] using case $p = 1$.

Theorem 4.3. *($API - BRM_\varepsilon$ **Error Propagation** - Theorem 3 of [36]) For $0 \leq k \leq K$ define $a_k = \frac{(1-\gamma)\gamma^{K-k-1}}{1-\gamma^{K+1}}$ and assume $Q_{max} \leq \frac{R_{max}}{1-\gamma}$. Then for any sequence $\{Q^k\}_{k=0}^{K-1}$ and the corresponding BR sequence $\{\varepsilon_k^{BR}\}_{k=0}^{K-1}$ with $\varepsilon_k^{BR} = Q_k - T^{\pi_k} Q_k$ we have*

$$\|Q^* - Q^{\pi_K}\|_{1,\rho} \leq \frac{2\gamma}{(1-\gamma)^2} \Big[\inf_{r \in [0,1]} \sqrt{C_{PI,\rho,\nu}(K,r) \cdot \mathcal{E}(\varepsilon_0^{BR}, ..., \varepsilon_{K-1}^{BR}, r)} + \gamma^{K-1} R_{max} \Big] \tag{4.15}$$

where $\mathcal{E}(\varepsilon_0^{BR}, ..., \varepsilon_{K-1}^{BR}, r) = \sum_{k=0}^{K-1} a_k^{2r} \|\varepsilon_k^{BR}\|_{2,\nu}$ and

$$C_{PI,\rho,\nu}(K,r) = \Big(\frac{1-\gamma}{2}\Big)^2 \sup_{\pi_0,...,\pi_K} \sum_{k=0}^{K-1} a_k^{2(1-r)} \Big(\sum_{m \geq 0} \gamma^m \Big(c_{PI_1,\rho,\nu}(K-k-1, m+1, \pi_{k+1}) + c_{PI_1,\rho,\nu}(K-k, m, \pi_k) \Big) \Big)^2 \tag{4.16}$$

The proof of the theorem can be found in [36].

4.6 $API - BRM_\varepsilon$ Performance Loss

Finally using Theorem 4.3 on error propagation as well as the policy evaluation error given in Theorem 4.2 it is possible to derive an upper bound on the performance loss $\|Q^* - Q^{\pi_K}\|_{1,\rho}$ of $API - BRM_\varepsilon$. To prove the result we define the set of all policies that are greedy w.r.t. a member of $\mathcal{F}^{|A|}$ as $\hat{\Pi}(\mathcal{F}^{|A|}) = \{\hat{\pi}(\cdot, Q) : Q \in \mathcal{F}^{|A|}\}$.

Theorem 4.4. *(API − BRM$_\varepsilon$ **Performance Loss**) Let $\{Q^k\}_{k=0}^{K-1}$ be the solutions of the regularized BRM$_\varepsilon$ optimization problems given in section 4.9 for each API step k with the choice of $\lambda^k = \left(\frac{1}{n\|Q^{\pi_k}\|^2_{\mathcal{H}}}\right)^{\frac{1}{1+2\alpha}}$. If the assumption 4.3,..,assumption 4.5 hold for any $\pi \in \hat{\Pi}(\mathcal{F}^{|A|})$ and $\inf_{r\in[0,1]} C_{PI,\rho,\nu}(K,r) < \infty$. Then, there exists $C_{API-BRM_\varepsilon}(\delta,K,\rho,\nu)$ such that for $n \in \mathbb{N}$ and $0 < \delta < 1$ and n sufficiently large we have*

$$\|Q^* - Q^{\pi_K}\|_{1,\rho} \leq \frac{2\gamma}{(1-\gamma)^2}\left[C_{API-BRM_\varepsilon}(\delta,K,\rho,\nu)\, n^{-\frac{1}{2(1+2\alpha)}} + \gamma^{K-1}R_{max}\right] \tag{4.17}$$

with probability $1-\delta$

Proof. For each iteration $k = 0,...,K-1$ consider $0 < \delta < 1$ according to Theorem 4.2 and with confidence parameter $\delta_K = \delta/K$ we may upper bound the BR taking the supremum over all polices as

$$\|Q^k - T^{\pi_k}Q^k\|_{2,\nu} \leq \sup_{\pi\in\hat{\Pi}} c(\delta_K) n^{-\frac{1}{1+2\alpha}} \tag{4.18}$$

which holds with probability $1-\delta_K$. Hence for any $r \in [0,1]$ and using the definition of a_k putting $c'(\delta_K) = \sup_{\pi\in\hat{\Pi}} c(\delta_K)$ we have

$$\mathcal{E}(\varepsilon_0^{BR},...,\varepsilon_{K-1}^{BR},r) = \sum_{k=0}^{K-1} a_k^{2r}\|\varepsilon_k^{BR}\|_{2,\nu} \leq c^{`}(\delta_K) n^{-\frac{1}{1+2\alpha}}\left(\frac{1-\gamma}{1-\gamma^{K+1}}\right)^{2r}\frac{1-(\gamma^{2r})^K}{1-\gamma^{2r}} \tag{4.19}$$

Now if we apply Theorem 4.3 to propagate the error in the API procedure with probability at least $1-\delta$ we have

$$\|Q^* - Q^{\pi_K}\|_{1,\rho} \leq \frac{2\gamma}{(1-\gamma)^2}\left[C_{API-BRM_\varepsilon}(\rho,\nu,K) n^{-\frac{1}{1+2\alpha}} + \gamma^{K-1}R_{max}\right] \tag{4.20}$$

where

$$C_{API-BRM_\varepsilon}(\rho,\nu,K) = \sqrt{c'(\delta_K)} \inf_{r\in[0,1]} \left(\frac{1-\gamma}{1-\gamma^{K+1}}\right)^r \sqrt{C_{PI,\rho,\nu}(K,r)\frac{1-(\gamma^{2r})^K}{1-\gamma^{2r}}} \tag{4.21}$$

□

Theorem 4.2 upper bounds the performance loss showing the connection between the number of samples n, the capacity of the function space α, the number of PIs K and the other properties of the cMDP indicating that the upper bound is $O(n^{-\frac{1}{2(1+2\alpha)}})$. The rate of convergence of $API-BRM_\varepsilon$ has a behavior similar to the one obtained using Least Square Regression in [5]. Exponential β-mixing as well as the form of the loss function dependence considered in this work have little effect on the efficiency of learning. The term $C_{API-BRM_\varepsilon}$ contains the effect of the sampling distribution ν and the evaluation distribution ρ as well as the transition probability kernel of the cMDP itself on the performance loss.

4.7 Conclusion and Discussion

$API - BRM_\varepsilon$ algorithm estimates the action value functions by solving an optimization problems with regularized objective functions in some RKHS. The RKHS formulation has some advantages such as the ease of choosing the kernel function and consequently the function space. Hence, we focused on the statistical properties of $API - BRM_\varepsilon$ and provided an error upper bound on the performance loss of the resulting policy. The error bound showed the role of the sample size, complexity of function space and the intrinsic properties of cMDP such as the behavior of concentrability coefficients. As a result our work is the first attempt to give some theoretical justification of using the non-parametric SVR to solve the value function approximation problem in RL. Kernel methods using SVR where used in [13] to solve the cost-to-go RL problem with a batch and model based approach. However no statistical guarantees about the convergence was presented. Similarly [51] apply incremental SVR to the approximation of action value function in an actor-critic basis without PI. However the paper lacks a rigorous statistical analysis of their method. The methodology used to find the statistical bound has been adapted from [35] where squares losses and i.i.d. samples have been used to find the $API - BRM$ bound. The analysis contained in [35] introduces regularization based API algorithms analyzing their statistical properties providing upper bounds on the performance loss of the resulted policy compared to the optimal one. The same authors provide the analysis of the rate of convergence of the estimation error in regularized least-squares regression when the data is exponentially β-mixing in [5]. [6] also introduced an API-BRM procedure and studying its statistical properties considering parametric function spaces, which have finite effective dimension while our work as well as [35] consider non-parametric function spaces, which essentially are infinite dimensional. [56] also performed a finite-sample analysis studying the statistical analysis in the framework of BRM using linear function space under the simpler hypothesis of i.i.d. data samples obtained using a generative model (to cope with the double sampling problem). Other authors like [50] analyzed LSPI with linear function approximators and the statistical properties of LASSO-TD is analyzed although these results address different algorithms. [43] studied Least Squares SVM with solution restricted to deterministic problems. Their main contribution is the development of fast incremental algorithm even though it has limited practical application. They also use sparsification selecting a subset of data points based on some criteria and used to identify the basis functions. In fact it is known that Least Squares SVM differently than SVR, are not sparse. Sparsification then is necessary to control the complexity of the estimate but the effect on the generalization error is not well-understood. [92] unified several kernelized RL algorithms showing the equivalence of kernelized value function approximators with a model-based RL algorithm that has certain regularization on the transition kernel estimator, reward estimators, or both.

4.8 Future work

As possible future works we aim to follow some potential improvements of our method. In particular we aim to better analyze some of the assumptions we made in order to find the theoretical bound. From one side empirical processes and statistical learning theory with

dependent data allow us to find the theoretical bound involving the β-mixing scenario. However, presently there is no simple way to actually estimate those coefficients from data. While general functional forms are known for some common classes of processes specific coefficients are generally beyond calculation. Nevertheless following [58] we foresee to investigate the empirical process contained in the collected data checking for consistent estimators for the β-mixing coefficients based on a single stationary sample path. On the other hand to evaluate the bound we eventually asked for a stronger notion of approximation which need some assumptions on the probability distribution P assuming an inequality of the form

$$\begin{aligned} \|f_{D,\lambda_n} - f^*_{\ell_\varepsilon,P}\|_{2,\nu_x} &\leq c_P \Big[R_{\ell_\varepsilon,P}(f_{D,\lambda_n}) - R^*_{\ell_\varepsilon,P} \Big] \\ &= c_P \Big[\mathbb{E}[\ell_\varepsilon(y, f_{D,\lambda_n}(x))\,|\,Z_n] - \mathbb{E}[\ell_\varepsilon(y, f^*_{\ell_\varepsilon,P}(x))] \Big] \end{aligned} \tag{4.22}$$

Hence, as suggested in [82] we foresee to further investigate under which conditions on P the inequality section 4.8 holds when dealing with ℓ_ε losses. Finally, in this work we deal with continuous state and finite actions cMDPs which are essential ingredients into the asymptotic convergence bound. However, looking at performance obtained in several benchmarks, it seems to be possible to extend our result to cMDPs with continue action space. Further investigation in this sense might be promising and also interesting from a practical point of view.

4.9 Appendix 4A: Proof of Theorem 4.2

Proof. The proof partly follows the scheme of Theorem 5 in [5] which is similar to Theorem 21.1 in [40]. Essentially consists in decomposing the error into two terms T_{1n} and T_{2n} (defined later), using the minimizer property of the empirical risk minimizer to control T_{1n} and then applying the peeling device to analyze T_{2n}.

Consider the regularized $API-BRM_\varepsilon$ optimization problem

$$\hat{Q} = \arg\min_{Q\in\mathcal{H}} \{\hat{L}_{API-BRM_\varepsilon}(Q,\Pi_n,D_n) + \lambda_n \|Q\|^2_{\mathcal{H}}\} \tag{4.23}$$

Using the data sets Π_n, D_n the empirical estimate $\hat{L}_{API-BRM_\varepsilon}(Q,\Pi_n,D_n)$ can be written as:

$$\hat{L}_{API-BRM_\varepsilon}(Q,\Pi_n,D_n) = \mathbb{E}_n[\ell_\varepsilon(\hat{D}^\pi Q - \hat{r})] = \mathbb{E}_n[\ell_\varepsilon(Q - \hat{T}^\pi Q)] \tag{4.24}$$

and the regression function as $\hat{D}^\pi Q(s_t,a_t,s'_t) = Q(s_t,a_t) - \gamma \sum_{a'_t\in A} \pi(a'_t|s'_t) Q(s'_t,a'_t)$. We assume that $API-BRM_\varepsilon$ solution to be bounded and according to assumption 4.4. Now according to the error analysis for the $API-BRM_\varepsilon$ procedure presented in section 3.6 and considering assumption 4.5 we only have to bound the estimation error which can be controlled according to the following expressions

$$\Big| \mathbb{E}[\ell_\varepsilon(\hat{D}^\pi Q - \hat{r})\,|\,\Pi_n, D_n] - \mathbb{E}[\ell_\varepsilon(r-\hat{r})] \Big| \leq \|Q - T^\pi Q\|_{1,\nu} \tag{4.25}$$

and if we meet the condition for the section 3.4 using

$$\|Q - T^\pi Q\|_{2,\nu} \leq c_P \Big[\mathbb{E}[\ell_\varepsilon(\hat{D}^\pi Q - \hat{r})|\Pi_n, D_n] - \mathbb{E}[\ell_\varepsilon(r-\hat{r})] \Big] \tag{4.26}$$

In any case the policy evaluation error relies on the expression

$$\mathcal{E}_n^{\pi}(\hat{Q}) = \mathbb{E}[\ell_{\varepsilon}(\hat{D}^{\pi}\hat{Q} - \hat{r})|\Pi_n, D_n] - \mathbb{E}[\ell_{\varepsilon}(r - \hat{r})]. \tag{4.27}$$

Define the following error decomposition:

$$\mathcal{E}_n^{\pi}(\hat{Q}) = T_{1n} + T_{2n} \tag{4.28}$$

where

$$\frac{1}{2}T_{1n} = \mathbb{E}_n[\ell_{\varepsilon}(\hat{D}^{\pi}\hat{Q} - \hat{r})] - \mathbb{E}_n[\ell_{\varepsilon}(r - \hat{r})] + \lambda_n\|\hat{Q}\|_{\mathcal{H}}^2 \tag{4.29}$$

$$T_{2n} = \mathbb{E}[\ell_{\varepsilon}(\hat{D}^{\pi}\hat{Q} - \hat{r})|\Pi_n, D_n] - \mathbb{E}[\ell_{\varepsilon}(r - \hat{r})] - T_{1n} \tag{4.30}$$

In this way we may try to control the estimation error working on T_{1n} and T_{2n}. Now we may use use the minimizer property of $\hat{Q}$ and the fact that for the action value function we have $Q^{\pi} = T^{\pi}Q^{\pi}$ and also $D^{\pi}Q^{\pi} = Q^{\pi} - T^{\pi}Q^{\pi} + r = r$. According to the assumption 4.4 we have $|\hat{r}| \leq R_{max}$ and $|Q| < Q_{max}$ for any $Q \in \mathcal{H}$. This imply that we may write

$$\begin{aligned} &\mathbb{E}_n[\ell_{\varepsilon}(\hat{D}^{\pi}\hat{Q} - \hat{r})] - \mathbb{E}_n[\ell_{\varepsilon}(r - \hat{r})] + \lambda_n\|\hat{Q}\|_{\mathcal{H}}^2 \leq \\ &\mathbb{E}_n[\ell_{\varepsilon}(D^{\pi}Q^{\pi} - \hat{r})] - \mathbb{E}_n[\ell_{\varepsilon}(r - \hat{r})] + \lambda_n\|Q^{\pi}\|_{\mathcal{H}}^2 \end{aligned} \tag{4.31}$$

suggesting that T_{1n} may be controlled by

$$\frac{1}{2}T_{1n} \leq \mathbb{E}_n[\ell_{\varepsilon}(D^{\pi}Q^{\pi} - \hat{r})] - \mathbb{E}_n[\ell_{\varepsilon}(r - \hat{r})] + \lambda_n\|Q^{\pi}\|_{\mathcal{H}}^2 \tag{4.32}$$

and considering that $D^{\pi}Q^{\pi} = r$ we have $T_{1n} \leq 2\lambda_n\|Q^{\pi}\|_{\mathcal{H}}^2$.

Now to bound T_{2n} fix a number t satisfying $t \geq \frac{1}{n}$ and the goal is to study $\mathbb{P}\{T_{2n} > t\}$. We have

$$\begin{aligned} \mathbb{P}\{T_{2n} > t\} = \mathbb{P}\Big\{ &2(\mathbb{E}[\ell_{\varepsilon}(\hat{D}^{\pi}\hat{Q} - \hat{r})|\Pi_n, D_n] - \mathbb{E}[\ell_{\varepsilon}(r - \hat{r})]) \\ &-2\mathbb{E}_n[\ell_{\varepsilon}(\hat{D}^{\pi}\hat{Q} - \hat{r}) - \ell_{\varepsilon}(r - \hat{r})] \\ &> t + 2\lambda_n\|\hat{Q}\|_{\mathcal{H}}^2 + \mathbb{E}[\ell_{\varepsilon}(\hat{D}^{\pi}\hat{Q} - \hat{r})|\Pi_n, D_n] - \mathbb{E}[\ell_{\varepsilon}(r - \hat{r})]\Big\} \end{aligned} \tag{4.33}$$

Let $w = (s, a, r, s')$ and for a fixed policy define the function $g : S \times A \times \mathbb{R} \times S \to \mathbb{R}$ as

$$g(w) = \left[\ell_{\varepsilon}\Big(Q(s,a) - \gamma\sum_{a'}\pi(s',a')Q(s',a') - \hat{r}\Big) - \ell_{\varepsilon}\Big(r - \hat{r}\Big)\right]$$

Hence the function space for $l = 0, 1, ...$

$$\mathcal{G}_l = \Big\{ g : S \times A \times \mathbb{R} \times S \to \mathbb{R}, Q \in \mathcal{H}, \|Q\|_{\mathcal{H}}^2 \leq \tfrac{2^l t}{\lambda_n} \Big\} \tag{4.34}$$

Note that the functions in $\mathcal{G}_l$ satisfy $\|g\|_{\infty} \leq K_1 = 2Q_{max}$. Now consider a stochastic process involving the random variables $w_t = (s_t, a_t, r_t, s_t') \in \mathcal{W}$. Applying the peeling device we get

$$\mathbb{P}\{T_{2n} > t\} \leq \sum_{l=0}^{\infty}\mathbb{P}\Big\{ \sup_{g\in\mathcal{G}_l} \tfrac{\mathbb{E}[g(W)|\Pi_n, D_n] - \mathbb{E}_n[g(W)]}{2^l t + \mathbb{E}[g(W)|\Pi_n, D_n]} > \tfrac{1}{2} \Big\} \tag{4.35}$$

Here we used the fact that $g \in \mathcal{G}_l$ so we have $\|Q\|_{\mathcal{H}}^2 \leq \frac{2^l t}{\lambda_n}$.

We want to study the behavior of the l^{th} term of the above summation by verifying the condition of Theorem 4.1. Hence we have to select an independent block sequence tuned separately for each value of l. If we fix some value $l \in \mathbb{N}_0$ and let the block size and the number of blocks defined according to:

$$a_{n,l} = \lfloor a'_{n,l} \rfloor \quad \mu_{n,l} = \lfloor \frac{n}{2a_{n,l}} \rfloor \tag{4.36}$$

assuming for certain $\gamma, p > 0$ that we have

$$a'_{n,l} = (nt)^\gamma (2^l)^p \quad \mu'_{n,l} = \frac{n}{2a'_{n,l}} = \frac{n^{1-\gamma}}{2t^\gamma (2^l)^p} \tag{4.37}$$

As by assumption results $t \geq \frac{1}{n}$ and $\gamma, p > 0$ we have $a_{n,l} \geq 1$. Now let R_l be the residual block in the $(a_{n,l}, \mu_{n,l})-$partitioning of $\{1, ..., n\}$. The block size $a_{n,l}$ and the residual block size $|R_l|$ have the properties:

$$n - |R_l| = 2a_{n,l}\mu_{n,l} \leq n \quad |R_l| < 2a_{n,l} \quad \mu'_{n,l} \leq \mu_{n,l} \tag{4.38}$$

If we analyze the summand expression in section 4.9 assuming $\gamma \leq p$ and $4nK_1 \leq (a'_{n,l})^{1/p}$ hold then

$$\frac{\mathbb{E}[g(W)|\Pi_n, D_n] - \mathbb{E}_n[g(W)]}{2^l t + \mathbb{E}[g(W)|\Pi_n, D_n]} \leq \frac{1}{2} \tag{4.39}$$

In particular results

$$\frac{\mathbb{E}[g(W)|\Pi_n, D_n] - \mathbb{E}_n[g(W)]}{2^l t + \mathbb{E}[g(W)|\Pi_n, D_n]} \leq \frac{2K_1}{2^l t} \tag{4.40}$$

Considering that $t \geq \frac{1}{n}$ and $\gamma \leq p$ we get $a'_{n,l} = (nt)^\gamma (2^l)^p \leq (nt2^l)^p$ which is equivalent to $n^{-1}(a'_{n,l})^{1/p}$ which combined with $4nK_1 \leq (a'_{n,l})^{1/p}$ gives the result. The expression $4nK_1 \leq (a'_{n,l})^{1/p}$ follows from the following assumptions:

$$p \leq \frac{1}{2} \leq 1 \quad a'_{n,l} \geq \frac{n}{8} \quad n \geq c_1 = 48^2 K_1 \geq 4^{\frac{p}{1-p}} 8^{\frac{1}{1-p}} K_1^{\frac{p}{1-p}} \tag{4.41}$$

and we may assume that conditions section 4.9 holds together with $t \geq \frac{1}{n}$. This help the analysis because it suffices to study the case when l is such that $a_{n,l} \leq n/8$. Hence following Proposition 6 in [5] we may prove the result:

Proposition 4.1. *Consider l such that $a_{n,l} \leq n/8$ and assume*

$$0 < \gamma < p \leq \frac{1}{2 + 6\alpha} \tag{4.42}$$

Then there exists constants $c_3, c_4 > 1$ and $c_5 > 0$ depending only on Q_{max} such that for any

$$t > c_3^{\frac{1}{1-\gamma(2+6\alpha)}} \frac{1}{n\lambda_n^{\frac{2\alpha}{1-\gamma(2+6\alpha)}}} + \frac{c_4}{n} \tag{4.43}$$

we have

$$\mathbb{P}\Big\{ \sup_{g\in\mathcal{G}_l} \frac{\mathbb{E}[g(W)|\Pi_n, D_n] - \mathbb{E}_n[g(W)]}{2^l t + \mathbb{E}[g(W)|\Pi_n, D_n]} > \frac{1}{2} \Big\} \leq 120 \exp\Big(- c_5 \frac{\mu_{n,l}^2 2^l t}{n} \Big) + \beta_{a_{n,l}} \mu_{n,l} \quad (4.44)$$

Proof of this Proposition can be found in Appendix 4.10.

We apply the last proposition to the terms of the RHS of expression section 4.9 when l is such that $a_{n,l} < n/8$. Remembering the conditions $t \geq \frac{1}{n}$ and conditions section 4.9 as well as proposition 4.1 we may write

$$\begin{aligned} \mathbb{P}\{T_{2n} > t\} &\leq \sum_{l\in\mathbb{N}_0,\, a_{n,l}<\frac{n}{8}} \Big[120 \exp\Big(- c_5 \tfrac{\mu_{n,l}^2 2^l t}{n} \Big) + \beta_{a_{n,l}} \mu_{n,l} \Big] \\ &\leq \sum_{l\in\mathbb{N}_0} \Big[120 \exp\Big(- c_5 \tfrac{\mu_{n,l}^2 2^l t}{n} \Big) + \beta_{a_{n,l}} \mu_{n,l} \Big] \end{aligned} \quad (4.45)$$

As it is necessary to bound $\beta_{a_{n,l}} \mu_{n,l}$ fix some $l \geq 0$ and using assumption 4.2 we have

$$\beta_{a_{n,l}}\, \mu_{n,l} \leq \bar{\beta}_0 \exp\left(-\bar{\beta}_1 a_{n,l} + \log\, \mu_{n,l}\right) \quad (4.46)$$

and if $\frac{\log \mu_{n,l}}{\bar{\beta}_1 a_{n,l}} < \frac{1}{2}$ holds we have $2\beta_{a_{n,l}} \mu_{n,l} \leq 2\bar{\beta}_0 \exp(-\frac{\bar{\beta}_1}{2} a_{n,l}) \leq c_6 \exp -\frac{\bar{\beta}_1}{2} a'_{n,l}$ where $c_6 = 2\bar{\beta}_0 \exp(\frac{\bar{\beta}_1}{2})$. Using the fact that $a'_{n,l} \leq 2a_{n,l}$ and $\mu_{n,l} \leq n$ as well as the definition of $a'_{n,l}$ condition section 4.9 is satisfied whenever

$$t > \frac{(\frac{4 \log n}{\bar{\beta}_1})^{\frac{1}{\gamma}}}{n} \quad (4.47)$$

Hence we have

$$\mathbb{P}\{T_{2n} > t\} \leq \sum_{l\geq 0} \Big[c_7 \exp\Big(- c_5 \tfrac{\mu_{n,l}^2 2^l t}{n} \Big) + c_6 \exp -\tfrac{\bar{\beta}_1}{2} a'_{n,l} \Big] \quad (4.48)$$

which substituting the expression of $\mu_{n,l}$ can be bounded as

$$\mathbb{P}\{T_{2n} > t\} \leq c_9 \exp(-c_8 (nt)^{1-2\gamma}) + c_{10} \exp(-c_{11} \bar{\beta}_1 (nt)^{\gamma}) \quad (4.49)$$

fixing $0 < \delta < 1$ we may invert section 4.9 which gives that if $t > \frac{1}{n}$ and also satisfies proposition 4.1 and section 4.9 with holding conditions proposition 4.1 and section 4.9 then we may conclude

$$T_{2n} \leq \frac{1}{n} \Big[\Big(\frac{\log(\frac{2c_{10}}{\delta})}{c_{11}\bar{\beta}_1} \Big)^{\frac{1}{\gamma}} + \Big(\frac{\log(\frac{2c_9}{\delta})}{c_8} \Big)^{\frac{1}{1-2\gamma}} \Big] \quad (4.50)$$

with probability $1 - \delta$. As a final step of the proof we may analyze the error decomposition combining conditions proposition 4.1 and section 4.9 and redefining in a suitable way the constants we get

$$\begin{aligned} \mathcal{E}_n^{\pi}(\hat{Q}) = T_{1n} + T_{2n} &\leq 2\lambda_n \|Q^{\pi}\|_{\mathcal{H}}^2 + c_2^{\frac{1}{1-\gamma(2+6\alpha)}} \frac{1}{n\lambda_n^{\frac{2\alpha}{1-\gamma(2+6\alpha)}}} \\ &+ \frac{\left(\frac{c_3}{\bar{\beta}_1} \ln \frac{c_7}{\delta}\right)^{\frac{1}{\gamma}}}{n} + \frac{\left(\frac{c_4}{\bar{\beta}_1} \log n\right)^{\frac{1}{\gamma}}}{n} + \frac{\left(\frac{c_5}{\bar{\beta}_1} \ln \frac{c_7}{\delta}\right)^{\frac{1}{1-2\gamma}}}{n} + \frac{c_6}{n} \end{aligned} \quad (4.51)$$

holding with probability at least $1-\delta$

Now let us assume that $0<\gamma\leq\frac{1}{8}<\frac{1}{2+6\alpha}$ and in this range of γ as n gets large the third element of the RHS of section 4.9 dominates the last two terms and it is only necessary to deal only with the first four terms. It is possible to investigate that the choice of λ_n which minimizes the sum of these terms disregarding the constants can be expressed as:

$$\lambda_n=\Big[\frac{1}{n\,\|Q^\pi\|^2_{\mathcal{H}}}\Big]^{\frac{1-\gamma(2+6\alpha)}{1-\gamma(2+6\alpha)+2\alpha}} \tag{4.52}$$

making the sum of the firsts terms proportional to

$$\lambda_n\|Q^\pi\|^2_{\mathcal{H}}=\frac{\Big[n\,\|Q^\pi\|^2_{\mathcal{H}}\Big]^{\frac{\alpha}{1-\gamma(2+6\alpha)+2\alpha}}}{n}=\frac{\exp\big(\frac{\alpha}{1-\gamma(2+6\alpha)+2\alpha}B\big)}{n} \tag{4.53}$$

where $B=\log\big(n\|Q^\pi\|^2_{\mathcal{H}}\big)$. Looking at section 4.9 the sum of the third and forth terms is upper bounded by a constant multiple of $\frac{\exp(A/\gamma)}{n}$ where $A=\log\big(\frac{c_8}{\bar{\beta}_1}\log\big(\frac{c_7}{\max(\delta,n)}\big)\big)$. In order to choose γ we have to study two different cases for $\alpha\geq 0$. The case $\alpha=0$ imply that being $\lambda_n\|Q^\pi\|^2_{\mathcal{H}}=1/n$ the best choice for $\gamma\in(0,\frac{1}{3}]$ is $\gamma=\frac{1}{3}$. In fact $\frac{\exp(A/\gamma)}{n}$ is decreasing in γ. When $A>0$ the dominating term in the bound is $\frac{\exp(A/\gamma)}{n}$ and a suitable choice for $p=\frac{1}{2}$. The case for $\alpha>0$ brings to the solution of the minimization of

$$\frac{1}{n}\Big(\exp(A/\gamma)+\exp\big(\frac{\alpha}{1-\gamma(2+6\alpha)+2\alpha}B\big)\Big) \tag{4.54}$$

which can obtained solving

$$A/\gamma=\frac{\alpha}{1-\gamma(2+6\alpha)+2\alpha}B \tag{4.55}$$

bringing to

$$\gamma=\frac{(1+2\alpha)A}{\alpha B+(2+6\alpha)A} \tag{4.56}$$

A suitable choice can be $\gamma\leq\frac{1}{3}$ where the term under investigation becomes

$$\begin{aligned}&\tfrac{1}{n}\exp(A/\gamma)=\tfrac{1}{n}(\exp(B))^{\frac{\alpha}{1+2\alpha}}(\exp(A))^{\frac{2+6\alpha}{1+2\alpha}}\\&=(\|Q^\pi\|^2_{\mathcal{H}})^{\frac{\alpha}{1+2\alpha}}n^{-\frac{1}{1+2\alpha}}\Big(\log\big(\tfrac{c_8}{\bar{\beta}_1}\log\big(\tfrac{c_7}{\max(\delta,n)}\big)\big)\Big)^{\frac{2+6\alpha}{1+2\alpha}}\end{aligned} \tag{4.57}$$

and as n gets larger $\gamma\to 0$. Moreover $\gamma>0$ when $A,B>0$ which are satisfied if

$$n\geq\max\big(\exp(\bar{\beta}_1/c_8),\big)1/\|Q^\pi\|^2_{\mathcal{H}}$$

Any p such that $0<\gamma<p<\frac{1}{2+6\alpha}$ satisfies all conditions only affecting the constants and this terminates the proof. □

4.10 Appendix 4B: Proof of Proposition 4.1

Proof. In order to prove the theorem we need to verify that the conditions of Theorem 4.1 apply with the choice $\varepsilon = \frac{1}{2}$ and $\eta = 2^l t$. Firstly conditions $\mathbf{C_1}$ and $\mathbf{C_2}$ are verified with choice $K_1 = K_2 = 2Q_{max}$. For conditions $\mathbf{C_3}$ since $Q_{max} \geq 1$ we have $a_{n,l} \geq 1$ which imply that $2K_1 a_{n,l} > \sqrt{2a_{n,l}K_2}$. Therefore it is enough to verify that $\sqrt{n}\varepsilon\sqrt{1-\varepsilon}\sqrt{\eta} \geq 1152K_1 a_{n,l}$. Considering that $a_{n,l} \leq a'_{n,l}$ suffices to verify the condition using $a'_{n,l}$. Putting this definition means that condition $\mathbf{C_3}$ becomes $\sqrt{n}\varepsilon\sqrt{1-\varepsilon}\sqrt{\eta} \geq 1152K_1(nt)^{\gamma}(2^l)^p$ can be satisfied when $t \geq \frac{c'_2}{n}$ for some $c'_1 > 0$ depending on Q_{max}. Regarding condition $\mathbf{C_4}$ by construction we have $|R_l| < 2a_{n,l} \leq 2a'_{n,l}$ and considering the conditions on γ, p results $\frac{2a'_{n,l}}{n} < \frac{2^l t}{12K_1}$ which can be satisfied by some $t \geq \frac{c'_2}{n}$ for some $c'_2 > 0$ depending on Q_{max}. Moreover we have by assumption that $a_{n,l} < \frac{n}{8}$ and $|R_l| < 2a_{n,l}$ thus results $|R_l| < \frac{n}{4}$. Finally we have to verify that the condition $\mathbf{C_5}$ hold in the form: for all $w_1, ..., w_n \in \mathcal{W}$ and all $\delta > \frac{2^l t a_{nl}}{8}$ we have

$$\frac{\sqrt{\mu_{n,l}}\varepsilon(1-\varepsilon)\delta}{96\sqrt{2}a_{n,l}\max\{K_1, 2K_2\}} \geq \int_{\frac{\varepsilon(1-\varepsilon)\delta}{16a_{n,l}\max\{K_1,2K_2\}}}^{\sqrt{\delta}} \log \mathcal{N}_1\Big(\frac{u}{4a_{n,l}}, \mathcal{G}_l, \|w\|_{1,n}\Big) du$$

Now let $z = (s,a)$ and $z'_j = (s', a'_j)$ consider the function space

$$\mathcal{F}_l = \{ Q \in \mathcal{H} : \|Q\|^2_{\mathcal{H}} \leq \frac{2^l t}{\lambda_n} \}$$

as results

$$|g_1(w) - g_2(w)| = |\ell_\varepsilon(\hat{Q}_{r1}(z,z') - \hat{r}) - \ell_\varepsilon(\hat{Q}_{r2}(z,z') - \hat{r})| \leq |\hat{Q}_{r1}(z,z') - \hat{Q}_{r2}(z,z')|$$

where we used the fact $\hat{Q}_r(z,z') = Q(z) - \gamma \sum_{z'_j} Q(z'_j) \quad j = 1, ..., |A|$ then we have

$$\mathcal{N}_1(u, \mathcal{G}_l, \|w\|_{1,n}) \leq \mathcal{N}_1^{|A|}(u, \mathcal{F}_l, \|z\|_{1,n}) \times \prod_{j=1}^{|A|} \Big[\mathcal{N}_1^{|A|}(u, \mathcal{F}_l, \|z_j^{`}\|_{1,n})\Big]$$

and according to assumption 4.3 we may conclude that

$$\log \mathcal{N}_1(\frac{u}{4a_{n,l}}, \mathcal{G}_l, \|w\|_{1,n}) \leq c(|A|)(a_{n,l})^{2\alpha}\Big(\frac{2^l t}{\lambda_n}\Big)^{\alpha} u^{-2\alpha}$$

and we may bound the condition using $c'_3(a_{n,l})^{2\alpha}(\frac{2^l t}{\lambda_n})^{\alpha}\delta^{\frac{1-2\alpha}{2}}$ for some constant $c'_3 > 0$. Now to verify condition $\mathbf{C_5}$ noting that $\mu_{n,l} \geq \mu'_{n,l}$ and also that we have to assume that $\delta \geq \frac{2^l t a_{n,l}}{8}$ it suffices to show that

$$\frac{\sqrt{\mu'_{n,l}}\delta}{a_{n,l}} \geq c'_4(a_{n,l})^{2\alpha}\Big(\frac{2^l t}{\lambda_n}\Big)^{\alpha}\delta^{\frac{1-2\alpha}{2}}$$

This condition can be satisfied whenever $t \geq c'_5 \frac{(a_{n,l})^{6\alpha+1}}{\mu'_{n,l} 2^l \lambda_n^{2\alpha}}$ for a give $c'_5 > 0$. Using $a'_{n,l} \geq a_{n,l}$ and $\mu'_{n,l} = \frac{n}{2a'_{n,l}}$ and $a'_{n,l} = (nt)^\gamma (2^l)^p$ one may show that it suffices to have

$$t \geq c'_6 \frac{1}{n(\lambda_n)^{\frac{2\alpha}{1-\gamma(2+6\alpha)}} (2^l)^{\frac{1-p(2+6\alpha)}{1-\gamma(2+6\alpha)}}}$$

Now using the assumption that $\gamma < p \leq \frac{1}{2+6\alpha}$ to get a non decreasing function on $(2^l)^{\frac{1-p(2+6\alpha)}{1-\gamma(2+6\alpha)}}$ the entropy condition can be satisfied if

$$t \geq c'_7 \frac{1}{n(\lambda_n)^{\frac{2\alpha}{1-\gamma(2+6\alpha)}}}$$

then by choosing the appropriate constants the conditions of Theorem 4.1 are satisfied. Therefore plugging all we got into theorem 4.1 of Theorem 4.1 and considering $a_{n,l}\mu_{n,l} \leq \frac{n}{2}$ and also that $Q_{max} \geq 1$ we have

$$\mathbb{P}\Big\{ \sup_{g \in \mathcal{G}_l} \frac{\mathbb{E}[g(W)|\Pi_n, D_n] - \mathbb{E}_n[g(W)]}{2^l t + \mathbb{E}[g(W)|\Pi_n, D_n]} > \frac{1}{2} \Big\} \leq 120 \exp\Big(- c_5 \frac{\mu_{n,l}^2 2^l t}{n} \Big) + \beta_{a_{n,l}} \mu_{n,l} \tag{4.58}$$

for $c_5 > 0$ which finally proves the theorem. □

Chapter 5

$API - BRM_\varepsilon$ Experimental Analysis

5.1 Introduction

Parametric approximators need Basic Functions (BFs) to build their solution while non-parametric approximators instead are automatically constructed from the data without the need of designing the BFs. Among other non-parametric approximators, kernel based are computationally demanding with cost growing with the number of samples. In practice since this number can be large, many approaches employ sparsification techniques to limit the number of samples contributing to the solution. However using $\varepsilon-$insensitive loss function with a given kernel allows to build intrinsically sparse approximator in a RKHS. Sparsity directly depends on the combination of kernel and loss function parameters. Hence one aspect that is needed to be discussed is the computational complexity of our kernel method included in $API - BRM_\varepsilon$. In the previous chapter we extensively analyze the statistical properties of $API - BRM_\varepsilon$ founding a bound which controls the performance error of the algorithm while considering the batch formulation of the problem. $API - BRM_\varepsilon$ algorithm eventually converges to the optimal policy using β-mixing distributed data samples.

5.2 $API - BRM_\varepsilon$ Computational Complexity

SVM are powerful tools, but their compute and storage requirements increase rapidly with the number of training vectors. Solving SVM relies on Quadratic Programming (QP) optimization, which is able to separate support vectors from the rest of the training data. For our SVR implementation test having an optimal solution involves $O(n^2)$ dot products, while solving the QP problem directly involves inverting the kernel matrix, which has complexity $O(n^3)$ where n is the size of the training set. Hence the computational complexity of a K-iteration $API - BRM_\varepsilon$ algorithm is dominated by three factors: first computing Gram matrix of the Bellman kernel $\tilde{\kappa}^\pi(z_i, z_j)$ for each pair $z_i = (s_i, a_i)$, $z_j = (s_j, a_j)$, second solving the regularized regression problem and third the number of PIs steps used by the algorithm. Computing $\tilde{\kappa}^\pi$ for each pair involves enumerating each successor state of both s_i and s_j and evaluating the base kernel $\kappa(\cdot,\cdot)$ for each pair of successor states. Empirically me may define the average branching factor of a finite state MDP $\hat{\beta}$ as the average number of possible successor states for any state $s \in S$ which in

the limit of continuous state space can be assumed $\hat{\beta}=1$. Equivalently given the dataset D_n and an estimation of the transition kernel $P^{s'}_{s,a}$, the branching factor $\hat{\beta}$ is the average number of terms $P^{s'}_{s,a}$ that are non zero given (s,a). Assuming we have a n samples data set with an cMDP that empirically presents an average branching factor $\hat{\beta}$ each state, computing a single element of the Gram matrix requires $O(\hat{\beta}^2)$ operations. Since the Gram matrix has dimension $n\times n$ but is symmetric there are $n(n+1)/2$ unique elements that must be computed. Therefore, the total number of operations to compute the full Gram matrix is $O(kernel)=O(\hat{\beta}^2 n(n+1)/2)$. Once the Gram matrix is constructed in principle its inverse must be evaluated. Since the Gram matrix is positive definite and symmetric, Cholesky decomposition might be used, resulting in a total complexity of $O(n^3)$. As a result, the total complexity of the BRM_ε algorithm is $O(n^3+\hat{\beta}^2 n(n+1)/2)$ and with K PIs steps we assume a complexity $O(K(n^3+\hat{\beta}^2 n(n+1)/2))$. This is clearly a pessimistic result and thanks to the sparsity property of SVR, assuming an average number of support vectors $n_{sv}\ll n$ one may obtain a posterior complexity $O(K(n_{sv}^3+\hat{\beta}^2 n_{sv}(n_{sv}+1)/2))$. For the incremental implementation of $API-BRM_\varepsilon$ if we consider the worst case, to add a new sample we need $O(n^3)\cdot O(kernel)$ when all the training samples are support vectors [49], [57]. On average the algorithm has complexity $O(n^2)$ and for the incremental $API-BRM_\varepsilon$ we may assume $O(K(n^2+\hat{\beta}^2 n(n+1)/2)))$ as complexity bound with an optimist posterior bound $O(K(\hat{n}_{sv}^2+\hat{\beta}^2\hat{n}_{sv}(\hat{n}_{sv}+1)/2)))$ with $\hat{n}_{sv}$ the average number of support vectors in K PI.

5.3 $API-BRM_\varepsilon$ Algorithm Implementation

Speed of learning depends mostly on the number of support vectors, influencing significatively the performance. This is the first reason to implement an incremental version of the $API-BRM_\varepsilon$ algorithm. Another reason comes from the fact that this version of the algorithm can be easily implemented in an online setting whenever the agent may interact with the environment. After each incremental step (or at least after some of them) which allows to implement a policy evaluation updating the approximation of the value function, one might perform the policy improvement updating the policy. In fact, differently from the offline case where only the final performance matters, in online learning the performance should improve once every few transition samples. PI can take this requirement into account by performing policy improvements once every few transition samples, before an accurate evaluation of the current policy can be completed. In the practical implementation of online $API-BRM_\varepsilon$, by using the current behavior policy the algorithm collects its own samples interacting with the system. As a consequence some exploration has to be added to the policy which becomes soft. As already mention an $\varepsilon-$greedy policy is a soft policy which for some $0\leq\varepsilon\leq 1$ picks deterministically a particular action with probability $1-\varepsilon$ and a uniformly random action with probability ε. Hence policy improvements have to be implemented without waiting for the action value function estimates to get close to their asymptotic values for the current policy. Henceforth these estimates have to be updated continuously without reset after some policy changes as this in practice corresponds to having multiple policies. In principle one may assume that value function estimates remain similar for subsequent policies or al least do not change

too much. Another possibility would be to rebuild the action value function estimates from the scratch before every update (some sort of purge for the SVR). Unfortunately this alternative can be computationally costly and might not be necessary in practice. The number of transitions between consecutive policy improvements is a crucial parameter of the algorithm and should not be too large, to avoid potentially bad policies from being used too long. Given the policy π the cMDP reduces to a Markov chain $\mathcal{M}^\pi = (S, R^\pi, P^\pi)$ with reward function $R^\pi = r(s, \pi(s))$ and transition kernel $P^\pi(\cdot|s) = \mathcal{P}(\cdot|, s, \pi(s))$ with a stationary distribution ρ if it admits one. Hereafter we make the assumption that the policy π induces a stationary β-mixing process on the cMDP with a stationary distribution ν. The fast β-mixing process starts from an arbitrary initial distribution and following the current policy, the state probability distribution rapidly tends to the stationary distribution of the Markov chain. In particular, we consider the case in which the samples $\{s_1, .., s_n\}$ are obtained by following a single trajectory in the stationary regime $\mathcal{M}^\pi$ considering that s_1 is drawn from ρ (see [50] for an insightful discussion on the subject). In fact, to guarantee the convergence of the online version of $API - BRM_\varepsilon$ we additionally require that the samples follow the stationary distribution over state-action pairs induced by the policy considered ρ. Intuitively this means that the weight of each state-action pair (s, a) is equal to the steady-state probability of this pair along an infinitely-long trajectory generated with the policy π. Moreover assuming that the distribution ρ is stationary the resulting Bellman equation has a unique solution as the Bellman operator is a contraction and it admits one unique fixed point. Consider now the incremental online $API - BRM_\varepsilon$ algorithm, the performance guarantees rely on small policy evaluation errors assuming that the policy is improved before an accurate value function is available. In practice this means that the policy evaluation error can be very large and this might affects the performance. Nevertheless, the algorithm works fine in practice for several standard RL benchmarks. In chapter 4 we provide a finite sample bound of the performance loss $\|Q^{\pi_K} - Q^*\|^q_{p,\rho}$ where π_K is the greedy policy w.r.t. Q^{K-1} after K PIs as depicted in Algorithm 1 and ρ is the performance evaluation measure. The analysis proceeds studying the policy evaluation error of $API - BRM_\varepsilon$ using the SVR regularization scheme where we suppose that given any policy π one may obtain Q by solving the regularized problem section 3.7 with a given π_k at step k of the API procedure.

5.4 $API - BRM_\varepsilon$ Experiments

$API - BRM_\varepsilon$ algorithm was implemented using a combination of Matlab and C routines and was tested on the following standard RL benchmarks: chain walk, inverted pendulum, cart pole, hill car, bicycle balancing and riding. The simulation is implemented using a generative model capable of simulating the environment while the learning agent is represented by the $API - BRM_\varepsilon$ algorithm. The chain walk class of problems is useful because allows to compare the approximations learned by $API - BRM_\varepsilon$ to the true action value functions. The domains are standard benchmark in RL literature featuring continuous state spaces and non-linear dynamics. Moreover in our experiments, we compare performance of $API - BRM_\varepsilon$ algorithm with other learning methods such as Q-learning or the parametric linear approximation architecture of LSPI algorithm implementations for offline [48] and online [19]. To rank performance it is necessary to introduce some metrics

measuring the quality of the solutions. In each benchmark a specific goal is foreseen and a performance measure is represented by the fulfillment of a given task. For example for the inverted pendulum, and the bicycle domains we may assess the quality of a policy through its ability to avoid crashing during a certain period of time. To measure the quality of a solution we can use the stationary policy it produces, compute the expected return and say that the higher this expected return is, the better the RL algorithm performs. This can be done defining a set of initial states S_0 where we compute the average expected return of the stationary policy chosen independently from the set of tuples D_n. Such kind of metric is called as the score of a policy (see [26] for more details). Given the learned policy $\hat{\pi}$ its score is defined by $Score_{\hat{\pi}} = \frac{\sum_{s_0 \in S_0} \hat{R}^{\hat{\pi}}(s_0)}{|S_0|}$ where $\hat{R}^{\hat{\pi}}(s_0)$ is the empirical estimate of $R^{\hat{\pi}}(s) = \mathbb{E}[\sum_{t=0}^{n-1} \gamma^t r(s_t, \hat{\pi}(s_t))|s_0 = s]$ the average return. In order to evaluate the score one has to estimate the average empirical return for every initial state $s_0 \in S_0$ by Monte-Carlo simulations. As we consider all the benchmarks non deterministic the average of the score in more than 10 different simulations. Another important aspect to keep in mind is the rather large flexibility of our method which is based upon the generalization ability of SVR and the use of incrementality. Generalization relies on the statistical properties of the structural risk minimization of SVR and the use of a suitable kernel function. In all our experiments for any pair of $z_i = (s_i, a_i)$ and $z_j = (s_j, a_j)$ we use the RBF kernel $\kappa(z_i, z_j) = e^{-\frac{1}{2}(z_i - z_j)^T \Sigma^2 (z_i - z_j)}$ where Σ is a diagonal matrix specifying the weight for any state-action vector component. Using this kernel also allows to manage possible variant of the problems where the action space may be considered continuous or eventually noisy. Even though we studied the statistical properties using finite action spaces, in practice the algorithm may also works fine using a continuous action space. A part from the matrix Σ we also have to define the SVR parameters (C, ε). We performed an grid search to find the appropriate set of parameters (Σ, C, ε) looking at the resulting performance of the learning system. In fact, using different set of parameters might help finding near-optimal policies whose performance can be measured using the score or their ability to reach the goal. Finally, another important aspect which may affect the performance of the algorithm is represented by the way we collect data and therefore how we manage the compromise between the need of exploration and exploitation of the learned policy. Thanks to the flexibility of our method, we may run experiments using three different methods:

1. Method-1 (offline $API-BRM_\varepsilon$) data are generated offline using a random behavior policy which produces a set D_n of tuples i.i.d. using eventually a set of different initial states S_0 or a fixed one s_0. Hence $API-BRM_\varepsilon$ algorithm learns a policy processing the collected data batch or incrementallly. In this case exploration relies on the stochastic initial state set and behavior policy used while exploitation is performed in the learning stage. In principle it is also possible to generate a new set of data in each PI.

2. Method-2 (online $API-BRM_\varepsilon$) some data are generated offline using a random behavior policy which produces a set D_0 of tuples i.i.d. using eventually a set of different initial states S_0 or a fixed one s_0. (This step can be also avoided setting directly $Q = 0$ in each state). This data set is only used to initialize the algorithm solving the BRM_ε and providing an initial approximation of the value function. Hence $API-BRM_\varepsilon$ algorithm proceed incrementally adding new experiences and

Algorithm 3 Online $API-BRM_\varepsilon$ with $\varepsilon-$greedy exploration

Require: $(\kappa, \lambda, \gamma, K_P, \varepsilon_k\, function)$
  $l \leftarrow 0$ initialize $\hat{Q}_0(s,a)$ $(\hat{\pi}_0)$
  solve initial SVR:
  $Q_1 \leftarrow API-BRM_\varepsilon(\Pi_0, D_0, \kappa, \lambda)$ (policy evaluation)
  store initial next state policy Π_0
  measure initial state s_0
  for all time step $k > 0$ **do**
    update exploration factor ε_k
    choose action: $a_k = \{\pi_k(\cdot)\, w.p.\, 1-\varepsilon_k \vee \text{random action}\, w.p.\, \varepsilon_k\}$
    apply a_k and measure next state s_{k+1} and reward r_{k+1}
    update training sample set:
    $D_k \leftarrow D_{k-1} \cup (s_k, a_k, r_{k+1}, s_{k+1})$
    update next state policy $\Pi_k \leftarrow \Pi_{k-1} \cup \pi_k(\cdot, s_{k+1})$
    solve incremental SVR:
    $\hat{Q}_k \leftarrow IncrementalAPI-BRM_\varepsilon(\Pi_k, D_k, \kappa, \lambda)$
    if $k = (l+1)K_P$ **then**
      update policy $\pi_l(\cdot) \leftarrow \hat{\pi}(\cdot, \hat{Q}_{k-1})$
    end if
    $l \leftarrow l+1$
  end for
  return

improving the policy any K_P steps (with K_P a tunable parameter) using an $\varepsilon-$greedy policy. The exploration partly relies on the initial data set D_0 and partly depends on the way we manage the exploration ε. Generally one may foresee an exponential decay for the ε factor starting from some value $\varepsilon_0 \leq 1$ and when no further exploration is required fix a minimum value $\varepsilon_\infty = 0.1$. We assume that the process underlying the collected data D_n follows an unknown β-mixing distribution.

3. Method-3 (online-growth $API-BRM_\varepsilon$) another possibility consists to alternate explorative samples using a random behavior policy with exploitative samples every K_e steps (with K_e a tunable parameter) and using the $\varepsilon-$greedy policy learned with a small exploration ε . This can be considered an online variant of the batch-growth method. We assume that the process underlying the collected data D_n follow an unknown β-mixing distribution.

Algorithm 3 illustrates the online variants of $API-BRM_\varepsilon$ using an $\varepsilon-$greedy exploration policy. The algorithm allows the definition of two parameters which are not present into the offline version: the number of transition $K_P \in N_0$ between consecutive policy improvements and the exploration schedule K_e. Policy is fully optimistic whenever $K_P = 1$ and the policy is updated after every sample while while is partially optimistic with $1 < K_P \leq K_{max}$ where in experiments we choose $K_{max} = 10$. Extensive study about how this parameters affects the quality of the solution for the online LSPI can be found in [17] which in principle might apply to our method. The exploration schedule can be controlled by the parameter K_e and the decay factor ε_{decay} which should be chosen not too

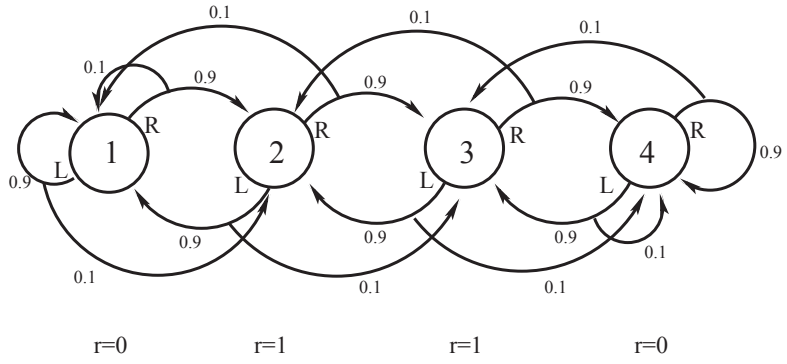

Figure 5.1: The four states chain walk control problem from ([48])

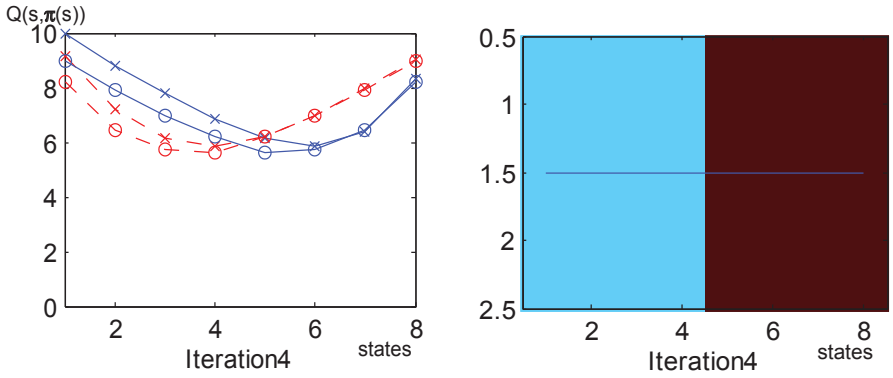

Figure 5.2: Chain walk (8-states) solved with LSPI (from [48]): Upper: state value function $Q^{\pi}(s,\pi(s))$ of the learned policy (LSPI - solid line; exact- dotted line). The approximations are shown with solid lines, whereas the exact values are connected with dashed lines. The values for action L are marked with **o** and for action R with **x**. Lower: Final policy (R action - dark/red shade; L action - light/blue shade; LSPI - top stripe; exact - bottom stripe)

large while a significant amount of exploration is necessary. However several alternatives are possible as described in section 3.2 and we also implemented an online variant of the batch-growth method. In this way we perform learning alternating exploration trials using some exploration factor ε_f with exploitation trials using a ε_0. In practice this method works well and allows to easily found local optimal solution starting from a fixed initial state s_0 . Nevertheless if we need to find a global optimal valid for any initial state, it becomes necessary to explore through the set of initial states and Method 1 or Method 2 must be applied while learning becomes slower.

5.5 The Chain Walk Control Problem

The chain walk control problem [47] consists of a chain with $S = \{1,2,...,N\}$ states shown in fig. 5.1 for case N=4 which can be easily generalized for any integer N. Two actions

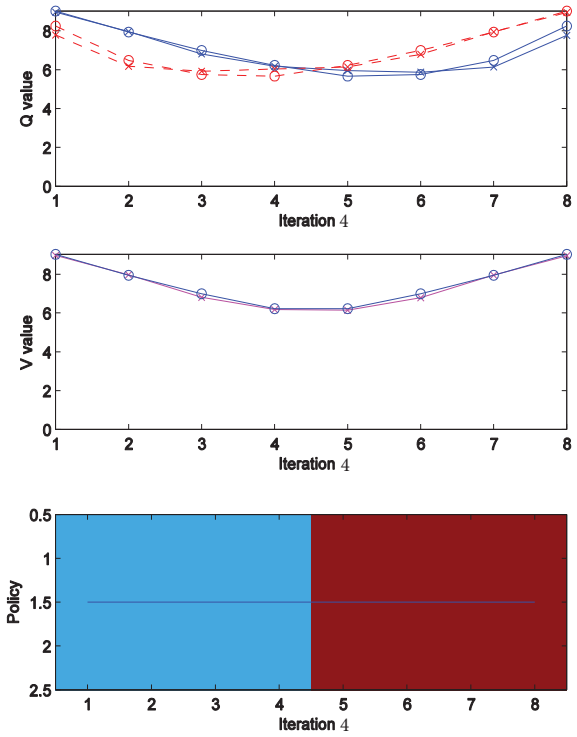

Figure 5.3: Chain walk (8-states) solved with $API-BRM_\varepsilon$: Upper: action value function $Q^\pi(s,\pi(s))$ of the learned policy ($API-BRM_\varepsilon$ **o** exact, **x** approximated values). Center: value function $V^\pi(s)$ of the learned policy ($API-BRM_\varepsilon$ **o** exact, **x** approximated values). Lower: Final policy (R action - dark/red shade; L action - light/blue shade; $API-BRM_\varepsilon$ - top stripe; exact - bottom stripe)

are allowed $A=\{-L,R\}$ moving to the left or right direction. Applying the actions $a\in A$ the probability of success in changing the state in the wanted direction is $P_s=0.9$ while $P_f=0.1$ is the probability to fail. The boundaries of the chain $S_b=\{1,N\}$ are dead-ends. The reward over N states is

$$r(s_t,a_t)=\begin{cases}0 & if \quad s_t\in S_b\\ +1 & otherwise\end{cases}$$

while the discount factor is set to 0.9. The optimal policy is moving to the right in any state $S_R=\{1,...,N/2\}$ while is moving to the left in any state $S_L=\{N/2+1,...,N\}$ and for the $N=4$ case is $\pi^*=\{RRLL\}$. For the case $N=4$ [47] shows that starting with the policy $\pi_R=\{RRRR\}$ the model oscillates between the suboptimal policies $\pi_R=\{RRRR\}$ and $\pi_L=\{LLLL\}$ which makes the chain walk problematic. For us chain walk is particularly interesting as it is possible to compare the approximations learned to the true underlying value functions, and therefore understand better the way the algorithm works. Offline LSPI [48] finds the optimal policy collecting 4000 samples using random actions for about 200 episodes of 20 steps and 4 PIs. A linear approximation architecture using polynomials of grade two Basis Functions (BFs). The approximation captures the qualitative structure of the value function. One problem with this benchmark arises from the

Parameter	Description	Value	UM
g	gravity constant	9.8	m/s^2
m	pole mass	2.0	Kg
M	cart mass	8.0	Kg
l	pole length	0.5	m
α	1/(m+M)	0.1	Kg^{-1}
dt	simulation step	0.1	s
r	reward	0/-1	
γ	discount factor	0.95	

Table 5.1: Parameters used in the simulation for the inverted pendulum control problem

state visitation distribution for this training which should be uniform to prevent uneven approximation errors over the state-action space. fig. 5.2 shows the policy learned by offline LSPI for the more problematic $8-$states variant. In this case we run $API-BRM_\varepsilon$ using Method-1 (offline $API-BRM_\varepsilon$) collecting 1000 samples using random actions simulating 50 episodes of 20 steps and 4 PIs. Results are shown in fig. 5.3 confirming that the algorithm is able to find the correct solution. The qualitative structure of the approximation is captured by the value function with relatively small quantitative errors. For this simulations an RBF kernel was used with $\Sigma = I_9\sigma$ where $\sigma = 1$ and the regression parameters where chosen as $C = 1$ and $\varepsilon = 0.01$ selected using an grid search which are able to well approximate the value function and the correct optimal policy. Experiments not shown using Methods-2 and 3 confirm that the same quality of the solution can be reached with a rather uniform state visitation distribution.

5.6 The Inverted Pendulum Control Problem

For the inverted pendulum benchmark, the control problem consists in balancing at the upright position a pendulum of unknown length and mass. This can be done by applying a force on the cart where the pendulum cart is attached to [102]. Due 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. In this version of the problem we only have one degree of freedom which can be obtained by fixing the pole to an axis of rotation. A version with two degrees of freedom is presented in the next section. The state space $S \setminus S_T = \{(\theta, \dot{\theta}) \in \mathbb{R}^2\}$ is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the inverted pendulum and a terminal state S_T described later. Three actions are allowed $A = \{-a_m, 0, a_m\}$ where $a_m = 50N$ and some uniform noise in $\sigma_a \in [-10, 10]$ might be added to the chosen action. The transitions are governed by the non-linear dynamics of the system as:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2 \sin(2\theta)/2 - \alpha\cos(\theta)u}{4/3l - m\alpha\cos(\theta)^2} \tag{5.1}$$

where θ is the angular position, m the mass of the pole, l the length of the pole, M the mass of the cart, $u = a + \sigma_a$ the control action with noise consisting in the acceleration applied to the cart, g the gravity constant. The parameters of the model used in the simulation are reported in table 5.1. The angular velocity $\dot{\theta}$ is restricted to $[-4\pi, 4\pi] rads^{-1}$ using

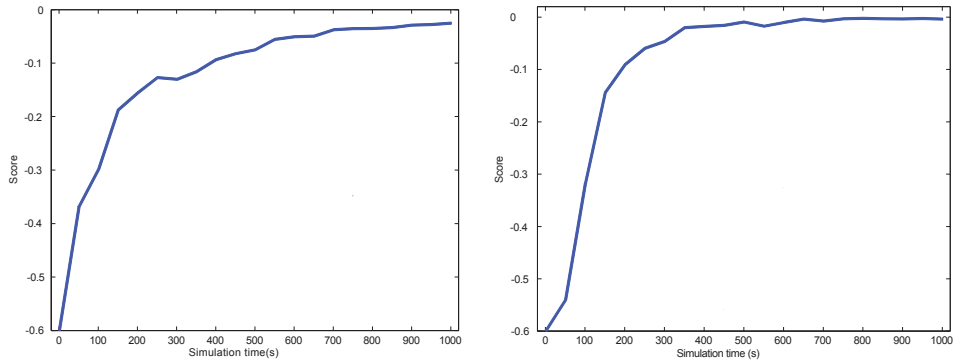

Figure 5.4: Inverted pendulum: average score for offline LSPI (left) and Q-learning with experience replay (right) from [48]

saturation. The discrete-time dynamics is obtained by discretizing the time between t and $t+1$ chosen with $dt = 0.1s$. If θ_{t+1} is such that $|\theta_{t+1}| > \theta_m$ a terminal state $S_T = \{|\theta| > \theta_m\}$ is reached where we fixed $\theta_m = \pi/2$. The reward function $r(s_t, a_t)$ is defined through the following expression:

$$r(s_t, a_t) = \begin{cases} -1 & if \quad |\theta| > \theta_m \\ 0 & otherwise \end{cases}$$

The discount factor γ has been chosen equal to 0.95. The dynamical system is integrated by using an Euler method with a $0.001s$ integration time step. To generate data samples we may consider episodes starting from the same initial state $s_0 = (\theta_0, \dot{\theta}_0)$ or using a random initial state and stopping when the pole leaves the region represented by $S \setminus S_T$ meaning enter in a terminal states S_T. In [48] an analysis of the same benchmark is reported comparing performance with offline LSPI and Q-learning. In this case simulation runs for $1000s$ (10000 samples) separated in 1000 trials of max $1s$ (10 samples) stopping eventually when reaching a terminal state. Offline LSPI uses a linear approximation architecture with a set of 10 Basis Functions (BFs) for each one of the 3 actions, thus a total of 30 basis functions, to approximate the value function. These 10 BFs included a constant term and 9 RBF functions arranged in a 3×3 grid over the $2-$dimensional state space with $BF_i(s) = e^{-\|s-\mu_i\|^2/(2\sigma^2)}$ where μ_i are the 9 points of the grid $\{-\pi/4, 0, +\pi/4\} \times \{-1, 0, +1\}$ and $\sigma = 1$. Training samples were collected starting in a randomly perturbed state very close to the equilibrium state $s_0 = (0,0)$ and following a policy that selected actions uniformly at random. Results are shown in fig. 5.4 showing the performance in terms of balancing steps from the analysis detailed in [48]. Each episode was allowed to run for a maximum of $300s$ (3000 steps) of continuous balancing. A run that balanced for this period of time was considered to be successful. The optimal policy found by LSPI is good enough using the initial state $s_0 = (0,0)$ while is much worse with other initial states. In our simulation of the same benchmark using online $API - BRM_\varepsilon$ we run for $1000s$ of simulated time collecting around 10000 samples. Run was split into separate learning episodes initiated at random initial states and stopping when a terminal state has been reached or otherwise after $30s$ (300 steps). Policy improvement were performed once every $K_P = 10$ steps $(0.1s)$ using an $\varepsilon-$greedy policy

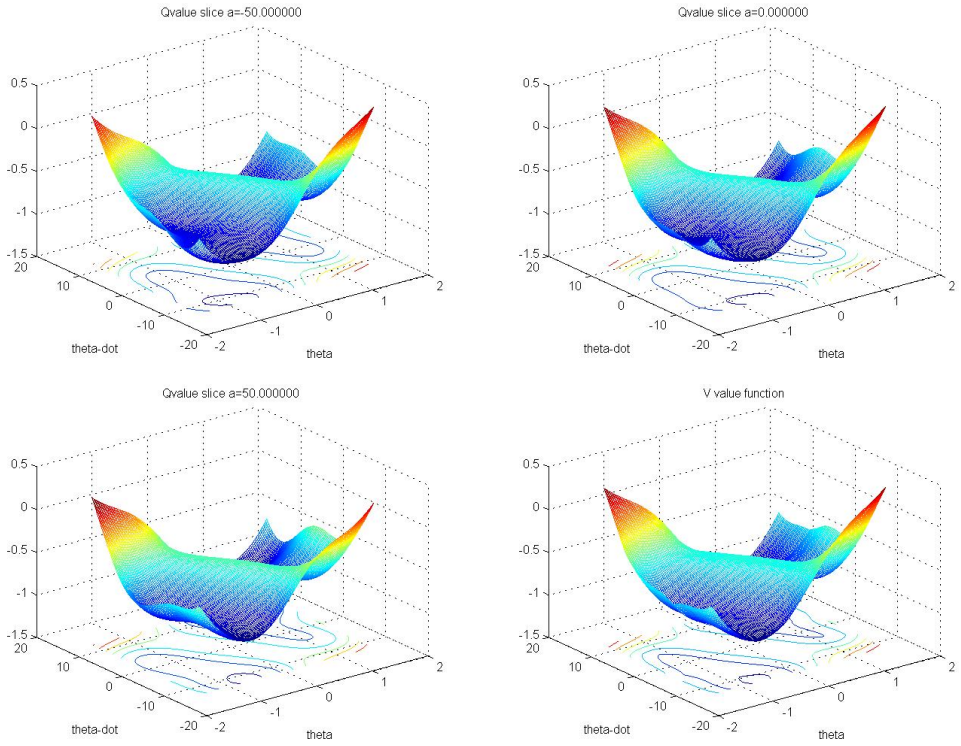

Figure 5.5: Inverted pendulum: approximate suboptimal Q and V value function found by $API-BRM_\varepsilon$

with $\varepsilon_0 = 1$ and reaching a value of $\varepsilon_\infty = 0.1$ after $350s$. We also used an RBF kernel with parameters $\Sigma = I_3\sigma$ with $\sigma = 0.5$ and the regression parameters where chosen as $C = 10$ and $\varepsilon = 0.01$ selected using an grid search. fig. 5.6 shows a subsequence of policies found during representative run taken after simulation times $t = 10s, 50s, 200s, 1000s$. Clearly the generalization ability of the SVR makes possible to capture the structure of the approximated policy only after $50s$ of simulation time which closely resembles the final policy obtained after $1000s$ of simulation time. fig. 5.5 shows the final approximation of Q and V value functions. fig. 5.7 shows the performance of the final policy found by online $API-BRM_\varepsilon$ along the online learning process. The performance was measured evaluating the score over a grid of initial states simulating balancing up to $300s$ (3000 steps). Using Q-learning requires state discretization which seriously influences the performance and it cannot estimate the state action value properly until the state is visited which slows the learning. On the contrary the generalization property of SVR our algorithm can estimate state values of unvisited states reasonably using the experience gained with the other states. Moreover being a non-parametric regression it can easily adapt to different situation while in general parametric approximation may work well but in a specific context without the possibility to eventually adapt to changes. In fig. 5.8 we report states and actions in subsequences of learning trials. Each trial lasts 30s max considered as the minimum balancing to reach. Using Method-3 (online growth) with a fixed initial state $S_0 = (0,0)$ an optimal local approximation can be found in less then $30s$ of simulation time. Finally the number of support vectors necessary to represents the approximate

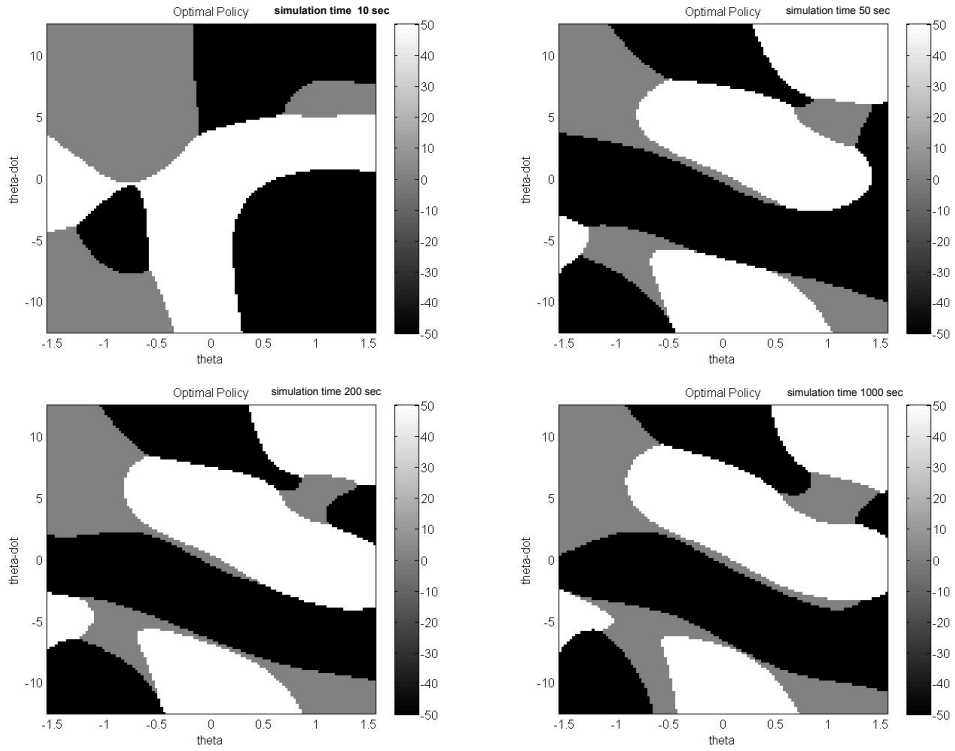

Figure 5.6: Inverted pendulum: representative subsequences of policy found by online $API - BRM_\varepsilon$ using Method-2 (Actions are discretized and only three grey levels show up)

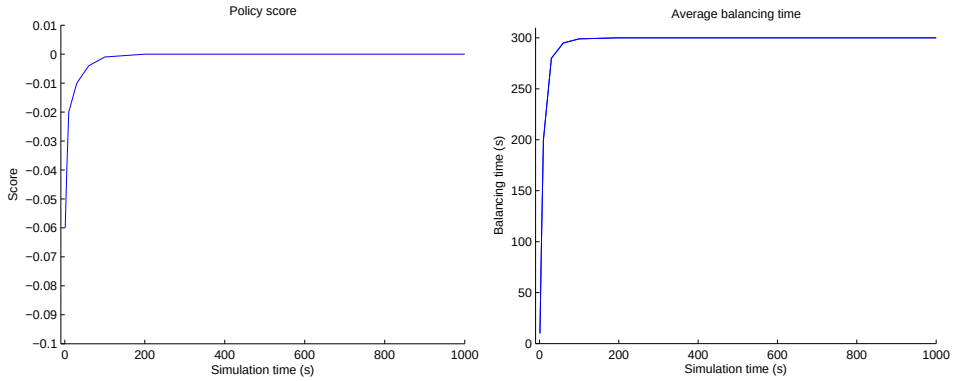

Figure 5.7: Inverted pendulum: (left) average score of online $API - BRM_\varepsilon$ with $K_P = 10$ over a grid of initial states; (right) average balancing time over the same grid of initial states using Method-2

action value function with the set of parameters used in the approximation usually stays below 5% of the total number of collected samples which is also a indication of the quality of the approximation.

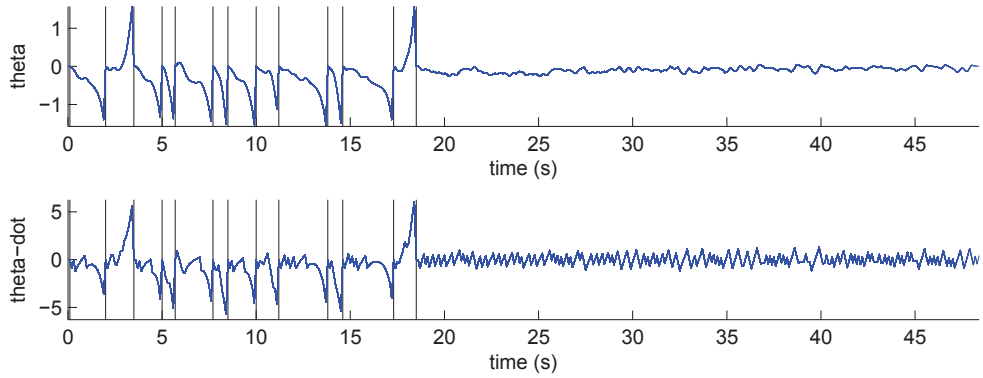

Figure 5.8: Inverted pendulum: States and actions in representative subsequences of learning trials. Each trial lasts 30s max considered as the minimum balancing to reach. Using Method-3 (online growth) with a fixed initial state $S_0 = (0,0)$ $API - BRM_\varepsilon$ learns a local optimal policy in a few episodes (20s of simulation time).

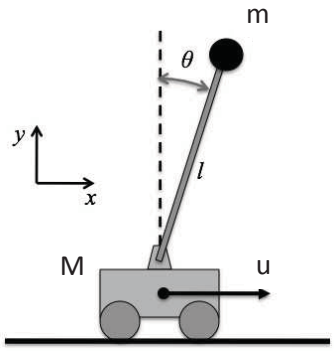

Figure 5.9: The cart pole balancing control problem

5.7 The Cart Pole Balancing Control Problem

The cart pole balancing control problem [89] consists in a pole mounted on cart that has to be stabilized inside an acceptable region by the motions of the cart (fig. 5.9). This benchmark is of interest since it involves a $5-$dimensional state action space. The state space $S \setminus S_T = \{(\theta, \dot{\theta}, x, \dot{x}) \in \mathbb{R}^4\}$ is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum, the cart position x and velocity $\dot{x}$ and a terminal state S_T described later. Two actions are allowed $A = \{-a_m, a_m\}$ with left force $LF = -a_m$, right force $RF = +a_m$ where $a_m = 10N$ and some uniform noise in $\sigma_a \in [-1,1]$ might be added to the chosen action. The transitions are governed by the nonlinear dynamics of the system:

$$\begin{aligned} \ddot{\theta} &= \frac{g\sin(\theta) - ml\alpha\sin(2\theta)/2 - u\alpha\cos(\theta) + \mu_c\cos(\theta)sign(\dot{x}) - \frac{\mu_p\dot{\theta}}{ml}}{4/3l - m\alpha\cos(\theta)^2} \\ \ddot{x} &= u + ml\alpha\dot{\theta}^2\sin(\theta) - ml\alpha\ddot{\theta}\cos(\theta) - \alpha\mu_c sign(\dot{x}) \end{aligned} \tag{5.2}$$

Parameter	Description	Value	UM
g	gravity constant	9.8	m/s^2
m	pole mass	0.1	Kg
M	cart mass	1.0	Kg
l	pole length	0.5	m
α	1/(m+M)	0.91	Kg^{-1}
μ_p	pole friction coefficient	0	$Kg\ rad/s^2$
μ_c	cart friction coefficient	0	$Kg\ m/s^2$
dt	simulation step	0.02	s
r	reward	0/-1	
γ	discount factor	0.9	

Table 5.2: Parameters used in the simulation for the cart pole balancing control problem

where x is the cart position, θ the angular position, m the mass of the pole, l the length of the pole, M the mass of the cart, $u = a + \sigma_a$ the control action with noise consisting in the acceleration applied to the cart, g the gravity constant and μ_c and μ_p the cart and pole friction coefficients respectively. The parameters of the model used in the simulation are reported in table 5.2. The pole angular velocity $\dot{\omega}$ is restricted to $[-\pi, \pi] rads^{-1}$ while the cart velocity $\dot{x}$ is restricted to $[-1, 1] ms^{-1}$ using saturation. The discrete-time dynamics is obtained by discretizing the time between t and $t+1$ chosen with $dt = 0.02s$. If θ_{t+1}, x_{t+1} are such that $|\theta_{t+1}| > \theta_m$ or $|x_{t+1}| > x_m$ a terminal state $S_T = |\theta| > \theta_m \quad or \quad |x| > x_m$ is reached where we fixed $\theta_m = \pi/15$ and $x_m = 2.4$. The reward function $r(s_t, a_t)$ is defined through the following expression:

$$r(s_t, a_t) = \begin{cases} -1 & if \quad |\theta| > \theta_m \quad or \quad |x| > x_m \\ 0 & otherwise \end{cases}$$

Variant of the cart pole problem are possible using a more informative reinforcement function but we keep with the original formulation in [89]. The discount factor γ has been chosen equal to 0.95. The dynamical system is integrated by using an Euler method with a $0.001s$ integration time step. To generate data samples we may consider episodes starting from the same initial state $s_0 = (\theta_0, \dot{\theta}_0, x_0, \dot{x}_0)$ or using a random initial state and stopping when the cart pole leaves the region represented by $S \setminus S_T$ meaning enter in a terminal states S_T. In our simulation of this benchmark using online $API - BRM_\varepsilon$ we run for $500s$ of simulated time collecting around 10000 samples. Run was split into separate learning episodes initiated at random initial states and stopping when a terminal state has been reached or otherwise after $10s$ (500 steps). Policy improvement were performed once every $K_P = 10$ steps $(0.1s)$ using an $\varepsilon-$greedy policy with $\varepsilon_0 = 1$ and reaching a value of $\varepsilon_\infty = 0.1$ after $400s$. We also used an RBF kernel with parameters $\Sigma = I_5\sigma$ with $\sigma = 1$ and the regression parameters where chosen as $C = 10$ and $\varepsilon = 0.02$ selected using an grid search. fig. 5.11 shows a subsequence of slices of policies fixing some state variables found during representative run taken after simulation times $t = 10s, 50s, 200s, 500s$. Clearly the generalization ability of the SVR makes possible to capture the structure of the approximated policy only after $50s$ of simulation time which is closely resembles the final policy obtained after $500s$ of simulation time. fig. 5.10 shows the final approxima-

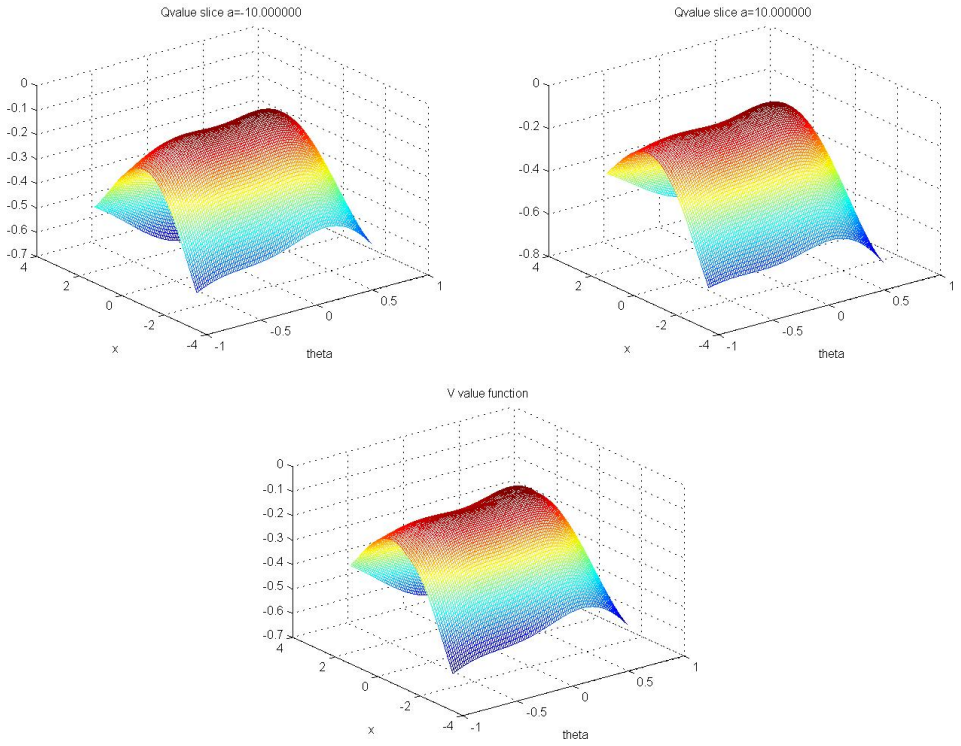

Figure 5.10: Cart pole: slices of approximate suboptimal Q and V value function found by online $API-BRM_\varepsilon$ using Method-2

tion V value functions and a slice of the Q value function fixing some state variables. fig. 5.12 shows the performance of the final policy found by online $API-BRM_\varepsilon$ along the online learning process. The performance was measured evaluating the score over a grid of initial states simulating balancing up to $60s$ (3000 steps). In fig. 5.13 we report states and actions in subsequences of learning trials. Each trial lasts 10s max considered as the minimum balancing to reach the goal. Using Method-3 (online growth) with a fixed initial state $S_0 = (0,0)$ an optimal local approximation can be found in less then $30s$ of simulation time. Finally the number of support vectors necessary to represents the approximate action value function with the set of parameters used in the approximation usually stays below 10% of the total number of collected samples which is also a indication of the quality of the approximation.

5.8 The Car On The Hill Control Problem

The hill car problem is a classical benchmark for approximate RL [63]. Here we present the much more problematic variant of [26]. This last version of the benchmark has been analyzed in [26] and [19] and due to the peculiar structure of the reward function it represents a challenge for the value function approximation task. A car modeled by a point mass is traveling on a hill with shape given by the function $H(p)$ in fig. 5.14. The state space S of dimension two is composed car position p and velocity

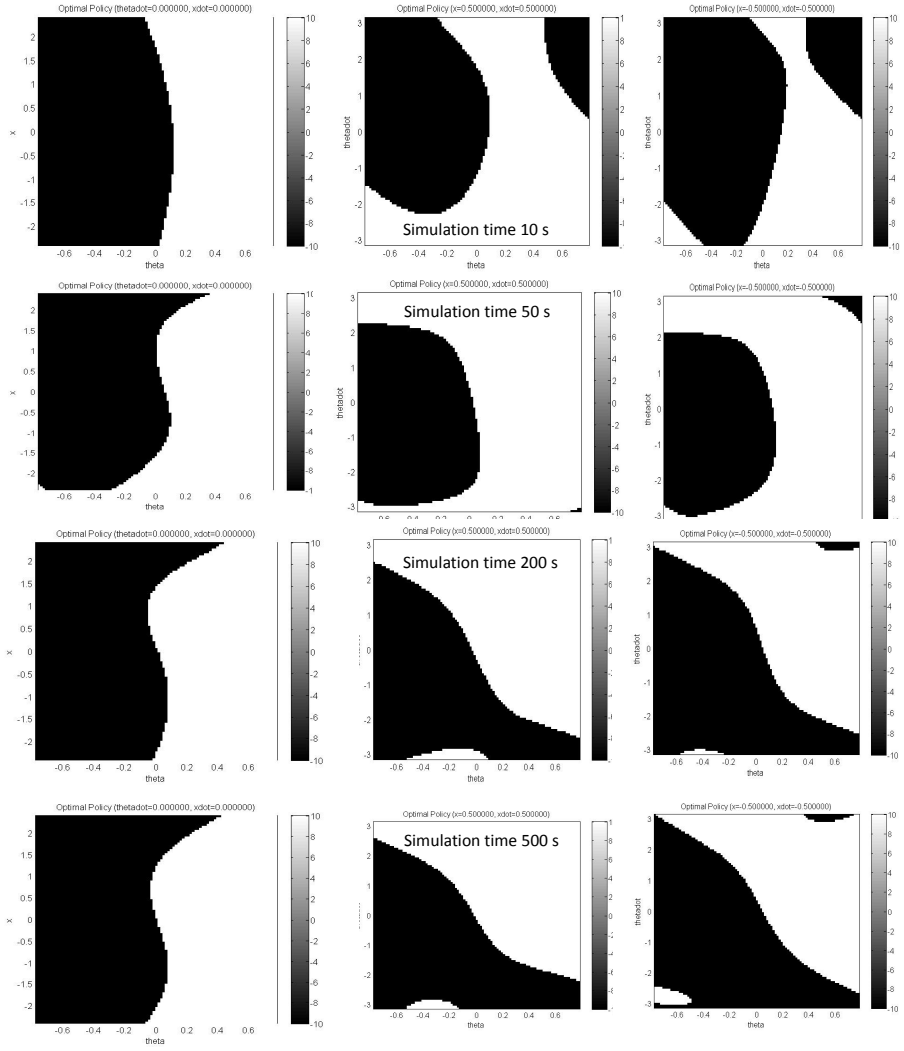

Figure 5.11: Cart pole: slices of representative subsequences of policy found by online $API - BRM_\varepsilon$ using Method-2

$\dot{p}$ $S \setminus S_T = \{(p,\dot{p}) \in \mathbb{R}^2 \, |p| \leq p_m = 1 \quad and \quad |\dot{p}| \leq \dot{p}_m = 3\}$ and a terminal state S_T described later with p position and $\dot{p}$ speed of the car. The action space A with dimension two defined as $A = \{-a_m, a_m\}$ acts directly on the acceleration of the car and can only assume two extreme values $a_m = 4$. The control problem objective is to bring the car as soon as possible to the top of the hill $(p = +p_m)$ while preventing the position p of the car to become smaller than $p = -p_m$ and its speed $\dot{p}$ to go outside the interval $|\dot{p}| \leq \dot{p}_m$. The system has a continuous-time dynamics:

$$\ddot{p} = \frac{a/m - g\,H'(p) - \dot{p}^2\,H'(p)H''(p)}{1+(H'(p))^2} \tag{5.3}$$

where the mass $m = 1\,g$ and $g = 9.81\,ms^{-2}$ are fixed parameters and

$$H(p) = \begin{cases} p^2 + p & if \quad p < 0 \\ \frac{p}{\sqrt{1+5p^2}} & if \quad p \geq 0 \end{cases}$$

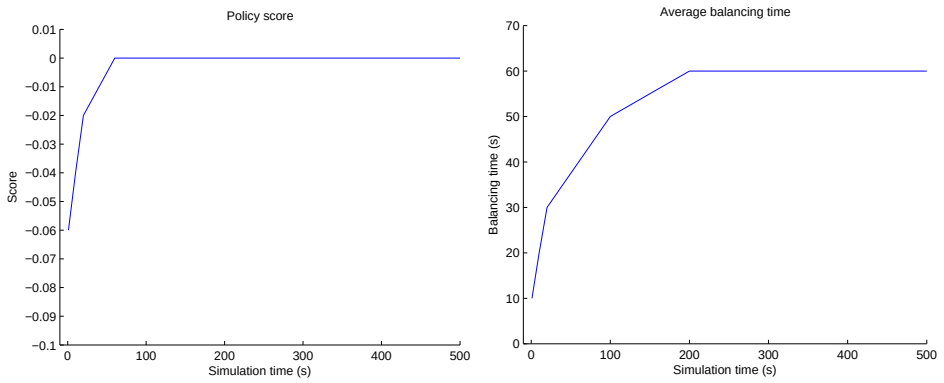

Figure 5.12: Cart pole: (left) average score of online $API-BRM_\varepsilon$ with $K_P = 10$ over a grid of initial states using Method-2; (right) average balancing time over the same grid of initial states using Method-2

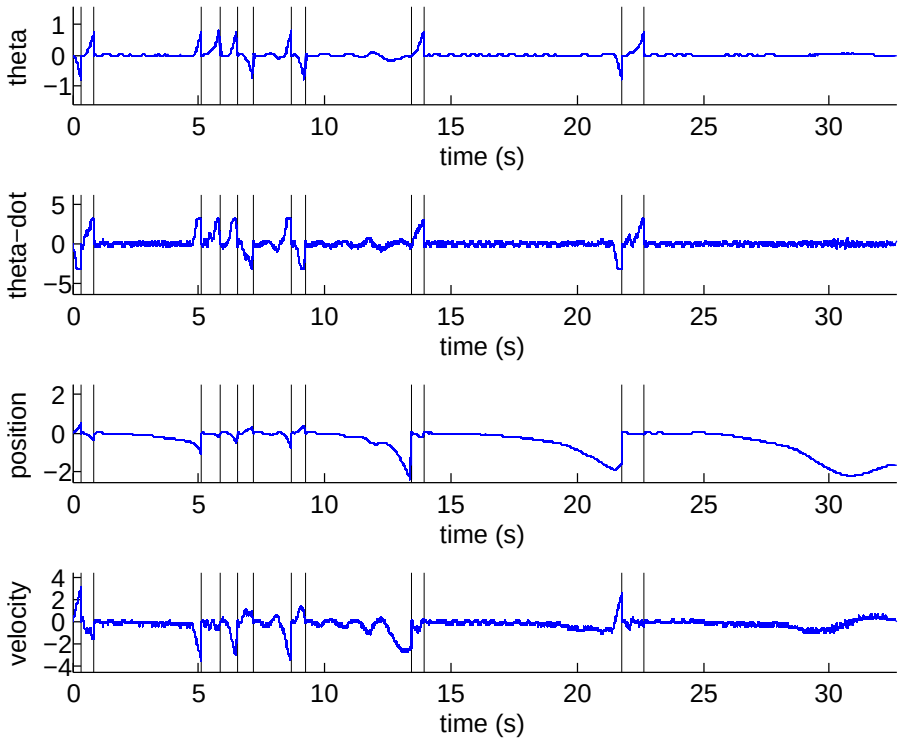

Figure 5.13: Cart pole: States and actions in representative subsequences of learning trials. Each trial lasts 10s max considered as the minimum balancing to reach. Using Method-3 (online growth) with a fixed initial state $S_0 = (0,0,0,0)$ $API-BRM_\varepsilon$ learns a local optimal policy in a few episodes ($20s$ of simulation time).

is a function defining the shape of the hill while $H(p)' = \frac{dH(p)}{dp}$ and $H(p)'' = \frac{d^2H(p)}{dp^2}$. The discrete-time dynamics is obtained by discretizing the time between t and $t+1$ chosen with $dt = 0.1s$. If $p_{t+1}, \dot{p}_{t+1}$ are such that $|p_{t+1}| > +p_m$ or $|\dot{p}_{t+1}| > +\dot{p}_m$ a terminal state

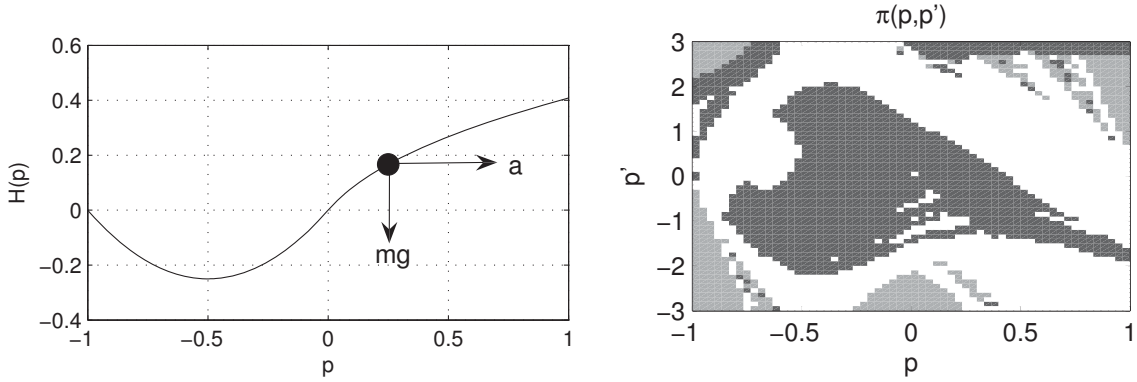

Figure 5.14: Car on the Hill: shape of the hill (left), near optimal policy $\pi(p,\dot{p})$ (black $a=-a_m$,white $a=+a_m$, grey equally good) from [19]

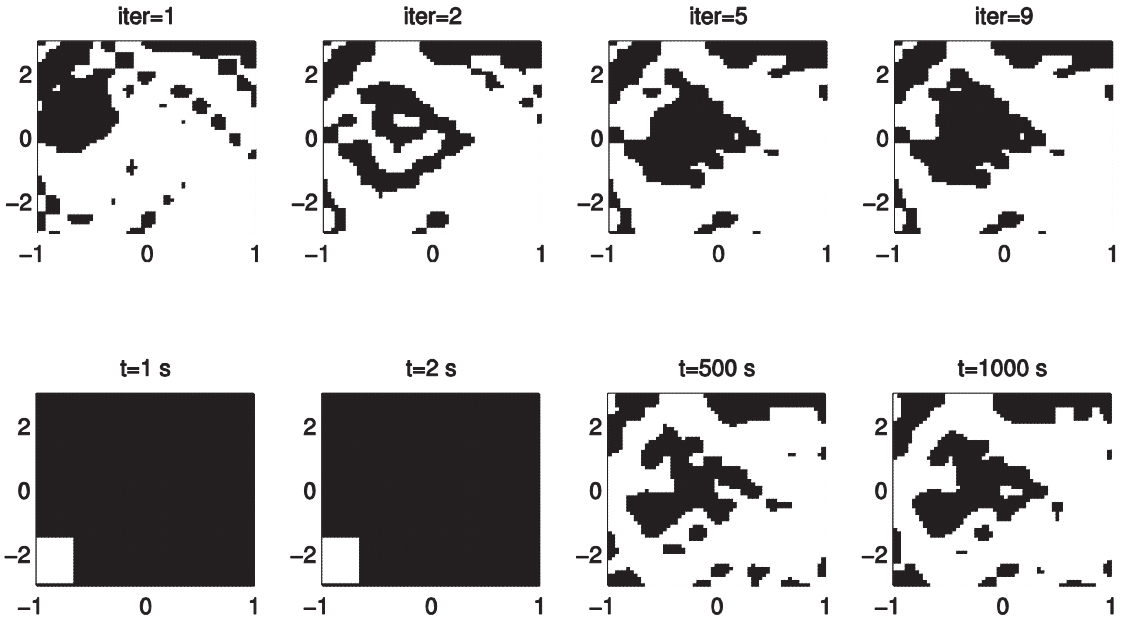

Figure 5.15: Car on the Hill: representative subsequences of policies found by offline (top) and online LSPI (bottom) from [19]. See fig. 5.14 for axis and color meanings.

S_T is reached. The reward function $r(s_t,a_t)$ is defined through the following expression:

$$r(s_t,a_t)=\begin{cases} -1 & if \quad p_{t+1}<-p_m \quad and \quad |\dot{p}_{t+1}|>\dot{p}_m \\ 1 & if \quad p_{t+1}>+p_m \quad and \quad |\dot{p}_{t+1}|\leq \dot{p}_m \\ 0 & otherwise \end{cases}$$

The discount factor γ has been chosen equal to 0.95. The dynamical system is integrated by using an Euler method with a $0.001s$ integration time step. As a potential

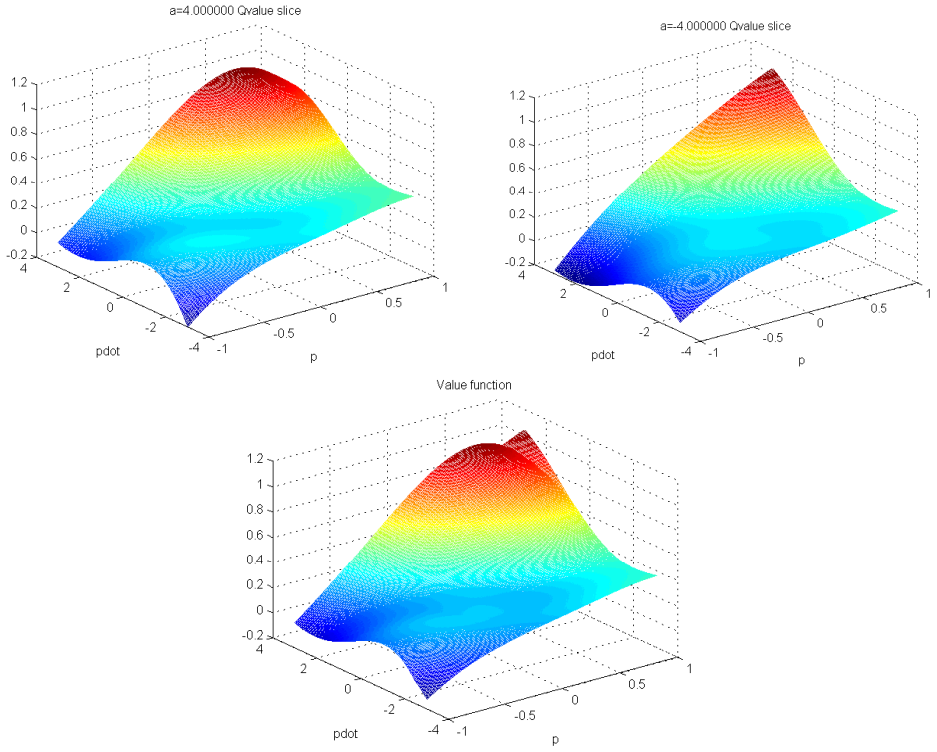

Figure 5.16: Car on the Hill: approximate suboptimal Q and V value function found by online $API - BRM_\varepsilon$ using Method-2

variants one may consider adding some uniform random noise $\sigma_a \ll a_m$ to the action value a as well as moving from a discrete action set to a continuous one in the interval $[-a_m, a_m]$. To generate data samples we may consider episodes starting from a fixed initial state corresponding to the car stopped at the bottom of the hill $s_0 = (p_0, \dot{p}_0) = (-0.5, 0)$ or with random initial state and stopping when the car leaves the region represented by $S \setminus S_T$ meaning a terminal state S_T.

As we want to compare the performance of our method with LSPI both online and offline, we refer to the analysis of the same benchmark presented in [19]. In this analysis a comparison among LSPI both online an offline are presented. LSPI was applied to the hill car benchmark using a random collected dataset of 10000 samples, i.i.d. distributed and collected according to some random behavior policy. As LSPI uses a parametric approach the action value function is approximated using a bilinear interpolation on an equidistant 13 grid. In this case the approximation can be expressed as $\hat{Q}_{LSPI}(s, a_j) = \sum_{t=1}^{169} \Phi_t(s) w_{t,j}$ where the Basic Functions (BFs) $\phi_t(s)$ provide the interpolation coefficients and $j = 1, 2$ for the two possible actions. Samples in LSPI are reused to evaluate the policy typically converging in less than 10 PI steps. The upper part of fig. 5.15 illustrates subsequences of policy found by these implementation of offline LSPI for the given benchmark. In this case simulation runs for $1000s$ (10000 samples) separated in 350 trials of max $3s$ (30 samples) stopping eventually when reaching a terminal state. Policy improvements are performed once every $1s$ (10 samples) using an $\varepsilon-$greedy exploration strategy with an initial value of $\varepsilon_0 = 1$ decaying exponentially and reaching the value of $\varepsilon_\infty = 0.1$ after

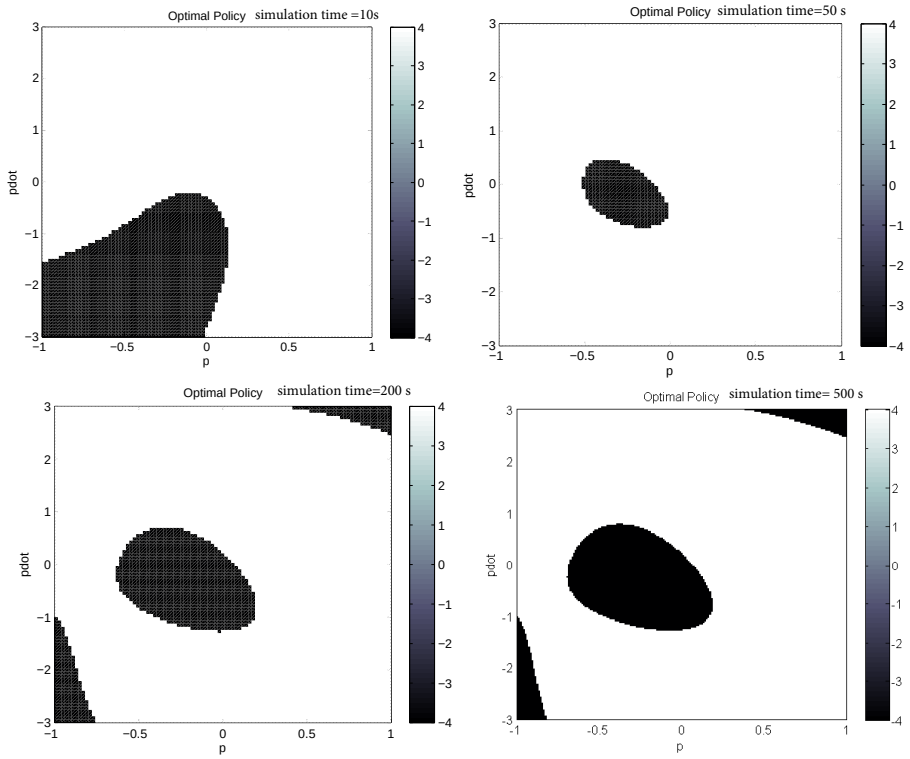

Figure 5.17: Car on the Hill: representative subsequences of policy found by online $API-BRM_\varepsilon$ using Method-2

$350s$ (3500 samples). Lower part of fig. 5.15 illustrates the subsequences of policies found during representative runs. Difference among online and offline LSPI arises basically from the fact that while offline LSPI processes the data once at every iteration, online e LSPI processes each sample only once.

For comparison the experiment runs for 1000s of simulated time so that ends up collecting 10000 samples. The samples have been split into separate episodes with an initial random state and stopping when a terminal state has been reached or in any case after $3s$ (30 samples). Policy improvements are performed every $1s$ (10 samples) and an $\varepsilon-$greedy strategy is used with an initial value of $\varepsilon_0 = 1$ decaying exponentially and reaching the value of $\varepsilon_\infty = 0.1$ after $350s$ (3500 samples). In fig. 5.15 in the upper part the results coming from iterations of an LSPI In our simulation of this benchmark using online $API-BRM_\varepsilon$ we run for $500s$ of simulated time collecting around 5000 samples. Run was split into separate learning episodes initiated at random initial states and stopping when a terminal state has been reached or otherwise after $3s$ max (30 steps). Policy improvement were performed once every $K_P = 10$ steps $(0.1s)$ using an $\varepsilon-$greedy policy with $\varepsilon_0 = 1$ and reaching a value of $\varepsilon_\infty = 0.1$ after $350s$ (3500 steps). We also used an RBF kernel with parameters $\Sigma = I_5\sigma$ with $\sigma = 2$ and the regression parameters where chosen as $C = 10$ and $\varepsilon = 0.01$ selected using an grid search. fig. 5.17 shows a subsequence of policies found during representative run taken after simulation times $t = 10s, 50s, 200s, 500s$. Clearly the generalization ability of the SVR makes possible to capture the structure of the approximated policy only after $50s$ of simulation time which is closely resemble the final

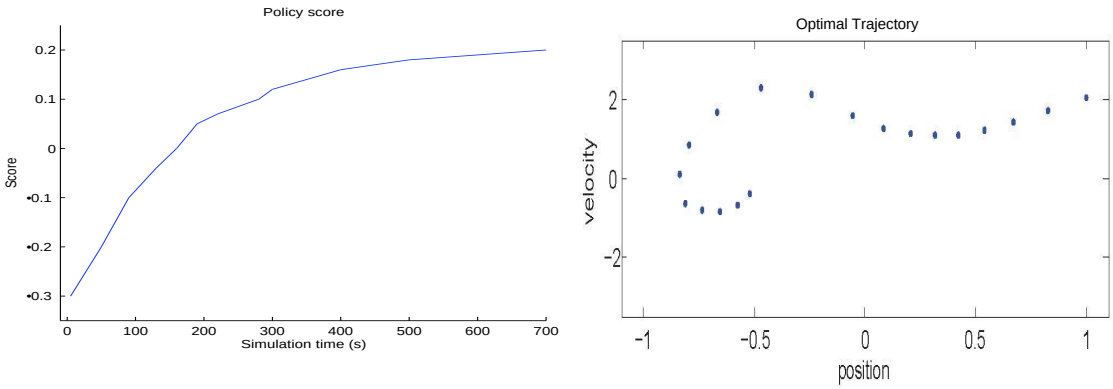

Figure 5.18: Car on the Hill: (left) average score of online $API - BRM_\varepsilon$ with $K_P = 10$ over a grid of initial states using Method-2; (right) typical trajectory using a suboptimal policy found by $API - BRM_\varepsilon$

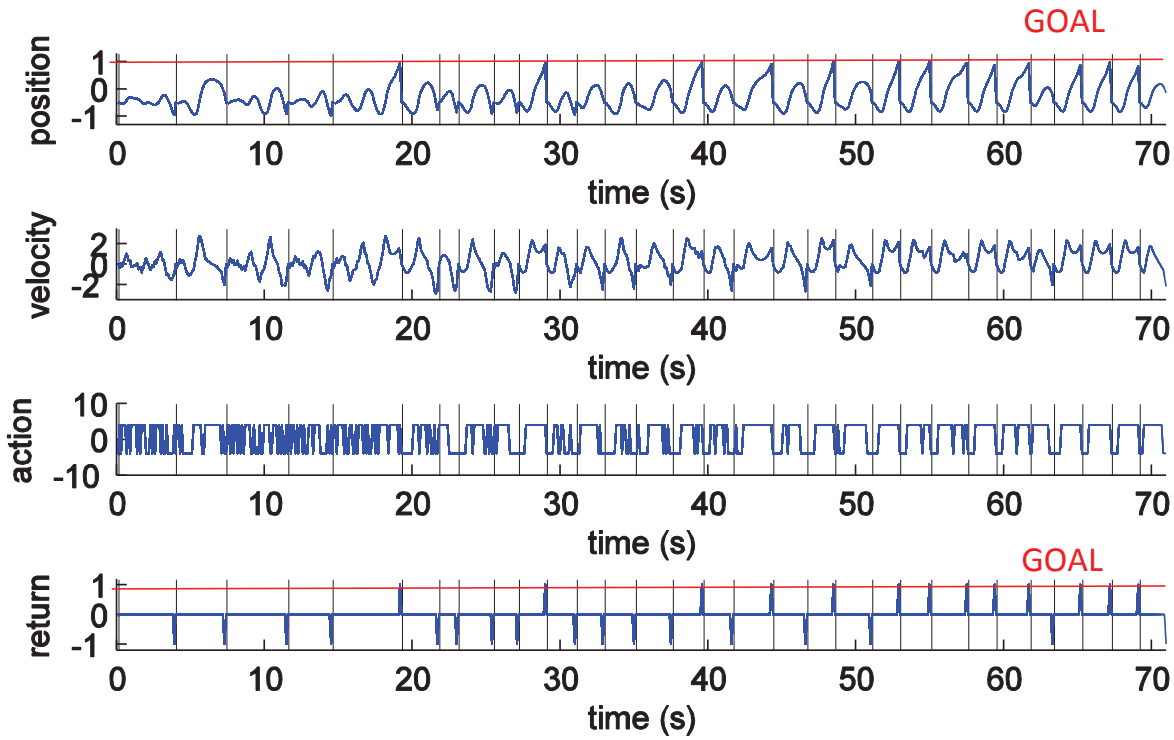

Figure 5.19: Car on the Hill: States and actions in representative subsequences of learning trials. Each trial lasts 3s max (30 steps) considered sufficient to reach the goal. Using Method-3 (online) with small perturbations of a fixed initial state $S_0 = (-0.5, 0)$ $API - BRM_\varepsilon$ may learn a local optimal policy in a few episodes (50s of simulation time).

policy obtained after 500s of simulation time. fig. 5.16 shows the final approximation V value functions and a slice of the Q value function fixing some state variables. In the right

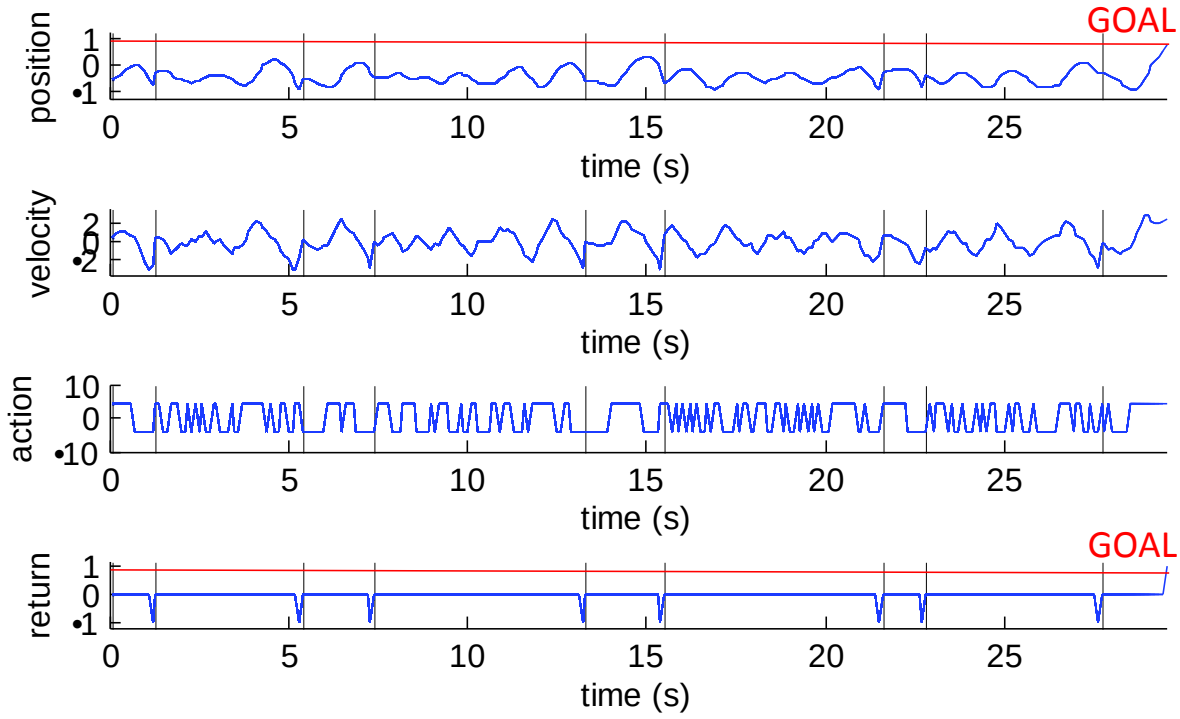

Figure 5.20: Car on the Hill: States and actions in representative subsequences of learning trials. Each trial lasts 8s max (80 steps) considered sufficient to reach the goal. Using Method-3 (online-growth) with small perturbations of a fixed initial state $S_0 = (-0.5, 0)$ $API - BRM_\varepsilon$ may learn a local optimal policy in a few episodes ($30s$ of simulation time).

of fig. 5.18 we represents a typical trajectory obtained when starting from s_0 and using the found policy $\hat{\pi}$ to control the system. In the left side of fig. 5.18 we show the performance of the final policy found by online $API - BRM_\varepsilon$ along the online learning process. The performance was measured evaluating the score over a grid of initial states. In fig. 5.20 we report states and actions in subsequences of learning trials. Each trial lasts 8s max (80 steps) considered sufficient to reach the goal using Method-3 (online-growth) using small perturbations of a fixed initial state $S_0 = (-0.5, 0)$. In this case policy improvement were performed once every $K_P = 10$ steps $(0.1s)$ and an optimal local approximation can be found in less then $30s$ (300 steps) of simulation time. However while the online-growth helps speeding the learn, the probability to reach the goal strongly depends on the distribution of the data. A local optimal approximation can be also found with experiments performed using Method-2 (online) with shown in fig. 5.19. Each trial lasts 3s max (30 steps) considered sufficient to reach the goal using using small perturbations of a fixed initial state $S_0 = (-0.5, 0)$. In this case we used an $\varepsilon-$greedy policy with $\varepsilon_0 = 1$ and reaching a value of $\varepsilon_\infty = 0.1$ after $40s$ (400 steps). Policy improvement were performed once every $K_P = 10$ steps $(0.1s)$ and an optimal local approximation can be found in less then $50s$ (500 steps) of simulation time. reveals that this method is quite sensitive to local minima of the value function. Hence, even though it is possible to find a good local approximation due to the peculiar structure of the problem exploration should be

Parameter	Description	Value	UM
g	gravity constant	9.8	m/s^2
Mr	Ryder mass	60.0	Kg
Mc	Bicycle mass	15.0	Kg
Md	Tyre mass	1.7	Kg
v	bicycle velocity	10.0/3.6	ms^{-1}
h	height from the ground	0.94	m
l	distance back-front tyres	1.11	m
r	wheel radius	0.34	m
d_{CM}	vertical distance bicycle-ryder CM	0.3	m
c	horizontal distance wheel-CM	0.66	m
dt	simulation step	0.01	s
r	reward	0/-1	
γ	discount factor	0.98	

Table 5.3: Parameters used in the simulation for the bicycle balancing and riding control problem

stressed with respect to exploitation. Finally the number of support vectors necessary to represents the approximate action value function with the set of parameters used in the approximation usually stays below 10% of the total number of collected samples which is also a indication of the quality of the approximation. The higher fraction of support vectors with respect to other benchmarks reveals the difficulty of the function approximation problem.

5.9 The Bike Balancing And Riding Control Problem

We consider two control problems related to a bicycle [71] moving at constant speed on a horizontal plane (fig. 5.21). For the bicycle balancing the agent has to learn how to balance the bicycle. For the bicycle balancing and riding he has not only to learn how to balance the bicycle but also how to drive it to a specific goal which in our simulation is located in a radius of ten meters around the point $(x_g, y_g) = (1000, 0)$. The system dynamics is composed of seven state variables $S \setminus S_T = \{(\omega, \dot{\omega}, \theta, \dot{\theta}, \psi) \in \mathbb{R}^5 \mid \quad \omega \in [-\omega_m, \omega_m] \quad \theta \in [-\theta_m, \theta_m] \quad \omega_m = \frac{\pi}{15} rad \quad \theta_m = \frac{\pi}{2.25} rad\}$ plus a terminal state S_T. Four states are related to the bicycle itself and three to the position of the bicycle on the plane. The state variables related to the bicycle are $\omega, \dot{\omega}$ (the angle and radial speed from vertical to the bicycle), $\theta, \dot{\theta}$ (the angle and radial speed the handlebars are displaced from normal). If $|\omega| > \omega_m$ the bicycle has fallen down reaching a terminal state S_T. The state variables related to the position of the bicycle on the plane are the coordinates (x_b, y_b) of the contact point of the back tire with the horizontal plane and the angle Ψ formed by the bicycle with the x-axis. The actions space $A = \{(u, T) \in \{-0.02, 0, 0.02\} \times \{-2, 0, 2\}\}$ is composed of 9 elements and depends on the the torque T applied to the handlebars and the displacement d of the rider. The noise in the system is a uniformly distributed term $\sigma_{d_t} = [-0.02, 0.02]$ added to action d. The system has a continuous time dynamics described by the following

differential equations:

$$\begin{aligned}
&\ddot{\omega} = I_{cp}^{-1}(Mgh\sin(\phi_t) - \cos(\phi_t)(I_{dc}\dot{\sigma}\dot{\theta}_t \\
&+sign(\theta_t)v^2(M_d r(invr_{f_t} + invr_{b_t}) + Mh\, invr_{CM_t}))) \\
&\ddot{\theta} = \frac{T - I_{dv}\dot{\sigma}\dot{\omega}}{I_{dl}} \quad \dot{\psi}_t = sign(\theta_t)v\, invr_{b_t} \\
&\dot{x}_{b_t} = v\cos(\psi_t) \quad \dot{y}_{b_t} = v\sin(\psi_t)
\end{aligned}$$

where

$$\begin{aligned}
&\phi_t = \omega_t + \frac{\arctan(d_t + \sigma_{d_t})}{h} \quad invr_{f_t} = \frac{|sin(\theta_t)|}{l} \\
&invr_{b_t} = \frac{|tan(\theta_t)|}{l} \quad invr_{CM_t} = \begin{cases} \frac{1}{\sqrt{(l-c)^2 + invr_{b_t}^{-2}}} & if \quad \theta_t \neq 0 \\ 0 \quad otherwise \end{cases}
\end{aligned}$$

The quantity used in the equations are

$$\begin{aligned}
&\dot{\sigma} = \frac{v}{r} \quad M = M_c + M_r \quad I_{dl} = \frac{1}{2}M_d r^2 \quad I_{dv} = \frac{3}{2}M_d r^2 \\
&I_{dc} = M_d r^2 \quad I_{cp} = \frac{13}{3}M_c h^2 + M_p(h + d_{CM})^2
\end{aligned}$$

with noise in the action σ_{d_t} drawn according to a uniform distribution. The various parameter with meanings are described in section 5.9 and in fig. 5.21, and the details of the dynamic can be found in [71]. The dynamic holds valid if $|\omega_{t+1}| \leq \omega_m$ while if $|\omega_{t+1}| > \omega_m$ the bicycle is supposed to have fallen down reaching a terminal state S_T. We suppose that the state variables (x_b, y_b) cannot be observed. Since these two state variables do not intervene in the dynamics of the other state variables nor in the reward functions considered. Hence they may be considered no relevant variables which does not make the control problem partially observable. The two optimal control problems have the same system dynamics and differ only by their reward function which for the bicycle balancing control problem is define hereafter

$$r(s_t, a_t) = \begin{cases} -1 \quad if \quad |\omega_{t+1}| > \omega_m \\ 0 \quad otherwise \end{cases}$$

while the reward function for the bicycle balancing and riding control problem is

$$r(s_t, a_t) = \begin{cases} -1 \quad if \quad |\omega_{t+1}| > \omega_m \\ c_r(\delta(\psi_t) - \delta(\psi_{t+1})) \quad otherwise \end{cases}$$

with $c_r = 0.1$ and $\delta(\psi) = \min_{k\in\mathbb{N}}|\psi + 2k\pi|$ representing some sort of distance between an angle ψ and the angle 0 and non-zero rewards are also observed when the bicycle is riding. The reward function for the bicycle balancing is such that zero rewards are always observed, except when the bicycle has fallen down and in that case the reward is equal to -1. For the bicycle balancing and riding control problem, a reward of -1 is also observed when the bicycle has fallen down. Positive rewards are therefore observed when the bicycle frame gets closer to the position $\psi = 0$ and negative rewards otherwise.

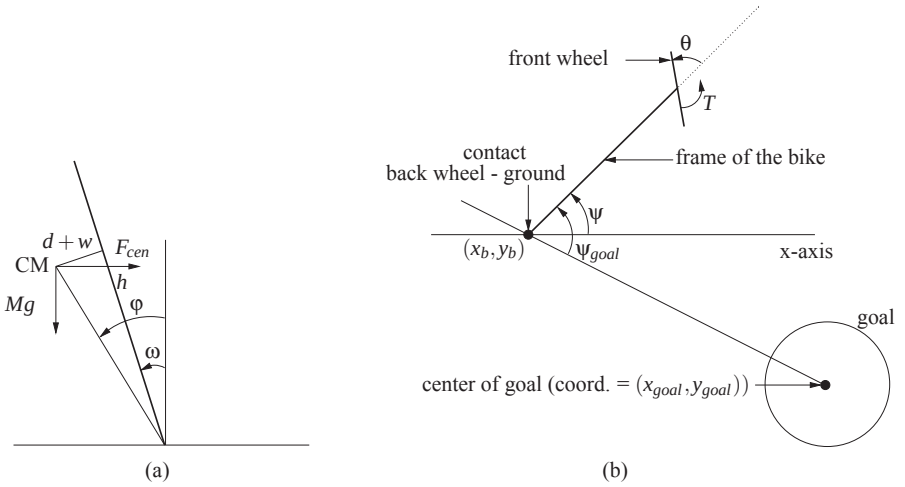

Figure 5.21: The bike control problem: Figure (a) represents the bicycle seen from behind where the thick line represents the bicycle. The center of mass of the bicycle+cyclist CM with height h from the ground, ω the angle from vertical to bicycle while ϕ represents the total angle of tilt of CM. Action d is agent displacement and w is some noise to simulate imperfect balance. Figure (b) represents the bicycle seen from above. θ is the angle the handlebars are displaced from normal, ψ the angle formed by the bicycle frame and the x axis and ψ_{goal} the angle between the bicycle frame and the line joining the back-wheel ground contact and the center of the goal. T is the torque applied by the cyclist to the handlebars. (x_b, y_b) is the contact point of the back-wheel with the ground (from [26])

With such a choice for the reward function, the optimal policy $\hat{\pi}$ tends to control the bicycle so that it moves to the right with its frame parallel to the x-axis. Such an optimal policy $\hat{\pi}$ can be used to drive the bicycle to a specific goal. If ψ_g represents the angle between the bicycle frame and a line joining the point (x_b, y_b) to the center of the goal (x_g, y_g) this is achieved by selecting at time t the action $\hat{\pi}(\omega_t, \dot{\omega}_t, \theta_t, \dot{\theta}_t, \psi_{g_t})$, rather than $\hat{\pi}(\omega_t, \dot{\omega}_t, \theta_t, \dot{\theta}_t, \psi_t)$. In this way, we proceed as if the line joining (x_b, y_b) to (x_g, y_g) were the x-axis when selecting control actions, which makes the bicycle moving towards the goal. See [26] for further discussion. The value of the discount factor γ has been chosen for both problems equal to 0.98. The dynamical system is integrated by using an Euler method with a $0.001s$ integration time step. To generate data samples we may consider episodes starting from the same initial state corresponding to the bicycle standing and going in straight line with $s_0 = (\omega_0, \dot{\omega}_0, \theta_0, \dot{\theta}_0, \psi_0) = (0, 0, 0, 0, \Psi_0)$ with a fixed value of Ψ or chosen at random $\Psi_0 \in [-\pi, \pi]$ and stopping when the bicycle leaves the region represented by $S \setminus S_T$ meaning a terminal state S_T.

5.9.1 Bike Balancing Control Problem

In our simulation of this benchmark using online $API - BRM_\varepsilon$ we run for $500s$ of simulated time collecting around 50000 samples. Run was split into separate learning episodes initiated at random initial states $s_0 = (0, 0, 0, 0, \Psi_0)$ with $\Psi_0 \in [-\pi, \pi]$ and stopping when a terminal state has been reached or otherwise after $1s$ (100 steps). Policy improvement

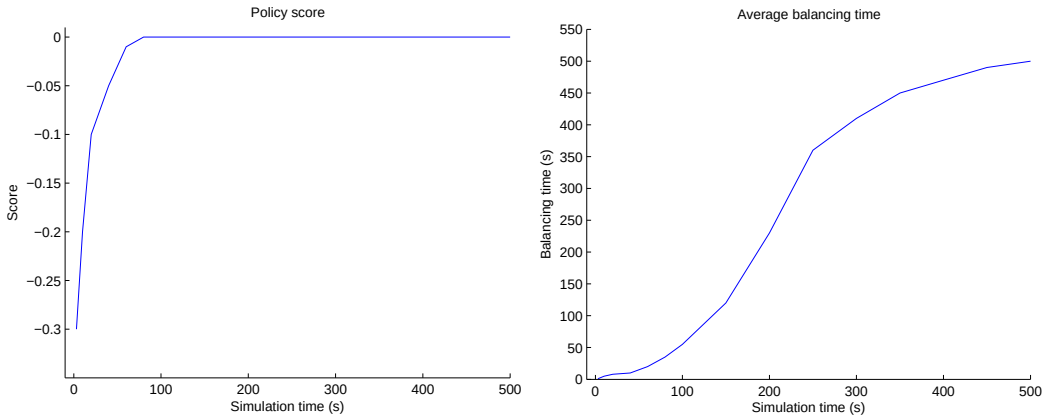

Figure 5.22: Bike balancing: performance of online $API - BRM_\varepsilon$ with $K_P = 10$ using Method-2

were performed once every $K_P = 10$ steps $(0.1s)$ using an $\varepsilon-$greedy policy with $\varepsilon_0 = 1$ and reaching a value of $\varepsilon_\infty = 0.1$ after $200s$. We also used an RBF kernel with parameters $\Sigma = I_7\sigma$ with $\sigma = 1.5$ and the regression parameters where chosen as $C = 10$ and $\varepsilon = 0.01$ selected using an grid search.

fig. 5.22 shows the performance of the final policy found by online $API - BRM_\varepsilon$ along the online learning process. The performance was measured evaluating the score over a grid of initial states $S_0 = \{(0,0,0,0,\Psi_0\}$ with $\Psi_0 \in [-\pi,\pi]$. In fig. 5.23 we report states and actions in subsequences of learning trials. Each trial lasts 50s max (5000 steps) considered as the minimum balancing to reach the goal. Using Method-3 (online growth) with a fixed initial state $S_0 = (0,0,0,0,\Psi_0)$ an optimal local approximation can be found in less then $50s$ of simulation time. In the lower part of fig. 5.23 we also show some of the trajectories during the learning process as well as the final one. Finally the number of support vectors necessary to represents the approximate action value function with the set of parameters used in the approximation usually stays below 5% of the total number of collected samples which is also a indication of the quality of the approximation.

5.9.2 Bike Balancing And Riding Control Problem

In our simulation of this benchmark using online $API - BRM_\varepsilon$ we run for $500s$ of simulated time collecting around 50000 samples. Run was split into separate learning episodes initiated at random initial states $s_0 = (0,0,0,0,\Psi_0)$ with $\Psi_0 \in [-\pi,\pi]$ and stopping when a terminal state has been reached or otherwise after $1s$ (100 steps). Policy improvement were performed once every $K_P = 10$ steps $(0.1s)$ using an $\varepsilon-$greedy policy with $\varepsilon_0 = 1$ and reaching a value of $\varepsilon_\infty = 0.1$ after $200s$. We also used an RBF kernel with parameters $\Sigma = I_7\sigma$ with $\sigma = 1.5$ and the regression parameters where chosen as $C = 10$ and $\varepsilon = 0.01$ selected using an grid search. As already mention the main difference in this case relies on the choice of the reward function which should be able to drive the bicycle to the right and parallel to the $x-axis$ as soon as the learned policy represents a good approximation of the optimal one. fig. 5.24 shows the performance of the final policy found by online $API - BRM_\varepsilon$ along the online learning process. The performance was measured evalu-

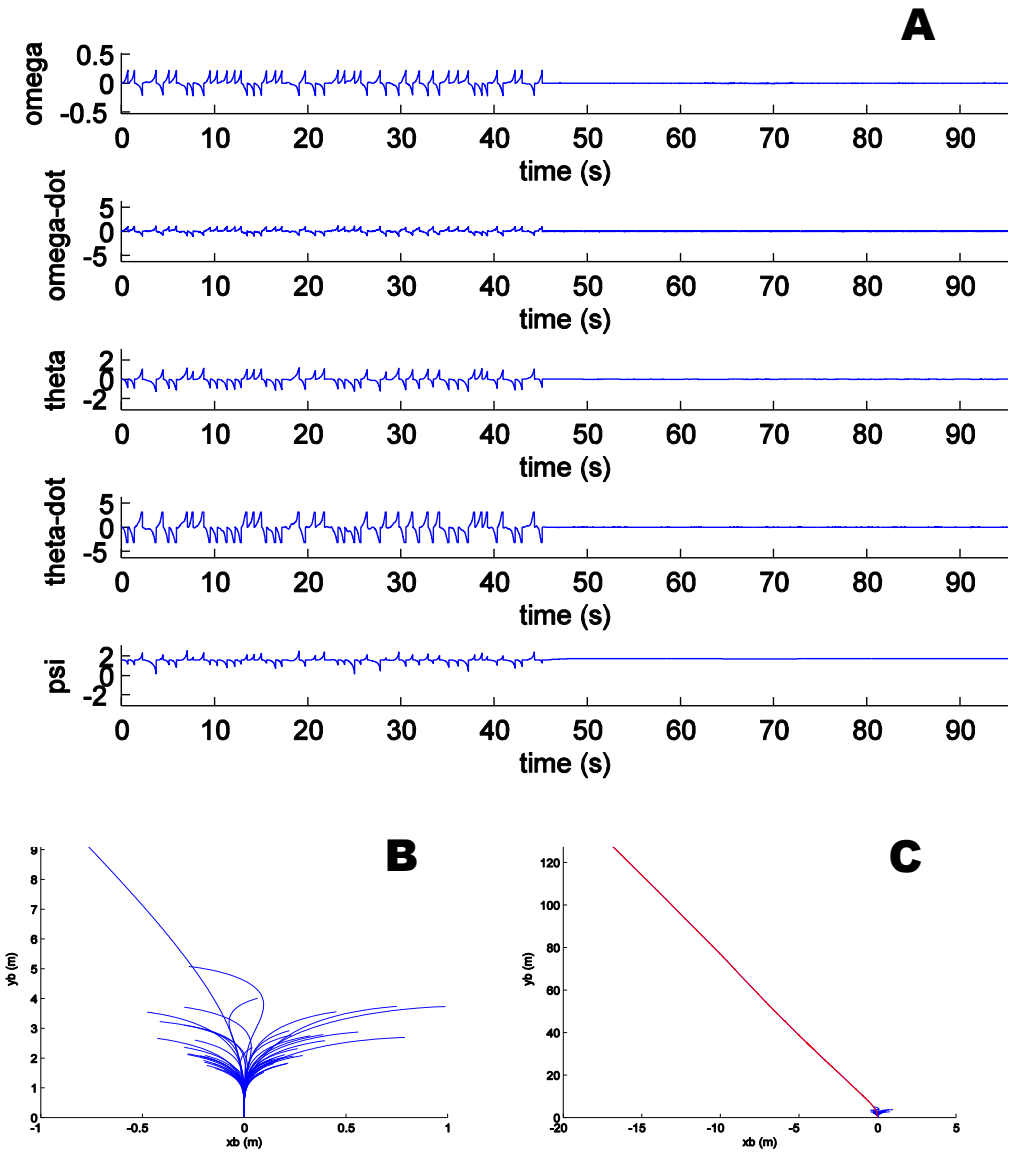

Figure 5.23: Bike balancing: (Upper A) States and actions in representative subsequences of learning trials. Each trial lasts 50s max (5000 steps) considered sufficient reach the goal. Using Method-3 (online-growth) with small perturbations of a fixed initial state $S_0 = (0,0,0,0,\pi/2)$ $API-BRM_\varepsilon$ may learn a local optimal policy in a few episodes ($50s$ of simulation time). (Lower) sketch of the trajectory (B zoom, C overall) in the time interval $(0,500s)$ for the bicycle on the (x_b,y_b) plane controlled by the final policy of $API-BRM_\varepsilon$

ating the score over a grid of initial states $S_0 = \{(0,0,0,0,\Psi_0\}$ with $\Psi_0 \in [-\pi,\pi]$. In fig. 5.25 we report states and actions in subsequences of learning trials. Each trial lasts $50s$ max (5000 steps) considered as the minimum balancing and riding to reach the goal. Using Method-3 (online growth) with a fixed initial state $S_0 = (0,0,0,0,\Psi_0)$ an optimal

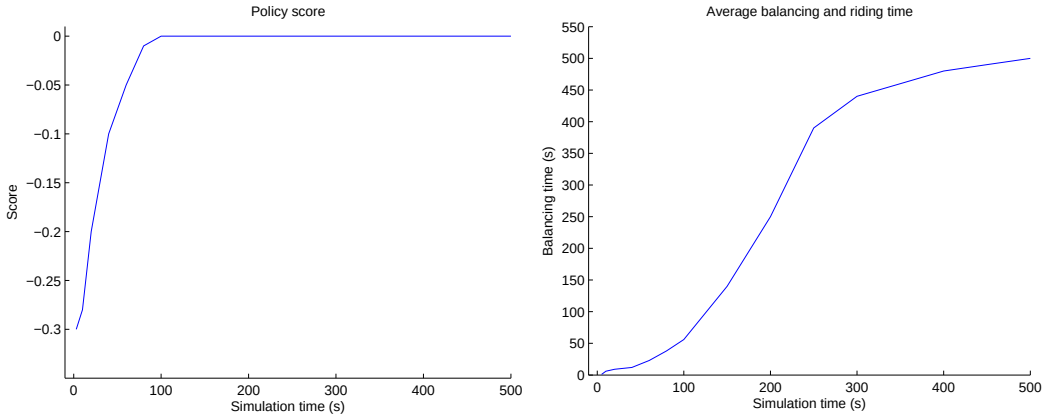

Figure 5.24: Bike balancing and riding: performance of online $API - BRM_\varepsilon$ with $K_P = 10$ using Method-2

local approximation can be found in less then $50s$ of simulation time. Since the bicycle rides at constant speed $v = 2.77\,ms^{-1}$ and the time step is $0.01s$ the bicycle has to cover a distance around $1278\,m$ before reaching the goal. Hence in the lower part of fig. 5.25 we show the learned trajectories and the final one which was able to reach the goal in less that 50000 time steps. As in [26] we also analyzed the influence of the parameters c_r, γ over the trajectories. Our analysis confirms that taking $c_r < 0.1$ the bicycle tends to turn more slowly and to take more time to reach the goal while choosing $c_r = 1$ trajectories reach rapidly to a terminal state. On the other hand taking the discount $\gamma = 0.95$, influences the trajectories obtained and the bicycle may crashes rapidly as smaller value of γ tends to increase the importance of short-term rewards over long-term ones. However it is still possible to reach the goal by modifying the parameters C, Σ, ε. Finally the number of support vectors necessary to represents the approximate action value function with the set of parameters used in the approximation usually stays below 5% of the total number of collected samples which is also a indication of the quality of the approximation.

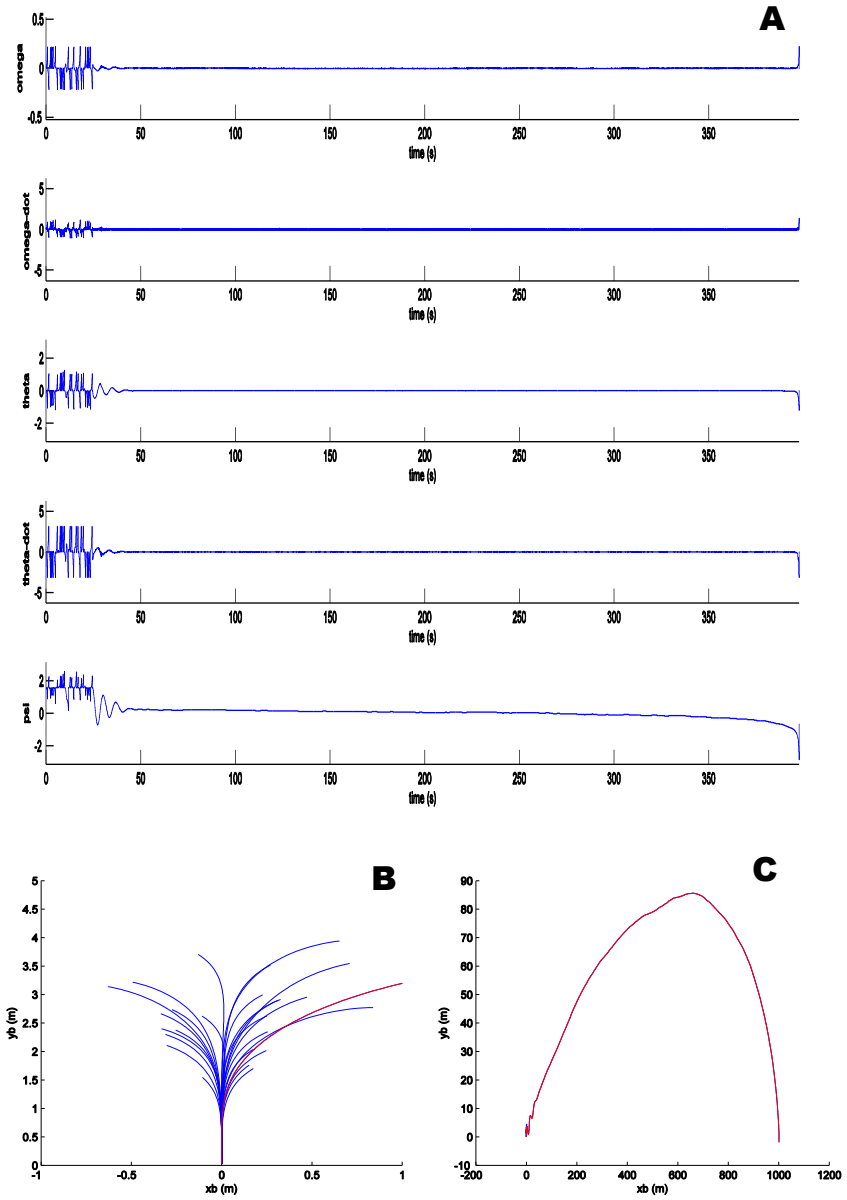

Figure 5.25: Bike balancing and riding: (Upper A) States and actions in representative subsequences of learning trials. Each trial lasts 50s max (5000 steps) considered sufficient reach the goal. Using Method-3 (online-growth) with small perturbations of a fixed initial state $S_0 = (0,0,0,0,\pi/2)$ $API - BRM_\varepsilon$ may learn a local optimal policy in a few episodes ($50s$ of simulation time). (Lower) sketch of the trajectory (B zoom, C overall) in the time interval $(0,500s)$ for the bicycle on the (x_b, y_b) plane controlled by the final policy of $API - BRM_\varepsilon$ reaching the goal located at $(x_b, y_b) = (1000,0)$

Conclusions and Future Work

Conclusions

Main contribution of this thesis are the theoretical and experimental analysis of a non-parametric approximation algorithm for the generalization problem in RL using PI and kernel methods. We developed a model free BRM approach called $API - BRM_\varepsilon$ able to find the optimal policy in continuous state RL problems, studied its theoretical properties and practical implementation issues. In particular, we demonstrated how the problem of finding the optimal policy solution minimizing the BR can be cast as a regression problem using SVR and an appropriate RKHS. The algorithm were proved to eventually converge to the optimal policy using β-mixing distributed data samples providing also a theoretical bound expressed in terms of number of events, capacity of the function space and number of PI steps. Some interesting properties of $API - BRM_\varepsilon$ algorithm are:

- $API - BRM_\varepsilon$ is quite efficient for problems where sampled experience is sparse. The algorithm is based on API, a very powerful framework met with success mostly among planning problems.It also open new research directions for the use of kernel based API in the context of learning.

- $API - BRM_\varepsilon$ is a model free algorithm need no to access to a model of the process but can be easily adapted to model based learning. In absence of generative models samples must be collected from the actual process in real-time then the algorithm may work in an online fashion. However when there is a generative model available the offline variant of the algorithm can be used.

- $API - BRM_\varepsilon$ is an API algorithms and compared to other API algorithms eliminates either the actor or the critic part of the actor-critic architecture. $API - BRM_\varepsilon$ makes a good use of function approximation implicitly constructing an approximate model using kernels. The algorithm place approximation directly in value function and use samples to perform directly the necessary operations (Bellman update and policy improvement) without going through any kind of model eliminating one potential source of error.

- $API - BRM_\varepsilon$ does not suffer from sub-optimality always finding the global optimal solution to the approximation problem. Also being a non-parametric learning method has the ability to automatically adapt to the complexity of the problem. Both properties rely on the use of SVR with $\varepsilon-$insensitive loss function, which is essentially a convex quadratic programming optimization problem.

- $API - BRM_\varepsilon$ uses incremental SVR which allows for the estimation and approximation of state action value functions in RL. PI can be done implicitly any time a new experience is obtained. Accordingly, computation lowers, while learning speed goes faster and generalization more effective than other existing methods. In fact $API - BRM_\varepsilon$ complexity strongly depends on the cost one has to pay in order to solve the SVR which is essentially a quadratic problem optimization. SVR can be solved in batch mode when the whole set of training sample are at disposal of the learning agents or incrementally (incremental SVM [70] and SVR by [57]). Incrementality enables the addition or removal of training samples very effectively.

- The approach taken by $API - BRM_\varepsilon$ makes full use of all samples at once either they are i.i.d. or strongly mixing. In contrast traditional RL algorithms use stochastic approximation where each sample is processed once and contributes with small changes. Usually a very large number of samples is required. Experience replay technique may partially solve the problem in traditional RL algorithms, storing samples through multiple passes over them. However, with $API - BRM_\varepsilon$ this is no longer necessary.

- Also in traditional RL algorithms, the accuracy of the approximation at different states or state-action pairs depends on the time, order state visitation. If the learning rate is high the algorithm risks oscillatory or divergent behavior. On the other hand, if the learning rate is kept small the learning becomes extremely slow. In contrast $API - BRM_\varepsilon$ has no risk of overshooting, oscillation, or divergence because it has no learning parameters to tune up and does not take gradient steps.

- $API - BRM_\varepsilon$ has an alternative representations from batch to incremental, from offline to online, showing effective generalization ability through SVR which makes use of the Structural Risk Minimization theory and his extension to mixing processes. Moreover it shows a theoretical bound on his performance and statistical convergence guarantee.

Future work

As possible future works we aim to follow some potential improvements of our method. In particular among other possibilities to use kernel based PI methods we foresee to investigate Bellman Advantage [7], [30] which uses a different approach to find the optimal policy within the approximation of the state action value function. Advantage learning seeks an approximation focusing on the minimal constraints that must be satisfied to ensure that the greedy policy computed from the action value function will be identical to the greedy policy computed from $\hat{Q}^\pi$ requiring that the backed up value of the optimal action a^* be greater than the backed up values of all other actions a in a given state s. The basic idea is to use preference learning and in particular a ranking SVM instead of a regression able to find a partial order in the action set for each state. Ranking SVM can be also implemented using icrementality as we did for SVR and we hope to raise a representation of the optimal policy using a reduced set of support vectors.

Another possibility could be to better analyze some of the assumptions we made in order to find the theoretical bound. From one side empirical processes and statistical learning theory with dependent data allow us to find the theoretical bound involving the β-mixing scenario. However, presently there is no simple way to actually estimate those coefficients from data. While general functional forms are known for some common classes of processes specific coefficients are generally beyond calculation. Nevertheless following the ideas in expressed in [58] we foresee to investigate the empirical process contained in the collected data checking for consistent estimators for the β-mixing coefficients based on a single stationary sample path.

Furthermore, to evaluate the convergence bound we asked for a stronger notion of approximation making assumptions on the probability distribution P and assuming that the inequality

$$\|f_{D,\lambda_n} - f^*_{\ell_\varepsilon,P}\|_{2,\nu_x} \leq c_P \Big[R_{\ell_\varepsilon,P}(f_{D,\lambda_n}) - R^*_{\ell_\varepsilon,P} \Big] = c_P \Big[\mathbb{E}[\ell_\varepsilon(y, f_{D,\lambda_n}(x)) | Z_n] - \mathbb{E}[\ell_\varepsilon(y, f^*_{\ell_\varepsilon,P}(x))] \Big] \quad (5.4)$$

holds true. As a results we need to further investigate under which conditions or limitations on P the we may correctly use the inequality 5.4 dealing with ℓ_ε losses.

In this work we managed continuous state and finite actions cMDPs as essential ingredients to achieve the asymptotic convergence bound. Besides in experiments performed in several benchmarks, it seems to be possible to extend the convergence result to cMDPs with continue action space. So further investigation in this sense might be promising and also interesting from a practical point of view.

Acknowledgments

During my thesis work I had the opportunity to meet several people whose help made my scientific experience and personal efforts fruitful and quite exciting.

First of all a cheerful word of thanks to all the member of the Knowledge Engineering and Machine Learning Group (KEMLG) sharing space and experiences during the past four years. In particular thanks to the senior members Profs. Kárina Gilbért, Miquél Sánchez and Jávier Vásquez for sharing their experience and ideas.

Then I'd like to thank you Prof. Jávier Bejár for the fruitful conversations he shared with me help me finding hints and suggestions useful form my work.

To Prof. Ulises Cortés, coordinator of the doctorate program, I would like to express my personal thanks and appreciations for the many stimulating discussions and the opportunity he gave me.

I am particulary grateful to prof. Csaba Szepesvári and Dr. Amir-massoud Farahmand which were willing to share their work on regularized API methods and also for the very helpful suggestion they give me.

I also have to thank Prof. Lucián Busoniú for his very helpful suggestions.

Finally I want to give a special word of thanks to Prof. Mario Martín for the amount of things he taught me and for the way he shared with me his knowledge and experience. Without his help and friendship this work would not have been possible.

Barcelona, Catalunya February 2015

Bibliography

Bibliography

[1] S. Abe. *Support vector machines for pattern classification*. Springer-Verlag, London, 2005.

[2] S. Abe and Y. Torii. Decomposition techniques for training linear programming svm. *Neurocomputing*, 72(4-6):973–984, 2009.

[3] Alejandro Gabriel Agostini. *Q-learning with degenerate function approximation*. PhD Thesis, UPC, Barcelona, 2011.

[4] Alejandro Gabriel Agostini and Enrique Celaya. Learning in complex environments with feature-based categorization. *In Proc. of the 8th Conference on Intelligent Autonomous Systems*, pages 446–455, 2004.

[5] Farahmand Amir-massoud and Csaba Csaba Szepesvári. Regularized least-squares regression: Learning from a β-mixing sequence. *Journal of Statistical Planning and Inference*, 142(2):493 – 505, 2012.

[6] András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning Journal*, 71:89–129, 2008.

[7] Leemon Baird. Advantage updating. *Report WL-TR-93-1146*, Wright Patterson(AFB), 1993.

[8] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.

[9] Lemon Baird. *Reinforcement learning trough gradient descent*. PhD Thesis, Carnegie Mellon University, Pittsburg, 1999.

[10] 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):834846, 1983.

[11] D. P. Bertsekas. *Dynamic Programming and Optimal Control vol. II*. Athena Scientific, Boston, 2007.

[12] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro Dynamic Programming*. Athena Scientific, Boston, 1996.

[13] Brett M. Bethke. *Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination*. PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, February 2010.

[14] J. Bi. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–1243, 2003.

[15] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[16] J. Burges and D. Crisp. Uniqueness of the svm solution. *Advances in Neural Information Processing Systems*, 12, 2000.

[17] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuska. Online least-squares policy iteration for reinforcement learning control. In US Baltimore, editor, *In Proceedings of American Control Conference ACC-10*, pages 486–491, 2010.

[18] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 2010.

[19] Lucian Busoniu, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, Robert Babuska, and Bart Schutter. Least-squares methods for policy iteration. In Marco Wiering and Martijn Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 75–109. Springer Berlin Heidelberg, 2012.

[20] M. Carrasco and X. Chen. Mixing and moment properties of various garch and stochastic volatility model. *Econometric Theory*, 18:17–39, 2002.

[21] Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19:1155–1178, 2007.

[22] Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[23] N. Christianini and J. Shawe-Taylor. *Support Vector Machines and Other kernel-based Learning Methods*. Cambridge University Press, 2000.

[24] Andreas Christmann and Ingo Steinwart. On robust properties of convex risk minimization methods for pattern recognition. *Journal of Machine Learning Research*, 5:1007–1034, 2004.

[25] Felipe Cucker and Steve Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39:1–49, 2002.

[26] Ernst Daniel, Geurts Pierre, and Whenkel Luis. Tree based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[27] G. B. Dantzig. Linear programming and extensions. *Princeton University Press*, page Princeton, 1963.

[28] Yu A. Davydov. Mixing conditions for markov chains. *Teor. Veroyatnost. i Primenen.*, 18:321–338, 1973.

[29] M. Deisenroth, J. Peters, and C. Rasmussen. Approximate dynamic programming with gaussian processes. *In Proceedings of the American Control Conference*, 2008.

[30] T. Diettrich and X. Wang. Batch value function approximation via support vectors. *NIPS*, MIT Press:14911498, 2001.

[31] T. Downs. Simplifications of support vector solutions. *Journ. of Machine Learning Resarch*, 2:293–297, 2001.

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

[33] Gennaro Esposito. *Lp-type methods for Optimal Transductive Suppport Vector Machines*. PhD Thesis, University of Perugia, 2011.

[34] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio. A unified framework for regularization networks and support vector machines. Technical report, MIT, Cambridge, MA, USA, 1999.

[35] Amir-massoud Farahmand. *Regularization in Reinforcement Learning*. PhD thesis, University of Alberta, 2011.

[36] Amir-massoud Farahmand, Rémi Munos, and Csaba Szepesvári. Error propagation for approximate policy and value iteration. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *NIPS*, pages 568–576. Curran Associates, Inc., 2010.

[37] G. Fung. Minimal kernel classifiers. *Journ. of Machine Learning Research*, 3:203–321, 2002.

[38] G. Gordon. Stable function approximation in dynamic programming. *Proceedings of 12th ICML*, pages 261–268, 1985.

[39] G. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie Mellon University, 1999.

[40] László Györfi, Michael Kohler, Adam Krzyzak, and Harro Walk. *A Distribution-Free Theory of Nonparametric Regression*. Springer, 2002.

[41] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[42] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 4:11851201, 1994.

[43] Tobias Jung and Daniel Polani. Least squares svm for least squares td learning. In *Proceedings of 17th European Conference on Artificial Intelligence*, pages 499–503, 2006.

[44] 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*, 2:749–756, 2009.

[45] R. L. Karandikar and M. Vidyasagar. Probably approximately correct learning with beta mixing input sequences, 2004.

[46] Michael Kohler, Adam Krzyzak, and Dominik Schfer. Application of structural risk minimization to multivariate smoothing spline regression estimates, 2000.

[47] D. Koller and R. Parr. Policy iteration for factored mdps. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 326–334, 2000.

[48] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[49] Pavel Laskov, Christian Gehl, Stefan Krüger, Klaus robert Mller, Kristin Bennett, and Emilio Parrado-Hern. Incremental support vector learning: Analysis, implementation and applications. *Journal of Machine Learning Research*, 7, 2006.

[50] Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, 13:3041–3074, 2012.

[51] Dong-Hyun Lee, Jeong-Jung Kim, and Ju-Jang Lee. Online support vector regression based actor-critic method. In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, pages 193–198, Nov 2010.

[52] Bo Liefeng, Wang Ling, and Licheng Jiao. Recursive finite newton algorithm for support vector regression in the primal. *Neural Computation*, 19(4):1082–1096, 2007.

[53] M. Maggioni and S. Mahadevan. Fast direct policy evaluation using multiscale analysis of markov diffusion processes. *ACM International Conference Proceeding Series*, 148:601–608, 2006.

[54] S. Mahadevan. Proto-value functions: Developmental reinforcement learning. *In International Conference on Machine Learning*, 2005.

[55] S. Mahadevan and M. Maggioni. Value function approximation with diffusion wavelets and laplacian eigenfunctions. *NIPS Conference 2005*, 2005.

[56] Odalric-Ambrym Maillard, Rémi Munos, Alessandro Lazaric, and Mohammad Ghavamzadeh. Finite-sample analysis of bellman residual minimization. In Masashi Sugiyama and Qiang Yang 0001, editors, *ACML*, JMLR Proceedings, pages 299–314. JMLR.org, 2010.

[57] Mario Martin. On-line support vector machine regression. In *Proceedings of the 13th European Conference on Machine Learning*, ECML '02, pages 282–294, London, UK, UK, 2002. Springer-Verlag.

[58] D. J. McDonald, C. Rohilla Shalizi, and M. Schervish. Estimating beta-mixing coefficients via histograms. *ArXiv e-prints*, September 2011.

[59] Ron Meir and Lisa Hellerstein. Nonparametric time series prediction through adaptive model selection. In *Machine Learning*, pages 5–34, 2000.

[60] Tony Mitchell. *Learning from Delayed Rewards*. McGraw-Hill Education, ISE Editions, 1997.

[61] Mehryar Mohri and Afshin Rostamizadeh. Stability bounds for stationary $\beta-$mixing and α-mixing processes. *Journal of Machine Learning Reasearch*, 11:789–814, 2010.

[62] 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.

[63] 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.

[64] Sayan Mukherjee. Statistical learning: Algorithms and theory. Technical report, Duke University, Duke University, USA, 2007.

[65] K.R. Muller and al. An introduction to kernel based learning algorithms. *IEEE Trans. on Neural Networks*, 12(2):181–202, 2001.

[66] Rémi Munos. Error bounds for approximate value iteration. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, pages 1006–1011. AAAI Press, 2005.

[67] G. Nehmauser. *Optimization, vol 1*. North-Holland, London, 1989.

[68] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2):161178, 2002.

[69] Singh S. P. and Sutton R. S. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[70] Thomaso Poggio and Geurth Cauvemberghs. Incremental and decremental support vector machine learning. *Adv. in Neural Inform. procesing*, MIT Press(13):409–415, 2001.

[71] Jette Randlov and Paul Alstrom. Learning to drive a bycicle using reinforcement learning an shaping. In *Proceeding of the fifth International Conference on Machine Learning*, pages 463–471, 1998.

[72] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Boston, 2006.

[73] 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*, ACM:317–328, 2005.

[74] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[75] 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*, page 30333038, 2009.

[76] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist sytems*. Technical Report CUED/F-INFENG-TR 166, Cambridge University, 1994.

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

[78] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In David P. Helmbold and Bob Williamson, editors, *COLT/EuroCOLT*, volume 2111 of *Lecture Notes in Computer Science*, pages 416–426. Springer, 2001.

[79] Rohan Shiloh Shah. *Support Vector Machines for Classification and Regression*. Master of Science Thesis, Montreal, Quebec, 2007.

[80] S. P. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[81] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Journal Statistics and Computing*, 14(3):199–222, 2004.

[82] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[83] Ingo Steinwart, Don Hush, and Clint Scovel. Learning from dependent observations. *J. Multivar. Anal.*, 100(1):175–194, January 2009.

[84] M. Sugiyama, H. Hachiya, C. Towell, and S. Vijayakumar. Geodesic gaussian kernels for value function approximation. *In Workshop on Information-Based Induction Sciences*, 2006.

[85] M. Sugiyama, H. Hachiya, C. Towell, and S. Vijayakumar. Value function approximation on non-linear manifolds for robot motor control. *In Proc. of the IEEE International Conference on Robotics and Automation*, 2007.

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

[87] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8*, MIT Press:10381045, 1996.

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

[89] R. S. Sutton, A. G. Barto, 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.

[90] P. J. Sweitzer and A. Seidmann. Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(6):568–582, 1985.

[91] C. Szepesvári and W. D. Smart. Interpolation-based q-learning. *In Proc. of the twenty-first ICML04*, page 791798, 2004.

[92] Gavin Taylor and Ronald Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1017–1024, 2009.

[93] J. Tobias and P. Daniel. Least squares svm for least squares td learning. *Proceedings of the 2006 conference on ECAI*, page 499503, 2006.

[94] M. A. Trick and S. E. Zin. Multilayer feedforward networks are universal approximators. *Macroeconomic Dynamics*, 1:255–277, 1997.

[95] Mario Valenti. *Approximate Dynamic Programming with Applications in Multi-Agent Systems*. PhD thesis, MIT, Boston, 2007.

[96] Sara van de Geer. *Empirical Processes in M-Estimation*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2009.

[97] Hado van Hasselt. *Insights in reinforcement learning*. PhD Thesis, Utrecht University, 2011.

[98] H. van Seijen and al. A theoretical and empirical analysis of expected sarsa. *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184, 2009.

[99] Harm van Seijen, Hado van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of expected sarsa. In *ADPRL 2009: Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184, March 2009.

[100] Robert J. Vanderbei. *Linear Programming*. Academic Press, New York, 1996.

[101] Vladimir Vapnik. *The nature of statistical learning theory*. Springer, New York, 1995.

[102] H. O. Wang, K. Tanaka, and M. F. Griffin. An approach to fuzzy control of nonlinear systems: stability and design issues. *Fuzzy Systems, IEEE Transactions on*, 4(1):14–23, Feb 1996.

[103] C. J. Watkins. *Learning from Delayed Rewards*. PhD thesis Kings College, Cambridge, 1989.

[104] Qiang Wu, Yiming Ying, and Ding-Xuan Zhou. Learning rates of least-square regularized regression. *Found. Comput. Math.*, 6(2):171–192, 2006.

[105] Bin Yu. Rates of convergence for empirical processes of stationary mixing sequences. *The Annals of Probability*, 22(1):94–116, 01 1994.

[106] W. D. Zhou and al. Linear programming support vector machines. *Pattern Recognition*, 35(12):2927–2936, 2002.

[107] Ding-Xuan Zhu and Steve Smale. Estimating the approximation error in learning theory. *Analysis and Applications*, 1-1:1–49, 2003.

Appendices

Appendix A

Statistical Learning

A.1 Introduction

In this chapter we briefly introduce some fundamentals about statistical learning theory largely referenced from [101], [25], [64], [79].

A natural approach to find for an optimal prediction function may be to define an optimization over a loss function $\ell(y, f(x))$ to each hypothesis in the hypothesis space $f \in \mathcal{H}$ able to measure the precision of admissible model functions over the training set $D = \{(\mathbf{x}_i, y_i) \in X \times Y\}_{i=1}^{n}$. The resulting space is defined as the loss class $\mathcal{L}(\mathcal{H}, \cdot) = \{\ell(y, f(x)) : f \in \mathcal{H}\}$ Performance of a test hypothesis can be evaluated by a fixed loss function over the whole observation space. We may assume the generation of observations distribution according to $P(\mathbf{x}, y)$ one has to integrate with losses with respect to this distribution. The *Expected Risk* (ER) is represented by the average loss produced by a fixed function over the observation space $X \times Y$ and integrated with respect to $P(\mathbf{x}, y)$

$$R_{\ell,P}(f) = \mathbb{E}[\ell(y, f(x))] = \int_{X,Y} \ell(y, f(x)) dP(\mathbf{x}, y) \tag{A.1}$$

A learning method minimize $R_{\ell,P}(f)$ for a fixed loss funtion over all measurable functions in the hypothesis space $\mathcal{H}$:

$$f^* = \arg\inf_{f \in \mathcal{H}} R_{\ell,P}(f) \tag{A.2}$$

and the optimal expected risk can be expressed as $R^*_{\ell,P} = \inf_{f \in \mathcal{H}} R_{\ell,P}(f)$. Finding f^* using section A.1 is not possible since $P(\mathbf{x}, y)$ is unknown. Usually one try to approximate the ER by modeling the distribution $P(\mathbf{x}, y) = \nu(\mathbf{x})P(y|\mathbf{x})$ using the Bayes rule estimating it from the training data and then integrating section A.1. An alternative is represented by estimating the *Empirical Risk* over the training data approximating the ER as

$$R_{\ell,D}(f) = \mathbb{E}_n[\ell(y, f(x))] = \frac{1}{n} \sum_{t=1}^{n} \ell(y_t, f(x_t)) \tag{A.3}$$

The methodology called *Empirical Risk Minimization* (ERM) tries to minimizes $R_{\ell,D}(f)$ looking for an hypothesis while minimizing ER. This is assumed to make prediction in

test samples coming from the same distribution $P(\mathbf{x}, y)$ as

$$f_n^* = \arg\inf_{f \in \mathcal{H}} R_{\ell,D}(f) \tag{A.4}$$

Usually a learning algorithm combines ERM with an exploration algorithm. ERM performance rely on the exploration algorithm inspecting the hypothesis space allowing the computation of the inf while it could find local minima. Consider measuring the deviation between ER of the hypothesis f_n^* obtained with ERM principle and his asymptotic behavior can be studied defining the *Sample Error* $R_S = R_{\ell,P}(f_n^*) - R_{\ell,P}^*$ Whenever until data is not available, empirical risk remains zero as such as long as the model function correctly predicts all components in the training set. Empirical risk is a function monotonically increasing function with the number of training data.

Analysis of the convergence for the empirical risk can be done introducing a probabilistic generalization bound.

Lemma A.1. ***(Chernoff's Inequality)*** *Given a fixed function $f \in \mathcal{H}$ and a loss function $A \leq \ell(y, \cdot) \leq B$, the probability of an absolute $\varepsilon-$difference (at least) in the middle empirical and expected risk is bounded*

$$\mathbb{P}\Big(R_{\ell,P}(f) - R_{\ell,D}(f) \geq \varepsilon\Big) \leq e^{-n\varepsilon^2/(B-A)^2} \tag{A.5}$$

and varies only with ε and n while the loss function bounds are A and B.

expressing essentially the law of large numbers (increasing n the bound $2e^{-2n\varepsilon^2}$ exponentially reduces implying an exponential convergence in probability). As a result, empirical risk can be considered a probabilistically unbiased estimate of ER

$$\begin{aligned} &\lim_{n\to\infty} R_{\ell,D}(f) \to^P R_{\ell,P}(f) \\ \Leftrightarrow \forall\, \varepsilon > 0 \,\exists\, \delta = e^{-n\varepsilon^2/(B-A)^2} \; s.t. \; &\mathbb{P}\Big(R_{\ell,P}(f) - R_{\ell,D}(f) \geq \varepsilon\Big) \leq \delta \\ \Leftrightarrow \forall\, \varepsilon > 0 \,\exists\, \delta = e^{-n\varepsilon^2/(B-A)^2} \; s.t. \; &\mathbb{P}\Big(R_{\ell,P}(f) - R_{\ell,D}(f) < \varepsilon\Big) > 1 - \delta \end{aligned} \tag{A.6}$$

with ε defining the confidence interval and δ corresponding confidence level. Notion of generalization comes from the proximity of the empirical risk to the expected risk which assures that as we minimize the empirical risk the closer we are to select a function having little ER. The *generalization error* can than be defined as the difference in the middle expected and empirical risk. In a learning method the potential of generalization depends on the ability to regulate the convergence rate which can be defined using a generalization bound. Expression in section A.1 defines what is called *Probably Approximately Correct Generalization* (PACG) occurring when the empirical risk is $\varepsilon-$approximately close to the expected risk with probability at least $1 - \delta$ and therefore the generalization error is *almost surely* close to zero. However, we cannot identify a model function satisfying lemma A.1 with large empirical risk using the ERM approach and model functions chosen by ERM sometimes are unable to generalize. The reason is because there can be infinitely many functions having minimal risk, in the middle of which a unique component having best generalization potential.

A.2 Uniformly Convergent Generalization Bounds

Due to the convergence weakness of the in section A.1, the function f^* minimizing ER it is not necessarily equivalent to the function f_n^* (for any n) minimizing the empirical risk. This means that there is a point-wise limit entailing that convergence rate may differ between the different functions in the function space $\mathcal{H}$. As a result, even for large value of n where the convergence for some subset of $\mathcal{H}$ it is possible, may exist functions not approaching their limits. Considering the worst case scenario for the convergence of the empirical risk, this means to extend the Chernoff's inequality. The way this could be done is to consider all the functions at once by bounding from above the sup of the generalization error as

$$\sup_{f\in\mathcal{H}}(R_{\ell,P}(f)-R_{\ell,D}(f))\leq\varepsilon \tag{A.7}$$

which is a generalization criteria stronger than lemma A.1. One has to contemplate the worst case for every function to form a uniform bound because it is unknown which function is optimal at during the learning process. Considering the union over H and the sub additivity coming from probability, allows to determine a generalization bound similar to Chernoff's inequality for all functions in $\mathcal{H}$ as:

$$\begin{aligned}&\mathbb{P}(\exists f\in\mathcal{H}\,(R_{\ell,P}(f)-R_{\ell,D}(f))\geq\varepsilon)=\mathbb{P}(\cup_{f\in\mathcal{H}}((R_{\ell,P}(f)-R_{\ell,D}(f))\geq\varepsilon))\\&\leq\textstyle\sum_{f\in\mathcal{H}}\mathbb{P}(R_{\ell,P}(f)-R_{\ell,D}(f))\geq\varepsilon)\leq\sum_{f\in\mathcal{H}}e^{-n\varepsilon^2/(B-A)^2}=|\mathcal{H}|e^{-n\varepsilon^2/(B-A)^2}\end{aligned} \tag{A.8}$$

It is possible to formulate section A.2 similarly to section A.2 taking into account that if the sup of the generalization error ε-bounded then all functions in $\mathcal{H}$ have to be also bounded by ε

$$\begin{aligned}&\mathbb{P}(\sup_{f\in\mathcal{H}}(R_{\ell,P}(f)-R_{\ell,D}(f))\leq\varepsilon)=\mathbb{P}(\,\forall f\in\mathcal{H}\,:\,(R_{\ell,P}(f)-R_{\ell,D}(f))\leq\varepsilon))=\\&1-\mathbb{P}(\exists f\in\mathcal{H}\,:\,(R_{\ell,P}(f)-R_{\ell,D}(f))\geq\varepsilon)>1-|\mathcal{H}|e^{-n\varepsilon^2/(B-A)^2}\end{aligned} \tag{A.9}$$

Putting $\delta=|\mathcal{H}|e^{-n\varepsilon^2/(B-A)^2}$ and solving for ε one may found

$$\varepsilon=\sqrt{\log\Big(\frac{|\mathcal{H}|}{\delta}\Big)\frac{(B-A)^2}{n}}$$

from which we resort the following Hoeffding's Inequality:

Lemma A.2. ***(Hoeffding's Inequality)*** *A distribution free bound quantifying the deviation over the empirical mean $R_{\ell,D}(f)$ from its expected value $R_{\ell,P}(f)$ over $\mathcal{H}$*

$$\sup_{f\in\mathcal{H}}(R_{\ell,P}(f)-R_{\ell,D}(f))\leq\sqrt{\log\Big(\frac{|\mathcal{H}|}{\delta}\Big)\frac{(B-A)^2}{n}} \tag{A.10}$$

holding with probability at least $1-\delta$ for a finite hypothesis space $|\mathcal{H}|<\infty$.

Convergence is exponentially fast while the generalization bound depends on the choice of function class $\mathcal{H}$ the dimension of the training set and a parameter $0\leq\delta\leq 1$. As

the inequality holds independently of $P(\mathbf{x}, y)$ it is distribution free. For the ERM model function f_n^* the bound holds with probability at least $1-\delta$ and being a uniform convergence bound, it holds with probability $1-\delta$ for any other hypothesis in the function space $\mathcal{H}$. Formally one may define the one sided uniform convergence as

$$\forall \varepsilon > 0 \exists N \in \mathbb{N} \, s.t. \forall n > N \, and \, \forall f \in \mathcal{H} \quad (R_{\ell,P}(f) - R_{\ell,D}(f)) < \varepsilon \tag{A.11}$$

where inequality lemma A.2 satisfies section A.2 being a uniform convergence bound. Taking a large value of $N(\varepsilon, \delta)$ such that Hoeffding's inequality is also bounded by ε for all $n > N(\varepsilon, \delta)$

$$\sqrt{\log\Big(\frac{|\mathcal{H}|}{\delta}\Big)\frac{(B-A)^2}{n}} < \varepsilon \tag{A.12}$$

Value of $N(\varepsilon, \delta)$ is the *sample complexity* of the learning algorithm. In other words it is an estimate of the necessary and sufficient quantity training examples for an algorithm to learn and generalize an unknown target concept. Now rather than of solving for ε one can evaluate n getting $n \geq (\frac{(B-A}{\varepsilon})\log(\frac{H}{\delta})$ with at least $n \leq N$ samples and probability at least $1-\delta$, the generalization error is $\varepsilon-$bounded for all functions.

In case all the function $f \in \mathcal{H}$ individually satisfies the Chernoff's inequality then they have to satisfy the Hoeffding's inequality collectively. There are two different way to tighten the generalization bound lemma A.2: by bounding the sensitivity of the model function or the capacity of the hypothesis space. Search of optimal model function is realized using the loss class defined over a given hypothesis space extendinf the notion of uniform convergence over the hypothesis space characterizing uniformly convergent loss classes as:

Definition A.1. ***(Uniform Glivenko-Cantelli Class*** *UGC) An UGC class of functions $\mathcal{L}(\mathcal{H}) = \{\ell(y, f) : f \in \mathcal{H}\}$ can be defined for a fixed loss function $A < \ell(y, f) < B$ such that the functions $f \in \mathcal{H}$ are integrable with respect to the probability measure $P(\mathbf{x}, y)$ and the one-sided uniform convergence is satisfied as*

$$\forall \varepsilon > 0 \lim_{n\to\infty} \mathbb{P}\Big(\sup_{\ell(y,f)\in\mathcal{L}(\mathcal{H})} (R_{\ell,P}(f) - R_{\ell,D}(f)) > \varepsilon\Big) = 0 \tag{A.13}$$

A sufficient and necessary condition for ERM consistency needs the class $\mathcal{L}(\mathcal{H})$ to be UGC.

A.3 Generalization and Consistency of ERM

A learning method is *consistent* for a function class $\mathcal{H}$ and distribution $P(\mathbf{x}, y)$ if the empirical risk for the model function f_n converges in probability to the ER

$$\begin{aligned} &\lim_{n\to\infty} R_{\ell,D}(f_n) \to^P \inf_{f\in\mathcal{H}} R_{\ell,P}(f) \\ \Leftrightarrow\ &\forall \varepsilon > 0 \exists \delta \, s.t. \mathbb{P}(|R_{\ell,D}(f_n) - \inf_{f\in\mathcal{H}} R_{\ell,P}(f)| \\ &\geq R_{\ell,D}(f_n) \to^P \inf_{f\in\mathcal{H}} R_{\ell,P}(f)| \geq \varepsilon) \leq \delta \end{aligned} \tag{A.14}$$

Consistency is defined through convergence in the empirical risk of the model function given by the learning algorithm where weaker generalization section A.1 is point-wise convergence over a fixed model and the stronger generalization lemma A.2 is uniform convergence over all models. As a result consistency depends on the learning algorithm while generalization does not. Consistency of ERM learning algorithm and uniform convergence and are essentially equivalent while the ER represents bound of the consistency convergence. Consistency is stronger than generalization but less strong than stronger generalization requiring a learning algorithm to guess on functions optimality in the hypothesis space before precisely estimating its ER. ERM performance is optimal if the function f_n^* minimizing the empirical risk is equivalent (in probability) to the function f^* minimizing the ER:

$$\exists N \in \mathbb{N}\, s.t. \forall n > N : f_n^* = \arg\inf_{f \in \mathcal{H}} R_{\ell,D}(f_n) = \arg\inf_{f \in \mathcal{H}} R_{\ell,P}(f) = f^* \tag{A.15}$$

For ERM consistency entails

$$\lim_{n \to \infty} \arg\inf_{f \in \mathcal{H}} R_{\ell,D}(f_n) =^P \arg\inf_{f \in \mathcal{H}} R_{\ell,P}(f) = f \tag{A.16}$$

whose choice in model function satisfies section A.1. Uniform convergence section A.2 in probability to zero of the generalization error is implied by consistency of the ERM learning algorithm (and vice versa). Moving from point-wise to uniform convergence allows for the consistency criteria section A.3 to be satisfied and UGC loss class is sufficient for consistency of ERM. Error of the optimal model function made by the empirical process is fixed by the sample error R_S. From generalization we have that $R_{\ell,D}(f_n^*)$ tends to $R_{\ell,P}(f_n^*)$ while from small sample error ensues that $R_{\ell,P}(f_n^*)$ goes to $R_{\ell,D}(f_n^*)$. Finally we may say that consistency essentially demands needs generalization of the empirical process with a sample error R_S tending to zero

$$R_{\ell,P}(f_n^*) \leq R_{\ell,D}(f_n^*) + \varepsilon \leq R_{\ell,D}(f^*) + \varepsilon \leq R_{\ell,P}(f_n^*) + 2\varepsilon \tag{A.17}$$

Exponentially fast amount of uniform convergence is close to half of the proportion at which there are guarantees that the sample error decrease to zero. The learning process rely on the distribution $P(\mathbf{x}, y)$ and the function space $\mathcal{H}$.

A.4 Vapnik-Chervonenkis Theory

As we assumed function space finite $|\mathcal{H}| < \infty$ (through the sub additivity of probability measures) Hoeffding's inequality is of limited use. It is possible to extend Hoeffding's bound for countably infinite hypothesis spaces. The idea is that one would like to study learning in an infinite uncountable function space where the union bound it is not defined. Cardinality of a function space represents the number of functions measuring its complexity. As one deals with infinite hypothesis spaces, it is necessary to consider some measures able to control the complexity of the hypothesis space related to the generalization error uniformly converging to zero.

Definition A.2. *(ε-cover) Given a function space $\mathcal{H}$ and some $\varepsilon > 0$ a subset $\mathcal{U} \subset \mathcal{H}$ is an ε-cover for $\mathcal{H}$ if*

$$\forall f \in \mathcal{H} \exists \hat{f} \in \mathcal{U} \, s.t. \, \|f - \hat{f}\| < \varepsilon \tag{A.18}$$

Representative of the set $\mathcal{U}$ are referred to as model functions. If for all $\varepsilon > 0$ $\mathcal{H}$ has a finite $\varepsilon-$cover then it is totally bounded. Together with Cauchy completeness this implies compactness (converse holds true). A bounded space has to be totally bounded (opposite is not necessarily implied).

Intuitively one may think that in any function space, functions are ε-close and it makes sense to assume they will behave similarly on a fixed training set. Generalization bound holding for one function will hold for the other. Measure defined above groups of functions together ensures that each one is totally contained into the ε-tube.

Definition A.3. *(ε-separation) Given a function space $\mathcal{H}$ and some $\varepsilon > 0$ a subset of l functions of $\mathcal{H}$ are $\varepsilon-$separated if*

$$\{f_i\}_{i=1}^{l} \subset \mathcal{H} \, satisfies \, \|f_i - f_j\| > \varepsilon \, \forall \, i \neq j \tag{A.19}$$

All hypothesis producing the same model function on a given training data set might be grouped into equivalence classes having the same empirical risk. VC-Entropy of $\mathcal{H}$ is the number of this equivalence classes with binary outputs $y \in \{+1, -1\}$. For regression this is called the *covering number* of $\mathcal{H}$.

Definition A.4. *(ε-covering Number) Let $\varepsilon > 0$ and $\mathcal{H}$ be a set of real-valued functions defined on X and ν_x a probability measure on X. Every finite collection of $N_\varepsilon = \{f_1, ..., f_{N_\varepsilon}\}$ defined on X with the property that for every $f \in \mathcal{H}$ there is a function $f' \in N_\varepsilon$ such that $\|f - f'\|_{p,\nu_x}^q \leq \varepsilon$ is called $\varepsilon-$cover of $\mathcal{H}$ w.r.t. $\|\cdot\|_{p,\nu_x}^q$. Let $\mathcal{N}_p(\varepsilon, \mathcal{H}, \|\cdot\|_{p,\nu_x}^q)$ be the size of the smallest $\varepsilon-$cover of $\mathcal{H}$ w.r.t. $\|\cdot\|_{p,\nu_x}^q$. If no finite $\varepsilon-$cover exists $\mathcal{N}(\varepsilon, \mathcal{H}, \|\cdot\|_{p,\nu_x}^q) = \infty$. Then $\mathcal{N}_p(\varepsilon, \mathcal{H}, \|\cdot\|_{p,\nu_x}^q)$ is called an $\varepsilon-$covering number of $\mathcal{F}$ and $\log \mathcal{N}_p(\varepsilon, \mathcal{H}, \|\cdot\|_{p,\nu_x}^q)$ is called the metric entropy of $\mathcal{H}$. Considering the empirical norm based $\|\cdot\|_{p,n}^q$ on the sequence of random variable $X_n = \{X_1, ..., X_n\}$ we may define the empirical covering number as $\mathcal{N}_p(\varepsilon, \mathcal{H}, \|\cdot\|_{p,n}^q)$*

It is smallest number of functions in $\mathcal{H}$ serving as an $\varepsilon-$cover for $\mathcal{H}$ while geometrically is the minimal number of balls in $\mathcal{H}$ with radius ε needed to cover $\mathcal{H}$. In a generalization bound the of the expected empirical covering number results in its dependence on $P(\mathbf{x}, y)$. Covering number for most compact real spaces of interest may be not calculable. Generally distribution independent bounds is practically impossible to evaluate.

Definition A.5. *(ε packing number) Given a function space $\mathcal{H}$ the packing number $\mathcal{D}(\varepsilon, \mathcal{H})$ is the maximal $l \in \mathbb{N}$ such that:*

$$\{f_i\}_{i=1}^{l} \subset \mathcal{H} \, satisfies \, \|f_i - f_j\|_p > \varepsilon \, \forall \, i \neq j \tag{A.20}$$

$\mathcal{D}(\varepsilon, \mathcal{H})$ is the maximal quantity of functions in $\mathcal{H}$ that can be $\varepsilon-$separated. To upper and lower bound the covering number using the packing number we may use the following inequalities

$$\mathcal{D}(2\varepsilon, \mathcal{H}) \leq \mathcal{N}_p(\varepsilon, \mathcal{H}, \|\cdot\|_{p,\nu_x}^q) \leq \mathcal{D}(\varepsilon, \mathcal{H}) \tag{A.21}$$

which can be used to get an approximation of the covering number.

If $\mathcal{H}_T$ is totally bounded have a finite minimal $\varepsilon-$cover $\mathcal{U} = \{\hat{f}_1, \hat{f}_2, ..., \hat{f}_c\} \subset \mathcal{H}_T$ (c covering number $\mathcal{N}(r_\varepsilon(\ell), \mathcal{H}_T)$). The radius $r_\varepsilon(\ell)$ of the covering depends on ε used to bound the sup of the generalization error and the loss function ℓ.

If we consider the ε-cover of the model function $\hat{f}_t \in \mathbb{C}_t$ satisfying $\cup_{t=1}^{c} \mathbb{C}_t = \mathcal{H}_T$ and a given function $f \in \mathbb{C}_t$ we may write

$$\begin{aligned} |R_{\ell,P}(f) - R_{\ell,D}(f) - R_{\ell,P}(\hat{f}_t) - R_{\ell,D}(\hat{f}_t)| &\leq |R_{\ell,P}(f) - R_{\ell,P}(\hat{f}_t)| + |R_{\ell,D}(\hat{f}) - R_{\ell,D}(\hat{f}_t)| \\ &\leq |\int (\ell(y,f) - \ell(y,\hat{f}_t)) dP(\mathbf{x},y)| + |\mathbb{E}_n[(\ell(y,f) - \ell(y,\hat{f}_t))]| \end{aligned} \tag{A.22}$$

A particular interesting class of loss functions (including $\varepsilon-$insensitive square loss and hinge loss) are Lipschitz class of functions satisfying the following inequality

$$\|\ell(\cdot, f_1) - \ell(\cdot, f_2)\|_\infty \leq L\|f_1 - f_2\|_\infty \tag{A.23}$$

for a given Lipschitz constant L. As a result, for any Lipschitz loss function, we can upper bound the integral in section A.4 as

$$\begin{aligned} \int (\ell(y,f) - \ell(y,\hat{f}_t)) dP(\mathbf{x},y) &\leq \int \|\ell(y,f) - \ell(y,\hat{f}_t)\|_\infty dP(\mathbf{x},y) \\ &\leq L\|f - \hat{f}_t\|_\infty \end{aligned} \tag{A.24}$$

and for the sum in section A.4

$$\mathbb{E}_n(\ell(y,f) - \ell(y,\hat{f}_t)) \leq L\|f - \hat{f}_t\|_\infty \tag{A.25}$$

It follows

$$|R_{\ell,P}(f) - R_{\ell,D}(f) - R_{\ell,P}(\hat{f}_t) - R_{\ell,D}(\hat{f}_t)| \leq 2L|\|f - \hat{f}_t\|_\infty| \tag{A.26}$$

Therefore using an arbitrarily set radius of the covering function of ε and Lipschitz loss functions we have $\forall\, f \in \mathbb{C}_t\ \|f - \hat{f}_t\|_\infty \leq r_\varepsilon(\ell) = \varepsilon/4L$ We see that the difference between the generalization errors of $\hat{f}_t$ and f is bounded by $\varepsilon/2$

$$\sup_{f \in \mathbb{C}_t} |R_{\ell,P}(f) - R_{\ell,D}(f) - R_{\ell,P}(\hat{f}_t) - R_{\ell,D}(\hat{f}_t)| \leq 2L|\|f - \hat{f}_t\|_\infty| \leq 2Lr_\varepsilon(\ell) \leq \varepsilon/2 \tag{A.27}$$

assuming largest generalization error of functions in $\mathbb{C}_t$ at least ε then the generalization error of the model function f_t must be at least $\varepsilon/2$:

$$\sup_{f \in \mathbb{C}_t} |R_{\ell,P}(f)) - R_{\ell,D}(f))| \geq \varepsilon \Rightarrow\ |R_{\ell,P}(f_t) - R_{\ell,D}(f))| \geq \varepsilon/2 \tag{A.28}$$

Hence, in probability this means

$$\mathbb{P}(\sup_{f \in \mathbb{C}_t} |R_{\ell,P}(f)) - R_{\ell,D}(f))| \geq \varepsilon) \leq \mathbb{P}(|R_{\ell,P}(f) - R_{\ell,D}(f)| \geq \varepsilon/2) \tag{A.29}$$

and we may use the Chernoff's inequality to $\hat{f}_t$ as

$$\mathbb{P}(|R_{\ell,P}(f) - R_{\ell,D}(f)| \geq \varepsilon/2) \leq 2e^{-n(\varepsilon/2)^2/(B-A)^2} \tag{A.30}$$

holding for all model functions while we may also use the union bound using $\cup_{t=1}^{c}\mathbb{C}_t = \mathcal{H}_T$ as well as section A.4 and section A.4 to get an exponentially fast converging PAC bound as

$$\mathbb{P}(\sup_{f\in\mathcal{H}_T}|R_{\ell,P}(f)-R_{\ell,D}(f)|\geq\varepsilon)\leq\sum_{t=1}^{|\mathcal{U}|}\mathbb{P}(\sup_{f\in\mathcal{C}_t}|R_{\ell,P}(f)-R_{\ell,D}(f)|\geq\varepsilon) \quad \text{(A.31)}$$

$$\leq\sum_{t=1}^{|\mathcal{U}|}\mathbb{P}(|R_{\ell,P}(\hat{f}_t)-R_{\ell,D}(\hat{f}_t)|\geq\varepsilon/2)\leq 2\mathcal{N}_p(r_\varepsilon(\ell),\mathcal{H}_T,\|\cdot\|^q_{p,\nu_x})e^{-n\frac{(\varepsilon/2)^2}{(B-A)^2}}$$

$$\simeq 2\mathbb{E}[\mathcal{N}_p(r_\varepsilon(\ell),\mathcal{H}_T,\|\cdot\|^q_{p,\nu_x}),\|\cdot\|_n]\,e^{-n\frac{(\varepsilon/2)^2}{(B-A)^2}} \quad \text{(A.32)}$$

(sup is taken over $\mathcal{H}_T$) and using section A.2 we finally get

$$\mathbb{P}(\sup_{f\in\mathcal{H}_T}|R_{\ell,P}(f)-R_{\ell,D}(f)|\leq\varepsilon)1-2\mathbb{E}[\mathcal{N}(r_\varepsilon(\ell),\mathcal{H}_T),\|\cdot\|_n]\,e^{-n\frac{(\varepsilon/2)^2}{(B-A)^2}} \quad \text{(A.33)}$$

which is similar to Hoeffding's inequality once we change $|\mathcal{H}|$ with the expected empirical covering number $\mathbb{E}[\mathcal{N}_p(r_\varepsilon(\ell),\mathcal{H}_T,\|\cdot\|^q_{p,\nu_x}),\|\cdot\|_n]$ Puting $\delta = 2\mathbb{E}[\mathcal{N}_p(r_\varepsilon(\ell),\mathcal{H}_T,\|\cdot\|^q_{p,\nu_x}),\|\cdot\|_n]\,e^{-n\frac{(\varepsilon/2)^2}{(B-A)^2}}$ and evaluating ε ensues

$$\sup_{f\in\mathcal{H}_T}|R_{\ell,P}(f)-R_{\ell,D}(f)|\leq 2(B-A)\sqrt{\frac{\log\left(2\mathbb{E}[\mathcal{N}_p(r_\varepsilon(\ell),\mathcal{H}_T,\|\cdot\|^q_{p,\nu_x}),\|\cdot\|_n]\right)+\log\left(1/\delta\right)}{n}} \quad \text{(A.34)}$$

With expression section A.4 and using the covering number (the other terns vanish to zero) we have the condition for uniform convergence necessary and sufficient for consistency

$$\lim_{n\to\infty}\frac{\log\left(\mathbb{E}[\mathcal{N}_p(r_\varepsilon(\ell),\mathcal{H}_T,\|\cdot\|^q_{p,\nu_x})]\right)}{n}=0,\,\forall\,\varepsilon \quad \text{(A.35)}$$

satisfied as long as the capacity of the hypothesis space increases polynomially at most in n while with exponentially increasing there is no convergence to zero. Sufficient criteria for UGC classes are compact hypothesis spaces (where finite cover always exists) and Lipschitz loss functions so that uniform convergence and consistency are implied.

Appendix B

Kernel Methods

B.1 Introduction

In this chapter we show how a certain class of kernel functions exist in all Hilbert spaces of real valued functions under a few simple conditions Kernel methods use kernel functions providing an implicit mapping of a training data set into a feature space $\mathcal{F}$ where regression or classification can be performed. A kernel function can be seen implicitly as an inner product between two data points into the feature space. Explicitly is a function evaluation for this data points in the input space X before applying any mapping. In the second part of the chapter we also briefly introduce some fundamentals about statistical learning theory and mixing processes. The material for this section was referenced from [79], [25], [64], [65] [23], [101], [45], [105].

B.2 Feature Space Induced By Kernel

The complexity of a training data set affects the performance of any learning algorithms. In given cases some classes of learning algorithms might not be able to appropriately learn a prediction function for a training data set. In this case to make the learning possible one has to manipulate the data. In some cases training data may have a format not suitable for the learning algorithm and is required a mapping of the data. Transform training data in feature space might offer a structure which could be exploited by the learning algorithm. A simple representation results from defining a non linear mapping function $\Phi(\cdot) \in \mathcal{H}$ over the inputs $\mathbf{x}_i \in X$ in the training set $D = \{(\mathbf{x}_i, y_i) \in X \times Y\}_{i=1}^{n}$ with $X \times Y \subset \mathbb{R}^r \times \mathbb{R}$ representing the data as the set of data mapped $D_\Phi = \{(\Phi(\mathbf{x}_i), y_i) \in X \times Y\}_{i=1}^{n}$ with $\Phi(\mathbf{x}_i) \in \mathcal{H}$, $y_i \in Y$ A potential computational problem may rise since Φ map elements into a feature space of possible infinite dimension. Kernel methods represent the data as a set of pairwise computations $\kappa : X \times X \to \mathbb{R}$ instead of map each training example $\mathbf{x}_i$ individually into features $\Phi(\mathbf{x}_i)$ using the map $\Phi : X \to \mathcal{F}$. A kernel function κ is defined over a space X which could be infinite. Consider observations in the training set D and define a finite kernel $\kappa_S : \mathbf{x}_i \times \mathbf{x}_j \to \mathbb{R} \quad \forall\, 1 \leq i \leq n$ which can be represented as $n \times n$ matrices with $\kappa_{ij} = \kappa_S(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}$. If inputs are defined in an inner product space, one may construct a function for linear comparison function by taking the dot product $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle_X$. The dot product evaluates the geometric angle between the

normalized vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ ($\|\mathbf{x}_i\| = \sqrt{\langle \mathbf{x}_i, \mathbf{x}_j \rangle} = 1$). A map Φ projecting the inputs into a dot product space must be applied in case the inner products are not well defined in the input space X. One may build the comparison function $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$.

The linear algebra associated by using finite kernel matrices over $D \times D$ comes in a finite dimensional vector space. Another possibility arises using kernels defined over a dense space and integral operator theory in an infinite dimensional function space have to be used as hypothesis function. In classification/regression tasks any possible hypothesis function has to be evaluated in a given data point requiring the function to be point-wise defined, meaning that all function evaluations exist in Y. $\mathbb{R}^X$ denotes the space of all real valued point-wise defined functions on the domain X. In RKHS is shown to hold that convergent sequences of functions are be point-wise convergent, which is not true in general for any Hilbert spaces (and in particular for $\mathcal{L}^2$).

$$\|f_n - f\|_{\mathcal{H}} \to 0 \Rightarrow \lim_{n \to \infty} f_n(\mathbf{x}) - f(\mathbf{x}) = 0. \forall \mathbf{x} \in X \tag{B.1}$$

and it can be shown that point-wise convergence in $\mathcal{H}$ implies the continuity of evaluation functionals on $\mathcal{H}$.

B.2.1 Hilbert Spaces

Hilbert spaces are complete inner product spaces with distances and angles that are well defined. Formally they are function spaces $\mathcal{H}$ having an inner product $\langle h, g \rangle$ defined for all $h, g \in \mathcal{H}$ such that the norm defined using the inner product $\|h\|_{\mathcal{H}} = \sqrt{\langle h.h \rangle_{\mathcal{H}}}$ completes the space. Given a closed or open subset N of a Hilbert space $\mathcal{H}$ one may define the complement orthogonal as the space: $N^{\perp} = \{h \in \mathcal{H} \,|\, \langle h, g \rangle = 0, \forall g \in \bar{N}\}$ The direct sum of these two spaces equals $\mathcal{H}$: $\mathcal{H} = N^{\perp} \oplus \bar{N} = \{g + h \,|\, g \in \bar{N}, h \in N^{\perp}\}$ Any function $h \in \mathcal{H}$ can be expressed as $h = g + l$ with $g \in N^{\perp}$ and $l \in \bar{N}$. Hilbert spaces $\mathcal{H}$ can be decomposed into two different closed subspaces. Infinite dimensional Hilbert spaces are similar to finite dimensional spaces thanks to the Zorn's Lemma and using the Gram-Schmidt orthogonalization. An orthonormal basis satisfies normalization, orthogonality and completeness so that every function in $\mathcal{H}$ can be represented uniquely as a linear combination of his elements For finite dimensional Hilbert spaces there exists a finite orthogonal basis so that every function in the space and every linear operator can be represented in matrix form. For infinite dimensional spaces cardinality is infinite while the span of the orthonormal must be dense in it. Hence, expressing every element in the space as a linear combination of given elements into the orthonormal basis it is not possible to. One assumes Hilbert spaces having countable orthonormal basis, separable and containing a dense subset with the entire space as closure.

B.2.2 Linear Functionals

A functional $\mathcal{F}$ is a real-valued function with arguments that are functions taken from space $\mathcal{H}$ as $\mathcal{F} : \mathcal{H}(X \to Y) \to \mathbb{R}$ An evaluation functional $\mathcal{E}_{\mathbf{x}}[f] : \mathcal{H}(X) \to Y$ simply evaluates a hypothesis function $f \in \mathcal{H}$ at some fixed point $\mathbf{x} \in X$ in the domain $\mathcal{E}_{\mathbf{x}}[f] = f(\mathbf{x})$. In the hypothesis space, point-wise convergence ensures continuity of the evaluation

functional:

$$f_n(\mathbf{x}) \rightarrow f(\mathbf{x}), \forall \mathbf{x}, \Rightarrow \mathcal{E}_{\mathbf{x}}[f_n] \rightarrow \mathcal{E}_{\mathbf{x}}[f], \forall \mathbf{x} \tag{B.2}$$

Linear functionals are defined over a linear space with elements which can be scaled and added through the functional: $\mathcal{F}(\alpha_1 h_1 + \alpha_2 h_2) = \alpha_1 \mathcal{F}(h_1) + \alpha_2 \mathcal{F}(h_2), \quad \forall\, h_1, h_2 \in \mathcal{H}$. If the set of functionals can be added and scaled they form a vector space $\mathcal{J}$ The *Null* space and *Image* space of the functional $\mathcal{F}$ are defined as: $\mathcal{N}_{\mathcal{F}} = \{h \in \mathcal{H} : \mathcal{F}(h) = 0\}$ $\mathcal{I}_{\mathcal{F}} = \{\mathcal{F}(h) : h \in \mathcal{H}\}$ are subspaces of the domain $\mathcal{H}$ and co-domain $\mathbb{R}$. For finite dimensional spaces states results $dim(\mathcal{H}) = dim(\mathcal{N}_{\mathcal{F}}) + dim(\mathcal{I}_{\mathcal{F}})$ A linear functional is bounded (then continuous) if $\exists\, \alpha \quad s.t. \quad |\mathcal{F}(h)| \leq \alpha \|h\|_{\mathcal{H}} \quad \forall\, h \in \mathcal{H}$. Any linear and continuous functional over an infinite dimensional Hilbert space may be decomposed as a linear combination of linear functionals as:

$$\mathcal{F}(h) = \sum_{i=1}^{\infty} \langle h, h_i \rangle \mathcal{F}(h_i) = \sum_{i=1}^{\infty} \langle h, h_i \mathcal{F}(h_i) \rangle \tag{B.3}$$

Given an Hilbert space $\mathcal{H}$ the associated inner product can be used to define a linear bounded functional: $\mathcal{F}_g(\cdot) = \langle g, \cdot \rangle_{\mathcal{H}} \in \mathcal{H}^*$ The functional defined in terms of a kernel function $\kappa(\mathbf{x}, \cdot) \in \mathcal{H}$ is given by $F_{\kappa}(\cdot) = \langle \kappa(\mathbf{x}, \cdot), \cdot \rangle_{\mathcal{H}} \in \mathcal{H}$ for some input vector $\mathbf{x} \in X$. Every element $g \in \mathcal{H}$ has a linear bounded functional corresponding in a dual space H^*: $g \mapsto \mathcal{F}_g(\cdot) = \langle g, \cdot \rangle_{\mathcal{H}} \in \mathcal{H}^*$ The dual space $\mathcal{H}^*$ of all linear bounded functionals on a Hilbert space $\mathcal{H}$ is a Hilbert space. This has a dual basis function of the orthonormal basis from the original space. The spaces $\mathcal{H}$ and its dual $\mathcal{H}^*$ are isomorphic.

A *functional* $\mathcal{F}$ is a real-valued function whose arguments are functions from space $\mathcal{H}$ defined as $\mathcal{F} : \mathcal{H}(X \rightarrow Y) \rightarrow \mathbb{R}$ An *evaluation functional* $\mathcal{E}_{\mathbf{x}}[f] : \mathcal{H}(X) \rightarrow Y$ evaluates a hypothesis function $f \in \mathcal{H}$ at some fixed point $\mathbf{x} \in X$ in the domain $\mathcal{E}_{\mathbf{x}}[f] = f(\mathbf{x})$. Point-wise convergence in the hypothesis space ensures the continuity of the evaluation functional:

B.3 Integrable Function Spaces

Consider the infinite dimensional space $L^2(\mathcal{Z})$ of all square integrable, real-valued and Lebesgue measurable functions on the measure space $(\mathcal{Z}, \Sigma, \mu)$ where Σ is a $\sigma-$algebra of subsets of $\mathcal{Z}$ and μ is a measure on Σ. Any open or closed subset of a finite dimensional real space $\mathcal{Z} = \mathbb{R}^n$ is Lebesgue measurable and therefore the space $L^2(\mathbb{R}^n)$ is infinite-dimensional. Considering an infinite dimensional measure space the Lebesgue measure is not well defined. Using the Lebesgue integral an inner product is given by: $\langle f, g \rangle_{L^2} = \int_{\mathcal{Z}} f(\mathbf{z}) g(\mathbf{z}) d\mu(\mathbf{z})$. One can define the norm in the space $L^2(\mathcal{Z})$ as as $\|f\|_{L^2} = \sqrt{\langle f, f \rangle_{L^2}}$ and having all functions that are square integrable on Z: $L^2(\mathcal{Z}) = \{f \in R^{\mathcal{Z}} : |f\|_{L^2} = \sqrt{\langle f, f \rangle_{L^2}} = (\int_Z f(\mathbf{z}) d\mu(\mathbf{z}))^{1/2}$ Hence, the function space $L^2(\mathcal{Z})$ is a Hilbert space and one can generalize the $L^2(\mathcal{Z})$ function space as follows: $L(\mathcal{Z})^p = \{f \in \mathbb{R}^{\mathcal{Z}} : \|f\|^p = \left(\int_{\mathcal{Z}} |f(\mathbf{z})|^p d\mu(\mathbf{z}) \right)^{1/p} < \infty\}$ and being only for $p = 2$ a Hilbert space, while for $p = 1$ the space contains all absolutely integrable functions on $\mathcal{Z}$: $L^1(\mathcal{Z}) = \{f \in \mathbb{R}^{\mathcal{Z}} : \|f\|_1 = \int_{\mathcal{Z}} |f(\mathbf{z})| d\mu(\mathbf{z}) < \infty\}$ while for $p = \infty$ the uniform norm is defined using the sup operator: $L^{\infty}(\mathcal{Z}) = \{f \in \mathbb{R}^{\mathcal{Z}} : \|f\|_{\infty} = \sup_{\mathbf{z} \in \mathcal{Z}} |f(\mathbf{z})| < \infty\}$. The norm in $L^p(\mathcal{Z})$ space is given

by: $\|\mathbf{z}\|_p = \left(\sum_{i=1}^{\infty} |z_i|^p\right)^{1/p}$ and convergence of the series depends on the vector $\mathbf{z}$ and the space ℓ^p is taken as the set of all vectors of infinite length that have a finite p-norm: $\ell^p(\mathcal{Z}) = \{\mathbf{z} \in Z : \|\mathbf{z}\|_p < \infty\}$.

A bounded continuous linear operator T is *compact* if the resulting image space is totally bounded when applied to the elements of any bounded subset of the domain. A bounded continuous linear operator $T : L^2(\mathbb{R}^X) \rightarrow L^2(\mathbb{R}^X)$ from one Hilbert space to another one is compact if for every bounded subset S of the domain $L^2(\mathbb{R}^X)$ the closure of the image space $(Tf) \ \bar{:} \ f \in S \subset L^2(\mathbb{R}^X)$ is compact

A linear operator T is *self-adjoint* if it is equal to its Hermitian adjoint T^* which satisfies $\langle Th, g\rangle = \langle h, T^*g\rangle$ and all the eigenvalues of a self-adjoint operator are real while in the finite dimensional case, a self-adjoint operator T is conjugate symmetric. Adjoint for every operator T exists defining a bounded continuous linear functional $\mathcal{F} : h \mapsto \langle g, Th\rangle, \forall h, g \in \mathcal{H}$ such that $\exists r_{\mathcal{F}} \in \mathcal{H}\, \mathcal{F}(h) = \langle g, Th\rangle = \langle r_{\mathcal{F}}, h\rangle, \forall \quad h \in \mathcal{H}$ and the adjoint $T^*g = r_{\mathcal{F}}$.

The following theorem can be used to characterize existence of image space basis for a self-adjoint and compact operator:

Theorem B.1. *(**Spectral Theorem**) Every compact, self-adjoint operator $T : \mathcal{H}_D \rightarrow \mathcal{H}_R$ when applied to a function in a Hilbert space $f \in \mathcal{H}$ has the following decomposition: $Tf = \sum_{i=1}^{\infty} \alpha_i P_{\mathcal{H}_i} \quad [f] \in \mathcal{H}$ α_i are complex numbers and each $\mathcal{H}_i$ is a closed subspace of $\mathcal{H}_D$ such that $P_{\mathcal{H}_i}[f]$ is the orthogonal projection of f onto $\mathcal{H}_i$.*

When the operator T induces $Tv_i = \lambda_i v_i$ where λ_i an eigenvalue of T and v_i eigenfunctions forming a complete and countable orthonormal basis of the image space.

A linear operator $T_K : L^2(X) \rightarrow L^2(X)$ is *integral* if the following transformation of one function space into another holds almost everywhere for all $f \in L^2(X)$ defined as $(T_K f)(\cdot) = \int_X \kappa(\cdot, \mathbf{x}) f(\mathbf{x}) d\mu(\mathbf{x})$ for a given kernel function $\kappa \in L^\infty(X \times X)$ where μ is the Lebesgue measure. In case of finite dimensional image space, the integral transformation can be expressed as a linear combination of a finite set of orthogonal basis functions as $(T_K f) = \sum_{i=1}^{b} \alpha_i f_i \quad s.t. \quad \langle f_i, f_j\rangle = 0 \quad \forall i, j < b$

A function $\kappa \in L^\infty(X \times X)$ is called a *positive kernel* if any quadratic form over it is positive:

$$\int_{X \times X} \kappa(\mathbf{x}, \mathbf{z}) v(\mathbf{x}) v(\mathbf{z}) d\mu(\mathbf{x}) d\mu(\mathbf{z})$$

For positive definite and finite kernel over all possible finite sets of vectors in the space $X \times X$ then the kernel is positive.

A *projection* $P : \mathcal{H} \rightarrow L$ over a vector space $\mathcal{H} = G \oplus L$ is a linear operator mapping points from $\mathcal{H}$ along the subspace G onto the subspace L. A projection is called orthogonal if its associated image space and null space are orthogonal complements.

The space of all real valued, continuous functions on the domain X that are differentiable up to k times is denoted by $C^k(\mathbb{R}^X)$. A function $\kappa \in C^0(X \times X)$ is continuous at a point $(\mathbf{b}, \mathbf{c}) \in X \times X$ if it satisfies: $\forall \varepsilon \, \exists \delta \, , \forall \mathbf{x}, \mathbf{s} \in X, \mathbf{b} - \delta < \mathbf{x} < \mathbf{b} + \delta, \quad \mathbf{c} - \delta < \mathbf{s} < \mathbf{c} + \delta, \Rightarrow \kappa(\mathbf{b}, \mathbf{c}) - \varepsilon < \kappa(\mathbf{x}, \mathbf{s}) < \kappa(\mathbf{b}, \mathbf{c}) + \varepsilon$ If the kernel κ is symmetric the integral operator T_K must be self-adjoint. Assume further that the kernel κ is continuous $\kappa \in C^0(X \times X)$ then results $\int_{X \times X} \kappa(\mathbf{x}, \mathbf{z})^2 d\mu(\mathbf{x}) d\mu(\mathbf{z}) < \infty$

For any bounded subspace of the domain $X \times X$ one can show the integral operator T_K is compact. As a result with symmetric, positive and square integrable kernel κ the resulting integral operator T_K is positive, self-adjoint and compact. Therefore from the Spectral Theorem follows that T_K must have a countable set of non-negative eigenvalues and the corresponding eigenfunctions $\{\lambda_1, \lambda_2, ...\}$ must form an orthonormal basis for $L^2(X)$.

Theorem B.2. *(**Mercer's Theorem**) For all positive* $(T_K f) = \sum_{i=1}^{b} \alpha_i f_i \quad s.t. \quad \langle f_i, f_j \rangle = 0 \quad \forall\, i, j < b$ *continuous and symmetric kernel functions* $\kappa \in L^2(X \times X)$ *over a compact domain* $X \times X$*, defining a positive, self-adjoint and compact integral operator* T_K *with an eigen-decomposition* $T_K v_i = \lambda_i v_i$ *the following five conditions are satisfied:*

1. $\{\lambda_1, \lambda_2, ...\} \in \ell^1$ *the sequence of eigenvalues are absolutely convergent*
2. $\lambda_i > 0, \forall\, i$ *the eigenvalues are strictly positive*
3. $v_i \in L^\infty(X)$ *the individual eigenfunctions are bounded.*
4. $\sup_i \|v_i\|_\infty < \infty$ *the set of all eigenfunctions is also bounded*
5. $\forall\, \mathbf{s}, \mathbf{x} \in X \quad \kappa(\mathbf{s}, \mathbf{x}) = \sum_{i=1}^{\infty} \lambda_i v_i(\mathbf{s}) v_i(\mathbf{x}) = \langle \Phi(\mathbf{s}), \Phi(\mathbf{x}) \rangle$ *converges absolutely for each* $(\mathbf{s}, \mathbf{x}) \in X \times X$ *converges uniformly for almost all* $(\mathbf{s}, \mathbf{x}) \in X \times X$.

B.4 Reproducing Kernel Hilbert Spaces

A RKHS is the hypothesis space for SVM. Points from the observation space are mapped into a RKHS having the necessary structure to define the regression or discrimination problem. In RKHS observations are mapped into features whose explicit are taken as a kernelized metric distance between any pairs of observations implicitly expressed as inner product. RKHS combines a Hilbert space with a positive kernel function.

Definition B.1. *(**Reproducing Kernel Hilbert Spaces**) Consider a subset of measurable functions* $\mathcal{F} : X \to \mathbb{R}$ *called the hypothesis space* $\mathcal{H}$*. A Reproducing Kernel Hilbert Space (RKHS)* $\mathcal{H} : X \to \mathbb{R}$ *is a point-wise Hilbert space defined in X with the inner product* $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ *characterized by a symmetric positive definite function* $\kappa : X \times Y \to \mathbb{R}$ *called reproducing kernel, continuous in X such that for each* $x \in X$ *the following reproducing property holds:*

$$\forall\, \mathbf{x} \in X \quad f(\mathbf{x}) = \langle f(\cdot), \kappa(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} \quad \kappa(\cdot, \mathbf{x}) \in \mathcal{H} \tag{B.4}$$

$\mathcal{H}$ *is the closure of the linear span of the set of functions* $\Phi_{span} = \{\Phi(\boldsymbol{x}) = \kappa(\cdot, \boldsymbol{x}) \quad \mathbf{x} \in X\}$ *considering the mapping* $\Phi : X \to C^0(X)$ *which denotes the function assigning the value* $\kappa(\cdot, \mathbf{x})$ *to* $\mathbf{x} \in X$ *and* $C^0(X)$ *the space of continuous functions on X with norm* $\|\cdot\|_\infty = \max\{...\}$.

Theorem B.3. *(**Moore-Aronszajn Theorem**) Every positive definite kernel* $\kappa(\cdot, \cdot)$ *on* $X \times X$ *is a reproducing kernel for a unique RKHS of functions in X. Conversely, every RKHS has an associated unique positive-definite kernel whose span is dense in it. A bijection exists between the set of all RKHS and the set of all positive kernel functions.*

Given any positive definite kernel function it is possible to build its associated RKHS (unique) and vice versa by using Mercer's Theorem. Consider the space spanned by the eigenfunctions of the eigen-decomposition of the integral operator defined using some kernel κ:

$$\mathcal{H}_K = \{f \in \mathbb{R}^X : f = \sum_{i=1}^{\infty} \alpha_i v_i, \|f\|_{\mathcal{H}_K} < \infty \quad \alpha_i \in \mathbb{R}, v_i \in L^{\infty}(X)\} \tag{B.5}$$

such that the dimension of the space $\mathcal{H}_K$ is equal to the number of non-zero eigenvalues of the integral operator. Defining the norm on this RKHS in terms of an inner product: $\langle f,g\rangle_{\mathcal{H}_K} = \langle \sum_{i=1}^{\infty} \alpha_i v_i, \sum_{i=1}^{\infty} \beta_i v_i \rangle_{\mathcal{H}_K} = \sum_{i=1}^{\infty} \alpha_i \beta_i / \lambda_i$ It then follows from Mercer's Theorem that the function $\kappa(\mathbf{x},\cdot)$ is a representer of the evaluation functional $\mathcal{E}_x$ and therefore reproduces in the RKHS $\mathcal{H}_K$:

$$\langle f(\cdot), \kappa(\mathbf{x},\cdot)\rangle_{\mathcal{H}_K} = \langle \sum_{i=1}^{\infty} \alpha_i v_i, \sum_{i=1}^{\infty} \lambda_i v_i(\mathbf{x}) v_i(\cdot)\rangle_{\mathcal{H}_K} = \sum_{i=1}^{\infty} \alpha_i v_i(\mathbf{x}) = f(\mathbf{x}) \tag{B.6}$$

Instead of minimizing the regularized risk functional defined as $R_{\ell,D}(f) = \frac{1}{n}\sum_{t=1}^{n} \ell(y_t, f(\mathbf{x}_t))$ over all functions in the hypothesis space

$$f^* = \arg\inf_{f\in\mathcal{H}} \left\{ \frac{1}{n} \sum_{t=1}^{n} \ell(y_t, f(\mathbf{x}_t)) + \lambda \|f\|^2_{\mathcal{H}_K} \right\} \tag{B.7}$$

we can minimize the following functional over all sequences of expansion coefficients $\{\alpha_1, \alpha_2, ...\}$:

$$f^* = \arg\inf_{\alpha} \left\{ \frac{1}{n} \sum_{t=1}^{n} \ell(y_t, \sum_{j=1}^{\infty} \alpha_j v_j(\cdot)) + \lambda \sum_{j=1}^{\infty} \frac{\alpha_j^2}{\lambda_j} \right\} \tag{B.8}$$

The number of expansion coefficients is equal to the number of non-zero eigenvalues which is also the dimension of the RKHS constructed. Since this number is possibly infinite the above optimization might be infeasible.

B.4.1 RKHS And Regularization

In a learning algorithm, the selected hypothesis needs to conform to the following criteria:

Definition B.2. *(**Well-Posed Optimization**) An optimization Ψ is well posed provided the solution $f^* : X \to Y$*

- *Exists: if the hypothesis space is too small then the solution may not exist $\exists\, \hat{f}^* \in \mathcal{H} : \hat{f}^* = \arg\inf_{f\in\mathcal{H}} \Psi$*
- *is Unique: if the hypothesis space is too large or the training set is too small then the solution may not be unique $\forall\, \hat{f}_1^*, \hat{f}_2^* \in \mathcal{H} : \hat{f}_1^*, \hat{f}_2^* = \arg\inf_{f\in\mathcal{H}} \Psi \Rightarrow \hat{f}_1^* = \hat{f}_2^*$*
- *is Stable: f^* depends continuously on the training set, so that slight perturbations in the training set do not affect the resulting solution, especially as the number of training examples gets larger.*

The model function of the learning algorithm must be generalizable and well-posed. The third criterion above is especially important as it relates to the generalization ability of a hypothesis. A stable transform is less likely to overfit the training set. ERM principle guarantees the existence of a solution assuming $\mathcal{H}$ is compact and the loss function ℓ is continuous. In general, neither of these conditions are satisfied. However, ERM does not guarantee the uniqueness nor the stability of the solution and the method is therefore ill-posed. We must use prior information to determine which solution of functions with minimal empirical risk is better for prediction. The question how to constrain the hypothesis space is answered by Occam's Razor stating that the simplest solution is often the best while all other variables remain constant.

Regularization attempts to provide well-posed solutions to a learning task by constraining the capacity of the hypothesis space eliminating complex functions unlikely to generalize. We may constrain the capacity of the hypothesis space (Ivanov Regularization) or implicitly optimize a parameter (Tikhonov Regularization) that regulates the capacity of the hypothesis space. Both methods are equivalent and make use of a measure of the smoothness of a function to regulate the hypothesis space.

A map $f : X \to Y$ is Lipschitz continuous if it satisfies: $|f(\mathbf{x_1}) - f(\mathbf{x}_2)| \leq M|\mathbf{x}_1 - \mathbf{x}_2|$ The smallest $M \geq 0$ that satisfies the above inequality for all $\mathbf{x}_1, \mathbf{x}_2 \in X$ is called the Lipschitz constant of the function. Every Lipschitz continuous map is uniformly continuous which is a stronger condition than simple continuity. Functions in a RKHS are Lipschitz continuous and the distance between two elements in the domain is given by the square of the difference of their kernelized positions.

Ivanov Regularization requires that all functions in the hypothesis space $f \in \mathcal{H}_T$ of which there might be an infinite number, exist in a T-bounded subset of a RKHS H_K :

$$\hat{f}^* = \underset{f \in H}{\arg\inf} R_{\ell,D}(f) \quad subject\,to \quad \|H\|_{\mathcal{H}_K} \leq \mathcal{T} \tag{B.9}$$

Another way to see why this works is to consider functions from two hypothesis spaces, one significantly less complex, smoother functions than the other

$$\mathcal{H}_{\mathcal{T}_i} = \{f : f \in \mathcal{H}_K \quad and \quad \|f\|^2_{\mathcal{H}_K} \leq \mathcal{T}_i\}, i \in \{1,2\}\, \mathcal{T}_1 \ll \mathcal{T}_2$$

Small perturbations in the training data cause prediction functions from the more complex class $\mathcal{H}_{\mathcal{T}_2}$ to fluctuate more whereas functions from the smoother class $\mathcal{H}_{\mathcal{T}_1}$ remains relatively stable. Bounded finite dimensional RKHS $\mathcal{H}_{\mathcal{T}_i}$ is a totally bounded space and hence must have a finite ε-cover which implies the covering number of $\mathcal{H}_{\mathcal{T}_i}$ may be used in deriving generalization bounds.

Tikhonov Regularization penalizes the complexity and instability of the hypothesis space in the objective function of the optimization instead of explicitly bounding it by some constant:

$$\hat{f}^* = \underset{f \in H, \lambda}{\arg\inf} \left\{ R_{\ell,D}(f) + \lambda \|f\|^2_{\mathcal{H}_K} \right\} \tag{B.10}$$

with λ a regularization parameter that has to be optimized to ensure good generalization performance, uniqueness of the solution and stability. Although the hypothesis space is a potentially infinite dimensional Hilbert function space, the solution of the Tikhonov optimization has the form of a finite basis expansion:

Theorem B.4. *(**Representer Theorem**) Consider the objective function of the Tikhonov Regularization method optimizing the sum of a loss function and a regularization term:*

$$f^* = \arg\inf_{f \in H} \left\{ \frac{1}{n} \sum_{t=1}^{n} \ell(y_t, f(\mathbf{x}_t)) + \Upsilon(\|f\|^2_{\mathcal{H}}) \right\} \tag{B.11}$$

If ℓ is a point-wise defined loss function and Υ monotonically increasing, the solution of the optimization exists and can be written as a linear combination of a finite set of functions defined over the training data; $f^* = \sum_{j=1}^{n} \alpha_j \kappa(\cdot, \mathbf{x}_j)$ with $\kappa(\cdot, \mathbf{x}_j)$ the representer of the functional $\mathcal{E}_{\mathbf{x}_j}[f] = f(\mathbf{x}_j)$ for all $f \in \mathcal{H}$.

As a consequence instead of searching the infinite dimensional hypothesis space H_K one must consider a finite dimensional subspace spanned by a finite number of basis functions.

B.4.2 The Kernel Trick

The kernel trick simplifies the quadratic optimizations used in SVM by replacing a dot product of feature vectors in the feature space with a kernel evaluation over the input space. Use of the reproducing kernel trick can be justified by mapping a vector $\mathbf{x} \in X$ in the input space to a vector in a feature RKHS. This can be derived from the Moore-Aronzajn construction of a RKHS with map defined as $\Phi : \mathbf{x} \to \kappa(\cdot, \mathbf{x}) \in \mathcal{H}$. Reproducing property is used to show that the inner product of two functions in the feature space is equivalent to a simple kernel evaluation $\langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle_{\mathcal{H}_K} = \kappa(\mathbf{x}, \mathbf{z})$

Applying the kernel trick in SVM gives the solution $f(\mathbf{x})$ of a kernelized classification or regression task expressed in terms of the weight vector $\mathbf{w}$ orthogonal to the separating hyperplane. This is computed using a constraint derived from the dual form of a quadratic optimization and expressed as a linear combination of support vectors which must be mapped into the feature space: The hypothesis function can be kernelized by mapping the test sample $\mathbf{x}_t$ in its definition using Φ and substituting a kernel evaluation with the dot product; Nevertheless, it is not necessary to know the structure of the implicit map (or feature space) associated with a kernel function and learning is performed implicitly in a complex non linear feature space while all computation are performed in the input space.

Appendix C

Support Vector Machines

C.1 Introduction

In this chapter we introduce some fundamental aspects about Support Vector Machines and Support Vector Regression in the original geometric formulation with main references to [101], [14], [22], [16] and [33].

Given a set of training data $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_l\}$ where $\mathbf{x}_i \in \mathbb{R}^d$, and their labels $\mathbf{Y} = \{y_1, ..., y_l\}$ such that $y_i \in \{\pm 1\}$, a *binary classifier* is a real valued function $f(\mathbf{x})$ where $\mathbf{x}$ is assigned to the positive class if $f(\mathbf{x}) \geq 0$ and otherwise to the negative class. The objective is to create a classifier assigning labels to set of interesting unlabeled points. Consider as hypothesis space a linear function of $\mathbf{x}$

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \sum_{i=1}^{r} w_i x_i + b \tag{C.1}$$

the $(\mathbf{w}, b) \in \mathbb{R}^r \times \mathbb{R}$ are parameters controlling the function. The *separating plane* of dimension $r-1$ defined by $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ is represented in fig. C.1 in the bidimensional case and splits the space into two half spaces which correspond to two distinct classes. An alternative is represented by non-linear separators which may be thought as linear separators in a different space. The complexity of learning the classifier function depends on the way it is represented, and so the difficulty of the learning task can vary accordingly. Quite often in ML changing the representation of the data might be a good strategy. This is equivalent to mapping the input space $\mathbf{X}$ to a new space $\mathbf{F}$:

$$\mathbf{X} \rightarrow \Phi(\mathbf{x}) = (\Phi_1(\mathbf{x}), ..., \Phi_m(\mathbf{x}))$$

where $\Phi : \mathbf{X} \subseteq \mathbb{R}^d \rightarrow \mathbf{F} \subseteq \mathbb{R}^m$ is a non linear map from the input space to some *feature space* (see fig. C.2) which might simplify the classification task.

Using the feature map Φ the learning machine learns a non-linear function hence we can write:

$$f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b = \sum_{i=1}^{m} w_i \Phi_i(\mathbf{x}) + b \tag{C.2}$$

Any SVM computes an hyperplane separating the training data in two classes. *SVM* classifier chooses the hyperplane maximizing some distance measure of the points on

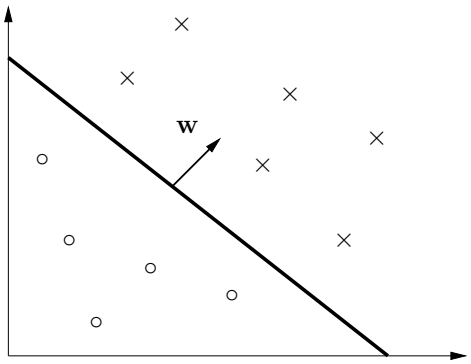

Figure C.1: A separating hyper-plane for a two dimensional training set controlled by $(\mathbf{w}, b)$)

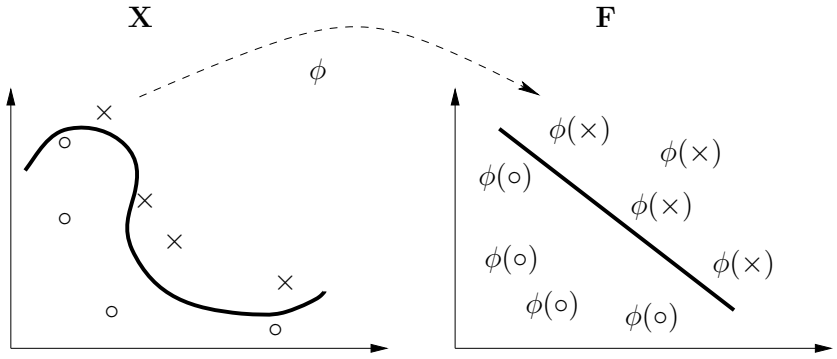

Figure C.2: Representation of the mapping from input to feature space

either side which is often referred as *the margin*. The closest instances of the training data lying to the hyperplane are called *support vectors*

Definition C.1. *Given an hyperplane* $(\mathbf{w}, b)$ *the functional margin of an example* $(\mathbf{x}_i, y_i)$ *is defined as:*

$$\gamma_i = y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$$

a positive margin $\gamma_i > 0$ implies the correct classification of the example $(\mathbf{x}_i, y_i)$. Hyperplane with largest margin can be considered the most confident for the separating task. *SVM* finds the maximal margin hyperplane optimizing the *geometric margin* γ_g corresponding to the functional margin

$$\gamma_{g_i} = y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)/\|\mathbf{w}\|$$

measuring the Euclidean distance of the points from the decision boundary in the input

space. Maximizing the geometric margin gives the following optimization problem:

$$\begin{aligned} \max_{\mathbf{w},b} \quad & \gamma_g \\ s.t. \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)/\|\mathbf{w}\| \geq \gamma_g \\ & i = 1, ..., l \end{aligned} \tag{C.3}$$

Instead of maximizing the margin typically one minimizes $\|\mathbf{w}\|^2$ as

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \|\mathbf{w}\|^2 \\ s.t. \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \\ & i = 1, ..., l \end{aligned} \tag{C.4}$$

following the fact that the geometric margin is $\frac{1}{\|\mathbf{w}\|^2}$ which can be easily derived considering the functional margin

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = \gamma_f$$

in two support vectors $\mathbf{x}^+$ and $\mathbf{x}^-$ from which one obtains respectively $+1(\langle \mathbf{w}, \mathbf{x}^+ \rangle + b) = \gamma_f$ and $-1(\langle \mathbf{w}, \mathbf{x}^- \rangle + b) = \gamma_f$. Then adding the two terms and substituting $\gamma_f = \gamma_g / \|\mathbf{w}\|^2$ results

$$2\gamma_g \|\mathbf{w}\|^2 = (\langle \mathbf{w}, \mathbf{x}^+ \rangle - \langle \mathbf{w}, \mathbf{x}^- \rangle)$$

from which fixing $\gamma_f = 1$ implies

$$(\langle \mathbf{w}, \mathbf{x}^+ \rangle - \langle \mathbf{w}, \mathbf{x}^- \rangle) = 2$$

and therefore $\gamma_g = 1/\|\mathbf{w}\|^2$.

SVM optimization problems are convex which guarantee a global optimum. Using the scalar product to define the margin $\|\mathbf{w}\|^2$ brings to the optimization problem

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \tfrac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle \\ s.t. \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \\ & i = 1, ..., l \end{aligned} \tag{C.5}$$

C.2 Optimization strategy

A strategy to minimize the problem section C.1 consists of finding its Lagrangian and applying the Karush-Kuhn-Tucker (*KKT*) [67] theorem which gives the necessary and sufficient conditions for the optimal solution of a constrained optimization:

Definition C.2. *Given the optimization problem defined as*

$$\begin{aligned} \min_{\mathbf{X}} \quad & f(\mathbf{X}) \\ s.t. \quad & g_i(\mathbf{X}) \leq 0, \, i = 1, ..., k \\ & h_i(\mathbf{X}) = 0 \quad j = 1, ..., m \end{aligned} \tag{C.6}$$

the Lagrangian primal is defined as:

$$\mathscr{L}(\mathbf{X},\alpha_i,\beta_i) = f(\mathbf{X}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{X}) + \sum_{j=1}^{m} \beta_j h_j(\mathbf{X}) \tag{C.7}$$

where α_i and β_j are the Lagrangian multipliers

KKT theorem gives necessary and sufficient conditions for the optimal solution.

Theorem C.1. *(KKT) Given the optimization problem*

$$\begin{aligned} \min_{\mathbf{X}} \quad & f(\mathbf{X}), \ \ \mathbf{X} \in \Omega \\ s.t. \quad & g_i(\mathbf{X}) \leq 0, \, i = 1,...,k \\ & h_i(\mathbf{X}) = 0 \quad j = 1,...,m \end{aligned} \tag{C.8}$$

with the convex domain $\Omega \subseteq \mathbb{R}^d$, $f \in \mathbb{C}^1$ convex and g_i, h_j affine, $\mathbf{X}^*$ is an optimum point if and only if there exist α_i^*, β_j^* such that the following conditions subsists:

$$\begin{aligned} & \frac{\partial \mathscr{L}(\mathbf{X}^*,\alpha_i^*,\beta_i^*)}{\partial \mathbf{w}^*} = 0, \\ & \frac{\partial \mathscr{L}(\mathbf{X}^*,\alpha_i^*,\beta_i^*)}{\partial \beta_i^*} = 0, \\ & \alpha_i^* g_i(\mathbf{X}^*) = 0, \ i = 1,...,k \\ & \beta_j^* h_j(\mathbf{X}^*) \leq 0, \ j = 1,...,m \\ & \alpha_i^* \geq 0, \ i = 1,...,k \end{aligned}$$

Applying the Lagrangian definition to the *SVM* problem we get:

$$\mathscr{L}(\mathbf{w},b,\alpha_i) = \frac{1}{2}\langle \mathbf{w},\mathbf{w}\rangle - \sum_{i=1}^{l} \alpha_i [y_i(\langle \mathbf{w},\mathbf{x}_i\rangle + b) - 1] \tag{C.9}$$

From the *KKT* conditions we have

$$\frac{\partial \, \mathscr{L}(\mathbf{w},b,\alpha_i)}{\partial \, \mathbf{w}} = 0 \longrightarrow \mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x_i}$$

$$\frac{\partial \, \mathscr{L}(\mathbf{w},b,\alpha_i)}{\partial \, b} = 0 \longrightarrow \sum_{i=1}^{l} \alpha_i y_i = 0$$

where $\alpha_i \geq 0$ are the Lagrangian multipliers. As a result the hypothesis $\mathbf{w}$ is expressed as linear combination of the support vectors ($\mathbf{x}_i$ with corresponding $\alpha_i > 0$). Putting the results in the Lagrangian after some algebra we obtain:

$$\mathscr{W}(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

Therefore instead of solving the primal problem we can solve its dual corresponding to maximizing the α_i in the Lagrangian primal subject to the necessary and sufficient conditions of the *KKT* theorem. The optimization problem becomes:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ s.t. \quad & \sum_{i=1}^{l} y_i \alpha_i = 0 \quad \alpha_i \geq 0, \quad i = 1, ..., l \end{aligned} \tag{C.10}$$

Once the solution α_i^* of the above problem is substituted in the corresponding equation of $\mathbf{w}^* = \sum_{i=1}^{l} y_i \alpha_i^* \mathbf{x}_i$ we get the geometrical margin $\gamma = \frac{1}{\|\mathbf{w}^*\|}$. The value b^* comes from the primal problem (b does not appear in the dual). From the constraints

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, \ \ i = 1, ..., l$$

substituting $\mathbf{w}^*$ for both labels we obtain $b \geq 1 - \langle \mathbf{w}^*, \mathbf{x}_i \rangle \ \ for \ \ y_i = 1$. Hence $b \leq -1 - \langle \mathbf{w}^*, \mathbf{x}_i \rangle \ \ for \ \ y_i = -1$ and the b^* giving the largest margin from both classes is

$$b^* = \frac{1}{2}[\max_{y_i=-1} (\langle \mathbf{w}^*, \mathbf{x}_i \rangle) - \min_{y_i=1} (\langle \mathbf{w}^*, \mathbf{x}_i \rangle)]$$

From the *KKT* complementary conditions must hold among the optimal values:

$$\alpha_i^* [1 - y_i(\langle \mathbf{w}^*, \mathbf{x}_i \rangle + b^*)] = 0, \ \ i = 1, ..., l$$

implying that only for $\mathbf{x}_i$ with functional margins equal to one are the corresponding $\alpha_i^* > 0$. Hence, all $\alpha_i^* = 0$ except those lying closest to the hyperplane.

This formulation *SVM* is called *hard margin* which can be applied when the classes are clearly separable. Whenever classes are not completely separable the *soft margin SVM* formulation one assumes that a certain amount of misclassification exists. Consider point $\mathbf{x}_i$ with label $y_i = +1$ but is instead classified as $y_i = -1$, hence $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = -\xi_i$ where $\xi_i > 0$. In order to satisfy the constraint $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ one has to add the distance ξ_i of that misclassified point from the true class to the left hand side of the constraint. For each misclassified point $\mathbf{x}_i$ a resulting slack variable ξ_i can be used and the generalized hyperplane problem is obtained with $p = 1, 2$ for linear and squared Hinge loss with $L_p(\xi_i) = \xi_i^p = \max(0, 1 - y_i f(x_i))^p$ as

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{p} \sum_{i=1}^{l} L_p(\xi_i) \\ s.t. \quad & y_i f(\mathbf{x}_i) \geq 1 - \xi_i \quad i = 1, ..., l \end{aligned} \tag{C.11}$$

with C a penalty parameter (correctly classified points shows $\xi_i = 0$). The way we penalize the misclassified points depends the loss function.

C.3 Using the Kernel trick

Learning a non-linear separators can be done by mapping the data into some feature space and then using a linear classifier in that space. Most of the time, this feature space has higher dimension than the original one, which suggests that finding a classifier will be

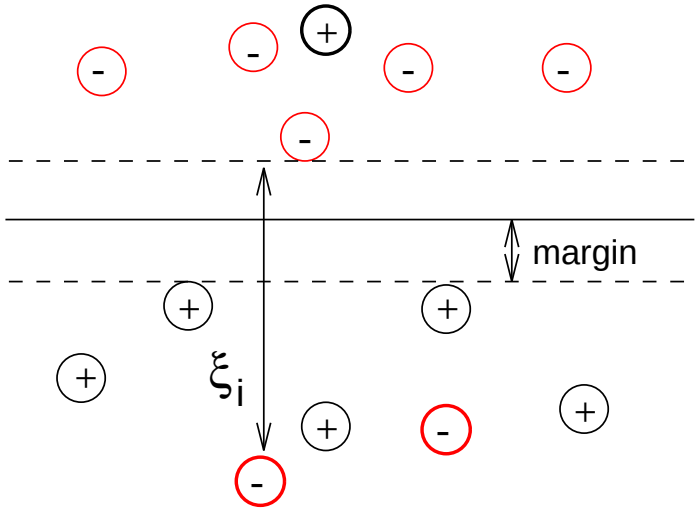

Figure C.3: Definition of a slack variable ξ_j for a misclassified point

computationally more expensive. However, by exploiting the fact that *SVM* finds a classifier based only on the inner products of the data points, it is sometimes possible to work in higher dimensions without paying computationally any price.

Given two points $\mathbf{x}, \mathbf{z}$ from the input space $\mathbf{X} \subseteq \mathbb{R}^d$, the kernel function $\mathbf{k}$ returning the inner product between their images into the feature space $\mathbf{F}$ can be written

$$\mathbf{k}(\mathbf{x}, \mathbf{z}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle, \ \forall \mathbf{x}, \mathbf{z} \in \mathbf{X} \tag{C.12}$$

where Φ is a mapping from input space to the feature space.

Defining $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$ and given a set of points $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_n)$ we call the $n \times n$ matrix $\mathbf{K}$ with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ the Gram matrix of k at $\mathbf{X}$.

Properties of kernel and RKHS have been presented in section B.4.2 and how to take profit of the Mercer's theorem. Mercer's theorem not only gives necessary and sufficient conditions for $\mathbf{k}$ to be a kernel, but also suggests a constructive way of obtaining features Φ from a given kernel $\mathbf{k}$.

Some commonly used kernels in SVM are:

- Identity Function (*IF*): $\mathbf{k}(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$
- Radial Basis Function (*RBF*): $\mathbf{k}(\mathbf{x}, \mathbf{z}) = e^{-\frac{1}{2\sigma} \|\mathbf{x} - \mathbf{z}\|^2}$
- Polynomial of degree p: $\mathbf{k}(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + 1)^p$

Accordingly, the decision function can be written as

$$f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x}) \rangle + b = \sum_{j=1}^{m} \sum_{i=1}^{l} \alpha_i y_i \Phi(\mathbf{x_i})_j \Phi(\mathbf{x})_j + b \tag{C.13}$$

which using the kernel definition becomes

$$f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i \mathbf{k}(\mathbf{x_i}, \mathbf{x}) + b \tag{C.14}$$

SVM task is to find the α_i corresponding to the maximal margin hyperplane in the feature space. The choice of feature space has an impact on the complexity of inner product. A good choice can allow the inner product to be easily computed. In particular, for some feature spaces, the inner product in feature space is a simple function of the point's coordinates in the original space.

C.4 Properties of the solution

Here we focusing on the properties of the solution fir the soft margin *SVM* using kernel. Assuming

$$f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b = \mathbf{w}^T \Phi(\mathbf{x}) + b$$

which can be summarized (see [1], [31], [37]) in:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + \tfrac{C}{p}\textstyle\sum_{i=1}^{l} L_p(\xi_i) \\ s.t. \quad & y_i f(\mathbf{x}_i) \geq 1 - \xi_i \quad i = 1, ..., l \end{aligned} \tag{C.15}$$

Similar to the linearly separable case, introducing the nonnegative Lagrange multipliers α_i and β_i we obtain the Lagrangian for the *L1 SVM* soft-margin *SVM*

$$\mathscr{L}(\mathbf{w}, b, \xi_i, \alpha_i, \beta_i) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i$$
$$-\sum_{i=1}^{l}\alpha_i[y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle + b) - 1 + \xi_i] - \sum_{i=1}^{l}\beta_i\xi_i$$

From the derivative of the *KKT* conditions we have

$$\begin{aligned} &\frac{\partial\, \mathscr{L}(\mathbf{w}, b, \xi_i, \alpha_i, \beta_i)}{\partial\, \mathbf{w}} = 0 \longrightarrow \mathbf{w} = \sum_{i=1}^{l}\alpha_i y_i \Phi(\mathbf{x_i}) \\ &\frac{\partial\, \mathscr{L}(\mathbf{w}, b, \xi_i, \alpha_i, \beta_i)}{\partial\, b} = 0 \longrightarrow \sum_{i=1}^{l}\alpha_i y_i = 0 \\ &\frac{\partial\, \mathscr{L}(\mathbf{w}, b, \xi_i, \alpha_i, \beta_i)}{\partial\, \xi_i} = 0 \longrightarrow \alpha_i + \beta_i = C \\ &\alpha_i[y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle + b) - 1 + \xi_i] = 0 \\ &\beta_i\xi_i = 0 \\ &\alpha_i \geq 0, \quad \beta_i \geq 0, \quad \xi_i \geq 0 \end{aligned}$$

and substituting those expressions we obtain the dual problem as:

$$\mathscr{W}(\alpha) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j}^{l}\alpha_i\alpha_j y_i y_j \mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) \tag{C.16}$$

subject to the constraints

$$\sum_{i=1}^{l} y_i\alpha_i = 0, \;\; 0 \leq \alpha_i \leq C, \;\; i = 1, ..., l \tag{C.17}$$

which is a convex quadratic programming problem. The only difference between *L1 SVM* soft-margin SVM and hard-margin SVM is that α_i cannot exceed C. From the *KKT* there are three cases for α_i:

- $\alpha_i = 0$ then $\xi_i = 0$ thus $\mathbf{x}_i$ is correctly classified.
- $0 < \alpha_i < C$ then $y_i f(\mathbf{x}_i) - 1 + \xi_i = 0$ and $\xi_i = 0$ therefore $y_i f(\mathbf{x}_i) = 1$ and $\mathbf{x}_i$ is a support vector. The support vectors with $0 < \alpha_i < C$ are called *unbounded*
- $\alpha_i = C$ then $\xi_i = 1 - y_i f(\mathbf{x}_i)$ and $\mathbf{x}_i$ is a *bounded* support vector. If $0 \leq \xi_i < 1$ $\mathbf{x}_i$ is correctly classified, if $\xi_i > 1$ $\mathbf{x}_i$ is misclassified.

Because α_i are nonzero, for unbounded support vectors $b = y_i - \langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle$ is satisfied, to ensure the precision of calculations the average of b that is calculated for unbounded support vectors.

Conversely for *L2SVM* the Lagrangian becomes:

$$\mathscr{L}(\mathbf{w}, b, \xi_i, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{i=1}^{l}\xi_i^2 - \sum_{i=1}^{l}\alpha_i[y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle + b) - 1 + \xi_i]$$

where we do not need to introduce the Lagrange multipliers associated with $\xi_i \geq 0$ due to the presence of ξ_i^2.

From the *KKT* conditions we have

$$\begin{aligned}
&\frac{\partial\, \mathscr{L}(\mathbf{w}, b, \xi_i, \alpha_i)}{\partial\, \mathbf{w}} = 0 \longrightarrow \mathbf{w} = \sum_{i=1}^{l}\alpha_i y_i \Phi(\mathbf{x_i}) \\
&\frac{\partial\, \mathscr{L}(\mathbf{w}, b, \xi_i, \alpha_i)}{\partial\, b} = 0 \longrightarrow \sum_{i=1}^{l}\alpha_i y_i = 0 \\
&\frac{\partial\, \mathscr{L}(\mathbf{w}, b, \xi_i, \alpha_i)}{\partial\, \xi_i} = 0 \longrightarrow C\xi_i = \alpha_i \\
&\alpha_i[y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle + b) - 1 + \xi_i] = 0 \quad i = 1, ..., l
\end{aligned}$$

from which we can understand that the optimal solution must satisfy either $\alpha_i = \xi_i$ or

$$\alpha_i[y_i(\sum_{j=1}^{l}\alpha_j y_j(\mathbf{k}(\mathbf{x}_j, \mathbf{x}_i) + \delta_{ij}/C) + b) - 1] = 0$$

Thus the value of the bias term b may be calculated for $\alpha_i > 0$ through

$$b = y_i - \sum_{j=1}^{l}\alpha_j y_j(\mathbf{k}(\mathbf{x}_j, \mathbf{x}_i) + \delta_{ij}/C)$$

which is different from the one of the *L1 SVM* but the decision function is the same. Now substituting the derived expression we obtain the dual objective function:

$$\mathscr{W}(\alpha) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j}^{l}\alpha_i\alpha_j y_i y_j(\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}/C) \tag{C.18}$$

subject to the constraints

$$\sum_{i=1}^{l} y_i \alpha_i = 0, \ 0 \leq \alpha_i, \quad i = 1, ..., l \tag{C.19}$$

which is again a convex *QP* problem. Therefore, for the *L1SVM* if we replace $\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j)$ with $\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}/C$ and remove the upper bound given by C for α_i, we obtain the *L2SVM*.

Because $1/C$ is added to the diagonal elements of the kernel matrix, the resulting matrix becomes positive definite. Thus the associated optimization problem is computationally more stable. Moreover, setting $C = \infty$ in *L1SVM* and *L2SVM*, we obtain the hard-margin *SVM*. Hence it is possible to show that for *L1SVM* and *L2SVM* as C approaches infinity, the sets of support vectors becomes the same, and the weight vectors in the feature space converges. One of the most important property offered by the solution of a *SVM* is that the solution is sparse in α_i i.e. many patterns are outside the margin area and the optimal $\alpha_i's$ are zero. Specifically the *KKT* conditions show that only such α_i connected to a training pattern $\mathbf{x}_i$, which is inside the margin area are non-zero. Without this sparsity property *SVM* learning would hardly be practical for large datasets.

C.5 Incremental SVM

Online learning is a learning scenario in which training data is provided one example at the time, opposed to the batch mode in which all examples are available at once. Such form of learning can be of any convenience when dealing with a very large or non stationary data and many problems in ML can be viewed as online ones. An exact solution to the problem of online SVM learning has been found by Cauwenberghs and Poggio [70] where the incremental algorithm updates an optimal solution after one training example is added (or removed).

The incremental algorithm for SVM ([70], [49]) updates of the trained SVM whenever a new sample x_c is added to the training set D and the basic idea is to change the coefficient α_c corresponding to the new sample x_c in a finite number of discrete steps until it meets the KKT conditions while ensuring that the existing samples in D continue to satisfy the KKT conditions at each step.

Moreover to build an exact on-line SVM one needs to define three primitive actions:

- add a new vector $D' = D \cup (x_c, y_c)$
- remove an existing vector $D' = D \setminus (x_c, y_c)$
- update an existing vector $D' = D \setminus (x_c, y_c) \cup (x_c, y_c')$

in each case the resulting incremental SVM should be the same that would be training from the scratch using the whole final set of data.

Considering the approximating function $f(x_i) = \langle \Phi(x_i), \mathbf{w} \rangle + b$ the SVM training problem is posed as the following quadratic optimization problem

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C\textstyle\sum_{i=1}^{n} \xi_i \\ s.t. \quad & y_i(\langle \Phi(x_i), \mathbf{w} \rangle + b) \geq 1 - \xi_i \\ & \forall\, i \in \{1, ..., n\} \end{aligned} \tag{C.20}$$

Introducing the Lagrange multipliers α and η we can write the corresponding Lagrangian as:

$$\begin{aligned} \mathscr{L}_P = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \eta_i \xi_i \\ - \sum_{i=1}^{n} \alpha_i [y_i(\langle \Phi(x_i), \mathbf{w} \rangle + b) - 1 + \xi_i] \end{aligned} \tag{C.21}$$

hence the dual variables have to satisfy positive constraints

$$\alpha_i, \eta_i, \geq 0 \tag{C.22}$$

It follows from the saddle point condition that the partial derivatives of $\mathscr{L}_P$ with respect to the primal variables $(\mathbf{w}, b, \xi)$ have to vanish for optimality:

$$\frac{\partial\, \mathscr{L}_P}{\partial\, \mathbf{w}} = 0 \quad \longrightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \Phi(x_i) \tag{C.23}$$

$$\frac{\partial\, \mathscr{L}_P}{\partial\, \xi_i} = 0 \quad \longrightarrow \quad C - \alpha_i - \eta_i = 0 \tag{C.24}$$

Substituting section C.5,section C.5,into section C.5 and eliminating the dual variables $\eta_i = C - \alpha_i$ yields to the dual optimization problem

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\textstyle\sum_{i,j=1}^{n} k(x_i, x_j) y_i y_j \alpha_i \alpha_j - \sum_{i=1}^{n} \alpha_i + b\sum_{i=1}^{n} y_i \alpha_i \\ s.t. \quad & \textstyle\sum_{i=1}^{n} y_i \alpha_i = 0 \quad \alpha_i \in [0, C] \quad \forall\, i \in \{1, ..., n\} \end{aligned} \tag{C.25}$$

where $k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$ and $Q_{ij} = y_i y_j k(x_i, x_j)$ is the kernel function. Moreover the Karush-Kuhn-Tucker (KKT) conditions require that at the point of the solution the product between the dual variables and constraints has to vanish:

$$\begin{aligned} \alpha_i [y_i(\langle \Phi(x_i), \mathbf{w} \rangle + b) + \xi_i - 1] = 0 \\ (C - \alpha_i)\xi_i = 0 \end{aligned} \tag{C.26}$$

Once the dual variables are known, the weight vector is given by

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \Phi(x_i) \tag{C.27}$$

and the function $f(x)$ can be computed using the support vector expansion

$$\begin{aligned} f(x) = \langle \Phi(x), \mathbf{w} \rangle + b &= \sum_{i=1}^{n} \alpha_i y_i \langle \Phi(x), \Phi(x_i) \rangle + b \\ &= \sum_{i=1}^{n} \alpha_i y_i k(x, x_i) + b \end{aligned} \tag{C.28}$$

The Lagrange formulation of section C.5

$$\begin{aligned}\mathscr{L}_D = \tfrac{1}{2}\textstyle\sum_{i,j=1}^{n} k(x_i,x_j)y_i y_j \alpha_i \alpha_j - \sum_{i=1}^{n} \alpha_i + b\sum_{i=1}^{n} y_i \alpha_i \\ -\textstyle\sum_{i=1}^{n} \delta_i \alpha_i + \sum_{i=1}^{n} \upsilon_i(\alpha_i - C)\end{aligned} \tag{C.29}$$

where δ_i, υ_i and b are the Lagrange multiplier. Optimizing this Lagrangian leads to the following KKT conditions:

$$\frac{\partial\, \mathscr{L}_D}{\partial\, \alpha_i} = 0 \quad \longrightarrow \quad \sum_{j=1}^{n} Q_{ij}\alpha_j + by_i - 1 - \delta_i + \upsilon_i = 0 \tag{C.30}$$

$$\delta_i \geq 0 \quad \delta_i \alpha_i = 0 \tag{C.31}$$

$$\upsilon_i \geq 0 \quad \upsilon_i(\alpha_i - C) = 0 \tag{C.32}$$

Defining $g_i = \sum_{j=1}^{n} Q_{ij}\alpha_j + by_i - 1$ is useful to define the margin function for the $i-th$ sample as

$$h(x_i) = y_i f(x_i) = \sum_{j=1}^{n} Q_{ij}\alpha_j + by_i \tag{C.33}$$

such that $g_i = h(x_i) - 1$ It is also worth noting that the following conditions descend from the KKT:

$$\delta_i = h(x_i) - 1 + \upsilon_i = g_i + \upsilon_i \quad \alpha_i[h(x_i) - 1 + \upsilon_i] = 0 \tag{C.34}$$

and combining $h(x_i)$ definition with the KKT conditions we can obtain:

$$\begin{cases} h(x_i) \geq 1 \;\; \alpha_i = 0 \\ h(x_i) = 1 \;\; 0 < \alpha_i < C \\ h(x_i) \leq 1 \;\; \alpha_i = C \end{cases} \tag{C.35}$$

There are three conditions in section C.5 and they can be identified with three subsets into which the samples in the training set D can be classified:

- the set E Error support vectors $E = \{i \,|\, \alpha_i = C\}$
- the set S Margin support vectors $S = \{i \,|\, 0 < \alpha_i < C\}$
- the set R Remaining samples $R = \{i \,|\, \alpha_i = 0\}$

In order to derive the incremental relations, consider x_c with the initially set value of $\alpha_c = 0$ and gradually change (increase o decrease) its value under KKT of section C.5 According to the KKT the incremental relation between $\Delta h(x_i)$, $\Delta\alpha_i$ and Δb is given by:

$$\Delta h(x_i) = Q_{ic}\Delta\alpha_c + \sum_{j=1}^{n} Q_{ij}\Delta\alpha_j + \Delta b \tag{C.36}$$

while from the equality condition coming from b we have

$$y_c\alpha_c + \sum_{i=1}^{n} y_i\alpha_i = 0 \tag{C.37}$$

Combining section C.5, section C.5 with definition of E,S and R sets we obtain:

$$\sum_{j\in S}^{n} Q_{ij}\Delta\alpha_j + \Delta b = -Q_{ic}\Delta\alpha_c \quad \forall\, i \in S \tag{C.38}$$

$$\sum_{j\in S}^{n} \Delta y_i \alpha_j = -\Delta y_c \alpha_c$$

Now if we define the index of the samples in the S set as $S = \{s_1, s_2, ..., s_{l_s}\}$ the section C.5 can be represented in matrix form as

$$\begin{bmatrix} 0 & 1 & ... & 1 \\ 1 & Q_{s_l s_1} & ... & Q_{s_l s_{l_s}} \\ . & . & . & . \\ 1 & Q_{s_{l_s} s_1} & ... & Q_{s_{l_s} s_{l_s}} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\alpha_{s1} \\ . \\ \Delta\alpha_{s_{l_s}} \end{bmatrix} = - \begin{bmatrix} 1 \\ Q_{s_1 c} \\ . \\ Q_{s_{l_s} c} \end{bmatrix} \Delta\alpha_c \tag{C.39}$$

that is

$$\begin{bmatrix} \Delta b \\ \Delta\alpha_{s_1} \\ . \\ \Delta\alpha_{s_{l_s}} \end{bmatrix} = \hat{\beta}\Delta\alpha_c \tag{C.40}$$

where

$$\hat{\beta} = \begin{bmatrix} \beta \\ \beta_{s_1} \\ . \\ \beta_{s_{l_s}} \end{bmatrix} = -\mathbf{R} \begin{bmatrix} 1 \\ Q_{s_1 c} \\ . \\ Q_{s_{l_s} c} \end{bmatrix} \tag{C.41}$$

where

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & ... & 1 \\ 1 & Q_{s_l s_1} & ... & Q_{s_l s_{l_s}} \\ . & . & . & . \\ 1 & Q_{s_{l_s} s_1} & ... & Q_{s_{l_s} s_{l_s}} \end{bmatrix}^{-1} \tag{C.42}$$

Define the non support vector set $N = E \cup R = \{n_1, ..., n_{l_s}\}$ and combining the above equations leads to:

$$\begin{bmatrix} \Delta h(x_{n_1}) \\ \Delta h(x_{n_2}) \\ . \\ \Delta h(x_{n_{l_s}}) \end{bmatrix} = \hat{\gamma}\Delta\alpha_c \tag{C.43}$$

where we define $\hat{\gamma}$ as

$$\hat{\gamma} = \begin{bmatrix} Q_{n_1 c} \\ Q_{n_2 c} \\ . \\ Q_{s_{l_s} c} \end{bmatrix} + \begin{bmatrix} 0 & 1 & ... & 1 \\ 1 & Q_{n_1 s_1} & ... & Q_{n_l s_{l_s}} \\ . & . & . & . \\ 1 & Q_{n_{l_s} s_1} & ... & Q_{n_{l_s} s_{l_s}} \end{bmatrix} \hat{\beta} \tag{C.44}$$

Given $\Delta\alpha_c$ one can update $\alpha_i, i \in S$ and b according to section C.5 and update $\Delta h(x_i), i \in N$ according to section C.5. Moreover section C.5 suggests that $\alpha_i, i \in N$ and $h(x_i), i \in S$ are constant if the set S stays unchanged. Hence it is only necessary to solve the problem on how to find the appropriate value $\Delta\alpha_c$.

All the above incremental equations are valid while the vectors do not migrate from set R, E or S to another one. This suggest a way to cope with the problem of finding the correct $\Delta\alpha_c$ choosing it to be the largest value that either can maintain the set S unchanged or eventually leads to the termination of the incremental algorithm.

As a matter of fact the first step is to determine if the change $\Delta\alpha_c$ according to the section C.5

$$sign(\Delta\alpha_c) = sign(y_c f(x_c)) = sign(h(x_c)) \geq 0 \tag{C.45}$$

Then eventually one has to find out the bound on $\Delta\alpha_c$ imposed by each sample in the training set D'. Considering that $\Delta\alpha_c > 0$ for a new sample (x_c, y_c) we have two cases:

- [1] $h(x_c)$ changes from $h(x_c) < 1$ to $h(x_c) = 1$ then the new sample is added to the set S and the algorithm terminates
- [2] if α_c increases from $\alpha_c < C$ up to $\alpha_c = C$ the new sample is added to the set E and the algorithm terminates

for each sample $x_i \in S$

- [3] if α_i changes from $0 < \alpha_i < C$ to $\alpha_i = C$ then sample x_i migrates from S to E. If $\alpha_i = 0$ sample x_i migrates from S to R

for each sample $x_i \in E$

- [4] if $h(x_i)$ changes from $h(x_i) < 1$ to $h(x_i) = 1$ then sample x_i migrates from E to S

for each sample $x_i \in R$

- [5] if $h(x_i)$ changes from $h(x_i) > 1$ to $h(x_i) = 1$ then sample x_i migrates from R to S.

The bookkeeping procedure then is to trace for each sample into the training set D' against these five cases and to determine the allowed $\Delta\alpha_c$ for each sample according to the section C.5 and section C.5 while the final $\Delta\alpha_c$ is defined as the one with the minimum absolute value among the possible $\Delta\alpha_c$. In order to identify the largest increase $\Delta\alpha_c$ such that some point migrates between the sets S and R we have to consider the following cases:

1. some $\alpha_i \in S$ reaches an upper or lower bound and considering a small number ε compute the sets $I_+^S = \{i \in S | \hat{\beta}_i > \varepsilon\}$ and $I_-^S = \{i \in S | \hat{\beta}_i < -\varepsilon\}$. Clearly examples in I_+^S have positive sensitivity and should be tested for reaching the upper bound C while the ones in I_-^S should be tested for reaching zero. Therefore examples in $-\varepsilon < \hat{\beta}_i < \varepsilon$ insensitive to $\Delta\alpha_c$ should be ignored. Hence possible updates are $\Delta\alpha_i^{max} = C - \alpha_i$ if $i \in I_+^S$ while $\Delta\alpha_i^{max} = -\alpha_i$ if $i \in I_-^S$ and the largest possible $\Delta\alpha_c^S = abs \min_{i \in I_+^S \cup I_-^S} \frac{\Delta\alpha_i^{max}}{\hat{\beta}_i}$ where $absmin_i(x) = \min_i |x_i| sign(x_{(arg\min_i |x_i|)})$

2. some g_i in R reaches zero then we have to compute the sets $I_+^R = \{i \in E | \hat{\gamma}_i > \varepsilon\}$ and $I_-^R = \{i \in R | \hat{\gamma}_i < -\varepsilon\}$. Examples in I_+^R have positive while those in I^R- negative sensitivity of the gradient with respect to weight of the current sample. Hence the largest possible increase of $\Delta\alpha_c^g$ before some point in R or in E moves to S can be evaluates according to $\Delta\alpha_c^R = \min_{i \in I_+^R \cup I_-^R} \frac{-g_i}{\hat{\gamma}_i}$

3. g_c becomes zero and this case is similar to the case 2 with feasibility test in the form of $\gamma_c > \varepsilon$. The largest possible update in this case looks like $\Delta\alpha_c^g = \frac{-g_c}{\hat{\gamma}_c}$

4. α_c reaches C and the largest possible increment is $\Delta\alpha_c^\alpha$

Finally the smallest of the four possible values will be

$$\Delta\alpha_c^{max} = \min(\Delta\alpha_c^S, \Delta\alpha_c^R, \Delta\alpha_c^g, \Delta\alpha_c^\alpha) \tag{C.46}$$

which constitutes the largest possible increment of α_c.

The migration process needs that the matrix $\mathbf{R}$ must be updated whenever the set S changes its composition. It is possible to do the task efficiently without explicitly computing the matrix inverse. When the sample $x_k \in S$ is removed the updated $\mathbf{R}^{\mathbf{new}}$ can be obtained as follows:

$$\begin{aligned} \mathbf{R}^{\mathbf{new}} &= \mathbf{R}_{\mathbf{I},\mathbf{I}} - \frac{\mathbf{R}_{\mathbf{I},\mathbf{k}}\mathbf{R}_{\mathbf{k},\mathbf{I}}}{R_{k,k}} \\ \mathbf{I} &= \{1, ..., k, k+2, ..., s_{l_s}+1\} \end{aligned} \tag{C.47}$$

while when a new sample is added to set S the update can be found through:

$$\mathbf{R}^{\mathbf{new}} = \begin{bmatrix} & \mathbf{R} & & 0 \\ & & & . \\ & & & . \\ 0 & . & . & 0 \end{bmatrix} + \frac{1}{\hat{\gamma}_i} \begin{bmatrix} \mu \\ 1 \end{bmatrix} \begin{bmatrix} \mu & 1 \end{bmatrix} \tag{C.48}$$

where μ and $\hat{\gamma}_i$ are defined as

$$\mu = -\mathbf{R} \begin{bmatrix} 1 \\ Q_{s_1 i} \\ . \\ Q_{s_{l_s} i} \end{bmatrix} \quad \hat{\gamma}_i = Q_{ii} + \begin{bmatrix} 1 \\ Q_{s_1 i} \\ . \\ Q_{s_{l_s} i} \end{bmatrix} \mu \tag{C.49}$$

when the sample x_i was moved from E to R. Whenever the sample x_c is added to S μ and $\hat{\gamma}_i$ can be obtained from the section C.5 and section C.5.

An initial SVM solution in general can be obtained from a batch SVM solver often being the most efficient approach. However, sometimes can be also convenient to build an incremental solution from scratch and in this case one need to provide a starting point and can be used the two sample solution. The sets E, S and R can be initialized from these two points based on Equations section C.5,section C.5,section C.5 and if S is non empty the matrix $\mathbf{R}$ can also be initialized from section C.5 while as long as the set S remains empty, the matrix $\mathbf{R}$ will not be used. Finally, to end the recursive definition of the $\mathbf{R}$

Algorithm 4 Algorithm incremental SVM

Require: $(\alpha, \mathbf{R}, S, E, R)$ if any
  read (x_c, y_c) set $\alpha_c = 0$ compute g_c
  while $g_c < 0$ *and* $\alpha_c < C$ **do**
    compute $\hat{\beta}$ and $\hat{\gamma}$ according to section C.7 and section C.7
    compute $\Delta\alpha_c^{max}$ according section C.5
    update $b, \alpha_c, \alpha_s, g_c, g_r, g_e$
    Let k be the index of sample yielding the minimum in section C.5
    if $k \in S$ **then**
      move k from S to R or E
    end if
    if $k \in E \cup R$ **then**
      move k from E or R to S
    end if
    if then
      {k=c do nothing and terminate}
    end if
    update $\mathbf{R}$ according to section C.7 or section C.7
  end while
  return

matrix it remains to define the base case. When adding the first margin support vector, the matrix should be initialized as follows:

$$\mathbf{R} = Q^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & Q_{cc} \end{bmatrix}^{-1} = \begin{bmatrix} -Q_{cc} & 1 \\ 1 & 0 \end{bmatrix} \tag{C.50}$$

If we want to unlearn an existing sample from the training set D we can also provide a decremental algorithm. If the sample to unlearn $x_c \in R$ clearly does not contribute to the SVM solution and its removal does not require adjustments. However, if on the other hand x_c has $\alpha_c \neq C$ one can gradually reduce α_c while ensuring all the other samples in the training set to satisfy the KKT conditions. As a result the decremental algorithm follows the same strategy of the incremental one taking into account:

- the direction of the change of α_c from

$$sign(\Delta\alpha_c) = sign(-y_c f(x_c)) = sign(-h(x_c)) \leq 0 \tag{C.51}$$

- there is no case [1] because the removed x_c does not need to satisfy the KKT conditions
- the condition of case [2] becomes: α_c changing from $\alpha_c > 0$ to $\alpha_c = 0$.

The computational complexity of Algorithm 4 constitutes a *minor iteration* of the overall training algorithm. In fact a *major iteration* corresponds to inclusion of a new example; therefore, all complexity estimates must be multiplied by a number of examples to be learned. This is, however, a worst case scenario, since no minor iteration is needed

for many points that do not become support vectors at the time of their inclusion. Asymptotically, the complexity of a minor iteration is quadratic in a number of examples learned so far $O(n_l^2)$ as re-computation of $\hat{\beta}$ and $\hat{\gamma}$ involves matrix-vector multiplications having quadratic complexity. Moreover the recursive update of an inverse matrix can be shown to be quadratic in the number of examples. These estimates have to be multiplied by a number of minor iterations needed to learn an example. The number of minor iterations depends on the structure of a problem, namely on how often examples migrate between the index sets until an optimal solution is found. This number cannot be controlled in the algorithm and may be potentially exponentially large as the structure of the problem is determined by the geometry of a set of feasible solutions in a feature space. On the basis of the pseudo-code of Algorithm 4 we may see the key issues of the implementation of the incremental SVM.

- computation of g_c requires partial computation of the kernel row for the current example and examples in the set S. Computation of a kernel row is expensive since a subset of input points has to be selected using the index sets S, E and R.
- computation of g_i is especially costly since a two-dimensional selection has to be performed to obtain the kernel matrix ($O(n-n_{sv})$ memory access operations), followed by a matrix-vector multiplication ($O(n-n_{sv})$ arithmetic operations). The influence of g_i scales with the size of set S and the number of data points.
- operations for sets S and R have inferior complexity. If a kernel row of the example k is not present (in case of x_k entering the S from R) then it has to be re-computed for the update of the inverse matrix.
- update the inverse matrix requires the $O(n_{sv})$ arithmetic operations) and the expansion requires memory operation which is expensive for a large inverse matrix.

In [49] a detailed complexity analysis shows that memory access operations dominate the runtime and particular attention should be taken in order to improve efficiency of the incremental SVM.

C.6 Support Vector Regression

This section provides a brief overview of SVR for more details see [81]. The objective of the SVR problem is to learn a function $f(x)$ of the form

$$f(x) = \sum_{l=1}^{r} \phi_l(x) w_l + b = < \Phi(x), w > + b \tag{C.52}$$

providing a good approximation to a given set of training data

$$D = \{(x_1, y_1), ..., (x_n, y_n)\}$$

where $x_i \in \mathbb{R}^m$ is the input data and $y_i \in \mathbb{R}$ is the observed output. The vector

$$\Phi(x) = (\phi_1(x), ..., \phi_r(x))^T$$

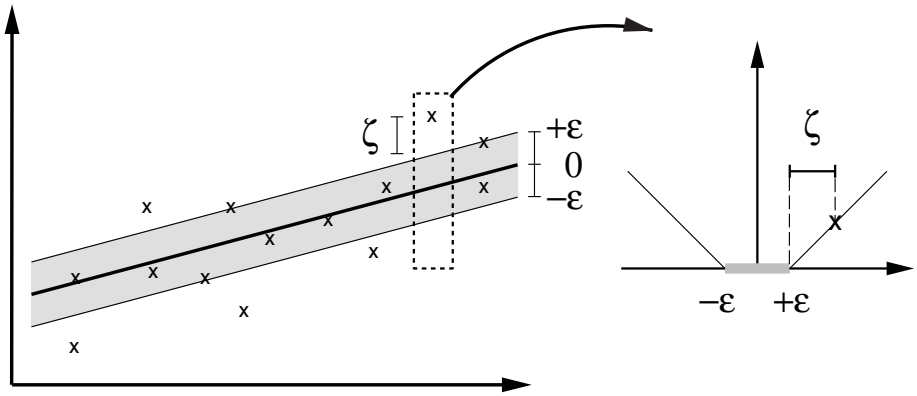

Figure C.4: The soft margin loss setting for a linear SVR.

is referred as features vector of the point x, where each features (also called a basis function) $\phi_i(x)$ is a scalar-valued function of x. The vector

$$\mathbf{w} = (w_1, ..., w_r)^T$$

is referred to as the weight vector. The notation $< \cdot, \cdot >$ is used to denote the standard inner product.

In SVR, following the maximum margin paradigm presented for the SVM, the training problem is posed as the following quadratic optimization problem

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\xi^*} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C\textstyle\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ s.t. \quad & y_i - \langle \Phi(x_i), \mathbf{w} \rangle - b \leq \varepsilon + \xi_i \\ & -y_i + \langle \Phi(x_i), \mathbf{w} \rangle + b \leq \varepsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \quad \forall i \in \{1, ..., n\} \end{aligned} \tag{C.53}$$

Here, the regularization term $\frac{1}{2}\|\mathbf{w}\|^2$ penalizes model complexity, and the ξ_i, ξ_i^* are slack variables which are active whenever a training point y_i lies farther than a distance ε from the approximating function $f(x_i)$, giving rise to the so-called $\varepsilon - insensitive$ loss function $|\xi|_\varepsilon$ described as

$$|\xi|_\varepsilon = \begin{cases} 0 \ if \ |\xi| \leq \varepsilon \\ |\xi| - \varepsilon \ \ otherwise. \end{cases} \tag{C.54}$$

The parameter C trades off model complexity with accuracy of fitting the observed training data. As C increases, any data points for which the slack variables are active incur higher cost, so the optimization problem tends to fit the data more closely (note that fitting too closely may not be desired if the training data is noisy).

The minimization problem section C.6 is difficult to solve when the number of features r is large, for two reasons. First, it is computationally demanding to compute the values of all r features for each of the data points. Second, the number of decision variables in the

problem is $r+2n+1$ (since there is one weight element w_i for each basis function $\phi_i()$ the coefficient b and two slack variables ξ_i, ξ_i^* for each training point), so the minimization must be carried out in an $(r+2n+1)$-dimensional space. To address these issues, one can solve the primal problem through its dual, which can be formulated by computing the Lagrangian and minimizing with respect to the primal variables $\mathbf{w}$ and ξ, ξ^*. Introducing the Lagrange multipliers α,α^*,η and η^* we can write the corresponding Lagrangian as:

$$\begin{aligned}\mathscr{L}_P = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i+\xi_i^*) - \sum_{i=1}^{n}(\eta_i\xi_i+\eta_i^*\xi_i^*) \\ -\sum_{i=1}^{n}\alpha_i[\varepsilon+\xi_i-y_i+\langle\Phi(x_i),\mathbf{w}\rangle+b] \\ -\sum_{i=1}^{n}\alpha_i^*[\varepsilon+\xi_i^*+y_i-\langle\Phi(x_i),\mathbf{w}\rangle-b]\end{aligned} \tag{C.55}$$

hence the dual variables have to satisfy positive constraints

$$\alpha_i, \alpha_i^*, \eta_i, \eta_i^* \geq 0 \tag{C.56}$$

It follows from the saddle point condition that the partial derivatives of $\mathscr{L}_P$ with respect to the primal variables $(\mathbf{w}, b, \xi, \xi^*)$ have to vanish for optimality:

$$\frac{\partial\,\mathscr{L}_P}{\partial\,\mathbf{w}} = 0 \quad \longrightarrow \quad \mathbf{w} = \sum_{i=1}^{n}(\alpha_i-\alpha_i^*)\Phi(x_i) \tag{C.57}$$

$$\frac{\partial\,\mathscr{L}_P}{\partial\,b} = 0 \quad \longrightarrow \quad \sum_{i=1}^{n}(\alpha_i-\alpha_i^*) = 0 \tag{C.58}$$

$$\frac{\partial\,\mathscr{L}_P}{\partial\,\xi_i} = 0 \quad \longrightarrow \quad C-\alpha_i-\eta_i = 0 \tag{C.59}$$

$$\frac{\partial\,\mathscr{L}_P}{\partial\,\xi_i^*} = 0 \quad \longrightarrow \quad C-\alpha_i^*-\eta_i^* = 0 \tag{C.60}$$

$$\tag{C.61}$$

Substituting section C.6,section C.6,section C.6,section C.6 into section C.6 and eliminating the dual variables $\eta_i = C-\alpha_i$ and $\eta_i^* = C-\alpha_i^*$ yields to the dual optimization problem

$$\begin{aligned}\min_{\alpha,\alpha^*} \quad & \tfrac{1}{2}\textstyle\sum_{i,j=1}^{n}k(x_i,x_j)(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*) \\ & +\varepsilon\textstyle\sum_{i=1}^{n}(\alpha_i+\alpha_i^*) - \sum_{i=1}^{n}y_i(\alpha_i-\alpha_i^*) + b\sum_{i=1}^{n}(\alpha_i-\alpha_i^*) \\ s.t. \quad & \textstyle\sum_{i=1}^{n}(\alpha_i-\alpha_i^*) = 0 \quad \alpha_i,\alpha_i^* \in [0,C] \quad \forall\, i \in \{1,\ldots,n\}\end{aligned} \tag{C.62}$$

where $k(x_i,x_j) = \langle\Phi(x_i),\Phi(x_j)\rangle = Q_{ij}$ is the kernel function. Moreover the Karush-Kuhn-Tucker (KKT) conditions require that at the point of the solution the product between the dual variables and constraints has to vanish:

$$\begin{aligned}\alpha_i[\varepsilon+\xi_i-y_i+\langle\Phi(x_i),\mathbf{w}\rangle+b] = 0 \\ \alpha_i^*[\varepsilon+\xi_i^*+y_i-\langle\Phi(x_i),\mathbf{w}\rangle-b] = 0 \\ (C-\alpha_i)\xi_i = 0 \\ (C-\alpha_i^*)\xi_i^* = 0\end{aligned} \tag{C.63}$$

From which one may come to some useful conclusions. First only samples (x_i, y_i) with corresponding $\alpha_i = C$ or $\alpha_i^* = C$ lie outside the $\varepsilon - insensitive$ tube. Secondly $\alpha_i \alpha_i^* = 0$ i.e. there can never be a set of dual variables α_i, α_i^* simultaneously non zero. This allow us to conclude that:

$$\begin{aligned} \varepsilon - y_i + \langle \Phi(x_i), \mathbf{w} \rangle + b \geq 0 \quad and \quad \xi_i = 0 \quad if \quad \alpha_i < C \\ \varepsilon - y_i + \langle \Phi(x_i), \mathbf{w} \rangle + b \leq 0 \quad if \quad \alpha_i = C \\ \varepsilon + y_i - \langle \Phi(x_i), \mathbf{w} \rangle + b \geq 0 \quad and \quad \xi_i^* = 0 \quad if \quad \alpha_i^* < C \\ \varepsilon + y_i - \langle \Phi(x_i), \mathbf{w} \rangle + b \leq 0 \quad if \quad \alpha_i^* = C \end{aligned} \tag{C.64}$$

from which one may conclude that

$$\begin{aligned} \max\{-\varepsilon + y_i - \langle \Phi(x_i), \mathbf{w} \rangle \mid \alpha_i < C \, or \, \alpha_i * > 0\} \leq b \leq \\ \min\{-\varepsilon + y_i - \langle \Phi(x_i), \mathbf{w} \rangle \mid \alpha_i > 0 \, or \, \alpha_i * < C\} \end{aligned} \tag{C.65}$$

and if some α_i or α_i^* the inequalities become equalities which allow to compute b.

Note that the features vectors $\Phi(x_i)$ now enter into the optimization problem only as inner products. This is important, because it allows the kernel function to be defined whose evaluation may avoid the need to explicitly calculate the vectors $\Phi(x_i)$, resulting in significant computational savings. Also, the dimensionality of the dual problem is reduced to only $2n$ decision variables, since there is one α_i and one α_i^* for each of the training points. When the number of features is large, this results in significant computational savings. Furthermore, it is well known that the dual problem can be solved efficiently using specialized techniques such as Sequential Minimal Optimization. Once the dual variables are known, the weight vector is given by (defining $\theta_i = \alpha_i - \alpha_i^*$)

$$\mathbf{w} = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \Phi(x_i) = \sum_{i=1}^{n} \theta_i \Phi(x_i) \tag{C.66}$$

and the function $f(x)$ can be computed using the support vector expansion

$$\begin{aligned} f(x) = \langle \Phi(x), \mathbf{w} \rangle + b = \sum_{i=1}^{n} \theta_i \langle \Phi(x), \Phi(x_i) \rangle + b \\ = \sum_{i=1}^{n} \theta_i k(x, x_i) + b \end{aligned} \tag{C.67}$$

The Lagrange formulation of section C.6

$$\begin{aligned} \mathscr{L}_D = \tfrac{1}{2} \textstyle\sum_{i,j=1}^{n} k(x_i, x_j)(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \\ + \varepsilon \textstyle\sum_{i=1}^{n} (\alpha_i + \alpha_i^*) - \sum_{i=1}^{n} y_i (\alpha_i - \alpha_i^*) + \hat{b} \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \\ - \textstyle\sum_{i=1}^{n} (\delta_i \alpha_i + \delta_i^* \alpha_i^*) + \sum_{i=1}^{n} [\upsilon_i (\alpha_i - C) + \upsilon_i^* (\alpha_i^* - C)] \end{aligned} \tag{C.68}$$

where $\delta_i, \delta_i^*, \upsilon_i, \upsilon_i^*$ and $\hat{b}$ are the Lagrange multiplier. Optimizing this Lagrangian leads

to the following KKT conditions:

$$\frac{\partial \mathscr{L}_D}{\partial \alpha_i} = 0 \quad \longrightarrow \quad \sum_{j=1}^{n} Q_{ij}\theta_j + \varepsilon - y_i + \hat{b} - \delta_i + \upsilon_i = 0 \tag{C.69}$$

$$\frac{\partial \mathscr{L}_D}{\partial \alpha_i^*} = 0 \quad \longrightarrow \quad -\sum_{j=1}^{n} Q_{ij}\theta_j + \varepsilon + y_i - \hat{b} - \delta_i^* + \upsilon_i^* = 0 \tag{C.70}$$

$$\delta_i \geq 0 \quad \delta_i \alpha_i = 0 \quad \delta_i^* \geq 0 \quad \delta_i^* \alpha_i^* = 0 \tag{C.71}$$

$$\upsilon_i \geq 0 \quad \upsilon_i(\alpha_i - C) = 0 \quad \upsilon_i^* \geq 0 \quad \upsilon_i^*(\alpha_i^* - C) = 0 \tag{C.72}$$

Note that b is equal to $\hat{b}$ at optimality and defining $g_i = \sum_{j=1}^{n} Q_{ij}\theta_j + \varepsilon - y_i + b$ results $g_i^* = -g_i + 2\varepsilon$. As the $\alpha_i\alpha_i^* = 0$ condition holds the $\theta_i = \alpha_i - \alpha_i^*$ parameter completely determine both α_i and α_i^*. Now is useful to define the margin function for the $i-th$ sample as

$$h(x_i) = f(x_i) - y_i = \sum_{j=1}^{n} Q_{ij}\theta_j + b - y_i = g_i - \varepsilon = -g_i^* + \varepsilon \tag{C.73}$$

It is also worth noting that the following conditions descend from the KKT:

$$\begin{aligned} \delta_i = h(x_i) + \varepsilon + \upsilon_i = g_i + \upsilon_i \quad \alpha_i[h(x_i) + \varepsilon + \upsilon_i] = 0 \\ \delta_i^* = -h(x_i) + \varepsilon + \upsilon_i^* = g_i^* + \upsilon_i^* \quad \alpha_i^*[-h(x_i) + \varepsilon + \upsilon_i^*] = 0 \end{aligned} \tag{C.74}$$

and combining $h(x_i)$ definition with the KKT conditions we can obtain:

$$\begin{cases} h(x_i) \geq \varepsilon \;\; \theta_i = -C,\, \alpha_i = 0,\, \alpha_i^* = C \\ h(x_i) = \varepsilon \;\; -C < \theta_i < 0,\, \alpha_i = 0,\, 0 < \alpha_i^* < C \\ -\varepsilon \leq h(x_i) \leq \varepsilon \;\; \theta_i = 0,\, \alpha_i = 0,\, \alpha_i^* = 0 \\ h(x_i) = -\varepsilon \;\; 0 < \theta_i < C,\, 0 < \alpha_i < C,\, \alpha_i^* = 0 \\ h(x_i) \leq -\varepsilon \;\; \theta_i = C,\, \alpha_i = C,\, \alpha_i^* = 0 \end{cases} \tag{C.75}$$

There are five conditions in section C.6 and they can be identified with three subsets into which the samples in the training set D can be classified:

- the set E Error support vectors $E = \{i \,|\, |\theta_i| = C\}$
- the set S Margin support vectors $S = \{i \,|\, 0 < |\theta_i| < C\}$
- the set R Remaining samples $R = \{i \,|\, |\theta_i| = 0\}$

C.7 Incremental SVR

The incremental algorithm for SVR ([70], [49] and [57]) updates of the trained SVR function whenever a new sample x_k is added to the training set D and the basic idea is to change the coefficient θ_c corresponding to the new sample x_c in a finite number of discrete steps until it meets the KKT conditions while ensuring that the existing samples in D continue to satisfy the KKT conditions at each step.

Moreover to build an exact on-line SVR one needs to define three primitive actions:

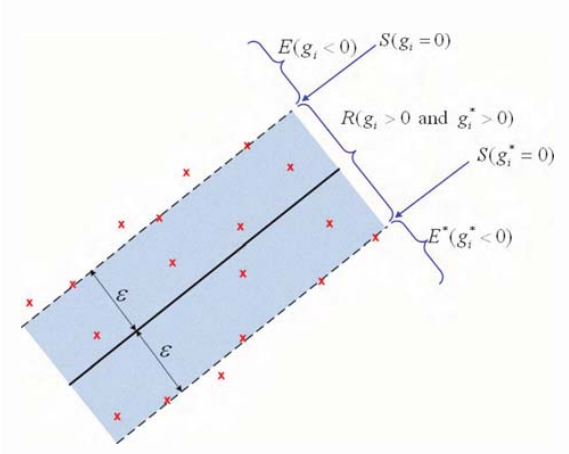

Figure C.5: Decomposition of D following the KKT conditions into Margin support vectors S, error support vectors E and E^*, and the remaining vectors R.

- add a new vector $D' = D \cup (x_c, y_c)$
- remove an existing vector $D' = D \setminus (x_c, y_c)$
- update an existing vector $D' = D \setminus (x_c, y_c) \cup (x_c, y_c')$

in each case the resulting incremental SVR should be the same that would be training from the scratch using the whole final set of data.

In order to derive the incremental relations, consider x_c with the initially set value of $\theta_c = 0$ and gradually change (increase o decrease) its value under KKT of section C.6 According to the KKT the incremental relation between $\Delta h(x_i)$, $\Delta\theta_i$ and Δb is given by:

$$\Delta h(x_i) = Q_{ic}\Delta\theta_c + \sum_{j=1}^{n} Q_{ij}\Delta\theta_j + \Delta b \tag{C.76}$$

while from the equality condition coming from b we have

$$\theta_c + \sum_{i=1}^{n} \theta_i = 0 \tag{C.77}$$

Combining section C.7, section C.7 with definition of E,S and R sets we obtain:

$$\sum_{j\in S}^{n} Q_{ij}\Delta\theta_j + \Delta b = -Q_{ic}\Delta\theta_c \quad \forall\, i \in S \tag{C.78}$$

$$\sum_{j\in S}^{n} \Delta\theta_j = -\Delta\theta_c$$

Now if we define the index of the samples in the S set as $S = \{s_1, s_2, ..., s_{l_s}\}$ the section C.7 can be represented in matrix form as

$$\begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q_{s_l s_1} & \dots & Q_{s_l s_{l_s}} \\ . & . & . & . \\ 1 & Q_{s_{l_s} s_1} & \dots & Q_{s_{l_s} s_{l_s}} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\theta_{s1} \\ . \\ \Delta\theta_{s_{l_s}} \end{bmatrix} = - \begin{bmatrix} 1 \\ Q_{s_1 c} \\ . \\ Q_{s_{l_s} c} \end{bmatrix} \Delta\theta_c \tag{C.79}$$

that is

$$\begin{bmatrix} \Delta b \\ \Delta\theta_{s_1} \\ . \\ \Delta\theta_{s_{l_s}} \end{bmatrix} = \hat{\beta}\Delta\theta_c \tag{C.80}$$

where

$$\hat{\beta} = \begin{bmatrix} \beta \\ \beta_{s_1} \\ . \\ \beta_{s_{l_s}} \end{bmatrix} = -\mathbf{R}\begin{bmatrix} 1 \\ Q_{s_1 c} \\ . \\ Q_{s_{l_s} c} \end{bmatrix} \tag{C.81}$$

where

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & ... & 1 \\ 1 & Q_{s_l s_1} & ... & Q_{s_l s_{l_s}} \\ . & . & . & . \\ 1 & Q_{s_{l_s} s_1} & ... & Q_{s_{l_s} s_{l_s}} \end{bmatrix}^{-1} \tag{C.82}$$

Define the non support vector set $N = E \cup R = \{n_1, ..., n_{l_s}\}$ and combining the above equations leads to:

$$\begin{bmatrix} \Delta h(x_{n_1}) \\ \Delta h(x_{n_2}) \\ . \\ \Delta h(x_{n_{l_s}}) \end{bmatrix} = \hat{\gamma}\Delta\theta_c \tag{C.83}$$

where we define $\hat{\gamma}$ as

$$\hat{\gamma} = \begin{bmatrix} Q_{n_1 c} \\ Q_{n_2 c} \\ . \\ Q_{s_{l_s} c} \end{bmatrix} + \begin{bmatrix} 0 & 1 & ... & 1 \\ 1 & Q_{n_1 s_1} & ... & Q_{n_l s_{l_s}} \\ . & . & . & . \\ 1 & Q_{n_{l_s} s_1} & ... & Q_{n_{l_s} s_{l_s}} \end{bmatrix} \hat{\beta} \tag{C.84}$$

Given $\Delta\theta_c$ one can update θ_i, $i \in S$ and b according to section C.7 and update $\Delta h(x_i)$, $i \in N$ according to section C.7. Moreover section C.6 suggests that θ_i, $i \in N$ and $h(x_i)$, $i \in S$ are constant if the set S stays unchanged. Hence it only necessary to solve the problem on how to find the appropriate value $\Delta\theta_c$.

All the above incremental equations are valid while the vectors do not migrate from set R, E or S to another one. This suggest a way to cope with the problem of finding the correct $\Delta\theta_c$ choosing it to be the largest value that either can maintain the set S unchanged or eventually leads to the termination of the incremental algorithm.

As a matter of fact the first step is to determine if the change $\Delta\theta_c$ should be positive or negative according to the section C.6

$$q_c = sign(\Delta\theta_c) = sign(y_c - f(x_c)) = sign(-h(x_c)) \tag{C.85}$$

Then eventually one has to find out the bound on $\Delta\theta_c$ imposed by each sample in the training set D'. Considering only the case $\Delta\theta_c > 0$ (the opposite case $\Delta\theta_c < 0$ is quite similar), for a new sample (x_c, y_c) we have two cases:

- [1] $h(x_c)$ changes from $h(x_c) < -\varepsilon$ to $h(x_c) = -\varepsilon$ then the new sample is added to the set S and the algorithm terminates then check $\Delta\theta_c^1 = (-h(x_c) - q_c\varepsilon)/\hat{\gamma}_c$
- [2] if θ_c increases from $\theta_c < 0$ up to $\theta_c = 0$ the new sample is added to the set E and the algorithm terminates then check $\Delta\theta_c^2 = (q_c C - \theta_c)$

for each sample $x_i \in S$

- [3] if θ_i changes from $0 < |\theta_i| < C$ to $|\theta_i| = C$ then sample x_i migrates from S to E. If $\theta_i = 0$ sample x_i migrates from S to R then check:
  - if $q_c\hat{\beta}_i > 0$ and $0 \geq \theta_i < C$ then $\Delta\theta_c^3 = (C - \theta_i)/\hat{\beta}_i$
  - if $q_c\hat{\beta}_i > 0$ and $-C \geq \theta_i < 0$ then $\Delta\theta_c^3 = -\theta_i/\hat{\beta}_i$
  - if $q_c\hat{\beta}_i < 0$ and $0 \geq \theta_i < C$ then $\Delta\theta_c^3 = -\theta_i/\hat{\beta}_i$
  - if $q_c\hat{\beta}_i < 0$ and $-C \geq \theta_i < 0$ then $\Delta\theta_c^3 = (-C - \theta_i)/\hat{\beta}_i$

for each sample $x_i \in E$

- [4] if $h(x_i)$ changes from $|h(x_i)| > \varepsilon$ to $|h(x_i)| = \varepsilon$ then sample x_i migrates from E to S then check $\Delta\theta_c^4 = (-h(x_i) - sign(q_c\hat{\beta}_i)\varepsilon)/\hat{\beta}_i$

for each sample $x_i \in R$

- [5] if $h(x_i)$ changes from $|h(x_i)| < \varepsilon$ to $|h(x_i)| = \varepsilon$ then sample x_i migrates from R to S then check $\Delta\theta_c^5 = (-h(x_i) - sign(q_c\hat{\beta}_i)\varepsilon)/\hat{\beta}_i$.

The bookkeeping procedure then is to trace for each sample into the training set D' against these five cases and to determine the allowed $\Delta\theta_c$ for each sample according to the section C.7 and section C.7 while the final $\Delta\theta_c$ is defined as the one with the minimum absolute value among the possible $\Delta\theta_c$ is defined as the one with the minimum absolute value among the possible $\Delta\theta_c$ according to

$$\Delta\theta_c^{max} = q_c \min(\Delta\theta_c^1, \Delta\theta_c^2, \Delta\theta_c^3, \Delta\theta_c^4, \Delta\theta_c^5) \tag{C.86}$$

which constitutes the largest possible increment of θ_c.

The migration process needs that the matrix $\mathbf{R}$ must be updated whenever the set S changes its composition. It is possible to do the task efficiently without explicitly computing the matrix inverse. When the sample $x_k \in S$ is removed the updated $\mathbf{R}^{\mathbf{new}}$ can be obtained as follows:

$$\begin{aligned} \mathbf{R}^{\mathbf{new}} &= \mathbf{R}_{\mathbf{I},\mathbf{I}} - \frac{\mathbf{R}_{\mathbf{I},\mathbf{k}}\mathbf{R}_{\mathbf{k},\mathbf{I}}}{R_{k,k}} \\ \mathbf{I} &= \{1, ..., k, k+2, ..., s_{l_s}+1\} \end{aligned} \tag{C.87}$$

while when a new sample is added to set S the update can be found through:

$$\mathbf{R}^{\mathbf{new}} = \begin{bmatrix} & & 0 \\ & \mathbf{R} & . \\ & & . \\ 0 & . \; . & 0 \end{bmatrix} + \frac{1}{\hat{\gamma}_i}\begin{bmatrix} \mu \\ 1 \end{bmatrix}\begin{bmatrix} \mu & 1 \end{bmatrix} \tag{C.88}$$

where μ and $\hat{\gamma}_i$ are defined as

$$\mu = -\mathbf{R}\begin{bmatrix} 1 \\ Q_{s_1 i} \\ . \\ Q_{s_{l_s} i} \end{bmatrix} \quad \hat{\gamma}_i = Q_{ii} + \begin{bmatrix} 1 \\ Q_{s_1 i} \\ . \\ Q_{s_{l_s} i} \end{bmatrix} \mu \tag{C.89}$$

when the sample x_i was moved from E to R. Whenever the sample x_c is added to S μ and $\hat{\gamma}_i$ can be obtained from the section C.7 and section C.7.

An initial SVR solution in general can be obtained from a batch SVR solver often being the most efficient approach. However, sometimes can be also convenient to build an incremental solution from scratch and in this case one need to provide a starting point and can be used the two sample solution. Considering as initial set $D_2 = \{(x_1, y_1), (x_2, y_2)\}$ with $y_1 \geq y_2$ a solution for the optimization problem section C.6 is

$$\theta_1 = \max(0, \min(C, \tfrac{y_1 - y_2 - 2\varepsilon}{2(K_{11} - K_{12})})) \tag{C.90}$$

$$\theta_2 = -\theta_1 \tag{C.91}$$

$$b = (y_1 - y_2)/2 \tag{C.92}$$

The sets E, S and R can be initialized from these two points based on section C.6,section C.6,section C.6 and if S is non empty the matrix $\mathbf{R}$ can also be initialized from section C.7 while as long as the set S remains empty, the matrix $\mathbf{R}$ will not be used. Finally, to end the recursive definition of the $\mathbf{R}$ matrix it remains to define the base case. When adding the first margin support vector, the matrix should be initialized as follows:

$$\mathbf{R} = Q^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & Q_{cc} \end{bmatrix}^{-1} = \begin{bmatrix} -Q_{cc} & 1 \\ 1 & 0 \end{bmatrix} \tag{C.93}$$

If we want to unlearn an existing sample from the training set D we can also provide a decremental algorithm. If the sample to unlearn $x_c \in R$ clearly does not contribute to the SVR solution and its removal does not require adjustments. However, if on the other hand x_c has $\theta_c \neq 0$ one can gradually reduce θ_c while ensuring all the other samples in the training set to satisfy the KKT conditions. As a result the decremental algorithm follows the same strategy of the incremental one taking into account:

- the direction of the change of θ_c from

$$sign(\Delta\theta_c) = sign(f(x_c) - y_c) = sign(h(x_c)) \tag{C.94}$$

- there is no case [1] because the removed x_c does not need to satisfy the KKT conditions
- the condition of case [2] becomes: θ_c changing from $|\theta_c| > 0$ to $|\theta_c| = 0$.

The computational complexity of Algorithm 5 follows the same history depicted for the Algorithm 4 which constitutes a *minor iteration* of the overall training algorithm. In fact a *major iteration* corresponds to inclusion of a new example; therefore, all complexity estimates must be multiplied by a number of examples to be learned. This is, however, a

Algorithm 5 Algorithm incremental SVR

Require: $(\theta, \mathbf{R}, S, E, R)$ if any
  read (x_c, y_c) set $\theta_c = 0$ compute g_c g_c^* and q_c
  while $(g_c \leq 0 \, or \, g_c^* \leq 0)$ **do**
    compute $\hat{\beta}$ and $\hat{\gamma}$ according to section C.7 and section C.7
    compute $\Delta\theta_c^{max}$ according section C.7
    update $b, \theta_c, \theta_s, g_c, g_r, g_e$
    Let k be the index of sample yielding the minimum in section C.5
    if $k \in S$ **then**
      move k from S to R or E
    end if
    if $k \in E \cup R$ **then**
      move k from E or R to S
    end if
    if then
      {k=c do nothing and terminate}
    end if
    update **R** according to section C.7 or section C.7
  end while
  return

worst case scenario, since no minor iteration is needed for many points that do not become support vectors at the time of their inclusion. Asymptotically, the complexity of a minor iteration is quadratic in a number of examples learned so far $O(n_t^2)$ as re-computation of $\hat{\beta}$ and $\hat{\gamma}$ involves matrix-vector multiplications having quadratic complexity. Moreover the recursive update of an inverse matrix can be shown to be quadratic in the number of examples. These estimates have to be multiplied by a number of minor iterations needed to learn an example. The number of minor iterations depends on the structure of a problem, namely on how often examples migrate between the index sets until an optimal solution is found. This number cannot be controlled in the algorithm and may be potentially exponentially large as the structure of the problem is determined by the geometry of a set of feasible solutions in a feature space.