Directed Exploration of Policy Space in Reinforcement Learning

by

Ioannis Rexakis

School of Electrical and Computer Engineering
Technical University of Crete, Greece

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the School of Electrical and Computer Engineering of the Technical University of Crete, Greece
2018

Directed Exploration of Policy Space in Reinforcement Learning

by

Ioannis Rexakis

Abstract

Reinforcement learning refers to a broad class of learning problems. Autonomous agents typically try to learn how to achieve their goal solely by interacting with their environment. They perform a trial-and-error search and they receive delayed rewards (or penalties). The challenge is to learn a good or even optimal decision policy, one that maximizes the total long-term reward. A decision policy for an autonomous agent is the knowledge of what to do in any possible state in order to achieve the long-term goal efficiently.

Several recent learning approaches within decision making under uncertainty suggest the use of classifiers for the compact (approximate) representation of policies. However, the space of possible policies, even under such structured representations, is huge and must be searched carefully to avoid computationally expensive policy simulations.

In this dissertation, our first contribution uncovers policy structure by deriving optimal policies for two standard two-dimensional reinforcement learning domains, namely the Inverted Pendulum and the Mountain Car. We found that optimal policies have significant structure and a high degree of locality, i.e. dominant actions persist over large continuous areas within the state space. This observation provides sufficient justification for the appropriateness of classifiers for approximate policy representation.

Our second and main contribution is the proposal of two Directed Policy Search algorithms for the efficient exploration of policy space provided by Support Vector Machines and Relevance Vector Machines. The first algorithm exploits the structure of the classifiers used for policy representation. The second algorithm uses an importance function to rank the states, based on action prevalence. In both approaches, the search over the state space is focused on areas where there is change of action domination. This directed focus on critical parts of the state space iteratively leads to refinement and improvement of the underlying policy and delivers excellent control policies in only a few iterations with a relatively small rollout budget, yielding significant computational time savings.

We demonstrate the proposed algorithms and compare them to prior work on three standard reinforcement learning domains: Inverted Pendulum (two-dimensional), Mountain Car (two-dimensional), Acrobot (four-dimensional). Additionally, we demonstrate the scalability of the proposed approaches on the problem of learning how to control a 4-Link, Under-Actuated, Planar Robot, which corresponds to an eight-dimensional problem, well-known in the control theory community. In all cases, the proposed approaches strike a balance between efficiency and effort, yielding sufficiently good policies without excessive steps of learning.

Κατευθυνόμενη Αναζήτηση του Χώρου Πολιτικών στην Ενισχυτική Μάθηση

Ιωάννης Ρεξάκης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πολυτεχνείο Κρήτης

Διατριβή που παρεδόθη για να καλύφει μερικώς τις απαιτήσεις απόκτησης του Διδακτορικού Διπλώματος στη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πολυτεχνείου Κρήτης

2018

Κατευθυνόμενη Αναζήτηση του Χώρου Πολιτικών στην Ενισχυτική Μάθηση

Ιωάννης Ρεξάκης

Περίληψη

Η ενισχυτική μάθηση αναφέρεται σε μια ευρεία κατηγορία προβλημάτων μάθησης. Οι αυτόνομες οντότητες τυπικά προσπαθούν να μάθουν να επιτυγχάνουν το στόχο τους αποκλειστικά μέσω της αλληλεπίδρασης με το περιβάλλον τους. Κάνουν διερευνητικές προσπάθειες αναζήτησης μέσω δοκιμών και ελέγχων και λαμβάνουν με καθυστέρηση ανταμοιβές (ή ποινές). Η πρόκληση είναι να μάθουν μια ικανοποιητική ή ακόμα και βέλτιστη πολιτική λήψης αποφάσεων, η οποία να μεγιστοποιεί τη συνολική μακροπρόθεσμη ανταμοιβή. Μια πολιτική λήψης αποφάσεων για μια αυτόνομη οντότητα είναι η γνώση του τι πρέπει να κάνει σε κάθε πιθανή κατάσταση προκειμένου να επιτευχθεί αποτελεσματικά ο μακροπρόθεσμος στόχος.

Πολλές πρόσφατες προσεγγίσεις μάθησης για τη λήψη αποφάσεων υπό αβεβαιότητα προτείνουν τη χρήση ταξινομητών για την συμπαγή (προσεγγιστική) αναπαράσταση πολιτικών. Ωστόσο, ο χώρος των πιθανών πολιτικών, ακόμα και κάτω από τέτοιες δομημένες αναπαραστάσεις, είναι τεράστιος και πρέπει να αναζητηθεί προσεκτικά για να αποφευχθούν υπολογιστικά ακριβές προσομοιώσεις πολιτικών.

Σε αυτή τη διατριβή, η πρώτη μας συμβολή σχετίζεται με την ανίχνευση δομής σε βέλτιστες πολιτικές. Εξετάσαμε βέλτιστες πολιτικές για δύο βασικά πεδία ενισχυτικής μάθησης δύο διαστάσεων, το Inverted Pendulum και το Mountain Car. Διαπιστώσαμε ότι οι βέλτιστες πολιτικές τους έχουν σημαντική δομή και υψηλό βαθμό τοπικότητας, δηλαδή οι κυρίαρχες ενέργειες παραμένουν ίδιες σε μεγάλες συνεχείς περιοχές εντός του χώρου καταστάσεων. Η παρατήρηση αυτή παρέχει επαρκή αιτιολόγηση για την καταλληλότητα των ταξινομητών για προσεγγιστική αναπαράσταση πολιτικών.

Η δεύτερη και κύρια συμβολή μας είναι η πρόταση δύο αλγορίθμων για την κατευθυνόμενη αναζήτηση του χώρου πολιτικών με τη χρήση των ταξινομητών SVM και RVM. Ο πρώτος αλγόριθμος εκμεταλλεύεται τη δομή των ταξινομητών που χρησιμοποιούνται για την αναπαράσταση της πολιτικής. Ο δεύτερος αλγόριθμος χρησιμοποιεί μια συνάρτηση σημαντικότητας των καταστάσεων, βάσει της επικράτησης των ενεργειών. Και στις δύο προσεγγίσεις, η αναζήτηση στον χώρο καταστάσεων επικεντρώνεται σε περιοχές όπου υπάρχει αλλαγή κυρίαρχης ενέργειας. Αυτή η κατευθυνόμενη εστίαση σε κρίσιμα τμήματα του χώρου καταστάσεων οδηγεί επαναληπτικά σε εκλέπτυνση και βελτίωση της τρέχουσας πολιτικής. Λίγες μόνο επαναλήψεις αρκούν για την παραγωγή εξαιρετικών πολιτικών με σχετικά χαμηλό αριθμό προσομοιώσεων, καταλήγοντας σε σημαντική εξοικονόμηση χρόνου.

Παρουσιάζουμε τους προτεινόμενους αλγόριθμους και τους συγκρίνουμε με τις προηγούμενες εργασίες σε τρία βασικά πεδία μελέτης της ενισχυτικής μάθησης: Inverted Pendulum (δύο διαστάσεων), Mountain Car (δύο διαστάσεων) και Acrobot (τεσσάρων διαστάσεων). Επιπροσθέτως, επιδεικνύουμε την επεκτασιμότητα των προτεινόμενων προσεγγίσεων στο πρόβλημα της μάθησης για τον έλεγχο ενός 4-Link Planar Robot, το οποίο αντιστοιχεί σε ένα πρόβλημα οκτώ διαστάσεων, γνωστό στην κοινότητα της θεωρίας ελέγχου. Σε όλες τις περιπτώσεις, οι προτεινόμενες προσεγγίσεις επιτυγχάνουν μια ισορροπία μεταξύ αποτελεσματικότητας και προσπάθειας, αποδίδοντας επαρκώς καλές πολιτικές σε σύντομο χρονικό διάστημα, χωρίς υπερβολικό αριθμό βημάτων μάθησης.

Contents

List of Figures

Symbols and Abbreviations

Symbols

$\mathbb{R}$	Set of real numbers
$\mathbb{N}$	Set of natural numbers
N	Dimensionality of input samples
$\mathcal{X}$	Input space $\mathbb{R}^N$
$\mathcal{Q}$	Space of all Q-functions
m	Number of input samples

Sequential decision-making context

$\mathcal{S}$	State space
$\mathcal{A}$	Action space
$\mathcal{D}$	Initial state distribution
l	The MDP time step, an integer variable, the first time step is zero ($l = 0$)
s_l	State of MDP at time step l
a_l	Action taken at time step l
$\mathcal{P}(s_l, a_l)$	Deterministic MDP transition function, returns next state s_{l+1} after taking action a_l at state s_l
$\tilde{\mathcal{P}}(s_l, a_l, s_{l+1})$	Stochastic MDP transition probability density function from state s_l taking action a_l, the probability that the next state s_{l+1} belongs to a region $\mathcal{S}_{l+1} \subseteq \mathcal{S}$ is: $$P(s_{l+1} \in \mathcal{S}_{l+1} \mid s_l, a_l) = \int_{\mathcal{S}_{l+1}} \tilde{\mathcal{P}}(s_l, a_l, s')ds'$$
$\bar{\mathcal{P}}(s_l, a_l, s_{l+1})$	Stochastic MDP transition function for countable or discrete state space, returns the probability to move to state s_{l+1} after taking action a_l at state s_l i.e., $P(s_{l+1} \mid s_l, a_l)$
$\tilde{\rho}(s_l, a_l, s_{l+1})$	Immediate reward function after taking action a_l at state s_l and moving to state s_{l+1}

r_l	$\tilde{\rho}(s_{l-1}, a_{l-1}, s_l)$ i.e., immediate reward value at time l after taking action a_{l-1} at state s_{l-1} and moving to state s_l
$\mathcal{R}(s,a)$	Expected reward function for countable space, after taking action a at state s, $$\mathcal{R}(s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s')\tilde{\rho}(s,a,s')$$ in case of deterministic MDP, $$\mathcal{R}(s,a) = \tilde{\rho}(s,a,\mathcal{P}(s,a))$$
γ	Discount factor $\in (0,1]$
π	Policy
π^*	Optimal policy
$\pi(s)$	Policy function, the action returned by policy π at state s. • A deterministic policy $\pi(s) : \mathcal{S} \mapsto \mathcal{A}$ returns the chosen action a at state s with probability 1. • A stochastic policy $\pi(s) : \mathcal{S} \mapsto \Omega(\mathcal{A})$ returns action a at state s with probability $\pi(a\|s)$ taken from the chosen probability distribution $\Omega(\mathcal{A})$ over $\mathcal{A}$.
$\pi(a\|s)$	Probability of policy π taking action a at state s
$V(s)$	State value function
$V_\pi(s)$	State value function for policy π
$V_{\pi^*}(s)$, $V^*(s)$	State value function for optimal policy π^*
$Q(s,a)$	State-action value function
$Q_\pi(s,a)$	State-action value function for policy π
$Q_{\pi^*}(s,a)$, $Q^*(s,a)$	State-action value function for optimal policy π^*
ϵ	probability of random action in ϵ-greedy policy

Supervised learning context

$\boldsymbol{x}$	Input vector $\boldsymbol{x} \in \mathcal{X}$
t	Target value $t \in \mathbb{R}$, for binary classification $t \in \{-1, +1\}$
$\{(\boldsymbol{x}_i, t_i)\}_{i=1}^N$	Input (training) data
α_i	Prediction function (alpha) coefficient for input vector $\boldsymbol{x}_i$, computed by the optimization process
$\mathcal{M}$	Set of indexes in the dataset $\{(\boldsymbol{x}_i, t_i)\}_{i=1}^N$ where the corresponding α_i is non-zero. The set of indexes for active vectors.
$y(\boldsymbol{x})$	Prediction function for input $\boldsymbol{x}$
$\kappa(\boldsymbol{x}', \boldsymbol{x})$	Kernel function, gives a measure of similarity for the input vectors $\boldsymbol{x}'$ and $\boldsymbol{x}$

Directed Policy Search

$\boldsymbol{s}$	State vector $\boldsymbol{s} \in \mathcal{X}$
$\Delta Q(\boldsymbol{s})$	Action advantage function in state $\boldsymbol{s}$
ϵ	Filtering limit
$\dot{\boldsymbol{s}}$	Active vector (with non-zero alpha coefficient)
G	Number of active input vectors
$\left\{(\dot{\boldsymbol{s}}_i, \dot{t}_i, \dot{\alpha}_i)\right\}_{i=1}^{G}$	Prediction function set of parameters: active vector $\dot{\boldsymbol{s}}_i$, target value $\dot{t}_i$, and non-zero $\dot{\alpha}_i$ coefficient
$\boldsymbol{u}_i$	Projection of active vector $\dot{\boldsymbol{s}}_i$ onto the separating border
$\boldsymbol{g_d}(\boldsymbol{s})$	Gradient vector of decision function $\boldsymbol{g_d}(\boldsymbol{s}) = \nabla_{\boldsymbol{s}} y(\boldsymbol{s})$
$\boldsymbol{g_q}(\boldsymbol{s})$	Gradient vector of action advantage function approximation $\boldsymbol{g_q}(\boldsymbol{s}) = \nabla_{\boldsymbol{s}} q(\boldsymbol{s})$
d_i	Distance of active vector $\dot{\boldsymbol{s}}_i$ from separating border

---

K	Number of repetitions (trials) of a rollout of policy π for state $\boldsymbol{s}$ and action a in order to estimate $Q_\pi(\boldsymbol{s}, a)$ (average)
H	Number of time steps (horizon) for a rollout
L	Number of attempts to get an improved policy π_{k+1} over π_k
U	Initial uniformly-sampled training set size
M	Directed sampled training set size ($M < U$)
k	Policy iteration number

Abbreviations

RL	Reinforcement Learning
API	Approximate Policy Iteration
MDP	Markov Decision Process
RBF	Radial Basis Function
SVM	Support Vector Machines
RVM	Relevance Vector Machines
LSPI	Least-Squares Policy iteration
RCPI	Rollout Policy Iteration
DRCPI	Directed Rollout Policy Iteration
DRCPI-AIV	Directed Rollout Policy Iteration using Active Input Vectors
DRCPI-IS	Directed Rollout Policy Iteration using Importance Sampling

Acknowledgements

Now that this long journey reaches its end, right here I would like to fulfill my need to expose my thoughts and feelings. Feelings come in many colors and flavors, but the strongest one is the gratefulness for all those who stood by me in this course of time. I would like to express my appreciation to all of them, even though I can not list everyone.

Let me start with my advisor Michail G. Lagoudakis, who led me through this process and thank him for the patient guidance, support, and advice he has provided all these years. It was a fun and fruitful trip for me. I still remember the first day I walked into his office as a bright day.

I would like to thank my friends, Athanasios Strilakos and Stelios Alexakis, for their support and encouragement that kept me motivated through hard times.

I acknowledge and thank my wife Charikleia and my daughter Sofia, for keeping a quiet and happy family environment.

Finally, I also need to thank my father Costas and my mother Sofia for their love and support throughout my life.

1 Introduction

Reinforcement Learning

Learning by doing or learning by trial-and-error is a significant way of learning in living creatures. Learners interact with their environment and use their experience to either choose or avoid certain actions based on rewards or penalties they receive. Reinforcement Learning (Sutton and Barto, 1998) describes a large class of such learning problems and typically refers to autonomous agents learning by interacting with their environment. These problems are sequential decision-making problems with delayed rewards. The agent, the decision maker, may take a long sequence of actions receiving little or no information about the quality of its decisions, and finally, may arrive at a terminal state with success or failure. Such problems are typically modeled as Markov Decision Processes (MDPs) (Puterman, 1994). The goal is to find a good decision policy, one that maximizes the cumulative reward received over time. The policy is the agent's knowledge of what to do in any particular state, to achieve the goal efficiently.

Deterministic policy

A deterministic policy is a mapping from states to actions. Good deterministic policies can be approximately represented using classifiers over the entire state space; each action is viewed as a distinct class and the states are the instances to be classified. Moreover, such policies for common domains are not arbitrary, but rather exhibit significant structure. Several learning approaches based on the approximate policy iteration framework suggest the use of classifiers for capturing this structure and representing policies compactly (Lagoudakis and Parr, 2003a; Fern et al., 2004; Langford and Zadrozny, 2005). Such classifiers can be learned using appropriate training data sets that reveal the desired action choice over a finite set of states. The most attractive benefit of this approach is the elimination of the need for value function representation; instead, the focus is put directly on policy learning. While it is known (Anderson, 2000) that it is easier to represent a policy, rather than a value function, the full potential of the reinforcement learning through classification approach, for various learning problems, has barely been explored.

Elimination of the Value Function

The space of possible policies, even under such structured representations, is huge and needs to be explored carefully to avoid computationally expensive simulations, i.e., rollouts. It is therefore desirable to have guidance for the selection of the subset of state space where the improved policy is probed, to form the training set for the classification problem. This aspect has been given little attention in the past, nevertheless it plays a crucial role, considering that each probe requires a significant amount of computational resources in simulation, and therefore they better be focused on critical states which can potentially lead to policy improvement.

1.1 Contribution Summary

Our Work

Our purpose is to efficiently explore the policy space and produce near optimal policies. Initially, we derived optimal policies for two standard two-dimensional reinforcement learning domains which are appropriate for visualization and inspection (Rexakis and Lagoudakis, 2008). Our goal was to uncover the structure that exists in optimal policies and get an idea of the kind of information we are looking for. An optimal deterministic policy is a map from states to actions and optimal actions tend to gather in large coherent areas within the state space. We direct the exploration of policy space using policy rollouts and uncover the areas where an action prevails over the others. We use a collection of binary classifiers to separate action areas within the state space and represent a policy; there is no representation for any kind of value function. At the core of this thesis, we developed and proposed two approaches for directed exploration of policy space. The first one exploits the structure of the classifiers used for policy representation. The second one uses a state importance function based on action prevalence. In both approaches, the search is focused on areas where there are changes of action domination. This directed focus on critical parts of the state space iteratively leads to refinement and improvement of the underlying policy and delivers excellent control policies in only a few iterations.

Our work, apart from contributing to the field of reinforcement learning, also has a potential for significant impact on real-world applications. In robotics, several control problems, such as balancing, walking, and recovering from disturbances, are commonly viewed as learning problems, as robots with many degrees of freedom (number of joints) become more and more common. Nowadays, a typical humanoid robot has at least 20 degrees of freedom, which implies a multi-dimensional state space that includes at least the angle and the angular velocity of each joint. Exploring such spaces efficiently is a crucial factor in achieving acceptable learning times. Although current robot technology does not allow extensive trial-and-error experimentation on the physical robots, in the future, algorithms, such as the ones proposed in this thesis, will be significant in making robot learning realizable in the real world in real time. Another field which could benefit from our work in the future is satellite technology, where the state space may be characterized by a large number of angles of the various satellite parts and components, whereas the degrees of control may be few (underactuated systems). Yet another field of application could be the development of competitive agents with learning capabilities in realistic, modern computer games, which are characterized by complex, multi-agent, multi-dimensional environments. In general, our work can be applicable to any multi-dimensional real-world domain, on the condition that a sampling procedure over the state space can generate valid states to which the underlying system can be initialized for the purpose of probing action values. Additionally, the impact of our work could become broader, if combined with the expressiveness and efficiency of deep learning classification technology.

1.2 Outline

Chapter 2 provides the basic definitions and introduces the notations necessary to follow the work presented in this thesis. It reviews several Reinforcement Learning topics, such as Markov Decision Processes, Policy Learning and Planning Algorithms, as well as Supervised learning topics, such as Classification, Regression, Loss Functions, and Regularization.

Chapter 3 describes the problem studied in this thesis and sets forth the goal of our research work. It begins with a review of the Rollout Classification Policy Iteration algorithm as the starting point of our work and lists a number of unanswered research questions.

Chapter 4 surveys previous work in the area of reinforcement learning combined with supervised learning (classification and regression), reviewing the most representative publications.

Chapter 5 details the Inverted Pendulum, Mountain Car, Acrobot and 4-Link Planar Robot domains, which are widely known benchmark domain and we used the test and evaluate our algorithms.

Chapter 6 is the core of our contribution; it describes our analysis of policy representations, our methods for uncovering and exploiting structure and finally our two Directed Policy Search approaches, which can be instantiated either using Support Vector Machines or Relevance Vector Machines.

Chapter 7 contains our experimental work. It includes first a computational approximation and visualization of optimal policies for the two-dimensional domains, Inverted Pendulum and Mountain Car. Then, it proceeds with an exhaustive experimental study of our approaches/algorithms for efficient exploration of the structure of policy space. Demonstration of the new algorithms is provided on a variety of control problems: Inverted Pendulum, Mountain Car, Acrobot and 4-Link Planar Robot.

Finally, Chapter 8 presents the summary and conclusion of this thesis, as well as some ideas for future research directions of our work.

2

Background

Machine learning is the field of artificial intelligence, probability, control, and optimization theory that deals with the extraction of a model from sample data and the use of that model to make a prediction or strategy. The gist of machine learning is the art of representation and generalization. How to extract a model that is strict enough to represent training data samples, and at the same time generalize well on unseen data samples.

Learning algorithms differ in the types of training data available, the order and method by which training data is received and the test data used to evaluate them. They are classified by the desired outcome of the algorithm or the type of input available during training.

Supervised learner uses labeled examples as training data and makes predictions for all unseen ones. This is (or at least it used to be) the most common case in machine learning, and it is associated with the classification, regression, and ranking problems. The email spam detection is an example of supervised learning.

Unsupervised learner receives unlabeled training data, discovers structure and patterns in the data, and delivers more compact approximate representations of the data. Since in general no labeled examples are available in this setting, it can be difficult to quantitatively evaluate the performance of a learner. Clustering is an example of unsupervised learning.

Semi-supervised learner gets labeled and unlabeled training samples and makes predictions for new ones. Semi-supervised learning is common in cases where unlabeled data are readily available, but labels are expensive to get. Various types of problems found in applications, including classification, regression, or ranking tasks, can be framed as instances of semi-supervised learning. The hope is that the distribution of unlabeled data accessible to the learner can help him achieve a better performance than in the supervised setting. The analysis of the conditions under which this can indeed be realized is the topic of much modern theoretical and applied machine learning research.

On-line learner involves multiple rounds of intermixed training and testing phases. The essential characteristic of online learning is that soon after the prediction is made, the true label of the instance is discovered. This information can then be used to refine the prediction hypothesis used by the algorithm. The goal of the algorithm is to make predictions that are close to the true labels.

Reinforcement learner learns by trial and error in a setting where training and testing phases are intermixed. To collect information, the learner actively interacts with the environment and receives an immediate reward for each action. The objective of the learner is to maximize the total reward over the course of interactions with the environment, a long-term reward. However, the learner faces the exploration versus exploitation dilemma; explore the unknown or exploit the already collected information?

2.1 Sequential Decision Making Model

The sequential decision-making methodology is applied in multi-stage planning or learning problems. Such a problem or *domain* consists of one or more decision makers, the agents, and their environment with which they interact. Sequential decision making has been studied in many diverse fields, including AI planning, decision analysis, operations research, control theory, and economics.

A basic sequential decision-making model has only two subsystems. One is the decision maker or *agent*, and the other one is the *environment*. A *state* of the system is the description of everything that may influence the decisions of the agent or be changed by its actions. At every time step, the agent is responsible for making a decision, and also take an *action* in response to that decision, to change the state of the environment (Figure 2.1). State changes are also affected stochastically by uncertainty in the environment. A rational agent has a *goal* to achieve. *Reward* is a numerical value given to the agent in response to an action. *Total reward* is a way of accumulating immediate rewards given to the agent and it indicates the goodness or badness of the agent's situation towards the goal. Rewards, either immediate or total, are delayed information about the agent's course of actions.

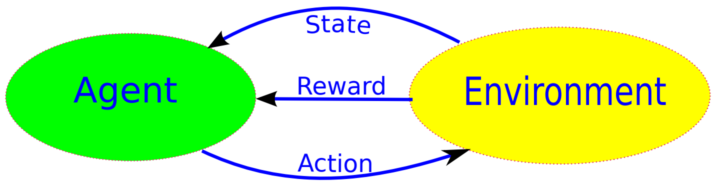


FIGURE 2.1: The agent interacts with the environment. At any state, the agent takes an action that changes the current state and receives a reward

The model is typically formulated using the Markov Decision Process framework. It consists of a set of states, a set of actions, rules of transitioning between states, rules that determine the immediate reward of a transition, and rules that describe what the agent observes. A policy is the agent's knowledge of what do at any specific state, in order to achieve its goal efficiently. Algorithms to find a good or optimal policy comes in two flavors depending of whether the model is known. Model-based algorithms are planning algorithms and they are usually implemented as *dynamic programming*[1] algorithms. Model-free algorithms are the *reinforcement learning algorithms*, where either the model is unknown or there is no analytic solution. A simulator may be available or information is collected by the agent interacting with the environment.

2.1.1 Markov Decision Process

Markov Decision Process or MDP (Bellman, 1957a; Puterman, 1994) is named after the Russian mathematician Andrey Markov and provides a mathematical framework for decision making in optimization problems where the outcome is random or unpredictable. An MDP is a discrete time stochastic control process. Discrete time is denoted by l.

State space

A state of the MDP is a situation where the agent has a decision to take. The different situations in which decisions must be made forms the entire state space $\mathcal{S}$ of the process. State space can be a finite or discrete set, $\mathcal{S} = \{s_1, s_2, \ldots, s_{|\mathcal{S}|}\}$, but infinite or continuous state spaces are also possible. A state $s \in \mathcal{S}$ is a complete description of the status of the process at a given time, and it is commonly given as a vector of real numbers. The size of that vector is the dimensionality of the state space and is related to the domain's complexity.

Action space

The agent's possible action choices at each state form the action space $\mathcal{A}$. We assume that action space $\mathcal{A}$ is a finite set, $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{A}|}\}$. It is possible that all actions are not available in each state, but for the sake of simplicity, it is assumed that all actions in $\mathcal{A}$ are available in all states.

[1] *programming* refers to planning, not to computer programming.

Transfer function

At time l, while in state s_l, the agent takes an action a_l and moves to the next state s_{l+1}. The next state s_{l+1} is given by the transition function. The MDP may have deterministic or stochastic transitions. The deterministic case is much simpler, the transition function is a map $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, and returns the next state s_{l+1} after taking action a_l at state s_l:

$$s_{l+1} = \mathcal{P}(s_l, a_l) \tag{2.1}$$

In a stochastic MDP, the transition probability density function $\tilde{\mathcal{P}}(s_l, a_l, s_{l+1})$ is a map $\tilde{\mathcal{P}} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$. The probability the next state s_{l+1} belongs to a region $\mathcal{S}_{l+1} \subseteq \mathcal{S}$ is:

$$P(s_{l+1} \in \mathcal{S}_{l+1} | s_l, a_l) = \int_{\mathcal{S}_{l+1}} \tilde{\mathcal{P}}(s_l, a_l, s')ds'$$

In case of a countable state space (e.g., discrete) the transition function is a map $\bar{\mathcal{P}} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and the probability to move to state s_{l+1} is:

$$P(s_{l+1} | s_l, a_l) = \bar{\mathcal{P}}(s_l, a_l, s_{l+1})$$

Rewards

The agent moves from state s_l to state s_{l+1} after taking action a_l and gets an immediate reward value r_{l+1} as an indication of how good the move was. The immediate reward in the stochastic MDP setting is given by a map $\tilde{\rho} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and is computed by:

$$r_{l+1} = \tilde{\rho}(s_l, a_l, s_{l+1})$$

All values of $\tilde{\rho}$ must be finite. The immediate reward function in the deterministic MDP setting is reduced to $\rho(s_l, a_l) = \tilde{\rho}(s_l, a_l, \mathcal{P}(s_l, a_l))$, since s_{l+1} is given by the deterministic transfer function $\mathcal{P}(s_l, a_l)$, and it is a map $\rho : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r_{l+1} = \rho(s_l, a_l)$$

Markov property

An MDP has the Markov property or memoryless property. Given the transition function $\mathcal{P}$ and the reward function ρ, the current state s_l and the current action a_l are enough to calculate the next state s_{l+1} and the reward r_{l+1} in the deterministic case, and the probability of the next state s_{l+1} and the reward r_{l+1} in the stochastic case. Therefore, there is no dependency on past states and actions; the current state and action are sufficient to predict the next step(s).

Discount factor

The agent will perform a series of actions, forming a trajectory in the state space, aiming to achieve his goal. However, in general, a shorter trajectory (less steps) is preferable to a longer one. We compute the total discounted reward as R

$$R = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \gamma^{H-1} r_H = \sum_{l=0}^{H-1} \gamma^l r_{l+1}$$

to evaluate a trajectory of H steps, where $\gamma \in (0, 1]$ is the discount factor, which discounts the reward values exponentially over time, since in general rewards received in the initial steps are more important than later ones. Infinite horizon ($H \to \infty$) is possible with a discount factor $\gamma \in (0, 1)$ (Sutton and Barto, 1998).

$$R = \sum_{l=0}^{\infty} \gamma^l r_{l+1}$$

Discount factor γ should be large enough, so that late rewards received upon reaching a terminal state are still detectable.

Initial state distribution

At time $l = 0$, the system is initialized, and the starting state is drawn from the initial state distribution $\mathcal{D}$ over $\mathcal{S}$. In common problems, the starting state is deterministically chosen, and therefore $\mathcal{D}$ assigns probability 1 to the departure state.

Episode

Some MDPs have terminal states. The agent's interaction with the environment breaks naturally into subsequences, called episodes or trials. Each episode is a trajectory that starts at some starting state and ends upon reaching a terminal or absorbing state. Then, a new episode starts.

The set of rules the agent uses to choose an action at each state forms a policy. The main objective is to find a policy π that serves best the given purpose. A stationary policy is a policy that does not change with time. A stationary deterministic policy π is a simple mapping from states to actions $\pi : \mathcal{S} \mapsto \mathcal{A}$; it is a function that returns the action a_l to take in state s_l.

Policy

$$a_l = \pi(s_l)$$

A stationary stochastic policy π is a mapping $\pi : \mathcal{S} \mapsto \Omega(\mathcal{A})$, where $\Omega(\mathcal{A})$ is the set of all probability distributions over $\mathcal{A}$. The function $\pi(a_l|s_l)$ is the conditional probability of action a_l to be taken at state s_l at time step l. Given a finite set of actions $\mathcal{A}$, the conditional probability π in any state $s \in \mathcal{S}$ has the following property: $\sum\limits_{a \in \mathcal{A}} \pi(a|s) = 1$.

The goal of the agent is to maximize the return, the expected cumulative reward over the course of agent's interactions with the environment. The expected total discounted reward for infinite horizon, starting from state s drawn from $\mathcal{D}$, following policy π (i.e., action a_l is drawn from $\pi(\cdot|s_l)$), making transitions according to the transition function $\tilde{\mathcal{P}}$ (i.e., s_{l+1} is drawn from $\tilde{\mathcal{P}}(s_l, a_l, \cdot)$), and receiving rewards r_l at each step, is formulated as:

$$E_{s\sim\mathcal{D};\, a_l\sim\pi;\, s_l\sim\tilde{\mathcal{P}};\, r_l\sim\tilde{\rho}} \left(\sum_{l=0}^{\infty} \gamma^l r_{l+1} \middle| s_0 = s \right)$$

This value measures the efficiency of a policy π. An optimal policy π^* is a policy that maximizes the total discounted reward for any possible starting state:

$$\pi^* = \arg\max_{\pi} \left\{ E_{s_0\sim\mathcal{D};\, a_l\sim\pi;\, s_l\sim\tilde{\mathcal{P}};\, r_l\sim\tilde{\rho}} \left(\sum_{l=0}^{\infty} \gamma^l r_{l+1} \middle| s_0 = s \right) \right\}$$

Infinite-horizon discounted-return MDPs, under certain conditions, have at least one stationary deterministic optimal policy (Bertsekas and Shreve, 1978, 1979). In this thesis, we mainly focus on stationary deterministic optimal and near-optimal policies.

Stationary deterministic policies

2.1.2 *Value Functions, Bellman Equation and optimality*

State value function

A *state value function* $V_\pi(s)$ is an estimate of "how good" is for an agent to be in a certain state s when following policy π. This value is a real number, and it is expressed as the expected discounted sum of rewards returned when the agent starts interacting with the environment at state s, and follows policy π thereafter:

$$V_\pi(s) = \left(\sum_{l=0}^{\infty} \gamma^l r_{l+1} \middle| s_0 = s \right), \text{ for the deterministic setting} \tag{2.2}$$

$$V_\pi(s) = E_{a_l \sim \pi;\ s_l \sim \tilde{\mathcal{P}};\ r_l \sim \tilde{\rho}} \left(\sum_{l=0}^{\infty} \gamma^l r_{l+1} \middle| s_0 = s \right), \text{ for the stochastic setting} \tag{2.3}$$

where $0 < \gamma < 1$ is the discount factor to keep the return finite. As a consequence, early accomplishments are preferable to later ones. With γ values near zero, the agent becomes myopic, i.e., time is critical to achieving the goal, with the risk of never reaching it. As γ approaches one, future rewards are nearly as significant as the early ones, and the time to achieve the goal becomes invariant. We use the term *V-function* in place of *state value function*, and the term *V-value* in place of *value of state value function.*

State action value function

A *state action value function* $Q_\pi(s, a)$ is a convenient measure of "how good" for an agent, is to take action a while being in state s and following policy π in future steps. That value is the expected discounted sum of rewards returned when the agent starts at state s, takes action a, and then follows policy π thereafter:

$$Q_\pi(s, a) = \left(r_1 + \sum_{l=1}^{\infty} \gamma^t r_{l+1} \middle| s_0 = s,\ a_0 = a \right), \text{ deterministic} \tag{2.4}$$

$$Q_\pi(s, a) = E_{a_l \sim \pi;\ s_l \sim \tilde{\mathcal{P}};\ r_l \sim \tilde{\rho}} \left(r_1 + \sum_{l=1}^{\infty} \gamma^t r_{l+1} \middle| s_0 = s,\ a_0 = a \right), \text{ stochastic} \tag{2.5}$$

We use the term *Q-function* in place of *state action value function*, and the term *Q-value* in place of *value of state action value function.*

The relation between V-function and Q-function is:

$$V_\pi(s) = Q_\pi(s, \pi(s)), \quad \text{for deterministic policy} \tag{2.6}$$

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s, a), \quad \text{for stochastic policy} \tag{2.7}$$

Deterministic vs. Stochastic setting

In the analysis below we use the term *deterministic setting* for an MDP with a deterministic transition function, following a deterministic policy, since the analysis is much simpler and instructive. Otherwise, we use the term *stochastic setting*.

2.1.2.1 Deterministic setting

Bellman equations

Bellman equations are recursive forms of the above value functions' equations. The Q-function (2.4) for policy π is a map $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. It returns the discounted cumulative reward when starting in state s, taking action a, and following policy π:

$$Q_\pi(s, a) = \rho(s, a) + \gamma V_\pi(\mathcal{P}(s, a)) \tag{2.8}$$

where the initial state and action is the pair (s, a), the next state is given by the transfer function $\mathcal{P}(s, a)$ (2.1), the value of the immediate reward function $\rho(s, a)$ is the r_1 reward in Equation (2.4), and $V_\pi(\mathcal{P}(s, a))$ (2.2) is the V-value of the next state following the current policy π. It is easy to derive the Bellman equation (2.8) from equation (2.4)

$$\begin{aligned} Q_\pi(s, a) &= \rho(s, a) + \sum_{l=1}^{\infty} \gamma^t r_{l+1} \\ &= \rho(s, a) + \gamma \sum_{l=1}^{\infty} \gamma^{l-1} r_{l+1} \\ &= \rho(s, a) + \gamma V_\pi(\mathcal{P}(s, a)) \end{aligned} \tag{2.9}$$

A recursive version of the above Q_π function is:

$$Q_\pi(s, a) = \rho(s, a) + \gamma Q_\pi(\mathcal{P}(s, a), \pi(\mathcal{P}(s, a))) \tag{2.10}$$

Greedy policy

A greedy policy is a deterministic policy that selects the action with the largest Q-value at every step:

$$\pi(s) = \arg\max_{a \in A} Q_{\pi}(s, a)$$

It is an obvious selection and leads to exploiting existing knowledge without allowing new actions to be explored. Therefore, regarding exploration, a greedy policy is myopic.

Optimal Q-function

The optimal state action value function $Q_{\pi*}$ or Q^* yields the largest Q-value achieved by any policy:

$$Q^*(s, a) = \max_{\pi} Q_{\pi}(s, a) \tag{2.11}$$

The recursive Bellman equation (2.10) can be written for the optimal Q-function:

$$Q^*(s, a) = \rho(s, a) + \gamma \max_{a' \in \mathcal{A}} Q^*(\mathcal{P}(s, a), a')) \tag{2.12}$$

Optimal policy

An optimal policy π^* selects actions that maximize the Q^*-value at any state:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \tag{2.13}$$

Optimal V-function

The optimal V^*-function yields the maximum of all V-values that can be obtained by any policy:

$$V^*(s) = \max_{\pi} V_{\pi}(s) = \max_{a \in \mathcal{A}} Q^*(s, a) \tag{2.14}$$

An optimal policy π^* is easily derived from the Bellman equation (2.9) and the definition of the optimal V^* value function (2.14):

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \Big\{ \rho(s, a) + \gamma V^*(\mathcal{P}(s, a)) \Big\} \tag{2.15}$$

The Bellman equation for the V-function is derived from equation (2.9):

$$V_{\pi}(s) = \rho(s, \pi(s)) + \gamma V_{\pi}(\mathcal{P}(s, \pi(s))) \tag{2.16}$$

and the Bellman equation for the optimal V-function is derived from (2.12):

$$V^*(s) = \max_{a \in \mathcal{A}} \Big\{ \rho(s, a) + \gamma V^*(\mathcal{P}(s, a)) \Big\} \tag{2.17}$$

2.1.2.2 Stochastic setting

Bellman equations

In a stochastic setting, transitions are stochastic and the next state s_{l+1} is drawn from $\tilde{\mathcal{P}}(s_l, a_l, \cdot)$. Q-function is the expected discounted return:

$$Q_\pi(s,a) = E_{s' \sim \tilde{\mathcal{P}}(s,a,\cdot)} \Big\{ \tilde{\rho}(s,a,s') + \gamma V_\pi(s') \Big\} \tag{2.18}$$

Optimal Q-function, and optimal policy

The definitions of the optimal state action value function Q^* and an optimal policy π^* are repeated here for convenience:

$$Q^*(s,a) = \max_\pi Q_\pi(s,a), \quad \pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a)$$

The Bellman equation for the Q-function is:

$$Q_\pi(s,a) = E_{s' \sim \tilde{\mathcal{P}}(s,a,\cdot)} \Big\{ \tilde{\rho}(s,a,s') + \gamma Q_\pi(s', \pi(s')) \Big\} \tag{2.19}$$

and the Bellman equation for the optimal Q^* value function is:

$$Q^*(s,a) = E_{s' \sim \tilde{\mathcal{P}}(s,a,\cdot)} \Big\{ \tilde{\rho}(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a') \Big\} \tag{2.20}$$

Countable state space

In the case of a countable (e.g., discrete) state space, the Bellman equation becomes:

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \Big\{ \tilde{\rho}(s,a,s') + \gamma Q_\pi(s', \pi(s')) \Big\} \tag{2.21}$$

and the optimal Bellman equation:

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \Big\{ \tilde{\rho}(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a') \Big\} \tag{2.22}$$

Bellman equation in matrix form

Equation (2.21) is a system of linear equations and can be rewritten as:

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \Big\{ \tilde{\rho}(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a') \Big\}$$

and in matrix form

$$\mathbf{Q}_\pi = \mathbf{R} + \gamma \mathbf{P} \mathbf{\Pi}_\pi \mathbf{Q}_\pi \tag{2.23}$$

where $\mathbf{Q}_\pi$ and $\mathbf{R}$ are vectors of size $|\mathcal{S}||\mathcal{A}|$, $\mathbf{P}$ is a stochastic matrix of size $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ that contains transition probabilities :

$$\mathbf{P}\big((s,a), s'\big) = \bar{\mathcal{P}}(s,a,s'),$$

and $\mathbf{\Pi}_\pi$ is a stochastic matrix of size $|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$

$$\mathbf{\Pi}_\pi\big(s', (s', a')\big) = \pi(a'|s')$$

Equation (2.23) is a linear system that can be solved analytically or iteratively:

$$\mathbf{Q}_\pi = (\mathbf{I} - \gamma \mathbf{P} \mathbf{\Pi}_\pi)^{-1} \mathbf{R}$$

Optimal V-function

The optimal V^*-function is repeated here:

$$V^*(s) = \max_\pi V_\pi(s)$$

The optimal policy π^* is given by:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} E_{s' \sim \tilde{\mathcal{P}}(s,a,\cdot)} \Big\{ \tilde{\rho}(s,a,s') + \gamma V^*_\pi(s') \Big\} \tag{2.24}$$

The Bellman equation for the V-function is:

$$V_\pi(s) = E_{s' \sim \tilde{\mathcal{P}}(s,\pi(s),\cdot)} \Big\{ \tilde{\rho}(s,\pi(s),s') + \gamma V_\pi(s') \Big\} \tag{2.25}$$

and the Bellman equation for the optimal V^*-function is:

$$V^*(s) = \max_{a \in \mathcal{A}} E_{s' \sim \tilde{\mathcal{P}}(s,a,\cdot)} \Big\{ \tilde{\rho}(s,a,s') + \gamma V^*(s') \Big\} \tag{2.26}$$

In the case of a countable (e.g., discrete) state space, the optimal Bellman equation becomes:

$$V^*(s) = \max_{a' \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \Big\{ \tilde{\rho}(s,a,s') + \gamma V^*(s') \Big\} \tag{2.27}$$

2.1.2.3 Bellman optimality operator - contraction mapping

The Bellman optimality operator T is a mapping from value functions to value functions. Let $\mathcal{Q}$ be the space of all Q-functions, then T is a mapping $T : \mathcal{Q} \to \mathcal{Q}$. Using the Bellman equations, the Bellman optimality operator T is defined as:

$$[TQ](s,a) = \rho(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_\pi(\mathcal{P}(s,a), a'), \text{ for deterministic setting} \tag{2.28}$$

$$[TQ](s,a) = E_{s' \sim \tilde{\mathcal{P}}(s,a,\cdot)} \left\{ \tilde{\rho}(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q_\pi(s',a') \right\}, \text{ for stochastic setting} \tag{2.29}$$

$$[TQ](s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \left\{ \tilde{\rho}(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q(s',a') \right\}, \text{ for countable spaces} \tag{2.30}$$

It maps a Q-function to an improved Q-function as follows:

$$Q \leftarrow TQ \tag{2.31}$$

The above equation is a contraction mapping with rate equal to the MDP discount factor $\gamma \in (0,1)$ under the infinity norm:

$$\|TQ - TQ'\|_\infty \leqslant \gamma \|Q - Q'\|_\infty$$

The contraction mapping $T : \mathcal{Q} \to \mathcal{Q}$ has a unique fixed point, i.e. the equation $Q = TQ$ has a unique solution Q^*, where $Q^* = TQ^*$ is the optimal solution, and there is no further improvement.

The Bellman equation is appropriate for iterative solution, since in each iteration k, T maps Q_k to an improved Q_{k+1}: Iterative solution

$$Q_{k+1} = TQ_k$$

2.2 Algorithms for Solving MDPs

The solution of an MDP is to find an optimal policy π^*, that is, one that maximizes the cumulative discounted return. In this section, we assume that there are enough computational resources (memory) to represent the V-value or Q-value of any state. There are two main categories of algorithms: the *model-based* or *dynamic programming* algorithms, where the transfer function $\mathcal{P}$ and the reward function ρ are known, and the *model-free* or *reinforcement learning* algorithms, where the data are obtained from interaction with the process. The most commonly used approaches in each category are, value iteration and policy iteration. Both approaches can be formulated in terms of either V or Q values. However, due to our interest in learning methods, in this section, we will focus only on the Q-values formulation of value iteration and policy iteration.

2.2.1 Value Iteration

Model-based value iteration

Value iteration (Bellman, 1957b) algorithm is just a repeated application of the Bellman optimality operator to the state action value function Q, shown in Algorithm 1. The Q-function is initialized to an arbitrary value, e.g., 0, and eventually, converges to the optimal Q^*. An optimal policy π^* is derived from a greedy policy over the computed Q^*. The algorithm terminates when the maximum change of the Q-values between successive iterations, is less than the stopping criterion ϵ, a small positive value. The loss is bounded by (Williams and Baird, 1993):

$$\|Q - Q^*\|_\infty \leqslant 2\epsilon\frac{\gamma}{1-\gamma}$$

where γ is the discount factor. The cost per iteration is $O(|\mathcal{S}|^2|\mathcal{A}|)$, but the number of iterations required to achieve a certain level of accuracy can grow exponentially with the contraction rate.

2.2.2 Q-Learning

Model-free value iteration. Q-Learning

Q-Learning algorithm (Watkins, 1989; Watkins and Dayan, 1992) is a widely used model-free value iteration algorithm. It uses action value function $Q(s, a)$ initialized to an arbitrary value, usually zero. Q values are updated without

Algorithm 1 Value iteration for stochastic countable state space

```
1: Input: set of states $\mathcal{S}$, set of actions $\mathcal{A}$, transition function $\bar{\mathcal{P}}$, immediate
   reward function $\tilde{\rho}$, stopping criterion $\epsilon$
2: $Q = 0$   {Initialize arbitrarily}
3: repeat
4:   $Q' = Q$
5:   for $every\_pair(s \in \mathcal{S}, a \in \mathcal{A})$
6:       $Q(s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \Big\{ \tilde{\rho}(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q'(s',a') \Big\}$
7:   end
8: until $\|Q - Q'\|_\infty < \epsilon$
9: $\forall s \in \mathcal{S}, \quad \pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$
10: return $\pi$
```

requiring a model. Updates are based on transition samples that come from the process as tuples of $(s_l, a_l, s_{l+1}, r_{l+1})$, where s_l is the MDP state at time l, a_l is the action taken at time l, s_{l+1} the next state at time $l+1$, and r_{l+1} the reward given at time $l+1$ for taking action a_l at state s_l and moving to state s_{l+1}. For each tuple, the update to the Q values is the following:

$$Q_{l+1}(s_l, a_l) = Q_l(s_l, a_l) + \alpha_l \big[r_{l+1} + \gamma \max_{a' \in \mathcal{A}} Q_l(s_{l+1}, a') - Q_l(s_l, a_l) \big] \qquad (2.32)$$

where $\alpha_l \in (0,1]$ is the learning rate, and l is the time step. The term $r_{l+1} + \gamma \max_{a' \in \mathcal{A}} Q_l(s_{l+1}, a')$ is the updated estimate of the $Q(s_l, a_l)$ and the term $Q_l(s_l, a_l)$ is its current estimate. The difference between these two terms is called the temporal difference. The Q_{l+1} value in the next time step is the current Q_l value updated by the learning rate portion of the temporal difference. Q-learning asymptotically converges to the optimal Q^* state-action value function under certain conditions:

Learning rate

1. The state and action spaces are finite.

2. The sum

$$\sum_{l=0}^{\infty} \alpha_l = \infty, \qquad \text{w.p.1}$$

3. There exist some constant $C \in \mathbb{R}$ such that

$$\sum_{l=0}^{\infty} \alpha_l^2 \leqslant C, \qquad \text{w.p.1}$$

4. All state-actions pairs are visited infinitely often.

Conditions 2 and 3 are easily satisfiable, e.g., $\alpha_l = 1/l$. These conditions have been studied by Watkins and Dayan (1992); Tsitsiklis (1994); Jaakkola et al. (1994).

ϵ-greedy action selection

Algorithm 2 presents Q-learning with ϵ-greedy exploration. The ϵ-greedy action selection, was suggested by Sutton and Barto (1998) and selects action a_l using the current policy with probability $1-\epsilon$ (exploitation), or a uniformly random action with probability ϵ (exploration). As time passes, the learned value function and policy improve, and the need for exploration diminishes. Thus, ϵ may be implemented as a function of time, e.g., $\epsilon_l = 1/l$.

Algorithm 2 Q-learning with ϵ-greedy exploration

```
Input: set of states $\mathcal{S}$, set of actions $\mathcal{A}$, exploration probability $\epsilon_l$, learning rate $\alpha_l$
initialize $Q_0$ arbitrarily, e.g. $Q_0(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
draw $s_0$ from initial state distribution
for time step $l = 0, 1, 2, \ldots, H$
    $a_l = \begin{cases} a \in \arg\max_{a' \in \mathcal{A}} Q_l(s_l, a'), & \text{with probability } 1-\epsilon_l \quad \text{// exploitation} \\ \text{a uniformly random action}, & \text{with probability } \epsilon_l \quad \text{// exploration} \end{cases}$
    take action $a_l$ in $s_l$ and get next state $s_{l+1}$ and reward $r_{l+1}$
    $Q_{l+1}(s_l, a_l) = Q_l(s_l, a_l) + \alpha_l[r_{l+1} + \gamma \max_{a' \in \mathcal{A}} Q_l(s_{l+1}, a') - Q_l(s_l, a_l)]$
end
$\forall s \in \mathcal{S}, \quad \pi(s) = \arg\max_{a \in \mathcal{A}} Q_{H+1}(s,a)$
return $\pi$
```

Q-learning is an "off policy" algorithm (Sutton and Barto, 1998), because it evaluates a greedy policy and uses a different policy for data collection, the one that controls the process of exploration.

2.2.3 Policy iteration

Policy iteration is a general algorithm (Figure 2.2) for solving MDPs, and it is a two step iterative process. In each iteration, the first step, *policy evaluation*, is the computation of the value function for the current policy. The second one, *policy improvement*, is the improvement of the current policy in a greedy way. The initial policy can be a random or an arbitrary one. Policy iteration, shown in Algorithm 3, converges to an optimal policy, usually in just a few iterations.

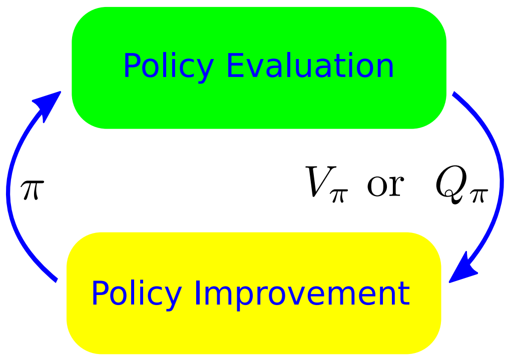


FIGURE 2.2: Policy iteration process. In the policy evaluation step, the value function is calculated for some or all states using the current policy. In the policy improvement step, the algorithm improves the previous policy based on values obtained in the policy evaluation step

Algorithm 3 Policy iteration

```
1:  Input: set of states $\mathcal{S}$, set of actions $\mathcal{A}$
2:  initialize $\pi$ arbitrarily
3:  repeat
4:      $\pi' = \pi$
5:      {Policy evaluation}
6:      compute $Q$ for policy $\pi$ e.g. by solving Bellman equation
7:      {Policy improvement}
8:      $\forall s \in \mathcal{S}, \quad \pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$
9:  until $\pi = \pi'$
10: return $\pi$
```

In the case that the model is known, i.e., transition and reward functions are known, the policy evaluation step is achieved by solving the Bellman equation. Bellman operator for policy π is a mapping $T_\pi : \mathcal{Q} \to \mathcal{Q}$ where $\mathcal{Q}$ is the space of all Q-functions.

Bellman operator for policy

The Bellman operator T_π for policy π is defined as:

$$[T_\pi Q](s,a) = \rho(s,a) + \gamma Q(\mathcal{P}(s,a), \pi(\mathcal{P}(s,a))), \text{ for deterministic setting} \tag{2.33}$$

$$[T_\pi Q](s,a) = E_{s' \sim \tilde{\mathcal{P}}(s,a,\cdot)} \{\tilde{\rho}(s,a,s') + \gamma Q(s', \pi(s')))\}, \text{ for stochastic setting} \tag{2.34}$$

$$[T_\pi Q](s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \{\tilde{\rho}(s,a,s') + \gamma Q(s', \pi(s'))\}, \text{ for countable spaces} \tag{2.35}$$

The iterative form of the Bellman equation solution for policy π and iteration k starting from an arbitrary Q_π^0

$$Q_\pi^{k+1} = T_\pi(Q_\pi^k) \tag{2.36}$$

will converge to a fixed point Q_π, since the operator T_π is a contraction, i.e., for discount factor $\gamma < 1$ and any pair of Q_π^k, Q_π functions:

$$\|T_\pi Q_\pi^k - T_\pi Q_\pi\|_\infty \leqslant \gamma \|Q_\pi^k - Q_\pi\|_\infty$$

Therefore, Q_π^k will converge to Q_π for increasing iterations k. The complete algorithm is shown as Algorithm 4.

Algorithm 4 Policy iteration

```
**Input:** set of states $\mathcal{S}$, set of actions $\mathcal{A}$, transition function $\bar{\mathcal{P}}$, immediate reward function $\tilde{\rho}$, stopping criterion $\epsilon$

initialize $\pi$, $Q$ arbitrarily
repeat
   $\pi' = \pi$
   {Policy Evaluation}
   repeat
      $Q' = Q$
      for $every\_pair(s \in \mathcal{S}, a \in \mathcal{A})$
         $Q(s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \{\tilde{\rho}(s,a,s') + \gamma Q'(s', \pi(s'))\}$
      end
   until $\|Q - Q'\|_\infty < \epsilon$
   {Policy Improvement}
   $\forall s \in \mathcal{S}, \quad \pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$
until $\pi = \pi'$
return $\pi$
```

Bellman equation 2.35 for policy iteration has a significant advantage over Bellman optimality equation 2.30 for value iteration, since the first one is linear, while the Bellman optimality equation is highly nonlinear due to the max operator at the right-hand side. Policy evaluation is easier to compute than value iteration.

2.2.4 SARSA

Model-free policy iteration

Policy iteration can be used in case the model is not known. Here we will describe a SARSA algorithm (Rummery and Niranjan, 1994). The name comes from the form of training data tuples, $(s_l, a_l, r_{l+1}, s_{l+1}, a_{l+1})$, i.e., at the current state s_l action a_l was taken and a reward r_{l+1} was given for moving to the next state s_{l+1}, where the next action taken was a_{l+1}. The SARSA algorithm starts with an arbitrary Q-value (Q_0) and updates Q_l values at each time step using the tuples described above.

$$Q_{l+1}(s_l, a_l) = Q_l(s_l, a_l) + \alpha[r_{l+1} + \gamma Q_l(s_{l+1}, a_{l+1}) - Q_l(s_l, a_l)] \tag{2.37}$$

where $\alpha \in (0, 1]$ is the learning rate and l is the time step. The term $r_{l+1} + \gamma Q_l(s_{l+1}, a_{l+1})$ is the updated estimate of the $Q(s_l, a_l)$ and the term $Q_l(x_l, a_l)$ is its current estimate. The difference between two terms is called the temporal difference. The Q_{l+1} value in the next time step is the current Q_l estimate updated by the learning rate portion of the temporal difference. Algorithm 5 presents SARSA with ϵ-greedy exploration. Learning rate a_l and ϵ-greedy exploration is discussed at the Q-learning section 2.2.2.

Algorithm 5 SARSA with ϵ-greedy exploration

```
Input: set of states $\mathcal{S}$, set of actions $\mathcal{A}$, exploration probability $\epsilon_l$, learning rate $\alpha_l$

initialize $Q_0$ arbitrarily, e.g. $Q_0(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
draw $s_0$ from initial state distribution
$a_0$ = a uniformly random available action

for time step $l = 0, 1, 2, \ldots, H$
    take action $a_l$ in $s_l$ and get next state $s_{l+1}$ and reward $r_{l+1}$

    compute next action $a_{l+1}$
    $a_{t+1} = \begin{cases} a \in \arg\max_{a' \in \mathcal{A}} Q_t(s_{l+1}, a'), & \text{with probability } 1 - \epsilon_l \quad // \text{ exploitation} \\ \text{a uniformly random action}, & \text{with probability } \epsilon_l \quad // \text{ exploration} \end{cases}$

    $Q_{l+1}(s_l, a_l) = Q_l(s_l, a_l) + \alpha_l[r_{l+1} + \gamma Q_l(s_{l+1}, a_{l+1}) - Q_l(s_l, a_l)]$
end
$\forall s \in \mathcal{S}, \quad \pi(s) = \arg\max_{a \in \mathcal{A}} Q_{H+1}(s, a)$
return $\pi$
```

SARSA is an "on policy" algorithm (Sutton and Barto, 1998), because it evaluates the policy that is used to control the process and collect the data.

2.3 Approximation Architectures

In the case of very large or infinite state space, exact representation is infeasible. Exact value function, either V or Q, requires storage of the return for every state or state-action pair respectively. Given that the value function is smooth enough, there are satisfactory approximation methods. Looking back at value iteration algorithm (Algorithm 1), there are some questions about how to modify the algorithm from exact (tabular) value function representation to an approximated one. In particular, there are issues beyond the obvious approximation error:

1. lines 5 to 7, sweep through all states performing Bellman backups on each one, which is impractical or even impossible in large state spaces.

2. line 6, moving from state s after taking action a to a next state $s' \in \mathcal{S}$, s' can be any possible state in the state space and each one of them must be accounted for, an enormous task to deliver.

3. line 6, assigns a value to $Q(s, a)$, but it is not clear how to assign a single value in a non-tabular, approximate representation.

Let's now take a look at the trajectory-sampled value iteration algorithm (Algorithm 6) and discuss the alternatives to issues 1 and 2. The first issue is resolved by replacing global state sweeps, with multiple trajectory sweeps using the current policy (Barto et al., 1995):

$$\{(s_i, a_i, r_{i+1})\}_{i=0}^{H}$$

where H is the trajectory length. The second one is alleviated by drawing L_1 samples, possible next states s'_j, per state s and action a from the $\bar{\mathcal{P}}(s, a, \cdot)$ distribution. Given state s and action a, $Q(s, a)$ is estimated as:

$$\frac{1}{L_1}\sum_{i=1}^{L_1}\left\{\tilde{\rho}(s, a, s'_i) + \gamma \max_{a' \in \mathcal{A}} Q(s'_i, a')\right\}$$

As $L_1 \to \infty$, the estimate becomes exact with probability one. The third issue will be discussed in next subsection in detail. For now, let us assume that the state action value function has a tabular representation. Also, note that there is no reason to store the policy explicitly; it is easily extracted from Q:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

Algorithm 6 Trajectory sampled value iteration

```
 1: Input: set of states $\mathcal{S}$, set of actions $\mathcal{A}$, transition function $\bar{\mathcal{P}}$, immediate
    reward function $\tilde{\rho}$, stopping criterion $\epsilon$, number of samples $L_1$ per state
 2: $Q = 0$    {Initialize arbitrarily}
 3: repeat {for each trajectory}
 4:   $Q' = Q$
 5:   repeat {for each $(s, a)$ pair in a trajectory following current policy}
 6:     Draw $L_1$ samples (next states) $\left\{s'_i \sim \bar{\mathcal{P}}(s, a, \cdot), i = 1..L_1\right\}$
 7:     $Q(s,a) = \frac{1}{L_1} \sum_{i=1}^{L_1} \left\{\tilde{\rho}(s,a,s'_i) + \gamma \max_{a' \in \mathcal{A}} Q'(s'_i, a')\right\}$
 8:   until the end of trajectory, i.e., reaching a terminal state
 9: until $\|Q - Q'\|_\infty < \epsilon$
10: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$ {there is no need to store policy}
11: return $\pi$
```

Classification of approximation methods is based on the explicit or implicit use of parameters. *Parametric approximation* uses explicitly declared parameters, while *non-parametric approximation* uses implicitly declared parameters that are automatically extracted from the given data.

2.3.1 Parametric Approximation

In this section we will study Q-function approximation (Sutton and Barto, 1998; Bertsekas and Tsitsiklis, 1996). The exact $Q(s,a)$ function will be replaced by the approximated function $\widehat{Q}(s,a;\boldsymbol{w})$, where $\boldsymbol{w}$ is a vector of n parameters forming the parameter space. Let's define operator F to be an approximation mapping $F : \mathbb{R}^n \to \mathcal{Q}$, from the parameter space $\boldsymbol{w} \in \mathbb{R}^n$ to the space of value functions $\mathcal{Q}$:

$$\widehat{Q}(s,a;\boldsymbol{w}) = [F\boldsymbol{w}](s,a)$$

Instead of storing a Q-value for every pair (s, a), we store only n parameters $\boldsymbol{w} = (w_1, w_2, \cdots, w_n)$. Usually the size n of the parameter space is much smaller than the size of the state-action space $(|\mathcal{S}| \cdot |\mathcal{A}|)$. However, since operator F maps parameters $\boldsymbol{w}$ to a subset of all possible value functions $\mathcal{Q}$, it introduces an approximation error.

Operator F may be nonlinear in terms of the parameter vector $\boldsymbol{w}$. However, linearly parameterized approximators are well studied, the resulting algorithms are easy to implement, and their theoretical properties easy to analyze. A common form of the approximated Q-function, $\widehat{Q}$, is a *weighted linear combination* of a vector $\boldsymbol{\phi}$ of n *basis functions* or *features*

$$\boldsymbol{\phi}(s, a) = \begin{bmatrix} \phi_1(s, a) \\ \phi_2(s, a) \\ \vdots \\ \phi_n(s, a) \end{bmatrix}$$

where $\phi_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a vector $\boldsymbol{w}$ of n parameters:

$$\begin{aligned} \widehat{Q}(s, a; \boldsymbol{w}) &= [F\boldsymbol{w}](s, a) \\ &= \sum_{i=1}^{n} \phi_i(s, a) w_i \\ &= \boldsymbol{\phi}^T(s, a)\boldsymbol{w} \end{aligned} \tag{2.38}$$

$\widehat{Q}$ approximation in matrix form

In the case of discrete state and action spaces the above equation 2.38 may be rewritten in matrix form:

$$\widehat{\boldsymbol{Q}} = \boldsymbol{\Phi}\boldsymbol{w} \tag{2.39}$$

where $\widehat{\boldsymbol{Q}}$ is a vector of size $|\mathcal{S}||\mathcal{A}|$ containing all $Q(s, a)$ values:

$$\widehat{\boldsymbol{Q}} = \begin{bmatrix} Q(s_1, a_1) \\ Q(s_1, a_2) \\ \vdots \\ Q(s_2, a_1) \\ Q(s_2, a_2) \\ \vdots \\ Q(s_{|\mathcal{S}|}, a_{|\mathcal{A}|}) \end{bmatrix}$$

and $\boldsymbol{\Phi}$ is a matrix of size $|\mathcal{S}||\mathcal{A}| \times n$:

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_1(s_1, a_1) & \phi_2(s_1, a_1) & \cdots & \phi_n(s_1, a_1) \\ \phi_1(s_1, a_2) & \phi_2(s_1, a_2) & \cdots & \phi_n(s_1, a_2) \\ & \vdots & & \\ \phi_1(s_2, a_1) & \phi_2(s_2, a_1) & \cdots & \phi_n(s_2, a_1) \\ \phi_1(s_2, a_2) & \phi_2(s_2, a_2) & \cdots & \phi_n(s_2, a_2) \\ & \vdots & & \\ \phi_1(s_{|\mathcal{S}|}, a_{|\mathcal{A}|}) & \phi_2(s_{|\mathcal{S}|}, a_{|\mathcal{A}|}) & \cdots & \phi_n(s_{|\mathcal{S}|}, a_{|\mathcal{A}|}) \end{bmatrix}$$

Each row of Φ contains the value of all the basis functions for a specific (s, a) pair and each column of Φ contains the value of a particular basis function for all pairs (s, a). The basis functions $\boldsymbol{\phi}$ are usually implemented as polynomial functions or Gaussian radial basis functions (RBF). The approximation is clearly linear with respect to the parameters $\boldsymbol{w}$, whereas in general the basis functions $\boldsymbol{\phi}$ are nonlinear. The linearity regarding parameters $\boldsymbol{w}$ greatly simplifies the analysis of approximated models. Parametric approximation efficacy depends on the design of the parametric model, i.e., the selection of basis functions, which is not a trivial task. Too many basis functions will result in an increased computational time and may introduce numerical errors, while too few may result in an inadequate representation. There are algorithms for automatic selection of the basis functions.

Figure 2.3 shows the best approximation $\widehat{\boldsymbol{Q}}$ of the state action value function $\boldsymbol{Q}$ given some set of basis functions. $\boldsymbol{Q} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ is projected onto the subspace spanned by matrix $\boldsymbol{\Phi}$, which depends on the selection of basis functions. The parameter vector $\boldsymbol{w}$ is acquired during the value or policy function optimization process.

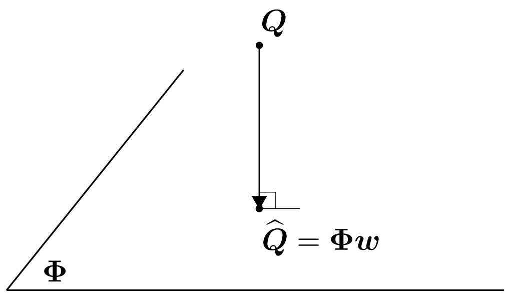


FIGURE 2.3: Projection of state-action value function Q onto the space spanned matrix $\boldsymbol{\Phi}$

Trajectory sampled value iteration algorithm

Let's now see the approximated version of our previous algorithm (Algorithm 6) "trajectory sampled value iteration". We use the above linear approximation $\widehat{Q}(s,a) = \boldsymbol{\phi}^T(s,a)\boldsymbol{w}$ for state action value function. The question now is to transform the update of $Q(s,a)$ at line 7 into an update of the $\boldsymbol{w}$ parameters. Let $Q(s,a)$ be the current estimate and Q_{new} the new target value. The idea is to incrementally update the parameters $\boldsymbol{w}$ using the reverse direction of the squared difference between $Q(s,a)$ and Q_{new}.

$$\begin{aligned}
\boldsymbol{\delta} &= \frac{\partial (Q_{new} - Q(s,a))^2}{\partial \boldsymbol{w}} \\
&= \frac{\partial (Q_{new} - \boldsymbol{\phi}^T(s,a)\boldsymbol{w})^2}{\partial \boldsymbol{w}} \\
&= -2\boldsymbol{\phi}(s,a)(Q_{new} - \boldsymbol{\phi}^T(s,a)\boldsymbol{w}) \qquad (2.40) \\
&= -2\boldsymbol{\phi}(s,a)(Q_{new} - Q(s,a)) \qquad (2.41)
\end{aligned}$$

The rule for parameter update is $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\boldsymbol{\delta}$ where $\alpha \in (0,1]$ is the learning rate. The complete algorithm for "trajectory sampled value iteration with approximated Q-function" is Algorithm 7.

Algorithm 7 Trajectory sampled value iteration - Approximated Q-function

```
1: Input: set of states $\mathcal{S}$, set of actions $\mathcal{A}$, transition function $\bar{\mathcal{P}}$, immediate
   reward function $\tilde{\rho}$, stopping criterion $\epsilon$, number of samples $L_1$ per state
2: $\boldsymbol{w} = 0$   {Initialize arbitrarily}
3: repeat {for each trajectory}
4:    $\boldsymbol{w}' = \boldsymbol{w}$
5:    repeat {for each $(s,a)$ pair in a trajectory following current policy}
6:       Draw $L_1$ samples (next states) $\{s'_i \sim \bar{\mathcal{P}}(s,a,\cdot)\}_{i=1}^{L_1}$
7:       $Q_{new} = \frac{1}{L_1}\sum_{i=1}^{L_1}\left\{\tilde{\rho}(s,a,s'_i) + \gamma \max_{a' \in \mathcal{A}} Q'(s'_i,a')\right\}$
8:       $\boldsymbol{\delta} = -2\boldsymbol{\phi}(s,a)(Q_{new} - Q(s,a))$
9:       $\boldsymbol{w} = \boldsymbol{w} - \alpha\boldsymbol{\delta}$
10:   until {the end of trajectory, i.e. a terminal state, is reached }
11: until $\|\boldsymbol{w}' - \boldsymbol{w}\| < \epsilon$ or {there is no more time}
12: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$ {there is no need to store policy}
13: return $\pi$
```

2.3.2 Non-parametric Approximation

In the case of non-parametric approximation, there are parameters, and they are automatically extracted from the data. Typical examples of the non-parametric approximator class are the kernel based ones. The unknown $Q(s,a)$ value for some input pattern (s,a) is expected to have some similarity with the known $Q(s_i,a_i)$ value for the input pattern (s_i,a_i) used for training. A similarity measure (the kernel) that serves this purpose has the form:

$$\kappa : |\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}| \times |\mathcal{A}| \to \mathbb{R}$$

$$(s,a,s_i,a_i) \mapsto \kappa(s,a,s_i,a_i),$$

where κ is a function of two input patterns (s,a) and (s_i,a_i) and returns a real value as a similarity measure. The kernel function κ is a symmetric function. The approximated Q-function takes the following form:

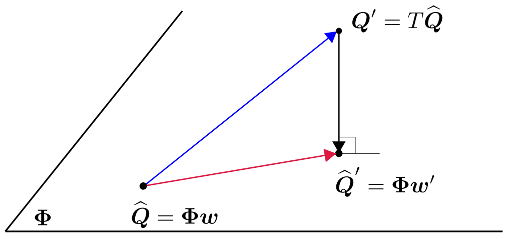
$$\widehat{Q}(s,a) = \sum_{i=1}^{m} \boldsymbol{w}_i \kappa(s,a,s_i,a_i) + \boldsymbol{w}_0$$

where $\{(s_i,a_i)\}_{i=1}^{m}$, are input patterns that come from data, $\boldsymbol{w}_i$ coefficients are non-zero values given by the optimization process, and $\boldsymbol{w}_0$ is the bias. A common choice for kernels are polynomial functions or Gaussian radial basis functions (RBF). Kernel based machine learning algorithms were introduced by Boser et al. (1992). An extensive study of kernel based algorithms was done by Scholkopf and Smola (2001).

2.3.3 Projection Methods for Linear Architectures

Here we discuss two methods for obtaining the parameter vector $\boldsymbol{w}$ optimized for the nearest approximation $\widehat{\boldsymbol{Q}}$ to $\boldsymbol{Q}$ (Figure 2.3), given the basis functions, i.e. matrix $\boldsymbol{\Phi}$. Recall that the exact $\boldsymbol{Q}$ is not known, and these methods have to rely on the information contained in the Bellman equation and the Bellman operator to find a "good" $\boldsymbol{w}$. Figure 2.4 illustrates an $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ space, the space of exact Q-functions, and a plane spanned by $\boldsymbol{\Phi}$, the subspace of approximated Q-functions and demonstrates the application of Bellman operator T

on approximated function $\widehat{\boldsymbol{Q}}$ giving $\boldsymbol{Q}'$. Finally $\boldsymbol{Q}'$ is projected back on plane $\widehat{\boldsymbol{Q}}'$, and the distance $\|\widehat{\boldsymbol{Q}}' - \boldsymbol{Q}'\|$ is minimum. The first method for projection minimizes the blue line and the second the red one.



$\boldsymbol{Q}' = T\widehat{\boldsymbol{Q}}$
$\widehat{\boldsymbol{Q}}' = \boldsymbol{\Phi w}'$
$\boldsymbol{\Phi}$
$\widehat{\boldsymbol{Q}} = \boldsymbol{\Phi w}$

FIGURE 2.4: The plane is spanned by $\boldsymbol{\Phi}$ and is the subspace of approximated Q-functions and $\widehat{Q}$ is a point on it. Operator T maps $\widehat{Q}$ to a point Q' anywhere in the space of Q-functions and, in general, outside the plane. Approximation of Q' is given by its $\widehat{Q}'$ projection onto the plane

2.3.3.1 Bellman Residual Minimizing Approximation

We are repeating here, for convenience, the matrix form of the Bellman equation (2.23) for finite discrete state and action spaces.

$$\mathbf{Q} = \mathbf{R} + \gamma \mathbf{P\Pi}_\pi \mathbf{Q}$$

We replace the $\boldsymbol{Q}$ with the approximated $\widehat{\boldsymbol{Q}}$ and if the approximation is successful the two sides of the equation should be close,

$$\widehat{\boldsymbol{Q}} \approx \mathbf{R} + \gamma \mathbf{P\Pi}_\pi \widehat{\boldsymbol{Q}}$$

The result is an overconstrained system of $|\boldsymbol{w}|$ equations. Replacing $\widehat{\boldsymbol{Q}}$ by $\Phi \boldsymbol{w}$

$$\Phi \boldsymbol{w} \approx \mathbf{R} + \gamma \mathbf{P\Pi}_\pi \Phi \boldsymbol{w}$$

and reforming the equation to get the parameter vector $\boldsymbol{w}$

$$(\Phi - \gamma \mathbf{P\Pi}_\pi \Phi)\boldsymbol{w} \approx \mathbf{R}$$

This is overconstrained linear system of equations in the form $\mathbf{A}\boldsymbol{w} = \mathbf{R}$. Where $\mathbf{A} = \Phi - \gamma \mathbf{P\Pi}_\pi \Phi$. The least squares solution is

$$\boldsymbol{w} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{R}$$

This method is graphically shown at Figure 2.4 as minimization of the blue line, and is trying to find the point on the $\boldsymbol{\Phi}$ plane where the Bellman operator makes the smallest jump towards the true value function, $\boldsymbol{Q}$.

2.3.3.2 Least-Squares Fixed-Point Approximation

The second method seeks to minimize the red line in Figure 2.4. In essence, it attempts to find a fixed point (see $\widehat{\boldsymbol{Q}}$) on the $\boldsymbol{\Phi}$ plane under one application of the Bellman operator (see $\boldsymbol{Q}'$) followed by orthogonal project back to the $\boldsymbol{\Phi}$ plane (see $\widehat{\boldsymbol{Q}}'$). We will see that this (red line) error can be made zero, therefore this fixed point property can be actually achieved. This second objective is formulated as

$$\min_{\boldsymbol{w}} \|\widehat{\boldsymbol{Q}} - \widehat{\boldsymbol{Q}}'\|^2$$

Projection of $\boldsymbol{Q}'$ onto the plane is orthogonal, and the projection operator is $P = \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T$. The goal becomes

$$\min_{\boldsymbol{w}} \|\widehat{\boldsymbol{Q}} - \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T(T\widehat{\boldsymbol{Q}})\|^2$$

We are forcing the above expression to zero, at the fixed point ($\widehat{\boldsymbol{Q}} = \boldsymbol{Q}'$), and we solve the system of equations, finding the parameters $\boldsymbol{w}$.

$$\begin{aligned}
\widehat{\boldsymbol{Q}} &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T(T\widehat{\boldsymbol{Q}}) \\
\widehat{\boldsymbol{Q}} &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T(\mathbf{R} + \gamma\mathbf{P}\boldsymbol{\Pi}\widehat{\boldsymbol{Q}}) \\
\boldsymbol{\Phi}\boldsymbol{w} &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T(\mathbf{R} + \gamma\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Phi}\boldsymbol{w}) \\
\boldsymbol{w} &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T(\mathbf{R} + \gamma\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Phi}\boldsymbol{w}) \\
\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} &= \boldsymbol{\Phi}^T(\mathbf{R} + \gamma\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Phi}\boldsymbol{w}) \\
\boldsymbol{\Phi}^T\mathbf{R} &= \boldsymbol{\Phi}^T(\boldsymbol{\Phi} - \gamma\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Phi})\boldsymbol{w} \\
\boldsymbol{w} &= (\boldsymbol{\Phi}^T(\boldsymbol{\Phi} - \gamma\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Phi}))^{-1}\boldsymbol{\Phi}^T\mathbf{R} \qquad (2.42)
\end{aligned}$$

The expression $\boldsymbol{\Phi}^T(\boldsymbol{\Phi} - \gamma\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Phi})$ is an $n \times n$ matrix, where n is the number of the basis functions, and this system has a solution as long as this matrix can be inverted.

The $\widehat{\boldsymbol{Q}}$ calculated using this method may be far from $\boldsymbol{Q}$. It has been shown that under certain conditions the approximation error is bounded (Tsitsiklis and Roy, 1999; Bertsekas, 2007).

$$\|\boldsymbol{Q} - \widehat{\boldsymbol{Q}}\| \leqslant \frac{1}{\sqrt{1-\gamma^2}} \|\boldsymbol{Q} - P\boldsymbol{Q}\|$$

There is an extended error bound study for approximations from projected linear equations (Yu and Bertsekas, 2010).

2.3.4 Least-Squares Policy Iteration

Here we will study an advanced reinforcement learning algorithm, the Least-Squares Policy Iteration (LSPI) (Lagoudakis and Parr, 2003b). First, we will cover the policy evaluation part, which is based on a modification of the Least-Squares Temporal Difference (LSTD) Learning algorithm (Bradtke and Barto, 1996), and then we will integrate it into a complete policy iteration algorithm following a greedy fashion.

Let us continue from the previous equation 2.42, the solution of the system of linear equations $\mathbf{A}\boldsymbol{w}^T = \mathbf{b}$, where $\mathbf{A} = \mathbf{\Phi}^T(\mathbf{\Phi} - \gamma\mathbf{P\Pi\Phi})$ and $\mathbf{b} = \mathbf{\Phi}^T\mathbf{R}$. In the general case, matrix $\mathbf{P}$ and vector $\mathbf{R}$ are unknown or impractical to compute, either because the complete MDP model is not available or because the state space is huge. However, in such cases, matrix $\mathbf{A}$ and vector $\mathbf{b}$ of the linear system can be estimated. Let's take a closer look at their structure:

$$\begin{aligned}\mathbf{A} &= \mathbf{\Phi}^T(\mathbf{\Phi} - \gamma\mathbf{P\Pi\Phi}) \\ &= \sum_{s\in S}\sum_{a\in\mathcal{A}} \boldsymbol{\phi}(s,a)\left(\boldsymbol{\phi}(s,a) - \gamma\sum_{s'\in S}\bar{\mathcal{P}}(s,a,s')\boldsymbol{\phi}\big(s',\pi(s')\big)\right)^T \\ &= \sum_{s\in S}\sum_{a\in\mathcal{A}}\sum_{s'\in S}\bar{\mathcal{P}}(s,a,s')\left[\boldsymbol{\phi}(s,a)\Big(\boldsymbol{\phi}(s,a) - \gamma\boldsymbol{\phi}\big(s',\pi(s')\big)\Big)^T\right] \end{aligned} \tag{2.43}$$

$$\begin{aligned}\mathbf{b} &= \mathbf{\Phi}^T\mathbf{R} \\ &= \sum_{s\in S}\sum_{a\in\mathcal{A}} \boldsymbol{\phi}(s,a)\sum_{s'\in S}\bar{\mathcal{P}}(s,a,s')\tilde{\rho}(s,a,s') \\ &= \sum_{s\in S}\sum_{a\in\mathcal{A}}\sum_{s'\in S}\bar{\mathcal{P}}(s,a,s')\Big[\boldsymbol{\phi}(s,a)\tilde{\rho}(s,a,s')\Big] \end{aligned} \tag{2.44}$$

Examining their strcture, it is easy to see that matrix $\mathbf{A}$ and vector $\mathbf{b}$ can be easily estimated from a set of samples of interaction with the process in the form of tuples $S = \{(s_i, a_i, r'_i, s'_i)\}_{i=1}^m$. These samples can be collected from complete trajectories in the state space or, in the presence of a simulator, sampled next states s' and rewards r' can be drawn from $\bar{\mathcal{P}}(s,a,\cdot)$ and $\tilde{\rho}(s,a,s')$ given any state s and action a. For m samples, $\mathbf{A}$ and $\mathbf{b}$ can be estimated as:

$$\widetilde{\mathbf{A}} = \frac{1}{m}\sum_{i=1}^{m}\left[\boldsymbol{\phi}(s_i,a_i)\Big(\boldsymbol{\phi}(s_i,a_i) - \gamma\boldsymbol{\phi}\big(s'_i,\pi(s'_i)\big)\Big)^T\right] \tag{2.45}$$

$$\widetilde{\mathbf{b}} = \frac{1}{m}\sum_{i=1}^{m}[\boldsymbol{\phi}(s_i,a_i)\; r'_i] \tag{2.46}$$

Factor $\frac{1}{m}$ can be dropped, since it appears on both sides of $\mathbf{A}\boldsymbol{w}^T = \mathbf{b}$. Given any arbitrary policy π and a large enough sample set, the above estimation scheme can be used to estimate the linear system and upon solution to provide the weights of the approximate Q_π-function of policy π. This is the Least-Squares Temporal Difference Learning for the Q-function (LSTDQ) algorithm, which is shown as Algorithm 8.

Algorithm 8 Least-Squares Temporal Difference Learning (LSTDQ) algorithm

```
Input: set of data tuples $D$, basis functions $\boldsymbol{\phi}$, discount factor $\gamma$, policy for evaluation $\pi$
$\widetilde{\mathbf{A}} = 0$
$\widetilde{\mathbf{b}} = 0$
for each $(s, a, r', s') \in D$
    $\widetilde{\mathbf{A}} = \widetilde{\mathbf{A}} + \boldsymbol{\phi}(s,a)\Big(\boldsymbol{\phi}(s,a) - \gamma\boldsymbol{\phi}\big(s', \pi(s')\big)\Big)^T$
    $\widetilde{\mathbf{b}} = \widetilde{\mathbf{b}} + \boldsymbol{\phi}(s,a)r'$
end
$\widetilde{\boldsymbol{w}} = \widetilde{\mathbf{A}}^{-1}\widetilde{\mathbf{b}}$
return $\widetilde{\boldsymbol{w}}$
```

Given that LSTDQ can be used to evaluate any policy using a single sample set, it is intuitive to use LSTDQ iteratively to evaluate progressively better policies obtained at each iteration through greedy policy improvement. It is important to note that there is no need to represent any such policy explicitly; only the weights of the approximate value function of the previous policy are sufficient to compute the greedy improved (next) policy on demand (at any given state). This is the main idea behind the Least-Squares Policy Iteration (LSPI) algorithm shown below as Algorithm 9.

Algorithm 9 Least-Squares Policy Iteration (LSPI) algorithm

```
Input: set of data tuples $D$, basis functions $\boldsymbol{\phi}$, discount factor $\gamma$, stopping criterion $\epsilon$
$\boldsymbol{w}' = 0$ // initialize parameters
repeat
    $\boldsymbol{w} = \boldsymbol{w}'$
    $\widetilde{\mathbf{A}} = 0$
    $\widetilde{\mathbf{b}} = 0$
    for each $(s, a, r', s') \in D$
        $a' = \arg\max_{a'' \in \mathcal{A}} \boldsymbol{\phi}(s', a'')^T \boldsymbol{w}$
        $\widetilde{\mathbf{A}} = \widetilde{\mathbf{A}} + \boldsymbol{\phi}(s,a)\Big(\boldsymbol{\phi}(s,a) - \gamma\boldsymbol{\phi}\big(s', a'\big)\Big)^T$
        $\widetilde{\mathbf{b}} = \widetilde{\mathbf{b}} + \boldsymbol{\phi}(s,a)r'$
    end
    $\boldsymbol{w}' = \widetilde{\mathbf{A}}^{-1}\widetilde{\mathbf{b}}$
until $\|\boldsymbol{w} - \boldsymbol{w}'\| < \epsilon$
return $\pi(s) = \arg\max_{a \in \mathcal{A}} \boldsymbol{\phi}^T(s,a)\boldsymbol{w}$
```

2.3.5 Fitted Q-iteration

Fitted Q-iteration is a model-free, approximate value iteration algorithm, where transition and reward functions are unknown. In such cases, only sets of samples in the form of tuples (s_i, a_i, r'_i, s'_i) are available. These samples may be collected from observed trajectories or from a simulator. For the Q-function approximation, any regression algorithm, parametric or non-parametric, may be used. The fitted Q-iteration algorithm was introduced by (Ernst et al., 2005). In this example, we are going to use least-squares regression to fit the estimated Q-values. Least-squares regression is a parametric method, and we assume that basis functions $\boldsymbol{\phi}$ are given or chosen. Approximated Q-function is expressed as $\widehat{Q} = \boldsymbol{\phi}^T(s,a)\boldsymbol{w}$. Each fitted Q-iteration uses a sample set $\{(s_i, a_i, r'_i, s'_i)\}_{i=1}^m$ to estimate the improved Q-value q_i for state s_i, action a_i and current parameters $\boldsymbol{w}$. Then, it uses all (s_i, a_i, q_i) tuples to train the least-squares regressor, that will be used in the next iteration. The complete fitted Q-iteration algorithm is shown as Algorithm 10 below.

Algorithm 10 Fitted Q-Iteration (FQI) algorithm

```
1: Input: basis functions $\phi$, discount factor $\gamma$, stopping criterion $\epsilon$, number
   of samples $m$
2: $\boldsymbol{w}' = 0$ // initialize parameters
3: repeat
4:    $\boldsymbol{w} = \boldsymbol{w}'$
5:    Get samples $\{(s_i, a_i, r'_i, s'_i)\}_{i=1}^m$ from the simulator
6:    for $i = 1$ to $m$
7:       $q_i = r'_i + \gamma \max_{a' \in \mathcal{A}} \left[\boldsymbol{\phi}^T(s'_i, a')\boldsymbol{w}\right]$
8:    end
9:    $\boldsymbol{w}' = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{m} \left(q_i - \boldsymbol{\phi}^T(s_i, a_i)\boldsymbol{w}\right)^2$
10: until $\|\boldsymbol{w} - \boldsymbol{w}'\| < \epsilon$
11: return $\pi(s) = \arg\max_{a \in \mathcal{A}} \boldsymbol{\phi}^T(s,a)\boldsymbol{w}$
```

2.4 Supervised Learning

Supervised learning is a learning process supervised by a teacher. Training data come in tuples or examples in the form $(\boldsymbol{x}, t)$, where $\boldsymbol{x} \in \mathcal{X}$ (i.e., $\mathbb{R}^N$) is the input pattern, and $t \in \mathbb{R}$ is the observed target value or label given by the teacher. A model is trained using a training data set $D_{train} = \{(\boldsymbol{x}_i, t_i)\}_{i=1}^m$ and an appropriate training algorithm and rules/parameters are extracted. Then, the trained model is tested for accuracy using a different set of unseen examples $D_{test} = \{(\boldsymbol{x}_i, t_i)\}_{i=1}^k$.

Classification is the identification of the category, class, or label the input pattern $\boldsymbol{x}$ belongs to. Binary classification is the most common form and target variable t takes two values ($t \in \{-1, +1\}$). Regression is the estimation of a real value for input pattern $\boldsymbol{x}$, and the target variable is real ($t \in \mathbb{R}$). Predicted values $y(\boldsymbol{x})$ are typically approximated as a linear sum of (typically, nonlinear) basis functions $\boldsymbol{\phi}(\boldsymbol{x})$:

$$\begin{aligned} y(\boldsymbol{x}) &= w_0 + w_1\phi_1(\boldsymbol{x}) + \cdots + w_1\phi_n(\boldsymbol{x}) \\ &= \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) + w_0 \end{aligned} \tag{2.47}$$

There are n basis functions $\phi_i(\boldsymbol{x})$, $n+1$ parameters w_i, and w_0 is the called the bias term. Sometimes, an additional basis function $\phi_0(\boldsymbol{x}) = 1$ is defined to cover the bias in a uniform way and $y(\boldsymbol{x})$ becomes

$$\begin{aligned} y(\boldsymbol{x}) &= w_0\phi_0(\boldsymbol{x}) + w_1\phi_1(\boldsymbol{x}) + \cdots + w_1\phi_n(\boldsymbol{x}) \\ &= \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) \end{aligned} \tag{2.48}$$

where $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)^T$ and $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_n)^T$.

Loss function

Loss function $\mathcal{L}(\boldsymbol{x}, t, y(\boldsymbol{x}))$ is a map $\mathcal{L} : \mathcal{X} \times \mathbb{R} \times \mathbb{R} \to [0, \infty)$, where $\boldsymbol{x}$ is the input pattern, t is the observed value, and $y(\boldsymbol{x})$ is the prediction function, and has the property of $\mathcal{L}(\boldsymbol{x}, t, t) = 0$ for all $\boldsymbol{x} \in \mathcal{X}$ and $t \in R$. It measures the discrepancy between the observed value t and the predicted value $y(\boldsymbol{x})$. The exact definition depends on the learning goal. In the binary classification case, $t \in \{-1, +1\}$, a common loss function simply counts the misclassified examples $\mathcal{L}(\boldsymbol{x}, t, y(\boldsymbol{x})) = \frac{1}{2}|t - y(\boldsymbol{x})|$. For regression problems, we may measure the residual $|t - y(\boldsymbol{x})|$ or the square of it $(t - y(\boldsymbol{x}))^2$.

Expected risk

Let's now define the expected risk. We assume that our examples $(\boldsymbol{x}, t)$ are drawn out of a $P(\boldsymbol{x}, t)$ probability distribution, and are independent and identically distributed. The risk is defined as the expected loss over all possible training patterns.

$$R[y] = \mathbf{E}[\mathcal{L}(\boldsymbol{x}, t, y(x))] = \int_{\mathcal{X}\times\mathbb{R}} \mathcal{L}(\boldsymbol{x}, t, y(x))dP(\boldsymbol{x}, t) \tag{2.49}$$

However, the computation of the expected risk is an intractable problem, since we don't know $P(\boldsymbol{x}, t)$ explicitly. We only have the training examples and therefore we can replace the unknown distribution $P(\boldsymbol{x}, t)$ by its empirical estimate. Now we are ready to define the empirical risk.

Empirical risk

$$R_{emp}[y] = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\boldsymbol{x}_i, t_i, y(\boldsymbol{x}_i)) \tag{2.50}$$

We can minimize the empirical risk function by fitting the predictor $y(\boldsymbol{x})$ to the specific training examples. This is called overfitting and leads to bad generalization performance and numerical instabilities. One technique to overcome this issue is regularization. We add a stabilization term $\Omega[y]$ to the empirical risk to penalize the complexity of the model, the prediction function $y(\boldsymbol{x})$.

Regularization

$$R_{reg}[y] = R_{emp}[y] + \lambda\Omega[y] \tag{2.51}$$

Minimizing $R_{reg}[y]$, the regularization parameter $\lambda > 0$ specifies the tradeoff between minimization of $R_{emp}[y]$ and the smoothness or simplicity of $y(\boldsymbol{x})$ which is enforced by small $\Omega[y]$. $\Omega[y]$ may be chosen to be convex, with one easy to find, global minimum. A common choice for the stabilization term is $\Omega[y] = \frac{1}{2}\|\boldsymbol{w}\|^2$.

2.4.1 Kernel-based Supervised Learning algorithms

Non-parametric methods do not use explicit parameters, but keep the training data points or a subset of them and use them during the prediction phase. Kernel-based methods, a case of non-parametric methods, use the inner product of the feature space mapping $\boldsymbol{\phi}(\boldsymbol{x})$:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}') \tag{2.52}$$

Kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ is a symmetric positive semidefinite function, which measures the similarity of $\boldsymbol{x}$ and $\boldsymbol{x}'$. The kernel concept first was introduced into the machine learning field by Aizerman et al. (1964) and in the context of large margin classifiers by Boser et al. (1992). Kernels are used in algorithms in place of the inner product $\boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}')$, thus there is no need to know the exact (and, possibly, high-dimensional or even infinite-dimensional) feature space mapping $\boldsymbol{\phi}(\boldsymbol{x}_i)$. This is called *kernel trick* or *kernel substitution.*

Kernel trick

The simplest kernel is the linear one, $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T\boldsymbol{x}'$, where $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{x}$. The homogeneous polynomial kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}) = (\boldsymbol{x}^T\boldsymbol{x})^d$, $d \in \mathbb{N}$, is widely used. Another commonly used kernel is $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right)$, where $\sigma > 0$, the Gaussian kernel, suggested by Boser et al. (1992); Guyon et al. (1993); Vapnik (1995). The Sigmoid kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\alpha\boldsymbol{x}^T\boldsymbol{x}' + \beta)$, where $\alpha > 0$ and $\beta < 0$, despite not being positive semidefinite, has been successfully used in practice (Vapnik, 1995). There are certain rules for combining basic kernels to construct composite ones. The inhomogeneous polynomial kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}) = (\boldsymbol{x}^T\boldsymbol{x} + c)^d$, ($d \in \mathbb{N}$, $c \geqslant 0$), is such an example.

2.4.2 Support Vector Machines for Classification (SVM)

Here, we will discuss a two-class or binary classification algorithm. There is a training set of m tuples, $\{(\boldsymbol{x}_i, t_i)\}_{i=1}^m$, where $\boldsymbol{x}_i \in \mathcal{X}$ is the input vector and target value $t_i \in \{-1, +1\}$ the observed class or label for data sample i. Our prediction is based on a linear model of the form

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) + b \tag{2.53}$$

Let us now study the toy problem shown in Figure 2.5. There are two separable classes of data, the circles and the squares, and three hyperplanes in a two-dimensional space. The first hyperplane is in the middle $\boldsymbol{w}^T\boldsymbol{x} + b = 0$, the decision border, a sample that comes to the left of it is classified as a circle, and a sample that resides to the right is classified as a square. Our decision function comes in the form[2] $f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$. The question is how to optimize $\boldsymbol{w}$ to get the "best" hyperplane, the one that behaves well on unseen examples and tolerates noise comfortably. The idea is to maximize the distance between the left $(\boldsymbol{w}^T\boldsymbol{x} + b = -1)$ and the right $(\boldsymbol{w}^T\boldsymbol{x} + b = +1)$ hyperplanes.

Let $\boldsymbol{x}_1$ be a point on right hyperplane and $\boldsymbol{x}_2$ a point on the left one.

$$\left.\begin{array}{ll} \boldsymbol{w}^T\boldsymbol{x}_1 + b & = +1 \\ \boldsymbol{w}^T\boldsymbol{x}_2 + b & = -1 \end{array}\right\} \Rightarrow \boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 2$$

$$\underbrace{\frac{\boldsymbol{w}^T}{\|\boldsymbol{w}\|}(\boldsymbol{x}_1 - \boldsymbol{x}_2)}_{\text{margin length}} = \frac{2}{\|\boldsymbol{w}\|} \tag{2.54}$$

where $\dfrac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$ is the unit vector at $\boldsymbol{w}$ direction and $\dfrac{2}{\|\boldsymbol{w}\|}$ the length of the margin. Our goal is to maximize the margin and keep the circles on the left and squares on the right. This can be formulated as quadratic optimization:

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{2.55}$$

$$\text{subject to} \quad t_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geqslant 1, \quad \text{for all } i = 1 \mathinner{..} m \tag{2.56}$$

The optimization objective (2.55) is equivalent to maximizing $\dfrac{2}{\|\boldsymbol{w}\|}$ (the margin). The constraints (2.56), for $t_i = +1$ resolve to $\boldsymbol{w}^T\boldsymbol{x}_i + b \geqslant +1$ and for $t_i = -1$ resolve to $\boldsymbol{w}^T\boldsymbol{x}_i + b \leqslant -1$. The examples that reside on and define the hyperplanes are called *support vectors*, i.e. $t_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$.

Support vectors

Let us now leave the convenient two-dimensional input space and map the input vector $\boldsymbol{x}$ to feature vector $\boldsymbol{\phi}(\boldsymbol{x})$. There is no need to know the exact $\boldsymbol{\phi}(\cdot)$ transformation. All we need to know about the feature space $\mathcal{H}$ is that the inner product is defined, and it is given by the kernel function $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}')$ we picked. Now, the optimization goal becomes:

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 \tag{2.57}$$

$$\text{subject to} \quad t_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geqslant 1, \quad \text{for all } i = 1 \mathinner{..} m \tag{2.58}$$

[2] sign$(\cdot)$ function returns values $\{-1, +1\}$.

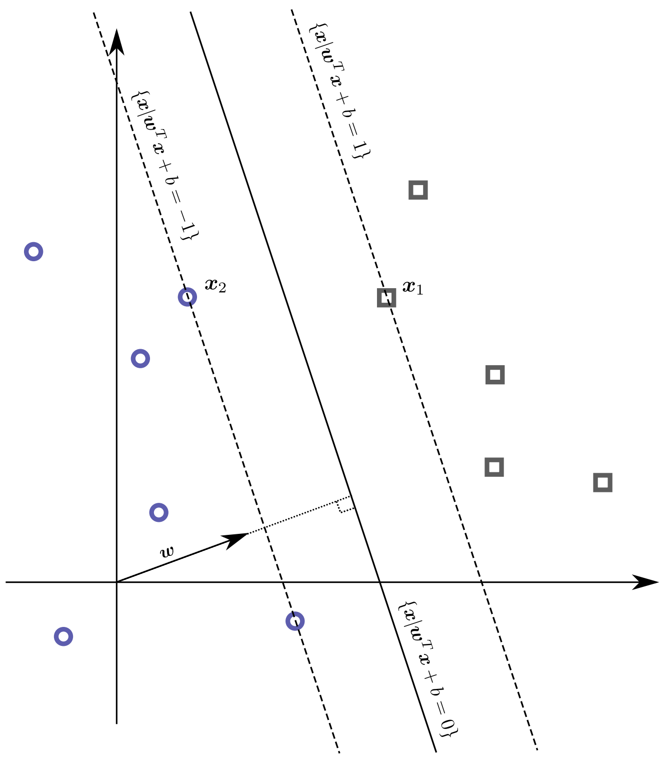

$\{\boldsymbol{x}|\boldsymbol{w}^T\boldsymbol{x}+b=1\}$
$\{\boldsymbol{x}|\boldsymbol{w}^T\boldsymbol{x}+b=-1\}$
$\boldsymbol{x}_2$
$\boldsymbol{x}_1$
$\boldsymbol{w}$
$\{\boldsymbol{x}|\boldsymbol{w}^T\boldsymbol{x}+b=0\}$

FIGURE 2.5: Maximum margin classifiers. The middle hyperplane is the decision boundary. The left and right hyperplanes are defining the margin. Margin is maximized with no examples in it. The examples that lay on the left and right hyperplanes are the support vectors

The first line (2.57) is the objective function and the second one (2.58) the inequality constrains. We introduce the *Lagrangian function* (Nocedal and Wright, 2006):

Lagrangian

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|w\|^2 - \sum_{t=1}^{m} \alpha_i (t_i(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i) + b) - 1) \tag{2.59}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$, $\boldsymbol{\alpha} \geqslant 0$ are the *Lagrangian multipliers* or *dual variables*. The goal now becomes: minimize L with respect to the primal variables $\boldsymbol{w}$ and b and maximize with respect to the dual variables $\boldsymbol{\alpha}$. The solution is a saddle point and the partial derivatives of L with respect to the primal variables $\boldsymbol{w}$ and b must vanish.

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} = 0 \qquad \Rightarrow \qquad \sum_{t=1}^{m} \alpha_i t_i = 0 \tag{2.60}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = 0 \qquad \Rightarrow \qquad \boldsymbol{w} = \sum_{i=1}^{m} \alpha_i t_i \boldsymbol{\phi}(\boldsymbol{x}_i) \tag{2.61}$$

By substituting (2.60) and (2.61) into the Lagrangian (2.59), we eliminate the primal variables $\boldsymbol{w}$ and b and the Lagrangian becomes:

$$\widehat{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j t_i t_j \boldsymbol{\phi}^T(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j)$$

By replacing the inner product $\boldsymbol{\phi}^T(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j)$ with the kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, that is the kernel trick, we get the dual optimization problem:

$$\text{minimize} \quad \widehat{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j t_i t_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.62}$$

$$\text{subject to} \quad \alpha_i \geqslant 0, \quad \text{for all } i = 1 \mathinner{..} m \tag{2.63}$$

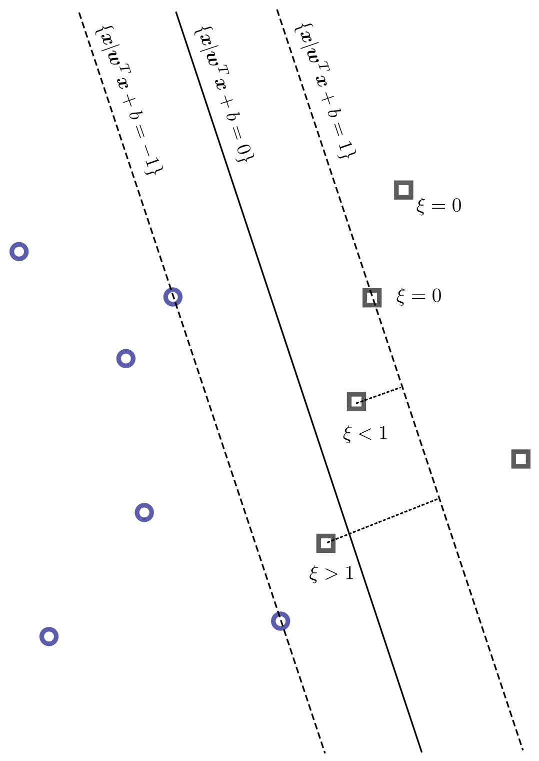
$$\sum_{t=1}^{m} \alpha_i t_i = 0 \tag{2.64}$$

The solution gives the $\boldsymbol{\alpha}$ coefficients. The α_i's that correspond to examples that lay on the left or on the right hyperplanes are non-zero. These examples are the support vectors; the remaining α_i coefficients are zero.

This optimization problem has a solution on the condition that examples are separable in the feature space, which is set by choosing the kernel function. However, in practice class conditional distributions overlap, noise corrupts data, and absolute separation of training data leads to poor generalization.

Soft margin hyperplanes

Let's now relax the above algorithm to allow some training examples to be misclassified. We introduce slack variables $\xi_i \geqslant 0$, one for each training example (Bennett and Mangasarian, 1992; Cortes and Vapnik, 1995). The examples on the margin borders and outside of it have $\xi_i = 0$, and the ones inside the margin yield $\xi_i = |t_i - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) - b|$. If an example lies on the separating border, it will have $\xi = 1$; examples with $\xi < 1$ are correctly classified and examples with $\xi > 1$ are misclassified (Figure 2.6).



$\{\boldsymbol{x}|\boldsymbol{w}^T\boldsymbol{x}+b=-1\}$
$\{\boldsymbol{x}|\boldsymbol{w}^T\boldsymbol{x}+b=0\}$
$\{\boldsymbol{x}|\boldsymbol{w}^T\boldsymbol{x}+b=1\}$
$\xi=0$
$\xi=0$
$\xi<1$
$\xi>1$

FIGURE 2.6: Soft margin hyperplanes. Two square class samples are in the margin. The one with $\xi < 1$ is correctly classified, the other one with $\xi > 1$ is misclassified. all points on or outside margin borders have $\xi = 0$

Thus, the primary optimization problem now becomes:

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \tag{2.65}$$

$$\text{subject to} \quad t_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i)+b) \geqslant 1-\xi_i, \qquad \text{for all } i = 1\,..\,m \tag{2.66}$$

$$\xi_i \geqslant 0 \tag{2.67}$$

where $C > 0$, is the trade off between the slack variables penalty and maximization of the margin. At the limit $C \to \infty$ slack variables vanish, and we get the earlier support vector machines for separable data.

The Lagrangian now becomes:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i - \sum_{t=1}^{m}\alpha_i(t_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i)+b)-1+\xi_i) - \sum_{i=1}^{m}\mu_i\xi_i \tag{2.68}$$

where α_i and μ_i are the Lagrange multipliers. The *Karush-Kuhn-Tucker*

(KKT) conditions (Nocedal and Wright, 2006) that must be satisfied are:

$$\alpha_i \geqslant 0 \tag{2.69}$$

$$t_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) - 1 + \xi_i \geqslant 0 \tag{2.70}$$

$$\alpha_i(t_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) - 1 + \xi_i) = 0 \tag{2.71}$$

$$\mu_i \geqslant 0 \tag{2.72}$$

$$\xi_i \geqslant 0 \tag{2.73}$$

$$\mu_i\xi_i = 0, \qquad \text{for all } i = 1 \mathinner{..} m \tag{2.74}$$

The partial derivatives of L with respect to the primal variables $\boldsymbol{w}$, b and ξ_i should vanish.

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = 0 \qquad \Rightarrow \qquad \boldsymbol{w} = \sum_{i=1}^{m} \alpha_i t_i \boldsymbol{\phi}(\boldsymbol{x}_i) \tag{2.75}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} = 0 \qquad \Rightarrow \qquad \sum_{t=1}^{m} \alpha_i t_i = 0 \tag{2.76}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \xi_i} = 0 \qquad \Rightarrow \qquad \alpha_i = C - \mu_i \tag{2.77}$$

We eliminate $\boldsymbol{w}$, b, and ξ_i in the Lagrangian above and we get the dual optimization problem:

$$\text{minimize} \quad \widehat{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j t_i t_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.78}$$

$$\text{subject to} \quad 0 \leqslant \alpha_i \leqslant C, \qquad \text{for all } i = 1 \mathinner{..} m \tag{2.79}$$

$$\sum_{t=1}^{m} \alpha_i t_i = 0 \tag{2.80}$$

This is a quadratic optimization problem. If we substitute (2.75) into the decision function $y(\boldsymbol{x})$, we get:

$$y(\boldsymbol{x}) = \operatorname{sign}\left(w^T\boldsymbol{\phi}(\boldsymbol{x}) + b\right) \tag{2.81}$$

$$= \operatorname{sign}\left(\sum_{i=1}^{m} \alpha_i t_i \boldsymbol{\phi}^T(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}) + b\right) \tag{2.82}$$

$$= \operatorname{sign}\left(\sum_{i=1}^{m} \alpha_i t_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}) + b\right) \tag{2.83}$$

However, only a few α_i coefficients that correspond to support vectors are non-zero. Let's say that $\mathcal{M}$ is the set of indexes in the training set that correspond to support vectors. The final form of the decision function is:

$$y(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i \in \mathcal{M}} \alpha_i t_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}) + b\right)$$

Now, only the bias term b needs to be estimated. Since slack variable ξ_i for support vectors vanish, we have:

$$t_i \left(\sum_{j \in \mathcal{M}} \alpha_i t_i \kappa(\boldsymbol{x}_j, \boldsymbol{x}_i) + b\right) = 1$$

For a stable numerical estimation we obtain b by averaging over all support vectors:

$$b = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \left(t_i - \sum_{j \in \mathcal{M}} \alpha_j t_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)\right) \tag{2.84}$$

2.4.3 Support Vector Machines for Regression (SVR)

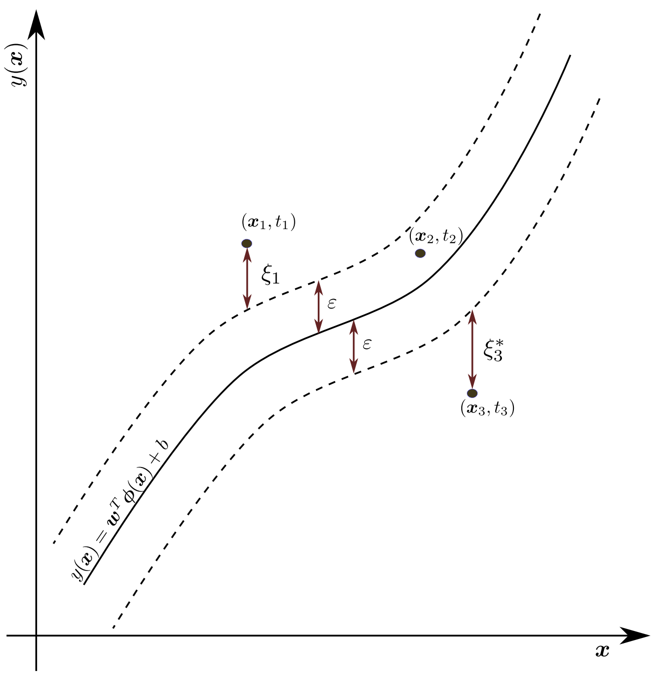
The concept of support vectors and large margin is extended to regression (Vapnik, 1995; Drucker et al., 1997). The margin here takes the form of an ε-insensitive tube (Figure 2.7). The solid line is the prediction and the doted lines are the borders of the ε-tube. Any examples inside the ε-tube are considered error free. Examples $\boldsymbol{x}_i$ outside the ε-tube have an error of ξ_i or ξ_i^* value. We have a training set of m examples, $\{(\boldsymbol{x}_i, t_i)\}_{i=1}^m$, where $\boldsymbol{x}_i \in \mathcal{X}$ is the input vector and $t_i \in \mathbb{R}$ is the target value. Our algorithm is based on a linear model of the form:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b \tag{2.85}$$

The ε-insensitive error function is given by:

$$E_\varepsilon(\boldsymbol{x}, t) = \max(0, |t - y(\boldsymbol{x})| - \varepsilon) \tag{2.86}$$

For each point i outside the tube, there is either a ξ_i or a ξ_i^* non-zero error. For these points above the ε-tube, the second is zero, for the points below the



$y(\boldsymbol{x})$
$(\boldsymbol{x}_1, t_1)$
ξ_1
$(\boldsymbol{x}_2, t_2)$
ε
ε
ξ_3^*
$(\boldsymbol{x}_3, t_3)$
$y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) + b$
$\boldsymbol{x}$

FIGURE 2.7: The solid line is the predicted values. Dotted lines form the ε-tube. Examples inside the ε-tube are considered error free. For those examples above the ε-tube an error value of ξ_i is defined, and for those examples below the ε-tube there is an error of ξ_i^* magnitude

ε-tube the first one is zero. Inside the ε-tube both are zero. The total error becomes $E_\varepsilon(\boldsymbol{x}_i, t_i) = \xi_i + \xi_i^*$ and the primal optimization problem is formulated as:

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \tag{2.87}$$

$$\text{subject to} \quad t_i - (\boldsymbol{w}^T\boldsymbol{\phi}_i + b) \leqslant \varepsilon + \xi_i, \quad \text{for all } i = 1 \mathinner{..} m \tag{2.88}$$

$$(\boldsymbol{w}^T\boldsymbol{\phi}_i + b) - t_i \leqslant \varepsilon + \xi_i^* \tag{2.89}$$

$$\xi_i \geqslant 0 \tag{2.90}$$

$$\xi_i^* \geqslant 0 \tag{2.91}$$

We define a Lagrangian:

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = &\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) - \sum_{i=1}^{m}(\mu_i\xi_i + \mu_i^*\xi_i^*) \\ &- \sum_{i=1}^{m}\alpha_i(\varepsilon + \xi_i + (\boldsymbol{w}^T\boldsymbol{\phi}_i + b) - t_i) \\ &- \sum_{i=1}^{m}\alpha_i^*(\varepsilon + \xi_i^* - (\boldsymbol{w}^T\boldsymbol{\phi}_i + b) + t_i) \end{aligned} \tag{2.92}$$

where $\alpha_i, \alpha_i^*, \mu_i, \mu_i^* \geqslant 0$ are the Lagrange multipliers. The Karush-Kuhn-Tucker (KKT) conditions (Nocedal and Wright, 2006) that must be satisfied are:

$$\alpha_i(\varepsilon + \xi_i + (\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) - t_i) = 0, \quad \text{for all } i = 1 \mathinner{..} m \tag{2.93}$$

$$\alpha_i^*(\varepsilon + \xi_i^* + (\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) - t_i) = 0 \tag{2.94}$$

$$(C - a_i)\xi_i = 0 \tag{2.95}$$

$$(C - a_i^*)\xi_i^* = 0 \tag{2.96}$$

The partial derivatives of L with respect to the primal variables $\boldsymbol{w}$, b, ξ_i and ξ_i^* should vanish, therefore we have:

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)}{\partial \boldsymbol{w}} = 0 \quad \Rightarrow \quad \boldsymbol{w} = \sum_{i=1}^{m}(\alpha_i + \alpha_i^*)\boldsymbol{\phi}(\boldsymbol{x}_i) \tag{2.97}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)}{\partial b} = 0 \quad \Rightarrow \quad \sum_{t=1}^{m}(\alpha_i + \alpha_i^*) = 0 \tag{2.98}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)}{\partial \xi_i} = 0 \quad \Rightarrow \quad \alpha_i + \mu_i = C \tag{2.99}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*)}{\partial \xi_i^*} = 0 \quad \Rightarrow \quad \alpha_i^* + \mu_i^* = C \tag{2.100}$$

Using the above equations and the kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in place of $\boldsymbol{\phi}^T(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j)$, we formulate the Lagrangian (2.92) with respect to α_i α_i^*:

$$\text{maximize} \quad \widehat{L}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$- \varepsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)t_i \tag{2.101}$$

$$\text{subject to} \quad 0 \leqslant \alpha_i \leqslant C, \qquad \text{for all } i = 1 \mathinner{..} m \tag{2.102}$$

$$0 \leqslant \alpha_i^* \leqslant C \tag{2.103}$$

$$\sum_{t=1}^{m}(\alpha_i - \alpha_i^*) = 0 \tag{2.104}$$

From the KKT conditions (2.93) we note that α_i coefficients can be non-zero only on the upper ε-tube border or above the ε-tube, and from (2.94) α_i^*

coefficients can be non-zero only on the lower ε-tube border or below the ε-tube. In all other cases, α_i and α_i^* are zero. Support vectors are the examples $(\boldsymbol{x}_i, t_i)$ that lay on the ε-tube border or outside of it, where either α_i or α_i^* is non-zero.

Let $\mathcal{M}$ be the set of indexes of support vectors in the training set. By substituting (2.97) into (2.85), we get the final kernel-based prediction function:

$$y(\boldsymbol{x}) = \sum_{i\in\mathcal{M}} (\alpha_i - \alpha_i^*)\kappa(\boldsymbol{x}, \boldsymbol{x}_i) + b \tag{2.105}$$

Again, here we have to estimate the bias b from (2.93). We choose a point for which $0 < \alpha_i < C$, which means from (2.95) that $\xi_i = 0$, and from (2.93) that the term $\varepsilon + \xi_i + (\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) - t_i = 0$.

$$\begin{aligned} b &= t_i - \varepsilon - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) \\ &= t_i - \varepsilon - \sum_{i\in\mathcal{M}} (\alpha_i - \alpha_i^*)\kappa(\boldsymbol{x}, \boldsymbol{x}_i) \end{aligned} \tag{2.106}$$

An analogous result can be obtained by considering the $0 < \alpha_i^* < C$ points and equation (2.94). A numerically stable b is given by averaging all $0 < \alpha_i, \alpha_i^* < C$ cases.

Support vector machines have been used in various classification and regression applications with excellent generalization properties and sparse kernel representation. However, there are some drawbacks. The most outstanding one is the growth of the number of the support vectors with the size of the training set. Also, there is the C parameter, and the ε parameter in the case of regression, to be set. Furthermore, predictions are not probabilistic. And finally, the kernel function must be positive definite.

2.4.4 Relevance Vector Machines (RVM) for Regression

The relevance vector machine is a probabilistic sparse kernel linear model that has a prediction function similar to that of the SVM. We have a training set of m examples, $\{(\boldsymbol{x}_i, t_i)\}_{i=1}^m$ where $\boldsymbol{x}_i \in \mathcal{X}$ is the input vector and target value $t_i \in \mathbb{R}$. Our prediction function has the form:

$$y(\boldsymbol{x}) = \sum_{i=1}^{m} w_i \kappa(\boldsymbol{x}, \boldsymbol{x}_i) + w_0, \tag{2.107}$$

where $\kappa(\boldsymbol{x}, \boldsymbol{x}_i)$ is the kernel function. By setting:

$$\boldsymbol{\phi}(\boldsymbol{x}_i) = [1, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_1)], \kappa(\boldsymbol{x}_i, \boldsymbol{x}_2), \ldots, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_m)]^T$$

the above prediction function (2.107) is rewritten as:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{2.108}$$

We define $\boldsymbol{\Phi} \in \mathbb{R}^{m \times (m+1)}$, the *design matrix*, as:

$$\boldsymbol{\Phi} = [\boldsymbol{\phi}^T(\boldsymbol{x}_1), \boldsymbol{\phi}^T(\boldsymbol{x}_2), \ldots, \boldsymbol{\phi}^T(\boldsymbol{x}_m)]^T$$

We assume that targets t_i are samples of the above model (2.108) with the addition of noise ϵ_i:

$$t_i = y(\boldsymbol{x}_i) + \epsilon_i, \tag{2.109}$$

where ϵ_i are independent and identically distributed samples drawn from a zero-mean Gaussian distribution with variance σ^2. The conditional distribution for the target variable t_i given the input vector $\boldsymbol{x}_i$ takes the form:

$$p(t_i|\boldsymbol{x}_i, \boldsymbol{w}, \sigma^2) = \mathcal{N}\left(t_i|y(\boldsymbol{x}_i), \sigma^2\right), \tag{2.110}$$

a Gaussian distribution over t_i with mean $y(\boldsymbol{x}_i)$ and variance σ^2. Based on the assumption that targets t_i are independent, the likelihood of the data is:

$$\begin{aligned} p(\boldsymbol{t}|\mathbf{X}, \boldsymbol{w}, \sigma^2) &= \prod_{i=1}^{m} p(t_i|\boldsymbol{x}_i, \boldsymbol{w}, \sigma^2) \\ &= (2\pi\sigma^2)^{-m/2} \exp\left\{-\frac{1}{2\sigma^2}\|\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}\|^2\right\} \end{aligned} \tag{2.111}$$

where $\mathbf{X}$ is a matrix whose i^{th} row is $\boldsymbol{x}_i^T$, and $\boldsymbol{t}$ is the vector $\{t_i\}_{i=1}^m$. For a less cluttered notation, we omit implicit conditioning on input vectors $\boldsymbol{x}_i$ and $\mathbf{X}$, e.g. the $p(\boldsymbol{t}|\mathbf{X}, \boldsymbol{w}, \sigma^2)$ will be noted as $p(\boldsymbol{t}|\boldsymbol{w}, \sigma^2)$.

Direct maximum likelihood estimation of $\boldsymbol{w}$ and σ^2 from the above equation (2.111) will lead to overfitting. We need to impose a complexity penalty on the parameters, like the margin in the SVM paradigm. We implement our preference for smoother functions by setting a zero mean Gaussian prior distribution over parameter $\boldsymbol{w}$:

$$p(\boldsymbol{w}|\boldsymbol{\alpha}) = \prod_{i=0}^{m} \mathcal{N}(w_i|0, \alpha_i^{-1}) \tag{2.112}$$

where $\boldsymbol{\alpha}$ is a vector of $m+1$ hyperparameters, one for every weight. The use of an individual hyperparameter a_i for every weight w_i is the key for the sparsity of RVMs. The hyperparameters $\boldsymbol{\alpha}$ and noise precision $\beta = \sigma^{-2}$ are given by Gamma priors (Berger (1985)):

$$p(\boldsymbol{\alpha}) = \prod_{i=0}^{m} \mathrm{Gamma}(\alpha_i|a, b), \tag{2.113}$$

$$p(\beta) = \mathrm{Gamma}(\beta|c, d), \tag{2.114}$$

where:

$$\mathrm{Gamma}(\alpha|a, b) = \frac{b^a \alpha^{a-1} e^{-b\alpha}}{\Gamma(a)}, \qquad \text{Gamma distribution}$$

$$\Gamma(a) = \int_0^{\infty} t^{a-1} e^{-t} dt, \qquad \text{Gamma function}$$

We make these priors non-informative, parameters a, b, c, d are set to zeros ($a = b = c = d = 0$) (Tipping, 2001). This setting leads to uniform hyperpriors. This setting is called Automatic Relevance Determination or ARD in the context of neural networks (MacKay, 1994; Neal, 1996), and it leads to sparse representations.

The predictive distribution takes the form: Inference

$$p(t_*|\boldsymbol{t}) = \int p(t_*|\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2|\boldsymbol{t})\, d\boldsymbol{w}\, d\boldsymbol{\alpha}\, d\sigma^2 \tag{2.115}$$

where t_* is prediction for the sample $\boldsymbol{x}_*$. We need the posterior over all unknowns:

$$p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2|\boldsymbol{t}) = \frac{p(\boldsymbol{t}|\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2)}{p(\boldsymbol{t})} \tag{2.116}$$

However, we cannot compute the normalizer $p(\boldsymbol{t})$ in the denominator. Alternatively, by decomposing the posterior:

$$p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2|\boldsymbol{t}) = p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\alpha}, \sigma^2)p(\boldsymbol{\alpha}, \sigma^2|\boldsymbol{t}) \tag{2.117}$$

equation (2.115) becomes:

$$p(t_*|\boldsymbol{t}) = \int p(t_*|\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2)p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\alpha}, \sigma^2)p(\boldsymbol{\alpha}, \sigma^2|\boldsymbol{t})\, d\boldsymbol{w}\, d\boldsymbol{\alpha}\, d\sigma^2 \tag{2.118}$$

The posterior distribution for the weights is Gaussian and is given by:

$$\begin{aligned} p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\alpha}, \sigma^2) &= \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= (2\pi)^{-\frac{m+1}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu})^T|\boldsymbol{\Sigma}|^{-1}(\boldsymbol{w}-\boldsymbol{\mu})\right\} \end{aligned} \tag{2.119}$$

where the posterior covariance and mean are given by:

$$\boldsymbol{\Sigma} = (\sigma^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \mathbf{A})^{-1} \tag{2.120}$$

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{t} \tag{2.121}$$

where $\mathbf{A} = \operatorname{diag}(\boldsymbol{\alpha})$. The second term of the right hand side in equation (2.117), the hyperparameter posterior $p(\boldsymbol{\alpha}, \sigma^2|\boldsymbol{t})$, cannot be computed analytically. A workaround is to approximate it with the delta function at its mode, i.e. the most probable values $\boldsymbol{\alpha}_{MP}$ and ${\sigma^2}_{MP}$:

$$p(\boldsymbol{\alpha}, \sigma^2|\boldsymbol{t}) \approx \delta(\boldsymbol{\alpha}_{MP}, {\sigma^2}_{MP}) \tag{2.122}$$

Looking for the hyperparameter posterior mode, we can maximize the hyperparameter posterior $p(\boldsymbol{\alpha}, \sigma^2|\boldsymbol{t}) \propto p(\boldsymbol{t}|\boldsymbol{\alpha}, \sigma^2)p(\boldsymbol{\alpha})p(\sigma^2)$ with respect to $\boldsymbol{\alpha}$, σ^2. We assume uniform hyperpriors and we ignore $p(\boldsymbol{\alpha})$ and $p(\sigma^2)$. We only need to maximize the marginal likelihood[3] $p(\boldsymbol{t}|\boldsymbol{\alpha}, \sigma^2)$:

$$\begin{aligned} p(\boldsymbol{t}|\boldsymbol{\alpha}, \sigma^2) &= \int p(\boldsymbol{t}|\boldsymbol{w}, \sigma^2)p(\boldsymbol{w}|\boldsymbol{\alpha})\, d\boldsymbol{w} \\ &= (2\pi)^{-m/2}|\mathbf{C}|^{-1/2} \exp\left\{-\frac{1}{2}\boldsymbol{t}^T\mathbf{C}^{-1}\boldsymbol{t}\right\} \end{aligned} \tag{2.123}$$

[3] Marginal likelihood is also known as *evidence for the hyperparameters*, (MacKay, 1992).

where $\mathbf{C} = \sigma^2\mathbf{I} + \mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^T$. This type of maximization is also known known as *type-II maximum likelihood* method (Berger, 1985).

We cannot get analytically the values $\boldsymbol{\alpha}$ and σ^2 that maximize equation (2.123). The log marginal likelihood takes the form:

$$\ln p(\boldsymbol{t}|\boldsymbol{\alpha}, \sigma^2) = -\frac{1}{2}\left\{m\ln(2\pi) + \ln|\mathbf{C}| + \boldsymbol{t}^T\mathbf{C}^{-1}\boldsymbol{t}\right\} \tag{2.124}$$

Differentiating equation (2.124) with respect to $\boldsymbol{\alpha}$ and equating to zero we get:

$$\alpha_i = \frac{\gamma_i}{\mu_i^2} \tag{2.125}$$

where μ_i is the i-th component of the posterior mean (2.121), and the quantity γ_i is defined as:

$$\gamma_i = 1 - \alpha_i\Sigma_{ii} \tag{2.126}$$

Σ_{ii} is the i-th diagonal component of the posterior covariance (2.120). Differentiation of (2.124) with respect to the noise variance σ^2 and setting to zero gives:

$$\sigma^2 = \frac{\|\boldsymbol{t} - \mathbf{\Phi}\boldsymbol{\mu}\|^2}{m - \Sigma_i\gamma_i} \tag{2.127}$$

Assuming that the kernel function $\kappa(\cdot,\cdot)$ is chosen and the training set is given, estimation of $\boldsymbol{\alpha}_{MP}$ and ${\sigma^2}_{MP}$ is done iteratively (Tipping, 2001; Tzikas et al., 2006):

1. create the design matrix $\mathbf{\Phi}$.
2. choose initial values for $\boldsymbol{\alpha}$, and σ^2.
3. calculate $\mathbf{\Sigma} = (\sigma^{-2}\mathbf{\Phi}^T\mathbf{\Phi} + \mathbf{A})^{-1}$ and $\boldsymbol{\mu} = \sigma^{-2}\mathbf{\Sigma}\mathbf{\Phi}^T\boldsymbol{t}$
4. update $\alpha_i = \dfrac{\gamma_i}{\mu_i^2}$ and $\sigma^2 = \dfrac{\|\boldsymbol{t} - \mathbf{\Phi}\boldsymbol{\mu}\|^2}{m - \Sigma_i\gamma_i}$
5. prune α_i and the basis functions $\boldsymbol{\phi}_i(\cdot)$, if $\alpha_i > \alpha_{threshold}$
6. repeat steps 3, 4 and 5 until convergence is satisfactory

A hyperparameters α_i greater than $\alpha_{threshold}$ is assumed that is tending to infinity and thus the respective w_i parameter is set to zero. The relevance vectors $\boldsymbol{x}_i$ are the remaining data points, namely those with α_i below $\alpha_{threshold}$. The prediction distribution (2.115) for a new point $\boldsymbol{x}_*$ using (2.119) becomes:

$$\begin{aligned} p(t_*|\boldsymbol{t}, \boldsymbol{\alpha}_{MP}, {\sigma^2}_{MP}) &= \int p(t_*|\boldsymbol{w}, {\sigma^2}_{MP}) p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\alpha}_{MP}, {\sigma^2}_{MP})\, d\boldsymbol{w} \\ &= \mathcal{N}(t|\boldsymbol{\mu}^T \boldsymbol{\phi}(\boldsymbol{x}_*), {\sigma^2}_*) \end{aligned} \tag{2.128}$$

The predictive mean is the prediction function $y(x_*) = \boldsymbol{\mu}^T \boldsymbol{\phi}(x_*)$ and the predictive variance is ${\sigma^2}_* = {\sigma^2}_{MP} + \boldsymbol{\phi}(\boldsymbol{x}_*)^T \boldsymbol{\Sigma} \boldsymbol{\phi}(\boldsymbol{x}_*)$ the estimated noise on the data.

2.4.5 Relevance Vector Machines (RVM) for Classification

The RVM framework has been extended to classification. We assume two classes with binary target variables $t_i \in \{0, 1\}$. The prediction function takes the form:

$$y(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}))) \tag{2.129}$$

where $\sigma(z) = \dfrac{1}{1 + \exp(-z)}$ is the logistic sigmoid function and $y(\boldsymbol{x}) \in (0, 1)$. The likelihood probability is based on the Bernoulli distribution :

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_{i=1}^{m} y(\boldsymbol{x}_i)^{t_i} [1 - y(\boldsymbol{x}_i)]^{1-t_i} \tag{2.130}$$

We use a prior $p(\boldsymbol{w}|\boldsymbol{\alpha})$ for the weights, as in the case of regression (equation 2.112). Unlike the RVM regression case, we cannot obtain an analytic form of the marginal likelihood $p(\boldsymbol{t}|\boldsymbol{\alpha})$ by integrating out weights $\boldsymbol{w}$. A workaround is to use Laplace approximation[4]. For fixed values of $\boldsymbol{\alpha}$, the mode of the posterior

[4] Laplace approximation for posterior distributions is to find the mode of the posterior and then fit a Gaussian centered at the mode.

distribution over weights $\boldsymbol{w}$ is given by maximizing:

$$\begin{aligned}
\ln p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{\alpha}) &= ln\frac{p(\boldsymbol{t}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{\alpha})}{p(\boldsymbol{t}|\boldsymbol{\alpha})} \\
&= \ln\{p(\boldsymbol{t}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{\alpha})\} - \ln p(\boldsymbol{t}|\boldsymbol{\alpha}) \\
&= \sum_{i=1}^{m}\{t_i \ln y_i + (1-t_i)\ln(1-y_i)\} - \frac{1}{2}\boldsymbol{w}^T A\boldsymbol{w} + const \qquad (2.131)
\end{aligned}$$

where $y_i = y(\boldsymbol{x}_i)$ and $\mathbf{A} = \text{diag}(\boldsymbol{\alpha})$. Equation (2.131) is a penalized logistic log-likelihood function, which can be maximized w.r.t. $\boldsymbol{w}$, using iterative reweighted least squares (IRLS) (Nabney, 1999), to find $\boldsymbol{w}_{MP}$, the mode of $\ln p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{\alpha})$. We need the gradient vector and the Hessian of the log posterior distribution:

$$\nabla \ln p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{\alpha}) = \boldsymbol{\Phi}^T(\boldsymbol{t}-\boldsymbol{y}) - \mathbf{A}\boldsymbol{w} \qquad (2.132)$$

$$\nabla\nabla \ln p(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{\alpha}) = -(\boldsymbol{\Phi}^T\mathbf{B}\boldsymbol{\Phi} + \mathbf{A}) \qquad (2.133)$$

where $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$ and $\mathbf{B} = \text{diag}[y_i(1-y_i)]_{i=1}^m$. By setting (2.132) to zero, we get the mean and the covariance of the Gaussian approximation, for the mode of the posterior distribution.

$$\boldsymbol{w}_{MP} = \mathbf{A}^{-1}\boldsymbol{\Phi}^T(\boldsymbol{t}-\boldsymbol{y}) \qquad (2.134)$$

$$\Sigma = (\boldsymbol{\Phi}^T\mathbf{B}\boldsymbol{\Phi} + \mathbf{A})^{-1} \qquad (2.135)$$

The prediction function finally becomes:

$$y(\boldsymbol{x}) = \sigma\left(\boldsymbol{w}_{MP}^T\boldsymbol{\phi}(\boldsymbol{x})\right) \qquad (2.136)$$

3

Challenge and Contributions

A key distinction among reinforcement learning algorithms relies on whether there is a representation of the value function, the policy, or both. Value function based algorithms have received much criticism in recent years due to difficulties associated with the estimation and representation of value functions. Many learning problems of interest lead to nonlinear and nonsmooth value functions that can hardly be compactly represented. On the other hand, advocates of direct policy learning have employed parametric representations that differ little from their value function representation counterparts. Most of them rely on representations of stochastic policies that take the form of a softmax over a parametric real-valued function, which is similar to a typical value function.

3.1 Starting Points

There is a trend in policy learning attempts (Lagoudakis and Parr, 2003a; Fern et al., 2004) to exploit the generalization abilities of modern classification technology, under the assumption that optimal or good deterministic policies for real-world learning problems are not arbitrarily complex, but rather exhibit a high degree of structure. Recall that a deterministic policy is a mapping from states to actions, $\pi(s) : \mathcal{S} \mapsto \mathcal{A}$. On the other hand, a multiclass classifier is a mapping of inputs to classes. Therefore, it is clear that a deterministic policy can be approximated over the entire state space of the process using a multiclass classifier over the same space; each action is viewed as a distinct class and the states of the process are the instances (inputs) to be classified (Figure 3.1). It should, therefore, be plausible to learn a good policy using only a small set of training data consisting of selected states over the state space labeled with the actions that are deemed to be best in those states.

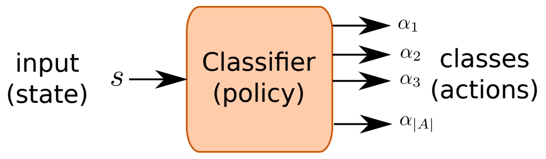


FIGURE 3.1: A multiclass classifier used to map states to actions

Reinforcement Learning as Classification

To illustrate the learning process under such representations, we briefly review the *Rollout Classification Policy Iteration* (RCPI) algorithm (Lagoudakis and Parr, 2003a). The key idea behind RCPI is to cast the problem of policy learning as a classification problem. Finding a good policy becomes equivalent to finding a classifier that generalizes well over the state space and maps states to "good" actions, where the goodness of an action is measured regarding its contribution to the long-term goal of the agent. The state-action value function Q_π

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \bar{\mathcal{P}}(s,a,s') \Big\{ \tilde{\rho}(s,a,s') + \gamma Q_\pi(s', \pi(s')) \Big\}$$

provides such a measure given a fixed base policy π; the action that maximizes Q_π in state s is a "good" action in that state, whereas any action with strictly smaller Q_π value is a "bad" one. The policy π' formed by choosing maximizing actions in each state

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q_\pi(s,a)$$

is guaranteed to be at least as good as π, if not better. A training set $\{(s_i, \pi'(s_i))\}_{i=1}^m$ for policy π' could be easily formed, if the Q_π values for all actions were available for a subset of states $\{s_i\}_{i=1}^m$. Thus, policy learning can be seen as classifier learning (Figure 3.2). Once trained, the classifier can deliver the estimated action choice $\widehat{\pi}'(s)$ of policy π' for any input state $s \in \mathcal{S}$.

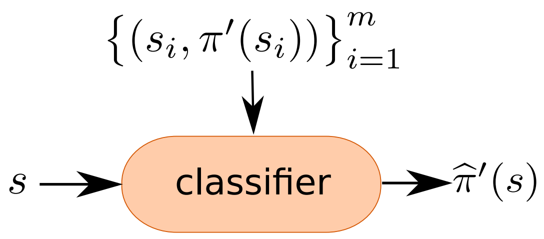


FIGURE 3.2: Policy learning as classifier learning

The Monte-Carlo estimation technique of *rollouts* (Bertsekas and Tsitsiklis, 1996; Tesauro and Galperin, 1997) provides a way of accurately estimating Q_π at any given state-action pair (s, a) without requiring an explicit representation of the value function or the full MDP model. Rollouts require only a generative model (a simulator) of the process; more specifically, given any state-action pair (s, a), such a model returns a next state s' and a reward r' sampled from the (unknown) true MDP model distributions (transition model and reward function). A rollout for (s, a) amounts to simulating a trajectory of the process beginning from state s, choosing action a for the first step and actions according to policy π thereafter up to a certain horizon H, and computing the total discounted reward along this trajectory. The observed total discounted reward is averaged over a number of rollouts (also called trials) K, to yield an accurate estimate of $Q_\pi(s, a)$. Thus, using a sufficient amount of rollouts, it is possible to create a training example of the improved policy π' in any chosen state s. A collection of such examples over a finite set of states forms a valid training set for the improved policy π' over any base policy π.

The goal of the learner is not only to improve a policy, but rather to find a good or even optimal policy. Therefore, RCPI employs an approximate policy

Algorithm 11 Rollout Classification Policy Iteration (RCPI)

```
Input: policy $\pi_0$, trials $K$, horizon $H$, sample size $U_r$

$k = -1$
repeat
    $k = k + 1$
    $S_{k+1}$ = a uniformly random subset of $\mathcal{S}$ of size $U_r$    (uniform sampling)
    $T_{k+1} = \varnothing$    (initialization of the training set)
    for (each $s \in S_{k+1}$)
        for (each $a \in \mathcal{A}$)
            estimate $Q_{\pi_k}(s, a)$ using $K$ rollouts of length $H$    (simulation)
        end
        if (a dominating action $a^*$ exists in state $s$) then
            $T_{k+1} = T_{k+1} \cup \{(s, a^*)^+\}$ (positive example for dominating action $a^*$)
            $T_{k+1} = T_{k+1} \cup \{(s, a)^-\}$ (negative examples for dominated actions $a$)
        end
    end
    $\pi_{k+1}$ = TrainClassifier($T_{k+1}$)    (classifier/policy learning)
until ($\pi_{k+1}$ is not better than $\pi_k$)    (end of policy iteration)
return $\pi_k$
```

iteration scheme for repeated improvements, as described in Algorithm 11. At each iteration, a new policy/classifier is produced using training data obtained by rolling out the previous policy on a generative model of the process. Beginning with any initial policy π_0, at each iteration k a training set over a subset of states S_k is formed by querying the rollout procedure to identify dominating actions in the states of S_k. Notice that the training set contains both positive (+) and negative (−) examples for each state, wherever a clear action domination is found. A new classifier is trained using these examples to yield an approximate representation of the improved policy over the previous one. This cycle is repeated until a termination condition is met. Given the approximate nature of this policy iteration scheme, the termination condition cannot rely on convergence to a single optimal policy. Rather, it terminates when the performance of the new policy (measured again via simulation) does not exceed that of the previous policy. The empirical expected total discounted reward from states drawn from $\mathcal{D}$ obtained from and averaged over multiple runs is used as the policy performance criterion.

The RCPI algorithm, shown graphically in Figure 3.3, yielded promising results (Lagoudakis and Parr, 2003a) in the pendulum and the bicycle domains using Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs) as

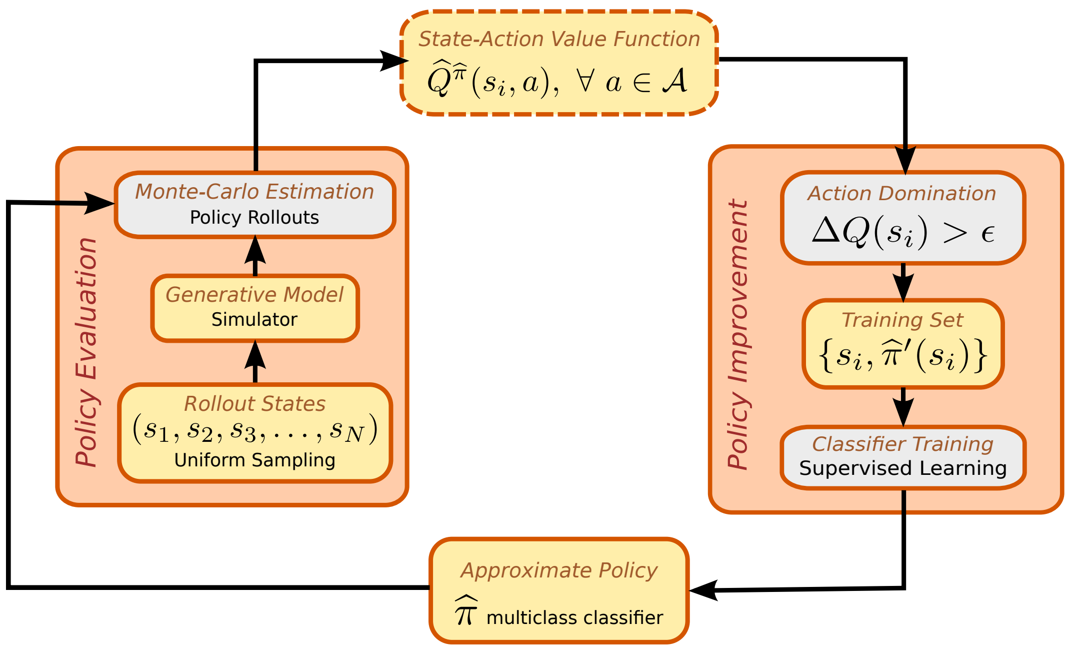


FIGURE 3.3: Rollout Classification Policy Iteration (RCPI) algorithm

classifiers.

3.2 Questions and Challenges

The reinforcement learning as classification experience, despite subsequent progress, still leaves us with several open research questions:

- Do good policies for typical decision problems exhibit significant structure?
- Can classifiers capture and generalize policy structure efficiently?
- How big of a training set for the classifier is required?
- What is the effect of the distribution of states in the training set?
- Are there critical areas of the state space where examples are required?
- How can such critical areas of the state space be identified?
- Does the classifier/policy itself reveal information about these areas?
- Can we use the classifier/policy to direct state sampling to critical areas?
- Which classification technology balances performance and complexity?

The Rollout Classification Policy Iteration (RCPI) algorithm gives us a strong indication that the questions posted above can be answered positively. There-

Hypothesis under test

fore, in this dissertation, we took up the challenge to show that *we can efficiently explore the policy space by exploiting hints of the classifiers we use for the representation of policies.*

3.3 Contributions

Optimal policy structure

Our initial effort was to uncover and study the optimal policy structure for typical two-dimensional reinforcement learning domains, such as the Inverted Pendulum and the Mountain Car, which are appropriate for visualization and inspection (Rexakis and Lagoudakis, 2008). The results clearly demonstrate that good policies exhibit significant structure, which can potentially be learned and exploited for representational purposes. An optimal deterministic policy is a mapping from states to actions, and optimal actions persist over large areas in the state space.

Our focus then moved to finding how to direct the exploration of policy space using rollouts and uncover areas, where an action prevails, in a smart and systematic way. We tried to avoid the use of value functions due to the known difficulties associated with their representation. Instead, we focused on using policy rollouts, which can provide accurate estimates of Q-values in any state by repeated simulations. We use a collection of binary classifiers to separate action areas and represent a policy. A binary classifier separates within the state space a dominant action from all the others.

Directed exploration of policy space

We developed two methods for directed exploration of policy space, using SVM and RVM classifiers. In the first one (Rexakis and Lagoudakis, 2011), we are exploiting the structure of the classifier to direct sampling. In the second one (Rexakis and Lagoudakis, 2012, 2014), we utilize a regressor, approximating the action advantage in any given state, to estimate the importance of each state, using readily available data, while improving upon the current policy. In both approaches, the search is focused on areas where there is a change/switch of action domination. This directed focus on critical parts of the state space iteratively leads to refinement and improvement of the underlying policy and delivers excellent control policies in only a few iterations.

The proposed methods have been thoroughly tested on three well-known learning domains: Inverted Pendulum, Mountain Car, and Acrobot. The first two domains are appropriate for visualization and inspection thanks to their low, two-dimensional state space. The third domain however is four-dimensional and reveals the usefulness of the proposed directed exploration methods towards discovering the most critical parts of the state space. Additionally, we demonstrate the scalability of the proposed approaches on the problem of learning how to control a 4-Link, Under-Actuated, Planar Robot, which corresponds to an eight-dimensional problem, well-known in the control theory community. In all cases, the proposed Directed RCPI algorithms yield policies of excellent performance and demonstrate significants savings in rollout sampling compared to the original RCPI algorithm.

4

Related Work

There is a large body of literature in the area of reinforcement learning combined with supervised learning (classification and regression). This chapter reviews the most representative publications in this area, following a chronological order.

Dietterich and Wang (2001) proposed a reinforcement learning approach based on batch value function approximation. They use a set of states with known values of the V-function and they train an SVM-like regressor with (state, V-value) tuples, to generalize the value function over the entire state space. They exploit the full MDP model and use three different linear programming formulations (supervised, Bellman, and advantage learning). Their algorithms were applied to 10 maze problems and delivered promising results.

Yoon et al. (2002) use an inductive policy selection method for probabilistic STRIPS domains (blocks world, paint world, logistics world). They do not compute a value function in large domains, but they attempt to generalize good policies from domains with few objects to get a useful policy for domains with many objects. Their policies are represented as ensembles of decision lists, using a taxonomic concept language, and they use bootstrap aggregation (bagging) to resolve.

Kakade and Langford (2002) reduced reinforcement learning to regression and they introduced the Conservative Policy Iteration algorithm. They define a policy advantage function, and they perform policy updates wherever the advantage function takes significant values.

The Rollout Classification Policy Iteration (RCPI) algorithm was proposed by Lagoudakis and Parr (2003a) as an alternative to standard approximate policy iteration. They replace the value function $V_\pi(s)$ learning step with rollout estimates of the action value function $Q_\pi(s, a)$ over a finite number of states for all the actions in the action space, and they cast the policy improvement step as a multi-class classification problem. The RCPI algorithm yielded promising results in the pendulum and the bicycle domains using Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs) as classifiers.

A similar algorithm proposed by Fern et al. (2004, 2006, 2007) yielded satisfying results in seven deterministic and stochastic relational planning domains from the AIPS-2000 planning competition using Decision Lists as the underlying classifier. The primary difference with RCPI is the form of the classification problem produced on each iteration. Lagoudakis and Parr (2003a) generate standard multi-class classification problems, whereas they generate cost-sensitive problems.

Bagnell et al. (2004) introduced an algorithm for learning non-stationary policies in reinforcement learning. For a specified horizon H, their approach learns a sequence of H policies. At each iteration, all policies are fixed except for one, which is optimized by forming a classification problem via policy rollout.

Langford and Zadrozny (2005) provided a formal reduction from reinforcement learning to classification, showing that ϵ-accurate classification implies near optimal reinforcement learning. They use an optimistic variant of sparse sampling to generate H classification problems, one for each horizon time step $(1, 2, 3, \ldots, H)$.

Li et al. (2004, 2007) suggested a focused attention reinforcement learning scheme. The main difference with Lagoudakis and Parr (2003a) is that they use a cost function based on a Q-value advantage function, to grade the state importance, construct a training set with $(state, action, cost)$ tuples and train a cost sensitive classifier.

Dimitrakakis and Lagoudakis (2008a) proposed a variant of a policy iteration scheme which addresses the core sampling problem in evaluating a policy through simulation as a multi-armed bandit machine. They view the set of rollout states as a multi-armed bandit machine, where each state corresponds to a single lever (arm). Pulling a lever corresponds to sampling the corresponding state once, i.e., perform a single rollout for each action in that state. They employed heuristic variants of well-known algorithms for bandit problems, such as Upper Confidence Bounds (Auer et al., 2002) and Successive Elimination (Even-Dar et al., 2006), and they presented experiments on two standard reinforcement learning domains, the Inverted Pendulum and the Mountain Car.

Dimitrakakis and Lagoudakis (2008b) offer theoretical insight into the rollout sampling problem. They analyze the sample allocation methods described in Dimitrakakis and Lagoudakis (2008a). They compared the performance of *Count* and *Fixed* allocation schemes with additional ones inspired by the *Upper Confidence Bounds* (Auer et al., 2002) and *Successive Elimination* (Even-Dar et al., 2006) algorithms. They found that all methods outperform the *Fixed* scheme in practice, sometimes by an order of magnitude.

Gabillon et al. (2010) suggest a sampling scheme for classification-based policy iteration algorithms similar to Dimitrakakis and Lagoudakis (2008a). They use strategies to allocate the available budget of rollouts at each iteration over states and actions. They applied their algorithm in two domains, the Inverted Pendulum and the Mountain Car.

Rachelson and Lagoudakis (2010) introduced the active learning scheme of

Localized Policy Iteration (LPI), an active learning version of Rollout-based Policy Iteration. They link the Lipschitz continuity of a Markov Decision Process to the Lipschitz continuity of value functions. They introduce the notion of influence radius of a state-action pair (s, a), where a is the best action in s, and define a volume around s in the state space, where the best action a is guaranteed to be dominant. They tested it on a standard Inverted Pendulum problem.

Lazaric et al. (2010) derive a finite-sample analysis of a Classification-based API algorithm - called Direct Policy Iteration (DPI). The use a cost-sensitive loss function, weighing each classification error by its regret (the difference between the action-value of the greedy action and the action chosen by the current policy). They explicitly assume that the action space is two dimensional and they study how the expected error propagates through algorithm iterations.

Gabillon et al. (2011) suggest the addition of a critic, a value function approximation component, to rollout classification-based policy iteration algorithms. They present a new RCPI algorithm, called Direct Policy Iteration with Critic (DPI-Critic). They provide its finite-sample analysis when the critic using the LSTD method. The idea is to use a critic to approximate the return after we truncate the rollout trajectories. This allows us to control the bias and variance of the rollout estimates of the action-value function. They use a cost-sensitive loss function, the same they used in the DPI paper (Lazaric et al., 2010), and they train a cost-sensitive multi-class classifier to return a policy that minimizes the empirical error.

Cheng et al. (2011); Fürnkranz et al. (2012) proposed a preference-based extension of approximate policy iteration. They use a preference learning method called label ranking, to allow sorting of available actions from most promising to least promising for each state, and they train a label ranker instead of a classifier. They present experiments on Inverted Pendulum, Mountain Car, and cancer clinical trials domains.

Farahmand et al. (2012, 2013, 2014, 2015) analyze a general algorithm of Classification-based Approximate Policy Iteration (CAPI) theoretically. They define an action-gap weighted loss function, and then they minimize that function. They exploit the action-gap regularity (Farahmand, 2011) in the analysis of classification-based reinforcement learning.

5

Benchmark Domains

In this thesis, we chose to study four standard domains in reinforcement learning: Inverted Pendulum, Mountain Car, Acrobot and 4-Link Planar Robot. The first two problems are defined on two-dimensional continuous state spaces, and therefore they are appropriate for visualization and inspection. The third and fourth problems, are defined on a four-dimensional continuous state space and a eight-dimensional continuous state respectively, and therefore are appropriate for verifying that rollout sampling for learning is indeed directed to important parts of the state space.

5.1 Inverted Pendulum

The *Inverted Pendulum* problem is to balance a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to (Figure 5.1). Three actions are allowed: left force LF (-50 Newtons), right force RF ($+50$ Newtons), or no force NF (0 Newtons). In the stochastic version of the problem, all three actions are noisy; Gaussian noise with $\mu = 0$ and $\sigma^2 = 10$ is added to the chosen action. There is no noise in the deterministic version. The state space of the problem is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. The transitions are governed by the nonlinear dynamics of the system (Wang et al., 1996) and depend on the current state and the current (noisy or noiseless) control u:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2\sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha ml\cos^2(\theta)}$$

where g is the gravity constant ($g = 9.8m/s^2$), m is the mass of the pendulum

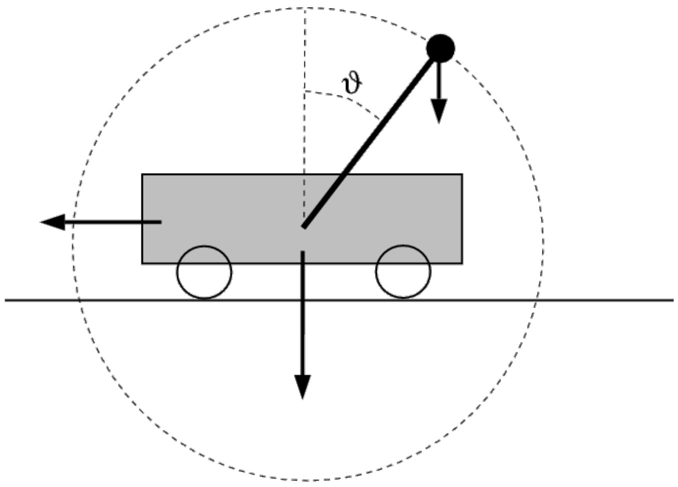


FIGURE 5.1: Inverted Pendulum

(default: $m = 2.0$ kg), M is the mass of the cart (default: $M = 8.0$ kg), l is the length of the pendulum (default: $l = 0.5$ m), and $\alpha = 1/(m + M)$. The simulation step is 0.1 seconds. Thus the control input is given at a rate of 10 Hz, at the beginning of each time step, and is kept constant during any time step. A reward of $+1$ is given as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value (the pendulum is above the horizontal line). An angle greater than $\pi/2$ in absolute value signals the end of the episode and a reward of 0. The discount factor of the process is set to 0.95.

5.2 Mountain Car

The *Mountain Car* problem is to drive an underpowered car from the bottom of a valley between two mountains to the top of the mountain on the right (Figure 5.2). The car is not powerful enough to climb any of the hills directly from the bottom of the valley even at full throttle; it must build some energy by climbing first to the left (moving away from the goal) and then to the right. Three actions are allowed: forward throttle FT (+1), reverse throttle RT (-1), or no throttle NT (0). In the deterministic version of the problem, as originally specified (Sutton and Barto, 1998), there is no noise. In the stochastic version, to make the problem a little more challenging, we have added noise to all three actions; Gaussian noise with $\mu = 0$ and $\sigma^2 = 0.2$ is added to the chosen action. The state space of the problem is continuous and consists of the position x

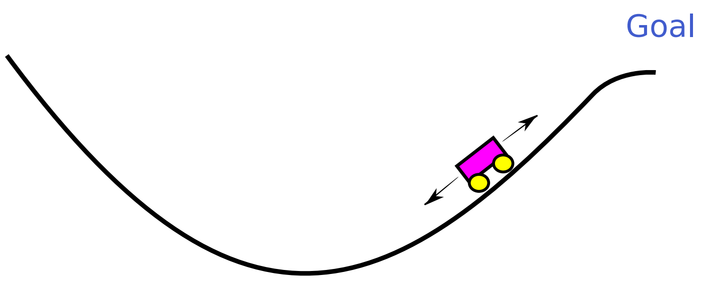


FIGURE 5.2: Mountain Car

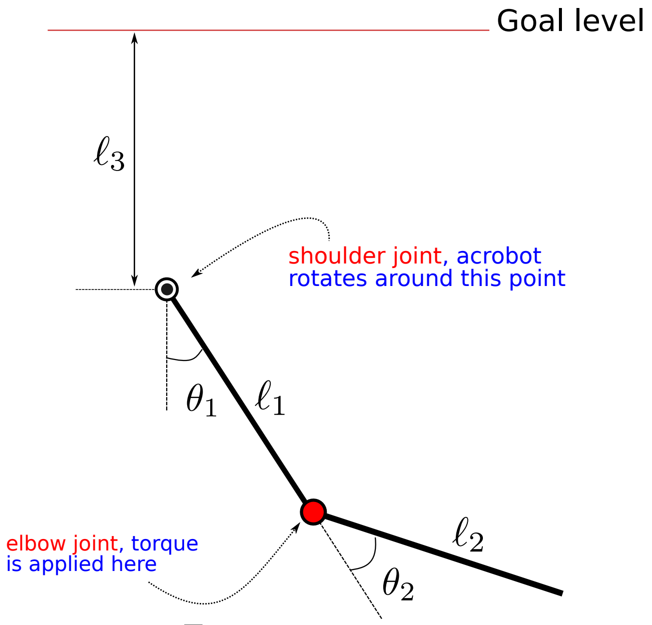
and the velocity $\dot{x}$ of the car. The transitions are governed by the simplified nonlinear dynamics of the system (Sutton and Barto, 1998) and depend on the current state $(x(t), \dot{x}(t))$ and the current (noisy or noiseless) control $u(t)$:

$$\begin{aligned} x(t+1) &= \textsc{Bound}_x[x(t) + \dot{x}(t+1)] \\ \dot{x}(t+1) &= \textsc{Bound}_{\dot{x}}[\dot{x}(t) + 0.001u(t) - 0.0025\cos(3x(t))] \end{aligned}$$

where $\textsc{Bound}_x$ is a function that keeps x within $[-1.2, 0.5]$, while $\textsc{Bound}_{\dot{x}}$ keeps $\dot{x}$ within $[-0.07, 0.07]$. The point $(-0.5, 0)$ corresponds to the bottom of the valley when the car is not moving. If the car hits the bounds of the position x, the velocity $\dot{x}$ is set to zero. A reward of 0 is given at each step as long as the position of the car is below the right bound (0.5). As soon as the car position hits the right bound of position, it has reached the goal; the episode ends successfully, and a reward of +1 is given. The discount factor of the process is set to 0.99.

5.3 Acrobot

The *Acrobot* is a nonlinear dynamical system composed of a two-link underactuated robot arm which is allowed to rotate around a fixed point at the shoulder joint (Figure 5.3). Torque τ can be applied to the elbow joint only. The goal is to swing the arm around the fixed point so that the other end reaches a height which is one link's length higher than the fixed point. This system has been studied by robotics and control engineers (Spong, 1994; DeJong and Spong, 1994).



Goal level
ℓ_3
shoulder joint, acrobot rotates around this point
θ_1
ℓ_1
ℓ_2
elbow joint, torque is applied here
θ_2

FIGURE 5.3: Acrobot

Three levels of torque are allowed: positive (+1), negative (−1), or no torque (0). Gaussian noise ($\mu = 0$, $\sigma^2 = 0.2$) is added to the chosen action. The system is described by four state variables: shoulder angular position $\theta_1 \in [0, 2\pi]$, shoulder angular velocity $\dot{\theta}_1 \in [-4\pi, +4\pi]$, elbow angular position $\theta_2 \in [0, 2\pi]$, elbow angular velocity $\dot{\theta}_2 \in [-9\pi, +9\pi]$. Any values of θ_1, θ_2 outside $[0, 2\pi]$ are wrapped into $[0, 2\pi]$ by subtracting $2k\pi$, where $k \in \mathbb{Z}$. The transitions are governed by the nonlinear dynamics of the system (Spong, 1994; Sutton and Barto, 1998) and depend on the current state $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ and the current control τ:

$$\begin{aligned}
\ddot{\theta}_1 &= -\frac{(d_2\ddot{\theta}_2 + \phi_1)}{d_1}, \qquad \ddot{\theta}_2 = \frac{(\tau + \frac{d_2}{d_1}\phi_1 - m_2\ell_1\ell_{c2}\dot{\theta}_1^2 \sin\theta_2 - \phi_2)}{(m_2\ell_{c2}^2 + I_{c2} - \frac{d_2^2}{d_1})} \\
d_1 &= m_1\ell_{c1}^2 + m_2(\ell_1^2 + \ell_{c2}^2 + 2\ell_1\ell_{c2}\cos\theta_2) + I_{c1} + I_{c2} \\
d_2 &= m_2(\ell_{c2}^2 + \ell_1\ell_{c2}\cos\theta_2) + I_2 \\
\phi_1 &= -m_2\ell_1\ell_{c2}\dot{\theta}_2^2\sin\theta_2 - 2m_2\ell_1\ell_{c2}\dot{\theta}_1\dot{\theta}_2\sin\theta_2 + \\
&\qquad (m_1\ell_{c1} + m_2\ell_1)g\cos\left(\theta_1 - \frac{\pi}{2}\right) + \phi_2 \\
\phi_2 &= m_2\ell_{c2}g\cos\left(\theta_1 + \theta_2 - \frac{\pi}{2}\right)
\end{aligned}$$

where $g = 9.8\text{m/s}^2$ is the gravity constant, $m_1 = m_2 = 1\text{Kg}$ are the masses of the links, $l_1 = l_2 = 1\text{m}$ are the lengths of links, goal level $l_3 = 1\text{m}$ above rotation point, $l_{c1} = l_{c2} = 0.5\text{m}$ are the lengths to the center of mass of the links, and $I_{c1} = I_{c2} = 1\ \text{Kg}\cdot\text{m}^2$ are the moments of inertia of the links. A time step of 0.05 seconds was used in the simulation, with actions chosen after every four time steps, a total of 0.2 seconds per action. The angular velocities are limited so that they stay within their bounds. The discount factor of the process is set to 0.98.

Acrobot Reward

The classic reward for the Acrobot, suggested by (Sutton and Barto, 1998), is 1 upon reaching the goal and 0 otherwise. We propose here a shaping reward for the Acrobot, because the original delayed reward scheme could not provide any guidance to the initial random controller. We define the reward for transitioning from state s to state s' by taking action a as $\tilde{\rho}(s, a, s') = E_1(s') \times E_2(s') - E_1(s) \times E_2(s)$, where $E_1(s)$ and $E_2(s)$ is the total mechanical energy for state s for the shoulder and the elbow arm, respectively. Consider the Acrobot shown in Figure 5.4. Intuitively, a policy that drives the Acrobot tip to the goal (level ℓ_3 above the shoulder joint) is a policy that maximizes the mechanical energy of both arms. In a system where the upper arm is moving slowly and the elbow is spinning, the mechanical energy of the system is high, but the upper arm will not make it high enough for the tip to reach the goal.

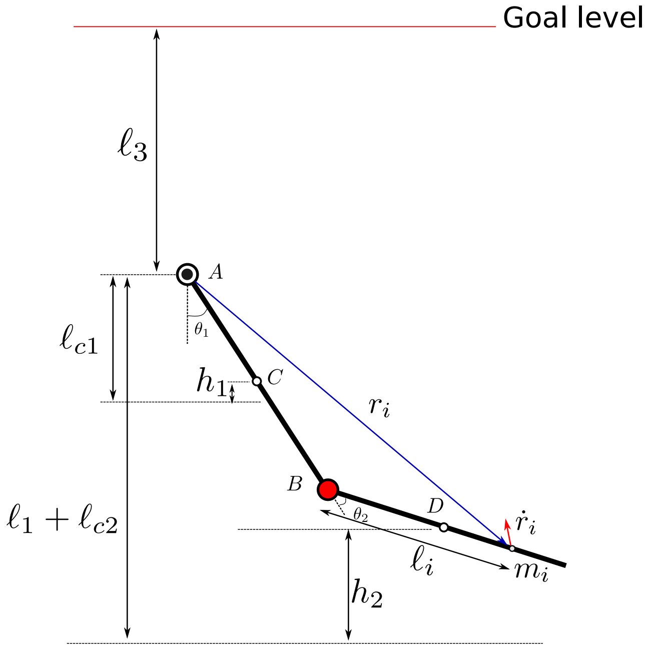


FIGURE 5.4: Acrobot reward

The shoulder arm is a line segment AB with length ℓ_1 and the elbow arm is a line segment BC with length ℓ_2. The center of mass is D for the shoulder arm and E for elbow arm. The mechanical energy for the shoulder arm is $E_1 = U_1 + K_1$. The gravitational potential energy is $U_1 = h_1 m_1 g$, where $h_1 = \ell_{c1}(1 - \cos(\theta_1))$. The kinetic energy of the shoulder arm due to rotation about the shoulder joint A is:

$$K_1 = \frac{1}{2} I_{1A} \dot{\theta_1}^2 \tag{5.1}$$

where the $I_{1A} = I_{c1} + m_1 \ell_{c1}^2$ computed from I_{c1} using parallel axis theorem. Similarly, the mechanical energy for the elbow arm is $E_2 = U_2 + K_2$. The potential energy due to the gravity is $U_2 = h_2 m_2 g$, where $h_2 = (\ell_1 + \ell_{c2}) - (\ell_1 \cos(\theta_1) + \ell_{c2} \cos(\theta_1 + \theta_2))$. The kinetic energy of the elbow arm:

$$K_2 = \frac{1}{2} \sum_i m_i \dot{\boldsymbol{r}}^T \dot{\boldsymbol{r}} \tag{5.2}$$

where m_i the mass of point i of the elbow arm, $\boldsymbol{r}$ is the distance from the shoulder joint A, and $\dot{\boldsymbol{r}}$ the speed of point i. Let ℓ_i be the distance of point i

from the elbow joint B.

$$\boldsymbol{r} = \begin{bmatrix} \ell_1 \sin(\theta_1) & \ell_i \sin(\theta_1 + \theta_2) \\ -\ell_1 \cos(\theta_1) & -\ell_i \cos(\theta_1 + \theta_2) \end{bmatrix}$$

and

$$\dot{\boldsymbol{r}} = \begin{bmatrix} \ell_1 \dot{\theta}_1 \cos(\theta_1) & \ell_i(\dot{\theta}_1 + \dot{\theta}_2) \cos(\theta_1 + \theta_2) \\ \ell_1 \dot{\theta}_1 \sin(\theta_1) & \ell_i(\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_1 + \theta_2) \end{bmatrix}$$

$$\begin{aligned} \dot{\boldsymbol{r}}^T \dot{\boldsymbol{r}} = & \begin{bmatrix} \ell_1 \dot{\theta}_1 \cos(\theta_1) & \ell_1 \dot{\theta}_1 \sin(\theta_1) \\ \ell_i(\dot{\theta}_1 + \dot{\theta}_2) \cos(\theta_1 + \theta_2) & \ell_i(\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_1 + \theta_2) \end{bmatrix} \\ & \begin{bmatrix} \ell_1 \dot{\theta}_1 \cos(\theta_1) & \ell_i(\dot{\theta}_1 + \dot{\theta}_2) \cos(\theta_1 + \theta_2) \\ \ell_1 \dot{\theta}_1 \sin(\theta_1) & \ell_i(\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_1 + \theta_2) \end{bmatrix} \\ = & \ell_1^2 \dot{\theta}_1^{\,2} \cos^2(\theta_1) + \ell_i^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 \cos^2(\theta_1 + \theta_2) + \\ & 2\ell_1 \dot{\theta}_1 \cos(\theta_1) \ell_i (\dot{\theta}_1 + \dot{\theta}_2) \cos(\theta_1 + \theta_2) + \\ & \ell_1^2 \dot{\theta}_1^{\,2} \sin^2(\theta_1) + \ell_i^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 \sin^2(\theta_1 + \theta_2) + \\ & 2\ell_1 \dot{\theta}_1 \sin(\theta_1) \ell_i (\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_1 + \theta_2) \end{aligned} \tag{5.3}$$

Using the identity $\cos(\theta_1)\cos(\theta_1 + \theta_2) + \sin(\theta_1)\sin(\theta_1 + \theta_2) = \cos(\theta_2)$ and the above K_2 equation becomes

$$K_2 = \frac{1}{2} \sum_i m_i \left(\ell_1^2 \dot{\theta}_1^{\,2} + 2\ell_1 \ell_i \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2) \cos(\theta_2) + \ell_i^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 \right) \tag{5.4}$$

given that the inertia of the elbow around joint B is:

$$I_{2B} = \sum_i m_i \ell_i^2$$

and the distance from B to the elbow center of mass:

$$\ell_{c2} = \frac{\sum_i m_i \ell_i}{m_2}$$

K_2 equation becomes:

$$\begin{aligned} K_2 = & \frac{1}{2} \left(m_2 \ell_1^2 + I_{2B} + 2 m_2 \ell_1 \ell_{c2} \cos(\theta_2) \right) \dot{\theta}_1^{\,2} + \\ & \frac{1}{2} I_{2B} \dot{\theta}_2^{\,2} + \\ & \left(I_{2B} + m_2 \ell_1 \ell_{c2} \cos(\theta_2) \right) \dot{\theta}_1 \dot{\theta}_2 \end{aligned} \tag{5.5}$$

where $I_{2B} = I_{c2} + m_2\ell_{c2}^2$.

A state s is a tuple $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ recording the two angles and the two angular velocities. Given the above formulas for the potential and kinetic energies, the immediate reward $\tilde{\rho}(s, a, s')$ is defined as:

$$\begin{aligned} \tilde{\rho}(s, a, s') &= E_1(s')E_2(s') - E_1(s)E_2(s), \\ &= (U_1' + K_1')(U_2' + K_2') - (U_1 + K_1)(U_2 + K_2) \end{aligned} \tag{5.6}$$

The justification for this definition is that the immediate reward represents the momentary difference in the products of total energy between the new and the old state; the product of total energies of the elbow and shoulder joints implies that energy gains are attributed to both arms.

5.4 4-Link Planar Robot

The *4-Link Planar Robot* is a nonlinear dynamical system composed of a four-link underactuated robot arm with actuation only on the middle joint (Figure 5.5). In particular, torque (τ_3) is applied to the third joint. The goal for the robot is to swing around the first joint, a fixed rotation point, so that the other end reaches the 80% of the maximum height the robot can achieve above the rotation point.

Three levels of torque are allowed: positive (+10), negative (−10), or no torque (0). Gaussian noise ($\mu = 0$, $\sigma^2 = 2$) is added to the chosen action. The system is described by eight state variables, the absolute angle of each joint $\theta_1, \theta_2, \theta_3, \theta_4 \in [0, 2\pi]$ and the angular velocity of each joint $\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4 \in [-2\pi, +2\pi]$. Any values of $\theta_1, \theta_2, \theta_3, \theta_4$ outside $[0, 2\pi]$ are wrapped back into $[0, 2\pi]$ by subtracting $2k\pi$, where $k \in \mathbb{Z}$. The dynamics of this system have been studied by robotics and control engineers (Xin and Liu, 2014). The transitions are governed by the nonlinear dynamics of the system and depend on the current state $(\theta_1, \theta_2, \theta_3, \theta_4, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4)$ and the current control τ_3 applied to third joint. The gravity constant is $g = 9.8\text{m/s}^2$, $m_1 = m_2 = m_3 = m_4 = 0.5\text{Kg}$ are the masses of the links, $\ell_1 = \ell_2 = \ell_3 = \ell_4 = 0.5\text{m}$ are the lengths of links, $\ell_{c1} = \ell_{c2} = \ell_{c3} = \ell_{c4} = 0.25\text{m}$ are the lengths to the center of mass of the links, and $J_1 = J_2 = J_3 = J_4 = 0.0417 \text{ Kg} \cdot \text{m}^2$ are the moments of inertia of the links. Time step is 0.06 seconds. The discount factor of the process is set to 0.92.

The motion equation of the robot is:

Equation of motion

$$M(q)\ddot{q} + H(q, \dot{q}) + G(q) = \tau,$$

where vector $q \in \mathbb{R}^4$ holds the relative angles $q_1 = \theta_1, \quad q_2 = \theta_2 - \theta_1, \quad q_3 = \theta_3 - \theta_2, \quad q_4 = \theta_4 - \theta_3$; $M(q) \in \mathbb{R}^{4\times4}$ is a symmetric positive definite inertia matrix; vector $H(q, \ddot{q}) \in R^4$ contains the Coriolis and centripetal terms; vector

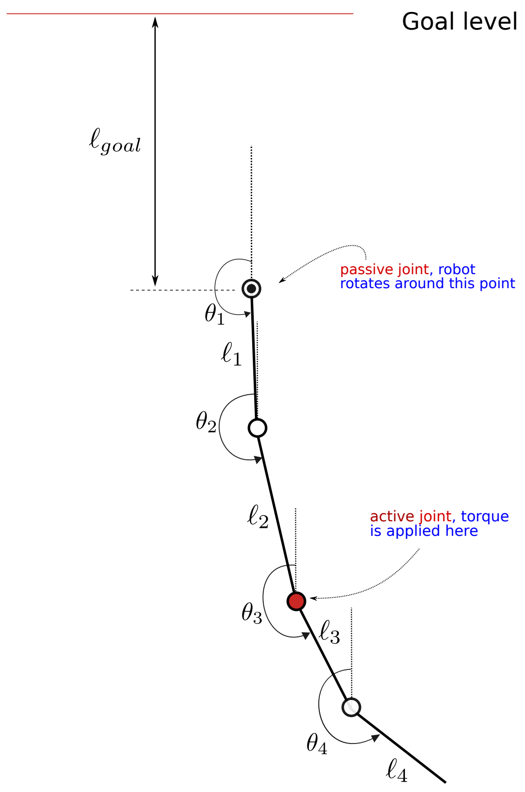


FIGURE 5.5: 4-Link Planar Robot with active third joint

$G(q) \in \mathbb{R}^4$ contains the gravitational terms; and vector $\tau \in \mathbb{R}^4$ holds the torque applied to each joint. In our case, $\tau = [0, 0, \tau_3, 0]$ where τ_3 is the torque applied to the third joint.

The non-zero entries of matrix $M(q) \in \mathbb{R}^{4\times4}$ are shown below; all others are zero.

$$
\begin{aligned}
M_{11} =& a_{11} + a_{22} + a_{33} + a_{44} + 2a_{12}\cos q_2 + 2a_{13}\cos(q_2+q_3) + \\
& 2a_{14}\cos(q_2+q_3+q_4) + 2a_{23}\cos q_3 + 2a_{24}\cos(q_3+q_4) + 2a_{34}\cos q_4 \\
M_{12} =& a_{22} + a_{33} + a_{44} + a_{12}\cos q_2 + a_{13}\cos(q_2+q_3) + a_{14}\cos(q_2+q_3+q_4) + \\
& 2a_{23}\cos q_3 + 2a_{24}\cos(q_3+q_4) + 2a_{34}\cos q_4 \\
M_{13} =& a_{33} + a_{44} + a_{13}\cos(q_2+q_3) + a_{14}\cos(q_2+q_3+q_4) + \\
& a_{23}\cos q_3 + a_{24}\cos(q_3+q_4) + 2a_{34}\cos q_4 \\
M_{14} =& a_{44} + a_{14}\cos(q_2+q_3+q_4) + a_{24}\cos(q_3+q_4) + a_{34}\cos q_4 \\
M_{22} =& a_{22} + a_{33} + a_{44} + 2a_{23}\cos q_3 + 2a_{24}\cos(q_3+q_4) + 2a_{34}\cos q_4 \\
M_{23} =& a_{33} + a_{44} + a_{23}\cos q_3 + a_{24}\cos(q_3+q_4) + 2a_{34}\cos q_4 \\
M_{24} =& a_{44} + a_{24}\cos q_3 + q_4 + a_{34}\cos q_4 \\
M_{33} =& a_{33} + a_{44} + 2a_{34}\cos q_4 \\
M_{34} =& a_{44} + a_{34}\cos q_4 \\
M_{44} =& a_{44}
\end{aligned}
$$

where the a_{xx} constants are

$$
\begin{aligned}
a_{11} &= J_1 + m_1\ell_{c1}^2 + (m_2+m_3+m_4)\ell_1^2 \\
a_{12} &= m_2\ell_1\ell_{c2} + (m_3+m_4)\ell_1\ell_2 \\
a_{13} &= m_3\ell_1\ell_{c3} + m_4\ell_1\ell_3 \\
a_{14} &= m_4\ell_1\ell_{c4} \\
a_{22} &= J_2 + m_2\ell_{c2}^2 + (m_3+m_4)\ell_2^2 \\
a_{23} &= m_3\ell_2\ell_{c3} + m_4\ell_2\ell_3 \\
a_{24} &= m_4\ell_2\ell_{c4} \\
a_{33} &= J_3 + m_3\ell_{c3}^2 + m_4\ell_3^2 \\
a_{34} &= m_4\ell_3\ell_{c4} \\
a_{44} &= J_4 + m_4\ell_{c4}^2
\end{aligned}
$$

the entries of the H vector are

$$\begin{aligned}
H_1 =& - a_{12}\dot{q}_2(2\dot{q}_1 + \dot{q}_2)\sin q_2 - a_{23}\dot{q}_3(2(\dot{q}_1 + \dot{q}_2) + \dot{q}_3)\sin q_3 \\
& - a_{34}(2\dot{q}_2\dot{q}_3 + \dot{q}_4(2(\dot{q}_1 + \dot{q}_3) + \dot{q}_4))\sin q_4 \\
& - a_{13}(\dot{q}_2 + \dot{q}_3)(2\dot{q}_1 + \dot{q}_2 + \dot{q}_3)\sin(q_2 + q_3) \\
& - a_{24}((\dot{q}_3 + \dot{q}_4)^2 + 2\dot{q}_1(\dot{q}_3 + \dot{q}_4) + 4\dot{q}_2\dot{q}_3)\sin(q_3 + q_4) \\
& - a_{14}(\dot{q}_2^2 + 4\dot{q}_2\dot{q}_3 + (\dot{q}_3 + \dot{q}_4)^2 + 2\dot{q}_1(\dot{q}_2 + \dot{q}_3 + \dot{q}_4))\sin(q_2 + q_3 + q_4) \\
H_2 =& a_{12}\dot{q}_1^2\sin q_2 - a_{23}\dot{q}_3(2(\dot{q}_1 + \dot{q}_2) + \dot{q}_3)\sin q_3 \\
& - a_{34}\dot{q}_4(2(\dot{q}_1 + \dot{q}_2 + \dot{q}_3) + \dot{q}_4)\sin q_4 + a_{13}\dot{q}_1^2\sin(q_2 + q_3) \\
& - a_{24}((\dot{q}_3 + \dot{q}_4)^2 + 2\dot{q}_1(\dot{q}_3 + \dot{q}_4) + 2\dot{q}_2(\dot{q}_3 - \dot{q}_4))\sin(q_3 + q_4)
\end{aligned}$$

$$\begin{aligned}
H_3 =& a_{23}(\dot{q}_1 + \dot{q}_2)^2\sin q_3 - a_{34}\dot{q}_4(2(\dot{q}_1 + \dot{q}_2 + \dot{q}_3) + \dot{q}_4)\sin q_4 \\
& + a_{13}\dot{q}_1^2\sin(q_2 + q_3) + a_{24}(\dot{q}_1 + \dot{q}_2)^2\sin(q_3 + q_4) \\
& + a_{14}\dot{q}_1^2\sin(q_2 + q_3 + q_4) \\
H_4 =& a_{34}(\dot{q}_1 + \dot{q}_2 + \dot{q}_3)^2\sin q_4 + a_{24}(\dot{q}_1 + \dot{q}_2)^2\sin(q_3 + q_4) \\
& + a_{14}(\dot{q}_1)^2\sin(q_2 + q_3 + q_4)
\end{aligned}$$

the entries of the G vector are

$$\begin{aligned}
G_1 =& - b_1\sin q_1 - b_2\sin(q_1 + q_2) - b_3\sin(q_1 + q_2 + q_3) \\
& - b_4\sin(q_1 + q_2 + q_3 + q_4) \\
G_2 =& - b_2\sin(q_1 + q_2) - b_3\sin(q_1 + q_2 + q_3) - b_4\sin(q_1 + q_2 + q_3 + q_4) \\
G_3 =& - b_3\sin(q_1 + q_2 + q_3) - b_4\sin(q_1 + q_2 + q_3 + q_4) \\
G_4 =& - b_4\sin(q_1 + q_2 + q_3 + q_4)
\end{aligned}$$

and the b_x constants are

$$\begin{aligned}
b_1 =& (m_1\ell_{c1} + (m_2 + m_3 + m_4)\ell_1)g \\
b_2 =& (m_2\ell_{c2} + (m_3 + m_4)\ell_2)g \\
b_3 =& (m_3\ell_{c3} + m_4\ell_3)g \\
b_4 =& m_4\ell_{c4}g
\end{aligned}$$

4-Link Planar Robot Reward

We define the reward for transitioning from state s to state s' by taking action a as $\tilde{\rho}(s, a, s') = E(s') - E(s)$, where $E(s)$ is the total mechanical energy of the system. Intuitively, a policy that drives the tip to the goal is a policy that maximizes the mechanical energy of the system. The total energy is the sum of the kinetic energy K_i and the potential energy U_i of all links:

$$E(s) = \sum_{i=1}^{4} K_i + \sum_{i=1}^{4} U_i$$

To simplify notation we use (Xin and Liu, 2014):

$$\ell_{ij} = \begin{cases} \ell_j, & \text{for } j < i, \\ \ell_{ci}, & \text{for } j = i, \\ 0 & \text{for } j > i \end{cases}$$

$$K_k = \frac{1}{2} J_k \dot{\theta_k}^2 + \frac{1}{2} \sum_{i=1}^{4} \sum_{j=1}^{4} m_k \ell_{ki} \ell_{kj} \cos{(\theta_j - \theta_i)} \dot{\theta_i} \dot{\theta_j}$$

The potential energy is calculated relative to the level $\ell_U = \ell_1 + \ell_2 + \ell_3 + \ell_{c4}$ below rotation point.

$$U_k = m_k g (\sum_{i=1}^{k} \ell_{ki} cos\theta_i + \ell_U)$$

6

Directed Sampling

In this section, we are going to explicate our main contribution in reinforcement learning through classification. In particular, we will show how, given a policy, we identify a subset of selected states, which will be probed to form a training set for the new (improved) policy. For this selection, we use hints from the current, approximate policy representation to focus on critical states, which can potentially lead to policy improvement using fewer computational resources. Before we proceed, let's define how we value states and how we perform classification of actions. Note that from this point on, we consider multi-dimensional, continuous state spaces and therefore states will be denoted as vectors, $\boldsymbol{s}$.

6.1 Action Advantage Function

We identify as important areas for probing, areas in the state space, where changes in action domination take place. Given policy π, we say that an action a *dominates* other actions in some state $\boldsymbol{s}$, when the state-action value function $Q_\pi(\boldsymbol{s}, a)$ for that action is greater that the rest of the actions. It's worth noting here that there may exist more than one dominating actions over large areas of the state space (Rexakis and Lagoudakis, 2008), all of which will share the same value in that area.

A first approach to identify areas in the state space that are worthy of further examination, is to find states that exhibit a difference in $Q_\pi(\boldsymbol{s}, a)$ values for at least two actions. The rationale is that if there is no difference between action values, all actions in that state are equally bad (or equally good) and makes no difference which action is chosen. We define the maximum difference in Q values for state $\boldsymbol{s}$ over all actions in the action as follows:

$$f(\boldsymbol{s}) = \max_{a' \in \mathcal{A}} \left\{ \widehat{Q}_\pi(\boldsymbol{s}, a') \right\} - \min_{a'' \in \mathcal{A}} \left\{ \widehat{Q}_\pi(\boldsymbol{s}, a'') \right\}$$

Then, we define the *action advantage function* $\Delta Q(\boldsymbol{s})$ as follows:

$$\Delta Q(\boldsymbol{s}) = 2 \left(\frac{1}{1 + \exp(-f(\boldsymbol{s}))} - 0.5 \right)$$

Function $\Delta Q(\boldsymbol{s})$ is based on a scaled and shifted sigmoid:

$$S(t) = 2 \left(\frac{1}{1 + \exp(-t)} - 0.5 \right)$$

which is shown in Figure 6.1.

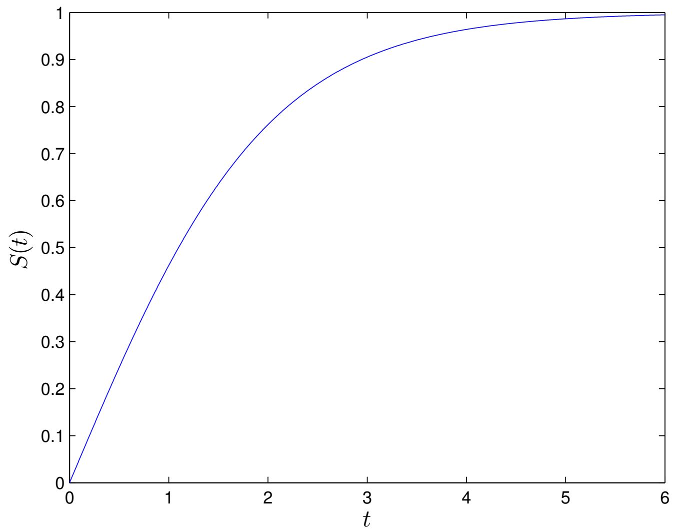


FIGURE 6.1: The scaled and shifted sigmoid function $S(t)$

The maximum difference in Q values for state $\boldsymbol{s}$ is always greater than or equal to zero ($f(\boldsymbol{s}) \geqslant 0$) and, thus, the advantage function $\Delta Q(\boldsymbol{s})$ is in the range $[0, 1)$. $\Delta Q(\boldsymbol{s})$ is used in both our directed exploration proposals, *sample filtering* and *importance sampling*, described later in this chapter.

6.2 Training Set Formation and Filtering Limit

In our work, we represent (approximately) a deterministic policy using a multi-class classifier. To obtain such a classifier-based policy, we need to train a classifier, and therefore we form a training set as follows. We identify a set of candidate states (using two directed sampling methods that will be described later) and we perform rollouts from these states, using the previous policy to estimate $Q(\boldsymbol{s}, a)$ values for all states in the set and for all actions. Then, we calculate the advantage function values $\Delta Q(\boldsymbol{s})$ for all these states. Finally, we form the training set for new the classifier using data only from states with $\Delta Q > \epsilon$, where ϵ is a filtering limit used to clear out noise in action domination. States with $\Delta Q \leqslant \epsilon$ have low action domination probability, are not reliable, and are therefore discarded from the formation of the training set.

Sample filtering

The main objective of filtering is to remove the smaller values of ΔQ, the ones that are close to zero. Let us define A to be the set of all distinct ΔQ values and $A_\epsilon \subseteq A$ be the set of all $\Delta Q \leqslant \epsilon$ values from A, where ϵ is a member of the A set. All states with ΔQ values in A_ϵ are discarded. The filtering limit ϵ is determined as follows:

$$h(\epsilon) = \sum_{q \in A_\epsilon} q - c \sum_{q' \in A} q'$$

$$\epsilon = \arg\min_{\epsilon \in A} \; h(\epsilon), \qquad \text{such that } h(\epsilon) \geqslant 0$$

where $c \in (0, 1)$ is a small positive value. The c value we used for all domains is 10^{-6}. The calculation of the filtering limit is shown in Algorithm 12. In short, this procedure isolates the smaller ΔQ values which add up to a tiny fraction (defined by c) of the total sum of all ΔQ values.

Algorithm 12 Filtering limit calculation

```
Input: $DeltaQ$ : array of $\Delta Q$ values, $c$: small constant
Output: $\epsilon$ : filtering limit

$limit = sum(DeltaQ) * c$
$DeltaQ\_sorted = sort(DeltaQ, AscendingOrder)$
$s = 0$
for each $\epsilon \in DeltaQ\_sorted$
    $s = s + \epsilon$
    if $s > limit$ then
        return $\epsilon$
    end
end
```

6.3 Binary Classifiers

In most reinforcement learning problems, there are more than two action choices in each state and, therefore, the resulting problem in RCPI is a multi-class classification problem. We propose the use of a set of binary classifiers (Rexakis and Lagoudakis, 2008) to represent a policy. Each binary classifier corresponds to one action; each action/classifier is trained *against all others.* If $y^a(\boldsymbol{s})$ is the prediction function of the classifier for action a in state s, we use:

$$\arg\max_{a \in \mathcal{A}} y^a(\boldsymbol{s})$$

to resolve conflicts (select the best action), when many action claim the state. For both SVM and RVM classifiers, we have:

$$y^a(\boldsymbol{s}) = \sum_{i=1}^{m} t_i^a \alpha_i^a \kappa(\boldsymbol{s}_i, \boldsymbol{s}) + b^a$$

While this scheme works well (Scholkopf and Smola, 2001), it is somewhat heuristic. The binary classifiers are trained on different binary classification problems and it is unclear whether their $y^a(\boldsymbol{s})$ values are scaled evenly. However, using class probabilities does not improve multiclass decisions; it only adds one more computational step.

Border in State Space

The main advantage of representing a policy that way is that each classifier separates one action from the rest and defines a clear border within the state space. The alternative method of training each action *against each other* and using voting to resolve conflicts, usually requires more binary classifiers, one classifier for each pair of actions, that is $\binom{|\mathcal{A}|}{2} = \frac{|\mathcal{A}|(|\mathcal{A}|-1)}{2}$ classifiers, instead of $|\mathcal{A}|$ classifiers needed in the *one against all others* case, which implies more time to train and many more separating borders to deal with.

6.4 Directed Policy Search

In this thesis, we propose two methods for directed policy search. Both are based on exploiting the structure of modern classifiers. Support Vector Machines [14; 18; 75] and Relevance Vector Machines [68; 69] share a common prediction function structure that depends only on a selected subset of input vectors, which we consider as *active*[1]. These are chosen by the corresponding training/optimization process to describe the separating border, by assigning non-zero value to their α_i coefficients. It is worth noting here that most of the alpha coefficients are zero, giving a sparse prediction function:

$$y(\boldsymbol{s}) = \sum_{i \in \mathcal{M}} t_i \alpha_i \kappa(\boldsymbol{s}_i, \boldsymbol{s}) + b \tag{6.1}$$

where $\mathcal{M}$ is the set of the active vector indexes in the training set $\{(\boldsymbol{s}_i, t_i)\}_{i=1}^{m}$ and $\{\alpha_i\}_{i=1}^{m}$ the corresponding alpha coefficients. For convenience, we define the active vectors tuple set $\left\{(\dot{\boldsymbol{s}}_i, \dot{t}_i, \dot{\alpha}_i)\right\}_{i=1}^{G}$, with $\dot{\alpha}$ being the non-zero α coefficients and $G = |\mathcal{M}|$ being the number of active vectors. Therefore, the prediction function becomes:

$$y(\boldsymbol{s}) = \sum_{i=1}^{G} \dot{t}_i \dot{\alpha}_i \kappa(\dot{\boldsymbol{s}}_i, \boldsymbol{s}) + b \tag{6.2}$$

These active input vectors hold significant “positions” in the state space. This observation gave rise to the idea that, if these classifiers are used to represent policies in RCPI, then there should be a way to guide the selection of the next set of rollout states around the action boundaries of the current policy.

[1] Active input vectors are the support vectors in SVMs and the relevance vectors in RVMs. They are the input vectors that have non-zero corresponding α coefficients in the decision function (6.1).

We use an ensemble of binary classifiers, one for each action to represent a deterministic policy. Our analysis focuses on the C-SVM and RVM classifiers. Given a set of training data $\{(\boldsymbol{s}_i, t_i)\}_{i=1}^{m}$ for a certain action a, where $\boldsymbol{s}_i$ is an N-dimensional input vector (a state) and $t_i \in \{-1, +1\}$ is a class identifier (action a chosen or not in that state), an $N-1$ dimensional surface is constructed by the decision function (6.2) of the classifier in the input (state) space to separate the two classes. Any point $\boldsymbol{s}$ in the input space can then be classified using equation (6.2). If $y(\boldsymbol{s})$ yields a positive number, $\boldsymbol{s}$ is classified in the $+1$ class and the action a claims state $\boldsymbol{s}$, otherwise it is classified in the -1 class and state $\boldsymbol{s}$ is left to be claimed by other actions.

6.4.1 Policy Evaluation

Since our policies are represented by classifiers, classification accuracy is necessary for reliability. However, this is not necessarily related to reinforcement learning performance. Therefore, performance of a learned policy is evaluated in terms of expected total discount reward. This kind of policy evaluation is achieved through rollout testing: we start a number of independent trajectories of several steps each, from states drawn from the initial state distribution to obtain empirical averages in all tested domains. This ensures us that we keep optimizing the reinforcement learning goal.

6.4.2 Directed Policy Search using Active Input Vectors (DRCPI-AIV)

We use binary classifiers to represent policies and the state space is the input space for the classifier. Active input vectors are some key states found in the training set and define the boundary (6.2) between different action choices. As discussed above, an attempt to improve the currently represented policy will certainly have to probe the states around this boundary. To perform state resampling, we initially draw the line that goes through an active input vector and is perpendicular to the separating boundary defined by the classifier. The normal to the boundary for point $\boldsymbol{s}$ in the input space has a direction given by (Riley et al., 2006):

$$\boldsymbol{g_d}(\boldsymbol{s}) = \nabla_{\boldsymbol{s}} y(\boldsymbol{s}) = \nabla_{\boldsymbol{s}} \left(\sum_{i=1}^{G} \dot{t}_i \dot{\alpha}_i \kappa(\dot{\boldsymbol{s}}_i, \boldsymbol{s}) + b \right)$$

For the gaussian kernel (radial basis functions)

$$\kappa_{RBF}(\boldsymbol{s}', \boldsymbol{s}) = \exp\big(-\beta \|\boldsymbol{s}' - \boldsymbol{s}\|^2 \big), \quad \beta > 0 \tag{6.3}$$

we have

$$\boldsymbol{g_d}(\boldsymbol{s}) = 2\beta \sum_{i=1}^{G} \dot{t}_i \dot{\alpha}_i (\dot{\boldsymbol{s}}_i - \boldsymbol{s}) \kappa_{RBF}(\dot{\boldsymbol{s}}_i, \boldsymbol{s}) \ .$$

Now that we have the direction, we seek to find the projections of the support vectors on the separating boundary, given that the support vectors lie at critical areas along the edge. In particular, for each active vector $\dot{\boldsymbol{s}}_i$, we seek a point $\boldsymbol{u}_i$ (Figure 6.2) that satisfies the following two conditions:

$$\begin{aligned} \lambda_i \boldsymbol{g_d}(\boldsymbol{u}_i) + \boldsymbol{u}_i - \dot{\boldsymbol{s}}_i &= 0 \\ \sum_{j=1}^{G} t_j \dot{\alpha}_j \kappa(\dot{\boldsymbol{s}}_j, \boldsymbol{u}_i) + b &= 0 \end{aligned}$$

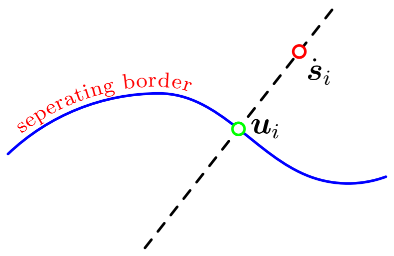


FIGURE 6.2: Projection $\boldsymbol{u}_i$ of active input vector $\dot{\boldsymbol{s}}_i$ onto separating border

For each support vector, this is a system of $N + 1$ non-linear equations with unknowns $\boldsymbol{u}_i$ (N-dimensional vector) and λ_i (scalar) (Riley et al., 2006). An efficient arithmetic solution to this system can be given by Powell's hybrid algorithm (Powell, 1970) using $[\boldsymbol{s}_i, 0]^T$ as an initial guess for $[\boldsymbol{u}_i, \lambda_i]^T$. Solutions $\boldsymbol{u}_i$ that fall outside the input space are discarded. Given a valid $\boldsymbol{u}_i$, the unit (normalized) vector $\boldsymbol{g_{d_i}}$ at point $\boldsymbol{u}_i$ which is perpendicular to the separating boundary is $\boldsymbol{g_{d_i}} = \boldsymbol{g_d}(\boldsymbol{u}_i)/\|\boldsymbol{g_d}(\boldsymbol{u}_i)\|$. A new input point $\boldsymbol{z}_i(d)$ along this line at distance d from $\boldsymbol{u}_i$ will be $\boldsymbol{z}_i(d) = \boldsymbol{u}_i + d\boldsymbol{g_{d_i}}$, where $d \in \mathbb{R}$.

Let k be the policy iteration number. Given the above derivation and any policy π_k represented as an SVM or RVM classifier with active vectors $\{\dot{\boldsymbol{s}}_i^k\}_{i=1}^{G^k}$, it is straightforward to select a new set of states around the separating boundary to probe with rollouts. We cover a zone wide enough to accommodate possible mistakes of the next policy in identifying a more precise border between different action choices. First, we define the locus for the centers used for Gaussian resampling to be the two parallel hyper surfaces to the border, one on each side, shown with a red line and a green line in Figure 6.3. These surfaces have a dimensionality of $N-1$ and lie at distance $\overline{d^k}$ from the separating border, where $\overline{d^k}$ is the average distance of all active vectors $\boldsymbol{s}_i^k$ from the border. Then, we perform resampling of rollout states from an N-dimensional Gaussian, centered on the intersection of the perpendicular line that passes through the active vector $\boldsymbol{s}_i^k$ and the two parallel surfaces, with covariance matrix $\Sigma = \overline{d^k} I$, where I is the N-dimensional unit matrix. The total number of rollout states is M, the resampling size, and they are equally distributed to all active vectors, that is m_i points per active vector, satisfying the condition $\sum_{i=1}^{G^k} m_i^k = M$. More specifically, the next set of rollout states (probes) is:

$$S_{k+1} = \{\boldsymbol{s}_{i,j} \sim \mathcal{N}(\boldsymbol{u}_i^k + (-1)^j \overline{d^k} \boldsymbol{g}_{\boldsymbol{d}_i}^k), \overline{d_i^k} I) : i = 1 \dots G_k, j = 1 \dots m_i^k\} \tag{6.4}$$

where i is an index over active vectors and j is an index over samples per active vector. Note that, for each active vector, half the samples are taken on one side and half on the other, based on the odd or even value of j. The proposed DRCPI-resampling procedure is depicted in Figure 6.3.

Note that the first iteration begins with a purely random, but deterministic, policy and therefore S_1 cannot be formed in the way described above, since there is no classifier to represents π_0. The set S_1 with size U is simply a uniformly random selection of states from the entire state space to guarantee full coverage at the beginning[2].

[2] Without domain knowledge, uniform sampling is the only option to ensure that no part

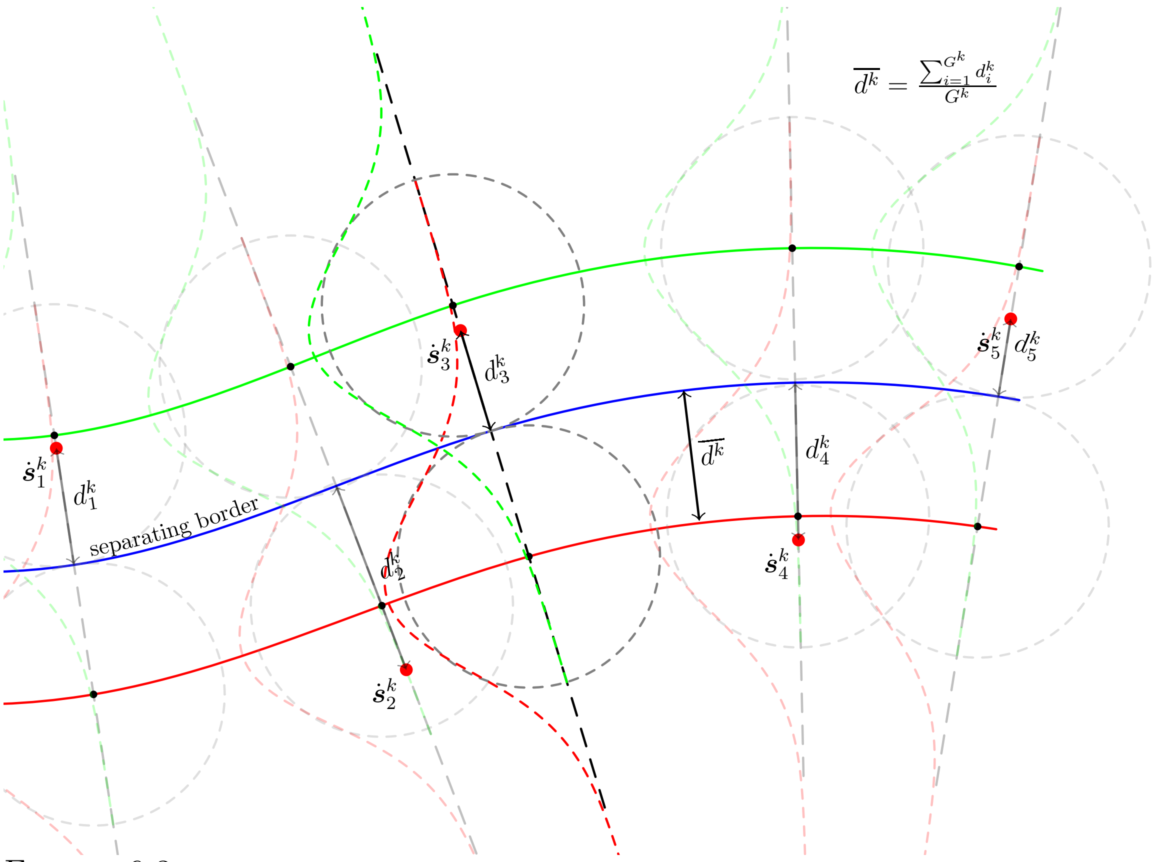


FIGURE 6.3: DRCPI-Resampling: using active input vectors from the k-th iteration to select a new set of states for the $(k+1)$-th iteration

After the selection of rollout states, we have to form a training set for each action to train the corresponding binary classifier for the improved policy. Probing a particular state $\boldsymbol{s}$ for the improved policy over a base policy π boils down to estimating the Q_π values for all actions in that state using rollouts and identifying the dominating action(s) (if any). Say that these estimated values are $\widehat{Q}_\pi(\boldsymbol{s}, a)$, $a \in \mathcal{A}$. To include state $\boldsymbol{s}$ in some training set, we need to identify at least one dominating action, whose value significantly exceeds the value of some other action. To quantify this difference we use the action advantage function $\Delta Q(\boldsymbol{s})$.

If $\Delta Q(\boldsymbol{s}) > \epsilon$, then any action $a^* = \arg\max_{a' \in \mathcal{A}}\{\widehat{Q}_\pi(\boldsymbol{s}, a')\}$ that maximizes $\widehat{Q}_\pi(\boldsymbol{s}, a')$ in state $\boldsymbol{s}$ is considered *dominating* and a pair $(\boldsymbol{s}, a^*)$ is inserted in the training set for the classifier of action a^* as a positive $(+)$ example; all

will remain initially unexplored. In particular cases, domain knowledge could be used to perform a focused non-uniform sampling.

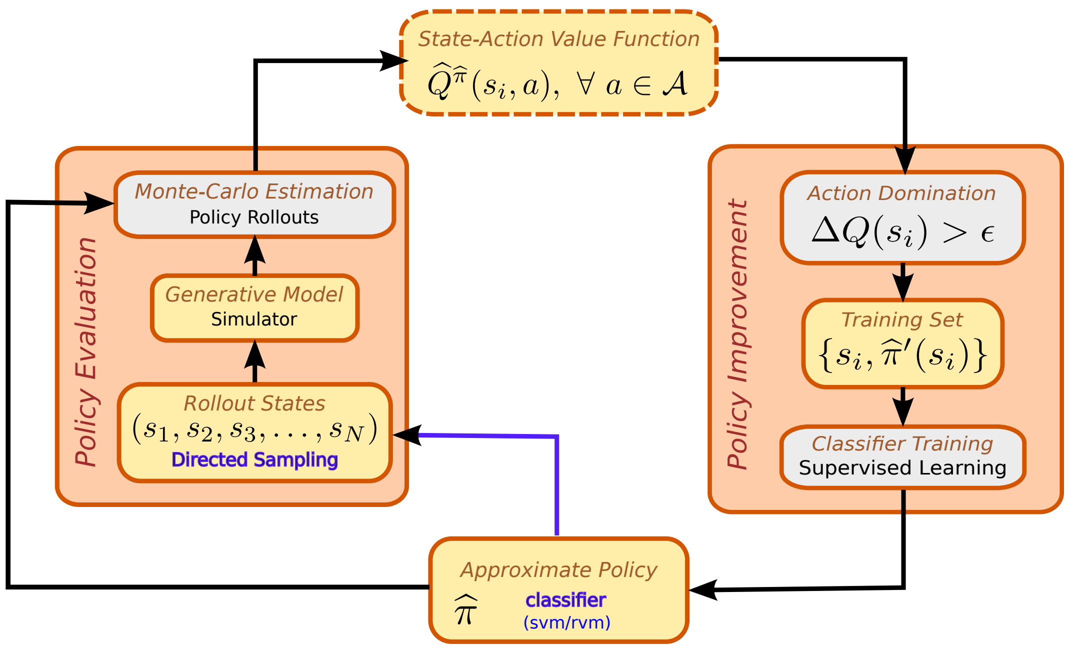


FIGURE 6.4: Directed RCPI with Active Input Vectors (DRCPI-AIV)

other actions $a \neq a^*$, whose value is significantly less (ϵ) than the value of a^*, are considered *dominated* and, for each one of them, a pair $(\boldsymbol{s}, a)$ is inserted in the training set for action a as a negative $(-)$ example. Note that some actions may be neither dominating, nor dominated; no training data will be produced for such actions. States with $\Delta Q(\boldsymbol{s}) \leqslant \epsilon$ do not yield any training data at all. This simple dominance criterion is sufficient in most cases for dealing with estimation noise.

We should stress that our approach requires the estimation of action values at few isolated points (rollout states) only; nowhere does it need a full value function over the entire state-action space. Therefore, the known issues of value function approximation are not applicable. These required values are obtained as unbiased estimates from Monte-Carlo simulation (policy rollouts) and can be estimated to any desired accuracy provided sufficient simulation time.

To summarize, the complete algorithm is shown graphically in Figure 6.4. Given a policy π_k at iteration k represented as an SVM/RVM classifier, the active input vectors in π_k are used to select the new subset of states S_{k+1} at which the improved policy π_{k+1} will be probed. Each probe will yield pairs of data in the training set T_{k+1}, if domination is detected. Once the training set is formed, a new classifier is trained to represent π_{k+1} and the process repeats from the beginning. Clearly, there is no guarantee for monotonic policy improvement in this iterative scheme, however the chances can be increased by repeating some iteration if no improvement was achieved; since each iteration is randomized, it is likely that another run may produce better results. Therefore, if policy π_{k+1} is not better than π_k (tested through simulation), iteration k is repeated for a maximum of L attempts until an improved policy is found. If all attempts are exhausted without improvement, the algorithm terminates. The entire Directed RCPI (DRCPI) algorithm for policy search utilizing Active Input Vectors (AIV) for guiding the search, called DRCPI-AIV, is shown in Algorithm 13.

Algorithm 13 Directed RCPI-AIV (DRCPI-AIV)

```
Input: policy $\pi_0$, attempts $L$, trials $K$, horizon $H$, size $U$, points $M$

$k = -1$                                                    ($policy\ iteration$)
repeat
  $k = k + 1$
  $l = 0$                                                   ($repeated\ attempts$)
  repeat
    $l = l + 1$
    if ($k = 0$) then
      $S_{k+1}$ =a uniformly random subset of $\mathcal{S}$ of size $U$ ($uniform\ sampling$)
    else
      {directed sampling}
      $\{\dot{\boldsymbol{s}}_i^k\}_{i=1}^{G^k} \leftarrow$ active input vectors in $\pi_k$
      $\{\boldsymbol{u}_i^k\}_{i=1}^{G^k} \leftarrow$ projections of $\dot{\boldsymbol{s}}_i^k$ on boundary
      $\{\boldsymbol{g_d}_i^k\}_{i=1}^{G^k} \leftarrow$ perpendicular directions at $\boldsymbol{u}_i^k$
      $\{d_i^k\}_{i=1}^{G^k} \leftarrow$ distances of $\dot{\boldsymbol{s}}_i$ from the boundary
      $\overline{d^k} \leftarrow$ average distance of of $\dot{\boldsymbol{s}}_i$ from the boundary
      $\{m_i^k\}_{i=1}^{G^k} \leftarrow$ num of samples, $\sum_{i=1}^{G^k} m_i^k = M$
      $S_{k+1} = \{\boldsymbol{s}_{i,j} \sim \mathcal{N}(\boldsymbol{u}_i^k + (-1)^j \overline{d^k} \boldsymbol{g_d}_i^k), \overline{d_i^k} I) : i = 1 \ldots G_k, j = 1 \ldots m_i^k\}$
    end
    $T_{k+1} = \varnothing$                                  ($initialization\ of\ training\ sets$)
    for (each $\boldsymbol{s} \in S_{k+1}$)
      for (each $a \in \mathcal{A}$)
        estimate $Q_{\pi_k}(\boldsymbol{s}, a)$ using $K$ rollouts of length $H$    ($simulation$)
      end
      $\epsilon$ = FilteringLimit($\Delta Q(\boldsymbol{s})\ \forall\ \boldsymbol{s} \in S_{k+1}$)
      if ($\Delta Q(\boldsymbol{s}) > \epsilon$) then
        $T_{k+1} = T_{k+1} \cup \{(\boldsymbol{s}, a^*)^+\}$        ($for\ dominating\ actions\ \alpha^*$)
        $T_{k+1} = T_{k+1} \cup \{(\boldsymbol{s}, a)^-\}$          ($for\ dominated\ actions\ \alpha$)
      end
    end
    $\pi_{k+1}$ = TrainClassifiers($T_{k+1}$)
  until ( ($\pi_{k+1}$ is better than $\pi_k$) or ($l = L$) )  ($end\ of\ repeated\ attempts$)
until ($\pi_{k+1}$ is not better than $\pi_k$)              ($end\ of\ policy\ iteration$)
return $\pi_k$
```

6.4.3 Directed Policy Search using Importance Sampling (DRCPI-IS)

In this section, we strive to identify important areas for probing by exploiting, this time, the action advantage function ΔQ and a particle-filter-like resampling procedure. Once again, we use rollouts to estimate state-action values $Q(\boldsymbol{s}, a)$ and $\Delta Q(\boldsymbol{s})$ in selected states $\boldsymbol{s}$.

Importance sampling

Important areas of the state space do not lie strictly on the policy boundaries.

Consider the case of only two actions; the advantage function $\Delta Q(\boldsymbol{s})$ is either zero along the border or some kind of discontinuity occurs at the border. Intuitively, the border itself is not so important; however, the area around the border, where ΔQ *is significant* and *changes fast* is quite important as it indicates a potential policy change. In order to identify these areas of significance, we use the *importance function*, which is a function based on the product $\Delta Q(\boldsymbol{s}) \cdot \|\nabla \Delta Q(\boldsymbol{s})\|_2$, where $\nabla \Delta Q(\boldsymbol{s})$ is the gradient vector of $\Delta Q(\boldsymbol{s})$. Note that this product takes high value, only if the value of $\Delta Q(\boldsymbol{s})$ is high *and* its gradient is high. Intuitively, the importance function offers a quantitative measure of potential action change and, therefore, can be used to characterize important parts of the state space.

But, where can one find this gradient vector, when even $\Delta Q(\boldsymbol{s})$ itself is not analytically known? Our approach towards estimating this gradient vector is a simple solution that takes advantage of the already available information to minimize overhead. At the end of each iteration, the estimated values of $\Delta Q(\boldsymbol{s})$ that were used to form the training set T_{k+1} for the next classifier(s) are still available. We use these values to form another training set T'_{k+1} of $\big(\boldsymbol{s}, \Delta Q(\boldsymbol{s})\big)$ pairs in order to generalize and approximate the function $\Delta Q(\boldsymbol{s})$ over the entire state space by regression. Since differences in $Q(\boldsymbol{s}, a)$ are not important for ΔQ estimation, we do not use filtering here. We train an SVM/RVM regressor on these data, not only because we seek to exploit the same technology and benefit from the advantages it offers, but also, more importantly, because the learned function can be *analytically differentiated* to yield the desired gradient vector. Specifically, the SVM/RVM regressor approximation of the action advantage function $\Delta Q(\boldsymbol{s})$ is

$$q(\boldsymbol{s}) = \sum_{i=1}^{G} \dot{\alpha}_i \Delta Q(\dot{\boldsymbol{s}}_i) \kappa(\dot{\boldsymbol{s}}_i, \boldsymbol{s}) + b$$

where $\{\dot{\boldsymbol{s}}_i\}_{i=1}^{G}$ are the active vectors of the regressor, $\Delta Q(\dot{\boldsymbol{s}}_i)$ are their target values from the corresponding training pairs, and $\dot{\alpha}_i$, b are the parameters of the SVM/RVM regressor.

For the Gaussian (radial basis functions) kernel[3],

$$\kappa_{RBF}(\boldsymbol{s}', \boldsymbol{s}) = \exp\big(-\beta\|\boldsymbol{s}' - \boldsymbol{s}\|^2\big), \quad \beta > 0, \tag{6.5}$$

the gradient of $q(\boldsymbol{s})$ can be analytically derived as

$$\boldsymbol{g_q}(\boldsymbol{s}) = 2\beta \sum_{i=1}^{G} \dot{\alpha}_i \Delta Q(\dot{\boldsymbol{s}}_i) \kappa_g(\dot{\boldsymbol{s}}_i, \boldsymbol{s})(\dot{\boldsymbol{s}}_i - \boldsymbol{s})$$

Thus, we estimate the desired gradient vector, practically at no additional cost, other than a single SVM/RVM regression, since training data are already available. Let $Sp \in \mathcal{S}$ be the set of rollout states for the current iteration. The approximation of $\Delta Q(\boldsymbol{s}) \cdot \|\nabla \Delta Q(\boldsymbol{s})\|_2$ is given by $q(\boldsymbol{s}) \cdot \|\boldsymbol{g_q}(\boldsymbol{s})\|_2$. We define our *state importance function* as follows:

$$importance(\boldsymbol{s}) = \log\left(q(\boldsymbol{s}) \cdot \|\boldsymbol{g_q}(\boldsymbol{s})\|_2 - \min_{\boldsymbol{s}' \in Sp}\Big(q(\boldsymbol{s}') \cdot \|\boldsymbol{g_q}(\boldsymbol{s}')\|_2\Big) + 1\right)$$

The logarithm function $\log(x)$ is positive for values of x greater than one. ΔQ is greater than or equal to zero by definition, but it's estimation $q(\boldsymbol{s})$ may slip to the negative side of reals, due to approximation errors. Therefore, to assure that the importance function is non-negative, the transformation $\big(q(\boldsymbol{s})\cdot\|\boldsymbol{g_q}(\boldsymbol{s})\|_2 - \min_{s' \in Sp}(q(\boldsymbol{s}')\cdot\|\boldsymbol{g_q}(\boldsymbol{s}')\|_2)+1\big)$ is used to shift all approximate values $q(\boldsymbol{s}) \cdot \|\boldsymbol{g_q}(\boldsymbol{s})\|_2$ above one. The use of the logarithm allows the $importance(\boldsymbol{s})$ function to retain significance within comparable values for several orders of magnitude of $q(\boldsymbol{s}) \cdot \|\boldsymbol{g_q}(\boldsymbol{s})\|_2$ (Figure 6.5).

The next obstacle we have to overcome is the identification of areas of the state space where $importance(\boldsymbol{s})$ values are large and direct our sampling of rollout states to those areas. Despite the availability of the estimation of the gradient in closed form, an analytical solution for the maxima poses a hard nonlinear problem. Additionally, we have to take into account that the set of rollout states becomes more and more focused over iterations and, therefore,

[3] Note that other kernels can be used as long as they can be differentiated.

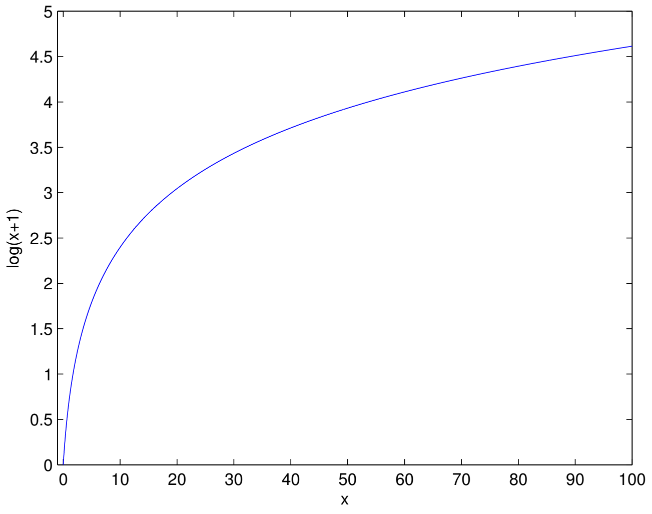


FIGURE 6.5: $\log(x+1)$ vs x graph

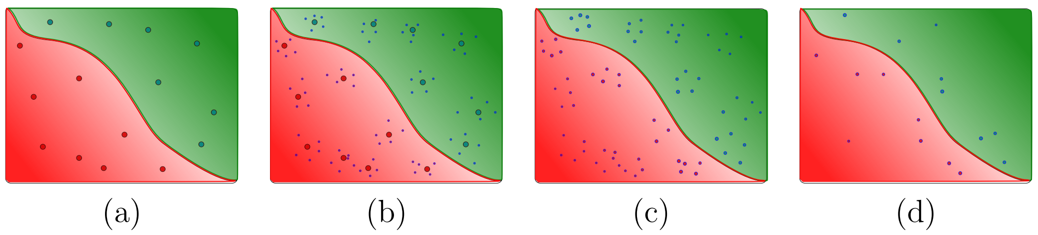


FIGURE 6.6: Directed sampling of rollout states: (a) rollout states of previous policy and class boundary, (b) generation of candidate rollout states for next policy (particles) around the previous states, (c) weighting of generated particles according to the importance function (size of particle proportional to weight), and (d) resampling of particles (more selection chances with higher weight)

the SVM/RVM regressor ends up being trained on points representing only a small part of the state space. As a result, the regressor function cannot be trusted in areas away from the rollout states of the previous iteration. To address these problems, we apply a resampling procedure inspired by the resampling operations in a particle filter (Doucet et al., 2001).

Initially, we sample a large number of candidate states (particles), by repeating the following: pick uniformly in random one point from the current unfiltered training set T' and add to it zero-mean normal noise to obtain a new sample state from its neighborhood. This step ensures that all candidate rollout states lie in areas where the regressor can be trusted. Next, to weigh each particle, we are using the $importance(\boldsymbol{s})$ function defined above. Finally, we apply a resampling procedure to keep only the desired number of rollout states for the

Algorithm 14 ImportanceSampling: Sampling Rollout States

```
Input: number of samples $M$, number of particles $Z$, regressor training set $T'$, covariance matrix $\Sigma$, $\Delta Q$ estimation function $q(\boldsymbol{s})$, $\nabla \Delta Q$ estimation function $\boldsymbol{g_q}(\boldsymbol{s})$

for $j = 1$ to $Z$
    sample $\big(\boldsymbol{s}, \Delta Q(\boldsymbol{s})\big) \sim T'$ uniformly        (sample state)
    $\boldsymbol{s}_j = \boldsymbol{s} + \mathcal{N}(\boldsymbol{0}, \Sigma)$        (add noise)
    $w_j = q(\boldsymbol{s}_j)\|\boldsymbol{g_q}(\boldsymbol{s}_j)\|_2$        (weigh state)
end
$\boldsymbol{w} = \log(\boldsymbol{w} - \min(\boldsymbol{w}) + 1)$        (importance function)
$\boldsymbol{w} = \boldsymbol{w}/\|\boldsymbol{w}\|_2$        (normalize weights)

{particle resampling}
$S = \varnothing$, $r \sim \text{uniform}(0, M^{-1})$, $c = w_1$, $j = 1$
for $m = 1$ to $M$
    $u = r + (m-1) \times M^{-1}$
    while $u > c$
        $j = j + 1$
        $c = c + w_j$
    end while
    $S = S \cup \{\boldsymbol{s}_j\}$
end
return $S$
```

next iteration. This particle-filtering-like resampling ensures that only those candidate states with significant weight are promoted. The entire procedure is illustrated by a simple example in Figure 6.6 and is shown in Algorithm 14. Note that its time complexity is only $\mathcal{O}(Z)$, where Z is the number of particles. If Z is taken to be a multiple of M, the number of rollout states in each iteration, the cost of obtaining the next set of rollout states is only linear in its size.

Now, we can summarize the entire algorithm, which is graphically shown in Figure 6.7. Given a policy π_k at iteration k represented as an SVM/RVM classifier, the corresponding SVM/RVM regressor is used to select a new focused subset of states S_{k+1} at which the improved policy π_{k+1} will be probed. Each probe will yield pairs of data in the training set T_{k+1} for the next classifier, if domination is detected, and a pair of data in the training set T'_{k+1} of the corresponding regressor. Once the training sets are formed, a new classifier and a new regressor are trained to represent π_{k+1} and the process repeats from the beginning.

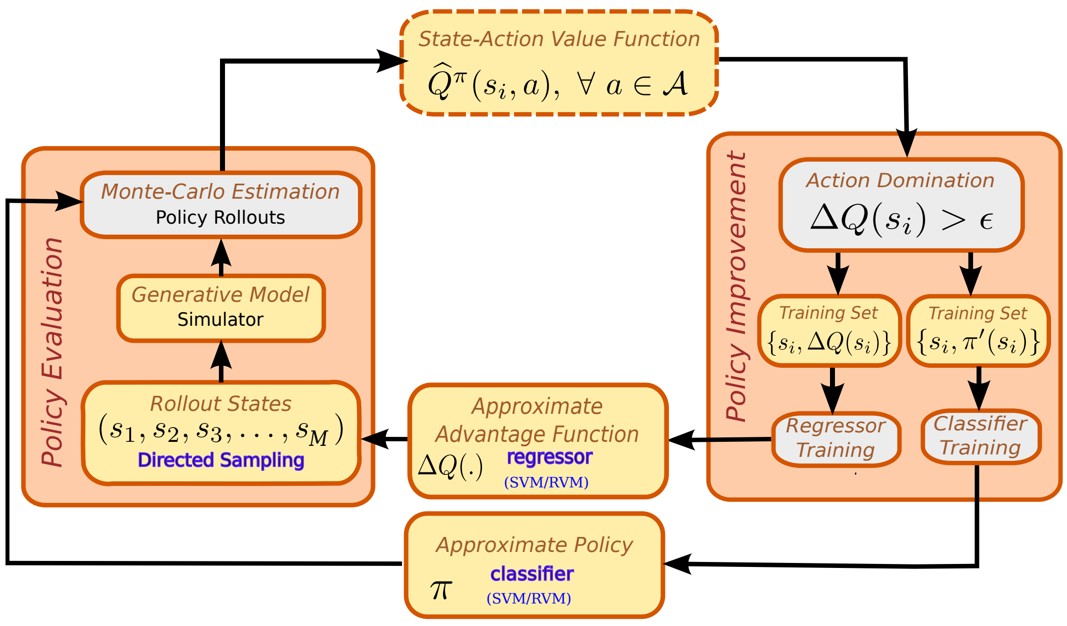


FIGURE 6.7: Directed Policy Search using Importance Sampling

There is no guarantee for monotonic policy improvement in this iterative scheme. However, the chances can be increased by attempting to repeat some iteration, if no improvement was achieved; since each iteration is randomized, it is likely that another attempt may produce better results. Therefore, if policy π_{k+1} is not better than π_k (tested through simulation), iteration k is repeated for a maximum of L attempts until an improved policy is found. If all attempts are exhausted without improvement, the algorithm terminates. Note that the ΔQ estimation regressor is trained only when the policy improves, that is, only once per iteration; it should not change, if an improvement attempt fails. Note also that the first iteration begins with a purely random, but deterministic, policy and therefore S_1 cannot be formed in the way described above, since there is no classifier and regressor for π_0. S_1 is simply a uniformly random selection of states from the entire state space to guarantee full coverage at the beginning. Without domain knowledge, uniform sampling is the only option to ensure that no part of the state space will remain initially unexplored. In particular cases, domain knowledge could be used to perform a focused non-uniform sampling over the state space. The entire Directed RCPI (DRCPI) algorithm for policy search utilizing Importance Sampling (IS) for guiding the search, called DRCPI-IS, is shown in Algorithm 15.

Algorithm 15 DRCPI-IS: Directed RCPI using Importance Sampling

```
Input: policy $\pi_0$, attempts $L$, trials $K$, horizon $H$, size $U$, points $M$, particles $Z$, covariance matrix $\Sigma$

$k = -1$                                                            (*policy iteration*)
repeat
  $k = k + 1$
  $l = 0$                                                           (*repeated attempts*)
  repeat
    $l = l + 1$
    if ($k = 0$) then
      $S_{k+1}$ = a uniformly random subset of $\mathcal{S}$ of size $U$          (*uniform sampling*)
    else
      $S_{k+1} = \textsc{ImportanceSampling}(M, Z, T'_{k+1}, \Sigma, \boldsymbol{g}_{\boldsymbol{q}_{k+1}})$   (*directed sampling*)
    end
    $T_{k+1} = \varnothing$, $T'_{k+1} = \varnothing$                   (*initialization of training sets*)
    for (each $\boldsymbol{s} \in S_{k+1}$)
      for (each $a \in \mathcal{A}$)
        estimate $Q_{\pi_k}(\boldsymbol{s}, a)$ using $K$ rollouts of length $H$      (*simulation*)
      end
      $\epsilon = \text{FilteringLimit}(\Delta Q(\boldsymbol{s})\ \forall \boldsymbol{s} \in S_{k+1})$
      if ($\Delta Q(\boldsymbol{s}) > \epsilon$) then
        $T_{k+1} = T_{k+1} \cup \{(\boldsymbol{s}, a^*)^+\}$              (*for dominating actions $a^*$*)
        $T_{k+1} = T_{k+1} \cup \{(\boldsymbol{s}, a)^-\}$               (*for dominated actions $a$*)
      end
      $T'_{k+1} = T'_{k+1} \cup \big\{\big(\boldsymbol{s}, \Delta Q(\boldsymbol{s})\big)\big\}$         (*regressor example*)
    end
    $\pi_{k+1} = \textsc{TrainClassifiers}(T_{k+1})$                  (*classifier learning*)
  until $\big($ ($\pi_{k+1}$ is better than $\pi_k$) or ($l = L$) $\big)$   (*end of repeated attempts*)
  $q_{k+1} = \textsc{TrainRegressor}(T'_{k+1})$                      (*regressor learning*)
  $\boldsymbol{g}_{\boldsymbol{q}_{k+1}} = \textsc{Differentiate}(q_{k+1})$                (*differentiation*)
until ($\pi_{k+1}$ is not better than $\pi_k$)                     (*end of policy iteration*)
return $\pi_k$
```

7

Experimental Results

Our experiments with the proposed methodologies were performed in two areas. The first area is the computational approximation and visualization of optimal policies for the two-dimensional domains, namely the Inverted Pendulum and the Mountain Car. The third domain, the acrobot (four-dimensional), and the fourth domain, the 4-Link Planar Robot (eight-dimensional), cannot be easily visualized. Our experiments yield evidence that indeed optimal policies exhibit significant structure. The second area is the experimental analysis of our directed exploration algorithms. All four domains were utilized in this case; our proposed algorithms were tested against the original RCPI algorithm that receives no guidance during the process of exploring the probes for the improved policy. Our results indicate that indeed our proposed algorithms allow for efficient exploration of policy space using guidance derived from the policy representation itself and thus learning of the same task in less time.

7.1 Optimal Policy Structure

Our first goal is to uncover the structure of optimal policies for each domain. Even though the model of the underlying MDP is known, applying an exact algorithm for solving the MDP and obtaining a truly optimal policy is infeasible due to the continuous nature of the state space. Instead, we used a fine discretization of the two-dimensional state space into a uniform grid, a large number of (s, a, r, s') samples at each discrete tile (s, a), and the Least-Squares Policy Iteration (LSPI) algorithm (Lagoudakis and Parr, 2003b) with indicator basis functions over the state grid and all actions, to converge to a near-optimal policy. Notice that the only source of suboptimality in this procedure is the resolution of the discretization itself, as well as the number of samples collected at each point; there is no error due to the approximation of the value function or the policy. The error due to discretization cannot be avoided. However, the error due to sampling could be practically eliminated by a large number of samples. Alternatively, one could analytically determine the transition model over the state grid, that is, the possible next states and the related transition probabilities at each tile of the grid. Despite the theoretical feasibility of such a difficult task, we chose to use a sampling approach instead, to accommodate any change in the system (different levels of control noise, simulation step, discretization resolution, parameter values, etc.) without changes. Finally, in deriving an optimal policy from the resulting optimal value function, the action values were compared within an ϵ-margin to eliminate small numerical errors. The settings we used were: a grid of 250×250 tiles with 25 uniform samples per tile for each action in the deterministic versions, a grid of 250×250 tiles with 500 uniform samples per tile for each action in the stochastic versions, and $\epsilon = 10^{-5}$.

7.1.1 Inverted Pendulum

The structure of an optimal policy for the stochastic Inverted Pendulum problem over the two-dimensional state space is shown in Figure 7.1. The horizontal axis is the angle θ of the pendulum ranging from $-\pi/2$ to $\pi/2$ and the vertical axis is the angular velocity $\dot{\theta}$ ranging from -6 to 6. The point of full balance is the point $(0, 0)$ in the middle of the state space.

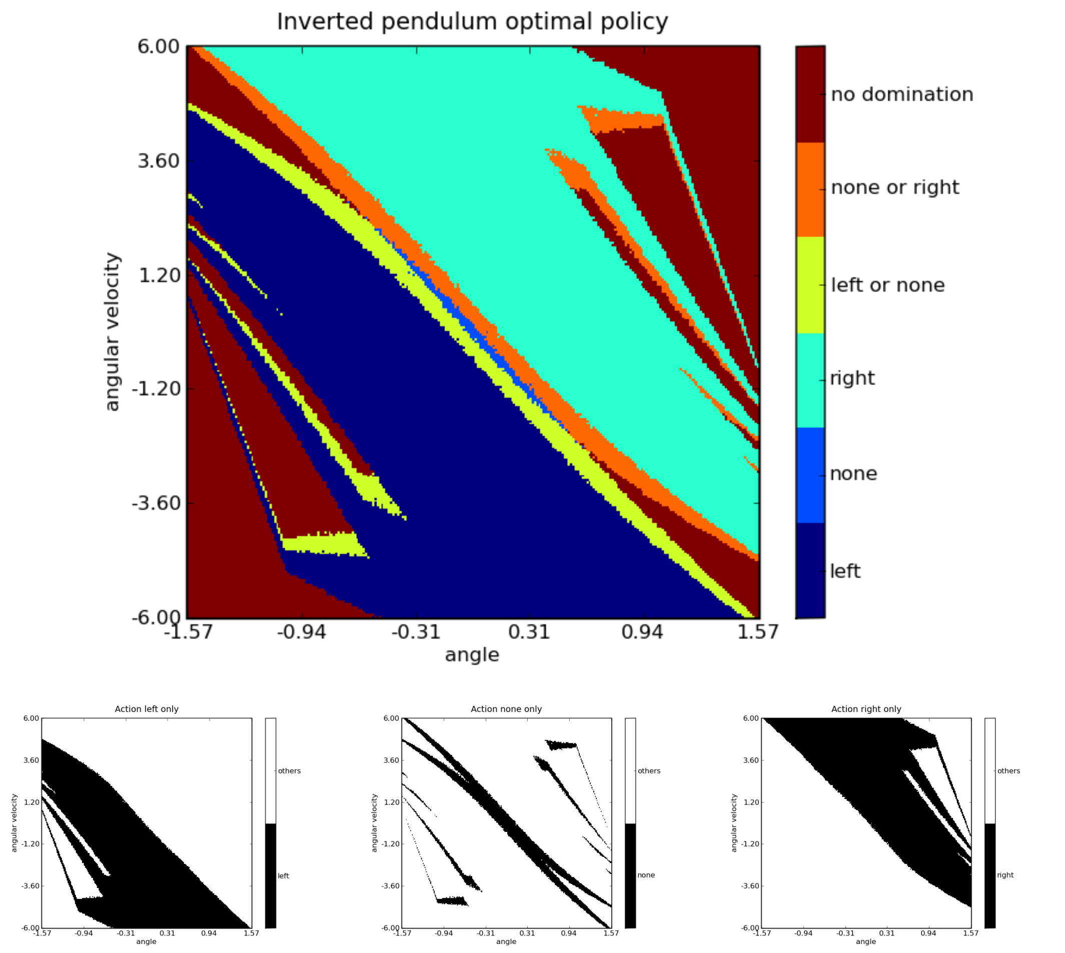


FIGURE 7.1: Optimal policy for the stochastic Inverted Pendulum. Top: dominating action loci including tied cases. Bottom: individual (non-pure) domination locus for each action.

One can clearly identify large areas where a single action consists the best choice. In general, as soon as the angle θ becomes positive enough, the best action choice is to apply a right force. This is also true when the angular velocity $\dot{\theta}$ takes on positive values, even though the angle θ itself might be negative; a right force will proactively prevent the pendulum from falling on the right side. Of course, beyond some values of θ and $\dot{\theta}$ any action is hopeless; the pendulum is doomed to falling. Since the problem is symmetric, similar action choices appear on the left side of the state space. Notice that there is a small area around the balancing point where the best action to perform is to apply no force at all and a small zone around this area where the left or no force actions are equally good, and the right or no force actions are equally good. As expected, there are no areas where the left and right force actions are equally good simultaneously. Apparently, the optimal policy bears sufficient structure to facilitate the classification problem. Especially, the critical area around the balancing point poses no difficulties in classifying the states correctly to the corresponding actions. The thin stripes on the left and right sides are not intuitive and may be challenging from a classification viewpoint. However, a good enough policy will most likely prevent the pendulum from ever reaching those areas. Therefore, the performance loss from misclassification in those areas will be minimal.

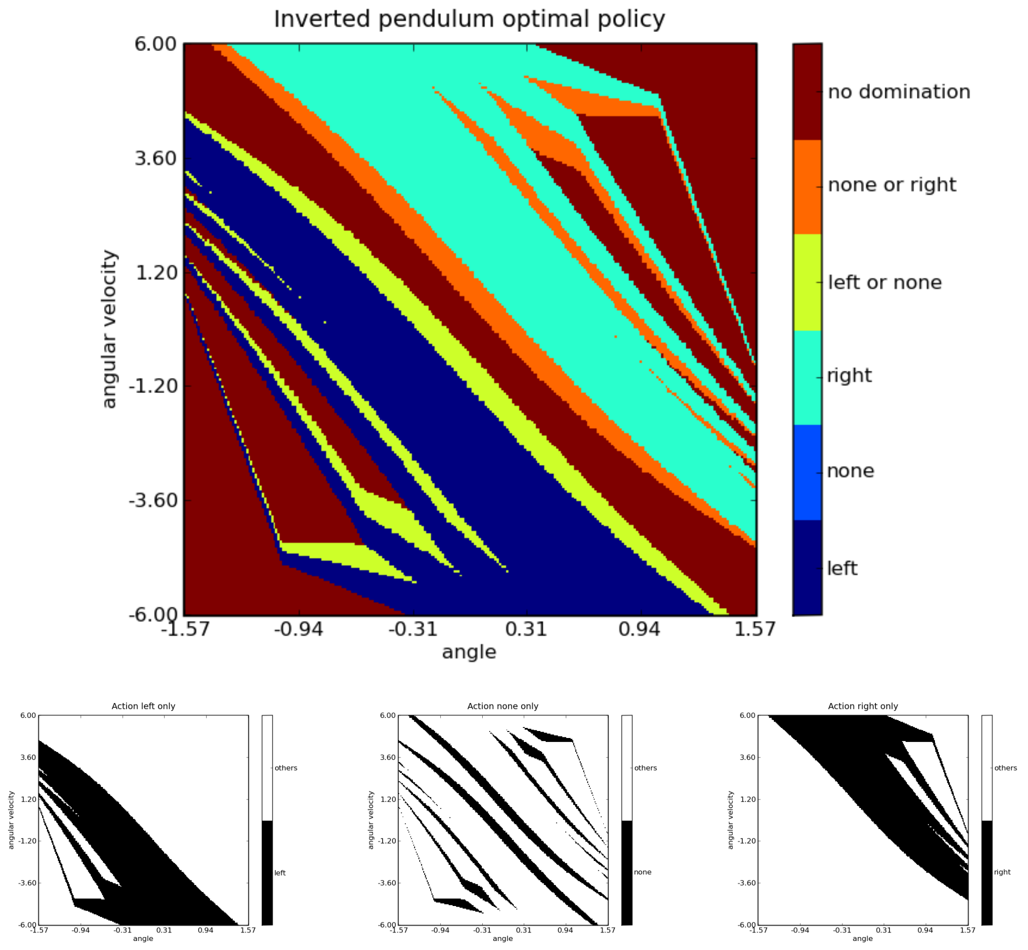


FIGURE 7.2: Optimal policy for the deterministic Inverted Pendulum. Top: dominating action loci including tied cases. Bottom: individual (non-pure) domination locus for each action.

The optimal policy for the deterministic Inverted Pendulum problem over the two-dimensional state space is shown in Figure 7.2. Surprisingly, this policy exhibits richer structure. The critical area around the balancing point is wider and somewhat more relaxed, but there are more and thinner stripes and many more ties between the left/none actions and the right/none actions. As expected there are no ties between the left/right actions. One would expect a simpler policy for a simpler problem, but apparently, the optimal policy for the deterministic version of the problem poses a somewhat more challenging classification problem. We conjecture that the richer and more complex structure is, in fact, an artifact of the deterministic nature of the problem, the discretization resolution, the time step of the simulation, and the boundary conditions.

7.1.2 Mountain Car

The optimal policy for the stochastic Mountain Car problem over the two-dimensional state space is shown in Figure 7.3. The horizontal axis is the position x of the car ranging from -1.2 to 0.5 and the vertical axis is the velocity $\dot{x}$ ranging from -0.07 to 0.07. The point $(-0.5, 0)$ corresponds to the bottom of the valley when the car is not moving. Again, one can clearly identify large areas where a single action consists the best choice. In general, the critical decision seems to be the choice of an action that gives more thrust in the direction the car is currently moving towards; forward/right throttle for positive velocity and reverse/left throttle for negative velocity. Apparently, such a policy can help the car build the necessary energy for exiting the valley. Note that the no throttle action barely constitutes the best action choice in any state, which is somewhat expected. Again, as expected, there are no areas where the left and right force actions are equally good. At bottom-left and top-right extremes, all actions are indifferent, corresponding to the cases where the car is going to exit the valley or hit the left barrier anyway, independently of the action choice. Again, the optimal policy bears sufficient structure to facilitate the classification problem. The critical area around the bottom of the valley point poses no particular difficulties in classifying the states correctly to the corresponding actions.

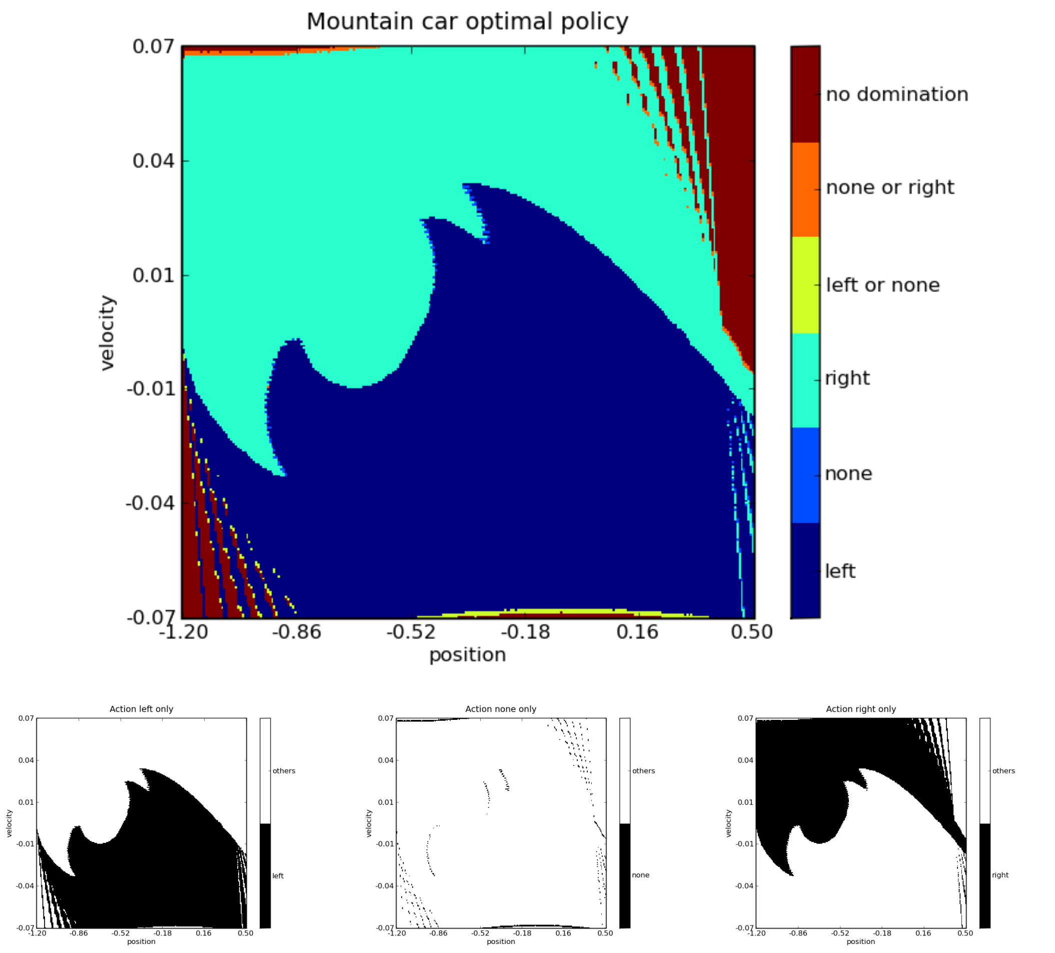


FIGURE 7.3: Optimal policy for the stochastic Mountain Car. Top: dominating action loci including tied cases. Bottom: individual (non-pure) domination locus for each action.

The optimal policy for the deterministic Mountain Car problem over the two-dimensional state space is shown in Figure 7.4. This policy does not differ significantly from the optimal one for the stochastic version, unlike the pendulum policies; this is explained by the lesser impact of the action choices on state changes in the Mountain Car domain.

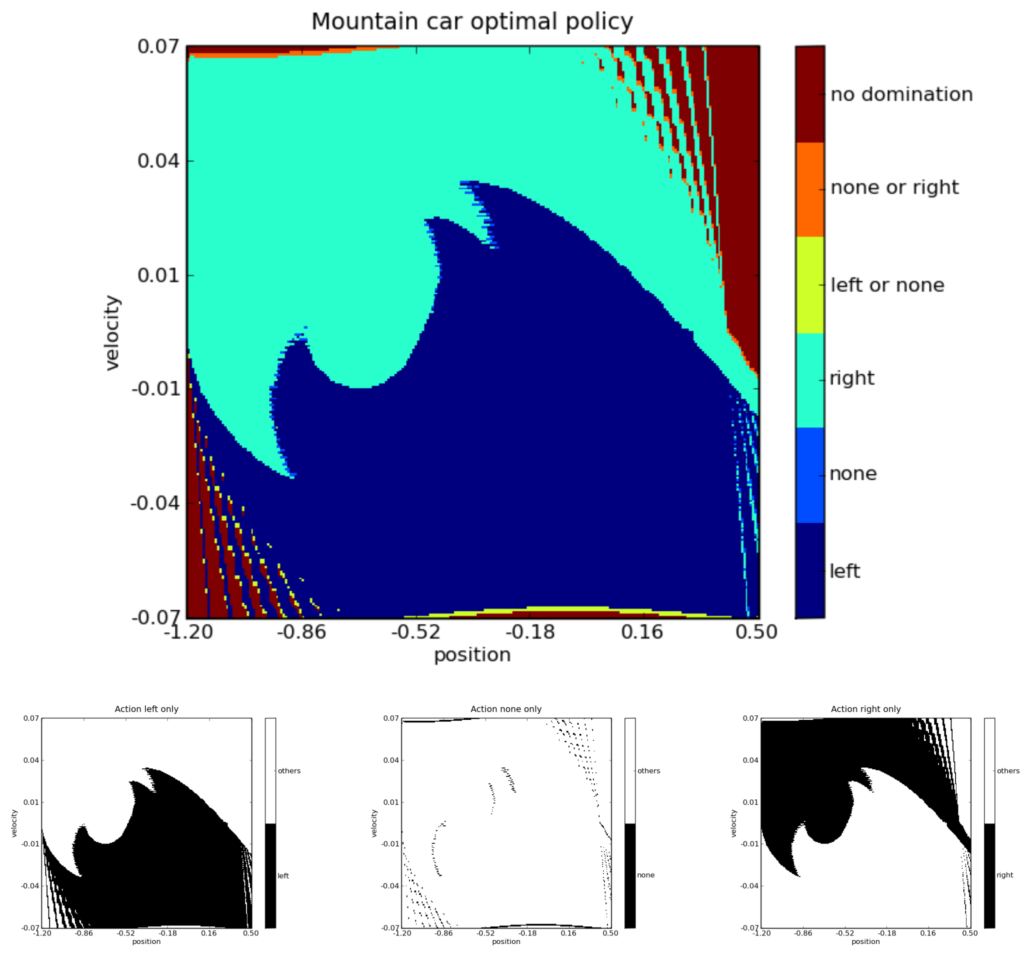


FIGURE 7.4: Optimal policy for the deterministic Mountain Car. Top: dominating action loci including tied cases. Bottom: individual (non-pure) domination locus for each action.

7.2 Directed RCPI experiments

Implementation

In this section we are going to test our hypothesis, namely that the policy representation offered by a classifier can be used to guide the exploration of policy space. Our only assumption is that a domain simulator is available. We applied our proposed approaches to the four benchmark domains described above to test their effectiveness: two domains with two dimensions, the Inverted Pendulum, and Mountain Car, one domain with four dimensions, the Acrobot, and one domain with eight dimensions, the 4-Link Planar Robot. Table 7.1 lists all algorithm tested and compared in the experiments. This list includes the original RCPI algorithm with uniform sampling, implemented both with SVMs and RVMs, as well as our four implementations of the proposed Directed RCPI (DRCPI) approaches based on the use of Active Input Vectors (AIV) or Importance Sampling (IS) and SVMs or RVMs.

All algorithms tested were implemented using Matlab; SVM classifiers and regressors are implemented using the libSVM (Chang and Lin, 2001) package, and RVM classifiers and regressors using the SparseBayesV2 (Tipping, 2009) package. All policies were represented using either SVM or RVM binary classifiers. In the approach based on Active Input Vectors, we use Powell's hybrid method, which was taken from the GNU Scientific Library (Galassi et al., 2003), for solving the nonlinear system. In the approach based on Importance Sampling, we also use SVM or RVM regressors. All kernels used were implemented using Radial Basis Functions (RBF) kernels. The initial policy π_0 in all experiments was a purely random deterministic policy. Each dimension of the state spaces was scaled to $[-1, +1]$. The ΔQ filtering limit ϵ is automatically calculated using the method described earlier. The parameters of our algorithms were tested selectively within their range to find operational values; the selected values are domain dependent.

Table 7.1: Algorithms tested

Nomenclature	State Sampling Method	Basic Algorithm	Classifier Technology
RCPI-SVM	uniform	RCPI	SVM
RCPI-RVM	uniform	RCPI	RVM
DRCPI-AIV-SVM	Active Input vectors	DRCPI	SVM
DRCPI-IS-SVM	Importance Sampling	DRCPI	SVM
DRCPI-AIV-RVM	Active Input vectors	DRCPI	RVM
DRCPI-IS-RVM	Importance Sampling	DRCPI	RVM

Resampling

We provide visualized examples of resampling methods used for the two-dimensional domains Inverted Pendulum and Mountain Car. All our experiments use uniform sampling for the first iteration. Resampling for the rest of iterations are either based on Active Input Vectors (AIV), or Importance Sampling (IS) for DRCPI. RCPI uses uniform sampling for all iterations.

Policy representation

Policies are approximated using SVM or RVM classifiers, and action advantage functions were approximated using SVM or RVM regressors. We did not try to optimize the parameters of the classifiers/regressors, since our ultimate goal is to employ simple off-the-shelf classification/regression solutions for the benefit of reinforcement learning. For Support Vector Machines in the libSVM library, there are two important parameters to select. The first is the regularization coefficient C in the primal optimization problem, which controls the trade off between minimizing training errors and controlling model complexity; the second is the β parameter of the radial basis functions kernel. For Relevance Vector Machines in the SparseBayesV2 library, there is only one parameter, namely the β parameter of the radial basis functions kernel.

Statistics

We provide separate statistics in each domain. We completed 200 runs for each algorithm in Table 7.1 and each domain. Each run used the exact same settings, but with different random seeds at initialization. For fairness, we added the multiple improvement attempts in the two RCPI variations (Algorithm 16), so that the only difference between the algorithms under comparison is the choice of the rollout states in terms of multitude and location. We run the two RCPI variants under two extreme conditions; on one hand, we used a *full* count of uniformly distributed rollout states throughout all iterations ($U_U = 200$, for Inverted Pendulum, Mountain Car, and Acrobot; and $U_U = 100$, for 4-Link Planar Robot). To demonstrate the value of directed sampling, on the other hand, we run the two RCPI variants using a *low* count of uniformly distributed rollout states throughout all iterations ($U_{UL} = 40$, for Inverted Pendulum, Mountain Car, and Acrobot; and $U_{UL} = 20$, for 4-Link Planar Robot). We run the four DRCPI variants using the full count of uniformly distributed states in the first iteration (due to lack of any domain knowledge) and the low count of selected rollout states (through directed sampling) in all subsequent iterations. As expected, in the latter RCPI case (the one with the low count) performance deteriorates (less in the Mountain Car, more in the Inverted Pendulum the Acrobot, and 4-link Planar Robot), im-

Algorithm 16 Rollout Classification Policy Iteration (RCPI) with attempts

```
Input: policy $\pi_0$, attempts $L$, trials $K$, horizon $H$, size $U_r$

$k = -1$
repeat
  $k = k + 1$
  $l = 0$
  repeat
    $l = l + 1$
    $S_{k+1}$ = a uniformly random subset of $\mathcal{S}$ of size $U_r$ (uniform sampling)
    $T_{k+1} = \varnothing$                         (initialization of the training set)
    for (each $s \in S_{k+1}$)
      for (each $a \in \mathcal{A}$)
        estimate $Q_{\pi_k}(s, a)$ using $K$ rollouts of length $H$    (simulation)
      end
      if (a dominating action $a^*$ exists in state $s$) then
        $T_{k+1} = T_{k+1} \cup \{(s, a^*)^+\}$        (for dominating action $a^*$)
        $T_{k+1} = T_{k+1} \cup \{(s, a)^-\}$          (for dominated actions $a$)
      end
    end
    $\pi_{k+1}$ = TrainClassifiers($T_{k+1}$)         (classifier/policy learning)
  until ( ($\pi_{k+1}$ is better than $\pi_k$) or ($l = L$) )   (end of repeated attempts)
until ($\pi_{k+1}$ is not better than $\pi_k$)           (end of policy iteration)
return $\pi_k$
```

plying that the proposed focused (directed) selection of rollout states plays a significant role in performance, when the rollout/simulation budget is low. All other settings were kept identical for all runs.

For each experiment, we provide averages for the total number of rollouts performed, the total number of simulated steps required, the total discounted reward accumulated by the learned policy, the number of improvement attempts before termination of policy iteration, the time to complete the entire experiment, and the number of successful runs. We also provide histograms showing the distribution of the total discounted reward, the number of successful runs, the number of steps to complete successful runs in the Mountain Car the Acrobot and the 4-Link Planar Robot domain, and the number of steps in failed runs for the Inverted Pendulum domain. A successful run corresponds to 3000 steps of balancing in the Inverted Pendulum domain, exiting from the valley in the first 3000 steps in the Mountain Car, reaching the goal in the first 3000 steps in the Acrobot domain, and reaching the goal in the first 500 steps in the 4-Link Planar Robot domain.

Table 7.2: DRCPI parameters for Pendulum domain

Symbol	DRCPI-AIV	DRCPI-IS	RCPI	Description	Pendulum
U	✓	✓		initial sample size for uniform sampling	200
M	✓	✓		subsequent sample size constructed using previous policy hints	40
L	✓	✓	✓	number of attempts to improve previous policy for a given iteration	4
K	✓	✓	✓	trials - the number of rollouts used to estimate $Q(\boldsymbol{s}, a)$ values for a given state $\boldsymbol{s}$	50
H	✓	✓	✓	horizon - number of steps per rollout	100
Z		✓		number of particles	$10 \cdot M$ (i.e. 400)
Σ		✓		covariance matrix Σ used in resampling using particles	diag(0.2)
U_U			✓	uniform sample size for RCPI, for all steps	U (i.e. 200)
U_{UL}			✓	uniform low sample size for RCPI for all steps	M (i.e. 40)
				Policy Assessment *(values are used to estimate the efficiency of the policy)*	
K_{test}	✓	✓	✓	trajectories - number of rollouts	100
H_{test}	✓	✓	✓	horizon - number of steps per rollout	3000

Table 7.3: Library parameters for Pendulum domain

		LibSVM		SparseBayesV2
Procedure	Algorithms used	β	C	β
Classification	DRCPI-AIV, DRCPI-IS, RCPI	$1/4$	100	5
Regression	DRCPI-IS	$1/4$	50	5

7.2.1 Inverted Pendulum

The values used in the experiments with the Inverted Pendulum are reported in Table 7.2 and the library parameters in Table 7.3. The initial policy π_0 was a random deterministic policy.

7.2.1.1 Sampling

Resampling for Active Input Vectors (AIV) using SVMs is shown in Figure 7.5. Active input vectors in the left sub-figure are the support vectors of the SVM binary classifiers that are used to represent the current policy. The support

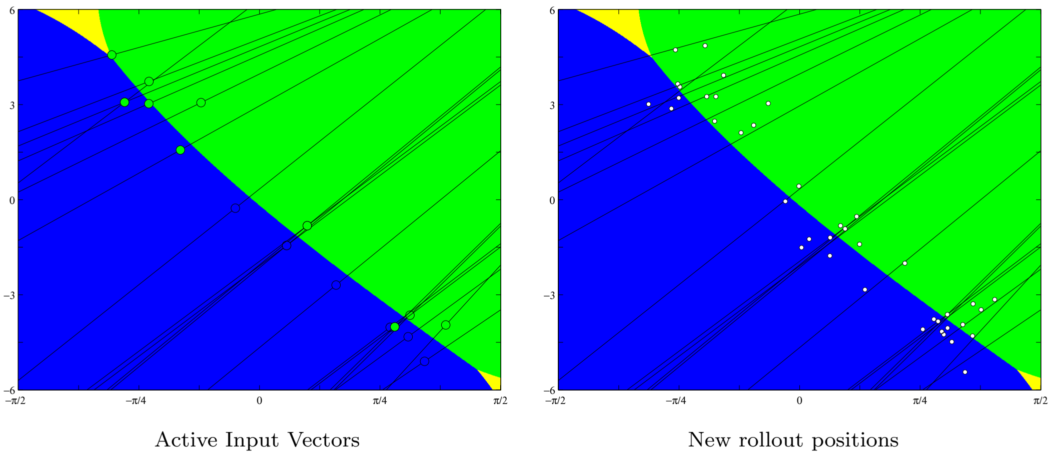
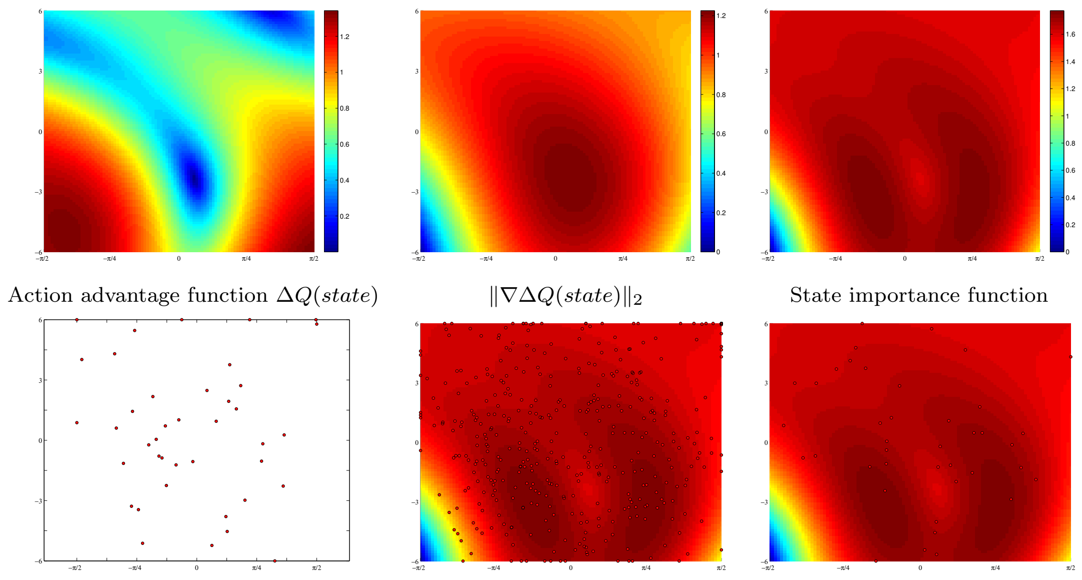


FIGURE 7.5: Inverted Pendulum domain using AIV and SVMs (DRCPI-AIV-SVM): Active Input Vectors (left) and resampled rollout states (right) over the state space.

vectors in SVMs are chosen by the optimization process to represent the border between the classes. Our goal is to refine the border to obtain an improved policy. Resampling is done as described in Section 6.4.2. In the right sub-figure the new set of states for sampling (rollout positions) is shown. Note that the new rollout states are placed around the border on both sides.

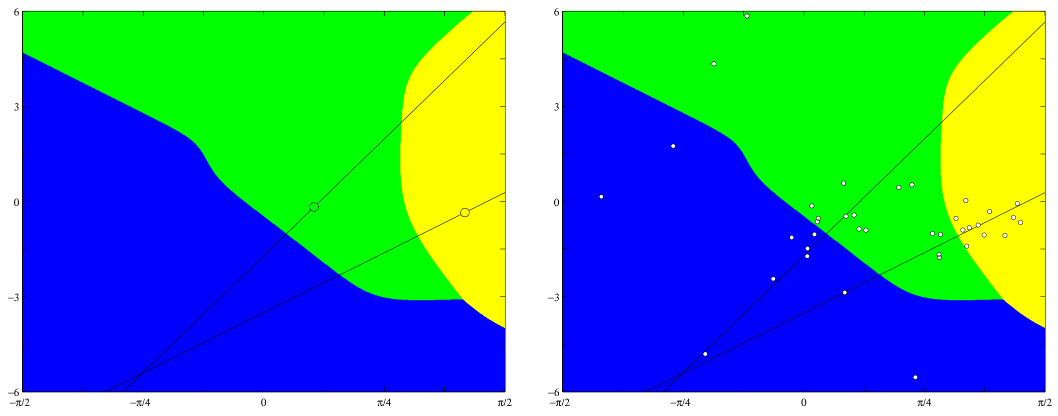
Importance sampling (IS) using SVMs is shown in Figure 7.6. The top three sub-figures are: the action advantage function $\Delta Q(state)$, the norm of it's gradient $\|\nabla\Delta Q(state)\|_2$, and the state importance 6.4.3 function. The three bottom sub-figures are: the previous iteration sampling points, the new particles that are normally distributed around the old sampling points, and in the last sub-figure the new sampling points (rollout positions) for the new and possibly improved policy. Resampling is done as described in Section 6.4.3.

Figure 7.7 shows Active Input Vectors (AIV) resampling using RVMs. Active input vectors in the left sub-figure are the relevance vectors of the RVM binary classifiers that are used to represent the current policy. Relevance vectors in RVMs are chosen by the optimization process to represent the border between the classes. Our goal is to refine the border to obtain an improved policy. Resampling is done as described in Section 6.4.2. In the right sub-figure the new set of states for sampling (rollout positions) is shown. It is worth noting that the number of relevance vectors of the RVM classifiers is much less than


Action advantage function $\Delta Q(state)$ $\|\nabla\Delta Q(state)\|_2$ State importance function

Previous sampling points Particles New sampling points

FIGURE 7.6: Inverted Pendulum domain using IS and SVMs (DRCPI-IS-SVM). Derivation of the state importance function (top) and importance sampling of rollout states (bottom) over the state space.



Active Input Vectors New rollout positions

FIGURE 7.7: Inverted Pendulum domain using AIV and RVMs (DRCPI-AIV-RVM): Active Input Vectors (left) and resampled rollout states (right) over the state space.

the support vectors of the SVM classifiers for similar policies in the same domain.

Importance sampling (IS) using RVMs is shown in Figure 7.8. The top three sub-figures are: the action advantage function $\Delta Q(state)$, the norm of its gradient $\|\nabla\Delta Q(state)\|_2$, and the state importance function (6.4.3). The three bottom sub-figures are: the previous iteration sampling points, the new particles that are normally distributed around the old sampling points, and in the last sub-figure the new sampling points (rollout positions) for the new and pos-

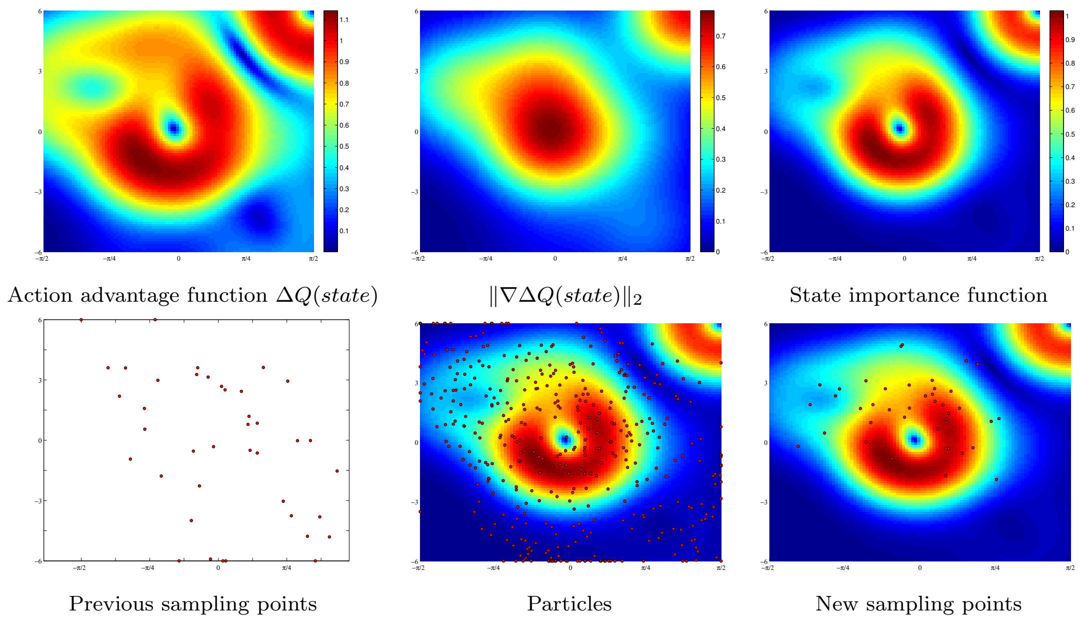


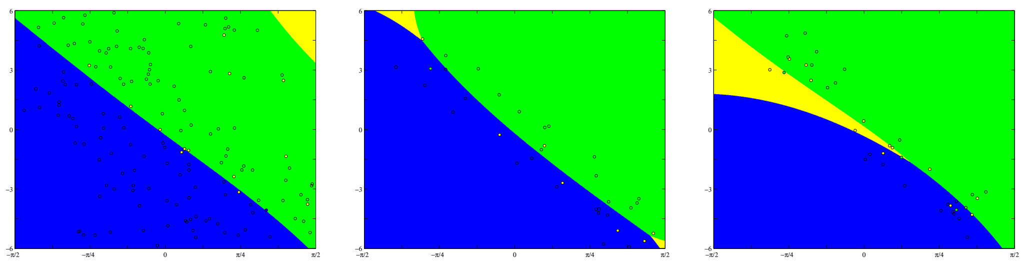
FIGURE 7.8: Inverted Pendulum domain using IS and RVMs (DRCPI-IS-RVM). Derivation of the state importance function (top) and importance sampling of rollout states (bottom) over the state space.

sibly improved policy. Resampling is done as described in Section 6.4.3.

7.2.1.2 Policy

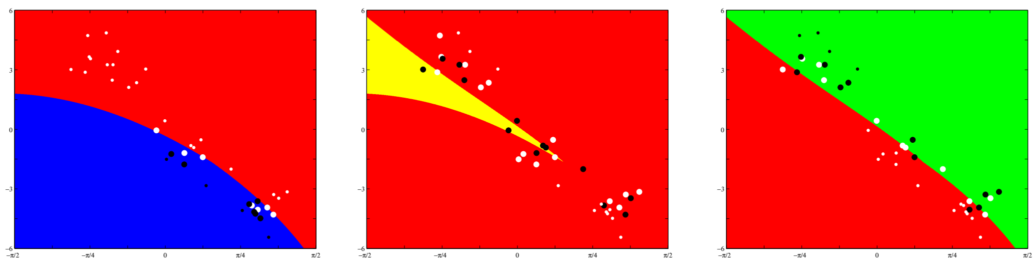
The Inverted Pendulum is a two-dimensional domain; the horizontal axis is the angle and the vertical axis is the angular velocity. We show here selected examples of policy improvement through policy iteration. The selected policies fully improve in three iterations, chosen for illustration. Unsuccessful attempts are not shown here. In all policy iteration examples discussed below, after the initial uniform distribution, the rollout states are positioned mostly around, but not on, the action boundaries. The goal in this domain is to balance the pendulum for 3000 steps.

Figure 7.9 shows a typical run of DRCPI-AIV-SVM on the Inverted Pendulum domain. The first iteration delivered a policy that yields a discounted return of 19.58 with no successful trials of balancing the pendulum, falling after 387 steps on average; this was subsequently improved with the second policy which yields a discounted return of 19.98, 53% of successful trials and 47% of unsuccessful trials, balancing for 1382 steps on average. The third policy in row yields a



policy π_1 and rollout states S_1 policy π_2 and rollout states S_2 policy π_3 and rollout states S_3

FIGURE 7.9: DRCPI-AIV-SVM on the Inverted Pendulum: three successive policies from a typical run. Dominating actions are shown in color: blue for left force, yellow for no force, and green for right force. Rollout states after filtering are shown as little circles colored with the dominating action color (inputs and targets of the training set).



π_3: binary classifier for left force π_3: binary classifier for no force π_3: binary classifier for right force

FIGURE 7.10: DRCPI-AIV-SVM learned policy for the Inverted Pendulum: the binary SVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 7.9. Each dominating class is shown in color: blue for left force, yellow for no force, green for right force; other classes are shown in red color. The corresponding training set T_3 is depicted in black for positive examples and white for negative examples; the derived support vectors are shown in bold.

discounted return of 20 (the highest possible) and 100% successful trials of balancing the pendulum. The fourth iteration did not manage to improve further the policy, as expected, and policy iteration terminated. Figure 7.10 shows the three binary SVM classifiers representing the final policy π_3.

Figure 7.11 shows a typical run of DRCPI-IS-SVM on the Inverted Pendulum domain. The first iteration delivered a policy that yields a discounted return of 19.90 with 42% successful trials to balance the pendulum, and unsuccessful trials 58% falling after 1187 steps on average; this was subsequently improved with the second policy which yields a discounted return of 19.99, 35% of successful trials and 65% of unsuccessful trials, balancing for 1248 steps on average. The third policy in row yields a discounted return of 20 (the highest possible) and 100% successful trials of balancing the pendulum. The fourth iteration did not manage to improve further the policy, as expected, and pol-

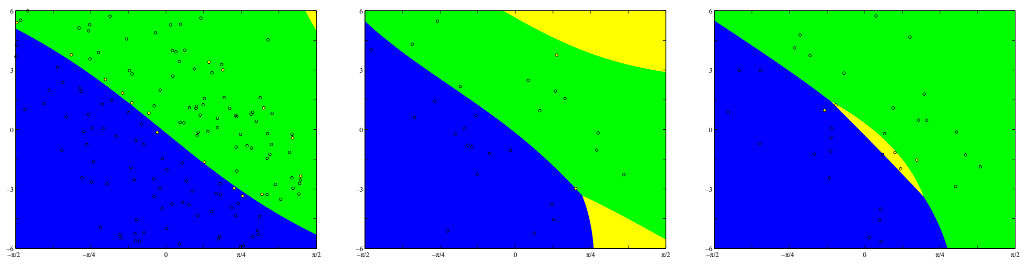

policy π_1 and rollout states S_1 policy π_2 and rollout states S_2 policy π_3 and rollout states S_3

FIGURE 7.11: DRCPI-IS-SVM on the Inverted Pendulum: three successive policies from a typical run. Dominating actions are shown in color: blue for left force, yellow for no force, and green for right force. Rollout states after filtering are shown as little circles colored with the dominating action color (inputs and targets of the training set).

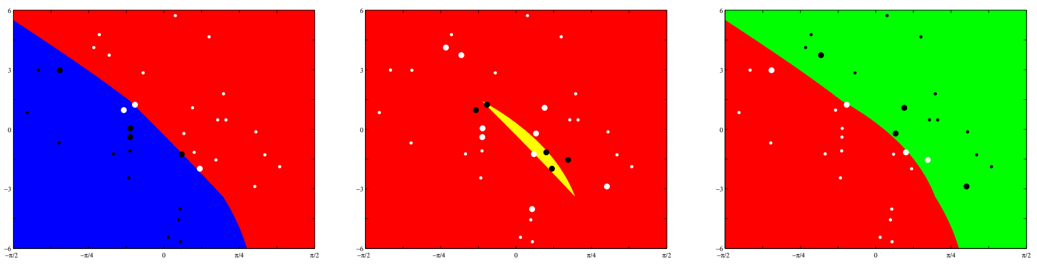
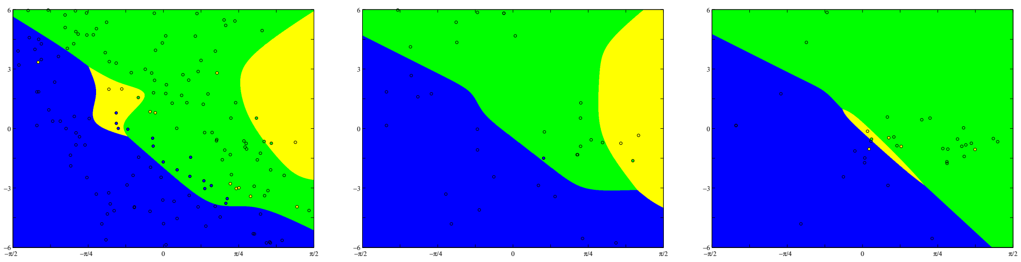

π_3: binary classifier for left force π_3: binary classifier for no force π_3: binary classifier for right force

FIGURE 7.12: DRCPI-IS-SVM learned policy for the Inverted Pendulum: the binary SVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 7.11. Each dominating class is shown in color: blue for left force, yellow for no force, green for right force; other classes are shown in red color. The corresponding training set T_3 is depicted in black for positive examples and white for negative examples; the derived support vectors are shown in bold.

icy iteration terminated. Figure 7.12 shows the three binary SVM classifiers representing the final policy π_3.

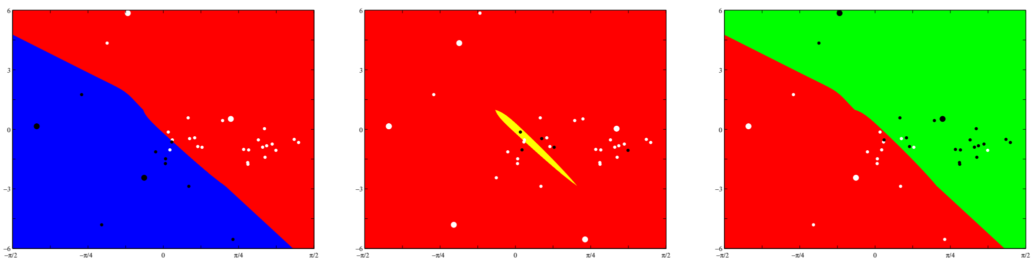
Figure 7.13 shows a typical run of DRCPI-AIV-RVM on the Inverted Pendulum domain. The first iteration delivered a policy that yields a discounted return of 18.70 with no successful trials of balancing the pendulum, falling after 166 steps on average; this was subsequently improved with the second policy which yields a discounted return of 19.99, 41% of successful trials and 59% of unsuccessful trials, balancing for 1211 steps on average. The third policy in row yields a discounted return of 20 (the highest possible) and 100% successful trials of balancing the pendulum. The fourth iteration did not manage to improve further the policy, as expected, and policy iteration terminated. Figure 7.14 shows the three binary RVM classifiers representing the final policy π_3.

Figure 7.15 shows a typical run of DRCPI-IS-RVM on the Inverted Pendu-



policy π_1 and rollout states S_1 policy π_2 and rollout states S_2 policy π_3 and rollout states S_3

FIGURE 7.13: DRCPI-AIV-RVM on the Inverted Pendulum: three successive policies from a typical run. Dominating actions are shown in color: blue for left force, yellow for no force, and green for right force. Rollout states after filtering are shown as little circles colored with the dominating action color (inputs and targets of the training set).



π_3: binary classifier for left force π_3: binary classifier for no force π_3: binary classifier for right force

FIGURE 7.14: DRCPI-AIV-RVM learned policy for the Inverted Pendulum: the binary RVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 7.13. Each dominating class is shown in color: blue for left force, yellow for no force, green for right force; other classes are shown in red color. The corresponding training set T_3 is depicted in black for positive examples and white for negative examples; the derived support vectors are shown in bold.

lum domain. The first iteration delivered a policy that yields a discounted return of 19.99 with 73% successful trials to balance the pendulum, and unsuccessful trials 27% falling after 1467 steps on average; this was subsequently improved with the second policy which yields a discounted return of 20.00, 75% of successful trials and 25% of unsuccessful trials, balancing for 1452 steps on average. The third policy in row yields a discounted return of 20 (the highest possible) and 100% successful trials of balancing the pendulum. The fourth iteration did not manage to improve further the policy, as expected, and policy iteration terminated. Figure 7.16 shows the three binary RVM classifiers representing the final policy π_3.

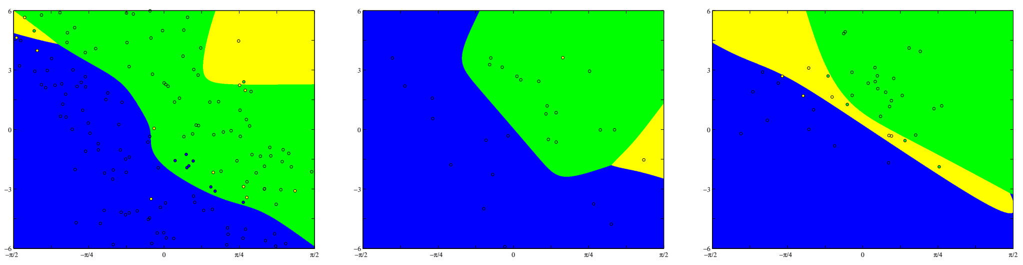

policy π_1 and rollout states S_1 policy π_2 and rollout states S_2 policy π_3 and rollout states S_3

FIGURE 7.15: DRCPI-IS-RVM on the Inverted Pendulum: three successive policies from a typical run. Dominating actions are shown in color: blue for left force, yellow for no force, and green for right force. Rollout states after filtering are shown as little circles colored with the dominating action color (inputs and targets of the training set).

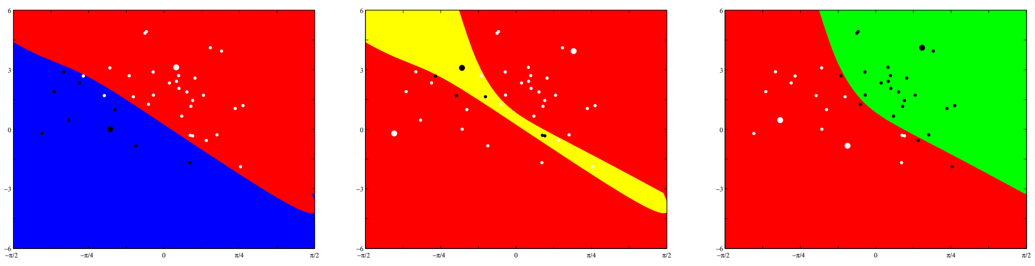

π_3: binary classifier for left force π_3: binary classifier for no force π_3: binary classifier for right force

FIGURE 7.16: DRCPI-IS-RVM learned policy for the Inverted Pendulum: the binary RVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 7.11. Each dominating class is shown in color: blue for left force, yellow for no force, green for right force; other classes are shown in red color. The corresponding training set T_3 is depicted in black for positive examples and white for negative examples; the derived support vectors are shown in bold.

7.2.1.3 Statistics

In this section, we provide statistics for the Inverted Pendulum. Each row in Table 7.4 represents averages of 200 independent runs with identical settings, but with different random seeds, for each algorithm shown in Table 7.1. The Simulation tab shows the total number of simulation steps needed for each run, while the Rollouts tab shows the total number of rollouts executed in each run. The Attempts tab shows the number of improvement attempts (the number of iterations is less than or equal to that) before termination and the Time tab shows the real time (seconds) taken by each run. Finally, the Return and Success tabs show the total expected discounted reward and the success rate respectively of the final learned policy (both measured by policy rollout from the initial state). Each run is evaluated by taking the average performance of 100 independent policy rollouts using the learned policy starting the pendulum at position $(0, 0)$ for $(\theta, \dot{\theta})$ and ending after 3000 steps of balancing or after the pendulum has fallen. Trajectories balancing the pendulum for 3000 simulation steps (5 minutes) are considered successful.

Consider first the two RCPI- algorithms with the full count of 200 uniformly distributed rollout states in each iteration. While both algorithms exhibit good learning performance (Return), the computational cost is high, as indicated by the Simulation and Time tabs. Moving on to the DRCPI- variations, we use a set of 200 (full count) uniformly distributed rollout states only in the first iteration and a set of 40 (low count) rollout states from directed sampling afterwards. Clearly, the DRCPI- variations yield significant savings in terms of Rollouts, Simulation, Attempts, and Time compared to the RCPI- algorithms with the full count, while delivering policies of comparable performance (Return). Finally, to appreciate the value of directed sampling, one can consider the RCPI- variations using a low count of only 40 uniformly distributed rollout states throughout all iterations. It is clear that in this case performance deteriorates in both Return and Success, implying that the proposed focused (directed) selection of rollout states plays a significant role in performance, when the rollout/simulation budget is low.

Comparing policy Return, which is the metric optimized by learning, SVM versions have a slight advantage over the corresponding RVM versions in all cases, albeit at the cost of increased Simulation and Time. Within the DRCPI variants, IS resampling also exhibits slight advantage over AIV resampling for both SVM and RVM. Finally, the Return delivered by the DRCPI variants is close to those delivered by RCPI with the full count of rollout states and significantly better compared to those delivered by RCPI with the low count. Nevertheless, all DRPCI variants exhibit lower simulation requirements and execution times compared to RCPI with full count and compare favorably to RCPI with low count.

Figures 7.17 through 7.24 include for each algorithm three histograms displaying the distribution of the corresponding 200 runs in terms of: (a) the policy return values, (b) the percentage of successful trails, and (c) the number of average steps per run before pendulum failure for nonbalancing trials.

Table 7.4: Inverted Pendulum (200/40 rollout states, 4 attempts): collective results of 200 runs and comparison of algorithms in terms of computational and learning performance.

Algorithm	States	Simulation	Rollouts	Attempts	Time	Return	Success
RCPI-SVM	200	4385104	1390	7.0	23.7	19.9968	87.26%
RCPI-RVM	200	4049872	1307	6.5	10.1	19.9894	94.12%
DRCPI-AIV-SVM	200/40	2508683	412	6.3	13.7	19.9807	83.34%
DRCPI-IS-SVM	200/40	936752	422	6.5	5.0	19.9880	81.48%
DRCPI-AIV-RVM	200/40	903757	384	5.7	3.2	19.9347	77.61%
DRCPI-IS-RVM	200/40	986029	404	6.1	3.5	19.9700	80.89%
RCPI-SVM	40	879649	304	7.6	2.9	19.8356	68.27%
RCPI-RVM	40	776526	314	7.8	2.6	19.5958	49.56%

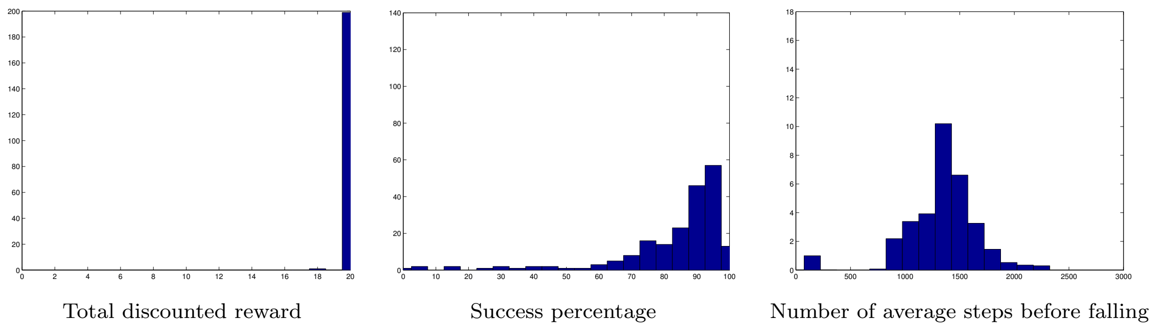


FIGURE 7.17: Inverted Pendulum using DRCPI-AIV-SVM.

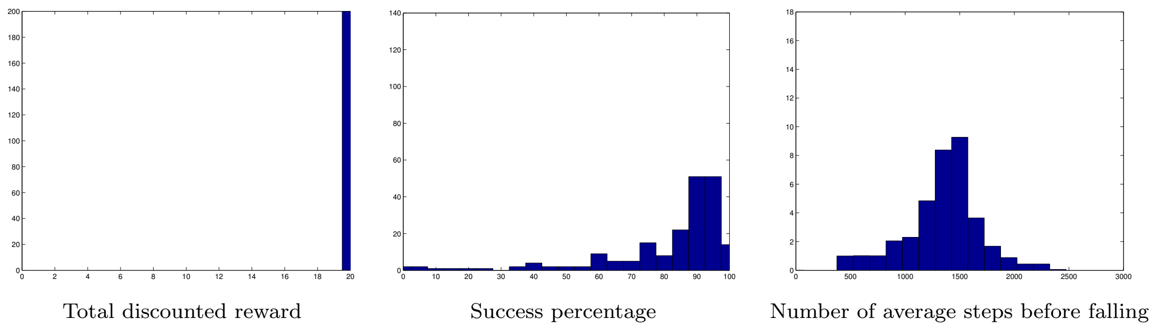


FIGURE 7.18: Inverted Pendulum using DRCPI-IS-SVM.

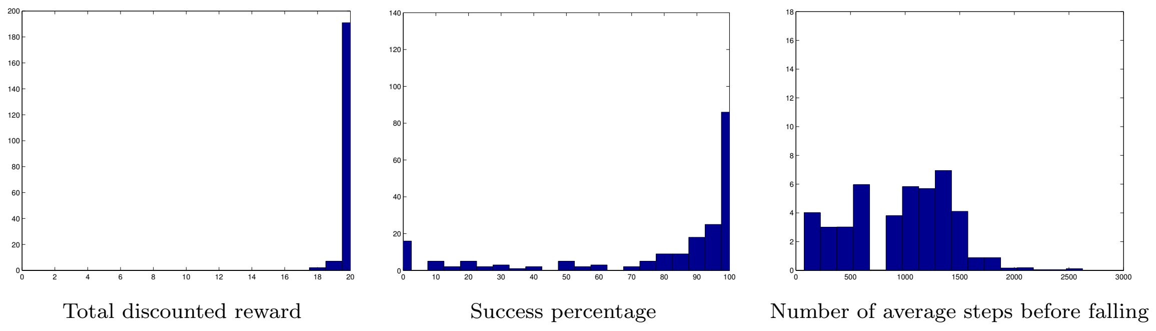


FIGURE 7.19: Inverted Pendulum domain using DRCPI-AIV-RVM.

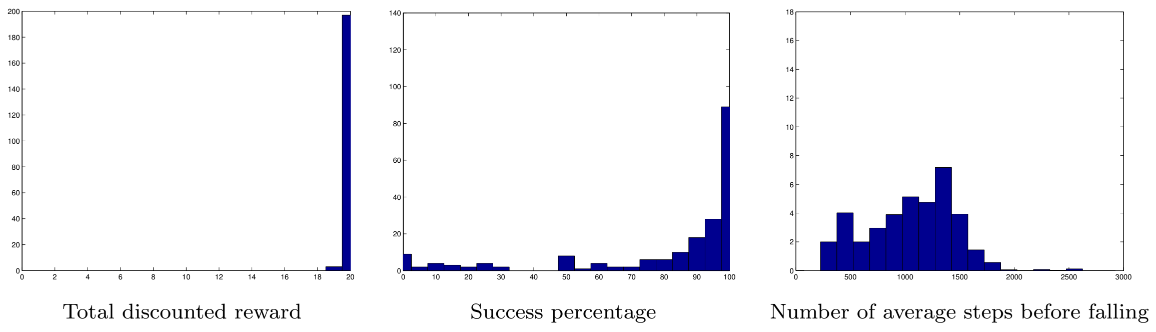


FIGURE 7.20: Inverted Pendulum using DRCPI-IS-RVM.

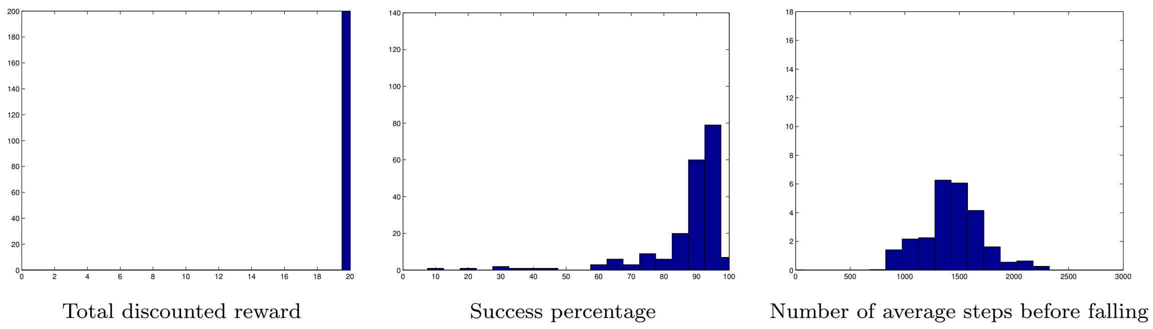


FIGURE 7.21: Inverted Pendulum using RCPI-SVM with a full count of rollout states.

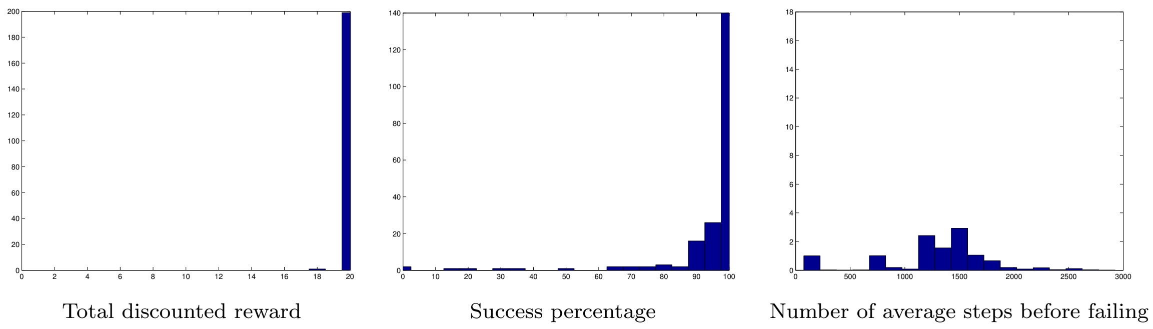


FIGURE 7.22: Inverted Pendulum using RCPI-RVM with a full count of rollout states.

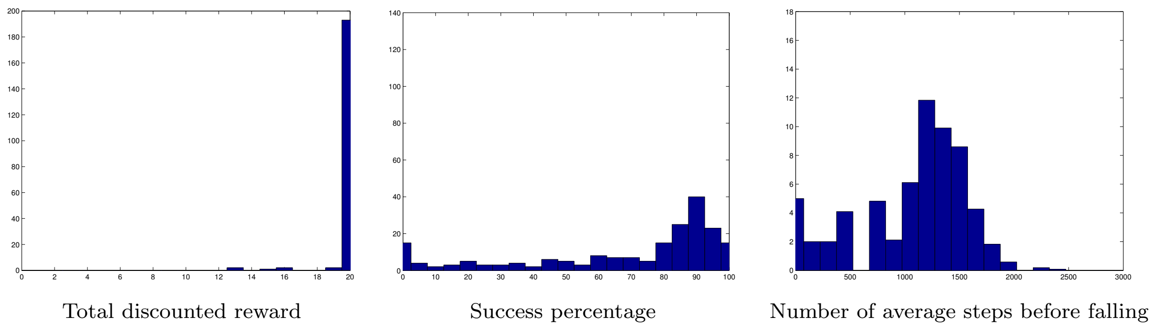


FIGURE 7.23: Inverted Pendulum using RCPI-SVM with a low count of rollout states.

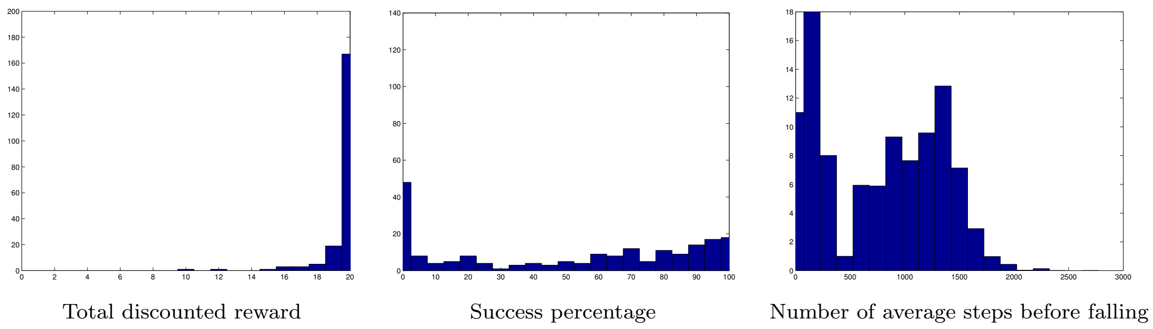


FIGURE 7.24: Inverted Pendulum using RCPI-RVM with a low count of rollout states.

7.2.2 Mountain Car

The values used in the experiments with the Mountain Car are reported in Table 7.5 and the library parameters in Table 7.6. The initial policy π_0 was a

Table 7.5: DRCPI parameters for Mountain Car domain

Symbol	DRCPI-AIV	DRCPI-IS	RCPI	Description	Mountain Car
U	✓	✓		initial sample size for uniform sampling	200
M	✓	✓		subsequent sample size constructed using previous policy hints	40
L	✓	✓	✓	number of attempts to improve previous policy for a given iteration	4
K	✓	✓	✓	trials - the number of rollouts used to estimate $Q(\boldsymbol{s}, a)$ values for a given state $\boldsymbol{s}$	50
H	✓	✓	✓	horizon - number of steps per rollout	100
Z		✓		number of particles	$10 \cdot M$ (i.e. 400)
Σ		✓		covariance matrix Σ used in resampling using particles	diag(0.2)
U_U			✓	uniform sample size for RCPI, for all steps	U (i.e. 200)
U_{UL}			✓	uniform low sample size for RCPI for all steps	M (i.e. 40)
				Policy Assessment *(values are used to estimate the efficiency of the policy)*	
K_{test}	✓	✓	✓	trajectories - number of rollouts	100
H_{test}	✓	✓	✓	horizon - number of steps per rollout	3000

Table 7.6: Library parameters for mountain Car domain

		LibSVM		SparseBayesV2
Procedure	Algorithms used	β	C	β
Classification	DRCPI-AIV, DRCPI-IS, RCPI	$1/2$	200	5
Regression	DRCPI-IS	$1/2$	200	5

random deterministic policy.

7.2.2.1 Sampling

Resampling for Active Input Vectors (AIV) using SVMs is shown in Figure 7.25. Active input vectors in the left sub-figure are the support vectors of the SVM binary classifiers that are used to represent the current policy. The support vectors in SVMs are chosen by the optimization process to represent the border between the classes. Our goal is to refine the border to obtain an improved policy. Resampling is done as described in Section 6.4.2. In the right

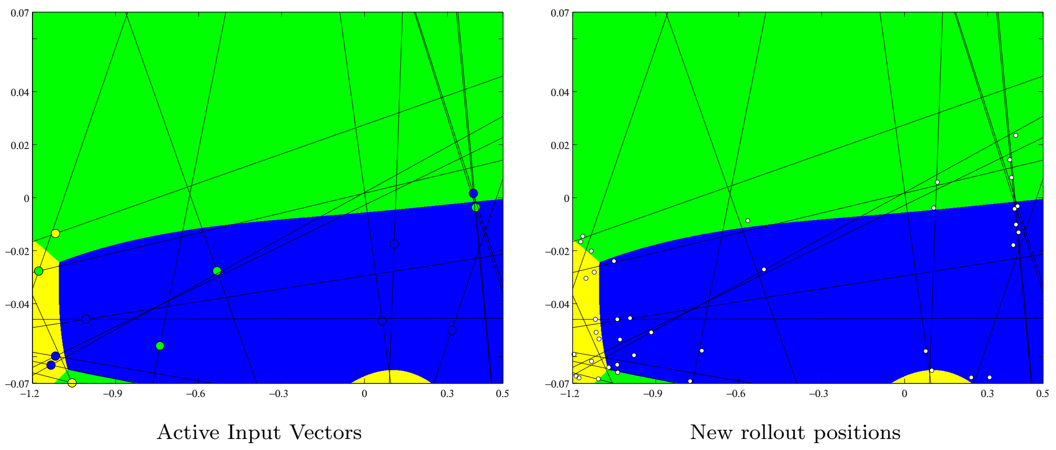


FIGURE 7.25: Mountain Car domain using AIV and SVMs (DRCPI-AIV-SVM): Active Input Vectors (left) and resampled rollout states (right) over the state space.

sub-figure the new set of states for sampling (rollout positions) is shown. Note that the new rollout states are placed around the border on both sides.

Importance sampling (IS) using SVMs is shown in Figure 7.26. The top three sub-figures are: the action advantage function $\Delta Q(state)$, the norm of it's gradient $\|\nabla \Delta Q(state)\|_2$, and the state importance 6.4.3 function. The three bottom sub-figures are: the previous iteration sampling points, the new particles that are normally distributed around the old sampling points, and in the last sub-figure the new sampling points (rollout positions) for the new and possibly improved policy. Resampling is done as described in Section 6.4.3.

Figure 7.27 shows Active Input Vectors (AIV) resampling using RVMs. Active input vectors in the left sub-figure are the relevance vectors of the RVM binary classifiers that are used to represent the current policy. Relevance vectors in RVMs are chosen by the optimization process to represent the border between the classes. Our goal is to refine the border to obtain an improved policy. Resampling is done as described in Section 6.4.2. In the right sub-figure the new set of states for sampling (rollout positions) is shown. It is worth noting that the number of relevance vectors of the RVM classifiers is much less than the support vectors of the SVM classifiers for similar policies in the same domain

Importance sampling (IS) using RVMs is shown in Figure 7.28. The top three

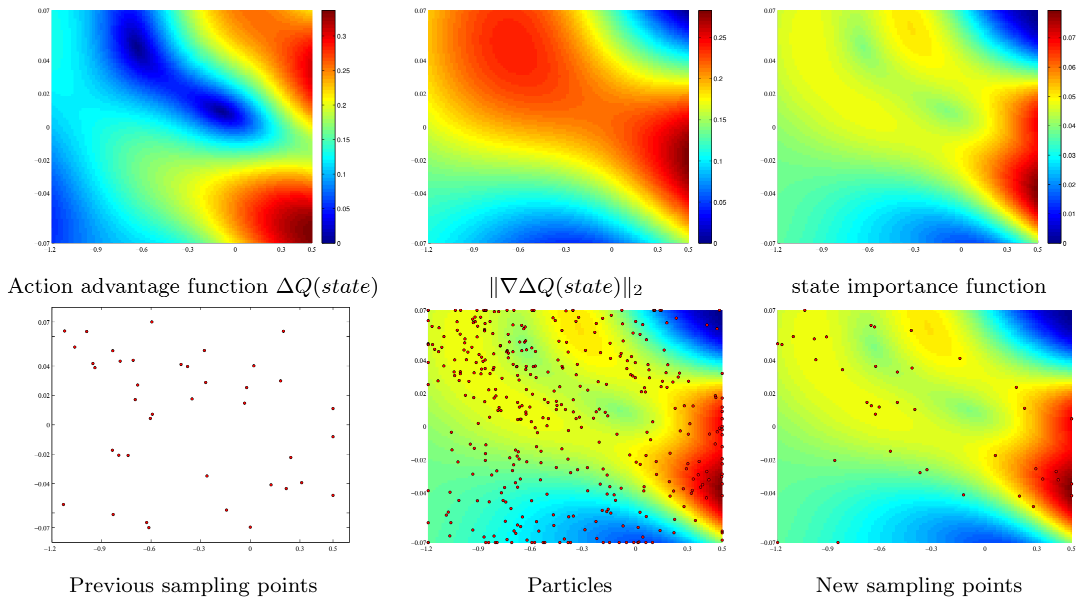


FIGURE 7.26: Mountain Car domain using IS and SVMs (DRCPI-IS-SVM). Derivation of the state importance function (top) and importance sampling of rollout states (bottom) over the state space.

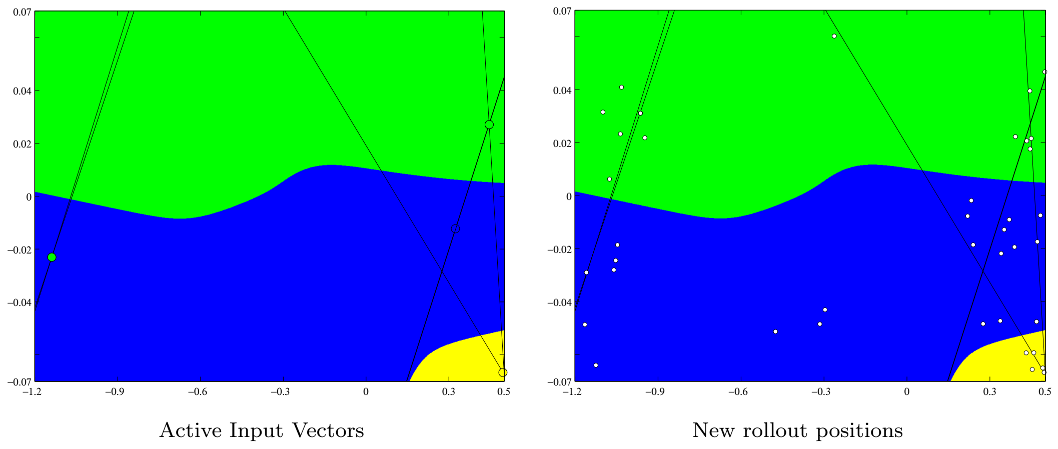


FIGURE 7.27: Mountain Car domain using AIV and RVMs (DRCPI-AIV-RVM): Active Input Vectors (left) and resampled rollout states (right) over the state space.

sub-figures are: the action advantage function $\Delta Q(state)$, the norm of its gradient $\|\nabla \Delta Q(state)\|_2$, and the state importance function (6.4.3). The three bottom sub-figures are: the previous iteration sampling points, the new particles that are normally distributed around the old sampling points, and in the last sub-figure the new sampling points (rollout positions) for the new and possibly improved policy. Resampling is done as described in Section 6.4.3.

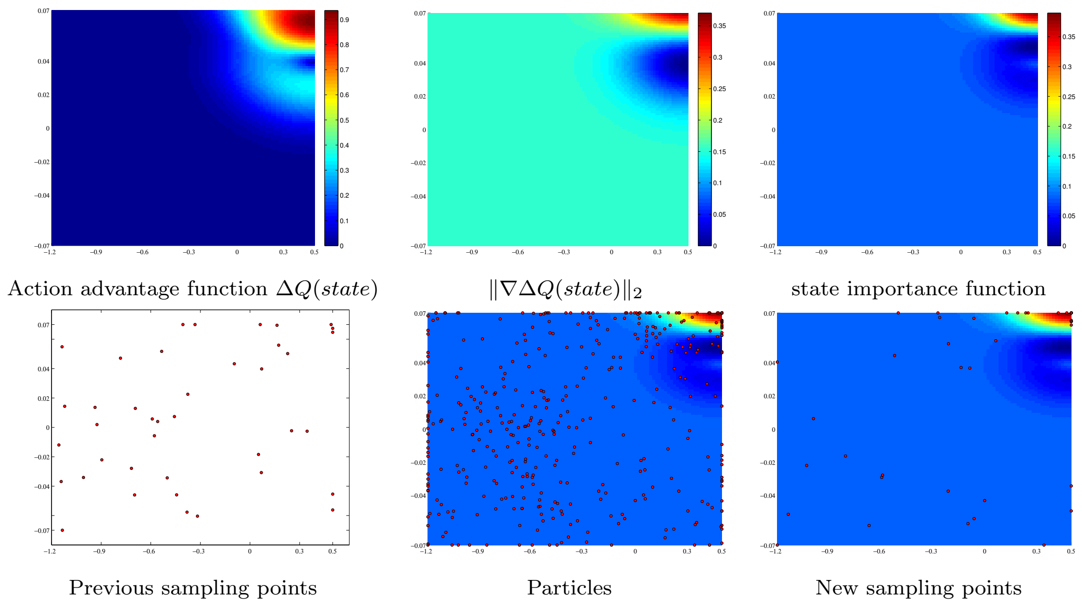
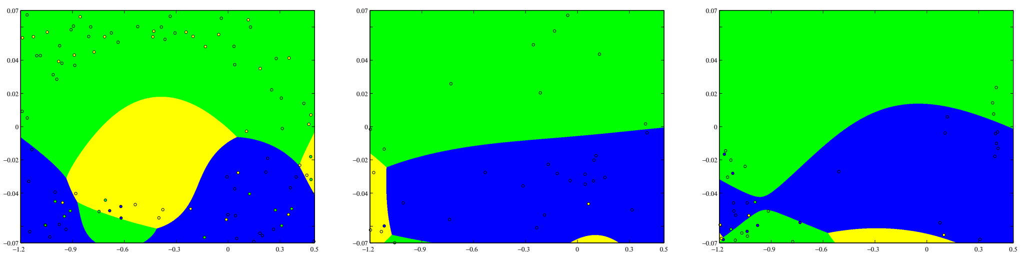


FIGURE 7.28: Mountain Car domain using IS and RVMs (DRCPI-IS-RVM). Derivation of the state importance function (top) and importance sampling of rollout states (bottom) over the state space.

7.2.2.2 Policy

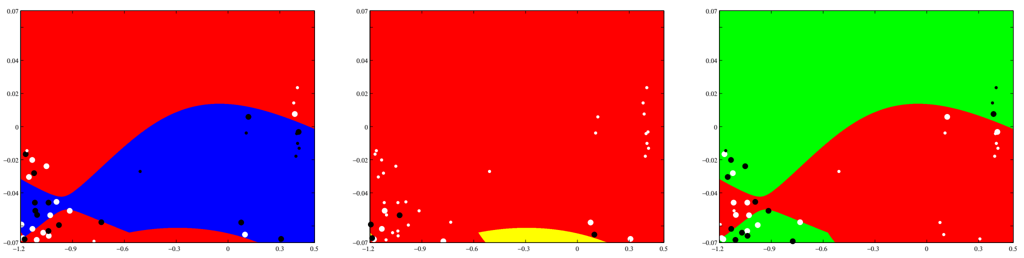
The Mountain Car is a two-dimensional domain; the horizontal axis is the position, and the vertical axis is the velocity. We show here selected examples of policy improvement through policy iteration. The selected policies fully improve in three iterations, chosen for illustration. Unsuccessful attempts are not shown here. In all policy iteration examples discussed below, after the initial uniform distribution, the rollout states are positioned mostly around, but not on, the action boundaries. The goal in this domain is to get the car out the value within the first 3000 steps.

Figure 7.29 shows a typical run of DRCPI-AIV-SVM on the Mountain Car domain. The first iteration delivered a policy that yields a discounted return of 0.0000 with 38% successful trials to exit the valley after 2018 steps on average; this was subsequently improved with the second policy which yields a discounted return of 0.2355, successful trials 99% to exit the valley after 245 steps on average. The third policy in row yields a discounted return of 0.3330, successful 100% to exit the valley after 109 steps on average. The fourth iteration did not manage to improve further the policy, as expected,



policy π_1 and rollout states S_1 policy π_2 and rollout states S_2 policy π_3 and rollout states S_3

FIGURE 7.29: DRCPI-AIV-SVM on the Mountain Car: three successive policies from a typical run. Dominating actions are shown in color: blue for left force, yellow for no force, and green for right force. Rollout states after filtering are shown as little circles colored with the dominating action color (inputs and targets of the training set).



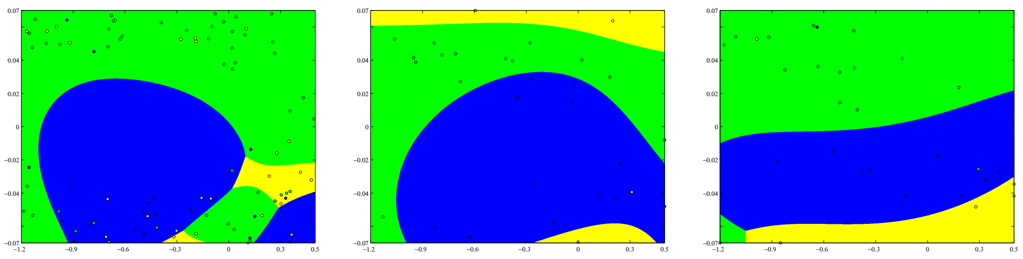
π_3: binary classifier for reverse throttle π_3: binary classifier for no throttle π_3: binary classifier for forward throttle

FIGURE 7.30: DRCPI-AIV-SVM learned policy for the Mountain Car: the binary SVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 7.29. Each dominating class is shown in color: blue for left force, yellow for no force, green for right force; other classes are shown in red color. The corresponding training set T_3 is depicted in black for positive examples and white for negative examples; the derived support vectors are shown in bold.

and policy iteration terminated. Figure 7.30 shows the three binary SVM classifiers representing the final policy π_3.

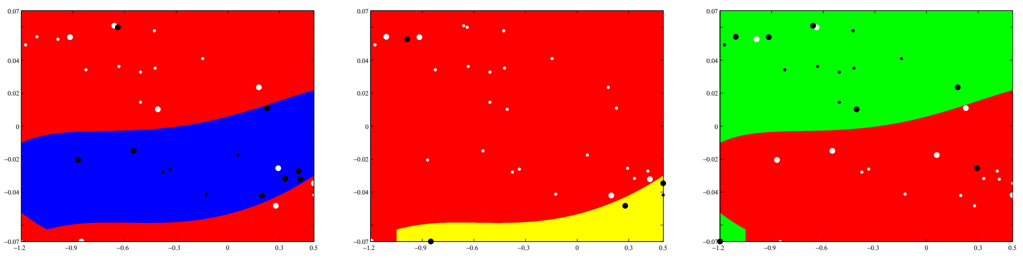
Figure 7.31 shows a typical run of DRCPI-IS-SVM on the Mountain Car domain. The first iteration delivered a policy that yields a discounted return of 0.0000 with 7% successful trials to exit the valley after 2271 steps on average; this was subsequently improved with the second policy which yields a discounted return of 0.0795, successful trials 92% to exit the valley after 704 steps on average. The third policy in row yields a discounted return of 0.3024, successful 100% to exit the valley after 119 steps on average. The fourth iteration did not manage to improve further the policy, as expected, and policy iteration terminated. Figure 7.32 shows the three binary SVM classifiers representing the final policy π_3.

Figure 7.33 shows a typical run of DRCPI-AIV-RVM on the Mountain Car



policy π_1 and rollout states S_1 policy π_2 and rollout states S_2 policy π_3 and rollout states S_3

FIGURE 7.31: DRCPI-IS-SVM on the Mountain Car: three successive policies from a typical run. Dominating actions are shown in color: blue for left force, yellow for no force, and green for right force. Rollout states after filtering are shown as little circles colored with the dominating action color (inputs and targets of the training set).

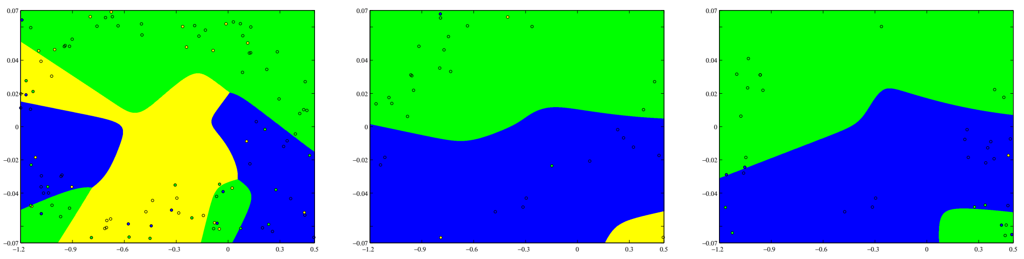


π_3: binary classifier for reverse throttle π_3: binary classifier for no throttle π_3: binary classifier for forward throttle

FIGURE 7.32: DRCPI-IS-SVM learned policy for the Mountain Car: the binary SVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 7.31. Each dominating class is shown in color: blue for left force, yellow for no force, green for right force; other classes are shown in red color. The corresponding training set T_3 is depicted in black for positive examples and white for negative examples; the derived support vectors are shown in bold.

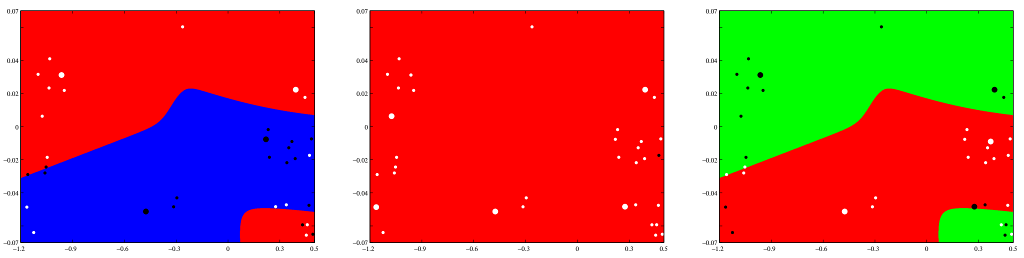
domain. The first iteration delivered a policy that yields a discounted return of 0.0010 with 90% successful trials to exit the valley after 1343 steps on average; this was subsequently improved with the second policy which yields a discounted return of 0.3111, successful trials 100% to exit the valley after 116 steps on average. The third policy in row yields a discounted return of 0.3493, successful 100% to exit the valley after 104 steps on average. The fourth iteration did not manage to improve further the policy, as expected, and policy iteration terminated. Figure 7.34 shows the three binary SVM classifiers representing the final policy π_3.

Figure 7.35 shows a typical run of DRCPI-IS-RVM on the Mountain Car domain. The first iteration delivered a policy that yields a discounted return of 0.0379 with 100% successful trials to exit the valley after 483 steps on average; this was subsequently improved with the second policy which yields a



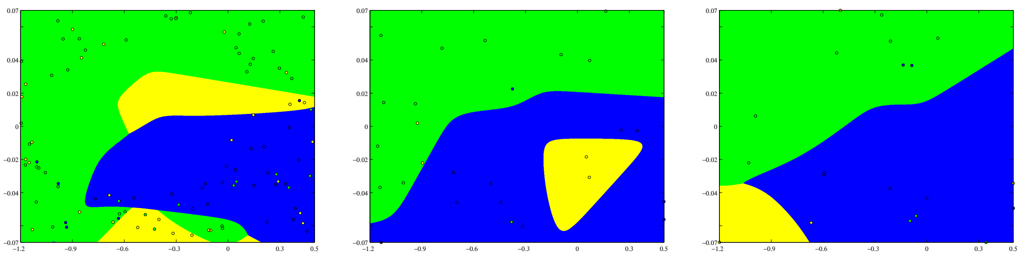
policy π_1 and rollout states S_1 policy π_2 and rollout states S_2 policy π_3 and rollout states S_3

FIGURE 7.33: DRCPI-AIV-RVM on the Mountain Car: three successive policies from a typical run. Dominating actions are shown in color: blue for left force, yellow for no force, and green for right force. Rollout states after filtering are shown as little circles colored with the dominating action color (inputs and targets of the training set).



π_3: binary classifier for reverse throttle π_3: binary classifier for no throttle π_3: binary classifier for forward throttle

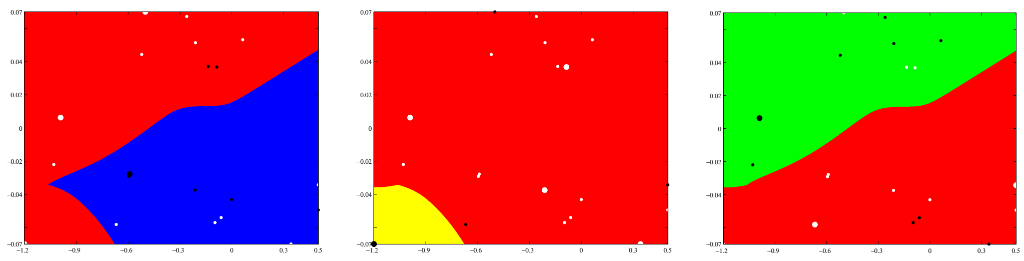
FIGURE 7.34: DRCPI-AIV-RVM learned policy for the Mountain Car: the binary RVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 7.33. Each dominating class is shown in color: blue for left force, yellow for no force, green for right force; other classes are shown in red color. The corresponding training set T_3 is depicted in black for positive examples and white for negative examples; the derived support vectors are shown in bold.



policy π_1 and rollout states S_1 policy π_2 and rollout states S_2 policy π_3 and rollout states S_3

FIGURE 7.35: DRCPI-IS-RVM on the Mountain Car: three successive policies from a typical run. Dominating actions are shown in color: blue for left force, yellow for no force, and green for right force. Rollout states after filtering are shown as little circles colored with the dominating action color (inputs and targets of the training set).

discounted return of 0.2521, successful trials 100% to exit the valley after 137 steps on average. The third policy in row yields a discounted return of 0.3265, successful 100% to exit the valley after 111 steps on average. The fourth iteration did not manage to improve further the policy, as expected, and policy iteration terminated. Figure 7.36 shows the three binary SVM classifiers representing the final policy π_3.



π_3: binary classifier for reverse throttle

π_3: binary classifier for no throttle

π_3: binary classifier for forward throttle

FIGURE 7.36: DRCPI-IS-RVM learned policy for the Mountain Car: the binary RVM classifiers for each action of the final balancing policy π_3 in the experiment of Figure 7.35. Each dominating class is shown in color: blue for left force, yellow for no force, green for right force; other classes are shown in red color. The corresponding training set T_3 is depicted in black for positive examples and white for negative examples; the derived support vectors are shown in bold.

7.2.2.3 Statistics

In this section, we provide statistics for the Mountain Car. Each row in Table 7.7 represents averages of 200 independent runs, but with different random seeds, for each algorithm shown in Table 7.1. The Simulation tab shows the total number of simulation steps needed for each run, while the Rollouts tab shows the total number of rollouts executed in each run. The Attempts tab shows the number of improvement attempts (the number of iterations is less than or equal to that) before termination and the Time tab shows the real time (seconds) taken by each run. Finally, the Return and Success tabs show the total expected discounted reward and the success rate respectively of the final learned policy (both measured by policy rollout from the initial state). Each run is evaluated by taking the average performance of 100 independent policy rollouts using the learned policy starting the car at position $(-.5, 0)$ for $(x, \dot{x})$ and ending when successfully exiting the valley in less than 3000 steps or after 3000 steps of simulation with no exit. Trajectories in which the car fails to exit within 3000 simulation steps are considered unsuccessful.

Consider first the two RCPI- algorithms with the full count of 200 uniformly distributed rollout states in each iteration. While both algorithms exhibit good learning performance (Return), the computational cost is high, as indicated by the Simulation and Time tabs. Moving on to the DRCPI- variations, we use a set of 200 (full count) uniformly distributed rollout states only in the first iteration and a set of 40 (low count) rollout states from directed sampling afterwards. Clearly, the DRCPI- variations yield significant savings in terms of Rollouts, Simulation, Attempts, and Time compared to the RCPI-algorithms with the full count, while delivering policies of comparable performance (Return) in most cases. Finally, to appreciate the value of directed sampling, one can consider the RCPI- variations using a low count of only 40 uniformly distributed rollout states throughout all iterations. It is clear that in this case performance deteriorates in both Return and Success (especially for SVM), implying that the proposed focused (directed) selection of rollout states plays a significant role in performance, when the rollout/simulation budget is low.

Comparing policy Return, which is the metric optimized by learning, RVM versions have a slight advantage over the corresponding SVM versions in all cases. Within the DRCPI variants, AIV resampling also exhibits advantage over IS resampling for both SVM and RVM. Finally, the Return delivered by the DRCPI variants is close to those delivered by RCPI with the full count of rollout states and significantly better compared to those delivered by RCPI with the low count. Nevertheless, all DRPCI variants exhibit lower simulation requirements and execution times compared to RCPI with full count and compare favorably to RCPI with low count.

Figures 7.37 through 7.44 include for each algorithm three histograms displaying the distribution of the corresponding 200 runs in terms of: (a) the policy return values, (b) the percentage of successful trails, and (c) the number of average steps per run for successful exit only for exiting trials.

Table 7.7: Mountain Car (200/40, four attempts), collective results of 200 runs and comparison of algorithms in terms of computational and learning performance

Algorithm	States	Simulation	Rollouts	Attempts	Time	Return	Success
RCPI-SVM	200	17031407	1986	9.9	95.8	0.2711	99.52%
RCPI-RVM	200	14402231	1683	8.4	14.6	0.2953	100.00%
DRCPI-AIV-SVM	200/40	5908725	573	8.6	16.3	0.2440	98.74%
DRCPI-IS-SVM	200/40	5198928	490	6.5	10.1	0.1786	89.07%
DRCPI-AIV-RVM	200/40	4572201	461	7.6	3.2	0.2663	98.93%
DRCPI-IS-RVM	200/40	4137240	480	7.3	3.0	0.2497	99.37%
RCPI-SVM	40	3302827	343	8.6	5.3	0.2183	84.30%
RCPI-RVM	40	3098182	338	8.5	3.3	0.2732	98.52%

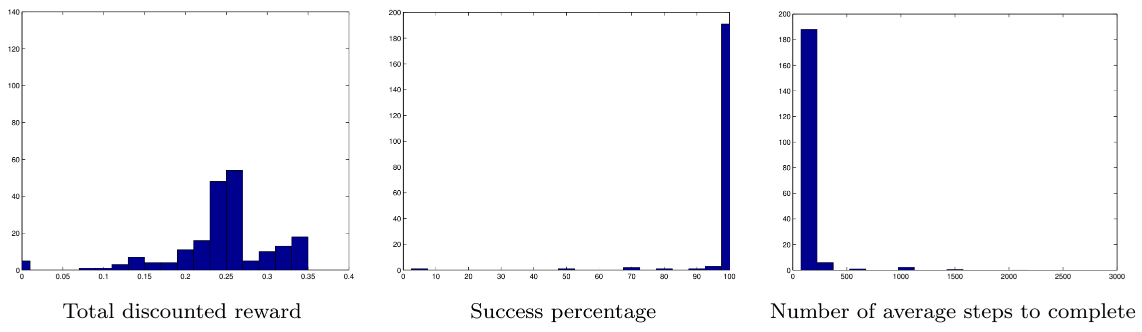


FIGURE 7.37: Mountain Car using DRCPI-AIV-SVM.

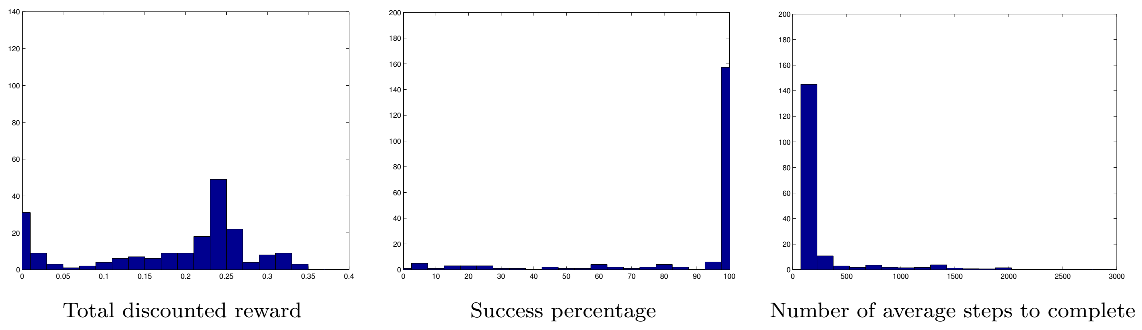


FIGURE 7.38: Mountain Car using DRCPI-IS-SVM.

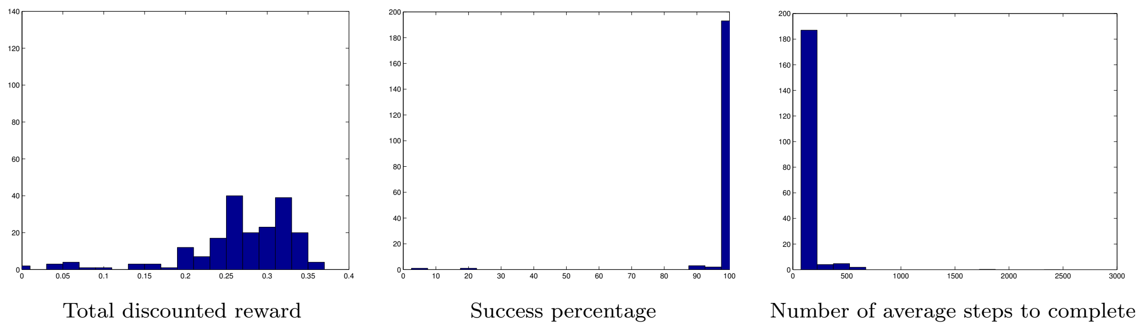


FIGURE 7.39: Mountain Car domain using DRCPI-AIV-RVM.

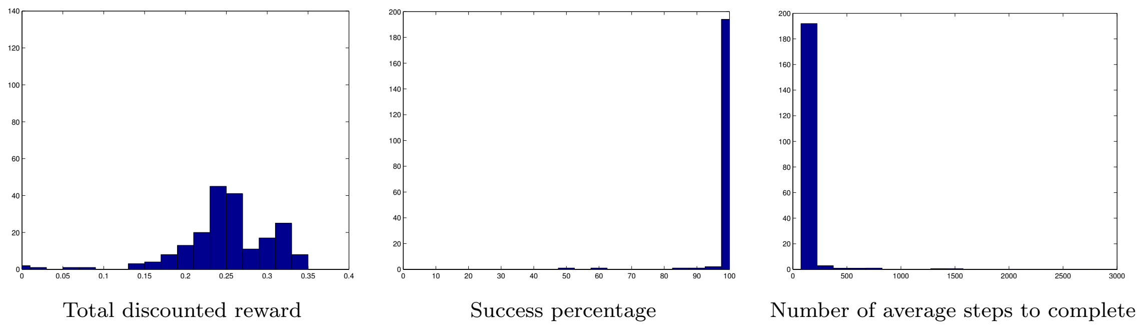


FIGURE 7.40: Mountain Car using DRCPI-IS-RVM.

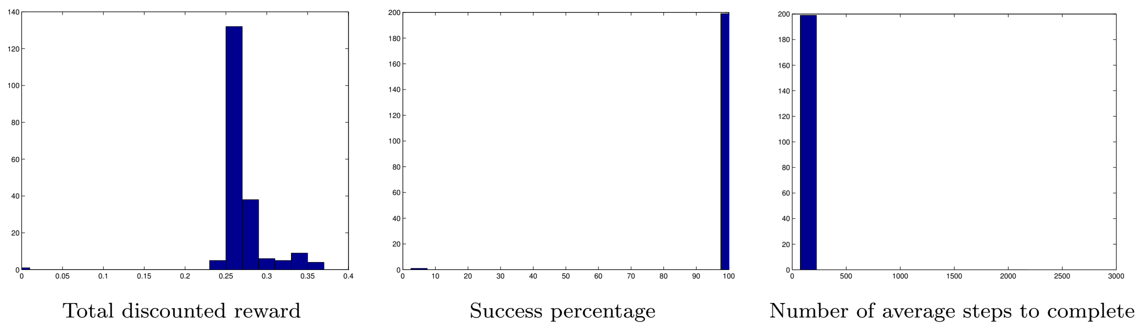


Figure 7.41: Mountain Car using RCPI-SVM with a full count of rollout states.

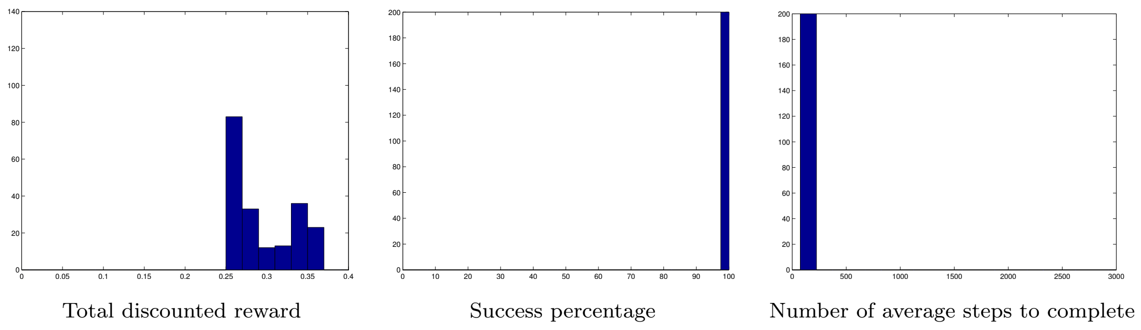


Figure 7.42: Mountain Car using RCPI-RVM with a full count of rollout states.

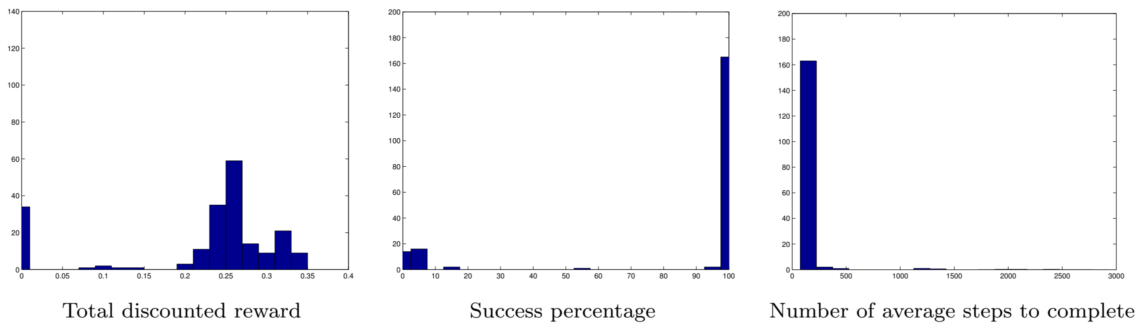


Figure 7.43: Mountain Car using RCPI-SVM with a low count of rollout states.

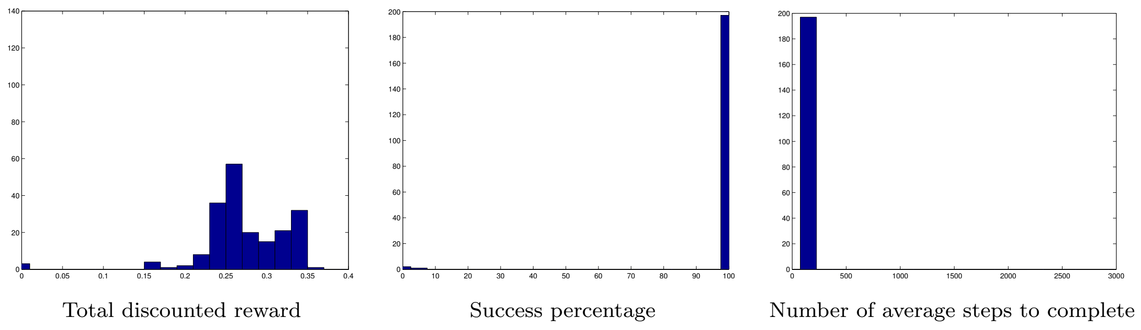


Figure 7.44: Mountain Car using RCPI-RVM with a low count of rollout states.

7.2.3 Acrobot

The values used in the experiments with the Acrobot are reported in Table 7.8 and the library parameters in Table 7.9. The initial policy π_0 was a random

Table 7.8: DRCPI parameters for Acrobot domain

Symbol	DRCPI-AIV	DRCPI-IS	RCPI	Description	Acrobot
U	✓	✓		initial sample size for uniform sampling	200
M	✓	✓		subsequent sample size constructed using previous policy hints	40
L	✓	✓	✓	number of attempts to improve previous policy for a given iteration	4
K	✓	✓	✓	trials - the number of rollouts used to estimate $Q(s, a)$ values for a given state s	50
H	✓	✓	✓	horizon - number of steps per rollout	100
Z		✓		number of particles	$10 \cdot M$ (i.e. 400)
Σ		✓		covariance matrix Σ used in resampling using particles	diag(0.2)
U_U			✓	uniform sample size for RCPI, for all steps	U (i.e. 200)
U_{UL}			✓	uniform low sample size for RCPI for all steps	M (i.e. 40)
				Policy Assessment *(values are used to estimate the efficiency of the policy)*	
K_{test}	✓	✓	✓	trajectories - number of rollouts	100
H_{test}	✓	✓	✓	horizon - number of steps per rollout	3000

Table 7.9: Library parameters for Acrobot domain

		LibSVM		SparseBayesV2
Procedure	Algorithms used	β	C	β
Classification	DRCPI-AIV, DRCPI-IS, RCPI	$^1/_6$	100	$^1/_6$
Regression	DRCPI-IS	$^1/_6$	50	$^1/_6$

deterministic policy.

In this section, we provide statistics for the Acrobot. We display experimental results for both the energy shaping reward and the classic reward. We added a column with *average steps to reach the goal*, for performance comparison between the two different reward schemes. Table 7.10 holds data for energy based reward, each row represents averages of 200 independent runs with identical settings, but with different random seeds, for each algorithm shown in Table 7.1. Table 7.11 shows results for classic reward, each row represents averages of 200 independent runs with identical settings, but with different random seeds, for each algorithm. The Simulation tab shows the total number of simulation steps needed for each run, while the Rollouts tab shows the total number of rollouts executed in each run. The Attempts tab shows the number of improvement attempts (the number of iterations is less than or equal to that) before termination and the Time tab shows the real time (seconds) taken by each run. Finally, the Return and Success tabs show the total expected discounted reward and the success rate respectively of the final learned policy (both measured by policy rollout from the initial state). Each run is evaluated by taking the average performance of 100 independent policy rollouts using the learned policy starting the Acrobot at position $(0, 0, 0, 0)$ for $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ and ending when successfully reaching the goal in less than 3000 steps or after 3000 steps of simulation with no success. Trajectories in which the Acrobot fails to reach the goal within 3000 simulation steps are considered unsuccessful.

Consider first the two RCPI- algorithms with the full count of 200 uniformly distributed rollout states in each iteration. While both algorithms exhibit good learning performance (Return), the computational cost is high, as indicated by the Simulation and Time tabs. Moving on to the DRCPI- variations, we use a set of 200 (full count) uniformly distributed rollout states only in the first iteration and a set of 40 (low count) rollout states from directed sampling afterwards. Clearly, the DRCPI- variations yield significant savings in terms of Rollouts, Simulation, Attempts, and Time (only for RVM) compared to the RCPI- algorithms with the full count, while delivering policies of comparable performance (Return) in most cases. Finally, to appreciate the value of directed sampling, one can consider the RCPI- variations using a low count of only 40 uniformly distributed rollout states throughout all iterations. It is clear that in this case performance deteriorates in both Return and Success (especially for SVM), implying that the proposed focused (directed) selection of rollout states plays a significant role in performance, when the rollout/simulation budget is low.

Comparing policy Return, which is the metric optimized by learning, RVM versions have a significant advantage over the corresponding SVM versions in all cases. The Return delivered by the DRCPI variants is not too far from those delivered by RCPI with the full count of rollout states and better compared to those delivered by RCPI with the low count. Nevertheless, all DRPCI variants exhibit lower simulation requirements and execution times compared to RCPI with full count and compare favorably to RCPI with low count. Finally, there is a clear difference in the number of steps to reach the goal with the energy based reward being superior to the classic one.

For the energy shaping reward we provide figures 7.45 through 7.52 include. For each algorithm we give three histograms displaying the distribution of the corresponding 200 runs in terms of: (a) the policy return values, (b) the percentage of successful trails, and (c) the number of average steps per run for successfully reaching the goal only for successful trials.

Table 7.10: Acrobot with energy based reward, (200/40, attempts 4), collective results of 200 runs and comparison of algorithms in terms of computational and learning performance

Algorithm	States	Simulation	Rollouts	Attempts	Time	Return	Success	Steps
RCPI-SVM	200	3154392	1553	7.8	49.8	42.080	98.19%	179
RCPI-RVM	200	3357657	1656	8.3	8.7	88.109	99.00%	119
DRCPI-AIV-SVM	200/40	1119830	460	7.5	16.0	25.221	97.34%	257
DRCPI-IS-SVM	200/40	987650	448	7.2	10.2	23.579	97.21%	261
DRCPI-AIV-RVM	200/40	921737	427	6.7	4.2	56.138	97.57%	194
DRCPI-IS-RVM	200/40	925931	416	6.4	4.6	42.235	97.00%	246
RCPI-SVM	40	649828	320	8.0	6.8	19.640	98.00%	294
RCPI-RVM	40	683208	334	8.3	4.7	31.746	96.00%	275

Table 7.11: Acrobot with classic reward, (300/120, attempts 4), collective results of 200 runs and comparison of algorithms in terms of computational and learning performance

Algorithm	States	Simulation	Rollouts	Attempts	Time	Return	Success	Steps
RCPI-SVM	300	5746237	2925	8.1	64.8	0.018965	98.43%	316
RCPI-RVM	300	5203445	2610	7.2	11.5	0.024965	96.67%	372
DRCPI-AIV-SVM	300/120	4244549	1373	8.2	42.9	0.013555	98.12%	355
DRCPI-IS-SVM	300/120	2611023	1239	7.2	20.4	0.012617	98.01%	387
DRCPI-AIV-RVM	300/120	2243563	1006	6.0	6.5	0.008405	93.37%	580
DRCPI-IS-RVM	300/120	2418309	1165	6.2	6.7	0.010783	95.96%	472
RCPI-SVM	120	1882391	932	7.8	13.0	0.008467	97.14%	430
RCPI-RVM	120	1482727	720	6.0	5.1	0.005368	91.66%	834

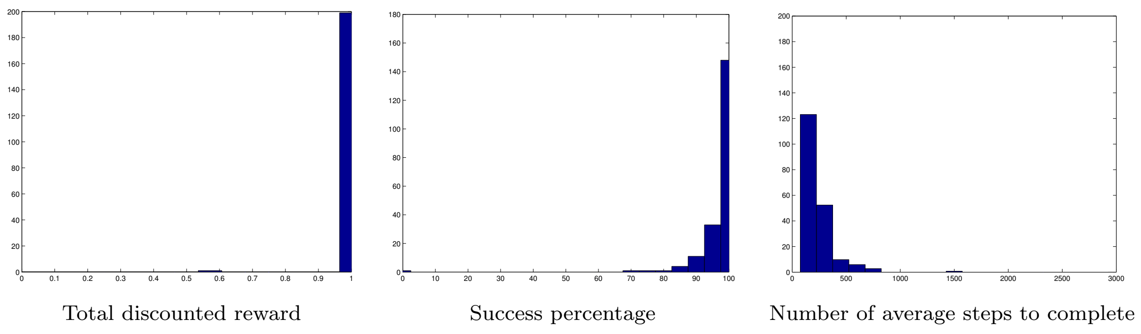


FIGURE 7.45: Acrobot using DRCPI-AIV-SVM.

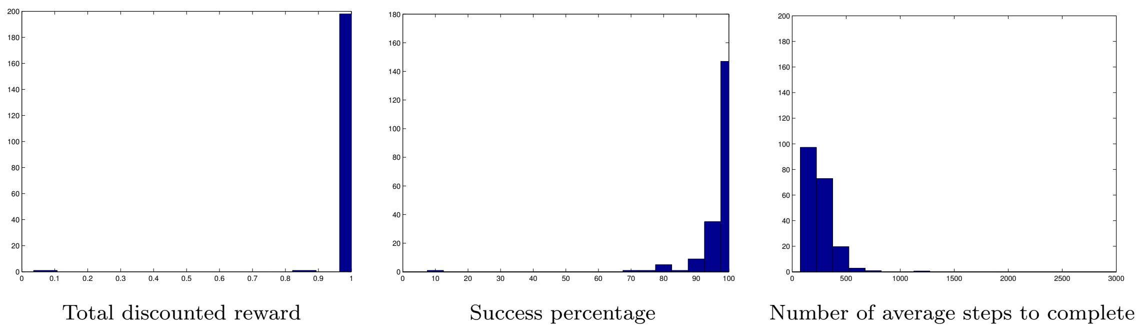


FIGURE 7.46: Acrobot using DRCPI-IS-SVM.

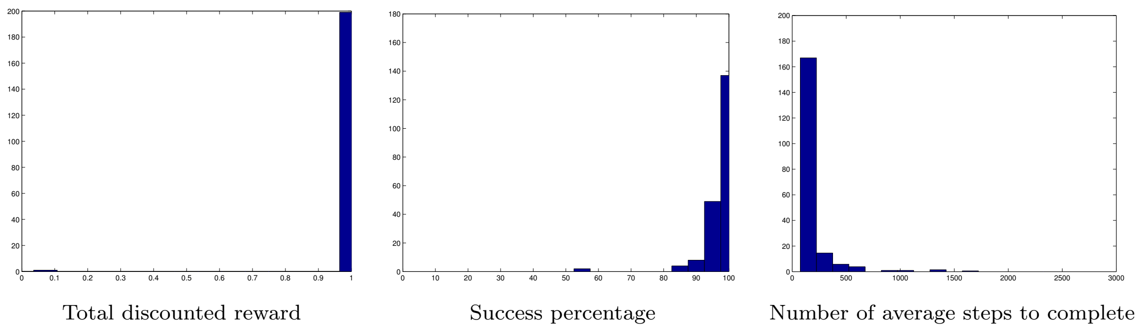


FIGURE 7.47: Acrobot domain using DRCPI-AIV-RVM.

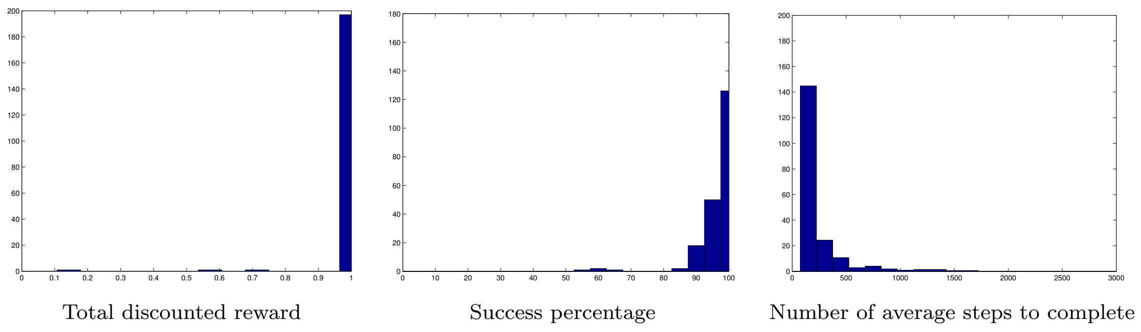


FIGURE 7.48: Acrobot using DRCPI-IS-RVM.

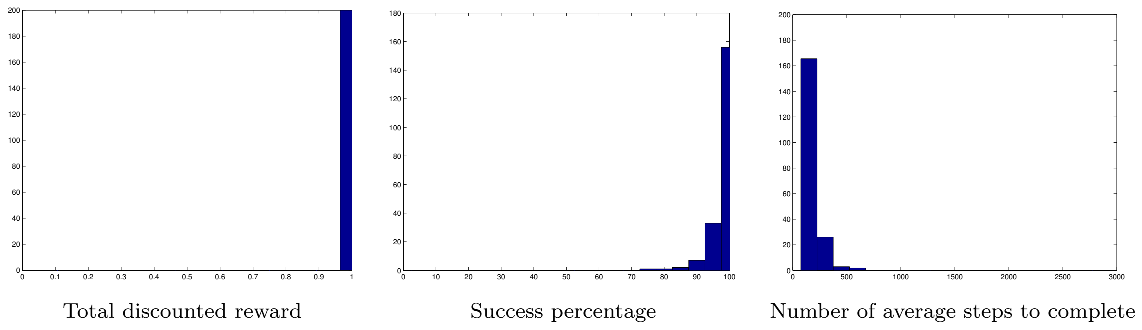


FIGURE 7.49: Acrobot using RCPI-SVM with a full count of rollout states.

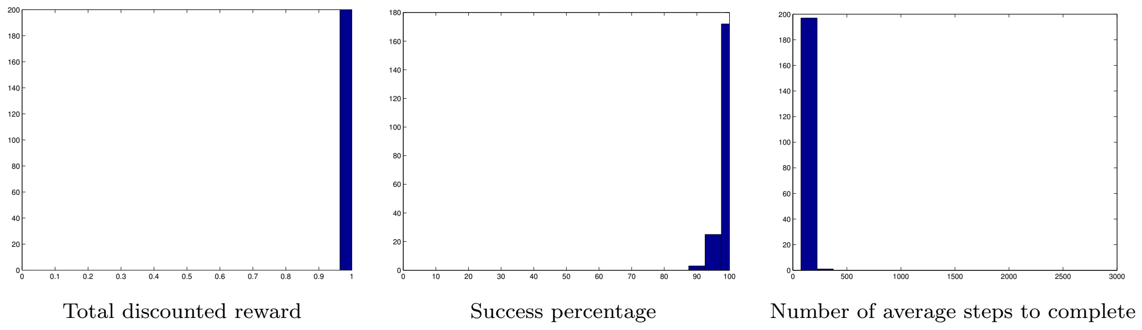


FIGURE 7.50: Acrobot using RCPI-RVM with a full count of rollout states.

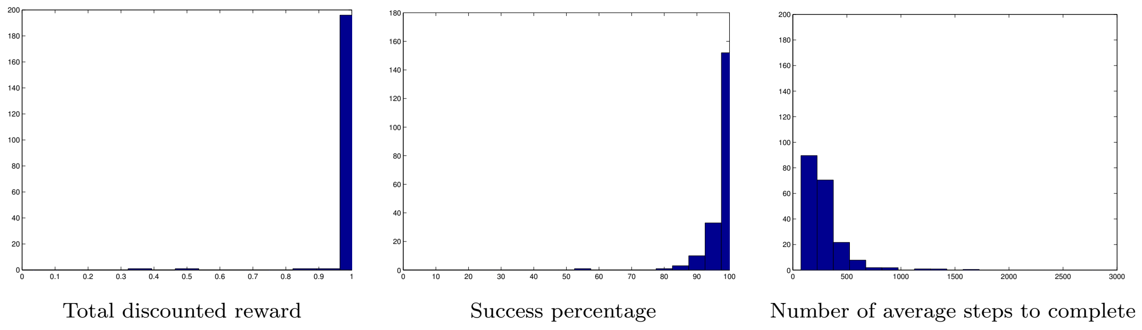


FIGURE 7.51: Acrobot using RCPI-SVM with a low count of rollout states.

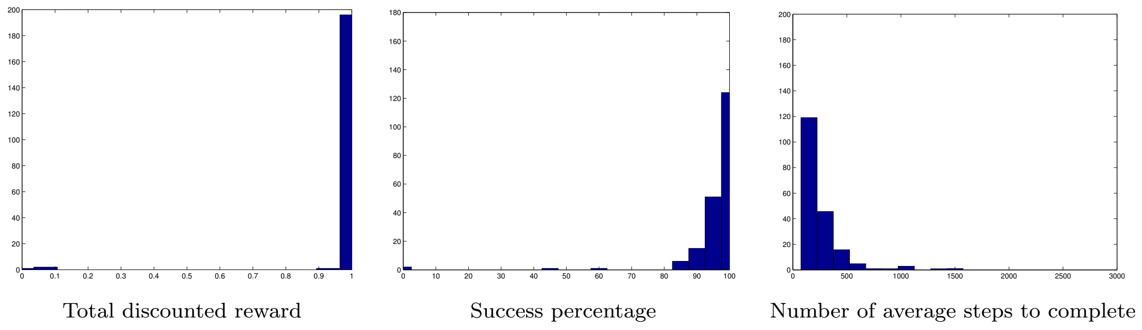


FIGURE 7.52: Acrobot using RCPI-RVM with a low count of rollout states.

Table 7.12: DRCPI parameters for 4-link Planar Robot domain

Symbol	DRCPI-AIV	DRCPI-IS	RCPI	Description	4-link Planar Robot
U	✓	✓		initial sample size for uniform sampling	100
M	✓	✓		subsequent sample size constructed using previous policy hints	20
L	✓	✓	✓	number of attempts to improve previous policy for a given iteration	4
K	✓	✓	✓	trials - the number of rollouts used to estimate $Q(s, a)$ values for a given state s	50
H	✓	✓	✓	horizon - number of steps per rollout	100
Z		✓		number of particles	$20 \cdot M$ (i.e. 400)
Σ		✓		covariance matrix Σ used in resampling using particles	diag(0.2)
U_U			✓	uniform sample size for RCPI, for all steps	U (i.e. 100)
U_{UL}			✓	uniform low sample size for RCPI for all steps	M (i.e. 20)
				Policy Assessment *(values are used to estimate the efficiency of the policy)*	
K_{test}	✓	✓	✓	trajectories - number of rollouts	100
H_{test}	✓	✓	✓	horizon - number of steps per rollout	500

Table 7.13: Library parameters for 4-link Planar Robot domain

		LibSVM		SparseBayesV2
Procedure	Algorithms used	β	C	β
Classification	DRCPI-AIV, DRCPI-IS, RCPI	$1/16$	64	$1/2$
Regression	DRCPI-IS	$1/30$	2	$1/4$

7.2.4 4-Link Planar Robot

The values used in the experiments with the 4-Link Planar Robot are reported in Table 7.12 and the library parameters in Table 7.13. The initial policy π_0 was a random deterministic policy.

In this section, we provide statistics for the 4-Link Planar Robot. Each row in Table 7.14 represents averages of 200 independent runs with identical settings, but with different random seeds, for each algorithm shown in Table 7.1. The

Simulation tab shows the total number of simulation steps needed for each run, while the Rollouts tab shows the total number of rollouts executed in each run. The Attempts tab shows the number of improvement attempts (the number of iterations is less than or equal to that) before termination and the Time tab shows the real time (seconds) taken by each run. Finally, the Return and Success tabs show the total expected discounted reward and the success rate respectively of the final learned policy (both measured by policy rollout from the initial state). Each run is evaluated by taking the average performance of 100 independent policy rollouts using the learned policy starting the 4-Link Planar Robot at position $(\pi, \pi, \pi, \pi, 0, 0, 0, 0)$ for $(\theta_1, \theta_2, \theta_3, \theta_4, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4)$ and ending when successfully reaching the goal in less than 500 steps or after 500 steps of simulation with no success. Trajectories in which the 4-link Planar Robot fails to reach the goal within 500 simulation steps are considered unsuccessful.

Consider first the two RCPI- algorithms with the full count of 100 uniformly distributed rollout states in each iteration. While both algorithms exhibit good learning performance (Return), the computational cost is high, as indicated by the Simulation and Time tabs. Moving on to the DRCPI- variations, we use a set of 100 (full count) uniformly distributed rollout states only in the first iteration and a set of 20 (low count) rollout states from directed sampling afterwards. Clearly, the DRCPI- variations yield significant savings in terms of Rollouts, Simulation, Attempts, and Time compared to the RCPI- algorithms with the full count, while delivering policies of comparable performance (Return) in most cases. Finally, to appreciate the value of directed sampling, one can consider the RCPI- variations using a low count of only 20 uniformly distributed rollout states throughout all iterations. It is clear that in this case performance deteriorates in both Return and Success, implying that the proposed focused (directed) selection of rollout states plays a significant role in performance, when the rollout/simulation budget is low.

Comparing policy Return, which is the metric optimized by learning, RVM versions have a significant advantage over the corresponding SVM versions in all cases. Finally, the Return delivered by the DRCPI variants is not too far from those delivered by RCPI with the full count of rollout states and better compared to those delivered by RCPI with the low count. Nevertheless, all DRPCI variants exhibit lower simulation requirements and execution times compared to RCPI with full count and compare favorably to RCPI with low count.

Figures 7.53 through 7.60 include for each algorithm three histograms displaying the distribution of the corresponding 200 runs in terms of: (a) the policy return values, (b) the percentage of successful trails, and (c) the number of average steps per run for successfully reaching the goal only for successful trials.

Table 7.14: 4-link Planar Robot (100/20, attempts 4), collective results of 200 runs and comparison of algorithms in terms of computational and learning performance

Algorithm	States	Simulation	Rollouts	Attempts	Time	Return	Success
RCPI-SVM	100	11393856	884	8.8	307.9	1.288465	93.41%
RCPI-RVM	100	11226298	888	8.9	222.4	1.293182	96.45%
DRCPI-AIV-SVM	100/20	3482916	258	8.8	75.8	1.286796	93.37%
DRCPI-IS-SVM	100/20	3525170	263	9.1	75.9	1.285163	92.46%
DRCPI-AIV-RVM	100/20	3423388	257	9.0	65.7	1.290313	95.01%
DRCPI-IS-RVM	100/20	3508169	273	9.6	67.2	1.293007	95.99%
RCPI-SVM	20	2116878	162	8.1	45.9	1.273024	89.61%
RCPI-RVM	20	2253847	176	8.8	45.0	1.286189	94.87%

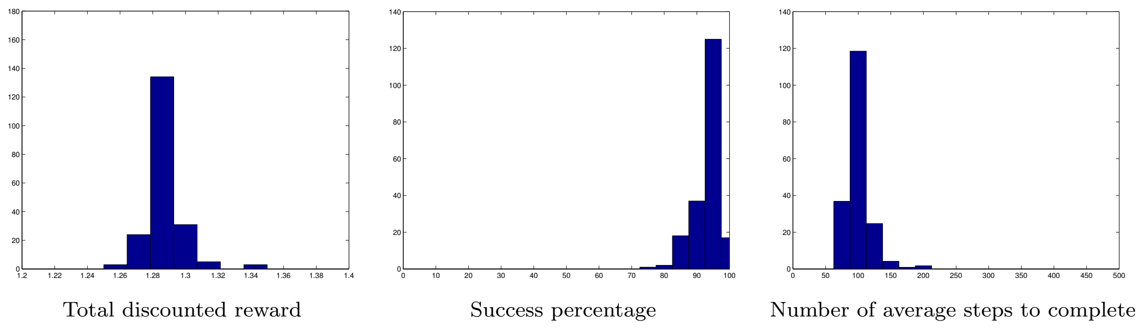


FIGURE 7.53: 4-link Planar Robot using DRCPI-AIV-SVM.

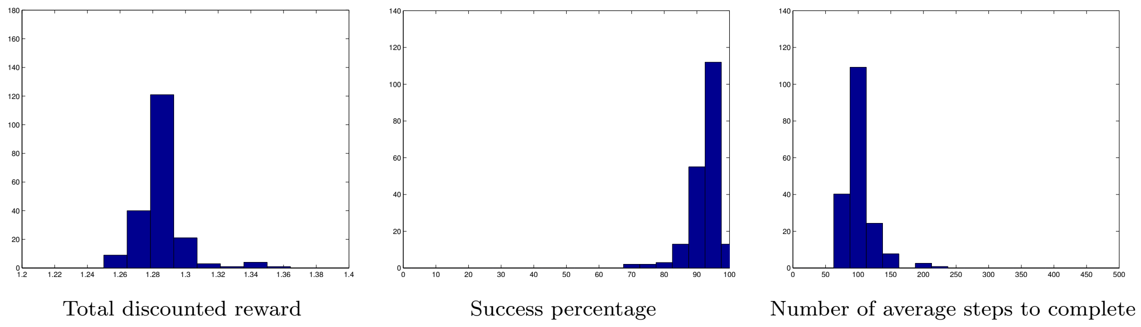


FIGURE 7.54: 4-link Planar Robot using DRCPI-IS-SVM.

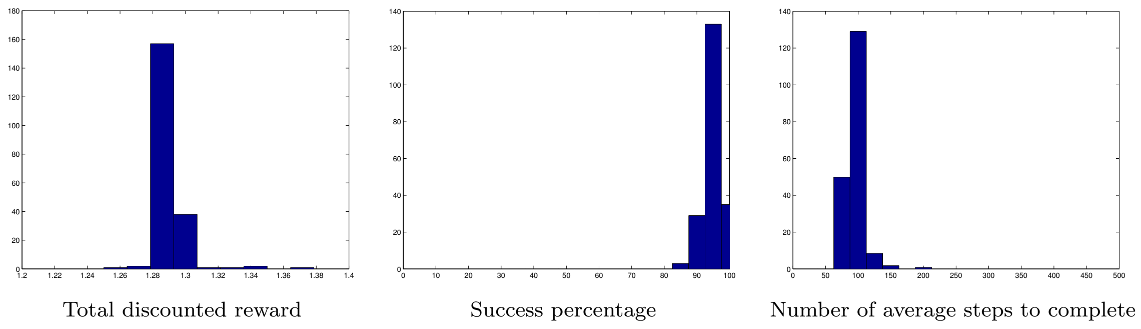


FIGURE 7.55: 4-link Planar Robot domain using DRCPI-AIV-RVM.

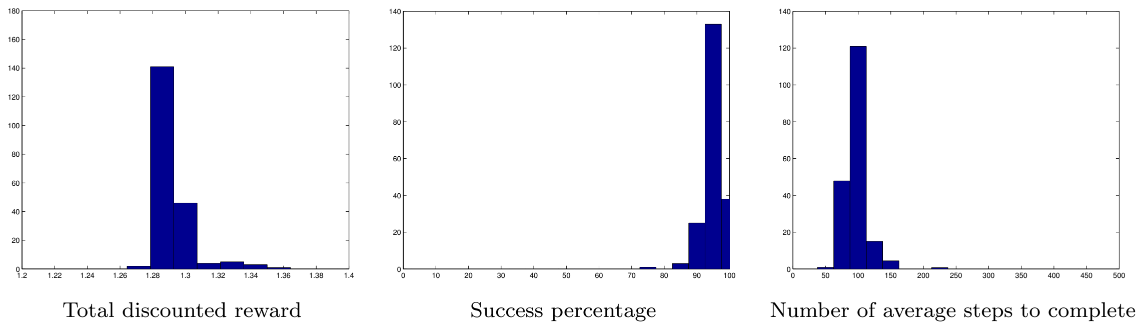


FIGURE 7.56: 4-link Planar Robot using DRCPI-IS-RVM.

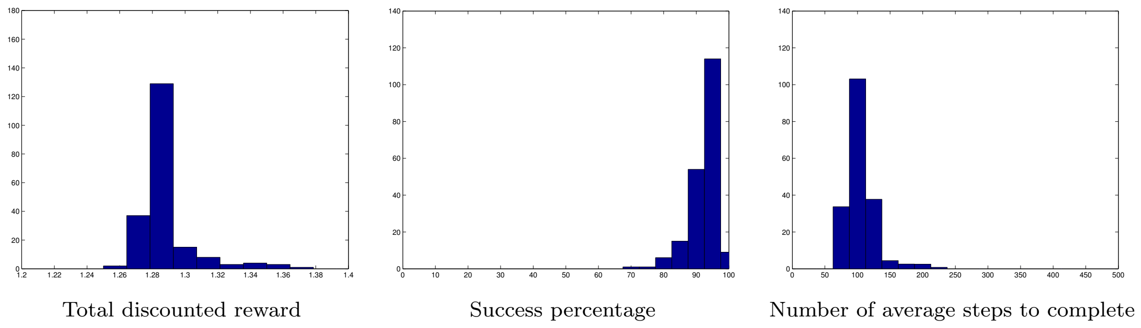


FIGURE 7.57: 4-link Planar Robot using RCPI-SVM with a full count of rollout states.

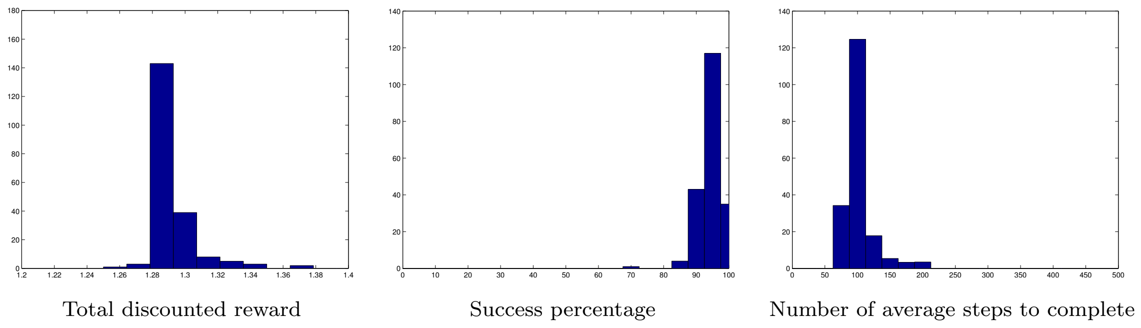


FIGURE 7.58: 4-link Planar Robot using RCPI-RVM with a full count of rollout states.

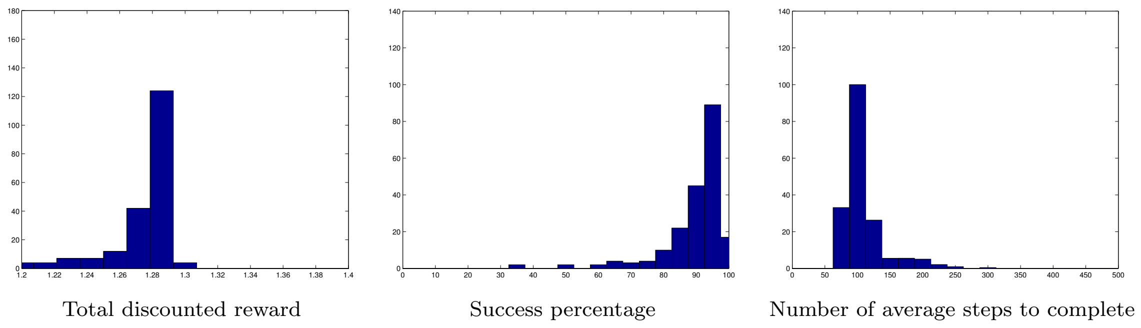


FIGURE 7.59: 4-link Planar Robot using RCPI-SVM with a low count of rollout states.

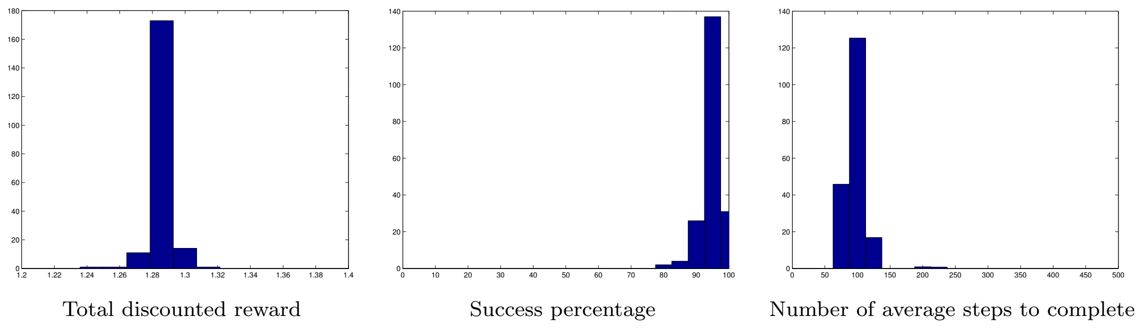


FIGURE 7.60: 4-link Planar Robot using RCPI-RVM with a low count of rollout states.

8

Conclusion and Future Work

In this dissertation we studied extensions of Rollout Classification Policy Iteration, a class of reinforcement learning algorithms that do not use explicit value function representation. These algorithms skip the difficult to approximate and possibly discontinuous value function and they learn good policies directly through rollouts (simulation). Rollouts are used to probe the improved policy at selected points in state space by repeatedly executing the current policy. Policies are generally greedy and deterministic and, thus, they can be represented by classifiers. A classifier representing a policy maps states into dominant actions (classes). The classifier training set at each iteration of policy improvement consists of (*state*, *dominant action*) pairs obtained using policy rollouts for estimating the return of the current policy from selected states and subsequently identifying the dominant action (if any).

8.1 Contributed Work

Our initial contribution was to uncover the structure that exists in optimal policies by deriving optimal policies for two standard two-dimensional reinforcement learning domains. We found that optimal policies have significant structure and a high degree of locality, i.e. dominant actions persist over large continuous areas within the state space. This observation provides sufficient justification for the appropriateness of classifiers for approximate policy representation.

Then, we focused our research on how to identify critical parts of the state space where there are changes in action domination. A change in action domination is represented by the separating border of a binary classifier. The border separates an action from all other actions. We use a collection of binary classifiers, one for each action, to form a multiclass classifier to represent a policy. Our aim is minimize the use of policy rollouts in our quest for an improved policy, by using the already caught policy structure. We developed two methods for directed exploration of policy space. The first one exploits the structure of the classifiers used for policy representation. The second one uses a state importance function based on action prevalence. In both approaches, the search is focused on areas where there is change of action domination. This directed focus on critical parts of the state space iteratively leads to refinement and improvement of the underlying policy and delivers excellent control policies in only a few iterations with a relatively small rollout budget.

8.2 Future Work

Future work may be applied to several directions. A challenging extension of DRCPI would be its adaptation for domains with continuous actions. Apparently, in such domains, multiclass or binary classifiers cannot be used directly for policy representation. Nevertheless, our work on DRCPI can be combined with the work of Pazis and Lagoudakis (2009), whereby a continuous action policy is approximated to any desired accuracy using a series of binary decisions for each continuous action choice. These binary decisions can be made by a binary policy represented using binary classifier(s) and therefore DRCPI could be used to learn such policies efficiently.

Another direction of future work, which may eliminate the need for the attempts loop in our algorithms, is to adopt a stepwise update during policy improvement. In particular, at each policy iteration the resulting policy will be derived as a combination of two policies, partly from the old policy and partly from the new one, combined in a way that guarantees improvement. Such an update can help eliminate possible performance discontinuities, which dictated the use of attempts in our work. However, under such "mixed" representations each policy will be represented by a collection of classifiers, which will require careful management.

Another possible direction of future work is to exploit online classification methods for incremental policy refinement. Under this idea, there is only a single policy representation by a classifier, which is gradually refined at each step by feeding the online training algorithm with additional training data. The benefit of this approach is the granular processing and exploitation of any additional information and the smooth policy improvement. Our directed exploration methods are still relevant in this context, since the increment policy refinement can be directed to focus at selective areas over the state space at each step.

Finally, the recent explosion of deep learning technologies (Mnih et al., 2015) both for classification and regression opens new future research directions. Our first approach, based on active input vectors, depends on exploiting the internal structure of the SVM and RVM classifiers, therefore cannot be combined easily with deep learning. However, our second approach, based on importance sampling, only requires the use of a general-purpose classifier and a general-purpose regressor; in this case, combining our work with deep learning classifiers and regressors is straight-forward. The expressiveness and efficiency of such technology could enhance the effectiveness of policy representation provided by our algorithms.

8.3 Epilogue

Our investigation of structure within policy representations produced many positive results. It allowed us to uncover this structure for the benefit of policy improvement. We introduced two novel reinforcement learning approaches that exploit iteratively the known policy structure while uncovering it, in the quest for an improved, and ideally optimal, policy. This dissertation sheds some light to the problem of learning by bringing together two large areas, namely reinforcement learning and supervised learning; our small contribution of scientific knowledge to the society.

Bibliography

M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25 in Automation and Remote Control,, pages 821–837, 1964.

Charles W. Anderson. Approximating a policy can be easier than approximating a value function. Technical Report CS-00-101, Department of Computer Science, Colorado State University, 2000.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002. ISSN 0885-6125.

J. A. Bagnell, Sham M Kakade, Jeff G. Schneider, and Andrew Y. Ng. Policy search by dynamic programming. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 831–838. MIT Press, 2004.

Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81– 38, 1995. ISSN 0004-3702.

Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957a.

Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957b.

K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

J. O. Berger. *Statistical decision theory and Bayesian analysis.* Springer, second edition, 1985.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II.* Athena Scientific, 3rd edition, 2007. ISBN 1886529302, 9781886529304.

Dimitri P. Bertsekas and Steven E. Shreve. *Stochastic optimal control : the discrete time case.* Mathematics in Science and Engineering. Elsevier Science, 1978. ISBN 9780080956480.

Dimitri P Bertsekas and Steven E Shreve. Existence of optimal stationary policies in deterministic optimal control. *Journal of Mathematical Analysis and Applications*, 69(2):607–620, 1979. ISSN 0022-247X.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT)*, pages 144–152, 1992.

Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, Mar 1996. ISSN 1573-0565.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines, 2001. Available at `http://www.csie.ntu.edu.tw/$\sim$cjlin/libsvm`.

Weiwei Cheng, Johannes Fürnkranz, Eyke Hüllermeier, and Sang-Hyeun Park. Preference-based policy iteration: Leveraging preference learning for reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I*, pages 312–327, 2011.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

G. DeJong and M. W. Spong. Swinging up the acrobot: an example of intelligent control. In *American Control Conference, 1994*, volume 2, pages 2158–2162 vol.2, June 1994.

Thomas G. Dietterich and Xin Wang. Batch value function approximation via support vectors. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1491–1498, 2001.

Christos Dimitrakakis and Michail G. Lagoudakis. Rollout sampling approximate policy iteration. *Machine Learning*, 72(3):157–171, 2008a.

Christos Dimitrakakis and Michail G. Lagoudakis. Algorithms and bounds for rollout sampling approximate policy iteration. In *Proceedings of the 8th European Workshop on Reinforcement Learning (EWRL)*, pages 27–40, 2008b.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice.* Springer Verlag, 2001.

Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and

Vladimir Vapnik. Support vector regression machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, December 2005. ISSN 1532-4435.

Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of Machine Learning Research*, 7:1079–1105, 2006.

Amir-massoud Farahmand. Action-gap phenomenon in reinforcement learning. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 172–180. Curran Associates, Inc., 2011.

Amir-massoud Farahmand, Doina Precup, and Mohammad Ghavamzadeh. Generalized classification-based approximate policy iteration. In *Proceedings of European Workshop on Reinforcement Learning (EWRL), 2012*, 2012.

Amir-massoud Farahmand, Doina Precup, André da Motta Salles Barreto, and Mohammad Ghavamzadeh. Capi: Generalized classification-based approximate policy iteration. In *Reinforcement Learning Decision Making (RLDM), 2013*, 2013.

Amir Massoud Farahmand, Doina Precup, André da Motta Salles Barreto, and Mohammad Ghavamzadeh. Classification-based approximate policy iteration: Experiments and extended discussions. *CoRR*, abs/1407.0449, 2014.

Amir Massoud Farahmand, Doina Precup, André da Motta Salles Barreto, and Mohammad Ghavamzadeh. Classification-based approximate policy iteration. *IEEE Transactions on Automatic Control*, 60(11):2989–2993, Nov 2015. ISSN 0018-9286.

Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 847–854. MIT Press, 2004.

Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006.

Alan Fern, SungWook Yoon, and Robert Givan. Reinforcement learning in relational domains: A policy-language approach. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*, Adaptive Computation and Machine Learning, chapter 18, pages 499–533. The MIT Press, 2007. ISBN 0262072882.

Johannes Fürnkranz, Eyke Hüllermeier, Weiwei Cheng, and Sang-Hyeun Park.

Preference-based reinforcement learning: A formal framework and a policy iteration algorithm. *Machine Learning*, 89(1-2):123–156, 2012.

Victor Gabillon, Alessandro Lazaric, and Mohammad Ghavamzadeh. Rollout allocation strategies for classification-based policy iteration. In *Proceedings of the ICML Workshop on Reinforcement Learning and Search in Very Large Spaces*, 2010.

Victor Gabillon, Alessandro Lazaric, Mohammad Ghavamzadeh, and Bruno Scherrer. Classification-based policy iteration with a critic. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 1049–1056, 2011.

Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Michael Booth, and Fabrice Rossi. *Gnu Scientific Library: Reference Manual.* GNU, 2003.

Isabelle Guyon, Bernhard E. Boser, and Vladimir Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 147–155, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-274-7.

Tommi S. Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *IN PROC. 19TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pages 267–274, 2002.

Michail G. Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 424–431, 2003a.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003b.

John Langford and Bianca Zadrozny. Relating reinforcement learning performance to classification performance. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pages 473–480, 2005.

Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Analysis of a classification-based policy iteration algorithm. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 607–614, 2010.

Lihong Li, Vadim Bulitko, and Russ Greiner. Focus of attention in sequential decision making, 2004.

Lihong Li, Vadim Bulitko, and Russell Greiner. Focus of attention in reinforcement learning. *Journal of Universal Computer Science*, 13(9):1246–1269, sep 2007.

David J.C. MacKay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.

David J.C. MacKay. Bayesian non-linear modelling for the prediction competition. In *In ASHRAE Transactions, V.100, Pt.2*, pages 1053–1062. ASHRAE, 1994.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

Ian T. Nabney. Efficient training of rbf networks for classification. In *Ninth International Conference on Artificial Neural Networks (ICANN99)*, volume 1, pages 210–215, 1999.

Radford M. Neal. *Bayesian Learning for Neural Networks.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996. ISBN 0387947248.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer, 2006.

Jason Pazis and Michail G. Lagoudakis. Binary action search for learning continuous-action control policies. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 793–800, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553476.

M.J.D. Powell. A Fortran subroutine for solving systems of nonlinear algebraic equations. In Philip Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*, chapter 7, pages 115–161. Gordon and Breach, Science Publishers Ltd., 1970.

Martin L. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming.* John Wiley and Sons, Inc., 1994.

Emmanuel Rachelson and Michail G. Lagoudakis. On the locality of action domination in sequential decision making. In *Proceedings of the 11th International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2010.

Ioannis Rexakis and Michail G. Lagoudakis. Classifier-based policy representation. In *Machine Learning and Applications, 2008. ICMLA '08. Seventh International Conference on*, pages 91–98, Dec 2008.

Ioannis Rexakis and Michail G. Lagoudakis. Directed exploration of policy space using support vector classifiers. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 112–119, April 2011.

Ioannis Rexakis and Michail G. Lagoudakis. Directed policy search using relevance vector machines. In *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*, volume 1, pages 25–32, Nov 2012.

Ioannis Rexakis and Michail G. Lagoudakis. Directed policy search for decision making using relevance vector machines. *International Journal on Artificial Intelligence Tools*, 23(4), 2014.

Ken F. Riley, Michael P. Hobson, and Stephen J. Bence. *Mathematical Methods for Physics and Engineering.* Cambridge University Press, 2006.

GA Rummery and M Niranjan. On-line q-learning using connectionist systems. techreport, University of Cambridge, UK, 1994.

Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

Mark W. Spong. Swing up control of the acrobot. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2356–2361. Publ by IEEE, 12 1994. ISBN 0818653329.

Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, Massachusetts, 1998.

Gerald Tesauro and Gregory R. Galperin. On-line policy improvement using monte-carlo search. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1068–1074. MIT Press, 1997.

M. E Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, pages 211–244, 2001.

M. E. Tipping and Anita Faul. Fast marginal likelihood maximisation for sparse Bayesian models. In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, pages 3–6, 2003.

M.E. Tipping. Sparse Bayes V2: a library for relevance vector machines, 2009.

John N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. In *Machine Learning*, pages 185–202, 1994.

John N. Tsitsiklis and Benjamin Van Roy. Average cost temporal-difference learning. *Automatica*, 35(11):1799–1808, 1999. ISSN 0005-1098.

Dimitris G Tzikas, Liyang Wei, Aristidis Likas, Yongyi Yang, and Nikolas P

Galatsanos. A tutorial on relevance vector machines for regression and classification with applications. *EURASIP News Letter*, 17(2):4, 2006.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94559-8.

Vladimir N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, 1998.

Hua O. Wang, Kazuo Tanaka, and Michael F. Griffin. An approach to fuzzy control of nonlinear systems: Stability and design issues. *IEEE Transactions on Fuzzy Systems*, 4(1):14–23, 1996.

C. J. C. H. Watkins. *Learning from Delayed Rewards.* phdthesis, King's College, Oxford, UK, 1989.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

RJ Williams and LC Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, College of Computer Science, Northeastern University, Boston, Massachusetts, 1993.

Xin Xin and Yannian Liu. *Control Design and Analysis for Underactuated Robotic Systems.* Springer Publishing Company, Incorporated, 2014. ISBN 1447162501, 9781447162506.

SungWook Yoon, Alan Fern, and Robert Givan. Inductive policy selection for first-order mdps. In *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, pages 568–576, 2002.

Huizhen Yu and Dimitri P. Bertsekas. Error bounds for approximations from projected linear equations. *Math. Oper. Res.*, 35(2):306–329, 2010.

Publications

- Journal Publications
  - Ioannis Rexakis and Michail G. Lagoudakis: Directed Policy Search for Decision Making Using Relevance Vector Machines , *International Journal on Artificial Intelligence Tools*,Volume 23, Issue 04, August 2014.
- Conference Publications
  - Ioannis Rexakis and Michail G. Lagoudakis, Classifier-based policy representation, *Proceedings of the 7th IEEE International Conference on Machine Learning and Applications (ICMLA), 2008, pp. 91-98.*
  - Ioannis Rexakis and Michail G. Lagoudakis, Directed exploration of policy space using support vector classifiers, *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2011, pp. 112-119.*
  - Ioannis Rexakis and Michail G. Lagoudakis, Directed Policy Search using Relevance Vector Machines, *Proceedings of the 2012 IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Athens, Greece, November 2012.*

    Best Student Paper Award.