LEARNING DECISIONS: ROBUSTNESS, UNCERTAINTY, AND APPROXIMATION

J. Andrew Bagnell

CMU-RI-TR-04-67

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

August 2004

Submitted in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Thesis Committee:
Jeff Schneider, Chair
Andrew Moore
Alfred Rizzi
Thomas Dietterich, Oregon State University

ABSTRACT

Decision making under uncertainty is a central problem in robotics and machine learning. This thesis explores three fundamental and intertwined aspects of the problem of learning to make decisions.

The first is the problem of uncertainty. Classical optimal control techniques typically rely on perfect state information. Real world problems never enjoy such conditions. Perhaps more critically, classical optimal control algorithms fail to degrade gracefully as this assumption is violated. Closely tied to the problem of uncertainty is that of approximation. In large scale problems, learning decisions inevitably requires approximation. The difficulties of approximation inside the framework of optimal control are well-known. [Gordon, 1995]

Often, especially in robotics applications, we wish to operate learned controllers in domains where failure has relatively serious consequences. It is important to ensure that decision policies we generate are robust both to uncertainty in our models of systems and to our inability to accurately capture true system dynamics.

We present new classes of algorithms that gracefully handle uncertainty, approximation, and robustness. We pay attention to the computational aspects of both the problems and algorithms developed. Finally, we provide case studies that serve as both motivation for the techniques as well as illustrate their applicability.

ACKNOWLEDGEMENTS

It seems customary for a thesis to contain a list of acknowledgements that is both short and sweet. I can hope for the latter, but I am in debt to too many and too deeply for the former.

I first thank my committee for both their interest and patience in my research. I hope some of it is worth their while.

I thank my classmates, colleagues and friends at the Robotics Institute who helped make Carnegie Mellon such an enriching experience. I have heard it said that the worst day at Carnegie Mellon is better than the best day elsewhere, and it just might be true. I've been far too lucky with friends to name them all here, but I must especially thank robo-friends Aaron Courville, Rosemary Emery, Nicholas Roy, David Tolliver, Joelle Pineau, Mike Montemerlo and Carl Wellington as well as Jenny Briggs and Kristine Roy for helping making my experience so enjoyable. Their close friendship and their thoughts on research (as well as on Bonsai Buffalo(TM)) have bettered my time in Pittsburgh. I also thank my officemates, especially Aaron Courville and Fernando Alfaro for putting up with crazy ideas and conversations.

I must also thank especially a few students who have profoundly influenced my thinking. I am deeply indebted to friends David Blei, Sham Kakade, John Langford, Andrew Ng, and Nicholas Roy for shaping my thoughts on reinforcement learning, sample complexity, and probabilistic inference. Our collaborations and chats gave me new appreciations for diverse issues. I also thank Sham and Nick for providing code and guidance for using it that was invaluable in my research.

I'm very grateful as well to many faculty at CMU. Sebastian Thrun reminds me just what a joy research should be. Andrew Moore graciously welcomed me into his lab and has always been a source of insights and ideas. I thank Chuck Thorpe for encouraging my research on learning and helicopter control when funding was tight. I thank Garth Zeglin for his generous help with walking robots. Chris Atkeson always reminds me to keep my eye on the prize. I thank Omead Amidi and the whole helicopter team for making helicopter experiments both possible and enjoyable.

I am very grateful to the staff at Carnegie Mellon who often went to great lengths to make things easier for me. I am especially grateful to Jean Harpley and Suzanne Muth.

I'd like to thank those who made graduate school a possibility for me as well. When I began as an undergraduate, being here today was essentially unthinkable to me. I remember a classmate, "John," asking me "What did I do in engineering?" I didn't even understand what he meant: what I *did* was go to class– on the good days. My encounter led me

to the Machine Intelligence Laboratory where Scott Jantz, Ivan Zapata, Michael Nechyba, Jenny Laine, Patrick O'Malley, A. Antonio Arroyo and others became great friends and great teachers. Most of all though, my mentorship by Keith Doty radically reshaped my understanding of academia and research. I learned it could truly be fun.

I am profoundly in debt to my advisor Jeff Schneider. Jeff truly fulfilled my needs as a student. When I needed a problem to work on, Jeff could always get me started. When I needed time and room to make progress on some area, Jeff gave me that as well. When I had an idea I thought worth pursuing, Jeff could quickly see its flaws, and occasionally, its merit. Jeff's clear thinking, and even more so, his knack for simple and lucid explanations are skills I wish I could fully learn from him. Not to mention that trick with the comb and the bottle-cap....

I also must thank my first advisors– my parents Bob and Judy. Without their patience and their pushes, their lessons and their love, none of the opportunities I've experienced would have been possible. I can never thank them enough.

Finally, I thank Nicolle, my wife. Nicolle has been with me through the whole graduate experience, and I've had the good luck to know her much longer. Nicolle has faith in me when I can't find it. She makes me laugh as no one else can. For her patience, laughter, and love I am eternally grateful.

TABLE OF CONTENTS

CHAPTER 1

Introduction

AUTOMATED decision making, in its various guises, is one of the primary aims of the fields of Robotics and Artificial Intelligence. For instance, much robotics work in recent years has considered the problem of autonomous helicopter control. Reliable autonomous control of helicopters would have profound implications in a variety of applications: it would allow for safer dam inspection, improved security around potential terrorist targets, and rapid response search-and-rescue. Later in this work (Chapter 10) we will elaborate on how current state-of-the-art helicopter controllers have been developed using learning control techniques.

Consider also problems of production scheduling. Decisions must be made regarding goods to produce, machines and labor to employ, repairs and tuning to perform, supplies to order and so on. These are decisions that must be made repeatedly and rapidly. The stakes are high as well: improved performance means greater profits and a higher standard of living for consumers of this producer. Learning good adaptive strategies for this problem is essential.

1.1. Characterization of the Problem

The examples described above (as well as throughout this work) share a number of points in common. Real-world decision making is rife with uncertainty. Agents operate with noisy effects; and action taken may lead to radically different states at one time than at another. Decisions are made under a great deal of uncertainty: with the rare exception of certain games, decision makers get no "God's eye view" of the world. Rather they see an incomplete and noisy characterization of the world in which they operate. FInally, agents

are not handed a complete description of the world. Rather, they learn to infer cause and effect relations in some approximate model of the world. Automated decision making on any large scale also requires approximation, even in problems as simple as board games, a complete policy would be unthinkably large. Careful considerations of the computational and information-theoretic limitations of a decision maker are in order.

1.2. Approaches

Attacks on the decision learning problem have largely been launched from the base-camp of stochastic optimal control. It has proven difficult to adopt classical optimal control techniques to the setting of limited information and particularly to the requirements of approximation in large-scale problems. Strong theoretical statement as well as good practical performance remains largely elusive despite a number of very encouraging early successes.

1.3. Approach of This Thesis

In this work, we take a more thorough- goingly machine learning approach to the problem of learning decision making policies. In particular, we focus on the requirements of approximation, uncertainty in state information, and robustness to modeling error. Each of these is intimately intertwined with the issue of learning and the problems faced by real decision makers.

1.4. Reinforcement Learning

The simulation-based learning of policies to optimize an expected reward is often referred to as Reinforcement Learning (RL). The work described here can be described as solving aspects of the RL problem. The early chapters, for instance, develop policy gradient methods that solve RL in a model-free setting. Later algorithms assume similar, but stronger access to the model [1]. Finally, the last chapters contribute to our understanding of model-based reinforcement learning where we first learn a model and then attempt to build controllers robust to inaccuracies in that model.

[1] See next chapter for details

1.5. Relation of Reinforcement Learning to Supervised Learning

The most common paradigm for learning is known as Supervised Learning (SL). In this model, a learner is handed input and output pairs and asked to make predictions based on these tuples. From the learning perspective, RL differs in two very significant ways: first, in Supervised Learning it is generally assumed a priori that different data points are independent and identically distributed. In RL, by contrast, decisions now influence the distributions of future observations seen. This represents a radical departure from the SL paradigm. Further, in SL, we are explicitly given the *correct* output. In RL, we only see an immediate reward for a particular action. We must *decide* which action is most advantageous in the long term. These two fundamental differences give rise to a variety of different aspects to the RL problem. These include the temporal credit assignment problem and the exploration-exploitation trade off. They also influence the applicability of techniques. For instance, there is no immediate way to blindly apply algorithms from supervised learning to the reinforcement learning problem. There are, however, important commonalities between the approaches, even beyond the obvious focus on approximation based on data. Our first set of algorithms, for instance, are stochastic gradient ascent algorithms, which have long been popular in the machine learning community. Later, we will consider algorithms that are based directly on inference (integration) techniques developed in the Bayesian machine learning community. Finally, we'll develop algorithms that use supervised learning methods as a type of oracle. The analysis techniques of supervised learning, including approximate integration, stochastic approximation, sample complexity, agnostic-PAC theory and convex optimization all have reflections as well. Some of these play important roles in our work.

1.6. Computation Complexity

As we are interested in automated decision making, we are invariably concerned with the computational tractability of the problems we are considering. There are multiple ways to formulate this. A popular one in the supervised learning community is information-based complexity- often referred to in the literature as "sample complexity." The idea is to consider the number of calls to a simulation. Kakade's [Kakade, 2003] recent thesis addresses just this issue. Sample complexity however does not fully capture our concern. The Pegasus algorithm, for instance, settles the sample complexity issue by providing a bound

to the number of samples [2] that grows only poly-logarithmically with the size of the policy space under consideration. Kakade though, notes: "Essentially, Pegasus is a variance reduction technique, not a means to solve exploration." We agree that Pegasus is a useful trick to reduce variance (it is used for instance in Chapter 10). However, Pegasus does formally solve the sample complexity problem in the sense of requiring a reasonable number of random numbers. Further, some algorithms work without sampling at all. (See, for instance, the exact algorithms for memoryless POMDPs as well as Derivative Propagation in this thesis.) We would argue that Kakade's objection is better captured by combinatorial complexity. At times we find it useful to refer to the combinatorial complexity relative to an oracle- for instance, a supervised learning algorithm called as a subroutine.

1.7. Contributions

This thesis make a number of contributions to the literature on learning to make decisions.

In chapter 4, we provide a novel gradient algorithm that allows the search through function spaces that define non-parametric policies. Further, we provide a new approach to using data-structures to speed up kernel machines in general, including their uses in supervised learning.

In chapter 5, we investigate the structure of the space of policies and develop *covariant* versions of policy gradient methods developed in previous chapters. These versions use information-theoretic notions of "closeness" to ensure that the resulting "test distribution" of examples after a gradient step closely matches the "training set" distribution from which the gradient was calculated. Further, these notions of closeness lead to parameterization independent algorithms.

The two previous chapters considered a very weak notion of simulative model and Monte-Carlo methods appropriate for it. In chapter 6 by contrast, we develop a novel deterministic approximation to the policy gradient. These methods rely on a much stronger notion of model– that of a probabilistic graphical model description of the system behavior. For this, however, we gain an efficient, simple, and very general gradient computation algorithm. This approach, termed *Derivative Propagation* extends [Minka, 2001] Expectation Propagation to compute gradients. It is applicable to the very wide context of probabilistic

[2] Sample here differs from our intuitive notion of such. Here it essentially refers to the number of different random numbers that are needed. Kakade's work reflects our more typical understanding of this concept.

inference as well as the special purpose of optimizing a policy. Both algorithm and analysis are novel.

In chapter 7, we develop the Policy Search by Dynamic Programming (PSDP) approach to learning policies. PSDP attempts to leverage reset interaction models and stronger prior information to create algorithms with stronger computational guarantees as well as good empirical performance.

The final three chapters of novel material deal with the issues of *robustness*. Robustness addresses uncertainty in the model of a system. The issue of mitigating model uncertainty is particularly important within the context of learning control as learned models are invariably data-starved and simplified for computational traction. In chapter 9, we take a novel set-based approach to uncertainty that naturally generalizes the ideas of stochastic and robust control that we term "stochastic robustness".

In chapter 10, by contrast, we take a Bayesian view of uncertainty. Both views are useful: the first provides computational benefits and the second naturally handles approximation.

1.8. Case studies

Each chapter with novel material presents at least one *case study.* These illustrate the theoretical contributions of the thesis by a variety of means. They provide some guidance regarding implementations and scalability of the approaches, give a measure of insight into the empirical properties of the algorithms, and finally serve to demonstrate some of the breadth of the Reinforcement Learning problem.

The problems presented very greatly in scope. Some, like the well-studied *Mountain Car* are classic benchmarks, while others like the helicopter control work presented in chapter 10 demonstrate a significant implementation and some real-world success of the techniques. We particularly thank Nicholas Roy for access to early CARMEN [Montemerlo *et al.*, 2003] code and to Sham Kakade for providing his implementation used in [Kakade, 2002] demonstrating the natural gradient on the Tetris problem. More information on benchmarks can be found in Appendix C.

CHAPTER 2

Formal Models for Sequential Decision Making

THIS thesis is concerned with the problem of how a system can choose decisions to act effectively. We rely on the framework of the Markov Decision Process and its extensions to model our decision making problem.

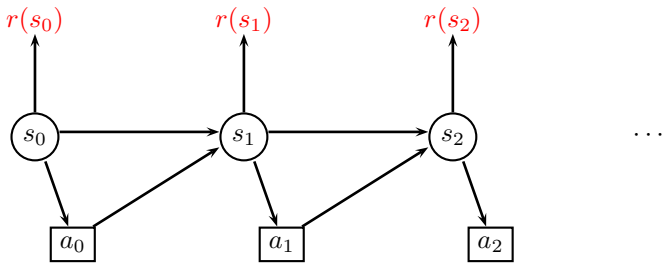
2.0.1. Markov Decision Processes

The Markov Decision Process (MDP) forms perhaps the most common model in control, operations research, and machine learning. In it, we assume our world can be described probabilistically. Further, we make the *Markov* assumption: that is, predictions of the future conditioned on the present state are independent of the past. Although for most of our discussion, we deal with the discrete state case, it extends naturally to continuous state (like the linear state-space models that dominate control theory).

Formally, in the MDP, we have a set of states from a space denoted $\mathcal{S}$ and controls denoted $\mathcal{A}$. We define a reinforcement function $r : \mathcal{S} \mapsto \mathbb{R}$ over the state space. In this work, we will generally assume (without any significant loss of generality) that r is bounded in $[0, 1]$. [1] r can also be extended to be a random variable, and to depend on the current state, control and next state. It is both notationally and pedagogically advantageous to ignore those potential generalities.

States are formally so in the sense that they obey the Markov property: s_{t+1} is conditionally independent of the history of the process given only the control and s_t, the previous state. In the language of graphical models [Jordan and Bishop, 2004] the MDP is a directed chain of states decorated with control and reinforcement nodes:

[1] This dispenses with the need for $R_{\min}$ and $R_{\max}$ constants floating throughout descriptions and proofs.



We use the notation $\xi = (s_0, a_0, s_1, a_1 \ldots s_{T-1})$ to refer to a path or trajectory the joint system and controller can take. We refer to the space of all such paths as Ξ and often speak of the probability distribution over paths induced by a particular controller $p(\xi|\pi)$. We denote by $P_{sa}(\cdot)$ the conditional probability distribution over next states given state s and action a.

The term "Markov Decision Problem" refers to the process as well as a performance criterion: the most common one is simply additive in time:

$$R_{path}(\xi) = \sum_{t=0}^{T-1} r(s_t) \tag{2.1}$$

We are interested in maximizing the expected reward of the system. We typically in this work will use the time averaged value instead of the straight sum:

$$R_{path}(\xi) = \frac{1}{T} \sum_{t=0}^{T-1} r(s_t) \tag{2.2}$$

Again, we choose this because it is convenient to keep rewards bounded in $[0, 1]$. Similarly, we define the time discounted reward as

$$R_{path}(\xi) = (1-\gamma) \sum_{t=0}^{T-1} \gamma^t r(s_t) \tag{2.3}$$

The discount factor can be justified as a mathematical convenience (to keep the value bounded), a smoother approximation of some finite-horizon, incentive to earn reward early, or an interest rate. It can also be seen as equivalent to simply adding a transition to a zero-reward absorbing state with probability $1-\gamma$ independently at each step.

Real problems with an indefinitely long horizon can be simulated with a sufficiently long horizon time T, and we will often consider the infinite-horizon time discounted problem. Convergence of the expected total reward integral is guaranteed for systems that

almost-surely reach a terminating state or terminate in finite-time, which includes, of course, the discounted case and finite horizon case. At times we consider the limiting cases where $T \Rightarrow \infty$ and the discount factor $\gamma \Rightarrow 1$. This is known as the infinite horizon average-reward problem. It most cases, it too can be approximated by policies with finite horizon of sufficient length[Putterman, 1994].

The standard approach to the solution of MDPs is to leverage results known as Bellman [Bellman, 1957] equations, which establish a relationship between the future values achieved starting from different initial states. The nonlinear Bellman equations for the infinte horizon time discounted problem are:

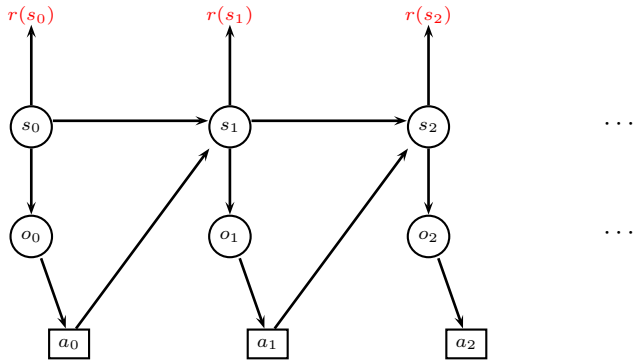
$$V(s) = \max_a (1-\gamma) r(s) + \gamma \mathrm{E}_{\mathrm{P_{sa}(s')}}[\mathrm{V(s')}] \tag{2.4}$$

The Bellman equations are either directly iterated as fixed point equations, or used in a linear program to solve directly for the optimal controller. An excellent elementary treatment of MDPs and the dynamic programming algorithms most commonly used in their solution can be found in [Putterman, 1994].

2.0.2. Partially Observed MDPs and other extensions

Although simple and understandable, the MDP does not capture enough of the problem we are interested in solving. The Markov Decision Process assumes that the exact system state is available to the controller at each time step. A policy then indicates which action to take from each state. In reality, state is "hidden" from a controller.

The Partially Observed Markov Decision Process (POMDP) or Controlled Hidden Markov Chain is a more realistic model and natural extension of the MDP where the underlying state structure is preserved while some random variable, measurable with respect to the state at each time, is observed. Below is depicted the graphical model description of a POMDP (given a memoryless controller):



It is well known that in Markov Decision Problems [Putterman, 1994] there are optimal memoryless policies. In the POMDP, however, total optimality requires memory– often a complete history of all previous observations and actions.

There are elegant structural properties to the optimal controller under the criterion of maximizing the time-additive (or discounted) reinforcement. Most importantly, POMDPs (*only* under this particular criterion) reduce to *belief-state MDPs*. That is, the Bayes-optimal filter's distribution over possible underlying states at each possible time step constitutes a sufficient statistic for control, and optimal controllers become simply memoryless policies mapping the output of this filter directly to the space of controls. This elegant result lies at the heart of the Kalman-filter/Linear Quadratic Regulator optimality (combined with the quadratic cost function). Despite this appeal, with the exception of special cases and tiny, toy discrete problems, exact POMDP solution techniques have thus far been computationally useless. Moreover, it is understood now that the problem is fundamentally a difficult one; in fact, in the average reward case (that is, where $T \rightarrow \infty$ and no discount factor), eminently tractable in the MDP case [Putterman, 1994]), finding the optimal control is simply non-computable.

A review of solution techniques can be found in [Murphy, 2000]. Many recent techniques, falling under the rubric of "policy-search" claim to be POMDP solution algorithms. While this is true, it is to some extent misleading. Many of these algorithms optimize controllers from some limited class applied to essentially any controlled stochastic process, and so the qualifier POMDP may be overly restrictive.

The bulk of this thesis takes the POMDP as fundamental. For those familiar with POMDP research, however, the approach taken in this thesis may be somewhat surprising. Most work in the POMDP setting attempts to solve problems as belief-state MDPs. While the realization that Bayes filters serve as sufficient statistics is an important one, this is only a small piece of the POMDP puzzle. In problems of any significant size, we cannot store or compute the belief state let alone compute plans that are optimal relative to it.[2] As soon as we use a form of sub-optimal filtering scheme, we in yet another (meta?) POMDP. Approximation equally forms another POMDP where the basis functions serve as deterministic observations of the underlying state (or more primitive observations). We certainly would not think of filtering the world relative to the basis functions!

In this sense, saying that we will deal with hidden state by simply using beliefs is a dodge. We believe that ultimately we are forced to confront the problem of mapping *observations* to *controls*. Certainly filters are a crucial part of interesting systems and provide us with the technology to reason about value-of-information and exploratory moves as well as smoothing noise. We, however, will treat a filter as a fixed part of the world that outputs a meta-observation. Again, the problem reduces to mapping an observation to actions. This is the problem we address throughout the thesis.

Unfortunately, even this more restricted notion of optimality leads in general to a very computationally hard problem. In Chapter 7 we will see that for a broad class of memoryless policies $\pi : o \rightarrow a$, finding the optimal policy is NP-hard.

2.1. Access Models

There are a variety of access assumptions we can make for the POMDP model. We present a set, in order of decreasing information content, that are used throughout the thesis. [3]

2.1.1. Explicit Probabilistic Description

Under this assumption, we are given complete tabular descriptions of transition matrices $P_{s,a}(\cdot)$, $p(o|s)$ and $r(s)$ as well as a start state s_0. This is the assumption that most

[2] In fact, approximate filtering is also an large research area in control and Artificial Intelligence as isz clear from the proceedings of the International Joint Conference on AI or Uncertainty in AI.

[3] I am particularly grateful to John Langford on the topic of access models. Our conversations and work on benchmarks for these models solidified my thoughts on these different oracles.

belief state POMDP planners make in the literature. We note that this is a large amount of information that may be both very difficult to enumerate, specify, or learn from an environment.

2.1.2. Deterministic Generative Model

In this case we are given a simulator that can be started at an arbitrary state. Further we are able to specify the "seed" of the random generator. This is the model described in Pegasus [Ng and Jordan, 2000] and elaborated on there.

2.1.3. Generative Model

A generative model is a simulator, as above, that can be set arbitrarily to an initial state, but for which we cannot control the random bits in any way. This is perhaps the most common intuitive notion of simulative model.

2.1.4. Reset Model

In the reset model, we are given a simulator that we can run trajectories through and can reset to an initial state at any time. It cannot, however, be placed in an arbitrary state as the generative models can.

2.1.5. Trace Model

This is the strong-sense reinforcement learning model. It captures the notion of life in the real-world as we must live it. It is a simulation that we encounter through one continuous, un-resettable trajectory.

It is interesting to note that any of these models can be used to easily construct a simulation of the models below it in the list. It is not, in general, easy to go from the lower models to the upper models.

The phrase "reinforcement learning" is often broadly used to refer to any system that interacts with the model through one of the simulation type interfaces we described above. The last model captures the strong-sense of reinforcement learning where we are constantly in direct interaction with the world.

CHAPTER 3

Policy Gradient Techniques

BELLMAN equation methods have traditionally been the preferred technique for solving large MDPs [Putterman, 1994]. The application of Bellman methods, however, becomes problematic in high dimensional state-spaces, with unknown system dynamics, and when controllers must use partially observed state. There appears to be a fundamental "information mismatch" [Langford, 2004] between the assumptions of Bellman equation methods and the simulation-based interfaces to large scale models that we are interested in solving.

In particular, Bellman methods suffer from an important flaw that manifests itself both empirically and theoretically. There are many facets to view this flaw from. One is that the Bellman equations do not hold over the space of observations. A particularly natural one in the context of learning is to note that small approximation error in any sense natural to learning (expected ℓ_1 norm, for instance) does *not* translate into small execution error or good performance.

In light of this important pitfall of the Bellman equation techniques under uncertainty and approximation, we will devote little time in this work to their study. The exception will be in chapter (9), where we develop a Bellman-Isaacs equation to solve the robust control problem.

Instead, we take a direct policy search approach to learning. In the early chapters we pursue a policy gradient approach to learning. We do so by defining a class of stochastic policies, parameterized by $\theta \in \mathbb{R}^n$ and compute a simulation-based approximation to the gradient of the performance.

There are an infinite variety of ways to define an appropriate class of stochastic policies. A few broad approaches are rather common in the literature however. One is the use of the exponential family distributions on controls conditioned on the observations. [Bertsekas and Tsitsiklis, 1996] Another very common family is that of neural networks, which can be naturally seen as a kind of chaining of exponential family models. These are also discussed in detail in [Bertsekas and Tsitsiklis, 1996].

3.1. Episodic Reinforce

In this section, we present the policy gradient algorithm known as Reinforce and then briefly describe its operation.

Algorithm 1: REINFORCE

Data: A path ξ drawn from the policy π in simulation of a POMDP
Discount Factor $\gamma \in [0, 1]$

An *eligibility trace* $e_0 = 0$ (vector of length n)

A update vector $\Delta = 0$ (vector of length n)

foreach $t \in \{0...T-1\}$ **do**

$e_{t+1} = \gamma e_t + \frac{\nabla \pi(a_t|o_t)}{\pi(a_t|o_t}$

$\Delta_{t+1} = \Delta_t + \frac{1}{t+1}(r_t e_t - \Delta_t)$

Note that the algorithm described above makes very weak demands on the system. It naturally handles the reset model[1] and partial observability. As stated it requires that we are able to calculate $\frac{\nabla \pi(a_t|o_t)}{\pi(a_t|o_t}$. This is usually straightforward for the kind of policies we are interested in considering. For instance, this is a straightforward computation of expected values in the exponential family of policies, and backpropagation [Rumelhart *et al.*, 1986] allows us to efficiently compute it in a neural network representation.

We sketch now the argument for the correctness of REINFORCE. More details can be found in [Williams, 1992].

[1] GPOMDP provides a single execution trace analysis of essentially the same algorithm.

3.2. Understanding the algorithm

The approach to computing the policy gradient taken by REINFORCE has a long history[Glynn, 1986, Williams, 1992, Baxter *et al.*, 1999a]. The basic approach is known as the likelihood ratio method. We are optimizing the *performance*:

$$J_\theta = \mathrm{E}_{\xi\sim \mathrm{P}_\theta}[\mathrm{R}(\xi)]$$

where ξ is a trajectory drawn from distribution P which is implicitly parameterized through the policy by θ. Assuming, as we do throughout this work, that the rewards are control independent[2] then we may both multiply and divide by $P_\theta(\xi)$ to get the derivative of the performance as follows:

$$\nabla J = \mathrm{E}_{\xi\sim \mathrm{P}_\theta}[\mathrm{R}(\xi)\frac{\nabla \mathrm{P}_\theta(\xi)}{\mathrm{P}_\theta(\xi)}]$$

Now a simulator allows us to draw a sample ξ according to P_θ by simply using the policy defined by the parameters. In this case

$$\widetilde{\nabla J} = R(\xi)\frac{\nabla P_\theta(\xi)}{P_\theta(\xi)}$$

or any average of multiple trajectories is an unbiased estimate of the performance gradient. It converges almost surely as the number of trajectories we average gets large.

Now in the case we are studying of a POMDP, the quantity $\frac{\nabla P_\theta(\xi)}{P_\theta(\xi)}$, known as the likelihood ratio, can be easily computed. In fact, it follows immediately from the chain rule that

$$\frac{\nabla P_\theta(\xi)}{P_\theta(\xi)} = \sum_{t=0}^{T-1}\frac{\nabla \pi_\theta(a_t|o_t)}{\pi_\theta(a_t|o_t)}$$

once we observe that all probabilities not involving the control policy directly simply cancel.

The further assumption, which is a standard part of the POMDP definition, that reward is additive in time has also been made in algorithm (1). This is useful because it leads to a recursive implementation as well as reduced variance. It is not, however, necessary for the basic approach. We can always simply compute:

$$\sum_{t=0}^{T-1}\frac{\nabla \pi_\theta(a_t|o_t)}{\pi_\theta(a_t|o_t)}$$

and multiply it by an arbitrary, path-dependent reward. The insight that leads to the recursive implementation is simply that rewards at a time step are not influenced by decisions

[2]Control dependent or parameter dependent rewards are a trivial addition that complicates presentation. [Baxter *et al.*, 1999a] provides details on implementation when this is the casee.

in the future relative to that time step. Showing this mathematically is straightforward, but rather messy. We first write the integral over paths as an iterated integral at each time step:

$$\int dP(\xi)\frac{\nabla\pi_\theta(\xi)}{\pi_\theta(\xi)} =$$
$$\int_{s_0,o_0,a_0}\pi_\theta(a_0|o_0)p(s_0)p(o_0|s_0)\frac{\nabla\pi_\theta(a_0|o_0)}{\pi_\theta(a_0|o_0)}\int_{s_1,o_1,a_1}\ldots[r(s_0)+r(s_1)+\ldots]$$

We can pull each of the rewards as far to the left as the rightmost integral that contains that state. Noting then that

$$\mathrm{E_{p(y|x)}}[\nabla \mathrm{p(y|x)}] = 0$$

(which follows immediately as probabilities sum to the constant 1) it follows that

$$\mathrm{E_{P_\theta(\xi)}}[\frac{\nabla\pi_\theta(\mathrm{a_t}|\mathrm{o_t})}{\pi_\theta(\mathrm{a_t}|\mathrm{o_t})}\mathrm{r_s}] = 0$$

for all $s \leq t$. Hence we can compute an unbiased update as

$$\Delta = \sum_{t=0}^{T-1}\frac{\nabla\pi_\theta(a_t|o_t)}{\pi_\theta(a_t|o_t)}\sum_{s=t+1}^{T} r_s$$

where $r_{T+1} := 0$. It is then easy to see that algorithm (1) is simply a recursive computation of Δ.

3.3. Policy Gradient Theorem

We can write what we have learned about REINFORCE another way. Note that:

$$\nabla J = \frac{1}{T}\sum_t \int dP(\xi)\frac{\nabla\pi(a_t|o_t)}{\pi(a_t|o_t)}[r_t + r_{t+1} + \ldots] \tag{3.1}$$

by linearity of expectation and the above results. Inspection of the graphical model describing the POMDP (2.0.2) makes it clear that the random variable that is the sum of future rewards is conditionally independent of the random variable that is the likelihood ratio $\frac{\nabla\pi_\theta(a_t|o_t)r_s}{\pi_\theta(a_t|o_t)}$ given the current state and action. This is a consequence of d-separation.[3] Since

$$\mathrm{E_{p(z)}}[\mathrm{E[XY|Z]}] = \mathrm{E_{p(z)}}[\mathrm{E[X|Z]E[Y|Z]}]$$

if X and Y are conditionally independent, we can conclude that the expression for equation (3.1) simplifies. In particular, if we use $Q_t^\pi(s_t, a_t)$ to denote the expected future (and present) rewards from a state given an action (a quantity that we will call the *action-value function*) we see that we can write it as:

[3] Clearly this can be demonstrated algebraically as well, although it is easier to read it off the graph.

$$\sum_t \frac{1}{T} \int_{s_t,a_t,o_t} dsdadoP_t(s_t)p(o_t|s_t)\pi(a_t|o_t)\frac{\nabla\pi(a_t|o_t)}{\pi(a_t|o_t)}Q_t^\pi(s_t,a_t) \tag{3.2}$$

We use P_t to represent the marginal probability of the state space at time t given we are using policy π. What we have just shown is implicit in the REINFORCE algorithm. It was first given in this explicit form in [Sutton *et al.*, 1999a]. We can phrase the result neatly in terms of an inner product. Define $d^\pi(s)$ as the probability distribution that is a uniform mixture from each time t of the probability distribution over states given the policy π. (In the time discounted case, the average is taken with weights proportional to γ^t.) Consider the *policy inner product* defined as:

$$\langle \phi, \psi \rangle_\pi = \int_s dsd^\pi(s) \int_a da\pi(a|s)\phi(s,a)\psi(s,a)$$

As we discuss in more detail in later chapters, $\langle \cdot, \cdot \rangle_\pi$ defines a metric, on vectors $\psi(s,a)$, that depends on our parameters θ. We call our final result of this section the policy gradient theorem:

THEOREM 3.3.1 (Policy Gradient Theorem).

$$\nabla J = \langle Q^\pi, \nabla \log \pi \rangle_\pi \tag{3.3}$$

The gradient is thus simply the projection of the action-value function onto basis functions defined by the policy.

3.4. Summary

The REINFORCE algorithm has many variant that are not covered in this work. [Baird and Moore, 1998] provide an algorithm that combines REINFORCE with a gradient descent procedure for various Bellman error metrics. [Sutton *et al.*, 1999a] and [Konda and Tsitsiklis, 2002] consider methods that compute a value-function estimate to learn the gradient. The former is equivalent to REINFORCE ; the latter differs in that it is a multi-time-scale approach. (i.e., the value-function and policy are updated at different time-scales.) Other versions show how reinforce can be used or modified to handle distributed learning, internal state, [Meuleau *et al.*, 2001] and re-using trajectories learned from other policies[Shelton, 2001].

In the next three chapters we will consider novel variants on the policy gradient scheme. REINFORCE serves as the jumping off point to the more sophisticated variants.

CHAPTER 4

Non-parametric Policy Gradients

THIS chapter shows how techniques from the *kernel machine* [Kivinen *et al.*, 2002] approach to learning can be extended to the policy search problem. The machinery of Reproducing Kernel Hilbert Spaces (RKHS) allows us to develop a natural generalization of policy search to search through a space of functions. From this we recover a fully non-parametric policy search. Further, we investigate the regularization of policy learning, a notion concomitant to learning in function space, but one that has seen little investigation in the reinforcement learning literature.

We briefly review the essentials of RKHS theory. We then derive kernelized versions of William's REINFORCE [Baxter *et al.*, 1999a] and closely related algorithms for policy search. We consider both online and batch approaches. We then consider techniques that render the approach computationally tractable. Finally, we consider the application of the techniques to two domains: the well-known dynamics problem *Mountain Car*, and a financial portfolio optimization considered in [Ormoneit and Glynn, 2001].

4.1. Reproducing Kernel Hilbert Spaces

Kernel methods, in their various incarnations (e.g. Gaussian Processes, Support Vector machines) have recently become a preferred approach to non-parametric machine learning and statistics. They enjoy this status because kernel methods form a kind of point-free linear-algebra, where inner product operations are performed by the kernel. As such, natural operations, like least-squares approximation in function space, can be described by simple algebraic methods that naturally generalize parametric linear counterparts.

By kernel here we mean a symmetric function $k(\cdot, \cdot)$ of two arguments that produces a real scalar. Further, we must assume that the kernel is positive-definite in the sense that when applied to any two lists of inputs $\{a\}_i$ and $\{b\}_j$ the matrix $K_{i,j} = k(a_i, b_j)$ is a positive definite one. We use positive definite in the sense commonly used with the learning and kernel communities. One definition is that all the eigenvalues of the matrix are greater than or equal to zero. This is more commonly called positive semi-definite among mathematicians and engineers.

Intuitively, a kernel forms a kind of "similarity" between two objects. It can also be thought of as specifying a covariance (as the kernel matrix is positive-definite) between various objects in the domain of the kernel.

EXAMPLE 4.1.1 (Gaussian Kernel). *A canonical example of a kernel is the Gaussian kernel over real vectors:*

$$k_{Gaussian}(x, y) = \exp \frac{-|x - y|^2}{2\sigma^2}$$

where σ^2 represents the length scale of the kernel. Gaussians come up as solutions to diffusions and have a number of useful properties.

EXAMPLE 4.1.2 (Linear Kernel). *Another very common kernel is the linear kernel:*

$$k_{Linear}(x, y) = \langle x, y \rangle$$

Our examples of linear kernels are given over R^n, but this is not a restriction on kernels, just a reflection of perhaps their most common usage. Much research on kernels is devoted to constructing natural and efficiently computable kernels on trees, graphs, finite-state machines, and various other spaces.

4.1.1. The RKHS

Kernel methods can be thought of as implementing a generalization of the notion of similarity defined by the standard inner-product, where the kernel computes the similarity between its inputs. Kernels as described can also be thought of as computing the standard inner product in a linear space formed by the kernel functions. For any object a in the domain of the kernel, the function $k(a, \cdot)$ can be thought of as a vector in a linear space.

We wish more than a linear space of functions, however. We would like the space to be a *Hilbert space*. Formally, a Hilbert space is a *complete*, linear inner product space. In

essence, it is a generalization of the finite dimensional Euclidean spaces we typically deal with in linear algebra. It is often useful to think of a Hilbert space as being like R^n, for possibly a very large (infinite) n. Completeness is a technical requirement that essentially means that we include limits in the space. Analysis would be impossible if a sequence of vectors getting progressively closer converged to a vector that lay outside the space under consideration.

We turn this linear space into a Hilbert space as follows. First we equip it with an inner product: for two vectors in the linear space, $f = \sum_i \alpha_i k(a_i, \cdot)$ and $g = \sum_i \beta_i k(b_i, \cdot)$ the inner product $\langle f, g \rangle_{k(\cdot,\cdot)}$ is $\sum_{i,j} \alpha_i \beta_j k(a_i, b_j)$. That this is a proper inner product may be easily checked. Finally, technical issues from analysis require us to complete the space by adding the limits of all Cauchy sequences. We denote the space $H_{k(\cdot,\cdot)}$.

EXAMPLE 4.1.3 (Gaussian Kernel Norm). *The Gaussian kernel function $k_{Gaussian}(x, \cdot)$ has norm 1. This follows immediately from $k_{Gaussian}(x, x) = 1$ for all x.*

4.1.2. The Reproducing Property

We note a very useful property of the Hilbert space we have constructed is the *reproducing* property. The idea is that kernel k is the representer for evaluating a function f in the Hilbert space:

PROPOSITION 4.1.1 (Reproducing Property). *For any $f \in H_{k(\cdot,\cdot)}$, the kernel reproduces the function: $\langle f, k(x, \cdot) \rangle_{k(\cdot,\cdot)} = f(x)$.*

If we want to know the value of a function at some point, we can consider this a projection of the function onto the kernel. This result follows from straightforward calculation of the definition of inner product we have given.

The reproducing property can also be used to directly define these Hilbert spaces (as opposed to starting with positive definiteness). In fact, given any Hilbert space of functions, it is a result of functional analysis that if the evaluation functionals (i.e. higher order functions that take f in H and return $f(x)$ for some x) are continuous, then there *must* exist a reproducing kernel that creates that space. RKHS are very general objects indeed.

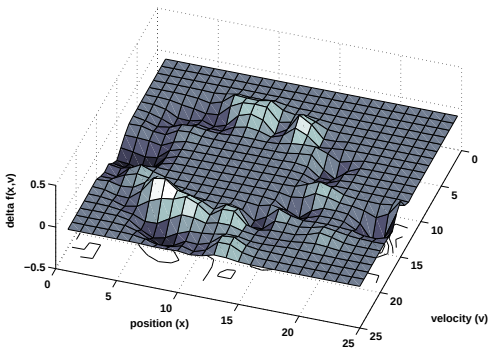


Figure 4.1. The figure shows a stochastic gradient update calculated from a single sample path in the *Mountain-Car* problem. Notice the gradient is a function on the state-space composed of kernels (here Gaussians) at points along the sample trajectory.

4.1.3. Inner products

In this chapter we need to consider a variety of inner products and their associated distance metrics. We denote the kernel inner product as above and its associated norm as $\|\cdot\|_{k(\cdot,\cdot)}$.

For POMDPs we will denote the *policy inner product* as:

$$\langle \phi, \psi \rangle_\pi = \int_s ds\, d^\pi(s) \int_a da\, \pi(a|s)\phi(s,a)\psi(s,a)$$

The same inner product can be considered for functions of observation and action by marginalizing out the state using the observation probabilities. The induced norm is written $\|\cdot\|_\pi$.

4.2. Search in RKHS

We will consider stochastic policies $\pi_f(a|o)$ that are defined by a function $f \in H_{k(\cdot,\cdot)}$ where the kernel $k(\cdot,\cdot)$ is defined on the observation space, or the cross-product observation-action space. That is, our policies will take an observation o and compute a probability distribution over controls.

4.2.1. Stochastic Gradient Methods

One general policy-search approach in reinforcement learning is to perform stochastic gradient ascent on J using sampled paths. The REINFORCE class of algorithms, as discussed in the last chapter, use the likelihood ratio method to approximate the gradient

using the identity:

$$\nabla J = \int_{\Xi} d\xi \; \nabla p(\xi) \; r(\xi) = \int_{\Xi} d\xi \; p(\xi) \frac{\nabla p(\xi)}{p(\xi)} r(\xi)$$

Here we must consider stochastic gradient descent in the function space $H_{k(\cdot,\cdot)}$. Functional gradient methods have been considered for classification and regression in [Mason *et al.*, 1999] and [Kivinen *et al.*, 2002]. In reinforcement learning the distribution of samples, $p(\xi)$, changes with the choice of function defining the policy, so we must consider the derivative of this distribution, not simply the derivative of the empirical cost function as in [Kivinen *et al.*, 2002]. This reflects a fundamental difference in the formulation of reinforcement learning as contrasted with classification or regression.

4.3. The Functional Gradient

To generalize a stochastic gradient algorithm like REINFORCE , we must formally define the notion of a gradient in the space of functions. The natural conception of gradient in $\mathbb{R}^n$ comes about by thinking about the linear approximation to a function at some point. Here we must think about linear approximations to *functionals*.

A scalar valued function F of a function f is known as a functional, and we write it as $F[f]$. The natural generalization of the gradient is to define $\nabla_f F[f]$ on the Hilbert space $H_{k(\cdot,\cdot)}$ as the linear deviation of the functional for a perturbation $\epsilon \dot{g}$, with $g \in H_{k(\cdot,\cdot)}$:

$$F[f + \epsilon g] = F[f] + \epsilon \langle \nabla_f F, g \rangle + O(\epsilon^2) \tag{4.1}$$

The existence of such an object, $\nabla_f F[f]$ for sufficiently "smooth" F is an important result in functional analysis.

Let us consider an important special case of a functional and compute its derivative. A particularly simple function is $F[h] = h(x)$, that is the functional that simply evaluates its argument at some particular point x. Now

$$\begin{aligned} F[h + \epsilon g] &= h(x) + \epsilon g(x) && (4.2) \\ &= F[h] + \epsilon g(x) && (4.3) \\ &= F[h] + \epsilon \langle k(x, \cdot), g \rangle && (4.4) \end{aligned}$$

Note that the last equality follows from the fundamental reproducing property described in equation (4.1.1). This implies that we can write $\nabla_f h(x_i) = k(x_i, \cdot)$. That is, the

functional gradient of a function in the Hilbert space applied at a point is simply the kernel itself at that point. [1]

To perform stochastic gradient descent, we "roll-out" a trajectory using the current policy (defined by f) and then compute:

$$(\nabla_f J_f)_{emp} = \frac{\nabla_f p_f(\xi_i)}{p_f(\xi_i)} r(\xi_i)$$

Gradient ascent takes a step in this direction. Fortunately, it proves easy to compute $\frac{\nabla_f p_f(\xi_i)}{p_f(\xi_i)}$. In particular, the derivative depends only on the controller probabilities and can be written as:

$$\sum_t \frac{\nabla_f \pi_f(a_t|o_t)}{\pi_f(a_t|o_t)}$$

This is simply a consequence of the equivalent of the product-rule for functional derivatives. All terms, for instance $p(s_{t+1}|s_t, a_t)$, that are independent of the parameterizing function cancel out.

It is important to contrast the approach described with standard policy gradient algorithms. The policy gradient theorem tells us that the gradient in a POMDP can be written as $\nabla J = \langle Q_\pi, \psi_i \rangle_\pi$ where $\psi = \frac{\partial \log \pi(a|y)}{\partial \theta_i}$. The functional generalization is $\nabla_f J = \langle Q_\pi, \nabla_f \log \pi(a|y) \rangle$. This agrees with the standard gradient when we take the inner product that defines ∇_f as the Euclidean inner product on policy coefficients. Here instead we take the metric to be given by the RKHS norm $\langle f, f \rangle_{k(\cdot,\cdot)}$ on functions, and this is how we extend to the non-parametric setting.

Let us consider the simplest example of a functional policy where π is a distribution over two actions (u in {-1,1}), f is a one dimensional function and we compute $\pi(a = 1|y) = 1/(1 + e^{f(y)})$ for our stochastic policy. These policies provides simple to visualize examples (and is the form of policy we use in our experimental section). Further, the sigmoidal policy easily generalizes to non-binary actions as $\pi(a|y) = e^{f_u(y)}/(\sum_i e^{f_i(y)})$ where we provide a different function f_i for each action.

We need one more result from functional analysis to complete the computation of the gradient. It is the functional equivalent of the chain rule:

$$\nabla_f g(F[h]) = \partial_x g(s)|_{F(h)} \nabla_f F|_h$$

[1] We note also that there are no higher order terms in equation (4.4). This is as we would expect as F is a linear functional.

Given this, and the preceding notes about the reproducing property, one can see the important result

$$\frac{\nabla_f \pi_f(a_t|o_t)}{\pi_f(a_t|o_t)} = -sign(a_t)(1 - \pi_f(a_t|o_t))k(o_t, \cdot) \tag{4.5}$$

That is, for each step along the trajectory we get a kernel basis center on that point with a coefficient related to the action chosen, and the appropriate probability of that action. To increase the reinforcement functional, we simply add to the current f the product of a learning rate and the computed stochastic gradient.

If we consider an exponential family of policies $\pi(a|s) = e^{f(o,a)}/Z$ defined by RKHS on the joint observation-action space $H_{k(\langle o,a\rangle,\cdot)}$), we find

$$\frac{\nabla_f \pi_f(a_t|o_t)}{\pi_f(a_t|o_t)} = (f(o_t, a_t) - E_{\pi(a'|o_t)}[f(o_t, a')])k(\langle o_t, a_t\rangle, \cdot) \tag{4.6}$$

Adding a regularization term that penalizes functions by their norm $\langle f, f\rangle_{k(\cdot,\cdot)}$ amounts to, in the stochastic gradient framework, shrinking the coefficients of all previous kernels by multiplying them by some constant $c \in (0, 1)$. We can see this by noting that adding $-\frac{\lambda}{2}\langle f, f\rangle$ to our reward function and then taking the functional gradient gives us back $-\lambda f$. The update is then

$$f_{t+1} := f_t + \alpha\left(-\lambda f_t + r(\xi)\frac{\nabla_f \pi_f(a_t|o_t)}{\pi_f(a_t|o_t)}\right) \tag{4.7}$$

where $r(\xi)$ is the total reinforcement for the current trajectory.

4.3.1. Metric Computation

It is interesting to observe that the functional gradient differs from that which one would compute [Kivinen *et al.*, 2002] by simply adding kernels at each point along the trajectory and computing partial derivatives with respect to the coefficients on these kernels.[2] Such an algorithm has no obvious convergence properties as it is continuously adding basis functions. Our algorithm, in contrast, has a clear interpretation as searching in a compact[3] space of functions.

[2] A quick computation shows that they differ by multiplication by the matrix $K_{ij} = k(o_i, o_j)$.

[3] Tikhonov regularization as we are using here amounts to limiting the consideration to a ball in function space[Schoelkopf *et al.*, 2000].

Further, this naive algorithm of adding kernels and then computing partial derivatives ignores the interaction the different kernel basis functions have with each other, which the function space approach takes explicitly into account. It can also be easily checked that this naive algorithm of simply adding kernels and computing the standard gradient has an $O(n^2)$ scaling as well, as it involves a matrix multiplication. This is because the kernel defines a different Riemannian metric than the Euclidean one on the space of coefficients. This metric can be considered a kind of prior on policies– implying policies should be similar where the kernel function is large. The regularization term enforces just this similarity.

The notion of metric begs the question of whether there is, in some sense, a canonical metric. Recent work [Bagnell and Schneider, 2003] [Kakade, 2002] considers what such a metric should be, concluding that the natural metric should be related to the distribution of paths induced by the controller π. Section (5.5.2) develops these considerations in the kernel framework.

4.3.2. On-line implementation

For time additive cost functions one can make another addition to make the procedure fully online. This is the observation that controls at a given time do not influence the past of that trajectory. It lowers the variance of the estimate to include only the future reward of the trajectory. Further, by allowing the policy to shift during the trajectory we can make our algorithm a fully online one like REINFORCE. We present the non-parametric version of REINFORCE [4] below.

[4] Since we give an average reward infinite-trajectory length algorithm, this is technically GPOMDP instead. REINFORCE would involve restarts.

Algorithm 2: Online Kernel Policy Search

Data: t=0
empty list of kernels k
empty list of eligibility coefficients e
empty list of gradient coefficients Δ

Result: Final kernel list and matching coefficient list Δ

repeat

Choose an action according to $\pi_f(a|o_{t+1})$

Observe reward r_t

Shrink each eligibility constant by γ– (a discount factor or bias/variance tradeoff parameter)

Add a new kernel $k(o_t, \cdot)$ to the list based on the observation seen at time t

Add the corresponding coefficient of $k(o_t, \cdot)$ from $\frac{\nabla_f \pi_f(a_t|o_t)}{\pi_f(a_t|o_t)}$ computed as in Equation (1) or (2) to the eligibility list as e_i

Set all Δ_i coefficients to $\Delta_i(1 - 1/(t+1)) + r_t e_i/(t+1)$

until *done*

return *Final kernel list and matching coefficient list* Δ

4.4. Efficient Variations

In practice it may be valuable to get more accurate gradient estimates before changing our policy. This may potentially be more stable, allow us to apply more sophisticated step length heuristics, and will lower the computational burden. To do so, we consider batch processing a functional gradient computed from a reasonably large number of sample trajectories.

4.4.1. Efficient computation

The central computational difficulty of the approach we outline in this work is the evaluation of the functional gradient. We are forced to do work linear in the number of observations seen when we evaluate $f(o_t, a_t)$ in the policy. In general, we would like to

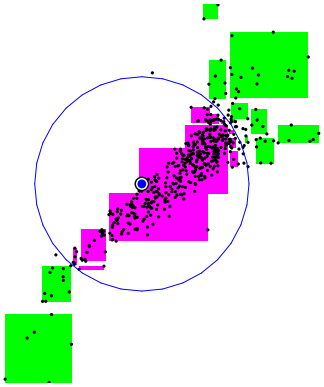


Figure 4.2. This image depicts a KD-TREE being used to compute the gradient evaluation at a point. (center) Parts of the computation get cutoff (outside the range depicted in the circle) as it can be shown they must have negligible effect on the query.

keep as faithful a recreation of the kernel gradient as possible, but doing so will generally make the computation more expensive.

We may make the gradient evaluations tractable, even when potentially millions of kernel bases spangle our observation space, by noting that the evaluation can be viewed as a kind of weighted n-body problem[Gray and Moore, 2001]. As such, data-structure and algorithm tricks from physics and computational geometry have the potential to render the approach tractable. For our research, we used the classical KD-TREE[Moore, 1990]. In effect, we create a tree that divides the samples in the observation space into regions.

Whenever called upon to make an evaluation, we descend into the tree adding samples from regions that are near our evaluation point. Throughout the tree, we cache the weight of all the kernels that fall underneath it. In such a way, we can compute whether at any given query a node in the tree (and all the observations it contains) can have an effect on our query. We compute the maximum and minimum impact, and if the difference between the two is small enough that it will have negligible influence on our prediction, we simply use the average value. Whenever this cannot be ruled out, we recurse further into the tree.

Geometric data-structures have seen wide-spread use in certain areas of machine learning–like density estimation and locally-weighted learning[Moore, 1990]. We believe that this approach has great potential in alleviating the computational burden of kernel methods in more general contexts than the reinforcement learning application we discuss here. For example, kernel logistic regression and kernel least squares classification both may be solved as special cases of the algorithm we present here and may utilize the same data structures.

The unique aspect of our work that makes possible the efficient use of the KD-TREE is the use of the functional gradient, where the bulk of the computational cost during learning comes in function evaluation. In contrast, it would be much more difficult to attempt this if we were also forced to perform matrix-multiplications or inversions as typical for other training algorithms.

4.4.2. Sparse Representation

Eventually the leverage from data-structures may not be enough to keep gradient computation sufficiently efficient, especially as it grows with each gradient step. To make matters more tractable, we consider approximations that compress the functional gradient to a more parsimonious representation. Such an approach is composed of two pieces. The first chooses a basis set, and the second computes the closest approximation given this set. (Algorithms may interleave these steps as well.) Although surely an area of interesting research, we have found that quite trivial strategies may suffice for basis selection. The simplest of all, random subset, seems to be adequate for at least some problems. Given the basis, we simply compute the least squares approximation to the gradient g in the RKHS:

$$\min_{\tilde{g}} ||\tilde{g} - g||_{H_k(\cdot,\cdot)}$$

where $\tilde{g}$ lies in the span of our reduced basis. The RKHS approach makes such computations very simple.

In coefficient form, the functions are $\tilde{g} = \sum_i \alpha_i k(\tilde{y}_i, \cdot)$ with the true gradient lying in the span of a different set of kernels $g = \sum_j \beta_j k(o_j, \cdot)$. This is not necessary for the general presentation, but suffices for our use. Then

$$\langle \tilde{g} - g, \tilde{g} - g \rangle_{k(\cdot,\cdot)} = \|\tilde{g}\|_{k(\cdot,\cdot)} - 2\langle \tilde{g}, g \rangle_{k(\cdot,\cdot)} + \|g\|_{k(\cdot,\cdot)}$$

The final term is irrelevant to maximize with respect to $\tilde{g}$ so we have in terms of the kernel representation:

$$\langle \sum_i \alpha_i k(\tilde{y}_i, \cdot), \sum_i \alpha_i k(\tilde{y}_i, \cdot) \rangle_{k(\cdot,\cdot)} - 2\langle \sum_i \alpha_i k(\tilde{y}_i, \cdot), \sum_j \beta_j k(o_j, \cdot) \rangle_{k(\cdot,\cdot)}$$

To maximize this with respect to the α parameters, we compute the gradient and set it to zero. Using this, and the definition of the kernel inner product, we get in vector notation:

$$\alpha^T K^{\alpha,\alpha} - \beta^T K^{\beta,\alpha} = 0$$

where $K^{\alpha,\alpha}$ is the kernel matrix for the $\tilde{y}_i$ and $K^{\beta,\alpha}$ the kernel matrices for the $\tilde{y}_i$ with the o_j. (i.e. $K^{\beta,\alpha}_{j,i} = k(o_j, o_i)$). By inspection, the second term is simply the function g applied at the points in our approximation. Thus the final answer is simply:$\alpha_i = (K^{\alpha,\alpha})^{-1} g(o_i)$

The matrix inverse is only the size of our reduced basis, so this may be computationally tractable. It is important to note that these sparse projections differ dramatically from choosing basis functions apriori and running parametric REINFORCE. Besides the more appropriate metric, we sparsify multiple past gradient steps each derived from multiple trajectories using bases selected from any of these steps. If our policy doesn't spend any time in some area of state-space any longer, we no longer need waste that representational power. In this way, our basis are picked depending on the steps we have taken so far. In all cases we can sensibly bound our approximation error in terms of Hilbert Space norm, so that we know the loss with respect to an optimal representation.

4.4.3. Sample and Computational Complexity

The kernel policy gradient algorithm presented here inherits the unfortunate sample complexity results of REINFORCE[Langford and Kakade, 2002]. It can be shown that computing the magnitude of the gradient requires a reasonable number of samples, but the direction scales poorly. Finally, the algorithms here have the potential to suffer from serious local minima problems. Interestingly, perhaps because of the degree of representational power, we never in any experiments found it converging to any other local minima.

4.5. Case Studies

We provide here two experiments that we hope can provide further insight into the methods we have developed.

4.5.1. Mountain-Car

First, we applied our method to the well-understood *Mountain Car* domain.[Moore, 1990] This is a simple dynamics problem, where the goal is to get an under-powered car up an incline. The problem is interesting in that it requires "non-minimum phase" behavior– that is, we must retreat from the goal before we can reach it. We use the standard dynamic equations, except that to make the problem somewhat more difficult; we count going out

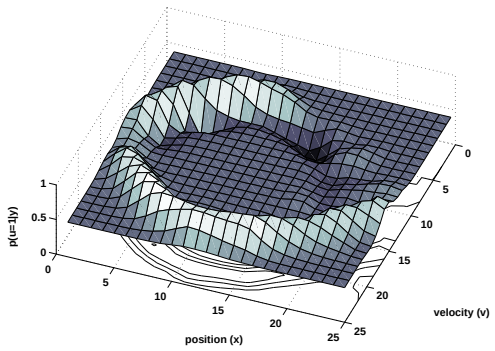


Figure 4.3. Here we depict a learned policy for the mountain car problem. High points represent high probability of applying forward control and low points represent high probability of applying backward control. Note how our kernel method concentrates representational power where it is required. The regions where the function is flat correspond to regions where f is regularized to 0 as the car spends no time there.

the back of the state-space as a penalty of -1. A time-discounted reward of $+1$ is applied for making it to the top of the hill.

Mountain car is a valuable domain because much intuition can be gained by visualizing solutions. In Figure (4.1) we show a stochastic gradient computed during the stochastic gradient ascent algorithm. It is noticeably jagged and only partially agrees with the true gradient at that point in policy space.

The learned controller for the *Mountain car* problem achieves near-optimal performance for the standard starting point within a small number of gradient steps– typically about 5. In Figure (4.3) we illustrate a policy during the process of learning control. This figure demonstrates the kernel approach concentrating its representational power where the policy actually spends time. In this example, the car starts at about (x=17,v=5). It follows the trench of low probability backwards and then begins going forward, from whence it follows the ridge of high probability of the forward action all the way to the goal.

4.5.2. Portfolio Optimization

Another valuable validation experiment is comparing our work to the motivating application of [Ormoneit and Glynn, 2001]. This paper is perhaps the closest in spirit to our own work, as it develops some theory for non-parametric methods, albeit in the value-function framework. As such, the problem they consider, a financial portfolio strategy problem seemed quite appropriate to consider.

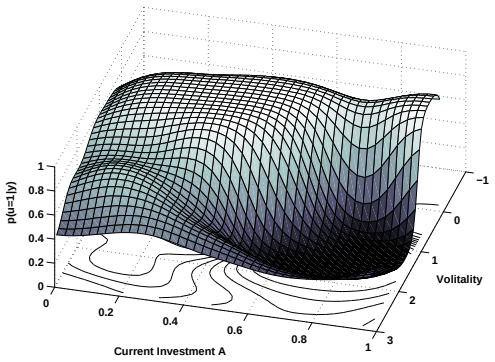


Figure 4.4. Here we depict a policy learned on the portfolio problem. The metric here is the modified Sharpe ratio metric we describe in the text. High regions indict where the trader will increase its market state, and low regions the reverse. The policy has the property that at high volatilities the trader prefers the riskless-asset.

In essence, the problem considers an investor trying to decide what fraction of their assets at each time step to devote to a stock market S_t and to a riskless money-market. As in [Ormoneit and Glynn, 2001], we consider discrete investment levels in the stock $A = \{0.0, 0.1, \ldots 1.0\}$. The market is assumed to move according to an Itô-type stochastic differential equation:

$$\begin{aligned} dS_t &= \pi S_t dt + \sqrt{v_t} S_t dB_t \\ dv_t &= \phi(\hat{v} - v_t) S_t dt + v_t \rho d\tilde{B}_t \end{aligned}$$

where dB_t and $d\tilde{B}_t$ represent independent Brownian motions. The first equation describes the movement in the market price of the stock. This is a standard equation describing a geometric Brownian motion with a varying volatility. The second equation describes the underlying volatility of the stock price, and can be seen to be mean-reverting with noise. The parameters are $\pi = 1.03$, $\hat{v} = 0.3$, $\phi = 10$ and $\rho = 5.0$.

It is assumed that the whole system is mean-adjusted so that the riskless asset provides zero interest. Given this, the reward to the system can be defined as the wealth of the investor at the end of the period relative to their initial wealth. In this case, we consider a risk averse investor with a concave utility function– in this case $\log$. This is convenient for dynamic programming solutions as it makes the reward look additive:

$$\sum_t log(1 + A_t \frac{(S_{t+1} - S_t)}{S_t})$$

for a sufficiently small time discretization.

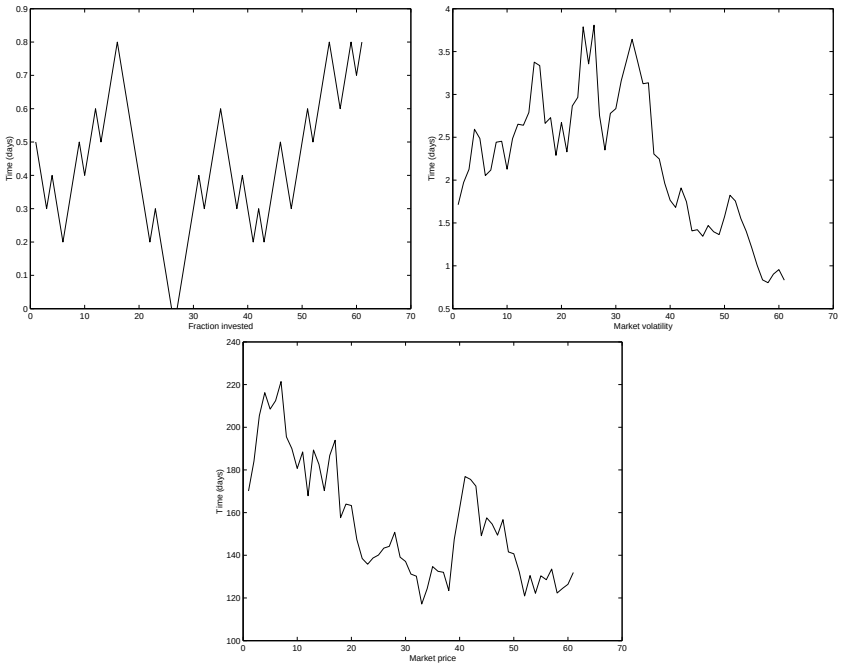


Figure 4.5. The above images depict trajectories encountered while learning. Note that when market volatility is high, the trader lowers the fraction invested in it to reduce its risk.

Unfortunately, this problem as described in [Ormoneit and Glynn, 2001] is not a true sequential decision making problem. Instead, the agent can simply choose their investment level at each time independently to maximize their one step gain, as they have no impact on the market. To make the problem more interesting, we have modified it so that decisions are instead required to be small modifications in the portfolio. This is a very reasonable restriction as it allows us to represent both capital limitations of the investor and the problem of "moving the market" that happens with large position changes in somewhat illiquid markets. Perhaps most importantly, this restores the sequential decision process to the problem, as the agent is forced to reason about how changing its position now will affect it further in time.

It is difficult to make quantitative comparison with [Ormoneit and Glynn, 2001] as we have made the problem significantly more challenging, and because only a fairly small sample set is represented there. Here we show performance over a two-month window, against the popular "buy-and-hold" strategy where one invests completely in the risky asset. With even a small number of gradient steps, our algorithm outperforms this baseline. We present results comparing a modified Sharpe ratio of each strategy. The Sharpe ratio,

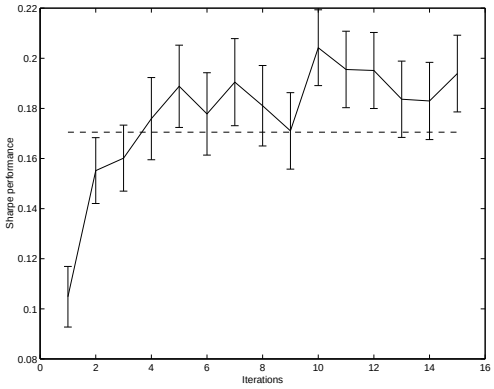


Figure 4.6. This graph illustrates the performance of our algorithm learning the portfolio strategy as a function of iterations. Error bars are negligible on the baseline performance of the best stationary strategy.

a common performance measure in the financial literature, divides our measure of performance by the average volatility (std. dev.) over the time period in consideration. This has the advantage of penalizing "luck" on the part of the investor due to high volatility swings in the risky asset. For time periods with a net loss, it is not a sensible metric to interpret. [5] There are many cost-functions that implement this– our simple version which we label as Sharpe ratio in Figure (4.6) simply uses the mean volatility whenever the investment amounts to a net loss. A useful feature of policy search algorithms is that they are able to directly optimize such path-dependent reinforcement functions.

We also visualize the resulting policies for the portfolio optimization problem. The policies here can depend on the full state-variables. In figure (4.4) we see a slice of the policy with control (A_t) and volatility (v_t) varying. The details of the policy are not in any way obvious, but it does have the intuitively gratifying property that for high volatilities, it prefers the riskless asset (Figure 4.5).

Finally, we note that in both the experiments described above we were unable to get a reasonable policy out of naive "non-parametric Reinforce" one might try by simply adding basis functions at each data point seen.

4.6. Extensions and Related Work

For the sake of clarity, we explicitly kept to "fully kernelized" algorithms; that is, without any parametric part. In reality, of course, this may be foolish– if a good parametric

[5] We do not have any wish to down-weight high-volatility investments that cost us. That is, it is better that our metric down-weight good luck, but not down-weight bad luck.

guess is available it should be added to the function f in defining the probability of actions. This kind of semi-parametric type approach gives us the best of both worlds– combining the domain knowledge advantage that parametric policy search brings with the flexibility of the non-parametric approach.

A number of generalizations are natural for the non-parametric approach we have described here. For instance, algorithms that re-use (importance weighted) old trajectories [Shelton, 2001] can be easily derived by simply changing the cost function to reflect the probability of these paths. If we can bear the computational burden, these trajectories can lower the experience cost of the algorithms. Next, simple modifications allow us to consider reinforcements that depend on both control and next observation, allowing us to guide policy search by simultaneously learning value-function approximations [Baird and Moore, 1998] [Ng *et al.*, 1999].

Very recent work including [Lagoudakis and Parr, 2003] and [Bagnell *et al.*, 2003b] also admit non-parametric policies. The former, however, relies on unstable approximate policy iteration as the outer loop, essentially treating reinforcement learning as classification. The latter inherits many of the attractive properties of other policy search methods, but requires a more powerful "reset" generative model than is standard in reinforcement learning. It is discussed in detail in Chapter 7.

CHAPTER 5

Covariant and Natural Policy Gradients

One interesting aspect of all policy search algorithms that we have previously described is that they are **non-covariant.** By this we mean that a simple re-parameterization of the policy typically leads to a different gradient direction (not that found by applying the Jacobian of the re-parameterization to the gradient). This is a odd result; it is intuitively difficult to justify such a gradient computation as actually indicating the direction of steepest descent. This problem is well recognized in the pattern-recognition and statistics communities and is a very active area of research. [Kakade, 2002] was, to the best of our knowledge, the first to identify this problem in reinforcement learning and to suggest that techniques from information geometry may prove valuable in its solution.

Inspired by the work of [Amari and Nagaoka, 2000], Kakade proposed a "natural gradient" algorithm. In particular, [Kakade, 2002] proposed a scheme for generating a metric on parameter space that has interesting theoretical properties.

Most convincingly, Kakade showed strong empirical evidence for the algorithm's usefulness. In particular, [Kakade, 2002] applies the algorithm to the Tetris problem of [Bertsekas and Tsitsiklis, 1996]. This problem is particularly interesting as value function methods (as for example described in [Bertsekas and Tsitsiklis, 1996]) demonstrate non-monotone performance; first policy iteration improves the policy dramatically, but after a small number of iterations the policy gets much worse. Normal gradient methods, including second-order and conjugate methods also prove very ineffective on this problem; even after a tremendous number of rounds they only mildly increase the performance of the game-player. The method presented in [Kakade, 2002] shows rapid performance improvement

and achieves a significantly better policy than value-function methods (at their peak) in comparable time.

However, despite recognizing an interesting defect in the general approach and intuiting an apparently powerful algorithm, Kakade concludes that his method also fails to be covariant, leaving the problem open. In this chapter, we present what we believe to be an appropriate solution.

In Kakade's work, there is no proper probability manifold, but rather a collection of such (one for each state) based on the policy. As such, Kakade must rely on an ad-hoc method for generating a metric. Here instead we take a proper manifold, the distribution over paths induced by the controller, and compute a metric based on that. In the special case appropriate to the average reward RL formulation, Kakade's scheme is shown to give rise to a bona-fide natural metric, despite the observation in the paper that the learning was non-covariant. We believe this is an artifact– perhaps of step-length. Further, we note that parametric invariance does **not** require this metric– there are numerous covariant metrics. Rather, a stronger *probabilistically natural* invariance demands the metric used. We describe this result to motivate our method. Importantly, the notion of the metric on the path-distribution allows us to provide very simple and natural extensions to Kakade's algorithm that cover the finite horizon, discounted start-state, and partially-observed reinforcement learning problems as well as the average reward one.

Finally, we discuss the relationship between the natural metric and the Policy Gradient Theorem (Chapter 3). We how how a result of Kakade's connecting compatible actor-critic [Sutton *et al.*, 1999b] methods generalizes, showing that the natural gradient is a least squares estimate of the action-value function, in contrast to the standard projection used in REINFORCE . This allows us to extend techniques from the previous chapter to develop kernelized natural policy gradients.

5.0.1. Problem Setup and Notation

Throughout this chapter we'll emphasize the stochastic control problem as consisting of paths ξ (also called system trajectories) in a space Ξ, a distribution over path-space, $p(\xi)$, that is a function of a sequence of (finite) controls $(a_t)_{t\in\{0,\ldots,T\}}$, indexed by time, from a space A. In particular, in this chapter we will consider problems with a *finite* number of paths, and appropriate limits (as $T \to \infty$) of that case.

We attempt to maximize the expected reinforcement $J(\theta) = \sum_{\Xi} p_\theta(\xi) r(\xi)$ with respect to θ. Again, we use a sum here instead of an integral as our derivations here are in terms of a finite number of paths. Under sufficient regularity conditions, the results generalize to an infinite number of paths. The study of these metrics in cases where the regularity conditions fail remains a very open area of research in the field of information geometry and does arise in some practical applications, although none that we consider in this chapter. The reward function on paths is assumed to be additive in time (or in the infinite time case, discounted or averaged), and a function $r(x)$ of state, although the algorithms discussed in this chapter are not necessarily predicated on that assumption.

We use the notation ∂_i throughout this chapter to denote $\frac{\partial}{\partial \theta_i}$ where the parameter θ should be clear from context. For vectors x and y, we use the notation $\langle x, y \rangle_M$ to denote the inner product with respect to metric M.

5.1. Covariance and Riemannian manifolds

5.1.1. Meaning of steepest ascent

For policy gradient methods, we are interested in finding the direction of *steepest ascent* with respect to our current parameters. That is, we wish to maximize the reward function $J(\theta + \Delta\theta)$ subject to $\langle \Delta\theta, \Delta\theta \rangle = \epsilon$, an infinitesimal. If we reparameterize our controller in terms of, e.g. ζ, and express the same effective infinitesimal policy change in terms of these parameters, ΔJ will of course remain the same. However, if we measure lengths using the naive dot product, the same effective change will not have the same size. It is this problem that leads to the odd behavior of "steepest ascent" rules.

5.1.2. Non-covariant learning rule

To demonstrate the problem we first consider a simple two state, two action MDP described in [Kakade02]. (Figure 5.1) In state 0 executing action 0 returns to the same state and gets a reward of 1. Executing action 1 get no reward but transits to state 1. In state 1, action 0 returns to state 1 and gets reward 2. Action 1 transits to state 0 and achieves no reward.

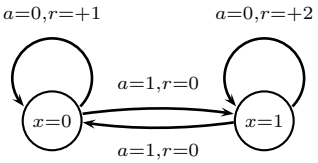


Figure 5.1. A two-state MDP. Each state has two controls one which self transitions (and earns the reward that labels the state) and another that simply transitions to the other state. Rewards occur only on the transitions between different states.

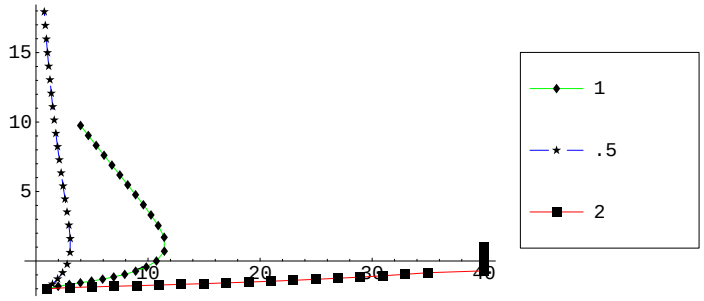


Figure 5.2. Log odds of actions for policies $\log p(a=0|s)/p(a=1|s)$ on the two-state MDP (horizontal axis corresponds to state 0). Different curves correspond to varying λ. Notice the non-covariant policy behavior.

Consider parameterized probabilistic policies of the form $\pi(a=0|s;\theta) = \frac{1}{1+e^{\lambda\theta_0\delta_{s=0}+\theta_1\delta_{s=1}}}$. λ is an arbitrary scale parameter that we use to demonstrate that even mildly different parameterizations lead to dramatically different behavior. In Figure 5.2, we plot the resulting track through the space of policies using the log odds ratio of probabilities of the actions for each of the possible states. We start from a policy that makes it somewhat more likely to choose action 0 in state 0 and action 1 in state 1. We then scale λ to 1, .5, and 2 and plot the log odds of the policy from state 1 (Figure 5.2). Here all the computations are done analytically up to the parameter change step where we use floating point arithmetic. This allows us to concentrate on the problems inherent in the gradient itself, not just our sample based algorithms to approximate it.

The non-covariant behavior is quite clear in this graph. Further, we note that it is *very* difficult to achieve a good policy using this algorithm from this starting point as the *wrong* action becomes overwhelmingly likely to be chosen from state 0. If sampling were used to compute the gradient, we would nearly never get samples from state 1– which we need to improve the policy.

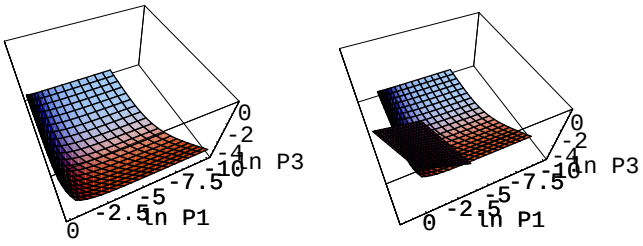


Figure 5.3. An example probability manifold over paths. Each axis represents the log probability of one of the three possible paths. The manifold is formed as we vary the parameters defining it throughout their domain. On the right we attach the tangent space at a particular point in parameter space.

5.1.3. Path distribution manifolds

The control problem is essentially a coupled one of optimization and integration over the space of possible paths, Ξ. This motivates the idea that instead of considering the (arbitrary) distance in terms of parameterization of the policy, we may consider distance in terms of the *change in distribution over paths resulting from the policy change*. We may view the distribution over paths $p(\xi;\theta)$ as a parameterized manifold (nominally embedded in $R^{|\Xi|}$) of dimension $|\theta|$. This takes some work to visualize; as an example consider a space of three possible paths. All possible distributions over these three paths can be smoothly represented with two parameters. In Figure (5.3) (left) we see one visualization with the embedding $\log p(\xi;\theta)$) :

With n parameters, assuming no redundancy, we generate an n dimensional manifold. In the case pictured, it is the set of all distributions on 3 paths, but in general, the dimensionality of the path probability manifold will be tremendously less than the number of possible paths. It is important to understand the manifold under consideration is that of the probability distribution over paths and not of paths themselves.

The study of parameterized manifolds like that pictured above is the domain of differential geometry. Here we are interested in establishing a *Riemannian* structure on the manifold of paths. By that we mean we wish to establish a metric on the tangent space (the local linear approximation to the manifold at a point ξ), so we can measure small parameter changes. We do this with the metric $\mathrm{E}_{\mathrm{P}_{\theta(\xi)}}[\langle \mathrm{X}, \mathrm{Y} \rangle]$ on the tangent space (which is spanned by the partials with respect to each parameter) as $\sum_{i,j} G^{\mathrm{ij}}(\xi) X_i Y_j$, where G is a positive definite matrix. This is a very natural thing to do– instead of just the standard dot product, we have a dot product that allows us to represent rotations and scalings (exactly what a

positive definite matrix can represent) and that can vary smoothly throughout the manifold. In Figure (5.3) (right) we depict the tangent space at a point in our parameterization as the local linear approximation at that point.

5.1.4. Steepest ascent on Riemannian manifold

There are two questions that then naturally arise– what is steepest descent on a function defined over a Riemannian space, and is there a Riemannian metric on the manifold of the paths that is in some sense natural. We quickly answer the first question, and pursue the second in the next two sections. A Lagrange multiplier argument (given schematically here) makes it easy to see the form of the steepest descent direction.

$$\min L(\theta + \delta\theta) = L(\theta) + \nabla L(\theta) \cdot \delta\theta \tag{5.1}$$

$$\text{subject to } \langle \theta, \theta \rangle = \epsilon \tag{5.2}$$

Form the Lagrangian:

$$\mathcal{L}(\theta, \lambda) = L(\theta) + \nabla L(\theta) \cdot \delta\theta + \tag{5.3}$$

$$\lambda(G^{\mathrm{ij}} \delta\theta_i \delta\theta_j - \epsilon) \tag{5.4}$$

and take derivatives with respect to each $\delta\theta$, then set to zero to solve for the optimal direction:

$$\nabla_{\delta\theta} \mathcal{L}(\delta\theta, \lambda) = \nabla L(\theta) + \lambda G^{\mathrm{ij}} \delta\theta_j = 0$$

Since G is positive definite and hence invertible, we get:

$$\delta\theta \propto G^{-1} \nabla L(\theta) \tag{5.5}$$

That is, the direction of steepest descent is simply the normal gradient times the inverse of the metric evaluated at the point of tangency. We call this the "natural gradient" for the Riemannian manifold[Amari and Nagaoka, 2000].

5.2. Invariance and Chentsov's Theorem

Some confusion seems to surround the choice of metric on probability manifolds. There is *no* unique answer for this metric, even under the supposition of parametric invariance as suggested by some authors. The issue is rather more subtle. Any metric that transforms under parameter change according to the Jacobian of the function connecting

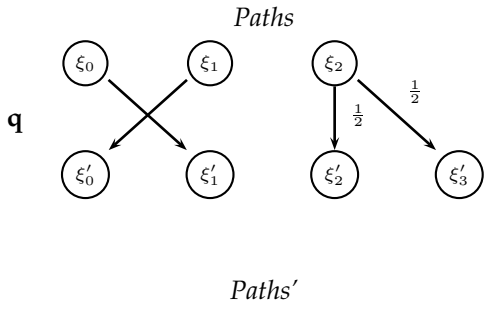


Figure 5.4. A congruent embedding q.

parameters meets this requirement. This type of parametric covariance is a minimum requirement for us. It is natural to suggest a metric that preserves essential probabilistic aspects of the manifold. Consider, for example, functions q (Markov mappings, congruent embeddings, or sufficient statistics, depending on your viewpoint) that carry one distribution over paths to another in a natural and recoverable way. For example, consider the mapping between the two distributions $P(\text{paths})$ and $P(\text{paths}')$ given by congruent embedding q depicted in Figure (5.4).

The mapping q above interchanges the role of two paths and splits one path into two (with probability $1/2$ for each split). In a probabilistically fundamental way, the manifolds $P(\text{path})$ and $P(\text{path}')$ (where we imagine P(path) is a smoothly parameterized set of distributions) are similar to each other. For each path$'$ we can uniquely recover the original probability. In this way, a parameterized distribution over paths can be embedded into a different path space. (This equivalence can actually be phrased in a category theoretic way where the morphisms are these congruent embeddings[Chentsov, 1980].) General congruent embeddings can be thought of as simply generalizations of the example depicted above to allow arbitrary permutations and arbitrary probabilities for different paths stemming from a single path, as well as compositions of these two. In the control problem this could arise by simple permutation of the state variables or change of co-ordinates to ones with redundant information. It is natural to require that with our metric the congruent embedding is an isometry on the tangent space. (i.e. preserves the length of the vectors). That is, if we make a change of size ϵ (under the metric G_θ) to the distribution $P_\theta(\text{paths})$then carrying that change through q we should measure the same change ϵ using $G_{q(P_\theta)}$. Adding the requirement of invariance with respect to congruent embeddings leads to a unique (up to scale) metric on the manifold[Chentsov, 1980]. This metric is well-known in statistical

inference as the Fisher-Rao metric and it can be written as the Fisher information matrix [DeGroot, 1970]:

$$G = \mathrm{E}_{\mathrm{p}(\xi;\theta)}[\partial_{\mathrm{i}} \log \mathrm{p}(\xi;\theta) \partial_{\mathrm{j}} \log \mathrm{p}(\xi;\theta)] \tag{5.6}$$

5.2.1. Information Theoretic Interpretation

Another way to derive the metric is to think about "distances" on probability spaces. The KL-divergence (or relative entropy) between two distributions is a natural divergence on changes in a distribution. It measures, in a unit like *nats* or *bits* how distinguishable two different distributions are. From a hypothesis testing point of view, the relative entropy is telling us how difficult it is to tell whether or not two distributions are actually different.

It is also manifestly invariant to re-parameterization. If we think about derivatives as small changes in parameters (differentials) then we discover that we also get a unique metric as the second-order Taylor expansion of the KL-divergence.

$$\partial_i \partial_j \sum_{\Xi} p(\xi) \log \frac{p(\xi)}{q(\xi)} = -\sum_{\Xi} p(\xi) \frac{\partial_i \partial_j q(\xi)}{q(\xi)} \tag{5.7}$$

This too agrees with the Fisher information (up to scale). We note that the direction of the KL-divergence is irrelevant as to second-order $\mathrm{KL}(p,q) = \mathrm{KL}(q,p)$ [Amari and Nagaoka, 2000].

5.2.2. Machine Learning Argument

The previous two arguments provide some compelling reasons why, from a normative point of view, the Fisher information provides a sensible metric on the manifolds described by a policy class. It is interesting to note that in practice (see Section (5.6)) this particular choice of covariant ascent directions performs very well; much better in fact that standard REINFORCE . This, of course, demands a certain explanation.

We believe that the information theoretic explanation provides the requisite insight. Reinforcement learning is hard from a machine learning point of view because our policy changes induce changes in the distribution of examples the policy experiences. This is the reason that straightforward approximate policy iteration can perform badly– although we

are choosing actions that improve the policy's action-values at its current distribution of states, we are radically altering the distribution of states the policy actually visits.

From this point of view the natural policy gradient is very sensible. It chooses to modify the policy in a way that maximizes the performance improvement subject to a change in policy that is small in an important sense: that the distribution of samples the policy experiences changes a small amount. We can see this by a Langrange multiplier argument again using the second-order Taylor series of the KL-divergence as in the last argument. Intuitively then, the natural gradient works because it makes as large a change in performance for as small a change in test distribution as possible.

5.3. Fisher-Rao Metric on the Path-space Manifold

The issue now becomes how to derive the Fisher metric on the space of path distributions. It turns out in the case of processes with underlying Markovian state to be rather easy, and involves only computations we already make in the likelihood ratio approach standard in gradient-based reinforcement learning.

5.3.1. Derivation of finite-time path metric

The Fisher information metric involves computing $\mathrm{E}_{\mathrm{p}(\xi;\theta)}[\partial_{\mathrm{i}} \log \mathrm{p}(\xi;\theta) \partial_{\mathrm{j}} \log \mathrm{p}(\xi;\theta)]$. Fortunately, this is easy to do. The essential algorithm within gradient methods like REINFORCE and GPOMDP is a clever method of computing $\mathrm{E}_{\mathrm{p}(\xi;\theta)}[\partial_{\mathrm{i}} \log \mathrm{p}(\xi;\theta)]$, the expected score function. Thus, while the expected score is the gradient, the correlation of the score is the Fisher matrix. The following simple algorithm is unbiased and converges almost surely (under the same regularity conditions as in [Baxter *et al.*, 1999a] as the number of sample paths m goes to infinity:

Algorithm 3: Finite-horizon metric computation

foreach i *in* $\{1$ *to* $m\}$ **do**

  Sample a path using the policy $\pi(\theta)$: $\xi_i = (s_1, a_1, s_2, a_2, \ldots, s_n)$

  foreach t *in* $\{1$ *to* $n\}$ **do**

    Compute $z_j = \frac{t-1}{t} z + \frac{1}{t} \partial_j \log \pi(a|s;\theta)$

  $G = \frac{i-1}{i} G + \frac{1}{i} z^T z$

We use the Markov property and the invariance of the transition probabilities (given the actions) in the algorithm above to compute $\partial_i \log p(\xi;\theta)$ simply. These details can be extracted from the proofs below, as well as other, potentially better ways to sample.

To compute the natural gradient, we simply invert this matrix and multiply it with the gradient computed using standard policy search methods.

5.3.2. Limiting metric for infinite horizon problems

A well-known result in statistics [DeGroot, 1970] gives a different form for the Fisher metric under appropriate regularity conditions. We quickly derive that result, and then show how this gives us a simple form for the path probability metric as $t \to \infty$.

$$\begin{aligned} G_\theta &= \mathrm{E}_{\mathrm{p}(\xi;\theta)}[\partial_{\mathrm{i}} \log \mathrm{p}(\xi;\theta) \partial_{\mathrm{j}} \log \mathrm{p}(\xi;\theta)] \\ &= \sum_{\Xi} \partial_i p(\xi;\theta) \partial_j (\log p(\xi;\theta)) \\ &= \sum_{\Xi} \partial_i (p(\xi;\theta) \partial_j log p(\xi;\theta)) - \\ &\quad \sum_{\Xi} p(\xi;\theta) \partial_i \partial_j \log p(\xi;\theta) \\ &= -\mathrm{E}_{\mathrm{p}(\xi;\theta)}[\partial_{\mathrm{i}} \partial_{\mathrm{j}} \log \mathrm{p}(\xi;\theta)] + \partial_{\mathrm{i}} \sum_{\Xi} \partial_{\mathrm{j}} \mathrm{p}(\xi;\theta) \\ &= -\mathrm{E}_{\mathrm{p}(\xi;\theta)}[\partial_{\mathrm{i}} \partial_{\mathrm{j}} \log \mathrm{p}(\xi;\theta)] \end{aligned} \tag{5.8}$$

The third line follows from the second by integrating by parts and the fifth follows by observing that the total probability is constant.

Now we show that in the limit this leads to a simple metric for the infinite horizon case. To get a convergent metric, we must normalize the metric, in particular by the total length of the path denoted t. Since the metric is defined only up to scale, this is perfectly justified.

THEOREM 5.3.1 (Infinite-Horizon Metric). *For an ergodic Markov process the Fisher information matrix limits to the expected Fisher information of the policy for each state and control under stationary distribution of states and actions.*

Proof: We use G_θ^t to indicate the t-step finite-horizon metric.

$$\begin{aligned}\lim_{t\to\infty}\frac{1}{t}G_\theta^t &= \lim_{t\to\infty} -\frac{1}{t}\mathrm{E}_{\mathrm{p}(\xi;\theta)}[\partial_{\mathrm{i}}\partial_{\mathrm{j}}\log \mathrm{p}(\xi;\theta)] \\ &= \lim_{t\to\infty} -\frac{1}{t}\sum_{\Xi} p(\xi;\theta)\partial_i\frac{\partial_j p(\xi;\theta)}{p(\xi;\theta)} \end{aligned} \tag{5.9}$$

For a Markov process we can write the likelihood ratio $\frac{\partial_i p(\xi;\theta)}{p(\xi;\theta)}$ as

$$\frac{\partial_i p(\xi;\theta)}{p(\xi;\theta)} = \sum\nolimits_t \frac{\partial_i p(s_{t+1}|s_t;\theta)}{p(s_{t+1}|s_t;\theta)}$$

Using the chain rule we continue the derivation with $d^\pi(s)$ indicating the stationary distribution:

$$\begin{aligned}\lim_{t\to\infty}\frac{1}{t}G_\theta &= \lim_{t\to\infty} -\frac{1}{t}\mathrm{E}_{\mathrm{p}(\xi;\theta)}[\sum_{\mathrm{t}}\frac{\partial_{\mathrm{i}}\partial_{\mathrm{j}}\mathrm{p}(\mathrm{s}_{\mathrm{t}+1}|\mathrm{s}_{\mathrm{t}};\theta)}{\mathrm{p}(\mathrm{s}_{\mathrm{t}+1}|\mathrm{s}_{\mathrm{t}};\theta)} - \\ &\quad \frac{\partial_i p(s_{t+1}|s_t;\theta)\partial_j p(s_{t+1}|s_t;\theta)}{p(s_{t+1}|s_t;\theta)p(s_{t+1}|s_t;\theta)}] \\ &= -\sum_{s,a} d^\pi(s)\pi(a|s;\theta)(\frac{\partial_i\partial_j\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta)}) + \\ &\quad \frac{\partial_i\pi(a_t|s_t;\theta)\partial_j\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta)\pi(a_t|s_t;\theta)}) \\ &= \sum_{s,a} d^\pi(s)\pi(a|s;\theta)\frac{\partial_i\pi(a_t|s_t;\theta)\partial_j\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta)^2} \\ &\quad -\sum_s d^\pi(s)\sum_a \partial_i\partial_j\pi(a_t|s_t;\theta) \\ &= \mathrm{E}_{\mathrm{d}^\pi(\mathrm{s})}[\mathrm{G}_{\mathrm{s};\theta}] \end{aligned}$$

where the second equality follows by noting that the likelihood ratio is independent of transition probability matrix given the action probability, and by application of the ergodic theorem. The last line follows (with $G_{s;\theta}$ the Fisher information for the policy at a given state), as the second term in line 3 vanishes since the total probability is constant with respect to θ. ■

It is also interesting to consider what happens in the start-state, discounted formulation of the problem. In this case we would like our metric, using the general definition over path distributions given above, to naturally weigh the start start more than it necessarily is in the infinite horizon average case. It is well-known that a discount factor γ is equivalent to an undiscounted problem where each trajectory terminates with probability $1-\gamma$ at each step. We can use this fact to derive a metric more appropriate for the discounted formalism.

THEOREM 5.3.2 (Start-State Metric). *For a discounted Markov process the Fisher information matrix equals the Fisher information of the policy for each state and control under the limiting distribution of states and actions.*

Proof: The proof is very similiar to the infinite horizon case and so we simply sketch it:

$$\begin{aligned}
G_\theta &= -\mathrm{E}_{\mathrm{p}(\xi;\theta)}[\partial_{\mathrm{i}} \log \mathrm{p}(\xi;\theta) \partial_{\mathrm{j}} \log \mathrm{p}(\xi;\theta)] \\
&= -\sum_{\Xi} p(\xi;\theta) \partial_i \frac{\partial_j p(\xi;\theta)}{p(\xi;\theta)} \\
&= -\sum_{\Xi} p(\xi;\theta) \partial_i \sum_t \frac{\partial_j \pi(a_t|s_t)}{\pi(a_t|s_t)} \\
&= \sum_{\Xi} p(\xi;\theta) \sum_t \frac{\partial_i \pi(a_t|s_t) \partial_j \pi(a_t|s_t)}{\pi(a_t|s_t)^2} - \\
&\quad \sum_{\Xi} p(\xi;\theta) \sum_t \frac{\partial_i \partial_j \pi(a_t|s_t)}{\pi(a_t|s_t)} \\
&= \mathrm{E}_{\mathrm{d}^{\pi(\mathrm{s})} \pi(\mathrm{a_t}|\mathrm{s_t})}[\frac{\partial_{\mathrm{i}} \partial_{\mathrm{j}} \pi(\mathrm{a}|\mathrm{s})}{\pi(\mathrm{a}|\mathrm{s})}]
\end{aligned}$$

■

where $d^\pi(s) = \sum_t \gamma^t Pr^\pi(s_t = s)$ is the limiting distribution of states. Thus the infinite horizon (undiscounted) and start-state metrics give essentially the same metric, with only the effective weighting by states differing.

5.3.3. Metrics for Partially Observed Problems

For policies that map the observation space of a partially-observed Markov decision process into distributions over actions, it is just as easy to derive appropriate metric using our approach. The tuple (s_t, a_t, o_t) is also a markov chain, and with only subtle changes to the arguments above we end up with the same metric except using the limiting distribution of observations instead of states.

5.4. Policy Gradient Theorem and Natural Gradient

5.5. Compatible Function Approximators

Kakade noticed a fascinating connection between the limiting metric given in Equation (5.6) and the improvement direction computed by a class of actor-critic methods that use a special compatible function approximation technique[Sutton *et al.*, 1999b, Konda and

Tsitsiklis, 2002]. Following Kakade we let $\psi(s,a) = \partial_i \log p(a|s;\theta)$ and let the compatible function approximator be linear in ψ:

$$f(s,a;\omega) = \sum\nolimits_i \omega_i \psi_i$$

This type of value function approximator was initially suggested as it may be used to compute the true gradient. In practice, it has not been clear what advantage this brings over the policy gradient estimation routines of [Baxter *et al.*, 1999a]. However, "folk-wisdom" has it that performing an that infinitesimal policy iteration (moving in the direction of the best policy according to $f^\pi(s,a;\omega)$) using this approximation has very good properties and often significantly outperforms standard gradient methods. The natural gradient provides insight into this behavior. Let ω minimize the squared value-function error:

$$\epsilon(\omega;\theta) = \sum\nolimits_{s,a} d^\pi(s)\pi(a|s)(f(s,a;\theta) - Q(s,a;\theta))^2$$

where $Q(s,a;\theta)$ is the exact advantage function[Sutton *et al.*, 1999b]. It is easy to check that ([Kakade, 2002], Theorem 1) $\omega \propto {G_\theta}^{-1}\partial_i J(\theta)$. That is, the direction of maximum policy improvement is exactly the natural gradient direction. This can be seen by simply differentiating $\epsilon(\omega;\theta)$ to minimize it with respect to ω and recalling the Policy Gradient Theorem (Theorem 3.3.1) that

$$\partial_i J(\theta) = \sum\nolimits_{s,a} d^\pi(s) Q(s,a;\theta) \partial_i p(a|s;\theta)$$

5.5.1. Value Function Approximation

The central insight of the argument above relates the natural and standard policy gradient. The gradient is seen from the policy gradient theorem as the dot product of the action-value function onto the vectors $\partial_i \log p(a|s;\theta)$:

$$\nabla J = \langle \nabla \log \pi, Q \rangle_\pi \tag{5.10}$$

The natural gradient however, is the *least-squares estimate of* Q on the same basis and in the same inner product space:

$$\mathit{argmin}_{w \in \mathbb{R}^n} \| w \cdot \nabla \log \pi - Q \|_\pi \tag{5.11}$$

This now seems much more natural: if the vectors we are projecting on are not orthonormal, then we "over-count" certain directions. The least squares estimate is equivalent to first computing an orthonormal basis for $\partial_i \log p(a|s;\theta)$ and then performing the projection. In an important sense, the natural gradient is thus more fundamental then the REINFORCE computed gradient direction.

5.5.2. Covariant and Natural Versions

In the previous chapter, we discussed kernel methods that allowed us to convert similarity measures into policy classes. The policy gradient presented there does not suffer the problem of non-covariance as the Hilbert space norm doesn't depend on parameterization. It fails, of course, to be natural in the senses elaborated on in this chapter.

Instead, the insight above tells us that we should compute a least squares approximation to the action-value function on some basis that will define our policy:

$$\widetilde{\nabla} J = \mathit{argmin}_{g \in H_{k_{(\cdot,\cdot)}}} \| g - Q \|_\pi \tag{5.12}$$

where G is the linear span of $\nabla_f \log \pi$, instead of computing $\nabla J = \langle \nabla_f \log \pi, Q \rangle_\pi$.

In general, it is not obvious that if we approximate a Q function using an arbitrary basis that it can be interpreted as the covariant gradient for any policy space. However, in the case that we use a powerful kernel like the radial basis function, we are already spanning a space that is dense in the space of all functions. Equation (5.12) gives the same policy even if we include a superset of G in the minimization. This establishes a point of tangency with value-function type methods. One can interpret the procedure

Algorithm 4: Covariant Kernel Policy Search

begin

Fit empirical Q-values from Monte-Carlo rollouts of current policy using kernel least-squares in a Hilbert space dense in the space of all functions. Call the best-fit g.

Compute new policy as $e^{f(o,a)+\epsilon g(o,a)}/Z$ for step-size ϵ

end

as a kernelized version of the natural gradient. It also is a form of infinitesimal policy iteration.

Regularization does break the exact equivalence with the natural gradient. Regularization acts as an informative prior that depends on more than just policy class, but rather on our knowledge of what good policies are likely to look like. We still retain all the other advantages of the natural gradient however.

5.6. Case Study

As a consequence of the results of section 5 and [Kakade, 2002], experimental results already exist demonstrating the effectiveness of the natural gradient method. That is, all results using policy improvement with compatible function approximators are implicitly computing this result. As a demonstration, we computed analytically the natural gradient for the problem given in Section (5.1.2). This problem is interesting as it reveals some of the properties of the natural gradient method. First, it is easily checked that for a complete Boltzmann policy in an MDP the natural gradient computes:

$$G^{-1}(\theta)\partial_i J(\theta) = \sum\nolimits_{s,a} Q(s,a;\theta)\partial_i p(a|s;\theta)$$

This means that it is very similiar to the standard gradient except that it removes the weighting of states by the stationary distribution. Rather, each state is treated equally. This leads to much more reasonable results in the problem as the partial derivative component for state 1 does not shrink as the policy changes initially.

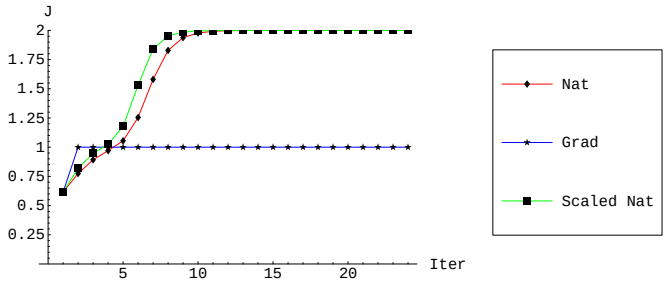


Figure 5.5. Performance comparison of covariant (nat and scaled) and non-covariant (grad) on the 2-state MDP. The covariant learning algorithm dramatically outperforms the standard gradient algorithm. The standard method achieves J=2 after more than 1000 iterations.

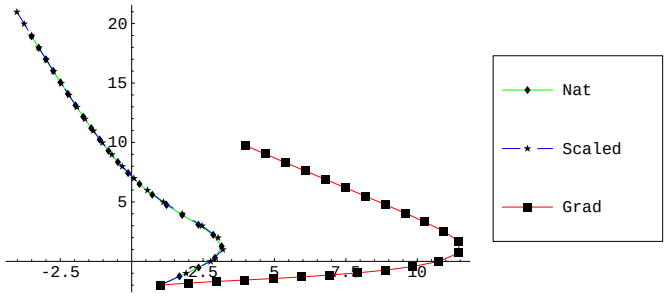


Figure 5.6. Path through policy space (showing log-odds of the actions with state 0 along the horizontal axis) of covariant (nat and scaled) and non-covariant (grad) algorithms on the 2-state MDP. Note that, as required, the path is independent of scaling for the covariant algorithm.

In Figure (5.5) we plot the performance of the natural gradient (using two different scalings) and the standard gradient methods in optimizing the policy for this problem.

It is interesting to note that in this graph the behavior of the natural gradient descent algorithm appears to be non-covariant. This is simply due to the step size heuristic not computing equivalent steps in the policy space. The direction however, *i*s constant as illustrated in Figure (5.6).

5.7. Conclusions

Kakade [Kakade, 2002] suggested that covariant behavior of gradient algorithms is an important desiderata in reinforcement learning. Unfortunately, working in the space of policies, it was difficult to generate such an algorithm. Here we proposed considering the induced path-distribution manifold and used notions from information geometry to

propose a natural covariant algorithm. This leads to interesting insight and a practical algorithm. Fortunately, it agrees with the heuristic suggested by Kakade (despite the suggestion in the paper that the algorithm there was actually not covariant) in the infinite horizon case and extends to cover new problems. [Peters *et al.,* 2003] independently developed theory related to ours (in particular related to the theorems in Section 5.3.2) and presented early results applying the natural gradient in the context of robot dynamic control.

CHAPTER 6

Derivative Propagation

THE previous three chapters have all developed solutions to the problem of learning a policy in terms of the likelihood ratio method (3.1) where the expected performance metric and its derivatives are computed by sampling. Sampling may in some instances be a very inefficient strategy. [Langford and Kakade, 2002], for instance, provides elementary examples that demonstrate that computing the *direction* of the gradient requires a number of samples that grows exponentially with the number of states of the system.

A focus of machine learning research, particularly among those interested in the Bayesian approach, has been the development of fast deterministic methods for approximate computation of various expectations. Deterministic approximations are widely used because they are often computationally more efficient than the best available Monte-Carlo style algorithms.

In this chapter, we explore the use of deterministic approximate inference algorithms for computing optimal policies in large scale problems given a compact graphical model description of the problem. We briefly discuss the fundamentals of these compact representations in the next section.

It is important to note that while the algorithmic developments in this chapter are designed in the context of decision making problems (in harmony with the thesis topic), the applications of the algorithm presented could be rather broader. At the end of the chapter, we provide some thoughts on future developments of these techniques.

6.1. Factored (Multi-agent) MDPs

Utilizing methods similar to the popular deterministic inference methods obviously requires a stronger notion of model than we have previously considered. In particular, we will assume in this chapter that we are approaching problems that are structured but have some compact graphical model description for an exponentially large state/observation space. The methods described also make sense when we have a multi-agent problem that leads to an exponentially large action space.

A factored collaborative multi-agent POMDP [Boutilier *et al.*, 1995] [Guestrin, 2003] describes most such scenarios. Most of the work in the area is based on linear programming representations of Bellman equations and assumes the MDP is fully-observable. Our work will again be developed in the context of policy gradients where there is no need to make that assumption.

Specifically, in this chapter we will assume the MDP transition and observation dynamics are compactly described in the language of graphical models [Jordan and Bishop, 2004] as a Dynamic Bayes network [Murphy, 2002]. We will further assume, as is typical in the factored MDP literature, that the reward function in our models factors as well as sum of reward functions defined on individual nodes in each time slice of the Bayes net. This turns out to be applicable to a fairly wide variety of problems. See Figure (6.1).

This kind of factored compact description of an MDP (or a POMDP in our case) has gained some currency over the last ten years [Boutilier *et al.*, 1995]. Methods have been developed that compute compactly expressible bounds on the solution to the Bellman equation [Guestrin, 2003]. In this chapter, we will pursue a policy search approach where we develop a deterministic approximation to the policy gradient.

6.2. Expectation Propagation

A recent advance in estimation was the development of the very general Expectation-Propagation (EP) algorithm. This algorithm extends the celebrated belief-propagation algorithm of Pearl (see also [Rabiner, 1989]) to general Bayesian inference with approximations in the exponential family. [Minka, 2001] has demonstrated that EP often out-performs

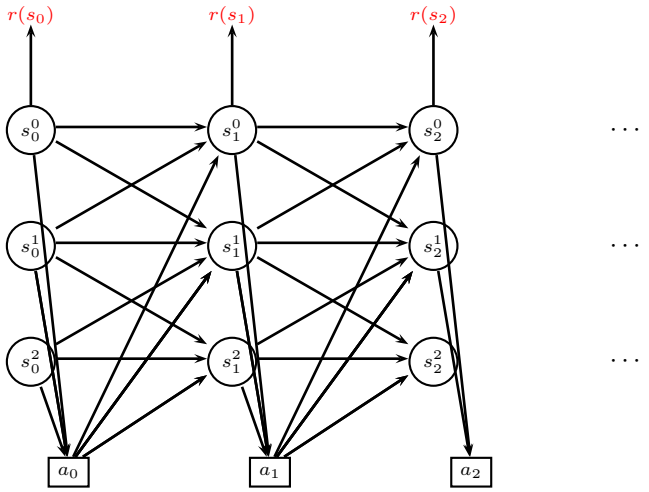


Figure 6.1. A graphical model depicting a factored probabilistic description of a Partially Observed Markov Decision Problem. We assume a (generally sparse) Dynamic Bayes Net structure to the dynamics and an additive factored reward function.

other well-known deterministic approximation techniques and is often much more computationally efficient then sampling algorithms. We first introduce EP and then suggest how it may be extended application in stochastic control problems.

Expectation-Propagation is best introduced by first discussing the Assumed Density Filter (ADF) [Maybeck, 1986]. In both EP and ADF, we employ approximating distributions of the form:

$$p(x) \propto \prod_i t_i(x) \tag{6.1}$$

We think of t_i as "evidence terms". A wide variety of problems, from Gaussian process classification to Bayes net inference to dynamic system filtering can be phrased in terms of approximating equation (6.1). Exact Bayesian inference can be performed recursively (also known as "filtering" or "online learning") by incorporating the terms one at a time. That is, we can write a posterior p' from a "prior" p that contains all the terms $i = 0 \ldots i-1$ as follows:

$$p'(x) = \frac{p(x)t_i(x)}{Z_i}$$

ADF attempts to do sub-optimal filtering by propagating an approximation to the true density in an online system. For example, a non-linear controlled system with a Gaussian distribution over states will immediately evolve in time to a non-Gaussian distribution. Generally, it becomes very difficult to maintain this complex belief state. ADF proposes that at each time step we instead maintain a distribution from a convenient approximating family of distributions, and when propagating forward in time we project back to this family to maintain this simplicity. A natural metric on beliefs is the relative-entropy or KL-divergence between distributions [Cover and Thomas, 1997]:

$$\mathcal{KL}(p||q) = \int_x dx p(x) \log \frac{p(x)}{q(x)} \tag{6.2}$$

and thus the canonical version of the ADF projects the resulting belief state at each time step to the distribution in its approximating family that minimizes this divergence. Besides the obvious applications in filtering, ADF has been proposed for numerous less obvious applications including classification, where it forms an online algorithm to compute the posterior distribution of allowed classifers. ADF, as it is clearly sensitive to the ordering of data-points (certainly odd in the classification example), can make poor approximations early in the filtering that are not easily corrected later.

Minka [Minka, 2001] made a doubly interesting realization: first, that ADF within the exponential family can be modified to correct these approximation errors due to ordering, and second that this approximation is a powerful extension to a number of existing algorithms, most notably Belief Propagation (BP). The key is to note that when performing ADF within the exponential family, there is a relationship between the prior and posterior belief states due to the approximate updates. Instead of treating ADF as performing an exact update and then projecting, equivalently it may be viewed as approximating evidence and then performing an exact update using this evidence. This is clear since we can define the effective update as simply the ratio of the posterior and prior (also canceling out the normalizer):

$$\tilde{t}_i(x) = Z_i \frac{p'(x)}{p(x)} \tag{6.3}$$

The effective update is also in the exponential family (of unnormalized distributions) as we are dividing two exponentials. With the realization that we can view ADF as an approximation of each of the evidence terms, we can take our approximate posterior made up of our original prior multiplied by these approximations:

$$q(x) = p_0(x) \prod_i \tilde{t}_i(x)$$

and *refine* it by removing one approximate term, and computing a new approximation, using the exact update t_i for it in the same way as ADF. Following [Minka, 2001], we denote an approximate distribution where we have divided out term t_i as $q^{\setminus i}$. EP iterates this procedure for each term in turn until we converge on an approximate posterior.

It is notationally quite useful[1] to join the evidence combining step (for a particular term t_i) with the projection step, (e.g. minimization of the KL divergence subject to the exponential family) into a single projection operator that we denote Γ_{t_i}.

We then present the EP algorithm in full, written in terms of Γ_{t_i}:

Algorithm 5: Expectation Propagation

Initialize term approximations $\tilde{t}_i$

Compute posterior on x from the product of $\tilde{t}_i$:

$$q(x) = \frac{\prod_i \tilde{t}_i(x)}{\int_x \prod_i \tilde{t}_i(x)}$$

repeat

Choose a $\tilde{t}_i$ to refine

Remove $\tilde{t}_i$ from the posterior to get an "old" posterior:

$$q^{\setminus i}(x) = \frac{q(x)}{\tilde{t}_i(x)}$$

Combine $q^{\setminus i}$ and t_i and thus compute a "new" posterior by performing an ADF projection:

$$q = \Gamma_{t_i}(q^{\setminus i})$$

Compute new approximate term $\tilde{t}_i$:

$$\tilde{t}_i \propto \frac{q}{q^{\setminus i}}$$

until *all* $\tilde{t}_i$ *have converged*

return

$$q(x) = \frac{\prod_i \tilde{t}_i(x)}{\int_x \prod_i \tilde{t}_i(x)}$$

[1] This also allows arbitrary projection steps instead of the standard relative entropy minimization.

6.3. Derivative Propagation

We wish now to extend EP to the problem of computing the derivatives of state variables as well as their expected values. In terms of policy learning, we are interested in the addition of the ability to compute derivatives with respect to parameters of the policy.

Clearly, the derivative of the performance function with respect to θ is simply $\int_{\Xi} d\xi R(\xi) \nabla_\theta p(\xi)$, at least in the case considered in this paper, where $R(\xi)$ is independent of θ. Thus we must compute derivatives of our approximating distribution. Below we provide an algorithm to do so using the notation of [Minka, 2001] except that we refer to the combination of incorporating evidence $t_i(\xi)$ and the subsequent ADF projection together as a single operator Γ_{t_i}.

We will assume that EP has already run, and we'd like to find the partial derivatives of the q.

Algorithm 6: Derivative Propagation

Data: Perform Expectation Propagation, computing the term approximations $\tilde{t}_i(x)$, the partial posteriors $q^{\backslash i}$ and their respective normalizers $Z_{q^{\backslash i}}$ and the total normalizer of q, Z.

Initialize $\widetilde{d_\theta t_i}$ to 0.

repeat

Choose an approximate derivative term $\widetilde{d_\theta t_i}$ to refine

Set:

$$\widetilde{d_\theta q}^{\backslash i} = \sum_l \widetilde{d_\theta t_l} \prod_{j \neq i,l} \tilde{t}_j(x) \tag{6.4}$$

Compute derivative of posterior:

$$\widetilde{d_\theta q} = \partial_\theta \Gamma_{t_i}(q^{\backslash i}) + \partial_{q^{\backslash i}} \Gamma_{t_i}(q^{\backslash i}) \widetilde{d_\theta q^{\backslash i}} \tag{6.5}$$

Compute term derivative approximations:

$$\widetilde{d_\theta t_i} = \frac{q^{\backslash i} \widetilde{d_\theta q} + q \widetilde{d_\theta q^{\backslash i}}}{{q^{\backslash i}}^2} \tag{6.6}$$

until *all* $\widetilde{d_\theta t_i}$ *have converged*

return

$$\widetilde{d_\theta q} = \frac{\sum_i \widetilde{d_\theta t_i} \prod_{i \neq j} \tilde{t}_i(x)}{Z} - \frac{\int_X dx \sum_i \widetilde{d_\theta t_i} \prod_{i \neq j} \tilde{t}_i(x)}{Z^2} \tag{6.7}$$

It is very important to note that the DP algorithm can be seen as the iterative solution of *linear* fixed point equations. While this fixed point form is often a convenient algorithm, the equations directly define an optimum and can always be solved directly. Although we have never noticed an example of non-convergence in practice, this clearly depends on the eigen-structure of the linear system. As such we can easily solve the equations directly using fast linear system solvers and guarantee fast, stable versions of DP.

A number of useful tricks can be performed to make DP slightly more efficient and numerically stable. In practice, for instance, we interweave the steps of Expectation Propagation and the derivative computations. We also use normalized messages and the appropriate derivatives to prevent numerical underflow.

It is interesting to ask what can be said theoretically about the approximation made in Derivative Propagation. The algorithm is essentially exact up to the approximations made during the projection step. Unfortunately, the error characteristics of EP remain generally poorly understood, except in very special cases. We can see, however, that when the EP approximation is exact, such as in multinomial random variables connected in a tree structure (dependency trees), the corresponding Derivative Propagation is also exact. Hopefully, further progress in understanding Expectation Propagation will shed light on Derivative Propagation's theoretical properties as well.

6.4. Instantiating EP and DP

Possibly the best known instance of Expectation Propagation is known as Loopy Belief Propagation (LBP) or the Sum-Product algorithm.

In Loopy Belief Propagation, we are attempting to compute marginal probabilities of discrete-value nodes in a factor graph or Bayes network. A factor graph consists of variables $\{x_i\}$ and *factors* ψ_α defined on subsets of the x_i. This graphical model specifies a probability distribution on the x_i by

$$p(x) = \prod_\alpha \psi_\alpha(x)$$

Recall the general form of the EP attempts to approximate is:

$$p(x) = \prod_i t_i(x)$$

This we can think about treating each factor as simply an evidence term in EP. Bayes networks [Jordan and Bishop, 2004] are a specific instance of a factor graph where each factor is a conditional probability distribution that normalizes locally so the product is also normalized. Our work focuses on inference in Dynamic Bayes Nets that have this normalization.

LBP is derived by instantiating EP with a totally factorized distribution on nodes (the vector x), and where the term approximations correspond one-to-one with the t_i. The factored approximation we describe means that we can write our approximate density as:

$$q(x) = \prod_k q_k(x_k)$$

Further, we note that the ADF projection step of minimizing the KL divergence, $\mathcal{KL}(p'(x)||q(x))$, amounts to preserving the marginal probabilities at each step. That is

$$\mathrm{E_q}[\delta(\mathrm{x_k} - \mathrm{v})] = \mathrm{E_{\hat{p}}}[\delta(\mathrm{x_k} - \mathrm{v})] \tag{6.8}$$

holds for all v that x_k can assume.

Taking these results into account, one can verify [Minka, 2001] that the framework of EP gives us the algorithm known as belief propagation. We introduce secondary subscripts on both the term approximations (known as messages in the community who use LBP) and the approximation distribution, as in $\tilde{t_{ik}}$, that represent which of the k nodes we are computing with respect to.

Algorithm 7: Loopy Belief Propagation

Initialize all $\tilde{t_{ik}} = 1$ where:

$$\tilde{t_i} = \prod_k \tilde{t_i k}$$

Compute node probabilities:

$$q_k(x_k) \propto \prod_i \tilde{t_i k} \tag{6.9}$$

repeat

Choose a $\tilde{t}_i$ to refine

Remove $\tilde{t}_i$

$$q_k^{\setminus i}(x_k) \propto \frac{q_k(x_k)}{t_{ik}\tilde{(x_k)}}$$
$$q_k^{\setminus i}(x_k) \propto \prod_{j \neq i} t_{jk}\tilde{(x_k)}$$

Compute node marginal probabilities:

$$q_k = \frac{1}{Z_i} \sum_{x \setminus x_k} t_i(x) q^{\setminus i}(x)$$

Compute term approximation ("messages"):

$$\tilde{t_{ik}} = \sum_{x \setminus x_k} t_i(x) \prod_{j \neq k} q_j^{\setminus i}(x_j) \tag{6.10}$$

until *all $\tilde{t}_i$ have converged*

As we noted before, the terms given in the algorithm here correspond to the conditional probability tables if we are working on a Bayes Network. Interpreting the $\tilde{t_{ik}}$ as

messages between nodes, we can see that each q_k is the product of all messages to a node. The $q_k^{\setminus i}$ is simply the products of all messages *except* those involving the conditional probability table described by term t_i. When $i \neq k$, $\tilde{t_{ik}}$ can be viewed as a message from i to a parent node k. For $\tilde{t_{ii}}$, on the other hand, we have a message to the node itself.

We can equally instantiate Derivative Propagation using the same approximating family and term types.

Algorithm 8: Belief Derivative Propagation

Data: Perform Loopy Belief Propagation, computing the term approximations $\tilde{t}_i(x)$, the partial posteriors $q^{\setminus i}$ and their respective normalizers $Z_{q^{\setminus i}}$.

Initialize $\widetilde{d_\theta t_i}$ to 0.

repeat

$$\widetilde{d_\theta q_k} = \sum_{l \neq i} \widetilde{d_\theta t_{lk}} \prod_{j \neq l,i} \tilde{t_{lk}} \tag{6.11}$$

$$\widetilde{d_\theta t_{ik}} = \sum_{x \setminus x_k} \partial_\theta t_i(x) \prod_{j \neq k} q_j^{\setminus i}(x_j) + \sum_{x \setminus x_k} t_i(x) \sum_{l \neq k} \widetilde{d_\theta q_j^{\setminus i}(x_j)} \prod_{j \neq i,l} q_j^{\setminus i}(x_j) \tag{6.12}$$

until *all* $\widetilde{d_\theta t_{ik}}$ *have converged*

return

$$\widetilde{d_\theta q} = \frac{\sum_i \widetilde{d_\theta t_{ik}} \prod_{j \neq i} \tilde{t_{jk}}}{Z} - \frac{\sum_{x_k} \sum_i \widetilde{d_\theta t_{ik}} \prod_{j \neq i} \tilde{t_{jk}}}{Z^2} \tag{6.13}$$

In practice, we have found it useful to perform LBP and Belief Derivative Propagation together. The algorithm dynamics are much harder to analyze, but asymptotically, Derivative Propagation still behaves like the linear system iteration.

6.5. Case Study: Multi-agent SysAdmin

To validate our algorithm we considered a variant on the SysAdmin problem. [Guestrin *et al.*, 2002a]

In this problem there are a number of computers in relative degrees of health. Each computer fails with some probability under normal conditions, and fails with greater probability if their neighbors in a directed graph (specified by the specific instance of the problem) have failed. For each computer a sys-admin can locally choose an independent action to reboot the computer, which spends the next time step rebooting. We earn reward for each computer that is operating properly, and this reward is summed over all the computers properly operating. This problem is most naturally proposed in the infinite-horizon average reward framework so we applied our algorithms in this form.

Unfortunately the precise description of the problem is not published. However, the single agent problem as published in [Schuurmans and Patrascu, 2002] provides parameters that can be adapted to the multi-agent case.

We instantiated the Belief Derivative Propagation algorithm above for the graphical model describing this problem. The dynamic bayes net that describes the problem is "rolled out" for a horizon length of approximately 20. This turns out to be well beyond the mixing time of the process. We approximated each the posterior probability of paths with a factored distribution on each computer at each time step. We experimented with a number of simple parameterized policies including one that reboots with probabilities determined by the state of the local machine, and another that considers its own state and the state of its neighbors. All policies were written as simple exponential family (i.e. log-linear functions) of the state.

The left graph in figure (6.4) illustrates the performance of BDP in calculating the derivative with respect to one parameter in a SysAdmin problem. Timing experiments here are not highly meaningful as they are all done in an interpreted language and the DP implementation is inefficient. Nonetheless, we see that the DP answer converges rapidly to a good estimate of this partial in comparison to sampling. It takes a large number of samples and a great deal of time to get similar properties from sampled gradient computation. As a deterministic approximiation, DP makes it easier to apply sophisticated optimization techniques, like conjugate gradient [Nocedal and Wright, 1999], than it is with a randomized approximation like sampling.

It is worth reiterating that both problem differences and implementation details make comparisons somewhat misleading on the problem we are considering. We are able to get

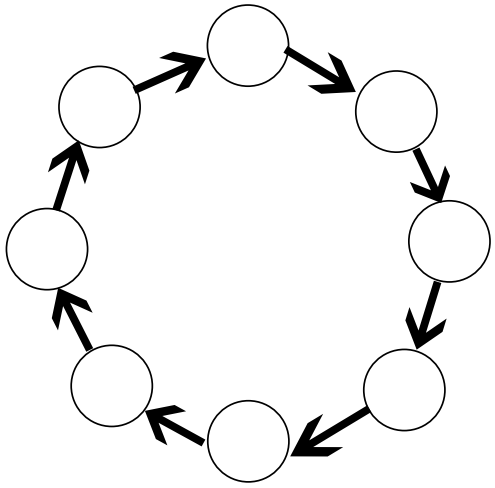


Figure 6.2. We show a ring topology for multi-agent SYSADMIN. Each computer causes the neighbor downstream in the graph to be more likely to fail.

a sense of the scaling of the Derivative Propagation approach, however. DP achieves excellent performance, and further, empirically the number gradient steps to converge appears to not scale with problem size. This leaves only scaling of the DP/EP computation itself, which appears to scale approximately linearly with the problem size. Figure 6.4 provides detailed timing experiments comparing published results for linear programming based techniques with the scaling of Derivative Propagation as we increase the number of agents in the SYSADMIN problem.

In Figure (6.3), we see a graph plotting the time dynamics of DP optimizing (here using only simple gradient ascent) the average reward. Following [Guestrin *et al.*, 2002a] we have here applied DP to maximize the average reward on a ring of 20 computers. (See Figure 6.2) This is a relatively large problem (having over a billion states and a million actions) for which simple policies suffice and are efficiently found using our algorithm. In fact, the instances solved in [Guestrin *et al.*, 2002b] and [Guestrin, 2003] are not given for graphs having over 12 agents despite run times approaching an hour. Performance graphs for sample-based REINFORCE look identical to DP except for multiplied by a constant that reflects the additional time to get a good derivative estimate. (Figure 6.4)

The DP approach here can also be applied to larger problems, and to ones with more dense connectivity, that might be quite difficult for value-function approximation methods that rely on non-serial dynamic programming [Guestrin *et al.*, 2002a] to efficiently represent the Bellman equations.

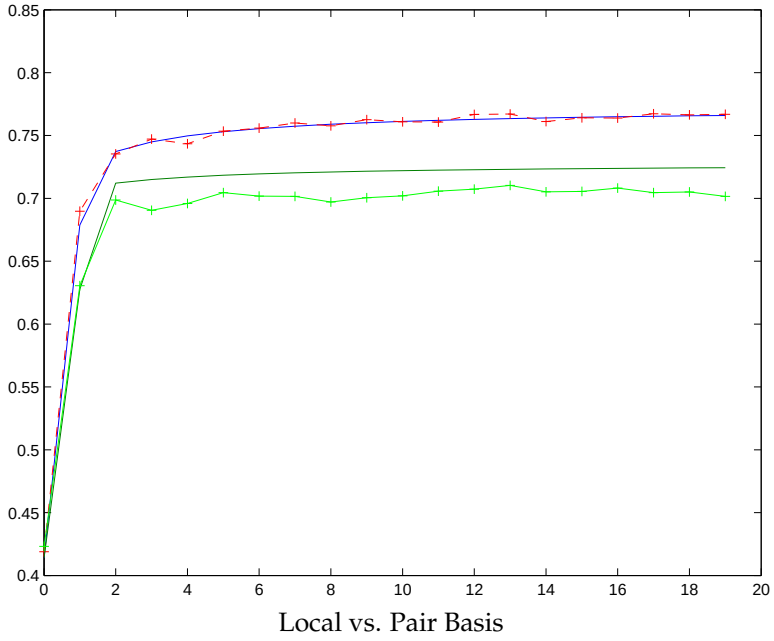


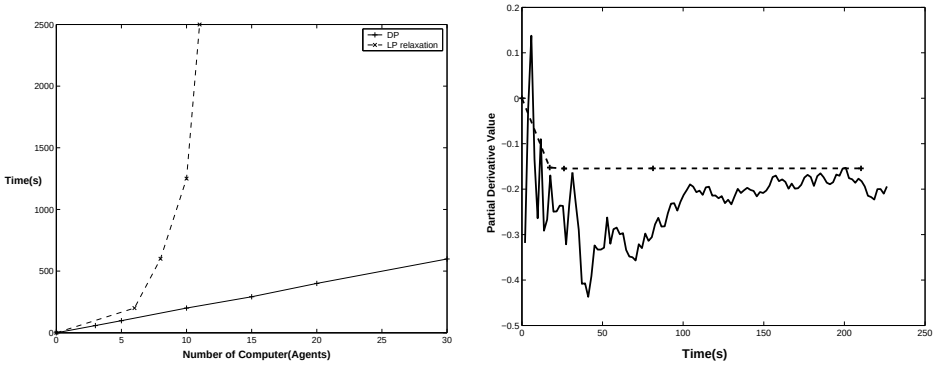
Figure 6.3. Comparison of policy performance vs. iterations of optimization: Dotted lower line is the Derivate Propagation approximated value of policy that uses only local state. Solid upper line is the approximation using local and neighbor state. Dashed lines reflect sampled evaluation (not derivative Propagation) for the policies. The bottom axis gives time in seconds.

It is interesting to notice that in the case of the local-state only policy, we see that optimization "over-fits" the EP approximation in that the approximate value becomes significantly larger than the sampling indicates is the true value of the policy. (Figure 6.3)

6.6. Uses of DP

Although we derived Derivative Propagation (DP) in terms of computing gradients of the value of policies in very large factored POMDPs, the algorithm is potentially valuable elsewhere. In multi-agent control problems, there is typically an exponentially large action space that makes computing joint, co-ordinated action probabilities and their derivatives expensive, even for sampling algorithms like REINFORCE . [Guestrin *et al.*, 2002b] has proposed a bucket-elimination style algorithm to lower the computational burden. In general, such algorithms remain exponentially expensive. Combined with Gibbs sampling for action sampling, DP can give approximations to the derivatives needed.

There are other potential uses further afield. For instance, Expectation Propagation is used to learn Gaussian process classifiers. Typically there the kernel defining the prior Gaussian process has a set of free hyper-parameters. These hyper-parameters are either



LP vs DP Gradient Comparison

Figure 6.4. (Left) We compare scalings of multi-agent SysAdmin based on a Linear Programming relaxation (data for figure from Guestin et.al.) and based on Derivative Propagation with gradient ascent. The problems are slightly incomparable (see text) however the scaling is approximately captured in the figure. (Right) Partial derivative value using DP (dotted line) and Monte Carlo (solid line) vs. time (s).

approximately integrated out using a Monte-Carlo integration technique or optimized to maximize the probability of the data. In either instance, current techniques used derivative-free optimization and integration techniques. Both could be improved by using Derivative Propagation to estimate the gradient of the evidence with respect to the hyper-parameters. Similar problems occur in setting potential functions in Markov Random Field models[Jordan and Bishop, 2004].

CHAPTER 7

Policy Search by Dynamic Programming

THE previous chapters of this thesis have considered the policy search problem as one of computing some estimate of the performance gradient, and the following that to achieve improvement. Unfortunately, that strategy, while simple and very general, can be very inefficient computationally. It may require a tremendous number of samples from the simulator, as well as be prone to falling into a potentially huge set of local minima.

Here instead we will consider an alternate approach based on *dynamic programming*. We use dynamic programming in the generic sense [1] of the kind of "algorithm design pattern" [2] where we reuse (optimal) sub-solutions. The idea of dynamic programming for control problems is summed up perhaps best by what Bellman called the principle of optimality [Bellman, 1957]:

"An optimal policy has the property that whatever the initial decision may be, the remaining decisions constitute an optimal policy with regard to the state resulting from the first decision."

The central idea is that it does not matter how a decision maker arrives at a state, rather, what matters is that given arrival at the state, the decision maker choose optimally thereafter. This insight allows us to solve problems recursively. Later in the chapter, we'll describe how our results imply a kind of "generalized principle of optimality".

[1] In contrast with the specific sense sometimes used in optimal control as referring to Bellman equation methods.

[2] I thank John Langford for that evocative phrase.

The approach taken in this chapter was directly inspired by the work of [Atkeson and Morimoto, 2003] (and related work on differential dynamic programming (DDP)) where policies are generated using as input an initial sample of trajectories. An insight provided by that paper is the usefulness of having information regarding where policies spend time. This work can be seen as a generalization of that technique that allows direct policy search instead of value functions, approximation, state uncertainty, as well as providing rigorous performance and computational guarantees.

Interestingly, an essentially similar algorithm was developed independently by Sham Kakade in his thesis work under the name *μ-policy search*. In [Kakade, 2003], Kakade focuses on the sample complexity issues in finding approximate policies for Markov Decision Processes. This complements the focus of our own work on computational complexity and partially observed decision problems. The resulting approach was jointly published in [Bagnell *et al.*, 2003a] with Andrew Ng and Jeff Schneider, who were pursuing similar insights. Here we provide more detail on these algorithms.

7.1. Preliminaries

We consider a Partially Observed Markov Decision Process with state transition probabilities compactly written as $P_{sa}(\cdot)$ (here, P_{sa} is the next-state distribution on taking action a in state s) and observation probabilities as $O_s(\cdot) = p(o|s)$. Here we take the reward function $r : S \mapsto \mathbb{R}$, which we assume to be bounded in the interval $[0, 1]$ throughout proofs.

In the setting in which the goal is to optimize the sum of discounted rewards in a Markov Decision Problem over an infinite-horizon, it is well known that an optimal policy which is both Markov and stationary (i.e., one where the action taken does not depend on the current time) always exists. Further, although this is known not to hold on the finite-horizon, under partial observability, or function approximation, it has generally been assumed to be computationally intractable to compute non-stationary policies as the size of the policy space grows exponentially with the horizon under consideration. For these reasons, learning approaches in MDPs have typically focused on searching for stationary policies (e.g., [Ng and Jordan, 2000, Langford and Kakade, 2002, Baxter *et al.*, 1999a]), often allowing stochasticity both because of the improved representational power and the existence of policy gradient algorithms like those described in previous chapters. In this work, we consider policy search in the space of non-stationary (that is, explicitly time-varying)

policies. We take advantage of what we call a "baseline distribution" that reflects a probability distribution (at each time-step) over states where a good policy spends time. Further, we show how this assumption allows us to derive an efficient algorithm.

We find in this chapter that knowing a distribution that captures where good policies spend time encapsulates much of the hardness of the learning problem. Given this distribution of states, we are able to build provably good policies efficiently. While this is obviously a strong assumption, it both illuminates the origin of the complexity of the control learning problem and provides a powerful place to insert domain knowledge. We discuss later in the chapter where the knowledge about this baseline distribution may come from, as well as techniques to apply when it is not readily available.

We consider a setting in which the goal is to maximize the sum of undiscounted rewards over a T step horizon: $\frac{1}{T}\mathrm{E}_{\mathrm{p}(\xi;\theta)}[\mathrm{r}(\mathrm{s}_0) + \mathrm{r}(\mathrm{s}_1) + \ldots + \mathrm{r}(\mathrm{s}_{\mathrm{T}-1})]$. Clearly, by choosing T sufficiently large, a finite-horizon problem can also be used to approximate arbitrarily well an infinite-horizon discounted problem (E.g., [Kearns *et al.*, 1999a]). Given a *non-stationary* policy $(\pi_t, \pi_{t+1}, \ldots, \pi_{T-1})$, where each $\pi_t : S \mapsto A$ is a (stationary) policy, we define the value

$$V_{\pi_t,\ldots,\pi_{T-1}}(s) \equiv \frac{1}{T}\mathrm{E}[\mathrm{r}(\mathrm{s}_\mathrm{t}) + \mathrm{r}(\mathrm{s}_{\mathrm{t}+1}) + \ldots + \mathrm{r}(\mathrm{s}_{\mathrm{T}-1})|\mathrm{s}_\mathrm{t} = \mathrm{s}; (\pi_\mathrm{t}, \ldots, \pi_{\mathrm{T}-1})]$$

as the expected (normalized) sum of rewards attained by starting at state s and the "clock" at time t, taking one action according to π_t, taking the next action according to π_{t+1}, and so on. Note that the linear Bellman equations hold in the form:

$$V_{\pi_t,\ldots,\pi_{T-1}}(s) \equiv \frac{1}{T}r(s) + \mathrm{E}_{\mathrm{s}'\sim\mathrm{P}_{\mathrm{s}\pi_\mathrm{t}(\mathrm{s})}}[\mathrm{V}_{\pi_{\mathrm{t}+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})], \tag{7.1}$$

where the "$s' \sim P_{s\pi_t(s)}$" subscript indicates that the expectation is with respect to s' drawn from the state transition distribution $P_{s\pi_t(s)}$. Further, we denote the action-value here for non-stationary policies by:

$$Q_{a_t,\pi_{t+1},\ldots,\pi_{T-1}}(s) \tag{7.2}$$

In the setting of this chapter, we consider a restricted class of deterministic, stationary policies Π (that is either finite or compact), where each $\pi \in \Pi$ is a map $\pi : S \mapsto A$, and a corresponding class of *non-stationary* policies $\Pi^T = \{(\pi_0, \pi_1, \ldots, \pi_{T-1}) \mid \text{ for all t, } \pi_t \in \Pi\}$. We can consider POMDP state-uncertainty problems here by simply considering the observations to be appended to the state space. In the partially observed, POMDP setting,

Π is obviously restricted to contain policies that depend only on the observable aspects of the state, in which case we obtain a class of memoryless/reactive policies. We abbreviate $V_{\pi_0,\pi_1\ldots,\pi_{T-1}}(s_0)$, as $V_\pi(s_0)$ when there is no risk of confusion.

7.2. Algorithm

Following [Langford and Kakade, 2002, Kakade, 2003], we assume that we are given a sequence of base distributions $\mu_0, \mu_1, \ldots, \mu_{T-1}$ over the states. We think of μ_t as indicating to the algorithm approximately how often we think a good policy visits each state at time t.

The algorithm, which we call Policy Search by Dynamic Programming (PSDP) is in the spirit of the traditional dynamic programming approach to solving MDPs [Putterman, 1994] where values are "backed up." In PSDP, it is the *policy* which is backed up.

The algorithm begins by finding π_{T-1}, then $\pi_{T-2}, \ldots$ down to π_0. Each policy π_t is chosen from the stationary policy class Π. Formally, the algorithm generically expressed is as follows:

Algorithm 9: Policy Search by Dynamic Programming (PSDP)

foreach $t \in T-1, T-2, \ldots, 0$ **do**

  Set $\pi_t = \arg\max_{\pi' \in \Pi} \mathrm{E}_{\mathrm{s}\sim\mu_\mathrm{t}}[\mathrm{V}_{\pi',\pi_{\mathrm{t}+1}\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})]$

Informally, the algorithm at each step chooses π from Π so as to maximize the expected sum of future rewards for executing actions according to the policy sequence $(\pi_t, \pi_{t+1}, \ldots, \pi_{T-1})$. The baseline distribution μ_t provides the start-state distribution for each step of the algorithm; that is, we can think of μ_t as indicating where a random initial state s is drawn.

7.3. The meaning of μ

Since $\mu_0, \ldots, \mu_{T-1}$ provides the distribution over the state space that the algorithm is optimizing with respect to, we might hope that if a good policy tends to visit the state space in a manner comparable to this base distribution, then PSDP will return a good policy. Define first for a non-stationary policy $\pi = (\pi_0, \ldots, \pi_{T-1})$ the future state distribution:

$$\mu_{\pi,t}(s) = \Pr(s_t = s | s_0, \pi).$$

This distribution, $\mu_{\pi,t}(s)$ is the probability that we will be in state s at time t if picking actions according to π and starting from state s_0.

Suppose now that we were given μ_{π^*}, that is, the state-time distribution associated with an optimal policy in the policy class Π^T. Then the following theorem holds for PSDP:

THEOREM 7.3.1 (Optimal Performance Guarantee). *Let $\pi = (\pi_0, \ldots, \pi_{T-1})$ be a non-stationary policy returned by exact PSDP given μ_{π^*}. Then*

$$V_\pi(s_0) = V_{\pi^*}(s_0) \tag{7.3}$$

Proof. The proof follows immediately by backwards induction: the policy found at $T-1$ performs at least as well as π^*_{T-1} We can continue backwards finding policies at each step that are as good as π^*_t, and π^* is assumed to be optimal. ■

It is in this sense that PSDP provides a kind of "generalized principle of optimality". We can say that the optimal policy has the property that whatever the initial decision may be, the remaining decisions constitute an optimal policy with regard to the *distribution* of states resulting from the first decision. Although a policy need not have direct access to state (and hence cannot act optimally with respect to the state it is in) it is enough to know the appropriate *distribution* over states that the optimal policy should find itself in. We must act optimally with respect to that distribution.

7.4. Performance Bound

In real applications we face a number of concerns. First, it is immediately clear that the applications where we have access to a μ corresponding to a good policy in the class we are considering are rather few. Further, in interesting applications we will not be able to perform the optimization step exactly. Rather, we will, as usual, be forced to approximate. Bellman equation methods are notoriously ill-behaved under approximation. The essential insight there is that backup operators may change the distribution of states to places where an approximation has been made poorly and thus amplify the errors made. It is intuitively clear that this shouldn't happen in PSDP. Rather, each time-step's policy is optimized given that the future policies are fixed. If at time step t we try to optimize the policy in a way that makes future policies perform poorly, that will be reflected in the performance at t. We

always consider the true performance of approximately optimal policies, as opposed to the approximate performance of approximately optimal policies.

The following theorem formalizes this intuition. The theorem also allows for the situation where the maximization step in the algorithm (the $\arg\max_{\pi' \in \Pi}$) can be done only approximately. We later give specific examples showing settings in which this maximization can (approximately or exactly) be done efficiently.

Some notation helps set up the result: Given two T-step sequences of distributions over states $\mu = (\mu_0, \ldots, \mu_t)$ and $\mu' = (\mu'_0, \ldots, \mu'_t)$, define the average **variational distance** between them to be

$$d_{\mathrm{var}}(\mu, \mu') \equiv \frac{1}{T} \sum_{t=0}^{T-1} \sum_{s \in S} |\mu_t(s) - \mu'_t(s)|$$

Hence, if π_{ref} is some policy, then $d_{\mathrm{var}}(\mu, \mu_{\pi_{\mathrm{ref}}})$ represents how much the base distribution μ differs from the future state distribution of the policy π_{ref}.

THEOREM 7.4.1 (Performance Bound). *Let $\pi = (\pi_0, \ldots, \pi_{T-1})$ be a non-stationary policy returned by an ε-approximate version of PSDP in which, on each step, the policy π_t found comes within ε of maximizing the value. I.e.,*

$$\mathrm{E}_{\mathrm{s} \sim \mu_{\mathrm{t}}}[\mathrm{V}_{\pi_{\mathrm{t}}, \pi_{\mathrm{t}+1} \ldots, \pi_{\mathrm{T}-1}}(\mathrm{s})] \geq \max_{\pi' \in \Pi} \mathrm{E}_{\mathrm{s} \sim \mu_{\mathrm{t}}}[\mathrm{V}_{\pi', \pi_{\mathrm{t}+1} \ldots, \pi_{\mathrm{T}-1}}(\mathrm{s})] - \varepsilon \,. \tag{7.4}$$

Then for all $\pi_{\mathrm{ref}} \in \Pi^T$ we have that

$$V_\pi(s_0) \geq V_{\pi_{\mathrm{ref}}}(s_0) - T\varepsilon - T d_{\mathrm{var}}(\mu, \mu_{\pi_{\mathrm{ref}}})$$

We prove this result by elementary means through reverse induction. As PSDP is essentially a recursive way to compute a T-step policy, this is an approach that naturally extends Theorem 7.3.1. For completeness, and because the elegant *performance difference lemma* provides useful insight into our algorithm, we include the alternate demonstration of the proof by Kakade [Kakade, 2003] [Bagnell and Schneider, 2003] in Appendix B as well.

Proof: It is enough to show that for all $t \in T-1, T-2, \ldots, 0$,

$$\mathrm{E}_{\mathrm{s} \sim \mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}}, \ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}}, \ldots}(\mathrm{s})] \leq (\mathrm{T} - \mathrm{t})\varepsilon + \sum_{\tau=\mathrm{t}}^{\mathrm{T}-1} \mathrm{d}_{\mathrm{var}}(\mu^\tau, \mu^\tau_{\pi_{\mathrm{ref}}})$$

We show that this holds by induction. The $t = T - 1$ case is a straightforward result in supervised learning, and is similar to the inductive case. The inductive argument goes as follows:

$$\begin{aligned} & \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}^{\mathrm{t}}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})] \qquad (7.5)\\ = \; & \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}^{\mathrm{t}}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s})] + \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}^{\mathrm{t}}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})]\\ = \; & \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}+1}_{\pi_{\mathrm{ref}^{\mathrm{t}+1}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}+1}},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}+1}\ldots}(\mathrm{s})] + \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}^{\mathrm{t}}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})] \end{aligned}$$

where the second equality follows directly from the Bellman equation as the immediate rewards cancel. It is easy to show that for a function f bounded in absolute value by B, a small change in probability distribution results in an equally small change in expected values of functions with respect to that distribution:

$$|E_{s\sim\mu_1}[f(s)] - E_{s\sim\mu_2}[f(s)]| \leq B\sum_s |\mu_1(s) - \mu_2(s)|$$

Since the values are bounded in the interval $[0, 1]$ we have:

$$\begin{aligned} & \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}+1}_{\pi_{\mathrm{ref}^{\mathrm{t}+1}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}+1}},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}+1}\ldots}(\mathrm{s})] + \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}^{\mathrm{t}}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})] \qquad (7.6)\\ \leq \; & \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}+1}_{\pi_{\mathrm{ref}^{\mathrm{t}+1}}}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}+1}},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}+1}\ldots}(\mathrm{s})] + \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})] + \mathrm{d}_{\mathrm{var}}(\mu^{\mathrm{t}}, \mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}})\\ \leq \; & \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi}}[\mathrm{V}_{\pi_{\mathrm{ref}^{\mathrm{t}}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})] + \varepsilon + \mathrm{d}_{\mathrm{var}}(\mu^{\mathrm{t}}, \mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}})\\ = \; & (T-t)\varepsilon + \sum_{\tau=t}^{T-1} d_{\mathrm{var}}(\mu^{\tau}, \mu^{\tau}_{\pi_{\mathrm{ref}}}) \end{aligned}$$

where the first step follows by the change of measure bound, the second by the assumption that we find an ε good policy at each time step, and the final step by the inductive hypothesis.

■

This theorem shows that PSDP returns a policy with performance that competes favorably against those policies π_{ref} in Π^T whose future state distributions are close to μ. Hence, we expect our algorithm to provide a good policy if our prior knowledge allows us to choose a μ that is close to a future state distribution for a good policy in Π^T.

Further, it is also known (see [Kakade, 2003]) that the dependence on d_{var} can be tight. Finally, it is straightforward to show (cf. [Kearns *et al.*, 1999a, Ng and Jordan, 2000]) that the ε-approximate PSDP can be implemented using a number of samples that is linear in

the VC dimension of Π, polynomial in T and $\frac{1}{\varepsilon}$, but otherwise independent of the size of the state space.

7.4.1. Benefits of Full Observability

A bound that has a multiplicative dependence on average error is also possible for the PSDP algorithm in the case that we are willing to compare ourselves against *all* policies rather than restricting to a limited class. (Although the search in PSDP may, of course, still be conducted over the limited class.) This can be seen as one of the benefits of being able to see the full state; it is reasonable in fully observed problems to learn policies at each step that compete favorably against any other policy only if we have full state observability. We state the bound as follows:

THEOREM 7.4.2 (MDP Performance Bound). *Let $\pi = (\pi_0, \ldots, \pi_{T-1})$ be a non-stationary policy returned by an ε-approximate version of PSDP in which, on each step, the policy π_t found comes within ε of maximizing the value over* all *policies. I.e.,*

$$\mathrm{E}_{\mathrm{s}\sim\mu_\mathrm{t}}[\mathrm{V}_{\pi_\mathrm{t},\pi_{\mathrm{t}+1}\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})] \geq \max_{\pi'}\mathrm{E}_{\mathrm{s}\sim\mu_\mathrm{t}}[\mathrm{V}_{\pi',\pi_{\mathrm{t}+1}\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})] - \varepsilon\,. \tag{7.7}$$

Then for all π_{ref} we have that

$$V_\pi(s_0) \geq V_{\pi_{\mathrm{ref}}}(s_0) - \sum_t \varepsilon||\frac{\mu^t_{\pi_{\mathrm{ref}}}}{\mu^t}||_\infty$$

where the infinity norm refers to the sup *of state space.*

The proof is similar to the bound given above, so we sketch the important difference to convey the benefit of observability:

Proof: It is enough to show that for all $t \in T-1, T-2, \ldots, 0$,

$$\mathrm{E}_{\mathrm{s}\sim\mu^\mathrm{t}_{\pi_\mathrm{ref}}}[\mathrm{V}_{\pi_{\mathrm{ref}^\mathrm{t}},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^\mathrm{t},\ldots}(\mathrm{s})] \leq \sum_{\tau=\mathrm{t}}^{\mathrm{T}-1}\varepsilon||\frac{\mu^\tau_{\pi_{\mathrm{ref}}}}{\mu^\tau}||_\infty$$

This again follows by induction, the inductive step being the non-trivial part:

$$
\begin{aligned}
& \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}}}[\mathrm{V}_{\pi_{\mathrm{ref}}{}^{\mathrm{t}},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})] \qquad (7.8)\\
= \;& \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}}}[\mathrm{V}_{\pi_{\mathrm{ref}}{}^{\mathrm{t}},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi_{\mathrm{ref}}{}^{\mathrm{t}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s})] + \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}}}[\mathrm{V}_{\pi_{\mathrm{ref}}{}^{\mathrm{t}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})]\\
\leq \;& \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}+1}_{\pi_{\mathrm{ref}}}}[\mathrm{V}_{\pi_{\mathrm{ref}}{}^{\mathrm{t}+1},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}+1}\ldots}(\mathrm{s})] + \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}}}[\max_{\mathrm{a}_{\mathrm{t}}}\mathrm{Q}_{\mathrm{a}^{\mathrm{t}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})]\\
\leq \;& \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}+1}_{\pi_{\mathrm{ref}}}}[\mathrm{V}_{\pi_{\mathrm{ref}}{}^{\mathrm{t}+1},\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}+1}\ldots}(\mathrm{s})] + \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi}}[\max_{\mathrm{a}_{\mathrm{t}}}\mathrm{Q}_{\mathrm{a}^{\mathrm{t}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})]||\tfrac{\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}}}{\mu^{\mathrm{t}}}||_\infty\\
\leq \;& \mathrm{E}_{\mathrm{s}\sim\mu^{\mathrm{t}}_{\pi}}[\mathrm{V}_{\pi_{\mathrm{ref}}{}^{\mathrm{t}},\pi^{\mathrm{t}+1}\ldots}(\mathrm{s}) - \mathrm{V}_{\pi^{\mathrm{t}},\ldots}(\mathrm{s})] + \varepsilon||\tfrac{\mu^{\mathrm{t}}_{\pi_{\mathrm{ref}}}}{\mu^{\mathrm{t}}}||_\infty\\
= \;& \sum_{\tau=t}^{T-1}\varepsilon||\frac{\mu^{\tau}_{\pi_{\mathrm{ref}}}}{\mu^{\tau}}||_\infty
\end{aligned}
$$

■

We are able to change measures with the infinity norm of the ratio of the probability distributions here because the action-value function optimized over a is always greater than the value of any other policy. This change of measure follows directly by multiplying inside the second expectation by $\frac{\mu_t}{\mu_t}$, and then bounding the result.

This bound is powerful in that it lets our error go to zero even if we do not get a perfect distribution μ_t as long as we drive our expected error ε to be low. We can also drop the dependence of μ_t on t, by simply averaging all the time slice distributions together

$$\frac{1}{T}\sum_{t=0}^{T-1}\mu_t$$

if we are willing to learn our policies to ε/T error.

Just as the first bound offers the insight that it is very important that the test distribution of examples match the training distribution, this bound offers an important insight as well. Essentially it tells us that it is very important when training each classifier that we put mass on all places where a good policy spends time so that the ratio $\frac{\mu^{\tau}_{\pi_{\mathrm{ref}}}}{\mu^{\tau}}$ is never too large. This makes intuitive sense– we're better to make sure our learner has seen examples of possible situations it can get into even if this means removing some of the mass from more probable instances. Crudely speaking, in choosing a μ we should err on the side of "smearing" our best guess on μ across neighboring states.

7.5. Instantiations

In this section, we provide detailed examples showing how PSDP may be applied to specific classes of policies, where we can demonstrate *computational* efficiency.

7.5.1. Discrete observation POMDPs

In the POMDP setting, in contrast with the MDP, it is known that the best memoryless, stochastic, stationary policy can perform better by an arbitrary amount than the best memoryless, deterministic policy. This is frequently cited as a reason for using stochastic policies. However, as we shortly show, there is no advantage to using stochastic (rather than deterministic) policies, when we are searching for non-stationary policies.

Four natural classes of memoryless policies to consider are as follows: stationary deterministic (*SD*), stationary stochastic (*SS*), non-stationary deterministic (*ND*) and non-stationary stochastic (*NS*). Let the operator opt return the value of the optimal policy in a class. The following specifies the relations among these classes.

PROPOSITION 7.5.1 (Policy ordering). *For any finite-state, finite-action POMDP,*

$$opt(SD) \leq opt(SS) \leq opt(ND) = opt(NS)$$

We now sketch a proof of this result.

Proof: To see that opt(ND) = opt(NS), let μ_{NS} be the future distribution of an optimal policy $\pi_{NS} \in NS$. Consider running PSDP with base distribution μ_{NS}. After each update, the resulting policy $(\pi_{NS,0}, \pi_{NS,1}, \ldots, \pi_t, \ldots, \pi_T)$ must be at least as good as π_{NS}.

Essentially, we can consider PSDP as sweeping through each time-step and modifying the stochastic policy to be deterministic, while never decreasing performance. A similar argument shows that opt(SS) $\leq$ opt(ND) while a simple example POMDP in the next section demonstrates this inequality can be strict.

■

The potentially superior performance of non-stationary policies contrasted with stationary stochastic ones provides further justification for their use. Furthermore, the last inequality suggests that only considering deterministic policies is sufficient in the non-stationary regime.

Unfortunately, one can show that it is NP-hard to exactly or approximately find the best policy in any of these classes (this was shown for *SD* in [Littman, 1994]). While many search heuristics have been proposed, we now show PSDP offers a viable, computationally

tractable, alternative for finding a good policy for POMDPs, one which offers performance guarantees in the form of Theorem 7.4.1.

A focus of this work on the problem of state uncertainty. We now address the state uncertainty problem apart from the issues of sampling error and approximation by considering finite state, observation, and control problems that are represented explicitly.

PROPOSITION 7.5.2 (PSDP complexity). *For any POMDP, exact PSDP ($\varepsilon = 0$) runs in time polynomial in the size of the state and observation spaces and in the horizon time T.*

Under PSDP, the policy update is as follows:

$$\pi_t(o) = \arg\max_a \mathrm{E}_{\mathrm{s}\sim\mu_\mathrm{t}}[\mathrm{p(o|s)Q}_{\mathrm{a},\pi_{\mathrm{t+1}}\ldots,\pi_{\mathrm{T-1}}}(\mathrm{s})] \tag{7.9}$$

where $p(o|s)$ is the observation probabilities of the POMDP and the policy sequence $(a, \pi_{t+1} \ldots, \pi_{T-1})$ always begins by taking action a. It is clear that given the policies from time $t+1$ onwards, $Q_{a,\pi_{t+1}\ldots,\pi_{T-1}}(s)$ can be efficiently computed and thus the update 7.9 can be performed in polynomial time in the relevant quantities. Intuitively, the distribution μ specifies here how to trade-off the benefits of different underlying state-action pairs that share an observation. Ideally, it is the distribution provided by an optimal policy for *ND* that optimally specifies this tradeoff.

This result does not contradict the NP-hardness results, because it requires that a good baseline distribution μ be provided to the algorithm. However, if μ is the future state distribution of the optimal policy in *ND*, then PSDP returns an optimal policy for this class in polynomial time.

Furthermore, if the state space is prohibitively large to perform the exact update in equation 7.9, then Monte Carlo integration may be used to evaluate the expectation over the state space. This leads to an ε-approximate version of PSDP where one can obtain an algorithm that is *independent* of the size of the state space and has a polynomial dependence on the number of observations, actions, T, and $\frac{1}{\varepsilon}$.

7.5.2. PSDP as Reduction

Perhaps the more common use of PSDP will be in the generative model setting where we see sample trajectories with initial states drawn from $\mu(t)$. In this setting, we can see

PSDP as a type of reduction [Langford and Zadrozny, 2004] that takes a reinforcement learning problem and converts it to a sequence of supervised learning problems. It is similar in spirit to *boosting* [Schapire, 1990] where a strong learning problem can be solved as a series of weak-learning problems.

7.5.2.1. PSDP as classification. Perhaps the natural instantiation of PSDP uses calls to a supervised learning classification algorithm.

Algorithm 10: PSDP using classification

Data: Given weighted classification algorithm C.

foreach $t \in T-1, T-2, \ldots, 0$ **do**

Sample a set of n states s_i according to μ_t:

foreach s_i **do**

foreach a *in* $\mathcal{A}$ **do**

Estimate $Q_{a,\pi_{t+1}\ldots,\pi_{T-1}}(s_i)$ by rolling out a trajectory of length $T-t$ starting from s_i and using the action a on the generative model. At each time step past the first use the previously computed (near-optimal) policies.

For each state s_i, compute the worst action and subtract its value from all the estimated state-action values of that state.

Create a training list L consisting of tuples of size $|\mathcal{A}|n$:
$\langle s_i, a, 1 - Q_{a,\pi_{t+1}\ldots,\pi_{T-1}}(s_i)\rangle$

Set $\pi_t = C(L)$, where the classifier C is attempting to minimize the weighted $0/1$ loss, or an appropriate surrogate.

In the algorithm above we have replaced the expectations with Monte Carlo estimates and the optimization step by a call to an arbitrary supervised learning algorithm. In general, if we can perform the supervised learning task at each step well, we will achieve good online performance. In some special cases, as below, it can be shown that this task as well has good computational complexity.

7.5.2.2. Linear policy MDPs. We now examine in detail a particular policy search example in which we have a two-action MDP, and a linear policy class is used. This case is

interesting because, if the term $\mathrm{E}_{\mathrm{s}\sim\mu_{\mathrm{t}}}[\mathrm{V}_{\pi,\pi_{\mathrm{t}+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})]$ (from the maximization step in the algorithm) can be nearly maximized by some linear policy π, then a good approximation to π can be found.

Let $A = \{a_1, a_2\}$, and $\Pi = \{\pi_\theta(s) : \theta \in \mathbb{R}^n\}$, where $\pi_\theta(s) = a_1$ if $\theta^T\phi(s) \geq 0$, and $\pi_\theta(s) = a_2$ otherwise. Here, $\phi(s) \in \mathbb{R}^n$ is a vector of features of the state s. Consider the maximization step in the PSDP algorithm. Letting $1\{\cdot\}$ be the indicator function ($1\{\mathrm{True}\} = 1, 1\{\mathrm{False}\} = 0$), we have the following algorithm for performing the maximization:

Algorithm 11: Linear Maximization

foreach $i = 1$ *in* $1 \ldots m_1$ **do**
- Sample $s^{(i)} \sim \mu_t$
- Use m_2 Monte Carlo samples to estimate $Q_{a_1,\pi_{t+1},\ldots,\pi_{T-1}}(s^{(i)})$ and $Q_{a_2,\pi_{t+1},\ldots,\pi_{T-1}}(s^{(i)})$. Call the resulting estimates q_1 and q_2
- Let $y^{(i)} = 1\{q_1 > q_2\}$, and $w^{(i)} = |q_1 - q_2|$
- Find $\theta = \arg\min_\theta \sum_{i=1}^{m_1} w^{(i)} 1\{1\{\theta^T\phi(s^{(i)}) \geq 0\} \neq y^{(i)}\}$
- Output π_θ

Intuitively, the algorithm does the following: It samples m_1 states $s^{(1)}, \ldots, s^{(m_1)}$ from the distribution μ_t. Using m_2 Monte Carlo samples, it determines if action a_1 or action a_2 is preferable from that state, and creates a "label" $y^{(i)}$ for that state accordingly. Finally, it tries to find a linear decision boundary separating the states from which a_1 is better from the states from which a_2 is better. Further, the "importance" or "weight" $w^{(i)}$ assigned to $s^{(i)}$ is proportional to the difference in the values of the two actions from that state.

The final maximization step can be approximated via a call to any standard supervised learning algorithm that tries to find linear decision boundaries, such as a support vector machine or logistic regression. However, using linear programming, it is possible to provably approximate this maximization. Let

$$T(\theta) = \sum_{i=1}^{m_1} w^{(i)} 1\{1\{\theta^T\phi(s^{(i)}) \geq 0\} \neq y^{(i)}\}$$

be the objective in the minimization. If there is a value of θ that can satisfies $T(\theta) = 0$, then it can be found via linear programming. Specifically, for each value of i, we let there be a constraint

$$\begin{cases} \theta^T\phi(s^{(i)}) > \kappa & \text{if } y^{(i)} = 1 \\ \theta^T\phi(s^{(i)}) < -\kappa & \text{otherwise} \end{cases}$$

otherwise, where κ is any small positive constant. In the case in which these constraints cannot be simultaneously satisfied, it is NP-hard to find $\arg\min_\theta T(\theta)$ [Amaldi and Kann,

1998]. However, the optimal value can be approximated. Specifically, if $\theta^* = \arg\min_\theta T(\theta)$, then [Amaldi and Kann, 1998] presents a polynomial time algorithm that finds θ so that

$$T(\theta) \leq (n+1)T(\theta^*).$$

Here, n is the dimension of θ. Therefore, if there is a linear policy that does well, we also find a policy that does well. (Conversely, if there is no linear policy that does well—i.e., if $T(\theta^*)$ above were large—then the bound would be very loose; however, in this setting there is no good linear policy, and hence we arguably should not be using a linear policy anyway or should consider adding more features.)

Although the [Amaldi and Kann, 1998] work presents a useful theoretical guarantee that we can adapt to our setting of weighted losses, in practice it may not be the most efficient route to building a good linear policy. Much research in machine learning has been applied to learning classifiers by minimizing a surrogate to the 0-1 loss function. The maximum-entropy family of classifiers provide such a useful approximation. [Della Pietra *et al.*, 1997] In particular, it is often useful to use techniques like weighted logistic regression.

Given a "training set" $\{(s^{(i)}, y^{(i)})\}_{i=1}^{m_1}$ (where $s^{(i)}$ are the inputs, and $y^{(i)}$ are the outputs), logistic regression minimizes the log-likelihood of the data

$$-\ell(\theta) = \sum_i \log p(y^{(i)}|s^{(i)}, \theta)$$

where $p(y = 1|s, \theta) = 1/(1 + \exp(-\theta^T s))$. In our setting, the data come with weights in proportion to the advantages, and instead we train based on:

$$-\ell(\theta) = -\sum_i w^{(i)} \log p(y^{(i)}|s^{(i)}, \theta)$$

where $p(y = 1|s, \theta) = 1/(1+\exp(-\theta^T s))$. It is straightforward to show that this is a (convex) upper-bound on the objective function $T(\theta)$.

7.5.2.3. Action-value approximation. PSDP can also be efficiently implemented if it is possible to efficiently find an approximate action-value function $\tilde{Q}_{a,\pi_{t+1}\ldots,\pi_{T-1}}(s)$, i.e., if at each timestep

$$\epsilon \geq \mathrm{E}_{\mathrm{s}\sim\mu_\mathrm{t}}[\max_{\mathrm{a}\in\mathrm{A}}|\tilde{\mathrm{Q}}_{\mathrm{a},\pi_{\mathrm{t}+1}\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s}) - \mathrm{Q}_{\mathrm{a},\pi_{\mathrm{t}+1}\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})|]\,.$$

(Recall that the policy sequence $(a, \pi_{t+1} \ldots, \pi_{T-1})$ always begins by taking action a.) If the policy π_t is greedy with respect to the action value $\tilde{Q}_{a,\pi_{t+1}\ldots,\pi_{T-1}}(s)$ then it follows

immediately from Theorem 7.4.1 that our policy value differs from the optimal one by $2T\epsilon$ plus the μ-dependent variational penalty term. It is important to note that this error is phrased in terms of an average error over state-space, as opposed to the worst case errors over the state space that are more standard in dynamic programming algorithms. We can intuitively grasp this by observing that value iteration style algorithms may amplify any small error in the value function by pushing more probability mass through where these errors are. PSDP, however, as it does not use value function backups, cannot make this same error; the use of the computed policies in the future keeps it honest. There are numerous efficient regression algorithms that can minimize this, or approximations to it.

7.6. Iterated PSDP

PSDP as presented takes as input the set of space-time distributions μ_t and generates a policy in polynomial time with non-trivial *global* performance guarantees with respect to μ_t. In many applications, we are able to provide a useful state-time distribution to the algorithm. For instance, in control and decision tasks human performance can often provide a baseline distribution to initialize PSDP. [3] We also often have heuristic policies that can be used to initialize the algorithm. Finally, domain knowledge often provides useful indications of where good policies spend time.

In any of these cases, we do not have an accurate estimate of μ for an optimal policy. A natural approach is to apply PSDP as the inner loop for a broader algorithm that attempts to simultaneously compute μs and uses PSDP to compute optimal policies with respect to it. Perhaps the most natural such algorithm is given below.

[3]The use of human performance as a baseline naturally raises the issue of trying to mimic the human's control strategy directly, instead of solving the optimal control learning problem. This is a common machine learning strategy which has met with at least a limited amount of success[Arimoto *et al.*, 1984]. There are two rather important reasons why this is not, generally speaking, the best attack on the problem. Firstly, the performance of the learner is then directly limited by the human's performance. In the case of using PSDP, however, the human merely provides baselines about where good policies spend time, and thus we may be able to achieve much better performance. The second reason is more subtle: trying to learn to match a human's control strategy using the classical techniques of supervised learning breaks many of the fundamental theoretical assumptions of supervised learning. The data, for instance, is nothing at all like identical and independently distributed. Much worse, the error metrics are seriously mismatched in a way similar to the way Bellman methods error metric are mismatched to regression error metrics. That is, a small regression learning error in terms of, for instance, squared error does not translate into small error in matching the human's control strategy. (Even in, for instance, squared error over paths.) Again, the reason is that slight deviations at one step lead to a very different test distribution at the next time step from the one the learner was trained on. It is interesting to note that some of the most serious attempts [Pomerleau, 1989] at learning human control applied a great deal effort to mitigating the mismatched error metric problem. Although it is outside the scope of this work, we believe the techniques we have developed in this chapter and with covariant policy search may provide better tools for trying to learn strategies from humans– even if the goal is only to mimic their performance.

Algorithm 12: ITERATEDPSDP(μ, π, v)

Let π_{new} = PSDP(μ)

v_{new} = Value(π_{new})

μ_{new} = ComputeInduced$\mu(\pi_{new})$ **in**

if $v_{new} \leq v$ **then**

  return π

else

  return ITERATEDPSDP ($\mu_{new}, \pi_{new}, v_{new}$)

Value here is a function that returns the performance of the policy and ComputeInducedμ returns a new baseline distribution corresponding to a policy. These can both be implemented a number of ways, perhaps the most important being by Monte-Carlo sampling.

We start ITERATED PSDP with $v = 0$ and a null policy in addition to our "best-guess" μ. ITERATED PSDP can be seen as a kind of search where the inner loop is done optimally. For exact PSDP, we can show the following:

PROPOSITION 7.6.1. *Exact* ITERATED PSDP *performance improves with each loop of the algorithm and converges in a finite number of iterations.*

Proof: Follows from Theorem 7.3.1 where $\varepsilon = 0$ and from the fact that in exact PSDP we have a finite number of policies. ■

In the case of approximation, it is less clear what guarantees we can make. Performance improvement occurs as long as we can learn policies at each step that have smaller average residual advantages than the policy we are attempting to improve over.

7.7. Case Studies

The experiments below demonstrate the instantiations described previously.

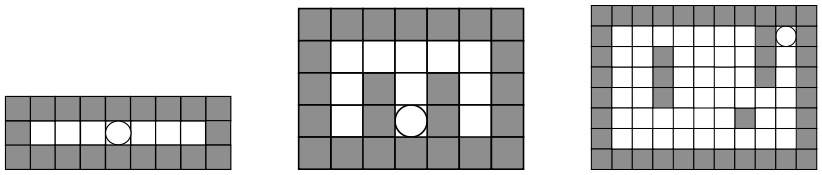


Figure 7.1. Illustrations of mazes: (a) Hallway (b) McCallum's Maze (c) Sutton's Maze

7.7.1. POMDP gridworld example

Here we apply PSDP to some simple maze POMDPs (Figure 7.7.1) to demonstrate its performance. In each maze the robot can move in any of the 4 cardinal directions. Except in (7.7.1c), the observation at each grid-cell is simply the directions in which the robot can freely move. The goal in each is to reach the circled grid cell in the minimum total number of steps from each starting cell.

Below we summarize the total number of steps to goal of our algorithm as compared with optimality for two classes of policy. Column 1 denotes PSDP performance using a uniform baseline distribution. The next column lists the performance of ITERATIVE PSDP, starting initially with a uniform baseline μ and then computing with a new baseline μ' based on the previously constructed policy. Column 3 corresponds to optimal stationary deterministic policy while the final column gives the best theoretically achievable performance given arbitrary memory.

First we consider the hallway maze in Figure (7.7.1a). The robot here is confounded by all the middle states appearing the same, and the optimal stochastic policy must take time at least quadratic in the length of the hallway to ensure it gets to the goal from both sides. PSDP deduces a non-stationary deterministic policy with much better performance: first clear the left half maze by always traveling right and then the right half maze by always traveling left.

	μ uniform	μ iterated	Optimal SD	Optimal
Hallway	21	21	∞	18
McCallum	55	48	∞	39
Sutton	412	412	416	≥ 408

McCallum's maze (Figure 7.7.1b) is a small maze discussed in the literature as admitting no satisficing determinisitic reactive policy. This is actually false as satisficing non-stationary policies do exist: PSDP provides a policy with a 55 total steps to goal from all starting locations. Although we clearly cannot outperform the 39 steps of policies with

memory in [Littman, 1994], we can achieve close to this performance. Further, by refining μ using our initial computed policy we reduce the number of steps to 48.

In another benchmark, Sutton's maze (Figure 7.7.1c) the observations are determined by the openness of all eight connected directions. PSDP generates a policy with 412 steps to goal— 4 less than the optimal memoryless policy.

It is worthwhile to note that the policy computations are very fast in all of these problems— always taking well under a second in an interpreted language. The techniques used to derive the other comparisons are all computationally burdensome: either NP-hard optimization techniques like branch-and-bound or PSPACE complete POMDP belief state planners.

7.7.2. Robot walking

Our work is related in spirit to Atkeson and Morimoto [Atkeson and Morimoto, 2003], which describes a differential dynamic programming (DDP) algorithm that learns quadratic value functions along trajectories. These trajectories, which serve as an analog of our μ distribution, are then refined using the resulting policies. A central difference is their use of value function backups as opposed to policy backups. In tackling the control problem presented in [Atkeson and Morimoto, 2003] we demonstrate ways in which PSDP extends that work.

Atkeson and Morimoto [2003] considers a planar biped robot that walks along a bar. The robot has two legs and a motor that applies torque where they meet. As the robot lacks knees, it walks by essentially brachiating (upside-down); a simple mechanism grabs the bar as a foot swings into position. The robot (excluding the position horizontally along the bar) can be described in a 5 dimensional state space using angles and angular velocities from the foot grasping the bar. The control variable that needs to be determined is the hip-torque.

In [Atkeson and Morimoto, 2003], significant manual "cost-function engineering" or "shaping" of the rewards was used to achieve walking at fixed speed. Much of this is due to the limitations of differential dynamic programming in which cost functions must always be locally quadratic. This rules out natural cost functions that directly penalize, for example, falling. As this limitation does not apply to our algorithm, we used a cost function

that rewards the robot for each time-step it remains upright. In addition, we penalize quadratically deviation from the nominal horizontal velocity of 0.4 m/s and control effort applied.

Samples of μ are generated in the same way [Atkeson and Morimoto, 2003] generates initial trajectories, using a parametric policy search. The initial policy class is a form of hybrid control linear policy. [4]

For our policy we approximate the action-value function with a locally-weighted linear regression[Atkeson, 1991]. As our policy is implicitly represented as a maximization of the action-value function over possible torques, at execution step of the policy we must perform an optimization over the action variable to choose actions during execution. The simplest optimization strategy of simply placing a fine grid over the action variable and choosing the optimum was effective.

PSDP's policy significantly improves performance over the parametric policy search; while both keep the robot walking we note that PSDP incurs 31% less cost per step. This improvement is equal to that obtained by DDP, while the algorithm is simpler. [5]

Further, DDP makes strong, perhaps unrealistic assumptions about the observability of state variables. PSDP, in contrast, can learn policies with limited observability. By hiding state variables from the algorithm, this control problem demonstrates PSDP's leveraging of non-stationarity and ability to cope with partial observability. PSDP can make the robot walk without *any* observations; open loop control is sufficient to propel the robot, albeit at a significant reduction in performance and robustness. In Figure (7.7.2) we see the signal generated by the learned open-loop controller. This complex torque signal would be identical for arbitrary initial conditions— modulo sign-reversals, as the applied torque at the hip is inverted from the control signal whenever the stance foot is switched.

[4] That is, there is a discrete state machine which selects which linear policy class to execute. State machine transitions are driven by hand-coded rules based on the continuous states of the true system.

[5] Within our ability to recreate the results. Details of the original cost function for DDP are unavailable.

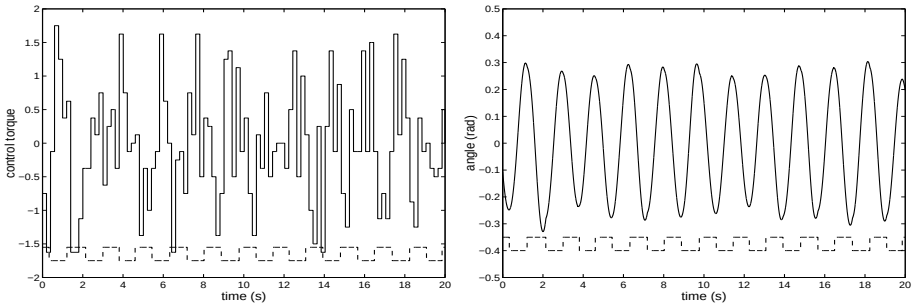


Figure 7.2. (Left) Control signal from open-loop learned controller. (Right) Resulting angle of one leg. The dashed line in each indicates which foot is grasping the bar at each time.

CHAPTER 8

Learning and Robustness

A problem that is of fundamental interest in planning and control problems is developing plans and policies that are robust to modeling errors. Robustness has always been a concern for control practitioners, and was arguably the dominant research direction of control theory in the late twentieth century. It has long been recognized in the control community that policies developed in the framework of stochastic optimal control may exhibit poor online performance due to small modeling errors.

Reinforcement learning algorithms, particularly applied to real systems where gathering data is costly and potentially dangerous, often include the need to learn some form of system model (either implicitly or explicitly) while determining an optimal policy, so as to be efficient with the data available. In this research area, however, robustness has only relatively recently been seriously explored[Schneider, 1996]. Greater autonomy for learning systems, as it necessarily precludes exhaustive validation by human engineers, demands greater emphasis on the robustness problem. Reinforcement learning methods are in fact particularly subject to concerns about robustness for at least three reasons:

1. We expect to be *learn* a system model from data on the dynamics. Physical systems are often high-dimensional so that it is quite impossible to have data for all parts of state-space. It is also unlikely than any model used by the learning algorithm is capable of capturing all of the subtlety of the real system dynamics, so we would like learning control algorithms to exhibit some degree of robustness to undermodeling. [1]

[1] More thoughts on the problem of undermodeling will be addressed in this chapter and the next.

2. We rely on optimal control (optimization) techniques. These are particularly dangerous as they may optimize a policy to spend time where approximation is actually poor. We have seen this same type of problem in policy and value-function approximation previously.
3. We are interested in systems that remove humans from the loop– including our reliance on the intuitions of control engineers.

8.1. Forms of Robustness

Traditional model-based reinforcement learning algorithms make a certainty equivalence assumption. That is, they simply take the single most probable (maximum-likelihood) model as ground truth. They then calculate optimal policies, using some variant on dynamic programming, for this maximum-likelihood Markovian model. This ignores two important types of uncertainty.

8.1.1. Model Uncertainty

The certainty-equivalent assumption fails first to account for *model uncertainty*. This kind of uncertainty is rather serious in any kind application where failure has consequences. In Chapter (10), for instance, we'll explore helicopter control. Helicopter control is a simple example of problems that are typical in robotics where we are hard pressed to collect data that allows us to learn dynamics all over state space. Further, capturing data from various parts of the state space may be dangerous.

Model uncertainty captures our lack of knowledge about the dynamics. It is important to note that model uncertainty is *not* structural; that is, we believe that some model we are considering is appropriate for the system.

Finally, it is important to note that model uncertainty differs dramatically from the noise used in stochastic control. Noise, as in the uncertainty of outcome in an MDP, is characterized by its independence. Model uncertainty, by contrast, is highly correlated. Previous work [Schneider, 1996] attempts to describe uncertainty in the model by increasing the noise. In some instances, this may improve robustness, but it fails to capture the nature of model error.

For instance, if we are uncertaint about steering bias in a ground vehicle, we cannot simply pretend that the model error is re-sampled at each time step adding to the noise. This would give us an overly conservative estimate of the drift of the vehicle that would compound like $O(\sqrt{t})$ where t is the number of time steps. In reality, any bias in the system leads to correlated error that compounds like $O(t)$. In terms of sampling, we must essentially fix a sample model at the start of a trajectory and hold that model for the entire length. This, in continuous states, typically leads to much heavier tailed t-step distributions over state than simple noise would.

8.1.2. Undermodeling

Undermodeling is a fundamentally more structural kind of uncertainty in dynamics. Undermodeling is a reflection of the truism that "All models are wrong; some models are useful" [2] Invariably, with the possible exception of some games, our models are not capable of capturing the dynamics of any of the systems we are interested in modeling. In this case, we are interested in being robust to modeling errors *that lie outside* the class of models we are learning. It seems clear that in some sense this is an essentially more difficult type of robustness.

8.2. Formal Approaches

In this section we briefly outline the approaches taken in the following chapters to the robustness problem.

8.2.1. Set Robustness

In the next chapter, we consider our uncertainty about models as a type of *set* uncertainty. We model this as a type of static game where Nature chooses from the set of possible models so as to minimize our performance, while we choose a controller that attempts to maximize our performance. The interesting part of this problem is setting up the game so that it captures a useful kind of uncertainty. That is, in what sets should our uncertainty lie so that it is both algorithmically tractable and the uncertainty set expresses well the failure

[2] That particular phrasing is often attributed to George Box.

in our knowledge of the model. We would like to include a broad range of plausible models, while eliminating as many implausible models as possible. Failure to effectively rule out models could lead to very conservative behavior and poor performance.

Later, we consider a type of dynamic game where Nature chooses to disturb the controller differently at each time step. Here we have a true response to undermodeling, where no Markov or Partially Obsserved Markov Decision Processes can capture the possible dynamic behavior. Here instead we provide a notion of bounded deviation from Markovian behavior that captures an infinitely large space of dynamics with arbitrarily long memory, or even "ESP" (predicting future moves of the controller).

Finally, we make the important observation that there are certain sets where the dynamic and static game are equivalent. This models a very natural kind of uncertainty, and give us a great deal of robustness. We show that there are efficient (polynomial time) algorithms that solve these types of problems. However, the set-based uncertainty is rather different then any of the learning decision techniques we have described before. Therefore, techniques for approximation and uncertainty here are not yet fully developed. We believe that this is an important area for future research.

8.2.2. Bayesian Robustness

Following the set-based uncertainty, we consider a different approach. Uncertainty is often well characterized by probability, understood in the Bayesian sense of "degrees of belief". We therefore explore a probabilistic account of uncertainty where we are unsure of the model dynamics. In this sense, we try to perform well for some notion of "on average" or "with high probability". The Bayesian account of uncertainty, although it may incidentally provide some measure of protection against undermodeling, is directed squarely towards model uncertainty.

This account has the advantage that we can embed it as a special kind of partial observability. Although the problem still remains computational difficult (in contrast with the set-based approach), there are some advantages over a generic POMDP. To that end, we explore the slightly broader class of POMDPs with uncontrolled state variables. We provide some analysis and development of algorithms appropriate to that class of problems.

The Bayesian robustness scheme provides some natural advantages over the set-based scheme. Perhaps most importantly, our appproaches to approximation and uncertainty all carry over. In this way, we can demonstrate these techniques on some large scale problems where other approaches have seemed to be unsuitable. In particular, we demonstrate our techniques on the control of an autonomous helicopter. This was, to the best of our knowledge, the first use of reinforcement learning techniques applied to the difficult problem of rotorcraft control, as well as one of the first applications of robustness in learning control to a physical system.

8.3. Duality with Exploration

Learning systems, and particularly those operating in the physical world where experiments are costly and time-consuming, must face the well-know exploration/exploitation dilemma. The learning system must trade off: 1) the desire to improve a model by trying out actions and states that have not been well explored (which could improve its overall performance in the future), and 2) the desire to take actions that are known to be good (which yields better near-term performance). The exploration/exploitation problem has received considerable attention. Developing strategies to explore and exploit efficiently is an extremely difficult problem– especially under constraints that are often present in real systems. As an example, consider a helicopter learning its dynamics and a control policy. We want to ensure that it will not crash while learning, or operating under a policy derived from a learned model. Intimately tied to this exploration/exploitation trade-off is the issue of building controllers that are exploration or risk-sensitive.

In fact, there is a close relationship between heuristics for good exploration and those for robustness. It turns out that in both robustness frameworks we develop, changing the optimization procedure slightly reveals an approach to exploration as opposed to one for robustness. In each chapter, we explore these connections.

CHAPTER 9

Stochastic Robustness

WITHIN some communities, and especially amongst control practitioners, it has long been appreciated that optimal control techniques can behave poorly in practice. These failures can be attributed to a failure to adequately capture model uncertainty in the optimal control framework. In this chapter, we explore approaches that take an essentially non-probabilistic approach to capturing model uncertainty.

One such approach is to abandon the framework of stochastic control entirely. [Heger, 1994] and [Morimoto and Doya, 2001] adopt this approach, replacing a probabilistic model of interactions with a deterministic, worst case evaluation criterion. [Heger, 1994]'s technique (mini-max Q-learning) assumes that at each step the worst possible result happens to an agent. As might be imagined, this can lead to rather conservative estimates of the value of a policy, particularly in the instances where a stochastic model is an accurate representation. In the worst case– a truly stochastic model where there is some small probability of catastrophe at each state– this algorithm evaluates all policies as equally abysmal performers.

Much work in the control community has been directed towards this problem, and the formulation known as H_∞ robust control has been a research focus for decades. The efficient computational solution to the linear problem has attracted a great deal of interest among practitioners. H_∞, generalized to non-linear problems, is closely related to mini-max criterion described above, as in this framework the controller is pitted against a *disturber* that can inject an L_2 bounded disturbance into the feedback loop. The controller seeks to maintain the stability of the system by attenuating this disturbance. The relation between disturbances in the H_∞ formulation and model error can be attributed to results

like the *small-gain theorem* [van der Schaft, 1999]. The relationship is particularly sharp in the linear case, where the disturber can be identified with a norm-bounded (in the Hardy-space, H_∞, hence the name) linear perturbation. In the context of finite-state machines (deterministic MDP's), H_∞ reduces to the mini-max framework described above with the addition of costs incurred against the disturber for the "energy" required for applying each disturbance. [Morimoto and Doya, 2001] show how H_∞ might be phrased in terms of reinforcement learning.

Stochastic models were developed to model situations where state transitions happen due to events that are largely unobservable and uncontrollable. While H_∞ is doubtless a useful scheme for control, there are many applications of practical interest where a stochastic model remains an excellent approximation of real system behavior, and a worst case deterministic model has both very poor fidelity with the real system and leads to controllers with performance so conservative as to have little practical value. Yet we still wish to ensure that our policies remain robust to errors made in modeling the stochastic process. A fundamentally different approach has been taken in *risk-sensitive* optimal control. In the risk sensitive framework, the basic structure of a stochastic model is retained, but the risk-neutral additive cost function is replaced with a cost function that emphasizes variance in cost occurred during control– in particular, risk-sensitive controllers will prefer a deterministic reward r to a lottery l with $E[l] = r$. This seems a natural preference– at least for human decision makers. Risk sensitive criteria can also be related to a type of model uncertainty; [Fleming and Hernandez-Hernandez, 1997] connects risk-sensitive control, where the value is defined as the time-average expectation of an exponential sum of costs, to a dynamic game, similar to the H_∞ problem, where the disturber influences next-state transition distributions but pays a penalty for the relative entropy incurred at each step between the nominal model and the disturber's preferred next state distribution. The risk sensitive criterion thus provides a link to our desiderata of robustness, but lacks strucuture: deviations in the transition probabilities at every state are considered equally and no bound is imposed to prevent possibly arbitrarily large deviations from a nominal model by the disturber.

9.1. Model Uncertainty Sets

In this chapter, we address the *stochastic robustness* problem more directly. We'll consider the problem of finding a stationary, Markovian policy that performs optimally in cumulative discounted reward in an uncertain Markov Decision Problem with a finite number of states and actions. Uncertainty will be described in terms of a set (henceforth the *uncertainty set*) of possible transition matrices, and optimality will be defined as maximizing cumulative discounted reward starting from a given state (equivalently a distribution over states). As the only thing to vary between the processes will be the transition functions, we will abuse notation somewhat and denote each MDP and its transition matrix, $P_{ia}(j) = Pr(j|i,a)$ with the same symbol. We denote the uncertainty set of transition probabilities by $\mathcal{P}$, and the set of next state distributions drawn from $\mathcal{P}$ corresponding to a state i and control a as $\mathcal{P}_i^a$.

Unlike previous chapters, we will specifically ignore issues related to approximation and uncertainty. Thus the observation model is considered to be direct state access.

9.2. Static Robustness Game

To ensure robustness we wish to find controllers that perform well on *all* models in the uncertainty set. Specifically, we envision a static game in which a stationary Markovian controller is chosen, and then a model is chosen (by "Nature") from the uncertainty set so as to minimize the expected reinforcement received by the controller.[1] That is, the game's value (to the controller) is defined as:

$$\max_{\pi\in\Pi} \min_{p\in\mathcal{P}} (1-\gamma)\mathrm{E}_{\mathrm{p},\pi}\left[\sum_{\mathrm{t}} \gamma^{\mathrm{t}}\mathrm{r}(\mathrm{s}_{\mathrm{t}})\right] \tag{9.1}$$

9.2.1. General Sets

It is unfortunately the case that finding policies that are guaranteed to be good across unrestricted sets of models is computationally challenging. A reduction similar to Littman's

[1] In the original description of this work, given first in [Bagnell and Schneider, 2001], the text correctly identifies the decision order of the game, but in the equation, the order between players is reversed. An interesting aspect of this particular dual game is that in a broad class of problems, the ordering turns out not to matter. [Nilim and el Ghaoui, 2004]

proof [Littman, 1994] of the hardness of finding optimal memoryless policies in POMDPs proves the following result:

THEOREM 9.2.1. *Finding the stationary memoryless policy that maximizes the least expected reward over an uncertainty set of Markovian Decision Problems is NP-hard.* ■

Proof: *As with similar proofs in this work, we show that an algorithm that could solve the general set stochastic robustness problem would allow us to efficiently find satisfying assignments in* 3-SAT*.*

Given a 3-CNF *formula (e.g.* $(x_1 + \bar{x_2} + x_3)(x_4 + \cdots)$*) we build a general set-uncertain MDP. To each clause we assign a model. In all models we start in state* s_0*. For model* i*, corresponding to clause* i*, there is a state for each variable. The state transition structure for that clause is such that we transition from* s_0 *to a state* s_j *corresponding to the first variable in clause* i*. If we choose an action* s_0 *so that the action (either* 0 *or* 1*) would satisfy this first term (i.e. if* $\pi(s_j) = 0$ *if clause contains* $\bar{x_j}$ *or* $\pi(s_j) = 1$ *if clause contains* x_j*) then we transition to a special state* s_{goal} *and earn reward* 1*. If our policy incorrectly assigns that variable, we move on to the next state representing the next term in the clause. We repeat this process for the whole clause. If we assign all the state terms incorrectly in a clause, we transition to an absorbing state denoted* s_{fail}*. An algorithm that could find* $\pi^\star$*, an optimal map from state to action, would give an assignment to each variable that satisfies if and only if there is a satisfying assignment. Therefore no such algorithm exists unless P=NP.* ■

9.2.2. Factored Convex Sets

Let us now consider more restricted uncertainty sets that we will show are both broad enough to be quite interesting with regard to applications, yet restrictive enough to admit tractable solutions.

DEFINITION 9.2.1. *We call an uncertainty set of transition functions* **factored-convex** *if for every action* a*, every state* i*, and any two transitions kernels* T_i^a *and* S_i^a*, all distributions "on the line" between the two sets are included. That is, all transition functions of the form*

$$U_i^a = \alpha(i,a)T_i^a + (1 - \alpha(i,a))S_i^a \tag{9.2}$$

for any function $\alpha(i,a)$ *taking its values in the range* $[0,1]$*, are also in the uncertainty set.*

To support the claim that *factored-convex uncertainty sets* are interesting, we provide some examples.

EXAMPLE 9.2.1. *The class of uncertainty sets known as* interval-MDPs *[Wurman and Wellman, 1996] where every element of the transition matrix has bound constraints such as $\zeta^a_{ij} \leq P^a_{ij} \leq \psi^a_{ij}$ is a factored-convex uncertainty set.*

EXAMPLE 9.2.2. *The class of uncertainty sets where each row of the the transition matrix is constrained to lie in some* relative-entropy *ball around a nominal distribution ζ^a_i, $\{P^a_i | \mathcal{KL}(P^a_i|\zeta^a_i) \leq \epsilon^a_i\}$, describes a factored-convex uncertainty set. Equally the set corresponding to the reversed divergence, $\{P^a_i | \mathcal{KL}(\zeta^a_i|P^a_i) \leq \epsilon^a_i\}$ is factored-convex. The latter we refer to as a KL-divergence ball around ζ^a_i, which is useful in that we can provide PAC-style performance guarantees for reinforcement learning using these uncertainty sets. (See section 9.4.2).*

EXAMPLE 9.2.3. *In the mini-max model next states are chosen independently and deterministically for each state-action pair. The uncertainty set consisting of for each state-action pair the convex hull of the possible next states in the probability simplex constitutes a factored-convex uncertainty set.*

A fundamental requirement of Definition 9.2.1 is that the uncertainty *factor* by state-action pair– that is, for each state-action pair the next state distribution can be chosen independently of the next state-distribution chosen elsewhere in the state space.

9.3. Dynamic Game

This condition, Equation 9.2, suggests the introduction of a stochastic *dynamic* game with turns between a controller that chooses an action a at each time step and a disturber who counters with the next step distribution $P(j|i', a)$ from a set of allowable next state distributions $\mathcal{P}^a_i$. The value of each state i for the controller in the game is defined, with $s_0 := i$, as the (lower) limit of the discounted cost averaged over trajectories.

$$\min_{\pi_{0\cdots\infty}\in\Pi^\infty} \max_{p_{0\cdots\infty}\in\mathcal{P}^\infty} (1-\gamma)\mathrm{E}^{\pi,\mathrm{p}}\left[\sum_{\mathrm{t}=0}^{\infty}\gamma^{\mathrm{t}}\mathrm{r}(\mathrm{s}_{\mathrm{t}})\right] \tag{9.3}$$

We note that the dynamic game described above (equation 9.3) is a zero-sum one of complete information. It follows from well know results in game theory [Basar and Olsder, 1995] that the following holds:

LEMMA 9.3.1. *If $\mathcal{P}_x^a$ defines a compact region of the probability simplex for every x and a, then there exists for both the controller and disturber Markovian, stationary, and deterministic optimal strategies.*

Proof: *This follows from noting that in the one time-step game the reinforcement incurred is continuous over a compact region and hence its minimum is attained. An appeal to the contraction mapping theorem standard in discounted MDPs and stochastic games proves the result [Littman and Szepesvri, 1996].* ■

The dynamic game is very natural in terms of the undermodeling problem that we described in detail in the previous chapter. In fact, it allows us to manage a broad new class of decision problems.

9.3.1. Solving Nearly Markovian Decision Problems

Undermodeling is a certainty; compromise is an inevitable consequence of the need for computationally tractable models– both learned and hand-designed. Ensuring that control algorithms are not brittle with respect to these unmodeled dynamics important in both model-based reinforcement learning and and planning problems. We briefly consider some notions of stochastic processes that behave as if they were "nearly" Markovian, and then discuss algorithms designed to behave robustly to this form of perturbation.

DEFINITION 9.3.1. *Two controlled stochastic processes, P and Q are ϵ-close in variation if $P(s'|s,a) - \epsilon \leq Q(s'|s,a) \leq P(s'|s,a) + \epsilon$ holds for all s, s', a and all possible histories of the process.*

DEFINITION 9.3.2. *Controlled stationary stochastic process P is ϵ-close in relative entropy to processes Q if for all possible histories of each process*

$$sup_{s,s',a}\mathcal{KL}(P(s'|s,u)|Q(s'|s,a)) \leq \epsilon$$

DEFINITION 9.3.3. *A controlled stochastic process X is* **boundedly non-Markovian in variation**, *with bound ϵ if there exists a Markov process with transition matrix T such that T and X are ϵ-close in variation. T is denoted the nominal model for X.*

DEFINITION 9.3.4. *A controlled stochastic process X is* **boundedly non-Markovian in relative-entropy** *if there exists a Markov process with transition matrix T such that X is ϵ-close in relative entropy to T. T is denoted the nominal model for X.*

We first observe that MDP solutions possess some inherent robustness to bounded non-Markovian behavior of a process:

THEOREM 9.3.1. *Solving for the optimal policy in a nominal Markovian decision problem T, corresponding to a boundedly non-Markovian decision problem (with bound ϵ in either variation or relative entropy) induces error over the optimal policy that is polynomially bounded in ϵ.*

Proof: *Note that bounded relative entropy implies polynomially bounded variation and thus w.l.o.g. we consider only the variation case. At each time step the loss is bounded by $2\epsilon r_{\max} = 2\epsilon(1-\gamma)$, so that the total error is bounded by 2ϵ.* ■

9.3.2. Robust Value Iteration

We would like an algorithm that specifically tackles the uncertainty however. It is easy to see from Lemma (9.3.1) that we can devise a dynamic programming type algorithm of the correct form:

Algorithm 13: Robust Value Iteration

Initialize V to the zero vector the size of the state space.

repeat

  foreach *states i and controls a* **do**

    Assign to matrix $Q_{\min}$ the solution to the optimization problem:

$$Q_{\min}(i,a) = \min_{p \in \mathcal{P}_i^a} \gamma E_p[V] + (1-\gamma) r(i)$$

  Update V by maximizing over the control set:

$$V(i) = \max_{a \in \mathcal{A}} Q_{\min}(i,a)$$

until *V converges*

Figure (9.1) illustrates how Robust Value-Iteration backups are performed. This algorithm can also be easily extended to run asynchronously and online. It is not clear however, that Algorithm (13) leads to a computable solution, as the inner loop requires a minimization over an uncountable set of probability distributions. Perhaps surprisingly it is not only computable but tractable:

THEOREM 9.3.2. *Finding a (near)-optimal policy to guarantee performance (within a given ϵ) for the dynamic game in Equation 9.3, if the uncertainty set that the probability distribution is*

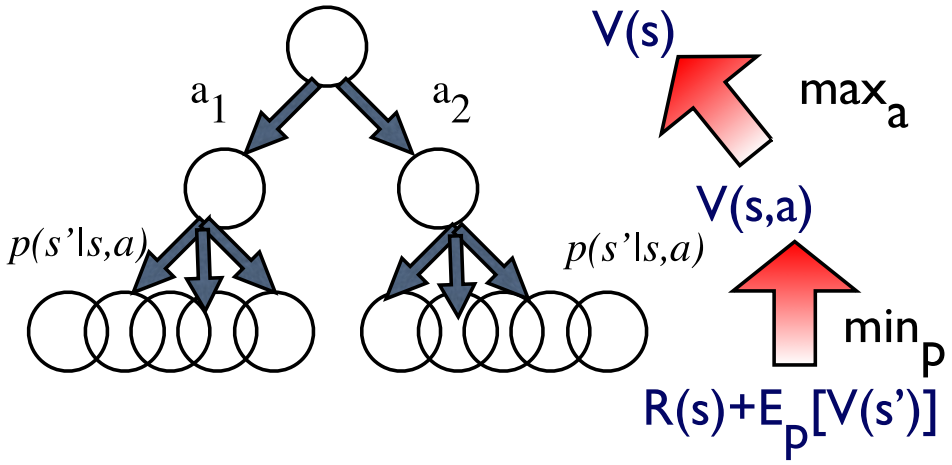


Figure 9.1. Above we show the backup diagram associated with Robust value-iteration. For each possible action of the controller, the disturber chooses from a (convex) set of possible next state distributions.

chosen from is compact and convex, is a polynomial time problem, and Algorithm (13) solves the problem in polynomial time.

Proof: *The critical thing to note that the minimization required in the algorithm is a convex program: The expectation is linear in p and thus forms a linear objective function subject to convex constraints. It is known that convex programs can be solved in polynomial-time (in the number of states) by interior point (e.g. ellipsoidal algorithm) methods [Boyd and Vandenberghe, 2004]. Further, each epoch of value-iteration reduces the error geometrically, so that in time logarithmic in ϵ, the maximum allowed sub-optimality, an approximately optimal value-function can be computed. A greedy policy with respect to the approximately optimal value function leads to a nearly optimal policy.* ■

9.3.3. Modeling H_∞ as a Stochastic Robustness Problem

As discussed in the introduction, methods other than stochastic optimal control with respect to the nominal model exist for mitigating the impact of non-Markovian behavior. Of particular interest is the H_∞ paradigm [Fleming and Hernandez-Hernandez, 1997], where energy costs are associated with disturbances injected at each time step.

The simple modification of step (3) of Algorithm (13) to include energy costs as follows:

$$Q_{\min}(i,a) = \min_{p \in \mathcal{P}_i^a} \mathrm{E_p}[\gamma \mathrm{V} + (1-\gamma)\mathrm{r(i)} + \lambda \mathrm{S(i,j)}] \tag{9.4}$$

where $S(i,j)$ is a non-negative cost associated with each transition (from i to j), representing energy injected by the disturber to induce unfavorable transitions and λ is the scalar H_∞ coefficient, generalizes the standard discrete state, discounted H_∞ problem. It is easy to recover the standard formulation by relaxing all of the next-state distribution constraints to include the entire simplex, and ensuring that for every state i there exists a j so $S(i,j) = 0$.

Noting that the dynamic game introduced in Section (9.3.1) does not require the Markov property, and recognizing that both definitions of boundedly non-Markovian processes form convex and compact uncertainty sets, we immediately have an algorithm that computes the strongest possible performance bounds on this very general class of controlled stochastic processes, and computes stationary, Markovian policies that achieve that bound.

9.4. Equivalence

Note that value-iteration in the dynamic game implicitly gives a policy for both the disturber and the controller by look-ahead in the one-step game. Surprisingly, although the dynamic game was introduced as a relaxation of the game defined in equation (9.1), as it appears to give much more power to the controller and disturber, the result in Lemma (9.3.1) implies that under the further assumption that $\mathcal{P}$ is a factored-convex uncertainty set (which as noted factors by state-action pairs), the dynamic game solution is *also* a solution to the static game (9.1) of uncertain models described above:

THEOREM 9.4.1. *If $\mathcal{P}$ is a compact and factored-convex uncertainty set, then the optimal disturbance is a stationary, Markovian distribution over next states and can be identified with an element of $\mathcal{P}$. Robust value-iteration in a compact factored-convex uncertainty set converges to the value function of the optimal policy with respect to the worst model in $\mathcal{P}$.*

Interestingly, the proof only relies on the factored aspect of the convex uncertainty set. Convexity of the next-state distributions is required for the stronger result:

COROLLARY 9.4.1. *Finding the (near)-optimal policy to guarantee performance (within a given ϵ) in a compact and convex uncertainty set of MDPs is a polynomial time problem, and Algorithm (13) solves the problem in polynomial time.*

Proof: *This follows immediately from equivalence in Theorem (9.4.1) and the efficiency result in Theorem (9.3.2).* ■

We see that in sharp contrast to the general problem, factored-convex uncertainty sets lead to a tractable solution to the uncertain model control problem. The algorithm presented, as it requires convex programs to be solved at each step can be quite expensive in practice, despite the polynomial time guarantees. However, the algorithm naturally takes advantage of structure in the uncertainty set. In the case of variation bounds on the probabilities, the convex program reduces to a linear program, and so for small structured uncertainty sets, the LP can reduce to a simple minimization over a finite set of possible disturbances corresponding to the vertices of the constraint set. Further, in the special case of relative entropy bounds the results of [Fleming and Hernandez-Hernandez, 1997] can be generalized to give a very efficient (constantly more work than standard value-iteration) solution to the robust MDP problem.

9.4.1. Fast Solutions for Relative Entropy Balls

In very recent work [Nilim and el Ghaoui, 2004], a duality result given first (to the best of our knowledge) in [Fleming and Hernandez-Hernandez, 1997] is generalized to make the solution to the inner optimization of Robust Value Iteration very fast for relative entropy balls. They also present a similar fast algorithm for the KL-divergence ball. We include the latter argument as it is particularly useful for giving an algorithm for model-based reinforcement learning that rapidly computes policies with strong PAC-style performance bounds. The argument is essentially an instance of Lagrange duality. (See Appendix A)

Suppose $\mathcal{P}_i^a$ is defined as the relative entropy ball around some nominal value q. Consider the inner problem of computing (for $r' = (1-\gamma)r$):

$$Q = \min_{p \in \mathcal{P}_i^a} \gamma E_p[V] + r'(i)$$

We now take the Legendre dual (see Appendix A) of this problem. We form the Lagrangian [2]

$$L(p, \lambda, \mu) = \gamma E_p[V] + r'(s) + \lambda \left(\sum_{s'} q(s') \log \frac{q(s')}{p(s')} - \epsilon \right) + \mu \left(\sum_{s'} p(s') - 1 \right) \tag{9.5}$$

[2] It will be turn out that the constraints enforcing non-negativity are naturally respected without explicitly adding these constraints, so we leave them out for brevity.

and minimize with respect to the p. Because of convexity and differentiability we can do this by setting $\frac{\partial L}{\partial p(s')}$ to zero. This gives us:

$$\frac{\partial L}{\partial p(s')} = \gamma V(s') - \lambda \frac{q(s)}{p(s)} - \mu \tag{9.6}$$

Solving this gives us the optimal p

$$p^*(s') = \frac{\lambda q(s')}{\gamma v(s') + \mu}$$

and plugging into the Lagrangian and simplifying gives us the following simple dual:

$$Q^*(\lambda, \mu) = -\mu + \lambda(1 - \epsilon) + r' + \lambda \sum_{s'} q(s') \log(\gamma v(s') + \mu) \tag{9.7}$$

It is easy to check that for any fixed μ, the Lagrange multiplier λ is maximized by setting it to normalize $p^*(s')$. This leaves us with a one dimensional optimization of the dual Q^* in terms of μ. We can efficiently solve this using a line search. Note that $\mu > 0$ is a valid starting point for the search, and the line search is convex. There are many efficient methods to solve such problems [Boyd and Vandenberghe, 2004]. Finally, we note that strong duality holds in this problem (by Slater's condition, Appendix A).

This dual gives us an algorithm that computes robust strategies with essentially *constantly* more work than standard value iteration. This is a significant improvement over the generic convex set optimization, or even for simple ℓ_∞ or ℓ_1 type bounds. Finally, we note that the reversed KL-divergence also has efficient (in fact simpler to prove) duals that are constantly more work than value-iteration. These duals are exactly those given in risk-sensitive control, but they don't share the strong statistical guarantees like those we provide in the next section.

9.4.2. Guaranteed Performance Reinforcement Learning

This framework of uncertain MDPs is very natural for model-based reinforcement learning. Through it, and the use of Probably Approximately Correct style sample-complexity bounds [Vapnik, 1995] on the possible transition matrices, we enable the learning of controllers that can guarrantee (with high probability) performance *while learning*, ensuring robustness that the certainty-equivalent approach cannot.

For example, it is easy to show using the *method of types* from information theory [Cover and Thomas, 1997] that given n independent samples of a particular state-action

transition, the probability of the empirical estimate being too far away (in KL-divergence) from the true distribution is exponentially small in the number of samples:

$$P(\mathcal{KL}(\hat{P_{sa}}|P_{sa}) \geq \epsilon) \leq 2^{-n\epsilon}(n+1)^{|S|} \tag{9.8}$$

For any desired probability of success, $1-\delta$, we can solve for the corresponding ϵ-ball above. Further, to be sure that the success probability applies to the entire MDP, we invoke the union (i.e. Bonferoni) bound and solve the above for ϵ_s^a with probability $\frac{\delta}{|S||A|}$.

It follows that robust value-iteration using the empirical probability distribution over state-action transitions $P_{sa}(\cdot)$ with the ϵ_s^a KL-bounds for each state-action pair returns a policy whose performance is guaranteed with probability $1-\delta$. As is typical of large-deviation bounds, because of the exponentially small probability of failure, we can typically choose δ to be very small.

As a result of the last section, we can provide these performance bounds for reinforcement learning with only a constant computational cost more than standard dynamic programming.

9.5. Case Studies

9.5.1. Path Planning

Robot path planning has been the subject of a great deal of research effort, but the vast majority of it concentrates on static obstacles. Recent research on mobile robots operating in crowded domains [Burgard *et al.*, 1999] has revealed some of the limitations of this approach, and further recent research [Montemerlo and Thrun, 2001] has demonstrated that tracking dynamic objects is feasible using sequential Monte-Carlo filtering applied to the laser range-finder data collected by the robot. A significant difficulty in planning with dynamic obstacles is the difficulty of modeling such objects– the dynamics of a human obstructing the robot's path are stochastic and possibly very high dimensional.

We developed a new planner for the CARMEN [Montemerlo *et al.*, 2003] project, modeling a dynamic obstacle in the robot's path in the uncertain MDP framework. Specifically, the dynamic obstacle is modeled as a "lazy" random walk with unknown and time-varying drift. In the eight-connected grid used for planning, the dynamic obstacle is subject only

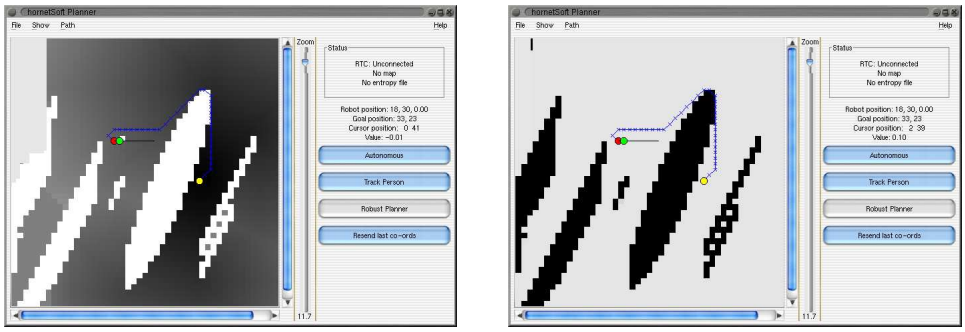


Figure 9.2. These figures illustrate a path planned by the robot (dark leftmost circle) giving a dynamic obstacle (lighter circle adjacent to the robot) a wide berth on its route to the goal (light circle on far right). The figure on the left illustrates the value function computed by the robust dynamic programming algorithm while the figure on the right illustrates the path taken.

to the linear constraints:

$$Pr(\textit{Obstacle doesn't move}) \geq .25$$

This gives a vast space of possible dynamics, including ones with arbitrary memory or predictive power, for the object whose details we leave unspecified. In this model the dynamic obstacle is thought of as "inadvertently adversarial" (perhaps not an unrealistic assumption for some humans interacting with the robot), and we require path planning to remain robust to all such obstacles.

In many situations, paths resulting from the uncertain MDP solutions show greater deference to the impedance caused by the dynamic obstacle than solutions that treat the object as static. Figure (9.2) illustrates the robot circuiting the obstacle, and the resulting value function. Lighter regions indicated regions of higher cost. As the resulting plans are full feedback strategies, the illustrations depict the plan resulting from the object staying still. (The computed strategy, of course, does not.) A conventional planner skirts the obstacle to minimize cost.

In the Figure (9.3) we compare solutions derived by the standard planner and the robust planner, noting that to avoid interference from the dynamic obstacle, the robust controller takes a longer and very different path to the goal, despite the existence of a feasible (although potentially quite difficult) path between the dynamic obstacle and the other impediments in the environment.

The behaviors engendered by this robust planner have some very desirable characteristics for planning with dynamic obstacles. The behavior is more conservative than

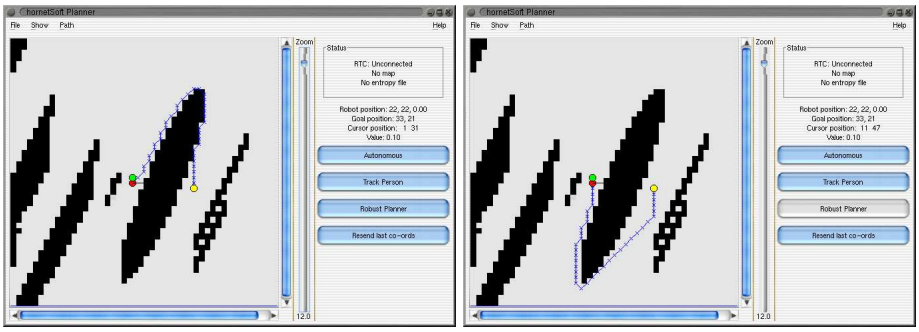


Figure 9.3. The left figure shows the plan generated by the conventional dynamic programming algorithm. On the right the uncertain MDP solution leads to a path (from the dark circle robot to the light circle goal) that makes a greater effort to avoid the dynamic obstacle.

stochastic control would induce. Mini-max control is all but useless, as the obstacle can almost always impede the robot's motions. Stochastic robustness provides a valuable compromise.

9.5.2. The Mountain-Car POMDP

We briefly illustrate how the algorithm described in the section above can be used to solve certain partially observed and discretized problems. In particular, we consider a variation on the well-studied "Mountain-Car" problem, where unlike the standard formulation, only an approximate, discrete representation of the state is available for control.

The dynamics used are the standard ones given in [Sutton and Barto, 1998] (and used in previous chapters), but complete state observability is replaced by discrete-valued sensors that output the car position to within .085 position units ($\frac{1}{20}$ of the car's position range) and .007 velocity units ($\frac{1}{20}$ of the maximum speed). Two approaches are presented to solving the problem. First the state uncertainty is treated by approximating the problem with a Markovian one. Assuming that for a given sensor measurement all states that result in this sensor measurement are equally likely, we compute transition probabilities by sampling one step forward from a uniform distribution over each sensor grid cell. Standard dynamic programming is run on the resulting discrete MDP. Next, we use the robust value iteration described above to model the possible error introduced by assuming the problem is a discrete, fully-observed MDP. Sampling is again used to compute possible next state distributions, and the minimization step of Algorithm (13) is approximate with a simple minimization over a finite sample set. To ensure that the mountain car doesn't get "stuck"

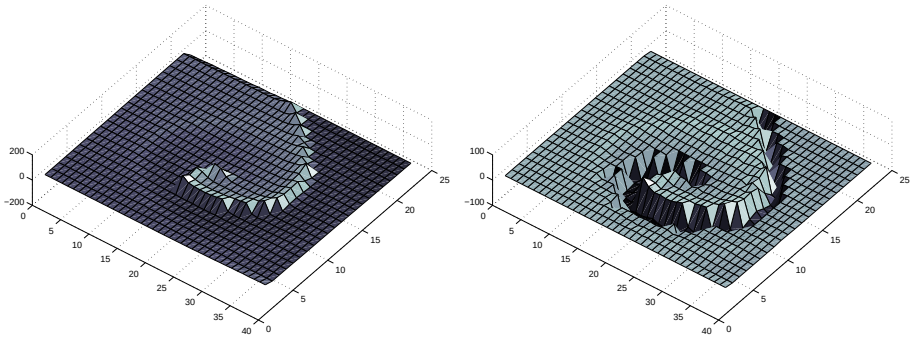


Figure 9.4. A graphical depiction over the coarsely discretized state-space of the relative performances of the robust and standard controllers. The left graph illustrates the degree to which the standard, certainty-equivalence based control underestimates the true value of each state, while the right graph illustrates the relative performance of the two controllers in simulations. Positive values indicated the robust solution out-performed the standard one.

in certain states, additional time-extended actions are available to the controller that simply perform the accelerate, decelerate, or do nothing for 2, 4, and 8 periods consecutively. These simply correspond to short, open-loop strategies the controller can choose to use.

Both methods are able to find satisficing controllers, however, the robust algorithm's value function serves as a lower bound on the actual cost when the controller is run on a simulation of the mountain car dynamics. The MDP solution often significantly underestimates the actual cost of a state. (Figure 2) The controller developed using the robust value iteration algorithm is quite conservative, preferring multiple swing-ups to ensure success. In a number of places it out-performs the standard controller (Figure 3).

It is interesting to observe that the sharp under-estimates of the true cost of a policy in the MDP do *not* vanish with increased resolution of the discretization. (See Figure 9.5.2). This is a very general phenomenon that occurs along switches in the optimal policy, so that given a δ-fine discretization, one can expect these kind of inaccurate value estimates to occur on $O(1/\delta)$ of the state space. MDP solutions are helped by noise in this case as it smoothes the dynamics, making the uncertainty in the transitions due to discretization relatively small.

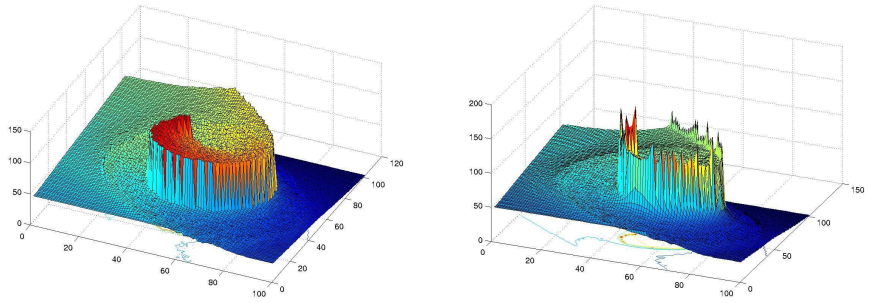


Figure 9.5. The left figure illustrates the mountain-car value-function computed by Algorithm (13) on a fine discretization of state-space. On the right the we see the simulated performance of the MDP solution. Note the difference in vertical scale.

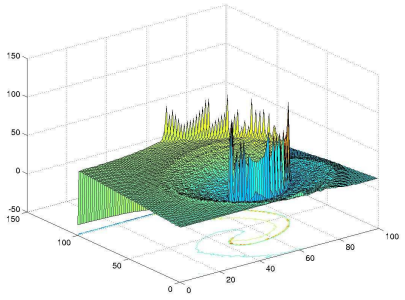


Figure 9.6. Error in the MDP cost estimate at a fine discretization.

9.6. Conclusions

9.6.1. Exploration

In recent unpublished work, [Strehl and Littman, 2004] have studied Model-based Interval Exploration (MBIE), a technique that takes an ℓ_1 type uncertainty bound derived by a PAC-type guarantee similiar to that given above. Interestingly, for that type of convex set, MBIE is essentially identical, for polytopically bounded uncertainty sets, to the framework we have presented except that the minimization of the disturber is replaced by a *maximization*. This assumption, that the decision maker is in the "best-possible" world, leads to a polytime reinforcement learning algorithm that combines exploration with exploitation like E^3 [Kearns and Singh, 1998]. It will be interesting to explore how the general framework presented here can be used to extend the MBIE approach.

9.6.2. Approximation

As pointed out in the introduction to this chapter, we have largely ignored the issue of approximation and partial observability in this chapter. Other work, for instance [Morimoto and Doya, 2001] has tried to develop approximate versions of robust control approaches including H_∞. This approach, besides suffering the standard drawbacks of Bellman equation methods, has a serious flaw in that when a policy is returned, it is very difficult to tell if the policy is particularly good for the controller or simply that the disturber's approximate policy is particularly bad. For general disturbances simply solving for the disturber's policy is at least as difficult as finding a good control, so this should come as no suprise.

We do, however, have some hope in the framework presented here. The Lagrange duality results from risk-sensitive control suggest a potential attack where the disturber's policy can be both efficiently computed and never explicitly represented. We leave serious effort in this direction to future work.

CHAPTER 10

Bayes Robustness

In the last chapter we considered an approach to robustness where our uncertainty was expressed in a set-based fashion. This approach had a number of nice properties including a fast (polytime) algorithm, the ability to handle undermodeling, performance guarrantees with a limited set of training data, and improved representational power with respect to H_∞ and stochatic control. Despite these advantages, the set-based approaches are a rather crude representation of uncertainty. If our concern is simply model uncertainty and not undermodeling, our admission of even a single difficult to control model, even if we believe it very unlikely, can lead to wildly conservative performance estimates and equally poor control strategies.

We know that probability theory provides a natural notion to degrees of belief [Jaynes, 2003, Cheeseman, 1985] and offers a more refined account of uncertainty than a set based one. In this chapter, we pursue a probabilistic account of uncertainty in modeling.

10.1. Embedding

A natural way to approach to modeling the uncertainty in the model is to augment the state-space by a uncontrolled and unobservable static state variable that represents the model. (See Figure 10.1) The same basic idea is used within modern control. It is not uncommon to see a Kalman filter used on an augmented state-space to simultaneously estimate both the state and the true model. [1]

[1] From a probabilistic standpoint, however, this is not well founded as the resulting dynamics are non-linear and the uncertainty non-Gaussian.

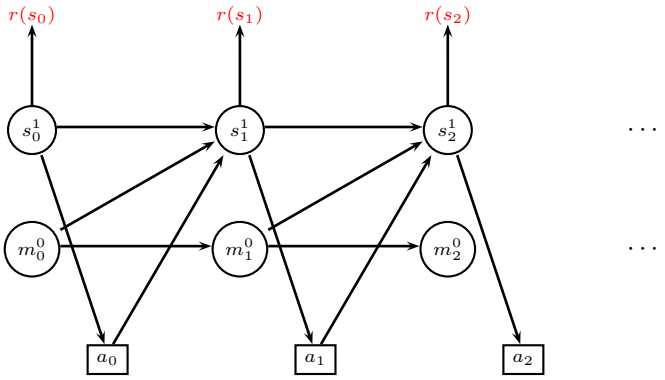


Figure 10.1. A graphical model depicting embedding model uncertainty as an uncontrolled and unobserved state variable in a POMDP.

Although in principle we can deal with uncertainty in modeling and noisy observations "optimally" by value-iteration of the resulting Partially-Observed Markov Decision Process, the computational complexity is overwhelming. The advantage of such an approach is that it would give us, in a meaningful sense, an *optimal* policy that could combine both robustness and exploratory actions. Unfortunately, such an approach remains wildly intractable even if we only have the static hidden state of the uncertain model.

Instead, we'll resume our progress along the vein of policy search taken in the first part of the thesis. We restrict the class of allowed controllers and we attempt to optimimze controllers that operate memorylessly on observables (as opposed to belief states). Policy search provides a clear method to build in our biases about good control strategies as well as taking advantage of domain knowledge. Control theorists, for instance, are often able to provide strong guidance on the structural properties of good controllers that we can build in to the optimization. Structured policies are very natural in the robotics field. It is natural to build restriction we would like on the controller directly into its structure and one can also easily limit the amount of computation required during the control cycle by suitably limiting the complexity of the controller structure. Finally, it is often the case that physical insight leads to good selections of controller class.

Further, limiting the complexity of the controller serves as a form of "regularization". Without structural guarantees, it could take an intractable number of Monte-Carlo rollouts of a policy on a simulation to evaluate its performance– there could always be some subtlety that is not apparent in any reasonable number of evaluations. This property is captured in theorems relating the uniform convergence of such estimates and the complexity of the searched policy class[Kearns *et al.*, 1999a].

10.1.1. Previous Work

In [Schneider, 1996], safety is addressed by treating learned model uncertainty as another source of noise to be incorporated into the stochastic transitions of an MDP. Good empirical results were obtained, but this method relies on an assumption that model error is uncorrelated through time and space, which is rarely the case. [Kearns and Singh, 1998] make exploration deliberative and guarantee near-optimal performance in polynomial time. Although this leads to nice theoretical results about the complexity of reinforcement learning, the aggressive exploration such an algorithm encourages is the antithesis of what we would hope for in building safe controllers. The literature on the exploration/exploitation problem in reinforcement learning is extensive. See [Sutton and Barto, 1998] for further discussion of the problem.

10.2. Preliminary Setup

For the purposes of this chapter, we will consider all off-line simulations to be on a deterministic simulative model [Ng and Jordan, 2000] where we can sample a typical event, $\omega \in \Omega$ under the distribution p (the joint distribution of initial states, models, and Markov noise in transitions and observations) and that each such ω can be stored and re-used for the evaluating different controllers. Deterministic simulative models are quite reasonable for model-based computations, but not so for model-free ones. It essentially amounts to being able to reset one's random number generator in a simulation to pick the same event for rolling out different policies in Monte-Carlo policy evaluation, and it provides a critical advantage in optimization, as it ensures that the reward criterion to be optimized will be a function (not noisy). This re-use of random numbers (the PEGASUS trick) is useful for reducing variance of estimates as well as allowing deterministic (as opposed to stochastic) optimization tools to be used. Also, the assumption of a deterministic simulative model provides complexity theoretic benefits in that it allows one to prove uniform convergence

of value-estimates to their means in time polynomial in the horizon length, even for infinite policy classes.

10.3. Optimal Policies

10.3.1. Performance Criterion

To formalize the notion of building optimal controllers we require a criterion on which to judge the performance of a given controller on a trajectory. In this chapter, we will consider the (discounted) sum of future rewards achieved under a controller. As before, we denote by $J_\pi(\xi)$ the empirical performance of a policy on a single trajectory:

$$J_\pi(\xi) = \frac{1}{1-\gamma} \sum_{t=0..T-1} \gamma^t R(\xi) \tag{10.1}$$

$\gamma \in (0, 1]$, $T = (0..\infty)$.

10.3.2. Bayesian Stationary Performance

To consider this as a metric on policies, we suggest that policies be ordered by mean trajectory performance, where the expectation is taken with respect to measure p (including Markov noise and *model distribution*). Note that the initial state, dynamic model, and effects of noise are all specified in the ω. Considering the expectation over model uncertainty and noise is a more complete way to look at model-based reinforcement learning solutions than is usually done when evaluating certainty-equivalence based approaches. We consider the entire posterior distribution on models, and not just the point maximum-likelihood estimate. Finding the optimal controller with this metric corresponds to the Bayesian decision-theoretic optimal controller, when we know the controller *cannot* be changed at a later time due to new information. Formally,

DEFINITION 10.3.1. *A policy π^* is ϵ near-optimal in Bayesian Stationary Performance if*

$$E^*_\pi[J^*_\pi(\omega)] \geq \sup_\Pi E_\pi[J_\pi(\omega)] - \epsilon \tag{10.2}$$

10.3.3. Guaranteed Performance

Another natural notion of good performance is that we achieve some level of performance with high probability. We call that notion *guaranteed performance*.

DEFINITION 10.3.2. *A policy π has α-fractile guaranteed performance r if $V_\pi = \sup_r P[J_\pi \leq r] \leq \alpha$. Policy π^* is $\epsilon - \alpha$ near optimal if $V_{\pi^*} \geq \sup_\pi V_\pi - \epsilon$.*

This definition captures the idea of a policy that we can rely on, despite both noise and model uncertainty, to perform up to some level.

10.3.4. Robust Expected Performance

In this notion of safe control, we are optimizing performance on almost all of the possible models, without trying to do well on all of them. Further, we separate safety with respect to model uncertainty from risk-sensitivity to Markov noise. We refer to the following definition as *robust expected performance criterion* to emphasize its similarity to metrics in robust control theory.

DEFINITION 10.3.3. *A policy π achieves α-robust performance r if*

$$P_q(m)[\mathrm{E}_{\mathrm{p}^\pi(\xi|\mathrm{m})}[\mathrm{R}(\xi)] \geq \mathrm{r}] \geq 1 - \alpha$$

That is, if the policy has high probability, under q, the distribution of models, of having large ($\geq r$) performance with respect to the noisy dynamics, then we say it has α-robust performance r. It is easy to provide a sampling algorithm that returns for any δ the $\alpha + \delta$ robust performance: First we sample n models m_i, and then given each model we sample k trajectories $\xi_{i,j}$ under that policy for each model. For each model we compute the empirical performance average:

$$\mathrm{E}[\hat{\mathrm{J}_\pi}|\mathrm{m_i}] = \frac{1}{n}\sum_{i=1}^{m} R(\xi_{i,j})$$

and then we return $\mathrm{E}[\hat{\mathrm{J}_\pi}|\mathrm{m_i}]$ for i chosen so that ceiling$(\alpha + \delta)m$ models have lower empirical reward. A standard Chernoff and union bound argument shows that this procedure will return the true $\epsilon + \delta$ performance of the policy with probability greater than $1 - \delta$ in time $O(\frac{1}{\delta})$.

10.3.5. Connections to robustness and exploration

In many applications it will be important to consider optimization criteria that more explicitly encourage robustness and exploration. Briefly, the central idea for safety and robustness criterion is to consider maximizing the performance on the worst model in a

large set of models, or on almost all trajectories the controller executes, so as to, with high-probability, bound the worse-case controller performance. Such robustness procedures when inverted to look at best, instead of worst, performance are similar to heuristic approaches commonly used in experiment design. (For a discussion of the application of stochastic optimization in artificial intelligence and a description of the algorithms mentioned here, see [Moore and Schneider, 1995].) Algorithms developing controllers to maximize this criterion can be seen as searching for a good experiment to perform to collect information; they are essentially designed according to the "optimism in the face of uncertainty" heuristic. Under this interpretation, the *Bayes optimal stationary controller* described here can be seen as being a version of PMAX– choosing an experiment at the point of largest expected value.

The other notions, like robust-expected performance, match closely with the ideas of *interval estimation* (IEMAX inverted) where we choose a policy because it has high performance with respect to the lower bound on some confidence interval. If, for example, we were to set $\epsilon = 0.95$ in ϵ-robust expected performance, we would obtain a controller resulting from an "optimism in the face of uncertainty" heuristic.

10.3.5.1. Convergence of Policy Value Estimates. We briefly note that the following theorem on the complexity of evaluating a policy class under the Bayesian Stationary Performance criterion follows immediately from [Ng and Jordan, 2000]:

THEOREM 10.3.1. *Let a discrete distribution of two-action POMDPs be given, and let* Π *be a class of strategies with Vapnik-Chervonenkis dimension* $d = VC(\Pi)$. *Also let any* $\epsilon, \delta > 0$ *be fixed, and let* $\hat{V}$ *be the policy estimates determined by a sampling algorithm using* m *samples from* Ω *(the same samples used to evaluate every policy) from scenarios where*

$$m = O\left(poly\left(d, \frac{R_{max}}{\epsilon}, \log\frac{1}{\delta}, \frac{1}{1-\gamma}\right)\right), \tag{10.3}$$

then with probability at least $1-\delta$, $\hat{V}$ *will be uniformly close to V (within* ϵ*) over all policies in that class.* ■

It is also important to note that the same arguments can be extended to encompass the robust-expected performance criterion. This type of result, while not typically leading to a useful number of samples to actually perform, is encouraging in terms of the tractability of the approach. It shows us that a reasonable (polynomial) number of random numbers

can be used to evaluate any policy at all in the class of policies we are considering. This result on the polynomial complexity of uniform bounds on the evaluation of performance criterion can be extended to the case of infinite action spaces (with suitable assumption on the complexity of the dynamics). See [Ng and Jordan, 2000] and [Kearns *et al.*, 1999a] for more discussion about policy search and the complexity of uniform bounds on evaluations.

10.3.6. Computational Complexity of Achieving Optimality

PROPOSITION 10.3.1. *Finding a deterministic stationary memoryless policy that achieves the largest expected reward on distributions over Markovian (or Partially Observed Markovian) Decision Processes is NP-hard.*

The distribution over models resulting from Bayes estimation in model-based RL leads to a difficult computational problem as we lose the Markov property that makes dynamic programming an efficient solution technique. The proof follows directly from Theorem 9.2.1.

10.3.7. Sampling Algorithms

Until this point we have deferred the question of sampling from the space of paths given a controller. In the case of Bayesian parametric approximators of system dynamics, sampling can be obtained simply by sampling from the posterior of the parameters and then rolling out trajectories as is standard in Monte-Carlo policy evaluation.

However, in many problems in robotics, it has been demonstrated that non-parametric regression techniques admirably serve to model the often highly non-linear and noisy dynamics[Atkeson, 1991]. These techniques make it impossible to directly sample from the space of possible models. Some non-parametric models like *Locally Weighted Bayesian Regression* do make it possible to sample from a set of posterior local parameters, and hence can generate samples from the 1-step predictive distribution due to model uncertainty. We argue that this, combined with the ability to re-estimate the model in the Bayes-optimal way, is sufficient to create arbitrary length trajectories that are independent samples from the n-step predictive distribution. If a regression algorithm like LWBR is **not** a Bayes optimal estimator, the technique described in this section provides biased n-step samples that we hope are close approximations to the ideal samples.

Algorithm 14: N-step predictive sampler

Algorithm to generate samples from the N-step predictive distribution of a learner with 1-step predictive distributions
Data: Given model and Bayes learner
repeat
  Generate a sample state transition from the 1-step predictive distribution and update the current state
  Update the learned model using the generated state transition as if it were a training point observed from the real system
until *a termination state is entered or effective horizon is reached (t in analysis below)*
Reset the learned model back to the original model

If our estimator were optimal in the Bayesian sense, we would expect that iteratively re-estimating the model using generated samples from the model, as the algorithm above suggests, would indeed allow us to sample from the n-step predictive distribution.

THEOREM 10.3.2 (Sufficiency of 1-step predictive learners). *If model M in algorithm (14) can be recursively updated in the Bayes-optimal way, the trajectories generated by the algorithm (14) are independent samples from the n-step predictive distribution.*

Proof: *We argue by induction. Consider the two step predictive distribution:*

$$p(s_2, s_1|s_0, \mathcal{T}) = p(s_2|s_1, s_0, \mathcal{T})p(s_1|s_0, \mathcal{T}) \tag{10.4}$$

where $\mathcal{T}$ is the observed data used to build the model. For a discrete model set,

$$\begin{aligned} p(s_2|s_1, s_0, \mathcal{T}) &= \sum_{\mathcal{M}} p(s_2|s_1, s_0, \mathcal{T}, M')p(M'|s_1, s_0, \mathcal{T}) \\ &= \sum_{\mathcal{M}} p(s_2|s_1, M')p(M'|\mathcal{T}') \end{aligned} \tag{10.5}$$

where $\mathcal{M}$ denotes the discrete class of models to be estimated from the data. The second distribution in each summation is just the posterior model M'; that is, the distribution over Markov models conditioned on the observed data and the transition from s_0 to s_1. But then the final equation shows that $p(s_2, s_1|s_0, \mathcal{T})$ is just another one-step distribution from the new distribution of models $P(M')$, simply the learned model under the old data and the new observed transition.

Similar results can be shown with more technical detail in the case of other model distributions. It follows then from the law of composition that if s_1 is first drawn i.i.d $p(s_1|s_0, \mathcal{T})$ and then s_2 is drawn from $p(s_2|s_1, s_0, \mathcal{T})$, the pair is i.i.d from the joint predictive distribution.

10.4. Learning algorithms

One important advantage of embedding the robustness problem as a kind of POMDP (with a potentially non-standard cost function) is that the full suite of algorithms we have developed in this work can be applied to the problem of computing good controllers. In the case study section of this work, we show that even crude Monte-Carlo optimization using a non-differentiable optimization technique like the Nelder-Mead downhill simplex [Press *et al.*, 1992] can perform quite well, especially with a simple policy class.

10.4.1. Robust Monte-Carlo PSDP

PSDP generally can be applied to the problem of finding policies by simply treating the model as yet another piece of state. We need only specify a *conditional* state-time distribution $\mu(s|m)$ for a good policy and run the algorithm directly. It is worth pointing out that there is a particularly efficient implementation of this approach if we have a reasonably-sized discrete set of states.

In particular, the approach is to draw a number of sample models m_i from our distribution over models. We then start with a uniform distribution over states for each model $p_t(s|m_i) \propto 1$. At each backup step of PSDP we compute $Q_t(s, a|m_i)$. This can be done efficiently as it is simply linear in $V_{t+1}(\cdot|m_i)$, the value of our future policies under that model. We then perform the optimization step of PSDP by for each state maximizing $\sum_i p_t(s|m_i)Q_t(s, a|m_i)$ over the actions a. We also cache the $V_{t+1}(\cdot|m_i)$ for the next iteration backwards. Once we have a policy, we can easily recompute $p_t(s|m_i)$ for the new policy, and iterate PSDP. Each loop of the algorithm we describe is *linear* in time horizon T, like exact ITERATED PSDP, but we can sample over a very large model set.

In fact, the number of samples we must use is completely independent of the number of models we are considering. Rather it depends only on the size of the state-action space, as this is what controls the size of our effective policy space. In this sense, we get "robustness" almost for free– at simply a constant cost more than a direct policy iteration approach would generate a policy. Dan Bohus [Bohus, 2004a, Bohus, 2004b] is currently pursuing implementations of Robust PSDP for large scale automated dialog management with multiple, uncertain dialog interaction models.

Figure 10.2. The CMU Yamaha R50 helicopter in autonomous flight.

10.4.2. Analysis

The Policy Search by Dynamic Programming approach is applicable very simply to the robustness problem. However, it takes rather limited advantage of the fact that the unobservable aspects of state may also be the only uncontrollable ones. We are not aware of much more leverage that can be obtained here, except for noting that in some problems $p_t(s|m)$ can be largely independent of the model m for a certain, good class of policies. Consider for instance the problem of helicopter control, which we elaborate on in the next section. A good policy should spend time hovering in approximately the same distribution of states independent of which specific model is instantiated with high probability over the set of models. In cases that we can hope this is true, we can simply specify a $\mu(s)$ in PSDP that ignores the model aspect entirely, and compute a good policy with respect to that.

10.5. Case Study: Helicopter Control

Robotic control problems generally are an area in which there is ample room to apply the techniques developed in machine learning. This is particularly true in the area of control of autonomous helicopters, where most current control schemes are based on engineer- and man-hour intensive hand tuning of Proportional Integral Derivative (PID) based controllers. Autonomous helicopter control is difficult as the dynamics are unstable, non-minimum phase, have large delays, and vary a great deal across the flight envelope. In this section we detail some of the results from applying the policy search methods described in the previous sections to the problem of the flight control of an autonomous helicopter.

10.5.1. Dynamics

We begin by specifying the problem. To provide a manageable goal in applying policy-search to the helicopter, we considered only the so-called "core dynamics" of the helicopter, the pitch, roll, and horizontal translations. The dynamic instabilities are known to lie in these dynamics, and control of these is therefore paramount[Mettler *et al.*, 1999]. Existing proportional-derivative (PD) controllers, tediously tuned by the helicopter team, were used on the yaw-heave dynamics. From a high-level, the goal will be the regulation of the helicopter hovering about a point, or a slowly varying trajectory. This will be formalized as a cost function to be optimized.

10.5.2. Modeling

Modeling a dynamical system is always challenging. To learn the dynamics of the helicopter, we chose to implement a LWBR state-space model of the following form (a locally affine model):

$$z\overrightarrow{x} = \mathbf{A}(\overrightarrow{x})\overrightarrow{x} + \overrightarrow{ref}(\overrightarrow{x}) + \mathbf{B}(\overrightarrow{x}) \begin{bmatrix} \delta_{lon} \\ \delta_{lat} \end{bmatrix} \tag{10.6}$$

$$\overrightarrow{x} = [x, v_x, \theta, \theta - \frac{1}{z}\theta, y, v_y, \phi, \phi - \frac{1}{z}\phi]^T$$

where z is the forward-shift operator. The inputs, δ_{lon} and δ_{lat} reflect the cyclic controls of the helicopter. The state variables x and y refer to translational deviation of the helicopter from its set point. Massive cross-validation was applied to determine appropriate kernel widths. Data was collected from pilot tele-operation of the helicopter. This data was recorded off the Kalman state-estimator at 100Hz and down-sampled to 10Hz. The down-sampling introduces aliasing into the data due to the higher-order dynamics of the helicopter, but has the advantage that it reduces the apparent delay in controls applied caused by the unobservable rotor and actuator dynamics. We hope to still capture much of the principal behavior of the helicopter with the lower frequency model. Interesting future work would involve building a different state-space model capable of capturing the higher frequency dynamics the helicopter demonstrates.

10.5.3. Controller design

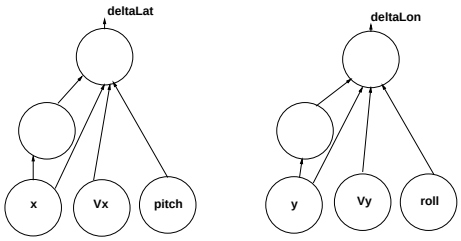


Figure 10.3. A graphical depiction as a neural network of the structure of the policy used to control the helicopter.

10.5.3.1. Controller structure. In proposing an initial controller structure, we looked towards simple controllers known to be capable of flying the helicopter. To this end, we proposed a simple, neural-network style structure (see Figure (10.3)) that is decoupled in the pitch and roll axis, and about equilibrium is similiar to a linear PD controller. There were 10 parameters to modify in the controller structure, corresponding to the weights between output nodes and parameters in the sigmoidal functions at the hidden and output layers. This is a fairly simple controller that leads to a policy differentiable in its parameters and nearly linear about equilibrium. Because of the hidden layer unit, it is able to adapt to large set-point shifts in the position variables, unlike a linear one.

10.5.3.2. Optimization. For the purposes of optimization, we maximized the Bayesian Stationary Performance criterion. It has previously been demonstrated that this criterion (or rather the approximation of it given in [Schneider, 1996]) typically leads to controllers that are neither too conservative, nor as aggressive as that obtained by a optimizing a maximum likelihood model. A variety of cost criteria were implemented, each leading to different (although mildly so) controllers. A typical example was the quadratic form:

$$x^2 + y^2 + \dot{x}^2 + \dot{y}^2 + .0001 * \delta_{lat}^2 + .0001 * \delta_{lon}^2 \tag{10.7}$$

Quadratic forms in the position variables are typically not reflective of our real performance goals. A better reflection of our preferences in control is one that emphasizes stability in velocities and angles rather than placing huge penalties on modest deviation from the nominal set point– especially as we will expect the controller to perform well even when started quite far (as distant as 25 meters) from the set point. To reflect this preference, our immediate cost criterion on the position variables is computed linear in the magnitude of the state variable, or of a form like the following:

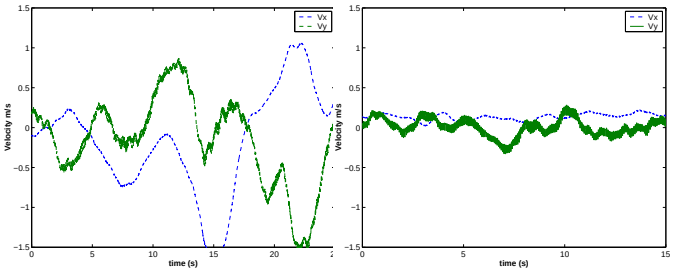


Figure 10.4. Data logs from the R-50 demonstrating performance hovering under (left) a highly trained pilot's control, and (right) the neural-net controller built by the robust policy search methods.

$$10\frac{x^2}{x^2+1} + 10\frac{y^2}{y^2+1} \tag{10.8}$$

Finally, we assigned a large penalty (10^6) on trajectories for which state-variables left the bounding box of the data originally observed.

After establishing the cost criterion, we considered the task of optimizing the parameters. Trajectories were rolled out using the sampling technique described in algorithm (14) for the LWBR model. Typical policy evaluations were 30 trajectories of horizon length 500 with discount factor $\gamma = .995$. The amoeba (simplex) optimization technique [Moore and Schneider, 1995] was used to modify the controller parameters and guide the search for optima. Random restarts were applied to initial weights to allow the optimizer to find a reasonable solution. Note that it is possible to use differentiable optimization techniques instead, if we were willing to smooth the "out-of-bounds" penalties introduced on the state-variables.

10.5.4. Validation

Initial validation experiments were performed on a linear model of the rotorcraft about hover given in [Mettler *et al.*, 1999]. Good performance on this model was encouraging as it is significantly higher-dimensional (14^{th} order) and larger bandwidth model than that obtained using the locally weighted regression technique described here, and was developed by a different set of techniques. In particular, to formulate a state-space model [Mettler *et al.*, 1999] uses the U.S. Army developed CIFER system, designed specifically for full-scale rotorcraft identification. CIFER performs system identification in the frequency domain using the Chirp-Z transform and treats the state space identification as an optimal matching problem. This approach has been validated numerous times and takes advantage

of the known physics of the helicopter. Further, because of the state space structure, CIFER explicitly models rotor dynamics (which introduce a significant delay into the system dynamics). This modeling enables Mettler et. al. to capture higher frequency dynamics than we can hope to. However, the frequency domain approach is fundamentally a linear one, and thus one can only capture dynamics in a small part of the flight envelope. Further, the inflexibility of the state-space model in CIFER forces the response into a particular structure that one cannot be sure is accurate for all rotorcraft, particularly ones of dramatically different scales then that for which CIFER was originally designed. Despite these reservations, it is apparent that [Mettler *et al.*, 1999] get excellent results for the hover centered model.

It is interesting to note that the controller developed by policy search on the maximum likelihood model had highly oscillatory (nearly unstable) performance on the linear simulator. The controller learned on the distribution of models, in contrast had significantly lower loop gain.

After a degree of confidence was achieved by simulation on the model, the controller was ported to Carnegie-Mellon's autonomous R-50 helicopter control computer. The estimation-control loop on-board operates at 100Hz (as opposed to the simulation 10Hz). To ensure the controller wasn't operating outside the modeled bandwidth, a first order low-pass digital filter was implemented on the control outputs.

The helicopter was then test flown. The results were encouraging, and demonstrate that the simple policy search technique can generate controllers that are applicable to robotic systems. Despite quite windy conditions, the rotorcraft was able to track moving set points and reject strong gusts. Figure (10.4) shows typical performance during the flight, contrasting the hovering of a highly trained human pilot with the controller obtained using the safe learning control methods described above.

10.5.5. Importance of Solution Elements

Any physical implementation like the helicopter control problem requires a fairly complex interaction of modeling and algorithmic components. Here we examine elements of that solution present to determine the relative importance of each.

10.5.5.1. Controller Structure. The controller we present is nearly linear and factored along the x, y axes. It is close to as simple as we can reasonably expect to work. The neural network non-linearity is quite useful in that it saturates large offsets so that the controller is able to function with even very large deviations from its setpoint. This is useful for trajectory following in particular.

10.5.5.2. Locally-weighted modeling. In this work we used LWBR as the technique to model the state-space dynamics of the helicopter. LWBR is a relatively sophisticated regression technology, and one naturally wonders whether a simpler (say linear regression) technique would suffice. A number of pieces of evidence suggest otherwise. For instance, our results using cross-validation strongly suggested that a reasonably small kernel width was appropriate for the problem.

This non-linearity is reflected rather dramatically in the predicted behavior of the helicopter. Using LWBR, we were able to calculate fixed-point controls for the helicopter at various forward velocities in the range of the data. For each forward velocity, and its fixed-point controls, one can compute the linear dynamics about that point using LWBR. Further, one can compute the eigenstructure of those dynamics. The *center-manifold theorem* [Khalil, 2001] tells us that around a fixed-point the stability of a system can be assesed by with the same criteria as a linear system– i.e. an eigenvalue in the right hand complex plane (for continuous-time) systems indicate the system is unstable. If all poles have their real components strictly less than 0, however, the fixed-point is stable. For the helicopter system, the roll-pitch dynamics radically change character from unstable to stable as the forward velocity increases. This is not an unexpected result, as a helicopter in rapid forward motion begins to aero-dynamically resemble a fixed-wing aircraft. It does, however, strongly indicate that the dynamics are rather unavoidably non-linear.

Further, although control policies have been built by hand for a few autonomous helicopter projects, the use of modern, automated control techinques has proved to be difficult. Anecdotal evidence [Ng, 2002a] [Amidi, 2002] suggests that despite using relatively sophisticated robust control techniques that are very popular in the aerospace community, it proved very difficulty to automatically build controllers for the helicopters. It is also difficult, of course, to make an educated guess about the cause of these failures, but at least one clear issue is that the modern robust control techniques described are invariably linear control techniques. They may simply be operating outside their domain of applicability.

10.5.5.3. PEGASUS. We described in this work the use of the PEGASUS trick to optimize the helicopter controller. In our experiments, we found it useful, but not essential. The same optimization and effectively the same controller was also found using "fresh" random samples, although we must be somewhat more careful about the optimization, and the time required to find a good solution rises by a factor of approximately 2-5 times.

10.5.5.4. Robustness. We introduced a series of novel robustness approaches in this chapter and in [Bagnell and Schneider, 2001] and demonstrated that they worked in practice. A few variants were tried as well. The certainty equivalent approach was *not* tried on the physical helicopter due to its aggressive behavior and poor performance on the CIPHER-based simulation. The Bayes expected performance controller was the most effective controller in practice. Two other controllers based on various ϵ values (.05 and .1) were tried for robust expected performance. These controllers successfully flew the aircraft, but were unable to reject the large wind disturbances that the craft experienced. As a result it would typically hover off the goal position.

Recent research [Ng *et al.*, 2003] has extended the work presented in this chapter to include the addition of special tunings for trajectories used in model helicopter competitions. In this work, robustness was only maintained by manual validation of the model. This suggests, as we would hope, that with enough data, the Bayes robustness technique becomes unnecessary.

10.5.5.5. Performance criteria. The behavior of the helicopter was very robust with respect to changes in the performance criteria, as long as structurally it roughly penalized small deviations and didn't become too wild at large deviations.

10.6. Conclusions

Our current work establishes a framework for the development of structured controllers sensitive to model uncertainty and then demonstrates the viability of the approach on a difficult physical control problem. In contrast with the previous chapter, we are able to optimize our performance under state uncertainty and using a limited notion of optimality (i.e. in a policy class).

From an experimental viewpoint, future research directions include more sophisticated control of the rotorcraft to exercise more of the flight envelope. Another interesting

area to pursue is the online implementation of the inner-loop policy search to dynamically reconfigure the controller. This is particularly interesting in the context of error-recovery. In the event of a failure of some sub-system of the helicopter, it will be critical, based only on very limited experience after the fault, to quickly search for a viable controller.

From a theoretical perspective, it is interesting that there seems to be only limited leverage one can gain over a standard POMDP (where state is affected by action). It would be interesting to see if there is any more restricted class, like factored-convex in the last chapter, that eases the computational burden.

CHAPTER 11

Conclusions and Discussion

The central thesis of this work is that state uncertainty, approximation, and robustness stand as core issues to be tackled in developing the technology to learn decision making strategies. In this work, we have presented and summarized progress in these directions. It is worth reviewing the applications and strengths of the algorithms presented in a broader framework.

11.1. Approaches to Learning Policies

If we paint in very broad strokes, we can identify a number of essentially different approaches to computing policies in POMDPs.

11.1.1. Crude Monte Carlo

Perhaps the simplest and most natural one is crude Monte-Carlo policy search. Essentially we take some number of policies, say a finite n, for the sake of this discussion, and compute the optimal values of each using independent Monte-Carlo trajectory rollouts. It is clear from a Chernoff and union bound argument that we can tell with high probability a policy that is near optimal (additively or multiplicatively). It is clear that we can do this for roughly $O(n)$ computation.

This is a simple and sometimes rather effective approach. We have essentially reduced the policy learning problem to a n-armed bandit problem. A wide variety of tricks can be used to make it practical: learning and regularization between related arms' value estimates, Hoeffding races [Maron and Moore, 1993] to eliminate non-contenders quickly,

Interval Estimation, and the Pegasus trick of re-using random numbers can reduce variance. The sample-based policy gradient methods are, in a sense, just cleverer versions of crude Monte-Carlo where we take advantage of local optimization techniques. Using the Pegasus trick allows us to consider infinite policy spaces rigorously as long as they have finite combinatorial complexity (say VC-dimension).

Using local optimization techniques (like policy gradient) combined with this kind of crude Monte-Carlo has proved empirically to be a surprisingly strong contender. The helicopter work given in this thesis, as well as more recent experimental results due to [Ng *et al.*, 2003], unpublished work on open-loop four legged walking [Ng, 2002b], Tetris [Kakade, 2002], and a body of experimental work due to Jonathan Baxter and collaborators [Baxter *et al.*, 1999b] all give credence to its broader scale use.

Typically, however, we wish to consider very large policy spaces. To do this effectively, we'd like to re-use information from one policy's experiments to evaluate other policies.

11.1.2. Model Predictive Control

Another approach to solving the problem is known in the control community as Model Predictive Control. Essentially, there is little off-line component to learning at all. Instead, at each time step in the control loop, we roll out trees consisting of the possible futures of the process, including possible future actions. Then we choose the root control that leads to, on sample average, the best future expect reward. That this is a well-founded practice for stochastic control problems as well as deterministic ones is due to the "sparse sampling" arguments in [Kearns *et al.*, 1999b].

That version requires full state observability to compute the rollouts. Other variants, including the Trajectory Tree method [Kearns *et al.*, 1999a] and [Ng and Jordan, 2000] [1] allow us to build policies off-line from some policy class using these same kind of rollout trees. The power of these tree rollout methods is that we can evaluate a large number of policies using the same trees. Formally, we need approximately $O(\log n)$ trees for n policies. Unfortunately, although we can compare a great many policies, the trees are of exponential size in the horizon length T. Further, we can still pay n computational cost to search through the policies.

[1] One natural view of PEGASUS is as an implicitly represented Trajectory Tree method.

This is an interesting tradeoff, and quite useful in practice. Model predictive control is widely used in the chemical industry, as well as within robotics. For instance, the well known systems RANGER and MORPHIN [Urmson *et al.*, 2003] use exactly this kind of technique. We can also interpret kino-dynamic planning techniques like Rapidly-exploring Random Trees (RRTs) in this light [LaValle and Kuffner, 2000]. These techniques are essential control tools for roboticists.

11.1.3. Bellman Methods

If we are making the Markov assumption (i.e. perfect observability), Bellman-equation methods (Value and Policy Iteration, Linear Programming) offer excellent scaling. For example, we get poly-logarithmic complexity in the number of policies, even if we do not know the model at all to begin with, as algorithms like E^3 provide provably approximately optimal efficient learners.

Bellman methods, of course, deal poorly with all the issues we viewed as important within this thesis. The behavior of Bellman methods degrades rapidly as we violate the Markov assumption or attempt to limit the policies we consider (e.g. approximation). In the work on *stochastic robustness* presented in this thesis, we were able to introduce a novel technique that generalizes Bellman methods to mitigate a form of non-Markovian behavior, as well as provide robustness to both under-modeling and unknown dynamics. It would be interesting to fit this kind of advance into the framework of the other general approaches discussed in this chapter.

Linear programming and certain weak forms of approximation [Guestrin *et al.*, 2002a] show potentially better behavior. An interesting aspect of the former methods, however, is that they depend rather strongly [de Farias and Roy, 2003] on a weighting scheme that plays essentially exactly the same role as our baseline distribution μ_t in Policy Search by Dynamic Programming. However, these methods only work for an explicitly represented set of dynamics (i.e. no generative or trace type sampling models), do not clearly extend to partial observability, and only compute a upper-bound value function, which may not be nearly as good as directly approximating a policy.

11.1.4. Policy Search by Dynamic Programming

The algorithm PSDP presented in this work offers a different trade-off. We can hope to efficiently search very large classes of policies and get poly-logarithmic performance in terms of sample (and even computational for some policy classes) complexity by relying on a powerful piece of domain knowledge: where good policies spend time, represented as a state-time distribution. The basic idea here was to use this baseline distribution to recursively compute a sequence of policies. Each policy optimization is naturally viewed as a classification (supervised learning) task, for which one typically is able to search for some approximation of optimal very efficiently.

Even in the case that we do not have a strong indication of where good policies spend time, PSDP is a natural heuristic where we iteratively compute policies that are optimal with respect to a current distribution, and then re-compute that distribution based on samples from the policy. This has much the flavor of the gradient methods (that is acts a kind of local search), but can take potentially larger steps. We typically stop such a process when we reach a point where the distribution induced and the policies are matched, which looks like a zero-gradient point in the space of stochastic time-varying policies. Convergence isn't guaranteed, unless we can do the optimization steps exactly (as in the analytic version), although it is very easy to check as we modify each time-slice policy whether that modification is improving the policy overall. This gives another natural stopping criterion.

11.1.5. Future Approaches

The broad discussion of these policies motivates a number of areas of future research. Each approach makes some major (and essentially unavoidable [Kakade, 2003]) trade-off to solve the problem– requiring exponential resources or using some important piece of domain knowledge. These limitations are essentially different, and one would hope that hybrid algorithms could mitigate the down-sides of each.

For instance, policy search methods that can include some kind of model-predictive (Predictive Policy Search?) aspects could be very powerful. This kind of approach has precedent among control practitioners (at least for deterministic systems) who often use "feed-forward control" that computes likely future points along a path to servo around.

This technique is powerful and used almost everywhere that requires serious control technology. Methods that take advantage of Bellman-equations to guide search or act as features in a policy, although not relying directly on the value-function for a policy, also have much potential.

11.2. Algorithm Summaries

In the following tables, we try to summarize the usefulness of certain algorithm types that we have discussed in this thesis. In the first we summarize the potential policies each algorithm can generate:

Method	Continuous Actions	Non-parametric	Partially Observable
Reinforce	●	○	●
Kernel Reinforce	●	●	●
Covariant Reinforce	●	●	●
Derivative Propagation	●	○	●
PSDP	●	●	●
Sparse Sampling	○	●	○
App. Policy Iteration	○	●	○

In the next, we outline the kind of performance guarantees one can hope for in the algorithm. We use a filled circle to indicate performance is good (i.e. poly-time computation).

Method	Computation	Convergence	Optimality	Access Model
Reinforce	○	●	Local	Trace
Kernel Reinforce	○	●	Local	Trace
Covariant Reinforce	○	●	Local	Trace
Derivative Propagation	●	●	Local	Complete
PSDP	●	●	Global[2]	Generative
Sparse Sampling	● [3]	●	Global	Generative
App. Policy Iteration	●	○	None	Reset

11.3. Robustness

In this thesis we also considered the issue of robustness, where we must contend with both our uncertainty about models, and our inability to accurately capture true system dynamics in the space of models. We presented two basic approaches. In stochastic robustness, we took a set-based approach to our uncertainty, and computed policies efficiently for a powerful class of such uncertainties that lies at the intersection of undermodeling

[3] Global given good μ, local otherwise.
[3] Well defined, but exponential in horizon.

and uncertain dynamics. In Bayesian robustness we embed our model uncertainty as partially observed state, essentially encoding a probability distribution over possible models. This allows us to leverage all the algorithms we developed in the first part of the thesis, but unfortunately, we cannot compute policies efficiently generally, even with full state observability. The following table captures the basic axes upon which the methods vary. We hope to, in future work, extend stochastic robustness to fill in the circles for both approximation and partial observability. Again, a full circle means good behavior (handles the situation), and an empty circle means does not:

Method	Undermodeling	Uncertain Dynamics	Computation	Approx.	POMDP
Stochastic	●	●	●	○	●[4]
Bayesian	○	●	○	●	●

11.4. Future Directions

Other areas of important future work include the development of techniques that are especially appropriate for belief-state features. In this work, although we used filters at times (Chapter 10), the primary focus was on the more essential problem of computing policies given only observables. It was argued in the introduction that this is critical, as we are always forced into the situation by our limited model access and the need to approximate. There is some interesting structure to belief states even when approximate, however, as features to our algorithms, and it is possible that additional progress can be made in that direction.

Other important future work that is hinted at in chapter 6 is the development of structured policies for multiple agents. Policy structuring that is appropriate given information and communication constraints, as well as alogorithms that can appropriately optimize all will most likely require some extension of our approaches. In a related vein, it is often appropriate to view the communication problem within distributed control systems as a type of POMDP[Tatikonda, 2001]. Applying our algorithms in this context could make very interesting future work.

[4] Only for weakly non-Markovian processes

REFERENCES

[Amaldi and Kann, 1998] Edoardo Amaldi and Viggo Kann. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, 209(1–2):237–260, 1998.

[Amari and Nagaoka, 2000] S. Amari and H. Nagaoka. *Methods of Information Geometry*. Oxford University Press, 2000.

[Amidi, 2002] O. Amidi. Personal communications, 2002. Regarding robust control techniques on automous helicopter.

[Arimoto *et al.*, 1984] S. Arimoto, S. Kawamura, and F. Miyazaki. Bettering operation of robots by learning. *Journal of Robotic Systems*, 1(2), 1984.

[Atkeson and Morimoto, 2003] C. Atkeson and J. Morimoto. Non-parametric representation of a policies and value functions: A trajectory based approach. In *In Neural Information Processing Systems 15*, 2003.

[Atkeson, 1991] C. Atkeson. Using locally weighted regression for robot learning. In *Proceedings of the 91 IEEE Int. Conference on Robotics and Automation*, April 1991.

[Bagnell and Schneider, 2001] J. Bagnell and J. Schneider. Autonomous helcopter control by policy-search based reinforcement learning. In *Proceedings of the 2001 IEEE Int. Conference on Robotics and Automation*. IEEE, 2001.

[Bagnell and Schneider, 2003] J. Bagnell and J. Schneider. Covariant policy search. In *International Joint Conference on Artificial Intelligence*, 2003.

[Bagnell *et al.*, 2003a] J. Bagnell, S. Kakade, A. Ng, and J. Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems*. MIT Press, 2003.

[Bagnell *et al.*, 2003b] James Bagnell, Sham Kakade, Andrew Ng, and Jeff Schneider. Policy search by dynamic programming. In *Neural Information Processing Systems*, volume 16. MIT Press, December 2003.

[Baird and Moore, 1998] L. Baird and A. Moore. Gradient descent for general reinforcement learning. In *Neural Information Processing Systems 11*, 1998.

[Basar and Olsder, 1995] T. Basar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. SIAM, 1995.

[Baxter *et al.*, 1999a] J. Baxter, L. Weaver, and P. Bartlett. Direct-gradient-based reinforcement learning I: Gradient estimation algorithms. Technical report, Computer Sciences Lab, Australian National University, 1999.

[Baxter *et al.*, 1999b] J. Baxter, L. Weaver, and P. Bartlett. Direct-gradient-based reinforcement learning ii: Gradient ascent algorithms and experiments. Technical report, Computer Sciences Lab, Australian National University, 1999.

[Bellman, 1957] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Bertsekas and Tsitsiklis, 1996] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[Bohus, 2004a] D. Bohus. Error awareness and recovery in task-oriented spoken dialogue systems, 2004. Ph.D. Thesis Proposal, CMU.

[Bohus, 2004b] D. Bohus. Personal communication, 2004.

[Boutilier *et al.*, 1995] Craig Boutilier, Richard Dearden, and Moise's Goldszmidt. Exploiting structure in policy construction. In Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, San Francisco, 1995. Morgan Kaufmann.

[Boyd and Vandenberghe, 2004] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Burgard *et al.*, 1999] W. Burgard, A.B. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. In *To appear in Artificial Intelligence*, 1999.

[Cheeseman, 1985] Peter Cheeseman. In defense of probability. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 1002–1009, Los Angeles, California, 1985. Morgan Kaufmann.

[Chentsov, 1980] N.N. Chentsov. *Statistical Decision Rules and Optimal Inference*. American Mathematical Society, 1980.

[Cover and Thomas, 1997] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1997.

[de Farias and Roy, 2003] D.P. de Farias and B. Van Roy. Approximate linear programming for average-cost dynamic programming. In *Advances in Neural Information Processing Systems 15*, 2003.

[DeGroot, 1970] M. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, 1970.

[Della Pietra *et al.*, 1997] Stephen Della Pietra, Vincent J. Della Pietra, and John D. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[Fleming and Hernandez-Hernandez, 1997] W. H. Fleming and D. Hernandez-Hernandez. Risk-sensitive control of finite state machines on an infinte horizon i. *SIAM Journal of Control and Optimization*, 35, 1997.

[Glynn, 1986] P. W. Glynn. Stochastic approximation for monte–carlo optimization. In *In Proceedings of the 1986 Winter Simulation Conference*, 1986.

[Gordon, 1995] G. Gordon. Stable function approximation in dynamic programming. In *The 12th International Conference on Machine Learning*, 1995.

[Gray and Moore, 2001] Alexander Gray and Andrew Moore. N-body problems in statistical learning. In Todd K. Leen and Thomas G. Dietterich, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2001.

[Guestrin *et al.*, 2002a] C. Guestrin, D. Koller, and R. Parr. Multi-agent planning with factored MDPs. In *Advances in Neural Information Processing Systems (NIPS-14)*, 2002.

[Guestrin *et al.*, 2002b] C. Guestrin, M. Lougadakis, and R. Parr. Coordinated reinforcement learning. In *International Conference on Machine Learning*, 2002.

[Guestrin, 2003] Carlos Guestrin. *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Stanford School of Computer Science, 2003.

[Heger, 1994] M. Heger. Consideration of risk in reinforcement learning. In *International Conference on Machine Learing*, 1994.

[Jaynes, 2003] E.T. Jaynes. *Probability: The Logic of Science*. Cambridge University Press, 2003.

[Jordan and Bishop,] M. Jordan and C. Bishop. *Graphical Models*. In Preparation.

[Jordan and Bishop, 2004] M. Jordan and C. Bishop. *Graphical Models*. In preparation, 2004.

[Kakade, 2002] S. Kakade. A natural policy gradient. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[Kakade, 2003] Sham Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, 2003.

[Kearns and Singh, 1998] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *International Conference on Machine Learing*, 1998.

[Kearns *et al.*, 1999a] M. Kearns, Y. Mansour, and A. Ng. Approximate planning in large POMDPs via reusable trajectories. In *Neural Information Processing Systems 12*, 1999.

[Kearns *et al.*, 1999b] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999.

[Khalil, 2001] H. K. Khalil. *Nonlinear Sytems*. Prentice Hall, 2001.

[Kivinen *et al.*, 2002] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[Konda and Tsitsiklis, 2002] V. Konda and J. Tsitsiklis. Actor-critic algorithms. *to appear in the SIAM Journal on Control and Optimization*, 2002.

[Lagoudakis and Parr, 2003] Michail G. Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of ICML-2003*, August 2003.

[Langford and Kakade, 2002] J. Langford and S. Kakade. Approximately optimal approximate reinforcement learning. In *Proceedings of ICML*, 2002.

[Langford and Zadrozny, 2004] J. Langford and B. Zadrozny. Reducing t-step reinforcement learning to classification, 2004.

[Langford, 2004] J. Langford. Personal communications, 2004. information mismatches in Bellman methods and Bayesian optimality.

[LaValle and Kuffner, 2000] S. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects, 2000. In Workshop on the Algorithmic Foundations of Robotics.

[Littman and Szepesvri, 1996] Michael Littman and Csaba Szepesvri. A generalized reinforcement-learning model: Convergence and applications. 1996.

[Littman, 1994] M. Littman. Memoryless policies: Theoretical limitations and practical results. In *From Animal to Animats 3: Proceedings of the 3rd International Conference on Simulation and Adaptive Behavior*, 1994.

[Maron and Moore, 1993] O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems (NIPS-6)*, 1993.

[Mason *et al.*, 1999] L. Mason, J.Baxter, P. Bartlett, and M. Frean. *Functional gradient techniques for combining hypotheses*. MIT Press, 1999.

[Maybeck, 1986] P. Maybeck. *Stochastic Models, Estimation and Control*, volume 1. Academic Press, 1986.

[Mettler *et al.*, 1999] B. Mettler, M. Tischler, and T. Kanade. System identification of small-size unmanned helicopter dynamics. In *Presented at the American Helicopter Society's* 55^{th} *Forum*, 1999.

[Meuleau *et al.*, 2001] N. Meuleau, L. Peshkin, and Kee-Eung Kim. Exploration in gradient-based reinforcement learning. Technical Report 03, Massachusetts Institite of Techology AI Lab, April 2001.

[Minka, 2001] T. Minka. *A Family of Algorithms for Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology, June 2001.

[Montemerlo and Thrun, 2001] M. Montemerlo and S. Thrun. Personal communications, 2001. regarding obstacle tracking in the Nursebot project.

[Montemerlo *et al.*, 2003] Michael Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 3, pages 2436–2441, Las Vegas, NV, October 2003.

[Moore and Schneider, 1995] A. Moore and J. Schneider. Memory based stochastic optimization. In *Advances in Neural Information Processing Systems (NIPS-8)*, 1995.

[Moore, 1990] A. Moore. *Efficient Memory-Based Learning for Robot Control*. PhD thesis, University of Cambridge, November 1990.

[Morimoto and Doya, 2001] Jun Morimoto and Kenji Doya. Robust reinforcement learning. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 1061–1067. MIT Press, 2001.

[Murphy, 2000] K. Murphy. A survey of POMDP solution techniques. Technical report, Berkeley Computer Science, 2000.

[Murphy, 2002] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California at Berkeley, 2002.

[Ng and Jordan, 2000] A. Ng and M. Jordan. Pegasus: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence, Proceedings of the Sixteenth Conference*, 2000.

[Ng *et al.*, 1999] A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[Ng *et al.*, 2003] A.Y. Ng, H.J. Kim, M. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *Neural Information Processing Systems*, volume 16. MIT Press, December 2003.

[Ng, 2002a] Andrew Ng. Personal communications, 2002. Regarding μ synthesis for autonomous helicopters.

[Ng, 2002b] Andrew Ng. Personal communications, 2002. Regarding policy search for walking control of four legged robot.

[Nilim and el Ghaoui, 2004] A. Nilim and L. el Ghaoui. Robust solutions to markov decision problems with uncertain transition matrices. In *Neural Information Processings Systems 16*, 2004.

[Nocedal and Wright, 1999] J. Nocedal and S. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[Ormoneit and Glynn, 2001] D. Ormoneit and P. Glynn. Kernel-based reinforcement learning in average cost problems: An application to optimal portfolio choice. In *Advances in Neural Information Processing Systems 13*, Cambridge, MA, 2001. MIT Press.

[Peters *et al.*, 2003] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robots,policy gradients and beyond. In *Third IEEE International Conference on Humanoid Robotics*, 2003.

[Pomerleau, 1989] D. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1*, 1989.

[Press *et al.*, 1992] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[Putterman, 1994] M. Putterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[Rabiner, 1989] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–285, 1989.

[Rumelhart *et al.*, 1986] D. Rumelhart, G. Hinton, and R. Williams. *Learning Internal Representations by Error Propagation*, volume 1, chapter 8. MIT Press, 1986.

[Schapire, 1990] R. E. Schapire. The strength of weak learnability. *Machine Learning*, pages 197–227, 1990.

[Schneider, 1996] J. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *Neural Information Processing Systems 9*, 1996.

[Schoelkopf *et al.*, 2000] B. Schoelkopf, R. Herbrich, A. J. Smola, and R. C. Williamson. A generalized representer theorem: Nc-tr-00-081. Technical report, NeuroCOLT Technical Report, 2000.

[Schuurmans and Patrascu, 2002] D. Schuurmans and R. Patrascu. Direct value-approximation for factored mdps, 2002.

[Shelton, 2001] C. R. Shelton. *Importance Sampling for Reinforcement Learning with Multiple Objectives*. PhD thesis, Massachusetts Institute of Technology, 2001.

[Strehl and Littman, 2004] Alexander L. Strehl and Michael L. Littman. Exploration via model-based interval estimation, 2004. Unpublished.

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

[Sutton *et al.*, 1999a] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation, 1999.

[Sutton *et al.*, 1999b] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems 12*, 1999.

[Tatikonda, 2001] S. Tatikonda. *Control under Communication Constrainsts*. PhD thesis, Massachusetts Institute of Technology, June 2001.

[Urmson *et al.*, 2003] Christopher Urmson, Reid Simmons, and Issa Nesnas. A generic framework for robotic navigation. In *IEEE Aerospace Conference 2003*, March 2003.

[van der Schaft, 1999] A. J. van der Schaft. *L_2-gain and Passivity techniques in Non-linear Control*. Springer-Verlag, 1999.

[Vapnik, 1995] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, NY, USA, 1995.

[Williams, 1992] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[Wurman and Wellman, 1996] P. Wurman and M. Wellman. Bounded parameter markov decision processes. 1996.

APPENDIX A

Duality

Convex (also known as Lagrangian or Fenchel) duality is an important concept within machine learning. It plays a central role in maximum entropy models, Bayesian inference techniques, and linear-programming Bellman equation methods. In this thesis, it appears a number of times, including to develop efficient algorithms for solving the stochastic robustness problem, in the development of covariant policy search methods, and in the relationship between functional and standard gradient methods.

Suppose we start with some optimization problem in standard form:

$$\begin{aligned} &\min_x r(x) \\ &\text{subject to:} \\ &f_i(x) \leq 0 \\ &g_j(x) = 0 \end{aligned} \tag{A.1}$$

That is we have a set of *constraints* on feasible solutions to the optimization problem, where each f_i and h_j is some scalar function and i and j run from $0 \ldots n$ and $0 \ldots m$ respectively.

The central idea of Lagrangian duality is to try to take into account the constraints by augmenting the objective function with a linear combination of the constraint functions that act as a kind of "penalty":

$$L(x, \lambda, \nu) = r(x) + \sum_{i=0}^{n} \lambda_i f_i + \sum_{j=0}^{m} \nu_i g_i \tag{A.2}$$

We call the resulting L the Lagrangian for the optimization problem A.2. We call the vectors λ and ν the Lagrange multipliers associated with the constraints or the *dual variables*. The dual variables can be seen as the variables we optimize over in another optimization problem we call the *dual*:

$$s(\lambda, \nu) = \inf_x L(x, \lambda, \nu) \tag{A.3}$$

It is easy to check that $s(\lambda, \nu)$ is a concave function: it is simply the minimum of a set of linear functions. Further, if we restrict $\lambda \geq 0$ the dual lower bounds the primal:

$$s(\lambda, \nu) \leq r(x^*)$$

where $r(x^*)$ is the optimum value of the primal program A.2. Naturally, then we try to solve the optimization problem:

$$\begin{aligned} &\max_{\lambda,\nu} s(\lambda, \nu) \\ &\lambda \geq 0 \end{aligned}$$

as this forms the *tightest* upper bound on the primal objective. Importantly, there are cases when the optimum of the lower bound $s(\lambda^*, \nu^*) = r(x^*)$. We say that *strong duality* holds when this happens. One commonly used criteria to verify strong duality is known as *Slater's Condition*, which says that if the primal is a convex function over convex constraints, and there exists at least one strictly feasible point, then strong duality holds in the dual. There are many good resources for convex duality and optimization. See for instance [Boyd and Vandenberghe, 2004].

APPENDIX B

Alternate Proofs based on Performance Difference Lemma

In Chapter 7, we provided demonstrations of the performance guarantees associated with PSDP. We repeat here the alternate proof due to [Kakade, 2003] (also in [Bagnell *et al.*, 2003a]) as it also provides useful insight. In particular, it leverages a beautiful result known as the *performance difference lemma* of [Langford and Kakade, 2002] which makes it easy to compare two different policies performances.

THEOREM B.0.1 (Performance Bound). *Let $\pi = (\pi_0, \ldots, \pi_{T-1})$ be a non-stationary policy returned by an ε-approximate version of PSDP in which, on each step, the policy π_t found comes within ε of maximizing the value. I.e.,*

$$\mathrm{E}_{\mathrm{s}\sim\mu_\mathrm{t}}[\mathrm{V}_{\pi_\mathrm{t},\pi_{\mathrm{t}+1}\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})] \geq \max_{\pi'\in\Pi}\mathrm{E}_{\mathrm{s}\sim\mu_\mathrm{t}}[\mathrm{V}_{\pi',\pi_{\mathrm{t}+1}\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})] - \varepsilon\,. \tag{B.1}$$

Then for all $\pi_{\mathrm{ref}} \in \Pi^T$ we have that

$$V_\pi(s_0) \geq V_{\pi_{\mathrm{ref}}}(s_0) - T\varepsilon - Td_{\mathrm{var}}(\mu, \mu_{\pi_{\mathrm{ref}}})\,.$$

LEMMA B.0.1 (Performance Difference Lemma). *For our reference policy $\pi_{\mathrm{ref}} = (\pi_{\mathrm{ref},0}, \ldots, \pi_{\mathrm{ref},T-1}) \in \Pi^T$, and a policy we wish to compare it to $\pi = (\pi_0, \ldots, \pi_{T-1})$ the difference in performance can be written as:*

$$V_{\pi_{\mathrm{ref}}}(s) - V_\pi(s) = \textstyle\sum_{t=0}^{T-1} \mathrm{E}_{\mathrm{s_t}\sim\mathrm{P_t}}[\mathrm{V}_{\pi_{\mathrm{ref_t}},\pi_{\mathrm{t}+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s_t}) - \mathrm{V}_{\pi_\mathrm{t},\pi_{\mathrm{t}+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s_t})] \tag{B.2}$$

Intuitively, the performance difference lemma tells us that using a policy π instead π_{ref} will cause the performance to change to by the *advantage* of using π_{ref} at each at each time step, weighed by the probability of under π_{ref}.

Proof (of lemma). For shorthand, let $P_t(s) = \Pr(s_t = s|s_0, \pi_{\rm ref})$. We have then:

$$\begin{aligned} V_{\pi_{\rm ref}}(s) - V_\pi(s) &= \tfrac{1}{T}\sum_{t=0}^{T-1} \mathrm{E}_{\mathrm{s_t}\sim\mathrm{P_t}}[\mathrm{r(s_t)}] - \mathrm{V}_{\pi_0,\ldots}(\mathrm{s}) \\ &= \sum_{t=0}^{T-1} \mathrm{E}_{\mathrm{s_t}\sim\mathrm{P_t}}[\tfrac{1}{\mathrm{T}}\mathrm{r(s_t)} + \mathrm{V}_{\pi_t,\ldots}(\mathrm{s_t}) - \mathrm{V}_{\pi_t,\ldots}(\mathrm{s_t})] - \mathrm{V}_{\pi_0,\ldots}(\mathrm{s}) \\ &= \sum_{t=0}^{T-1} \mathrm{E}_{\mathrm{s_t}\sim\mathrm{P_t},\mathrm{s_{t+1}}\sim\mathrm{P}_{\mathrm{s_t}\pi_{\mathrm{ref},t}(\mathrm{s_t})}}[\tfrac{1}{\mathrm{T}}\mathrm{r(s_t)} + \mathrm{V}_{\pi_{t+1},\ldots}(\mathrm{s_{t+1}}) - \mathrm{V}_{\pi_t,\ldots}(\mathrm{s_t})] \\ &= \sum_{t=0}^{T-1} \mathrm{E}_{\mathrm{s_t}\sim\mathrm{P_t}}[\mathrm{V}_{\pi_{\mathrm{ref},t},\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s_t}) - \mathrm{V}_{\pi_t,\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s_t})] \end{aligned}$$

The lemma is intuitive if seen the following light. Each of the terms:

$$\mathrm{E}_{\mathrm{s_t}\sim\mathrm{P_t}}[\mathrm{V}_{\pi_{\mathrm{ref},t},\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s_t}) - \mathrm{V}_{\pi_t,\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s_t})]$$

tells us how much our performance changes if we to use $\pi_{\rm ref}$ all the way up until time t and then were to switch to π for all future steps. If we sum up the advantages at each time step of using policy $\pi_{\rm ref}$ instead of π, weighed by the probability that we would find ourselves in that state at time t using $\pi_{\rm ref}$, that sum is the total difference in performance.

Proof (of theorem).

It is easy to show that for a function f bounded in absolute value by B, a small change in probability distribution results in an equally small change in expected values of functions with respect to that distribution:

$$|E_{s\sim\mu_1}[f(s)] - E_{s\sim\mu_2}[f(s)]| \leq B\sum_s |\mu_1(s) - \mu_2(s)|$$

Since the values are bounded in the interval $[0, 1]$ and since

$$\begin{aligned} & \sum_{t=0}^{T-1} \mathrm{E}_{\mathrm{s_t}\sim\mathrm{P_t}}[\mathrm{V}_{\pi_{\mathrm{ref_t}},\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s_t}) - \mathrm{V}_{\pi_t,\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s_t})] \\ \leq & \sum_{t=0}^{T-1} \mathrm{E}_{\mathrm{s}\sim\mu_t}[\mathrm{V}_{\pi_{\mathrm{ref_t}},\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s}) - \mathrm{V}_{\pi_t,\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})] - \sum_{\mathrm{t}=0}^{\mathrm{T}-1} |\mathrm{P_t(s)} - \mu_\mathrm{t}(\mathrm{s})| \\ \leq & \sum_{t=0}^{T-1} \max_{\pi'\in\Pi} \mathrm{E}_{\mathrm{s}\sim\mu_t}[\mathrm{V}_{\pi',\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s}) - \mathrm{V}_{\pi_t,\pi_{t+1},\ldots,\pi_{\mathrm{T}-1}}(\mathrm{s})] - \mathrm{Td}_{\rm var}(\mu_{\pi_{\rm ref}}, \mu) \\ \leq & \; T\varepsilon + Td_{\rm var}(\mu_{\pi_{\rm ref}}, \mu) \end{aligned}$$

where we have used equation (B.1) and the fact that $\pi_{\rm ref} \in \Pi^T$. The result now follows.

■

APPENDIX C

RLBENCH Project

Much research in reinforcement learning is hampered by the lack of adequate benchmarks. Many benchmarks problems are published, but they often are missing important details as well as parameters. Other software benchmarks have been published, but this has suffered from the lack of agreed upon and standard interfaces.

An effort to remedy this was discussed during the Neural Information Processing 2003 Workshop on Planning in the Real World run by the author, Nicholas Roy, and Joelle Pineau. Reinforcement learning presents a more difficult benchmarking problem than supervised learning as we cannot simply provide data sets. Instead, we must provide a form of interactive system. Since the workshop, John Langford and the author have developed a set of standard interfaces using Unix pipes that provide language independence and the various access models discussed in Chapter 2.

Problems discussed within the thesis, as well as some that are not, are being modified to fit this new standard and will become available to test new algorithms. More information, including the current code for the benchmark as well as the interface standard are available at

`http://hunch.net/~rlbench`

Document Log:

Manuscript Version 1 — August 30, 2004

Typeset by $\mathcal{AMS}$-LaTeX — 10 January 2005

J. ANDREW BAGNELL

ROBOTICS INSTITUTE, CARNEGIE MELLON UNIVERSITY, 5000 FORBES AVE., PITTSBURGH, PA 15213, USA, *Tel.* : (412) 268-4858

E-mail address: dbagnell@@ri.cmu.edu

Typeset by $\mathcal{AMS}$-LaTeX